CARL VON OSSIETZKY universität OLDENBURG

- ABSCHLUSSBERICHT -

Berghaus, Chermette, Duman, Groenhoff,
Heitmann, Hillmer, Kessler, Otto, Tomović, Yang

VERY LARGE
BUSINESS
APPLICATIONS

Betreuer:

Dmitriyev, Giesen, Marx-Gómez, Wunderlich

Danksagung

Zunächst möchten wir uns an dieser Stelle bei all denjenigen bedanken, die uns während der gesamten Projektdauer unterstützt und begleitet haben.

Großer Dank gilt unseren Universitätsbetreuern - *Prof. Dr. Jorge Marx Gómez*, *Nils Giesen*, *Stefan Wunderlich* & *Viktor Dimitrijev* - die unsere Arbeit und somit auch uns betreut haben. Wir verdanken euch jede erdenktliche Fürsprache, hilfreiche Unterstützung sowie anregende Gespräche. Ihr habt uns in jeder Phase des Projekts sehr sachkundig und richtungsweisend begleitet, uns stets ermuntert und viel Geduld gezeigt.

Darüber hinaus möchten wir uns besonders bei der abat AG und insbesondere unseren Betreuern - *Markus Fischer*, *Dag Oeing*, *Joachim Mannherz* & *Jonas Schlemminger* - sowie dem ganzen Team für die ausgiebige Unterstützung danken. Nicht nur durch die Bereitstellung von technischen Ressourcen, sondern auch durch stetig kritisches Hinterfragen und konstruktive Kritik habt ihr uns zu der Erreichung unserer Ziele verholfen.

Vielen Dank für die Zeit und Mühe, die ihr in unsere Arbeit investiert habt.

gez.

DoHA - „Demosystem on HANA“

Abstract

In der Projektgruppe DoHA (Demosystem on HANA) wurden fünf Szenarien prototypisch bzw. konzeptionell umgesetzt.

Im Szenario *logistische Prozesskette* wurde eine Prozesskette analysiert und vollständig über mehrere Organisationseinheiten hinweg im SAP S/4HANA-System abgebildet. Die Implementierungen der Szenarien *Sensorik* und *Projektcontrolling* liefern Erkenntnisse über heterogene Anwendungsmöglichkeiten der SAP HANA-Datenbank. Hierbei wurde die Echtzeitanalyse verschiedener Sensordaten durch effiziente und integrierte Datenerfassung und -verarbeitung mit Hilfe von Raspberry Pis sowie ein Projekt-Reporting basierend auf innerbetrieblicher Zeiterfassung realisiert.

Um die Erweiterbarkeit und Vielfältigkeit des Szenarios Sensorik aufzuzeigen, wurde dieses um die Komponente *Echtzeit-Stromanalyse* erweitert, indem intelligente Steckdosen Daten in die SAP HANA-Datenbank einspeisen, welche in Echtzeit ausgewertet werden.

Das Szenario *Produktnachhaltigkeit in SAP S/4HANA* prüft in einem Konzept, welche Funktionen im Bereich der Nachhaltigkeit im aktuellen SAP System existieren und welche Funktionen durch Customizing realisierbar sind.

Inhaltsverzeichnis

Abkürzungen

ABAP Advanced Business Application Programming

AJAX Asynchronous JavaScript and XML

CC Creative Commons

CDN Content Delivery Network

CSS Cascading Style Sheets

CSV Comma-seperated values

DOM Document Object Model

GUI Graphical User Interface

HTML Hypertext Markup Language

ID Identifikation

IP Internet Protocol

JSON JavaScript Object Notation

JSON JavaScript Object Notation

OData Open Data Protocol

SAPUI5 SAP User Interface Development Toolkit für HTML5

UI User Inerface

URL Uniform Resource Locator

Viz visualization initialization

XML Extensible Markup Language

Abbildungen

Tabellen

Listings

1. Einleitung

In Kooperation mit dem Bremer SAP-Beratungshaus abat AG beschäftigt sich die Projektgruppe DoHA (Demosystem on HANA) im Rahmen eines einjährigen Masterprojektes mit fünf innovativen Anwendungsszenarien für SAP S/4HANA und SAP HANA.
Im Szenario „Logistische Prozesskette“ wurde in SAP S/4HANA eine vollständige logistische Prozesskette über die Organisationseinheiten Einkauf, Lager, Produktion und Vertrieb hinweg in einem Modellunternehmen abgebildet.
Das Anwendungsszenario „Sensorik“ verarbeitet die Daten verschiedener Sensoren. Das Ziel ist es, die Informationen der Sensoren in eine SAP HANA-Datenbank zu überführen, aufzubereiten und in einer SAP User Interface Development Toolkit für HTML5 (SAPUI5)-Anwendung grafisch darzustellen.
Aufgabe des Szenarios „Projektcontrolling“ war es, Arbeitszeitinformationen von einem Quellsystem in SAP HANA zu importieren und das Auswerten von Projekten über SAPUI5-Oberflächen zu ermöglichen.
Das Szenario „Echtzeit Stromanalyse“ befasst sich mit der Verarbeitung von Stromdaten. Die Daten eines Gerätes werden von einer elektronischen Steckdose erfasst. Ziel ist, die gemessenen Daten an die SAP HANA Datenbank zu schicken und darin zu speichern und zu analysieren.
Das Konzept „Produktnachhaltigkeit in SAP S/4HANA“ fokussiert sich auf SAP S/4HANA und zeigt auf, inwiefern Funktionen zum Management der Produktnachhaltigkeit in diesem System enthalten sind und wie gegebenenfalls Funktionen durch Customizing ergänzt werden können.

1.1. Motivation

Aufgrund der stark ansteigenden Menge an Daten, haben Unternehmen vermehrt die Herausforderung Daten in strukturierter Weise darzustellen und zu analysieren. Der Wunsch nach einer Auswertung in Echtzeit wird zudem größer. SAP HANA greift mit der In-Memory-Technologie diese Problematik auf und schafft neue betriebliche Anwendungsszenarien. Durch eine beschleunigte Zugriffsgeschwindigkeit auf die Daten, wird eine Grundlage für Echtzeitanalyse-Anwendungen gelegt. Das Aufzeigen der durch die Technologie generierten Potenziale, war Ziel der jeweiligen Szenarien, welche in den nachfolgenden Kapiteln vorgestellt werden.

1.2. Problemstellung und Zielsetzung

Die Herausforderung in der Projektgruppe war es, Anwendungsfelder im Umfeld von SAP S/4HANA und SAP HANA aufzuzeigen. Die Szenarien beschäftigen sich mit Fragestellungen für die Anbindung von SAP HANA und SAP S/4HANA im betrieblichen Bereich. Die initialen Szenarien (Logistische Prozesskette, Sensorik und Projektcontrolling) stammen aus Problemstellungen, welche von der abat AG und dessen Kundenkreis stammen. Da diese praxisnahen Anwendungsfelder eine existierende Anwendungslücke behandeln, lassen sich aus den Ergebnissen der Prototypen weitreichende Erkenntnisse gewinnen.
Ziel der Projektgruppe war es weiterhin, eigene Ideen zu entwickeln und diese auf dem SAP HANA-System umzusetzen. In einem Brainstorming wurden Ideen entwickelt, welche in einem Prototyp (Echtzeit Stromanalyse) und als Konzept (Produktnachhaltigkeit in SAP S/4HANA) umgesetzt wurden.

1.3. Aufbau der Dokumentation

Die gesamte Abschlussdokumentation ist in 9 Kapitel gegliedert. Die Dokumentation beginnt mit einem einleitenden Kapitel (Kapitel 1), in dem das Projekt motiviert wird. Des Weiteren findet in diesem Kapitel eine Beschreibung der Problemstellung und der hiermit verbundenen Zielsetzung statt.
Im darauffolgenden Kapitel (Kapitel 2) erfolgt die Beschreibung der Rahmenbedingungen. Hierzu zählt sowohl die Rolle der abat AG, als auch die der Universität Oldenburg.
Im nachfolgenden Kapitel (Kapitel 3) wird auf das Projektmanagement eingegangen. Dies beinhaltet neben der Beschreibung der Projektgruppe, die Dokumentation der im Projekt vertretenen Rollen. Hierzu zählt unter anderem die Rolle des Projektmanagements, des Testbeauftragten, der Serveradministration etc. In einem weiteren Unterpunkt des Kapitels Projektmanagement erfolgt die Erläuterung der Projektphasen. Die Projektphasen sind hierbei in die Phasen „Vorbereitung“, „Implementierung 1“, „Implementierung 2“ und „Abschluss“ gegliedert.
In den folgenden Kapiteln (Kapitel 4-8) wird ausführlich auf die jeweiligen Szenarien eingegangen. Hierbei wird jedes Szenario mit einer Beschreibung eingeleitet, in welcher der entsprechende Anwendungsfall erläutert wird. Im Folgenden werden die erhobenen Anforderungen und die damit zusammenhängende Umsetzung ausführlich dokumentiert. Ebenfalls wurde der zu jedem Szenario durchgeführte Abnahmetest entsprechend aufgeführt. Zu allen Szenarien wird abschließend ein Fazit gezogen und ein Ausblick über eine mögliche zukünftige Weiterentwicklung des Anwendungsfalls bzw. der verschiedenen Prototypen gegeben.
Zum Abschluss der gesamten Dokumentation (Kapitel 9) erfolgt ein übergreifendes Fazit über das gesamte Projekt.

2. Rahmenbedingungen

2.1. Rolle der abat AG

Die abat AG hat während der Projektlaufzeit die Studierenden als Praxispartner betreut. Hierbei standen jedem Szenario feste Ansprechpartner zur Verfügung. Mit diesen Ansprechpartnern wurden regelmäßige Termine vereinbart, um Anforderungen zu definieren und den aktuellen Fortschritt zu besprechen. Dies hatte zur Folge, dass frühzeitig auf Missverständnisse oder unvorteilhafte Vorangehensweisen reagiert werden konnte. Bei Problemen, fachlich oder organisatorisch, konnten sich die Studierenden stets an die Ansprechpartner wenden. Dabei wurden auch häufig innerhalb von kurzer Zeit weitere Ansprechpartner bei speziellen Problemen vermittelt, beispielsweise Experten zu einem bestimmten SAP-Bereich.
Des Weiteren hat die abat AG sämtliche technischen Systeme zur Verfügung gestellt, mit denen die Studierenden während der Projektlaufzeit gearbeitet haben.

2.2. Rolle der Universität

Die Universität betreute die Studierenden vor allem in organisatorischen Angelegenheiten. Hierbei standen den Studierenden über die gesamte Projektlaufzeit hinweg, zwei Ansprechpartner zur Verfügung. Mit diesen wurde wöchentlich ein Statusmeeting abgehalten und darüber hinaus auch weitere Termine für Besprechungen vereinbart.
Des Weiteren wurden die Leistungen der einzelnen Projektmitglieder von den Betreuern der Universität in regelmäßigen Feedbackgesprächen diskutiert und benotet. Aus diesen Teilleistungen setzte sich die Gesamtnote zusammen.

3. Projektmanagement

3.1. Projektgruppe

Abbildung 3.1.: V. l. n. r.: Milan Tomović, Gerrit Berghaus, Jingyu Yang, Julia Chermette, Nils Groenhoff, Henrik Hillmer, Felix Otto, Sezer Duman, Henrik Heitmann, René Kessler

3.2. Rollen innerhalb der Projektgruppe

Tabelle 3.1.: Übersicht: Rollen der Projektmitglieder

Rolle	**Besetzung**
Projektmanagement (1. Phase)	Gerrit Berghaus und René Kessler
Projektmanagement (2. Phase)	Henrik Hillmer und Milan Tomović

Tabelle 3.1.: Übersicht: Rollen der Projektmitglieder

Rolle	**Besetzung**
Projektmanagement (3. Phase)	Nils Groenhoff und Jingyu Yang
Testbeauftragter	Julia Chermette und Henrik Heitmann
Dokumentationsbeauftragter	Sezer Duman
Serveradministrator	René Kessler und Felix Otto
Webseitenbeauftragter	Julia Chermette
CeBIT-Beauftragter	Sezer Duman
Social-Event-Beauftragter	Milan Tomović
Kassenwart	Milan Tomović

3.2.1. Projektmanagement

Das Projektmanagement war die umfangreichste Rolle der Projektgruppe und wurde daher von insgesamt sechs Studierenden ausgeübt (siehe Tabelle 3.1). Die gesamte Projektphase wurde in drei Projektmanagement-Teilphasen aufgeteilt. Für jede Phase nahmen jeweils zwei Studierende die Rolle des Projektmanagements ein.
Das Projektmanagement übernahm in jeder Phase wesentliche Kernaufgaben, welche für die Koordination und Arbeitsgestaltung der Projektgruppe eine elementare Rolle spielten. Hierzu gehörten unter anderem die kontinuierliche Kommunikation mit dem Praxispartner, das Vorbereiten interner und externer Meetings sowie das Verteilen und Strukturieren szenarienübergreifender Arbeitspakete. Das Projektmanagement war zudem erster Ansprechpartner der Betreuer und erste Instanz der Konflikt- und Problemlösung projektinterner Schwierigkeiten und Diskussionen.

3.2.2. Testbeauftragter

Die Aufgabe der zwei Testbeauftragten war es, ein Vorgehen für das Durchführen von Tests zu entwickeln sowie eine Vorlage zu erstellen, um die Ergebnisse festzuhalten. Ursprünglich war es geplant Testszenarien zu erstellen, welche von Mitarbeitern von abat durchgeführt werden sollten. Sollten bei Schritten des Szenarios Fehler auffallen, würden diese auf einem Testbogen vermerkt werden, um sie anschließend zu berichtigen. Dieser Zyklus sollte dann mehrfach wiederholt werden, um eine möglichst fehlerfreie Anwendung zu erstellen.
Es wurde sich jedoch im Projektverlauf dagegen entschieden die Tests wie oben erläutert durchzuführen. Der Grund hierfür war die mangelnde Zeit, um den Vorgang mehrfach

wiederholen zu können. Es wurden deshalb Abnahmedokumente erstellt, auf denen die einzelnen Anforderungen abgebildet sind. Der fertige Prototyp wurde dann gegen das Dokument geprüft und es wurde vermerkt, ob die jeweilige Anforderung realisiert wurde oder nicht. Die Dokumente sind in den jeweiligen Szenarien-Dokumentationen zu finden.

3.2.3. Dokumentationsbeauftragter

Der Dokumentationsbeauftragte war verantwortlich für das Erstellen von einheitlichen Vorlagen (bspw. LaTeX- oder Präsentationsvorlagen) und die Koordination sowie Verwaltung aller entstandenen Dokumente wie bspw. Protokolle, Präsentationen oder Entwürfe die während der einjährigen Projektzeit entstanden sind. Er überwachte und kommunizierte die Einhaltung von festgelegten Layout-Vorgaben und betreute maßgeblich dokumentarische Aufgaben. Die Hauptaufgabe war dabei die Betreuung und Koordination der Abschlussdokumentation während der Abschlussphase des Projekts.

3.2.4. Serveradministrator

Das Amt des Serveradministators wurde von zwei Personen übernommen, die sich die Aufgaben gemeinsam aufteilten. Wichtige Aufgaben waren die allgemeine Pflege und Wartung des Servers, das Anlegen von Nutzerkonten, der initiale Aufbau der Website und die Installation von verschiedenen Tools zur Versionsverwaltung von Dateien sowie einer Projektmanagementsoftware. Unter anderem wurde zur Verwaltung der Dateien ein GIT aufgesetzt. Die meisten Aufgaben wurden in enger Absprache mit dem Serveradministrator der VLBA erledigt.

3.2.5. Webseitenbeauftragter

Die Projektgruppe hatte eine Webseite, welche auch einen Blog beinhaltete (vgl. Abbildung 3.2). Zur Aufgabe des Webseitenbeauftragten gehörte die Erstellung und Pflege der Wordpress Seite (www.demosystem-on-hana.de). Unter anderem wurden Texte für den Blog verfasst, welcher über aktuelle Events und Statusnachrichten der Projektgruppe informierte. Die Info-E-Mail Adresse pg-doha-info@informatik.uni-oldenburg.de wurde für die Webseite eingerichtet. Außerdem fiel das Design des Logos unter die Aufgaben des Webseitenbeauftragten.

3.2.6. CeBIT-Beauftragter

Die Rolle des CeBIT-Beauftragten ist während der Abschlussphase des Projektes entstanden, da der verwaltungstechnische Aufwand der Exponats-Planung dies erfordert hat. Der CeBIT-Beauftragte koordinierte, betreute und verwaltete, in Abstimmung mit der Projektgruppe und den Betreuern der Universität, die verwaltungstechnischen Aufgaben des CeBIT-Standes. Zudem verpflichtete er sich am Ausstellertreffen des Niedersachen-Gemeinschaftsstandes in Hannover teilzunehmen und regelmäßigen Kontakt mit der verantwortlichen Messeagentur zu pflegen.

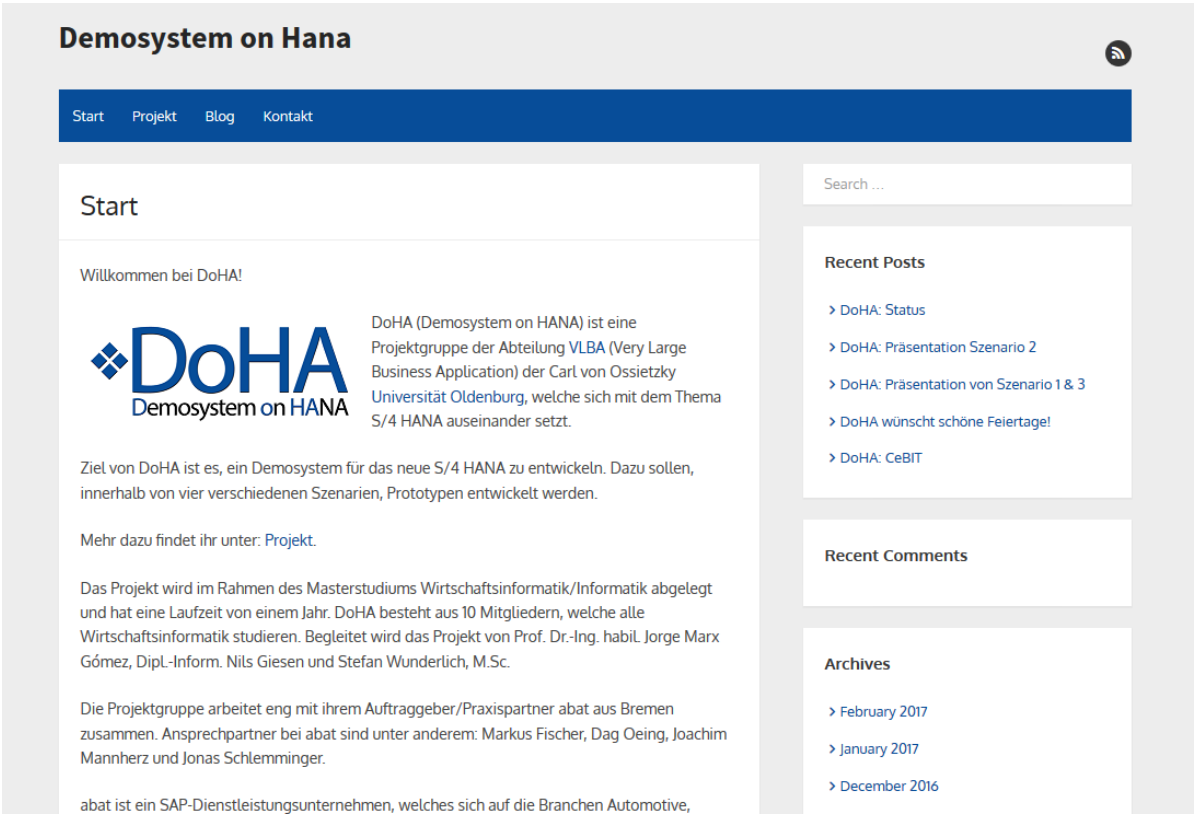


Abbildung 3.2.: Projektmanagement: DoHA Website

3.2.7. Social-Event-Beauftragter

Der Social-Event-Beauftragte hat Veranstaltungen und Aktivitäten organisiert, welche zu Zwecke des Teambuildings durchgeführt wurden. Beispiele hierfür waren Abende in Oldenburger Lokalitäten mit maßvollem Alkoholverzehr, Grille, eine Weihnachtsfeier, jedoch auch sportliche Events wie ein Go-Kart-Rennen, Fußballspiele gegen andere Projektgruppen oder auch ein Paintball-Match. Auf der Webseite der Projektgruppe wurden die meisten Events im Blog erwähnt. Die Termine für diese Veranstaltungen wurden per Doodle-Umfrage abgestimmt. Die gesamte Projektgruppe, oft sogar die Betreuer der Universität Oldenburg und Mitarbeiter der abat AG haben sich aktiv an Social-Events beteiligt. Da der Social-Event-Beauftragte auch der Kassenwart in einer Person war, wurden Finanzielle Belange, die in Verbindung mit diesen Events standen, von ihm koordiniert.

3.2.8. Kassenwart

Der Kassenwart war für Finanzielle Belange der Projektgruppe zuständig. Für folgendes Fehlverhalten wurde ein spezieller Strafenkatalog aufgestellt, der auch vom Kassenwart im Freedcamp gepflegt wurde.

- Eine E-Mail ohne [DoHA]-Tag versenden. Strafe: 0,50 Euro.

- Verwechslung vom Prototyp und Szenario. Strafe: 0,50 Euro.
- Nicht entschuldigtes verspätetes Erscheinen bei einer internen oder externen Sitzung. Strafe: 2,00 Euro.
- Nicht entschuldigtes fehlen bei einer internen oder externen Sitzung. Strafe: 10,00 Euro.

3.3. Projektphasen

3.3.1. Vorbereitung

Vor dem ersten Treffen mussten alle Studenten der Projektgruppe, sich mit Hilfe der SAP HANA Academy Videos über das SAP HANA System informieren. Dies diente dazu alle Teilnehmer der Projektgruppe auf den gleichen Wissensstand, in Bezug auf SAP HANA, zu bringen. Das erste Kickoff der Projektgruppe fand dann in Oldenburg statt, wo der Ablauf und die Rollen der Projektgruppe kurz vorgestellt wurden. Ein weiteres Kickoff fand beim Projektpartner, der abat AG, in Bremen statt. Dabei wurde die Aufteilung der Projektgruppe in Teilszenarien, der Inhalt dieser Teilszenarien, die Seminarthemen und die Ansprechpartner bei abat vorgestellt.

3.3.2. Szenarien

Das Gesamtprojekt dieser Projektgruppe wurde von Seiten des Projektpartners abat in Szenarien aufgeteilt. In diesen Szenarien wurden mehr oder weniger unabhängige Applikationen entwickelt. Dabei behandelten drei dieser Szenarien feste, von der abat AG vorgegebene, Themen. Diese drei Szenarien hießen *Logistische Prozesskette*, *Projektcontrolling* und *Sensorik*. Des weiteren war eine weitere Vorgabe, ein oder mehrere weitere Szenarien zu entwickeln. Hierbei sollte die Projektgruppe eigene Ideen entwickeln, die zum einen die Potentiale des SAP HANA Systemes hätte nutzen können und zum anderen für abat und Ihre Kunden von Interesse gewesen wären. Nach Beginn des Projektes haben sich zunächst alle Studenten der Projektgruppe auf die ersten drei Szenarien aufgeteilt. Diese Aufteilung hat sich im Laufe des Projektes noch etwas verändert. Vor allem zum Ende der ersten Entwicklungsphase haben sich drei der Studierenden zu einer neuen Teilgruppe zusammengefunden, welche das weitere Szenario mit der eigenen Idee der Projektgruppe behandelt haben. Das Thema dieses Szenarios war dabei die *Echtzeit Stromanalyse*. Außerdem haben die Mitglieder des Teilszenarios *Logistische Prozesskette* nach dem Abschluss ihres Szenarios, ein Konzept zu einer weiteren Idee erstellt. Der Titel dieses Konzeptes war dabei *Produktnachhaltigkeit in SAP S/4HANA*. Die Entwicklung dieser Ideen fand zum einen mit Hilfe von Brainstormings in den internen Sitzungen der Studenten statt. Zum anderen wurden die Ergebnisse dieser Brainstormings mit den Betreuern und den Ansprechpartnern der abat AG besprochen. Diese Vorgänge fanden zudem in mehreren Iterationen statt.

Logistische Prozesskette

In dem Szenario *logistische Prozesskette* war die Aufgabe die Prozesskette eines produzierenden Unternehmens und deren Geschäftsprozesse in SAP S/4HANA abzubilden. Dazu wurden die Bereiche Einkauf, Lager, Produktion und Vertrieb im System betrachtet. Es wurde untersucht, in wie weit die RDS-Inhalte für das Abbilden der logistischen Prozesskette genutzt werden können und wo diese auf ihre Grenzen stoßen. Die Prozesse, welche nicht im RDS enthalten waren, wurden von der Teilgruppe im SAP S/4HANA System implementiert, soweit dies möglich war. Somit wurde in diesem Szenario kein Code geschrieben.

Sensorik

In dem Szenario *Sensorik* wurde eine Anwendung entwickelt, welche Sensordaten von Raspberry Pis und Druckern analysiert und in einer mobilen SAP HANA XS-/SAP FIORI-Oberfläche darstellt. Dazu wurde unter anderem Datenbankmodelle innerhalb von SAP HANA entwickelt und die Daten von den Sensoren in die Datenbank importiert und verarbeitet.

Projektcontrolling

Ähnlich wie in Szenario *Sensorik* wurde auch im Szenario *Projektcontrolling* ein Datenbankmodell innerhalb von SAP HANA entwickelt sowie Daten importiert, verarbeitet und in einer mobilen SAP HANA XS-/SAP FIORI-Oberfläche dargestellt. Jedoch wurden hier statt Sensordaten Daten aus dem Projektcontrolling der abat AG importiert und visualisiert. Dadurch unterschied sich die Benutzeroberfläche und die Schnittstelle zur Datenquelle deutlich vom Szenario *Sensorik*.

Echtzeit Stromanalyse

Bei diesem Szenario sollte die Projektgruppe eine eigene Idee entwickeln und umsetzen, welche für die abat AG und ihre Kunden interessant gewesen wäre und die Potentiale des SAP HANA Systems hätte nutzen können. Nach einigen ausgearbeiteten und mit abat besprochenen Ideen wurde sich für das Thema *Echtzeit Stromanalyse* entschieden. Hierbei sollten, ähnlich wie im Szenario *Sensorik*, Sensordaten importiert, analysiert und auf einer mobilen SAP HANA XS-App/SAPUI5-Oberfläche dargestellt werden. Bei diesen Sensordaten handelte es sich jedoch um Stromdaten ,welche später mit Hilfe von intelligenten Steckdosen gemessen wurden.

Produktnachhaltigkeit in SAP S/4HANA (Konzept)

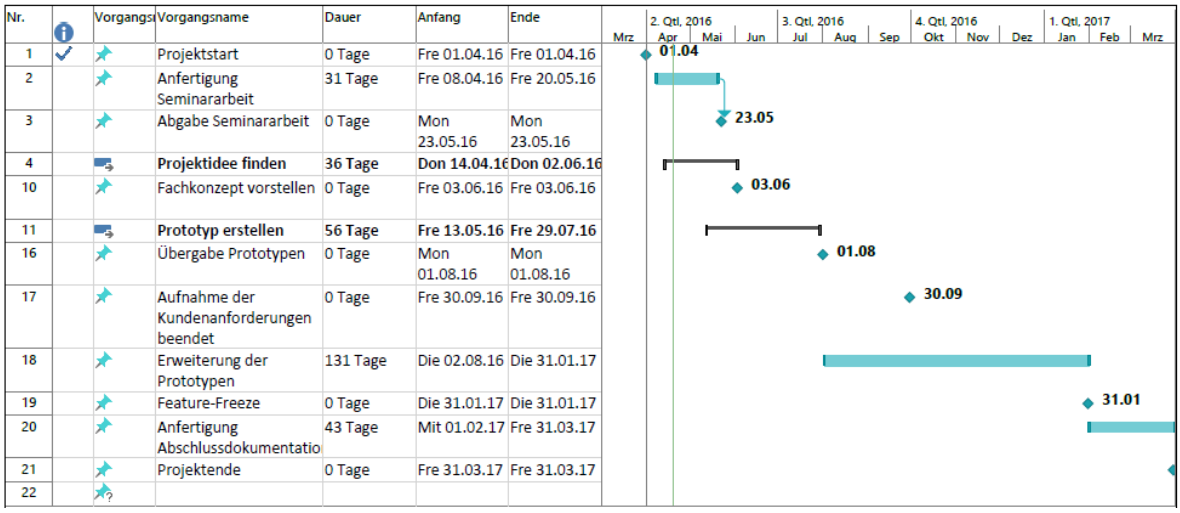
Der Ideenfindung des Szenarios *Echtzeit Stromanalyse* ist außerdem dieses Szenario mit dem Thema *Produktnachhaltigkeit in SAP S/4HANA* entsprungen. Hierbei wurde jedoch nur ein Konzept erstellt, da die Ressourcen und die verbleibende Zeit zu gering waren um einen Prototypen zu diesem Thema zu realisieren. In diesem Konzept setzte das Szenario

sich mit der Frage auseinander, welche Nachhaltigkeitsdaten im SAP S/4HANA-System enthalten sind und wie diese analysiert und genutzt werden könnten.

3.3.3. Implementierung Phase 1

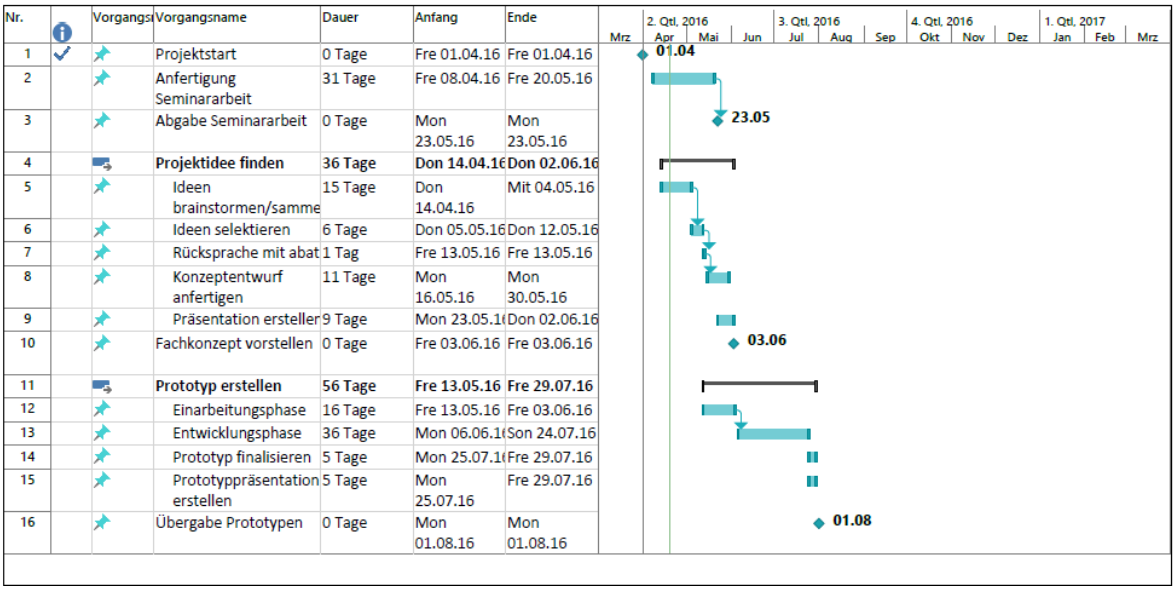
Nach den Kickoffs begann schließlich das eigentliche Projekt. Für den regelmäßigen Austausch innerhalb der Projektgruppe und mit dem Projektpartner abat AG, fanden innerhalb des Projektes jede Woche Treffen statt. Dabei fand zunächst nur ein wöchentliches Treffen mit den Betreuern statt. Alle drei Wochen fand dieses Treffen dann bei und mit dem Projektpartner abat statt. Bei diesen Treffen wurden aktuelle Ergebnisse der Projektgruppe besprochen und anfangs Aufgaben verteilt. Nach ein paar Wochen wurde zudem ein zusätzliches Treffen jede Woche eingeführt, bei welchem sich nur die Studierenden trafen. Bei diesem Treffen wurden aktuellen Ergebnisse sowie Aufgaben besprochen und dadurch die Treffen mit den Betreuern und dem Projektpartner abat vor- bzw. aufbereitet. Im ersten regulärem Treffen der Projektgruppe wurden vor allem die Seminarthemen vergeben, zu welchen die Studierenden innerhalb eines Monats eine Seminararbeit verfasst haben. Diese Seminarbeiten dienten dazu Informationen über das Projektmanagement, die Entwicklungsumgebung und die Teilszenarien für alle Mitglieder der Projektgruppe aufzubereiten. In der Anfangsphase der Projektgruppe wurden zudem der Name der Projektgruppe und ein dazugehöriges Logo festgelegt. Der Projektgruppe wurde dabei der Name Demosystem on HANA, kurz DoHA, gegeben. Außerdem wurden die Rollen verteilt, die Studenten wurden in die ersten drei festen Teilszenarien aufgeteilt, das Projektmanagement Tool festgelegt und ein erster Meilensteinplan vom ersten Projektmanagement erstellt (siehe Abbildungen 3.3 und 3.4 auf der nächsten Seite). Zudem fand im ersten internen Treffen der Studenten ein Brainstorming zu der Projektidee von dem Teilszenario 4 statt. Ab dem 17.06.16 war dann die Anforderungsanalyse für die ersten drei festen Szenarien beendet und die Entwicklung begann. Für das offene 4. Szenario wurden weiter Brainstormings und Absprachen mit abat durchgeführt. Zu dieser Zeit wurden zudem erste Vorlagen für die Tests und die Dokumentation erstellt sowie eine Website zu der Projektgruppe eingerichtet.
Gegen Ende der ersten Entwicklungsphase traten jedoch merkliche zeitliche Probleme auf. Der interne Meilenstein, Entwicklungsstop am 25.07.2016, wurde nicht erreicht. Die Teilgruppen haben auch nach dem 25.07.2016 weiterhin entwickeln müssen, um die Abgabe bis zum 01.08.2016 zu gewährleisten. Dennoch wurden die Teilgruppen mit der Entwicklung bis zum Abgabetermin fertig. Auch die Dokumentation dieses ersten Prototypen bzw. der ersten Entwicklungsphase konnte dennoch fertiggestellt werden. Am 19.08.2016 fand dann die Präsentation vom ersten Prototypen statt.
Insgesamt lief die die erste Entwicklungsphase recht gut ab. Trotzdem gab es einige Probleme und Herausforderungen. So war es z.B. schwierig die Aufwände richtig zu schätzen und die Meilensteine richtig zu setzen, was vor allem an der noch fehlenden Erfahrung mit dem System und größeren Projekten lag. Zudem war die Aufgabenverteilung und Kommunikation anfangs noch etwas schwieriger, was unter anderem daran lag, dass die Projektgruppe sich noch nicht so gut kannte und die Studenten eine unterschiedliche Arbeitsmoral aufwiesen. Zudem gab es einige technische Schwierigkeiten mit dem Server

der Projektgruppe und dem Projektmanagement Tool. Es konnten jedoch einige Erfahrungen gesammelt werden, welche in der zweiten Entwicklungsphase genutzt werden konnten.



Nr.	Vorgangsname	Dauer	Anfang	Ende
1	Projektstart	0 Tage	Fre 01.04.16	Fre 01.04.16
2	Anfertigung Seminararbeit	31 Tage	Fre 08.04.16	Fre 20.05.16
3	Abgabe Seminararbeit	0 Tage	Mon 23.05.16	Mon 23.05.16
4	**Projektidee finden**	**36 Tage**	**Don 14.04.16**	**Don 02.06.16**
10	Fachkonzept vorstellen	0 Tage	Fre 03.06.16	Fre 03.06.16
11	**Prototyp erstellen**	**56 Tage**	**Fre 13.05.16**	**Fre 29.07.16**
16	Übergabe Prototypen	0 Tage	Mon 01.08.16	Mon 01.08.16
17	Aufnahme der Kundenanforderungen beendet	0 Tage	Fre 30.09.16	Fre 30.09.16
18	Erweiterung der Prototypen	131 Tage	Die 02.08.16	Die 31.01.17
19	Feature-Freeze	0 Tage	Die 31.01.17	Die 31.01.17
20	Anfertigung Abschlussdokumentatio	43 Tage	Mit 01.02.17	Fre 31.03.17
21	Projektende	0 Tage	Fre 31.03.17	Fre 31.03.17
22				

Abbildung 3.3.: Meilensteinplan Phase 1 - Grobplanung



Nr.	Vorgangsname	Dauer	Anfang	Ende
1	Projektstart	0 Tage	Fre 01.04.16	Fre 01.04.16
2	Anfertigung Seminararbeit	31 Tage	Fre 08.04.16	Fre 20.05.16
3	Abgabe Seminararbeit	0 Tage	Mon 23.05.16	Mon 23.05.16
4	**Projektidee finden**	**36 Tage**	**Don 14.04.16**	**Don 02.06.16**
5	Ideen brainstormen/samme	15 Tage	Don 14.04.16	Mit 04.05.16
6	Ideen selektieren	6 Tage	Don 05.05.16	Don 12.05.16
7	Rücksprache mit abat	1 Tag	Fre 13.05.16	Fre 13.05.16
8	Konzeptentwurf anfertigen	11 Tage	Mon 16.05.16	Mon 30.05.16
9	Präsentation ersteller	9 Tage	Mon 23.05.16	Don 02.06.16
10	Fachkonzept vorstellen	0 Tage	Fre 03.06.16	Fre 03.06.16
11	**Prototyp erstellen**	**56 Tage**	**Fre 13.05.16**	**Fre 29.07.16**
12	Einarbeitungsphase	16 Tage	Fre 13.05.16	Fre 03.06.16
13	Entwicklungsphase	36 Tage	Mon 06.06.16	Son 24.07.16
14	Prototyp finalisieren	5 Tage	Mon 25.07.16	Fre 29.07.16
15	Prototyppräsentation erstellen	5 Tage	Mon 25.07.16	Fre 29.07.16
16	Übergabe Prototypen	0 Tage	Mon 01.08.16	Mon 01.08.16

Abbildung 3.4.: Meilensteinplan Phase 1 - Detailplanung

3.3.4. Implementierung Phase 2

Mit dem Beginn der zweiten Entwicklungsphase wechselte zum einen die Besetzung des Projektmanagements und zum anderen das Projektmanagement Tool. Dabei wechselte

die Projektgruppe von Redmine zu Freedcamp als Projektmanagement Tool, da es einige technische Probleme mit Redmine gab. Vom neuen Projektmanagement wurde dann auch ein neuer Meilensteinplan für den Rest des Projektes erstellt (siehe Abbildung 3.5 auf der nächsten Seite). Da jedoch die einzelnen Szenarien unterschiedlich hohen, verbleibenden Entwicklungsaufwand hatten, wurden sowohl Meilensteinpläne von und für die einzelnen Teilzenarien erstellt, als auch der allgemeine Meilensteinplan überarbeitet. Zudem wurden in den einzelnen Szenarien, zusammen mit den Anpsrechpartnern von abat, die Ergebnisse der ersten Entwicklungsphase besprochen und neue Anforderungen festgelegt bzw. bisherige überarbeitet. Diese Ergebnisse wurden dann, wie in der ersten Phase, in entsprechenden Entwürfen festgehalten. Ein wichtiger Punkt in der zweiten Entwicklungsphase war der Wechsel eines der zwei universitären Betreuer der Projektgruppe. So übernahm Viktor Dmitriyev den Platz von Nils Giesen als Betreuer. Nils Giesen blieb der Projektgruppe jedoch als Ansprechpartner bei der abat AG erhalten. Eine weitere Umstrukturierung innerhalb der Projektgruppe war die Aufteilung der Teilszenarien. So trennten sich die bisherigen Szenarien von jeweils einem Mitglied. Diese drei Studierenden bildeten dann das Team für die Entwicklung des 4. Szenarios. Somit begannen dann die ersten drei Szenarien mit der Weiterentwicklung ihres ersten Prototypen. Szenario 4 sammelte hingegen zunächst noch einige Informationen und Anforderungen.
Da die Szenarien unterschiedlich viel Aufwand beinhalteten, wurden diese auch zu unterschiedlichen Terminen mit der Entwicklung fertig und hatten an unterschiedlichen Terminen ihre Übergabepräsentationen bei abat. Die Szenarien *Prozesskette* und *Sensorik* hielten diese Übergabepräsentation am 13.01.17. An diesem Tag fand zudem offiziell der zweite Wechsel des Projektmanagements statt, welches noch einmal die letzten Meilensteine überarbeitet hatte und diese vorstellte. Die Übergabepräsentation von Szenario *Projektcontrolling* fand am 02.03.17 statt. Und zuletzt folgten die Szenarien *Echtzeit Stromanalyse* und *Produktnachhaltigkeit in SAP S/4HANA* am 10.03.17. Das Szenario *Echtzeit Stromanalyse* konnte somit auch die Entwicklungsphase abschließen, wodurch sich dann alle Studenten der Projektgruppe auf die Enddokumentation konzentrieren konnten.
Generell waren die Teilszenarien in der zweiten Entwicklungsphase eingespielter und die Projektgruppe hat insgesamt besser zusammengearbeitet. Dennoch gab es einige Problem. So war es trotz gesammelter Erfahrung schwierig die Aufwände richtig abzuschätzen und die Termine langfristig genau vorauszusagen. Dies lag unter anderem an dem unterschiedlichen Stand der Teilszenarien und den teilweise unterschiedlichen Erwartungen der abat AG, der Betreuer und der Projektgruppe. Die Koordination dieser Punkte erwies sich besonders in der zweiten Entwicklungsphase als schwierig. Bezüglich der Software, welche die Projektgruppe nutzte, gab es in der zweiten Phase noch einige technische Probleme. Dennoch war das Arbeiten in der zweiten Phase routinierter. So wurden auch die Meetings effizienter und es konnte mehr in den internen Meetings bzw. allgemein innerhalb der Projektgruppe und der Teilszenarien geklärt werden, wodurch die externen Meetings schlanker wurden. Auch fanden in der zweiten Projektphase wieder Teambuildings statt und diesmal auch mehr mit den Betreuern und den Ansprechpartnern von abat. Dies trug auch zu einem besseren Zusammenhalt im Team bei.

Meilensteinplan

Vorgang	Anfang	Ende	Dauer (Arbeitstage)
Aufnahme der Kundenanforderungen beendet	23.09.2016	23.09.2016	1
Workload und Prioritäten der Szenarien festlegen	23.09.2016	30.09.2016	6
Erweiterter Entwurf erstellen	26.09.2016	21.10.2016	20
Erweiterten Entwurf abgeben	21.10.2016	21.10.2016	1
Erstellung Prototyp 2	24.10.2016	27.01.2017	59
Erste Testphase	05.12.2016	16.12.2016	10
Zweite Testphase	20.01.2017	27.01.2017	6
Feature Freeze	20.01.2017	20.01.2017	1
Fehlerbehbung	30.01.2017	24.02.2017	20
Doku finalisieren	30.01.2017	03.03.2017	25
Endabgabe	10.03.2017	10.03.2017	1
Endpräsentation erstellen	13.03.2017	24.03.2017	10
Endpräsentation vorstellen	31.03.2017	31.03.2017	1

Abbildung 3.5.: Meilensteinplan Phase 2

3.3.5. Abschluss

Nachdem die Entwicklung in der Projektgruppe beendet wurde, verblieben noch drei große Aufgaben, auf welche sich die Projektgruppe konzentrierte. Diese Aufgaben waren die Fertigstellung der Abgabedokumentation, die Vorbereitung auf die CeBIT sowie Besuch dieser, die Vorbereitung der Abschlusspräsentation und die Durchführung dieser. Ein wichtiger Punkt war die Teilnahme an der CeBIT. Die Projektgruppe erhielt einen kleinen Stand an der CeBIT, am Niedersachsenstand. Die Präsentation der Projektgruppe wurde dann im Laufe der zweiten Entwicklungs- und der Abschlussphase immer wieder besprochen. Dabei wurde festgelegt, dass vor allem das Szenario *Sensorik* mit einem Raspberry Pi vorgestellt werden sollte. Außerdem wurde festgelegt, dass die Studenten in Zweierteams den Stand auf der CeBIT betreuen, als auch auf- und abbauen. Zudem erhielt die Projektgruppe sowohl von abat, als auch von der Universität Oldenburg Unterstützung in Form von Materialien für die CeBIT, wie z.B. Tische und Monitore.
Bezüglich der Abschlussdokumentation haben die Mitglieder der jeweiligen Teilszenarien, ihr Szenario auch dokumentiert. Die allgemeinen Teile der Dokumentation wurden dann nach Ende der zweiten Entwicklungsphase unter den Studenten aufgeteilt. Insgesamt mussten viele Inhalte, die während der ersten und am Anfang der zweiten Entwicklungsphase dokumentiert wurden, noch einmal für die Abschlussdokumentation überarbeitet oder neu geschrieben werden. Dadurch sollte sichergestellt werden, dass alle Teile der Enddokumentation einheitlich sind.

4. Szenario - Logistische Prozesskette

4.1. Beschreibung

Diese Dokumentation beschreibt die Umsetzung der Anforderungen, die für das Szenario „Logistische Prozesskette“ der Projektgruppe DoHA erhoben wurden.
Im ersten Abschnitt dieses Dokuments werden die erhobenen und umgesetzten Anforderungen tabellarisch aufgelistet.
Im Hauptteil werden die Aktivitäten beschrieben und dokumentiert, die nötig waren um die Anforderungen umzusetzen. Hierzu gehört die Definition der Organisationseinheiten und deren Zuordnungen, die Definition von Stammdaten sowie die nötigen Einstellungen im Customizing, um die Prozesse den Anforderungen entsprechend abzubilden.
Abschließend werden im letzten Kapitel Probleme und deren Lösungen beschrieben, die im Laufe der Arbeiten am ersten Prototypen auftraten.
In der zweiten Phase stand vor allem die Analyse der Nutzbarkeit der RDS-Inhalte im Mittelpunkt, wozu ein eigenes Dokument angefertigt wurde.
In dem gewählten Szenario soll eine vollständige, logistische Prozesskette über die Organisationseinheiten Einkauf, Lager, Produktion und Vertrieb hinweg abgebildet werden. Hierzu wurde als Modellunternehmen ein mittelständischer Automobilzulieferer gewählt. Gefertigt werden von diesem Automobilzulieferer Bremsanlagen. Jede Bremsanlage besteht im wesentlichen aus einer Bremsscheibe und einem Bremsblock. Der Bremsblock setzt sich aus einem Bremssattel und zwei Bremsbelägen zusammen.
Die Bremsscheiben, -sättel und -beläge werden von einem Zulieferer eingekauft und in einem Produktionsprozess in zwei Schritten zusammengesetzt. Es werden lediglich Bremsanlagen gefertigt und die Bremsblöcke werden erst in eigentlichen Produktionsprozess zusammengefügt. Eine vorgelagerte Produktion dieser Bremsblöcke findet nicht statt.

4.2. Anforderungen

Die aufgenommen Anforderungen wurden in fünf Kategorien eingeteilt. Jede Kategorie stellt hierbei eine Organisationseinheit dar. Dies wurde aus Gründen der Übersicht vorgenommen und um Arbeitspakete besser planen zu können. Dennoch bestehen zum Teil große Abhängigkeiten zwischen einzelnen Anforderungen. Bei den jeweiligen Anforderungen handelt es sich in diesem Szenario um (Teil-)Prozesse, die nötig sind um die logistische Prozesskette korrekt und umfassend abzubilden.

4.2.1. Einkauf

Die in Tabelle 4.1 dargestellten Anforderungen wurden für den Bereich des Einkauf definiert.

Tabelle 4.1.: Logistische Prozesskette: Anforderungen - Einkauf

Nr.	Anforderung:	Beschreibung:
1	**Einkauf**	
1.1	Bestellung manuell anlegen	Bestellung für Rohstoffe/Materialien anlegen
1.2	BAnf manuell anlegen	Bestellanforderungen für Rohstoffe/Materialien anlegen
1.3	BAnf in Bestellung umwandeln	Bestellanforderungen zu Bestellungen umwandeln, Bezugsquellenfindung
1.4	Bestellung genehmigen	Bestellungen (mit bestimmten finanziellem Volumen) genehmigen
1.5	Wareneingang (zu Bestellung) buchen	Wareneingang mit Bezug zu Bestellung buchen
1.6	Lieferantenrechnung (zu Bestellung) erfassen	Rechnungen für gelieferte Bestellungen anlegen
1.7	Lieferpläne	Beschaffung von Rohstoffen bzw. Materialien über Lieferpläne
1.8	Geplanter Wareneingang	Waren sind ab dem geplanten WE disporelevant

4.2.2. Produktion

Die in Tabelle 4.2 auf der nächsten Seite dargestellten Anforderungen wurden für den Bereich der Produktion definiert.

Tabelle 4.2.: Logistische Prozesskette: Anforderungen - Produktion

Nr.	Anforderung:	Umgesetzt:
2	**Produktion**	
2.1	MRP-Lauf durchführen	Materialbedarfsplanung
2.2	Materialdeckung auswerten	Material mit Unterdeckung ermitteln
2.3	Fertigungsauftrag anlegen	Auftrag zur Produktion von Bremsanlagen
2.4	Fertigungsauftrag rückmelden	Stand von Fertigungsaufträgen erledigt/-zum Teil erledigt
2.5	Produktionskostenkalkulierung	Die Produktionskosten sollen unter Berücksichtigung der Personal-, Maschinen-, Materialkosten ermittelt werden
2.6	Kundeneinzelfertigung	Umsetzung von individuellen Kundenaufträgen

4.2.3. Vertrieb

Die in Tabelle 4.3 auf der nächsten Seite dargestellten Anforderungen wurden für den Bereich des Vertriebs definiert.

Tabelle 4.3.: Logistische Prozesskette: Anforderungen - Vertrieb

Nr.	Anforderung:	Umgesetzt:
3	**Vertrieb**	
3.1	Kundenauftrag anlegen	Kundenauftrag zum Verkauf von FE anlegen
3.2	Auslieferung anlegen	Auslieferung mit Bezug zu Kundenauftrag anlegen
3.3	Kommissionierung durchführen	Waren für Auslieferung kommissionieren
3.4	Ware verpacken	Waren einer Auslieferung verpacken
3.5	Avis durchführen	Nach dem WA Avis versenden
3.6	Faktura anlegen	Kundenaufträge fakturieren

Tabelle 4.3.: Logistische Prozesskette: Anforderungen - Vertrieb

Nr.	Anforderung:	Umgesetzt:
3	**Vertrieb**	
3.7	Konditionsliste anlegen	Je nach Kunden sollen verschiedene Konditionen im System hinterlegt werden.
3.8	Kundenbonus	Neben Staffelrabatten sollen Kunden auch Boni gewährt werden, bspw. bei sehr vielen Bestellungen in einem Jahr
3.9	Gutschriftverfahren	Es soll möglich sein die Rechnungsprüfung an den Kunden auszulagern
3.10	Lieferpläne (SD)	Verkauf von Produkten über Lieferpläne

4.2.4. Lager

Die in Tabelle 4.4 dargestellten Anforderungen wurden für den Bereich des Lagers definiert.

Tabelle 4.4.: Logistische Prozesskette: Anforderungen - Lager

Nr.	Anforderung:	Umgesetzt:
4	**Lager**	
4.1	Wareneingang einlagern	Ware von Lieferanten einlagern
4.2	Ware umbuchen	Ware von Lieferanten einlagern
4.3	Fertigerzeugnis einlagern	Ware von Produktionsaufträgen einlagern
4.4	Fertigerzeugnis auslagern	Ware von Produktionsaufträgen einlagern

4.2.5. Qualitätsmanagement

Die in Tabelle 4.5 dargestellten Anforderungen wurden für den Bereich des Qualitätsmanagements definiert.

Tabelle 4.5.: Logistische Prozesskette: Anforderungen - Qualitätsmanagement

Nr.	**Anforderung:**	**Umgesetzt:**
5	**Qualitätsmanagement**	
5.1	Wareneingang	Angelieferte Waren sollen vor dem Buchen des WE überprüft werden
5.2	Fertigung	Qualitätsprüfung nach Fertigungsprozess

4.3. Umsetzung

Aufgrund der Herangehensweise an die Aufgabe wurden auch Customizing-Aktivitäten durchgeführt, die schon in den RDS-Inhalten implementiert sind. Hierbei fand jegliches Customizing im Buchungskreis 2020 statt.

4.3.1. Organisationsstruktur

Die Organisationsstruktur ist stark an die im System vorhandenen Best Practices orientiert, da dies laut Entwurf als Basis für das Szenario dienen sollte. Zusätzlich wurde als zusätzliche Orientierung eine Fallstudie des SAP University Competence Center Magdeburg von Stefan Wieder aus dem Jahre 2007 verwendet. Zunächst musste das System mit Hilfe des Customizing und der nachfolgend sehr häufig genutzten Transaktion SPRO für dieses Szenario angepasst werden. Die jeweiligen Organisationseinheiten müssen erstellt bzw. kopiert werden und zugeordnet werden.

Definition

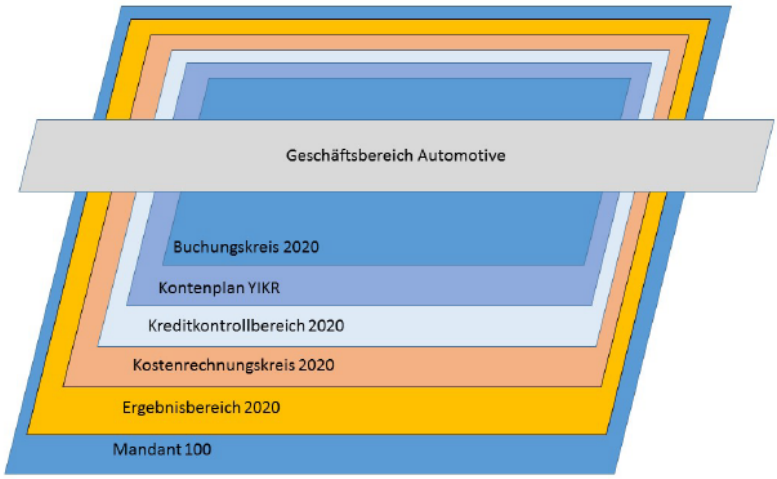


Abbildung 4.1.: Logistische Prozesskette: Gesellschaft (Unternehmen)

Gesellschaft kopieren *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Finanzwesen → Gesellschaft definieren*
Die Gesellschaft 1010 aus den Best Practices wurde kopiert und die kopierte Gesellschaft unter 2020 angelegt. In den Detaildaten werden folgende Daten gepflegt:

- Gesellschaft: 2020
- Name der Gesellschaft: Bremer Bremsanlagen
- Straße: An der Reeperbahn 10
- Postleitzahl: 28127
- Ort: Bremen
- Land: DE
- Sprachenschlüssel: DE
- Währung: EUR

Kostenrechnungskreis anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Controlling → Kostenrechnungskreis pflegen*
Der Kostenrechnungskreis wurde mit den folgenden Daten in der Tabelle ergänzt:

Tabelle 4.6.: Logistische Prozesskette: Definition - Kostenrechnungskreis

KKrs	Bezeichnung
2020	Kostenrechnungskreis 2020

In den Detaildaten des Kostenrechnungskreis wurden folgende Daten eingegeben:

- Zuordnungssteuerung: Kostenrechnungskreis analog Buchungskreis
- Währungstyp: 10
- Währung: EUR
- Kontenplan: YIKR
- GeschJahresvariante: K0

Buchungskreis anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Finanzwesen → Buchungskreis bearbeiten, kopieren, löschen, prüfen*
Um den Buchungskreis anzulegen wurde der Buchungskreis 1010 aus den Best Practices kopiert. Der neue Buchungskreis wurde mit den folgenden Daten definiert:

Tabelle 4.7.: Logistische Prozesskette: Definition - Buchungskreis

BuKr.	Name der Firma
2020	Bremer Bremsanlagen

- Ort: Bremen
- Land: DE
- Währung: EUR
- Sprache: DE

Verkaufsorganisation anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Vertrieb → Verkaufsorganisation definieren, kopieren, löschen, prüfen*
Die Verkaufsorganisation wurde mit den folgenden Daten angelegt:

Tabelle 4.8.: Logistische Prozesskette: Definition - Verkaufsorganisation

VkOrg	Bezeichnung
2020	VK Organisation 2020

Vertriebsweg anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Vertrieb → Vertriebsweg definieren, kopieren, löschen, prüfen (Vertriebsweg definieren)* Hierbei wurde kein neuer Vertriebsweg angelegt. Es wird daher der Vertriebsweg 10 (Direktverkauf) aus den Best Practices verwendet.

Sparte anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Logistik Allgemein → Sparte definieren, kopieren, löschen, prüfen (→ Sparte definieren)* Die Sparte wurde mit den folgenden Daten angelegt:

Tabelle 4.9.: Logistische Prozesskette: Definition - Sparte

Sparte	Bezeichnung
22	Automotive

Werk anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Logistik Allgemein → Werk definieren, kopieren, löschen, prüfen (→ Werk kopieren, löschen, prüfen)*

Wie beim *Buchungskreis kopieren* oder dem *Kostenrechnungskreis kopieren*, muss auch beim Werk kopieren vorgegangen werden. Das Werk 1010 aus den Best Practices wurde kopiert um das folgende Werk anzulegen:

Tabelle 4.10.: Logistische Prozesskette: Parameter - Werk

Werk	Name 1	Name 2
2020	Werk Bremsanlagen	Werk Bremsanlagen

In den Details des Werkes wurden folgende Daten gepflegt:

- Sprachenschlüssel: DE
- Straße und Hausnr: An der Reeperbahn 10
- Postleitzahl: 28217
- Ort: Bremen
- Länderschlüssel: DE

- Region: HB
- Fabrikkalender: 01

Lagerorte pflegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Materialwirtschaft → Lagerort pflegen.*

Es wurden zwei Lager für das Werk **2020** anlegt:

- Lagerort **202A** Zentrallagerort: Dieses Lager gilt als zentraler Lagerort an dem sowohl Rohstoffe, Halbfabrikate und Fertigerzeugnisse eingelagert werden.
- Lagerort **202P** Produktionslagerort: Dieses Lager gilt als Produktionslager. Es werden Produktionskomponenten zwischengelagert, welche unmittelbar in die Produktion gehen.

Diese beiden Lagerorte wurden daher mit folgenden Werten angelegt:

Tabelle 4.11.: Logistische Prozesskette: Definition - Lagerorte

Lagerort	Bezeichnung
202A	Bremsanl. Zentr.
202P	Produktion

Die Adressen der Lagerorte entsprechen der Adresse des Werkes.

Versandstellen anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Logistics Execution → Versandstelle definieren, kopieren, löschen, prüfen.*

Die Versandstelle wurden mit folgenden Werten angelegt:

Tabelle 4.12.: Logistische Prozesskette: Definition - Versandstelle

Versandstelle	Bezeichnung
2020	Bremen Bremsen

Einkaufsorganisation anlegen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Definition → Materialwirtschaft → Einkaufsorganisation pflegen*

Neue Einträge wurden ausgewählt um eine Einkaufsorganisation 2020 anzulegen. Folgende Parameter wurden eingegeben:

Tabelle 4.13.: Logistische Prozesskette: Definition - Einkaufsorganisation

EinkOrganisation	Bezeichnung EkOrg
2020	Einkaufs. Org. 2020

Zuordnung

Nachfolgend werden die Zuordnungen der jeweiligen Organisationsstrukturen anhand von Screenshots dargestellt.

Sparte – Verkaufsorganisation zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Vertrieb → Sparte - Verkaufsorganisation zuordnen*

Tabelle 4.14.: Logistische Prozesskette: Parameter - Verkauforganisation - Sparte

VkOrg	Bezeichnung	SP	Bezeichnung
2020	VK Organisation 2020	22	Automotive

Versandstelle – Werk zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Logistics Execution → Versandstelle -Werk zuordnen*

In der Zeile des Werkes 2020 wurde die Versandstelle 2020 eingetragen und gespeichert.

Tabelle 4.15.: Logistische Prozesskette: Parameter - Versandstellen - Werk

Werk	Bezeichnung	Versandstelle	Bezeichnung
2020	Werk Bremsanlagen	1010	Versandstelle 1010
2020	Werk Bremsanlagen	101R	Versandstelle 101R
2020	Werk Bremsanlagen	2020	Bremer Bremsen

Einkaufsorganisation – Buchungskreis zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Materialwirtschaft → Einkaufsorganisation - Buchungskreis zuordnen*

Tabelle 4.16.: Logistische Prozesskette: Parameter - Einkaufsorganisation – Buchungskreis

Einkaufsorg.	Bezeichnung	Buchungskreis	Name der Firma
2020	Einkaufsorg. 2020	2020	Bremer Bremsanlagen

Einkaufsorganisation – Werk zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Materialwirtschaft → Einkaufsorganisation - Werk zuordnen*

Tabelle 4.17.: Logistische Prozesskette: Parameter - Einkaufsorganisation – Werk

Einkaufsorg.	Bezeichnung	Werk	Name
2020	Einkaufsorg. 2020	2020	Werk Bremsanlagen

Verkaufsorganisation – Buchungskreis zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Vertrieb → Verkaufsorganisation - Buchungskreis zuordnen*

Tabelle 4.18.: Logistische Prozesskette: Parameter - Verkaufsorganisation – Buchungskreis

Verkaufsorg.	Bezeichnung	Buchungskreis	Name der Firma
2020	Verkaufsorg. 2020	2020	Bremer Bremsanlagen

Vertriebsweg – Verkaufsorganisation zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Vertrieb → Vertriebsweg - Verkaufsorganisation zuordnen*

Tabelle 4.19.: Logistische Prozesskette: Parameter - Vertriebsweg – Verkaufsorganisation

Verkaufsorg.	Bezeichnung	Vertriebsweg	Bezeichnung
2020	Verkaufsorg. 2020	10	Direktverkauf

Verkaufsbüro – Vertriebsbereich zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Vertrieb → Verkaufsbüro -Vertriebsbereich zuordnen*

Tabelle 4.20.: Logistische Prozesskette: Parameter - Verkaufsbüro – Vertriebsbereich

VkOrg	Bezeichnung	VWeg	Bezeichnung	SP	Bezeichnung	Vkbüro	Bezeichn.
2020	Verkaufsorg. 2020	10	Direktverkauf	22	Automotiv	200	VKBüro 200

Verkaufsorganisation – Vertriebsweg – Werk zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Vertrieb → Verkaufsorganisation - Vertriebsweg - Werk zuordnen*

Tabelle 4.21.: Logistische Prozesskette: Parameter - Verkaufsorga. – Vertriebsweg – Werk

Vkorg.	Bezeichnung	VWeg	Bezeichnung	Werk	Name
2020	Verkaufsorg. 2020	10	Direktverkauf	2020	Werk Bremsanlagen

Vertriebsbereich bilden *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Vertrieb → Vertriebsbereich bilden*

Tabelle 4.22.: Logistische Prozesskette: Parameter - Vertriebsbereich

Vkorg.	Bezeichnung	VWeg	Bezeichnung	SP	Bezeichnung
2020	Verkaufsorg. 2020	10	Direktverkauf	22	Automotive

Werk - Buchungskreis zuordnen *SPRO → SAP Referenz-IMG → Unternehmensstruktur → Zuordnung → Logistik Allgemein → Werk - Buchungskreis zuordnen*

Tabelle 4.23.: Logistische Prozesskette: Parameter - Werk - Buchungskreis

BuKr	Werk	Name des Werks	Name der Firma
2020	2020	Werk Bremsanlagen	Bremer Bremsanlagen

Buchungskreis - Kontenplan zuordnen *SPRO → SAP Referenz-IMG → Finanzwesen → Hauptbuchhaltung → Stammdaten → Sachkonten → Vorarbeiten→Buchungskreisdaten einem Kontenplan zuordnen*

Tabelle 4.24.: Logistische Prozesskette: Parameter - Buchungskreis - Kontenplan

BuKr	Name der Firma	Ort	Kontenplan
2020	Bremer Bremsanlagen	Bremen	YIKR

4.3.2. Stammdaten

Produktion

Material Es wurden insgesamt fünf Materialien für die Umsetzung der Fertigung angelegt. Das Anlegen der Material erfolgte über die Transaktion MM01. Die Organisationsebenen sind bei allen Materialien identisch und sind wie folgt festgelegt:

- Werk: 2020
- Verkaufsorganisation 2020
- Lagerort: 202A/202P
- Vertriebsweg 10 (Direktverkauf)

Nachfolgend werden die einzelnen Material mit ihren Parametern tabellarisch aufgeführt.

Tabelle 4.25.: Logistische Prozesskette: Parameter - Bremsbelag

Material	**Sicht**/Feld	**Parameter**
Bremsbelag	**Allgemeine Daten**	
	Grunddaten1	
	Allgemeine Daten	
	Materialart	Rohstoffe
	Basismenge	Stück
	Warengruppe	L002 - Rohstoffe
	Sparte	22
	Positionstypengruppe	Normalposition
	Abmessungen	
	Gewichtseinheit	KG
	Bruttogewicht	0,5
	Nettogewicht	0,4
	Vertrieb1	
	Allgemeine Daten	
	Verkaufsorga	2020
	Vertriebsweg	10
	Skontofähig	aktiv
	Steuerklasse	1 - Volle Steuer
	Vertrieb2	
	allg. Positionstyp	Normalposition
	Vertrieb: allg.	
	Allgemeine Daten	
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarfsprüfung
	Transport Daten	
	Transportgruppe	0001 - Paletten
	Ladegruppe	0001 - Kran
	Einkauf	
	Allgemeine Daten	
	Einkäufergruppe	20A
	Sonstige Daten	
	WE-Bearbeitungszeit	1 Tag
	Disposition1	
	Allgemeine Daten	
	Dispogruppe	0000 - Fremdbeschaffung

Tabelle 4.25.: Logistische Prozesskette: Parameter - Bremsbelag

Material	**Sicht**/Feld	**Parameter**
	Dispoverfahren	
	Dispomerkmal	PD - Prognoseverrechnung
	Fixierunghorizont	10
	Diponent	001
	Losgrößendaten	
	Dispolosgröße	Exakt
	Disposition2	
	Beschaffung	
	Beschaffungsart	Fremdbeschaffung
	Retrograde Entnahme	aktiv
	Produktionlagerort	202P
	Fremd.Besch.Lager	202A
	Terminierung	
	WE-Bearbeitungszeit	1 Tag
	Planlieferzeit	1 Tag
	Disposition3	
	Prognosebedarfe	
	Periodenkennzeichen	Monatlich
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarf
	Disposition4	
	Stücklistenauflösung	
	Einzel/Sammel	1 - ausschließlich Einzelbedarf
	Buchhaltung1	
	Allgemeine Bewertungsdaten	
	Bewertungsklasse	3000 - Rohstoffe
	Preisermittlung	2 - Vorgangsbezogen

Tabelle 4.26.: Logistische Prozesskette: Parameter - Festsattel

Material	**Sicht**/Feld	**Parameter**
Festsattel	**Allgemeine Daten**	
	Grunddaten1	
	Allgemeine Daten	
	Materialart	Rohstoffe
	Basismenge	Stück
	Warengruppe	L002 - Rohstoffe
	Sparte	22
	Positionstypengruppe	Normalposition

Tabelle 4.26.: Logistische Prozesskette: Parameter - Festsattel

Material	**Sicht**/Feld	**Parameter**
	Abmessungen	
	Gewichtseinheit	KG
	Bruttogewicht	0,3
	Nettogewicht	0,2
	Vertrieb1	
	Allgemeine Daten	
	Verkaufsorga	2020
	Vertriebsweg	10
	Skontofähig	aktiv
	Steuerklasse	1 - Volle Steuer
	Vertrieb2	
	allg. Positionstyp	Normalposition
	Vertrieb: allg.	
	Allgemeine Daten	
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarfsprüfung
	Transport Daten	
	Transportgruppe	0001 - Paletten
	Ladegruppe	0001 - Kran
	Einkauf	
	Allgemeine Daten	
	Einkäufergruppe	20A
	Sonstige Daten	
	WE-Bearbeitungszeit	1 Tag
	Disposition1	
	Allgemeine Daten	
	Dispogruppe	0000 - Fremdbeschaffung
	Dispoverfahren	
	Dispomerkmal	PD - Prognoseverrechnung
	Fixierunghorizont	10
	Diponent	001
	Losgrößendaten	
	Dispolosgröße	Exakt
	Disposition2	
	Beschaffung	
	Beschaffungsart	Fremdbeschaffung
	Retrograde Entname	aktiv
	Produktionlagerort	202P
	Fremd.Besch.Lager	202A

Tabelle 4.26.: Logistische Prozesskette: Parameter - Festsattel

Material	**Sicht**/Feld	**Parameter**
	Terminierung	
	WE-Bearbeitungszeit	1 Tag
	Planlieferzeit	1 Tag
	Disposition3	
	Prognosebedarfe	
	Periodenkennzeichen	Monatlich
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarf
	Disposition4	
	Stücklistenauflösung	
	Einzel/Sammel	1 - ausschließlich Einzelbedarf
	Buchhaltung1	
	Allgemeine Bewertungsdaten	
	Bewertungsklasse	3000 - Rohstoffe
	Preisermittlung	2 - Vorgangsbezogen

Tabelle 4.27.: Logistische Prozesskette: Parameter - Bremsblock

Material	**Sicht**/Feld	**Parameter**
Bremsblock	**Allgemeine Daten**	
	Grunddaten1	
	Allgemeine Daten	
	Materialart	Halbfabrikate
	Basismenge	Stück
	Warengruppe	L003 - Halbfabrikate
	Sparte	22
	Positionstypengruppe	Normalposition
	Abmessungen	
	Gewichtseinheit	KG
	Bruttogewicht	8
	Nettogewicht	7
	Vertrieb1	
	Allgemeine Daten	
	Verkaufsorga	2020
	Vertriebsweg	10
	Skontofähig	aktiv
	Steuerklasse	1 - Volle Steuer
	Vertrieb2	
	allg. Positionstyp	Normalposition

Tabelle 4.27.: Logistische Prozesskette: Parameter - Bremsblock

Material	**Sicht**/Feld	**Parameter**
	Vertrieb: allg.	
	Allgemeine Daten	
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarfsprüfung
	Transport Daten	
	Transportgruppe	0001 - Paletten
	Ladegruppe	0001 - Kran
	Einkauf	
	Allgemeine Daten	
	Einkäufergruppe	20A
	Sonstige Daten	
	WE-Bearbeitungszeit	1 Tag
	Disposition1	
	Allgemeine Daten	
	Dispogruppe	0002 - Eigenfertigung mit Primärbedarf
	Dispoverfahren	
	Dispomerkmal	PD - Prognoseverrechnung
	Fixierunghorizont	10
	Diponent	001
	Losgrößendaten	
	Dispolosgröße	Exakt
	Disposition2	
	Beschaffung	
	Beschaffungsart	Eigenfertigung
	Retrograde Entname	aktiv
	Produktionlagerort	202P
	Fremd.Besch.Lager	202A
	Terminierung	
	WE-Bearbeitungszeit	1 Tag
	Planlieferzeit	1 Tag
	Disposition3	
	Prognosebedarfe	
	Periodenkennzeichen	Monatlich
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarf
	Disposition4	
	Stücklistenauflösung	
	Einzel/Sammel	1 - ausschließlich Einzelbedarf
	Fertigungsversion	Bremsblock

Tabelle 4.27.: Logistische Prozesskette: Parameter - Bremsblock

Material	**Sicht**/Feld	**Parameter**
	Buchhaltung1	
	Allgemeine Bewertungsdaten	
	Bewertungsklasse	7900 - Halbfabrikate
	Preisermittlung	2 - Vorgangsbezogen

Tabelle 4.28.: Logistische Prozesskette: Parameter - Bremsscheibe

Material	**Sicht**/Feld	**Parameter**
Bremsscheibe	**Allgemeine Daten**	
	Grunddaten1	
	Allgemeine Daten	
	Materialart	Material Allgemein
	Basismenge	Stück
	Warengruppe	L003 - Halbfabrikate
	Sparte	22
	Positionstypengruppe	Normalposition
	Abmessungen	
	Gewichtseinheit	KG
	Bruttogewicht	8
	Nettogewicht	7
	Vertrieb1	
	Allgemeine Daten	
	Verkaufsorga	2020
	Vertriebsweg	10
	Skontofähig	aktiv
	Steuerklasse	1 - Volle Steuer
	Vertrieb2	
	allg. Positionstyp	Normalposition
	Vertrieb: allg.	
	Allgemeine Daten	
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarfsprüfung
	Transport Daten	
	Transportgruppe	0001 - Paletten
	Ladegruppe	0001 - Kran
	Einkauf	
	Allgemeine Daten	
	Einkäufergruppe	20A
	Sonstige Daten	
	WE-Bearbeitungszeit	1 Tag

Tabelle 4.28.: Logistische Prozesskette: Parameter - Bremsscheibe

Material	**Sicht**/Feld	**Parameter**
	Disposition1	
	Allgemeine Daten	
	Dispogruppe	0000 - Fremdbeschaffung
	Dispoverfahren	
	Dispomerkmal	PD - Prognoseverrechnung
	Fixierunghorizont	10
	Diponent	001
	Losgrößendaten	
	Dispolosgröße	Exakt
	Disposition2	
	Beschaffung	
	Beschaffungsart	Fremdbeschaffung
	Retrograde Entname	aktiv
	Produktionlagerort	202P
	Fremd.Besch.Lager	202A
	Terminierung	
	Planlieferzeit	1 Tag
	Disposition3	
	Prognosebedarfe	
	Periodenkennzeichen	Monatlich
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarf
	Disposition4	
	Stücklistenauflösung	
	Einzel/Sammel	1 - ausschließlich Einzelbedarf
	Buchhaltung1	
	Allgemeine Bewertungsdaten	
	Bewertungsklasse	3100 - Material Allgemein
	Preisermittlung	2 - Vorgangsbezogen

Tabelle 4.29.: Logistische Prozesskette: Parameter - Bremsanlage

Material	**Sicht**/Feld	**Parameter**
Bremsblock	**Allgemeine Daten**	
	Grunddaten1	
	Allgemeine Daten	
	Materialart	Halbfabrikate
	Basismenge	Stück
	Warengruppe	L004 - Fertigerzeugnisse
	Sparte	22
	Positionstypengruppe	Normalposition

Tabelle 4.29.: Logistische Prozesskette: Parameter - Bremsanlage

Material	**Sicht**/Feld	**Parameter**
	Abmessungen	
	Gewichtseinheit	KG
	Bruttogewicht	11
	Nettogewicht	10,5
	Verpackungsmaterialien	
	Materialgruppe PM	PG 30
	Vertrieb1	
	Allgemeine Daten	
	Verkaufsorga	2020
	Vertriebsweg	10
	Skontofähig	aktiv
	Steuerklasse	1 - Volle Steuer
	Vertrieb2	
	allg. Positionstyp	Normalposition
	Vertrieb: allg.	
	Allgemeine Daten	
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarfsprüfung
	Transport Daten	
	Transportgruppe	0001 - Paletten
	Ladegruppe	0002 - Gabelstapler
	Einkauf	
	Allgemeine Daten	
	Einkäufergruppe	20A
	Sonstige Daten	
	WE-Bearbeitungszeit	1 Tag
	Disposition1	
	Allgemeine Daten	
	Dispogruppe	0002 - Eigenfertigung mit Primärbedarf
	Dispoverfahren	
	Dispomerkmal	PD - Prognoseverrechnung
	Fixierunghorizont	10
	Diponent	001
	Losgrößendaten	
	Dispolosgröße	Exakt
	Disposition2	
	Beschaffung	
	Beschaffungsart	Eigenfertigung
	Retrograde Entname	aktiv
	Produktionlagerort	202P
	Fremd.Besch.Lager	202A

Tabelle 4.29.: Logistische Prozesskette: Parameter - Bremsanlage

Material	**Sicht**/Feld	**Parameter**
	Terminierung	
	WE-Bearbeitungszeit	1 Tag
	Planlieferzeit	1 Tag
	Disposition3	
	Prognosebedarfe	
	Periodenkennzeichen	Monatlich
	Verfügbarkeitsprüfung	Y2 - Einzelbestand, Bedarf
	Disposition4	
	Stücklistenauflösung	
	Einzel/Sammel	1 - ausschließlich Einzelbedarf
	Fertigungsversion	Bremsanlage
	Buchhaltung1	
	Allgemeine Bewertungsdaten	
	Bewertungsklasse	7920 - Fertigwaren
	Preisermittlung	2 - Vorgangsbezogen

Stückliste Für die Umsetzung des Fertigungsprozesses der Bremsanlage wurde eine entsprechende Stückliste im System angelegt. Ebenso wurde für die Fertigung des Halbfabrikats „Bremsblock“ eine Stückliste angelegt. Aus diesen Stückliste gehen die einzelnen Komponten und Mengen hervor.
Die Stückliste wurde durch die Transaktion CS01 im System angelegt. Der nachfolgende Gozintograph illustriert die Zusammensetzung des Fertigerzeugnisses.

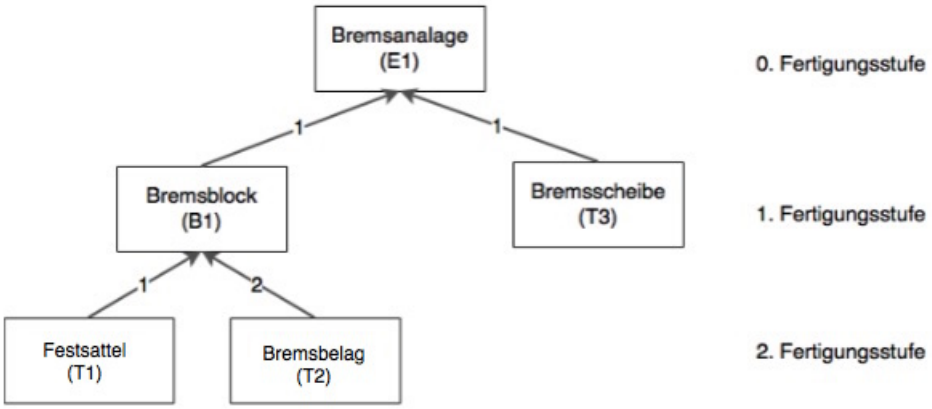


Abbildung 4.2.: Logistische Prozesskette: Produktion - Stückliste

Nachfolgend sind die Parameter detailliert aufgelistet:

Tabelle 4.30.: Logistische Prozesskette: Parameter - Stückliste Bremsblock

Bezeichnung	**Parameter**
Material	Bremsblock
Verwendung	1 - Fertigung
Stücklistennr.	00000015
Basismenge	1
Komponenten	Bremsbelag (2 Stück) Festsattel (1 Stück)
Positionstypen	Lagerposition

Tabelle 4.31.: Logistische Prozesskette: Parameter - Stückliste Bremsanlage

Bezeichnung	**Parameter**
Material	Bremsanlage
Verwendung	1 - Fertigung
Stücklistennr.	00000016
Basismenge	1
Komponenten	Bremsblock (1 Stück) Bremsscheibe (1 Stück)
Positionstypen	Lagerposition

Arbeitsplan Ein Arbeitsplan definiert alle Arbeitsschritte die zur Fertigung eines Erzeugnisses erforderlich sind. Des Weiteren gibt der Arbeitsplan die Reihenfolge der auszuführenden Arbeitsschritte vor.
Wie bereits aus der Stückliste ersichtlich, ist die Fertigung von zwei Erzeugnissen vorgesehen (Bremsblock, Bremsanlage). Demzufolge wird hierfür jeweils ein Arbeitsplan benötigt. Die Arbeitspläne wurden über die Transaktion CA01 angelegt. Nachfolgend sind die Parameter der beiden Arbeitspläne dargestellt.

Tabelle 4.32.: Logistische Prozesskette: Parameter - Arbeitsplan Bremsblock

Bezeichnung	**Parameter**
Material	Bremsblock
Werk	2020
Steuerschlüssel	YBP1
Bezeichnung	Montage der Komponenten
Basismenge	1
Vorgangsmengeneinheit	Stück
Rüstzeit	1 Min
Maschinenzeit	1 Min
Personenzeit	1 Min
Komponenten	Feststattel, Bremsbelag

Tabelle 4.33.: Logistische Prozesskette: Parameter - Arbeitsplan Bremsanlage

Bezeichnung	**Parameter**
Material	Bremsanlage
Werk	2020
Steuerschlüssel	YBP1
Bezeichnung	Montage der Komponenten
Basismenge	1
Vorgangsmengeneinheit	Stück
Rüstzeit	1 Min
Maschinenzeit	1 Min
Personenzeit	1 Min
Komponenten	Bremsblock, Bremsscheibe

Arbeitsplatz Es wurde ein zentraler Arbeitsplatz angelegt. An diesem Arbeitsplatz werden alle Produktionsschritte (welche zuvor im Arbeitsplan definiert wurden) durchgeführt. Der Arbeitsplatz wurde über die Transaktion CR01 angelegt und trägt den Namen „Montage“. Folgende Parameter wurden festgelegt:

Tabelle 4.34.: Logistische Prozesskette: Parameter - Arbeitsplatz

Bezeichnung	**Parameter**
Arbeitsplatz	Montage
Werk	2020
Arbeitsplatzart	0007 - Fertigungslinie
Verantwortlicher	001 - Work center supervisor
Planverwendung	Alle Plantypen
Retrograde Entnahme	aktiv
Vorgabewertschlüssel	SAP1 - Fertigung normal

Einkauf

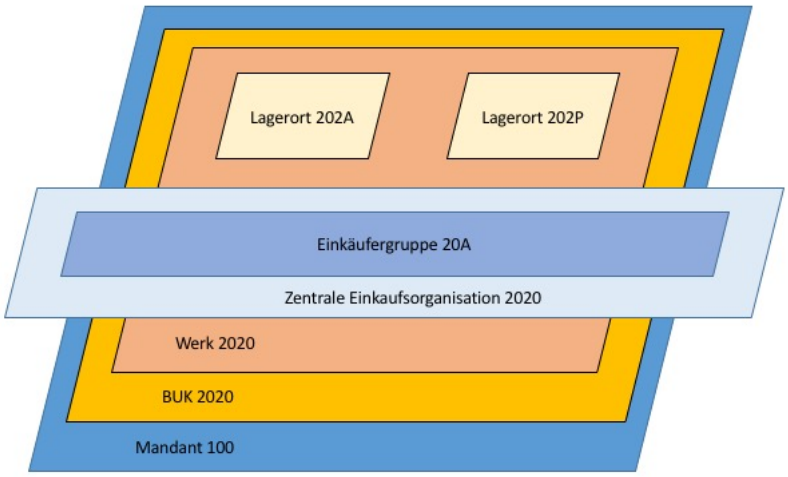


Abbildung 4.3.: Logistische Prozesskette: Einkauf

Planprimärbedarf Für die Bedarfsplanung wurden Planprimärbedarfe der einzelnen Materialien gepflegt. Diese Primärbedarfen wurden angelegt unter: *Produktion* → *Produktionsplanung* → *Programmplanung* → *Planprimärbedarf* → *Anlegen* (Alternativ funktioniert das Anlegen der Planprimärbedarfe auch über die Fiori-App „Planprimärbedarf bearbeiten“)

Die Planprimärbedarfe wurden zu Testzwecken im Laufe der Testphasen verändert. Die nachfolgenden Tabelle zeigt die Planprimärbedarf für den Monat Juli (Stand 25.7.16):

Tabelle 4.35.: Logistische Prozesskette: Parameter - Planprimärbedarf

Bezeichnung	**Parameter**
Material Planprimärbedarf	Bremsanlage 1000
Material Planprimärbedarf	Bremsblock 1000
Material Planprimärbedarf	Bremsscheibe 1000
Material Planprimärbedarf	Festsattel 1000
Material Planprimärbedarf	Bremsbelag 2000

Lieferanten Zur Produktion der Bremsanlagen werden drei fremdbeschaffte Materialien benötigt (Bremsbelag, Festsattel, Bremsscheibe). Gemäß den Anforderungen wird jedes Material über jeweils einen Lieferanten beschafft.
Hierzu wurden drei verschiedene Lieferanten über die Transaktion BP angelegt. Dabei wurden folgende Parameter festgelegt:

Tabelle 4.36.: Logistische Prozesskette: Parameter - Lieferanten (1000021)

Lieferant	**GP-Rolle**/**Sicht**/Feld	**Parameter**
1000021	**Kreditor**	
	Allgemeine Daten	
	Anschrift	
	Anrede	Gesellschaft
	Name	Automobilbedarf Adam
	Briefanrede	00
	Suchbegriffe	AUTOMOBILBEDARF ADAM
	Standardadresse	
	Straße/Nr.	Hauptstraße 42
	PLZ	22043 Hamburg
	Land	DE
	Region	HH
	Zeitzone	CET
	Identifikation	
	Organisationsdaten	
	Rechtsform	06 - Offene Handelsgesellschaft
	Steuerung	
	Steuerparameter	
	Geschäftspartnerart	001 - Partnerart 001
	Zahlungsverkehr	
	Bankverbindungen	
	ID	01
	Land	DE
	Bankschlüssel	51420500
	Bankkonto	0123456789
	Kontrollschlüssel	12
	Juristische Daten	
	Zielgruppe	3 - Großunternehmen
	Buchungskreis	
	Buchungskreis	2020 - Bremer Bremsanlagen
	Kontoführung	
	Abstimmkonto	211001

Tabelle 4.36.: Logistische Prozesskette: Parameter - Lieferanten (1000021)

Lieferant	**GP-Rolle**/**Sicht**/Feld	**Parameter**
	Lieferant	
	Einkauf	
	Einkaufsdaten	
	Konditionen	
	Bestellwährung	EUR
	Zahlungsbedingung	0001
	Mindestbestellwert	0.00
	Vorschlagswerte	
	Material	
	Einkäufergruppe	20A
	Planlieferzeit	1
	Zusätzliche Einkaufsdaten	
	Schemagruppe Lieferung	01
	Partnerrollen	
	PR	LF
	Partnerrolle	Lieferant
	Nummer	1000021
	Bedeutung	Automobilbedarf Adam

Tabelle 4.37.: Logistische Prozesskette: Parameter - Lieferanten (1000031)

Lieferant	**GP-Rolle**/**Sicht**/Feld	**Parameter**
1000031	**Kreditor**	
	Allgemeine Daten	
	Anschrift	
	Anrede	Gesellschaft
	Name	Schmidt Bremsen
	Briefanrede	03
	Suchbegriffe	BREMSEN, SCHMIDT
	Standardadresse	
	Straße/Nr.	Lange Straße 12
	PLZ	45879 Gelsenkirchen
	Land	DE
	Region	NW
	Zeitzone	CET
	Identifikation	
	Organisationsdaten	
	Rechtsform	08 - Gesellschaft mit beschränkter Haftung

Tabelle 4.37.: Logistische Prozesskette: Parameter - Lieferanten (1000031)

Lieferant	**GP-Rolle**/**Sicht**/Feld	**Parameter**
	Steuerung	
	Steuerparameter	
	Geschäftspartnerart	001 - Partnerart 001
	Zahlungsverkehr	
	Bankverbindungen	
	ID	01
	Land	DE
	Bankschlüssel	26487591
	Bankkonto	123456789
	Kontrollschlüssel	12
	Juristische Daten	
	Zielgruppe	4 - Kleinbetrieb
	Buchungskreis	
	Buchungskreis	2020 - Bremer Bremsanlagen
	Kontoführung	
	Abstimmkonto	211001
	Lieferant	
	Einkauf	
	Einkaufsdaten	
	Konditionen	
	Bestellwährung	EUR
	Zahlungsbedingung	0001
	Mindestbestellwert	0.00
	Vorschlagswerte	
	Material	
	Einkäufergruppe	20A
	Planlieferzeit	1
	Zusätzliche Einkaufsdaten	
	Schemagruppe Lieferung	01
	Partnerrollen	
	PR	LF
	Partnerrolle	Lieferant
	Nummer	1000031
	Bedeutung	Schmidt Bremsen

Tabelle 4.38.: Logistische Prozesskette: Parameter - Lieferanten (1000030)

Lieferant	**GP-Rolle**/**Sicht**/Feld	**Parameter**
1000030	**Kreditor**	
	Allgemeine Daten	
	Anschrift	
	Anrede	Gesellschaft
	Name	Stuttgarter Bremsscheiben GmbH
	Briefanrede	03
	Suchbegriffe	BREMSSCHEIBEN, STUTTGARTER BS
	Standardadresse	
	Straße/Nr.	Obstbaumallee 1
	PLZ	70173 Stuttgart
	Land	DE
	Region	BW
	Zeitzone	CET
	Identifikation	
	Organisationsdaten	
	Rechtsform	08 - Gesellschaft mit beschränkter Haftung
	Steuerung	
	Steuerparameter	
	Geschäftspartnerart	001 - Partnerart 001
	Zahlungsverkehr	
	Bankverbindungen	
	ID	01
	Land	DE
	Bankschlüssel	51420501
	Bankkonto	0123456799
	Kontrollschlüssel	12
	Juristische Daten	
	Zielgruppe	4 - Kleinunternehmen
	Buchungskreis	
	Buchungskreis	2020 - Bremer Bremsanlagen
	Kontoführung	
	Abstimmkonto	211001
	Lieferant	
	Einkauf	
	Einkaufsdaten	
	Konditionen	
	Bestellwährung	EUR
	Zahlungsbedingung	0001
	Mindestbestellwert	0.00

Tabelle 4.38.: Logistische Prozesskette: Parameter - Lieferanten (1000030)

Lieferant	**GP-Rolle**/**Sicht**/Feld	**Parameter**
	Vorschlagswerte	
	Material	
	Einkäufergruppe	20A
	Planlieferzeit	1
	Zusätzliche Einkaufsdaten	
	Schemagruppe Lieferung	01
	Partnerrollen	
	PR	LF
	Partnerrolle	Lieferant
	Nummer	1000030
	Bedeutung	Stuttgarter Bremsscheiben GmbH

Einkaufsinfosätze Um zuzuordnen welche Materialien bei welchem Lieferanten und zu welchem Preis beschafft werden können, wurden Einkaufsinfosätze im System angelegt. Hierzu wurde für den Einkaufsinfosatz für das Material Bremsbelag die Transaktion ME11 mit folgenden Eingabewerten genutzt:

- Lieferant: 1000021
- Material: Bremsbelag
- Einkaufsorganisation: 2020
- Werk: 2020
- Infotyp: Normal

In den verschiedenen Sichten wurden anschließend folgende Parameter definiert:

Tabelle 4.39.: Logistische Prozesskette: Parameter - Einkaufsinfosätze (5300000200)

Infosatz	**Sicht**/Feld	**Parameter**
	Allgemeine Daten	
	Infosatz	5300000200
	Lieferant	1000021 - Automobilbedarf Adam
	Material	Bremsbelag
	Warengruppe	L002 - Rohstoffe

Tabelle 4.39.: Logistische Prozesskette: Parameter - Einkaufsinfosätze (5300000200)

Infosatz	**Sicht**/Feld	**Parameter**
	Ursprungsdaten	
	Ursprungsland	DE
	Region	HH
	Liefermöglichkeit	
	Regellieferant	Aktiv
	Einkaufsorganisationsdaten	
	Steuerung	
	Planlieferzeit	1
	Einkäufergruppe	20A
	Normalmenge	50 ST
	Konditionen	
	Nettopreis	50.00 EUR
	Preisdatumstyp	1 - Bestelldatum

Der Einkaufsinfosatz für das Material Bremsscheibe wurde mit der Transaktion ME11 und folgenden Eingabewerten angelegt:

- Lieferant: 1000030
- Material: Bremsscheibe
- Einkaufsorganisation: 2020
- Werk: 2020
- Infotyp: Normal

In den verschiedenen Sichten wurden anschließend folgende Parameter definiert:

Tabelle 4.40.: Logistische Prozesskette: Parameter - Einkaufsinfosätze (5300000210)

Infosatz	**Sicht**/Feld	**Parameter**
	Allgemeine Daten	
	Infosatz	5300000210
	Lieferant	1000030 - Stuttgarter Bremsscheiben GmbH
	Material	Bremsscheibe
	Warengruppe	L003 - Halbfabrikate
	Ursprungsdaten	
	Ursprungsland	DE
	Region	BW
	Liefermöglichkeit	
	Regellieferant	Aktiv

Tabelle 4.40.: Logistische Prozesskette: Parameter - Einkaufsinfosätze (5300000210)

Infosatz	**Sicht**/Feld	**Parameter**
	Einkaufsorganisationsdaten	
	Steuerung	
	Planlieferzeit	1
	Einkäufergruppe	20A
	Normalmenge	50 ST
	Konditionen	
	Nettopreis	80.00 EUR
	Preisdatumstyp	1 - Bestelldatum

Der Einkaufsinfosatz für das Material Bremssattel wurde mit der Transaktion ME11 und folgenden Eingabewerten angelegt:

- Lieferant: 1000031
- Material: Bremssattel
- Einkaufsorganisation: 2020
- Werk: 2020
- Infotyp: Normal

In den verschiedenen Sichten wurden anschließend folgende Parameter definiert:

Tabelle 4.41.: Logistische Prozesskette: Parameter - Einkaufsinfosätze (5300000211)

Infosatz	**Sicht**/Feld	**Parameter**
	Allgemeine Daten	
	Infosatz	5300000211
	Lieferant	1000031 - Schmidt Bremsen
	Material	Bremssattel
	Warengruppe	L003 - Halbfabrikate
	Ursprungsdaten	
	Ursprungsland	DE
	Region	NW
	Liefermöglichkeit	
	Regellieferant	Aktiv

Tabelle 4.41.: Logistische Prozesskette: Parameter - Einkaufsinfosätze (5300000211)

Infosatz	**Sicht**/Feld	**Parameter**
	Einkaufsorganisationsdaten	
	Steuerung	
	Planlieferzeit	1
	Einkäufergruppe	20A
	Normalmenge	20 ST
	Konditionen	
	Nettopreis	90.00 EUR
	Preisdatumstyp	1 - Bestelldatum

Vertrieb

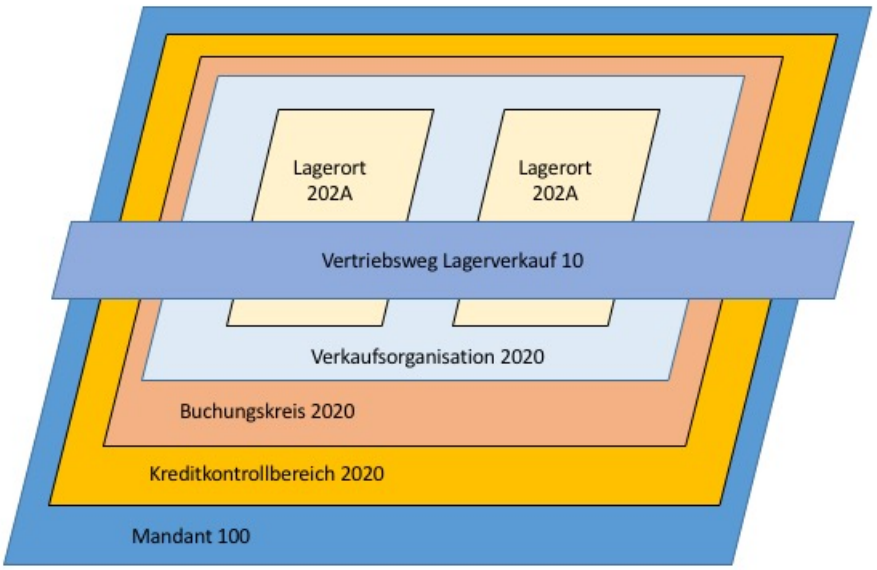


Abbildung 4.4.: Logistische Prozesskette: Vertrieb

Kunden Um die produzierten Bremsanlagen zu verkaufen, wurde drei verschiedene Kunden angelegt, die jeweils über gesonderte Konditionen verfügen.
Hierzu wurde die Transaktion BP mit der GPartner-Rolle Kunde genutzt. Bei den verschiedenen Kunden wurden folgende Parameter definiert:

Tabelle 4.42.: Logistische Prozesskette: Parameter - Kunden (1000022)

Kunde	**Sicht**/**Tab**/Feld	**Parameter**
1000022		
	Allgemeine Daten	
	Anschrift	
	Anrede	Gesellschaft
	Name	Autohandel Müller GmbH
	Briefanrede	00
	Suchbegriffe	AUTOHANDEL MÜLLER, AUTO MÜLLER
	Standardadresse	
	Straße/Nr.	Verdener Landstraße 12
	PLZ	31582 Nienburg
	Land	DE
	Region	NI
	Zeitzone	CET
	Identifikation	
	Organisationsdaten	
	Rechtsform	08 - Gesellschaft mit beschränkter Haftung
	Steuerung	
	Steuerparameter	
	Geschäftspartnerart	001 - Partnerart 001
	Vertrieb	
	Vertriebsbereich	
	Verkaufsorganisation	2020 - VK Organisation 2020
	Vertriebsweg	10 - Direktverkauf
	Sparte	22 - Automotive
	Auftrag	
	Kundenbezirk	DE0001 - Norddeutschland
	Verkaufsbüro	200
	Verkaufsgruppe	20A
	Auftr.WK	100 %
	Währung	EUR
	Versand	
	Versandstelle	2020
	Versandbedingung	01
	Preisfindung/Statistik	
	Preisgruppe	C1
	Kundenschema	01
	Versand	
	Versandbedingung	01

Tabelle 4.42.: Logistische Prozesskette: Parameter - Kunden (1000022)

Kunde	**Sicht**/**Tab**/Feld	**Parameter**
	Faktura	
	Land	DE
	Steuertyp	TTX1
	Partnerrollen	
	PR	AG, RE, RG, WE

Tabelle 4.43.: Logistische Prozesskette: Parameter - Kunden (1000032)

Kunde	**Sicht**/**Tab**/Feld	**Parameter**
1000032		
	Allgemeine Daten	
	Anschrift	
	Anrede	Gesellschaft
	Name	Karlsruher Automobil Manufaktur
	Briefanrede	03
	Suchbegriffe	AUTOMOBIL MANUFAKTUR, KARLSRUHER
	Standardadresse	
	Straße/Nr.	Lange Straße 22
	PLZ	76131 Karlsruhe
	Land	DE
	Region	NI
	Zeitzone	CET
	Identifikation	
	Organisationsdaten	
	Rechtsform	07 - Privatunternehmen
	Steuerung	
	Steuerparameter	
	Geschäftspartnerart	001 - Partnerart 001
	Vertrieb	
	Vertriebsbereich	
	Verkaufsorganisation	2020 - VK Organisation 2020
	Vertriebsweg	10 - Direktverkauf
	Sparte	22 - Automotive
	Auftrag	
	Kundenbezirk	DE0002 - Süddeutschland
	Verkaufsbüro	200
	Verkaufsgruppe	20A
	Auftr.WK	100 %
	Währung	EUR

Tabelle 4.43.: Logistische Prozesskette: Parameter - Kunden (1000032)

Kunde	**Sicht**/**Tab**/Feld	**Parameter**
	Versand	
	Versandstelle	2020
	Versandbedingung	01
	Preisfindung/Statistik	
	Preisgruppe	C1
	Kundenschema	01
	Versand	
	Versandbedingung	01
	Faktura	
	Land	DE
	Steuertyp	TTX1
	Partnerrollen	
	PR	AG, RE, RG, WE

Tabelle 4.44.: Logistische Prozesskette: Parameter - Kunden (1000033)

Kunde	**Sicht**/**Tab**/Feld	**Parameter**
1000033		
	Allgemeine Daten	
	Anschrift	
	Anrede	Gesellschaft
	Name	Autoservice Meyer
	Briefanrede	03
	Suchbegriffe	AUTOSERVICE, MEYER
	Standardadresse	
	Straße/Nr.	Potsdamer Straße 23
	PLZ	10115 Berlin
	Land	DE
	Region	BE
	Zeitzone	CET
	Identifikation	
	Organisationsdaten	
	Rechtsform	01 - Gesellschaft
	Steuerung	
	Steuerparameter	
	Geschäftspartnerart	001 - Partnerart 001

Tabelle 4.44.: Logistische Prozesskette: Parameter - Kunden (1000033)

Kunde	**Sicht**/**Tab**/Feld	**Parameter**
	Vertrieb	
	Vertriebsbereich	
	Verkaufsorganisation	2020 - VK Organisation 2020
	Vertriebsweg	10 - Direktverkauf
	Sparte	22 - Automotive
	Auftrag	
	Kundenbezirk	DE0004 - Ostdeutschland
	Verkaufsbüro	200
	Verkaufsgruppe	20A
	Auftr.WK	100 %
	Währung	EUR
	Versand	
	Versandstelle	2020
	Versandbedingung	01
	Preisfindung/Statistik	
	Preisgruppe	C1
	Kundenschema	01
	Versand	
	Versandbedingung	01
	Faktura	
	Land	DE
	Steuertyp	TTX1
	Partnerrollen	
	PR	AG, RE, RG, WE

Konditionssätze (DCM1) anlegen Im Anschluss an die Einstellungen im Customizing wurden Konditionssätze für die Preisfindung für jeden Kunden angelegt. Hierzu wurde die Transaktion VK11 genutzt. Die Konditionssätze wurden mit der Konditionsart DCM1 (Kunde/Material) angelegt, da hierdurch eine Preisfindung für das Material je Kunde möglich ist. Die Konditionssätze wurden mit den folgenden Daten angelegt:

Konditionssatz für Debitor 1000022:

- Verkaufsorganisation: 2020
- Vertriebsweg: 10
- Kunde: 1000022

Tabelle 4.45.: Logistische Prozesskette: Parameter - Konditionssatz 1000022

Material	Betrag	Einheit	Mengeneinheit	Rechenregel	Bezug
Bremsanlage	1.000	Euro	ST	C	C

Der Preis 1000 Euro dient hierbei als Basispreis für das Material Bremsanlage. Zusätzlich ist eine Staffelung eingerichtet worden, um Mengenrabatte zu realisieren (F2). Hierbei wurden in der Tabelle Staffeln folgende Daten hinterlegt:

Tabelle 4.46.: Logistische Prozesskette: Staffelung - Debitor 1000022

Staffelart	Staffelmenge	ME	Betrag	Einheit
ab	1	ST	1000	Euro
	1000	ST	950	Euro
	5000	ST	800	Euro

Konditionssatz für Debitor 1000022:

- Verkaufsorganisation: 2020
- Vertriebsweg: 10
- Kunde: 1000032

Tabelle 4.47.: Logistische Prozesskette: Parameter - Konditionssatz 1000032

Material	Betrag	Einheit	Mengeneinheit	Rechenregel	Bezug
Bremsanlage	950	Euro	ST	C	C

Bei diesem Konditionssatz ist ein fester Preis von 950 Euro ohne Mengenrabatt hinterlegt.

Konditionssatz für Debitor 1000022:

- Verkaufsorganisation: 2020
- Vertriebsweg: 10
- Kunde: 1000033

Tabelle 4.48.: Logistische Prozesskette: Parameter - Konditionssatz 1000033

Material	Betrag	Einheit	Mengeneinheit	Rechenregel	Bezug
Bremsanlage	980	Euro	ST	C	C

Der Preis 980 Euro dient hierbei als Basispreis für das Material Bremsanlage. Zusätzlich

ist eine Staffelung eingerichtet worden, um Mengenrabatte zu realisieren (F2). Hierbei wurden in der Tabelle Staffeln folgende Daten hinterlegt:

Tabelle 4.49.: Logistische Prozesskette: Staffelung - Debitor 1000022

Staffelart	Staffelmenge	ME	Betrag	Einheit
ab	1	ST	980	Euro
	1000	ST	950	Euro
	2000	ST	900	Euro

Verpackungsmaterial Das Verpacken der verkauften Bremsanlagen stellt einen Teil des Lieferprozesses dar. Um dies zu ermöglichen war das Anlegen eines Verpackungsmaterials notwendig. Dies wurde mit der TA MM01 durchgeführt.
Hierbei wurden folgende Sichten gepflegt:

- Grunddaten 1
- Grunddaten 2
- Vertrieb: VerkaufsorgDaten 1
- Vertrieb: VerkaufsorgDaten 2
- Vertrieb: allg./Werksdaten

Als Materialart wurde die Materialart VERP (Verpackungsmaterial gewählt). Bei den Organisationseben wurde das Werk 2020, die Verkaufsorganisation 2020 sowie der Vertriebsweg 10 gewählt. In den Sichten wurden folgende Parameter gepflegt:

Tabelle 4.50.: Logistische Prozesskette: Parameter - Karton klein

Sicht	Feld	Parameter
Grunddaten 1	Bruttogewicht	0.5kg
	Gewichtseinheit	KG
Vertrieb: allg./Werk	Packmittelart	30
	Zul. VerpGewicht	100kg

4.3.3. Produktion

Fertigungssteuerung

Gemäß den vorab definiert Anforderungen sollen die Fertigungsaufträge automatisiert freigegeben werden. Hierfür musste ein Fertigungssteuerungsprofil angelegt werden. Das Anlegen erfolgte über *SPRO → SAP Referenz-IMG → Produktion → Fertigungssteue-*

rung → Stammdaten → Fertigungssteuerungsprofil definieren .
Das Fertigungssteuerungsprofil wurden mit folgenden Parametern angelegt:

Tabelle 4.51.: Logistische Prozesskette: Parameter - Fertigungssteuerung

Bezeichnung	Parameter
Steuerungsprofil (Bezeichnung)	FSP1
Bei Eröffnung: Freigabe durchführen	aktiv
Bei Freigabe: Auftrag terminieren	aktiv
Materialverfügbarkeitsprüfung: Teilmengen bestätigen	aktiv
Wareneingang: Auto. Wareneingang	aktiv
Auftragsart	Lagerfertigung

Im nächsten Schritt wurde das Fertigungssteuerungsprofil den Materialien Bremsblock und Bremsanlage zugewiesen. Hierzu wurde in der Sicht Arbeitsvorbereitung im Feld Fertigungssteuerungsprofil der Parameter „FSP1“ eingetragen.

Bedarfsplanung aktivieren

Die Bedarfsplanung wurde durch die Transaktion „OMDU“ aktiviert. Hierbei wurden folgende Parameter gepflegt:

Tabelle 4.52.: Logistische Prozesskette: Parameter - Bedarfsplanung

Werk	Name des Werkes	Bedarfsplanung
2020	Bremer Bremsanlagen	aktiv

Kalkulationsschema

In der zweiten Projektphase sollte es möglich sein, die real anfallenden Kosten für hergestellt Produkte zu ermitteln. Für die Kalkulation von Material- und Produktionskosten mussten einige Schritte im System durchgeführt werden. So muss als Erstes sichergestellt werden, dass ein Preis im Materialstamm eingetragen ist. Dies geschieht in der Sicht „Buchhaltung1“ unter Standardpreis. Des Weiteren müssen in den Arbeitsplänen die zugehörigen Leistungsarten zu den jeweiligen Positionen Personalkosten, Maschinenkosten und Rüstzeit eingetragen werden. Dies erfolgt in den entsprechenden Arbeitsplan unter der Sicht „Kalkulation“. Ebenfalls muss sichergestellt werden, dass jeder Leistungsart eine Kostenstelle zugeordnet ist. Eine abschließende Voraussetzung für die Kalkulation der Produktion ist die Erstellung eines passenden Elementeschemata.

Elementeschema definieren Im Buchungskreis 2020 wird mit dem Kontenplan „YIKR“ gearbeitet. Für diesen Kontenplan existiert jedoch kein Elementschema. Das Elementeschema ist für die Zuordnung von Kostenarten und Verrechnungskonto zuständig, was für die Produktionskostenkalkulierung von Relevanz ist. Über die Transaktion „OKTZ“ kann die Definition eines neuen Elementeschemata erfolgen. Das Elementsche-

ma „Y2“ wurde mit folgenden Elementen definiert:

Tabelle 4.53.: Logistische Prozesskette: Produktionskosten - Elementzuordnung

Element	Bezeichnung	Kostenart
201	Personenzeit	11111
202	Maschienenzeit	11113
203	Rüstzeit	11112
211	Einzelk. Material	11115

Abschließend erfolgt die Zuordnung des neue definierten Elementeschmatas zum Buchungskreis. Dies erfolgt über die Dialogstruktur “’Zuordnung Organisationseinheit – Elementeschema“’.
Sind diese Schritte erfolgreich im System umgesetzt, sollte eine korrekte Kalkulation von Material- und Produktionskosten möglich sein.

4.3.4. Finanzen

Sachkonten anlegen

Über die Transaktion FS02 wurden verschieden Sachkonten definiert. Nachfolgende sind die angelegten Konten mit den jeweiligen Parametern zusammengefasst aufgeführt:

Tabelle 4.54.: Logistische Prozesskette: Finanzen - Sachkonten

Sachkonto	Bezeichnung	Sachkontenart	Kostengruppe
111112	Betriebskosten	Sekundärkosten	Sekundärkosten
111113	Maschienenkosten	Sekundärkosten	Sekundärkosten
111114	Materialkosten	Sekundärkosten	Sekundärkosten
111116	Forderung Inland	Bestandskonto	Abgrenzungskonto-KR-DE
399110	Materialkonto	Bestandskonto	Erfolgskonto
200000	Bestand Rohstoffe	Bestandskonto	Konto der Materialwirtschaft
211001	Verbindlichkeiten	Bestandskonto	Abgrenzungskonto-KR-DE
280001	Bank1 - Geldausgang	Bestandskonto	Konto für flüssige Mittel
293000	WE-RE-Verrechnung-Fremdbezug	Bestandskonto	Konto der Materialwirtschaft
440000	Verbindlichkeiten - Inland	Bestandskonto	Sachkonto allgemein

Die angelegten Sachkonten müssen in den Buchungskreis 2020 aufgenommen werden. Dies erfolgt unter:

SPRO → SAP Referenz-IMG → Finanzwesen → Hauptbuchhaltung → Stammdaten → Sachkonten → Anlegen und Bearbeiten der Sachkonten→Sachkonten bearbeiten (Einzelbearbeitung)
→Buchungskreisdaten bearbeiten.
Hier wird das enstprechende Konto ausgewählt und im Feld „Buchungskreis“ der Parameter 2020 gesetzt. Dieser Vorgang wird für alle Sachkonten die in der Tabelle 4.54 aufgelistet sind durchgeführt.

Kostenstellen anlegen

Um Kosten korrekt abrechnen zu können, müssen entsprechende Kostenstellen angelegt werden. Daher wurde eine Kostenstelle für den Einkauf sowie eine Kostenstelle für die Produktion angelegt. Das Anlegen der Kostenstellen erfolgte über die Transaktion „KS01“. Es gibt im Buchungskreis 2020 zwei Organisationseinheiten in denen Kosten entstehen. Dementsprechend wurde eine Kostenstelle für den Einkauf und eine Kostenstelle für die Produktion angelegt. Beim Anlegen der Kostenstelle ist darauf zu achten, dass Kostenstellenart korrekt hinterlegt ist. In dem Fall der Kostenstelle „Produktion“ ist die korrekte Art bspw. „F“ für Produktion.

Leistungsarten anlegen

Leistungsarten definieren die erbrachten Leistungen, welche in den entsprechenden Kostenstellen entstehen. Der Zusammenhang von Leistungsarten und Kostenstellen ist Voraussetzung für die korrekte Abrechnung des Kostenrechnungskreis. Im System wurden in der Transaktion „KL01“ die zwei Leistungsarten „Personalkosten“ und „Betriebskosten“ mit folgenden Parametern definiert:

Tabelle 4.55.: Logistische Prozesskette: Finanzen - Leistungsarten

Leistungseinheit:	H (Stunde)
Kostenstellenart:	F (Produktion)
Leistungsartentyp:	1 (manuelle Erfassung/Verrechnung)
Tarifkennzeichen:	1 (automatisch auf Basis Planleistung)

Kontenfindung

Sobald im System eine Kontierung durchgeführt wird, sollte das Systems in der Lage sein, automatisch zu ermitteln auf welches Konto die entsprechenden Beträge verbucht werden sollten. Hierzu ist ein Customzing der Kontenfindungstabellen unter der Transaktion OBYC zwingend notwendig. In den Kontenfindungstabellen wurde hauptsächlich in der Tabelle „GBB- Gegenbuchung zur Bestandsbuchung“ gearbeitet. In der „GBB“ wurden folgende Zuweisungen in der Kontenfindungstabellen getätigt:

Tabelle 4.56.: Logistische Prozesskette: Finanzen - Kontenfindungstabellen

Bewertung	Modifikation	Bewertungsklasse	Soll	Haben
-	AUF	3000	399110	399110
-	AUF	3100	399110	399110
-	VKA	3000	399110	399110
-	VKA	3100	399110	399110
-	VNG	3000	399110	399110
-	VNG	3100	399110	399110
-	VXX	3000	399110	399110
-	VXX	3100	399110	399110
-	ZNG	3000	399110	399110
-	ZNG	3100	399110	399110
-	ZOB	3000	399110	399110
-	ZOF	3000	399110	399110
-	ZOF	3100	399110	399110
0001	AUA	7920	399110	399110
0001	AUF	7900	399110	399110
0001	AUF	7920	399110	399110
0001	ZOF	3000	111115	111115
0001	VAX	3000	399110	399110
0001	VAX	7920	399110	399110
0001	VAY	3000	399110	399110
0001	VAY	7920	399110	399110
0001	VBR	3000	111115	111115
0001	VBR	3100	111115	111115
0001	VBR	7900	111115	111115
0001	VBR	7920	111115	111115
0001	VKA	7920	399110	399110
0001	ZOB	3000	111115	111115
0001	ZOB	3100	111115	111115
0001	ZOB	7900	399110	399110
0001	ZOB	7920	399110	399110

4.3.5. Vertrieb

Preisfindung

Laut den Anforderungen soll die Preisfindung je nach Kunden erfolgen. Hierzu mussten nachfolgende Schritte umgesetzt werden.

Gemeinsame Vertriebswege definieren *SPRO → SAP Referenz-IMG → Vertrieb → Stammdaten → Gemeinsame Vertriebswege definieren*
Dort wurde gemäß den Best Practices ein Eintrag für das Szenario ergänzt. Dabei wurden

folgende Daten eingegeben.

