

# Abschlussarbeit

Zur Erlangung des akademischen Grades

**Bachelor of Science**

Entwicklung einer Musik App zur Ermittlung von  
Hörgewohnheiten der Benutzer als Ausgangsbasis für  
Musikempfehlungen

Internationaler Studiengang Medieninformatik

Hochschule für Technik und Wirtschaft Berlin  
Fachbereich 4

1. Prüfer: Prof. Dr. Klaus Jung
2. Prüfer: Prof. Dr.-Ing. Carsten Busch

**Hans Seiffert | 531153**

08. März 2016

Aus Gründen der besseren Lesbarkeit wird in dieser Bachelorarbeit die Sprachform des generischen Maskulinums angewendet. Die ausschließliche Verwendung der männlichen Form soll geschlechtsunabhängig verstanden werden.

# **Inhaltsverzeichnis**

<b>Abbildungsverzeichnis</b>	<b>IV</b>
<b>Quellcodeverzeichnis</b>	<b>V</b>
<b>1. Einführung</b>	<b>1</b>
1.1. Lösungsansatz . . . . .	2
1.2. Zielsetzung . . . . .	2
1.3. Vorgehensweise . . . . .	2
<b>2. Grundlagen</b>	<b>4</b>
2.1. Musikempfehlungsverfahren . . . . .	4
2.2. Klassifikationsverfahren . . . . .	5
<b>3. Anforderungsanalyse</b>	<b>8</b>
3.1. Abgrenzung . . . . .	8
3.2. Anwendungsumgebung . . . . .	8
3.2.1. Rahmenbedingungen . . . . .	9
3.3. Anwendungsszenarien . . . . .	10
3.3.1. Musikbibliothek . . . . .	10
3.3.2. Musikplayer . . . . .	11
3.3.3. Musikempfehlung . . . . .	12
3.3.4. Debugbereich . . . . .	13
3.4. Qualität der Musikempfehlungen . . . . .	15
3.5. Datenschutz . . . . .	16
<b>4. Systementwurf</b>	<b>17</b>
4.1. Systemanforderungen . . . . .	17
4.2. Schnittstellen . . . . .	18
4.3. Anbindung der lokalen Musikbibliothek . . . . .	19
4.4. Ermittlung des Kontextes . . . . .	22
4.4.1. Merkmale zur Abbildung eines Kontextes . . . . .	22
4.5. Erfassen der Hörgewohnheiten . . . . .	25
4.6. Berechnung der Ähnlichkeit von Kontexten . . . . .	26
4.6.1. Abbildung im Merkmalsraum . . . . .	27

4.6.2. Berechnung der einfachen Distanz . . . . .	28
4.6.3. Normalisierung der einfachen Distanzen . . . . .	29
4.6.4. Beschränkung auf Wertebereiche . . . . .	30
4.6.5. Gewichtung der Merkmale . . . . .	31
4.6.6. Berechnung der Ähnlichkeit . . . . .	31
4.6.7. Beispielhafte Berechnung . . . . .	32
4.7. Generierung der Musikempfehlungen . . . . .	34
4.8. Benutzeroberfläche . . . . .	35
4.8.1. Grundstruktur . . . . .	36
4.8.2. Musikbibliothek . . . . .	37
4.8.3. Musikempfehlungen . . . . .	39
4.8.4. Musikplayer . . . . .	40
4.8.5. Debugbereich . . . . .	42
<b>5. Implementierung</b>	<b>44</b>
5.1. Projektgrundlagen . . . . .	44
5.2. Verwendete Codebibliotheken . . . . .	45
5.3. Anbindung der lokalen Musikbibliothek . . . . .	48
5.3.1. Abfragen der ersten Ebene der Musikbibliothek . . . . .	49
5.3.2. Abfragen der tieferen Ebenen der Musikbibliothek . . . . .	51
5.4. Berechnung der Ähnlichkeit von Kontexten . . . . .	51
5.4.1. Festlegen der Gewichte und Wertebereiche . . . . .	52
5.4.2. Abbildung im Merkmalsraum . . . . .	53
5.4.3. Berechnung der einfachen Distanzen . . . . .	53
5.4.4. Normalisierung der einfachen Distanzen . . . . .	55
5.4.5. Berechnung der Ähnlichkeit . . . . .	56
5.5. Generierung der Musikempfehlungen . . . . .	57
<b>6. Qualität der Musikempfehlungen</b>	<b>60</b>
6.1. Testdatensatz . . . . .	60
6.2. Ergebnisse . . . . .	61
6.3. Auswertung . . . . .	62
<b>7. Zusammenfassung</b>	<b>63</b>
7.1. Ausblick . . . . .	63

<b>Literatur</b>	<b>65</b>
<b>Abkürzungsverzeichnis</b>	<b>68</b>
<b>Glossar</b>	<b>69</b>
<b>Anhang A. Übersicht über alle Merkmale</b>	<b>71</b>
<b>Anhang B. Übersicht über die UI des Prototypen</b>	<b>72</b>
<b>Anhang C. Entity-Relationship-Modell</b>	<b>74</b>
<b>Anhang D. Werte des Testdatensatzes</b>	<b>75</b>
<b>Anhang E. Empfehlungen laut Analyse A</b>	<b>76</b>
<b>Anhang F. Empfehlungen laut Analyse B</b>	<b>77</b>
<b>Anhang G. Empfehlungen laut App</b>	<b>78</b>
<b>Anhang H. Ergebnisse der Auswertung - Überblick</b>	<b>79</b>
<b>Anhang I. Ähnlichkeit der Empfehlungen der Analysen A zu B</b>	<b>80</b>
<b>Anhang J. Ähnlichkeit der Empfehlungen der App zu Analyse A</b>	<b>81</b>
<b>Anhang K. Ähnlichkeit der Empfehlungen der App zu Analyse B</b>	<b>82</b>
<b>Anhang L. Beiliegende CD</b>	<b>83</b>
<b>Eidesstattliche Versicherung</b>	<b>84</b>

## **Abbildungsverzeichnis**

1.	Diagramm eines Entscheidungsbaums . . . . .	6
2.	Anwendungsfalldiagramm: M1 . . . . .	10
3.	Anwendungsfalldiagramm: M2 . . . . .	11
4.	Anwendungsfalldiagramm: P1 . . . . .	12
5.	Anwendungsfalldiagramm: E1 . . . . .	13
6.	Anwendungsfalldiagramm: D1 . . . . .	14
7.	Anwendungsfalldiagramm: D2 . . . . .	15
8.	Tabelle: Auslesen aller Lieder . . . . .	20
9.	Tabelle: Auslesen aller Lieder und Zugriff auf die Inhalte . . . . .	20
10.	Arbeitsspeicherverbrauch der Musikbibliothek . . . . .	21
11.	Tabelle: Distanzen ohne Normalisierung . . . . .	29
12.	Tabelle: Distanzen mit Normalisierung . . . . .	29
13.	Tabelle: Wertebereich bei der Normalisierung . . . . .	30
14.	Normalisierung des Temperaturunterschiedes . . . . .	30
15.	UI der TabBar ohne Miniplayer . . . . .	36
16.	UI der TabBar mit Miniplayer . . . . .	36
17.	UI des Menüs in der Musikbibliothek . . . . .	37
18.	UI der 1. Ebene der Musikbibliothek . . . . .	38
19.	UI der Suche in der Musikbibliothek . . . . .	38
20.	UI der Listeneinträge der Musikbibliothek . . . . .	38
21.	UI eines Albums in der Musikbibliothek . . . . .	39
22.	UI der Empfehlungsübersicht . . . . .	40
23.	UI der Empfehlungsdetailseite . . . . .	40
24.	UI des Musikplayers . . . . .	41
25.	UI der Zellen des Debugbereichs I . . . . .	42
26.	UI der Zellen des Debugbereichs II . . . . .	42
27.	Debug-UI der Merkmale eines Kontextes . . . . .	43
28.	Debug-UI des Vergleichs zweier Kontexte . . . . .	43

## **Quellcodeverzeichnis**

1.	Der Inhalt der bachelorarbeit-prototyp-Bridging-Header.h Datei . . . . .	45
2.	Inhalt der Podfile . . . . .	48
3.	Installieren von Pods . . . . .	48
4.	Suchanfrage an das Media Player Framework, gruppiert nach Interpreten.	49
5.	Gruppierung der <i>MPMediaItemCollections</i> nach dem Alphabet. . . . .	49
6.	Auslesen aller Alben - mit Gruppierung nach dem Alphabet . . . . .	50
7.	Auslesen aller Alben, die den Suchbegriff im Titel beinhalten. . . . .	50
8.	Festlegung der Wertebereiche der einzelnen Merkmale . . . . .	52
9.	Festlegung der Gewichtung der einzelnen Merkmale . . . . .	52
10.	Auslesen der Werte eines Merkmals für die Abbildung im Merkmalsraum	53
11.	Berechnen der einfachen Distanzen . . . . .	54
12.	Erstellen der normalisierten Distanzen für zwei Kontexte . . . . .	55
13.	Datentyp des Distanz-Vektors . . . . .	56
14.	Erstellen des Distanz-Vektors für zwei Kontexte. . . . .	56
15.	Sammeln aller ähnlichen Kontexte, die mit einer Sitzung verknüpft sind. .	58
16.	Erstellen einer Empfehlung. . . . .	59

## 1. Einführung

Einhergehend mit der Digitalisierung der Musikbibliotheken haben sich diese sowohl in ihrem Umfang als auch in ihrer Verfügbarkeit verändert. Spätestens mit der Verbreitung des iPods von Apple tragen viele Menschen nahezu ihre gesamte Musikbibliothek auf einem mobilen Gerät bei sich. Besonders umfangreiche Musikbibliotheken erschweren es dem Hörer, die Musik seiner Wahl zu finden oder überhaupt eine Wahl zu treffen. Im alltäglichen Gebrauch einer herkömmlichen Musik-App sind dabei folgende Probleme vorstellbar:

- Ist der genaue Musikwunsch noch nicht bekannt, steht der Benutzer vor der Qual der Wahl. Durchforstet er seine Musikbibliothek nach Liedern, die ihn in diesem Moment ansprechen, sind mitunter viele Interaktionen und damit Zeit nötig, bis er ein passendes Lied gefunden hat.
- Wird die Zufallswiedergabe benutzt, kann es sein, dass erst nach vielen unpassenden Liedern ein passendes abgespielt wird.
- Ist der Musikwunsch bereits bekannt, muss der Benutzer im Normalfall erst einige Interaktionen durchführen, bis das entsprechende Lied erreicht ist.

Somit kann bei der Auswahl der Musik ein hoher Aufwand entstehen.

Trotz der großen Anzahl an verfügbaren Interpreten und Liedern, ergibt es sich, dass besonders bei geregelten Tagesabläufen in ähnlichen Kontexten dieselbe Musik gehört wird. Je nach Ort, Zeitpunkt, Aktivität und Wetter kann für bestimmte Lieder oder Musikrichtungen eine Präferenz entstehen. So hört der Autor beispielsweise häufig:

- Instrumentalmusik während des Frühstücks am Wochenende
- Minimal-House unterwegs in der Bahn
- Drum’n’Bass während der Arbeit am späten Nachmittag
- Reggae an einem heißen Sommertag

Motivation dieser Arbeit ist es, die oben genannten Aufwände für Situationen, in denen in ähnlichen Kontexten dieselbe Musik gehört wird, zu reduzieren.

### 1.1. Lösungsansatz

Der Ansatz dieser Arbeit besteht darin, den Aufwand der Musikauswahl zu reduzieren, indem dem Benutzer Lieder aus seiner eigenen Musikbibliothek, die er bereits in ähnlichen Kontexten gehört hat, vorgeschlagen werden. Möchte der Benutzer nachmittags bei der Arbeit Musik hören, würden ihm Lieder, die er bereits in der Vergangenheit nachmittags bei der Arbeit gehört hat, empfohlen werden. Im Idealfall träfen diese seinen momentanen Musikwunsch und die Anzahl der Schritte, die bis zur Wiedergabe der Musik benötigt werden, wären minimiert.

Neben der Aufzeichnung der Hörgewohnheiten des Benutzers, wird dazu der vorliegende Kontext abgebildet und festgehalten. Bei der Generierung von kontextbasierten Musikempfehlungen werden diese miteinander verglichen und bei einer hohen Übereinstimmung die zugehörigen Lieder empfohlen. Die Abbildung des Kontextes wird durch das Auslesen von 28 festgelegten Merkmalen erreicht. Dazu gehören:

- Merkmale der Zeit (Tageszeit, Wochentag und Monat)
- Merkmale des Ortes (u.a. Aufenthaltsort und *WLAN*)
- Merkmale der Aktivität (Bewegungsdaten des Gerätes)
- Merkmale des Wetters (u.a. die aktuelle Temperatur)
- Merkmale des Gerätes (u.a. ob die App aktiv genutzt wird)

Da ein einzelnes Lied wenig über die Lieder aussagt, die zuvor und danach gehört wurden, werden diese bei der Auswertung in Sitzungen zusammengefasst. Diese beinhalten alle Lieder, die ab dem Start bis zum Beenden der Wiedergabe gehört wurden.

### 1.2. Zielsetzung

Ziel dieser Arbeit ist es, den Ansatz mit einer iOS App als Prototypen umzusetzen. Dabei soll neben der normalen Benutzung des Prototypen auch der Grundstein für die Weiterentwicklung und Auswertung des Lösungsansatzes gelegt werden.

### 1.3. Vorgehensweise

Nach der Vorstellung von relevanten Grundlagen zu existierenden Musikempfehlungs- und Klassifikationsverfahren, folgt die Abgrenzung zu diesen sowie die Spezifizierung der

Anforderungen an den Prototypen. Basierend auf diesen wird die Funktionalität des Prototypen und deren Benutzeroberfläche entworfen. Anhand von Code-Beispielen werden Schlüsselstellen der Implementierung beschrieben. Darauf folgend findet eine Auswertung der Qualität der Musikempfehlungen statt. Abschließend wird das Ergebnis dieser Arbeit bewertet und ein Ausblick auf Verbesserungs- und Weiterentwicklungsmöglichkeiten gegeben.

## 2. Grundlagen

Der in dieser Arbeit entworfene Lösungsansatz zur Erstellung von kontextbasierten Musikempfehlungen baut in Teilen auf bereits existierenden Musikempfehlungs- sowie Klassifikationsverfahren auf. Im Folgenden werden diese vorgestellt.

### 2.1. Musikempfehlungsverfahren

#### Klassische Verfahren zur Musikempfehlung

Bei der klassischen Musikempfehlung werden dem Benutzer – ihm noch unbekannte – Lieder empfohlen, die entweder eine Ähnlichkeit zu Liedern aufweisen, die dem Benutzer gefallen (inhaltsbasiert) oder die von Benutzern mit einem ähnlichen Musikgeschmack gemocht werden (kollaborativ) [RRS10]. Die Grundlage für die Bestimmung der Ähnlichkeit beim inhaltsbasierten Empfehlungsverfahren können u.a. Metadaten (z.B. dasselbe Genre) oder Charakteristika der Musik selbst (z.B. ein ähnliches Tempo) sein. Beim kollaborativen Empfehlungsverfahren wird dagegen das Verhalten von anderen Benutzern betrachtet. Ein prominentes Beispiel für ein solches Vorgehen ist Amazon mit der Rubrik „Kunden, die diesen Artikel gekauft haben, kauften auch“. Neben hybriden Musikempfehlungsverfahren, welche inhaltsbasierte und kollaborative Verfahren miteinander verbinden, wird zunehmend Musik abhängig vom aktuell vorliegenden Kontext vorgeschlagen. So versucht z.B. Google die Aktivität des Benutzers anhand der Uhrzeit und des benutzten Gerätes zu erkennen, um darauf basierend unterschiedliche Musik zu empfehlen [Spi15].

#### Kontextbasierte Musikempfehlungen

Die Forschungsprojekte „Logmusic“ und „Location-aware music recommendation“ verwenden eine andere Herangehensweise, indem sie manuell erstellte Musikempfehlungen mit dem aktuell vorliegenden Kontext verknüpfen. Lee und Cho entwickelten mit „Logmusic“ ein soziales Netzwerk für Android Smartphones, welches es den Benutzern ermöglicht, selbst Musikempfehlungen für den aktuell vorliegenden Kontext zu erstellen, der sich aus dem Ort, dem Wetter und der Zeit ergibt. Befinden sich andere Benutzer in einem ähnlichen Kontext, wird ihnen diese Empfehlung angeboten. In einem kurzen Test zeigte sich, dass die Benutzer die auf diesem Weg empfohlenen Lieder als passender zu ihrem eigenen Musikgeschmack und zu der jeweiligen Situation bewertet haben als

Lieder der Vergleichsgruppen, die entweder zufällig oder aus Liedern der aktuellen Charts bestimmt wurden [LC14].

Die Autoren des Papers „Location-aware music recommendation“ verfolgen den Ansatz, manuelle Musikempfehlungen mit interessanten Orten (*POIs*) zu verknüpfen. Dabei werden Lieder und *POIs* mit Tags versehen, die jeweils eine Emotion beschreiben. Integriert man diesen Ansatz in einen Reiseführer, könnte einem Touristen z.B. kraftvolle Musik empfohlen werden, wenn er den Bundestag besucht. Durch ihre Teststudie, die ergab, dass die vorgeschlagenen Lieder als etwas passender zu den *POIs* bewertet wurden als die Lieder der Vergleichsgruppe, sehen die Forscher ihren Ansatz als bestätigt an [BKR13].

Das Projekt „Mobile Music Genius“ kreiert die Musikempfehlungen dagegen anhand eines Algorithmus, der ebenfalls den aktuell vorliegenden Kontext einbezieht. Schedl, Breitschopf und Ionescu haben dazu eine Android App entwickelt, welche die Hörgewohnheiten des Benutzers inklusive des vorliegenden Kontextes aufzeichnet. Der Kontext setzt sich dabei aus circa 60 Merkmalen (Zeit, Ort, Gerätedaten, benutzte Apps, Bewegung, etc.) zusammen. Basierend auf diesen Daten ermittelt ein Server per Algorithmus des *maschinellen Lernens* das Lied, welches in dem jeweiligen Kontext am häufigsten gehört wurde. Aus diesem Lied wird dem Benutzer dann eine Wiedergabeliste mit ähnlichen Liedern als Musikempfehlung vorgeschlagen. Um zu überprüfen, wie gut sich die Musik aufgrund der Kontextmerkmale voraussagen lässt, wurden Testdaten gesammelt, um diese mit verschiedenen *maschinellen Lern-Algorithmen* auszuwerten. In einem ersten Zwischenergebnis zeigte sich, dass die Genauigkeit der Vorhersage je nach Algorithmus zwischen 15% und 55% lag [SBI14].

Unterschiede und Gemeinsamkeiten zu dem Ansatz in dieser Arbeit werden in Kapitel 3.1 herausgearbeitet.

## 2.2. Klassifikationsverfahren

Für die Generierung von kontextbasierten Musikempfehlungen müssen verschiedene Kontexte zunächst zueinander in Beziehung gestellt werden. Allgemein kann dies in der Informatik als Klassifikationsproblem bezeichnet werden. Die dafür entwickelten Klassifikationsverfahren dienen dazu, Objekte in Klassen einzurichten. Objekte innerhalb einer Klasse besitzen ähnliche Eigenschaften, die diese von Objekten aus anderen Klassen un-

terscheiden. Zu den möglichen Verfahren gehören u.a. der Entscheidungsbaum und die Nächste-Nachbarn-Klassifikation.

### Entscheidungsbaum

Mit einem Entscheidungsbaum können Objekte anhand ihrer Merkmale innerhalb eines Entscheidungspfades in festgelegte Klassen eingeordnet werden. Zu jedem Merkmal wird eine eindeutige Entscheidung getroffen und je nach Entscheidung ein anderer Folgepfad gewählt. Jedes Objekt durchläuft den Entscheidungsbaum von vorne bis zu dem Merkmal, an dem es in eine Klasse eingeordnet wird. In dem in Abbildung 1 gezeigten Beispiel wird ein Kontext  $\{WLAN: \text{deaktiviert}, \text{Temperatur}: 25^\circ C\}$  auf diese Weise der *Klasse C* zugeordnet.

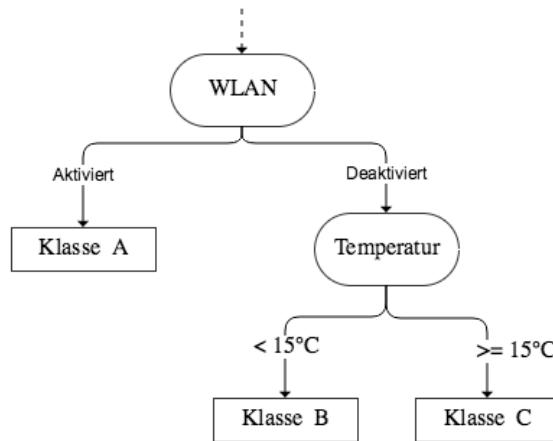


Abbildung 1: Beispiel eines einfachen Entscheidungsbaums

### Nächste-Nachbarn-Klassifikation

Die Nächste-Nachbarn-Klassifikation beschreibt Objekte anhand von Merkmalen, die durch einen Vektor repräsentiert werden. In einem Merkmalsraum werden alle Merkmalsvektoren miteinander verglichen und deren Distanz zueinander ermittelt. Basierend auf den Abständen werden nah beieinander liegende Objekte in Cluster bzw. Klassen zusammengefasst. Für die Berechnung der Distanz können verschiedene Abstandsmaße genutzt werden. Sind keine speziellen Anforderungen vorhanden, kann die simple „Cityblock-Metrik“ als erste Wahl angesehen werden. Diese gibt den Abstand an, der sich durch die Addition aller Teilabstände (zwischen den einzelnen Merkmalen) ergibt.

Im Falle der Entfernung von zwei Orten würde dies der Distanz entsprechen, die benötigt wird, um die Strecke über Straßen zurückzulegen und nicht per Luftlinie.

Inwieweit die genannten Klassifikationsverfahren in dieser Arbeit zum Einsatz kommen, wird in den Kapiteln 4.6 und 7.1 erläutert.

### 3. Anforderungsanalyse

Im Folgenden werden die Anforderungen an den Prototypen festgelegt. Nach der Abgrenzung gegenüber bereits existierenden Verfahren zur Musikempfehlung werden die Anwendungsumgebung, die Anwendungsszenarien und die zu erreichende Qualität der Musikempfehlung festgelegt sowie Aspekte des Datenschutzes erörtert.

#### 3.1. Abgrenzung

Der in dieser Arbeit entworfene Ansatz zur Erstellung von kontextbasierten Musikempfehlungen baut in Teilen auf den in Kapitel 2.1 vorgestellten Verfahren auf. Im Ganzen betrachtet, beschreitet er jedoch andere Wege. Während die klassischen Verfahren Musik empfehlen, von der angenommen wird, dass diese dem Benutzer noch unbekannt ist, liegt der Fokus dieser Arbeit auf Musik, die der Benutzer bereits besitzt und gehört hat. Eine Betrachtung des Musikgeschmacks von anderen Benutzern sowie die Auswertung der Audiodateien selbst ist für die erste Version des Prototypen nicht vorgesehen. Die Auswahl der empfohlenen Lieder basiert somit allein auf den Hörgewohnheiten des jeweiligen Benutzers.

Im Gegensatz zu „Logmusic“ und „Location-aware music recommendation“ wird auf manuelle Eingaben des Benutzers verzichtet. Alle betrachteten Informationen werden automatisch von dem Prototypen im Hintergrund gesammelt. Bis auf eine einmalige Zugriffsbestätigung zur Verwendung des Aufenthaltsortes und der Bewegungsdaten sowie dem Musikhören selbst, sind keine Interaktionen des Benutzers nötig. Die Auswahl der Merkmale, welche einen Kontext abbilden, ähnelt denen des „Mobile Music Genius“. Die Merkmale werden jedoch ohne *maschinelles Lernen* und auf dem Gerät selbst (und nicht auf einem ausgelagerten Server) ausgewertet.

#### 3.2. Anwendungsumgebung

Neben dem normalen Benutzen des Prototypen auf kompatiblen Smartphones und MP3-Playern, soll sie internen Benutzern das Testen sowie Nachvollziehen der Musikempfehlungen ermöglichen. Im Detail ergeben sich daraus folgende Nutzungen:

- Einsehen und Abspielen der lokal verfügbaren Musik
- Einsehen und Abspielen von Musikempfehlungen

- Einsehen von Debuginformationen
- Ausführen von Debugaktionen

Bei der Benutzung sowie (Weiter-)Entwicklung des Prototypen kann zwischen normalen und internen Benutzern unterschieden werden:

### **Normale Benutzer**

Normale Benutzer verwenden den Prototypen, um Musik zu hören und die Musikempfehlungen einzusehen. Neben den Endnutzern würden dazu auch Probanden einer Teststudie gehören.

