CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Projektdokumentation

Datenstrombasierte Recommender-Systeme

vorgelegt von

Patrick Bruns, Christian Eilts, Martin Kuhl,
Max Leonhardt, Christoph Schröer, Achim Völz,
Lukas Weinel, Christian Wigger, Christof Wolke, Marius Wybrands

Gutachter:

Prof. Dr. Dr. h.c. H.-J. Appelrath
Dr. Marco Grawunder
Dipl.-Inform. Cornelius Ludmann

Oldenburg, 8. April 2016

Inhalt

1 Einleitung

1.1 Zielsetzung des Dokuments

Für das von der Abteilung *Informationssysteme* der *Universität Oldenburg* entwickelte Datenstrom-Management-System (DSMS) *Odysseus*[1] [AGG+12] soll von der Projektgruppe ein Anwendungsfall für die vom System bereitgestellten Recommender-Funktionalitäten hergeleitet und implementiert werden. Ziel ist es, ein Szenario zu entwerfen, mit dem potenziellen Kunden oder Nutzern von Odysseus die Recommender-Funktionen anhand eines Anwendungsbeispiels anschaulich gemacht werden können. Wie dieser Anwendungsfall konkret ausgestaltet wird und welche Funktionalitäten er umfassen soll, bleibt der Projektgruppe frei gestellt. Neben dieser übergeordneten Aufgabenstellung setzt sich die Projektgruppe als Ziele, die Qualität der vom Recommender-System generierten Empfehlungen zu verbessern und schnelle Antwortzeiten zu realisieren. Dies impliziert zum einen die Verringerung der Abweichung zwischen den vorhergesagten Bewertungen des Systems und den tatsächlichen Bewertungen der Nutzer. Zum anderen soll bei der Erstellung der Systemarchitektur und der Implementierung des Anwendungsfalls darauf geachtet werden, die Zeit zwischen Empfehlungsanfrage und -antwort minimal zu halten. Erst im weiteren Projektverlauf wird erarbeitet, wie die Umsetzung dieser Ziele konkret aussieht. Die übergeordnete Aufgabenstellung setzt die Anwendung bestimmter Funktionalitäten von Odysseus in den Fokus. Aus diesem Grund werden von der Projektgruppe o.g. Qualitätsmetriken, die direkt mit dem Recommender-System in Verbindung gebracht werden können, in den Vordergrund gerückt. Weitere, potenziell mögliche Kriterien, wie beispielsweise Usability-Kriterien, sind explizit nicht die primären Ziele. Diese würden sich in erster Linie auf das Produkt als solches beziehen, unabhängig vom erläuterten Rahmen.

1.2 Vision

1.2.1 Motivation

Die Projektgruppe *Datenstrombasierte Recommender-Systeme*, bestehend aus zehn Masterstudenten der Wirtschaftsinformatik, bearbeitet über zwei Semester ein Software-Projekt, das sich im Bereich *Recommender Systems (RecSys)* (dt.: *Empfehlungs-Systeme/Software*) und Datenströmen bewegt. RecSys sind einer breiteren Masse z. B. durch Produktempfehlungen des Internet-Versandhauses *Amazon*, Musikempfehlungen des Musikstreamingdienstes *Spotify* oder zahlreichen weiteren Unternehmen bereits bekannt. Die Vorhersagen eines RecSys für das Interesse eines Nutzers zielen darauf ab, aus einer großen Datenmenge gezielt diejenigen Objekte auszuwählen, die den Nutzer des Systems zum jeweiligen Zeitpunkt am meisten interessieren könnten. Für die Ermittlung des Interesses in Echtzeit unter Berücksichtigung großer Datenmengen bieten sich DSMSs an. Im Gegensatz zu Datenbank-Management-Systemen (DBMSs) arbeiten DSMSs nicht mit persistenten Daten, sondern mit flüchtigen Datenströmen, auf denen Abfragen kontinuierlich und sequenziell ablaufen müssen [GO03].

Die Projektgruppe wird als Anwendungsfall für die Recommender-Funktionalitäten von Odysseus eine Mobile Application (App) für Endgeräte mit Betriebssystem *Android* entwickeln (Begründung der gewählten Technologie siehe Kapitel 6.3.1).

[1] *Odysseus*: The Event Processing System, `http://odysseus.informatik.uni-oldenburg.de`, besucht am 06.07.2015.

1.2.2 Problemstellung

Grundsätzlich soll die App Datenströme liefern und mithilfe von Odysseus diese Datenströme verarbeiten können und ihren Nutzern in unterschiedlichen Kontexten Empfehlungen für einen Besuch von unterschiedlichen Örtlichkeiten (*Locations*), wie z. B. Restaurants, Kneipen, Bars, Bistros, Cafés etc. geben. Die Empfehlungen müssen auf Basis von Nutzer- und diversen Kontextdaten generiert werden. Ziel ist es, durch eine hohe Qualität der Empfehlungen dem Nutzer in einem zeitlich relevanten Kontext einen großen Mehrwert zu liefern.

Die direkte oder indirekte Verknüpfung mit dem DSMS wird eine essenzielle Anforderung an die App sein. Die Verarbeitung von Kontextdaten wie z. B. temporären Präferenzen des Nutzers oder ggf. auch Wetter- oder Sensordaten, wird im Bereich des DSMS liegen. Die Verarbeitung und Speicherung von Stammdaten der App-Nutzer und Bewertungen aus der Vergangenheit wird vom DBMS übernommen. Für die Benutzerfreundlichkeit (Usability) der App müssen die Interaktionen mit dem Nutzer visuell und funktional ansprechend gestaltet werden. Neben der Identifikation und Anbindung potenzieller Datenquellen stellt die zeitkritische Verarbeitung der Datenströme eine weitere Herausforderung dar. Dem Nutzer sollten annähernd in Echtzeit Empfehlungen für Locations in seinem jeweiligen Kontext gegeben werden können. Neben der Auswahl und Implementierung bestehender Algorithmen von Odysseus wird die Projektgruppe gegebenenfalls Anpassungen an den Funktionalitäten vornehmen oder eigene Algorithmen entwickeln und in Odysseus integrieren müssen.

1.2.3 Entwurf des Anwendungsszenarios

In dem System interagieren die Akteure *Nutzer*, *Applikation*, *Recommender-System* und *Service Provider* untereinander. Der Nutzer kann mithilfe der App Bewertungen für bestimmte Besuche von Locations abgeben. Es ist nicht vorgesehen, dass lediglich eine Location als solche bewertet wird. Durch die Bewertung von Besuchen wird berücksichtigt, dass Locations in der Realität häufig Wochentags-Aktionen o. ä. anbieten und daher innerhalb einer Woche für ein und denselben Nutzer unterschiedlich attraktiv sein können. Für die Qualität der herausgegebenen Empfehlungen pflegt der Nutzer ein Profil, in das er bestimmte Präferenzen und weitere Angaben über sich selbst anlegen kann. Diese Stammdaten sind in einer Datenbank gespeichert und werden von der App bei Bedarf abgerufen. Zusätzlich besteht für den Nutzer die Möglichkeit, aktuelle Präferenzen anzugeben, die lediglich in einem bestimmten Kontext zu einem bestimmten Zeitpunkt gelten. Dadurch können präzisere Empfehlungen gegeben werden. Weiter kann der Nutzer die Empfehlung als solche bewerten. Damit können die vorausgesagten mit den tatsächlichen Präferenzen verglichen werden und damit die Qualität der Ergebnisse durch bestimmte Lernalgorithmen des RecSys verbessert werden. Das von der App genutzte RecSys verwendet Funktionalitäten, die durch Odysseus bereitgestellt werden. Die Datengrundlage, mit der die Empfehlungsalgorithmen arbeiten, wird aus mehreren Quellen sichergestellt. Neben den oben genannten Daten des Nutzers erhält das System Daten über Application Programming Interfaces (APIs) von Diensten wie z. B. Anbietern von Wetterdaten, sozialen Netzwerken oder Eventkalendern.

Die Projektgruppe lässt sich zum Start des Projektes offen, ob zusätzlich weitere Daten aus eigenen Hardwarekomponenten wie Lautstärke-Sensoren oder WLAN-Routern für die Verwendung im RecSys generiert werden sollen. Vorstellbar ist beispielsweise, über in Locations installierte Hotspots die Anzahl der eingewählten Gäste zu ermitteln, um Schätzungen über die Auslastung einer Location machen zu können. Durch die Hinzunahme derartiger Kontextdaten ließe sich die Qualität der Empfehlungen weiter optimieren.

1.3 Aufbau des Dokuments

Für das Grundverständnis über das Datenstrommanagementsystem und die Recommender-Funktionalitäten werden in dieser Dokumentation zunächst die Grundlagen in Kapitel 2 erläutert. Im Anschluss werden die Rahmenbedingungen, konkret die Projektrollen, die Stakeholder und das verwendete Vorgehensmodell für das Projekt vorgestellt (siehe Kapitel 3).

Die User Stories (siehe Kapitel 4) bilden den Kern der Anforderungsanalyse. Hier sind sämtliche Anforderungserhebungsmethoden gelistet und potenzielle Anforderungen bereits identifiziert.

Die tatsächlichen Anforderungen, die im Rahmen des Projekts umgesetzt werden sollen, werden in Kapitel 5 zunächst anhand von Use-Cases beschrieben. Die konkreten Anforderungen werden anschließend formuliert und in funktionale und nicht-funktionale Anforderungen unterteilt (Kapitel 5.2 und 5.3).

Das Systemkonzept (siehe Kapitel 6) besteht aus den Bereichen Domänenmodell (Kapitel 6.1), Komponenten (Kapitel 6.2), Architektur (Kapitel 6.3) und der grafischen Benutzeroberfläche (Kapitel 6.4). Das Domänenmodell zeigt sämtliche Entitäten, Attribute und Beziehungen, die für das System relevant sind. Daneben verschafft das Komponentendiagramm einen Überblick über die grundlegenden Komponenten des Systems. Die Architektur beschreibt insbesondere die Systemarchitektur der Middleware, der App und des Dashboards. Zudem werden an dieser Stelle Technologieentscheidungen begründet und aufgeführt sowie die Verteilungssicht auf das System beschrieben. Als Grundlage für die Implementierung der App und des Dashboards sind für die grafische Umsetzung als erstes Mockups erstellt worden. Diese werden im Abschnitt “Grafische Benutzeroberfläche“ vorgestellt und dienen als Einstieg in die Dokumentation der implementierten Anwendungsfälle im darauf folgenden Kapitel 7.

Das Implementierungs-Kapitel gliedert sich nach den erhobenen Anforderungen auf. Je nach Umfang und Komplexität der jeweiligen Anforderung sind hier die Abschnitte entsprechend strukturiert. Diese enthalten Beschreibungen der Implementierung und Screenshots zu den einzelnen Anwendungsfällen.

Im Anschluss an die Dokumentation von Testprotokollen und der Testautomatisierung (siehe Kapitel 8) wird in Kapitel 9 das Projekt rückblickend betrachtet und die Evaluation des Projekts dokumentiert. Die Projektreview (siehe Kapitel 9.1) befasst sich mit der Umsetzung der gestellten Anforderungen und begründet gegebenenfalls nicht vollständig implementierte Anforderungen. Zudem wird das Projekt unter nicht-inhaltlichen Aspekten rückblickend betrachtet und bewertet. Im Rahmen der Evaluation (Kapitel 9.2 und 9.3) werden grundlegende Zielsetzungen des Projekts einer Evaluation unterzogen.

Durch den Ausblick (siehe Kapitel 10), der potenzielle Erweiterungsoptionen und -szenarien im Anschluss an das Projekt aufzeigt, wird die Dokumentation abgeschlossen.

2 Grundlagen

In diesem Abschnitt sind wesentliche Grundlagen beschrieben, die für das weitere Verständnis dieser Dokumentation benötigt werden. Es handelt sich dabei um Ausarbeitungen, die zu Beginn der Projektgruppe erstellt worden sind und der Projektgruppe einen ersten Einstieg in die Thematik der Recommender-Systeme und Datenstrommanagement-Systeme geben.

2.1 Datenstrommanagementsysteme und Odysseus

In einer steigenden Zahl von Anwendungsfällen müssen große Mengen an mit hoher Datenrate kontinuierlich auftretenden Daten zeitnah verarbeitet werden. Beispiele für solche Szenarien sind die Verarbeitung von Sensordaten oder die laufende Überwachung von Zuständen. Diese kontinuierlichen Datenaufkommen mit theoretisch unbegrenzter Menge und oftmals variabler Datenrate werden Datenströme genannt. Die Reihenfolge der Datenelemente wird implizit durch den Zeitpunkt des Eingangs oder explizit durch einen zugeordneten Zeitstempel festgelegt [GO03]. Die eingehenden Datenelemente werden in der Regel sequenziell angeordnet und innerhalb des Datenstroms nicht mehr aktualisiert oder gelöscht, ebenso hat das Managementsystem keinen Einfluss auf das Eintreffen der Elemente [BBD+02]. Das große Aufkommen an Datenelementen stellt hohe Anforderungen an die zur Verarbeitung genutzten Managementsysteme, bedingt durch Datenmengen, Datenauftreten und zeitkritische Anforderungen von Anwendungen [Gei13]. Traditionelle Datenbankmanagementsysteme (DBMS) wurden entwickelt, um Anfragen auf persistenten Relationen auszuführen und eignen sich somit nur bedingt zur kontinuierlichen Verarbeitung von Datenströmen mithilfe von kontinuierlichen Anfragen. Aus diesen Gründen entstanden Datenstrommanagementsysteme (DSMS), welche als Alternative zu DBMS auf die Verarbeitung von Datenströmen spezialisiert sind. Ihr Hauptanwendungsbereich sind im Allgemeinen Punkte, an welchen kontinuierlich Daten anfallen, welche nicht ohne größere Probleme persistent gespeichert und nachträglich verarbeitet werden können, beispielsweise aufgrund der anfallenden Datenmengen oder unterschiedlicher eingehender Datenstrukturen - also dort, wo eine laufende Verarbeitung von Datenströmen notwendig ist [BBD+02].

Bei DSMS handelt es sich um ein vergleichsweise junges Forschungsfeld mit großem Forschungsbedarf und bereits vielfältigen Lösungsansätzen.

2.1.1 Unterschiede zwischen Datenstrom- und Datenbankmanagementsystemen

Aufgrund der Natur von Datenströmen ist es aus Speicherplatz- und Performanzgründen oft nur schwer möglich, die eingehenden Datenelemente in persistenten Relationen zu speichern und per Datenbankmanagementsystem (DBMS) nachträglich zu verarbeiten. Daher werden Datenstrommanagementsysteme (DSMS) entwickelt, welche als Alternative zu DBMS eine strombasierte Verarbeitung und gegebenenfalls spätere Archivierung der Daten ermöglichen. DSMS arbeiten ähnlich wie DBMS mit deklarativen Anfragen, welche durch das Managementsystem in logische Anfragepläne übersetzt, optimiert und durchgeführt werden. Diese Anfragen können im Falle von DSMS jedoch kontinuierlich auf einem Datenstrom ausgeführt werden, nicht nur punktuell oder anhand von Triggern auf persistenten Relationen, wie es bei DBMS in der Regel der Fall ist. Ein weiterer Unterschied ist die Form der ausgegebenen Informationen: DBMS produzieren eine Ergebnismenge zum Zeitpunkt der Anfrage, während DSMS durch kontinuierlich laufende Anfragen wieder einen Datenstrom als Ergebnis liefern

[BGJ+09]. DSMS können durch Anpassung der laufenden Anfragen schnell auf wechselnde Datenmodelle und Bedingungen reagieren und Anfragepläne zur Laufzeit optimieren. Während der Fokus von DBMS auf festen, einmalig vor der Ausführung optimierten Anfrageplänen und präzisen Ergebnissen basierend auf persistenten Relationen liegt, sind DSMS auf eine hohe Anpassbarkeit und möglichst genaue Approximation der eingehenden Datenelemente spezialisiert [BBD+02].

2.1.2 Datenstrommanagementsysteme

Bei DSMS werden laufend große Datenmengen in das System gegeben, während bei traditionellen DBMS traditionell eher Daten aus dem System heraus gezogen werden. DSMS führen sowohl laufende als auch ad-hoc Anfragen auf Datenströmen durch, welche aufgrund der sequenziellen Verarbeitung selbiger durch Gültigkeitsfenster in zu betrachtende Datenmengen eingegrenzt werden müssen – aufgrund von begrenzter Leistung und aus Speichergründen können Daten nur einmalig betrachtet werden [GO03]. Dies ist auch bedingt durch die Tatsache, dass DSMS in der Regel die zu verarbeitenden Daten im Hauptspeicher halten müssen, da die verfügbare Prozessorzeit je Element begrenzt ist und Zugriffe auf die Festplatte zusätzlich zu Latenzproblemen führen können. Aus diesem Grund werden die zu betrachtenden Elemente anhand von Algorithmen bestimmt [BBD+02]. Die Bestimmung der gültigen Datenelemente geschieht im Regelfall durch (gleitende) Zeitfenster, Tupelfenster oder Prädikatfenster. Der gängigste Ansatz sind Zeitfenster, durch welche nur Datenelemente innerhalb eines zeitlichen Abschnitts betrachten werden. Tupelfenster grenzen die zu verarbeitenden Elemente auf eine definierte Anzahl ein, während Prädikatfenster sich anhand der eingehenden Elemente definieren (bspw. Events). Hierbei kann es sich sowohl um absolute als auch um relative Fenster handeln, welche beispielsweise einen bestimmten absoluten Zeitraum in der Vergangenheit als gültig definieren – oder eine Anzahl Elemente seit Eingang des letzten Datenelements. Gültigkeits-Fenster ermöglichen somit die Eingrenzung der zum betrachteten Zeitpunkt gültigen Daten für die durchzuführenden Operationen und stellen einen Ausschnitt dar und realisieren somit den Kompromiss zwischen möglichst genauer Approximation des Datenstroms und Speicherbedarf [BGJ+09, GO03]. Hierdurch wird die inkrementelle Erstellung von Ergebnismengen ermöglicht, welche dann mit bereits vorhandenen Ergebnissen oder persistenten Relationen aktualisiert werden können. Weiterhin kann ein Load-Shedding- Algorithmus implementiert werden, welcher bei hoher Systemlast anhand von hinterlegten Strategien und Approximation Elemente des Datenstroms entfernt und somit Überlastung vorbeugt oder die Verarbeitung innerhalb eines festgelegten Zeitfensters sicherstellt [Gei13].

Da mehrere Anfragen auf demselben Datenstrom laufen und weiterhin dynamisch zur Laufzeit aktualisiert, neu registriert oder gelöscht werden können, erfordern diese eine spezielle Optimierung der Anfragepläne zur Laufzeit, um eine zeitnahe Verarbeitung der Daten gewährleisten zu können [BBD+02]. Die Anfragen in DSMS werden deklarativ in SQL-Dialekten oder imperativ über algebraische Operatoren formuliert. Auch eine Kombination aus beidem ist je nach DSMS möglich [GO03]. Obwohl bereits einige DSMS im akademischen und kommerziellen Bereich entwickelt wurden und verfügbar sind, gibt es jedoch bisher keinen einheitlichen Standard für SQL-Sprachen, Operatoren oder deren Semantik [Gei13].

Zu den Aufgaben eines DSMS gehört auch die Kombination von bereits vorhandenen Daten (bspw. aus persistenten Relationen) mit neu eingehenden Daten oder die Zusammenführung verschiedener Datenströme – beispielsweise die Fusion von Sensordaten zur Beurteilung des Gesamtbilds [BGJ+09, BBD+02].

Die möglichen Anwendungsbereiche für DSMS sind vielfältig und nehmen stetig zu. Beispiele für mögliche Anwendungsfelder sind die Energiebranche oder Börsentransaktionen. Beim dezentralen Energiemanagement müssen zunehmend größere Mengen an Sensordaten von mehr und weit verteilten Anlagen verarbeitet werden. Die dafür zuständigen Systeme sind unflexibel, aufwändig anzupassen und oftmals nicht für steigende Datenraten ausgelegt. Ein DSMS könnte die Flexibilität in Bezug auf wechselnde Gegebenheiten erhöhen und große Datenmengen bzw. –raten leichter händelbar machen [BGJ+09]. Im Fall der Überwachung von Börsentransaktionen sind DSMS in der Lage, große Mengen an dynamisch anfallenden Daten zu verarbeiten um Trends zu identifizieren, Korrelationen aufzudecken, Zukunftsdaten vorherzusagen und Kursgewinn-Chancen sichtbar zu machen [GO03].

Als Beispiele für akademische Projekte im Bereich DSMS lassen sich unter anderem nennen: Aurora, COUGAR, NiagaraCQ, OpenCQ, STREAM, TelegraphCQ, Odysseus [GO03]. Auch im kommerziellen Bereich sind DSMS seit einiger Zeit in der Entwicklung, wie InfoSphere (IBM), Oracle Event Processing (Oracle), StreamInsight (Microsoft) oder SAP Sybase Event Stream Processor (Sybase) [Gei13].

2.1.2.1 Kontinuierliche Anfragen und algebraische Operatoren

Bei kontinuierlichen Anfragen handelt es sich um Anfragen, welche eine bestimmte Menge an Elementen des Datenstroms verarbeiten. Hierbei kann die Verarbeitung der zu betrachtenden Elemente nach zeitbasierten (periodisch), elementbasierten (bei Ankunft eines neuen Elements) oder eventbasierten (vordefinierte Events) Modellen geschehen [Gei13].

Ein Problem kontinuierlich laufender Anfragen (oder Operatoren) auf Datenströmen sind blockierende Operatoren, welche eine weitere Verarbeitung des Datenstroms verhindern, beispielsweise weil auf das Ende des Datenstroms gewartet werden muss bevor eine Ergebnismenge errechnet werden kann [BGJ+09, GO03]. Hierzu lassen sich als Beispiel Operatoren nennen, welche die Daten sortieren oder aggregieren (SUM, COUNT, MIN, MAX, AVG) [BBD+02]. Als Lösung für dieses Problem werden bei DSMS nicht- blockierende Operatoren verwendet, welche inkrementell oder periodisch Ergebnisse ausgeben [Gei13]. Partiell blockierende Operatoren liefern Zwischen- und Endergebnisse und sind daher ebenfalls nicht geeignet für die Verwendung in kontinuierlichen Anfragen. Hier kommen die bereits erläuterten Gültigkeits-Fenster zur Anwendung, welche den Operatoren die Einteilung und Verarbeitung des Datenstroms ermöglichen. Weiterhin können Operatoren eine Approximation des Datenstroms verwendet werden oder inkrementell gestaltet werden, wodurch sie das Ergebnis mit Zwischenergebnissen aktualisieren können beim Eintreffen neuer Daten [Gei13]. Da nicht alle blockierenden Operatoren problemlos weggelassen werden können, versucht man, sie durch ähnlich funktionierende nicht-blockierende Operatoren zu ersetzen. Ein Beispiel hierfür ist eine angepasste Version eines sort-Operators, welcher den Datenstrom lokal neu anordnet und somit die Reihenfolge der Elemente beeinflussen kann [BBD+02].

2.1.2.2 Aufbau von DSMS

Der funktionelle Aufbau von DSMS lässt sich wie folgt skizzieren: um einen eingehenden Datenstrom im DSMS zur Verarbeitung verfügbar zu machen, wird dieser auf der Eingabeseite zwischengespeichert, sortiert und die Elemente in das Datenformat es DSMS konvertiert. Dies geschieht zumeist in ein relationales Format und resultiert in der Darstellung in Tupeln [KS09]. Danach folgt das Einreihen in die Warteschlange für die Verarbeitung durch die Operatoren. Ein Queue Manager verwaltet die

Warteschlangen sowie die Zwischenspeicher und ist für das Ressourcenmanagement zuständig. Der Storage Manager ermöglicht die Kombination von persistenten Daten mit (Zwischen-)Ergebnissen aus dem Datenstrom. Der Scheduler entscheidet über die Reihenfolge der Ausführung der Anfragen und interagiert mit dem Query Processor, welcher letztendlich die Operatoren ausführt. Eine Monitoring-Komponente erstellt Statistiken über Leistung, Durchsatz und Verzögerungen, welche vom Query Optimizer zur Optimierung von Anfrageplänen verwendet werden können [Gei13].

Die deklarativen Anfragen werden vom DSMS zunächst in logische Anfragepläne übersetzt, danach optimiert und anschließend in physische, auf den Daten durchführbare Anfragepläne übersetzt und ausgeführt [KS09]. Sollten mehrere Anfragen vorliegen, so werden diese vor der Optimierung zu einem großen Anfrageplan zusammengefasst [GO03]. Die Anfragepläne enthalten Operatoraufrufe, Zwischenspeicher und die Struktur der Synopsen, also mögliche Algorithmen sowie Datenstrukturen und Beschreibung des Datenstroms [BBD+02].

Abbildung 2.1 illustriert den abstrakten Aufbau der Architektur von DSMS.

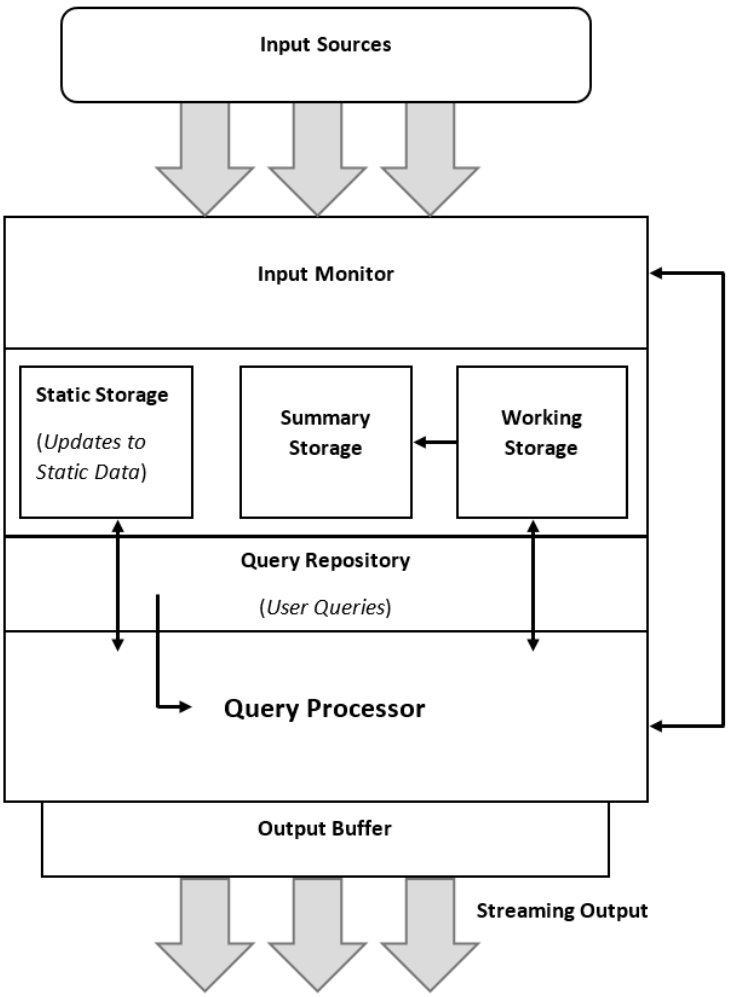


Abbildung 2.1: Abstrakte Darstellung der Architektur eines DSMS (nach [GO03]).

Die allgemeinen Komponenten eines DSMS werden später noch am Beispiel Odysseus exemplarisch erläutert, ohne jedoch zu tief auf die algebraischen Mechanismen im Hintergrund einzugehen. Interessierte können sich in die Implementation von Semantik und kontinuierlichen Anfragen unter [KS09] einlesen.

2.1.3 Odysseus

Odysseus ist als Forschungsprototyp an der Universität Oldenburg entstanden und teilt sich in zwei Haupt-Applikationen auf: Odysseus Studio und Odysseus Server. Odysseus Server stellt hierbei das System mit Verarbeitungsmechanismen, Anfrageschnittstelle, Benutzerverwaltung und anderem bereit [The15a]. Odysseus Studio dient hauptsächlich der Administration des Servers und der Entwicklung von Anfragen über den eingebauten Editor sowie der Verwaltung von Projekten [The15b]. Durch die vollständige Open Source-Implementation in Java ist Odysseus betriebssystemunabhängig einsetzbar und umfassend erweiterbar [The15c].

Bei Odysseus handelt es sich nicht um ein fertiges DSMS, sondern vielmehr um eine Plattform welche ein Framework für die anpassbare Entwicklung eines DSMS bereitstellt [BGG+10]. Durch die Erweiterbarkeit und Modularität der Komponenten ist Odysseus umfangreich an verschiedene Anwendungsbereiche anpassbar und ermöglicht unter anderem die Implementierung von Verarbeitungsmechanismen für verschiedenartige Datenmodelle [BGJ+09]. Hierbei stellt Odysseus in der Regel die mittlere Ebene eines Drei-Schicht-Architekturmodells dar und vermittelt zwischen Datenquellen und Anwendungsebene [The15a].

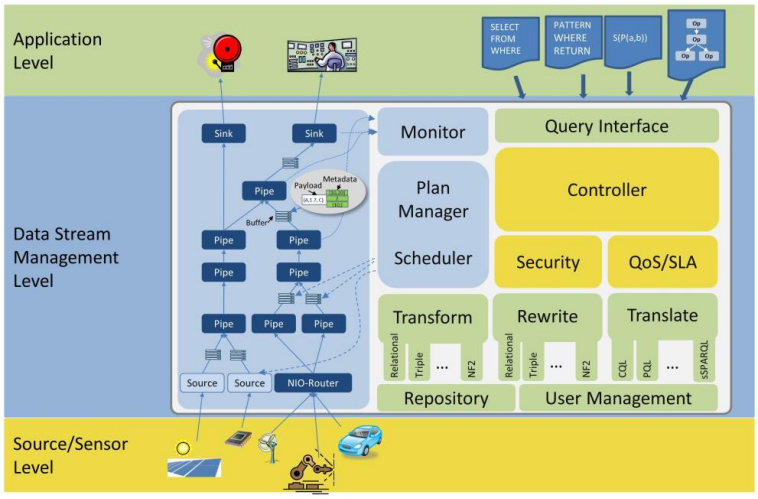


Abbildung 2.2: Architektur von Odysseus [The15a].

Aufgrund der Flexibilität von Odysseus kann dieses für vielfältige Schwerpunkts- Forschungen an DSMS in diversen Anwendungsdomänen eingesetzt werden, unter anderem bei der Überwachung dezentraler Energieversorgungsanlagen und für die Sensordatenfusion in Fahrerassistenzsystemen. [BGJ+09].

Um eine möglichst hohe Flexibilität zu gewährleisten, sind die Komponenten von Odysseus in Fixpunkte und Variationspunkte unterteilt. Fixpunkte bilden hierbei vorwiegend die internen Verwaltungsstrukturen des Frameworks ab und dienen als Schnittstelle, während Variationspunkte der Steuerung der Verarbeitung von Daten dienen und eine Anpassung an spezifische Anforderungen ermöglichen. Die implementierten Komponenten decken den gesamten typischen Arbeitsbereich von DSMS ab, wie die Eingabe von Anfragen und die anschließende Weiterverarbeitung im DSMS [BGJ+09]. Alle Komponenten sind durch die OSGi-basierte Architektur in einzelne Pakete gekapselt, wodurch eigene Komponenten implementiert oder je nach Anwendungsfall auch weggelassen werden können [BGG+10].

Nachfolgend ein kurzer Überblick über die einzelnen Komponenten von Odysseus.

2.1.3.1 Translate-Komponente

Die Übersetzungskomponente ist für das Übersetzen der Anfragen in logische Anfragepläne zuständig. Ihr liegt eine logische Algebra-Basis zugrunde, welche anwendungsunabhängige Verwaltungsstrukturen bereitstellt und als Fixpunkt dient. Die Algebra-Basis kann um zusätzliche Operatoren ergänzt werden und dient somit als Variationspunkt innerhalb des Frameworks [BGJ+09]. Die Anfragen können sowohl deklarativ in Form von SQL-Sprache als auch prozedural über Operatoren formuliert werden. Grundsätzlich können auch eigene SQL-Dialekte und Operatoren implementiert und verwendet werden [BGG+10].

2.1.3.2 Restruct-Komponente

Die Restrukturierung und Optimierung der logischen Anfragepläne ist Aufgabe der Restrukturierungskomponente. Die Komponente ist der regelbasierte Fixpunkt und stellt Funktionen zur internen Verwaltung von Regeln und deren Anwendung auf logische Anfragepläne bereit. Durch benutzerdefinierte Regeln kann die Komponente erweitert werden und stellt somit einen Variationspunkt dar [BGJ+09].

2.1.3.3 Transform-Komponente

Innerhalb der Transformationskomponente werden die logischen in physische Anfragepläne übersetzt. Es erfolgt eine regelbasierte Algorithmen-Auswahl der verschiedenen Operatoren und die Evaluation der Anfragepläne durch ein Kostenmodell. Auch hier dient eine Algebra-Basis als Fixpunkt. Durch die Möglichkeit zur Ergänzung mit weiteren ausführbaren Operatoren ist der Variationspunkt der Komponente gegeben [BGJ+09].

2.1.3.4 Execute-Komponente

Die Ausführungskomponente ist für die abschließende Ausführung der vorher ermittelten physischen Anfragepläne zuständig. Der Scheduler stellt als Fixpunkt die Ausführungsumgebung bereit, während die Strategien für den Zugriff und die Ausführung der Anfragepläne den Variationspunkt innerhalb der Komponente abbilden [BGJ+09].

2.1.3.5 Monitor-Komponente

Die Überwachung der Ausführung physischer Anfragepläne geschieht durch die Monitoring-Komponente. Ein Publish-Subscribe Mechanismus verteilt und verarbeitet Metadaten über die Operatorausführung und ist der Fixpunkt der Komponente. Die Art der Metadaten und ihre Verarbeitung ist erweiterbar und somit der Variationspunkt [BGJ+09].

2.2 Ansätze für Datenstrombasierte Recommendersysteme

Durch die immer größeren werdenden und schneller produzierten Datenmengen aus verschiedensten Quellen ist es nicht mehr möglich alle Daten auf einmal zu verarbeiten, weshalb neue Ansätze zur Datenverarbeitung entwickelt werden müssen. So auch bei den derzeitigen Ansätzen von Recommendersystemen. Um diesen Datenstrom bewältigen zu können werden in dieser Ausarbeitung die Ansätze StreamRec[CLEM11] und Distributed Stochastic Gradient Descent[AJT11] vorgestellt.

Als erstes wird in diesem Abschnitt eine Einführung in die Thematik gegeben. Dabei werden kurz die Ansätze vorgestellt, die in dieser Ausarbeitung behandelt werden und ein Überblick über den aktuellen Stand der Forschung gegeben. Danach folgen die Grundlagen und Rahmenbedingungen für datenstrombasierte Recommendersysteme. In Kapitel 2.2.2 und 2.2.3 wird dann auf die beiden Ansätze für datenstrombasierte Recommendersysteme eingegangen. Zum Schluss folgt ein Fazit bei dem die beiden Ansätze gegenübergestellt, auf Optimierungspotenziale und die in den Grundlagen definierten Anforderungen eingegangen wird.

Im Bereich der datenstrombasierten Recommendersystemen gibt es erst sehr wenige Arbeiten. So gehört StreamRec von Badrish Chandramouli et al.[CLEM11] und Distributed Stochastic Gradient Descent von Muqeet Ali et al.[AJT11] zu den wenigen veröffentlichten Ansätzen, die eine Datenstromverarbeitung bei Recommendersystemen erlauben. Der Ansatz von Chandramouli basiert auf dem objektbasierte kollaborativen Filtern und wurde mit dem Datenstrommanagementsystem(DSMS) StreamInsight[1] von Microsoft prototypisch umgesetzt. Bei dem Ansatz von Ali dagegen wird die Matrix-Faktorisierung verwendet. Die Besonderheit hierbei ist, das er die Ansätze von Abernethy[ACLS07] und Gemulla[GNHS11] weiterentwickelt hat, damit eine Verteilung der Arbeitslast auf mehrere Server möglich ist. Datenstrombasierte Recommendersysteme bauen stark auf die Ansätze der inkrementellen Recommendersysteme auf. Beispielsweise wurde von Vinagra[VJG14] bereits ein Ansatz auf Basis des Stochastic Gradient Descent, um eine inkrementelle Verarbeitung bei Recommendersystemen zu ermöglichen, entwickelt.
In dieser Ausarbeitung wird hauptsächlich auf die Datenstromverarbeitung von Recommendersystemen eingegangen. Auf eine detaillierte Beschreibung der Berechnung der Vorhersagen wird daher verzichtet.

2.2.1 Grundlagen

Da datenstrombasierte Recommendersysteme nur eine Weiterentwicklung herkömmlicher Recommendersysteme sind, wird in diesem Abschnitt eine kurze Einführung in die Thematik gegeben. Zusätzlich werden die Besonderheiten und Herausforderungen der Datenstromverarbeitung erläutert, um Rahmenbedingungen für die datenstrombasierten Recommendersysteme festzulegen.

[1] `https://msdn.microsoft.com/en-us/sqlserver/ee476990.aspx`

Das Ziel der Recommendersysteme ist nun die fehlenden Bewertungen vorherzusagen, um Empfehlungen für die Benutzer zu ermitteln. Um dieses Ziel zu erreichen, gibt es eine Vielzahl von verschiedenen Ansätzen. Unter anderem lassen sich alle Ansätze entweder dem kollaborativen Filtern (Collaborative Filtering, CF) oder den inhaltsbasierten Recommendersystemen zuordnen. Beim kollaborativen Filtern werden die Bewertungen anderen Benutzer herangezogen. Entweder es werden Benutzer ermittelt, die einen ähnlichen Geschmack haben oder es wird eine Ähnlichkeitsmatrix erstellt um Zusammenhänge zwischen Objekten zu erkennen. Bei inhaltsbasierten Recommendersystemen dagegen werden die Objekte direkt analysiert und z. B. anhand von Keywords oder Beschreibungen werden ähnliche Objekte ermittelt [AT05]. In dieser Ausarbeitung lassen sich alle Ansätze dem kollaborativen Filtern zuordnen. Im Detail handelt es sich um die Matrix-Faktorisierung und dem objektbasierte CF.

Als weitere Grundlage wird die Verarbeitung von Datenströmen benötigt. Die größte Besonderheit bei der Datenstromverarbeitung gegenüber herkömmlicher Datenverarbeitung ist, dass es sich immer um einen potenziell unendlichen Datenstrom handelt. Des Weiteren können die Daten sehr unregelmäßig eintreffen. Daher ist es nicht möglich alle Daten zu speichern und zu verarbeiten. Um dieses Problem zu lösen wird nur ein kleiner Teil aller Daten verarbeitet. Hierfür werden beispielsweise Fenster verwendet, die nur eine bestimmte Menge an Elementen oder einen bestimmten Zeitraum betrachten [BBD+02]. Diese Elemente werden auch direkt verarbeitet. So wird kontinuierlich dieselbe Anfrage auf die eintreffenden Daten ausgeführt, im Gegensatz zu der herkömmlichen Datenverarbeitung, bei der jede Anfrage üblicherweise nur einmalig ausgeführt wird [Krä07]. Eintreffende Daten werden auch Events genannt, daher wird es häufig auch Event-Processing genannt. Events können aus vielen verschiedenen Quellen stammen.

Auf Basis dieser beiden Thematiken lässt sich nun ein allgemeiner Ablauf eines datenstrombasierten Recommendersystems ableiten. Hierfür ist in Abbildung 2.3 die Ablauf eines herkömmlichen Recommendersystems zu sehen. Damit nun das Recommendersystem den potenziell unendlichen Datenstrom bewältigen kann, muss dieses Model angepasst werden. Die herkömmlichste Methode dies zu tun, ist ein Fensteroperator. Dieser gibt jedem eintreffenden Rating Event einen Gültigkeitszeitraum. So werden nur Bewertung bearbeitet, die noch gültig sind.

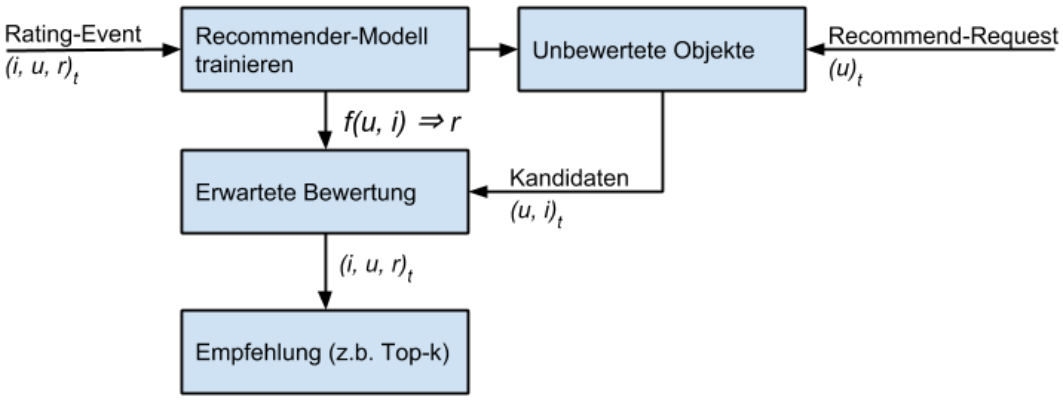


Abbildung 2.3: Allgemeiner Ablauf eines datenstrombasierten Recommendersystems.

Somit ist die wichtigste Anforderung an ein datenstrombasierten Recommendersystem die adäquate Verarbeitung des potenziell unendlichen Datenstroms. Im Detail bedeutet das, dass der Ansatz in irgendeiner Weise die eintreffenden Daten in kleinere Mengen aufteilen und verarbeiten sollte. Des Weiteren sollte dadurch nicht die Qualität der Vorhersagen drastisch abnehmen. Sowie eine schnelle Bearbeitung gewährleistet werden.

Auf Basis dieser Informationen wurden bereits verschiedene Ansätze für datenstrombasierte Recommendersysteme entwickelt. Zwei davon werden in den folgenden Abschnitten vorgestellt. Einerseits handelt es sich um Stream ein Ansatz von Chandramouli et al.[CLEM11] und andererseits um Distributed Stochastic Gradient Descent speziell für die Verarbeitung von Datenströmen von Ali et al.[AJT11], die in den folgenden Kapiteln vorgestellt werden.

2.2.2 StreamRec

Dieser Ansatz lässt sich dem objektbasierte Collaborative Filtering zuordnen. Dabei wird die Ähnlichkeit zwischen Objekten berechnet und dem Benutzer die ähnlichsten Objekte empfohlen. Er wurde von Microsoft-Mitarbeitern entwickelt und wurde prototypisch in zwei Anwendungen auf Basis von dem DSMS StreamInsight von Microsoft umgesetzt. Die Besonderheit bei diesem Ansatz ist, dass für die Umsetzung nur Basisoperatoren vom DSMS benutzt werden können und keine zusätzlichen implementiert werden müssen.

Die Events sind immer nachdem Schema *(Timestamp, StreamId, UserId, ItemId, Rating)* aufgebaut. Die eintreffenden Events werden in zwei Kategorien unterteilt. Einerseits gibt es Update-Events, die eine neue Bewertung eines Objekts von einem Benutzer enthält. Diese werden dadurch gekennzeichnet, dass die StreamId den Wert 0 hat. Anderseits gibt es Recommend-Events, die eine Anfrage nach Empfehlungen für einen bestimmten Benutzer darstellen. Die wiederum dadurch gekennzeichnet werden, dass die StreamId den Wert 1 hat. Ein Recommend-Event lässt sich wiederum in Edge- und Point-Events unterteilen. Ein Edge-Event hat keine begrenzte Lebensdauer. So werden dauerhaft die Empfehlungen aktualisiert. Das Point-Event ist dagegen nur für einen genauen Zeitpunkt gültig. Weshalb es auch nur die Empfehlungen für genau diesem Zeitpunkt erhält.

Das Vorgehen von StreamRec wird in die Schritte Model Building und Recommendation Generation unterteilen.

2.2.2.1 Model Building

Im Schritt Model Building werden alle Update-Events verarbeitet und Ähnlichkeiten zwischen den Objekten berechnet. Um die Ähnlichkeit zu berechnen, wird die Kosinus-Ähnlichkeit benutzt, wie sie in Gleichung 2.1 zu sehen ist. Um die Ähnlichkeit zu ermitteln, wird der Funktion alle Bewertungen von zwei Objekten übergeben [CLEM11]. Die Kosinus-Ähnlichkeit wird unteranderem auch in ähnlicher Form von Amazon verwendet und eignet sich besonders gut für diesen Anwendungsfall [LSY03].

$$sum(i_p, i_q) = k \frac{\overrightarrow{i_p} * \overrightarrow{i_q}}{||\overrightarrow{i_p}|| ||\overrightarrow{i_q}||} \tag{2.1}$$

In die Berechnung des Modells fließen aktuelle Bewertungen, sowie einige ältere Bewertungen ein. Ein vorher angegebenes Zeitfenster legt fest, welche Bewertung aktuell sind. Zusätzlich wird

mit dem *AlterLifetime*-Operator der Gültigkeitszeitraum von älteren Bewertungen geändert, damit sie sich wieder im gültigen Fenster befinden. Anschließend wird ein Self-Join über den Benutzer(*UserId*) durchgeführt. Hierfür wird der *Temporal-Join* benutzt. Der Unterschied zu einem normalen Join ist, dass bei einem *Temporal-Join* nur Bewertungen mit einander gejoint werden, bei denen sich der Gültigkeitszeitraum überschneidet. Dadurch bildet jedes vom Benutzer bewerteten Objekt mit jedem anderen von ihm bewerteten Objekt ein Paar nachdem Schema *(UserId, Item1, Rating1, Item2, Rating2)*.

Auf diese Menge wird nun ein *Group*-Operator über beide Objekte(*Item1* & *Item2*) durchgeführt. Auf die entstehenden Gruppen wird nun die Kosinus-Ähnlichkeit nach der Gleichung 2.1 als Aggregationsfunktion ausgeführt. Daraus resultiert für jedes Objekt eine Liste von allen anderen Objekte und deren Ähnlichkeit zueinander. Zum Schluss kann das Ergebnis entweder abgespeichert werden oder für das weitere Model Building im Speicher gehalten werden.

Dieser Prozess wird jedes Mal durchgeführt, sobald eine neue Bewertung eintrifft [CLEM11]. Zur Veranschaulich dieses Vorgehen ist in Abbildung 2.4 der entsprechende Anfrageplan zu sehen.

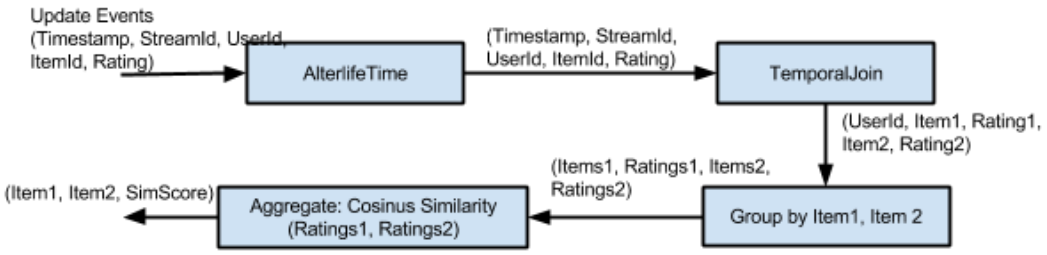


Abbildung 2.4: Anfrageplan vom Model Building.

2.2.2.2 Recommendation Generation

Im Schritt Recommendation Generation werden alle Recommend-Events verarbeitet und die Liste der empfohlenen Objekte zurückgegeben.

Sobald ein Recommend-Event eintrifft werden als erstes alle vom Benutzer bewerteten Objekt abgefragt, indem wieder ein *Temporal-Join* über die *UserId* durchgeführt wird. Diese Daten werden nun wieder nach (*Item1* & *Item2*) gruppiert, wodurch die zuvor berechnete Ähnlichkeit ermittelt wird. Da es sich bisher nur um ein Ähnlichkeitsmaß handelt und um noch keine erwartete Bewertung muss die Ähnlichkeit nun in eine konkrete Bewertung umgewandelt werden. Hierfür wird, statt die zuvor verwendete Kosinus-Ähnlichkeit als Aggregationsfunktion, die gewichtete Summe P nach der Gleichung 2.2 berechnet.

$$P(u_t, i) = \frac{\sum_{l \in \mathcal{L}} sim(i,l) * r_{u_q,l}}{\sum_{l \in \mathcal{L}} |sim(i,l)|} \tag{2.2}$$

Da bei diesen erwarteten Bewertungen auch noch bewertete Objekte enthalten sind, werden sie mit einem *Left-Join* entfernt. Aus diesen erwarteten Bewertungen werden nun wiederum die Top-k ermittelt. Das sind nun die endgültigen Empfehlungen, die an den Benutzer weitergeleitet werden.

In Abbildung 2.5 ist noch einmal der Anfrageplan vom Recommendation Generation zum besseren Verständnis abgebildet.

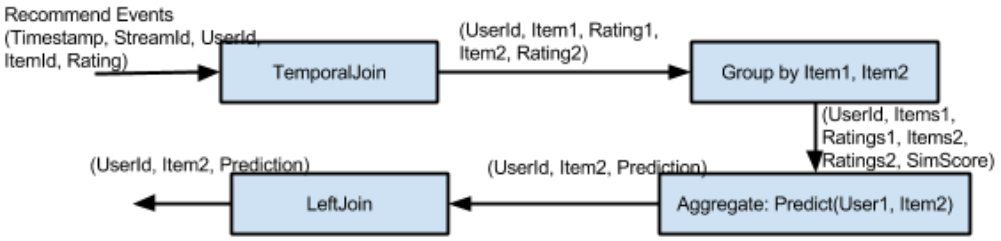


Abbildung 2.5: Anfrageplan vom Recommendation Generation.

Dieser Ansatz wurde in zwei mobilen Applikation prototypisch umgesetzt. Die erste Anwendung erstellt einen persönlichen Newsfeed, ähnlich wie Digg[2]. Mit der zweiten Anwendung lassen sich Filme bewerten und dem Benutzer werden entsprechend Empfehlungen zurückgegeben. Beide Anwendungen wurden auch als Webapplikation umgesetzt. Bei der mobilen Version wurde ein pushbasierter Ansatz benutzt, wofür Edge-Events verwendet wurden. Bei der Webversion ein pullbasierter Ansatz angewendet, wohingegen Point-Events benutzt wurden.

Zu einer Validierung des Ansatzes in Folge der prototypischen Implementierung kam es nicht, weshalb eine Bewertung der Performance nicht stattfinden kann. Es wird allerdings davon ausgegangen, dass durch die Nutzung nativer Operatoren eine gewisse Performance gewährleistet werden kann und durch dem zugrunde liegenden DSMS auch eine Skalierung stattfinden kann [CLEM11].

2.2.3 Distributed Stochastic Gradient Descent

Ein weiterer Ansatz für datenstrombasierte Recommenandersysteme ist der Distributed Stochastic Gradient Descent-Algorithmus (DSGD). Dieser Ansatz nutzt die Matrix-Faktorisierung, um die zu erwartenden Bewertungen zu ermitteln. Es gibt zwei Varianten dieses Algorithmus. Einerseits ist es möglich auf einer festen Datenbasis zu arbeiten (Batch DSGD) und andererseits wurde der Algorithmus auch für die Verarbeitung von Datenströmen angepasst (Streaming DSGD). In diesem Abschnitt wird ausschließlich auf die Verarbeitung von Datenströmen mit diesem Algorithmus eingegangen. Ein wesentlicher Unterschied zum vorherigen Ansatz ist, dass auch eine Verteilung die Rechenlast auf mehrere Server bei dem Algorithmus ausdrücklich berücksichtigt wird. Es werden jedoch nur Rating-Events berücksichtigt. Auf die Verarbeitung von Recommend-Events wird nicht genauer eingegangen.

Um die Matrix-Faktorisierung durchzuführen wird der Stochastic Gradient Descent(SGD) benutzt. Eine detaillierte Beschreibung von SGD findet sich in Abschnitt 2.8. Grundlegend lässt sich zu der Matrix-Faktorisierung sagen, dass versucht wird fehlende Werte einer Matrix R zu berechnen, indem zwei zusätzliche Matrizen U und V erstellt werden, die die Länge und die Höhe der eigentlichen Matrix R haben. Die Matrizen U und V werden üblicherweise mit zufälligen Werten initialisiert. Diese Matrizen müssen nun mit Werten befüllt werden, die multipliziert miteinander ungefähr die Werte der

[2] `http://digg.com`

Matrix R ergeben sollen. Diese Werte zu finden ist ein Optimierungsproblem welches mit SGD gelöst werden kann. Hierfür werden folgende Funktionen benutzt.

$$u_i \leftarrow u_i - 2\eta((u_i^T v_j - R_{ij})v_j + \frac{\lambda}{m}u_i) \tag{2.3}$$

$$v_j \leftarrow v_j - 2\eta((u_i^T v_j - R_{ij})u_i + \frac{\lambda}{n}v_j) \tag{2.4}$$

Ziel dieser Funktionen ist es sich mit der Schrittgröße η der Nullstelle anzunähern. Bei der Nullstelle befindet der optimierte Wert. Sobald dieser Wert erreicht wurde, wird der entsprechende Wert in U bzw. V angepasst. Nun lässt sich die jeweilige Spalte bzw. Zeile in R, für die dieser neue Wert ermittelt wurde, aktualisieren. Hierfür wird lediglich jeder Wert in der Spalte bzw. Zeile neuberechnen.

Um Overfitting zu verhindern gibt es die sogenannten Bestrafungsterme $\frac{\lambda}{m}u_i$ und $\frac{\lambda}{n}v_j$. Sie verhindern zu hohe Ausreißer. Hierbei definiert λ wie hoch diese Strafe ausfällt. Falls sie zu hoch ist, ist die Vorhersage zu ungenau. Falls sie hingegen zu niedrig ist, kommt es zum Overfitting. Das bedeutet, dass das gelernte Model nur auf die Trainingsdaten anwendbar ist. Da die Matrix R eine sehr große Menge an Benutzer und Objekten enthalten kann, kann die Aktualisierung auch viel Zeit in Anspruch nehmen. Um dieses Problem zu lösen wird zusätzlich eine Verteilung der Arbeitslast vorgenommen. Hierfür wurde unteranderem der Ansatz von MapReduce aufgegriffen. Zu diesem Zweck enthält der Algorithmus drei verschiedenen Rollen. Der *Master* verteilt eintreffende Events auf die *Worker*. Diese verarbeiten die Events und beziehen entsprechende Benutzer- und Objektmatrizen vom *Matrix Store*. Zum Schluss wird das Ergebnis zurück an den Master gesendet

Bevor die Verarbeitung beginnen kann muss die Rating-Matrix R, der Benutzervektor U und der Objektvektor V in Blöcke unterteilt werden, die auf die Worker verteilt werden können. Die Anzahl der Blöcke B ergibt sich aus der Anzahl der zu Verfügung stehender Worker d. Die Höhe und Breite eines Blocks entspricht dabei der Menge an Workern. So ergeben sich d^2 gleichgroße Blöcke [AJT11]. So ergeben sich folgende Matrizen.

$$R = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1d} \\ B_{21} & B_{22} & \cdots & B_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ B_{d1} & B_{d2} & \cdots & B_{dd} \end{bmatrix} U = \begin{bmatrix} U_1 & \cdots & U_d \end{bmatrix} V = \begin{bmatrix} V_1 & \cdots & V_d \end{bmatrix}$$

Ein Rating Event ist nach dem Schema (i, j, R_{ij}) aufgebaut. Das bedeutet der Benutzer i hat das Objekt j mit R_{ij} bewertet. Dieses Event trifft immer beim *Master* ein. Er prüft nun in welchem Block B_{pq} sich dieses Rating befindet. Dementsprechend wird das Event nun an *Worker* p weitergeleitet. Dieser speichert das Event vorerst ab, bis der *Master* ihm einen Block zum bearbeiten zuweist. Zu diesem Zweck wählt er aus dem Pool, der aus nicht gesperrten Blöcken besteht, Blöcke aus und weist sie den entsprechenden Workern zu. Bevor ein Worker mit der Optimierung beginnt, werden alle Blöcke in Zeile p und Spalte q gesperrt. Das ist nötig, da nachdem die Vektoren zur Berechnung der Matrix optimiert wurden, alle Werte in der jeweiligen Zeile und Spalte neu berechnet werden müssen. Des Weiteren muss er vom *Matrix Store* die entsprechende Matrix U_p und V_q beziehen. Nun beginnt der eigentlichen Optimierungsprozess, wie er bereits im vorherigen Absatz beschrieben wurde. Anschließend werden die optimierten Matrizen wieder zum *Matrix Store* gesendet. Das ist nötig damit alle *Worker* mit konsistenten Matrizen arbeiten. Zum Schluss wird der optimierte Block B_{pq} dem *Master* übermittelt und die Blöcke in Zeile p und Spalte q wieder freigegeben. Dieses Vorgehen ist in Algorithmus 1 noch einmal zum besseren Verständnis in Pseudocode abgebildet.

Algorithm 1 Streaming DSGD

Master
if an event (i, j, R_{ij}) received **then**
  Let B_{pq} be the block to wich (i, j, R_{ij}) belongs.
  Forward (i, j, R_{ij}) to worker p.
end if
Choose a unlocked block B_{pq}.
Lock all blocks in row p and column q.
Send (p, q) to worker p.
if (p, q) is received from a worker **then**
  Unlock all blocks in row p and column q.
end if
Worker
if an event (i, j, R_{ij}) received **then**
  save (i, j, R_{ij})
end if
if (p, q) is received **then**
  Request U_p from matrix store.
  Request V_q from matrix store.
  when U_p and V_q received
  for each (i, j, R_{ij}) in block B_{pq} **do**
    $u_i \leftarrow u_i - 2\eta((u_i^T v_j - R_{ij})v_j + \frac{\lambda}{m}u_i)$
    $v_j \leftarrow v_j - 2\eta((u_i^T v_j - R_{ij})u_i + \frac{\lambda}{n}v_j)$
  end for
  Send U_p to matrix store.
  Send V_q to matrix store.
  Send (p, q) to master.
end if

Der streaming DSGD-Algorithmus wurde auf Basis von Apache Strom[3] prototypisch implementiert. Apache Strom ist ein Framework zur Anwendung vom MapReduce-Algorithmus auf Datenströme. Der batch DSGD-Algorithmus wurde mit Hadoop umgesetzt.

Abschließend fand eine Validierung des Ansatzes statt. Hierbei wurden die beiden Varianten des DSGD-Algorithmus verglichen. Als Datenbasis wurde die Filmdatenbank Movie Lens[4] verwendet. Es wurde Verglichen wie schnell der RMSE konvergiert und wie viele Iterationen dafür nötig sind. Diese Daten sind wichtig, da sobald dieser Punkt erreicht ist, keine weitere Optimierung möglich ist und der Algorithmus abgeschlossen ist bis neue Daten eintreffen. RMSE steht für Root-mean-squared error. Dieser Wert gibt an wie stark die Vorhersage von dem eigentlichen Wert abweicht.

Dabei kam es zu dem Ergebnis das der Streaming DSGD-Algorithmus wesentlich schneller und nach weniger Iterationen abgeschlossen ist. Er weißt jedoch eine höhere Abweichung auf, welche wohl auf das zufällige Eintreffen der Events zurück zu führen ist [AJT11]. Der Unterschied ist nur

[3] `https://storm.apache.org`
[4] `https://movielens.org`

gering, weshalb der Performancevorteil überwiegt und der streaming DSGD-Algorithmus, der bessere Ansatz für ein datenstrombasiertes Recommendersystem ist.

2.2.4 Fazit

In diesem Abschnitt wird ein Fazit gezogen und die beiden Ansätze für datenstrombasierte Recommendersysteme gegenübergestellt. Hierfür wird hauptsächlich betrachtet, wie gut die Ansätze mit Datenströmen umgehen können.

Die wichtigste Eigenschaft von datenstrombasierte Recommendersysteme ist ein guter Umgang mit dem Datenstrom, indem beispielsweise Fensteroperatoren Anwendung finden. Dadurch darf allerdings nicht die Qualität der Empfehlungen drastisch sinken und die Verarbeitung der Events sollte schnell durchgeführt werden.

StreamRec sticht durch die Benutzung der nativen Operatoren vom DSMS hervor. Damit ist keine weitere Implementierung neuer Operatoren nötig. Dadurch wird der ohne hin schon geringe Implementierungsaufwand von StreamRec weiter reduziert. Um den unendlichen Datenstrom zu bewältigen, werden nur Elemente ausgewertet die sich in einem bestimmten Zeitfenster befinden. So wird die Menge der Objekte dessen Ähnlichkeit berechnet werden muss, stark reduziert. Da durch einen Datenstrom die Daten unregelmäßig eintreffen, kann es vorkommen, dass in einigen Zeitperioden nur sehr wenige Elemente ausgewertet werden können, wo drunter die Qualität der Empfehlungen leidet. Deshalb ist zu empfehlen das Zeitfenster dynamisch zu erweitern oder ein elementbasiertes Fenster zu benutzen. Durch die Verwendung von nativen Operatoren wird auch eine dadurch gegebene höhere Performance gewährleistet und die Möglichkeit gegeben, die Berechnung der Ähnlichkeit zu skalieren. Es muss jedoch noch überprüft werden ob sich dieser Ansatz auch auf andere DSMS neben StreamInsight übertragen lässt.

Abschließend lässt sich zu StreamRec sagen, dass der Ansatz durch die Verwendung nativer Operatoren leicht zu implementieren ist und eine gute Verarbeitung des Datenstroms gewährleistet.

Der DSGD-Algorithmus dagegen ist DSMS-Unabhängig gestaltet und bewältiget den Datenstrom indem die Arbeitslast verteilt wird. Hierfür wird die Bewertungsmatrix in Blöcke unterteilt und auf die verschiedenen Worker verteilt. Hinzu kommt das durch die Verwendung von Apache Strom eine einfache Anwendung von MapReduce stattfinden kann. Dadurch ist auch eine einfache Skalierung möglich, indem beliebig viele Worker hinzugefügt werden. Die gesamte Berechnung den Vorhersagen findet auf Basis von einer Matrix-Faktorisierung statt, wofür es keinen nativen Operator gibt. Auch der benötigte *Matrix-Store* existiert nicht in dieser Form. Somit ist ein höherer Implementierungsaufwand als bei StreamRec nötig. Dieser wird aber durch die sehr gute Skalierbarkeit kompensiert. Ein weiterer Vorteil ist, dass der Algorithmus keine Fenster verwendet, sondern durch die inkrementelle Optimierung der Matrizen durch die Matrix-Faktorisierung immer alle Bewertungen berücksichtigt werden können. Im Gegensatz zu StreamRec fehlt auch eine Betrachtung der Recommend Events. Darüber hinaus kann es zu einer ungleichmäßigen Verteilung der Daten auf die einzelnen Blöcke kommen, wodurch einige Worker eine sehr hohe Auslastung haben und andere eine geringe. An dieser Stelle besteht noch Optimierungsbedarf.

So ist der DSGD-Algorithmus zwar aufwändiger zu implementieren, demgegenüber steht die sehr gute Skalierbarkeit durch Apache Strom.

Abschließend lässt sich sagen, dass beide Ansätze ihre Stärken haben und die geforderten Anforderungen erfüllen. So ist StreamRec leicht auf Basis eines DSMS umzusetzen und der DSGD-Algorithmus durch den Einsatz von MapReduce sehr gut skalierbar und performant. Um sich dennoch für einen Ansatz zu entscheiden muss ein konkreter Anwendungsfall vorliegen und evaluiert werden welcher Ansatz zum kollaborativen Filtern sich besser für den Anwendungsfall eignet.

2.3 Überblick über Anwendungsmöglichkeiten von Recommender-Systemen und deren domänenspezifischen Eigenschaften

Recommender Systems (Empfehlungssysteme) werden genutzt um Benutzern von Diensten, wie z. B. Onlineshops oder Videoplattformen, individuell Inhalte zu empfehlen, die vom Recommender System unter Verwendung verschiedener Technologien als relevant für den Nutzer ermittelt wurden. Für die Generation von Empfehlungen werden in der Regel kollaborative oder inhaltsbasierte Filter genutzt. Bei ersteren werden anhand von Bewertungen, die der Nutzer für Items getätigt hat, Empfehlungen von Items, die für den Nutzer relevant sind weil diese auch von dem Nutzer ähnlichen Nutzern (die z. B. ebenfalls viele der vom Nutzer bewerteten Produkte bewertet haben) positiv bewertet wurden, getätigt. Items können kontextabhängig z. B. Produkte, Videos oder auch Musikstücke sein. Inhaltsbasierte Filter ermitteln die zu empfehlenden Items hingegen auf Basis der Eigenschaften der Items, die der Nutzer sich angesehen bzw. gekauft oder angehört hat. Diese Eigenschaften können z. B. Produktkategorie, Preis oder im Falle von Musik das Musikgenre sein [WS12]. Dies bedeutet, dass dem Nutzer bspw. Produkte empfohlen werden, die den Produkten, die er bisher gekauft hat, ähnlich sind weil sie bspw. der gleichen Produktkategorie angehören und sich im Preis nur geringfügig unterscheiden. Recommender Systeme dienen dazu, dem Nutzer z. B. Produkte oder Videos zu empfehlen, die er noch nicht kennt, die für ihn aber dennoch wahrscheinlich interessant sind [LSY03, DLL$^+$10]. Für Betreiber von Recommender-Systemen bedeutet dies, dass sie bspw. in der Lage sind den, Nutzern neue (d. h. insbes. dem Nutzer unbekannte), für den individuellen Nutzer relevante, Produkte oder Videos zu zeigen, die ein hoher Anteil der Nutzer auch nachfragt [ZKG10], was sich z. B. in höheren Verkaufszahlen der empfohlenen Produkte oder längeren Aufenthalten des Nutzers auf der Videoplattform äußern kann.

2.3.1 Anwendung von Recommender Systemen

Recommender Systeme finden in verschiedenen Bereichen Anwendung. Im Nachfolgenden wird betrachtet, wie Onlinehändler, Videoplattformen und Musikempfehlungsdienste Recommender Systeme nutzen. Dies geschieht am Beispiel des Onlineshops Amazon, der zu Google gehörenden Videoplattform YouTube und einer wissenschaftlichen Studie zur automatischen Generation von Musikplaylisten. Diese Beispiele wurden gewählt weil es sich bei Amazon und YouTube (von Google) um sehr bekannte Anbieter handelt und sich andererseits die Vorgehensweise aufgrund der unterschiedlichen zu empfehlenden Objekte unterscheidet. Weiterhin wurde eine Studie aus dem Bereich der Musikempfehlung gewählt, da bei der automatischen Generierung von Wiedergabelisten weitere Faktoren, wie der Takt der Musik, berücksichtigt werden. Als Quellen werden insbes. für die Herleitung der Algorithmen offizielle Paper von Amazon [LSY03] und YouTube [DLL$^+$10] sowie von den Entwicklern des Musikplaylistendienstes [BJ14] genutzt. Für die Bewertung der Einflüsse der Recommender-Systeme auf das Nutzerverhalten wird im Falle Amazons eine Nutzerstudie [LR07] und im Falle YouTubes eine Studie.

2.3.1.1 Amazon

Die Recommender Systems des Onlinehändlers Amazon.com nutzen verschiedene Technologien für die Erstellung von Empfehlungen und die Personalisierung des Onlineshops. Dies bedeutet, dass sich das Erscheinungsbild des Onlineshops grundlegend in Abhängigkeit des Nutzers und dessen Interessen ändert [LSY03].

Amazon nutzt einen eigens entwickelten Recommender Algorithmus, das Item-to-Item Filtering für die Erstellung der individualisierten Empfehlungen. Dadurch umgehen sie die aus ihrer Sicht bestehenden Nachteile des Collaborative Filtering sowie von Clustering- und suchbasierten Algorithmen hinsichtlich Skalierbarkeit und Qualität der Empfehlungen [LSY03]. Im Folgenden werden das Collaborative Filtering sowie Clustering- und suchbasierte Methoden hinsichtlich ihrer Anwendbarkeit auf Ecommerce- Lösungen betrachtet und gezeigt, warum Amazon sich für die Entwicklung eines eigenen Algorithmus entschieden hat.

Im Rahmen der kollaborativen Filterung (Collaborative Filtering) werden die Kunden als n-dimensionale Vektoren betrachtet. n steht für die Anzahl an Produkten, die der jeweilige Kunde bewertet oder gekauft hat. Für Produkte, die der Kunde gekauft oder positiv bewertet hat, ist der Eintrag positiv, während die Einträge von Produkten, die der Kunde schlecht bewertet hat, negativ sind (genaueres wird hierzu von Amazon nicht genannt). Anschließend wird dieser Vektoreninhalt mit der inversen Frequenz (die Inverse der Anzahl der Kunden, die das Produkt gekauft oder bewertet haben) multipliziert um Produkte, die häufig bewertet oder gekauft werden, abzuwerten während eher unbekannte Produkte (Produkte die relativ selten bewertet oder verkauft werden) aufgewertet werden. Dies erfolgt um zu vermeiden, dass Kunden nur populäre Produkte empfohlen werden. Stattdessen sollen den Kunden auch neue und unbekannte Produkte empfohlen werden, soweit sie für den Kunden relevant sein könnten Als Grundlage für die Empfehlungen dienen jeweils wenige Kunden, die dem Kunden, für den Empfehlungen erstellt werden sollen, sehr ähnlich sind. Die Ähnlichkeit der Kunden wird mittels folgender Formel ermittelt [LSY03]:

$$similatity(\overrightarrow{A}, \overrightarrow{B}) = cos(\overrightarrow{A}, \overrightarrow{B}) = \frac{\overrightarrow{A} * \overrightarrow{B}}{\|\overrightarrow{A}\| * \|\overrightarrow{B}\|} \tag{2.5}$$

A und B stehen in dieser Formel für die Kunden, deren Ähnlichkeit ermittelt werden soll. Eine Möglichkeit daraus Empfehlungen abzuleiten ist, jedes Produkt anhand der Anzahl, wie viele Kunden es gekauft haben, zu bewerten [LSY03].

Neben dem Collaborative Filtering lassen sich Clustering-Modelle nutzen um die Kunden zu segmentieren und so ähnliche Kunden zu ermitteln. Dies erfolgt indem zuerst zufällig Kunden gewählt werden, die jeweils ein Segment darstellen. Anschließend wird für jeden Kunden die Ähnlichkeit zu jedem Segment berechnet und die Kunden dem Segment, dem sie am ähnlichsten sind, zugeteilt. Da für jeden Kunden nur die Ähnlichkeit zu den Segmenten berechnet werden muss, ist die Laufzeit von Clustering-Algorithmen geringer als beim Collaborative Filtering. Jedoch ist für sehr große Datensätze mit vielen Dimensionen ebenfalls eine Reduktion der Dimensionen und Samples notwendig. Nachteil von Cluster-Modellen ist, dass nicht der jeweils ähnlichste Kunde ermittelt wird, wie es beim Collaborative Filtering der Fall ist, sondern nur das ähnlichste Segment, dass eine geringere Ähnlichkeit aufweist als zum ähnlichsten Kunden. Dadurch ist die Qualität der Empfehlungen geringer als beim Collaborative Filtering [LSY03].

Such- oder Inhalts-basierte Methoden liefern Empfehlungen mit einer schlechten Qualität. Dies bedeutet, dass die individuellen Empfehlungen nur eine geringe Relevanz für den jeweiligen Nutzer aufweisen. Diese Methoden basieren auf der Suche nach verwandten Artikeln, also z. B. Bücher desselben Genres, desselben Autors oder mit gleichen Schlüsselworten. Die Laufzeit ist hierbei von der Anzahl der Artikel, die der Kunde gekauft oder bewertet hat abhängig, da alle betreffenden Artikel in die Suche einbezogen werden müssen. Empfehlungen auf Grundlage dieser Eigenschaften sind jedoch häufig zu allgemein oder zu sehr eingeschränkt (aktuelle Bestseller des Genres versus alle Bücher des Autors) und verfehlen so in den Augen von Amazon die Ziel von Recommender System, Kunden dabei zu helfen, neue, interessante und relevante Artikel zu finden und zu entdecken [LSY03].

Es gibt jedoch einige grundsätzliche Faktoren, die beim Einsatz von Recommender Systemen in Onlineshops bedacht werden müssen: Um treffende Empfehlungen erstellen zu können, ist es erforderlich, dass das Recommender System über eine ausreichende Datenbasis verfügt. Diese Datenbasis können vorherige Einkäufe und Bewertungen, die der Kunde für Produkte abgegeben hat sein. Dies bedingt auch, dass Onlineshops mit einer sehr hohen Kunden- und Produktanzahl treffendere Empfehlungen abgeben können. Für Neukunden, über die das System noch keine ausreichenden Informationen hat, ist die Empfehlung von Produkten dementsprechend schwerer als für langjährige Bestandskunden über die sehr viele Informationen, generiert aus Bewertungen und Einkäufen, vorliegen. Da im Grunde genommen jede Aktion des Kunden in einem Onlineshop für die Recommender Systeme verwertbare Informationen generiert, die die zukünftigen Empfehlungen beeinflussen können, ist es erforderlich, dass Empfehlungen in Echtzeit erstellt werden und trotzdem eine hohe Qualität aufweisen. Collaborative Filtering hat hingegen theoretisch eine Laufzeit von etwa O(m+n), wobei n für die Anzahl der Artikel pro Kunden und m für die Anzahl der Kunden steht. Vergrößern sich die Konstanten, z. B. die Anzahl der Kunden (m), erhöht sich dementsprechend die Laufzeit. Verringern ließe sich die Laufzeit indem z. B. Kunden mit wenigen Einkäufen ausgeblendet würden oder die Anzahl der betrachteten Artikel (n) verringert werden würde. Diese Verringerung der Datenbasis bewirkt jedoch qualitativ schlechtere Empfehlungen da im Fall einer Reduzierung von m die als ähnlich ermittelten Kunden eine geringere Ähnlichkeit hätten. Eine Reduzierung von n um die am stärksten oder geringsten nachgefragten Artikel führt hingegen dazu, dass diese Artikel nicht mehr empfohlen werden und Kunden, die nur solche Artikel gekauft haben würden keine Empfehlungen bekommen [LSY03].

Da für Amazon sowohl eine hohe Qualität der Empfehlungen (also eine hohe Relevanz für den Nutzer) als auch eine geringe Laufzeit hinsichtlich der enormen Datenmengen wichtig sind, haben sie das Item-to-Item Filtering entwickelt. Hierbei handelt es sich um eine Form des Collaborative Filtering welches auf die Produkte angewendet wird. Hierfür wird für jedes Produkt eine Tabelle mit ähnlichen Produkten, die oft zusammen gekauft werden, angelegt. Diese Berechnungen erfolgen anhand der Formel (A) und erfolgen offline, da die Laufzeit mit theoretisch O(n2m), praktisch eher O(nm), sehr hoch ist. Beim Abruf der Empfehlung müssen hingegen nur noch die Werte der Produkte, die dem vom Kunden gekauften Produkten ähnlich sind, aggregiert werden, um die Produkte die am ehesten mit den gekauften verwandt sind oder am beliebtesten sind, zu empfehlen. Dadurch, dass es sich um die Artikel mit der größten Ähnlichkeit handelt, ist die Empfehlungsqualität sehr hoch [LSY03].

Eine Studie, die 2007 in Finnland durchgeführt wurde, zeigt die Wichtigkeit der Empfehlungen bei der Produktsuche auf Amazon auf. Ein Ergebnis ist, dass die Empfehlungen von den Probanden vor allem genutzt wurden um eine Erstauswahl der Produkte zu treffen, während bei der weiteren Entscheidungsfindung Rezensionen eine wichtige Rolle spielen. Für drei der sechs Probanden waren zudem die in den Produktlisten angezeigten durchschnittlichen Bewertungen der Produkte relevant [LR07].

Die Rezensionen gaben den Probanden vor allem einen Überblick über das Produkt (in diesem Fall Bücher), wobei die Bewertung an sich eine untergeordnete Rolle spielt. Außerdem glichen die Probanden die in den Rezensionen geäußerten Ansprüche an das Produkt mit ihren eigenen Ansprüchen ab um so zu entscheiden, ob die Rezension für sie relevant ist [LR07].

Rezensionen sind somit die direkteste Empfehlung von Kunden zu Kunden, Technologien wie Tags oder Suchvorschläge wurden von den Nutzern hingegen nicht genutzt [LR07].

2.3.1.2 YouTube

YouTube ist eine Videoplattform mit mehr als einer Milliarde Nutzern und Milliarden täglichen Aufrufen und gehört zu Google. Auf YouTube ist jeder Nutzer in der Lage, eigenes Videomaterial hochzuladen, pro Minute sind das 300 Stunden. Gleichzeitig werden täglich mehrere hundert Millionen Stunden Videomaterial von den Nutzern betrachtet [You16].

Ähnlich wie Amazon hat auch Google den Anspruch, dass dem Nutzer nicht nur Inhalte desselben Autors empfohlen werden, sondern der Nutzer zur Entdeckung weiterer ähnlicher Videos angeregt wird. Es spielt dabei eine wichtige Rolle, ob der Nutzer eingeloggt ist oder nicht, da für eingeloggte Nutzer die Historien der geschauten Videos mit dem Profil verknüpft gespeichert werden [DLL+10].

Faktoren die bei der Entwicklung des Recommender-Algorithmus von YouTube bedacht werden mussten, sind, dass von Nutzern hochgeladene Videos oft über keine oder nur sehr geringe Metadaten verfügen und meistens kürzer als zehn Minuten sind, was dazu führt, dass die Interaktion der Nutzer mit diesem Video ebenfalls eher kurz ist (verglichen z. B. mit deutlich längeren Kinofilmen auf Amazon oder Plattformen wie Netflix). Außerdem sind viele Videos auf YouTube für die Nutzer nur kurzzeitig interessant, weswegen Empfehlungen neue und für den individuellen Nutzer relevante (für den Nutzer interessante) Inhalte enthalten sollen [DLL+10].

Das Recommender System nutzt für die Erstellung der Empfehlungen videobezogene (d. h. die Eigenschaften des Videos wie bspw. Videotitel, -länge oder -aufnahmedatum) und nutzerbezogene (d. h. Verläufe der Interaktionen des Nutzers mit der Videoplattform) Daten. Bei den videobezogenen Daten ist zu beachten, dass sie inkorrekt, bspw. veraltet oder falsch, sein können. Die nutzerbezogenen Daten gliedern sich wiederum in zwei Gruppen: Daten aus expliziten Aktivitäten, wie z. B. die Bewertung von Videos, das „liken“ von Videos oder das abonnieren von Kanälen. Es werden jedoch auch implizit Daten über die Nutzung gesammelt. Dies ist z. B., welche Videos der Nutzer angefangen hat zu schauen und wie viel er von diesem Video angesehen hat. Diese impliziten Daten werden asynchron gesammelt und nicht direkt an YouTube gesendet sondern im Browser zwischengespeichert. So führt ein Schließen des Browsers bzw. Verlassen der Seite, bevor diese Daten an YouTube übertragen worden sind, zum Verlust dieser Daten [DLL+10].

Der erste Schritt des auf dem Collaborative Filtering basierenden Recommender Algorithmus ist die Ermittlung von verwandten Videos. Bei YouTube ist dies so umgesetzt, dass ermittelt wird, wie oft Videos gemeinsam, das heißt innerhalb einer Sitzung, angesehen werden. Daraus wird für jedes Video ein Set gebildet, mit den n Videos, die am häufigsten gemeinsam mit dem Video angesehen werden und einen bestimmten (nicht näher definierten) Mindestwert erreichen. Problematisch ist dies für Videos, die entweder generell selten angeschaut wurden oder die selten gemeinsam mit anderen Videos angesehen wurden [DLL+10].

Die hier berechneten Daten werden gespeichert und mit dem Nutzungsverhalten des Nutzers abgeglichen um eine Personalisierung der Empfehlungen zu erreichen. Um zu erreichen, dass dem Nutzer nicht nur Videos, die seinen Interessen entsprechen, empfohlen werden, sondern auch dem Nutzer unbekannte Videos, wird die Auswahl um Videos, weniger den Interessen des Nutzers entsprechen, erweitert [DLL+10]. Letztendlich ist jeder Empfehlungskandidat mit mindestens einem Video aus dem Seed (Pool der Videos, die vom Nutzer geschaut wurden oder mit denen er interagiert hat) verknüpft. Dies dient einerseits zur Bewertung der Empfehlungen und andererseits zum User-Feedback um im Web-Interface anzeigen zu können, auf Grundlage welcher Videos dem Nutzer jedes einzelne Video empfohlen wurde [DLL+10].

Für die Bewertung der Empfehlungskandidaten werden qualitative Aspekte des Videos, wie die Anzahl, wie häufig es gesehen wurde, wie häufig es favorisiert und kommentiert wurde, sowie nutzerspezifische Daten um Kandidaten, die die Interessen des Nutzers treffen, hervorzuheben. Außerdem werden die Videos hinsichtlich ihrer Diversität in Bezug auf den Nutzer, bewertet. Das Ergebnis ist eine Liste mit Empfehlungen, welche jedoch nicht komplett ausgegeben wird. Die Auswahl der Empfehlungen aus den Kandidaten erfolgt nicht starr anhand der ermittelten Relevanz, sondern auch anhand der Diversität, so dass ein Gleichgewicht zwischen relevanten und nicht mit den Themen, mit denen der Nutzer sich sonst hauptsächlich beschäftigt, verwandten Videos. Dies erfolgt, da die Nutzer in der Regel über mehrere Interessen verfügen [DLL+10].

Das Ergebnis der Bewertung der Empfehlungskandidaten sind die Empfehlungen. Dem Nutzer werden jeweils vier bis sechzig Empfehlungen angezeigt, berechnet hat das System jedoch weitaus mehr. Dadurch wird erreicht, dass die dem Nutzer angezeigten Empfehlungen nicht bei jedem Seitenaufruf erneut berechnet werden müssen. Stattdessen werden die Empfehlungen mehrmals täglich neuberechnet. Da somit bei jedem Seitenaufruf die Empfehlungen nur noch abgerufen werden müssen, wird eine geringe Latenz bei der Ausgabe der Empfehlungen erreicht. Für die Neuberechnung der Empfehlungen werden die Logs von YouTube auf einzelne Nutzer heruntergebrochen und die entsprechenden Signale (z. B. die Wiedergabe eines Videos) extrahiert. Täglich sind dies mehrere Terrabyte Signaldaten, die vom Recommender-System für die Empfehlungserstellung genutzt werden [DLL+10].

Für YouTube und seine Nutzer sind Recommender Systeme sehr wichtig. In einem 2010 veröffentlichten Dokument berichteten Google-Mitarbeiter darüber, dass 60 % der Klicks auf Videos auf der Homepage auf die empfohlenen Videos entfallen [DLL+10]. In einer anderen Studie wurde festgestellt, dass die Vorschlagfunktion zu den wichtigsten Quellen bei der Entscheidung, welches Video der Nutzer schaut, zählt. Auch wurde ein Zusammenhang zwischen Häufigkeiten, wie oft die Videos angesehen wurden (im Folgendem View-Anzahl) von Videos in Zusammenhang mit dem Recommender System gefunden wurde (siehe Abbildung 2.6) [ZKG10].

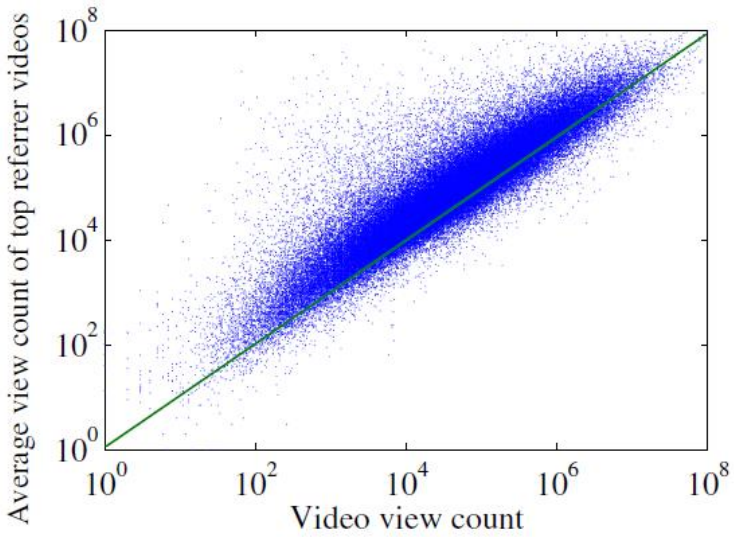


Abbildung 2.6: Viewzahlen des referenzierenden Videos im Vergleich zum empfohlenen Video [ZKG10].

o sind die View-Anzahlen von Videos, die neben häufig gesehenen Videos empfohlen werden, deutlich höher als von Videos, auf die nur von anderen Videos mit niedrigen View-Zahlen verwiesen wird. So besteht eine Wahrscheinlichkeit von 41,6 %, dass der Nutzer eins der empfohlenen Videos anschaut Weiterhin wurde festgestellt, dass in den Empfehlungen nicht nur sehr populäre Videos zu finden sind, sondern auch vom Nutzer noch unentdeckte Videos, die seinen Interessen entsprechen. [ZKG10] Dies bestätigt die Zielsetzungen der Google-Mitarbeiter bei der Entwicklung des Recommender Algorithmus [DLL+10].

Insgesamt kommt [ZKG10] zu dem Ergebnis, dass 30 % aller Videoaufrufe auf YouTube durch die Recommender Systeme zustande kommen und das die Empfehlungen für die Mehrheit der Videos die wichtigste Quelle ist, um Nutzern Videos, die ihren Interessen entsprechen und ihnen neu sind, zu zeigen [ZKG10].

2.3.1.3 Online Musikplattformen

Plattformen für Musikstreaming, wie bspw. Spotify oder Deezer, haben in der Vergangenheit stark an Popularität gewonnen und ihnen wird weiterhin ein starkes Wachstumspotenzial zugeschrieben [BH13].

Wie Videos enthalten auch Musikstücke verschiedene Metainformationen, wie bspw. Erscheinungsjahr, Titel, Genre oder Interpret, die für die automatisierte Erstellung von Playlisten durch Recommender Systeme genutzt werden können. Neben den Metadaten können aber auch Nutzungsdaten sowie Daten aus sozialen Medien genutzt werden sowie das Audiosignal selbst. Das Audiosignal kann auf Faktoren wie Lautstärke oder Rhythmus analysiert werden, diese Analyse ist jedoch rechenintensiv [BJ14].

Bestandteil der Daten aus sozialen Medien sind auch Bewertungen, die die Nutzer für die jeweiligen Musikstücke abgegeben haben. Außerdem können über die Profile der Interpreten in sozialen

Netzwerken Interpreten, die sich dort mit untereinander vernetzt haben, ermittelt werden [BJ14]. Die Nutzungsinformationen bestehen aus Daten, die die Popularität der Songs betreffen (bspw. eine interne Chartliste) und Daten zum Nutzungsverhalten. Dies sind insbesondere, ähnlich wie im Fall YouTube, Protokolle aus denen hervorgeht welche Songs der Nutzer gehört hat und auch welche er übersprungen hat. Jedoch kann es auch sein, dass ein Nutzer einen Song nur nicht übersprungen hat, weil er gerade nicht zugehört hat. Um Beziehungen zwischen Songs aufzudecken, die nicht aus den Metadaten oder den Audiosignalen hervorgehen, können auch die manuell erstellten Playlisten der Nutzer einbezogen werden [BJ14].

Um automatisch Playlisten zu generieren, ist es notwendig, dass der Zweck der zu erstellenden Playlist spezifiziert wird. Hierbei kann es sich um explizite Angaben, wie z. B. die Angabe von Referenztiteln. Weitere Eingabeparameter können z. B. Keywords, wie bei last.fm, und Bewertungen sein. Weitere Gruppen von Eingabedaten sind nutzerbezogene Daten wie von ihm manuell erstellte Playlisten und Informationen über den Wiedergabekontext. Informationen über den Wiedergabekontext können z. B. mit Smartphones über die verbauten Sensoren ermittelt werden. Dies sind z. B. der Ort (GPS- Sensor) oder die Umgebung (Kamera und Mikrofone) [BJ14].

Für die Erstellung der Playlisten werden Recommender-Algorithmen genutzt. In der vorliegenden Studie sind dies Algorithmen, die die Ähnlichkeit der Titel untersuchen und daraus kompatible Empfehlungen ableiten sowie Algorithmen die auf Collaborative Filtering, Frequent Pattern Mining oder statistischen Modellen basieren. Darüber hinaus können diese Ansätze kombiniert werden [BJ14].

Bei der Evaluation der verschiedenen Ansätze (Algorithmen) zeigt sich, dass Algorithmen, die auf der Ähnlichkeitsfindung von Titeln basieren, den besten Ansatz bieten, wenn das Ziel ist möglichst homogene Playlisten zu erstellen. Um möglichst homogene Playlisten zu generieren ist es notwendig auch die Reihenfolge der Songs zu berücksichtigen [BJ14].

2.3.2 Probleme und Herausforderungen

Bei der Planung, Entwicklung und Implementierung von Recommender Systemen sind verschiedene Aspekte zu berücksichtigen. So spielt für viele Anwendungen die Laufzeit eine wichtige Rolle weil sie über sehr große und umfassende Datenbestände verfügen. Dieser Aspekt muss bei der Entwicklung des Recommender-Algorithmus berücksichtigt werden, z. B. indem geprüft wird in wieweit sich die Laufzeit durch eine Reduzierung der Eingabedaten reduzieren lässt. In vielen Fällen bringt eine Reduktion der Eingabedaten aber den negativen Effekt mit sich, dass sich die Qualität der Empfehlungen verringert. Wie am Beispiel Amazon mit seinen auf dem Collaborative Filtering basierenden Item- to-Item Filtering zu sehen, können Methoden und Algorithmen speziell für ihren jeweiligen Anwendungszweck weiterentwickelt werden. Dadurch kann bspw. der Nachteil der hohen Laufzeit von Collaborative Filtering Algorithmen vermieden werden verbunden mit dem Vorteil der hohen Qualität der Empfehlungen (d. h. hohe Relevanz der Empfehlungen) [LSY03].

Außerdem muss bedacht werden, dass grundsätzlich Informationen über den Nutzer und seine bisher getätigten Aktionen vorhanden sein müssen, um treffsichere Empfehlungen generieren zu können [LSY03, Da10]. Wie am Fall YouTube zu sehen ist, werden sämtliche Interaktionen des Nutzers mit der Videoplattform protokolliert und durch das Recommender-System ausgewertet [DLL$^+$10]. Für die Nutzer solcher Plattformen bedeutet dies, dass sie davon ausgehen müssen, dass jede ihrer Aktion nachvollziehbar und ihnen zuordenbar ist.

Im Fall von kontextsensitiven Recommender-Systemen muss außerdem der aktuelle Nutzungskontext erfasst werden. Dies kann z. B. erfolgen indem die in einem Smartphone verbauten Sensoren verwendet werden, wie es im Falle der App InCarMusic, die während der Autofahrt die Musik angelehnt an die aktuelle Fahrsituation auswählen soll [BKL+11]. In anderen Fällen, bspw. bei der Empfehlung von Smartphone-Apps in Abhängigkeit von der aktuellen Nutzungssituation (bspw. im privaten oder beruflichen Umfeld oder während einer Autofahrt) der Nutzer seinen aktuellen Kontext eingeben muss [BBK10].

2.3.3 Zusammenfassung

Für verschiedene Dienste, wie bspw. Onlineshops oder Musik- und Videoplattformen sind Recommender-Systeme sehr wichtig um den Nutzer weitere Inhalte, die für ihn potenziell relevant sein könnten zu zeigen. Für die Nutzer der Dienste, die Recommender-Systeme nutzen, ergibt sich der Vorteil, dass ihnen das Finden neuer Inhalte, die für sie potenziell interessant sind, erleichtert wird. Wie an den verschiedenen Anwendungsfällen zu sehen ist, ist es wichtig die Anforderungen an das Recommender-System und die Systemumgebung zu definieren um das Recommender-System individuell für seinen Anwendungszweck zu entwickeln, so dass es die an ihm gestellten Anforderungen erfüllt. In Umgebungen mit hohen Datenmengen ist, wie Amazon und YouTube zeigen, insbesondere die Laufzeit der Recommender-Systeme wichtig.

2.4 Recommender-System-Methoden

Recommender systems oder recommendation systems sind Systeme, die einem Nutzer aus Basis von Informationen die das System über ihn gesammelt hat, Empfehlungen geben. Solche Systeme werden in der Regel im Internet eingesetzt um dort zum Beispiel in einem Online-Shop einem Kunden Vorschläge in Form von Produkten zu machen, die dem Kunden mit einer hohen Wahrscheinlichkeit gefallen, sodass er diese dann aufgrund dieser Aufforderung bzw. des Vorschlags kauft. Ein weiterer Anwendungsfall wäre zum Beispiel eine Nachrichtenseite, bei der die Interessen eines Lesers gesammelt werden, sodass ihm nur noch Artikel vorgeschlagen werden, die ihn interessieren.

Um einem Anwender einen geeigneten Vorschlag zu machen gibt es im Wesentlichen zwei unterschiedliche Herangehensweisen.

- *Content-Based Systeme* sammeln Eigenschaften verschiedener Gegenstände oder auch Items wie Produkte oder Artikel. Andere Items können dann aufgrund ihrer Eigenschaften untereinander verglichen werden. Das System empfiehlt einem Nutzer dann Gegenstände die ihm laut den Daten, die das System gesammelt hat gefallen könnten. (Gefallen einem Nutzer viele Sportartikel über einen besonderen Verein werden ihm in Zukunft vermehrt Artikel über diesen Verein empfohlen).

- *Collaborative-Filtering Systeme* vergleichen die Anwender untereinander und geben Empfehlungen basierend auf dem, was anderen Nutzern die ähnliche Präferenzen wie man selbst hat gefallen, um so Rückschlüsse auf die eigenen Wünsche schließen zu können. (Trifft man selbst positive Bewertungen über Produkte vergleicht das System diese Bewertungen mit anderen Nutzern. Haben andere Nutzer zusätzlich weitere Produkte positiv bewertet kann davon ausgegangen werden das es mir selber auch gefallen könnte) [RU11].

	Harry Potter 1	Harry Potter 1	Star Wars 1	Star Wars 2
A	1	1	4	4
B			5	4
C	1	?	5	
D	1	2	5	4

Tabelle 2.1: Utility Matrix eines Empfehlungssystems für Filme.

Um Empfehlungen erstellen zu können müssen vorher Daten gesammelt werden. Dies kann zum Beispiel erfolgen indem der Anwender Items bewertet, nachdem er sie zum Beispiel gekauft oder Artikel gelesen hat. Auf diese Weise kann das System Profile über die Benutzer erstellen. Außerdem müssen für das contentbasierte System noch Profile über die Items erstellt werden.

Bewertungen werden in der Regel in einer so genannten Utility Matrix oder auch Rating Matrix dargestellt. Die Utility Matrix ist eine Tabelle in der alle Bewertungen der Benutzer bezüglich der vorhandenen Items gespeichert sind. Mit ihrer Hilfe können zum Beispiel beim Collaborative-Filtering Methoden angewendet werden um Rückschlüsse von einem Nutzer auf einen anderen zu ziehen. Tabelle 2.1 zeigt eine vereinfachte Utility Matrix eines Empfehlungssystems für Filme. Diese Tabelle zeigt das Problem natürlich nur stark vereinfacht dar, da in der realen Welt oft tausende von Filmen angeboten werden, die alle bewertet werden können und dazu millionen von Nutzern vorhanden sind, die diese Bewertungen tätigen. Klar wird, dass Recommender Systeme die Nützlichkeit eines Items in Form einer Bewertung eines Nutzers darstellen (Benutzer A bewertet Harry Potter einem mit einem Punkt/Stern von fünf möglichen). Ist ein Feld leer so bedeutet das, dass keine Auskunft darüber vorliegt ob dem Benutzer das Item gefällt oder nicht [AT05]. Ein Empfehlungssystem würde nun versuchen die leeren Felder zu berechnen um zu prüfen, welcher Film welchem Nutzer empfohlen werden sollte.

2.4.1 Content-Based Systeme

Bei der Content-Based Methode geht das System von den Items aus, vergleicht diese auf Gemeinsamkeiten und versucht dann einem Nutzer ein Item zu empfehlen, das ähnlich ist wie das, welches er bereits positiv bewertet hat. Dabei wird für die Berechnung der Empfehlung ein sogenanntes Item-Profil über das Item und ein User-Profil für den aktiven Benutzer, das heißt der Benutzer für den die Empfehlung errechnet wird, erstellt. Diese Methode wird vermehrt eingesetzt um textuelle Quellen, also zum Beispiel Artikel einer Nachrichtenplatform zu empfehlen.

Es wird also die Nützlichkeit $u(c, s)$ eines Items s für den Benutzer c errechnet, basierend auf der Bewertung $u(c, s_i)$ von Benutzer c des Items $s_i \in S$ das ähnlich ist wie s [AT05].

2.4.1.1 Item-Profile

In einem Contend-Based System wird für jeden Gegenstand (Item) ein Profil angelegt. Dieses Profil beinhaltet alle relevanten Daten die das Item beschreiben. Bei einem Film wären solche Daten zum Beispiel die Schauspieler die in diesem Film mitspielen, der Regisseur der den Film gedreht hat, das Genre des Films oder das Jahr in dem der Film gedreht wurde. All diese Daten sind wichtig für eine etwaige Empfehlung da Unterschiedliche Nutzer verschiedene Vorlieben für Filme haben. Einige mögen Filme mit gewissen Schauspielern gerne, andere bevorzugen ein gewissen Regisseur oder ein

bestimmtes Genre [RU11]. Generell kann der Inhalt eines Item-Profils als $Content(s)$ angesehen werden.

Um eine Art Item-Profil für ein Dokument zu erstellen wird häufig die term frequency/inverse document frequency (TF-IDF) Methode benutzt. Mit dieser Methode ist es möglich Schlagwörter oder keywords von Dokumenten zu bestimmen, die den Inhalt eines Artikels wiedergeben sollen. Sie ist folgendermaßen definiert:

Es gibt eine Anzahl an N Dokumenten die für einen Nutzer vorgeschlagen werden können und es kommt keyword k_j in einem Dokument n_i vor. Dann ist $f_{i,j}$ die Häufigkeit mit der keyword k_i in Dokument d_j vorkommt. Somit ist die normalisierte Vorkommenshäufigkeit $TF_{i,j}$ eines keywords k_i in einem Dokument d_j definiert als [AT05]:

$$TF_{i,j} = \frac{f_{i,j}}{max_z f_{z,j}} \tag{2.6}$$

Wobei das Maximum aus allen Häufigkeiten $f_{z,j}$ eines keywords k_z im Dokument d_j errechnet wird. Da aber eine hohe Häufigkeit an keywords nicht charakteristisch für ein relevantes oder nicht relevantes Dokument ist wird die normalisierte Vorkommenshäufigkeit oft mit der inversen Dokumentenhäufigkeit (IDF) kombiniert. Die inverse Dokmentenhäufigkeit eines keywords k_j ist definert als [AT05]:

$$IDF_i = log\frac{N}{n_i} \tag{2.7}$$

Somit ist das TF-IDF Maß eines keywords k_i in einem Dokument d_j definiert als [AT05]:

$$w_{i,j} = TF_{i,j} \times IDF_i \tag{2.8}$$

Und der Inhalt eines Dokuments d j ist definiert als [AT05]:

$$Content(d_j) = (w_{1j}, \ldots, w_{kj}) \tag{2.9}$$

2.4.1.2 User-Profile

Genauso wie für die einzelnen Gegenstände Item-Profile angelegt werden muss auch für den jeweiligen Benutzer ein User-Profil angelegt werden. Es wird als $ContentBasedProfile(c)$ definiert. Das User-Profil enthält alle Informationen über den jeweiligen Benutzer. Die Informationen werden durch Analysen anhand des Verhaltens des Benutzers gewonnen, also welche Artikel er als relevant oder irrelevant bezeichnet, oder zum Beispiel wie viele Sterne er bestimmten Filmen gibt. Für das Empfehlen eines Artikels kann man das $ContentBasedProfile(c)$ wieder als Vektor von Gewichtungen betrachten $(w_{c1}, \ldots, w_{ck})$, wobei jede Gewichtung w_{ci} die Relevanz eines keywords k_i für den Benutzer c wiedergibt [AT05].

2.4.1.3 Empfehlungen geben

Hat das System für die Items ein Item-Profil und für die Benutzer ein User-Profil erstellt, so ist es möglich ausgehend von diesen beiden Profilen einen Wert zu errechnen der angibt, wie wahrscheinlich

es ist das das Item und der Benutzer zusammenpassen. Die Funktion dies zu berechnen ist in einem Content-Based System $u(c, s)$ generell definiert als [AT05]:

$$u(c, s) = score(ContentBasedProfile(c), Content(s)) \tag{2.10}$$

Da sowohl das Item-Profile als auch das User-Profile als TF-IDF Vektoren und betrachtet werden können, lässt sich mithilfe der Cosinus-Korrelation der Cosinus des Winkels zwischen den beiden Vektoren berechnen. Dieser ist definiert als [AT05, Str11]:

$$u(c, s) = cos(\overrightarrow{w_c}, \overrightarrow{w_s}) = \frac{\overrightarrow{w_c} \times \overrightarrow{w_s}}{||\overrightarrow{w_c}||^2 \times ||\overrightarrow{w_s}||^2} = \frac{\sum_{i=1}^{K} w_{i,c} w_{i,s}}{\sqrt{\sum_{i=1}^{K} w_{i,c}^2} \sqrt{\sum_{i=1}^{K} w_{i,s}^2}} \tag{2.11}$$

Mit K als Anzahl aller keywords im System.

Wenn Nutzer c zum Beispiel viele Online Artikel über Fußball liest, dann wird ein Content-Basiertes System in der Lage sein diesem Nutzer weitere Artikel über Fußball zu empfehlen. Dies liegt zu Grunde, da in einem Artikel mehr keywords wie Tor, Aufstellung, Mannschaft usw. drin vorkommen als in anderen Artikeln. Daher repräsentiert das $ContentBasedProfile(c)$, definiert durch den Vektor $\overrightarrow{w_c}$ solche keywords k_i mit höherem Gewicht w_{ic}. Benutzt man nun also die Cosinus-Korrelation kommt für $u(c, s)$ höhere Nützlichkeit bei Artikeln s bei denen das Gewicht für diese keywords hoch ist als bei den Artikeln s bei denen das Gewicht niedrig ist [AT05].

2.4.2 Memory-Based Collaborative-Filtering

Im Gegensatz zu einem Content-Based System, bei dem Items aufgrund ihrer Ähnlichkeit verglichen und empfohlen werden konzentriert sich das Collaborative- Filtering System darauf, Ähnlichkeiten zwischen den Benutzern zu erfassen. Aufgrund dieser Ähnlichkeit sollen dann Schlüsse gezogen werden die zu einer Empfehlung eines Items führen. Anstatt Vektoren für die Benutzer zu erstellen werden sie hier als Spalte eines Items in der Utility Matrix betrachtet [RU11].

Angenommen es gibt zwei Benutzer A und B. Beide haben den Film “Der Herr der Ringe – die Gefährten"mit 5 Sternen bewertet. Darüber hinaus hat Benutzer B noch die anderen Teile der Trilogie mit 5 Sternen bewertet. Das System würde nun basierend auf den Werten von Benutzer A und B, Benutzer A, die anderen Herr der Ringe Teile empfehlen da es annimmt die beiden Benutzer seien ähnlich. Dies ist natürlich ein sehr vereinfachtes Beispiel und dient nur der Anschauung.

Bei Collaborative-Filtering Systemen gibt es nochmal zwei verschiedene Ansätze; den Memory-basierten und den Model-basierten. Bei den Memory-Basierten Ansätzen werden in der Regel die gesamte Benutzer-Item Datenbank für eine Empfehlung genutzt, da oft Ähnlichkeiten zwischen Nutzern oder Items errechnet werden und somit alle Nutzer und Items betrachtet werden müssen.

Eine verbreiter Algorithmus im memorybasierten Collaborative-Filtering ist das Nachbarschaftsbasierte Collaborative Filtering. Hier wird zuerst die Ähnlichkeit oder Gewichtung zweier Benutzer oder Items errechnet. Anhand dieser Ähnlichkeit wird dann eine Prognose für den aktiven Benutzer (der Benutzer dem eine Empfehlung gegeben werden soll) basierend auf den Bewertungen des Items oder der Bewertungen der Benutzer errechnet. Dabei gibt es noch die Möglichkeit eine top-N Empfehlung

zu geben was bedeutet, dass die k ähnlichsten Benutzer bzw. Items basierend auf den Ähnlichkeiten berechnet werden (nächsten Nachbarn) [SK09].

2.4.2.1 Similarity Computation (Ähnlichkeitsberechnung)

Bei der Ähnlichkeitsberechnung zweier Items oder Benutzer wird die Ähnlichkeit zwischen zwei Items i und j bzw. zwischen den Benutzern u und v bestimmt. Bei einem gegenstandbasiertem (item-based) Vorgehen wird zuerst geschaut welche Benutzer beide dieser Items bewertet haben, um so mit einer Ähnlichkeitsberechnung die Ähnlichkeit dieser Items $w_{i,j}$ zu ermitteln. Bei einem benutzerbasiertem Ansatz wird die Ähnlichkeit der Benutzer $w_{u,v}$ der Nutzer ermittelt, die das selbe Item bewertet haben.

Es gibt verschiedene Methoden die Ähnlichkeit dieser zu bestimmen. Eine Methode ist die Correlation-Based Similarity. Bei dieser Methode wird die Ähnlichkeit zweier Benutzer $w_{u,v}$ oder zweier Items $w_{i,j}$ mit beispielsweise der Pearson-Correlation berechnet [SK09]. Der Pearson-Korrelationskoeffizient beschreib den linearen Zusammenhang zwischen zwei Datenmengen. Er kann dabei Werte zwischen -1 und +1 annehmen wobei bei -1 und +1 ein hoher linearer Zusammenhang besteht. Je weiter sich der Wert Richtung 0 annähert, desto geringer wird der Zusammenhang. Bei einem Wert von 0 gibt es keinen linearen Zusammenhang [Str11]. Bei einem benutzerbasierten Ansatz wird der Korrelationskoeffizient der Benutzer u und i wie folgt gebildet [SK09]:

$$w_{u,i} = \frac{\sum_{i \in I}(r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in I}(r_{u,i} - \overline{r}_u)^2}\sqrt{\sum_{i \in I}(r_{v,i} - \overline{r}_v)^2}}, \tag{2.12}$$

wobei die $i \in I$ Summe über alle Items gebildet wird, die Sowohl Benutzer u als auch Benutzer v bewertet haben und $\overline{r}_u$ die Durchschnittsbewertung des Benutzers u für dieselben Items angibt.

Bei der gegenstandbasierenden Berechnung des Pearson-Korrelationskoeffizienten wird von einer Menge an Benutzern $u \in U$ ausgegangen, die dieselben Items i und j bewertet haben. Daraus ergibt sich nach [SK09]:

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \overline{r}_i)(r_{u,i} - \overline{r}_j)}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r}_j)^2}\sqrt{\sum_{u \in U}(r_{u,j} - \overline{r}_j)^2}}, \tag{2.13}$$

mit r_u, i als Bewertung des Benutzers u für das Item i. Weiterhin ist Durchschnittsbewertung von i aller betrachteten Benutzer.

2.4.2.2 Prediction and Recommendation Empfehlungsberechnung)

Das Treffen von Vorhersagen und das Geben von Empfehlungen ist der wichtigste Schritt in einem Collaborative Filtering System. Nachdem die nächsten Nachbarn eines aktiven Benutzers gefunden sind, das heißt die Benutzer die ihm am ähnlichsten sind, wird mit der gewichteten Gesamtbewertung der Benutzer eine Vorhersage für den aktiven Benutzer erstellt [SK09].

Eine Methode, eine Vorhersage für einen aktiven Benutzer zu treffen, ist mit der *Weighted Sum of Others' Rating*, also der gewichteten Gesamtbewertung aller anderen Benutzer der Nachbarschaft, also alle anderen Ähnlichen Benutzer. Um einem aktiven Benutzer a ein Item i zu empfehlen wird die gewichtete Durchschnittsbewertung aller Benutzer, die dieses Item bewertet haben hinzugezogen. Damit ergibt sich nach [SK09] folgende Gleichung:

$$P_{a,i} = \overline{r}_a + \frac{\sum_{u \in U}(r_{u,i} - \overline{r}_u) * w_{a,u}}{\sum_{u \in U} |w_{a,u}|} \tag{2.14}$$

Hierbei sind $\overline{r}_a$ und $\overline{r}_u$ die Durchschnittsbewertungen der Nutzer a und u aller anderen Items, also alle außer i. Mit wird die Ähnlichkeit zwischen Nutzer a und Nutzer u angegeben (siehe Formel 2.12). Die Summe wird über alle Nutzer $u \in U$ gebildet, die das Item bewertet haben. Hierbei ist nochmal zu erwähnen, dass nur die Nutzer aus der Nachbarschaft zum aktiven Nutzer gemeint sind.

Während die weighted sum of others‘ rating ein benutzerbasierender Ansatz ist kann man mit dem *Simple Weighted Average*, also der einfachen Mittelgewichtung noch eine Gegenstands-basierte Vorhersage treffen. Somit lässt sich die Bewertung $P_{u,i}$ eines Nutzers u für ein Item i nach [SK09] mit

$$P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} |w_{i,n}|} \tag{2.15}$$

vorhersagen. Hierbei wird die Summe über alle bewerteten Items $n \in N$ des Benutzers u gebildet. Mit $w_{i,n}$ ist die Ähnlichkeit des Items i und n zueinander gemeint (vergleiche Formel 2.13). $r_{u,n}$ ist dann die Bewertung des Benutzers u für das Item i.

2.4.2.3 Top-N Recommendations

Top-N recommendations sind Empfehlungen bei denen eine bestimmte Menge von Items, die mit dem höchsten Empfehlungswert, einem Benutzer empfohlen werden. Bekannt sind solche Empfehlungen aus Amazon, Netflix oder Youtube, bei denen jeweils eine Liste von Artikeln, Filmen oder Videos angezeigt wird, die einen Nutzer interessieren könnten. Für das Geben von Top-N recommendations gibt es wieder einen benutzerbasierten und einen gegenstandsbasierten Ansatz.

Bei dem Benutzerbasierten Ansatz werden zuerst k möglichst Ähnliche Benutzer gesucht. Dies kann zum Beispiel mit dem Pearson-Korrelationskoeffizienten berechnet werden. Nachdem alle gefunden sind werden ihre entsprechenden Reihen in der Utility Matrix vereinigt um eine Menge an Items C herauszubekommen, die diese Gruppe an Benutzern erworben oder gesehen hat. Aus der Menge C können dann mit benutzerbasierenden CF Methoden die top-N Artikel, Filme oder Videos empfohlen werden, die der aktive Benutzer noch nicht gekauft oder gesehen hat. Der benutzerbasierte Ansatz der top-N recommendation hat allerdings eine Grenze der Skalierbarkeit und der Echtzeitperformance, da das Suchen von k Ähnlichen Nutzern inklusive ihrer erworbenen Artikel bei großen Datenmengen sehr aufwändig werden kann [SK09].

Der gegenstandsbasierte top-N Algorithmus wurde daraufhin entwickelt um dem Problem der Skalierbarkeit des benutzerbasierten Algorithmus entgegenzuwirken. Als erstes werden die k ähnlichsten Items für jedes Item berechnet, den der Benutzer erworben oder gesehen hat. Dann wird eine Menge C an Kandidaten gebildet, die empfohlen werden können indem die Vereinigung der k ähnlichsten Items von der Menge U, die der Benutzer schon erworben oder gesehen hat abgezogen wird. Dann werden die Ähnlichkeiten zwischen den Gegenständen der Menge C und U berechnet. Sortiert man nun die Ergebnisse in absteigender Reihenfolge erhält man eine Liste mit den top-N Artikeln, Filmen oder Videos für einen Benutzer [SK09].