Tabelle 4.57.: Logistische Prozesskette: Parameter - Gemeinsame Vertriebswege definieren

VkOrg	Vertriebsweg	Vweg Kond.	Vweg Ku/Ma
2020	10	10	10

Gemeinsame Sparten definieren *SPRO → SAP Referenz-IMG → Vertrieb → Stammdaten → Gemeinsame Sparten definieren*

Tabelle 4.58.: Logistische Prozesskette: Parameter - Gemeinsame Sparte definieren

VkOrg	SP	SP Ko.	SP Ku.
2020	22	22	22

Kalkulationsschema definieren und zuordnen *SPRO → SAP Referenz-IMG → Vertrieb → Grundfunktionen → Preisfindung → Steuerung der Preisfindung → Kalkulationsschema definieren und zuordnen → Kalkulationsschema pflegen*
Hier wurden die Einträge A2020 und Y2020 aus den Einträgen aus den Best Practices kopiert.
SPRO → SAP Referenz-IMG → Vertrieb → Grundfunktionen → Preisfindung → Steuerung der Preisfindung → Kalkulationsschema definieren und zuordnen → Kalkulationsschemaermittlung festlegen
Folgende Einträge wurden ergänzt:

Tabelle 4.59.: Logistische Prozesskette: Parameter - Kalkulationsschema

VkOrg	Vweg	SP	BeSm	KuSm	Kal.Sm	KArt
2020	10	22	A		A2020	PPR0
2020	10	22	A	01	A2020	PPR0
2020	10	22	Y1		Y2020	PPR0
2020	10	22	Y1	01	Y2020	PPR0

Kundenbonus

Neben Staffelrabatten die Kunden gewährt werden, wenn Sie eine bestimmte Menge an einem bestimmten Produkt abnehmen, soll es auch möglich sein, dass Kunden einen Bonus erhalten. Beispielsweise beim Erreichen einer bestimmten Jahresbestellmenge. Um dies umzusetzen, waren verschiedene Einstellungen im Customizing notwendig.
Zunächst musste hierzu eine neue Konditionsart gepflegt werden. Die Zugriffsfolge „NA00“ existiert bereits in den RDS und konnte daher genutzt werden.
SPRO → Vertrieb → Grundfunktionen → Naturalrabatt → Konditionstechnik für Naturalrabatt → Konditionsarten pflegen Hier wurde folgender Eintrag festgehalten:

Tabelle 4.60.: Logistische Prozesskette: Pflegen der Konditionsart

KArt	Bezeichnung	ZuFg	Bezeichnung
FREE	Free Goods	NA00	Naturalrabatt(SD)

Leider wurde hierbei der SAP-Namensraum genutzt. Sollte bei einem folgenden Update diese Konditionsart enthalten sein, würde diese angelegte überschrieben werden.
Im nächsten Schritt wurde ein Kalkulationsschema für diese Konditionsart gepflegt: *SPRO → Konditionstechnik für Naturalrabatt → Kalkulationsschema pflegen*

Tabelle 4.61.: Logistische Prozesskette: Kalkulationsschema für Konditionsart pflegen

Schema	Bedeutung
FREE	Free Goods

Unter Steuerung wurden zu diesem Kalkulationsschema folgende Einstellungen getroffen:

Tabelle 4.62.: Logistische Prozesskette: Steuerung der Kalkulationsschema

Stufe	Zähler	KArt	Bezeichnung
10	10	FREE	Free Goods

Dieses Kalkulationsschema muss daraufhin der Verkaufsorganisation, die es anwendet, zugeordnet werden (Ermittlung Naturalrabattschema Verkauf). Dies geschieht über die *TA V/N6*. In dieser Transaktion wurden folgende Parameter eingetragen:

Tabelle 4.63.: Logistische Prozesskette: Zuordnung - Kalkulationsschema zu Verkaufsorga.

VkOrg	VWeg	SP	BeSm	KuSm	Schema
2020	10	22	A		FREE
2020	10	22	A	01	FREE
2020	10	22	Y1		FREE
2020	10	22	YI	01	FREE

Die Drauf- bzw. Dreingaben, die der Kunde durch die Naturalrabatte erhält, sollen kostenlos für den Kunden sein. Somit muss die Preisfindung bei diesen Positionen (Positionstyp TANN) deaktiviert werden. Dies kann über *SPRO → Vertrieb → Grundfunktionen → Naturalrabatt → Steuerung der Preisfindung für Naturalrabatt* erfolgen. Abschließend muss noch die Positionstypenfindung für Naturalrabattpositionen angepasst werden. Über *SPRO → Vertrieb → Grundfunktionen → Naturalrabatt → Positionstypenfindung für die Naturalrabattposition* wurden folgende Einträge ergänzt:

Tabelle 4.64.: Logistische Prozesskette: Positionstypenfindung für Naturalrabattpositionen

VArt	MTPOS	Verw	PsTyÜPos	PsTyD	PsTyM	PsTym	PsTyM
SO02	NORM			TAN		TAQ	TANN
SO02	NORM	FREE	TAN	TANN			

Ware Verpacken

Das Verpacken der verkauften Waren stellt einen wesentlichen Prozess innerhalb der Lieferung dar. Hierzu mussten verschiedene Einstellungen im Customizing vorgenommen werden. Hierbei wurde zur Orientierung ein Beispiel der SAP[1] genutzt.

Packmittelart definieren

Im ersten Schritt wurden eine neue Packmittelart definiert.
SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Verpacken → Packmittelarten definieren

Tabelle 4.65.: Logistische Prozesskette: Parameter - Packmittelart

Packmittelart	Bezeichnung
0030	Karton (klein)

Materialgruppe für Packmittel definieren Nach der Definition der Packmittelarten wurde eine Materialgruppe für Packmittel definiert. *SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Verpacken → Materialgruppe Packmittel definieren*

Tabelle 4.66.: Logistische Prozesskette: Parameter - Materialgruppe für Packmittel

GrPM	Bezeichnung
PG30	Verpackungsgruppe 30

Zuordnung: Packmittelgruppe und Packmittelart Anschließend mussten der Packmittelart die erlaubten Packmittel zugeordnet werden.
SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Verpacken → Packmittelarten definieren

[1] `http://help.sap.com/saphelp_scm41/helpdata/de/bd/608ad65a56418bb9dcb75ae3366c28/content.htm`

Tabelle 4.67.: Logistische Prozesskette: Zuordnung - Packmittelgruppe und Packmittelart

Matgr. PM	Bezeichnung	PMArt	Bezeichnung
PG30	Verpackungsgruppe 30	0030	Karton (klein)

Packvorschrift anlegen Um beim Verpackprozess Packvorschläge zu erhalten, ist die Definition einer Packvorschrift nötig (TA POP1).
In dieser Transaktion wurde die Packvorschrift ICH41_KARTON_KLEIN angelegt, welche dazu dient Bremsanlagen in kleine Kartons zu verpacken.
Im Reiter Komponenten wurden folgende Parameter eingegeben:

Tabelle 4.68.: Logistische Prozesskette: Parameter - Packvorschrift

Positionstyp	Komponente	Soll-Menge	Minimale Menge
P	Karton klein	1	
M	Bremsanlage	4	1

Materialgruppe PM im Material Bremsanlage zuordnen Über die TA MM02 wurde das Material Bremsanlage um die Materialgruppe PM ergänzt. Hier wurde in der Sicht Grunddaten 1 im entsprechenden Feld der Wert PG30 ergänzt.

Findungssatz für Packvorschrift anlegen Um einen Findungssatz für die Findung von Packvorschriften anzulegen wurde die TA POF1 genutzt. Hier wurde der Findungssatz Z001 angelegt. Die Schlüsselkombination erfolgt nach dem Material. In dem Findungssatz wurden folgende Daten definiert:

Tabelle 4.69.: Logistische Prozesskette: Parameter - Findungssatz für Packvorschrift

Material	Bezeichnung	Packvorschrift
Bremsanlage	Bremsanlage	ICH41_KARTON_KLEIN

Zuordnung: Konditionsart und Zugriffsfolge Damit die Packvorschrift je nach Material automatisch gefunden werden kann, war eine Zuordnung der Konditionsart zur Zugriffsfolge notwendig. Dazu wurde mit der TA OFP3 folgende Zuordnung realisiert:

Tabelle 4.70.: Logistische Prozesskette: Zuordnung - Konditionsart und Zugriffsfolge

KArt	Bezeichnung	ZuFg	Bezeichnung
Z001	Find. Packvorschrift	SHIP	Findung Versand-Packvorschrift

4.4. Nutzbarkeit der RDS bei der Implementierung der logistischen Prozesskette

Im Folgenden soll Aufschluss darüber gegeben werden, inwiefern die Inhalte der RDS Unterstützung bei der Realisierung einer logistischen Prozesskette bieten können. Dabei soll aufgezeigt werden, inwiefern das RDS bei der realen, produktiven Implementierung eines SAP S/4HANA-Systems unterstützend wirken kann und inwiefern es nutzbar ist. Hierzu soll dargestellt werden, welche Customizingaktivitäten zusätzlich zur Erfüllung der definierten Anforderungen notwendig waren.

4.4.1. Überblick RDS-Unterstützung

Die nachfolgende Tabelle bietet einen Überblick über die Unterstützung durch RDS-Inhalte. In den nachfolgenden Kapiteln, wird der über die RDS-Inhalte hinausgehende Customizingaufwand beschrieben.

Tabelle 4.71.: Logistische Prozesskette: RDS-Überblick

Anforderung	**RDS-Unterstützung**	**RDS-Prozess**
Einkauf		
Verwaltung von Bezugsquellen	vorhanden	Beschaffung von Direktmaterialien (J45-DE)
Bestellung anlegen	vorhanden	Beschaffung von Direktmaterialien (J45-DE)
Bestellanforderung erstellen	vorhanden	Beschaffung von Direktmaterialien (J45-DE)
Bestellanforderung in Bestellung umwandeln	vorhanden	Beschaffung von Direktmaterialien (J45-DE)
Wareneingang	vorhanden	Beschaffung von Direktmaterialien (J45-DE)
Lieferantenrechnung	vorhanden	Beschaffung von Direktmaterialien (J45-DE)
Lieferpläne	keine	-
Quälitatsmanagment	keine	-
Gutschriftverfahren	keine	-
Produktion		
MRP-Lauf einplanen	vorhanden	Materialbedarfsplanung (J44)
Materialunterdeckungen überwachen	vorhanden	Materialbedarfsplanung (J44)
Materialunterdeckungen verwalten	vorhanden	Materialbedarfsplanung (J44)
Materialbereitstellung	vorhanden	Lagerfertigung - Fertigungsindustrie (BJ5-DE)

Tabelle 4.71.: Logistische Prozesskette: RDS-Überblick

Anforderung	**RDS-Unterstützung**	**RDS-Prozess**
Fert.auftrag anlegen	vorhanden	Lagerfertigung -Fertigungsindustrie (BJ5-DE)
Fert.sauftrag durchführen	vorhanden	Lagerfertigung - Fertigungsindustrie (BJ5-DE)
Materialbewegungen	vorhanden	Lagerfertigung - Fertigungsindustrie (BJ5-DE)
Vertrieb		
Kundenauftrag anlegen	vorhanden	Verkauf ab Lager (BD9)
Auslieferung anlegen	vorhanden	Verkauf ab Lager (BD9)
Kommissionierung	vorhanden	Verkauf ab Lager (BD9)
Warenausgang buchen	vorhanden	Verkauf ab Lager (BD9)
Faktura anlegen	vorhanden	Verkauf ab Lager (BD9)
Naturalrabatt	vorhanden	Naturalrabattabwicklung (BKA-DE)
Streckengeschäft	vorhanden	Streckengeschäft (ohne Lieferavis) (BDK)
Ware verpacken	keine	-
Lieferpläne	keine	-
Gutschriftverfahren	keine	-

4.4.2. Überblick Fiori-Durchdringung

Tabelle 4.72.: Logistische Prozesskette: Überblick - Fiori-Durchdringung

Prozess	**Fiori-Durchdringung**	**Begründung**
Einkauf		
Bestellung anlegen	Hoch	Vollständig in Fiori umgesetzt
BAnf anlegen	Mittel	Tile vorhanden, WebDynpro wird geladen (TA ME51N)
BAnf zu Bestellung umwandeln	Hoch	Vollständig in Fiori umgesetzt
Bestellungen freigeben	Niedrig	Prinzipiell laut RDS über die „Mein Eingang“-App möglich, allerdings ist dieser Weg nicht funktionsfähig

Tabelle 4.72.: Logistische Prozesskette: Überblick - Fiori-Durchdringung

Prozess	**Fiori-Durchdringung**	**Begründung**
WE buchen	mittel	Wareneingänge zu Bestellungen sind vollständig in Fiori umgesetzt. Wareneingänge aus z.B. Lieferplänen können nur in der App „Warenbewegung buchen“ gebucht werden. Hierbei wird allerdings nur ein WebDynpro geladen (TA MIGO)
Lieferantenrechnung erfassen	Hoch	Vollständig in Fiori umgesetzt
Lieferpläne anlegen	Mittel	Tiles für das Anlegen von Lieferplänen und Einteilungen vorhanden, WebDynpros weden geladen (ME31L und ME38)
Gutschriften anlegen	Niedrig	Kein Tile vorhanden. WebDynpro manuell über TA aufrufbar.
Produktion		
MRP-Lauf durchführen	Hoch	Vollständig in Fiori umgesetzt, Tile MRP-Materialprobleme ohne Funktion
Materialdeckung auswerten	Hoch	Vollständig in Fiori umgesetzt
Fertigungsaufträge anlegen	Mittel	Tile vorhanden, WebDynpro wird geladen (TA CO01)
Fertigungsaufträge rückmelden	Niedrig	Kein Tile/App vorhanden, WebDynpro kann manuell über TA aufgerufen werden
Produktionskostenkalkulation durchführen	Niedrig	Kein Tile/App vorhanden, WebDynpro kann manuell über TA aufgerufen werden
Vertrieb		
Kundenauftrag anlegen	Mittel	Tile vorhanden, WebDynpro wird geladen (TA VA01)
Auslieferung anlegen	Mittel	Tile vorhanden, WebDynpro wird geladen (TA VL01N)
Kommissionierung durchführen	Mittel	Tile vorhanden (Auslieferung ändern), WebDynpro wird geladen (TA VL02N)
Ware verpacken	Mittel	Tile vorhanden (Auslieferung ändern), WebDynpro wird geladen (TA VL02N)

Tabelle 4.72.: Logistische Prozesskette: Überblick - Fiori-Durchdringung

Prozess	**Fiori-Durchdringung**	**Begründung**
WA buchen	Mittel	Tile vorhanden (Auslieferung ändern), WebDynpro wird geladen (TA VL02N)
Faktura durchführen	Mittel	Tile vorhanden, WebDynpro wird geladen (TA VF01)
Kundenbonus einrichten	Niedrig	Kein Tile vorhanden, WebDynpro über TA aufrufbar
Gutschrift anlegen	Mittel	Tile vorhanden (Faktura anlegen), WebDynpro wird geladen (TA VF01)

4.4.3. Einkauf

Der Einkauf beinhaltet alle Prozesse die nötig sind, um Einzelteile bei Lieferanten zu bestellen. Sofern Bedarf an Material entsteht, kann Ware bestellt werden. Dies kann zum einen durch das manuelle Anlegen von Bestellungen erfolgen und zum anderen durch den festgestellten Bedarf und der daraus resultierenden Bestellanforderung geschehen. Wenn eine Bestellanforderung gestellt wird, hat der Einkäufer die Möglichkeit diese Bestellanforderung in eine Bestellung umzuwandeln. Übersteigt eine Bestellung ein bestimmtes finanzielles Volumen, muss diese je nach Freigabestrategie von einem Einkaufsleiter genehmigt werden.
Der nächste Schritt ist die Buchung des Wareneingangs durch den Lageristen, nachdem die Bestellung mit den bestellten Materialien eingegangen ist. Mit der Ware wird zudem eine Lieferantenrechnung empfangen, welche daraufhin vom Kreditorenbuchhalter im System erfasst werden kann.

RDS-Stammdaten

Zur Produktion der Bremsanlagen werden drei fremdbeschaffte Materialien benötigt (Bremsbelag, Festsattel, Bremsscheibe). Gemäß den Anforderungen wird jedes Material über jeweils einen Lieferanten beschafft.
Die RDS-Inhalte bieten verschiedene vorkonfigurierte Lieferanten, wovon insbesondere die Inlandslieferanten für die umgesetzten Anforderungen als Vorlage genutzt werden konnten:

Tabelle 4.73.: Logistische Prozesskette: RDS - Lieferanten

Kreditor	**Details**
10300001	**Inlandslieferant 1**
10300002	**Inlandslieferant 2**
17300001	Domestic US Supplier 1
17300002	Domestic US Supplier 1

Zusätzlich sind im RDS zahlreiche vorkonfigurierte Materialien vorhanden:

Tabelle 4.74.: Logistische Prozesskette: RDS - Material (Einkauf)

Material	**Details**
SG21	- VERÄNDERT -
TG12	- VERÄNDERT -
SG24	- VERÄNDERT -
SG124	- VERÄNDERT -
SG22	- VERÄNDERT -
TG10	Handelsware 10, PD, Strecke
TG11	Handelsware 11, PD, Normaler Handel
TG13	Handelsware 13, Bestellpunkt, Strecke
TG14	Handelsware 14, PD, Zukauf, H14
TG20	Handelsware 20, Bestellpunkt, Serialnr.
TG0001	Handelsware für Verbrauch
TG0012	Handware 0012, Meldebest. Reg.Besch.
TG21	HAWA 21, Bestellpunkt, FIFO Charge
TG22	HAWA 22, Bestellpunkt, Verfalldatum Charge
TG0011	- VERÄNDERT -
RP001	Leergut, ND
RM09	RAW09, PD, FIFO Charge
RM12	RAW12, PD, Serienfertigung
RM120	RAW120, PD, Qualitätsgeprüft
RM122	RAW122, PD, FIFO Charge, Import
RM124	RAW124, VB, Verbrauch, Fixlagerplatz
RM128	RAW128, PD, Konsignation
RM13	RAW13, PD, Lohnbearbeitung
RM14	RAW14, PD, Lohnbearbeitung
RM15	RAW15, PD
RM16	RAW16, PD
RM17	RAW17, PD
RM18	RAW18, PD
RM19	RAW19, PD, FIFO CHARGE, LEAN QM
RM20	RAW20, PD
RM27	RAW27, PD, Verpackungsbox

Tabelle 4.74.: Logistische Prozesskette: RDS - Material (Einkauf)

Material	**Details**
RM28	RAW28, PD, Verpackungsfolie
RM30	RAW30, VB, FIFO Charge
FG29	- VERÄNDERT -
FG126	- VERÄNDERT -
FG226	- VERÄNDERT -
SG23	SEMI23, PD, Lohnbearbeitung
SG25	SEMI25, PD, Fremdbeschaffung
SG325A	SEMI325A, PLM, PD, Fremdbeschaffung
SG325B	SEMI325B, ECM, PD, Fremdbeschaffung
SG224	- VERÄNDERT -

Zu diesen genannten Materialien und Lieferanten bestehen jeweils Einkaufsinfosätze, sodass die Materialien zu allen Lieferanten zugeordnet werden können. Hierbei sind auch unterschiedliche Konditionen vorhanden.

Einkaufsorganisation Die Einkaufsorganisation ist für alle Einkaufsvorgänge im Unternehmen verantwortlich und handelt die Konditionen mit den Lieferanten aus. Sie ist somit der zentrale Punkt der Beschaffung. Im RDS sind die Buchungskreise 1010 und 1710 enthalten, für die jeweils auch Einkaufsorganisationen angelegt wurden. Diese sind als Vorlage nutzbar, im Zuge der Ausarbeitungen innerhalb der Projektgruppe wurde insbesondere die Einkaufsorganisation 1010 genutzt.

Einkäufergruppe Analog zu der Einkaufsorganisation existieren im RDS auch Einkäufergruppen (001, 002 und 003), die auch zu den Einkaufsorganisationen zugeordnet wurden. Für die Ausarbeitung der logistischen Prozesskette wurde die Einkäufergruppe 001 als Vorlage genutzt.

Delta

Im Folgenden werden die nötigen Anpassungen des Systems aufgezeigt, die über das RDS hinaus nötig waren, um die Anforderungen umzusetzen.

Lieferpläne Lieferpläne sind in den RDS-Dokumentationen nicht vorgesehen und es sind auch keine Lieferpläne im System hinterlegt. Dennoch sind Lieferpläne mit den RDS Lieferanten und Materialien nutzbar.

Freigabe von Bestellungen Laut Testskript (J45 - Beschaffung von Direktmaterialien) muss zur Bestellfreigabe einem Nutzer eine Planstelle für Bestellfreigabe-Workflow-Verwaltung zugeordnet werden. Wenn dies erfolgt ist, erhält der Benutzer jeweils ein Workflowitem wenn die Freigabe einer Bestellung notwendig ist. Die eigentliche Freigabe erfolgt dann über die Fiori-App „Mein Eingang“.

Im Zuge der Bearbeitung dieser Anforderung hat sich herausgestellt, dass die genannten Schritte nicht ausführbar sind, da keine Planstellen im System existieren. Die Ausführungen der RDS können somit nicht genutzt werden und eine Bestellfreigabe mit dem ausgelieferten RDS ist nicht möglich. Daher waren zusätzliche Customizingaktivitäten notwendig, die im folgenden erläutert werden.
Bei der Definition der Freigabestrategie im System wurde ein Leitfaden aus dem SAP Community Network[2] genutzt.
Im ersten Schritt mussten Merkmale angelegt werden, die die jeweiligen Felder in der Struktur CEKKO repräsentieren:
SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Merkmal bearbeiten
Hier wurden drei Merkmale mit den folgenden Attributen definiert:

Tabelle 4.75.: Logistische Prozesskette: Parameter - Freigabestrategie (BP_POTYPE)

Merkmal	**Tab**/**Group**/Feld	**Parameter**
BP_POTYPE (Art der Bestellung)	**Basisdaten**	
	Grunddaten	
	Bezeichnung	Bestellart
	Formatangaben	
	Datentyp	Zeichenformat
	Anzahl Stellen	4
	Bewertung	
	Einwertig	Aktiv
	Bezeichnungen	
	Sprache	DE
	Bezeichnung	Bestellart
	Werte	
	Zulässige Werte	
	Merkmalwert	NB
	Bezeichnung	Standard PO
	Zusatzdaten	
	Verweis auf Tabellenfeld	
	Tabellenname	CEKKO
	Feldname	BSART
	Verhalten bei Bewertung	
	Nicht eingabebereit	Aktiv

[2]http://scn.sap.com/docs/DOC-46564

Tabelle 4.75.: Logistische Prozesskette: Parameter - Freigabestrategie (BP_POTYPE)

Merkmal	**Tab**/**Group**/Feld	**Parameter**
	Einschränkungen	
	Gültige Klassenarten	
	Klassenart	032
	Klassenarttext	Freigabestrategie

Tabelle 4.76.: Logistische Prozesskette: Parameter - Freigabestrategie (BP_EKORG)

Merkmal	**Tab**/**Group**/Feld	**Parameter**
BP_EKORG (Einkaufsorganisation)	**Basisdaten**	
	Grunddaten	
	Bezeichnung	Bestellart
	Formatangaben	
	Datentyp	Zeichenformat
	Anzahl Stellen	4
	Bewertung	
	Einwertig	Aktiv
	Bezeichnungen	
	Sprache	DE
	Bezeichnung	Einkaufsorganisation
	Werte	
	Zulässige Werte	
	Merkmalwert	2020
	Bezeichnung	Standard EKORG
	Zusatzdaten	
	Verweis auf Tabellenfeld	
	Tabellenname	CEKKO
	Feldname	EKORG
	Verhalten bei Bewertung	
	Nicht eingabebereit	Aktiv
	Einschränkungen	
	Gültige Klassenarten	
	Klassenart	032
	Klassenarttext	Freigabestrategie

Tabelle 4.77.: Logistische Prozesskette: Freigabestrategie (BP_GESNETVAL)

Merkmal	**Tab/Group/Feld**	**Parameter**
BP_GESNETVAL (Gesamter Bestellnettowert)	**Basisdaten**	
	Grunddaten	
	Bezeichnung	Bestellart
	Formatangaben	
	Datentyp	Währungsformat
	Anzahl Stellen	15
	Dezimalstellen	2
	Währung	EUR
	Bewertung	
	Mehrwertig	Aktiv
	Intervallwerte erlaubt	Aktiv
	Bezeichnungen	
	Sprache	DE
	Bezeichnung	Gesamter Bestellnettowert
	Werte	
	Zulässige Werte	
	Merkmalwert	>=5000.00
	Zusatzdaten	
	Verweis auf Tabellenfeld	
	Tabellenname	CEKKO
	Feldname	GNETW
	Verhalten bei Bewertung	
	Nicht eingabebereit	Aktiv
	Behandlung der Benutzereingaben	
	Formatfreie Eingabe	Aktiv
	Einschränkungen	
	Gültige Klassenarten	
	Klassenart	032
	Klassenarttext	Freigabestrategie

Im nächsten Schritt muss eine Klasse angelegt für die Freigabestrategie angelegt und die zuvor definierten Merkmale dieser zugeordnet werden.
SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Klasse bearbeiten

Tabelle 4.78.: Logistische Prozesskette: Klassenparameter - Freigabestrategie

Klasse	**Tab**/**Group**/Feld	**Parameter**
BP_PORELEASE		
	Klassenart	32
	Basisdaten	
	Grunddaten	
	Bezeichnung	BP Purchase Order Release Class
	Status	1
	Gleiche Klassifizierung	
	Nicht prüfen	Aktiv
	Merkmale	
	Merkmal	BP_POTYPE
	Merkmal	BP_EKORG
	Merkmal	BP_GESNETVAL

Im letzten Schritt bei der Definition der Freigabestrategie muss das Freigabeverfahren für Bestellungen festgelegt werden. Hierbei müssen Freigabegruppen, -codes, -kennzeichen und -strategien sowie Einstellungen zum Workflow zugeordnet werden.
SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Freigabeverfahren für Bestellungen festlegen → Freigabegruppen

Tabelle 4.79.: Logistische Prozesskette: Parameter - Freigabegruppen

FrgGruppe	FrgObjekt	Klasse	Bezeichnung
PO	2	BP_PORELEASE	Strategy for PO

SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Freigabeverfahren für Bestellungen festlegen → Freigabecodes

Tabelle 4.80.: Logistische Prozesskette: Definition - Freigabecodes

Grp	Code	Workflow	Bezeichnung
PO	10	1	Manager

SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Freigabeverfahren für Bestellungen festlegen → Freigabekennzeichen

Tabelle 4.81.: Logistische Prozesskette: Definition - Freigabekennzeichen

Freigabekennz.	Freigeg.	Änderbar	Wertänd. in %	Bezeichnung
0	nicht aktiv		1	blocked
1	aktiv	6		Released

SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Freigabeverfahren für Bestellungen festlegen → Freigabestrategien

Tabelle 4.82.: Logistische Prozesskette: Definition - Freigabestrategie

Grp	Strategie	Bezeichnung
PO	P1	PO Release Strategy

SPRO → SAP Referenz-IMG → Materialwirtschaft → Einkauf → Bestellung → Freigabeverfahren für Bestellungen → Freigabeverfahren für Bestellungen festlegen → Workflow

Tabelle 4.83.: Logistische Prozesskette: Definition - Workflow (Freigabe)

Grp	Code	Bezeichnung	O.	Bearbeiterid
PO	10	Manager	US	UOREK

Nach dieser Freigabestrategie müssen alle Bestellungen mit einem höheren Volumen als 5000 Euro durch einen Einkaufsmanager (in diesem Fall beispielhaft der Nutzer UOREK) freigegeben werden. Die Werte sind hierbei alle beispielhaft gewählt und können nachträglich noch angepasst werden.

Auch nach diesen zusätzlichen Customizing ist die Freigabe von Bestellungen ausschließlich über die TA ME29N möglich und nicht über die Fiori-App.

Qualitätsmanagement In der ersten Projektphase wurde der Wareneingang zur Bestellung direkt in den freien Lagerbestand gebucht. Bei diesem Prozess sollte nun eine Qualitätssicherung für die vom Lieferanten angelieferten Waren erfolgen. Hierfür sollten alle eingehenden Waren zuerst in den Qualitätssicherungsbestand gebucht werden. Anschließend sollte eine Stichprobenkontrolle erfolgen, anhand derer die Qualität der gelieferten Waren beurteilt werden kann. Des Weiteren war vorgesehen, dass eine Qualitätssicherung in der Fertigung analog zu dem Qualitätsmanagementprozess im Wareneingang ablaufen soll.
Bei der Auseinandersetzung mit Qualitätsmanagementprozessen wurde schnell ersichtlich, dass das RDS hier keine Hilfestellungen bietet. Dies bedeutet, dass im System keinerlei Inhalte existieren, welche als Vorlage o.ä. genutzt werden konnten.
Aufgrund dessen wurde versucht ein Qualitätsmanagement durch die nachfolgenden Schritten umzusetzen. Probleme, welche die Umsetzung dieser Anforderung behindert haben, wurden hierbei ebenfalls festgehalten.

Stichprobenverfahren definieren In dem Customizingbereich der Prüfplanung wurden verschiedene Einstellungen getätigt, um eine Qualitätsprüfung umzusetzen. Die Prüfplanung definiert, wie eine Qualitätsprüfung konkret stattfinden soll.
Die Anforderungen besagen, dass eine Qualitätsprüfung per Stichprobenverfahren durchführbar sein sollte. Daher wurde im ersten Schritt eine entsprechende Stichprobenermittlung unter *SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsplanung → Grunddaten → Stichprobe → Stichprobenermittelung* mit folgenden Parametern definiert:

Tabelle 4.84.: Logistische Prozesskette: Definition - Stichprobenermittelung

Stichprobenart	Bezeichnung	Definition
SP1	Standard-Qualiätskontrolle fest	Feste Stichprobe

Zur erstellten Stichprobenermittelung wurde eine entsprechende Stichprobenregel unter *SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsplanung → Grunddaten → Stichprobe → Stichprobenermittelung → Regel zur Stichprobenermittelung* angelegt. Diese Regel definiert eine feste Stichprobengröße.
Abschließend wird ein Bewertungsmodus zum Stichprobenverfahren erstellt. Unter dem Pfad *SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsplanung → Grunddaten → Stichprobe → Bewertung definieren* wurde hierbei ein Bewertungsmodus mit folgenden Parametern angelegt:

Tabelle 4.85.: Logistische Prozesskette: Definition - Bewertungsmodus

Bewertungsmodus	Bezeichnung	Definition
SP	Standard-Qualitätskontrolle (Attribut)	Attributsprüfung nach Anzahl fehlerhafter Einheiten

Pflege des Nummernkreises Ein erstes Problem, welches sich bei der Implementierung des Qualitätsmanagementprozesses ergab, waren fehlenden Einträge in den Nummernkreistabellen. Die Nummernkreisdaten sind für die Prüflosherkunft relevant. Die Tabelle ist unter folgendem Pfad aufzurufen: *SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsprüfung → Prüfloseröffnung → Nummernkreis pflegen.*
Bei Aufruf der Tabelle wurde ersichtlich, dass keine Einträge vorhanden waren. Die Eintragung von neuen Daten über das SAP-Menu blieb hierbei erfolglos. Die Bearbeitung über die manuelle Tabellenpflege (Transaktion „SM30“) blieb ebenfalls ohne Erfolg, da diese nur funktioniert, wenn bereits Einträge vorhanden sind und es sich um eine Ergänzung dieser handelt.
Aufgrund der beschriebenen Problemstellungen mussten die Einträge auf ABAP-Ebene (über die Transaktion „SE16n“) manuell eingepflegt werden.

Prüfart pflegen Wie bereits erläutert, ist die Qualitätsprüfung sowohl für den Wareneingang als auch für die Fertigung vorgesehen. Dies bedeutet, dass jeweils zwei unterschiedliche Prüfarten vorhanden sein müssen. Die Prüfarten wurden unter *SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsprüfung → Prüfloseröffnung → Prüfart pflegen* angelegt.
Exemplarisch wurde hier vorerst mit einer Prüfart gearbeitet. Daher wurde die Prüfart zum Wareneingang wie folgt definiert:

Tabelle 4.86.: Logistische Prozesskette: Definition - Prüfart

Prüfart:	QMW
Bezeichnung:	Prüfung bei Wareneingang zur Bestellung
Probenart:	Wareneingangsprobe
Erfassungssicht:	Einzelwerte und summarische Ergebnisse
Planverwendung:	Wareingang (4)

Nach Definition dieser Prüfart, musste im nächsten Schritt die Zuordnung zwischen Prüfart und Material erfolgen. Dazu wurde versucht die Prüfart in den Materialstamm (Transaktion „MM02 - Sicht Qualitätsmangement - Prüfeinstellungen“) aufzunehmen. Hierbei gab das System eine Fehlermeldung aus, die besagte, dass keine Prüflosherkunft zur Prüfart gepflegt worden ist.
Die Zuordnung zwischen Prüfart und Prüflosherkunft erfolgt über *SPRO → SAP Referenz-*

IMG → Qualitätsmanagement → Qualitätsprüfung → Prüfloseröffnung → Prüflosherkunft pflegen und Prüfart zuordnen. Die Tabelle mit den Prüflosherkünften ist hierbei jedoch leer. Eine Pflege dieser Tabelle und somit die Definition einer neuen Prüflosherkunft, ist nicht vorgesehen. Aufgrund dieser Problemstellung war eine erfolgreiche Zuordnung von Prüfart und Material nicht umsetzbar.

Prüfplan Ein weiterer Versuch das Qualitätsmanagement umzusetzen war die Erstellung eines Prüfplans. Der Prüfplan sollte dann in einen Arbeitsplan integriert werden und hierdurch die Qualitätsprüfung in der Fertigung erfolgen.
Daher wurde unter *SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsplanung → Allgemein → Planverwendung definieren* eine Planverwendung für die Fertigung definiert.
Anschließend wurde der Plantyp unter **SPRO → SAP Referenz-IMG → Qualitätsmanagement → Qualitätsplanung → Allegmein → Plantyp der Materialart zuordnen** der Materialart zugeordnet.

Tabelle 4.87.: Logistische Prozesskette: QM - Plantyp der Materialart zuordnen

Plantyp	Bezeichnung	Materialart
Q	Prüfplan	Fertigerzeunis
Q	Prüfplan	Halbfabrikat
Q	Prüfplan	Material
Q	Prüfplan	Rohstoff

Die Zuordnung des Prüfplans zur Fertigung des Bremsblocks wurde nun über den Pfad **Qualitätsmanagement → Prüfplanung → Prüfplan → anlegen** durchgeführt. Jedoch blieb der Prüfplan nach der Zuordnung ohne jegliche Berücksichtigung im Arbeitsplan. Eine Vermutung hierbei ist, dass der verwendete Steuerschlüssel nicht passend ist.
In Anbetracht der Zeitplanung und nach Rücksprache mit Joachim Manherz, wurden an dieser Stelle alle weiteren Versuche ein Qualitätsmanagementprozess umzusetzen eingestellt. Es erfolgte eine Fokussierung auf die verbleibenden Anforderungen.

Gutschriftverfahren Einkauf Eine weitere Anforderungen stellt die Einführung des Gutschriftverfahrens dar. Bei dem Gutschriftverfahren geht es darum, dass die Abrechnungslast nicht beim Lieferanten liegt, sondern vom Kunden getragen wird. Für die Umsetzung das Gutschriftverfahren in SAP stellt das RDS keine Anwendungsfälle bereit. Aufgrund dessen, wurde das Gutschriftverfahren durch die nachfolgenden Schritte implementiert.

Lieferantenstamm Um das Lieferantengutschriftsverfahren umzusetzen muss der Lieferant als ERS-fähig gelten. Hierfür bedarf es einer Bearbeitung des Lieferantenstammsatzes über die Transaktion „XK02“. Im Lieferantenstammsatz wird die Sicht

„Einkaufsdaten“ gewählt und der Parameter „Automatische Wareneingangsabrechnung“ aktiv gesetzt. Des Weiteren wird in den Einkaufsdaten der Parameter „Wareneingangsbezogene Rechnungsprüfung“ aktiv gesetzt. Nach diesen Einstellung gilt der Lieferant als ERS-fähig.

Nachrichtenart pflegen Um den Rechnungsbeleg nach einem Wareneingang generieren zu können muss die Nachrichtensteuerung der Logistischen-Rechnungsprüfung entsprechend konfiguriert werden. Dies geschieht über *SPRO → SAP Referenz-IMG → Materialwirtschaft → Logistischen-Rechnungsprüfung → Nachrichtenfindung → Nachrichtenarten pflegen*. Hierbei wurde eine neue Nachrichtenart mit folgenden Parametern angelegt:

Tabelle 4.88.: Logistische Prozesskette: Gutschriftverfahren - Nachrichtenart pflegen

Nachrichtenart	Bezeichnung	Sprache
Z001	Test	DE

Im nächsten Schritt wird der Nachrichtenart über die Dialogstruktur folgende Verarbeitungsroutinen zugewiesen:

Tabelle 4.89.: Logistische Prozesskette: Gutschriftverfahren - Verarbeitungsroutinen

Medium	Programm	Form-Routine	Formular	Art
Druckausgabe	RM08NAST	Entry-ERS	MR-Print	PDF
Einfaches Mail	RSNASTS0	SAPOFFICE-Aufruf	MR-Print	PDF

Als abschließenden Schritt wird der Nachrichtenart entsprechende Partnerrollen zugewiesen. Dies erfolgt erneut über die Dialogstruktur unter dem Punkt „Partnerrollen“. Die Partnerrollen für die Nachrichtenart „Z001“ wurden wie folgt gewählt:

Tabelle 4.90.: Logistische Prozesskette: Gutschriftverfahren - Partnerrollen

Medium	Rolle	Bezeichnung
Druckausgabe	LF	Lieferant
Druckausgabe	RS	Rechnungsteller
Einfaches Mail	LF	Lieferant
Einfaches Mail	RS	Rechnungsteller

Nachrichtenschemata pflegen Als nächsten muss die neu erstellte Nachrichtenart in einem neuen Nachrichtenschemata integriert werden. Dies erfolgt über *SPRO → SAP Referenz-IMG → Materialwirtschaft → Logistischen-Rechnungsprüfung → Nachrichtenfindung → Nachrichtenschemata pflegen*. Unter diesem Pfad wird ein neues Nachrichtenschema mit der Bezeichnung „MR0004“ angelegt (**Der Name sollte unbedingt „MR0004“ sein, da nur dieser für die Logistische-Rechnungsprüfung greift**).

Dem neu angelegten Nachrichtenschemata werden unter dem Punkt Steuerung in der Dialogstruktur folgende Parameter zugewiesen:

Tabelle 4.91.: Logistische Prozesskette: Gutschriftverfahren - Nachrichtenschemata

Stufe	Zähler	KArt	Bezeichnung
10	10	Z001	Test

Konditionen pflegen Es ist erforderlich einige Konditionen für die neu angelegte Nachrichtenart zu pflegen. Dies geschieht über *SPRO → SAP Referenz-IMG → Materialwirtschaft → Logistischen-Rechnungsprüfung → Nachrichtenfindung → Konditionen pflegen → Konditionen anlegen*. Im Dialogfenster wird die Nachrichtenart „Z001“ ausgewählt. Im nachfolgenden Dialog werden folgende Parameter gewählt:

Tabelle 4.92.: Logistische Prozesskette: Gutschriftverfahren - Konditionen

Lieferant	Bezeichnung	Rolle	Medium	Zeitpunkt	Sprache
1000021	Autom. Adam	LF	Druckausgabe	3	DE

Nachrichten: Ausgabeprogramme Im abschließenden Schritt werden der definierten Nachrichtenart SAPScriptformulare und Programme zugeordnet. Dies geschieht über *SPRO → SAP Referenz-IMG → Materialwirtschaft → Logistischen-Rechnungsprüfung → Nachrichtenfindung → Formulare und Programme zuordnen.*
In dieser Tabelle werden zwei Einträge mit folgenden Parametern ergänzt:

Tabelle 4.93.: Logistische Prozesskette: Gutschriftverfahren - Ausgabeprogramm

NArt	Bezeichnung	Medium	Programm	Routine	Formular
Z001	Test	Druckausgabe	RM08NAST	Entry-ERS	MR-Print
Z001	Test	Einfaches Mail	RSNASTS0	SAPOFFICE-Aufruf	MR-Print

Nach diesen Einstellungen sollten alle Voraussetzungen für das Gutschriftverfahren auf Lieferantenseite erfüllt sein. Dies bedeutet, dass nun über die Transaktion „MRRL“ Wareneingänge per Gutschriftanzeige abgerechnet werden können.

4.4.4. Produktion

Die Produktion hat das Ziel der Fertigung einer Bremsanlage. Hierzu wird mithilfe des MRP-Laufes der Materialbestand im Lager evaluiert. Falls die Materialversorgung durch den Lagerbestand nicht abgedeckt werden kann, wird eine Bestellanforderung generiert und an den Einkauf übermittelt. Im nächsten Schritt wird ein Fertigungsauftrag angelegt, woraufhin die entsprechenden Halbfabrikate aus dem Lager in die Produktion umgelagert werden. Die Fertigung wird hierbei als zusammenhängender Fertigungsprozess durchgeführt. Nach erfolgreicher Fertigstellung erfolgt die Rückmeldung des Fertigungs-

prozesses und das Fertigerzeugnis wird eingelagert.

RDS-Stammdaten

Material Für die Umsetzung der Produktion das Fertigerzeugnis Bremsanlage werden folgende Materialien benötigt:

- Fertigerzeugnis: Bremsanlage
- Halbfabrikat: Bremsblock
- Rohstoff: Festsattel, Bremsbeläge, Bremsscheibe

Wie die nachfolgende Tabelle zeigt, bietet das RDS vorkonfigurierte Materialien, die entsprechend genutzt werden konnten:

Tabelle 4.94.: Logistische Prozesskette: RDS - Material

Bezeichnung	Details
FG126	Fertigerzeugnis MTS
SG21	Halbfabrikat für Serienfertigung
SG124	Halbfabrikat Unterbaugruppe
SG25	Halbfabrikat ext. Beschaffung
SG22	Halbfabrikat Dummy
RM16	Rohstoff ext. Beschaffung
RM17	Rohstoff ext. Beschaffung
RM18	Rohstoff ext. Beschaffung
RM20	Rohstoff ext. Beschaffungsvereinbarung
RM27	Rohstoff-Versandkarton
RM120	Rohstoff ext. Beschaffung mit QM bei Beschaffung
RM122	Rohstoff ext. Beschaffung, chargenpflichtig (FIFO-Strategie)
RM124	Rohstoff ext. Beschaffung, verbrauchsgesteuert
RM128	Rohstoff ext. Beschaffung mit Konsignation

Stücklisten Das RDS bietet eine vorkonfigurierte Stückliste zum Fertigerzeugnis „FG126“. Die Stücklistenstruktur umfasst 3 Ebenen und insgesamt 16 Materialkomponenten. Somit kann die RDS-Stückliste eine grundsätzliche Stücklistenstruktur vermitteln und als Vorlage dienen.

Arbeitspläne Für die Fertigung des Fertigerzeugnisses „Bremsanlage“ wird ein Arbeitsplan benötigt. Der Arbeitsplan beschreibt alle Vorgänge die während des Produktionsprozesses ablaufen. Des Weiteren definiert dieser an welchem Arbeitsplatz die Vorgänge durchgeführt werden.
Das RDS bietet bereits einen vordefinierten Arbeitsplan, welcher für die Fertigung des

Fertigerzeugnisses „FG126“ verwendet werden kann. Somit kann dieser Arbeitsplan entweder als Vorlage genutzt werden oder unter der Voraussetzung der Anpassung direkt genutzt werden.

Arbeitsplatz Das RDS bringt eine Reihe von vorkonfigurierten Arbeitsplätzen mit, welche definieren wo die Umsetzung der Fertigungsschritte erfolgt. Die nachfolgende Tabelle zeigt alle Arbeitsplätze die durch das RDS abgebildet werden:

Tabelle 4.95.: Logistische Prozesskette: RDS - Arbeitsplatz

Art	Arbeitsplatz	Kurzbeschreibung
Maschine	Assembly	Baugruppe
Maschine	Technic	CNC-Maschine
Fertigungslinie	Winding	Federwicklung
Prozesseinheit	Bottling	Tintenabfüllung
Prozesseinheit	Mixing	Tintenmischung
Prozesseinheit	Packing	Tintenverpackung

Werk Das Werk gilt als Betriebsstätte und ist somit für die Durchführung des Produktionsprozesses erforderlich. Das RDS bietet hierzu zwei vorkonfigurierte Werke. Bei der Umsetzung der logistischen Prozesskette wurde sich überwiegend an „Werk 1010“ orientiert.

Tabelle 4.96.: Logistische Prozesskette: RDS - Werk

Werk	Name
1010	Plant 1 DE
1070	Plant 1 US

Delta

Im Folgenden werden die nötigen Anpassungen des Systems aufgezeigt, die über das RDS hinaus nötig waren, um die Anforderungen umzusetzen.

Automatische Freigabe von Fertigungsaufträgen Der im Testskript Lagerfertigung (BJ5) beschriebene Prozessschritt „Fertigungsaufträge freigeben“, verlangt eine manuelle Freigabe der Fertigungsaufträge. Gemäß den vorab definierten Anforderungen sollen die Fertigungsaufträge automatisiert freigegeben werden. Da die automatische Freigabe in dem durch das RDS vorkonfigurierte Fertigungssteuerungsprofil nicht vorgesehen ist, muss für die Umsetzung der automatisierten Freigabe ein neues Fertigungssteuerungsprofil angelegt werden.
Das Anlegen erfolgte über *SPRO → SAP Referenz-IMG → Produktion → Fertigungssteuerung → Stammdaten → Fertigungssteuerungsprofil definieren* .
Das Fertigungssteuerungsprofil wurden mit folgenden Parametern angelegt:

Tabelle 4.97.: Logistische Prozesskette: Parameter - Fertigungssteuerung

Bezeichnung	Parameter
Steuerungsprofil	FSP1
Bei Eröffnung: Freigabe durchführen	aktiv
Bei Freigabe: Auftrag terminieren	aktiv
Materialverfügbarkeitsprüfung: Teilmengen bestätigen	aktiv
Wareneingang: Auto. Wareneingang	aktiv
Auftragsart	Lagerfertigung

Abschließend muss das Fertigungssteuerungsprofil den Materialien zugewiesen werden. Hierzu wird in der Sicht Arbeitsvorbereitung im Feld Fertigungssteuerungsprofil der Parameter „FSP1“ für die Materialien „Bremsblock“ und „Bremsanlage“ ergänzt.

Kalkulationsschema Um die Produktionskosten von RDS-Erzeugnissen zu ermitteln, kann die Transaktion „Ck11N“ genutzt werden. Für alle im RDS definierten Materialien funktioniert diese Transaktion. Dies bedeutet, dass bei der Nutzung von RDS-Materialien, eine Produktionskostenkalkulation bereits funktionsfähig ist. Lediglich kleinere Einstellungen müssen hierbei beachtet werden. So muss beispielsweise sichergestellt werden, dass abweichende Preise korrekt im Materialstamm angepasst sind. Ebenfalls müssen Maschinen-, Personalzeiten etc. entsprechend im Arbeitsplan angegeben werden, sodass diese in der Produktionskostenkalulation berücksichtigt werden.

4.4.5. Vertrieb

Im Vertrieb werden die produzierten Bremsanlagen an Kunden verkauft. Dabei trifft zuerst der Kundenauftrag ein und wird mit den entsprechenden Daten im System angelegt. Darauf folgend wird die, zum Kundenauftrag gehörige, Auslieferung im System angelegt. Entsprechend der benötigten Anzahl an Bremsanlagen werden diese kommissioniert und anschließend im Lager verpackt. Wenn somit die Lieferung vorbereitet wurde, werden die Waren ausgebucht, automatisch eine Avis durchgeführt und die Lieferung losgeschickt. Zuletzt wird die Fakturierung des Auftrags vorgenommen.

RDS-Stammdaten

Im RDS sind vorkonfigurierte Organisationseinheiten und Stammdaten für den Verkauf von Waren enthalten. So können die Verkaufsorganisation, Verkaufsgruppe sowie Vertriebsweg und Sparte als Vorlage genutzt werden. Neben den vorkonfigurierten Materialien (siehe 4.4.3), die verkauft werden können, sind auch Beispielkunden (sowohl national als auch international) im System hinterlegt, die als Vorlage genutzt werden können:

Tabelle 4.98.: Logistische Prozesskette: Beispielkunden

Debitor	**Name**
10100273	Inlandskunde DE CPD
17100273	Domestic US CPD
17100001	Domestic US Customer 1
17100002	Domestic US Customer 2
17100003	Domestic US Customer 3
17100004	Domestic US Customer 4
17100005	Domestic US Customer 5
17100006	Domestic US Customer 6
10100001	Inlandskunde DE 1
10100002	Inlandskunde DE 2
10100003	Inlandskunde DE 3
10100004	Inlandskunde DE 4
10100005	Inlandskunde DE 5
10100006	Inlandskunde DE 6

Neben den Materialien und Kunden sind auch verschiedene Belegtypen gemäß den Testskripten vordefiniert und können genutzt werden.

Delta

Im Folgenden werden die nötigen Anpassungen des Systems aufgezeigt, die über das RDS hinaus nötig waren, um die Anforderungen umzusetzen.

Ware verpacken Im RDS sind zwar Verpackungsmaterialien (RM27 und RM28) vorhanden, allerdings ist das Verpacken von verkauften Waren kein Bestandteil eines RDS-Prozesses. Daher war es hier nötig, eigene Packregeln und Verpackungsmaterialien zu definieren.Dies wurde mit der TA MM01 durchgeführt.
Hierbei wurden folgende Sichten gepflegt:

- Grunddaten 1
- Grunddaten 2
- Vertrieb: VerkaufsorgDaten 1
- Vertrieb: VerkaufsorgDaten 2
- Vertrieb: allg./Werksdaten

Als Materialart wurde die Materialart VERP (Verpackungsmaterial gewählt). Bei den Organisationseben wurde das Werk 2020, die Verkaufsorganisation 2020 sowie der Vertriebsweg 10 gewählt. In den Sichten wurden folgende Parameter gepflegt:

Tabelle 4.99.: Logistische Prozesskette: Parameter - Karton klein

Sicht	Feld	Parameter
Grunddaten 1	Bruttogewicht Gewichtseinheit	0.5kg KG
Vertrieb: allg./Werk	Packmittelart Zul. VerpGewicht	30 100kg

Im Anschluss daran musste eine Strategie implementiert werden, die die Packmittel zu dem jeweiligen Material zuordnet. Dies geschah folgendermaßen:

Packmittelart definieren Im ersten Schritt wurden eine neue Packmittelart definiert.
SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Verpacken → Packmittelarten definieren

Tabelle 4.100.: Logistische Prozesskette: Parameter - Packmittelart

Packmittelart	Bezeichnung
0030	Karton (klein)

Materialgruppe für Packmittel definieren Nach der Definition der Packmittelarten wurde eine Materialgruppe für Packmittel definiert. *SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Verpacken → Materialgruppe Packmittel definieren*

Tabelle 4.101.: Logistische Prozesskette: Parameter - Materialgruppe für Packmittel

GrPM	Bezeichnung
PG30	Verpackungsgruppe 30

Logistische Prozesskette: Zuordnung - Packmittelgruppe und Packmittelart Anschließend mussten der Packmittelart die erlaubten Packmittel zugeordnet werden.
SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Verpacken → Packmittelarten definieren

Tabelle 4.102.: Logistische Prozesskette: Zuordnung - Packmittelgruppe und Packmittelart

Matgr. PM	Bezeichnung	PMArt	Bezeichnung
PG30	Verpackungsgruppe 30	0030	Karton (klein)

Packvorschrift anlegen Um beim Verpackprozess Packvorschläge zu erhalten, ist die Definition einer Packvorschrift nötig (TA POP1).
In dieser Transaktion wurde die Packvorschrift ICH41_KARTON_KLEIN angelegt, welche dazu dient Bremsanlagen in kleine Kartons zu verpacken.
Im Reiter Komponenten wurden folgende Parameter eingegeben:

Tabelle 4.103.: Logistische Prozesskette: Parameter - Packvorschrift

Positionstyp	Komponente	Soll-Menge	Minimale Menge
P	Karton klein	1	
M	Bremsanlage	4	1

Materialgruppe PM im Material Bremsanlage zuordnen Über die TA MM02 wurde das Material Bremsanlage um die Materialgruppe PM ergänzt. Hier wurde in der Sicht Grunddaten 1 im entsprechenden Feld der Wert PG30 ergänzt.

Findungssatz für Packvorschrift anlegen Um einen Findungssatz für die Findung von Packvorschriften anzulegen wurde die TA POF1 genutzt. Hier wurde der Findungssatz Z001 angelegt. Die Schlüsselkombination erfolgt nach dem Material. In dem Findungssatz wurden folgende Daten definiert:

Tabelle 4.104.: Logistische Prozesskette: Parameter - Findungssatz für Packvorschrift

Material	Bezeichnung	Packvorschrift
Bremsanlage	Bremsanlage	ICH41_KARTON_KLEIN

Zuordnung: Konditionsart und Zugriffsfolge Damit die Packvorschrift je nach Material automatisch gefunden werden kann, war eine Zuordnung der Konditionsart zur Zugriffsfolge notwendig. Dazu wurde mit der TA OFP3 folgende Zuordnung realisiert:

Tabelle 4.105.: Logistische Prozesskette: Zuordnung - Konditionsart und Zugriffsfolge

KArt	Bezeichnung	ZuFg	Bezeichnung
Z001	Find. Packvorschrift	SHIP	Findung Versand-Packvorschrift

Lieferpläne Neben der Nutzung von Lieferplänen im MM-Bereich sollte auch der Verkauf von produzierten Produkten über Lieferpläne möglich sein. Ebenso wie im MM-Bereich beinhaltet das RDS diesen Anwendungsfall im SD-Bereich nicht. Es gibt auch in SAP Fiori keine App zu diesem Anwendungsfall, sodass die Nutzung der Lieferpläne im SD-Bereich ausschließlich über die SAP GUI bzw über die TA erfolgen kann. Zwar gibt es eine vorkonfigurierte Verkaufsbelegart „DL - Auftragsart LiefPlan“ (siehe VOV8), allerdings ist diese im System gesperrt und auch wenn diese Sperrung aufgehoben wird, dann wird diese Belegart nur in der VA01 (Kundenauftrag anlegen) angezeigt und nicht in der VA31 (Lieferplan anlegen). Der Grund dafür ist, dass der Vertriebsbelegtyp der Belegart DL „C“ ist und nicht wie notwendig „E“. Dies hat zur Folge, dass zur Nutzung von Lieferplänen im SD-Bereich ein eigene Verkaufsbelegart angelegt werden musste. Dies wurde wie folgt getan:

TA VOV8 oder SPRO → SAP Referenz-IMG → Vertrieb → Verkauf → Verkaufsbelege → Verkaufsbelegarten definieren
Folgende Parameter wurden verwendet:

Tabelle 4.106.: Logistische Prozesskette: Verkaufsbelegart ZLP

Verkaufsbelegart	ZLP - Lieferplan SD
Vertriebsbelegtyp	E
Nummernsysteme	
Nummernkr.int.Verg.	01
Nummernkr.ext.Verg.	02
Inkrement Pos.Nummer	10
Inkrement Upos.Num.	10
Allgemeine Steuerung	
Wahrscheinlichkeit	0
Kreditlimit prüfen	D
Kreditgruppe	01
Nachrichtenappl.	V1
Sparte Position	aktiviert
Infosatz lesen	aktiviert
Transaktionsablauf	
Gr. Bildfolge	LP
Gr. Transakt.Vorgang	3
Belegschema	A
Anzeigeumfang	UALL
FCODE Übersichtsbild	UER1

Versand	
Lieferart	LF
Faktura	
Lieferbez. Fakt.Art	CI01
WunschliefDat/PreisDat/BestDat	
LiefDat vorschlagen	aktiviert

Nach dem Anlegen der Verkaufsbelegart müssen diesem Verkaufsbeleg Positiontypen zugeordnet werden, damit die Lieferpläne auch Positionen enthalten können. Dies erfolgte über *SPRO → SAP Referenz-IMG → Vertrieb → Verkauf → Verkaufsbelege → Verkaufsbelegpositionen → Positionstypen zuordnen*. In dieser Tabelle wurden zwei Einträge ergänzt:

Tabelle 4.107.: Logistische Prozesskette: Zuordnung - Positionstypen zu Verkaufsbelegart

VArt	MTPos	Verw.	PsTyÜPos	PsTyD	PsTyM	PsTyM	PsTyM
SO02	NORM			TAN		TAQ	TANN
SO02	NORM	FREE	TAN	TANN			

Um einen Kunden im Lieferplan angeben zu können muss zudem die Partnerfindung für diese Belegart konfiguriert werden. Dies geschieht über *SPRO → SAP Referenz-IMG → Vertrieb → Grundfunktionen → Partnerfindung → Partnerfindung einstellen → Partnerfindung für die Verkaufsbelegposition* . Dort wurde nun das Partnerschemata das Partnerschema SO02 der Verkaufsbelegart ZLP zugeordnet.

In einem letzten Schritt musste nun noch die Kopiersteuerung für Lieferungen festgelegt werden, damit die Lieferpläne auch tatsächlich beliefert werden können. Hierzu wurden unter *SPRO → SAP Referenz-IMG → Logistics Execution → Versand → Kopiersteuerung → Kopiersteuerung für Lieferungen festlegen* folgende Einträge ergänzt:

Tabelle 4.108.: Logistische Prozesskette: Kopiersteuerung für Lieferpläne

Ziel	Lieferart	Quelle	VerBelegart
LF	Lieferung	ZLP	Lieferplan SD

Die Detaildaten des Eintrags wurden wie folgt konfiguriert:

- Bedingungen
  - Bedingung an Auftrag: 001
  - Bed. zur Zusammenfg.: 051
- Datenübernahme

- Kopfdaten: 001
- Kopfdaten Fremdsyst.: 0
- Handling Units: 000

Diesem Eintrag wurden danach die Positionstypen „IM01“, „TAN“ sowie „TANN“ zugeordnet.

Kundenbonus Eine Anforderung des zweiten Prototypen war die Einrichtung von Kundenboni. Hierbei sollte es möglich sein, dass Kunden, die ein gewisses Bestellvolumen über einen Zeitraum (z.B. Jahr) erreicht haben, ein Bonus für die nächste Bestellung gewährt werden kann. Dieser Bonus könnte beispielsweise die Ausprägung einer Vergünstigung der nächsten Bestellung aufweisen.
Dieser Anwendungsfall ist im RDS nicht vorgesehen. Daher wurde im zweiten Prototypen versucht, eine Bonusabsprache über *SAP Menü* → *Logistik* → *Vertrieb* → *Stammdaten* → *Absprachen* → *Bonusabsprache* → *Anlegen* anzulegen. Hierbei stößt man allerdings auf Fehler im System[3].
Daher wurde die Anforderung auf das Anlegen von Naturalrabatten modifiziert. Im RDS sind bereits verschiedene Naturalrabatte eingepflegt, sodass z.B. bei der Verwendung des Materials TG11 in einem Terminauftrag eine Naturalrabattlogik hinterlegt ist[4]. Diese vorkonfigurierten Inhalte könnten somit als Vorlage genutzt werden.

Gutschriftverfahren (SD) Ebenfalls sollte das Gutschriftverfahren für die SD-Seite implementiert werden. Hierbei ist es notwendig, dass die Faktura mit dem Kennzeichen Gutschrift zu versehen, sodass das System auf eine eingehende Gutschriftanzeige von Kunden wartet und diese entsprechend verrechnet werden kann.
Das implementierte RDS bietet, wie bereits auch bei dem Gutschriftverfahren auf Lieferantenseite, keinen spezifischen Anwendungsfall, der hierzu genutzt werden kann. Lediglich der RDS-Prozess „Abwicklung von Gutschriften (BLK)“ zeigt, wie einem Kunden, im Fall einer Retoure, die ausstehende Summe als Gutschrift angerechnet werden kann. In den von uns erhoben Anwendungsfällen erwarten wir jedoch vom Kunden eine „Zahlung“ per Gutschrift. Somit unterschiedet sich dieser Anwendungsfall von RDS-Prozess. Um in der Faktura das Kennzeichen „Gutschrift“ zu setzten sind mehrere Aktivitäten im Customizing notwendig. Im Fall der umgesetzten Prozesskette muss hierzu die Kopiersteuerung für das Anlegen der Faktura definiert werden. Dies geschieht über den Pfad: *SPRO* → *SAP Referenz-IMG* → *Vertrieb* → *Fakturierung* → *Fakturen* → *Kopiersteuerung für Fakturen pflegen* → *Kopiersteuerung: Lieferbeleg nach Faktura*. Dort kann nun die Fakturaart der Lieferart zugeordnet werden. In diesem Fall bedeutet das:

[3]siehe auch: Simplification List for SAP S/4HANA, on-premise edition 1511 (S.49)
[4]siehe Testskript (Naturalrabattabwicklung (BKA)

Tabelle 4.109.: Logistische Prozesskette: Zuordnung - Fakturaart und Lieferart

Ziel	Fakturaart	Quelle	Lieferart
G2	Gutschrift	LF	Lieferung
G2	Gutschrift	OD01	Auslieferung

Im Anschluss daran müssen beiden Einträgen noch Positionstypen zugeordnet werden:

Tabelle 4.110.: Logistische Prozesskette: Zuordnung - Positionstypen

Positionstyp	Bezeichnung
IM01	Bestandsposition
TAN	Normalposition
TANN	Kostenlose Position

Streckengeschäft Die Abwicklung von Verkäufen über das Streckengeschäft war eine Anforderung im zweiten Prototyp. Hierbei bot das RDS-Paket umfangreiche Unterstützung. So ist neben dem implementierten Prozess auch schon ein vorkonfiguriertes Streckenmaterial vorhanden. Dieses konnte als Vorlage genutzt werden. Zusätzlich mussten im Materialstamm entsprechende Parameter gesetzt werden.

Bearbeiten Materialstamm Die zusetzenden Parameter kennzeichnen das Material als Streckenmaterial. Für diese Umsetzung musste im Materialstamm - Sicht Grunddaten, die allgemeine Positionstypengruppe „CBNA“ ausgewählt werden. Ebenfalls wird diese Positionstypengruppe im Materiallstamm - Sicht Vertrieb2 hinterlegt. Nach diesen Einstellungen sollten alle im Materialstamm erforderlichen Voraussetzungen getroffen sein um das Material über das Streckengeschäft zu vertreiben.

4.5. Abnahme

Die Abnahme des Szenarios wurde am 24.01.2017 in Bremen mit Joachim Mannherz durchgeführt. Zur Abnahme wurden dabei alle Anforderungen geprüft und mit „abgenommen“ oder nicht „nicht abgenommen“ bewertet.

Tabelle 4.111.: Logistische Prozesskette: Abnahme - Einkauf

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
1	**Einkauf**		
1.1	Bestellung manuell anlegen	umgesetzt	abgenommen
1.2	BAnf manuell anlegen	umgesetzt	abgenommen

Tabelle 4.111.: Logistische Prozesskette: Abnahme - Einkauf

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
1	**Einkauf**		
1.3	BAnf in Bestellung umwandeln	umgesetzt	abgenommen
1.4	Bestellung genehmigen	umgesetzt	abgenommen
1.5	Wareneingang (zu Bestellung) buchen	umgesetzt	abgenommen
1.6	Lieferantenrechnung (zu Bestellung) erfassen	umgesetzt	abgenommen
1.7	Lieferpläne	umgesetzt	abgenommen
1.8	Geplanter Wareneingang	umgesetzt	abgenommen

Tabelle 4.112.: Logistische Prozesskette: Abnahme - Produktion

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
2	**Produktion**		
2.1	MRP-Lauf durchführen	umgesetzt	abgenommen
2.2	Materialdeckung auswerten	umgesetzt	abgenommen
2.3	Fertigungsauftrag anlegen	umgesetzt	abgenommen
2.4	Fertigungsauftrag rückmelden	umgesetzt	abgenommen
2.5	Produktionskostenkalkulierung	umgesetzt	abgenommen
2.6	Kundeneinzelfertigung	nicht umgesetzt	-

Tabelle 4.113.: Logistische Prozesskette: Abnahme - Vertrieb

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
3	**Vertrieb**		
3.1	Kundenauftrag anlegen	umgesetzt	abgenommen
3.2	Auslieferung anlegen	umgesetzt	abgenommen
3.3	Kommissionierung durchführen	umgesetzt	abgenommen
3.4	Ware verpacken	umgesetzt	abgenommen
3.5	Avis durchführen	nicht umgesetzt	-
3.6	Faktura anlegen	umgesetzt	abgenommen
3.7	Konditionsliste anlegen	umgesetzt	abgenommen
3.8	Kundenbonus	umgesetzt	abgenommen
3.9	Gutschriftverfahren	umgesetzt	abgenommen
3.10	Lieferpläne (SD)	umgesetzt	abgenommen

Tabelle 4.114.: Logistische Prozesskette: Abnahme - Lager

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
4	**Lager**		
4.1	Wareneingang einlagern	umgesetzt	abgenommen
4.2	Ware umbuchen	umgesetzt	abgenommen
4.3	Fertigerzeugnis einlagern	umgesetzt	abgenommen
4.4	Fertigerzeugnis auslagern	umgesetzt	abgenommen

Tabelle 4.115.: Logistische Prozesskette: Abnahme - Qualitätsmanagement

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
5	**Qualitätsmanagement**		
5.1	Wareneingang	nicht umgesetzt	-
5.2	Fertigung	nicht umgesetzt	-

4.6. Fazit

Im Rahmen des Szenarios wurde eine logistische Prozesskette prototypisch auf einen SAP S/4HANA System implementiert. Durch die Umsetzung der verbundenen Anforderungen, mussten die Studenten sich mit betriebswirtschaftlichen Abläufen auseinandersetzen. Konkret bedeutet dies, dass sie sich grundsätzliches Wissen über logistische Prozessabläufe aneignen mussten. Neben diesem betrieblichen Wissen musste eine Einarbeitung in die technischen Umgebung erfolgen. Hierbei wurden erweiterte Kenntnisse im Customzing von SAP Systemen gesammelt und angewandt.
Zusammenfassend kann daher festgestellt werden, dass das Szenario - Logistische Prozesskette einen sehr praxisnahen Anwendungsfall widergespiegelt hat. Dies hatte zur Folge, dass den Studierenden nicht nur betriebswirtschaftliche und technische Aspekte näher gebracht worden sind sondern auch grundsätzliche Projektmanagementmethodik ausgearbeitet und angewandt werden musste. Aufgrund dessen ist abschließend festzuhalten, dass das Szenario einen großen Mehrwert für die Studierenden dargestellt hat.

4.7. Ausblick

In der ersten Projektphase wurden grundlegende, standardisierte Prozesse im SAP System implementiert. Aufbauend auf diesen Prozessen wurden in der zweiten Phase weitere speziellere Prozesse eingebunden bzw. die bereits implementierten Prozessabläufe erweitert.
Trotz dieser Erweiterungen gibt es zahlreiche Ansatzpunkte um weitere Prozesse zu ergänzen und damit die Funktionalität des Prototypen entsprechend zu erweitern. Ein möglicher Schritt wäre hierbei die Integration von kundenspezifischen Einzelfertigungen oder die Just-In-Time Lagerverwaltung.
Ein weitere möglicher nächster Schritt, wäre ein Vergleich mit den neuen 16/10 Systemrelease. Hierbei könnte beispielsweise der Umfang von neuen RDS-Inhalten evaluiert

werden oder eine Analyse über neue Funktionsmöglichkeiten der Fiori 2.0 Oberfläche durchgeführt werden.

5. Szenario - Sensorik

5.1. Beschreibung

Diese Dokumentation ist im Rahmen der Masterprojektgruppe „Demosystem on HANA“ der Universität Oldenburg entstanden. Das vorliegende Dokument beschreibt die Dokumentation des „3. Szenarios - Sensorik“. Das Anwendungsszenario befasst sich mit der Verarbeitung von Sensordaten. Diese Daten werden von unterschiedlichen Sensoren zur Verfügung gestellt, welche z.B. an einem Raspberry Pi angeschlossen sind oder durch Drucker bereitgestellt werden. Das Ziel ist es, die Informationen der Sensoren in eine SAP HANA-Datenbank zu überführen, aufzubereiten und in einer SAPUI5-Anwendung grafisch darzustellen.
Diese Dokumentation beschreibt zu Beginn kurz die zu Grunde liegenden Anforderungen des Szenarios. Anschließend wird die genaue Umsetzung dieser Anforderungen detailliert beschrieben. Es werden die erstellte HANA-Applikation, die UI5-Applikation sowie die Java-Applikation der Raspberry Pis genauer erläutert. Die Beschreibung der Implementierungen wird hierbei an repräsentativen Codebeispielen vorgenommen. Der gesamte Code ist auf der beigelegten CD zu finden. Abschließend werden die durchgeführten Abnahmetests inklusive ihrer Ergebnisse dargestellt.

5.2. Anforderungen

Die aufgenommenen Anforderungen wurden in insgesamt sechs Kategorien eingeteilt. Diese Kategorien bilden jeweils einzelne Anforderungsbereiche, welche logisch voneinander getrennt wurden. Diese Kategorisierung sorgt für mehr Übersichtlichkeit und vereinfacht gleichzeitig die Aufgabenplanung und das Zeitmanagement.

5.2.1. Bereitstellung der Daten

Für die Bereitstellung der Daten wurden die Anforderungen für Drucker und Raspberry Pi unterschieden (siehe tabelle 5.1 auf der nächsten Seite). Die Drucker können ihre Daten als XML-Datei bereitstellen und benötigen daher weniger Anforderungen und Implementierungen. Der Raspberry Pi muss die Daten der Sensoren erst Abfragen und anschließend der Datenbank bereitstellen.

Tabelle 5.1.: Sensorik: Anforderungen - Bereitstellung der Daten

Nr.	**Anforderung:**
1	**Bereitstellung der Daten**
1.1	Messdaten müssen in der Datenbank aufbereitet werden.
Raspberry Pi	
1.2	Messdaten müssen abgerufen werden können.
1.3	Messdaten müssen zyklisch an die Datenbank gesendet werden (Push-Verfahren).
1.4	Ausfallsicherheit: Nach einem Neustart muss die Datenbereitstellung wieder automatisch aufgenommen werden.
1.5	Fehlermeldungen müssen an die Datenbank gesendet werden.
Drucker	
1.6	Die Messdaten müssen im XML-Format bereitgestellt werden.
1.7	Die Messdaten müssen von der Datenbank abgerufen werden (Pull-Verfahren).

5.2.2. Datenmodell

Die Anforderungen an das Datenbankmodell basieren auf eine möglichst modulare Struktur der Datenbank (siehe Tabelle 5.2 auf der nächsten Seite). Das Datenbankmodell sollte demnach für möglichst alle Sensoren verwendet werden können. Gleichzeitig sollte bei der Implementierung auf die Exportierbarkeit des Projektes und der Datenbank geachtet werden. Die Sammeltabelle wurde in Absprache mit abat auf 20 verschiedene zu speichernde Werte pro Datenbankeintrag beschränkt.

Tabelle 5.2.: Sensorik: Anforderungen - Datenmodell

Nr.	**Anforderung:**
2	**Datenmodell**
2.1	Ein geeignetes Datenbankschema muss erstellt werden.
2.2	Geeignete Datenbanktabellen müssen erzeugt werden.

Tabelle 5.2.: Sensorik: Anforderungen - Datenmodell

Nr.	Anforderung:
2	**Datenmodell**
2.2.1	Für alle eingehenden Messdaten, unabhängig von der Art des Sensors, ist eine Sammeltabelle anzulegen.
2.2.2	Die Sammeltabelle soll 20 Parameter speichern können.
2.2.3	Es muss eine separate Logging-Tabelle angelegt werden.
2.2.4	Für die Erstellung der Datenbank sind Datenbankstrukturdateien zu verwenden.

5.2.3. Daten empfangen

Die Anforderungen für das Empfangen der Daten wurden wieder in Raspberry Pi und Drucker geteilt (siehe Tabelle 5.3), da die Funktionsweise dieser beiden Sensortypen zu unterscheiden ist. Es wurde vorgegeben, dass die Raspberry Pis mithilfe eines Push-Verfahrens, die Daten an die HANA-Datenbank senden sollen. Im Gegenzug sollten die Drucker mithilfe eines Pull-Verfahrens ausgelesen werden. Diese Unterscheidung diente der Analyse und Machbarkeitsstudie beider Verfahren.

Tabelle 5.3.: Sensorik: Anforderungen - Daten empfangen

Nr.	Anforderung:
3	**Daten empfangen**
3.1	Die Empfangenen Daten müssen in einer HANA-Datenbank gespeichert werden.
Raspberry Pi	
3.2	Die Datenbank muss die Messdaten, die mittels des Push-Verfahrens zur Verfügung gestellt werden sollen, empfangen können.
3.3	Die Datenbank muss Fehlermeldungen empfangen können.
Drucker	
3.4	Die Datenbank muss die Messdaten, die mittels des Pull-Verfahrens zu beziehen sind, empfangen können.

5.2.4. Daten aufbereiten

Für das Aufbereiten der Daten wurden zwei Anforderungen gestellt, die über das gesamte Projekt einzuhalten waren (siehe Tabelle 5.4). Die gespeicherten Daten wollen für die GUI durch verschiedene Services aufbereitet werden. Dies soll rein auf Datenbankebene geschahen, um die Vorteile der HANA-Datenbank komplett auszunutzen und der GUI so wenig Geschäftslogik wie möglich zu überlassen.

Tabelle 5.4.: Sensorik: Anforderungen - Daten aufbereiten

Nr.	**Anforderung:**
4	**Daten aufbereiten**
4.1	Die Datenbank muss die gespeicherten Daten aufbereiten.
4.2	Die Aufbereitung der Daten muss auf Datenbank-Ebene geschehen.

5.2.5. Grafische Oberfläche

Die Anforderungen der grafischen Oberfläche beinhalten allgemeine Anforderungen aber auch Anforderungen, welche speziell für die Raspberry Pis oder Drucker definiert wurden (siehe Tabelle 5.5 auf der nächsten Seite). die allgemeinen Anforderungen besagen, dass die Applikation mithilfe von SAPUI5 und dem Model-View-Controller-Prinzip erstellt werden soll. Die grafische Oberfläche sollte einheitlich als Master-Detail-Struktur dargestellt werden.
Für die Raspberry Pis sollten vor allem historische Werte und Charts angezeigt werden können, wo hingegen die Drucker Füllstände und Status anzeigen sollten.

Tabelle 5.5.: Sensorik: Anforderungen - Grafische Oberfläche

Nr.	**Anforderung:**
5	**Grafische Oberfläche**
5.1	Muss modular gestaltet sein.
5.2	Die Daten müssen zyklisch aktualisiert werden.
5.3	Das SAPUI5-Framework muss verwendet werden.
5.3.1	Es ist das Model-View-Controller Architekturmuster zu verwenden.

Tabelle 5.5.: Sensorik: Anforderungen - Grafische Oberfläche

Nr.	Anforderung:
5	**Grafische Oberfläche**
5.3.2	Als Datenübertragungsformat ist die JavaScript Object Notation zu verwenden.
5.3.3	Die allgemeine Darstellung muss in der Master-/Detail-Struktur des SplitApp-Controls (SAPUI5) erfolgen.
5.4	Messwerte müssen beim Auswählen eines Sensors angezeigt werden.
Raspberry Pi	
5.5	Messwerte müssen angezeigt werden.
5.6	Historische Sensorwerte müssen grafisch aufbereitet (Chart, Tabellen o.ä.) dargestellt werden.
5.7	Es muss eine Tabelle zum Anzeigen von Ausnahmewerten erstellt werden.
5.8	Pro Raspberry Pi soll eine Kachel erstellt werden können.
Drucker	
5.9	Aktuelle Farbfüllstände müssen angezeigt werden (Yellow, Magenta, Cyan, Schwarz).
5.10	Papierfüllstand muss angezeigt werden.
5.11	Die Status des Druckers muss angezeigt werden.
5.12	Pro Drucker soll eine Kachel erstellt werden können.

5.2.6. Weitere Anforderungen

Abschließend wurden noch zwei weitere allgemeine Anforderungen an das Projekt gestellt (siehe Tabelle 5.6 auf der nächsten Seite). Die erstellte Applikation sollte als native HANA-Anwendung implementiert werden. Ebenfalls sollte es die Möglichkeit geben, eine Kachel in der HANA Administrationsoberfläche zu generieren.

Tabelle 5.6.: Sensorik: Anforderungen - weitere Anforderungen

Nr.	Anforderung:
6	**Daten aufbereiten**
6.1	Es ist eine native HANA-Anwendung zu implementieren.
6.2	Für die Anwendung ist eine Kachel auf der nativen HANA Administrationsoberfläche zu erstellen.

5.3. Umsetzung

In diesem Kapitel wird die Umsetzung und Implementierung der zuvor dargestellten Anforderungen genauer erläutert. Zuerst wird in Kapitel 5.3.1 die Architektur inklusive aller wesentlichen Elemente dargestellt und kurz beschrieben. Anschließend werden diese einzelnen Elemente, unterteilt in "HANA-Applikation", "Raspberry Pi" und "Graphical User Interface", detaillierter dargestellt. Hierbei werden die wesentlichen Implementierungen anhand von Code-Beispielen erläutert und aufgezeigt. Der gesamte Code der Implementierung dieses Szenarios ist auf der beigefügten CD zu finden.

5.3.1. Architektur

Die Architektur der Applikation des Sensorik Szenarios basiert auf logisch voneinander getrennten Paketen. Diese Pakete stellen jeweils ein wesentliches Logikelement der Gesamtstruktur dar und enthalten die jeweiligen Implementierungsobjekte. Insgesamt können zwölf elementare Pakete dieser Implementierung unterschieden werden (siehe Übersicht in Abbildung 5.1 auf der nächsten Seite).

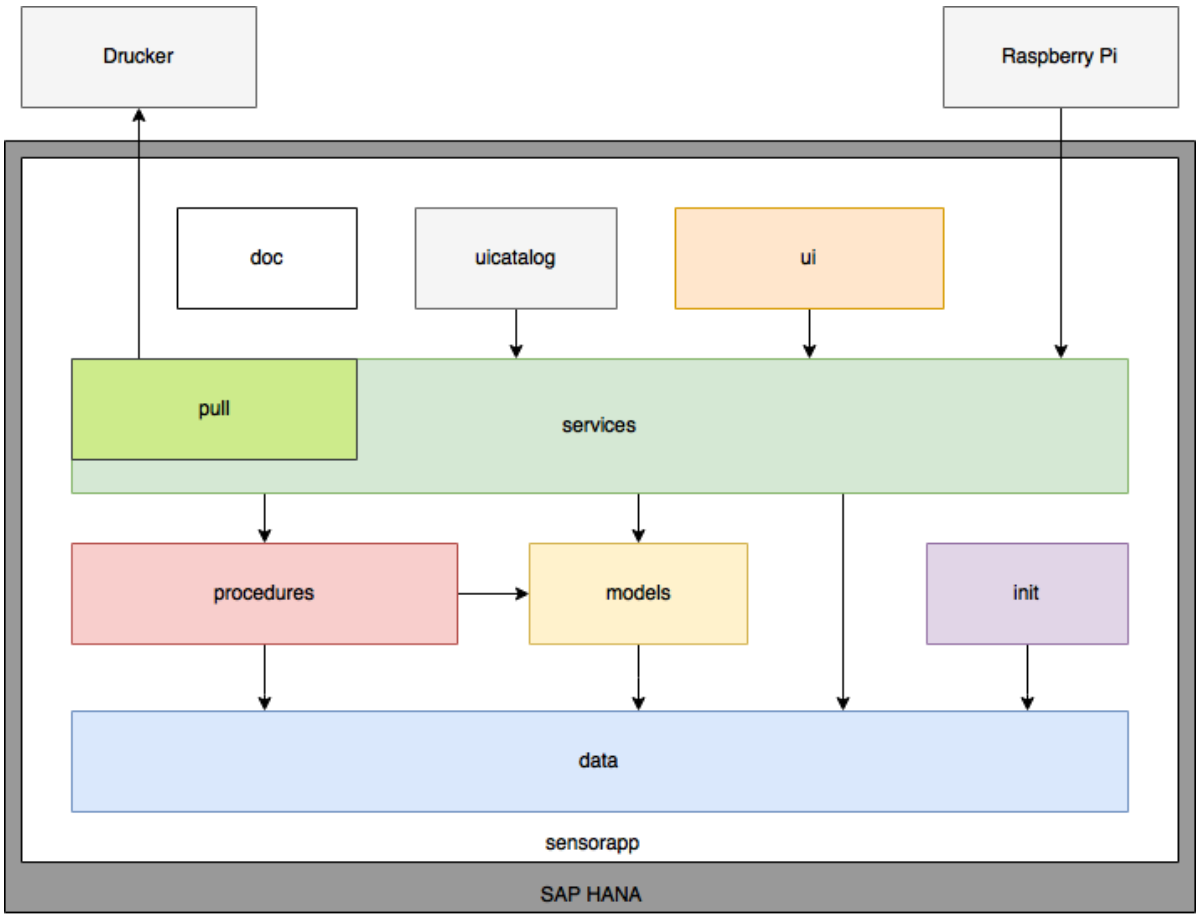


Abbildung 5.1.: Sensorik: Vereinfachte Darstellung der Gesamtarchitektur

Wie in Abbildung 5.2 auf der nächsten Seite dargestellt, wurden diese Pakete ebenfalls in der Projektstruktur berücksichtigt und bilden gleichzeitig den logischen Aufbau der gesamten Applikation. Durch diese Maßnahme wird eine höchstmögliche Übersicht sowie ein schnelles Verständnis des Projektes gefördert.

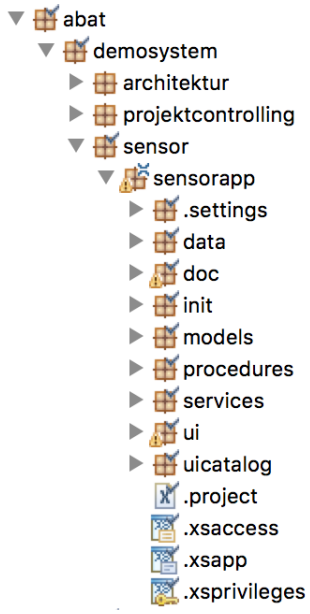


Abbildung 5.2.: Sensorik: Darstellung der Projektstruktur und Architektur

Im Folgenden wird die Funktion jedes Paketes für die Applikation kurz beschrieben, ohne dabei auf die jeweilige spezielle Implementierung einzugehen.

Drucker und Raspberry Pi

Zwei wesentliche Pakete des Szenarios sind die Drucker sowie die Raspberry Pis, welche außerhalb von SAP HANA dargestellt wurden. Beide sind für die Bereitstellung der Daten für die Datenbank elementar und bilden zusammen die Datengrundlage des Szenarios. Eine detaillierte Dokumentation der Implementierung des Raspberry Pis ist in Kapitel 5.3.3 auf Seite 124 zu finden.

sensorapp

Das größte Paket bildet die sensorapp-Applikation an sich. In ihr enthalten sind wiederum neun Pakete, die für die Umsetzung der HANA internen Applikation benötigt wurden. Dieses Paket bildet demnach den groben äußeren Rahmen der weiteren Implementierung (siehe Kapitel 5.3.2 auf Seite 100). Datenhaltung, Geschäftslogik sowie die grafische Benutzeroberfläche sind in diesem Paket enthalten. Ein einfacher Export und Import des Projektes ist somit Problemlos möglich.

data

Das Data-Paket enthält die gesamte Logik der Datenhaltung der Applikation. Datenbanktabellen, Sequenzen, benötigte Rollen und Schemata sind in diesem Paket enthalten und implementiert. Eine genauere Beschreibung der Implementierung ist im Abschnitt „data“ des Kapitel 5.3.2 auf Seite 100 zu finden.

init

Im Init-Paket sind Funktionen zum initialen Laden von Daten in die, durch das Data-Paket erstellte, Datenbank enthalten. Eine besondere Rolle spielt dieses Paket für den Im- und Export des Projektes und garantiert durch die enthaltene Implementierung (siehe Abschnitt „init“ in Kapitel 5.3.2 auf Seite 105) eine einfache Installation der erstellten Applikation.

models

Dieses Paket beinhaltet die erstellten Modelle, die die Abfrage von Daten aus der Datenbank durch bereitgestellte Funktionalitäten für die implementierten Services vereinfachen. Diese Modelle werden von den Paketen „procedures“ und „Services“ verwendet und nutzen die Implementierung des data-Pakets. Insgesamt wurden zwei Modelle für die Implementierung der Applikation erstellt. Eine detaillierte Beschreibung ist in Kapitel 5.3.2 auf Seite 107 im Abschnitt „models“ zu finden.

procedures

Das Procedures-Paket enthält alle angelegten Datenbank-Prozeduren. Diese enthalten die Geschäftslogik der Datenbankabfragen und dienen dem Speichern und Abrufen von Daten. Es werden Get-Prozeduren, Insert-Prozeduren und Update-Prozeduren unterschieden. Diese Prozeduren verwenden die Implementierung des Data-Pakets und werden ausschließlich vom Service-Paket verwendet. Eine Auflistung aller Prozeduren mit ihren Input- und Output-Parametern sowie ihrer Funktionalität ist in Kapitel 5.3.2 auf Seite 108 im Abschnitt „procedures“ zu finden.

services

Alle erstellten und von der GUI verwendeten Services sind im Service-Paket implementiert. Dieses Paket bildet die Hauptschnittstelle zwischen GUI und Datenbank. Die Services verwenden die Procedures und die Datenbank um Daten für die GUI aufzubereiten und bereitzustellen. Eine Auflistung aller implementierten Services sowie einer detaillierten Beschreibung dieser ist in Kapitel 5.3.2 auf Seite 112 im Abschnitt „services“ zu finden.

pull

Das Pull-Paket ist ein Unterpaket des Services-Paket und wurde wegen seiner besonderen Stellung jedoch trotzdem in die Grafik 5.1 auf Seite 97 und in diese Auflistung aufgenommen. Die implementierten Services dieses Pakets dienen ausschließlich der Datenabfrage der bereitgestellten Druckerdaten. Der Abschnitt „pull“ in Kapitel 5.3.2 auf Seite 117 beschreibt die Implementierung dieser Abfragen.

ui

Dieses Paket beinhaltet das gesamte SAPUI5-Projekt und enthält eine eigene Unterstruktur mit eigenen Paketen und Abhängigkeiten (vgl. Abbildung 5.21 auf Seite 132). Das Ui-Paket verwendet zur Kommunikation mit der Applikation ausschließlich die implementierten Services des Services-Paket. Das UI5-Projekt ist für jede Interaktion mit dem Endanwender verantwortlich. Dieses Paket sowie die darin enthaltenen Implementierungsobjekte sind in Kapitel 5.3.4 auf Seite 131 genauer dargestellt und erläutert.

uicatalog

Das Paket „uicatalog“ ist eine Erweiterung es Ui-Pakets und ist für die Darstellung einzelner Kacheln in der SAP HANA Administrations-Oberfläche verantwortlich. Die Daten der Kacheln erhält das Paket ebenfalls durch die implementierten Services.

doc

Das Doc-Paket enthält eine von SAP HANA automatisch generierte Javascript-Dokumentation. Diese dient der Übersichtlichkeit und der schnellen Informationsbereitstellung für Entwickler. In diesem Paket sind einzelne Funktionen und Objekte genauer beschrieben.

5.3.2. HANA-Applikation

In diesem Kapitel werden die zuvor kurz beschriebenen Applikationspakete genauer erläutert und dargestellt. Die Implementierung der einzelnen erstellten Objekte wird hierbei nur anhand von Codebeispielen aus der Applikation beschrieben. Der vollständige Code der HANA-Applikation ist auf der beigefügten CD zu finden

data

Im Folgenden wird die Implementierung des Data-Pakets detailliert erläutert. Das Data-Paket beinhaltet alle Implementierungsobjekte, die für die Datenhaltung und den Aufbau der Datenbank benötigt werden.
Um einen einfachen Export und Import dieser Applikation zu garantieren wurden alle Datenbankimplementierungen als eigenständige Objekte angelegt. Tabellen wurden als “hdbtable“-Objekt, Sequenzen als „hdbsequence“-Objekte, Schema als „hdbschema“-Objekte und Rollen als „hdbrole“-Objekte angelegt. Auf das Ausführen spezieller SQL-Befehle ohne Objekt wurde verzichtet (vergleiche Abbildung 5.3 auf der nächsten Seite).

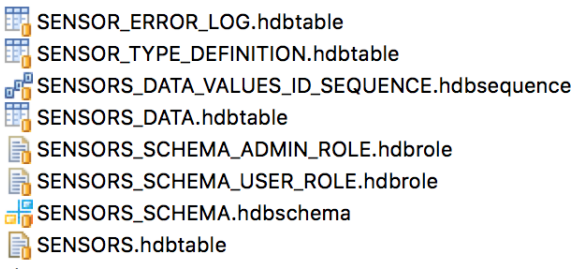


Abbildung 5.3.: Sensorik: Inhalt des data-Packages

Ebenfalls war bei der Erstellung und Konzeption des Datenbankmodells die Modularität der Datenstruktur ein entscheidender Faktor. Als Ergebnis konnte ein Datenbankmodell entwickelt werden, welches Werte unabhängig von der Art des Sensors oder der Art des gemessenen Wertes speichern und bereitstellen kann. In Abstimmung mit abat wurde lediglich eine Beschränkung von maximal 20 unterschiedlichen Werten (zehn numerische und 10 textuelle Werte) pro Sensor festgelegt (vgl. Abbildung 5.4 auf der nächsten Seite). Diese Einschränkung diente im Projektverlauf der Übersichtlichkeit und besseren Wartbarkeit des Prototypen. Eine nachträgliche Erweiterung um beliebig viele Werte ist dabei jedoch möglich. Eine grafische Darstellung des entwickelten Datenbankmodells ist in Abbildung 5.4 auf der nächsten Seite zu finden.

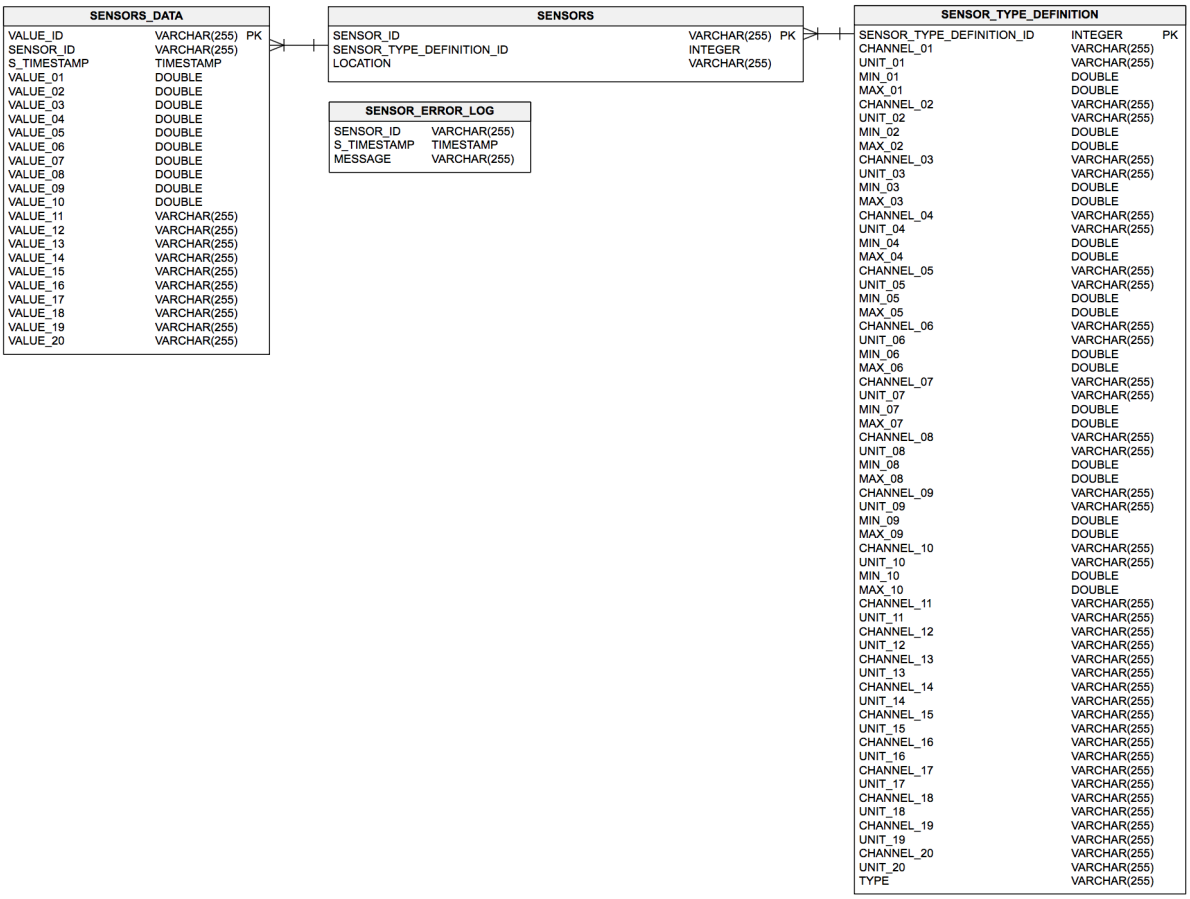


Abbildung 5.4.: Sensorik: Datenbankmodell

Alle dargestellten Tabellen, Rollen und Sequenzen sind in einem einheitlich angelegten Schema („SENSORS_SCHEMA“) gespeichert. Dieses ist durch die Datei „SENSORS_SCHEMA.hdbschema“ mit folgendem Code abgebildet:

```
1 schema_name = "SENSORS_SCHEMA";
```

Listing 5.1: Sensorik: Quellcode - SENSORS_SCHEMA.hdbschema

Das im Schema implementierte Datenbankmodell besteht aus insgesamt vier Tabellen, wobei drei der vier Tabellen für die eigentliche Funktionalität der Applikation verantwortlich sind. Die vierte „SENSOR_ERROR_LOG“ Tabelle bildet eine Hilfstabelle, welche zum Speichern von Fehlermeldungen der Raspberry Pis verwendet wird. Diese Tabelle dient der Wartbarkeit des Systems und bietet einen zentralen Speicherort von Fehlern und Warnungen. Technisch umgesetzt wurde diese Tabelle durch das „SENSOR_ERROR_LOG.hdbtable“ Objekt. Die Implementierung dieser Tabelle ist im Listing 5.2 auf der nächsten Seite zu finden.

```
table.schemaName = "SENSORS_SCHEMA";
table.tableType = COLUMNSTORE;

table.columns =
[
  {name = "SENSOR_ID";   sqlType = VARCHAR; length = 255; nullable = false
    ;},
  {name = "S_TIMESTAMP";   sqlType = TIMESTAMP; nullable = false;},
  {name = "MESSAGE"; sqlType = VARCHAR; length = 255;}
];
```

Listing 5.2: Sensorik: Quellcode - SENSOR_ERROR_LOG.hdbtable

Der Aufbau dieser hdbtable-Datei ist auf alle weiteren angelegten Tabellen übertragbar. Zu Beginn muss definiert werden, zu welchem Schema die Tabelle gehört und wie Daten in dieser Tabelle gespeichert werden sollen. Die von uns implementierten Tabellen gehören, wie bereits beschrieben, alle zum „SENSORS_SCHEMA“ und speichern ihre Daten als „COLUMNSTORE“. Wir haben uns für diesen Tabellentyp entschieden, um alle Vorteile von SAP HANA ausnutzen zu können. Durch den „COLUMNSTORE“ können große Datenmengen schneller gespeichert und im Anschluss effizienter Analysiert werden. Im Anschluss muss der genaue Inhalt der Tabelle beschrieben werden. Hierfür müssen die einzelnen Spalten der Tabelle sowie deren Typ und ggf. die gewünschte Länge dieser definiert werden.
Die Tabelle „SENSORS_DATA“ bildet die Haupttabelle der Datenbankmodells und ist für das Speichern der jeweiligen Sensorwerte verantwortlich. Diese Tabelle wird am häufigsten von der Applikation verwendet, da alle Sensorwerte in ihr gespeichert und aus ihr ausgelesen werden. Insgesamt werden in dieser Tabelle jeweils zehn numerische sowie zehn textuelle Werte gespeichert. Für die numerischen Werte wurde der Typ „Double“ verwendet, um Dezimalzahlen abbilden zu können. Für textuelle Werte wurde der Typ „VARCHAR“ mit einer Zeichenlänge von 255 Zeichen verwendet.
Zusätzlich wird pro Eintrag der aktuelle Zeitstempel, die jeweilige Sensor-ID und eine generierte Wert-ID gespeichert. Diese, durch eine Sequenz automatisch generierte, Wert-ID dient als Primärschlüssel der Tabelle und zur eindeutigen Identifizierung einzelner Einträge (vgl. Implementierung im Listing 5.3).

```
table.primaryKey.pkcolumns = ["VALUE_ID"];
```

Listing 5.3: Sensorik: Definition des PrimaryKeys der SENSORS_DATA Tabelle

Die hier verwendete Sequenz ist als hdbsequence-Datei implementiert (vgl. Listing 5.4) und wird ausschließlich für die „SENSORS_DATA“ verwendet.

```
schema="SENSORS_SCHEMA";
start_with = 5;
depends_on_table="abat.demosystem.sensor.sensorapp.data::SENSORS_DATA";
```

Listing 5.4: Sensorik: Quellcode - SENSORS_DATA_VALUES_ID_SEQUENCE .hdbsequence

Hierbei ist zu beachten, dass das verwendete Schema (vgl. Zeile 1 im Listing 5.4 auf der vorherigen Seite) sowie die abhängige Tabelle der Sequenz (vgl. Zeile 3 im Listing 5.4 auf der vorherigen Seite) definiert sind. Der Aufruf dieser Sequenz findet in der angelegten „insert_value_procedure“ statt.

Die Tabelle „SENSORS“ speichert die verschiedenen Sensoren, sowie ihren Standort. Diese Tabelle dient dem Mapping der einzelnen Werte aus der „SENSORS_DATA“ Tabelle mit den Wert-Definitionen der „SENSOR_TYPE_DEFINITION“ Tabelle (siehe Datenbankmodell in Abbildung 5.4 auf Seite 102).

Die von der Spaltenanzahl größte Tabelle ist die „SENSOR_TYPE_DEFINITION“ Tabelle. Sie enthält für jeden Wert der Tabelle „SENSORS_DATA“ den zugehörigen Channel sowie die damit verbundene Einheit des Wertes (z.B. Channel: „Temperatur“, Einheit (Unit): „°C“). Durch diese Tabelle ist die Modularität des Datenmodells gegeben und es können beliebige Werte in die „SENSORS_DATA“ Tabelle eingefügt und in der „SENSOR_TYPE_DEFINITION“ Tabelle definiert werden. Zusätzlich können für die zehn numerischen Werte jeweils ein Minimal- und ein Maximalwert angegeben werden. Diese Angaben helfen der Applikation bei der Analyse von Fehlerwerten und Anomalien.

Für die Verwendung dieser erstellten Datenbankobjekte werden Rollen benötigt. Diese müssen den einzelnen Nutzern manuell zugewiesen werden. Aus Datenschutz- und Datensicherheitsgründen wurden für diese Applikation zwei unterschiedliche Rollen angelegt. Die Rolle „SENSORS_SCHEMA_USER_ROLE“ definiert alle Berechtigungen, die ein Standardnutzer der Anwendung Nutzen darf. Um Fehler und mögliche Schäden durch die Nutzer zu verringern wurden diese Rechte auf das Minimum reduziert. Die Implementierung dieser Rolle ist im Listing 5.5 dargestellt.

```
1  //Eine Rolle fuer die einfache Nutzung des Schemas
2
3  role abat.demosystem.sensor.sensorapp.data::SENSORS_SCHEMA_USER_ROLE {
4    catalog schema "SENSORS_SCHEMA" : SELECT;
5    schema abat.demosystem.sensor.sensorapp.data:SENSORS_SCHEMA.hdbschema:
       SELECT;
6    catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::insert_value_procedure" : EXECUTE;
7    catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::getValuesForDash_procedure" : EXECUTE;
8    catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::getAllSensorsForType_procedure" : EXECUTE;
9    catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::getChannelValuesForTime_procedure" : EXECUTE;
10   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::updateExtremeFlag_procedure" : EXECUTE;
11   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::insert_log_procedure" : EXECUTE;
12   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::getCurrentValueForId_procedure" : EXECUTE;
13   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       procedures::getValuesForIdAndChannel_procedure" : EXECUTE;
14   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       models::AT_SENSOR_VALUES" : SELECT;
```

```
15   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       data::SENSORS_DATA" : SELECT, INSERT;
16   catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.
       data::SENSOR_ERROR_LOG" : SELECT, INSERT;
17   application privilege: abat.demosystem.sensor.sensorapp::Basic;
18   application privilege: abat.demosystem.sensor.sensorapp.uicatalog::
       SensorAppLauncher;
19 }
```

Listing 5.5: Sensorik: Quellcode - SENSORS_SCHEMA_USER_ROLE.hdbrole

In dieser Rolle müssen für jedes verwendete SQL-Objekt die gewünschten Rechte definiert werden. In der User-Role sind ausschließlich die Rechte „EXECUTE“, „SELECT“ und „INSERT“ vergeben worden. Ein ungewolltes Löschen oder Ändern von Datenbankeinträgen durch einen Standardnutzer wird somit verhindert. Neben den SQL-Objekt-Rechten werden in der Rolle ebenfalls die Applikationsrechte definiert (vgl. Zeile 17f. im Listing 5.5 auf der vorherigen Seite). Diese werden zum Ausführen und Anmelden in die erstellte Applikation vom Nutzer benötigt.
Die Rolle „SENSORS_SCHEMA_ADMIN_ROLE“ sollte allen Administratoren der Applikation zugeteilt werden. Diese Erweitert die Rechte der User-Role (vgl. Zeile 4 im Listing 5.6) um „DELETE“, „UPDATE“ und „DROP“ Rechte der einzelnen Datenbankobjekte.

```
1  // Eine Rolle fuer die Administration des Schemas
2
3  role abat.demosystem.sensor.sensorapp.data::SENSORS_SCHEMA_ADMIN_ROLE
4    extends role abat.demosystem.sensor.sensorapp.data::
       SENSORS_SCHEMA_USER_ROLE
5    {
6      catalog schema "SENSORS_SCHEMA" : SELECT, INSERT, UPDATE, DROP, DELETE
       ;
7      schema abat.demosystem.sensor.sensorapp.data:SENSORS_SCHEMA.hdbschema:
       SELECT, INSERT, UPDATE, DROP, DELETE;
8
9      application privilege: abat.demosystem.sensor.sensorapp::Admin;
10     application privilege: abat.demosystem.sensor.sensorapp.uicatalog::
       SensorAppLauncher;
11
12     system privilege: USER ADMIN;
13   }
```

Listing 5.6: Sensorik: Quellcode - SENSORS_SCHEMA_ADMIN_ROLE.hdbrole

init

In diesem Paket finden sich jeweils für die „SENSORS“ und „SENSORS_TYPE_DEFINITION“ Tabellen eine hdbti- sowie eine csv-Datei wieder (siehe Abbildung 5.5 auf der nächsten Seite). Diese wurden angelegt um beide Tabellen initial mit Werten zu befüllen. So können Sensoren inklusiver ihrer Wertdefinitionen zentral in diesen Dateien gewartet werden. Das manuelle Ausführen von speziellen SQL-Befehlen ist somit nicht mehr nötig,

ein Aktivieren dieser Objekte genügt um die Daten der Tabellen zu manipulieren oder zu erweitern.

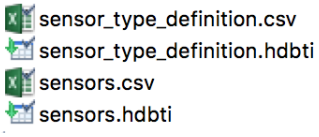


Abbildung 5.5.: Sensorik: Inhalt des init-Packages

Die jeweilige CSV-Datei enthält die einzelnen Werte und Zeilen, die in der gewünschten Tabelle gespeichert werden sollen. Hierbei ist zu beachten, dass die Reihenfolge der Werte mit der Reihenfolge der Spalten in der angelegten Tabelle übereinstimmen muss. Im Listing 5.7 ist die CSV-Datei zum befüllen der "SENSORS" Tabelle zu finden. Wie zu erkennen ist, wurde hier die durch die Tabelle vorgegebene Spaltenreihenfolge von ID, Definition_ID und Standort eingehalten (vgl. Datenbankmodell Abbildung 5.4 auf Seite 102).

```
1 "172.18.5.2";1;"2.OG Azudenten"
2 "172.18.5.4";1;"2.OG Treppe"
3 "172.18.5.5";1;"2.OG IT-Abteilung"
4 "172.18.4.254";1;"3.OG Berater"
5 "172.18.1.12";2;"2.OG"
6 "172.18.1.13";3;"3.OG"
```

Listing 5.7: Sensorik: Implementierung der sensors.csv

Um diese CSV-Datei in die jeweilige Tabelle zu importieren, wird eine hdbti-Datei benötigt, die entsprechend der Zieltabelle und des Formats konfiguriert werden muss. Im Listing 5.8 ist die Implementierung dieser Datei für die oben beschriebene CSV-Datei der „SENSOR"-Tabelle dargestellt.

```
1 import = [
2   {
3     table = "abat.demosystem.sensor.sensorapp.data::SENSORS";
4     schema = "SENSORS_SCHEMA";
5     file = "abat.demosystem.sensor.sensorapp.init:sensors.csv";
6     header = false;
7     useHeaderNames = false;
8     delimField = ";";
9     delimEnclosing="\"";
10    distinguishEmptyFromNull = true;
11  }
12 ];
```

Listing 5.8: Sensorik: Quellcode - sensors.hdbti

Zu beginn dieses Imports muss definiert werden, in welche Tabelle und in welches Schema die Daten importiert werden sollen (vgl. Zeile 3f. im Listing 5.8). Anschließend wird der Pfad zur zuvor erzeugten CSV-Datei angegeben (Zeile 5). Zuletzt muss der Aufbau der

CSV-Datei für den Import definiert werden. Es muss angegeben werden, ob die CSV-Dateien einen Header besitzen, ob dieser verwendet werden soll und mit welchem Zeichen Spalten und Zeilen getrennt werden (Zeilen 6-10).
Werden beide Objekte Aktiviert, so lädt die HANA Datenbank automatisch alle Werte aus den CSV-Dateien in die angegebenen Tabellen.
Die Implementierung dieser Dateien für die „SENSORS_TYPE_DEFINITION“ Tabelle ist analog zur oben beschriebenen Implementierung.

models

In diesem Paket wurden alle verwendeten Models implementiert und gespeichert. Die Implementierung dieser Models erfolgte ausschließlich über den grafischen Editor (siehe Abbildung 5.6). Der zugehörige XML-Code, sowie die Model-Reports wurden automatisch generiert.

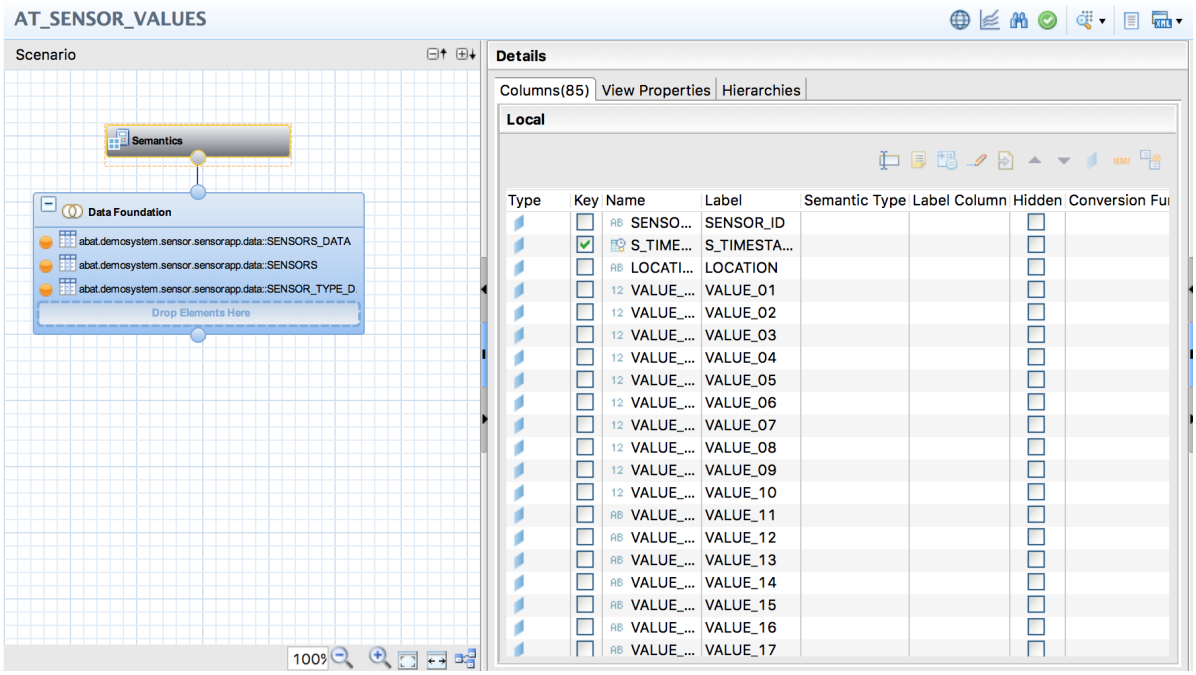


Abbildung 5.6.: Sensorik: Screenshot des grafischen Model-Editors

Insgesamt wurden für diese Applikation zwei Views erstellt (siehe Abbildung 5.7), welche jedoch lediglich das Joinen verschiedener Tabellen abbilden und somit nur der Vereinfachung und Übersichtlichkeit dienen und keine komplexe Logik oder Kalkulation enthalten.

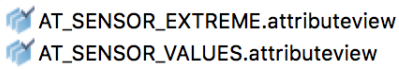


Abbildung 5.7.: Sensorik: Inhalt des models-Packages

Die „AT_SENSOR_VALUES“ View verknüpft alle drei Sensor-Tabellen miteinander und gibt eine Tabelle mit jeder enthaltenen Spalte aller Tabellen aus. Insgesamt enthält der Output dieser View 85 Spalten. Mithilfe dieser View können zum Beispiel erste Auswertungen bezüglich der bisher gesammelten Werte pro Sensor dargestellt werden (siehe Abbildung 5.8).

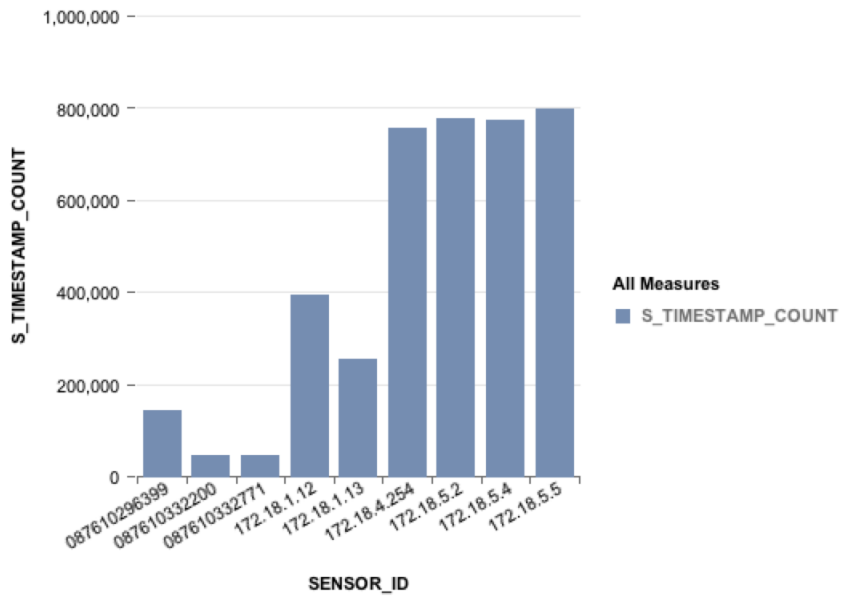


Abbildung 5.8.: Sensorik: Beispielhafte grafischen Auswertung der „AT_SENSOR_VALUES“ View (Stand 18.01.17)

Die „AT_SENSOR_EXTREME“ joint ebenfalls alle drei Tabellen miteinander, gibt als Output aber nur relevante Spalten für die Berechnung von Extrem und Fehlerwerten aus. Der Output dieser View beträgt 45 Spalten. Auch diese View dient der Übersichtlichkeit und enthält keine eigene Kalkulationen.

procedures

Im „procedures“-Paket sind alle implementierten Procedures enthalten (siehe Abbildung 5.9).

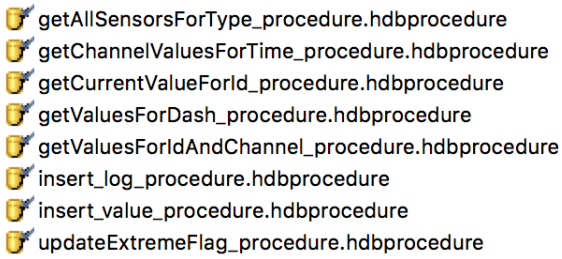


Abbildung 5.9.: Sensorik: Inhalt des procedures-Packages

Insgesamt wurden fünf Get-Prozeduren (vgl. Tabelle 5.7) sowie zwei Insert- und eine Update-Prozedur (vgl. Tabelle 5.8 auf der nächsten Seite) erstellt. Diese Prozeduren können von den Services verwendet werden um SQL-Befehle auf der Datenbank auszuführen, um so Daten zu manipulieren oder auszugeben. Der Aufruf von Procedures bringt gegenüber dem direkten Aufruf der Datenbank durch die Services den Vorteil einer weiteren Sicherheitsschicht (vgl. Architekturmodell 5.1 auf Seite 97). So lassen sich zum Beispiel Fehler besser vermeiden oder unbefugte Zugriffe verhindern.

Tabelle 5.7.: Sensorik: Get-Procedures

Name	**Input**	**Output (Table/View)**
getAllSensorsForType	sensor_type: VARCHAR(255)	SENSORS
getChannelValuesForTime	sensor_id: VARCHAR(255), from_Time: TIMESTAMP	AT_SENSOR_VALUES
getCurrentValueForId	sensor_id: VARCHAR(255)	AT_SENSOR_VALUES
getValuesForDash	sensor_id: VARCHAR(255), amount: int	AT_SENSOR_VALUES
getValuesForIdAndChannel	sensor_id: VARCHAR(255), amount: int	AT_SENSOR_VALUES

Tabelle 5.8.: Sensorik: Insert/Update-Procedures

Name	**Input**
insert_log	SENSOR_ID: VARCHAR(255), MESSAGE: VARCHAR(255))
insert_value	SENSOR_ID: VARCHAR(255), VAL_01: DOUBLE, VAL_02: DOUBLE, VAL_03: DOUBLE, VAL_04: DOUBLE, VAL_05: DOUBLE, VAL_06: DOUBLE, VAL_07: DOUBLE, VAL_08: DOUBLE, VAL_09: DOUBLE, VAL_10: DOUBLE, VAL_11: VARCHAR(255), VAL_12: VARCHAR(255), VAL_13: VARCHAR(255), VAL_14: VARCHAR(255), VAL_15: VARCHAR(255), VAL_16: VARCHAR(255), VAL_17: VARCHAR(255), VAL_18: VARCHAR(255), VAL_19: VARCHAR(255), VAL_20: VARCHAR(255), EXTREME_FLAG: VARCHAR(255))

Tabelle 5.8.: Sensorik: Insert/Update-Procedures

Name	**Input**
updateExtremeFlag	VALUE_ID: VARCHAR(255)

Der Aufbau einer Procedure folgt dabei immer dem gleichen Muster und wird im Folgenden anhand der „getAllSensorsForType“-Procedure genauer erläutert (siehe Listing 5.9).

```
1  PROCEDURE "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.procedures::
       getAllSensorsForType_procedure" (
2    IN sensor_type VARCHAR(255),
3    OUT result "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.data::
       SENSORS")
4
5    LANGUAGE SQLSCRIPT
6    SQL SECURITY INVOKER
7    DEFAULT SCHEMA "SENSORS_SCHEMA"
8    READS SQL DATA AS
9  BEGIN
10   /*****************************
11   Alle Sensoren fuer einen bestimmten Typ ausgeben
12   *****************************/
13   result = SELECT SENSORS.SENSOR_ID, SENSORS.SENSOR_TYPE_DEFINITION_ID,
       SENSORS.LOCATION FROM "SENSORS_SCHEMA"."abat.demosystem.sensor.
       sensorapp.data::SENSORS" SENSORS, "SENSORS_SCHEMA"."abat.demosystem.
       sensor.sensorapp.data::SENSOR_TYPE_DEFINITION" DEF
14   WHERE DEF.S_TYPE = :sensor_type AND SENSORS.SENSOR_TYPE_DEFINITION_ID =
       DEF.SENSOR_TYPE_DEFINITION_ID;
15 END;
```

Listing 5.9: Sensorik: Quellcode - getAllSensorsForType_procedure.hdbprocedure

Diese Prozedur erhält den Sensor-Typ als Input-Parameter (siehe Zeile 2 Listing 5.9) und soll eine Tabelle im Format der „SENSORS“-Tabelle (siehe Datenbankmodell 5.4 auf Seite 102) mit den abgefragten Werten ausgeben (siehe Zeile 3). Anschließend wird definiert, dass die Sprache dieser Prozedur „SQLSCRIPT“ und das zu verwendende Schema das „SENSORS_SCHEMA“ ist (siehe Zeilen 5-8). Nach dem „BEGIN“ Schlagwort folgt dann die eigentliche Logik der Prozedur. In diesem Beispiel ist dies ein einfacher SQL-Befehl, welcher die Tabellen „SENSORS“ und „SENSOR_TYPE_DEFINITION“ verknüpft und alle Sensoren sucht, die mit dem Input-Sensor-Typ übereinstimmen (Zeilen 9-15).