### **Interne Benutzer**

Interne Benutzer sind zusätzlich an Hintergrundinformationen interessiert, die es ermöglichen, den technischen Ablauf der Musikempfehlung nachzuvollziehen bzw. zu überprüfen. Dazu gehören Debug- und Metainformationen wie die Werte des aktuellen Kontextes und der protokollierten Kontexte. Neben dem reinen Einsehen wollen interne Benutzer auch Daten importieren sowie zurücksetzen. Zu den internen Benutzern gehören Entwickler, Betatester oder Forscher, die eine Teststudie durchführen.

#### **3.2.1. Rahmenbedingungen**

Als Ausgangsbasis für die Entwicklung des Prototypen sind folgende Rahmenbedingungen festgelegt:

- Als Plattform wird iOS verwendet.
- Da die Musikbibliothek des Prototypen aus den Liedern bestehen soll, die sich bereits auf dem Gerät befinden, ist es für den Gebrauch des Prototypen notwendig, dass der Benutzer Musik mit iTunes oder der Apple Music App synchronisiert hat.
- Der Anspruch dieser Arbeit ist es, einen möglichst realitätsnahen Prototypen zu entwickeln. So sollen grundsätzlich die Kriterien für eine Veröffentlichung im App Store von Apple bereits von dem entwickelten Prototypen erfüllt werden, da dies der einzige offizielle Weg ist, auf dem iPhone eine Drittanbieter App zu beziehen [App08]. Insbesondere für die Erfassung des Kontextes ist hervorzuheben, dass die

Verwendung von privaten *APIs* untersagt ist [App15a]. Bei diesen handelt es sich um *APIs*, die zwar vorhanden sind und von iOS verwendet werden aber nicht für die Benutzung durch Dritte freigegeben sind. Meta-, Umgebungs- sowie Systemdaten können somit nur über offizielle Schnittstellen abgefragt werden.

### 3.3. Anwendungsszenarien

Der Prototyp kann in vier Bereiche aufgeteilt werden: Die Musikbibliothek, den Player, die Musikempfehlungen sowie den Debugbereich für interne Benutzer. Für die weitere Umsetzung des Prototypen ist es hilfreich, dezidierte Anwendungsfälle festzulegen. Folgend werden die sechs Anwendungsfälle für die einzelnen Bereiche vorgestellt.

#### 3.3.1. Musikbibliothek

##### M1: Einsehen der Inhalte der Musikbibliothek

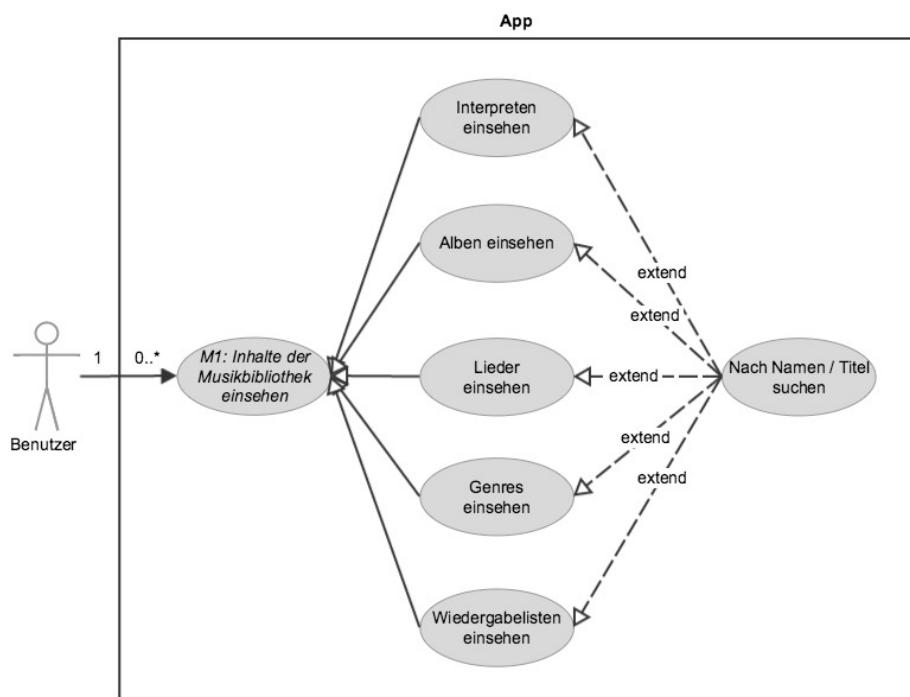


Abbildung 2: Anwendungsfalldiagramm - M1: Einsehen der Inhalte der Musikbibliothek

Um mit dem Prototypen Musik zu hören, müssen die Benutzer die Musikbibliothek einsehen können. Wie im Anwendungsfalldiagramm [Abb. 2] dargestellt, soll es möglich sein, diese nach den Kategorien Interpreten, Alben, Lieder, Genres und Wiedergabelisten einzusehen. Die Inhalte der Kategorien sollen sich nach dem jeweiligen Namen bzw. Titel durchsuchen lassen.

### **M2: Abspielen von lokal verfügbaren Inhalten der Musikbibliothek**

Alle lokal verfügbaren Medien sollen abgespielt werden können. Dazu zählen auch die Musikempfehlungen. Nachfolgende Lieder werden dabei automatisch der Warteschlange hinzugefügt.

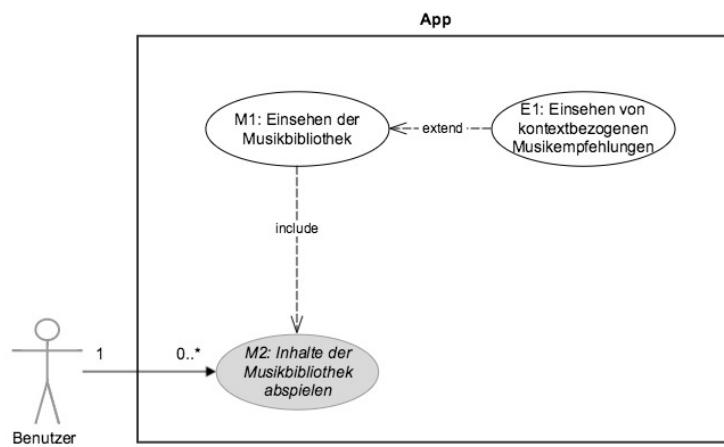


Abbildung 3: Anwendungsfalldiagramm - M2: Abspielen von lokal verfügbaren Inhalten der Musikbibliothek

#### **3.3.2. Musikplayer**

##### **P1: Aktuelle Musikwiedergabe steuern**

Wird aktuell Musik wiedergegeben oder befinden sich Musiktitel in der Warteschlange, soll die Wiedergabe gesteuert werden können. Neben dem Pausieren und Starten, dem Spulen innerhalb des Liedes und dem Springen zum vorherigen oder nachfolgenden Lied gehört dazu auch das Anpassen der Lautstärke [Abb. 4].

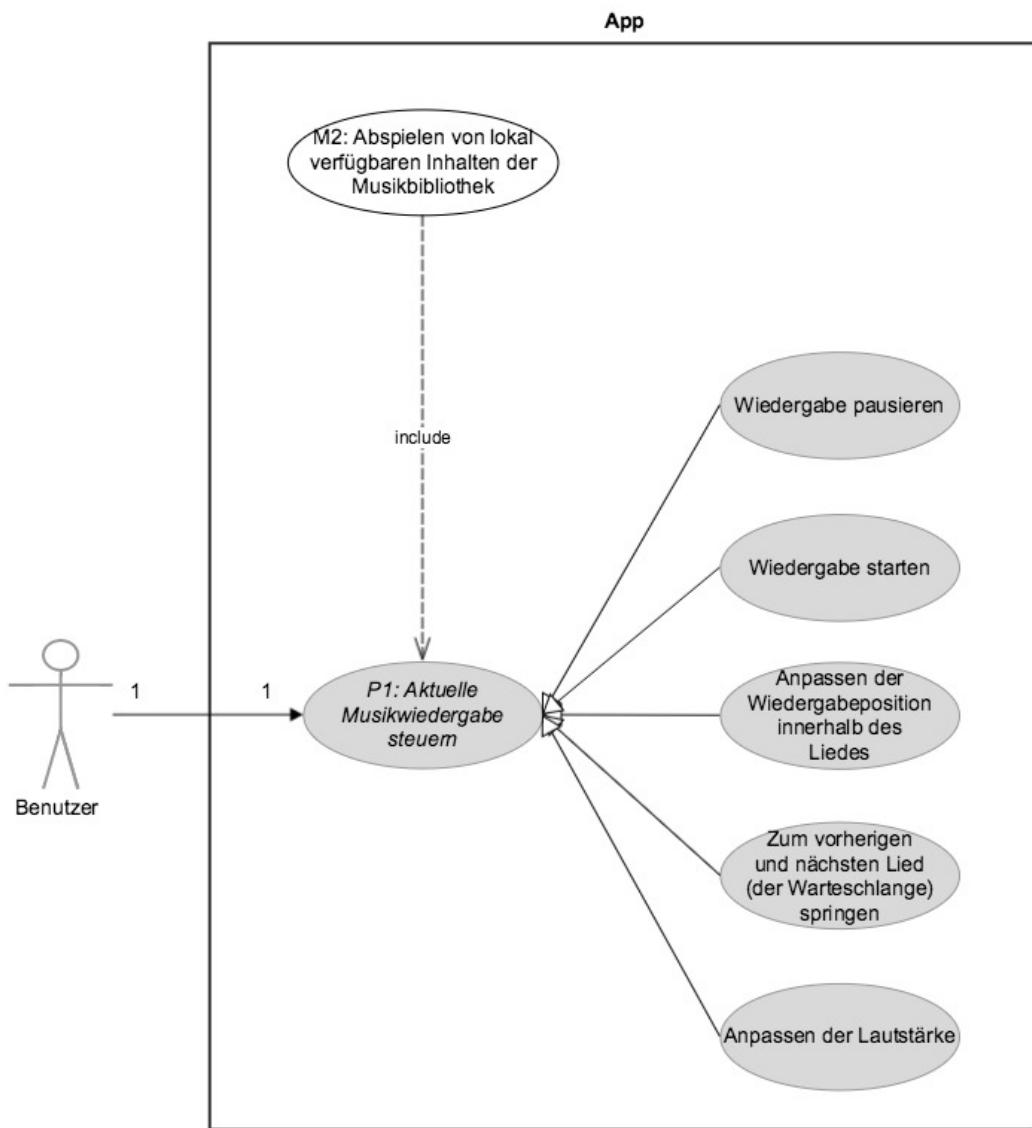


Abbildung 4: Anwendungsfalldiagramm - P1: Aktuelle Musikwiedergabe steuern

### 3.3.3. Musikempfehlung

#### E1: Einsehen von kontextbezogenen Musikempfehlungen

Als Alternative zum Durchsuchen der Musikbibliothek soll der Benutzer kontextbezogene Musikempfehlungen einsehen können. Die Empfehlung bezieht sich dabei auf eine vergangene Sitzung bzw. eine Gruppe von Sitzungen und nicht auf einzelne Lieder. Zusätzlich

zu Empfehlungen, die durch den Vergleich von allen Merkmalen eines Kontextes ermittelt wurden, sollen auch Empfehlungen angezeigt werden, die nur auf einzelnen Merkmalsgruppen (zum Beispiel nur den Wetterdaten) basieren. Dadurch ist es auch möglich, dass Sitzungen bzw. Lieder mehrmals empfohlen werden. Zum Verständnis über den Grund der Empfehlung soll die jeweilige Art der zutreffenden Merkmalsgruppen und ein Auszug der übereinstimmenden Werte angezeigt werden [Abb. 5].

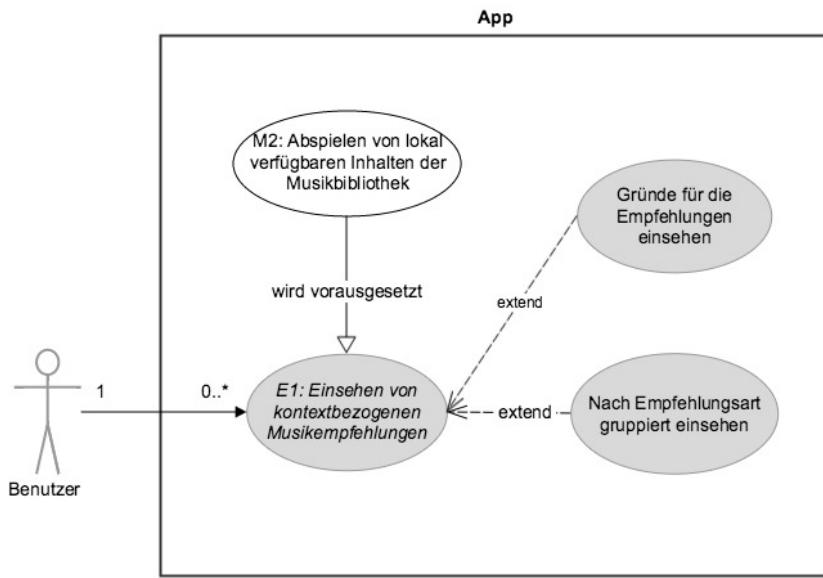


Abbildung 5: Anwendungsfalldiagramm - E1: Einsehen von kontextbezogenen Musikempfehlungen

### 3.3.4. Debugbereich

#### D1: Einsehen von Debuginformationen

Internen Benutzern sollen in einem zusätzlichen Bereich Debuginformationen bereitstellen. Diese dienen dazu, die Inhalte des Wiedergabeprotokolls zu untersuchen und damit die Funktionalität des Prototypen nachzuvollziehen. Einsehbar sind:

- der Inhalt des Wiedergabeprotokolls, d.h. alle protokollierten Sitzungen inklusive der einzelnen Lieder und der dazugehörigen Kontexte
- die Metadaten des aktuellen Kontextes
- die Metadaten aller in der Datenbank protokollierten Kontexte

- die Metadaten der protokollierten Kontexte, welche dem aktuellen Kontext ähnlich sind
- der Vergleich zweier Kontexte

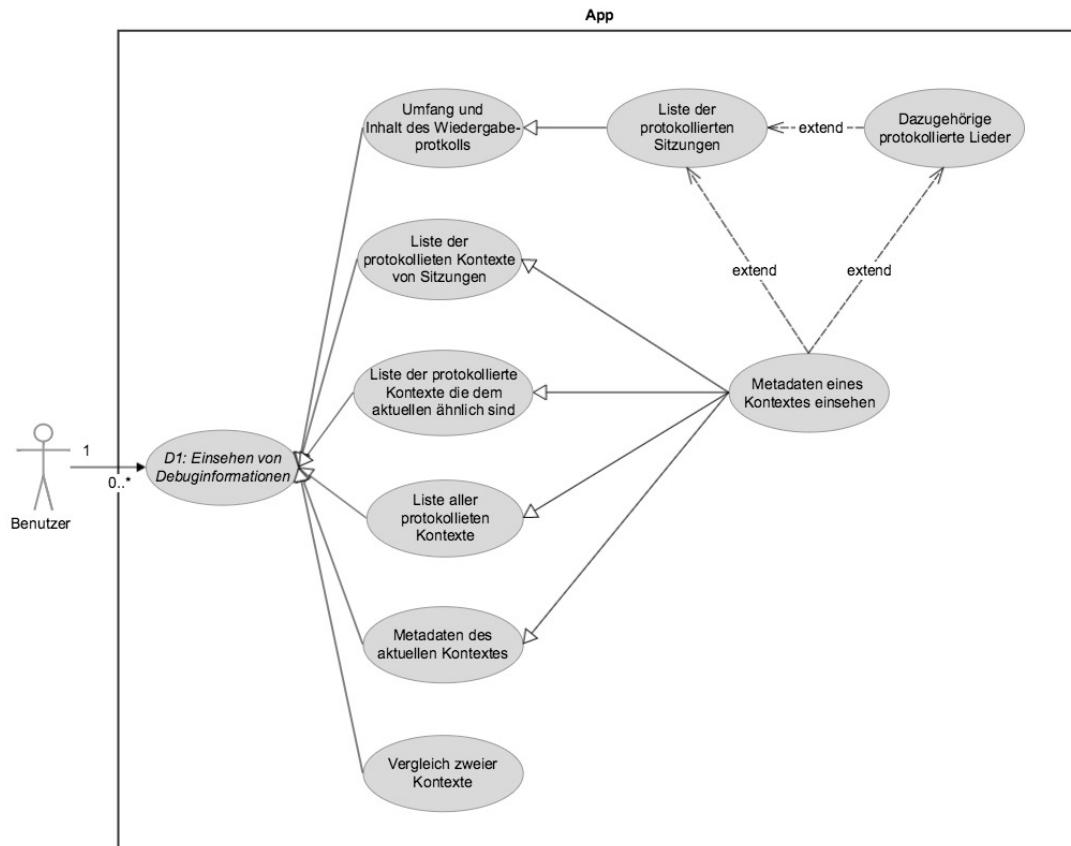


Abbildung 6: D1: Einsehen von Debuginformationen

## D2: Ausführen von Debugaktionen

Zusätzlich zum Einsehen von Debuginformationen soll es möglich sein, folgende Debugaktionen auszuführen:

- das Anpassen der Mindestähnlichkeit, die für Empfehlungen, die auf allen Merkmalen eines Kontextes beruhen, nötig ist
- der Export des Wiedergabeprotokolls
- der Import eines Wiedergabeprotokolls

- das Zurücksetzen des gesamten Wiedergabeprotokolls

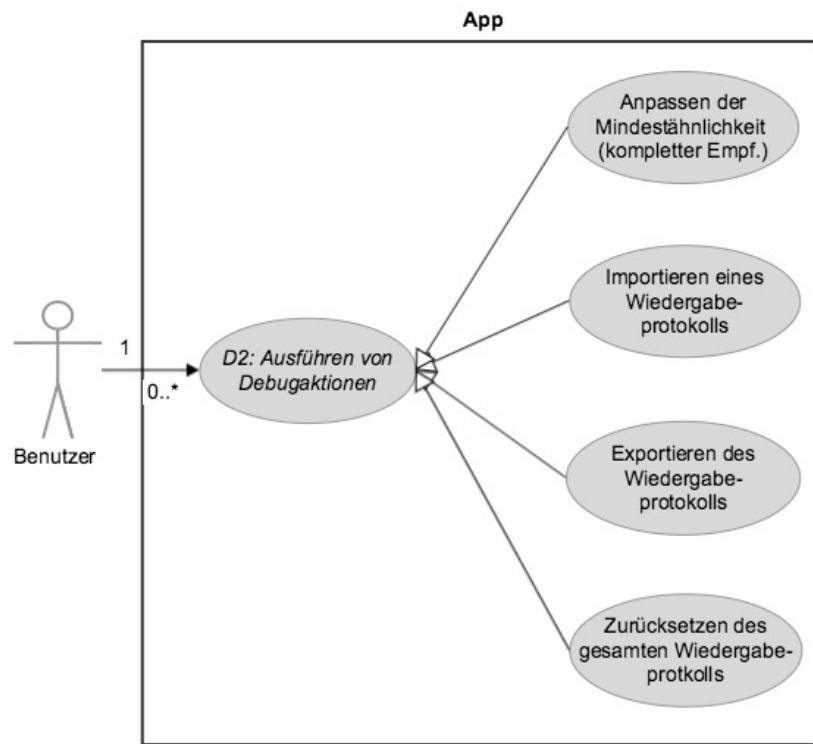


Abbildung 7: Anwendungsfalldiagramm - D2: Ausführen von Debugaktionen

### 3.4. Qualität der Musikempfehlungen

Wie in Kapitel 1.1 beschrieben, erfüllen die kontextbasierten Musikempfehlungen nur dann ihren Zweck, wenn in ähnlichen Kontexten dieselbe Musik gehört wird. Liegt eine solche Situation nicht vor, wird kein Anspruch darauf erhoben, dass die Musikempfehlungen den momentanen Musikwunsch des Benutzers treffen. Inwieweit und in welchem Ausmaß solche Situationen für alle Benutzer vorliegen, wird in dieser Arbeit nicht untersucht. Der Prototyp kann aber dazu genutzt werden, Daten, die für eine Teststudie nötig sind, zu sammeln.

Die Qualität der Musikempfehlungen kann somit durch den Erfolg der Kontexterkennung bestimmt werden. Basieren die Empfehlungen auf Kontexten, die von dem Benutzer als ähnlich betrachtet werden, wird dies als Erfolg gewertet.

### 3.5. Datenschutz

Für das Abbilden des Kontextes ist es notwendig, Daten der aktuellen Umgebung des Gerätes zu erfassen. Ist die Identität der Gerätebesitzer bekannt, handelt es sich dabei um personenbezogenen Daten, die Rückschlüsse auf vergangene Aufenthaltsorte und Aktivitäten der Benutzer ermöglichen. Um Verletzungen der Privatsphäre zu vermeiden, sollen die Daten grundsätzlich nur auf den Geräten der Benutzer gespeichert werden. Ohne das ausdrückliche Exportieren und Versenden der Daten wird es innerhalb des Prototypen keine Möglichkeit geben, dass die Daten das Gerät verlassen. Der unberechtigte Zugriff auf die Datenbank des Prototypen durch andere Apps wird von iOS durch das App Sandboxing unterbunden. Dieses ist Bestandteil der Maßnahmen, welche die Sicherheit von iOS garantieren sollen. Für jede App wird dabei ein spezieller Bereich im Dateisystem – die sogenannte Sandbox – bereitgestellt [App15c]. Der Zugriff auf diese ist nur durch die jeweilige App möglich. Der Austausch von Daten zwischen Apps oder mit dem Betriebssystem ist nur über APIs möglich, die von diesen bereitgestellt werden. Entwickler können so den Zugriff auf die Daten ihrer Apps kontrollieren und einen Missbrauch durch Dritte verhindern.

Somit bleiben folgende potenzielle Schwachstellen:

- die Benutzer selbst, falls anderen der Zugang zum Prototypen ermöglicht wird; durch die kontextbezogenen Musikempfehlungen können Rückschlüsse auf das Vorhandensein von ähnlichen Kontexten gezogen werden
- das Auslesen der Datenbank, falls Angreifer während der Benutzung oder auf einem unverschlüsselten Gerät, Zugriff auf das Dateisystem erlangen sollten
- das fehlen eines Passcodes; wird kein Passcode benutzt, sind die Daten auf dem Gerät nicht verschlüsselt [App15g], dadurch wird ein Einbruch in das Dateisystem und somit der Zugriff auf die Datenbank erleichtert

Eine geeignete Maßnahme, um die Datenbank auch bei einem Einbruch in das Dateisystem zu schützen, wäre die Verschlüsselung der Datenbankeinträge selbst. Dies wäre z.B. sinnvoll wenn Passwörter oder ähnlich hoch sensible Daten gespeichert werden müssten. Da die Daten, welche von dem Prototypen erfasst werden, nicht dieser Schutzbedürftigkeit entsprechen, wird dies vorerst als zweitrangig erachtet.

## 4. Systementwurf

Die Funktionalität des Prototypen entspricht den in Kapitel 3 festgelegten Anforderungen. Auf diesen aufbauend, wird nun der Systementwurf des Prototypen entwickelt. Darauf folgend wird mit der Ermittlung des Kontextes, dem Erfassen der Hörgewohnheiten, der Berechnung der Ähnlichkeit von Kontexten sowie der Generierung der Musikempfehlungen ausführlich auf die Umsetzung des Lösungsansatzes eingegangen. Abschließend wird die Benutzeroberfläche des Prototypen entworfen. Die Beachtung der unterschiedlichen Anforderungen, durch normale und interne Benutzer, wird mit dem Bereitstellen von zwei App Versionen erreicht. Diese unterscheiden sich lediglich durch die ein- bzw. ausgeblendeten Debugfunktionen. Obwohl somit zwei eigenständige Apps entstehen, werden diese im weiteren Verlauf weiterhin als eine App betrachtet, da beide aus dem selben Quellcode bestehen und sich die Funktionen, die jeweils zugänglich sind, nicht unterscheiden.

### 4.1. Systemanforderungen

Bei dem Prototypen handelt es sich um eine iOS App, welche mit der Programmiersprache Swift umgesetzt wird. Die älteste unterstützte Betriebssystemversion ist die aktuelle Version iOS 9. Die Eingrenzung auf die aktuellste Version erlaubt es, alle neuen Funktionen des *SDKs* zu benutzen. Mit einer momentanen Verbreitung von 75% auf allen aktiven Geräten [App16] wird außerdem eine ausreichend hohe Anzahl an Benutzern erreicht. Prinzipiell ist der Prototyp auf allen Geräten mit einer unterstützten iOS Version lauffähig. Eine angepasste Benutzeroberfläche wird aber nur für das iPhone und den iPod, aber nicht für das iPad implementiert.

