Konzeption und Realisierung einer Client-Server-Lösung zur Erfassung von Fahrplandaten im öffentlichen Personenverkehr

Bachelorarbeit

im Studiengang
Medieninformatik

vorgelegt von

Stefan Humm
Matr.-Nr.: 26205

am 01. August 2016
an der Hochschule der Medien Stuttgart

Erstprüfer/in:	Prof. Walter Kriha
Zweitprüfer/in:	Dipl.-Ing. Thomas Hecker

Sperrvermerk

Die vorliegende Bachelorarbeit mit dem Titel „Konzeption und Realisierung einer Client-Server-Lösung zur Erfassung von Fahrplandaten im öffentlichen Personenverkehr“ von Stefan Humm enthält vertrauliche Daten der Firma moovel Group GmbH.

Die Bachelorarbeit darf nur dem Erst- und Zweitgutachter sowie
befugten Mitgliedern des Prüfungsausschusses zugänglich gemacht werden.
Eine Veröffentlichung und Vervielfältigung der Bachelorarbeit
ist - auch in Auszügen - nicht gestattet.
Die Sperrfrist beträgt 2 Jahre, beginnend mit dem auf dem Titelblatt
vermerkten Vorlagedatum.

Eine Einsichtnahme der Arbeit durch Unbefugte bedarf während der Sperrfrist einer
ausdrücklichen Genehmigung des Verfassers und der
Firma moovel Group GmbH.

Eidesstattliche Versicherung

Name:	Humm	Vorname:	Stefan
Matrikel-Nr.:	26205	Studiengang:	Medieninformatik

Hiermit versichere ich, Stefan Humm, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Konzeption und Realisierung einer Client-Server-Lösung zur Erfassung von Fahrplandaten im öffentlichen Personenverkehr“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), §23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Stuttgart, 29.07.2016 _____

Ort, Datum Unterschrift

Kurzfassung

Die ständige Verfügbarkeit von Informationen ist heutzutage ein nicht mehr weg zu denkendes Gut. Fortwährend werden neue Informationen digital erfasst. Im Bereich von Fahrplandaten gibt es hierzu noch Potenzial, da die freie Verfügbarkeit von Fahrplandaten weltweit noch stark begrenzt ist. Nur wenige Verkehrsbetriebe sind bereit zusätzlichen Aufwand zu betreiben, um ihre Daten der Öffentlichkeit frei zugänglich zu machen oder bieten diese zum Verkauf an. Darüber hinaus gibt es in vielen sozial schwachen Gebieten auf der Welt, aufgrund schlecht ausgebauter Infrastruktur, einen Mangel an verfügbaren Fahrplandaten. Diesem Mangel an freien und offenen Fahrplandaten soll mittels Crowdsourcing entgegengewirkt werden. Diese Bachelorarbeit beschreibt die Konzeption und Realisierung einer Client-Server-Lösung zur Erfassung von Fahrplandaten im öffentlichen Personenverkehr. Das hierzu entwickelte Konzept beschreibt, wie die Daten von Nutzern einfach erfasst werden können, die Kommunikation mit dem Server realisiert und wie die Verwendbarkeit des Clients für die Mehrheit der Menschen in verschiedenen Ländern gewährleistet werden kann. Als Ergebnis werden die Daten vom Server im international standardisierten GTFS-Format (General Transit Feed Specification) bereitgestellt.

Abstract

The permanent availability of information is part of our today’s world, we do not want to miss anymore. Continuously, new information is captured digitally. The availability of free timetable data is globally still very limited. This offers potential for improvements. Only a few transportation organisations are willing to provide public access to their data or offer it for sale. Moreover, there are areas of deprivation in the world, which have a lack of available timetable data because of weak infrastructure. Crowdsourcing can be one method to fill this gap and to enable open access to timetable data for public use. This bachelor thesis is about the conception and realization of a client-server solution capturing timetable data of public transportation. The concept describes how users themselves can easily capture data and how the communication between server and client can be realized. Furthermore, it defines how the usability of the client needs to be designed in order to allow access for the majority of people living in different countries worldwide. As a result, the data on the server will be provided in the international standardized GTFS format (General Transit Feed Specification).

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
APK	Android Package
BSD	Berkeley Software Distribution
CC-BY-SA	Creative Commons / Share Alike
CSV	Comma-separated Values
GmbH	Gesellschaft mit beschränkter Haftung
GTFS	General Transit Feed Specification
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HdM	Hochschule der Medien
ID	Identification
JSON	JavaScript Object Notation
LGPL	Library General Public License
MIT	Massachusetts Institute of Technology
MySQL	My Structured query Language
ODbL	Open Data Commons Open Database License
OS	Operating System
OSM	OpenStreetMap
REST	Representational State Transfer
SDK	Software Development Kit
SQL	Structured Query Language
UML	Unified Modeling Language

1 Einleitung

Die Bachelorarbeit entstand in Zusammenarbeit mit der moovel Group GmbH. Moovel ist ein hundertprozentiges Tochterunternehmen der Daimler AG. Sie verbindet intelligent verschiedene Mobilitätsanbieter zur Bereitstellung von Mobilitätsdiensten. Zu den Produkten zählen unter anderem Webanwendungen und mobile Apps zur Navigation im öffentlichen Personen- und Individualverkehr (vgl. Moovel, 2016).

In den beiden folgenden Abschnitten wird zunächst die Entstehung des GTFS-Formates beschrieben und anschließend auf das Potential von offenen Daten und Anwendungsmöglichkeiten eingegangen, sowie die Zielsetzung dieser Bachelorarbeit definiert.

1.1 Motivation

Für viele Menschen ist es bei der Navigation von A nach B zum Standard geworden, über eine Webseite oder einer App eine entsprechende Verbindung im öffentlichen Verkehr zu erfragen. Wie am Beispiel Österreichs zu sehen ist, besitzen 58% der 15- bis 29-Jährigen eine Verkehrsmittel-App (vgl. Statista, 2015a). Die Daten für solche Apps und Webseiten stammen von den einzelnen Verkehrsbetrieben und werden, entweder für ihre eigenen Applikationen oder für Drittanbieter, über entsprechende Schnittstellen bereitgestellt. Oft sind diese Fahrplandaten in proprietären Formaten und nicht kompatibel mit den Formaten anderer Anbieter. Das führt dazu, dass Unternehmen, die Fahrplandaten verschiedener Quellen kombinieren möchten, vor eine große Herausforderung gestellt werden.

Gemeinsam mit dem Verkehrsunternehmen TriMet entwickelte Google die GTFS-Spezifikation und damit den ersten offenen Standard für Fahrplandaten (vgl. Google Blog, 2005). Dieser ermöglicht Fahrplandaten standardisiert für die Öffentlichkeit zugänglich zu machen. Google und diverse weitere Kartenanbieter konnten hierdurch ihre Dienste aufwerten und mit Fahrplandaten der einzelnen Verkehrsunternehmen anreichern.

Bei Betrachtung von Sammlungen von GTFS-Daten im Internet wie beispielsweise bei TransitFeeds[1] wird deutlich, dass es in Deutschland im Vergleich zu den USA, noch immer wenige Verkehrsunternehmen gibt, die ihre Fahrplandaten öffentlich bzw. nur auf Anfrage im einheitlichen GTFS-Format zur Verfügung stellen. Die Konvertierung in das GTFS-Format und die Wartung der Daten ist mit hohen Kosten verbunden. Daher zögern viele Unternehmen ihre Daten zu veröffentlichen. Dagegen kann die Bereitstellung in Gebieten mit mehreren Anbietern existenziell notwendig sein, um im Konkurrenzkampf zu bestehen und das Unternehmensimage in der Öffentlichkeit, als Anbieter

[1] TransitFeeds Webseite: https://transitfeeds.com (Datum des Zugriffs: 23. Juli 2016)

von offenen Daten, zu steigern. Darüber hinaus ist die Situation in Ländern ohne Verkehrsunternehmen zu betrachten. Dort gibt es oft für den öffentlichen Verkehrsbetrieb weder Webseiten, über die Fahrpläne abgerufen werden können, noch mobile Anwendungen. Häufig sind gedruckte Fahrpläne nicht vorhanden und man ist darauf angewiesen, sich direkt bei dem jeweiligen Transportmittel über den Zielort und die Haltestellen zu erkundigen. Dies macht es zum Beispiel für Reisende schwer, bereits vorab eine Route beziehungsweise ihren Aufenthalt dort zu planen.

Mit der gemeinschaftlichen Erhebung von Fahrplandaten durch Crowdsourcing könnten Anbieter von Mobilitätsplattformen die Daten in ihre Anwendung integrieren und es könnten Fahrplandaten für Gebiete bereitgestellt werden, in denen zuvor keine Informationen vorhanden waren. Dies wäre ein Schritt weiter in Richtung offene Daten und der freien Verfügbarkeit von Fahrplandaten.

1.2 Zielsetzung

Es soll untersucht werden, wie eine Client-Server-Struktur aufgebaut werden kann, um mittels Crowdsourcing Fahrplandaten zu erfassen und diese zentral auf einem Server zu sammeln. Bei den Nutzern des Clients handelt es sich um natürliche Personen, die freiwillig bei der Erfassung von Fahrplandaten mitwirken möchten. Hierzu soll die Eingabe möglichst einfach gestaltet werden und mobil möglich sein. Dabei gilt zu prüfen, welche Anforderungen der Client erfüllen muss und welche Technologien sich hierfür eignen, um auch in Infrastrukturschwachen Gebieten eine vollständige Funktionalität zu gewährleisten. Die gesammelten Daten sollen schließlich in Form von GTFS-Daten vom Server, öffentlich für die Nutzung Dritter, bereitgestellt werden. Hierzu soll ein entsprechendes Konzept erarbeitet und darauf aufbauend ein Prototyp realisiert werden.

1.3 Begriffsdefinition

In den nachfolgenden Kapiteln werden Bahnhöfe und Haltestellen einheitlich als „Stationen“ bezeichnet.

2 Stand der Technik

Im folgenden Abschnitt wird in die relevanten Standards und Formate eingeführt. Außerdem werden existierende Lösungen untersucht und miteinander verglichen, um festzustellen ob diese bereits die Zielstellung (siehe Abschnitt 1.2) erfüllen.

2.1 Kartendaten

Karten lassen sich auf digitalen Geräten entweder raster- oder vektorbasiert darstellen. Rasterkarten sind optisch mit Papierkarten vergleichbar und werden in ein Raster eingeteilt. Jede Zelle eines Rasters besitzt eine eigene Kennung und alle Informationen dieses Kartenabschnittes lassen sich anhand dieser Kennung zusammenführen (vgl. Davis, 2001, S.63). Darüber hinaus besitzen Rasterkarten nur eine Ebene. Alle Informationen sind auf dieser Ebene zusammengeführt und lassen keine individuelle Gestaltung der Karten zu (vgl. Liqiu, 2008, S.130). Vektorkarten sind gegenüber Rasterkarten deutlich präziser (vgl. Davis, 2001, S.89). Unabhängig von der Vergrößerungsstufe können die Karten ohne Qualitätsverlust skaliert und dargestellt werden (vgl. Dickmann, 2004, S.50). Des Weiteren ist es möglich Bereiche der Karte auf verschiedenen unabhängigen Ebenen aufzuteilen (vgl. Liqiu, 2008, S.130) und damit die Karte sehr flexibel zu machen. Objekte können ein- oder ausgeblendet werden und Vektoren können Eigenschaften mitgegeben werden (vgl. Blömeke, 2014, S.27). Dadurch ist es möglich eine Vektorkarte zu personalisieren und individuell zu gestalten.

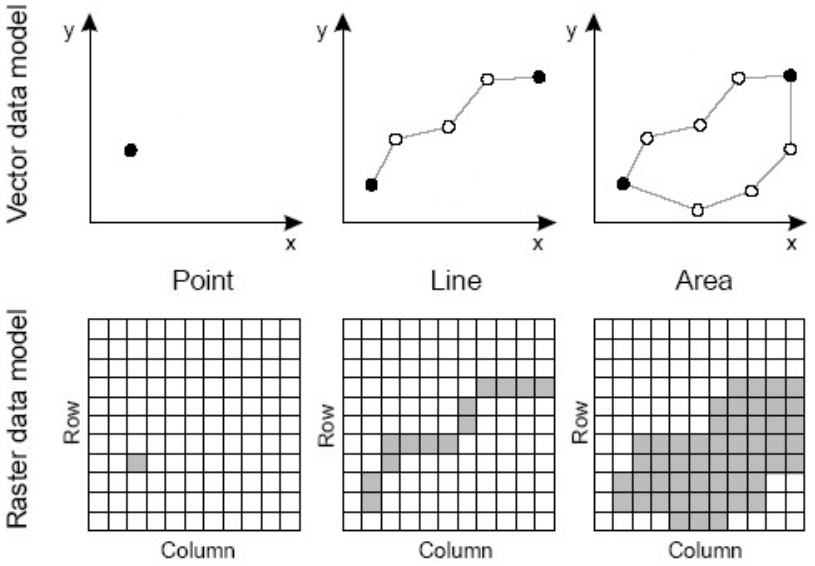


Abbildung 1: Vergleich von Raster- und Vektortechnik

Die Abbildung 1 vergleicht die beiden Techniken anhand von drei Beispielen. Das erste Beispiel bezieht sich auf einen einzelnen Punkt (Knoten). Während bei der Vektortech-

nik nur die Information des Knotens benötigt wird, muss bei der Rastertechnik ebenfalls die Information aller anderen Zellen vorhanden sein. Das zweite Beispiel beschreibt wie eine Linie durch mehrere Knoten mittels Kanten verbunden und definiert werden kann. Das letzte Beispiel zeigt die Realisierung eines Polygons. Dieses startet und endet im selben Knoten und schließt eine Fläche ein.