2.4.2.4 Klassifikation

Eine weitere Möglichkeit im memorybasiertem Collaborative-Filtering Annahmen zu treffen ist mit dem Simple Bayesian CF Algorithm. Hier wird ausgehend vom naive Bayes versucht eine Vorhersage über eine Bewertung zu treffen. Hierbei kann für ein leeres Feld in der Rating Matrix eine Annahme getroffen werden, wie ein Benutzer dieses Feld ausfüllen würde. Berechnet wird das Feld mit der Wahrscheinlichkeitsrechnung:

$$class = arg\ max_{j\ \in\ classSet} p\ (class_j) \Pi_o P(X_o = x_o | class_j) \tag{2.16}$$

Der Index o steht dabei für angenommene Werte, für den Fall das Werte, die für die Berechnung benötigt werden fehlen. In diesem Fall sollte man Werte aus Beobachtungen annehmen. Berechnet wird dann die Wahrscheinlichkeit mit der eine Klasse zutrifft. Klasse meint in diesem Sinne bei einer Bewertungsform von eins bis fünf Sternen, dass es fünf verschiedene Klassen gibt. Damit keine bedingte Wahrscheinlichkeit von null auftreffen kann wird in [SK09] zusätzlich ein Laplace Estimator benutzt.

$$P(X_i = x_i | Y = y) = \frac{\#(X_i = x_i, Y = y) + 1}{\#(Y = y) + |X_i|} \tag{2.17}$$

2.4.3 Model-Based Collaborative Filtering

Memory basierte Collaborative-Filtering Systeme sind in der Regel nicht immer schnell und haben ein Problem hinsichtlich der Skalierbarkeit, besonders in echten Systemen in denen Echtzeitempfehlungen auf Grundlage von riesigen Datenmengen gegeben werden sollen [LSST07]. Aus diesem Grund werden in solchen Fällen meist modelbasierte Collaborative Filtering Systeme verwendet. Bei einem modellbasierten Ansatz wird nur eine Teilmenge der gesamten Datenmenge verwendet und als “Model"verwendet. Dann werden Berechnungen nur noch auf Grundlage des Models gemacht, sodass nicht mehr die gesamte Datenmenge betrachtet werden muss. Dies hat zur Folge, dass eine höhere Skalierbarkeit entsteht, da die Models wesentlich kleiner als die gesamte Datenmenge ist, also bei riesigen Datenmengen immer noch mit einem “kleinen"Model gearbeitet wird. Außerdem wird die Geschwindigkeit erhört, da nicht mehr die gesamte Datenmenge sondern nur noch das Model betrachtet werden muss.

2.4.3.1 Clustering

Eine Methode so ein Model zu erstellen ist mithilfe des Clusterings. Ein Cluster ist eine Ansammlung von Objekten die Ähnlichkeiten zueinander aufweisen und keine oder kaum Ähnlichkeit zu Objekten in anderen Clustern haben sollten. Um zu errechnen welche Objekte ähnlich sind eignet sich wiederrum unter anderem der Pearson- Korrelationskoeffizient. Eine weitere Möglichkeit diese für Datenobjekte $X = (x_1, x_2, \ldots, x_n)$ und $Y = (y_1, y_2, \ldots, y_n)$ zu berechnen ist die Minikowski-Entfernung, definiert als [SK09]

$$d(X,Y) = \sqrt[q]{\sum_{n}^{i=1} |x_i - y_i|^q}, \tag{2.18}$$

mit n als Dimensionsanzahl und x_i , y_i als Werte an der i-ten Dimension des Objekts X und Y. Die Variable q kann beliebig mit einer positiven Ganzzahl vergeben werden. Ist sie 1, entspricht die Gleichung der Manhatten-Abstandsgleichung und ist sie 2, entspricht sie dem Euklidischen Abstand.

Einer der bekanntesten Clustering-Methoden ist k-means. K-means ist eine partitionierende Methode bei der zufällig Clustermittelpunkte gesetzt werden. Danach wird iterationsweise für alle Objekte der Abstand zu einem Clustermittelpunkt berechnet um so die Objekte dem Mittelpunkt mit der geringsten Entfernung zuzuteilen. Danach wird der Clustermittelpunkt neu berechnet. Nun wird wieder die Entfernung jedes einzelnen Objekts zum Clustermittelpunkt berechnet. Dies geschieht so lange bis keine neue Zuordnung auftritt [SY13].

Eine Möglichkeit Clustering zu verwenden wäre mittels der k-Means Methode die Datenmenge zu partitionieren, um dann in den Partitionen einen Memory basierten Ansatz mit beispielsweise Pearson zu benutzen um Vorhersagen innerhalb des Clusters zu treffen [SK09].

Ein durchaus anderer modelbasierter Ansatz, der vor allem das Fehlen von Bewertungen überwinden und treffende Vorhersagen über die Bewertungen geben kann ist die Matrix Faktorisierung (s. Abschnitt 2.8).

2.4.4 Hybride Methoden

Es gibt einige Empfehlungssysteme die einen hybriden Ansatz verwenden indem sie den collaborativen und den contentbasierten Ansatz miteinander verbinden. Dies sorgt dafür die jeweiligen Probleme der Ansätze zu umgehen. Man kann hybrid Empfehlungssysteme in verschiedene Kategorien aufteilen. Eine Methode ist es, eine Empfehlung aus einem collaborativen System und eine aus einem contentbasiertem System zu erstellen um diese beiden Vorhersagen dann zu verbinden. Des Weiteren gibt es die Möglichkeit collaborative Charakteristika in einen contentbasierten Ansatz zu überführen und umgekehrt [AT05]. So ist es zum Beispiel möglich die Vorteile des contentbasiertem Ansatzes hinsichtlich der Empfehlungen aus textuellen Quellen, mehr Informationen über die Benutzer zu erlangen um diese wiederum in einem collaborativem System einzusetzen können.

2.4.4.1 Fazit

Zusammenfassend erkennt man die Unterschiede der Ansätze der Contend-Based Methode und des Collaborative Filterings. Aufgrund der Schwierigkeit der Content- Based Methode Informationen für die Itemprofile zu erlangen wird klar, dass sie sich besonders für textuelle Quellen eignet, da hier Stichwörter leichter extrahiert werden können als zum Beispiel bei der Bestimmung des Inhalts eines Videos. Beim memorybasiertem Collaborative Filtering entsteht das Problem der Skalierbarkeit, da dort, vor allem bei der Nachbarschaftsbestimmung immer rechenintensivere Berechnungen erfolgen müssen, je größer die Datenmenge wird. Aus diesem Grund eignet sich ein Modellbasierter Ansatz besser für Collaborative Filtering in der Realität. Hier ist der klare Vorteil, nicht den gesamten Datensatz für die Bestimmung von Empfehlungen benutzen zu müssen, wie es zum Beispiel bei der Matrixfaktorisierung der Fall ist.

2.5 Bewertungen (Ratings) in Recommender-Systemen

Bewertungen bzw. Raitings sind für Recommender-Systeme wichtig, um dem Benutzer die Produkte vorschlagen zu können, die Ihn auch interessieren. Es gibt zwei verschiedene Feedbackmethoden, mithilfe derer der Nutzer die Produkte bewerten kann. Die erste Methode ist das explizite Feedback und die zweite das implizite Feedback.

Im ersten Teil dieser Arbeit wird auf das explizite Feedback eingegangen. Dabei wird erläutert welche Vorgehensweisen es gibt um diese Bewertungsmethode umzusetzen. Eine dieser Methoden ist zum Beispiel die Sternbewertung von Produkten. Anschließend werden die Vor- und Nachteile dieser Feedbackmethode genannt. Im zweiten Teil dieser Arbeit wird dann auf das implizite Feedback eingegangen. Auch hier werden die Vor- und Nachteile dieser Methode genannt. Außerdem werden zwei verschiedene Vorgehensweisen erklärt, wie das implizite Feedback angewendet werden kann. Die erste Möglichkeit besteht in der Verwendung von Log-Files, um das Verhalten des Users zu beobachten und die zweite Möglichkeit beinhaltet verschiedene Klick-Strategien. Aus dem beobachteten Verhalten lassen sich die Präferenzen und Absichten des Users ableiten. Anschließend können dadurch Produktempfehlungen vorgeschlagen werden. Zum Schluss dieser Arbeit folgt eine kurze Zusammenfassung, in der das Wichtigste nochmal aufgezählt wird.

2.5.1 Explizites Feedback

Eine mögliche Methode um die Meinung eines Nutzers über ein Produkt (Items) beziehungsweise (bzw.) verschiedener Produktgruppen zu erhalten, ist das explizite Feedback. Dabei kann das Produkt sowohl negativ, als auch positiv vom Nutzer bewertet werden. Explizites Feedback bedeutet, dass die Bewertung des Produktes direkt vom Benutzer (User) kommt. Dieses wird in Form von Raitings, wie zum Beispiel (z. B.) Sternbewertungen ermittelt. Der Vorteil dieser Ratings ist, dass sie für den Nutzer nicht viel Zeit in Anspruch nehmen und sehr intuitiv sind. Ein weiteres Beispiel so eines Ratings ist der Daumen hoch von Facebook. Eine weitere Methode, für das explizite Feedback ist die Rezession (Reviews), welche der User für ein Produkt verfasst. Diese Methode nimmt allerdings sehr viel mehr Zeit in Anspruch, sowohl für den Nutzer, der dieses Review erst noch verfassen muss und für das Recommender-System, welches das Review auswerten und bewerten muss, ob es positiv oder negativ ist. In Abbildung 2.7 ist ein Beispiel für ein explizites Feedback zu sehen.

Abbildung 2.7: Sternbewertung eines Produktes [JZFF10].

Eine weitere Form zur Ermittlung von expliziten Feedback ist die Nutzung von jeglichen Angaben, die der Nutzer bereitstellt. Diese Informationen hinterlegt der User in seinem Profil, wie z. B. die Angabe von Hobbies oder Lieblingsbüchern. Der Nutzer teilt dem System also freiwillig mit, was seine Präferenzen sind. Ein Vorteil dieser Methode ist, dass der Nutzer einen direkten Einfluss auf den Empfehlungsprozess hat und dadurch entsprechen die Empfehlungen besser den Präferenzen des Nutzers. Ein Nachteil dieser Methode ist allerdings, dass der User viel Zeit aufwenden muss und daraus eher einen geringen Nutzen zieht [LPP08].

2.5.2 Implizites Feedback

Beim Impliziten Feedback gibt der Nutzer keine direkte Bewertung ab. Die Bewertung der Produkte wird aus dem Benutzerverhalten geschlossen. Es wird also das natürliche Verhalten des Benutzers mit dem System beobachtet, ohne dass dieser etwas davon mitbekommt. Mögliche Quellen zum Schließen auf die Präferenzen des Nutzers können z. B. das auswählen eines Produktes durch einen Mausklick oder das Verweilen bei einem bestimmten Produkt sein. Auf die verschiedenen Arten wird in Kapitel 2.5.2.1 und 2.5.2.2 genauer eingegangen [LPP08].

Ein Nachteil beim impliziten Feedback ist es, dass keine negative Bewertung ermittelt werden kann. Denn es kann nicht davon ausgegangen werden, dass der Nutzer ein Item negativ bewertet, nur weil er dieses noch nie betrachtet hat, denn es besteht auch die Möglichkeit, dass der Nutzer dieses Produkt einfach noch nicht kennt. Auch ist es möglich, dass der Benutzer gerade nicht für sich selber auf der Suche ist, sondern für eine andere Person. Dies verfälscht die späteren Empfehlungen, die dem User gegeben werden. Daher ist es mit dieser Methode schwer zu sagen welches Item dem User gefällt und welches nicht. Die Vorlieben und wahren Motive können also nur geschätzt werden. Außerdem ist die Menge an Daten die bei dieser Methode erhoben werden naturgemäß sehr hoch. Es kann aber nur ein geringer Teil dieser erhobenen Informationen genutzt werden. Es ist also eine Vorverarbeitung der Daten nötig [LPP08].

Der einfachste Fall eines impliziten Feedbacks ist die binäre Bewertung (hat das Objekt genutzt / nicht genutzt). Diese Bewertung reicht aber nur in den seltensten Fällen, denn die Bewertung eines Produktes hängt nicht nur davon ab, ob der User das Produkt nutzt, es hängt auch z. B. davon ab, wie

lange es genutzt wird. Außerdem reicht es nicht aus, dass beim impliziten Feedback nur ein Verhalten (wie z. B. das Speichern) zu beobachten, um ein genaues Feedback zu erhalten [LPP08].

2.5.2.1 Log-Files

Anfangs wurde beim impliziten Feedback ausschließlich die Webseite aufgezeichnet und anhand der Anzeigendauer Empfehlungen für den Nutzer ermittelt. Als Weiterentwicklung dieser Methode wurde neben dem Aufzeichnen der Webseite noch das Verhalten des Users beobachtet. Dieses Verhalten (drucken, speichern, etc.) wird automatisch in Log-Files gespeichert. Sie müssen also nicht extra erhoben werden. Im Folgenden wird anhand eines Beispiels die mögliche Unterteilung von verschiedenen Verhaltensweisen erläutert. Das Verhalten liefert dem Empfehlungssystem ein Feedback. Es wird anhand dieses Feedbacks ermittelt, welchen Wert das jeweilige Objekt für den Nutzer hat [Kul13].

Die verschiedenen Feedbackmethoden werden anhand einer digitalen Bibliothek erklärt, in der der User auf verschiedene Bücher bzw. Büchersammlungen und Dokumente zugreifen kann. Das Verhalten des Users, auf welches geachtet wird um ein Feedback zu ermitteln, kann anhand von zwei Achsen in ein Koordinatensystem unterteilt werden. Die erste Achse ist die Verhaltenskategorie und die zweite Achse ist der Mindestumfang. Die Verhaltenskategorie beinhaltet Aktionen, die der User auf ein Buch oder Dokument anwendet und an dem sein Verhalten beobachtet werden kann. Dazu gehört das Kommentieren, Referenzieren, Behalten oder das Prüfen von verschiedenen Büchern bzw. Dokumenten. Beim Mindestumfang wird die Größe des beobachteten Bereiches berücksichtigt. Dieser Umfang lässt sich in Segment, Objekt und Klasse einteilen. Ein Segment kann z. B. ein Teilbereich eines Buches sein, welches der User betrachtet, wie z. B. die Zusammenfassung. Ein Objekt dagegen ist das gesamte Buch und die Klasse steht für eine Sammlung von Büchern, die das gleiche Thema behandeln.

Die Unterteilung ist in Tabelle 2.2 abgebildet. Es ist möglich, dass sich einzelne Verhaltensweisen in mehreren Kategorien wiederfinden, da Sie nicht ganz eindeutig zuzuordnen sind. Außerdem ist die Tabelle nicht allgemein gültig. Sie dient nur zum besseren Verständnis des Beispiels [KT03].

Die Aktionen, welche in der Tabelle aufgelistet sind beschreiben das Verhalten, welches man bei dem User wahrscheinlich beobachten kann. Das zentrale Ordnungsprinzip für dieses Beispiel ist die vertikale Achse, wo der Zweck des Verhaltens aufgeführt wird. Zusätzlich wird aber noch unterschieden in welchem Umfang das Objekt manipuliert wird [KT03].

Die erste Kategorie „Prüfen“ besteht aus Anschauen, Anhören und Auswählen. Zunächst liefert das System kurze Zusammenfassungen von Büchern, welche dem User vorgeschlagen werden. Wenn der User eine Zusammenfassung länger liest und anschließend das dementsprechende Buch auswählt, registriert das Empfehlungssystem dieses Verhalten und schließt daraus, dass dem User dieses Buch gefällt. Es gibt auch die Möglichkeit, dass das System schon beim Lesen der Zusammenfassung davon ausgeht, dass dem Anwender das Buch oder Dokument gefällt. Dies ist aber nicht sehr genau, da viele Leser verschiedene Zusammenfassungen lesen und danach entscheiden, welches Buch für Sie am interessantesten ist. Als weitere Möglichkeit ist in dieser Kategorie auch das Anhören mit aufgeführt, der Grund dafür ist, dass dem Anwender z. B. die Möglichkeit angeboten werden kann, sich Zusammenfassungen anhören zu können. In diesem Fall wird dann beobachtet, wie lange der User sich die jeweilige Zusammenfassung anhört [ASST05].

Die zweite Kategorie „Behalten“ beinhaltet das Drucken, Speichern, Löschen, Kaufen, das Setzten eines Lesezeichens und die Beschreibung einer Bücher - bzw. Dokumentensammlung. Wenn der User

	Mindestumfang			
Verhaltenskategorien		Segment	Objekt	Klasse
	Prüfen	Anschauen, Anhören	Auswählen	
	Behalten	Drucken	Speichern, Lesezeichen, Löschen, Kauf	Beschreiben
	Referenzieren	Copy-and-paste, Weiterleiten	Verlinken, Weiterleiten	
	Kommentieren	Kennzeichnen	Bewertung	Organisieren

Tabelle 2.2: Klassifikation von Nutzerverhalten für das implizite Feedback [KT03].

Teilbereiche eines Buches bzw. ein Dokument druckt, dann kann davon ausgegangen werden, dass dem Anwender dieses Produkt gefällt. Wenn das System erkennt, dass der User ein Dokument oder Buch kürzer angeschaut hat als ein anderes, das Erste aber gedruckt hat, so sollte sich das System merken, dass dem Anwender das erste Buch bzw. Dokument besser gefallen hat. Wenn der User ein Dokument oder Buch abspeichert oder ein Lesezeichen setzt, dann muss das Empfehlungssystem das Produkt für den Anwender hoch bewerten. Beim Löschen eines Items muss davon ausgegangen werden, dass die Bewertung des gelöschten Objektes geringer ausfallen muss, als die Bewertung der noch abgespeicherten Bücher bzw. Dokumente. Wenn der User eine Kaufentscheidung zu Gunsten eines bestimmten Produktes trifft, ist dies ein starkes Feedback, das dem Anwender das entsprechende Buch bzw. Dokument gefällt. Das letzte Verhalten, welches beim User in dieser Kategorie möglicherweise Beobachtet werden kann, ist das Beschreiben von Bücher – bzw. Dokumentensammlungen. Wenn dieses Verhalten erkannt wird, dann kann davon ausgegangen werden, dass der Anwender sich für dieses bestimmte Themengebiet interessiert [ASST05].

In der dritten Kategorie „Referenzieren“ gibt es Verhaltensweisen, wie Copy-and-Paste, Weiterleiten oder Verlinken. Copy-and-Paste wird dabei auf ein Segment eines Buches oder Dokumentes angewendet. Das Weiterleiten eines Buches oder Dokuments ist dagegen für ein Teilbereich oder das Gesamte Objekt vorstellbar. Das Verlinken eines Buches oder Dokuments ist allerdings nur für das Gesamte Objekt sinnvoll. Aufgrund dieses Verhaltens alleine lässt sich aber nicht darauf schließen, dass dem User dieses Produkt besonders gut gefällt [ASST05].

In der Kommentier Kategorie gibt es die Aktionen Kennzeichnen, Bewertung und Organisieren. Der User kann das Buch oder Dokument z. B. an unterschiedlichen Stellen Kennzeichnen, indem er einzelne Textstellen markiert und Notizen hinzufügt, falls das System diese Funktion anbietet. Daraus kann das Empfehlungssystem dann schließen, dass der Nutzer sich mit dem Objekt umfassender beschäftigt hat. Dadurch nimmt das Produkt an Wichtigkeit für den jeweiligen Nutzer zu. Ein weiteres mögliches zu beobachtende Verhalten eines Users wäre das explizite Bewerten von ganzen Büchern oder Dokumenten, mittels Sternbewertungen. Diese Methode gehört allerdings zum expliziten

Feedback. Bei der Klassenskala ist das Verhalten möglich, dass der User eine Dokumenten- oder Buchreihe für sich selber organisiert bzw. sortiert, um später einen schnelleren Zugang zu dem Thema zu bekommen. Daraus sollte das System schließen, dass sich der User mit diesem Thema beschäftigt [ASST05].

Wie das Beispiel gezeigt hat gibt es verschiedene Verhaltensweisen, welche ein Recommender-System in seiner Bewertung mit einfließen lassen kann bzw. mit einfließen lassen muss. Dieses Verhalten lässt allerdings, für das implizite Feedback typisch, nur auf positives Feedback schließen.

2.5.2.2 Klick Strategien

In diesem Abschnitt wird eine weitere Form vom impliziten Feedback erklärt. Mithilfe von unterschiedlichen Klick Strategien kann ebenfalls auf die Wichtigkeit von Produkten für den jeweiligen User geschlossen werden. Diese Klick Strategien finden dabei meistens bei Suchanfragen oder bei der Aufreihung von Produkten (Listing) auf einer Webseite Anwendung [Kul13].

1. Click > Skip Above: Das geklickte Element hat einen höheren Wert, als die vorherigen Elemente, welche Übersprungen wurden. Bsp.: Bei einer Suchanfrage, in unserem Beispiel, nach Büchern haben die Bücher welche angeklickt wurden eine höhere Relevanz für den Nutzer, als die vorherigen Bücher.

2. Las Click > Skip Above: Das letzte Element welches angeklickt wurde hat eine höhere Relevanz, als die zuvor Übersprungenden. Bsp.: Im Gegensatz zur ersten Strategie hat nur das Buch eine höhere Relevanz als die anderen, nicht angeklickten Bücher, welches zuletzt angeklickt wurde.

3. Click > Earlier Click: Die Klickreihenfolge gibt die Relevanz der Elemente wieder. Bsp.: Werden bei der Suchanfrage die Bücher an der Stelle 3, 5 und 6 angeklickt, hat das Buch an der Stelle drei die höchste Relevanz. Anschließend kommt das Buch an Stelle 5 und zuletzt das Buch an der 6. Stelle.

4. Click > Skip Previous: Das geklickte Element hat ausschließlich eine höhere Relevanz, als das vorherige. Bsp.: Das Buch an der 5. Stelle der Suchanfrage hat eine höhere Relevanz, wenn es geklickt wurde, als das vorher übersprungene Buch an Stelle 4.

5. Click > No-Click Next: Das geklickte Element hat eine höhere Relevanz, als das nachfolgende, wenn der User anschließend kein weiteres Element angeklickt hat. Bsp.: Das Buch an Stelle 4 hat eine höhere Relevanz, wenn es geklickt wurde, als das nachfolgende Element. Das Buch hat aber nur eine höhere Relevanz, wenn anschließend kein weiteres Buch dieser Suchanfrage geklickt wurde.

6. Click > Skip Earlier Query Chain: Das Element, welches geklickt wurde hat eine höhere Relevanz, als die zuvor übersprungenen. Diese Strategie ist wie die erste, Sie gilt allerdings für mehrere Suchanfragen (multiple Suchanfragen). Bsp.: Der User führt mehrere Suchanfragen hintereinander durch und klickt bei dem dritten Suchergebnis das fünfte Buch an. In diesem Fall hat dieses Buch eine höhere Relevanz, als aller Bücher der drei Suchanfragen, welche nicht geklickt wurden.

7. Las Click > Skip Earlier Query Chain: Diese Strategie ist analog zur 2. Strategie und gilt für multiple Suchanfragen.

8. Click > Click Earlier Query Chain: Diese Strategie ist analog zur 3. und gilt ebenfalls für multiple Suchanfragen.

9. Click > Top One No-Click Earlier Query Chain: Das Element, welches geklickt wurde hat eine höhere Relevanz, als das erste in der vorherigen Suchanfrage. Dies gilt aber auch nur, wenn in der vorherigen Anfrage kein Element geklickt wurde. Bsp.: Der User wählt in der ersten Suchanfrage kein Buch aus, sondern erst in der 2. Anfrage das Buch an der 4. Stelle. Dann hat dieses Buch eine höhere Relevanz, als das Buch an der ersten Stelle der ersten Anfrage.

10. Click > Top Two No-Click Earlier Query Chain: Diese Strategie ist analog zur 9. Außer, dass das geklickte Element höher als das zweite der vorherigen Suchanfrage bewertet wird.

11. Top One > Top One Earlier Query Chain: Das erste Element einer neuen Suchanfrage wird immer höher bewertet, als das erste Element der vorherigen Anfrage. Bsp.: Das erste Buch der zweiten Suchanfrage hat eine höhere Relevanz, als das erste Buch der ersten Suchanfrage.

Die ersten fünf Strategien sind eher unpraktisch, da Sie nicht berücksichtigen, dass ein Benutzer mehrere Suchanfragen startet, um an sein gewünschtes Ziel zu gelangen. Außerdem besteht auch die Möglichkeit, dass unter der ersten Suchanfrage keine relevanten Ergebnisse auftauchen. Das hat zur Folge, dass auch keine Elemente gewichtet werden können und somit kann auch nicht auf die Präferenzen des Users geschlossen werden. Die anderen Strategien berücksichtigen diesen Punkt besser und setzten die Suchergebnisse miteinander in Beziehung [Kul13] Die Strategien sollten nicht als absolutes Feedback für ein Empfehlungssystem dienen. Sie sind vielmehr eine Ergänzung zu dem Feedback, welchen aus den Log-Files geschlossen wird. Denn bei dieser Methode wird die Zeit die der User auf einer Seite verbringt nicht mit berücksichtigt.

2.5.3 Zusammenfassung

In diesem Abschnitt wurden die unterschiedlichen Methoden von Bewertungen in Recommender-Systemen behandelt. Eine dieser Methoden ist das explizite Feedback. Bei dieser Bewertungsmethode kommt das Feedback direkt und bewusst vom User. Um die Meinung des Users über ein bestimmtes Produkt zu erhalten, können wiederum verschiedene Methoden verwendet werden. Eine dieser Methode ist zum Beispiel ein Rating-System. Dieses Rating-System kann aus Sternbewertungen oder wie bei Facebook aus einem Daumen hoch oder einem Daumen runter bestehen. Vorteile des expliziten Feedbacks sind, dass der Nutzer einen direkten Einfluss auf das Empfehlungssystem hat und das es möglich ist ein Produkt sowohl negativ, als auch positiv zu bewerten. Ein Nachteil ist allerdings, dass der Nutzer sehr viel Zeit aufwenden muss und keinen wirklichen Nutzen für sich selber sieht.

Eine weitere Feedbackmethode ist das implizite Feedback. Bei dieser Methode ermittelt das Recommender-System die Empfehlungen aus dem Userverhalten. Das bedeutet, dass das System kein direktes Feedback vom Kunden, in Form von Ratings usw. bekommt, sondern dass das System das Verhalten des Users beobachtet und daraus Produktvorschläge ermittelt. Dies ist ein großer Vorteil im Vergleich zur expliziten Methode, denn das System ist nicht direkt vom Feedback des Nutzers abhängig, sondern erschließt sich die Meinung des Users unbemerkt, indem es das Verhalten beobachtet. Für diese Methode gibt es wiederum zwei verschiedene Vorgehensweisen, die sich gegenseitig ergänzen können bzw. sollen. Die erste Vorgehensweise ist das analysieren des Verhaltens über das Aufzeichnen der Webseite und das Verwenden von Log-Files. In diesen Log-Files sind Aktionen, wie das Speichern

oder Drucken von Objekten hinterlegt. Anhand dieser Aktionen kann ermittelt werden wie hoch die Relevanz eines Objektes für den User ist. Ergänzend dazu besteht noch die Möglichkeit durch verschiedene Klick-Strategien die Präferenzen des Nutzers genauer zu bestimmen. Ein Nachteil dieser Methode ist, dass eine große Menge an Daten entsteht. Diese Menge an Informationen muss dann vorverarbeitet werden, damit das Recommender-System überhaupt erst daraus Produktalternativen für den Nutzer vorschlagen kann. Ein weiterer Nachteil ist, dass das Verhalten des Nutzers interpretiert werden muss. Es ist nicht so einfach, wie beim expliziten Feedback, wo ein Daumen hoch bedeutet dem User gefällt das Produkt und ein Daumen runter ihm gefällt es nicht. Die verschiedenen Aktionen des Users müssen miteinander in Verbindung gebracht werden. Es reicht also nicht aus nur ein Teil des Userverhaltens zu berücksichtigen.

Abschließend kann man sagen, dass beide Methoden ihre Vor- und Nachteile haben. Bei der expliziten Methode wird dem Produkt ein Wert zugeordnet, wie z. B. eine fünf für die höchste Bewertung in einer Sternenbewertung. Diese Bewertung gilt unabhängig von anderen Produktbewertungen. Beim impliziten Feedback dagegen bekommt das Produkt keinen eindeutigen Wert zugeordnet. Es wird nur ermittelt, ob der Nutzer das Produkt mag oder nicht bzw. welche Produkte er gegenüber anderen Produkten bevorzugt. Es ist zu empfehlen, dass eine Mischung aus beiden Methoden angewendet wird, um dem Nutzer optimale Produktvorschläge unterbreiten zu können.

2.6 Context-Aware Recommender Systems

Ein Großteil der in Recommender Systemen angewandten Lösungsansätze bezieht lediglich zwei Entitäten in ihre Empfehlungsberechnung mit ein: Die Objekte (Items) und die Benutzer (User). Zusätzlicher Kontext, wie zum Beispiel Zeit, räumliche Umgebung oder in welcher Begleitung sich der User gerade befindet, findet in diesen traditionellen Ansätzen keine Beachtung. Je nach Anwendungsbeispiel kann der Kontext, in welcher eine Empfehlung berechnet werden soll, wichtig werden. Im gleichen Maße kann auch bei der Bewertung von Items Kontext eine Rolle spielen. Die Verwendung von Kontext kann nicht nur Empfehlungen verbessern, sondern auch eventuelle falsche Schlussfolgerungen vorbeugen. Wenn beispielsweise ein Produkt bei einem Online-Warenhaus nicht für einen selbst, sondern als Geschenk bestellt wurde, ist dies für zukünftige Empfehlungen für den User von Bedeutung. In dieser Arbeit wird sich allgemein mit dem Konzept von Context-Aware-Recommender Systemen beschäftigt. Eingestiegen wird mit einer Definition von Kontext und Kontext in Recommender Systemen. Danach werden einige Verfahrensarten erläutert.

2.6.1 Was ist Kontext?

Der Begriff Kontext wird je nach wissenschaftlicher Disziplin unterschiedlich definiert.

In den Sprachwissenschaften und der Kommunikationstheorie bezeichnet man Kontext als die Zusammenstellung aller Elemente einer Kommunikationssituation, die das Verständnis der Äußerung bestimmen [Bußo2]. Kontext kommt von dem lateinischen Wort *contexere*, welches zu Deutsch „zusammenweben“ bedeutet. So wird der Begriff allgemein im Duden als „umgebener Text; Zusammenhang; Inhalt“ definiert. Etwas konkreter kann auch gesagt werden, dass Kontext die Vorrausetzungen oder Gegebenheiten sind, in welchen jede Aussage interpretiert werden wird oder interpretiert werden sollte. Kontext kann als eine Zusammenstellung von Einschränkungen oder Limitierungen verstanden werden, die Grenzen und Thesen erschaffen, welche bestimmte Fakten einer Situation etablieren.

Solche Grenzen können zum Beispiel Überzeugungen oder logische Schlussfolgerungen sein. Die Definition von Grenzen hängt dabei stark von der derzeitigen Arbeitsumgebung ab. Betrachtet man ein konkretes Problemlösungsverfahren, hängt die beste Lösung maßgeblich von ihrem Kontext ab. Ändert sich der Kontext, muss die Lösung vielleicht angepasst werden [McG05]. In Bezug auf Recommender Systeme definiert [ADB+99] den Begriff Kontext als jede Information, die verwendet werden kann, um eine Situation einer Entität zu charakterisieren. Eine Entität kann eine Person, ein Ort oder ein Objekt sein, das relevant für die Interaktion zwischen einem Benutzer (User) und einer Applikation ist, inklusive dem User und der Applikation selbst.

2.6.2 Kontext-Informationen in Recommender Systemen

Recommender Systeme verwenden in der Regel Modelle, die zur Berechnung der Empfehlungen zwei grundlegende Objekte verwenden: Items, die empfehlt werden sollen und User, dem ein oder mehrere Items empfohlen werden soll. Der Empfehlungsprozess startet typischerweise mit der Erstellung von User-Profilen. Jeder User wird zu Beginn dazu angehalten, ihm bekannte Items zu bewerten. Hat der User alle ihm möglichen Bewertungen eingetragen, besteht die Aufgabe des Recommender Systems darin, mithilfe einer Rating-Funktion Empfehlungen von Items für den User zu generieren. Im Allgemeinen kann die Rating-Funktion demnach wie folgt definiert werden [AT08]:

$$R : User \times Item \rightarrow Rating \tag{2.19}$$

User stellt dabei einen aller in der Dimension vorhandenen Benutzer des Systems dar, während Item analog ein Item aller Items im System dar. Sobald die Funktion R für alle User und Items im System vorausberechnet wurde, können dem User die bestmögliche Empfehlung, oder eine Reihe der „besten Empfehlungen“ präsentiert werden. Solche Systeme werden traditionell oder zwei-dimensional (2D) genannt, da sie lediglich zwei Dimensionen in Betracht ziehen (User und Item) [AT08].

Das Empfehlungsproblem wird also auf diese zwei Dimensionen reduziert. Es werden Empfehlungen aufgrund von Bewertungen des Users zu anderen Items oder Bewertungen anderer Usern zu einem bestimmten Item angestellt. Nicht beachtet werden dabei Kontext-Informationen wie Raum und Zeit.

Context-Aware Recommender-Systeme (CARS) hingegen beziehen diese Kontextinformationen in ihren Empfehlungsprozess mit ein, indem sie diese als eine weitere Dimension von wertvollen Informationen behandeln. Neben den Dimensionen der traditionellen Systeme User und Item, gibt es bei CARS eine weitere Dimension *Context* [AT08].

$$R : User \times Item \times Context \rightarrow Rating \tag{2.20}$$

Context kann dabei für mehr als nur eine zusätzliche Dimension stehen, weswegen man in einem CARS nicht mehr von einem zwei-dimensionalen, sondern von einem Multidimensionalen System spricht [AT08]. Jede Dimension besteht dabei aus einer Reihe von Attributen. Es ist auch möglich, dass eine Dimension Sub-Dimensionen hat. Eine Film-Empfehlungsapplikation könnte beispielsweise folgende Dimensionen haben [AT08]. Zunächst werden die grundlegenden Dimensionen User und Item betrachtet. In diesem Fall User und Film.

- User: Die Benutzer, denen Filme empfohlen werden sollen. Kann definiert werden als: (User: UserID, Name, Alter, Geschlecht, Beruf)
- Film: Alle Filme, die einem User empfohlen werden können. (Film: FilmID, Titel, Regisseur, Genre, Jahr, Länge)

Zusätzlich zu diesen traditionellen Dimensionen gibt es im CARS die Kontext- Dimensionen, wie zum Beispiel:

- Kino: Der Ort, an denen die Filme gezeigt werden. (Kino: KinoID, Name, Adresse(Stadt, Bundesland, Land))
- Zeit: Die Zeit, der Zeitpunkt oder der Zeitraum, an dem der Film zu sehen ist, oder gesehen wurde. (Zeit: Datum, Wochentag, Monat, Quartal, Jahr)
- Begleitung: In welcher Begleitung ein User einen Film gesehen hat oder sehen möchte. Es ist definiert als (Begleitung: BegleitungsTyp), wobei BegleitungsTyp verschiedene Werte wie „alleine“, „Freundin“ oder „Familie“ annehmen kann.

Die Bewertung eines Filmes hängt jetzt nicht mehr nur von dem Film selbst ab, sondern von zusätzlichen Kontext-Informationen nämlich wo, wann und mit wem der Film gesehen wurde. Die Art des Filmes, die einer Person empfohlen wird, kann stark davon abhängen, ob sie diesen am Wochenende im Kino mit ihrer Familie sehen möchte oder in der Woche mit ihrem Freund.

2.6.3 Hierarchische Gliederung von Kontext-Informationen

Jede Kontext-Dimension repräsentiert einen anderen Aspekt des Kontexts, in dem sich der User zum Zeitpunkt der Bewertung oder der Empfehlung befindet. Je nach Kontext- Art können die Zuordnungen innerhalb einer Kontext-Dimension sehr komplex werden, weswegen diese Daten gerne hierarchisch strukturiert werden. Dadurch ist es möglich Kontext-Informationen in Bäumen darzustellen, wie es in den meisten CARS gemacht wird [AT08].

Bezogen auf das bereits genannte Beispiel einer Film-Empfehlungsapplikation könnte es folgende Hierarchien geben:

$$Kino : KinoID \rightarrow Stadt \rightarrow Bundesland \rightarrow Land \tag{2.21}$$

$$Zeit : Datum \rightarrow Wochentag \rightarrow Tageszeit, Datum \rightarrow Monat \rightarrow Quartal \rightarrow Jahr \tag{2.22}$$

Kontext-Informationen können demnach als eine Reihe von Kontext-Dimensionen C definiert werden. Jede Dimension K in C kann ist definiert als eine Reihe von q Attributen $K = (K_1, \ldots, K_q)$, welche hierarchisch strukturiert sind und einen bestimmten Kontext-Typ beschreiben, wie beispielsweise Zeit oder Kommunikationsgerät. Die Attribute K_q beschreiben den Kontext auf einer feineren Stufe (höhere Granularität), während K_1 weniger granulare Kontext-Informationen beschreibt [AT08].

2.6.4 Multidimensionale Datenmodelle

Formal lässt sich ein Multidimensionale data (MD) models wie folgt beschreiben: $D_1, D_2, \ldots, D_n$ stehen für verschiedene Dimensionen. Zwei davon sind die Dimensionen User und Item. Die restlichen Dimensionen sind zusätzliche Kontext-Dimensionen. Jede Dimension Di besteht aus einer Reihe von kartesischen Produkten von Attributen $A_{ij}, (j = 1, ..., k_i)$, beispielsweise $D_i \subseteq A_{i1} \times A_{i2} \times \ldots \times A_{ik_i}$, wobei jedes Attribut eine Domäne oder eine Reihe von Werten definiert [AT08]. Darüber hinaus formen ein oder mehrere Attribute einen einzigartigen Schlüssel, mit dem die anderen Attribute zugeordnet werden können [AT08].

Angenommen man hätte eine drei-dimensionale Empfehlungsfunktion [AT08] $User \times Item \times Zeit$, bei der die User Dimension definiert ist als

$$User \subseteq UName \times Adresse \times Einkommen \times Alter \tag{2.23}$$

Demnach hat jeder User einen Namen, eine Adresse, ein Einkommen und hat ein bestimmtes Alter. Die Dimension Item ist definiert als

$$Item \subseteq IName \times Typ \times Preis, \tag{2.24}$$

und besteht aus einer Reihe von Items, welche durch ihren Namen, den Typ und dem Preis definiert werden und die Zeit-Dimension

$$Zeit \subseteq Jahr \times Monat \times Tag, \tag{2.25}$$

und besteht eine Liste von Tagen von einem Startdatum bis zu einem Enddatum (z. B. vom 1. Januar 2015 bis zum 31. Dezember 2015) [AT08]. Im Allgemeinen lässt sich also festhalten, dass der Empfehlungsraum in einem Multidimensionalen Modell als ein Kartesisches Produkt $S = D_1 \times D_2 \times \ldots \times D_n$ definiert werden kann. Die Rating-Funktion kann nach diesem Konzept also wie folgt erweitert werden:

$$R : D_1 \times D_2 \times \ldots \times D_n \rightarrow Rating \tag{2.26}$$

Im Falle des Empfehlungsraumes $User \times Item \times Zeit$ kann eine Rating Funktion definiert werden, die spezifiziert, wie sehr Benutzer u (als Element aus der Dimension User) ein bestimmtes Item i (als Element aus der Dimension Item) zu einer bestimmten Zeit t (als Element aus der Dimension Zeit) mochte: $R(u, i, t)$.

2.6.5 Verwendung von Kontext-Informationen in Recommender Systemen

Die verschiedenen Ansätze Kontext-Informationen in Recommender Systemen zu nutzen, können allgemein in zwei verschiedene Kategorien eingeteilt werden [AT08]:

1. Empfehlungen auf Basis von Kontext-getriebenen Abfragen und Suchen

2. Empfehlungen auf Basis von Erhebungen und Schätzungen von kontext- abhängigen Vorlieben

Systeme, die kontext-getriebene Abfragen und Suchen verwenden benutzen typischerweise Kontext-Informationen (z.B. explizite Informationen wie Stimmung oder aktuelle Interessen des Users; implizite Informationen wie die Uhrzeit, das Wetter oder der Standort des Users), um aus einer Reihe von Items (z. B. Restaurants), das am besten passende, oder eine Reihe von passenden Items auszuwählen (z. B. das beste griechische Restaurant, welches am nächsten dran ist, geöffnet und eine Terrasse hat) und dem User zu präsentieren.

Der zweite Ansatz beschäftigt sich im Gegensatz dazu mit der Erhebung und Abschätzung von kontext-abhängigen Vorlieben der User. Während in kontext- getriebenen Abfragen aktuelle Kontext-Informationen verwenden, wie die aktuellen Interessen des Users, um darauf aufbauend Abfragen zu generieren, welche die bestmöglichen Items empfehlen, wird bei diesem Ansatz versucht die Vorlieben und das Verhalten des Users zu erlernen und zu modellieren, indem sie Interaktionen des Users und anderer User mit dem System überwachen oder indem sie über einen Feedback-Kanal vom User Informationen über die Qualität der vorangegangenen Empfehlungen erhalten. Um diese Vorlieben-Profile zu modellieren und Empfehlungen daraus zu generieren, werden meistens standardmäßige Verfahren aus den 2D Recommender Systemen wie Collaborative Filtering, Content-Based oder Mischformen verwendet. Auch intelligente Daten Analysen von Data-Mining bis zu Maschinen-Learning Ansätze können für diese Aufgaben verwendet werden [AT08].

Da der Trend eher zum zweiten Ansatz geht [AT08], wird sich im Folgenden auf Methoden dieses Ansatzes konzentriert. Ein Recommender System kann al seine Funktion beschrieben werden, welche Nutzer spezifische Daten als Input verwendet, um daraus eine Liste von Empfehlungen für jeden User zu als Output zu generieren. Der lässt traditionelle 2D Empfehlungsprozess sich mit drei Komponenten darstellen: Daten (Data, Input), 2D Recommender System (Function) und die Liste von Empfehlungen (Output) [AT08].

In diesem traditionellen Prozess werden jedoch keine Kontext-Informationen beachtet, welche in jedem der Schritte mit einbezogen werden könnten. Insgesamt gibt es drei verschiedene Vorgehensformen [AT08]:

1. Contextual Pre-Filtering. Bei dieser Methode werden die Kontext-Informationen direkt innerhalb des Daten- Auswahlprozesses (Auswahl von Ratings) mit einbezogen, um eine Sammlung von in diesem Kontext relevanten Daten zu generieren, auf dem dann traditionelle 2D Methoden angewendet werden (siehe Abbildung 2.8a).

2. Contextual Post- Filtering. Bei dieser Methode werden die Kontext-Informationen zunächst ignoriert und erst nach der Verwendung der 2D Methoden beachtet (siehe Abbildung 2.8b).

3. Contextual Modeling. Bei dieser Methode werden die Kontext-Informationen direkt in der Modellierungstechnik verwendet (Multidimensional Data Modelling) (siehe Abbildung 2.8c).

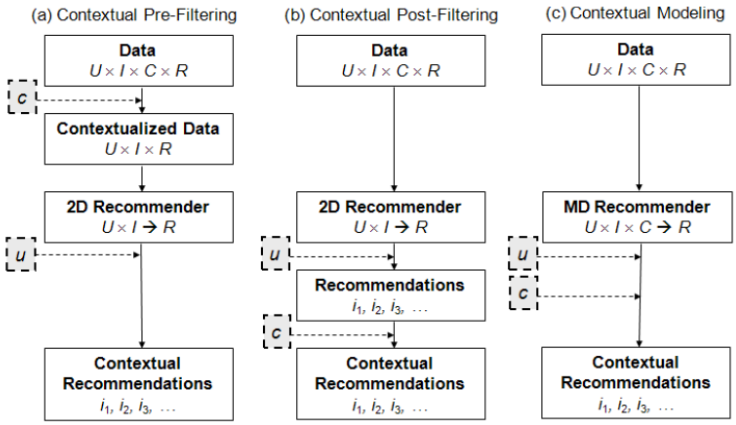


Abbildung 2.8: Möglichkeiten Kontext-Informationen in Recommender Systemen [AT08].

2.6.5.1 Contextual Pre-Filtering

Im Contextual Pre-Filtering werden Kontext-Informationen verwendet, um bei der Generierung von Empfehlungen nur diejenigen Daten zu beachten, die derzeitig relevant sind. Möchte ein User beispielsweise bei einer Filmempfehlungs-Applikation einen Film an einem Sonntag sehen, werden nur Bewertungen in die Berechnung der Empfehlungen miteinbezogen, die an einem Sonntag getätigt wurden. Es werden also Abfragen erstellt, welche den Empfehlungsraum durch die Kontext-Informationen eingrenzt. Ein Vorteil dieser Methode ist, dass nach der Filterung das Problem wieder auf das traditionelle 2D Modell reduziert wurde $(User \times Item)$ und dementsprechend deren Methoden verwendet werden können, um die Empfehlungen zu generieren [AT08].

Diesen Ansatz nennt man auch reduction-based Ansatz, indem es darum geht ein multidimensionales Problem auf ein standardmäßiges 2D $User \times Item$ Problem zu reduzieren [ASST05]. Formal ausgedrückt kann $R^D{}_{User \times Item} : U \times I \rightarrow Rating$ jede 2D Bewertungsfunktion sein, wobei die bereits vorhanden Bewertungen D (D enthält die Daten <$user$, $item$, $rating$> für jede abgegebene Bewertung) und jede mögliche Bewertung für 2D Probleme kalkulieren. Wenn ein 3-Dimensionales Problem demnach definiert werden als $R^D_{User \times Item \times Time} : U \times I \times T \rightarrow Rating$ kann, wobei D alle Datensätze <$user, item, time, rating$> enthält, können 3D-Empfehlungsfunktionen z. B. definiert werden als:

$$\forall (u,i,t) \in U \times I \times T, R^D_{User \times Item \times Time}(u,i,t) = R_{User \times Item}{}^{D[Time=t](User,Item,Rating)}(u,i) \tag{2.27}$$

In diesem Fall ist $[Time = t]$ das Contextual Pre-Filtering und $D[Time = t]$ $(User, Item, Rating)$ repräsentiert die Ergebnisdimension dieser Filterung, welche aus allen Usern und Items besteht, die die Kontextfilterung $Time = t$ erfüllen. Dadurch bleiben nur noch die Dimensionen $User$ und $Item$ übrig und stellen wieder ein klassisches 2D Empfehlungsproblem dar, der sich auf den Empfehlungsraum D bezieht [AT08].

Die Filterung nach Kontext-Informationen kann jedoch zu enggefasst sein. Beispielsweise könnte eine Filterung nach Filmen, die an einem Sonntag bewertet wurden, fast identisch mit den Bewertungen

am Samstag sein. Gleichzeitig können sich Bewertungen von einem Mittwoch jedoch stark von denen am Samstagen und Sonntagen unterscheiden. In diesem Fall würde also die Unterscheidung Wochenende und Werktag sinnvoller sein, als lediglich nach Sonntag zu filtern. Ein weiteres Problem könnte dann entstehen, wenn es nicht genügend Bewertungen an Sonntagen gibt, um eine Aussage darüber treffen zu können, wie gut einem User ein Film an einem Sonntag gefallen könnte. Deswegen kann es sinnvoll sein, Kontext-Informationen zu generalisieren [AT08, ASST05].

Kontext Generalisierung ist ein Vorgehen im Contextual Pre-Filtering, bei dem die Kontextfilterung von einem einzelnen Wert auf eine Reihe von Werten erweitert wird, indem die hierarchisch übergeordnete Größe des Kontextes verwendet wird. In diesem Fall wird nicht mehr vom exakten Kontext, z.B. $[Time = t]$, ausgegangen, sondern von einem generalisierten Kontext $Time \in S_t$. Hier wird S_t auch Kontext-Segment genannt.

Möchte ein User also einen Film am Montag ($Time = Montag$) sehen, können entweder nur die Bewertungen von Montagen miteinbezogen werden, oder der Kontext Montag wird generalisiert und die nächst höhere hierarchische Ebene betrachtet, bei dem alle Bewertungen an Werktagen ($Time \in Werktag$) miteinbezogen werden [AT08, ASST05].

Den richtigen Kontext-Umfang zu definieren wird zu einer der zentralen Aufgabe des Contextual Pre-Filtering. Eine Möglichkeit wäre es Expertenmeinungen mit in die Entscheidung einzubeziehen, welche z. B. sagen, dass es immer besser ist Werktage und Wochenenden zu betrachten, anstatt einzelne Tage. Ein anderer Ansatz wäre zum Beispiel eine automatische Abschätzung des Aufwandes, der Rechenzeit und der Performance der jeweiligen Generalisierungs-Stufen und die Auswahl der besten Variante mit dem besten Ergebnis[AT08]. Darüber hinaus gibt es natürlich noch viele andere, denkbare Methoden. Welche die Richtige ist, hängt vor allem von der Art der Anwendung selbst ab.

2.6.5.2 Contextual Post-Filtering

Die Contextual Post-Filtering Methode ignoriert Kontext-Informationen zunächst komplett und behandelt das Empfehlungsproblem ganz normal im traditionellen 2D- Sinne (Mit den Dimensionen $User \times Item$). Erst nachdem die Liste der besten möglichen Empfehlungen erstellt wurde, werden Kontext-Informationen mit eingebracht und die Liste unter Berücksichtigung dieser Informationen angepasst. Es gibt dabei folgende Anpassungsmöglichkeiten [AT08]:

1. Herausfiltern der Empfehlungen, die durch den zugenommenen Kontext irrelevant werden
2. Anpassung des Rankings innerhalb der Liste, unter der Berücksichtigung des gegebenen Kontextes.

In einem Film-Empfehlungsdienst könnte beispielsweise für einen User, der am Wochenende nur Komödien schaut, alle nicht Komödien aus den Ergebnisempfehlungen herausgefiltert werden. Allgemeiner kann Contextual Post-Filtering als eine Methode beschrieben werden, die versucht ein Vorlieben-Muster für einen bestimmten User innerhalb eines bestimmten Kontextes herauszufiltern (zum Beispiel, dass ein User an Wochenenden nur Komödien schaut) und diese Muster zu verwenden, um die Liste der zuvor erstellen Top-Empfehlungen anzupassen [AT08].

Im Contextual Post-Filtering gibt es zwei grundlegende Techniken: heuristische Techniken und model-based Techniken [AT08].

Heuristische Techniken beschäftigen sich mit dem Finden von einfachen Item- Charakteristiken für einen User in einem bestimmten Kontext (z. B. bevorzugte Schauspieler in einem Film in einem

bestimmten Kontext), die dann verwendet werden, um die Liste der Empfehlungen anzupassen. In diesen Methoden fallen die beiden Anpassungsmöglichkeiten Filtern und Ranking wie folgt aus:

1. Filtern: Herausfiltern der empfohlenen Items, die festgelegte Charakteristiken nicht in einem bestimmten Umfang erfüllen. So könnten zum Beispiel alle Filme herausgefiltert werden, in denen nicht mindestens zwei Lieblingsschauspieler mitspielen
2. Ranking: Bewerten von empfohlen Items auf Grundlage der Menge an relevanten Charakteristiken, die sie erfüllen. So würde der Film, in dem die meisten Lieblingsschauspieler eines Nutzers mitspielen, am höchsten bewertet und platziert werden, während Filme, die diese Charakteristiken nicht erfüllen, weiter untern platziert werden

Bei dem model-based Ansatz werden Vorhersagemodelle erstellt, mithilfe derer eine Wahrscheinlichkeit errechnet wird, mit der ein User eine bestimmte Item-Art in einem bestimmten Kontext wählen würde. Genauer könnte die Wahrscheinlichkeit der Relevanz eines Items (z.B. wie Wahrscheinlich es ist, dass ein User einen Film eines bestimmten Genres in einem gegebenen Kontext auswählt) errechnet werden und diese genutzt werden, um die Liste der Empfehlungen mit den bekannten Anpassungsmöglichkeiten zu bearbeiten [AT08]:

1. Filtern: Herausfiltern von empfohlenen Items, deren Wahrscheinlichkeit auf Relevanz kleiner ist, als ein vorher definierter Minimalwert. So würden Filme, deren Genre mit einer geringen Wahrscheinlichkeit vom User gewählt wird, entfernt
2. Ranking: Items werden je nach Wahrscheinlichkeit, dass sie ausgewählt werden, bewertet und innerhalb der Liste platziert (Gewichten der Empfehlungsbewertungen mit der Wahrscheinlichkeit auf Relevanz in einem bestimmten Kontext)

2.6.5.3 Contextual Modeling

Der Contextual Modeling Ansatz hingegen verwendet Kontext-Informationen direkt in der Empfehlungsfunktion verwendet. Während Contextual Pre-Filtering und Post- Filtering auf traditionelle 2D-Modelle zurückgreifen können, verwendet das Contextual Modeling eine multidimensionale Empfehlungsfunktion, welche durch Vorgehensmodelle oder heuristische Kalkulationen die User und Item Daten ergänzt werden. Methoden des Contextual Modeling können z.B. heuristische Nachbarschafts-Verfahren sein, wie es sie auch in einer 2D Umgebung gibt oder Model-based Ansätze, bei denen beispielsweise 2D Techniken wie Collaborative Filtering (CF) Methoden um Kontext-Informationen erweitert werden [AT08]. Ein weiterer Ansatz ist die Erweiterung von der CF Methode der Matrix-Faktorisierung ist die sogenannte Multiverse Recommendation-Technik, welches für context-aware CF eine N-dimensionales „Tensor Factorization“-Methode verwendet [KABO10].

2.6.6 Zusammenfassung

Durch die Erweiterung des traditionellen zwei-dimensionalen Empfehlungsraumes durch weitere Kontext-Dimensionen, können sehr viel relevantere Empfehlungen für den User generiert werden, gerade weil der Kontext, in welchen sich der User befindet, ausschlaggebend für die Qualität der Empfehlungen sein kann. Der klassische Empfehlungsraum $User \times Item$ wird um eine (oder mehrere)

Kontext-Dimensionen erweitert, wodurch sich die Ratingfunktion spezifiziert: $R : User \times Item \times Kontext \rightarrow Rating$. Allgemeiner könnte man auch definieren $R : D_1 \times \ldots \times D_n \rightarrow Rating$. Durch diese Art der Problembetrachtung sind erweiterte Verfahren nötig, um die passenden Empfehlungen zu generieren. Diese Verfahren sind Contextual Pre-Filtering und Post-Filtering, sowieso Contextual Modeling. Während die ersten beiden Verfahren die gängigen Methoden der 2D Recommender Systeme, wie Collaborative Filtering oder Content-Based Ansätze, verwendet werden können und entweder vorher oder hinterher Kontext-Informationen in die Empfehlungen miteinbezogen werden, verwendet Contextual Modeling diese Kontext-Informationen direkt innerhalb der Berechnung der Empfehlungen. Contextual Modeling verwendet dabei Multidimensionale Data Modeling Methoden, wie z. B. auch in OLAP System verwendet. Die Einschränkungen des Empfehlungsraumes durch Kontext-Information erzeugen offensichtlich Empfehlungen mit höherer Relevanz als diejenigen ohne. Die Herausforderung besteht jedoch vor allem daran, Kontext-Informationen zu extrahieren, den Grad der Generalisierung von Informationen und die Relevanz von Kontext- Dimensionen festzulegen.

2.7 Concept Drift und temporale Effekte

Heute kann fast alles geliked, bewertet oder empfohlen werden. Recommender Systeme kommen auf vielen Seiten zum Einsatz und schlagen dem Nutzer Dinge vor, die vermeintlich seinem Geschmack entsprechen. Wie lange behält eine einmal gegebene Bewertung aber ihre Gültigkeit? Menschen entwickeln sich ständig weiter und so auch ihre Vorlieben und Bedürfnisse, also sind Daten die eben noch für einen Menschen relevant waren nun irrelevant. Dieses Phänomen, der plötzlichen Verschiebung von wichtig zu unwichtig, wird gemeinhin als Concept Drift bezeichnet. Als Beispiel für einen Concept Drift zeigt die nachfolgende Abbildung einen abrupten Sprung der durchschnittlichen Bewertungen für Filme in der Netflix Datenbank.

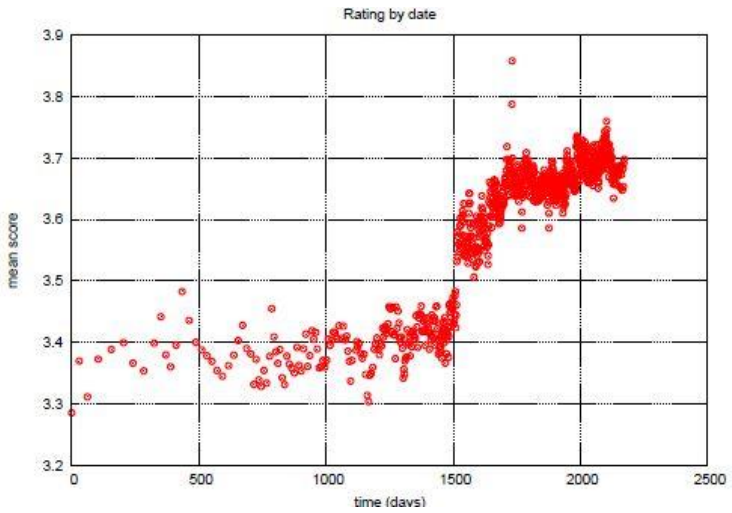


Abbildung 2.9: Sprung der durchschnittlichen Nutzerbewertungen. 1 Punkt stellt 100.000 Bewertungen dar [Kor09].

Recommender Systeme stellt der Umgang mit diesen temporalen Effekten vor einige Herausforderungen. Sie sind nur bedingt vorhersehbar und können auf viele verschiedene Arten auftreten.

Zusätzlich steigt die Datenmenge die ein System verarbeiten muss immer weiter an und wird heute oft als kontinuierlicher Stream präsentiert. Für die Verarbeitung eigenen sich besonders adaptive Lernalgorithmen die die Daten online verarbeiten und es gibt eine Vielzahl von Strategien um einen solchen Algorithmus umzusetzen.

Die Arbeit gliedert sich wie folgt. Der nächste Abschnitt befasst sich mit dem Phänomen des Concept Drifts und wie dieser entsteht. Die dritte Sektion stellt die verschiedenen Formen von adaptiven Lernalgorithmen vor. Im vierten Teil werden die Strategien im Umgang mit Concept Drift präsentiert und erläutert. Der fünfte Abschnitt fasst die Arbeit zusammen.

2.7.1 Concept Drift

Concept Drift kann sich auf verschiedene Arten manifestieren und stellt Recommender Systeme vor komplizierte Probleme. Ein Drift kann global entstehen und Auswirkung auf alle Kunden haben. Durch die Einführung eines neuen Produktes oder Services am Markt kann sich der Fokus der Kunden verändern. Beispielsweise entwickelte sich der Handymarkt hin zu immer kleineren Geräten, durch die Einführung von Smartphones wurde dieser Trend komplett gedreht und läuft nun auf immer größere Geräte hinaus. Einen ähnlich globalen Einfluss weisen saisonale Unterschiede, Feiertage und Events auf. Der Großteil der Ursachen für einen Drift sind allerdings persönliche Faktoren, so hat zum Beispiel die Gründung einer Familie ganz erhebliche Auswirkungen auf das Kaufverhalten der betroffenen Personen. Geschmäcker und Vorlieben ändern sich über die Jahre und entwickeln sich stetig weiter, das ist ein ganz natürlicher Vorgang und muss von Algorithmen bei der Verarbeitung von Informationen zur Erstellung von Empfehlungen berücksichtigt werden [GZB+14].

Für den Concept Drift können folgende Annahmen getroffen werden. Er geschieht unerwartet und ist bis auf wenige Ausnahmen, wie zum Beispiel wiederkehrende Ereignisse, nicht vorhersehbar. Bei Methoden die Concept Drift modellieren muss auf individueller Eben operiert werden, da durch die globalen Drifts die Konsumenten zwar in eine bestimmte Richtung gelenkt werden, aber nicht jeder einzelne zum gleichen Zeitpunkt, in die exakt gleiche Richtung. Einige Konsumenten werden von einem Trend gar nicht erfasst und Driften nicht mit dem Markt mit. Durch die Notwendigkeit auf individueller Ebene zu arbeiten und keine globalen Annahmen treffen zu können verringern sich die zu Verfügung stehenden Daten erheblich. Das hat zur Folge, dass alle Daten die ein Nutzer erzeugt hat, zur Auswertung herangezogen werden müssen, um zwischen einer wirklich persistenten Entwicklung seiner Vorlieben und einzelnen Ausreißern (noise) unterscheiden zu können [GZB+14].

2.7.1.1 Arten des Drift

Es kann zwischen zwei Arten von Drift unterschieden werden:

1. Als Realer Concept Drift wird eine wirkliche Verschiebung des Interesses und der Vorlieben eines Konsumenten verstanden.

2. Virtueller Concept Drift bezeichnet die Veränderung in der Übermittlung, Präsentation oder Verarbeitung der Daten.

Zur Verdeutlichung dient folgendes Beispiel:

Gegenstand der Betrachtung ist ein online Nachrichten Stream über verschiedene Formen der Geldanlage. Die präsentierten Nachrichten müssen nun in relevante und irrelevante Informationen kategorisiert werden. Gehen wir davon aus, dass sich ein bestimmter Nutzer für Anlagen in Aktien interessiert. Daraufhin werden alle Nachrichten, die den Aktienmarkt betreffen, als relevant eingestuft und Nachrichten die beispielsweise Immobilien thematisieren sind irrelevant. Tauscht der Nachrichtendienst jetzt den Redakteur aus oder ändert das Layout der News, bleiben Nachrichten über den Aktienmarkt weiterhin relevant. Dieses Szenario beschreibt einen virtuellen Drift. Trifft der Anleger die Entscheidung zukünftig in Immobilien statt in Aktien zu investieren, werden Nachrichten über Aktien plötzlich irrelevant und Immobilen werden zu relevanten Daten. Das entspricht einem realen Drift [GZB+14].

Abbildung 2.10 zeigt, dass nur der reale Drift die zuvor getroffene Klassifizierung von relevanten und irrelevanten Daten aufhebt. Nach einem realen Drift müssen Entscheidungsmodelle für Empfehlungen neu angepasst werden.

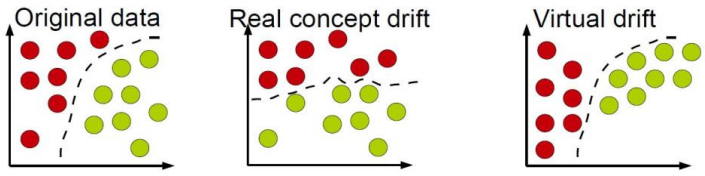


Abbildung 2.10: Darstellung der verschiedenen Driftarten [GZB+14].

2.7.2 Zeitliche Veränderung von Daten

Wie bereits zu Anfang dieser Sektion angesprochen verändern sich Daten und Kundenvorlieben auf die verschiedensten Arten über die Zeit. Abbildung 2.11 stellt das grafisch dar.

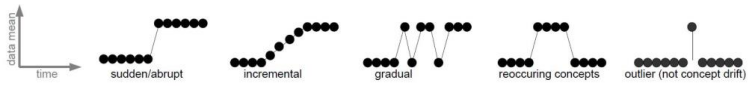


Abbildung 2.11: Arten der zeitlichen Veränderungen [GZB+14].

Ein Drift kann plötzlich auftreten (sudden/abrupt) indem sich das Concept von einem Moment auf den anderen verschiebt. Eine andere Möglichkeit ist die sukzessive (incremental) Veränderung des Concepts. So kann beispielsweise ein alternder Sensor über die Zeit immer ungenauer werden und andere Ergebnisse liefern. Die sprunghaften/schrittweisen (gradual) Veränderungen treten, ähnlich der abrupten, plötzlich auf, springen aber in der Übergangszeit gelegentlich zum alten Concept zurück. So könnte der aus dem Beispiel bekannte Anleger zunächst unsicher sein, ob er wirklich sein Kapital aus Aktien abziehen und in Immobilen stecken soll und verfolgt deshalb beide Märkte. Wiederkehrende Drifts (reoccuring) sind unter anderem saisonal bedingt. Ein Movie Recommender System braucht seinen Nutzern beispielsweise im Sommer keine Filme zu empfehlen die eine weihnachtliche Handlung verfolgen. Ausreißer (hier outliner) stellen die Algorithmen vor eine größere Herausforderung. Es handelt sich dabei nicht um einen Drift und das muss erkannt und umgesetzt werden [GZB+14].

2.7.3 Adaptive Learning

Lernalgorithmen müssen häufig in einem sich ständig verändernden Umfeld arbeiten. Wie das Beispiel des Anlegers, der auf der Suche nach der optimalen Anlage sein Interessensgebiet gewechselt hat, zeigt, dass die meisten Applikationen mit realem Bezug mit Concept Drift in Berührung kommen.

Für maschinelles Lernen gibt es unterschiedliche Ansätze. Für uns interessant ist der Ansatz des „Überwachten Lernens“ (supervised learning). Dabei wird dem Algorithmus ein Set aus Paaren übergeben (X,y), an dem er lernt und ein Vorhersagemodell bilden kann. Als Beispiel dient nochmal das Nachrichtenportal für Anleger. Der Algorithmus bekommt eine große Menge an Daten übergeben, mit denen er nun ein Model anlernen kann. In diesem Trainingsset sind alle X bestimmte Nutzer und alle y die aufgerufenen Artikel. Da alle (X,y) Paare bekannt sind kann ein Model zur Vorhersage angelernt werden, welches bei der Empfehlung interessanter Artikel für einen bestimmten Nutzer Verwendung findet [GZB+14].

Es wird zwischen zwei Lernmethoden unterschieden: Offline und Online. Bei der offline Methode müssen, zum Zeitpunkt des Model Trainings, sämtliche Daten des Trainings Sets vorhanden sein. Nachdem das Training abgeschlossen und das Model angelernt wurde ist es einsatzbereit. Die online Methode verarbeitet die Daten dagegen sequenziell. Es wird ein Model auf Basis eines unvollständigen Trainings Sets gebildet und dann zum Einsatz gebracht. Das Model wird während es sich im Einsatz befindet kontinuierlich aktualisiert sobald neue Trainingsdaten zur Verfügung stehen. Über weniger Einschränkungen verfügt das sogenannte Incremental Learning. Bei dieser Methode werden die Trainings Sets one-by-one oder auch batch-by-batch verarbeitet und nach jedem einzelnen Beispiel wird das Vorhersagemodel aktualisiert [GZB+14].

Adaptive Learning Algorithmen können als Weiterentwicklung der Incremental Learning Algorithmen angesehen werden. Sie müssen über Mechanismen zur Erkennung von Concept Drift verfügen und in der Lage sein ihr Vorhersagemodel an den neuen Gegebenheiten anzupassen. Dafür können die Modelle entweder aktualisiert oder mittels der neuen Daten komplett neu entworfen werden [GZB+14].

Der Prozess lässt sich in 3 Teile gliedern:

- **Vorhersage**. Sobald neue Daten eintreffen kann eine Vorhersage getroffen werden, die auf dem aktuellen Model basiert.
- **Diagnose**. Nach einiger Zeit wird dem Algorithmus Feedback übermittelt (z.B. durch Nutzerbewertung). Mit diesen Informationen wird dann die Abweichung zur Vorhersage berechnet.
- **Aktualisierung**. Das Model zur Vorhersage kann mit den neu gewonnen Informationen aktualisiert werden.

Nachdem der Prozess durchgelaufen ist, müssen die Daten, abhängig von den zur Verfügung gestellten Ressourcen, gelöscht werden. Alternativ können ältere Datensätze zum Abgleich gespeichert werden, das ist allerdings von der gewählten Strategie im Umgang mit Concepct Drift abhängig. Auf die unterschiedlichen Strategien wird im nächsten Abschnitt detaillierter eingegangen [GZB+14].

Im Anschluss an das Update des Models kommt ein neuer Datensatz und der Prozess startet von neuem. Daraus entsteht ein unendlicher Loop aus empfangen, vorhersagen und aktualisieren. Die folgende Grafik illustriert das Zusammenspiel der einzelnen Module eines online Adaptive Learning

Algorithmus. Das Memory Modul entscheidet wie und welche Daten dem Lernalgorithmus übergeben werden. Im Loss Estimation Modul wir die Performance zwischen der Vorhersage und dem Feedback bestimmt und gibt die Informationen danach an das Change Detection Modul weiter, welches über die Anpassung des Vorhersagemodells entscheidet. Für jedes der Module können unterschiedliche Strategien angewendet werden [GZB+14].

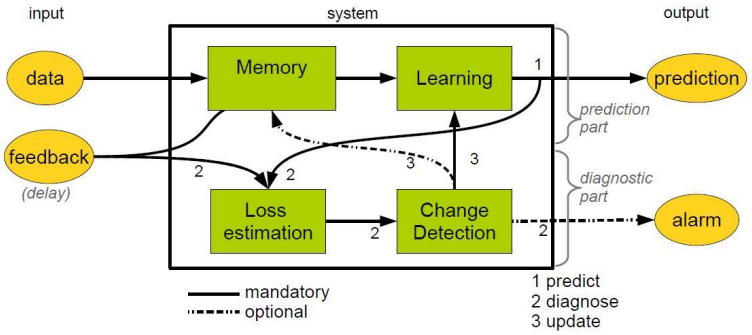


Abbildung 2.12: Die Module eines Adaptive Learning Algorithmus [GZB+14].

2.7.4 Strategien im Umgang mit Concept Drift

In diesem Abschnitt werden verschiedene strategische Ansätze für Lernalgorithmen vorgestellt, die Vorhersagemodelle auf Basis von sich stetig verändernden Daten erzeugen. Dabei wird die Idee dahinter dargestellt und erläutert. Die Strategien sind in die bereits bekannten, und in Abbildung 2.13 noch einmal dargestellten vier Module: Memory, Change Detection, Learning und Loss Estimation aufgeteilt.

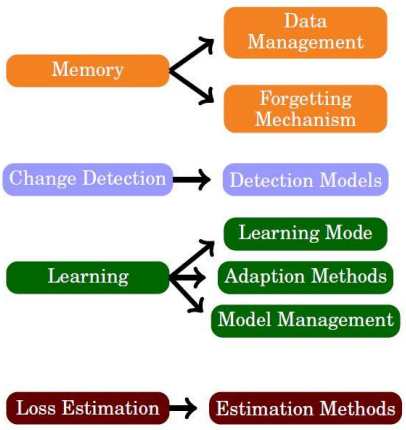


Abbildung 2.13: Die 4 Module und ihre einzelnen Methoden [GZB+14].

Das Leitmotiv zur Darstellung in Modulen ist die Austauschbarkeit der einzelnen Methoden untereinander. Nur weil ein bestimmter Ansatz im Memory Modul gewählt wurde hat das keine Auswirkung

auf die Wahl in anderen Modulen. Im Anschluss an die Erläuterung der Module wird ein davon unabhängiger Ansatz von Koren vorgestellt, der einzelne Tendenzen in sein Model mit einfließen lässt.

2.7.4.1 Memory Modul

In diesem Modul wird der Umgang mit neuen Daten geregelt und es gliedert sich in die Kategorien *Data Management* und *Forgetting Mechanism* auf.

Data Management bestimmt welche Daten für das Lernen verwendet werden. Dabei gehen die meisten Adaptive Learning Algorithmen davon aus, dass die neusten Daten auch die relevantesten für die aktuelle Vorhersage sind. Deshalb werden im Data Management typischerweise die neusten Daten zum Lernen verwendet. Das geschieht entweder in einem single example oder multiple example Ansatz.

Das single example Verfahren hat seinen Ursprung in online Adaptive Learning Algorithmen, die nur von jeweils einem Datensatz lernen können und später keinen Zugriff mehr auf ältere Datensätze haben. Die Daten werden nacheinander, abhängig von ihrer Ankunft, abgearbeitet, ohne das gesamte Trainings Set im Speicher zu behalten. Bei Ankunft neuer Daten wird eine Vorhersage mit dem aktuellen Model gemacht. Mithilfe des Feedbacks wird die Abweichung der Vorhersage erstellt und das Model gegebenfalls aktualisiert. Durch diese Variante entwickelt sich das Model ganz natürlich stetig weiter und braucht auch keinen explizit formulierten Forgetting Mechanism, da keinerlei Datensätze gespeichert werden. Aus diesem Grund ist der Algorithmus sehr stabil gegenüber noise, wodurch er aber sehr träge auf einen plötzlichen Drift reagiert [GZB+14].

Beim multiple example Ansatz werden die Daten nicht sofort gelöscht. Der Algorithmus hält ein Set aus den aktuellsten Daten in seinem Speicher und lernt dann daraus ein Model an. Dieser Prozess teilt sich in zwei Schritte auf. Zuerst wird das Model mit den neuen Daten aktualisiert, danach werden alte Daten aus dem Speicher gelöscht (nach dem FIFO Prinzip). Die Herausforderung bei dieser Methode besteht darin, einen adäquat großen Datensatz, Window genannt, zum Lernen des Models zu halten. Ein kleines Window reflektiert den momentanen Verlauf der Daten besser und kann wesentlich schneller auf Veränderungen reagieren. In stabilen Phasen beeinträchtigt ein kleines Window allerdings die Performance des Algorithmus. Bei einem großen Window tritt der genau gegenteilige Effekt ein. Im Allgemeinen ist ein Window entweder fix oder variabel. Fixe Windows speichern eine immer gleiche Anzahl von Daten ab. Wird ein variables Window gewählt, variiert die Anzahl der gehaltenen Daten. Typischerweise meldet das Change Detector Modul, ob das Window vergrößert oder verkleinert werden soll. Sollte es zu einem abrupten Concept Drift kommen, kann das Window verkleinert werden, um das Vorhersagemodel schneller an die neuen Gegebenheiten anzupassen. In stabileren Phasen kann das Window wieder vergrößert werden, um die Performance zu verbessern [GZB+14].

Der **Forgetting Mechanism** entscheidet darüber auf welche Art die alten Daten verworfen werden. Es ist der am weit verbreitetste Ansatz um mit dynamischen Veränderungen bei Daten umzugehen. Dabei werden die veralteten Daten einfach verworfen. Auch bei dieser Methode gibt es wieder einen Tradeoff zwischen der Robustheit gegenüber Noise und Reaktionsfähigkeit auf Drifts. Je abrupter das Vergessen der Daten, desto schneller wird die Reaktionsfähigkeit, aber desto anfälliger wird das Model für Noise. Forgetting wird in abrupt und gradual unterschieden [GZB+14].

Abrupt Forgetting. Zu jedem Zeitpunkt definiert ein Set aus Beobachtungen ein Window, welches die Informationen enthält, die für das Lernen eines Models in Frage kommen. Beim Abrupt Forgetting, auch partial memory genannt, befindet sich eine Beobachtung entweder innerhalb des definierten Win-

dows oder außerhalb. Es gibt auch für das Abrupt Forgetting verschiedene Ansätze zum Umgang mit den Daten. Sie unterscheiden sich in der Größe des definierten Windows oder dem Speicherzeitpunkt. Sämtliche Daten die keine Relevanz für das anlernen des Models haben werden in dieser Methode verworfen [GZB+14].

Gradual Forgetting. Im Gegensatz zum vorigen Ansatz ist das Gradual Forgetting ein full memory Ansatz. Die Daten werden nach ihrem Alter gewichtet mit der Prämisse, dass sie über die Zeit immer unwichtiger werden. Demnach basiert das Vorhersagemodel immer auf allen gesammelten Daten. Aufgrund der Gewichtung beeinflussen ältere Daten das aktuelle Modell aber nur noch in geringem Umfang [CDC14].

2.7.4.2 Change Detection

Das Modul Change Detection beinhaltet Techniken und Mechanismen um Abweichungen von der Vorhersage zu interpretieren und daraus einen eventuellen Drift zu erkennen. Wie bereits erwähnt brauchen online Learning Systeme kein Change Detection, da sie Veränderungen sofort in ihr Model aufnehmen und verarbeiten. Was zunächst wie ein Vorteil aussieht. Aber ohne ein Change Detection erhält der Algorithmus keine weiteren Informationen über mögliche Wiederholungen und zeitliche Zusammenhänge von Drifts. Eine typische Strategie im Bereich des Change Detections ist die Überwachung und der statistische Abgleich von Performanceindikatoren anhand einer vorher definierten Baseline. Als Indikatoren dienen Werte wie die Genauigkeit der Vorhersage oder die Abdeckung des aktuellen Models. Es besteht auch die Möglichkeit ein Change Detection mit zwei Referenzwindows zu implementieren. Dabei fast ein Window alle bisherigen Informationen zusammen und das zweite Window beinhaltet nur die aktuellsten Daten. Die Ergebnisse der beiden werden miteinander mittels Nullhypothese verglichen. Wird die Hypothese verworfen liegt ein Drift vor und das Model muss angepasst werden [GZB+14].

2.7.4.3 Learning Modul

Das Learning Modul enthält Methoden die aus den Daten, die zum Anlernen an das Modul übergeben wurden, eine generelle Annahme trifft und das Vorhersagemodel mit den neuen Trainingsdaten aktualisiert. Abbildung 2.14 gibt die Struktur dieses Abschnitts wieder, nach der die einzelnen Komponenten des Moduls analysiert werden. Learning Mode beinhaltet verschiedene Techniken die zur Aktualisierung verwendet werden. Danach werden bei Adaption Methods die Möglichkeiten zur Analyse des Verhaltens von Vorhersagemodellen an Daten mit zeitlichem Bezug dargestellt. Im Model Management werden am Schluss noch die Methoden zur Aufrechterhaltung eines aktiven Vorhersagemodels gezeigt [GZB+14].

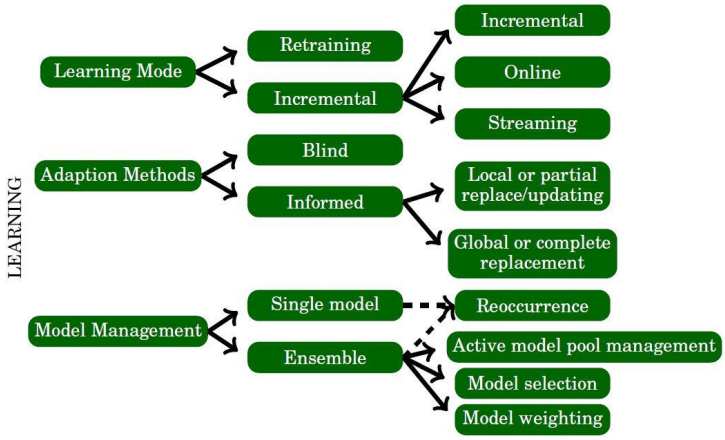


Abbildung 2.14: Methoden des Learning Moduls [GZB+14].

Der Learning Mode kann in Retraining und Incremental aufgeteilt werden. Wird der Retraining Ansatz gewählt ist ein Speicher notwendig, der zwischenzeitlich die neuen Daten buffern kann. Erhält der Algorithmus neue Daten, mit denen er das Model anpassen muss, so wird das alte Model komplett verworfen. Die neuen Daten werden mit den alten Daten zusammengeworfen und aus diesem Set wird dann ein neues Model angelernt. Incremental Learning hingegen bewahrt das bereits gelernte Model und aktualisiert die entscheidenden statistischen Größen. Es kann noch weiter in Online und Streaming Learning Modes unterschieden werden. Beide aktualisieren das Model anhand der aktuellsten Daten die zur Verfügung stehen. Der Streaming Modus ist allerdings auf die Verarbeitung von großen kontinuierlichen Datenmengen ausgelegt und bietet die Möglichkeit die Daten über mehrere Durchläufe zu betrachten, wobei sie typischerweise nur für einen Durchlauf gespeichert werden [GZB+14].

Bei den Adaption Methods wird in Blind und Informed Adaption unterschieden. Der **Blind** Modus ist proaktiv und läuft immer wieder ohne explizite Information über einen Drift ab. Online und Streaming Learning sind zwei Beispiele dafür. Das Model wird jedes Mal aktualisiert wenn es zwischen der Vorhersage und dem Feedback eine Abweichung gibt. Dadurch werden alte Daten immer mit einer konstanten Geschwindigkeit verworfen, was bei einem möglichen Concept Drift von Nachtteil ist. Der Algorithmus hat keine Möglichkeit die Daten schneller oder langsamer zu verwerfen und kann damit nicht geeignet auf einen Drift reagieren. **Informed** Adaption Strategien sind dagegen reaktiv. Es wird ein Trigger gesetzt der den Algorithmus auslöst, das kann entweder ein Change Detector sein oder speziell definierte Events. Die Reaktion auf einen Drift kann zwei unterschiedliche Formen annehmen. Bei einer globalen Reaktion wird das gesamte Model verworfen und danach neu entworfen. Für lokale Reaktionen muss nur ein Teil des Models abgeändert werden. In den meisten Fällen ist eine lokale Änderung des Models ausreichend, da ein Concept Drift nicht immer das ganze Model betrifft. Welche Reaktion erfolgt hängt aber letztendlich davon ab welche Algorithmen zum Einsatz kommen und wie diese mit einem Drift umgehen [GZB+14].

2.7.4.4 Model Management

Ensemble Learning speichert mehrere Modelle ab, die eine kombinierte Vorhersage erstellen. Beim Ensemble Learning wird davon ausgegangen, dass während eines Drifts, die Daten aus unterschiedlichen Quellen beeinflusst werden. Aus diesem Grund werden sie von verschiedenen Modellen verarbeitet. Das Endergebnis ist ein gewichteter Durchschnitt der Einzelergebnisse. Die Gewichtung hängt von der Performance der Modelle bei der Verarbeitung der aktuellsten Daten ab und kann über die Zeit angepasst werden. Hinter dem **Reoccuring Concept Management** steckt die Idee, dass Drifts über die Zeit wiederkehren könnten. Dabei werden alte Modelle nicht verworfen sondern abgespeichert und in einen Schlafmodus versetzt. Wiederholt sich ein Drift kann das alte Model wieder geladen werden und zum Einsatz kommen [GZB+14].

2.7.4.5 Loss Estimation

Das Loss Estimation Modul kann Model Dependent oder Independent gestaltet sein. Die **Model Dependent** Variante erkennt Concept Drifts mittels Support Vector Machine Eigenschaften. Die Methode bildet ein Window über das Trainings Set. Auf das Set wird eine Leave-One-Out-Kreuzvalidierung ausgeführt welche zwar sehr genau aber auch sehr teuer ist. Beim **Model Independent** Verfahren werden zwei Windows gebildet. Ein kleines mit den aktuellsten Daten und ein großes, welches als Referenz verwendet wird. Das große Fenster ist gegenüber Noise robuster und reagiert verhalten auf neue Daten. Im Gegensatz dazu wird das kleine Window stärker reagieren. Ist das Ergebnis des kleinen erheblich über dem des großen Windows liegt ein Drift vor und es muss reagiert werden [GZB+14].

2.7.4.6 Nutzerverzerrung

Dieser Ansatz geht davon aus, dass Empfehlungen nicht nur von temporalen Effekten betroffen sind, sondern auch von Verzerrungen oder Tendenzen, die bei der Modellierung beachtet werden müssen. Als Beispiel ist die Datenbank von Netflix gegeben, in der Filme mit durchschnittlich 3,6 Sternen bewertet werden. Bei einer Empfehlung muss auch darauf geachtet werden welcher Nutzer welchen Film bewertet und wie groß deren Verzerrung zum Durchschnitt ist. So kann zum Beispiel Nutzer X grundsätzlich alle Filme um 0,4 Punkte schlechter bewerten als alle anderen Nutzer. Damit hat eine Bewertung von 3,0 eine andere Bedeutung für ihn, als für den Rest der Nutzer. Genauso kann ein Film Y um durchschnittlich 0,6 Punkte besser bewertet werden als die restlichen Filme [Kor09].

Die Überlegung hinter diesem Verfahren ist die Annahme, dass bei einer Skala von 0-5 die einzelnen Stufen für jeden Nutzer eine leicht andere Bedeutung haben und das mit in die Empfehlung einbezogen werden muss.

2.7.5 Zusammenfassung

Die Arbeit soll zeigen dass, bei der Entwicklung eines Recommender Systems, temporalen Effekten ein hohes Maß an Bedeutung eingeräumt werden muss. Wie das Problem des Concept Drifts zeigt, verändern sich Kundenmeinungen einzeln und im Kollektiv. Dabei driftet jeder Nutzer individuell, angetrieben von seinen eigenen Vorlieben und Bedürfnissen, und gesamtheitlich mit dem Markt mit, zum Beispiel bei der Einführung neuer Produkte.

Ein Drift kann auf unterschiedliche Arten auftreten. Neben dem abrupten, incremental und gradual Drift ist besonders der reoccuring Drift herauszustellen. Der durch seine wiederkehrende Art, ein plumpes Verwerfen der vorhandenen alten Daten, zukünftig redundante Operationen auslöst.

Ein kurzer Einblick in die Funktionsweise von adaptiven Lernalgorithmen zeigt ihren Umgang mit Daten in einem dynamischen Umfeld. Die wichtigsten Funktionen eines solchen Algorithmus sind die Vorhersage, die Diagnose und die Aktualisierung. Beim Eintreffen neuer Daten wird eine Vorhersage auf Basis der neuen gewonnen Informationen ausgegeben. Nach dem Erhalt von Feedback kann die Genauigkeit der getroffenen Vorhersage überprüft und danach überarbeitet werden. Für die Umsetzung eines solchen Algorithmus wurden verschiedene Strategien aufgezeigt und die Idee und Funktionsweise erläutert. Ein wichtiger Ansatz dabei war die Modularität der einzelnen Strategien, welche sich teilweise untereinander ergänzen, aber zum großen Teil frei wählbar sind.

2.8 Matrix-Faktorisierungs-Methoden in Recommender-Systemen

Matrix-Faktorisierung ist ein mathematisches Verfahren, welches beim Collaborative Filtering angewendet werden kann. Collaborative Filtering in Recommender Systemen zeichnet sich grundsätzlich dadurch aus, dass neue Empfehlungen für sog. Items für Nutzer allein aus dem Nutzerverhalten ermittelt werden können. Collaborative Filtering-Methoden berücksichtigten dabei die durch den Nutzer hergestellten Beziehungen zu Items (z. B. durch Produktratings). (vgl. [KBV09], S.30 f.)

Eine Alternative zu Collaborative Filtering-Methoden sind Content Filtering-Methoden. Diese ermitteln Empfehlungen für Items basierend auf externen Informationen. Eigenschaften von Nutzern werden mit Eigenschaften von Items in Verbindung gesetzt. Dabei sind sowohl die Eigenschaften der Nutzer (z. B. Alter, Geschlecht) zu einem Nutzerprofil als auch die Eigenschaften der Items (z. B. Produktkategorie) zu einem Itemprofil zu erheben. (vgl. [KBV09], S.30 f.)

Zwei wichtige Methoden des Collaborative Filterings sind Nachbarschaftsmethoden und Latent Factor Models. Mittels der Nachbarschaftsmethoden werden Beziehungen zwischen Nutzern oder zwischen Items hergestellt. Latent Factor Models, zu denen auch die Matrix-Faktorisierung zählt, berücksichtigen sowohl Nutzer als auch Items und bilden diese zusammen im selben „Faktoren-Raum“ ab. Dadurch werden Nutzer und Items durch dieselben Faktoren beschrieben. (vgl. [KB11], S. 145 f.)

Ein Beispiel für die Darstellung von Präferenzen der Nutzer für bestimmte Items sind Ratings. In Tabelle 2.3 sind die Ratings, die in einem Wertebereich zwischen 1 und 5 liegen, von fünf Nutzern abgebildet. Von Nutzern nicht bewertete Items werden mit „-“ markiert. Durch die Matrix-Faktorisierung sollen diese fehlenden Bewertungen vorhergesagt werden, um Empfehlungen geben zu können.

Nutzer	Item 1	Item 2	Item 3	Item 4
1	5	3	-	1
2	4	-	-	1
3	1	1	-	5
4	1	-	-	4
5	-	1	5	4

Tabelle 2.3: Nutzer bewerten Items zwischen 1 und 5. (vgl. [Yeu10]).

2.8.1 Grundlagen zur Matrix-Faktorisierung

Das Collborative Filtering setzt bei zwei grundlegende Entitäten an: bei den Nutzern und den Items. Für die Methode der Matrix-Faktorisierung wird das Nutzerverhalten herangezogen, welches explizit oder implizit bekannt ist. Explizites Feedback bedeutet, dass der Nutzer seine Präferenzen zu Items explizit angibt, z. B. in Form von Bewertungen. Implizites Feedback bedeutet, dass das Verhalten des Nutzers beobachtet wird, z. B. welche Items der Nutzer verwendet oder welche Mausbewegungen ausgeführt werden. (vgl. [KBV09], S. 32 und [KB11], S. 146)

Die Eingabedaten (sowohl explizites als auch implizites Feedback) können in einer Nutzer-Item-Matrix (sog. Präferenzmatrix) zusammengefasst werden (s. Gleichung 2.28). In Gleichung 2.28 ist auch das Beispiel aus Tabelle 2.3 als Nutzer-Item-Matrix dargestellt.

$$\boldsymbol{R} = \begin{pmatrix} r_{11} & \dots & r_{1j} & \dots & r_{1n} \\ \vdots & & \vdots & & \vdots \\ r_{i1} & \dots & r_{ij} & \dots & r_{in} \\ \vdots & & \vdots & & \vdots \\ r_{m1} & \dots & r_{mj} & \dots & r_{mn} \end{pmatrix}, \quad \text{Beispiel:} \quad \boldsymbol{R} = \begin{pmatrix} 5 & 3 & - & 1 \\ 4 & - & - & 1 \\ 1 & 1 & - & 5 \\ 1 & - & - & 4 \\ - & 1 & 5 & 4 \end{pmatrix} \tag{2.28}$$

Das Element r_{ij} gibt die Präferenz des i-ten Nutzers für das j-te Item an, wie z. B. in der Beispielmatrix in Gleichung 2.28 $r_{32} = 1$. Bei m Nutzern und n Items ist $\boldsymbol{R} \in \mathbb{R}^{m \times n}$. (vgl. [TPNT08], S.555 und [Yeu10])

Die Nutzer-Item-Matrix gilt es in eine Nutzer-Matrix $\boldsymbol{P} \in \mathbb{R}^{m \times f}$ und eine Item-Matrix $\in \mathbb{R}^{n \times f}$ zu faktorisieren, sodass Gleichung 2.29 gilt (vgl. [TPNT08], S. 555).

$$\boldsymbol{R} \approx \boldsymbol{P} \cdot \boldsymbol{Q}^T \quad \text{mit } \boldsymbol{P} = \begin{pmatrix} p_{11} & \dots & p_{1f} \\ \vdots & & \vdots \\ p_{m1} & \dots & p_{mf} \end{pmatrix} \quad \text{und } \boldsymbol{Q} = \begin{pmatrix} q_{11} & \dots & q_{1f} \\ \vdots & & \vdots \\ q_{n1} & \dots & q_{nf} \end{pmatrix} \tag{2.29}$$

Mit f wird die Anzahl der Faktoren (auch Features genannt) angegeben, mit denen die bestehenden Präferenzen erklärt werden können. Das Element p_{11} gibt beispielsweise an, wie stark das Feature 1 des Nutzers 1 alle Präferenzen des Nutzers 1 beeinflusst.

Die in Gleichung 2.29 angegebene Matrix-Faktorisierung ähnelt der Singulärwertzerlegung von Matrizen (kurz SVD für singular value decomposition). Mittels der SVD existiert ein etabliertes Verfahren, eine Matrix in drei Matrizen zu faktorisieren. Die Voraussetzung zur Anwendung der SVD ist eine voll besetzte Matrix, die faktorisiert werden soll. In Recommender Systemen existieren dünn besetzte Nutzer-Item-Matrizen, da Nutzer nicht alle Items bewerten und Items nicht von allen Nutzern bewertet werden. Die bekannten Präferenzen müssen demnach zum Faktorisieren ausreichen. Dazu können lernende Algorithmen herangezogen werden, die in den folgenden Abschnitten näher erläutert werden (vgl. [KBV09], S. 32).

2.8.2 Grundlegendes Modell der Matrix-Faktorisierung in Recommender-Systemen

$\boldsymbol{P}$ und $\boldsymbol{Q}$ dienen nicht nur zur Erklärung der bestehenden Präferenzen, sondern auch zur Vorhersage der unbekannten Präferenzen, um folglich Nutzern für bestimmte Items Empfehlungen zu geben. Ziel der Matrix-Faktorisierung ist es, die Matrizen $\boldsymbol{P}$ und $\boldsymbol{Q}$ zu ermitteln, sodass das Produkt $\boldsymbol{PQ}^T$ die ursprüngliche Nutzer-Item-Matrix $\boldsymbol{R}$ gut abbildet und gleichwohl die in $\boldsymbol{R}$ unbekannten Elemente vorhersagt. Gleichung 2.29 wird somit zu Gleichung 2.30 erweitert. $\hat{\boldsymbol{R}}$ ist ein die vorhergesagte Nutzer-Item-Matrix.

$$\boldsymbol{R} \approx \boldsymbol{P} \cdot \boldsymbol{Q}^T \quad \wedge \quad \hat{\boldsymbol{R}} = \boldsymbol{P} \cdot \boldsymbol{Q}^T \tag{2.30}$$

Ein einzelnes Element der Matrix $\hat{\boldsymbol{R}}$ wird wie in Gleichung 2.31 angegeben berechnet.

$$\begin{aligned} \hat{r}_{ij} = \vec{p_i} \cdot \vec{q_j}^T \; \forall \, (i,j) \in \mathcal{R} \; : \vec{p_i} &= \begin{pmatrix} p_{i1} & \dots & p_{if} \end{pmatrix} \; \wedge \\ \vec{q_j} &= \begin{pmatrix} q_{j1} & \dots & q_{jf} \end{pmatrix} \; \forall \, (i,j) \; \in \; \mathcal{R} \end{aligned} \tag{2.31}$$

Alle bekannten (i,j)-Präferenz-Paare sind Elemente der Menge $\mathcal{T}$. Die Trainingsmenge $\mathcal{R} \subseteq \mathcal{T}$ wird zum Ermitteln der Nutzer- und Item-Matrizen herangezogen. Darüber hinaus kann es eine Validierungsmenge $\mathcal{V} \subset \mathcal{T} \setminus \mathcal{R}$ geben, gegen die die Ergebnisse der Matrix-Faktorisierung evaluiert werden können. (vgl. [TPNT08], S. 554)

Das Ermitteln der bestmöglichen Nutzer- und Item-Matrizen stellt ein Optimierungsproblem dar. Dabei gilt es, den Fehler zwischen bekannten Präferenzen und ermittelten (vorhergesagten) Präferenzen zu minimieren. In der Literatur finden sich dabei unterschiedliche Fehlerfunktionen, z. B. die von [TPNT08], die Gleichung 2.32 angegeben ist. (vgl. [TPNT08], S. 555) Das Quadrieren der Differenz dient dazu, zum einen negative Fehler zu vermeiden und zum anderen größere Differenzen stärker, d. h. quadratisch, zu bestrafen.

Zwei mögliche Optimierungsprobleme sind in Gleichung 2.33 als Summe aller Fehlerwerte aus Gleichung 2.32 und in Gleichung 2.35 als sog. „Root Mean Square Error“ (kurz RMSE) aufgestellt. Dabei handelt es sich jeweils um Minimierungsprobleme, die äquivalent zueinander sind. In der Literatur ist der RMSE als Zielfunktion für die Minimierung der Fehler zwischen bekannten und vorhergesagten Werten etabliert. (vgl. [TPNT08], S. 555)

$$e_{ij} \; = \; \frac{1}{2}(r_{ij} - \hat{r}_{ij})^2 \quad \forall \, (i,j) \in \mathcal{R} \tag{2.32}$$

$$SSE \; = \; \sum_{(i,j) \in \mathcal{R}} \left(e_{ij}\right) \rightarrow min \tag{2.33}$$

$$se_{ij} \; = \; (r_{ij} - \hat{r}_{ij})^2 \quad \forall \, (i,j) \in \mathcal{R} \tag{2.34}$$

$$RMSE \; = \; \sqrt{\frac{\sum\limits_{(i,j) \in \mathcal{R}} \left(se_{ij}\right)}{|\mathcal{R}|}} \rightarrow min \tag{2.35}$$

Beispiel: Initiale Nutzer- und Item-Matrizen können zufällig erstellt werden. Mit zwei Faktoren sind die Nutzer-Matrix $\boldsymbol{P}$ sowie die Item-Matrix $\boldsymbol{Q}$ und die vorhergesagte Matrix $\hat{\boldsymbol{R}}$ in Gleichung 2.36 aufgestellt. Die Fehler SSE und $RMSE$ betragen $844,25$ und $8,059$. Die grau hinterlegten Elemente markieren die in der Ausgangsmatrix unbekannten Präferenzen.

$$\boldsymbol{P} = \begin{pmatrix} 1 & 2,5 \\ 1 & 3 \\ 0,5 & 3 \\ 1 & 3,5 \\ 1 & 3 \end{pmatrix} \text{ und } \boldsymbol{Q} = \begin{pmatrix} 2 & 4 \\ 2 & 2 \\ 2 & 5 \\ 1 & 2 \end{pmatrix}, \text{ sodass } \hat{\boldsymbol{R}} = \boldsymbol{P}\boldsymbol{Q}^T = \begin{pmatrix} 12 & 7 & 14,5 & 6 \\ 14 & 8 & 17 & 7 \\ 13 & 7 & 16 & 6,5 \\ 16 & 9 & 19,5 & 8 \\ 14 & 8 & 17 & 7 \end{pmatrix}. \quad (2.36)$$

2.8.3 Lösungsmöglichkeiten

Zum Ermitteln der Nutzer- und Item-Matrizen können lernende Algorithmen herangezogen werden. Der grundlegende Ablauf ist in Algorithmus 2 dargestellt. Die Initialisierung der Matrizen $\boldsymbol{P}$ und $\boldsymbol{Q}$ kann zufällig erfolgen. Iterativ werden die Matrizen angepasst, sodass die Fehlerfunktion minimiert wird. Der Algorithmus terminiert, wenn das Minimum oder eine definierte Iterationenanzahl erreicht ist. In den folgenden zwei Abschnitten werden zwei Methoden zur Anpassung der Matrizen $\boldsymbol{P}$ und $\boldsymbol{Q}$ vorgestellt.

Algorithm 2 Algorithmus zur Matrix-Faktorisierung in Pseudocode. (vgl. [TPNT08], S. 555)

Data: Präferenzmatrix $\boldsymbol{R}$
Result: $\boldsymbol{P}, \boldsymbol{Q}$
init $\boldsymbol{P}$
init $\boldsymbol{Q}$
berechne SSE oder RMSE;
while Terminierungsbedingung (z. B. Iterationenanzahl, SSE kleiner 0,1) **do**
    Anpassung $\boldsymbol{P}$
    Anpassung $\boldsymbol{Q}$
    berechne SSE oder $RMSE$;
end while

2.8.3.1 Lösungsmöglichkeit über partielle Ableitungen

Das Minimierungsproblem in Gleichung 2.33 kann mittels partieller Ableitungen gelöst werden mit dem Ziel, ein lokales Minimum zu finden. Diese Verfahren wird auch als Gradientenabstiegsverfahren bezeichnet. Mittels der partiellen Ableitungen können die Gradienten berechnet werden. Durch die Gradienten (im zweidimensionalen Raum geben Gradienten die Steigung in einem Punkt an, in einem Hyperraum wird die „Steigung“ als Gradient bezeichnet) kann die Richtung bestimmt werden, in die die Elemente der Nutzer- und Item-Matrizen angepasst werden müssen. Die Elemente der Nutzer- und Item-Matrizen werden dabei in Richtung des invertierten Gradienten angepasst, um so das Minimum anzunähern. Je weiter die Elemente vom Minimum entfernt sind, desto größer ist der Gradient und in umso größeren Schritten wird das Minimum angenähert. Zur Steuerung der Anpassung mittels Gradienten wird der Faktor η eingeführt. Dieser Faktor wird auch als Schrittweite bezeichnet. Ein praktikabler Wert ist beispielsweise $0,02$ (vgl. [TPNT08], S. 555, S.558).

Zur Bildung der Gradienten wird jede Fehlerfunktion ($e_{ij} \; \forall \; (i,j) \; \in \; \mathcal{R}$) partiell nach p_{ik} und partiell nach q_{jk} abgeleitet (s. wie Gleichung 2.37 und 2.38). Somit kann für jedes Element in $\boldsymbol{P}$ und $\boldsymbol{Q}$ der Gradient berechnet werden.

$$p_{ik}^{'} = p_{ik} + \eta \cdot (-\frac{\partial e_{ij}}{\partial p_{ik}}) = p_{ik} + \eta \cdot (r_{ij} - \hat{r}_{ij}) \cdot q_{jk} \quad \forall \, (i,j) \in \mathcal{R} \; \wedge \; k \in \{1, ..., f\} \tag{2.37}$$

$$q_{jk}^{'} = q_{jk} + \eta \cdot (-\frac{\partial e_{ij}}{\partial q_{jk}}) = q_{jk} + \eta \cdot (r_{ij} - \hat{r}_{ij}) \cdot p_{ik} \quad \forall \, (i,j) \in \mathcal{R} \; \wedge \; k \in \{1, ..., f\} \tag{2.38}$$

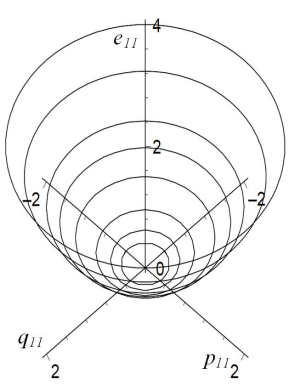
Die Bedeutung der Gradienten soll noch einmal grafisch veranschaulicht werden. Es wird nun angenommen, dass die Nutzer- und Item-Matrizen nur ein Feature besitzen. Dadurch ist beispielsweise der Fehler e_{11} nur von p_{11} und q_{11} abhängig. In Abbildung 2.15a wird für diesen Fall eine Fehlerfunktion skizziert. In Abbildung 2.15b werden die Fehlerwerte als Höhenlinien abgebildet und der Gradient als Pfeil dargestellt, welcher in Richtung des Minimums zeigt. Die Gradienten werden nun gebildet, indem zum einen die Fehlerfunktion nach p_{11} und zum anderen nach q_{11} partiell abgeleitet werden, welche auch in den Abbildungen 2.15c und 2.15d als blaue Tangente skizziert werden. Die Gradienten helfen nun dabei, eine Richtung vorzugeben, in die die Werte für p_{11} und q_{11} verändert werden sollen, um e_{11} zu minimieren.

Beispiel: Über mehrere Iterationen werden nun die Elemente in der Nutzer- und Item-Matrix mit $\eta = 0,02$ angepasst, um den Fehler SSE nach Gleichung 2.33 zu minimieren. Gleichung 2.39 zeigt die Matrizen nach 100 Iterationen sowie die zugehörige Nutzer-Item-Matrix. Die grau hinterlegten Elemente zeigen nun die vorhergesagten Präferenzen, die vorher unbekannt waren. Nach 100 Iterationen liegen die Fehler SSE bei etwa $0,016$ und $RMSE$ bei $0,049$ und somit beide nahe bei 0. Der geringe Fehler gibt an, dass die bekannten Elemente in der Nutzer-Item-Matrix durch die einzelnen Nutzer- und Item-Matrizen gut angenähert werden konnten.

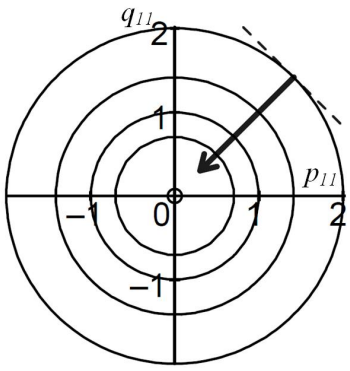
$$\boldsymbol{P} \approx \begin{pmatrix} 1,38 & 1,18 \\ 1,06 & 1,02 \\ -0,83 & 2,1 \\ -0,58 & 1,72 \\ -0,64 & 1,65 \end{pmatrix} \text{ und } \boldsymbol{Q} \approx \begin{pmatrix} 2,41 & 1,41 \\ 1,83 & 1,24 \\ 2,13 & 3,88 \\ -0,99 & 2 \end{pmatrix}, \text{ sodass } \hat{\boldsymbol{R}} \approx \begin{pmatrix} 5 & 4 & 7,53 & 1 \\ 4 & 3,22 & 6,23 & 1 \\ 0,95 & 1,08 & 6,37 & 5,01 \\ 1,02 & 1,07 & 5,44 & 4,02 \\ 0,78 & 0,88 & 5,06 & 3,94 \end{pmatrix} \tag{2.39}$$

2.8.3.2 Lösungsmöglichkeit über alternierende Fixierung

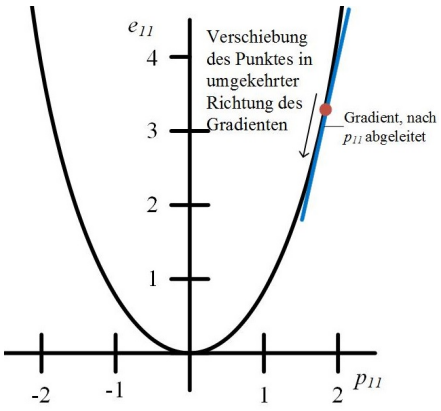
Die erläuterte Lösungsmöglichkeit in Abschnitt 2.8.3.1 bedingt, dass das Optimierungsproblem konvex. Dadurch kann sichergestellt werden, dass das gefundene lokale Minimum gleichzeitig global ist. Das Optimierungsproblem in den Gleichungen 2.33 und 2.35 müssen dagegen nicht konvex sein. Eine Alternative zum Gradientenabstiegsverfahren ist, dass alternierend $\boldsymbol{P}$ und $\boldsymbol{Q}$ festgesetzt werden und dann die unbekannten Elemente mittels der Methode der kleinsten Quadrate ermittelt werden. Durch das Festsetzen einer Matrix auf feste Werte wird das Optimierungsproblem quadratisch. (vgl. [KBV09], S. 33).



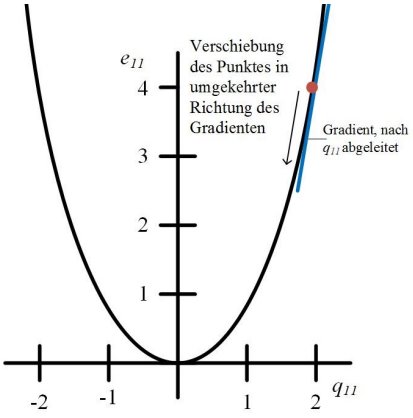
(a) Dreidimensionale Darstellung der Fehlerfunktion.



(b) Fehlerwerte werden als Höhenlinien dargestellt.



(c) Darstellung der partiellen Ableitung nach p_{11}.



(d) Darstellung der partiellen Ableitung nach q_{11}.

Abbildung 2.15: Geometrische Interpretation der partiellen Ableitungen. (vgl. [Neu05], S. 3).

2.8.4 Regularisierung

Wird das Optimierungsproblem nach den Gleichungen 2.33 und 2.35 gelöst, ist es möglich, dass große Werte in den Nutzer- und Item-Matrizen auftreten können. Dadurch können diese Matrizen zwar die Trainingsmenge gut oder exakt abbilden, aber es besteht dadurch die Gefahr, dass die Vorhersagen nicht adäquat sind. Dieses Problem wird auch als Overfitting bezeichnet. Große Werte in den Nutzer- und Item-Matrizen sollen vermieden werden. Deswegen wird das grundlegende Modell der Matrix-Faktorisierung regularisiert (vgl. [TPNT08], S. 555) und [KBV09], S. 32 f.)

Durch die Regularisierung wird das Optimierungsproblem um das Quadrat der euklidischen Norm der Zeilenvektoren der Nutzer- und Item-Matrizen (s. Gleichungen 2.40 und 2.41) erweitert. Zusätzlich werden die regularisierten Terme mit dem Faktor λ multipliziert, um den Einfluss der Regularisierung steuern zu können. Die neue Fehlerfunktion ist in Gleichung 2.42 aufgestellt. Das neue Optimierungsproblem, welches Regularisierung berücksichtigt, findet sich in Gleichung 2.43 (vgl. [TPNT08], S. 555) und [KBV09], S. 32 f.)

$$p_{i_{regularisiert}} = \|\vec{p_i}\|^2 = \vec{p_i} \cdot \vec{p_i}^T \tag{2.40}$$

$$q_{j_{regularisiert}} = \|\vec{q_j}\|^2 = \vec{q_j} \cdot \vec{q_j}^T \tag{2.41}$$

$$e_{ij_{regularisiert}} = \frac{1}{2}(r_{ij} - \hat{r}_{ij})^2 + \lambda(p_{i_{regularisiert}} + q_{j_{regularisiert}}) \quad \forall\,(i,j) \in \mathcal{R} \tag{2.42}$$

$$SSE_{regularisiert} = \sum_{(i,j)\in\mathcal{R}} \left(e_{ij_{regularisiert}}\right) \rightarrow min \tag{2.43}$$

Im Rahmen des Gradientenabstiegsverfahrens ändern sich nun auch die Gradienten. Die Anpassungen erfolgen nun wie in den Gleichungen 2.44 und 2.45 angegeben.

$$p_{ik}^{'} = p_{ik} + \eta \cdot (-\frac{\partial e_{ij_{regularisiert}}}{\partial p_{ik}}) = p_{ik} + \eta \cdot ((r_{ij} - \hat{r}_{ij}) \cdot q_{jk} - \lambda \cdot p_{ik}) \quad \forall\,(i,j) \in \mathcal{R} \;\wedge\; k \in \{1,...,f\} \tag{2.44}$$

$$q_{jk}^{'} = q_{jk} + \eta \cdot (-\frac{\partial e_{ij_{regularisiert}}}{\partial q_{jk}}) = q_{jk} + \eta \cdot ((r_{ij} - \hat{r}_{ij}) \cdot p_{ik} - \lambda \cdot q_{jk}) \quad \forall\,(i,j) \in \mathcal{R} \;\wedge\; k \in \{1,...,f\} \tag{2.45}$$

Beispiel: In Gleichung 2.46 sind die Nutzer- und Item-Matrizen sowie die vorhergesagte Nutzer-Item-Matrix unter Berücksichtigung der Regularisierung aufgestellt. Die vorhergesagten Bewertungen und die bekannten Bewertungen liegen nahe beieinander. Unterschiede zu Gleichung 2.39 gibt es bzgl. der Vorhersage der unbekannten Bewertungen. In Gleichung 2.46 sind diese kleiner als in Gleichung 2.39.

$$\boldsymbol{P} \approx \begin{pmatrix} 1,54 & 1,2 \\ 1,17 & 1,03 \\ -0,76 & 2,07 \\ -0,54 & 1,69 \\ -0,51 & 1,72 \end{pmatrix} \text{ und } \boldsymbol{Q} \approx \begin{pmatrix} 2,22 & 1,3 \\ 1,7 & 1,1 \\ 1,66 & 3,38 \\ -0,93 & 2,04 \end{pmatrix}, \text{ sodass } \hat{\boldsymbol{R}} \approx \begin{pmatrix} 4,97 & 3,93 & 6,6 & 1,01 \\ 3,93 & 3,12 & 5,42 & 1 \\ 1 & 1 & 5,74 & 4,94 \\ 1 & 0,95 & 4,82 & 3,95 \\ 1,08 & 1,02 & 4,95 & 3,98 \end{pmatrix} \tag{2.46}$$

Tabelle 2.4 gibt die Entwicklung der Fehlerwerte (sowohl regularisiert als auch nicht regularisiert) in Abhängigkeit der Iterationenanzahl an. Nach 100 Iterationen liegen die Fehler SSE und $RMSE$ nahe bei 0, der Fehler $SSE_{regularisiert}$ ist dagegen noch größer 1.

2.8.5 Weitere MF-Methoden und -Techniken

Das grundlegende Modell zur Matrix-Faktorisierung sowie die wichtige Erweiterung der Regularisierung sind nun vorgestellt worden. In der Literatur existieren noch zahlreiche Erweiterungen, die

Iteration Fehler	0	1	2	3	4	10	20	50	100
SSE	422,125	27,76	19,25	16,49	15,05	9,61	2,11	0,13	0,02
$RMSE$	8,06	2,07	1,72	1,59	1,52	1,22	0,57	0,14	0,05
$SSE_{regularisiert}$	424,97	28,89	20,34	17,59	16,16	10,8	3,35	1,36	1,26

Tabelle 2.4: Entwicklung der Fehler in Abhängigkeit der Iterationenanzahl.

sich hinsichtlich der Qualität der Lösungen als auch hinsichtlich der Performanz (z. B. Anzahl der Iterationen bis zur Erreichung des Minimums) unterscheiden. Einige sollen nun vorgestellt werden.

Verwendung von Konstanten: Die Verwendung von Konstanten in den Nutzer- und Item-Matrizen kann die Performanz der regularisierten Matrix-Faktorisierung erhöhen. Dazu wird die erste Spalte der Nutzer-Matrix und die zweite Spalte der Item-Matrix bei der Initialisierung auf 1 gesetzt. AUch von einer Anpassung während der Laufzeit des Algorithmus wird abgesehen. (vgl. [TPNT08], S. 555)

Positive Nutzer- und Item-Matrizen: Eine weitere Technik ist, in den Nutzer- und Item-Matrizen nur positive Werte zuzulassen. D. h., sobald die Anpassung gemäß Gleichungen 2.44 und 2.45 zu einem negativen Element führt, wird das Element auf 0 gesetzt. Diese Erweiterung hat auf die Performanz keinen Einfluss, kann allerdings bessere Erklärungsansätze für die Features bieten. (vgl. [TPNT08], S. 556)

Momentum-Methode: Diese Methode berücksichtigt die Veränderungen vorangegangener Anpassungen. Dadurch wird die Anpassung gemäß den Gleichungen 2.44 und 2.45 derart erweitert, dass die Veränderung der vorherigen Anpassung multipliziert mit einem sog. Momentum-Faktor hinzuaddiert wird. (vgl. [TPNT08], S. 556)

Trends (Bias): Die Beziehungen zwischen Nutzern und Items können unter Umständen nicht ausreichen, um geeignete Empfehlungen geben zu können. Dies ist dann der Fall, wenn hinsichtlich der Nutzer oder der Items bestimmte Trends zu erkennen sind.