Um die optimale Kontexterkennung zu erzielen, müssen folgende Voraussetzungen erfüllt sein:

- ein Gerät mit Bewegungskoprozessor, d.h. iPhones ab dem iPhone 5S, iPads ab dem iPad Air sowie dem iPad Mini 2 und iPods ab der 6. Generation [Wik16]
- der Zugriff auf die Bewegungsdaten muss vom Benutzer erlaubt werden
- ein Gerät mit *GPS*-Modul, d.h. alle unterstützten iPhones und iPads mit Funkmodul [Wik16]

- der Zugriff auf die Ortungsdienste mit Hintergrundberechtigung muss vom Benutzer erlaubt werden
- eine funktionierende Internetverbindung; diese wird für das Abfragen der Wetterverhältnisse benötigt

## 4.2. Schnittstellen

Neben der Verwendung von Standard Funktionen des iOS *SDKs* werden folgende Schnittstellen angebunden:

- das iOS Media Player Framework als Quelle der Musikbibliothek. Dieses ermöglicht das Abspielen von Audio- und Videodateien sowie den Zugriff auf die iPod Bibliothek [App14]. Die Bibliothek funktioniert zugleich als Datenquelle der Apple Music App und beinhaltet alle über iTunes oder iCloud synchronisierten Medien. Somit bietet das Media Player Framework den kompletten Zugriff auf den Inhalt der Apple Music App. Auch auf Lieder, deren Metadaten dem Gerät bekannt sind, aber deren Audiodateien nicht lokal vorliegen. Diese, sowie Lieder, die mit einem Kopierschutz versehen sind, können zwar angezeigt, aber nicht abgespielt werden.
- das iOS Core Location Framework zur Bestimmung des Aufenthaltsortes. Dieses bietet APIs, die den Zugriff auf die Daten des aktuellen Aufenthaltsortes des Benutzers ermöglichen [App15f]. Neben den Geokoordinaten beinhalten diese Daten auch Informationen über die Bewegungsrichtung sowie die Genauigkeit und Aktualität der Daten. Als Quelle für die Positionsbestimmung werden – je nach Gerät – das GPS-Modul, das WLAN-Modul, das Bluetooth-Modul sowie das Funkmodul genutzt.
- das iOS Core Motion Framework zur Bestimmung der Aktivität. Dieses bietet Zugriff auf die Daten der Bewegungssensoren (Beschleunigungsmesser, Gyroskop, Kompass, etc.), welche von einem Bewegungskoprozessor gesammelt und ausgewertet werden [App13]. Durch die Auslagerung aus dem Hauptprozessor kann der Stromverbrauch, trotz regelmäßiger Abfrage der Bewegungssensoren, reduziert werden [Wik15]. Basierend auf den Bewegungsdaten kann das System die Aktivitäten Stationär, Gehen, Rennen, Auto, Radfahren und Unbekannt erkennen und vermerken. Der Zugriff auf diese Aktivitäten (CMMotionActivities) kann mit dem

CMMotionActivityManager erfolgen. Mit diesem ist das permanente Zugreifen auf aktuelle Informationen zu den Bewegungsarten ebenso möglich wie das nachträgliche Abfragen für einen festgelegten Zeitraum [App15b].

- die Yahoo Wetter *API* zur Bestimmung des Wetters. Diese ist ein Webservice, welcher das kostenlose Abfragen von Wetterdaten ermöglicht [Yah]. Anhand der Koordinaten des aktuellen Standortes werden die aktuellen Wetterverhältnisse sowie die Prognose für den aktuellen Tag abgefragt.

### 4.3. Anbindung der lokalen Musikbibliothek

Bei der Anbindung der lokalen Musikbibliothek sind zwei Verfahren möglich: Der Import des Inhalts des Media Player Frameworks in eine eigene Datenbank (oder ähnliches) und der permanente Zugriff auf das Media Player Framework. Folgend werden deren Vorteile sowie Nachteile abgewogen und die Entscheidung über das gewählte Verfahren getroffen sowie begründet.

#### Import in eine eigene Datenbank

Der Import in eine eigene Datenbank bietet u.a. folgende Vorteile:

- Die Inhalte können nach eigenen Kriterien durchsucht und Beziehungen zueinander erstellt werden. Da das Media Player Framework nur *MPMediaItems* oder *MPMediaItemCollections* zurückgibt, bietet es kein Datenmodell für Interpreten, Alben oder Genres. Stattdessen sind die jeweiligen Informationen in den *MPMediaItems* selbst abgebildet. Möchte man z.B. alle Alben oder Lieder eines Interpreten erhalten, muss dies entweder mit einer Suchanfrage nach dem Interpretennamen und der Gruppierung nach Alben erfolgen oder durch eine eigene Logik, welche die Lieder nach dem Albumtitel unterteilt.
- Werden die Inhalte aus der Datenbank abgefragt, kann der Gebrauch an Arbeitsspeicher durch die Verwendung des *NSFetchedResultsController* reduziert werden.

Dem gegenüber stehen folgende Nachteile:

- Die Suchanfrage an das Media Player Framework selbst ist zwar sehr schnell, allerdings werden dabei vermutlich nicht alle Attribute der *MPMediaItems* geladen. Da Apples Implementierung nicht öffentlich einsehbar ist, kann diese These nicht

nachgewiesen werden, doch die Beobachtungen des Verhaltens der *API* legen diesen Rückschluss nahe. Um dies zu demonstrieren, wurde die *API* während der Implementierung auf einem iPhone 6 mit einer Musikbibliothek, welche 10439 *MPMediaItems* enthält, getestet. Dabei hat sich gezeigt, dass die erste Abfrage, die das Array an *MPMediaItems* zurückgibt, im Durchschnitt 0,23196 Sekunden benötigt und nachfolgende Abfragen durchschnittlich nur noch 0,00004 bis 0,00016 Sekunden [Abb. 8]. Wird durch alle Lieder iteriert und auf den Titel des Liedes, des Albums und des Interpreten zugegriffen, dauert der erste Durchlauf mit 2,18108 Sekunden vergleichsweise sehr lange. Bei einer Wiederholung dieser Prozedur ohne vorheriges Schließen des Prototypen benötigt der Durchlauf nur noch 0,06433 bis 0,06727 Sekunden [Abb. 9]. Da dieser Unterschied in allen Tests aufgetreten ist, liegt die Vermutung nahe, dass zumindest ein Teil der Attribute des *MPMediaItems* erst bei Bedarf nachgeladen wird. Damit würde allein das Auslesen der Daten bei einem Import in die Datenbank schon mehrere Sekunden dauern.

	Durchlauf 1	Durchlauf 2	Durchlauf 3	Durchlauf 4	Durchlauf 5	$\emptyset$
<b>1. Abfrage</b>	0,24135	0,26303	0,26252	0,13025	0,26263	<b>0,23196</b>
<b>2. Abfrage</b>	0,00010	0,00008	0,00040	0,00009	0,00011	<b>0,00016</b>
<b>3. Abfrage</b>	0,00002	0,00002	0,00010	0,00003	0,00002	<b>0,00004</b>
<b>4. Abfrage</b>	0,00006	0,00002	0,00010	0,00002	0,00003	<b>0,00005</b>
<b>5. Abfrage</b>	0,00002	0,00002	0,00009	0,00002	0,00003	<b>0,00004</b>

Abbildung 8: Messergebnisse für das Abfragen von 10439 *MPMediaItems*.

	Durchlauf 1	Durchlauf 2	Durchlauf 3	Durchlauf 4	Durchlauf 5	$\emptyset$
<b>1. Abfrage</b>	2,19273	2,23427	2,27692	2,15142	2,05007	<b>2,18108</b>
<b>2. Abfrage</b>	0,06473	0,06435	0,06591	0,07430	0,06704	<b>0,06727</b>
<b>3. Abfrage</b>	0,06497	0,06446	0,06319	0,06686	0,06665	<b>0,06523</b>
<b>4. Abfrage</b>	0,06458	0,06361	0,06340	0,06391	0,06614	<b>0,06433</b>
<b>5. Abfrage</b>	0,06463	0,06499	0,06399	0,06472	0,06570	<b>0,06481</b>

Abbildung 9: Messergebnisse für das Abfragen von 10439 *MPMediaItems* inklusive des Zugriffs auf die Interpretennamen sowie die Titel der Alben und Lieder.

### Permanente Anfrage an das Media Player Framework

Werden die Inhalte nicht importiert, sondern das Media Player Framework bei Bedarf erneut angefragt, ist die Performance für den Echtzeitbetrieb geeignet. Ohne die oben erwähnte Logik nehmen die Inhalte der Musikbibliothek allerdings viel Platz im Arbeitsspeicher ein. Wird in der Musikbibliothek mit 10439 *MPMediaItems* bis zum Ende gescrollt, steigt der verbrauchte Arbeitsspeicher permanent an und liegt im Bereich von mehreren hundert MB. Dieser hohe Verbrauch ist vor allem den Albumcovern geschuldet. Wird nicht auf diese zugegriffen, liegt der Arbeitsspeicherverbrauch bei circa 35 MB. Die Referenz zu den Covern wird in den *MPMediaItems* gehalten. Eine Optimierung des Arbeitsspeicherverbrauchs ist somit nicht möglich. Gibt iOS eine Warnung aus, dass zu wenig Arbeitsspeicher zur Verfügung steht, gibt das *MPMediaItem* den Speicher der nicht sichtbaren Bilder aber frei [Abb. 10].

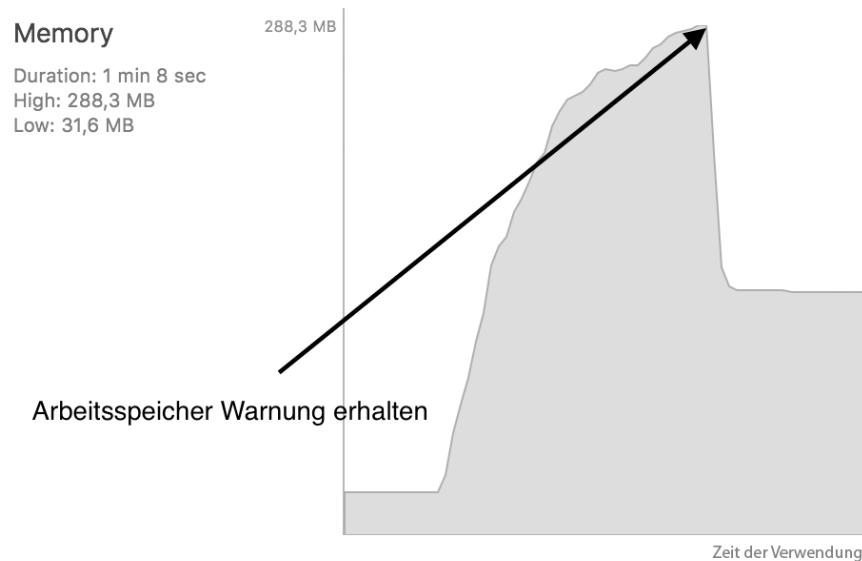


Abbildung 10: Übersicht über den Verbrauch des Arbeitsspeichers. Es wurde durch die Liste von 10439 Liedern gescrollt. Der Arbeitsspeicherverbrauch steigt dabei immer weiter an. Zum Zeitpunkt der Arbeitsspeicherwarnung wird ein Großteil des Speichers wieder freigegeben.

### Fazit

Die Vorteile eines Imports in die appinterne Datenbank sind zwar bedeutsam, allerdings ist die damit einhergehende sehr schlechte Performance für eine mobile App als Aus-

schlusskriterium zu werten. Somit wird für den Prototypen ein permanenter Zugriff auf das Media Player Framework implementiert.

#### 4.4. Ermittlung des Kontextes

Wie bereits in der Abgrenzung, Kapitel 3.1, erwähnt, wird bei der Ermittlung des Kontextes auf manuelle Eingaben des Benutzers verzichtet. Um ein möglichst simples und unkompliziertes Benutzererlebnis zu erreichen, werden so alle Merkmale eines Kontextes automatisch im Hintergrund von dem Prototypen selbst gesammelt.

Die Auswahl dieser wird durch die von iOS angebotenen Schnittstellen begrenzt. Eine Erfassung von Apps, die während des Musikhörens verwendet werden, wäre zwar ein interessantes Merkmal. Da iOS jedoch keine offiziellen Möglichkeiten bietet, auf diese Informationen zuzugreifen, ist dies nicht umsetzbar. Bei der Auswahl der Merkmale spielen zwar Vermutungen über deren Bedeutung eine Rolle; die Verwendung innerhalb des Prototypen erfolgt aber unvoreingenommen. Ob z.B. eine hohe Temperatur als gutes oder schlechtes Wetter angesehen wird, ist nicht von Belang, da die Werte nicht eingeordnet werden, sondern nur deren Ähnlichkeit zueinander betrachtet wird. Hört der eine Benutzer immer um 19 Uhr ein Schlaflied und ein anderer immer um 24 Uhr, würden beide zu der jeweiligen Zeit das Schlaflied vorgeschlagen bekommen.

Neben den Kontexten, die jeweils einen einzelnen Zeitpunkt widerspiegeln, werden in dem Prototypen Kontexte angelegt, die mehrere Kontexte zu einem zusammenfassen. Die Art der Merkmale ändert sich dabei nicht, lediglich der Datentyp der Werte kann sich ändern. Alle Merkmale, welche als Zahl oder boolescher Wert vorliegen, werden durch deren Mittelwert abgebildet. Andere Datentypen (z.B. Zeichenketten) werden durch ein Array, welches deren Werte enthält, abgebildet.

##### 4.4.1. Merkmale zur Abbildung eines Kontextes

Nach Untersuchung der vorhandenen Schnittstellen, Sensoren und Informationen, die für die Bestimmung von Merkmalen interessant sein könnten, wurden die unten genannten Merkmale für die Abbildung des Kontextes festgelegt. Diese 28 Merkmale lassen sich in die Merkmalsgruppen Ort, Tageszeit, Wochentag, Monat, Wetter, Aktivität und Gerätedataen unterteilen.

## Ort

- **Aufenthaltsort:** Dieser wird durch die Geokoordinaten, die das Core Location Framework liefert, gebildet.
- **Genauigkeit des Aufenthaltsortes:** Neben den Geokoordinaten wird auch die Genauigkeit, mit der das Core Location Framework den Aufenthaltsort bestimmt hat, als Dezimalzahl erfasst. Diese variiert je nach Art der Lokalisierung (*GPS*, *WLAN*, Telefonnetz, etc.). Da Orte – z.B. durch fehlenden *GPS*-Empfang in Gebäuden – unterschiedliche Genauigkeiten aufweisen können, wird diese als Merkmal des Ortes angesehen.
- **Land:** Da der Aufenthaltsort für den präzisen Ortsvergleich und keine allgemeine Distanzberechnung genutzt wird, wird zusätzlich das Land als Zeichenkette erfasst. Damit soll es möglich sein, Reisen über die Ländergrenze hinweg zu erfassen.
- **WLAN-Name (SSID):** Neben dem Aufenthaltsort wird zusätzlich die *WLAN SSID* als Zeichenkette gespeichert. Befindet sich der Benutzer zum Beispiel an dem Campus Treskowallee, würde der Aufenthaltsort nicht mit dem Campus Wilhelmshof übereinstimmen. Beide Orte verbindet aber, dass sie zu derselben Institution, nämlich der HTW Berlin, gehören und über *WLANs* mit derselben *SSID* „eduroam“ verfügen. Mit der *WLAN SSID* ist so eine Zuordnung zu der HTW Berlin bzw. der Hochschul-/Universitätsumgebung möglich. Da viele Institutionen über einheitliche *SSIDs* verfügen, können diese als hilfreiches Merkmal für die Art des Aufenthaltsortes gesehen werden.

## Tageszeit

Als präzisestes Merkmal der Zeit wird die Tageszeit als ganze Zahl ermittelt. Dazu wird die aktuelle Stunde des Tages (24-Stunden-Zählung) erfasst und durch die Betrachtung der Minutenanzahl entschieden, ob auf- oder abgerundet wird. Die stündliche Einordnung soll es ermöglichen, Situationen zu erkennen, die oft zu denselben Uhrzeiten stattfinden.

## Wochentag

Neben der Einordnung innerhalb eines Tages, wird der Wochentag als ganze Zahl erfasst, um Situationen zu erkennen, die sich an bestimmten Tagen innerhalb einer Woche wiederholen, z.B. am Beginn der Woche oder am Wochenende.

### Monat

Um auch Jahreszeiten beachten zu können, wird zusätzlich der Monat als ganze Zahl erfasst.

### Wetter

Da Stimmung und Aktivität der meisten Menschen und damit auch die gehörte Musik vom Wetter beeinflusst wird, werden zum Kontext auch Merkmale des Wetters gezählt.

- Die **aktuelle Temperatur**, **Tageshöchsttemperatur** sowie **Tagestiefsttemperatur** werden als Dezimalzahlen in Grad Celsius erfasst.
- Die **Windgeschwindigkeit** wird als Dezimalzahl in Kilometern pro Stunde erfasst.
- Die **Luftfeuchtigkeit** wird als Dezimalzahl in Prozent erfasst.
- Die **Sichtweite** wird als Dezimalzahl in Kilometern erfasst.
- **Tageslicht:** Ist die Sonne schon auf- und noch nicht untergegangen, herrscht Tageslicht. Diese Information wird als boolescher Wert festgehalten.
- **Wetterlage:** Neben den präzisen Wetterdaten wird die Beschreibung der Wetterlage des Wetterdienstes von Yahoo als Zeichenkette erfasst. Es werden 48 zur Verfügung stehende Wetterlagebeschreibungen verwendet, dazu gehören z.B. „sunny“, „heavy snow“ und „mostly cloudy (day)“.

### Aktivität

Basierend auf den Daten des Bewegungskoprozessors wird die Aktivität des Benutzers erfasst. Jede Kategorie (**Stationär**, **Gehen**, **Rennen**, **Auto**, **Radfahren**, **Unbekannt**) wird dabei als einzelnes Merkmal abgebildet. Um eine höhere Genauigkeit der Bewegungsdaten zu erzielen, wird die Schnittstelle mehrmals hintereinander abgefragt. Aus den ursprünglich booleschen Werten der einzelnen Kategorien ergibt sich so eine Dezimalzahl, die jeweils dem Mittelwert aus den einzelnen Abfragen entspricht.

### Gerätedaten

Unter Gerätedaten werden Informationen über die Einstellungen und die Benutzung des Gerätes verstanden.

- **Musikausgangstyp:** Mit der AVAudioSessionPortDescription ist es möglich, die Art des Musikausgangstyps zu erfahren. Diese wird als Zeichenkette erfasst. Die Art des Musikausgangs ist als Merkmal interessant, da z.B. das Hören mit Kopfhörern darauf hinweisen kann, dass der Benutzer allein Musik hört, während die Verwendung von AirPlay eher auf das Hören innerhalb einer Gruppe hindeutet.
- **Befindet sich die App im Vordergrund:** Befindet sich die App im Vordergrund, d.h. sie ist geöffnet und nimmt den kompletten Bildschirm ein, kann davon ausgegangen werden, dass sie aktiv benutzt und somit keine andere App verwendet wird. Ob dieser UIApplicationState vorherrscht, wird als boolescher Wert erfasst.
- **Ist das Gerät gesperrt:** Wird von dem Benutzer ein Passcode benutzt, um das Gerät zu sichern, lässt sich herausfinden, ob dieser aktiviert ist. Der boolesche Wert entspricht dabei eins, wenn der Passcode aktiv ist (und sich das Gerät im gesperrten Zustand befindet) und null, wenn generell kein Passcode verwendet wird bzw. das Gerät momentan nicht gesperrt ist. Bei der Benutzung eines Passcodes lässt sich somit herausfinden, ob das Gerät benutzt wird oder nicht.
- **Internet über Telefonnetz:** Ob die Internetverbindung über das Telefonnetz läuft und das Netz erreichbar ist, wird als boolescher Wert erfasst.
- **Internet über WLAN:** Ob die Internetverbindung über ein WLAN läuft und das Netz erreichbar ist, wird als boolescher Wert erfasst.
- **Bildschirmhelligkeit:** Der Wert der Bildschirmhelligkeit wird als Dezimalzahl in Prozent erfasst.
- **Lautstärke:** Der Wert der Lautstärke wird als Dezimalzahl in Prozent erfasst.

## 4.5. Erfassen der Hörgewohnheiten

Für die Auswertung der Hörgewohnheiten, ist es nötig, diese zu erfassen. Neben den Kontexten, in denen die Musik gehört wurde, ist dafür auch das Speichern von Informationen zu den einzelnen Liedern nötig. Das so entstehende Wiedergabeprotokoll setzt sich aus folgenden Bestandteilen zusammen:

- Informationen über die Wiedergabe: Hat der Benutzer ein Lied angehört, wird für dieses das **Startdatum** sowie das **Enddatum** der Wiedergabe und die **Dauer** des reinen Hörens (ohne Pausen) erfasst.

- Informationen über das Lied: Die wichtigste Information ist die **PersistentID**. Mit dieser einmaligen Kennung ist es möglich, einen Interpreten, ein Lied etc. innerhalb des Media Player Frameworks eindeutig zu identifizieren und damit das entsprechende *MPMediaItem* abzufragen. Damit ein Mindestmaß an Informationen über ein gehörtes Lied vorliegt – selbst dann, wenn dieses bereits von dem Gerät gelöscht wurde und eine Anfrage an das Media Player Framework somit kein Ergebnis liefern würde – werden zusätzlich folgende Metadaten festgehalten: der **Titel des Liedes**, der **Titel des Albums**, der **Name des Interpreten**, das **Genre** sowie die **Dauer des Liedes**.
- Verknüpfung zum vorliegenden Kontext: Zu jedem Wiedergabeeintrag eines Liedes wird der zum Zeitpunkt des Hörens vorliegende **Kontext** erfasst und dem Eintrag im Wiedergabeprotokoll hinzugefügt.

Generell werden Lieder erst ab einer reinen Hördauer von mindestens 15 Sekunden erfasst. Lieder, die während dieser Zeit beendet wurden (durch Springen zu einem anderen Lied oder das Beenden der Wiedergabe), werden ignoriert, da davon ausgegangen wird, dass das Lied nicht gemocht wurde. Durch den frühzeitigen Ausschluss dieser Lieder wird der Energie- und Netzwerksbedarf verbessert, da für diese Lieder kein Kontext erfasst werden muss.

Da bei der Auswertung der Hörgewohnheiten nicht einzelne Lieder betrachtet werden, sondern alle Lieder, die ab dem Start bis zum Beenden der Wiedergabe gehört wurden, werden zusätzlich Sitzungen angelegt. Die Wiedergabe wird dann als beendet betrachtet, wenn diese gestoppt – nicht pausiert – wurde. Dieser Fall tritt ein, wenn kein Lied in der Warteschlange folgt und das aktuelle Lied bis zum Ende abgespielt wurde oder wenn der Prototyp geöffnet bzw. verlassen wird während die Wiedergabe pausiert ist. Neben den Referenzen zu den oben beschriebenen Einträgen über die Wiedergabe von einzelnen Liedern, enthalten Sitzungen einen Kontext, der die Werte aller Kontexte der einzelnen Wiedergaben in einem zusammenfasst.

## 4.6. Berechnung der Ähnlichkeit von Kontexten

Bevor die kontextbasierten Empfehlungen aufgrund der Hörgewohnheiten getroffen werden können, muss die Ähnlichkeit des aktuell vorliegenden Kontextes zu denen der gespeicherten Sitzungen ermittelt werden. Die Berechnung der Ähnlichkeit basiert dabei

in weiten Teilen auf der Nächste-Nachbarn-Klassifikation. Im Gegensatz zu dieser, werden die Kontexte als Ergebnis jedoch nicht in Gruppen zusammengefasst sondern deren Ähnlichkeit aus dem resultierenden Abstand ermittelt. Dies begründet sich darin, dass keine feste Anzahl an Situationen festgelegt werden soll, die eine Gruppenmitgliedschaft erzwingen, sondern jede Kontextkombination für sich alleine ähnliche Situationen bilden können soll. Von der Verwendung eines reinen Entscheidungsbaumes wird abgesehen, da dessen Konstruktion für die Anzahl von 28 Merkmalen sehr aufwendig wäre und keine boolesche Entscheidung, sondern der genaue Grad der Ähnlichkeit erfasst werden soll.