2.2 Kartendienste

Kartendienste sind Online-Dienste, welche Kartenmaterial unter Berücksichtigung der Lizenzvereinbarungen öffentlich zugänglich machen. Durch Navigationselemente und Zoomfunktionen ist es möglich, sich innerhalb der Karte zu bewegen. Nutzer sind in der Lage Wegbeschreibungen abzurufen, sowie nach Orten und Unternehmen zu suchen. Außerdem haben sie die Wahl, sich zwischen verschiedenem Bildmaterial für die Ansicht zu entscheiden. So bieten die meisten Dienste an, die Karte als Straßenkarte oder Luft beziehungsweise Satellitenbild zu betrachten.

Damit die Karten performant dargestellt werden können, werden Raster- und Vektorkarten in ein Gitter aufgeteilt. Jede Zelle dieses Gitters ist eine Kachel, welche im Vorfeld vom Server berechnet und gecacht wird. Damit Kartendienste auch bei höheren Vergrößerungsstufen detailliertere Informationen in den Kacheln anzeigen können, wird für jede Vergrößerungsstufe eine eigene Kachelgröße definiert. Das bedeutet, dass die Reichweite pro Kachel bei steigender Vergrößerungsstufe abnimmt und der Informationsgehalt pro Kachel zunimmt (vgl. Mapbox, 2016a). Abbildung 2 zeigt beispielhaft das Gitternetz der Erde bei Vergrößerungsstufe 2 der GoogleMaps API (s.u.).

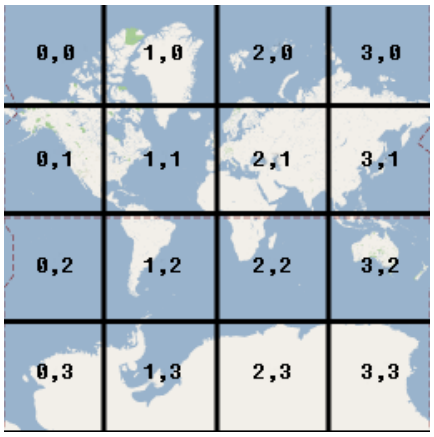


Abbildung 2: Gitternetz bei Vergrößerungsstufe 2 der GoogleMaps API

2.2.1 Google Maps

Google Maps ist neben Apple Maps und Bing Maps einer der bekanntesten Kartendienste in Deutschland. Der Kartendienst von Google startete im Jahr 2005 und bietet die Möglichkeit die Welt digital zu erkunden, Routen zu planen oder die Karte auf der eigenen Webseite einzubinden. Für nicht kommerzielle Nutzung ist der Dienst frei

verfügbar, sofern die Nutzungsbedingungen von Google eingehalten werden. Google Maps bietet ebenfalls eine API für Entwickler an, welche eine kommerzielle Nutzung und Einbindung in Projekten möglich macht (vgl. Svennerberg, 2010, S.2-3). Die geographischen Vermessungsdaten und das Kartenmaterial für den Dienst stammen unter anderem von Landesvermessungsämtern und kommerziellen Anbietern (vgl. Google, 2016).

2.2.2 OpenStreetMap

OpenStreetMap ist ein 2004 gegründetes Projekt zur Erfassung von geografischen Daten und macht diese in einer Datenbank frei für jeden zugänglich. Beginnend mit dem Straßennetz, über Gebäude, bis hin zu Pipelines werden alle Daten mittels Crowdsourcing zusammengetragen (vgl. Bennett, 2010, Chapter 1 [o.S.]). Der Begriff Crowdsourcing ist ein durch das Internet ermöglichtes Phänomen, das „[...] den interaktiven und kollaborativen Wissensaustausch [...]“ zulässt. Kunden oder Nutzer werden hierbei an der Lösung eines Problems zur Wertschöpfung und dem gegenseitigen Vorteil beteiligt (vgl. Yavuz, 2011, S.1). Bei OpenStreetMap werden die gesammelten Daten verwendet, um digitale Karten zu erstellen oder diese zur Navigation einzusetzen. Die Daten stehen unter der „Open Database Licence (ODbL) 1.0“ und dürfen daher von jedem frei und für jeden Zweck, unter Berücksichtigung der Nutzungsbedingungen, verwendet werden (vgl. ODbL, 2016). Daher wird OpenStreetMap häufig für Anwendungen oder von Unternehmen eingesetzt, welche die Daten mit weiteren Informationen anreichern oder aufbereiten, um sie für ihre eigenen Produkte zu verwenden.

2.2.3 Weitere Kartendienste

Neben Google Maps und OpenStreetMap existieren noch einige andere Kartendienste. Viele dieser Dienste erhalten Kartenmaterial von Drittanbietern wie Google Maps und einige basieren auf den geographischen Daten von OpenStreetMap. Tabelle 1 zeigt eine Auflistung von Kartendiensten, deren Lizenz, Kachelformat und Datenbasis.

Tabelle 1: Auflistung von Kartendiensten

Dienst	**Lizenz**	**Kachelformat**	**Basiert auf**
Google Maps	Copyright	Vektor/Raster	
Bing Maps	Copyright	Raster	
HERE Maps	Copyright	Vektor/Raster	
Apple Maps	Copyright	Vektor/Raster	
OpenStreetMap	ODbL 1.0	Raster	
Mapbox	BSD	Vektor/Raster	OSM
MapQuest Open	CC-BY-SA	Raster	OSM

MapForge	LGPL3	Vektor	OSM
Mapzen	MIT	Vektor	OSM

2.3 Fahrplandaten

Die Planung und der Betrieb von öffentlichen Verkehrsmitteln erfordern, erzeugen und verwenden eine Vielzahl von Daten. Zu diesen Daten zählen unter anderem Bahnhöfe und Haltestellen. Neben ihrem Namen und geographischen Position, können sie zusätzliche detailliertere Informationen beinhalten, wie zum Beispiel Informationen zur Barrierefreiheit. Eine Gruppe von mindestens zwei verbundenen Stationen ergibt ein Verbindungsnetz. Ausgehend von diesem Verbindungsnetz können Routen definiert werden. Diese beschreiben die sequenzielle Reihenfolge der befahrenen Stationen und besitzen eine Kennung, wie zum Beispiel Bus „N8“, welche auf die Route zurückschließen lässt. Für die Zeitplanung der Routen werden diese einem Serviceplan untergeordnet. Dieser Serviceplan beschreibt zu welchen Uhrzeiten, an welchen Tagen der Service angeboten wird und wann es zu Ausnahmen vom Zeitplan kommt (vgl. Kaufmann, S.9, 2014). Aus der Summe dieser Daten ist es möglich Fahrpläne zu erstellen und können daher zusammengefasst als Fahrplandaten bezeichnet werden.

2.3.1 GTFS

GTFS ist ein von Google und dem Verkehrsunternehmen TriMet entwickelter Standard für Fahrplandaten im öffentlichen Personenverkehr und den damit verknüpften geografischen Informationen. Dieser Standard ermöglicht Fahrplandaten der Öffentlichkeit in einem international einheitlichen Format zur Verfügung zu stellen. Dadurch können Anwendungen entwickelt werden, die leicht für andere Länder angepasst werden können. Das Format ist an das CSV-Format von Microsoft Excel angelehnt, um es möglichst einfach zu halten. Das bedeutet, dass die einzelnen Attribute mittels eines Kommas getrennt werden und jede Zeile ein separater Eintrag ist. Neben einigen notwendigen Attributen, existieren ebenfalls optionale Attribute, welche zusätzliche Informationen bereitstellen können. Darüber hinaus ist die Struktur in 13 verschiedene Dateien aufgeteilt, welche partiell über ID-Attribute untereinander verknüpft sind. Diese Dateien mit der Endung .TXT können anschließend zusammengeführt im ZIP-Format in GTFS-Feeds veröffentlicht werden. GTFS-Feeds sind Plattformen im Internet, die GTFS-Daten und Quellen sammeln, um sie zentral für die Öffentlichkeit zugänglich zu machen (vgl. Google Developers, 2016).

In Anhang A befindet sich ein UML-Diagramm über die Dateien des GTFS-Formates und dessen Attribute und Abhängigkeiten.

2.4 Marktanalyse

Die Einführung von GTFS hat dafür gesorgt, dass viele Unternehmen und Startups neue Produkte auf den Markt bringen konnten. Diese Produkte sind größtenteils kommerzielle Lösungen, welche anderen Unternehmen helfen sollen ihre eigenen Daten in das GTFS-Format zu überführen und editierbar zu machen. Da GTFS ein offenes Format ist, haben sich ebenfalls Open-Source-Lösungen etabliert. Alle Produkte sind größtenteils ähnlich aufgebaut. Sie bestehen in der Regel aus einer Karte und verschiedenen Eingabeformularen. Die Karte soll die eingetragenen Bahnhöfe, Haltestellen und Routen visuell darstellen und dem Benutzer so eine bessere Übersicht ermöglichen.

Tabelle 2 vergleicht Anbieter von GTFS-Editoren anhand von Grundfunktionalitäten, die aus den definierten Anforderungen aus Abschnitt 3 hervorgehen.

Tabelle 2: Anbieter von GTFS-Editoren

	GTFS Editor Anbieter			
	TransitEditor	**Conveyal**	**AddTransit**	**TransitArt**
Website	http://transiteditor.com	http://conveyal.com	https://addtransit.com	http://transitart.io
Kommerziell	Ja	Nein	Ja	Ja
Open-Source	Nein	Ja	Nein	Nein
Plattform	Web	Web	Web	Web
Stationen anlegen	Ja	Ja	Ja	Ja
Routen anlegen	Ja	Ja	Ja	Ja
Offline Verfügbarkeit	Nein	Nein	Nein	Nein
Kartenansicht	Ja	Ja	Ja	Nein
GTFS-Export	Ja	Ja	Ja	Ja

2.4.1 Fazit

Bei den in Tabelle 2 verglichenen Lösungen für GTFS-Editoren handelt es sich ausnahmslos um reine Webanwendungen. Die Produkte sind für die Benutzung auf einem Desktop-Computer oder Notebook ausgelegt und entgegen der Zielsetzung (siehe Abschnitt 1.2) weder für den mobilen Gebrauch optimiert, noch als mobile App verfügbar. Daher werden in den folgenden Abschnitten Anforderungen an eine eigene mobile Anwendung definiert, basierend darauf ein entsprechendes Konzept erstellt und anschließend als Prototyp realisiert.

3 Anforderungsanalyse

Im Folgenden wird die Anforderungsanalyse basierend auf der Zielsetzung (siehe Abschnitt 1.2) durchgeführt. Hierzu wurden zu Beginn Anwendungsfälle (siehe Abschnitt 3.1) erarbeitet und im Anschluss die funktionalen (siehe Abschnitt 3.2) und nichtfunktionale Anforderungen (siehe Abschnitt 3.3) davon abgeleitet. Diese Anforderungen wurden daraufhin den einzelnen Komponenten zugewiesen und in Abhängigkeit zueinander gebracht.

3.1 Anwendungsfälle

- Der Nutzer legt neue Stationen an, die noch nicht existieren
- Der Nutzer definiert eine Route zwischen Stationen
- Der Nutzer legt fest, in welchem Intervall eine Route befahren wird
- Der Nutzer legt fest, wie lange eine Route gültig ist
- Der Nutzer legt Ausnahmen fest, zu welchen Zeitpunkten eine Route nicht wie üblich befahren wird
- Der Nutzer möchte erfahren, welche Routen von einer Station abfahren
- Der Nutzer möchte eine Stationen anhand eines Suchbegriffes suchen
- Der Nutzer möchte einer Station ein Foto hinzufügen
- Der Nutzer möchte wissen, wann das nächste Verkehrsmittel einer ausgewählten Station fährt
- Der Nutzer möchte wissen, ob eine Station barrierefrei ist
- Der Nutzer möchte wissen, ob auf einer Route die Mitnahme eines Fahrrads möglich ist
- Der Nutzer speichert den aktuellen Kartenausschnitt zur späteren Offline-Verwendung auf seinem Gerät
- Der Nutzer löscht die, für die Offline-Verwendung gespeicherten, Kartendaten von seinem Gerät
- Der Nutzer überprüft, wie viel Speicherplatz die Kartendaten benötigen, die für die Offline-Verwendung gespeichert wurden
- Der Nutzer stellt das Datum und Zeitformat für seine Region ein
- Der Nutzer synchronisiert seine gesammelten Verkehrsdaten mit dem Server
- Der Nutzer entscheidet, ob er manuell synchronisieren möchte oder automatisch, sobald Internet verfügbar ist
- Der Nutzer bricht die Synchronisierung mit dem Server ab
- Der Nutzer bricht das Laden von Kartendaten ab
- Der Nutzer meldet nicht valide Daten

3.2 Funktionale Anforderungen

„Funktionale Anforderungen beschreiben Funktionen, die ein System ausführen können muss. Sie definieren die Transformationen, die ein System durchführt, um aus einem Input einen Output zu erzeugen“ (Schneider, 2001, S. 82). Des Weiteren führt Schneider aus, dass die funktionalen Anforderungen vom Wesen der Software, den erwarteten Benutzern und der Art des geplanten Systems abhängt (Schneider, 2001, S.83). Entsprechend dieser Definitionen wurden funktionale Anforderungen erstellt und in Tabelle 3 aufgelistet.