Zum Testen der erstellten Procedure kann diese mithilfe eines SQL-Call-Befehls manuell aufgerufen werden (siehe Abbildung 5.10 auf der nächsten Seite). Man erhält so einen Überblick über den Output der Procedure und kann so kontrollieren, ob die gewünschte Funktion durch die Prozedur gegeben ist.

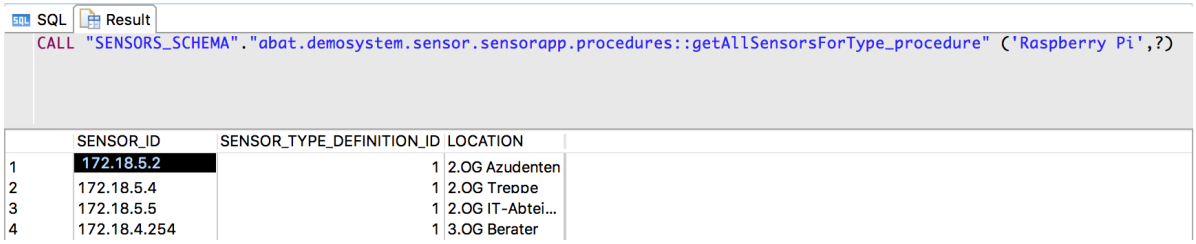


Abbildung 5.10.: Sensorik: Manueller Test einer Procedure durch die SQL-Konsole

Im folgenden werden die Funktionen der weiteren Prozeduren kurz erläutert (vgl. mit Tabellen 5.7 auf Seite 109 und 5.8 auf der vorherigen Seite). Ihre Implementierung ist analog zur oben beschriebenen Procedure:

- **getChannelValuesForTime** Diese Procedure gibt alle Werte eines bestimmten Sensors ab einem angegebenen Zeitpunkt aus.

- **getCurrentValueForId** Mit Aufruf dieser Prozedur erhält man die aktuellsten Werte eines bestimmten Sensors.

- **getValuesForDash** Diese Prozedur gibt eine bestimmte Anzahl der aktuellsten Werte eines Sensors aus. Sie wird für das Anzeigen der X aktuellsten Werte durch das Dashboard verwendet.

- **getValuesForIdAndChannel** Ein Aufruf dieser Procedure gibt eine bestimmte Anzahl alle Werte eines angegebenen Sensors zurück. Diese Prozedur wird hauptsächlich zum dynamischen Darstellen der Info-Tabellen verwendet.

- **insert_log** Durch diese Prozedur können Nachrichten und Warnmeldungen in die Log-Tabelle (vgl. Datenbankmodell 5.4 auf Seite 102) geschrieben werden.

- **insert_value** Diese Prozedur schreibt die empfangenen Sensorwerte in die „SENSORS_DATA“ Tabelle.

- **updateExtremeFlag** Mithilfe dieser Procedure lässt sich die Extrem-Markierung in der „SENSORS_DATA“ Tabelle manipulieren. Sie wird aufgerufen, wenn der Nutzer Warnmeldungen ausblenden möchte.

Eine Darstellung der Procedures mit ihren jeweiligen Aufrufen aus der Datenbank ist in Abbildung 5.11 auf der nächsten Seite zu finden.

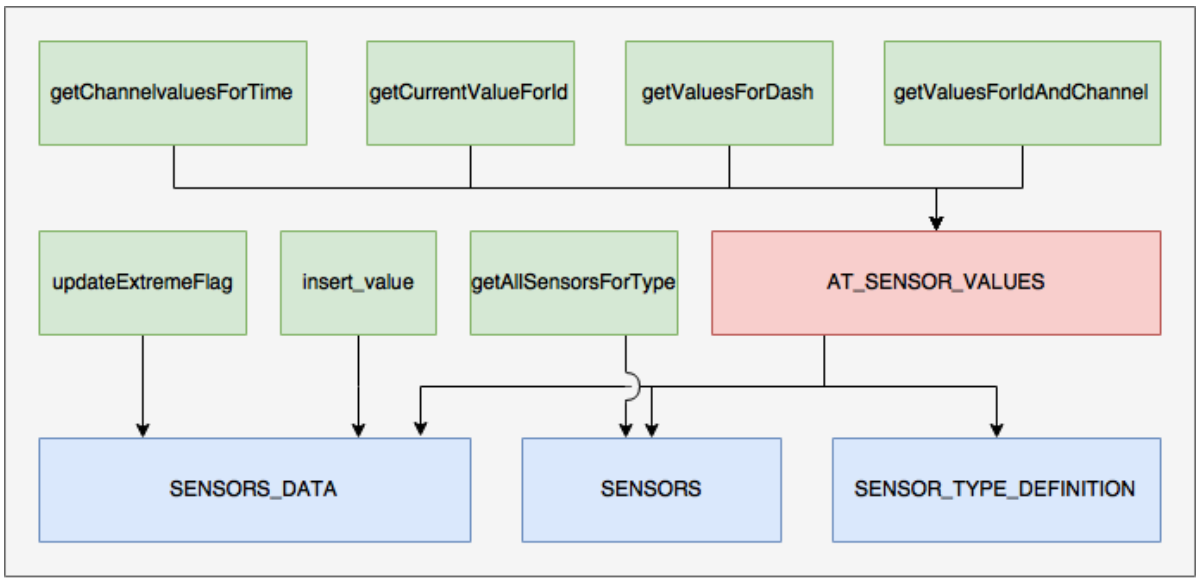


Abbildung 5.11.: Sensorik: Darstellung der Datenbank-Aufrufe jeder Procedure

services

In diesem Paket befinden sich alle erstellten Services, die für die Funktionalität der Applikation benötigt werden. Insgesamt wurden zwölf Services (ein xsodata- und elf xsjs-Services) für die Bereitstellung der Daten implementiert. Zusätzlich wurden drei Services zum Speichern und Manipulieren von Daten entwickelt. Für eine bessere Übersicht und um Code-Redundanzen zu vermeiden, wurde ebenfalls eine xsjslib-Datei erstellt (siehe Übersicht in Abbildung 5.12).

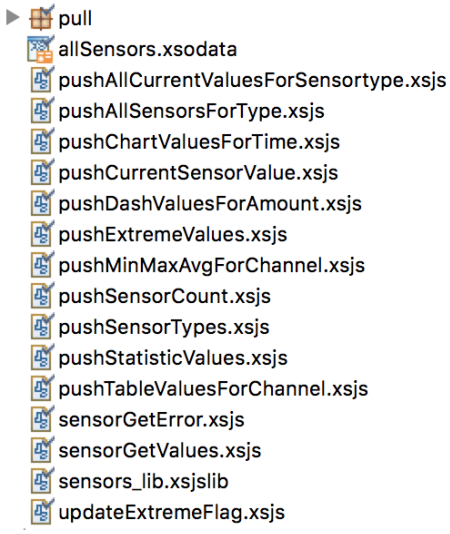


Abbildung 5.12.: Sensorik: Darstellung des Service-Pakets

Die „sensors_lib.xsjslib“-Datei dient für das gesamte Projekt als eigene JS-Bibliothek. In

dieser Datei sind zwei Funktionen enthalten, welche in vielen Services benötigt werden. Um diese nicht immer wieder neu zu Implementieren, kann ein Service die Funktion dieser Bibliothek aufrufen. Coderedundanzen wurden so vermieden.
Um einen Aufruf der Lib-Datei zu implementieren, benötigt der jeweilige Service zuerst einen Verweis auf die entsprechende Lib-Datei (siehe Listing 5.10).

```
1 $.import("abat.demosystem.sensor.sensorapp.services", "sensors_lib");
2 var SENSORS_LIB = $.abat.demosystem.sensor.sensorapp.services.sensors_lib;
```

Listing 5.10: Sensorik: Implementierung eines Verweises auf die “sensors_lib.xsjslib“

Nachdem der Verweis im Service implementiert wurde, können die Funktionen der Bibliotheksdatei durch die Aufruf „SENSORS_LIB.FUNKTIONSNAME“ verwendet werden. Der allgemeine Aufbau eines Services ist für die Push-, sowie Get-Services überwiegend identisch und wird im folgenden für jeweils einen Service der Kategorie genauer erläutert.

Get/Update Der „sensorGetError.xsjs“ Service empfängt die Fehlermeldungen der Raspberry Pis und speichert diese in der „SENSOR_ERROR_LOG“ Tabelle. Um einem Service Parameter zu übergeben, müssen diese in der Aufruf-URL hinterlegt sein („?id=XX“). Um diese Parameter auszulesen, kann der Service die Parameter des Requests anfordern (siehe Listing 5.11):

```
1 // Zuweisen der Variablen in der Aufruf-URL
2 var sensorId = $.request.parameters.get("id");
3 var message = $.request.parameters.get("message");
```

Listing 5.11: Sensorik: Quellcode - sensorGetError.xsjs (URL-Parameter)

Anschließend können die Parameter verarbeitet werden. In diesem Beispiel werden die Parameter für einen Procedure-Call benötigt, welcher diese in die jeweilige Datenbanktabelle speichert. Für einen Procedure-Aufruf benötigt der Service zuerst eine Datenbank-Verbindung (vgl. Listing 5.12 Zeile 19). Anschließend kann der Datenbank-Aufruf vorbereitet werden. Hierfür wird der Aufruf mit variablen Parametern „?“ als Variable gespeichert (Zeile 20). Diese Variablen werden dann im Folgenden mit den Werten der zuvor enthaltenen Parametern gefüllt (Zeile 21 ff). Ist der Procedure-Aufruf vollständig definiert, kann dieser durch ein „execute“ Statement auf der Datenbank ausgeführt werden (Zeile 26). Gleichzeitig wird das ResultSet (das Ergebnis der Datenbankabfrage) in einer Variablen gespeichert. Sollten bei der Abfrage Fehler auftreten, so werden diese Abgefangen und ausgegeben (Zeile 33 ff).

```
15   $.response.contentType = "text/plain";
16   body = sensorId;
17   try {
18     // Den Procedure Call vorbereiten und mit den Variablen fuellen
19     var conn = $.db.getConnection();
20     var query = 'call \"abat.demosystem.sensor.sensorapp.procedures::
     insert_log_procedure\"(?,?)';
21     var cst = conn.prepareCall(query);
22     cst.setString(1, sensorId);
23     cst.setString(2, message);
```

```
24
25      //Den Procedure-Call ausfuehren
26      var rs = cst.execute();
27      conn.commit();
28      if(rs === 0){
29        body = "TRUE";
30      }
31      conn.close();
32      // Falls Fehler auftauchen, diese Anzeigen lassen
33    } catch (e) {
34      rs.body = e.stack + "\nName:"+ e.name+"\nMsg" + e.message + e.
        printStackTrace();
35      $.response.status = $.net.http.BAD_REQUEST;
36      }
37    }
```

Listing 5.12: Sensorik: Quellcode - sensorGetError.xsjs (Procedure-Call)

Abschließend wird eine Antwort des Services generiert (siehe Listing 5.13). Hierbei wird der Typ der Antwort definiert und der Body gesetzt.

```
38 $.response.contentType = "text/plain";
39 $.response.setBody(body);
```

Listing 5.13: Sensorik: Quellcode - sensorGetError.xsjs (Response)

Dieses Vorgehen ist für alle Services dieser Kategorie einheitlich. Im Folgenden werden die Funktionen der weiteren Services genauer Erläutert.

- **sensorGetValues.xsjs** Dieser Service erhält als Parameter durch die Raspberry Pis insgesamt 20 Werte (zehn numerische und zehn textuelle Werte), welche anschließend in der Tabelle „SENSORS_DATA“ gespeichert werden.

- **updateExtremeFlag.xsjs** Mithilfe dieses Services kann die gesetzte ExtremeFlag der „SENSORS_DATA“ Tabelle geändert werden. Als Input-Parameter erhält dieser Service die jeweilige Werte-ID

Push Für die Bereitstellung der Daten aus der Datenbank für die GUI wurden ein xsodata-Service sowie elf xsjs-Services implementiert.
Der allSensors.xsodata-Service gibt alle Sensoren der „SENSORS“ Tabelle als Odata-Service aus (vgl Listing 6.8 auf Seite 222).

```
1 service  {
2   "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.data::SENSORS" as "
      sensors";
3 }
```

Listing 5.14: Sensorik: Quellcode - allSensors.xsodata

Mithilfe folgendem Aufrufs gibt der Service eine JSON-Liste aller Sensoren wieder (siehe Abbildung 5.13 auf der nächsten Seite):

http://hana-d21.abatgroup.de:8002/abat/demosystem/sensor/sensorapp/services/allSensors.xsodata/sensors?$format=json


```
{
  - d: {
     - results: [
        - {
           - __metadata: {
                type: "abat.demosystem.sensor.sensorapp.services.allSensors.sensorsType",
                uri: "http://hana-
                d21.abatgroup.de:8002/abat/demosystem/sensor/sensorapp/services/allSensors.xsodata/sensors('087610296399')"
             },
             SENSOR_ID: "087610296399",
             SENSOR_TYPE_DEFINITION_ID: 4,
             LOCATION: "2.OG Wasserkocher"
          },
```


Abbildung 5.13.: Sensorik: Ergebnis eines OData-Aufrufs

Die Struktur dieses JSON-Objektes ist dabei durch das OData-Format vorgegeben und automatisch generiert. Die erstellten xsjs-Services geben eine angepasste, selbst erstellte JSON-Struktur ihrer Ergebnisse aus.
Diese Services sind analog zu den oben beschriebenen Get- und Update-Services aufgebaut. Auch hier werden wesentliche Parameter über die URL übergeben und Datenbankabfragen durch Procedure-Calls oder direkte Aufrufe der Datenbank getätigt (vgl Listing 5.15).

```
13 var conn = $.db.getConnection();
14
15 // Die Query zum Errechnen der Anzahl der Sensoren
16 var rs = SENSORS_LIB.executeSQLQuery(conn, 'SELECT COUNT(*) FROM "
   SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.data::SENSORS"');
```

Listing 5.15: Sensorik: Implementierung eines direkten Datenbankaufrufes durch einen Service (pushSensorCount.xsjs)

Bei einem direkten Datenbankzugriff durch Ausführen eines SQL-Statements ist die Datensicherheit und Integrität besonders zu beachten. Sicherheitsmechanismen, welche durch die Procedures gegeben sind, werden bei einem direkten Aufruf nicht beachtet. Die Möglichkeit einer SQL-Injektion sollte bei dieser Art des Zugriffes immer kritisch betrachtet werden.
Nachdem der Service das Resultset des Datenbankaufrufes erhalten hat, kann dieses mit einer While-Schleife durchlaufen werden (vgl. Listing 5.16).

```
28 rs = cst.getResultSet();
29 // Alle Werte des Resultsets speichern
30 while(rs.next()){
31   data.sensors.push({
32     "sensor_id": rs.getNString(1),
33     "type_definition_id": rs.getDouble(2),
34     "location": rs.getNString(3),
35     "type": sensorType
36   });
37 }
```

Listing 5.16: Sensorik: Durchlaufen eines ResultSets (pushAllSensorsForType.xsjs)

Gleichzeitig können die Einträge des ResultSets in Variablen gespeichert werden, welche für die Struktur des JSON-Outputs verantwortlich sind. Diese Variablen müssen anschließend in das JSON-Format konvertiert werden (vgl. Listing 5.17).

```
28 // Die gespeicherten Daten als JSON dem body uebergeben
29 body = JSON.stringify(data);
```

Listing 5.17: Sensorik: JSON-Konvertierung (pushAllSensorsForType.xsjs)

Das Ergebnis eines Aufrufes des pushAllSensorsForType.xsjs Services mit dem Parameter „Rasp berry Pi“ enthält demnach folgenden Output:


```
{
  - sensors: [
      - {
            sensor_id: "172.18.5.2",
            type_definition_id: 1,
            location: "2.OG Azudenten",
            type: "Raspberry Pi"
        },
      - {
            sensor_id: "172.18.5.4",
            type_definition_id: 1,
            location: "2.OG Treppe",
            type: "Raspberry Pi"
        },
      - {
            sensor_id: "172.18.5.5",
            type_definition_id: 1,
            location: "2.OG IT-Abteilung",
            type: "Raspberry Pi"
        },
      - {
            sensor_id: "172.18.4.254",
            type_definition_id: 1,
            location: "3.OG Berater",
            type: "Raspberry Pi"
        }
    ]
}
```


Abbildung 5.14.: Sensorik: Ergebnis eines XSJS-Aufrufs

Im Folgenden werden die Funktionalitäten der nicht im Detail beschriebenen Services kurz dargestellt. Das Vorgehen bei der Implementierung dieser Services ist analog zu den oben beschrieben Services.

- **pushAllCurrentValuesForSensortype** Dieser Service stellt alle aktuellen Werte jedes Sensors des angegebenen Typs bereit. Zudem werden die Einheiten, sowie die Min- und Max-Werte und der Durchschnitt berechnet und angegeben.
- **pushChartValuesForTime** Mithilfe dieses Services werden eine bestimmte Anzahl an Werten (berechnet durch einen angegebenen Zeitraum) für einen definierten Sensor und einen definierten Channel übergeben. Dieser Service dient dem Anzeigen von Charts.

- **pushCurrentSensorValue** Dieser Service stellt alle aktuellen Werte eines bestimmten Sensors bereit.

- **pushDashValuesForAmmount** Durch diesen Service werden eine bestimmte Anzahl an Werten für einen übergebenen Channel eines bestimmten Sensors dargestellt. Dieser Service wird in der GUI für das Darstellen der letzten 15 Werte im Dashboard des Sensors verwendet.

- **pushExtremeValues** Dieser Service wird für die Darstellung aller Extrem- und Warnwerte in der GUI verwendet. Er liefert alle Werte eines übergebenen Channels eines besteimmten Sensors wieder, welche außerhalb der in der Datenbank definierten Min- und Max-Werte liegen.

- **pushMinMaxAvgForChannel** Dieser Service stellt die in der Datenbank definierten Min- und Max-Werte für einen bestimmten Channel eines Sensors bereit. Zudem wird der Durchschnitt aller Werte des übergebenen Channels des Sensors berechnet und dargestellt.

- **pushSensorCount** Für die dynamische Anzeige der Kacheln wird dieser Service verwendet. Er liefert die Gesamtanzahl aller Sensoren, welche in der Datenbank gespeichert sind.

- **pushSensorTypes** Dieser Service gibt alle verschiedenen Sensortypen, welche in der Datenbank vorhanden sind, aus. Dieser Service wird für die Unterscheidung der einzelnen Sensoren in der GUI verwendet.

- **pushStatisticValues** Dieser Service stellt alle statistischen Werte bereit, die für die Darstellung des initialen Datanbank-Dashboards verwendet werden. Es wird die Gesamtanzahl aller Datenbank-Einträge, die Anzahl aller Sensoren sowie der älteste und aktuellste Datenbankeintrag berechnet und ausgegeben. Zudem erfolgt eine Berechnung der aktuellen Performance der Datenbank. Hierzu werden die neuen Einträge pro Minute, pro Stunde und pro Tag berechnet.

- **pushTableValuesForChannel** Mithilfe dieses Services werden die einzelnen tabellen der Channels der Sensoren dargestellt. Dieser Service stellt alle Werte eines bestimmten Channels eines Sensors bereit. Für eine bessere Performance kann zudem die Anzahl der gewünschten Werte begrenzt werden.

pull

Das Pull-Paket ist ein Unterpaket des Service-Pakets (siehe Abbildung 5.1 auf Seite 97) und enthält alle Objekte und Dateien, die zum zyklischen Abrufen der Druckerwerte benötigt werden (siehe Abbildung 5.15 auf der nächsten Seite). Der erstellte Service „pullPrinterValues“ enthält dabei hauptsächliche Funktionalität des Abrufens und wird im Folgenden detaillierter beschrieben.

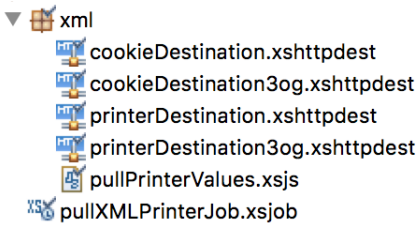


Abbildung 5.15.: Sensorik: Inhalt des pull-Pakets

Die größte Schwierigkeit des Abrufens der Daten bestand in dem Login-Prozess, der durch die Drucker vor jeder Datenabfrage verlangt wird. Um diesen manuellen Prozess des Einloggens zu umgehen, bestand der erste Abschnitt der Abfrage aus dem Abfangen und Speichern des erstellten Cookies der Drucker durch die Applikation. Diese Cookie-Abfrage wurde durch einen WebRequest auf die Index-Seite der Drucker realisiert. Für einen Aufruf durch einen WebRequest durch einen Clienten wird eine „Destination“ Datei benötigt, welche alle Verbindungsinformationen enthält (siehe Listing 5.18).

```
1 host = "172.18.1.12";
2 port = 80;
3 pathPrefix = "/wcd/index.html";
4 proxyType = none;
5 authType = none;
6 useSSL = false;
7 timeout = -1;
```

Listing 5.18: Sensorik: Quellcode - cookieDestination.xshttpest

Die Implementierung einer xshttpdest-Datei folgt dabei immer dem gleichen Muster. Zu Beginn der Datei muss der Host sowie der zu verwendende Port definiert werden. Anschließend muss der Pfad des Hosts zur gewünschten Datei angegeben werden. Optional können Proxy- oder SSL-Einstellungen vorgenommen werden. Zum Abschluss der Destination muss der gewünschte Timeout eingestellt werden. In diesem Fall wurde der Timeout auf „-1“ gesetzt, dadurch wird in jeden Fall auf eine Antwort des Servers gewartet.
Diese erstellte Destination-Datei kann im Service eingelesen werden und bestimmt die Verbindung des Erstellten Requests (vgl. Zeilen 72 & 77 des Listings 5.19)

```
70 try {
71   // Den WebRequest zum Abrufen des Cookies vorbereiten
72   destCookie = $.net.http.readDestination(destination_package,
       cookieDestinations[i]);
73   clientCookie = new $.net.http.Client();
74   reqCookie = new $.web.WebRequest($.net.http.GET, "/");
75
76   // Den Request abschicken und die Anrwort speichern
77   clientCookie.request(reqCookie, destCookie);
78   responseCookie = clientCookie.getResponse();
79
80   // Den Cookie aus dem Header filtern und speichern
81   cookie = responseCookie.headers.get("set-cookie");
```

```
82 } catch (e) {
83   $.trace.warning(e);
84   return;
85 }
```

Listing 5.19: Sensorik: Implementierung der Cookie-Abfrage der pullPrinterValues.xsjs

Anschließend kann der Request gesendet werden und die Antwort des Servers in einer Variable gespeichert werden (Zeile 78). Der zu entnehmende Cookie kann aus dem Header der Response als dem Wertepaar „set-cookie“ entnommen werden. Dieser Cookie wird gespeichert und für den eigentlichen Aufruf verwendet.
Dieser erfolgt nach dem gleichen Muster des Cookie-Aufrufes. Es wurde für jeden Drucker eine Destination erstellt, welche als Pfad den eigentlichen Dateipfad der Daten des Drucker beinhaltet. Bevor der Request abgeschickt wird muss jedoch der Cookie dem Request übergeben werden (siehe Listing 5.20).

```
93   // ENTSCHEIDEND: Hier den Cookie setzen
94   req.cookies.set("Cookie", cookie);
```

Listing 5.20: Sensorik: Implementierung der Drucker-Abfrage der pullPrinterValues.xsjs

Anschließend kann der Body der Response gespeichert werden. Dieser ist eine XML-Datei mit allen Daten des jeweiligen Druckers. Um die für diese Applikation relevanten Daten aus der XML Datei zu extrahieren, musste eine Funktion erstellt werden, welche die übergebene XML-Datei nach den jeweiligen Suchparametern durchsucht und das Ergebnis zurück gibt (siehe Listing 5.21).

```
8  /**
9  * Diese Funktion sucht einen Bestimmten Wert, welcher durch einen
       vorgegebenen Tag umschlossen ist
10 * @param searchTag Der zu suchende Tag
11 * @param xmlString Der zu durchsuchende String
12 * @param iteration Wie oft soll der Tag "uebersprungen" werden
13 * @returns Den gefundenen String zwischen den Tags
14 */
15 function getValue(searchTag, xmlString, iteration){
16   // Der gefundene Wert
17   var value;
18   // Die Start- und Endposition
19   var startPos;
20   var endPos;
21   //Der Start- und End-Tag
22   var startTag = "<"+searchTag+">";
23   var endTag = "</"+searchTag+">";
24
25   var i = 0;
26   var tempString = xmlString;
27   //Zum X Wert durchlaufen
28   while(i<iteration) {
29     endPos = tempString.search(endTag) + endTag.length;
30     tempString = tempString.slice(endPos);
31     i++;
```

```
32    }
33    //Start- und Endpositonen suchen
34    endPos = tempString.search(endTag);
35    startPos = tempString.search(startTag) + startTag.length;
36    // Value aus dem String "schneiden"
37    value = tempString.slice(startPos,endPos);
38    return value;
39  }
```

Listing 5.21: Sensorik: Implementierung der XML-Selektion der pullPrinterValues.xsjs

Dieser Funktion wird der XML-String sowie der Suchtag übergeben und wird beim Vorbereiten des Procedure-Aufrufs ausgeführt (siehe Listing 5.22).

```
125         cst.setString(16, getValue("ScanStatus", xmlString));
```

Listing 5.22: Sensorik: Implementierung des Aufrufs der XML-Abfrage der pullPrinterValues.xsjs

Um die Funktion zyklisch durch die HANA Datenbank ausführen zu lassen, wird ein xsjob benötigt. Dieser ist in der „pullXMLPrinterJob“-Datei implementiert (siehe Listing 5.23).

```
1  {
2    "description": "Pull Printer Values",
3    "action": "abat.demosystem.sensor.sensorapp.services.pull.
       xml:pullPrinterValues.xsjs::pullPrinterValues",
4    "schedules": [
5      {
6        "description": "Alle 10 Sekunden abrufen",
7        "xscron": "* * * * * * */10"
8      }
9    ]
10 }
```

Listing 5.23: Sensorik: Implementierung des pullXMLPrinterJob.xsjob

In dieser Datei wird für jeden erstellten Job eine Beschreibung eingefügt. Außerdem muss der Pfad der auszuführenden Funktion definiert werden. Anschließend muss der Zeitplan in der CRON-Syntax beschrieben werden. Die Aussage „* * * * * * */10“ beschreibt in diesem Beispiel, dass die angegebene Funktion alle 10 Sekunden ausgeführt werden soll.

uicatalog

In dem Paket „uicatalog“ befinden sich eine xsprivileges-Datei sowie eine xswidget-Datei. Beide werden benötigt, um auf der HANA-Administrationsoberfläche eine dynamische Kachel (siehe Abbildung 5.16 auf der nächsten Seite) darzustellen.

Abbildung 5.16.: Sensorik: Darstellung einer dynamischen Kachel auf der HANA-Administrationsoberfläche

Um diese Kachel anzeigen zu könne, benötigt der Nutzer die entsprechenden Rechte für den verwendeten Service sowie für die eigentliche Kachel. Dieses Kachel-Privileg wird in der .xsprivileges-Datei definierty (siehe Listing 5.24).

```
{
  "privileges": [
      {
      "name": "SensorAppLauncher",
      "description": "Access SensorAppLauncher Widget"
      }
    ]
}
```

Listing 5.24: Sensorik: Implementierung der .xsprivileges für Kacheln

Dieses Privileg wird anschließend in der entsprechenden Rolle des Nutzers hinterlegt (vergleiche Kapitel “data“ Listing 5.5 auf Seite 104).
Um die erstellte Kachel zu konfigurieren wurde der grafische Editor des HANA-Studios verwendet. Zuerst mussten die allgemeinen Einstellungen der Kachel übernommen werden (siehe Abbildung 5.17).

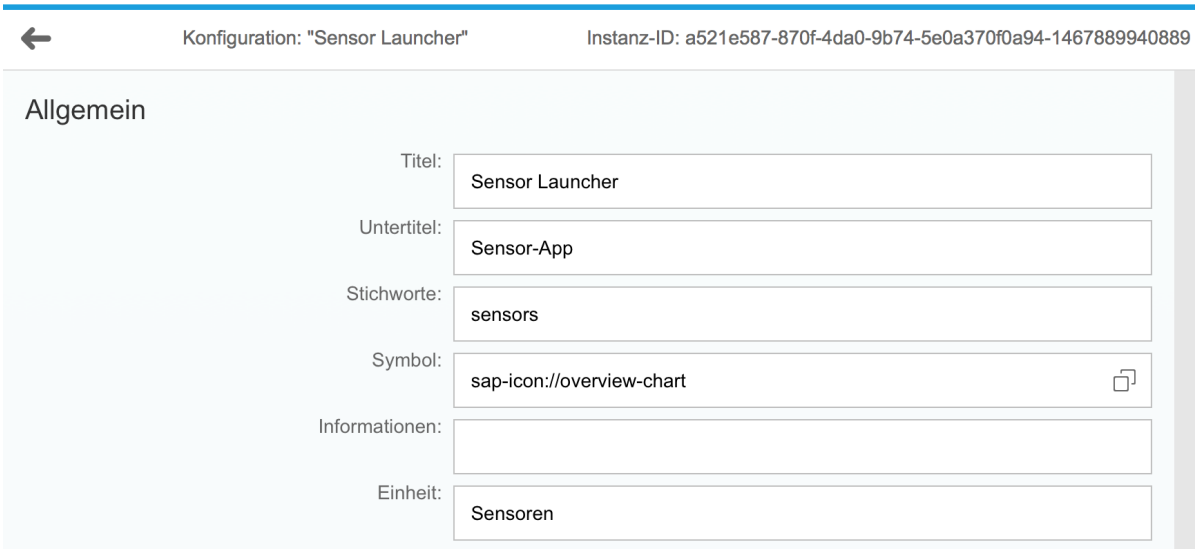


Abbildung 5.17.: Sensorik: Darstellung des grafischen Editors zur Erstellung einer Kachel (Allgemeine Einstellungen)

In dieser Maske werden die sichtbaren Standardeinstellungen der Kachel vorgenommen. Hierzu gehören unter Anderem Titel, Einheit, Stichworte und Icon.
Anschließend müssen die dynamischen Einstellungen vorgenommen werden. Bei dieser Kachel wurde der jeweilige Service für die Anzahl der Sensoren als „Service-URL“ übergeben. Zusätzlich wurde ein Aktualisierungsintervall von 10 Sekunden bestimmt (siehe Abbildung 5.18 auf der nächsten Seite). Abschließend muss die Ziel-URL der Kachel angeben werden. Hier wurde die URL zur grafischen Oberfläche der Applikation verwendet.

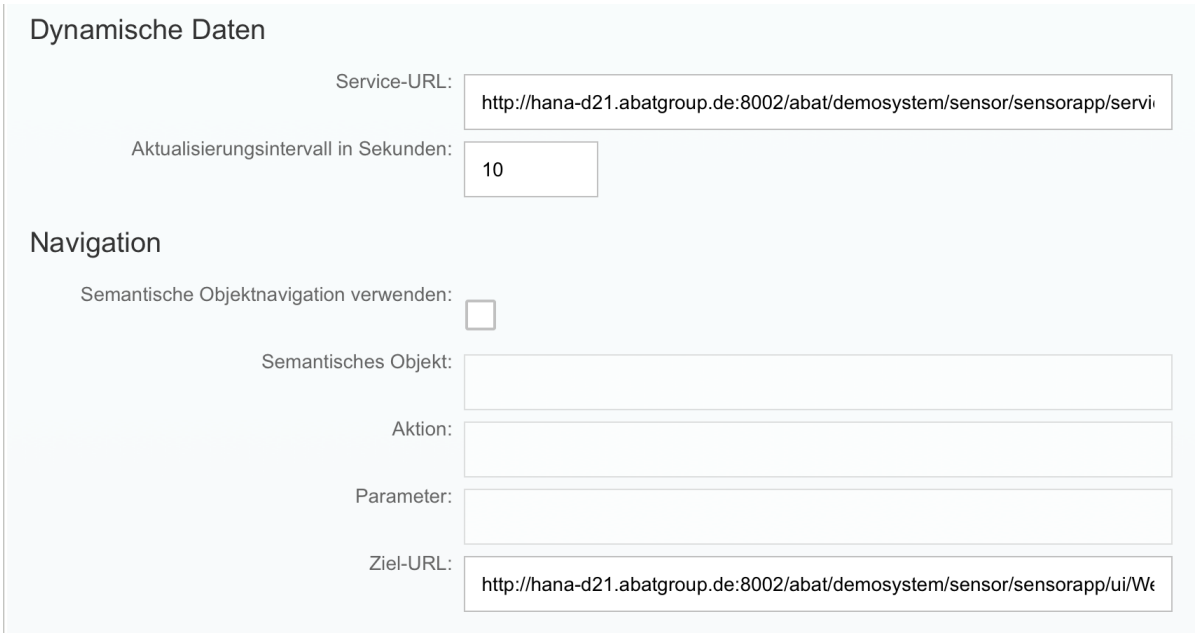
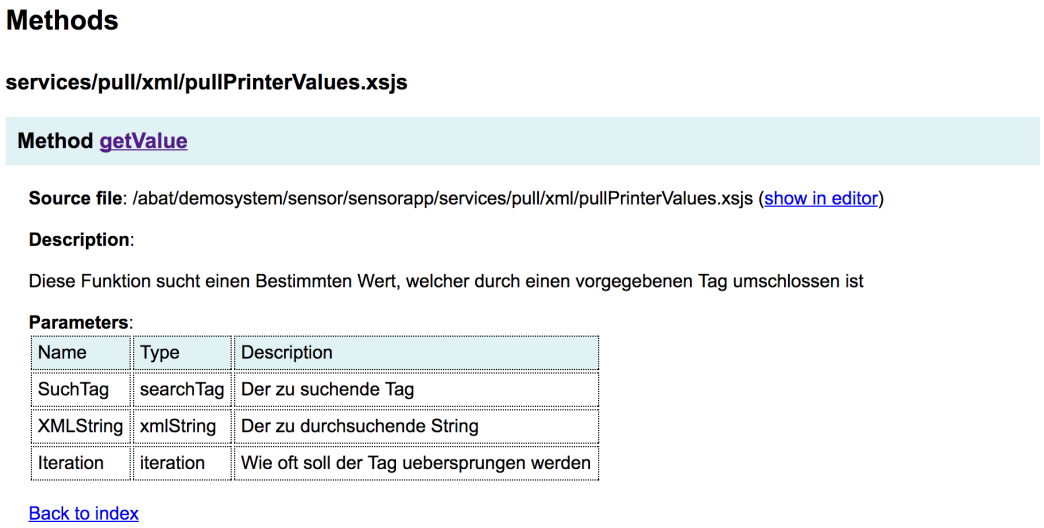


Abbildung 5.18.: Sensorik: Darstellung des grafischen Editors zur Erstellung einer Kachel (Dynamische Einstellungen)

doc

Das doc-Paket enthält eine index.html Datei, welche automatisch durch das HANA-Studio generiert wurde und eine automatisch JSDokomentation enthält. Diese HTML-Datei ist navigierbar und enthält alle Funktionen sowie deren implementierten Kommentare (siehe Abbildung 5.19).
Dieses Paket dient hauptsächlich der Übersichtlichkeit für Entwickler und dem schnellen Suchen von Funktionen und ihren Parametern.



Methods

services/pull/xml/pullPrinterValues.xsjs

Method getValue

Source file: /abat/demosystem/sensor/sensorapp/services/pull/xml/pullPrinterValues.xsjs (show in editor)

Description:

Diese Funktion sucht einen Bestimmten Wert, welcher durch einen vorgegebenen Tag umschlossen ist

Parameters:

Name	Type	Description
SuchTag	searchTag	Der zu suchende Tag
XMLString	xmlString	Der zu durchsuchende String
Iteration	iteration	Wie oft soll der Tag uebersprungen werden

Back to index

Abbildung 5.19.: Sensorik: Teildarstellung der Index.html der JSDoc

5.3.3. Raspberry Pi

Die Java-Applikation des Raspberry Pi dient dem Senden und Auslesen der Sensordaten. Die Applikation enthält vier logisch voneinander getrennte Klassen (siehe Abbildung 5.20), die im folgenden detaillierter beschrieben werden.

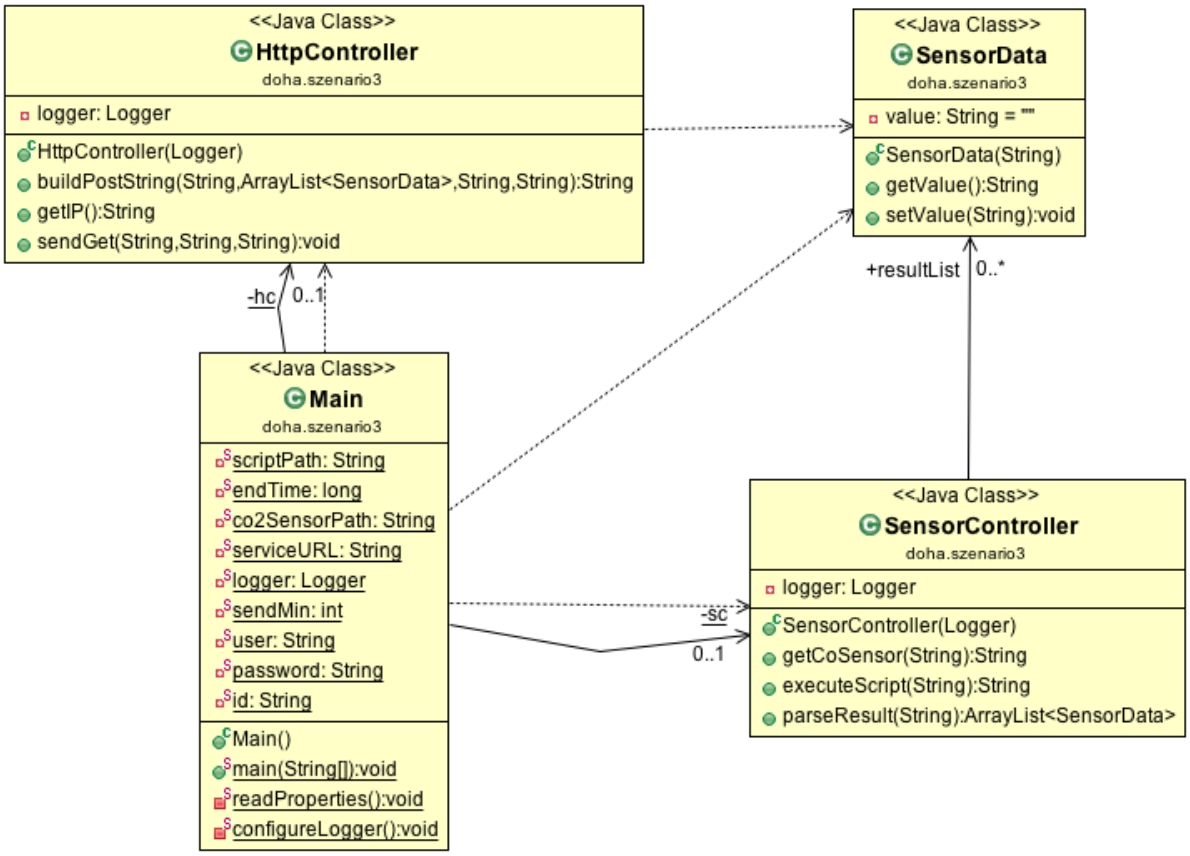


Abbildung 5.20.: Sensorik: Architektur der JAVA-Applikation der Raspberry Pis

HTTPController

Die „HTTPController“-Klasse enthält die Funktionalität zum senden der bisher ausgelesenen Daten an die HANA-Datenbank. Hierfür sind zwei Methoden elementar. Die „buildPostString“ Methode generiert die aufzurufende URL. Die zu übergebenen Werte werden an die URL angeknüpft (vgl. Listing 5.25).

```
30 public String buildPostString(String serviceURL, ArrayList<SensorData>
       resultList, String CO2, String id) {
31   String postUrl = serviceURL;
32   postUrl = postUrl + "?id=" + getIP();
33
34   int i = 0;
35   int j = 1;
36   while( i < resultList.size()) {
37     if(j == 3) {
```

```
38          postUrl = postUrl + "&value" + j + "=" + CO2;
39          j++;
40        } else {
41          postUrl = postUrl + "&value" + j + "=" + resultList.get(i).getValue
          ();
42          i++;
43          j++;
44        }
45      }
46      while (i<=18) {
47        postUrl = postUrl + "&value" + j + "=null";
48        i++;
49        j++;
50      }
51      return postUrl;
52    }
```

Listing 5.25: Sensorik: Implementierung der HTTPController.java (URL)

Auch hier hat der CO2-Wert eine gesonderte Rolle, da dieser nicht über das erstellte Script abgefragt wird.
Die Methode „sendGet“ ruft die zuvor erstellte URL auf. Wichtig zu beachten ist hierbei, dass sich die Applikation mit dem Aufruf der URL gleichzeitig authentifizieren muss. Dies wurde implementiert, um nicht gewünschte Zugriffe auf die Datenbank zu verhindern. Diese Authentifizierung gelingt durch eine, für die Raspberry Pis erstellte, User-Passwort Kombination. Diese muss als RequestPrperty der zuvor aufgebauten Connection übergeben werden (siehe Listing 5.26 Zeile 98 f.).

```
89  public void sendGet(String user, String password, String postUrl) {
90    //Sending the Request
91    URL obj;
92    try {
93      obj = new URL(postUrl);
94      HttpURLConnection con = (HttpURLConnection) obj.openConnection();
95      con.setRequestMethod("GET");
96
97      String userpass = user + ":" + password;
98      String basicAuth = "Basic " + new String(new Base64().encode(userpass.
        getBytes()));
99      con.setRequestProperty ("Authorization", basicAuth);
100
101     int responseCode = con.getResponseCode();
102     if(responseCode == 200){
103       //logger.info("Send Data: " + responseCode);
104     }else{
105       logger.warning("Error: " + responseCode + " - postURL: " + postUrl +
        " - " + con.getResponseMessage());
106     }
107   } catch (MalformedURLException e) {
108     logger.warning(e.getMessage() + " - " + e.getStackTrace().toString());
109   } catch (ProtocolException e) {
110     logger.warning(e.getMessage() + " - " + e.getStackTrace().toString());
```

```
111   } catch (IOException e) {
112     logger.warning(e.getMessage() + " - " + e.getStackTrace().toString());
113   }
114 }
```

Listing 5.26: Sensorik: Implementierung der HTTPController.java (SendGet)

Anschließend kann der ResponseCode der Connection abgerufen werden (Zeile 101). Sollten bei dem Aufruf Fehlercodes entstehen, so werden diese Abgefangen und in der Log-Datei gespeichert. Der Aufruf zum Senden der Fehlermeldungen an die HANA-Datenbank erfolgt analog zu dem oben beschriebenen Aufruf. Hier ändert sich lediglich die Aufruf-URL.

Main

Die Main-Klasse initiiert die Applikation und startet die gesamte Logik zum Senden der Daten an die HANA-Datenbank. Hierzu enthält diese Klasse insgesamt drei Funktionen. Die Funktion „configureLogger“ konfiguriert einen Logger, welcher zum Speichern von Warnungen, Fehlermeldungen und Konsolenausgaben benötigt wird. Dieser Logger erstellt ein Logging-File, welches im Verlauf der Applikation immer weiter beschrieben wird (siehe Listing 5.27).

```
141 private static void configureLogger(){
142   //Initialisiere den Logger
143   logger = Logger.getLogger("DoHA");
144   logger.setUseParentHandlers(false);
145   FileHandler fileHandler;
146   try {
147     // Den Logger mit dem Formatter und dem handler konfigurieren
148     fileHandler = new FileHandler("DoHALogFile.log");
149     logger.addHandler(fileHandler);
150     SimpleFormatter formatter = new SimpleFormatter();
151     fileHandler.setFormatter(formatter);
152   } catch (SecurityException e) {
153     logger.warning(e.getStackTrace().toString());
154   } catch (IOException e) {
155     logger.warning(e.getStackTrace().toString());
156   }
157 }
```

Listing 5.27: Sensorik: Implementierung der Main.java (Logger)

Als Format des Loggers wurde sich für ein „SimpleFormat“ entschieden. Um Fehler in die Log-Datei zu schreiben wird der Ausdruck “‘ogger.warning“ (siehe Zeile 16) verwendet. Die Funktion „readProperties“ liest die angelegte Properties-Datei aus und speichert die enthaltenen Elemente in Variablen (siehe Listing 5.28). Der Pfad zur Properties-Datei ist bei Applikationsstart als Parameter mit anzugeben.

```
102 private static void readProperties(){
103   FileReader reader = null;
104   try {
```

```
105       reader = new FileReader(props + "doha.properties");
106     } catch (FileNotFoundException e) {
107       logger.warning(e.getStackTrace().toString());
108     }
109     if (reader != null) {
110       Properties properties = new Properties();
111       try {
112         properties.load(reader);
113         scriptPath = properties.getProperty("scriptPath");
114         co2SensorPath = properties.getProperty("co2SensorPath");
115         serviceURL = properties.getProperty("serviceURL");
116         errorServiceURL = properties.getProperty("errorServiceURL");
117         String min = properties.getProperty("sendMin");
118         if (min.equals("")){
119           Date max = new Date(Long.MAX_VALUE);
120           sendMin = max.getTime();
121           endTime = sendMin;
122         } else {
123           sendMin = (Long.valueOf(min) * 60000);
124           endTime = System.currentTimeMillis() + (sendMin);
125         }
126         user = properties.getProperty("user");
127         password = properties.getProperty("password");
128         id = properties.getProperty("id");
129         if (id.equals("")) {
130           id = hc.getIP();
131         }
132       } catch (IOException e) {
133         logger.warning(e.getStackTrace().toString());
134       }
135     }
136 }
```

Listing 5.28: Sensorik: Implementierung der Main.java (Properties)

Diese Eigenschaften sind für die weitere Logik entscheidend. Bei falschen oder fehlenden Angaben der Properties können Fehler oder ggf. Ausfälle der Applikation entstehen. Die Main-Methode initialisiert die Applikation und ruft die oben beschriebenen Methoden der Properties und der Logger auf. Zudem wird eine Schleife gestartet, die die Sensorwerte durch die anderen Klassen in regelmäßigen Abständen abfragt und an die HANA-Datenbank schickt (siehe Listing 5.29).

```
62 public static void main( String[] args )
63 {
64   if(args[0] != null) {
65     props = args[0];
66   } else {
67     props = "";
68   }
69   configureLogger();
70   logger.info("System started.");
71
72   sc = new SensorController(logger);
```

```
73    hc = new HttpController(logger);
74    readProperties();
75
76    while(endTime > System.currentTimeMillis())
77    {
78      ArrayList<SensorData> resultList;
79      try {
80        resultList = sc.parseResult(sc.executeScript(scriptPath));
81        if(!resultList.get(0).getValue().equals("0")) {
82          String postString = hc.buildPostString(serviceURL, resultList, sc.
      getCoSensor(co2SensorPath), id);
83          hc.sendGet(user, password, postString);
84        }
85        sc.resultList.clear();
86      } catch (Exception e1) {
87        logger.warning("Sensorwarning: Failed to get reading. Try again!");
88      }
89      try {
90        Thread.sleep(10000);
91      } catch(InterruptedException e){
92        logger.warning(e.getStackTrace().toString());
93      }
94    }
95    logger.info("System finished.");
96  }
```

Listing 5.29: Sensorik: Implementierung der Main.java (Main)

SensorController

Die „SensorController“ Klasse enthält die Funktionalität zum Abrufen der verschiedenen Sensorwerte. Insgesamt beinhaltet die Klasse drei Methoden zum Abfragen und Speichern der Sensorwerte. Die Methode „getCoSensor“ liest eine Datei ein, welche den aktuellen Wert des CO2-Sensors enthält (siehe Listing 5.30).

```
27  public String getCoSensor(String co2SensorPath) {
28    FileReader fileReader = null;
29    try {fileReader = new FileReader(co2SensorPath);
30    } catch (FileNotFoundException e) {
31      logger.warning(e.getMessage());
32    }
33    BufferedReader br = new BufferedReader(fileReader);
34    String co2Value = null;
35    try {co2Value = br.readLine();
36    } catch (IOException e) {
37      logger.warning(e.getMessage());
38    }
39    return co2Value;
40  }
```

Listing 5.30: Sensorik: Implementierung der SensorController.java (Co2 Sensor)

Da dieser Sensor ebenfalls von einem anderen Projekt verwendet wurde, konnten dieser nicht direkt abgefragt werden. Bei häufigen synchronen Abfragen entstanden Fehlerwerte, welche durch den Umweg über eine zentrale Datei reduziert wurden. Der Wert des Sensors wird als String zurückgegeben.
Die Methode „executeScript“ führt das unten beschriebene Script (siehe Listing 5.33 auf der nächsten Seite) aus und gibt die Werte als String aus.

```
47 public String executeScript(String scriptPath) {
48   Process pro = null;
49   Process p = null;
50   try {
51     //Die Rechte ueberpruefen
52     pro = Runtime.getRuntime().exec("chmod 777 " + scriptPath);
53     pro.waitFor();
54     //Das Script ausfuehren
55     p = Runtime.getRuntime().exec(scriptPath);
56     p.waitFor();
57   } catch (IOException e) {
58     logger.warning(e.getStackTrace().toString());
59   } catch (InterruptedException e) {
60     logger.warning(e.getStackTrace().toString());
61   }
62   //Der Output muss als String gespeichert werden
63   BufferedReader reader =
64   new BufferedReader(new InputStreamReader(p.getInputStream()));
65
66   StringBuilder builder = new StringBuilder();
67   String line = null;
68   try {
69     while ( (line = reader.readLine()) != null) {
70       builder.append(line);
71     }
72   } catch (IOException e) {
73     logger.warning(e.getStackTrace().toString());
74   }
75   return builder.toString();
76 }
```

Listing 5.31: Sensorik: Implementierung der SensorController.java (Script)

Besonders wichtig beim Ausführen des Scripts ist das Prüfen der Rechte (Zeile 6) sowie das Auslesen des Outputs (Zeilen 17 ff.).
Der Output wird anschließend in der Funktion „parseResult“ geparst. Der String-Output wird hierbei in SensorData-Objekte gewandelt. Diese werden als Ergebnis dieser Funktion als ArrayListe für die weitere Verarbeitung zurückgegeben.

SensorData

Die „SensorData“ Klasse repräsentiert ein Datenobjekt und ist im Sinne der objektorientierten Programmierung Datenträger für einzelne Sensorwerte. In diesem Objekt kann genau ein Wert („value“) gespeichert werden (siehe Listing 5.32 Zeile 5). Dieser Wert

kann durch die Hilfsmethoden „getValue“ und „setValue“ manipuliert und ausgegeben werden (Zeilen 12 ff.).

```
1  package doha.szenario3;
2
3  public class SensorData {
4    //Das Value
5    private String value;
6
7    //Eine Methode zum initialisieren eines neues Wertes
8    public SensorData(String value) {
9      this.value = value;
10   }
11
12   public String getValue() {
13     return value;
14   }
15
16   public void setValue(String value) {
17     this.value = value;
18   }
19 }
```

Listing 5.32: Sensorik: Implementierung der SensorsData.java

Python-Script

Das in der Java-Applikation verwendete Script ist in Python geschrieben und ruft die verschiedenen Scripte der einzelnen Sensoren auf. Es werden hierbei die Luftfeuchtigkeit, der Luftdruck und die Temperatur zurück gegeben (siehe Listing 5.33).

```
1  #!/bin/bash
2  #
3  SCRIPT1="python /usr/local/airpimon/bmp180/Adafruit-Raspberry-Pi-Python-
       Code/Ad$
4  SCRIPT2="/usr/local/airpimon/Adafruit_Python_DHT/examples/hum.py 11 4"
5  SCRIPT3="/usr/local/airsensor/airsensor -v"
6  SCRIPT4="python /usr/local/airpimon/bmp180/Adafruit-Raspberry-Pi-Python-
       Code/Ad$
7  TEMPRATURE=`$SCRIPT1`
8  HUMIDITY=`$SCRIPT2`
9  #CO2=`$SCRIPT3`
10 PRESSURE=`$SCRIPT4`
11
12 echo $TEMPRATURE";"$HUMIDITY";"$PRESSURE
```

Listing 5.33: Sensorik: Implementierung des studenten.sh Scriptes

5.3.4. Grafische Oberfläche

Dieser Abschnitt dokumentiert den Aufbau der grafischen Oberfläche. Beginnend mit der Beschreibung der inhaltlichen Struktur des User Inerface (UI)-Pakets und der genutzten Ressourcen, werden darauf folgend alle funktionalen und nicht-funktionalen Bausteine schrittweise beschrieben, mit Screenshots zur besseren Darstellung verdeutlicht und wichtige Codefragmente erläutert.

Paketstruktur

Alle für die Darstellung der grafischen Oberfläche notwendigen Ressourcen und Dateien befinden sich im Paket *abat/demosystem/sensor/sensorapp/ui*. Innerhalb dieses Pakets stellt ein initial angelegtes SAPUI5-Projekt den Grundbaustein der Anzeige der Anwendung und somit die grafische Schnittstelle zum Anwender dar. Durch Anlegen eines solchen Projekts werden alle Bausteine im Paket *WebContent* gegliedert. Somit ergibt sich, wie auf Abbildung 5.21 zu erkennen ist, folgender Projektpfad innerhalb der Gesamtanwendung: *abat/demosystem/sensor/sensorapp/ui/WebContent*.

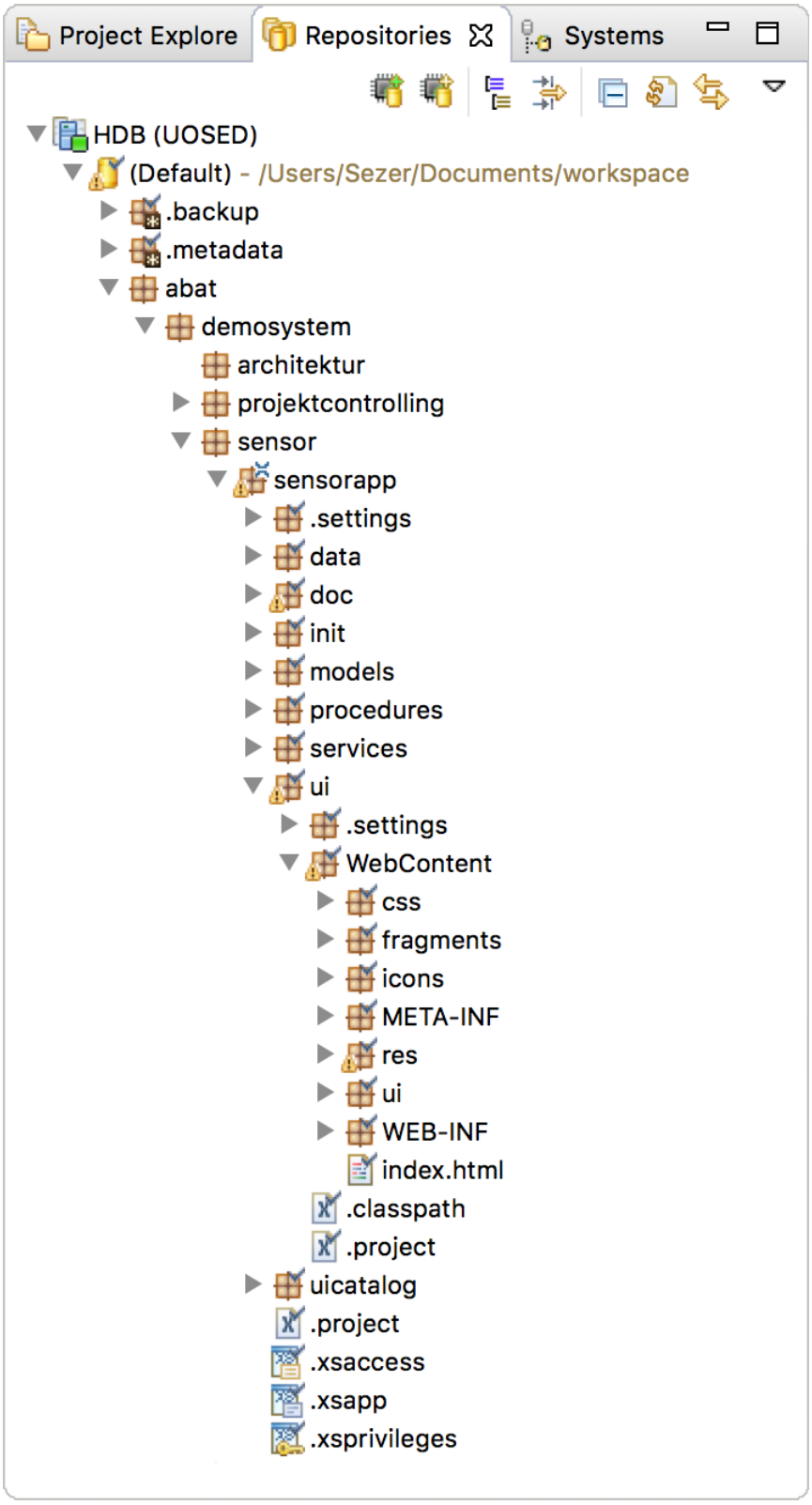


Abbildung 5.21.: Sensorik: Projektpfad zu den UI-Ressourcen

css Das SAPUI5-Framework bietet die Möglichkeit seinen grundlegenden Gestaltungselementen, den Controls, eigene Cascading Style Sheets (CSS)-Klassen hinzuzufügen. Um diese Funktionalität nutzen und umsetzten zu können, erfordert es eigene CSS-Stylesheets anzulegen und diese in die Anwendung einzubinden. Im CSS-Paket (siehe Abbildung 5.22) sind im Rahmen dieser Anwendung zum einen eigene Fonts definiert, die anwendungsspezifische Icons erzeugen, und zum anderen beinhaltet die styles.css-Datei jegliche, speziell für die Anwendung angepassten Gestaltungsoptionen des Sensorik Szenarios.
Die *addSytleClass()*-Funktion fügt CSS-Klassen, im Document Object Model (DOM)-Baum, an die vom Framework vorgesehene Stelle ein. Im Rahmen dieser Anwendung haben sich diese Bereiche im DOM-Baum jedoch als nicht sinnvoll erwiesen, da Ebenen darüber oder auch darunter angesprochen werden müssen. Alle Selektoren innerhalb des Stylesheets sind aus dem DOM-Baum über die Entwicklertools des Browsers bezogen worden.

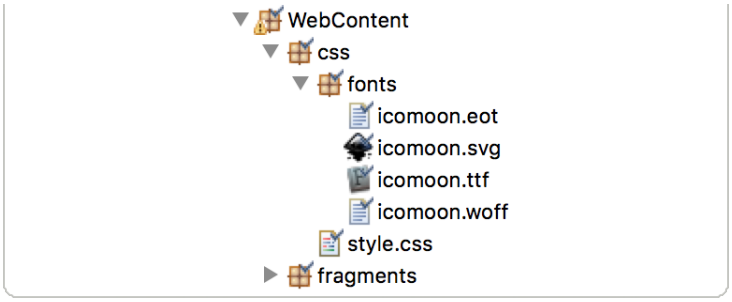


Abbildung 5.22.: Sensorik: Inhalt des css-Pakets

fragments Das fragments-Paket (siehe Abbildung 5.23) beinhaltet Codefragmente, welche innerhalb der initialen Grunddarstellungsdatei (siehe Abbildung 5.27) als zusätzliche Darstellungsobjekte genutzt werden. Fragmente agieren nicht als eigenständige Anzeigen, sondern vielmehr als kleinere, von mehreren Views nutzbare View-Objekte, die Popup-Fenster oder Dialoge realisieren können. Für die Anwendung werden zwei Fragmente, genutzt die innerhalb der tableSettingsDialog.fragments.js-Datei und tilesSettingPopover.fragment.js-Datei definiert werden. Wenn ein fragments-Package angelegt wird, muss dieser als lokale Ressource in der index.html-Datei (siehe Abschnitt index.html) angegeben werden. Ansonsten sind sie nicht nutzbar.

- **tableSettingsDialog.fragments.js**
  In diesem Fragment ist eine View definiert, die im Info-Bereich des ObjectHeaders (siehe Abschnitt oDetailPage) genutzt wird. Dieses ermöglicht das Sortieren und Gruppieren von Ausnahmewerten, die sich in der unteren Tabelle befinden. Das Fragment findet nur innerhalb des Info-Bereichs Anwendung.

- **tilesSettingPopover.fragment.js**
  In diesem Fragment ist eine View definiert, die zur Geschwindigkeitsregulierung

beim Übersichts-Dashboard der Raspberry Pis (siehe Abschnitt oRaspberryStatistics) genutzt wird. Sie beinhaltet ein Slider-Control und findet lediglich innerhalb des Übersichts-Dashboards Anwendung.

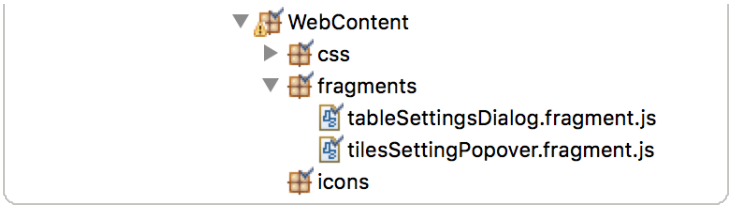


Abbildung 5.23.: Sensorik: Inhalt des fragments-Pakets

icons In diesem Paket befinden sich keine Inhalte (siehe Abbildung 5.24). Falls, wie im Abschnitt css erwähnt, keine Icons manuell über Cascading Style Sheets definiert werden, können diese in diesem Paket hinterlegt werden. Dieses Paket wurde der Vollständigkeit halber für eine mögliche zukünftige Nutzung angelegt.

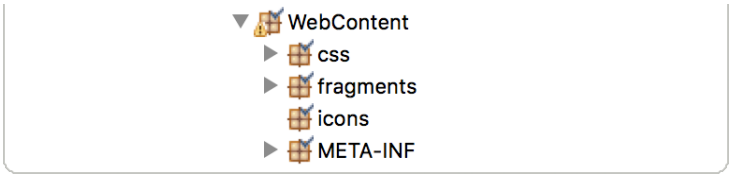


Abbildung 5.24.: Sensorik: Inhalt des icon-Pakets

META-INF In der MANIFEST.MF-Datei innerhalb des META-INF Pakets (siehe Abbildung 5.25) können weitere Angaben, wie z. B. der Klassenpfad, angegeben werden. Diese Datei verfügt im Rahmen dieses Szenarios über keine anwendungsspezifischen Angaben.

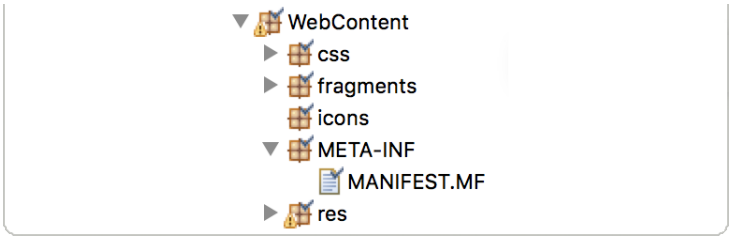


Abbildung 5.25.: Sensorik: Inhalt des META-INF-Pakets

res Das res-Paket (siehe Abbildung 5.26) beinhaltet alle anderen, für die Anwendung des Szenarios notwendigen, Ressourcen, die nicht in den zuvor beschrieben Paketen logisch hinterlegt werden können. Innerhalb des Pakets befindet sich das printer_properties-Paket, dass die möglichen Statuswerte der Druckergeräte (siehe Abschnitt oPrinterPage)

beinhaltet, die in einer xml-Datei beschrieben sind. Aus dieser Datei werden die jeweiligen Informationen der unterschiedlichen Druckzustände bezogen. Des Weiteren ist eine Bilddatei hinterlegt, welche für die Darstellung der Logos genutzt wird.

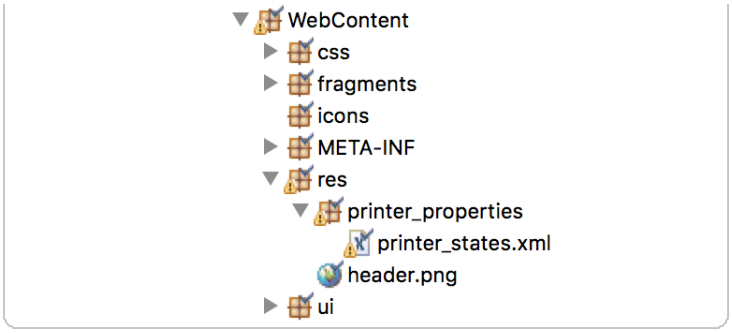


Abbildung 5.26.: Sensorik: Inhalt des res-Pakets

ui Die elementarsten Bestandteile, damit die grafische Schnittstelle zum Anwender überhaupt aufgebaut und gesteuert werden kann, befinden sich im ui-Paket (siehe Abbildung 5.27). Dabei wird die Anzeige selbst durch die initial.view.js-Datei repräsentiert. Die dazugehörige notwendige Logik wird im Controller innerhalb der initial.controller.js-Datei realisiert. Alle implementierten Funktionen befinden sich im Controller. Das SAP-UI5-Framework gibt vor, dass der Controller stets, anders als die View, mit der Skriptsprache JavaScript implementiert werden muss. Bei der View ergeben sich zudem noch die Möglichkeiten neben JavaScript auch eine Extensible Markup Language (XML), Hypertext Markup Language (HTML) oder eine Anzeige in der JavaScript Object Notation anzulegen. Im Rahmen dieser Anwendung des Szenarios ist die Implementierung der grafischen Oberfläche mit JavaScript erfolgt. Dadurch konnte der Fokus auf nur eine Implementierungssprache gelegt und eine Einheitlichkeit zwischen der View und dem Controller erreicht werden.

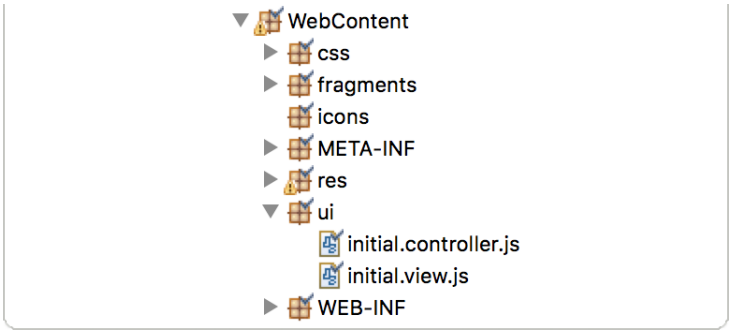


Abbildung 5.27.: Sensorik: Inhalt des ui-Pakets

Im Laufe der folgenden Abschnitte wird die auf Abbildung 5.27 dargestellte View sowie der Controller schrittweise dokumentiert. Die grafische Oberfläche, die mithilfe von Screenshots dokumentiert ist, wird an sensiblen Bereichen, das genauerer Betrachtung Bedarf, anhand von Quellcode erläutert.

WEB-INF Die web.xml-Datei innerhalb des WEB-INF Pakets (siehe Abbilsung 5.28) legt grundlegende Einstellungen für die Ressourcenverwaltung fest. Wenn die Anwendung lokal getestet werden soll, bei dem auf Advanced Business Application Programming (ABAP)-Systeme oder Open Data Protocol (OData)-Dienste zugegriffen wird, stellt sie zudem ein einfaches Proxy Servlet zur Verfügung.

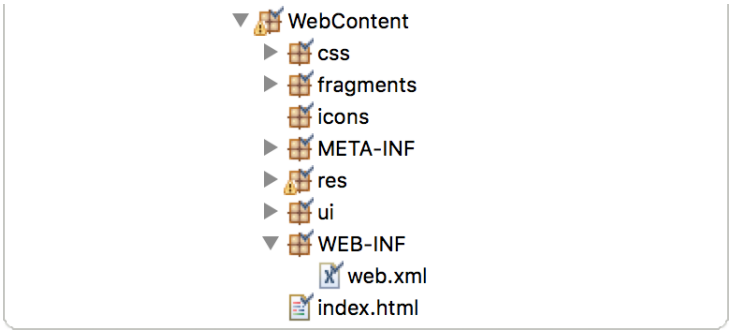


Abbildung 5.28.: Sensorik: Inhalt des WEB-INF-Pakets

index.html Das Rendern web-basierter Anwendungen erfordert den Aufruf einer initialen Datei, die grundlegende web-spezifische Informationen zum Aufbau der jeweiligen Anwendung definiert. Da es sich bei SAPUI5 um ein web-basiertes Framework handelt, ist zum Ausführen der Anwendung solch eine Datei unerlässlich und wird im Rahmen dieses Szenarios durch die index.html-Datei (siehe Abbilsung 5.29) repräsentiert.

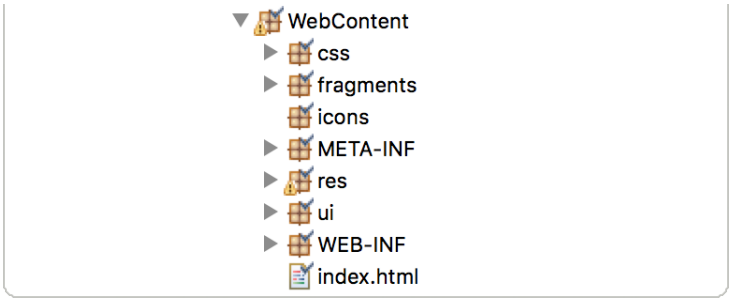


Abbildung 5.29.: Sensorik: Inhalt der index.html-Datei

Neben HTML-spezifischen Auszeichnungen und Meta-Informationen, gliedert sich der Inhalt (siehe Listing 5.34) in den Bootstrap-, Anwendungs- und UI-Bereich. Die Ressourcen der SAPUI5-Bibliotheken befinden sich nicht auf der Datenbank. Betrachtet man den Bootstrap-Bereich in Listing 5.34, so ist zu erkennen, dass die Ressourcen über einen *Content Delivery Network* (https://sapui5.hana.ondemand.com/resources/sap-ui-core.js) eingebunden werden. Dadurch ergibt sich der Vorteil, dass jederzeit alle grafischen Elemente auf dem aktuellen Stand sind. Das setzt jedoch voraus, dass bei Verwendung der Anwendung stets eine aktive Internetverbindung vorhanden sein muss.
Es wurde darauf geachtet, dass die Applikation im Rahmen des Szenarios so gestaltet ist, dass ein Verwenden auf unterschiedlichen Endgeräten mit verschiedenen Displaygrößen

möglich sein soll. Es ist zu erwähnen, dass viele, aber lange nicht alle Darstellungselemente zur Verfügung stehen, welche das Responsive-Design gut umsetzten. Diese Anwendung kombiniert mobile und nicht-mobile Elemente und setzt sie in einigen Bereichen gut und an anderen Bereichen weniger gut um. Aus Performance-technischen gründen sind im Bootstrap-Bereich nur diejenigen Bibliotheken eingebunden worden, die auch wirklich benötigt werden:

- **sap.m**
  Beinhaltet Controls, die sich den Displaygrößen mobiler Endgeräte anpassen. Einige Elemente (z. B. sap.m.ObjectHeader) bieten zusätzlich eine setResponsive()-Funktion, mit der die Eigenschaft explizit gesetzt werden kann. Unterschiede bei der nicht-Setzung wurden keine festgestellt.

- **sap.viz**
  Beinhaltet Controls zur grafischen Darstellungen von Zahlen die im weiteren Verlauf als „Charts“ bezeichnet werden.

- **sap.suite.ui.commons**
  Bietet Möglichkeiten Charts in Containern zu platzieren.

- **sap.ui.layout**
  Ermöglicht das Verwenden von verschiedenen Layouts (z. B. GridLayout).

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta http-equiv='Content-Type' content='text/html;charset=UTF-8'/>

    <!--Bootstrap-->
    <script src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js"
        id="sap-ui-bootstrap"
        data-sap-ui-libs="sap.m, sap.viz, sap.suite.ui.commons, sap.ui.layout, sap.ui.commons"
        data-sap-ui-theme="sap_bluecrystal">
    </script>

    <!--Anwendungsbereich-->
    <script>
        sap.ui.localResources("ui");
        sap.ui.localResources("fragments");
        var app = new sap.m.App({initialPage:"idinitial1"});
        var page = sap.ui.view({id:"idinitial1", viewName:"ui.initial", type:sap.ui.core.mvc.ViewType.JS});
        app.addPage(page);
        app.placeAt("content");
    </script>

```

```
24      <!--UI-Bereich-->
25    </head>
26    <body class="sapUiBody" role="application">
27      <div id="content"></div>
28    </body>
29  </html>
```

Listing 5.34: Sensorik: Implementierung der index.html

Im Anwendungsbereich werden lokale Ressourcen angegeben. Initial ist immer das ui-Package definiert, das die View und den Controller beinhaltet. Zusätzlich wird das fragments-Package eingebunden, die Start-View festgelegt und im UI-Bereich platziert. Der UI-Bereich stellt den tatsächlich für den Endanwender sichtbaren Bereich dar.

Grundgerüst

Das Grundgerüst teilt sich in zwei zentrale Bereiche unter Berücksichtigung der Anforderung 5.3.3 im Abschnitt 5.2.5 ein, eine Master-/Detailstruktur zu implementieren. Dabei umfasst die Aufgabe der Masteransicht zum einen das Navigieren innerhalb der Anwendung zu ermöglichen und zum anderen unterschiedliche Sensoren und ihre jeweiligen Komponenten anzuzeigen und auszuwählen. Auf Abbildung 5.30 sieht man das diese Sicht eine gewisse Breite auf der Linken Seite einnimmt.

Abbildung 5.30.: Sensorik: Grundgerüst der GUI (ausgeführt)

Die Detailansicht wiederum befindet sich auf der rechts liegenden Seite. Die maßgebliche Interaktion zum Wechsel der angezeigten Informationen erfolgt beim Navigieren auf

der linken Seite. Technisch realisiert wird diese Struktur durch das SplitApp-Control. Die unterschiedlichen Ebenen beider Bereiche werden durch verschiedenste Elemente realisiert, die dann dem SplitApp-Control hinzugefügt worden sind. Die Tabelle 5.9 listet eine grobe Übersicht über die obersten Bestandteile auf, die sich in diesem Control befinden. Das SplitApp-Control in Listing 5.35 bildet dabei stets das oberste Control, in dem alle anderen Controls hinzugefügt werden.

```
979  /**
980   * Erzeugen eines SplittApp-Controls. Grundgeruest der Anwendung welches
       ermoeglicht die Darstellung in der Master-/DetailAnsicht zu
      realisieren.
981   */
982    var oSplitApp = new sap.m.SplitApp("splitApp1",{
983      showNavButton: true,
984    });
```

Listing 5.35: Sensorik: Erzeugen des SplitApp-Controls als oberste Komponente der View

Tabelle 5.9.: Sensorik: Ebenen innerhalb der Master- und Detailstruktur

Paket:	abat/demosystem/sensor/sensorapp/ui/WebContent/ui		
Datei:	initial.view.js		
	Variable:	**Control:**	**Zweck:**
Masteransicht:	oMasterCategoryPage	sap.m.Page	Zur Darstellung der Sensorkategorien
	oMasterSensorsList	sap.m.Page	Zur Darstellung aller Sensoren der jeweiligen Sensorkategorie
	oMasterChannelList	sap.m.Page	Zur Darstellung aller Messensensoren die an einem Sensor angeschlossen sind. Betrifft nicht die Drucker
Detailansicht:	oDashboard	sap.m.Page	Zur Darstellung eines Dashboards (Charts) beim Auswählen eines Raspberry Pis. Betrifft nicht die Drucker

Tabelle 5.9.: Sensorik: Ebenen innerhalb der Master- und Detailstruktur

Paket:	abat/demosystem/sensor/sensorapp/ui/WebContent/ui		
Datei:	initial.view.js		
	oDetailPage	sap.m.Page	Zur Darstellung einer Detailansicht beim Auswählen eines Messsensors. Betrifft nicht die Drucker
	oInitialPage	sap.m.Page	Zur Darstellung einer Kachelansicht der Datenbankstatistiken
	oPrinterPage	sap.m.Page	Zur Darstellung von Druckerinformationen. Betrifft nicht die Raspberry Pis
	oRaspberryStatistics	sap.m.Page	Zur Darstellung einer Übersicht über alle Raspberry Pis in der Kachelansicht deren Inhalt sich wechselt. Betrifft nicht die Drucker

Die konkrete strukturelle Zugehörigkeit für die jeweiligen Sichten aus Tabelle 5.9 kann der Abbildung 5.31 entnommen werden. Dieser stellt ebenfalls nur die obersten Komponenten dar sowie die dazugehörigen Fragmente aus dem Abschnitt fragments. Die genaue Beschreibung der darunterliegenden Controls erfolgt im weiteren Verlauf dieser Dokumentation. Alle Controls aus der Tabelle 5.9 müssen, wie in Listing 5.36 zu sehen ist, dem SplitApp-Control hinzugefügt werden, das dann letzten Endes zur Anzeige der Anwendung zurückgegeben wird.

```
1278  /**
1279   * Alle erzeugten Seiten werden dem oSplitApp-Objekt hinzugefuegt.
1280   */
1281    oSplitApp.addMasterPage(oMasterCategoryPage);
1282    oSplitApp.addMasterPage(oMasterSensorsList);
1283    oSplitApp.addMasterPage(oMasterChannelList);
1284    oSplitApp.addDetailPage(oDashboard);
1285    oSplitApp.addDetailPage(oDetailPage);
1286    oSplitApp.addDetailPage(oInitialPage);
```

```
1287      oSplitApp.addDetailPage(oPrinterPage);
1288      oSplitApp.addDetailPage(oRaspberryStatistics);
1289      oSplitApp.setDefaultTransitionNameDetail("fade");
1290      oSplitApp.setInitialMaster("masterPage");
1291      oSplitApp.setInitialDetail("initialPage");
1292
1293    /**
1294     * oSplitApp wird zurueckgegeben. Dieses Control wird schliesslich zur
          Darstellung genutzt.
1295     */
1296
1297      return oSplitApp;
```

Listing 5.36: Sensorik: Erzeugen des SplitApp-Controls als oberste Komponente der View

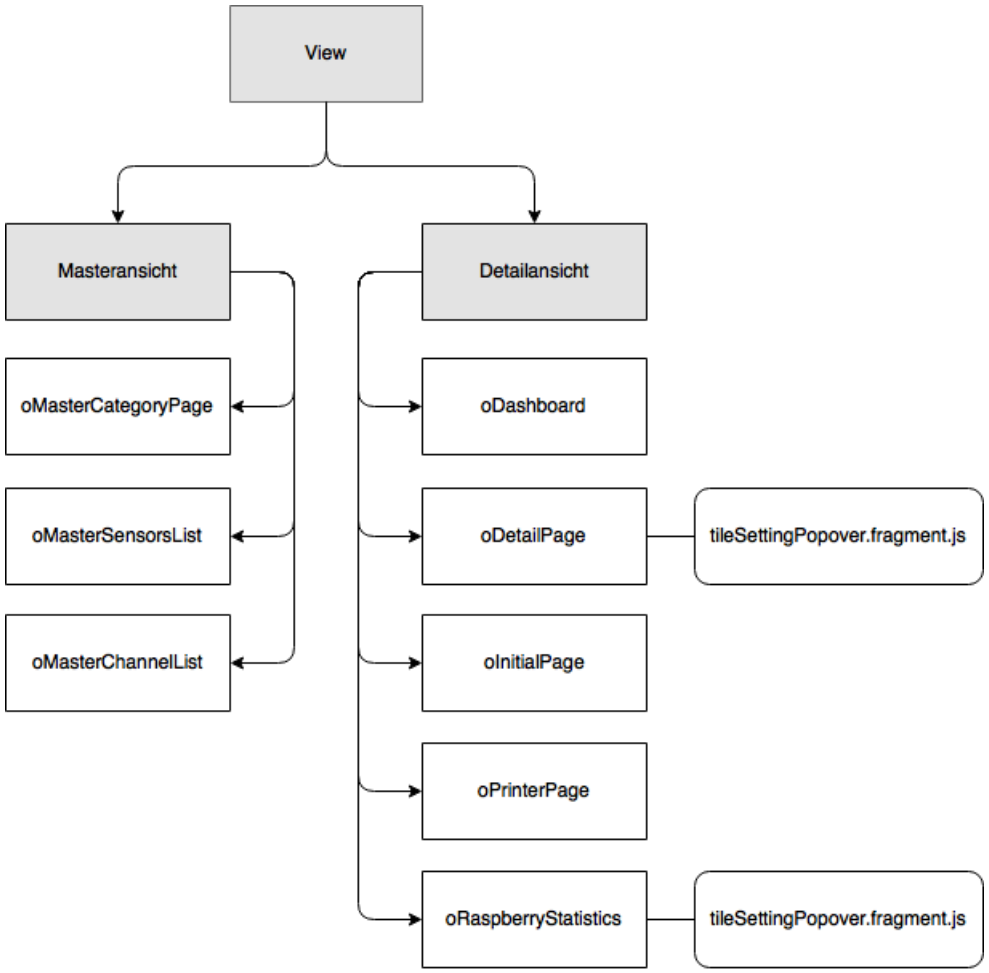


Abbildung 5.31.: Sensorik: Grundgerüst der GUI (Modell)

Datenanbindung

Als Datenlieferant für die grafischen Elemente werden die Services aus dem Abschnitt services verwendet. Dafür werden im Controller entsprechende JavaScript Object Notation (JSON)-Datenmodelle in der onInit()-Funktion erzeugt, der View bekannt gemacht und mit einer Identifikation (ID) in der setModel()-Funktion versehen. Die onInit()-Funktion wird standardmäßig ausgeführt sobald das Rendern der Anwendung abgeschloßen ist und der Controller instanziiert wurde. Da die View jedes JSON-Modell kennt, wird die ID zum einen für das Data-Binding der Controls verwendet und zum anderen können die Datenmodelle direkt beim Aufrufen der Services angesprochen und mit Daten befüllt werden.

```
9   onInit: function() {
10 /**
11  * Erzeugen aller notwendingen Datenmodelle. Datenmodell "oSensorsModel"
      beinhaltet alle in der Datenbank gespeicherten Sensoren (Raspberry Pis)
      . Der Aufruf ist nur einmalig notwendig. Daher wird der Service, der
      die Daten bereitstellt, einmalig bei der Initialisierung des Modells
      genutzt.
12  */
13     var oSensorsModel =     new sap.ui.model.json.JSONModel();
14     var oChannelModel =     new sap.ui.model.json.JSONModel();
15     var oTableModel   =     new sap.ui.model.json.JSONModel();
16     var oExtremeTableModel=  new sap.ui.model.json.JSONModel();
17     var oChartModel =     new sap.ui.model.json.JSONModel();
18     var oTempChartModel =     new sap.ui.model.json.JSONModel();
19     var oHumChartModel =    new sap.ui.model.json.JSONModel();
20     var oCo2ChartModel =    new sap.ui.model.json.JSONModel();
21     var oAirChartModel =    new sap.ui.model.json.JSONModel();
22     var oMinMaxAvgModel =     new sap.ui.model.json.JSONModel();
23     var oStatisticValues =    new sap.ui.model.json.JSONModel();
24     var oCategoryModel =    new sap.ui.model.json.JSONModel();
25     var oRpStatisticsModel = new sap.ui.model.json.JSONModel();
26 /**
27  * Alle Datenmodelle werden der View zur Verfuegung gestellt und erhalten
      eine eindeutige ID.
28  */
29     this.getView().setModel(oChannelModel,'channelModel_1');
30     this.getView().setModel(oTableModel,'tableModel_1');
31     this.getView().setModel(oSensorsModel,'sensorsModel_1');
32     this.getView().setModel(oExtremeTableModel,'extremeTableModel_1');
33     this.getView().setModel(oChartModel,'chartModel_1');
34     this.getView().setModel(oTempChartModel,'tempDashModel_1');
35     this.getView().setModel(oHumChartModel,'humDashModel_1');
36     this.getView().setModel(oCo2ChartModel,'co2DashModel_1');
37     this.getView().setModel(oAirChartModel,'airDashModel_1');
38     this.getView().setModel(oMinMaxAvgModel,'minMaxAvgModel_1');
39     this.getView().setModel(oStatisticValues,'statisticValuesModel_1');
40     this.getView().setModel(oCategoryModel,'categoryModel_1');
41     this.getView().setModel(oRpStatisticsModel,'rpStatisticsModel_1');
```

Listing 5.37: Sensorik: Erzeugen der JSON-Models im Controller

Listing 5.37 gibt einen Überblick über alle verwendeten Models. Welche Controls, welche Services und Models nutzen, erfolgt im jeweiligen Dokumentationsabschnitt der entsprechenden Controls.

Datenabruf mittels Asynchronous JavaScript and XML (AJAX) Um die Datenmodelle mit Daten zu befüllen, wird die Datenübertragungsmethode AJAX genutzt. Sie ermöglicht den Datenaustausch über Internetprotokolle, ohne dass ein Refresh der Anwendung durchgeführt werden muss. Die nähere Betrachtung des Begriffs AJAX gibt Anlass zu glauben, dass die Datenübertragung nur asynchron erfolgt. Die ist jedoch nicht der Fall. Die Übertragungsart kann beliebig bestimmt werden.

```
1189 /**
1190  * Eine Funktion welche alle, in der Datenbank registrierten,
       Sensorkategorien
1191  * laedt (Raspberry Pi, Drucker etc.).
1192  */
1193   loadCategories: function(){
1194     var self = this;
1195
1196    $.ajax({
1197      url: self.sPath+"pushSensorTypes.xsjs",
1198      async: false,
1199      type: "GET",
1200      dataType: "json",
1201      cache: "false",
1202      success: function(response){
1203          self.getView().getModel('categoryModel_1').setData(response);
1204      },
1205      complete: function(){
1206
1207      },
1208      error: function(xhr,status,error){
1209
1210      }
1211    });
1212   },
```

Listing 5.38: Sensorik: Eine Funktion zum Laden der Sensorkategorien

Listing 5.38 zeigt eine Funktion, in dem ein jQuery basierter AJAX-Aufruf mit folgenden Parametern stattfindet:

- **url**
  Angabe des Services, der die Daten bereitstellt.
- **async**
  Ein Flag das angibt, ob die Daten synchron oder asynchron geladen werden sollen.
- **type**
  Gibt an, welche Anfragemethode genutzt werden soll.

- **dataType**
  Angabe des Datentyps.

- **cache**
  Ein Flag das festlegt, ob ein caching erfolgen soll oder nicht.

- **success**
  Beinhaltet eine Funktion die ausgeführt wird, wenn die Anfrage erfolgreich war. Mit dem Ergebnis der Anfrage wird ein JSON-Datenmodell durch Ansprechen der ID mit Daten befüllt. Wenn AJAX-Aufrufe mehrmals ausgeführt werden und ein bereits mit Daten befülltes Objekt überschrieben wird, so werden auch die sichtbaren Datenangaben innerhalb der Anzeige überschrieben ohne das ein manuelles Laden der Anwendung angestoßen werden muss.

- **complete**
  Beinhaltet eine Funktion, die ausgeführt wird, wenn die Anfrage abgeschlossen ist.

- **error**
  Beinhaltet eine Funktion, die ausgeführt wird, wenn die Anfrage fehlgeschlagen ist.

Alle im Rahmen dieser Dokumentation durchgeführten AJAX-Aufrufe sind, wie in dem Listing 5.38 angegebenem Schema, erstellt worden. Auf die genaue Beschreibung aller implementierten AJAX-Aufrufe wird verzichtet, da der Grundaufbau stets derselbe ist.

Zyklischer Datenabruf Das zyklische Erneuern von Messdaten ist ebenfalls mit der Datenübertragungsmethode AJAX implementiert worden. Dabei gliedert sich der Gesamtprozess jeweils in eine *start*-, *poll*- (engl. polling; deu. Abfrage) und *stop*-Funktion. Im Rahmen der Anwendung des Sensorik Szenarios gibt es mehrere grafische Komponenten, deren Daten bei der Nutzung der Anwendung zyklisch aktualisiert werden. Die jeweiligen start-Funktionen leiten den Prozess ein. Die poll-Funktionen enthalten die konkrete Logik, welche den jeweiligen Service aufrufen und die stop-Funktionen leiten ein Beenden der Aktualisierung ein. Wie im vorherigen Abschnitt wird der zyklische Datenaufruf exemplarisch für eine Komponente anhand von Quellcode erläutert. Für alle weiteren Komponenten ist der Datenabruf stets gleich aufgebaut. Sie unterscheiden sich lediglich in ihren Variablennamen und Services. Die Abbildungen 5.32-5.37 zeigen eine Übersicht über alle Komponenten, deren Daten sich kontinuierlich erneuern mit ihren dazugehörigen start-, poll- und stop-Funktionen. Dabei gliedert sich die Darstellung in einer Baumstruktur, um übersichtlich darzustellen, wie die jeweiligen Komponenten miteinander verbunden sind. Die verwendeten Datenmodelle sind ebenfalls aufgeführt.

```
391 /**
392  * Eine Funktion zum Starten des Poll-Verfahrens fuer die Messsensoren.
       Erwartet IP-Adresse des Sensors zum laden der korrekten Daten.
393  *
394  * @param sIp
395  */
```

```
396   startPolling: function(sIp){
397     this.currentSensorIp = sIp;
398     console.log("Aktualisierung der Sensorwerte wurde gestartet!");
399     this.poll(sIp,false);
400   },
```

Listing 5.39: Sensorik: Eine Funktion welche den Vorgang zum zyklischen Datenabruf startet

Listing 5.39 zeigt eine start-Funktion, mit dem ein zyklischer Vorgang angestoßen wird, um die Messwerte von einem Sensor abzurufen. Sie erwartet als Parameter die Internet Protocol (IP)-Adresse des ausgewählten Sensors, damit nur diejenigen Werte abgefragt werden, zu denen auch der Sensor zugeordnet ist. Nicht jede start-Funktion benötigt solch einen Parameter. Die meisten Services erfüllen genau einen bestimmten Zweck, sodass solch ein Parameter nicht notwendig ist. Die IP-Adresse des ausgewählten Sensors wird in der Variable *currentSensorIp* hinterlegt, damit andere Funktionen immer auf die IP-Adresse des aktuell ausgewählten Sensors zugreifen können. Eine Konsolenausgabe gibt zudem den Hinweis, dass ein Aktualisierungsprozess gestartet wurde und danach wird die poll-Funktion ausgeführt.

```
406 /**
407  * Eine Funktion zum laden der Messdaten. Erwartet als Parameter die IP-
       Adresse des Messsensors und ob Synchron oder Asynchron aktualsiert
       werden soll.
408  *
409  * @param sIp
410  * @param isAsync
411  */
412   poll: function(sIp, isAsync){
413     var self = this;
414
415 /**
416  * AJAX-Call zum Abrufen der aktuellen Messwerte.
417  */
418     $.ajax({
419       url: self.sPath+"pushCurrentSensorValue.xsjs?id="+self.
      currentSensorIp,
420       async: isAsync,
421       type: "POST",
422       dataType: "json",
423       cache: "false",
424       success: function(response){
425           self.getView().getModel('channelModel_1').setData(response);
426           self.setChannelStates();
427
428           if(self.sensorCategory==="Drucker"){
429             self.setTrayStates();
430             self.setPrinterStates();
431           }
432       },
433       complete: function(){
434         self.timeoutId = setTimeout(function(){
```

```
435             console.log("Daten aktualisiert fuer: "+self.currentSensorIp);
436             self.poll(self.currentSensorIp,true);
437           },1000*self.s_waitInSec);
438         },
439         error: function(xhr,status,error){
440
441         }
442       });
443
444   },
```

Listing 5.40: Sensorik: Eine Funktion zum zyklischen Abruf von Messdaten

In Listing 5.40 wird die eigentliche Logik des zyklischen Datenabrufs dargestellt. Wenn der AJAX-Request (deu. Anfrage) erfolgreich war, wird das Datenmodell mit dem neuen Datensatz überschrieben und die jeweiligen Status der Messsensoren neu gesetzt. Handelt es sich bei dem ausgewählten Sensor um ein Druckergerät, so werden die Status der Fächer, der Scan- sowie Druck-Status gesetzt.
Wenn die gesamte AJAX-Abfrage abgeschlossen (complete) ist, wird anschließend ein neuer Abruf mittels der setTimeout()-Funktion angestoßen. In dieser wird mittels einer anonymen Funktion die poll-Funktion rekursiv mit der aktuellen IP-Adresse erneut aufgerufen. Beim erstmaligen Auswählen eines Sensors erfolgt die Datenabfrage synchron, damit sofort Daten verfügbar sind und diese nicht erst nach der angegebenen Zeit sichtbar sind. Danach erfolgen die Aufrufe stets asynchron. Die setTimeout()-Funktion liefert eine ID zurück, in der intern hinterlegt ist, nach wie vielen Millisekunden, angegeben als Parameter, die Funktion ausgeführt werden soll. Diese ID wird in der Variable *timeoutId* gespeichert und kontinuierlich, nach jedem rekursiven Aufruf überschrieben.

```
477 /**
478  * Eine Funktion zum Stoppen des Poll-Verfahrens fuer die Messsensoren.
479  */
480   stopPolling: function(){
481     clearTimeout(this.timeoutId);
482     console.log("Sensordaten werden nicht mehr aktualisiert");
483   },
```

Listing 5.41: Sensorik: Eine Funktion welche den Vorgang zum zyklischen Datenabruf stoppt

Listing 5.41 verdeutlicht den Prozess, der zum Beenden des rekursiven Aufrufes führt. Die ID die in der Variable *timeoutId* hinterlegt ist und den Aufruf nach der angegebenen Zeit durchführt, wird überschrieben, sodass der letzte rekursive Aufruf nicht mehr durchgeführt wird und zum Beenden des gesamten Prozesses führt.
Die Aktualisierungslogik aller weiteren Komponenten sind in dem selben Muster realisiert. Einige Komponenten haben in ihrer poll-Funktion mehrere AJAX-Aufrufe. Die Aktualisierung wird dann lediglich nur von einem AJAX-Aufruf angestoßen. Jedoch werden die Daten für alle erneuert. Dies ist beim Dashboard (siehe Abbildung 5.35 (oDashboard)) eines ausgewählten Raspberry Pis der Fall. Der selbe Service wird vier mal mit unterschiedlichen Parametern aufgerufen, um die Daten jedes Messensors gesondert ab-

zuspeichern. Die AJAX-Aufrufe befinden sich dann in der dazugehörigen poll-Funktion. Die Abbildungen 5.32-5.37 zeigen eine kurze Übersicht über diejenigen Komponenten, deren Daten zyklisch abgerufen werden. Die unterste Ebene repräsentiert dabei jene Komponenten, bei denen ein Data-Binding stattfindet.

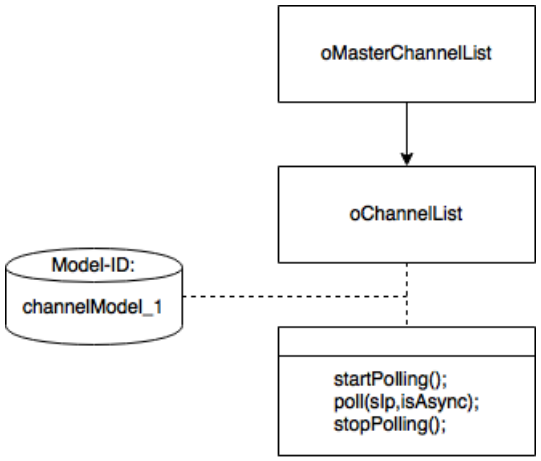


Abbildung 5.32.: Sensorik: Zyklischer Datenabruf oMasterChannelList

Abbildung 5.32 zeigt die einzelnen Komponenten der oMasterChannelList. Das Data-Binding findet in der untersten Ebene (oChannelList) innerhalb der View statt. Verwendet wird das Datenmodell mit der ID *channelModel_1*. Die genaue Dokumentation erfolgt im Abschnitt oMasterChannelList.

- start-Funktion: startPolling(sIp);
- poll-Funktion: poll(sIp,isAsync);
- stop-Funktion: stopPolling();

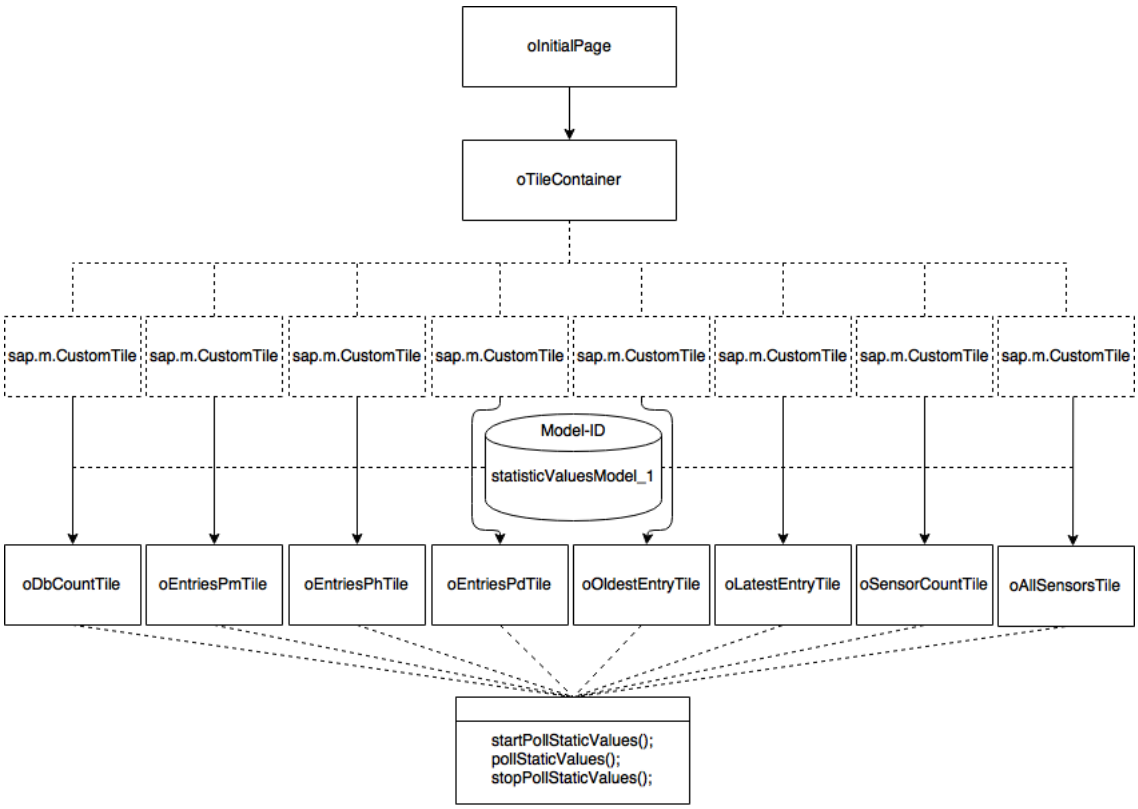


Abbildung 5.33.: Sensorik: Zyklischer Datenabruf oInitialPage

Abbildung 5.33 zeigt die einzelnen Komponenten der oInitialPage. Das Data-Binding findet in der untersten Ebene innerhalb der View statt. Verwendet wird das Datenmodell mit der ID *statisticValuesModel_1*. Die genaue Dokumentation dieser Komponente erfolgt im Abschnitt oInitialPage.

- start-Funktion: startPollStaticValues;
- poll-Funktion: pollStaticValues();
- stop-Funktion: stopPollStaticValues();

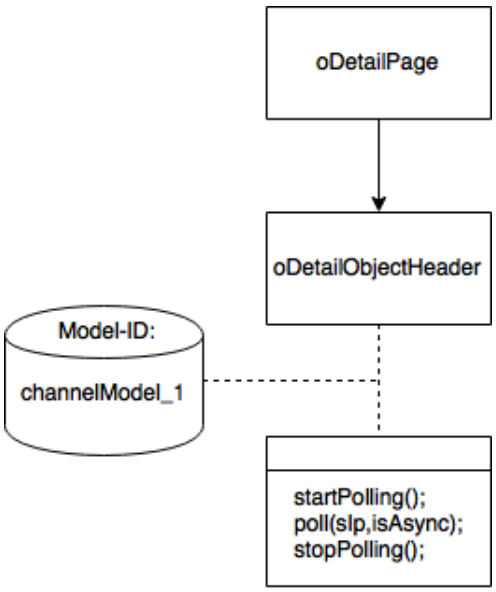


Abbildung 5.34.: Sensorik: Zyklischer Datenabruf oDetailPage

Abbildung 5.34 zeigt diejenigen Komponenten der oDetailPage, deren Daten zyklisch aktualisiert werden. Das Data-Binding findet auf der Ebene des ObjectHeaders im Controller statt. In der Funktion setObjectHeaderValues() werden die Eigenschaften des Controls an die Schlüssel-/Wert-Paare des Datenobjekts gebunden. Verwendet wird das Datenmodell mit der ID *channelModel_1*. Die genaue Dokumentation dieser Komponente erfolgt im Abschnitt oDetailPage.

- start-Funktion: startPolling(sIp);
- poll-Funktion: poll(sIp,isAsync);
- stop-Funktion: stopPolling();

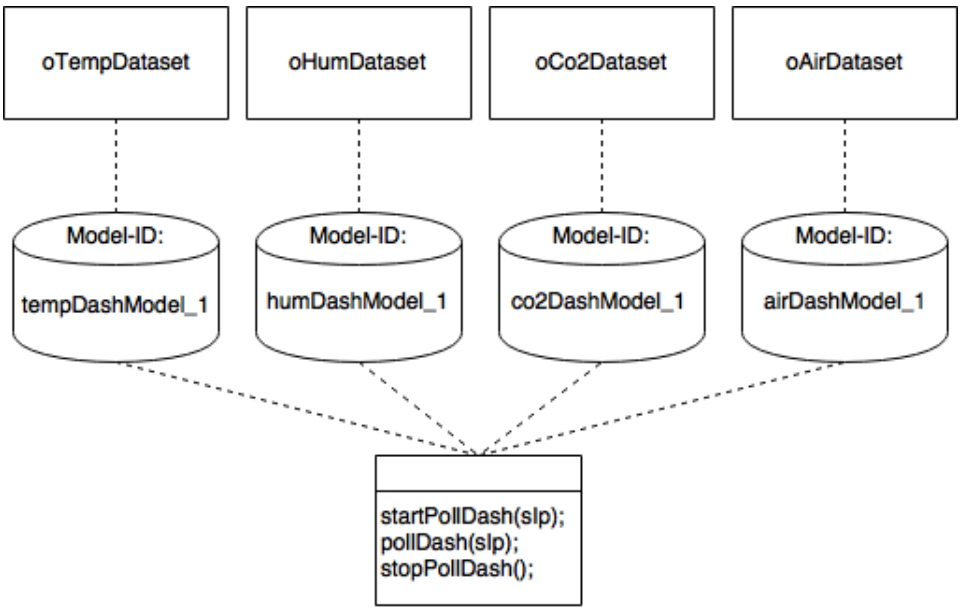


Abbildung 5.35.: Sensorik: Zyklischer Datenabruf oDashboard

Abbildung 5.35 zeigt diejenigen Komponenten, die dem Dashboard (oDashboard) eines Raspberry Pis als eigenständige Datensätze dienen, das dann für das jeweilige Chart explizit als Datenlieferant gesetzt werden muss. Diese Komponenten sind eigenständige Controls, die kein Wurzelelement haben. Jede in dem Dashboard dargestellte Chart nutzt einen eigenen Datensatz, dem wiederum ein eigenes Datenobjekt zugewiesen ist. Das Temperaturen-Chart nutzt das Datenmodell mit der ID *tempDashModel_1*, das Luftfeuchtigkeits-Chart das Datenmodell mit der ID *humDashModel_1*, das CO2-Chart das Datemodell mit der ID *co2DashModel_1* und das Luftdruck-Chart das Datenmodell mit der ID *airDashModel_1*. Die genaue Dokumentation und das Zusammenspiel dieser Komponenten erfolgt im Abschnitt oDashboard.

- start-Funktion: startPollDash(sIp);
- poll-Funktion: pollDash(sIp);
- stop-Funktion: stopPollDash();

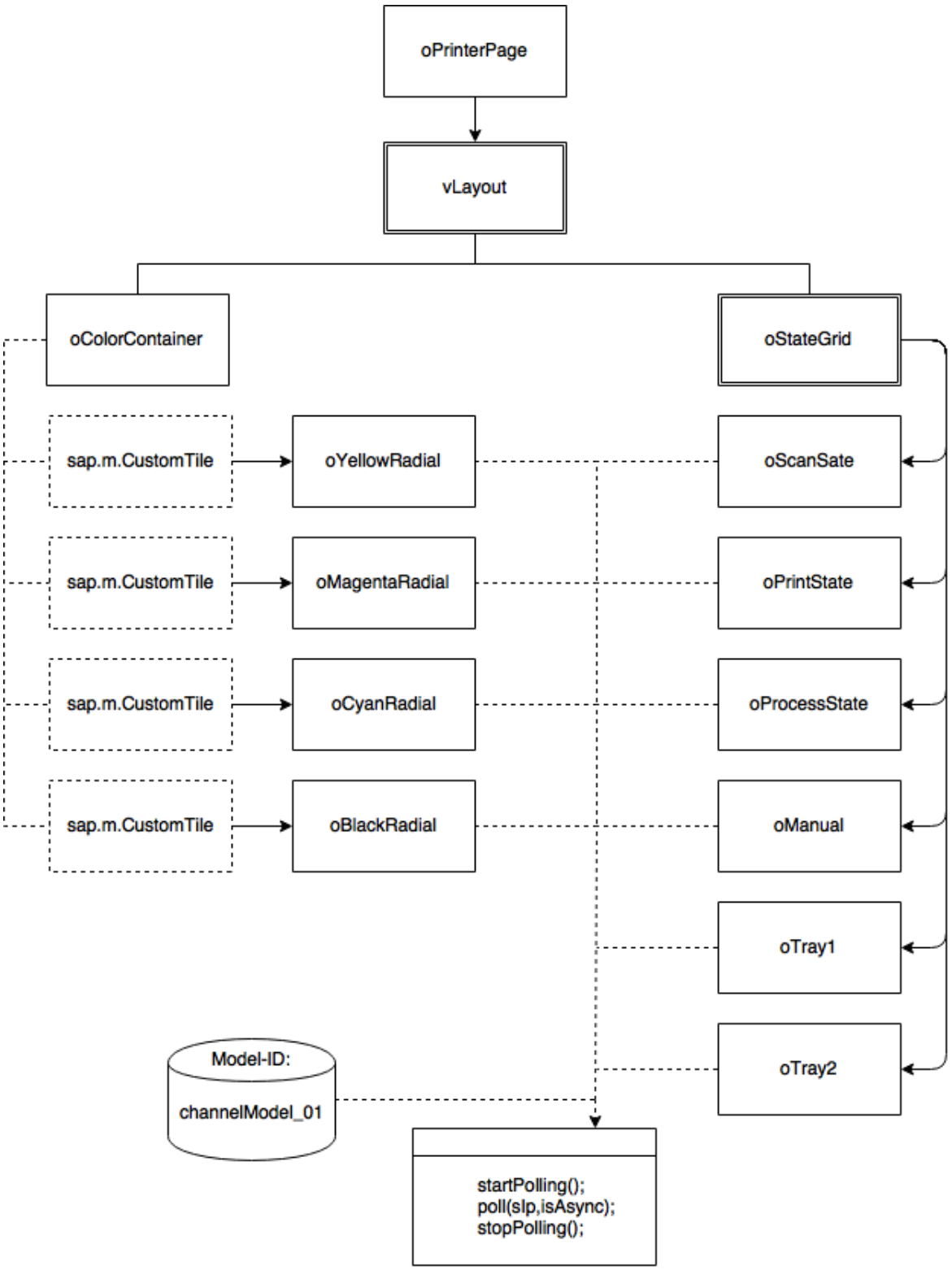


Abbildung 5.36.: Sensorik: Zyklischer Datenabruf oPrinterPage

Abbildung 5.36 zeigt die einzelnen Komponenten der Detailansicht (oPrinterPage) der Druckergeräte. Das Data-Binding findet auf beiden Seiten in den untersten Ebenen innerhalb der View statt. Alle Elemente nutzen das Datenobjekt mit der ID *channelModel_01*. Die genaue Dokumentation dieser Komponente erfolgt im Abschnitt oPrinterPage.

- start-Funktion: startPolling(sIp);
- poll-Funktion: poll(sIp,isAsync);
- stop-Funktion: stopPolling();

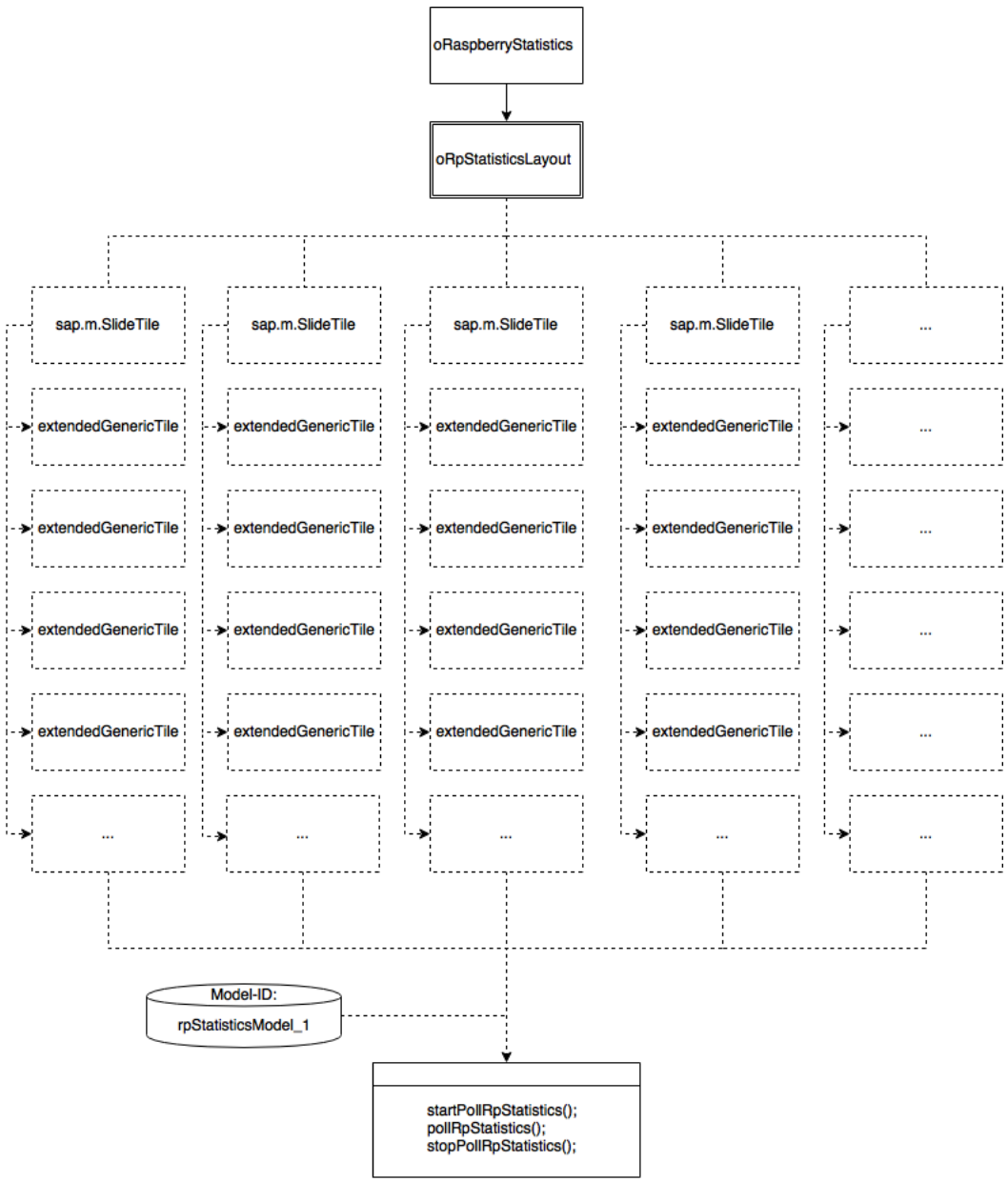


Abbildung 5.37.: Sensorik: Zyklischer Datenabruf oRaspberryStatistics

Abbildung 5.37 zeigt die einzelnen Komponenten, die eine Übersicht über alle Raspberry Pis als sich verändernde Kacheln (oRaspberryStatistics) darstellen. Diese Kacheln sind generisch aufgebaut und richten sich automatisch nach der Anzahl der Raspberry Pis und deren angeschlossenen Messsensoren. Die Anzahl der Kacheln und deren Inhalt können, je nach Verfügbarkeit, variieren. Die genaue Dokumentation erfolgt im Abschnitt oRaspberryStatistics. Das Data-Binding erfolgt im Controller mittels der Funktion setRpStatistics() und nutzt das Datenobjekt mit der ID *rpStatisticsModel_1*. Die genaue Dokumentation dieser Komponente erfolgt im Abschnitt oRaspberryStatistics.

- start-Funktion: startPollRpStatistics();
- poll-Funktion: pollRpStatistics();
- stop-Funktion: stopPollRpStatistics();

Masteransicht

Dieser Abschnitt dokumentiert alle Elemente der Masteransicht (siehe Abbildung 5.30), die auf Abbildung 5.31 aufgeführt sind, im Detail. Neben den grafischen Elementen wird Bezug auf die wesentlichen Bestandteile der Logik genommen. Diese Ansicht dient hauptsächlich der Navigation zwischen den Sensoren. Der in den folgenden Listings aufgezeigte Quellcode spiegelt schrittweise die wesentlichen Inhalte der initial.view.js- und initial.controller.js-Datei wieder.

oMasterCategoryPage Der rötlich markierte Bereich auf Abbildung 5.38 stellt denjenigen Bereich dar, dessen Control in der Variable *oMasterCategoryPage* gespeichert ist. Das Page-Control besitzt eine aggregierende Eigenschaft, die sich *content* nennt. Mittels dieser Eigenschaft können dem Control weitere Controls hinzugefügt werden, um so komplexe Anwendungen zu gestalten. Die Sensorkategorien werden durch die Verwendung des Services *pushSensorTypes.xsjs* abgerufen, die in dem Datenmodell mit der ID *categoryModel_1* gespeichert werden.
Die Sensorkategorien selbst werden als eine Art Liste in einem List-Control dargestellt. Als konkrete, einzelne Listenelemente, die dann eine Sensorkategorie darstellen, wird das ObjectListItem-Control verwendet. Wie in Listing 5.42 zu sehen, ist es entscheidend, dass zuerst das List-Control erzeugt wird (oMasterCategorieList), dann die einzelnen Listenelemente an das Datenobjekt gebunden werden und anschließend dem Page-Control (oMasterCategoriePage) übergeben wird. Zum Erzeugen der Listenelemente wird eine *template*-Funktion genutzt. Dieser durchläuft iterativ den angegebenen Pfad und erzeugt für jeden Eintrag ein Listenelement. Beim Auswählen einer Kategorie wird die onCategoryPress()-Funktion im Controller ausgeführt. Das korrekte Icon stellt die getCategoryIcon()-Funktion durch Übergeben des Sensortypes zur Verfügung. Die Sensorkategorien werden einmalig in der onInit()-Funktion durch Aufrufen der Funktion loadCategories() beim erstmaligen Rendern der Anwendung geladen.

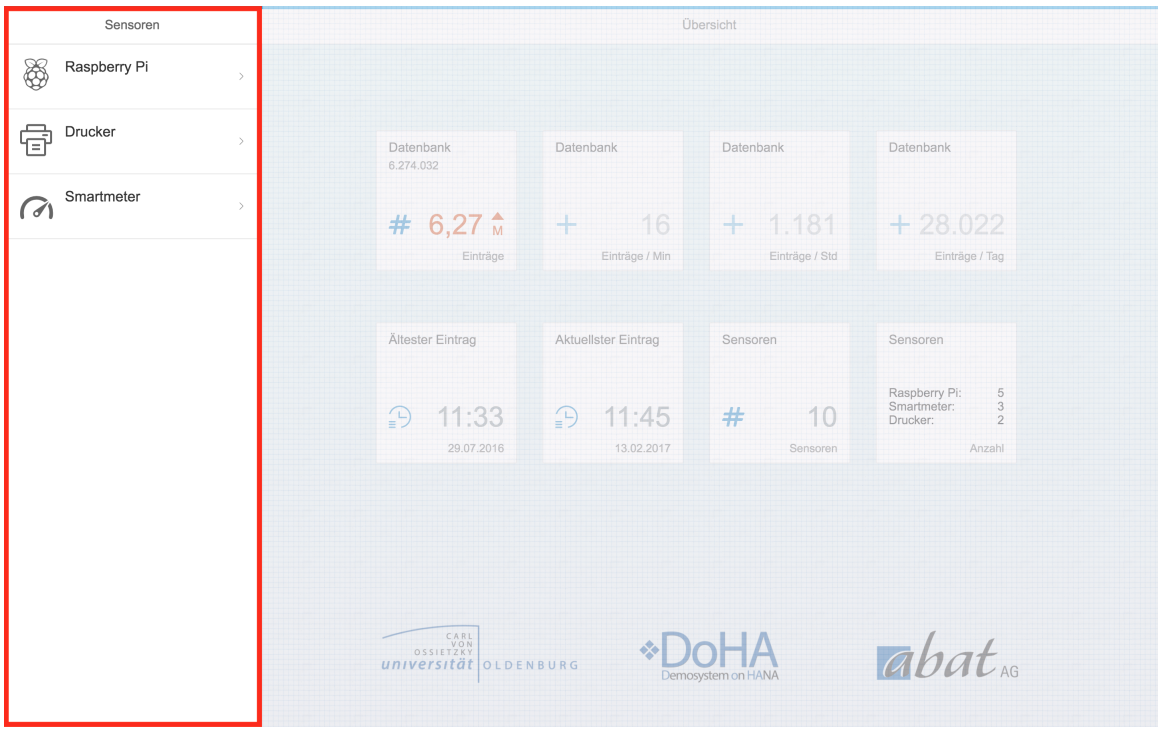


Abbildung 5.38.: Sensorik: Anzeige der Sensorkategorien

```
944  /**
945   * Erzeugen einer Liste zum Anzeigen der Sensor-Kategorien (Rapsberry Pi
      , Drucker, Smartmeter etc.)
946   */
947    var oMasterCategoryList = new sap.m.List("masterCategoryList_1",{});
948  /**
949   * Die Listenelemente der oMasterCategoryList-Liste werden an das
      Datenobjekt mit der ID "categoryModel_1" gebunden.
950   */
951    oMasterCategoryList.bindItems({
952      path: 'categoryModel_1>/sensors',
953      template: new sap.m.ObjectListItem({
954        title : '{categoryModel_1>type}',
955        type: "Navigation",
956        press:[oController.onCategoryPress,oController],
957        icon: {
958          path: 'categoryModel_1>type',
959          formatter: function(sensor){
960            return oController.getCategoryIcon(sensor);
961          }
962        }
963      }),
964    });
965  /**
966   * Erzeugen eines Page-Objekts zum Anzeigen der Sensor-Kategorien.
      Aggregiert im Content-Bereich die oMasterCategoryList.
967   */
```

```
968     var oMasterCategoryPage = new sap.m.Page("masterPage",{
969       title: "Sensoren",
970       content:[oMasterCategoryList]
971     });
```

Listing 5.42: Sensorik: Erzeugen der Sensorkategorien mit anschließendem Data-Binding

oMasterSensorsList Die rötlich markierten Bereiche auf den Abbildungen 5.39 und 5.40 stellen denjenigen Bereich dar, dessen Control in der Variable *oMasterSensorsList* gespeichert ist. Das Page-Control besitzt eine aggregierende Eigenschaft, die sich *content* nennt. Mittels dieser Eigenschaft können dem Control weitere Controls hinzugefügt werden, um so komplexe Anwendungen zu gestalten. Wie im vorherigen Abschnitt auch, werden in diesem Abschnitt die Sensoren wieder durch eine Liste mittels des List-Controls dargestellt. Bei den auf den Abbildungen 5.39 und 5.40 dargestellten Sensoren handelt es sich um ein und dasselbe List-Control mit den dazugehörigen Listenelementen. Welche Sensoren nun geladen werden, wird dadurch bestimmt, welche Sensorkategorie zuvor ausgewählt wurde. Die jeweiligen Sensoren einer Sensorkategorie werden durch die Verwendung des Services *pushAllSensorsForType.xsjs* abgerufen, die in dem Datenmodell mit der ID *sensorsModel_1* gespeichert werden. Im zugehörigen AJAX-Aufruf wird diesem Service der Sensortyp mitgegeben. Der AJAX-Aufruf erfolgt in der loadSensors()-Funktion.

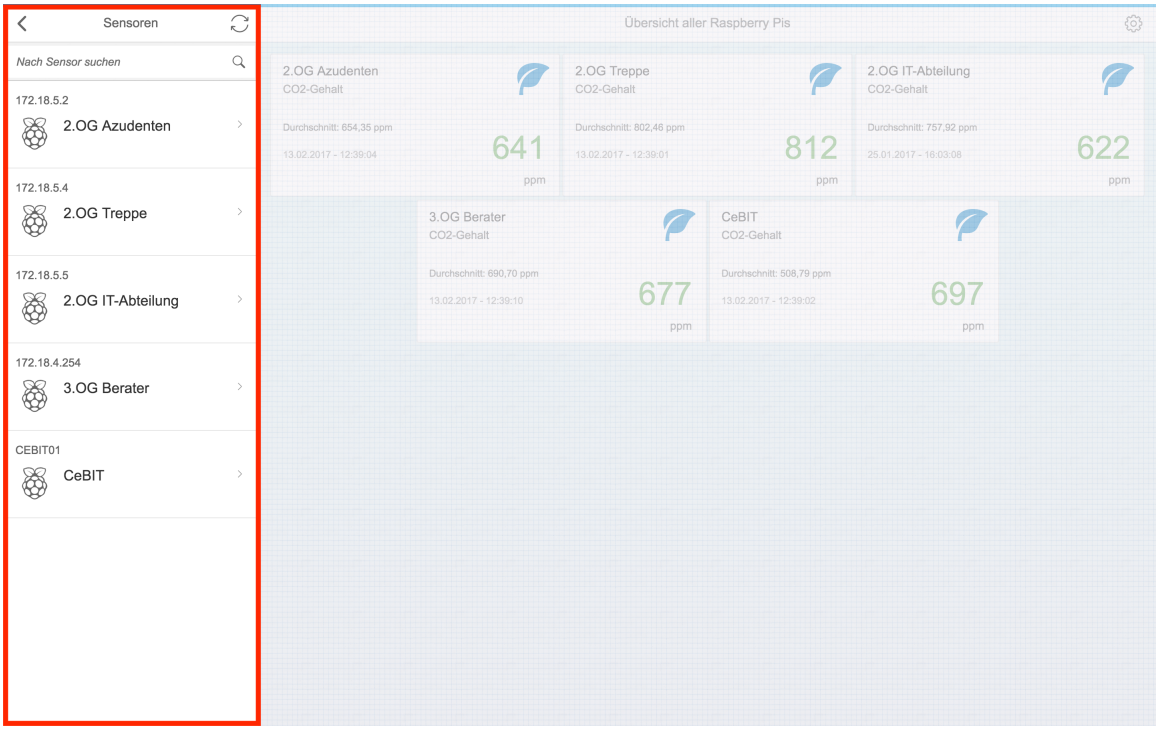


Abbildung 5.39.: Sensorik: Anzeige der Sensoren (Raspberry Pi)

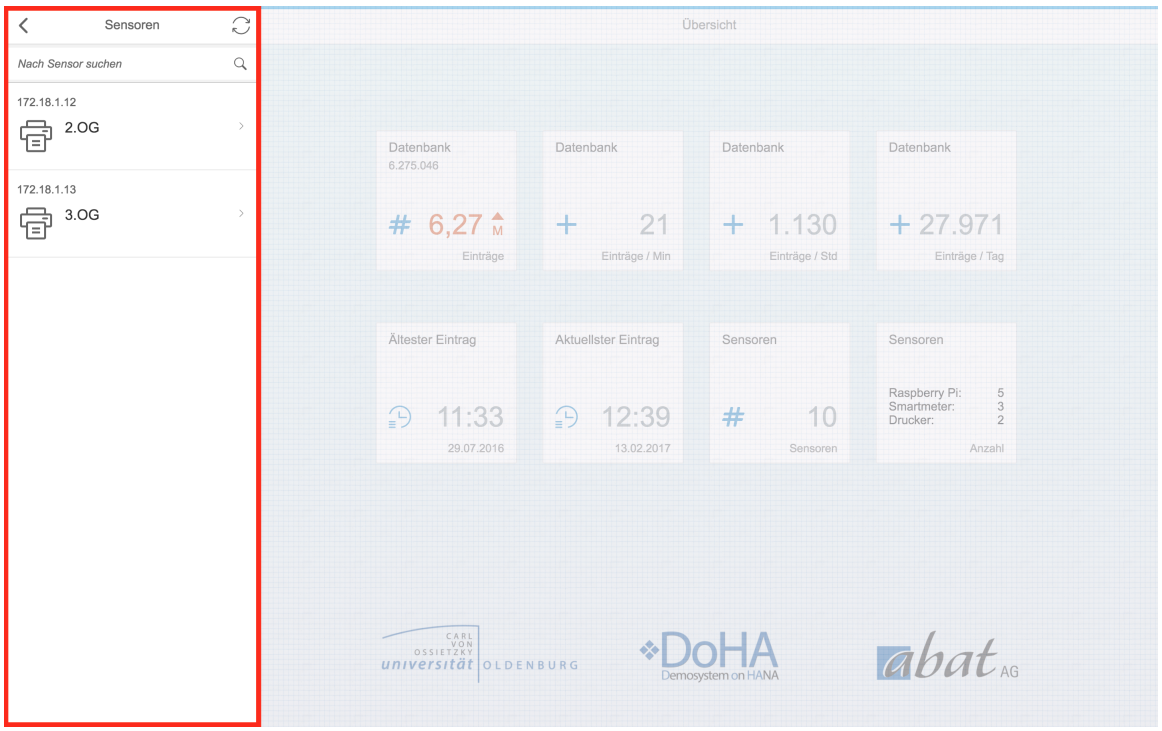


Abbildung 5.40.: Sensorik: Anzeige der Sensoren (Drucker)

```
73  /**
74   * Erzeugen einer Liste zum Anzeigen der Raspberry-Pis. Die einzelnen
     Listenelemente werden mit der bindItems()-Fuktion an das Datenobjekt
     mit der ID "sensorModel\_1" gebunden.
75   */
76    var oSensorsList = new sap.m.List("sensorsList_1",{});
77
78  /**
79   *Die einzelnen Listenelemente werden an das Datanobjekt mit der ID "
     sensorsModel\_1" gebunden. Dabei wird ein Template verwendet
80   *das alle Elemente des Objekts durchlaeuft und anschliessend anzeigt.
     Das richtige Icon liefert die getSensorIcon(sensor)-Funktion.
81   *Beim betaetigen eines Listenelements wird die onSensorPress()-Funktion
      ausgefuehrt.
82   */
83    oSensorsList.bindItems({
84      path: 'sensorsModel_1>/sensors',
85      template: new sap.m.ObjectListItem({
86        title : '{sensorsModel_1>location}',
87        intro: '{sensorsModel_1>sensor_id}',
88        type: "Navigation",
89        press:[oController.onSensorPress,oController],
90        icon: {
91          path: 'sensorsModel_1>type',
92          formatter: function(sensor){
93            return oController.getSensorIcon(sensor);
94          }
```

```
95          }
96        }),
97
98      });
99
100   /**
101    * Erzeugen eines Page-Objekts zum Anzeigen der Raspberry-Pis. Der
       Content-Bereich aggregiert die Liste der Raspberry-Pis. Im
       headerContent befindet sich ein Button zum
102    * synchronisieren der Raspberry Pis mit der Datenbank. Bei einem Press-
       Event wird die syncCategories-Funktion im Controller ausgefuehrt. Im
       subHeader-Bereich befindet sich
103    * eine Toolbar mit einem Suchfeld. Bei der live-suche nach einem Sensor
        wird die onLiveSensorSearch-Funktion im Controller ausgefuehrt.
104    */
105     var oMasterSensorsList = new sap.m.Page("sensorsMaster",{
106       title: "Sensoren",
107       showNavButton: true,
108       navButtonPress:[oController.onNavButtonPressSensors,oController],
109       headerContent:[
110                       new sap.m.Button("sensorSyncButton_1",{
111                         icon: "sap-icon://synchronize",
112                         press:[oController.syncCategories,oController]
113                       })
114       ],
115       subHeader:[
116                   new sap.m.Toolbar({
117                     content:[
118                       new sap.m.SearchField({
119                         placeholder: "Nach Sensor suchen",
120                         liveChange:[oController.onLiveSensorSearch,
    oController]
121                       })
122                     ]
123                   })
124       ],
125       content: [oSensorsList]
126     });
```

Listing 5.43: Sensorik: Erzeugen der Sensoren mit anschließendem Data-Binding

In Listing 5.43 ist zu erkennen, dass zu Beginn wieder ein List-Control erzeugt wird (oSensorsList). Danach werden wieder mithilfe einer *template*-Funktion der angegebene Pfad des Datenobjekts iterativ durchlaufen und pro Eintrag ein Listenelement durch das ObjectListItem-Control erzeugt und ausgewählte Eingenschaften an das Datenmodell gebunden. Zudem aggregiert das Page-Control zwei Header, die einen Aktualisierungs-Button sowie ein Suchfeld beinhalten.