Schematisch ergibt sich daraus folgender Ablauf, der im Anschluss ausführlich erläutert wird:

1. Zu Beginn werden die Merkmale der Kontexte aller gespeicherten Sitzungen in Merkmalsvektoren übertragen. Zusätzlich wird aus dem – zum Zeitpunkt der Nutzung – aktuell vorliegenden Kontext ebenso ein Merkmalsvektor gebildet. Zusammengenommen bilden diese den zu untersuchenden Merkmalsraum.
2. Mit dem aktuellen Kontext als Basisvektor werden alle anderen Merkmalsvektoren verglichen und deren einfachen Distanzen ermittelt.
3. Um eine ungewollte Gewichtung auszuschließen, werden die einfachen Distanzen durch eine Normalisierung auf ein einheitliches Intervall gebracht.
4. Im Zuge dieser werden die einfachen Distanzen außerdem auf pro Merkmal festgelegte Wertebereiche begrenzt. So kann der Charakter der Ähnlichkeit für jedes Merkmal individuell gestaltet werden.
5. Die normalisierten und begrenzten einfachen Distanzen werden mit Koeffizienten, welche die Gewichtung der Merkmale festlegen, multipliziert.
6. Abschließend wird aus der resultierenden Gesamtdistanz die Ähnlichkeit als Prozentszahl ermittelt.

#### 4.6.1. Abbildung im Merkmalsraum

Für die Abbildung eines Kontextes im Merkmalsraum wird ein Merkmalsvektor erstellt, welcher dessen Werte für alle Merkmale enthält. Dieser schränkt die Art der Datentypen nicht ein. Zeichenketten, die *CLLocation* und Dezimalzahlen werden ebenso wie Arrays

mit den jeweiligen Datentypen akzeptiert. Dies ist möglich, da die Abbildung als metrische Zahl erst bei dem Ausrechnen der Distanz erfolgt. Verschiedene Datentypen können so gesondert behandelt werden können.

#### 4.6.2. Berechnung der einfachen Distanz

Für die Berechnung der Ähnlichkeit wird zuerst die einfache Distanz in metrischer Darstellung berechnet. Je nach Datentyp sind hierfür unterschiedliche Vorgehensweisen nötig:

- Die Distanz  $d$  von **Dezimalzahlen**  $w$  kann ohne eine Anpassung direkt berechnet werden.

$$d = |w_A - w_B|$$

- **Boolesche Werte**  $b$  sind als 0 und 1 darstellbar, die Distanz kann somit ebenso direkt berechnet werden.

$$d = |b_A b_B| = |\{0, 1\} - \{0, 1\}|$$

- Die Merkmale, welche als **Zeichenketten**  $z$  vorliegen, haben untereinander keine Beziehung, die sich in Nähe ausdrücken lässt. So ist z.B. das *WLAN „eduroam“* dem *WLAN „Telekom \_ICE“* nicht näher als das *WLAN „Starbucks“*. Eine Abbildung in dem metrischen System ist somit nicht möglich. Wird dagegen für jeden möglichen Wert einer Zeichenkette – z.B. jeden im Wiedergabeprotokoll gespeicherten *WLAN* – ein Merkmal mit einem booleschen Wert angelegt, ist es möglich, diese abzubilden. Obwohl sich so die Anzahl der Merkmale erhöht, wird das Ergebnis der Distanzberechnung zweier Kontexte nicht verfälscht, da immer nur maximal eine Übereinstimmung möglich ist. Da eine Distanz von 0 nur vorliegt, wenn beide Zeichenketten identisch sind, lässt sich das Verfahren soweit vereinfachen, dass die beiden Zeichenketten auf die komplette Übereinstimmung hin verglichen werden und die Distanz dem booleschen Ergebnis entspricht.

$$d = |z_A z_B| = (z_A \neq z_B) = \{0, 1\}$$

- Die Distanz der **Aufenthaltsorte** wird durch das Ausrechnen der Luftlinie zwischen den beiden Geokoordinaten erstellt.

Für Werte, welche als Array vorliegen, wird die Distanz verwendet, welche am kürzesten ist. Dies begründet sich darin, dass die Abbildung des aktuell vorliegenden Kontextes vorerst nur einen einzelnen Zeitpunkt widerspiegelt. Da somit z.B. keine Fahrtwege erkannt werden, wird die Überschneidung von mindestens einem Aufenthaltsort als ähnlich betrachtet.

#### 4.6.3. Normalisierung der einfachen Distanzen

Bevor aus dem Distanz-Vektor, welcher aus dem Vergleich zweier Merkmalsvektoren entsteht, die Gesamtdistanz errechnet werden kann, müssen die Werte normalisiert werden. Dies ist nötig, da die Gewichtung von Merkmalen desto höher ist, je größer deren Wertebereich ist. Wird eine hohe Distanz mit einer niedrigen verglichen, hat das Vorhandensein bzw. Ausbleiben der niedrigen fast keine Auswirkung mehr [Abb. 11]. Werden alle Werte jedoch normalisiert, indem die minimalen und die maximalen Werte auf ein einheitliches Intervall übertragen werden, hat die maximale Distanz für alle Merkmale dieselbe Gewichtung [Abb. 12]. Eine unterschiedliche Gewichtung von verschiedenen Merkmalen

Distanz der Booleschen Werte	Distanz der Tageszeiten	Addierte Distanz
Min (0)	Min (0)	0
Max (1)	Max (12)	13
Min (0)	Max (12)	12
Max (1)	Min (0)	1

Abbildung 11: Darstellung der minimalen und maximalen Distanzen ohne eine Normalisierung.

Distanz der Booleschen Werte	Distanz der Tageszeiten	Addierte Distanz
Min (0)	Min (0)	0
Max (1)	Max (1)	2
Min (0)	Max (1)	1
Max (1)	Min (0)	1

Abbildung 12: Darstellung der minimalen und maximalen Distanzen mit einer Normalisierung auf das Intervall 0 bis 1.

ist zwar gewünscht, allerdings soll diese nicht auf den unterschiedlichen Wertebereichen, sondern auf speziell festgelegten Konstanten basieren. Bevor die Distanzen der Merkmale gewichtet werden, werden diese daher auf ein Intervall von 0 bis 1 übertragen [Abb. 13].

Mögliche Werte	Min				Max
<b>Ohne Normalisierung</b>	0	1	2	3	4
<b>Mit Normalisiert:</b>	0	0,25	0,5	0,75	1

Abbildung 13: Beispiel eines Wertebereichs ohne und mit einer Normalisierung auf die Intervalle 0 bis 1.

#### 4.6.4. Beschränkung auf Wertebereiche

Im Zuge der Normalisierung werden die Distanzen auf für jedes Merkmal individuelle Wertebereiche beschränkt. Diese erlauben es den Charakter der Ähnlichkeit unterschiedlich zu gestalten. So wird z.B. für den Aufenthaltsort ein Wertebereich von genau 200 Metern (plus die Distanz der Genauigkeit der Positionsdaten) festgelegt. Damit wird erreicht, dass die genaue Nähe zum aktuellen Aufenthaltsort keine Rolle spielt, sondern nur die Tatsache, ob es als derselbe Ort interpretiert wird oder nicht. Bei der Temperatur dagegen liegt der Wertebereich zwischen minimal 2°C und maximal 10°C Unterschied. So werden größere Temperaturunterschiede zwar als solche erkannt, eine exakte Übereinstimmung ist aber nicht nötig damit diese als ähnlich bewertet werden.



Abbildung 14: Normalisierung des Temperaturunterschiedes mit Beachtung des Wertebereichs von minimal 2°C bis maximal 10°C.

Ob die Temperatur *extrem* oder *sehr unterschiedlich* ist, spielt keine Rolle – in beiden Fällen ist sie unterschiedlich. Indem große Distanzen nicht näher bewertet werden, können Nahbereiche stärker differenziert werden, da sie nicht innerhalb eines hohen Wertebereichs an Gewichtung verlieren. Die Wertebereiche für alle weiteren Merkmale sind von ihrem Charakter her ähnlich gestaltet und können dem Anhang A entnommen werden.

#### 4.6.5. Gewichtung der Merkmale

Da nicht allen Merkmalen eines Kontextes dieselbe Bedeutung zugeschrieben wird, ist es nötig, diese zu gewichten. Ob der Benutzer z.B. die Musik über denselben Musikausgang gehört hat, wird im Rahmen dieser Arbeit als weniger wichtig angesehen als ein identischer Aufenthaltsort. Die Gewichte  $g$  sind so eingeteilt, dass die Summe  $G$  aller Gewichte 1 ergibt.

$$G = \sum_{m=1}^n g_m = 1$$

Ein Gewicht kann somit als Prozentzahl angesehen werden, welche den Anteil eines Merkmals am Gesamtergebnis festlegt. Die Merkmale können grob in zwei Gruppen unterteilt werden: primäre Merkmale sind in der Regel ausschlaggebend für die Ähnlichkeit eines Kontextes, wohingegen sekundäre Merkmale vorwiegend die Sortierung dieser beeinflussen. Zu den primären Merkmalen gehören die zeitlichen Merkmale (Tageszeit, Wochentag und Stunde), der Aufenthaltsort, die *WLAN SSID* sowie die aktuelle und die Tageshöchsttemperatur. Als weniger wichtig sind u.a. die Aktivitätsdaten, die Gerätedaten und die Genauigkeit des Aufenthaltsortes eingeordnet. Die Gewichtungen für alle Merkmale können dem Anhang A entnommen werden.

#### 4.6.6. Berechnung der Ähnlichkeit

Nachdem die einfachen Distanzen pro Merkmal berechnet, normalisiert und gewichtet wurden, kann die Gesamtdistanz  $D$  erzeugt werden. Diese wird mit Hilfe der „Cityblock-Metrik“ berechnet. Dabei werden die Distanzen der Werte  $w$  aller Merkmale  $m$  summiert.

$$D = \sum_{m=1}^n |w_{m_A} - w_{m_B}|$$

Da die normalisierten und gewichteten Distanzen  $N$  an diesem Punkt schon berechnet wurden, lässt sich die Formel wie folgt vereinfachen:

$$D = \sum_{m=1}^n N_m$$

Zur weiteren Verwendung in dem Prototypen wird die Gesamtdistanz als Ähnlichkeit in Prozent umgewandelt. Dabei wird die Relation aus der tatsächlichen Distanz und der maximal möglichen Distanz  $M$  ermittelt. Die maximale Distanz entspricht der Summe

aller mit ihrer Gewichtung multiplizierten maximal möglichen Werte. Da der maximal mögliche Wert 1 ist, reicht es, die Gewichte  $g$  aller Merkmale zu addieren. Mit der Voraussetzung, dass die Summe aller Gewichte ebenfalls 1 ist, ergibt sich so immer eine maximale Gesamtdistanz  $M$  von 1.

$$M = \sum_{m=1}^n 1 * g_m = 1$$

Da das Ergebnis einer Division durch 1 immer dem Dividenden entspricht, kann dieser Schritt ausgelassen werden. Die Ähnlichkeit  $\ddot{A}$  kann also durch das Abziehen der mit 100 multiplizierten Distanz ( $D * 100\%$ ) von 100% berechnet werden.

$$\ddot{A} = 100\% - (D * 100\%)$$

#### 4.6.7. Beispielhafte Berechnung

Beispielhaft folgt die Berechnung der Ähnlichkeit eines Kontextes mit zwei anderen Kontexten. Zum besseren Verständnis sind die Merkmale auf die Temperatur  $T$  mit einer Gewichtung von 0,3 und das WLAN  $W$  mit einer Gewichtung von 0,7 begrenzt.

Kontext  $A$ : {5°C, „eduroam“}

Kontext  $B$ : {30°C, „Telekom\\_ICE“}

Kontext  $C$ : {15°C, „eduroam“}

Berechnung der einfachen Distanz:

$$d_{T_{AB}} = |5 - 30| = 25$$

$$d_{W_{AB}} = (\text{„eduroam“} \neq \text{„Telekom\_ICE“}) = \text{Wahr} = 1$$

$$d_{T_{AC}} = |5 - 15| = 10$$

$$d_{W_{AC}} = (\text{„eduroam“} \neq \text{„eduroam“}) = \text{Falsch} = 0$$

Begrenzung der einfachen Distanzen auf die jeweiligen Wertebereiche:

$$d_{T_{AB}} = 0 \leq d_{T_{AB}} \leq 10 = 10$$

$$d_{W_{AB}} = 0 \leq d_{W_{AB}} \leq 1 = 1$$

$$d_{T_{AC}} = 0 \leq d_{T_{AC}} \leq 10 = 10$$

$$d_{W_{AC}} = 0 \leq d_{W_{AC}} \leq 1 = 0$$

Normalisierung und Gewichtung der Distanzen:

$$N_{T_{AB}} = \frac{15}{15} * 0,3 = 0,3$$

$$N_{W_{AB}} = \frac{1}{1} * 0,7 = 0,7$$

$$N_{T_{AC}} = \frac{15}{15} * 0,3 = 0,3$$

$$N_{W_{AC}} = \frac{0}{1} * 0,7 = 0$$

Berechnung der Gesamtdistanz:

$$D_{AB} = 0,3 + 0,7 = 1$$

$$D_{AC} = 0,3 + 0 = 0,3$$

Berechnung der Ähnlichkeit:

$$\ddot{A}_{AB} = 100\% - (1 * 100\%) = 100\% - 100\% = 0\%$$

$$\ddot{A}_{AC} = 100\% - (0,3 * 100\%) = 100\% - 30\% = 70\%$$

## 4.7. Generierung der Musikempfehlungen

Sind die Ähnlichkeiten des aktuellen Kontextes zu denen der gespeicherten Sitzungen ausgerechnet, werden diese für die Erstellung der Musikempfehlungen genutzt. In der ersten Version des Prototyps bestehen die Empfehlungen dabei aus den Liedern, die in den jeweiligen Sitzungen gehört wurden. Eine Empfehlung, die auf allen Merkmalen beruht, entspricht somit einer Sitzung, die in einem ähnlichen Kontext entstanden ist. Aus Benutzersicht bedeutet dies, dass eine Empfehlung nicht einem Lied sondern einer Gruppe von Liedern entspricht. Dabei werden Sitzungen, deren Kontext dem aktuell vorliegenden Kontext zu mindestens 70% ähnlich ist, als empfehlenswert betrachtet. Die Anzahl der angezeigten Empfehlungen wird auf die drei ähnlichsten begrenzt.

Da davon auszugehen ist, dass eine hohe Ähnlichkeit aller Merkmale selten vorkommt, Empfehlungen jedoch auch gewünscht sind, wenn nur einzelne Merkmale übereinstimmen, werden zusätzlich Empfehlungen generiert, die auf einzelnen Merkmalsgruppen beruhen. Die Merkmalsgruppen, welche betrachtet werden, sind der Ort, die Tageszeit, der Wochentag und das Wetter. In diesen Empfehlungen werden die ähnlichen Kontexte in jeweils einer Empfehlung pro Merkmalsgruppe gesammelt. Je nach Art werden dabei unterschiedliche Mindestähnlichkeiten benötigt, deren Festlegung durch die Größe der Merkmalsgruppe und Verteilung der Gewichte bestimmt wird. Maximal werden so sieben Kontexte bzw. Sammlungen von Kontexten empfohlen: bis zu drei Empfehlungen, die jeweils auf allen Merkmalen basieren und bis zu vier Empfehlungen, die jeweils auf einer Merkmalsgruppe basieren.

Damit dem Benutzer nicht permanent dieselben Lieder vorgeschlagen werden, ist mittelfristig geplant, die Empfehlungen so zu erweitern, dass neben Liedern, die tatsächlich in den entsprechenden Kontexten gehört wurden, auch ähnliche Lieder Teil der Empfehlung sind. Die Kenntnis über ähnliche Lieder könnte durch die Anbindung eines externen Empfehlungsdienstes erlangt werden, der für bestimmte Interpreten ähnliche Interpreten ausgibt. Die Schnittstelle würde in diesem Fall so oft angefragt werden, bis sie Interpreten zurückliefert, von denen der Benutzer Lieder auf dem Gerät hat. Aus diesen könnten dann Lieder zu der Empfehlung hinzugefügt werden. Da diese Funktion nicht im Fokus dieser Arbeit liegt und für den ersten Prototypen nicht elementar ist, wird sie vorerst nicht implementiert.

## 4.8. Benutzeroberfläche

Die Benutzeroberfläche soll die Funktionalität des Prototypen in einem modernen und ansprechenden Design bereitstellen. Analog zu der Einordnung im Kapitel 3.3, kann sie in die vier Bereiche Musikbibliothek, Musikempfehlung, Musikplayer und Debugbereich aufgeteilt werden. Ein ansprechendes Design wird insbesondere für die Bereiche als wichtig erachtet, die von normalen Benutzern verwendet werden. So sollen auch Testnutzer den Prototypen gerne benutzen und im besten Fall ausblenden, dass es sich um eine Studie handelt. Die Verwendung einer sehr schlichten oder technischen Benutzeroberfläche würde dies verhindern. Im Gegensatz dazu wird für den Debugbereich auf eine aufwendige Gestaltung verzichtet.

Der Entwurf und die Implementierung der Benutzeroberfläche werden unter Beachtung der Design-Richtlinien von Apple durchgeführt. Diese sind Teil der Dokumentation von iOS und geben vor, wie ein einheitliches Benutzererlebnis in Apps ermöglicht werden soll. Zu den grundlegenden Richtlinien gehören unter anderem folgende Vorgaben:

- Der Inhalt sollte der wichtigste Bestandteil der Benutzeroberfläche sein [App15d]. Dies könnte unter anderem durch die Verwendung von Tiefe, die Wahl der Schriftarten und -farben sowie das Weglassen von nicht-funktionalen Elementen (z.B. Logos) erreicht werden.
- Das Aussehen und die Funktion von Elementen der Benutzeroberfläche soll den Erwartungen der Benutzer entsprechen und innerhalb des Betriebssystems einheitlich sein [App15e]. Dies könnte unter anderem durch die Benutzung von bereits vorhandenen UI-Elementen des Betriebssystems für Standardaktionen erreicht werden. Dabei sei zu beachten, dass diese jeweils ihren Richtlinien folgen.

Die Einhaltung der Richtlinien ist zwar gewünscht, ein Zwang existiert aber nicht. Die Umsetzung einer komplett eigenen UI – wie z.B. in Spielen – ist somit trotzdem möglich. Gemäß dieser wird die Benutzeroberfläche mit Komponenten der iOS UI-Elemente<sup>1</sup> gebaut. Durch die gewünschte Nähe zum Betriebssystem und das Design der Apple Music App<sup>2</sup> orientiert sich das Design des Prototypen in weiten Teilen an diesem. Die Ressourcen, die für den Entwurf eines komplett eigenen Layouts eingespart werden, können so für die Implementierung des Prototypen selbst verwendet werden.

<sup>1</sup>Übersicht über die UI-Elemente: <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html>

<sup>2</sup>Beschreibung der Apple Music App: <https://support.apple.com/de-de/HT204951>

Alle verwendeten Screenshots sind dem entwickelten Prototypen entnommen und entsprechend dem implementierten und funktionierenden Stand des Prototypen. Neben der Veranschaulichung des Entwurfs der Benutzeroberfläche, ermöglichen diese damit einen Teileinblick in das Ergebnis der Implementierung.

#### 4.8.1. Grundstruktur

Als Navigationselement kommt eine *UITabBar* zum Einsatz. Die Grundstruktur des Prototypen besteht aus den Musikempfehlungen, der Musikbibliothek sowie – in der Version für interne Benutzer – dem Debugbereich. Jeder dieser Bereiche ist als Tab erreichbar. Wenn Musik abgespielt wird oder sich Inhalte in der Warteschlange befinden, wird über der *UITabBar* ein kleiner Miniplayer eingeblendet, welcher den Liedtitel, den Interpretennamen sowie den Albumtitel anzeigt und einen Wiedergabe- bzw. Pause-Button beinhaltet [Abb. 16]. Der eigentliche Musikplayer ist durch Tippen auf den Miniplayer oder durch Wischen nach oben erreichbar.

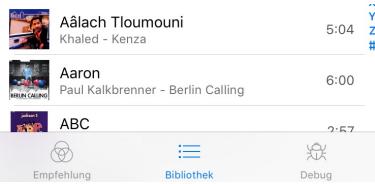


Abbildung 15: Die UITabBar mit ausgeblendetem Miniplayer.

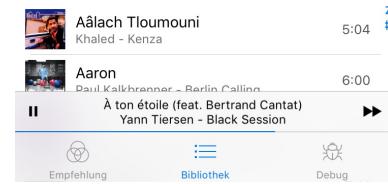


Abbildung 16: Die UITabBar mit eingeblendetem Miniplayer.

Durch diese Struktur ist ein schnelles Wechseln zwischen den verschiedenen Bereichen und ein eigener Navigations-Stack innerhalb der einzelnen Bereiche möglich.

### 4.8.2. Musikbibliothek

#### M1: Einsehen der Inhalte der Musikbibliothek

Die Musikbibliothek ist in die fünf Bereiche Interpreten, Alben, Lieder, Genres sowie Wiedergabelisten eingeteilt. Prinzipiell würde sich hierfür auch eine *UITabBar* anbieten; durch die bereits vorhandene *UITabBar* der Grundstruktur scheidet dies aber aus. Um dem Inhalt soviel Platz wie möglich zu geben und doppelte UI-Elemente zu verhindern, dient der Titel in der *UINavigationBar* als Button für ein Menü, über welches die Ansichten gewechselt werden können [Abb. 17]. Die Abgrenzung zwischen Menü und Liste wird durch den Hintergrund erreicht, welcher den dahinterliegenden Bereich dunkel weichzeichnet.

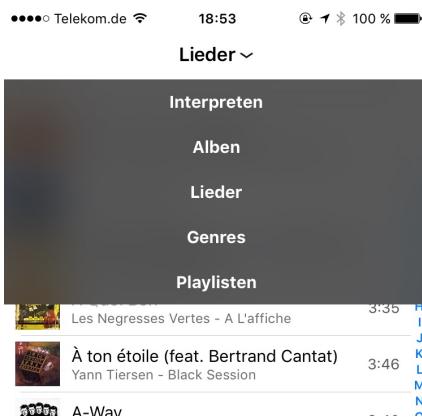


Abbildung 17: Das Menü der Musikbibliothek. Durch Antippen des Titels in der *UINavigationBar* öffnet bzw. schließt es sich.

Das schnelle Erreichen von allen Inhalten wird durch die Suchfunktion und alphabetische Gruppierung der Inhalte ermöglicht [Abb. 18]. Ist die Suche geöffnet, [Abb. 19] nimmt diese den gesamten Bildschirm ein. Zum besseren Verständnis der Suchtreffer wird der Teil des Textes, welcher mit dem Suchbegriff übereinstimmt, hervorgehoben.

Je nach Listenart zeigen die Einträge unterschiedliche Inhalte an. Neben dem Titel bzw. Namen des Eintrages können dies u.a. auch die Anzahl von darin enthaltenen Objekten sein [Abb. 20].

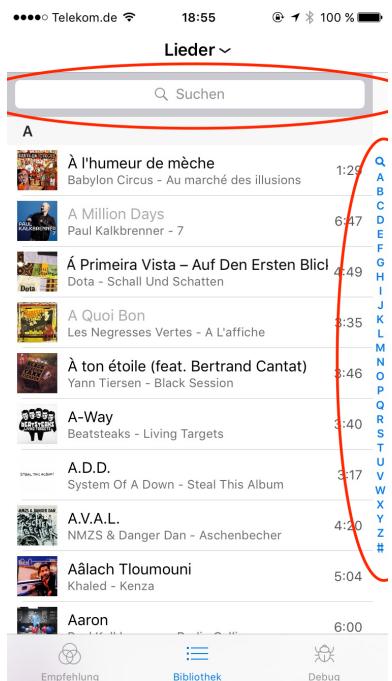


Abbildung 18: Oberhalb der Liste ist das Suchfeld zu sehen. An der rechten Seite befindet sich die Übersicht über alle Sektionen.

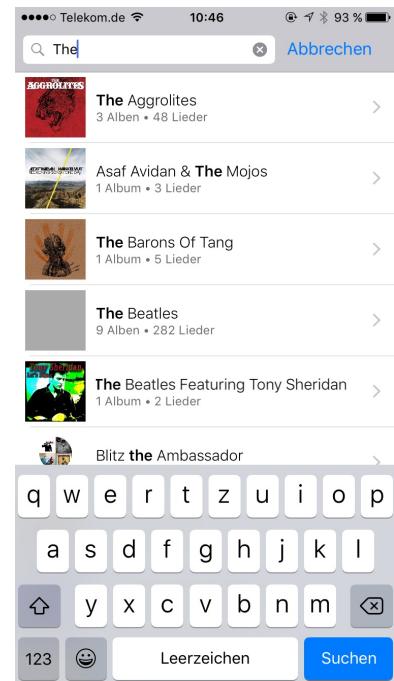


Abbildung 19: UI der Suche in der Musikbibliothek.

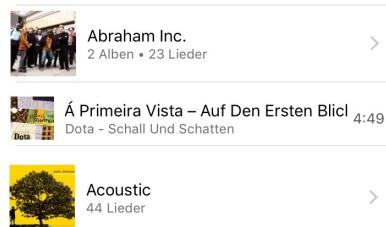


Abbildung 20: UI der Listeneinträge der Musikbibliothek.