Tabelle 3: Funktionale Anforderungen

Nr.	Beschreibung	K.	A.
1	Einbinden einer Karte eines Kartendienstes	C	
2	Die Karte wird mit Gesten gesteuert (Position, Zoom)	C	1
3	Die aktuelle Position des Nutzers wird auf der Karte angezeigt	C	1
4	Darstellen von bereits eingetragenen Stationen auf der Karte	C	1
5	Anlegen und Bearbeiten von Stationen	C	
6	Anzeigen, Erstellen und Bearbeiten von Routen	C	
7	Suchen von Stationen zur leichteren Eingabe beim Bearbeiten von Routen	C	
8	Kartenausschnitt, Stationen und Routen für die Offline-Nutzung verfügbar machen	C	
9	Meldefunktion für nicht valide Daten	C	
10	Datensynchronisation zwischen Client und Server	C	
11	REST API für den Datenaustausch mit den Clients	S	
12	Generierung von GTFS-Daten	S	
13	Datenhaltung der vom Nutzer erfassten Fahrplandaten	S	

Nr. = Nummer
K. = Komponente (S = Server, C = Client)
A. = Abhängigkeiten (von Nummer)

3.3 Nicht-funktionale Anforderungen

Nach Moser (2012, S.102) lassen die nicht-funktionalen Anforderungen einen Rückschluss über die Qualität des Produktes zu. „Sie beschreiben nicht was, sondern wie etwas umgesetzt werden muss“ (Moser, 2012, S.102). Vogel (2009, S.108) sieht nicht-funktionale Anforderungen als Verkörperung von „Erwartungen und Notwendigkeiten,

die von Interessenvertretern [...] neben den funktionalen Anforderungen als wichtig erachtet werden und über die reine gewünschte Funktionalität hinausgehen“. Abgrenzend zu den funktionalen Anforderungen (siehe Abschnitt 3.2), wurde anhand dieser Definitionen die nachfolgende Tabelle mit nicht-funktionalen Anforderungen erstellt.

Tabelle 4: Nicht-funktionale Anforderungen

Nr.	Beschreibung	K.
1	Das Erstellen und Bearbeiten von Stationen und Routen soll für den Nutzer möglichst einfach gestaltet werden	C
2	Die App soll für Nutzer ohne Vorkenntnisse des GTFS-Formates bedienbar und verständlich sein	C
3	Die vom Nutzer einzutragenden Daten sollen auf die Notwenigsten reduziert sein	C
4	Der Client soll mobil einsetzbar sein	C
5	Der Client soll weltweit eingesetzt werden können	C
6	Der Client soll die größtmögliche Anzahl an Geräten unterstützen	C
7	Der Nutzer soll durch spieltypische Elemente (Gamifizierung) motiviert werden Daten zu erfassen	C
8	Geringer Datenverbrauch und Latenz bei der Client-Server-Kommunikation	S

Nr. = Nummer
K. = Komponente (S = Server, C = Client)

3.3.1 Einsatzgebiete

Der Client soll in den verschiedensten Gebieten eingesetzt werden können, um damit global den Bestand an offenen Daten voranzutreiben. Das schließt sowohl Industrieländer, als auch Schwellen- und Entwicklungsländer mit ein. Besonders in Entwicklungsländern, deren Infrastruktur oft nicht ausreichend ausgebaut ist und es daher keine Verkehrsbetriebe oder Fahrpläne an Haltestellen gibt, ist es wichtig die Erfassung von Daten voranzutreiben und frei verfügbar zu machen.

3.4 Zusammenfassung

Basierend auf der Zielsetzung (siehe Abschnitt 1.2) und den definierten Anforderungen soll der Client als Editor dienen, um schnell und einfach Bahnhöfe und Haltestellen anzulegen und anschließend Routen zwischen diesen Stationen definieren und bearbeiten zu können. Diese Routen können in Summe als Fahrpläne betrachtet werden.

Auf der Serverseite soll eine Schnittstelle definiert werden, über welche die erfassten Daten des Nutzers an den Server übertragen und abgerufen werden können. Zudem ist der Server für die Datenhaltung zuständig und kann aus diesen Daten valide GTFS-Daten generieren.

4 Systemkonzeption

Die Evaluierung der bestehenden Lösungen in Abschnitt 2.4 ergab, dass keine der untersuchten Applikationen die in Abschnitt 3 definierten Anforderungen vollständig erfüllen. Basierend auf den in Stand der Technik (siehe Abschnitt 2) beschriebenen Technologien, Dienste und den definierten Anforderungen (siehe Abschnitt 3) wird im folgenden Abschnitt eine Client-Server-Anwendung konzipiert. Zunächst werden der Systemaufbau und dessen Komponenten allgemein betrachtet. Daraufhin wird eine geeignete Plattform für den Client und die für den Funktionsumfang nötigen Frameworks gewählt. Anschließend werden die Komponenten und Schnittstellen des Applikationsservers näher betrachtet.

4.1 Systemaufbau

Um ein geeignetes Konzept für die Client-Server-Anwendung konzipieren zu können, werden zunächst die benötigten Komponenten festgelegt und anschließend die Beziehung zwischen den Komponenten definiert.

Für den Nutzer steht der Client im Mittelpunkt. Dieser Client soll nach den in Abschnitt 3.2 definierten funktionalen Anforderungen eine Karte darstellen können. Für die Kartendarstellung wird ein Kartendienst benötigt, der ein entsprechendes Application Programming Interface (API) und Software Development Kit (SDK) für die Zielplattform des Clients anbietet. Damit aus den erfassten Daten der einzelnen Clients im Anschluss GTFS-Daten generiert werden können, überträgt der Client seine Daten an einen Applikationsserver. Dieser Applikationsserver ist für die Persistenz der Daten und die Bereitstellung einer Schnittstelle, zum Abrufen der Daten im GTFS-Format, zuständig. Zusätzlich versorgt der Applikationsserver mittels einer API den Client mit geografischen Objekten und Daten der anderen Clients. Als Datengrundlage für Stationen dienen die durch Crowdsourcing erfassten Daten von OpenStreetMap. Neu ergänzte Stationen können zu einem späteren Zeitpunkt in OpenStreetMap zurückgeführt werden.

Abbildung 3 zeigt eine Übersicht der beteiligten Komponenten und deren Beziehungen.

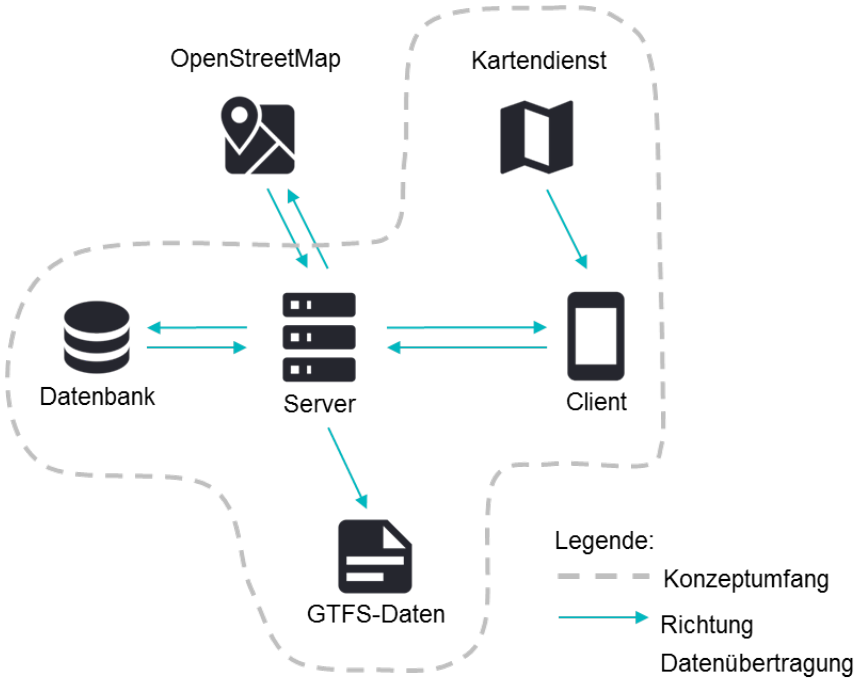


Abbildung 3: Systemaufbau

4.2 Client

Die Konzeption des Clients basiert auf den in Abschnitt 3 beschriebenen funktionalen und nicht-funktionalen Anforderungen. Hierbei soll eine mobile Anwendung entwickelt werden, mit der es möglich ist Fahrplandaten aus dem öffentlichen Personenverkehr zu erfassen. Dazu zählen die Möglichkeiten Daten direkt in einem Editor zu erstellen und zu bearbeiten, sowie Daten über einen Rekorder während der Fahrt aufzuzeichnen. Die erfassten Daten sollen auf einer Karte dargestellt werden und dem Nutzer als Informationsanzeige dienen. In einem Profil soll der Nutzer eine Übersicht über seine bisherigen Aktionen erhalten und Einstellungen vornehmen können, um die Anwendung seinen Bedürfnissen anzupassen. In den folgenden Abschnitten wird unter anderem die Zielplattform des Clients festgelegt, die einzelnen Komponenten näher beschrieben, die benötigten Frameworks evaluiert und die Datenstrukturen definiert.

4.2.1 Auswahl der Zielplattform

Durch die Auswahl einer geeigneten Zielplattform soll erreicht werden, dass der Client die größtmögliche Anzahl an Geräten, abhängig von den in Abschnitt 3.2 definierten nicht-funktionalen Anforderungen, unterstützt. Eine weitreichende Hardware-Kompatibilität ist notwendig, um die verschiedenen Nutzer in unterschiedlichen Regionen erreichen zu können. Basierend darauf wurde entschieden den Client als Android-Smartphone-App umzusetzen. Laut Statista besitzen in Deutschland 45,6 Millionen (Stand: 2015) Menschen, mit steigender Tendenz, ein Smartphone (vgl. Statista, 2015b). Smartphones erleichtern bereits den Alltag vieler Menschen. In Entwicklungsländern hat sich das Smartphone als ein unverzichtbares Gerät für Kommunikation,

Banking, Bildung und in vielen anderen Bereichen etabliert (vgl. CNN, 2012). Abbildung 4 und Abbildung 5 zeigen am Beispiel Afrikas, dass das Android-Betriebssystem von Google nicht nur in Entwicklungsländern das am meisten eingesetzte Betriebssystem auf Smartphones ist, sondern auch weltweit das weitverbreitetste mobile Betriebssystem.

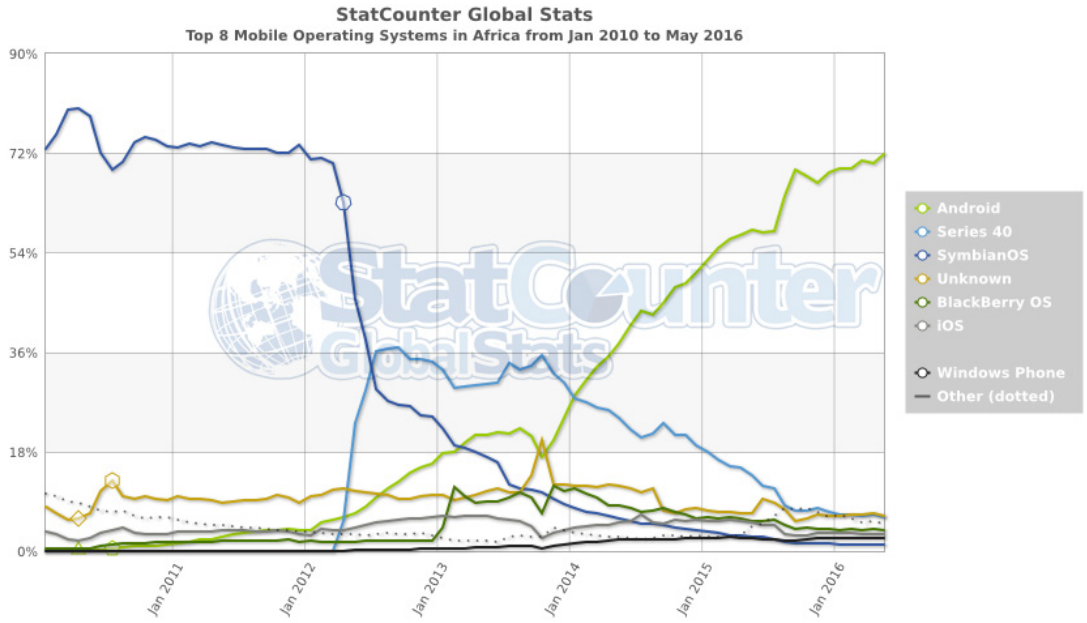


Abbildung 4: Top 8 Mobile Betriebssysteme in Afrika von Jan 2010 bis Mai 2016

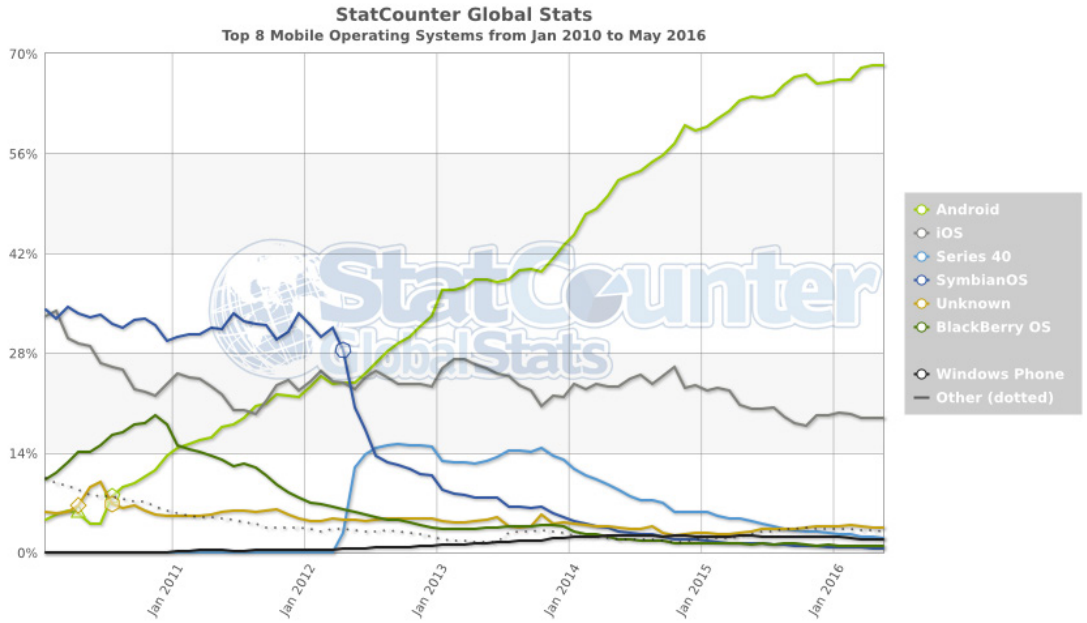


Abbildung 5: Top 8 Mobile Betriebssysteme Weltweit von Jan 2010 bis Mai 2016

Die Unterstützung weiterer Plattformen wäre zu einem späteren Zeitpunkt denkbar, um die Anzahl der potentiellen Nutzer zu erhöhen.