Diese Trends können daran erkannt werden, dass bestimmte Items überdurchschnittlich gut bewertet werden (z. B. Items werden gehypt) oder dass bestimmte Nutzer besonders kritisch bewerten. Diese Trends können dann bei der Vorhersage berücksichtigt werden, indem diese korrigiert werden. (vgl. [KBV09], S.33)

Weitere Eingabedaten: Recommender-System können keine Empfehlungen geben, wenn noch kein auswertbares Nutzerverhalten beobachtet werden konnte. Dieses Problem tritt dann auf, wenn sich neue Nutzer registrieren (sog. „Cold start“-Problem)

Um dieses Problem zu lösen, können weitere Daten (implizites oder explizites Feedback) hinzugezogen werden. Beispielsweise kann der Nutzer darum gebeten werden, bestimmte Präferenzen anzugeben. Eine weitere Möglichkeit besteht durch das Hinzuziehen von z. B. demografischen Daten. Das Optimierungsproblem wird um „Dummy“-Attribute erweitert, die angeben, ob ein Nutzer zu einer bestimmten Attribut-Gruppe (z. B. Altersgruppe) gehört. (vgl. [KBV09], S.33 f.)

Zeitbezug: Time-Drifting bedeutet, dass abgegebene Bewertungen der Nutzer mit der Zeit ihre Bedeutung sowie dass Items an Aktualität verlieren können. Um einen Zeitbezug im Optimierungs-

problem herzustellen, können länger in der Vergangenheit zurückliegende Bewertungen geringer gewichtet und somit weniger stark berücksichtigt werden. (vgl. [KBV09], S.34)

Konfidenzniveaus: Nicht immer kann davon ausgegangen werden, dass Bewertungen tatsächlich ohne äußere Beeinflussungen getätigt werden. Beispielsweise können Präferenzen stark von Marketing-Strategien für bestimmte Items abhängen. Somit dürfen nicht alle von den Nutzern abgegebenen Präferenzen gleichermaßen vertraut werden.

Um dieses Problem zu entgegnen, werden Präferenzen gewichtet. Das Nutzerverhalten wird weiter beobachtet, um herauszufinden, wie stark sich Nutzer mit Items auseinandersetzen oder wie oft Nutzer bestimmte Items erwerben. Gibt es z. B. eine starke Auseinandersetzung mit dem Item, kann eher von einer größeren Präferenz ausgegangen werden. (vgl. [KBV09], S.34 f.)

2.8.6 Zusammenfassung

Matrix-Faktorisierung ist ein mathematisches Verfahren, welches gerne für das Collaborative Filtering herangezogen wird. Aus gegebenen Präferenzen der Nutzer zu bestimmten Items sollen neue Präferenzen vorhergesagt werden, um Empfehlungen geben zu können. Dazu wird die Präferenzmatrix in eine Nutzer- und eine Item-Matrix faktorisiert. Zur Faktorisierung kann das Gradientenabstiegsverfahren genutzt werden. Ziel ist es, den Fehler zwischen vorhergesagten und bekannten Präferenzen zu minimieren. Eine wichtige Erweiterung ist die Einbeziehung der Regularisierung, um ein Overfittung zu vermeiden und dadurch bessere Vorhersagen treffen zu können. Weiterhin gibt es weitere diverse Erweiterungen, die sich in qualitativer Hinsicht als auch hinsichtlich der Performanz unterscheiden.

2.9 Inkrementelle / Online / Stream-based RecSys-Algorithmen

Recommender-Systeme (RS) sind ein wichtiger Bestandteil des kommerziellen Internets. Online-Versandhändler wie Amazon, Online-Videothenken wie Netflix und Webportale für Musik wie Laut.fm nutzen RS um ihren Kunden aus ihrer großen Auswahl an Produkten, Filmen oder Musiktiteln diejenigen Elemente anzubieten, die den „Präferenzen“ des Nutzers am ähnlichsten sind. Dieses soll den Effekt der „Informationsüberflutung“ [BKWG10, S. 1] beim Nutzer verringern und sie bei der Auswahl- bzw. der Kaufentscheidung unterstützen [TPNT09, S. 623]. Den Stellenwert von einem leistungsfähigen RS erkannte die Online-Videothek Netflix und schrieb 2006 den „Netflix Prize“ über eine Millionen Dollar aus. Ziel war es, die Genauigkeit der Empfehlungen um 10 % zu steigern [MDA+11, S. 23].

Es gibt verschiedene Ansätze um RS zu Klassifizieren: Es gibt nach Takács et al. [TPNT09, S. 624] zwei grundlegende Strategien zur Erstellung von Empfehlungen: *Collaborative-Filtering* (CF) und *Content-Based-Filtering*. Takács et al. nutzen die Datengrundlage als Eigenschaft zur Klassifikation. Andere Autoren wie Burke [Bur02, S. 2] unterteilen RS in fünf verschiedene Typen. Auch hybride Strategien, die eine Mischform zwischen Collaborative- und Conent-Based-Filtering darstellen, sind in der Literatur zu finden [Kla09, S. 68]. Diese Ausarbeitung beschränkt sich auf *CF-Algorithmen*.

Beim CF wird nicht, wie bei dem Content-Based-Filtering auf Basis von einem vorhandenen Nutzer oder von Elementinformationen eine Empfehlung ausgesprochen, sondern auf Grundlage von Informationen, die andere Nutzer durch ihr aktives oder passives Nutzungsverhalten erzeugen. Es gibt somit eine Nutzer-zu-Nutzer Korrelation. Durch Algorithmen werden fehlende Bewertungen möglichst

genau angenähert. Eine Möglichkeit die Korrelation zwischen verschiedenen Nutzern, Elementen und Empfehlungen darzustellen ist eine Nutzer-Elemente-Matrix:

In einer Matrix $R = (r_{u,i})$ mit $i = 1..n$ und $j = 1..m$ sind alle Nutzer und Elemente dargestellt. Dabei ist n die Anzahl aller Nutzer und m die Anzahl aller Elemente. Der Wert $r_{u,i}$ ist dabei die die Bewertung des Empfehlungselementes j durch den Nutzer i.

Die Bewertung kann beispielsweise auf einer binären Nominalskala oder auf einer Ordinalskala erfolgen. Ein Beispiel für eine RS mit einer Ordinalskala ist die 5-Sterne-Bewertung von dem Online-Versandhändler Amazon. Auf diesen Daten können verschiedene Algorithmen angewendet werden, um fehlende Bewertungen zu ermitteln. Diese Matrix ist in der Regel eine sehr dünn besetzte Matrix, da nicht jeder Nutzer jedes Element bewertet hat [VJG14, S. 1] [Kla09, S. 63].

2.9.1 Herausforderung von Online Recommender Systemen

In klassischen CF Ansätzen wird das genutzte Modell periodisch neu berechnet. Dieses ist jedoch sehr kostenintensiv und es ist nur eine Momentaufnahme der Empfehlungen aus der Vergangenheit: Jedes mal wenn neue Elemente, Nutzer oder Bewertungen in das System einfließen, verändert sich die Datenbasis und das System muss auf diese Veränderung reagieren. Dieses bedeutet, dass das neue Modell neu angelernt werden muss bzw. die neuen Daten in die Berechnung mit einfließen müssen [WHZL13, S. 1].

Dieses geschieht bei „real-world“ [WHZL13, S. 237] RS in der Regel sehr häufig und ist nicht praktikabel. Wenn das Modell in einer Lernphase zuerst erzeugt wird, nennt man dieses auch eine *Batch Learing*-Strategie. Das Eintreffen von neuen Informationen kann jedoch nach Vinagre et. al. [VJG14, S. 2] auch als Datenstrom angesehen werden. Techniken, die mit solchen Datenströmen umgehen werden auch als *Online-Learing*-Strategien bezeichnet. Bei Online-Learing-Strategien ist es nicht nötig eine vollständige Datenbasis bereitzustellen. Neue Informationen wirken sich dynamisch auf das Modell aus. Vinagre et al. [VJG14, S. 2] bezeichnen Systeme, die Online-Learing-Strategien nutzen als „Incremental Learning“ und Wang et. al [WHZL13, S. 237] bezeichnen diese als On-the-Fly- oder Online RS. Einen Kriterienkatalog, welche Eigenschaften ein System haben muss, dass mit großen Datenströmen agieren soll, geben Domingos und Hulten [HD01] (Übersetzung durch den Autor):

- Das System benötigt nur eine kurze und konstante Zeitspanne um einen Datensatz zu verarbeiten. Andernfalls würde es zwangsläufig in Verzug geraten.

- Das System darf nur einen festgelegten Bereich des Hauptspeichers belegen. Unabhängig von der Anzahl der bearbeiteten Datzensätze.

- Das System muss in der Lage sein, nach einer Betrachtung der Daten, ein Modell zu erzeugen, da es wahrscheinlich aufgrund von Zeitproblemen nicht erneut alte Daten betrachten kann. Zudem kann es sein, dass der Zugriff auf die Daten in der Zukunft nicht mehr möglich ist.

- Es muss immer ein einsatzfähiges Modell existieren. Dieses darf nicht erst nach dem vollständigen abarbeiten der Daten zur Verfügung stehen, da immer neue Daten zur Verfügung stehen wird es niemals einen Endzustand erreichen.

- Das System muss die oben genannten Kriterien erfüllen und möglichst ein Modell erzeugen, dass vergleichbare Ergebnisse liefert, wie ein Modell, das durch einen gewöhnlichen Algorithmus erzeugt wurde.
- Das System sollte auf Phänomene wie einem Concept-Drift reagieren können und gleichzeitig immer aktuell sein. Zudem sollten relevante Informationen aus der Vergangenheit mit in das Modell einfließen.

In der Literatur finden sich einige Ansätze um mit diesen Problemen umzugehen. Im folgenden wird zuerst das Prinzip der *Matrix Faktorisierung* (MF) beschrieben, da es das grundlegende Vorgehen für fast alle Änsätze darstellt. Im zweiten Schritt werden ausgewählte Ansätze beschrieben.

2.9.2 Inkrementelle Collaborate Filtering Algorithmen für Datenströme

Algorithmen für CF können in zwei Arten unterteilt werden [RRSK11]: *Nachbarschafts-basierte* (neighborhood approaches) und *Modell-basierte* (latent factor models) Algorithmen. Nachbarschafts-basierte Algorithmen nutzen als Datenbasis die vorhandenen Nutzer- oder Elementebewertungen und errechnen eine Ähnlichkeit zwischen diesen. Beispielsweise gibt es die Pearson-Korrelation, Kosinus-Ähnlichkeit oder den Euklidschen Abstand zur Bestimmung von Ähnlichkeiten [Kla09, S. 72ff] [PRPT05, S. 2] zwischen zwei Vektoren. Diese Ähnlichkeiten können in einer Ähnlichkeitsmatrix A dargestellt werden wobei der Eintrag A_{u_i,u_j} die Ähnlichkeit zwischen den Nutzern u_i und u_j repräsentiert.

Modell-basierte Algorithmen erzeugen zuerst ein Modell, um die Komplexität der Aufgabe zu verringern und errechnen mithilfe des Modells die fehlenden Bewertungen. Eine Möglichkeit die Komplexität einer Nutzer-Elemente-Matrix zu verringern ist die MF. Der Begriff MF ist ein Oberbegriff für eine Reihe von mathematischen Techniken, in dem eine Matrix zerlegt wird. Aus diesen Zerlegungen kann die ursprüngliche Matrix in Annäherung wieder errechnet werden. Es gibt verschiedene Verfahren wie die Low Rank MF oder die Singular Value Decomposition. Durch die MF kann die Nutzer-Elemente-Matrix R durch die Matritzen P und Q dargestellt werden. Das Produkt aus P und Q ist eine Annäherung an die Matrix R (Gleichung 2.47):

$$R \approx PQ, \tag{2.47}$$

wobei R eine $n \times m$, P eine $n \times k$ und Q eine $k \times m$ Matrix ist. Die Matrix P ist die Nutzer-Matrix und die Matrix Q ist die Elemente-Matrix. Die Matritzen werden auch als Nutzerraum bzw. Elementeraum bezeichnet. Der Zeilenvektor P_u der Matrix P stellt den Nutzer u dar. Die Variable k gibt die Anzahl der *Features* wieder. Der Spaltenvektor Q_i der Matrix Q stellt das Element i dar. Durch die Multiplikation dieser zwei Vektoren kann die angenäherte Bewertung $\widehat{r}_{u,i}$ des Nutzers u für das entsprechende Element i errechnet werden. Die Variable k stellt dabei die Anzahl der Komponenten des Vektors dar. Durch die Umformung der Ausgangsmatrix kann die Anzahl an Komponenten, die nötig werden um R zu beschreiben von $n * m$ auf $n * k + k * m$ reduziert werden [TPNT09, S. 628].

Die Annäherung der Matritzen P und Q an R kann als Optimierungsproblem angesehen werden. Durch Methoden wie *Alternating Least Squares* oder *Stochastic Gradient Desent* kann die Matrix optimiert werden [VJG14, S. 3]. Ziel der Annäherung ist, dass die angenäherte Bewertung $\widehat{r}_{u,i}$ möglichst der realen Bewertung $r_{u,i}$ entspricht. Um dieses zu evaluieren kann die Distanz zwischen

der Annäherung und dem tatsächlichen Ergebnis genommen werden. In Gleichung 2.48 ist dieses Verfahren formal für alle $r_{u,i}$ und $\widehat{r}_{u,i}$ beschrieben.

$$RMSE = \sqrt{\frac{1}{n * m} \sum_{u,i} (\widehat{r}_{u,i} - r_{u,i})^2} \tag{2.48}$$

Es ist der *Root Mean Square Error* und wird genutzt, um die Güte eines Algorithmus zu evaluieren. Ziel ist es den RMSE zu minimieren.

In Tabelle 2.5 sind verschiedene, ausgewählte Arbeiten aufgelistet, die sich mit skalierbaren und inkrementellen Algorithmen für Online RS auseinandersetzen. Die Auflistung erhebt keinen Anspruch auf Vollständigkeit. Es wurde darauf geachtet, dass möglichst verschiedene Ansätze in dieser Arbeit beschrieben werden. So nutzt beispielsweise der Algorithmus von Papagelis et al. [PRPT05] keine MF, sondern eine Nutzer-Ähnlichkeitsmatrix. Andere Autoren wie z. B. Wang et al. [WHZL13] nutzen eine Kombination aus MF und Nutzer-Ähnlichkeismatrix. Neben diesen Arbeiten gibt es noch weitere Arbeiten, die sich mit der Problemstellung von Online RS und der Skalierbarkeit auseinander setzen. Diaz-Aviles et. al. [DADSTN12] beschreiben einen Algorithmus für Social Streams (z. B. der Mikroblogging-Dienst Twitter) und Miranda et al. [MJ09] nutzen eine Elemente-Ähnlichkeitsmatrix. Die Tabelle gibt einen Überblick über die genutzte Methode, die Besonderheiten des Algorithmus und die Integration von neuen Nutzern, neuen Elementen, sowie das Aktualisieren von Bewertungen, die durch einen existierenden Nutzer abgebenen wurden.

2.9.2.1 Ähnlichkeitsmatrix

Eine frühe Arbeit aus dem Bereich des inkrementellen CF ist der Algorithmus von Papagelis et al. [PRPT05]. Sie beschreiben es als eine Methode, die skalierbar ist und es ermöglicht eine Aktualisierung der Nutzer-Ähnlichkeitsmatrix zu erzeugen, ohne den gesamten Datensatz erneut als Grundlage für die Berechnung miteinzubeziehen. Als Berechnungsmethode wird der Pearson-Korrelationskoeffizienten, zur Bestimmung der Ähnlichkeit von zwei Nutzer, genutzt. Dieser ist in Gleichung 2.49 zu sehen und wird im Folgenden näher beschrieben.

Sie nutzen eine Nutzer-Elemente-Matrix und definieren einen Nutzervektor als eine Zeile der Matrix. Eine Zeile ist ein n-dimensionaler Vektor, wobei n die Anzahl von Elementen ist. Dieser Vektor ist sehr dünn besetzt, da ein Nutzer in der Regel nur wenige Elemente aus n bewertet hat. Aus diesem Grund definieren sie die Teilmenge $I' = \{i_x : x1, 2, ..., n' \; und \; n' <= n\}$ auf der sie die Ähnlichkeitsberechnung für zwei Nutzer bzw. zwei Vektor ausführen. Die Anzahl der bewerteten Elemente durch die zwei Nutzer ist n'. Diese Teilmenge enthält nur Bewertungen von Elementen, die beide Nutzer schon bewertet haben. Zudem ist r_{u_x,i_h} die Bewertung des Elements i_h durch den Nutzer u_x sowie $r_{\bar{u_x}}$ und $r_{\bar{u_y}}$ sind die durchschnittlichen Bewertungen der Nutzer u_x bzw. u_y.

$$sim(u_x, u_y) = \frac{\sum_{h=1}^{n'} (r_{u_x,i_h} - r_{\bar{u_x}})(r_{u_y,i_h} - r_{\bar{u_y}})}{\sqrt{\sum_{h=1}^{n'} (r_{u_x,i_h} - r_{\bar{u_x}})^2} \sqrt{\sum_{h=1}^{n'} (r_{u_y,i_h} - r_{\bar{u_y}})^2}} \tag{2.49}$$

Um das Prinzip der Inkrementellen CF Methode deutlicher hervorzustellen zeigt Gleichung 2.50 eine vereinfachte Form der Gleichung 2.49.

Autoren	Verfahren	Neue Nutzer/Elemente/Aktualisierung der existierenden Bewertungen	Besonderheit
[SKKR02]	MF, Fold-In	Ja/Ja/Nein	Aktualisiertes Modell wird mit jedem Eintrag ungenauer.
[PRPT05]	Nutzer-Ähnlichkeitsmatrix mit Pearsonkorrelationskoeffizent	Nein/Nein/Ja	Keine Annäherung durch Modell
[TPNT09]	MF	Ja/Nein/Ja	Verkürzte Trainingsphase durch erhalt der Elementematrix
[WHZL13]	MF und Nutzer-Ähnlichkeitsmatrix	Ja/Ja/Ja	Kombination von MF und Nutzer-Ähnlichkeitsmatrix
[VJG14]	MF	Ja/Ja/Ja	Methode für Positive-Only-Feedback-Systeme
[MJ09]	Nutzer bzw. Element-Ähnlichkeitsmatrix	Ja/Ja/Ja	Verwendung einer Elemente-Ähnlichkeitsmatrix
[DADSTM12]	MF	Ja/Ja/Ja	Positive-Only-Feedback in Social Streams (z. B. Twitter)

Tabelle 2.5: Überblick über verschiedene Algorithmen für Online RS.

$$A = \frac{B}{\sqrt{C}\sqrt{D}} \Rightarrow A = sim(u_x, u_y), B = \sum_{h=1}^{n'}(r_{u_x,i_h} - \bar{r_{u_x}})(r_{u_y,i_h} - \bar{r_{u_y}}), C = \sum_{h=1}^{n'}(r_{u_x,i_h} - \bar{r_{u_x}})^2,$$
$$D = \sum_{h=1}^{n'}(r_{u_y,i_h}) - \bar{r_{u_y}})^2 \tag{2.50}$$

Sobald es eine Aktualisierung von Elementen, Nutzern oder Bewertungen gibt, müssen die Faktoren B, C und D erneut berechnet werden. Als Resultat entstehen die Faktoren B', C' und D' mit einem neuen Korrelationseffizienten A'. Dieses ist in Gleichung 2.51 abgebildet. Ebenso zeigt Gleichung 2.51, wie mithilfe der Inkremente e, f und g aus den ursprünglichen Faktoren die neuen Faktoren errechnet werden können. Die Berechnung kann für jeden Faktor autonom erfolgen.

$$A' = \frac{B'}{\sqrt{C'}\sqrt{D'}} \Rightarrow A' = \frac{B+e}{\sqrt{C+f}\sqrt{D+g}},\ B' = B + e,\ C' = C + f,\ D' = D + g \tag{2.51}$$

Durch diesen Ansatz ist es möglich, teure Vektoroperationen durch Skalaroperationen zu ersetzen und die Berechnung von großen Nutzer-Elemente-Matritzen zu beschleunigen [PRPT05, S. 9]. Sie

beweisen, dass mithilfe der Inkremente die neue Ähnlichkeitsmatrix berechnet werden kann. Auf eine detailierte Berechnung der Inkremente sei an dieser Stelle verzichtet. Vorraussetzung ist jedoch, dass die Werte A, B, C für jedes Nutzer-Paar, die durchschnittliche Bewertung eines Nutzers und die Anzahl der Bewertungen eines Nutzers gespeichert wird.

2.9.2.2 Optimierung der Modellbildung

Eine spezielle Technik der MF ist die *Singular Value Decomposition*: Gegeben ist eine $n \times m$ Matrix A, dann ist die Singular Value Decomposition $SVD(A)$ in Gleichung 2.52 definiert.

$$SVD(A) = U \times S \times V^T \tag{2.52}$$

Diese Form der MF unterscheidet sich in der Art und Weise, wie die angenäherte Bewertung mithilfe des Nutzer- Elementeraums berechnet wird. Der Vorteil der Singular Value Decomposition ist, dass es nach Sarwar et al. [SKKR02] die beste Annäherung an die original Matrix A ist. Zudem wird jeder Nutzer und jedes Element durch einen entsprechenden, unabhängigen Eigenvektor repräsentiert [SKKR02, S. 2].

Auf dieser Grundlage nutzen Sarwar et al. [SKKR02, S. 2] die *Fold-In Methode*, um neue Nutzer in das reduzierte Modell einzufügen. Es ist eine Optimierung der Modellberechnung. Der erste Schritt erfolgt wie oben beschrieben: Es wird aus einer Nutzer-Elemente Matrix ein Modell berechnet. Damit dieses Modell nicht bei jedem Nutzer erneut berechnet werden muss, wird mit der Fold-In Methode ein neuer Nutzer in das Modell eingefügt. Dieses geschieht in Schritt zwei und drei: Schritt zwei wird eine Projektion des neuen Eintrages berechnet und im dritten Schritt wird der Nutzerraum um den neu berechneten Eintrag erweitert. Durch diese Methode ist es nicht mehr nötig das Modell erneut zu berechnen, sondern es wird inkrementell erweitert. Der Nachteil ist, dass durch die Erweiterung das Modell mit jedem Datensatz ungenauer wird [TS J, S. 3].

2.9.2.3 Verkürzte Matrix Faktorisierung

Takács et al. [TPNT09] stellen in ihrer Arbeit eine Reihe von Techniken vor, mit denen sie die Genauigkeit und die Laufzeit von CF-Algorithmen verbessern. Sie nutzen eine Form der MF, um die Anzahl der Komponenten zu reduzieren. Durch ein inkrementelles Gradientenverfahren lösen sie das Optimierungsproblem und füllen die Matritzen P und Q. Das genaue Verfahren, wie die Matritzen gefüllt werden kann an dieser Stelle vernachlässigt werden. Durch Multiplikation von Spalten- und Reihenvektoren der Matritzen kann die Bewertung von einem Nutzer für ein Element errechnet werden. Sie nutzen eine Trainingsphase, um aus den bekannten Bewertungen ein Modell zu erzeugen. Mit diesem kann in der zweiten Phase eine Empfehlung für den Nutzer geben werden. Dieses ist eine typische Batch-Learning-Strategie.

Die Trainingsphase hat jedoch Nachteile: Während der Berechnung der optimalen Matritzen P und Q können sich Komponenten verändern. Takács et. al. beschreiben zwei Wege, um dieses Problem zu lösen: Der erste Weg ist, dass die berechneten Werte direkt nach jedem Nutzer gegengeprüft werden. Der zweite Weg ist, die Erweiterung des Algorithmus um eine zweite Trainingsphase, in der nicht das ganze Modell, sondern nur die Matrix P, der Nutzerraum, erneut berechnet wird. Die Matrix Q bleibt erhalten. Es werden alle Komponenten des Nutzerraumes neu, randomisiert initialisiert.

Diese Vorgehensweise, dass nicht die gesamte Trainingsphase absolviert werden muss, kann ebenso genutzt werden um neue Nutzer oder neue Bewertungen von existierenden Nutzer in das Modell mit einzugliedern. Es ist nicht nötig den gesamten Trainingsprozess zu starten, sondern es wird nur der Nutzerraum erneut berechnet. Sofern neue Elemente zur Verfügung stehen, muss jedoch die vollständige Trainingsphase durchlaufen werden. Ebenso kann der Fall eintreten, dass zu viele neue Bewertungen gegeben werden. Hier empfehlen sie, die erneute Berechnung von R durch den gesamten Algorithmus mit einer erneuten Initialisierung von P und Q [TPNT09, S. 632f].

Takács et. al. evaluieren diese Vorgehensweise gegen den Datensatz, der von der Online-Videothek Netflix im Zuge des Netflix Price veröffentlicht wurde, da dieser Datensatz die größte Herausfoderung für ihre Algorithmen bietet [TPNT09, S. 639]. Die Evaluation ergibt, dass der Algorithmus neue Nutzer händeln kann und es nur einen kleinen Unterschied zwischen dem Ergebnis und der Probe gibt. Gleiches gilt auch für neue Bewertungen von existierenden Nutzern [TPNT09, S. 649].

2.9.2.4 Online Multitasking Collaborative Filtering

Wang et al. [WHZL13] stellen ein Framework von Algorithmen vor, mit dem sie die Effizenz und Effektivität von Online RS verbessern. Sie bezeichnen Online Systeme auch als On-The-Fly RS, was verdeutlicht, dass sie nicht von einer statischen Datenbasis, sondern von einem dynamischen System ausgehen. In diesem System verändert sich die Anzahl von Nutzern, Elementen und Bewertungen. Sie nutzen MF und erzeugen somit zwei Matritzen P und Q, die den Nutzer- bzw. Elementeraum repräsentieren. Durch Minimierung des RMSE wird das Modell solange angepasst, bis es möglichst den Ausgangsdaten entspricht. Jede Zeile bzw. repräsentiert einen Nutzer bzw. ein Element. Durch die Multiplikation entsprechender Zeilen- bzw. Spalten-Vektoren kann die angenäherte Bewertung errechnet werden.

Die Besonderheit bei ihrem Ansatz ist, dass sie den Ansatz des CF mit einer Methode aus dem Bereich des *Online Multi-Task Learning* kombinieren: Wenn eine neue Bewertung durch einen existierenden Nutzer abgegeben wird, wird in „existierenden“ Online CF Vorgehensweisen nur der entsprechende Nutzer, der die Bewertung abgegeben hat, angepasst. Sie erweitern dieses Vorgehen, indem sie ähnliche Nutzer ebenso aktualisieren. Sie nutzen dafür eine Nutzer-Ähnlichkeitsmatrix A, in der der Eintrag $A_{i,j}$ die Ähnlichkeit zwischen den Nutzern i und j repräsentiert [WHZL13, S. 239].

Die Ähnlichkeitsmatrix kann auf drei verschiedene Arten mit Einträgen gefüllt werden: Es werden konstante Werte verwendet, die Einträge beruhen auf bekannten Informationen oder es wird aus der Nutzermatrix P die Kovarianzmatrix als A definiert und sequenziell aktualisiert. Ein Beispiel für bekannte Informationen ist z. B., wenn das Alter der Nutzer bekannt ist. Mithilfe des Alters kann eine Ähnlichkeit zwischen Nutzern ermittelt werden. Die Matrix könnte mit den Werten 0 und 1 gefüllt werden. Sofern die Nutzer in der gleichen Altersgruppe sind, wäre der Wert 1 und der entsprechende Nutzervektor würde ebenso aktualisiert werden. Bei einer 0 würde keine Aktualisierung des Nutzervektors erfolgen. Mit ihrem Algorithmus ist es möglich das Modell inkrementell um neue Nutzer und neue Elemente zu erweitern. Zudem ist eine Aktualisierung von Bewertungen vorgesehen.

In der Evaluation zeigt sich, dass der RMSE bei ihren Algorithmen niedriger ist als mit anderen „state-of-the-art“ [WHZL13, S. 237] Algorithmen. Sie nutzen als Vergleich den Online CF-Algorithmus, der durch Abernethy et. al. [ACLS07] entwickelt wurde [WHZL13, S. 244]. Jedoch verlängert sich die Berechnungszeit. Dieses liegt an der Berechnung der Ähnlichkeitsmatrix [WHZL13, S. 242].

2.9.2.5 Postitive-Only Feedback

Vinagre et al. [VJG14] stellen einen Algorithmus für eine inkrementelle MF von Positive-Only Feedback-Systemen vor. Positive-Only Feedback bedeutet, dass es keine Ordinalskala gibt auf der ein Element durch einen Nutzer bewertet wird, sondern es gibt nur die Möglichkeit, dass ein Nutzer ein Element als positiv bewertet hat. Ein Element kann den Zustand „true“ bzw. 1 oder „false“ bzw. 0 besitzen. Dabei können dem Zustand „false“ zwei Bedeutungen zugeteilt werden: Der Nutzer hat das Element als negativ bewertet oder der Nutzer hat das Element nicht bewertet. Vinagre et al. entscheiden sich dafür, dass der Zustand „false“ bedeutet, dass der Nutzer das Element noch nicht bewertet hat. Dieses kommt ihrer Meinung nach eher in realen RS vor. Ein Beispiel ist der Besuch einer Webseite oder das Aufrufen eines Links.

Sie nutzen wie Wang et al. [WHZL13] und Takács et. al. [TPNT09] MF. Als Ausgangspunkt nutzen sie den Batch Stochastic Gradient Descent Algorithmus von Funk [Fun06]. Der Algorithmus ist jedoch nicht auf Datenströme ausgelegt. Dies bedeutet, dass alle Informationen vorab bekannt sein müssen und die Daten sich über die Zeit nicht verändern. Durch zwei Modifikationen verändern sie den Algorithmus so, dass er für Datenströme genutzt werden kann: Die erste Veränderung betrifft die Häufigkeit der Iterationen über die Daten. Ihr Algorithmus muss nur einmal über jedes Datentupel, bestehend aus einem Nutzer und einem Element iterieren.

Es ist möglich mit ihrem Algorithmus neue Nutzer und neue Elemente in das Modell miteinzubeziehen. Das Modell muss nicht erst neu berechnet werden. Vereinfacht wird dieses dadurch, dass sie nur mit Positive-Only Datensätzen arbeiten. Die Aktualisierung von bestehenden Nutzern kann somit sehr effektiv vollzogen werden.

2.9.3 Zusammenfassung

Es gibt viele Arten, wie das Problem der Skalierbarkeit von Online RS gelöst werden kann. Neue Arbeiten nutzen häufig MF, um ein leistungsfähiges Modell zu erschaffen. Es wird dabei an verschiedenen Stellen des Prozesses optimiert. Vinagre et al. [VJG14] und Wang et. al. [WHZL13] nutzen einen Algorithmus, der das Modell inkrementell erweitert. Sie sind somit nicht mehr ausschließlich von einem optimalen Lernprozess abhängig und ihre Empfehlungen beruhen immer auf den aktuellen Daten. Aber auch inkrementell erweiterte Nutzer-Ähnlichkeitsmatritzen sind eine Strategie um neue Informationen zu verarbeiten. Dieses zeigen Sarwar et al. [SKKR02] in ihrer Arbeit. Eine weitere Möglichkeit ist die Modellbildung zu optimieren. Dieses ziehen Takács et al. [TPNT09] in betracht, in dem sie nur einen Teil des Modells neu berechnen. Wang et. al. zeigen, dass es auch hybride Lösungsansätze gibt, in denen sowohl MF als auch eine Nutzer-Ähnlichkeitsmatrix zum Einsatz kommt. Von den betrachteten Algorithmen scheint der Algorithmus von Vinagre et al. [VJG14] sehr vielversprechend für Datenströme, da sie das Modell dynamisch erweitern, keine weiteren Datenstrukturen benötigen um Empfehlungen abzugeben und neuen Nutzer, Elementen und Aktualisierungen von existierenden Nutzern berücksichtigen. StreamBasedRecSysAlgorithmen

3 Projektorganisation und -management

Im folgenden Kapitel wird das Projektmanagement und die Projektorganisation genauer erklärt. Dabei wird besonderen Fokus auf das verwendete Vorgehensmodell gelegt werden.

3.1 Vorgehensmodell Scrum

Scrum ist als ein Vorgehensmodell oder Rahmenwerk zu verstehen, das die Prinzipien der agilen Softwareentwicklung anwendet und das einen inkrementellen und iterativen Ansatz verfolgt. Frühzeitig und kontinuierlich können z. B. der aktuelle Status oder bestehende Probleme erfasst werden. Die Implementierung mit Scrum basiert auf den drei grundlegenden Prinzipien *Transparenz*, *Überprüfung* und *Anpassung* [SS13].

Zu Beginn der 1990er wurden agile Prozesse und Modelle als Alternativen zu konventionellen Softwareentwicklungsmethoden entwickelt. Dazu zählen unter anderem neben Scrum z. B. *Extreme Programming (XP)*, *Crystal* oder *Kanban* [San14]. Ursprünglich fanden die ersten Einführungen agiler Prozesse häufig dort statt, wo schnelle Reaktionen auf Probleme erforderlich waren und wo unter großem Druck riskante und wichtige Projekte durchgeführt werden mussten, bei denen herkömmliche Modelle wie das *V-Modell* oder das *Wasserfallmodell* zu keinem Erfolg führten [Glo13].

Im Jahr 2001 wurden im *Agilen Manifest* die Werte der agilen Software-Entwicklung festgehalten, die auch für Scrum ein wichtiges Fundament darstellen. Menschen und Interaktionen sind wichtiger als Prozesse und Werkzeuge, funktionierende Software höher priorisiert als umfassende Dokumentation, die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen und das Reagieren auf Veränderung hat einen höheren Stellenwert als das Befolgen eines fixen Plans. Dies bedeutet nicht, dass Werkzeuge, Dokumentation, Pläne etc. unwichtig sind [BBB$^+$01a].

Neben diesen Werten sind im Agilen Manifest klare Prinzipien formuliert worden. Höchste Priorität hat die frühe und kontinuierliche Auslieferung zufriedenstellender Software an den Kunden. Anforderungsänderungen sind jederzeit willkommen und können gut in den Prozess einbezogen werden. Regelmäßig und innerhalb möglichst kurzer Zeitspannen soll funktionierende Software geliefert werden. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams, die regelmäßig reflektieren, wie effektiver zusammengearbeitet werden kann [BBB$^+$01b].

In den folgenden Abschnitten werden die Scrum-Komponenten *Rollen*, *Aktivitäten* und *Artefakte* erklärt. Zudem werden exemplarisch unterstützende Werkzeuge vorgestellt. In Abschnitt 3.1.4 findet eine kritische Auseinandersetzung mit der Anwendung von Scrum in einer aus Studenten bestehenden Projektgruppe statt.

3.1.1 Rollen

Bei Scrum gibt es die Rollen *Product Owner*, *Scrum Master*, das *Entwicklungsteam* sowie externe Rollen (*Stakeholder*) (siehe Abbildung 3.1)

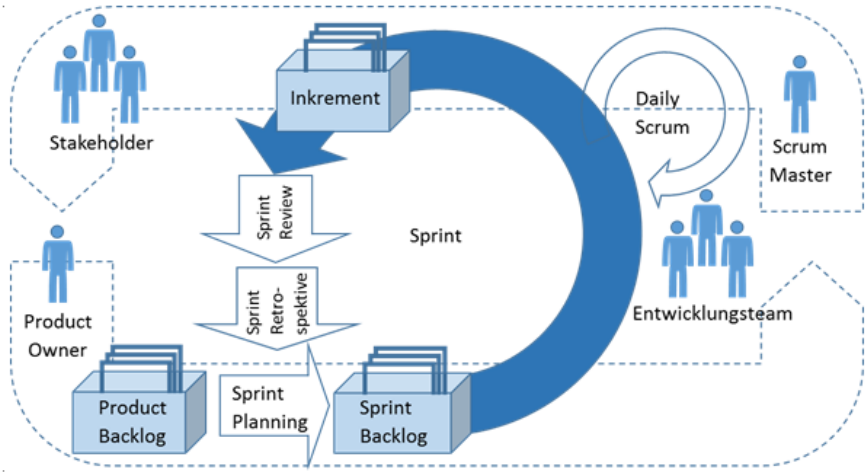


Abbildung 3.1: Rollenkonzept, Aktivitäten und Artefakte. Quelle: Eigene.

Der Product Owner ist dafür verantwortlich, dass das Entwicklungsteam die geforderten Funktionalitäten in der korrekten Reihenfolge erstellt. Er plant und lenkt die Entwicklung in ständiger Zusammenarbeit mit dem Team und arbeitet zudem kontinuierlich am *Product Backlog* (siehe Abschnitt 3.1.3), für das er als einzige Person verantwortlich ist.

Das Entwicklungsteam erstellt und liefert das eigentliche Produkt. Es organisiert sich selbst und managt die zu erledigende Arbeit, unter Einhaltung sämtlicher Regeln und Standards, in Eigenregie. Dadurch wird erhofft, dass die Gesamteffizienz optimiert wird [SS13]. Das *Was* wird vom Product Owner durch das Product Backlog festgelegt, während außer dem Entwicklungsteam keine Rolle darüber entscheidet, *wie* diese geforderten Funktionalitäten erstellt werden sollen [Glo13].

Die Hauptaufgabe des Scrum Masters besteht darin, den am Projekt Beteiligten das erforderliche Verständnis für Scrum zu vermitteln und für eine korrekte Durchführung zu sorgen [SS13]. Er hat allerdings keine weisende Funktion [Glo13]. Zudem kann er bei der Durchführung von Aktivitäten (siehe Abschnitt 3.1.2) helfen und hat diese zu organisieren. Für das Entwicklungsteam füllt er eine unterstützende und coachende Rolle aus. Er hat dafür zu sorgen, dass das Team die gestellten Anforderungen versteht. Er hilft dabei, Hindernisse und Probleme zu überwinden und muss gleichzeitig sicherstellen, dass alle Scrum-Regeln und -Praktiken eingehalten werden. Dazu zählt auch das Einhalten sämtlicher zeitlicher Beschränkungen, etwa im *Daily Scrum* (siehe Abschnitt 3.1.2) [SS13]. Der Scrum Master dokumentiert und teilt sämtliche Probleme und Hindernisse, wie etwa Blockaden (*Impediments*), die aufgehoben werden müssen, um nachfolgende Schritte angehen zu können.

Externe Rollen, auch Stakeholder, die nicht direkt an der Anwendung von Scrum beteiligt sind, sind der Kunde, der Anwender und das Management. Der Kunde ist der Auftraggeber, der die Produktentwicklung mit Scrum bezahlt. Ihm gegenüber hat der Product Owner die finanziellen Aufwendungen durch Ergebnisse zu rechtfertigen. Der Kunde ist derjenige, der zum Abschluss des Projektes das fertige Produkt verlangt. Der Anwender nutzt letztendlich das Produkt. Er kann gleichzeitig der Kunde sein, muss es aber nicht. Die Rolle des Managements besteht im Wesentlichen darin, die Grundlagen für eine Einführung von Scrum in einer Organisation zu schaffen.

3.1.2 Aktivitäten

Der Ablauf des Scrum-Prozesses ist möglichst einfach gestaltet (siehe Abbildung 3.1). Zu Beginn steht die Vision des Product Owners: Hier werden das Produkt beschrieben, Aufwand und mögliche Meilensteine abgeschätzt und vor allem geklärt, ob die Durchführung überhaupt machbar und sinnvoll ist. Falls sich der Product Owner dazu entschließt, ein Projekt mit Scrum anzugehen, beginnt der Prozess mit dem Formulieren, Sammeln und Priorisieren von Anforderungen im Product Backlog [Han10]. Alle Aktivitäten haben bei Scrum eine zeitliche Beschränkung. Der Zweck von den formalen Aktivitäten ist die kontinuierliche Schaffung von Transparenz und Überprüfung und damit Risikominimierung. Der *Sprint* bildet den Kern von Scrum und ist als Container für alle übrigen Aktivitäten zu sehen (siehe Abbildung 3.1) [SS13].

Sprints sind die Iterationszyklen, in denen das Entwicklungsteam das Produkt entwickelt. Nach jedem Sprint sollte ein potenziell nutzbares Produkt-Inkrement stehen, das theoretisch ausgeliefert werden könnte. Sprints haben innerhalb eines Projekts stets die gleiche Länge, höchstens jedoch 30 Tage. Dies soll die Komplexität begrenzen und Vorhersehbarkeit ermöglichen. Es ist nicht erlaubt, die Dauer eines Sprints zu verändern. Einzig wenn die Fortführung aufgrund sich geänderter Rahmenbedingungen, Anforderungen oder Ähnlichem sinnlos wäre, darf der Product Owner für den vorzeitigen Abbruch sorgen. Ein neuer Sprint beginnt sofort nach der Beendigung des vorherigen Sprints [SS13]. Ein Sprint umfasst formal die Aktivitäten *Sprint Planning*, Daily Scrum, *Sprint Review* und *Sprint Retrospektive* [SS13].

Beim Sprint Planning werden die Arbeit für den kommenden Sprint geklärt und die Ziele durch den Product Owner festgelegt. Bei dieser Aktivität, für das bei einem Sprint mit einer Dauer von 30 Tagen maximal acht Stunden pro Monat verplant werden, werden die Pläne vom gesamten Scrum-Team erarbeitet. Im Wesentlichen werden beim Sprint Planning die zwei Fragen geklärt, was in dem jeweiligen Sprint fertig gestellt werden kann und wie die ausgewählte Arbeit erledigt werden kann [SS13]. Das *Was* orientiert sich an den Zielen die vom Product Owner im Product Backlog (siehe Abschnitt 3.1.3) formuliert wurden. Das Scrum-Team diskutiert, welche Ziele im folgenden Sprint erreicht werden sollten und welche Backlog-Einträge zu diesem Ziel passen und ausgewählt werden sollten. Einzig das Entwicklungsteam muss jedoch final bestimmen, wie viele Backlog-Einträge es innerhalb des nächsten Sprints schaffen kann [SS13]. Anschließend wird bestimmt, *wie* die ausgewählten Ziele erreicht werden können. Hierbei wird üblicherweise geklärt, welche Architektur gewählt werden muss, wie die Applikation aufgebaut werden soll, welche Interfaces geschrieben werden sollen, usw. Einzig das Entwicklungsteam bestimmt, wie es die Arbeit für den jeweiligen Sprint organisiert [Glo13]. Das *Sprint Backlog* steht am Ende dieses Schritts und enthält alle Product Backlog-Einträge, für den jeweiligen Sprint, inklusive den Umsetzungsplan des Entwicklungsteams (siehe Abschnitt 3.1.3). Falls sich während des Sprints herausstellt, dass der erwartete Aufwand überstiegen oder unterschritten wird, sind Anpassungen des Umfangs vom Sprint Backlog in Absprache mit dem Product Owner möglich [SS13].

Täglich werden während des gesamten Projekts 15-minütige Daily Scrum-Treffen durch den Scrum Master, der als Moderator auftritt, einberufen. In diesen Treffen stimmen die Mitglieder des Entwicklungsteams ihre Arbeit für den jeweiligen Tag ab. Es wird bestimmt, welche Arbeiten bis zum nächsten Tag möglich sind und welches Teammitglied welche Aufgabe übernimmt [Glo13]. Jeder Entwickler schildert dem Team zudem kurz, was er am vergangenen Tag zur Erreichung des Sprint-Ziels erreicht hat, was er am Tag des Daily Scrums zur Zielerreichung vorhat und welche Hindernisse ihn selbst oder das Team beim Erreichen der Ziele aufgefallen sind [SS13]. Am Daily Scrum-Treffen nehmen aktiv

nur das Entwicklungsteam und der Scrum Master teil. Jedes Teammitglied weiß jederzeit, mit welchen Aufgaben die anderen Mitglieder aktuell beschäftigt sind. Zudem werden Hindernisse und Fortschritt täglich überprüft und allgemein die Kommunikation verbessert. Weitere, zeitraubende Meetings sind neben den Daily Scrum-Treffen in der Regel nicht notwendig [SS13].

Das maximal vierstündige Sprint Review bildet den Abschluss eines Sprints. Hier prüft der Product Owner das im Sprint erstellte Inkrement. Nur wenn das Produkt fehlerfrei und vollständig ist, wird es einzig von ihm abgenommen und akzeptiert [Han10]. Fehlerhafte, unvollständige oder nicht getestete Funktionalitäten werden als nicht geliefert angesehen. Diese Ergebnisse können wieder in das Product-Backlog aufgenommen und im nächsten Sprint bearbeitet werden. Das Entwicklungsteam präsentiert sämtliche fertiggestellte Arbeit, die mit *Done* (siehe Abschnitt 3.1.3) gekennzeichnet wurde, beschreibt Probleme, die aufgetaucht sind und beantwortet Fragen zum Inkrement. Der Product Owner schildert den aktuellen Stand des Product Backlogs und erklärt, welche der Einträge mit Done gekennzeichnet sind. Im Sprint Review wird zudem bereits besprochen, wie die folgenden Schritte aussehen könnten [SS13].

Die Sprint Retrospektive unterstützt den kontinuierlichen Lernprozess des Teams und die Suche nach Verbesserungsmöglichkeiten für den nächsten Sprint. Das Ziel ist eine Optimierung des Prozesses in Bezug auf die beteiligten Menschen, die Zusammenarbeit, verwendete Werkzeuge und Arbeitsweisen [Han10]. Die Aktivität ist auf maximal drei Stunden pro Sprint angesetzt und findet zwischen dem Sprint Review und dem Sprint Planning statt. Auch wenn das Scrum-Team jederzeit Verbesserungen umsetzen kann, bietet die Retrospektive formalisiert die Möglichkeit, sich konkret und ausschließlich auf die Überprüfung und Anpassung zu konzentrieren. Im Gegensatz zum Sprint Review darf der Scrum Master aufgrund seiner Verantwortung für den gesamten Scrum-Prozess bei der Sprint Retrospektive auch selbst an den Entscheidungen mitwirken [SS13].

3.1.3 Artefakte

Artefakte stehen bei Scrum für *Arbeit* oder *Wert*, können aber auch als *Dokumente* verstanden werden [SS13, Han10]. Eines der Scrum-Prinzipien, die Transparenz, spielt bei der Definition der Artefakte in Scrum eine große Rolle. Die Artefakte sind dazu gedacht, für alle Prozessbeteiligten eine optimale Transparenz über die wesentlichen Informationen sowie ein gleiches Verständnis zu gewährleisten. Insbesondere ein gemeinsames Verständnis darüber was es heißt, wenn Arbeit wirklich fertiggestellt wurde, ist hierbei wichtig. Bei Scrum wird dies vom gesamten Team vor Projektbeginn mit der *Definition of Done* diskutiert und festgelegt. Je strenger hier die Kriterien angelegt werden, desto höhere Qualitätsstandards ergeben sich [SS13]. Die wichtigsten Artefakte sind das Product Backlog das Sprint Backlog und das Inkrement [SS13, Han10, Glo13].

Das Product Backlog ist eine Liste von Funktionalitäten, auch Product Backlog Items, die vom Product Owner priorisiert und gesammelt werden und die sämtliche Anforderungen, die für das fertige Produkt gelten sollen, enthält [Glo13]. Das Team kann den Product Owner vor allem bei der Schätzung des Realisierungsaufwands unterstützen [Han10]. Allerdings ist der Product Owner für sämtliche Einträge, Änderungen und Zugriffe und allgemein die Inhalte verantwortlich. Das Product Backlog wird dynamisch gehandhabt und kann sich kontinuierlich und im Normalfall solange, bis das fertige Produkt steht, ändern und anpassen. Zu Beginn der Scrum-Anwendung werden lediglich die ersten abgestimmten Anforderungen festgehalten. Aus Erfahrung und nach den Prinzipien der agilen Softwareentwicklung geht man davon aus, dass sich Anforderungen ständig ändern können

und gestaltet daher den Prozess und die Handhabung der Artefakte flexibel [SS13]. Der zeitlich und terminlich nicht konkret festgelegte Prozess, bei dem Details hinzugefügt, Schätzungen erstellt und die Reihenfolge der Einträge im Backlog bestimmt werden, wird als *Refinement* bezeichnet. Hierbei werden vom Product Owner und vom Entwicklungsteam Einträge begutachtet, verbessert und detailliert [SS13].

Das Sprint Backlog enthält die Einträge aus dem Product Backlog, die für den jeweiligen Sprint ausgewählt wurden. Zudem enthält es den im Sprint Planning formulierten Plan, mit dem die Sprint-Ziele erreicht werden sollen [SS13]. Ebenso wie beim Product Backlog werden auch am Sprint Backlog während eines Sprints kontinuierlich Änderungen vorgenommen. Das Sprint Backlog hilft dem Entwicklungsteam dabei, die erforderlichen Arbeitsschritte zu überblicken und zu sehen, was bereits erledigt wurde und was noch getan werden muss [Glo13]. Außerdem können vom Team weitere Arbeitsschritte je nach Bedarf hinzugefügt oder entfernt werden. Die Schätzung der verbleibenden Arbeit muss kontinuierlich aktualisiert werden. Das Sprint Backlog ist einzig für das Entwicklungsteam gedacht und bietet für die Mitglieder eine Art Echtzeit-Bild der Arbeit, die während des jeweiligen Sprints erreicht werden soll [SS13].

Am Ende eines Sprints muss das Entwicklungsteam Funktionalitäten liefern, die sich in einem auslieferbaren Zustand befinden [Glo13]. Das Inkrement ist die Summe aller in den bisherigen Sprints und dem aktuellen Sprint fertiggestellten Product Backlog-Einträgen. Nach Abschluss eines Sprints muss das Inkrement im Zustand Done sein [SS13].

3.1.4 Eignung und Anpassung für die Projektgruppe

Die Projektgruppe, bestehend aus zehn Studenten, nutzt Scrum als agiles Vorgehensmodell. In einem solchen Projekt liegen andere Rahmenbedingungen vor als in einer Organisation, in der Experten in Vollzeit an einem Projekt arbeiten können. Eine Anpassbarkeit ist dadurch gegeben, dass Scrum als Rahmenwerk zu verstehen ist und Regeln individuell erstellt werden können.

Der Product Owner wird zwar formal der Betreuer der Studenten sein, allerdings müssen seine Aufgaben vom gesamten Team und nicht einer Person übernommen werden. Die Tätigkeit als solche, z. B. das Identifizieren und Formulieren von Anforderungen oder das Verwalten des Product Backlogs, ändert sich nicht. Der Student, der die Rolle des Scrum Masters übernimmt, sollte sich ausreichend mit Scrum auseinandersetzen. Ein fest definierter Moderator, der für die Einhaltung von Regeln, zeitlichen Beschränkungen etc. zuständig ist und sämtliche Meetings wie z. B. Daily Scrum oder Sprint Review organisiert, unterstützt die Gruppe auch ohne vorherige Scrum-Erfahrungen. In der Projektgruppe nehmen zwei Studenten die Rolle des Scrum Masters ein. Das Team wird wöchentlich lediglich zu zwei Kernarbeitstagen zusammenkommen. Damit alle Teilnehmer der Projektgruppe die höchstmögliche Transparenz und Übersicht über den Fortschritt des Sprints haben, wird an jedem Kernarbeitstag ein Daily Scrum abgehalten.

Durch das Sprint Backlog und die Anwendung vom Projektmanagement-Tool *JIRA*[1] ist jederzeit ersichtlich, wer aktuell an welcher Aufgabe arbeitet und welchen Status die Aufgaben haben. So ist es als Teil der von Scrum geforderten Selbstorganisation zu sehen, dass sich die Mitglieder gegenseitig unterstützen, falls ein Entwickler eine Aufgabe nicht alleine erledigen kann oder die Aufgaben werden

[1] *JIRA*: Projektmanagement-Tool für agile Projekte. `https://de.atlassian.com/software/jira`, besucht am 13.08.2015.

in derartigen Fällen direkt von jemand anderem übernommen. Im Workflow innerhalb von JIRA wird zwischen vier verschiedenen Status, die die Tickets durchlaufen können, unterschieden. Im Status *Sprintbacklog (SBL)* steht zunächst jedes Ticket, das in dem jeweiligen Sprint bearbeitet werden soll. Für die Bearbeitung wird ein Ticket in *In Progress* verschoben und gleichzeitig dem jeweiligen Bearbeiter automatisch zugeordnet. Anschließend wird es in *Ready for Review* verschoben und ist zunächst Niemandem zugewiesen. Sobald sich ein Projektmitglied das Ticket zur Review vornimmt, ist es diesem Bearbeiter wieder automatisch zugewiesen und steht im Status *In Review*. Wenn das Ticket noch nicht als auslieferbar befunden wird, wird es zurück den Status In Progress geschoben und dem ursprünglichen Bearbeiter zugeteilt. Bei erfolgreicher Review kommt das Ticket in den Status done. Bei jeder Statusänderung ist der jeweilige Bearbeiter angewiesen, verständliche Kommentare für die Review, Weiterbearbeitung oder schlicht als Information im Ticket zu hinterlassen. Diese Gliederung dient der Übersicht über den Sprint, die Ticketstatus und die jeweiligen Bearbeiter der Tickets. Neben der agilen Anforderung an Transparenz genügt es durch die Status Ready for Review und In Review den Anforderungen nach Überprüfung (siehe Kapitel 3.1). Bestimmte User Stories, die ein hohes Maß an Umfang, Abhängigkeiten und Komplexität mit sich bringen, werden im Backlog zunächst dem *Epic Product Owner* zugeordnet. Epics dienen der Übersicht und stehen für eine bestimmte, thematische Zuordnung. User Stories, die im Product Owner Epic stehen, werden als reguläres Ticket im Sprint behandelt. Hier werden sie allerdings nicht implementiert, sondern detailliert beschrieben, sodass die Arbeit der intensiven Einarbeitung und des Überblick-Verschaffens von der eigentlichen Implementierung getrennt ist. Nach dieser Bearbeitung wird die User Story dem Epic *Ungeplante Userstory* zugeteilt, bis ihr Aufwand im Backlog Grooming geschätzt wird. Die Bearbeitung einer User Story im Product Owner Epic ist somit die Grundlage für eine gute Aufwandsschätzung.

Insbesondere in einem unerfahrenen Team ist die bei Scrum geforderte, kontinuierliche Überprüfung ein wichtiger Faktor. Es wird den Studenten nicht gelingen, von vorneherein sämtliche Anforderungen zu erfassen. Der inkrementelle Ansatz von Scrum und die beispielsweise durch Daily Scrum oder das Sprint Planning geschaffene Transparenz für jedes Teammitglied erleichtern die Hinzunahme neuer Anforderungen oder die Anpassung der Bestehenden. Auch die Prozesse werden von Beginn an kontinuierlich verbessert werden müssen. Daher ist es sinnvoll, die Länge der Sprints auf ein bis zwei Wochen zu begrenzen, damit Optimierungspotenzial durch Sprint Review und die Sprint Retrospektive zeitnah erkannt wird. Die Gruppe wählt für das Projekt einen Sprintzeitraum von zwei Wochen.

Großen zeitlichen Aufwand wird das alle zwei Wochen stattfindende Backlog Refinement oder -*Grooming* erfordern. Sämtliche Anforderungen müssen in diesen Sitzungen identifiziert und in das zu Beginn leere Product Backlog aufgenommen werden. Neben angemessen granularer Beschreibung der User Stories erfordert hier das Schätzen mithilfe von *Story Points* oder *Aufwandspunkten* für jede Anforderung Zeit. Diese Aufgabe ist, ebenso wie das anschließende Priorisieren der Anforderungen, insbesondere zu Beginn des Projektes anspruchsvoll. Erst mit der Zeit und zunehmender Erfahrung wird ein Gefühl für das Schätzen mit Story Points entstehen. In der Projektgruppe werden mit Story Points u.a. Risiken, Abhängigkeiten, erforderliches Fachwissen, zeitlicher Aufwand und fachlicher Anspruch verbunden. Das Backlog Grooming findet in zweiwöchigem Abstand immer in den Wochen statt, in denen der Sprint nicht beendet wird.

Nach jedem Sprint sollte ein auslieferbares Produkt stehen. Daher sind die Studenten dazu gezwungen, von Anfang an Ergebnisse zu liefern. Die Gefahr, dass am Ende der Aufwand und der Ergebnisdruck deutlich steigen und ein nicht-funktionierendes, halbfertiges Produkt geliefert wird, ist dadurch deutlich geringer. Für jeden Sprint wird eine gleichmäßige Anzahl an Arbeitsstunden verplant, sodass der Arbeitsaufwand während der gesamten Projektlaufzeit konstant bleibt.

3.2 Projektrollen

In Tabelle 3.1 werden die Projektrollen innerhalb der Projektgruppe beschrieben. Dabei wird erklärt, welche Aufgaben konkret mit den Rollen einher gehen und welche Personen diese Rollen übernehmen.

Rolle	**Name**	**Rollenbeschreibung**
Product Owner	Cornelius Ludmann, Prof. Dr. Dr. h. c. Hans-Jürgen Appelrath	siehe Kapitel 3.1.4
Scrum Master	Marius Wybrands, Max Leonhardt	siehe Kapitel 3.1.4
Dokumentation	Christoph Schröer	Verantwortlich für die Erstellung und Wartung der Dokumentationsvorlage in Latex
Qualitätsmanagement	Christoph Schröer	Erstellung von JUnit-Tests
Git/Repository	Christian Wigger	Aufsetzen der Repositories, Erstellung des Git-Workflows
Confluence	Achim Völz, Martin Kuhl	Einrichten, Verwalten, Ordnen des Wiki
JIRA	Christof Wolke, Lukas Weinel	Einrichten, Verwalten von JIRA, Erstellung der JIRA-Workflows
Master of Time	Christian Eilts	Überwachung der Meilensteinplanung und der Einhaltung von Deadlines
Entwicklerteam Domain-Schicht	Christoph Schröer, Marius Wybrands, Martin Kuhl, Patrick Bruns	Verantwortlich für grundlegende Architektur-Entscheidungen auf der Middleware, Implementierung grundlegender Funktionalitäten auf der Middleware, Erstellung von Queries auf Odysseus und Einrichtung und Anbindung von Odysseus und Datenbank
Entwicklerteam App	Achim Völz, Christian Eilts, Christian Wigger, Christof Wolke, Lukas Weinel, Max Leonhardt	Verantwortlich für grundlegende Architektur-Entscheidungen auf der App, Implementierung der App-GUI, der App-Funktionalitäten und grundlegender Funktionalitäten auf der Middlware

Tabelle 3.1: Projektrollen, Verantwortliche und Erläuterungen.

4 Anforderungserhebung

In diesem Kapitel geht es um die Anforderungsermittlung an das Produkt. Aus verschiedenen Informationsquellen sollen potenzielle Anforderungen an das Produkt ermittelt werden.

Eine erste Idee, wie das Projekt aussehen könnte, wird in einem Kick-off-Meeting (Kapitel 4.1) durch die Projektleitung dargestellt. Auf dieser Grundlage wird im zweiten Schritt ein interner Workshop (Kapitel 4.2) abgehalten. Das Ergebnis des Workshops ist ein Anwendungsfall, in dem Odysseus eingesetzt wird, um Nutzer mithilfe einer App verschiedene Locations wie z. B. Kneipen oder Restaurants vorzuschlagen. Im dritten Schritt werden verschiedene Akteure befragt, um eine möglichst ganzheitliche Sicht auf die Problemstellung zu erhalten. Ein Akteur ist z. B. der Projektleiter. Durch die Befragung der Akteure werden verschiedene funktionale Anforderungen erfasst. Durch ein internes Brainstorming der Projektgruppe werden weitere funktionale Anforderungen ermittelt (Kapitel 4.3). Die Bedürfnisse potenzieller Nutzer werden durch einen Online-Umfrage mit 170 Teilnehmern und durch 37 problemzentrierte Interviews in der *Oldenburger* Innenstadt erfasst. Um die Anforderungen an das Backend zu definieren, werden diverse wissenschaftliche Quellen durchgearbeitet. Die potenziellen Anforderungen an das Backend sind im Kapitel Analyse von wissenschaftlichen Quellen (Kapitel 4.7) zu finden. Zusätzlich werden noch vier Konkurrenzprodukte gesichtet. Daraus ergeben sich weitere potenzielle Anforderungen für die App.

Die konkreten Anforderungen ergeben sich aus den potenziellen Anforderungen, die in diesem Kapitel aufgeführt und extrahiert sind. Die Anforderungen, die im laufe dieser Projektgruppe umgesetzt werden sollen, werden im Kapitel spezifische Anforderungen (Kapitel 5) festgehalten.

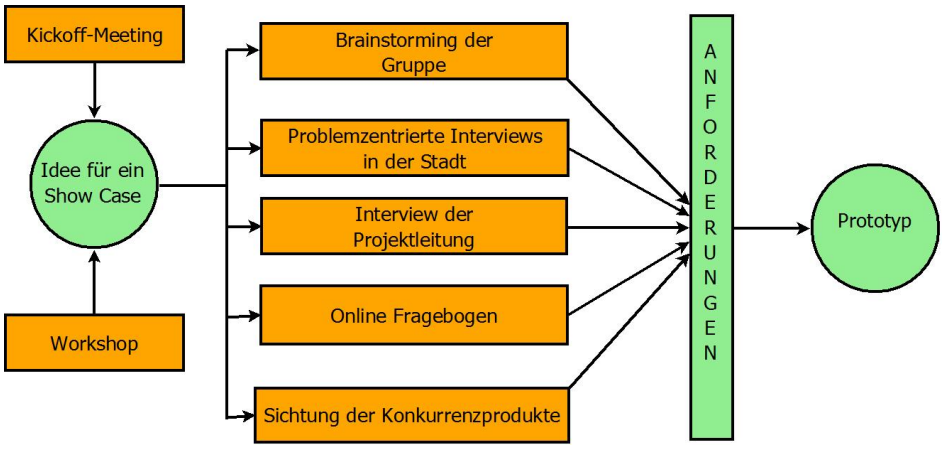


Abbildung 4.1: Durch das Kick-off-Meeting und einen internen Workshop der Projektgruppe wird zunächst eine Idee für einen Showcase erarbeitet. Die Anforderungen werden durch ein Brainstorming der Gruppe, verschiedenen Interviews in der Stadt, einen Online Fragebogen und ein Interview mit dem Projektleiter erhoben. Quelle: Eigene.

4.1 Anforderungen des Auftraggebers

In einem ersten Kick-off-Meeting am 17.04.2015 haben sich die Projektgruppe und der Projektleiter getroffen, um eine erste Idee der Problemstellung zu diskutieren. Ziel der Projektgruppe ist die Auswahl

und Umsetzung eines konkreten Anwendungsszenarios für ein datenstrombasiertes RecSys auf Grundlage von Odysseus. Als Beispiele sind verschiedene Bereiche, in denen ein RecSys Empfehlungen geben kann, genannt worden:

- Musik und Filme
- Nachrichten
- Restaurants und Kneipen
- Smartphone-Apps

Des Weiteren ist durch den Projektleiter eine mögliche Systemarchitektur vorgestellt worden. Diese ist in Abbildung 4.2 dargestellt. Die Abbildung stellt den Ausgangspunkt für die Anforderungsanalyse und das weitere Vorgehen dar. Sie ist generisch gehalten und könnte jeden genannten Bereich abbilden.

Aus der Abbildung lassen sich verschiedene Anforderungen an das Projekt ableiten. Es gibt im Anwendungsfall zwei Rollen. Die erste Rolle ist der Nutzer, der durch ein Eingabegerät bzw. eine App, die auf dem Eingabegerät installiert ist, Empfehlungen erhält. Das Eingabegerät kann zum Beispiel ein Smartphone, ein Tablet, ein Laptop oder ein Computer sein. Die zweite Rolle ist der Administrator (Admin), der mithilfe eines Dashboards relevante Informationen über das RecSys erhält. Dies dient der kontinuierlichen Überwachung und Evaluation des RecSys. Zudem soll es dem Admin möglich sein, Parameter und Einstellungen zu verändern und somit das RecSys zu konfigurieren. Für die Konfiguration könnte auch eine eigene Sprache Domain-Specific Language (DSL) entwickelt werden.

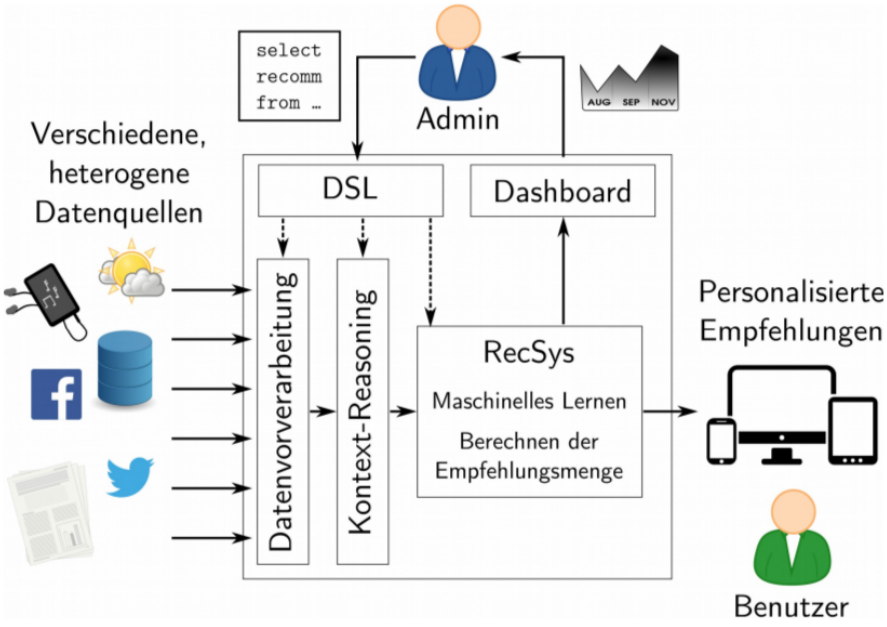


Abbildung 4.2: Systemarchitektur, die von der Projektleitung im Kick-off-Meeting vorgestellt worden ist. Anhand dieser Abbildung können verschiedene Anforderungen an das Projekt abgeleitet werden. Quelle: Folien der Präsentation im Kick-off-Meeting.

Um Empfehlungen zu unterstützen, sollen verschiedene APIs eingebunden werden. Dies kann zum einen einen Mehrwert für die Qualität der Empfehlung liefern, indem die Daten verarbeitet werden und in die Berechnungen der Empfehlung einfließen. Zum anderen können durch externe Programmierschnittstellen weitere kontextbezogene Informationen bereitgestellt werden. Als Beispiele sind u. a. die APIs von *Twitter*[1] und Facebook[2] genannt.

Es wird betont, dass die Erweiterung von Odysseus nicht ausgeschlossen ist, da schon geeignete Komponenten in Odysseus existieren, diese aber evtl. nicht sämtlichen Anforderungen genügen. Vor allem die Implementierung und ggf. Erweiterung von geeigneten strombasierten Online-RecSys-Algorithmen könnte unumgänglich sein. Bei der Integration und Verarbeitung von heterogenen Daten bietet Odysseus zwar Funktionalität, diese müsste je nach Ausarbeitung des Anwendungsszenarios ggf. ebenfalls erweitert werden.

Zusammengefasst ergeben sich folgende Anforderungen aus dem Kick-off-Meeting:

- Das Dashboard sollte dem Admin die Möglichkeit bieten, relevante Informationen über das RecSys einzusehen.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, das RecSys zu konfigurieren.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, das RecSys zu evaluieren.
- Das System sollte aus verschiedenen externen APIs Informationen beziehen, um die Qualität der Empfehlung zu verbessern.
- Das System sollte dem Nutzer eine personalisierte Empfehlung mithilfe der App anzeigen können.
- Das System sollte dem Nutzer mithilfe der App die Möglichkeit, geben eine Bewertung für ein Objekt abgeben zu können.
- Das System sollte auf dem Informationssystem Odysseus basieren.
- Der Admin sollte mithilfe einer DSL das RecSys konfigurieren können.

4.2 Workshop zur Auswahl eines Anwendungsszenarios

In dem Workshop zur Auswahl eines Anwendungsszenarios entstehen mit den Informationen aus dem Kick-off-Meeting verschiedene Anwendungsfälle. Das Ergebnis des Workshops ist ein erster Anwendungsfall, der die Grundlage für die Anforderungserhebung darstellt. Es sind verschiedene Kreativitätstechniken zum Einsatz gekommen. Im ersten Teil des Workshops entsteht durch ein *Brainwriting* eine Sammlung von Ideen. In einer großen Runde findet eine Diskussion auf Grundlage der in Kleingruppen generierten Ergebnisse statt 11.1. Zusammengefasst können die Ideen auf folgende Bereiche reduziert werden:

- Locations
- Veranstaltungen

[1] *Twitter*: Kurznachrichten- und Microbloggingdienst, `https://twitter.com`, besucht am 29.06.2015.
[2] *Facebook*: Großes soziales Netzwerk, `https://www.facebook.com`, besucht am 29.06.2015.

- Produkte

Bei der Entscheidung spielen verschiedene Faktoren eine Rolle: Da von der Projektleitung keine Einschränkung in eine bestimmte Richtung gegeben ist, wird ein Bereich ausgewählt, mit dem alle Projektmitglieder einverstanden sind. Zur Diskussion steht außerdem, ob das gewählte Szenario die Vorteile von einem System, das auf Datenströmen basiert, ausreichend hervorhebt. Weitere, zu diskutierende Aspekte sind Aktualität der Daten, die Menge der Daten und die Frage, wieso dieses Anwendungsszenario nicht auf einer relationalen Datenbank basiert. Die Wahl fällt auf den Bereich Locations. Der Nutzer soll jedoch nicht die Location als solche bewerten, sondern den Besuch der Location. Dadurch werden die Aktualität der Bewertungen sowie Kontextabhängigkeit der Kriterien sichergestellt.

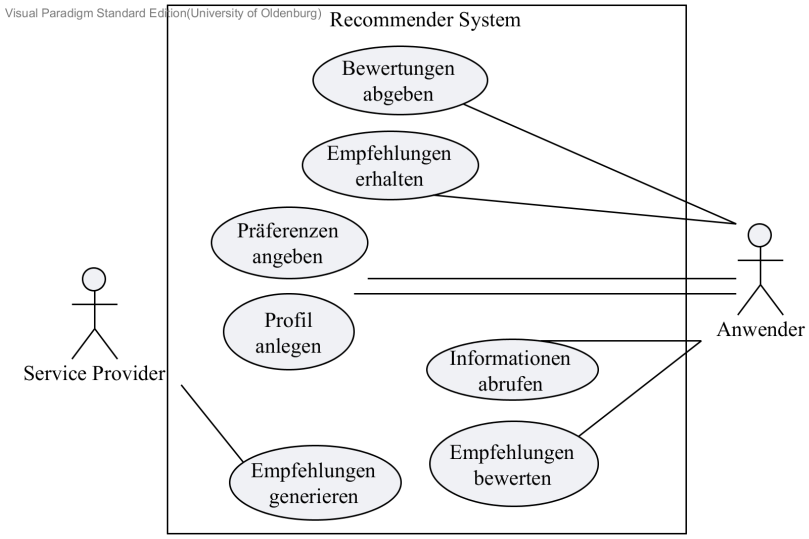


Abbildung 4.3: Ergebnis des Workshops in Form eines Use-Case-Diagramms dargestellt. Quelle: Eigene.

Im zweiten Teil des Workshops geht es um die Erstellung potenzieller Anwendungsfälle. In Kleingruppen werden drei Anwendungsfälle entwickelt und analysiert. Die drei Use Cases werden zu einem Use Case zusammenfasst und in einem Akteur-Diagramm festgehalten. Das erarbeitete Use-Case-Diagramm ist in Abbildung 4.3 abgebildet. Das System bietet dem Nutzer die Möglichkeit, Besuche zu bewerten und Empfehlungen für Locations zu erhalten. Zusätzlich kann der Nutzer Informationen über die Locations abrufen. Bei der Generierung der Empfehlung erhält das System zusätzliche Informationen von verschiedenen Service Providern. Eine weitere Funktionalität ermöglicht dem Nutzer, erhaltene Empfehlungen zu bewerten. Dies soll die Qualität der Empfehlungen erhöhen. Der Nutzer kann zusätzlich Präferenzen und persönliche Daten in einem Profil anlegen.

4.3 Brainstorming zur Anforderungsanalyse

Als zusätzliche Informationsquelle wird ein Brainstorming der Projektgruppe genutzt. Das Ergebnis des Brainstormings ist in Abbildung 4.4 zu sehen. Die verschiedenen Funktionalitäten, die für die App als potenziell wichtig angesehen werden, sind zunächst unstrukturiert festgehalten. Neben dem

Thema, das sich mit der Ausgabe von Empfehlungen beschäftigt, werden verschiedene Möglichkeiten zur Anzeige von Empfehlungen diskutiert.

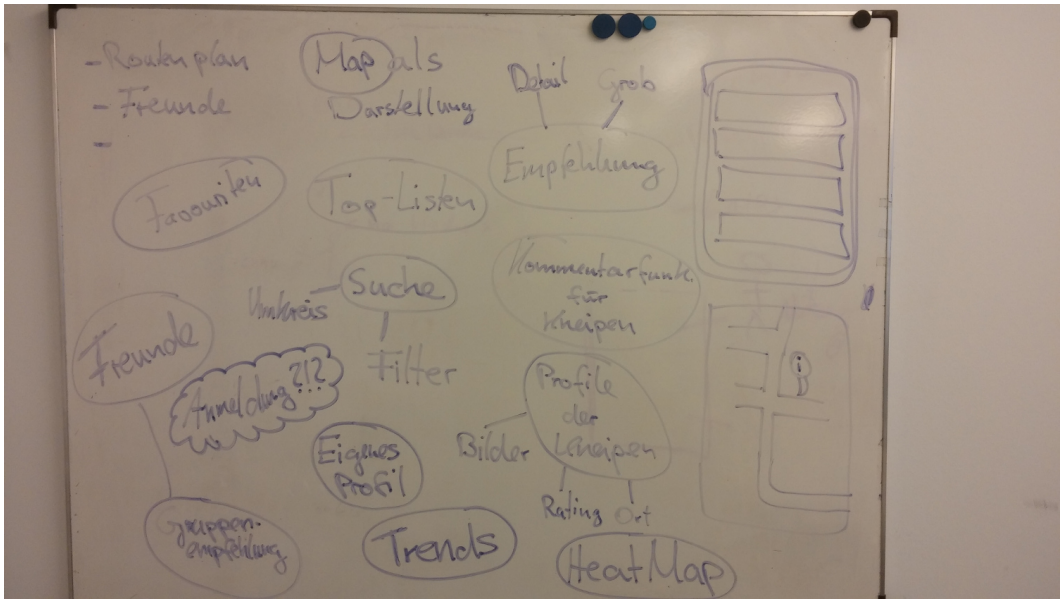


Abbildung 4.4: Ergebnisse des Brainstormings der Projektgruppe als Teil der Anforderungsanalyse. Quelle: Eigene.

Empfohlene Locations sollten in einer geordneten Liste angezeigt werden. Zusätzlich wird in Betracht gezogen, die Darstellung auf einer Karte zu realisieren. Damit den Nutzern ergänzende Informationen über die Locations bereitgestellt werden können, sollte es eine Detailansicht geben, in der diese Informationen dargestellt werden. Dies impliziert, dass in der normalen Ansicht lediglich wenige Basisinformationen über die Locations angezeigt werden. Eine optionale Konfigurierungsfunktion soll die Anzeige personalisierter Informationen in der normalen Ansicht ermöglichen. Diese zusätzlichen Informationen können in dem Profil der jeweiligen Location dargestellt werden. Dieses Profil soll Bilder, verschiedene Ratings, den Ort an sich sowie eine Kommentarfunktion beinhalten. Damit der Nutzer aktuelle Trends erkennt, soll die App einen *Trendalarm* implementieren. Die App soll außerdem eine Suchfunktion implementieren. Durch Filter und durch eine Umkreissuche kann der Nutzer seine Präferenzen eingeben. Diese Informationen können anschließend für eine bessere Empfehlungsgenerierung genutzt werden.

Ein weiteres Thema befasst sich mit kollaborativen Funktionalitäten: Die Kommunikation zwischen einzelnen Nutzern über die App ist eine Option. Eine weitere Möglichkeit ist eine Art Gruppenempfehlung, bei der mehrere Nutzer ihre Profile zusammenlegen und dann die optimale Location für genau diese Konstellation erhalten. In diesem Zusammenhang wird gleichzeitig über das Thema *Anmeldung* diskutiert. Aus der Diskussion geht jedoch hervor, dass auf eine Anmeldung zunächst verzichtet werden soll. Dies schränkt einige Funktionalitäten ein. Insbesondere die kollaborativen Funktionalitäten sind von dieser Einschränkung betroffen. Eine Möglichkeit, diese trotzdem noch umzusetzen, ist eine anonyme Anmeldung. Zusammengefasst ergeben sich folgende Anforderungen aus dem Brainstorming:

- Die App sollte dem Nutzer die Locations auf einer Karte anzeigen.
- Die App sollte dem Nutzer die Locations in einer geordneten Liste anzeigen.
- Die App sollte dem Nutzer durch eine Detailansicht zusätzliche Informationen über die Locations bereitstellen.
- Die App sollte dem Nutzer in der normalen Ansicht nur relevante Informationen anzeigen.
- Die App sollte dem Nutzer in der normalen Ansicht personalisierte Informationen anzeigen.
- Die App sollte dem Nutzer die Möglichkeit bieten, mit anderen Nutzern zu kommunizieren.
- Die App sollte dem Nutzer die Möglichkeit geben, sich mit anderen Nutzern in einer Gruppe zusammenzuschließen.
- Die App sollte dem Nutzer die Möglichkeit geben, eine Gruppenempfehlung auf Grundlage der verschiedenen Nutzer in der Gruppe zu geben.
- Die App sollte ohne Anmeldung funktionieren.
- Die App sollte dem Nutzer die Möglichkeit bieten, Locations zu suchen.
- Die Detailansicht sollte dem Nutzer Bilder der Locations anzeigen.
- Die Detailansicht sollte dem Nutzer ein Profil der Location anzeigen.
- Die Detailansicht sollte dem Nutzer die Möglichkeit bieten, einen Kommentar zu hinterlassen.
- Die Detailansicht sollte dem Nutzer die Möglichkeit bieten, die Bewertung der Location im Detail zu betrachten.
- Die Suchfunktion sollte dem Nutzer die Möglichkeit bieten, durch Filter die Suche zu verfeinern.
- Die Suchfunktion sollte dem Nutzer die Möglichkeit bieten, durch eine Umkreissuche eine Location zu finden.

Über die endgültige Aufnahme dieser potenziellen Anforderungen wird erst nach Beenden aller Methoden zur Anforderungsermittlung diskutiert. Nicht sämtliche der im Brainstorming diskutierten Ideen werden durchführbar oder relevant sein.

4.4 Online-Umfrage

Im Rahmen der Anforderungserhebung wird eine Online-Umfrage durchgeführt, um bestimmte Anforderungen potenzieller Nutzer an die App zu sammeln. Die detailierten Ergebnisse der Online-Umfrage sind Anhang (Kapitel 11.3) zu finden. Zum einen soll durch Abfragen möglicher Funktionen der App herausgefunden werden, welche davon tatsächlich gewünscht sind. Zum anderen soll ermitteln werden, welche Attribute und Kontextinformationen einem Nutzer wichtig sind, damit diese in das Empfehlungssystem eingebunden werden können.

Die Online-Umfrage wird als Methode der Anforderungsermittlung ausgewählt, da sie es ermöglicht, in einem kurzen Zeitraum viele Teilnehmer zu erreichen. Zudem wird viel Zeit durch die automatische

Auswertung und Präsentation der Umfrage-Ergebnisse eingespart und dadurch mögliche Fehlerquellen manueller Eingaben vermieden. Außerdem lässt sich in einer Online-Umfrage die Darstellung von Bildern oder Mockups einfach realisieren. Einen weiteren Vorteil stellen diverse Akzeptanzkriterien einer Online-Umfrage dar. Befragte können eine Online-Umfrage meist freiwillig und anonym durchführen und sind so eher gewillt, ihre Meinung preiszugeben [Wel12].

Da das Nutzer-Interface eine App werden soll, ist die Umfrage an Nutzer von Smartphones adressiert. Die App soll Locations basierend auf den individuellen Interessen der Nutzer empfehlen können. Da sich die einzelnen Kategorien von Locations unterscheiden, werden folgende vier Klassen gebildet: *Bars/Kneipen*, *Restaurants/Lokale*, *Cafés/Eisdielen* sowie *Discos/Clubs*. Eine Einteilung in Klassen ist notwendig, da sich die relevanten Kriterien der einzelnen Klassen unterscheiden. Für jede Klasse von Location wird jeweils abgefragt, welche ihrer Eigenschaften dem Nutzer wichtig sind.

Die Bewertung erfolgt hierbei auf einer Skala von *nicht wichtig* bis *sehr wichtig* über *weniger wichtig* bis *wichtig*. Es besteht zudem die Möglichkeit, keine Angabe bei der Bewertung zu machen. Zur Auswahl stehen die Eigenschaften *Preise*, *Qualität der Getränke*, *Qualität der Speisen*, *Auswahl der Speisen und Getränke*, *Musik*, *Außengastronomie*, *Service und Ambiente*. Durch diese Abfrage soll ermittelt werden, inwieweit die Eigenschaften einer bestimmten Location den Befragten relevant erscheinen.

Des Weiteren sollen die Teilnehmenden angeben, anhand welcher Kriterien sie eine konkrete Location auswählen. Es können zusätzlich weitere Kriterien, die in der Umfrage nicht aufgeführt sind, in ein Freitextfeld geschrieben werden. Die zur Auswahl stehenden Kriterien unterscheiden sich teilweise in Abhängigkeit von der Art der Location. Für jede Klasse wird abgefragt, ob die aktuelle *Anzahl an Gästen*, die *Vielfältigkeit des Angebots*, der *Wochentag* und die *Uhrzeit*, die *Parkplatzsituation vor Ort* sowie die *Anbindung an den Öffentlichen Personennahverkehr* wichtig für die Auswahl einer bestimmten Location sind. Spezifisch ist das Kriterium *Musikgenre*, das nur für Clubs und Discos sowie Kneipen und Bars abgefragt wird. Spezifische Kriterien von Restaurants und Lokalen sind *Reservierungsmöglichkeiten*, *Schnelligkeit*, also die Dauer von der Bestellung bis zum Servieren der Speise, der *Preis*, das *Kinderangebot* und das *Wetter*. Für Cafés und Eisdielen wird spezifisch abgefragt, ob Angebote für Kinder bei der Auswahl eines Cafés oder einer Eisdiele für die Befragten eine Rolle spielen. Für Clubs und Discos wird zusätzlich abgefragt, ob die Anzahl der *Floors*, das Angebot von *Mottoparties* und die *Anzahl von Sitzgelegenheiten* wichtige Kriterien für die Auswahl dieser Location sind.

Anschließend werden die Teilnehmenden nach ihrem Interesse an einer App, die basierend auf persönlichen Interessen Empfehlungen für Locations geben kann, befragt. Falls diese Frage mit *ja* beantwortet wird, werden weitere Fragen zu einer solchen App gestellt. Ist kein Interesse vorhanden, wird der folgende Teil des Fragebogens übersprungen und der Teilnehmende zu den demographischen Fragen geleitet. Diese beinhalten die Angabe der *Altersklasse*, der *Berufsgruppe* und des *Geschlechts*. Die demographischen Angaben sollen bei der Definition einer möglichen Zielgruppe für die App unterstützen.

Die Fragen zur Recommender-App beziehen sich auf die Funktionen der App und auf deren Design. Hinsichtlich der Funktionalitäten wird gefragt, ob die App einen Stadtplan, eine Routenplanung zur Location, eine Funktion zum Austausch mit Freunden, eine Kommentar- oder Review-Funktion, die Möglichkeit des Anlegens von Favoriten sowie eine Umkreissuche beinhalten soll. Die Auswahl dieser Funktionen basiert auf der Sammlung von Ideen durch die Gruppe aus den wöchentlichen Sitzungen. Falls den Teilnehmenden der Umfrage weitere Funktionen für eine derartige App einfallen,

können diese in einem Freitextfeld notiert werden. In der folgenden Frage werden das Interesse für eine automatische Benachrichtigung von potenziell interessanten Locations in der Nähe ermittelt und diverse Designvorschläge der Bewertungsfunktion vorgestellt. Hierfür liegen vier Mockups zur Bewertung vor. Sie zeigen Bewertungen mittels Daumen hoch / Daumen runter, Sterne für den Besuch einer gesamten Location, Sterne für die einzelnen Kriterien eines Besuches und eine Bewertung anhand einer Skala. Die Umfrage wird mit einem Block von demographischen Fragen abgeschlossen.

Die Umfrage ist in sozialen Medien verteilt worden, um eine breite Masse zu erreichen. Außerdem ist die Umfrage im *Stud.IP*[3] der *Universität Oldenburg* veröffentlicht worden.

4.4.1 Probleme

Bei der Durchführung der Online-Umfrage sind diverse Probleme aufgetreten. Zu Beginn ist die Plattform *Surveymonkey*[4] benutzt worden. Die Funktionalität ist aufgrund der wenigen kostenlosen Basisfunktionen limitiert. Daher ist die kostenlose Variante von *Google*[5] alternativ verwendet worden. Der Vorteil dieser Option besteht darin, dass kostenlos beliebig viele Fragen gestellt und Bilder eingefügt werden können. Außerdem ist es möglich, bestimmte Fragen in Abhängigkeit von den Antworten der Teilnehmenden zu überspringen.

Ein weiteres Problem hat sich durch unpräzise formulierte Fragestellungen ergeben. So ergibt die Auswertung beispielsweise, dass den Teilnehmenden die Nationalität eines Restaurants nicht wichtig ist, wohingegen die Spezialität, also die Landesküche (z. B. Griechisch, Italienisch, etc.) als sehr wichtig angesehen wird. Hier ist der Begriff *Nationalität* wahrscheinlich fehlinterpretiert worden.

4.4.2 Interpretation der Ergebnisse der Online-Umfrage

Durch die Online-Umfrage 11.3 sind zusätzliche, potenzielle Anforderungen ermittelt worden. Es ist unter anderem abgefragt worden, welche Kriterien von Bars und Kneipen, Restaurants und Lokalen, Cafés und Eisdielen sowie Clubs und Discos für die Umfrageteilnehmer bei der Auswahl einer Location eine Rolle spielen. Die Ergebnisse zeigen, dass je nach Klasse der Location unterschiedliche Kriterien als wichtig eingestuft werden. Für alle Klassen spielen das Preisniveau bzw. das Preisleistungsverhältnis der jeweiligen Location für die meisten Umfrageteilnehmer eine wichtige Rolle. Beide Kriterien sind mehrheitlich mit wichtig oder sehr wichtig gewichtet worden. Ebenso sind die Kriterien Auswahl an Speisen und Getränken sowie deren Qualität, das Ambiente und der Service häufig als wichtige Kriterien bei der Auswahl einer Location eingestuft worden. Die in der Location gespielte Musik ist hingegen für die Umfrageteilnehmer nur für Bars und Kneipen sowie Discos und Clubs ein wichtiges Kriterium. Daher ergeben sich hinsichtlich der Bewertungskriterien für Locations folgende potenzielle Anforderungen:

- Die App sollte dem Nutzer die Möglichkeit geben, die Preise zu bewerten.

[3] *Stud.IP*: Lern-, Informations- und Projekt-Management-System. `http://www.studip.de`, besucht am 29.06.2015.
[4] *Surveymonkey*: Kostenloses Softwaretool für Fragebögen und Online-Umfragen, `https://de.surveymonkey.com`, besucht am 18.06.2015.
[5] *Google Formulare*: Kostenloses Softwaretool für Fragebögen und Online-Umfragen, `https://docs.google.com/forms`, besucht am 21.06.2015.

- Die App sollte dem Nutzer die Möglichkeit geben, die Auswahl an Speisen und Getränken zu bewerten.
- Die App sollte dem Nutzer die Möglichkeit geben, die Qualität der Speisen und Getränke zu bewerten.
- Die App sollte dem Nutzer die Möglichkeit geben, den Service zu bewerten.
- Die App sollte dem Nutzer die Möglichkeit geben, das Ambiente zu bewerten.

Den Teilnehmern werden vier Mockups möglicher Bewertungsformen gezeigt (s.o.). Mehrfachnennungen sind erlaubt, um mögliche Kombinationen zu berücksichtigen. Die Bewertung mittels Sternen für die einzelnen Kriterien ist 124 mal ausgewählt worden. Den nächsthöchsten Zuspruch erhält die Bewertung anhand der Vergabe von Sternen für den Besuch mit 37 Klicks. Um den Ansprüchen der Umfrageteilnehmer gerecht zu werden, wird folgende potenzielle Anforderung formuliert:

- Die Bewertungen sollten detailliert mit Sternen erfolgen.

Das Umfrageergebnis zeigt den Wunsch der Befragten, weitere Informationen wie z.B. die Speisekarte und aktuelle Angebote, in der App angezeigt zu bekommen. Um dem gerecht zu werden, soll es für jede Location eine Informationsseite geben, auf der die Informationen dargestellt werden. Weiterhin geben die Nutzer in den Freitextfeldern wiederholt an, dass sie Bilder der Location angezeigt bekommen möchten. Vorhandene Außengastronomie ist den Nuzern außer bei Clubs und Discos ebenfalls wichtig. Die Möglichkeit, Kommentare zu Locations schreiben zu können, ist für Nutzer eine wichtige Funktion. Daher ergeben sich folgende potenzielle Anforderungen:

- Die App sollte für jede Location eine Informationsseite enthalten.
- Die Informationsseite sollte die Speisekarte enthalten.
- Die Informationsseite sollte die aktuellen Angebote enthalten.
- Die Informationsseite sollte für Locations der Kategorie Restaurant die Spezialitäten des Hauses bzw. die jeweilige Landesküche enthalten.
- Die Informationsseite sollte die Information enthalten, ob eine Außengastronomie vorhanden ist.
- Die Informationsseite sollte Bilder der Location enthalten.
- Die Informationsseite sollte Kommentare enthalten.
- Der Nutze sollte die Möglichkeit bekommen, Locations zu kommentieren.

Für alle Klassen von Locations soll die Entfernung vom aktuellen Standort des Nutzers bis zur jeweiligen Location berechnet werden können. Zudem sollen die Uhrzeit und der Wochentag, das Wetter sowie vorhandene Außengastronomie (außer bei Clubs und Discos) berücksichtigt werden. Darüber hinaus spielen länderspezifische Spezialitäten im Angebot von Restaurants und Lokalen eine wichtige Rolle. Bei dieser Location wird ferner auf die Musik und das entsprechende Genre geachtet. Ein weiterer Aspekt der den Teilnehmern für alle Klassen von Locations mehrheitlich wichtig ist, ist die Anzahl der aktuell in einer Location anwesenden Gäste. Aus den Freitextantworten

ergibt sich zusätzlich, dass soziale Faktoren relevant sind. Eine mögliche Interpretation lässt den Schluss zu, dass die Umfrageteilnehmer bevorzugt eine Location aufsuchen, in der sich momentan ihre Freunde aufhalten oder die ihnen von ihren Freunden empfohlen wurde. Daraus ergeben sich folgende potenzielle Anforderungen:

- Beim Geben von Empfehlungen sollte die Entfernung zum aktuellen Standort berücksichtigt werden.
- Beim Geben von Empfehlungen sollten die Uhrzeit und der Wochentag berücksichtigt werden.
- Beim Geben von Empfehlungen sollte das Wetter einbezogen werden.
- Beim Geben von Empfehlungen sollte mit Ausnahme von Clubs und Discos für alle Klassen berücksichtigt werden, ob eine Außengastronomie vorhanden ist.
- Beim Geben von Empfehlungen für Restaurants und Lokale sollten die Spezialitäten der Restaurants einbezogen werden.
- Beim Geben von Empfehlungen sollte für alle Klassen das gespielte Musikgenre berücksichtigt werden.
- Beim Geben von Empfehlungen sollte die Anzahl der aktuell anwesenden Gäste der jeweiligen Location berücksichtigt werden.
- Beim Geben von Empfehlungen sollten soziale Aspekte berücksichtigt werden.

Die Umsetzung dieser potenziellen Anforderungen erfordert die Erhebung von diversen Daten. Bei der Datenerfassung spielen Eingaben des Nutzers eine wesentliche Rolle. So kann sich bspw. das Genre der gespielten Musik in Abhängigkeit vom Tag aufgrund von Mottoparties unterscheiden. Dadurch werden folgende potenzielle Anforderungen begründet:

- Der Nutzer sollte das Genre der gespielten Musik für alle Klassen von Locations in der App erfassen können.
- Der Nutzer sollte die länderspezifische Spezialitäten eines Restaurants in der App erfassen können.

Rund 95 % der Umfrageteilnehmer geben an, dass ihnen eine Übersicht über alle Locations innerhalb der App wichtig oder sehr wichtig ist. Ähnlich hohen Zuspruch erreichen die Funktionen, einen Stadtplan innerhalb der App anzuzeigen (ca. 85 %), eine Routenplanung in der App auszuführen (ca. 85 %) und das Anlegen von Favoriten (ca. 58 %). Innerhalb der Freitextfelder geben mehrere Teilnehmer an, dass neben Locations auch Events in die App eingebunden werden sollten. Potenzielle Anforderungen lassen sich wie folgt formulieren:

- Die App sollte über einen Stadtplan verfügen.
- Über den Stadtplan sollte die Funktion der Umkreissuche umgesetzt werden.
- Über den Stadtplan sollte die Funktion einer Routenplanung zur Location umgesetzt werden.
- Die App sollte über eine Übersicht der Locations verfügen.

- Die App sollte es ermöglichen Favoriten anzulegen.
- Die App sollte Events anzeigen.

Über die Hälfte der Teilnehmer gibt an, dass sie nicht über für sie interessante Locations in ihrer Nähe benachrichtigt werden will. Aus den Ergebnissen der Online-Umfrage ergibt sich, dass eine derartige Push-Notification keine potenzielle Anforderung darstellt. Bestimmte Qualitätskriterien für Umfragen können mit der Online-Umfrage nicht erfüllt werden. Im Rahmen des Stundentenprojektes ist dies nie als Anspruch gestellt worden. 64,4 % der Teilnehmer sind zwischen 20 und 24 Jahre alt (25 - 29 Jahre: 23 %). Bei dem Großteil der Teilnehmer handelt es sich um Studenten (74,7 %) oder Angestellte bzw. Beamte (19,5 %). 55,7 % der Teilnehmer sind männlich.

4.5 Problemzentriertes Interview

Neben dem Online-Fragebogen dient der zweite Fragebogen in Form eines *problemzentrierten Interviews* ebenfalls dem Ermitteln von Anforderungen. Vor allem um bestimmte Nachteile von Online-Umfragen abzufangen, aber auch um weitere Möglichkeiten des Sammelns von Anforderungen zu erschließen, wird bewusst eine unterschiedliche Vorgehensweise gewählt. Beim problemzentrierten Interview ist der Ablauf stark vom Befragten bzw. dessen Antworten abhängig. Der Interviewer teilt sein vorläufiges, theoretisches Konzept für die Befragung bewusst nicht mit, um den Befragten nicht in seinen Antworten zu beeinflussen. Dennoch hat er einen im Vorhinein erarbeiteten Leitfaden im Kopf. Mittels Zurückspiegelung stellt der Interviewer die Antworten in seinen eigenen Worten und seiner Interpretation dar. So werden Missverständnisse vermieden und zugleich dem Befragten die Möglichkeit geboten, seine Aussagen gegebenenfalls zu korrigieren oder zu modifizieren. Weitere Optionen, die Aussagen des Befragten auf Konsistenz zu prüfen, sind Verständnisfragen. So können vom Interviewer Widersprüche oder ausweichende Aussagen angesprochen werden. Er kann den Interviewten auch direkt mit Unstimmigkeiten konfrontieren. Durch diese Art der Interviewführung ist der Verlauf jedes Interviews unterschiedlich. Überhaupt nicht angesprochene Themenbereiche, die jedoch zentral für die Erhebung sind, können abschließend vom Interviewer anhand direkter Fragen gestellt werden. Dadurch wird der Interviewfluss zum einen nicht mittendrin unterbrochen und zum andern erleichtert es die Vergleichbarkeit der Befragungen. Einen Kompromiss zwischen qualitativen und quantitativen Interviews kann man durch das Voranstellen eines standardisierten Kurzinterviews herstellen. Dies erleichtert außerdem den thematischen Einstieg in die Befragung [Wit85]. Als ein solcher Einstieg in das Interview werden von der Projektgruppe die folgenden Fragen gestellt:

- Besitzen Sie ein Smartphone?
- Nutzen Sie häufig Mobile-Applikationen?
- Was fallen Ihnen bei dem Begriff Location oder Örtlichkeit spontan für konkrete Beispiele ein?

Die Kernfragen, die die Interviewer bei der Befragung in Kopf haben und die zumindest kurz in jedem Interview angesprochen werden sollten, sind die Folgenden:

- Welche Kriterien sind Ihnen bei einer App wichtig?
- Sind Sie oft auf der Suche nach Locations?

- Welche Kriterien sind Ihnen bei einer Location am wichtigsten?
- Auf welche Schwierigkeiten stoßen Sie auf der Suche nach Locations?
- Geben Sie online z. B. bei Amazon Bewertungen ab und könnten Sie es sich vorstellen, dies auch bei einer App zu machen?
- Würden Sie eine App nutzen, die Ihnen Empfehlungen für Locations gibt? Falls ja, was sollte diese App auf alle Fälle können?
- Würden Sie eine Funktion nutzen, bei der Sie freiwillig angeben können, in welcher Location Sie sich befinden, sodass ausgewählte Personen, die ebenfalls diese App nutzen, dies sehen können?

Die Frage nach Beispielen für Locations im Einstiegsblock zielt darauf ab, die Bedeutung der Bezeichnung *Location* für die potenziellen Nutzer zu ermitteln. Sie hilft außerdem dabei, einen Fokus auf bestimmte, besonders häufig genannte Beispiele zu setzen und diese als Kategorie in die App mit aufzunehmen.

Über die erste Hauptfrage sollen nicht-funktionale Anforderungen an die App gesammelt werden. Mithilfe des nächsten Punktes sollen Ideen für relevante Kontextdaten generiert werden. Dies wird durch die hier nicht aufgelistete, aber im Normalfall gestellte Nachfrage danach, wann und zu welchen Gelegenheiten die Befragten auf der Suche nach Locations sind, konkretisiert. Die nachfolgende Hauptfrage dient der Projektgruppe dazu, die Kriterien, die in das Bewertungsschema der App einfließen sollen, zu priorisieren. Nur die am häufigsten genannten Punkte sollten demnach in die Bewertung mit einfließen oder zumindest mit höherer Priorität bei der Implementierung versehen werden. Die Frage danach, welche konkreten Schwierigkeiten bei der Suche nach Locations auftreten, kann Aufschluss darüber geben, welche Funktionalitäten die App bereitstellen sollte, um diesen Problemen zu begegnen und und dem Nutzer Nutzen zu bringen. Speziell bezogen auf die geplante Bewertungsfunktion der App zielt die anschließende Frage ab. Wo und in welcher Form die Befragten bewerten, was ihnen bei der Bewertung wichtig ist und unter welchen Umständen z. B. selbst Bewertungen über eine App abgegeben würden, soll Ansätze liefern, welche Ausprägungen die Bewertungsfunktion der App haben sollte und was ggf. vermieden werden sollte. Die Frage, ob die Befragten eine App, wie sie zu Beginn des Projektes im Entwurf geplant ist, nutzen würden, kann ebenfalls wertvolle Erkenntnisse liefern.

Insbesondere über Nachfragen, z. B. weshalb die Frage verneint wurde oder welche Funktionen die App im Falle eines Bejahens bereithalten sollte, hilft dabei, funktionale Anforderungen und evtl. Falltüren zu ermitteln, um sie bei der Anforderungsanalyse berücksichtigen zu können. Da eine Social-Media-Funktion oder die Lokalisierung der Nutzer ein heikles Thema mit Bezug auf den Datenschutz ist, wird zum Ende die Frage gestellt, ob und unter welchen Umständen die potenziellen Nutzer eine derartige Funktion akzeptieren würden. Ohne die Akzeptanz dieser Nutzer müsste diese Funktionalität nicht implementiert werden.

Neben diesen Leitfragen werden zu jeder Frage Nachfragen gestellt und direkt auf getroffene Aussagen Bezug genommen. Die Reihenfolge der Fragen orientiert sich an den Antworten der Befragten und ist daher variabel. Die Anzahl der Fragen ist so gehalten, dass der zeitliche Umfang eines Interviews im Normalfall nicht länger als 15 Minuten beträgt. Dadurch erhöht sich die Chance, möglichst viele Teilnehmer zu finden und befragen zu können.

Einen Block mit den Kurzinformationen nach Altersgruppe (zwischen 15 und 20, 20 und 25, usw.) sowie nach Berufsgruppe (Schüler, Student, Angestellter, Beamter, Selbstständiger oder Rentner) und

Wohnort wird bewusst erst an das Ende des Interviews gelegt. Direkt zum Start des Interviews nach persönlichen Daten zu fragen könnte potenzielle Interviewpartner abschrecken. Diese Informationen dienen als Interpretationshilfe um die Bedeutung der Aussagen nach Beendigung aller Befragungen besser einschätzen zu können. Im Unterschied zu den überwiegend geschlossenen Fragen der Online-Umfrage erhofft sich die Projektgruppe durch die offenen Fragen des problemzentrierten Interviews, neue Ansichten und Ideen für die Anforderungen an die App zu erhalten. Die überlegten Kernfragen sind so formuliert, dass hierbei funktionale und nicht-funktionale Anforderungen ermittelt werden können.

4.5.1 Potenzielle Anforderungen

Ein Vorteil der offene-Fragen-Struktur besteht darin, dass die Befragten frei und unvoreingenommen in ihren Aussagen sind. Dadurch können idealerweise vollkommen neue Anforderungen und Anregungen für die Projektgruppe ermittelt werden. Aussagen, die eine User Story darstellen könnten, allerdings entweder nicht umsetzbar oder nicht relevant erscheinen, werden nicht aufgenommen. Dies ist bei sehr speziellen Kriterien, die lediglich von einem oder sehr wenigen Befragten genannt worden sind und die zugleich in der Praxis kaum oder nur mit sehr hohem Aufwand realisierbar wären, der Fall. Die User Stories werden nach Anzahl der Nennungen und themenspezifisch priorisiert aufgelistet. Die finale Priorisierung bei der Aufnahme in das Product Backlog erfolgt nach Abstimmung mit den User Stories aus den anderen Quellen der Anforderungsermittlung. Die Mitschriften der Befragungen befindet sich im Anhang (11.4).

- Die App sollte wenige Berechtigungen beim Zugriff auf persönliche Daten haben.
- Die App sollte werbefrei sein.
- Die App sollte reibungslos funktionieren.
- Die App sollte kostenlos sein.
- Die App sollte schnell Ergebnisse liefern.
- Die App sollte dem Nutzer die Möglichkeit bieten, das Design personalisiert einzustellen.
- Die App sollte fähig sein, das Attribut Ambiente in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, in die Bewertung einer Location einfließen zu lassen, ob die Musik der Location entsprechen angepasst ist.
- Die App sollte fähig sein, das Attribut *Personal/Gastfreundlichkeit* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Sauberkeit* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Preisniveau* in die Bewertung einer Location einfließen zu lassen.

- Die App sollte fähig sein, das Attribut *Gäste/Umfeld* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Verkehrsanbindung* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Auslastung* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Qualität des Essens* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *freie Plätze* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Preis/Leistungsverhältnis* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Angebote/Specials* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Altersbeschränkungen* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte fähig sein, das Attribut *Rauchfrei* in die Bewertung einer Location einfließen zu lassen.
- Die App sollte dem Nutzer die Möglichkeit bieten, Bewertungen über Sterne abzugeben.
- Die App sollte dem Nutzer die Möglichkeit bieten, Bewertungen über eine Skala abzugeben.
- Die App sollte fähig sein, Bewertungen über eine Gesamtlocation und über einzelne Attribute anzunehmen.
- Die App sollte fähig sein, die Kategorie *Disco* als Empfehlung rauszugeben.
- Die App sollte fähig sein, die Kategorie *Kneipe/Bar* als Empfehlung rauszugeben.
- Die App sollte fähig sein, die Kategorie *Restaurant* als Empfehlung rauszugeben.
- Die App sollte fähig sein, per Push-Funktion den Standort des Users zu ermitteln.
- Die App sollte dem Nutzer die Möglichkeit bieten, bei einer Push-Funktion zur Ermittlung des Standortes die Personen zu selektieren, die diese Information erhalten.
- Die App sollte dem Nutzer die Möglichkeit bieten, eine Bewertung optional über ein Textfeld abzugeben.
- Die App sollte fähig sein, ein Budget-Limit als Kriterium für die Empfehlung einzuberechnen.
- Die App sollte fähig sein, dem Nutzer per Push-Funktion mitzuteilen, wenn favorisierte Locations Specials und Angebote haben.

- Die App sollte dem Nutzer die Möglichkeit bieten, die Stadt, für die Empfehlungen gegeben werden sollen, unabhängig vom jeweiligen eigenen Standort auszuwählen.
- Die App sollte dem Nutzer die Möglichkeit bieten, eine empfohlene Location direkt zu kontaktieren.

Bei den Attributen für eine Location sind nicht berücksichtigt: *Randalefrei/Niveau*, *Auswahl*, *Anwesenheit von Bekannten Personen* sowie *Alleinstellungsmerkmal/Spezielles*. *Atmosphäre*, *Ambiente* und *Charme* sind aufgrund inhaltlicher Überschneidung und nicht eindeutiger Abgrenzungsmöglichkeit unter dem Attribut *Ambiente* zusammengefasst. Die Kategorien *Bar* und *Kneipe* werden aus den gleichen Gründen als eine Location aufgelistet. Die Frage nach Beispielen für Locations sollte potenziell umsetzbare und interessante Locations, die bis dahin nicht in Betracht gezogen worden sind, zur Diskussion stellen. Zahlreiche von den Befragten genannte Beispiele, wie *Tankstelle*, *Freizeitpark* oder *Turnhalle* bieten sich für den Anwendungsfall einer Empfehlungs-App jedoch nicht an und werden daher nicht als User Story aufgeführt. Einige Kriterien und Anregungen, die mehrfach genannt sind, werden nicht in die Liste mitaufgenommen, während seltener genannte Aussagen als User Story formuliert werden können. Dies macht Sinn, wenn Anregungen einfach zu implementieren sind, anderen Anforderungen nicht widersprechen oder trotz seltener Nennung die App bereichern könnten. *Standort des Nutzers ermitteln* ist als mögliche Funktionalität der App zwar von sechs Befragten genannt worden, allerdings hatte eine Vielzahl der Befragten datenschutzbezogene Sorgen hinsichtlich dieser Funktion. Die User Story *Die App sollte dem Nutzer die Möglichkeit bieten, eine empfohlene Location direkt zu kontaktieren* wurde zwar nur einmal genannt, lässt sich allerdings voraussichtlich leicht umsetzen und wäre insbesondere bei Empfehlungen für Restaurants eine nützliche Funktion, um Reservierungen und freie Plätze zu erfragen. Gleiches gilt für die Story *Die App sollte dem Nutzer die Möglichkeit bieten, die Stadt, für die Empfehlungen gegeben werden sollen, unabhängig vom jeweiligen eigenen Standort auszuwählen*. Laut vielen Befragten besteht das Bedürfnis nach Empfehlungen insbesondere dann, wenn sich der Nutzer an einem Standort aufhält, an dem er sich nicht auskennt. Wenn beispielsweise ein Trip in eine fremde Stadt geplant ist, könnte es Sinn machen, von der App im Vorhinein für eben diese Stadt Empfehlungen zu bekommen.

4.5.2 Probleme

Bei der Durchführung der Umfrage mittels des problemzentrierten Interviews sind unterschiedliche Schwierigkeiten aufgetreten. Zahlreiche potenzielle Interviewpartner konnten aufgrund von individuell begrenzten Zeitfenstern entweder gar nicht oder nur sehr eingeschränkt befragt werden. Dadurch ist es schwer gewesen, als Interviewer adäquat zurückzuspiegeln, Verständnisfragen zu stellen oder die Befragten mit ihren Aussagen zu konfrontieren. In den Fällen, in denen diese Methoden nicht angewandt werden konnten, fällt die Interpretation der Aussagen daher schwieriger. Die Informationen aus dem Block am Ende jedes Interviews (Alter, Berufsgruppe, Wohnort) sind dagegen hilfreich für die Interpretation. Aussagen lassen sich dadurch besser relativieren und verstehen.

Ebenfalls aus zeitlichen Gründen seitens der Befragten sind nicht in allen Interviews die im Vorhinein überlegten Kernfragen angesprochen worden. Dadurch sinkt der Inhaltliche Gehalt dieser Interviews. Die Befragungen sind in der Stadt Oldenburg und im Ort Schillighoern in Niedersachsen durchgeführt worden. Im Gegensatz zu Interviews in der Stadt hat es sich im Urlaubsort Schillighoern angeboten, Urlauber zu interviewen, da diese situationsbedingt keine größeren zeitlichen Beschränkungen hatten. In beiden Fällen hat es lange gedauert, Interviewpartner zu finden und zu befragen. Durch die begrenzten Kapazitäten der Interviewer liegt die Anzahl der mittels des problemzentrierten Interviews Befragten

bei 37. Das genügt zwar nicht den Anforderungen an eine repräsentative Umfrage, liefert der Projektgruppe aber dennoch nützliche Anregungen und Hilfe bei der Priorisierung diverser Anforderungen. Ein weiteres Problem stellt der zeitliche Aufwand des Extrahierens von konkret formulierten User Stories dar. Dies liegt am Konzept mit offenen Fragen und an der kontextabhängigen Interviewstruktur, die eine Vergleichbarkeit der Inhalte und insbesondere das Aggregieren von Gemeinsamkeiten aufwändig machen. Ein Beispiel dafür sind die unterschiedlich verwendeten Begrifflichkeiten. Für manche Befragte gibt es zwischen Ambiente und Atmosphäre keinerlei Unterschied, andere erachten eine Unterscheidung als sehr wichtig und sinnvoll. Die abschließende Kategorisierung der Antworten nahm viel Zeit in Anspruch und gestaltete sich als schwierig.

4.6 Interview mit der Projektleitung

Um die Informationen aus dem Kick-off-Meeting zu konkretisieren und Missverständnissen vorzubeugen, wird ein weiteres Interview mit dem Projektleiter geführt. Ausgangspunkt für die Befragung ist die Abbildung 4.2 aus dem Kick-off-Meeting (Kapitel 4.1) und die daraus entstehenden Fragen für die Projektgruppe. Der ausgefüllte Fragebogen ist in Abschnitt 11.2 zu finden. Das Interview besteht aus 14 Fragen. Hiervon sind 12 Fragen vor dem Interview und 2 weitere Fragen im Dialog mit dem Projektleiter entstanden:

1. Können Sie Ihre Ideen noch einmal grob umreißen?

   Ziel: Die erste Frage ist eine Einstiegsfrage und dient dazu, den Einstieg in das Interview zu erleichtern. Der Projektleiter soll kurz seine Vision darstellen und noch einmal einen Überblick über die Problemstellung geben.

   Antwort: Es soll ein *Showcase* entwickelt werden, mit welchem der Funktionsumfang von Odysseus demonstriert wird. Die fiktive Hintergrundgeschichte des Auftrags an die Projektgruppe ist es hierbei, dass ein Unternehmen mit dem Produkt Odysseus einen neuen Anwendungsfall in der Domäne RecSys abdecken möchte und diese Funktionalität vermarktet werden kann. Damit dies möglichst interaktiv geschieht, soll das Showcase so realistisch wie möglich gehalten werden. Es ist gegebenenfalls nötig, die Funktionalität von Odysseus in Form von neuen Operatoren, Algorithmen oder ähnlichem zu erweitern. Beispielsweise ist im vergangenen Jahr die Peer-to-Peer-Funktionalität von Odysseus durch das System *Herakles*[6] demonstriert worden. Das System ermöglicht Liveanalysen im Sport mit Odysseus. Wie auch bei Herakles wäre eine App sehr wünschenswert.

2. Was verstehen Sie unter dem Begriff *personalisierte Empfehlung*?

   Ziel: Mit dieser Frage soll der Begriff *personalisierte Empfehlung* genauer definiert werden.