Für die Seiten eines Interpreten und eines Albums werden zwei prägnante Farben aus dem Cover des Albums ausgelesen und als Farben für den Hintergrund und die sekundären Texte verwendet [Abb. 21].

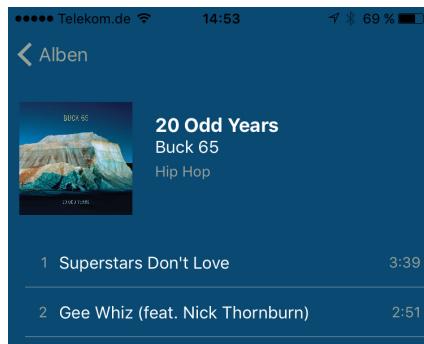


Abbildung 21: Ansicht eines Albums.

## M2: Abspielen von lokal verfügbaren Inhalten der Musikbibliothek

Das Abspielen eines Liedes kann durch Auswählen eines Songs gestartet werden. Dabei werden automatisch alle Lieder, die dem ausgewählten Lied folgen, der Warteschlange hinzugefügt.

### 4.8.3. Musikempfehlungen

Um die kontextbasierten Musikempfehlungen gegenüber den Einträgen der Musikbibliothek hervorzuheben, werden diese in einem eigenen Bereich umgesetzt. Jede empfohlene Sitzung bzw. Gruppe von Sitzungen wird dabei in einem Kartenlayout angezeigt [Abb. 22]. Die Symbole (oben links) geben die Merkmalsgruppen an, die dem aktuellen Kontext ähnlich sind. Werden mehrere Symbole angezeigt, bedeutet dies, dass für die Empfehlung alle Merkmale des Kontextes verglichen wurden. Wird nur ein Symbol angezeigt, handelt es sich um eine Empfehlung, die nur auf einer Merkmalsgruppe basiert. Zusätzlich wird die Ähnlichkeit in Prozent (oben rechts) angegeben. Zur schnellen Einsicht werden alle enthaltenen Lieder in einer horizontal scrollbaren *UICollectionView* dargestellt. Über den Wiedergabe-Button können diese direkt abgespielt werden.

Durch Berühren der Karte bzw. des „Details ansehen“-Buttons wird die Detailseite einer Empfehlung geöffnet. In dieser werden die Lieder, ebenso wie in der Musikbibliothek, in einer vertikalen *UITableView* angezeigt [Abb. 23]. Neben den Symbolen, die bereits auf der Übersichtseite verwendet wurden, werden die charakteristischen Werte der jeweiligen Merkmalsgruppen angezeigt. Im Falle eines ähnlichen Ortes werden die Aufenthaltsorte in einer Landkarte markiert.

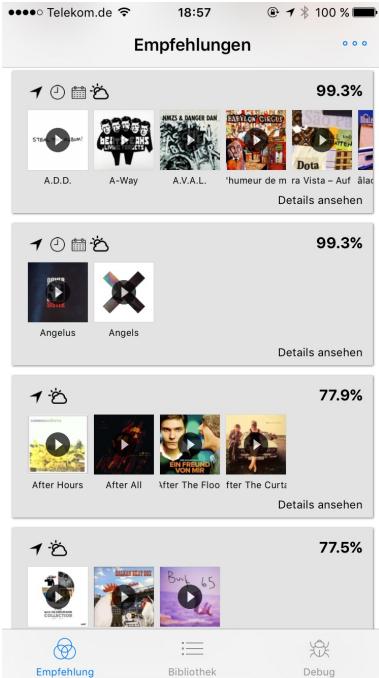


Abbildung 22: Übersicht über alle Empfehlungen.

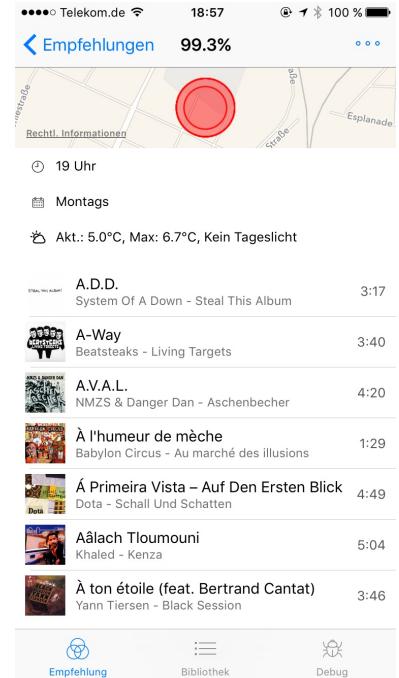


Abbildung 23: UI einer Empfehlung, die auf allen Merkmalsgruppen basiert.

#### 4.8.4. Musikplayer

##### P1: Aktuelle Musikwiedergabe steuern

Mit dem Miniplayer und dem ganzseitigen Player gibt es zwei Darstellungen des Musikplayers. Wie oben beschrieben wird der Miniplayer immer dann angezeigt, wenn Musik abgespielt wird oder sich noch welche in der Warteschlange befindet – z.B. wenn die Wiedergabe pausiert wird. Der ganzseitige Player ist nur über den Miniplayer erreichbar. Somit benötigen beide Player-Darstellungen keine spezielle Ansicht für den Fall, dass kein Lied anzuzeigen ist. Im Gegensatz zu der Musikbibliothek, bei der viele Inhalte dargestellt werden sollen, konzentriert sich die UI des Players immer auf das aktuell abzuspielende Lied. Von besonderer Bedeutung ist dabei das Cover des dazugehörigen Albums, welches circa die Hälfte des Bildschirms einnimmt [Abb. 24]. Als Hintergrund für den Kontrollbereich, der den restlichen Teil des Bildschirms einnimmt, wird das Cover unter einer weißen Überlagerung weichgezeichnet. Durch diese Gestaltung soll das schnelle Erkennen des aktuellen Albums und Interpreten erreicht werden.

Der Kontrollbereich beinhaltet alle Elemente die für die Kontrolle der aktuellen Wiedergabe nötig sind:

- einen Schieberegler, der die aktuelle Position innerhalb des Liedes anzeigt und mit dem es möglich ist, diese zu verschieben
- zwei Textfelder, welche die vergangene bzw. verbleibende Dauer des Liedes anzeigen
- den Titel des Liedes
- den Namen des Interpreten sowie des dazugehörigen Albums
- Buttons, um zum vorherigen und nächsten Lied zu springen sowie zum Pausieren bzw. Starten der Wiedergabe
- einen Schieberegler, über den die Lautstärke der Audioausgabe geregelt werden kann



Abbildung 24: Der ganzseitige Musikplayer.

Der ganzseitige Player kann sowohl über einen Button geschlossen werden, der sich oben links befindet und aus einem nach unten zeigenden Pfeil besteht, oder durch eine Wischgeste nach unten.

#### 4.8.5. Debugbereich

Aufgabe des Debugbereichs ist es, technische Informationen anzuzeigen und internen Benutzern die Möglichkeit zu geben, Aktionen auszuführen. Wie oben beschrieben, ist eine aufwendige Gestaltung für diese Nutzergruppe zweitrangig. Die optimale Lösung, die einfach sowie flexibel umzusetzen ist und den Anforderungen gerecht wird, ist die Verwendung von *UITableViews*. Für die Zellen wird dabei größtenteils der Standardstil des Betriebssystems genutzt [Abb. 25].



Abbildung 25: Beispiele für verwendete Zellen des Debugbereichs. Die oberen Zellen rufen einen nachfolgenden Bildschirm auf, die unteren zeigen Informationen an.

Sind Aktionen möglich, lassen sich diese direkt in der Zelle ausführen [Abb. 26].



Abbildung 26: Beispiele für verwendete Zellen, die eine Aktion ausführen.

Im Gegensatz zu den Übersichtslisten (Wiedergabeprotokoll, alle gespeicherten Kontexte, ähnliche Kontexte etc.), die ebenfalls aus Standardzellen bestehen, werden bei der Darstellung der Merkmale eines Kontextes sowie dem Vergleich zweier Kontexte spezielle Zellen verwendet.

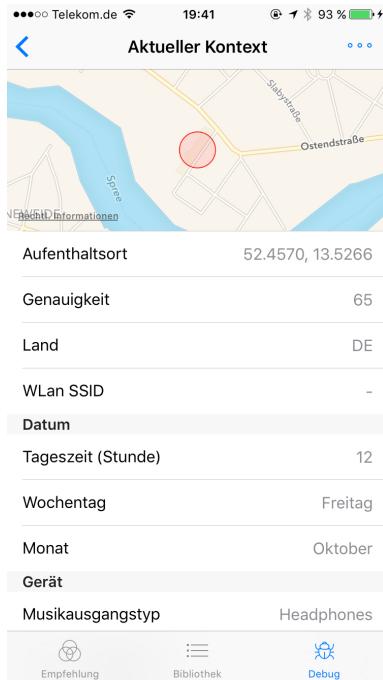


Abbildung 27: Debugansicht eines Kontextes. Werte, die als Array vorliegen, werden mehrzeilig angezeigt.

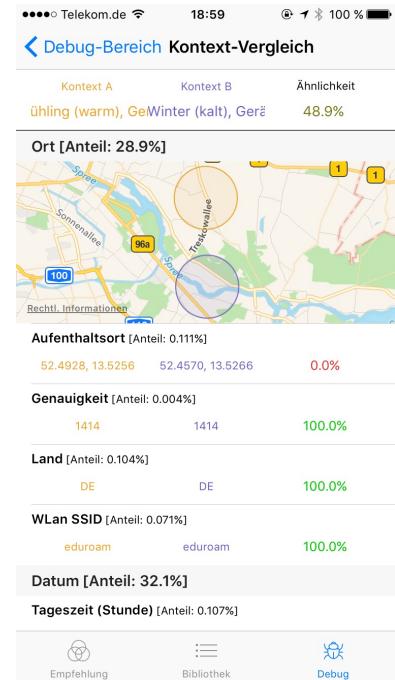


Abbildung 28: Debug-Ansicht des Vergleichs zweier Kontexte. Neben dem Kontext A (links) und dem Kontext B (mittig), wird die Ähnlichkeit (rechts) angezeigt.

Neben einer Landkarte können hier auch mehrere Werte pro Merkmal [Abb. 27] dargestellt werden. Im Falle des Vergleichs von Kontexten werden pro Zeile jeweils die zwei Werte eines Merkmals sowie deren Ähnlichkeit zueinander angezeigt [Abb. 28].

Die Übersicht über den Großteil der Screens des Prototypen kann dem Anhang B entnommen werden.

## 5. Implementierung

In diesem Kapitel werden Teile der Implementierung des Prototypen beschrieben. Neben dem Vorstellen der Projektgrundstruktur sowie der verwendeten Codebibliotheken wird ausführlich auf die Anbindung des Media Player Frameworks als Musikbibliothek sowie die Generierung der Musikempfehlungen eingegangen. Da diese nicht die gesamte Implementierung abdecken können, wird ein zusätzliches Einsehen des beigelegten Quellcodes empfohlen.

### 5.1. Projektgrundlagen

Der Prototyp ist als Standard *Xcode* Projekt angelegt. Neben der obligatorischen Projekt-Datei (.xcodeproj) wird auch eine Workspace-Datei (.xcworkspace) für die Anbindung von Codebibliotheken verwendet. Da der Prototyp keine angepasste UI für das iPad anbieten soll, ist unter Deployment Devices nur das **iPhone** festgelegt. Mit dieser Einstellung läuft die App auf einem iPad in dem skalierten Modus. Damit der Prototyp Musik abspielen und das Wiedergabeprotokoll befüllen kann, wenn sich diese im Hintergrund befindet, muss eine Hintergrundberechtigung angegeben werden. Dafür ist in den *Capabilities* unter **Background Mode** die Option **Audio, AirPlay and Picture in Picture** ausgewählt.

Vor der Veröffentlichung des fertigen Produkts ist es hilfreich u.a. eine Alpha App und eine Beta App zu erstellen. Durch diese Trennung ist es möglich, verschiedene Entwicklungsstände einer App anzubieten, welche gleichzeitig auf demselben Gerät installiert sein können. Die Beta App kann als stabile Umgebung genutzt werden, ohne dass während der Entwicklung Dinge kaputt gehen, z.B. durch Änderungen am Datenbankmodell. Mit der Verwendung von Targets ist dies ohne doppelten Code möglich. Jedes Target verfügt über eine eigene Konfigurationsdatei. So unterscheiden sich die Bundle Identifier und alle Konfigurationen, z.B. Icons, *Capabilities* und Provisioning Profiles, können getrennt voneinander bestimmt werden. Über Preprocessor Macros (Objective-C) bzw. Compiler Flags (Swift) können Flags gesetzt werden, die eine Unterscheidung der Targets im Code ermöglichen. Neben den Alpha und Beta Apps, welche der Version für interne Benutzer entsprechen, wird die Prototyp Version für normale Benutzer ebenso mit diesem Verfahren umgesetzt.

Da der Prototyp in Swift geschrieben ist und auch Codebibliotheken verwendet werden, welche in Objective-C geschrieben wurden, ist die Verwendung eines Bridging Headers

nötig [Quellcode 1]. Dabei handelt es sich um eine C Header-Datei, welche es erlaubt, Objective-C Klassen zu importieren. Wird diese in den Build Einstellungen des Projekts angegeben, ist die Verwendung der importierten Klassen in Swift möglich.

Quellcode 1: Der Inhalt der bachelorarbeit-prototyp-Bridging-Header.h Datei

```
1 //  
2 // Use this file to import your target's public headers that you  
3 // would like to expose to Swift.  
4 //  
5 #import <DKHelper/DKHelper.h>  
6 #import <MarqueeLabel/MarqueeLabel.h>  
7 #import "LEColorPicker.h"  
8 ...
```

Damit der Code möglichst modular gekapselt ist, wird Logik, die nicht unmittelbar mit der UI zusammenhängt, in separate Manager- oder Helperklassen ausgelagert. So sind z.B. das Abfragen des aktuellen Standortes, die Erstellung eines Kontextes und die Musikplayer-Funktionalität jeweils in einer eigenen Klasse implementiert. Die Benutzeroberfläche wurde, soweit möglich, mit dem Interface Builder von *Xcode* erstellt. *UIViewController* sind dabei in Storyboards und separate *UIViews* als *Xibs* abgebildet. Die Anpassung an unterschiedliche Displaygrößen wird mit der Verwendung von Auto-Layout sichergestellt.

Die Persistenz der anfallenden Daten wird durch die Verwendung von zwei Techniken umgesetzt: Einstellungen und Daten, die zusammenhangslos sind, werden in den *NSUserDefaults* vermerkt; die relationalen Daten des Wiedergabeprotokolls werden mit *Core Data* festgehalten. Ein komplettes Entity-Relationship-Modell des Wiedergabeprotokolls kann dem Anhang C entnommen werden.

## 5.2. Verwendete Codebibliotheken

Bei der Programmierung wird permanent auf existierenden Code zurückgegriffen. Neben eigenen Methoden und Klassen aus demselben Projekt, dem *SDK* des Betriebssystems und der Programmiersprache selbst, zählen dazu auch Bibliotheken, die von Dritten geschrieben wurden. Für die Entwicklung des Prototypen werden die folgend aufgezählten Codebibliotheken benutzt.

- DKHelper (2.1.1): Der DKHelper ist eine Sammlung von *Categories*, die häufig verwendete Codeabschnitte beinhalten. Dazu gehören zum Beispiel Methoden, die es erlauben, einen bestimmten Datentyp aus einem *NSDictionary* auszulesen und dabei zu prüfen, ob alle verwendeten Objekte existieren und der Datentyp dem gewünschten entspricht.

Link: <https://github.com/kevindelord/DKHelper>

- HockeySDK (3.8.5): HockeyApp ist eine Plattform, die Analysen, z.B. Abstürze und Installationszahlen, bereitstellt und die Distribution, z.B. von Alpha und Beta oder AdHoc-Apps, ermöglicht. HockeySDK ist das offizielle iOS *SDK* von HockeyApp.  
Link: <https://github.com/bitstadium/HockeySDK-iOS>

- MagicalRecord (2.3.0): MagicalRecord ist eine Bibliothek, die hauptsächlich aus *Categories* für Klassen aus *Core Data* besteht. Ziel ist es, möglichst viel Boilerplatecode, der bei der Benutzung von *Core Data* nötig ist, überflüssig zu machen. So werden unter anderem die Einrichtung der Datenbank und das Speichern sowie Auslesen von Datenbankeinträgen vereinfacht.

Link: <https://github.com/magicalpanda/MagicalRecord>

- NSManagedObject+ADVCopying: Hierbei handelt es sich um eine kleine *Category*, die es ermöglicht, eine tiefe Kopie eines *Core Data* Objektes zu erstellen. Die tiefe Kopie beinhaltet neben allen Attributen eines Objekts auch alle Objekte, zu denen eine Beziehung besteht.

Link: <https://gist.github.com/advantis/7642084>

- Reachability (3.2): Reachability ist eine Klasse, die es ermöglicht zu testen, ob eine Netzwerkverbindung existiert und um welche Art (Telefonnetz oder *WLAN*) es sich handelt.

Link: <https://github.com/tonymillion/Reachability>

- MarqueeLabel (2.5.0): Das MarqueeLabel ist eine Unterklasse, die das Standardelement für das Anzeigen eines Textes um einen Marquee-Effekt erweitert. Bei diesem wird ein Text, der zu lang für die Breite des Elementes ist, wie auf einem Laufband automatisch gescrollt.

Link: <https://github.com/cbpowell/MarqueeLabel>

- LECOLORPICKER (1.0.3): Der LECOLORPICKER ist ein Nachbau der Farbschema-Funktionalität von iTunes 11. Dabei werden aus einem Bild zwei prägnante Farben als Titel und Hintergrundfarbe sowie als sekundäre Farbe entweder schwarz oder weiß – je nach dem, ob die Hintergrundfarbe hell oder dunkel ist – gewählt.

Link: <https://github.com/luisespinoza/LEColorPicker>

- LNPOPUPCONTROLLER (1.2): Der LNPOPUPCONTROLLER ist ein Nachbau des Miniplayers der Apple Music App. Dieser stellt einen *UIView* für die Miniplayer-Leiste bereit sowie das animierte Öffnen des eigentlichen (Player-) *UIViewController*s. Im Gegensatz zu dem Miniplayer beinhaltet der LNPOPUPCONTROLLER keine Implementierung der Player UI.

Link: <https://github.com/LeoNatan/LNPopupController>

- YWEATHERAPI (1.0.6): Die YWEATHERAPI ist ein Wrapper für die REST API des Yahoo Wetterdienstes. Durch die Verwendung fallen eine Menge Boilerplatecode weg.

Link: <https://github.com/nishanths/YWeather\acrshort{api}>

- UIDevice-PASSCODESTATUS (0.0.2): Hierbei handelt es sich um eine kleine Category für die UIDevice Klasse. Durch den Zugriff auf den Schlüsselbund und den daraus resultierenden Erfolg oder der Fehlermeldung kann festgestellt werden, ob das Gerät momentan mit einem Passcode gesperrt ist oder nicht.

Link: <https://github.com/liamnichols/UIDevice-PasscodeStatus>

- MBPROGRESSHUD (0.9.2): Der MBPROGRESSHUD ist ein *UIView*, der verschiedene Varianten bereitstellt, mit denen ein Fortschritt (z.B. eines Downloads) in einem Overlay angezeigt werden kann.

Link: <https://github.com/jdg/MBProgressHUD>

- NSMANAGEDOBJECT+SERIALIZATION: Hierbei handelt es sich um eine kleine Category, die das Umwandeln eines Core Data Objektes in ein NSDictionary und umgekehrt ermöglicht.

Link: <https://gist.github.com/pkclsoft/4958148>

Der LNPOPUPCONTROLLER, die NSMANAGEDOBJECT+SERIALIZATION und das NSMANAGEDOBJECT+ADVCOPYING werden dem Projekt als selbständige Klassen in dem Ordner „External“ hinzugefügt. Alle anderen Codebibliotheken sind mit CocoaPods eingebunden.

Diese ist eine unabhängige Open Source Paketverwaltung für Swift und Objective-C [Coc]. Nachdem Code-Bibliotheken, welche CocoaPods unterstützen, in einem einfachen Textdokument, dem Podfile [Quellcode 2], festgelegt wurden, können diese durch einen Befehl auf der Konsole in das *Xcode*-Projekt integriert werden [Quellcode 3]. Cocoa Pods übernimmt dabei das Herunterladen der Code-Bibliothek sowie das Einbinden dieser in eine *Xcode* Workspace-Datei.

Quellcode 2: Inhalt der Podfile

```

1 source 'https://github.com/CocoaPods/Specs.git'
2
3 platform :ios, '9.0'
4
5 pod 'DKHelper', '~> 2.1.1'
6 pod 'MarqueeLabel', '~> 2.5.0'
7 ...

```

Das Herunterladen und Einbinden kann mit dem Befehl `install` aus der Konsole heraus gestartet werden:

Quellcode 3: Installieren von Pods

```
1 $ pod install
```

### 5.3. Anbindung der lokalen Musikbibliothek

Die Anbindung der lokalen Musikbibliothek erfolgt durch den `BSCMusicLibraryManager`. Dieser führt die einzelnen Anfragen an das Media Player Framework durch. Dafür werden grundsätzlich `MPMediaQueries` benutzt. Neben `MPMediaItems` sind `MPMediaItemCollections` die potenziellen Rückgabewerte. Eine `MPMediaItemCollection` enthält ein Array von `MPMediaItems`, die Kategorie der `MPMediaItems`, deren Anzahl sowie ein repräsentatives `MPMediaItem`, das z.B. dafür genutzt werden kann, den Titel eines Albums zu erfahren. Als Suchparameter stehen folgende Möglichkeiten zur Verfügung:

- mit dem `MPMediaPropertyPredicate` können Bedingungen für die Attribute der `MPMediaItems` festgelegt werden. So kann zum Beispiel nach Interpreten gesucht werden, deren Namen „The“ enthalten.
- mit dem `MPMediaGrouping` kann die Gruppierung der `MPMediaItemCollection` festgelegt werden. Dabei kann zwischen der Gruppierung nach dem Titel der Lieder,

der Alben, der Interpreten, der Alben-Interpreten, der Komponisten, der Genres, der Wiedergabelisten und der Podcasts gewählt werden.

Ist die Suche konfiguriert kann mit `items` auf das Array, welches alle passenden *MPMediaItems* enthält und mit `collections` auf das Array, welches alle passenden *MPMediaItemCollections* enthält, zugegriffen werden. Mit `itemSections` bzw. `collectionSections` gibt es zusätzlich die Möglichkeit ein Array an *MPMediaQuerySections* zu bekommen. Diese entsprechen jeweils einem Buchstaben des Alphabets. Neben den Buchstaben selbst, enthält die *MPMediaQuerySection* den Bereich der Indizes, die mit diesem Buchstaben beginnen. In Verbindung mit den eigentlichen Daten lässt sich so eine Gruppierung der Medien – z.B. in einer Liste – umsetzen.

### 5.3.1. Abfragen der ersten Ebene der Musikbibliothek

Für die erste Ebene der Musikbibliothek werden jeweils alle Inhalte der verschiedenen Kategorien (Interpreten, Alben etc.) benötigt. Um dies zu erreichen, wird zuerst eine Suchanfrage mit den jeweiligen Gruppierungen erstellt [Quellcode 4].

Quellcode 4: Suchanfrage an das Media Player Framework, gruppiert nach Interpreten.

```
1 MPMediaQuery.artistsQuery()
```

Da die Listen der ersten Ebene der Musikbibliothek eine alphabetische Gruppierung beinhalten, wird aus dem Ergebnis ein *Dictionary* erstellt, welches den Titel des Abschnitts und die dazugehörigen Medien enthält [Quellcode 5].

Quellcode 5: Gruppierung der *MPMediaItemCollections* nach dem Alphabet.

```
1 private class func sectionsFromCollections(query: MPMediaQuery)
2     -> [(title: String, sectionData: [MPMediaEntity])]?
3 {
4     // Create the dictionary which will be returned
5     var sectionedCollections =
6         [(title: String, sectionData: [MPMediaEntity])]()
7
8     if let
9         _collectionSections = query.collectionSections,
10        _collections = query.collections {
11         // Iterate through every section and add it to the dictionary
12         for sectionDescription in _collectionSections {
13             // Create the array which will contain all collections
14             var sectionContent = [MPMediaItemCollection]()
15
16             for collection in _collections {
17                 if collection.sections?.contains(sectionDescription) ?? false {
18                     sectionContent.append(collection)
19                 }
20             }
21
22             sectionedCollections.append((title: sectionDescription.rawValue, sectionData: sectionContent))
23         }
24     }
25
26     return sectionedCollections
27 }
```

```

11     for index in sectionDescription.range.location
12         .. < (sectionDescription.range.location
13             + sectionDescription.range.length) {
14             // Add all items from the sections range to the array
15             sectionContent.append(_collections[index])
16         }
17         // Set the title and the items array
18         sectionedCollections.append(
19             (sectionDescription.title, sectionContent))
20     }
21 }
```

Durch die Verwendung eines MPMediaQueries als Parameter kann die Methode für alle Kategorien genutzt werden. Pro Kategorie ist somit nur noch eine Methode nötig, die den MPMediaQuery erstellt und an die Methode `sectionsFromCollections(...)` weiterreicht [Quellcode 6].