4.2.2 Komponenten

Der Client wurde in 5 Komponenten eingeteilt und jede Komponente besitzt jeweils eine eigene Funktion. Im Folgenden werden die einzelnen Komponenten erläutert und ihre Funktionen beschrieben.

4.2.2.1 Karte

Die *Karte* bildet als eine der wichtigsten Komponenten die Basis des Clients. Sie informiert den Nutzer über seine aktuelle Position und zeigt Stationen und Routen im gewählten Kartenbereich an. Durch Interaktion mit den auf der Karte dargestellten Objekten lassen sich verfügbare Detailinformationen anzeigen. Mit einem Fadenkreuz, das beim Anlegen einer neuen Station über der Karte eingeblendet wird, kann die genaue Position der Station festgelegt werden.

4.2.2.2 Editor

Für das Anlegen und Bearbeiten von Stationen und Routen ist der *Editor* zuständig. Hierbei erfolgt das Eingeben der Daten manuell und bietet, im Gegensatz zum *Rekorder* (siehe Abschnitt 4.2.2.3), zusätzlich die Möglichkeit detailliertere Informationen eines Objektes zu hinterlegen. Der Umfang der einzugebenden Daten beschränkt sich auf die Attribute des GTFS-Formates (siehe Abschnitt 2.3.1) und die Option ein Foto der Station zu hinterlegen.

4.2.2.3 Rekorder

Der *Rekorder* ist eine Komponente, welche die Eingabe von Daten für den Nutzer, im Vergleich zum Editor, deutlich erleichtert. Hierbei kann der Nutzer während des Reisens Fahrplandaten bequem, teilweise automatisch, erfassen. Dazu muss der Nutzer dem *Rekorder* zu Beginn Informationen bezüglich der Route und des Verkehrsmittels mitteilen. Im Anschluss benachrichtigt der Nutzer den *Rekorder* kontinuierlich, wenn das Verkehrsmittel an einer Station angekommen ist und benennt diese, falls sie sich nicht bereits in der Datenbank befindet. Daraus lassen sich neue Stationen anlegen, bestehende Stationen bestätigen und neue Routen erfassen. Da es bei diesem Verfahren vorkommen kann, dass nur eine Teilstrecke aufgenommen wird, werden diese Teilstrecken im Anschluss serverseitig zusammengesetzt beziehungsweise ergänzt, um daraus vollständige Routen zu erstellen. Wird das Aufzeichnen des Rekorders gestoppt, erhält der Nutzer eine Übersicht über die zurückgelegte Route und den erfassten Stationen. Sollte es während der Aufzeichnung zu Fehleingaben von Stationen gekommen sein, können diese hier vom Nutzer bearbeitet werden, bevor er die Daten bestätigt und sie bereit sind an den Server übertragen zu werden.

Neben den Pflichtinformationen, die für valide GTFS-Daten benötigt werden, können mit dem *Rekorder* zusätzlich einige optionale Informationen erfasst werden. Tabelle 5

zeigt eine Übersicht der Informationen, die während der Aufnahme durch den *Rekorder* aufgezeichnet werden können.

Tabelle 5: Erfassbare Informationen durch den Rekorder

Kategorie	**Information**	**Erforderlich**
Station	Name	Ja
	Standort	Ja
	Zeitzone	Nein
Route	Name / Nummer	Nein
	Richtung	Ja
	Transportmittel	Ja
	Ankunft- / Abfahrtszeiten	Ja
	Reisedauer zwischen Stationen	Nein
	Servicezeiten	Ja
	Streckenverlauf	Nein
	Streckendistanz	Nein

In Anhang G befindet sich ein Ablaufplan der einzelnen Schritte des *Rekorders*.

4.2.2.4 Profil

Jeder Nutzer muss beim erstmaligen Start des Clients einen Account erstellen und kann anschließend auf sein Profil zugreifen. Dort kann der Nutzer seine Profilinformationen einsehen und diese ändern. Im Profil erhält der Nutzer außerdem eine Übersicht über seine erfassten Stationen und Routen. Darüber hinaus werden Statistiken und weitere Elemente der Gamifizierung (siehe Abschnitt 4.2.3) im Profil hinterlegt.

4.2.2.5 Einstellungen

In den Einstellungen soll der Nutzer die Möglichkeit haben, den Client in ausgewählten Bereichen auf seine Bedürfnisse anzupassen. Dazu zählt zum Beispiel die Zeitzone des Clients einzustellen, offline gespeicherte Daten zu löschen und Spracheinstellungen vorzunehmen.

4.2.3 Gamifizierung

Der Begriff Gamifizierung beschreibt die Einbindung von spieltypischen Elementen in trivialen und monotonen Anwendungen, um diese interessant zu gestalten. Während bei Crowdsourcing die Gruppe hauptsächlich aus Freiwilligen besteht, die durch das

Sammeln von Daten einen direkten Mehrwert für sich selbst oder Anerkennung erhalten, soll durch Gamifizierung ein zusätzlicher Anreiz geschaffen und gleichzeitig die Einbindung in die Community gefördert werden. Das ist nötig, um auch weitere Nutzergruppen anzusprechen und zu motivieren. Über ein Punktesystem beim Erfassen von Stationen und Routen soll der Nutzer mit anderen Nutzern konkurrieren können. Dieses Punktesystem kann beliebig zum Beispiel durch Abzeichen oder Titel erweitert werden.

4.2.4 Auswahl des Kartendienstes

Für den Client wird ein Kartendienst benötigt, um Stationen und Routen auf einer Karte darzustellen. Der Kartendienst muss für den Einsatz in der App verschiedene Kriterien erfüllen, die aus den definierten Anforderungen aus Abschnitt 3 hervorgehen.

4.2.4.1 Vergleich der Kartendienste

Tabelle 6 nennt Kriterien, die festgelegt wurden, um die einzelnen Kartendienste miteinander zu vergleichen und anhand eines Punktesystems zu bewerten. Sollte ein Dienst nicht die in Abschnitt 4.2.1 definierte Zielplattform Android unterstützen ist dies ein Ausschlusskriterium und der Dienst kann selbst bei hoher Punktezahl nicht berücksichtigt werden.

Tabelle 6: Kriterien für Vergleich von Kartendiensten

Kriterium	Punktevergabe	Begründung
Offline-Modus verfügbar	Ja = 1 Punkt Nein = 0 Punkte	Die Funktionalität muss nach den definierten funktionalen Anforderungen (siehe Abschnitt 3.2) auch im Offline-Modus gewährleistet sein.
Kachelformat	Vektor = 2 Punkte Raster = 1 Punkt	Vektorkacheln haben gegenüber Rasterkacheln einige Vorteile (siehe Abschnitt 2.1)
Plattform-Support	Jede Plattform = 1 Punkt Maximal: 3 Punkte	Um später weitere Plattformen unterstützen zu können, soll ein Dienst, der viele Plattformen unterstützt, bevorzugt werden.
Karten Formatierung	Ja = 1 Punkt Nein = 0 Punkte	Individuelles Design für die Karte, um diese an die Farbgebung und dem Layout des Clients anpassen zu können.
Offene Daten	Ja = 1 Punkt Nein = 0 Punkte	Der Zweck der Applikation ist es Daten zu sammeln und als offene

		Daten der Öffentlichkeit zugänglich zu machen. Deshalb sollen Unterstützer und Dienste, die diese benutzen, bevorzugt werden.
Kommerziell	Nein = 1 Punkt Ja = 0 Punkte	Kostenfreie Dienste sollen gegenüber kommerziellen Diensten bevorzugt werden, da die Applikation selbst keinen direkten Gewinn generiert und um während der Entwicklung zusätzliche Kosten vermeidet.

Die Tabelle mit dem Vergleich und der Bewertung der Kartendienste befindet sich in Anhang B.

4.2.4.2 Entscheidung

Die Auswertung der Kartendienste in Anhang B hat ergeben, dass Mapbox die höchste Punktzahl erreicht und alle Anforderungen erfüllt. Darüber hinaus bietet Mapbox weitere Funktionen und Eigenschaften wie zum Beispiel Reverse-Geocoding, welches anhand einer Koordinate eine Adresse zurückliefert. Mapbox ist ein kommerzieller Anbieter, der zusätzlich eine kostenfreie Lizenz mit Anfragenlimit anbietet. Diese Lizenz ist für die Entwicklung eines Prototyps ausreichend. Für eine Veröffentlichung besteht die Möglichkeit, zu einer kommerziellen Lizenz zu wechseln.

4.2.5 Offline-Verfügbarkeit

In den funktionalen Anforderungen (siehe Abschnitt 3.2) wurde festgelegt, dass der Client auch ohne aktive Internetverbindung offline funktionsfähig sein soll. Dies beinhaltet die Offline-Verfügbarkeit eines Kartenausschnitts, sowie die in diesem Kartenausschnitt befindlichen Stationen und Routeninformationen.

4.2.5.1 Kartenmaterial

Durch die Bewertung von Kartendiensten (siehe Abschnitt 4.2.4.1) wurde Mapbox als geeigneter Kartendienst ausgewählt. Mapbox unterstützt in seinem SDK das Herunterladen und Verwenden von Kartenausschnitten. Der zu wählende Kartenausschnitt unterliegt allerdings Beschränkungen. Bei Mapbox ist es nur möglich Ausschnitte mit maximal 6000 Kacheln herunterzuladen. Die Kachelanzahl für ein Gebiet steht in Abhängigkeit mit der Vergrößerungsstufe. Der Download beginnt bei der niedrigsten gewählten Vergrößerungsstufe und wird bei jedem Durchgang um 1 erhöht, bis die höchste gewählte Vergrößerungsstufe erreicht ist. Wird zuvor das Limit von 6000 Kacheln erreicht, wird der Download vorzeitig beendet und die Karte ist unvollständig. Daher muss eine geeignete Vergrößerungsstufe festgelegt werden, ab welcher das Herunterladen von Kartenausschnitten zugelassen wird, um einen vollständigen Kartenaus-

schnitt zu gewährleisten. Mapbox bietet dazu eine Webanwendung *Offline-Estimator*[2] zur Schätzung der Kachelanzahl eines ausgewählten Kartenausschnittes an.

Anhang C zeigt einen beispielhaften Kartenausschnitt von Stuttgart bei Vergrößerungsstufe 10 und einer Begrenzung von 6000 Kacheln.

Die Abschätzungen der Kachelanzahl der Anwendung *Offline-Estimator* variieren bei unterschiedlichen Kartenbereichen bei Vergrößerungsstufe 10 um ± 400 Kacheln (siehe Anhang D). Um trotz Toleranz einen vollständigen Kartenausschnitt zu gewährleisten, wird die Mindest-Vergrößerungsstufe für den Offline-Modus auf 11 festgelegt.

Voraussetzung für den Offline-Modus ist, dass ausreichend Speicherplatz auf dem Gerät zur Verfügung steht. Die Größe der Daten ist laut Mapbox (2016b) von verschiedenen Faktoren abhängig:

- Geografische Ausdehnung der Region
- Vergrößerungsstufe
- Merkmalanzahl und Dichte
- Kartenstil, Schriftarten und verwendete Icons
- Größe individueller Kacheln der Region
- Region-Überschneidungen

Tabelle 7 zeigt Beispiele von Mapbox, um zu verdeutlichen, in welchem Bereich sich der Speicherbedarf befindet:

Tabelle 7: Mapbox Beispiele für Speicherbedarf von Offline-Karten

Gebiet	**Standard Mapbox Straßenkartenstil**	**Mapbox Satellitenkarte**
Barcelona	83 MB	45 MB
Großraum London mit M25 Autobahn Vergrößerungsstufe 0-15	120 MB	400 MB
USA Vergrößerungsstufe 0-9	290 MB	315 MB

Um die Angaben zum Speicherbedarf von Mapbox zu validieren und den Speicherbedarf bei Vergrößerungsstufe 11 zu ermitteln, wurde ein Kartenausschnitt von Stuttgart über die Mapbox-API heruntergeladen. Abbildung 6 zeigt einen Screenshot des gewählten Kartenausschnitts im Standard Mapbox-Straßenkartenstil.

[2] Offline-Estimator Webseite: https://www.mapbox.com/labs/offline-estimator/ (Datum des Zugriffs: 23. Juli 2016)

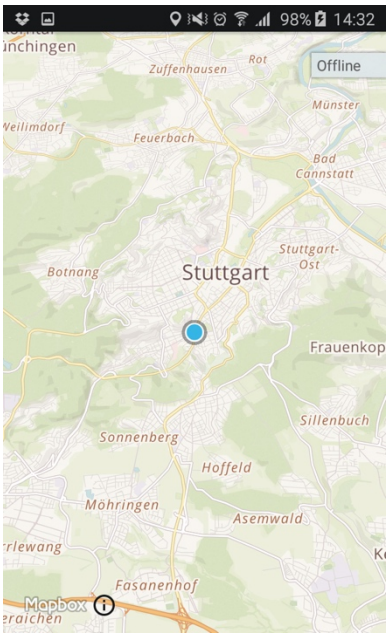


Abbildung 6: Screenshot eines Kartenausschnitts von Stuttgart (Standard Mapbox Straßenkartenstil)

Die Auswertung des Speicherbedarfes ergab, dass die Größe der heruntergeladenen Daten 88,64 Megabyte beträgt (siehe Abbildung 7). Damit entspricht der Speicherbedarf den von Mapbox angegeben Beispielen.