   Antwort: Die Empfehlung sollte auf eine Person zugeschnitten sein. Dies bedeutet, dass es eine spezifische Empfehlung für diese Person ist. Eine unspezifische Empfehlung wäre beispielsweise, dass das Produkt empfohlen wird, das am meisten verkauft wird. Es gibt verschiedene Algorithmen, mit welchen dies sichergestellt werden kann. Das bedeutet aber nicht, dass kontextbasierte Informationen interpretiert werden müssen. Dies wäre zwar positiv, wird aber nicht zwingend für eine personalisierte Empfehlung benötigt.

[6] Projektgruppenabschlussbericht *LSOP*: Liveanalysen im Sport mit Odysseus P2P, `http://www.uni-oldenburg.de/fileadmin/user_upload/informatik/download/studium/pg/2015-LSOP.pdf`, besucht am 11.06.2015.

3. Haben Sie schon eine Idee, welche Informationen für einen Administrator wichtig sein könnten? Haben Sie schon Erfahrungen gemacht?

   Ziel: Mit dieser Frage soll die Rolle des Admins näher definiert werden. In der Projektgruppe gab es Diskussionen, welche Informationen ein Admin benötigt. Diese können von Informationen über die Nutzerzahlen oder die Güte der Empfehlungen bis hin zu Informationen über die Hardware oder die Auslastung des Systems reichen.

   Antwort: Dem Administrator soll die Güte der Empfehlung angezeigt werden. Dieses kann beispielsweise durch den Root Mean Squared Error (RMSE) geschehen. Schön wäre auch die Anzeige der Klickrate. Dieses soll grafisch ansprechend aufbereitet werden. Ein Dashboard wäre nützlich. Hier könnte der Administrator unter anderem verschiedene Konfigurationen ausprobieren. Durch den Vergleich von zwei Lernalgorithmen könnte beispielsweise die Güte der Empfehlungen verbessert werden.

4. Was verstehen Sie unter dem Begriff *Kontext-Reasoning*?

   Ziel: In der Abbildung aus dem Kick-off-Meeting wurde der Begriff Kontext-Reasoning genannt. Damit die Projektgruppe ein eindeutiges Verständnis von diesem Begriff hat, soll der Projektleiter diesen noch einmal präzise definieren.

   Antwort: Unter Kontext-Reasoning verstehe ich das Interpretieren der Situation, in welcher sich Nutzer gerade befinden. Kontextdaten sollen zu Kontextinformationen verarbeitet werden. Diese Informationen sollen die Empfehlung, welche der Person gegeben wird, beeinflussen. Es wäre schön wenn dies umgesetzt wird, ist jedoch nicht verpflichtend.

5. Haben Sie eine oder mehrere Funktionalitäten, die Sie gerne für den Administrator bereitstellen möchten?

   Ziel: Hier soll noch einmal genauer auf die gewünschten Funktionalitäten für den Admin eingegangen werden.

   Antwort: Die Funktionalitäten sind bereits in Frage 3 beantwortet worden. Dem Administrator sollen wichtige Informationen über das System zur Verfügung stehen. Er soll das System anpassen und optimieren können. Ein Vergleich zwischen verschiedenen Algorithmen wäre gut.

6. Haben Sie eine oder mehrere Funktionalitäten, die Sie gerne für den Benutzer bereitstellen möchten?

   Ziel: Es soll herausgefunden werden, ob der Projektleiter schon eine konkrete Vorstellung der App hat.

   Antwort: Kernfunktionalität soll das Abgeben von Bewertungen und das Erhalten von Empfehlungen sein. Die Empfehlungen sollen dem Nutzer angezeigt werden und er soll eine optimale Entscheidung treffen können. Da die App sehr persönliche Daten verarbeitet ist es wahrscheinlich sehr wichtig, dem Nutzer möglichst viele Informationen zur Verfügung zu stellen, damit die Verarbeitung der Daten möglichst transparent für ihn ist.

7. Was verstehen Sie unter einer DSL?

   Ziel: Der Begriff DSL sollte genauer definiert werden, um herausfinden, wie wichtig dem Projektleiter eine eigene Sprache ist.

Antwort: DSL ist eine Möglichkeit das RecSys zu konfigurieren. Durch einen eigenen Editor könnten z. B. Anfragen verändert werden, sodass sich der Administrator nicht mehr in den Kontext Odysseus einarbeiten muss. Dies ist jedoch nur eine Idee und wichtig ist erst einmal, dass der Nutzer Bewertungen abgeben kann und Empfehlungen angezeigt bekommt.

8. Was sind Ihrer Meinung nach die wichtigsten Funktionalitäten eines Dashboards?

   Ziel: Es soll noch einmal genauer auf gewünschte Funktionen des Dashboards eingegangen werden.

   Antwort: Diese Frage wurde schon im Zuge der Frage 3 und 5 beantwortet. Das Dashboard muss keine eigenständige App sein. Der Odysseus-Client bietet bereits Funktionalitäten, die für das Dashboard eventuell genutzt werden können. Die komfortable Konfiguration und Auswertung über ein Backend ist nicht so wichtig wie die Nutzerfunktionalität des Systems.

9. Gibt es die Möglichkeit sich die Administrations-Sicht bzw. Analyse-Sicht auf ein RecSys anzugucken?

   Ziel: Um die vorhandenen Vorstellungen eines RecSys-Dashboard zu konkretisieren, wird nach Beispielen gefragt.

   Antwort: Ich kenne leider kein RecSys-Dashboard.

10. Welche Aufgaben hat aus Ihrer Sicht der Administrator?

    Ziel: In der Projektgruppe gab es Unklarheiten über den Zuständigkeitsbereich des Admins.

    Antwort: Der Administrator ist eher aus technischer Sicht zu betrachten. Er ist kein Wirtschaftler, welcher Kennzahlen auswerten kann. Er soll eine gute Übersicht über das Gesamtsystem haben.

11. Für welche Rolle sollen die Informationen auf dem Dashboard bereitgestellt werden?

    Ziel: In der Projektgruppe gab es Unklarheiten über den Zuständigkeitsbereich des Admins.

    Antwort: Dieses wurde schon implizit durch andere Fragen beantwortet.

12. Was ist für Sie wichtig (Offene Frage)?

    Ziel: Diese Frage ergab sich aus dem Interview.

    Antwort: Wichtig ist die Interaktion zwischen Odysseus und der Anwendung des Nutzers sowie das Ausgeben von Empfehlungen. Es soll ein „Showcase“ sein. Der administrative Teil ist nur eine Komfort-Funktion und soll nicht im Fokus stehen.

13. Muss der Datenschutz beachtet werden (Offene Frage)?

    Ziel: Da dieses Projekt nur einen Prototypen erstellt, ist der Datenschutz hinderlich. Dieses bedeutet nicht, dass er nicht wichtig ist, jedoch bedeuten z. B. eine gehärtete Verschlüsselung auch einen Mehraufwand.

    Antwort: Ziel ist es, einen Prototypen zu entwickeln. Dies bedeutet, dass er nicht der Öffentlichkeit zugänglich gemacht wird. Datenschutz steht somit nicht im Mittelpunkt des Projekts. Sollte der Prototyp in eine offene Evaluation gehen, müsste über dieses Thema noch einmal diskutiert werden. An wichtigen Stellen soll jedoch die Dokumentation auf sicherheitskritische Implementationen hinweisen.

Durch das Interview werden wichtige Fragen der Projektgruppe beantwortet. Vor allem die Rolle des Admins ist nun klarer definiert. Der Fokus ist die Entwicklung eines *Showcases* in welchem die Funktionalität von Odysseus in einem neuen Anwendungsfall dargestellt ist. Folgende potenzielle Anforderungen ergeben sich aus dem Gespräch:

- Der Admin sollte den RMSE einsehen können.
- Der Admin sollte die Klickrate angezeigt bekommen.
- Dem Admin sollte ein Dashboard zur Verfügung stehen.
- Der Admin sollte das System konfigurieren können.
- Der Admin sollte verschiedene Lernalgorithmen vergleichen können.
- Der Nutzer sollte Empfehlungen über eine Smartphone App erhalten.
- Der Nutzer sollte Bewertungen über eine Smartphone App abgeben können.
- Der Nutzer sollte über die Verarbeitung der Daten informiert werden.
- Zur Konfiguration des Systems könnte eine DSL genutzt werden.

4.7 Analyse von wissenschaftlichen Quellen

Im Rahmen der Anforderungsanalyse findet eine Literaturrecherche statt, um Anforderungen an das Dashboard zu ermitteln. Das Resultat ist, dass es wenig konkrete Veröffentlichungen zu den Funktionen und Anforderungen an ein Dashboard von RecSys gibt. Die Hauptaufgabe eines Dashboards ist das Monitoring des RecSys. Es gibt drei wesentliche Veröffentlichungen zum Monitoring von RecSys:

Félix et al. [FSJV14] beschreiben einen Business-Intelligence- Ansatz zum Monitoring von RecSys. Hierbei wird hauptsächlich versucht, ein RecSys mit einem Data-Warehouse zu verbinden. Dafür werden unter anderem Anforderungen an das Monitoring von RecSys erhoben und beispielhaft Funktionen vorgestellt. So ist die Überwachung der akzeptierten und abgelehnten Empfehlungen für bestimmte Zeitperioden wichtig. Des Weiteren sollte das RecSys verschiedene Profile mit verschiedenen Einstellungen abspeichern können, so dass ein direkter Vergleich der Performance beider Einstellungen möglich ist. Als Beispiel gehen Félix et al. auf verschiedene Analysen der akzeptierten bzw. abgelehnten Empfehlungen ein. So können die Geschlechterverteilung, die Länderverteilung und Verteilung über den Tag hinweg betrachtet werden. Darüber hinaus ist es möglich, verschiedene Einstellungsprofile für das RecSys abzuspeichern. Folglich kann schnell zwischen verschiedenen Einstellungsprofilen gewechselt und ein Vergleich dieser durchgeführt werden, um die optimale Konfiguration zu finden.

Ronen et al. 2013 [RKZ+13] beschreiben das RecSys mit dem Namen *Sage*. Es ist eine Entwicklung von *Microsoft*. Es wird beschrieben, wie ein RecSys als Cloud-Service umgesetzt werden kann. In diesem Paper wird hauptsächlich auf das Training des Models eingegangen. Es werden zudem Anforderungen an das Dashboard erläutert. Verschiedene Metriken, wie z. B. die Akzeptanz der Empfehlungen, können so betrachtet werden.

Ben-Shimon et al. 2014 [BSATFH14] beschreiben ein Vorgehen, das Monitoring eines RecSys als Service zu implementieren. Bei dem Service handelt es sich um *YooChoose*. Es wird der grobe

Funktionsumfang von YooChoose dargestellt. In der umfangreichen Dokumentation von YooChoose finden sich sehr viele Informationen über die Funktionalitäten von Dashboards. Diese Dokumentation ist die größte Quelle von Anforderungen an ein Dashboard, da sie detailliert den Aufbau und die Funktionen eines solchen Dashboards beschreibt.

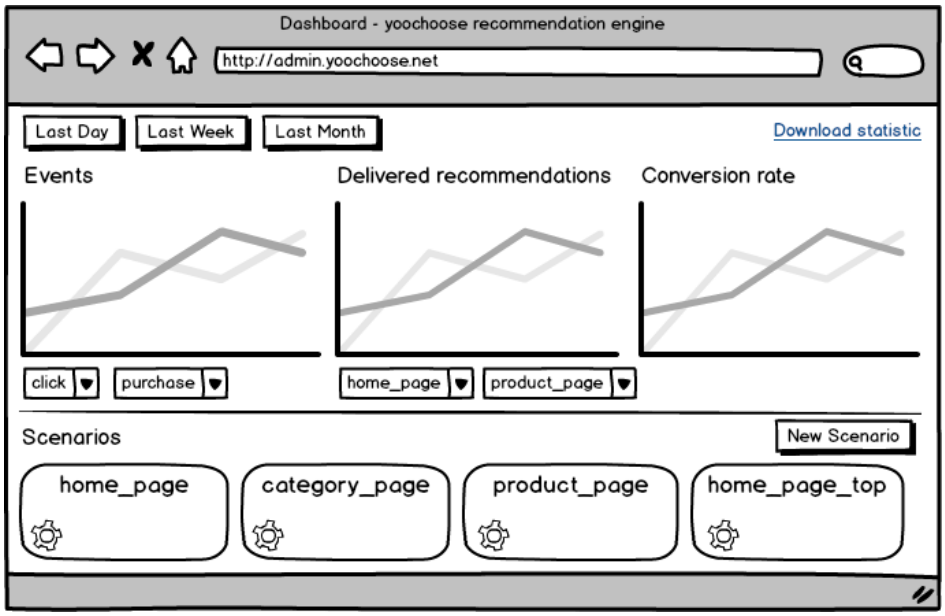


Abbildung 4.5: Darstellung des YooChoose Dashboards zum Monitoring von RecSys. Quelle: [YOO13, S. 5].

Grundlegend lässt sich das Dashboard in die grafische Aufbereitung der verschiedenen Informationen bzw. Metriken und den sogenannten Szenarien unterteilen (siehe Abbildung 4.5). Es lassen sich Statistiken für verschiedene Zeiträume anzeigen. Auch hier lässt sich die Akzeptanz der Empfehlungen darstellen. Diese Statistiken lassen sich herunterladen. Des Weiteren kann eingesehen werden, wie viele Empfehlungen gegeben wurden und wie viele Events eingetroffen sind. Zu den Events gehören explizite und implizite Bewertungen. Mit Szenarien lässt sich einstellen, wie die Empfehlungen gegeben werden. Jedes Szenario beschreibt einen gewissen Kontext. Beispielsweise können Produkte oder Produktgruppen ausgeschlossen werden oder es lässt sich anzeigen welche Produkte andere Kunden in diesem Zusammenhang kaufen [YOO13].

Da in jeder Veröffentlichung die Akzeptanz von Empfehlungen ein wichtiges Thema ist, wird nach Veröffentlichungen, die von Metriken und Evaluation von RecSys handeln, gesucht. Dabei sticht die Arbeit von *Schröder et al.* [Sa11] hervor. Sie geht detailliert auf die Akzeptanz und Genauigkeit von Empfehlungen ein. Grundlegend lassen sich Empfehlungen in vier Kategorien einteilen, wie es in Tabelle 4.1 zu sehen ist.

Wenn eine Empfehlung gegeben und vom Nutzer akzeptiert wird, ist sie *true-positive*. Falls eine Empfehlung gegeben, aber vom Nutzer nicht akzeptiert wird, handelt es sich um *false-positive*. Falls jedoch eine Empfehlung nicht gegeben wird und sie für den Nutzer dennoch relevant ist, handelt es sich um *false-negative*. Falls eine Empfehlung nicht gegeben wird und sie auch nicht für den Nutzer relevant ist, ist es *true-negative*. Es gilt also die Menge von false-negative und false-positive möglichst

	Relevant	Irrelevant	Total
Empfohlen	$true - positive$	$false - positive$	$tp + fp$
Nicht Empfohlen	$false - negative$	$true - negative$	$fn + tn$
Total	$tp + fn$	$fp + tn$	N

Tabelle 4.1: Wahrheitstabelle für Empfehlungen. Quelle: [Sa11].

zu minimieren. Ausgehend von dieser Einteilung lassen sich nun verschiedene Metriken, die die Genauigkeit des RecSys angeben, bestimmen. Zu den Metriken gehören *Precision*, *Recall* und *Fallout*.

Die Precision gibt das Verhältnis zwischen allen empfohlenen Objekten und den empfohlen und tatsächlich relevanten Objekten an.

$$precision = \frac{tp}{tp + fp}$$

Der Recall gibt das Verhältnis zwischen allen empfohlenen und allen relevanten Objekten an. Damit lässt sich ermitteln, wie gut relevante Objekte erkannt werden.

$$recall = \frac{tp}{tp + fn}$$

Der Fallout gibt das Verhältnis zwischen empfohlenen, irrelevanten Objekten und der Gesamtheit aller irrelevanter Objekte an. Damit lässt sich ermitteln, wie gut irrelevante Objekte erkannt werden.

$$fallout = \frac{fp}{fp + nt}$$

Diese drei Metriken stellen die wichtigsten Metriken dar. Neben der Klassifikation der Empfehlungen gibt es noch die Möglichkeit, die Genauigkeit des RecSys mit dem RMSE zu ermitteln [Sa11]. Zusammenfassend lässt sich zu den wissenschaftlichen Quellen sagen, das immer wieder die selben Anforderungen an ein Dashboard eines RecSys gestellt werden. So ist die Überwachung der Genauigkeit der Empfehlungen von hoher Bedeutung und als potenzielle Anforderung aufzunehmen. Zusätzliche ist im Zusammenhang mit der Datenstromverarbeitung die durchschnittliche Dauer zwischen Empfehlungsanfrage und Empfehlungserstellung wichtig. Darüber hinaus bietet YooChoose einen hilfreichen Einblick in ein reales RecSys. Es zeigt, dass eine Export-Funktion der verschiedenen Metriken und Statistiken vorteilhaft sein kann. Auf das Erstellen von Szenarien kann jedoch verzichtet werden. Diese Funktion ist vorhanden, um einen möglichst generischen Einsatz von YooChoose zu gewährleisten. Für die Projektgruppe genügt die Möglichkeit, verschiedene Einstellungen am Recommender-Algorithmus vorzunehmen, da die Menge der Szenarien begrenzt ist. Im Folgenden ist eine Liste aller aus den Quellen resultierenden, potenziellen Anforderungen aufgeführt:

- Das Dashboard sollte dem Admin die Möglichkeit bieten, den Verlauf des RMSE für einen bestimmten Zeitraum einsehen zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, die Einstellung verschiedener Parameter des Recommender-Algorithmus verändern zu können.

- Das Dashboard sollte dem Admin die Möglichkeit bieten, die angenommenen und abgelehnten Empfehlungen einsehen zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, die Locations einsehen zu können, die trotz fehlender Empfehlung besucht werden.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, den Recall für einen bestimmten Zeitraum und nach verschiedenen Filtern einsehen zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, die Precision für einen bestimmten Zeitraum und nach verschiedenen Filtern einsehen zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, den Fallout für einen bestimmten Zeitraum und nach verschiedenen Filtern einsehen zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, alle Ergebnisse der Metriken exportieren zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, die eintreffenden Bewertungen einsehen zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, verschiedene Einstellungsprofile für das RecSys abspeichern zu können.
- Das Dashboard sollte dem Admin die Möglichkeit bieten, den durchschnittlichen Zeitraum zwischen Empfehlungsanfrage und Empfehlungserstellung einsehen zu können.

4.8 Sichtung von Konkurrenzprodukten

Um weitere Anforderungen an die App zu ermitteln, werden verschiedene Konkurrenzprodukte gesichtet. Unter den Produkten sind *Yelp*, *Yeti*, *TripAdvisor* und *FourSquare*. Die Apps werden analysiert und auf ihre Funktionsweise hin überprüft. Es wird vor allem nach Alleinstellungsmerkmalen und Besonderheiten, die die App in ihrer Art von den anderen Produkten abheben, Ausschau gehalten. Aus den Funktionalitäten der Apps werden zunächst potenzielle Anforderungen an die Recommender-App abgeleitet und zur Diskussion gestellt. Anschließend werden konkrete Anforderungen formuliert und als Funktionen aufgenommen. Jede App wird in einem kurzen Absatz vorgestellt und durch die Auflistung potenzieller Funktionalitäten ergänzt. Die Auswahl der Apps erfolgt anhand der Popularität im *Google PlayStore*[7] und *AppStore*[8].

4.8.1 Yelp

Die App *Yelp*[9] des amerikanischen Internetunternehmen Unternehmen *Yelp, Inc.* unterstützt den Nutzer bei der Suche nach Locations in seiner Nähe. Auf Basis von Nutzerbewertungen und Reviews

[7] *Google PlayStore*: Software bzw. Webseite zum Herunterladen von Android Apps, `https://play.google.com/store`, besucht am 28.07.2015.
[8] *AppStore*: Software bzw. Webseite zum Herunterladen von Apple Apps, `https://itunes.apple.com/de/genre/ios/id36?mt=8`, besucht am 28.07.2015.
[9] *Yelp*: Finden von Orten zum Essen, `https://play.google.com/store/apps/details?id=com.yelp.android`, besucht am 29.06.2015.

werden dem Suchenden auf Grundlage seiner Suchanfragen die Locations mit den besten Bewertungen angezeigt. Die App ermöglicht das Selektieren nach Kategorie, Städteviertel, Entfernung, Preis und Öffnungszeiten. Der Suchbegriff wird über ein Textfeld eingegeben. Locations, die mit dem Begriff in Verbindung gebracht werden, werden in einer Liste, nach Relevanz sortiert, angezeigt. In der Liste werden die Locations mit Titelfoto, Namen, Anzahl der Rezensionen, Durchschnittsbewertung in Form von Sternen, Adresse und Tags angezeigt. Zusätzlich wird eine grobe Kategorisierung nach dem Preisniveau der Locations mithilfe von Dollarzeichen vorgenommen. Der Nutzer kann direkt Kontakt zu für ihn relevanten Locations aufnehmen und sich von der App per Global Positioning System (GPS) den Weg zur Location anzeigen lassen. In einer separaten Ansicht lassen sich Rezensionen und Beiträge oder Fotos anzeigen. Selbige können vom Nutzer hinzugefügt und auf der Aktivitäten-Seite des Nutzers angezeigt werden. Außerdem sind in der App versteckte Funktionen und Inhalte (*Easterreggs*) enthalten. Potenziell relevante Anforderungen für die Recommender-App der Projektgruppe lassen sich wie folgt beschreiben:

- Die App sollte in der Liste, in der die Empfehlungen angezeigt werden, Titelbild, Name, Adresse, Anzahl der Reviews sowie die Durchschnittsbewertung aufführen.
- Die App sollte in der Liste, in der die Empfehlungen angezeigt werden, über Symbole die Preiskategorie der einzelnen Empfehlungen anzeigen.
- Die App sollte die Empfehlungen in einer Liste, nach ihrer Relevanz sortiert, anzeigen.
- Die App sollte eine Rezensionen-Funktionalität bereitstellen.
- Die App sollte Rezensionen über eine Location in einer separaten Liste anzeigen.
- Die App sollte Fotos einer Location in einer separaten Bildergalerie zu jeder Location hinterlegen.
- Die App sollte Easterreggs anbieten.
- Die App sollte eine Selektion nach Öffnungszeiten einer Location ermöglichen.

4.8.2 Yeti

Yeti[10] ist eine stark auf Bilder ausgerichtete Recommender-App für Locations. Sie ist in Bildkarten organisiert, die durch eine einfache Wischgeste als relevante oder irrelevante Empfehlung kategorisiert werden können. Zum Einstieg wird dem Nutzer dieser Wischmechanismus mit zehn Bildern von Locations aus der Umgebung näher gebracht. Yeti greift hier auf einen Gamification-Ansatz zurück und zeigt dem Nutzer einen Erfahrungsbalken, der sich pro Bild weiter füllt und ihm am Ende die Foto-Funktion freischaltet. Durch diesen Einstieg sammelt die App schon zu Beginn wertvolle Daten, um die Interessen des Nutzers besser einordnen zu können. In der Standardansicht der App ist zunächst der starke Fokus auf Bilder auffällig. Sie zeigt groß ein Foto einer interessanten Location in der Nähe und einen großen Button zur Aktivierung der eigenen Kamera. Der Button hebt sich in Größe und Farbe von allen anderen sichtbaren Buttons ab und ist dadurch leicht zu finden. Die Nutzer laden ihre eigenen Bilder hoch und berichten anderen Nutzern, was die Location besonders macht und warum sie einen Besuch wert ist. Durch einen Fingertipp auf das Bild kommt der Nutzer zu einer größeren

[10] *Yeti*: Recommender-App für Orte in der Nähe, `https://itunes.apple.com/de/app/yeti-discover-local/id882096384`, besucht am 29.06.15.

Ansicht, die eine Kommentarfunktion, eine Speicherfunktion, eine Share-Funktion, eine Karte und die Möglichkeit zum Liken des Bildes bietet. Darüber hinaus stellt die App die Möglichkeit bereit, einzelnen Nutzern zu folgen und über deren Aktivitäten auf dem Laufenden gehalten zu werden. Es ergeben sich folgende potenzielle Anforderungen:

- Die App sollte Kategorisierungen von Locations ermöglichen.
- Die App sollte eine abstrahierte Sicht ermöglichen, in der Locations als relevant und irrelevant markiert werden können.
- Die App sollte eine detaillierte Ansicht bieten, die weitere Aktionen zulässt und Kommentare einblendet.
- Die App sollte eine Follow-Funktion anbieten.
- Die App sollte es ermöglichen, eigene Bilder hochzuladen.

4.8.3 TripAdvisor

TripAdvisor[11] ist eine App mit der Reisen geplant und gebucht werden können. Die Anmeldung erfolgt über *Google+*[12], Facebook oder per TripAdvisor-Account. Die Standardsicht der App zeigt eine Liste mit großer Suchleiste. Die Suche nimmt einen großen Bereich im Kopf der App mit dem Schriftzug *Plan and Book Your Perfect Trip* ein und richtet den Fokus auf das eigenständige Suchen geeigneter Hotels, Restaurants oder anderer Locations. Unter der Suchleiste werden die letzten drei Städte, in denen eine Location angeschaut wurde, aufgeführt. Die App bietet zudem die Option, vorher eine Kategorie auszuwählen und anschließend nach Locations in der Umgebung oder einer ausgewählten Stadt zu suchen. Wird nach Locations in der Nähe gesucht, werden die Suchergebnisse nach Entfernung vom jeweiligen Standort sortiert. Die Location die am nächsten ist, wird als erstes aufgeführt. Wird eine eine Stadt ausgewählt, in der nach Locations zu gesucht werden soll, wird die Liste nach Bewertungen sortiert. In den Profileinstellungen zeigt die App eine Übersicht über die vom Nutzer favorisierten Städte, die Anzahl der geschriebenen Reviews, die Anzahl der eingestellten Photos sowie die Anzahl der erhaltenen Auszeichnungen und Erfolge in Form von Abzeichen (Badges). Die App erlaubt den Zugriff auf die Foren und die über TripAdvisor getätigten Buchungen. Es ergeben sich folgende potenzielle Anforderungen:

- Die App sollte Reservierungen in Resaturants oder ähnlichen Locations ermöglichen.
- Die App sollte eine standortunabhängige Suche von Locations ermöglichen.
- Die App sollte eine Auflistung der zuletzt gesuchten Locations ermöglichen.
- Die App sollte eine Bezahlfunktion ermöglichen.

[11] *TripAdvisor*: App zur Planung von Reisen, `https://play.google.com/store/apps/details?id=com.tripadvisor.tripadvisor`, besucht am 29.06.2015.
[12] *Google+*: Soziales Netzwerk des Unternehmens *Google Inc.*, `https://plus.google.com`, besucht am 29.06.2015

4.8.4 FourSquare

FourSquare[13] ist eine mobile Anwendung, die dem Nutzer standortunabhängig Locations empfiehlt. Zu den Locations gehören Restaurants, Einkaufsmöglichkeiten, Bars, Clubs sowie Unterhaltungsangebote. Beim Start der Anwendung werden dem Nutzer die zwei Restaurants und Bars mit den höchsten Bewertungen angezeigt. Des Weiteren werden Highlights in der Umgebung angezeigt. Über verschiedene Reiter kann der Nutzer seine Suche eingrenzen, indem er beispielsweise den Reiter *Restaurant* auswählt. Es werden die besten Restaurants in der Nähe gezeigt. Diese Suche lässt sich weiter detaillieren, indem der Radius verändert wird und Filter hinzugefügt werden. Es lässt sich nach Preis, Öffnungszeit und nach verschiedenen Besonderheiten filtern. In der Detailansicht der Location werden Kommentare, Bewertungen und der Standort angezeigt. Neben den Kategorien der Locations gibt es zusätzlich angesagte Locations und Favoriten, die der Nutzer eigenständig auswählen kann. Zusätzlichen lassen sich persönliche Vorlieben in Form von Tags eintragen und Locations bewerten. Außerdem wird eine Facebook und Google+ Anbindung angeboten, wodurch Vorlieben besser erkannt und Inhalte geteilt werden können. FourSquare bietet viele gute Ansätze für eine Empfehlungs-App. Es ergeben sich folgende potenzielle Anforderungen:

- Der Nutzer sollte persönliche Vorlieben eintragen können.
- Der Nutzer sollte Location bewerten können.
- Der Nutzer sollte Favoriten speichern können.
- Der Nutzer sollte den Radius der Empfehlungen anpassen können.
- Der Nutzer sollte die Empfehlungen nach verschiedenen Gesichtspunkten filtern können.
- Der Nutzer sollte sein *Google+* und *Facebook-Profil* mit der Anwendung verbinden können.
- Der Nutzer sollte über Social-Media-Inhalte teilen können.
- Der Nutzer sollte eine Detailansicht mit Kommentaren und dem Standort der Location enthalten.
- Der Nutzer sollte Kommentare für eine Location abgeben können.

4.9 Anforderungen aus dem Backlog Grooming

Die Anforderungsermittlung (siehe Kapitel 4) dient in erster Linie dazu, das *Product Backlog* zu füllen und Anforderungen für das gesamte Projekt zu sammeln. Das Product Backlog enthält sämtliche User Stories priorisiert. Die Projektgruppe muss, anders als in Scrum-Projekten üblich, zu Beginn selbst mit User Stories füllen. Die Ergebnisse sollen als Ausgangspunkt und Basis für weitere Anforderungen verstanden werden. Wie in Scrum üblich, werden bis zum Projektende kontinuierlich Anforderungen erhoben, verfeinert, aufgesplittet und priorisiert. Viele dieser neuen Anforderungen ergeben sich aus neu hinzugewonnen Kenntnissen und Notwendigkeiten, die im Vorhinein nicht abzusehen sind. Aus diesem Grund wird im Zuge des alle zwei Wochen stattfindenden *Backlog Groomings* ermittelt, ob neue User Stories in das Backlog aufgenommen werden müssen. Das Backlog Grooming dient allgemein

[13] *FourSquare*: App zum finden und Entdecken von Orten, `https://play.google.com/store/apps/details?id=com.joelapenna.foursquared`, besucht am 29.06.2015.

dem Priorisieren von User Stories, der Spezifizierung und verständlicheren Formulierung derselben, dem Identifizieren und Formulieren neuer Stories sowie allgemein dem Prüfen und Überarbeiten des Product Backlogs. Sämtliche neue User Stories, die nicht der Anforderungsermittlung zu Projektbeginn entstammen, werden in diesem Kapitel aufgelistet und in 5.2 oder 5.3 als konkrete Anforderungen mit Referenznummer formuliert.

4.9.1 Benutzerverwaltung

Aus dem Backlog Grooming am 18.06.15 in dem die potenziellen Anforderungen aus Kapitel 4 betrachtet und ausgewählt wurden, ergibt sich, dass eine Benutzerverwaltung benötigt wird. Durch die Benutzerverwaltung kann sich der Nutzer registrieren, anmelden, sein Passwort verändern bzw. zurücksetzen, sein Profilbild festlegen und ändern sowie seinen Benutzernamen und seine E-Mail-Adresse ändern.

Damit der Nutzer die App nutzen kann, muss er sich zuerst anmelden. Voraussetzung dafür ist seine Registrierung. Dies soll einfach über die Angabe eines Passworts, eines Benutzernamens und einer E-Mail-Adresse erfolgen. Um Funktionen wie das Änderns des Passworts realisieren zu können, ist es notwendig, dass die angegebene E-Mail-Adresse des Nutzers eine reale Adresse ist.

- Die App sollte es dem Nutzer ermöglichen sich zu registrieren.
- Die App sollte es dem Nutzer ermöglichen sich anzumelden.
- Die App sollte es dem Nutzer ermöglichen sein Passwort zu verändern.
- Die App sollte es dem Nutzer ermöglichen seine E-Mail-Adresse zu verändern.
- Die App sollte es dem Nutzer ermöglichen seinen Benutzernamen zu verändern.
- Die App sollte es dem Nutzer ermöglichen sein Passwort zurückzusetzen.
- Die App sollte es dem Nutzer ermöglichen ein Profilbild anzulegen.
- Die App sollte es dem Nutzer ermöglichen sein Profilbild zu ändern.

4.9.2 Benutzerfreundlichkeit

Bei der Entwicklung der App sollten allgemeine Prinzipien der Benutzerfreundlichkeit beachtet werden. Der Fokus der Projektarbeit liegt allerdings auf Anforderungen, die direkt mit dem DSMS in Verbindung stehen, z.B. Empfehlungsqualität oder Antwortzeiten. Daher wird sich die Projektgruppe aus Kapazitätsgründen lediglich in allgemeiner Form auf Anforderungen hinsichtlich Benutzerfreundlichkeit festlegen. Die wichtigsten Operationen sollen direkt sichtbar und erreichbar sein und der Nutzer soll sich vorhersehbar durch die Funktionalitäten der App navigieren können. Die Schlüsselbereiche, z.B. die Startseite, sollen von jeder Stelle aus schnell erreichbar sein. Um eine einfache und benutzerfreundliche Navigation zu ermöglichen wird eine Menüsteuerung implementiert. Dies ermöglicht ein übersichtliches und schnelles Wechseln zwischen den verschiedenen Sichten. Weiterhin sollen Inhalte mit hoher Priorität zentral und gut sichtbar positioniert werden. Sämtlicher Text soll gut lesbar sein, nicht überlappen und konsistent in Sprache und Schriftart sein. Da es sich um einen Prototyp

handelt, wird auf Mehrsprachigkeit verzichtet. Style-bezogene Anforderungen, wie Skalierung für unterschiedliche Endgeräte, weitere Anforderungen an die Lesbarkeit, z.B. Kontraste und Farben, sowie Benutzerführung über Text Labels, Tooltips oder ein Hilfe-Menü, stehen zunächst nicht im Vordergrund und werden nicht als Anforderung aufgenommen.

- Die App sollte es dem Nutzer ermöglichen, sich vorhersehbar durch die App zu navigieren.
- Die wichtigsten Operationen sollten in der App schnell ersichtlich sein.
- Die Schlüsselbereiche der App sollten von jeder Stelle aus schnell erreichbar sein.
- Die App sollte es dem Nutzer ermöglichen, sich über eine Menüsteuerung durch die App zu navigieren.
- Inhalte mit hoher Priorität sollten in der App zentral und gut sichtbar positioniert werden.
- Die App sollte Text gut lesbar darstellen.
- Die App sollte Text nicht überlappend darstellen.
- Die App sollte Text in konsistenter Sprache darstellen.
- Die App sollte Text in konsistenter Schriftart darstellen.

4.9.3 Wartungsfreundlichkeit

Das System sollte es gewährleisten, dass potenzielle Fehlerquellen durch die Entwickler ohne großen Aufwand auffindbar sind. Dafür soll es eine solide Testabdeckung ebenso wie eine sinnvolle Fehlerbehandlung in sämtlichen verwendeten Systemen und Komponenten geben. Unterstützend sollte eine geeignete Architektur gewählt werden, die gekapseltes Eingreifen zur Fehlerbehandlung ermöglicht. Die Projektgruppe versucht zudem, ausgewählte Werkzeuge und Frameworks unterstützend für die Projektarbeit einzusetzen, z.B. über Frameworks für *objektrelationales-Mapping* oder *Dependency-Injection*.

- Das System sollte es ermöglichen, Fehlerquellen einfach zu identifizieren.
- Das System sollte über eine solide Testabdeckung verfügen.
- Die System-Architektur sollte Fehlerbehandlung vereinfachen.

4.9.4 Erweiterbarkeit

Das agile Vorgehen der Projektgruppe impliziert eine iterative Entwicklung des Produkts. Dadurch ergibt sich die Anforderung, die Architektur so zu gestalten, dass eine iterative Entwicklung im Projektverlauf einfach umgesetzt werden kann und damit neue Funktionalitäten problemlos hinzugefügt werden können. Das Projekt ist außerdem als Prototyp zu verstehen. Daher ist es aus Sicht der Projektgruppe sinnvoll, auf Erweiterbarkeit zu achten, um ggf. für nachfolgende Arbeiten und Projekte eine gute Grundlage zu bilden. Erweiterbarkeit soll unter anderem durch den Einsatz generischer Entwurfsmuster, die einen zentralisierten Zugriff auf unterschiedliche Datenquellen oder Datenbanktabellen ermöglichen, erreicht werden.

- Die System-Architektur sollte für iterative Entwicklung geeignet sein.
- Das System sollte erweiterbar sein.
- Das System sollte generische Entwurfsmuster nutzen.

4.9.5 Locations als besucht markieren

Bei der Bewertung einer Location durch den Nutzer ergibt sich die Problematik, dass sich der Zeitpunkt, zu dem sich der Nutzer tatsächlich in der bewerteten Location aufgehalten hat, nicht einfach nachvollziehen lässt. Das stellt jedoch eine wichtige Information dar, da nicht Locations, sondern zeitpunktabhängige Besuche in Locations bewertet werden sollen. Analog beziehen sich Empfehlungen nicht unabhängig vom Zeitpunkt auf eine Location, sondern auf Besuche in Locations zu einem bestimmten Wochentag. Aus dieser Problemstellung heraus hat sich die Projektgruppe nach Evaluierung der Mockups (siehe Kapitel 6.4.1) dazu entschieden, eine Funktionalität bereitzustellen, mit der der Nutzer Locations als besucht markieren kann. So soll nachvollziehbar sein, wann ein Nutzer eine Location besucht hat. Damit lässt sich die Bewertung des Nutzers einem Besuch zuordnen. Ein Nutzer kann demzufolge lediglich Besuche von den Locations bewerten, die er als besucht markiert hat. Der Nutzer soll auf der App direkt von der Liste von Empfehlungen aus Locations als besucht markieren können und die Liste dieser Locations in einer separaten Ansicht einsehen können. Aus dieser Liste soll zudem draus hervor gehen, welche der als besucht markierten Locations bereits bewertet wurden und welche nicht. Dies dient der Übersicht und als Anreiz, Besuche zu bewerten. Falls sich der Nutzer nachträglich gegen einen Besuch in der markierten Location entscheidet, soll die Location wieder aus der Liste löschen können.

- Die App sollte es dem Nutzer ermöglichen, empfohlene Locations als besucht zu markieren.
- Die App sollte es dem Nutzer ermöglichen, die als besucht markierten Locations in einer separaten Liste einzusehen.
- Die App sollte es dem Nutzer ermöglichen, über die Liste der markierten Locations die Bewertung für Besuche vorzunehmen.
- Die App sollte es dem Nutzer ermöglichen, über die Liste der markierten Locations einzusehen, welche der markierten Locations bereits bewertet wurden.
- Die App sollte es dem Nutzer ermöglichen, markierte Locations wieder aus der Liste zu entfernen.

4.9.6 Anlegen neuer Locations

Bei der prototypischen Entwicklung der App werden zunächst Locations beschränkt auf einen kleinen, regionalen Raum angelegt. Aus rechtlichen Gründen wird auf die Verwendung von Locationdaten externer Anbieter verzichtet. Aus Kapazitätsgründen kann daher für den Prototyp nur eine begrenzte Anzahl an Locations angelegt werden. Für die Erweiterbarkeit soll jedoch die Möglichkeit geboten werden, auch nach Projektabschluss neue Locations, unabhängig von der Lage, anzulegen. Ob dies von Nutzerseite oder Administratorseite aus ermöglicht werden soll, wird an dieser Stelle noch offengehalten. Gleiches gilt für die Frage, über welche Maske, z.B. App oder Dashboard, diese

Funktionalität bereit gestellt werden soll. Das Anlegen neuer Locations muss nach vorgegebenen Schema erfolgen und mit dem Schema der initial eingepflegten Locations konform sein.

- Das System sollte das Anlegen neuer Locations ermöglichen.
- Das Anlegen neuer Locations sollte nach vorgegebenen Schema erfolgen.

5 Spezifische Anforderungen

Nicht alle der potenziellen Anforderungen, die in der Anforderungsanalyse (siehe Kapitel 4) erfasst worden sind, können im Rahmen des Projekts umgesetzt werden. In diesem Kapitel werden die spezifischen Anforderungen, die die Projektgruppe umsetzen möchte, in Use Cases dargestellt (Kapitel 5.1) und im Anforderungskatalog in funktionale und nicht-funktionale Anforderungen untergliedert und aufgelistet (Kapitel 5.2 und 5.3). In einer gemeinsamen Sitzung im Anschluss an die Anforderungserhebung sind hierfür alle potenziellen Anforderungen von der Projektgruppe analysiert und priorisiert worden. Anforderungen, die direkt die Umsetzung von Recommender-Funktionalitäten in einem Anwendungsfall betreffen, sind z.B. höher priorisiert worden als individuelle Anforderungen, die sich ausschließlich auf Randfunktionalitäten der App beziehen. Einige Anforderungserhebungsmethoden, wie die Umfragen (Kapitel 4.4 und 4.5) oder das Brainstorming (Kapitel 4.3), haben zahlreiche individuelle Wünsche und Anregungen hervorgebracht. Die Häufigkeit der Nennung hat beim Priorisieren der potenziellen Anforderungen eine wichtige Rolle gespielt. Weitere Kriterien wie Umsetzbarkeit oder Sinnhaftigkeit haben außerdem Einfluss auf die Auswahl in die spezifischen Anforderungen gehabt. Einige der potenziellen Anforderungen sind aufgrund ihrer gleichen Bedeutung im Anforderungskatalog aggregiert aufgelistet.

5.1 Use Cases

In diesem Kapitel werden die Use Cases konkretisiert und mit den im vorherigen Kapitel erhobenen Anforderungen sowie die in Abschnitt 5.2 beschriebenen Anforderungssätze verbunden. Es werden zwischen Hauptfunktionen der App und des Backends unterschieden. Während der Akteur Nutzer die App verwendet, nutzt der Admin das Backend. Zusätzlich wird ein Use-Case-Diagramm für die Benutzerverwaltung der App modelliert. Die in diesem Abschnitt referenzierten Anforderungen sind in Kapitelabschnitt 5.2 aufgeführt.

5.1.1 App

Insgesamt werden zwölf Use Cases identifiziert. Diese sind in Abbildung 5.1 dargestellt. Die detaillierte Beschreibung der Use Cases erfolgt in den Tabellen 5.1 bis 5.10.

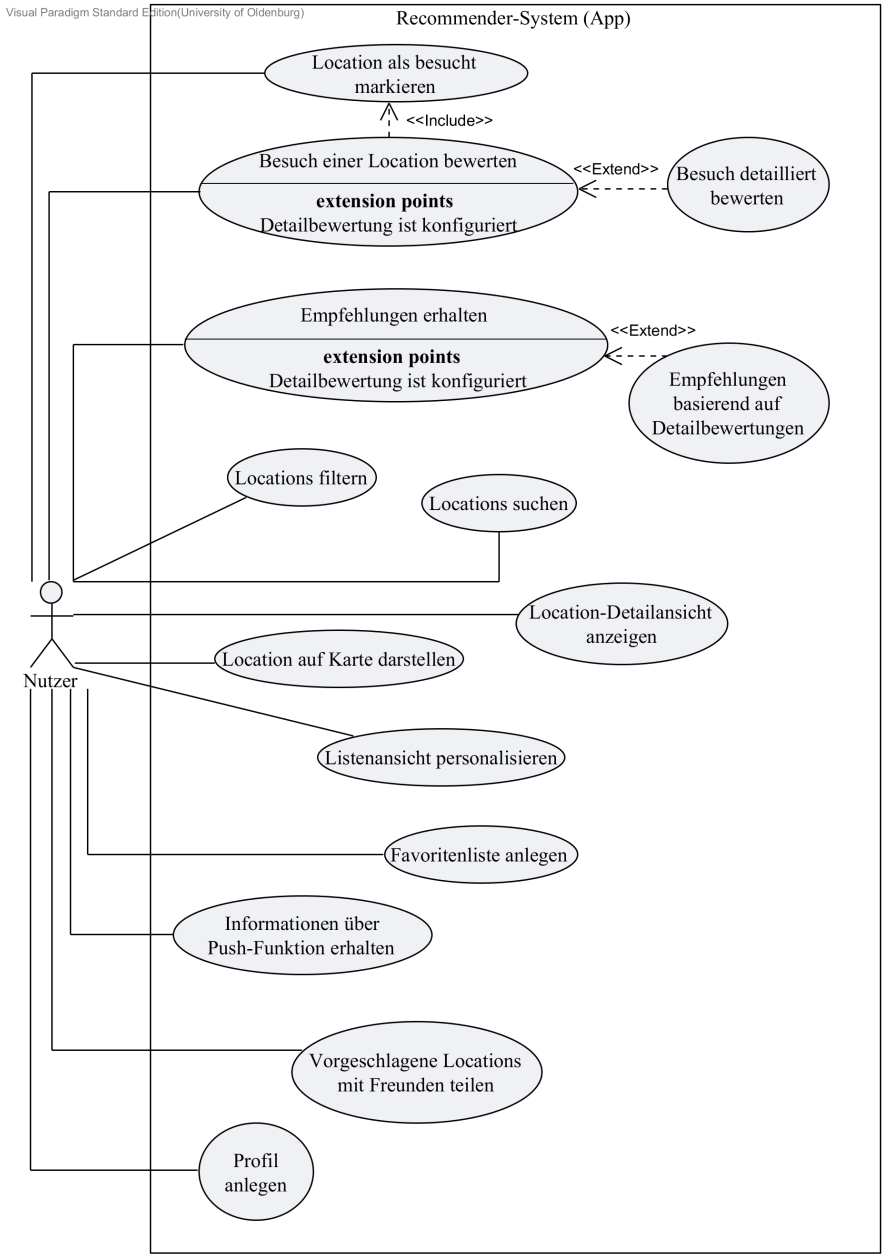


Abbildung 5.1: Use-Case-Diagramm für die App. Quelle: Eigene.

Der Use Case *Besuch einer Location bewerten* wird in dem Workshop zur Auswahl eines Anwendungsszenarios ermittelt (s. Abschnitt 4.2). Die Bewertung des Besuchs stellt eine Hauptfunktionalität für den Akteur Nutzer dar, welche sich als funktionale Anforderung in AN-F-01 wiederfindet. Der Nutzer soll eine empfohlene Location oder eine durch die Suche gefundene Location als besucht markieren können. Die Bewertung des Besuchs sollte sowohl allgemein als auch detailliert erfolgen können. Die Anforderungssätze werden in AN-F-01.02 und AN-F-01.03 formuliert.

Die Bewertungen sollten in Form einer Sterne-Skala erfolgen (s. AN-F-01.01). Dies folgt den Ergebnissen sowohl aus der Online-Umfrage als auch aus den problemzentrierten Interviews (vgl.

Abschnitte 4.4.2 und 4.5.1). Zudem sollte die Bewertung auch einen zusätzlichen Kommentar umfassen (s. AN-F-01.04). Diese Anforderung folgt aus der Online-Umfrage sowie aus dem Konkurrenzprodukt Yeti in Abschnitt 4.8.2.

In Tabelle 5.1 ist die Beschreibung des Use Cases zu finden.

Name	**Besuch einer Location bewerten**
Kurzbeschreibung	Die App sollte dem Nutzer ermöglichen, eine Bewertung des Besuchs abzugeben.
Akteure	Nutzer
Vorbedingungen	Nutzer muss angemeldet sein, Location muss eingetragen sein
Fachlicher Auslöser	Nutzer besucht eine Location (unabhängig, ob dieser diese selbst oder auf Empfehlung der App hin besucht) und möchte diese bewerten.
Hauptszenario/ -prozess	1. Empfohlene oder gesuchte Location als besucht markieren. 2. Als besucht markierte Locations anzeigen lassen. 3. Bewertungsansicht der zu bewertenden Location aus Liste aufrufen. 4. Gesamtbewertung in der Bewertungsansicht abgeben. 5. Bewertung mittels Middleware und DSMS verarbeiten.
Alternativszenario/ -prozess	1. Empfohlene oder gesuchte Location als besucht markieren. 2. Als besucht markierte Locations anzeigen lassen. 3. Bewertungsansicht der zu bewertenden Location aus Liste aufrufen. 4. Gesamtbewertung in der Bewertungsansicht abgeben. 5. Detailbewertung in der Bewertungsansicht abgeben. 6. Bewertung mittels Middleware und DSMS verarbeiten.
Alternativszenario/ -prozess	1. Empfohlene oder gesuchte Location als besucht markieren. 2. Als besucht markierte Locations anzeigen lassen. 3. Bewertungsansicht der zu bewertenden Location aus Liste aufrufen. 4. Gesamtbewertung in der Bewertungsansicht abgeben. 5. Detailbewertung in der Bewertungsansicht abgeben. 6. Kommentar in der Bewertungsansicht abgeben. 7. Bewertung mittels Middleware und DSMS verarbeiten.

Tabelle 5.1: Natürlichsprachliche Beschreibung des Use Cases.

Der Use Case *Empfehlungen erhalten* ist eine Kernanforderung an das Projekt. Der Use Case ergibt sich zum einem aus dieser Aufgabenstellung selbst, wird zum anderen zusätzlich im Workshop zur Auswahl eines Anwendungsszenarios 4.2 festgehalten. Der Auftraggeber unterstreicht die Priorität des

Use Cases, indem er im Projektleiterinterview (s. Abschnitt 4.6) den Wunsch äußert, dass der Nutzer Empfehlungen über eine Smartphone-App erhalten sollte. Die Empfehlungen sollten auf Grundlage von Gesamtbewertungen und Detailbewertungen generiert werden. Der Use Case ist in 5.2 beschrieben und findet sich als Anforderung in AN-F-02.

Name	**Empfehlungen erhalten**
Kurzbeschreibung	Der Nutzer erhält eine Empfehlung.
Akteure	Nutzer
Vorbedingungen	Der Nutzer ist angemeldet.
Fachlicher Auslöser	Der Nutzer möchte eine Empfehlung erhalten.
Hauptszenario/ -prozess	1. Nach Login Anfrage für Empfehlungsliste senden. 2. Empfehlungen werden durch die Middleware und das DSMS auf Grundlage von Gesamtbewertung generiert. 3. Liste mit Top-Empfehlungen wird von der App empfangen und angezeigt.
Alternativszenario/ -prozess	1. Nach Login Anfrage für Empfehlungsliste senden. 2. Empfehlungen werden durch die Middleware und das DSMS auf Grundlage von Detailbewertung generiert. 3. Liste mit Top-Empfehlungen wird von der App empfangen und angezeigt.

Tabelle 5.2: Natürlichsprachliche Beschreibung des Use Cases.

Der Use Case *Locations suchen* dient dem Nutzer dazu, gezielt nach Locations zu suchen. Der Nutzer soll über den Namen der Location, die Straße und die Postleitzahl suchen können. Diese Anforderung wird im Rahmen des Brainstormings (s. Abschnitt 4.3) erhoben. Die Suchfunktion wird beispielsweise auch von den Konkurrenzprodukten Yelp und TripAdvisor implementiert (s. Abschnitte 4.8.1 und 4.8.3). Die Beschreibung des Use Cases ist in Tabelle 5.3 zu finden. Die Anforderung wird in AN-F-03 formuliert.

Der Use Case *Location-Detailansicht anzeigen* beschreibt die detaillierte Ansicht, die dem Nutzer auf seinen Wunsch hin für eine bestimmte Location separat angezeigt werden kann. Der Use Case basiert auf den Ergebnissen des Brainstormings in der Gruppe und der Sichtung von Konkurrenzprodukten (s. Abschnitt 4.3 und 4.8.2). Er ist in AN-F-04 festgehalten. Der Nutzer erhält über die normale Liste eine Auflistung an Empfehlungen über Locations oder nach eigenständiger Suche bestimmte Suchergebnisse. In beiden Fällen besteht für den Nutzer die Möglichkeit, detaillierte Informationen über für ihn potenziell interessante Locations angezeigt zu bekommen. Diese Detailansicht enthält weitere Informationen über die Location. Zudem soll es dem Nutzer in dieser Ansicht möglich sein, Kommentare anderer Nutzer über diese Location einzusehen.

Der Use Case *Location auf Karte darstellen* wird in dem Brainstorming zur Anforderungsanalyse des Workshops ermittelt. (s. Abschnitt 4.2) Die zusätzliche Karten-Darstellung der Empfehlungen

Name	**Locations suchen**
Kurzbeschreibung	Der Nutzer sucht eigenständig nach Locations.
Akteure	Nutzer
Vorbedingungen	Nutzer ist angemeldet.
Fachlicher Auslöser	Nutzer möchte eine Location besuchen oder bewerten.
Hauptszenario/ -prozess	1. Menü aufklappen. 2. Funktion *Suche* über Menü aufrufen. 3. Suchfilter auswählen. 4. Im Suchfeld das ausgewählte Suchattribut eingeben. 5. Suchformular abschicken. 6. Über Middleware nach Locations suchen. 7. Ergebnisliste anzeigen.

Tabelle 5.3: Natürlichsprachliche Beschreibung des Use Cases.

Name	**Location-Detailansicht anzeigen**
Kurzbeschreibung	Dem Nutzer wird auf Wunsch eine Detailansicht der Location angezeigt.
Akteure	Nutzer
Vorbedingungen	Nutzer ist angemeldet, Detail-Informationen über Location liegen vor.
Fachlicher Auslöser	Der Nutzer möchte detailliertere Informationen über eine Location, für die er sich interessiert, erhalten.
Hauptszenario/ -prozess	1. Eine Listenansicht, die empfohlene Locations zeigt, öffnen. 2. Nutzer wählt Location, die als Detailansicht angezeigt werden soll, aus. 3. Neue Ansicht mit Detail-Informationen zu dieser Location wird angezeigt. 4. Der Nutzer kann bei Bedarf Kommentare zur angezeigten Location abgeben.

Tabelle 5.4: Natürlichsprachliche Beschreibung des Use Cases.

soll dem Nutzer vor allem der Übersichtlichkeit dienen. Durch die Darstellung auf einer Karte wird es dem Nutzer erleichtert, eine für ihn passende Empfehlung auszuwählen. Durch die durchgeführte Online-Umfrage (s. Abschnitt 4.4.2) wird die Wichtigkeit dieser Darstellungsform bestätigt und deswegen als Anforderung AN-F-05 festgehalten. Weitere Funktionalitäten, die mit der Karten- bzw. Stadtplan-Darstellung zusammenhängen (z.B. Routenplanung), haben zunächst nicht die höchste Priorität, können jedoch zu einem späteren Zeitpunkt Beachtung erhalten.

In Tabelle 5.5 ist die Beschreibung des Use Cases zu finden.

Der Use Case *Listenansicht personalisieren* wird als eine der Funktionen aus dem Vergleich mit Konkurrenzprodukten aufgenommen. Konkret handelt es sich um die Umsetzung der Anforderung,

Name	**Location auf Karte darstellen**
Kurzbeschreibung	Die App sollte dem Nutzer ermöglichen, empfohlene Locations in einer Kartenansicht ablesen zu können.
Akteure	Nutzer
Vorbedingungen	Nutzer ist angemeldet. Es wurden Empfehlungen für den Nutzer generiert.
Fachlicher Auslöser	Der Nutzer erhält eine Liste von Empfehlungen und möchte den Standort der Empfehlungen auf einer Karte (z. B.Stadtplan) dargestellt haben.
Hauptszenario/ -prozess	1. Karten-Icon auf Startseite anklicken 2. Standortkoordinaten der Locations aus Datenbank lesen. 3. Top-Empfehlungen in Kartenansicht anzeigen. 4. Weitere Informationen per Klick auf Infokasten der Location anzeigen lassen.

Tabelle 5.5: Natürlichsprachliche Beschreibung des Use Cases.

dass der Nutzer die Empfehlungen nach verschiedenen Kriterien auch Offline filtern können sollte (s. Abschnitt 4.8.4). Eine mögliche Umsetzung soll dem Nutzer das Filtern nach Locationart ermöglichen. Der Use Case wird in Tabelle 5.6 beschrieben und als AN-F-06 formuliert.

Name	**Listenansicht personalisieren**
Kurzbeschreibung	Veränderung der Listensortierung und der in der Liste dargestellten Kriterien.
Akteure	Nutzer
Vorbedingungen	Nutzer muss angemeldet sein, es muss mindestens ein Element in der Liste enthalten sein.
Fachlicher Auslöser	Nutzer wählt ein Kriterium aus, das er in der Listenansicht sehen will oder nach welchem er die Liste sortieren will.
Hauptszenario/ -prozess	1. Es wird eine Listenansicht angezeigt. 2. Nutzer wählt das Kriterium, das in der Liste angezeigt oder für die Sortierung genutzt werden soll, aus. 3. Liste wird aktualisiert.

Tabelle 5.6: Natürlichsprachliche Beschreibung des Use Cases.

Der Use Case *Favoritenliste anlegen* wird als eine der vorgeschlagenen Zusatzfunktionen der App in der Online-Umfrage (s. Abschnitt 4.4.2) aufgenommen. Der Nutzer soll in der Lage sein, von ihm ausgewählte Locations zu favorisieren, damit er diese gespeicherten Favoriten zu einem späteren Zeitpunkt komfortabel wieder abrufen kann. Dies stellt eine Zusatzfunktion der App dar, da sie nicht direkt mit der Hauptfunktion, der Generierung von Empfehlungen, in Verbindung steht. Trotzdem wird sie als Anforderung AN-F-07 aufgenommen, da sie häufig als gewünschte Funktionalität in der Online-Umfrage (s. Abschnitt 4.4.2) aufgeführt wird und zusätzlich dazu in Konkurrenzprodukten

wie Foursquare (4.8.4) zu finden ist. Der Use Case wird in Tabelle 5.7 beschrieben und als AN-F-07 formuliert.

In Tabelle 5.7 ist die Beschreibung des Use Cases zu finden.

Name	**Favoritenliste anlegen**
Kurzbeschreibung	Die App sollte es dem Nutzer ermöglichen, eine Favoritenliste anzulegen.
Akteure	Nutzer
Vorbedingungen	Nutzer ist angemeldet.
Fachlicher Auslöser	Nutzer wählt eine Location aus, die er favorisieren möchte.
Hauptszenario/ -prozess	1. Nutzer gelangt über eine Listenansicht auf die Detailseite einer Location. 2. Nutzer wählt eine Location aus, die er favorisieren möchte. 3. DBMS verarbeitet und speichert Favorisierung. 4. Im Offline Modus wird die Favorisierung zunächst lokal gespeichert. 5. Sobald Verbindung zum DBMS besteht, wird die Eingabe synchronisiert und die Location im DBMS hinterlegt. 6. Nutzer wird bestätigt, dass die Location favorisiert wird. 7. Bei Bedarf wird Favorit wieder aus Liste gelöscht.

Tabelle 5.7: Natürlichsprachliche Beschreibung des Use Cases.

Der Use Case *Informationen über Push-Funktion erhalten* wird trotz der ermittelten Kritik an Push-Nachrichten im Allgemeinen (s. Abschnitt 4.4.2) in Anforderung AN-F-08 festgehalten. Der potenzielle Nutzen wird im problemzentrierten Interview (s. Abschnitt 4.5.1) festgestellt und gleichzeitig von der Projektgruppe als ansprechendes Feature angesehen. Über die Push-Nachrichten wird der Nutzer über weitere Informationen (z.B. Specials oder Angebote) zu favorisierten Locations in Kenntnis gesetzt. (s. Abschnitt 4.5.1) In Tabelle 5.8 ist die Beschreibung des Use Cases zu finden.

Name	**Informationen über Push-Funktion erhalten**
Kurzbeschreibung	Der Nutzer soll über Push-Nachrichten Informationen bereitgestellt bekommen.
Akteure	Nutzer
Vorbedingungen	Nutzer ist angemeldet.
Fachlicher Auslöser	Zusatzinformationen für den Nutzer werden ermittelt.
Hauptszenario/ -prozess	1. Relevante Informationen werden ermittelt. 2. Nutzer erhält Informationen über eine Push-Nachricht.

Tabelle 5.8: Natürlichsprachliche Beschreibung des Use Cases.

Der Use Case *Vorgeschlagene Locations mit Freunden teilen* wird aus den Ergebnissen der Umfragen hergeleitet. Insbesondere in der Online-Umfrage (s. Abschnitt 4.4.2) haben die Teilnehmer häufig angegeben, dass es ihnen wichtig ist, etwas mit Freunden zu unternehmen. Genauer wird mit diesem

Use Case die Anforderung berücksichtigt, dass die App soziale Aspekte einbeziehen soll (s. Abschnitt 4.4.2). Daneben haben Teilnehmer auch geäußert, dass sie ihre persönlichen Empfehlungen mit Freunden teilen möchten. Durch das Teilen von Locations mit Freunden wird berücksichtigt, dass Locations in der Regel nicht von einer einzelnen Person besucht werden, sondern in der Regel von (Klein-) Gruppen. Durch das Teilen der Locations ist es den Nutzern somit ermöglicht, ihren Freunden mitzuteilen, welche Locations sie besuchen möchten. Der Use Case wird in Anforderung AN-F-09 formuliert.

Name	**Vorgeschlagene Locations mit Freunden teilen**
Kurzbeschreibung	Die App sollte es dem Nutzer ermöglichen, seine Vorschläge für Locations mit Freunden zu teilen.
Akteure	Nutzer
Vorbedingungen	Nutzer kennt Nutzer, welchen er Empfehlung schicken will.
Fachlicher Auslöser	Nutzer wählt eine Empfehlung aus, die er teilen möchte.
Hauptszenario/ -prozess	1. Nutzer wählt eine Empfehlung aus, die er teilen möchte. 2. Nutzer betätigt in Detailansicht oder Bewertungsansicht den Teilen-Button. 3. Relevante Daten über die Location werden an andere Applikationen weitergegeben.

Tabelle 5.9: Natürlichsprachliche Beschreibung des Use Cases.

Ein Profil dient dazu, persönliche Informationen und Angaben des Nutzers zu erfassen. Der Use Case *Profil anlegen* (Beschreibung in Tabelle 5.10) entstammt dem Workshop zur Auswahl eines Anwendungsszenarios in Abschnitt 4.2 und wird als Anforderung AN-F-10 formuliert. Der Nutzer soll ein Profilbild einstellen und seine Login-Angaben auf der Profilseite verwalten können.

Name	**Profil anlegen**
Kurzbeschreibung	Nutzer legt ein Profil mit seinen persönlichen Daten an.
Akteure	Nutzer
Vorbedingungen	Nutzer muss angemeldet sein.
Fachlicher Auslöser	Der Nutzer möchte ein Profilbild hochladen oder seine Login-Daten verändern.
Hauptszenario/ -prozess	1. Eigene Profilseite öffnen. 2. Auswahl der persönlichen Informationen, die eingestellt oder verändert werden sollen. 3. Speichern der persönlichen Daten.

Tabelle 5.10: Natürlichsprachliche Beschreibung des Use Cases.

5.1.2 Benutzerverwaltung

Die Benutzerverwaltung konzentriert sich primär auf die Nutzer der App. Als Vorbedingung der Use Cases in Abschnitt 5.1.1 wird oft genannt, dass der Nutzer angemeldet sein muss. Diese Vorbedingung wird durch die Benutzerverwaltung gelöst. Die Use Cases umfassen die Registrierung des Nutzers und die Anmeldemöglichkeit. Zudem sollte die Möglichkeit bestehen, dass der Nutzer sein Passwort ändern kann (s. Tabellen 5.11-5.13). Dies wurde durch das Backlog Grooming als potenzielle Anforderung identifiziert (Kapitel 7.8).

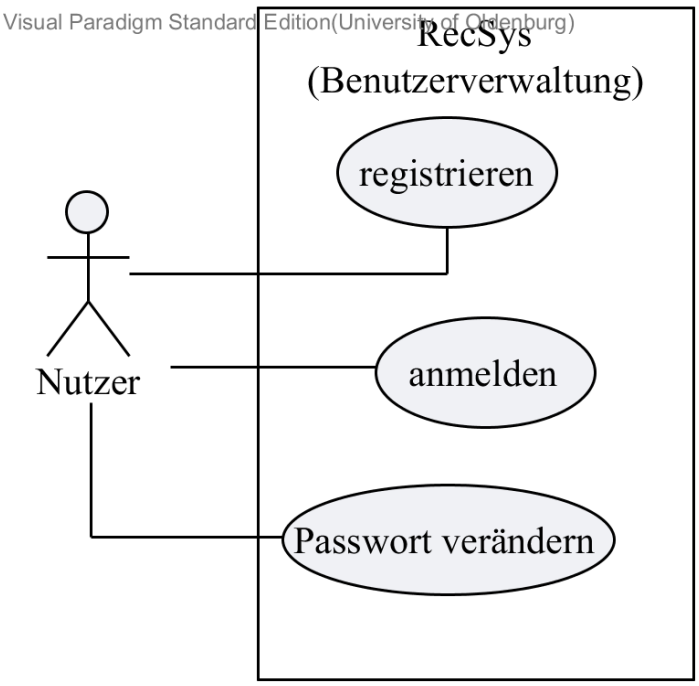


Abbildung 5.2: Use-Case-Diagramm für die Benutzerverwaltung. Quelle: Eigene.

Der Use Case *Registrieren* ist erforderlich, bevor der Nutzer die App nutzen und sich anmelden kann. Die Anforderung wird in AN-F-11.01 formuliert und die Beschreibung ist in Tabelle 5.11 zu finden. Dieses wurde durch das Backlog Grooming als potenzielle Anforderung identifiziert (Kapitel 7.8).

Der Use Case *Anmelden* wird benötigt, um die App nutzen zu können. In den Abschnitten 4.8.3 (TripAdvisor) und 4.8.4 (FourSquare) wird eine Anmeldemöglichkeit über verschiedene soziale Netzwerke beschrieben. Eine eigene Implementierung für das Anmelden ist auch denkbar. Die Anforderung ist in AN-F-11.02 formuliert. Die Kurzbeschreibung findet sich in Tabelle 5.12.

Der Use Case *Passwort ändern* ist im Zusammenhang mit der geforderten Benutzerverwaltung notwendig. Das Ändern des Passworts ist einerseits notwendig, um eine gewisse Sicherheit für den Nutzer und sein Profil zu gewährleisten. Andererseits muss im Falle eines vergessenen Passworts das Passwort zurückgesetzt und geändert werden können. Daher gibt es zwei Möglichkeiten das Passwort zu ändern. Das Passwort kann über die Einstellungen durch Bestätigung mit dem alten Passwort geändert werden. Durch die Bestätigung mit dem alten Passwort wird sichergestellt, dass es

Name	**Registrieren**
Kurzbeschreibung	Bevor der Nutzer sich anmelden kann, muss dieser im System registriert sein.
Akteure	Nutzer
Vorbedingungen	-
Fachlicher Auslöser	Nutzer möchte die App nutzen.
Hauptszenario/ -prozess	1. App aufrufen. 2. Registrierungsseite aufrufen. 3. Nutzerdaten eingeben. 4. Zugangsdaten eingeben. 5. Bestätigen. 6. Bestätigungs-E-Mail bestätigen.

Tabelle 5.11: Natürlichsprachliche Beschreibung des Use Cases.

Name	**Anmelden**
Kurzbeschreibung	Nutzer meldet sich mit seinen Zugangsdaten an.
Akteure	Nutzer
Vorbedingungen	Nutzer ist registriert.
Fachlicher Auslöser	Der Nutzer möchte die App nutzen.
Hauptszenario/ -prozess	1. App aufrufen. 2. Zugangsdaten eingeben. 3. Bestätigen. 4. App-Startseite öffnet sich.

Tabelle 5.12: Natürlichsprachliche Beschreibung des Use Cases.

sich tatsächlich um den Besitzer dieses Profils handelt. Falls das Passwort vergessen wird, kann im Login-Bildschirm eine E-Mail angefordert werden, um das Passwort zurückzusetzen. Hier wird die Identität des Benutzer durch die E-Mail bestätigt. Hier wird vorausgesetzt, dass einzig der jeweilige Nutzer selbst Zugriff auf seinen Posteingang hat.

Dieser Use Case ist in der Anforderung AN-F-11.02 zu finden. In Tabelle 5.13 ist die Beschreibung des Use Cases zu finden.

Name	**Passwort verändern**
Kurzbeschreibung	Das System sollte dem Nutzer ermöglichen, sein Passwort zu verändern.
Akteure	Nutzer
Vorbedingungen	Nutzer muss angemeldet sein und für das Alternativszenario muss der Nutzer abgemeldet sein.
Fachlicher Auslöser	Der Nutzer möchte sein Passwort ändern oder der Nutzer hat sein Passwort vergessen.
Hauptszenario/ -prozess	1. Einstellungen aufrufen. 2. *Passwort ändern* wählen. 3. Altes Passwort eingeben. 4. Neues Passwort eingeben. 5. Neues Passwort nochmal eingeben. 6. Vorgang bestätigen.
Alternativszenario/ -prozess	1. Login-Bildschirm aufrufen. 2. *Passwort vergessen?* wählen. 3. E-Mail eingeben. 4. Vorgang bestätigen. 5. Auf Empfang der E-Mail warten. 6. Link aus der E-Mail aufrufen. 7. Neues Passwort eingeben. 8. Neues Passwort nochmal eingeben. 9. Vorgang bestätigen.

Tabelle 5.13: Natürlichsprachliche Beschreibung des Use Cases.

5.1.3 Backend

Das Backend umfasst Funktionalitäten, die dem Nutzer der App nicht zugänglich gemacht werden, da sie primär aus Sicht des Admins erforderlich sind. In Abbildung 5.3 ist das zugehörige Use-Case-Diagramm dargestellt und die Tabellen 5.14 und 5.15 beschreiben die Use Cases genauer.

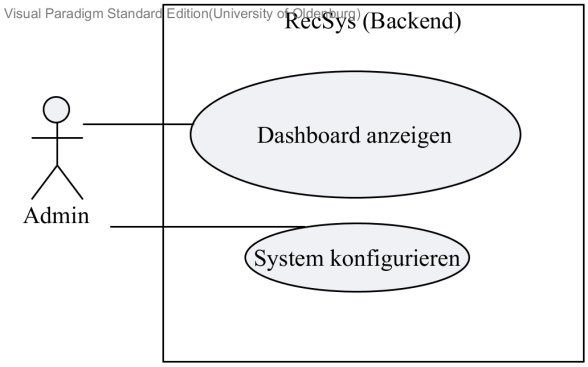


Abbildung 5.3: Use-Case-Diagramm für das Backend. Quelle: Eigene.

Der Use Case *Dashboard anzeigen* wird in dem Interview mit der Projektleitung ermittelt (s. Abschnitt 4.6). Das Dashboard stellt die Hauptfunktionalität für den Akteur Admin dar, welche sich als funktionale Anforderung in AN-F-12 wiederfindet. Das Dashboard soll hauptsächlich zum Darstellen verschiedener Qualitätsmetriken dienen, wie es in AN-F-12.01 festgehalten ist. Die Qualitätsmetriken werden in der Analyse von wissenschaftlichen Quellen (s. Abschnitt 4.7) erhoben.

Neben den Qualitätsmetriken sollen auch relevante Informationen über das System und die Empfehlungen anzeigt werden. Zu den relevanten Informationen gehören der Status der Serviceprovider, ob Empfehlungen angenommen oder abgelehnt worden sind, alle eintreffenden Bewertungen sowie der durchschnittliche Zeitraum zwischen Empfehlungsanfrage und Empfehlungserstellung. Diese Informationen können auch den wissenschaftlichen Quellen entnommen werden. Die entsprechenden Anforderungen sind AN-F-12.02 und AN-F-12.03.

Außerdem soll das Dashboard einen Vergleich verschiedener Algorithmen ermöglichen. Hierfür sollen die Qualitätsmetriken und relevanten Informationen von Algorithmen direkt gegenüber gestellt werden. Diese Anforderung stammt sowohl aus dem Interview mit der Projektleitung als auch aus den wissenschaftlichen Quellen und findet sich in der Anforderung AN-F-12.04 wieder.

Zudem soll ein Export der Qualitätsmetriken und relevanten Informationen möglich sein. Diese Anforderung stammt ebenfalls aus den wissenschaftlichen Quellen und ist in der Anforderung AN-F-12.05 festgehalten. In Tabelle 5.14 ist die Beschreibung des Use Cases zu finden.

Der Use Case *System konfigurieren* wird für den Akteur Admin modelliert. Er beschreibt, wie der Admin das RecSys konfigurieren kann. Die in Abschnitt 4.6 beschriebene Idee impliziert, dass der Admin über das Dashboard (s. Tabelle 5.14) verschiedene Lernalgorithmen für die Empfehlungsberechnung einstellen und diese selbst konfigurieren kann. Dadurch sollte dem Admin auch die Möglichkeit gegeben werden, Lernalgorithmen zu vergleichen. Diese Konfigurationen sollten als Einstellungsprofile gespeichert werden können, um das optimale Profil finden und verwenden zu können (s. Abschnitt 4.7). Die Konfiguration sollte über das Dashboard oder über eine DSL vorgenommen werden können.

Name	**Dashboard anzeigen**
Kurzbeschreibung	Das System sollte dem Admin ein Dashboard anzeigen.
Akteure	Admin
Vorbedingungen	Admin muss angemeldet sein.
Fachlicher Auslöser	Admin möchte den Status des RecSys abrufen.
Hauptszenario/ -prozess	1. Dashboard aufrufen. 2. Ggf. muss der Admin sich einloggen.

Tabelle 5.14: Natürlichsprachliche Beschreibung des Use Cases.

Der Use Case ist in Tabelle 5.15 beschrieben und findet sich als Anforderung in AN-F-13 wieder.

Name	**System konfigurieren**
Kurzbeschreibung	Konfigurieren des RecSys über das Dashboard
Akteur	Admin
Vorbedingungen	Admin muss angemeldet sein.
Fachlicher Auslöser	Finden und Einstellen der optimalen Konfiguration, Vergleich von Konfigurationen (Einstellungsprofilen)
Hauptszenario/ -prozess	1. Dashboard aufrufen. 2. Konfigurationsseite aufrufen. 3. Bestehendes Einstellungsprofil aufrufen oder neues erstellen (über einen Namen identifizierbar). 4. Einen neuen Lernalgorithmus erstellen, einen bestehenden Lernalgorithmus konfigurieren oder einen bestehenden Lernalgorithmus kopieren und diesen konfigurieren. 5. Lernalgorithmus speichern. 6. Einstellungsprofil speichern. 7. Aktivieren eines Einstellungsprofils.

Tabelle 5.15: Natürlichsprachliche Beschreibung des Use Cases.

5.2 Funktionale Anforderungen

AN-F-01 Die App sollte es dem Nutzer ermöglichen, eine Bewertung des Besuches abzugeben.

- **AN-F-01.01** Die Bewertung sollte in einer Sterne-Skala erfolgen.
- **AN-F-01.02** Die Bewertung sollte als allgemeine Bewertung des Besuches abgegeben werden.
- **AN-F-01.03** Die Bewertung sollte als Detailbewertung für verschiedene Kriterien abgegeben werden.
- **AN-F-01.04** Die App sollte dem Nutzer ermöglichen, bei der Bewertung eines Besuches einen zusätzlichen Kommentar zu verfassen.
- **AN-F-01.05** Die App sollte es dem Nutzer ermöglichen, Locations als besucht zu markieren.

AN-F-02 Die App sollte es dem Nutzer ermöglichen, personalisierte Empfehlungen zu erhalten.

- **AN-F-02.01** Die Empfehlungen sollten Kontext-Informationen miteinbeziehen.
- **AN-F-02.02** Die App sollte es dem Nutzer ermöglichen, über einen zusätzlichen Filter Kontext-Informationen in die Empfehlung einfließen zu lassen.
- **AN-F-02.03** Die Empfehlungen sollten in einer geordneten Liste angezeigt werden.
- **AN-F-02.04** Der Nutzer sollte die Möglichkeit haben, über eine Detailansicht weitere Informationen über die Empfehlungen zu erhalten.

AN-F-03 Die App sollte es dem Nutzer ermöglichen, Locations zu suchen.

AN-F-04 Die App sollte es dem Nutzer ermöglichen, eine Detailansicht für Locations anzeigen zu lassen.

- **AN-F-04.01** Die App sollte es dem Nutzer ermöglichen, Kommentare in der Detailansicht zu verfassen.
- **AN-F-04.02** Die Detailansicht sollte um weitere statische Informationen erweitert werden.
- **AN-F-04.03** Die App sollte dem Nutzer Bewertungen einer Location in einer separaten Liste anzeigen.

AN-F-05 Die App sollte es dem Nutzer ermöglichen, die Locations in einer Kartenansicht anzeigen zu lassen.

AN-F-06 Die Listenansicht sollte durch personalisierte Informationen individualisiert werden.

AN-F-07 Die App sollte es dem Nutzer ermöglichen, eine Favoritenliste anzulegen.

AN-F-08 Die App sollte mithilfe einer Push-Funktion dem Nutzer weitere Informationen bereitstellen.

AN-F-09 Die App sollte es dem Nutzer ermöglichen, vorgeschlagene Locations mit Freunden zu teilen.

AN-F-10 Die App sollte es dem Nutzer ermöglichen, ein Profil anzulegen.

AN-F-11 Das System sollte dem Nutzer eine Benutzerverwaltung bereitstellen.

- **AN-F-11.01** Das System sollte es dem Nutzer ermöglichen, sich zu registrieren.

AN-F-11.02 Das System sollte es dem Nutzer ermöglichen, sein Passwort zu verändern.

AN-F-11.03 Das System sollte es dem Nutzer ermöglichen, sich anzumelden.

AN-F-11.04 Die App sollte es dem Nutzer ermöglichen, sein Passwort zurückzusetzen.

AN-F-11.05 Die App sollte es dem Nutzer ermöglichen, seine E-Mail-Adresse zu ändern.

AN-F-11.06 Die App sollte es dem Nutzer ermöglichen, sich abzumelden.

AN-F-12 Das System sollte dem Admin ein Dashboard anzeigen.

AN-F-12.01 Das Dashboard sollte dem Admin Qualitätsmetriken darstellen.

AN-F-12.02 Das Dashboard sollte dem Admin eine Übersicht über relevante Informationen über die Empfehlungen bieten.

AN-F-12.03 Das Dashboard sollte dem Admin eine Übersicht über relevante Informationen über das System bieten.

AN-F-12.04 Das Dashboard sollte es dem Admin ermöglichen, Algorithmen zu vergleichen.

AN-F-12.05 Das Dashboard sollte es dem Admin ermöglichen, Informationen zu exportieren.

AN-F-12.06 Das Dashboard sollte es dem Admin ermöglichen, neue Locations anzulegen.

AN-F-12.07 Das Dashboard sollte es dem Admin ermöglichen, die Middleware zu initialisieren.

AN-F-13 Das System sollte dem Admin Konfigurationsmöglichkeiten bieten.

5.3 Nicht-funktionale Anforderungen

AN-NF-01 Die App sollte schnelle Ergebnisse liefern.

AN-NF-02 Die App sollte die Berechtigungen transparent gestalten.

AN-NF-03 Das System sollte wartungsfreundlich aufgebaut sein.

AN-NF-04 Das System sollte generisch entwickelt werden.

AN-NF-05 Die App sollte bedienungsfreundlich sein.

6 Systemkonzept

Das Systemkonzept gibt einen umfassenden Überblick über sämtliche Komponenten und deren Zusammenhänge im System sowie über die grundlegende Systemarchitektur. Die hier beschriebenen Komponenten werden zur Nachvollziehbarkeit zu den gestellten Anforderungen in Kapitel 5 in Verbindung gesetzt.

6.1 Domänenmmodell

Das Domänenmodell gibt einen Überblick über die Entitäten und ihre Attribute im System sowie über ihre Kardinalitäten und Beziehungen zueinander. Es bildet die fachlichen Daten im System ab.

6.1.1 Zentrale Entitäten

Abbildung 6.1 zeigt die Entitäten `User`, `Check-In`, `Recommendation` und `Location` sowie die Entitäten `Comment` und `BusinessHour`, die in diesem Abschnitt detailliert erläutert werden.

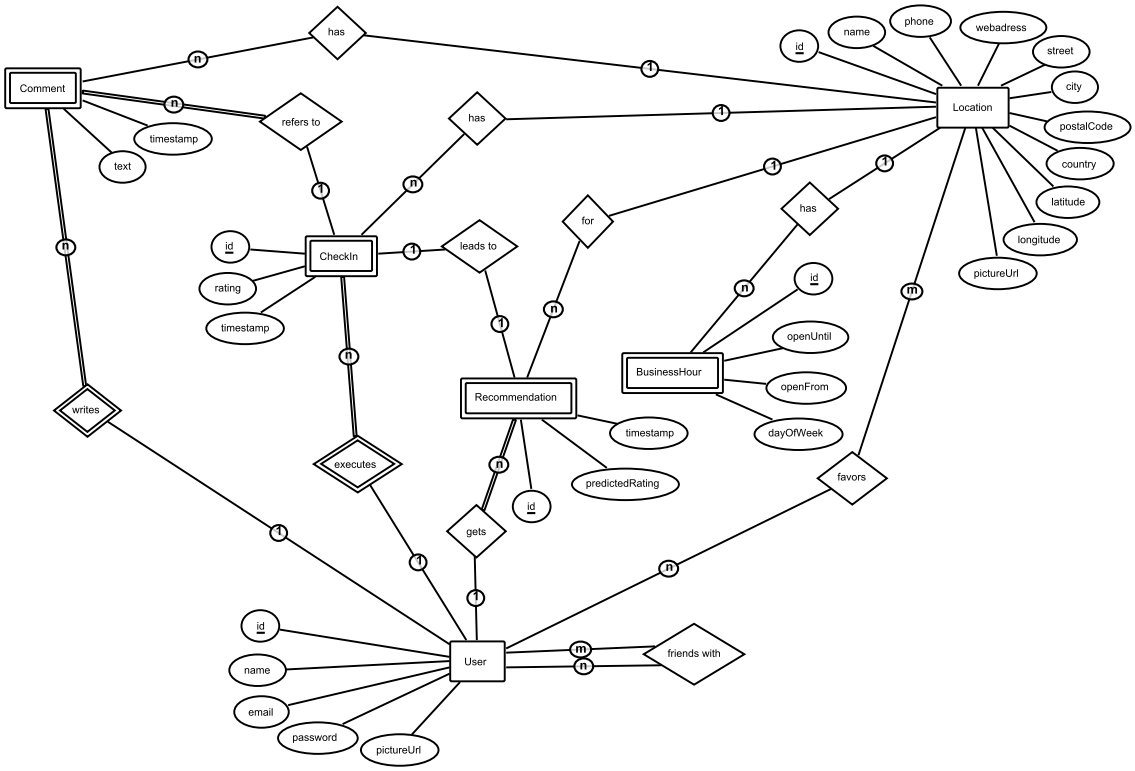


Abbildung 6.1: EER-Domänenmodell der zentralen Entitäten (ohne Filter). Quelle: Eigene.

Die zentralen Entitäten sind `User` und `Location`. User enthalten ihre eigene ID, den Namen, die E-Mail-Adresse, das Passwort (als MD5-Hash) und die URL ihres Profilbildes.

Die m:n-Beziehung vom `User` auf sich selbst besteht aufgrund der Annahme, dass Nutzer miteinander befreundet sein können. Hierüber können Beziehungen zwischen den Nutzern abgebildet werden.

Die zweite zentrale Entität sind die Locations. Die Relation Locations verfügt über die Attribute *Id, Name, Phone, Street, Postcode, City, Country, Longitude, Latitude* und die *Webadress.*.

Die Entität `BusinessHour` verfügt über die Attribute *Id*, *openUntil* und *openFrom* (geben an, von wann bis wann die Location geöffnet ist) und *dayOfWeek* (gibt den Wochentag an, an dem die Öffnungszeiten gelten). Es besteht eine 1:n-Beziehung zu der `Location`, da eine Location mehrere Öffnungszeiten haben kann.

Über die m:n- Beziehung zwischen User und Location wird die Favoriten-Funktionalität abgebildet. Dies bedeutet, dass Nutzer Locations favorisieren können.

Die Entität `Recommendation` wurde angelegt, um an die Nutzer gegebene Empfehlungen zu speichern. Diese Empfehlungen verfügen über eine Id, eine vorhergesagte Bewertung und einen Zeitstempel. Es existieren 1:n-Beziehungen zu `Location` und `User`, so dass jede Empfehlung für einen bestimmten User eine bestimmte Location enthält. Außerdem existiert eine 1:n-Beziehung zu `CheckIn`. In diesen Fall steht *n* für 0 bis 1. Über diese Verbindung sollte der Recall der Empfehlungen gemessen werden, um die Qualität der Empfehlungen beurteilen zu können. Dies wurde nicht umgesetzt, Recommendations liegen zur Laufzeit vor, werden jedoch nicht persistiert.

Zu einem `CheckIn` gehören die Attribute *Id*, die abgegebene Bewertung (*Rating*) und der Zeitstempel (*timestamp*) des CheckIns. Die Bewertung kann im Nachhinein abgegeben werden. Der Schlüssel des `CheckIn` setzt sich aus der Id des Users und einer Id für den CheckIn des jeweiligen Nutzers zusammen (i.d.R. fortlaufend für jeden Nutzer). Der `CheckIn` verfügt analog zur `Recommendation` über Beziehungen zum `User` und der `Location`. Außerdem existiert eine 1:n-Beziehung zur Entität `Comment`, wobei n für 0 bis 1 steht, sodass jeder Kommentar mit einem CheckIn verbunden sein muss, aber nicht jeder CheckIn über einen Kommentar verfügen muss.

Zusammen mit der Bewertung kann ein Kommentar angelegt werden. Die Entität `Comment` verfügt analog zu den Entitäten `Recommendation` und `CheckIn` über Beziehungen zu den Entitäten `Location` und `User`. Attribute von Kommentaren sind ein Zeitstempel (*timestamp*) sowie der Text (*text*) des Kommentars.

6.1.2 Filtermodell

Die Filter sollen generisch sein, so dass es jederzeit möglich ist, weitere Filter anzulegen und bestehende Filter zu bearbeiten oder zu löschen. Wichtig ist dabei, dass für diese Veränderungen weder am Quellcode der App noch an dem der Middleware strukturelle Änderungen notwendig werden (siehe AN-F-02.02). Zudem ist dies Teil der Umsetzung der Anforderung AN-NF-03 auf Seite 125 in Abschnitt AN-NF-03. Das Filtermodell bezeichnet dabei die Klassen beziehungsweise Relationen, die für die Umsetzung der Filter genutzt werden.

Die Filter werden über fünf weitere Entitäten abgebildet (s. Abbildung 6.2). `FilterAttribute` stellen dabei den Oberbegriff der Filter da. Ein `FilterAttribute` ist zum Beispiel der Filter Rauchfrei. `FilterAttribute` verfügt über eine *Id* und einen *Type*. Dieser gibt an, um was für eine Art Filter es sich hat. Beispielsweise enthält er den Text 'boolean'. Dies dient der Festlegung, wie der

Filter innerhalb der App angezeigt werden soll (beispielsweise als Schalter) und wie er ausgewertet werden soll.

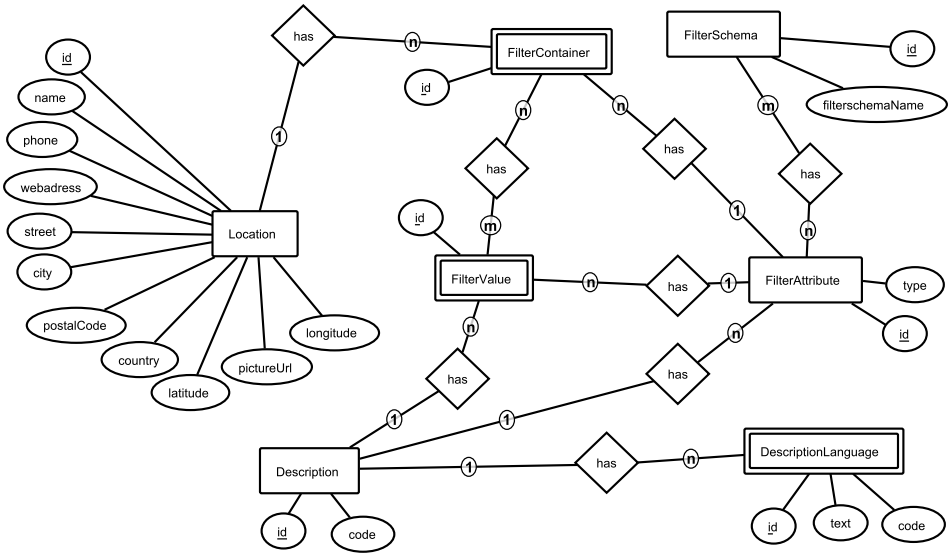


Abbildung 6.2: EER-Domänenmodell der für die Filter notwendigen Entitäten inkl. Location. Quelle: Eigene

`FilterValue` bezeichnet die Werte, die die Filter annehmen können. Diese Objekte verfügen über eine n:1-Beziehung zu einem `FilterAttribute`. Sie verfügen lediglich über eine Id. Ein `FilterAttribute` des Typs `Boolean` hätte zum Beispiel zwei `FilterValues`, eins für den Wert wahr und eins für den Wert falsch. Einem `FilterAttribute` des Typs `List` können beliebig viele `FilterValue`-Objekte zugeordnet werden, wobei jedes einen Listenpunkt abbilden würde.

Um eine spätere Umsetzung einer mehrsprachigen App zu ermöglichen, werden mit `Description` und `DescriptionLanguage` weitere Entitäten angelegt, um diese abbilden zu können ohne Veränderungen am Quellcode vornehme zu müssen.

`Description`-Objekte verfügen über eine *Id* und einen *Code* (zum Beispiel 'true'). Objekte der Klassen `FilterValue` und `FilterAttribute` verfügen jeweils über eine n:1-Beziehung zur Entität `Description`. Dies bedeutet, dass jedes `FilterValue` und `FilterAttribute` über eine `Description` verfügt, eine `Description` aber von mehreren Objekten der Klassen `FilterValue` und `FilterAttribute` genutzt werden kann. So können zum Beispiel alle `FilterValues`, die den Wahrheitswert *false* eines booleschen Filters abbilden, das selbe Beschreibungselement nutzen.

`Description` verfügt über eine beliebige Anzahl an `DescriptionLanguages`. Hierüber könnte die Mehrsprachigkeit abgebildet werden. `DescriptionLangauges` verfügen über eine *Id*, einen *Code* und einen *Text*. Der *Code* steht in diesen Fall für eine Sprache, also zum Beispiel 'DE', der Text für die Übersetzung in der jeweiligen Sprache, also zum Beispiel 'Nein'. Da Mehrsprachigkeit jedoch als Anforderung abgegrenzt worden ist (vgl. Abschnitt 4.9.2), wird diese Entität nicht weiter beschrieben.

Weiterhin existiert die Entität `FilterSchema`. Sie verfügt über eine *Id* und den *filterSchemaName* sowie über das boolesche Feld *isActive*. Sie hat eine n:m-Beziehung zu `FilterAttribute`. Filterschemata dienen dazu, festzulegen, welche Filter an die App übermittelt werden. So ist es beispielsweise möglich, im Dashboard ein neues `FilterAttribute` mit `FilterValues` anzulegen, diese aber noch nicht an die App zu übermitteln. Dies ermöglicht es beispielsweise, neue Filter anzulegen und erst die `FilterValues` den Locations zuzuordnen, bevor die Nutzer der App diese Filter auswählen können und dann möglicherweise keine Ergebnisse erhalten, weil noch keine `Locations` mit den `FilterValues` verknüpft sind.

Die Verknüpfung der `Locations` mit den `FilterValues` geschieht über die Entität `FilterContainer`. Diese verfügt über eine Id und n:1-Beziehungen zu `Location` und `FilterAttribute`. Außerdem verfügt sie über eine n:m-Beziehung zu `FilterValue`. Konkret bedeutet dies, dass einem `FilterContainer` eine `Location` und ein `FilterAttribute` zugeordnet ist. Daneben verfügt sie über eine Liste von `FilterValues`. In der Umsetzung bedeutet dies, dass jede `Location` für jedes `FilterAttribute`, von welchem ihr mindestens ein `FilterValue` zugeordnet ist, über einen `FilterContainer` verfügt. Diese Struktur dient dazu, um eine Sortierung der `FilterValues` einer `Location` nach `FilterAttributes` innerhalb der App zu realisieren. Durch die `FilterContainer` wären die `FilterValues` direkt nach `FilterAttributes` sortiert.

Die Tabelle 6.1 zeigt exemplarisch den Aufbau und die Struktur von Locations und den zugeordneten Filtern.

6.2 Komponenten

Das Komponentendiagramm dient dazu einen Überblick über die Architektur des RecSys zu schaffen. Das Diagramm ist in Abbildung 6.3 zu sehen. Das Kernstück stellt das RecSys selber dar. Es nutzt und bietet verschiedene Schnittstellen an. Das RecSys soll Bewertungen verarbeiten, Empfehlungen geben und Informationen für das *Dashboard* bereit stellen. Neben dem RecSys gibt es die App, die Empfehlungen anfordern kann und Bewertungen an das RecSys weitergibt. Zusätzlich greift die App auf die Datenbank zu, um Profildaten abzurufen und zu speichern. Außerdem werden Informationen über *Locations* direkt von den *Serviceprovidern* bezogen.

Das RecSys teilt sich in *Empfehlungsserver*, *Bewertungsserver* und *Metrikserver*. Da Empfehlungen und Bewertungen zwei getrennte Vorgänge sind, sollten sie auch getrennt von einander verarbeitet werden. Weshalb es einen *Empfehlungs-* und *Bewertungsserver* gibt. Der *Bewertungsserver* bietet eine Schnittstelle an, über die Bewertungen gegeben werden können. Anschließend werden diese Bewertungen verarbeitet und fließen über die Schnittstelle des DSMSs in den Empfehlungsalgorithmus ein. Des Weiteren werden die Bewertungen in der Datenbank abgespeichert. Der *Empfehlungsserver* wiederum bietet eine Schnittstelle an, über die Empfehlungen angefordert werden können. Diese Empfehlungen werden entweder unmittelbar über die Schnittstelle zurückgegeben oder der *Empfehlungsserver* greift auf die *Push-API* der App zu, um die Empfehlungen zu übergeben. Um kontextabhängige Empfehlungen geben zu können, reichert der *Empfehlungsserver* den Datenstrom mit Kontextdaten und Profildaten aus der Datenbank an. Dieser Datenstrom wird vom DSMS verarbeitet und Empfehlungen werden zurückgegeben. Ein *Serviceprovider* wird für den Kartendienst der App genutzt.

Location		FilterCont.	FilterAttribute		Description		Desc.Lang.			Filt.Val.	Description		Desc.Lang.		
id	name	id	id	type	id	code	id	code	text	id	id	code	id	code	text
967	Akdeniz	4510	110	list	15	Typ	52	DE	Locationtyp	550	660	Restaurant	659	DE	Restaurant
		4514	114	boolean	74	Rauchen	51	DE	Raucherraum	555	669	false	668	DE	Nein
		4516	116	boolean	52	Wifi	21	DE	WLAN	559					
696	Amadeus	4538	110	list	96	Genre	65	DE	Musikgenre	576	681	Disco	680	DE	Disko
													681	EN	Disco
										577	682	Rock	682	DE	Rock
1041	Patio	5585	110	list	15	Typ	52	DE	Locationtyp	550	660	Restaurant	659	DE	Restaurant
		5590	114	boolean	74	Rauchen	51	DE	Raucherraum	554	668	true	667	DE	Ja
		5592	116	boolean	41	PLOT	84	DE	Parkplatz	711	668	true	667	DE	Ja

Tabelle 6.1: Beispieldatensatz von Locations und den ihnen zugeordneten Filtern.

Der *Metrikserver* bezieht von allen vorhanden Schnittstellen zu anderen Systemen relevante Informationen über das RecSys, um sie über die *Dashboard-API* bereitzustellen. In den folgenden Abschnitten dieses Kapitels werden alle Komponenten und Schnittstellen genauer beschrieben.

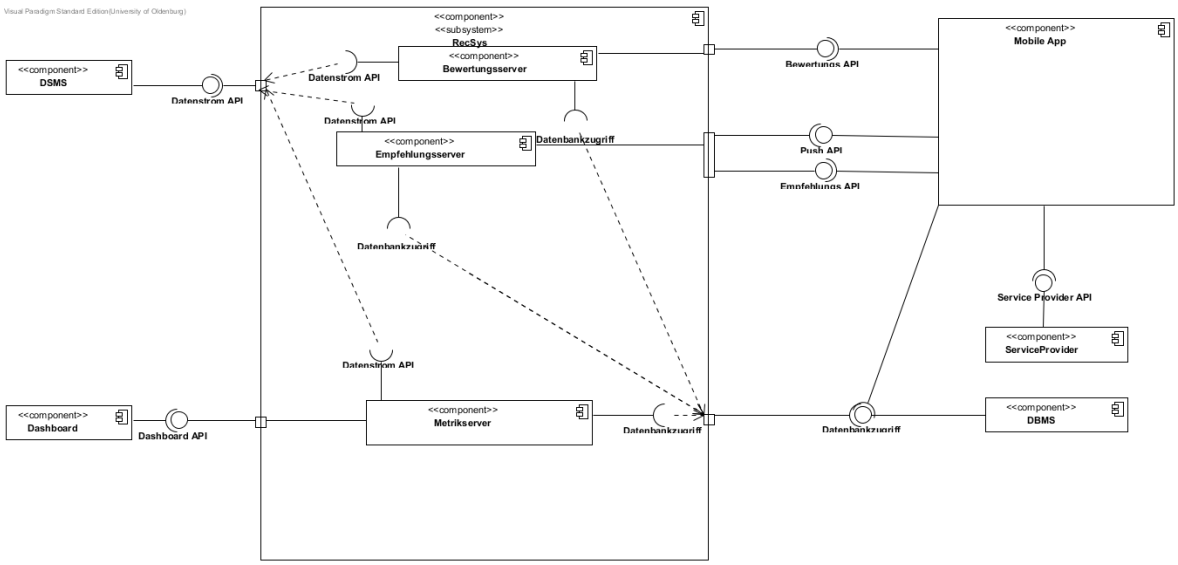


Abbildung 6.3: Komponenten und Kontext des Systems. Quelle: Eigene.

6.3 Architektur

Dieser Abschnitt gibt einen detaillierteren Blick auf das System, die einzelnen Komponenten aus dem Abschnitt 6.2 werden erläutert. Es werden wichtige Technologieentscheidungen getroffen und die Architektur der Middleware und der App aus unterschiedlichen Sichten dargestellt (Baustein-, Laufzeit- und Verteilungssicht).

6.3.1 Randbedingungen und Technologieentscheidungen

Durch die Projekt-Aufgabenstellung sind bereits einige Rahmenbedingungen vorgegeben gewesen. Es sollte zum einen das Datenstrommanagementsystem Odysseus verwendet werden, zum anderen sollte als Client-Applikation eine mobile App für Smartphones erstellt werden. Zu Projektbeginn sind die sämtlichen Kompetenzen der Projektmitglieder ausführlich diskutiert und erfasst worden. Als Schwerpunkte sind hier u.a. Fähigkeiten im Bereich der Java-Entwicklung und im Umgang mit MySQL-Datenbanken identifiziert worden. Diese Ergebnisse stellen die Grundlage für die Entscheidung, den Großteil der Komponenten im Projekt in Java zu implementieren. Gleiches gilt für die Entscheidung, die Mobile-Applikation als Android-Applikation zu entwickeln, da diese fast vollständig in Java implementiert ist. Auch die Middleware ist aus diesem Grund als Webservice in Java unter Verwendung des *Spring-Frameworks* [1] (siehe Kapitel 6.3.2.1) aufgebaut. Für die App-Entwicklung wird *Android Studio* [2] als Entwicklungsumgebung und *Gradle* [3] als Build-Management-Automatisierungstool

[1] *Spring*: `https://spring.io/` besucht am 29.02.2016
[2] *Android Studio*: `http://developer.android.com/sdk/index.html` besucht am 06.03.2016
[3] *Gradle*: `http://gradle.org/` besucht am 06.03.2016

verwendet, das bereits standardmäßig in Android Studio integriert ist. Für die Middleware wird die Entwicklungsumgebung *Eclipse* [4] verwendet, da diese auch bereits für das Odysseus-Studio genutzt wird. Zusätzlich wird das Java Build-Management-Automatisierungstool *Apache Maven* [5] integriert. Das freie, relationale Open-Source-Datenbankverwaltungssystem *MariaDB* [6] wird als DBMS, das parallel zum DSMS Odysseus [7] verwendet wird, genutzt. MariaDB ist ein relationales Open-Source DBMS der früheren Hauptentwickler von MySQL und wird aufgrund der Expertise der Gruppe in diesem Gebiet ausgewählt. Des Weiteren wird das ORM-Framework *Hibernate* [8] verwendet, über welches mit der relationalen Datenbank kommuniziert wird. Auch hier haben vorhandene Kompetenzen und Vorwissen innerhalb der Projektgruppe wieder eine Rolle gespielt. Die Verwendung weiterer Technologien wie das Aufsetzen eines Build-Servers wurden innerhalb der Gruppe diskutiert, jedoch aus Zeitersparnis/Nutzen-Gründen wieder verworfen.

6.3.2 Architektur Middleware

Dieser Abschnitt gibt einen grundsätzlichen Überblick über die Architektur der Middleware. Es wird beschrieben, wie der Quellcode der Middleware grundsätzlich strukturiert ist und welche wesentlichen Konzepte zur Strukturierung herangezogen worden sind.

6.3.2.1 Bausteinsicht der Middleware

Eine Schichtenarchitektur gliedert die Middleware in drei Schichten (siehe Abbildung 6.4):

- Zugriffsschicht
- Service-Schicht
- Domain-Schicht

In der *Zugriffsschicht* werden Schnittstellen bereitgestellt, sodass die Middleware von außen zugreifbar ist. Dazu zählen sowohl REST-Schnittstellen (oder auch *REST-Controller*), die als Zugriffsmöglichkeit für die App dienen, als auch eine Weboberfläche (das Dashboard, siehe Abschnitt 7.10).

Die *Service-Schicht* ermöglicht die Kapselung der Funktionalitäten in einzelne Services. In dieser Schicht werden alle fachlichen Aspekte strukturiert.

[4] *Eclipse*: `https://eclipse.org/` besucht am 06.03.2016
[5] *Apache Maven*: `https://maven.apache.org/` besucht am 06.03.2016
[6] *MariaDB*: `https://mariadb.org/` besucht am 06.03.2016
[7] *Odysseus*: `http://odysseus.informatik.uni-oldenburg.de/` besucht am 06.03.2016
[8] *Hibernate*: `hibernate.org/` besucht am 06.03.2016

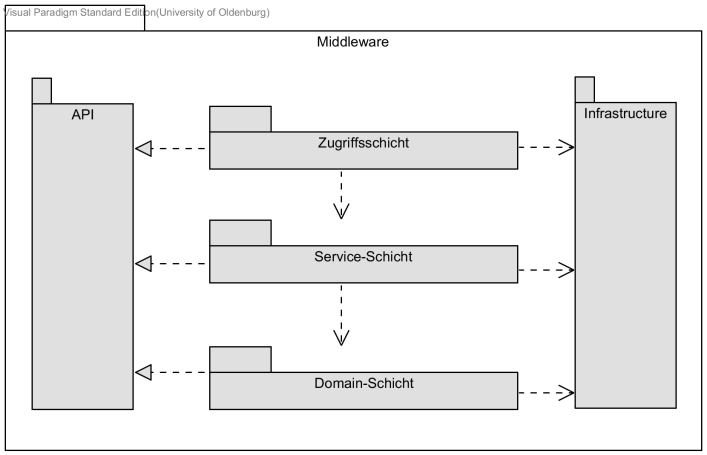


Abbildung 6.4: Schichten der Middleware. Quelle: Eigene.

Die Domain-Schicht enthält infrastrukturelle Komponenten wie die technische Anbindung an Odysseus und an die Datenbank. Die Anbindung an Odysseus wird mittels Web-Sockets implementiert. Für die Anbindung an die Datenbank wird die Datenbankabstraktionsschicht von *Hibernate*[9] verwendet. Hibernate wird zudem als Objekt-relationaler Mapper verwendet. In der Domain-Schicht werden die Entitäten in Form von Klassen erstellt.

Die Entscheidung der Schichtenarchitektur wird wie folgt begründet (vgl. auch [Sta15], S. 110 und die nicht-funktionalen Anforderungen AN-NF-03 und AN-NF-04):

Reduzierung von Abhängigkeiten Jede Schicht erfüllt bestimmte Funktionalitäten. Wenn Schnittstellen bekannt sind, können die Schichten unabhängig von einander implementiert werden.

Klar definierte Schnittstellen Zum Datenaustausch zwischen den Schichten werden Schnittstellen benötigt, die vorher definiert werden müssen.

Austauschbare Schichten Die einzelnen Schichten können einfach durch andere konkrete Implementierungen ausgetauscht werden.

Die Erstellung von Schnittstellen wird in einer eigenen Struktur vorgenommen, der sog. API. In dieser übergreifenden Struktur werden alle Schnittstellen der Domain- und Service-Schichten in Form des Java-Sprachkonstrukts Interface erstellt.

Die *Infrastructure*-Komponente in Abbildung 6.4 dient dazu, übergreifende, v. a. technische Funktionalitäten zu schaffen, die unabhängig von der Domäne sind und von allen Schichten verwendet werden. Dies betrifft v. a. die Serialisierung und Deserialisierung von Objekten mittels *Gson*[10] von nach JavaScript Object Notation (JSON). Zudem wird auch das ordnungsgemäße Beenden von Datenbankverbindungen und Socketverbindungen in diesem Projekt vorgenommen.

Detaillierte Bausteinsichten finden sich konkret zu den einzelnen Anwendungsfällen in Abschnitt 7.

[9] *Hibernate*: `http://hibernate.org/` besucht am 29.02.2016
[10] *Gson* `https://github.com/google/gson`, besucht am 07.04.2016

Technische Datenbankanbindung

Die technische Datenbankanbindung wird mittels der Java Persistence API (JPA) umgesetzt. Für die konkrete Implementierung der JPA wird Hibernate eingesetzt. JPA spezifiziert einen objektrelationalen Mapper und eine Datenbankabstraktionsschicht in Java für verschiedene Datenbanktechnologien. Durch den Einsatz von JPA und Hibernate werden folgende Vorteile angeführt:

- Unabhängigkeit vom zugrundeliegenden Datenbankmanagementsystem
- Einsatz der objektorientierten Programmierung für das Domänenmodell

Das Komponentendiagramm, das die Datenbankanbindung im Projekt verdeutlicht, ist in der Abbildung 6.5 zu sehen.

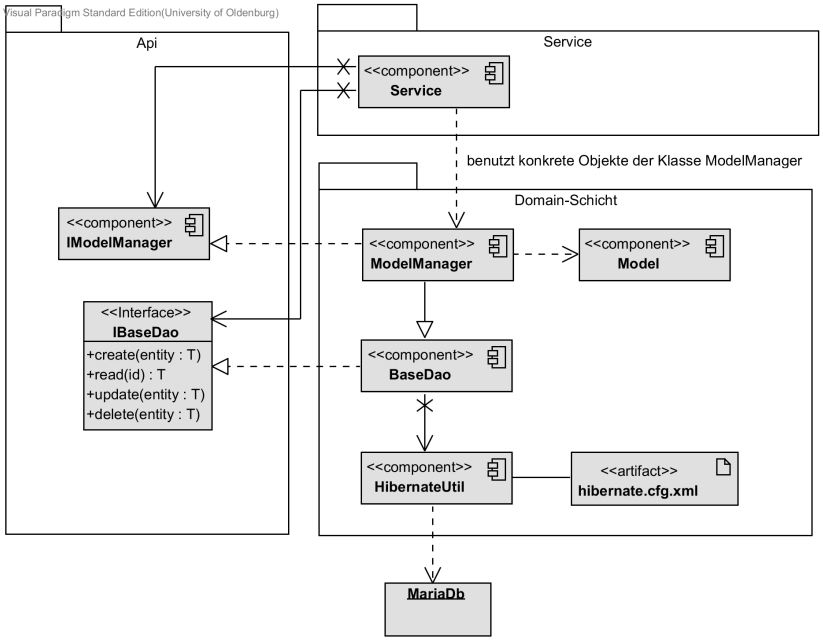


Abbildung 6.5: Struktur der Datenbankanbindung. Quelle: Eigene.

Technische Anbindung von Odysseus

Als DSMS wird Odysseus verwendet. Dieses wird mittels des *Access Frameworks* von Odysseus an die Middleware angebunden (vgl. [Mar15]). Damit wird in Odysseus die Middleware als externe Datenquelle (*Source*) eingebunden. das Anfragen der Empfehlungen jeweils eigene Sources in Odysseus eingerichtet. Damit die Middleware umgekehrt Daten von Odysseus empfangen kann, wird in Odysseus eine *Sink* eingerichtet. Auch hier werden für die Empfehlungen, basierend auf Gesamtbewertungen und auf Detailbewertungen, jeweils eigene Sinks eingerichtet.

Die Einrichtung der Sources in Odysseus wird mittels des `ACCESS`-Operators umgesetzt. Der verwendete *TransportHandler* wird mittels Transmission Control Protocol (TCP) umgesetzt. Das Format des Source-Datenstroms der Bewertungen ist CSV, der Empfehlungsanfragen JSON.

In der Middleware wird die DSMS-Anbindung grundsätzlich in zwei Unterpakete aufgeteilt, in Sources und Sinks. In Abbildung 6.6 wird die DSMS-Anbindung in der Middleware am Beispiel der Request for Recommendation (RfR) veranschaulicht. Sources und Sinks implementieren jeweils das Interface `IDsmsSocket`. Dieses enthält die Methodendeklaration `renewSockets`. Diese Methode soll dazu dienen, dass bei einem Verbindungsabbruch oder beim Neustarten von Queries in Odysseus die Socket-Verbindung neu aufgebaut werden kann.

Unter-Interfaces von `IDsmsSocket` für die Sources sind `IDetailRatingSource`, `IOverallRatingSource`, jeweils für das Abgeben von Bewertungen und `IRecommendationsSource` für das Senden einer Empfehlungsanfrage. In den jeweils konkreten Klassen wird bei Instanziierung durch Spring versucht, eine TCP-Verbindung zum Odysseus-Server aufzubauen. Schlägt das fehl, wird eine Exception geworfen, aber nicht an die aufrufende Schicht hochgereicht. So kann die Instanz korrekt erzeugt werden. Zu einem späteren Zeitpunkt kann versucht werden, eine neue Verbindung aufzubauen.

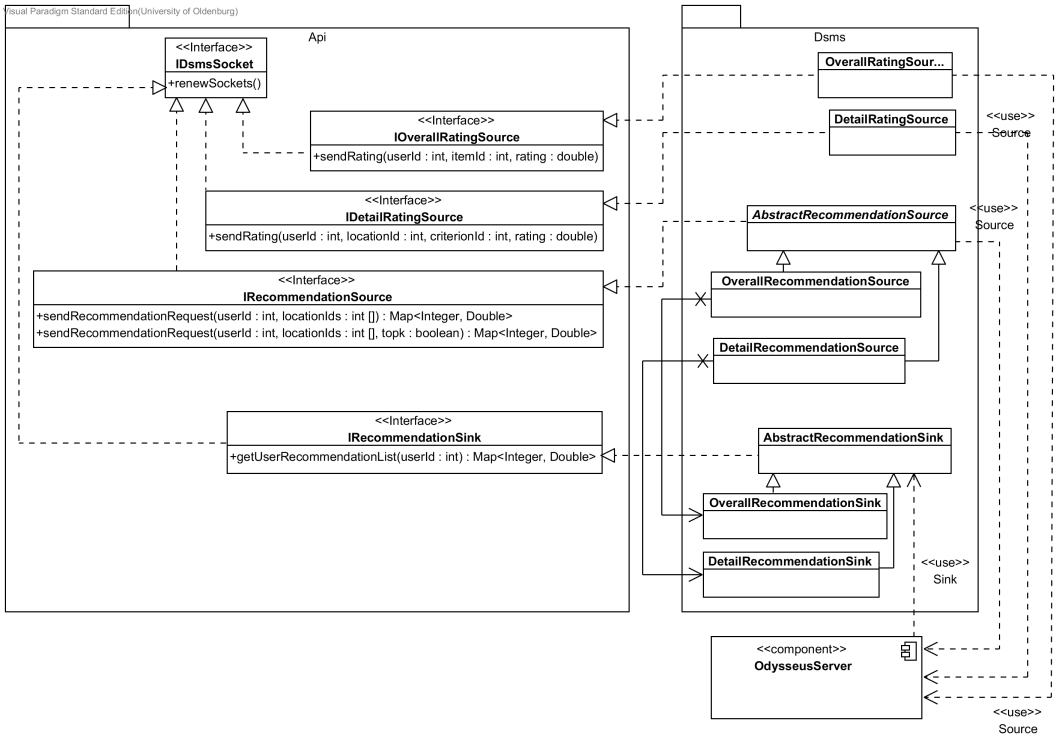


Abbildung 6.6: Struktur der Odysseus-Anbindung. Quelle: Eigene.