Quellcode 6: Auslesen aller Alben - mit Gruppierung nach dem Alphabet

```

1 class func albumsSections() ->
2     [(title: String, sectionData: [MPMediaEntity])]? {
3     return self.sectionsFromCollections(MPMediaQuery.albumsQuery())
4 }
```

Für die Suche nach Titeln der Inhalte der verschiedenen Kategorien wird die `filter` Methode von Swift benutzt. Die Verwendung des MPMediaPropertyPredicate bietet sich nicht an, da die Übereinstimmung der Inhalte der allgemeinen MPMediaQueries mit anderen Inhalten nicht immer gewährleistet ist. Bei der Suche nach MPMediaItem-PropertyAlbumArtist würden z.B. auch Interpreten ausgegeben werden, für die keine Inhalte bereitstehen. Um die Ausgangsdaten kongruent zu halten, wird deswegen eine Filterung der Ergebnisse aus den bereits implementierten Abfragen umgesetzt [Quellcode 7]. Dabei wird das Array, welches die jeweiligen Inhalte fasst, mit `filter` gefiltert (Zeile 3). `filter` nimmt ein *Closure* als Parameter, welches für jedes einzelne Element des Arrays aufgerufen wird und je nach Rückgabe von `true` bzw. `false` über den Verbleib des Elements bestimmt.

Quellcode 7: Auslesen aller Alben, die den Suchbegriff im Titel beinhalten.

```

1  class func albums(searchTerm: String) -> [MPMediaItemCollection]? {
2    if let _collections = self.albums() {
3      return _collections.filter{
4        $0.representativeItem?.albumTitle?.lowercaseString
5          .containsString(searchTerm.lowercaseString) == true
6      }
7    }
8    return nil
}

```

### 5.3.2. Abfragen der tieferen Ebenen der Musikbibliothek

Wählt der Benutzer einen Inhalt aus der ersten Ebene der Musikbibliothek aus, werden die Daten, die für die nachfolgenden Bildschirme benötigt werden, dem Array der Daten der ersten Ebene entnommen. Wählt der Benutzer z.B. ein Genre aus, wird die *MPMediaItemCollection*, welche das Genre abbildet, an den nachfolgenden *UIViewController* übergeben. Die Anzeige eines Interpreten bildet hierbei eine Ausnahme. Damit auch Compilations angezeigt werden - also Alben, die Lieder eines Interpreten beinhalten, der nicht ihr Interpret ist - werden die Alben für einen Interpreten gesondert von dem Media Player Framework angefragt.

## 5.4. Berechnung der Ähnlichkeit von Kontexten

Nachdem die Logik der Berechnung der Ähnlichkeit von Kontexten bereits in dem Systementwurf (Kapitel 4.6) ausführlich beschrieben wurde, wird nun auf einzelne Aspekte der Implementierung eingegangen. Für den Vergleich von Kontexten wird das *Enum BSCContextFeature* als Datenmodell, welches jedes Merkmal abbilden kann, genutzt. Es enthält die Merkmale als Fälle (cases), die komplette Distanz und die Ähnlichkeit sowie Methoden, welche die Konstanten und Logiken für die einzelnen Merkmale umsetzen. Neben dem Kontext wird an manche Methoden auch das *Enum RecommendationType* übergeben. Dieser gibt die Art der Empfehlung an. Das Ausführen der einzelne Schritte wird in dem *BSCContextComparisonHelper* umgesetzt. Die einzelnen Methoden und deren Funktion wird im Folgenden vorgestellt. Dabei wird das Beispiel aus dem Systementwurf aufgegriffen, indem der Code, der den Vergleich der Merkmale Temperatur und *WLAN* durchführt, vorgestellt wird. Die Reihenfolge der einzelnen Schritte ist durch den Aufbau des Quellcodes bestimmt.

### 5.4.1. Festlegen der Gewichte und Wertebereiche

Der Wertebereich der einzelnen Merkmale wird in der Methode `range(...)` festgelegt [Quellcode 8]. Als Rückgabewert wird ein *Tuple* verwendet, welches den minimal sowie den maximal beachteten Wert und den maximalen zyklischen Wert enthält. Ist ein zyklischer Wert angegeben, bedeutet dies, dass der auf den angegebenen Wert folgende Wert wieder dem minimalen Wert entspricht. Zu den zyklischen Werten gehören die Tageszeit, die Woche und der Monat. So beträgt z.B. die einfache Distanz zwischen 23 Uhr und 3 Uhr 20 Stunden. Da die Uhrzeit zyklisch ist, beträgt die höchstmögliche Distanz aber 12 Stunden. Sobald diese Grenze überschritten ist, ist die Distanz zu der Uhrzeit vom nächsten oder vorherigen Tag näher. Somit ergibt sich zwischen 23 Uhr und 3 Uhr eine Distanz von 4 Stunden. Merkmale, die nicht zyklisch sind, haben keinen maximal zyklischen Wert. Im Falle der Temperatur und des *WLANs* entspricht dieser daher `nil`.

Quellcode 8: Festlegung der Wertebereiche der einzelnen Merkmale

```

1 func range(contextA: BSCContext?, contextB: BSCContext?) {
2     -> (min: Float, max: Float, cyclicMaxValue: Float?) {
3         switch self {
4             case WifiSSID:    return (min: 0, max: 1, cyclicMaxValue: nil)
5             case Temperature: return (min: 0, max: 15, cyclicMaxValue: nil)
6             ...
7         }
}

```

Die Gewichtung der einzelnen Merkmale wird in der Methode `weight(...)` festgelegt [Quellcode 9]. Dies erfolgt dabei mittels Gewichten, die zusammengenommen der Anzahl an Merkmalen entsprechen (Zeilen 13 und 15). Für den Vergleich aller Merkmale beträgt diese 28, bei dem Vergleich des Wetters wären es acht. Die finalen Gewichte, welche als Summe eins ergeben, werden durch die Division mit der Anzahl an Merkmalen berechnet (Zeile 19).

Quellcode 9: Festlegung der Gewichtung der einzelnen Merkmale

```

1 func weight(recommendationType: BSCRecommendation.RecommendationType) {
2     -> Float {
3         // Get the number of features
4         var number_of_features = 1
5         if (recommendationType == .Complete) {
6             number_of_features = 28
}

```

```

6     } else {
7         number_of_features
8             = self.dynamic_type.all_cases(recommendation_type).count
9     }
10    // Get the human readable weight
11    var weight = Float(0)
12    switch self {
13        case WifiSSID:
14            weight = (recommendation_type == .Complete ? 2 : 1.0)
15        case Temperature:
16            weight = (recommendation_type == .Complete ? 3 : 2.0)
17            ...
18    }
19    // Calculate the usable weight
20    return weight / Float(number_of_features)
}

```

#### 5.4.2. Abbildung im Merkmalsraum

Die Werte eines Merkmals lassen sich mit `value(...)` abfragen [Quellcode 10]. Wie im Systementwurf festgelegt, werden die jeweiligen Datentypen beibehalten. Da Werte auch fehlen können, ist der Rückgabewert ein Optional. Der Kontext, aus dem die Werte für die entsprechenden Merkmale ausgelesen werden sollen, wird der Methode als Parameter `context` übergeben.

Quellcode 10: Auslesen der Werte eines Merkmals für die Abbildung im Merkmalsraum

```

1 func value(context: BSCContext) -> AnyObject? {
2     switch self {
3         case Temperature:    return context.weather_context?.temperature
4         case WifiSSID:       return context.wifiSSIDs
5         ...
6     }
7 }

```

#### 5.4.3. Berechnung der einfachen Distanzen

Die unterschiedlichen Datentypen werden bei der Berechnung der einfachen Distanzen beachtet [Quellcode 11]. Die errechneten Distanzen sind dabei als Kommazahl festgelegt. So wird z.B. die Distanz der WLANs mit einem Vergleich der Zeichenketten und den Rück-

gabewerten 0 oder 1 bestimmt (Zeilen 4 bis 14 und 30). Die Distanz der Temperaturen, welche als Zahlen vorliegen, kann dagegen als Subtraktion der beiden Kommazahl-Werte berechnet werden (Zeilen 19 bis 24).

Quellcode 11: Berechnen der einfachen Distanzen

```

1 func distance(contextA: BSCContext, contextB: BSCContext) -> Float {
2     switch self {
3         case WifiSSID:
4             if let
5                 _wifiSSIDsA = self.value(contextA) as? [String],
6                 _wifiSSIDsB = self.value(contextB) as? [String] {
7                     for _wifiSSIDA in _wifiSSIDsA {
8                         for _wifiSSIDB in _wifiSSIDsB
9                             where _wifiSSIDB == _wifiSSIDA {
10                                if (_wifiSSIDA != "-") {
11                                    return Float(0)
12                                }
13                            }
14                        }
15                    ...
16                default:
17                    if let ... {
18                        ...
19                    } else if let
20                        _valueA = self.value(contextA) as? NSNumber,
21                        _valueB = self.value(contextB) as? NSNumber {
22                            // Use their float values if the values are NSNumbers
23                            return fabsf(_valueA.floatValue - _valueB.floatValue)
24                        }
25                    }
26                    if (self.value(contextA) == nil && self.value(contextB) == nil) {
27                        ...
28                    } else {
29                        // The values aren't the same or sth. went wrong: return 1 then
30                        return Float(1)
31                    }
32    }
}

```

#### 5.4.4. Normalisierung der einfachen Distanzen

Basierend auf den einfachen Distanzen werden die normalisierten Distanzen, unter Beachtung der Wertebereiche, mit der Methode `normalizedDistance(...)` gebildet [Quellcode 12]. Bei zyklischen Merkmalen wird hierbei auch deren zyklische Distanz berechnet (Zeilen 10 bis 18). Überschreitet die einfache Distanz die gerundete Hälfte `border` des maximalen zyklischen Wertes, wird diese von dem maximalen Wert subtrahiert und entspricht damit der kürzeren Distanz.

Quellcode 12: Erstellen der normalisierten Distanzen für zwei Kontexte

```

1 func normalizedDistance(contextA: BSCContext, contextB: BSCContext)
2     -> Float {
3     // Calculate the normal distance
4     var distance = self.distance(contextA, contextB)
5     if (distance == -1) {
6         return 0.5
7     } else {
8         // Get the range
9         let range = self.range(contextA, contextB)
10
11        if let _cyclic.MaxValue = range.cyclic.MaxValue {
12            // A cyclic max value is declared. This means that the value is
13            // looped (eg. the time -> 23, 24, 01, 02...).
14            // Example: 19:00 to 02:00 = 17h distance - 24 = 5h distance
15            let border = round(_cyclic.MaxValue / 2)
16            if (distance > border) {
17                // Shift if the distance is greater than the border (would be
18                // 4 if the shift is 7)
19                distance = _cyclic.MaxValue - distance
20            }
21        }
22        if (range.min == range.max) {
23            if (distance >= range.max) {
24                return 1
25            } else {
26                return 0
27            }
28        } else {
29            // Make sure the distance isn't smaller or greater than the
30            // defined min and max values
31        }
32    }
33}
```

```

27     if (distance < range.min) {
28         distance = range.min
29     } else
30     if (distance > range.max) {
31         distance = range.max
32     }
33     // Calculate the normalized distance
34     let rangeDistance = fabsf(range.max - range.min)
35     distance -= range.min
36     return distance / rangeDistance
37 }
38 }
39 }
```

#### 5.4.5. Berechnung der Ähnlichkeit

Nachdem die einzelnen Schritte mit dem BSCContextFeature umgesetzt wurden, werden diese von dem BSCContextComparisonHelper für die komplette Berechnung der Ähnlichkeit zwischen zwei Kontexten genutzt. Die Rückgabe der Ähnlichkeit erfolgt dabei als Teil des Distanz-Vektors. Bei diesem handelt es sich um einen *Typealias*, welcher einem Dictionary, das ein Merkmal als Schlüssel und eine Zahl als Wert nimmt, entspricht:

Quellcode 13: Datentyp des Distanz-Vektors

```
1 typealias DistanceVector = [BSCContextFeature: NSNumber]
```

Mit der Methode `distanceVector(...)` kann, basierend auf dem übergebenen `RecommendationType` und den beiden Kontexten, deren Distanz-Vektor abgefragt werden [Quellcode 14]. Dafür wird durch alle zu beachtenden Merkmale iteriert (Zeile 7 bis 16). Mittels `normalizedDistance(...)` und `weight(...)` wird die gewichtete und normalisierte Distanz erstellt (Zeile 11) und dem Distanz-Vektor hinzugefügt (Zeile 13). Die gesamte Ähnlichkeit wird nach der Iteration durch alle Merkmale anhand der absoluten Distanz berechnet (Zeile 22).

Quellcode 14: Erstellen des Distanz-Vektors für zwei Kontexte.

```

1 class func distanceVector(recommendationType: BSCRecommendation.
2                             RecommendationType, contextA: BSCContext, contextB: BSCContext)
3                             -> DistanceVector {
4                             // Variables to store the vector and the total distance
```

```

3   var distanceVector = DistanceVector()
4   var totalDistance = Float(0)
5
6   // Iterate through every feature and calculate the normalized
7   // distance
8   for feature in BSCContextFeature.allCases(recommendationType) {
9     // Calculate the normalized distance
10    let normalizedDistance
11      = feature.normalizedDistance(contextA, contextB: contextB)
12    // Get the weighted distance
13    let weightedDistance
14      = feature.weight(recommendationType) * normalizedDistance
15    // Set the distance for this feature
16    distanceVector[feature] = weightedDistance
17    // Add the distance to the totalDistance
18    totalDistance += weightedDistance
19  }
20
21  // Set the total distance
22  distanceVector[.TotalDistance] = totalDistance
23  // Get the total similarity (which is 1 - distance / max distance).
24  // As the maximal distance is 1, it's not calculated in this step
25  distanceVector[.Similarity] = 1 - totalDistance
26
27  return distanceVector
28}

```

## 5.5. Generierung der Musikempfehlungen

Anhand der Distanz-Vektoren für zwei Kontexte können nun Musikempfehlungen erstellt werden. Als Datenmodell dient dazu die `BSCRecommendation`. Dieses entspricht einer Empfehlung, die – je nach Art – entweder aus einer Sitzung oder einer Gruppe von maximal drei Sitzungen besteht und folgende Daten beinhaltet:

- die Distanz-Vektoren
- die Kontexte
- die in den Sitzungen gehörten Lieder (`BSCPlaylogSongs`)
- die dazugehörigen *MPMediaItems*
- die Art der Empfehlung (`RecommendationType`)

- den Vergleichskontext
- zusätzliche Helfermethoden

Die Empfehlungen für den aktuell vorliegenden Kontext, können mit der Methode `currentRecommendations(...)` des `BSCRecommendationHelpers` erzeugt werden [Quellcode 15]. Dieser fragt zuerst den aktuell vorliegenden Kontext ab und verwendet diesen dann als Ausgangsbasis für den Vergleich der Kontexte. Dazu wird der Kontext und der übergebene `RecommendationType` sowie das übergebene *Closure*, welches die Empfehlungen asynchron zurückgibt, an die Methode `recommendations` weitergereicht. In dieser wird durch alle gespeicherten Kontexte, die mit einer Sitzung verknüpft sind, iteriert und der jeweilige Distanz-Vektor abgefragt (Zeile 10). Wird die minimal nötige Ähnlichkeit erreicht, wird eine Empfehlung für den jeweiligen Kontext erzeugt (Zeile 16).

Quellcode 15: Sammeln aller ähnlichen Kontexte, die mit einer Sitzung verknüpft sind.

```

 1  class func recommendations(sourceContext: BSCContext,
 2     recommendationType: BSCRecommendation.RecommendationType,
 3     completion: (recommendations: [BSCRecommendation]) -> (Void)) {
 4
 5     if let allSessionContexts = BSCSessionContext.MR_findAllSortedBy(
 6         BSCContext.Key.StartDate, ascending: true) as? [BSCSessionContext]
 7     {
 8
 9         // Array which will contain all recommendations
10         var allRecommendations = [BSCRecommendation]()
11         var recommendation = BSCRecommendation()
12
13         ...
14
15         for sessionContext in allSessionContexts {
16             // Get the distance vector and store it in the sessionContext
17             let distanceVector = sourceContext.distanceVector(
18                 recommendationType, otherContext: sessionContext)
19
20             var minSimilatiry
21                 = BSCRecommendationManagerConstants
22                     .MinSimilarity(recommendationType)
23
24             if (recommendationType == .Complete) {
25                 minSimilatiry = BSCPreferences.minRecommendationsSimilarity
26             }
27
28             if ((distanceVector[.Similarity]?.floatValue ?? 0)
29                 >= minSimilatiry) {
30                 // Create the recommendation
31             }
32
33         }
34
35     }
36
37 }
```

```

18     }
19     // Sort the contexts after their Similarity and limit the number
20     // of recommendations
21     ...
22     completion(recommendations: allRecommendations)
23 } else {
24     completion(recommendations: [])
25 }
```

Je nachdem, ob die Empfehlung einer Sitzung oder einer Gruppe von Sitzungen entspricht, wird die bestehende `BSCRecommendation` erweitert oder eine neue erzeugt [Quellcode 16].

Quellcode 16: Erstellen einer Empfehlung.

```

1 // Create the recommendation
2 if (recommendationType.isCollection == false) {
3     // The recommendation isn't a collection, use new recommendation
4     then.
5     recommendation = BSCRecommendation()
6     recommendation.addSessionContext(sessionContext)
7     recommendation.distanceVectors.append(distanceVector)
8 } else {
9     subRecommendationContexts.append(
10         (context: sessionContext, distanceVector: distanceVector))
11 }
12 // Set the recommendation content
13 recommendation.sourceContext = sourceContext
14 recommendation.type = recommendationType
15 ...
```

Nachdem durch alle Sitzungen iteriert wurde, werden die Empfehlungen nach der größten Ähnlichkeit ihrer Kontexte sortiert, deren Anzahl beschränkt und das *Closure* `completion(...)` aufgerufen [Quellcode 15, Zeilen 21 und 23].

## 6. Qualität der Musikempfehlungen

Wie in Kapitel 3.4 erläutert, kann die Qualität der Musikempfehlungen durch die Qualität der Kontexterkennung repräsentiert werden. Eine Feldstudie, welche Feedback von mehreren Test- und Vergleichsgruppen sowie deren Interaktion mit den Musikempfehlungen sammelt, wäre zwar wünschenswert, ist aber aufgrund des beschränkten Rahmens dieser Arbeit keine Option. Um trotzdem einen ersten Eindruck der Qualität der Kontexterkennung zu erlangen, wird diese mit Hilfe eines Testdatensatzes – unter Laborbedingungen – ermittelt. Dieser wurde von dem Verfasser (A) sowie einer Testperson, die zwar mit dem Konzept der Arbeit vertraut ist, aber nicht die genaue Gewichtung und Wertebereiche der Merkmale kennt (B), analysiert. Für die Merkmalsgruppen Ort, Tageszeit, Wochentag und Wetter wurden die Testkontakte jeweils subjektiv nach ihrer Ähnlichkeit zueinander beurteilt. Empfehlungen, die auf allen Merkmalen basieren, wurden für diese Auswertung nicht beachtet. Pro Kontext-Kombination und Merkmalsgruppe wurde festgelegt, ob diese zueinander passen, d.h. ob eine Empfehlung oder keine Empfehlung ausgesprochen werden soll. Die gleichen Angaben wurden für den Prototypen ermittelt, indem die Empfehlungen für denselben Testdatensatz von dem Prototypen generiert wurden. Durch den Vergleich dieser mit den manuell erstellten Daten wird deren Übereinstimmung sowie die Anteile an zutreffenden, nicht gewünschten und fehlenden Empfehlungen als Qualitätsmerkmale ermittelt.

### 6.1. Testdatensatz

Der Testdatensatz wurde aus 24 Kontexten gebildet. Damit auch verschiedene Jahreszeiten, Tageszeiten etc. abgedeckt werden, sind diese gleichmäßig auf alle zwölf Monate des Jahres 2015 sowie die sieben möglichen Wochentage und auf acht feste Uhrzeiten aufgeteilt. Die Aufenthaltsorte beschränken sich auf die Standorte Wilhelminenhof sowie Treskowallee der HTW Berlin, den Arbeits- sowie Wohnort des Autors und zwei Ausreißer-Orte. Die Wetterdaten wurden für die jeweiligen Zeitpunkte – soweit verfügbar<sup>3</sup> – dem Archiv des Wetter-Informations-Dienstes Berlin<sup>4</sup> entnommen. Die Information über das Tageslicht wurde anhand der Uhrzeiten ermittelt. So wird eine breite Abdeckung, die genug Schnittmengen für ähnliche Kontexte liefert, erreicht. Eine detaillierte Übersicht

---

<sup>3</sup>Das Archiv des Wetter-Informations-Dienstes Berlin (WInD) beschränkt sich auf den Tagesverlauf der Temperatur sowie der Windgeschwindigkeit. Somit lassen sich die Merkmale aktuelle Temperatur, Tageshöchsttemperatur, Tagestiefsttemperatur und Windgeschwindigkeit ermitteln.

<sup>4</sup><http://wind.met.fu-berlin.de/wind/main.php>

über alle Werte des Testdatensatzes kann dem Anhang D, die getroffenen Empfehlungen der Analysen A und B den Anhängen E und F entnommen werden.

### **Unterschiede zwischen den Beurteilungen A und B**

Werden die Analysen A und B miteinander verglichen, ergibt sich eine Ähnlichkeit von 86,9%. Unterschiede ergeben sich dabei u.a. bei der Betrachtung der folgenden Merkmale:

- *WLAN*: Während derselbe Aufenthaltsort auch mit unterschiedlichen *WLANS* in A als ausreichend ähnlich gewertet wird, ist dies in B nicht der Fall.
- *Wochentag*: Montag bis Freitag sowie Samstag und Sonntag wurden in A jeweils als ähnlich betrachtet, in B wird Freitag dagegen als ähnlich zu Samstag und Sonntag gesehen.
- *Wetter*: In A wurde vorwiegend die Tageshöchsttemperatur für die Bestimmung der Ähnlichkeit betrachtet, in B wurde dagegen die aktuelle Temperatur priorisiert.

Die detaillierte Gegenüberstellung beider Analysen kann den Anhängen H und I entnommen werden.

## **6.2. Ergebnisse**

Während der Implementierung wurde die Auswertung in mehreren Iterationen mit angepassten Wertebereichen und Gewichtungen durchgeführt. Basierend auf den Ergebnissen wurden die Werte für die finale Version verwendet, die zu der höchsten Übereinstimmung mit den Analysen des Testdatensatzes führten. Eine detaillierte Übersicht über alle Ergebnisse kann den Anhängen H bis K entnommen werden. Wie der Vergleich der Analysen A und B untereinander, erreichen die Vergleiche mit dem Prototypen mit durchschnittlich 86,0% Übereinstimmung zu A und 89,3% zu B hohe Werte. Der Anteil an fehlenden Empfehlungen – d.h. Empfehlungen, die in der Analyse vorgesehen, von dem Prototypen jedoch nicht generiert wurden – beträgt zu A im Mittelwert 33,4% und zu B 24,4%. Der Anteil an falschen Empfehlungen – also von dem Prototypen generierte, in der Analyse jedoch nicht vorgesehene Empfehlungen – beträgt zu A im Mittelwert 18,4% und zu B 19,2%. Die Übereinstimmung der Empfehlungen, die auf dem Wochentag basieren weisen in den Analysen A mit 64,9% und B mit 74,6% die deutlich niedrigsten Werte auf.

### 6.3. Auswertung

Mit einer Ähnlichkeit der Vergleiche von über 85% wurden hohe Werte erreicht. Ebenso ist hervorzuheben, dass die Empfehlungen des Prototypen und die der Analyse B eine höhere Übereinstimmung aufweisen als die der Analysen A und B untereinander. So stellt der Prototyp zwar durch die Wertebereiche und Gewichte zwangsläufig subjektive Empfehlungen aus, diese sind den manuellen subjektiven Bewertungen – in diesem Test – aber ebenbürtig. Bei genauerer Betrachtung der Ergebnisse lassen sich Unterschiede zwischen der Herangehensweise der manuellen Analysen und dem Algorithmus des Prototypen ableiten. Interessant sind vor allem die folgenden zwei Unterschiede:

- Die relativ schlechten Werte der Empfehlungen zum Wochentag zeigen eine Schwäche der jetzigen Implementierung. Der Algorithmus ist bewusst so gestaltet, dass er keine Unterscheidung von Werktagen und Wochenende beinhaltet. Dies wurde vermieden, da diese Tage je nach Benutzer unterschiedlich ausfallen und so keine zuverlässigen Ergebnisse ermittelt werden können. Da die Wochentage in beiden Analysen jedoch in Werkstage und Wochenenden getrennt wurden – also eine Einteilung in feste Gruppen von Tagen –, der Prototyp dagegen nur einfache Abstände vergleicht, führt dies zu unpassenden Empfehlungen.
- Bei der Tageszeit zeigt sich außerdem, dass besonders in der Analyse A kein einheitlicher Wertebereich angewandt wurde. Während ein Unterschied von drei Stunden tagsüber dazu führt, dass sich die Tageszeiten nicht ähnlich sind, würde dieser Abstand nachts als ähnlich betrachtet werden. Dies kann ebenso durch die Anwendung von festen Gruppen statt einer Einordnung nach Abständen oder durch eine unterschiedliche Betrachtung von Extremwerten (Werte außerhalb des Hauptbereichs) erklärt werden.