```
1172 /**
1173  * Eine Funktion welche ausgefuehrt wird wenn eine Sensorkategorie
       ausgefwaehlt wird. Wechselt zur entsprechenden Master-Seite und laedt
       nur die
1174  * Sensoren der entsprechenden Kategorie.
```

```
1175  *
1176  * @param oEvent
1177  */
1178    onCategoryPress: function(oEvent){
1179      var src = oEvent.getSource();
1180      this.sensorCategory = src.getTitle();
1181
1182      if(this.sensorCategory==="Raspberry Pi" || this.sensorCategory==="
      Drucker"){
1183        this.loadSensors(this.sensorCategory);
1184        this.oSplitApp.toMaster("sensorsMaster");
1185
1186        if(this.sensorCategory=="Raspberry Pi"){
1187          this.oSplitApp.toDetail("raspberryPage_1");
1188          this.stopPollStaticValues();
1189          this.startPollRpStatistics();
1190        }
1191      }else{
1192        window.open(this.smartmeterPath,'_blank');
1193      }
1194    },
```

Listing 5.44: Sensorik: Quellcode der onCategoryPress()-Funktion

Listing 5.44 stellt den Quellcode der onCategoryPress()-Funktion dar. Handelt es sich bei der ausgewählten Sensorkategorie um einen Raspberry Pi oder Drucker, so werden dann die entsprechenden Sensoren mittels der loadSensors()-Funktion geladen. Das SplitApp-Control wechselt dann die Masteransicht zu jenem Control, das in der Variable oMasterSensorsList hinterlegt ist. Die toMaster()-Funktion des SplitApp-Controls realisiert zudem auch den optischen Effekt der „nach Rechts“-Navigation. Handelt es sich bei dem ausgewählten Sensor um einen Raspberry Pi, wird die Detailansicht zu der Übersichtsanzeige der Raspberry Pis navigiert (siehe Abschnitt oRaspberryStatistics) und entsprechende Aktualisierungsvorgänge von Messdaten gestoppt bzw. gestartet. Handelt es sich bei dem Sensor um einen Smartmeter, wird in einem neuen Browsertab die Anwendung des vierten Szenarios (siehe Kapitel 7) geöffnet, da diese nicht Bestandteil des Sensorik Szenarios ist.

oMasterChannelList Der rötlich markierte Bereich auf Abbildung 5.41 stellt denjenigen Bereich dar, dessen Control in der Variable *oMasterChannelList* gespeichert ist. Das Page-Control besitzt eine aggregierende Eigenschaft, die sich *content* nennt. Mittels dieser Eigenschaft können dem Control weitere Controls hinzugefügt werden, um so komplexe Anwendungen zu gestalten. Die Messsensoren werden durch die Verwendung des Services *pushCurrentSensorValue.xsjs* abgerufen, die in dem Datemodell mit der ID *channelModel_1* gespeichert werden.
Die Messsensoren werden nur beim Auswählen eines Raspberry Pis angezeigt. Die Messsensoren selbst werden als eine Art Liste in einem List-Control dargestellt. Als konkrete, einzelne Listenelemente, die jeweils einen Messsensor darstellen, wird eine Erweiterung des ObjectListItem-Controls verwendet.

Abbildung 5.41.: Sensorik: Anzeige der Messsensoren

Wie in den vorherigen beiden Abschnitten erläutert, ist es wie in Listing 5.45 erforderlich, zu Beginn ein List-Control zu erzeugen. Der verwendete Service liefert nicht nur die aktuellen Messergebnisse der Sensoren, sondern auch festgelegte Tiefst- und Höchstgrenzen, die als Toleranzwerte für eine farbliche Darstellung als Orientierung dienen, zurück. Um die Tiefst- und Höchstgrenzen nun für ein Listenelement in einem Control hinterlegen zu können, erfordert es, dass das ObjectListItem-Control um die Eigenschaften *min* und *max* erweitert werden muss, da es diese Eigenschaft selbst nicht besitzt. Die Erweiterung des Controls zeigt Listing 5.46.

```
18 /**
19  * Erzeugen einer Liste zum Anzeigen der Messsensoren. Die einzelnen
    Listenelemente werden mit der bindItems()-Funktion an das Datenobjekt
    mit der ID "channelModel_1" gebunden.
20  */
21   var oChannelList = new sap.m.List("channelList_1",{});
```

Listing 5.45: Sensorik: Erzeugen eines List-Controls für die Messsensoren

```
30 /**
31  * Erzeugen eines Custom-Controls. Die Klasse ObjectListItem wird hier
    um die Properties min und max erweitert, da diese Werte fuer
32  * die Farbliche Darstellung der Messwerte benoetigt werden. Diese
    Properties koennen an ein Datenobjekt gebunden werden.
33  */
34
```

```
35
36    sap.m.ObjectListItem.extend("extendedObjectListItem",{
37      metadata:{
38        properties:{
39          "min":"",
40          "max":"",
41        }
42      },
43      renderer:{}
44    });
```

Listing 5.46: Sensorik: Erweitern des ObjectListItem-Controls

```
45  /**
46   *Als Listenelemente werden Objekte des Custom-Controls "
     extendedObjectListItem" erzeugt. Die einzelnen Listenelemente werden an
      das Datanobjekt mit der ID "channelModel\_1" gebunden.
47   *Dabei wird ein Template verwendet das alle Elemente des Objekts
     durchlaeuft und anschliessend anzeigt. Das richtige Icon liefert die
     getChannelIcon(channel)-Funktion.
48   *Beim betaetigen eines Listenelements wird die onChannelPress()-
     Funktion ausgefuehrt. Diese Liste dient zum Anzeigen der jeweiligen
     Sensoren die an einem Pi angeschlossen sind.
49   */
50    oChannelList.bindItems({
51      path:    'channelModel_1>/sensors',
52      template: new extendedObjectListItem({
53        type :  "Navigation",
54        title:  '{channelModel_1>channel}',
55        icon: {
56          path: 'channelModel_1>channel',
57          formatter: function(channel){
58            return oController.getChannelIcon(channel);
59          },
60        },
61        number: '{channelModel_1>value}',
62        numberUnit: '{channelModel_1>unit}',
63        min: '{channelModel_1>min}',
64        max: '{channelModel_1>max}',
65        press: [oController.onChannelPress,oController],
66        tap: function(){
67          oSplitApp.toDetail("detailPage");
68        }
69      })
70    });
```

Listing 5.47: Sensorik: Data-Binding der Listenelemente für die Messsensoren

Die einzelnen Listenelemente werden nun, wie in Listing 5.47 aufgeführt, wieder mit Hilfe einer *template*-Funktion iterativ erzeugt. Als Listenelement wird das erweitere ObjectListItem-Control, das Custom-Control *extendedObjectListItem* verwendet. Beim Auswählen eines Messsensors wird die onChannelPress()-Funktion im Controller aus-

geführt. Zudem wird sofort auf die dazugehörige Detailansicht navigiert (siehe Abschnitt oDetailPage). Die onChannelPress()-Funktion initiiert das setzten der Werte im ObjectHeader-Control (siehe Abbildung 5.49 im Abschnitt oDetailPage). Zudem werden die (grafischen) Eigenschaften der Chart im Chart-Bereich festgelegt und die Daten für den Chart-, Info- und Warning-Bereich (siehe Abschnitt oDetailPage) geladen. Das letztendliche List-Control, das in der Variable *oChannelList* gespeichert ist, wird anschließend der aggregierenden Eigenschaft (siehe Listing 5.48) des Page-Controls (oMasterChannelList) übergeben.

```
126  /**
127   * Erzeugen eines Page-Objekts zum Anzeigen der Messsensoren. Dem Inhalt
       des Content-Bereichs wird die Liste der Messsensoren hinzugefuegt. Da
      beim betaetigen eines
128   * Raspberry-Pis auf diese Seite navigiert wird, ist deshalb der
      Navigationsbutton aktiviert. Dieser fuehrt beim Betaetigen die
      onNavButtonPressValues()-Funktion aus.
129   * Der Controller binhaltet mehrere onNavButtenPress...-Funktionen damit
       immer genau klar ist, vom welcher ebene (Sensor-Kategorie, Sensor,
      weitere an den Sensor angeschlossene
130   * Sensoren wie Temperaturfuehler etc.) zurueck navigiert wird.
131   * Der Header-Bereich binhaltet zusaetzlich einen Dashboard-Button der
      beim Betaetigen die backToDashboard()-Funktion ausfuehrt.
132   */
133    var oMasterChannelList = new sap.m.Page("sensorListMaster",{
134      title: "Alle Sensoren",
135      showNavButton: true,
136      navButtonPress:[oController.onNavButtonPressValues,oController],
137      content:[oChannelList],
138      headerContent:[
139                     new sap.m.Button("homeButton_1",{
140                       icon: "sap-icon://home",
141                       press:[oController.pressBackToHomescreen,
     oController]
142                     })
143      ]
144    });
```

Listing 5.48: Sensorik: Übergeben der Listenelemente (Messsensoren) an das Page-Control

Detailansicht

Dieser Abschnitt dokumentiert alle Elemente der Detailansicht (siehe Abbildung 5.30), die auf Abbildung 5.31 aufgeführt sind im Detail. Neben den grafischen Elementen wird Bezug auf die wesentliche Logik genommen. Dieser Bereich der Ansicht dient der Darstellung von detaillierteren Informationen, sei es über die dahinter liegende HANA-Datenbank oder der vorhandenen Sensoren. Zu Beginn jedes Abschnitts folgt zuerst eine Abbildung, die den dokumentierten Bereich grafisch darstellt und eine Orientierungshilfe gibt. Komplexere Ansichten sind nochmals in ihre Komponenten unterteilt.

oInitialPage

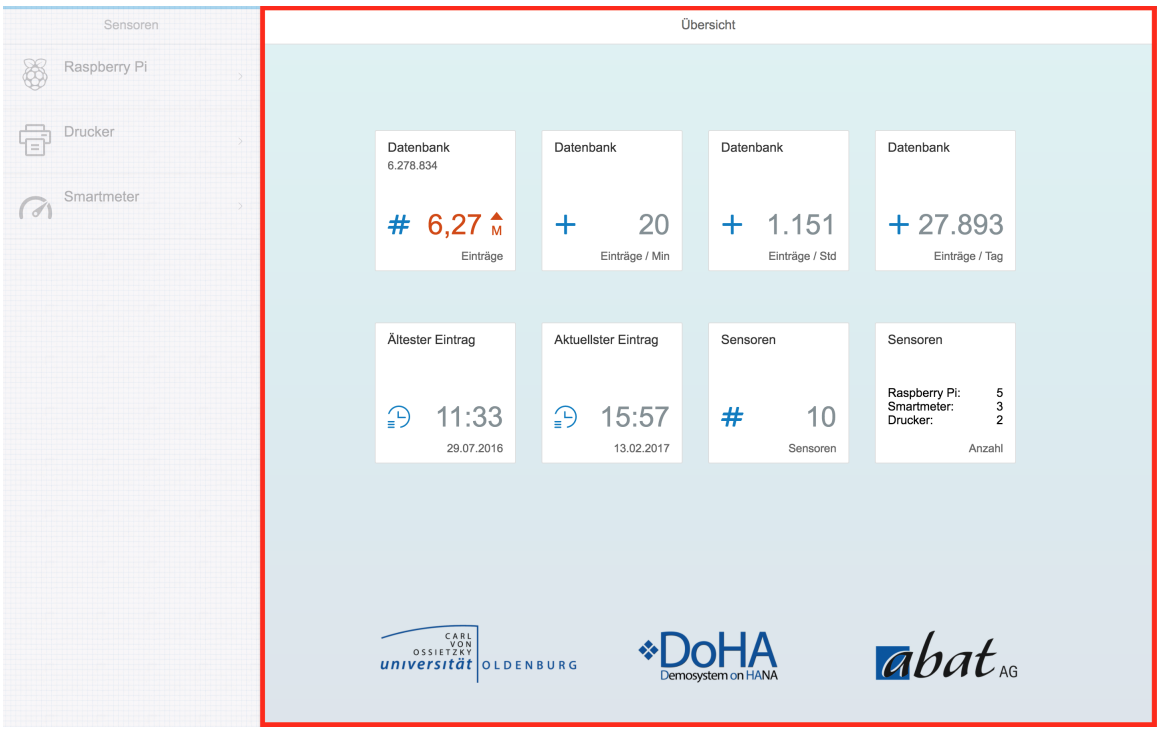


Abbildung 5.42.: Sensorik: Übersicht der Datenbankstatistiken

Abbildung 5.42 zeigt das Control, welches in der Variable *oInitialPage* gespeichert ist. Der strukturelle Aufbau kann der Abbildung 5.33 im Abschnitt Zyklischer Datenabruf entnommen werden. Dieses Control ist im SplitApp-Control *oSplitApp* als initiale Detailansicht festgelegt. Diese Detailansicht dient dazu, Informationen über die darunter liegende HANA-Datenbank anzuzeigen, deren Daten sich zyklisch aktualisieren. Zur Darstellung wurde eine Kachel-Ansicht gewählt, die für jede Kachel durch ein GenericTile-Control implementiert ist. Datenlieferant ist der Service *pushStatisticValues.xsjs* und nutzt das Datenmodell mit der ID *statisticValuesModel_1*. Die „Auswählbarkeit“ wurde mithilfe von CSS (siehe Abschnitt css) deaktiviert. Die nachfolgende Tabelle beschreibt in komprimierter Form alle Kacheln:

Tabelle 5.10.: Sensorik: Übersicht über die Kacheln auf der Seite der Datenbankstatistiken

Variable	Beschreibung
oDbCountTile	Zeigt die Anzahl der gesamten Datenbankeinträge an.
oEntriesPmTile	Zeigt an, wie viele Einträge pro Minute hinzukommen.

Tabelle 5.10.: Sensorik: Übersicht über die Kacheln auf der Seite der Datenbankstatistiken

Variable	Beschreibung
oEntriesPhTile	Zeigt an, wie viele Einträge pro Stunde hinzukommen.
oEntriesPdTile	Zeigt an, wie viele Einträge am Tag hinzukommen.
oOldestEntryTile	Zeigt an, von wann der älteste Eintrag ist.
oLatestEntryTile	Zeigt an, von wann der aktuellste Eintrag ist.
oSensorCountTile	Zeigt an, wie viele Sensoren registriert sind.
oAllSensorsTile	Zeigt an, in welcher Menge welcher Sensor registriert ist.

Das Erzeugen einer Kacheln ist, bis auf den, der in der Variable *oAllSensorsTile* hinterlegt ist, gleich aufgebaut. Listing 5.49 zeigt das Erzeugen der Kachel zum Anzeigen aller Datenbankeinträge. Alle weiteren Kacheln werden im Rahmen dieser Dokumentation, bis auf den, der in der Variable *oAllSensorsTile* gespeichert ist, nicht näher erläutert. Jede Kachel verfügt über eine Header- und Subheader-Eigenschaft. In der Header-Eigenschaft wird eine Überschrift angegeben. Der Inhalt jeder Kachel wird in der aggregierenden Eigenschaft *tileContent* festgelegt. Die in Listing 5.49 erzeugte Kachel wurde deren Eigenschaft *subheader* als einzige noch eine weitere Information hinterlegt. Die Gesamtanzahl der Einträge wird zudem noch mittels einer *fomatter*-Funktion formatiert. Die Funktion convertDigit() trennt, je nach übergebenem Flag, nach jeder hundertstel-Angabe mit einem Punkt (z. B. 1000000 wird zu 1.000.000).

```
146 /**
147  * Erzeugen einer Kachel zum Anzeiger aller Datenbankeintraege. Nutzt
     das Datenobjekt mit der ID "statisticValuesModel_1".
148  */
149   var oDbCountTile = new sap.m.GenericTile({
150     header: "Datenbank",
151     subheader:{
152       path: 'statisticValuesModel_1>/db_count',
153       formatter: function(value){
154         return oController.convertDigit(value,0);
155       }
156     },
157     tileContent:[
158                  new sap.m.TileContent({
159                    footer: "Eintraege",
160                    content:[
161                             new sap.m.NumericContent({
162                             animateTextChange: false,
163                               truncateValueTo: 4,
```

```
164                                 valueColor: "Critical",
165                                 indicator: "Up",
166                                 scale: "M",
167                                 value: {
168                                   path: "statisticValuesModel_1>/db_count"
    ,
169                                   formatter: function(value){
170                                     var temp = value/1000000;
171                                     return oController.convertDigit(temp
    ,1);
172                                   }
173
174                                 },
175                                 icon: "sap-icon://number-sign",
176                               }),
177                     ]
178                   })
179       ]
180     });
```

Listing 5.49: Sensorik: Erzeugen einer Kachel für die Datenbankstatistiken

Damit alle Kacheln geordnet angezeigt und zudem beim Verkleinern des Browsers automatisch gruppiert werden, werden die Kacheln in einen entsprechenden Container hinzugefügt, der diese Anforderungen bereits realisiert. Das Framework bietet zur Zeit keinen Container, in dem GenericTile-Controls gespeichert werden können. Es ist jedoch zulässig, dass CustomTile-Controls in einem Container gespeichert werden können. Listing 5.50 zeig, wie die Anwendung des Sensorik Szenarios dieses Problem löst.

```
338   /**
339    * Erzeugen eines Tile-Containers indem alle GenericTiles
       zusammengefasst und zentriert angezeigt werden. Abstaende werden
       automatisch dem Fenster
340    * angepasst. Kacheln ordnen sich automatisch an. "TwoByOne"-Kacheln
       sind in deinem "TileContainer" nicht moeglich.
341    */
342     var oTileContainer = new sap.m.TileContainer({
343       height: "85%",
344       tiles:[
345             new sap.m.CustomTile({
346               content:[oDbCountTile]
347             }),
348             new sap.m.CustomTile({
349               content:[oEntriesPmTile]
350             }),
351             new sap.m.CustomTile({
352               content:[oEntriesPhTile]
353             }),
354             new sap.m.CustomTile({
355               content:[oEntriesPdTile]
356             }),
357             new sap.m.CustomTile({
358               content:[oOldestEntryTile]
```

```
359                 }),
360                 new sap.m.CustomTile({
361                   content:[oLatestEntryTile]
362                 }),
363                 new sap.m.CustomTile({
364                   content:[oSensorCountTile]
365                 }),
366                 new sap.m.CustomTile({
367                   content:[oAllSensorsTile]
368                 })
369       ]
370     });
```

Listing 5.50: Sensorik: Hinzufügen von generischen Kacheln in einen Container

Listing 5.50 zeigt das Hinzufügen von mehreren Kacheln in die aggregierende Eigenschaft *tiles*. Das TileContainer-Control sorgt für eine optimale Darstellung. Da das GenericTile-Control zur Zeit in keinem Container-Control hinterlegt werden kann, wurde zuvor ein CustomTile-Control erzeugt. In dessen aggregierenden Eigenschaft *content* wird nun eine generische Kachel hinzugefügt. CustomTile-Controls können mit dem TileContainer-Control aggregiert werden. Die CustomTile-Controls haben sich für die Anzeige der Datenbankstatistiken nicht geeignet, sodass die Wahl auf das GenericTile-Control gefallen ist.
Eine gesonderte Erläuterung Bedarf es der Kachel, die in der Variable *oAllSensorsTile* gespeichert ist. Diese wird innerhalb der View erzeugt und lediglich mit einer Überschrift in der *header*-Eigenschaft versehen (siehe Listing 5.51). Das Data-Binding findet im Controller statt.

```
320   /**
321    * Erzeugen einer Kachel um anzuzeigen wie viele Sensoren von welcher
       Art in der Datenbank registriert sind. Das Data-Binding findet im
       Controller in der Funktion createHtmlForTile() statt. Diese Funktion
       wird in der onInit()-Funktion aufgerufen.
322    */
323
324     var oAllSensorsTile = new sap.m.GenericTile("chartTile_1",{
325       header: "Sensoren"
326     });
```

Listing 5.51: Sensorik: Erzeugen einer Kachel (oAllSensorsTile)

Um die Anzeige so zu verwirklichen, wie sie in Abbildung 5.43 dargestellt wird, war es erforderlich, die gewünschte Modifikation mittels CSS vorzunehmen. Durch das Aggregieren der Eigenschaft *tileContent* mit einem Tile-Content-Control ist es nicht möglich, die numerischen Angaben mittels CSS (siehe Abschnitt 5.3.4) anzusprechen, da sie von keinen HTML-Tags eingeschlossen werden, welche wiederum als Selektoren dienen könnten. Die createHtmlForTile()-Funktion im Controller ruft die entsprechenden Daten aus dem Datenobjekt ab, speichert diese in einem selbst definierten String, wobei die Daten zwischen HTML-Tags gesetzt werden und fügt sie anschließend der aggregierenden Eigenschaft *tileContent* der Kachel in der Variable oAllSensorsTile hinzu. Die

createHtmlForTile()-Funktion wird in der onInit()-Funktion gleich nach dem Rendern der Anwendung aufgerufen und wird schon vorher erzeugt, obwohl sie nicht gebraucht wird. Listing 5.52 zeigt den Quellcode der createHtmlForTile()-Funktion.

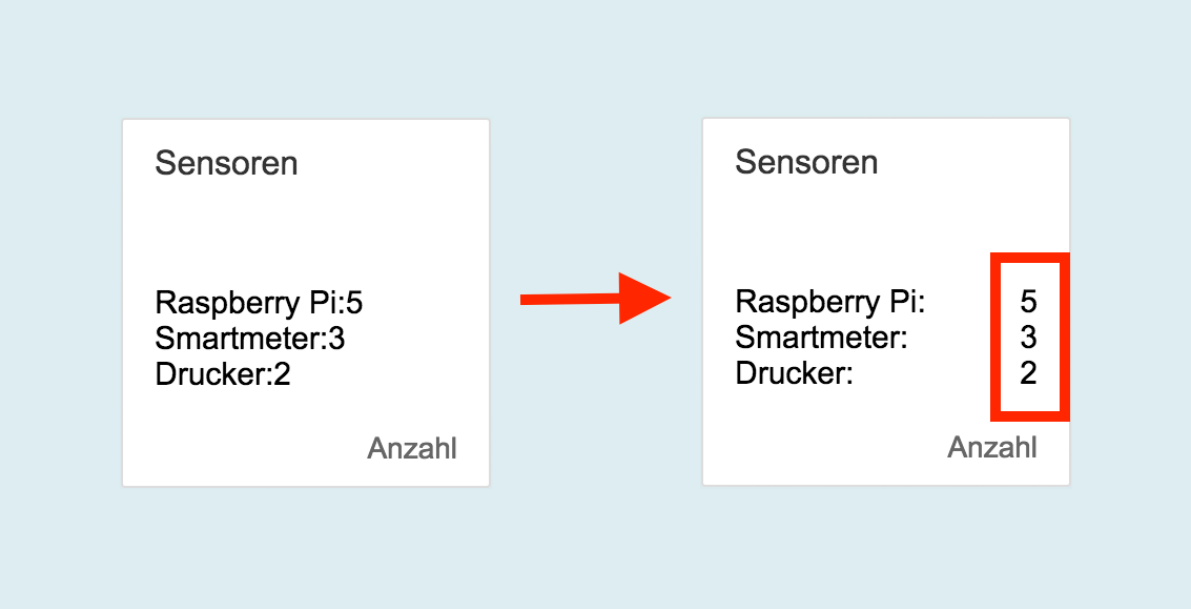


Abbildung 5.43.: Sensorik: Verbesserte Darstellung einer generischen Kachel (oAllSensorsTile)

```
1681 /**
1682  * Eine Funktion die zusaetzliche HTML-Tags in die Anwendung hinzufuegt.
        Betrifft eine Kachel auf der Startseite der Datenbankstatistiken.
1683  * Die Kachel welche die Anzahl und die Art der vorhandenen Sensoren
        anzeigt werden <span>-Tags hinzugefuegt damit die nummerische
1684  * Darstellung rechtbuendig angezeigt werden kann.
1685  */
1686
1687   createHtmlForTile: function(){
1688     var oModel = this.getView().getModel("statisticValuesModel_1");
1689
1690     var piName = oModel.getProperty("/types/0/type");
1691     var piCnt = oModel.getProperty("/types/0/count");
1692     var pName = oModel.getProperty("/types/1/type");
1693     var pCnt = oModel.getProperty("/types/1/count");
1694     var sName = oModel.getProperty("/types/2/type");
1695     var sCnt = oModel.getProperty("/types/2/count");
1696
1697     var htmlStr = "<span>"+piName+":<space></span>"+"<span>"+piCnt+"</span
       ><br><span>"+pName+":<space></span><span>"+pCnt+"</span><br>"+"<span>"+
       sName+":<space></span><span>"+sCnt+"</span>";
1698
1699     var oHtml = new sap.ui.core.HTML({
1700       content: htmlStr
```

```
1701      });
1702
1703      var oTileContent = new sap.m.TileContent({
1704        footer: "Anzahl",
1705        content:[oHtml]
1706      });
1707
1708      this.oAllSensorsTile.addTileContent(oTileContent);
1709    }
```

Listing 5.52: Sensorik: Quellcode der createHtmlForTile()-Funktion

oRaspberryStatistics

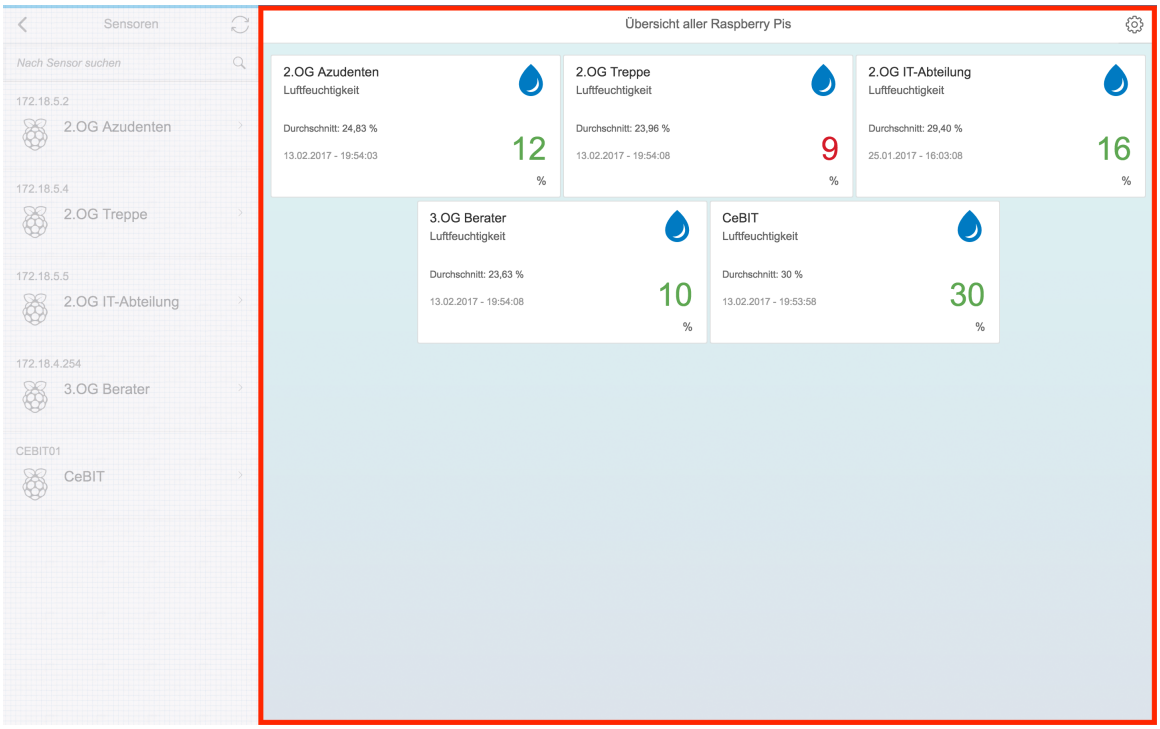


Abbildung 5.44.: Sensorik: Übersicht über alle Raspberry Pis (oRaspberryStatistics)

Abbildung 5.37 zeigt das Control, welches in der Variable *oRaspberryStatistics* gespeichert ist. Der strukturelle Aufbau kann der Abbildung 5.37 im Abschnitt 5.3.4 entnommen werden. Diese Detailansicht dient dazu, eine Übersicht über alle Raspberry Pis anzuzeigen. Die Darstellung erfolgt in einer Kachelansicht. Pro Raspberry Pi wird ein SlideTile-Control erzeugt. Dieser realisiert eine Kachel, deren Inhalte sich verändern. Der Inhalt eines SlideTile-Controls ist ein GenericTile-Control. Datenlieferanten sind die Services *pushStatisticValues.xsjs* sowie *pushStatisticValues.xsjs* und befüllen die Datenmodelle mit den IDs *statisticValuesModel_1* und *rpStatisticsModel_1*. Die „Auswählbarkeit“ wurde mithilfe von CSS (siehe Abschnitt css) deaktiviert.

Das Erzeugen der SlideTile-Controls geschieht in der View (siehe Listing 5.53) wieder mittels einer *template*-Funktion. Mit dem angegebenen Pfad im Datenobjekt wird so oft ein SlideTile-Control erzeugt, wie es Raspbery Pis gibt. Diese sind vorerst für den Anwender nicht sichtbar, da sie mit keinem anderen Control aggregiert sind. Das Data-Binding sowie das Erzeugen generischen Kacheln erfolgt im Controller innerhalb der setRpStatistics()-Funktion. Zum besseren Verständnis ist Listing 5.54 zu betrachten.

```
999  /**
1000  * Anlegen eines FixFlex-Layouts zum Darstellen der Slide-Kachel des
      Raspberry Pis. Fuer die Anzahl der Slide-Kacheln richtet sich der
      fixContent nach
1001  * der Anzahl der Raspberry Pis im Model mit der ID "rpStatisticsModel_1
      ". Als Template wird das SchlideTile-Control genutzt und wird fuer
      jeden vorhanden
1002  * Pi einmal erzeugt.
1003  */
1004   var oRpStatisticsLayout = new sap.ui.layout.FixFlex("
      responsiveLayout_1",{
1005     fixContent:{
1006       path:"rpStatisticsModel_1>/output",
1007       template: new sap.m.SlideTile()
1008     }
1009   });
```

Listing 5.53: Sensorik: Erzeugen der SlideTile()-Controls

Die Funktion, die in Listing 5.54 angegebenen ist, befüllt iterativ die zuvor erzeugten SlideTile-Controls mit GenericTile-Controls durch dessen aggregierende Eigenschaft *tiles*. Dieser Vorgang wurde deshalb so gewählt, um ein dynamisches Erzeugen der Kacheln zu erreichen. Wenn sich die Anzahl der Raspberry Pis verringert oder erhöht, muss die Anwendung lediglich einmalig neu geladen werden, damit sich die Übersicht den neuen Gegebenheiten anpasst. So muss nichts manuell im Quellcode verändert werden.
Die Anzahl der Raspberry Pis wird dem Datenmodell mit der ID *statisticsValuesModel_1* entnommen. Dieser dient zugleich als Abbruchkriterium für das Durchlaufen der SlideTile-Controls. Die Anzahl registrierter Raspberry Pis ist auch die Anzahl der SlideTile-Controls, die zuvor in der View (siehe Listing 5.53) erzeugt worden sind. Die ID (automatisch durch das Framework festgelegt) des ersten SlideTile-Controls muss über die Entwicklerwerkzeuge des Browsers identifiziert werden. Dieser wird dann in der inneren Schleife iterativ erhöht, damit alle durchlaufen werden.
Die innere Schleife durchläuft jeden Messsensor und bindet die Eigenschaften an einem *extendedGenericTile*-Control, der dann dem SlideTile-Control iterativ hinzugefügt wird. Das *extendesGenericTile*-Control ist ein erweitertes Custom-Control des GenericTile-Controls. Dadurch das die Eigenschaften explizit an das Datenobjekt gebunden werden, werden beim Verändern des Datenmodells auch die Messwerte aktualisiert. Die Anzahl der Messsensoren ist im Rahmen dieser Anwendung fest mit 4 angegeben. Fällt ein Messsensor weg oder kommt ein neuer hinzu, so kann sich die Anwendung nicht mehr aufbauen da, auf den entsprechenden Eintrag im Datenmodell nicht mehr zugegriffen werden kann. Dieser muss an dieser Stelle manuell angepasst werden.

Wenn bei der Programmierung im Laufe der Zeit neue Kacheln erzeugt werden, die vor den Kacheln der Raspberry Pi-Übersicht erzeugt werden, kann es vorkommen, dass sich die ID aus dem DOM der SlideTile-Controls ändern und sich die Anwendung wieder nicht aufbauen kann, da die angegebene ID (__tile15-responsiveLayout_1-**[0-4]**) nicht existiert. Dann muss eine Anpassung manuell im Quellcode vorgenommen werden.

```
1372 /**
1373  * Eine Funktion zum Erzeugen und Laden der Daten furr die generischen
        Kacheln in den Slide-Kacheln. Die Slide-Kacheln werden in der View
        erzeugt,
1374  * diese sind jedoch initial leer und fuer den User nicht sichtbar. Diese
        Funktion befuellt die Slide-Kacheln mit generischen Kacheln und
1375  * realisiert das Data-Binding.
1376  */
1377   setRpStatistics: function(){
1378     var self = this;
1379     //Speichert die Anzahl der Raspberry Pis
1380     var rpAmount = this.getView().getModel("statisticValuesModel_1").
      getProperty("/types/0/count");
1381
1382     var oTile;
1383
1384     //Die aeussere Schleife wird so oft durchlaufen wie es Raspberry Pis
      gibt
1385     for(var i=0;i<rpAmount;++i){
1386       //In oTile wird schrittweise jede Slide-Kachel gespeichert. Die
      entsprechende ID ist dem DOM zu entnehmen.
1387       oTile = sap.ui.getCore().getElementById("__tile15-
      responsiveLayout_1-"+i);
1388
1389       //Der Zeitstempel und die Location des aktuellen Eintrages wird
      abgespeichert
1390       var timestamp = this.getView().getModel("rpStatisticsModel_1").
      getProperty("/output/"+i+"/timestamp");
1391       var location = this.getView().getModel("rpStatisticsModel_1").
      getProperty("/output/"+i+"/location");
1392
1393       //Die innere Schleife wie so oft durchlaufen wie es Messensoren an
      einem Pi gibt damit jeder Messsensor als generische Kachel der Slide-
      Kachel hinzugefuegt werden kann
1394       for(var j=0;j<4;++j){
1395         //Speichert den Messwert und den Durchschnitt ab
1396         var value = this.getView().getModel("rpStatisticsModel_1").
      getProperty("/output/"+i+"/values/"+j+"/value");
1397         var avg   = this.getView().getModel("rpStatisticsModel_1").
      getProperty("/output/"+i+"/values/"+j+"/avg");
1398
1399         //Eine temporaere generische Kachel wird erzeugt. Die Daten des
      durchlaufenen Messsensors wird an die Eigenschaften der Kachel gebunden
      .
1400         var tempTile = new extendedGenericTile({
1401           headerImage: {
```

```
1402              path: 'rpStatisticsModel_1>/output/'+i+'/values/'+j+'/channel
    ',
1403              formatter: function(channel){
1404                return self.getChannelIcon(channel);
1405              }
1406            },
1407            frameType: "TwoByOne",
1408            header: '{rpStatisticsModel_1>/output/'+i+'/location}',
1409              subheader: '{rpStatisticsModel_1>/output/'+i+'/values/'+j+'/
    channel}',
1410              min: '{rpStatisticsModel_1>/output/'+i+'/values/'+j+'/min}',
1411              max: '{rpStatisticsModel_1>/output/'+i+'/values/'+j+'/max}',
1412              tileContent:[
1413                            new sap.m.TileContent({
1414                              unit: '{rpStatisticsModel_1>/output/'+i+'/
    values/'+j+'/unit}',
1415                              content:[
1416                                       new sap.m.FeedContent({
1417                                         contentText:{
1418                                           parts:[
1419                                                  {path: '
    rpStatisticsModel_1>/output/'+i+'/values/'+j+'/avg'},
1420                                                  {path: '
    rpStatisticsModel_1>/output/'+i+'/values/'+j+'/unit'}
1421                                           ],
1422                                           formatter: function(avg,unit){
1423                                             return "Durchschnitt: "+self.
    convertDigit(avg,2)+" "+unit
1424                                           }
1425                                         },
1426                                         value: '{rpStatisticsModel_1>/
    output/'+i+'/values/'+j+'/value}',
1427                                         subheader: {
1428                                           path: 'rpStatisticsModel_1>/
    output/'+i+'/timestamp',
1429                                           formatter: function(timestamp){
1430                                             return self.convertTimestamp(
    timestamp,0);
1431                                           }}})
1432                              ]
1433                            })
1434              ]
1435        });
1436        //Die temporaer erzeugte Kachel wird dem SlideTile-Control
    hinzugefuegt.
1437        oTile.addTile(tempTile);
1438      }
1439    }
1440
1441  },
```

Listing 5.54: Sensorik: Dynamisches erzeugen von generischen Kacheln (oRaspberryPiStatistics)

oDashboard

Abbildung 5.45.: Sensorik: Übersicht aller Messsensoren als Dashboard (oDashboard)

Abbildung 5.45 zeigt das Control, welches in der Variable *oDashboard* gespeichert ist. Der strukturelle Aufbau der Datensätze kann der Abbildung 5.34 im Abschnitt 5.3.4 entnommen werden. Diese Detailansicht dient einer kurzen zeitlichen Übersicht über den Verlauf von Messergebnissen für jeden an einem Raspberry Pi angeschlossenem Messsensor. Die Darstellung erfolgt in einer Chart-Ansicht wobei jeder Messsensor durch ein Linien-Chart dargestellt wird. Die Daten aktualisieren sich zyklisch. Als Datenlieferant wird der Service *pushDashValuesForAmount.xsjs* verwendet, der jeweils die ID des Rapsberry Pis, der Channel des Messsensor und die Menge an abzurufenden Daten mitgegeben werden. Für jedes Chart erfolgt jeweils ein eigener AJAX-Abruf in der entsprechenden poll-Funktion. Tabelle 5.11 zeigt auf, welche Chart welches Datenmodell nutzt.

Tabelle 5.11.: Sensorik: Übersicht über die Datenmodelle des RP-Dashboards (oDashboard)

Chart	**ID des Datenmodells**
Temperatur	tempDashModel_1
Luftfeuchtigkeit	humDashModel_1

Tabelle 5.11.: Sensorik: Übersicht über die Datenmodelle des RP-Dashboards (oDashboard)

Chart	ID des Datenmodells
CO2	co2DashModel_1
Luftdruck	airDashModel_1

Ein Chart, die als visualization initialization (Viz)Frame bezeichnet wird, besteht dabei aus einem sap.viz.ui5.controls.VizFrame-Control (siehe Listing 5.55), einem sap.viz.ui5.data.FlattenedDataset-Datensatz (siehe Listing 5.56), der die Daten enthält und die *sap.viz.ui5.controls.common.feeds.FeedItem()*-Controls (siehe Listing 5.57) jeweils für die x- und y-Achse setzt. Die Beschreibung dieser Elemente ist den Kommentaren aus dem Quellcode zu entnehmen. Die Erzeugung der restlichen Linien-Charts erfolgt analog zu denen in den Listings 5.55-5.57. Durch die *initDashboardProperties()*-Funktion werden weitere Eigenschaften, wie die farbliche Gestaltung oder die Anzeige der Legende und Beschriftungen, gesetzt. Die Dokumentation der VizFrame-Charts befindet sich nicht in der SAPUI5-API-Dokumentation sondern ist unter `https://sapui5.netweaver.ondemand.com/sdk/docs/vizdocs/index.html` zu finden.

```
688  /**
689   * d_width, d_height und d_type legen die Breite, Hoehe und den Typ
      aller Dashboard-Charts fest.
690   */
691    var d_width = "100%";
692    var d_height = "25%";
693    var d_type= "line";
694
695  /**
696   * DASHBOARD
697   * Die Charts fuer das Dashboard werden nicht dynamisch erzeugt. Fuer
      jeden Messsensor der neu hinzukommt (dynamisch) ist manuell
698   * ein Chart (inkl. Datenmodell und AJAX-Call) anzulegen. Wenn alle
      Charts die selbe Breite, Hoehe und und den Typ haben sollen koennen die
       Variablen
699   * d_width, d_height und d_type verwendet werden. Ansonsten sind diese
      manuell anzugeben.
700   */
701
702    /*
703     * TEMPERATUR-CHART
704     * Erzeugen des Temperatur-Charts und festlegen der Breite, Hoehe und
      des Typs.
705     */
706    var oTempDashboard = new sap.viz.ui5.controls.VizFrame("
      tempDashChart_1",{
```

```
707       width: d_width,
708       height: d_height,
709       vizType: d_type,
710     });
```

Listing 5.55: Sensorik: Erzeugen eines VizFrame-Controls (oDashboard)

```
712   /**
713    * Erzeugen eines Datensatzen zum Fuellen des Charts mit Daten. Die
       Daten kommen aus dem Datenobjekt mit der ID "tempDashModel_1".
714    */
715     var oTempDataset = new sap.viz.ui5.data.FlattenedDataset({
716       measures:[{
717           name: "Temperatur",
718           value: "{tempDashModel_1>value}"
719       }],
720       dimensions:[{
721         name: "Zeitraum",
722         value: "{tempDashModel_1>timestamp}"
723       }],
724       data:{
725         path: "tempDashModel_1>/values"
726       }
727     });
```

Listing 5.56: Sensorik: Erzeugen eines Datensatzen für ein VizFrame (oDashboard)

```
729   /**
730    * Erzeugen eines FeedItems (y-Achse). Hier wird festegelegt welche
       Daten auf welcher Achse des Charts dargestellt werden sollen. Die
       Eintraege in der Eigenschaft
731    * "values" muessen mit den Eintraegen des Datensatzen uebereinstimmen (
       name-Property). Ansonsten kann sich das Chart nicht aufbauen. Die
       Achsenbeschriftungen,
732    * sowie die Legende, koennen jederzeit ueber die VizFrame-Properties
       geaendert werden (siehe Doku: https://sapui5.netweaver.ondemand.com/sdk
       /docs/vizdocs/index.html).
733    */
734     var temp_feedValueAxis = new sap.viz.ui5.controls.common.feeds.
       FeedItem({
735           "uid": "valueAxis",
736           "type": "Measure",
737           "values": ["Temperatur"]
738     });
739
740   /**
741    * Erzeugen eines FeedItems (x-Achse). Hier wird festegelegt welche
       Daten auf welcher Achse des Charts dargestellt werden sollen. Die
       Eintraege in der Eigenschaft
742    * "values" muessen mit den Eintraegen des Datensatzen uebereinstimmen (
       name-Property). Ansonsten kann sich das Chart nicht aufbauen. Die
       Achsenbeschriftungen,
```

```
743     * sowie die Legende, koennen jederzeit ueber die VizFrame-Properties
        geaendert werden (siehe Doku: https://sapui5.netweaver.ondemand.com/sdk
        /docs/vizdocs/index.html).
744     */
745      var temp_feedCategoryAxis = new sap.viz.ui5.controls.common.feeds.
        FeedItem({
746             'uid': "categoryAxis",
747             'type': "Dimension",
748             'values': ["Zeitraum"]
749      });
750
751    /**
752     * Datensatz fuer das Temperatur-Chart setzen und FeedItems hinzufuegen.
753     */
754      oTempDashboard.setDataset(oTempDataset);
755      oTempDashboard.addFeed(temp_feedValueAxis);
756      oTempDashboard.addFeed(temp_feedCategoryAxis);
```

Listing 5.57: Sensorik: Erzeugen der Axenbeschriftungen für ein VizFrame (oDashboard)

oDetailPage

Abbildung 5.46.: Sensorik: Detailansicht eines Messsensors, Chart-Bereich (oDetailPage)

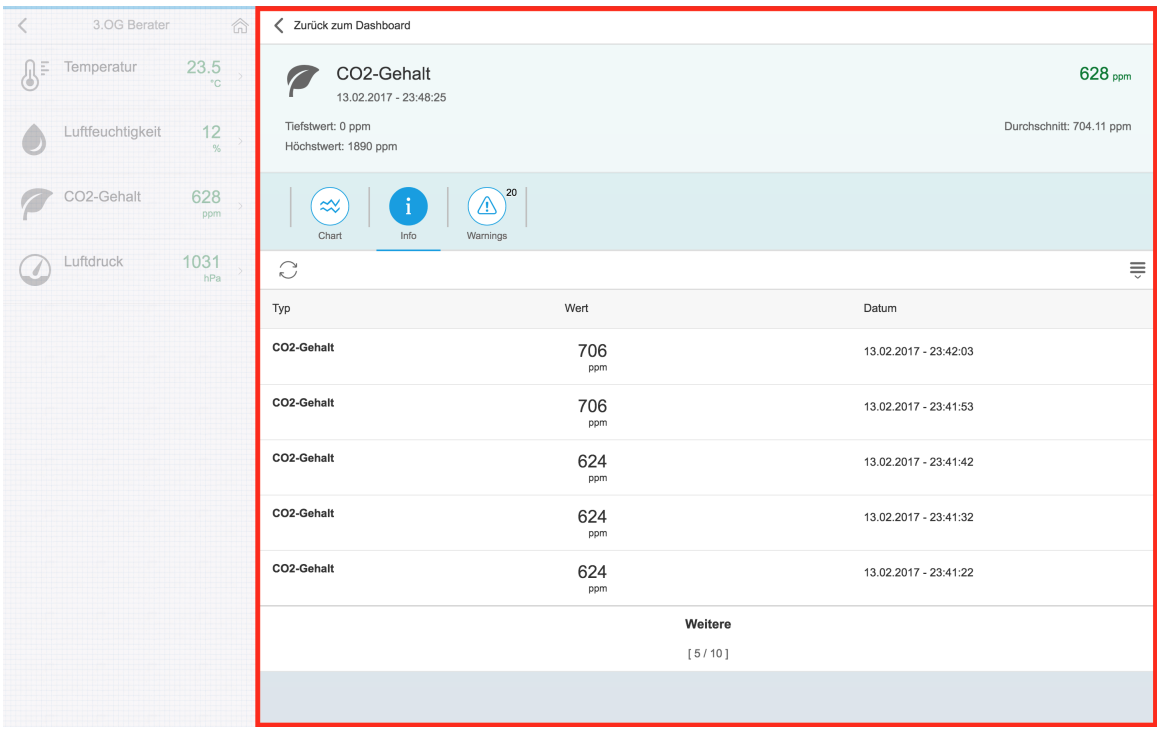


Abbildung 5.47.: Sensorik: Detailansicht eines Messsensors, Info-Bereich (oDetailPage)

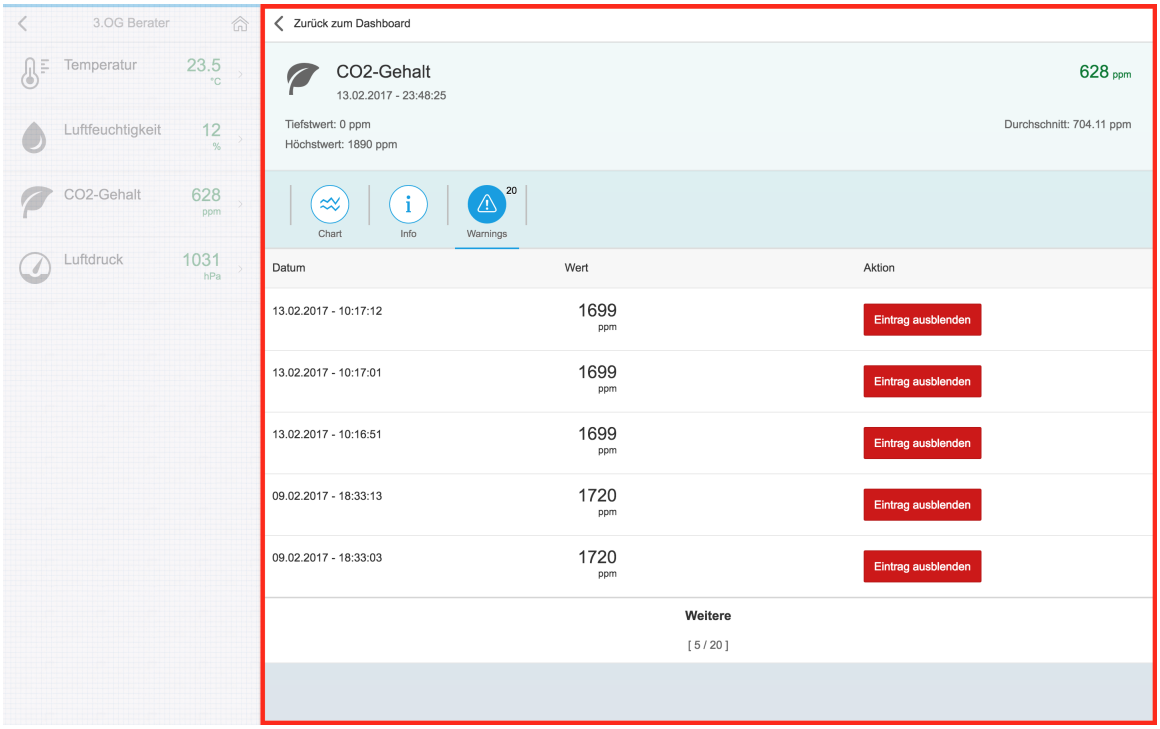


Abbildung 5.48.: Sensorik: Detailansicht eines Messsensors, Warnings-Bereich (oDetail-Page)

Die Abbildungen 5.46, 5.47 und 5.48 stellen das Control dar, welches in der Variable *oDetailPage* gespeichert ist. Diese Detailansicht dient der detaillierten Darstellung von Informationen und Messergebnissen über einen Messsensor. Die aggregierende Eigenschaft dieses Page-Controls fügt ein ObjectHeader- und IconTabBar-Control ein. Der Header-Bereich wird bereits im Abschnitt 5.3.4 (siehe Abbildung 5.34) kurz beschrieben. Als Datenlieferant dient der Service *pushMinMaxAvgForChannel.xsjs*, dem die ID des Raspberry Pis sowie der entsprechende Channel (Messsensor) mitgegeben werden muss. Damit kann der jemals gemessene Tiefst-, Höchst-, und Durchschnittswert (siehe Abbildung 5.49) eines Sensors für einen bestimmten Raspberry Pi abgerufen werden. Indirekt wird auch der Service *pushCurrentSensorValue.xsjs* genutzt. Diesem wird eine ID mitgegeben, die dann die aktuellen Messwerte für einen Raspberry Pi zurückliefert. Der Datenabruf erfolgt deswegen indirekt, weil lediglich beim Auswählen eines Messsensors auf dessen Datenobjekt zugegriffen wird um den Titel, den Wert, die Einheit und das Icon im Header anzupassen. Ein reines Data-Binding innerhalb der View ist nicht möglich, da sich die Werte des Headers dynamisch ändern müssen.

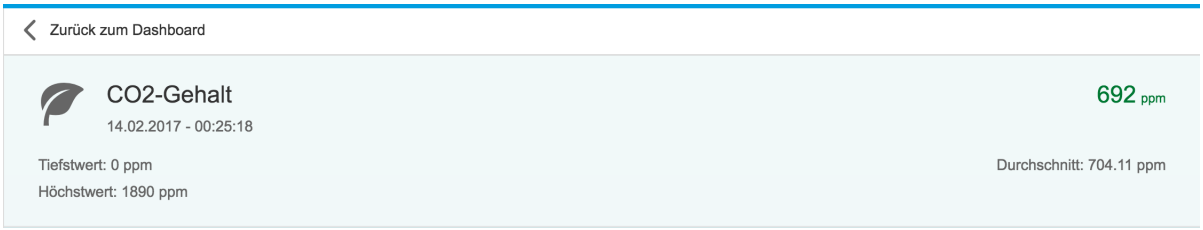


Abbildung 5.49.: Sensorik: Der Header der Detailansicht (oDetailPage)

Die setObjectHeaderValues-Funktion in Listing 5.58 wird jedes mal mit aufgerufen, wenn ein Press-Event (onChannelPress-Funktion im Controller) aus der Masteransicht der Messsensoren ausgelöst wird. Dabei wird auf die Daten des Messensors zugegriffen. Anschließend erfolgt das Data-Binding der Eigenschaften des Headers. Ein einfaches Setzten der Eigenschaften reicht nicht aus, da sich die Werte ansonsten nicht aktualisieren würden wenn die Daten des Datenmodells neu gesetzt werden. Zudem werden bei jedem Aufruf die Status neu gesetzt. Das Setzen des Status dient rein der besseren farblichen Darstellung der Messwerte.

```
294 /**
295  * Eine Funktion zum setzten der Properties des ObjectHeaders.
296  *
297  * @param oBinding
298  */
299  setObjectHeaderValues: function(oBinding){
300    var self  = this;
301    var min   = oBinding.getProperty("min");
302    var max   = oBinding.getProperty("max");
303    var value   = oBinding.getProperty("value");
304
305    this.oDetailObjectHeader.setBindingContext(oBinding);
```

```
306     this.oDetailObjectHeader.bindProperty("title","channelModel_1>/sensors
        /"+oBinding.getProperty("id")+"/channel");
307     this.oDetailObjectHeader.bindProperty("number","channelModel_1>/
        sensors/"+oBinding.getProperty("id")+"/value");
308     this.oDetailObjectHeader.bindProperty("numberUnit","channelModel_1>/
        sensors/"+oBinding.getProperty("id")+"/unit");
309     this.oDetailObjectHeader.setIcon(this.getChannelIcon(oBinding.
        getProperty("channel")));
310
311     this.oDetailObjectHeader.bindProperty("intro","channelModel_1>/
        timestamp",function(timestamp){
312       return self.convertTimestamp(timestamp,0);
313     });
314
315     if(Number(value)>=Number(min) && Number(value)<=Number(max)){
316       this.oDetailObjectHeader.setNumberState("Success");
317     }else{
318       if(Number(value)<min){
319         this.oDetailObjectHeader.setNumberState("Warning");
320       }else{
321         this.oDetailObjectHeader.setNumberState("Error");
322       }
323     }
324
325   },
```

Listing 5.58: Sensorik: Data-Binding des Headers der Detailansicht (oDetailPage)

Neben dem ObjectHeader-Control aggregiert das Page-Control ein IconTabBar-Control, dessen aggregierender Eigenschaft *item* wiederum ein IconTabFilter-Control für die Anzeige des Chart- (siehe Abbildung 5.46), Info- (siehe Abbildung 5.47) und Warnings-Bereichs (siehe Abbildung 5.48) hinzugefügt wurde. Listing 5.59 verdeutlicht dies. Zudem aggregiert die Eigenschaft *customHeader* eine Toolbar mit einem Button, der die Möglichkeit bietet, zurück zum Dashboard (siehe Abbildung 5.45) zu navigieren.

```
654   /**
655    * Erzeugen eines Page-Objekts. Stellt die Detail-Ansicht der Anwendung
       dar. Wird angezeigt beim Auswaehlen eines Messsensors. Im Content-
       Bereich ist der
656    * ObjectHeader mit den jeweilgen "Buttons" (IconTabFilter) fuer den
       Chart-, Info- und Extremwertebereich. Die Eigenschaft "
       applyContentPadding"
657    * entfernt die Aussenabstaende der Seite die ansonsten automatisch
       hinzugefuegt werden.
658    */
659     var oDetailPage = new sap.m.Page("detailPage",{
660       title: "Details",
661       customHeader:[
662                 new sap.m.OverflowToolbar({
663                   content:[
664                     new sap.m.Button("dashboardButton_1",{
665                       text:"Zurueck zum Dashboard",
666                       type:"Back",
```

```
667                     enabled: false,
668                       press:[oController.pressBackToDashboard,oController
    ]
669                   }),
670                 ]
671               })
672     ],
673     content: [oDetailObjectHeader,
674           new sap.m.IconTabBar("iconTabBar_1", {
675             expandable: false,
676             applyContentPadding:false,
677             items: [new sap.m.IconTabSeparator(),
678                     oIconTabChartButton,
679                     new sap.m.IconTabSeparator(),
680                     oIconTabInfoButton,
681                     new sap.m.IconTabSeparator(),
682                     oIconTabWarningButton,
683                     new sap.m.IconTabSeparator()
684               ]
685           })]
686   });
```

Listing 5.59: Sensorik: Erzeugen der Detailansicht der Messsensoren (oDetailPage)

Listing 5.60 verdeutlicht das Erzeugen des IconTabBar-Controls, dessen aggregierende Eigenschaft *content* die Variable *oChartContainer* beinhaltet. In der Variable oChartContainer wird ein einzelnes VizFrame Linien-Chart repräsentiert, das nach dem Muster wie im Abschnitt oDashboard erzeugt wird. Als Datenlieferant wird der Service *pushChartValuesForTime.xsjs* genutzt. Als Parameter sind diesem die ID des Raspberry Pis, die Channel-ID (Messsensor) sowie eine Stunden- oder Minutenangabe mitzugeben. Die Anwendung des Sensorik Szenarios lädt für den Chart-Bereich innerhalb der Detailansicht die Daten einer letzten ganzen Stunde. Der Service befüllt das Datemodell mit der ID *chartModel_1*.
Die IconTabBar-Controls des Info- sowie Warnings-Bereichs wird analog zu dem in Listing 5.60 erzeugt. Die aggregierende Eigenschaft des Info-Bereichs beinhaltet eine Tabelle, die den Service *pushTableValuesForChannel.xsjs* als Datenlieferant nutzt und das Datenmodell mit der ID *tableModel_1* verwendet. Der Warnings-Bereich nutzt den Service *pushExtremeValues.xsjs* als Datenlieferant und verwendet das Datenmodell mit der ID *extremeTableModel_1*. Dieser beinhaltet ebenfalls eine Tabelle zum Anzeigen von Extremwerten.

```
644  /**
645   * Erzeugen eines IconTabFilters. Dieser wird als ein "Button" unterhalb
        des ObjectHeaders dargestellt. Sein Inhaltsbereich beinhaltet
646   * einen Chart-Container.
647   */
648    var oIconTabChartButton = new sap.m.IconTabFilter("chartButton_1",{
649      icon: "sap-icon://line-charts",
650      text: "Chart",
651      content: [oChartContainer]
652    });
```

Listing 5.60: Sensorik: Erzeugen eines IconTabBar()-Controls (oDetailPage)

oPrinterPage

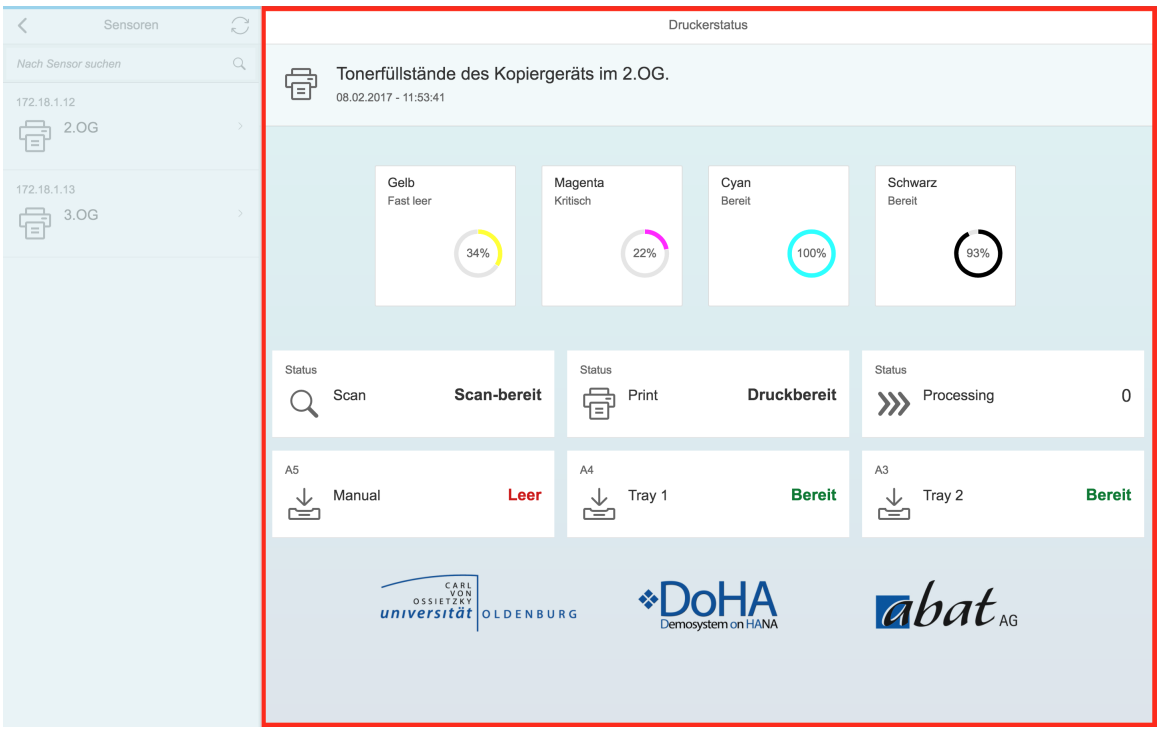


Abbildung 5.50.: Sensorik: Detaillierte Ansicht von Druckerdaten (oPrinterPage)

Abbildung 5.50 zeigt das Control, welches in der Variable *oPrinterPage* gespeichert ist. Diese Detailansicht dient der Darstellung von Druckerinformationen. Innerhalb der Räumlichkeiten der abat AG sind zur Zeit zwei Druckergeräte vorhanden. Der strukturelle Aufbau kann der Abbildung 5.36 im Abschnitt Zyklischer Datenabruf entnommen werden. Zusätzlich ist ein Header mithilfe des ObjectHeader-Controls hinzugefügt worden. Der obere Bereich stellt die Farbfüllstände der Toner (Yellow, Magenta, Cyan und Black) mit ihren Status in einer Kachelansicht, wie im Abschnitt oInitialPage, dar. Auch

hier sind die Kacheln (siehe Listing 5.61) in einem TileContainer-Control angeordnet, deren generische Kacheln von einem CustomTile-Control umgeben sind. Wie in Listing 5.62 zu erkennen ist, enthalten die Kacheln jeweils ein RadialMicroChart-Control. Neben der prozentualen Anzeige ermöglicht das Control zusätzlich eine farbliche Darstellung der Füllstände. Der Aufbau der restlichen Kacheln erfolgt analog zu dem in Listing 5.62. Als Datenlieferant wird der Service *pushCurrentSensorValue.xsjs* verwendet und nutzt das Datenmodell mit der ID *channelModel_1*.

```
1153  /**
1154   * Erzeugen eines Kachel Containers. Die generischen Kacheln mit den
       Farbfuellstaenden werden in diesem Container aggregiert. Generische
       Kacheln koennen nicht direk in einem Kachel-Container
1155   * hinterlegt werden. Darum muessen zusaetzlich noch CustomTile-Controls
        erzeugt werden, dessen Content-Bereich die generischen Kacheln
       aggregiert.
1156   */
1157
1158    var oColorContainer = new sap.m.TileContainer("colorContainer_1",{
1159      height: "280px",
1160      tiles:[
1161        new sap.m.CustomTile({
1162          content:[oYellowRadial]
1163        }),
1164        new sap.m.CustomTile({
1165          content:[oMagentaRadial]
1166        }),
1167        new sap.m.CustomTile({
1168          content:[oCyanRadial]
1169        }),
1170        new sap.m.CustomTile({
1171          content:[oBlackRadial]
1172        }),
1173      ]
1174    });
```

Listing 5.61: Sensorik: Hinzufügen von Kacheln in einen Container (oPrinterPage)

```
1049  /**
1050   * Erzeugen einer Generischen Kachel zum Anzeigen des Farbfuellstandes (
       Gelb) des Druckers. Die Werte werden bezogen aus dem Datenobjekt
1051   * mit der ID "channelModel_1".
1052   */
1053
1054    var oYellowRadial = new sap.m.GenericTile({
1055      header: '{channelModel_1>/sensors/0/channel}',
1056      subheader: '{channelModel_1>/sensors/10/value}',
1057      tileContent:[
1058        new sap.m.TileContent({
1059          content:[
1060            new sap.suite.ui.microchart.RadialMicroChart({
1061              total: 100,
1062              valueColor: "yellow",
```

```
1063                percentage: {
1064                  path: 'channelModel_1>/sensors/0/value',
1065                  formatter: function(value){
1066                    return parseFloat(value);
1067                  }
1068                }
1069              })
1070            ]
1071          })
1072        ]
1073      });
```

Listing 5.62: Sensorik: Kachel mit gelbem Farbfüllstand (oPrinterPage)

Die jeweiligen Status der Drucker und der Papierfächer werden, wie im Listing 5.63 aufgeführt ist, durch das ObjectListItem-Control dargestellt. Jedes Control wird dabei einem Grid-layout hinzugefügt. Der Header (oPrinderHeader), der Kachel-Container und das Grid-Layout mit den ObjectListItem-Controls werden einem vertikalen Layout hinzugefügt und der aggregierenden Eigenschaft des Page-Controls (oPrinterPage) übergeben. Alle Elemente werden fest und nicht dynamisch im Code angelegt, da hier das dynamische Erzeugen nicht möglich bzw. sinnvoll ist.

```
1231  /**
1232   * Erzeugen eines ObjectListItem-Controls zur Darstellung des Status (
       Tray 2) des Papierfaches des Druckers. Die Properties des Controls sind
       an das Datenobjekt mit der ID "channelModel_1! gebunden.
1233   */
1234    var oTray2 = new sap.m.ObjectListItem("trayState_2",{
1235      title: '{channelModel_1>/sensors/19/channel}',
1236      intro: '{channelModel_1>/sensors/19/unit}',
1237      number: '{channelModel_1>/sensors/19/value}',
1238      icon: "sap-icon://inbox",
1239    });
1240
1241  /**
1242   * Erzeugen eines Grid-Layouts zum Anordnen der jeweilges Status des
       Druckers.
1243   */
1244
1245    var oStateGrid = new sap.ui.layout.Grid({
1246      defaultSpan: "L4 M6 S6",
1247      content:[oScanState,oPrintState,oProcessState,oManual,oTray1,oTray2]
1248    });
1249
1250  /**
1251   * Erzeugen eines Vertikalen Layouts indem sich der Header, der Farb-
       Container, die Status und das Logo der Universitaet, der abat AG und
       DoHA befindet.
1252   */
1253
1254    var vLayout = new sap.ui.layout.VerticalLayout({
1255      width: "100%",
```

```
1256        content:[oPrinterHeader,oColorContainer,oStateGrid,oLogoImage2]
1257      });
1258
1259    /**
1260     * Erzeugen eines Page-Objekts in dem die Darstellung der Druckerdaten
          erfolgt. Im content-Bereich aggregiert es das vertikale layout "vLayout
          ".
1261     */
1262
1263      var oPrinterPage = new sap.m.Page("printerPage_1",{
1264        title: "Druckerstatus",
1265        content:[vLayout]
1266      });
```

Listing 5.63: Sensorik: Controls der Druckerinformationen (oPrinterPage)

Die Status der Druckergeräte (Scan & Print) werden vom Gerät selbst mittels eines Statuscodes angegeben. Die Funktion setPrinterStates() im Listing 5.64 löst diesen Statuscode in einen alphanumerischen Begriff auf. Für den Status *Processing* existiert seitens des Herstellers keine Statusdefinition. Die Properties-Datei befindet sich im *res*-Paket (siehe Abschnitt res).

```
1661 /**
1662  * Eine Funktion das den Status der Drucker setzt. Der Status wird extern
        aus einer Properties-Datei bezogen (WebContent/res/printer_properties/
        printer-states.xml).
1663  */
1664
1665   setPrinterStates: function(){
1666     var self = this;
1667     var printState = this.oPrintState.getNumber();
1668     var scanState = this.oScanState.getNumber();
1669
1670     $(document).ready(function(){
1671         $.ajax({
1672             type: "GET",
1673             url: "/abat/demosystem/sensor/sensorapp/ui/WebContent/res/
        printer_properties/printer_states.xml",
1674             dataType: "xml",
1675             success: function(xml) {
1676                 var pState = $(xml).find("Item[name="+printState+"]").text
        ();
1677                 var sState = $(xml).find("Item[name="+scanState+"]").text
        ();
1678
1679                 self.oPrintState.setNumber(pState);
1680                 self.oScanState.setNumber(sState);
1681             }
1682         });
1683     });
1684   },
```

Listing 5.64: Sensorik: Funktion zum auflösen der Statuscodes (oPrinterPage)

5.4. Abnahme

Der finale Abnahmetest wurde am Freitag 20.01.2017 mit dem Betreuer des Sensorik Szenarios durchgeführt. Hierbei wurden alle zuvor festgelegten Anforderungen (siehe Kapitel 5.2) kontrolliert und begutachtet. Der erstellte Prototyp wurde offiziell abgenommen und übergeben.
Die Anforderungen zur Bereitstellung der Daten wurden vollständig umgesetzt sowie abgenommen (siehe Tabelle 5.12). Zur Anforderung 1.3 "Messdaten müssen zyklisch an die Datenbank gesendet werden (Push-Verfahren)" gab es die Bemerkung, dass bei fehlender Verbindung des Raspberry Pizur Datenbank die abgefragten Werte verloren gehen. Diese werden nicht zwischengespeichert und nachgesendet.
Bei Anforderung 1.4 "Nach einem Neustart muss die Datenbereitstellung wieder automatisch aufgenommen werden" ist zu beachten, dass die Applikation zwar nach einem Neustart der Raspberry Pis startet, sich jedoch nicht automatisch neustartet, sobald die Applikation abgestützt ist.
Anforderung 1.7 (Drucker) "Die Messdaten müssen von der Datenbank abgerufen werden (Pull-Verfahren)" beschränkt sich auf die von abat gegebenen Druckermodelle und Hersteller. Um Daten von Druckern anderer Hersteller auszulesen, müssen Codeanpassungen vorgenommen werden, da die Bereitstellung der Daten durch die Drucker variiert (XML, HTML, etc.).

Tabelle 5.12.: Sensorik: Abnahme - Bereitstellung der Daten

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
1	**Bereitstellung der Daten**		
1.1	Messdaten müssen in der Datenbank aufbereitet werden.	umgesetzt	abgenommen
Raspberry Pi			
1.2	Messdaten müssen abgerufen werden können.	umgesetzt	abgenommen
1.3	Messdaten müssen zyklisch an die Datenbank gesendet werden (Push-Verfahren).	umgesetzt	abgenommen
1.4	Ausfallsicherheit: Nach einem Neustart muss die Datenbereitstellung wieder automatisch aufgenommen werden.	umgesetzt	abgenommen
1.5	Fehlermeldungen müssen an die Datenbank gesendet werden.	umgesetzt	abgenommen

Tabelle 5.12.: Sensorik: Abnahme - Bereitstellung der Daten

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
1	**Bereitstellung der Daten**		
Drucker			
1.6	Die Messdaten müssen im XML-Format bereitgestellt werden.	bereits gegeben	
1.7	Die Messdaten müssen von der Datenbank abgerufen werden (Pull-Verfahren).	umgesetzt	abgenommen

Alle Anforderungen des Datenmodells wurden ebenfalls komplett umgesetzt und abgenommen (siehe Tabelle 5.13). Die Anforderung 2.2.3 “Es muss eine separate Loggig-Tabelle angelegt werden“ bezieht sich nur auf Fehlermeldungen der Raspberry Pis. Warnungen oder Fehler der Datenbank oder der Drucker werden in dieser Tabelle nicht gespeichert.

Tabelle 5.13.: Sensorik: Abnahme - Datenmodell

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
2	**Datenmodell**		
2.1	Ein geeignetes Datenbankschema muss erstellt werden.	umgesetzt	abgenommen
2.2	Geeignete Datenbanktabellen müssen erzeugt werden.	umgesetzt	abgenommen
2.2.1	Für alle eingehenden Messdaten, unabhängig von der Art des Sensors, ist eine Sammeltabelle anzulegen.	umgesetzt	abgenommen
2.2.2	Die Sammeltabelle soll 20 Parameter speichern können.	umgesetzt	abgenommen
2.2.3	Es muss eine separate Loggig-Tabelle angelegt werden.	umgesetzt	abgenommen
2.2.4	Für die Erstellung der Datenbank sind Datenbankstrukturdateien zu verwenden	umgesetzt	abgenommen

Es wurden alle Anforderungen zum Daten empfangen umgesetzt und vollständig abgenommen (siehe Tabelle 5.14). Für die Anforderung 3.3 gilt die gleiche Bemerkung, wie bereits für Anforderung 2.2.3 oben erwähnt.

Tabelle 5.14.: Sensorik: Abnahme - Daten empfangen

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
3	**Daten empfangen**		
3.1	Die Empfangenen Daten müssen in einer HANA-Datenbank gespeichert werden.	umgesetzt	abgenommen
Raspberry Pi			
3.2	Die Datenbank muss die Messdaten, die mittels des Push-Verfahrens zur Verfügung gestellt werden sollen, empfangen können.	umgesetzt	abgenommen
3.3	Die Datenbank muss Fehlermeldungen empfangen können.	umgesetzt	abgenommen
Drucker			
3.4	Die Datenbank muss die Messdaten, die mittels des Pull-Verfahrens zu beziehen sind, empfangen können.	umgesetzt	abgenommen

Die Anforderungen zum Aufbereiten der Daten wurden ebenfalls vollständig umgesetzt und abgenommen (siehe Tabelle 5.15).
Für Anforderung 4.1 “Die Datenbank muss die gespeicherten Daten aufbereiten“ ist anzumerken, dass bei nicht gesetzten Min- und Max-Werten eines Channels Warnungen nicht aufbereitet werden. Ebenfalls wurde auf eine Anomalieerkunngen auf Datenbankebene verzichtet. Es werden alle empfangenen Werte gespeichert und ausgegeben.

Tabelle 5.15.: Sensorik: Abnahme - Daten aufbereiten

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
4	**Daten aufbereiten**		
4.1	Die Datenbank muss die gespeicherten Daten aufbereiten.	umgesetzt	abgenommen
4.2	Die Aufbereitung der Daten muss auf Datenbank-Ebene geschehen.	umgesetzt	abgenommen

Die Anforderungen der grafischen Oberfläche wurden zum größten Teil umgesetzt. Es wurden die Anforderungen 5.8 und 5.12 “Kacheln im SAP S/4HANA“ aus technischen Gründen und in Absprache mit abat nicht umgesetzt. Alle umgesetzten Anforderungen wurden abgenommen.
Zur Anforderungen 5.6 “Historische Sensorwerte müssen grafisch aufbereitet (Chart, Tabellen o.ä.) dargestellt werden“ ist zu bemerkten, dass es keine Möglichkeit gibt, alle Sensoren in einem gemeinsamen Chart abzubilden.
Anforderung 5.10 “Papierfüllstand muss angezeigt werden“ wurde auf den textuellen Status beschränkt. Eine genaue Anzahl des Papierfüllstands wird nicht von den verwendeten Druckern bereitgestellt und kann daher auch nicht berechnet werden.

Tabelle 5.16.: Sensorik: Abnahme - Grafische Oberfläche

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
5	**Grafische Oberfläche**		
5.1	Muss modular gestaltet sein.	umgesetzt	abgenommen
5.2	Die Daten müssen zyklisch aktualisiert werden.	umgesetzt	abgenommen
5.3	Das SAPUI5-Framework muss verwendet werden.	umgesetzt	abgenommen
5.3.1	Es ist das Model-View-Controller Architekturmuster zu verwenden.	umgesetzt	abgenommen
5.3.2	Als Datenübertragungsformat ist die JavaScript Object Notation zu verwenden.	umgesetzt	abgenommen

Tabelle 5.16.: Sensorik: Abnahme - Grafische Oberfläche

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
5	**Grafische Oberfläche**		
5.3.3	"Die allgemeine Darstellung muss in der Master-/Detail-Struktur des SplitApp-Controls (SAPUI5) erfolgen.	umgesetzt	abgenommen
5.4	Messwerte müssen beim Auswählen eines Sensors angezeigt werden.	umgesetzt	abgenommen
Raspberry Pi			
5.5	Messwerte müssen angezeigt werden.	umgesetzt	abgenommen
5.6	Historische Sensorwerte müssen grafisch aufbereitet (Chart, Tabellen o.ä.) dargestellt werden.	umgesetzt	abgenommen
5.7	Es muss eine Tabelle zum Anzeigen von Ausnahmewerten erstellt werden.	umgesetzt	abgenommen
5.8	Pro Raspberry Pi soll eine Kachel erstellt werden können.	nicht umgesetzt	-
Drucker			
5.9	Äktuelle Farbfüllstände müssen angezeigt werden (Yellow, Magenta, Cyan, Schwarz).	umgesetzt	abgenommen
5.10	Papierfüllstand muss angezeigt werden.	umgesetzt	abgenommen
5.11	Die Status des Druckers muss angezeigt werden.	umgesetzt	abgenommen
5.12	Pro Drucker soll eine Kachel erstellt werden können.	nicht umgesetzt	-

Abschließend wurden noch zwei weitere (Rahmen-) Anforderungen betrachtet und vollständig abgenommen.

Tabelle 5.17.: Sensorik: Abnahme - weitere Anforderungen

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
6	**Daten aufbereiten**		
6.1	Es ist eine native HANA-Anwendung zu implementieren.	umgesetzt	abgenommen
6.2	Für die Anwendung ist eine Kachel auf der nativen HANA Administrationsoberfläche zu erstellen.	umgesetzt	abgenommen

5.5. Fazit

Zusammenfassend gelang die Integration verschiedener Sensoren, inklusive ihrer Sensorwerte, in eine HANA-Datenbank erfolgreich. Es wurde aufgezeigt, dass verschieden Integrationsverfahren, wie zum Beispiel Push- und Pull-Verfahren, zum Abrufen von verschiedenen Sensorwerten (Raspberry Pi und Drucker) verwendet werden können.
Die grafische Darstellung dieser Massenwerte sowie deren Analyse und Auswertung wurde ebenfalls erfolgreich umgesetzt. Hierbei wurde gezeigt, dass große Mengen an Werten mit guter Performance durch eine HANA-Datenbank bearbeitet und dargestellt werden können.
Zudem wurden mögliche Flaschenhälse dieses Prototypen identifiziert. Das Abrufintervall der Sensorwerte hängt maßgeblich von der möglichen Abrufgeschwindigkeit der verwendeten Sensoren ab. Wurden einzelne Sensoren zu schnell nacheinander abgefragt, so entstanden Fehlerwerte. Ein weiteres Problem ist die gesendete Datenmenge der Services an die GUI. Durch synchrone Aufrufe komplexer Services durch die GUI verzögerte sich teilweise die Darstellung.

5.6. Ausblick

Der erstellte Prototyp könnte in Zukunft durch weitere Sensoren und Funktionen erweitert werden. Prinzipiell ist eine Erweiterung um jeden beliebigen Sensor möglich, da die Datenbank sowie die GUI modular gestaltet und implementiert wurden. Für eine Erweiterung um neue Sensoren ist lediglich die Implementierung der Abruf-Services nötig.

Mögliche Erweiterungsszenarien im Unternehmenskontext wären zum Beispiel das Abrufen und Speichern von Maschinensensoren (Temperatur, Vibration, Wärme) um anschließend den erstellten Prototypen um eine Predictive-Maintenance Komponente zu erweitern. Allgemein bietet der Prototyp die Möglichkeit eine breite Datenbasis jeglicher Sensoren zu liefern, um diese in verschiedenen Bereichen zu analysieren.

6. Szenario - Projektcontrolling

6.1. Beschreibung

In diesem Abschnitt wird das Szenario „Projektcontrolling“ der Masterprojektgruppe „Demosystem on HANA“ der Universität Oldenburg erläutert und der entwickelte Prototyp für das Projektcontrolling beschrieben. Kernaufgabe des Szenarios war es, Arbeitszeitinformationen von einem Quellsystem in SAP HANA zu importieren und anschließend das Auswerten von Projekten über diverse UI5-Oberflächen zu ermöglichen. Ebenfalls sollte es möglich sein, Projekte zu planen und die Auswertung mit den Planwerten zu vergleichen.
Zu Beginn werden alle Anforderungen des Szenarios in tabellarischen Auflistungen dargestellt. Hierbei wurden die Anforderungen in logische und funktionell getrennte Kategorien gegliedert.
Anschließend wird die Architektur beschrieben und für ein besseres Verständnis grafisch dargestellt. Abschließend folgt die Dokumentation aller in den Tabellen beschriebenen Anforderungen. Hierbei wird zu jeder Anforderung die Umsetzung textuell beschrieben. Der dabei beschriebene Quelltext befindet sich in Auszügen direkt unter der jeweiligen Umsetzungsbeschreibung. Sollten Anforderungen spezielle Konfigurationsmöglichkeiten bieten, so werden diese zum Schluss jeder Anforderungsdokumentation erläutert. Der komplette Quellcode der Anwendungen befindet sich auf der beiliegenden CD. Dieses Dokument richtet sich zum einen an die Betreuer der Projektgruppe DoHA, von der Universität Oldenburg und zum anderen an die Betreuer der abat AG.

6.2. Anforderungen

Die gestellten Anforderungen sind in Tabellen aufgelistet und beschrieben. Dabei werden die Anforderungen in die vier folgenden Kategorien aufgeteilt: „Darstellung und Auswertung“, „Einlesen der Daten“, „Berechnungsgrundlagen und Algorithmen“ und „Extra Schnittstelle/ Funktionsbaustein“.

6.2.1. Darstellung und Auswertung der Daten

Für die Auswertung der Projektdaten war es erforderlich, eine interaktive Oberfläche mittels UI5 zu implementieren, welche eine detaillierte Auswertung der Projekte ermöglicht. Eine Auswertung mittels einer Excel-Vorlage war ebenfalls gewünscht, welche eine ähnliche Funktionalität wie die UI bereitstellt (vgl. Tabelle 6.1 auf Seite 193).

Tabelle 6.1.: Projektcontrolling: Anforderungen - Darstellung und Auswertung

Nr.	**Anforderung:**	**Beschreibung:**
1	**Darstellung und Auswertung**	
1.1	Die Fiori-Oberfläche soll eine grafische Darstellung des Projektstatus ermöglichen	Die grafische Oberfläche bietet eine anschauliche Standardansicht der Projektkennzahlen und Informationen.
1.2a	In der grafischen Oberfläche sollen abgerechnete Arbeitspakete gekennzeichnet werden	Abgerechnete Arbeitspakete sollen gekennzeichnet werden. Die Kennzeichnung erfolgt über die Einteilung in abgerechnete und nicht abgerechnete Datensätze. Die Kennzeichnung muss manuell zurücksetzbar sein. Das Arbeitspaket wird in der Datenbank mit einem X in der Spalte abgerechnet markiert. Ist das Paket noch nicht abgerechnet, ist das Feld leer. Die Werte werden über eine Schnittstelle bereitgestellt. Die Schnittstelle muss von abat erstellt und bereitgestellt werden.
1.2b	In der Standard-Excel-Auswertung sollen abgerechnete Arbeitspakete gekennzeichnet werden	Das Arbeitspaket wird in der Übersicht markiert. Die Kennzeichnung erfolgt über die Einteilung in *abgerechnete* und *noch nicht abgerechnete* Datensätze. Ist das Paket noch nicht abgerechnet, ist das Feld leer.
1.3a	In der grafischen Fiori-Oberfläche sollen Schwellenwerte nach Status gekennzeichnet werden	Die Übersicht der Schwellenwerte erfolgt in der Fiori Oberfläche mittels einer Ampeldarstellung. Bei Überschreiten des Budgetverbrauchs der Rot-Schwelle werden die Werte rot dargestellt, gelb unterhalb der Rot-Schwelle und über der Gelb-Schwelle. Die Darstellung in grün erfolgt wenn der Budgetverbrauch unter der Gelb-Schwelle liegt.
1.3b	In der Standard-Excel-Auswertung sollen Schwellenwerte nach Status gekennzeichnet werden	Das zu kennzeichnende Feld wird in der Excel-Auswertung grün/gelb/rot gefärbt. Der Status wird identisch wie in Anforderung 1.3a ermittelt.

Tabelle 6.1.: Projektcontrolling: Anforderungen - Darstellung und Auswertung

Nr.	Anforderung:	Beschreibung:
1	**Darstellung und Auswertung**	
1.4	Das System muss eine Exportmöglichkeit nach Excel bieten	Ergebnisse sollen für die Weiterverarbeitung in Excel bereit gestellt werden. Ein Standard-Excel-Sheet kann so für Kunden manuell angepasst werden (z.B. Pivot-Tabelle).

6.2.2. Einlesen der Daten

Die Anforderungen zum Einlesen der Daten umfassen die Bereitstellung eines Datenmodells und Eingabeoberflächen zum Speichern von Plan- und Stammdaten. Die Eingabe soll dabei ähnlich wie in Excel erfolgen (Massendateneingabe). Zusätzlich ist ein Monitoring erforderlich, welches prüft, ob die Daten des Funktionsbausteins (vgl. Tabelle 6.4 auf Seite 199) korrekt überführt wurden. Die Prüfung auf Vollständigkeit der Daten ist als Werkzeug für Projektleiter zu verstehen, welche damit nachvollziehen können, welcher Mitarbeiter wann seinen letzten Stundenzettel abgegeben hat (vgl. Tabelle 6.2 auf Seite 197).

Tabelle 6.2.: Projektcontrolling: Anforderungen - Einlesen der Daten

Nr.	Anforderung:	Beschreibung:
2	**Einlesen der Daten**	
2.1	Das System muss ein Datenmodell bereitstellen	Als Grundlage der verschiedenen Applikationen dient ein Datenmodell, welches zum Speichern der verschiedenen Daten genutzt wird. Hierzu müssen auf der HANA-Datenbank verschiedene Tabellen angelegt und verknüpft werden.

Tabelle 6.2.: Projektcontrolling: Anforderungen - Einlesen der Daten

Nr.	**Anforderung:**	**Beschreibung:**
2	**Einlesen der Daten**	
2.2 / 2.3	Fiori-Oberflächen zur Massendatenerfassung müssen bereitgestellt werden	Zur Erfassung der benötigten Daten in der HANA-DB ist es erforderlich, zwei Fiori-Applikation zu erstellen. Die Erfassung der Daten soll dabei in Tabellenform, ähnlich wie in Excel, erfolgen. Die erste Applikation wird genutzt um Stammdaten zu erfassen, die zweite Applikation wird genutzt um Planwerte zu einem Projekt zu erfassen. Die einzelnen Applikationen werden in den folgenden Anforderungen erläutert. Falsche Eingaben sollen im Vorfeld vermieden werden. Dies wird gewährleistet, indem die eingegebenen Werte auf ihre Konsistenz geprüft werden.
2.2.1	Mobile Oberfläche (App)	Die Oberfläche soll so gestaltet werden, dass sie auch mobil verwendbar ist. Dabei soll es funktional keine Einschränkungen geben. Die Übersichtlichkeit soll in der Form gewährleistet werden, dass es verschiedene Reiter geben soll, um Daten einzugeben und anzuzeigen. Dadurch soll die Übersichtlichkeit in der mobilen App nicht eingeschränkt werden und gleichzeitig alle Funktionalitäten vorhanden sein.
2.3.1	Eine Oberfläche muss Stammdaten erfassen	Die Fiori-Oberfläche zur Stammdatenerfassung muss diverse Felder zur Zuordnung bereitstellen.

Tabelle 6.2.: Projektcontrolling: Anforderungen - Einlesen der Daten

Nr.	**Anforderung:**	**Beschreibung:**
2	**Einlesen der Daten**	
2.3.2	Eine Oberfläche muss Planwerte erfassen.	Um die Auswertung einzelner Projekte bzw. Arbeitspakete zu ermöglichen, ist es erforderlich Planwerte zu den Projekten hinzuzufügen. Die Oberfläche muss es ermöglichen, Projekten ein Planbudget oder Planstunden zuzuweisen, die als Berechnungsgrundlage des Projektstatus dienen. Zusätzlich kann angegeben werden, ob die Werte nur geplant oder fix sind.
2.4	E-Mail-Benachrichtigung und Monitoring Tool für Fehlerfälle	In Fehlerfällen soll vom System eine E-Mail-Benachrichtigung erfolgen. Das Monitoring-Tool soll überwachen, ob die Daten korrekt ins HANA-System übernommen wurden. Ist ein Fehler bei der Übernahme aufgetreten, soll eine E-Mail versendet werden.
2.5	Die Vollständigkeit der Daten (Timesheets) muss geprüft werden	Im System soll angezeigt werden, zu welchen Mitarbeitern Time Sheets fehlen. Es soll mit Kürzel angezeigt werden, welcher Mitarbeiter ab einer bestimmten Schwelle noch keine Abgabe getätigt hat. Fehlen Time Sheets, sollen die entsprechenden Mitarbeiter angezeigt werden können. Hierzu wird eine View erstellt, die zu jedem Mitarbeiter das letzte Abgabedatum enthält. Eine fehlende Abgabe kann so leicht ermittelt werden.

Tabelle 6.2.: Projektcontrolling: Anforderungen - Einlesen der Daten

Nr.	**Anforderung:**	**Beschreibung:**
2	**Einlesen der Daten**	
2.6	Ein Rabatt/Risikoaufschlag muss mit einbezogen werden können	Ist bei einem Projekt ein Rabatt bzw. Risikoaufschlag gegeben, muss dieser in der Oberfläche der Planwerterfassung mit angegeben werden können. Hierzu werden zu der Oberfläche zwei Textfelder hinzugefügt, mit denen angegeben werden kann, ob ein Rabatt oder Zuschlag gegeben ist. Der Wert kann in diesen Feldern als Prozentwert eingegeben werden. Eine 0 oder ein leeres Feld kennzeichnet keinen Zuschlag/Rabatt. Es wird anschließend eine Berechnung durchgeführt um das korrekte Budget zu ermitteln, welches dann in die Datenbank geschrieben wird.
2.7	Der Sperrstatus von PSP-Elementen muss abgerufen werden	Nachträglich oder verspätet abgegebene Stundenzettel sollen nicht dafür sorgen, dass sich das Projektbudget sprunghaft erhöht. Neue Stundenzettel sollen bei gesperrten PSP-Elementen nicht in die Auswertung aufgenommen werden und können nur noch manuell eingepflegt werden. Die Stundenzettel sollen dann nur von Benutzern mit bestimmten Berechtigungen eingepflegt werden können. Die Information, ob ein PSP-Element gesperrt ist, kann über eine Schnittstelle abgerufen werden, sodass keine manuelle Sperrung über eine HANA-Applikation erforderlich ist (die Sperrung erfolgt im SAP ERP 6.0-System).

Tabelle 6.2.: Projektcontrolling: Anforderungen - Einlesen der Daten

Nr.	Anforderung:	Beschreibung:
2	**Einlesen der Daten**	
2.8	Konsistenz der Daten prüfen (keine ungültigen Zuordnungen speicherbar)	Es dürfen keine ungültigen Zuordnungen in der Datenbank gespeichert werden. Es darf z.B. kein Mitarbeiter eingeplant werden der nicht existiert, Planwerte müssen vorhandenen PSP-Elementen zugeordnet werden, es können nur Zuordnungen zur bereits existierenden Projekten vorgenommen werden (etc.).

6.2.3. Berechnungsgrundlagen und Algorithmen

Um die korrekte Berechnung der Projektstatus zu gewährleisten, gibt es Anforderungen an die Berechnungsgrundlagen und Algorithmen. Hierzu gehört, dass Algorithmen die korrekte Budget- und Stundensituation errechnen können. Ebenfalls kann hiermit der Projektstatus berechnet werden. Eine Grundlage hierfür bietet der korrekte Abruf der Stundensätze der jeweiligen Mitarbeiter eines Projekts. Zusätzlich soll neben der Auswertung des aktuellen Projektstatus auch eine Auswertung der Vergangenheit möglich sein, um Änderungen der Budget- bzw. Stundensituation nachzuvollziehen. Tabelle 6.3 auf der nächsten Seite zeigt die Anforderungen an die Berechnungsgrundlagen und Algorithmen.

Tabelle 6.3.: Projektcontrolling: Anforderungen - Berechnungsgrundlagen und Algorithmen

Nr.	Anforderung:	Beschreibung:
3	**Berechnungsgrundlagen und Algorithmen**	
3.1	Ein Algorithmus zur Berechnung der aktuellen Budget/Stunden-Situation muss erstellt werden	Um die aktuelle Budget/Stunden-Situation zu ermitteln, ist es erforderlich zu einem gewähltem Element (z.B. PSP-Element oder ein gesamtes Projekt) alle geleisteten Stunden zu ermitteln und diese mit den entsprechenden Stundensätzen zu bewerten. Das Ergebnis spiegelt die aktuelle Kosten- bzw. Stundensituation des Projekts wieder und wird für weitere Auswertungen benötigt.
3.2	Ein Algorithmus muss die Berechnung des Projektstatus gewährleisten	Zur Bewertung der Situation eines Projekts wird die Budget/-Stunden-Situation als Grundlage genutzt. Der Ist-Wert des Projekts wird mit vorher festgelegten Schwellwerten verglichen, woraufhin ein Projektstatus ausgegeben wird.
3.3	Ein Algorithmus muss den Abruf des Stundensatzes ermöglichen	Zur korrekten Verrechnung der Stunden ist es erforderlich, die entsprechenden Stundensätze zu ermitteln. Hierfür wird eine Abfrage erstellt, welche den zugehörigen Stundensatz in Abhängigkeit zu einem PSP-Element und einem Skill-Level ermittelt. Der Abruf kann dabei über Abfragen der benötigten Tabellen realisiert werden.
3.4	Historienvergleich über verschiedene Zeitscheiben muss möglich sein	Die Historie von Projekten und PSP-Elementen soll abrufbar sein. Hierzu werden verschiedene Datenstände in das System geladen (Zeitscheiben), welche einen Vergleich von unterschiedlichen Terminen ermöglichen.

6.2.4. Extra Schnittstelle / Funktionsbaustein

Um die Auswertung der Projekte zu ermöglichen, ist es erforderlich regelmäßig Projektdaten aus einem Fremdsystem auf die HANA-Datanbank zu übertragen. Tabelle 6.4 zeigt die Anforderungen im Bezug auf die Schnittstelle.

Tabelle 6.4.: Projektcontrolling: Anforderungen - Extra Schnittstelle / Funktionsbaustein

Nr.	**Anforderung:**	**Beschreibung:**
4	**Extra Schnittstelle / Funktionsbaustein**	
4.1	Über eine Schnittstelle soll eine PSP-Elementbeschreibung bereitgestellt werden	Über die Eingabe der PSP-Nummer soll die Beschreibung eines PSP-Elements abrufbar sein (beschreibender Text). Dies wird benötigt, um die Beschreibung eines PSP-Elements abzubilden und weitere beschreibende Informationen (Rechnungsnummer, Sperrstatus) abzurufen. Der Funktionsbautein wird von abat erstellt und bereitgestellt.
4.2	Die HANA-DB muss täglich mit Daten versorgt werden	Es ist ein Job zu erstellen, welcher täglich Projektdaten aus einem SAP ERP 6.0-System in die HANA-DB lädt. Der Funktionsbaustein hierfür ist bereits vorhanden.

6.3. Umsetzung

Nachdem im vorherigen Kapitel die einzelnen Anforderungspakete erläutert wurden, wird in diesem Kapitel auf die Umsetzung eingegangen. Hierzu werden die einzelnen Elemente der Applikation schichtweise erläutert. Begonnen wird mit dem Datenmodell, anschließend folgen Auszüge der Views und Prozeduren. Danach wird auf die Services und Models eingegangen und abschließend wird die GUI erläutert. Zum besseren Verständnis werden Code-Beispiele gegeben und beschrieben. Der komplette Quellcode der Anwendung ist auf dem beiliegenden Datenträger enthalten.

6.3.1. Architektur

Abbildung 6.1 zeigt eine vereinfachte Architektur der Anwendung des Szenarios „Projektcontrolling“. Die Architektur besteht aus verschiedenen Paketen, welche verschiedenen Schichten zugeordnet werden können. Das „data“-Paket stellt die unterste Schicht dar, auf welcher die Daten abgelegt werden und das „UI“-Paket ist die oberste Schicht, welche dem Nutzer die Interaktion mit den Daten mittels einer grafischen Oberfläche ermöglicht. Der Export als Comma-seperated values (CSV)-Datei macht die Daten für die Excel-Vorlage nutzbar.

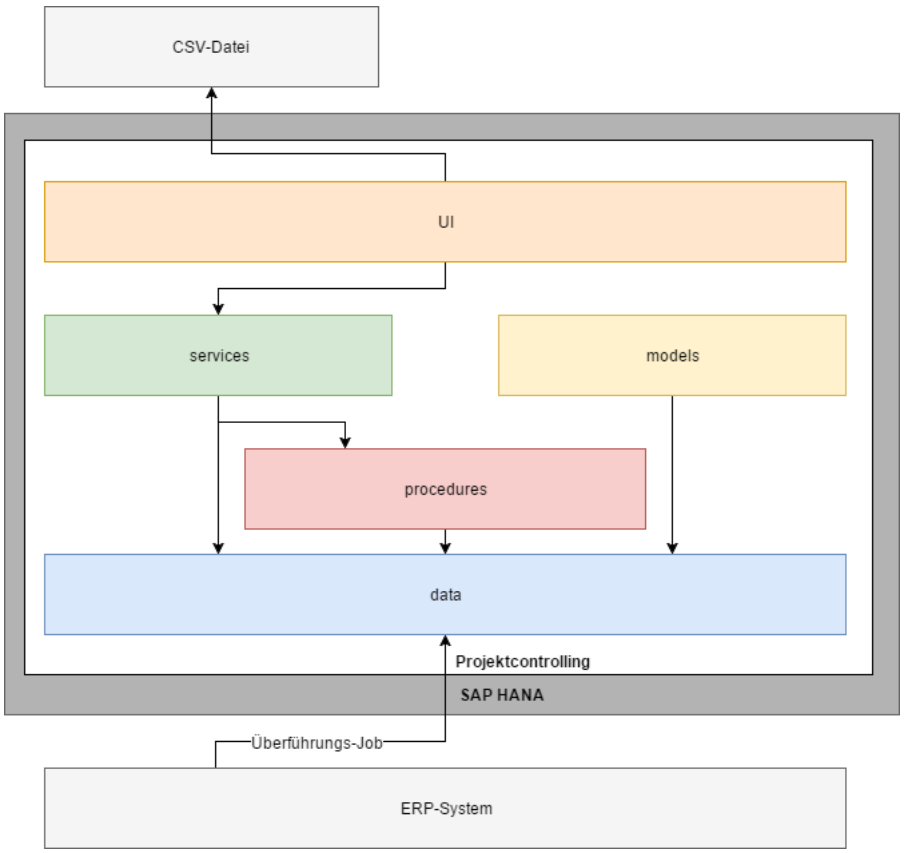


Abbildung 6.1.: Projektcontrolling: vereinfachte Darstellung der Gesamtarchitektur

Wie in Abbildung 6.2 dargestellt, wurden diese Pakete, bis auf die externen Bereiche (CSV und ERP-System), ebenfalls in der Projektstruktur berücksichtigt und bilden gleichzeitig den logischen Aufbau der gesamten Applikation.

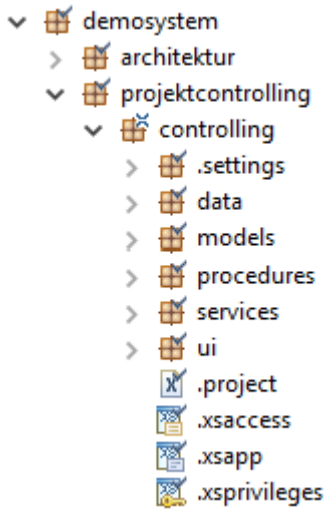


Abbildung 6.2.: Projektcontrolling: Darstellung der Projektstruktur und Architektur

Die einzelnen Pakete der Architektur und Projektstruktur werden im Folgenden kurz beschrieben.

projektcontrolling

Das Paket „projektcontrolling“ ist die eigentliche Anwendung, welche alle Bestandteile beinhaltet. In diesem Paket sind alle anderen Pakete, wie UI, data, procedures etc. enthalten, welche auf dem HANA-Server vorhanden sind.

data

Das „data“-Paket enthält alle Bestandteile des Datenbankschemas der Anwendung. Hierbei handelt es sich um die Tabellen, welche zur Datenhaltung verwendet werden sowie Schemata und Nutzerrollen.

procedures

Im „procedures“-Paket sind alle Datenbankprozeduren enthalten, die für die Anwendung erforderlich sind. Hierbei handelt es sich um Get-, Set-, Update-, und Delete-Prozeduren, welche benötigt werden, um die Daten in korrekter Form gemäß der Anforderungen an die GUI zu übergeben oder, um die konsistente Bearbeitung bzw. Löschung der Daten zu gewährleisten. Die Prozeduren werden dabei von den Services aufgerufen. Eine Auflistung der Prozeduren inklusive Input- und Outputparametern ist in Abschnitt 6.3.2 auf Seite 211 zu finden.

models

Das Paket „models“ enthält alle erstellten Datenbank-Views. Die Views stellen zum einen sicher, dass die Daten für die Get-Prozeduren im korrekten Format abgerufen und an die Services weitergegeben werden können. Zum anderen werden die Views genutzt, um Informationen aus mehreren Tabellen zu konsolidieren und somit einen zentralen Anlaufpunkt für Abfragen zu erhalten ohne, dass komplexe Joins erforderlich werden (Siehe hierzu auch Abbildung 6.4 auf Seite 204).

services

Die Schnittstelle zwischen UI und der Datenbank bilden die Services. Diese sind für das Abrufen und Speichern der Daten zuständig. Gespeichert werden die Daten, indem sie als Parameter über die Services an die Prozeduren übergeben werden. Komplexe Abfragen erfolgen ebenfalls über Prozeduren, wohingegen Abfragen mit geringerer Komplexität durch einen direkten Zugriff der Services auf die Datenbank erfolgen.

UI

Zur Kommunikation mit dem Anwender wurde die Benutzerschnittstelle (UI) mit der Model-View-Controller-Architektur geschrieben. Dabei wurde, um eine Vereinheitlichung innerhalb der Projektgruppe zu schaffen, in den Controller- und View-Klassen JavaScript verwendet. Alle dazugehörigen Komponenten weisen den gleichen Aufbau auf, wie bei dem Szenario „Sensorik“ (vgl. UI in Abschnitt 5.3.1 auf Seite 96). Auf diese wird im Folgenden daher nicht im Detail eingegangen.

ERP-System

Das ERP-System ist außerhalb des HANA-Systems angesiedelt und wird regelmäßig angesprochen, um die aktuellen Arbeitszeitdaten abzurufen und auf der HANA-Datenbank abzuspeichern. Der Aufruf des ERP-Systems erfolgt dabei mittels eines ABAP-Programms, welches in Abschnitt 6.3.3 auf Seite 232 erläutert wird.

CSV-Datei

Die CSV-Datei bildet die Grundlage für die Daten der Excel-Vorlage und ermöglicht, die Daten eines Projekts aus dem System zu exportieren, um diese extern weiter zu nutzen (z.B. einem Kunden zur Verfügung zu stellen). Die CSV-Datei muss dabei in die Vorlage importiert werden, damit diese korrekt genutzt werden kann. Dieser Vorgang ist in Abschnitt 6.3.6 auf Seite 305 beschrieben.

6.3.2. HANA-Applikation

Die einzelnen Pakete der Anwendung werden in diesem Kapitel genauer erläutert und dargestellt. Hierzu wird der Code beispielhaft an einzelnen Codeteilen erläutert. Der komplette Quellcode der Anwendung ist auf dem beigelegten Datenträger zu finden.

data

In diesem Abschnitt wird die Implementierung des „data“-Pakets erläutert. Abbildung 6.3 zeigt alle Elemente, die im Paket enthalten sind. Hierbei handelt es sich um alle Datenbanktabellen, die mit Hilfe des HANA-Studios erstellt worden sind (die Tabellen, welche mittels der SAP GUI erstellt worden sind, sind nicht enthalten und werden im Abschnitt 6.3.3 auf Seite 232 mit der ABAP-Applikation erläutert). Die Datenbanktabellen, Rollen und Schemata wurden dabei über Objekte als „.hdbtable“-, „.hdbrole“- und „.hdbschema“-Dateien erstellt. Auf das Anlegen von Tabellen o.ä. über SQL-Befehle wurde verzichtet.

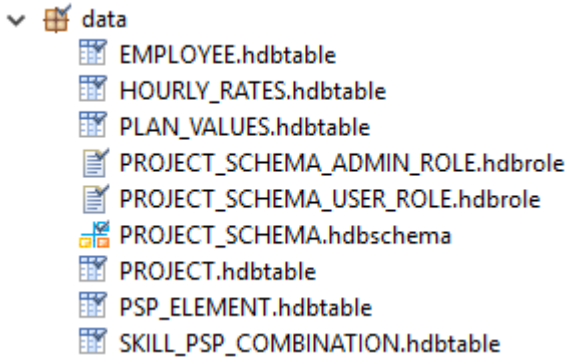


Abbildung 6.3.: Projektcontrolling: Inhalt des data-Packages

Bei der Erstellung des Datenbankmodells war es eine Herausforderung, die komplexe Projektstruktur mit allen Bestandteilen konsistent abzubilden. Das implementierte Datenmodell ist dabei in Abbildung 6.4 auf der nächsten Seite dargestellt. Die Struktur ist so aufgebaut, dass Projekten Teilprojekte (PSP-Elemente) zugeordnet werden. Diesen werden wiederum Stundensätze für die verschiedenen Skilllevel (wie Entwickler, Berater, etc.) zugewiesen. Ebenfalls werden den PSP-Elementen Mitarbeiter-Skill-Kombinationen zugewiesen. Mittels dieser Zuordnungen kann die Arbeitszeit der Mitarbeiter für die PSP-Elemente eindeutig monetär bewertet werden, was eine Kernberechnung der Anwendung darstellt.
Die Tabellen „ZDOHA_PC“ und „ZDOHA_MA“ sind nicht im Paket enthalten, da diese Tabellen mittels der SAP GUI erstellt worden sind und somit auch dem „SAPHANADB“-Schema zugeordnet sind. Die Tabellen werden genutzt, um die Daten des ERP-Systems abzuspeichern.
Um die Daten im Projektschema verfügbar zu machen, sind die Views „WORK_DONE“ und „EMPLOYEE“ erstellt worden. Die Daten der verschiedenen Tabellen werden in der

View „WORK_DONE“ konsolidiert. Sie stellt somit einen zentralen Punkt zum Abruf der Daten dar (Siehe hierzu auch Listing 6.3 auf Seite 208).
Die View „EMPLOYEE“ bringt die Mitarbeiterdaten des ERP-Systems in die Form der anderen Datenbanktabellen (Anpassen der Spaltenbezeichnungen).

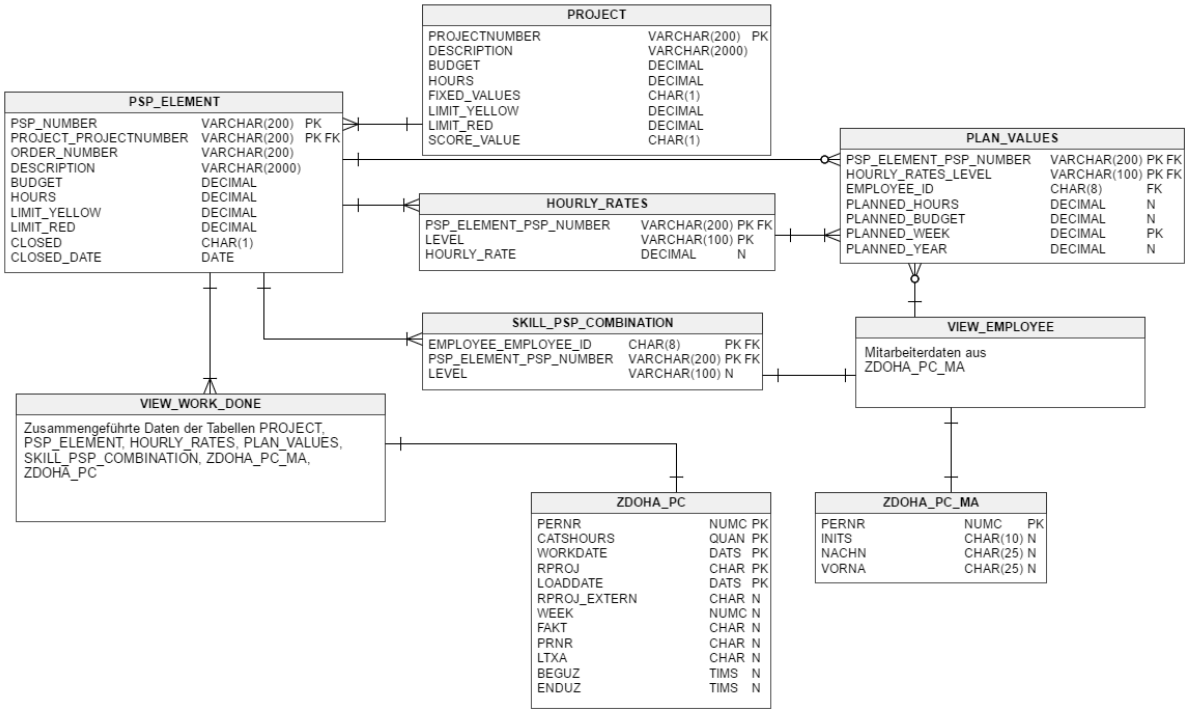


Abbildung 6.4.: Projektcontrolling: Datenbankmodell

Das Datenbankmodell besteht aus fünf direkt implementierten Tabellen, zwei Tabellen, welche über die SAP GUI erstellt wurden und zwei Views, welche die Daten aus den SAP GUI-Tabellen zur Verfügung stellen.
Die Tabellen des Datenbankschemas sind prinzipiell ähnlich aufgebaut. Lediglich die Spalten und ggf. einige der Datentypen weichen voneinander ab. Als beispielhafte Implementierung einer Datenbanktabelle ist die „PROJECT“-Tabelle in Listing 6.1 zu finden, von der sich die Implementierung anderer Tabellen ableiten lässt.