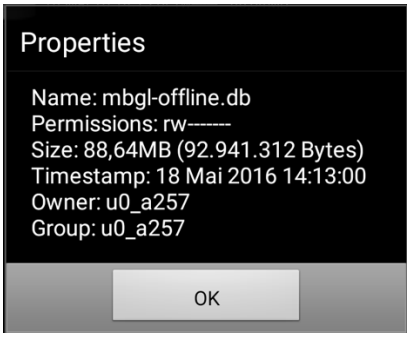


Abbildung 7: Dateieigenschaften der Offline-Daten von Mapbox

4.2.5.2 Stationen und Routeninformationen

Im Offline-Modus müssen neben dem Kartenausschnitt ebenfalls Stationen und Routen offline verfügbar sein. Wird der Client in den Offline-Modus versetzt, werden zuvor die entsprechenden Datensätze vom Server heruntergeladen und auf dem Gerät gespeichert.

4.2.6 Datenhaltung

Die gesammelten Daten der Nutzer müssen auf dem Gerät gespeichert werden, da eine Internetverbindung nicht immer gewährleistet ist. Neben der Karte müssen auch

bereits bestehende Stationen und Routen eines Gebietes offline verfügbar gemacht werden können. Zusammen mit den Daten, die der Nutzer erfasst hat, müssen diese auf dem Gerät gespeichert werden. Android Developers nennt hierbei 5 verschiedene Möglichkeiten der Datenhaltung (vgl. 2016a).

- Shared Preferences (Key-Value)
- Interner Speicher
- Externer Speicher (SD-Karte)
- Datenbank
- Netzwerkverbindung

4.2.6.1 Fahrplandaten

Shared Preferences wird zur Speicherung von Key-Value-Paaren eingesetzt und eignet sich daher nur für das Speichern einzelner Attribute und Nutzerkonfigurationen. Interner und externer Speicher bieten die Möglichkeit Daten in Form von Dateien auf dem Gerät zu speichern. Die Daten über eine Netzwerkverbindung auf einem Server zu speichern kann als Möglichkeit ausgeschlossen werden, da eine durchgängige Internetverbindung nicht gewährleistet ist. Eine von Android empfohlene SQLite-Datenbank ist gegenüber normalen Dateien deutlich performanter und kann Relationen zwischen Datensätzen abbilden. Das GTFS-Format ist dem Aufbau einer relationalen Datenbank sehr ähnlich, weshalb eine SQLite-Datenbank zum Speichern der Fahrplandaten am geeignetsten ist.

Das GTFS-Format ist in 13 verschiedene Dateien aufgeteilt, welche über ID-Attribute verknüpft sind. In SQLite können die einzelnen Dateien mit ihren Attributen in jeweils eine Tabelle übertragen und mittels Primär- und Fremdschlüssel die Beziehung der Tabellen untereinander optimal abgebildet werden (siehe ER-Modell in Anhang A).

Um für die Synchronisierung veränderte und neue Datensätze erkennen zu können, müssen die im GTFS-Format definierten Attribute in der Datenbank durch zusätzliche Attribute ergänzt werden.

Tabelle 8: Zusätzliche Attribute in der Datenbankstruktur

Tabelle	**Datentyp**	**Attribute**	**Verwendung**
Alle GTFS-Tabellen	Long Boolean Boolean	Timestamp Temp Deleted	Synchronisierung Temporäre Daten Löschindikator

Timestamp: Für die Synchronisierung zwischen Client und Server wird ein Zeitstempel benötigt. Wurde ein Datensatz vom Nutzer geändert oder neu hinzugefügt, wird bei diesem Datensatz der Zeitstempel aktualisiert bzw. gesetzt. Sollten die Serverdaten aktueller sein als die auf dem Client, haben die Serverdaten Priorität und die Clientdaten werden verworfen (siehe Abschnitt 4.3.2).

Temp: Dieses boolesche Attribut dient zur Unterscheidung von Daten abgeschlossener Aufzeichnungen und Daten die zum Beispiel durch den *Rekorder* derzeit aufgezeichnet werden. Temporäre Daten benötigen die explizite Bestätigung durch den Nutzer, um gültige permanente Daten zu werden und synchronisiert werden zu können.

True: Datensatz ist temporär
False: Datensatz ist permanent

Deleted: Wird ein Datensatz vom Nutzer gelöscht, wird bei diesem Datensatz das Attribut *Deleted* gesetzt. Dadurch kann der Datensatz beim Synchronisierungsprozess entsprechend verarbeitet werden (siehe Abschnitt 4.3.2).

True: Datensatz wurde als gelöscht markiert
False: Datensatz ist gültig

4.2.6.2 Profil- und Einstellungsdaten

Bei den Profildaten und Einstellungen handelt es sich ausschließlich um Key-Value Paar-Informationen. Diese werden deshalb in den *Shared Preferences* der Android-App gespeichert.

4.2.6.3 Offline-Kartenmaterial

Das Kartenmaterial wird vom Mapbox SDK auf dem internen Speicher des Gerätes in einer Datenbank-Datei abgelegt. Nur die eigene Anwendung hat Zugriff auf diesen Speicherbereich. Bei Deinstallation der App werden alle zugehörigen Dateien der App gelöscht (vgl. Android Developers, 2016a).

4.2.7 Client-Server-Synchronisierung

Der Client kommuniziert in verschiedenen Situationen über eine API (siehe Abschnitt 4.3.1) mit dem Applikationsserver. Hierbei wird zwischen 3 Anwendungsfällen unterschieden:

4.2.7.1 On-The-Fly

On-The-Fly beschreibt das Nachladen von Stationen des Applikationsservers, mit dem Zweck diese auf der Karte des Clients anzuzeigen. Auf der Karte werden Stationen des aktuellen Kartenausschnitts dargestellt. Sobald der Nutzer mit der Karte interagiert und der Kartenausschnitt sich ändert, kann es nötig sein, Stationen zu diesem Gebiet nachzuladen, sofern diese nicht bereits auf dem Gerät vorhanden sind. Stationen werden erst ab einer Vergrößerungsstufe von 9 vom Server angefordert, um den Datenverbrauch beim Übertragen gering zu halten und damit die Karte auch in niedrigeren Vergrößerungsstufen übersichtlich bleibt.

4.2.7.2 Sync-Back

Um die vom Nutzer erfassten Daten im GTFS-Format zu veröffentlichen und anderen Nutzern des Clients zugänglich machen zu können, müssen die Daten an einen Appli-

kationsserver übertragen werden. Dazu werden die neu erfassten und geänderten Daten über eine API an den Applikationsserver gesendet.

In Abschnitt 4.3.2 werden die verschiedenen Fälle, die beim Synchronisieren eintreten können, gegenübergestellt und definiert wie diese server-/clientseitig verarbeitet werden.

4.2.7.3 Offline-Mode

Ein Großteil der Datensätze kann bereits durch den *On-The-Fly* Anwendungsfall auf dem Gerät vorhanden sein. Neue Datensätze und Änderungen werden vor Aktivierung des Offline-Modus beim Server angefragt und für die Offline-Nutzung auf dem Gerät gespeichert.

4.2.8 Designkonzept

Das Designkonzept und die damit verbundenen Mockups wurde parallel zu dieser Bachelorarbeit von Jan Asmus und Raphael Dirr, Studenten der Hochschule für Gestaltung in Schwäbisch Gmünd, bei der moovel Group GmbH erstellt. Aufgrund dessen wird auf die Erstellung weiterer Mockups verzichtet und das vorhandene Designkonzept zur Realisierung verwendet.

4.3 Applikationsserver

Der Applikationsserver stellt Schnittstellen bereit, über die der Client mit dem Applikationsserver kommunizieren und Daten austauschen kann. Hierzu sollen die Daten vom Applikationsserver in einer Datenbank verwaltet und bei Bedarf im GTFS-Format bereitgesellt werden können. In den folgenden Abschnitten werden die zu realisierenden Serverkomponenten im Detail spezifiziert und konzipiert.

4.3.1 Definition der Schnittstellen

In Bezug auf die Anforderungen (siehe Abschnitt 3.2) werden auf dem Applikationsserver Schnittstellen entsprechend ihrer Funktionalität definiert. Hierzu zählen Schnittstellen für die Kommunikation mit dem Client und die Bereitstellung der bereits erfassten GTFS-Daten. Angelehnt an die im Unternehmen eingesetzten Technologien werden die Schnittstellen auf Basis der REST-Architektur mit dem HTTP-Protokoll definiert.

Bei einer REST-API können Client und Server unabhängig entwickelt werden, sofern die Kommunikation über das *Uniform Interface* unverändert bleibt. Uniform Interface beschreibt Einschränkungen, die bei der Kommunikation von Webkomponenten eingehalten werden müssen. Für den Einsatz von HTTP müssen daher die HTTP-Methoden GET, POST, PUT und DELETE verwendet werden. Die REST API ist standardisiert und daher mit weiteren Plattformen kompatibel. Des Weiteren bietet sie durch Caching und Uniform Interface hohe Zuverlässigkeit und Skalierbarkeit. Die Skalierbarkeit wird durch die Zustandslosigkeit (*stateless*) des Servers unterstützt. Hierdurch muss jeder Schnittstellenzugriff die gesamten Kontextinformationen der Interaktion enthalten (vgl. Massé, S. 3-4, 2012).

Für die Übermittelung der Daten wird JSON als Übertragungsdatenformat eingesetzt. JSON ist sehr leichtgewichtig in Bezug auf Verarbeitungszeit und Einsparung von Datenkapazität bei der Kommunikation. Zudem ist das Format für Mensch und Maschine gleichermaßen einfach zu lesen und zu schreiben. JSON ist außerdem programmiersprachen-unabhängig, aber folgt zugleich vielen Konventionen und bietet daher eine hohe Kompatibilität (vgl. JSON, 2016).

4.3.1.1 On-The-Fly Schnittstelle

Im Anwendungsfall *On-The-Fly* werden nur Daten übertragen, die unmittelbar dem Nutzer auf der Karte angezeigt werden. Hierbei handelt es sich ausschließlich um Stationsattribute, welche im GTFS-Format unter der Tabelle „stops“ definiert sind (siehe Anhang A). Für die Anfrage wird eine Bounding Box benötigt, welche die aktuelle Kartenprojektion auf dem Gerät repräsentiert. Eine Bounding Box ist ein rechteckiger Bereich, der durch zwei geografische Koordinaten, welche aus jeweils einem Breitengrad (Latitude) und einem Längengrad (Longitude) bestehen, definiert wird (vgl. Bounding Box, 2015). Die Koordinaten dieser Bounding Box werden benötigt um Stati-

onen innerhalb dieses Gebietes aus der Datenbank zu ermitteln und als Rückgabewert zurück an den Client geben zu können.

Tabelle 9: Schnittstellendefinition zur Bereitstellung von Stationsdaten eines Gebietes

Funktion	Liefert Stationsdaten eines Gebietes
Methode	GET
Pfad	/transit/stations/
Parameter	Latitude1,Longitude1;Latitude2,Longitude2
Rückgabetyp	JSON
Beispiel	GET /transit/stations/48.7746675,9.1372266;48.7581915,9.1662271

In Anhang H befindet sich eine Beispielantwort des Servers.

4.3.1.2 Offline-Mode Schnittstelle

Im *Offline-Mode* (siehe Abschnitt 4.2.7.3) werden vollständige Stations- und Routendaten durch den Client angefordert. Diese werden für die Offline-Nutzung auf dem Gerät verfügbar gemacht. Hierbei können nach Region größere Datenmengen im zweistelligen Megabyte-Bereich anfallen. Wie im *On-The-Fly*-Anwendungsfall beschrieben, wird im *Offline-Mode* ebenfalls eine Bounding Box benötigt, welche das umfassende Gebiet eingrenzt und Standortinformationen enthält.

Tabelle 10: Schnittstellendefinition zur Bereitstellung von Stations- und Routendaten

Funktion	Liefert Station- und Routendaten eines Gebietes
Methode	GET
Pfad	/transit
Parameter	Latitude1,Longitude1;Latitude2,Longitude2
Rückgabetyp	JSON
Beispiel	GET /transit/48.7746675,9.1372266;48.7581915,9.1662271

In Anhang H befindet sich eine Beispielantwort des Servers.

4.3.1.3 Sync-Back Schnittstelle

Während der Datensynchronisierung zwischen Client und Server werden zuerst die geänderten und neu erfassten Daten des Clients im JSON-Format an den Server gesendet. Daraufhin werden die Daten vom Server verarbeitet (siehe Abschnitt 4.3.2) und dieser sendet sie zusammen mit Datensätze, die sich seit der letzten Synchronisierung

geändert haben, zurück an den Client. Der Client übernimmt die Datenaktualisierung in den vorhandenen Datenbestand.

Tabelle 11: Schnittstellendefinition der Client-Server-Synchronisierung

Funktion	Client-Server-Synchronisierung
Methode	POST
Pfad	/transit
Übergabetyp	JSON
Rückgabetyp	JSON
Beispiel	POST /transit

In Anhang H befindet sich eine Beispielanfrage und eine Beispielantwort des Servers.

4.3.1.4 Station Details Schnittstelle

Während bei *On-The-Fly* nur Stationsdaten vom Server abgefragt werden, müssen in der Detailansicht der Station ebenfalls dazugehörige Informationen, wie zum Beispiel Routen und Abfahrtszeiten, dargestellt werden. Dazu wird die eindeutige ID der Station als Parameter der Anfrage angegeben um die Station zu identifizieren.