## 7. Zusammenfassung

Mit dieser Arbeit wurde ein Ansatz für kontextbasierte Musikempfehlungen, die auf den Hörgewohnheiten der Benutzer aufbauen, entwickelt und mit einem Prototypen umgesetzt. Dieser bietet Zugriff auf alle lokal verfügbaren Lieder, welche mit dem iOS-Gerät synchronisiert wurden. Basierend auf dem vorliegenden Kontext und den Hörgewohnheiten, die innerhalb des Prototypen aufgezeichnet werden, werden dem Benutzer Lieder empfohlen, die er bereits zuvor in ähnlichen Kontexten gehört hat. Speziell für interne Benutzer bietet der Debugbereich Informationen, die das Nachvollziehen der Musikempfehlungen ermöglichen, indem u.a. der Inhalt des Wiedergabeprotokolls, der aktuelle Kontext und der Vergleich zweier Kontexte einsehbar ist. Mit dem Im- und Export des Wiedergabeprotokolls wird außerdem ermöglicht, den Prototypen mit Testdaten zu benutzen und die angefallenen Daten außerhalb des Prototypen auszuwerten.

In der ersten Auswertung des Prototypen konnte dessen Kontexterkennung ähnlich hohe Schnittmengen zu den Empfehlungen der manuellen Analysen aufweisen, wie diese untereinander. Da mit dieser ersten Auswertung zwar noch keine Empfehlungen untersucht werden konnten, die auf allen Merkmalsgruppen eines Kontextes gleichzeitig beruhen, und eine Evaluation der Interaktion mit den Empfehlungen durch Benutzer im Realbetrieb aussteht, kann die Auswertung nur einen ersten Eindruck geben. Dieser fällt aber für die betrachteten Aspekte positiv aus und kann als Basis für weitere Untersuchungen dienen.

### 7.1. Ausblick

Basierend auf der ersten Auswertung sind bereits einige Aspekte des Ansatzes sichtbar geworden, die erweitert werden könnten. Die Berechnung der Ähnlichkeit von Kontexten könnte so angepasst werden, dass ein Merkmal je nach Wert auf unterschiedliche Wertebereiche begrenzt wird bzw. diese je nach Merkmal durch Cluster ersetzt werden, die aus den bisher gespeicherten Werten gebildet werden. Des Weiteren könnte die Gewichtung und Mindestähnlichkeit, die für eine Empfehlung nötig ist, durch einen Entscheidungsbaum erweitert werden. Je nach Wert eines Merkmals könnten damit die Gewichtung und Mindestähnlichkeit eines anderen Merkmals geändert werden. Denkbar wäre z.B. die Gewichtung der Tageshöchsttemperatur nachts geringer ausfallen zu lassen als tagsüber oder die Gewichtung des Aufenthaltsortes, wenn sich das Land unterscheidet.

Bei der Generierung der Musikempfehlungen wäre eine Logik interessant, die nicht per se Sitzungen empfiehlt, deren Kontext ähnlich ist, sondern nur diejenigen Lieder der Sitzungen, welche mehrfach zusammenhängend in einem ähnlichen Kontext gehört wurden. So könnten Empfehlungen durch eine engere Musikauswahl verfeinert werden. Wie bereits in Kapitel 4.7 erwähnt, könnte die Auswahl der Lieder außerdem dahingehend verändert werden, dass neben den gehörten Liedern auch solche, die diesen ähnlich sind (und der lokalen Musikbibliothek entstammen), empfohlen werden.

## Literatur

- [App08] APPLE INC.: *Apple Announces iPhone 2.0 Software Beta.* <http://www.apple.com/pr/library/2008/03/06Apple-Announces-iPhone-2-0-Software-Beta.html>. Version: März 2008, Abruf: 04.01.2016
- [App13] APPLE INC.: *iOS Developer Library - Core Motion Framework Reference.* [https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion\\_Reference/](https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/). Version: September 2013, Abruf: 27.02.2016
- [App14] APPLE INC.: *iOS Developer Library - Media Player Framework Reference.* [https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MediaPlayer\\_Framework/index.html](https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MediaPlayer_Framework/index.html). Version: März 2014, Abruf: 05.01.2016
- [App15a] APPLE INC.: *App Store Review Guidelines.* <https://developer.apple.com/app-store/review/guidelines/>. Version: 2015, Abruf: 04.01.2016
- [App15b] APPLE INC.: *iOS Developer Library - Core Motion Framework Reference: CMMotionActivityManager Class Reference.* [https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CMMotionActivityManager\\_class/](https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CMMotionActivityManager_class/). Version: September 2015, Abruf: 27.02.2016
- [App15c] APPLE INC.: *iOS Developer Library - File System Programming Guide: About the iOS File System.* <https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html>. Version: März 2015, Abruf: 05.01.2016
- [App15d] APPLE INC.: *iOS Developer Library - iOS Human Interface Guidelines: UI Design Basics: Designing for iOS.* [https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html#/apple\\_ref/doc/uid/TP40006556-CH66-SW1](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html#/apple_ref/doc/uid/TP40006556-CH66-SW1). Version: Februar 2015, Abruf: 17.01.2016

- [App15e] APPLE INC.: *iOS Developer Library - iOS Human Interface Guidelines: UI Design Basics: Integrating with iOS.* <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Integration.html>. Version: November 2015, Abruf: 17.01.2016
- [App15f] APPLE INC.: *iOS Developer Library - Location and Maps Programming Guide: About Location Services and Maps.* [https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/Introduction/Introduction.html#/apple\\_ref/doc/uid/TP40009497-CH1-SW1](https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/Introduction/Introduction.html#/apple_ref/doc/uid/TP40009497-CH1-SW1). Version: November 2015, Abruf: 27.02.2016
- [App15g] APPLE INC.: *iOS: Understanding data protection.* <https://support.apple.com/en-us/HT202064>. Version: Februar 2015, Abruf: 04.01.2016
- [App16] APPLE INC.: *Support: App Store.* <https://developer.apple.com/support/app-store/>. Version: Januar 2016, Abruf: 16.01.2016
- [BKR13] BRAUNHOFER, Matthias ; KAMINSKAS, Marius ; RICCI, Francesco: Location-aware music recommendation. In: *International Journal of Multimedia Information Retrieval* 2 (2013), Januar, Nr. 1, S. 31–44
- [Coc] COCOAPODS: *About CocoaPods.* <https://cocoapods.org/about>, Abruf: 28.02.2016
- [LC14] LEE, Mirim ; CHO, Jun-Dong: Logmusic : Context-Based Social Music Recommendation Service on Mobile Device . In: *the 2014 ACM International Joint Conference*. New York, New York, USA : ACM Press, 2014, S. 95–98
- [RRS10] RICCI, Francesco ; ROKACH, Lior ; SHAPIRA, Bracha: Introduction to Recommender Systems Handbook. In: *Recommender Systems Handbook*. Boston, MA : Springer US, Oktober 2010, S. 1–35
- [SBI14] SCHEDL, Markus ; BREITSCHOPF, Georg ; IONESCU, Bogdan: Mobile Music Genius: Reggae at the Beach, Metal on a Friday Night? In: *ICMR* (2014), S. 507–510
- [Spi15] Play Musik: Google will Songs passend zur Stimmung abspielen - SPIEGEL ONLINE. (2015), August. <http://www.spiegel.de/netzwelt/gadgets/>

[google-play-music-fuehrt-kuratierte-playlists-ein-a-1049790.html](http://www.google.com/search?hl=de&q=cache%3Ahttp%3A%2F%2Fwww.pc-magazin.de%2Fartikel%2F1049790%2Fgoogle-play-music-fuehrt-kuratierte-playlists-ein-a-1049790.html),  
Abruf: 25.02.2016

[Wik15] WIKIPEDIA: *Wikipedia: Apple motion coprocessors.* [https://en.wikipedia.org/wiki/Apple\\_motion\\_coprocessors](https://en.wikipedia.org/wiki/Apple_motion_coprocessors). Version: November 2015, Abruf: 28.02.2016

[Wik16] WIKIPEDIA: *List of iOS devices.* [https://en.wikipedia.org/wiki/List\\_of\\_iOS\\_devices](https://en.wikipedia.org/wiki/List_of_iOS_devices). Version: Januar 2016, Abruf: 16.01.2016

[Yah] YAHOO: *Yahoo Weather Developer Documentation: Yahoo Weather RSS Feed.* <https://developer.yahoo.com/weather/documentation.html>, Abruf: 22.01.2016

## **Abkürzungsverzeichnis**

**API** Application Programming Interface.

**GPS** Global Positioning System.

**POI** Point of Interest.

**REST** Representational State Transfer.

**SDK** Software Development Kit.

**WLAN** Wireless Local Area Network.

## Glossar

**Capabilities** Teil der Xcode-Projektdatei, in der die Verwendung von bestimmten Funktionen und Berechtigungen angegeben wird.

**Category** Funktion von Objective-C, die es ermöglicht, Funktionen zu bestehenden Klassen hinzuzufügen.

**CLLocation** Klasse des Core Location Frameworks, welche einen Ort darstellt.

**Closure** Funktion in Swift, welche wie ein Datentyp behandelt werden kann und Zugriff auf die Inhalte ihres Erstellungskontextes bietet.

**Core Data** Framework, welches relationale Daten in einem Objektgraphen persistiert.

**Dictionary** Datentyp in Swift, welcher Werte in einem Key-Value-Store hält.

**Enum** Aufzählungsdatentyp in Swift, der das Anlegen einer begrenzten Menge von Werten ermöglicht.

**Maschinelles Lernen** Gebiet der Informatik, welches Algorithmen entwickelt, die bekannte Datensätze auswerten und ihre Funktion anhand der Ergebnisse verbessern.

**MPMediaItem** Klasse des Media Player Frameworks, welches einem Lied entspricht und neben dem Link zu der Audiodatei alle Metadaten hält.

**MPMediaItemCollection** Klasse des Media Player Frameworks, welches eine Ansammlung von MPMediaItems enthält.

**NSDictionary** Klasse, welche Daten in einem Key-Value-Store hält.

**NFetchedResultsController** Cora Data Klasse, welche das effiziente Abfragen von Daten aus Core Data ermöglicht.

**NSUserDefaults** Klasse von iOS, welche das langfristige Persistieren von Werten in einem Key-Value-Store innerhalb der App ermöglicht.

**SSID** Bezeichnung eines WLANs, welche nicht einmalig ist und an verschiedenen Orten identisch sein kann.

**Tuple** Funktion in Swift, welche es erlaubt, mehrere Variablen zu einem Objekt – ohne zusätzliche Deklaration – zu verbinden.

**Typealias** Funktion von Swift, die es ermöglicht, bestehende Datentypen durch einen eigenen Typ zu repräsentieren.

**UICollectionView** UI-Element, welches eine dynamische Anzahl an Inhalten als horizontale und vertikale Liste bzw. als Galerie anzeigt.

**UINavigationBar** UI-Element, mit dem sich die Navigation der App steuern lässt und welches meist den Titel des aktuellen UIViewController beinhaltet.

**UITabBar** UI-Element, welches am unteren Bildschirmende positioniert ist und mehrere Tabs enthält, mit denen sich UIViewController ansteuern lassen.

**UITableView** UI-Element, welches eine dynamische Anzahl an Inhalten als horizontale Liste darstellt.

**UIView** Basisklasse für Elemente, die auf dem Screen angezeigt werden.

**UIViewController** Basisklasse, die unter iOS einen Screen darstellt.

**Xcode** Offizielle IDE von Apple, die für die Entwicklung von iOS und OS X Apps genutzt werden kann.

**XIB** Dateiformat des Xcode Interface Builders.

## A. Übersicht über alle Merkmale

Übersicht über alle Merkmale						
	Bezeichnung	Datentyp	Wertebereich	Max. zyklischer Wert	Gewichtung (Komplett)	Gewichtung als Merkmalsgruppe
<b>Ort</b>	Aufenthaltsort	CLLoaction	< 200m + Genauigkeit	-	0,11071	0,52500
	Genauigkeit	Float	50 - 500m	-	0,00357	0,02500
	Land	String	-	-	0,10357	0,20000
	WLAN-Name	String	-	-	0,07143	0,25000
<b>Tageszeit</b>	Tageszeit	Int	1 - 3	24	0,10714	1,00000
<b>Wochentag</b>	Wochentag	Int	0 - 2	7	0,10714	1,00000
<b>Monat</b>	Monat	Int	1 - 3	12	0,10714	-
<b>Wetter</b>	Aktuelle Temperatur	Float	2 - 5°C	-	0,10714	0,25000
	Tageshöchsttemperatur	Float	2 - 10°C	-	0,10714	0,25000
	Tagestiefsttemperatur	Float	2 - 10°C	-	0,00357	0,05000
	Sichtweite	Float	1 - 3km	-	0,00357	0,05000
	Tageslicht	Bool	0 / 1	-	0,10714	0,25000
	Luftfeuchtigkeit	Float	10 - 30%	-	0,00357	0,05000
	Windgeschwindigkeit	Float	5 - 10km/h	-	0,00357	0,05000
	Wetterlage	String	-	-	0,00357	0,05000
<b>Bewegungsdaten</b>	Auto	Float	0 - 1	-	0,00357	-
	Fahrrad	Float	0 - 1	-	0,00357	-
	Rennen	Float	0 - 1	-	0,00357	-
	Stationär	Float	0 - 1	-	0,00357	-
	Unbekannt	Float	0 - 1	-	0,00357	-
	Gehen	Float	0 - 1	-	0,00357	-
<b>Gerätedaten</b>	Internet über Telefonnetz	Bool	0 / 1	-	0,00357	-
	Internet über WLAN	Bool	0 / 1	-	0,00357	-
	Musikausgangstyp	String	-	-	0,00357	-
	App im Vordergrund	Bool	0 / 1	-	0,00357	-
	Gerät gesperrt	Bool	0 / 1	-	0,00357	-
	Lautstärke	Float	0 - 1	-	0,00714	-
	Bildschirmhelligkeit	Float	0 - 1	-	0,00357	-
	Minimale Ähnlichkeit, die für eine Empfehlung nötig ist				1,00000	
<b>Komplett</b>	70 %					
<b>Ort</b>	25 %					
<b>Tageszeit</b>	1 %					
<b>Wochentag</b>	1 %					
<b>Wetter</b>	80 %					

## B. Übersicht über die UI des Prototypen

**Player**

Album: Birdman - 100%  
Song: 100%  
Artist: Night Cheetah  
Genre: Latin American  
Length: 1:52  
Lyrics: No lyrics available

**Empfehlung Details**

Rating: 98.3%  
Song: Living Targets  
Artist: A.Way  
Genre: Latin American  
Length: 1:40  
Lyrics: No lyrics available

**Empfehlungen**

Rating: 98.3%  
Song: Living Targets  
Artist: A.Way  
Genre: Latin American  
Length: 1:40  
Lyrics: No lyrics available

**Musikbibliothek: Lieder**

Rating: 100%  
Song: Living Targets  
Artist: A.Way  
Genre: Latin American  
Length: 1:40  
Lyrics: No lyrics available

**Musikbibliothek: Interpreten**

Rating: 100%  
Song: Living Targets  
Artist: A.Way  
Genre: Latin American  
Length: 1:40  
Lyrics: No lyrics available

**Interpreten Details**

Rating: 100%  
Song: Living Targets  
Artist: A.Way  
Genre: Latin American  
Length: 1:40  
Lyrics: No lyrics available

**Album**

Rating: 100%  
Song: Living Targets  
Artist: Dimanche à Bamako  
Genre: Latin American  
Length: 1:40  
Lyrics: No lyrics available

## B ÜBERSICHT ÜBER DIE UI DES PROTOTYPEN

**Debugbereich**

- Relaisstatus: 16:55    & + 100% ■
- Debug-Bereich
- Wiederholungsprotokoll
- Wiedergabeortfolie
- Alle gespeicherten Kontexte
- Alle gespeicherten Kontexte (von Sitzungen)
- AKTUELLER KONTEXT
- Aktueller Kontext
- KONTEXTVERGLEICH
- Vergleich von Kontexten
- Momentan ähnliche Kontexte
- Momentan ähnliche Kontexte (von Sitzung...)
- EINSTELLUNGEN
- Minimale Ähnlichkeit für Empfehlungen: 72%
- Erweiterung
- Erweiterung
- Erweiterung
- Erweiterung
- Erweiterung

**Aktueller Kontext Detail**

Naturschule ▶ 19:21    & + 100% ■

Debug-Bereich Aktueller Kontext

Wiederholungsprotokoll

Wiedergabeortfolie

Alle gespeicherten Kontexte

Alle gespeicherten Kontexte (von Sitzungen)

AKTUELLER KONTEXT

Aktueller Kontext

KONTEXTVERGLEICH

Vergleich von Kontexten

Momentan ähnliche Kontexte

Momentan ähnliche Kontexte (von Sitzung...)

EINSTELLUNGEN

Minimale Ähnlichkeit für Empfehlungen: 72%

Erweiterung

Erweiterung

Erweiterung

Erweiterung

Erweiterung

Erweiterung

**Gespeicherte Kontexte von Sitzungen**

Zeitraum	Anzahl	Prozent
12.03.16, Sa, 05:41 - 05:41	1	92,1%
12.03.16, Sa, 05:38 - 05:38	1	9,3%
07.03.16, Mo, 19:02 - 19:04	1	0,7%
02.03.16, Mo, 19:00 - 19:00	1	0,7%
07.03.16, Mo, 18:48 - 18:48	1	0,7%
07.03.16, Mo, 18:32 - 18:36	1	0,7%
2010 DE	1	0,7%
Wlan SSID	1	0,7%
Datum	1	0,7%
Tagszeit (Stunde)	1	0,7%
Wochentag	1	0,7%
Monat	1	0,7%
Gerät	1	0,7%
Markusungstyp	1	0,7%
Erweiterung	1	0,7%

**Ähnliche Kontexte**

Relaisstatus: 19:29    & + 100% ■

Debug-Bereich Ähnliche Kontexte

Relaisstatus: 19:29    & + 100% ■

Debug-Bereich Ähnliche Kontexte

Ort (Anzahl: 28,9%)

Autobahnen (Anzahl: 0,1%)

Gemeindegebiet (Anzahl: 0,0%)

Landkreis (Anzahl: 0,0%)

Wlan SSID (Anzahl: 0,0%)

Autobahn (Anzahl: 0,0%)

Autobahnen (Anzahl: 0,0%)

Gemeindegebiet (Anzahl: 0,0%)

Landkreis (Anzahl: 0,0%)

DE (Anzahl: 100%)

Wlan SSID (Anzahl: 0,0%)

Autobahn (Anzahl: 0,0%)

Datum (Anzahl: 32,1%)

Tagszeit (Blended) (Anzahl: 0,0%)

Erweiterung

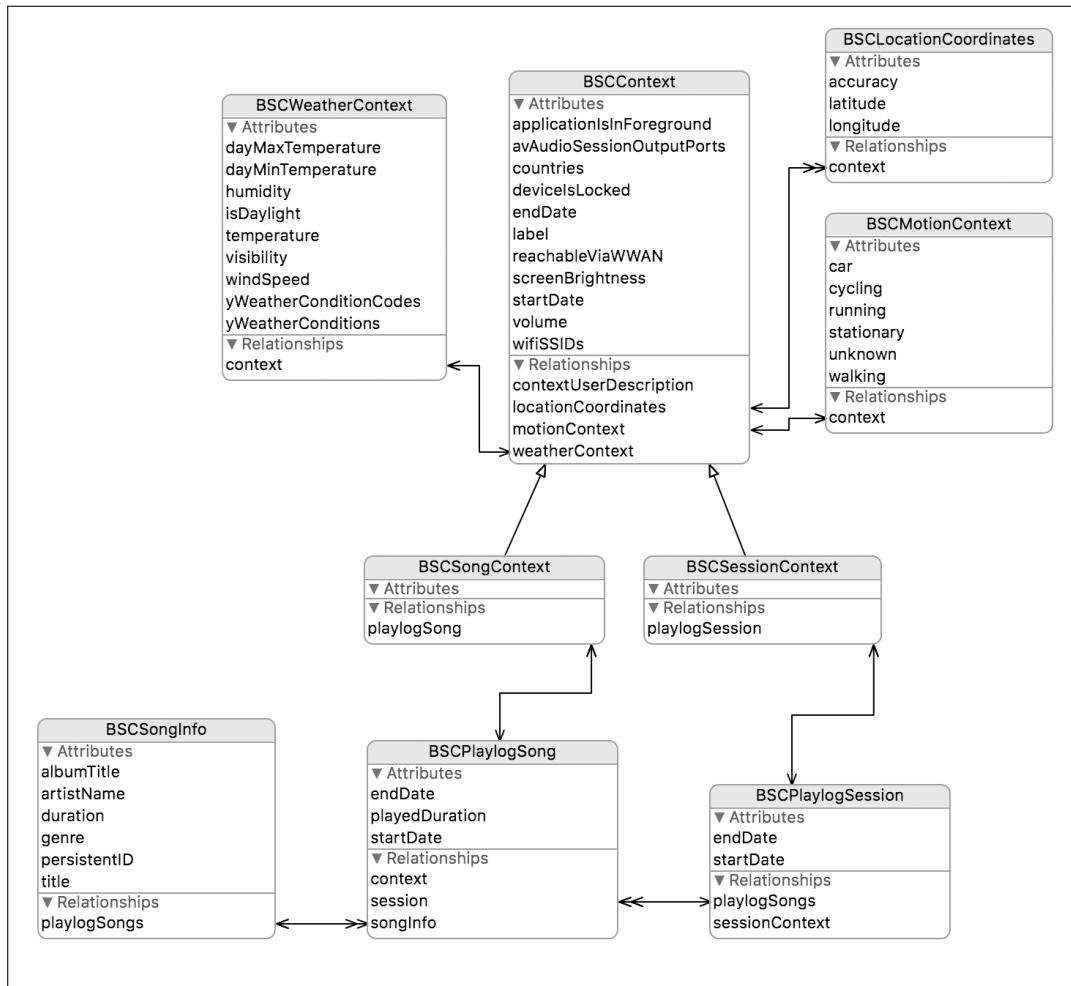
Erweiterung

Erweiterung

Erweiterung

Erweiterung

## C. Entity-Relationship-Modell



## D. Werte des Testdatensatzes

	Ort	Latitude	Longitude	Genaugkeit	Land	WLan SSID	Datum
1	Heim	52.558322	13.391544	65	DE	Kabellos	Mo., 5. Jan. 2015 07:00
2	Arbeit	52.514710	13.390465	65	DE	SMF	Di., 20. Jan. 2015 09:00
3	HTW W.	52.456972	13.526578	1414	DE	eduroam	Mi., 4. Feb. 2015 12:00
4	HTW T.	52.492805	13.525629	65	DE	eduroam	Do., 19. Feb. 2015 15:00
5	Arbeit	52.514710	13.390465	65	DE	SMF	Fr., 6. März 2015 18:00
6	Heim	52.558322	13.391544	65	DE	Kabellos	Sa., 14. März 2015 20:00
7	Heim	52.558322	13.391544	65	DE	Kabellos	So., 5. Apr. 2015 23:00
8	Heim	52.558322	13.391544	65	DE	-	Mo., 20. Apr. 2015 02:00
9	HTW W.	52.456972	13.526578	65	DE	eduroam	Di., 5. Mai 2015 07:00
10	Arbeit	52.514710	13.390465	1414	DE	-	Mi., 20. Mai 2015 09:00
11	HTW T.	52.492805	13.525629	1414	DE	eduroam	Do., 4. Juni 2015 12:00
12	Arbeit	52.514710	13.390465	1414	DE	SMF	Fr., 19. Juni 2015 15:00
13	Heim	52.558322	13.391544	65	DE	Kabellos	Sa., 4. Juli 2015 18:00
14	Landwehrkanal	52.495863	13.411131	65	DE	-	So., 19. Juli 2015 20:00
15	Heim	52.558322	13.391544	65	DE	-	Mo., 3. Aug. 2015 23:00
16	Heim	52.558322	13.391544	65	DE	-	Di., 18. Aug. 2015 02:00
17	HTW T.	52.492805	13.525629	65	DE	-	Mi., 2. Sep. 2015 07:00
18	Arbeit	52.514710	13.390465	65	DE	-	Do., 17. Sep. 2015 09:00
19	HTW W.	52.456972	13.526578	65	DE	-	Fr., 2. Okt. 2015 12:00
20	Heim	52.558322	13.391544	65	DE	Kabellos	Sa., 17. Okt. 2015 15:00
21	Heim	52.558322	13.391544	65	DE	Kabellos	So., 1. Nov. 2015 18:00
22	Arbeit	52.514710	13.390465	65	DE	SMF	Mo., 16. Nov. 2015 20:00
23	Berlin Hbf	52.525014	13.369447	2000	DE	-	Di., 1. Dez. 2015 23:00
24	Heim	52.558322	13.391544	1414	DE	Kabellos	Mi., 16. Dez. 2015 02:00

Wetter	Temperatur	Tageshöchsttemperatur	Tagestiefstemperatur	Windgeschwindigkeit	Luftfeuchtigkeit	Sichtweite	Tageslicht	Wetterlage	Wetterlage Code
1	3,4	3,8	2,2	7	?	?	0,0	?	?
2	1,1	1,6	0,6	3	?	?	1,0	?	?
3	-0,2	0	-1,8	5	?	?	1,0	?	?
4	7	8	0	4	?	?	1,0	?	?
5	6,5	8	1	3	?	?	0,0	?	?
6	5,5	6	3	4	?	?	0,0	?	?
7	2	10	-1	2	?	?	0,0	?	?
8	6	18	4,5	3	?	?	0,0	?	?
9	17	24	14	5	?	?	1,0	?	?
10	15,5	17	9	2	?	?	1,0	?	?
11	21	22	9	3	?	?	1,0	?	?
12	16	17,5	11,5	3,5	?	?	1,0	?	?
13	34	37	22	2	?	?	1,0	?	?
14	17	21,5	16	3	?	?	1,0	?	?
15	21	32	14	2	?	?	0,0	?	?
16	20,3	20,3	17	2	?	?	0,0	?	?
17	15	21	12	6	?	?	1,0	?	?
18	22	26	14	3	?	?	1,0	?	?
19	18	20	4	2	?	?	1,0	?	?
20	9,1	9,1	7,6	2	?	?	1,0	?	?
21	11,5	15,5	4,5	2	?	?	0,0	?	?
22	11,3	12	8,5	4	?	?	0,0	?	?
23	5,5	7,5	4,5	3	?	?	0,0	?	?
24	4,7	7,5	4,7	2	?	?	0,0	?	?