Die Sinks werden in Odysseus mit dem Operator `SENDER` eingebunden. Auch hier ist der Transportweg TCP und das Übertragungsformat ist CSV.

In der Middleware dient das Interface `IRecommendationSink` als Schnittstelle zu einer Sink in Odysseus, an die Empfehlungen gesendet werden. Es werden zwei konkrete Klassen implementiert:

1. Eine Klasse, die auf die Empfehlungen, die auf Gesamtbewertungen basieren, horcht und

2. eine Klasse, die auf die Empfehlungen, die auf Detailbewertungen basieren, horcht.

Die `RecommendationSources` greifen auf Instanzen der `RecommendationSinks` zu, um die Empfehlungen über diese abzufragen.

Im DSMS-Projekt werden die Verbindungsdaten Port und Hostname je nach Umgebung in der Datei dsms_**UMGEBUNG*.properties (s. auch Abschnitt 6.3.5) gespeichert, auf die über die Klasse `DSMSProperties` zugegriffen wird.

Damit RfR über die Sources zu der richtigen Antwort über die Sink zugeordnet werden können, wird bei der Anfrage eine eindeutige Request-Id erzeugt (Universally Unique Identifier (UUID)). Diese wird an Odysseus übergeben und über die Sink wieder empfangen. Abbildung 6.7 zeigt das dazugehörige Aktivitätsdiagramm. Die Sinks sind als Threads implementiert. Sources warten auf die Sink. Sobald eine Antwort eingetroffen ist, werden alle wartenden Source-Threads informiert. Die Sources gleichen die Request-Id der Antwort aus der Sink ab. Wenn die zugehörige Antwort vorhanden ist, wird diese genommen, ansonsten wird weiter auf die nächste Antwort gewartet.

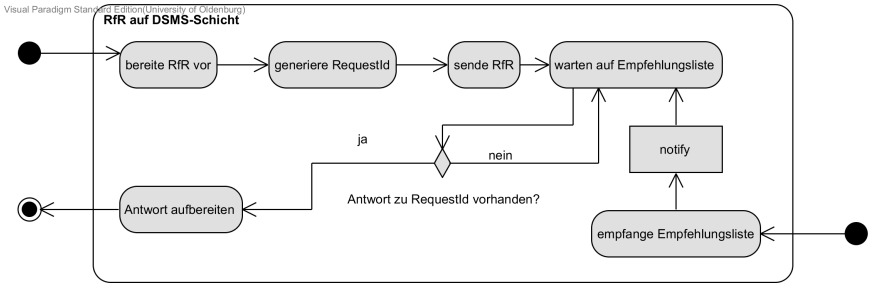


Abbildung 6.7: Ablauf einer Odysseus-Anfrage mit Antwort.

Als Entwicklungsumgebung für die Queries wird das *Odysseus Studio* genutzt. Die Struktur des Projektes im Odysseus Studio ist in zwei Verzeichnisse aufgeteilt: Queries für die Gesamtbewertung (*Filter*-Verzeichnis) und für die Detailbewertungen (*Detailbewertung*-Verzeichnis). Unter diesen ist jeweils ein Verzeichnis *Sources*, welches die Queries zur Einrichtung der Sources enthält. Ein weiteres Verzeichnis ist *Queries*, in dem Queries sowohl für das Abgeben der Bewertungen als auch für die Empfehlungsanfragen (inkl. Einrichtung der Sinks) zu finden sind.

Dependency Injection

Zur Reduzierung von Abhängigkeiten soll *Dependency Injection* verwendet werden. Dadurch werden Klassen, die von anderen Klassen abhängen, voneinander entkoppelt. Spring unterstützt eine Implementierung von Dependency Injection. Abbildung 6.5 zeigt, dass die Service- und Domain-Schicht von einander entkoppelt werden. Der Service greift auf Interfaces der API zu, arbeitet zur Laufzeit jedoch mit Objekten der konkreten Klasse (siehe Beziehung *«benutzt konkrete Objekte der Klasse ModelManager»*). Der *IoC-Container* von Spring ist für die konkrete Injizierung der Domain-Klassen in die Services verantwortlich. (vgl. [Sta15], S. 77 f. und [JHD+16], S. 3)

6.3.2.2 Laufzeitsicht auf die Middleware

Ein Charakteristikum der Schichtenarchitektur ist, dass jede Schicht nur von der nächst höheren Schicht angesprochen wird. Der grundsätzliche Ablauf wird in 6.8 in Form eines Aktivitätsdiagramms dargestellt.

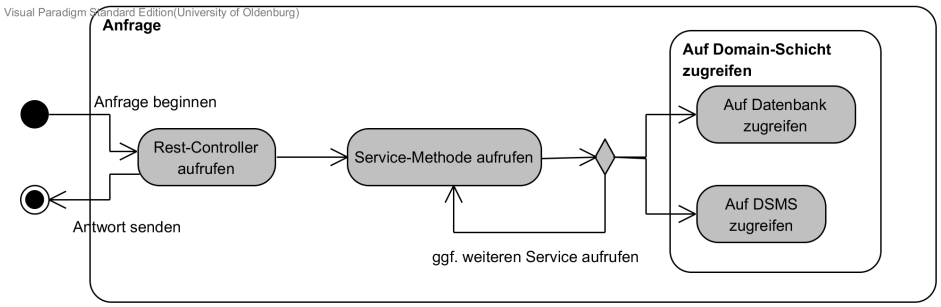


Abbildung 6.8: Schichten der Middleware. Quelle: Eigene.

6.3.3 Architektur App

An dieser Stelle wird auf eine vollständige Beschreibung einer allgemeinen Android-Systemarchitektur verzichtet, da dies für den Anwendungsfall der Projektgruppe nicht von größerer Relevanz ist. Im Folgenden werden die wichtigsten Bausteine einer Android-Applikation erläutert, da diese in der Recommender-App die Grundstruktur bilden.

6.3.3.1 Grundlegende Bausteine in Android

Jede Android-App besteht aus mindestens einer, meist jedoch mehreren *Activities*, zwischen denen sich navigieren lässt. Über eine Activity können andere Activities gestartet werden. In diesen werden jede Nutzerinteraktion verarbeitet und Ausgaben realisiert, sie sind also das, was der Nutzer auf seinem Endgerät sieht und womit er interagiert. Hinter Activities steckt die Logik einer Applikation. Eine Activity kann unterschiedliche Zustände haben: Wird sie ausgeführt, ist ihr Zustand *active*. Wenn z.B. Activity A Activity B startet, ist Activity A *stopped*, aber das System speichert ihren Zustand. Wenn der Zurück-Button betätigt wird, wird der Zustand wieder hergestellt. Activity B ist dann *destroyed*, das heißt ihr Zustand wird nicht vom System wieder hergestellt.

Activities müssen von übergeordneten Activities erben und in der *Manifest-XML* hinterlegt werden, um eine Activity zu sein.

Content Provider verwalten in Android den Zugriff auf strukturierte Daten. Ressourcen wie Bilder, Schaltflächen, Styles, Texte und Farben werden in XML-Layoutdateien definiert. Damit ist die Logik in Java vom Layout in XML getrennt. Der Zugriff auf Dateien, die sich auf dem Endgerät befinden, erfolgt standardmäßig über das Paket *java.io*, das die normalen I/O Methoden von Java implementiert. So kann auf das Linux Filesystem von Android zugegriffen werden, wobei dieser limitiert ist. Jede Applikation hat ihren eigenen Bereich im Dateisystem unter */data/data/packagename*. Android nutzt *SQLite*. Die SQLite-Datenbank wird selbst als Datei unter */data/data/packagename/databases* angelegt. Jeder Activity ist eine View zugeordnet. Eine View ist eine XML-Ressourcendatei, in der die Informationen über die grafische Oberfläche beschrieben werden. Sämtliche dieser Layoutdateien sind in einem Projekt unter den *Application-Ressources*, wo auch Bild-, Audio- oder Stringdateien gespeichert werden, hinterlegt. Die Ressourcen sind von dieser zentralen Stelle aus von überall in der App aus aufrufbar. In der Android-Manifest-XML werden diverse Metadaten der Applikation hinterlegt. Dazu zählen z.B. Zugriffsrechte oder Informationen über Android-Versionen. Zudem muss jede Activity der Applikation an dieser Stelle registriert sein.

Services sind dazu da, (langlebige) Prozesse im Hintergrund laufen zu lassen.

Über sogenannte *Broadcast Receiver* reagiert die App auf Systemereignisse. Als *Intent* werden Nachrichten zwischen den Anwendungen bezeichnet. Sie werden für die Kommunikation zwischen Activities, Services und Receivern genutzt. Andere Activities lassen sich über einen Intent aufrufen.

6.3.3.2 Aufbau der App

Die Grundstruktur der von der Projektgruppe entwickelten App ist maßgeblich vom oben erläuterten Konzept beeinflusst. Die Architektur der App (siehe Abbildung 6.9) teilt sich in drei wesentliche Komponenten auf:

1. GUI-Komponenten (Activities, Fragmente, Bild-, Text und Layoutressourcen)
2. Services (ORM, Server-Kommunikation)
3. Models (Für ORM, JSON-Parser)

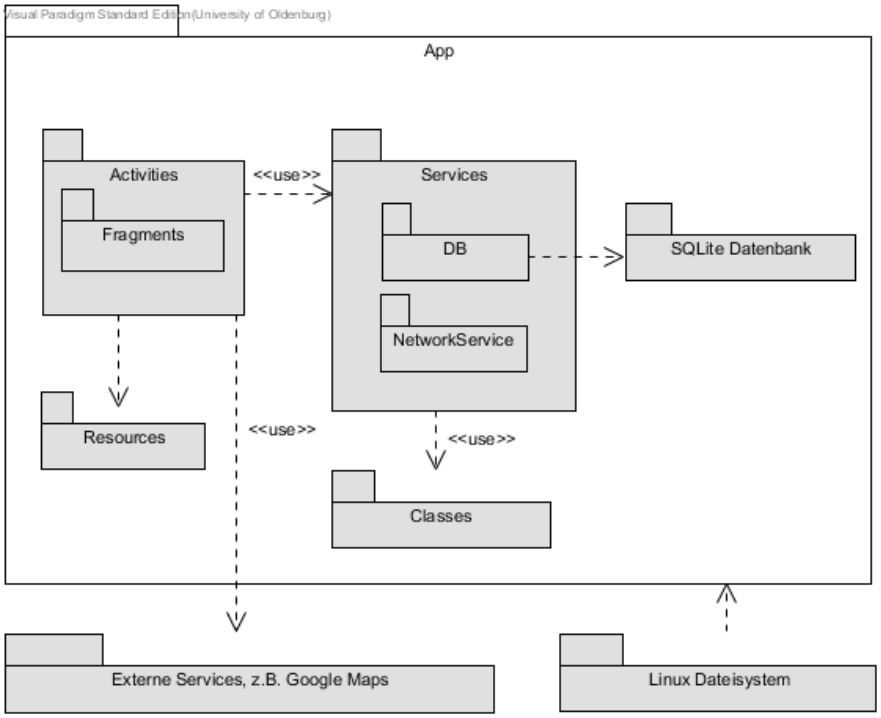


Abbildung 6.9: Architektur der App. Quelle: Eigene.

Die GUI-Komponenten umfassen alle Activities, Fragmente und Ressourcen. Eine XML-Layout-Ressource und eine Activity bilden einen *Screen*. Die Activities hier bilden die Logik als Controller hinter der View. Fragmente sind einzelne View-Controller-Objekte, die in das Activity-Layout eingebunden werden. Activies können über diese Fragmente (teil-)strukturiert werden. Die Projektgruppe verwendet ein Listen-Fragment, das einmal implementiert wird und in seiner Grundstruktur in unterschiedlichen Activies wiederverwendet werden kann. Auch das Menü, das in den meisten Sichten

aufrufbar ist, wird in seiner Form in vielen Activies aufgerufen und verwendet. Im Ressourcen-Ordner (*/res*) sind sämtliche Bilddateien, Strings und XML-Layoutdateien angelegt. Als zweiten Baustein liegt im *manifests*-Ordner die Manifest-XML der App.

Die Services umfassen im Wesentlichen zwei verschiedene Service-Funktionalitäten: Der erste Service ist der *Network-Service*. Dieser regelt die Anfragen (*Requests*) an die Middleware und ihre Rückmeldung (*Response*). Dafür wurde die Bibliothek *OkHttp* [11] verwendet. Diese ermöglicht es, über asynchrone Tasks Anfragen an die REST-Schnittstelle der Middleware zu schicken und die Antworten auszuwerten. Der zweite Service ist die Helfer-Klasse *DB-Helper*. Durch die Verwendung des *Object-relational-mapping*-Framework *RushORM* [12] wird ein Offline-Betrieb der App ermöglicht. Dieses Framework greift vor allem auf die dritte Komponente der *Models* zu.

Models bilden Daten ab, die zum Erstellen von Layout-Komponenten oder zur Kommunikation zwischen der App und dem Middleware-Server benötigt werden. Diese Modelle werden hauptsächlich von den Services verwendet. Über RushORM werden diese Klassen als Objekte behandelt, die persistent auf dem Endgerät gespeichert werden sollen. Außerdem werden sie verwendet, um Responses von der Middleware einfach auswerten zu können. Die JSON-Strings werden mithilfe der Bibliothek *LoganSquare* [13] geparst und verwendet werden können.

6.3.4 Architektur Dashboard

In diesem Abschnitt wird die grundsätzliche Architektur des Dashboards beschrieben. Weiterhin wird erklärt, welche Technologien eingesetzt werden. Es wird darauf verzichtet, die einzelnen Technologien im Detail zu beschreiben.

Für die Umsetzung des Dashboards wird das *MVC Web Framework* von Spring verwendet. Mit dem MVC Framework ist es möglich dynamische und REST-konforme Webapplikationen mit Java umzusetzen. Außerdem erlaubt das Framework die Nutzung von etablierten Technologien, wie z. B. JPA. MVC steht dabei für Model, View und Controller. Das Model ist ein Datencontainer und erfüllt keine weiteren Funktionen. Die View stellt die Daten aus dem Model mittels *Java Server Pages (JSP)* in einem Webbrowser dar. Es wird JSP verwendet, da es die Möglichkeit bietet dynamische Inhalte mit den statischen Inhalten von HTML zu mischen. Weiterhin müssen keine weiteren Sprachen, wie z. B. *PHP* verwendet werden. Der Controller wird angesprochen, um die entsprechenden Services aufzurufen und die benötigten Daten mittels des Models an die View zu übergeben [14]. Die Architektur des Dashboards mit den Komponenten des MVC Frameworkes ist in Abbildung 6.10 zu erkennen.

[11] *OkHttp*: `http://square.github.io/okhttp/` besucht am 26.02.2016

[12] *RushOrm*: Bibliothek für objektrelationales Mapping in Android, `https://www.rushorm.com`, besucht am 26.02.2016

[13] *LoganSquare*: `https://github.com/bluelinelabs/LoganSquare`, besucht am 26.02.2016

[14] *MVC*: `http://docs.spring.io/autorepo/docs/spring/3.2.x/spring-framework-reference/html/mvc.html` besucht am 25.03.2016

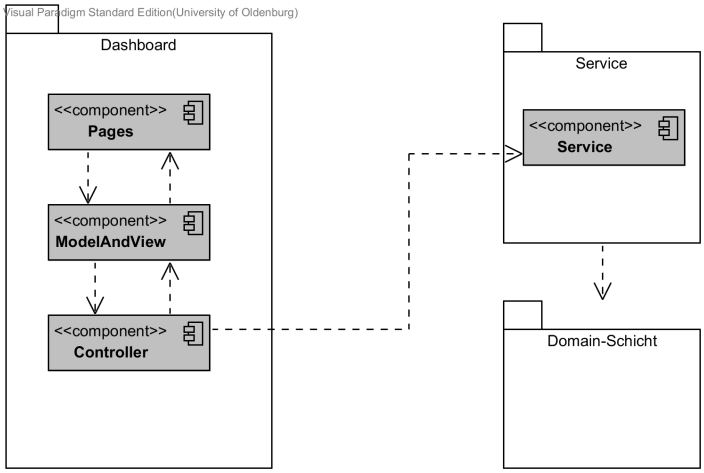


Abbildung 6.10: Architektur des Dashboards. Quelle: Eigene.

Das Dashboard stellt die Zugriffsschicht dar. Innerhalb des Dashboard-Packages werden die REST-Controller bereitgestellt, um auf die Service-Schicht der Middleware (s. 6.3.2), zugreifen zu können. Die Controller liefern im Unterschied zum REST-Package jedoch kein JSON-Objekt, sondern ein Objekt vom Typ *ModelAndView*, welches vom MVC Framework bereitgestellt wird, zurück. Dieses ModelAndView Objekt wird an die JSP weitergegeben. Die Daten können, nachdem sie auf den JSPs dargestellt worden sind, durch unterschiedliche JavaScript Methoden durch den Nutzer verändert werden. Dafür werden Bibliotheken wie JQuery und Bootstrap verwendet. In einigen Anwendungsfällen besitzt der Nutzer die Möglichkeit, Daten über das Dashboard zu löschen oder zu bearbeiten. Weiterhin wird für einige Anwendungsfälle *Ajax* verwendet, um asynchrone Requests schicken zu können.

Die Architekturentscheidung fiel aus folgenden Gründen:

REST-konforme Implementierung Das Dashboard berücksichtigt ebenfalls die Schichtentrennung und wird über einen Controller aufgerufen.

Spring MVC Framework Da im Voraus bereits die Technologieentscheidung auf Spring gefallen ist, wird die Implementierung des Dashboards durch die Verwendung des MVC Frameworks realisiert.

Wiederverwendung von Servicefunktionalitäten Bereits geschriebene Anwendungslogik kann im Dashboard-Projekt wiederverwendet werden. Es werden nur geringfügige Erweiterungen durchgeführt.

6.3.5 Verteilungssicht des Systems

Dieser Abschnitt beschreibt die Verteilungssicht der Middleware, der Odysseus-Server und der Datenbank sowie der Git-Repositories während der Projektphase.

Es existieren jeweils drei Umgebungen. Es gibt die sog. feature-Umgebung, in der neue Features resp. fachliche Funktionalitäten implementiert werden. Es gibt die sog. develop-Umgebung, auf der gereviewte Features zusammengeführt werden. Die develop-Umgebung wird somit während eines Sprints laufend aktualisiert. Zuletzt gibt es die stage-Umgebung, auf der nach Sprintende die aktuelle develop-Umgebung deployt wird.

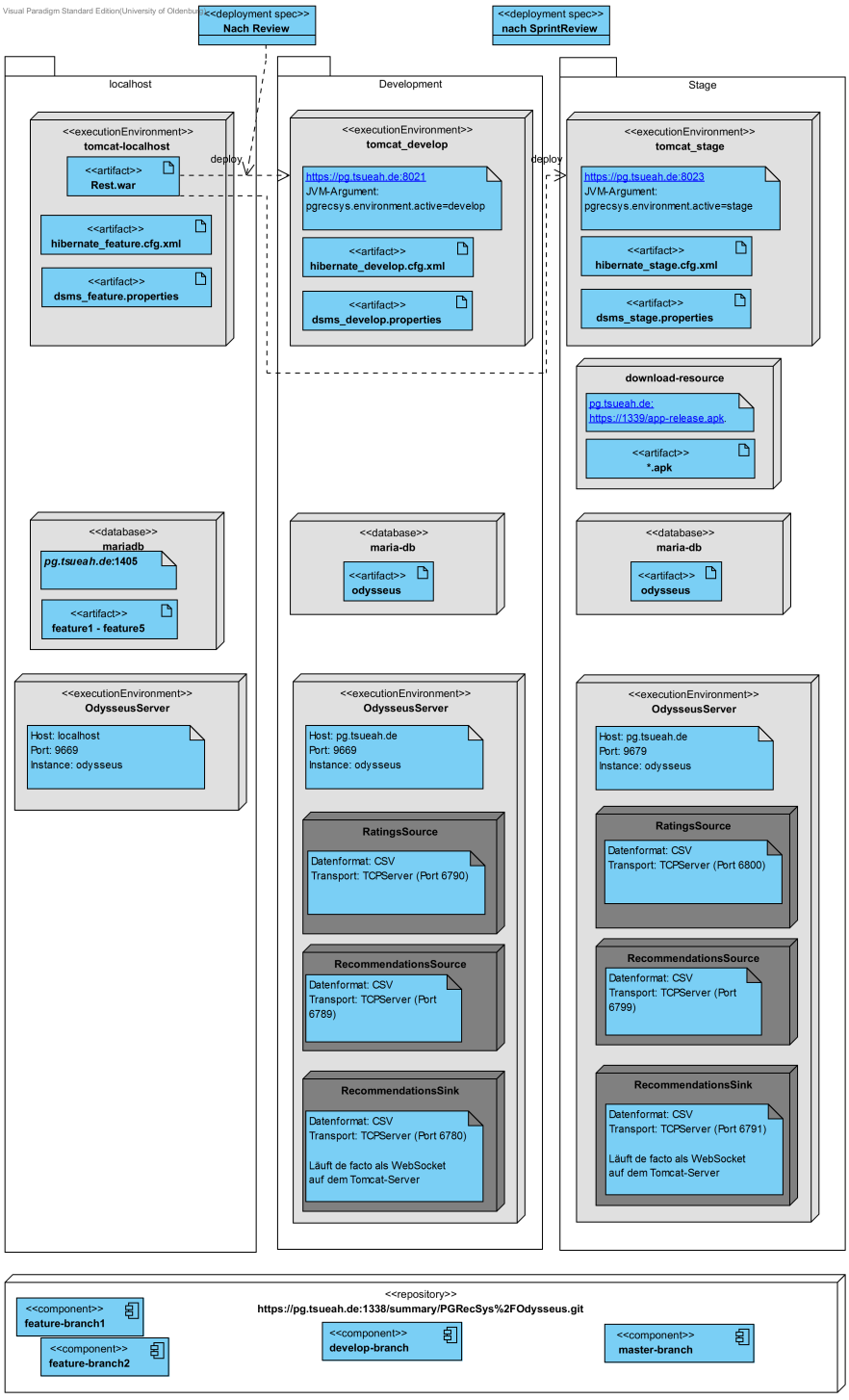


Abbildung 6.11: Schichten der Middleware. Quelle: Eigene.

Die Sources werden in entsprechende Branches eines git-Repositories (mehrere Feature-Branches, einen develop-Branch und einen master-Branch, der die Sources der Stage-Umgebung impliziert)

gespeichert. Pro Umgebung gibt es mindestens einen Tomcat-Server (in der feature-Umgebung lokal), eine Datenbank und einen Odysseus-Server (in der feature-Umgebung lokal).

Damit die Umgebungen der Tomcat-Instanzen klar definiert werden können, werden die Tomcat-Instanzen mit einem Argument für die Java Virtual Machine (JVM) gestartet: `pgrecsys.environment.active`. Dieser wird entweder auf `develop` oder `stage` gesetzt. In der Middleware wird der Parameter abgefragt und je nach Ausprägung entsprechend reagiert (z. B. Laden umgebungsabhängiger Konfigurationen).

Für jede Umgebung gibt es eigene Schemata für die Datenbank. Zudem werden unterschiedliche Ports für die Socketverbindungen definiert.

6.4 Grafische Benutzeroberfläche

Dieser Abschnitt beschreibt die grafische Benutzeroberfläche der App und des Dashboards.

6.4.1 App-seitige Benutzeroberfläche

Im folgenden Abschnitt wird die Benutzeroberfläche der zu erstellenden Android-Applikation beschrieben. Die Mockups sollen vor allem einen Überblick über die graphische Umsetzung des Applikationskonzeptes geben, aber auch Sachverhalte wie die Navigation durch die unterschiedlichen Views verständlicher machen. Die Mockups werden mit dem Online-Mockup-Tool *Ninja-Mock*[15] erstellt.

Die Login-Seite (siehe Abbildung 6.12) zeigt die geplante Umsetzung der Anforderungen AN-F-11.01, AN-F-11.03 und AN-F-11.04.

Die Startseite (Home View) soll die grafische Umsetzung der Anforderungen AN-F-01.05, AN-F-02.02 und AN-F-02.03 beinhalten. Die Listen-Ansicht (siehe Abbildung 6.14) besteht aus den drei verschiedenen Bereichen Filter, Check-In-Bereich und der Empfehlungsliste, die geordnet nach ihrem Ranking die Top-k-Empfehlungen anzeigen sollen. Über die Liste sollen grundsätzliche Informationen über die empfohlenen Locations angezeigt werden. Dazu zählt die vorhergesagte Bewertung. Über den Button rechts im Listenelement kann der Nutzer eine Location als besucht markieren (*einchecken*). Zudem gelangt der Nutzer per Klick auf die Location zur jeweiligen Detailseite. Der Filter-Bereich lässt sich auf-und zuklappen, um Platz auf der View zu sparen.

Die Karten-Ansicht (siehe Abbildung 6.16) besteht aus den Bereichen Filter, Check-In-Bereich und der Karte, auf der die Empfehlungen dargestellt werden. Über das Karten-Icon lässt sich zwischen den beiden Darstellungsformen hin-und her wechseln. Die Top-k Empfehlungen werden als Marker in der Karte auf ihrem Standort dargestellt. Per Klick auf einen Marker öffnet sich eine Kurzbeschreibung (analog zur List-View) der Location (siehe Abbildung 6.17), sowie ein Button zum Aufruf der Detailansicht und ein Button für das Merken der Location für einen späteren Besuch (*Check-In*).

Die Check-In-View (siehe Abbildung 6.20) stellt die Locations dar, in die ein Nutzer eingecheckt hat (siehe AN-F-01.05). Diese werden in einer geordneten Liste dargestellt, bei der zuletzt eingecheckte Locations ganz oben stehen und danach absteigend sortiert werden. Über die Liste sollen, wie in der Empfehlungsliste, grundsätzliche Informationen über die empfohlenen Locations angezeigt werden. Auch hier gelangt der Nutzer per Klick auf die Locations zur jeweiligen Detailseite. Zusätzlich wird

[15] *Ninja-Mock*: Online-Mockup-Tool, `http://ninjamock.com/`, besucht am 18.08.15.

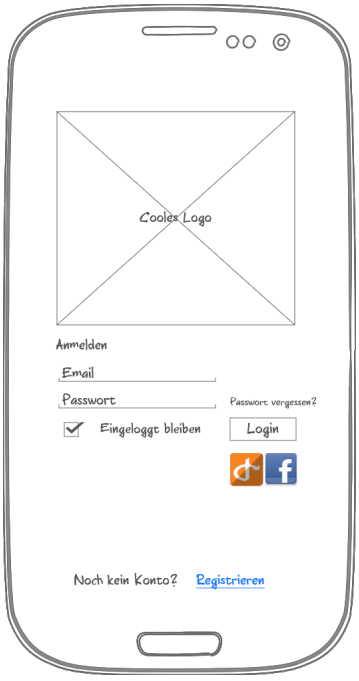


Abbildung 6.12: Mockup der Login-View mit verschiedenen Funktionalitäten zur Benutzerverwaltung. Quelle: Eigene.

Abbildung 6.13: Das Menu zur Navigation durch die App. Quelle: Eigene.

Abbildung 6.14: Die Empfehlungen werden dem Nutzer in einer Liste angezeigt. Quelle: Eigene.

für jede Location die vom Nutzer abgegebene Bewertung angezeigt, sofern vom Nutzer bereits eine Bewertung übermittelt worden ist. Ist dies nicht der Fall, kann er über eine Verlinkung (Hier: Sterne mit Fragezeichen) die Bewertungsansicht aufrufen (siehe auch Abbildung 6.18).

Die Bewertungs-View (siehe Abbildung 6.18 und 6.19) lässt sich in drei wesentliche Bereiche aufteilen und veranschaulicht die grafische Umsetzung der Anforderungen AN-F-01.01, AN-F-01.02, AN-F-01.03 und AN-F-01.04. Die Gesamtbewertung des Besuchs wird in Sternen abgegeben. Dabei einspricht 1 Stern dem Attribut *sehr schlecht* und 5 Sterne dem Attribut *sehr gut*. Über das Kommentar-Textfeld sollen Kommentare abgegeben werden können. Die Anforderung AN-F-09 für das Teilen von Locations mit Freunden spiegelt sich in der Benutzeroberfläche in den Buttons unterhalb des Kommentarfeldes wieder.

Das Menü (siehe Abbildung 6.13), das über jede Sicht aufrufbar sein soll, bedient Kriterien der Bedienungsfreundlichkeit (siehe AN-NF-05). Das Menü besteht aus den Menüpunkten *Location Suche*, *Checked-In*, *Favoriten*, *Profil*, *Empfehlungen*, *Filter*, *Einstellungen* und *Ausloggen*. Sämtliche Views sollen über das Menü erreichbar sein.

Das Nutzerprofil (siehe Abbildung 6.23) zeigt die Sicht, in der die Anforderungen AN-F-10, AN-F-11.02 und AN-F-11.05 umgesetzt werden sollen.

Abbildung 6.15: Durch das Auswählen einer Location kann der Nutzer einen Besuch bewerten. Quelle: Eigene.

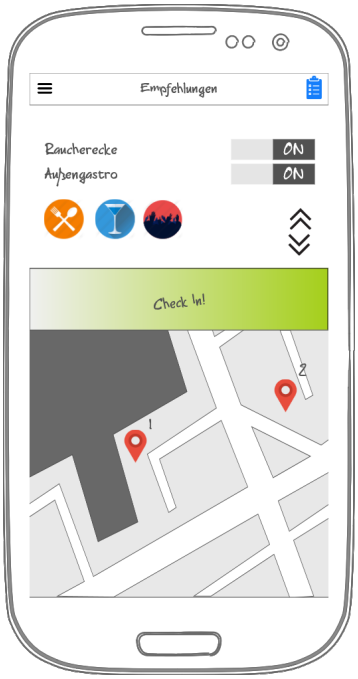


Abbildung 6.16: Die Empfehlungen werden dem Nutzer in einer Karte angezeigt. Quelle: Eigene.

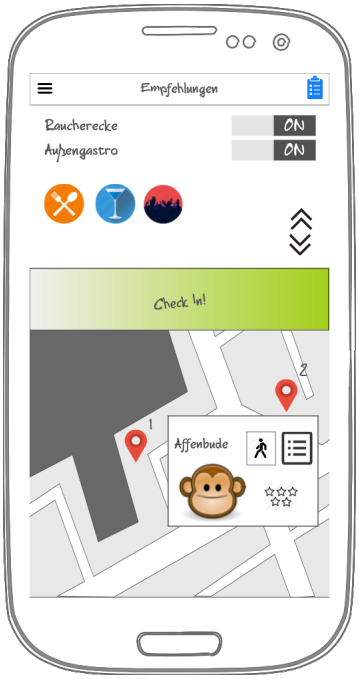


Abbildung 6.17: Die Details der Location können in der Kartenansicht betrachtet werden. Quelle: Eigene.

In den Einstellungen (siehe Abbildung 6.22) sollen Anforderungen an die Benutzerfreundlichkeit (siehe AN-NF-05) realisiert werden. Die Einstellmöglichkeiten sollen auf verschiedene Arten gestaltet werden können: über Buttons, modale Dialog-Popups oder auch Checkboxen.

Über Verlinkungen in Location-Listen oder in der Kartenansicht ist es möglich, sich weitere Informationen zu einer Location in der Detailansicht (siehe Abbildung 6.24) anzeigen zu lassen. Hier werden die Anforderungen AN-F-04.01, AN-F-04.02 und AN-F-04.03 in ihrer grafischen Realisierung simuliert. Die Detailansicht besteht aus den allgemeinen Informationen über die Location, einem Bereich für die Verlinkung zur Favoritensicht und zur Detailbewertung (siehe Abbildung 6.24) sowie einem Kommentarbereich, der die zeitlich sortierten Kommentare inklusive hinterlegtem Nutzer anzeigt.

Die Favoritensicht (siehe Abbildung 6.21) zeigt alle Favoriten in einer Liste an und wird auf Grundlage der Anforderung AN-F-07 erstellt. Die Bausteine dieser Sicht sind die selben wie die der Empfehlungsliste.

Dem Ausloggen-Button im Menü liegt die Anforderung AN-F-11.06 zugrunde.

Abbildung 6.18: Durch eine 5-Sterne-Skala kann der Nutzer einen Besuch bewerten, ein Kommentar abgeben und es mit Freunden teilen. Quelle: Eigene.

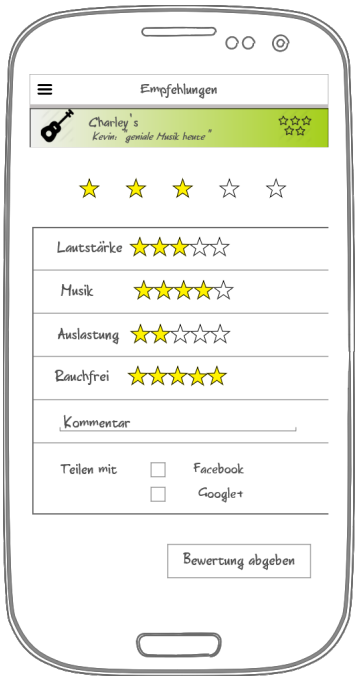


Abbildung 6.19: Der Nutzer kann nach der einfachen Bewertung durch eine Detail-Ansicht seine Bewertung verfeinern. Quelle: Eigene.

Abbildung 6.20: Damit der Nutzer weiß, in welche Locations er sich eingetragen sind diese in einer Liste aufgeführt. Quelle: Eigene.

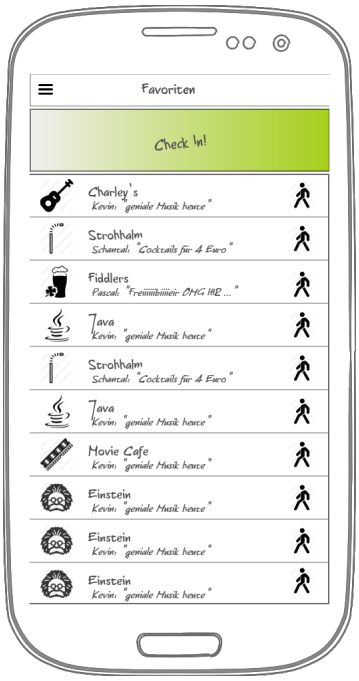


Abbildung 6.21: Der Nutzer kann in einer Liste seine favorisierten Locations betrachten. Quellle: Eigene.

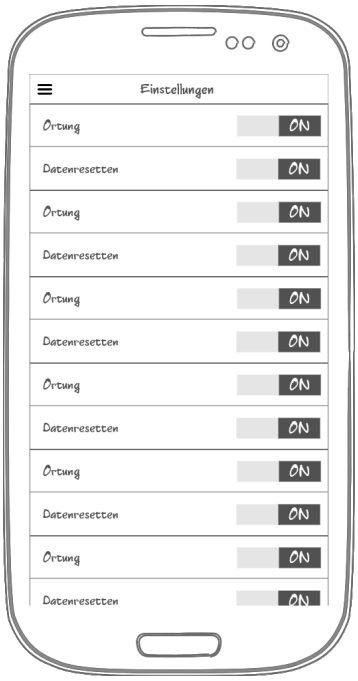


Abbildung 6.22: Der Nutzer kann in den Einstellungen die App konfigurieren. Quelle: Eigene.

Abbildung 6.23: Der Nutzer kann in seinem Profil persönliche Informationen angeben. Quelle: Eigene.

Abbildung 6.24: In der Detailansicht der Location sind wichtige Informationen und die letzten Kommentare der Bewertungen zu sehen. Quelle: Eigene.

Abbildung 6.25: Detailbewertung einer Location. Quelle: Eigene.

6.4.2 Dashboard-seitige Benutzeroberfläche

Dieser Abschnitt beschreibt die Benutzeroberfläche des Dashboards, mit dessen Hilfe der Admin den Systemstatus überprüfen und Einstellungen am System vornehmen kann (siehe AN-F-12). Die Mockups wurden ebenfalls mit Ninjamock (siehe Kapitel 6.4.1) erstellt und sollen einen Überblick über die Funktionalitäten des Dashboards geben. Insgesamt wurden fünf Seiten für den Admin realisiert, auf denen er Einstellungen vornehmen oder Informationen abrufen kann. Auf der Seite *LiveRequestsForRecommendations* kann der Admin die aktuellsten Empfehlungen einsehen, die von Odysseus generiert und an die Nutzer gegeben werden (Abbildung 6.26). Diese Seite dient als grafische Übersicht und Information für den Admin um die Anforderung AN-F-12.02 umzusetzen.

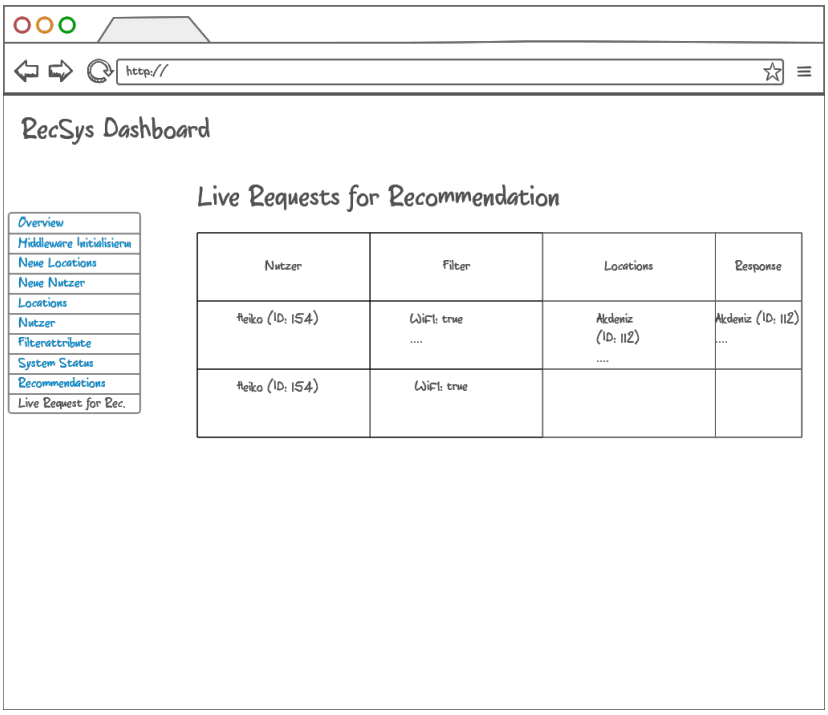


Abbildung 6.26: Der Admin kann die aktuellsten Recommendations einsehen. Quelle: Eigene.

Die Seite *Middleware initialisieren* (Abbildung 6.27) gibt dem Admin die Möglichkeit, eine neue Middleware zu initialisieren (siehe Anforderung AN-F-12.07. Hier kann der Admin Testdaten wie Testlocations, Testnutzer und ein zugehöriges Filterschema initialisieren und initiale Bewertungen abgeben, damit das komplette System läuft.

Für die Übersicht über die aktuellen Locations und das dazugehörige Filterschema dienen die Seiten *Locationverwaltung* (Abbildung 6.28) und *Filterattribute* (Abbildung 6.29). Die Locationverwaltung soll alle Locations in Form einer Tabelle übersichtlich darstellen. Zudem soll der Admin hier die Möglichkeit erhalten, die vollständigen Details einer Location einsehen oder ggf. eine Location löschen oder hinzufügen zu können. Hierdurch wird die Anforderung AN-F-12.06 umgesetzt werden.

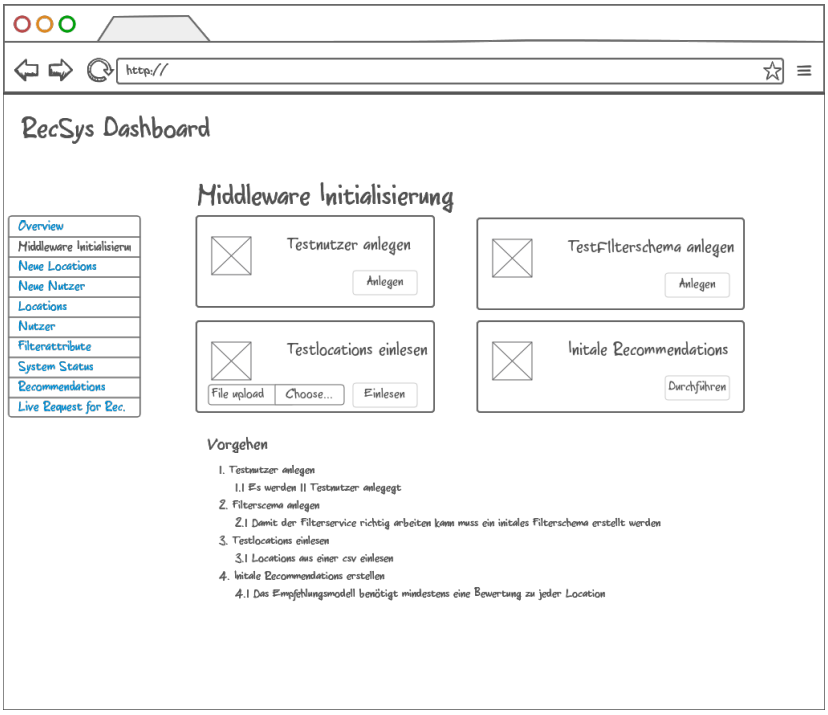


Abbildung 6.27: Der Admin Middleware mit Testdaten initialisieren. Quelle: Eigene.

Auf der Seite *Filterattribute* kann der Admin Einstellungen am aktuellen Filterschema vornehmen. Dazu gehört das Anlegen und Löschen von einem neuen Filterschema, das Hinzufügen von neuen Filtern zum aktuellen Schema sowie das Umbenennen, Anlegen und Löschen von Filterattributen.

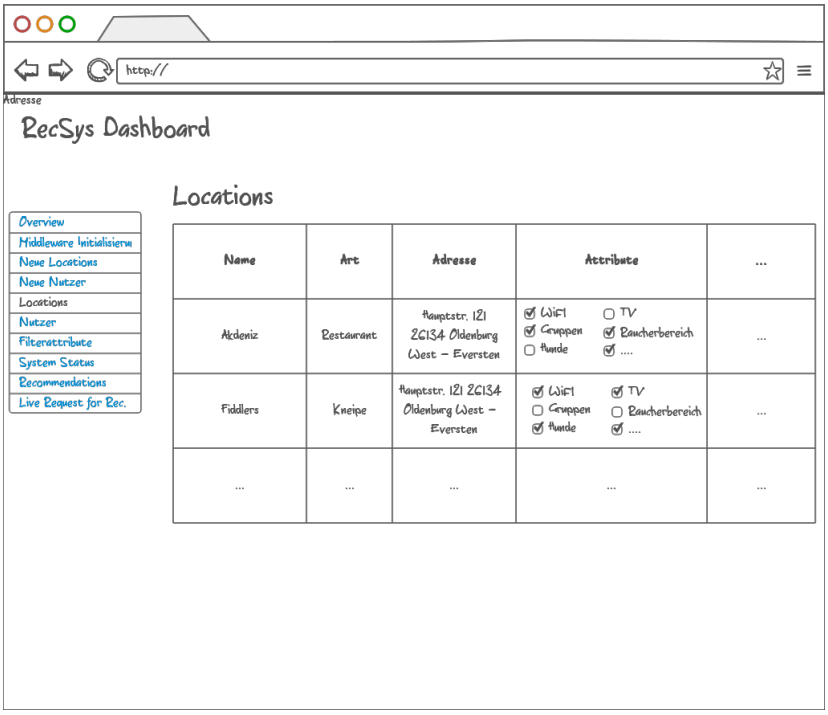


Abbildung 6.28: Der Admin kann alle gespeicherten Locations einsehen. Quelle: Eigene.

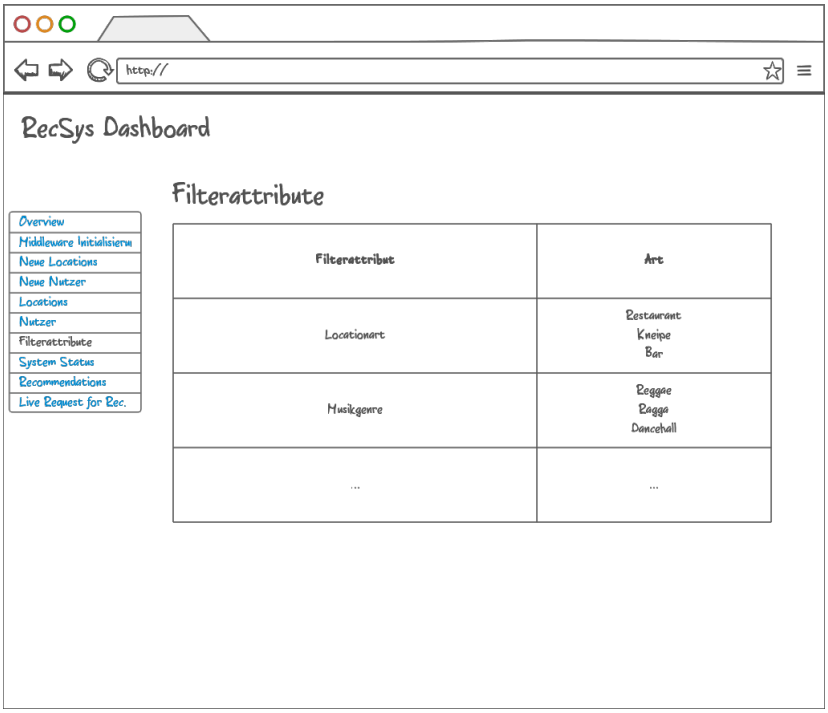


Abbildung 6.29: Der Admin kann das aktuelle Filterschema einsehen und bearbeiten. Quelle: Eigene.

Die fünfte Seite soll dem Admin die Möglichkeit geben, mithilfe des Dashboards Einstellungen am Top-k-Operator vorzunehmen. Zusätzlich ist hier in Form eines Graphens der Verlauf des `RMSE` zu beobachten. Dies stellt die Umsetzung der Anforderung AN-F-12.01 dar.

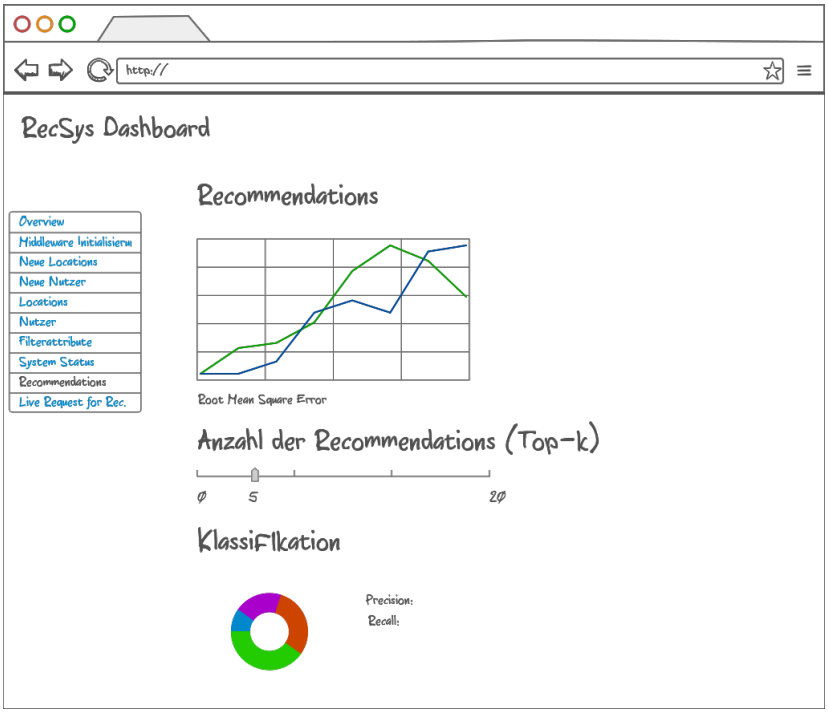


Abbildung 6.30: Der Admin kann den Top-k ändern und den RMSE einsehen. Quelle: Eigene.

7 Implementierung

Dieses Kapitel beschreibt die umgesetzten Anforderungen, welche in Abschnitt 5.2 definiert werden. Dabei werden wichtige Implementierungsdetails sowohl für die App, die Middleware (inkl. REST-Schnittstellen), Odysseus, Datenbank und Dashboard erklärt.

7.1 Locations als besucht markieren

Im Folgenden wird die Umsetzung der Anforderung AN-F-01.05, die das Markieren einer Location als besucht beschreibt, erläutert. Eine solche Markierung soll signalisieren, dass der Nutzer sich eine bestimmte Location (beispielsweise aufgrund einer Empfehlung) ausgesucht hat und diese nach deren Besuch bewerten möchte. Gleichzeitig sollen diese Besuche als eine Besuchshistorie dem Nutzer zur Verfügung gestellt werden. Um diesen Service zu ermöglichen, müssen auf der Client-Seite entsprechende Interaktions- und Darstellungsmöglichkeiten angeboten werden, die auf der Server-Seite mit einer persistenten Datenspeicherung unterstützt werden müssen.

Auf der App-Seite werden beim Aufrufen der Startseite alle Check-Ins des eingeloggten Benutzers über die REST-Schnittstelle angefordert. Dafür sendet die App über den `Networkservice` (siehe Kapitel 6.3.3) einen Request mit dem Parameter *userId*. Diese ID ist eine eindeutige Identifikation des eingeloggten Nutzers, über die die Server-Seite die entsprechenden Check-Ins für diesen Nutzer finden und als JSON-Object zurück an die App senden kann. Auf dieser wird dann das entsprechende Objekt geparst und persistent auf dem Endgerät des Nutzers gespeichert.

Check-In Liste - Request/Response. REST-Path: /checkins

Request:

userId - ID des Nutzers, für den alle Check-Ins angefordert werden sollen.

Response:

```
{
    "checkInList":[
        {
            "checkInId":1,
            "userId":2,
            "locationId":185,
            "locationName":null,
            "timestamp":"2015-10-19 21:02:36",
            "rating":null,
            "image":null
        }
    ]
}
```

checkInList - JSON-Object, das ein JSON-Array bestehend aus Check-In JSON-Objects enthält.

checkInId - ID des Check-Ins.

userId - ID des Nutzers.

locationId - ID der eingecheckten Location.

locationName - Name der eingecheckten Location.

timestamp - Zeitstempel des Check-Ins.

rating - vom User vergebenes Rating für den Check-In.

image - Pfad zum Bild der Location.

Diese gespeicherten Check-Ins werden dann in verschiedenen Views verwendet, aber lediglich in zwei wesentlichen View-Komponenten dargestellt: `CheckInFragment` und `CheckInActivity`.

Das `CheckInFagment` hat die Aufgabe, den Nutzer daran zu erinnern, welche zuletzt besuchten Locations er noch nicht bewertet hat. Dies wird in verschiedenen Views der App dargestellt, in denen das Einchecken in eine Location möglich ist, um ihn bei möglichst vielen Gelegenheiten dazu zu motivieren, eine Bewertung abzugeben. Bewertungen spielen in dem System eine große Rolle, da sie maßgeblich für die Qualität der Empfehlungen verantwortlich sind. Die Motivation des Nutzers eine Bewertung abzugeben, soll über die allgegenwärtige Erinnerung innerhalb der Views erhöht werden. Deswegen stellt das Fragment den zeitlich zuletzt durchgeführten, unbewerteten Check-In mit Informationen wie Location-Name, Location-Bild und Zeitstempel dar. Der Nutzer hat die Möglichkeit über dieses Fragment den angezeigten Check-In zu bewerten oder (sofern vorhanden) andere nicht bewertete Check-Ins in der `CheckInActivity` anzeigen zu lassen. Gibt es keine anderen nicht bewerteten Check-Ins, erhält der Nutzer die Möglichkeit, den angezeigten Check-In zu löschen. Diese Funktionalität ist normalerweise nur in der `CheckInActivity` möglich, kann jedoch durch den frei werdenden Platz direkt in das `CheckInFragment` verlagert werden, ohne dass die View geändert werden muss. Die `CheckInActivity` stellt alle durchgeführten Check-Ins des Nutzers dar, unabhängig davon, ob sie bewertet wurden oder nicht. Diese Activity dient dem Nutzer als Übersicht über seine Besuche und stellt damit eine History-Funktionalität dar. Die Informationen werden in einer Liste abgebildet und gleichen denen des `CheckInFragments` (siehe Abbildung 7.1) .

Die Activity bietet ebenfalls die Möglichkeit einen Check-In zu bewerten. Außerdem ist es möglich, über einen Check-In erneut in eine Location einzuchecken. Allgemein wird diese Funktionalität des Eincheckens sämtlichen Listenansichten von Location-Informationen bereitgestellt. Konkret sind dies die Empfehlungsliste auf der Startseite, die Anzeige der Favoriten und die Ansicht der Check-In-Historie. Beim Eichecken in eine Location wird ein Request über den `NetworkService` an die Server-Middleware ausgelöst, der drei Parameter `userId`, `locationId` und `checkInId` übergibt.

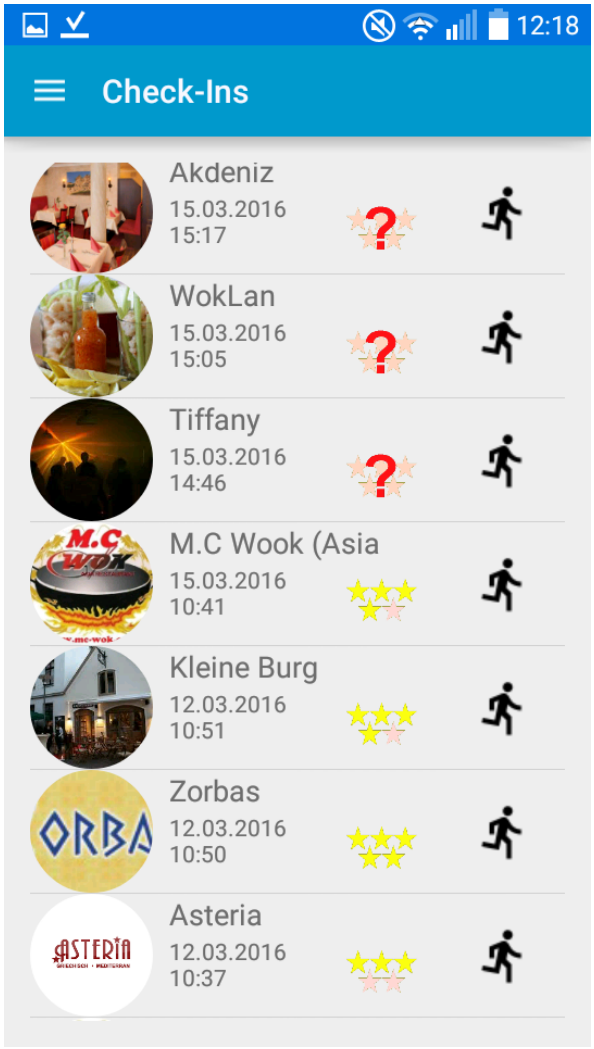


Abbildung 7.1: Checkin-Seite der App. Quelle: Eigene.

Check-In Durchführen - Request/Response. REST-Path: /checkin

Request:

userId - eindeutige Identifikation des derzeit eingeloggten Nutzers.

locationId - eindeutige Identifikation der Location, in der der Nutzer einchecken möchte.

checkInId - auf dem Device eindeutige Identifikation des neuen Check-Ins, die an den Server zur Überprüfung geschickt wird. Durch die Vergabe der ID auf der Client-Seite wird dem Nutzer ein Offline-Betrieb ermöglicht, da nicht auf eine Antwort vom Server gewartet werden muss, sondern mit der internen ID gearbeitet werden kann. Im Gegensatz zum Speichern von z.B. Favoriten ist die checkInId ein wichtiger Identifikator, der für das Bewerten von Check-Ins erforderlich ist, um diese Bewertung richtig speichern zu können.

Response:

```
{
    "status": "ok",
    "checkInId": 17
}

{
    status: "error"
}
```

status - war die Operation erfolgreich: 'ok' = war erfolgreich; 'error' war nicht erfolgreich.

checkInId - die ID, mit der der CheckIn in der DB nach eventueller Konfliktbehandlung gespeichert wird.

Über die Check-In-Liste der `CheckInActivity` ist es außerdem per Swipe-Geste möglich, einen Check-In zu löschen. Dafür wird ein entsprechender Request über den `NetworkService` der App an die Middleware gesendet, der die `checkInId` des zu löschenden Check-Ins enthält.

Check-Out Durchführen - Request/Response. REST-Path: /checkout

Request:

userId - eindeutige Identifikation des derzeit eingeloggten Nutzers.

checkInId - eindeutige Identifizierung des CheckIns (auf dem Device oder auch auf der Server-Datenbank).

Response:

```
{
    "status": "ok"
}

{
    "status": "error"
}
```

status - war die Operation erfolgreich: 'ok' = war erfolgreich; 'error' war nicht erfolgreich.

Da auf der App-Seite ein Offline-Betrieb angeboten wird, wird jede Interaktion des Nutzers mit einem Check-In (einchecken, auschecken, bewerten) gespeichert, unabhängig davon, ob die Operation auf der Server-Seite erfolgreich durchgeführt werden konnte. Um die Konsistenz der Daten sicherzustellen, wurde daher eine Synchronisationsstratgie entwickelt und implementiert. Diese geht fast ausschließlich von der App-Seite aus und funktioniert wie folgt:

- Es existieren vier verschiedene Zustände, die den Status eines Check-Ins beschreiben:
  1. *is_sync*: Check-In ist auf Client- und Serverseite identisch.
  2. *is_not_sync*: Check-In ist clientseitig verändert worden und noch nicht auf dem Server gespeichert worden.
  3. *is_sync_rated*: Check-In, der bereits auf dem Server vorhanden war, wurde bewertet. Dieser Bewertungsvorgang wurde noch nicht synchronisiert.
  4. *is_not_sync_rated*: Check-In, der noch nicht auf dem Server vorhanden war, wurde bewertet. Sowohl die Erstellung des Datensatzes, als auch der Bewertungsvorgang müssen synchronisiert werden.
  5. *is_deleted*: Check-In wurde auf dem Client gelöscht, aber Datensatz ist noch auf dem Server vorhanden.
- Jedes Check-In-Objekt wird je nach Interaktion und Antwort vom Server mit einem entsprechenden Status versehen.
- Beim Anfordern einer Check-In-Liste vom Server wird überprüft, ob es Check-Ins gibt, die noch nicht synchronisiert wurden (Jeden Status außer *is_sync*). Ist dies der Fall, wird versucht die entsprechenden Aktionen nachzuholen und dafür die benötigten Requests an die Middleware erneut zu senden.

Die Server-Seite nimmt die Requests der App über die REST-Schnittstelle entgegen. Es gibt jeweils einen REST-Controller für jeden einzelnen Service: Einchecken, Auschecken und Bewerten. Die Controller geben die benötigten Parameter an die Services weiter, die wiederum über das Hibernate-Framework auf das DBMS zugreifen und die Check-In-Informationen speichern, ändern oder löschen. Beim Einchecken gibt es ein abweichendes Vorgehen: Da das Endgerät eine `checkInId` vorgibt, muss diese noch auf der Server-Seite auf ihre Gültigkeit überprüft werden. In dem speziellen Fall, dass zwei verschiedene Devices das selbe Nutzerkonto verwenden, muss die Eindeutigkeit der `checkInId` gewährleistet sein. Deswegen wird vor dem Einfügen des Datensatzes überprüft, ob die jeweilige ID gültig ist. Ist dies nicht der Fall, wird die nächste gültige ID genommen und diese Information zurück an die App geschickt, die ihre ursprünglich vergebene ID mit der neuen ID überschreibt. Bei allen anderen Operationen wird an den Client intern zurückgemeldet, ob die Aktion erfolgreich *ok* oder nicht erfolgreich *error* war. Auf der App werden diese Status nicht angezeigt, daher ist es für den Nutzer visuell unerheblich, ob eine Operation synchronisiert werden konnte oder nicht.

7.2 Bewertung von Locations und Besuchen

Dieser Abschnitt beschreibt alle Funktionen und Komponenten die zur Umsetzung der Anforderung AN-F-01 und deren untergeordneten Anforderungen dienen.

Der Nutzer kann auf der App die Locationbesuche bewerten, die er zuvor durch einen Check-In markiert hat. Um zu der Bewertungsview zu gelangen, muss der Nutzer auf das Bild mit den nicht bewerteten Sternen drücken. Dadurch wird die `RatingActivity` gestartet. Die View (siehe Abbildung 7.2) enthält Informationen über die Location (Bild, Name) sowie eine `RatingBar` für die Gesamtbewertung und je eine für die Detailbewertungen. Der Nutzer kann mit der `RatingBar` die Anzahl an Sternen auswählen, mit denen er den Besuch bewerten möchte. Zusätzlich kann er

die festgelegten Details *Lautstärke*, *Musik*, *Auslastung* und *Service* analog zur Gesamtbewertung des Besuchs abgeben und einen Kommentar verfassen.

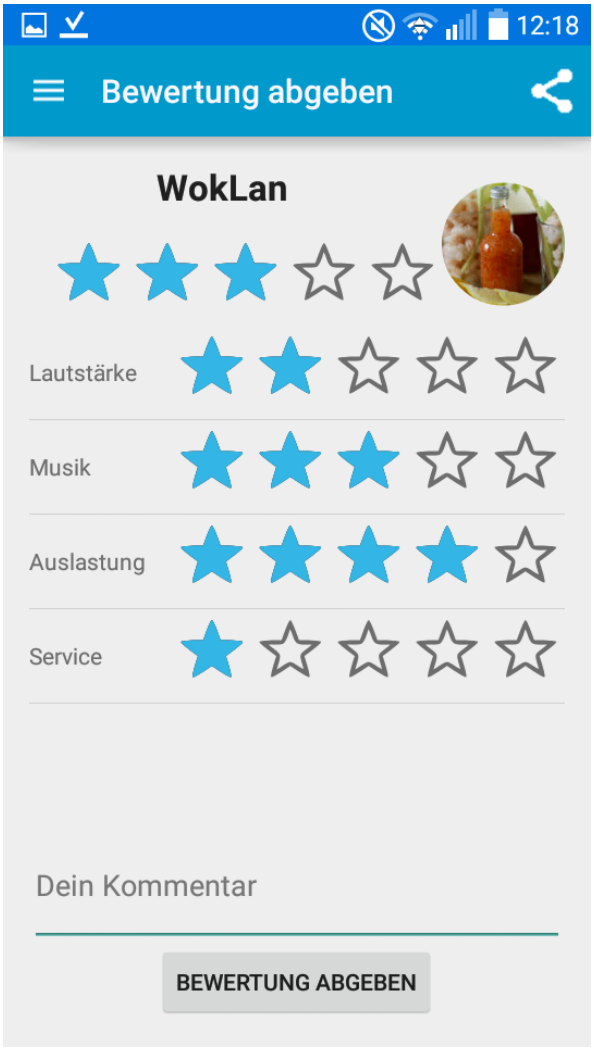


Abbildung 7.2: Rating-Seite der App. Quelle: Eigene.

Bestätigt der Nutzer seine Eingaben durch das Betätigen des Buttons, wird der `NetworkService` aufgerufen, um die Bewertungen und den Kommentar an die Middleware zu senden. Vorher wird die gesamte Bewertung in einer Liste gespeichert, wobei das erste Listenelement die Gesamtbewertung und die folgenden Elemente die Detailbewertung angeben. Der Nutzer muss eine Gesamtbewertung abgeben, kann sich aber entscheiden, Details nicht zu bewerten. Die Bewertungen für die nicht ausgefüllten Details werden dann mit dem Gesamtrating gleichgesetzt, da der Nutzer das Detail offenbar als nicht wichtig genug einstuft, um eine Abweichung zur Gesamtbewertung herauszustellen.

In Abbildung 7.3 ist der Verlauf einer Bewertung auf der Middleware zu sehen.

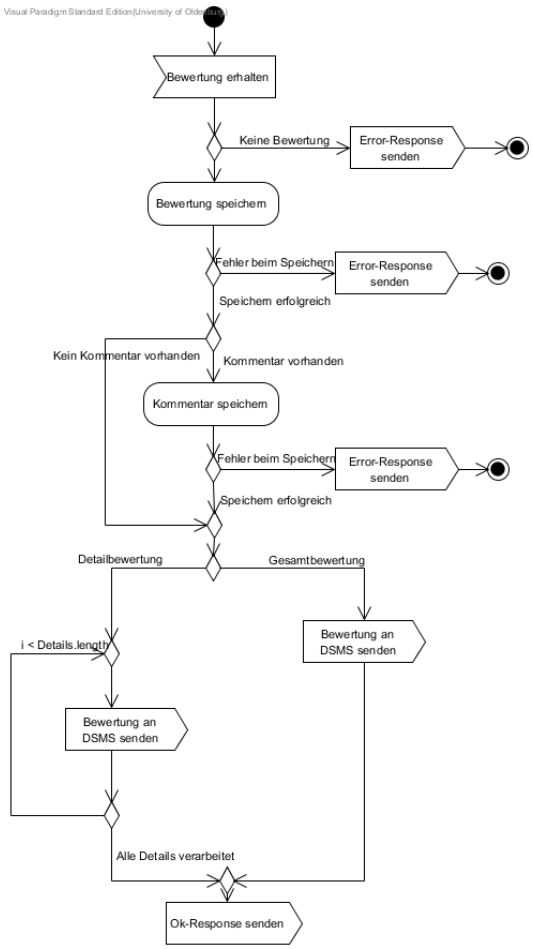


Abbildung 7.3: Verlauf einer Bewertung als Aktivitätsdiagramm. Quelle: Eigene.

Zum Durchführen von Bewertungen bietet die Middleware einen entsprechenden Service über die Representational State Transfer (REST)-Schnittstelle an. Der REST-Service hat folgende Parameter:

Detailrating - Request/Response. REST-Path: /detailrate

Request:

userId - die Id des Benutzers, der die Bewertung durchführen möchte.

checkInId - die Id des Besuchs, der bewertet werden soll.

rating - dieser Parameter ist eine Liste, die an erster Stelle die Gesamtbewertung enthält, auf der die Detailbewertungen der einzelnen Kriterien folgen.

comment (optional) - optional kann der Bewertung ein Kommentar hinzugefügt werden, der auf der Detailseite der Location angezeigt wird.

Response:

```
{
    status: "ok"
}

{
    status: "error"
}
```

status - 'ok': Bewertung erfolgreich; 'error': Bewertung nicht erfolgreich

An diese Schnittstelle können sowohl Gesamtbewertungen, als auch Detailbewertungen gesendet werden. Die Schnittstelle wird in der Klasse `RatingController` realisiert, wie sie in Abbildung 7.4 zu sehen ist. Der `RatingController` ruft den `RatingService` auf und übergibt diesem die von der App gesendeten Bewertungen sowie einen eventuellen Kommentar.

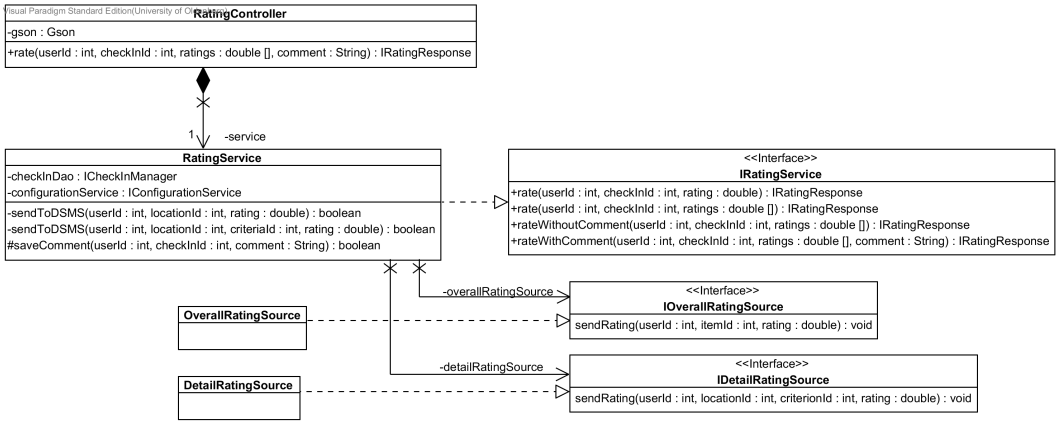


Abbildung 7.4: An der Durchführung einer Bewertung beteiligte Klassen in der Middleware als Klassendiagramm. Quelle: Eigene.

7.2.1 Gesamtbewertung

Im Folgenden wird die Implementierung der Gesamtbewertung im Middleware- und Odysseussystem näher erläutert.

7.2.1.1 Implementierung auf der Middleware

Dem `RatingService` (s. Abbildung 7.4) werden die `userId`, `checkInId`, falls vorhanden der Kommentar `comment` und die Bewertungen `ratings` übergeben. Der Service ermittelt zunächst die aktuelle Konfiguration der Middleware über den `ConfigurationService`. Wenn als Einstellung `Both` oder `Overall` ermittelt wird, übergibt der `RatingService` die Gesamtbewertung einerseits an die DSMS-Schnittstelle `OverallRatingSource`, um die Bewertung an das DSMS zu senden und andererseits an den `CheckInManager`, um die Bewertung und gegebenenfalls zusätzlich den Kommentar in der Datenbank zu speichern. Bei der Einstellung `Both` wird zusätzlich eine Detailbewertung durchgeführt (siehe Abschnitt 7.2.2). Falls es zu einem Fehler kommt, gibt der `RatingService`

einen *Error-Response* zurück an die REST-Schnittstelle, welche den *Response* an die App übermittelt. Im erfolgreichen Fall bekommt die App einen *Ok-Response*, die Gesamtbewertung wird gespeichert und an das DSMS gesendet. Diese Funktionalität dient der Erfüllung der Anforderung AN-F-01.02.

7.2.1.2 Gesamtbewertung-Queries auf Odysseus-Server

Von der Middleware erhält der Odysseus-Server im Falle einer Bewertung die Benutzer-Id(u), die Location-Id(i) und die Bewertung(r). Die für die Bewertung verwendeten Operatoren sind in Abbildung 7.5 zu sehen. Sowohl die Bewertung als auch die Empfehlungsgebung basiert auf der Arbeit von Cornelius Ludmann [LGA15]. Im Rahmen dieser Arbeit wurden verschiedene Operatoren, die die Aufgaben eines RecSys übernehmen, implementiert. Als erstes wird die Bewertung vom `ExtractTestData`-Operator verarbeitet. Dieser Operator hat die Aufgabe, die eintreffenden Bewertungen in Test- und Lerndaten zu sortieren. Hierfür können verschiedene Strategien verwendet werden. Im System werden die Bewertungen mit der Interleaved Test then Train (ITTT)-Strategie in Test- und Lerndaten aufgeteilt. Dabei werden alle Bewertungen sowohl als Test- als auch als Lerndaten verwendet. Bei einer höheren Auslastung des Datenstroms ist es denkbar, die Menge der Testdaten zu reduzieren, da es nicht nötig ist, minütlich oder sekündlich die Empfehlungen auf ihre Qualität hin zu überprüfen. Auf Port 0 gibt der Operator die Lerndaten aus und auf Port 1 die Testdaten. Die Testdaten werden genutzt, um den RMSE zu berechnen. Darauf wird in Abschnitt 7.10.4 genauer eingegangen. Die Lerndaten werden wiederum vom `TrainRecSysModel`-Operator verarbeitet.

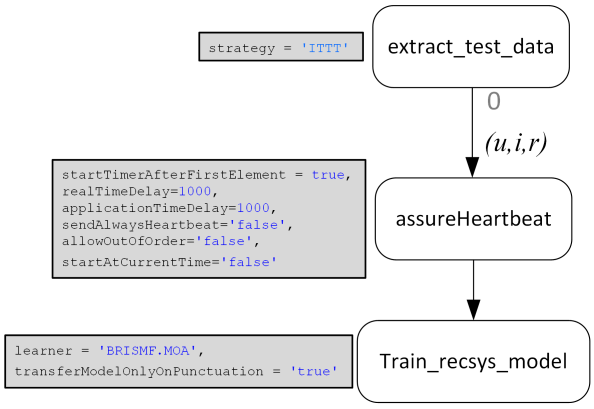


Abbildung 7.5: Übersicht über die Operatoren, die genutzt werden, um eine Gesamtbewertungsrating zu verarbeiten. Quelle: Eigene.

Der `TrainRecSysModel`-Operator erstellt auf Basis einer Matrix-Faktorisierung ein Modell, anhand dessen der `PredictRating`-Operator im späteren Verlauf Vorhersagen für die Nutzer treffen kann. Als Lernalgorithmus wird der BRISMF-Algorithmus verwendet (vgl. [TPNT09]). Um eine geringe Auslastung des Datenstroms auszugleichen, sendet der `AssureHeartbeat`-Operator jede Sekunde einen `Heartbeat`. Zusätzlich sendet der `TrainRecSysModel`-Operator nur bei jedem eintreffenden `Heartbeat` das Modell und nicht bei jeder eintreffenden Bewertung. Das gesendet Modell hat als Startzeitstempel den Endzeitstempel des vorangegangen Modells und als Endzeitstempel die aktuelle Zeit. So befindet sich auf dem Datenstrom immer nur ein gültiges Modell, das für eine Sekunde gültig ist. Falls mehrere Modelle gleichzeitig gültig sind, führt es später in der Empfehlungsgebung zu einer mehrfachen Ausgabe von Empfehlungen. Parallel zum Modell sendet der

Operator auf Port 1 ein Objekt durch, sodass man für jeden Benutzer ermitteln kann, ob ein bestimmtes Item bereits bewertet wurde und auf Port 2 wird nur der `Heartbeat` gesendet. Das dient dazu, den Datenstrom für die Empfehlungsgebung zeitlich synchron mit dem Bewertungsdatenstrom zu halten.

7.2.2 Detailbewertung

Im Folgenden wird genauer auf die Implementierung der Detailbewertung im Middleware- und Odysseussystem eingegangen. Eine Detailbewertung bedeutet in diesem Fall das Bewerten einzelner Aspekte eines Items (z.B. den Service eines Restaurants).

7.2.2.1 Implementierung auf der Middleware

Bei einer Detailbewertung ist der Ablauf im `RatingService` zum größten Teil analog zu dem im Abschnitt Gesamtbewertung (siehe 7.2.1.1) beschriebenen Ablauf. Wenn die Einstellung des `ConfigurationService` auf `Both` oder `Detail` gesetzt ist, wird eine Detailbewertung durchgeführt. Hierbei wird, wie auch bei der Gesamtbewertung, über den `RatingService` die Gesamtbewertung und – falls vorhanden – der Kommentar in der Datenbank gespeichert. Die einzelnen Kriterien der Bewertung dagegen werden vom `RatingService` über die `DetailRatingSource` an das DSMS gesendet. Dabei enthalten die einzelnen Bewertungen neben der `userId`, der `locationId` und dem `rating` noch die `criteriaId`. Sie bestimmt, welchem Kriterium die Bewertung zuzuordnen ist. Auch bei der Detailbewertung führen eventuelle Fehler zur Rückgabe einer *Error-Response*. Diese Funktionalität dient der Erfüllung der Anforderung AN-F-01.03.

7.2.2.2 Modellbeschreibung

Die grundlegende Idee wird aus [AK07] entnommen und wird als *aggregation-function-based-approach* bezeichnet. In diesem Ansatz wird dem Nutzer nicht nur eine detailliertere Möglichkeit geboten, die verschiedenen Aspekte eines Items (hier: Location) zu bewerten, sondern gleichzeitig die Nutzerpräferenzen bezüglich dieser Aspekte zu betrachten. Das bedeutet, dass ein Predicted Rating nicht nur ein durchschnittliches Rating aus den verschiedenen Aspekten eines Items ist, sondern mithilfe einer Aggregationsfunktion die Beziehungen des Nutzers zu einem Aspekt miteinbezogen werden. Es werden die Gesamtbewertung eines Items und die Bewertungen einzelner Aspekte in Zusammenhang gebracht. Über diesen Zusammenhang können Nutzerpräferenzen abgeleitet werden. (vgl. [AK07])

Diese Zusammenhänge zwischen den Aspekten und der gesamten Bewertung eines Items können genutzt werden, um die Qualität der Predicted Ratings noch mehr auf den Nutzer zuzuschneiden. Beispielsweise kann für einen Nutzer der Service ausschlaggebend für eine gute Bewertung einer Location sein, sodass andere Aspekte bei der Erstellung des Predicted Ratings für diesen Nutzer als weniger wichtig betrachtet werden können. Nur Locations, für die eine hohe Service-Bewertung ermittelt wurde, wären in diesem Fall interessant für diesen Nutzer.

Der daraus folgende aggregation-function-based-approach sieht drei Schritte vor, um dieses Verfahren umzusetzen (siehe Abbildung 7.6).

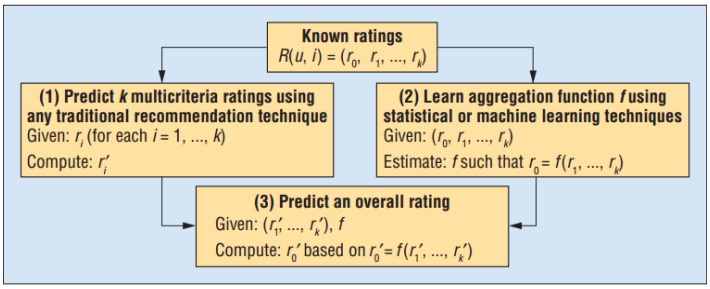


Abbildung 7.6: Überblick aggregation-function-based-approach. Quelle: [AK07].

Schritt 1: Predicted Ratings für Details erstellen (Predict multicriteria ratings) - ein Detailbewertungs-Problem kann als ein multidimensionales Recommendation-Problem angesehen werden. In diesem Verfahren wird jedoch eine Detailbewertung in mehrere Bewertungen pro Kriterium unterteilt und als Teil-Recommendation-Problem betrachtet. Für jedes Kriterium wird ein eigenes Modell erstellt. Bei k Kriterien werden k Recommendation-Probleme erstellt: $R : UserxItems \rightarrow R_i \forall i \in \{1, \ldots, k\}$. Dieser Ansatz bietet viel Flexibilität, da es bereits bestehende Recommender-Techniken verwenden kann und einfach neue Details (Aspekte) eines Items hinzugefügt werden können. (vgl. [AK07], S. 53)

Schritt 2: Nutzerpräferenzen lernen (Learn the aggregation function) - das System ist nach Schritt 1 in der Lage, für jedes Detail ein Predicted Rating zu erstellen. Im nächsten Schritt müssen die Nutzerpräferenzen ermittelt werden. Dafür wird eine Aggregationsfunktion aufgestellt, welche die Predicted Ratings der Details verschieden gewichtet und dann ein gesamtes Predicted Rating ermittelt. Um diese Funktion zu ermitteln können, müssen verschiedene Verfahren wie Domänen-Expertise (z.B. vergangene Erfahrungen, Domänen-Wissen oder Expertenbefragung), statische Techniken oder Verfahren aus dem Machinen Learing verwendet werden [AK07]. Diese Funktion muss das gesamte Predicted Rating dahingehend verändern, dass Nutzerpräferenzen bezüglich verschiedener Details mit einbezogen werden können.

Schritt 3: Berechnen der Predicted Ratings (Predict overall ratings) - nach Schritt 1 und Schritt 2 ist es möglich, über die Aggregationsfunktion und den Predicted Ratings der einzelnen Detailaspekte eines Items ein Gesamtrating für dieses Item zu bestimmen.

7.2.2.3 Detailbewertung-Queries auf Odysseus-Server

Technisch gesehen hat der eingehende Datenstrom in Odysseus die Struktur (u, i, c_i, r_i), wobei c_i die ID des i.-Kriteriums und r_i die Detailbewertung des i.-Kriteriums darstellen. Der Datenstrom wird in Odysseus aufgeteilt in k Datenströme der Struktur (u, i, r).

Der Operatoren-Plan in Odysseus ist in Abbildung 7.7 dargestellt. Die Aufteilung in k Datenströme wird mittels des `ROUTE`-Operators durchgeführt. Der Operatoren-Plan berücksichtigt vier Detailkriterien. Die jeweils folgenden `PROJECT`-Operatoren reduzieren den eingehenden Datenstrom auf das Tripel (u, i, r), wie es jeweils der `TRAIN_RECSYS_MODEL`-Operator erwartet.

Die folgende Operatorenfolge ähnelt der der Gesamtbewertung bzw. eines einzelnen Recommendation-Problems, welches in Abschnitt 7.2.1.2 beschrieben wird.

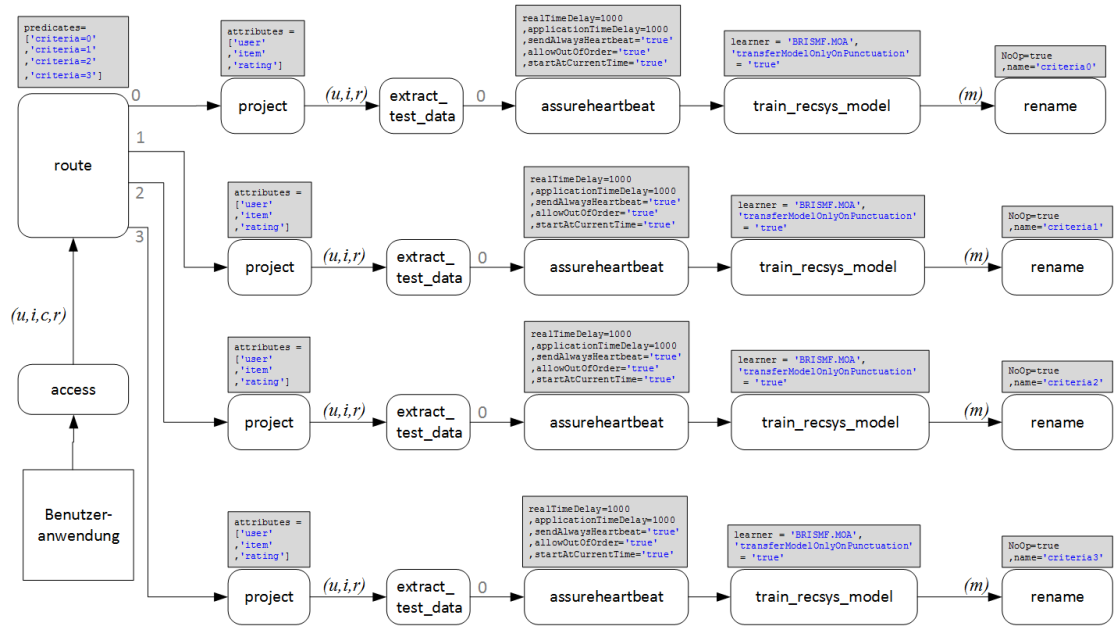


Abbildung 7.7: Operatoren-Plan für die Bewertung eines Besuches mittels mehrere Kriterien (Detailbewertung). Quelle: Eigene.

Die verwendete Aggregationsfunktion aus Schritt 2 der Modellbeschreibung ist derzeit nur ein einfacher Durchschnitt aus allen Detail-Predicted-Ratings des Nutzers. Der Entscheidung für diese Funktion liegen in erster Linie zeitliche Gründe zugrunde, da sie vergleichsweise geringe Komplexität beherbergt. Um wirklich auf die Nutzerpräferenzen eingehen zu können (wie es im aggregation-function-based-approach vorgesehen ist), müsste diese Funktion erweitert werden.

7.3 Empfehlungsgebung

Dieser Abschnitt beschreibt alle Funktionen und Komponenten, die zur Umsetzung der Anforderung AN-F-02 und deren untergeordneten Anforderungen (außer AN-F-02.04) dienen.

7.3.1 Empfehlungsgebung in der App

Für die Anforderung AN-F-02 werden Empfehlungen für einen bestimmten Nutzer über einen Http-Request an die Middleware gesendet und eine sinnvolle Darstellung der erhaltenen Empfehlungen implementiert. Empfehlungen werden in der App hauptsächlich in Listenform dargestellt. Sie enthalten den Namen der Location, ein Bild und das vorhergesagte Rating des Nutzers. Die Liste ist absteigend sortiert und ermöglicht dem Nutzer über Verlinkungen, weitere Informationen über die Location anzeigen zu lassen. Zudem kann der Nutzer in diese empfohlene Location einchecken, um einen späteren Besuch bewerten zu können (siehe Abschnitt 7.1). Dafür wird, wie für alle Listen, das `SuperListFragment` verwendet (siehe Abschnitt 6.3.3). Neben der Listendarstellung gibt es auch eine Darstellung der Locations in einer Kartenansicht (siehe Kapitel 7.4). Empfehlungen werden hauptsächlich auf der Startseite, der `HomeActivity` (siehe Abbildung 7.8) dargestellt.

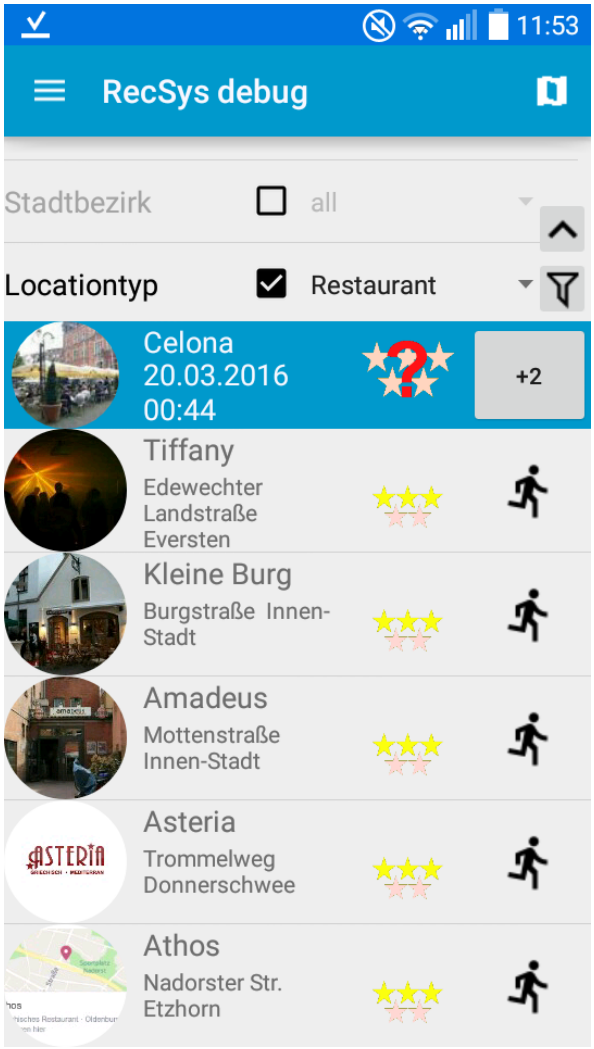


Abbildung 7.8: Startseite mit Empfehlungen. Quelle: Eigene.

In dieser Activity wird der Großteil der Interaktionen mit Empfehlungen behandelt. Deshalb wird an dieser Stelle auch die Behandlung der von der Middleware erhaltenen Filter (siehe Abschnitt 7.3.2) implementiert. Die Filter werden beim Login des Nutzers von der Middleware mit angefordert und dann zusätzlich zu den Nutzerinformationen auf der App persistent gespeichert. Filter werden ebenfalls in einer Liste dargestellt. Dafür wird erneut das `SuperListFragment` verwendet. In der `HomeActivity` werden zunächst die beiden ersten Filter der Filterliste über der Liste mit den Recommendations angezeigt. Die anderen können über eine Scroll-Aktion aufgeklappt und angezeigt werden. Es ist außerdem möglich, sich alle Filter über den entsprechenden Eintrag im Menü in einer Liste in der `FilterActivity` anzeigen zu lassen. Außerdem wird das vorhergesagte Rating bei den Favoriten in der `FavoriteActivity` und bei der Location-Suche in der `SearchActivity` angezeigt.

Der Request an die Middleware verwendet die `userId` des Nutzers. Zusätzlich dazu werden die Filter-Optionen, die der Nutzer mitgeschickt hat, als JSON-Array mitgesendet. Diese werden bei der Ermittlung der Empfehlungen miteinbezogen. Die Suchfunktion der App benutzt ebenfalls den

normalen Empfehlungs-Request, verwendet dabei allerdings einen speziellen Filter, der ausschließlich für die Suche bestimmt ist.

Die empfangenen Empfehlungen liegen im JSON-Format vor und werden von der App analysiert und dann zeitweise persistent gespeichert. Die Empfehlungen werden derzeit nach sechs Stunden neu angefordert, sollte der Nutzer nicht zwischenzeitlich manuell neue Empfehlungen angefordert haben. Die Empfehlungen enthalten ebenfalls Informationen, die zum Filtern auf der Middleware verwendet werden. Diese benutzt der `PostFilterService` im Falle einer nicht gelungenen Interaktion mit dem Server, um auch in diesem Fall zumindest die auf der App gespeicherten Empfehlungen der Anfrage entsprechend anzeigen zu können.

7.3.2 Empfehlungsgebung in der Middleware

Der REST-Controller `RecommendationController` verarbeitet den folgenden Request von der App:

Empfehlungen anfordern - Request/Response. REST-Path: /getRecommendation

Request:

```
{
"userId":123,
"filterList": [
    {
      "id":22,
      "values":[
         "29"
      ]
    },
    {
      "id":123,
      "values":[
         "wok"
      ]
    }
]

}
```

userId - eindeutige Identifikation des eingeloggten Nutzers.

filterList - JSON-Array mit den ausgewählten Filter-Optionen des Nutzers.

id - ID des Filters

values - Liste von ausgewählten Values des Nutzers

Response:

```
{
  "recommendationList": [
    {
      "id": 114,
      "name": "Gaststätte Zur Krone",
      "rating": 4.199999809265137,
      "moreInformation": "Holler Landstraße, Neuenwege",
      "image": "zurkrone.jpg",
      "latitude": 53.1340964,
      "longitude": 8.3038082,
      "attributeList": [
        {
          "id": 1,
          "name": "Locationtyp",
          "type": "list",
          "values": [
            {
              "id": 1,
              "name": "Restaurant"
            }
          ]
        },
        {
          "id": 6,
          "name": "Raucherbereich",
          "type": "boolean",
          "values": [
            {
              "id": 8,
              "name": "false"
            }
          ]
        }
      ]
    }
  ]
}
```


recommendationList - Liste von Empfehlungen

id - ID der Location

name - Name der Location

rating - Predicted Rating für die Location

moreInformation - weitere Informationen zu der Location (derzeit Adresse)

image - Bildpfad der Location

latitude - Breitengrad der Location

longitude - Längengrad der Location

attributeList - (Filter-) Attribute einer Location, zum Post-Filtern auf der App

id - ID des Filters

name - Name des Filters

type - Typ des Filters (derzeit unterstützt: Boolean, List, Search)

values - eingetragene Werte der Location für diesen Filter

id - ID des FilterValues

name - Name des FilterValues

Das Klassendiagramm 7.9 zeigt die Zusammenhänge zwischen denen an der Empfehlungsgebung beteiligten Klassen in der Middleware auf. Die Hauptlogik für das Pre-Filtering befindet sich dabei im `RecommendationService`, `LocationService` und `LocationManager`. Der `RecommendationController` wird für die Kommunikation mit der App über die REST-Schnittstelle genutzt.

Sind Filter gesetzt, werden diese im `RecommendationService` zunächst nach sogenannten Standardfiltern und Filtern, die eine Sonderbehandlung erfordern, getrennt. Standardfilter können über die `FilterValues` und `FilterAttribute` abgebildet werden. Beispiele hierfür sind boolesche oder listenorientierte Filter. Filter die eine Sonderbehandlung erfordern, werden anders ausgewertet, zum Beispiel indem direkt auf Attribute der Locations gefiltert wird. Dies ist etwa der Fall, wenn anhand des Ortsnamens, des Straßennamens, der Postleitzahl oder des Namens der Locations gefiltert werden soll. Außerdem zählt der Filter anhand der Uhrzeit bzw. der Öffnungszeiten der Locations zu den Spezialfiltern, da hier anhand der `BusinessHours`-Relation gefiltert wird.

Die zwei so entstandenen Listen werden dem `LocationService` übergeben, der den `LocationManager` aufruft. Dort werden die übermittelten Filter zu einem Kriterium zusammengestellt und die Datenbankanfrage durchgeführt (siehe Abbildung 7.10).

Für die Standardfilter wird die EXISTS-Bedingung genutzt, die über das *Criterion* in Hibernate zur Verfügung steht und dem *SQL-Exists*-Befehl entspricht. Dieser Ausdruck ist *true*, wenn ein Element existiert, für das die angegebene Bedingung gilt. In diesem Fall ist die Bedingung, dass ein `FilterContainer` existiert, der das jeweilige `FilterValue` enthält. Werden mehrere Filter angegeben, werden diese mit *AND* verknüpft. Für Listenfilter wird für jedes der `FilterValues`, nach denen gefiltert werden soll, ein eigenes *Criterion* erstellt. Diese werden dann mittels *OR* in einem neuen *Criterion* zusammengefasst. Dies wird anschließend dem übergeordneten *Criterion* hinzugefügt.

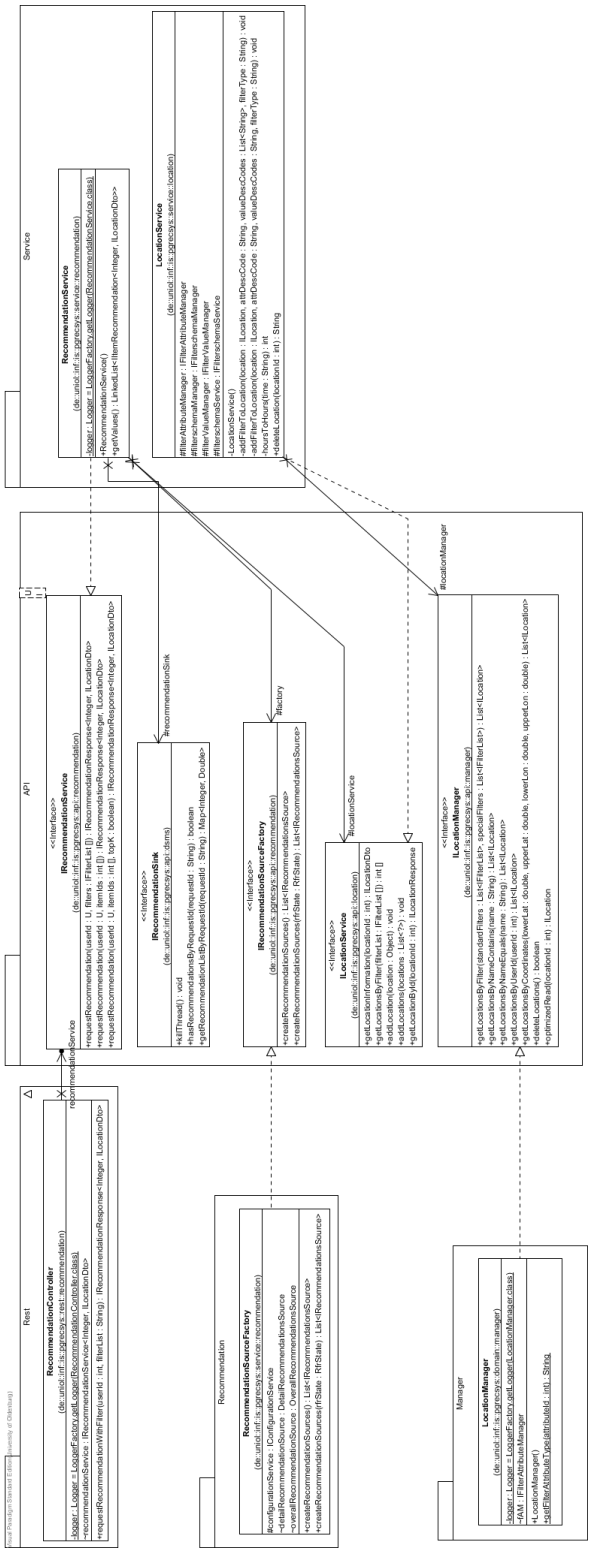


Abbildung 7.9: Klassendiagramm der an der Empfehlungsgebung in der Middleware beteiligten Klassen. Quelle: Eigene.

Die *Criterions* für Filter, die eine Spezialbehandlung benötigen, wie zum Beispiel dem Namensfilter, werden ebenfalls dem übergeordneten *Criterion* hinzugefügt. Durch die Zusammenfassung aller für das Pre-filtering gesetzten Filter in einem *Criterion* sollen mehrfache Anfragen vermieden und so die Performanz gesteigert werden. Die Alternative hierzu wäre, das Pre-filtering für jeden gesetzten Filter separat in einem *Criterion* durchzuführen und anschließend die Schnittmenge aus den Ergebnissen aller *Criterions* zu bilden. Dieses Vorgehen führt jedoch zu eine höhere Menge an Datenbankanfragen.

Dadurch, dass alles zu einem Kriterium zusammengefasst wird, soll die Performanz gesteigert werden, da nicht manuell die Schnittmenge aus mehreren Ergebnismengen gebildet werden muss. Das Resultat der Datenbankabfrage wird an den `RecommendationService` zurückgegeben.

Der `RecommendationService` erhält die Benutzer-Id und eine Liste von Location-Ids, die er an Odysseus weitergeben muss. Hierfür muss als erstes entschieden werden, ob eine Empfehlungsgebung anhand von Gesamt- oder Detailbewertungen durchgeführt werden soll. Das ist Aufgabe der `RecommendationSourceFactory`. Sie ermittelt über den `ConfigurationService` die aktuelle Einstellung der Middleware. Entsprechend der Einstellung erstellt die Factory eine `RecommendationSource`, die die RfRs an den entsprechenden Port von Odysseus sendet. Über die `RecommendationSource` werden nun die Benutzer-Id und die Location-Ids an Odysseus übermittelt. Auf die genaue Funktionsweise der `RecommendationSource` wird in Abschnitt 6.3.2 eingegangen.

Sobald Odysseus eine Empfehlungsmenge mit den dazugehörigen, vorhergesagten Bewertungen zurückgibt, werden dem `LocationService` die einzelnen Location-Ids übergeben. Dieser reichert die Locations mit weiteren Informationen, wie z. B. die Adresse und dem Namen an. Diese Liste von Locations wird zurück an den `RecommendationController` gegeben, der sie an die App sendet.

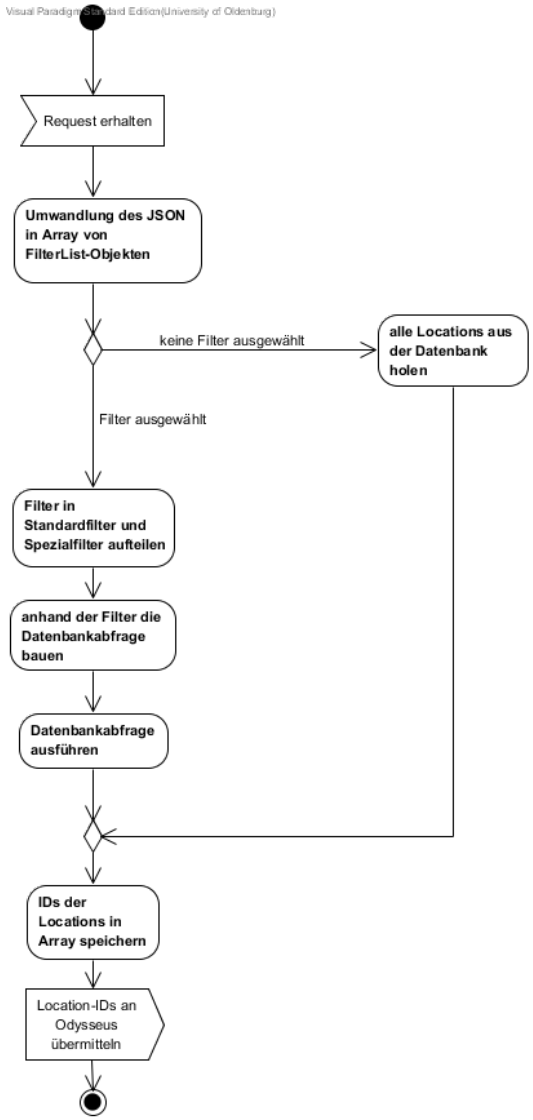


Abbildung 7.10: Aktivitätsdiagramm des Prefilterings in der Middleware. Quelle: Eigene.

7.3.3 Empfehlungsgebung von Odysseus

Über die Middleware erhält, wie im Abschnitt zuvor beschrieben, Odysseus einen Request for Recommendations (RfR). Diese Anfrage hat das Ziel, für einen bestimmten Benutzer eine Menge von Empfehlungen zu geben. Odysseus erhält von der Middleware neben der `userId` eine Liste von `locationId`, auf deren Basis eine Menge von empfohlenen Locations ermittelt werden sollen und den Boolean Parameter isK, der angibt ob nur k Empfehlungen gegeben werden sollen oder alle angefragten Locations. Bei Odysseus können Empfehlungen auf Basis von Gesamtbewertungen oder auf Basis von Detailbewertungen angefragt werden. Hierfür werden verschiedene Sources auf unterschiedlichen Ports angeboten. Im Folgenden wird einerseits die Empfehlungsgebung auf Basis von Gesamtbewertungen und Detailbewertungen getrennt beschrieben.

7.3.3.1 Empfehlungsgebung anhand von Gesamtbewertungen

Falls der RfR an den Port für die Empfehlungsgebung anhand von Gesamtbewertungen gesendet wird, wird der in Abbildung 7.11 angegebene Operatoren-Plan verwendet. Da es sich um eine JSON-Schnittstelle handelt, muss als erstes das JSON-Objekt in ein Tupel umgewandelt werden. Der Grund dafür ist, dass die meisten Operatoren von Odysseus nur mit Tupeln arbeiten können. Der `KeyValueToTuple`-Operator übernimmt diese Aufgabe. Das Tupel besteht nun aus der `userId` (u), der Liste von `locationIds` (L) und dem booleschen Attribut isK. Dieses Attribut gibt an, ob von der angefragten Menge von Locations nur die besten k oder alle zurückgegeben werden sollen. Bei der Initialisierung der Queries liegt k standardmäßig bei 5. k lässt sich über das Dashboard konfigurieren (siehe Abschnitt 7.10.4).

Anschließend folgt der `UnnestRecommendationCandidates`-Operator. Dieser Operator ist im Rahmen der Projektarbeit hinzugefügt worden. Dabei handelt es sich um Abwandlung des `Unnest`-Operators. Der Operator extrahiert die Elemente einer Liste und gibt sie als einzelne Tupel wieder aus. Als Parameter erhält der Operator das Attribut, das die Liste enthält. Die ausgehenden Tupel sind bis auf die Liste, die durch das einzelne Element ersetzt wurde, identisch mit den eingehenden Tupeln. Im Gegensatz zum `Unnest`-Operator sendet der `UnnestRecommendationCandidates`-Operator nach dem letzten Element der Liste eine `TuplePunctuation` mit der `userId`, um das Ende der Liste zu signalisieren.

Die `TuplePunctuation` wird 1000 ms nach dem letzten Element gesendet. Grund dafür sind Operatoren die Elemente sammeln, um die zeitliche Reihenfolge dieser zu gewährleisten. Da `Punctuations` direkt vom Operator weiter gesendet werden, erreicht die `Punctuations` vor den anderen Elementen die folgenden Operatoren. Dadurch wird der Zeitfortschritt der folgenden Operatoren auf den der `Punctuation` gesetzt und die folgenden Elemente werden aussortiert, weil sie außerhalb der zeitlichen Folge sind. Diese Umsetzung ist nicht optimal, daher sollte in Zukunft eine Optimierung stattfinden, da an dieser Stelle ein hoher Performance-Verlust eintritt.

Darauf folgt ein `Merge`-Operator, in den einerseits der beschriebene Datenstrom und andererseits der Datenstrom von Port 2 des `TrainRecSysModel`-Operators aus dem Queryplan für Gesamtbewertungen einfließt. Auf Port 2 sendet, wie bereits in Abschnitt 7.2 beschrieben, der Operator bei jedem Senden eines Modells eine `HeartbeatPunctuation`. Dadurch wird gewährleistet, dass dieser Datenstrom zeitlich synchron mit dem Rating-Datenstrom ist und `Sweepareas` folgender Operatoren regelmäßig geleert werden, falls keine Anfragen eintreffen. Das ist nötig, da sich ansonsten in den `Sweepareas` immer mehr Tupel ansammeln und erst durch einen RfR die `Sweepareas` entleert werden. Daher wird durch `Heartbeat-Punctuations` eine hohe Auslastung simuliert.

Im nächsten Schritt wird mit einem `Join`-Operator ein Kreuzprodukt mit der Ausgabe vom `TrainRecSysModel`-Operator von Port 0 erstellt. Auf diesem Port wird, wie bereits in Abschnitt 7.2 beschrieben, das Modell ausgegeben anhand dessen Bewertungen vorhergesagt werden können. Das Tupel besteht nun aus der `userId`, einer `locationId`, isK und dem Modell (m).

Auf Basis der `userId`, der `locationId` und dem Modell kann nun der `PredictRating`-Operator eine Bewertungsvorhersage für jede Location geben. Auf dieser Menge an Locations und Vorhersagen wird im Anschluss der `TopK`-Operator durchgeführt. Dabei werden die Locations absteigend nach der vorhergesagten Bewertung sortiert und die obersten 10000 weitergegeben. Diese hohe Zahl besteht deshalb, um scheinbar nicht von der Top-k-Grenze betroffenen RfR zu beantworten, aber

dem Benutzer auch keine zu große Menge von Locations zu senden, falls die Filter sie im Vorfeld nicht weit genug eingrenzen.

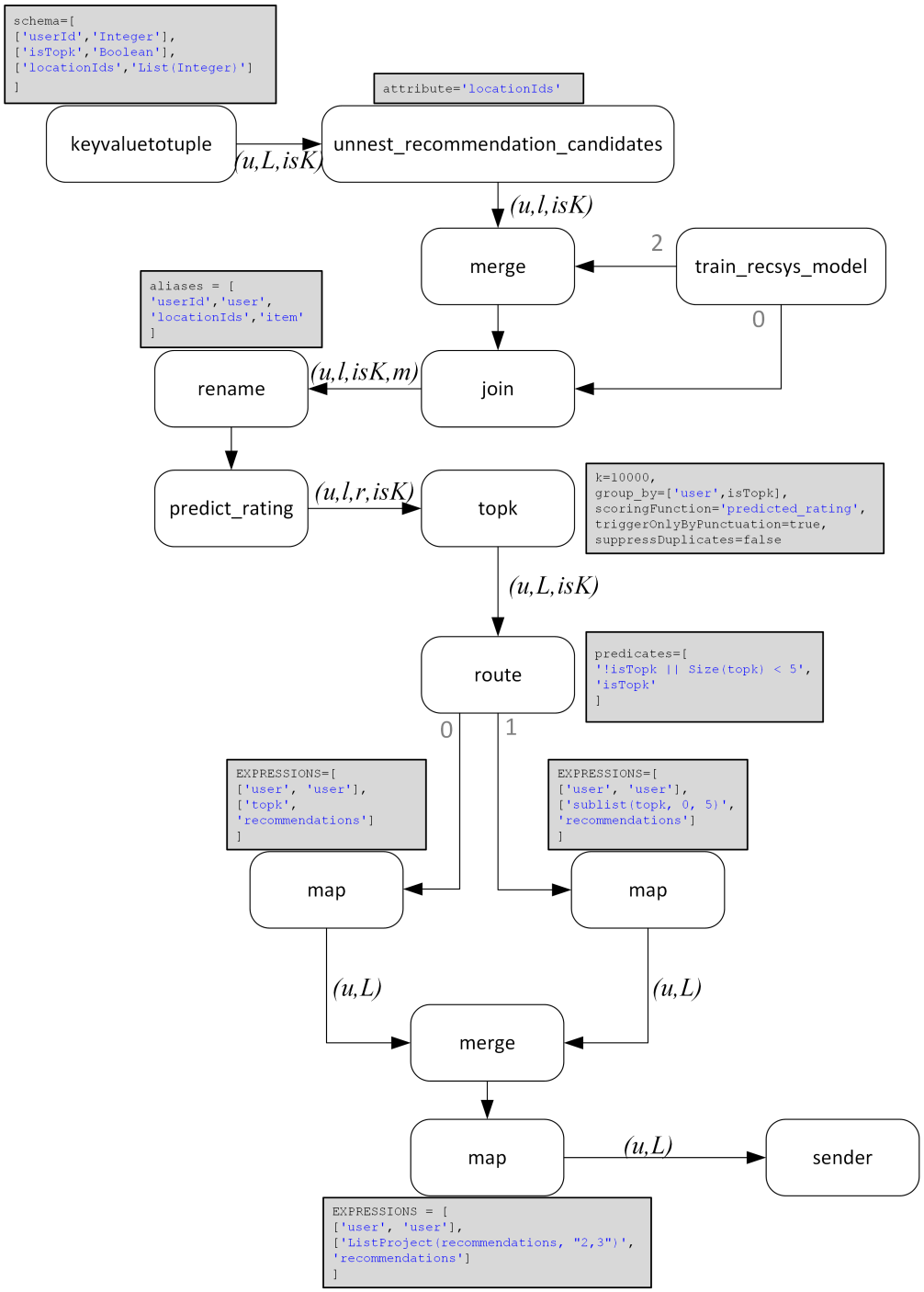


Abbildung 7.11: Operatoren-Plan für die Empfehlungsgebung anhand von Gesamtbewertungen. Quelle: Eigene.

Der TopK-Operator sammelt die Tupel gruppiert nach der userId in Listen. Sobald eine TuplePunctuation mit seiner userId eintrifft, werden die Top-k für diesen Benutzer ausgegeben. Diese Punctuation wird vom UnnestRecommendationCandidate-Operator am Ende der Liste gesendet. Ab diesem Punkt gibt es für jeden RfR wieder nur ein Tupel, das eine Liste von allen Locations samt der vorhergesagten Bewertung enthält.

Im Folgenden wird durch den Route-Operator entschieden, ob alle Elemente zurückgegeben werden oder nur die besten k Elemente. Zusätzlich wird der Sonderfall überprüft, ob die Menge an angefragten Elementen kleiner als k ist. In diesem Fall wird der RfR so behandelt als wenn kein Top-k genutzt werden soll. Das wird anhand des booleschen Wert für isK festgemacht, der bei der Anfrage übergeben wird. Falls die besten k Elemente zurückgegeben werden sollen, wird ein Map-Operator aufgerufen. Dieser Operator führt auf der Liste eine sublist-Operation durch und gibt nur die Elemente bis zum Index k zurück. Im anderen Fall wird auch ein Map-Operator aufgerufen, der jedoch nur dazu dient, ein Rename und Project durchzuführen.

Es wurde ein alternatives Vorgehen getestet, indem zwei TopK-Operatoren mit unterschiedlichem k genutzt werden, statt die Liste über sublist zu verkleinern. Dabei ist es jedoch nötig, den gesamten Prozess der Empfehlungsgebung doppelt im Operatoren-Plan durchzuführen. Ansonsten würde die TuplePunctuation vom Route-Operator auf allen Ports weitergegeben werden und somit jeder TopK-Operator eine Ausgabe generieren. Das würde den Operatoren-Plan unnötig kompliziert gestalten und zudem die Wartbarkeit erschweren. Daher wurde die einfachere Variante gewählt, womit die Liste über sublist verkleinert wird.

Das Ergebnis beider Map-Operatoren sind Tupel, die nur die userId und die Liste der Locations samt der vorhergesagten Bewertungen enthält. Beide Datenströme werden anschließend durch einem Merge-Operator wieder vereint.

Zum Schluss wird ein weiteres Map auf dem Datenstrom ausgeführt, um die Elemente in der Liste von überflüssigen Attributen zu bereinigen, sodass nur die Location und die vorhergesagte Bewertung übrig bleiben.

Diese Tupel werden nun über eine Senke zurück an die Middleware gesendet und dort weiterverarbeitet.

7.3.3.2 Empfehlungsgebung anhand von Detailbewertungen

Falls der RfR an den Port für die Empfehlungsgebung anhand von Detailbewertungen gesendet wird, wird der in Abbildung 7.12 angegebene Queryplan verwendet. Der grundlegende Ablauf ist mit dem einer Empfehlungsgebung anhand von Gesamtbewertungen identisch. Es muss lediglich für jedes Kriterium die erwartete Bewertung vorhergesagt und zu einer Gesamtbewertung aggregiert werden.

Im Operatoren-Plan ist zu sehen, dass nach dem UnnestRecommendationCandidates von vier Merge-Operatoren auf das Ergebnis des Operators zugegriffen wird. Jeder dieser Merge-Operatoren leitet die Vorhersage eines Kriteriums ein, wie es bereits im vorherigen Abschnitt 7.3.3.1 für die Empfehlungsgebung anhand von Gesamtbewertungen erläutert wird.

Nachdem durch jedes Modell eine vorhergesagte Bewertung ermittelt wurde, werden durch mehrere Union-Operatoren hintereinander die Datenströme wieder zusammengeführt. Anschließend wird eine Aggregation, in der der Durchschnitt der nach userId und locationId gruppierten Tupel berechnet wird, durchgeführt.