Tabelle 12: Schnittstellendefinition zur Bereitstellung von detaillierten Stationsdaten

Funktion	Liefert Detailinformationen über eine angegebene Station
Methode	GET
Pfad	/transit/station
Parameter	ID
Rückgabetyp	JSON
Beispiel	GET /transit/station/123

Siehe Anhang H für eine Beispielantwort des Servers.

4.3.1.5 GTFS-Daten Schnittstelle

Für den Zugriff auf die GTFS-Daten gibt es verschiedene Strategien. Zu Beginn ist es möglich die GTFS-Daten gesamt auf Anfrage zu generieren und bereitzustellen. Bei globaler Nutzung jedoch wächst der Datenbestand kontinuierlich und es wäre nötig bei einer Anfrage ein entsprechendes Gebiet als Parameter zu übergeben, um die Daten einzugrenzen. Im nächsten Schritt wäre eine tägliche Generierung der GTFS-Daten, aufgeteilt nach Ländern oder Städten, möglich und diese anschließend lokal abzule-

gen. Bei einer Anfrage wäre es folglich nicht mehr nötig die Daten zuvor zu generieren. Dadurch wäre es ebenfalls möglich die Dateien zu cachen, um bei einer steigenden Anzahl an Anfragen performant antworten zu können.

Im Rahmen der Konzeption wird nur der einfachste Fall betrachtet und als Schnittstelle definiert.

Tabelle 13: Schnittstellendefinition zur Bereitstellung von Serverdaten im GTFS-Format

Funktion	Generiert GTFS-Dateien aus Serverdaten und bietet diese als ZIP-Archiv zum Download an
Methode	GET
Pfad	/gtfs/download
Rückgabetyp	ZIP
Beispiel	GET /gtfs/download

In Anhang H befindet sich ein Beispiel eines ZIP-Archivs mit GTFS-Daten des Servers.

4.3.1.6 Komprimierung

Betrachtet man die Anzahl der Daten die erfasst werden, kann die Größe der Daten beim Datenaustausch zwischen Client und Server mehrere Megabyte erreichen. Dies wird deutlich bei der Betrachtung bereits existierender GTFS-Daten von Verkehrsbetrieben. Als Beispiel umfassen die GTFS-Daten des Verkehrsverbunds Berlin-Brandenburg 28,1 Megabyte (vgl. VBB, 2015). Um den Datenverbrauch bei der Kommunikation weiter zu verringern, können die JSON-Daten vor der Übertragung über ein Komprimierungsverfahren, wie zum Beispiel Gzip[3] nochmals komprimiert werden.

4.3.1.7 Status-Codes

Die REST API nutzt HTTP-Status-Codes, um den Client über das Ergebnis einer Anfrage zu informieren. So bestehen Status-Codes aus einer dreistelligen Nummer, deren erste Ziffer das Ergebnis in 5 Kategorien einteilt.

Tabelle 14: HTTP-Status-Code Kategorien

1xx	Informationen
2xx	Erfolgreiche Operation
3xx	Umleitung
4xx	Client-Fehler

[3] Gzip Webseite - http://www.gzip.org/ (Datum des Zugriffs: 21. Juli 2016)

5xx	Server-Fehler

Die zweite und dritte Ziffer ist zusammen mit der Bedeutung frei wählbar. Dennoch gibt es Status-Codes die sich als Standard etabliert haben. Dazu zählt zum Beispiel der Status-Code „404“, der häufig als Fehlermeldung auf Webseiten angezeigt wird, wenn eine Seite nicht gefunden wurde.

Tabelle 15 zeigt HTTP Status-Codes, welche für die REST API des Servers definiert wurden.

Tabelle 15: HTTP Status-Codes der REST API des Servers

200	Anfrage erfolgreich
400	Fehlerhafte Syntax der Anfrage
401	Authentifizierungsfehler
404	Keine Schnittstelle definiert
500	Fehler auf Serverseite – Anfrage kann nicht bearbeitet werden

4.3.2 Verarbeitung Clientdaten

Die folgenden Fallunterscheidungen beziehen sich auf den Anwendungsfall *Sync-Back* (siehe Abschnitt 4.2.7.2).

Tabelle 16: Fallunterscheidung bei der Verarbeitung von Clientdaten

Fall	**Server- / clientseitige Verarbeitung**
Neue Datensätze vom Client	Server speichert neue Datensätze und aktualisiert Punktezahl des Nutzers
Datensätze wurden auf dem Client geändert	Die Änderungen werden auf dem Server übernommen, sofern die Daten nicht zuvor von einem anderen Nutzer geändert wurden. In diesem Fall werden die Änderungen des Clients verworfen.
Datensätze wurden auf dem Client als gelöscht markiert	
Datensätze wurden auf dem Client geändert und auf dem Server gelöscht	Die Änderungen des Clients werden verworfen und gelöscht
Datensätze, die ebenfalls auf dem Client vorhanden sind, wurden auf dem Server geändert oder gelöscht	Client ändert bzw. löscht Datensätze

4.3.3 Datenhaltung

Die Datensätze werden serverseitig, wie bereits im Client, in relationaler Form persistent gespeichert. Die serverseitige Datenhaltung muss gegenüber der Datenhaltung des Clients weitere Anforderungen erfüllen. Die Datenbank verwaltet die Daten und Anfragen aller Clients und muss daher eine Großzahl an Anfragen verarbeiten. Durch den weltweiten Einsatz muss die Datenbank zusätzlich mit großen Datenmengen umgehen können und skalierbar sein. Aufgrund der gegeben Anforderungen eignet sich hierfür eine Datenbank auf SQL-Basis.

Für das Datenbankschema wird das gleiche ER-Modell (siehe Anhang A) wie für den Client verwendet. Parallel zur Client Datenhaltung müssen alle Tabellen des Schemas ebenfalls durch ein zusätzliches Attribut *timestamp* erweitert werden, um während der Synchronisierung ausschließlich die Übertragung von geänderten und neuen Datensätzen sicherzustellen.

4.3.4 Datenvalidierung und Missbrauch

Für die Datenvalidierung wird eine Kontrollinstanz benötigt, welche regelmäßig die von den Nutzern erfassten Daten überprüft. Sollten dennoch nicht valide Daten im System auftauchen haben Nutzer die Möglichkeit diese explizit zu melden, damit eine erneute Kontrolle stattfindet. Diese Kontrollinstanz ist besonders wichtig für die Rückführung von Stationsdaten in OpenStreetMap. Um Daten in OpenStreetMap zu ändern oder neu hinzuzufügen wird ein Account benötigt. Jede Datenübermittelung an OSM wird über diesen Account getätigt. Folglich sollten nicht valide Daten vermieden werden, da es sonst zu einer Sperrung des Accounts kommen kann. Um einen Missbrauch und nicht valide Daten weitestgehend zu vermeiden ist der Client ebenfalls nur durch einen angelegten Account nutzbar. Bei vorsätzlicher Eingabe von falschen Daten kann die Kontrollinstanz einzelne Accounts sperren und somit von der Community ausschließen.

5 Realisierung

Zusammen mit dem Betreuer wurde vereinbart, dass die Realisierung auf den *Rekorder* und überschneidende Funktionen beschränkt wird (vgl. Hecker, 2016). Dazu zählen notwendige Funktionen wie die Offline-Verfügbarkeit, die Client-Server-Kommunikation, sowie die Visualisierung von Fahrplandaten auf der Karte des Client. Dadurch ist es bereits möglich einfach Fahrplandaten zu erfassen, diese zu visualisieren und an den Server zu übertragen. Andere Clients wären außerdem in der Lage die an den Server übertragenden Daten zu empfangen und darzustellen. Durch die Implementierung dieser Funktionen ist ein funktionierender Kreislauf der Daten gegeben.

5.1 Client-Projektstruktur

Die Struktur einer Android-App wird bereits von Google sehr genau definiert. Hierbei werden die verschiedenen Dateitypen einem eigenen Verzeichnis zugeordnet.

So gibt es ein Verzeichnis *res* mit Unterverzeichnissen für Nicht-Code-Dateien, XML-Layouts, UI-Strings und Grafikdateien. Das *Manifests*-Verzeichnis beinhaltet essentielle Informationen, wie zum Beispiel Berechtigungen und App-Struktur-Informationen. Im Verzeichnis *java* werden alle selbstgeschriebenen Code-Dateien der App abgelegt (vgl. Android Developer, 2016). Um zu vermeiden, dass der Bereich unübersichtlich wird, werden die Code-Dateien in entsprechenden Unterverzeichnissen zusammengefasst.

In Anhang E befindet sich eine Übersicht über die Verzeichnisstruktur der Client-App.

5.2 Client-Datenhaltung

In der Konzeption der Datenhaltung des Clients wurde eine für diesen Zweck bei Android übliche SQLite-Datenbank gewählt. Für die Realisierung wurde allerdings eine *Realm*-Datenbank verwendet. Während der Konzeption wurden ausschließlich die von Android empfohlenen Speichermöglichkeiten eruiert. Die Realm-Datenbank ähnelt einer SQLite-Datenbank in vielen Bereichen und bietet zusätzlich weitere Vorteile. Die Realm-Datenbank besteht ebenfalls nur aus einer Datei, welche im internen Speicher der App angelegt wird. Des Weiteren unterstützt *Realm* ebenfalls verschiedene Plattformen und ermöglicht damit die gleichen Strukturen auf allen Plattformen zu verwenden. Ein für die Entwicklung wichtiger Vorteil ist das integrierte Object-realtional mapping (OSM). Damit lässt sich eine direkte Verbindung zwischen Datenbank-Tabellen und Objekten herstellen. Dies macht die Datenbankverwaltung sehr flexibel, einfacher und deutlich schneller, da der Entwickler sich nicht zusätzlich um die Datenbankstruktur kümmern muss. Abbildung 8 zeigt, dass Realm zusätzlich im Vergleich zu anderen mobilen Datenbanken wie zum Beispiel SQLite, FMDB und Core Data im Bereich von Abfragen, Zählen und Schreibzugriffen deutlich schneller ist (vgl. realm.io, 2016).

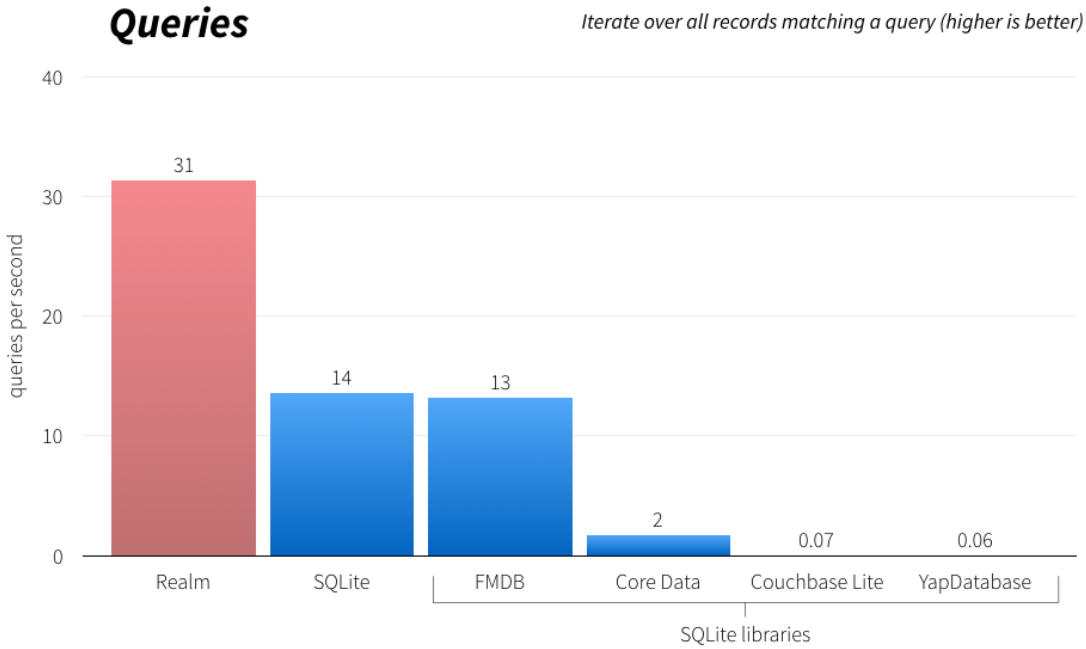


Abbildung 8: Geschwindigkeitsvergleich mobiler Datenbanken im Bereich Abfragen (Realm.io, 2014)

5.3 Offline-Mode

Die folgenden Abschnitte beschreiben die Realisierung der in Abschnitt 4.2.5 definierten Offline-Verfügbarkeit des Clients.

5.3.1 Kartenausschnitt

Um einen Kartenausschnitt offline verfügbar zu machen, bietet das eingesetzte SDK des Kartendienstes Mapbox bereits eine entsprechende Klasse *OfflineManager* an. Hierzu muss diese mit einem Access Token initialisiert werden, der von Mapbox bereitgestellt wird und die Applikation gegenüber den Mapbox-Servern identifiziert. Die Methode *createOfflineRegion* initiiert den Download der Kartendaten und persistiert diese auf dem Gerät. Die Methode besitzt 3 Parameter, die beim Aufruf der Methode übergeben werden müssen.