```
1  table.schemaName = "PROJECT_SCHEMA";
2  table.tableType = COLUMNSTORE;
3
4  table.columns =
5  [
6  {name = "PROJECTNUMBER";   sqlType = VARCHAR; length = 200;
7  nullable = false;},
8  {name = "BUDGET";   sqlType = DECIMAL;},
9  {name = "HOURS";   sqlType = INTEGER;},
10 {name = "FIXED_VALUES";   sqlType = CHAR; length = 1;},
11 {name = "LIMIT_YELLOW";   sqlType = DECIMAL;},
12 {name = "LIMIT_RED";   sqlType = DECIMAL;},
```

```
13 {name = "DESCRIPTION";   sqlType = VARCHAR; length = 2000;},
14 {name = "SCORE_VALUE";   sqlType = CHAR; length = 1;}
15 ];
16
17 table.primaryKey.pkcolumns = ["PROJECTNUMBER"];;
```

Listing 6.1: Projektcontrolling: Implementierung PROJECT.hdbtable

Es ist erforderlich die Tabellen dem korrekten Schema zuzuordnen (Zeile 1) und den Tabellentyp zu definieren (Zeile 2, „COLUMNSTORE“). Anschließend werden mit „table.columns“ in Zeile 4 die Spalten der Tabelle definiert. Bei der Spaltendefinition ist zu beachten, dass die Länge und der Datentyp korrekt sind, da sonst unvorhergesehene Fehler auftreten können. Für Texte wurde i.d.R. der Datentyp „VARCHAR“ verwendet. Ausnahmen sind Felder, die die Datensätze beispielsweise als gesperrt, fix oder abgerechnet beschreiben. Hierfür wurde der Datentyp „CHAR“ mit der Länge 1 gewählt, da in diese Felder nur ein markierendes Zeichen eingefügt werden darf (z.B. „X“). Für numerische Werte wurde „DECIMAL“ genutzt.
Nach der Spaltendefinition wird mittels „table.primaryKey.pkcolumns“ in Zeile 17 der Primärschlüssel der Tabelle definiert. Es kann sich hierbei auch um mehrere Spalten handeln, um einen zusammengesetzten Primärschlüssel zu bilden.

Eine Ausnahme vom oben beschriebenen Vorgehen zum Anlegen von Tabellen bilden die Tabellen „ZDOHA_PC“ und „ZDOHA_MA“, welche mittels der SAP GUI erstellt worden sind und von der erstellten ABAP-Anwendung befüllt werden. Weitere Informationen zur ABAP-Anwendung sind im Abschnitt 6.3.3 auf Seite 232 zu finden. Die beiden Tabellen lassen sich aber problemlos mittels SQL ansprechen und können somit mit geringem Aufwand (per Datenbankview) in die Anwendung eingebunden werden.

Zur Nutzung der erstellten Datenbankobjekte und Prozeduren sind Rollen erforderlich, welche den Zugriff der Nutzer regulieren.
Es wurden zwei verschiedene Rollen erstellt, „PROJECT_SCHEMA_ADMIN_ROLE-.hdbrole“ für Administratoren und „PROJECT_SCHEMA_USER_ROLE.hdbrole“ für Anwender. Der Administrator hat dabei die Berechtigung jede Prozedur auszuführen, der Anwender ist auf GET- und einige wenige SET-Proceduren eingeschränkt, das Löschen und Bearbeiten von Daten ist nicht freigegeben bzw. weitestgehend eingeschränkt.
Die Implementierung der „PROJECT_SCHEMA_ADMIN_ROLE.hdbrole“ ist in Listing 6.2 auf der nächsten Seite abgebildet.

```
// Eine Rolle fuer die Administration des Schemas
role abat.demosystem.projektcontrolling.controlling.data::
    PROJECT_SCHEMA_ADMIN_ROLE
extends role abat.demosystem.projektcontrolling.controlling.data::
    PROJECT_SCHEMA_USER_ROLE
{
catalog schema "PROJECT_SCHEMA" : SELECT, INSERT, UPDATE, DROP, DELETE;

catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::WORK_LAST" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::GET_PROJECT_STATUS" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::GET_PSP_STATUS" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::GET_PSP_WEEK_INFORMATION" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::GET_PSP_DETAIL_WEEK_INFO" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::GET_PSP_STATUS_DETAIL" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::SET_SKILL_PSP_MA" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::SET_PROJECT" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::SET_SKILL_PSP_RATE" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::SET_PSP" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::UPDATE_SKILL_PSP_MA" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::UPDATE_PROJECT" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::UPDATE_SKILL_PSP_RATE" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::UPDATE_PSP" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::DELETE_SKILL_PSP_MA" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::DELETE_PROJECT" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::DELETE_SKILL_PSP_RATE" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::DELETE_PSP" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::CLOSE_PSP" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::SET_PLAN_VALUES" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::UPDATE_PLAN_VALUES" : EXECUTE;
catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::DELETE_PLAN_VALUES" : EXECUTE;
```

```
29 catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.procedures::UPDATE_CLOSE_DATE" : EXECUTE;
30 catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.procedures::GET_PLAN_VALUES" : EXECUTE;
31 catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.procedures::GET_CLOSED_ENTRYS" : EXECUTE;
32 catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.procedures::EXCEL_EXPORT" : EXECUTE;
33 catalog sql object "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.procedures::UPDATE_PROJECT_SCORE_VALUE" : EXECUTE;
34
35 application privilege: abat.demosystem.projektcontrolling.controlling::
       Admin;
36 }
```

Listing 6.2: Projektcontrolling: Implementierung PROJECT_SCHEMA_ ADMIN_ROLE.hdbrole

models

Dieses Paket beinhaltet alle Models (Datenbankviews) des Szenarios. Anders als beim Sensorikszenario wurde die Implementierung mittels SQL vorgenommen, nachdem die „.hdbview“-Dateien erstellt wurden. Somit wurden beide Varianten ausgiebig getestet. Abbildung 6.5 zeigt alle Models, welche für das Szenario erforderlich sind.

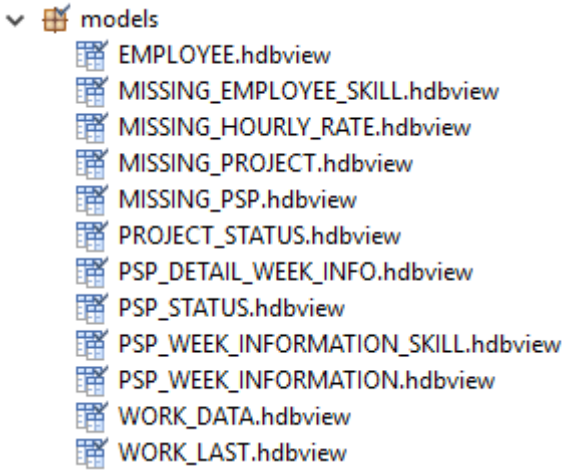


Abbildung 6.5.: Projektcontrolling: Models

In der Regel war die Aufgabe der Views, das Ausgabeformat für die verschiedenen Prozeduren zu definieren, damit diese über die Services korrekt aufgerufen werden konnten. Eine Ausnahme bildet dabei die View „WORK_DATA“, in der für jeden Arbeitseintrag eine Zeile vorhanden ist. Die View wurde, wie in Abbildung 6.4 auf Seite 204 dargestellt, genutzt, um die Daten der Tabellen zu konsolidieren und so einen zentralen Anlaufpunkt für alle Auswertungen bereitzustellen. Die Implementierung dieser View ist in Listing 6.3 auf der nächsten Seite abgebildet.

```
schema = "PROJECT_SCHEMA";

query = "select
DONE.RPROJ_EXTERN AS PSP_NUMBER,
ELEMENTS.PROJECTNUMBER,
ELEMENTS.DESCRIPTION AS ELEMENT_DESCRIPTION,
ELEMENTS.ORDER_NUMBER,
DONE.CATSHOURS AS HOURS,
RATES.HOURLY_RATE,
DONE.CATSHOURS * RATES.HOURLY_RATE as RATED,
ELEMENTS.LIMIT_YELLOW AS PSP_LIMIT_YELLOW,
ELEMENTS.LIMIT_RED AS PSP_LIMIT_RED,
ELEMENTS.BUDGET AS PLANNED_PSP_BUDGET,
ELEMENTS.HOURS AS PLANNED_PSP_HOURS,
PROJECT.LIMIT_YELLOW AS PROJECT_LIMIT_YELLOW,
PROJECT.LIMIT_RED AS PROJECT_LIMIT_RED,
PROJECT.BUDGET AS PLANNED_PROJECT_BUDGET,
PROJECT.HOURS AS PLANNED_PROJECT_HOURS,
DONE.FAKT AS INVOICED,
CONCAT(right(left(DONE.LOADDATE, 8),2),
CONCAT('.',
CONCAT(right(left(DONE.LOADDATE, 6),2),
CONCAT('.',left(DONE.LOADDATE,4))))) AS ENTRY_DATE,
CONCAT(right(left(DONE.WORKDATE, 8),2),
CONCAT('.',
CONCAT(right(left(DONE.WORKDATE, 6),2),
CONCAT('.',left(DONE.WORKDATE,4))))) AS DATE,
EXTRACT(YEAR FROM DONE.WORKDATE) AS YEAR,
TO_DECIMAL(DONE.WEEK) AS WEEK,
DONE.LTXA AS WORK_DESCRIPTION,
SKILL.EMPLOYEE_ID,
SKILL.LEVEL,
EMPLOYEE.TOKEN,
EMPLOYEE.SURNAME,
EMPLOYEE.FIRST_NAME,
ELEMENTS.CLOSED,
ELEMENTS.CLOSED_DATE,
(CASE WHEN ELEMENTS.CLOSED_DATE < DONE.WORKDATE AND CLOSED = 'X' THEN 'X'
    ELSE '' END) AS CLOSED_ENTRY
FROM
\"PROJECT_SCHEMA\".\"abat.demosystem.projektcontrolling.controlling.data::
    PROJECT\" PROJECT,
\"PROJECT_SCHEMA\".\"abat.demosystem.projektcontrolling.controlling.data::
    SKILL_PSP_COMBINATION\" SKILL,
\"PROJECT_SCHEMA\".\"abat.demosystem.projektcontrolling.controlling.data::
    HOURLY_RATES\" RATES,
\"SAPHANADB\".\"ZDOHA_PC\" DONE,
\"PROJECT_SCHEMA\".\"abat.demosystem.projektcontrolling.controlling.data::
    PSP_ELEMENT\" ELEMENTS,
\"PROJECT_SCHEMA\".\"abat.demosystem.projektcontrolling.controlling.models
    ::EMPLOYEE\" EMPLOYEE
WHERE
```

```
47 DONE.RPROJ_EXTERN = ELEMENTS.PSP_NUMBER
48 AND SKILL.PSP_NUMBER = RATES.PSP_NUMBER
49 AND RATES.PSP_NUMBER = DONE.RPROJ_EXTERN
50 AND SKILL.LEVEL = RATES.LEVEL
51 AND SKILL.EMPLOYEE_ID = DONE.PERNR
52 AND SKILL.EMPLOYEE_ID = EMPLOYEE.EMPLOYEE_ID
53 AND PROJECT.PROJECTNUMBER = ELEMENTS.PROJECTNUMBER" ;
```

Listing 6.3: Projektcontrolling: Implementierung WORK_DATA.hdbview

Der Aufbau des Quellcodes für die implementierten Views ist für alle Views identisch. Als erstes wird, wie bei Datenbanktabellen in Zeile 1, das Schema referenziert. Anschließend wird hinter dem Schlüsselwort „query =“ der select-Befehl für die View definiert, welcher festlegt, welche Spalten in der View vorhanden sind (Zeile 3 - 38). Abschließend folgen die Tabellen bzw. Views, aus denen die Daten geladen werden sollen (Zeile 39 - 45) und die Join-Bedingungen in Zeile 46 - 53.
Während es sich bei den meisten Befehlen um einfaches Hinzufügen von Spalten aus anderen Tabellen handelt, werden in dieser View auch wichtige Berechnungen durchgeführt. So wird in Zeile 10 die Arbeitszeit des Eintrags mit dem hinterlegten Stundensatz bewertet, was für die Berechnung des gesamten Budgetverbrauchs eines Projekts erforderlich ist.
In Zeile 20 - 27 werden zusätzlich die Datumsformate aus der Tabelle „ZDOHA_PC“ bereinigt. Dies ist erforderlich, da diese Tabelle mittels der SAP GUI erstellt wurde und das Datumsformat von den HANA-Studio-Tabellen abweicht.
Vor der Umformung hat das Datum in der Ursprungstabelle das Format „YYYYMMDD“, anschließend liegt das Format „DD.MM.YYYY“ vor, welches z.B. in den Prozeduren mit diversen Datumsfunktionen kompatibel ist.
Ebenfalls wird in Zeile 38 die Prüfung durchgeführt, ob ein Datensatz als gesperrt gebucht markiert werden muss. Dies ist der Fall, wenn das Datum eines Arbeitseintrags gleich oder größer dem Sperrdatum eines PSP-Elements ist. Erfüllt ein Eintrag diese Bedingung wird dieser mit einem „X“ markiert.
Abbildung 6.6 zeigt einen Auszug der View „WORK_DATA“.

	PSP_NUMBER	PROJECTNUMBER	ELEMENT_DESCRIPTION	ORDER_NUMBER	HOURS	HOURLY_RATE	RATED	PSP_LIMIT_YELLOW	PSP_LIMIT_RED
1	T-1990-02-R	1990	Element 2 Remote		0	42,5	0	40	60
2	T-1990-02-R	1990	Element 2 Remote		4	42,5	170	40	60
3	T-1990-02-R	1990	Element 2 Remote		7	42,5	297,5	40	60
4	T-1990-02-R	1990	Element 2 Remote		4	42,5	170	40	60
5	T-1990-02-R	1990	Element 2 Remote		7	42,5	297,5	40	60

ENTRY_DATE	DATE	YEAR	WEEK	WORK_DESCRIPTION	EMPLOYEE_ID	LEVEL	TOKEN	SURNAME	FIRST_NAME
31.01.2017	05.08.2016	2.016	31	Reise	00000017	Senior Consultant	TAW	Witte	Tanja
31.01.2017	05.08.2016	2.016	31	Reise	00000017	Senior Consultant	TAW	Witte	Tanja
07.02.2017	01.08.2016	2.016	31	Reise	00000017	Senior Consultant	TAW	Witte	Tanja
24.02.2017	05.08.2016	2.016	31	Reise	00000017	Senior Consultant	TAW	Witte	Tanja
26.02.2017	01.08.2016	2.016	31	Reise	00000017	Senior Consultant	TAW	Witte	Tanja

Abbildung 6.6.: Projektcontrolling: WORK_DATA (Auszug)

Abfragen auf diese View ermöglichen beispielsweise die Auswertung nach Projekten, PSP-Elementen oder Skills. Ebenfalls kann die Auswertung mittels Zeitscheiben erfol-

gen, wie es im fertigen Prototyp der Fall ist. Zeitscheiben beschreiben Datenstände von verschiedenen Zeitpunkten, mit denen eine historische Auswertung von Projekten möglich wird. In diesem Szenario wird täglich eine Zeitscheibe erstellt (vgl. Abschnitt „ABAP-Applikation“ auf Seite 232). Die Möglichkeiten für Auswertungen sind aufgrund des hohen Informationsgehalts der View sehr vielseitig.
Abschließend folgt eine Auflistung der Views mit einer kurzen Funktionserläuterung:

- **EMPLOYEE** bindet die Mitarbeiterdaten aus den Tabellen der SAP GUI bzw. ABAP-Applikation in das Schema ein, damit vorher erstellte Prozeduren mit geringerem Anpassungsaufwand auf die Tabelle zugreifen können.

- **MISSING_EMPLOYEE_SKILL** enthält Mitarbeiter, die Arbeitszeit verbucht haben, denen kein Skill zugewiesen ist.

- **MISSING_HOURLY_RATE** enthält Skills, denen kein Stundensatz zugewiesen ist.

- **MISSING_PROJECT** enthält Projekte, auf die Arbeitszeit gebucht wurde, welche jedoch noch nicht geplant wurden.

- **MISSING_PSP** enthält PSP-Elemente, auf die Arbeitszeit gebucht wurde, welche jedoch noch nicht geplant wurden.

- **PROJECT_STATUS** enthält den Status aller Projekte (Budget-, Stundenverbrauch, Planwerte, prozentuale Verbräuche etc.) für die verschiedenen Zeitscheiben.

- **PSP_DETAIL_WEEK_INFO** enthält die Arbeitszeitdaten auf detaillierter Ebene. Der Inhalt wird in dieser Form nicht genutzt. Die View wird lediglich dafür genutzt, das Ausgabeformat von Prozeduren zu definieren, die ihre Daten aus der WORK_DATA-View beziehen.

- **PSP_STATUS** enthält den Status aller PSP-Elemente für die verschiedenen Zeitscheiben.

- **PSP_WEEK_INFORMATION_SKILL** enthält die Arbeitszeitdaten aggregiert nach PSP-Element, Skill und Kalenderwoche.

- **PSP_WEEK_INFORMATION** enthält die Arbeitszeitdaten, aggregiert nach PSP-Element und Kalenderwoche.

- **WORK_DATA** dient aus Anlaufpunkt für die meisten Auswertungen (vgl. Listing 6.3 auf Seite 208).

- **WORK_LAST** enthält das Datum der letzten Abgabe von Arbeitszeitdaten aller Mitarbeiter.

procedures

Die Prozeduren des Szenarios sind integraler Bestandteil für Datenauswertung und Dateneingabe. In diesem Abschnitt werden einige Prozeduren exemplarisch erläutert. Prinzipiell sind die Prozeduren unterteilt und GET-, SET-, UPDATE-, und DELETE-Funktionen, um einen konsistenten Datenstand zu gewährleisten. Abbildung 6.7 zeigt die implementierten Prozeduren.

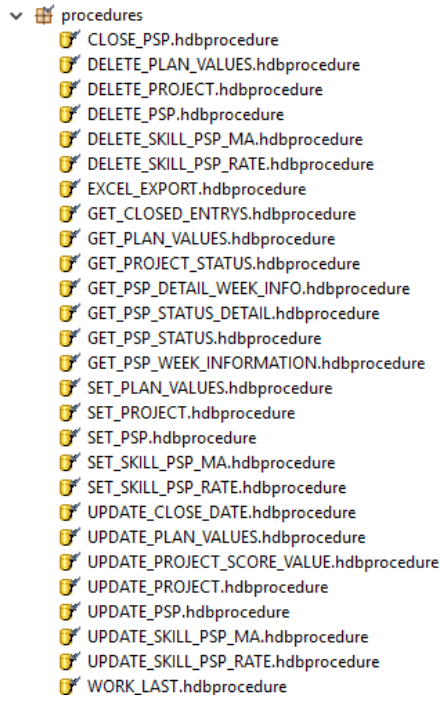


Abbildung 6.7.: Projektcontrolling: Procedures

GET-Prozeduren Die GET-Prozeduren werden zur Auswertung der vorhandenen Daten genutzt. In der Regel werden hierbei Projekte bzw. PSP-Elemente gemäß verschiedener Anforderungen ausgewertet, z.B. ein Projekt in der Projekt-Auswertung oder eine Wochenauswertung für ein PSP-Element.
Listing 6.4 zeigt die Implementierung der Prozedur „GET_PSP_STATUS“.

```
1  /* Procedure zur Abfrage des Status aller PSP-Elemente eines Projekts.
2  Aufruf mit Projektnummer, Zeitscheibe;
3  Ausgabe = Status aller zugehoerigen PSP-Elemente zum Projekt
4  mit den geleisteten Stunden sowie den sich ergebenden Kosten.
5  Ausgabeformat wie View "PSP_STATUS" */
6  PROCEDURE "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling
       .procedures::GET_PSP_STATUS" (
7  IN pnr varchar (200),
8  IN ENTRY_DATE date,
9  OUT opt "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.
       models::PSP_STATUS")
10 LANGUAGE SQLSCRIPT
```

```
11 AS
12 BEGIN
13 /* Abfragen des Status der PSP-Elemente des Projekts, sortiert aufsteigend
       nach PSP-Nummer */
14 opt = select * from "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.models::PSP_STATUS"
15 WHERE PROJECTNUMBER = :pnr
16 AND ENTRY_DATE = :ENTRY_DATE
17 ORDER BY PSP_NUMBER ASC;
18 END;
```

Listing 6.4: Projektcontrolling: Implementierung GET_PSP_STATUS.hdbprocedure

Die Prozedur wird in Zeile 7 - 8 mit Projektnummer und Zeitscheibe aufgerufen. Anschließend werden aus der zugeordneten View („PSP_STATUS“) alle Einträge aufsteigend nach PSP-Nummer zurückgegeben, die zur eingegebenen Projektnummer und zur Zeitscheibe passen (Zeile 14-17). Diese Daten werden auf der grafischen Oberfläche für den Nutzer visualisiert.
Abbildung 6.8 zeigt den Aufruf und die Rückgabewerte der Prozedur für das Projekt „1990“ und die Zeitscheibe vom 28.02.2017.

```
CALL "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.procedures::GET_PSP_STATUS" ('1990','28.02.2017',?)
```

	PROJECTNUMBER	PSP_NUMBER	ELEMENT_DESCRIPTION	PLANNED_BUDGET	PLANNED_HOURS	TOTAL_HOURS	TOTAL_RATED
1	1990	T-1990-01	Element 1	7.000	40	35	2.975
2	1990	T-1990-02	Element 2	7.000	30	0	0
3	1990	T-1990-02-R	Element 2 Remote	1.000	10	11	467,5

BUDGET_PERCENT	HOURS_PERCENT	PSP_LIMIT_YELLOW	PSP_LIMIT_RED	ORDER_NUMBER	INVOICED	CLOSED	CLOSED_DATE	ENTRY_DATE
43	88	40	60		X		?	28.02.2017
0	0	50	85				?	28.02.2017
47	110	40	60				?	28.02.2017

Abbildung 6.8.: Projektcontrolling: Ergebnis der Prozedur „GET_PSP_STATUS“

In Tabelle 6.5 auf der nächsten Seite sind alle erstellten GET-Prozeduren, sowie ihre Ein- und Ausgabewerte aufgeführt.

Tabelle 6.5.: Projektcontrolling: GET-Procedures

Name und Funktion	**Input**	**Output (Table/View)**
GET_CLOSED_ENTRYS (Abruf gesperrter Datensätze eines Projekts)	PROJECTNUMBER: VARCHAR(200), ENTRY_DATE: DATE	PSP_DETAIL_WEEK_INFO
GET_PLAN_VALUES (Abruf von Planwerten zu einem PSP-Element nach Wochen)	PSP_NUMBER: VARCHAR(200), WEEK: DECIMAL, YEAR: DECIMAL	PSP_WEEK_ INFORMATION_SKILL
GET_PROJECT_STATUS (Abruf des Projektstatus eines Projekts)	PROJECTNUMBER: VARCHAR(200), ENTRY_DATE: DATE	PROJECT_STATUS

Tabelle 6.5.: Projektcontrolling: GET-Procedures

Name und Funktion	**Input**	**Output (Table/View)**
GET_PSP_DETAIL_WEEK_INFO (Abruf von Arbeitseinträgen zu einem PSP-Element nach Woche)	PSP_NUMBER: VARCHAR(200), WEEK: DECIMAL, ENTRY_DATE: DATE	PSP_DETAIL_WEEK_INFO
GET_PSP_STATUS_DETAIL (Abruf des Status eines PSP-Elements)	PSP_NUMBER: VARCHAR(200), ENTRY_DATE: DATE	PSP_STATUS
GET_PSP_STATUS (Abruf des Status aller PSP-Elemente eines Projekts)	PROJECTNUMBER: VARCHAR(200), ENTRY_DATE: DATE	PSP_STATUS
GET_PSP_WEEK_INFORMATION (Abruf von Arbeitseinträgen zu einem PSP-Element nach Wochen gruppiert)	PSP_NUMBER: VARCHAR(200), ENTRY_DATE: DATE	PSP_WEEK_ INFORMATION
EXCEL_EXPORT (Abruf granularer Projektinformationen für den Export als CSV-Datei)	PROJECTNUMBER: VARCHAR(200), ENTRY_DATE: DATE	WORK_DATA
WORK_LAST (Abruf von Mitarbeitern, die nach dem Delimiter keine Arbeitsinformationen eingereicht haben)	DELIMITER: DATE	WORK_LAST

SET-Prozeduren Die SET-, DELETE- und UPDATE-Prozeduren werden z.B. genutzt, um neue Projekte anzulegen, das Budget von PSP-Elementen anzupassen oder Projekte inklusive aller zugeordneten Elemente (PSP-Elemente, Stundensätze, Planwerte, etc.) zu löschen. Die vordefinierten Befehle innerhalb der Prozeduren sorgen dafür, dass der Datenstand immer korrekt ist und keine unerlaubten SQL-Statements abgeschickt werden. Alle SET-, DELETE-, und UPDATE-Prozeduren geben einen Prüfwert zurück (0 oder 1), welcher anzeigt, ob erfolgreich in die Datenbank geschrieben wurde. Listing 6.5 zeigt die Implementierung der Prozedur „SET_PSP.hdbprocedure“, welche genutzt wird, um neue PSP-Elemente anzulegen.

```
1 /* Procedure zum Anlegen von PSP-Elementen.
2 Aufruf mit PSP-Nummer, Projektnummer, Budget, Planstunden, Beschreibung,
    Auftragsnummer, Schwelle Rot, Schwelle Gelb, Auswertungszeichen;
    Ausgabe = Pruefwert */
3 PROCEDURE "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling
```

```
    .procedures::SET_PSP" (
IN PSP VARCHAR(200),
IN DESCRIPTION VARCHAR(2000),
IN PROJECTNUMBER VARCHAR(200),
IN ORDER_NUMBER VARCHAR(200),
IN LIMIT_YELLOW DECIMAL (34),
IN LIMIT_RED DECIMAL (34),
IN BUDGET DECIMAL (34),
IN HOURS DECIMAL (34),
OUT opt DUMMY)
LANGUAGE SQLSCRIPT
AS
BEGIN

/* Variablendeklaration:
"found" enthaelt die Pruefung, ob zum PSP-Element das angegebene Projekt
    vorhanden ist
"PSPs" enthaelt die Information, ob bereits weitere PSP-Elemente zum
    Projekt geplant wurden.
"reststunden" enthaelt die Information, wieviele Stunden noch zum planen
    verfuegbar sind
"restbudget" enthaelt die Information, wieviel Budget noch zum planen
    verfuegbar ist
"ROWCOUNT" enthaelt Anzahl der geschriebenen Zeilen als Pruefwert */

DECLARE found INT := 1;
DECLARE PSPs INT := 1;
DECLARE restbudget DECIMAL(10,4) :=1;
DECLARE reststunden DECIMAL(10,4) :=1;
DECLARE ROWCOUNT INT :=1;

SELECT count(*) INTO found FROM "PROJECT_SCHEMA"."abat.demosystem.
    projektcontrolling.controlling.data::PROJECT" WHERE PROJECTNUMBER = :
    PROJECTNUMBER;
SELECT count (*) INTO PSPs FROM "PROJECT_SCHEMA"."abat.demosystem.
    projektcontrolling.controlling.data::PSP_ELEMENT" WHERE PROJECTNUMBER =
     :PROJECTNUMBER;

/* Wenn bereits PSP-Elemente geplant wurden, wird geprueft, ob die
    Planwerte der vorhandenen PSP-Elemente +
das zusaetzliche PSP-Element mit dem zugeordneten Planbudget und den
    zugeordneten Planstunden die Planstunden oder
das Planbudget des uebergeordneten Projekts uebersteigt.
Wird das Projekt im Budget oder den Stunden ueberplant, wird das PSP-
    Element nicht gespeichert
Die Anzahl der Reststunden wird in die Variable "reststunden" geschrieben.
Das Restbudget wird in die Variable "restbudget" geschrieben. */
IF :PSPs > 0
THEN
SELECT ((PROJECT.BUDGET - (
select sum(BUDGET) from "PROJECT_SCHEMA"."abat.demosystem.
    projektcontrolling.controlling.data::PSP_ELEMENT"
```

```
43 WHERE PROJECTNUMBER = :PROJECTNUMBER)) - :BUDGET) INTO restbudget
44 FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.data
       ::PROJECT" PROJECT
45 WHERE PROJECT.PROJECTNUMBER = :PROJECTNUMBER
46 GROUP BY PROJECT.BUDGET;
47 SELECT ((PROJECT.HOURS - (
48 select sum(HOURS) from "PROJECT_SCHEMA"."abat.demosystem.
       projektcontrolling.controlling.data::PSP_ELEMENT"
49 WHERE PROJECTNUMBER = :PROJECTNUMBER)) - :HOURS) INTO reststunden
50 FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.data
       ::PROJECT" PROJECT
51 WHERE PROJECT.PROJECTNUMBER = :PROJECTNUMBER
52 GROUP BY PROJECT.HOURS;
53 ELSE
54
55 /* Sind keine weiteren PSP-Elemente vorhanden wird ebenfalls geprueft, ob
       das neue PSP-Element
56 die Planstunden oder das Planbudget des uebergeordneten Projekts
       uebersteigt.
57 Wird das Projekt im Budget oder den Stunden ueberplant, wird das PSP-
       Element nicht gespeichert
58 Die Anzahl der Reststunden wird in die Variable "reststunden" geschrieben.
59 Das Restbudget wird in die Variable "restbudget" geschrieben. */
60 SELECT PROJECT.BUDGET - :BUDGET INTO restbudget
61 FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.data
       ::PROJECT" PROJECT
62 WHERE PROJECT.PROJECTNUMBER = :PROJECTNUMBER
63 GROUP BY PROJECT.BUDGET;
64 SELECT PROJECT.HOURS - :HOURS INTO reststunden
65 FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.data
       ::PROJECT" PROJECT
66 WHERE PROJECT.PROJECTNUMBER = :PROJECTNUMBER
67 GROUP BY PROJECT.HOURS;
68 END IF;
69
70 /* Der Wert wird nur in die Datenbank geschrieben, wenn "found" groesser
       null ist und "restbudget" und "reststunden" groesser / gleich null sind
       . */
71 IF :found > 0 AND :restbudget >= 0 AND reststunden >= 0
72 THEN
73 INSERT INTO "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.data::PSP_ELEMENT"
74 VALUES(:PSP, :DESCRIPTION, :PROJECTNUMBER, :ORDER_NUMBER, :LIMIT_YELLOW, :
       LIMIT_RED, :BUDGET ,:HOURS,'','');
75 ELSE
76 END IF;
77 /* Rueckgabe der Anzahl der geschriebenen Zeilen fuer Pruefungszwecke in
       der GUI (aktualisieren erfolgreich ja/nein) */
78 SELECT ::ROWCOUNT into ROWCOUNT FROM DUMMY;
79 opt = SELECT :ROWCOUNT AS DUMMY FROM DUMMY;
80 END;
```

Listing 6.5: Projektcontrolling: Implementierung SET_PSP.hdbprocedure

Bevor der Datensatz in der Datenbank gespeichert wird, erfolgen diverse Prüfungen, ob die eingegebenen Daten zum vorhandenen Datenstand passen.
Zunächst wird in Zeile 30 ermittelt, ob das angegebene Projekt vorhanden ist, zu dem das PSP-Element gehört. Hierzu wird die Anzahl der Projekte ermittelt, die die angegebene Nummer haben und in einer Variable gespeichert. Da ein Projekt nur einmal vorhanden sein kann, ist in der Variable „found“ also eine 1 bzw. 0 gespeichert. In Zeile 31 wird ermittelt, ob bereits weitere PSP-Elemente zum Projekt vorhanden sind. Dies ist erforderlich, um das bereits eingeplante Projektbudget zu ermitteln und somit eine Überplanung zu verhindern (das Budget der einzelnen PSP-Elemente kann in der Summe nicht höher als das des übergeordneten Projekts sein).
Sollten weitere PSP-Elemente vorhanden sein, wird ermittelt, ob das geplante Budget des neuen PSP-Elements - addiert mit dem bereits geplanten Budget - das Projektbudget übersteigt. Diese Prüfung erfolgt in Zeile 39 - 46.
In Zeile 47 - 52 wird die Prüfung der Planstunden analog zum Planbudget vorgenommen.
Sollten keine anderen PSP-Elemente vorhanden sein, müssen nur die eingegebenen Planwerte mit den Projektwerten verglichen werden (Zeile 60 - 68). Das Restbudget und die Reststunden werden in den Variablen „restbudget“ und „reststunden“ gespeichert.
Ist das angegebene Projekt vorhanden (Variable „found“ = 1) und wurden Planstunden sowie Planbudget nicht überplant (Variablen „reststunden“ und „restbudget“ >= 0), wird das eingegebene PSP-Element in Zeile 71 - 76 gespeichert.
Abschließend erfolgt die Rückgabe eines Prüfwerts (1 oder 0), der zeigt, ob ein Wert in die Datenbank geschrieben wurde. Der Prüfwert wird in der GUI als Feedback für den Anwender genutzt (Zeile 78 - 80).
Die Erläuterung dieser SET-Prozedur soll exemplarisch darstellen, wie die SET-Prozeduren des Szenarios arbeiten. Die übergebenen Werte werden zunächst auf Korrektheit geprüft und sofern diese gegeben ist, in der Datenbank gesichert. Tabelle 6.6 auf der nächsten Seite zeigt alle SET-Prozeduren des Szenarios.

Tabelle 6.6.: Projektcontrolling: SET-Procedures

Name und Funktion	**Input**
SET_PLAN_VALUES (Anlegen von Planwerten zu einem PSP-Element)	PSP_NUMBER: VARCHAR(200), LEVEL: VARCHAR(100), PLANNED_HOURS: DECIMAL, PLANNED_WEEK: DECIMAL, PLANNED_YEAR: DECIMAL
SET_PROJECT (Anlegen eines Projekts)	PROJECTNUMBER: VARCHAR(200), BUDGET: DECIMAL, HOURS: DECIMAL, FIXED_VALUES: CHAR(1), DESCRIPTION: VARCHAR(2000), LIMIT_YELLOW: DECIMAL, LIMIT_RED: DECIMAL, SCORE_VALUE: CHAR(1)

Tabelle 6.6.: Projektcontrolling: SET-Procedures

Name und Funktion	Input
SET_PSP (Anlegen eines PSP-Elements)	PSP_NUMBER: VARCHAR(200), DESCRIPTION: VARCHAR(2000), PROJECTNUMBER: VARCHAR(200), ORDER_NUMBER: VARCHAR(200), LIMIT_YELLOW: DECIMAL, LIMIT_RED: DECIMAL, BUDGET: DECIMAL, HOURS: DECIMAL
SET_SKILL_PSP_MA (Anlegen von Skill-PSP-Mitarbeiter-Kombinationen)	LEVEL: VARCHAR(100), PSP_NUMBER: VARCHAR(200), NAME: VARCHAR(200)
SET_SKILL_PSP_RATE (Anlegen von Skill-PSP-Stundensatz-Kombinationen)	LEVEL: VARCHAR(100), PSP_NUMBER: VARCHAR(200), HOURLY_RATE: DECIMAL
CLOSE_PSP (Sperren von PSP-Elementen)	PSP_NUMBER: VARCHAR(200)

UPDATE-Prozeduren Zum Aktualisieren vorhandener Datensätze werden die UPDATE-Prozeduren genutzt. Diese können beispielsweise genutzt werden, um die Beschreibung und das Budget eines Projekts anzupassen oder die Stundensätze für einen bestimmten Skill zu aktualisieren.
Listing 6.6 zeigt die Implementierung der Prozedur „UPDATE_PROJECT.hdbprocedure“.

```
1  /* Procedure zum Aktualisieren von Projekten.
2  Aufruf mit Projektnummer, Budget, Stunden, Fixwertkennzeichen,
       Beschreibung, Limit gelb, Limit Rot; Ausgabe = Pruefwert */
3  PROCEDURE "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling
       .procedures::UPDATE_PROJECT" (
4  IN PROJECTNUMBER VARCHAR(200),
5  IN BUDGET DECIMAL (34),
6  IN HOURS DECIMAL,
7  IN FIXED_VALUES CHAR(1),
8  IN DESCRIPTION VARCHAR(2000),
9  IN LIMIT_YELLOW DECIMAL (34),
10 IN LIMIT_RED DECIMAL (34),
11 OUT opt DUMMY)
12 LANGUAGE SQLSCRIPT
13 AS
14 BEGIN
15
16 /* Variablendeklaration:
17 "altbudget" enthaelt das urspruengliche Budget des Projekts
18 "altstunden" enthaelt die urspruenglichen Planstunden des Projekts
19 "ROWCOUNT" enthaelt Anzahl der geschriebenen Zeilen als Pruefwert */
20 DECLARE altbudget DECIMAL(10,4) :=1;
```

```
21 DECLARE altstunden DECIMAL(10,4) :=1;
22 DECLARE ROWCOUNT INT :=1;
23
24 /* Speichern von Altbudget und Altstunden fuer Pruefzwecke */
25 select BUDGET into altbudget from "PROJECT_SCHEMA"."abat.demosystem.
       projektcontrolling.controlling.data::PROJECT"
26 where PROJECTNUMBER = :PROJECTNUMBER;
27
28 select HOURS into altstunden from "PROJECT_SCHEMA"."abat.demosystem.
       projektcontrolling.controlling.data::PROJECT"
29 where PROJECTNUMBER = :PROJECTNUMBER;
30
31 /* Sind die neuen Stunden oder das neue Budget geringer als die Altwerte,
       werden bei allen zugehoerigen PSP-Elementen
32 die Planwerte auf null gesetzt, um ein Ueberplanen zu verhindern. */
33 IF :altbudget > :BUDGET OR :altstunden > :HOURS
34 THEN
35 UPDATE "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.
       data::PSP_ELEMENT" SET
36 BUDGET = 0, HOURS = 0
37 WHERE PROJECTNUMBER = :PROJECTNUMBER;
38 END IF;
39
40 /* Aktualisieren des Projekts mit den neuen Werten */
41 UPDATE "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.
       data::PROJECT" SET
42 BUDGET = :BUDGET, HOURS = :HOURS, FIXED_VALUES = :FIXED_VALUES,
43 DESCRIPTION = :DESCRIPTION, LIMIT_YELLOW = :LIMIT_YELLOW, LIMIT_RED = :
       LIMIT_RED
44 WHERE PROJECTNUMBER = :PROJECTNUMBER;
45
46 /* Rueckgabe der Anzahl der geschriebenen Zeilen fuer Pruefungszwecke in
       der GUI (Aktualisieren erfolgreich ja/nein) */
47 SELECT ::ROWCOUNT into ROWCOUNT FROM DUMMY;
48 opt = SELECT :ROWCOUNT AS DUMMY FROM DUMMY;
49
50 END;
```

Listing 6.6: Projektcontrolling: Implementierung UPDATE_PROJECT.hdbprocedure

Ähnlich wie bei der erläuterten SET-Prozedur (vgl. Listing 6.5 auf Seite 213), erfolgen auch bei dieser Art von Prozedur Prüfungen, bevor ein Datensatz aktualisiert wird. Bei der Aktualisierung von Projekten wird geprüft, ob sich das geplante Budget- bzw. Stundenkontigent verändert hat. Sollte sich einer dieser Werte verringern, werden bei allen zugeordneten PSP-Elementen die Planwerte auf 0 gesetzt, um zu verhindern, dass die Summen der bereits vergebenen Planwerte der PSP-Elemente die des Projekts überschreiten und es zu Inkonsistenzen kommt.
Hierzu werden erst die ursprünglichen Werte ermittelt und in Variablen gesichert (Zeile 25 - 29). In den folgenden Zeilen 33 - 38 werden anschließend die ursprünglichen Planwerte mit den übergebenen Planwerten verglichen und ggf. die Planwerte der PSP-Elemente zurückgesetzt. Abschließend erfolgt das Update des Projekts in Zeile 41 - 44

und ein Prüfwert mit dem Speicherstatus wird zurückgegeben (Zeile 47 - 48).
Tabelle 6.7 zeigt alle UPDATE-Prozeduren des Szenarios.

Tabelle 6.7.: Projektcontrolling: UPDATE-Procedures

Name und Funktion	**Input**
UPDATE_CLOSE_ DATE (Anpassen des Sperrdatums von gesperrten PSP-Elementen)	PSP_NUMBER: VARCHAR(200), DATE: DATE
UPDATE_PLAN_ VALUES (Anpassen von Planwerten zu PSP-Elementen)	PSP_NUMBER: VARCHAR(200), LEVEL: VARCHAR(100), PLANNED_WEEK: DECIMAL, PLANNED_YEAR: DECIMAL, PLANNED_HOURS (new): DECIMAL, PLANNED_WEEK (new): DECIMAL, PLANNED_YEAR (new): DECIMAL
UPDATE_PROJECT _SCORE_VALUE (Anpassen des Auswertungskennzeichens von Projekten)	PROJECTNUMBER: VARCHAR(200), SCORE_VALUE: CHAR(1)
UPDATE_PROJECT (Anpassen von Projekten)	PROJECTNUMBER: VARCHAR(200), BUDGET: DECIMAL, HOURS: DECIMAL, FIXED_VALUES: CHAR(1), DESCRIPTION: VARCHAR(2000), LIMIT_YELLOW: DECIMAL, LIMIT_RED: DECIMAL
UPDATE_PSP (Anpassen von PSP-Elementen)	PSP_NUMBER: VARCHAR(200), DESCRIPTION: VARCHAR(2000), PROJECTNUMBER: VARCHAR(200), ORDER_NUMBER(200), LIMIT_YELLOW: DECIMAL, LIMIT_RED: DECIMAL, BUDGET: DECIMAL, HOURS: DECIMAL
UPDATE_SKILL_ PSP_MA (Anpassen von Skill-PSP-MA-Kombinationen)	LEVEL: VARCHAR(100), PSP_NUMBER: VARCHAR(200), NAME: VARCHAR(200)
UPDATE_SKILL_ PSP_RATE (Anpassen von Stundensätzen)	LEVEL: VARCHAR(100), PSP_NUMBER: VARCHAR(200), HOURLY_RATE: DECIMAL

DELETE-Prozeduren Zum Löschen von Datensätzen werden die DELETE-Prozeduren genutzt. Die Prozeduren entfernen dabei Daten gemäß ihrer Zuordnung. Wird z.B. ein Projekt gelöscht, werden auch alle zugeordneten PSP-Elemente, Stundensätze, Mitarbeiterzuordnungen und Planwerte gelöscht. Listing 6.7 auf der nächsten Seite zeigt die

Implementierung der Prozedur „DELETE_PROJECT.hdbprocedure“, welche zum Entfernen von Projekten genutzt wird.

```
1  /* Procedure zum Loeschen von Projekten und allen Eintraegen, die auf das
       Projekt verweisen (PSP, Stundensaetze, etc.).
2  Aufruf mit Projektnummer; Ausgabe = Pruefwert */
3  PROCEDURE "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling
       .procedures::DELETE_PROJECT" (
4  IN PROJECTNUMBER VARCHAR(200),
5  OUT opt DUMMY
6  )
7  LANGUAGE SQLSCRIPT
8  AS
9  BEGIN
10
11 /* Variable "ROWCOUNT" enthaelt Anzahl der geschriebenen Zeilen als
       Pruefwert */
12 DECLARE ROWCOUNT INT :=1;
13
14 /* Entfernen aller Stundensaetze von PSP-Elementen, die zum Projekt
       gehoeren */
15 DELETE FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.data::HOURLY_RATES"
16 WHERE PSP_NUMBER IN (select PSP_NUMBER FROM "PROJECT_SCHEMA"."abat.
       demosystem.projektcontrolling.controlling.data::PSP_ELEMENT" WHERE
       PROJECTNUMBER = :PROJECTNUMBER);
17
18 /* Entfernen aller Skill-PSP-Kombinationen von PSP-Elementen, die zum
       Projekt gehoeren */
19 DELETE FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.data::SKILL_PSP_COMBINATION"
20 WHERE PSP_NUMBER IN (select PSP_NUMBER FROM "PROJECT_SCHEMA"."abat.
       demosystem.projektcontrolling.controlling.data::PSP_ELEMENT" WHERE
       PROJECTNUMBER = :PROJECTNUMBER);
21
22 /* Entfernen aller Planwerte von PSP-Elementen, die zum Projekt gehoeren
       */
23 DELETE FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.data::PLAN_VALUES"
24 WHERE PSP_NUMBER IN (select PSP_NUMBER FROM "PROJECT_SCHEMA"."abat.
       demosystem.projektcontrolling.controlling.data::PSP_ELEMENT" WHERE
       PROJECTNUMBER = :PROJECTNUMBER);
25
26 /* Entfernen aller PSP-Elemente die zum Projekt gehoeren */
27 DELETE FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.data::PSP_ELEMENT" WHERE PROJECTNUMBER = :PROJECTNUMBER;
28
29 /* Loeschen des Projekts */
30 DELETE FROM "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
       controlling.data::PROJECT" WHERE PROJECTNUMBER = :PROJECTNUMBER;
31
32 /* Rueckgabe der Anzahl der geschriebenen Zeilen */
33 SELECT ::ROWCOUNT into ROWCOUNT FROM DUMMY;
```

```
34 opt = SELECT :ROWCOUNT AS DUMMY FROM DUMMY;
35
36 END;
```

Listing 6.7: Projektcontrolling: Implementierung DELETE_PROJECT.hdbprocedure

Die Prozedur durchläuft mit den hinterlegten Anweisungen mehrere Tabellen, um alle Elemente eines Projekts zu entfernen.
In Zeile 15 - 16 werden die Stundensätze des Projekts entfernt, indem alle Werte, die zu PSP-Elementen des Projekts gehören, aus der „HOURLY_RATES“-Tabelle entfernt werden.
In Zeile 19 - 20 erfolgt das Entfernen aller Skill-Mitarbeiter-Kombinationen aus der „SKILL_ PSP_COMBINATION“-Tabelle sowie in Zeile 23 - 24 das Entfernen aller Plan-Einträge aus der Tabelle „PLAN_VALUES“ analog zu den Stundensätzen.
Abschließend werden alle PSP-Elemente (Tabelle „PSP_ELEMENT“, Zeile 27) und das angegebene Projekt selbst (Tabelle „PROJECT“, Zeile 30) entfernt. In Zeile 33 - 34 erfolgt die Rückgabe des Prüfwerts.
Tabelle 6.8 zeigt alle DELETE-Prozeduren des Szenarios.

Tabelle 6.8.: Projektcontrolling: DELETE-Procedures

Name und Funktion	**Input**
DELETE_PLAN_ VALUES (Löschen von Planwerten zu PSP-Elementen)	PSP_NUMBER: VARCHAR(200), LEVEL: VARCHAR(100), PLANNED_WEEK: DECIMAL, PLANNED_YEAR: DECIMAL
DELETE_PROJECT (Löschen von Projekten)	PROJECTNUMBER: VARCHAR(200)
DELETE_PSP (Löschen von PSP-Elementen)	PSP_NUMBER: VARCHAR(200)
DELETE_SKILL_ PSP_MA (Löschen von Skill-PSP-MA-Kombinationen)	PSP_NUMBER: VARCHAR(200), NAME: VARCHAR(200)
DELETE_SKILL_ PSP_RATE (Löschen von Stundensätzen)	PSP_NUMBER: VARCHAR(200), LEVEL: VARCHAR(100)

services

Services, die mit der Datenbank kommunizieren sind im gleichnamigen Package enthalten. Für die Applikation werden xsjs-Services und xsodata-Services verwendet. Das Abrufen von Daten der xsjs-Services erfolgt über Prozeduren oder durch einen direkten Zugriff auf die Daten mit Hilfe eines im Service enthaltenen SQL-Befehls.

Auch xsodata-Services greifen direkt (ohne den Weg über Prozeduren) auf die Datenbank zu. Abgespeichert werden die Werte hingegen ausschließlich durch xsjs-Services, welche Prozeduren aufrufen. Die Möglichkeit einer beidseitigen Kommunikation anhand von xsodata-Services wurde nicht genutzt, da das Abspeichern der Werte Logik bedarf, die alleine mit xsodata-Services nicht abzudecken ist. Beispielsweise werden Fehlerbehandlungen mit dem Aufruf der Prozeduren realisiert.

Die Services lassen sich in vier Kategorien gliedern:

- Datenbankabfrage/ Missing Views
- Insert
- Update
- Export

Die Update- und Insert-Services sind schreibende Services, die Daten an Prozeduren übermitteln, welche wiederum die entsprechenden Daten im richtigen Format in die Datenbank schreiben. Die Services zur Datenbankanfrage bzw. Missing Views sind lesende Services, welche die Daten aus der Datenbank übermitteln. Unter Export fällt ein lesender Service, der die Daten im CSV-Format ausgibt.

Datenbankabfrage Um die Daten aus der Datenbank bereitstellen zu können, wurden xsodata-Services und xsjs-Services erstellt. Der Service „hourly_rates.xsodata“ ist ein Service, der für jede PSP-Nummer und Skill-Level einen Stundensatz ausgibt.

```
1 service{
2
3 "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.controlling.data::
    HOURLY_RATES" as
4  "HOURLY_RATES";
5
6  }
```

Listing 6.8: Projektcontrolling: hourly_rates.xsodata

Aufgerufen wird der Service mittels einer URL, indem der Pfad zum Service und das Ausgabeformat mit angegeben wird. Als Ausgabeformat wird in der Applikation JSON verwendet. Für den vorgestellten Service ist dies die folgende URL:

http://192.168.252.42:8002/abat/demosystem/projektcontrolling/controlling/services/hourly_rates.xsodata/HOURLY_RATES?$format=json

Als Ergebnis wird ein JSON ausgegeben, in dem die Daten der angesprochenen View bzw. Tabelle ausgegeben werden. Des Weiteren werden Metadaten in xsodata-Services standardmäßig mit ausgegeben. Das Ergebnis eines xsodata-Services entspricht Abbildung 6.9 auf der nächsten Seite.

```
{
  - d: {
      - results: [
          - {
              - __metadata: {
                    type: "abat.demosystem.projektcontrolling.controlling.services.hourly_rates.HOURLY_RATESType",
                    uri: "http://192.168.252.42:8002/abat/demosystem/projektcontrolling/controlling/services/hourly_rates.xsodata/HOURLY_RATES(LEVEL='Consultant',PSP_NUMBER='I-601-01-HIM')"
                },
                LEVEL: "Consultant",
                PSP_NUMBER: "I-601-01-HIM",
                HOURLY_RATE: "60"
            },
          - {
              - __metadata: {
                    type: "abat.demosystem.projektcontrolling.controlling.services.hourly_rates.HOURLY_RATESType",
                    uri: "http://192.168.252.42:8002/abat/demosystem/projektcontrolling/controlling/services/hourly_rates.xsodata/HOURLY_RATES(LEVEL='Consultant',PSP_NUMBER='I-610-01')"
                },
                LEVEL: "Consultant",
                PSP_NUMBER: "I-610-01",
                HOURLY_RATE: "60"
            },
```


Abbildung 6.9.: Projektcontrolling: Service Result hourly_rates.xsodata

xsodata-Services wurden für Datenbankabfragen genutzt, in denen keine komplexen Abfragen erstellt werden mussten. Damit liefern diese Services alle Daten, die in der jeweiligen View bzw. Table vorhanden sind.
Die weiteren für die App erstellten Services sind im Folgenden aufgelistet. Der Aufruf dieser Services funktioniert äquivalent zu dem oben vorgestellten Service.

- **project_status.xsodata** liefert die Werte der View PROJECT_STATUS, welche Informationen und Statuswerte zum Projekt enthält.
- **skill_psp_combination.xsodata** liefert die Werte der View SKILL_PSP_COMBINATION, welche die Zuordnung von Skill-Level, PSP-Nummer und der Mitarbeiter-ID enthält.
- **missingEmployeeSkill.xsodata** übergibt die Werte der View MISSING_EMPLOYEE_SKILL, welche den Namen und das PSP-Element von Mitarbeitern ausgibt, denen kein Skill-Level zugeordnet ist.
- **missingHourlyRates.xsodata** ist ein Service, der die Werte der View MISSING_HOURLY_RATES liefert, zum Ermitteln der Skill-Level eines PSP-Elements, denen kein Stundensatz zugeordnet ist.
- **missingProject.xsodata** gibt die Werte der View MISSING_PROJECT aus. Diesen Projekten wurden keine Statuswerte und Planwerte zugeordnet.
- **missingPsp.xsodata** liefert die Werte der View MISSING_PSP, also PSP-Nummern welchen keine Statuswerte bzw. Planwerte zugeordnet wurden.

Eine weitere Möglichkeit Daten aus der Datenbank auszulesen wird im UI durch xsjs-Services realisiert. Durch diese Services ist es möglich eine Logik innerhalb der Services abzubilden und auf Prozeduren zuzugreifen.
Aufgerufen wird diese Art von mit einer URL. Der Service getProjectStatus.xsjs liefert bei Übergabe der Projektnummer und des Datums (Stand der Datenbank) projektspezifische Informationen:

http://192.168.252.42:8002/abat/demosystem/projektcontrolling/controlling/services/getProjectStatus.xsjs?PNR=1116&DATE=31.12.2016

Zur Übergabe von Parametern in xsjs-Services, werden diese in der URL angegeben. In der Beispiel-URL oben ist die Projektnummer (PNR) „1116“ und das Datum, welches den Stand der Datenbank repräsentiert, (DATE) „31.12.2016“. Auf diese Weise können beliebig viele Parameter über die URL an den Service übergeben werden, indem sie durch ein „&“ voneinander getrennt werden.
Zunächst wird im Service die übergebene Projektnummer und das Datum als Request ausgelesen und in Variablen abgespeichert. Des Weiteren wird eine Fehlerbehandlung vorgenommen, falls die Projektnummer als „undefined“ definiert ist (vgl. Listing 6.9).

```
1  var pNumber = $.request.parameters.get("PNR");
2  var date = $.request.parameters.get("DATE");
3  var body = "error";
4  var data ={
5  "values":[]
6  };
7
8  // bei keiner Angabe wird "Invalid Key !!!" ausgegeben
9
10 body = pNumber;
11 if(pNumber === undefined){
12
13   $.response.setBody( "Invalid key !!!");
14 }
```

Listing 6.9: Projektcontrolling: getProjectStatus.xsjs: Service mit Procedure-Call

Daraufhin wird eine Connection zur Datenbank hergestellt und der Aufruf der Procedure realisiert. Hierbei werden alle von der Procedure benötigten Parameter übergeben. Die Anzahl der „?“ im Procedure-Call muss mit der Anzahl der übergebenen und der zurückgegeben Parameter übereinstimmen. Die ersten beiden „?“ sind durch die Variablen „pNumber“ (Projektnummer) und „date“ (Stand der Datenbank) vergeben, das letzte durch den Rückgabewert der Procedure. Nachdem die Procedure definiert wurde, erfolgt die Ausführung des calls. Außerdem wird das Result Set in eine Variable gespeichert (vgl. Listing 6.10 Zeile 18).
Daraufhin wird das Result Set solange durchlaufen, wie Zeilen durch die Procedure zurückgegeben wurden. Die entsprechenden Daten werden in die Variable „body“ gespeichert (vgl. Listing 6.10 Zeile 40). Fehler werden entsprechend abgefangen, falls diese beim Procedure-Aufruf auftreten (vgl. Listing 6.10 Zeile 43 ff.).
Das Result des Services liefert im JSON gespeicherten Werte als „text“ (vgl. Listing 6.10 Zeile 48 f.).

```
1  else{
2
3    $.response.contentType = "text/plain";
4    $.response.setBody(pNumber);
5    try {
```

```
6
7  //Connection zur Datenbank
8
9      var conn = $.db.getConnection();
10
11     var query = 'call "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling
       .controlling.procedures::GET_PROJECT_STATUS"(?,?,?)';
12     var cst = conn.prepareCall(query);
13     cst.setString(1, pNumber);
14     cst.setString(2, date);
15     var rs = cst.execute();
16     conn.commit();
17     rs = cst.getResultSet();
18
19 // JSON kreieren
20
21     while(rs.next()){
22       data.values.push(
23       {
24         "PNR":rs.getNString(1),
25         "DESCRIPTION":rs.getNString(2),
26         "STATUS_BUDGET":rs.getString(3),
27         "STATUS_HOURS":rs.getNString(4),
28         "PLANNED_BUDGET":rs.getNString(5),
29         "PLANNED_HOURS":rs.getNString(6),
30         "BUDGET_PERCENT":rs.getNString(7),
31         "HOURS_PERCENT":rs.getNString(8),
32         "PROJECT_LIMIT_YELLOW":rs.getNString(9),
33         "PROJECT_LIMIT_RED":rs.getNString(10),
34         "FIXED_VALUES":rs.getNString(12),
35         "SCORE_VALUE":rs.getNString(13)
36       }
37       );
38
39     }
40     body = JSON.stringify(data);
41     conn.close();
42
43   } catch (e) {
44     body = e.stack + "\nName:"+ e.name+"\nMsg" + e.message;
45     $.response.status = $.net.http.BAD_REQUEST;
46   }
47 }
48 $.response.contentType = "text/plain";
49 $.response.setBody(body);
```

Listing 6.10: Projektcontrolling: getProjectStatus.xsjs: Service mit Procedure-Call Query definieren

Die oben beschriebene Vorgehensweise ist für die xsjs-Services zur Datenbankabfrage, aber auch für die update-, delete und insert-Services ähnlich. Daher wird im Folgenden lediglich auf relevante Abweichungen des oben aufgeführten Services eingegangen.

Weitere Services zur Datenbankabfrage sind im Folgenden aufgelistet:

- **getClosedEntry.xsjs** ist ein Service, zum Liefern der gesperrten Einträge für jedes Projekt, damit diese in der Projekt-Ansicht dargestellt werden können. Dabei werden Einträge für jede im Projekt enthaltene PSP-Nummer ausgegeben.
- **getPlanValues.xsjs** liefert die Planwerte einer Woche auf Basis der übergebenen Parameter.
- **getProjectStatus.xsjs** übergibt die zum Projekt gehörenden Informationen und Statuswerte, welche in der Projekt-Ansicht dargestellt werden.
- **getPspWeekInformation.xsjs** übergibt für jede Woche im PSP-Element die gesamten geleisteten Stunden und das aufgewendete Budget.
- **getDetailPspWeekInformation.xsjs** ist ein Service, der für eine bestimmte Woche eines PSP-Elements die Arbeitszeitdaten ausgibt.
- **getPspStatusDetail.xsjs** liefert die Statuswerte eines bestimmten PSP-Elements. Dazu wird das entsprechende PSP-Element, das näher ausgewertet wird, übergeben.
- **getWorkLast.xsjs** übergibt die Mitarbeiter, welche bis zu einem übergebenen Datum noch keine Stundenzettel abgegeben haben.

Um Werte aus der Datenbank auslesen zu können, welche keine komplexeren Abfragen bedürfen, wurden xsjs-Services verwendet, welche per SQL-Befehl direkt auf die Datenbank zugreifen. Anders als bei Procedure-Calls wurden Views und Tabellen daher direkt angesprochen. Auch diese Services liefern ein JSON und sind mittels einer URL aufrufbar. Die Services unterscheiden sich lediglich im Aufruf zu den Procedure-Call-Services (vgl. Listing 6.11 Zeile 2).

```
1    var conn = $.db.getConnection();
2  var query = 'select * from "PROJECT_SCHEMA"."abat.demosystem.
     projektcontrolling.controlling.data::PROJECT" order by PROJECTNUMBER
     ASC';
3  var stmt = conn.prepareStatement(query);
4  stmt.execute();
5  var rs = stmt.getResultSet();
6
```

Listing 6.11: Projektcontrolling: getCurrentProjectStatus.xsjs Service mit SQL-Aufruf

Der Service „getCurrentProjectStatus.xsjs“ ruft alle Projektnummern mit den jeweiligen Statuswerten auf. Dabei ruft er die Tabelle „PROJECT“ auf und sortiert die Projektnummern in aufsteigender Reihenfolge. Der Aufbau des JSONs und die Ausgabe der Werte erfolgt äquivalent zu den Services mit Procedure-Call.

- **getSuggestionValues** übergibt je nach Auswahl alle vorhandenen Skill-Level, Projekte und PSP-Elemente. Diese dienen dazu Vorschläge der Input-Felder zum erleichterten Eingeben der Werte zu realisieren.

- **getMaxEntryDatabase.xsjs** ist ein Service, der den aktuellen Stand der Datenbank ausliest, basierend auf der View „WORK_DATA“.

- **getCurrentPspStatus.xsjs** ist ein Service, der alle aktuellen Statuswerte zum PSP-Element ausgibt.

- **getPlanValuesData.xsjs** gibt alle aktuell vorhandenen wochenbasierten Planwerte aller Projekte aus.

- **getProjectData.xsjs** übergibt alle aktuellen Statuswerte zum Projekt.

- **getSkillPspMa.xsjs** liefert alle aktuellen Skill-Level mit dem jeweils zugeordneten PSP-Element und Mitarbeiter.

- **getSkillPspRate.xsjs** liefert zu jedem aktuellen Skill-Level das jeweils zugeordnete PSP-Element sowie den Stundensatz.

Insert Auch der Aufbau der Insert-Services ist mit dem der Procedure-Call-Funktionen des Datenbankaufrufes vergleichbar. Übergeben werden die Parameter, welche in die Datenbank gespeichert werden sollen, lediglich der Rückgabewert unterscheidet sich. In diesem wird gespeichert, ob die Übernahme der übergebenen Parameter in die Datenbank erfolgreich war. Die vorher ausgelesen Parameter (vgl. Listing 6.12 auf der nächsten Seite Zeile 5-9) werden entsprechend der in der Procedure festgelegten Reihenfolge für den Procedure-Call definiert. In diesem Service wird ein PSP-Element auf Wochenbasis geplant und die in der UI eingegeben Werte in die Datenbank geladen. Die Rückgabewerte werden in ein JSON-Format gebracht. Dies ermöglicht eine Erweiterbarkeit des Services, falls noch andere Fehlerbehandlungen in Zukunft realisiert werden sollten. Im vorliegenden Prototyp speichert es die Meldung „ROW_COUNT“, welche die Werte „0“ oder „1“ annehmen kann. Ein Datenbankeintrag war erfolgreich, sobald ROW_COUNT größer oder gleich 1 ist. Bei einem Wert von 0 war die Übernahme der Daten somit nicht erfolgreich (vgl. Abbildung 6.10). ROW_COUNT kann in der UI übernommen werden, um Fehlermeldungen vorzunehmen.

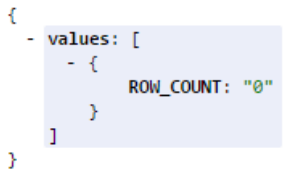


Abbildung 6.10.: Projektcontrolling: erfolglose Übernahme der Werte in die Datenbank

```
1    var conn = $.db.getConnection();
2    var query = 'call "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling
     .controlling.procedures::SET_PLAN_VALUES"(?,?,?,?,?,?)';
3    var cst = conn.prepareCall(query);
4
5    cst.setString(1, pspNumber);
6    cst.setString(2, level);
7    cst.setString(3, hours);
8    cst.setString(4, week);
9    cst.setString(5, year);
10   var rs = cst.execute();
11   conn.commit();
12   rs = cst.getResultSet();
13
14   while(rs.next()){
15     data.values.push({
16       "ROW_COUNT": rs.getString(1)
17     });
18   body = JSON.stringify(data);
19   }
20   conn.close();
21
```

Listing 6.12: Projektcontrolling: setPlanValues.xsjs

Der Aufbau aller Insert-Services weist eine ähnliche Struktur wie die oben beschriebene auf. Daher wird die Funktion dieser Art von Service nur kurz beschrieben.

- **setProjectData.xsjs** speichert die zu einem Projekt gehörenden Daten ab.
- Durch den Service **setPspData.xsjs** werden PSP-Daten in die Datenbank geladen.
- Mit **setSkillPspMa.xsjs** wird die Zuordnung von Skill-Level, PSP-Element und Mitarbeiter vorgenommen.
- **setSkillPspRate.xsjs** speichert die Zuordnung von Skill-Level, PSP-Element und Stundensatz.

Update/Delete Auch die Services zum Ändern und Löschen der Daten haben einen ähnlichen Aufbau wie die Insert-Services:

Update-Services

- **updateCloseDate.xsjs** setzt das Sperrdatum eines PSP-Elements durch Aufruf der Prozedur CLOSE_DATE um.
- **updateClosePsp.xsjs** ändert den Sperrzustand eines PSP-Elements auf „gesperrt“ bzw. „nicht gesperrt“ durch Aufruf der Prozedur CLOSE_PSP.

- **updatePlanValues.xsjs** ändert die wochenbasierten Planwerte durch Aufruf der Prozedur UPDATE_PLAN_VALUES.

- **updateProjectData.xsjs** ändert die projektspezifischen Daten durch Aufruf der Prozedur UPDATE_PROJECT_DATA.

- **updatePspData.xsjs** ändert die PSP-spezifischen Daten durch Aufruf der Prozedur UPDATE_PSP_DATA.

- **updateScoreValues.xsjs** ändert den Zustand, ob die Statusanzeige in der GUI nach Stunden oder nach Budget erfolgen soll durch Aufruf der Prozedur UPDATE_SCORE_ VALUES.

- **updateSkillPspMa.xsjs** ändert die Zuordnung von Skill-Level, PSP-Nummer und Mitarbeiter durch Aufruf der Prozedur UPDATE_SKILL_PSP_MA.

- **updateSkillPspRate.xsjs** ändert die Zuordnung von Skill-Level, PSP-Nummer und Stundensatz durch Aufruf der Prozedur UPDATE_SKILL_PSP_RATE.

Delete-Services

- **deletePlanValues.xsjs** löscht alle wochenbasierten Planwerte durch Aufruf der Prozedur DELETE_PLAN_VALUES.

- **deleteProjectData.xsjs** löscht alle Projektdaten inklusive aller PSP-Daten und Zuordnungen von Stundensätzen und Skill-Leveln durch Aufruf der Prozedur DELETE_PROJECT_DATA.

- **deletePspData.xsjs** löscht alle PSP-Daten, inklusive aller Zuordnungen von Stundensätzen und Skill-Leveln durch Aufruf der Prozedur DELETE_PSP_DATA.

- **deleteSkillPspMa.xsjs** löscht die Zuordnung von Skill-Level, PSP-Element und Mitarbeiter durch Aufruf der Prozedur DELETE_SKILL_PSP_MA.

- **deleteSkillPspRate.xsjs** löscht alle Zuordnungen von Skill-Level, PSP-Element und Stundensatz durch Aufruf der Prozedur DELETE_SKILL_PSP_RATE.

Export Ein weiterer Service zur Datenbankabfrage, welcher nicht zum Bereitstellen der Daten, sondern zum Exportieren der Daten als CSV-Datei dient, ist der Service „csv_export“.
Ähnlich wie bei den bereits vorgestellten Procedure-Call-Services werden zunächst Parameter ausgelesen, welche im späteren Verlauf an die Procedure übergeben werden. In diesem Fall sind das die Projektnummer und das Datum. Außerdem wird ein „body“ gesetzt, welcher später mit Werten befüllt wird (vgl. Listing 6.13 auf der nächsten Seite Zeile 4).

```
var pnr = $.request.parameters.get("PNR");
var date = $.request.parameters.get("DATE");
//Einen leeren body setzen der spaeter mit Werten befuellt wird
var body = '';
```

Listing 6.13: Projektcontrolling: CSV-Export Parameter

Nach dem schon bekannten Procedure-Aufruf, erfolgt neben dem Abrufen des Reseult Sets (vgl. Listing 6.14 Zeile 17), auch das Abrufen der Metadaten (vgl. Listing 6.14 Zeile 21) und der Anzahl der Spalten (Zeile 24), welche im weiteren Verlauf des Services benötigt werden.

```
try {
  //Die Datenbankverbindung herstellen
  var conn = $.db.getConnection();

  //Die aufzurufende Procedure
  var query = 'call "PROJECT_SCHEMA"."abat.demosystem.projektcontrolling.
    controlling.procedures::EXCEL_EXPORT"(?,?,?)';
  var cst = conn.prepareCall(query);

  //Der Procedure die oben definierten Parameter uebergeben
  cst.setString(1, pnr);
  cst.setString(2, date);

  //Die Procedure ausfuehren
  var rs = cst.execute();
  conn.commit();

  //Das ResultSet der Procedure abrufen
  rs = cst.getResultSet();

  //die MetaDaten des ResultSets abrufen
  var rsmeta = rs.getMetaData();

  //Die Anzahl der Spalten abfragen
  var numberOfColumns = rsmeta.getColumnCount();

  //Die erste Spalte ermitteln
  body += rsmeta.getColumnName(1);

  //Eine Variable zum Hochzaehlen im Loop
  var i;

```

Listing 6.14: Projektcontrolling: CSV-Export Procedure-Call

Nun werden die einzelnen Spalten des Result Sets in einer Schleife ermittelt, um die Spaltenname zu bestimmen. Diese werden daraufhin im „body“ gespeichert (vgl. Listing 6.15 Zeile 7-10). Es folgt die Bestimmung der Werte, welche ebenfalls in einer Schleife durchlaufen und im „body“ gespeichert werden.

```
  //Die erste Spalte ermitteln
  body += rsmeta.getColumnName(1);

  //Eine Variable zum Hochzaehlen im Loop
  var i;

  //Alle weiteren Spalten durchlaufen
  for (i = 2; i < numberOfColumns + 1; i++) {
    body += "\t" + rsmeta.getColumnName(i);
  }

  //Zeilenumbruch einfuegen
  body += "\n";


  //Alle Eintraege dieses Sets durchlaufen
  while(rs.next()) {
    //Die erste Spalte ermitteln
    body += rs.getNString(1);

    //Alle weiteren Spalten durchlaufen
    //Mit Loop geloest um einfache Erweiterbarkeit zu ermoeglichen (neue
     Spalte = kein Problem)
    for (i = 2; i < numberOfColumns + 1; i++) {
      body += "\t" + rs.getNString(i);
    }

    //Zeilenumbruch einfuegen
    body += "\n";
  }

  //Connection schliessen
  conn.close();

} catch (e) {
  //Fehler abfangen
    $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
    $.response.setBody(e.message);
}
```

Listing 6.15: Projektcontrolling: CSV-Export Loop

Abschließend erfolgt die Antwort des Services mit den entsprechenden Werten. Der Content Type wird dabei auf „csv“ gesetzt und der Dateiname nach der Projektnummer und dem Datum benannt (vgl. Listing 6.16).

```
  //setzen des Bodys
$.response.setBody(body);
  $.response.contentType = 'text/csv; charset=UTF-8';

```

```
5  //Header (inkl. Dateinamen) festlegen
6  $.response.headers.set('Content-Disposition',
7      'attachment; filename=Export_'+pnr+'_'+date+'.csv');
8
9  //Status ok
10 $.response.status = $.net.http.OK;
11
```

Listing 6.16: Projektcontrolling: CSV-Export Antwort des Services

6.3.3. ABAP-Applikation

In diesem Abschnitt wird auf die ABAP-Applikation eingegangen, welche dazu genutzt wird Arbeitszeit- und Mitarbeiterdaten aus dem ERP-System auf das HANA-System zu überführen. Hierzu war es zunächst erforderlich zwei Tabellen mit Hilfe der SAP GUI anzulegen, wie in Abbildung 6.11 dargestellt. Dies konnte mit der Transaktion SE11 umgesetzt werden. Es sind zwei Tabellen erforderlich, da der Funktionsbaustein Arbeitszeitinformationen sowie Personaldaten bereitstellt, die in separaten Tabellen gesichert werden müssen.

Transp.Tabelle ZDOHA_PC aktiv
Kurzbeschreibung DoHA Daten für Szenario 2 (Projectcontrolling)
Eigenschaften | Auslieferung und Pflege | Felder | Eingabehilfe/-prüfung | Währungs-/Mengenfelder
Suchhilfe | Eingebauter Typ | 1 / 12

Feld	Key	Ini...	Datenelement	Datentyp	Länge	DezS...	Kurzbeschreibung
PERNR	☑	☑	PERSNO	NUMC	8	0	Personalnummer
CATSHOURS	☑	☑	CATSHOURS	QUAN	4	2	Stunden
WORKDATE	☑	☑	CATSDATE	DATS	8	0	Datum
RPROJ	☑	☑	PS_POSID	CHAR	24	0	Projektstrukturplanelement (PSP-Element)
LOADDATE	☑	☑	ZLOADDATE	DATS	8	0	DoHA Ladedatum der Projektdaten
RPROJ_EXTERN	☐	☐	PS_POSID	CHAR	24	0	Projektstrukturplanelement (PSP-Element)
WEEK	☐	☐	ZWEEK	NUMC	2	0	DoHA S2 Wochennummer
FAKT	☐	☐	ZFAKT	CHAR	1	0	DoHA S2 PSP fakturierbar oder nicht fakturierbar
PRNR	☐	☐	ZPRNR	CHAR	8	0	DoHA Projektnummer zu PSP-Element
LTXA	☐	☐	LTXA1	CHAR	40	0	Kurztext Vorgang
BEGUZ	☐	☐	BEGUZ	TIMS	6	0	Beginnuhrzeit
ENDUZ	☐	☐	ENDUZ	TIMS	6	0	Endeuhrzeit

Transp.Tabelle ZDOHA_PC_MA aktiv
Kurzbeschreibung [DoHA] Mitarbeiterdaten für Szenario 2
Eigenschaften | Auslieferung und Pflege | Felder | Eingabehilfe/-prüfung | Währungs-/Mengenfelder
Suchhilfe | Eingebauter Typ | 1 / 4

Feld	Key	Ini...	Datenelement	Datentyp	Länge	DezS...	Kurzbeschreibung
PERNR	☑	☑	PERSNO	NUMC	8	0	Personalnummer
INITS	☐	☐	INITS	CHAR	10	0	Initialen
NACHN	☐	☐	NACHN	CHAR	25	0	Nachname
VORNA	☐	☐	VORNA	CHAR	25	0	Vorname

Abbildung 6.11.: Projektcontrolling: erstellte Tabellen mittels SAP GUI

Die Datentypen und Spalten der Tabellen leiten sich aus dem gegebenen Funktionsbaustein ab, sodass es nicht zu Kompatibilitätsproblemen kommt. Die beiden erstellten

Tabellen lassen sich im HANA-Studio mittels SQL-Befehlen ansprechen, sodass die Daten weiter ausgewertet werden können.
Zusätzlich zu den Daten des Funktionsbausteins war es erforderlich, bei den Arbeitszeitdaten das Ladedatum als Primärschlüsselattribut abzuspeichern.
Da den Datensätzen ein eindeutiges Schlüsselattribut fehlt, sind Änderungen nur schwer zu identifizieren. Um dieses Problem zu lösen und um einen Historienvergleich zu ermöglichen, werden täglich alle vorhanden Arbeitszeitdaten von 2016 und 2017 abgerufen und mit dem Ladedatum gespeichert. So entstehen verschiedene Zeitscheiben, was ermöglicht Datenstände von anderen Zeitpunkten darzustellen und somit Veränderungen nachzuvollziehen.
Nachdem die Tabellen verfügbar waren, musste ein Programm geschrieben werden, das den Funktionsbaustein aufruft und die Daten in die Tabellen schreibt.
Ein Ausschnitt der Implementierung der ABAP-Applikation ist in Listing 6.17 dargestellt.

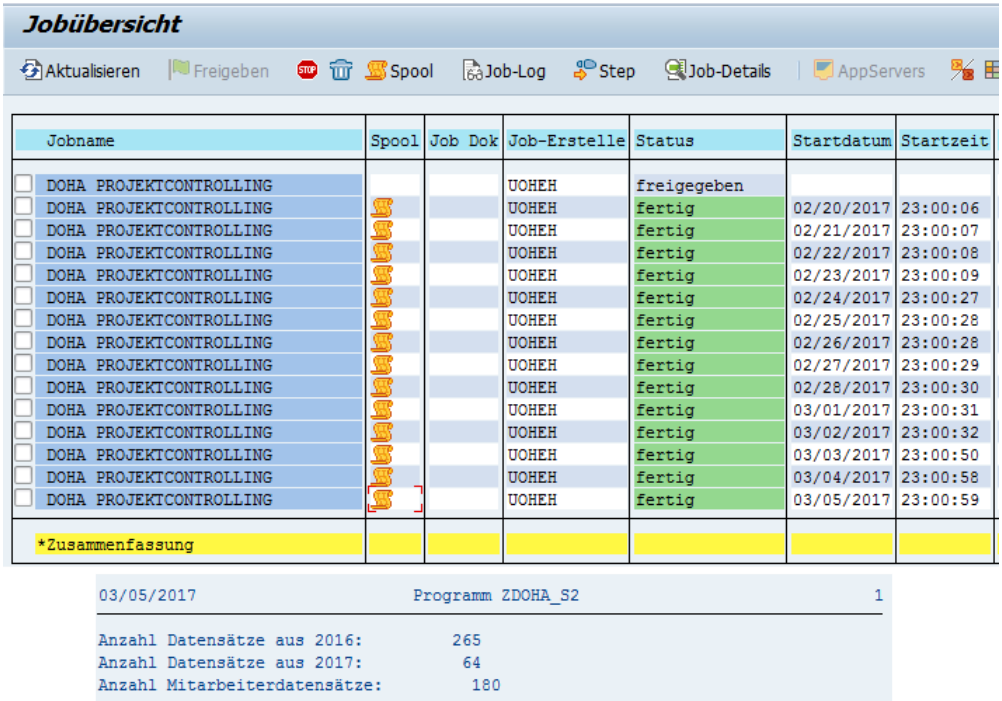
```
1  REPORT ZDOHA_S2.
2
3  TABLES: ZDOHA_PC,
4  ZDOHA_PC_MA.
5
6  DATA: iv_jahr       type n LENGTH 4,
7  iv_rfc_desti TYPE rfcdest.
8
9  DATA: ltr_ersda TYPE zabat_t_range.
10
11 DATA: ltr_filter TYPE RANGE OF sy-uname. " intialien
12
13 DATA: lt_catsdb    TYPE zabat_tt_catsdb,
14 lt_pers_data TYPE zabat_tt_pa0002,
15 PSPNR(60)     TYPE c,
16 p1(20)        TYPE c VALUE 'def',
17 PRNR(20)      TYPE c VALUE 'def',
18 p3(20)        TYPE c VALUE 'def',
19 p4(20)        TYPE c VALUE 'def',
20 del(1)        TYPE c VALUE '-',
21 counter       TYPE i VALUE 0.
22
23 DATA: wa_ZDOHA_PC LIKE ZDOHA_PC.
24
25 FIELD-SYMBOLS: <fs_cats>  TYPE zabat_s_catsdb,
26
27 <fs_pa002> TYPE zabat_s_pa0002.
28
29 iv_rfc_desti = 'E66'.
30 iv_jahr = 2016.
31
32 "Jahr 2016
33 APPEND INITIAL LINE TO ltr_ersda ASSIGNING FIELD-SYMBOL(<ls_ersda>).
34 <ls_ersda>-low =  '01.01.2016'.
35 <ls_ersda>-high = '31.12.2016'.
```

```
36 <ls_ersda>-option = 'BT'.
37 <ls_ersda>-sign = 'I'.
38
39 CALL FUNCTION 'Z_ABAT_AUSLL_RFC' DESTINATION iv_rfc_Desti
40 EXPORTING
41 iv_jahr               = iv_jahr
42 itr_filter            = ltr_filter
43 itr_ersda             = ltr_ersda
44 TABLES
45 et_catsdb             = lt_catsdb
46 et_pa0002             = lt_pers_data
47 EXCEPTIONS
48 system_failure        = 1
49 communication_failure = 2
50 OTHERS                = 3.
51
52 LOOP AT lt_catsdb ASSIGNING <fs_cats>.
53
54 wa_ZDOHA_PC-PERNR          = <fs_cats>-PERNR.
55 wa_ZDOHA_PC-CATSHOURS      = <fs_cats>-CATSHOURS.
56 wa_ZDOHA_PC-WORKDATE       = <fs_cats>-WORKDATE.
57 wa_ZDOHA_PC-RPROJ          = <fs_cats>-RPROJ.
58 wa_ZDOHA_PC-RPROJ_EXTERN   = <fs_cats>-RPROJ_EXTERN.
59 wa_ZDOHA_PC-WEEK           = <fs_cats>-WEEK.
60 wa_ZDOHA_PC-FAKT           = <fs_cats>-FAKT.
61 PSPNR                     = wa_ZDOHA_PC-RPROJ_EXTERN.
62 SPLIT PSPNR AT del INTO p1 PRNR p3 p4.
63 wa_ZDOHA_PC-PRNR          = PRNR.
64 wa_ZDOHA_PC-LTXA          = <fs_cats>-LTXA1.
65 wa_ZDOHA_PC-BEGUZ         = <fs_cats>-BEGUZ.
66 wa_ZDOHA_PC-ENDUZ         = <fs_cats>-ENDUZ.
67 wa_ZDOHA_PC-LOADDATE      = sy-datum.
68
69 Counter = Counter + 1.
70
71 INSERT ZDOHA_PC FROM wa_ZDOHA_PC.
72
73 ENDLOOP.
74
75 IF Counter > '0'.
76 WRITE: / 'Anzahl Datensaetze aus 2016: ', Counter.
77 NEW-LINE.
78 ELSE.
79 MESSAGE 'Keine Datensaetze aus 2016 vorhanden!' TYPE 'X'.
80 ENDIF.
81 Counter = 0.
```

Listing 6.17: Projektcontrolling: Ausschnitt Implementierung ABAP-Programm

Der Quellcode zeigt, wie der Funktionsbaustein aufgerufen wird und wie die Arbeitszeitdaten von 2016 in die erstellte Tabelle „ZDOHA_PC“ geschrieben werden. In Zeile 34 - 37 werden die Filter gesetzt, mit denen der Funktionsbaustein aufgerufen wird.

Dies umfasst zwei Datumsangaben, zwischen denen die Daten abgerufen werden. Die Variable „option“ mit dem Wert „BT“ steht für „between“ und bewirkt, dass die Daten zwischen den Datumsangaben abgerufen werden. Die Variable „sign“ mit dem Wert „I“ steht für „include“ und bewirkt, dass die abgerufenen Daten nicht ausgeschlossen werden. In Zeile 39 wird der Funktionsbaustein „Z_ABAT_AUSLL_RFC“ mit den vorher gesetzten Filtern aufgerufen. In Zeile 52 wird nun so lange eine Schleife durchlaufen, wie Datensätze vorhanden sind. Die Daten werden dabei in die Working-Area „wa_ZDOHA_PC“ geschrieben. In Zeile 62 und 63 wird zudem noch aus der gegebenen PSP-Nummer die Projektnummer ermittelt, indem der String geteilt wird. So kann aus der PSP-Nummer „I-605“ die Projektnummer „605“ extrahiert werden. Hierzu wird die PSP-Nummer am Symbol „-“ geteilt. In Zeile 67 wird das aktuelle Systemdatum als Ladedatum gespeichert. Sind alle Datensätze in der Working-Area gesichert, wird in Zeile 71 ein Insert der Working-Area in die Datenbanktabelle vorgenommen. Abschließend wird ausgegeben, wie viele Datensätze übermittelt wurden, was im Spool als Monitoringtool eingesehen werden kann. Sollten keine Daten übermittelt worden sein bricht das Programm mit einem Fehler ab. Dieser Vorgang ist für das Laden der Daten aus 2017 und der Personaldaten identisch, lediglich die Filter wurden angepasst (z.B. Änderung der Datumsangaben für Arbeitsdaten aus 2017). Der komplette Quellcode der Applikation ist auf dem beigelegten Datenträger enthalten. Um die tägliche Versorgung der Datenbank zu gewährleisten, war es noch erforderlich einen Job einzuplanen, der täglich ausgeführt wird. Dies kann mittels der Transaktion SM36 und dem Job Wizard konfiguriert werden. Der Job ist so geplant, dass das Programm täglich um 23:00 Uhr startet und die Daten überführt.



Jobübersicht

Aktualisieren | Freigeben | Spool | Job-Log | Step | Job-Details | AppServers

Jobname	Spool	Job Dok	Job-Erstelle	Status	Startdatum	Startzeit
DOHA PROJEKTCONTROLLING			UOHEH	freigegeben		
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/20/2017	23:00:06
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/21/2017	23:00:07
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/22/2017	23:00:08
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/23/2017	23:00:09
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/24/2017	23:00:27
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/25/2017	23:00:28
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/26/2017	23:00:28
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/27/2017	23:00:29
DOHA PROJEKTCONTROLLING			UOHEH	fertig	02/28/2017	23:00:30
DOHA PROJEKTCONTROLLING			UOHEH	fertig	03/01/2017	23:00:31
DOHA PROJEKTCONTROLLING			UOHEH	fertig	03/02/2017	23:00:32
DOHA PROJEKTCONTROLLING			UOHEH	fertig	03/03/2017	23:00:50
DOHA PROJEKTCONTROLLING			UOHEH	fertig	03/04/2017	23:00:58
DOHA PROJEKTCONTROLLING			UOHEH	fertig	03/05/2017	23:00:59
*Zusammenfassung						

03/05/2017 Programm ZDOHA_S2 1

Anzahl Datensätze aus 2016: 265
Anzahl Datensätze aus 2017: 64
Anzahl Mitarbeiterdatensätze: 180

Abbildung 6.12.: Projektcontrolling: Jobübersicht und Spool Datenüberführung

Die gelaufenen Jobs können über die Transaktion SMX eingesehen werden. Ebenfalls kann hierbei auch der Status der Jobs und der Inhalt des jeweiligen Spools eingesehen werden. Abbildung 6.12 auf der vorherigen Seite zeigt die Jobübersicht und den Spool eines gelaufenen Jobs, der anzeigt wie viele Datensätze übertragen wurden. Zusätzlich zum Monitoring mittels der Spoolübersicht wurde auch eingestellt, dass der Spool des Jobs täglich per E-Mail versendet wird. Da es sich um ein Entwicklungssystem handelt, ist der E-Mail-Versand jedoch geblockt, weshalb die E-Mails manuell freigegeben werden müssten.

Weitere Ansätze zur Anbindung des Fremdsystems

Zur Datenversorgung des Projektcontrolling-Prototyps war es erforderlich, das HANA-System mit einem SAP ERP-System zu verbinden, welches die Arbeitszeitinformationen enthält. Nach der Verbindung sollen die Daten in regelmäßigen Abständen (z.B. täglich) über einen Funktionsbaustein abgerufen und in das HANA-System geladen werden. Die Anbindung des ERP-Systems war im Szenario problematisch, da viele der geplanten Methoden nicht erfolgreich waren, bzw. nicht umgesetzt werden konnten. Folgend sind einige weitere Anbindungskonzepte, neben dem oben erläuterten, kurz dargestellt.

- **SAP Landscape-Transformation Replication Server:** Der SAP Landscape Transformation Replication Server kann dazu genutzt werden, diverse Datenbanken mit SAP HANA zu verbinden, um Daten auf das neue System zu übertragen. Zusätzlich gibt es die Möglichkeit, die Daten triggerbasiert in Echtzeit zu replizieren, daher sind alle neuen Einträge im Quellsystem unmittelbar auf dem HANA-Server verfügbar und können in die Auswertung mit aufgenommen werden.

  Vorteile dieser Lösung sind, dass das Tool von SAP erstellt wurde und deshalb eine hohe Kompatibilität mit anderen SAP Produkten gewährleistet sein sollte. Ebenfalls sollte die Konfiguration einfacher sein als mit einer selbst programmierten Lösung. Zusätzlich werden Tools zum Monitoring bereitgestellt und das Tool ist SAP-Standard und erfordert damit weniger Wartungsaufwand als eine Eigenentwicklung.

  Nachteilig ist allerdings, dass das Tool gesondert lizenziert werden muss und deshalb zusätzliche Kosten anfallen, sofern SAP LT nicht bereits Bestandteil der vorhandenen HANA Lizenz ist. Eine Testversion reicht durch den begrenzten Zeitraum nicht aus. Beispiele für die Nutzung und Einrichtung von SAP LT können unter folgenden Links gefunden werden:

  - `https://www.youtube.com/watch?v=oFhTD1tnuR0`
  - `https://www.youtube.com/watch?v=AzVf2IdOvfE`

  Die Schritte zur Konfiguration von SAP LT sind in folgender Quelle dargestellt.

  - `http://tinyurl.com/SAPLTPrasentation` (Folie 20 ff.)

  Ebenfalls problematisch ist, dass eine Datenquelle genau repliziert wird. Im gegebenen Szenario muss jedoch ein Funktionsbaustein genutzt werden, welcher ebenfalls

Daten aus mehreren Quellen zusammenführt und interne Logik beinhaltet. SAP LT konnte deshalb nicht genutzt werden.

- **Dateibasierter Ansatz:** In der ersten Abgabe des Prototypen wurde nicht mit Daten eines Fremdsystems gearbeitet. Stattdessen wurden Testdaten einer CSV-Datei in das System geladen und ausgewertet. Es ist ebenfalls möglich mittels eines Loadjobs automatisch CSV-Dateien in das System zu laden. Die Herausforderung war hierbei jedoch, aus dem Quellsystem eine CSV-Datei zu exportieren und diese in einer Form bereitzustellen, die zum einen für das HANA-System abrufbar ist, weiterhin aber auch den Datenschutz von personenbezogenen Daten gewährleistet. Das Schaffen dieser Grundbedingungen wäre für die Projektphase zu zeitintensiv gewesen, weshalb von diesem Ansatz abgesehen wurde.

- **OData:** Ein weiterer Ansatz war das Ansprechen des Funktionsbausteins mittels eines OData-Services. Problematisch war hierbei, dass abat die Schnittstelle per OData hätte ansprechbar machen müssen, was mit einem hohen Entwicklungsaufwand verbunden gewesen wäre. Ebenfalls wäre diese Lösung weit weg vom SAP Standard und die Überführung von hohen Datenmengen per Webservice hätte problematisch werden können. Es wurde sich also gegen diese Lösung entschieden, zumal auch ungewiss war, ob die Anbindung mittels OData realisierbar gewesen wäre.

6.3.4. Grafische Oberfläche

Im Folgenden wird die grafische Oberfläche der Projektcontrolling-App vorgestellt. Entwickelt wurde die Applikation im HANA-Studio, welche in der Entwicklungsumgebung Eclipse läuft. Die webbasierte Entwicklungsumgebung SAP Web IDE wurde selten genutzt. Die grafischen Elemente befinden sich in den „.view.js-Klassen“ und die Models und Funktionen in den „.controller.js-Klassen“. Der Aufbau der verschiedenen Seiten ist ähnlich, weshalb nicht alle UI-Elemente und Funktionen beschrieben werden. Der Quelltext ist dokumentiert und kann für mehr Informationen zu allen Elementen der Implementierung herangezogen werden. Zur Kommunikation mit dem Anwender wurde die Benutzerschnittstelle (UI) mit der Model-View-Controller-Architektur geschrieben.

Der Anwendungsaufruf erfolgt durch die Ausführung der index.html-Datei im Browser, welcher durch das Betätigen des Run(F8)-Buttons auf der SAP HANA Web-based Development Workbench eingeleitet werden kann. Hierbei wird die folgende URL aufgerufen:

http://192.168.252.42:8002/abat/demosystem/projektcontrolling/controlling/ui/split/WebContent/index.html

Die Nutzung der Anwendung ist an eine aktive Internetanbindung gebunden.

Architektur der UI Die Applikation lässt sich in zwei wesentliche Bereiche unterteilen: die Dateneingabe und die Datenauswertung. In der Dateneingabe werden Projekte und alle dazugehörigen Daten eingepflegt und über die Schnittstelle kommende Daten ergänzt. Basierend auf den in der Dateneingabe vorgenommenen Einstellungen, befindet sich in der Datenauswertung ein Reporting der Daten. Die gesamte Applikation ist in einer Master-Detail-Ansicht dargestellt. Links befindet sich dabei die Masteransicht, in der unterschiedliche Navigationen vorgenommen bzw. Auswahlen getroffen werden können. Auf der rechten Seite befindet sich die Detail-Ansicht, in der die wesentlichen Informationen der UI dargestellt sind (vgl. Abbildung 6.13).

Abbildung 6.13.: Projektcontrolling: Master- und Detail-Ansichten

Die Navigation zu den einzelnen Sichten der Applikation erfolgt nicht ausschließlich über die in der Master-Ansicht getroffenen Auswahl. Es ist ebenfalls möglich zwischen einigen Detail-Sichten zu navigieren. Dies geschieht auf zwei unterschiedliche Arten: In der Datenauswertung ist eine Breadcrumb-Navigation implementiert, die es ermöglicht in einem Drill-Down nähere Informationen zu einem Projekt zu erhalten und zu der vorherigen Ansicht zurückzukehren. Mit Hilfe der Buttons „zurück“ und „weiter“ ist es in der Dateneingabe möglich zwischen den Eingabeoberflächen zu wechseln. Abbildung 6.14 auf der nächsten Seite) zeigt die Navigation zwischen den verschiedenen Detail- und Master-Pages und wie diese zusammenhängt.

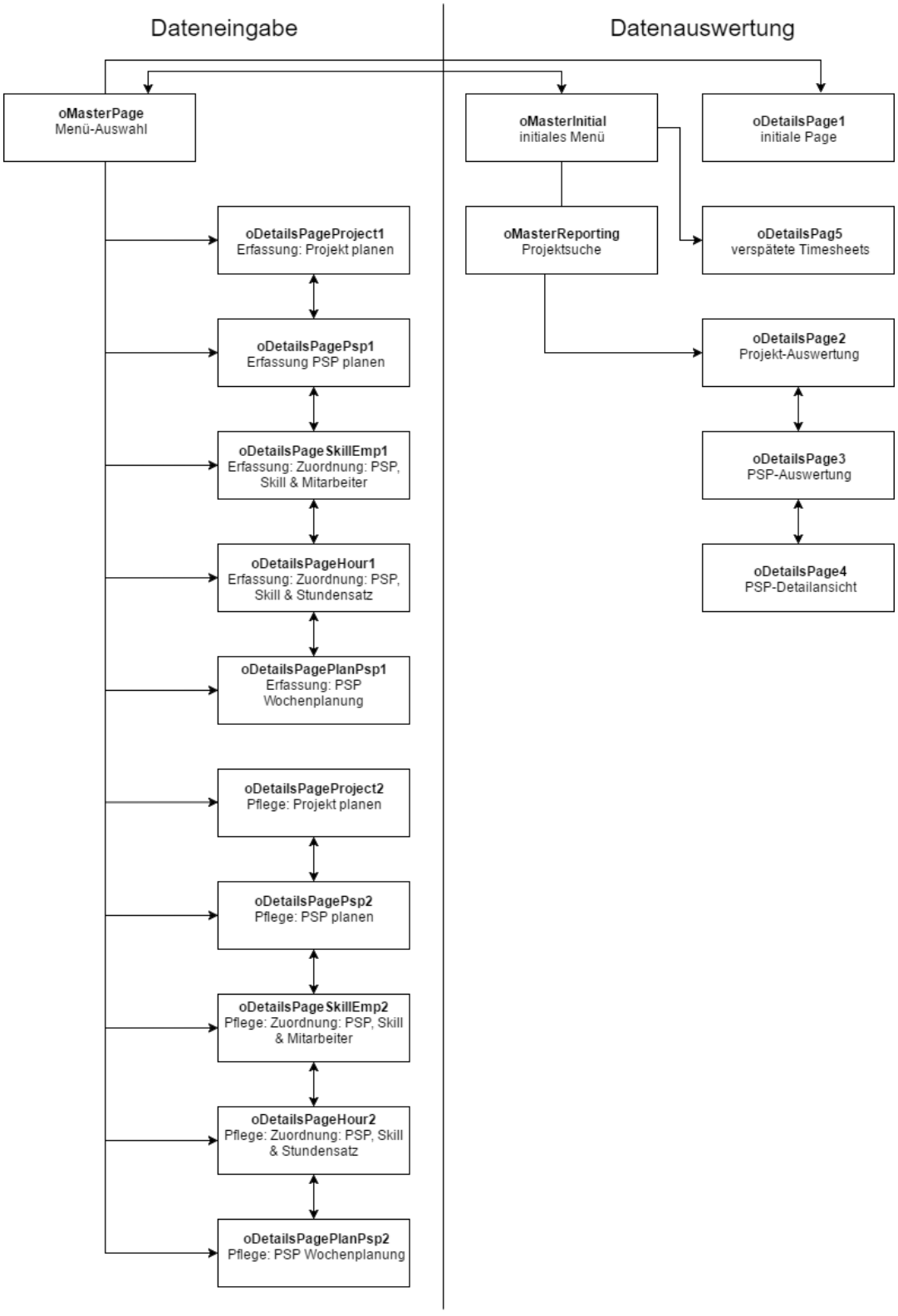


Abbildung 6.14.: Projektcontrolling: Struktur und Navigation der UI

Datenanbindung Zur Schnittstelle zwischen der Datenbank und der grafischen Oberfläche wurden Services, welche im gleichnamigen Abschnitt erläutert wurden, verwendet. Um die Daten für die Oberfläche vorzuhalten wurden JSON-Models (JSON-Datenmodelle) erzeugt, welche eine eindeutige ID besitzen. Durch ein Binding der Properties des Models ist es mit dieser ID möglich die Daten für die UI zu nutzen. Die Models werden jeweils in den beiden Oberflächen „Dateneingabe“ und „Datenauswertung“ in der onInit()-Funktion geladen.

AJAX-Aufrufe werden genutzt, um Daten aus der Datenbank in die Models zu laden. Die Funktionsweise von AJAX-Aufrufen wurde im Kapitel 5.3.4 auf Seite 143 erläutert.

6.3.5. UI Datenauswertung

Zur Auswertung werden die Daten, welche über die Schnittstelle kommen in einem Reporting dargestellt. Der Nutzer der Applikation erhält die Informationen über ein Projekt in anschaulichen Grafiken und Tabellen. In einem Drill-Down wird dem Anwender das Projekt dargestellt. Von der gröbsten Ansicht, welche die Daten des Gesamtprojektes darstellt bis hin zur Detailansicht, welche Arbeitszeitdaten enthält, navigiert der Nutzer durch ein ausgewähltes Projekt. Um eine hohe Übersichtlichkeit zu gewährleisten, wurde in der Entwicklung darauf geachtet, dass keine der Ebenen einen zu hohen Informationsgehalt enthält.

Die Navigation der Applikation ist in den Klassen der Datenauswertung angesiedelt. Die Elemente befinden sich in den Klassen „Intro.controller.js“ und „Intro.view.js“. Die Ordnerstruktur befindet sich unter: „abat/demosystem/projekt-controlling/controlling/ui /split“. Im Ordner „res“ befinden sich zudem alle externen Elemente, welche die App verwendet.

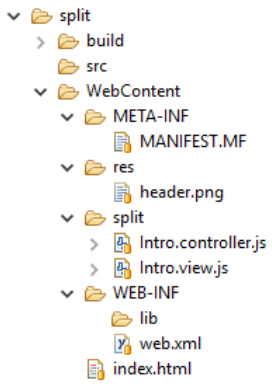


Abbildung 6.15.: Projektcontrolling: Ordnerstruktur Datenauswertung

Initial werden alle Models erstellt, welche später die Daten für die App vorhalten (vgl. Listing 6.18 auf der nächsten Seite). Die Models werden initial geladen, da sich der Inhalt entsprechend der vom Nutzer vorgenommenen Auswahl verändert. Auf diese Weise muss

bei einer Auswahl nicht jedes mal das Model neu erstellt werden, da auf bestehende Models zugegriffen wird.

Tabelle 6.9.: Projektcontrolling: Models Datenauswertung

Paket:	abat/demosystem/projektcontrolling/controlling/ui/split	
Datei:	intro.controller.js	
Model	**Service**	**Control**
projectTotalReportingModel	getProjectStatus	JSON-Model
modelpspweek	getPspWeekInformation	JSON-Model
modelpspdetail	getPspStatusDetail	JSON-Model
modelworklast	getWorkLast	JSON-Model
modelpspdate	getPspDetailWeekInformation	JSON-Model
modelclosedentry	getClosedEntry	JSON-Model
planvalues	getPlanValues	JSON-Model
pspdata	JSON-Model	JSON-Model

```
1   /**
2    * Models, welche die Daten fuer die Auswertung vorhalten und durch
    einen Service mit den entsprechenden Daten versorgt werden
3    */
4   var oModelProject = new sap.ui.model.json.JSONModel();
5   sap.ui.getCore().setModel(oModelProject, 'projectTotalReportingModel');
6
7   var oModelPspWeek = new sap.ui.model.json.JSONModel();
8   sap.ui.getCore().setModel(oModelPspWeek, 'modelpspweek');
9
10  var oModelPspDetail = new sap.ui.model.json.JSONModel();
11  sap.ui.getCore().setModel(oModelPspDetail, 'modelpspdetail');
12
13  var oModelWorkLast = new sap.ui.model.json.JSONModel();
14  sap.ui.getCore().setModel(oModelWorkLast, 'modelworklast');
15
16  var oModelPspDate = new sap.ui.model.json.JSONModel();
17  sap.ui.getCore().setModel(oModelPspDate, 'modelpspdate');
18
19  var oModelClosedEntry = new sap.ui.model.json.JSONModel();
20  sap.ui.getCore().setModel(oModelClosedEntry, 'modelclosedentry');
21
```

```
22      var oModelPlanValues = new sap.ui.model.json.JSONModel();
23      sap.ui.getCore().setModel(oModelPlanValues, 'planvalues');
24
25      var oModelPSP = new sap.ui.model.json.JSONModel("/abat/demosystem/
        projektcontrolling/controlling/services/getProjectData.xsjs?");
26      sap.ui.getCore().setModel(oModelPSP, 'pspdata');
```

Listing 6.18: Projektcontrolling: Datenmodelle Datenauswertung

Neben den Models wurden in der onInit()-Funktion der Datenauswertung Variablen deklariert, welche für die gesamte Klasse gelten und auf die mehrfach zugegriffen wird. Dies vermeidet Code-Redundanzen, da die Variablen in mehreren Funktionen verwendet werden können. Die verwendeten Variablen sind in Listing 6.19 auf der nächsten Seite aufgeführt und werden im Folgenden erläutert.

- **pattern:** ein Pattern, mit dem das Datum umformatiert wird
- **scoreValue:** Variable zum Deklarieren der Statusanzeige im PSP-Status nach Stunden („H“) oder nach Budget („B“)
- **searchContent:** Variable, die in der Ansicht „verspaetete Timesheets“ deklariert, ob nach „TOKEN“ (Kürzel) oder „SURNAME“ (Nachname des Mitarbeiters) gesucht werden soll. Initial ist „TOKEN“ voreingestellt.
- **app:** Variable mit Informationen über die App, welche verwendet wird, um zwischen den Ansichten zu navigieren
- **selectedProject:** das in der Master-Ansicht ausgewählte Projekt
- **projectBudgetPercent:** verbrauchtes Budget des ausgewählten Projekts in Prozent
- **projectHoursPercent:** aufgewendetes Stunden des ausgewählten Projekts in Prozent
- **projectLimitYellow:** die Grenze des gelben Limits des ausgewählten Projekts in Prozent
- **projectLimitRed:** die Grenze des roten Limits des ausgewählten PSP in Prozent
- **selectedPSP:** das ausgewählte PSP in der Projekt-Auswertung
- **selectedWeek:** verbrauchtes Budget des ausgewählten PSP-Elements in Prozent
- **pspBudgetPercent:** verbrauchtes Budget des ausgewählten PSP-Elements in Prozent
- **pspHoursPercent:** aufgewendete Stunden des ausgewählten PSP-Elements in Prozent

- **pspLimitYellow:** die Grenze des gelben Limits des ausgewählten PSP-Elements in Prozent
- **pspLimitRed:** die Grenze des roten Limits des ausgewählten PSP-Elements in Prozent
- **selectedYear:** das Jahr der in der PSP-Auswertung ausgewählten Woche
- **valueOfDatePicker:** das Datum, das im DatePicker ausgewählt wurde

```
1     this.pattern = /(\d{2})\.(\d{2})\.(\d{4})/;
2     this.scoreValue= "B";
3     this.searchContent ="TOKEN";
4     this.app = sap.ui.getCore().byId("splitApp1");
5     this.selectedProject;
6     this.projectBudgetPercent;
7     this.projectHoursPercent;
8     this.projectLimitYellow;
9     this.projectLimitRed;
10    this.selectedPSP;
11    this.pspBudgetPercent;
12    this.pspHoursPercent;
13    this.pspLimitYellow;
14    this.pspLimitRed;
15    this.selectedWeek;
16    this.selectedYear;
17    this.valueOfDatePicker;
```

Listing 6.19: Projektcontrolling: Variablen Datenauswertung

Des Weiteren wurde eine Variable erstellt, die das aktuellste Datum aus der Datenbank mit einem AJAX-Aufruf abfragt. Dies wird realisiert, indem der Service „getMaxEntryDatabase.xsjs“ aufgerufen wird. Dieser ruft mittels SQL-Befehls das aktuellste Datum der Datenbank ab. Die dazugehörige Funktion „setValueOfDatePicker()“ speichert den Wert in die oben deklarierte Variable „valueOfDatePicker“ (vgl. Listing 6.20).

```
1     /**
2      *
3      *Durch das Aufrufen des Services wird der neuste Stand der Datenbank
       abgefragt und an die Funktion setValueOfDatePicker uebergeben,
4      *welche wiederum die DatePicker in den Ansichten der Projekt-, PSP-
       und Wochenstatus aendert
5      */
6     var self = this
7     $.ajax({
8       url: "/abat/demosystem/projektcontrolling/controlling/services/
       getMaxEntryDatabase.xsjs?",
9       async:false,
10      type:"GET",
11      dataType:"json",
12      cache:"false",
13      success: function(response){
```

```
14          var date = new Date((response.values[0].ENTRY_DATE).replace(self.
    pattern,'\$3-$2-$1'));
15          self.setValueOfDatePicker(date);
16        },
17        complete: function(){
18
19        },
20        error:function(xhr,status,error){
21
22        }
23      });
```

Listing 6.20: Projektcontrolling: setValueOfDatePicker()

Master-Page - initialMaster Die initiale Master-Seite, also das „Hauptmenü“ der Applikation, beinhaltet eine Liste mit den drei Objekten: „Datenauswertung“, „verspätete Timesheets“ und „Dateneingabe“ (vgl. Abbildung 6.13 auf Seite 238 und Listing 6.21).