Ab diesem Punkt befindet sich auf dem Datenstrom für jede angefragte Location nur noch ein Tupel mit der durchschnittlichen vorhergesagten Bewertung aller Modelle. Die folgenden Operatoren und Schritte stimmen wieder mit denen bei einer Empfehlungsgebung anhand von Gesamtbewertungen überein. Es werden die Top-k ermittelt, bei Bedarf wird die Liste auf k Elemente reduziert und die Empfehlungen werden über einen TCP-Socket zurück an die Middleware gegeben.

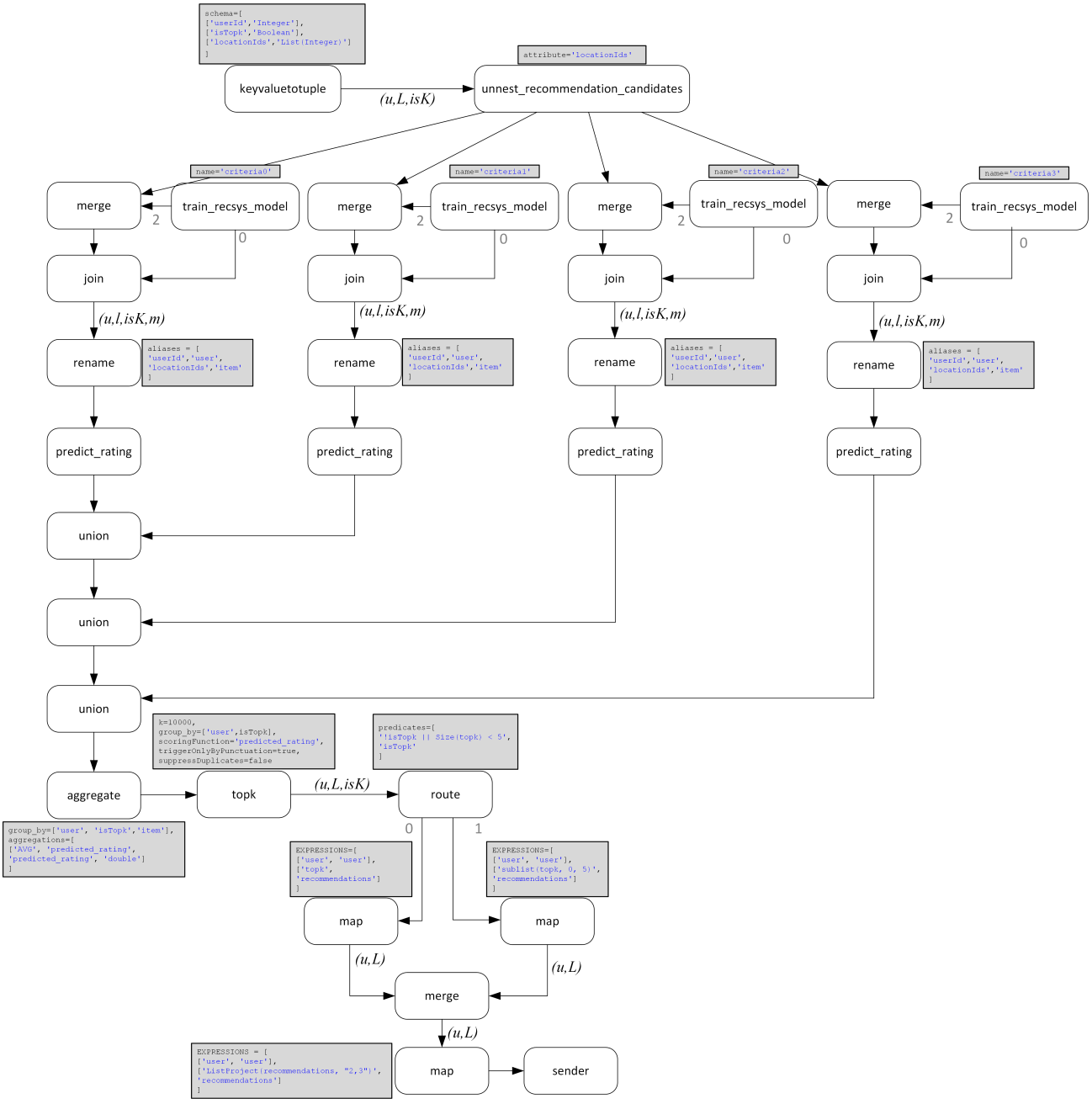


Abbildung 7.12: Operatoren-Plan für die Empfehlungsgebung anhand von Detailbewertungen. Quelle: Eigene.

7.4 Kartenansicht für Locations

Um dem Nutzer eine weitere Art der Darstellung der Empfehlungen zu bieten, als die Listenansicht in der Home-View, hat die Projektgruppe noch eine Kartenansicht für die Empfehlungen implementiert (siehe AN-F-05). Für die Kartenansicht wurde die API von Google Maps verwendet. Der Nutzer kann, indem er in der `ActionBar` der HomeView der App auf das Kartensymbol drückt, zur Kartenansicht wechseln. Es wird lediglich das `ListFragment` ausgetauscht und so die Karte unter dem `CheckInFragment` angezeigt. Da die Koordinaten der Locations schon bei dem Senden der Empfehlungen übermittelt werden, ist hier keine weitere Serverkommunikation notwendig. Auf der Karte werden die Locations dann an ihren Standorten mit *Markern* gekennzeichnet. Drückt der Nutzer auf einen dieser Marker, wird das Listenelement dieser Location mit dem Bild, der Adresse, der vorhergesagten Bewertung und dem Check-In Button angezeigt. Hier sind die Funktionen wieder analog zu der Listenansicht. Sobald die Karte geladen wird, ändert sich das Bild des Buttons in der `ActionBar`. Durch erneute Betätigung dieses Buttons gelangt der Nutzer wieder zur Listenansicht.

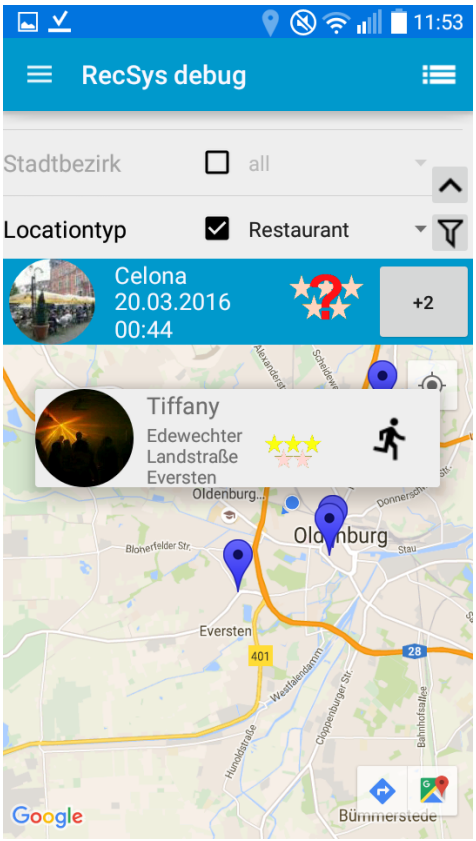


Abbildung 7.13: Kartenansicht für Empfehlungen. Quelle: Eigene.

7.5 Location Suche

Wie in Anforderung AN-F-03 beschrieben, wird eine Suche implementiert, über die ein Nutzer eine oder mehrere Locations über den Location-Namen, eine Straße oder eine Postleitzahl suchen kann.

Grundlegend wird die Suche mithilfe der bereits implementierten `Filter-Recommendation-Schnittstelle` umgesetzt. Die unterschiedlichen Suchmöglichkeiten sind als weitere Filter vom Filtertyp *search* implementiert. Diese werden in das Datenbankschema eingetragen und dann beim Login mit den anderen Filtern an die App geschickt. Ist ein search Filter vorhanden, kann der Nutzer über das Menü die Search-View mit der `SearchActivity` aufrufen.

Über die `SearchActivity` kann der Nutzer einen der drei Suchtypen auswählen und dann über eine Suchphrase suchen. Um eine Location mit *Wok* im Namen zu suchen, muss also der Location-Namens-Filter ausgewählt werden und danach die Suchphrase *Wok* in das Textfeld eingegeben werden. Die Suchphrase wird dann als Filter-Value an die REST-Schnittstelle geschickt. Diese sendet Empfehlungen zurück, die analog zur Startseite in Listen- oder Kartenansicht angezeigt werden können. Diese Empfehlungen werden jedoch zu den Empfehlungen aus der Startseite nicht persistent auf der App gespeichert.

Location suchen und Empfehlung anzeigen lassen - Request/Response. REST-Path: /getRecommendation

Request:

```
{
"userId":123,
"filterList": [
    {
      "id":123,
      "values":[
         "wok"
      ]
    }
]

}
```

userId - eindeutige Identifikation des eingeloggten Nutzers.

filterList - JSON-Array mit den ausgewählten Filter-Optionen des Nutzers.

- **id** - ID des Suchfilters.
- **values** - Suchbegriff.

Response: siehe Response-Abschnitt 7.3.2.

Auf der Server-Seite wird der Search-Filter als ein “Spezial-Filter“ mit spezieller Handhabung behandelt, fügt sich aber in den normalen Ablauf ein. Für die implementierten Such-Filter wird eine Volltext-Abfrage über die Location-Tabelle durchgeführt, die überprüft, ob ein (Teil-)Begriff enthalten ist. Ist das der Fall, wird diese Location im Pre-Filter-Verfahren ausgewählt und an Odysseus weitergeleitet. Jedoch wird bei der Suche der Top-k-Operator nicht beachtet, damit alle möglichen Ergebnisse angezeigt werden können.

7.6 Detailansicht für Locations

Nach Anforderung AN-F-04 sollen dem Nutzer weitere Informationen über eine Location in einer eigenen View angezeigt werden können. Der Nutzer gelangt zur Detailansicht einer Location, indem er im `ListenFragment` der Empfehlungen, der Check-In-Liste oder in der Favoritenansicht auf das Profilbild der Location drückt. Sobald er auf ein Locationbild drückt, wird die `LocationDetailsActivity` der App aufgerufen. In der Activity wird zuerst ein Request über den `Networkservice` (siehe Kapitel 6.3.3) durchgeführt. Der Methode wird die `locationId` der jeweiligen Location, für die detaillierteren Informationen angezeigt werden sollen, übergeben, um dann den RestController der Middleware über das Mapping */locationDetails* aufzurufen.

Detail Informationen zu Locations - Request/Response. REST-Path: /detailrate

Request:

locationId - die Id der Location.

Response:

```
{
    "locationName": "Bleur",
    "adress": "Neue Straße 1",
    "locationImage": ".jpg",
    "phoneNumber": "010120",
    "webadress": "http://....",
    "rating": "4.5",
    "longitude": "12.9332",
    "latitude": "4.9238",
    "businessHours": [
            {
                "dayOfWeek": "1",
                "start1": "12:00",
                "start2": "18:00",
                "end1": "14:00",
                "end2": "23:00"
            },
            {
                "dayOfWeek": "2",
                "start1": "12:00",
                "start2": "18:00",
                "end1": "14:00",
                "end2": "23:00"
            }
    ],
    "comments": [
            {
                "username": "Max",
```

```
                "userimage": ".jpg",
                "text": "this is a comment"
            },
            {
                 "username": "Max",
                "userimage": ".jpg",
                "text": "this is a comment"
            }
    ]



}
```

locationName - Name der Location.

locationImage - Bildpfad der Location.

phoneNumber - Telefonnummer der Location.

webadress - URL der Webseite der Location.

longitude - Breitengrad der Location.

latitude - Längengrad der Location.

businessHours - Öffnungszeiten der Location. Es gibt 7 Öffnungszeiten für jeden Tag.

- **dayOfWeek** - Wochentag als Integer. Beginnend bei 0 = Sonntag bis 6 = Samstag.
- **start1** - Startzeit der ersten Öffnungszeit an diesem Tag.
- **start2** - Startzeit der zweiten Öffnungszeit an diesem Tag.
- **end1** - Endzeit der ersten Öffnungszeit an diesem Tag.
- **end2** - Endzeit der zweiten Öffnungszeit an diesem Tag.

comments -

- **username** - Name des Nutzers, der den Kommentar abgegeben hat.
- **userimage** - Bildpfad des Nutzers, der den Kommentar abgegeben hat.
- **text** - Kommentartext.

Der REST-Controller erwartet für die Verarbeitung der Anfrage lediglich den Parameter `locationId`, mit der er die Details der jeweiligen Location in der Datenbank abrufen kann. Der Controller gibt die `locationId` an den `LocationService` weiter, der die entsprechende Location aus der Datenbank lädt. Dazu wird der `LocationManager` des Domain-Projekts initialisiert und über die `locationId` die Location gelesen, um dann die Attribute Name, Adresse, Pfad zum Bild, Telefonnummer, Webseite, Längengrad, Breitengrad, Öffnungszeiten, Kommentare über die Location und die durchschnittliche Bewertung aus Check-Ins dieser Location dem `LocationResponse-`

Objekt zuzuordnen. Durch diese Detailinformationen werden die Anforderungen AN-F-04.01 und AN-F-04.02 umgesetzt. Der REST-Controller erstellt einen JSON-String aus dem Objekt, sodass alle Attribute der Location in Form von *Key-Value-Paaren* oder Listen gespeichert sind. Dieser JSON-String wird dann als Antwort zurück an die App gesendet. Wenn die Kommunikation erfolgreich ist, wird der JSON-String geparsed und dann eine entsprechende GUI mit den Daten aus dem String erstellt, wodurch der Nutzer eine Übersicht über alle relevanten Informationen der Location dargestellt bekommt. Als Darstellungsvariante hat sich die Projektgruppe dazu entschieden, die einzelnen Attribute wie Homepage, Telefonnummer, Anzeige auf der Karte und Öffnungszeiten in Form von Image-Buttons anzuzeigen, damit die View übersichtlich bleibt und nicht überladen wirkt (siehe Abbildung 7.14).

Durch einen Klick auf den jeweiligen Image-Button werden dem Nutzer dann die jeweiligen Informationen angezeigt. Bei dem Marker für die Karte wird *Google Maps* geöffnet und auf den Standort der Location fokussiert. Der Nutzer kann sich über die von Google bereitgestellte Routenplanung zur Location navigieren lassen. Der Webseiten-Image-Button öffnet in einem Browser, der auf dem Gerät installiert ist, die jeweilige Webseite der Location.

Je nachdem, ob die Location gerade geöffnet hat oder geschlossen ist werden verschiedene Bilder als Image-Button angezeigt, sodass der Nutzer hier direkt eine Auskunft bekommt. Durch einen Klick auf den Button wird dann ein Dialog aufgerufen, der die angegebenen Öffnungszeiten darstellt. Unter dem Profilbild der Location wird das Listen-Fragment angezeigt. In diesem werden die aktuellsten Kommentare zu der Location inklusive Profilbild der Nutzer aufgeführt.

Treten bei dem Vorgang oder der Kommunikation Fehler auf, etwa weil die Middleware nicht erreicht wird oder keine Datenbankverbindung besteht, wird anstelle der Detailinformationen ein Text angezeigt, der mitteilt, dass die Location in dem Moment nicht gefunden werden kann.

Abbildung 7.14: Detailview der App. Quelle: Eigene.

7.7 Locations teilen

Laut Anforderung AN-F-09 soll dem Nutzer eine Funktionalität zur Verfügung gestellt werden, Location-Informationen mit Freunden zu teilen. Diese Funktionalität ist unabhängig davon, ob die andere Person, mit der die Informationen geteilt werden sollen, die Empfehlungs-App benutzt oder nicht.

Dafür wird in der `LocationDetailActivity` und in der `RatingActivity` ein Teilen-Button eingefügt, der sich in der `ActionBar` der Activity befindet. Diese Informationen sind Daten über eine Location (Name, Website, etc.) oder Daten über ein Rating zu einem Locationbesuch (Gesamtbewertung, Kommentar, etc.). Diese werden über den `Send-Intent` [1] von Android gesendet. Diese Funktion sendet einen String an andere Applikationen, die diesen Intent unterstützen.

7.8 Benutzerverwaltung

Im folgenden Abschnitt wird die Implementierung der Benutzerverwaltung genauer erklärt. Dabei bezieht sich das Kapitel hauptsächlich auf Informationen über den Benutzer, die vom Benutzer selbst

[1] `Sending Simple Data to Other Apps: http://developer.android.com/training/sharing/send.html`

in das DBMS eingetragen werden. Es beschäftigt sich nicht mit der administrativen Verwaltung von Nutzerinformationen.

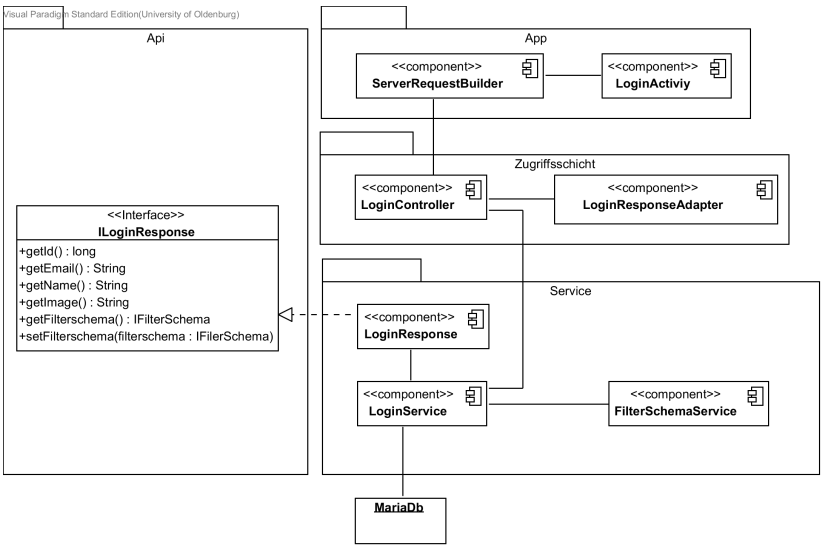


Abbildung 7.15: Sämtliche Komponenten, die in der Benutzerverwaltung involviert sind. Quelle: Eigene.

7.8.1 Registrierung

Die Registrierung eines Nutzers (siehe AN-F-11.01) legt ein neues Nutzerkonto an und füllt es mit den übergebenen Nutzerinformationen. Dieses Konto kann dazu verwendet werden, den Nutzer eindeutig zu identifizieren.

Vom Login-Startbildschirm der App aus erreicht der Nutzer die Registrierungs-Ansicht. Für die Eingabe gibt er seine E-Mail-Adresse, seinen Benutzernamen und sein Passwort ein. Das Passwort muss der Nutzer zweimal eingeben, um die erste Eingabe zu bestätigen. Falls die Passworteingaben nicht übereinstimmen, wird dies über eine entsprechende Nachricht angezeigt. Sollte es sich bei der E-Mail-Adresse nicht um eine realistische Adresse handeln, wird dies ebenfalls über eine Mitteilung ausgegeben. Sind die Eingaben korrekt und klickt der Nutzer auf den Registrieren-Button, erscheint ein Text, der die erfolgreiche Registrierung mitteilt. Die Eingaben werden anschließend über den `NetworkService` (siehe 6.3.3) an die Middleware geschickt. Dieser empfängt nach Bearbeitung durch die Middleware auch die Response. Sollte die Registrierung fehlgeschlagen sein, z.B. weil die eingegebene E-Mail-Adresse bereits vergeben ist, erreicht die App eine Fehlermeldung.

Middlewareseitig verarbeitet der `RegisterController` den Request mit den Requestparametern `email`, `name` und `pass` und ruft den `RegisterService` auf. Die Eingabe `email` wird mit der Datenbank abgeglichen und geprüft, ob sie bereits von einem anderen Nutzer verwendet wird. Sind die Daten korrekt, wird der Nutzer neu in der Datenbank angelegt und ein JSON Response-Objekt mit Status 'ok' erstellt. Sollte die Registrierung fehlschlagen, wird ein Response-Objekt mit dem Status 'error' zurückgegeben.

Registrieren - Request/Response. REST-Path: /register

Request:

email - E-Mail-Adresse des Nutzers (Verwendet für Login).

name - Anzeigename des Nutzers (im Profil oder z.B. Kommentaren).

pass - Passwort des Nutzers (verwendet für Login).

Response:

```
{
    "status": "ok"
}
```

```
{
    "status": "error"
}
```

status - Ergebnis der angeforderten Aktion: 'ok': hat funktioniert; 'error': hat nicht funktioniert (E-Mail schon verwendet).

7.8.2 Login

Beim Login (siehe AN-F-11.03) verifiziert der Nutzer durch die Eingabe seiner E-Mail-Adresse und seines Passworts seine Identität. Dadurch kann ein Device klar einem Nutzer zugeordnet werden.

In der App trägt der Nutzer seine Daten in der Anmeldemaske (siehe 6.12 ein. Diese werden anschließend über den appseitigen `NetworkService` an die Middleware geschickt. Dieser empfängt schließlich wieder die Response von der Middleware.

Der `LoginController` verarbeitet den Request mit den Requestparametern `email` und `pass` und ruft den `LoginService` auf. Die Parameter werden mit der Datenbank abgeglichen. Außerdem wird das aktuelle Filterschema von der Datenbank geholt. Sind die Daten korrekt, wird ein Respose-Objekt für diesen Nutzer erstellt. Für das Serialisieren von Objekten des Typs `ILoginResponse` in JSON ist der `LoginResponseAdapter` verantwortlich. Dieser ist notwendig, da auf eine Datenstruktur mit bidirektionalen Beziehungen (Hibernate-Models) zugegriffen wird. Über diesen Adapter wird Gson mitgeteilt, wie diese Beziehungen gehandhabt werden sollen. Der `LoginResponse`-Service holt sich aus der Datenbank für diesen Nutzer die Id, die E-Mail, den Namen und das Profilbild. Sollte der Login fehlschlagen, wird eine leeres JSON Response-Objekt zurückgegeben.

Login - Request/Response. REST-Path: /login

Request:

email - E-Mail-Adresse des Nutzers.

pass - Passwort des Nutzers.

Response:

```
Erfolgreich:
{
    "id": 14,
    "email": "MaxMustermann@mustermail.de",
    "name": "MaxMaster3000",
    "image": "maxMaster3000p1289312312.png",
    "filter": [
    {
         "id" : 12,
         "name": "Raucherbereich",
         "type": "boolean"
         "values": [
             {
                 "id": 23,
                 "value":"true"
            },
            {
                "id":24,
                 "value":"false"
             }
        ]
        },
        {
         "name": "Locationart",
        "type": "list",
        "values": [
             {
                "id": 20,
                "value":"Restaurant"
            },
            {
                "id":21,
                "value":"Bar"
            },
            {
                "id": 22,
                "value":"Disko",
            }
            ]
        }
    ]
 }
```

```
Nicht erfolgreich:
{
}
```

id - ID des Nutzers.

email - E-Mail des Nutzers.

name - Benutzername des Nutzers.

image - Bildername.

filter - Der aktuelle Filter aus der DB. Besteht aus:

id - eindeutige Id des Filters.

name - Name des Filters, der in der GUI angezeigt wird.

type - Typ des Filters (z.B. Boolean oder List).

values - Liste von Filter Values. Diese bestehen aus:

id - eindeutige Id des Filter Values.

value - value als String für die GUI.

Für den Fall, dass der Nutzer sein Passwort vergisst und sich somit nicht mehr einloggen kann, besteht die Möglichkeit, das Passwort zurücksetzen zu lassen (siehe AN-F-11.04). Die Maske für das Zurücksetzen des Passworts wird vom Login-Startbildschirm aus erreicht. Für die Eingabe gibt der Nutzer seine E-Mail-Adresse ein. Es erscheint eine Mitteilung, dass erfolgreich ein neues Passwort an die E-Mail-Adresse geschickt wurde. Die Eingaben werden über den NetworkService an die Middleware geschickt. Dieser empfängt wieder die Response. Sollte das Passwort-Zurücksetzen fehlgeschlagen sein, z.B. weil die eingegebene E-Mail-Adresse nicht korrekt war, erreicht die App eine Fehlermeldung. Der Nutzer kann das Passwort über sein Profil wieder ändern und ein eigenes Passwort eintragen.

In der Middleware verarbeitet der `ResetPasswordController` den Request mit dem Requestparameter `email` und ruft den `ResetPasswordService` auf. Die Eingabe des Parameters wird mit der Datenbank abgeglichen und geprüft, ob diese E-Mail-Adresse und damit der Nutzer überhaupt existiert. Sind die Daten korrekt, wird das Passwort des Nutzers überschrieben und ein neues, automatisch generiertes für ihn in die Datenbank geschrieben, bevor ein JSON Response-Objekt mit Status 'ok' erstellt wird. Die automatische Generierung erfolgt mithilfe des `PasswordHelper` aus dem *Infrastructure Projekt*. An dieser Stelle wird aus den Groß-und Kleinbuchstaben des Alphabets und den Zahlen von 0 bis 9 ein zufälliges, 16-stelliges Passwort generiert. Dieses Passwort wird über den `MailSender` von der E-Mail-Adresse "pgrecsys@gmail.com" an die angegebene Adresse des Nutzers geschickt.

Passwort Zurücksetzen - Request/Response. REST-Path: /resetpassword

Request:

email – E-Mail-Adresse des Nutzers.

Response:

```
{
    "status": "ok"
}

{
    "status": "error"
}
```

status - Ergebnis der angeforderten Aktion: 'ok': hat funktioniert; 'error': hat nicht funktioniert (z.B. E-Mail nicht gefunden).

Das Logout des Nutzers (siehe AN-F-11.06) ist über das Menü in der App möglich. Dadurch beendet der Nutzer seine aktuelle Sitzung. Es werden persönliche Daten, Empfehlungen, Check-Ins, Favoriten, Nutzerinformationen (E-Mail, Name) etc. vom Device entfernt. Auf der Server-Datenbank sind sie jedoch noch vorhanden und werden bei einem Login wieder geladen.

7.8.3 Profilseite

Auf der Profilseite der App können benutzerspezifische Inhalte verwaltet werden (siehe Anforderung AN-F-10). Dazu zählen der Benutzername, das Passwort, die E-Mail Adresse und das Profilbild.

In der App gelangt der Nutzer über das Menü zu seinem Profil. Die `ProfileActivity` (siehe Abbildung 7.16) gibt dem Nutzer die Möglichkeit, E-Mail-Adresse und Benutzernamen zu ändern. Durch den 'Speichern'-Button wird jeweils ein Request über den `NetworkService` für die entsprechende Operation gemacht, sofern dies durch eine Änderung des Nutzers notwendig ist. Im Anschluss wird dem Nutzer visuelles Feedback über die erfolgreiche oder fehlgeschlagene Durchführung seines Requests gegeben.

7.8.3.1 Profil-Name

Die Parameter werden vom `ProfileController` auf der Middleware empfangen. Für das Ändern des Benutzernamens werden folgende Parameter durch den `NetworkService` an den Controller gesendet: `userId` und `userName`. Der Controller übergibt die Parameter an den *ProfileChangeService*, durch den der Nutzer anhand der `userId` aus der Datenbank geladen wird, um anschließend den neuen Nutzernamen zu setzen. Sobald der Vorgang abgeschlossen ist, wird eine 'ok'-Response an die App gesendet und eine entsprechende Rückmeldung ausgegeben. Der Nutzername wird auf der App auf den neuen Namen gesetzt. Geht bei dem Vorgang etwas schief, wird eine 'error'-Response gesendet und ausgegeben.

Profile-Namen ändern - Request/Response. REST-Path: /changeprofile/username

Request:

userId - eindeutige Identifikation des angemeldeten Nutzers.

userName - neuer Nutzername, den der Nutzer zukünftig verwenden möchte.

Response:

```
{
    "status": "ok"
}

{
    "status": "error"
}
```

status - Ergebnis der angeforderten Aktion: 'ok': hat funktioniert; 'error': hat nicht funktioniert.

7.8.3.2 E-Mail-Adresse

Die E-Mail Adresse (und damit die Login-Daten) werden analog zum Benutzernamen in der `ProfileActivity` geändert und auf der Middleware auf die gleiche Weise bearbeitet. Auch hier wird der Nutzer durch den `ProfileChangeService` anhand der `userId` geladen, um die neue E-Mail zu setzen. Bei erfolgreicher Durchführung wird das 'ok'-Response an die App gesendet und entsprechend verarbeitet.

E-Mail-Adresse ändern - Request/Response. REST-Path: /changeprofile/email

Request:

userId - eindeutige Identifikation des angemeldeten Nutzers.

newEmail - neue E-Mail-Adresse, die der Nutzer zukünftig verwenden möchte.

Response:

```
{
    "status": "ok"
}

{
    "status": "error"
}
```

status - Ergebnis der angeforderten Aktion: 'ok': hat funktioniert; 'error': hat nicht funktioniert.

7.8.3.3 Passwort

In der App gelangt man über die `ProfileActivity` über eine Verlinkung in die `ChangePasswordActitivy`, in der der Nutzer durch die Bestätigung seines alten Passwortes und des Eintragens

eines neuen Passwortes ein neues Passwort setzen kann. Das Drücken des Bestätigungsbuttons löst einen Request durch den `NetworkService` an die Middleware aus, dessen Response als visuelles Feedback dargestellt wird.

Die Middleware-REST-Schnittstelle läuft ebenfalls über den `ProfileController` und den `ProfileChangeService`. Die Parameter die hierbei an den Controller gesendet werden sind: `userId`, `oldPassword`, `newPassword`. Der Service lädt den Nutzer aus der Datenbank und vergleicht anschließend das gesendete `oldPassword` mit dem Passwort, das in der Datenbank hinterlegt ist. Stimmen die Passwörter überein, wird das `newPassword` als neues Passwort gesetzt und eine entsprechende Antwort an die App gesendet.

Passwort ändern - Request/Response. REST-Path: /changeprofile/password

Request:

userId - eindeutige Identifikation des angemeldeten Nutzers.

oldPassword - bisheriges Passwort, das das Nutzerkonto schützt.

newPassword - zukünftiges Passwort, mit dem der Nutzer sein Konto schützen möchte.

Response:

```
{
    "status": "ok"
}

{
    "status": "error"
}
```

status - Ergebnis der angeforderten Aktion: 'ok': hat funktioniert; 'error': hat nicht funktioniert.

7.8.3.4 Profilbild

Das Ändern des Profilbilds ist eine weitere Funktionalität, die auf der Profilseite bereitgestellt wird. Hier sucht der Nutzer ein Bild auf seinem Smartphone aus, um dieses hochzuladen und als sein Profilbild zu verwenden. Der Nutzer gelangt über einen Button neben dem Profilbild in der `ProfileActivity` in die `ImageUploadActivity`. In dieser wird dem Nutzer sein aktuelles Profilbild angezeigt. Außerdem hat er die Möglichkeit, ein neues Bild über einen *Action_Pick*-Intent von seinem Device auszuwählen oder direkt ein neues Bild aufzunehmen, das über den *ACTION_IMAGE_CAPTURE*-Intent [2] erstellt und gespeichert wird. Das Bild wird anschließend hochgeladen und als Profilbild verwendet. Der Upload-Request läuft über den `ImageUploadService`, der als Response den Pfad des Bildes, das hochgeladen wurde, erhält und dieses wieder in der `ImageUploadActiviy` anzeigt.

[2] *ACTION_IMAGE_CAPTURE* Intent `http://developer.android.com/training/camera/photobasics.html`

Auch der `ImageUploadService` verwendet wie der `NetworkService` das *OkHttp-Framework*, um die Files an den Server zu übertragen.

Auf der Middlewareseite wird der `ProfileController` angesprochen. Dieser übergibt die Daten an den `ProfileChangeImageService`. Der Nutzer wird anhand der `userId` aus der Datenbank gelesen, um den neuen Bildnamen zu setzen. Der Bildname wird aus einem Hashwert der `userId` und einem Hashwert eines *Universally Unique Identifier* (*UUID*) gebildet. Unter diesem Bildnamen wird das Profilbild in einer Server-Directory gespeichert und ein Response-Objekt mit dem neuen Bildnamen an die App gesendet.

Profil-Bild hochladen - Request/Response. REST-Path: /changeprofile/image

Request:

userId - eindeutige Identifikation des angemeldeten Nutzers.

image - Bild als MultipartFile, das hochgeladen und als neues Profilbild verwendet werden soll.

Response:

```
 Erfolgreich:
{
    "status": "path"
}

Nicht erfolgreich:
{
    "status": "error"
}
```

status - Ergebnis der angeforderten Aktion: den Image-Path zum neu hochgeladenen Bild: hat funktioniert; 'error': hat nicht funktioniert.

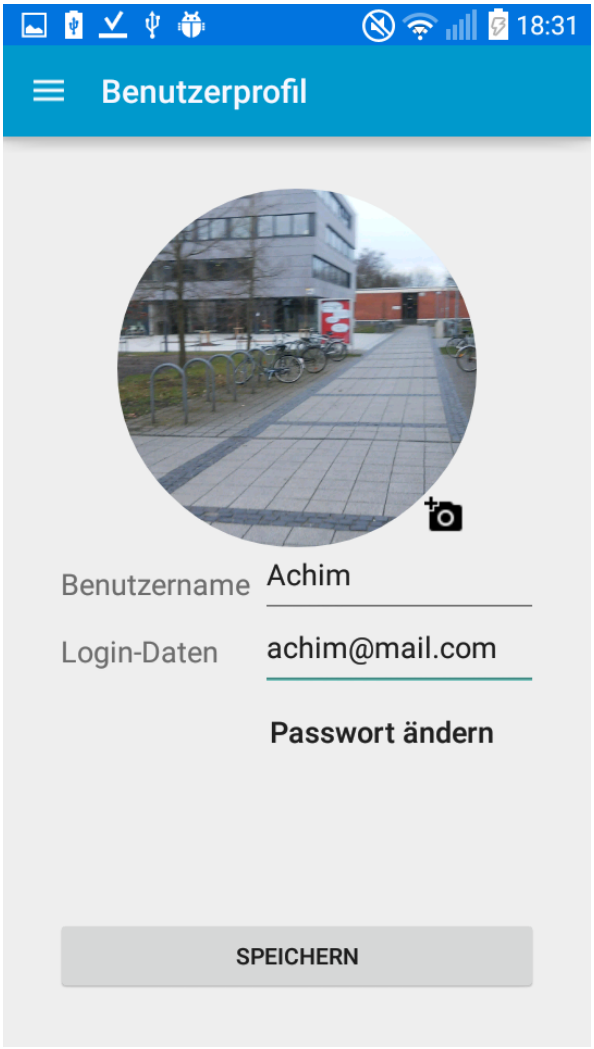


Abbildung 7.16: Profil-Seite der App. Quelle: Eigene.

7.9 Favoriten hinzufügen

Eine weitere Funktion der App ermöglicht es dem Nutzer, bestimmte Locations zu favorisieren. Damit wird die Anforderung AN-F-07 umgesetzt. Jedes Mal, wenn der Nutzer sich einloggt, wird ein Request an die Middleware gesendet, der alle vom Nutzer favorisierten Locations mit Predicted Ratings zurücksendet. Diese werden anschließend für die Dauer der Sitzung des Nutzers auf dem Device gespeichert. Favoriten unterliegen jedoch genauso wie Recommendations und Check-Ins einem 6 Stunden- Aktualisierungsrhythmus, damit das Predicted Rating immer aktuell bleibt. Außerdem werden Favoriten genauso wie Check-Ins mit der implementierten Strategie im Offline-Betrieb synchronisiert (siehe Abschnitt 7.1).

Favoriten Anfordern - Request/Response. REST-Path: /fav/get

Request:

userId - die eindeutige Identifikation des derzeit eingeloggten Nutzers.

Response:

```
[
    {
        "name": "Fiddlers Green",
        "imagePath": "fiddlersgreen.jpg",
        "moreInformation": "Wallstraße 19 Innen Stadt",
        "id": 1106,
        "rating": 0.0
    }
]
```

name - Name der Location.

imagePath - Bildpfad der Location.

moreInformation - weitere Informationen zu der Location (derzeit Adresse).

id - ID der Location.

rating - Prediced Rating zu der Location.

Der `FavoriteController` leitet die Parameter an den `FavoriteService` weiter. Dieser ließ über die `userId` alle `locationIds` aus, die der Nutzer favorisiert hat und schickt diese an Odysseus. Da ein Nutzer so viele Locations favorisieren kann, wie er möchte, unterliegt diese Anfrage keinem Top-k-Operator, der normalerweise bei Empfehlungsanfragen verwendet wird, um nur die besten Ergebnisse herauszugeben. Die favorisierten Locations werden mit dem Predicted Rating als JSON-String zurück an die App gesendet.

Der Nutzer kann in der `FavoriteActivity` alle seine als Favorit markierten Locations einsehen, die analog zu den Empfehlungen in einer Liste angezeigt werden (siehe Abbildung 7.17).

Auf der Detailseite jeder Location kann der Nutzer über das Herzsymbol eine Location zu seinen persönlichen Favoriten hinzufügen. Sobald der Nutzer das Herzsymbol betätigt, werden folgende Parameter an den REST-Controller `FavoriteController` geschickt und ein Favoriten-Objekt auf dem Device gespeichert. Dieses enthält jedoch noch nicht die Informationen über das Predicted Rating, die aus dem Empfehlungssystem kommen. Deswegen wird nach jedem Hinzufügen eines Favoriten beim Zurückkehren in die `FavoritActivity` die Favoritenliste neu vom Server angefordert.

Favoriten Hinzufügen - Request/Response. REST-Path: /fav/add

Request:

locationId - ID der Location, die der Nutzer favorisieren möchte.

userId - ID des Nutzers, der eine Location favorisieren möchte.

Response:

```
{
    "status": "ok"
}
```

```
{
    "status": "error"
}
```

status - 'ok': Operation erfolgreich; 'error': Operation nicht erfolgreich.

Möchte der Nutzer eine Location aus seiner Favoritenliste löschen, so kann er dies tun, indem er erneut das Herzsymbol in der Detailansicht der Location auswählt oder den Favoriten per Wischgeste aus der Favoritenliste schiebt. Über den `FavoriteController` und den `FavoriteService` wird die Location daraufhin aus der Favoritenliste der Datenbank gelöscht.

Favoriten Löschen - Request/Response. REST-Path: /fav/delete

Request:

locationId - ID der Location, die der Nutzer entfavorisieren möchte.

userId - ID des Nutzers, der eine Location entfavorisieren möchte.

Response:

```
{
    "status": "ok"
}
```

```
{
    "status": "error"
}
```

status - 'ok': Operation erfolgreich; 'error': Operation nicht erfolgreich.

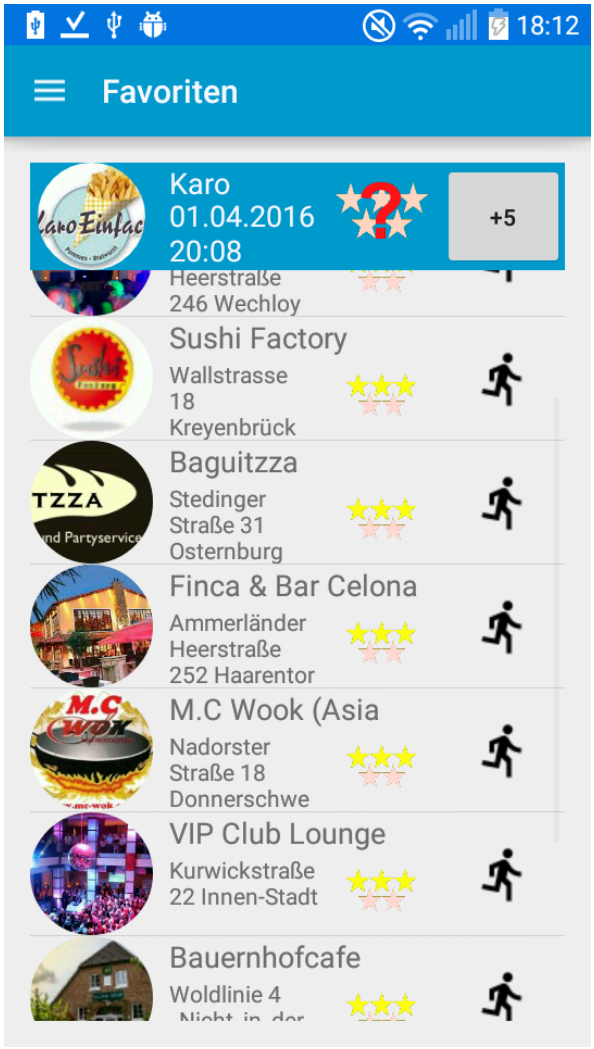


Abbildung 7.17: Favoritensicht der App. Quelle: Eigene.

7.10 Dashboard

In diesem Abschnitt wird die Umsetzung der Anforderung AN-F-12 sowie der Unter-Anforderungen dokumentiert.

7.10.1 Initialisierung

In Anforderung AN-F-12.07 wird umgesetzt, dass über das Dashboard die Middleware mit Daten initialisiert werden kann. Diese Funktion dient dazu, ein lauffähiges System bereitstellen zu können.

Die Initialisierung fügt Daten und Schemata in das angeschlossene DBMS ein. Diese Daten umfassen:

1. Test-Nutzer

2. Test-Filterschema (mit Spezialfiltern für die Suche)

3. Test-Locations (aus einer CSV-Datei)

4. Test-Ratings

Dafür werden im `InitialisationController` entsprechende REST-Methoden angelegt, die über den Client angesprochen werden können. Dabei gilt zu beachten, dass sie in der hier aufgeführten Reihenfolge aufgerufen werden müssen. Um ein Rating generieren zu können, muss es Nutzer, die bewerten und eine Location, die bewertet werden kann, geben. Zudem muss, um eine Location anlegen zu können, ein Filterschema vorhanden sein. Die 7.18 zeigt die Umsetzung der Initialisierung im Dashboard.

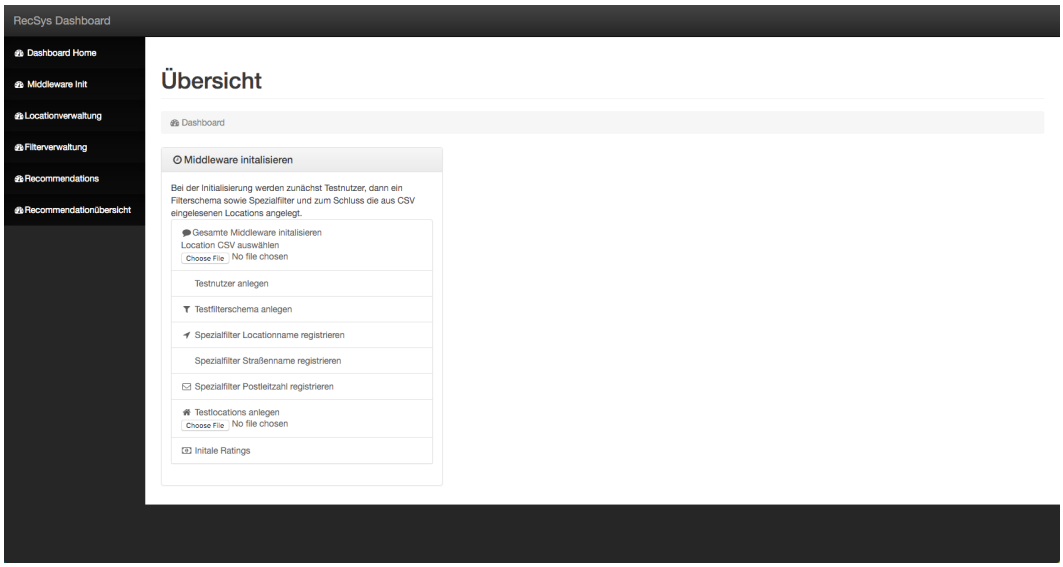


Abbildung 7.18: Ansicht der Initialisierung der Middleware im Dashboard. Quelle: Eigene.

Test-Nutzer werden aus der `InitUser`-Klasse heraus geladen und in die Datenbank gespeichert.

Das Test-Filterschema löscht alle vorhandenen Schemata aus der Datenbank, bevor es ein neues Schema erstellt. Danach können Spezialfilter für die Suche (Namen-Suche, Straßen-Namen-Suche und Postleitzahl-Suche) angelegt werden.

Die Test-Locations werden aus einer CSV-Datei entnommen und müssen dem festgelegten Datenbankschema entsprechen. Es ist wichtig, dass das Filterschema bereits vorhanden ist, damit die Filterattribute der Locations richtig angelegt werden können.

Die Test-Ratings erstellen für jede vorhandene Location ein festgelegtes Gesamtrating und eine vorgegebene Menge von Detailratings.

Außerdem gibt es einen REST-Controller `initAll`, der alle hier beschriebenen Methoden in der richtigen Reihenfolge anspricht und so die Middleware vollständige initialisiert.

7.10.2 Locations

Sobald der Admin die Locationverwaltung aufruft, wird der `LocationManagementController` aufgerufen. Dieser ruft zunächst den `LocationService` auf und lädt alle Locations, die in der Datenbank gespeichert, sind in eine Liste. Diese Liste wird dem `ModelAndView` übergeben und dadurch an das Dashboard gereicht.

Im Dashboard wird mit dieser Liste ein `DataTable` dynamisch erstellt, in der alle Locations mit den Informationen Location-ID, Locationname und Anschrift dargestellt werden. Zudem werden pro Eintrag zwei Buttons hinzugefügt. Mit einem lässt sich die Location löschen, mit dem anderen können die Details der Location abgerufen werden (siehe 7.19).

Für das Löschen einer Location wird der `LocationManagementController` aufgerufen und bekommt die Location-ID der Location übergeben, die gelöscht werden soll. Der Controller ruft dann den `LocationService` auf, der das Löschen der Location über den `LocationManager` durchführt.

Drückt der Admin auf den Detailbutton, wird eine Anfrage an den `LocationController` gesendet. Hier wird wieder die Location-ID der jeweiligen Location übergeben. Der Controller holt über den `LocationService`, dem die Location-ID übergeben wird, die Detailinformationen der Location. Die Detailinformationen werden als JSON der neuen `LocationDetailView` übergeben, zu der der Admin im Anschluss gelangt. Hier wird das JSON-Objekt ausgelesen und alle vorhandenen Informationen wie Adresse, Filter, Standort und Bild auf der Seite angezeigt.

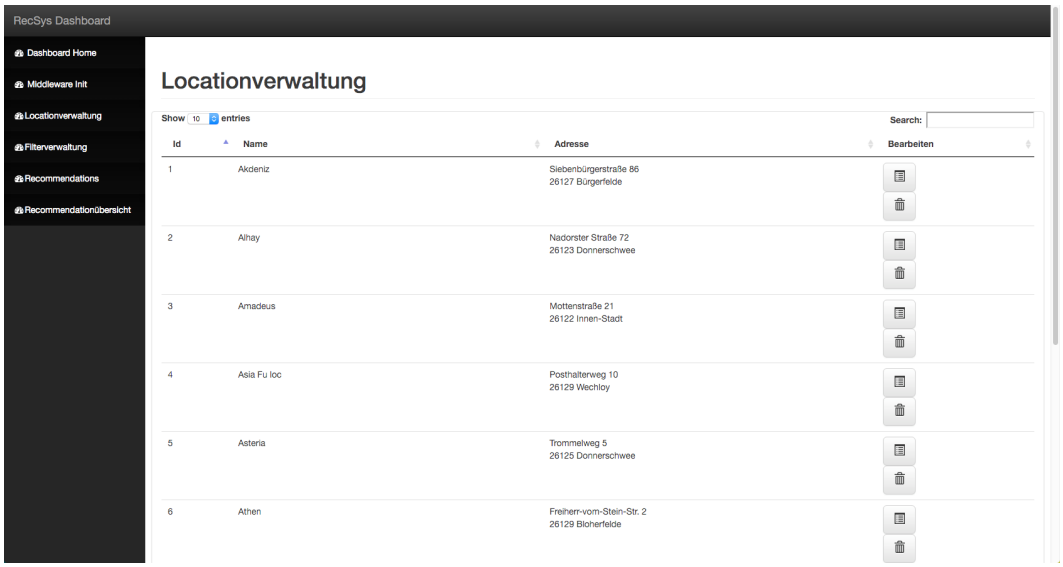


Abbildung 7.19: Abbildung zeigt Locationverwaltung des Dashboards. Quelle: Eigene.

In der Verwaltungsansicht hat der Admin zudem die Möglichkeit, neue Locations anzulegen. Wählt der Admin diese Funktion aus, wird der `LocationController` aufgerufen, der den Admin auf eine neue Seite weiterleitet. Auf der neuen Seite findet der Admin ein detailliertes Formular, in das er sämtliche Informationen der Location eintragen kann. Dazu gehören neben Name und Anschrift auch die Details wie Öffnungszeiten, Musikgenre und andere Filter. Der Admin kann die Location, sobald er alle Daten eingetragen hat, abschließend in der Datenbank speichern. Dazu wird der `Lo-`

cationController angesprochen, der über den LocationService die neue Location in der Datenbank speichert.

7.10.3 Filter

Das FilterManagement im Dashboard ermöglicht die Verwaltung der Filter-Entitäten (vgl. Kapitel 6.1.2). Es existieren Funktionen zum Anlegen, Bearbeiten und Löschen von FilterAttributen und FilterValues (siehe 7.20). Außerdem können die vorhandenen FilterSchemata verwaltet werden, in dem das jeweils aktive Filterschema ausgewählt wird und neue Filterschemata angelegt werden. Zusätzlich können Filterschemata, die nicht aktiv sind, gelöscht werden.

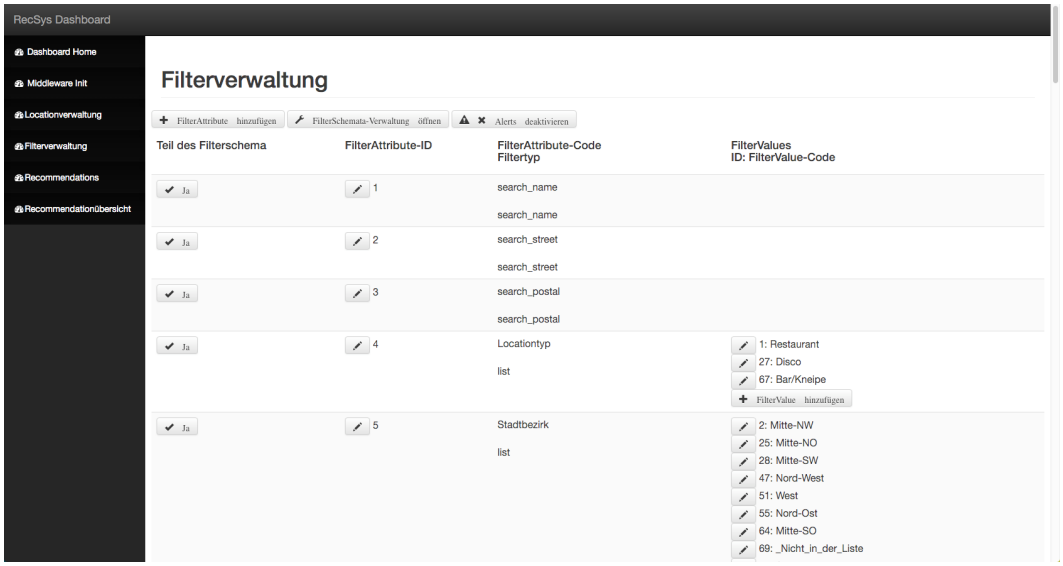


Abbildung 7.20: Ansicht der Filterverwaltung im Dashboard. Quelle: Eigene.

Nachdem ein Filterschema aktiviert wurde, können Filterattribute diesem hinzugefügt werden. Beim Anlegen und Bearbeiten von Filterattributen und Filtervalues können direkt die passenden Übersetzungen eingegeben werden. Es können Übersetzungen in Deutsch, Englisch und Niederländisch eingegeben werden. Dies wird nicht direkt über die Datenbank gelöst, kann aber jederzeit so erweitert werden, dass die verfügbaren Sprachen aus der Datenbank geladen werden. Filtervalues können nur für Filterattribute des Typs *list* hinzugefügt, bearbeitet oder gelöscht werden. Beim Anlegen eines Filterattributs des Typs *boolean* werden automatisch die zwei Filtervalues `true` und `false` angelegt und dem Filterattribut zugeordnet.

Hierbei werden die in der Datenbank vorhandenen Description-Objekte mit den Codes `true` und `false` verwendet.

Beim Anlegen von Filterattributen kann ein beliebiger Text als Filtertyp angegeben werden, sodass später auch Spezialfilter hinzugefügt werden können. Bei Spezialfiltern können keine Filtervalues hinzugefügt werden. Diese Designentscheidung wird dadurch begründet, dass Spezialfilter eine Sonderbehandlung erfordern, da sie nicht über Filtervalues abbildbar sind und weil über Attribute der Locations gefiltert wird.

7.10.4 Odysseus

Die Odysseus-Seite des Dashboards gibt dem Admin die Möglichkeit, neben dem Systemstatus auch den RMSE einzusehen und Einstellungen am Top-k vorzunehmen. Für alle drei Funktionen greift der `OdysseusController`, der die Seite bereitstellt (siehe Abbildung 7.21), auf unterschiedliche Service-Klassen zu.

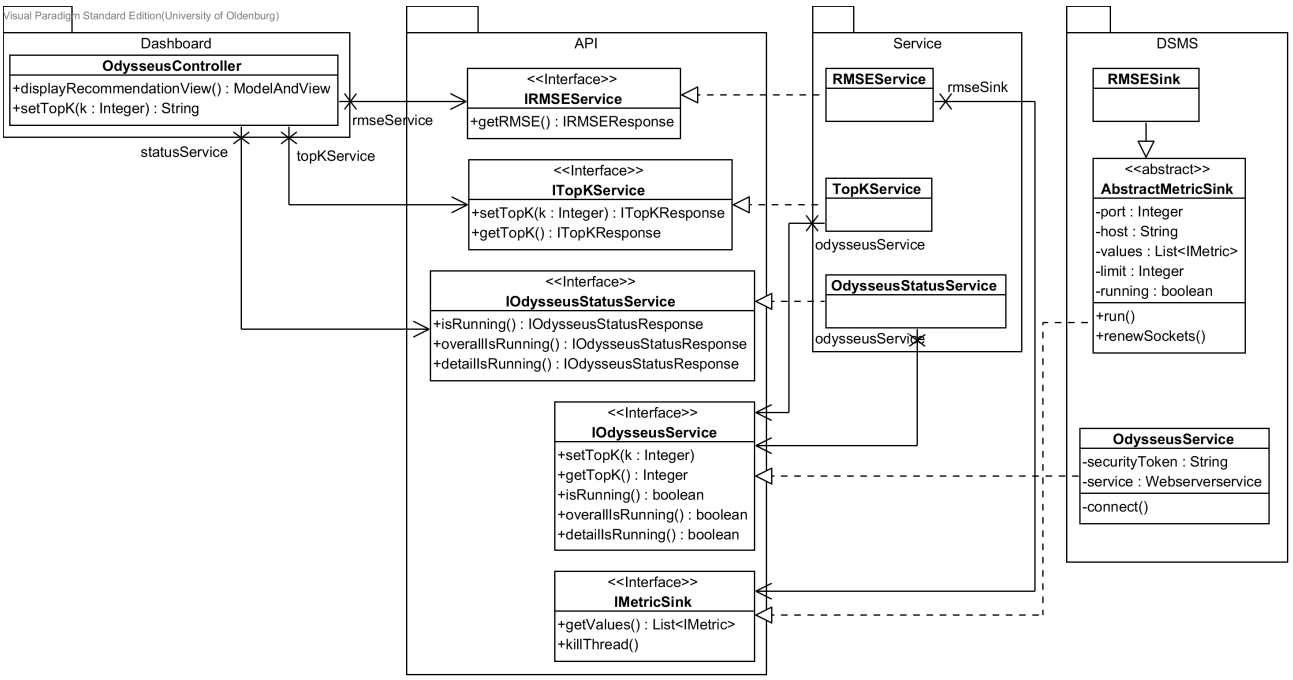


Abbildung 7.21: Klassendiagramm zu den Services, die die Dashboardseite Odysseus benutzt. Quelle: Eigene.

Sowohl der `TopKService` als auch der `OdysseusStatusService` nutzen den `OdysseusService` aus der DSMS-Schnittstelle. Die Aufgabe des `OdysseusService` ist es, eine Verbindung zur Web Services Description Language (WSDL)-Schnittstelle von Odysseus herzustellen und verschiedene Funktionen davon bereitzustellen. Über die WSDL-Schnittstelle lassen sich beispielsweise auf dem Odysseus-Server Queries starten, beenden oder auslesen. Odysseus bietet zwar eine REST-Schnittstelle an, diese bieten jedoch nur einen geringen Funktionsumfang, weshalb die umfangreiche WSDL-Schnittstelle genutzt wird.

Über den `OdysseusService` lässt sich aktuell das k vom Top-k bearbeiten. Hierfür werden alle Queries mit einem regulären Ausdruck nach Top-k-Parametern durchsucht und durch das neue k ersetzt. Anschließend wird das entsprechende Query beendet und mit dem neuen k neu gestartet.

Des Weiteren lässt sich überprüfen, ob der Odysseus-Server überhaupt erreichbar ist und ob die Queries für Detailbewertungen oder Gesamtbewertungen auf dem Server eingerichtet sind. Hierfür wird lediglich überprüft, ob entsprechende Sources und Sinks aktiv sind, da es im Gegensatz zu den Queries dort seltener Änderungen bei dem Umfang oder der Benennung gibt. Auf der Service-Schicht werden diese Funktionalitäten in verschiedene Services gekapselt, um eine bessere Modularität zu erlauben. Dadurch ist es auch möglich, bei Bedarf nur einen Teil der Funktionen der Services durch eine andere Realisierung des Interfaces zu ersetzen.

Neben dem `OdysseusStatusService` und `TopKService` gibt es als letzte Funktion der Odysseus-Seite des Dashboards den `RMSEService`. Dieser Service dient dazu, den RMSE von Odysseus zu beziehen und zur Verfügung zu stellen. Dieser wird wiederum in der Benutzeroberfläche als Diagramm dargestellt (siehe 7.22). Auch der `RMSEService` greift auf die DSMS-Schnittstelle zu. Die DSMS-Schnittstelle stellt hierfür die `RMSESink` zur Verfügung. Die `RMSESink` stellt eine Verbindung zu einer Senke des Odysseus-Servers, auf der RMSE ausgegeben wird, her.

Der RMSE ist eine gängige Metrik zur Bestimmung der Qualität von Empfehlungen[Sa11] (siehe 4.7). Die `RMSESink` selber enthält nur die Informationen die nötig sind, um eine Verbindung zur Senke herzustellen. Die gesamte Logik ist in `AbstractMetricSink` enthalten. Diese Klasse dient zum Empfangen verschiedener Metriken, die nur aus einem Wert vom Typ `Double` bestehen. Die `AbstractMetricSink` startet bei der Initialisierung einen Thread, der dauerhaft Metrikwerte von der Senke empfängt. Sobald ein Metrikwert empfangen wird, werden der Wert und der aktuelle Zeitstempel als `Metric`-Objekt in einer `LinkedList` gespeichert. Die Liste kann nun von den Services jeder Zeit abgerufen werden. Die Metrikwerte werden jedoch nur zur Laufzeit gespeichert, da im Falle eines Neustarts des Systems auch ein neues Modell zur Empfehlungsgebung erstellt wird und die Berechnung der Metrik von vorne beginnt. Neben den Verbindungsinformationen lässt sich der `AbstractMetricSink` ein Limit für die gespeicherten Metrikwerte übergeben. Aus diesem Grund wird eine `LinkedList` verwendet. So ist es möglich, sobald das Limit erreicht wird, mit wenig Aufwand das erste Element in der Liste zu entfernen. Wenn das Limit auf -1 gesetzt wird, werden alle Metrikwerte gespeichert.

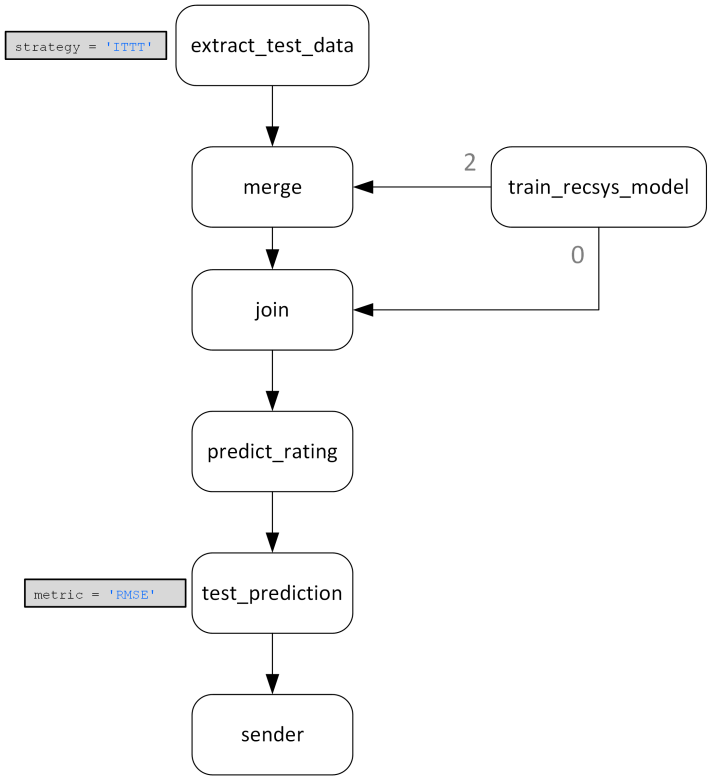


Abbildung 7.22: Operatoren für die Berechnung und Ausgabe des RMSE. Quelle: Eigene.

Zur Ausgabe des RMSE ist auf dem Odysseus-Server das Query, das in Abbildung 7.22 zu sehen ist, aktiv. Es greift auf den `ExtractTextData`-Operator zu. Dabei handelt es sich um den selben Operator, der auch bei der Bewertung von Locations aufgerufen wird. An dieser Stelle teilt der Operator die eingehenden Bewertungen in Testdaten und Lerndaten, wie es bereits im Abschnitt 7.2.1 beschrieben ist. Dabei werden die Testdaten auf Port 1 ausgegeben und in dem Query für den RMSE weiterverarbeitet. Um den RMSE zu ermitteln ist es als erstes notwendig, eine vorhergesagte Bewertung für die gerade bewertete Location zu ermitteln. Das Vorgehen dabei ist identisch mit dem einer normalen Empfehlungsgebung. Vom `TrainRecSysModel`-Operator werden Heartbeats und das Modell bezogen. Mit dem Modell und der Bewertung wird ein Kreuzprodukt gebildet. Anschließend wird für die Location mit dem `PredictRating`-Operator die Bewertung vorhergesagt. Anhand der eigentlichen Bewertung und der vorhergesagten Bewertung kann anschließend der `TextPrediction`-Operator den RMSE berechnen. Als Parameter wird die gewünschte Metrik übergeben. In diesem Fall handelt es sich um den RMSE. Weitere Metriken müssen in Zukunft hinzugefügt werden. Der `Sender` gibt zum Schluss nur den RMSE aus, der von der Middleware entgegen genommen wird.

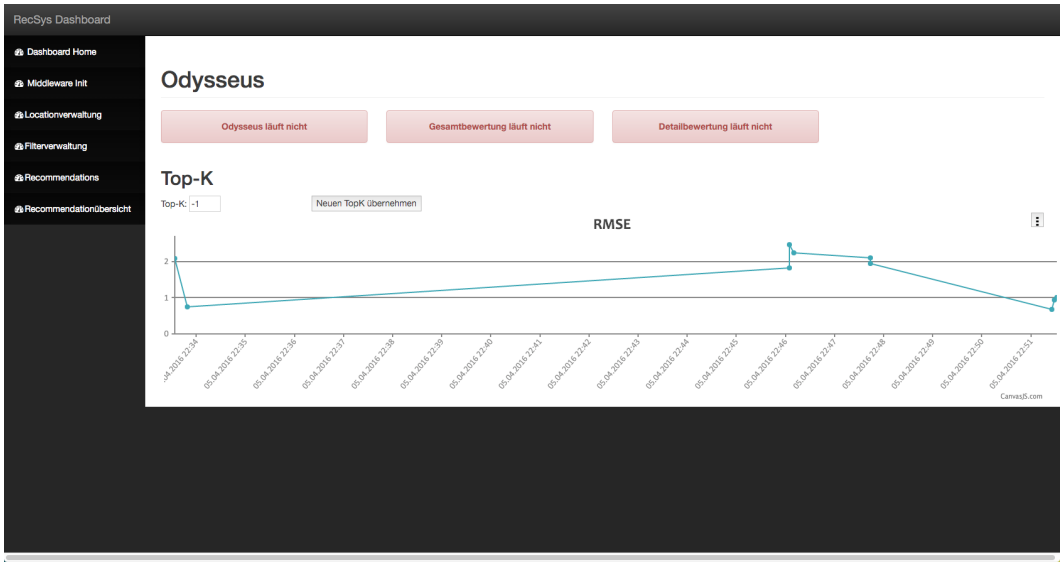


Abbildung 7.23: Ansicht der Odysseusübersicht mit Top-k, Status und RMSE . Quelle: Eigene.

7.10.5 Recommendationübersicht

Auf der Recommendationübersicht werden dem Admin die letzten 100 ausgegebenen Recommendations angezeigt. Die Tabelle wird dynamisch aus einer Liste von JSON-Objekten, die dem `ModelAndView` angehängt ist, aufgebaut. Abbildung 7.24 zeigt eine fertig aufgebaute Seite der Übersicht.

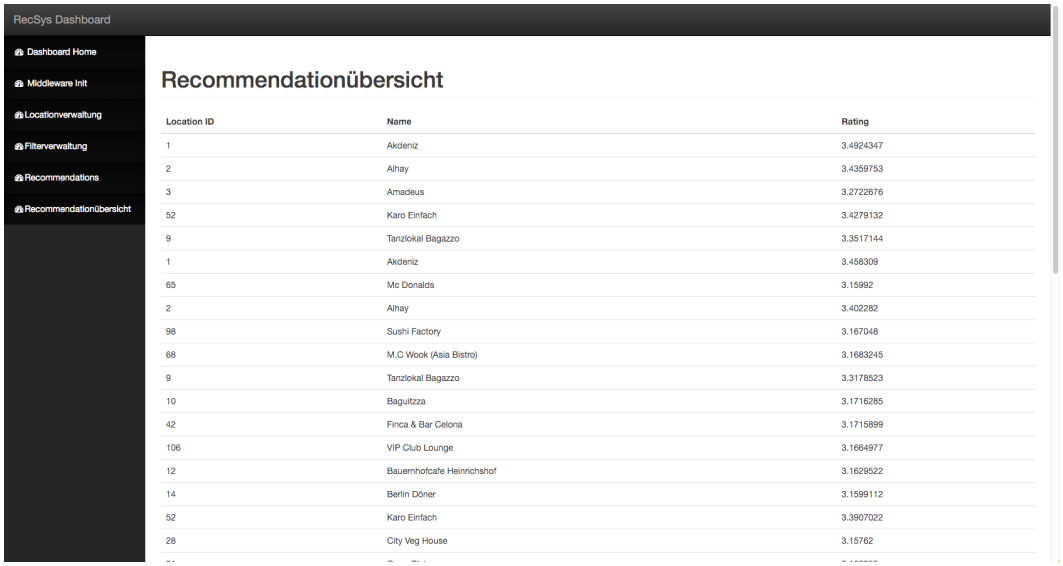


Abbildung 7.24: Die Recommendationübersicht im Dashboard. Quelle: Eigene.

Für die Realisierung ist die Klasse `AbstractRecommendationSink`, die eine Socketverbindung zu Odysseus aufbaut und auf einem eigenen Thread läuft, implementiert. Sie überwacht den Ausgabestrom der Recommendations von Odysseus und speichert sie in einer `LinkedList` ab.

Der `StreamDataController` holt sich bei einem Seitenaufruf des Livestreams die `LinkedList`, die in der `AbstractRecommendationSink` gespeichert ist und wandelt sie in eine Liste aus JSON-Objekten um. Das JSON-Objekt enthält die `itemId`, `locationName` und `predictedRating`. Die neue `LinkedList` wird dem `ModelAndView` angehängt und damit dem Dashboard übergeben. Im Dashboard wird dann über dieses JSON-Objekt iteriert und dynamisch die Tabelle erzeugt.

8 Test und Validierung

Im Folgenden werden zunächst beispielhaft durchgeführte Testprotokolle, die der Validierung von Kernfunktionalitäten dienen, vorgestellt. In der Testautomatisierung wird insbesondere dargestellt, welche Bereiche des Systems zu welchem Anteil über Tests abgedeckt sind. Mit dem Testen wird allgemein der Qualitätssicherung, die übergreifend für Anforderungen notwendig ist, Rechnung getragen.

8.1 Testprotokolle

Bewertung eines Check-Ins

Zu testende Funktionen Dieser Test soll die Funktionalität des Bewertens eines Check-Ins validieren (siehe AN-F-01).

Vorbedingungen Damit ein Nutzer einen Check-In in der App bewerten kann, muss sich dieser zunächst ein Nutzerkonto anlegen, indem er sich registriert. Danach muss sich der Nutzer mit seinen Daten in der App anmelden. Eine weitere Vorbedingung ist, dass der Nutzer Empfehlungen bekommt und dann einen Check-In in eine der ihm empfohlenen Locations durchführt. Sind diese Vorbedingungen erfüllt, kann der Nutzer den Check-In in der App bewerten.

Einzelschritte

1. Eingaben

   Der Nutzer wählt aus, dass er einen Check-In bewerten möchte, indem er auf ein nicht bewerteten Check-In drückt. In der neuen View gibt der Nutzer zuerst die Gesamtbewertung und danach optional die Detailbewertung für den Besuch an. Anschließend kann der Nutzer optional einen Kommentar verfassen und dann die Bewertung abschicken.

2. Ausgaben

   Nachdem der Nutzer seine Bewertung abgegeben hat, wird der zugehörige Eintrag aus dem Check-In-Fragment entfernt und in der Liste der letzten Check-Ins wird die angegebene Bewertung angezeigt. Hat der Benutzer einen Kommentar verfasst, wird dieser noch in der Detailseite der Location, die besucht wurde, angezeigt.

Umgebung Getestet wurde mit der Androidversion Android 5.0.2 (API 21).

Besonderheiten Damit die Bewertung vom Empfehlungssystem berücksichtigt werden kann, muss eine Internetverbindung bestehen.

Erhalten von Empfehlungen

Zu testende Funktionen Dieser Test soll das Erhalten von Empfehlungen validieren (siehe AN-F-02) .

Vorbedingungen Damit der Nutzer Empfehlungen erhalten kann, muss er sich zunächst in der App registrieren und danach mit seinem Konto anmelden.

Einzelschritte

1. Eingaben

   Sobald der Nutzer sich mit seinem Konto in der App anmeldet, werden Top-k Empfehlungen für diesen Nutzer abgerufen. Eine weitere Möglichkeit ist, durch einen expliziten Wunsch eine Aktualisierung dieser Liste anzufordern.

2. Ausgaben

   Der Nutzer erhält, je nach Einstellung, Top-k Empfehlungen in Form einer Liste angezeigt. Wünscht der Nutzer eine aktualisierte Liste wird, diese neu angezeigt.

Umgebung Getestet wurde mit der Androidversion Android 5.0.2 (API 21).

Besonderheiten Damit neue Empfehlungen auf der App angezeigt werden können, muss eine Internetverbindung bestehen.

8.2 Testautomatisierung

Für die Qualitätssicherung sollen zentrale Funktionalitäten mit automatisierten Tests geprüft werden. Dazu wird die Junit-Bibliothek genutzt. Neben Junit werden auch *Mock* und *Powermock* von Mockito[1] verwendet. Die Tests werden beim Maven-Install ausgeführt.

Die Tests liegen in dem Verzeichnis *src/test/java*. Das Package, in dem die Testklasse erstellt wird, ist das gleiche wie das Package mit der Klasse, die getestet wird. Das hat den Vorteil, dass z. B. aus der Testklasse auf protected-Attribute der zu testenden Klasse zugegriffen werden kann (beim Mocken z.B. notwendig). Es werden Tests für die Projekte Service, DSMS und Domain geschrieben.

Beim Testen werden die entityspezifischen Manager-Klassen gemockt. Dafür werden die Klassen mit *@Mock* annotiert. Durch diese Annotation werden gemockte Instanzen initialisiert. Der zu testende Service wird mit *@InjectMocks*, *@Resource* und *@Spy* annotiert. Damit wird ermöglicht, dass einzelne Methoden beim Testen abgefangen und gemockt werden können.

Wenn die Datenbankverbindung für den Test der Services nicht gemockt wird, werden die Tests so aufgebaut, dass keine Voraussetzungen, wie zum Beispiel das Vorhandensein von Usern oder Locations mit bestimmten IDs in der Datenbank, erfüllt sein müssen. Stattdessen sollen vor dem Test (Methode mit *@BeforeClass* annotiert) die benötigten Daten erstellt, in der Datenbank persistiert und nach der Durchführung (*@AfterClass*) gelöscht werden. So kann einerseits der Test jederzeit ausgeführt werden und andererseits verbleiben keine Reste in der Datenbank.

Für die Ermittlung der Testabdeckung wird die Bibliothek *Jacoco* in der Version 1.0 genutzt. Jacoco wurde vom *EclEmma Java Code Coverage* Team der *Mountainminds GmbH* entwickelt.[2] Die Bibliothek wird in der Pom.xml eingebunden. Die ermittelten Ergebnisse der einzelnen Packages sind in der Tabelle 8.1 zu finden.

Im Domain-Projekt werden 46 % der insgesamt 4.933 Instructions mit Tests abgedeckt. Das DSMS-Projekt hat eine Testabdeckung von 46 % bei 46.336 Instructions und im Service-Projekt werden 1.323

[1] http://mockito.org/ besucht am 30.3.2016.
[2] http://eclemma.org/jacoco/index.html besucht am 30.03.2016.

Projekt	**Testabdeckung**
Domain	46 %
DSMS	46 %
Service	29 %

Tabelle 8.1: Testabdeckung in den einzelnen Packages.

von 4.511 Instructions (29 %) getestet. Die aus Sicht der Projektgruppe wichtigsten Funktionalitäten sind damit mit Tests abgedeckt. Hierbei lag der Fokus insbesondere auf den Anwendungsfällen Bewertung und Empfehlungsgebung.

9 Evaluation und Projektreview

9.1 Projektreview

Aus der Anforderungsermittlung und -analyse in den Kapiteln 4 und 5 ist ein umfangreiches Product Backlog entstanden. Die Projektgruppe hat von Beginn an die User Stories, deren Grundlage die Anforderungen stellen, kontinuierlich priorisiert und dementsprechend in den Sprints abgearbeitet. Durch die agile Vorgehensweise sind einige der Anforderungen erst im Projektverlauf ermittelt und aufgenommen worden (siehe Kapitel 4.9). Eine Vielzahl an Tätigkeiten, wie z.B. die Beseitigung von Fehlern, sind ebenfalls als zusätzliche Aufgaben hinzugekommen. Durch diese Vorgehensweise ist sichergestellt worden, dass die für den Projekterfolg signifikanten Anforderungen allesamt erfüllt sind. Anforderungen, die mit niedriger Priorisierung in das Product Backlog aufgenommen worden sind, sind dagegen teilweise nicht mehr bearbeitet worden. Hierbei handelt es sich größtenteils um solche Anforderungen, die Kernfunktionalitäten um wünschenswerte Features erweitert hätten, für die eigentliche Zielsetzung aber als weniger relevant einzustufen sind. Die folgenden Anforderungen aus Kapitel 5 sind nicht direkt umgesetzt worden:

AN-F-02.04 Der Nutzer sollte die Möglichkeit haben, über eine Detailansicht weitere Informationen über die Empfehlungen zu erhalten.

AN-F-04.03 Die App sollte dem Nutzer Bewertungen einer Location in einer separaten Liste anzeigen.

AN-F-06 Die Listenansicht sollte durch personalisierte Informationen individualisiert werden.

AN-F-08 Die App sollte mithilfe einer Push-Funktion dem Nutzer weitere Informationen bereitstellen.

AN-F-12.04 Das Dashboard sollte es dem Admin ermöglichen, Algorithmen zu vergleichen.

Die Detailbewertung ist implementiert worden, die Unter-Anforderung AN-F-02.04, für die diese Bewertungen dem Nutzer angezeigt werden sollen, ist dagegen nicht umgesetzt. Die Information, auf welcher Grundlage von Detailbewertungen sich die Empfehlung zusammensetzt, wird aus folgendem Grund nicht angezeigt: Zwar wird die Bewertung für eine Location für einen User auf Grundlage mehrerer Modelle (für jede Detailbewertung eins) erstellt, wird als Ergebnis der Berechnung innerhalb von Odysseus lediglich die berechnete Gesamtbewertung an die App geschickt. Das Senden jeder Detailbewertung als Zusatzinformation für eine Location würde Mehraufwand mit sich bringen, da Gesamtbewertung und Detailbewertungen extra übermittelt werden müssten, der letztliche Nutzen für den Endbenutzer jedoch in Frage zu stellen ist. Diese Art von Informationen sind eher für den Admin oder den Betreiber interessant, die bereits über das Dashboard Einsicht in diese Informationen haben. Aus einem ähnlichen Grund ist die Anforderung AN-F-04.03 nicht in die Tat umgesetzt worden. Diese hätte erfordert, sämtliche Detailbewertungen über eine Location separat zu speichern und der App zur Verfügung zu stellen. Auch hier wäre der Aufwand des zusätzlichen Traffics und der Anpassungen nicht dem Ertrag angemessen gewesen. In beiden Fällen handelt es sich lediglich um Zusatzinformationen, die dem Nutzer bereitgestellt werden. Sie betreffen damit weder eine Kernanforderung noch können sie als gewichtig für die Usability gewertet werden.

Eine Individualisierung der Listen mittels personalisierter Informationen (AN-F-06) ist nur indirekt umgesetzt worden. Diese Anforderung impliziert die Anpassung der angezeigten Listeninhalte, ist

aber nicht weiter konkretisiert. Eine Möglichkeit, beispielsweise im Offline-Modus die Listen auf der App zu filtern, ist implementiert.

Die Anforderung AN-F-08 war in der Projektgruppe Gegenstand kontroverser Diskussion: Einerseits wurden im Zuge der Anforderungsanalyse deutliche Vorbehalte und Abneigung potenzieller Nutzer gegen Push-Funktionalitäten ermittelt (siehe Kapitel 4.4.2). Dies stand meistens im direkten Zusammenhang mit dem Datenschutz. Andererseits ist für die Projektgruppe früh klar gewesen, dass die App nicht zu kommerziellen Zwecken auf den Markt kommt. Die Gefahr, den Datenschutz der Nutzer zu verletzen, bestand demnach von Vornherein nicht. Zudem wurde diese Funktionalität in einer anderen Erhebungsmethode als potenzielle Anforderung identifiziert (siehe Kapitel 4.5.1) und gleichzeitig von der Projektgruppe als spannendes Feature angesehen und ist daher trotz der Ergebnisse eines Teiles der Anforderungsermittlung in den Anforderungskatalog aufgenommen worden. Allerdings ist hier die Priorität als niedrig eingestuft worden, ebenfalls aus den o.g. Gründen. Aus Kapazitätsgründen ist diese Anforderung daher zum Ende des Projektes nicht mehr in den Sprint Backlog gezogen worden.

Für die Anforderung AN-F-12.04 ist zwar die Grundlage geschaffen, der direkte Vergleich jedoch nicht mehr implementiert worden. Die Detail- und Gesamtbewertungen können separat über das Dashboard angezeigt werden, eine extra Sicht, die diese Algorithmen direkt miteinander vergleicht, existiert nicht. Der Vergleich ist somit lediglich indirekt möglich. Auch in diesem Fall ist aus zeitlichen Gründen auf die Umsetzung der Anforderung verzichtet worden. Hier standen die übrigen Anforderungen und Grundfunktionalitäten, die dem Dashboard zugeordnet werden können, im Vordergrund.

In der Zielsetzung des Projekts sind Usability-Kriterien explizit nicht als primäre Ziele angeführt. Dennoch sind in der App über die rudimentäre Implementierung der Funktionalitäten hinaus diverse Feature implementiert, die die Bedienung für den Nutzer vereinfachen. Die Umsetzung der Anforderung AN-NF-05 lässt sich lediglich subjektiv bewerten. Die Menü-Steuerung vereinfacht die Navigation zwischen den Views in der App und sorgt für die Übersichtlichkeit. Diese wird außerdem über die eingeblendeten Titel in jeder einzelnen View unterstützt. Auch die Wiederverwendung der Liste in mehreren Sichten ist sinnvoll für ein konsistentes Erscheinungsbild. Die Möglichkeiten, ein Profilbild zu speichern und mit anderen Nutzern z.B. über Kommentare oder die Teilen-Funktion sozial zu interagieren, sind als Anregungen zur Nutzung der App zu verstehen. Die Verwendung von Image-Buttons anstelle von Text-Buttons, die Darstellung sämtlicher Bilder als Round-Images und die Vermeidung von zuviel Text, z.B. in der Empfehlungsliste, genügen gängigen Usability-Kriterien. Die Benutzerverwaltung beinhaltet Funktionalitäten, die dem Nutzer Sicherheit gewährleisten. Vergisst er sein Passwort, kann er es zurücksetzen lassen. Zudem kann er es jederzeit ändern. Unter Berücksichtigung der Kernziele des Projektes und der niedrigen Priorität dieser Anforderung kann die Ausgestaltung der App hinsichtlich Benutzerfreundlichkeit als voll ausreichend angesehen werden.

Alle im oberen Abschnitt nicht genannten und damit der Großteil der Anforderungen aus dem Anforderungskatalog, ist von der Projektgruppe umgesetzt worden. Das in Kapitel 1.1 festgelegte, übergreifende Ziel ist damit erreicht. Es ist ein konkreter Anwendungsfall, mit dem die Recommender-Funktionen von Odysseus potenziellen Kunden und Nutzern anschaulich gemacht werden können, umgesetzt worden.

Im Detail lag der Fokus auf den Anforderungen, die direkt mit Odysseus in Verbindung stehen. Das sind primär die Anforderungen AN-F-01, AN-F-02 und AN-F-12. Gesamt- und Detailbewertungen können vom Nutzer abgegeben werden und fließen in die Empfehlungsgebung von Odysseus ein. Zusätzlich werden Kontextinformationen in Form von zahlreichen möglichen Filtern in der Empfeh-

lungsgebung miteinbezogen. Das Dashboard für den Admin ist mit wesentlichen Funktionalitäten aufgesetzt.

Aus Sicht des Projektmanagements kann das Projekt als gut strukturiert mit abschließend zufriedenstellenden Ergebnis bewertet werden. Auf der Grundlage einer ausführlichen Anforderungsanalyse erlaubte Scrum im weiteren Projektverlauf die Anpassung an die flexiblen Rahmenbedingungen, die kontinuierliche Priorisierung der Anforderungen sowie das Hinzufügen neuer Anforderungen (siehe Kapitel 4.9). Die Anforderungsanalyse ist aus der Notwendigkeit des Erstellen eines Product Backlogs heraus durchgeführt worden. Die Anforderungsanalyse wird als gute Grundlage und Einstieg in das Projekt gesehen. Aufgrund der mangelnden Projekterfahrung der Projektgruppe ist die Analyse sehr zeitaufwändig ausgefallen, um dem Projekt von Vornherein einen klaren Rahmen und einen eindeutigen Startpunkt zu geben. Rückblickend hätte die Anforderungsanalyse weniger umfangreich durchgeführt werden können, da im Nachhinein mehrere Anpassungen und Ergänzungen vorgenommen werden mussten. Die aus der Anforderungsanalyse abgeleiteten Aufgaben wurden in Jira-Tasks (Tickets) umgesetzt. Das Beschreiben von Tickets in den Teamsitzungen hat zu Beginn viel Zeit gekostet. Mit der Entscheidung, für die Beschreibung komplexer Aufgaben eigene Tickets zu erstellen, konnten die Sitzungen stark verschlankt werden. Durch diese sogenannten *Product Owner*-Tickets sind zudem die Qualität der Beschreibungen der Tickets verbessert worden und die Aufwandsschätzung präziser ausgefallen. Die Entscheidung erleichterte grundsätzlich die Pflege des Product Backlogs.

9.2 Evaluation der Performance

Im Rahmen der Implementierung der Funktionalitäten des Projekts wurde auf eine leistungsfähige Ausgestaltung geachtet. Die folgenden Abschnitte zeigen auf, wie im Einzelfall die Performance optimiert wurde, um der Anforderung AN-NF-01 zu begegnen. Zudem gibt es eine Analyse der Anfragedauer der verschiedenen Funktionalitäten und Methoden, die im Rahmen des Projektes implementiert wurden. Im ersten Abschnitt ist die Optimierung der Anfrage mithilfe von Hibernate beschrieben. Im zweiten Abschnitt sind verschiedene Einflussfaktoren betrachtet, die Auswirkungen auf die Performance der Empfehlungsanfragen haben könnten.

9.2.1 Performance von Hibernate

Hibernate stellt viele Basisfunktionalitäten bereit, die zur persistenten Speicherung von Objekten innerhalb des Projekts genutzt werden können. Die Logik für die Behandlung komplexerer Anwendungsfälle muss jedoch weiterhin über eigene Methoden bereitgestellt werden, die JPA- beziehungsweise Hibernate-Funktionen kapseln. Hierbei muss in jedem Fall die Performance komplexer Anfragen beachtet werden, besonders dann, wenn Objekte `Collections`, bestehend aus Kind-Objekten und unidirektionale Beziehungen, enthalten. Hibernate entlastet standardmäßig durch die Verwendung von *Lazy Loading* die Datenbank, da `Collections` somit erst nachträglich beim tatsächlichen Zugriff auf Objekte oder Attribute geladen werden.

Je nach Anwendungsfall kann Lazy Loading allerdings auch zu Performance-Einbußen führen. Dies ist im Projekt besonders beim Login sowie bei der Anforderung von Empfehlungen über die REST-Schnittstelle deutlich geworden. Hierbei wird für den Login das in der Datenbank hinterlegte Filterschema mit sämtlichen Filtermöglichkeiten erstellt und für die Empfehlungen werden die er-

mittelten Locations mit zusätzlichen Informationen angereichert. Da diese beiden Szenarien zudem Kernfunktionalitäten des Projekts darstellen, sind hierfür spezielle Anfragen geschrieben worden.

Diese deaktivieren für fest definierte Szenarien partiell das Lazy Loading, wozu einige Model-Klassen angepasst werden mussten. Der Grund hierfür ist die abweichende Behandlung der `Collection`-Typen `List` und der `Set`-Typen durch Hibernate. Diese erlaubt je Anfrage kein paralleles Laden von mehr als einer `List`, wenn die Beziehung als *OneToMany* oder *ManyToMany* definiert ist. Der Grund dafür liegt darin, dass Objekte vom Typ `List` keine *Index*-Spalte besitzen, um die enthaltenen Objekte eindeutig identifizieren zu können und somit unter Umständen Objekt-Duplikate enthalten, was bei Hibernate zur mehrfachen Bildung des kartesischen Produktes führen kann. Bei der Verwendung von `Collections` vom Typ `Set` hingegen dürfen keine zwei Objekte enthalten sein, für die `obj.equals(other)` gilt. Für diese Objekte muss in der Folge daher eine eigene Implementierung der Methoden `hashcode()` und `equals()` vorgenommen werden. Nach der Anpassung können die für die JSON-Ausgabe benötigten Kind-Objekte gezielt über die Verwendung von *JOIN*-Anweisungen beim ersten Aufruf der Anfrage mitgeladen werden. Somit kann die Ausgabe an die App direkt und ohne weitere Datenbankoperationen erfolgen. Die Anzahl der für die Bereitstellung des *Filterschemas* notwendigen Anfragen hat sich dabei von 337 auf 1 reduziert. Beim Betrieb eines lokal gestarteten Projekts unter Verwendung der Remote-Datenbank des Projektgruppen-Servers konnte die zum Login benötigte Zeit von 30000 ms auf 700 ms verringert werden. Die zur Anforderung von 5 Empfehlungen nötigen Anfragen wurden von 291 auf 5 reduziert. Gleichzeitig hat sich die Laufzeit der Anforderung von 91000 ms auf 4300 ms verringert.

Im selben Zug wurde Hibernate dahingehend konfiguriert, mehrere Anfragen unter Verwendung von *Batching* durchzuführen. Dazu wird eine Funktionalität des Java Database Connectivity (JDBC)-Treibers genutzt, welche das gleichzeitige Senden mehrerer SQL-Anfragen ermöglicht. Somit können beispielsweise beim nachträglichen Laden von `Collections` mehrere Objekte parallel statt seriell angefordert werden.

Diese Maßnahmen sind einer positiven Nutzer-Erfahrung auf der App zuträglich, da der Vorgang des Logins und der Empfehlungs-Anforderung als deutlich fließender empfunden wird.

In einer weiteren Evaluation sind die positiven Auswirkungen der Optimierung ermittelt worden. Dabei sind die Datenbank, die Middleware und der Odysseus-Server lokal auf einer Instanz installiert: Abbildung 9.1 zeigt die durchschnittliche Anfragedauer einer Empfehlung vor der Optimierung. Die Middleware benötigt bei bei der Gesamtbewertung durchschnittlich 897 ms um eine Anfrage zu bearbeiten. Bei der Detailbewertung beträgt die Verarbeitungszeit der Middleware durchschnittlich 1826 ms. Die Anfragedauer des Odysseus-Server beträgt bei der Gesamtbewertung durchschnittlich 1461 ms bzw. für die Detailbewertung 2358 ms.

Die Anfragedauer nach der Optimierung ist in Abbildung 9.2 zu sehen. Die Dauer der Bearbeitungszeit der Middleware bei der Gesamtbewertung ist von 897 ms auf 33 ms abgefallen. Ebenso ist die Bearbeitungszeit bei der Detailbewertung von 1826 ms auf 32 ms abgesunken. Diese Optimerung sollte keine Auswirkungen auf die Bearbeitungszeit des Odysseus-Servers haben.

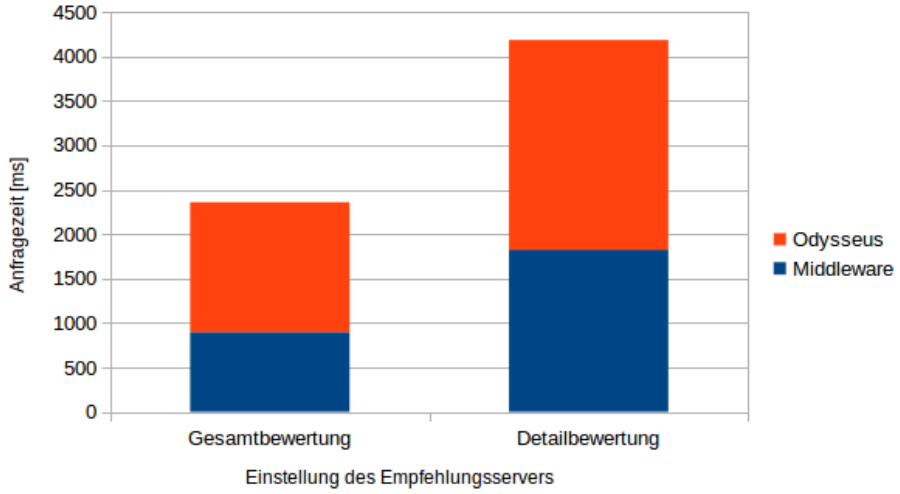


Abbildung 9.1: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen Optimierung von Hibernate auf die Performance der Empfehlungsgebung. Anfragezeit vor der Optimierung. Quelle: Eigene.

9.2.2 Performance der Middleware und Odysseus

Im Rahmen des Projektes sind verschiedene Methoden und Funktionalitäten zur Empfehlungsgebung implementiert worden. In diesem Abschnitt werden verschiedene Einflussfaktoren betrachtet, die Einfluss auf die Performance haben könnten.

Es wird dabei zwischen der Empfehlungsgebung anhand von Detailbewertungen und anhand von Gesamtbewertungen unterschieden. Diese zwei Methoden werden in jedem Szenario betrachtet.

Durch die Projektgruppe sind vier Einflussfaktoren identifiziert worden:

- Anzahl der Nutzer in den Modellen
- Anzahl der Locations in den Modellen
- Anzahl der Empfehlungen (Top-k)
- Boolesche Filter und Listenfilter

Aus den Einflussfaktoren sind Thesen vorformuliert, die mit der Evaluation überprüft werden sollen:

1. Die Empfehlungsgebung anhand einer Detailbewertung hat Einfluss auf die Performance der Middleware und auf den Odysseus-Server.
2. Filter, die der Nutzer in der App zur besseren Eingrenzung der Empfehlung auswählt, haben einen negativen Effekt auf die Performance der Middleware.
3. Je mehr Filter der Nutzer auswählt, desto schlechter wird die Performance.

Abbildung 9.2: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Empfehlungsmethoden auf die Performance der Empfehlungsgebung. Anfragezeit nach der Optimierung. Quelle: Eigene.

4. Die Anzahl der Elemente, die durch den Odysseus-Server empfohlen werden, hat Einfluss auf die Performance der Empfehlungsgebung.

5. Die Anzahl der Nutzer im Modell hat keinen Einfluss auf die Performance der Middleware und auf den Odysseus-Server.

6. Die Anzahl der Locations im Modell hat Einfluss auf die Performance der Middleware und auf den Odysseus-Server.

Abbildung 9.3 zeigt die vier Messpunkte, die zur Erhebung der Performance genutzt werden. Sofern eine Empfehlungsanfrage an die Middleware gesendet wird, wird der `RecommendationController` aufgerufen. In diesem Controller existiert der erste und letzte Messpunkt. Somit wird hier die Gesamtzeit der Anfrage erfasst. Zwei weitere Messpunkte (drei und vier) sind direkt bevor die Anfrage an den Odysseus-Server gesendet wird und direkt nachdem die Antwort des Odysseus-Server an die Middleware gesendet wurde. Somit schließt die erste Messung die Zweite ein. Dadurch kann ermittelt werden, wie lange die Middleware für eine Empfehlungsanfrage benötigt, wie lange der Odysseus-Server für eine Antwort benötigt und wie lange die Gesamtzeit der Empfehlungsgebung ist.

Ein weiterer Faktor, der nicht betrachtet wird, ist die Antwortzeit zwischen Middleware und App. Da diese stark von der Art der Verbindung des mobilen Gerätes abhängig ist, wird diese nicht näher betrachtet.

Die Middleware, der Odysseus-Server und die relationale Datenbank sind alle auf einem lokalen System installiert. Dieses System hat folgende Eigenschaften:

- Betriebssystem: Linux Mint 17.1 Cinnamon 64-bit
- Arbeitsspeicher: 7.7 GiB

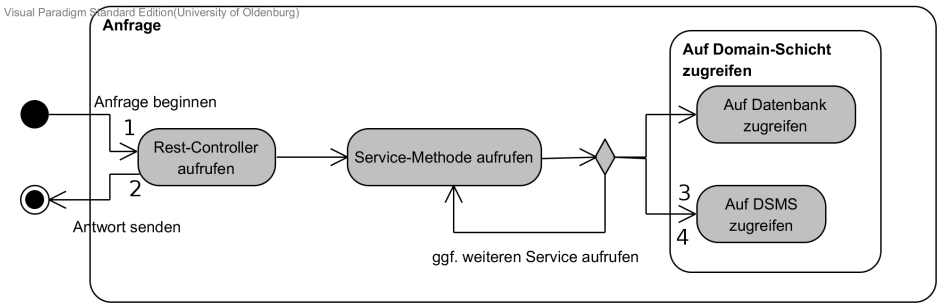


Abbildung 9.3: Schichten der Middleware und Messpunkte für die Evaluation. Quelle: Eigene.

- Prozessor: Intel Core i7-4510U CPU 2.00GHz x2

Im Folgenden sind diese Thesen anhand verschiedener Messungen überprüft worden. Bei allen Messungen ist zu beachten das Odysseus bei einer Anfrage bis zu einer Sekunde benötigt um eine Antwort zu senden, da die `TuplePunctuation`, die die Antwort letztendlich auslöst, erst eine Sekunde nach der Anfrage gesendet wird. Auf diese Problematik wird in Abschnitt 7.3.3.1 genauer eingegangen. Des Weiteren wird nur jede Sekunde ein gültiges Modell zur Empfehlungsgebung gesendet. Auch hier kann es zu Wartezeiten kommen, da im schlechtesten Fall fast eine Sekunde auf ein Modell gewartet werden muss.

9.2.2.1 Gesamtbewertung gegen Detailbewertung

Im ersten Schritt wird die Performance der Gesamtbewertung gegen die Performance der Detailbewertung betrachtet. Folgendes Szenario wurde dabei erstellt:

- Nutzer: 100
- Locations: 1000
- Top-k: 5
- Keine Filter ausgewählt

Es sind jeweils 25 Messungen mit der Detailbewertung und Gesamtbewertung durchgeführt worden. Durch die Ermittlung des Mittelwertes ist die durchschnittliche Anfragezeit ermittelt worden. Abbildung 9.4 zeigt die Ergebnisse der ersten Messung. Die Anfragezeit zwischen den zwei Methoden unterscheidet sich.

Somit ist die These, dass die Empfehlung mit Detailbewertung eine Verschlechterung der Performance mit sich bringt, bestätigt. Sowohl die Middleware als auch der Odysseus-Server verzeichnen Einbußen in der Performance. Dies lässt sich durch die Anzahl der Operatoren in den Anfrageplänen erklären. Bei der Detailbewertung wird die Anfrage mithilfe von vier Modellen beantwortet. Bei der Gesamtbewertung gibt es nur ein Modell.

Jedoch lässt sich die höhere Anfragezeit durch die Middleware nicht erklären.

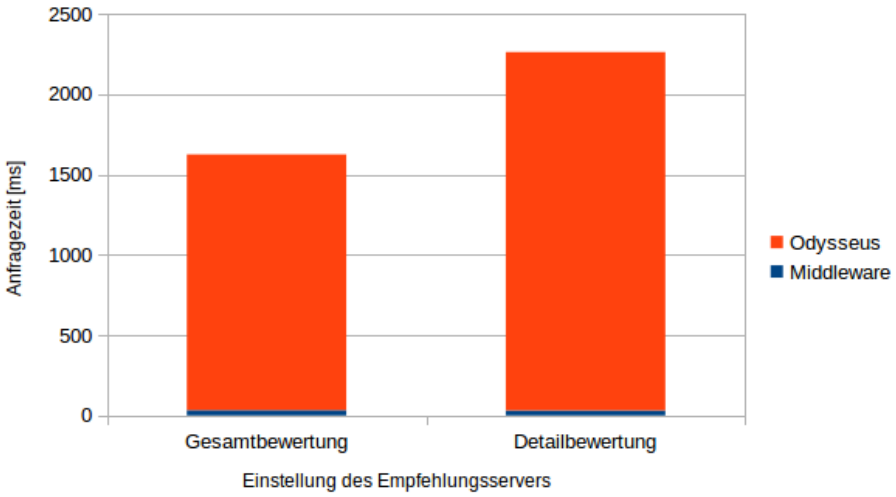


Abbildung 9.4: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Empfehlungsmethoden auf die Performance der Empfehlungsgebung. Quelle: Eigene.

9.2.2.2 Auswirkungen der Filter auf die Performance

In diesem Abschnitt wird die These untersucht, dass die Filter, die der Nutzer in der App zur besseren Eingrenzung der Empfehlung auswählt, einen negativen Effekt auf die Performance haben.

Hierfür werden verschiedene Einstellungen des Filters betrachtet. Es wird evaluiert, wie sich die Anfragezeit mit einem, zwei und drei boolesche Filtern verhält. Ebenso wird betrachtet, wie sich die Anfragezeit mit einem, zwei und drei Listenfiltern verhält. Dabei ist jeweils nur ein Attribut des Listenfilters aktiv gesetzt. Mit folgendem Szenario wurde gemessen:

- Nutzer: 100
- Locations: 1000
- Top-k: 5
- Variable Filter

Abbildung 9.5 und 9.6 zeigen die Ergebnisse der ersten Messung mit den verschiedenen booleschen Filtern. Sowohl bei der Detailbewertung als auch bei der Gesamtbewertung haben die booleschen Filter einen positiven Effekt auf die Antwortzeit. Bei der Gesamtbewertung ist dieser Effekt noch stärker ausgeprägt als bei der Detailbewertung. Die Antwortzeit der Middleware verändert sich kaum. Somit ist die These, dass die Filter einen negativen Effekt auf die Performance haben, für die booleschen Filter widerlegt. Dieses liegt wahrscheinlich daran, dass je mehr boolesche Filter aktiv sind, entsprechend weniger Locations an den Odysseus-Server gesendet werden. Dadurch verringert sich die Anzahl der Elemente im Datenstrom.

Abbildung 9.7 und 9.8 zeigen die Messergebnisse für die Listenfilter. Hier tritt jedoch ein anderes Verhalten als bei den booleschen Filtern auf. Bei einem Listenfilter sind im Durchschnitt die Anfragen

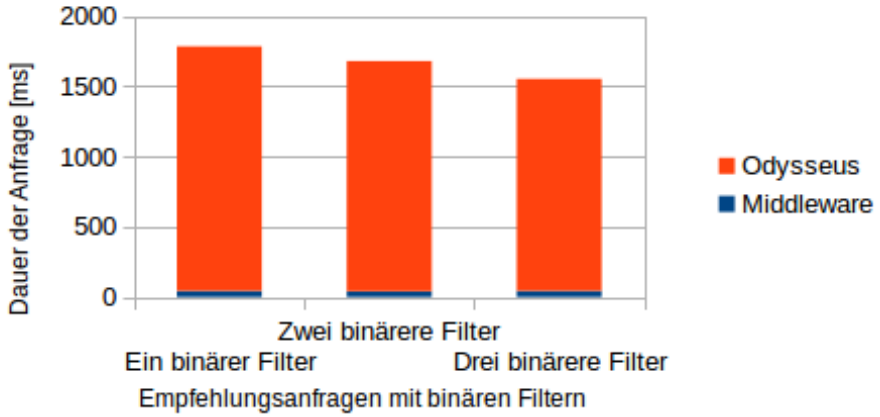


Abbildung 9.5: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Filter auf die Performance der Empfehlungsgebung: Binärfilter (Gesamtbewertung). Quelle: Eigene.

schneller als bei zwei oder drei Listenfiltern. Dieses Verhalten lässt sich jedoch nicht erklären, da die Antwortzeit der Middleware konstant bleibt und nur die Antwortzeit des Odysseus-Servers variiert. Dieser bekommt jedoch unabhängig davon, ob es sich um einen booleschen oder um einen Listenfilter handelt, stets die selben Typen von Datenstrom-Elementen.

Wie Abbildung 9.9 und 9.10 zeigen, gibt es eine Korrelation zwischen den gesetzten Filtern und der Antwortzeit des Systems.

9.2.2.3 Auswirkungen der Empfehlungsmenge auf die Performance

In diesem Abschnitt wird die These untersucht, dass die Anzahl der Elemente, die durch den Empfehlungsserver empfohlen werden, Einfluss auf die Performance der Empfehlungsgebung hat.

Hierfür werden verschiedene Einstellungen des Top-k-Parameters betrachtet. Es werden die Einstellungen 10, 50 und 100 Empfehlungen betrachtet. Mit folgendem Szenario wurde gemessen:

- Nutzer: 100
- Locations: 1000
- Top-k: 10, 50, 100
- Filter: unverändert

Abbildung 9.11 und 9.12 zeigen die Ergebnisse der Messung. Sowohl bei der Detailbewertung als auch bei der Gesamtbewertung steigt die Antwortzeit der Middleware mit der Menge der empfohlenen Elemente. Es ist eine klare Korrelation zwischen diesen zwei Parametern zu erkennen. Dies liegt wahrscheinlich an der Anreicherung der Empfehlungen mit weiteren Informationen über die Location.

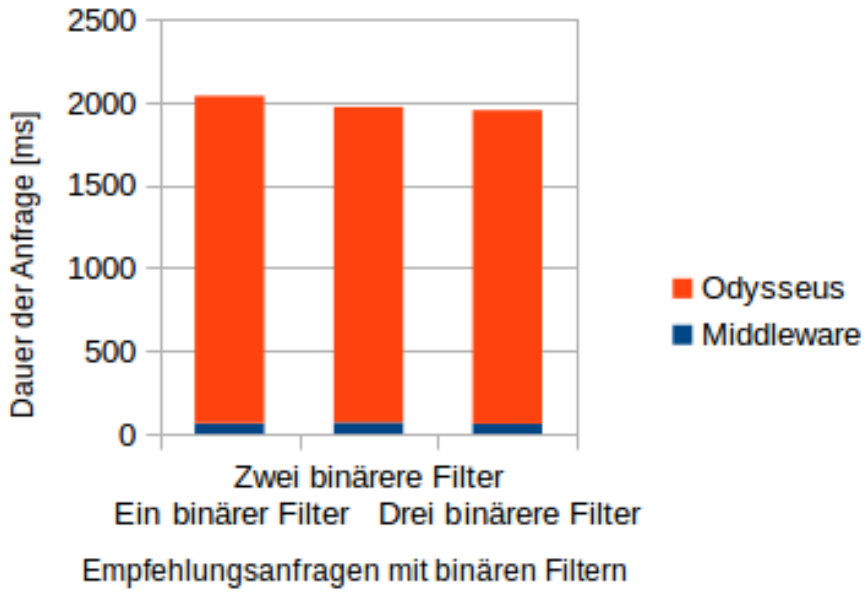


Abbildung 9.6: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Filter auf die Performance der Empfehlungsgebung: Binärfilter (Detailbewertung). Quelle: Eigene.

9.2.2.4 Auswirkungen der Menge an Nutzer auf die Performance

Neben der Menge an empfohlenen Locations wird vermutet, dass die Menge an Nutzern im Modell keine Auswirkungen auf die Performance hat. Abbildung 9.13 und 9.14 zeigen die Ergebnisse der Messung.

Mit folgendem Szenario wurde gemessen:

- Nutzer: 300, 500
- Locations: 1000
- Top-k: 5
- Filter: unverändert
- Angefragte Locations: 1000

Es ist zu erkennen, dass die Dauer der Anfragezeit der Detailbewertung höher ist als die Anfragezeit der Gesamtbewertung. Jedoch verändern sich die Anfragezeiten bei einer variablen Anzahl von Nutzern nicht. Somit ist die These, dass die Anfragezeit abhängig von den Nutzern im Modell ist, widerlegt. Eine mögliche Begründung könnte nahe legen, dass bei einer Empfehlungsanfrage immer nur für einen Nutzer die Empfehlungen berechnet werden. Auch auf der Middleware hat dies keine Auswirkungen.

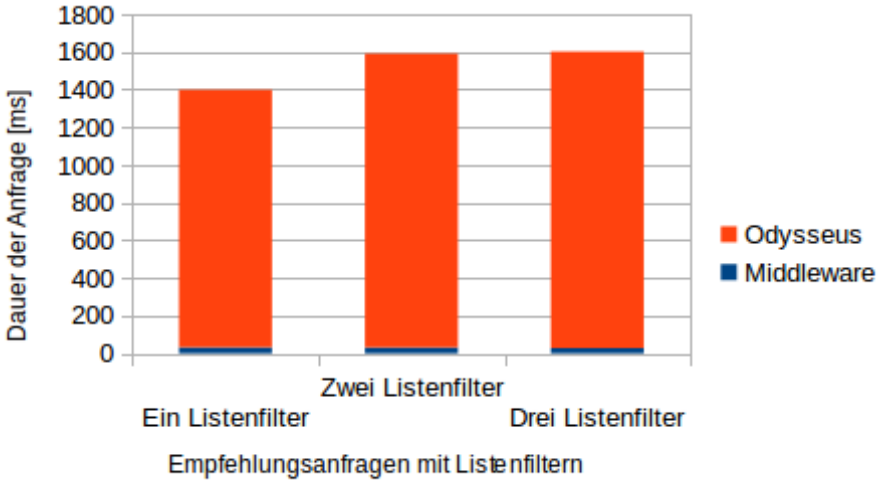


Abbildung 9.7: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Filter auf die Performance der Empfehlungsgebung: Listenfilter (Gesamtbewertung). Quelle: Eigene.

9.2.2.5 Auswirkungen der Menge an Locations auf die Performance

Die Anzahl der Locations im Modell sollte Einfluss auf die Berechnung der Empfehlungen haben, da bei einer Anfrage bei gleicher Anzahl von angefragten Locations die Menge der Berechnungen mit der Anzahl der Locations im Modell skaliert. Die Ergebnisse der Messung sind in Abbildung 9.15 und 9.16 dargestellt. Die Messung ist mit folgenden Einstellungen gemessen worden:

- Nutzer: 100
- Locations: 100, 500
- Top-k: 5
- Filter: unverändert
- Angefragte Locations: 100 und 500

Die Messungen zeigen, dass die Anzahl der Bewertungen Einfluss auf die Anfragedauer bei der Bewertung haben. Die Middleware benötigt etwas länger für die Verarbeitung der Anfrage bei einer steigenden Anzahl von Locations. Dies tritt sowohl bei der Gesamtbewertung, als auch bei der Detailbewertung auf. Der Odysseus-Server benötigt im Schnitt bei der Gesamtbewertung die gleiche Zeit, um eine Anfrage zu beantworten. Bei der Detailbewertung skaliert die Anfragedauer mit der Anzahl der Locations im Modell. Dieses Verhalten sollte eigentlich auch bei der Gesamtbewertung auftreten.

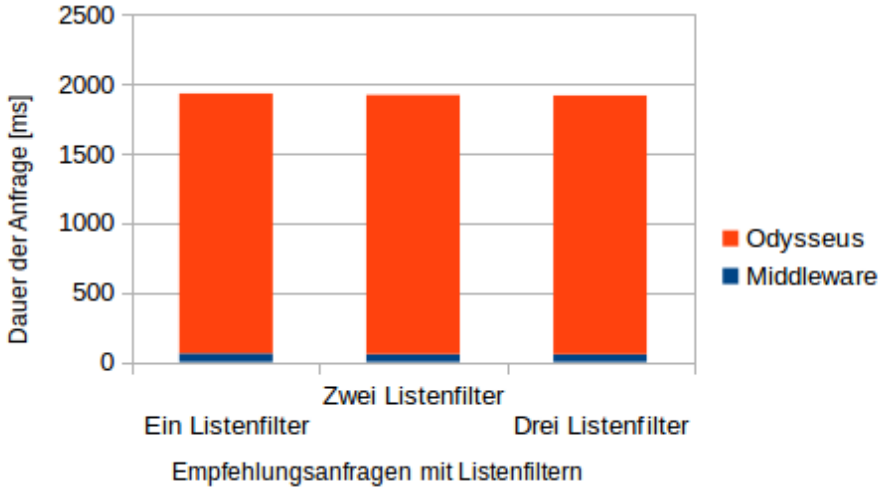


Abbildung 9.8: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Filter auf die Performance der Empfehlungsgebung: Listenfilter (Detailbewertung). Quelle: Eigene.

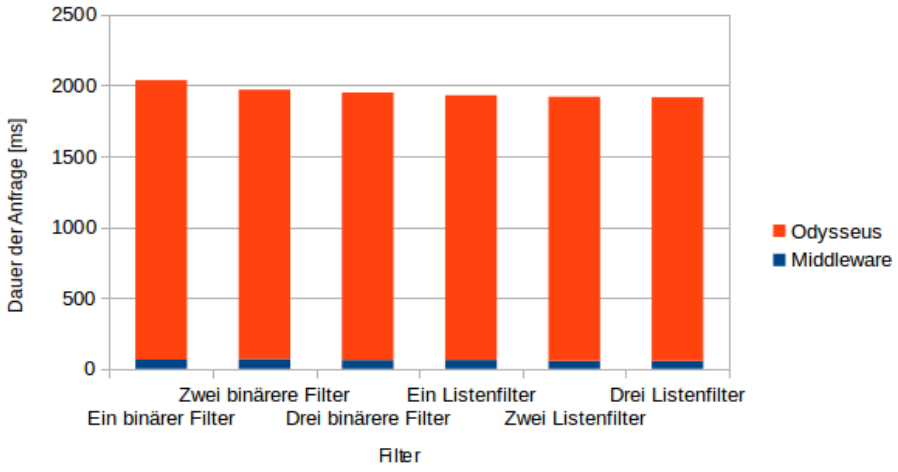


Abbildung 9.9: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Filter auf die Performance der Empfehlungsgebung: Alle Filter (Gesamtbewertung). Quelle: Eigene.