Quellcode 1: OfflineManager Initialisierung

```
// Set up the OfflineManager
OfflineManager offlineManager = OfflineManager.getInstance(context);
offlineManager.setAccessToken(context.getString(R.string.accessToken));

// Create the region asynchronouly
offlineManager.createOfflineRegion(definition, metadata,
new OfflineManager.CreateOfflineRegionCallback() {

    @Override
    public void onCreate(OfflineRegion offlineRegion) { ... }
```

```
    @Override
    public void onError(String error) { ... }
    }
});
```

Der erste Parameter ist ein Definitions-Objekt, welches die nötigen Informationen für den Mapbox-Server beinhaltet. Dazu gehören der gewünschte Kartenstil, eine Bounding Box, welche den Kartenausschnitt geographisch beschreibt, sowie Minimum- und Maximum-Werte der Vergrößerungsstufe. Die Bounding Box des Kartenausschnitts, die dem Nutzer auf seinem Display dargestellt wird, kann über den *MapController* der Mapbox-Karte direkt ausgelesen werden. Die entsprechenden Minimum- und Maximum-Werte der Vergrößerungsstufen wurden in Abschnitt 4.2.5.1 auf mindestens 11 und maximal 16 festgelegt. Der letzte Parameter des Definition-Objekts beschreibt die Pixeldichte des Gerätedisplays, um die Daten vor dem Übertragen an die Eigenschaften des Displays anzupassen.

Quellcode 2: Definitions-Objekt des Kartenausschnitts

```
// Define the offline region
OfflineTilePyramidRegionDefinition definition =
    new OfflineTilePyramidRegionDefinition(
        mapView.getStyleUrl(),
        latLngBounds,
        minzoom,
        maxzoom,
        context.getResources().getDisplayMetrics().density);
```

Der zweite Parameter besteht aus Metadaten im JSON-Format. Die Metadaten beziehen sich nach aktuellem Stand ausschließlich auf die Definition eines Bezeichners für die gewählte Region.

Quellcode 3: Metadaten des Kartenausschnitts

```
// Set the metadata
byte[] metadata;
try {
    JSONObject jsonObject = new JSONObject();
    jsonObject.put(JSON_FIELD_REGION_NAME, "default");
    String json = jsonObject.toString();
    metadata = json.getBytes(JSON_CHARSET);
} catch (Exception e) {
    Log.e(MainActivity.TAG, "Failed to encode metadata: " + e.getMessage());
    metadata = null;
}
```

Im letzten Parameter (siehe Quellcode 1) ist es möglich eine Callback-Instanz von *CreateOfflineRegionCallback* zu übergeben. Dieses Callback-Objekt besitzt eine über-

schreibbare *onCreate*- und *onError*-Methode. Mapbox bietet die Möglichkeit in *onCreate* ein Observer-Objekt *OfflineRegionObserver* zu registrieren, worüber der Fortschritt überwacht und dem Nutzer in Form eines Fortschrittbalkens visualisiert werden kann.

5.3.2 Fahrplandaten

Um dem Nutzer auch im Offline-Modus den Zugriff auf Detailinformationen von Stationen zu ermöglichen, werden im Anschluss an den Download des Kartenausschnittes Detailinformationen der Station im ausgewählten Bereich vom Server angefordert. Diese Daten werden über eine REST API des Servers dem Client im JSON-Format bereitgestellt (Siehe Abschnitt 4.3.1). Um die Daten in der App verwenden zu können, müssen diese zuerst geparst und anschließend in der Datenbank persistiert werden.

Quellcode 4: JSON-Daten des Servers parsen und persistieren

```
public void readJSON(String data)

       // Parse json data to json object
       JsonObject jsondata = parser.parse(data)
                  .getAsJsonObject().get("data").getAsJsonObject();

       String[] tables = new String[8];
       Class[] classes = { Agency.class, Calendar.class, ... };

       if(jsondata.has("agencies"))
            tables[0] = jsondata.getAsJsonArray("agencies").toString();
       if(jsondata.has("calendars"))
            tables[1] = jsondata.getAsJsonArray("calendars").toString();
       ...

       JSONArray array;

       for(int i = 0; i < tables.length; i++) {

            ...

            array = new JSONArray(tables[i]);

            // Persistieren
            Realm realm = Realm.getDefaultInstance();
            realm.createOrUpdateAllFromJson(classes[i], array);

       }
}
```

5.4 Datenvisualisierung

Die Visualisierung von Daten beziehungsweise die Informationsanzeige ist wichtig für den Nutzer, um Rückmeldung über seine Aktionen zu bekommen. Dazu zählen die Darstellung der eigenen Position, bereits erfasste Stationen und Routen zwischen den Stationen auf der Karte. Auf der Karte platzierte Objekte werden bei Mapbox als *Annotations* bezeichnet. Mit Ausnahme der eigenen Position, welche separat von Mapbox verwaltet wird. *Annotations* werden nochmals in *Marker*, *Polygon* und *Polyline* unterteilt und beschreiben die Darstellung von *Points of Interest* (POI), Flächen und Linien. Im Projektkontext zählen Stationen zu den POI und werden mittels Marker auf der Karte

visualisiert. Der Nutzer kann durch das Tippen auf den Marker mit diesem interagieren und erhält dadurch zusätzliche Informationen über eine Station.

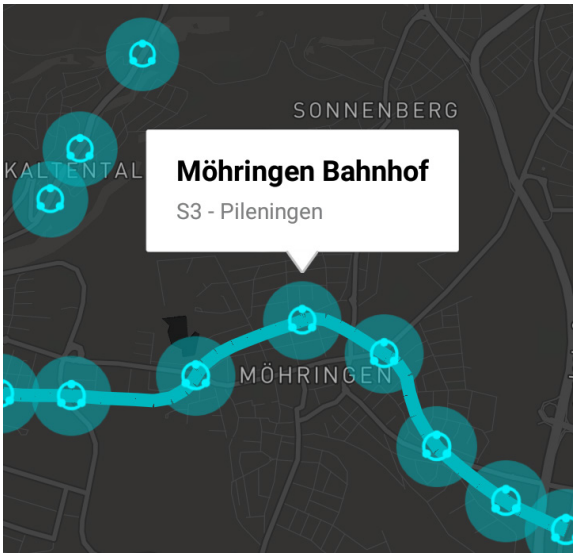


Abbildung 9: Beispiel Stationen und Routen Visualisierung

Polylines werden dagegen für die Darstellung von Routen auf der Karte verwendet. Hierzu werden Positionsdaten nacheinander mittels einer *Polyline* verbunden. Diese Positionsdaten gehören zum GTFS-Format und werden in der Datei S*hapes* verwaltet und unter anderem durch den *Rekorder* mitaufgezeichnet. Shapes beinhalten neben einer Längen- und Breitengrad-Information eine Sequenznummer, welche die Reihenfolge der Shapes definiert. Optional ist es möglich die Distanz, welche in diesem Shape zurückgelegt wurde, mitanzugeben. Diese Information kann für Visualisierungsanwendungen verwendet werden, um Strecken genauer darzustellen.

5.4.1 Eigene Position

Die Darstellung der eigenen Position hilft dem Nutzer sich zu orientieren und seine aktuelle, vom Gerät festgestellte Position auf der Karte darzustellen. Dies muss bei der Initialisierung der Karte explizit aktiviert werden. Die Darstellung der eigenen Position, sowie den Umkreis der Genauigkeit der Position, können zudem farblich angepasst werden.

Quellcode 5: Eigene Position auf der Karte aktivieren und optisch anpassen

```
map.getMapAsync(new OnMapReadyCallback() {
    @Override
    public void onMapReady(final MapboxMap mapboxMap) {

        ...
        // Acitvate own location marker on map
        mapboxMap.setMyLocationEnabled(true);

        // Accuracy options
        mapboxMap.getMyLocationViewSettings().setAccuracyAlpha(40);
        mapboxMap.getMyLocationViewSettings()
           .setAccuracyTintColor(ResourcesCompat.getColor(getResources(),
           R.color.colorAccent, null));

        // Marker options
        mapboxMap.getMyLocationViewSettings()
           .setBackgroundTintColor(ResourcesCompat.getColor(getResources(),
           R.color.colorAccent, null));
        mapboxMap.getMyLocationViewSettings()
           .setForegroundTintColor(Color.WHITE);
    }
});
```

5.5 Standorterfassung

Mit der Standorterfassung ist es möglich, die Position eines Geräts auf wenige Meter genau zu orten. Für die Standorterfassung bietet Android verschiedene Schnittstellen an. Diese unterscheiden sich hauptsächlich anhand der Genauigkeit und Schnelligkeit der bereitgestellten Geokoordinaten.

5.5.1 Ortungs-Provider

5.5.1.1 Mobilfunknetz-Ortung

Am wenigsten präzise ist die Positionsbestimmung über das Mobilfunknetz. Hierbei ist die Genauigkeit von der Anzahl an Mobilfunkstationen abhängig und kann mehrere hundert Meter betragen. Allerdings kann durch das Mobilfunknetz im Gegensatz zum Global Positioning System (GPS) sehr schnell eine ungefähre Geoposition ermittelt werden und benötigt dabei keine zusätzliche Akkuleistung (vgl. Calvo, 2015, S.423).

5.5.1.2 WLAN-basierte Ortung

Ebenfalls sehr schnell ist die Positionsbestimmung über eine WLAN-basierte Ortung. Hierbei wird die Position durch das WLAN-Signal verschiedener Router ermittelt und lässt dabei ebenfalls eine Ortung mit einer Genauigkeit unter 100 Metern innerhalb von Gebäuden zu.

5.5.1.3 GPS Ortung

GPS kann nach der Aktivierung einige Sekunden benötigen, bis die Position des Gerätes ermittelt wurde. Im Vergleich zur Positionsbestimmung über Mobilfunknetz und WLAN ist mit GPS die genauste Ortung bis auf wenige Meter möglich. Allerdings benö-

tigt die Nutzung von GPS sehr viel Strom und beansprucht den Geräte-Akku daher stark.

5.5.2 Lokalisierungsstrategien in Android

Google verfolgt bei Android verschiedene Strategien bei der Lokalisierung des Gerätes. Es ist möglich sich bei der Entwicklung für eine Ortungs-Variante zu entscheiden oder eine Kombination der einzelnen Varianten zu verwenden, um die Vorteile von Schnelligkeit und Genauigkeit zu vereinen.

Im Jahr 2013 wurde von Google der *Fused Location Provider* vorgestellt. Dieser ist Teil der *Google Play Services*, welche, sofern eine Google Play-Store App installiert ist, auf dem Gerät vorhanden sind. Der *Fused Location Provider* verwendet intern den *LocationManager*, der zuvor für die Lokalisierung bei Android eingesetzt wurde (vgl. Google I/O, 2016). Mit dem *LocationManager* war es bereits möglich verschiedene Ortungs-Varianten zu kombinieren und über Kriterien zu bestimmen, wie präzise und häufig der Standort abgefragt wird (vgl. Android Developers, 2016c). Der *Fused Location Provider* hat diese Eigenschaft übernommen und vereinfacht. Der Entwickler kann nun zwischen 3 Prioritäten wählen, die sich anhand des Akkuverbrauchs und Genauigkeit unterscheiden (siehe Abbildung 10). Hierzu nutzt er zusätzliche im Smartphone integrierte Sensoren, um mehr Informationen über die Bewegungsart des Nutzers zu ermitteln. Außerdem kann Google dadurch schneller und unkomplizierter Updates durchführen, ohne dass das Betriebssystem aktualisiert werden muss (vgl. Google I/O, 2016).

In Anhang F befindet sich eine Grafik, welche die Änderungen zwischen der *Location API* und dem *Fused Location Provider* optisch verdeutlicht.

Fused Location Provider: Priority options

Galaxy Nexus (over multiple tests)

Priority	Typical Interval	Battery Drain per Hour (%)	Accuracy*
HIGH_ACCURACY	5 seconds	7.25%	~10 meters
BALANCED_POWER	20 seconds	0.6%	~40 meters
NO_POWER	N/A	small	~1 mile?

Abbildung 10: Fused Location Provider Prioritäten

5.5.3 Präzisierung der Position

Durch die Kombination verschiedener Ortungs-Provider ist es möglich, die Position bereits auf wenige Meter genau zu bestimmen. Dennoch kann es vorkommen, dass die

aktuell ermittelte Position sehr weit von der vorherigen Position, welche wenige Sekunden zuvor erfasst wurde, abweicht. Ursachen hierfür können unter anderem die Umgebung des Nutzers oder die Anzahl der verfügbaren GPS-Satelliten sein.

Um die Präzision der Ortung zusätzlich zu verbessern, wird die von *Fused Location Provider* bereitgestellte Position durch zwei Verfahren aufgewertet.

5.5.3.1 Verfahren 1: Ungültige Positionsdaten

Das erste Verfahren ist die Verwerfung von Positionsdaten, welche ausgehend von der letzten Position weiter entfernt sind, als durch ein Fahrzeug des öffentlichen Personenverkehrs, in der Zeit technisch möglich ist zurückzulegen. Hierzu wird der schnellste Zug der Welt (vgl. DIE WELT, 2014) als Referenzgeschwindigkeit verwendet. Der Zug CRH380 aus China hat eine Höchstgeschwindigkeit von 350 km/h. Die Intervalldauer der Positionsortung durch den *Fused Position Provider* beträgt 5 Sekunden (siehe Abbildung 10). Daraus ergibt sich eine mögliche Distanz von 1750 Metern, die in einem Zeitraum von 5 Sekunden erreicht werden kann.

Quellcode 5: Verwerfen von fehlerhaften Positionsdaten

```
private LocationListener locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(final Location location) {

        // Discard wrong locations
        if(location.distanceTo(lastlocation1) > 1750.0f)
            return;

        ...
        }
};
```

Mit diesem Verfahren lassen sich sehr einfach fehlerhafte Positionsdaten verwerfen. Die Toleranz von 1750 Metern ist allerdings sehr hoch. Unter Berücksichtigung der Durchschnittsgeschwindigkeit des sich bewegenden Fahrzeuges, könnte dieses Verfahren zu einem späteren Entwicklungsschritt weiter präzisiert werden.