```
1     var oMasterInitial = new sap.m.Page("master1",{
2       title : "Menue",
3       icon: "sap-icon://action",
4       class : "sapUiStdPage",
5       content : [new sap.m.List({
6         items :
7           [
8            new sap.m.ObjectListItem({
9             title : "Datenauswertung",
10            type : "Navigation",
11            press : function() {
12              oController.slideToMaster("2");
13            },
14            icon: 'sap-icon://pie-chart',
15        }),
16        new sap.m.ObjectListItem({
17          title : "verspaetete Timesheets",
18          type : "Navigation",
19          press :  function() {
20            oController.slideToDetails("5");
21          },
22          icon: 'sap-icon://timesheet'
23        }),
24          new sap.m.ObjectListItem({
25            title : "Dateneingabe",
26            type : "Navigation",
27            press : function() {
28              //Aufruf der Report Seite
29              sap.m.URLHelper.redirect('http://192.168.252.42:8002/abat/
    demosystem/projektcontrolling/controlling/ui/ui_data/ WebContent/index.
    html');
30            },
31            icon: 'sap-icon://table-chart'
32          })]
```

```
33          })]
34      });
```

Listing 6.21: Projektcontrolling: initiale Master Page oMasterInitial

Zur Navigation zwischen den Pages wurde der „type“ der ObjectListItems auf „Navigation“ gesetzt und es wurde eine Funktion implementiert, welche die entsprechende ID der Seite empfängt, um ein Wechsel der Seite zu veranlassen. Mit der Funktion toMaster() ist es möglich die aktuelle Master-Page auf eine andere zu wechseln. Um diese Funktion auszuführen muss die ID der jeweiligen Master-Page angegeben werden. Der Befehl „slide“ erzeugt weiterhin die Animation, dass die neue Seite von rechts ins Fenster gleitet. Derzeit mögliche Alternativen dieses Befehls sind „show“ und „fade“.

```
1   slideToMaster:function(master){
2
3     this.app.toMaster('master'+master,'slide');
4
5     },
```

Listing 6.22: Projektcontrolling: initiale Master Page slideToMaster

Master-Page - Reporting Wird in der initialen Master-Ansicht „Datenauswertung“ ausgewählt, gelangt der Nutzer auf die Master-Seite „oMasterReporting“ (vgl. Abbildung 6.16). Diese Seite ist für die Navigation zwischen den einzelnen Projekten zuständig und enthält grobe Informationen über den Status des Projektes.

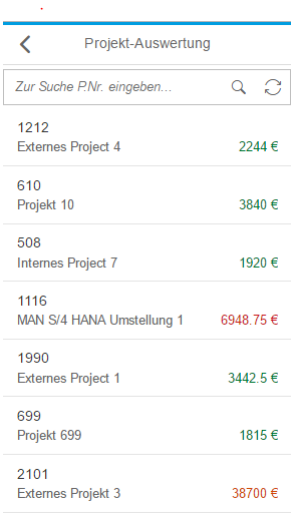


Abbildung 6.16.: Projektcontrolling: oMasterReporting

Die Master-Page enthält neben den Attributen, die ebenfalls in der Page „oMasterInitial“ verwendet wurden, einen Navigations-Button (vgl. Listing 6.23 auf der nächsten Seite Zeile 3), welcher die Navigation zur davor befindlichen Master-Seite veranlasst. Der Inhalt (content) der Seite beinhaltet eine Liste, welche alle verfügbaren Projektnummern

enthält („oListMasterReporting“) und ein Suchfeld („oSearchProjectnumber“), mit dem nach diesen Projektnummern gesucht werden kann (vgl. Listing 6.23 Zeile 7).

```
1     var oMasterReporting = new sap.m.Page("master2",{
2       title: "Projekt-Auswertung",
3       showNavButton: true,
4       navButtonPress: function() {
5         oController.slideToMaster("1");
6         },
7       content: [oSearchProjectnumber,oListMasterReporting]
8     });
9
```

Listing 6.23: Projektcontrolling: Reporting Master Page oMasterReporting

In dieser Liste ist es möglich die beinhalteten Objekte auszuwählen (vgl. Listing 6.24 Zeile 8). Da der ListMode auf SingleSelectMaster gestellt ist, ist immer nur ein Eintrag der Liste auswählbar. Bei einer entsprechenden Auswahl, wird die Funktion itemSelected() ausgeführt (vgl. Listing 6.24), welche den Wechsel der Detail-Seite entsprechend der getroffenen Auswahl veranlasst.

```
1     /**
2      * List, in der alle Projektnummern dargestellt werden. Durch eine
      Auswahl der entsprechenden
3      * Projektnummer, gelangt man zur Auswertung der Projektnummer. Wenn
      ein Element der List ausgewaehlt wird,
4      * wird die Funktion itemSelected() aufgerufen. Diese Funktion
      wiederum veranlasst das Wechseln der Detail-Page
5      * entsprechend der getaetigten Auswahl.
6      */
7     var oListMasterReporting = new sap.m.List("listMasterReporting",{
8       mode: sap.m.ListMode.SingleSelectMaster,
9       select: function() {
10        oController.itemSelected();
11      }
12    });
```

Listing 6.24: Projektcontrolling: Reporting Master Page oListMasterReporting

Um die Daten an die Liste zu binden, wird ein Template verwendet, welches auf das Model „pspdata“ zugreift (vgl. Listing 6.25). Die entsprechenden Werte werden an die Properties „title“, „description“ und „info“ gebunden. In „title“ sind die jeweiligen Projektnummern gebunden, „description“ enthält die Beschreibung des Projektes und „info“ das aktuell aufgewendete Budget in Euro. Die beiden Properties „info“ und „infostate“ enthalten zusätzlich einen Formatter.

```
1     /**
2      * Erstellen eines Templates fuer die List welches auf das model "
      pspdata" zugreift
3      * Der title ist hierbei die PSP-Nummer selbst, description beinhaltet
       die description (Kurztext)
4      * der PSP-Nummer und info enthaelt das Budget, welches nach dem
      Status markiert ist.
```

```
5       */
6
7
8       var oListMasterReportingTemplate = new sap.m.StandardListItem({
9         id: "sList",
10        title:  "{pspdata>PROJECTNUMBER}",
11        description: "{pspdata>DESCRIPTION}",
12        info:{path: 'pspdata>STATUS_BUDGET',
13                formatter: function(item){
14                return oController.getValueEuro(item);
15              }
16            },
17
18        infoState: {
19              path: 'pspdata>',
20                formatter: function(item){
21                return oController.getProjectInfoStateList(item);
22              }
23            }
24      });
```

Listing 6.25: Projektcontrolling: Reporting Master Page oListMasterReportingTemplate

Formatter sind Funktionen, die verwendet werden, um bestehende Werte eines Models in einer anderen Weise darstellen zu können. Es gibt Formatter, welche aus der Bibliothek entnommen werden können (beispielsweise zum Umwandeln eines String-Datentyps in float) und Custom-Formatter, die benutzerdefinierte Umformatierungen ermöglichen. Die in „oListMasterReporting“ verwendeten Formatter sind Custom-Formatter.

Durch den Formatter „getValueEuro()“ wird an den übergeben Wert die Einheit Euro gehängt (vgl. Listing 6.26). Durch die Platzierung des Formatters als Funktion im Controller kann dieser mehrfach in der View verwendet werden.

```
1     getValueEuro: function(item){
2
3     return item + " Euro";
4   },
5
```

Listing 6.26: Projektcontrolling: Formatter getValueEuro()

Der Formatter „getProjectInfoStateList()“ gibt den Status des Budgets an (vgl. Listing 6.27 auf der nächsten Seite). Der Status kennzeichnet das Budget eines Projektes. Somit erhält das Budget die Farbe grün, falls das Budget unter den Schwellenwerten liegt, orange, bei Erreichen des gelben Schwellenwertes und rot, wenn der rote Schwellenwert überschritten ist. Für die Darstellung dieser drei Farben werden die in SAPUI5 vorgegebenen Standardwerte für die Kennzeichnung des „infoStates“ verwendet. Die grüne Darstellung erfolgt mit dem Wert „Success“ (vgl. Listing 6.27 auf der nächsten Seite Zeile 24), die orange Darstellung mit „Warning“ (vgl. Listing 6.27 auf der nächsten Seite Zeile 27) und die rote Darstellung mit „Error“ (vgl. Listing 6.27 auf der nächsten Seite

Zeile 30). Falls kein Schwellenwert vorhanden ist, erfolgt eine Darstellung im Default-Wert „Neutral“ (vgl. Listing 6.27 Zeile 32), welcher schwarz dargestellt wird. Aufgrund der Tatsache, dass zum Zeitpunkt der Entwicklung des Prototyps kein gelber Wert zur Darstellung des „infoStates“ vorliegt, erfolgt die Darstellung des gelben Schwellenwertes in orange.

```
1  getProjectInfoStateList: function(item){
2
3      if (item === undefined){
4        return;
5      }else{
6
7    budgetPercent = parseInt(item.BUDGET_PERCENT);
8    limitYellow = parseInt(item.PROJECT_LIMIT_YELLOW);
9    limitRed = parseInt(item.PROJECT_LIMIT_RED);
10
11
12   if(this.scoreValue=="H"){
13     caseValue=hoursPercent;
14
15   }else{
16
17     caseValue=budgetPercent;
18   }
19
20 //Vergleich der Variablen und deklarieren von state entsprechend des
    Status
21 var state = "";
22
23   if (caseValue <= limitYellow){
24     state = "Success"
25   }
26   else if(caseValue <= limitRed && caseValue >  limitYellow){
27     state = "Warning"
28   }
29   else if(caseValue > limitRed){
30     state = "Error"
31   }else
32     state = "Neutral";
33   }
34     return state;},
```

Listing 6.27: Projektcontrolling: Formatter getProjectInfoStateList()

Da die App beliebig viele Projektnummern beinhalten kann und die Anzahl der Projektnummern daher sehr groß werden kann, wurde ein Suchfeld implementiert, mit dem der Nutzer nach konkreten Projektnummern filtern kann. Durch Eingabe eines Wertes in das Suchfeld wird nach der Projektnummer gesucht, welche den eingegeben Wert beinhaltet. Bei einer Eingabe von „10“ würden in der Liste beispielsweise die Projektnummern „1012“ und „2101“ auftauchen, da beide den eingegeben Wert beinhalten. Auf diese Weise ist es möglich auch Projekte zu finden, von denen der Nutzer nicht ganz

genau die zugehörige Nummer weiß. Dies wurde mit dem Event „liveChange“ und einer entsprechenden Funktion „filterProjectnumber()“ realisiert (vgl. Listing 6.28 Zeile 4).

```
1     var oSearchProjectnumber = new sap.m.SearchField({
2         placeholder: "Zur Suche P.Nr. eingeben...",
3         showRefreshButton: true,
4         liveChange : [oController.filterProjectnumber, oController],
5             });
```

Listing 6.28: Projektcontrolling: Reporting Master Page oSearchProjectnumber

Der Funktion „filterProjectnumber()“ wird das Objekt „oEvent“ übergeben, welches neben anderen Parametern auch die Information „newValue“ enthält. Das liveChange des SearchFields sorgt dafür, dass die Funktion immer dann ausgeführt wird, wenn Daten vom Nutzer in das Suchfeld eingegeben oder entfernt werden. „newValue“ besitzt also immer den Wert, der aktuell im Suchfeld enthalten ist. In die Variable „tpmla“ wird dieser Wert gespeichert und an einen Filter übergeben (vgl. Listing 6.29 Zeile 3). Der Filter enthält weiterhin die Information, welches Attribut der Tabelle er filtern soll. In diesem Fall ist das Attribut „PROJECTNUMBER“. Anschließend wird die Liste aus der View gelesen und die Funktion „filter()“ wird mit der deklarierten Filter-Variable angewendet (vgl. Listing 6.29 Zeile 11f.).

```
1       filterProjectnumber: function(oEvent) {
2
3               var tpmla = oEvent.getParameter("newValue");
4               var filters = new Array();
5
6               var oFilter = new sap.ui.model.Filter("PROJECTNUMBER", sap.
    ui.model.FilterOperator.Contains, tpmla);
7               filters.push(oFilter);
8
9               //List aus der View holen
10
11              this.oListMasterReporting = sap.ui.getCore().byId("
    listMasterReporting");
12              this.oListMasterReporting.getBinding("items").filter(filters
    );
13          },
```

Listing 6.29: Projektcontrolling: Reporting Master Page filterProjectnumber

Die Funktion „itemSelected()“ (vgl. Listing 6.30) ist dafür zuständig bei Auswahl einer Projektnummer aus der Master-Page „oMasterReporting“ die darauffolgende Detail-Page vorzubereiten.

```
1     /**
2      * Wurde eine Projektnummer in der Master-Ansicht ausgewaehlt, wird
    die Funtkion itemSelected aufgerufen
3      * Diese ruft wiederum die folgenden Funktionen auf:
4      * getPspModel() - Model, das die Daten zu jedem im Projekt
    befindlichen PSPs enthaelt und setzt die globalen Werte aus dem Model
5      * setValueLockedEntry() - zaehlt die Anzahl der gesperrten Elemente
    und zeigt diese beim IconTabFilter in der GUI an
```

```
6       * pushScoreValue() - Stellt den Wert nach dem gefiltert werden soll
    fest (nach Stunden bzw. nach Budget) fesr
7       * setProjectInfoState() - faerbt den IconTabFilter entsprechend der
    Schwellenwerte des Projektes und des scoreValues
8       */
9      itemSelected: function() {
10
11
12       this.getPspModel();
13       this.setValueLockedEntry();
14 //       var app = sap.ui.getCore().byId("splitApp1");
15       this.pushScoreValue();
16       this.app.toDetail('details2','slide')//Alternativen: show, fade,
    slide
17       this.setProjectInfoState();
18     },
```

Listing 6.30: Projektcontrolling: Reporting Master Page itemSelected()

Zunächst werden das Datenmodell und die nötigen Variablen, welche in der Detail-Page genutzt werden, erstellt. Dies ist die Aufgabe der Funktion „getPspModel()“.
Um das Model des ausgewählten Projekts mit den richtigen Daten zu füllen, wird dieses aus der Liste ausgelesen. Dazu wird die instance der Liste mit Hilfe der ID ausgelesen und mit der Funktion „getSelectedItem()“ wird das ausgewählte Item in die Variable „sItem“ gespeichert. Nun wird der Pfad zum Item ausgelesen, um mit diesem die Property zu bekommen, welche dann in die Variable „item“ gespeichert wird. In „item“ sind nun die Inhalte des Models, welche zu der ausgewählten Projektnummer gehören gespeichert. Jetzt können die entsprechenden Variablen deklariert werden, welche für die Funktionalität der Details-Page „Projekt-Auswertung“ notwendig sind.

Das Datum, welches in den DatePicker der Detail-Page eingetragen wurde, wird ausgelesen, um die aktuellsten Daten der Datenbank zu bekommen. Initial ist das Datum des DatePickers auf diese aktuellen Daten eingestellt. Um im späteren Verlauf die Funktion getPspModel() (vgl. Listing 6.31) erneut ausführen zu können, wenn der DatePicker bereits geändert wurde, wurde der Umweg des Auslesens des Datums über den DatePicker gewählt. Mit der PSP-Nummer und dem Datum kann nun der Service „getProjectStatus.xsjs“ mittels AJAX-Aufruf gestartet werden. Mit dem response, welcher einen JSON mit den Daten des Projektes zurückliefert, wird nun das Model „projectTotalReportingModel“ gesetzt.
Anschließend wird der Parameter „scoreValue“ aus dem bereits erstellten Model ausgelesen, welcher angibt, ob die Statusanzeige des Projekts nach Stunden oder nach Budget erfolgt. Zuletzt wird die Funktion „getClosedEntryModel()“ aufgerufen, welches in einem AJAX-Aufruf einen Service ausführt, um das Model der gesperrten Elemente des Projekts anzuzeigen.

```
1    getPspModel: function(){
2
3    // Die instance der List
4    var list = sap.ui.getCore().byId("listMasterReporting");
```

```
5       // Welches item wurde ausgewaehlt? Liefert einen "index".
6       var sItem = list.getSelectedItem();
7
8       // den Path vom ausgewaehlten Item herausfinden
9       var sPath = sItem.oBindingContexts.pspdata.sPath;
10
11      var item = sap.ui.getCore().getModel('pspdata').getProperty(sPath);
12
13
14      // Variablen deklarieren
15      var pNumber = item.PROJECTNUMBER;
16      this.selectedProject = item.PROJECTNUMBER;
17      this.projectBudgetPercent = item.BUDGET_PERCENT;
18      this.projectHoursPercent = item.HOURS_PERCENT;
19      this.projectLimitYellow = item.PROJECT_LIMIT_YELLOW;
20      this.projectLimitRed  = item.PROJECT_LIMIT_RED;
21
22
23      //  Variable valueOfDatePicker setzen
24      var date = sap.ui.getCore().byId("pspDate")._lastValue;
25      this.valueOfDatePicker = date;
26
27      this.getPSPData(pNumber,date);
28
29      // Model setzen mit den Daten aus
30      var self = this;
31
32      $.ajax({
33        url: "/abat/demosystem/projektcontrolling/controlling/services/
        getProjectStatus.xsjs?PNR="+pNumber + "&DATE="+date,
34        async:false,
35        type:"GET",
36        dataType:"json",
37        cache:"false",
38        success: function(response){
39          self.getView().getModel("projectTotalReportingModel").setData(
        response);
40        },
41        complete: function(){
42
43        },
44        error:function(xhr,status,error){
45
46        }
47      });
48
49      this.scoreValue = sap.ui.getCore().getModel('
        projectTotalReportingModel').getData().values[0].SCORE_VALUE;
50      this.getClosedEntryModel();
51    },
```

Listing 6.31: Projektcontrolling: Reporting Master Page getPspModel()

Durch den Aufruf der Funktion „setValueLockedEntry()“ werden die im Projekt vorhandenen Sperreinträge gezählt und die Anzahl an den IconTabFilter „Sperreinträge“ übergeben (vgl. Listing 6.32). Hierdurch ist dem Nutzer die Anzahl der vorhandenen Sperreinträge bekannt, ohne dass dieser den IconTabFiler ausklappen muss, bzw. die einzelnen Einträge zählen muss. Den Wert für die Anzahl wird aus dem erstellten Model „modelclosedentry“ gelesen. Mit Hilfe der ID und der Funktion „setCount()“ wird daraufhin die Anzahl der Einträge in den Filter geschrieben.

```
1  /**
2   * Setzt die Anzahl der Sperreintraege sichtbar neben den IconTabFilter
    auf der Details-Page "Projekt-Status".
3   */
4  setValueLockedEntry: function(){
5    var lockedFilter = sap.ui.getCore().byId("lockedFilter");
6    var item = sap.ui.getCore().getModel('modelclosedentry').aBindings[0].
    oList.length;
7    lockedFilter.setCount(item);
8  },
```

Listing 6.32: Projektcontrolling: Reporting Master Page setValueLockedEntry()

Die in der Funktion „itemSelected()“ gesetzte Variable „scoreValue“ wird in der Funktion „setValueLockedEntry()“ ausgelesen (vgl. Listing 6.33). Entsprechend dieses Wertes wird die RadioButtonGroup „buttonsProjectState“ auf den ausgelesenen Wert gesetzt. Hierbei wird ein Index übergeben, der den jeweiligen Button auf „selected“ stellt.

```
1    /**
2   * Die Radio-Buttons, mit denen ausgewaehlt wird, ob die Statusanzeige
    nach Stunden oder nach Budget
3   * erfolgt, werden auf den eingestellten Wert gesetzt.
4   * "H" = nach Stunden
5   * "B" = nach Budget
6   */
7  pushScoreValue: function(){
8    var index;
9
10   if(this.scoreValue=="H"){
11     index=1;
12   }else
13     index=0;
14
15   sap.ui.getCore().byId("projectState").setSelectedIndex(index);
16
17 },
```

Listing 6.33: Projektcontrolling: Reporting Master Page setValueLockedEntry()

Um den Status in der Projekt-Auswertung farblich zu markieren, wurde die Funktion „setProjectInfoState()“ implementiert (vgl. Listing 6.34 auf der nächsten Seite). Diese parst die Werte „projectBudgetPercent“, „projectHoursPercent“, „projectLimitYellow“ und „projectLimitRed“ in Integer-Werte, um sie miteinander vergleichen zu können. Außerdem wird der Parameter „scoreValue“ ausgelesen, um festzustellen, nach welchem

Kriterium ausgewertet werden soll. Die im scoreValue vorhandenen Kriterien sind dabei „H“ für eine Statusanzeige nach Stunden und „B“ für eine Anzeige nach Budget. Anschließend findet ein Vergleich der Werte statt und der Status wird an den IconTab-Filter „Gesamtstatus“ übergeben. Die übergebenen IconColors sind:

- grüner Status: sap.ui.core.IconColor.Positive
- gelber Status (orange): sap.ui.core.IconColor.Critical
- roter Status: sap.ui.core.IconColor.Negative
- blauer Status: sap.ui.core.IconColor.Neutral

```
1  /**
2   * Eine Funktion, die den Status des IconTabFilters entsprechend der
    aktuellen Werte (projectBudgetPercent, projectHoursPercent)
3   * aendert. Ist die Grenze zum gelben Schwellenwert ueberschritten (
    projectLimitYellow), wird der IconTabFilter als "Critical" (orange)
    gefaerbt.
4   * Ist die Grenze zum roten Schwellenwert (projectLimitRed)
    ueberschritten, wird der IconTabFiler als "Negative" (rot) gefaerbt.
    Bei nicht erreichen
5   * des gelben Schwellenwertes wird der IconTabFilter als "Positive" (
    gruen) markiert. Wurde kein Schwellenwert hinterlegt erscheint der
    IconTabFilter
6   * als "Neutral" (blau).
7   */
8  setProjectInfoState: function(){
9
10 var budgetPercent = parseInt(this.projectBudgetPercent);
11 var hoursPercent = parseInt(this.projectHoursPercent);
12 var limitYellow = parseInt(this.projectLimitYellow);
13 var limitRed = parseInt(this.projectLimitRed);
14
15 if(this.scoreValue=="H"){
16   caseValue=hoursPercent;
17
18 }else{
19
20   caseValue=budgetPercent;
21 }
22
23 //Vergleich der Variablen und deklarieren von state entsprechend des
    Status
24 if (caseValue <= limitYellow){
25   sap.ui.getCore().byId("projectStatusFilter").setIconColor(sap.ui.core.
    IconColor.Positive);
26 }
27 else if(caseValue <= limitRed && caseValue >  limitYellow){
28   sap.ui.getCore().byId("projectStatusFilter").setIconColor(sap.ui.core.
    IconColor.Critical);
```

```
29    }
30    else if(caseValue > limitRed){
31      sap.ui.getCore().byId("projectStatusFilter").setIconColor(sap.ui.core.
      IconColor.Negative);
32    }else{
33      sap.ui.getCore().byId("projectStatusFilter").setIconColor(sap.ui.core.
      IconColor.Neutral);
34    }
35  },
```

Listing 6.34: Projektcontrolling: Reporting Master Page setProjectInfoState()

Detail-Page: Projekt-Auswertung Wird in der Reporting Master-Page ein Projekt ausgewählt, gelangt der Nutzer zur Detail-Page „Projekt-Auswertung“ (vgl. Abbildung 6.17). Auf dieser wird der Status des Projektes in einem Reporting dargestellt und anhand von Grafiken anschaulich gemacht. In der oberen Hälfte der Seite ist der Status des Projektes, der CSV-Export und die gesperrten Einträge enthalten. Darunter befindet sich eine Tabelle mit allen im Projekt befindlichen PSP-Nummern. Im unteren Bereich befindet sich die grafische Auswertung der einzelnen PSP-Elemente.
Im Folgenden werden die einzelnen Bereiche der Seite an Code-Beispielen erläutert.

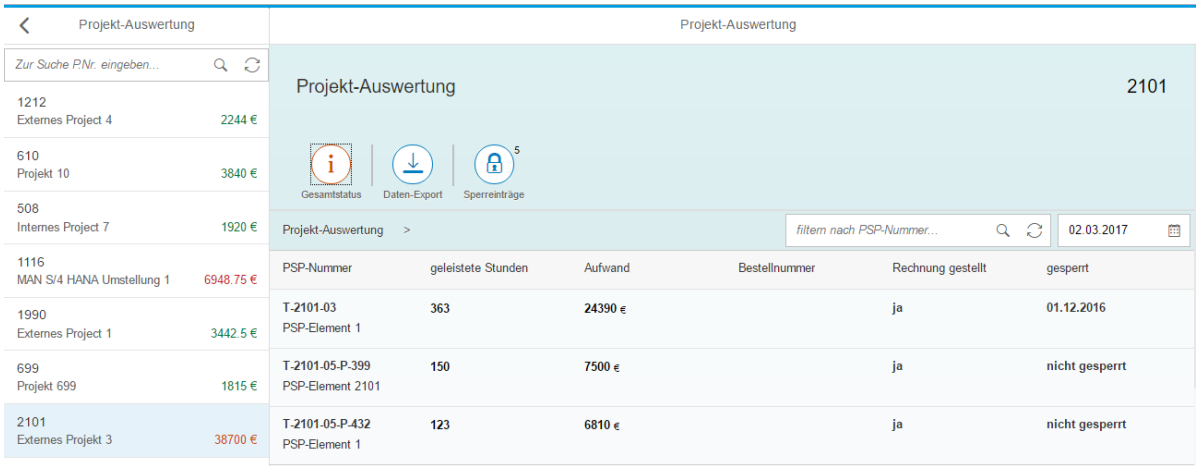


Abbildung 6.17.: Projektcontrolling: Projekt-Auswertung

Neben dem ObjectHeader, welcher den Titel der Seite und die Projektnummer angibt, befindet sich im oberen Bereich die IconTabBar. Die dazugehörigen IconTabFilter „Gesamtstatus“, „Daten-Export“ und „Sperreinträge“ sind initial bei Aufruf der Seite eingeklappt. Um zum Inhalt zu gelangen, klickt der Nutzer auf den jeweiligen IconTabFilter, woraufhin sich dieser ausklappt. Jeder der IconTabFilter besitzt ein Icon, welche aus dem SAP Icon-Explorer stammt. [1] IconTabFilter sind durch IconTabSeparator (einem Trennstrich zwischen den Filtern) voneinander getrennt.

[1] https://openui5.hana.ondemand.com/iconExplorer.html

```
var oTabBar = new sap.m.IconTabBarProject({
  expanded: false,
  items : [ new sap.m.IconTabFilter("projectStatusFilter",{
        text : "Gesamtstatus",
            icon : "sap-icon://hint",
            content: [oBoxPsp],
        }),

        new sap.m.IconTabSeparator({

        }),

        new sap.m.IconTabFilter({
            text : "Daten-Export",
            icon : "sap-icon://download",
            content: [
                new sap.m.Button({
            text: "csv-Export",
            icon:"sap-icon://excel-attachment",
            press:  function() {
          oController.csvExport();
              }})
                  ] ,
            iconColor: "Neutral"
        }),
        new sap.m.IconTabSeparator({

        }),

        new sap.m.IconTabFilter("lockedFilter",{
            text : "Sperreintraege",
            count: "7",
            icon : "sap-icon://locked",
            content: [oTableClosedEntry] ,
            iconColor: "Neutral"
        })
      ]
    });
```

Listing 6.35: Projektcontrolling: Projekt-Auswertung IconTabBarProject

IconTabFilter Gesamtstatus: Der Inhalt des IconTabFilters „Gesamtstatus“ ist der umfangreichste, da dieser die relevanten Informationen zum Projektstatus enthält (vgl. Abbildung 6.18 auf der nächsten Seite). Der Inhalt befindet sich in einer HBox[2] (oBoxPSP), welche wiederum VBoxen[3] enthält, die den eigentlichen content darstellen. Aufgeteilt ist der Status dabei in drei VBoxen, welche das verbrauchte Budget / die verbrauchten Stunden, die Einstellung der Anzeige des Status und Kennzahlen des Projektes enthalten.

[2]sap.m.HBox ist ein Container, welcher die Einträge horizontal anordnet
[3]sap.m.VBox ist das Gegenstück der HBox und ordnet die Einträge vertikal an

Abbildung 6.18.: Projektcontrolling: Projekt-Auswertung Status

Zur Status-Darstellung der aufgewendeten Stunden und des aufgewendeten Budgets werden BulletCharts verwendet. BulletCharts sind Balkendiagramme, mit denen es möglich ist Schwellenwerte und die aktuell aufgewendeten Werte darzustellen. Die Datenbereitsteller, BulletChartData, enthalten die relevanten Daten und werden an das BulletChart übergeben (vgl. Listing 6.36 Zeile 53f.). Die Daten für die BulletCharts in der Projekt-Auswertung stammen aus dem Model „projectTotalReportingModel“. Das BulletChart „bulletHdataProject“ enthält die gesamten Stunden eines Projektes in Prozent und stellt die jeweiligen Schwellenwerte dar.

```
1       /**
2         * Ein BulletChart, welches die aktuellen Stunden des Projektes in
     Prozent anzeigt. In den BulletChartData (bulletHdataProject,
3         * bulletHdataProjectYellow, bulletHdataProjectRed) werden die
    relevanten Daten des Projektes geladen. Es wird ein Formatter verwendet
    ,
4         * der die als String uebergebenen Werte in den Datentyp Float
    parst.
5         * Uebergeben werden die Werte:
6         * HOURS_PERCENT (bulletHdataProject): aktuell aufgewendete
    Stunden
7         * PROJECT_LIMIT_YELLOW (bulletHdataProjectYellow): gelbes Limit
    des Projektes
8         * PROJECT_LIMIT_RED (bulletHdataProjectRed): rotes Limit des
    Projektes
9         * Im BulletChart werden die deklarierten BulletChartData als
    Werte angegeben:
10        * actual: aufgewendete Stunden des Projektes
11        * thresholds: gelbes und rotes Limit des Projektes
12        */
13      var bulletHdataProject =  new sap.suite.ui.commons.BulletChartData
    ({
14
15          value : {
```

```
16          path: "projectTotalReportingModel>/values/0/HOURS_PERCENT",
17            formatter: function(value){
18              return parseFloat(value);
19        }},
20
21
22
23           });
24
25          var bulletHdataProjectYellow =  new sap.suite.ui.commons.
    BulletChartData({
26
27            value : {
28        path: "projectTotalReportingModel>/values/0/PROJECT_LIMIT_YELLOW",
29          formatter: function(value){
30            return parseFloat(value);
31
32       }},
33
34          color : sap.suite.ui.commons.InfoTileValueColor.Critical
35
36          });
37
38           var bulletHdataProjectRed =  new sap.suite.ui.commons.
    BulletChartData({
39
40            value : {
41          path: "projectTotalReportingModel>/values/0/PROJECT_LIMIT_RED",
42            formatter: function(value){
43              return parseFloat(value);
44        }},
45        color : sap.suite.ui.commons.InfoTileValueColor.Error,
46
47         });
48
49          var bulletProjectH =  new sap.suite.ui.commons.BulletChart({
50
51          size : sap.suite.ui.commons.InfoTileSize.Auto,
52          actual :[bulletHdataProject],
53          thresholds : [bulletHdataProjectYellow, bulletHdataProjectRed],
54          showValueMarker :false,
55          minValue : 0,
56          maxValue : 120
57        });
```

Listing 6.36: Projektcontrolling: Projekt-Auswertung BulletChart

Äquivalent zu dem oben dargestellten BulletChart, erfolgt die Darstellung des Budgets eines Projektes im BulletChart „bulletHdataProject“.

Die Anzeige des Status im IconTabFilter kann nach Stunden oder nach Budget erfolgen. Nach welchem Kriterium die Anzeige erfolgt, kann innerhalb des IconTabFilters

festgelegt werden. Dies geschieht mit RadioButtons in einer RadioButtonGroup (vgl. Listing 6.37), welche in der mittleren VBox implementiert ist. Innerhalb dieser erhält jeder Button einen Index, welcher bei Auswahl eines Buttons mit Hilfe vom Objekt oEvent im Property „select“ abgefangen werden kann. Dieser Index wird daraufhin an die Funktion „updateScoreValue()“ übergeben.

```
/**
      * Buttons zur Statusaenderung des Projektes. Die Anzeige des Status kann nach
      * Budget und Stunden erfolgen.
      * Zur Aenderung der Statusanzeige wird die Funktion updateSoreValue ausgefuehrt.
      */
 var buttonsProjectState = new sap.m.RadioButtonGroup("projectState",{

   buttons: [

     new sap.m.RadioButton("budgetProjectButton",{text: "Budget",
       }),

    new sap.m.RadioButton("hoursProjectButton",{text: "Stunden"})
      ],

    select: function(oEvent){

            index = oEvent.getParameters().selectedIndex;

            oController.updateScoreValue(index);
     }
    });
```

Listing 6.37: Projektcontrolling: Projekt-Auswertung buttonsProjectState

Der an die Funktion „updateScoreValue()“ (vgl. Listing 6.38) übergebene Index wird in einen Wert gebracht, um eine Übergabe an den entsprechenden Service vorzunehmen. Da der erste Button eine Auswahl nach dem Budget impliziert, wird der zu übergebene Parameter „scoreValue“ auf „B“ gesetzt, ist der Index = 0, wird scoreValue als „H“ deklariert. Durch den Aufruf des Services mittels der entsprechenden URL, wird dieser Wert für das ausgewählte Projekt in der Datenbank gespeichert, damit die Statusanzeige auch bei erneutem Aufruf der Seite nach der getätigten Einstellung erfolgt.

```
/**
 * Der Wert nach dem die Statusanzeige erfolgt, wird entsprechend der Auswahl in die Datenbank geschrieben
 * index - kann 1 (nach nach Budged <B>) oder 0 (nach Stunden <H>) annhemen
 */

updateScoreValue: function(index){

  this.selectedProject

```

```
10      if (index==0){
11        this.scoreValue="B";
12      }
13      else{
14        this.scoreValue="H";
15      }
16      var self= this;
17      $.ajax({
18      url: "/abat/demosystem/projektcontrolling/controlling/services/update/
        updateScoreValues.xsjs?PROJECTNUMBER="+this.selectedProject+"&SCORE="+
        this.scoreValue,
19      async:false,
20      type:"GET",
21      dataType:"json",
22      cache:"false",
23      success: function(response){
24
25      },
26      complete: function(){
27
28      },
29      error:function(xhr,status,error){
30
31      }
32    });
33
34      this.setProjectInfoState();
35    },
```

Listing 6.38: Projektcontrolling: Projekt-Auswertung updateScoreValue()

Um beim Aufrufen der Details-Page „Projekt-Auswertung“ die Buttons auf den eingestellten scoreValue zu setzen, wird die Funktion „pushScoreValue()“ verwendet (vgl. Listing 6.39). Diese liest die RadioButtonGroup aus und setzt den Index entsprechend dieser Einstellung.

```
1     /**
2     * Die Radio-Buttons, mit denen ausgewaehlt wird, ob die Statusanzeige
      nach Stunden oder nach Budget
3     * erfolgt, werden auf den eingestellten Wert gesetzt.
4     * "H" = nach Stunden
5     * "B" = nach Budget
6     */
7
8   pushScoreValue: function(){
9     var index;
10
11    if(this.scoreValue=="H"){
12      index=1;
13    }else
14      index=0;
15
```

```
16      sap.ui.getCore().byId("projectState").setSelectedIndex(index);
17    },
```

Listing 6.39: Projektcontrolling: Projekt-Auswertung pushScoreValue()

Die dritte implementierte VBox enthält die wesentlichen Projektkennzahlen, welche mit Labels dargestellt werden. Zur Darstellung dieser Kennzahlen werden Formatter verwendet, welche die Werte in Volltext-Form als String darstellen. Als Design der Labels wurde „Bold“ verwendet, welches die Labels in fetter Schrift darstellen (vgl. Listing 6.40).

```
1
2                   new sap.m.VBox({
3
4                   items:[
5                           new sap.m.Label({
6                             text:{
7                               path: 'projectTotalReportingModel>/values/0/
    FIXED_VALUES',
8                               formatter: function(fixedValue){
9                                 return oController.parseFixedValue(
    fixedValue);
10                              }
11                            },
12                            design:"Bold"
13                          }),
14
15                          new sap.m.Label({
16                            text:{
17                              path: 'projectTotalReportingModel>/values/0/
    PLANNED_BUDGET',
18                              formatter: function(plannedBudget){
19                                return "geplantes Budget: " +
    plannedBudget+"EURO";
20                              }
21                            },
22                            design:"Bold"
23                          }),
24                          new sap.m.Label({
25                            text:{
26                              path: 'projectTotalReportingModel>/values/0/
    STATUS_BUDGET',
27                              formatter: function(statusBudget){
28                                return "aktueller Aufwand: " +
    statusBudget+"EURO";
29                              }
30                            },
31                            design:"Bold"
32                          }),
33
34                          new sap.m.Label({
35                            text:{
36                              path: 'projectTotalReportingModel>/values/0/
    PLANNED_HOURS',
```

```
37                                formatter: function(plannedHours){
38                                  return "geplante Stunden: " + plannedHours
    ;
39                                }
40                              },
41                              design:"Bold"
42                            }),
43                            new sap.m.Label({
44                              text:{
45                                path: 'projectTotalReportingModel>/values/0/
    STATUS_HOURS',
46                                formatter: function(statusHours){
47                                  return "aktuell aufgewendete Stunden: " +
    statusHours;
48                                }
49                              },
50                              design:"Bold"
51                            })
52                          ]
53                  })
```

Listing 6.40: Projektcontrolling: Projekt-Auswertung pushScoreValue()

IconTabFilter Daten-Export Aufgrund der Tatsache, dass im IconTabFilter „Daten-Export“ vergleichsweise wenig Inhalt ist, wurde dieser direkt, also ohne Variablen in den content implementiert (vgl. Listing 6.35 auf Seite 255 Zeile 14 ff.). Durch das Klicken des Buttons zum CSV-Export wird eine Funktion ausgeführt welche diesen Export veranlasst (vgl. Listing 6.41 auf der nächsten Seite).

In der Variable „sUrl“ wird der Aufruf des Services gespeichert. Dabei werden auch die ausgewählte Projektnummer und das ausgewählte Datum mit übergeben (vgl. Listing 6.41 auf der nächsten Seite Zeile 6). Daraufhin wird die URL in das entsprechende Format kodiert, um beispielsweise Sonderzeichen für den Aufruf umzuformatieren. Des Weiteren wird die URL im selben Fenster ausgeführt, indem für den Parameter „bNewWindow“ der Wert „false“ übergeben wird (vgl. Listing 6.41 auf der nächsten Seite Zeile 8 f.).

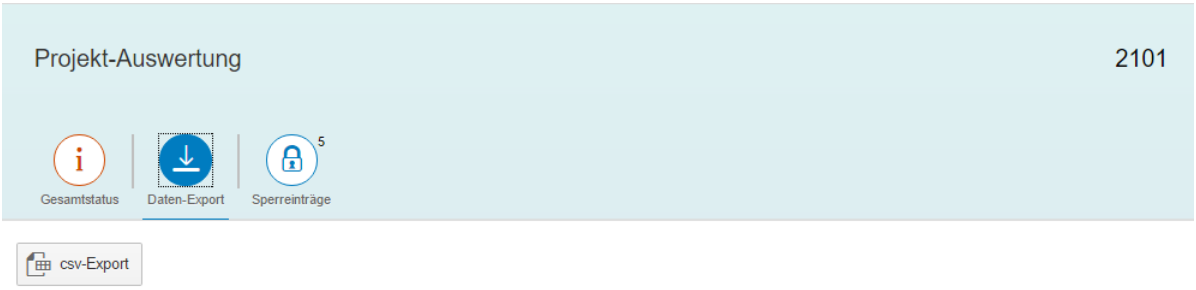


Abbildung 6.19.: Projektcontrolling: Projekt-Auswertung Export

```
1     /**
2    * Fuehrt den entsprechenden Service aus, der den CSV-Export durchfuehrt
       .
3    */
4     csvExport: function(){
5
6     var sUrl = "/abat/demosystem/projektcontrolling/controlling/services/
     csv_export.xsjs?pnr=" + this.selectedProject+"&DATE="+this.
     valueOfDatePicker;
7
8     var encodeUrl = encodeURI(sUrl);
9     sap.m.URLHelper.redirect(encodeUrl,false);
10    },
```

Listing 6.41: Projektcontrolling: Projekt-Auswertung csvExport()

Zur Auswertung der geladenen CSV-Datei, wurde eine Excel-Vorlage erstellt. Diese wird im Abschnitt 6.3.6 auf Seite 305 erläutert.

IconTabFilter Sperreinträge Wurde ein PSP-Element gesperrt und sind Einträge nach dem Sperrdatum eingegangen, tauchen diese in der Table „oTableClosedEntry“ auf (vgl. Listing 6.20), welche sich im IconTabFilter „Sperreinträge“ befindet. Der mode dieses Tables ist auf „sap.m.ListMode.MultiSelect“ eingestellt, was bedeutet, dass mehrere Elemente dieser Tabelle ausgewählt werden können. Die Toolbar beinhaltet einen Button, welcher die ausgewählten Einträge der Tabelle entsperrt. Dabei wird die Funktion „unlockPSP()“ aufgerufen, welche die Einträge übergibt (vgl. Listing 6.42).
Die Spalten der Tabelle werden deklariert, indem für jede Spalte eine sap.m.Column mit dem Spaltennamen als Label erstellt wird.

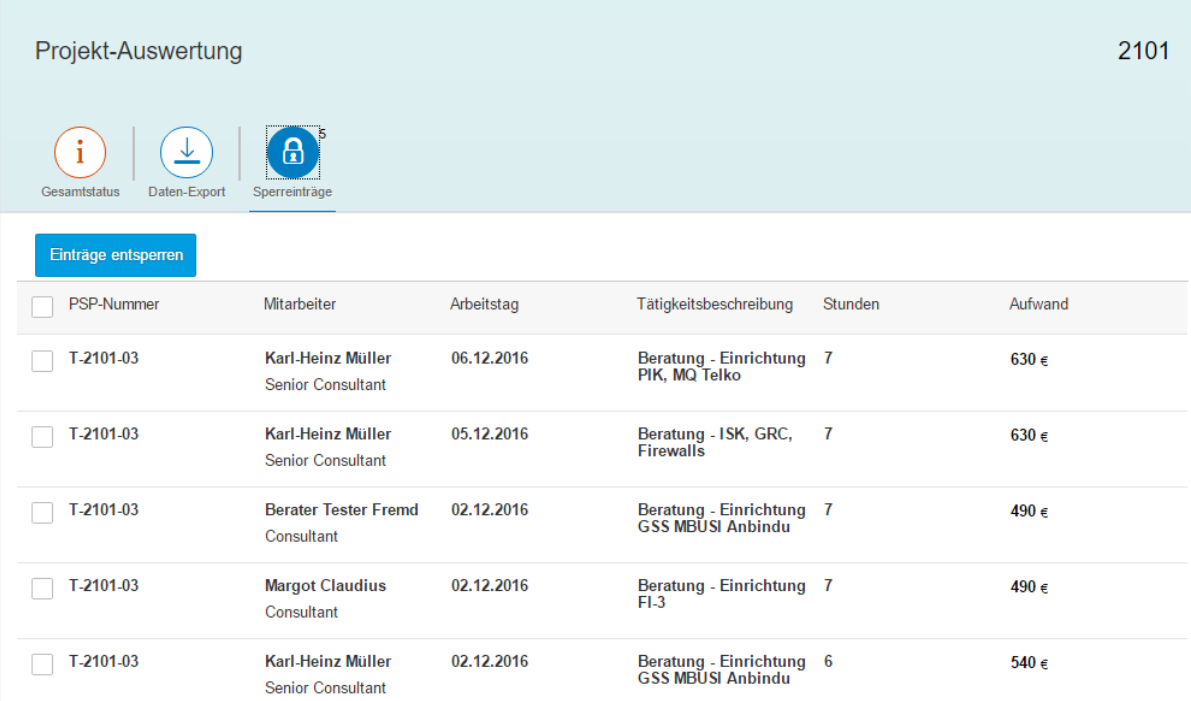


Abbildung 6.20.: Projektcontrolling: Projekt-Auswertung gesperrte Elemente

```
/**
 * Tabelle, die alle gesperrten Eintraege eines Projektes enthaelt. Die
   Eintraege koennen ausgewaehlt werden (mode: sap.m.ListMode.MultiSelect)
 * Durch den Button "unlockPsp" werden alle ausgewaehlten Eintraege
   entsperrt (oController.unlockPSP(contexts)). Die Variable context
   repraesentiert
 * die in der Tabelle ausgewaehlten Eintraege.
 *
 */


    var oTableClosedEntry = new sap.m.Table("tableclosedentry",{

      mode: sap.m.ListMode.MultiSelect,
      headerToolbar: [
                new sap.m.Toolbar({
                content: [
        new sap.m.Button("unlockPsp",{
        type: sap.m.ButtonType.Emphasized,
          text: 'Eintraege entsperren',
          press: function() {
            var contexts = oTableClosedEntry.getSelectedContexts();
            oController.unlockPSP(contexts);

          }
        })
                                        ]})],
```

```
25
26      columns:[
27                new sap.m.Column({
28                  header: new sap.m.Label({
29                    text: "PSP-Nummer"
30                  })
31                }),
32                new sap.m.Column({
33                  header: new sap.m.Label({
34                    text: "Mitarbeiter"
35                  })
36                }),
37
38                new sap.m.Column({
39                  header: new sap.m.Label({
40                    text: "Arbeitstag"
41                  })
42                }),
43                new sap.m.Column({
44                  header: new sap.m.Label({
45                    text: "Taetigkeitsbeschreibung"
46                  })
47                }),
48                new sap.m.Column({
49                  header: new sap.m.Label({
50                    text: "Stunden"
51                  })
52                }),
53                new sap.m.Column({
54                  header: new sap.m.Label({
55                    text: "Aufwand"
56                  })
57                })
58                ]
59    });
```

Listing 6.42: Projektcontrolling: Projekt-Auswertung oTableClosedEntry

Das Binding wird ausgeführt, indem die Werte aus dem entsprechenden Model zunächst einem Template zugewiesen werden (vgl. Listing 6.43). Dazu wird für jede Spalte eine ObjectNumber bzw. ein ObjectIdentifier erzeugt. ObjectNumbers werden für Zahlenwerte genutzt, die eine Einheit benötigen, ObjectIdentifier für Spalten mit Text oder einfache Zahlenwerte ohne Einheit. Abschließend wird das Binding durchgeführt, indem die Table und das Template übergeben werden.

```
1
2     var oTableClosedEntryTemplate= new sap.m.ColumnListItem("
    tableclosedentrystemplate", {
3       cells:[
4               new sap.m.ObjectIdentifier({
5                 title:"{modelclosedentry>PSP_NUMBER}"
6               }),
7               new sap.m.ObjectIdentifier({
```

```
                title:{
                path: "modelclosedentry>",
              formatter: function(value){
                return oController.getFullName(value);

          }},
                  text: "{modelclosedentry>LEVEL}"
              }),
              new sap.m.ObjectIdentifier({
                title:"{modelclosedentry>DATE}"
              }),
              new sap.m.ObjectIdentifier({
                title:"{modelclosedentry>WORK_DESCRIPTION}"
              }),
              new sap.m.ObjectIdentifier({
                title:"{modelclosedentry>HOURS}"
              }),
              new sap.m.ObjectNumber({
                 number: "{modelclosedentry>RATED}",
                 unit: "EURO"
                })
              ],

});
  oTableClosedEntry.bindAggregation("items","modelclosedentry>/values",
      oTableClosedEntryTemplate);
  oTableClosedEntry.setNoDataText("Keine gesperrten Elemente vorhanden")
```

Listing 6.43: Projektcontrolling: Projekt-Auswertung oTableClosedEntryTemplate

Durch das Betätigen des Buttons zum Entsperren, wird die Funktion „unlockPSP()“ ausgeführt. Der übergebene Parameter „contexts“ birgt alle Einträge, welche vom Nutzer ausgewählt wurden.
In der Variable “i“ der Funktion, wird die Anzahl der Einträge gespeichert. Falls nichts ausgewählt wurde, erscheint auf dem Bildschirm eine Message-Box, welche darauf hinweist (vgl. Listing 6.44). Die Funktion wird daraufhin verlassen. Ist die Anzahl der ausgewählten Einträge größer als 0, werden die Einträge in einer Schleife durchlaufen und durch den entsprechenden Service (updateCloseDate) aufgerufen, bis alle Einträge entsperrt sind. Zum Entsperren werden die PSP-Nummer und das Datum, an dem laut Eintrag gearbeitet wurde, an den Service übergeben. Abschließend werden die Daten der Models neu geladen, in denen sich durch das Entsperren Änderungen ergeben haben. Auch die Funktion „setValueLockedEntry()“ wird erneut aufgerufen, um die neue Anzahl der Sperreinträge über dem IconTabFilter anzuzeigen.

```
/**
 * unlockPSP setzt das Sperrdatum entsprechend der ausgewaehlten
  Eintraege um
 * Zunaechst wird ueberprueft,ob Ein Eintrag ausgewaehlt wurde. Wurde
  nichts ausgewaehlt erscheint eine entsprechende Fehlermeldung.
 * In einer Schleife werden die Sperreintraege geloescht, welche
  ausgewaehlt wurden.
```

```
   */

  unlockPSP:function(contexts){

    var i = contexts.length;
    if (i == 0){

      jQuery.sap.require("sap.m.MessageBox");
      sap.m.MessageBox.show("Kein Eintrag ausgewaehlt!", {
          icon: sap.m.MessageBox.Icon.ERROR,
          title: "Error",
          actions: [sap.m.MessageBox.Action.OK],
          onClose: function(oAction) {


          }.bind(this)
      });

      return;
    }

    while(i!=0){
      var sPath= contexts[i-1].sPath;
      var item = sap.ui.getCore().getModel('modelclosedentry').getProperty
  (sPath);

      //Variablen deklarieren

        var pspNumber=item.PSP_NUMBER;
        var date = item.DATE;

      //AJAX-Aufruf

      var self= this;
        $.ajax({
        url: "/abat/demosystem/projektcontrolling/controlling/services/
  update/updateCloseDate.xsjs?PSP="+pspNumber+"&DATE="+date,
        async:false,
        type:"GET",
        dataType:"json",
        cache:"false",
        success: function(response){

        },
        complete: function(){

        },
        error:function(xhr,status,error){

        }
      });

```

```
55
56          i--;
57        }
58        this.getPspModel();
59        this.getClosedEntryModel();
60        this.setValueLockedEntry();
61      },
```

Listing 6.44: Projektcontrolling: Projekt-Auswertung MessageBox Sperreinträge

oTablePSP Die einzelnen PSP-Elemente eines Projektes werden in der Tabelle “oTablePSP“ aufgeführt. Folgende Spalten sind in dieser Tabelle enthalten:

- PSP-Nummer: zum Projekt gehörende PSP-Nummer
- geleistete Stunden: aggregierte Stunden eines PSP-Elements
- Aufwand: mit den Stundensätzen der verschiedenen Skill-Level verrechnete geleistete Stunden
- Bestellnummer: die zum Projekt gehörende Bestell- bzw. Rechnungsnummer
- Rechnung gestellt: Tatsache, ob das PSP-Element in Rechnung gestellt wurde
- gesperrt: Status, ob das PSP-Element gesperrt wurde

Der Aufbau und das Binding der Tabelle entspricht der schon beschriebenen Tabelle „oTableClosedEntry“. Damit ein PSP-Element ausgewählt werden kann, wurde der mode auf „sap.m. ListMode.SingleSelectMaster“ gesetzt. Somit ist es ähnlich wie bei der schon beschriebenen List in der Master-Page „oMasterReporting“ möglich ein PSP-Element auszuwählen, um nähere Details dazu zu erfahren.

```
1       mode: sap.m.ListMode.SingleSelectMaster,
2       select: [oController.pspSelected,oController],
3       headerToolbar: [
4             new sap.m.Toolbar({
5             content: [
6                 new sap.m.Label({
7                   text:" Projekt-Auswertung "
8                 }),
9
10                new sap.m.ToolbarSpacer({
11                    width: "1rem"
12            }),
13
14                new sap.m.Label({
15                text:" > "}),
16                      new sap.m.Title({
17                      }),
18                new sap.m.ToolbarSpacer({
19                }),
20
```

```
21                  new sap.m.SearchField("pspfilter",{
22                  placeholder: "filtern nach PSP–Nummer...",
23                  liveChange: [oController.filterPsp, oController],
24                  showRefreshButton: true,
25                  width: "20rem"
26                     }),pspDate
27                 ]})],
```

Listing 6.45: Projektcontrolling: Projekt-Auswertung oTablePSP Binding

Grafiken Projektauswertung Die aufgewendeten Stunden und das aufgewendete Budget pro PSP-Element werden in Grafiken anschaulich dargestellt (vgl. Abbildung 6.21). Ähnlich wie im IconTabFilter werden diese anhand von Balkendiagrammen veranschaulicht. Das Augenmerk liegt jedoch in den einzelnen PSP-Elementen und nicht im Gesamtprojekt. Auch das verwendete Chart unterscheidet sich zur Projektauswertung, da ein VizFrame zur Darstellung der einzelnen PSP-Elemente verwendet wird.

Mit der VizFrame Control wird das Chart instanziiert und das Layout festgelegt. Dabei werden die Höhe und die Breite, aber auch der Typ des Charts festgelegt (vgl. Listing 6.46).

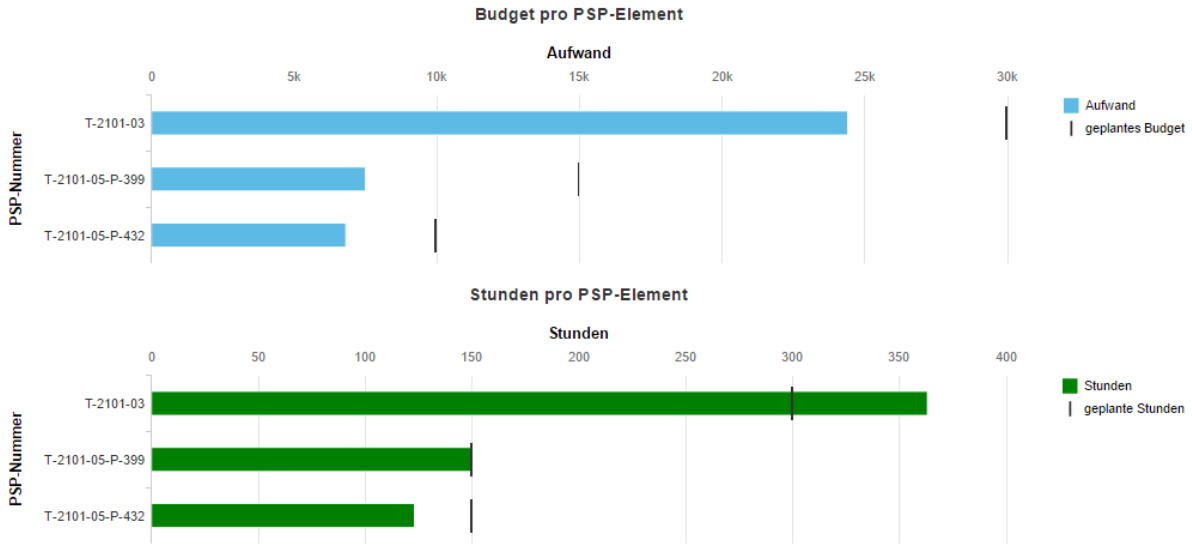


Abbildung 6.21.: Projektcontrolling: Projekt-Auswertung Grafiken

```
1     /**
2      * Darstellung der Stunden eines Projektes anhand von VizFrame Charts.
      Im VizFrame werden
3      * hierbei zunaechst der Typ, die Hoehe und die Breite des Charts
      bestimmt.
4      */
5
6
7     var oChartProject = new sap.viz.ui5.controls.VizFrame("chartproject",{
```

```
      width: "100%",
      height: "40%",
      vizType: 'bullet',

    });
```

Listing 6.46: Projektcontrolling: Projekt-Auswertung oChartProject

Zur Bestimmung der im Chart genutzten Werte, wird ein Dataset Control erstellt, welches zum einen die x-Achse und die y-Achse bestimmt und zum anderen den Zielwert festlegt. Die x-Achse beinhaltet dabei den gesamten „Aufwand“ in Euro, der im PSP-Element angefallen ist, auf der y-Achse sind die jeweiligen PSP-Elemente zu sehen und im Zielwert wird das geplante Budget angegeben.

```
      /**
     * Die Datasets werden erstellt, indem auf das Datenmodell "
    pspElementsModel" zugegriffen wird.
     * Mit diesem Dataset wird das Chart im Anschluss befuellt.
     */

    var oProjetDataset = new sap.viz.ui5.data.FlattenedDataset({
      measures:[{
          name: "Aufwand",
          value: "{pspElementsModel>TOTAL_RATED}"
      },
      {
          name: "geplantes Budget",
          value: "{pspElementsModel>PLANNED_BUDGET}"
      }

      ],
      dimensions:[{
        name: "PSP-Nummer",
        value: "{pspElementsModel>PSP_NUMBER}"
      }],
      data:{
        path: "pspElementsModel>/values"
      }
    });
```

Listing 6.47: Projektcontrolling: Projekt-Auswertung oProjetDataset

Wurden die Feeds erstellt, werden die Properties für das Chart gesetzt. Dabei wird die Sichtbarkeit auf „true“ gesetzt und der Titel des Charts angegeben. Zuletzt wird der Datensatz gesetzt und die Feeds werden dem Chart hinzugefügt (vgl. Listing 6.48).

```
    /**
     * Setzen der Properties des Charts. Hierbei wird visible=true und der
    name als
     * Text angegeben
     */
    oChartProject.setVizProperties({
      title: {
```

```
7                 visible: true,
8                 text: 'Budget pro PSP-Element'
9             }
10    });
11
12    /**
13     * Der Datensatz wird fuer das VizFrame gesetzt und die Feeds werden
      hinzugefuegt.
14     */
15    oChartProject.setDataset(oProjetDataset);
16    oChartProject.addFeed(project_feedValueAxis);
17    oChartProject.addFeed(project_feedCategoryAxis);
18    oChartProject.addFeed(project_feedBudgetValue);
```

Listing 6.48: Projektcontrolling: Projekt-Auswertung Properties für das Chart setzen und Feeds hinzufügen

Äquivalent zu oProjectDataset, wird das Chart oProjetDatasetHours aufgebaut, bei dem die PSP-Elemente nicht nach Aufwand in Euro ausgewertet werden, sondern nach aufgewendeten Stunden.

Breadcrumb Navigation Um nähere Informationen zum Projekt zu erlangen, kann der Nutzer die jeweilige PSP-Nummer in der Tabelle auswählen. Nun öffnet sich eine neue Details-Page. In der Toolbar der neuen Seite erscheint ein Link, mit dem es möglich ist zur vorherigen Ansicht zurückzunavigieren. Diese Art der Navigation wird als Breadcrumb Navigation bezeichnet. In einem Drill-Down gelangt der Nutzer Schritt für Schritt zur nächsten Detaillierungsebene. Auf diese Weise wurden in der Projekt-Auswertung drei verschiedene Detaillierungsebenen implementiert:

- Projekt-Auswertung: Die oben bereits vorgestellte Ebene bezeichnet die Projekt-Auswertung. Sie stellt die gröbste Detaillierungsebene dar (vgl. Abbildung 6.18 auf Seite 256).

- PSP-Auswertung: In der PSP-Auswertung gelangt der Nutzer an Informationen zu einem ausgewählten PSP-Element. Außerdem werden die Daten für jede Woche in einer Tabelle und in einem Liniendiagramm dargestellt (vgl. Abbildung 6.22 auf der nächsten Seite).

- Die PSP-Detailansicht stellt die detaillierteste Ansicht dar. Die entsprechenden Stunden werden für die jeweiligen Arbeitstage und Mitarbeiter aufgelistet. Des Weiteren sind die Planwerte für die ausgewählte Woche ersichtlich (vgl. Abbildung 6.25 auf Seite 272).

Die einzelnen Detail-Pages, welche die Breadcrumb-Navigation realisieren, werden im Folgenden näher beschrieben.

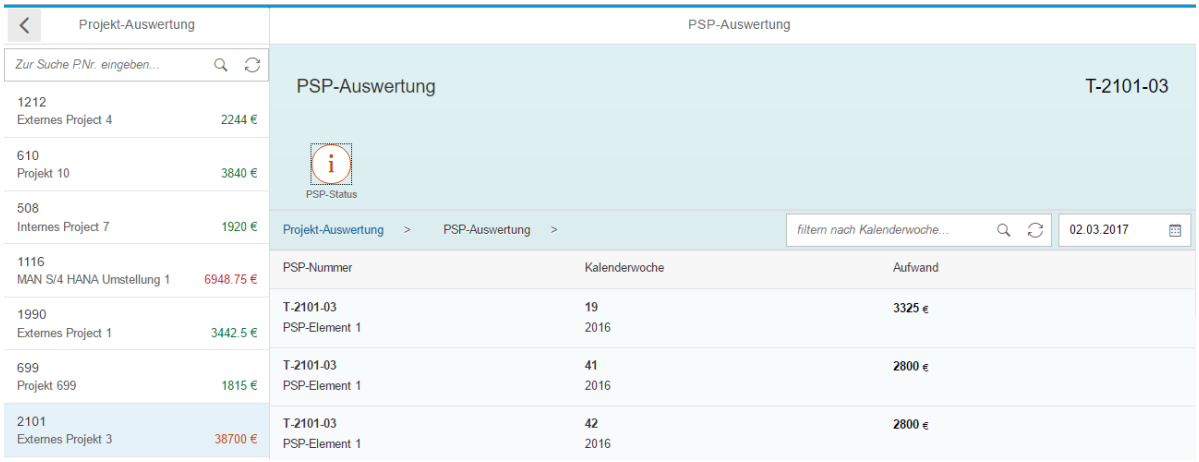


Abbildung 6.22.: Projektcontrolling: PSP-Auswertung

Detail-Page: PSP-Auswertung Der Aufbau der Detail-Page „PSP-Auswertung“ entspricht dem der Projekt-Auswertung (vgl. Abbildung 6.23). In der PSP-Auswertung ist jedoch lediglich ein IconTabFilter enthalten, da das Entsperren der Elemente und der Daten-Export in der Ebene Projekt-Auswertung stattfindet. Der IconTabFilter „PSP-Status“ ist dem „Gesamtstatus“ der darüberbefindlichen Ebene ähnlich. Das PSP-Element wird anhand von Kennzahlen und Balkendiagramme dargestellt (vgl. Abbildung 6.23).

Abbildung 6.23.: Projektcontrolling: PSP-Auswertung Status

In der Tabelle zum PSP-Element befindet sich der Aufwand in Euro, welcher pro Kalenderwoche angefallen ist. Diese Auflistung ist aufsteigend nach der Kalenderwoche sortiert. Im Suchfeld ist es möglich nach den Kalenderwochen zu suchen. Im unteren Bereich der Page befinden sich zwei Liniendiagramme, welche den Verlauf des Budgets und der Stunden anzeigen. Hierbei handelt es sich um Viz Charts, welches vom Typ „line“ sind (vgl. Abbildung 6.24 auf der nächsten Seite).

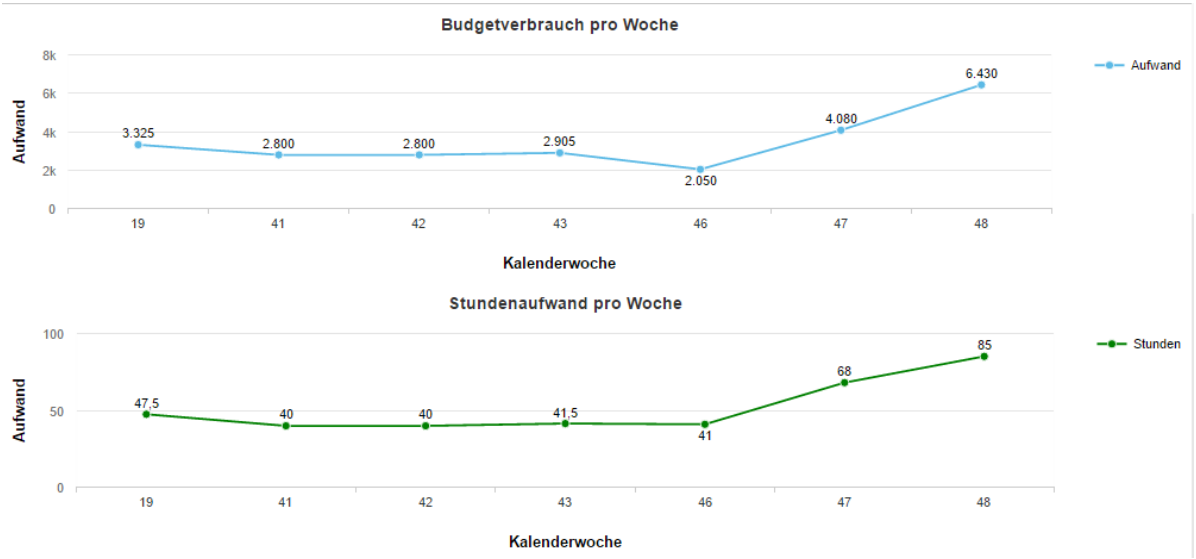


Abbildung 6.24.: Projektcontrolling: PSP-Auswertung Grafiken

Detail-Page: PSP-Detailansicht Die granularste Detaillierungsebene stellt die PSP-Detailansicht dar, welche für jeden Mitarbeiter, der in der ausgewählten Woche gearbeitet hat, die Arbeitsdaten darstellt (vgl. Abbildung 6.25). Explizit werden neben der PSP-Nummer der Name mit dem jeweiligen Skill-Level, der Arbeitstag als Datum, die Tätigkeitsbeschreibung, die Stunden und der Aufwand in Euro dargestellt. Im Suchfeld lassen sich die Mitarbeiter nach Nachnamen filtern

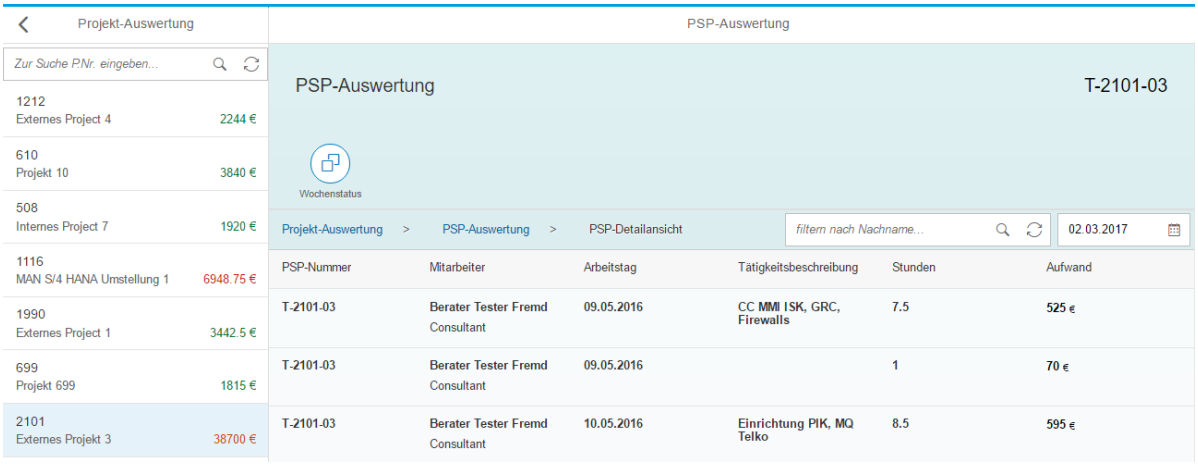


Abbildung 6.25.: Projektcontrolling: PSP-Detailansicht

Wird der IconTabFilter „PSP-Wochenauswertung“ ausgewählt, öffnet sich eine Tabelle, welche die geleisteten Stunden, den Aufwand in Euro und, falls vorhanden, die dazugehörigen Planwerte anzeigt (geplante Stunden und geplantes Budget). Damit Planwerte in der Auswertung mit auftauchen, müssen diese in der Dateneingabe angegeben worden sein (vgl. Abbildung 6.26 auf der nächsten Seite).

Abbildung 6.26.: Projektcontrolling: PSP-Detailansicht Status

Detail-Page: verspätete Timesheets Zur Detail-Page „verspaetete Timesheets“ gelangt der Nutzer über das Hauptmenü der App. In dieser Detail-Page kann nach Mitarbeitern gesucht werden, die bis zu einem bestimmten Zeitpunkt noch keine Stundenzettel abgegeben haben (vgl. Abbildung 6.27). Dieser Zeitpunkt wird im DatePicker angegeben, wobei initial immer das aktuelle Datum angegeben ist. Die Tabelle beinhaltet den Namen des Mitarbeiters zusammen mit dem entsprechendem Kürzel und das jeweilige Datum, zu dem zuletzt Arbeitsdaten eingegangen sind.

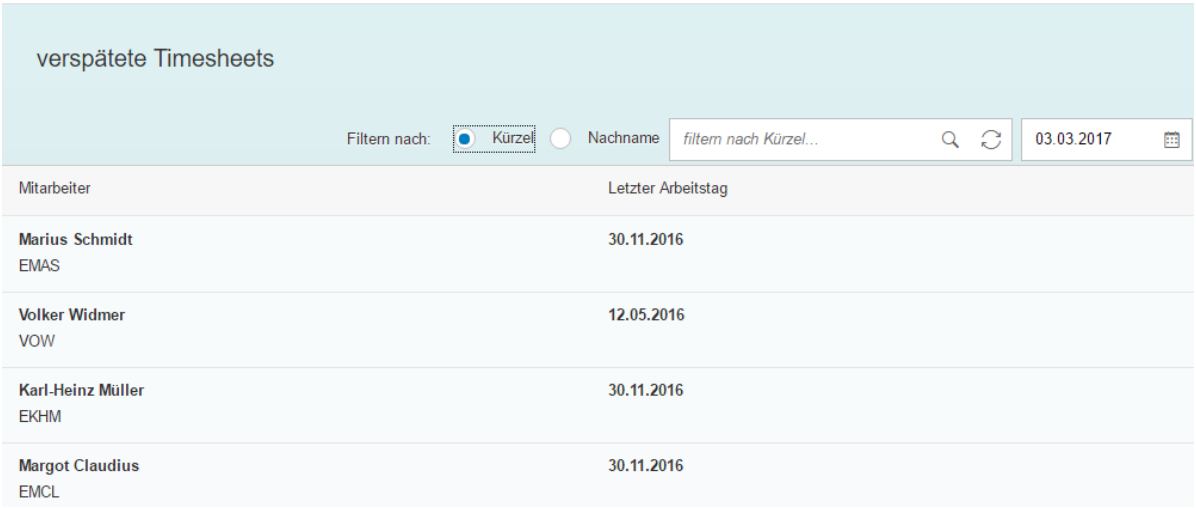


Abbildung 6.27.: Projektcontrolling: verspätete Timesheets

Wird anhand des DatePickers das Datum geändert, wird die Funktion „getModelWorkLast()“ ausgeführt, welche das Datum aus dem DatePicker liest und mit einem AJAX-Aufruf an den Service „getWorkLast.xsjs“ übergibt. Zurück kommen die Daten der Mitarbeiter, welche ein Arbeitsdatum besitzen, das vor dem ausgewählten Datum liegt.
In einem Suchfeld kann, wie in jeder anderen Detail-Page der Projektauswertung, innerhalb der dargestellten Tabelle gesucht werden. Das Merkmal, nach dem in der Tabelle gesucht werden soll, kann in der Toolbar durch zwei RadioButtons ausgewählt werden.

Zur Verfügung steht dabei die Suche nach dem Mitarbeiterkürzel und dem Nachnamen. Initial ist die Suche nach dem Kürzel voreingestellt. Wird die Auswahl anhand der Buttons geändert, wird die Funktion „selectedSearchContent()“ ausgeführt (vgl. Listing 6.49). Dieser Funktion wird als Parameter der Index des Buttons übergeben. Der Index wird dem entsprechendem Merkmal zugeordnet und in der Variable „searchContent“ gespeichert. Je nach Auswahl wird auch der Platzhalter des SearchFields entsprechend angepasst.

```
/**
 * Setzt den searchContent (Content, nach dem in der Ansicht "verspaetete Timesheets" gesucht werden soll)
 * auf den ausgewaehlten Wert
 * index - uebergibt den index des RadioButtons, 0 steht fuer Token (kuerzel), 1 steht fuer Surname (Nachname des Mitarbeiters)
 */
    selectedSearchContent: function(index){
            if(index==0){
              this.searchContent = "TOKEN";
              sap.ui.getCore().byId("workLastSearch").setPlaceholder("filtern nach Kuerzel...");
            }else{
             this.searchContent = "SURNAME";
             sap.ui.getCore().byId("workLastSearch").setPlaceholder("filtern nach Nachname...");
             }
      },
```

Listing 6.49: Projektcontrolling: verspätete Timesheets selectedSearchContent()

6.3.6. UI Dateneingabe

Folgender Abschnitt beschreibt die grafische Applikation der Dateneingabe. Zur Erfassung der benötigten Daten in der SAP HANA-Datenbank wurde ein SAPUI5 Projekt erstellt. Die Applikation wird genutzt um Stammdaten und Planwerte eines Projektes zu erfassen. Die UI der Dateneingabe befindet sich im Paket: „abat/demosystem/projektcontrolling/controlling/ui/ui_data“. Abbildung 6.28 auf der nächsten Seite zeigt das SAPUI5 Projekt der gesamten Dateneingabe.

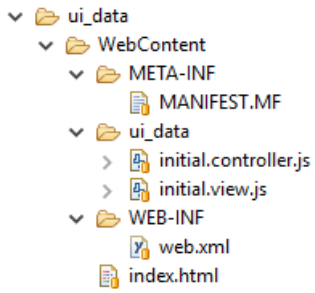


Abbildung 6.28.: Projektcontrolling: Dateneingabe Package

Master- und Detail-Seiten Wie in Abbildung 6.13 auf Seite 238 ist auch diese UI in Master- und Detail-Seiten unterteilt. Eine detaillierte Übersicht ist Abbildung 6.14 zu entnehmen. Alle Controls müssen, wie in Listing 6.50 zu sehen ist, dem SplitApp-Control hinzugefügt werden, welches dann letzten Endes zur Anzeige der Anwendung zurückgegeben wird.

```
/**
* SPLIT APP
* oSplitApp = Erzeugen eines SplitApp-Objektes zur initialisierung der
    Master und Detail Seiten. Hier wird festgelegt welche Seite Master,
    welche Detail und welche zu erst angezeigt wird.
*/
  var oSplitApp = new sap.m.SplitApp("splitApp1",{
    showNavButton: true,
  });
  oSplitApp.addMasterPage(oMasterPage);
  oSplitApp.addDetailPage(oDetailsPageSkillEmp1);
  oSplitApp.addDetailPage(oDetailsPageHour1);
  oSplitApp.addDetailPage(oDetailsPageProject1);
  oSplitApp.addDetailPage(oDetailsPagePsp1);
  oSplitApp.addDetailPage(oDetailsPagePlanPsp1);
  oSplitApp.addDetailPage(oDetailsPageSkillEmp2);
  oSplitApp.addDetailPage(oDetailsPageHour2);
  oSplitApp.addDetailPage(oDetailsPageProject2);
  oSplitApp.addDetailPage(oDetailsPagePsp2);
  oSplitApp.addDetailPage(oDetailsPagePlanPsp2);
  oSplitApp.setInitialMaster("master1");
  oSplitApp.setInitialDetail("navProject1");

  return oSplitApp;
}
```

Listing 6.50: Projektcontrolling: Dateneingabe Master - Detail - Pages

Eine detaillierte Beschreibung zum Thema SplitApp-Control ist im vorherigen Abschnitt zur Datenauswertung (vgl. Abschnitt 6.3.5 auf Seite 240) sowie in der grafischen Oberfläche von Szenario Sensorik (vgl. Abschnitt 5.3.4 auf Seite 131) zu finden.

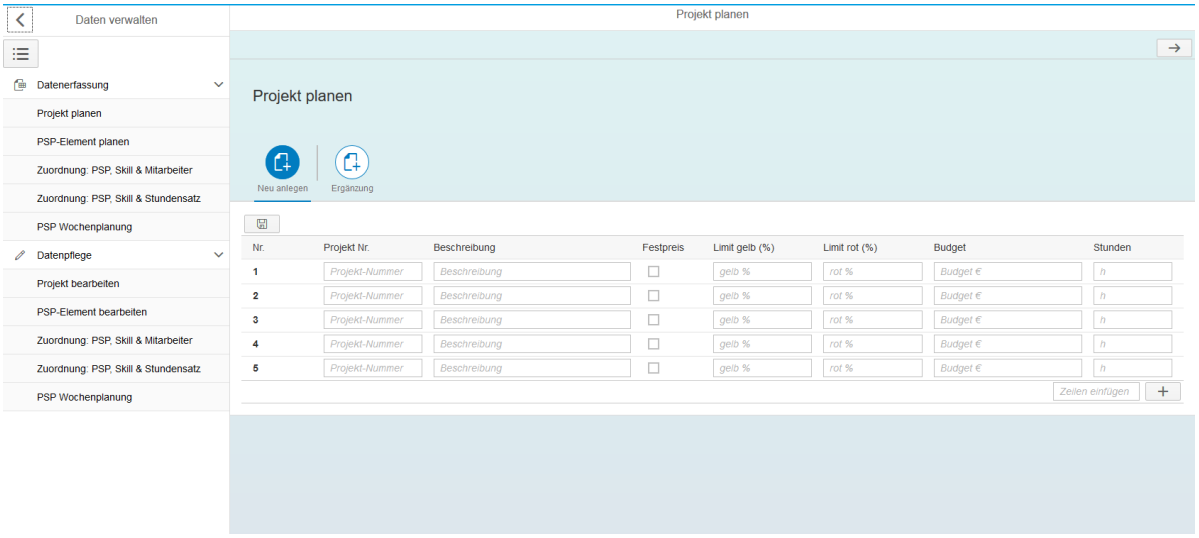


Abbildung 6.29.: Projektcontrolling: Dateneingabe

Abbildung 6.29 zeigt die Startansicht der Eingabe UI. Unterschieden wird dabei zwischen Dateneingabe und Datenpflege. In den Unterpunkten der Dateneingabe kann der User neue Datensätze speichern. In der Datenpflege können Datensätze aktualisiert oder gelöscht werden. Zur Veranschaulichung wird das Beispiel „Projekt planen“ herangezogen. Die weiteren vier Seiten sind fast identisch aufgebaut, daher werden nur die Abweichungen zur vorgestellten Seite erläutert.

Master-Page Wie auch in der Report UI teilt sich die App in zwei Teile. Die Master-Page ist bei der Dateneingabe ausschließlich für die Navigation innerhalb der App und zur Navigation zurück zur Startseite vorhanden.

Navigation In der Master-Page wurde eine Seitennavigation implementiert. Navigiert werden kann zum einen über die Navigationsliste (vgl. Abbildung 6.30 auf der nächsten Seite). Zum anderen kann auch über Buttons, welche sich oben rechts in einer Toolbar befinden, innerhalb der Detail-Seiten navigiert werden.

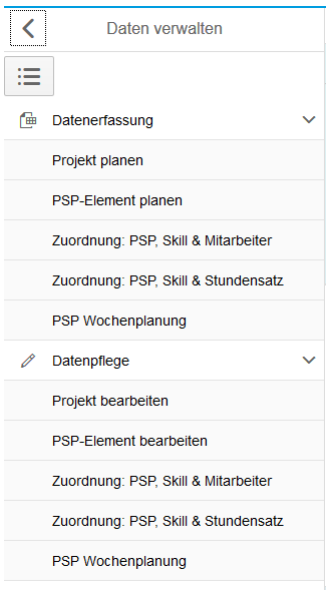


Abbildung 6.30.: Projektcontrolling: Dateneingabe Navigation Liste

„sap.tnt.NavigationList“ ermöglicht eine Baumstruktur in der Navigation. Mit „sap.tnt.NavigationListItem“ können beliebig viele Elemente hinzugefügt werden. Die Navigation erfolgt über einen Funktionsaufruf aus dem Controller mit dem Event „select“. Die gesamte Liste kann über den oberen Button mit Listen-Icon auf und zu geklappt werden. Die einzelnen Wurzeln können ebenfalls auf und zu geklappt werden. Exemplarisch wird im untenstehenden Listing 6.51 gezeigt, wie eine solche Liste initialisiert wird und mit Elementen bestückt wird. In Zeile 12 ist der Button zu sehen, welcher die gesamte Liste schließt und öffnet.

```
1  /**
2  * MASTER SEITE
3  * Erzeugen einer Seitennavigation zur Navigation zwischen den Detail-
       Seiten.
4  * sap.tnt.NavigationList erzeugt eine Baumstruktur in der Navigation.
5  * sap.tnt.NavigationListItem erzeugen Element
6  * Die Navigation erfolgt mit einem Funktionsaufruf aus dem Controller
       unter select: function
7  * Variablen:
8  * navButton = Die gesamte Liste kann auf und zu geklappt werden.
9  * navlist = neue Navigationsliste initialisieren - sap.tnt.NavigationList.
10 */
11
12   var navButton = new sap.m.Button({
13     width: '50px',
14     icon: 'sap-icon://menu',
15     press: function () {
16     navlist.setExpanded(!navlist.getExpanded());
17     }
18   });
19
```

```
20   var navlist = new sap.tnt.NavigationList({
21       expanded: true,
22       id: "listId",
23       mode: sap.m.ListMode.SingleSelectMaster,
24       items: [
25       new sap.tnt.NavigationListItem('navtest',{
26       text: 'Datenerfassung',
27       expanded: true,
28       icon: 'sap-icon://excel-attachment',
29       items: [
30       new sap.tnt.NavigationListItem('navProject1ID',{
31       text: 'Projekt planen',
32       key: "navProject1",
33       select: function() {
34       oController.onPressProject();
35   })),
```

Listing 6.51: Projektcontrolling: Dateneingabe Navigation Liste

Mit Hilfe der Buttons „zurück“ und „weiter“ kann zwischen den Eingabeoberflächen gewechselt werden (vgl. Abbildung 6.31).

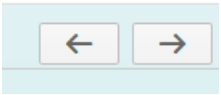


Abbildung 6.31.: Projektcontrolling: Dateneingabe Navigation Buttons

Wie in Listing 6.52 zu sehen ist, befinden sich die Buttons für die Navigation in einer Toolbar. Es werden hier dieselben Funktionen wie für die Navigationsliste aufgerufen.

```
1  /**
2  * DETAIL PAGE 2 - PSP
3  * oToolbarPsp
4  */
5    var oToolbarPsp = new sap.m.Toolbar({
6      content : [ new sap.m.ToolbarSpacer({}),
7        new sap.m.Button({
8        tooltip: "Zurueck",
9        icon: 'sap-icon://arrow-left',
10       width : '50px',
11       press : function() {
12       oController.onPressProject();
13       }
14       }),new sap.m.Button({
15       tooltip: "Weiter",
16       icon: 'sap-icon://arrow-right',
17       width : '50px',
18       press : function() {
19       oController.onPressEmp();
20   }
```

Listing 6.52: Projektcontrolling: Dateneingabe Navigation Buttons

Mit den folgenden Funktionen wurde die Navigation implementiert:

- onPressProject()
- onPressPSP()
- onPressEmp()
- onPressHour()
- onPressPlanPSP()
- onPressProject2()
- onPressPSP2()
- onPressEmp2()
- onPressHour2()
- onPressPlanPSP2()

Zu sehen ist hier die Funktion für die Navigation zur Projekt-Eingabe-Seite (vgl. Listing 6.53):

```
1 /**onPressProject
2 */
3   onPressProject : function (oEvent) {
4     var app = sap.ui.getCore().byId("splitApp1");
5     this.getModelMVProject();
6     app.toDetail('navProject1','show')
7   },
```

Listing 6.53: Projektcontrolling: Dateneingabe Navigation Projekt

Über den Befehl „toDetail“ kann zu einer Detail-Page navigiert werden: “app.toDetail('navProject1', 'show')“. Um in der Ergänzungs-Ansicht aktuelle Daten zu erhalten, wird die Funktion „getModelMVProject()“ aufgerufen (vgl. Listing 6.53 Zeile 5). Hierbei wird ein Service aufgerufen, welcher die aktuellen Daten lädt.

Detail-Pages Der User kann die Seiten unabhängig voneinander Nutzen, kann aber beispielsweise kein PSP-Element anlegen ohne vorher ein übergeordnetes Projekt angelegt zu haben. Im ersten Schritt muss ein Projekt vorhanden sein oder neu geplant werden. Anschließend können dazu beliebig viele PSP-Elemente geplant werden. Der User kann mit einem vorhandenen PSP-Element die Zuordnungen von PSP-Skill-Mitarbeiter und PSP-Skill- Stundensatz vornehmen. Bei Bedarf kann der User eine Wochenplanung für das PSP-Element durchführen.

Folgende Daten können auf den jeweiligen Detail-Seiten bearbeitet werden:

- Datenerfassung: Speichern von neuen Datensätzen
  - Projekt planen:
    Projekt-Nr., Beschreibung. Festpreis, Limit gelb, Limit rot, Budget, Stunden
  - PSP-Element planen:
    Projekt-Nr., PSP-Nr., Rechnungs-Nr., Beschreibung, Limit gelb, Limit rot, Budget, Stunden
  - Zuordnung: PSP, Skill, Mitarbeiter
    PSP, Skill, Mitarbeiter
  - Zuordnung: PSP, Skill, Stundensatz
    PSP, Skill, Stundensatz
  - PSP Wochenplanung:
    PSP-Nr., Skill Level, Stunden, Woche, Jahr
- Datenpflege: Aktualisieren und Löschen von bestehenden Datensätzen
  - Projekt bearbeiten:
    Projekt-Nr., Beschreibung. Festpreis, Limit gelb, Limit rot, Budget, Stunden
  - PSP-Element bearbeiten:
    Projekt-Nr., PSP-Nr., Rechnungs-Nr., Beschreibung, Limit gelb, Limit rot, Budget, Stunden, Sperren
  - Zuordnung: PSP, Skill, Mitarbeiter
    PSP, Skill, Mitarbeiter
  - Zuordnung: PSP, Skill, Stundensatz
    PSP, Skill, Stundensatz
  - PSP Wochenplanung:
    PSP-Nr., Skill Level, Stunden, Woche, Jahr

Models Models werden benötigt, um Tabellen mit Inhalten oder leeren Zeilen in der UI zu erzeugen. Es werden diverse Services (vgl. Abschnitt 6.3.2 auf Seite 221) und damit Daten aus der Datenbank oder Prozeduren in der UI aufgerufen und an Tabellen gebunden oder in Funktionen genutzt. Tabelle 6.10 auf Seite 283 gibt einen Überblick über alle verwendeten Models. Welche Controls, welche Services und Models nutzen, wird im jeweiligen Dokumentationsabschnitt erläutert. Alle genutzten Services sind im Abschnitt „Services“ 6.3.2 auf Seite 221 beschrieben und befinden sich im Paket: abat/-demosystem/projektcontrolling/controlling/services.

Tabelle 6.10.: Projektcontrolling: Models Dateneingabe

Paket:	abat/demosystem/projektcontrolling/controlling/ui/ui_data		
Datei:	initial.controller.js		
	Variable:	**Control:**	**Zweck:**
Neu anlegen:			Tabelle mit leeren Zeilen wird erzeugt und per Button-Klick können leere Zeilen hinzugefügt werden.
	projektModel	JSON-Model	Tabelle erzeugen
	pspModel	JSON-Model	Tabelle erzeugen
	skillPspEmployeeModel	JSON-Model	Tabelle erzeugen
	hourModel	JSON-Model	Tabelle erzeugen
	planPspData	JSON-Model	Tabelle erzeugen
Ergänzung:			JSON-Model, welches Werte für die Tabellen der Wertergänzung anzeigt
	projectData-MissingValueModel	JSON-Model	Projekt-Nr. und zu ergänzende Felder

Tabelle 6.10.: Projektcontrolling: Models Dateneingabe

Paket:	abat/demosystem/projektcontrolling/controlling/ui/ui_data		
Datei:	initial.controller.js		
	Variable:	**Control:**	**Zweck:**
	pspData-MissingValueModel	JSON-Model	Projekt-Nr., PSP-Nr. und zu ergänzende Felder
	skillPspEmployee-MissingValueModel	JSON-Model	Skill Level muss ergänzt werden
	skillPspRate-MissingValueModel	JSON-Model	Stundensatz muss ergänzt werden
Update/Delete:			JSON-Model, welches die Tabellen erzeugt und mit Werten füllt
	projectDataModel	JSON-Model	
	pspDataModel	JSON-Model	
	skillPspMaModel	JSON-Model	
	skillPspRateModel	JSON-Model	
	planValuesDataModel	JSON-Model	
Sugggest:			Ein Service für alle Models, Wertvorschläge werden so ermöglicht
	suggestionLevelModel	JSON-Model	Die Skill-Level Daten sind fest in einem JSON-String gesetzt. Dieser String wird dann an ein JSON-Model gebunden.

Tabelle 6.10.: Projektcontrolling: Models Dateneingabe

Paket:	abat/demosystem/projektcontrolling/controlling/ui/ui_data		
Datei:	initial.controller.js		
	Variable:	**Control:**	**Zweck:**
	suggestion-EmployeeModel	JSON-Model	Mitarbeiter Name
	suggestion-PspModel	JSON-Model	PSP-Nr.
	suggestion-ProjectModel	JSON-Model	Projekt-Nr.

Tabellen Alle Tabellen der Dateneingabe und -pflege werden in der „initial.view.js“ initialisiert und über die Datenbindung an eines der Models, welche in der Klasse „initial.controller.js“ erstellt werden, erzeugt. Bis auf die leeren Tabellen, werden alle weiteren Tabellen über Service-Aufrufe mit Werten aus der Datenbank erstellt.

Dateneingabe Die UI wurde in zwei Ansichten unterteilt - „Neu anlegen“ und „Ergänzung“ (vgl. Abbildung 6.32). Die Navigation zwischen den beiden Seiten erfolgt über eine „sap.m.IconTabBar“ (vgl. Listing 6.54). Per Klick auf einen der beiden „sap.m.IconTabFilter“ kann der User zwischen den beiden Optionen wechseln. Im jeweiligen „sap.m.IconTabFilter“ wird auch der Content (Zeile 10) gesetzt, welcher angezeigt werden soll.

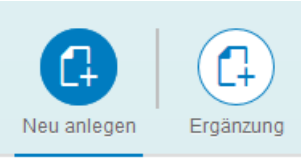


Abbildung 6.32.: Projektcontrolling: Dateneingabe zwei Ansichten

```
1 /**
2 * DETAIL PAGE 1 - PROJECT
3 * oTabBarProject1 = IconBar zum wechseln zwischen den Eingaben "Neu
    anlegen" und "Ergaenzung".
4 */
5   var oTabBarProject1 = new sap.m.IconTabBar({
6     expanded: true,
7     items : [
```

```
8       new sap.m.IconTabFilter({
9       text : "Neu anlegen",
10      icon : "sap-icon://add-document",
11      content: [oToolbarProject2, oTableProject, oToolbarProjectRows],
12      iconColor: "Neutral"
13      }),new sap.m.IconTabSeparator({
14      }),
15      new sap.m.IconTabFilter({
16      text : "Ergaenzung",
17      icon : "sap-icon://add-document",
18      content: [oTableMVProject],
19      iconColor: "Neutral"
20  }),
```

Listing 6.54: Projektcontrolling: Dateneingabe IconTabBar

UI Neu anlegen Abbildung 6.33 zeigt die Massendatenerfassung. Genauer kann hier die Seite zum „Projekt planen“ betrachtet werden. Hier können beliebig viele neue Datensätze gleichzeitig gespeichert werden. Der User kann mehrere Projekte in eine Tabelle eintragen und gleichzeitig speichern. Der Button zum Speichern mit einem Disketten-Icon befindet sich links über der Tabelle. Zudem kann der User weitere leere Zeilen hinzufügen. Oben rechts befindet sich der Pfeil zur Navigation auf die nächste Seite „PSP-Element planen“. In die Datenbank werden nur vollständige Datensätze geschrieben.

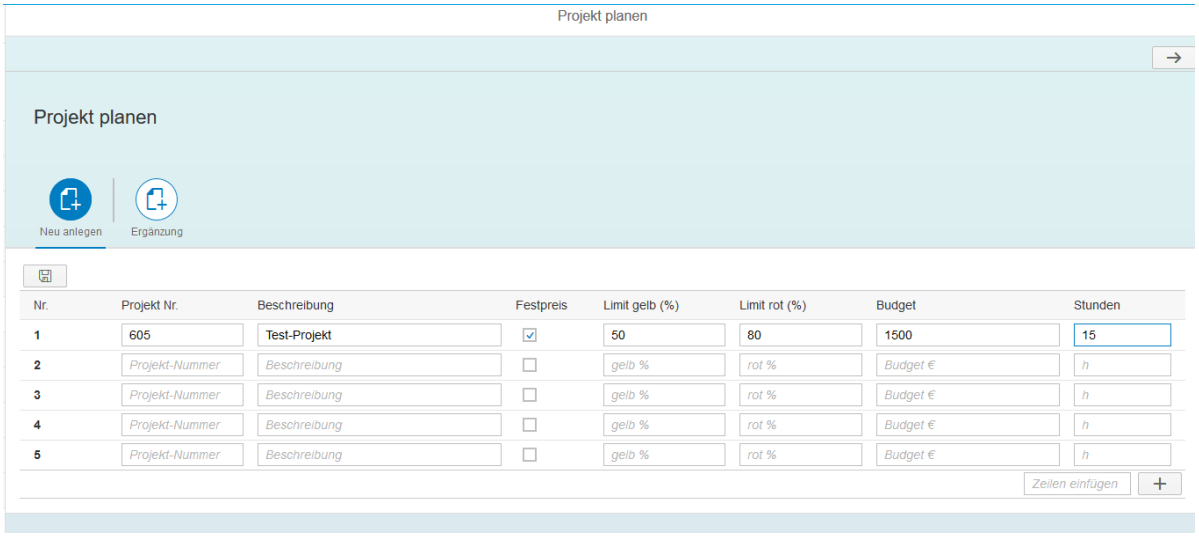


Abbildung 6.33.: Projektcontrolling: Dateneingabe Projekt neu anlegen

Tabellen mit leeren Zeilen Zum Erzeugen der Tabellen mit leeren Zeilen, werden die in Tabelle 6.10 auf der vorherigen Seite aufgelisteten Models der Kategorie „Neu anlegen“ genutzt. Um leere Zeilen erzeugen zu können, müssen die Daten manuell in Form eines JSON erstellt werden und über ein Model an die Tabelle gebunden werden. Initial

werden immer fünf leere Zeilen erzeugt. Listing 6.55 zeigt wie das JSON in eine Variable übergeben und damit anschließend ein JSON-Model (Zeile 14) erstellt wird. Das „B“ im Tabellenwert „SCORE_VALUE“ wird immer übergeben und in der UI unsichtbar. Hier wird bestimmt, dass die Auswertung nach Budget und nicht nach Stunden ausgewertet wird. Einstellungen dafür können in der Report-UI vorgenommen werden.

```
1  /**
2  * Model PAGE 1 Projekt
3  * projektModel: Tabelle wird erzeugt und per Button-Klick koennen leere
       Zeilen hinzugefuegt werden.
4  */
5    var pData = {
6      peditData: [
7        {NR: "1", PROJECTNUMBER: "", BUDGET: "", HOURS: "", FIXED_VALUES: "
       ", DESCRIPTION: "", LIMIT_YELLOW: "", LIMIT_RED: "", SCORE_VALUE: "B"},
8        {NR: "2", PROJECTNUMBER: "", BUDGET: "", HOURS: "", FIXED_VALUES: "
       ", DESCRIPTION: "", LIMIT_YELLOW: "", LIMIT_RED: "", SCORE_VALUE: "B"},
9        {NR: "3", PROJECTNUMBER: "", BUDGET: "", HOURS: "", FIXED_VALUES: "
       ", DESCRIPTION: "", LIMIT_YELLOW: "", LIMIT_RED: "", SCORE_VALUE: "B"},
10       {NR: "4", PROJECTNUMBER: "", BUDGET: "", HOURS: "", FIXED_VALUES: "
       ", DESCRIPTION: "", LIMIT_YELLOW: "", LIMIT_RED: "", SCORE_VALUE: "B"},
11       {NR: "5", PROJECTNUMBER: "", BUDGET: "", HOURS: "", FIXED_VALUES: "
       ", DESCRIPTION: "", LIMIT_YELLOW: "", LIMIT_RED: "", SCORE_VALUE: "B"}
12     ]
13   };
14   var projektModel = new sap.ui.model.json.JSONModel(pData); // created a
       JSON model
15   sap.ui.getCore().setModel(projektModel, "projektModel");
```

Listing 6.55: Projektcontrolling: Dateneingabe Model leere Tabelle

Datenbindung Tabellen Die Datenbindung wird in der View bei der Initialisierung der Tabellen durchgeführt. Listing 6.56 zeigt als Beispiel für Datenbindung die Tabelle ‘oTableProject“ aus der UI „Projekt planen“. Der Pfad muss festgelegt (Zeile 2) und die einzelnen Felder entsprechend zugewiesen werden (Zeile 5).

```
1  oTableProject.bindItems({
2    path: "projektModel>/peditData",
3    template: new sap.m.ColumnListItem({
4      cells:[new sap.m.ObjectIdentifier({
5      title:"{projektModel>NR}",
6      value:"{projektModel>NR}",
7      }),new sap.m.Input({
8      title:"{projektModel>PROJECTNUMBER}",
9      placeholder: "Projekt-Nummer",
10     value: "{projektModel>PROJECTNUMBER}",
11 }),
```

Listing 6.56: Projektcontrolling: Binding Tabelle

Suggestions Eine Besonderheit waren die Input-Felder, welche einen Wert wie z.B. einen Mitarbeiternamen vorschlagen sollen. Dafür wurden die oben aufgeführten Suggest Models erstellt (vgl. Tabelle 6.10 auf Seite 283). Diese Models rufen den selben Service auf, um die Werte abrufen zu können. Sollen Projektnummern, PSP-Nummern oder Mitarbeiternamen vorgeschlagen werden, wird der Service „getSuggestionsValues“ aufgerufen und das jeweilige Model gebunden (vgl. Listing 6.57).

```
/**
* Sugggest Model - PROJECT
* Die Projekt-Nr. werden ueber einen Service aufgerufen.
*/
  var suggestionProjectModel = new sap.ui.model.json.JSONModel("http
    ://192.168.252.42:8002/abat/demosystem/projektcontrolling/controlling/
    services/getSuggestionsValues.xsjs?ITEM=project"); // created a JSON
    model
  sap.ui.getCore().setModel(suggestionProjectModel, "
    suggestionProjectModel");
```

Listing 6.57: Projektcontrolling: Suggest Model Projekt

Mit dem Model kann ein Template erstellt werden, welches an verschiedenen Stellen eingesetzt werden kann, um Wertvorschläge bereit zu stellen (vgl. Listing 6.58).

```
  var oSuggestionProject = new sap.ui.core.ListItem("items4",{
    text:'{suggestionProjectModel>SUGGESTION_ITEM}'
  });
```

Listing 6.58: Projektcontrolling: Suggest Template Projekt

Das Template wird an ein Input-Feld in der UI gebunden (vgl. Listing 6.59). Im Folgenden Beispiel können so Projektnummern in der Eingabetabelle von PSP-Elementen vorgeschlagen werden, sofern diese bereits in der Datenbank sind. Dafür muss bei einem „sap.m.Input“-Feld **showSuggestion** auf **true** gesetzt werden. Ab Zeile 6 ist zu sehen wie das Template an das Feld gebunden wird.

```
  new sap.m.Input({
    value:"{pspModel>PROJECTNUMBER}",
    placeholder: "Projekt-Nr.",
    showSuggestion:true,
    suggestionItems:{
    path: "suggestionProjectModel>/values",
    template: oSuggestionProject,
    templateShareable:true,}
  }),
```

Listing 6.59: Projektcontrolling: Binding Suggest Input Projekt

Zeilen hinzufügen Für die fünf Seiten der Massendatenerfassung wurden zum Hinzufügen von leeren Zeilen folgende Funktionen implementiert:

- addRows1()

- addRows2()
- addRows3()
- addRows4()
- addRows5()

Das in Listing 6.55 auf Seite 285 erstellte JSON-Model „projektModel“ wird hier genutzt, um weitere Zeilen zu erzeugen. Wie in Listing 6.60 in Zeile 24 zu sehen ist, nutzt die Funktion dieses Model.
Der Button dafür befindet sich inklusive eines Input-Felds in einer Toolbar, welche unter der Tabelle ist (vgl. Abbildung 6.34). Klickt der User einmal auf den Button wird eine Zeile eingefügt. Wird eine Zahl in das Input-Feld getippt und anschließend der Button gedrückt, werden mehrere Zeilen auf einmal hinzugefügt.

Abbildung 6.34.: Projektcontrolling: Dateneingabe Projekt Zeilen

```
1  /**
2  * FUNCTIONS - ADD ROWS
3  * Zum Erzeugen von leeren Zeilen in der Massendateneingabe
4  * Variable:
5  * inputZeile  ruft per ID - sap.ui.getCore().byId() -das jeweilige Input
       Felde auf.
6  * input     liefert den zuletzt eingetippten Wert des Input Feldes:
       inputZeile._lastValue.
7  * oModel    liefert das jeweilige Model aus der init funciton.
8  * modelData   liefert die jeweiligen Werte des Models.
9  * numberOfRows zaehlt die Zeilen hoch und nummeriert diese so automatisch.
10 *  modelData.push ({}) erzeugt einen neuen Wert im Model und somit eine
       neue leere Tabellenzeile.
11 * Anschliessend wird der Wert im Model gesetzt.
12 * Wird nur der Button geklickt wird genau eine Zeile hinzugefuegt.
13 * Wird eine Zahl eingegeben und auf den Button geklickt wird eine while-
       Schleife
14 * so lange ausgefuehrt bis die angegebenen Zeilen-Zahl erreicht wird.
15 */
16
17 /**
18 * FUNCTIONS - ADD ROWS - PROJECT
19 * Besonders ist hier, dass SCORE_VALUE: "B" immer mit einem B uebergeben
       wird. So wird in der spaeteren Auswertung gewaehrleistet, dass diese
       per Budget erfolgt.
20 */
21   addRows1: function () {
22     var inputZeile = sap.ui.getCore().byId("rowsProject");
23     var input = inputZeile._lastValue;
```

```
24      var oModel = sap.ui.getCore().getModel('projektModel');
25      var modelData = oModel.getProperty("/peditData");
26      var numberOfRows = sap.ui.getCore().getModel('projektModel').
        getProperty("/peditData/length")+1;
27
28      if(input <= 0){
29        modelData.push({NR: "" + numberOfRows, PROJECTNUMBER: "", BUDGET: ""
        ,    HOURS: "", FIXED_VALUES: "", DESCRIPTION: "", LIMIT_YELLOW: "",
        LIMIT_RED: "", SCORE_VALUE: "B"});
30        oModel.setProperty("/peditData", modelData);
31        input++;
32
33      }else{
34      while(input>0){
35        modelData.push({NR: "" + numberOfRows, PROJECTNUMBER: "", BUDGET: ""
        ,    HOURS: "", FIXED_VALUES: "", DESCRIPTION: "", LIMIT_YELLOW: "",
        LIMIT_RED: "", SCORE_VALUE: "B"});
36        oModel.setProperty("/peditData", modelData);
37        input--;
38        numberOfRows++;}
39      }
40    },
```

Listing 6.60: Projektcontrolling: Funktion Zeilen hinzufügen

UI Neu anlegen - Insert Zum Speichern der Massendaten wurden folgende Funktionen implementiert:

- saveDataProject1()
- saveDataPsp1()
- saveDataskillPspEmployee()
- saveDataskillPspRates()
- saveDataplanPsp()

Die in Abbildung 6.35 zu sehenden Services werden in diesem Abschnitt genutzt.

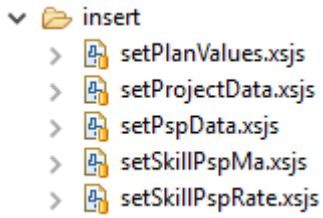


Abbildung 6.35.: Projektcontrolling: Dateneingabe Insert Services

Folgende Listings - 6.61, 6.62, 6.63, 6.64 - zeigen die Insert Funktion „saveDataProject1“. Es muss ermittelt werden, wie viele Zeilen zu speichern sind (vgl. Listing 6.61 Zeile 7). Anschließend werden alle nötigen Variablen deklariert (Zeile 8 ff.). Dazu gehören Variablen, welche zum Speichern der Werte sind, Hilfs-Variable für die Durchführung der Schleife und Variable für die Fehleranalyse.

```
1  /**
2  * FUNCTIONS - INSERTS:
3  * saveDataProject1
4  */
5
6    saveDataProject1: function (oEvent) {
7    var numberOfRows = sap.ui.getCore().getModel('projektModel').getProperty
       ("/peditData/length")
8    var pNumber;
9    var budget;
10   var hours;
11   var fixedValues;
12   var description;
13   var limitYellow;
14   var limitRed;
15   var scoreValue;
16
17   var insertResponse;
18   var insertError;
19   var wrongRows = "";
20   var insertErrorStatus;
21   var i = 0;
22   var saveModel;
```

Listing 6.61: Projektcontrolling: Funktion Insert - Projekt

Listing 6.62 zeigt einen Auszug aus der Funktion zum Speichern von Projektdaten. Die while-Schleife wird so lange durchlaufen bis alle Zeilen gespeichert wurden (Zeile 1). Es wird immer geprüft, ob alle Muss-Felder ausgefüllt sind (Zeile 14). Ist dies der Fall, wird über einen AJAX Aufruf der entsprechende Service aufgerufen und die Daten werden in die Datenbank geschrieben. Ab Zeile 23 ist zu sehen, wie mit dem Parameter „response“ der Erfolg der Speicherung überprüft wird. Es wird geprüft, ob die Daten in der Datenbank ankommen sind und in einem Dialog-Fenster eine entsprechende Text-Nachricht in der UI ausgegeben.

```
1    while (i< numberOfRows){
2
3      saveModel = sap.ui.getCore().getModel('projektModel').getProperty("/
       peditData/"+i);
4
5      var pNumber = saveModel.PROJECTNUMBER;
6      budget = saveModel.BUDGET;
7      hours = saveModel.HOURS;
8      fixedValues = saveModel.FIXED_VALUES;
9      description = saveModel.DESCRIPTION;
10     limitYellow = saveModel.LIMIT_YELLOW;
```

```
11      limitRed = saveModel.LIMIT_RED;
12      scoreValue = saveModel.SCORE_VALUE;
13
14      if (pNumber != "" && budget != "" && hours !=  ""&& description !=  ""
    && limitYellow !=  ""&& limitRed !=  ""){
15
16         var self = this
17         $.ajax({
18         url: "http://192.168.252.42:8002/abat/demosystem/projektcontrolling/
    controlling/services/insert/setProjectData.xsjs?PROJECTNUMBER="+pNumber
    +"&BUDGET="+budget+"&HOURS="+hours+"&FIXEDVALUES="+fixedValues+"&
    DESCRIPTION="+description+"&LIMITYELLOW="+limitYellow+"&LIMITRED="+
    limitRed+"&SCORE_VALUE="+scoreValue,
19         async:false,
20         type:"GET",
21         dataType:"json",
22         cache:"false",
23         success: function(response){
24
25         self.refreshProjectData();
26         self.refreshSkillPSPMa();
27         self.refreshPSPData();
28         self.refreshSkillPSPRate();
29
30         insertResponse = response.values[0].ROW_COUNT;
31
32         if(insertResponse >= 1){
33         wrongRows += i+1+ " ";
34         insertError= "Speichervorgang war erfolgreich";
35         insertErrorStatus = "Folgende Zeilen wurden gespeichert: " +
    wrongRows;
36         }else{
37         wrongRows += i+1+ " ";
38         insertError = "Beim Speichervorgang ist ein Fehler aufgetreten!";
39         insertErrorStatus = "Beim Speichervorgang ist ein Fehler aufgetreten
    !  Folgende Zeile(n): " + wrongRows;
40         }
41         },
42         complete: function(){},
43         error:function(xhr,status,error){
44         wrongRows += i+1+ " ";
45         insertError = "Beim Speichervorgang ist ein Fehler aufgetreten!";
46         insertErrorStatus = "Beim Speichervorgang ist ein Fehler aufgetreten
    !  Folgende Zeile(n): " + wrongRows;
47         }
48         });
49         }
50      i++;
51      }
52   }
```

Listing 6.62: Projektcontrolling: Dateneingabe Vollständigkeit der Eingabe

```
if(pNumber == "" && budget != "" && hours !=  ""&& description !=  ""&&
 limitYellow !=  ""&& limitRed !=  ""){
  wrongRows += i+1+ " ";
  insertErrorStatus = "Folgende Zeilen sind unvollstaendig: " +
 wrongRows;
}
```

Listing 6.63: Projektcontrolling: Dateneingabe unvollständig

Fehlt eine einzelne Eingabe, wird der User über die „sap.m.MessageBox“ darauf hingewiesen. Anschließend werden mit einer IF-Bedingung die Felder einzeln geprüft und der Status für die MessageBox „insertErrorStatus“ festgelegt. Dabei wird dem User die Zeile ausgegeben, welche nicht vollständig ist (vgl. Listing 6.63).

Wird das Projekt erfolgreich in die Datenbank geschrieben, erhält der User eine Textnachricht mit Hilfe einer „sap.m.MessageBox“ (vgl. Abbildung 6.36). Tritt ein Fehler auf, wird eine Fehlermeldung ausgegeben. Die MessageBox ist ebenfalls Bestandteil der Funktion aus Listing 6.61 auf Seite 289.

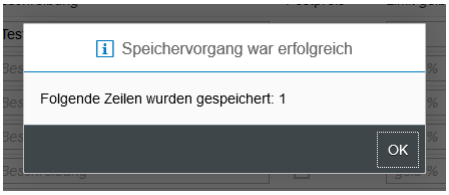


Abbildung 6.36.: Projektcontrolling: Dateneingabe Projekt neu anlegen MessageBox

Die „sap.m.MessageBox“ ist in der jeweiligen Speicher-Funktion implementiert und in Listing 6.64 dargestellt.

```
sap.m.MessageBox.show(insertErrorStatus, {
  icon: sap.m.MessageBox.Icon.INFORMATION,
  title: insertError,
  actions: [sap.m.MessageBox.Action.OK],
  onClose: function(oAction) {
  }.bind(this)
});
```

Listing 6.64: Projektcontrolling: Dateneingabe IconTabBar

Je Oberfläche werden verschiedene Nachrichten über die MessageBox ausgegeben. Der Titel der Box ist „insertError“ und der Text in der Box ist „insertErrorStatus“. Bei den Massendaten spielt auch die Nummer der Zeilen eine Rolle. Die MessageBox kann folgende Nachrichten ausgeben:

- Massendaten
  - Titel: „Speichervorgang war erfolgreich“
    Inhalt: „Folgende Zeilen wurden gespeichert: “ + Zeile(n).

  - Titel: „Beim Speichervorgang ist ein Fehler aufgetreten!“
    Inhalt: „Beim Speichervorgang ist ein Fehler aufgetreten! Folgende Zeile(n): “ + Zeile(n).;
  - Titel: “Beim Speichervorgang ist ein Fehler aufgetreten!“
    Inhalt: „Beim Speichervorgang ist ein Fehler aufgetreten! Folgende Zeile(n): “
  - Inhalt: „Folgende Zeilen sind unvollständig:“ + Zeile(n).

- Wertergänzung
  - Titel: „Speichervorgang war erfolgreich.“
  - Titel: „Beim Speichervorgang ist ein Fehler aufgetreten!“
  - Titel: „Angaben unvollständig!“

- Aktualisieren
  - Titel: Update-Status
  - Inhalt: „Das Projekt „Projekt-Nr“. wurde erfolgreich aktualisiert.“
  - Inhalt: „Update war nicht erfolgreich.“

- Löschen
  - Titel: Delete-Status
  - Inhalt: „Projekt „Projekt-Nr.“ wurde entfernt.“
  - Inhalt: „Löschen war nicht erfolgreich.“

CheckBox Für die Spalte Festpreis wurde eine „sap.m.CheckBox“ gewählt (vgl. Abbildung 6.33 auf Seite 284). Es kann ein Haken bei Festpreis-Projekten gesetzt werden. Es wird also festgelegt, ob ein Projekt fix ist oder nicht. Die Datenbindung sowie die Übergabe der Daten unterschied sich bei der Implementierung von den anderen UI-Elementen. Hierfür werden zwei Funktionen (vgl. Listing 6.65 und 6.66 auf der nächsten Seite) aufgerufen, um den gesetzten oder nicht gesetzten Wert zu übergeben. „getFixedValue“ setzt den Wert auf true oder false, „setFixedValuesInsert“ trägt den passenden Wert für die Speicherung in der Datenbank ein.

```
1  getFixedValue: function(item){
2    if (item =="X"){
3      return true;
4    }else
5      return false;
6  },
```

Listing 6.65: Projektcontrolling: Dateneingabe Binding CheckBox: getFixedValue

```
setFixedValuesInsert: function(oEvent){
  var fixedValue = oEvent.getParameters().selected;
  var fixedValueString;
  var sPathofTable = oEvent.getSource().getBindingContext('projektModel').sPath;
  sPathFixedValus = sPathofTable+"/FIXED_VALUES";

  if(fixedValue===true){
    fixedValueString="X";
  }else{
    fixedValueString="";
  }

  sap.ui.getCore().getModel('projektModel').setProperty(sPathFixedValus, fixedValueString);
},
```

Listing 6.66: Projektcontrolling: Dateneingabe Binding CheckBox: setFixedValuesInsert

Die beiden Funktionen werden dann, wie in folgendem Listing 6.67, innerhalb des UI Elements CheckBox aufgerufen.