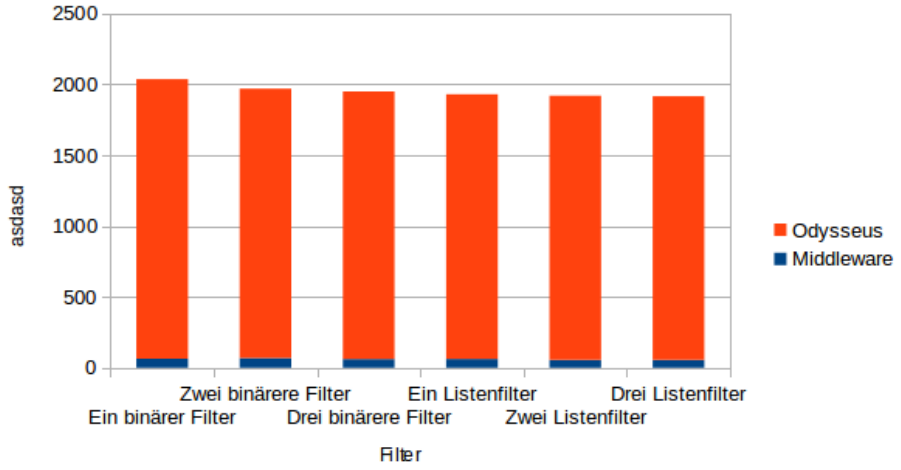


Abbildung 9.10: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Filter auf die Performance der Empfehlungsgebung: Alle Filter (Detailbewertung). Quelle: Eigene.

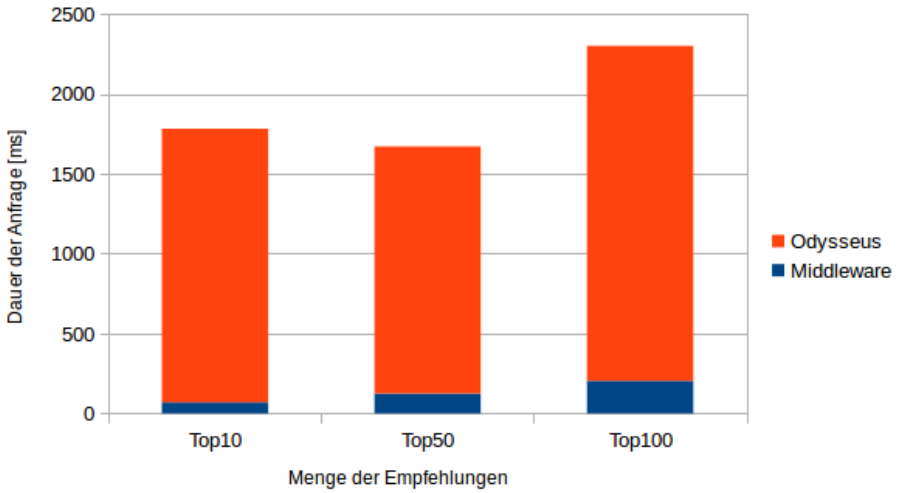


Abbildung 9.11: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Empfehlungsmengen auf die Performance der Empfehlungsgebung (Gesamtbewertung). Quelle: Eigene.

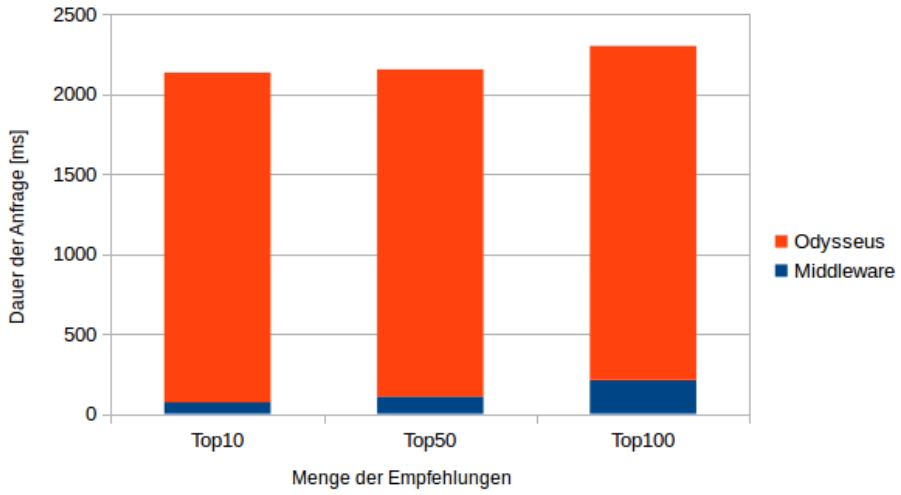


Abbildung 9.12: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Empfehlungsmengen auf die Performance der Empfehlungsgebung (Detailbewertung). Quelle: Eigene.

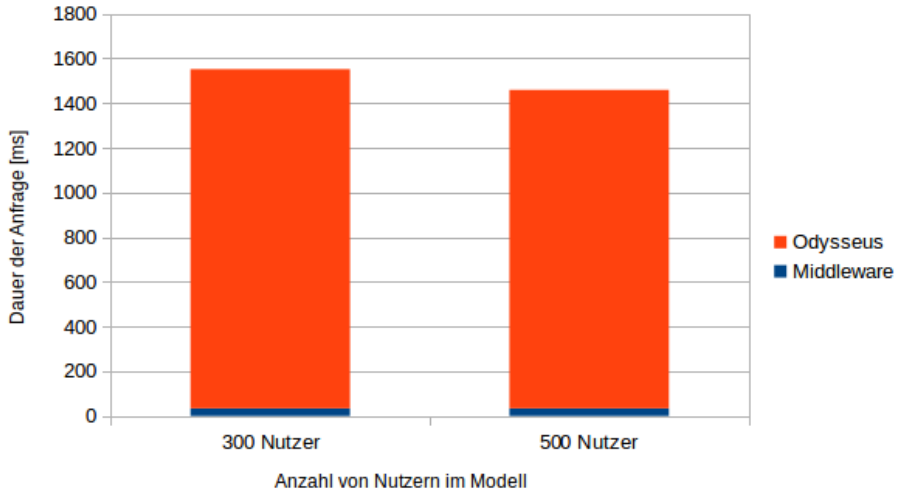


Abbildung 9.13: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Anzahlen von Nutzern auf die Performance der Empfehlungsgebung (Gesamtbewertung). Quelle: Eigene.

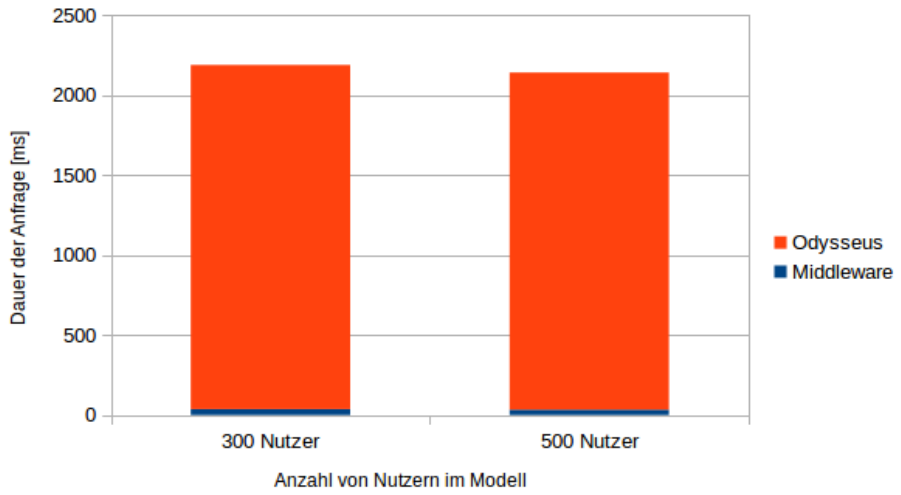


Abbildung 9.14: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Anzahlen von Nutzern auf die Performance der Empfehlungsgebung (Detailbewertung). Quelle: Eigene.

Abbildung 9.15: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Anzahlen von Locations auf die Performance der Empfehlungsgebung (Gesamtbewertung). Quelle: Eigene.

Abbildung 9.16: Darstellung der Messergebnisse zur Überprüfung der Auswirkungen der verschiedenen Anzahlen von Locations auf die Performance der Empfehlungsgebung (Detailbewertung).

9.3 Evaluation der Qualität

Zur Qualitätsbewertung des Empfehlungsmodells wird der RMSE herangezogen. Die Berechnung des RMSE wird in Gleichung 2.35 auf Seite 59 erläutert. Der RMSE ist eine bekannte und häufig verwendete Metrik zur Beurteilung eines Empfehlungsmodells (vgl. [RRSK11], S. 109), sodass dieser auch für die Qualitätsevaluation des Anwendungsszenarios herangezogen wird. Dadurch wird die Abweichung zwischen den vorhergesagten Bewertungen des Systems und den tatsächlichen Bewertungen der Nutzer ermittelt. Dies wird auch in der Zielsetzung des Projektes in Abschnitt 1.2.2 erwähnt.

Dabei werden die RMSEs folgender Modelle verglichen:

Gesamtbewertung Das Modell zur Berücksichtigung von Gesamtbewertungen basiert auf ein Matrix -faktorisierungs-Modell (BRISMF.MOA), welches genauer in dem Abschnitt 2.8 sowie deren Implementierung in Abschnitt 7.2.1 beschrieben wird.

Detailbewertung Der Ansatz zur Berücksichtigung der Detailbewertungen ist in Abschnitt 7.2.2 beschrieben wird. Dabei wird der Durchschnitt der RMSE für den Vergleich herangezogen.

BaselinePredictor Dieses Modell wird eigens für die Evaluation herangezogen (D. h. es ist nicht möglich, dass Empfehlungen für die App basierend auf dem BaselinePredictor errechnet werden.) Der BaselinePredictor errechnet anhand der Durchschnittsbewertungen aller abgegebenen Bewertungen sowie der Durchschnittsbewertungen pro Nutzer und Item eine vorhergesagte Bewertung. (vgl. [RRSK11], S. 148 f.)

9.3.1 Durchführung

Zur Erhebung der BaselinePredictor-Daten wird ein weiterer Query-Plan erstellt, der sich an den aus Abbildung 7.5 orientiert, wobei in diesem das Modell `BaselinePrediction` verwendet wird. Dieser greift auf die gleiche Source zu. Das heißt, wenn eine Gesamtbewertung eingeht, wird neben dem BRISMF.MOA auch der BaselinePrediction gelernt. Dadurch muss die Middleware diesbezüglich nicht angepasst werden.

Es werden sechs Sinks eingerichtet, die jeweils den RMSE der Gesamtbewertung, die vier RMSE der Teilmodelle für die Detailbewertung und der BaselinePredictor in eine CSV-Datei ausgeben, die zur Auswertung herangezogen wird. Damit wird der Verlauf der RMSE dokumentiert.

Bei der Berechnung der RMSE mit dem `TEST_PREDICTION` wird mit unterschiedlichen Fenstern konfiguriert:

- Kein Fenster (alle vorher berechneten RMSE werden berücksichtigt.)
- Fenster von einer Minute (alle in der letzten Minute errechneten RMSE werden berücksichtigt.)

Die Middleware wird um einen weiteren Controller `EvaluationController` erweitert, der eine Datei mit Ratings einliest und die Funktionalitäten der Middleware nutzt (`RatingService`), um Gesamt- und Detailbewertungen abzugeben. Nach jeder Bewertungsabgabe wird auf der Middleware etwas mehr als eine Sekunde gewartet. Dies liegt daran, dass die `TuplePunctuations` auf dem

Odysseus-Server jede Sekunde gesendet werden. Erst dann wird das neu erlernte Modell ausgegeben und zur RMSE-Berechnung herangezogen werden. Durch das Warten auf der Middleware wird sichergestellt, dass der errechnet RMSE auf Basis des neu erlernten Modells basiert.

In dem Testszenario bewerten 100 verschiedene Nutzer 893 verschiedene Locations. Dieses Testszenario wird mithilfe des Tools *RecSysStreamSimulator*[1] erstellt, in dem Ähnlichkeiten zwischen Nutzern berücksichtigt werden.

In dem Datenset geben die Nutzer durchschnittlich 22,29 Bewertungen ab. Jede Location wird im Durchschnitt von 2,49 Nutzern bewertet. Es werden sowohl 22229 Gesamtbewertungen als auch Detailbewertungen (4 Kriterien) abgegeben. Die Gesamtbewertungen betragen im Durchschnitt 3,42.

Für die Request for Recommendations wird Top-k auf 5 gesetzt. Insgesamt werden 5 Request for Recommendations für einen Nutzer angefragt. Es werden keine Filter gesetzt.

9.3.2 Ergebnisse

In diesem Abschnitt werden die Ergebnisse beschrieben und hinsichtlich der Qualität der Algorithmen interpretiert.

9.3.2.1 Ohne Fenster

In Abbildung 9.17 werden die RMSEs der Modelle in Abhängigkeit der eingegangen Bewertungen (Baseline, Gesamt- und Detailbewertung) dargestellt.

Die RMSE aller drei Modelle (basierend auf einem Matrixfaktorisierungs-Modell) sind zu Beginn sehr hoch: Der RMSE der Baseline startet bei ca. 3,5 und reduziert sich nach 73 Bewertungen auf unter 1, der der Gesamtbewertung startet bei 3,5 und reduziert sich nach 54 Bewertungen auf unter 1. Nach etwa 400 Bewertungen ist der RMSE der Baseline kleiner als der der Gesamtbewertung.

Nach allen abgegeben Bewertungen liegt der RMSE der Baseline bei 0,79, der Gesamtbewertung bei 0,81 und der Detailbewertung im Durchschnitt bei 0,98.

9.3.2.2 Fenster 1 Minute

In Abbildung 9.18 zeigt sich der Verlauf der RMSE der Gesamtbewertung (Overall), der Baseline und der Durchschnitt der RMSE der vier Teilmodelle.

Bei einem Fenster von einer Minute beginnen die RMSE aller drei Modelle ebenfalls sehr hoch, verringern sich nach etwa 55 Ratings aber auf unter 1.

Die RMSE der Detailbewertung liegen fast immer über denen der Gesamtbewertung- und Baseline-Modelle. Die RMSE der Gesamt- und Baseline-Modelle schwanken dagegen. In 2632 von 4397 gemessenen RMSE ist der der Baseline besser. Nach den letzten Messdaten liegt der Baseline-RMSE bei 0,699 und damit unter dem der Gesamtbewertung (0,73). Dennoch lässt sich kein genauer Trend festellen, welcher der beiden Modelle tatsächlich besser ist.

[1] Das Tool ist im aktuellen SVN von Odysseus zu finden: `http://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Development+with+Odysseus`. Besucht am: 01.04.2016.

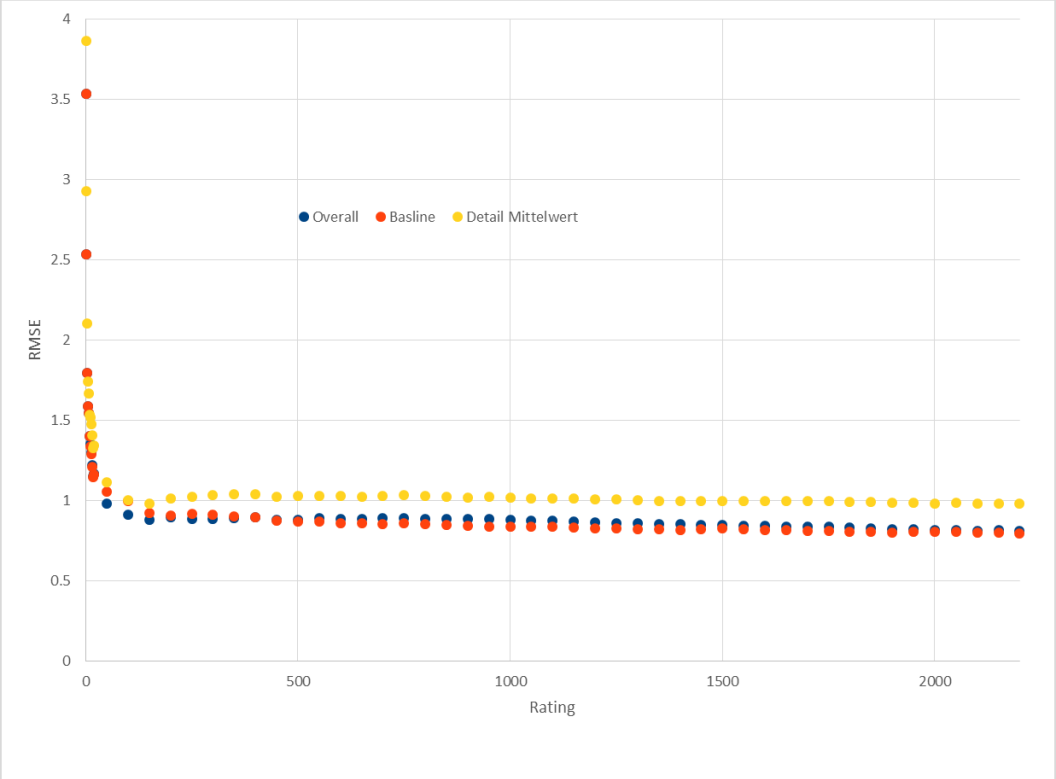


Abbildung 9.17: Verlauf des RMSE in Abhängigkeit der eingegangen Bewertungen ohne Berücksichtigung von Fenstern. (Wegen der Übersicht wird nur jeder 50. Datensatz angezeigt.) Quelle: Eigene.

In Abbildung 9.19 ist der Verlauf der RMSE der Teilmodelle der bewerteten Details zu sehen. Neben ebenfalls einer Startphase mit hohen RMSE, schwanken ab dem 50. Rating die RMSE zwischen 0,75 und 1,15. Ein Trend ist nicht zu erkennen. Die RMSE der einzelnen Teilmodelle unterscheiden sich nicht erkennbar.

9.3.2.3 Interpretation

Dass die RMSE bei Baseline, Gesamtbewertung und Detailbewertung anfänglich sehr hoch liegen, liegt an einem noch wenig trainierten Modell, das keine guten Bewertungen vorhersagen kann. Danach gleichen sich die Modelle annähernd an.

Der BRISMF.MOA ist so implementiert, dass bei einem noch untrainierten Modell nicht die Feature-Matrizen zur Berechnung des Predicted Ratings herangezogen werden. In diesem Fall wird der Mittelwert über alle abgegebenen Ratings berechnet und als Predicted Rating gesetzt. Dies trifft bei der Gesamtbewertung in den ersten 120 Ratings zu, danach wird das Predicted Rating über die Feature-Matrizen berechnet. Dies hat eine zusätzliche Analyse ergeben, ab wann die Predicted Ratings nicht mehr mit dem Mittelwert aus dem BRISMF.MOA-Modell übereinstimmen.

Dass der Wert zu Beginn so hoch liegt, liegt an der Berechnungssystematik. Das erste Rating wird mit einem völlig untrainierten Modell getestet. Demnach gibt das Modell ein Predicted Rating von

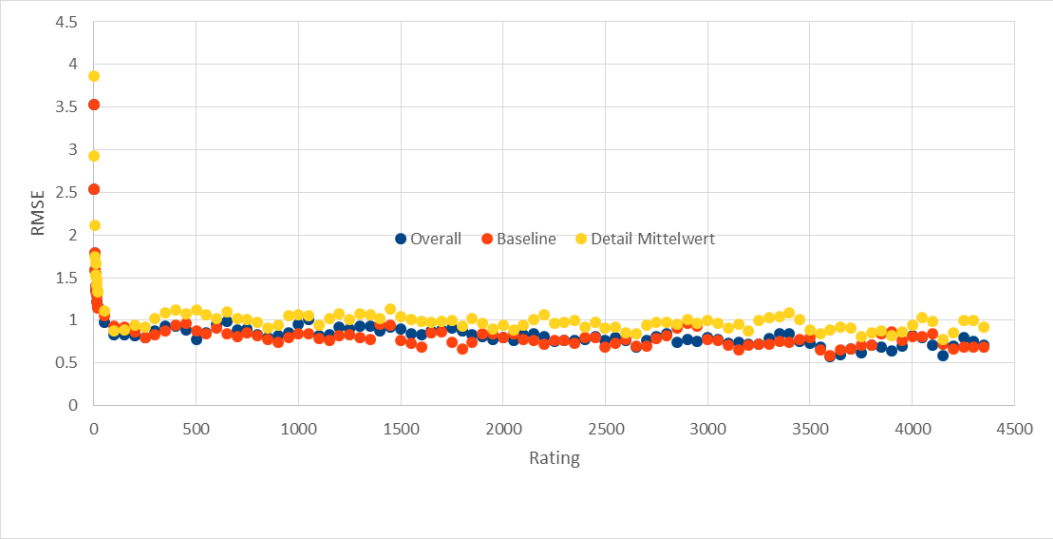


Abbildung 9.18: Verlauf des RMSE in Abhängigkeit der eingegangen Bewertungen mit einem Fenster von einer Minute. (Wegen der Übersicht wird nur jeder 50. Datensatz angezeigt.) Quelle: Eigene.

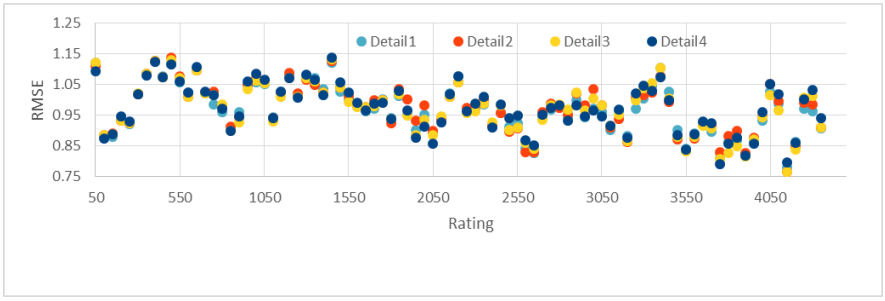


Abbildung 9.19: Verlauf des RMSE der Detail-Teilmodelle in Abhängigkeit der eingegangen Bewertungen mit einem Fenster von einer Minute. (Wegen der Übersicht wird nur jeder 50. Datensatz angezeigt, gestartet ab dem 50. Rating.) Quelle: Eigene.

0 aus, da der Mittelwert 0 beträgt (bei noch 0 eingeflossenen Ratings). Somit entspricht der erste gemessene RMSE exakt dem des ersten Ratings.

10 Projektabschluss

In diesem Kapitel wird eine Installationsanleitung des Systems bereitgestellt. Abschließend wird ein Ausblick über mögliche Erweiterungen und Funktionalitäten für die App und das RecSys gegeben.

10.1 Installationsanleitung

In diesem Abschnitt wird kurz erklärt, welche Schritte beachtet werden müssen, um das in diesem Dokument vorgestellte Produkt lauffähig zu machen. Die Anleitung gliedert sich dabei in folgende, wesentliche Teile: 1. Middleware und Odysseus Server und 2. die Android App.

10.1.1 Middleware und Odysseus Server

Auf der DVD enthaltene Datei mit der Dateiendung „.ova “ enthält eine virtuelle Maschine (VM), mit der der Odysseus-Server, die relationale Datenbank und die Middleware gestartet werden können. Die VM kann mit *VirtualBox*[1] gestartet werden. Dabei muss die VM so konfiguriert sein, dass die VM im Bridge-Modus gestartet wird. Die VM erhält somit eine eigene IP, und die App kann auf die Middleware zugreifen. Folgende Nutzerdaten hat das Betriebssystem:

- Nutzer: pgrecsys
- Passwort: pgrecsys

Mit dem Start der virtuellen Maschine startet auch die relationale Datenbank direkt mit. Die relationale Datenbank kann über die URL

http://localhost/phpmyadmin/

konfiguriert werden. Die Webanwendung *phpMyAdmin*[2] ist zur Konfiguration installiert. Zusätzlich ist diese URL auch als Lesezeichen im *Mozilla Firefox* abgespeichert. In der Datenbank *feature5* speichert die Middleware alle Informationen. Der Zugriff erfolgt über den Root-Account.

Im ersten Schritt muss der Odysseus Server gestartet werden: Der Odysseus-Server kann über den Link *Odysseus* auf dem Desktop gestartet werden. Der Workspace kann beibehalten werden. Im Workspace befinden sich zwei Projekte. Das Filter-Projekt ist enthält die Anfragepläne für die Gesamtbewertung und das Projekt Detailbewertung enthält die Anfragepläne für die Detailbewertung. Diese zwei Queries müssen immer gestartet werden.

In dem Filter-Projekt gibt es die Datei *Start_develop.qry* mit der alle Queries für die Gesamtbewertung gestartet werden können. Ebenso liegt eine solche Datei im Detailbewertungs-Projekt. Um die Queries der Detailbewertung zu starten muss die Datei *Start_develop_detail.qry* ausgeführt werden.

Nachdem diese zwei Dateien ausgeführt wurden, ist der Odysseus-Server vollständig initialisiert. Im nächsten Schritt kann nun der *Tomcat* über die Links auf dem Desktop gestartet werden. Dieser initialisiert die Middleware und das Dashboard. Die zwei Projekte sind nach ca. einer Minute initialisiert.

[1] https://www.virtualbox.org/ besucht am 06.04.2016.
[2] https://www.phpmyadmin.net/ besucht am 06.04.2016.

Es gibt eine Unterscheidung zwischen Detailbewertung und Gesamtbewertung in der Middleware. Die Methode kann nicht wie geplant über das Dashboard im Betrieb gewechselt werden. Daher gibt es zwei verschiedene Tomcats, die gestartet werden können. Wenn zwischen den zwei Methoden gewechselt werden soll, muss der laufende Tomcat zuerst heruntergefahren werden. Dieses kann auch über den zugehörigen Link auf dem Desktop ausgeführt werden. Es bietet sich an, danach auch den Odysseus-Server neuzustarten.

Nach diesen zwei Schritten sind die Middleware, das Dashboard und der Odysseus-Server initialisiert. Im letzten Schritt können nun Testdaten in das System eingespielt werden. Dies kann einfach über das Dashboard gestartet werden. Unter der URL

http://localhost:8080/Dashboard/

kann dies aufgerufen werden. Unter dem Reiter *Middleware Init* können Testnutzer, Testlocations und Testratings ins System eingespielt werden. Dies sollte in folgender Reihenfolge ausgeführt werden:

1. Testnutzer anlegen
2. Testfilterschema anlegen
3. Testlocations anlegen. Hierfür muss zuerst eine Datei mit Locations eingelesen werden. 115 Testlocations aus Oldenburg befinden sich in der Datei *locations.csv*, die auf dem Desktop liegt.
4. Initale Ratings

Alternativ kann auch die ganze Middleware über den ersten Punkt initialisiert werden. Hierbei werden jedoch keine Ratings ins System gespielt. Als Erweiterung zur minimalen Konfiguration können über das Dashboard weitere Spezial-Filter initialisiert werden. Wenn Testnutzer, Locations und Filterschema zusammen initialiert werden, sind diese Filter direkt enthalten.

Die Middleware sollte bei jedem Schritt eine Rückmeldung geben, dass die Testnutzer, Locations und Ratings erfolgreich initialiert worden sind. In einigen fällen gibt es einen Timeout, wenn die Initialisierung der Locations nicht erfolgreich war. Dieses tritt auf, wenn der Prozess zu lange dauert. In diesem Fall sollte über phpMyAdmin überprüft werden, ob alle Locations in die Datenbank gespeichert wurden. Die Anzahl der Testlocations beläuft sich auf 115.

Nach diesen Schritten ist es möglich, sich mit folgendem Account anzumelden:

- Nutzer: t
- Passwort: t

Alternativ kann auch ein neuer Nutzer registriert werden. Nachfolgend ist beschrieben, wie die App konfiguriert wird.

10.1.2 Android App

Für die Projektabgabe werden mehrere Build-Varianten der Android App, die für verschiedene Anwendungsszenarien ausgelegt sind, zur Verfügung gestellt. Für den Echtbetrieb ist die *release*-Variante

gedacht. Sie enthält alle für den Nutzer wichtigen Funktionen und erlaubt es dem Nutzer, nicht App-interne Schnittstellen zu konfigurieren.

Die *admin*-Variante enthält alle Funktionen der *release*-Variante und zusätzlich die Möglichkeit, die REST-Schnittstelle, die zur Server-Kommunikation verwendet werden soll, während der Laufzeit zu ändern. Dafür ist eine View eingerichtet, in der die wesentlichen Schnittstellen eingestellt werden können (siehe Abbildung 10.2). Diese ist sowohl vor dem Login über das Menü oben rechts oder nach dem Login jederzeit innerhalb des normalen App-Menüs erreichbar (siehe Abbildung 10.1).

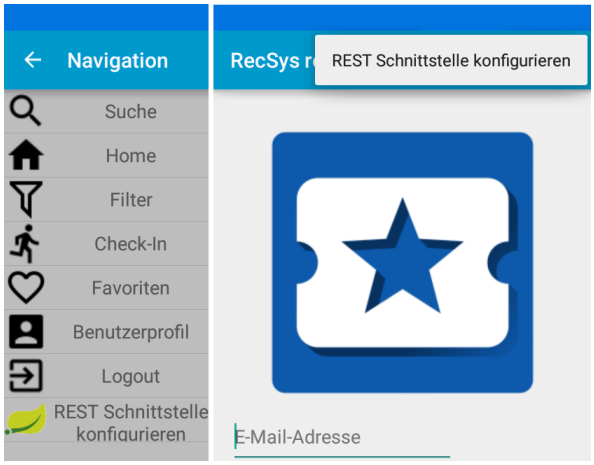


Abbildung 10.1: Menü-Führung zur REST-Config-Seite der App. Links: Menü nach Login. Rechts: Menü vor Login. Quelle: Eigene.

Die ersten drei Eingabefelder ermöglichen die Einstellung der verwendeten REST-Schnittstelle. Dabei können die drei Teile: IP, Port, REST-Path einzeln angegeben werden. Die daraus resultierende URL wird darunter angezeigt. Es ist wichtig, dass es mit “http://“ oder “https://“ beginnt und eine gültige URL ergibt. Die unteren beiden Eingabefelder beziehen sich auf das Verzeichnis, in dem sich Nutzerprofilbilder und Locationbilder befinden. Diese müssen vollständig angegeben werden, d.h. mit “http://“ beginnen und mit einem “/“ enden.

Die Änderungen werden erst übernommen, wenn die Eingabe durch das Drücken des “Speichern“-Buttons bestätigt wurde. Wird die View vorher verlassen, werden alle Änderungen verworfen.

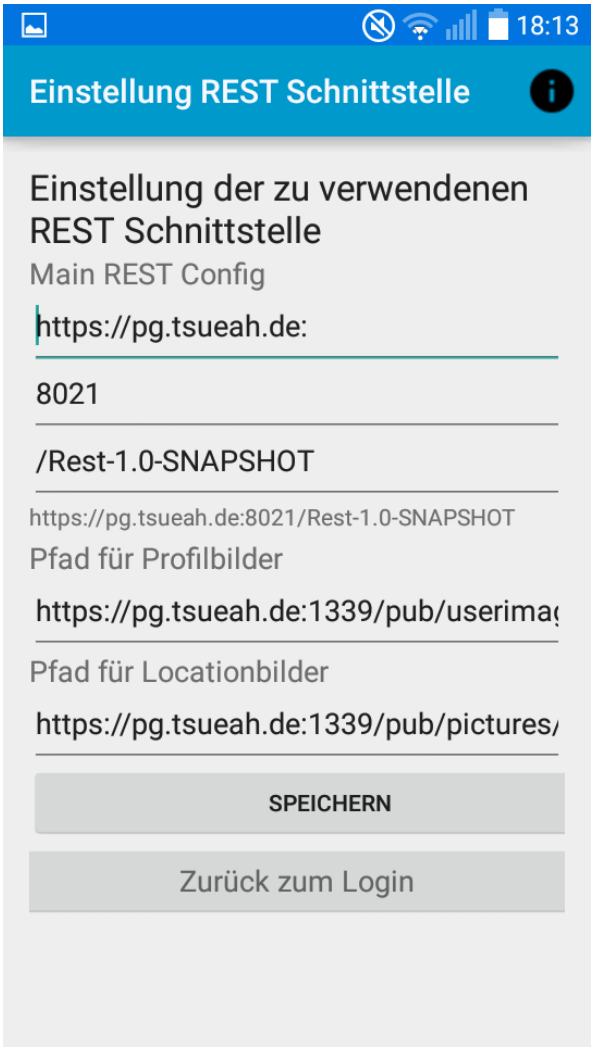


Abbildung 10.2: REST-Config-Seite der App. Quelle: Eigene.

Bei der hier zur Verfügung gestellten APK namens *pgrec-admin.apk* handelt es sich um die *admin*-Variante. Außerdem wird die *pgrec-release.apk* mit der *release*-Variante zur Verfügung gestellt. Diese funktioniert allerdings nur so lange, wie der während des Projekts genutzte Server noch aktiv ist.

10.2 Ausblick

Neben der vollständigen Implementierung derjenigen Anforderungen, die im Projektverlauf nicht umgesetzt werden konnten (siehe Kapitel 9.1), gibt es eine Reihe an weiteren Features und Ideen, die im Anschluss an das Projekt für zukünftige Arbeiten umgesetzt werden könnten. Diese Ansatzpunkte werden nachfolgend erläutert.

10.2.1 Evaluation mit realen Nutzern

Für die Bewertung der Qualität und vor allem der Akzeptanz des bisher implementierten Systems ist eine Evaluation 'im Feld', also mit realen Nutzern und unter echten Bedingungen notwendig. Dabei soll der Fokus auf die subjektive Qualität der Empfehlungen für den Nutzer liegen, aber auch auf Nutzerfreundlichkeit und Akzeptanz der grundlegenden Idee, Besuche anstatt Locations zu bewerten.

10.2.2 Externe Serviceprovider anbinden

Die derzeitige Datenbasis, die als Items für die Empfehlungsgenerierung dient, ist bisher von der Projektgruppe manuell eingetragen worden. Diese Daten beziehen sich lediglich auf den Raum Oldenburg. Die grundlegende Idee ist jedoch, die Informationen über externe Serviceprovider wie Yelp oder TripAdvisor anzufordern, damit diese über die Stadtgrenzen hinaus erweitert werden können, ohne selbst eine große Datenbank aufbauen zu müssen. Um das System skalierbar zu machen ist eine Umstellung auf die Serviceprovider notwendig.

10.2.3 Dashboard Erweiterung

Das zur Verfügung gestellte Dashboard enthält bereits die wichtigsten Funktionalitäten, kann aber noch beliebig erweitert werden. Beispielsweise wären eine Benutzerverwaltung oder die Möglichkeit zur Bearbeitung der Locations über das Dashboard eine sinnvolle Erweiterung der Funktionalitäten.

10.2.4 Location-Datenbank und Filterschema

Die Location-Datenbank (Item-Datenbank) wird derzeit in einem relationalen Datenbankmodell abgespeichert. Die Locationtabelle verfügt über eine Reihe von Stammdaten und darüber hinaus über zahlreiche Filterattribute, die über das Filterschema realisiert werden. Das implementierte Schema erlaubt es zwar, dem Admin weitere Filter zu erstellen und weitere Filterattribute für die Locations einzutragen, jedoch ist das implementierte System für Außenstehende nicht einfach verständlich umgesetzt. Ein ähnliches System könnte mit einem schemalosen *NoSQL*-Modell umgesetzt werden, da in diesem nur eine Location-Tabelle gebraucht werden würde, die einfach erweiterbar wäre.

10.2.5 Odysseus

Einer der wichtigsten Erweiterungspunkte betrifft das Recommender-System des DSMS Odysseus selbst. Die bisher eingesetzten Funktionen sind an einigen Stellen optimierbar. Unter anderem stellen folgende Features für den beschriebenen Anwendungsfall mögliche Ansatzpunkte zur Erweiterung:

10.2.5.1 Machine Learning für Detail-Recommender System

In dem derzeitig implementierten System für Detail-Ratings und Empfehlungen wird jedes Detail bei der Berechnung der Predicted Ratings gleich gewichtet und ein einfacher Durchschnitt für jedes Detail zu einer Gesamtempfehlung berechnet. Allerdings kann dieses Verfahren verbessert werden, wenn man die Gewichtung der unterschiedlichen Detail-Predicted-Ratings dem entsprechenden Nutzer anpasst.

Über Machine Learning-Verfahren kann das System lernen, welche Details für einen Nutzer besonders wichtig sind, indem sie die Abhängigkeit zwischen der Gesamtbewertung und die Bewertung der einzelnen Details beachtet. Ein solches Verfahren kann die individuellen Empfehlungen für den Nutzer und damit die Nutzererfahrung erheblich verbessern.

10.2.5.2 Modelbased Filtering

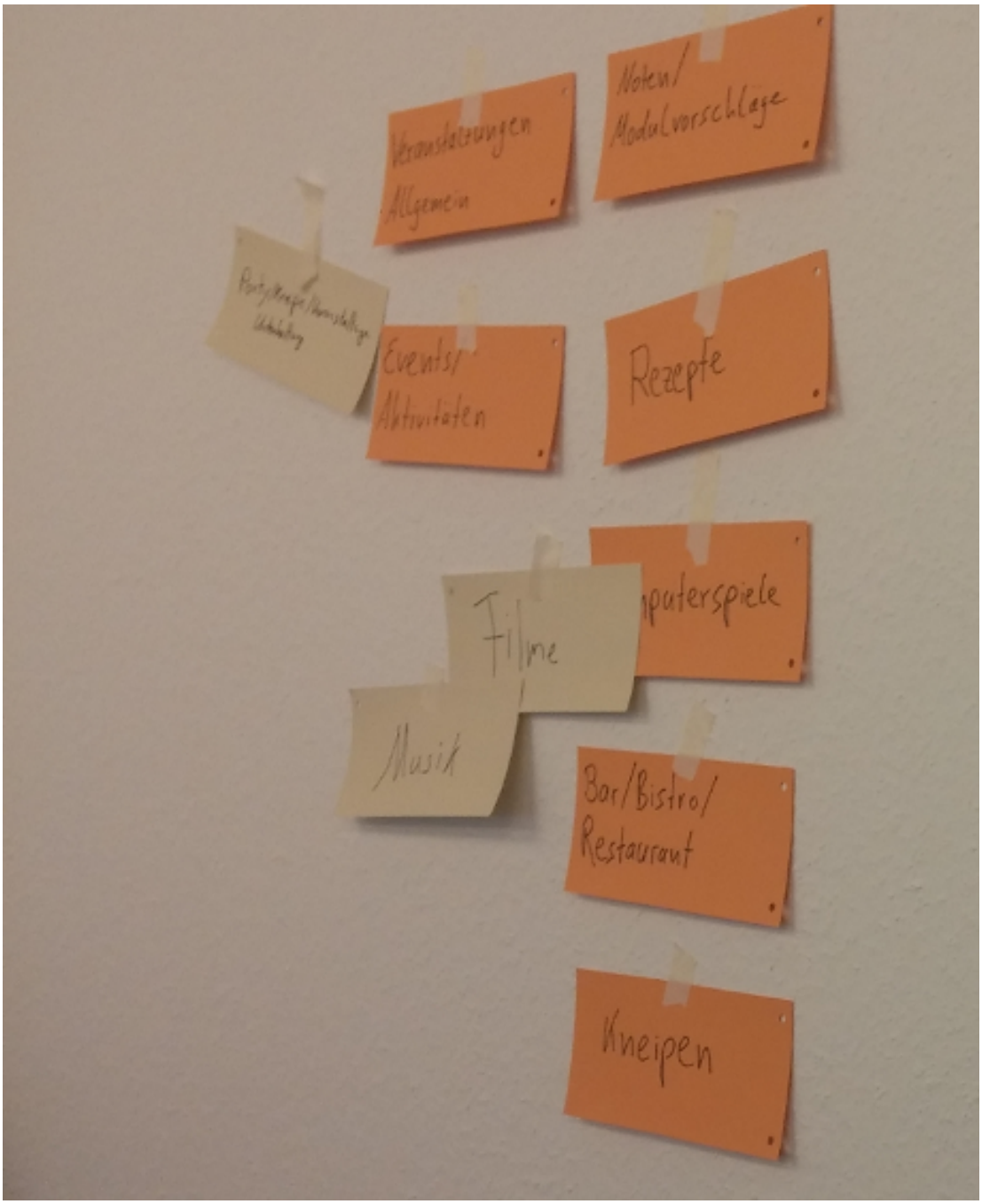
Da in dem implementierten System keine Locations, sondern Besuche von Locations bewertet werden, ist es sinnvoll, eine zeitliche Komponente bei der Bewertung des Nutzers in ein Modelbased Filtering-Verfahren zu integrieren und bei der Empfehlungsgebung zu beachten. Eine Anfrage, die an einem Mittwoch gestellt wird, sollte vorzugsweise Bewertungen in Betracht ziehen, die an einem Mittwoch abgegeben worden sind. Je nach Fenstergröße ist die vorhandene Datenbasis immer sehr aktuell und kann dem Nutzer dadurch bessere Ratings übermitteln und zu qualitativ hochwertigeren Bewertungen beitragen.

10.2.5.3 Concept Drifts

Durch die Entscheidung, das Recommender System mit einem DSMS umzusetzen, wird bereits zu einem gewissen Maß Concept Drift behandelt. Auf zeitliche Veränderungen oder Trends wird mithilfe des DSMS-typischen Fenster-Operators reagiert. Je nach Größe des Fensters können die Aktualität der Datensätze unterschiedlich gut ausfallen. Einen ähnlichen Einfluss hat die Grundidee des Szenarios, in dem Besuche von Locations bewertet werden und nicht Locations selbst. Der Nutzer hat dadurch die Möglichkeit zu einem Item mehr als eine Bewertung abzugeben, wodurch diese ebenfalls immer aktuell sind und sich automatisch auf aktuelle Entwicklungen anpassen. Alle Bewertungen, die von einem Nutzer getätigt werden, fließen komplett in seine Predicted Ratings ein. Ausreißer Bewertungen können vom System nicht erkannt und rausgefiltert werden. Dadurch können die berechneten Predicted-Ratings leicht verzerrt werden. Für einen optimierten Umgang mit Concept Drift müsste dieser Tatsache begegnet werden.

11 Anhang

11.1 Erarbeitete Bereiche des Ideen Workshops



11.2 Fragebogen des Projektleiter Interviews

Projekt: Datenstrombasierte Recommender Systeme - Sommersemester 2015 - Projektgruppe

PROJEKTLEITERBEFRAGUNG FÜR DIE ANFODERUNGSANALYSE

Interviewer: Christian Wiegert
Protokollant: Marius Wybranski
Ort: Oldenburg
Zeitraum:
Datum: 7.6.15

1. Können Sie Ihre Ideen noch einmal grob umreißen?

ShowCase Funktionalität darstellen
↳ Erweitern
Produkt vermarkten
Funktionalität von Odysseus vermarkten

2. Was verstehen Sie unter dem Begriff „personalisierte Empfehlung“?

Person zugeschnitten, nicht am meisten verkauft
Ratings, intelligente Empfehlungen
spezifisch

3. Haben Sie schon eine Idee, welche Informationen für einen Administrator wichtig sein könnten? Haben Sie schon Erfahrungen gemacht?

Wie gut sind die Empfehlungen.
Serverseitig und Client
Klic - Throw Rate,
Grafische Anzeige
Systemauslastung, Konfiguration
↳ Lernalgorithmus
↳ Vergleich von Algorithmen

4. Was verstehen Sie unter dem Begriff „Kontext-Reasoning“?

Situation einer Person ... verarbeiten von Kontextdaten zu Informationen.

Oberstes wirklich Kontext-Reasoning ist bleibt offen. ~~Wichtig~~ Informationen sollen Empfehlungen beeinflussen.

5. Haben Sie eine oder mehrere Funktionalitäten, die Sie gerne für den Administrator bereitstellen möchten?

siehe 3

6. Haben Sie eine oder mehrere Funktionalitäten, die Sie gerne für den Benutzer bereitstellen möchten?

Kern-Funktionen { Empfehlungen anzeigen
Bewerten, Ermitteln von Bewertungen

Wieso brauchen wir diese Informationen.
Zustimmung in der App.

Projekt: Datenstrombasierte Recommender Systeme - Sommersemester 2015 - Projektgruppe

7. Was verstehen Sie unter einer DSL? Domain Specific Language

Eigener Editor, Optional -> Wäre ganz wichtig,

8. Was sind Ihrer Meinung nach die wichtigsten Funktionalitäten eines Dashboards?

siehe 3

9. Gibt es die Möglichkeit sich die Administrations-Sicht bzw. Analyse-Sicht auf ein RecSys anzugucken?

gibt es schon so ein Dashboard?

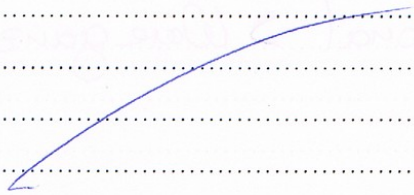
Keine Dashboard funktionalität extra

11. Welche Aufgaben hat aus ihrer Sicht der Administrator?

Eher an den Techniker nicht

Eher aus informatischer Sicht als ~~als~~ wissenschaftliche Sicht

12. Für welche Rolle sollen die Informationen auf dem Dashboard bereitgestellt werden?



13. Offene Frage: Was ist für dich wichtig

Benutzeranwendung (=> Odysseus
ShowCase
Comfort Funktionen zum Administrieren

14. Offene Frage: Datenschutz

AppStore wäre ganz schön.
Datenschutz steht für den Prototypen
nicht im Mittelpunkt.

11.3 Ergebnisse der Online-Umfrage

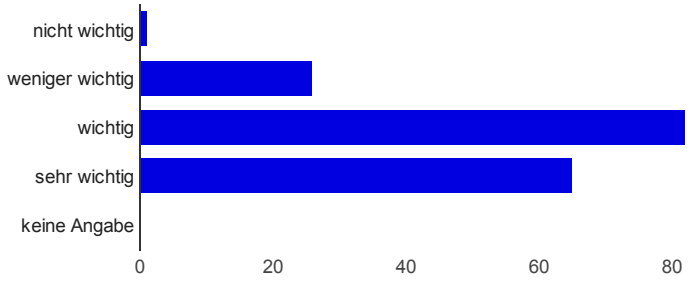
174 Antworten

Alle Antworten ansehen Analytics veröffentlichen

Zusammenfassung

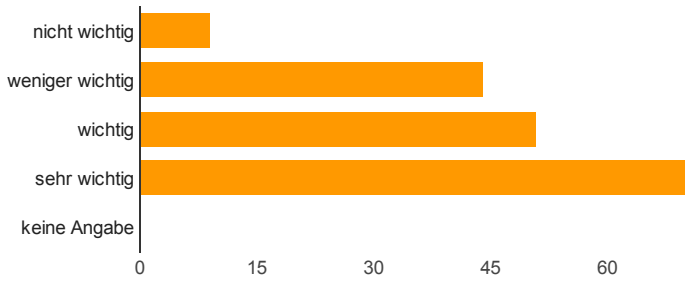
Bars und Kneipen

Preise [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



nicht wichtig	**1**	0.6 %
weniger wichtig	**26**	14.9 %
wichtig	**82**	47.1 %
sehr wichtig	**65**	37.4 %
keine Angabe	**0**	0 %

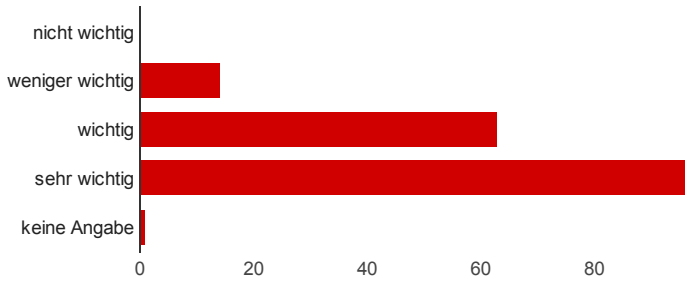
Qualität der Speisen [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



nicht wichtig	**9**	5.2 %
weniger wichtig	**44**	25.3 %

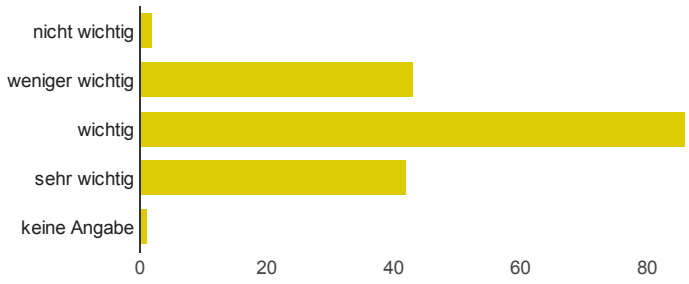
wichtig	**51**	29.3 %
sehr wichtig	**70**	40.2 %
keine Angabe	**0**	0 %

Qualität der Getränke [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



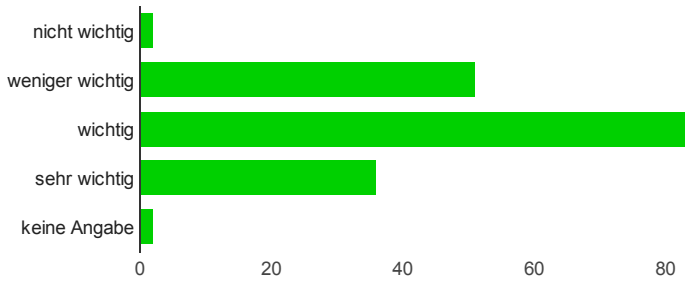
nicht wichtig	**0**	0 %
weniger wichtig	**14**	8 %
wichtig	**63**	36.2 %
sehr wichtig	**96**	55.2 %
keine Angabe	**1**	0.6 %

Auswahl (Speisen / Getränke) [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



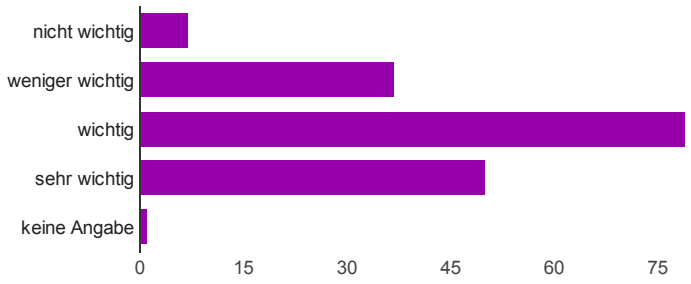
nicht wichtig	**2**	1.1 %
weniger wichtig	**43**	24.7 %
wichtig	**86**	49.4 %
sehr wichtig	**42**	24.1 %
keine Angabe	**1**	0.6 %

Entfernung vom aktuellen Standort [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



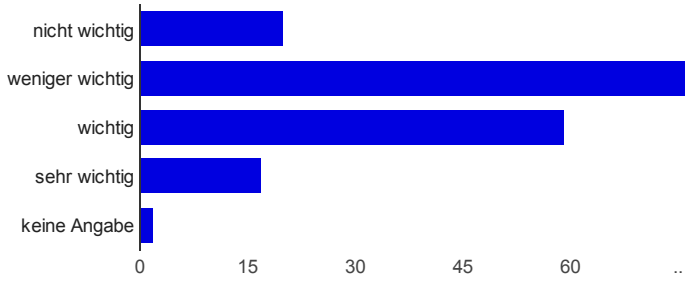
nicht wichtig	**2**	1.1 %
weniger wichtig	**51**	29.3 %
wichtig	**83**	47.7 %
sehr wichtig	**36**	20.7 %
keine Angabe	**2**	1.1 %

Musik [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



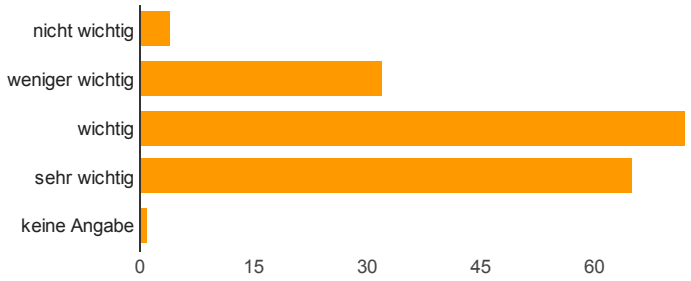
nicht wichtig	**7**	4 %
weniger wichtig	**37**	21.3 %
wichtig	**79**	45.4 %
sehr wichtig	**50**	28.7 %
keine Angabe	**1**	0.6 %

Außengastronomie (z.B. Biergarten) [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



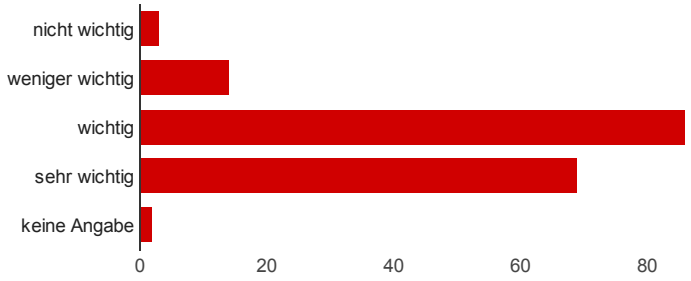
nicht wichtig	**20**	11.5 %
weniger wichtig	**76**	43.7 %
wichtig	**59**	33.9 %
sehr wichtig	**17**	9.8 %
keine Angabe	**2**	1.1 %

Service [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



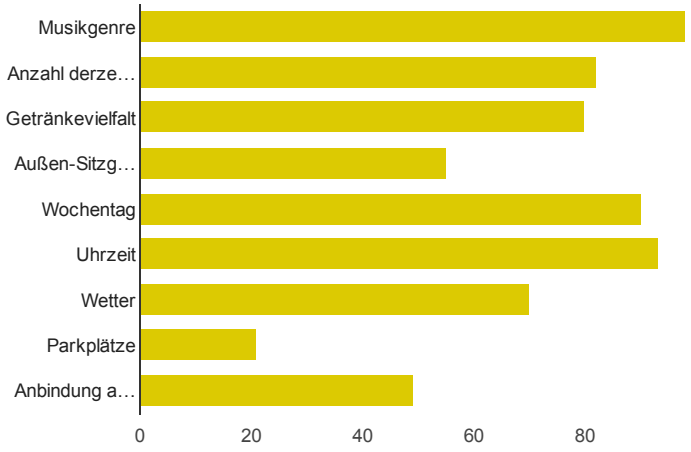
nicht wichtig	**4**	2.3 %
weniger wichtig	**32**	18.4 %
wichtig	**72**	41.4 %
sehr wichtig	**65**	37.4 %
keine Angabe	**1**	0.6 %

Ambiente [Welche Eigenschaften von Bars und Kneipen sind Ihnen wichtig?]



nicht wichtig	**3**	1.7 %
weniger wichtig	**14**	8 %
wichtig	**86**	49.4 %
sehr wichtig	**69**	39.7 %
keine Angabe	**2**	1.1 %

Stellen Sie sich vor, Sie wollen eine Bar oder Kneipe besuchen. Anhand welcher Kriterien wählen Sie die Bar oder Kneipe aus?



Musikgenre	**98**	56.6 %
Anzahl derzeitiger Gäste	**82**	47.4 %
Getränkevielfalt	**80**	46.2 %
Außen-Sitzgelegenheiten	**55**	31.8 %
Wochentag	**90**	52 %
Uhrzeit	**93**	53.8 %
Wetter	**70**	40.5 %
Parkplätze	**21**	12.1 %
Anbindung an den Öffentlichen Nahverkehr	**49**	28.3 %

Welche weiteren Kriterien sind Ihnen bei der Auswahl einer Kneipe oder Bar wichtig?

Heiße Kellnerinnen

Menschen

Service

vegane Gerichte/Snacks (oder zumindest vegetarisch)

happy hour

Bekanntheitsgrad

keine

Reputation, Gewohnheit

Preis

Rücksicht für diejenigen mit Intolleranzen

Preise

Anzahl Freunde / Bekannte die aktuell dort sind

Programm

wo Freunde hingehen, gehe ich auch mit

Klimatisierung/Luftqualität; Sauberkeit, typisches Gästemileu (Image)

Bekannte anwesend

Geile Dienstmädchen mir tiefen Ausschnitt

Entfernung zum aktuellen Standort

Ob sie die Spiele vom SC Paderborn zeigen

Licht (bspw offen geschnitten oder abgedunkelt)

Preis/Leistung

Guter / netter Service, Ambiente

Preis / Leistung

Empfehlungen von Freunden

location, interior, people

günstige preise

Raucher/ Nicht-Raucher, Spielgelegenheiten (Dart, Kicker usw.), Tanzmöglichkeiten, Preis- Leistung

Lautstärke (durch andere Gäste). Will ich ruhig und entspannt mit meiner Freundin sitzen, soll es leise sein. Bin ich mit einer Gruppe unterwegs ist es mir egal.

Alter/Erscheinungsbild/Auftreten der anderen Gäste

-Atmosphäre

Größe der Location

Raucher/Nichtraucher

Geräuschpegel darf nicht zu hoch sein

sanitäre Anlagen

Musik meiner lieblings-Genre, Wenig Personen pro Quadratmeter, Gute Getränke

nicht die vielfalt der getränke, sondern die qualität(kälte und kohlensäure) ist wichtig. Niemand mag warmes abgestandenes bier. Klima in der kneipe. Ambiente, gemütlichkeit, lautstärke, akustik,

Gemütlichkeit

Wünsche des Freundeskreises

Möglichkeiten für Gruppenplätze

Preise und Einrichtung

Ambiente (wie wirkt es von außen/innen)

Schocken spielen

Das Aussehen der Bedienung

Ambiente

gutes Essen

Lage, "wohlfühl-faktor'

Positiver lokaler Bekanntheitsgrad

Stimmung

Ob's schmeckt

AMbiente/Stil der Kneipe oder Bar

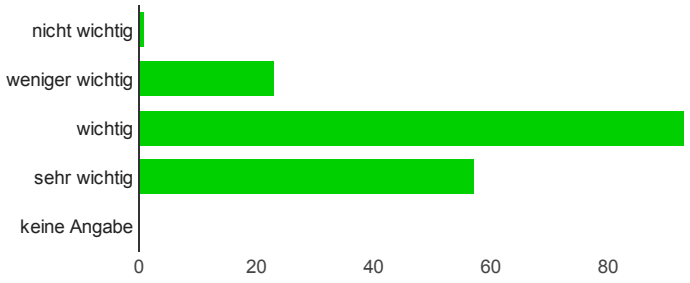
Preise der Getränke

Bewertung

Die oben genannten "wichtigen", Raucher/Nichtraucher, Lärmpegel,

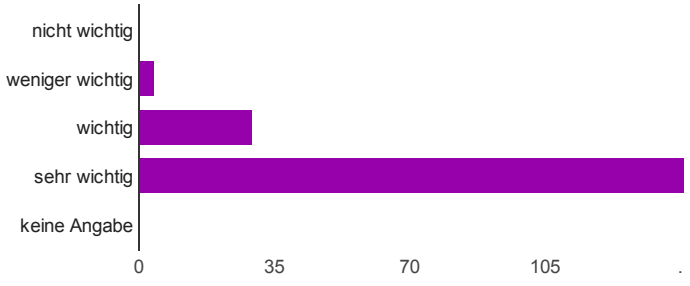
Restaurants und Lokale

Preise [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



nicht wichtig	**1**	0.6 %
weniger wichtig	**23**	13.2 %
wichtig	**93**	53.4 %
sehr wichtig	**57**	32.8 %
keine Angabe	**0**	0 %

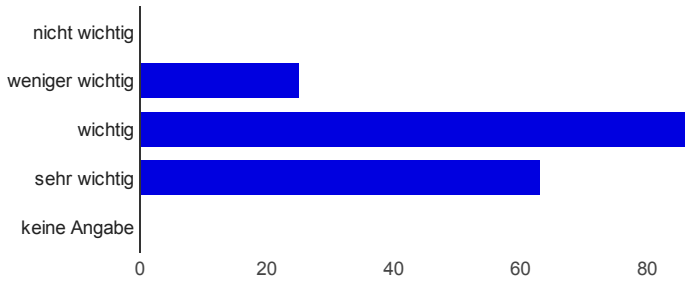
Qualität der Speisen [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



nicht wichtig	**0**	0 %
weniger wichtig	**4**	2.3 %

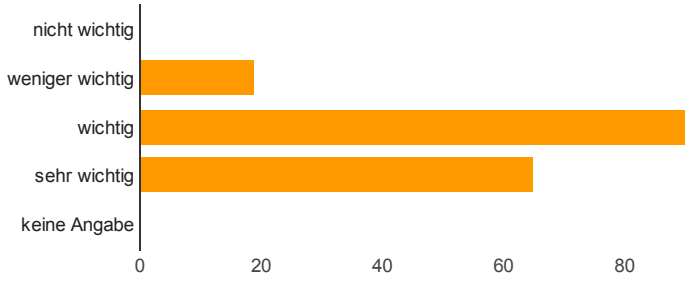
wichtig	**29**	16.7 %
sehr wichtig	**141**	81 %
keine Angabe	**0**	0 %

Qualität der Getränke [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



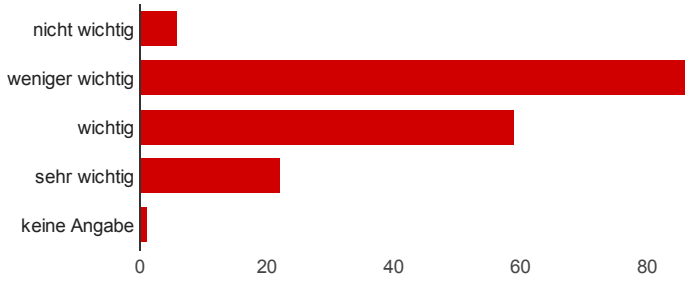
nicht wichtig	**0**	0 %
weniger wichtig	**25**	14.4 %
wichtig	**86**	49.4 %
sehr wichtig	**63**	36.2 %
keine Angabe	**0**	0 %

Auswahl (Speisen / Getränke) [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



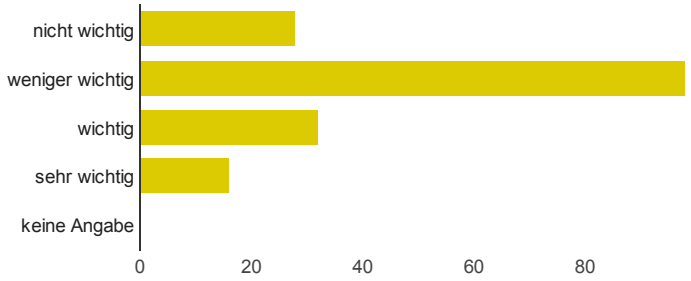
nicht wichtig	**0**	0 %
weniger wichtig	**19**	10.9 %
wichtig	**90**	51.7 %
sehr wichtig	**65**	37.4 %
keine Angabe	**0**	0 %

Entfernung vom aktuellen Standort [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



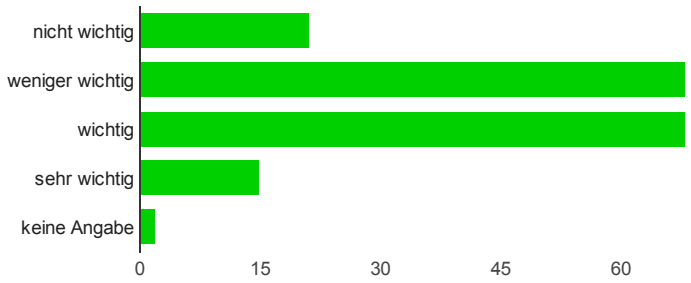
nicht wichtig	**6**	3.4 %
weniger wichtig	**86**	49.4 %
wichtig	**59**	33.9 %
sehr wichtig	**22**	12.6 %
keine Angabe	**1**	0.6 %

Musik [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]

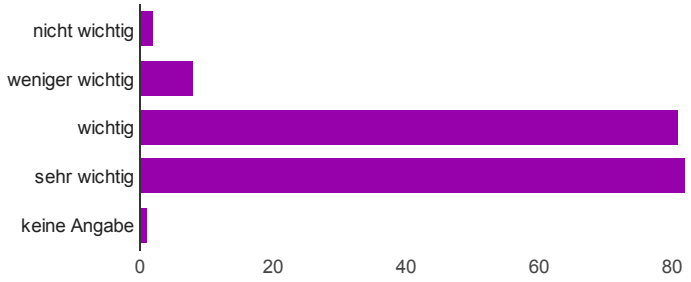


nicht wichtig	**28**	16.1 %
weniger wichtig	**98**	56.3 %
wichtig	**32**	18.4 %
sehr wichtig	**16**	9.2 %
keine Angabe	**0**	0 %

Außengastronomie (z.B. Biergarten) [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]

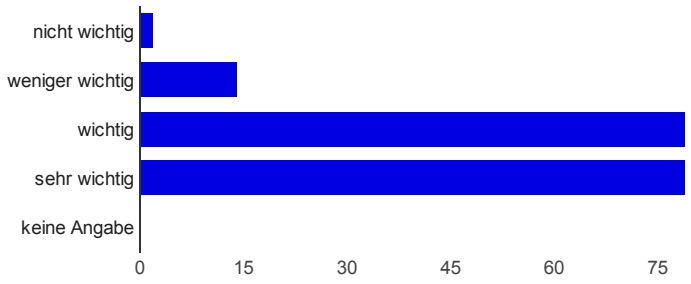


Service [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



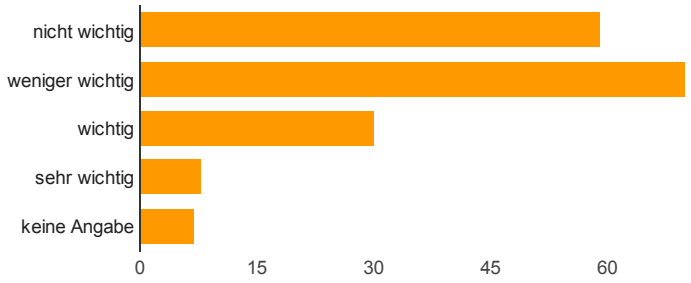
nicht wichtig	**2**	1.1 %
weniger wichtig	**8**	4.6 %
wichtig	**81**	46.6 %
sehr wichtig	**82**	47.1 %
keine Angabe	**1**	0.6 %

Ambiente [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



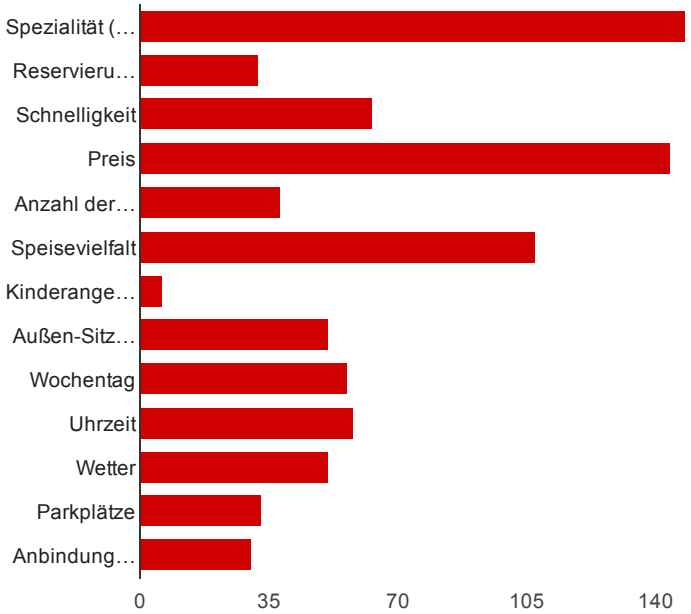
nicht wichtig	**2**	1.1 %
weniger wichtig	**14**	8 %
wichtig	**79**	45.4 %
sehr wichtig	**79**	45.4 %
keine Angabe	**0**	0 %

Nationalität [Welche Eigenschaften von Restaurants und Lokalen sind Ihnen wichtig?]



nicht wichtig	**59**	33.9 %
weniger wichtig	**70**	40.2 %
wichtig	**30**	17.2 %
sehr wichtig	**8**	4.6 %
keine Angabe	**7**	4 %

Stellen Sie sich vor, Sie wollen ein Restaurant oder Lokal besuchen. Anhand welcher Kriterien treffen Sie Ihre Auswahl?



Spezialität (Landesküche. Imbiss, Fischrestaurant, usw.) **148** 85.5 %

Reservierungsmöglichkeit	**32**	18.5 %
Schnelligkeit	**63**	36.4 %
Preis	**144**	83.2 %
Anzahl derzeitiger Gäste	**38**	22 %
Speisevielfalt	**107**	61.8 %
Kinderangebote	**6**	3.5 %
Außen-Sitzgelegenheiten	**51**	29.5 %
Wochentag	**56**	32.4 %
Uhrzeit	**58**	33.5 %
Wetter	**51**	29.5 %
Parkplätze	**33**	19.1 %
Anbindung an den Öffentlichen Nahverkehr	**30**	17.3 %

Welche weiteren Kriterien sind Ihnen bei der Auswahl eines Restaurants oder Lokals wichtig?

Service

Erfahrungsberichte

Das es schmeckt

Lokaler positiver Bekanntheitsgrad

Tagesangebote / Aktionen

keine

specials, set menu, location

Geile Bedienung

siehe Angaben zu Bars/Kneipen

wie bei kneipen, vegetarisches Angebot

Ebenfalls die Lautstärke (siehe Bars/Kneipen)

vegetarische Gerichte

Preis/Leistung

Die Kriterien ändern sich stets, je nach Gemütszustand, Appetit und verfügbarer Zeit

Bewertungen, ruf

Empfehlungen von Freunden

Qualität der Speisen

Sauberkeit

Empfehlungen von Bekannten/Freunden

Privatsphäre

Speisevielfalt in Bezug auf Vegan, Vegetarisch, Flexibilität des Restaurants (Kundenwünsche umsetzen)

Bewertungen

Ambiente

Sauberkeit, Hygienestandard (va. auch WC-Bereich)

Gutscheine (z.B. 2 Essen für den Preis von einem)

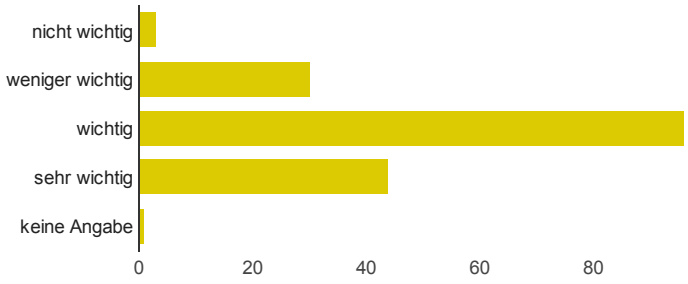
Momentane Lust (Lust auf Chinesisch,...)

Keine

Ambiente, Service

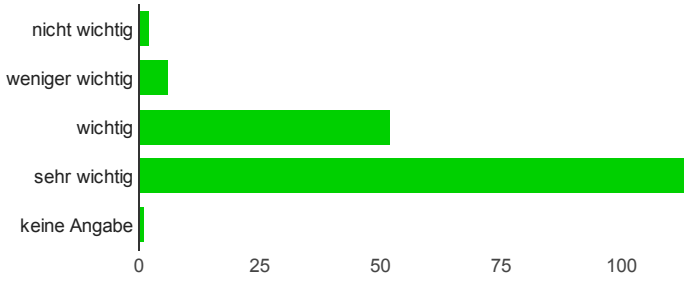
Cafés und Eisdielen

Preise [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



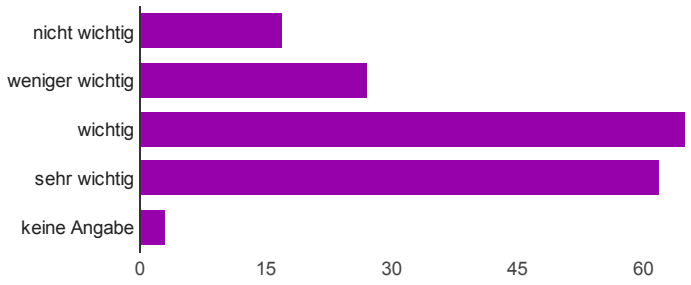
nicht wichtig	**3**	1.7 %
weniger wichtig	**30**	17.2 %
wichtig	**96**	55.2 %
sehr wichtig	**44**	25.3 %
keine Angabe	**1**	0.6 %

Qualität der Speisen [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



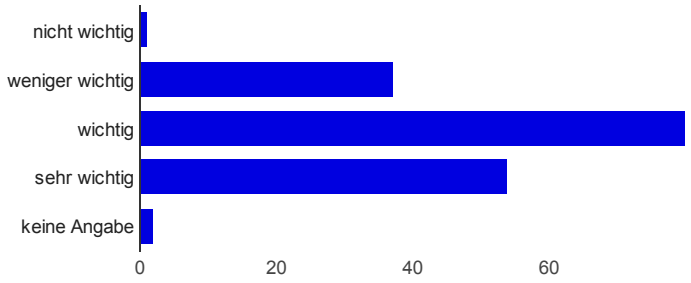
nicht wichtig	**2**	1.1 %
weniger wichtig	**6**	3.4 %
wichtig	**52**	29.9 %
sehr wichtig	**113**	64.9 %
keine Angabe	**1**	0.6 %

Qualität der Getränke [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



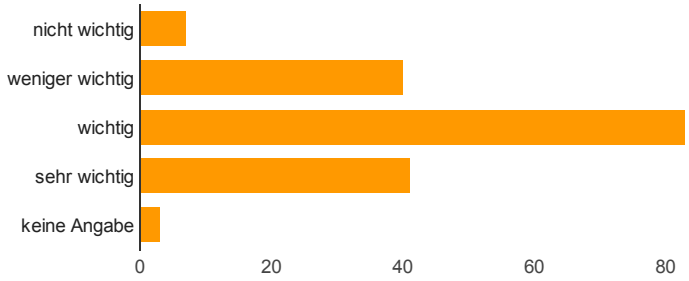
nicht wichtig	**17**	9.8 %
weniger wichtig	**27**	15.5 %
wichtig	**65**	37.4 %
sehr wichtig	**62**	35.6 %
keine Angabe	**3**	1.7 %

Auswahl (Speisen / Getränke) [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



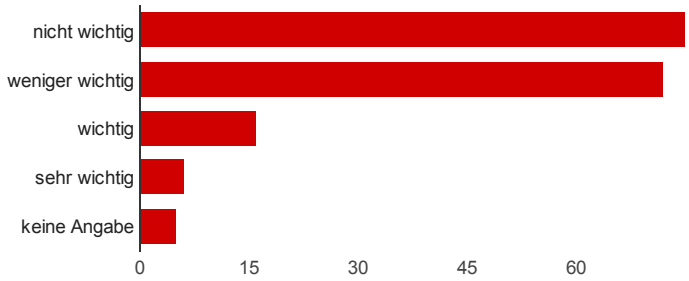
nicht wichtig	**1**	0.6 %
weniger wichtig	**37**	21.3 %
wichtig	**80**	46 %
sehr wichtig	**54**	31 %
keine Angabe	**2**	1.1 %

Entfernung vom aktuellen Standort [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



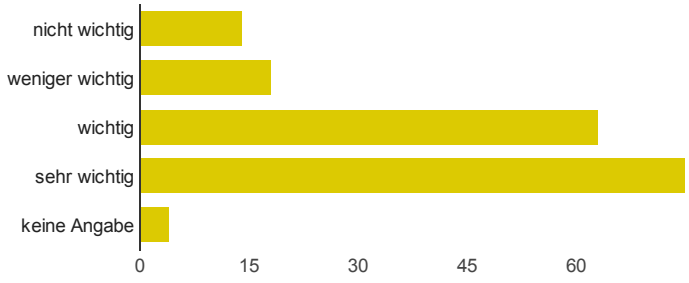
wichtig	**83**	47.7 %
sehr wichtig	**41**	23.6 %
keine Angabe	**3**	1.7 %

Musik [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



nicht wichtig	**75**	43.1 %
weniger wichtig	**72**	41.4 %
wichtig	**16**	9.2 %
sehr wichtig	**6**	3.4 %
keine Angabe	**5**	2.9 %

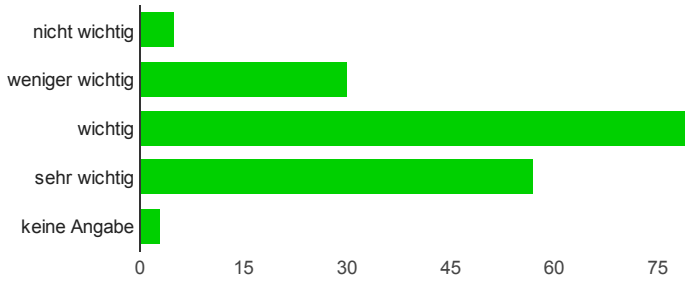
Außengastronomie (z.B. Biergarten) [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



nicht wichtig	**14**	8 %
weniger wichtig	**18**	10.3 %

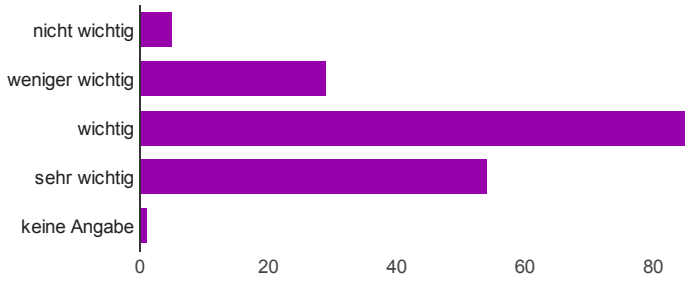
wichtig	**63**	36.2 %
sehr wichtig	**75**	43.1 %
keine Angabe	**4**	2.3 %

Service [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



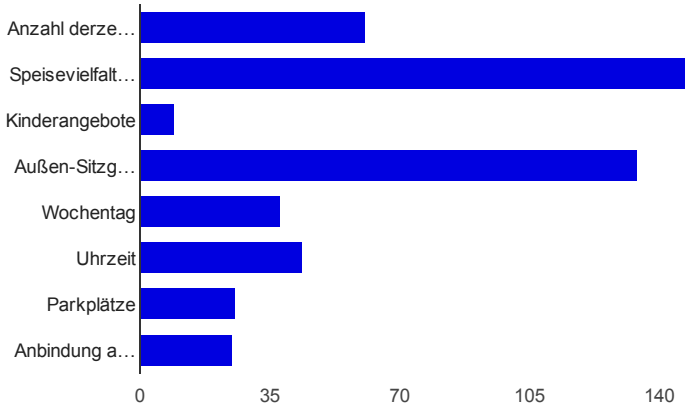
nicht wichtig	**5**	2.9 %
weniger wichtig	**30**	17.2 %
wichtig	**79**	45.4 %
sehr wichtig	**57**	32.8 %
keine Angabe	**3**	1.7 %

Ambiente [Welche Eigenschaften von Cafés und Eisdielen sind Ihnen wichtig?]



nicht wichtig	**5**	2.9 %
weniger wichtig	**29**	16.7 %
wichtig	**85**	48.9 %
sehr wichtig	**54**	31 %
keine Angabe	**1**	0.6 %

Stellen Sie sich vor, Sie wollen ein Café oder eine Eisdiele besuchen. Anhand welcher Kriterien treffen Sie Ihre Auswahl?



Anzahl derzeitiger Gäste	**61**	35.5 %
Speisevielfalt bzw. Eisauswahl	**147**	85.5 %
Kinderangebote	**9**	5.2 %
Außen-Sitzgelegenheiten	**134**	77.9 %
Wochentag	**38**	22.1 %
Uhrzeit	**44**	25.6 %
Parkplätze	**26**	15.1 %
Anbindung an den Öffentlichen Nahverkehr	**25**	14.5 %

Welche weiteren Kriterien sind Ihnen bei der Auswahl eines Cafés oder einer Eisdiele wichtig?

Service

wie bei Kneipen, warum sollte das variieren?

keine

merklich selbst hergestelltes Eis

freundliches Personal

Dicke Busen

Hochwertiger Tee ist wichtig.

keine Angabe

Entfernung

Laktosefreies Eis/ Sorbet

Sauberkeit

serieusität

versch. Angebote (Intolleranzen/Vegan)

Wetter

Ambiente

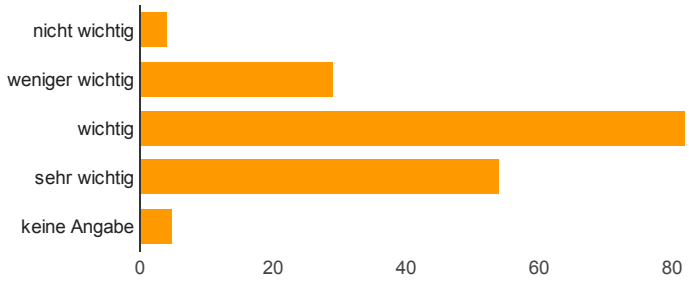
Bewertung

lactose free ice cream availability

Bewertung

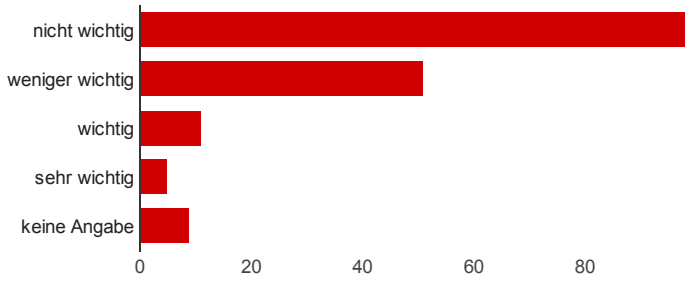
Discos und Clubs

Preise [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]



nicht wichtig	**4**	2.3 %
weniger wichtig	**29**	16.7 %
wichtig	**82**	47.1 %
sehr wichtig	**54**	31 %
keine Angabe	**5**	2.9 %

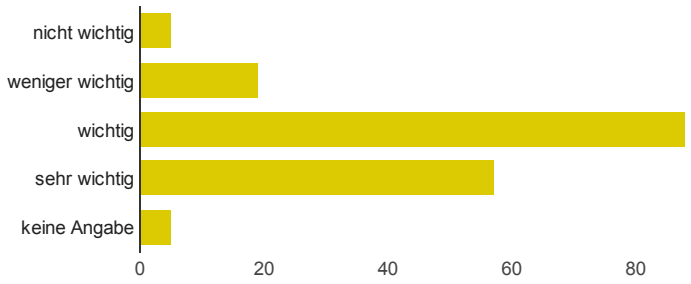
Qualität der Speisen [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]



nicht wichtig	**98**	56.3 %
weniger wichtig	**51**	29.3 %
wichtig	**11**	6.3 %
sehr wichtig	**5**	2.9 %
keine Angabe	**9**	5.2 %

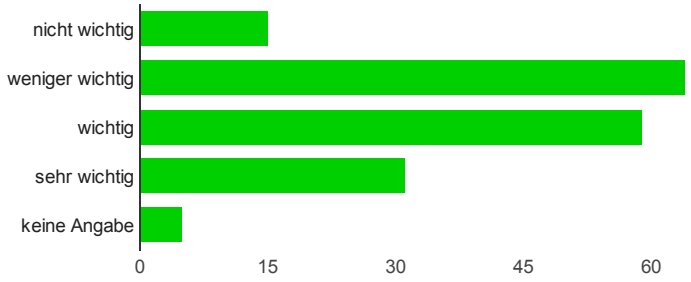
Qualität der Getränke [Welche Eigenschaften von Discos und Clubs sind

Ihnen besonders wichtig?]



nicht wichtig	**5**	2.9 %
weniger wichtig	**19**	10.9 %
wichtig	**88**	50.6 %
sehr wichtig	**57**	32.8 %
keine Angabe	**5**	2.9 %

Auswahl (Speisen / Getränke) [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]

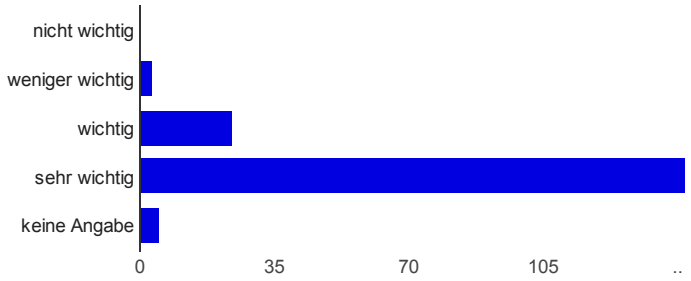


nicht wichtig	**15**	8.6 %
weniger wichtig	**64**	36.8 %
wichtig	**59**	33.9 %
sehr wichtig	**31**	17.8 %
keine Angabe	**5**	2.9 %

Entfernung vom aktuellen Standort [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]

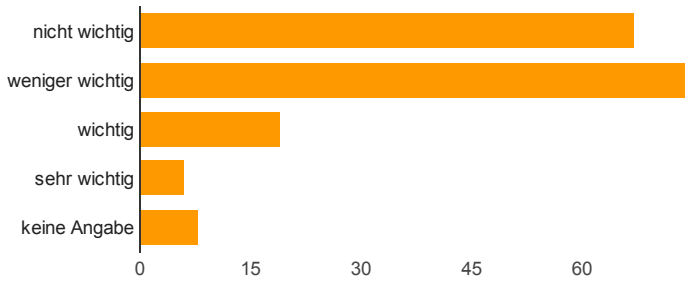
keine Angabe	**6**	3.4 %

Musik [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]



nicht wichtig	**0**	0 %
weniger wichtig	**3**	1.7 %
wichtig	**24**	13.8 %
sehr wichtig	**142**	81.6 %
keine Angabe	**5**	2.9 %

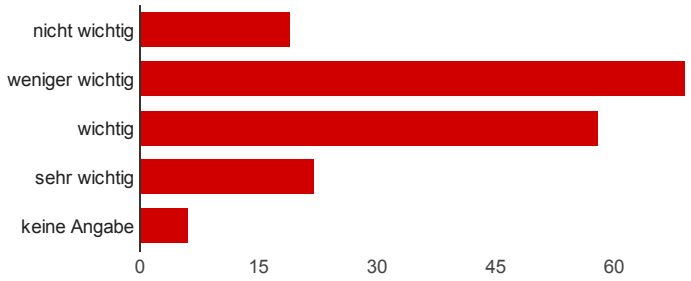
Außengastronomie (z.B. Biergarten) [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]



nicht wichtig	**67**	38.5 %
weniger wichtig	**74**	42.5 %
wichtig	**19**	10.9 %

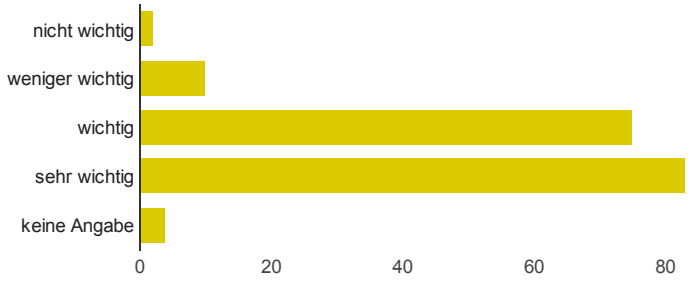
sehr wichtig	**6**	3.4 %
keine Angabe	**8**	4.6 %

Service [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]



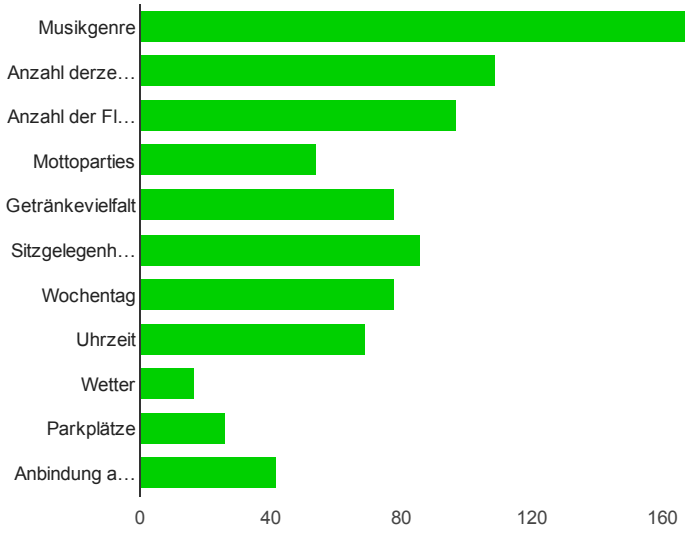
nicht wichtig	**19**	10.9 %
weniger wichtig	**69**	39.7 %
wichtig	**58**	33.3 %
sehr wichtig	**22**	12.6 %
keine Angabe	**6**	3.4 %

Ambiente [Welche Eigenschaften von Discos und Clubs sind Ihnen besonders wichtig?]



nicht wichtig	**2**	1.1 %
weniger wichtig	**10**	5.7 %
wichtig	**75**	43.1 %
sehr wichtig	**83**	47.7 %
keine Angabe	**4**	2.3 %

Welche Details sind Ihnen bei der Auswahl einer Disco oder eines Clubs wichtig?



Musikgenre	**167**	97.7 %
Anzahl derzeitiger Gäste	**109**	63.7 %
Anzahl der Floors bzw. Größe der Disco / des Clubs	**97**	56.7 %
Mottoparties	**54**	31.6 %
Getränkevielfalt	**78**	45.6 %
Sitzgelegenheiten	**86**	50.3 %
Wochentag	**78**	45.6 %
Uhrzeit	**69**	40.4 %
Wetter	**17**	9.9 %
Parkplätze	**26**	15.2 %
Anbindung an den Öffentlichen Nahverkehr	**42**	24.6 %

Welche weiteren Kriterien sind Ihnen bei der Auswahl einer Disco oder eines Clubs wichtig?

Lage

Freunde

Clubs in denen auch Livemusik gespielt wird bevorzugt

Wartezeiten beim Enitritt und der Bestellung

Kurze Röcke und große Brüste

Toiletten

Alter/Erscheinungsbild/Auftreten der anderen Gäste

Empfehlungen von Freunden

Musik

Raucher/Nichtraucher

speisen? who wants to eat in a club???

Ambiente

Bewertung

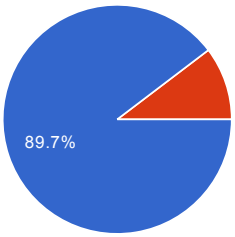
Live-Bands!

sind mir sowieso zu laut

Auftreten und Päsens der Service- sowie Sicherheitskräfte, abwechslungsreiche Musik (Themenabende), Spielgelegenheiten (Dart, Kicker..)

Applikation

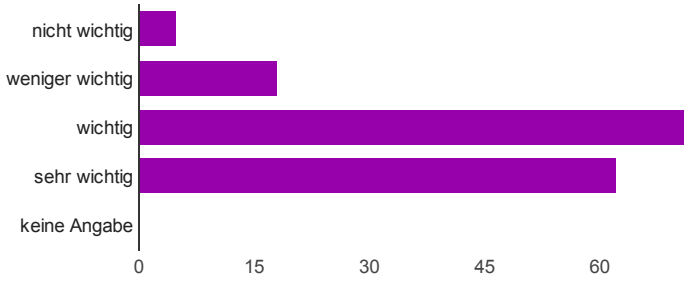
Stellen Sie sich vor, Sie möchten abends eine Bar oder einen Club besuchen, wissen jedoch noch nicht welchen. Könnten Sie sich vorstellen, eine App zu nutzen, die Ihnen basierend auf Ihren persönlichen Interessen eine Location empfiehlt?



ja	**156**	89.7 %
nein	**18**	10.3 %

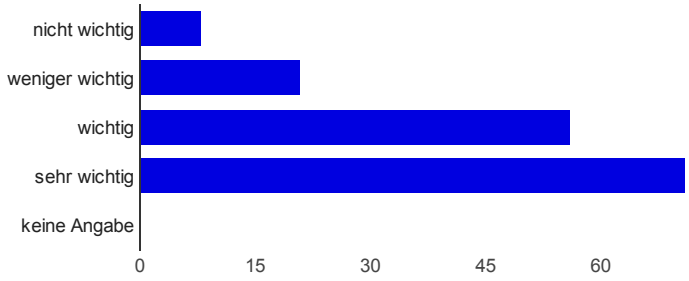
Weitere Fragen zur Applikation

Stadtplan anzeigen [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



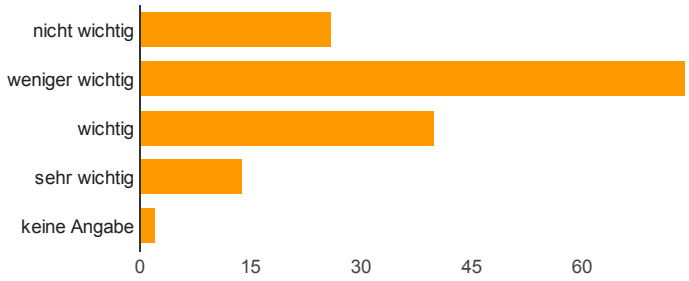
nicht wichtig	**5**	3.2 %
weniger wichtig	**18**	11.5 %
wichtig	**71**	45.5 %
sehr wichtig	**62**	39.7 %
keine Angabe	**0**	0 %

Routenplanung [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



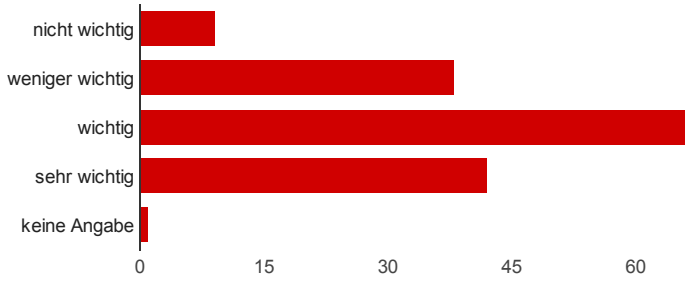
nicht wichtig	**8**	5.1 %
weniger wichtig	**21**	13.5 %
wichtig	**56**	35.9 %
sehr wichtig	**71**	45.5 %
keine Angabe	**0**	0 %

Austausch mit Freunden [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



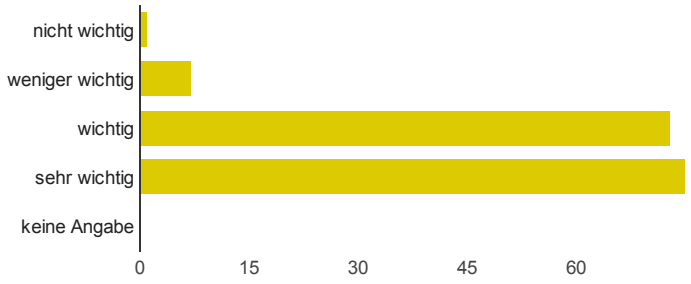
nicht wichtig	**26**	16.7 %
weniger wichtig	**74**	47.4 %
wichtig	**40**	25.6 %
sehr wichtig	**14**	9 %
keine Angabe	**2**	1.3 %

Kommentarfunktion / Reviews schreiben [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



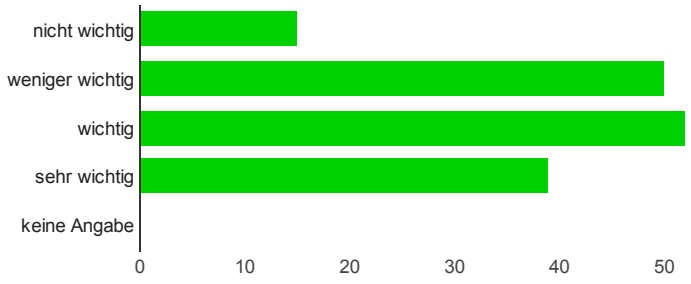
wichtig	**66**	42.3 %
sehr wichtig	**42**	26.9 %
keine Angabe	**1**	0.6 %

Übersicht aller Locations [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



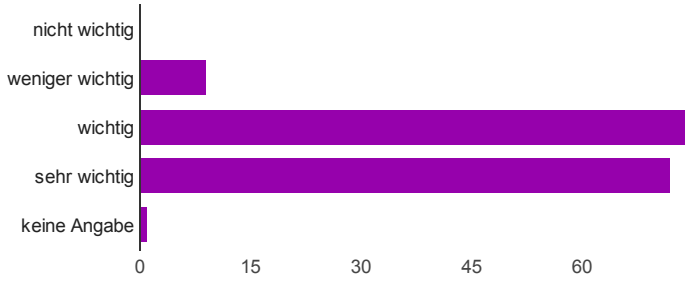
nicht wichtig	**1**	0.6 %
weniger wichtig	**7**	4.5 %
wichtig	**73**	46.8 %
sehr wichtig	**75**	48.1 %
keine Angabe	**0**	0 %

Anlegen von Favoriten [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



nicht wichtig	**15**	9.6 %
weniger wichtig	**50**	32.1 %
wichtig	**52**	33.3 %
sehr wichtig	**39**	25 %
keine Angabe	**0**	0 %

Umkreissuche von Locations [Wie wichtig wären Ihnen die folgenden Funktionen einer App, die Ihnen personalisierte Empfehlungen für Locations gibt?]



nicht wichtig	**0**	0 %
weniger wichtig	**9**	5.8 %
wichtig	**74**	47.4 %
sehr wichtig	**72**	46.2 %
keine Angabe	**1**	0.6 %

Welche weiteren Funktionen wären Ihnen wichtig?

Veranstaltungsübersicht

Event

Anzeige der Speise Karte in der App

Partybeschreibung z.B. Musikgenre, Getränkespecials

Events der verschiedenen locations

Bilder um sich ein eigenes bild zu machen.

Übersicht der Happy Hours

Aktuelle Events/Parties anzeigen

Simple Bewertungsfunktion wie Sterne oder Punkte

Informationen zu den Locations (Öffnungszeiten. ..)

neutrale, allgemeine Beschreibung der Location mit Bildern

Fotos der Location (Innenraum)

Anzeige von Live-Auftritten eines festgelegten Musik-Genres

Infos zu einzelnen Locations/Details

Bilder der location

Sparangebote durch App

Muschisensor

Vorschläge sollten auf persönliches Profil des Appnutzers abgestimmt sein

nur private Speicherung, Datenschutz

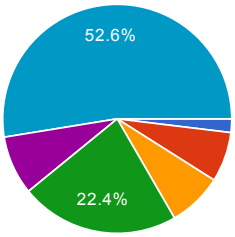
eine Übersicht über mein Profil anhand die Empfehlungen generiert werden und dass dieses angepasst werden kann

Nur wenn ich in der Stadt bin möchte ich aktuelle Trends berichtet bekommen.

speise- und getränkekarten

Teilen-Funktion

Würden Sie benachrichtigt werden wollen, wenn eine für Sie interessante Location in der Nähe ist?

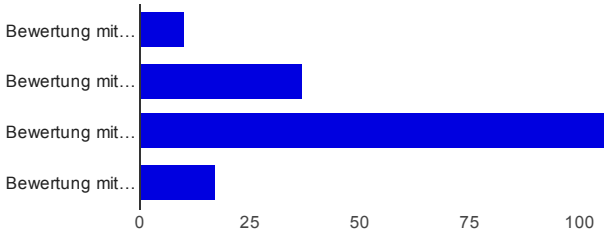


mehrmals täglich	**3**	1.9 %
einmal täglich	**11**	7.1 %
mehrmals pro Woche	**12**	7.7 %
einmal pro Woche	**35**	22.4 %
weniger als einmal pro Woche	**13**	8.3 %
ich möchte nicht benachrichtigt werden	**82**	52.6 %

Bewertung von Besuchen von Locations

Arten von Bewertungen

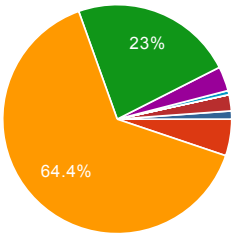
Welche Art der Bewertung wünschen Sie sich?



Daumen runter	**10**	6.4 %
uch insgesamt	**37**	23.7 %
nes Besuches	**124**	79.5 %
Bewertung mittels Skala	**17**	10.9 %

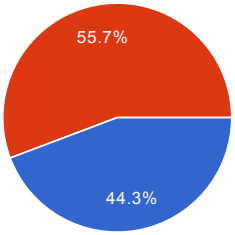
Persönliche Informationen

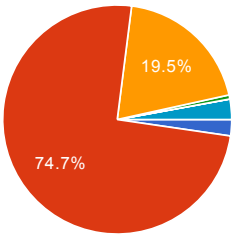
Alter



unter 15	**0**	0 %
15 - 19 Jahre	**9**	5.2 %
20 - 24 Jahre	**112**	64.4 %
25 - 29 Jahre	**40**	23 %
30 - 34 Jahre	**6**	3.4 %
35 - 39 Jahre	**1**	0.6 %
40 - 44 Jahre	**0**	0 %
45 - 49 Jahre	**0**	0 %
50 - 54 Jahre	**4**	2.3 %
55 - 59 Jahre	**2**	1.1 %
60 - 65 Jahre	**0**	0 %
über 65 Jahre	**0**	0 %

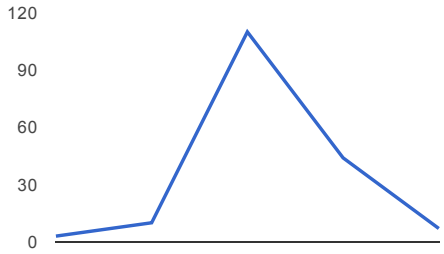
Geschlecht



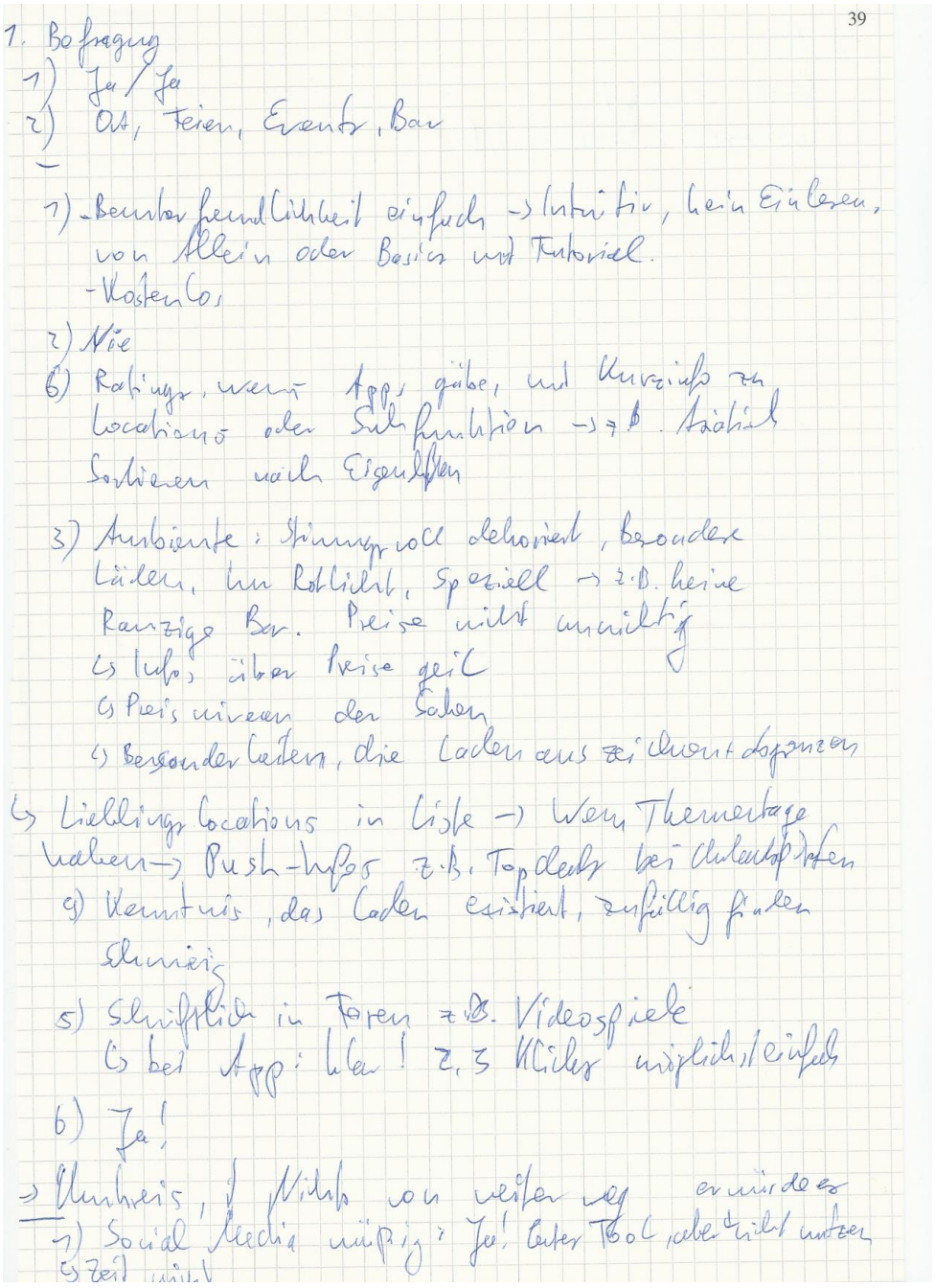


Schüler/in	**4**	2.3 %
Student/in	**130**	74.7 %
Angestellte/r / Beamte/r	**34**	19.5 %
Selbstständig/e	**1**	0.6 %
Rentner/in	**0**	0 %
sonstige	**5**	2.9 %

Anzahl der täglichen Antworten



11.4 Ergebnisse des Problemzentrierten Interviews



39

1. Befragung
1) Ja / Ja
2) Bar, Feiern, Events, Bar

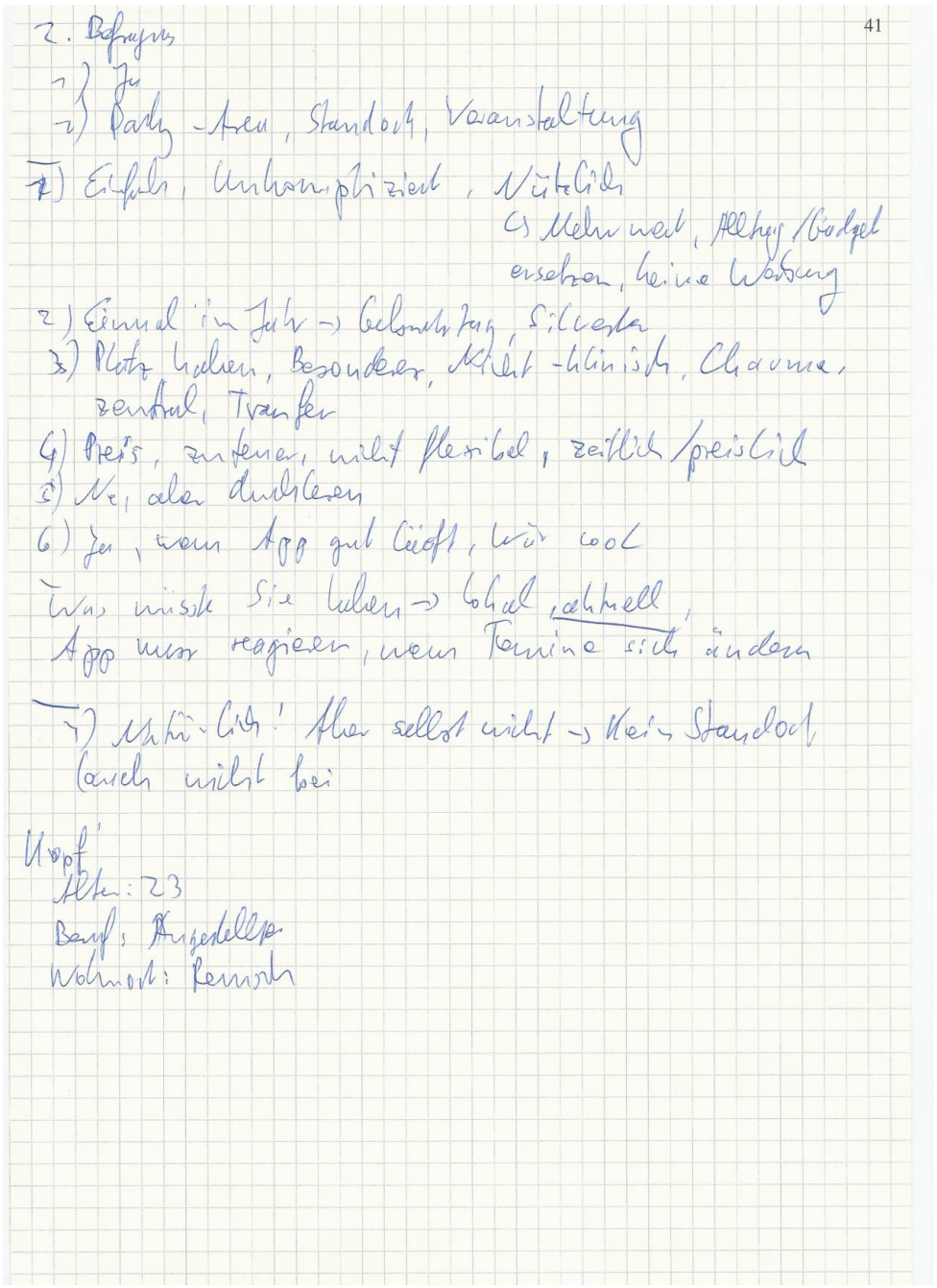
1) - Benutzerfreundlichkeit einfach → intuitiv, kein Einlesen, von Allein oder Basics und Tutorial.
- Kostenlos
2) Nie
3) Ratings, wenn App gäbe, und Kurzinfo zu Locations oder Subfunktion → z.B. Stadtteil
Sortieren nach Eigenschaften
3) Ambiente: Stimmung voll elementar, besonders Lieder, Ton Qualität, Speziell → z.B. keine Ramsiger Bar. Preise nicht unwichtig
↳ Info, über Preise geil
↳ Preis niveau der Bars
↳ Besonder Leute, die Leute aus einem Ort sprechen

↳ Lieblings Locations in Liste → wenn Thementage haben → Push-Infos z.B. Topdeals bei Clubnächten
4) Kenntnis, das Bars existiert, zufällig finden
Skurrig
5) Skriptlich in Foren z.B. Videospiele
↳ bei App: Idee! z.B. Killer möglich/einfach
6) Ja!

→ Skurrig, / Nicht so weiter weg er würde es
1) Social Media wichtig: Ja! Eher Text oder Bild unten
↳ Zeit wichtig

40

- Alter: 25
Beruf: Student
Wohnort: Remscheid



41

2. Befragung

1) Ja
2) Party-Area, Standort, Veranstaltung

3) Einfach, Unkompliziert, Nützlich
c) Mehrwert, Aufwand/Budget ersetzen, keine Werbung

2) Einmal im Jahr → Geburtstag, Silvester
3) Platz haben, Besonderer, Nicht-klinisch, Charme, zentral, Transfer
4) Preis, anderer, nicht flexibel, zeitlich/preislich
5) Nein, aber durchlesen
6) Ja, wenn App gut läuft, wär cool

Was müsste Sie laden → Schnell, aktuell, App muss reagieren, wenn Termine sich ändern

7) Nein. Nicht selbst nicht → Kein Standort, auch nicht bei

Kopf
Alter: 23
Beruf: Angestellter
Wohnort: Remscheid

43

3. Befragung

1) Ja / Ja
2) Umfeld, Stadt, Party, Sehenswürdigkeiten, Disko, Kneipen

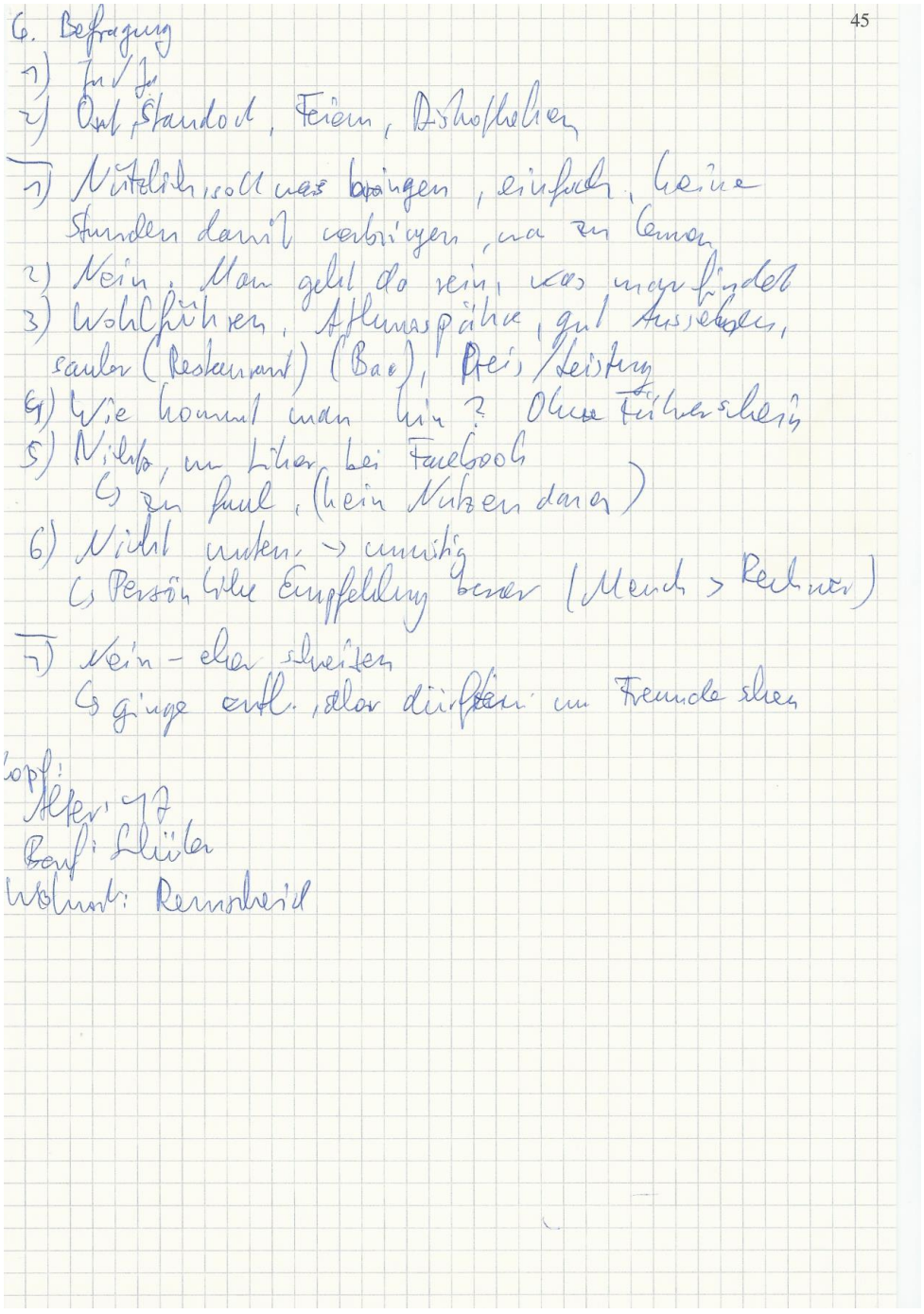
---

1) Funktion chillen, möglichst unkompliziert
keine Werbung ↳ schnell, Wetterapp →
2) Ja → Attraktivitäten, nicht weiter weg, Diskos, Kneipen, in Stadt gibt's wenig
↳ Wochenende / Ferien
3) Erreichbarkeit, ÖPNV, Preise, Angebote, Umfeld
Gäste sollten selbstständig werden, für sich sein können, keine Randale
4) Erreichbarkeit / ÖPNV, Alter einschränken,
5) Ja → nur bei Extrema. Aber selten. Aber Kommentar wichtig, weil kontaktieren kann.
Begründung, weshalb wenig Sterne z. B.
6) Auf jeden Fall! Bei Locations weiß man wo, was da läuft → Empfehlung sollte gut sein

---

7) Frage nach: Darf man Abstimmen
↳ Entscheidung soll bei Dir liegen!