## E. Empfehlungen laut Analyse A

Kombinationen der Analyse A - Aufenthaltsort																								Kombinationen der Analyse A - Stunde																								
Kombinationen der Analyse A - Wochentag																								Kombinationen der Analyse A - Wetter																								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
1																									1																							
2	0																								2	1																						
3	0	0																							3	0	0																					
4	0	0	0	1																				4	0	0	0																					
5	0	1	0	0	0																			5	0	0	0	0	0																			
6	1	0	0	0	0	0																		6	0	0	0	0	0	0																		
7	1	0	0	0	0	0	1																7	0	0	0	0	0	0	1																		
8	1	0	0	0	0	0	0	1	1													8	0	0	0	0	0	0	0	0																		
9	0	0	1	1	0	0	0	0	0	0												9	1	1	0	0	0	0	0	0	0																	
10	0	1	0	0	0	1	0	0	0	0												10	1	1	0	0	0	0	0	0	0	0	1															
11	0	0	1	1	0	0	0	0	0	0	1											11	0	0	1	0	0	0	0	0	0	0	0	0														
12	0	1	0	0	0	1	0	0	0	0	0	1									12	0	0	0	1	0	0	0	0	0	0	0	0															
13	1	0	0	0	0	0	1	1	0	0	0	0									13	0	0	0	0	0	1	0	0	0	0	0	0															
14	0	0	0	0	0	0	0	0	0	0	0	0									14	0	0	0	0	0	1	1	0	0	0	0	0															
15	1	0	0	0	0	0	1	1	1	0	0	0	0								15	0	0	0	0	0	1	1	0	0	0	0	0	1														
16	1	0	0	0	0	0	1	1	1	0	0	0	0	1							16	0	0	0	0	0	0	0	1	0	0	0	0	0														
17	0	0	1	1	0	0	0	0	0	0	0	1	0	0							17	1	1	0	0	0	0	0	0	0	1	1	0	0	0													
18	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0						18	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1												
19	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0						19	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0											
20	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0		20	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
21	1	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	1		21	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
22	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0		22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
23	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0		23	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0		
24	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0		24	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0		

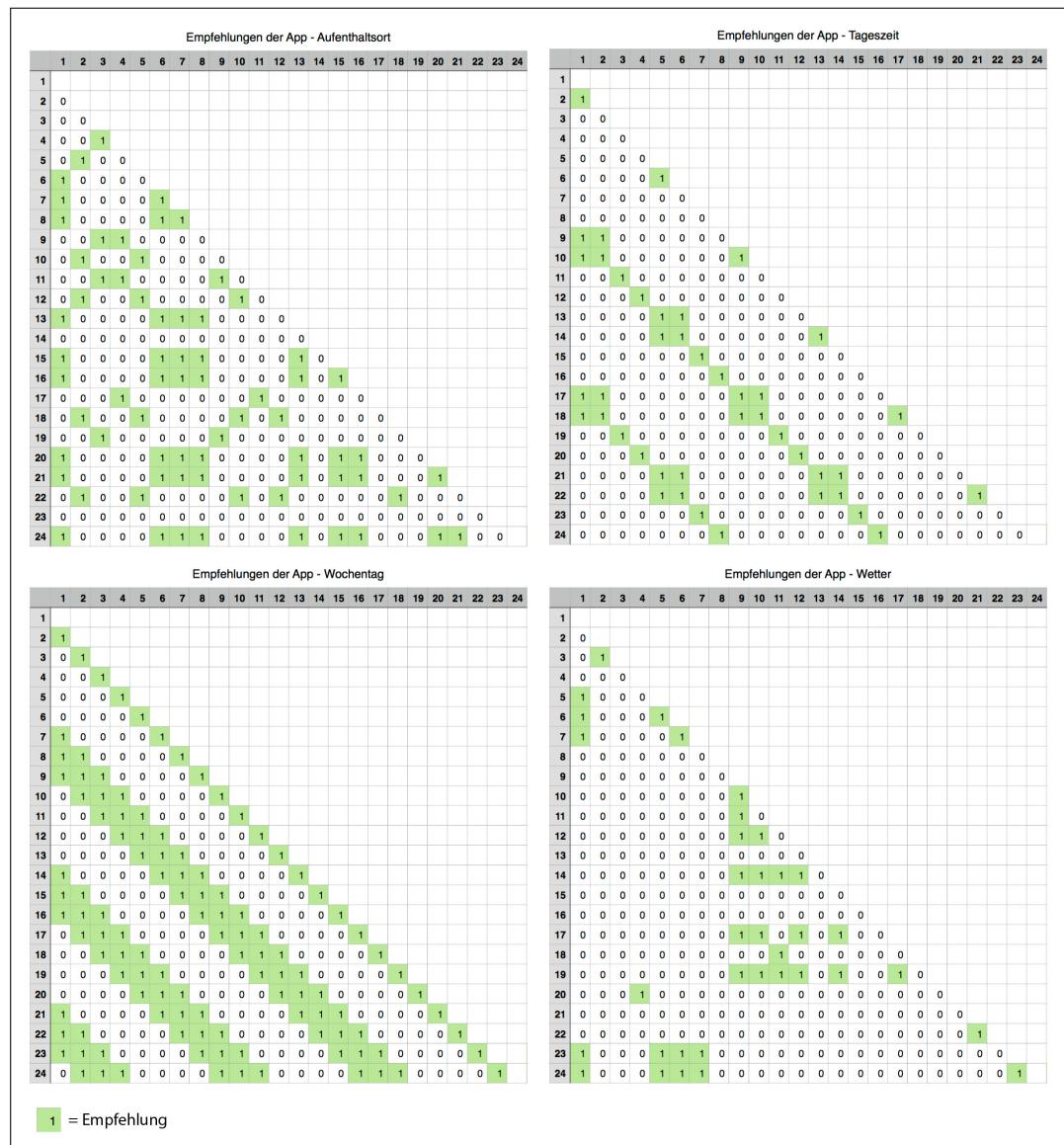
1 = Empfehlung

## F. Empfehlungen laut Analyse B

Kombinationen der Analyse B - Aufenthaltsort																								Kombinationen der Analyse B - Stunde																								
Kombinationen der Analyse B - Wochentag																								Kombinationen der Analyse B - Wetter																								
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
1																									1																							
2	0																								2	1																						
3	0	0																							3	0	0																					
4	0	0	1																						4	0	0	1																				
5	0	1	0	0																					5	0	0	0	0																			
6	1	0	0	0	0																				6	0	0	0	0	1																		
7	1	0	0	0	0	0	1																	7	0	0	0	0	0	0																		
8	0	0	0	0	0	0	0	0																8	0	0	0	0	0	0	0																	
9	0	0	1	1	0	0	0	0																9	1	1	0	0	0	0	0	0																
10	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1																							
11	0	0	1	1	0	0	0	0	0	0	1	0												11	0	0	1	1	0	0	0	0	0	0														
12	0	1	0	0	0	1	0	0	0	0	0	1	0										12	0	0	1	1	0	0	0	0	0	0	1														
13	1	0	0	0	0	1	1	0	0	0	0	0	0										13	0	0	0	0	1	1	0	0	0	0	0	0													
14	0	0	0	0	0	0	0	0	0	0	0	0	0										14	0	0	0	0	0	1	1	0	0	0	0	0	1												
15	0	0	0	0	0	0	0	0	0	0	0	0	0										15	0	0	0	0	0	0	0	0	0	0	0	0	0												
16	0	0	0	0	0	0	0	0	0	0	0	0	0	1									16	0	0	0	0	0	0	0	0	0	0	0	0	0												
17	0	0	1	1	0	0	0	0	0	0	0	0	0	0									17	1	1	0	0	0	0	0	0	0	0	0	0	0												
18	0	1	0	0	0	1	0	0	0	0	0	0	0	0									18	1	1	0	0	0	0	0	0	0	0	0	0	0												
19	0	0	1	1	0	0	0	0	0	0	0	0	0	0									19	0	0	1	1	0	0	0	0	0	0	0	0	0												
20	1	0	0	0	0	1	1	0	0	0	0	0	0	0								20	0	0	1	0	0	0	0	0	0	0	0	0	0													
21	1	0	0	0	0	1	1	0	0	0	0	0	0	0								21	0	0	0	1	1	0	0	0	0	0	0	0	0													
22	0	1	0	0	0	1	0	0	0	0	0	0	0	0								22	0	0	0	0	0	1	0	0	0	0	0	0	0													
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0								23	1	0	0	0	0	1	1	0	0	0	0	0	0													
24	1	0	0	0	0	1	1	0	0	0	0	0	0	0								24	1	0	0	0	1	1	1	0	0	0	0	0	0													

1 = Empfehlung

#### G. Empfehlungen laut App



## H. Ergebnisse der Auswertung - Überblick

Ergebnisse des Datensatzes A zu B (Ziel)						
	Aufenthaltsort	Tageszeit	Wochentag	Wetter	Ø	
<b>Anzahl der Kombination:</b>	276		276		276	
Ergebnis übereinstimmend:	255 <b>92,4 %</b>	249 <b>90,2 %</b>	213 <b>77,2 %</b>	242 <b>87,7 %</b>	240 <b>86,9 %</b>	
Ergebnis unterschiedlich:	21 <b>7,6 %</b>	27 <b>9,8 %</b>	63 <b>22,8 %</b>	34 <b>12,3 %</b>	36 <b>13,1 %</b>	
<b>Empfehlungen in B (Ziel):</b>	54		51		141	
Anzahl der fehlenden Empfehlungen:	0 <b>0,0 %</b>	18 <b>35,3 %</b>	18 <b>12,8 %</b>	16 <b>48,5 %</b>	13 <b>18,6 %</b>	
<b>Empfehlungen in A:</b>	75		42		168	
davon zutreffend:	54 <b>72,0 %</b>	33 <b>78,6 %</b>	123 <b>73,2 %</b>	17 <b>48,6 %</b>	57 <b>70,9 %</b>	
davon nicht zutreffend:	21 <b>28,0 %</b>	9 <b>21,4 %</b>	45 <b>26,8 %</b>	18 <b>51,4 %</b>	23 <b>29,1 %</b>	
Ergebnisse der App zu dem Datensatz A (Ziel)						
	Aufenthaltsort	Tageszeit	Wochentag	Wetter	Ø	
<b>Anzahl der Kombination:</b>	276		276		276	
Ergebnis übereinstimmend:	271 <b>98,2 %</b>	258 <b>93,5 %</b>	179 <b>64,9 %</b>	241 <b>87,3 %</b>	237 <b>86,0 %</b>	
Ergebnis unterschiedlich:	5 <b>1,8 %</b>	18 <b>6,5 %</b>	97 <b>35,1 %</b>	35 <b>12,7 %</b>	39 <b>14,0 %</b>	
<b>Empfehlungen in A (Ziel):</b>	75		42		168	
Anzahl der fehlenden Empfehlungen:	5 <b>6,7 %</b>	9 <b>21,4 %</b>	76 <b>45,2 %</b>	17 <b>48,6 %</b>	27 <b>33,4 %</b>	
<b>Empfehlungen der App:</b>	70		42		113	
davon zutreffend:	70 <b>100,0 %</b>	33 <b>78,6 %</b>	92 <b>81,4 %</b>	18 <b>50,0 %</b>	53 <b>81,6 %</b>	
davon nicht zutreffend:	0 <b>0,0 %</b>	9 <b>21,4 %</b>	21 <b>18,6 %</b>	18 <b>50,0 %</b>	12 <b>18,4 %</b>	
Ergebnisse der App zu dem Datensatz B (Ziel)						
	Aufenthaltsort	Tageszeit	Wochentag	Wetter	Ø	
<b>Anzahl der Kombination:</b>	276		276		276	
Ergebnis übereinstimmend:	250 <b>90,6 %</b>	267 <b>96,7 %</b>	206 <b>74,6 %</b>	263 <b>95,3 %</b>	247 <b>89,3 %</b>	
Ergebnis unterschiedlich:	26 <b>9,4 %</b>	9 <b>3,3 %</b>	70 <b>25,4 %</b>	13 <b>4,7 %</b>	30 <b>10,7 %</b>	
<b>Empfehlungen in B (Ziel):</b>	54		51		141	
Anzahl der fehlenden Empfehlungen:	5 <b>9,3 %</b>	9 <b>17,6 %</b>	49 <b>34,8 %</b>	5 <b>15,2 %</b>	17 <b>24,4 %</b>	
<b>Empfehlungen der App:</b>	70		42		113	
davon zutreffend:	49 <b>70,0 %</b>	42 <b>100,0 %</b>	92 <b>81,4 %</b>	28 <b>77,8 %</b>	53 <b>80,8 %</b>	
davon nicht zutreffend:	21 <b>30,0 %</b>	0 <b>0,0 %</b>	21 <b>18,6 %</b>	8 <b>22,2 %</b>	13 <b>19,2 %</b>	

## I. Ähnlichkeit der Empfehlungen der Analysen A zu B

Vergleich des Datensatzes A zu B (Ziel): Aufenthaltsort																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Anzahl der Kombinationen: 276																								
Ergebnis übereinstimmend: 255 92,4 %																								
Ergebnis unterschiedlich: 21 7,6 %																								
Anzahl der Empfehlungen in B: 54																								
Anzahl der in A fehlenden Empfehlungen: 0 0,0 %																								
Von A getroffene Empfehlungen: 75																								
davon zutreffend: 54 72,0 %																								
davon nicht zutreffend: 21 28,0 %																								
-1 = Empfehlung hätte getroffen werden sollen																								
1 = Empfehlung hätte nicht getroffen werden sollen																								
0 = Übereinstimmend Empfehlung oder Nicht-Empfehlung																								
Vergleich des Datensatzes A zu B (Ziel): Tageszeit																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Anzahl der Kombinationen: 276																								
Ergebnis übereinstimmend: 249 90,2 %																								
Ergebnis unterschiedlich: 27 9,8 %																								
Anzahl der Empfehlungen in B: 51																								
Anzahl der in A fehlenden Empfehlungen: 18 35,3 %																								
Von A getroffene Empfehlungen: 42																								
davon zutreffend: 33 78,6 %																								
davon nicht zutreffend: 9 21,4 %																								
-1 = Empfehlung hätte getroffen werden sollen																								
1 = Empfehlung hätte nicht getroffen werden sollen																								
0 = Übereinstimmend Empfehlung oder Nicht-Empfehlung																								
Vergleich des Datensatzes A zu B (Ziel): Wochentag																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Anzahl der Kombinationen: 276																								
Ergebnis übereinstimmend: 213 77,2 %																								
Ergebnis unterschiedlich: 63 22,8 %																								
Anzahl der Empfehlungen in B: 141																								
Anzahl der in A fehlenden Empfehlungen: 18 12,8 %																								
Von A getroffene Empfehlungen: 168																								
davon zutreffend: 123 73,2 %																								
davon nicht zutreffend: 45 26,8 %																								
-1 = Empfehlung hätte getroffen werden sollen																								
1 = Empfehlung hätte nicht getroffen werden sollen																								
0 = Übereinstimmend Empfehlung oder Nicht-Empfehlung																								
Vergleich des Datensatzes A zu B (Ziel): Wetter																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Anzahl der Kombinationen: 276																								
Ergebnis übereinstimmend: 242 87,7 %																								
Ergebnis unterschiedlich: 34 12,3 %																								
Anzahl der Empfehlungen in B: 33																								
Anzahl der in A fehlenden Empfehlungen: 16 48,5 %																								
Von A getroffene Empfehlungen: 35																								
davon zutreffend: 17 48,6 %																								
davon nicht zutreffend: 18 51,4 %																								
-1 = Empfehlung hätte getroffen werden sollen																								

#### J. Ähnlichkeit der Empfehlungen der App zu Analyse A

Vergleich des Datensatzes A: Aufenthaltsort																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
Anzahl der Kombinationen:																								
3	0	0																						
Ergebnis übereinstimmend:																								
4	0	0	0																					
5	0	0	0	0																				
6	0	0	0	0	0																			
7	0	0	0	0	0	0																		
Anzahl der gewünschten Empfehlungen:																								
8	0	0	0	0	0	0	0																	
9	0	0	0	0	0	0	0	0																
10	0	0	0	0	0	0	0	0	0															
11	0	0	0	0	0	0	0	0	0	0														
12	0	0	0	0	0	0	0	0	0	0	0													
13	0	0	0	0	0	0	0	0	0	0	0	0												
14	0	0	0	0	0	0	0	0	0	0	0	0	0											
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
17	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1 = Empfehlung hätte getroffen werden sollen																								
1 = Empfehlung hätte nicht getroffen werden sollen																								
0 = Übereinstimmung Empfehlung oder Nicht-Empfehlung																								
Vergleich des Datensatzes A: Wochentag																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
Anzahl der Kombinationen:																								
3	-1	0																						
Ergebnis übereinstimmend:																								
4	-1	-1	0																					
5	-1	-1	-1	0																				
6	0	0	0	0	1																			
7	1	0	0	0	0	0																		
Anzahl der gewünschten Empfehlungen:																								
8	0	0	-1	-1	-1	0																		
9	0	0	0	1	-1	-1	0																	
10	-1	0	0	0	0	1	0																	
11	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	-1	-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	-1	-1	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	-1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	-1	-1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	-1	-1	-1	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	-1	-1	0	1	0	0	-1	0	0	1	0	0	-1	-1	0	0	1	0	0	0	0	0
23	0	0	0	-1	-1	0	0	0	0	-1	0	0	0	0	0	-1	-1	0	0	0	0	0	0	0
24	-1	0	0	-1	0	0	-1	0	0	0	-1	0	0	0	-1	0	0	-1	0	0	0	-1	0	0
-1 = Empfehlung hätte getroffen werden sollen																								
1 = Empfehlung hätte nicht getroffen werden sollen																								
0 = Übereinstimmung Empfehlung oder Nicht-Empfehlung																								
Vergleich des Datensatzes A: Tageszeit																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
Anzahl der Kombinationen:																								
3	0	0	0																					
Ergebnis übereinstimmend:																								
4	0	0	0	0																				
5	0	0	0	0	0																			
6	0	0	0	0	0	1																		
7	0	0	0	0	0	0	1																	
Anzahl der gewünschten Empfehlungen:																								
8	0	0	0	0	0	0	0	1																
9	0	0	0	0	0	0	0	0	1															
10	0	0	0	0																				

## K. Ähnlichkeit der Empfehlungen der App zu Analyse B

Vergleich des Datensatzes B: Aufenthaltsort																								
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
3	0	0																						
4	0	0	0																					
5	0	0	0	0																				
6	0	0	0	0	0																			
7	0	0	0	0	0	0																		
8	1	0	0	0	0	0	1	1																
9	0	0	0	0	0	0	0	0	0															
10	0	0	0	0	0	0	0	0	0	0														
11	0	0	0	0	0	0	0	0	0	0	0													
12	0	0	0	0	0	0	0	0	0	0	0	0												
13	0	0	0	0	0	0	0	0	1	0	0	0												
14	0	0	0	0	0	0	0	0	0	0	0	0	0											
15	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0									
16	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0									
17	0	0	-1	0	0	0	0	-1	0	0	0	0	0	0	0	0								
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
19	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0	-1	0								
20	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0							
21	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0						
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
24	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	-1	= Empfehlung hätte getroffen werden sollen																						
	1	= Empfehlung hätte nicht getroffen werden sollen																						
	0	= Übereinstimmend Empfehlung oder Nicht-Empfehlung																						
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
3	-1	0																						
4	-1	-1	0																					
5	0	0	0	1																				
6	0	0	0	0	0																			
7	1	0	0	0	-1	0																		
8	0	0	-1	-1	0	0	1																	
9	0	0	0	1	0	0	0	0																
10	-1	0	0	0	0	0	0	-1	0															
11	-1	-1	0	0	1	0	0	-1	0															
12	0	0	0	1	0	0	-1	0	0	0	1													
13	0	0	0	0	0	0	0	0	0	0	0	0												
14	1	0	0	0	-1	0	0	1	0	0	-1	0												
15	0	0	-1	-1	0	0	0	1	0	0	-1	0	0	1										
16	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0									
17	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0								
18	-1	-1	0	0	1	0	0	-1	0	0	0	1	0	0	-1	-1	0							
19	0	0	1	0	0	-1	0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	1	0	0	-1	0	0	1	0	0	-1	0	0	1	0	0	-1	0	0	0	0	0	0	0	
22	0	0	-1	-1	0	0	1	0	0	-1	0	0	1	0	0	-1	0	0	0	0	0	0	0	
23	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	
24	-1	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	-1	0	0	
	-1	= Empfehlung hätte getroffen werden sollen																						
	1	= Empfehlung hätte nicht getroffen werden sollen																						
	0	= Übereinstimmend Empfehlung oder Nicht-Empfehlung																						
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
3	0	0																						
4	0	0	0	-1																				
5	0	0	0	0	0																			
6	0	0	0	0	0	0																		
7	0	0	0	0	0	0	-1	0																
8	0	0	0	0	0	0	0	0																
9	0	0	0	0	0	0	0	0	0															
10	0	0	0	0	0	0	0	0	0	0														
11	0	0	0	0	0	0	0	0	0	0	0													
12	0	0	0	0	0	0	0	0	0	0	0	0												
13	0	0	0	0	0	0	0	0	0	0	0	0	0											
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0									
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
24	-1	0	0	0	0	0	0	-1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0
	-1	= Empfehlung hätte getroffen werden sollen																						
	1	= Empfehlung hätte nicht getroffen werden sollen																						
	0	= Übereinstimmend Empfehlung oder Nicht-Empfehlung																						
Kontext	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1																								
2	0																							
3	0	0																						
4	0	0	0	-1																				
5	0	0	0	0	0																			
6	0	0	0	0	0	0					</													

## **L. Beiliegende CD**

Die beiliegende CD beinhaltet:

- diese Bachelorarbeit als PDF
- den Quellcode der App
- den Testdatensatz als JSON Export

## **Eidesstattliche Versicherung**

Ich versichere hiermit, dass ich die vorliegende Bachelorthesis selbstständig und ohne fremde Hilfe angefertigt und keine andere als die angegebene Literatur benutzt habe. Alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit sind besonders gekennzeichnet. Diese Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Berlin, 08.03.2016

---

Hans Seiffert