```
new sap.m.CheckBox("checkBoxUpdateProject1",{
  selected:{path: 'projektModel>FIXED_VALUES',
  formatter: function(item){
  return oController.getFixedValue(item);
  }},
  select:[oController.setFixedValuesInsert, oController],
}),
```

Listing 6.67: Projektcontrolling: Dateneingabe Binding CheckBox

UI Ergänzung Abbildung 6.37 zeigt die Wertergänzung. Sind bereits Werte eines Projektes in der Datenbank vorhanden, aber noch nicht im UI gepflegt worden, kann der User hier die Daten vervollständigen und speichern.

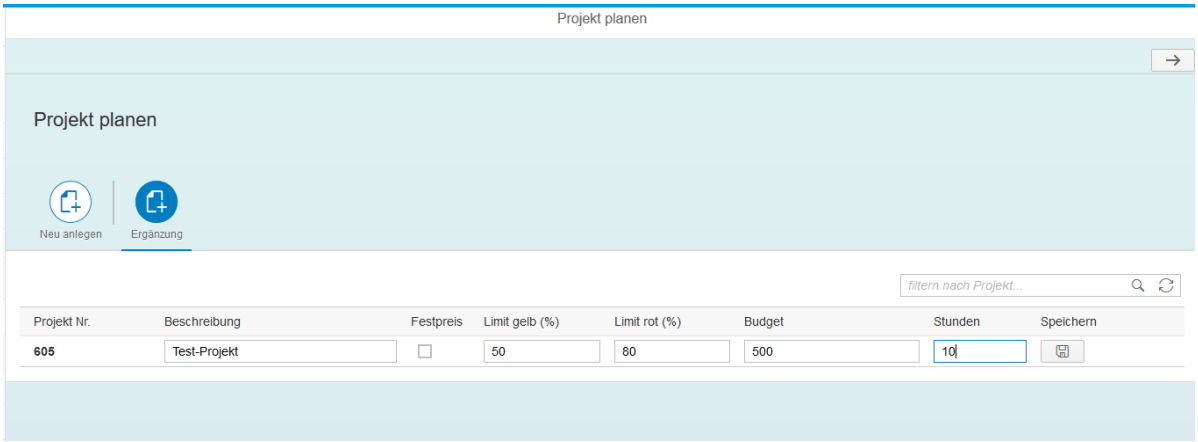


Abbildung 6.37.: Projektcontrolling: Dateneingabe Projekt Ergänzung

UI Ergänzung - Insert Die Funktionen zum Speichern der einzelnen Datensätze sind wie die Insert Funktion „saveDataProject1()“ (UI Neu anlegen - Insert) aufgebaut. Der einzige Unterschied liegt darin, dass lediglich ein Datensatz und nicht mehrere Daten auf einmal gesichert werden. Zum Speichern der Daten wurden folgende Funktionen implementiert:

- saveDataProject2()
- saveDataPsp2()
- saveDataskillPspEmployee2()
- saveDataskillPspRates2()
- saveDataplanPsp2()

Da der Aufbau der Funktionen fast identisch ist, wird an dieser Stelle auf ein Beispiel verzichtet. Die in Abbildung 6.35 auf Seite 288 zu sehenden Services werden auch in diesem Abschnitt zum Speichern der Daten genutzt.

Zum Einlesen der Daten mussten weitere Funktionen entwickelt werden, welche die Services in Abbildung 6.38 nutzen. Die Funktionen ermöglichen das Anzeigen von Zeilen mit teilweise vorhandenen Werten. Folgende Funktionen wurden für die Oberflächen implementiert:

- getModelMVProject()
- getModelMVPsp()
- getModelMVEmployee()
- getModelMVRate()

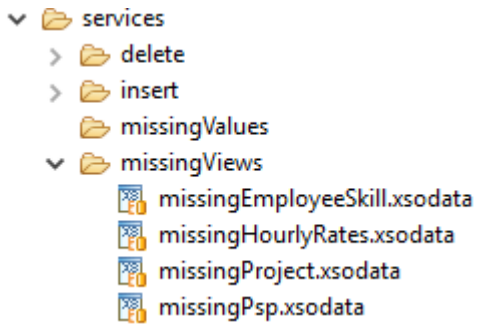


Abbildung 6.38.: Projektcontrolling: Dateneingabe Wertergänzung Services

Listing 6.68 auf der nächsten Seite zeigt die Funktion „getModelMVProject()“. Der Service „MISSING_PROJECT“ wird mit AJAX aufgerufen. Der Service lädt die bestehenden Werte aus der Datenbank, in diesem Fall die Projektnummer.

```
/**FUNCTIONS Missing Value
*
* Beschreibung:
* Um die Werte in der UI anzeigen zu koennen und die leeren Felder in der
    jeweiligen Zeile zu erzeugen. Der vorhandene Wert wird mit der Hilfe
    eines Services bereitgestellt. Die fehlenden Werte bzw. leeren Felder
    der Tabellen muessen ueber eine for-Schleife in das Model gepusht
    werden.
*/

/**FUNCTIONS Missing Value
* getModelMVProject
*/
  getModelMVProject: function(){
    var oModel = sap.ui.getCore().getModel('projectDataMissingValueModel')
    ;
    var self= this;

    $.ajax({
      url: "http://192.168.252.42:8002/abat/demosystem/projektcontrolling/
    controlling/services/missingViews/missingProject.xsodata/
    MISSING_PROJECT/$count",
      async:false,
      type:"GET",
      dataType:"json",
      cache:"false",
      success: function(response){
      self.lengthMVModel = response;
      },
      complete: function(){
      },
      error:function(xhr,status,error){
    }
  });
```

Listing 6.68: Projektcontrolling: Funktion Missing Values

Die Tabelle muss um die fehlenden Werte, also leere Felder ergänzt werden. Dies erfolgt in Listing 6.69 mit einer for-Schleife. Zum Erzeugen von leeren Tabellenfeldern muss das richtige Feld mit einem leeren String an das Model „projectDataMissingValueModel“ übergeben werden. Mit dem Befehl „setProperty“ können die Daten in das Model gepusht werden (vgl. Listing 6.69, Zeile 3).

```
  for (var i = 0; i < this.lengthMVModel; i++) {
    var pathBUDGET = "/d/results/" + i + "/BUDGET";
    oModel.setProperty(pathBUDGET, "");
    var pathHOURS = "/d/results/" + i + "/HOURS";
    oModel.setProperty(pathHOURS, "");
    var pathFIXEDVALUES = "/d/results/" + i + "/FIXEDVALUES";
    oModel.setProperty(pathFIXEDVALUES, "");
    var pathDESCRIPTION = "/d/results/" + i + "/DESCRIPTION";
    oModel.setProperty(pathDESCRIPTION, "");
```

```
10      var pathLIMITYELLOW = "/d/results/" + i + "/LIMITYELLOW";
11      oModel.setProperty(pathLIMITYELLOW, "");
12      var pathLIMITRED = "/d/results/" + i + "/LIMITRED";
13      oModel.setProperty(pathLIMITRED, "");
14      var pathSCORE_VALUE = "/d/results/" + i + "/SCORE_VALUE";
15      oModel.setProperty(pathSCORE_VALUE, "B");
16    }},
```

Listing 6.69: Projektcontrolling: Funktion Missing Values - leere Felder

Wie oben bereits beschrieben wird auch hier bei erfolgreicher Speicherung eine Textnachricht ausgegeben (vgl. Abbildung 6.39). Tritt ein Fehler auf, wird auch hier eine Fehlermeldung ausgegeben.

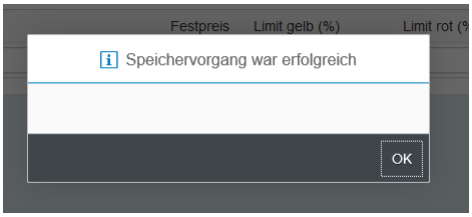


Abbildung 6.39.: Projektcontrolling: Dateneingabe Projekt Ergänzung MessageBox

Datenpflege In der Datenpflege kann auch zwischen je zwei Ansichten gewechselt werden - Aktualisieren und Löschen. In beiden Sichten werden die gesamten Daten aus der Datenbank angezeigt. Der User kann die Daten einzeln ändern und abspeichern oder sogar löschen. Abbildung 6.40 zeigt die erste Seite “Projekt bearbeiten“ der Datenpflege.

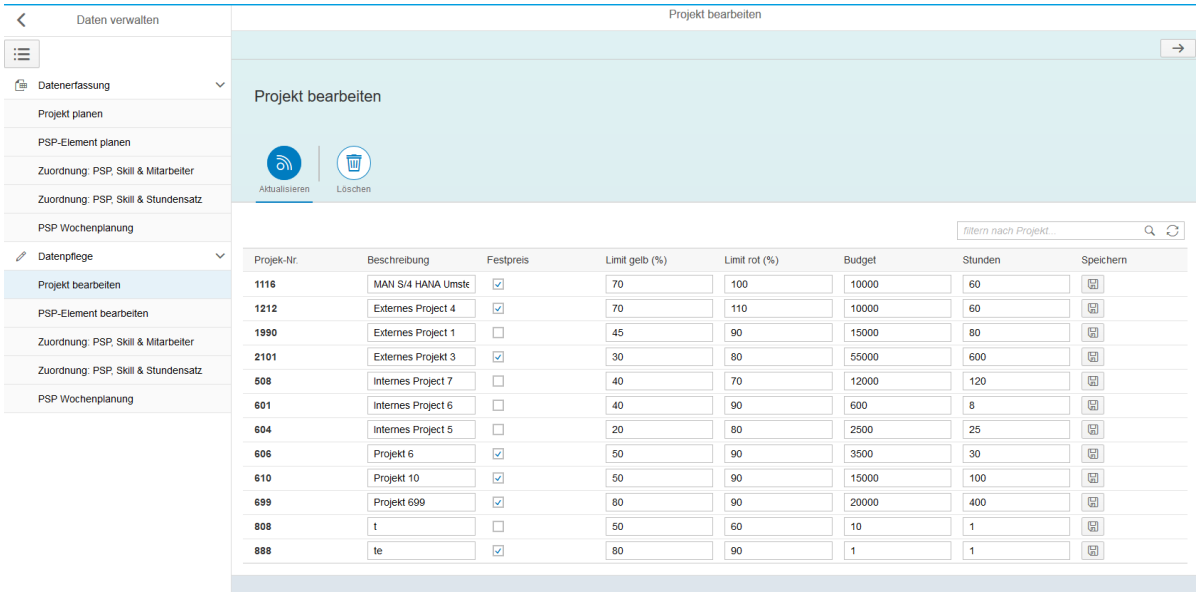


Abbildung 6.40.: Projektcontrolling: Datenpflege

Zur Anzeige der gesamten Daten werden beim Aktualisieren und Löschen die gleichen Models aus der anfangs deklarierten Tabelle 6.10 auf Seite 283 genutzt. Dabei werden folgende Services aufgerufen:

- getCurrentProjectStatus.xsjs
- getCurrentPspStatus.xsjs
- getSkillPspMa.xsjs
- getSkillPspRate.xsjs
- getPlanValuesData.xsjs

Zum Aktualisieren der Daten werden die Services in Abbildung 6.41 genutzt:

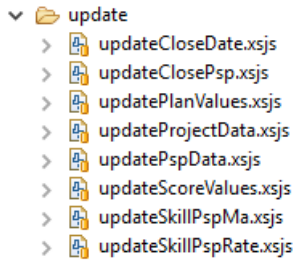


Abbildung 6.41.: Projektcontrolling: Datenpflege Update Services

Zum Löschen der Daten werden die Services in Abbildung 6.42 genutzt:

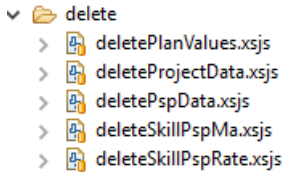


Abbildung 6.42.: Projektcontrolling: Datenpflege Delete Services

Wie in der Dateneingabe kann auch hier über Icons zwischen den Ansichten navigiert werden (vgl. Abbildung 6.43 und Listing 6.54 auf Seite 283).

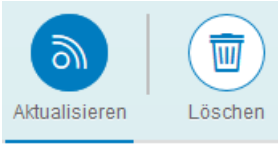


Abbildung 6.43.: Projektcontrolling: Datenpflege zwei Ansichten

UI Aktualisieren Abbildung 6.44 zeigt die UI „Projekt bearbeiten“. Über den Service „getCurrentProjectStatus“ werden alle bestehenden Projekte aus der Datenbank in die Tabelle geladen. Bis auf die Projekt-Nr. können alle Felder bearbeitet werden und per Klick auf den Button in der letzten Spalte gesichert werden.

Abbildung 6.44.: Projektcontrolling: Datenpflege Aktualisieren

Folgende Funktionen wurden für die unterschiedlichen UIs implementiert:

- updateProject()
- updatePsp()
- updateSkillPspRate()
- updateSkillPspMa()
- updatePlanValues()
- setCloseDate()

Listing 6.70 zeigt die Funktion „updateProject()“, welche die veränderten Projektdaten sichert.

```
1  /**
2  * FUNCTION - UPDATE:
3  *   Variablen:
4  * updateContent: Liest das Model aus, welches die (geaenderten) Daten
       enthaelt, um diese an den Service zu uebergeben
5  * Beschreibung:
6  * Die Daten werden aus dem Model gelesen, um diese als Variablen
       abzuspeichern und an den Service zu uebergeben. Wurden
7  * die se ausgelesen, wird mit Hilfe eines AJAX-Aufrufes der Service
       angesprochen und in der URL die in der GUI angegebenen
8  * Variablen uebergeben. War der Aufruf erfolgreich, werden die refresh-
       Mothoden aufgerufen und es erscheint eine Message-Box,+
9  * welche ausgibt, ob die Daten erfolgreich in die Datenbank geladen wurde.
10 * Bei ROW_COUNT = 1 war das Laden in die Datenbank erfolgreich und es
       erscheint eine positive Rueckmeldung
11 * Bei ROW_COUNT = 0 war das Laden in die Datenbank nicht erfolgreich und
       eine negative Rueckmeldung erfolgt
```

```
12 **/
13
14 /**
15 * FUNCTION - UPDATE:
16 updateProject
17 */
18   updateProject: function(oEvent){
19   var updateContent = oEvent.getSource().getBindingContext('
       projectDataModel').getProperty();
20   //Variablen, die aus dem Model "gelesen" werden, um diese im spaeteren
       Verlauf an den Service zu uebergeben
21   var pNumber = updateContent.PROJECTNUMBER;
22   var description = updateContent.DESCRIPTION;
23   var plannedBudget = updateContent.PLANNED_BUDGET;
24   var plannedHours = updateContent.PLANNED_HOURS;
25   var limitYellow = updateContent.PROJECT_LIMIT_YELLOW;
26   var limitRed = updateContent.PROJECT_LIMIT_RED;
27   var fixedValues = updateContent.FIXED_VALUES;
28
29   var self = this
30   $.ajax({
31     url: "/abat/demosystem/projektcontrolling/controlling/services/update/
       updateProjectData.xsjs?PROJECT="+pNumber+"&DESCRIPTION="+description+"&
       PLANNED_BUDGET="+plannedBudget+"&PLANNED_HOURS="+plannedHours+"&
       PROJECT_LIMIT_YELLOW="+limitYellow+"&PROJECT_LIMIT_RED="+limitRed+"&
       FIXED_VALUES="+fixedValues,
32     async:false,
33     type:"GET",
34     dataType:"json",
35     cache:"false",
36     success: function(response){
37     self.refreshProjectData();
38     self.refreshSkillPSPMa();
39     self.refreshPSPData();
40     self.refreshSkillPSPRate();
41     var infoMessage;
42     if (response.values[0].ROW_COUNT == "1"){
43       infoMessage = "Das Projekt "+ "\""+pNumber+"\"" + " wurde
       erfolgreich aktualisiert."
44     }else{
45       infoMessage = "Update war nicht erfolgreich."
46     }
47     jQuery.sap.require("sap.m.MessageBox");
48     sap.m.MessageBox.show(infoMessage, {
49     icon: sap.m.MessageBox.Icon.INFORMATION,
50     title: "Update-Status",
51     actions: [sap.m.MessageBox.Action.OK],
52     onClose: function(oAction) {
53     }.bind(this)
54     });
55     },
56     complete: function(){
```

```
57      },
58      error:function(xhr,status,error){
59      }
60    });
61
62 },
```

Listing 6.70: Projektcontrolling: Datenpflege Update Funktion

Bei erfolgreicher Aktualisierung wird eine MessageBox mit Textnachricht ausgegeben.

PSP-Element sperren PSP-Elemente können gesperrt werden und Stundenzettel, welche nach Sperrdatum eingereicht wurden, können nur noch manuell eingepflegt werden. Die Stundenzettel können dann von Usern in der Auswertung eingepflegt werden. Die Sperrung eines PSP-Elements erfolgt manuell in der Datenpflege (vgl. Abbildung 6.45).

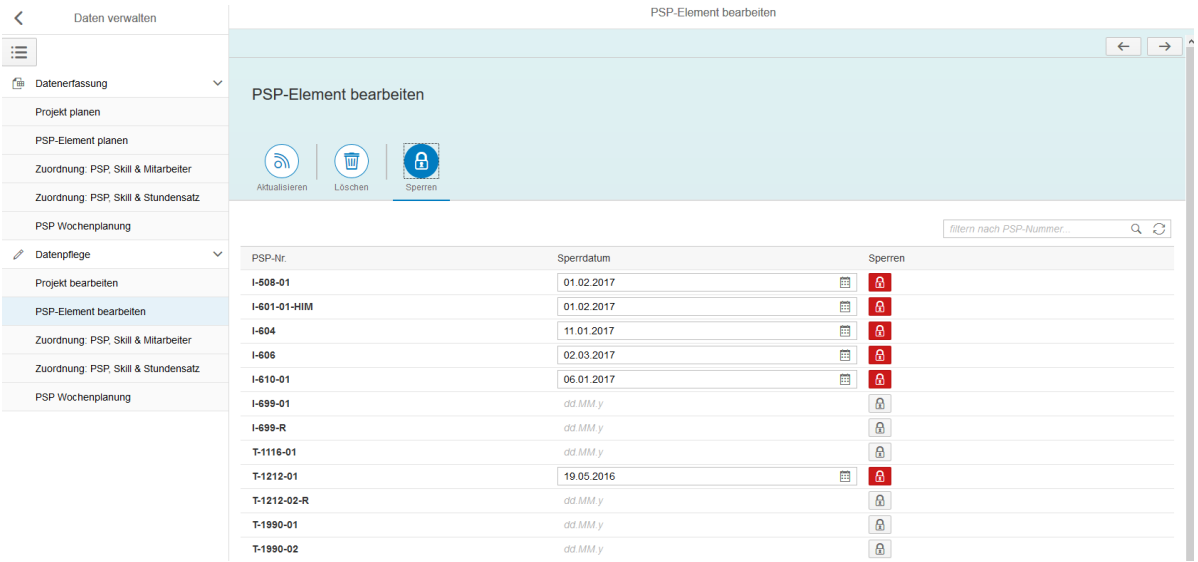


Abbildung 6.45.: Projektcontrolling: Datenpflege PSP sperren

Zu finden ist dies in der Navigation unter „PSP-Element bearbeiten“. Neben den Icons „Aktualisieren“ und „Löschen“, kann der User die Option „Sperren“ auswählen (vgl. Abbildung 6.46). Der User kann ein Datum wählen, ab welchem die Stundenzettel nicht mehr automatisch mit in die Auswertung übernommen werden. Zu Anfang war geplant das Sperrdatum, ebenfalls wie die Stundenzettel-Daten, über eine Schnittstelle abzurufen und die Sperrung sollte im SAP ERP 6.0 erfolgen. Aufgrund der fehlenden Information der Schnittstelle war dies aber nicht möglich.

Abbildung 6.46.: Projektcontrolling: Datenpflege PSP sperren IconTab

Das Sperren eines PSP-Elements erfolgt über den in der Tabelle enthaltenen Sperr-Button. Falls der Button rot dargestellt wird, ist das PSP-Element gesperrt und ein entsprechendes Sperrdatum kann angegeben werden. Beim initialen bzw. erneuten Sperren des PSP-Elements befindet sich der DatePicker, mit dem das Sperrdatum angegeben wird auf dem aktuellen Datum.

Die Sperrfunktion realisiert die Methode „updateClosePsp()“ (vgl. Listing 6.71). Diese wird ausgeführt, sobald der Button geklickt wird. In dieser Funktion wird an den Service „updateClosePsp.xsjs“ die zu sperrende PSP-Nummer übergeben. Außerdem wird der Zustand, ob das PSP-Element gesperrt wird, in die Variable „close“ gespeichert. Anschließend wird der Zustand des Buttons entsprechend der Variable „close“ verändert, indem dieser mit der ID aufgerufen wird (vgl. Listing 6.71 Zeile 15 - 19).

```
1  updateClosePsp: function(oEvent){
2    var updateContent = oEvent.getSource().getBindingContext('pspDataModel
     ').getProperty();
3    var pspNumber =updateContent.PSP_NUMBER;
4    var close = updateContent.CLOSED;
5
6    var self = this;
7    $.ajax({
8      url: "/abat/demosystem/projektcontrolling/controlling/services/
     update/updateClosePsp.xsjs?PSP="+pspNumber,
9      async:false,
10     type:"GET",
11     dataType:"json",
12     cache:"false",
13     success: function(response){
14
15       if(close == "X"){
16         sap.ui.getCore().byId(oEvent.getSource().sId).setType(sap.m.
     ButtonType.Default);
17       }else{
18         sap.ui.getCore().byId(oEvent.getSource().sId).setType(sap.m.
     ButtonType.Reject);
19       }
20       self.refreshPSPData();
21     },
22     complete: function(){
23
24     },
```

```
25          error:function(xhr,status,error){
26
27          }
28      });
29  },
```

Listing 6.71: Projektcontrolling: Datenpflege updateCloseDate()

Mit dem Ändern des Datums im DatePicker wird das Sperrdatum des PSP-Elements verschoben. Dies realisiert die Funktion „setCloseDate()“. In ähnlicher Weise, wie die oben dargestellte Funktion, ruft der Service zunächst die PSP-Nummer ab. Zusätzlich wird das Datum des DatePickers abgefragt. Beide Parameter werden dann ebenfalls an den Service„updateClosePsp()“ übergeben.

UI Löschen Abbildung 6.47 zeigt die UI „Projekt bearbeiten“. Über den Service „getCurrentProjectStatus“ werden alle bestehenden Projekte aus der Datenbank in die Tabelle geladen. User können hier gesamte Projekt oder in den anderen UIs einzelne Datensätze löschen.

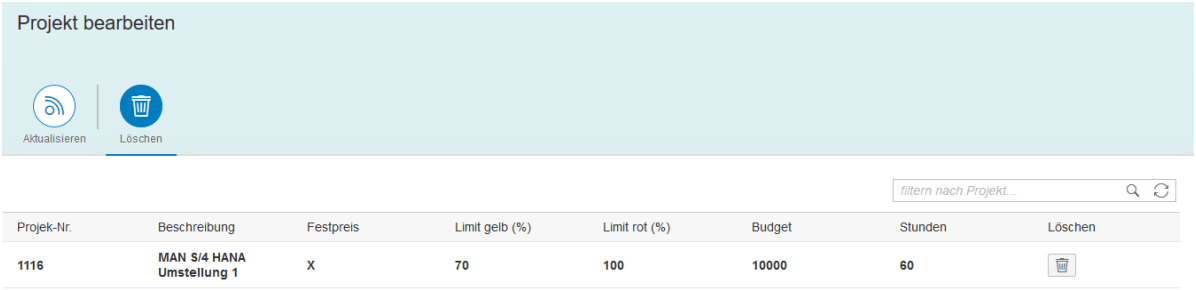


Abbildung 6.47.: Projektcontrolling: Datenpflege Delete

Es werden auch alle zugeordneten Elemente eines Projektes gelöscht. In der letzten Spalte befindet sich der Button, welcher die Funktion „confirmDeleteProject()“ aufruft und einen Dialog startet. Wird dieser mit „Ja“ bestätigt, wird die Funktion „deleteProject()“ aufgerufen und das Projekt wird gelöscht. Abbildung 6.48 zeigt das Dialog-Fenster.

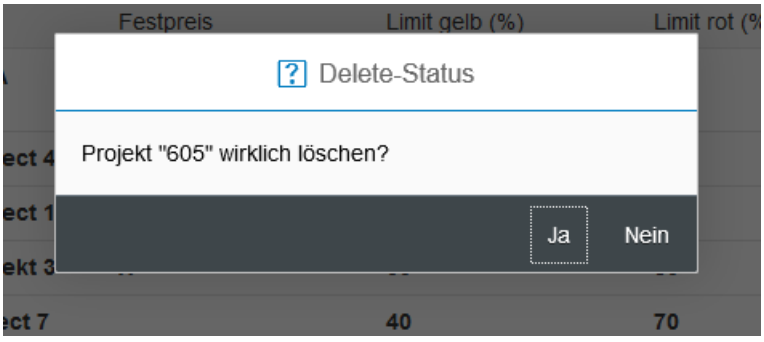


Abbildung 6.48.: Projektcontrolling: Datenpflege Delete Dialog

Folgende Funktionen wurden für die unterschiedlichen UIs implementiert:

- confirmDeleteProject()
- deleteProject()
- confirmDeletePsp()
- deletePsp()
- confirmDeleteSkillPspRate()
- deleteSkillPspRate()
- deletePlanValues()

Listing 6.72 zeigt die Funktion „confirmDeleteProject()“, welche den Dialog startet.

```
1  confirmDeleteProject: function(oEvent){
2    var deleteContent = oEvent.getSource().getBindingContext('
     projectDataModel').getProperty();
3    var project = deleteContent.PROJECTNUMBER;
4    jQuery.sap.require("sap.m.MessageBox");
5    sap.m.MessageBox.show("Projekt  \""+ project+"\"  wirklich loeschen?",
     {
6    icon: sap.m.MessageBox.Icon.QUESTION,
7    title: "Delete-Status",
8    actions: [sap.m.MessageBox.Action.YES, sap.m.MessageBox.Action.NO],
9    onClose: function(oAction) {
10     if (oAction == "YES") {
11       this.deleteProject(project);
12     }
13   }.bind(this)
14   });
15 },
```

Listing 6.72: Projektcontrolling: Datenpflege Delete Confirm

Listing 6.73 zeigt die Funktion „deleteProject()“, welche den Datensatz und alle damit in Verbindung stehenden Datensätze, wie PSP-Elemente, löscht.

```
1  deleteProject:function(project){
2    var self = this
3    $.ajax({
4      url: "/abat/demosystem/projektcontrolling/controlling/services/
     delete/deleteProjectData.xsjs?PROJECT="+project,
5      async:false,
6      type:"GET",
7      dataType:"json",
8      cache:"false",
9      success: function(response){
10     self.refreshProjectData();
11     self.refreshSkillPSPMa();
12     self.refreshPSPData();
```

```
13        self.refreshSkillPSPRate();
14        var infoMessage;
15        if (response.values[0].ROW_COUNT === "1"){
16          infoMessage = "Projekt "+ "\""+project+"\"" + " wurde entfernt."
17        }else{
18          infoMessage = "Loeschen war nicht erfolgreich."
19        }
20        jQuery.sap.require("sap.m.MessageBox");
21        sap.m.MessageBox.show(infoMessage, {
22        icon: sap.m.MessageBox.Icon.INFORMATION,
23        title: "Delete-Status",
24        actions: [sap.m.MessageBox.Action.OK],
25        onClose: function(oAction) {
26        }.bind(this)
27        });
28        },
29        complete: function(){
30        },
31        error:function(xhr,status,error){
32        }
33      });
34    },
```

Listing 6.73: Projektcontrolling: Datenpflege Delete

Wie in Listing 6.73 auf der vorherigen Seite ab Zeile 14 zu sehen ist wird eine „sap.m.MessageBox“ verwendet. Ist der Löschvorgang erfolgreich/erfolglos erhält der User erneut eine Nachricht (vgl. Abbildung 6.49).

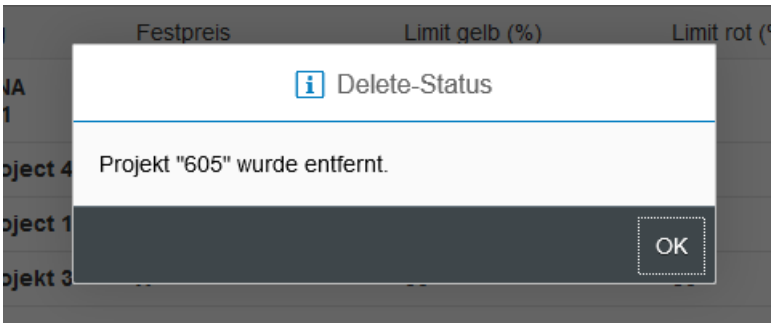


Abbildung 6.49.: Projektcontrolling: Datenpflege Delete Message

Verwendung der Excel-Vorlagen Um es zu ermöglichen Kunden einen Überblick über ihre Projekte zu geben, wurden zwei Excel-Vorlagen erstellt, welche mit den exportierten CSV-Dateien der Anwendung arbeiten können. In diesem Abschnitt wird erläutert, wie die Vorlagen genutzt werden. Nachdem über den CSV-Export-Button eine CSV-Datei des Projekt exportiert wurde, muss diese in die Exceldatei importiert werden. Hierzu wird auf das Tabellenblatt „Datenpflege“ der Vorlage „Projektcontrolling“ gewechselt. Anschließend wird über das Menü *Daten* → *Aus Text* zur gespeicherten CSV-Datei navigiert. Es öffnet sich der Textkonvertierungs-Assistent, in welchem folgende Angaben gemacht werden müssen:

- **Seite 1:** Dateiursprung: „UTF-8“; Haken bei „Die Daten haben Überschriften.“
- **Seite 2:** Trennzeichen: Tabstopp
- **Seite 3:** Button „Weitere...“ → Dezimaltrennzeichen = „.“; 1000er-Trennzeichen = „,“

Die Daten müssen anschließend im bestehenden Datenblatt in Zeile A1 importiert werden. Nach dem Datenimport kann auf das Tabellenblatt „Datenauswertung“ gewechselt werden. Hier müssen abschließend die beiden Tabellen „Projektauswertung“ und „PSP-Status“ mittels *Rechtsklick* → *Aktualisieren* aktualisiert werden. Die importierten Daten werden nun in den Pivot-Tabellen dargestellt und es ist möglich die Daten zu aggregieren bzw. aufzureißen. Werte mit der Kennzeichnung „(Leer)“ können über das Filtersymbol der Pivot-Tabellen ausgeblendet werden. Abbildung 6.50 zeigt einen Auszug der Excel-Auswertung mit importierten Daten.

Projektauswertung:

Projektnummer	Stunden	Aufwand
1990		
T-1990-01 (Element 1)		
01.08.2016	4	340,00 €
02.08.2016	9	765,00 €
03.08.2016	9	765,00 €
04.08.2016	9	765,00 €
05.08.2016	4	340,00 €
T-1990-02 (Element 2)		
01.08.2016	0	0,00 €
02.08.2016	0	0,00 €
03.08.2016	0	0,00 €
04.08.2016	0	0,00 €
05.08.2016	0	0,00 €
T-1990-02-R (Element 2 Remote)		
01.08.2016	7	0,00 €
05.08.2016	4	170,00 €
Gesamtergebnis	46	3.145,00 €

Projektstatus:

Projektnummer:	Gesamtaufwand:	Planaufwand:	Stundenaufwand:	Planstunden:	Budgetverbrauch:
1990	3.145,00 €	15.000,00 €	46	80	21%

PSP-Status:

PSP-Nummer:	Gesamtaufwand:	Planaufwand:	Stundenaufwand	Planstunden:	Budgetverbrauch:
T-1990-01	2.975,00 €	7.000,00 €	35,00	40,00	43%
T-1990-02	0,00 €	7.000,00 €	0,00	30,00	0%
T-1990-02-R	170,00 €	1.000,00 €	11,00	10,00	17%

Abbildung 6.50.: Projektcontrolling: Excel-Auswertung (Auszug)

Bei der zweiten Excel-Vorlage handelt es sich um ein Werkzeug zum Vergleich eines Projektes zu verschiedenen Zeitpunkten (Zeitscheiben). Der Vorgang zum Datenimport funktioniert dabei identisch wie bei der Auswertungsdatei, es steht jedoch ein zweites Tabellenblatt zur Datenpflege bereit, in welches die CSV-Datei einer anderen Zeitscheibe importiert werden kann.

Dies realisiert die Gegenüberstellung zweier Datenstände eines Projekts, was es ermöglicht Änderungen in einem zeitlichen Verlauf zu ermitteln.
Abbildung 6.51 zeigt einen Historienvergleich von Projekt „2101“. Es kann ermittelt werden, dass in den Daten vom 06.03.2017 im PSP-Element „T-2101-05-P-399“ ein erhöhter Budgetverbrauch im Vergleich vorliegt.

Projektauswertung:					
Datum der Daten:			**Datum der Daten:**		
06.03.2017			**26.12.2016**		
Projektnummer	**Stunden**	**Aufwand**	**Projektnummer**	**Stunden**	**Aufwand**
2101			**2101**		
T-2101-03 (PSP-Element 1)	**397**	**27.170,00 €**	**T-2101-03 (PSP-Element 1)**	**397**	**27.170,00 €**
T-2101-05-P-399 (PSP-Element 2101)			**T-2101-05-P-399 (PSP-Element 2101)**		
28.11.2016	6	300,00 €	28.11.2016	6	300,00 €
29.11.2016	6	300,00 €	29.11.2016	6	300,00 €
05.12.2016	7	350,00 €	05.12.2016	7	350,00 €
06.12.2016	7	350,00 €	06.12.2016	7	350,00 €
12.12.2016	8	400,00 €	12.12.2016	8	400,00 €
13.12.2016	8	400,00 €	13.12.2016	8	400,00 €
14.12.2016	8	400,00 €	14.12.2016	8	400,00 €
15.12.2016	8	400,00 €	15.12.2016	8	400,00 €
16.12.2016	8	400,00 €	16.12.2016	8	400,00 €
19.12.2016	8,5	425,00 €	**T-2101-05-P-432 (PSP-Element 2)**	**123**	**6.810,00 €**
20.12.2016	9,5	475,00 €	**Gesamtergebnis**	**586**	**37.280,00 €**
21.12.2016	10	500,00 €			
22.12.2016	10	500,00 €			
23.12.2016	6	300,00 €			
02.01.2017	7,5	375,00 €			
03.01.2017	8,5	425,00 €			
04.01.2017	9	450,00 €			
05.01.2017	9	450,00 €			
06.01.2017	6	300,00 €			
T-2101-05-P-432 (PSP-Element 2)	**123**	**6.810,00 €**			
Gesamtergebnis	**670**	**41.480,00 €**			

Abbildung 6.51.: Projektcontrolling: Excel-Historienvergleich

6.4. Abnahme

Die Abnahme des Szenarios wurde am 10.03.2017 in Bremen mit Markus Fischer durchgeführt. Zur Abnahme wurden dabei alle Anforderungen geprüft und mit „umgesetzt“, „teilweise umgesetzt“ und „nicht umgesetzt“ bewertet. Anforderungen wurden nur abgenommen, wenn der Status „umgesetzt“ bzw. „teilweise umgesetzt“ war.

Tabelle 6.11.: Projektcontrolling: Abnahme - Darstellung und Auswertung

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
1	**Darstellung und Auswertung**		
1.1	Die Fiori-Oberfläche soll eine grafische Darstellung des Projektstatus ermöglichen.	umgesetzt	abgenommen
1.2a	In der grafischen Oberfläche sollen abgerechnete Arbeitspakete gekennzeichnet werden	umgesetzt	abgenommen
1.2b	In der Standard-Excel-Auswertung sollen abgerechnete Arbeitspakete gekennzeichnet werden	umgesetzt	abgenommen
1.3a	In der grafischen Fiori-Oberfläche sollen Schwellenwerte nach Status gekennzeichnet werden	umgesetzt	abgenommen
1.3b	In der Standard-Excel-Auswertung sollen abgerechnete Arbeitspakete gekennzeichnet werden	umgesetzt	abgenommen
1.4	Das System muss eine Exportmöglichkeit nach Excel bieten	umgesetzt	abgenommen

Anmerkungen:

- Anforderung 1.2a: In Weiterentwicklungen sollte berücksichtigt werden, dass die Eingabe mehrerer Rechnungsnummern realisiert wird. Ebenfalls sollte die Abrechnung auf Arbeitseinträge verfeinert werden.

Tabelle 6.12.: Projektcontrolling: Abnahme - Einlesen der Daten

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
2	**Darstellung und Auswertung der Daten**		
2.1	Das System muss ein Datenmodell bereitstellen	umgesetzt	abgenommen
2.2 / 2.3	Fiori-Oberflächen zur Massendatenerfassung müssen bereitgestellt werden	umgesetzt	abgenommen
2.2.1	Mobile Oberfläche (App)	teilweise umgesetzt	abgenommen
2.3.1	Eine Oberfläche muss Stammdaten erfassen	umgesetzt	abgenommen
2.3.2	Eine Oberfläche muss Planwerte erfassen.	umgesetzt	abgenommen
2.4	E-Mail-Benachrichtigung und Monitoring Tool für Fehlerfälle	umgesetzt	abgenommen
2.5	Die Vollständigkeit der Daten (Timesheets) muss geprüft werden	umgesetzt	abgenommen
2.6	Ein Rabatt/Risikoaufschlag muss mit einbezogen werden können	nicht umgesetzt	-
2.7	Der Sperrstatus von PSP-Elementen muss abgerufen werden	umgesetzt	abgenommen
2.8	Konsistenz der Daten prüfen (keine ungültigen Zuordnungen speicherbar)	umgesetzt	abgenommen

Anmerkungen:

- Anforderung 2.2.1: Die Navigationselemente lassen sich auf einem Handy bzw. Tablet nicht einblenden. Um die Auswertung neu starten, muss die Seite neu geladen werden.
- Anforderung 2.5: Das Filtern nach PSP-Elementen wäre in Weiterentwicklungen sinnvoll.

- Anforderung 2.6: Die Anforderung wurde aus Zeitgründen nicht realisiert, da kein Feld für den Rabatt im Datenmodell vorhanden ist. Eine Anpassung hätte Aufgrund des knappen Zeitplans nicht realisiert werden können.
- Anforderung 2.7: Die Information über den Sperrstatus ist nicht vorhanden, es kann aber manuell gesperrt werden. Das Sperren erfolgt aufgrund des fehlenden Abgabedatums anhand des angegeben Arbeitstags. Ein Abgabedatum wäre daher bei Weiterentwicklungen des Funktionsbausteins sinnvoll.

Tabelle 6.13.: Projektcontrolling: Abnahme - Berechnungsgrundlagen und Algorithmen

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
3	**Berechnungsgrundlagen und Algorithmen**		
3.1	Ein Algorithmus zur Berechnung der aktuellen Budget/Stunden-Situation muss erstellt werden	umgesetzt	abgenommen
3.2	Ein Algorithmus muss die Berechnung des Projektstatus gewährleisten	umgesetzt	abgenommen
3.3	Ein Algorithmus muss den Abruf des Stundensatzes ermöglichen	umgesetzt	abgenommen
3.4	Historienvergleich über verschiedene Zeitscheiben muss möglich sein	umgesetzt	abgenommen

Tabelle 6.14.: Projektcontrolling: Abnahme - Extra Schnittstelle / Funktionsbaustein

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
4	**Extra Schnittstelle / Funktionsbaustein**		
4.1	Über eine Schnittstelle soll eine PSP-Elementbeschreibung bereitgestellt werden	umgesetzt	abgenommen
4.2	Die HANA-DB muss täglich mit Daten versorgt werden	umgesetzt	abgenommen

Anmerkungen:

- Anforderung 4.1: Diese Information wird von der Schnittstelle nicht bereitgestellt, die Bezeichnung kann aber manuell eingegeben werden.

6.5. Fazit

Die Ergebnisse des Szenarios Projektcontrolling verdeutlichen, dass sich SAPUI5 sehr gut für die Entwicklung von Controlling-Applikationen eignet. Sowohl das Reporting und die (grafische) Darstellung des Projektstatus, als auch die Erfassung und Pflege projektspezifischer Daten gelang erfolgreich. In der Entwicklung hat sich die Aufgabeneinteilung in die Zuständigkeitsbereiche Datenbank und grafische Oberfläche bewährt, da innerhalb des Szenarios auf diese Weise Expertisen aufgebaut werden konnten.
Die Schwierigkeit einer kontinuierlichen Datenversorgung durch ein SAP ERP 6.0 - System ließ sich nicht ohne eine Programmierung in ABAP bzw. größeren Entwicklungsaufwand implementieren. Daher wurde unter der Berücksichtigung knapper Kapazitäten die ABAP-Programmierung gewählt. Die für die Datenübernahme vorgesehene „Landscape Transformation“, ist für das replizieren von Tabellen zwar gut geeignet, jedoch erfolgte der Datenabruf über einen Funktionsbaustein, welcher im Abruf wiederum Daten aus mehreren Tabellen bündelt. Landscape Transformation war so leider nicht einsetzbar.

6.6. Ausblick

Der Grundstein für ein IT-gesteuertes Projektcontrolling ist gelegt und beinahe alle Anforderungen an den Prototypen wurden umgesetzt. abat möchte den Prototypen in den folgenden Monaten in einem kleinen Projekt einsetzen und anschließend weiterentwickeln. Die Applikation wird aktuell noch mit Test-Daten versorgt. Der nächste Schritt wäre daher der Wechsel zu einer Schnittstelle, welche reale Daten liefert. Weitere mögliche Schritte sind die Implementierung eines Berechtigungs-/Rollensystems, in dem nicht jeder User Zugriff auf alle Daten hat. Beispielsweise könnten Kunden Zugriff auf einen Teil der Projektauswertung erhalten und Projektleiter sehen nur die jeweiligen zugewiesenen Projekte. Denkbar wäre neben der Wochenplanung auch eine Monatsplanung von PSP-Elementen. Die Prüfung der Vollständigkeit der Timesheets kann um einen Filter für die PSP-Nr. erweitert werden. Da die Navigation auf einem Smartphone nur eingeschränkt funktioniert, besteht auch in der mobilen Darstellung noch Verbesserungspotenzial. Auch wichtig ist die Berücksichtigung von mehreren Rechnungsnummern pro PSP-Element, aktuell kann nur eine Rechnungsnummer je PSP-Element hinterlegt werden.

7. Szenario - Echtzeit Stromanalyse

7.1. Beschreibung

Diese Dokumentation beschreibt die Verarbeitung des Szenarios „Echtzeit Stromanalyse“ der Masterprojektgruppe „Demosystem on HANA “ der Universität Oldenburg. Das Szenario ist ähnlich wie das Szenario Sensorik. Das Anwendungsszenario befasst sich mit der Verarbeitung von Stromdaten. Die Daten eines Gerätes werden von einer intelligenten Steckdose erfasst, um z.B. den Stromverbrauch eines Geschirrspülers zu messen. Ziel ist, die gemessenen Daten an die SAP HANA Datenbank zu schicken und darin zu speichern und zu analysieren. Danach ist mit SAPUI5 eine grafische Oberfläche zu programmieren, um die analysierten Daten in der darzustellen, z.B. wo das Gerät steht, der Höchstwert, Line-Chart Darstellung usw.

Diese Dokumentation beschreibt zuerst im Teil Themenfindung wie das Thema gefunden und wieso diese Steckdose ausgewählt wurde. Anschließend werden die Anforderungen mit Tabellen dargestellt, in welchen die einzelnen Anforderungen kurz beschrieben werden und welche Ergebnisse das Szenario letztendlich erreichen soll. Ein weiterer Punkt der Dokumentation ist die Umsetzung, hier wird die technische Umsetzung Schritt für Schritt vorgestellt und wie die Anforderungen realisiert wurden, z.B. die Dokumentation des Codes der grafischen Oberfläche. Schließlich werden die durchgeführten Abnahmetests inklusive erreichter Anforderungen dargestellt.

7.2. Themenfindung

7.2.1. Vorgehen

Die Vorgabe aus dem Kickoff zu Beginn des Projektes war es, dass ein weiteres Szenario nach der Übergabe der ersten Prototypen umgesetzt werden soll. Hierzu war es zunächst nötig Ideen für ein Solches Szenario zu finden. Als Kreativitätstechniken für Themen- und Ideenfindung wurde das Brainstorming und Mindmaps eingesetzt. Diese wurden in den internen, wöchentlichen Meetings erstellt. Anschließend wurden diese in den Meetings bei der abat AG vorgetragen und daraus entstanden die unten folgenden Projektideen. Der Stand der einzelnen Mindmaps ist den folgenden Abbildungen zu entnehmen.

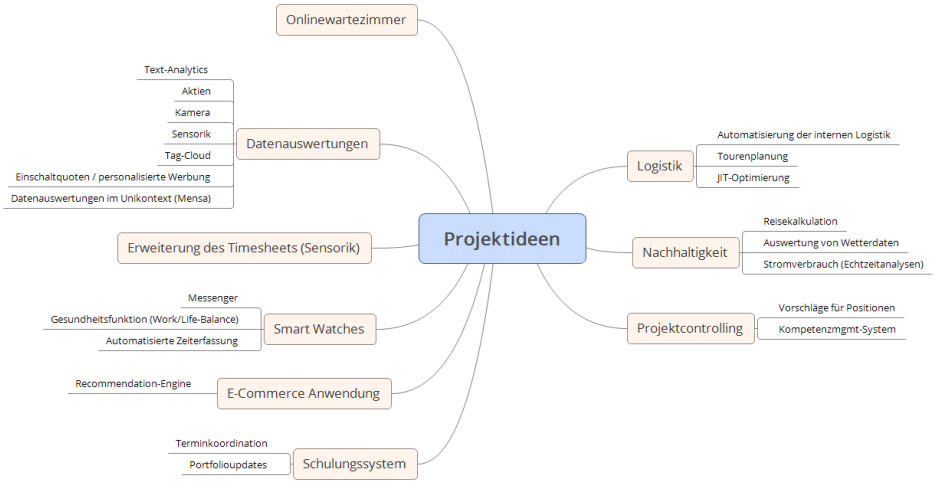


Abbildung 7.1.: Echtzeit Stromanalyse: Erstes Brainstorming

Das oben gezeigte Mindmap wurde am 27.04.2016 erstellt und war das Ergebnis des ersten Brainstormings.

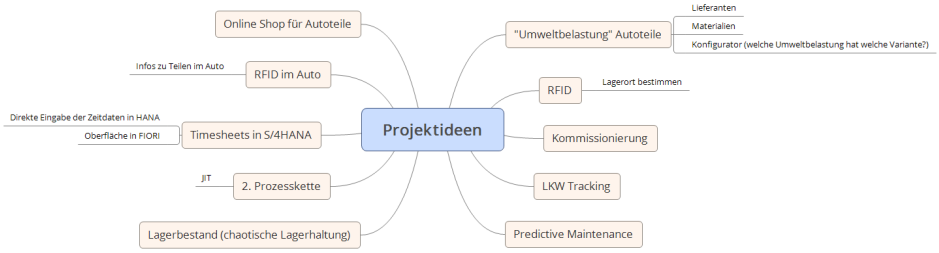


Abbildung 7.2.: Echtzeit Stromanalyse: Zweites Brainstorming

Die oben gezeigte Mindmap ist eine Überarbeitung der vorherigen Mindmap. Sie wurde am 22.06.2016 erstellt. Wie zu sehen ist, sind Ideen verworfen und neue Ideen hinzugefügt worden.

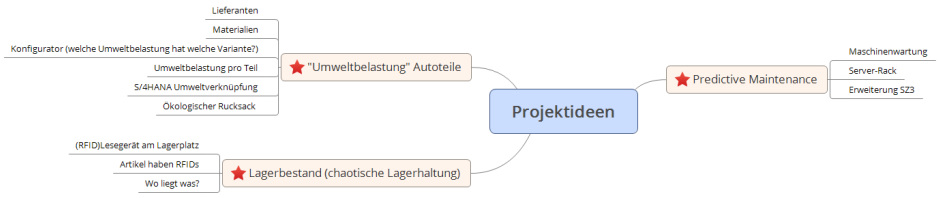


Abbildung 7.3.: Echtzeit Stromanalyse: Drittes Brainstorming

Aus dieser Mindmap sind die im Folgenden weiter beschriebenen Projektideen entstanden. Diese wurden als grobe Idee beschrieben und bei der abat AG vorgestellt.

7.2.2. Erste Projektideen

Das Ziel dieses Abschnittes ist die Erläuterung der Ideen zum vierten Prototypen, zur Ideenfindung hat die Projektgruppe DoHA innerhalb eines Brainstormings nach Ideen gesucht, um ein Szenario zu finden, welches einen aktiven Nutzen für die abat AG bzw. dessen Kunden hat. Aus diesem Brainstorming wurden drei Szenarien ausgewählt, die kurz erläutert werden. Der Stand der folgenden Projektideen war der 09.06.2016.

Schulungssystem/Projektmanagement

Das Szenario Schulungssystem/Projektmanagement hat das Ziel die Kenntnisse eines Mitarbeiters transparent darzustellen und Mitarbeiter möglichst in den Projekten einzusetzen, die den jeweiligen Erfahrungen entsprechen. Im Schulungssystem hat jeder Mitarbeiter eine Bandbreite von Skills ,welche wiederum in Erfahrungsstufen eingeteilt sind. Die Skills und Erfahrungsstufen hängen davon ab, welche Vorkenntnisse ein Mitarbeiter vorweist, welche Schulungen er absolviert hat und in welchen Projekten er mitgewirkt hat. Durch die Teilnahme an Schulungen und das Arbeiten in Projekten kann die Erfahrungsstufe eines Mitarbeiters steigen oder neue Skills können hinzukommen. Dem Projektmanager werden beim Anlegen eines Projektes Mitarbeiter als potenzielle Projektmitglieder vorgeschlagen, welche aufgrund ihrer Kenntnisse und Erfahrungsstufen den Anforderungen des Projekts am meisten entsprechen. Dabei werden die Faktoren Mitarbeiterkosten, Skill/Erfahrungsstufe und die verfügbare Zeitkapazität des Mitarbeiters berücksichtigt. Dadurch, dass die Kenntnisse jedes Mitarbeiters transparent sind, ist es möglich Schulungen anzubieten, die den Bedürfnissen der Mitarbeiter am ehesten entsprechen. Ein weiterer Vorteil, der sich aus diesem System ergibt, ist dass Mitarbeiter Ansprechpartner durch dieses System suchen können. Besitzt ein Mitarbeiter z.B. den Skill Java, kann er bei Aufkommen eines Problems oder bei einer Frage mit Hilfe des Systems einen möglichen Ansprechpartner mit einer (höheren) Erfahrungsstufe in diesem Skill innerhalb des Unternehmens suchen.

Echtzeitanalyse Stromverbrauch

Im Szenario Echtzeitanalyse Stromverbrauch wird der Stromverbrauch eines Unternehmens in Echtzeit aufgezeichnet und den Mitarbeitern visualisiert dargestellt. Ziel ist es den Mitarbeitern zu verdeutlichen, wie hoch der momentane Stromverbrauch ist und sie zu ermutigen diesen möglichst gering zu halten. Die Visualisierung in SAPUI5 ist dabei anschaulich zu gestalten. Die CO2-Emission, die durch den verbrauchten Strom verursacht wird kann beispielsweise so dargestellt werden, dass die Fläche an Bäumen, die aufgeforstet werden muss, um die Emission zu kompensieren, angezeigt wird. Der Vorteil dieser Darstellung ist, dass dem Mitarbeiter die CO2-Emission auf diese Weise veranschaulicht wird. Wird dem Mitarbeiter ein Ausstoß von 13 Tonnen CO2 pro Jahr dargestellt, kann er sich die Folgen schlecht vorstellen. Wird ihm hingegen diese Zahl als ein Hektar Waldfläche, die aufgeforstet werden muss, um den CO2-Ausstoß zu kompensieren präsentiert, wird die Zahl für ihn greifbar. Durch das Messen des Verbrauchs für jeden Arbeitsplatz, ist eine noch detailliertere Darstellung möglich, die den genauen

Stromverbrauch pro Arbeitsplatz misst und analysiert. Bei einem zu hohen Verbrauch wird eine Benachrichtigung gesendet und Maßnahmen zur Verringerung des Verbrauches können vorgenommen werden. Mit der Echtzeitanalyse des Stromverbrauchs macht das Unternehmen z.B. Mitarbeiter auf den Stromverbrauch aufmerksam. Auf diese Weise ist es möglich den Verbrauch zu verringern, Kosten einzusparen und die Umweltbilanz des Unternehmens zu verbessern.

Zeiterfassung Smartwatch

Smartwatches bieten eine Reihe von Möglichkeiten auswertbare Daten zu erfassen. Zwei Szenarien, die im Kontext von SAP S/4HANA interessant sind, werden im Folgenden erläutert.

Zeiterfassung mit Smartwatches
Die Zeiterfassung mit Hilfe von Smartwatches bietet die Möglichkeit die gearbeiteten Stunden pro Projekt bequem in das System zu laden. Durch Auswahl des jeweiligen PSP-Elements kann der Mitarbeiter die Zeit erfassen, die er an einem Projekt arbeitet. Wurde die Zeit erfrasst, werden die Daten in die SAP HANA-Datenbank geladen. Der Vorteil dieser Art der Zeiterfassung ist, dass die Daten stets aktuell sind. Anstatt dem wöchentlichen Pflegen und Versenden der Timesheets, werden die Daten tagesaktuell an die Datenbank gesendet. Der Vorteil der Smartwatch liegt darin, dass der Mitarbeiter diese jederzeit bei sich tragen kann. Er kann seine Zeit also sowohl beim Kunden, als auch seinem festen Arbeitsplatz erfassen.

Stress-Level ermitteln
Durch den Pulsmesser der Smartwatch wird der durchschnittliche Stress-Level eines Mitarbeiters berechnet. Die Daten werden an die Datenbank übertragen. Ist der Stress-Level über einen längeren Zeitraum hoch, wird sowohl der Mitarbeiter, als auch sein Vorgesetzter benachrichtigt. Der Vorgesetzte kann in diesem Fall Gegenmaßnahmen einleiten und dem Mitarbeiter werden vom System Tipps zum Abbau des Stress-Levels gegeben.

7.2.3. Überarbeitete Projektideen

Der Stand der folgenden Projektideen war der 01.08.2016.

Predictive Maintenance

Das Szenario des „Predictive Maintenance“ beschreibt eine Erweiterung des dritten Szenarios „Sensorik“. Da durch dieses Szenario bereits das Auslesen und Anzeigen von einzelnen Sensoren umgesetzt wurde, erweitert die Idee des „Predicted Maintenance“ die Sensorik durch automatisches Auswerten und Analysieren der gespeicherten Werte. So könnten z.B. spezielle Sensoren einer Maschine ausgelesen, gespeichert und anschließend genauer analysiert werden. Ziel dieser Analyse sind Datenmuster, die mögliche Ausfälle, Fehler oder Probleme einer Maschine bereits vor Auftreten dieser erkennen lassen. Da

durch vorhersagbare potenzielle Probleme Unternehmen bereits vor Ausfall der Maschine Maßnahmen oder Reparaturen einleiten können, bietet dieses Szenario den Vorteil verringerter Ausfallzeiten und Produktionsstillstände für Unternehmen. Das Unternehmen könnte somit Wartungszyklen besser abstimmen, Reparaturen effizienter gestalten und so eine möglichst kontinuierliche Auslastung der Maschinen erwirken. Unternehmen erhalten so die Möglichkeit der besseren Produktionsplanung, einer längeren Laufzeit der Maschinen und der Vermeidung von ungeplanten und kostenintensiven Stillständen. Mögliche Sensoren zur Analyse von Produktionsmaschinen sind Ölsensoren, Vibrationssensoren, Ultraschallsensoren oder Thermografiesensoren. Schwierigkeit dieses Szenarios ist das Entwickeln der verschiedenen Algorithmen, welche die gespeicherten Daten anhand der Maschine und der historischen Daten auswertet und eine zuverlässige Empfehlung für die Zukunft ausgibt. Zusammenfassend können Unternehmen dank der „Predictive Maintenance“ nicht mehr nur reaktiv sondern bereits proaktiv auf potenzielle Fehler reagieren.

Umweltbelastung von Autoteilen

Unter dem Szenario „Umweltbelastung von Autoteilen“ ist zu verstehen, dass berechnet werden soll, wie hoch die Umweltbelastung eines Autos anhand der darin verbauten Teile ist.
Dabei wird als erstes für jedes mögliche Teil eines bestimmten Autos die Umweltbelastung berechnet. Hierfür sind verschiedene Vorgehensweisen möglich. Es kann z.B. ermittelt werden wie hoch der CO2 Ausstoß über den gesamten Lebenszyklus hinweg ist. Eine weitere Möglichkeit ist, dass der ökologische Rucksack des Autos berechnet wird. Dieser entspricht dem gesamten Materialbedarf, welcher während des Lebenszyklus des Produktes, anfällt. Es gibt jedoch noch weitere Möglichkeiten und Kombinationen um die Umweltbelastung eines Autoteiles zu ermitteln. In jedem Fall soll der gesamte Lebenszyklus eines Autoteils betrachtet werden. Dieser Lebenszyklus beginnt bei der Gewinnung der Rohstoffe, welche für die Produktion des Produktes benötigt werden und erstreckt sich bis zur Entsorgung des Produktes. Dabei sind nicht nur die Materialien zu betrachten, welche direkt in das Autoteil einfließen sondern z.B. auch die Hilfsgüter in der Produktion.
Im nächsten Schritt soll das Auto mit Hilfe eines Konfigurators aus möglichen Teilen zusammengestellt werden. Dabei kann bei einigen Teilen, wie z.B. den Sitzen, zwischen verschiedenen Varianten gewählt werden. Und einige Teile sind optional, wie z.B. ein eingebautes Navigationsgerät.
Wenn das Auto dann aus diesen Teilen zusammengestellt wurde, soll das System anzeigen, wie hoch die Umweltbelastung des gesamten Autos ist. Falls vorhanden sollen zudem alternative Konfigurationen des Autos angezeigt werden, welche die Umwelt in geringerem Umfang belasten.
Ein mögliches Problem in diesem Szenario ist die Beschaffung der nötigen Daten um die Umweltbelastung zu berechnen. Denn dabei müssen viele Faktoren betrachtet werden, welche nicht zwingend in präziser Form vorliegen oder genau einem Autoteil zugewiesen werden können.

Insgesamt richtet sich dieses Szenario an Autohersteller, welche unter anderem Kunden von abat sind. Diese können wiederum ihren Kunden auf einfache Weise Transparenz und einen weiteren Gesichtspunkt bei der Konfiguration ihres Autos bieten.

Lagerbestand (chaotische Lagerhaltung)

Das Szenario „Lagerbestand (chaotische Lagerhaltung)“ beschreibt eine hochgradig dynamische Art der Lagerhaltung, welche durch die RFID-Technologie und SAP HANA unterstützt wird. Der Grundgedanke ist es, mittels günstiger RFID-Chips Lagerbestände eindeutig identifizierbar zu machen, sowie Lagerplätze mit RFID-Lesegeräten intelligent zu machen.
Die Lagerplätze sollen in der Lage sein, die gelagerten Artikel zu identifizieren und diese selbstständig in SAP HANA abzubilden. Hierfür müssten die Artikel nur einmalig im Wareneingang gebucht werden, um die RFID zu den Artikeln zuzuordnen. Anschließend würden die Lesegeräte der Lagerorte automatisch aktualisieren, wo welcher Artikel in welcher Menge zu finden ist, ohne dass ein Mitarbeiter den Artikel händisch umbuchen muss.
Hierdurch verringert sich zum einen die benötigte Arbeitszeit zum Umlagern von Gütern, zum anderen sinkt die Wahrscheinlichkeit von Fehlbuchungen durch Mitarbeiter drastisch. Auch spiegelt der Lagerbestand des Systems bei korrekter Funktionalität immer den tatsächlich vorhandenen Lagerbestand wieder, da die Lesegeräte praktisch permanent prüfen welche Artikel vorliegen.
Ein denkbarer Rahmen für die Umsetzung des Projekts ist die Ausstattung kleinerer Räume mit RFID-Lesegeräten, welche die Lagerorte simulieren und durch das Bewegen von Gütern zwischen diesen Räumen im System anzeigen, was an welchem Ort eingelagert ist.
In der Praxis müsste gewährleistet werden, dass immer nur ein Lesegerät zu einer Zeit ein Gut identifiziert, damit es nicht zu Konflikten oder doppelten Anzeigen kommt. Ebenfalls muss das Aktualisieren der Lagerorte und -mengen nahezu in Echtzeit erfolgen, um ein korrektes Bild der Realität widerzuspiegeln, was hohe technische Anforderungen darstellt.
Das Szenario ist zusammenfassend ein Schritt in die Richtung vom „Internet of Things“ und stellt somit ein Beispiel für ein hochaktuelles Thema mit hoher praktischer Relevanz dar.
Aus den oben beschriebenen Ideen sind die Szenarien Echtzeit Stromanalyse und Produktnachhaltigkeit in SAP S/4HANA entstanden. Im folgenden Abschnitt wird die Endgültige Projektidee nochmals in der Endgültigen Variante beschrieben.

7.3. Hardwareauswahl

7.3.1. Vorgehen

Für die weitere Ausarbeitung eines Konzeptes wurden Vorschläge für mögliche Hardware, die im Szenario zum Einsatz kommen könnte, verschriftlicht. Diese werden im folgenden Abschnitt aufgeführt. Nach weiteren Absprachen mit allen Projektbeteiligten ist jedoch klar geworden, dass zunächst eine Überprüfung der Machbarkeit und Umsetzbarkeit im Zusammenhang mit der ausgewählten Hardware bezüglich der verbleibenden Zeit, der Installationsmöglichkeiten bei der abat AG sowie weiteren technischen-, funktionalen- und Kosten-Kriterien durchgeführt werden sollte. Die weitere Hardwareauswahl fand dann nach den Kriterien, welche in den Anforderungen beschrieben wurden, statt.

7.3.2. Vorschläge in der Konzeption

digitalSTROM-Hochvolt-Chip

Die erste Komponente, welche benötigt wird, um den Strom von einzelnen Geräten messen zu können, ist der digitalSTROM-Hochvolt-Chip. Dieser ist nur 4 mal 6 Millimeter groß und kann entweder direkt in die Geräte eingebaut werden oder in Form einer Lüsterklemme vor diese geschaltet werden. Mit Hilfe des Hochvolt-Chips lässt sich der Stromverbrauch von einzelnen Geräten messen. Außerdem lassen sich darüber Verbrauchergeräte steuern. Um die Messwerte zu sammeln und an einen Server zu schicken wird der digitalSTROM-Meter benötigt.

digitalSTROM-Meter

Der digitalSTROM-Meter ist die zweite Komponente, welche für dieses Szenario benötigt wird, wenn die Lösung von digitalSTROM verwendet wird. Der digitalSTROM-Meter wird direkt in den Stromkasten verbaut. Dieser misst den Stromverbrauch des Stromnetzes, an den er angeschlossen ist und erkennt automatisch die installierten digitalSTROM-Klemmen bzw. Hochvolt-Chips. Diese werden dann vom digitalSTROM-Meter zu einem System verbunden und gesteuert. Zudem sendet der digitalSTROM-Meter alle gesammelten Daten dieses Systems an einen digitalSTROM-Server.

digitalSTROM-Server

Der digitalSTROM-Server dient dazu das System mit einem internen Netzwerk oder dem Internet zu verbinden. Über diese Verbindung kann das digitalSTROM-System und deren Anwendungen konfiguriert und abgerufen werden. Die Funktionen des digitalSTROM-Systems können dabei von anderen Systemen über eine JSON API abgerufen werden.

digitalSTROM-Filter

Der digitalSTROM-Filter dient dazu das System in einem 230-V-Netz zu betreiben und die Qualität der Kommunikation des Systems in diesem Netz sicherzustellen.

Software

Der digitalSTROM-Server bietet standardmäßig einige Funktionen zur Steuerung des Systems an. Diese können über das Internet oder zum Beispiel über einen Webbrowser aufgerufen werden.
Für Entwickler bietet dieser Server zudem eine entsprechende Plattform an, die auch in einer lizenzfreien Version, mit den wichtigsten Funktionen, vorliegt. Dabei können mit Hilfe einer JSON-API scriptbasierte Anwendungen geschrieben werden. So können auch externe Systeme auf die Funktionen des Servers zugreifen und diese erweitern.
Damit wäre es möglich die entsprechenden Daten über den Stromverbrauch zu nutzen und auf der SAPUI5 Oberfläche darzustellen.

Ergebnis der Überprüfung der Umsetzbarkeit

Nach einem Treffen mit der technischen Abteilung der abat AG ist klar geworden, dass die oben beschriebene Hardware nicht in diesem Szenario umsetzbar ist. Aus diesem Grund wurde diese Idee verworfen und es wurde als Ziel gesetzt eine möglichst einfache und unkomplizierte Hardware für die Datenerhebung und das Messen von Stromdaten zu finden. Diese sollte kostengünstig und am Markt innerhalb weniger Tage zu erhalten sein.

7.3.3. Auswahl und Bewertung nach Abnahme der Anforderungen

Die folgende Auflistung zeigt die Hardware, die im Szenario zur Auswahl stand:

1. Energiekosten-Messgerät VOLTCRAFT PLC3000 DE Powerline
    - Beschreibung: Der Energieverbrauch, die Spannung, der Strom, die Frequenz und der CO2-Ausstoß sollen mit diesem Gerät gemessen werden können.
    - Watt: 0.23 Watt bis 3.680 Watt.
    - Preis: 79,99 Euro.
    - Vorteile: Hohe Belastbarkeit an Watt und eine hohe Messgenauigkeit.
    - Nachteile: Es wird ein weiteres Gerät (Adapter) benötigt, welcher sich im selben Stromkreis befindet muss.
2. AVM Intelligente Steckdose AVM FRITZ!DECT 200
    - Beschreibung: Watt und Temperaturmessung.
    - Watt: 0 bis 2300 Watt.
    - Preis: 44,00 Euro.
    - Vorteile: WLAN Schnittstelle, JavaScript fähig.
    - Nachteile: Das AVM Gerät ist für weniger Watt ausgelegt als andere Geräte.

3. COST CONTROL 2
    - Beschreibung: Funk - Leistungsmesser mit 2 Steckdosensensoren und einem zentralen Überwachungsmonitor.
    - Watt: 0 bis 3600 Watt.
    - Preis: 39,40 Euro.
    - Vorteile:
    - Nachteile: Keine Lan oder W-Lan Schnittstelle.
4. EMG PL PM231E
    - Beschreibung: Messung von Spannung, Frequenz, Strom, Leistungsfaktor und Leistung.
    - Watt: 0 bis 3600 Watt.
    - Preis: 14,80 Euro.
    - Vorteile:
    - Nachteile: Keine Lan oder W-Lan Schnittstelle.
5. GEM PWM-LAN
    - Beschreibung: Messung des Stromverbrauchs bis zu 24 Std.
    - Watt: 0 bis 3600 Watt.
    - Preis: 69,20 Euro
    - Vorteile:
    - Nachteile: Keine W-Lan Schnittstelle.

Nach einer internen Abstimmung und Absprache mit der abat AG wurde entschieden die AVM Intelligente Steckdose AVM FRITZ!DECT 200 anzuschaffen.

7.4. Anforderungen

7.4.1. Tabellarische Anforderungen

Tabelle 7.1.: Echtzeit Stromanalyse: Hardware

Nr.	Anforderung:	Beschreibung:
1	**Hardware**	
1.1	Hardware auswählen	Festlegen welche Hardware genutzt werden soll (und diese beschaffen)
1.2	Hardware installieren	Hardware an ein Gerät in der Küche der abat AG anschließen

Tabelle 7.2.: Echtzeit Stromanalyse: Übersicht der Anforderungen des Datenmodells

Nr.	**Anforderung:**	**Beschreibung:**
2	**Datenmodell**	
2.1	Datenbankschema erstellen	An SAP HANA angepasstes Datenmodell erstellen
2.2	Tabellen anlegen	Modularer Aufbau, möglichst wenige Tabellen
2.3	System Modular aufbauen	Es sollen weitere, unterschiedliche Arten von Geräten ins System integriert werden können

Tabelle 7.3.: Echtzeit Stromanalyse: Übersicht der Anforderungen der Messung

Nr.	**Anforderung:**	**Beschreibung:**
3	**Messung**	
3.1	Stromdaten messen	Stromverbrauch von einem Gerät messen
3.2	Stromdaten aufbereiten	Stromdaten für die Übertragung an die Datenbank aufbereiten
3.2.1	Relevante Messwerte festlegen	Festlegen welche Messwerte für das Szenario benötigt werden
3.2.2	Granularität der Messwerte festlegen	Häufigkeit der Messung / Datenübertragung festlegen

Tabelle 7.4.: Echtzeit Stromanalyse: Übersicht der Anforderungen für die Schnittstelle

Nr.	Anforderung:	Beschreibung:
4	**Schnittstelle**	
4.1	Schnittstelle zum SAP HANA System einrichten	-
4.2	Messdaten and Datenbank senden	Die Messdaten des Stromverbrauchs vom Messgerät an die Datenbank schicken
4.3	Messdaten speichern	Messdaten des Stromverbrauchs zusammen mit weiteren Daten (Uhrzeit, Datum, Ort) in der Datenbank speichern

Tabelle 7.5.: Echtzeit Stromanalyse: Übersicht der Anforderungen für die Darstellung

Nr.	Anforderung:	Beschreibung:
5	**Darstellung der Daten**	
5.1	Stromverbrauch in der SAPUI5 Oberfläche darstellen	Der Stromverbrauch soll im ersten Schritt in einfacher Form in der SAPUI5 Oberfläche dargestellt werden
5.2	Stromverbrauch grafisch darstellen	Der Stromverbrauch soll anschaulich dargestellt werden. Dafür soll im zunächst der Verlauf des Stromverbrauchs über den Tag dargestellt werden

Tabelle 7.6.: Echtzeit Stromanalyse: Übersicht der Anforderungen für die Datenanalyse

Nr.	Anforderung:	Beschreibung:
6	**Datenanalyse**	
6.1	Messdaten auswerten	weitere Statistiken
6.2	Vergleichsdaten erheben	Den Stromverbrauch von z.B. verschiedenen Spülprogrammen der Spülmaschine oder verschiedenen Getränken des Kaffeeautomaten messen

Tabelle 7.7.: Echtzeit Stromanalyse: Übersicht der Anforderungen für die Erweiterungen

Nr.	Anforderung:	Beschreibung:
7	**Erweiterungen**	
7.1	Vergleichsdaten auswerten	Den Stromverbrauch von z.B. verschiedenen Spülprogrammen der Spülmaschine oder verschiedenen Getränken des Kaffeeautomaten erkennen können (Algorithmus)
7.2	Verschiedenen Programme der Geräte grafisch darstellen	In der SAPUI5 Oberfläche darstellen z.B. welches Spülprogramm gerade genutzt wird.

7.4.2. Hardware

Hardware auswählen

- Aufwand: Mittel
- Beschreibung: Es muss recherchiert werden welche Hardware zur Strommessung für die Systemumgebung und die anzuschließenden Geräte im Gebäude der abat AG, geeignet ist. Das Messgerät sollte eine W-LAN oder LAN Schnittstelle besitzen, um die zu erfassenden Daten möglichst einfach an die SAP HANA Datenbank übertragen zu können. Dabei muss vor allem darauf geachtet werden, dass die Messdaten an die Datenbank geschickt werden können. Es wurde sich im Vorfeld darauf geeinigt eine intelligente Steckdose für die Messung zu benutzen. Bei dieser

sollte im speziellen darauf geachtet werden für welche Stromstärke (Watt) die intelligente Steckdose ausgelegt ist und ob diese somit für die anzuschließende Geräte geeignet ist. Außerdem sollte die Messgenauigkeit und der Aufwand bzw. die Möglichkeit der Installation bei der Auswahl beachtet werden. Zuletzt sind der Preis und die Lieferzeit der möglichen Hardware als Kriterien zur Auswahl der Hardware zu nennen. Wenn die passende Hardware ausgewählt wurde muss diese anschließen beschafft werden.

Hardware installieren

- Aufwand: Niedrig
- Beschreibung: Nachdem die passende Hardware zur Strommessung ausgewählt und beschafft wurde, muss diese im Gebäude von der abat AG installiert werden. Dazu muss die Hardware konfiguriert werden, so dass die Messdaten an die SAP HANA Datenbank gesendet werden können. Außerdem sollte ein Gerät zur Messung ausgewählt werden welches zum einen die Maximalbelastung des Messgerätes nicht überschreitet und zum anderen regelmäßig im Einsatz ist, um alle Stromverbräuche messen zu können.

7.4.3. Datenmodell

Datenbankmodell erstellen

- Aufwand: Mittel
- Beschreibung: Es muss ein an die SAP HANA Datenbank und die zu messenden Daten angepasstes Datenbankmodell erstellt werden. Dabei ist darauf zu achten, dass möglichst wenig einzelne Tabellen erstellt werden müssen. Außerdem sollen die Tabellen erweiterbar sein, falls z.B. weitere Messgeräte integriert werden.

Tabellen anlegen

- Aufwand: Mittel
- Beschreibung: Es müssen die Tabellen entsprechend dem Datenbankmodell auf der SAP HANA Datenbank angelegt werden.

7.4.4. System Modular aufbauen

- Aufwand: Hoch
- Es sollen weitere, unterschiedliche Arten von Geräten ins System integriert werden können.

7.4.5. Messung

Stromdaten messen

- Aufwand: Niedrig
- Beschreibung: Es müssen die Stromdaten der entsprechenden Geräte in den Räumen der abat AG gemessen werden. Am wichtigsten ist dabei der Watt Wert, da dieser den Stromverbrauch angibt. Weitere Werte wie z.B. die Temperatur der Steckdose könnten aber auch gemessen werden.

Stromdaten aufbereiten

- Aufwand: Hoch
- Die Stromdaten sollen für die Übertragung an die Datenbank aufbereitet werden. Das heißt, dass alle Daten im selben Wertebereich an die Datenbank gesendet werden um eine Erweiterung bzw. andere Anordnung der Daten zu ermöglichen.

Relevante Messwerte festlegen

- Aufwand: Mittel
- Beschreibung: Es muss festgelegt werden welche Messdaten für das Szenario relevant sind. Dazu muss zunächst überprüft werden welche Messdaten die ausgewählte Hardware überhaupt messen und an die Datenbank versenden kann. Anschließend müssen anhand des gewünschten Ergebnisses des Szenarios festgelegt werden, welche Daten hierfür benötigt werden.

Granularität der Messwerte festlegen

- Aufwand: Mittel
- Beschreibung: Es muss festgelegt werden in welchen Zeitabständen die Stromdaten gemessen bzw. in die Datenbank geschrieben werden sollen. Dies hängt zum einen davon ab, wie oft hardwareseitig überhaupt Messwerte generiert werden können. Zum anderen hängt es davon ab wie stabil die Datenabfrage funktioniert. Eine sekündliche Abfrage könnte Probleme bei der Datenabfrage verursachen. Und nicht zuletzt hängt es davon ab wie oft es sinnvoll ist die Messdaten zu speichern. Was wiederum z.B. davon abhängt in welchen Zeitabständen sich die Messwerte signifikant verändern.

7.4.6. Schnittstelle

Schnittstelle zum SAP HANA System einrichten

- Aufwand: Hoch

- Beschreibung: Es muss eine Schnittstelle von der Messhardware zum SAP HANA System eingerichtet werden, so dass die Messdaten in die Datenbank geschrieben werden können. Diese soll mit Hilfe von JavaScript realisiert werden, weil die SAP HANA Datenbank ohne weitere Installation mit JavaScript arbeiten kann.

Messdaten an Datenbank senden

- Aufwand: Mittel
- Die Messdaten müssen vom Messgerät an die Datenbank geschickt werden. Die Übertragung soll mit Hilfe einer WLAN-Verbindung stattfinden. Die Daten sollen dabei mit Hilfe einer HTTP-Request von dem Messgerät abgefragt werden.

Messdaten speichern

- Aufwand: Hoch
- Beschreibung: Die Messdaten vom Strom müssen zusammen mit weiteren, dazu in Bezug stehenden, Daten wie z.B. Uhrzeit, Datum und Standort des Messgeräts gespeichert werden.

7.4.7. Darstellung der Daten

Stromverbrauch in der SAPUI5 Oberfläche darstellen

- Aufwand: Hoch
- Beschreibung: Im ersten Schritt sollen die Stromdaten in einfacher Form als Kachel in SAPUI5 dargestellt werden. Zum Beispiel soll der aktuelle Stromverbrauch in Watt dargestellt werden.

Stromverbrauch grafisch darstellen

- Aufwand: Hoch
- Beschreibung: Der Stromverbrauch soll in der SAPUI5 Oberfläche anschaulich dargestellt werden. Dazu soll im ersten Schritt der Verlauf über einen Tag und pro Gerät grafisch dargestellt werden. Dies kann z.B. in Form eines einfachen Diagramms erfolgen.

7.4.8. Soll-Anforderungen:

Datenanalyse

Messdaten auswerten

- Aufwand: Hoch
- Beschreibung: Neben den Verlauf des Stromverbrauchs über den Tag, sollen die Stromdaten noch für weiter Statistiken ausgewertet werden.

Vergleichsdaten erheben

- Aufwand: Hoch
- Beschreibung: Der Stromverbrauch von verschiedenen Programmen der Geräte, wie z.B. von verschiedenen Spülprogrammen der Spülmaschinen, soll gemessen werden. Diese Daten sollen in weiteren Schritten analysiert werden.

Systemerweiterung

Weiter Geräte anschließen

- Aufwand: Mittel
- Beschreibung: Es sollen beliebig viele Geräte zum Messen angeschlossen werden können und beliebig viele Messgeräte anschließbar sein. Diese Anforderung ist bereits im Datenbankmodell erfasst.

7.4.9. Kann-Anforderungen:

Vergleichsdaten auswerten

- Aufwand: Hoch
- Beschreibung: Es sollen die verschiedenen Programme der angeschlossen Geräten, wie z.B. die Spülprogramme der Spülmaschinen, mit Hilfe eines Algorithmus analysiert werden. Diese Analyse soll dazu genutzt werden, die verschiedenen Programme anhand des Stromverbrauchs erkennen zu können. Außerdem könnte eine weiter Anforderung die Disaggregation einer Lastkurve sein. Diese Lastkurve besteht dabei aus dem zusammen erfassten Stromverbrauch verschiedener Geräte.

Verschiedenen Programme der Geräte grafisch darstellen

- Aufwand: Hoch
- Beschreibung: Nachdem die verschiedenen Programme der angeschlossenen Geräte bzw. die verschiedenen Geräte an sich, erkannt werden können, sollen diese anschaulich in SAPUI5 dargestellt werden. So soll der Benutzer auf der SAPUI5 Oberfläche z.B. erkennen können welches Spülprogramm gerade genutzt wird oder welche Geräte momentan in Betrieb sind.

7.5. Umsetzung

In diesem Kapitel wird die Umsetzung und Implementierung der zuvor dargestellten Anforderungen genauer erläutert. Zuerst wird in Kapitel 7.5.1 die Architektur inklusive aller wesentlichen Elemente dargestellt und kurz beschrieben. Anschließend werden diese einzelnen Elemente in „HANA-Applikation“ und „Graphical User Interface“ unterteilt und detaillierter dargestellt. Hierbei werden die wesentlichen Implementierungen anhand von Code-Beispielen erläutert und aufgezeigt. Der gesamte Code der Implementierung dieses Szenarios ist auf der beigefügten CD zu finden.

7.5.1. Architektur

Die Architektur dieses Prototypen ist analog zur Architektur des Sensor-Szenarios aufgebaut (vgl. Kapitel 5.3.1 auf Seite 96). Es gibt lediglich kleine Anpassungen oder Einsparungen (siehe Abbildung 7.4 auf der nächsten Seite).

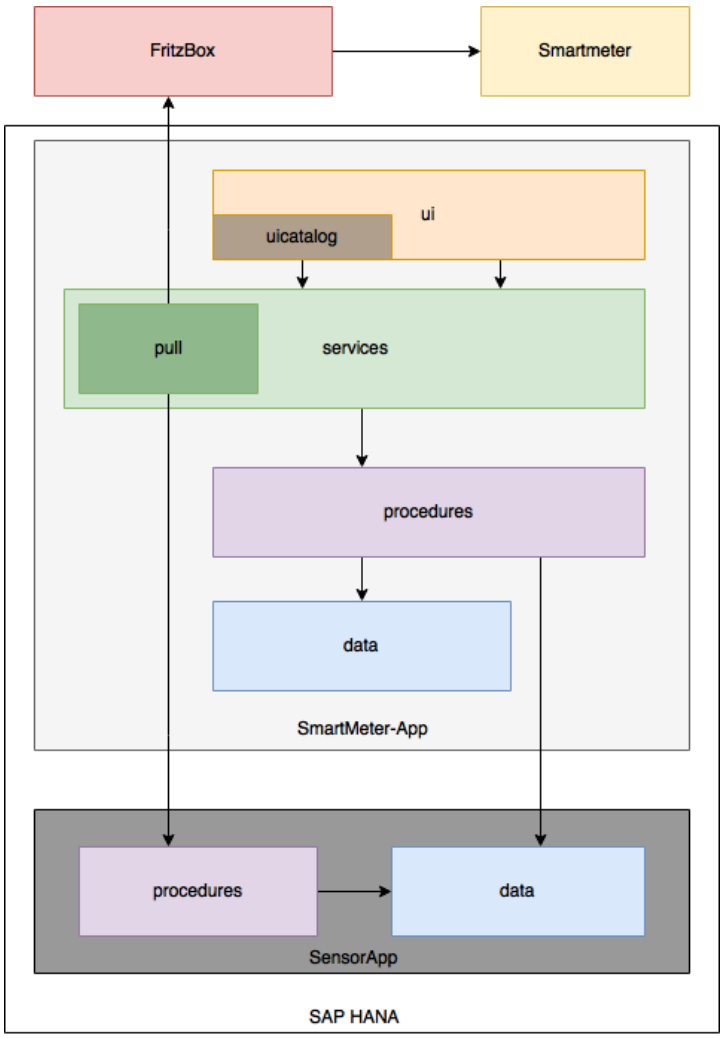


Abbildung 7.4.: Echtzeit Stromanalyse: Darstellung der Architektur

Ein wesentlicher Unterschied zum Sensorik-Szenario ist die verwendete Hardware. Die Fritzbox ist für das Abfragen der Werte der einzelnen Smartmeter verantwortlich. Diese können durch bestimmte Befehle der Fritzbox durch die erstellte Applikation abgerufen werden. Die Pakete „Services“, „Pull“, „Procedures“ haben die gleiche Funktionalität wie Anfangs für Szenario Sensorik beschrieben. Eine Besonderheit des Procedures- sowie des Pull-Packages ist, dass sie die Datenbanktabellen sowie das Datenbankschema des Szenarios Sensorik verwendet. Im Paket „Data“ sind demnach keine eigenen Datenbanktabellen oder Schema angelegt. Es beinhaltet lediglich eine angelegte Rolle für die Zuteilung der Aufrufrechte der Applikation durch die Nutzer.

7.5.2. HANA Applikation

Dieses Kapitel beschreibt die zuvor dargestellten Pakete im Detail. Die Implementierung der einzelnen erstellten Objekte wird hierbei nur anhand von Codebeispielen aus der Applikation beschrieben. Der vollständige Code der HANA-Applikation ist auf der beigefügten CD zu finden.

data

Im Data-Package wurde lediglich eine Datenbank-Rolle (hdbrole) implementiert, da die Datenbanktabellen, Schema und Sequenzen aus dem Prototyp des Sensor-Szenarios verwendet werden (siehe Kapitel 5.3.2 auf Seite 100). Diese Rolle übergibt dem zugeteilten Nutzer alle benötigten Rechte für das verwendete Schema sowie die Ausführungsrechte aller erstellten Procedures (siehe Listing 7.1).

```
1  role abat.demosystem.smartmeter.FritzBox.data::smartmeter {
2  catalog schema "SENSORS_SCHEMA" : SELECT, INSERT, UPDATE, DROP, DELETE;
3  schema abat.demosystem.sensor.sensorapp.data:SENSORS_SCHEMA.hdbschema:
       SELECT, INSERT, UPDATE, DROP, DELETE;
4
5  catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getAllSmartmeter" : EXECUTE;
6  catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getCurrentValueForSmartmeter" : EXECUTE;
7  catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getChartValues" : EXECUTE;
8  catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getMinMaxAvg" : EXECUTE;
9  catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getEnergyConsumption" : EXECUTE;
10 catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getAllEnergy" : EXECUTE;
11 catalog sql object "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
       procedures::getAllEnergyByDevice" : EXECUTE;
12
13 catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.data
       ::SENSORS_DATA" : SELECT, INSERT;
14 catalog sql object "SENSORS_SCHEMA"."abat.demosystem.sensor.sensorapp.data
       ::SENSOR_ERROR_LOG" : SELECT, INSERT;
15 }
```

Listing 7.1: Echtzeit Stromanalyse: Quellcode - smartmeter.hdbrole

procedures

Für die Umsetzung dieses Szenarios wurden insgesamt sechs Procedures angelegt (siehe Abbildung 7.5 auf der nächsten Seite).

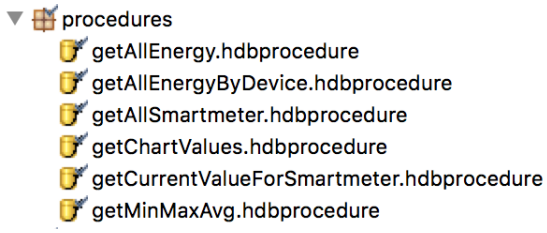


Abbildung 7.5.: Echtzeit Stromanalyse: Inhalt des procedures-Packages

Es handelt sich ausschließlich um Get-Procedures, da für das Speichern der Daten die Procedure aus dem Szenario Sensorik genutzt wird (siehe Kapitel 5.3.2 auf Seite 108). Ebenfalls nutzen die angelegten Procedures die Tabellen und das Model auf dem Szenario Sensorik. Durch diese Entscheidung wurde viel redundanter Code vermieden, als Nachteil wurde jedoch eine direkte Abhängigkeit dieses Szenarios vom Szenario der Sensorik erschaffen.
Im Folgenden wird die Implementierung der „getAllEnergyByDevice-Procedure“ im Detail beschrieben. Der Aufbau und die Vorgehensweise der anderen Procedures sind analog zu dieser und werden im Anschluss nur grob beschrieben.

```
1  PROCEDURE "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.procedures
       ::getAllEnergyByDevice" (
2    OUT output TABLE(LOCA VARCHAR(255), VAL VARCHAR(255)))
3    LANGUAGE SQLSCRIPT
4    SQL SECURITY INVOKER
5    DEFAULT SCHEMA "SENSORS_SCHEMA"
6    READS SQL DATA AS
7  BEGIN
8  /*****************************
9  Berechnung des gesamten Verbrauchs in kWH pro Sensor
10 *****************************/
11   output = SELECT LOCATION as LOCA, ((SUM(VALUE_02)/1000000)*10)/(3600) as
       VAL
12     FROM "_SYS_BIC"."abat.demosystem.sensor.sensorapp.models/
       AT_SENSOR_VALUES" DATA
13     WHERE S_TYPE = 'Smartmeter'
14     GROUP BY LOCATION
15     ORDER BY LOCATION ASC;
16 END
```

Listing 7.2: Echtzeit Stromanalyse: Quellcode - getAllEnergyByDevice.hdbprocedure

Der Output dieser Procedure ist eine Tabelle, bestehend aus der in der Datenbank hinterlegten Location, sowie dem berechneten Gesamtverbrauch (in kWh) eines Smartmeters (siehe Listing 7.2 Zeile 2). Für diese Berechnung werden zuerst alle Werte (Value02) sowie alle Locations aus dem im Szenario Sensorik erstellten Model mit dem Typ „Smartmeter“ abgefragt. Diese werden anschließen nach ihrer Location gruppiert und sortiert (siehe Listing 7.2 Zeilen 12 ff.).
Da die Stromwerte von der Datenbank in der Einheit mW gespeichert werden, wird für die korrekte Ausgabe der kWh noch eine Umrechnung benötigt. Zuerst werden alle Wer-

te in Watt umgerechnet (multipliziert mit 1000000). Zudem muss dieses Ergebnis mit 10 multipliziert werden, da die Werte nur alle 10 Sekunden in der Datenbank gespeichert werden und ansonsten Wertlücken entstehen würden. Diese Berechnung ist stark vereinfacht und ist daher nur als Näherung zu betrachten. Anschließend muss das Ergebnis noch durch 3600 geteilt werden, da eine Stunde 3600 Sekunden beinhaltet (siehe Listing 7.2 auf der vorherigen Seite Zeile 11). Als Endergebnis erhält man die bisher Verbrauchten kWh pro Smartmeter.
Alle weiteren implementierten Procedures werden im Folgenden kurz beschrieben. Eine Auflistung aller Procedures inklusive ihrer Input und Output Variablen ist in tablle 7.8 auf der nächsten Seite zu finden.

- **getAllEnergy:** Diese Procedure berechnet den Gesamtverbrauch aller Smartmeter in kWh.
- **getAllSmartmeter:** Gibt alle in der Datenbank vorhandenen Smartmeter aus.
- **getChartValues:** Diese Procedure gibt entweder alle Power-Werte oder alle Temperatur-Werte eines angegebenen Smartmeters für eine angegebene Zeitspanne aus.
- **getCurrentValueForSmartmeter:** Durch Aufruf dieser Procedure erhält man für den angegebenen Smartmeter die aktuellsten Werte jedes Channels.
- **getMinMaxAvg:** Diese Procedure gibt für den angegebenen Smartmeter den Maximal, Minimal und Durchschnittswert des angegebenen Values aus.

Tabelle 7.8.: Echtzeit Stromanalyse: Get-Procedures

Name	**Input**	**Output (Table/View)**
getAllEnergy		kWh VARCHAR(255)
getAllEnergyByDevice		loca VARCHAR(255), val VARCHAR(255)
getAllSmartmeter		SENSORS
getChartValues	cmd VARCHAR(255), sensor_id VARCHAR(255), from_Time TIMESTAMP	s_timestamp TIMESTAMP, s_channel VARCHAR(255), s_value VARCHAR(255)

Tabelle 7.8.: Echtzeit Stromanalyse: Get-Procedures

Name	Input	Output (Table/View)
getCurrentValueForSmartmeter	sensor_id VARCHAR(255)	s_timestamp TIMESTAMP, channel01 VARCHAR(255), val01 VARCHAR(255), unit01 VARCHAR(255), channel02 VARCHAR(255), val02 VARCHAR(255), unit02 VARCHAR(255), channel03 VARCHAR(255), val03 VARCHAR(255), unit03 VARCHAR(255)
getMinMaxAvg	cmd VARCHAR(255), sensor_id VARCHAR(255)	s_min VARCHAR(255), s_max VARCHAR(255), s_avg VARCHAR(255)

services

Das „Service“ Package beinhaltet alle erstellten Services, die für die Integration der Daten in die GUI verwendet werden (siehe Abbildung 7.6). Außerdem enthält das Paket ein weiteres Unterpaket, welches im Kapitel „pull“ genauer beschrieben wird (siehe Kapitel 7.6).

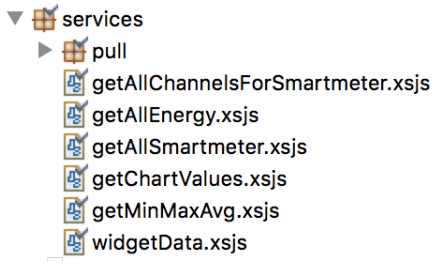


Abbildung 7.6.: Echtzeit Stromanalyse: Inhalt des Services Paket

Im folgenden wird die Implementierung des „widgetData“ Service beschrieben, da bisher kein Service für Kacheln beschrieben wurde. Die Implementierung der anderen erstellten Services ist analog zum Beschriebenen und bereits ausführlich im Kapitel 5.3.2 auf Seite 112 dargestellte. Diese Services werden anschließend nur in ihrer Funktionalität kurz erläutert.
Zum Abrufen der Daten für die Kachel, wird zuerst die Procedure „getAllEnergy“ aufgerufen (siehe Listing 7.3 auf der nächsten Seite).

```
28  // Eine DB-Verbindung zum Aufruf der Procedure aufbauen
29  var conn = $.db.getConnection();
30
31  var query = 'call "SENSORS_SCHEMA"."abat.demosystem.smartmeter.FritzBox.
      procedures::getAllEnergy" (?)';
32  var cst = conn.prepareCall(query);
33  var rs = cst.execute();
34  conn.commit();
35  rs = cst.getResultSet();
```

Listing 7.3: Echtzeit Stromanalyse: Quellcode - widgetData.xsjs (Procedure-Call)

Wie bei anderen Procedure-Calls wird auch hier zuerst eine Datenbankverbindung geöffnet (Zeile 29). Mit dieser wird dann die angegebene Query auf der Datenbank ausgeführt (Zeile 31 ff). Anschließend kann das Resultset abgerufen werden (Zeile 35). Dieses Resultset wird anschließend durchlaufen und das Ergebnis in einer Variable gespeichert (siehe Listing 7.4).

```
37 // Alle Werte des Resultsets speichern
38 while(rs.next()){
39   kwh = rs.getNString(1);
40 }
```

Listing 7.4: Echtzeit Stromanalyse: Quellcode - widgetData.xsjs (Resultset)

Eine Besonderheit der Kachel-Services ist, dass sie ausschließlich einen Wert ausgeben und diesen auch nicht in ein JSON Format konvertieren müssen. Eine Datenstruktur ist somit nicht vorhanden und der Body des Service beinhaltet lediglich eine Zahl (siehe Listing 7.5).

```
50 body = prepareString(Math.round(kwh * 1000) / 1000);
```

Listing 7.5: Echtzeit Stromanalyse: Quellcode - widgetData.xsjs (Body)

Um die durch die Procedure erhaltene Zahl für die Kachel aufzubereiten, wurde eine weitere Funktion implementiert, welche die erhaltene Zahl rundet und nach zwei Nachkommastellen trennt (siehe Listing 7.6).

```
5  /**
6  * Diese Funktion prepariert einen String und fuegt Nullen ans Ende
7  * @param value Der String
8  * @returns der preparierte String
9  */
10 function prepareString(value){
11   var valueString = value.toString();
12   var strings = valueString.split(".", 2);
13   var output = strings[0];
14   if(strings[1].length<3){
15     if(strings[1].length<2){
16       output = output + "." +  strings[1] + "00";
17     } else {
18       output = output + "." +  strings[1] + "0";
19     }
```

```
20    } else {
21      output = output + "." + strings[1];
22    }
23    return output;
24  }
```

Listing 7.6: Echtzeit Stromanalyse: Quellcode - widgetData.xsjs (prepareString)

Somit ist garantiert, dass die Kachel die abgefragten Werte ohne Probleme und Darstellungsfehler anzeigen kann. Im Folgenden wird die Funktionalität der weiteren Services kurz erläutert. Der gesamte Quellcode dieser ist auf der beigefügten CD zu finden.

- **getAllChannelsForSmartmeter:** Dieser Service gibt für den angegebenen Smartmeter alle Channels sowie ihre aktuellsten Werte (State, Power und Temperatur) aus.
- **getAllEnergy:** Durch diesen Service werden der berechnete Gesamtverbrauch, die CO2-Belastung sowie die einzelnen berechneten Ressourcen für die Startseite der GUI bereitgestellt.
- **getAllSmartmeter:** Dieser Service gibt alle in der Datenbank gespeicherten Smartmeter aus.
- **getChartValues:** Durch Aufruf dieses Services und mit der Angabe der gewünschten Stunden und des Sensors, gibt dieser Service alle vorhandenen Werte des angegebenen Channels für die angegebene Zeitspanne aus.
- **getMinMaxAvg:** Dieser Service gibt für einen angegebenen Channel eines Sensors den Minimal-, Maximal- und Durchschnittswert aus

pull

Das Pull Paket besteht aus insgesamt drei erstellten Implementierungsobjekten (siehe Abbildung 7.7).

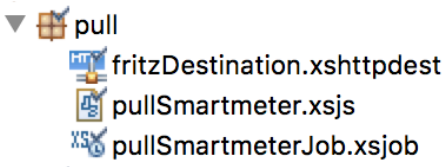


Abbildung 7.7.: Echtzeit Stromanalyse: Inhalt des Pull-Pakets

Für diesen Aufruf wurde, wie im Kapitel Sensorik, ebenfalls eine xshttpdest-Datei angelegt, die den Host, den Port sowie den gewünshcten Timeout der gewünschten Verbindung definiert (siehe Listing 7.7 auf der nächsten Seite).

```
host = "fritzboxstrom.abatgroup.de";
port = 80;
pathPrefix = "";
proxyType = none;
useSSL = false;
timeout = 3000;
```

Listing 7.7: Echtzeit Stromanalyse: Quellcode - fritzDestination.xshttpdest

Die meiste Logik zum Abrufen der Smartmeter-Werte ist im „pullSmartmeter.xsjs“ Service enthalten. Dieser Prozess des Abrufens war sehr aufwändig zu implementieren, da Fritzbox-interne Abrufabfolgen, Protokolle und Standards eingehalten werden mussten. Diese mussten zu beginn der Implementierung intensiv recherchiert und getestet werden. Das Abrufen der Werte teilt sich in insgesamt vier Teilschritte, die jeweils elementar für den Erfolg der abfrage sind (siehe Abbildung 7.8).

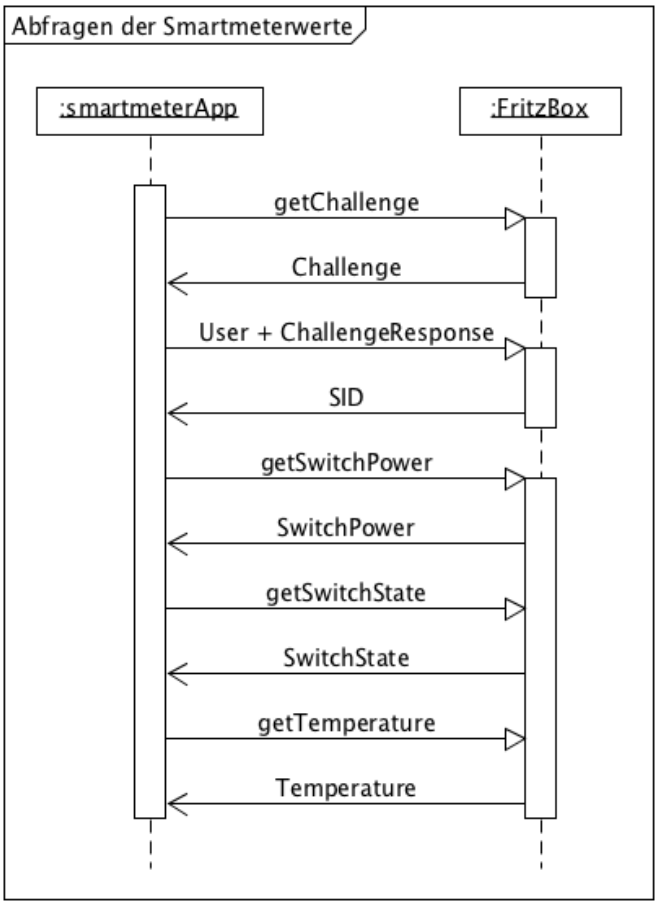


Abbildung 7.8.: Echtzeit Stromanalyse: Sequenzdiagramm zur Abfrage der Smartmeter-Werte

1. **Challenge generieren:** Zuerst muss eine so genannte Challenge durch die Fritzbox generiert und abgerufen werden. Dies geschieht durch das Aufrufen der zuvor angelegten Destination mit dem Adresszusatz „login_sid.lua“ (siehe Listing 7.8).

```
128   // Den WebRequest zum Abrufen der Challenge
129   clientChallenge = new $.net.http.Client();
130   reqChallenge = new $.web.WebRequest($.net.http.GET, "/login_sid.lua
        ");
131
132   // Den Request abschicken und die Anrwort speichern
133   clientChallenge.request(reqChallenge, destChallenge);
134   var responseChallenge = clientChallenge.getResponse();
135
136   //Den Challengestring rausfiltern und speichern
137   var xmlString = responseChallenge.body.asString();
138   challenge = getValue("Challenge", xmlString, 0);
139
```

Listing 7.8: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

Hier wird analog zu Szenario Sensorik (siehe Kapitel 5.3.2 auf Seite 117)ein Webrequest generiert und ausgeführt. Der Body der erhaltenen Antwort wird anschließend durch eine erstellte Funktion nach dem String „Challange“ durchsucht und gibt den gefundenen Wert zurück (siehe Listing 7.9).

```
91  function getValue(searchTag, xmlString, iteration){
92    // Der gefundene Wert
93    var value;
94    // Die Start- und Endposition
95    var startPos;
96    var endPos;
97    //Der Start- und End-Tag
98    var startTag = "<"+searchTag+">";
99    var endTag = "</"+searchTag+">";
100
101   var i = 0;
102   var tempString = xmlString;
103   //Zum X Wert durchlaufen
104   while(i<iteration) {
105     endPos = tempString.search(endTag) + endTag.length;
106     tempString = tempString.slice(endPos);
107     i++;
108   }
109   //Start- und Endpositonen suchen
110   endPos = tempString.search(endTag);
111   startPos = tempString.search(startTag) + startTag.length;
112   // Value aus dem String "schneiden"
113   value = tempString.slice(startPos,endPos);
114   return value;
115 }
```

Listing 7.9: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

2. **Challenge-Response generieren:** Wurde der Challenge-String korrekt aus der Antwort extrahiert, so kann dieser zur Generierung der Challenge-Response verwendet werden. Um die Response den Anforderungen und Standards der Fritzbox anzupassen, muss zuerst die Challenge mit dem Passwort des Fritzbox-Nutzers kombiniert werden (siehe Zeile 146 Listing 7.10).

```
146 // Die Challenge-Response generieren:
147 var challengeResponse = challenge + '-' + boxpwd;
148 var MD5 = convertStringToMD5(challengeResponse);
149 var finalChallengeResponse = challenge + "-" + MD5;
150
```

Listing 7.10: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

Anschließend muss der MD5-Hash des neu gebildeten Strings generiert werden. Dies wurde durch die implementierte Funktion „convertStringToMD5“ realisiert (Listing 7.11).

```
76 function convertStringToMD5(string) {
77   var buf = $.security.crypto.md5(str2ab(string));
78   var str = String.fromCharCode.apply(null, new Uint16Array(buf));
79   var hexString = convertToHex(str);
80   return convertString(hexString);
81 }
82
```

Listing 7.11: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

Zu Beginn der Konvertierung muss der String in ein ArrayBuffer transformiert werden, da die von SAP HANA bereitgestellte Funktion „$.security.crypto.md5“ einen Arraybuffer als Eingabe erwartet (siehe Zeile 77 Listing 7.11). Diese Transformation des zuvor erstellten Strings wurde durch die Funktion „str2ab“ gelöst (siehe Listing 7.12)

```
76   function str2ab(str) {
77     var buf = new ArrayBuffer(str.length*2); // 2 bytes for each
   char
78     var bufView = new Uint16Array(buf);
79     var u;
80     var strLen=str.length;
81     for (u=0; u < strLen; u++) {
82       bufView[u] = str.charCodeAt(u);
83     }
84     return buf;
85   }
86
```

Listing 7.12: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

Die „$.security.crypto.md5“ Funktion liefert als Ergebnis ebenfalls ein Arraybuffer des erstellten MD5-Hash. Dieser Buffer muss anschließend wieder in einen speziellen Hex-String gewandelt werden (siehe Zeile 78f. Listing 7.11). Da die Fritzbox

auch hier ein sehr spezielles Format des Strings erwartet mussten zudem zwei Funktionen implementiert werden, die diese Umwandlung des Strings vornehmen (siehe Listing 7.13 und 7.14).

```
76  function convertToHex(str) {
77    var hex = '';
78    var i;
79    for(i=0;i<str.length;i++) {
80      hex += str.charCodeAt(i).toString(16);
81    }
82    return hex;
83  }
84
```

Listing 7.13: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

```
57  function convertString(str) {
58    var output = '';
59    var i;
60    var array = str.match(/.{1,4}/g);
61    var arrayLength = array.length;
62    for (i = 0; i < arrayLength; i++) {
63      output = output + array[i].charAt(2);
64      output = output + array[i].charAt(3);
65      output = output + array[i].charAt(0);
66      output = output + array[i].charAt(1);
67    }
68    return output;
69    }
70  }
71
```

Listing 7.14: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

Besonders die Funktion „ConvertString“ war für eine erfolgreiche Abfrage ausschlaggebend, da die Fritzbox eine standard Byteorder ablehnt und der zuvor sortierte String wieder in seine Ursprungsform transformiert werden musste.

Wurde der MD5-Hash erfolgreich erstellt und korrekt transformiert, so muss als letzter Schritt der Response-Erstellung die anfangs erhaltene Challenge mit dem generierten MD5-Hash zusammengefügt werden (siehe Ziele 149 Listing 7.10 auf der vorherigen Seite).

3. **SID abfragen:** Mit der erstellten Challenge-Response kann nun die eigentlich benötigte SID abgefragt werden. Dies geschieht wieder durch einen einfachen Request der Destination unter Angabe des Users sowie der Response (siehe Listing 7.15)

```
153 // Den WebRequest zum Abrufen der Challenge
154 clientChallenge = new $.net.http.Client();
155 reqChallenge = new $.web.WebRequest($.net.http.GET, "/login_sid.lua
     ?username=" + boxuser + "&response=" + finalChallengeResponse);
156
```

```
157  // Den Request abschicken und die Anrwort speichern
158  clientChallenge.request(reqChallenge, destChallenge);
159  var response = clientChallenge.getResponse();
160
161  //Den Challengestring rausfiltern und speichern
162  var responseString = response.body.asString();
163  sid = getValue("SID", responseString, 0);
164
```

Listing 7.15: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs (SID)

Auch hier wird die Serverantwort wieder durch die erstellte Funktion „getValue“ durchsucht. Das Ergebnis ist die erhaltene SID von der Fritzbox.

4. **Werte abfragen:** Mit dieser SID können nun die verschiedenen Werte der einzelnen Smartmeter abgefragt werden. Diese Abfrage erfolgt ebenfalls durch einen Web-Request (siehe z.B. Listing 7.16).

```
242  // Den WebRequest zum Abrufen der Temperatur
243  var destChallenge = $.net.http.readDestination(destination_package,
       destinationChallenge_name);
244  var clientChallenge = new $.net.http.Client();
245  var reqChallenge = new $.web.WebRequest($.net.http.GET, "/
       webservices/homeautoswitch.lua?ain=" + ain + "&switchcmd=
       gettemperature&sid=" + sid);
246
247  // Den Request abschicken und die Anrwort speichern
248  clientChallenge.request(reqChallenge, destChallenge);
249  var responseChallenge = clientChallenge.getResponse();
250
251  //Die Temperatur speichern
252  temperature = responseChallenge.body.asString();
253
```

Listing 7.16: Echtzeit Stromanalyse: Quellcode - pullSmartmeter.xsjs

Hierbei muss dem eigentlichen Request zuerst die AIN des Smartmeters durch den Zusatz „/webservices/homeautoswitch.lua?ain=“ übergeben werden. Anschließend muss der jeweilige Befehl angegeben werden (z.B. „switchcmd=getswitchpower“ zum Abfragen des aktuellen Stromverbrauchs). Zuletzt wird die zuvor erstellte SID durch den Zusatz „sid=“ angegeben. Diese verifiziert den Aufruf durch die Fritzbox und verhindert so unbefugten Zugriff auf die Daten und Funktionen der Smartmeter. Weitere verwendete Funktionen sind „getswitchstate“ zum Abrufen des aktuellen Status eines Smartmeters und „gettemperature“ zum Abfragen der aktuellen Temperatur des Smartmeters.

Der somit erstellte Service wurde anschließend analog zum Pull-Services des Szenario Sensorik durch eine xsjob-Datei erweitert. Diese wird für den zyklischen Aufruf der erstellten Pull-Funktion benötigt und enthält die Logik und Parameter dieses Aufrufs (siehe Listing 7.17 auf der nächsten Seite).

```
{
  "description": "Pull SmartMeter Values",
    "action": "abat.demosystem.smartmeter.FritzBox.services.pull:
    pullSmartmeter.xsjs::pullSmartmeter",
    "schedules": [
    {
      "description": "Alle 10 Sec. abrufen",
      "xscron": "* * * * * * */10"
    }
  ]
}
```

Listing 7.17: Echtzeit Stromanalyse: Quellcode - pullSmartmeterJob.xsjob

Der Parameter „description“ definiert eine Beschreibung des Jobs, ist jedoch für die eigentliche Logik des Jobs irrelevant. Durch den „action“ Parameter wird der aufzurufende Service sowie die darin enthaltene Funktion definiert (hier der oben beschriebene Service mit der Funktion „pullSmartmeter“). Der parameter „schedules“ beschreibt abschließend den Aufrufzyklus. Durch die Angabe „* * * * * * */10“ wird der Job und damit die Funktion des Service alle 10 Sekunden ausgeführt.

uicatalog

Das „uicatalog“ Package enthält lediglich zwei Dateien (siehe Abbildung 7.9).

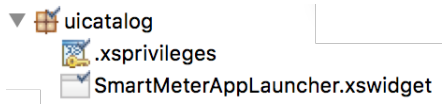


Abbildung 7.9.: Echtzeit Stromanalyse: Inhalt des Uicatalog Pakets

Die „.xsprivileges“ Datei enthält die Berechtigungen, welche zum Anzeigen der Kachel benötigt werden (siehe Listing 7.18).