Alter: 17
Beruf: Schüler
Wohnort: Wermelskirchen



6. Befragung 45

1) Ja / Ja
2) Ort, Standort, Feiern, Öffentlichkeit
3) Nützlich, soll was bringen, einfach, keine Stunden damit verbringen, um zu lernen
2) Nein, Man geht da rein, was man findet
3) Wohlfühlen, Atmosphäre, gut Aussehen, sauber (Restaurant) (Bar), Preis/Leistung
4) Wie kommt man hin? Ohne Führerschein
5) Nützlich, um Liker bei Facebook
   ↳ zu faul, (kein Nutzen daran)
6) Nicht unten → wichtig
   ↳ Persönliche Empfehlung besser (Mund > Radio)
7) Nein - eher streiten
   ↳ ginge evtl., aber diskutiere im Freundeskreis

Kopf:
Alter: 27
Beruf: Lehrer
Wohnort: Remscheid

47

5) Befragung

1) Ja/Ja

2) Wald, Schule, Turnhalle, Kino, Freibad, Diskotheken

---

1) ~~Fu~~ Funktioniert, neuester Stand, kein Datenabsturz. ~~oder~~ zumindest ehrlich sein. Versprechen muss stimmen!

2) Meistens weiß man es.
↳ Oder nach Empfehlungen von Freunden / Bekannten

3) Gastfreundlich, Ansprüche erfüllen, Essen muss so sein wie auf Bildern, Sauberkeit, ruhige Atmosphäre oder der Location entsprechende Atmosphäre

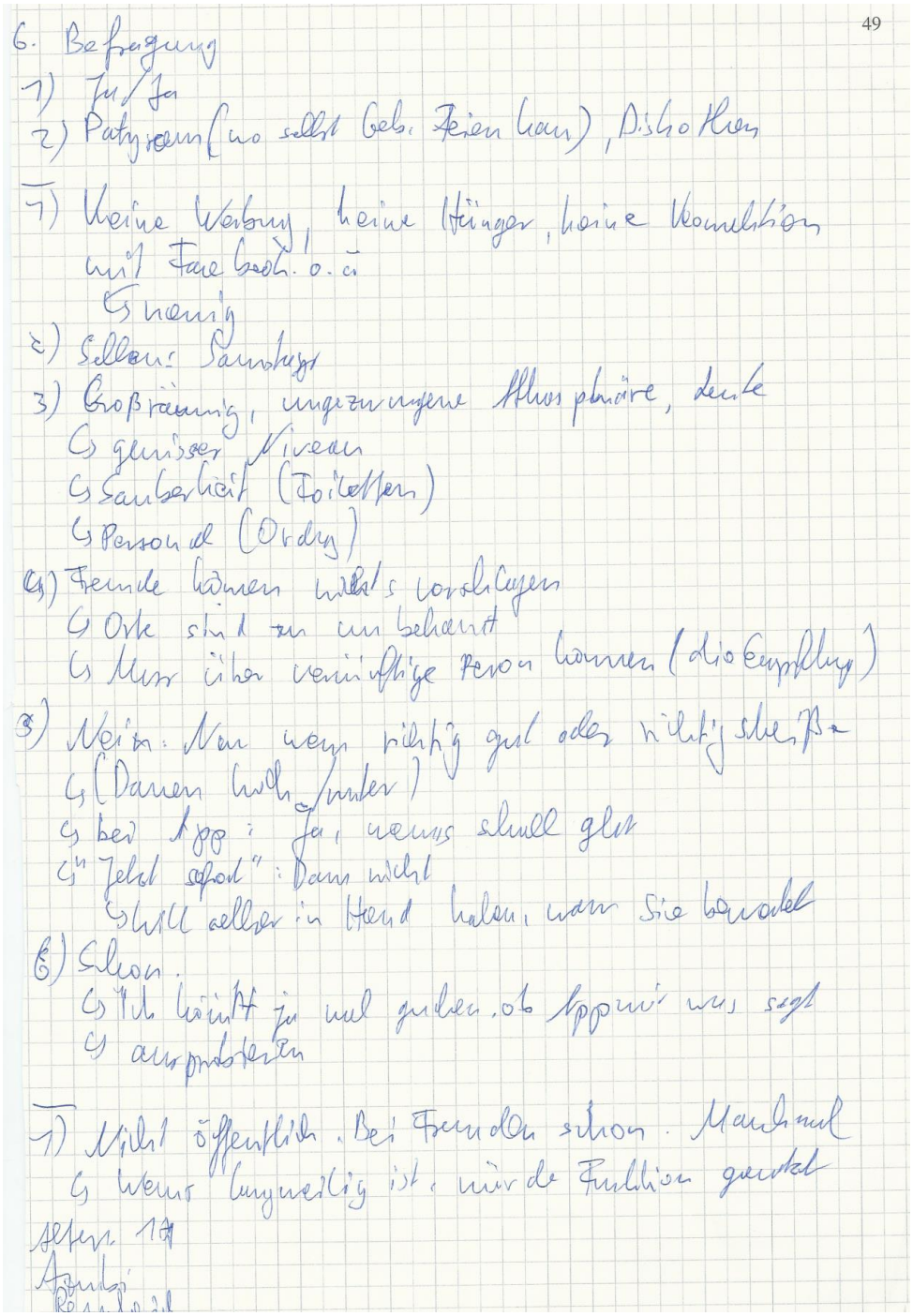
4) Gibt viele Angebote, man muss selbst entscheiden, was das Beste ist so, oder Maps funzt nicht

5) Nein. Aber ist hilfreich. Selbst zu lästig
↳ Persönliche Daten angeben schlecht
↳ Selber nicht → soll Chef nicht sehen
↳ Datenschutzgründe
↳ Wenn schnell: Skala
↳ Wenn gut sein soll: Text

6) Über sich preisgeben: Daher nicht. Aber könnte nützlich sein,
↳ Besser selbst erkundigen
↳ Freunde / Familie kennt einen besser
↳ Wenn Erfolgreicher Treffer, dann schon mal guter Start

Alter: 17
Schüler
Wenigerbücher

7) Wenn automatisch: App nicht kaufen
Wenn Erhaltung, wie es sieht: dann entf.
↳ eher bei Lieblingsapp schreiben

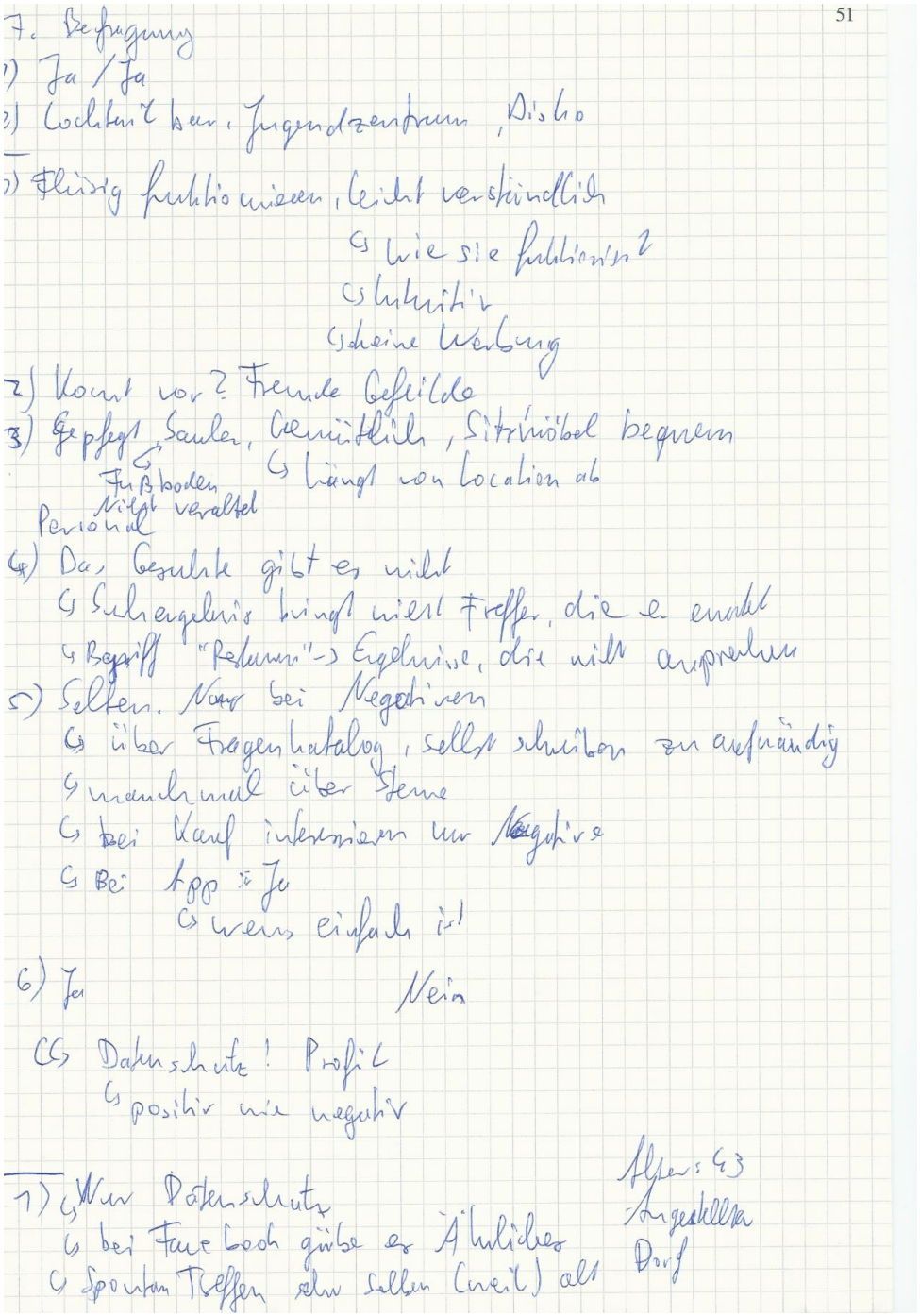


6. Befragung 49

1) Ja/Ja
2) Partyraum (wo selbst Geb. feiern kann), Disko Klo

1) Keine Werbung, keine Hänger, keine Kondition auf Face book o.ä.
↳ wenig
2) Selbst: Samstags
3) Großräumig, ungezwungene Atmosphäre, Leute
↳ gehobenes Niveau
↳ Sauberkeit (Toiletten)
↳ Personal (Ordner)
4) Freunde können mich's vorschlagen
↳ Orte sind mir unbekannt
↳ Muss über vernünftige Person kommen (die Empfehlung)
5) Nein. Nur wenn richtig gut oder richtig schlecht
↳ (Damen auch/mehr)
↳ bei Kopp: ja, wenns schnell geht
↳ "Zelt sofort": Dann nicht
↳ Schild selber in Hand halten, wenn Sie bewacht
6) Schon.
↳ Ich könnte ja mal gucken, ob Kopp mir was sagt
↳ ausprobieren

1) Nicht öffentlich. Bei Freunden schon. Manchmal
↳ wenns langweilig ist, würde Funktion gerne
sehen. 1x
Abends
Randvoll



7. Befragung 51

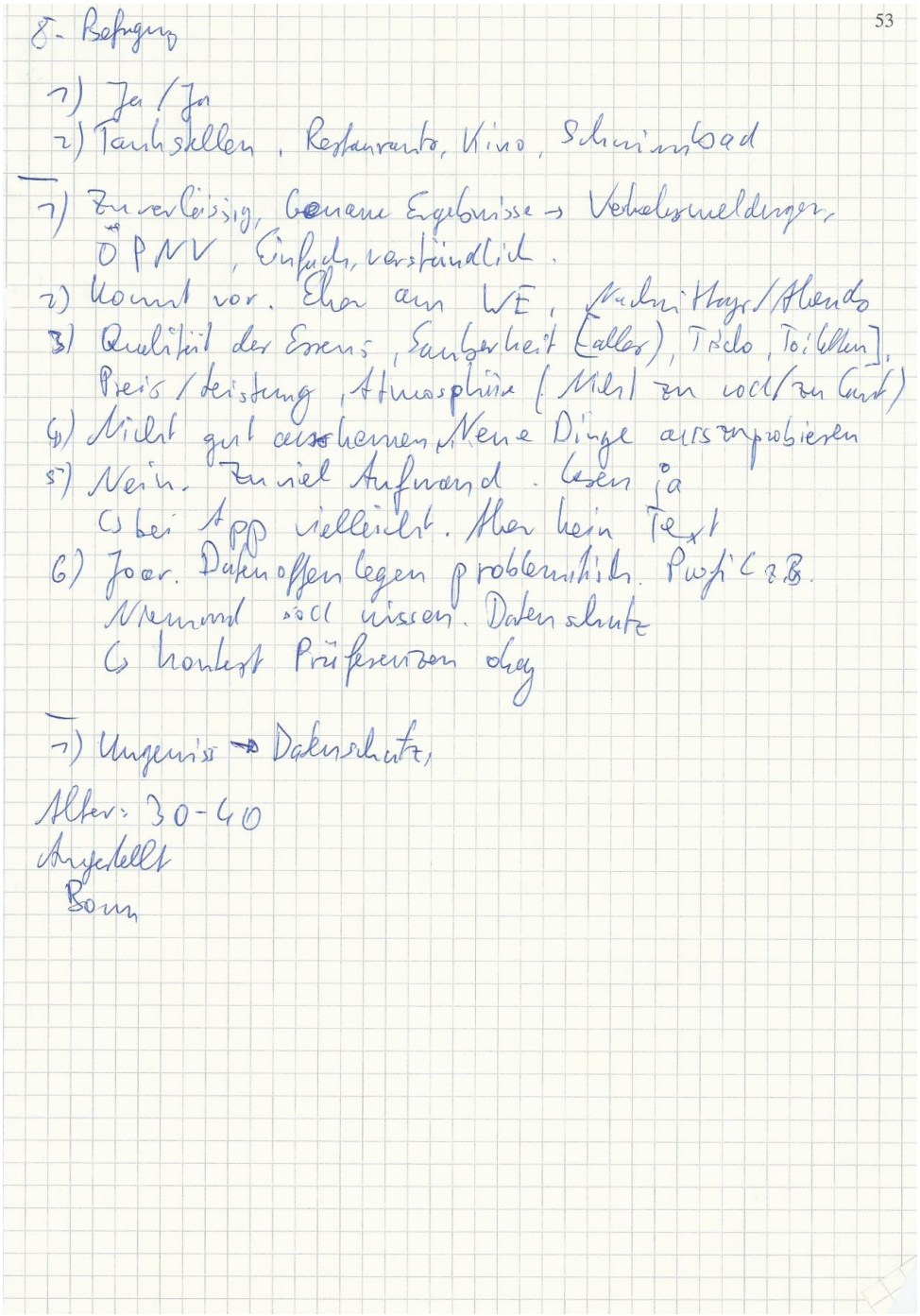
1) Ja / Ja
2) Locations bes. Jugendzentrum, Disko
3) flüssig formulieren, leicht verständlich
   - wie sie formulieren?
   - Exklusiv
   - keine Werbung
4) Kommt vor? Fremde Gefilde
5) Gepflegt, Sauber, Sanitär, Sitzmöbel bequem
   - Fußboden
   - hängt von Location ab
   - Personal nicht veraltet
6) Das Gesuchte gibt es nicht
   - Suchergebnis bringt viel Treffer, die es enthält
   - Begriff "Reisen" -> Ergebnisse, die nicht anspr.
7) Selten. Nur bei Negativem
   - über Fragenkatalog, selbst schreiben zu aufwändig
   - manchmal über Sterne
   - bei Kauf interessiert Negativ
   - Bei App ist Je
     - wenn einfach ist

6) Ja Nein

- Datenschutz! Profil
  - positiv wie negativ

7) Wenn Datenschutz
   - bei Facebook gäbe es Ähnliches
   - Spontan Treffen ohne selber (weit) als

Alter: 43
Angestellter
Darf



8. Befragung 53

1) Ja / Ja
2) Tankstellen, Restaurants, Kino, Schwimmbad

1) Zuverlässig, genaue Ergebnisse → Verbalmeldungen, ÖPNV, Einfach, verständlich.
2) kommt vor. Eher am WE, Nachmittags/Abends
3) Qualität des Essens, Sauberkeit (allg.), Tische, Toiletten, Preis/Leistung, Atmosphäre (Musik zu laut/zu leise)
4) Nicht gut aussehen, Neue Dinge ausprobieren
5) Nein. Zu viel Aufwand. Lesen ja
   ↳ bei App vielleicht. Aber kein Text
6) Joar. Daten offen legen problematisch. Profil z.B.
   Niemand soll wissen. Datenschutz
   ↳ konkret Präferenzen okay

1) Ungenau → Datenschutz,

Alter: 30-40
Angestellt
Bonn

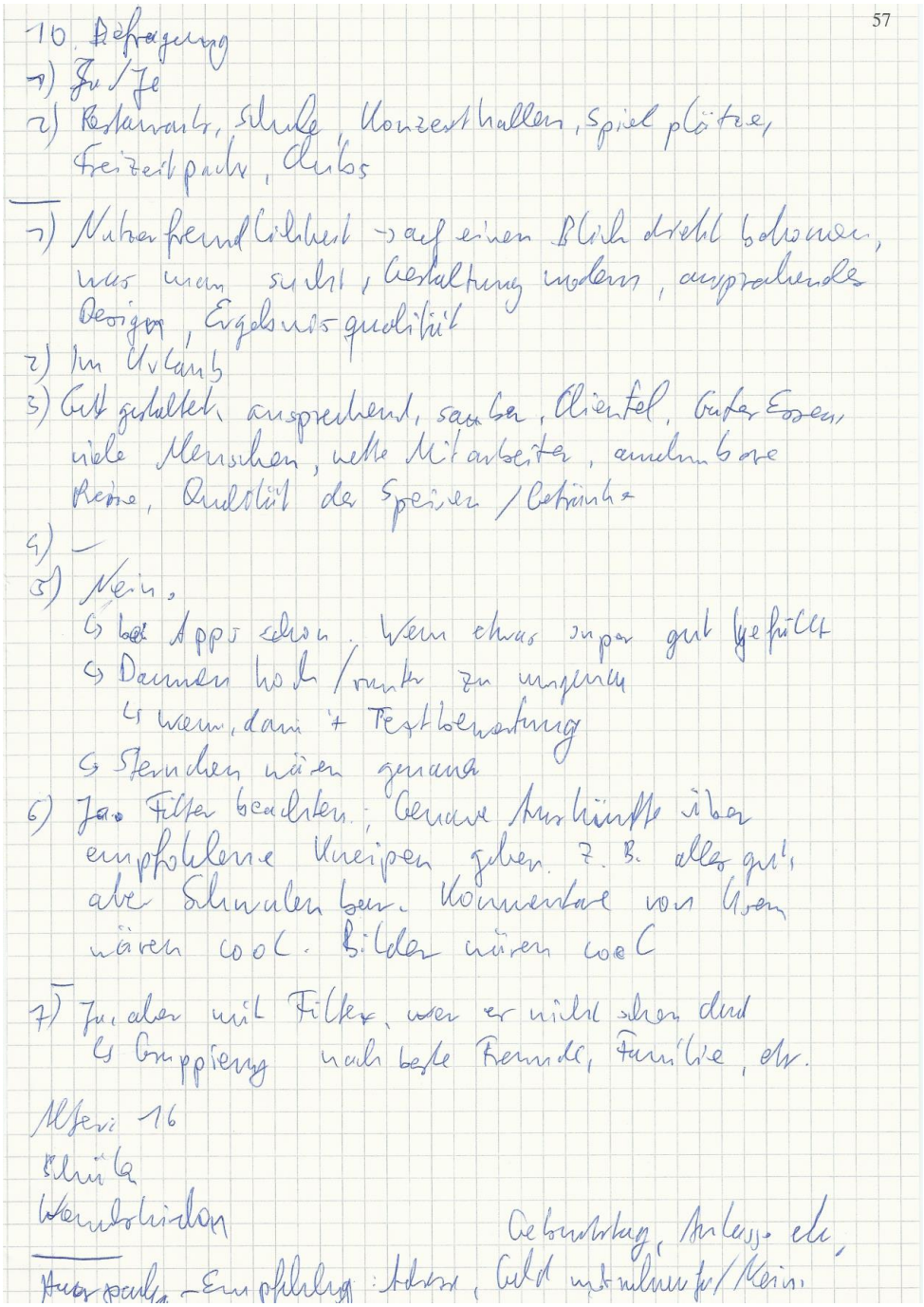
55

9, Befragung
1) Ja / Ja
2) Diskotheken, Bars, ..

1) Übersicht → keine 30 Menüpunkte, schneller Zugriff, funktioniert, wenn Werbung, dann kein Nachfragen, ob für 2 € entfernt werden soll, Zweck erfüllen
2) Kommt selten vor. Meist Planung im Voraus, erst 1, 2 Mal passiert, dass er nicht wusste
↳ 1x in Frankreich
3) Musik sollte angepasst sein, Sauber (Toiletten)
Preise angemessen
4) Anbindung
5) Nein. Zu umständlich, kein Extra Website
Bock dafür
↳ bei App: schnell (wichtige Kriterien)
↳ Skala + (kleines Feld für Anmerkungen)
↳ gibt immer was, das Sterne nicht angeben
6) Ja. Nur nicht, wenn kostenpflichtig.
↳ bei guter Bewertung der App
↳ sollte + Anfahrtswege wissen
+ Eintrittsgeld ja/nein
+ Verabredungsfunktion

7) Ja, wenn freiwillig ist, alle Bestimmte ausschließen

Alter: 16
Schüler
Wermelskirchen



57

10. Befragung
1) Ja / Je
2) Restaurants, Schule, Konzerthallen, Spielplätze, Freizeitparks, Clubs

1) Naheliegend/Gelobt → auf einen Blick direkt bekommen, was man sucht, Gestaltung modern, ansprechendes Design, Ergebnis qualitativ
2) Im Urlaub
3) Gut gestaltet, ansprechend, sauber, Ambiente, Preis, Essen, viele Menschen, nette Mitarbeiter, angenehme Preise, Qualität der Speisen / Getränke
4) —
5) Nein,
↳ bei Apps schon. Wenn etwas super gut gefällt
↳ Dann noch / mehr zu unserem
↳ wenn, dann + Testbewertung
↳ Sternchen wären genau
6) Ja. Filter beachten; Genaue Auskünfte über empfohlene Kneipen geben z. B. alles gut, aber Schwulen bar. Kommentare von Usern wären cool. Bilder wären cool
7) Ja, aber mit Filter, wer er nicht sehen darf
↳ Gruppierung nach beste Freunde, Familie, etc.

Alter: 16
Schule
Wohnsituation

Geburtstag, Anlass etc.

Hobbys – Empfehlung: Adresse, Geld mitnehmen ja/nein

59

11. Befragung

1) Ja / Nein

2) Kneipe, Restaurant, Innenstadt, Einkaufscenter,

3) Aktualität, Bekanntheit, Günstig (Preis)

4) Eher nicht, weil Kenntnis v. Programm (Nein
↳ wenn noch anderer, dann schon
↳ weil man viel selbst ausprobiert

5) Musik, Publikum, Preis / Leistungsverhältnis,

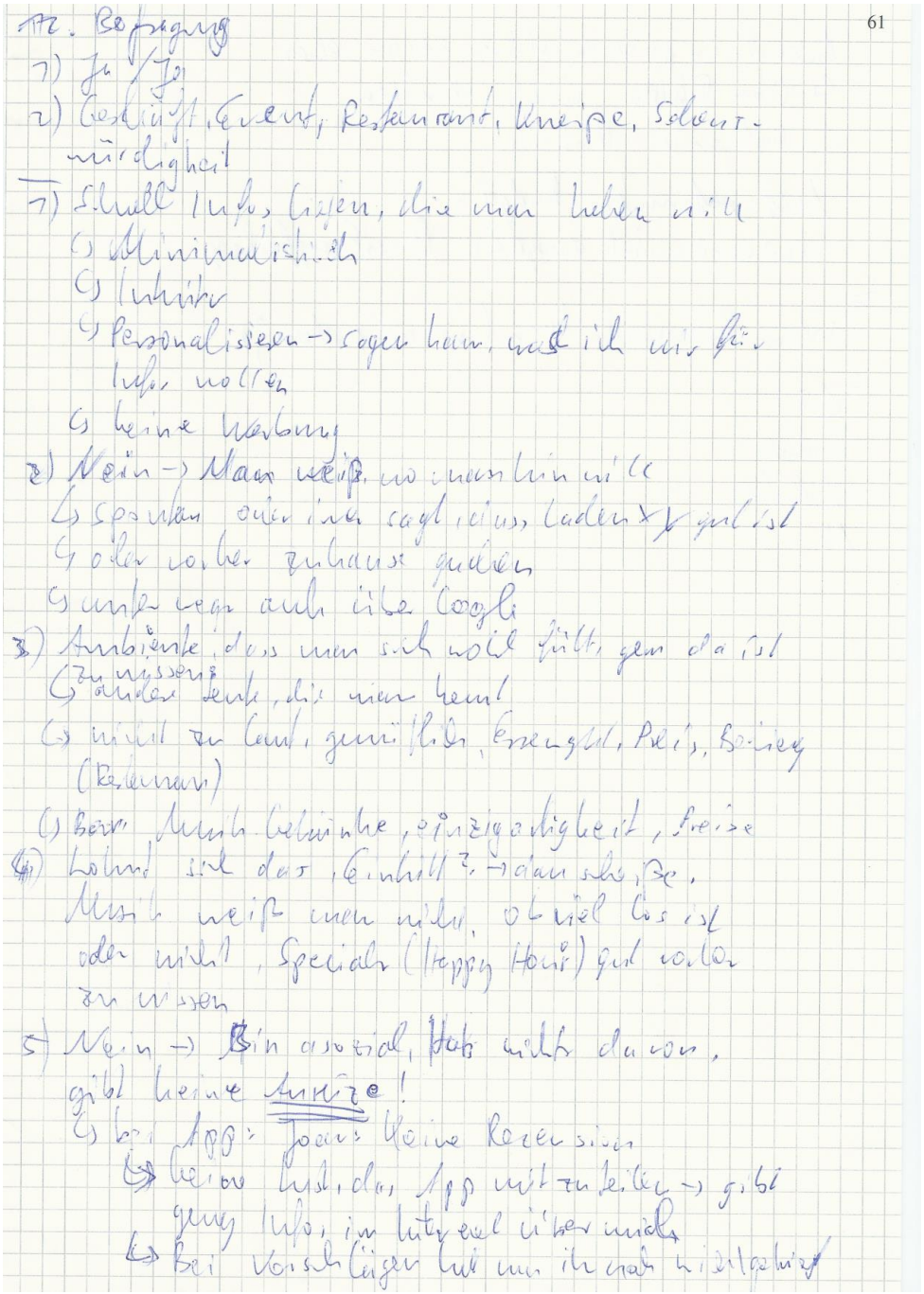
6) /

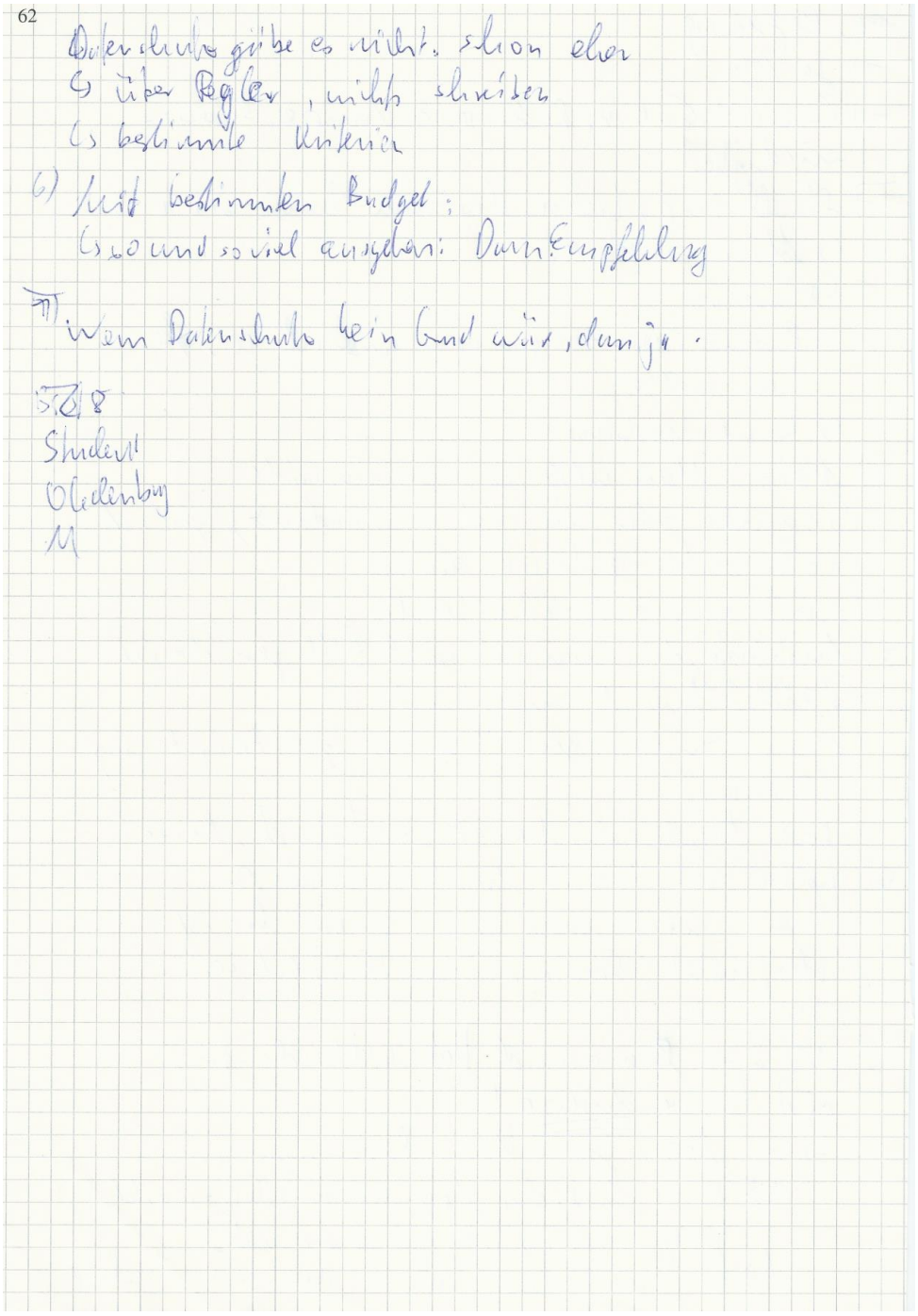
7) Nein. Zu Lästig. (Wunsch, dass Benachrichtigen getrennt)
↳ Ja, bei App → ~~Quiz~~ Multiple Choice
↳ Test nicht. Fragen von 7 bis 20
↳ Nicht zu viel

8) Ja. Tür beide, die immer auf Achse sind
↳ Profile sind scheiße
↳ Über Spalten: Musik Günstig
Kneipe X 2
Kneipe Y M
↳ Ausprobieren ↳ bist du dabei gehen nicht

9) Eher nicht
↳ weil man was von sich preis geben muss

Alter: 6 ✓
Angestellt
Rentnerin




62
Datenschutz gibt es nicht. schon eher
↳ über Regler, nichts schreiben
↳ bestimmte Kriterien
b) fest bestimmten Budget:
↳ so und so viel ausgeben. Durch Empfehlung
wenn Datenschutz kein Grund wäre, dann ja.
5.08
Student
Oldenburg
M

63

13. Befragt

1) Ja / Ja

2) Restaurant, Bar, Shisha Bar, Clubs

1) Kommunikation, App zur Unterhaltung
Datenschutz hoch, Kontakt

2) Nein, kommt nicht vor

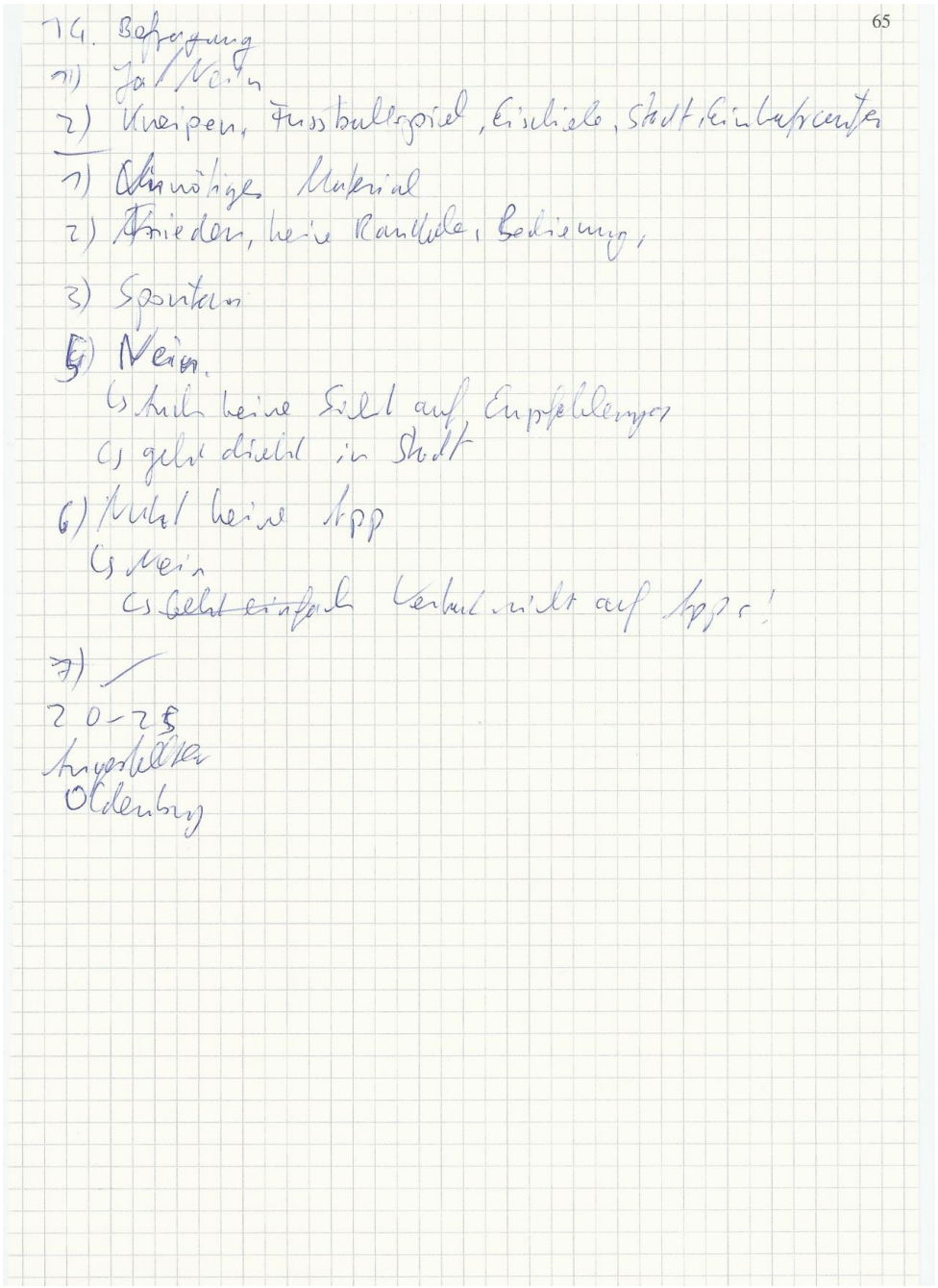
3) Viele Sitzplätze, Nette Bedienung, Auswahl der Getränke, ab 18 zumindest abends.

5) Nein. Nicht viel im Internet unterwegs, aber Bewertungen werden angeguckt
↳ bei App schon möglich
↳ einfach per Anklicken
↳ schnell, überschaubar
↳ Sternchen

6) Würde es nutzen
↳ Budget-Eintrag wäre gut
↳ mit Ortung + Uhrzeit
↳ Altersklasse

7) Würde es nutzen
↳ sollten nur bestimmte Leute sehen können

21
Azubi
[illegible]



TG. Befragung 65

1) Ja / Nein

2) Kneipen, Fussballspiel, Eisdiele, Stadt, Einkaufscenter

1) Abwechslung, Material

2) Frieden, keine Randale, Bedienung,

3) Spontan

4) Nein.

Es sind keine Stell auf Empfehlungen

Es geht direkt in Stadt

5) Nicht keine App

Es Nein

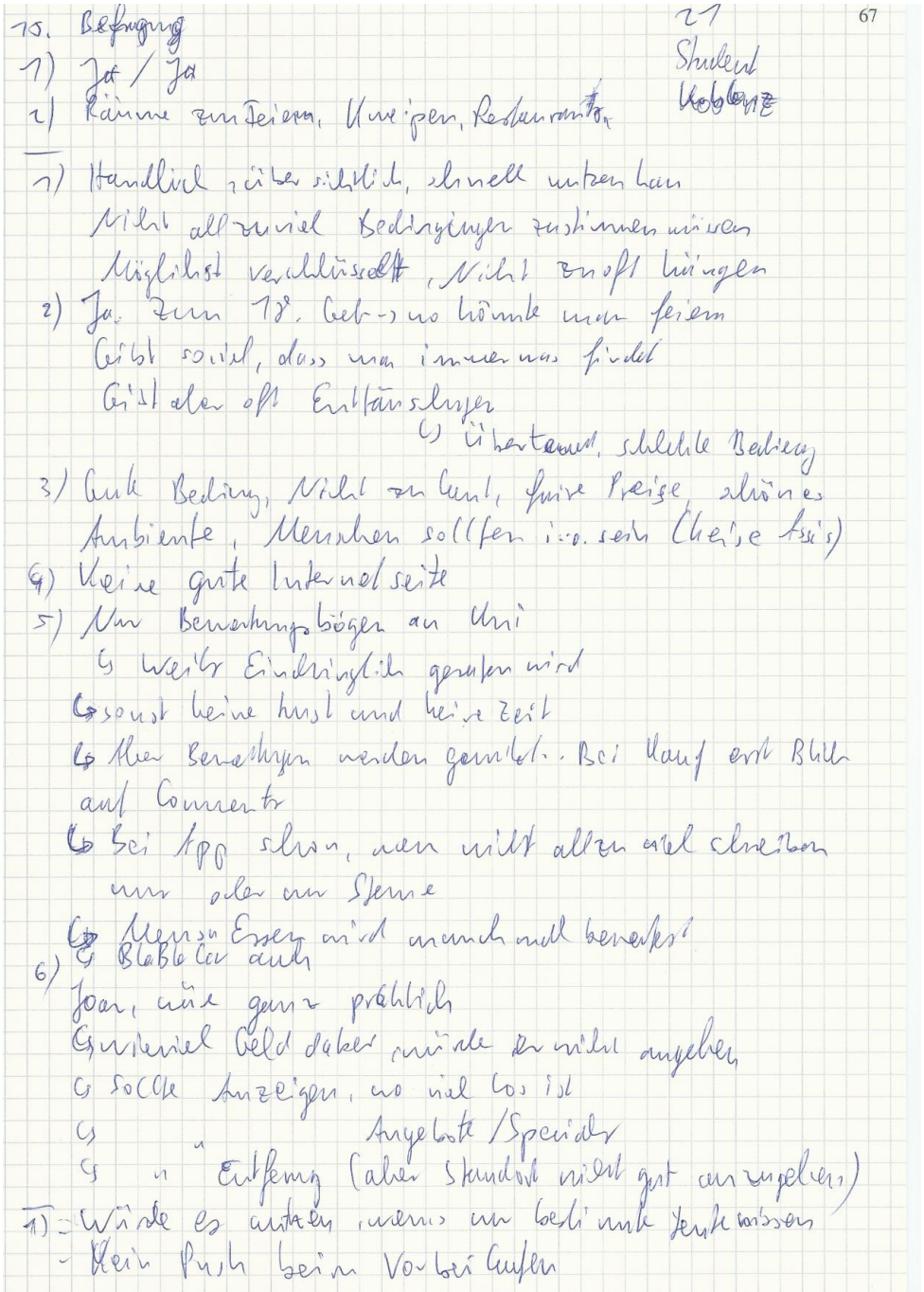
Es ~~Geht einfach~~ Verlass nicht auf Apps!

6) /

20-25

Langenhagen

Oldenburg



15. Befragung 27 67
Student
1) Ja / Ja
2) Räume zu Feiern, Kneipen, Restaurants,

1) Handlich, übersichtlich, schnell unten kann
Nicht allzuviel Bedingungen zustimmen müssen
Möglichst verschlüsselt, Nicht zu oft Lösungen
2) Ja. Zum 18. Geb.-> wo könnte man feiern
Gibt soviel, dass man immer was findet
Ist aber oft Entfernungsfrage
↳ Übersichtl, schnelle Bedienung
3) Gute Bedienung, Nicht zu laut, faire Preise, schönes
Ambiente, Menschen sollten i.o. sein (Preis)
4) Keine gute Internetseite
5) Um Bewertungsbögen an Uni
↳ Weiter Eindringlich gebeten wird
↳ sonst keine Lust und keine Zeit
↳ Alle Bewertungen werden gemittelt. Bei Klick erst Blick
auf Comments
↳ Bei Kneipe schon, wenn nicht allzu viel schreiben
nur oder nur Sterne
↳ Wenn Essen wird manch nicht bewertet
↳ Blabla Car auch
6) Ja, wäre ganz praktisch
↳ Zuviel Geld daher würde es nicht angeben
↳ Socke anzeigen, wo viel los ist
↳ " Angebote / Specials
↳ " Entfernung (aber Standort nicht gut anzugeben)
7) Würde es anbieten, wenn um bestimmte Zeit müssen
- Kein Push beim Vorbeigehen

16. Befragung 69

1) Ja / Ja
2) Kneipe, Bar, Disko, Feiern,

---

1) Einfach zu bedienen, nicht zuviel Funktionalitäten, Begrenzung auf Kernkompetenz, keine Zugriffe auf Kamera, Kontakte etc → wenig Möglichkeiten, auf Daten zuzugreifen, gut im App-Store zu finden, kostenfrei

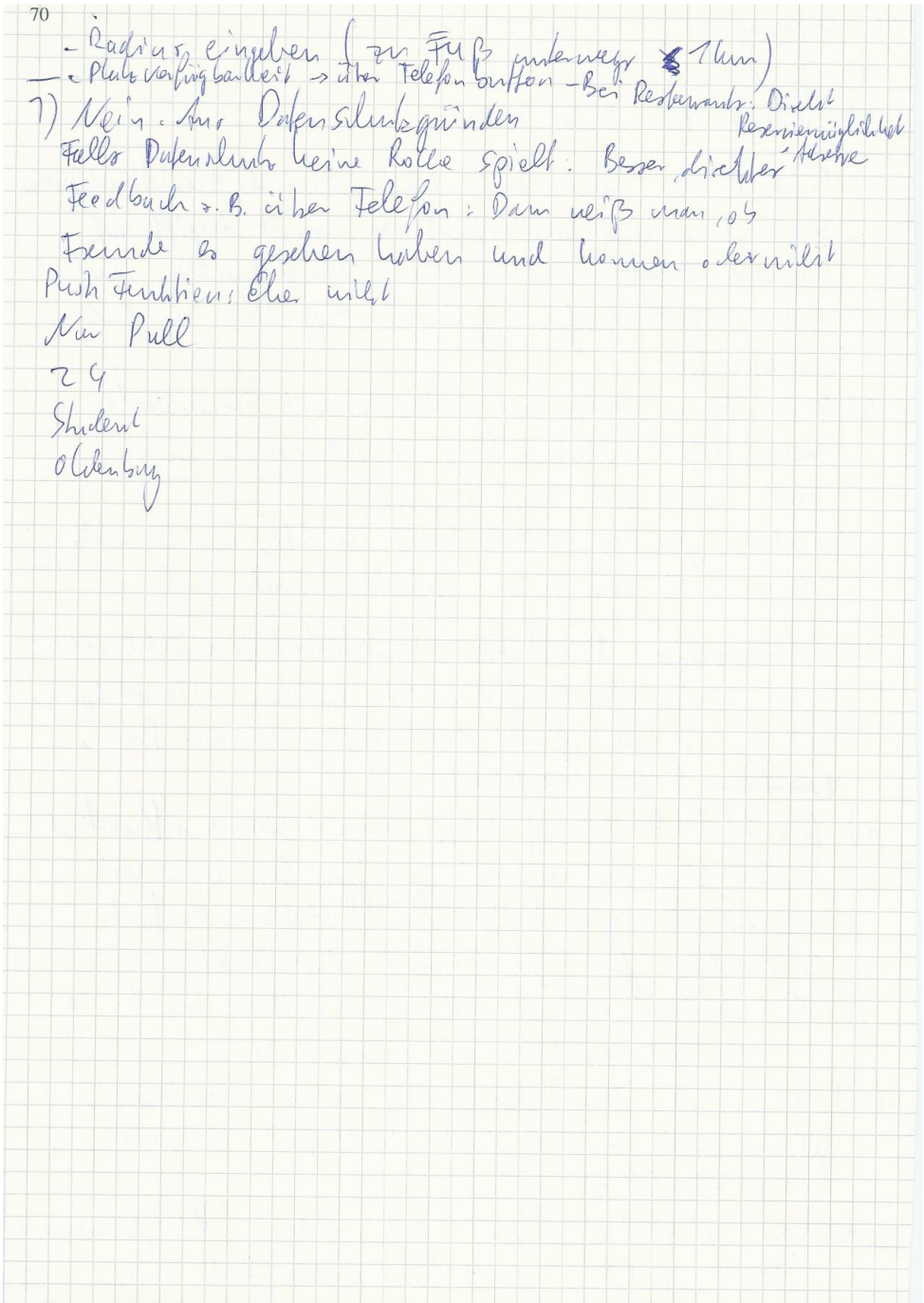
2) kommt vor. Wenn man abends mit Freunden in neue oder unbekannte Stadt geht, wenn man noch nicht viele Leute kennt

4) ↳ Schwierigkeiten: Man weiß nicht, was es alles gibt und was einem gefallen könnte

3) Atmosphäre, Rauchfrei, Lautstärke (gemütlich oder muss man sich anschreien?), welches Klientel (jung, alt, assi, normal), (Preis / Leistung) → Nicht nur Preis, ~~Erreichbar~~ Nähe + Erreichbarkeit

5) Ja. Auf BlaBlaCar, auf Ebay, → meist Sterne + Kurztext
↳ aber Kurztext nicht gern, nur fürs Gewissen wie man es selbst gern hätte (bei Personen) → bei Produkten zu faul
↳ über App: Ja möglichst einfach + ohne Text
↳ entweder Gesamtlocation mit Stern von 1 – 10
↳ + Unterpunkten wenn man will

6) Prinzipiell ja. Aber lieber jemand anders machen lassen, wenn's mal in eine andere Stadt geht. Müsste KOSTENFREI sein!
↳ Was ist Nähe? Oder Ort Eingeben. GPS geht manchmal nicht + oft will man's für andere Stadt raussuchen


70
- Radius eingeben (zu Fuß zurücklegen / Uhrzeit)
- Planbarkeit → über Telefon buchen - Bei Restaurants: Direkt
1) Nein - Aus Datensicherheitsgründen
Reservierungsmöglichkeit
Falls Datensicherheit keine Rolle spielt: Besser, direkter Ansprache
Feedback z.B. über Telefon: Dann weiß man, ob's
Freunde es gesehen haben und kennen oder nicht
Push Funktion: Ehe nicht
Max Pull
24
Student
Oldenburg

71

17) Befragung

1) Ja / Ja

2) Ort, Bar, Kneipe

---

1) Nicht viel Speicherplatz
   Einfach zu verstehen
   Sicher ist
   ↳ Buch App
   ↳ Kein Datenzugriff

2) Meistens vorher

3) Man muss sich unterhalten können
   Guter Service. Bezahlbare Preise.
   Gemütlich. Man muss gerne sitzen

4)

5) Nein. Desinteresse.
   ↳ aber Nutzen: ↳ Bei Produkten
   ↳ Bei Rezepten

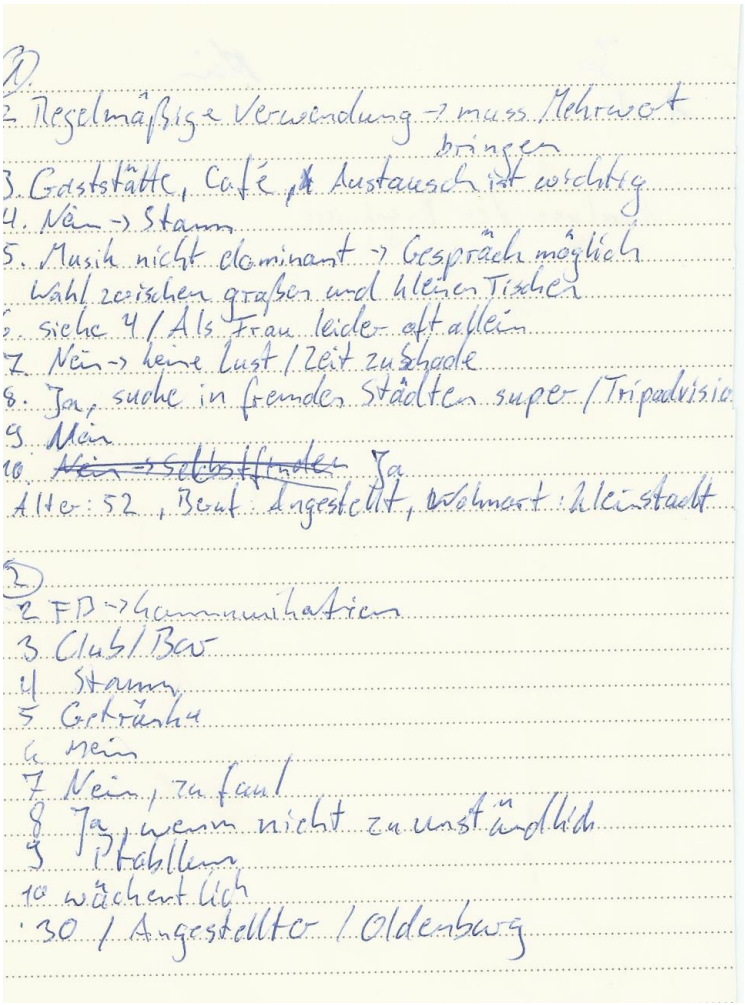
   ↳ Bei App: Nein
   ↳ kein Bedürfnis danach

6) Nein, weil sie es vorher weiß
   ↳ Man geht dahin, wo man eh gern ist
   ↳ Vorschlagen ist gruselig

7) Nein

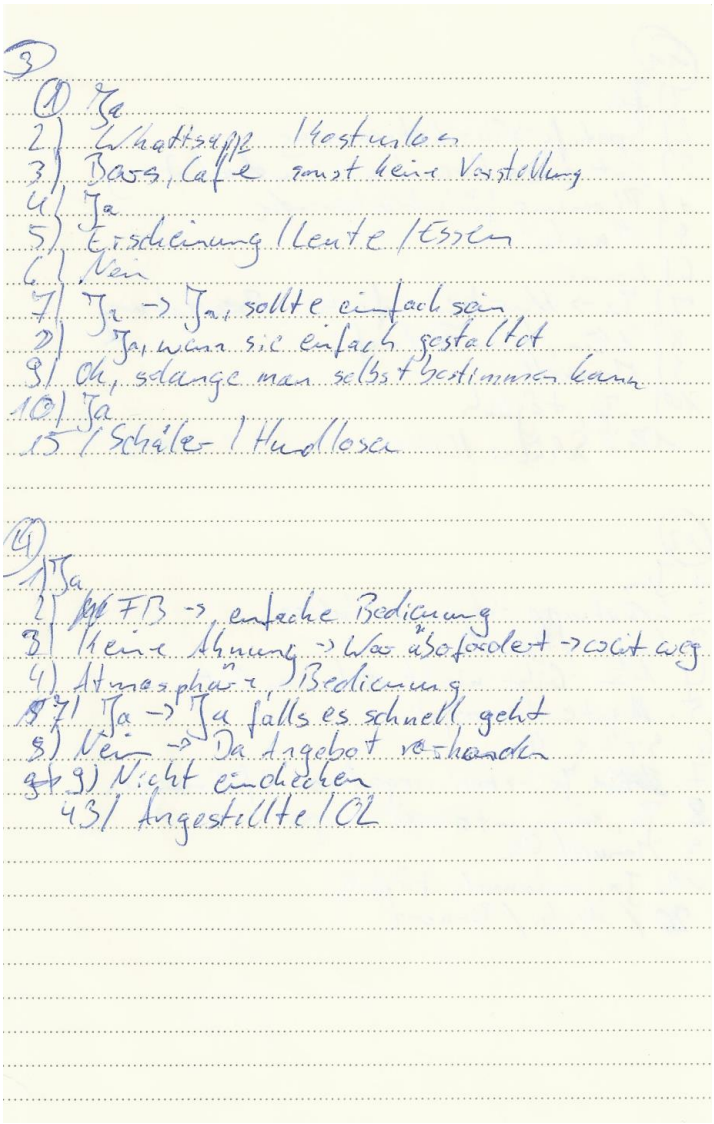
---

20 – 25
Student
Großstadt



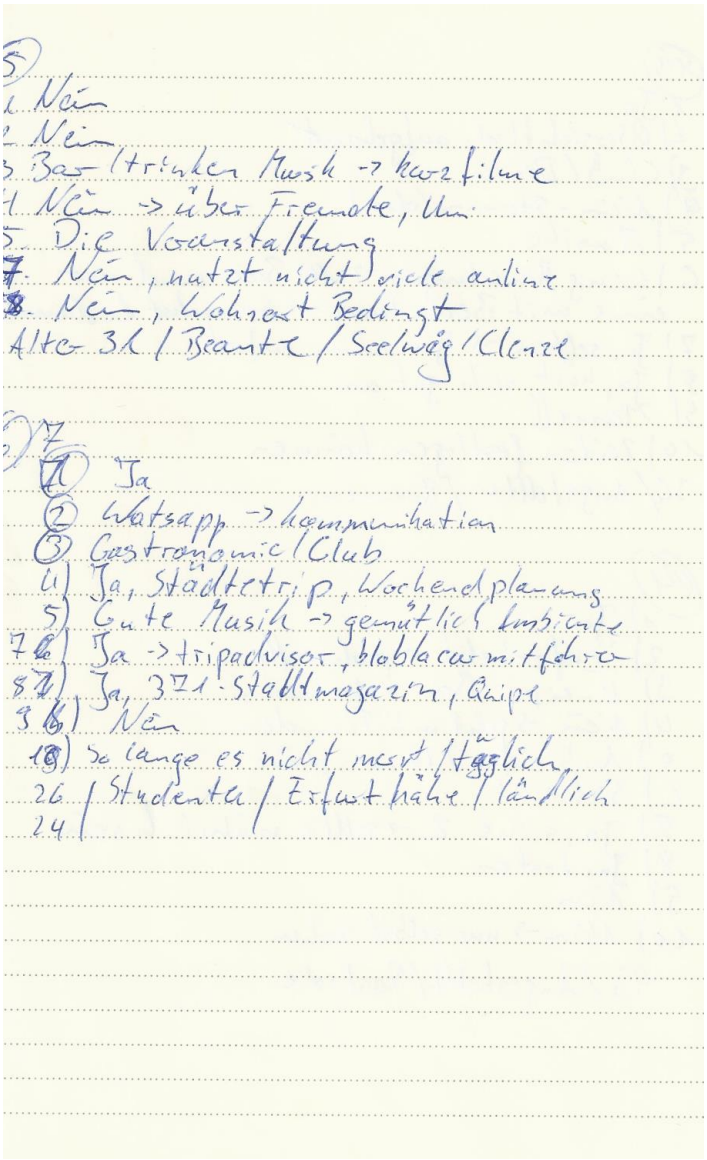
1)
2. Regelmäßige Verwendung → muss Mehrwert bringen
3. Gaststätte, Café ✓ Austausch ist wichtig
4. Nein → Stamm
5. Musik nicht dominant → Gespräch möglich
Wahl zwischen großen und kleinen Tischen
6. siehe 4 / Als Frau leider oft allein
7. Nein → keine Lust / Zeit zu schade
8. Ja, suche in fremden Städten super / Tripadvisor
9. Nein
10. ~~Nein → Selbstfinder~~ Ja
Alter: 52, Beruf: Angestellt, Wohnort: Kleinstadt

2)
2 FB → Kommunikation
3 Club / Bar
4 Stamm
5 Getränke
6 nein
7 Nein, zu fern
8 Ja, wenn nicht zu unständlich
9 Probleme
10 wöchentlich
30 / Angestellte / Oldenburg



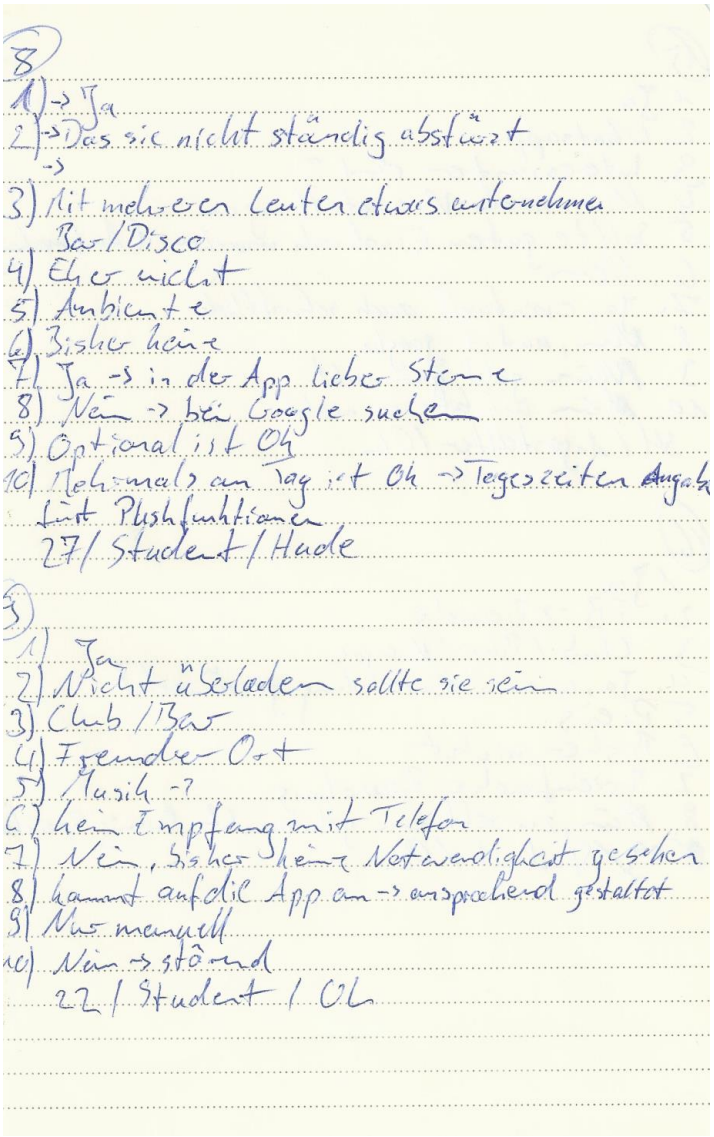
3)
1) Ja
2) Whatsapp / Internet
3) Bars, Cafe somit keine Vorstellung
4) Ja
5) Erscheinung / Leute / Essen
6) Nein
7) Ja -> Ja, sollte einfach sein
8) Ja, wenn sie einfach gestaltet
9) Ok, solange man selbst bestimmen kann
10) Ja
15 / Schüler / Handlosen

4)
1) Ja
2) Nie FB -> einfache Bedienung
3) Keine Ahnung -> Wer absolviert -> weit weg
4) Atmosphäre, Bedienung
5) 7) Ja -> Ja falls es schnell geht
8) Nein -> Da Angebot vorhanden
9) Nicht einbrechen
43) Angestellte / CL



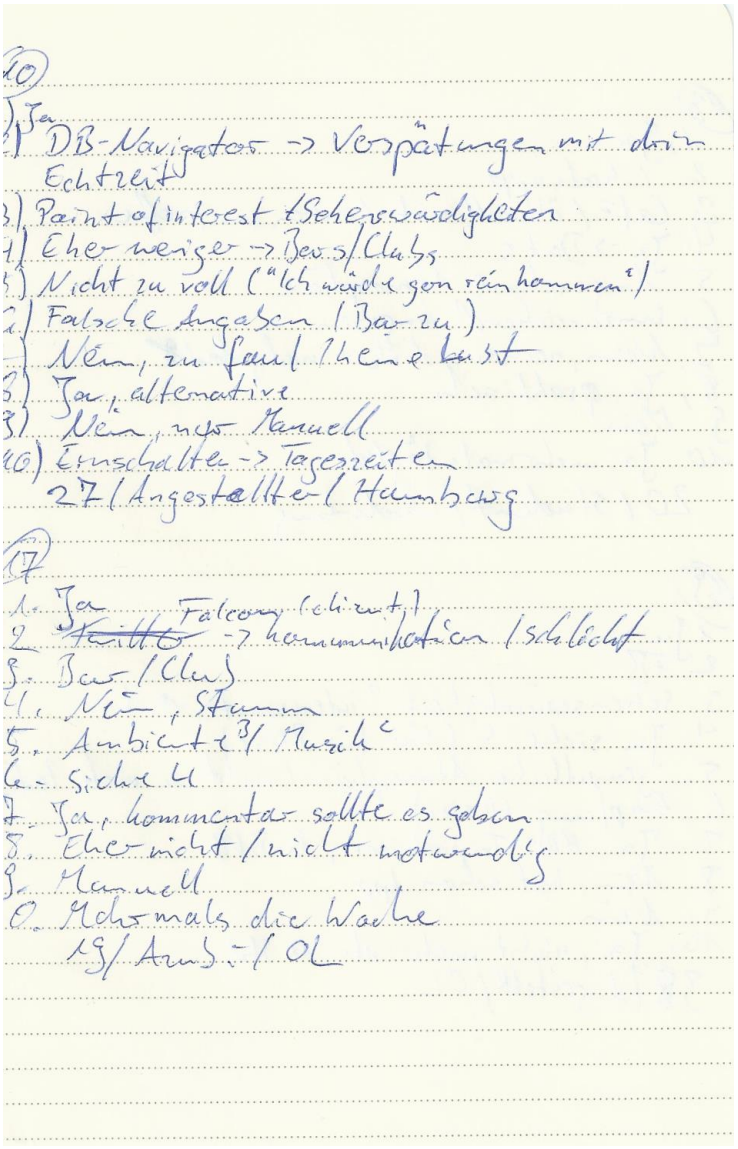
5)
1. Nein
2. Nein
3. Bar / trinken Musik -> kurz filme
4. Nein -> über Freunde, Uni
5. Die Veranstaltung
7. Nein, nutzt nicht viele online
8. Nein, Wohnort Bedingt
Alter 31 / Beamte / Seelwäg / Clenze

7)
1) Ja
2) Watsapp -> kommunikation
3) Gastronomie / Club
4) Ja, Städtetrip, Wochendplanung
5) Gute Musik -> gemütliche Ambiente
6) Ja -> tripadvisor, blablacar mitfahre
7) Ja, 3721 - stalltmagazin, Quipe
8) Nein
9) Solange es nicht mehr täglich,
26 / Studenten / Erfurt Nähe / ländlich
24



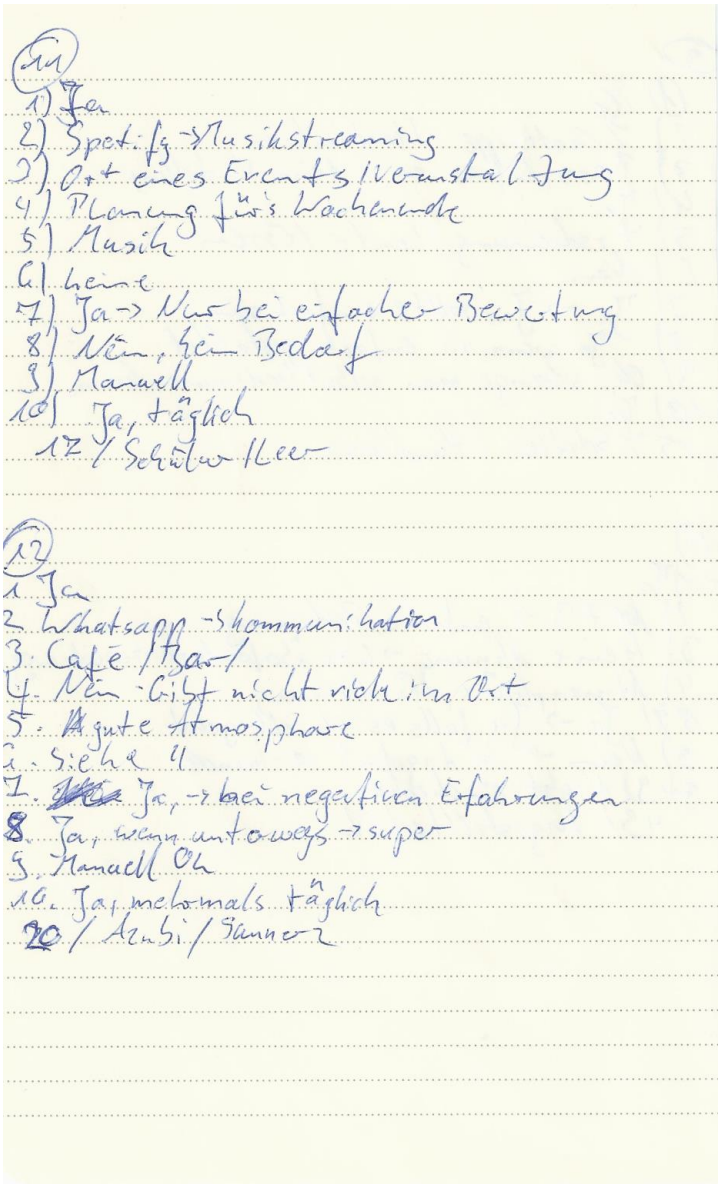
8)
1) -> Ja
2) -> Das sie nicht ständig abstürzt
->
3) Mit mehreren Leuten etwas unternehmen
Bar / Disco
4) Eher nicht
5) Ambiente
6) Bisher keine
7) Ja -> in der App lieber Sterne
8) Nein -> bei Google suchen
9) Optional ist Ok
10) Mehrmals am Tag ist Ok -> Tageszeiten Angab
Mit Pushfunktionen
27 / Student / Hunde

3)
1) Ja
2) Nicht überladen sollte sie sein
3) Club / Bar
4) Fremder Ort
5) Musik -?
6) kein Empfang mit Telefon
7) Nein, bisher keine Notwendigkeit gesehen
8) kommt auf die App an -> ansprechend gestaltet
9) Nur manuell
10) Nein -> störend
22 / Student / OL



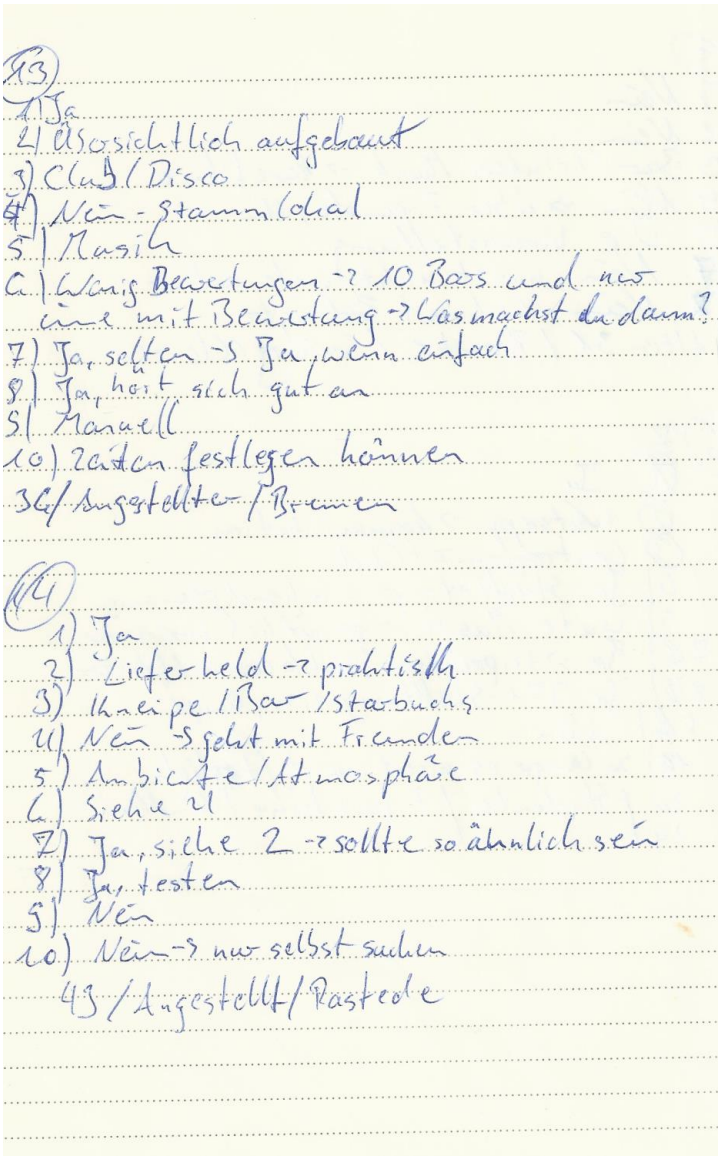
(10)
1) Ja
2) DB-Navigator -> Verspätungen mit drin
Echtzeit
3) Point of interest / Sehenswürdigkeiten
4) Eher weiger -> Bars / Clubs
5) Nicht zu voll ("Ich würde gern reinkommen")
6) Falsche Angaben (Bar zu)
7) Nein, zu faul / keine Lust
8) Ja, alternative
9) Nein, nur Manuell
10) Einschalten -> Tageszeiten
27 / Angestellter / Hamburg

(7)
1. Ja ~~Falcon (schnell)~~
2. ~~Keine~~ -> kommunikation / schlecht
3. Bar / Club
4. Nein, Stamm
5. Ambiente / Musik
6. Sicher
7. Ja, kommentar sollte es geben
8. Eher nicht / nicht notwendig
9. Manuell
10. Mehrmals die Woche
19 / Arch. - / OL



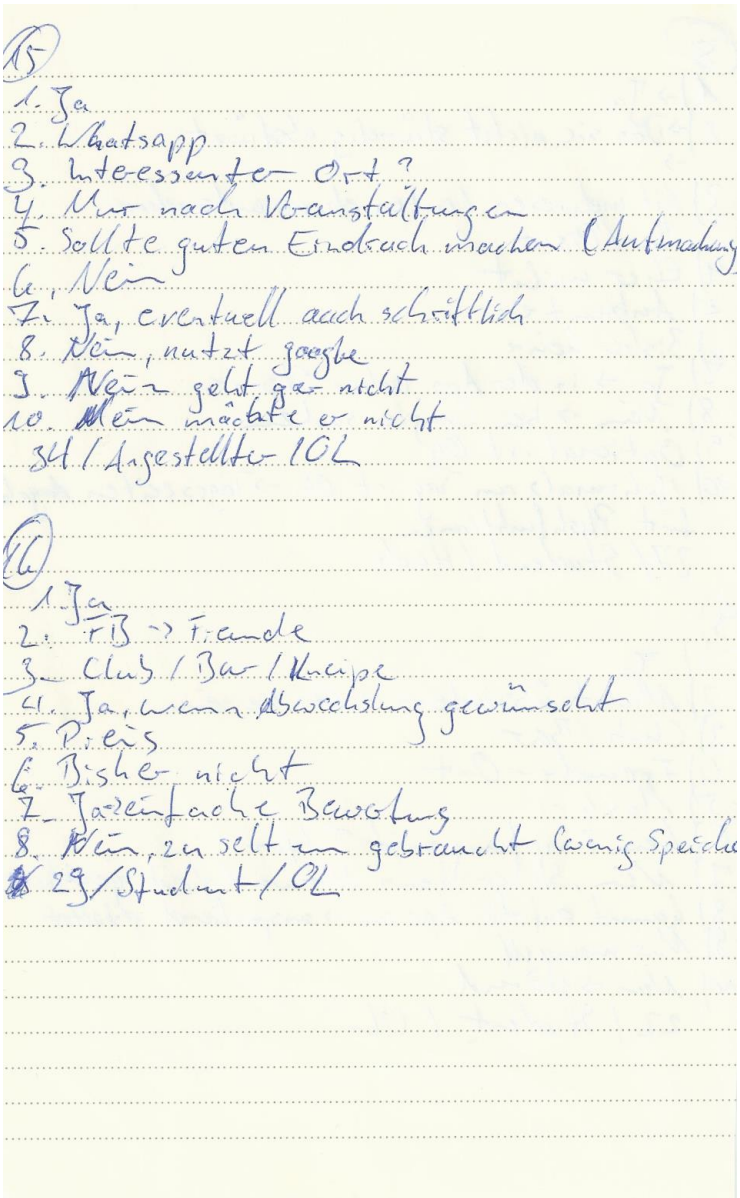
11
1) Ja
2) Spot. fg -> Musikstreaming
3) Ort eines Events/Veranstaltung
4) Planung fürs Wochenende
5) Musik
6) keine
7) Ja -> Nur bei einfacher Bewertung
8) Nein, kein Bedarf
9) Manuell
10) Ja, täglich
12 / Schüler Heer

12
1. Ja
2. Whatsapp -> kommunikation
3. Café / Bar /
4. Nein - Gibt nicht viele im Ort
5. Gute Atmosphäre
6. Siehe 4
7. ~~Nein~~ Ja, -> bei negativen Erfahrungen
8. Ja, wenn unterwegs -> super
9. Manuell Ok
10. Ja, mehrmals täglich
20 / Arbi / Sänner 2



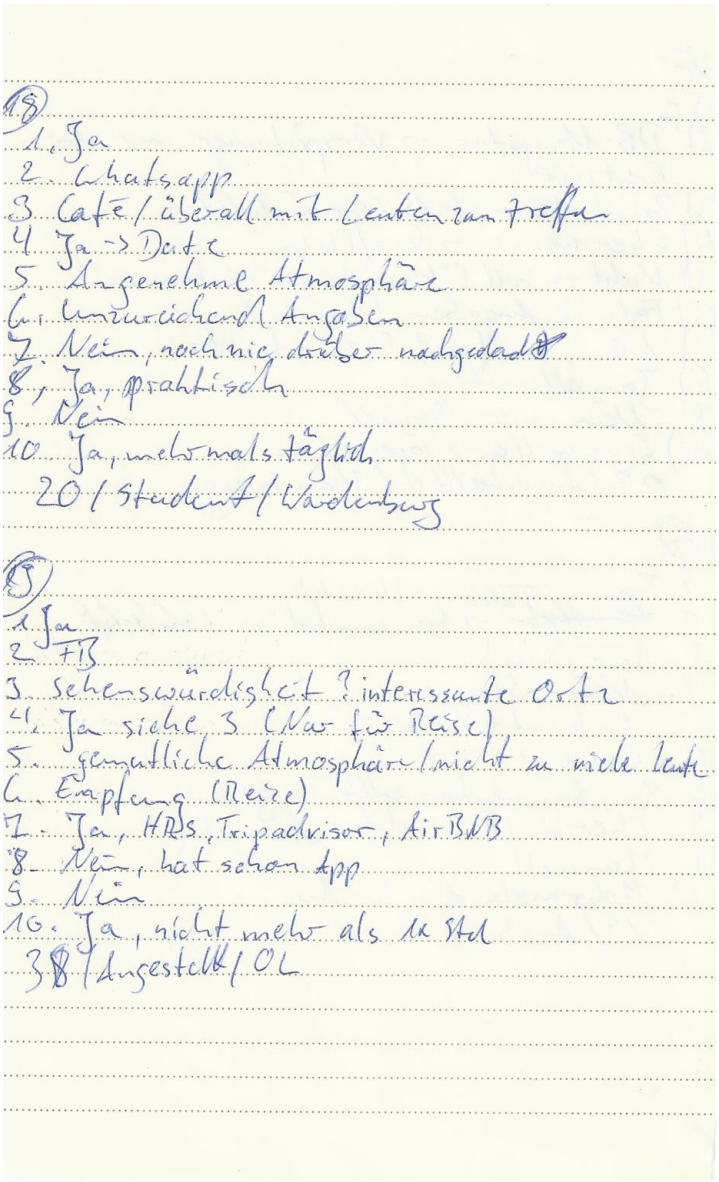
(I3)
1) Ja
2) Offensichtlich aufgebaut
3) Club / Disco
4) Nein - Stammlokal
5) Musik
6) Wenig Bewertungen -> 10 Bars und nur eine mit Bewertung -> Was machst du dann?
7) Ja, selten -> Ja, wenn einfach
8) Ja, hört sich gut an
9) Manuell
10) Zeiten festlegen können
36 / Angestellter / Bremen

(I4)
1) Ja
2) Lieferheld -> praktisch
3) Kneipe / Bar / Stammbuchs
4) Nein -> geht mit Freunden
5) Ambiente / Atmosphäre
6) Siehe 2
7) Ja, siehe 2 -> sollte so ähnlich sein
8) Ja, testen
9) Nein
10) Nein -> nur selbst suchen
43 / Angestellt / Pastorale



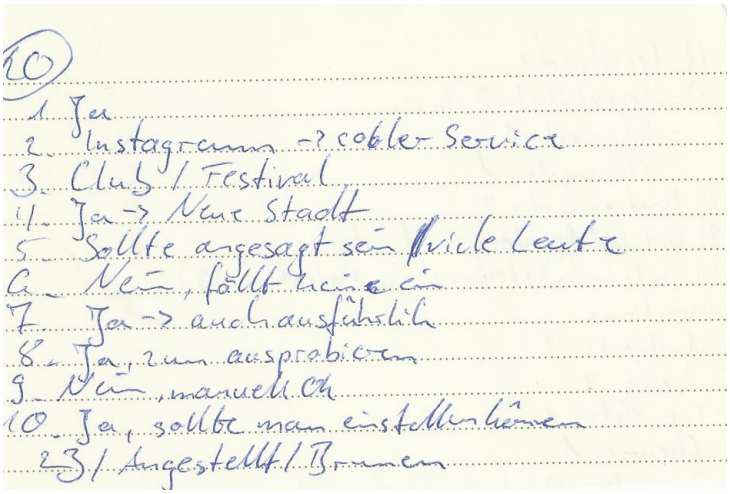
15
1. Ja
2. Whatsapp
3. Interessanter Ort?
4. Nur nach Veranstaltungen
5. Sollte guten Eindruck machen (Aufmachung
6. Nein
7. Ja, eventuell auch schriftlich
8. Nein, nutzt google
9. Nein geht gar nicht
10. Nein möchte er nicht
34 / Angestellter OL

16
1. Ja
2. FB -> Freunde
3. Club / Bar / Kneipe
4. Ja, mehr Abwechslung gewünscht
5. Preis
6. Bisher nicht
7. Ja einfache Bewertung
8. Nein, zu selten gebraucht (wenig Speich
29 / Student / OL



(18)
1. Ja
2. Whatsapp
3. Cafe / überall mit Leuten zum treffen
4 Ja -> Date
5. Angenehme Atmosphäre
6. Umfassende Angaben
7. Nein, noch nie, aber nachgedacht
8. Ja, praktisch
9. Nein
10 Ja, mehrmals täglich
20 / Student / Studentin

(19)
1. Ja
2 FB
3. Sehenswürdigkeit ? interessante Orte
4. Ja siehe 3 (Nav für Reise)
5. gemütliche Atmosphäre / nicht zu viele Leute
6. Empfang (Reize)
7. Ja, HRS, Tripadvisor, AirBNB
8. Nein, hat schon App
9. Nein
10. Ja, nicht mehr als 1x Std
38 / Angestellt / OL


10
1 Jes
2 Instagramm -> cooler Service
3 Club / Festival
4 Ja -> Neue Stadt
5 Sollte angesagt sein viele Leute
6 Nein, fällt keine ein
7 Ja -> auch ausführlich
8 Ja, zum ausprobieren
9 Nein, manuell ok
10 Ja, sollte man einstellen können
23 / Angestellt / Bremen

Glossar

Application Programming Interface (API) Eine ∼ ermöglicht es, dass verschiedene Applikationen über eine definierte Sprache miteinander kommunizieren können. Mittlerweile haben sehr viele Webdienste eine eigene API, über die Informationen für andere Anwendungen bereitgestellt sind. Solche Dienste sind z. B. Twitter, Facebook oder Google Maps [Cam12].

Brainstorming Das ∼ ist eine Methode zur Ideenfindung, in der eine Gruppe von Personen ihre Ideen zentral auf einer Tafel, Whiteboard oder einem ähnlichem Hilfsmittel visualisiert [Hes09].

Brainwriting Das ∼ ist eine erweiterte Form des ↑Brainstoming, der eine Einzelarbeitsphase vorausgeht. Dies hat den Vorteil, dass einzelne Personen nicht beeinflusst werden [Hes09].

BRISMF.MOA Bei ∼ handelt es sich um einen Algorithmus zur Matrix-Faktorisierung.

Dashboard Ein ∼ bezeichnet den Admin-Bereich, der Informationen über das System in einer aufbereiteten Form darstellt. Die Informationen können z. B. aufbereitet in Form von Diagrammen dargestellt sein.

Data Warehouse Ein ∼ ist eine Datenbank, die themenorientierte, zeitbezogene, persistente Daten enthält. Das ∼ soll bei Managemententscheidungen unterstützen [Far11].

Domain-Specific Language (DSL) Eine ∼ (dt. anwendungsspezifische Sprache), ist eine abstrakte Sprache, durch die Konzepte und Funktionen aus einer bestimmten Domäne, wie z. B. RecSys abgebildet werden [VDK02].

Heartbeat Ein ∼ ist ein Datenstrom-Element welches nur den aktuellen Zeitstempel enthält.

Hibernate ∼ ist ein ORM-Framework welches für die Speicherung von Objekte in relationalen Datenbanken genutzt wird. Dabei erfolgt die Kommunikation mit der Datenbank sowie das Aufbauen der SQL-Anfragen durch ∼ .

Gamification Der Begriff ∼ beschreibt den Einsatz von Spielmechanismen, wie Ranglisten, Abzeichen oder Punkten, in einem nicht-spielerischen Kontext zur positiven Motivation von Personen [DDKN11].

Klickrate Die ∼ (eng. click through rate) gibt an, wie oft ein Objekt durch einen Nutzer ausgewählt wurde. Wenn beispielsweise ein Objekt dem Nutzer hundertmal angezeigt wird und er es davon fünfmal auswählt, liegt die ∼ bei 0.05 [RDR07].

Location Als ∼ werden in diesem Kontext Lokalitäten oder Örtlichkeiten wie z. B. Gaststätten, Diskotheken oder Bars bezeichnet. Andere öffentliche Orte wie z. B. Turnhallen, zählen in diesem Kontext nicht als Location.

Mockup Ein ∼ ist ein erster Entwurf ohne Funktionalität. Er dient dazu, das Design zu visualisieren.

Odysseus ∼ ist ein Framework für Datenstrommanagementsysteme, welches als Forschungsplattform von der Universität Oldenburg entwickelt wird.

Object-Relational Mapping (ORM) Bei ∼ handelt es um eine Technik zur Speicherung von Objekten in relationalen Datenbanken. ∼ wird unter anderem von Hibernate genutzt.

Prototyp Ein ∼ ist eine erste Version von einem Produkt. Wesentliche Funktionalitäten sind darin enthalten. Diese sind so umgesetzt, dass Stakeholder an dem ∼ diese Funktionalitäten begutachten können.

PQL Um ∼ handelt es sich um die Procedural Query Language, mit der in ↑Odysseus Query-Pläne erstellt werden können.

Root-Rechte Durch ∼ hat ein Nutzer auf seinem Android-Smartphone erweiterte administrative Rechte, durch die er das System manipulieren kann [VVC11].

Service Provider Ein Service Provider bietet eine ↑Application Programming Interface (API), über die Informationen abgefragt werden können.

Show Case Ein ∼ ist ein Anwendungsfall mit einem sehr starken Praxisbezug. Er dient dazu, die Funktionalität eines Produktes zu präsentieren.

Spring Framework Um ∼ handelt es sich um Framework für Java, welches die Entwicklung von Java-Anwendungen unterstützen und vereinfachen soll, indem wichtige Design Pattern vorimplementiert werden und diese durch ∼ einfach angewendet werden können.

Top-k ∼ ist ein Verfahren um die besten k Elemente aus einer Menge auszuwählen. Dabei werden die Elemente nach einem vorher festgelegten Attribut unterschieden.

Usability ∼ ist das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen [Int11].

Abkürzungen

Admin Administrator

API Application Programming Interface

App Mobile Application

DBMS Datenbank-Management-System

DSMS Datenstrom-Management-System

DSL Domain-Specific Language

GPS Global Positioning System

ITTT Interleaved Test then Train

JDBC Java Database Connectivity

JPA Java Persistence API

JSON JavaScript Object Notation

JVM Java Virtual Machine

ORM Object-Relational Mapping

RecSys Recommender-System

REST Representational State Transfer

RfR Request for Recommendation

RMSE Root Mean Squared Error

TCP Transmission Control Protocol

UUID Universally Unique Identifier

WSDL Web Services Description Language

XP Extreme Programming

Abbildungen

Literatur

[ACLS07] ABERNETHY, Jacob ; CANINI, Kevin ; LANGFORD, John ; SIMMA, Alex: Online collaborative filtering. In: *University of California at Berkeley, Tech. Rep.* (2007)

[ADB+99] ABOWD, Gregory D. ; DEY, Anind K. ; BROWN, Peter J. ; DAVIES, Nigel ; SMITH, Mark ; STEGGLES, Pete: Towards a Better Understanding of Context and Context-Awareness. In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. London, UK, UK : Springer-Verlag, 1999 (HUC '99). – ISBN 3–540–66550–1, 304–307

[AGG+12] APPELRATH, H.-Jürgen ; GEESEN, Dennis ; GRAWUNDER, Marco ; MICHELSEN, Timo ; NICKLAS, Daniela: Odysseus: A Highly Customizable Framework for Creating Efficient Event Stream Management Systems. In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. New York, NY, USA : ACM, 2012 (DEBS '12). – ISBN 978–1–4503–1315–5, S. 367–368

[AJT11] ALI, Muqeet ; JOHNSON, Christopher C. ; TANG, Alex K.: *Parallel Collaborative Filtering for Streaming Data*. 2011

[AK07] ADOMAVICIUS, Gediminas ; KWON, YoungOk: New Recommendation Techniques for Multicriteria Rating Systems. In: *IEEE INTELLIGENT SYSTEMS* (2007), S. 48–55

[ASST05] ADOMAVICIUS, Gediminas ; SANKARANARAYANAN, Ramesh ; SEN, Shahana ; TUZHILIN, Alexander: Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach. In: *ACM Trans. Inf. Syst.* 23 (2005), Januar, Nr. 1, 103–145. http://dx.doi.org/10.1145/1055709.1055714. – DOI 10.1145/1055709.1055714. – ISSN 1046–8188

[AT05] ADOMAVICIUS, Gediminas ; TUZHILIN, Alexander: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. In: *IEEE Trans. on Knowl. and Data Eng.* 17 (2005), Juni, Nr. 6, S. 734–749

[AT08] ADOMAVICIUS, Gediminas ; TUZHILIN, Alexander: Context-aware Recommender Systems. In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2008 (RecSys '08). – ISBN 978–1–60558–093–7, 335–336

[BBB+01a] BECK, Kent ; BEEDLE, Mike ; BENNEKUM, Arie van ; COCKBURN, Alistair ; CUNNINGHAM, Ward ; FOWLER, Martin ; GRENNING, James ; HIGHSMITH, Jim ; HUNT, Andrew ; JEFFRIES, Ron ; KERN, Jon ; MARICK, Brian ; MARTIN, Robert C. ; MELLOR, Steve ; SCHWABER, Ken ; SUTHERLAND, Jeff ; THOMAS, Dave: *Manifesto for Agile Software Development*. http://www.agilemanifesto.org/, 2001. – [Online, besucht 13.08.2015].

[BBB+01b] BECK, Kent ; BEEDLE, Mike ; BENNEKUM, Arie van ; COCKBURN, Alistair ; CUNNINGHAM, Ward ; FOWLER, Martin ; GRENNING, James ; HIGHSMITH, Jim ; HUNT, Andrew ; JEFFRIES, Ron ; KERN, Jon ; MARICK, Brian ; MARTIN, Robert C. ; MELLOR, Steve ; SCHWABER, Ken ; SUTHERLAND, Jeff ; THOMAS, Dave: *Principles behind the Agile Manifesto*. http://www.agilemanifesto.org/principles.html, 2001. – [Online, besucht 13.08.2015].

[BBD+02] BABCOCK, Brian ; BABU, Shivnath ; DATAR, Mayur ; MOTWANI, Rajeev ; WIDOM, Jennifer: Models and Issues in Data Stream Systems. In: *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA : ACM, 2002 (PODS '02). – ISBN 1–58113–507–6, 1–16

[BBK10] BÖHMER, Matthias ; BAUER, Gernot ; KRÜGER, Antonio: Exploring the design space of context-aware recommender systems that suggest mobile applications. In: *2nd Workshop on Context-Aware Recommender Systems* (2010)

[BGG+10] BOLLES, Andre ; GEESEN, Dennis ; GRAWUNDER, Marco ; JACOBI, Jonas ; NICKLAS, Daniela ; APPELRATH, Hans-Jürgen: Sensordatenverarbeitung mit dem Open Source Datenstrommanagementframework Odysseus. In: *Informatik 2010: Service Science - Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2, 27.09. - 1.10.2010, Leipzig*, 2010, 404–409

[BGJ+09] BOLLES, Andre ; GRAWUNDER, Marco ; JACOBI, Jonas ; NICKLAS, Daniela ; APPELRATH, Hans-Jürgen: Odysseus: Ein Framework für massgeschneiderte Datenstrommanagementsystem. In: *GI Jahrestagung*, 2009, S. 2000–2014

[BH13] BALLHAUS, Werner ; HERMANN, Dr. A.: *Media Trend Outlook: Musikstreaming: das verheißungsvolle Potenzial der Musik on demand*. 2013

[BJ14] BONNIN, Geoffray ; JANNACH, Dietmar: Automated Generation of Music Playlists: Survey and Experiments. In: *ACM Comput. Surv.* 47 (2014), November, Nr. 2, 26:1–26:35. http://dx.doi.org/10.1145/2652481. – DOI 10.1145/2652481. – ISSN 0360–0300

[BKL+11] *Kapitel* InCarMusic: Context-Aware Music Recommendations in a Car. In: BALTRUNAS, Linas ; KAMINSKAS, Marius ; LUDWIG, Bernd ; MOLING, Omar ; RICCI, Francesco ; AYDIN, Aykan ; LÜKE, Karl-Heinz ; SCHWAIGER, Roland: *E-Commerce and Web Technologies: 12th International Conference, EC-Web 2011, Toulouse, France, August 30 - September 1, 2011. Proceedings*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. – ISBN 978–3–642–23014–1, 89–100

[BKWG10] BOLLEN, Dirk ; KNIJNENBURG, Bart P. ; WILLEMSEN, Martijn C. ; GRAUS, Mark: Understanding Choice Overload in Recommender Systems. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2010 (RecSys '10), 63–70

[BSATFH14] BEN-SHIMON, David ; ALEXANDER TSIKINOVSKY, Alexander ; FRIEDMANN, Michael ; HÖRLE, Johannes: Configuring and Monitoring Recommender System As a Service. In: *Proceedings of the 8th ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2014 (RecSys '14), S. 363–364

[Bur02] BURKE, Robin: Hybrid Recommender Systems: Survey and Experiments. In: *User Modeling and User-Adapted Interaction* 12 (2002), November, Nr. 4, 331–370. http://dx.doi.org/10.1023/A:1021240730564. – DOI 10.1023/A:1021240730564. – ISSN 0924–1868

[Bußß02] BUSSMANN, H.: *Lexikon der Sprachwissenschaft*. 3. Kröner, 2002

[Cam12] CAMPAIGN (UK): THE API REVOLUTION. In: *Campaign (UK)* (2012), Nr. 16, S. 24 – 25

[CDC14] CAMPOS, Pedro G. ; DÍEZ, Fernando ; CANTADOR, Iván: Time-aware Recommender Systems: A Comprehensive Survey and Analysis of Existing Evaluation Protocols. In: *User Modeling and User-Adapted Interaction* 24 (2014), Februar, Nr. 1-2, 67–119. http://dx.doi.org/10.1007/s11257-012-9136-x. – DOI 10.1007/s11257–012–9136–x. – ISSN 0924–1868

[CLEM11] CHANDRAMOULI, Badrish ; LEVANDOSKI, Justin J. ; ELDAWY, Ahmed ; MOKBEL, Mohamed F.: StreamRec: A Real-time Recommender System. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA : ACM, 2011 (SIGMOD '11), S. 1243–1246

[DADSTN12] DIAZ-AVILES, Ernesto ; DRUMOND, Lucas ; SCHMIDT-THIEME, Lars ; NEJDL, Wolfgang: Real-time Top-n Recommendation in Social Streams. In: *Proceedings of the Sixth ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2012 (RecSys '12). – ISBN 978–1–4503–1270–7, 59–66

[DDKN11] DETERDING, Sebastian ; DIXON, Dan ; KHALED, Rilla ; NACKE, Lennart: From game design elements to gamefulness: defining gamification. In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments* ACM, 2011, S. 9–15

[DLL+10] DAVIDSON, James ; LIEBALD, Benjamin ; LIU, Junning ; NANDY, Palash ; VAN VLEET, Taylor ; GARGI, Ullas ; GUPTA, Sujoy ; HE, Yu ; LAMBERT, Mike ; LIVINGSTON, Blake ; SAMPATH, Dasarathi: The YouTube Video Recommendation System. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2010 (RecSys '10). – ISBN 978–1–60558–906–0, 293–296

[Far11] FARKISCH, Kiumars: *Data-Warehouse-Systeme kompakt : Aufbau, Architektur, Grundfunktionen*. Berlin, Heidelberg : Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2011

[FSJV14] FÉLIX, Catarina ; SOARES, Carlos ; JORGE, Alípio ; VINAGRE, João: Monitoring Recommender Systems: A Business Intelligence Approach. In: *Computational Science and Its Applications - ICCSA 2014 - 14th International Conference, Guimarães, Portugal, June 30 - July 3, 2014, Proceedings, Part VI*, 2014, S. 277–288

[Fun06] FUNK, Simon: *Netflix Update: Try This at Home*. http://www.sifter.org/~simon/journal/20061211.html, 2006. – Besucht am: 7.5.2015

[Gei13] GEISLER, Sandra: Data Stream Management Systems. Version: 2013. http://dx.doi.org/10.4230/DFU.Vol5.10452.275. In: *Data Exchange, Integration, and Streams*. 2013. – DOI 10.4230/DFU.Vol5.10452.275, 275–304

[Glo13] GLOGER, Boris: *Scrum: Produkte zuverlässig und schnell entwickeln*. Carl Hanser Verlag GmbH Co KG, 2013

[GNHS11] GEMULLA, Rainer ; NIJKAMP, Erik ; HAAS, Peter J. ; SISMANIS, Yannis: Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM, 2011 (KDD '11), S. 69–77

[GO03] GOLAB, Lukasz ; ÖZSU, M. T.: Issues in Data Stream Management. In: *SIGMOD Rec.* 32 (2003), Juni, Nr. 2, S. 5–14

[GZB+14] GAMA, Joao ; ZLIOBAITE, Indre ; BIFET, Albert ; PECHENIZKIY, Mykola ; BOUCHACHIA, Abdelhamid: A Survey on Concept Drift Adaptation. In: *ACM Comput. Surv.* 46 (2014), März, Nr. 4, 44:1–44:37. http://dx.doi.org/10.1145/2523813. – DOI 10.1145/2523813. – ISSN 0360–0300

[Han10] HANSER, Eckhart: *Agile Prozesse: von XP über Scrum bis MAP*. Springer-Verlag, 2010

[HD01] HULTEN, Geoff ; DOMINGOS, Pedro: Catching Up with the data: research issues in mining Data Streams. In: *Proc. of Workshop on Research issues in Data Mining and Knowledge Discovery*, 2001

[Hes09] HESLIN, Peter A.: Better than brainstorming? Potential contextual boundary conditions to brainwriting for idea generation in organizations. In: *Journal of Occupational & Organizational Psychology* 82 (2009), Nr. 1, S. 129 – 145

[Int11] INTERNATIONAL STANDARDS ORGANISATION: *EN ISO 9241 - Ergonomie der Mensch-System-Interaktion*. 2011

[JHD+16] JOHNSON, Rod ; HOELLER, Juergen ; DONALD, Keith ; SAMPALEANU, Colin ; HARROP, Rob ; RISBERG, Thomas ; ARENDSEN, Alef ; DAVISON, Darren ; KOPYLENKO, Dmitriy ; POLLACK, Mark ; TEMPLIER, Thierry ; VERVAET, Erwin ; TUNG, Portia ; HALE, Ben ; COLYER, Adrian ; LEWIS, John ; LEAU, Costin ; FISHER, Mark ; BRANNEN, Sam ; LADDAD, Ramnivas ; POUTSMA, Arjen ; BEAMS, Chris ; ABEDRABBO, Tareq ; CLEMENT, Andy ; SYER, Dave ; GIERKE, Oliver ; STOYANCHEV, Rossen ; WEBB, Phillip ; WINCH, Rob ; CLOZEL, Brian ; NICOLL, Stephane ; DELEUZE, Sebastien: *Spring Framework Reference Documentation*. http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/pdf/spring-framework-reference.pdf, 2016. – [Online, besucht 26.09.2016].

[JZFF10] JANNACH, Dietmar ; ZANKER, Markus ; FELFERNIG, Alexander ; FRIEDRICH, Gerhard: *Recommender Systems: An Introduction*. 1st. New York, NY, USA : Cambridge University Press, 2010. – ISBN 0521493366, 9780521493369

[KABO10] KARATZOGLOU, Alexandros ; AMATRIAIN, Xavier ; BALTRUNAS, Linas ; OLIVER, Nuria: Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2010 (RecSys '10). – ISBN 978–1–60558–906–0, 79–86

[KB11] KOREN, Yehuda ; BELL, Robert: Advances in Collaborative Filtering. In: [RRSK11], S. 145–186

[KBV09] KOREN, Yehuda ; BELL, Robert ; VOLINSKY, Chris: Matrix Factorization Techniques for Recommender Systems. In: *Journal Computer* 42 (2009), August, S. 30–37

[Kla09] KLAHOLD, André: *Empfehlungssysteme: Recommender Systems – Grundlagen, Konzepte und Lösungen*. Wiesbaden : Vieweg + Teubner, 2009. – ISBN 978–3–8348–0568–3

[Kor09] KOREN, Yehuda: Collaborative Filtering with Temporal Dynamics. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA : ACM, 2009 (KDD '09). – ISBN 978–1–60558–495–9, 447–456

[Krä07] KRÄMER, Jürgen: *Continuous Queries over Data Streams - Semantics and Implementation*. 2007

[KS09] KRÄMER, Jürgen ; SEEGER, Bernhard: Semantics and Implementation of Continuous Sliding Window Queries over Data Streams. In: *ACM Trans. Database Syst.* 34 (2009), April, Nr. 1, 4:1–4:49. http://dx.doi.org/10.1145/1508857.1508861. – DOI 10.1145/1508857.1508861. – ISSN 0362–5915

[KT03] KELLY, Diane ; TEEVAN, Jaime: Implicit Feedback for Inferring User Preference: A Bibliography. In: *SIGIR Forum* 37 (2003), September, Nr. 2, 18–28. http://dx.doi.org/10.1145/959258.959260. – DOI 10.1145/959258.959260. – ISSN 0163–5840

[Kul13] KULAS, Christian: *Entwicklung und wirtschaftliche Evaluation eines personalisierten Nachrichtenaggregators*, TU Darmstadt, Knowledge Engineering Group, Diplomarbeit, 2013. http://www.ke.tu-darmstadt.de/lehre/arbeiten/master/2013/Kulas_Christian.pdf. – Master's Thesis

[LGA15] LUDMANN, Cornelius A. ; GRAWUNDER, Marco ; APPELRATH, Hans-Jürgen: OdysseusRecSys: Collaborative Filtering based on a Data Stream Management System. In: *UM*, 2015

[LPP08] LEE, Tong Q. ; PARK, Young ; PARK, Yong-Tae: A Time-based Approach to Effective Recommender Systems Using Implicit Feedback. In: *Expert Syst. Appl.* 34 (2008), Mai, Nr. 4, 3055–3062. http://dx.doi.org/10.1016/j.eswa.2007.06.031. – DOI 10.1016/j.eswa.2007.06.031. – ISSN 0957–4174

[LR07] LEINO, Juha ; RÄIHÄ, Kari-Jouko: Case Amazon: Ratings and Reviews As Part of Recommendations. In: *Proceedings of the 2007 ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2007 (RecSys '07). – ISBN 978–1–59593–730–8, 137–140

[LSST07] LEW, Daniel ; SOWELL, Ben ; STEINBERG, Leah E. ; TULADHAR, Amrit S.: *Model-based recommendation systems*. http://cs.carleton.edu/cs_comps/0607/recommend/recommender/modelbased.html, 2007. – Besucht am 23.02.16

[LSY03] LINDEN, Greg ; SMITH, Brent ; YORK, Jeremy: Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. In: *IEEE Internet Computing* 7 (2003), Januar, Nr. 1, S. 76–80

[Mar15] MARCO GRAWUNDER: *Odysseus - Access framework*. http://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Access+framework, 2015. – Besucht am: 28.02.16

[McG05] MCGREGOR, John: Context. In: *Journal of Object Technology* 4 (2005), Nr. 7, 35–44. http://dx.doi.org/10.5381/jot.2005.4.7.c4. – DOI 10.5381/jot.2005.4.7.c4

[MDA+11] MARTIN, Francisco J. ; DONALDSON, Justin ; ASHENFELTER, Adam ; TORRENS, Marc ; HANGARTNER, Rick: The Big Promise of Recommender Systems. In: *AI Magazine* 32 (2011), Nr. 3, 19-27. http://dblp.uni-trier.de/db/journals/aim/aim32.html#MartinDATH11

[MJ09] MIRANDA, Catarina ; JORGE, Alípio M.: Item-Based and User-Based Incremental Collaborative Filtering for Web Recommendations. In: LOPES, LuisSeabra (Hrsg.) ; LAU, Nuno (Hrsg.) ; MARIANO, Pedro (Hrsg.) ; ROCHA, LuisM. (Hrsg.): *Progress in Artificial Intelligence* Bd. 5816. Springer Berlin Heidelberg, 2009, S. 673–684

[Neu05] NEUNDORF, Werner: *Zur Konvergenz des Gradientenverfahrens*. http://www.db-thueringen.de/servlets/DerivateServlet/Derivate-8268/IfM_Preprint_M_05_13.pdf, 2005. – letzter Zugriff am 10.05.2015

[PRPT05] PAPAGELIS, Manos ; ROUSIDIS, Ioannis ; PLEXOUSAKIS, Dimitris ; THEOHAROPOULOS, Elias: Incremental Collaborative Filtering for Highly-scalable Recommendation Algorithms. In: *Proceedings of the 15th International Conference on Foundations of Intelligent Systems*. Berlin, Heidelberg : Springer-Verlag, 2005 (ISMIS'05). – ISBN 3–540–25878–7, 978–3–540–25878–0, 553–561

[RDR07] RICHARDSON, Matthew ; DOMINOWSKA, Ewa ; RAGNO, Robert: Predicting clicks: estimating the click-through rate for new ads. In: *Proceedings of the 16th international conference on World Wide Web* ACM, 2007, S. 521–530

[RKZ+13] RONEN, Royi ; KOENIGSTEIN, Noam ; ZIKLIK, Elad ; SITRUK, Mikael ; YAARI, Ronen ; HAIBY-WEISS, Neta: Sage: recommender engine as a cloud service. In: 0001, Qiang Y. (Hrsg.) ; KING, Irwin (Hrsg.) ; LI, Qing (Hrsg.) ; PU, Pearl (Hrsg.) ; KARYPIS, George (Hrsg.): *RecSys*, ACM, 2013, S. 475–476

[RRSK11] RICCI, Francesco (Hrsg.) ; ROKACH, Lior (Hrsg.) ; SHAPIRA, Bracha (Hrsg.) ; KANTOR, Paul B. (Hrsg.): *Recommender Systems Handbook*. New York : Springer, 2011

[RU11] RAJARAMAN, Anand ; ULLMAN, Jeffrey D.: *Mining of Massive Datasets*. New York, NY, USA : Cambridge University Press, 2011. – ISBN 1107015359, 9781107015357

[Sa11] SCHRÖDER, Gunnar ; AL. et: *Setting Goals and Choosing Metrics for Recommender System Evaluations*. 2011

[San14] SANTOS, Jose Maria D.: *XP, FDD, DSDM, and Crystal Methods of Agile Development*. http://project-management.com/xp-fdd-dsdm-and-crystal-methods-of-agile-development/, 2014. – [Online, besucht 13.08.2015].

[SK09] SU, Xiaoyuan ; KHOSHGOFTAAR, Taghi M.: A Survey of Collaborative Filtering Techniques. In: *Adv. in Artif. Intell.* 2009 (2009), Januar, 4:2–4:2. http://dx.doi.org/10.1155/2009/421425. – DOI 10.1155/2009/421425. – ISSN 1687–7470

[SKKR02] SARWAR, Badrul ; KARYPIS, George ; KONSTAN, Joseph ; RIEDL, John: Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. In: *Fifth International Conference on Computer and Information Science*, 2002, S. 27–28

[SS13] SCHWABER, Ken ; SUTHERLAND, Jeff: *The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game*. 2013

[Sta15] STARKE, Gernot: *Effektive Softwarearchitekturen - Ein praktischer Leitfaden*. 7., überarbeitete Auflage. München : Carl Hanser Verlag GmbH Co KG, 2015

[Str11] STRICKROTH, S.: *Empfehlungssysteme für kleine Online-Communities mit regionaler Bindung*, Clausthal University of Technology, Department of Informatics, Master Thesis, 2011

[SY13] SINGH, SP ; YADAV, Asmita: Study of K-Means and Enhanced K-Means Clustering Algorithm. In: *International Journal of Advanced Research in Computer Science* 4 (2013), Nr. 10

[The15a] THE ODYSSEUS TEAM: *Odysseus Server - Der Kern*. http://odysseus.informatik.uni-oldenburg.de/index.php?id=84, 2015. – Besucht am: 23.02.16

[The15b] THE ODYSSEUS TEAM: *Odysseus Studio - Die Benutzeroberfläche*. http://odysseus.informatik.uni-oldenburg.de/index.php?id=85, 2015. – Besucht am: 23.02.16

[The15c] THE ODYSSEUS TEAM: *Über Odysseus*. http://odysseus.informatik.uni-oldenburg.de/index.php?id=77, 2015. – Besucht am: 23.02.16

[TPNT08] TAKACS, G. ; PILASZY, I. ; NEMETH, B. ; TIKK, Domonkos: Investigation of Various Matrix Factorization Methods for Large Recommender Systems. In: *Data Mining Workshops, 2008. ICDMW '08. IEEE International Conference on*, 2008, S. 553–562

[TPNT09] TAKÁCS, Gábor ; PILÁSZY, István ; NÉMETH, Bottyán ; TIKK, Domonkos: Scalable Collaborative Filtering Approaches for Large Recommender Systems. In: *J. Mach. Learn. Res.* 10 (2009), Juni, 623–656. http://dl.acm.org/citation.cfm?id=1577069.1577091. – ISSN 1532–4435

[TS J] TOUGAS, Jane E. ; STERN, Henry: Updating the Partial SVD: Making LSI Run Faster. (o. J.)

[VDK02] VAN DEURSEN, Arie ; KLINT, Paul: Domain-specific language design requires feature descriptions. In: *CIT. Journal of computing and information technology* 10 (2002), Nr. 1, S. 1–17

[VJG14] VINAGRE, João ; JORGE, Alípio Mário ; GAMA, João: Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback. In: *User Modeling, Adaptation, and Personalization - 22nd International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings*, 2014, S. 459–470

[VVC11] VIDAS, Timothy ; VOTIPKA, Daniel ; CHRISTIN, Nicolas: All Your Droid Are Belong to Us: A Survey of Current Android Attacks. In: *WOOT*, 2011, S. 81–90

[Wel12] WELTZIN, Meinald T. Thielsch & S.: *Online-Umfrage und Online-Mitarbeiterbefragung*. 2012

[WHZL13] WANG, Jialei ; HOI, Steven C. ; ZHAO, Peilin ; LIU, Zhi-Yong: Online Multi-task Collaborative Filtering for On-the-fly Recommender Systems. In: *Proceedings of the 7th ACM Conference on Recommender Systems*. New York, NY, USA : ACM, 2013 (RecSys '13). – ISBN 978–1–4503–2409–0, 237–244

[Wit85] WITZEL, Andreas: *Das problemzentrierte Interview*. 1985

[WS12] WÖRNDL, Wolfgang ; SCHLICHTER, Johann: *Empfehlungssysteme*. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Business-Intelligence/Analytische-Informationssysteme--Methoden-der-/empfehlungssysteme, 2012. – Besucht am: 23.02.16

[Yeu10] YEUNG, Albert A.: *Matrix Factorization: A Simple Tutorial and Implementation in Python*. http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/, 2010. – letzter Zugriff am 30.04.2015

[YOO13] YOOCHOOSE GMBH: *YOOCHOOSE Recommendation Home*. YOOCHOOSE GmbH, 2013

[You16] YOUTUBE: *Statistik*. https://www.youtube.com/yt/press/de/statistics.html, 2016. – Besucht am 23.02.16

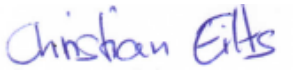
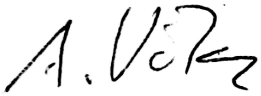
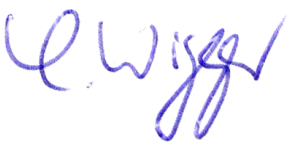
[ZKG10] ZHOU, Renjie ; KHEMMARAT, Samamon ; GAO, Lixin: The Impact of YouTube Recommendation System on Video Views. In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA : ACM, 2010 (IMC '10). – ISBN 978–1–4503–0483–2, 404–410

Index

Versicherung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 8. April 2016

Patrick Bruns	
Christian Eilts	
Martin Kuhl	
Max Leonhardt	
Christoph Schröer	
Achim Völz	
Lukas Weinel	
Christian Wigger	
Christof Wolke	
Marius Wybrands