```
{
  "privileges": [
    {
      "name": "SmartMeterAppLauncher",
      "description": "Access SmartMeterAppLauncher Widget"
    }
  ]
}
```

Listing 7.18: Echtzeit Stromanalyse: Implementierung der .xsprivileges (Kachel)

Die „SmartMeterAppLauncher.xswidget“ Datei enthält die gesamte Logik der erstellten Kachel (siehe Abbildung 7.10 auf der nächsten Seite). Auch diese Kachel wurde wie im Szenario Sensorik durch den Grafischen Editor erstellt. Dieser Vorgang wird im folgenden nicht weiter erläutert und ist in Kapitel 5.3.2 auf Seite 120 zu finden.

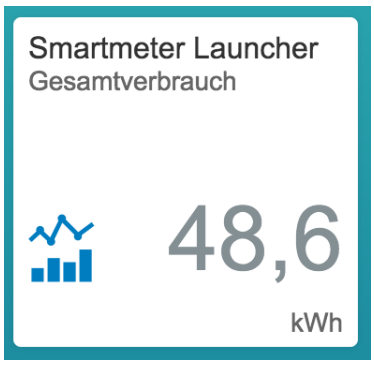


Abbildung 7.10.: Echtzeit Stromanalyse: Darstellung der erstellten Kachel

Die entscheidenden Parameter dieser Kachel sind die Service URL, welche den Service „widgetData“ anspricht[1], sowie die Ziel-URL, die auf die erstellte Index.hmtl Datei der SAPUI5 Applikation verweist [2].

7.5.3. Grafische Oberfläche

Dieser Abschnitt dokumentiert den Aufbau der grafischen Oberfläche. Bezugnehmend auf die Paketstruktur des Sensorikszenarios in Kapitel 5, werden in diesem Kapitel nur diejenigen Pakete näher beschrieben, welche wesentliche Unterschiede aufweisen. Darauf folgend werden alle funktionalen und nicht-funktionalen Bausteine schrittweise beschrieben, mit Screenshots zur besseren Darstellung verdeutlicht und wichtige Codefragmente erläutert.

Paketstruktur

Alle, für die Darstellung der grafischen Oberfläche notwendigen, Ressourcen und Dateien befinden sich im Paket *smartmeterui* im Pfad *abat/demosystem/smartmeter/FritzBox/ui/smartmeterui*. Wie bereits im Sensorikszenario stellt ein SAPUI5-Projekt den Grundbaustein für die Anzeige dar. Die Paketstruktur des Szenarios der Echtzeit Stromanalyse (siehe Abbildung 7.11) beinhaltet größtenteils die selbe Struktur wie die des Sensorikszenarios. Alle notwendigen Bausteine werden gleichermaßen im Paket WebContent gegliedert, welche zusätzlich noch das Paket *licences* beinhaltet.

[1] Verwendete URL: `http://hana-d21.abatgroup.de:8002/abat/demosystem/smartmeter/FritzBox/services/widgetData.xsjs`

[2] Verwendete URL: `http://hana-d21.abatgroup.de:8002/abat/demosystem/smartmeter/FritzBox/ui/smartmeterui/WebContent/index.html`

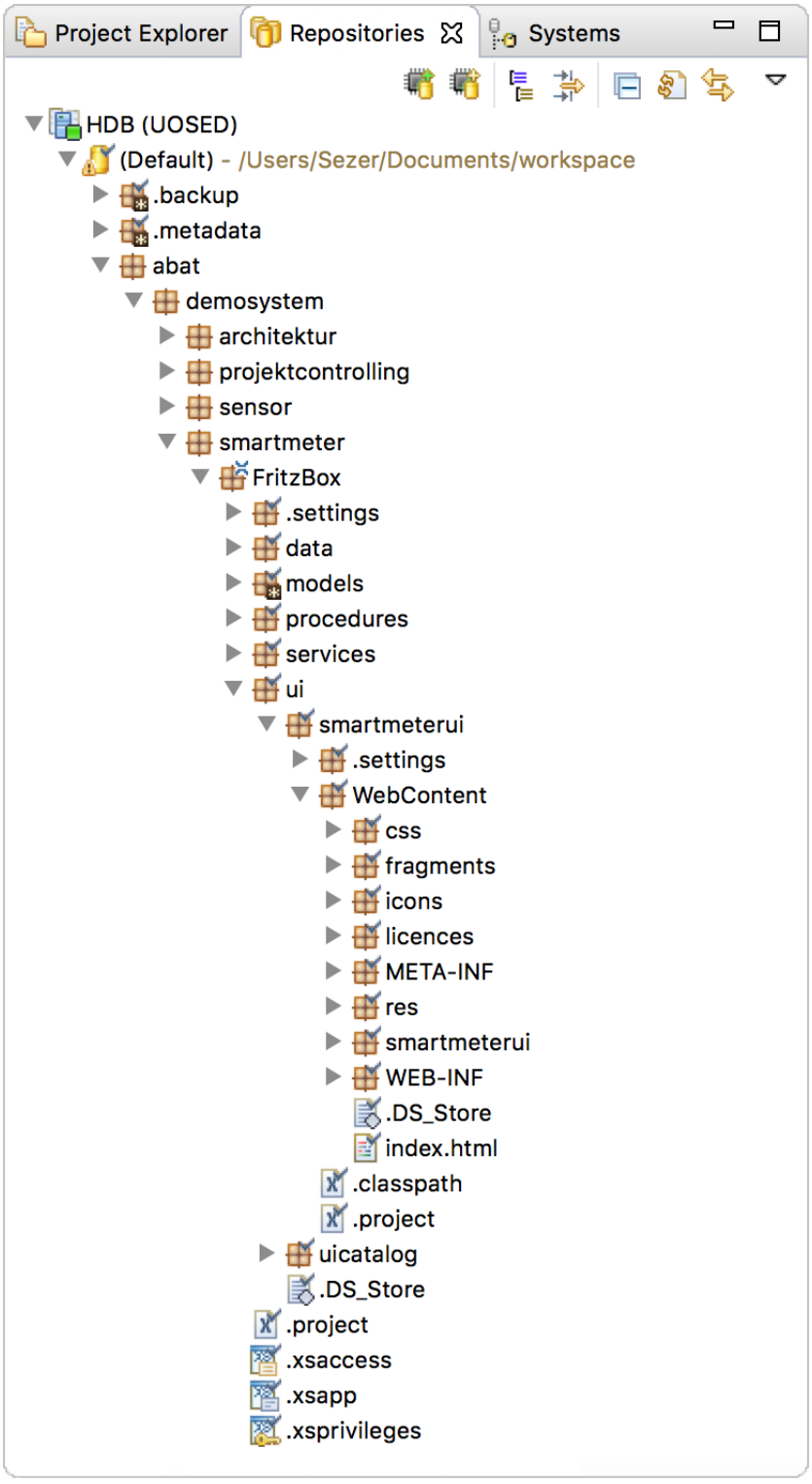


Abbildung 7.11.: Echtzeit Stromanalyse: Projektpfad zu den UI-Ressourcen

css Die Beschreibung dieses Pakets ist analog zu dem im Szenariosensorik. Das Paket beinhaltet eigene Font Definitionen zum Darstellen von anwendungsspezifischen Icons, die nicht in den SAPUI5-Bibliotheken enthalten sind. CSS-Befehle zur Realisierung spezifischer grafischer Darstellungen werden in der style.css-Datei definiert. Die selektierten HTML-Elemente werden anhand von Kommentaren innerhalb dieses Stylesheets beschrieben.

fragments Die Beschreibung dieses Pakets ist analog zu dem im Szenariosensorik. Im Rahmen dieses Szenarios wurden drei Fragmente (siehe Abbildung 7.12) implementiert und genutzt, welche innerhalb der Dateien licenceInformationPopover.fragment.js, tilesInformationPopover.fragment.js und tilesInformationChartPopover.fragment.js definiert werden. Wie auch im Sensorikszenario muss das fragments-Paket als lokale Ressource innerhalb der index.html-Datei angegeben werden. Ansonsten sind die Fragmente nicht nutzbar, da bei der Nutzung nicht auf sie zugegriffen werden kann.

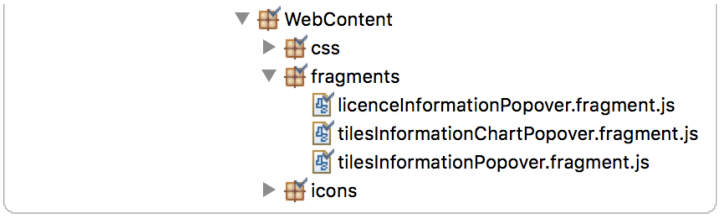


Abbildung 7.12.: Echtzeit Stromanalyse: Inhalt des fragments-Pakets

- **licenceInformationPopover.fragment.js**
  In diesem Fragment ist eine View definiert, die auf der Detailansicht des Startbildschirms zur Angabe von Lizenzinformationen genutzt wird (siehe Abbildung 7.13). Die genutzten Icons verfügen über eine Creative Commons (CC) 3.0 Lizenz. Bei dieser Lizenzart muss der Urheber genannt und mittels einer Uniform Resource Locator (URL) zur entsprechenden Lizenz verwiesen werden.

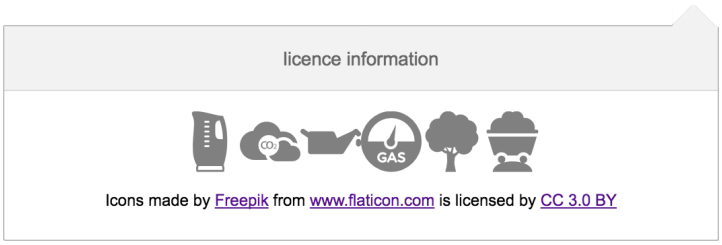


Abbildung 7.13.: Echtzeit Stromanalyse: Fragment für Lizenzinformationen

- **tilesInformationChartPopover.fragment.js**
  In diesem Fragment ist eine View definiert, die auf der Detailansicht des Startbildschirms genutzt wird. Das Fragment wird angezeigt, sobald diejenige Kachel ausgewählt wird, welche den Gesamtverbrauch aller verfügbaren Geräte darstellt.

Der angezeigte Informationsbereich enthält eine Donut-Chart (siehe Abbildung 7.14), die den Gesamtverbrauch auf die jeweiligen Geräte aufteilt.

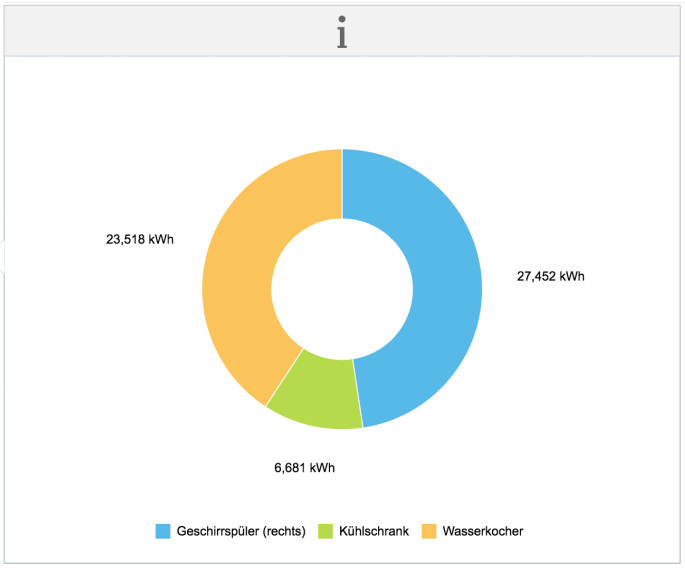


Abbildung 7.14.: Echtzeit Stromanalyse: Fragment zum Darstellen einer Donut-Chart

- **tilesInformationPopover.fragment.js**
  In diesem Fragment ist eine View definiert, die auf der Detailansicht des Startbildschirms genutzt wird. Das Fragment wird angezeigt, sobald eine Informationskachel, außer die des Gesamtverbrauchs, ausgewählt wird. In dem angezeigten Informationsbereich (siehe Abbildung 7.15) kann die Formel eingesehen werden, durch die die Angabe auf der Kachel berechnet wird.

Abbildung 7.15.: Echtzeit Stromanalyse: Fragment zum Anzeigen von Vergleichsinformationen

icons Analoge Beschreibung zum Sensorikszenario im Abschnitt icons aus Kapitel 5.

META-INF Analoge Beschreibung zum Sensorikszenario im Abschnitt META-INF aus Kapitel 5.

res Analoge Beschreibung zum Sensorikszenario im Abschnitt res aus Kapitel 5. Das res-Paket hat im Rahmen dieses Szenarios jedoch keinen weiteren Inhalt, da keine zusätzlichen Ressourcen genutzt wurden.

smartmeterui Ähnlich wie im Sensorikszenario (siehe Abschnitt ui-Paket, Sensorik) befinden sich in diesem Szenario die elementarsten Bestandteile, um eine grafische Schnittstelle zum Anwender aufbauen und steuern zu können, im smartmeterui-Paket. Dabei wird die Anzeige selbst durch die initial.view.js-Datei repräsentiert. Die dazugehörige notwendige Logik wird im Controller innerhalb der initial.controller.js-Datei realisiert. Die View sowie der Controller sind in der Skriptsprache JavaScript implementiert, um so die Einheitlichkeit bei der Entwicklung zu gewährleisten. Im Laufe der folgenden Abschnitte wird die View als auch der Controller schrittweise dokumentiert, die grafische Oberfläche mithilfe von Screenshots untermauert und sensible Bereiche, die genauerer Betrachtung bedürfen, anhand von Quellcode erläutert.

WEB-INF Analoge Beschreibung zum Sensorikszenario im Abschnitt WEB-INF aus Kapitel 5.

index.html Analoge Beschreibung zum Sensorikszenario im Abschnitt WEB-INF aus Kapitel 5. Die index.html-Datei dieses Szenarios ist ebenfalls in einen Bootstrap-, Anwendungs- und UI-Bereich aufgeteilt. Dem Listing 7.19 ist zu entnehmen, dass die SAPUI5-Bibliotheken nicht lokal aus der Datenbank, sondern durch ein Content Delivery Network (CDN) (`https://sapui5.hana.ondemand.com/resources/sap-ui-core.js`) bezogen werden. Dies garantiert stets die Aktualität der Bibliotheken. Durch das Beziehen der Ressourcen von einem CDN setzt voraus, dass stets eine aktive Internetverbindung vorhanden sein muss. Ansonsten kann die gesamte grafische Oberfläche nicht instanziiert werden. Es wurde darauf geachtet, dass die Applikation im Rahmen des Szenarios so gestaltet ist, dass ein Verwenden auf unterschiedlichen Endgeräten mit verschiedenen Displaygrößen möglich sein soll. Es ist zu erwähnen, dass viele, aber lange nicht alle Darstellungselemente zur Verfügung stehen, welche das Responsive-Design gut umsetzten. Aus Performance-technischen gründen sind im Bootstrap-Bereich nur diejenigen Bibliotheken eingebunden worden, die auch wirklich zum Ausführen der Anwendung benötigt werden:

- **sap.m**
- **sap.viz**
- **sap.suite.ui.commons**
- **sap.ui.layout**

Alle oben aufgeführten Bibliotheken sind bereits im Sensorikszenario (siehe Abschnitt index.html in Kapitel 5) kurz erläutert worden. Es ergeben sich, mit Betrachtung des Szenarios der Echtzeit Stromanalyse, keine Unterschiede, sodass die verwendeten Bibliotheken nicht wiederholt beschrieben werden.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta http-equiv='Content-Type' content='text/html;charset=UTF-8'/>

    <!--Bootstrap-->
    <script src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js
    "
        id="sap-ui-bootstrap"
        data-sap-ui-libs="sap.m, sap.viz, sap.suite.ui.commons, sap.ui.
    layout"
        data-sap-ui-theme="sap_bluecrystal">
    </script>

    <!--Anwendungsbereich-->
    <script>
        sap.ui.localResources("smartmeterui");
        sap.ui.localResources("fragments");
        var app = new sap.m.App({initialPage:"idinitial1"});
        var page = sap.ui.view({id:"idinitial1", viewName:"smartmeterui.
    initial", type:sap.ui.core.mvc.ViewType.JS});
        app.addPage(page);
        app.placeAt("content");
    </script>

  </head>

  <!--UI-Bereich-->
  <body class="sapUiBody" role="application">
    <div id="content"></div>
  </body>
</html>
```

Listing 7.19: Echtzeit Stromanalyse: Quellcode - index.html

Im Anwendungsbereich werden lokale Ressourcen angegeben. Initial ist das smartmeterui-Package definiert, das die View und den Controller beinhaltet. Zusätzlich wird das fragments-Package eingebunden, die Start-View festgelegt und im UI-Bereich platziert. Der UI-Bereich stellt den tatsächlichen, für den Endanwender sichtbaren, Bereich dar.

Grundgerüst

Das Grundgerüst der GUI teilt sich in zwei zentrale Bereiche ein. Die allgemeine Präsentation der Anwendung gegenüber dem Endanwender gliedert sich in eine Master- sowie Detailansicht. Dabei umfasst die Aufgabe der Masteransicht zum einen das Navigieren innerhalb der Anwendung zu ermöglichen und zum anderen verfügbare Geräte und ihre jeweiligen Messeinheiten auszuwählen und anzuzeigen. Auf Abbildung 7.16 sieht man, dass diese Sicht eine gewisse Breite auf der linken Seite einnimmt, welche das Framework selbst festlegt.

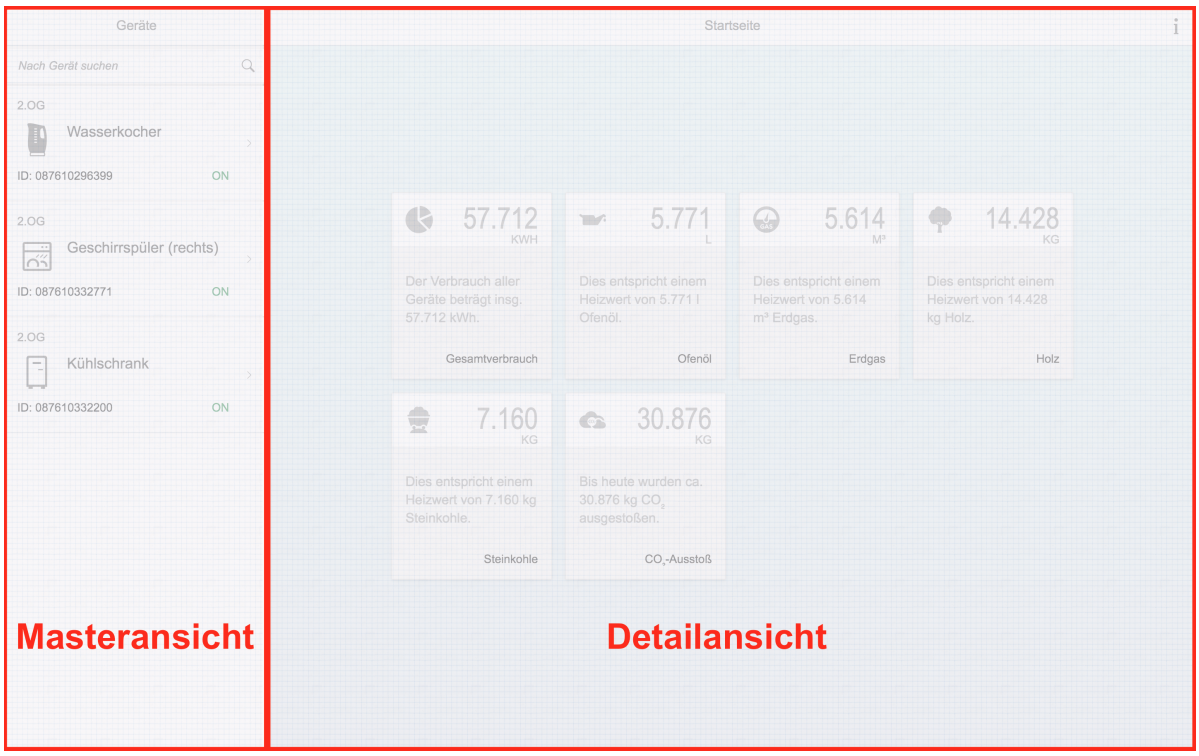


Abbildung 7.16.: Echtzeit Stromanalyse: Grundgerüst der GUI

Die Detailansicht wiederum befindet sich auf der rechts liegenden Seite. Die maßgebliche Interaktion zum Wechseln der angezeigten Informationen erfolgt beim Navigieren auf der linken Seite. Technisch realisiert wird diese Struktur durch das SplitApp-Control des SAPUI5-Frameworks. Die unterschiedlichen Ebenen beider Bereiche werden durch verschiedene Elemente realisiert, die dann dem SplitApp-Control hinzugefügt worden sind. Tabelle 7.9 zeigt eine grobe Übersicht über die obersten Bestandteile auf, die sich in diesem Control befinden. Wie auch im Sensorikszenario (siehe Listing 5.35) bildet dieses Element das oberste Control, in dem alle weiteren Controls hinzugefügt werden.

Tabelle 7.9.: Echtzeit Stromanalyse: Ebenen innerhalb der Master- und Detailstruktur

Paket:	abat/demosystem/smartmeter/FritzBox/ui/smartmeterui		
Datei:	initial.view.js		
	Variable:	**Control:**	**Zweck:**
Masteransicht:	oMasterDevices	sap.m.Page	Zur Darstellung von Geräten die an eine intelligente Stechdose angeschlossen sind

Tabelle 7.9.: Echtzeit Stromanalyse: Ebenen innerhalb der Master- und Detailstruktur

Paket:	abat/demosystem/smartmeter/FritzBox/ui/smartmeterui		
Datei:	initial.view.js		
	Variable:	**Control:**	**Zweck:**
	oMasterValues	sap.m.Page	Zur Darstellung von verfügbaren Messeinheiten (Stromverbrauch und Temperatur)
Detailansicht:	oDetailInitial	sap.m.Page	Zur Darstellung von errechneten Informationen in einer Kachel-Ansicht.
	oDetailPage	sap.m.Page	Zur Darstellung einer Detailansicht beim Auswählen einer Messeinheit.

Die konkrete strukturelle Zugehörigkeit für die jeweiligen Ansichten aus Tabelle 7.9 kann der Abbildung 7.17 entnommen werden. Diese stellt ebenfalls nur die obersten Komponenten sowie die dazugehörigen Fragmente aus dem Abschnitt fragments dar. Die genaue Beschreibung der darunterliegenden Controls erfolgt im weiteren Verlauf dieses Kapitels. Alle Controls aus der Tabelle 7.9 müssen, analog zum Listing 5.36 des Sensorikszenarios, dem SplitApp-Control hinzugefügt werden, das dann schlussendlich zur Anzeige der Anwendung zurückgegeben wird.

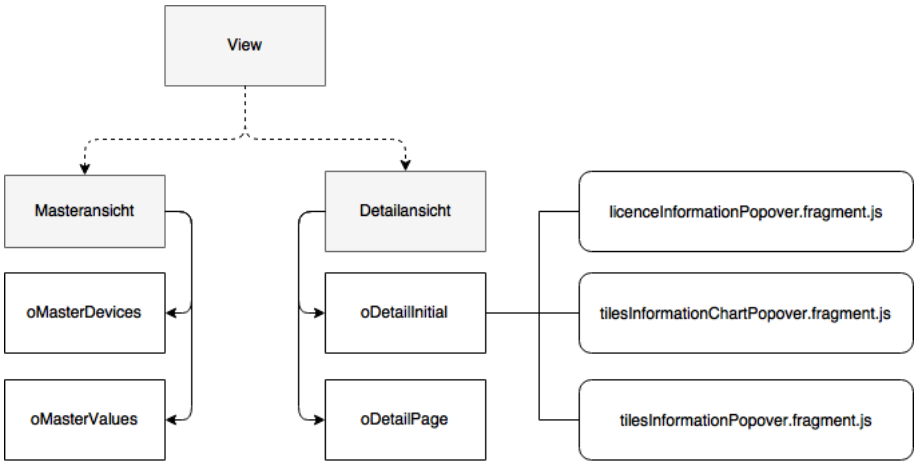


Abbildung 7.17.: Echtzeit Stromanalyse: Grundgerüst der GUI (Modell)

Datenanbindung

Als Datenlieferant für die grafischen Elemente werden die Services aus dem Abschnitt services verwendet. Dafür werden im Controller entsprechende JavaScript Object Notation (JSON)-Datenmodelle in der onInit()-Funktion erzeugt, der View bekannt gemacht und mit einer ID mittels der setModel()-Funktion versehen. Die onInit()-Funktion wird standardmäßig ausgeführt sobald das Rendern der Anwendung abgeschlossen ist und der Controller instanziiert wurde. Da die View jedes JSON-Modell kennt, wird die ID zum einen für das Data-Binding der Controls verwendet und zum anderen können die Datenmodelle direkt beim Aufrufen der Services angesprochen und mit Daten befüllt werden.
Listing 7.20 gibt einen Überblick über alle verwendeten Models. Welche Controls, welche Services und Models nutzen, erfolgt im jeweiligen Dokumentationsabschnitt der entsprechenden Controls.

```
34  // Erzeugen von Datenmodellen (JSON)
35
36     var oDeviceModel = new sap.ui.model.json.JSONModel();
37     var oValuesModel = new sap.ui.model.json.JSONModel();
38     var oHeaderAttributesModel = new sap.ui.model.json.JSONModel();
39     var oChartModel = new sap.ui.model.json.JSONModel();
40     var oTilesModel = new sap.ui.model.json.JSONModel();
41     var oIconLicencesModel = new sap.ui.model.json.JSONModel(lPath+"
       iconLicences.json");
42
43  // Datenmodelle werden der View bekannt gemacht und eine ID wird
       vergeben
44
45     this.getView().setModel(oDeviceModel,'deviceModel_1');
46     this.getView().setModel(oValuesModel,'valuesModel_1');
47     this.getView().setModel(oHeaderAttributesModel,'
       headerAttributesModel_1');
48     this.getView().setModel(oChartModel,'chartModel_1');
49     this.getView().setModel(oTilesModel,'tilesModel_1');
50     this.getView().setModel(oIconLicencesModel,'iconLicencesModel_1');
```

Listing 7.20: Echtzeit Stromanalyse: Erzeugen der JSON-Models im Controller

Datenabruf mittels AJAX Um die Datenmodelle mit Daten zu befüllen, wird auch in diesem Szenario die Datenübertragungsmethode AJAX verwendet analog zum Sensorikszenario. Der Aufbau eines AJAX-Aufrufes ist im Abschnitt Datenabruf mittels AJAX im 5. Kapitel detailliert beschrieben. Alle AJAX-Aufrufe, welche im Rahmen dieser Dokumentation genutzt werden, sind entsprechend dem im Listing 5.38 angelegten Schema erstellt worden. Auf die genaue Beschreibung aller implementierten AJAX-Aufrufe wird verzichtet, da der Grundaufbau stets der gleiche wie im Listing 5.38 ist.

Zyklischer Datenabruf Das zyklische Erneuern der Messwerte von der intelligenten Steckdose ist ebenfalls mit der Datenübertragungsmethode AJAX implementiert wor-

den. Der Gesamtprozess gliedert sich dabei jeweils in eine *start*-, *poll*- und *stop*-Funktion, welche im Abschnitt Zyklischer Datenabruf bereits detailliert beschrieben werden und analog zu diesem Szenario aufgebaut sind. Die start-Funktionen leiten den zyklischen Aktualisierungsvorgang ein, die poll-Funktionen beinhalten die Abruf-Logik und die stop-Funktionen beenden den zyklischen Abruf. Die genaue Implementierung eines zyklischen Abrufs kann im Abschnitt Zyklischer Datenabruf genauer betrachtet werden.
Die Abbildungen 7.18 bis 7.20 zeigen eine kurze Übersicht über diejenigen Komponenten, deren Daten zyklisch abgerufen werden. Die unterste Ebene repräsentiert dabei jene Komponenten, bei denen das zugehörige Data-Binding stattfindet.

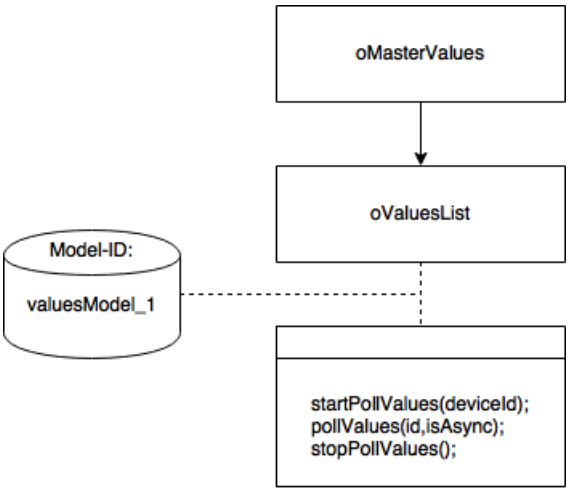


Abbildung 7.18.: Echtzeit Stromanalyse: Zyklischer Datenabruf oMasterValues

Abbildung 7.18 zeigt die einzelnen Komponenten, die in der Variablen oMasterValues gespeichert sind. Das Data-Binding findet in der untersten Ebene (oValuesList) innerhalb der View statt. Verwendet wird das Datenmodell mit der ID *valuesModel_1*. Die genaue Dokumentation erfolgt im Abschnitt oMasterValues.

- start-Funktion: startPollValues(deviceId);
- poll-Funktion: pollValues(id,isAsync);
- stop-Funktion: stopPollValues();

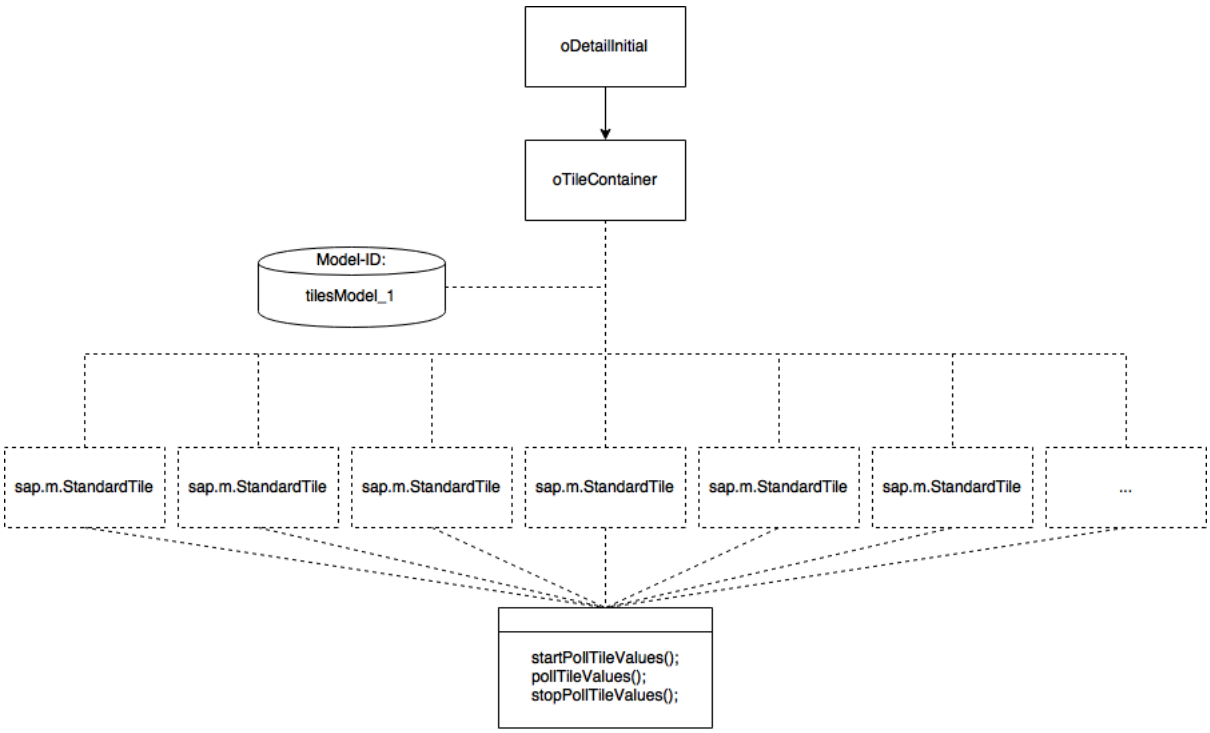


Abbildung 7.19.: Echtzeit Stromanalyse: Zyklischer Datenabruf oDetailInitial

Abbildung 7.19 zeigt die einzelnen Komponenten, die in der Variablen oDetailInitial gespeichert sind. Das Data-Binding findet in der untersten Ebene (sap.m.StandardTile) innerhalb der View statt. Verwendet wird das Datenmodell mit der ID *tilesModel_1*. Die Kacheln werden variable nach der Anzahl der JSON-Objekte mittels einer template-Funktion erzeugt. Je nach Anzahl der JSON-Objekte, die vom verwendeten Service zur Verfügung gestellt werden, kann die Anzahl der dargestellten Kacheln variieren. Die genaue Dokumentation erfolgt im Abschnitt oDetailInitial.

- start-Funktion: startPollTileValues();
- poll-Funktion: pollTileValues();
- stop-Funktion: stopPollTileValues();

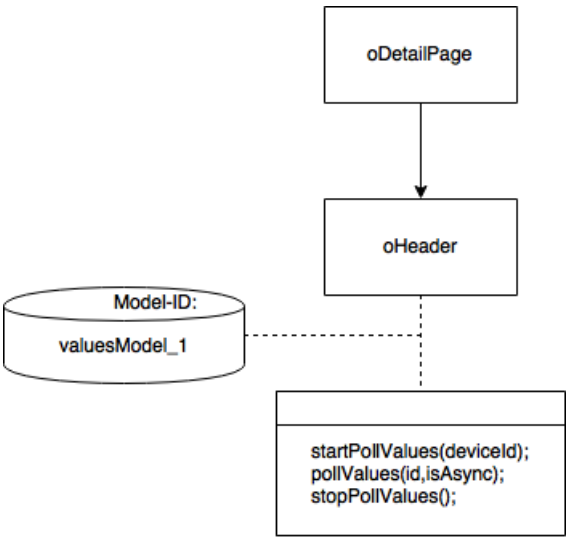


Abbildung 7.20.: Echtzeit Stromanalyse: Zyklischer Datenabruf oDetailPage

Die Abbildung 7.20 zeigt diejenigen Komponenten der oDetailPage, deren Eigenschaften zyklisch aktualisiert werden. Das Data-Binding findet auf der Ebene des ObjectHeader-Controls im Controller statt. In der Funktion bindHeader() werden die Eigenschaften des Controls an die Schlüssel-/Wert-Paare des Datenobjekts gebunden. Verwendet wird das Datenmodell mit der ID *valuesModel_1*. Die genaue Dokumentation dieser Komponente erfolgt im Abschnitt oDetailPage.

- start-Funktion: startPollValues(deviceId);
- poll-Funktion: pollValues(id,isAsync);
- stop-Funktion: stopPollValues();

Masteransicht

Dieser Abschnitt dokumentiert alle Elemente der Masteransicht (siehe Abbildung 7.16), die auf Abbildung 7.17 aufgeführt sind, im Detail. Neben den grafischen Elementen wird Bezug auf die wesentlichen Bestandteile der Logik genommen. Diese Ansicht dient hauptsächlich der Anzeige von verfügbaren Geräten und der Navigation zu den Messeinheiten der intelligenten Steckdose. Der in den folgenden Listings aufgezeigte Quellocde spiegelt schrittweise die wesentlichen Inhalte der initial.view.js- und initial.controller.js-Datei wieder.

oMasterDevices Der rötlich markierte Bereich in Abbildung 7.21 stellt denjenigen Bereich dar, dessen Control in der Variablen *oMasterDevices* gespeichert ist. Das Page-Control besitzt eine aggregierende Eigenschaft, die sich *content* nennt. Mittels dieser Eigenschaft können dem Control weitere Controls hinzugefügt werden, um so komplexe Anwendungen zu gestalten. Die verfügbaren Geräte, die an eine intelligente Steckdose

angeschlossen sind, werden durch die Verwendung des Services *getAllSmartmeter.xsjs* abgerufen, die in dem Datenmodell mit der ID *deviceModel_1* gespeichert werden.
Die anzuzeigenden Geräte werden als eine Art Liste in einem List-Control dargestellt. Als konkrete, einzelne Listenelemente, die dann ein Gerät darstellen, wird das ObjectListItem-Control verwendet. Wie in Listing 7.21 zu sehen ist, ist es entscheidend, dass zuerst das List-Control erzeugt wird (oDeviceList), dann die einzelnen Listenelemente an das Datenobjekt gebunden werden und anschließend dem Page-Control (oMasterDevices) übergeben wird. Zum Erzeugen der Listenelemente wird eine *template*-Funktion genutzt. Diese durchläuft iterativ den zuvor angegebenen Pfad und erzeugt für jeden Eintrag ein Listenelement (ObjectListItem-Control). Beim Auswählen eines Geräts wird die onDevicePress()-Funktion im Controller ausgeführt. Das korrekte Icon stellt die getIcon()-Funktion durch Übergeben der Gerätebezeichnung zur Verfügung.

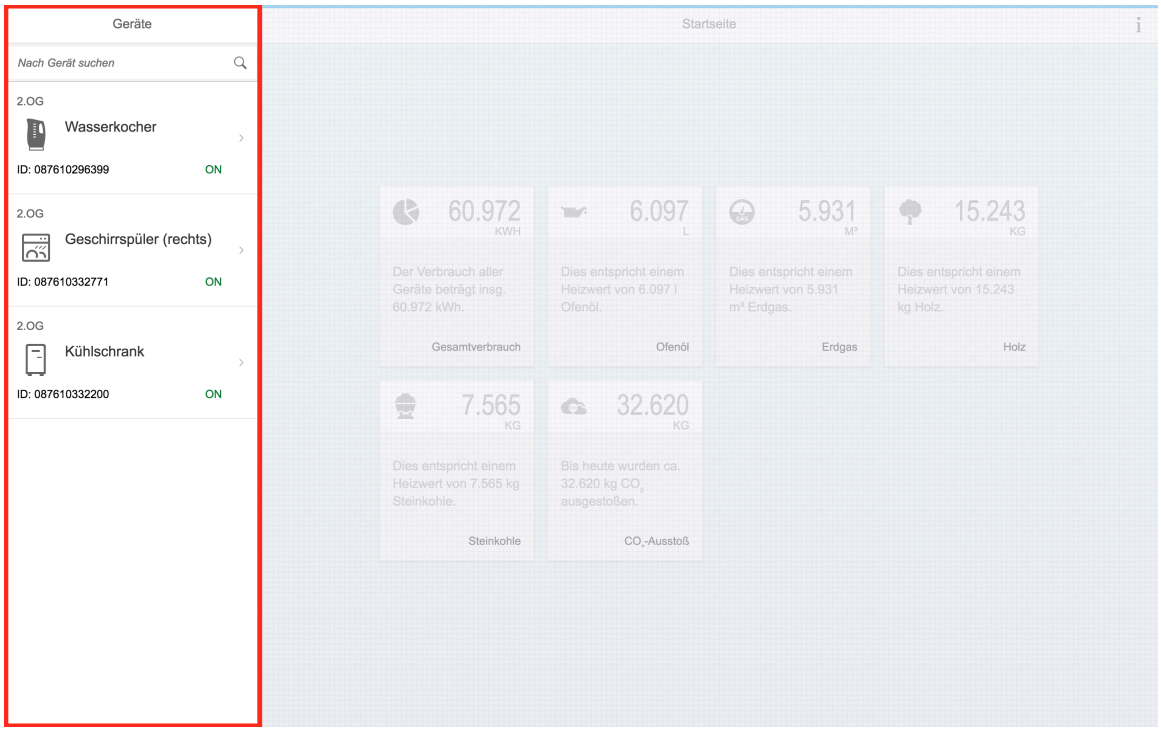


Abbildung 7.21.: Echtzeit Stromanalyse: Anzeige der Geräte an einer intelligenten Steckdose

```
17  /**
18   * Control zum Anzeigen der Geraete als Liste
19   */
20
21    var oDeviceList = new sap.m.List('deviceList_1',{});
22
23  /**
24   * Data-Binding der Geraete-Liste
25   */
26
```

```
27      oDeviceList.bindItems({
28        path: 'deviceModel_1>/smartmeter',
29        template: new sap.m.ObjectListItem({
30          title: '{deviceModel_1>type}',
31          intro: '{deviceModel_1>location}',
32          type: "Navigation",
33          press: [oController.onDevicePress,oController],
34          attributes: [
35            new sap.m.ObjectAttribute({
36              title: "ID",
37              text: '{deviceModel_1>sensor_id}'
38            })
39          ],
40          icon: {
41            path: 'deviceModel_1>type',
42            formatter: function(type){
43              icon = oController.getIcon(type);
44              return icon;
45            }
46          },
47          firstStatus: [
48            new sap.m.ObjectStatus({
49              text: {
50                path: 'deviceModel_1>state',
51                formatter: function(state){
52                  text = oController.getStateTextForDevice(state);
53                  return text;
54                }
55              },
56              state: {
57                path: 'deviceModel_1>state',
58                formatter: function(state){
59                  avState = oController.getStateForDevice(state);
60                  return avState;
61                }
62              }
63            })
64          ]
65        })
66      });
67
68    /**
69     * Control zum Anzeigen der Geraete (Master)
70     */
71
72      var oMasterDevices = new sap.m.Page('masterDevices_1',{
73        title: "Geraete",
74        content:[oDeviceList],
75        subHeader:[
76                  new sap.m.Toolbar({
77                    content:[
78                      new sap.m.SearchField({
```

```
79                      placeholder: "Nach Geraet suchen",
80                      liveChange:[oController.onLiveDeviceSearch,
    oController]
81                  })
82                ]
83              })
84          ]
85      });
```

Listing 7.21: Echtzeit Stromanalyse: Erzeugen der Geräte als Listenelemente mit anschließendem Data-Binding

oMasterValues Der rötlich markierte Bereich in Abbildung 7.22 stellt denjenigen Bereich dar, dessen Control in der Variablen *oMasterValues* gespeichert ist. Das Page-Control besitzt eine aggregierende Eigenschaft, das als *content* bezeichnet wird. Mittels dieser Eigenschaft können dem Control weitere Controls hinzugefügt werden, um so komplexe Anwendungen zu gestalten. Wie im vorherigen Abschnitt auch, werden in diesem Abschnitt die Messeinheiten eines ausgewählten Geräts wieder durch eine Liste mittels des List-Controls (siehe Listing 7.22) dargestellt. Damit nun die richtigen Messwerte für die Temperatur und gegenwärtig verbrauchte Energie für ein ausgewähltes Gerät geladen werden können, wird die ID des Geräts dem Service *getAllChannelsForSmartmeter.xsjs* als Parameter beim Auslösen des zyklischen Datenabrufs durch die startPollValues(deviceId)-Funktion mitgegeben. Der Service nutzt das Datenmodell mit der ID *valuesModel_1*.

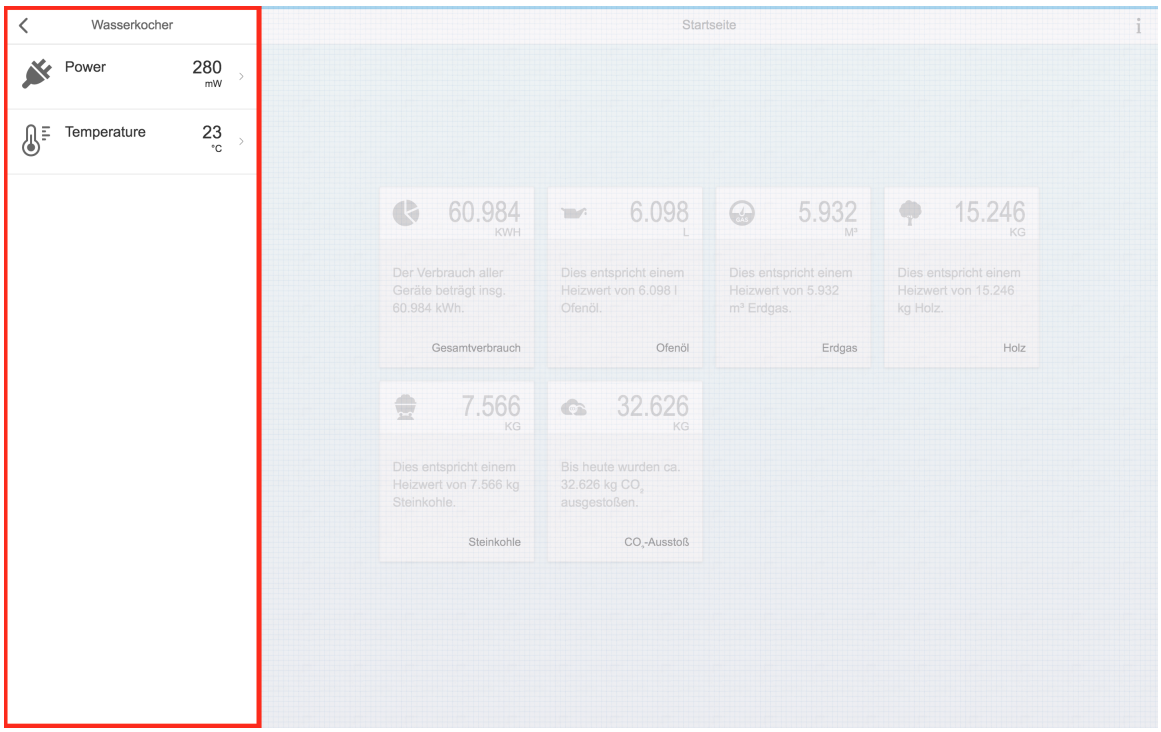


Abbildung 7.22.: Echtzeit Stromanalyse: Anzeige der Messeinheiten eines Geräts

```
99  /**
100  * Control zum anzeigen der Smartmeter-Werte als Liste
101  */
102
103   var oValuesList = new sap.m.List('valuesList_1',{});
104
105 /**
106  * Data-Binding der Smartmeter-Werte
107  */
108
109   oValuesList.bindItems({
110     path: 'valuesModel_1>/values',
111     template: new sap.m.ObjectListItem({
112       title: '{valuesModel_1>channel}',
113       number: '{valuesModel_1>value}',
114       numberUnit: '{valuesModel_1>unit}',
115       type: "Navigation",
116       icon: {
117         path: 'valuesModel_1>channel',
118         formatter: function(channel){
119           icon = oController.getIcon(channel);
120           return icon;
121         }
122       },
123       press: [oController.onChannelPress,oController]
124     })
125   });
126
127 /**
128  * Control zum Anzeigen der Smartmeter-Werte (Master)
129  */
130
131   var oMasterValues = new sap.m.Page('masterValues_1',{
132     title: "Messwerte",
133     showNavButton: true,
134     navButtonPress:[oController.onNavButtonPress,oController],
135     content:[oValuesList]
136   });
```

Listing 7.22: Echtzeit Stromanalyse: Erzeugen der Smartmeter-Werte als Listenelemente mit anschließendem Data-Binding

In Listing 7.22 ist zu erkennen, dass zu Beginn wieder ein List-Control erzeugt wird (oValuesList). Danach wird wieder mithilfe einer *template*-Funktion der angegebene Pfad des Datenobjekts iterativ durchlaufen und pro Eintrag ein Listenelement als ein ObjectListItem-Control erzeugt und ausgewählte Eigenschaften an das Datenmodell gebunden.

Detailansicht

Dieser Abschnitt dokumentiert die zwei Elemente der Detailansicht (siehe Abbildung 7.16), die auf Abbildung 7.17 aufgeführt sind, genauer. Neben den grafischen Elementen wird Bezug auf die wesentliche Logik genommen. Dieser Bereich der Ansicht dient der Darstellung von detaillierteren Informationen über die erhobenen Messwerte jedes einzelnen Geräts sowie allgemeine Vergleichsinformationen in Bezug zur verbrauchten Gesamtenergie in einer Kacheloptik. Zu Beginn jedes Abschnitts folgt zuerst eine Abbildung, die den dokumentierten Bereich grafisch darstellt und eine Orientierungshilfe gibt.

oDetailInitial

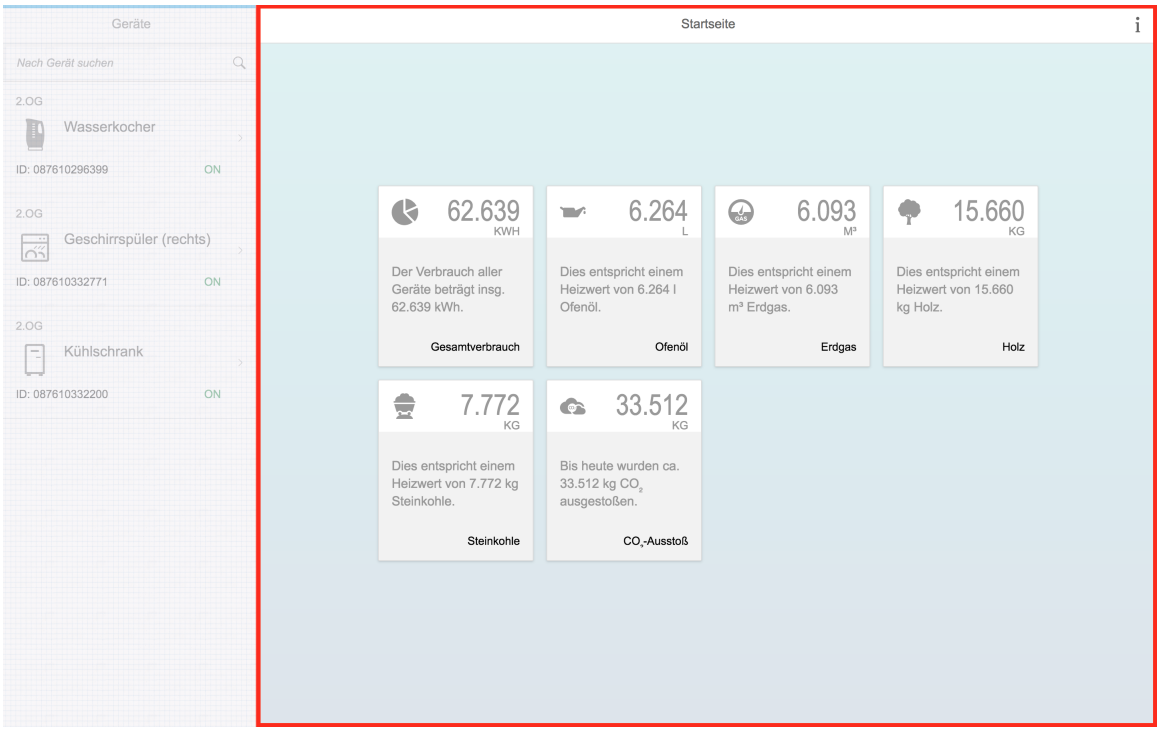


Abbildung 7.23.: Echtzeit Stromanalyse: Vergleichsinformationen zum Stromverbrauch in einer Kachelansicht

Abbildung 7.23 zeigt das Control, welches in der Variablen *oDetailInitial* gespeichert ist. Der strukturelle Aufbau kann der Abbildung 7.19 im Abschnitt Zyklischer Datenabruf entnommen werden. Dieses Control ist im SplitApp-Control *oSplitApp* als initiale Detailansicht festgelegt. Diese Detailansicht dient dazu, Vergleichsinformationen in Bezug auf die verbrauchte Energie anzuzeigen. Die Daten aktualisieren sich zyklisch. Zur Darstellung wurde eine Kachel-Ansicht ausgewählt, wobei jede Kachel durch ein StandardTile-Control repräsentiert wird. Datenlieferant ist der Service *getAllEnergy.xsjs* und nutzt das Datenmodell mit der ID *tilesModel_1*. Die Kacheln werden dynamisch nach der Anzahl

der Einträge im Datenobjekt mittels einer *template*-Funktion erzeugt. Als übergeordnetes Element wird, wie in Listing 7.23 aufgezeigt, ein TileContainer-Control verwendet. Dadurch wird sichergestellt, dass die Kacheln sich automatisch anordnen und sich verändernden Bildschirmgrößen anpassen.

```
155  /**
156   * Control, dass einen Container fuer die Kacheln darstellt
157   */
158
159    var oTileContainer = new sap.m.TileContainer('tileContainer_1',{
160      tiles:{
161        path: 'tilesModel_1>/values',
162        template: new sap.m.StandardTile({
163          title: {
164                  parts:[
165                          {path: 'tilesModel_1>channel'},
166                          {path: 'tilesModel_1>value'},
167                          {path: 'tilesModel_1>unit'}
168                  ],
169            formatter: function(channel,value,unit){
170              var desc = oController.getTileDescription(channel,value,unit
    );
171              return desc;
172            }
173          },
174          type: "Monitor",
175          number: '{tilesModel_1>value}',
176          numberUnit: {
177            path: 'tilesModel_1>unit',
178            formatter: function(unit){
179              if(unit != "m3"){
180                return unit;
181              }else{
182                return "m\u00b3";
183              }
184            }
185          },
186          icon: {
187            path: 'tilesModel_1>channel',
188            formatter: function(unit){
189              icon = oController.getIcon(unit);
190              return icon;
191            }
192          },
193          info: {
194            path: 'tilesModel_1>channel',
195            formatter: function(channel){
196              formChnl = oController.formatterChannel(channel);
197              return formChnl;
198            }
199
200          },
```

```
201             press: [oController.showTileCalc, oController]
202           })
203       }
204     });
```

Listing 7.23: Echtzeit Stromanalyse: Quellcode der dynamischen Kachelerzeugung (oDetailInitial)

Die Kacheln sind, im Gegensatz zum Sensorikszenario, auswählbar. Jede Kachel führt bei einem Press-Event die showTileCalc()-Funktion aus dem Controller aus. Im Listing 7.24 ist zu sehen, dass differenziert wird, ob die ausgewählte Kachel jene Kachel ist, die den Gesamtverbrauch anzeigt oder nicht. Wird diejenige Kachel angezeigt, welche die Information über den Gesamtverbrauch beinhaltet, so wird dann das Fragment anzeigt, der die Donut-Chart (siehe Abschnitt fragments) realisiert. Beim Auswählen aller anderen Kacheln wird das Fragment angezeigt, welches die Berechnung der Vergleichsinformation beinhaltet.

```
563   // Oeffnet das Fragment der Kachelinformationen. Wenn nicht "
        Gesamtverbrauch" Kachel ausgewaehlt wird: Content wird vorher immer
        geloescht damit immer
564   // ein neuer Content angezeigt werden kann und sich das nicht ueberlappt
        . Oeffnet das Chart-Fragment wenn "Gesamtverbrauch" ausgewaehlt wird,
        dessen Inhalt
565   // wird nicht geloescht da das Chart nur einmal initialisiert wird.
566
567   showTileCalc: function(oEvent){
568     this.getView().addDependent(this.oCalcDialog);
569     this.getView().addDependent(this.oCalcChartDialog);
570
571     var popup = this.oCalcInfoPopup;
572     var chartPopup = this.oCalcChartInfoPopup;
573
574     var src = oEvent.getSource();
575     var chnl = src.getInfo()
576
577     var calcText;
578
579     this.oCalcDialog.destroyContent();
580
581     calcText = this.getCalcDesc(chnl);
582
583     if(chnl != "Gesamtverbrauch"){
584       popup.addContent(this.getCalcInfo(calcText,chnl));
585       this.oCalcDialog.openBy(src);
586     }else{
587       chartPopup.addContent(this.getTileChart());
588       this.oCalcChartDialog.openBy(src);
589     }
590   },
```

Listing 7.24: Echtzeit Stromanalyse: Quellcode der showTileCalc()-Funktion das ein Fragment öffnet

Das TileContainer-Control, dass in der Variablen oTileContainer gespeichert ist, wird anschließend der aggregierenden Eigenschaft *content* des Page-Controls (oDetailInitial) übergeben.

oDetailPage

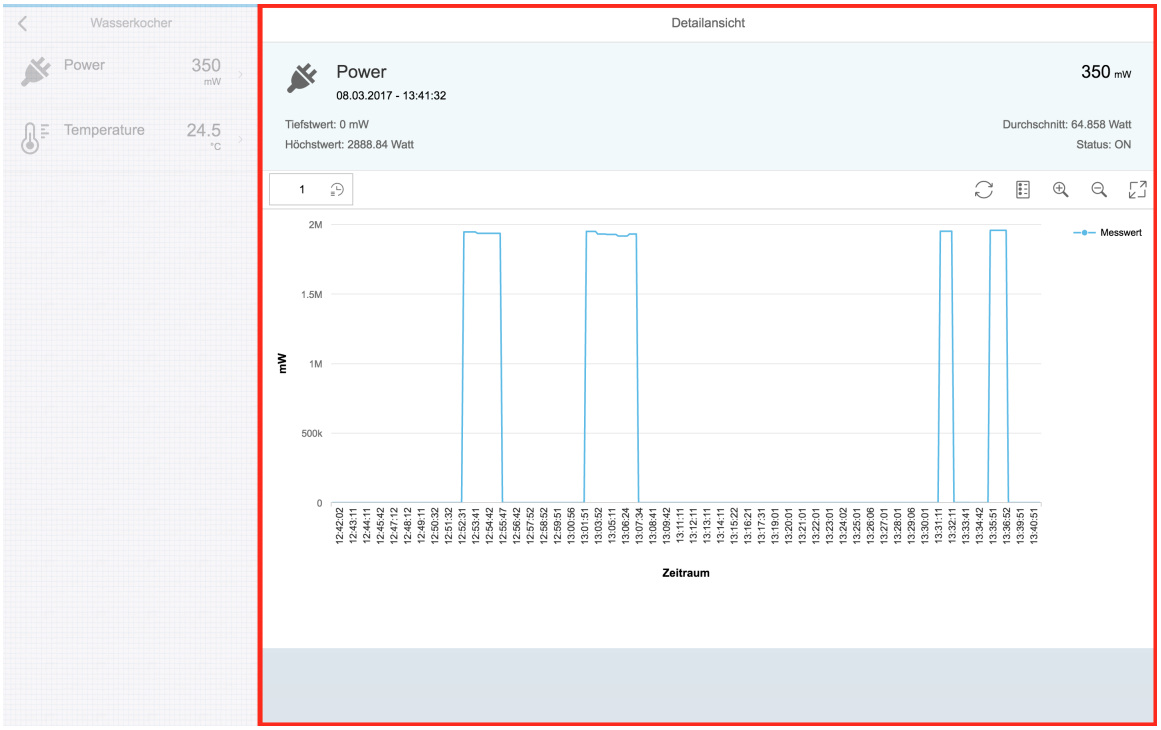


Abbildung 7.24.: Echtzeit Stromanalyse: Detailansicht einer Messeinheit (oDetailPage)

Abbildung 7.24 stellt das Control dar, welches in der Variablen *oDetailPage* gespeichert ist. Diese Detailansicht dient der genaueren Darstellung von Informationen und Messergebnissen über eine Messeinheit. Die aggregierende Eigenschaft dieses Page-Controls beinhaltet zum einen ein ObjectHeader-Control, das eine Kopfzeile darstellt (siehe Abbildung 7.25) und ein ChartContainer-Control zum Anzeigen von Messwerten in einer Chart (siehe Abbildung 7.26). Als Datenlieferant der Kopfzeile dient der Service *getAllChannelsForSmartmeter.xsjs* und nutzt das Datenmodell mit der ID *valuesModel_1*. Durch Mitgeben der Smartmeter-ID werden zudem der jemals gemessene Tiefstwert in milliwatt (mW), der Höchstwert und Durchschnitt in Watt (W), sowie der Status des Smartmeters angezeigt. Die setAttributeUnits()-Funktion in Listing 7.25 bestimmt die anzuzeigende Einheit.

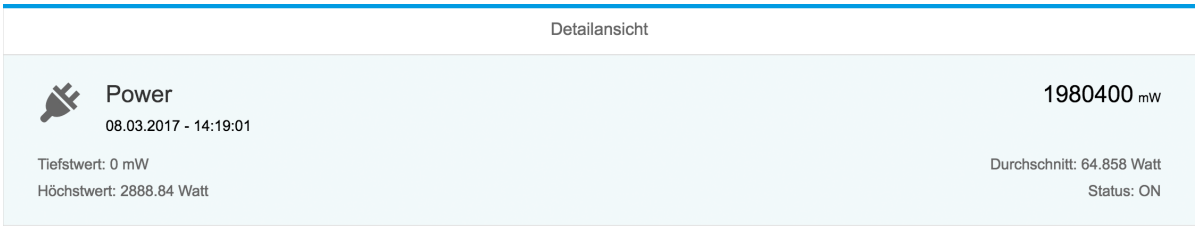


Abbildung 7.25.: Echtzeit Stromanalyse: Kopfzeile der Detailansicht (oDetailPage)

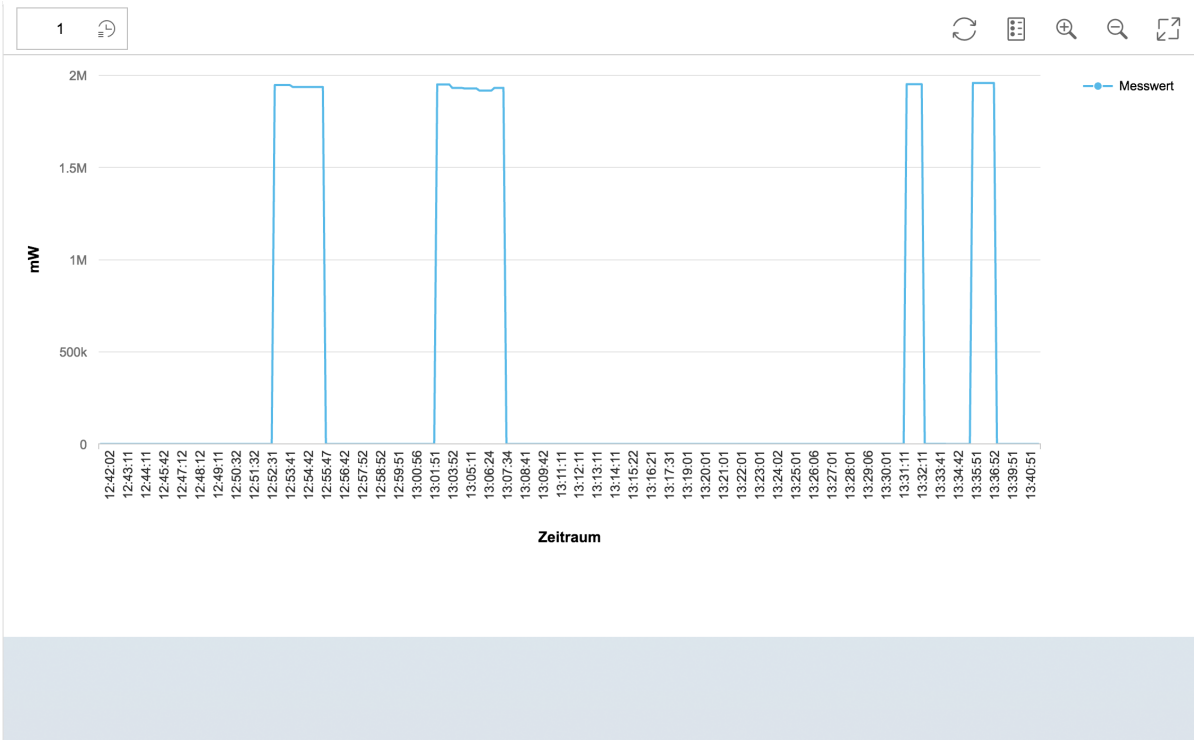


Abbildung 7.26.: Echtzeit Stromanalyse: Chart-Bereich der Detailansicht (oDetailPage)

Die Implementierung der Charts auf Abbildung 7.26 ist analog zu dem im Abschnitt oDetailPage des Sensorikszenarios. Als Datenlieferant in diesem Szenario wird der Service *getChartValues.xsjs* genutzt und verwendet das Datenmodell mit der ID *chartModel_1*. Der Service erwartet dabei drei Parameter: Die ID der Steckdose an, dem ein Gerät angeschlossen ist, die konkrete Messeinheit (Temperatur oder Power), von dem die Daten dargestellt werden sollen, und eine Stundenangabe. Durch die Stundenangabe liefert der Service die letzten Daten der angegebenen Stunde zurück. Die Implementierung der Chart ist analog zu dem des Sensorikszenarios und kann im Abschnitt oDetailPage nachgelesen werden. Die Stundenangabe (siehe Abbildung 7.27) wird durch ein TimePicker-Control realisiert.

```
349 // Setzt die Unit der Attribute im Header
350
351 setAttributeUnits: function(){
```

```
352     var text;
353     var title;
354     var newText;
355     var attributes = this.oHeader.getAttributes();
356
357     for(var i=0;i<attributes.length-1;++i){
358       text = attributes[i].getText();
359       title = attributes[i].getTitle();
360
361       if(this.selectedunit != "mW"){
362         newText = attributes[i].getText()+" "+this.selectedunit;
363       }else{
364         if(title != "Tiefstwert"){
365           newText = attributes[i].getText()+" Watt";
366         }else{
367           newText = attributes[i].getText()+" mW";
368         }
369       }
370
371       attributes[i].setText(newText);
372     }
373   },
```

Listing 7.25: Echtzeit Stromanalyse: Quellcode der setAttributeUnits()-Funktion das die Einheiten im Header setzt

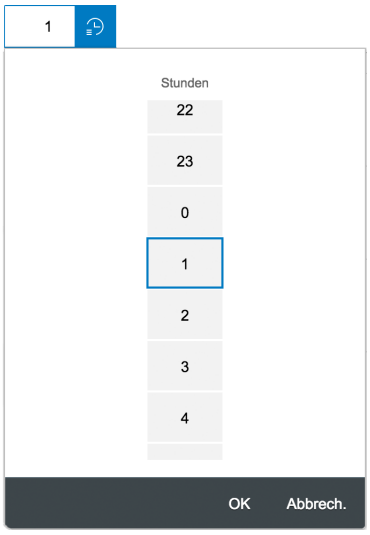


Abbildung 7.27.: Echtzeit Stromanalyse: Stundenauswahl durch ein TimePicker-Control (oDetailPage)

7.6. Abnahme

Der finale Abnahmetest wurde am Freitag den 10.03.2017 mit Nils Giesen und Jonas Schlemminger, von der abat AG, durchgeführt. Hierbei wurden alle zuvor festgelegten Anforderungen (siehe Kapitel 5.2) kontrolliert und begutachtet. Der erstellte Prototyp wurde offiziell abgenommen und übergeben.

Die Muss-Anforderungen wurden vollständig umgesetzt sowie abgenommen (siehe Tabelle 7.10). Zu den Anforderungen 2.1 “Datenbankschema erstellen“ und 2.2 “Tabellen anlegen“ kann angemerkt werden, dass die Datenbankschema und Tabellen von dem Szenario *Sensorik* übernommen wurden, da die Struktur der Daten der beiden Szenarien nahezu identisch war. Zu der Anforderungen 5.2 “Messdaten an Datenbank senden“ sei zudem anzumerken, dass das Datenbanksystem die Daten mit Hilfe eines Pull Verfahrens von der Fritzbox abgerufen hat.

Tabelle 7.10.: Echtzeit Stromanalyse: Abnahme - Muss-Anforderungen

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
1	**Hardware**		
1.1	Hardware auswählen	umgesetzt	abgenommen
1.2	Hardeware installieren	umgesetzt	abgenommen
2	**Datenmodell**		
2.1	Datenbankschema erstellen	von Szenario Sensorik genutzt/angepasst	abgenommen
2.2	Tabellen anlegen	von Szenario Sensorik genutzt/angepasst	abgenommen
3	**System Modular aufbauen**		
	Die Anwendung muss erweiterbar sein	umgesetzt	abgenommen
4	**Messung**		
4.1	Stromdaten messen	umgesetzt	abgenommen
4.2	Stromdaten aufbereiten	umgesetzt	abgenommen

Tabelle 7.10.: Echtzeit Stromanalyse: Abnahme - Muss-Anforderungen

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
4.2.1	Relevante Messwerte festlegen	umgesetzt	abgenommen
4.2.2	Granularität der Messwerte festlegen	umgesetzt	abgenommen
5	**Schnittstelle**		
5.1	Schnittstelle zum HANA System einrichten	umgesetzt	abgenommen
5.2	Messdaten an Datenbank senden	umgesetzt durch Pull Verfahren	abgenommen
5.3	Messdaten speichern	umgesetzt	abgenommen
6	**Darstellung der Daten**		
6.1	Stromverbrauch in SAPUI5 darstellen	umgesetzt	abgenommen
6.2	Stromverbrauch grafisch darstellen	umgesetzt	abgenommen

Die Soll-Anforderungen wurden bis auf die Anforderung 1.2 “Vergleichsdaten erheben“ vollständig umgesetzt sowie abgenommen (siehe Tabelle 7.11). Diese Anforderung wurde aus Zeitgründen nicht umgesetzt. Zudem steht diese in keinem direktem Zusammenhang mit anderen Muss- oder Soll-Anforderungen. Die Vergleichsdaten, also z.B. die verschiedenen Spülprogramme der Spülmaschine, sind jedoch in den Charts dieses Szenarios *Echtzeit Stromanalyse* durch die unterschiedlichen Werte erkennbar.

Tabelle 7.11.: Echtzeit Stromanalyse: Abnahme - Soll-Anforderungen

Nr.	**Anforderung:**	**Umgesetzt:**	**Abgenommen:**
1	**Datenanalyse**		
1.1	Messdaten auswerten	umgesetzt	abgenommen
1.2	Vergleichsdaten erheben	nicht umgesetzt	-

Tabelle 7.11.: Echtzeit Stromanalyse: Abnahme - Soll-Anforderungen

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
2	**Systemerweiterung**		
2.1	Weitere Geräte anschließen	umgesetzt	abgenommen
2.2	Testen	umgesetzt	abgenommen

Die Kann-Anforderungen wurde beide nicht umgesetzt (siehe Tabelle 7.12). Denn diese bauen auf der Soll-Anforderungen 1.2 "Vergleichsdaten erheben" auf welche aus Zeitmangel auch nicht umgesetzt wurde. Zudem waren diese Anforderungen nicht fest eingeplant sondern nur als Zusatz bei Zeitüberschuss gedacht.

Tabelle 7.12.: Echtzeit Stromanalyse: Abnahme - Kann-Anforderungen

Nr.	Anforderung:	Umgesetzt:	Abgenommen:
1	Vergleichsdaten auswerten	nicht umgesetzt	-
2	Verschiedenen Programme der Geräte grafisch darstellen	nicht umgesetzt	-

7.7. Fazit

Zusammenfassend kann gesagt werden, dass die Messung, Analyse und Darstellung von Stromdaten in der SAP HANA Umgebung erfolgreich war. Auch wenn die Umsetzung, vor allem bezüglich der Hardware, anders verlaufen ist als ursprünglich geplant. Es konnte jedoch trotzdem ein guter Überblick über die Stromdaten gegeben werden wodurch sich das Verständnis und Bewusstsein gegenüber Stromverbräuchen verbessern kann. Es zeigten sich im Laufe des Projektes zudem viele Parallelen zum Szenario *Sensorik* auf, wodurch bisherige Arbeit und Erfahrung vom Szenario *Sensorik* in diesem Szenario *Echtzeit Stromanalyse* genutzt werden konnten. Zudem war die Schnittstelle von der Fritzbox zu der HANA Datenbank komplexer als Anfangs vermutet, besonders im Zusammenhang mit Java Script, da hier einige Schritte notwendig sind welche ohne entsprechendes Hintergrundwissen sehr schwer nachzuvollziehen sind. Insgesamt konnte eine gute Grundlage für einen bewussten Stromverbrauch mit Darstellung im SAP

HANA System erstellt werden.

7.8. Ausblick

Dieses Szenario könnte für die Zukunft noch verbessert und weiterentwickelt werden. Es könnten z. B. weitere Funktionen realisiert werden, um diese Anwendung besser zu verwenden und Energie zu sparen.

7.8.1. Weitere mögliche Funktionen

Stromkosten zeigen

Entsprechende Stromkosten könnten berechnet werden, um aktuellen Kosten z. B. wieder in einem Line-Chart darzustellen. Es könnten auf der grafischen Oberfläche der aktuelle Stromverbrauch und die Stromkosten vom Geschirrspüler für seine verschiedenen Modi dargestellt werden. Bspw. wie viel Strom in einem bestimmten Modus verbraucht wurde oder wie viel Geld es gekostet hat den Wasserkocher einmalig zu benutzen.

Vergleichsdaten auswerten

Algorithmen könnten innerhalb der HANA-Plattform entwickelt werden, um den Stromverbrauch von bspw. verschiedenen Spülprogrammen der Spülmaschine oder verschiedene Getränke des Kaffeeautomaten zu erkennen und zu analysieren.

Verschiedenen Programme der Geräte grafisch darstellen

Entsprechende Detailansichten müssten für die grafischen Oberfläche erstellt werden, um die im vorherigen Punkt genannten Daten darzustellen, bspw. welches Spülprogramm gerade genutzt wird. Es wäre auch möglich, dass historische Daten anschaulich dargestellt werden, z.B. wie viel Strom von welchem Spülprogramm verbraucht wurde.

7.8.2. Energiesparendes Produzieren und Verwenden

Mit dieser Anwendung könnten die Benutzer Stromverbrauchsinformationen von einzelnen Geräten und deren Einsatzprogrammen jederzeit erfassen.
Unternehmen könnten überlegen, wie sie gleich bei der Fertigung von Waren stromsparende Geräte produzieren können, mit dem Fokus auf diejenigen Geräte oder Programme, die in der Regel deutlich mehr Strom verbrauchen.
Aus Verbrauchersicht könnte der Endanwender ökonomische Geräte oder Programme auswählen. Spürbare Kosteneinsparungen könnten den Verbraucher dazu bewegen, langfristig energiesparende Geräte oder ökonomisch effizientere Programme zu nutzen, um so auf Dauer Geld und Energie zu sparen.

8. Szenario - Produktnachhaltigkeit in SAP S/4HANA

Abstract

Nachhaltigkeit nimmt in der heutigen Wirtschaft eine immer größer werdende Rolle ein. Auch produzierende Unternehmen sind von diesem Wandel betroffen und müssen sich daher vermehrt mit der Nachhaltigkeit von Produkten beschäftigen. Dieses Konzept fokussiert sich auf SAP S/4HANA und zeigt auf, inwiefern Funktionen zum Management der Produktnachhaltigkeit in diesem System enthalten sind und wie gegebenenfalls Funktionen durch Customizing ergänzt werden können.

8.1. Einleitung

Im Zuge der stetig wachsenden Bedeutung der Nachhaltigkeit in der Gesellschaft, hat der Begriff Nachhaltigkeit auch in die Wirtschaft Einzug gefunden. Nachhaltigkeit kann unterschiedlich definiert werden, jedoch beinhaltet der Begriff immer eine soziale, eine ökologische und eine ökonomische Dimension. Jeder dieser Dimensionen sollten Unternehmen im täglichen Geschäft beachten. In einer Umfrage der Unternehmensberatung McKinsey aus dem Jahr 2011 [1] wurde gezeigt, dass Unternehmen in ihren verschiedensten Organisationseinheiten und Geschäftsprozessen Ziele der Nachhaltigkeit verfolgen. Auffällig ist, dass über 60% der Befragten Unternehmen angaben, dass in den täglichen Geschäftstätigkeiten versucht wird, Energie oder auch Abfall zu reduzieren. Dies kann insbesondere für das produzierende oder verarbeitende Gewerbe von großer Relevanz sein. Eine Statistik des statistischen Bundesamtes hat zudem ergeben, dass in Deutschland 93% der Unternehmen mit über 250 Mitarbeitern ERP-Software einsetzen [2]. Im verarbeitenden Gewerbe ergibt sich sogar ein Anteil von 98%. Durch Verbindung dieser beiden Studienergebnisse, lässt sich schließen, dass Vorteile entstehen würden, wenn ERP-Systeme die Nachhaltigkeitsaktivitäten eines Unternehmens durch entsprechende Module unterstützen könnten. Häufig bieten ERP-Systeme in diesem Bereich nur Module und Funktionen aus dem EHSQ-Bereich (siehe auch SAP EHS). Eine vollständige und umfassende Abbildung eines Nachhaltigkeitsmanagement ist oftmals nicht enthalten.

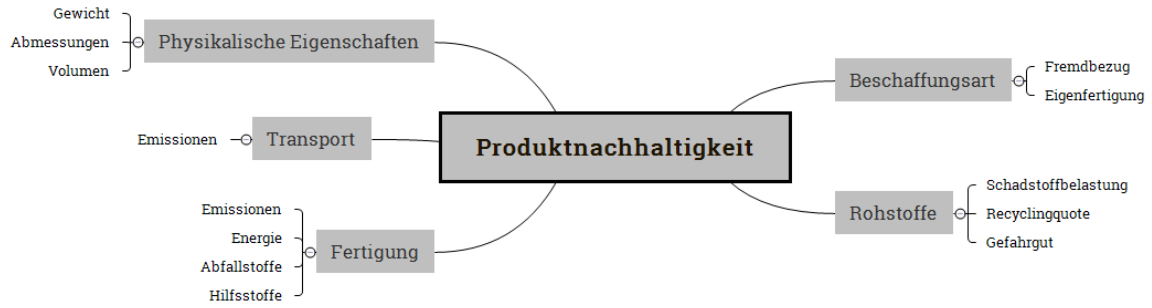


Abbildung 8.1.: Produktnachhaltigkeit: Daten im Umfeld der Produktnachhaltigkeit

8.2. Zielbestimmung

Dieses Konzept soll sich mit der Abbildung von ökologischen und sozialen Aspekten im SAP S/4HANA auseinandersetzen. Konkret wird hierbei die Produktnachhaltigkeit fokussiert, welche unter anderem durch die Recyclingquote bestimmt werden kann. In dem anzufertigen Konzept soll aufgezeigt werden, wie SAP S/4HANA diesen Anwendungsfall unterstützen kann. So muss evaluiert werden, wo das System bereits nutzbare Informationen bietet und wie diese gepflegt bzw. verwendet werden können. Des Weiteren soll geprüft werden, wie ökologische Informationen entsprechend erweiterbar sind.
Im zweiten Schritt würde eine Kachel im System implementiert werden, welche nach Eingabe der Materialnummer die Materialzusammensetzung ausgibt (ebenfalls könnte angezeigt werden, zu welchem Anteil das Material aus Fremdbezug stammt). Die Implementierung der Kachel wird vorerst als optional angesehen.

8.3. Daten

Im Zuge der Recherche zu diesem Konzept wurden Daten identifiziert mit denen Aussagen über die Nachhaltigkeit eines Produktes getroffen werden können bzw. mit denen die Nachhaltigkeit eines Produktes analysiert werden kann. Diese Daten werden im Folgenden näher beschrieben. Zur Übersicht wurde auch eine MindMap (siehe 8.1) erstellt.
Grundsätzlich muss bei der Beschreibung von Daten der ökologischen Nachhaltigkeitsdimension zwischen Umweltdaten und umweltrelevanten Daten unterschieden werden. In SAP S/4HANA werden beide Begriffe häufig fälschlicherweise synonym verwendet. Umweltdaten haben immer einen direkten Bezug zu Eigenschaften der Medien Luft, Wasser oder Erde. Mit den Daten kann somit beispielsweise der Zustand dieser Medien beurteilt werden (z.B. CO2-Gehalt in der Luft). Umweltrelevante Daten haben dagegen nur einen mittelbaren Bezug zu Umweltwirkungen. Somit haben diese Daten bzw. die damit verbundenen Folgen einen Einfluss auf die Umwelt (z.B. Emissionsausstoß einer Maschine).
Die *Physikalischen Eigenschaften* umfassen das Volumen und Gewicht sowie die Abmessungen eines Produktes. Diese Daten können relevant für weitere Kennzahlen zur

Ermittlung der Produktnachhaltigkeit sein. So kann zum Beispiel das Gewicht eines Produktes maßgeblichen Einfluss auf die ausgestoßenen Emissionen beim Transport nehmen. Im System können die physikalischen Eigenschaften von Produkten direkt im Materialstamm gepflegt werden. Hierzu kann die SAP-Transaktion MM02 genutzt werden. In der Sicht „Grunddaten1“ ist es möglich Parameter wie „Bruttogewicht“, „Nettogewicht“, „Volumen“ und Angaben zur Abmessung des Produktes anzugeben.
Bei der *Beschaffungsart* muss eine Unterteilung in Fremd- und Eigenfertigung vorgenommen werden. Dies ist von besonderer Relevanz, da bei einem fremdbeschafften Material nachvollzogen werden muss, welche einzelnen Materialien bzw. Rohstoffe verwendet worden sind um eine konkrete Aussage über die Produktnachhaltigkeit treffen zu können. Die Beschaffungsart kann ebenfalls aus dem Materialstamm entnommen werden. Unter der SAP-Transaktion MM02 in der Sicht „Disposition2“ ist es möglich die Beschaffungsart zu pflegen. Hierbei ist eine Auswahl zwischen Eigenfertigung oder Fremdbeschaffung möglich.
Beim *Transport* von Produkten entstehen Emissionen, die Berücksichtigung bei der Betrachtung der Produktnachhaltigkeit finden müssen. Hierbei ist sowohl der Transport von Rohstoffen und Halbfabrikaten zum Produktionswerk gemeint als auch die Auslieferung des Endproduktes zum Kunden. SAP S/4HANA stellt keine Möglichkeit bereit, konkrete Emissionswerte, die beim Transport enstehen, im Datenstamm zu pflegen.
Um die Produktnachhaltigkeit zu bestimmen ist eine Berücksichtigung des *Fertigungsprozesses* notwendig. So müssen zum Beispiel die in der Produktion verwendeten Hilfs- und Betriebsstoffe mit in die Oköbilanz der Endprodukte einfließen. Weiter ist sowohl der Energieaufwand als auch die im Produktionsprozess entstehenden Emissionen für die Ermittlung der Produktnachhaltigkeit relevant. In diesem Zusammenhang müssen ebenfalls Daten über die im Produktionsprozess anfallende Abfallstoffe erhoben werden. Dies bedeutet, dass eine vollständige Betrachtung des Energie- und Stoffstrommanagements erfolgen muss. Für die Abbildung bzw. Analyse dieser Daten bietet das SAP S/4HANA in seiner Standardausführung keinerlei Optionen.
Es fließen verschiedene *Rohstoffe* in die zu produzierenden Halbfabrikate und Fertigerzeugnisse ein. Diese Rohstoffe besitzen verschiedene Eigenschaften, die es im Hinblick auf die Produktnachhaltigkeit zu analysieren gilt. So müssen Rohstoffe, die ein Gefahrenpotential aufweisen (z.B. in Form ihrer chemischen Eigenschaften) entsprechend gekennzeichnet werden. In diesen Zusammenhang sind auch andere Schadstoffbelastungen der Rohstoffe zu betrachten, auch wenn diese keine direkten Gefahrgüter darstellen.
Im Materialstamm können einige Umweltdaten gepflegt werden, welche ein Gefahrenpotential im Produkt kennzeichnen. Hierzu wird die Sicht „Grunddaten2“ unter der SAP-Transaktion MM02 gewählt. Im Dialogfenster besteht die Möglichkeit ein Profil zur Gefahrgutkennzeichung zu hinterlegen. Dieses Profil kann dann zur Steuerung gefahrgutrelevanter Anwendungen genutzt werden. Das Gefahrgutprofil setzt sich aus einer Kombination aus verschieden Parameter zusammen. Diese Parameter beschreiben zum Beispiel, die Notwendigkeit einer Gefahrgutprüfung oder die Austellung von Gefahrgutpapieren. Es besteht für den Anwender die Möglichkeit verschiedene Gefahrgutprofil über das Customizing anzulegen.
Des Weiteren kann in der Sicht „Grunddaten2“ das Kennzeichen „umweltrelevant“ ak-

tiv gesetzt werden. Dieser Parameter kennzeichnet ein Produkt als umweltrelevant. So kann bei der Lieferung eines umweltrelevanten Rohstoffes der Vertrieb informiert werden und entsprechende Schritte einleiten, wie zum Beispiel des Erstellen eines passenden Sicherheitsdatenblattes. Ergänzende Produktmerkmale wie die Aggregatzustände können hierbei ebenfalls hinterlegt wegen.

8.4. Recyclingquote

Wie in den vorangegangen Kapiteln ersichtlich ist, kann die Produktnachhaltigkeit durch viele verschiedene Daten definiert werden. In diesem Kapitel erfolgt eine Fokussierung auf die Recyclingquote als Nachhaltigkeitsindikator von Produkten. Die Recyclingquote beschreibt den Anteil an wiederverwendbaren Rohstoffen eines Produktes. Hierbei wird zwischen Sekundärbauteilen und Sekundärrohstoffen unterschieden. Sekundärbauteile können als separierte und wiederverwendbare Bauteile bezeichnet werden, wie beispielsweise die Tür eines PKWs. Sekundärrohstoffe sind Rohstoffe, wie beispielsweise Metall, die wiederverwertbar sind.
Zur Implementierung einer Recyclingquote im SAP S/4HANA können verschiedene Umsetzungswege eingeschlagen werden. Im Folgenden werden zwei mögliche Varianten der Umsetzung vorgestellt.

8.4.1. Variante 1

Die Variante mit dem womöglich niedrigsten Implementierungsaufwand ist die Integration der Recyclingquote als Parameter in den Materialstamm. Konkret bedeutet dies, dass ein zusätzliches Feld in der SAP-Datenbanktabelle MARA ergänzt wird. Diese Tabelle beinhaltet umfassende Parameter zu den einzelnen Materialien, welche im System gepflegt werden können. Des Weiteren müsste eine Anpassung der Oberfläche zur Materialpflege vorgenommen werden. Hierbei könnte es sinnvoll sein, den neuen Parameter in den Reiter der Umweltdaten im Dialogfenster „Grunddaten2“ zu ergänzen. Der Anwender hat hierdurch die Möglichkeit, die Recyclingquote manuell über diese Oberfläche zu pflegen. Mögliche Schwierigkeiten bei diesem Ansatz ergeben sich aus der Verfügbarkeit der Recyclinginformationen. Insbesondere wenn es sich um fremdbeschaffte Materialien handelt muss gewährleistet sein, dass der Zulieferer über derartige Informationen verfügt und diese in geeigneter Form veröffentlicht.

8.4.2. Variante 2

Eine weitere Möglichkeit ist die Berechnung der Recyclingquote über die verschiedenen Bestandteile (Rohstoffe) eines Produktes. Hierzu können die bereits in der SAP-Datenbanktabelle MARA existierenden Felder „Bestandteile“ genutzt werden. Diese Felder könnten Aufschluss über die verarbeiteten Rohstoffe eines Produktes geben. Des Weiteren beinhaltet die Tabelle ein Feld, welches den prozentualen Anteil des Bestandteils

ausgibt. Durch die Ergänzung eines zusätzlichen Feldes, welches angibt ob ein Bestandteil recyclingfähig ist, könnte in Kombination mit dem verwendeten prozentualen Anteil die Berechnung der Recyclingquote erfolgen. Zusätzlich wären weitere Analysen im Hinblick auf Gewicht oder Volumen von Recyclingmaterial möglich. Diese Erkenntnisse könnten beispielsweise für die produktionsintegrierte Rückführung der Recyclingmaterialien relevant sein.
Eine mögliche Variation dieses Ansatzes wäre die Ermittlung der Bestandteile bzw. Rohstoffe über die Stücklisten. Dies setzt voraus, dass keine fremdbeschafften Halbfabrikate verwendet werden. Falls dies doch der Fall sein sollte müssten entsprechende Informationen über die verwendeten Bestandteile weitergeben werden.

8.5. Implementierung

Um einen Anwendungsfall aus dem Bereich der Nachhaltigkeit zu implementieren, muss auf umweltrelevante Produktdaten zugegriffen werden. Potenziell relevante Daten wurden hierzu in Kapitel 8.3 auf Seite 368 näher beschrieben.
In der nachfolgenden Tabelle (siehe Tabelle 8.1) wurden diese Daten und deren Verfügbarkeit in den Datenbanktabellen inklusive der Feldbezeichner beschrieben. Als besonders relevant haben sich dabei die Tabellen MARC - Werksdaten zum Material, MARA - Allgemeine Materialdaten, MAST - Material/Stücklisten-Zuordnung sowie STPO - Stücklistenposition herausgestellt. Mit diesen Parameter über die Datenbanktabellen und den Feldbezeichnern der jeweiligen Daten können Datenbankabfragen durchgeführt werden.

Tabelle 8.1.: Produktnachhaltigkeit: Umweltrelevante Daten in SAP S/4HANA - Datenbanktabellen

Beschreibung	**Datenbanktabelle**	**Feldbezeichner**
Beschaffung		
Beschaffungsart	MARC	BESKZ
Umweltdaten		
Gefahrgutkennzeichenprofil	MARA	PROFL
Hochviskos	MARA	IHIVI
Lose Schüttung / Flüssigkeit	MARA	ILOOS
Umweltrelevant	MARA	KZUMW

Tabelle 8.1.: Produktnachhaltigkeit: Umweltrelevante Daten in SAP S/4HANA - Datenbanktabellen

Beschreibung	**Datenbanktabelle**	**Feldbezeichner**
Grunddaten		
Größe Abmessung	MARA	GROES
Werkstoff	MARA	WRKST
Bruttogewicht	MARA	BRGEW
Nettogewicht	MARA	NTGEW
Gewichtseinheit	MARA	GEWEI
Volumen	MARA	VOLUM
Volumeneinheit	MARA	VOLEH
Gefahrstoffnummer	MARA	STOFF
Länge	MARA	LAENG
Breite	MARA	BREIT
Höhe	MARA	HOEHE
Längeneinheit	MARA	MEABM
Stücklistendaten		
Zuordnung von Material und Stückliste	MAST	STLNR
Stücklistenkomponente	STPO	IDNRK
Menge der Komponente	STPO	MENGE

8.6. Fazit

Dieses Konzept hat aufgezeigt, welche ökologische Daten bereits im SAP S/4HANA System vorhanden sind. Hierbei wurde ersichtlich, dass es sich bei den abgebildeten Daten um Parameter handelt, welche oft nicht im konkreten Zusammenhang mit einer Ökologischen Betrachtung stehen und somit nur in Kombination mit weiteren Parametern eine ökologische Aussagekraft besitzen. So ist es für die konkrete Abbildung der Recylingquote eines Produktes notwendig, eine Erweiterung der SAP-Datenbanktabellen vorzunehmen. Dies bedeutet, dass der Materialstamm um weitere Parameter ergänzt

werden muss. Hierfür wurden zwei konkrete Lösungsansätze aufgezeigt.
Zusammenfassend kann festgehalten werden, dass das SAP S/4HANA System (ohne Erweiterung des Materialstammes) wenig Optionen für die ökologische Betrachtung von Produkten bietet.

8.7. Ausblick

Um eine umfassende ökologische Betrachtung des SAP S/4HANA Systems vornehmen zu können, wäre eine ergänzende Evaluierung der SAP Business-Add-Ins vorzunehmen. In diesem Zusammenhang sollte das Business-Add-In mit der Bezeichnung „Environment Health and Safety“ genauer analysiert werden, da dieses Unterstützung bei der Abbildung von ökologischen Daten verspricht.
Des Weiteren wurden Datenbanktabellen im System identifiziert, die wichtige Materialparameter enthalten. Diese Parameter könnten für die Implementierung von weiteren Anwendungen zur Produktnachhaltigkeit relevant sein. Eine denkbare Anwendung wäre eine Kachel in der SAP Fiori-Oberfläche, welche bei Eingabe der Produktnummer die Recyclingquote ausgibt oder Aufschluss über die Beschaffungsart der einzelnen Rohstoffe gibt.

9. Fazit

In diesem Kapitel wird aus dem Ablauf des gesamten Projekts ein Fazit gezogen und die gewonnen Erfahrungen reflektiert. Hierzu wird die Organisation und der allgemeine Ablauf sowie die Zusammenarbeit und Kommunikation im Rahmen des Projekts beschrieben und kritisch betrachtet.

Organisation und allgemeiner Ablauf

Die Hauptphasen der Projektgruppe teilten sich grob in die Phasen Seminararbeit, Konzeption, Entwicklung und Dokumentation auf. Die zeitlich intensivste und komplexeste Phase war dabei die Entwicklung. Dieser Phase hätte mehr Zeit bereitgestellt werden können, um eine hohe Qualität der Ergebnisse zu gewährleisten. Die trotzdem erreichte hohe Qualität der einzelnen Szenarien konnte aufgrund des knappen Zeitplans nur durch das Engagement der Teammitglieder erreicht werden. Mehrarbeit und ungenutzte Urlaubstage waren deshalb keine Seltenheit.

Besonders die Phase der Seminararbeit hätte verkürzt werden können. Ideal wäre es gewesen, anstatt der Seminararbeiten nur Präsentationen über die verteilten Themen zu halten. Ebenfalls entstand der Eindruck, dass der Inhalt der Arbeiten für den weiteren Projektverlauf wenig relevant war, da es hierzu nie ein Feedback gab.

Die Phase der Konzeption vermittelte den Projektmitgliedern die Fähigkeit, Anforderungen aus einer Aufgabenstellung zu erheben und mit einem Praxispartner zu validieren. Das Vermitteln dieser Fähigkeiten über Vorlesungen oder Seminare wäre in dieser Form nicht möglich gewesen. Den Projektmitgliedern ist es also gelungen, sich durch die Arbeit mit einem Praxispartner, welcher wie ein Kunde aufgetreten ist, persönlich weiterzuentwickeln und Fähigkeiten zu erlernen, welche besonders in einem späteren beruflichen Umfeld extrem wichtig sein können.

In der Entwicklungsphase waren besonders Gruppenmitglieder gefordert, welche wenig Erfahrung im Bereich der Softwareentwicklung bzw. Programmierung hatten. Die initiale Lernschwelle war allerdings für alle Mitglieder hoch, sobald diese jedoch überwunden war, konnten schnell fachliche Kompetenzen aufgebaut werden. Die Entwicklung lief deshalb mit fortschreitender Projektlaufzeit immer schneller.

Besonders das Szenario „Echtzeit Stromanalyse“ wurde in kürzester Zeit entwickelt, war aber trotzdem qualitativ hochwertig. Zurückblickend konnten die Mitglieder ihre fachlichen und technischen Kompetenzen also entscheidend verbessern.

Zusammenarbeit und Kommunikation

Das frühe Aufteilen der Projektgruppe in die anfänglichen drei Teilgruppen, ermöglichte das parallele Arbeiten an den ersten drei Szenarien. Zum Austausch der gesamten Projektgruppe wurden neben dem wöchentlichen Treffen mit den Betreuern und dem dreiwöchigen Treffen in Bremen mit abat auch ein wöchentlicher interner Termin abgehalten, der im späteren Projektverlauf immer wichtiger wurde. Die Projektmitglieder hatten hier die Möglichkeit, Probleme zu besprechen, gemeinsam zu arbeiten und sich zu koordinieren, was i.d.R. sehr gut geklappt hat. Lediglich der kleine Gruppenraum bot für 10 Personen kaum Platz, was die gemeinsame Arbeit teilweise erschwerte. Regelmäßige Gruppenaktivitäten wie Paintball oder Kartfahren sorgten zusätzlich für einen starken Gruppenzusammenhalt.
Der Praxispartner abat nahm im Projekt die Rolle eines Kunden ein, welcher verschiedene Szenarien umgesetzt haben wollte. Die Herangehensweise an die Szenarien wurde dabei nie vorgegeben, alles wurde im Dialog erarbeitet, um eine möglichst realitätsnahes Projekt zu simulieren. Neben der Rolle des Kunden stellte abat ebenfalls die HANA-Infrastruktur bereit, auf der die Szenarien realisiert wurden. Bei fachlichen oder technischen Problemen konnte ebenfalls der Kontakt zum Praxispartner gesucht werden, woraufhin i.d.R. ein Treffen oder eine Telefonkonferenz stattfand, um das Problem zu lösen.
Zusammenfassend lief sowohl die interne als auch die externe Zusammenarbeit größtenteils gut, es wäre jedoch in einigen Fällen förderlich gewesen, wenn Mitarbeiter von abat häufiger auch an Treffen in Oldenburg teilgenommen hätten. Die Abstimmung wäre so sicherlich in manchen Bereichen einfacher gefallen und es wäre eine höhere Transparenz der Kommunikation entstanden. Dies hätte ebenfalls den Betreuern der Universität geholfen, welche auch nicht bei jedem Treffen in Bremen anwesend sein konnten.

Ziel der Projektgruppe

Zusammenfassend war das übergeordnete Ziel der Projektgruppe die neue In-Memory Plattform SAP HANA eingehend zu testen und Eigenentwicklungen auf dieser zu realisieren. Die umgesetzten Szenarien zeigen dabei das Potential der neuen Technologie und liefern einen Einblick in mögliche Projekte und entstehenden Kundennutzen.

Neben der Analyse von SAP HANA war ein weiteres Ziel allerdings auch das selbst gesteuerte Management der Projektgruppe und das Aufbauen von Kompetenzen, was fast uneingeschränkt gelungen ist. Es ist davon auszugehen, dass alle Teilnehmer der Projektgruppe sowie alle weiteren Stakeholder von dem Projekt profitieren konnten, sodass weitere Projekte dieser Art in Zukunft absolut sinnvoll sind.

Literaturverzeichnis

[1] McKinsey: *The business of sustainability: McKinsey Global Survey results*. Oktober 2011. – http://www.mckinsey.com/business-functions/sustainability-and-resource-productivity/our-insights/the-business-of-sustainability-mckinsey-global-survey-results, abgerufen am 18.02.2017

[2] Statistisches Bundesamt: *Unternehmen und Arbeitsstaetten: Nutzung von IKT in Unternehmen*. Dezember 2015. – https://www.destatis.de/DE/Publikationen/Thematisch/UnternehmenHandwerk/Unternehmen/InformationstechnologieUnternehmen5529102157004.pdf?__blob=publicationFile, abgerufen am 18.02.2017

Glossar

AVM FRITZ!DECT 200 Die AVM FRITZ!DECT 200 (intelligente Steckdose) ist ein Produkt des Herstellers AVM. Sie wurde entwickelt, um bei Endnutzern den Stromverbrauch von Endgeräten zu messen. Sie funktioniert nur in Verbindung mit einer AVM Fritzbox ab Firmwareversion 05.50. Der maximale Stromverbrauch für Endgeräte liegt bei 2300 Watt. Die Messgenauigkeit liegt bei +-100mW. 318, 319

Freedcamp Freedcamp ist ein webbasiertes Projektmanagement-Tool. Es hat folgende Funktionen: Tasks, Discussions, Milestones, Time, Files, Wiki und Calendar. 8, 13

Fritzbox Die Fritzbox ist ein WLAN Router vom Hersteller AVM mit integriertem Modem. Er kann per WLAN mit anderen Geräten von AVM wie z.B. einer intelligenten Steckdose (AVM FRITZ!DECT 200) und einem Computer eine Verbindung herstellen. Sie bietet eine grafische Oberfläche, in der mehrere AVM FITZ!DECT 200 Geräte verwaltet werden können. Durch eine VPN Verbindung kann auf die Fritzbox auch außerhalb des abat AG Gebäudes zugegriffen werden. Die installierte Firmware lief auf der Version 06.60. 328, 336–339, 363, 365

GIT GIT ist eine frei zugängliche Software zur Versionsverwaltung von Dateien. Sie lässt sich zur gemeinsamen Erstellung von Texten oder Programmen nutzen. Es gibt unterschiedliche Distributionen zur Verwendung von GIT. Beispiele hierfür sind Sourcetree, GITKraken, TortoiseGit und andere. 7

HANA-Studio Eclipse ist ein quelloffenes Entwicklungswerkzeug zur Entwicklung von Softwareanwendungen. Für Eclipse gibt es viele Erweiterungen. Unter anderem zählt das HANA Studio dazu, welches zum entwickeln der Apps genutzt wurde. 121, 123, 203, 209, 233, 237

LaTeX Latex ist ein Softwarepaket, das die Benutzung des Textsatzsystems TeX mit Hilfe von Makros ermöglicht und sich für die Erstellung größerer Schriftstücke wie Bücher, Abschlussarbeiten oder wissenschaftliche Artikel gut eignet.. 7

Raspberry Pi Der Raspberry Pi ist ein Einplatinencomputer und wurde von der britischen Raspberry Pi Foundation entwickelt. Der Computer besteht aus einem Ein-Chip-System von Broadcom und einem ARM-Mikroprozessor. Die Grundfläche der Platine hat ca. die Größe einer Kreditkarte und das Volumen hat etwa die Größe einer Streichholzschachtel. Geräte und Sensoren können an den Raspberry Pi angeschlossen werden. 14, 91, 94–96, 98, 102, 113, 114, 124, 125, 134, 139, 140, 146, 150, 153, 155, 158, 167–169, 171, 176, 178, 183–185, 187, 188, III, VI, XI, XII

Redmine Redmine ist ein freies, webbasiertes Open-Source-Software-Tool, welches ermöglicht Projekte zu verwalten. Nutzer können mehrere Projekte und zugehörige Teilprojekte verwalten. Die wichtigsten Funktionen sind Diskussionsforen, Wikis, Ticketverwaltung und Dokumentenablage. 13

SAP ERP 6.0 SAP ERP ist ein Softwareprodukt und steht für Enterprise-Resource-Planning. Es ermöglicht Unternehmen die Betrachtung aller prozess- und geschäftsrelevanten Bereiche. Die aktuelle Version ist SAP ERP [Central Component (ECC)] 6.0 Enhancement Package 8. 196, 199, 300, 310

SAP GUI SAP GUI ist ein Akronym von SAP Graphical User Interface. Die grafische Benutzeroberfläche ermöglicht SAP-Usern den Zugriff. Auf der anderen Seite steht das auch für dasjenige Programm, das diese Oberfläche bereitstellt und betreibt. 83, 203–205, 209, 210, 232, XII

SAP HANA SAP HANA ist eine Entwicklungsplattform für Softwareanwendungen, die im Kern aus einer In-Memory-Datenbank besteht. 1, 2, 9, 10, 91, 98, 100, 103, 191, 236, 311, 314, 316, 320–325, 337, 365, 376, III

SAP S/4HANA SAP S/4HANA ist ein Softwareprodukt für ein Echtzeit-Enterprise Resource Management. Es basiert auf der In-Memory-Plattform SAP HANA und kann in der Cloud oder on Premise eingesetzt werden. 1, 2, 9–11, 13, 61, 85, 89, 186, 314, 316, 367–373, III, VII, XX

SAP Web IDE SAP Web IDE ist ein erweiterbares und webbasiertes Entwicklungswerkzeug. Es kann zum entwickeln von Apps genutzt werden. Dabei erhält der User beispielsweise Hilfe durch Assistenten oder Vorlagen. 237

Wordpress Wordpress ist ist eine Online Applikation mit der sich Webseiten und andere Web-Präsenzen ohne wesentliche Programmierkenntnisse erstellen lassen. Hierzu werden vorgefertigte Vorlagen kostenlos oder auch kostenpflichtig zur Verfügung gestellt. 7

A. Anhang

A.1. Identifizierung und Abbildung von Nachhaltigkeitsdaten in SAP S/4HANA

Identifizierung und Abbildung von Produktnachhaltigkeitsdaten in SAP S/4HANA

Nils Groenhoff, Rene Kessler, Stefan Wunderlich und Jorge Marx Gomez

Zusammenfassung Nachhaltigkeit nimmt in der heutigen globalen Wirtschaft eine immer größer werdende Rolle ein. Auch produzierende Unternehmen sind von diesem Wandel betroffen und müssen sich daher vermehrt mit Themen der Nachhaltigkeit befassen. Aufgrund ihrer heutzutage sehr weiten Verbreitung, müssen auch moderne betriebliche Informations- und Anwendungssysteme diesen Entwicklungen Rechnung tragen. Hierzu wurde ein Vorgehensmodell erarbeitet, welches einen Lösungsansatz zur Identifizierung von Produktnachhaltigkeitsdaten in SAP S/4HANA darstellt.

1 Motivation und Zielbestimmung

Im Zuge der stetig wachsenden Bedeutung der Nachhaltigkeit in der Gesellschaft, hat der Begriff Nachhaltigkeit auch in die Wirtschaft Einzug gefunden. Nachhaltigkeit wird in der Literatur unterschiedlich definiert, jedoch beinhaltet der Begriff immer eine soziale, eine ökologische und eine ökonomische Dimension [1]. Jeder dieser Dimensionen sollten Unternehmen im täglichen Geschäft beachten. In einer Befragung der Unternehmensberatungsfirma *McKinsey* aus dem Jahr 2011 [2] wurde gezeigt, dass Unternehmen in ihren verschiedensten Organisationseinheiten und Geschäftsprozessen grundsätzlich Ziele der Nachhaltigkeit verfolgen. Auffällig

Nils Groenhoff
Carl von Ossietzky Universität, 26129 Oldenburg, Germany, E-mail: nils.groenhoff@uol.de

Rene Kessler
Carl von Ossietzky Universität, 26129 Oldenburg, Germany, E-mail: rene.kessler@uol.de

Stefan Wunderlich
Carl von Ossietzky Universität, 26129 Oldenburg, Germany, E-mail: stefan.wunderlich@uol.de

Jorge Marx Gomez
Carl von Ossietzky Universität, 26129 Oldenburg, Germany, E-mail: jorge.marx.gomez@uol.de

ist, dass über 60% der Befragten Unternehmen angaben, dass in den täglichen Geschäftstätigkeiten versucht wird, Energieverbrauch oder auch Abfallaufkommen zu reduzieren. Dies kann insbesondere für das produzierende oder verarbeitende Gewerbe von großer Relevanz sein. Eine Statistik des statistischen Bundesamtes hat zudem ergeben, dass in Deutschland 93% der Unternehmen mit über 250 Mitarbeitern ERP-Software einsetzen [3]. Im verarbeitenden Gewerbe ergibt sich sogar ein Anteil von 98%. Durch Verbindung dieser beiden Studienergebnisse, lässt sich schließen, dass umfassende Mehrwerte und Synergien entstehen würden, wenn ERP-Systeme die Nachhaltigkeitsaktivitäten eines Unternehmens besser abbilden und unterstützen würden.

Die beschriebene Konzeption setzt sich mit der Identifizierung und Abbildung von Produktnachhaltigkeitsdaten im betrieblichen Informations- und Anwendungssystemsystem SAP S/4HANA auseinander. Bei der Abbildung dieser Daten sind verschiedene Nutzungsszenarien, wie beispielsweise die Aufbereitung dieser Daten in einem Dashboard oder datengetriebe Echtzeitfrühwarnsysteme, denkbar. Hierzu wird ein generisches Vorgehensmodell erarbeitet, welches im Anschluss auf einen konkreten Anwendungsfall angewandt wird.

Konkret wird hierbei die Produktnachhaltigkeit fokussiert, welche durch die Recyclingquote eines Produktes bestimmt werden kann. Dabei wird aufgezeigt, in welchem Umfang SAP S/4HANA diesen Anwendungsfall unterstützen kann. So wird evaluiert werden, ob das System bereits nutzbare Informationen zur Abbildung der Recyclingquote bietet und auf welche Art und Weise fehlende Inhalte bzw. Daten ergänzt werden können.

2 Methodisches Vorgehen

Zur Identifizierung von Produktnachhaltigkeitsdaten muss zunächst eine umfassende Datenerhebung durchgeführt werden. Zum Einen können intuitiv-kreative Methoden wie Brainstorming oder Mind Maps angewandt werden [4], zum Anderen sind auch konventionelle Methoden zur Datenerhebung wie die Literaturrecherche denkbar. Insbesondere eine Kombination verschiedenartiger Methoden erscheint in diesem Ansatz als sinnvoll, da so eine umfassende Betrachtung sichergestellt werden kann.

Im Anschluss daran können die erhobenen Daten kategorisiert werden. Ist dies erfolgt, können die kategorisierten Daten in Hinblick auf ihre Bedeutung im Kontext der Nachhaltigkeit bewertet werden. Festzustellen ist dabei, ob aus der Kombination verschiedener Daten Mehrwerte geschaffen werden können.

Um zu evaluieren welche der erhobenen Daten tatsächlich in SAP S/4HANA abgebildet werden, sollte eine *Fit/Gap-Analyse* durchgeführt werden. Die daraus resultierenden *Fits* zeigen, welche Daten bereits im System abgebildet wurden und genutzt werden können, während die *Gaps* Aufschluss über Ansatzpunkte für Erweiterungen des Systems geben.

Im letzten Schritt können aus diesen Ergebnissen Handlungsempfehlungen abgeleitet werden, mit denen die tatsächliche Umsetzung einer noch nicht im SAP S/4HANA vorhandenen Funktion im Bereich der Nachhaltigkeit geplant werden kann. Das methodische Vorgehen zur Identifizierung von Produktnachhaltigkeitsdaten wurde in der Abbildung 2 ergänzend graphisch dargestellt.

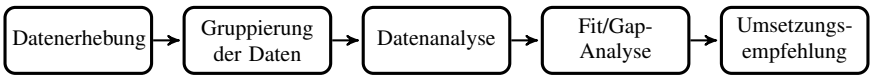


Abb. 1 Prozess: Methodisches Vorgehen

3 Datenerhebung, Gruppierung und Interpretation

Im ersten Schritt werden Daten erhoben mit denen die Nachhaltigkeit eines Produktes bestimmt werden kann. Im nächsten Schritt wurden die Daten kategorisiert (siehe Tabelle 1). Anschließend erfolgte eine Interpretation der Daten. Zusammenfassend bedeutet dies, dass die ersten drei Schritte des zuvor erarbeiteten Vorgehensmodells in diesem Kapitel kurz beschrieben werden.

Grundsätzlich muss bei der Beschreibung von Daten der ökologischen Nachhaltigkeitsdimension zwischen Umweltdaten und umweltrelevanten Daten unterschieden werden [5]. In SAP S/4HANA werden beide Begriffe häufig fälschlicherweise synonym verwendet. Umweltdaten haben immer einen direkten Bezug zu Eigenschaften der Medien Luft, Wasser oder Erde. Mit den Daten kann somit beispielsweise der Zustand dieser Medien beurteilt werden (z.B. CO2-Gehalt in der Luft). Umweltrelevante Daten haben dagegen nur einen mittelbaren Bezug zu Umweltwirkungen. Somit haben diese Daten bzw. die damit verbundenen Folgen einen Einfluss auf die Umwelt (z.B. Emissionsausstoß einer Maschine).

Die *Physikalischen Eigenschaften* umfassen das Volumen und Gewicht sowie die Abmessungen eines Produktes. Diese Daten können relevant für weitere Kennzahlen zur Ermittlung der Produktnachhaltigkeit sein [6]. So kann zum Beispiel das Gewicht eines Produktes maßgeblichen Einfluss auf die ausgestoßenen Emissionen beim Transport nehmen.

Bei der *Beschaffungsart* muss eine Unterteilung in Fremd- und Eigenfertigung vorgenommen werden. Dies ist von besonderer Relevanz, da bei fremdbeschafften Materialien nachvollzogen werden muss, welche einzelnen Ausgangsmaterialien bzw. Rohstoffe verwendet worden sind, um eine konkrete Aussage über die Produktnachhaltigkeit treffen zu können [7].

Beim *Transport* von Produkten entstehen heterogene Emissionen, die bei der Betrachtung der Produktnachhaltigkeit berücksichtigt werden müssen [8]. Hierbei ist sowohl der Transport von Rohstoffen und Halbfabrikaten zum Produktionswerk gemeint als auch die Auslieferung des Endproduktes zum Kunden. Insbesondere muss hierbei der Modal Split berücksichtigt werden, da die jeweiligen Transport-

modi unterschiedliche Umweltwirkungen aufweisen (bspw. Transport zur See oder Luftfracht).

Um die Produktnachhaltigkeit zu bestimmen ist eine Berücksichtigung des *Fertigungsprozesses* notwendig. So müssen zum Beispiel die in der Produktion verwendeten Hilfs- und Betriebsstoffe mit in die Oköbilanz der Endprodukte einfließen. Weiter ist sowohl der Energieaufwand als auch die im Produktionsprozess entstehenden Emissionen für die Ermittlung der Produktnachhaltigkeit relevant. In diesem Zusammenhang müssen ebenfalls Daten über die im Produktionsprozess anfallende Abfallstoffe erhoben werden. Dies bedeutet, dass eine vollständige Betrachtung des Energie- und Stoffstrommanagements erfolgen muss [9].

Daten zur Bewertung der Produktnachhaltigkeit	
Physikalische Eigenschaften	Gewicht
	Abmessungen
	Volumen
Transport	Emissionen
Fertigung	Emissionen
	Energie
	Abfallstoffe
	Hilfsstoffe
Beschaffungsart	Fremdfertigung
	Eigenfertigung
Rohstoffe	Schadstoffbelastung
	Recyclingquote
	Gefahrgut

Tabelle 1 Daten im Umfeld der Produktnachhaltigkeit

Es fließen verschiedene *Rohstoffe* in die zu produzierenden Halbfabrikate und Fertigerzeugnisse ein. Diese Rohstoffe besitzen verschiedene Eigenschaften, die es im Hinblick auf die Produktnachhaltigkeit zu analysieren gilt. So müssen Rohstoffe, die ein Gefahrenpotential aufweisen (z.B. in Form ihrer chemischen Eigenschaften) entsprechend gekennzeichnet werden. In diesen Zusammenhang sind auch andere Schadstoffbelastungen der Rohstoffe zu betrachten, auch wenn diese keine direkten Gefahrgüter darstellen.

4 Fit/Gap-Analyse

Um den Anwendungsfall der Recyclingquote zu implementieren muss auf umweltrelevante Produktdaten zugegriffen werden. Potenziell relevante Daten wurden hierzu im Kapitel 5 näher beschrieben. Nun gilt es, diese Daten mit dem bereits im System vorhandenen Parametern zu vergleichen und mögliche Übereinstimmungen sowie Erweiterungspotentiale zu identifizieren.

Beschreibung	**Datenbanktabelle**	**Feldbezeichner**
Beschaffung		
Beschaffungsart	MARC	BESKZ
Umweltdaten		
Gefahrgutkennzeichenprofil	MARA	PROFL
Hochviskos	MARA	IHIVI
Lose Schüttung / Flüssigkeit	MARA	ILOOS
Umweltrelevant	MARA	KZUMW
Grunddaten		
Größe Abmessung	MARA	GROES
Werkstoff	MARA	WRKST
Bruttogewicht	MARA	BRGEW
Nettogewicht	MARA	NTGEW
Gewichtseinheit	MARA	GEWEI
Volumen	MARA	VOLUM
Volumeneinheit	MARA	VOLEH
Gefahrstoffnummer	MARA	STOFF
Länge	MARA	LAENG
Breite	MARA	BREIT
Höhe	MARA	HOEHE
Längeneinheit	MARA	MEABM
Stücklistendaten		
Zuordnung von Material und Stückliste	MAST	STLNR
Stücklistenkomponente	STPO	IDNRK
Menge der Komponente	STPO	MENGE

Tabelle 2 Umweltrelevante Daten in SAP S/4HANA - Datenbanktabellen

In der vorangestellt gezeigten Tabelle (siehe Tabelle 2) werden bereits vorhandene Daten und deren Verfügbarkeit in den SAP Datenbanktabellen inklusive der Feldbezeichner beschrieben. Als besonders relevant haben sich dabei die Tabellen MARC - Werksdaten zum Material, MARA - Allgemeine Materialdaten, MAST - Material/Stücklisten-Zuordnung sowie STPO - Stücklistenposition herausgestellt, da in diesen Tabellen die beschreibenden Parameter eines Materials bzw. Produktes gespeichert werden.

Diese Parameter können unter anderem durch Kombination mit anderen Parametern genutzt werden, um Anwendungsfälle im Bereich der Nachhaltigkeit zu implementieren. In der Tabelle 2 sind verschiedene Felder dieser SAP-Datenbanktabellen aufgelistet. Hierbei sind unter *Grunddaten* Parameter enthalten, die die äußeren Eigenschaften eines Produktes beschreiben. Die *Stücklistendaten* und Daten über die *Beschaffung* können genutzt werden, um Aussagen über die Zusammensetzung eines Produktes treffen zu können. Den stärksten direkten Bezug zur Nachhaltigkeit besitzen die *Umweltdaten*, welche direkt im Materialstamm erfasst werden können.

5 Umsetzungsempfehlung

Wie in den vorangegangen Kapiteln ersichtlich ist, kann die Produktnachhaltigkeit durch viele verschiedene Daten definiert werden. In diesem Kapitel erfolgt eine Fokussierung auf die Recyclingquote als Nachhaltigkeitsindikator von Produkten. Die Recyclingquote beschreibt den Anteil an wiederverwendbaren Rohstoffen eines Produktes. Hierbei wird zwischen Sekundärbauteilen und Sekundärrohstoffen unterschieden. Sekundärbauteile können als separierte und wiederverwendbare Bauteile bezeichnet werden, wie zum Beispiel die Tür eines PKWs. Sekundärrohstoffe sind Rohstoffe, wie beispielsweise Metall, die wiederverwertbar sind.

Auf Basis der Fit/Gap-Analyse wurde ersichtlich, dass für die Abbildung der Recyclingquote einige ergänzende Implementierungen vorgenommen werden müssen.

Zur Implementierung einer Recyclingquote im SAP S/4HANA können verschiedene Umsetzungswege eingeschlagen werden. Im Folgenden werden zwei mögliche Varianten der Umsetzung beschrieben.

Variante 1: Integration in Materialstamm

Die Variante mit dem womöglich niedrigsten Implementierungsaufwand ist die Integration der Recyclingquote als Parameter in den Materialstamm. Konkret bedeutet dies, dass ein zusätzliches Feld in der SAP-Datenbanktabelle MARA ergänzt wird. Diese Tabelle beinhaltet umfassende Parameter zu den einzelnen Materialien, welche im System gepflegt werden können. Des Weiteren müsste eine Anpassung der Oberfläche zur Materialpflege vorgenommen werden. Hierbei könnte es sinnvoll sein, den neuen Parameter in den Reiter der Umweltdaten im Dialogfenster „Grund-

daten 2“ zu ergänzen. Der Anwender hat hierdurch die Möglichkeit, die Recyclingquote manuell über diese Oberfläche zu pflegen. Mögliche Schwierigkeiten bei diesem Ansatz ergeben sich aus der Verfügbarkeit der Recyclinginformationen. Insbesondere wenn es sich um fremdbeschaffte Materialien handelt muss gewährleistet sein, dass der Zulieferer über derartige Informationen verfügt und diese in geeigneter Form veröffentlicht.

Variante 2: Berechnung über verschiedene Bestandteile

Eine weitere Möglichkeit ist die Berechnung der Recyclingquote über die verschiedenen Bestandteile (Rohstoffe) eines Produktes. Hierzu können die bereits in der SAP-Datenbanktabelle MARA existierenden Felder „Bestandteile“ genutzt werden. Diese Felder könnten Aufschluss über die verarbeiteten Rohstoffe eines Produktes geben. Des Weiteren beinhaltet die Tabelle ein Feld, welches den prozentualen Anteil des Bestandteils ausgibt. Durch die Ergänzung eines zusätzlichen Feldes, welches angibt ob ein Bestandteil recyclingfähig ist, könnte in Kombination mit dem verwendeten prozentualen Anteil die Berechnung der Recyclingquote erfolgen. Zusätzlich wären weitere Analysen im Hinblick auf Gewicht oder Volumen von Recyclingmaterial möglich. Diese Erkenntnisse könnten beispielsweise für die produktionsintegrierte Rückführung der Recyclingmaterialien relevant sein.

Eine mögliche Variation dieses Ansatzes wäre die Ermittlung der Bestandteile bzw. Rohstoffe über die Stücklisten. Dies setzt voraus, dass keine fremdbeschafften Halbfabrikate verwendet werden. Falls dies doch der Fall sein sollte müssten entsprechende Informationen über die verwendeten Bestandteile weitergeben werden.

6 Fazit und Ausblick

Die Anwendung des in Kapitel 2 erarbeiteten Vorgehensmodell hat gezeigt, welche ökologische Daten bereits im SAP S/4HANA System vorhanden sind. So ist es für die konkrete Abbildung der Recylingquote eines Produktes notwendig, eine Erweiterung der SAP-Datenbanktabellen vorzunehmen. Dies bedeutet, dass der Materialstamm um weitere Parameter ergänzt werden muss. Hierfür wurden zwei konkrete Lösungsansätze aufgezeigt. Des Weiteren wurden Datenbanktabellen im System identifiziert, die wichtige Materialparameter enthalten, welche bei der Implementierung von weiteren Anwendungen zur Produktnachhaltigkeit relevant sind.

Es kann festgehalten werden, dass das SAP S/4HANA System (ohne Erweiterung des Materialstammes) wenig Optionen für die ökologische Betrachtung von Produkten bietet.

Insgesamt hat sich das Vorgehensmodell zur Identifizierung und Abbildung von Nachhaltigkeitsdaten in SAP S/4HANA als anwendbar erwiesen. Mithilfe dieses Vorgehens konnte gezielt festgestellt werden, an welchen Stellen Anpassungen in

SAP S/4HANA nötig sind, um den hier gewählten Anwendungsfall umsetzen zu können.

Um eine umfassende Beurteilung des SAP S/4HANA Systems vornehmen zu können, wäre eine ergänzende Evaluierung der SAP Business-Add-Ins als sinnvoll anzusehen. Hierbei würde entsprechend des Anwendungsfalles geprüft werden, ob das Abbilden einer Recyclingquote auf Basis von Business-Add-Ins möglich wäre.

Ein weiterer Forschungsbedarf besteht in der Fit/Gap-Analyse des Vorgehensmodell. Hier wäre eine weiterführende Untersuchung angebracht, welche Aufschluss über ein effizientes Vorgehen zur Durchführung der Fit/Gap-Analyse im System gibt. Zusammenfassend lassen sich daher die Ergebnisse dieser Konzeption als Basis für weitere Forschungsarbeiten betrachten.

Literaturverzeichnis

1. World Commission on Environment and Development. (1987). Unsere gemeinsame Zukunft: der Brundtland-Bericht. V. Hauff (Ed.). Eggenkamp.
2. McKinsey: The business of sustainability: McKinsey Global Survey results (2011). http://www.mckinsey.com/business-functions/sustainability-and-resource-productivity/our-insights/the-business-of-sustainability-mckinsey-global-survey-results. Abgerufen am 18.02.17
3. Statistisches Bundesamt: Unternehmen und Arbeitsstätten: Nutzung von IKT in Unternehmen (2015). https://www.destatis.de/DE/Publikationen/Thematisch/UnternehmenHandwerk/Unternehmen/InformationstechnologieUnternehmen5529102157004.pdf Abgerufen am 18.02.17
4. Brunner, A. (2009) Kreativer Denken: Konzepte und Methoden von A-Z. Oldenbourg.
5. Jaeschke A., et al. (2013) Informatik für den Umweltschutz: 7. Symposium 1993, Ulm, Springer-Verlag
6. Bretzke, W. R. (2014). Nachhaltige Logistik: Zukunftsfähige Netzwerk-und Prozessmodelle. Springer-Verlag.
7. Stahlmann, V. (2013). Umweltorientierte Materialwirtschaft: das Optimierungskonzept für Ressourcen, Recycling, Rendite. Springer-Verlag.
8. Kraus, S. (1997). Distributionslogistik im Spannungsfeld zwischen Ökologie und Ökonomie. Gesellschaft für Verkehrsbetriebswirtschaft und Logistik.
9. Schmidt M., & Schorb, A. (2013). Stoffstromanalysen: in Ökobilanzen und Öko-Audits. Springer-Verlag.

Abschließende Erklärung

Wir versichern hiermit, dass die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt wurde und, dass alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anliegenden Ausführungen dieser Arbeit besonders gekennzeichnet und Quellen zitiert sind.

gez.

Berghaus, *Chermette*, *Duman*, *Groenhoff*, *Heitmann*, *Hillmer*, *Kessler*, *Otto*, *Tomović* & *Yang*

29. März 2017