5.5.3.2 Verfahren 2: Interpolation

Aufgrund der Ungenauigkeit der ermittelten Positionen können selbst bei Stillstand die Positionsdaten von den wenigen Sekunden zuvor erfassten Positionsdaten abweichen. Um diese daraus resultierende Streuung zu minimieren, wird mittels linearen Interpolationsverfahren ein Mittelwert der Positionsdaten ermittelt.

$$Latitude(interp.) = \frac{Latitude1+Latitude2}{2} \qquad Longitude(interp.) = \frac{Longitude1+Longitude2}{2}$$

Abbildung 11 zeigt die lineare Interpolation von zwei Positionsdaten (Ortung 1, Ortung 2), welche durch den *Fused Location Provider* in einem Zeitabstand von 5 Sekunden ermittelt wurden.

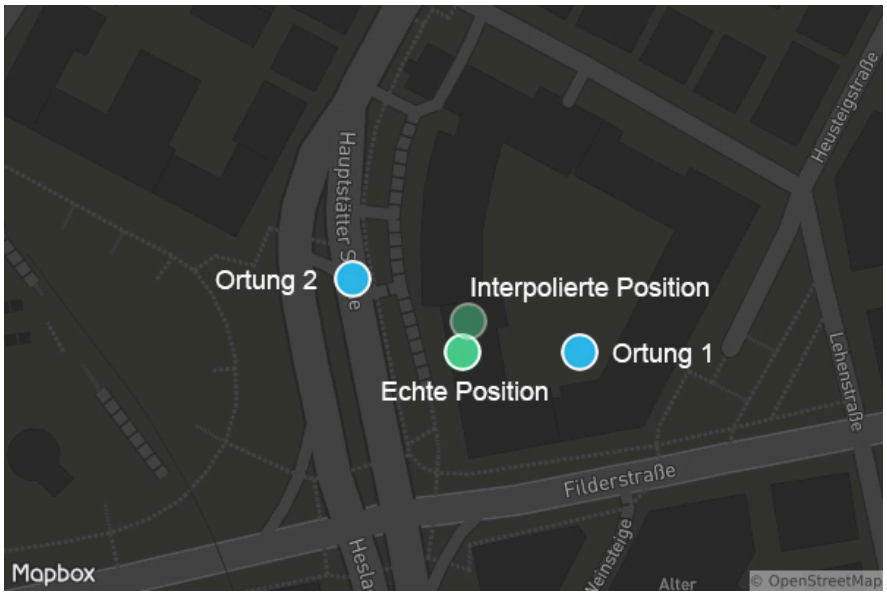


Abbildung 11: Beispiel einer linearen Interpolation von 2 Positionsdaten

Für die lineare Interpolation werden mindestens zwei Positionsdaten benötigt. Eine größere Anzahl an Positionsdaten würde die Ortung eines stillstehenden Gerätes zunehmend präzisieren. Die Ortung eines sich bewegenden Gerätes würde dagegen durch Interpolation zu einem unpräziseren Ergebnis führen.

5.6 Applikationsserver

Für die Implementierung der Serverkomponente wurde Node.js verwendet. Nach Wilson ist Node.js ein Wrapper für die hochperformante V8-JavaScript-Runtime des Google Chrome Browsers und damit nicht nur ein eventbasierter, leichtgewichtiger und skalierbarer Server, sondern gleichzeitig eine Laufzeitumgebung für JavaScript (2012, S.1-7). Der Applikationsserver besteht im Wesentlichen aus einer REST-API, zur Annahme und Verarbeitung von Anfragen, und einer MySQL-Datenbank, welche für die Persistierung und Bereitstellung der Daten zuständig ist. In den nachfolgenden zwei Abschnitten werden die verwendeten Module, sowie der Verbindungsaufbau zur Datenbank näher erläutert.

5.6.1 Module

Durch Module ist es möglich die Funktionalität des Servers durch externe Bibliotheken und Frameworks zu erweitern. Folglich ist der Node.js-Server sehr flexibel und dynamisch einsetzbar. Tabelle 17 zeigt Module, die für die Umsetzung der Serverkomponente zusätzlich verwendet wurden.

Tabelle 17: Verwendete Node.js-Server Module

Modul	**Beschreibung und Verwendungszweck**
mysql	MySQL-Treiber für Node.js. Wird im Server für den Zugriff auf die MySQL-Datenbank verwendet.
express	Einfaches und flexibles Node.js-Framework mit leistungsfähigen Features und Funktionen für Webanwendungen und mobile Anwendungen (vgl. Express, 2016). Wird im Server für die Definition der REST-API verwendet.
bodyParser	Parst eingehende Anfragen bevor diese von Eventhandlern verarbeitet werden. Wird im Server für das Parsen der Anfrage im JSON-Format verwendet.
async	Framework für den Umgang mit asynchronem JavaScript. Wird im Server für strukturierte Datenbankzugriffe verwendet.
fs	Framework für Operationen im Dateisystem. Wird im Server für die Generierung der GTFS-Dateien verwendet.
archiver	Framework für Datenkomprimierung. Wird im Server für die Erstellung von ZIP-Dateien im GTFS-Format verwendet.

5.6.2 Datenbank

Für die serverseitige Datenhaltung wurde eine MySQL-Datenbank nach dem in Anhang A definierten ER-Modell angelegt und konfiguriert. Zusätzlich wurden, wie in Abschnitt 4.3.3 beschrieben, allen Tabellen ein Attribut *timestamp* hinzugefügt.

Der Verbindungsaufbau erfolgt über das Modul *mysql*, welches im vorherigen Abschnitt beschrieben wurde. Quellcode 6 zeigt wie zunächst die Login- und Verbindungsdaten definiert werden und anschließend eine Verbindung zur Datenbank hergestellt wird.

Quellcode 6: Server Datenbankverbindung

```
var mysql = require('mysql');
var db = mysql.createConnection({
    host : 'localhost',
    user : 'root',
    password : '*****',
    database : 'gtfs'
});

// Connect Database
db.connect(function(err) {
    if (err) {
        console.error('error connecting: ' + err.stack);
        return;
    }
    console.log('connected as id ' + db.threadId);
});
```

5.6.3 Schnittstellen

Ohne die Kommunikation zwischen Client und Server wäre ein Datenaustausch zwischen den Clients nicht möglich. Der Server dient als zentraler Datenendpunkt, verwaltet die übertragenen Daten der Clients und stellt diese wiederum den anderen Clients zur Verfügung. Die Kommunikation erfolgt über die in Abschnitt 4.3.1 definierten Schnittstellen. Für die Realisierung wurde das Modul Express.js (siehe Abschnitt 5.6.1) verwendet. Hierbei werden die Schnittstellen durch Angabe der HTTP-Methode (GET, POST, PUT, DELETE), einer URL und optional mit Parametern festgelegt. Parameter können innerhalb der URL vorkommen und werden durch einen führenden Doppelpunkt gekennzeichnet.

Quellcode 7: Definition von Schnittstellen

```
var app = express();

// parse application/json
app.use(bodyParser.json());

//   GET: /transit/stations/
//   REST API for getting stops within a bounding box
app.get('/transit/stations/:bounding', function(req, res){
...
});

//   GET: /transit/station/
//   REST API for getting trips, routes and all related data of a stop
app.get('/transit/station/:stop_id', function(req, res){
...
});

//   GET: /transit/
//   REST API for getting all data for offline use within a bounding box
app.get('/transit/:bounding', function(req, res){
...
});

//   POST: /transit/
//   REST API pushing new/changed data from the client to the server
app.post('/transit', function(req, res) {
...
});

//  GET: /gtfs/download
//  Generates GTFS Data and promp download to the user
app.get('/gtfs/download', function (req, res) {
...
});
```

5.7 Rekorder

Der *Rekorder* wurde als einer der Hauptkomponenten ausgewählt, die im Rahmen der Bachelorarbeit realisiert werden soll. Er wurde gegenüber der manuellen Eingabe von Stationen und Routen bevorzugt, da er eine einfachere, schnellere und bequemere Methode zur Eingabe bietet.

Die *Rekorder*-Komponente besteht aus 2 Teilkomponenten. Die erste Komponente ist eine Activity, welche für das Darstellen der GUI und das Verarbeiten von Nutzerinteraktionen zuständig ist. Die GUI beinhaltet eine Karte, um die aktuelle Position des Nutzers anzuzeigen und um erfasste Stationen und die bereits gefahrene Route visuell dazustellen.

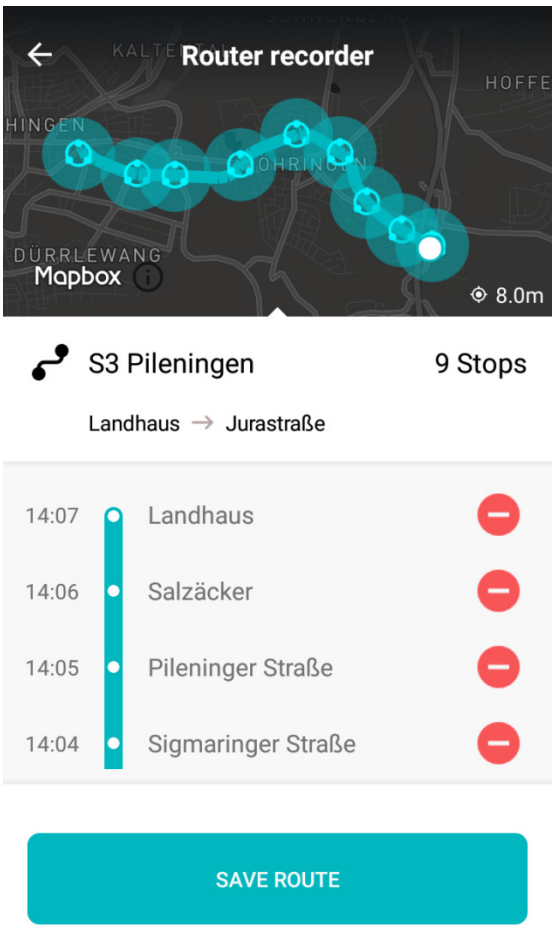


Abbildung 12: Rekorder Screenshot einer beendeten Aufzeichnung

Die zweite Komponente ist ein Service (*RecorderService*), der für die Geschäftslogik, Positionsupdates und die Persistierung der aufgezeichneten Daten zuständig ist. Für das regelmäßige Update der Position wird der *Fused Location Provider* (siehe Abschnitt 5.5) eingesetzt. Dieser wird von der Klasse *LocationService* verwaltet und der *RecorderService* erhält in einem 5 Sekunden Intervall die aktuelle Position des Nutzers. Im Zuge dieses Updates wird die Position des Nutzers mittels Interpolation angenähert (siehe Abschnitt 5.5.3.2). Des Weiteren wird mit jedem Update eine

Polyline zwischen der vorherigen Position und der aktuellen Position auf der Karte gezeichnet und als *Shape* des GTFS-Formates in der Datenbank gespeichert (siehe Abschnitt 5.4). Neben *Shapes* werden alle anderen erfassten Daten ebenfalls direkt einem GTFS-Objekt, wie *Stop*, *StopTime*, *Trip*, *Route*, oder *Calendar* zugeordnet und in der Datenbank gespeichert. Während der *Rekorder* aktiv ist und Daten aufzeichnet, wird bei allen Datensätzen das *temp*-Flag gesetzt. Dadurch kann zwischen temporären und bereits bestätigten validen Datensätzen unterschieden werden. Zugleich ist es möglich, im Falle eines Absturzes der App, den Nutzer auf die bestehenden temporären Daten hinzuweisen und ihm anzubieten diese Daten nachträglich zu bestätigen. So kann der Verlust von Daten verhindert und erfasste Daten dennoch als valide Datensätze übernommen werden.

Bei Android ist es möglich, dass Service zwangsmäßig beendet werden, wenn das System oder eine andere App, die im Fokus ist, mehr Arbeitsspeicher benötigt. Um zu verhindern, dass der Service von Android vorzeitig während der Aufnahme beendet wird, ist es nötig die *startForeground(int, Notification)* API zu verwenden. Diese API soll nach Android Developers (2016d) nur verwendet werden, wenn der Nutzer sich bewusst ist, dass es sich hierbei um einen aktiven Prozess handelt. Beim Aufruf dieser Methode muss eine eindeutige ID und ein *Notification*-Objekt übergeben werden. Dies führt, bis zum Aufruf von *stopForeground(int)*, zu einer permanenten Benachrichtigung im Benachrichtigungsbereich des Gerätes.

6 Ergebnis

Der Client, sowie der Server wurden erfolgreich umgesetzt. Bei Betrachtung der in Abschnitt 3 definierten funktionalen- und nicht funktionalen Anforderungen wurden nicht alle Anforderungen erfüllt. Der Grund hierfür ist die eingegrenzte Realisierung des Clients. So war es nicht möglich den Client im vollen Umfang im Rahmen der Bachelorarbeit umzusetzen. Als Ziel der Realisierung wurde ein funktionierender Kreislauf definiert, beginnend bei der Datenerfassung, über die Übertragung zum Server und im Anschluss die Bereitstellung der Daten für andere Clients. Dieses Ziel wurde erreicht und während praktischen Tests konnten zufriedenstellende Ergebnisse erzielt werden. Das Konzept der Gamifizierung konnte bei der Realisierung aus zeitlichen Gründen nicht berücksichtigt werden.

In den folgenden Abschnitten werden nochmals die Datenqualität, die Einsetzbarkeit und die Schwachstellen der Client-Server-Lösung näher betrachtet.

6.1 Datenqualität

Um eine Aussage über die Datenqualität zu machen, ist es wichtig die Präzision der erfassten Fahrplandaten, sowie die Validität dieser Daten zu betrachten.

Die Positionsbestimmung mit GPS lieferte stets auf wenige Meter genaue Daten, wodurch es möglich war, Stationen und Routen sehr genau aufzuzeichnen. Probleme bezüglich der Genauigkeit wurden bei unterirdischen Strecken deutlich. Dort gab es große Abweichungen in der Positionsbestimmung, welche zu nicht zufriedenstellenden Daten geführt haben.

Um sicher zu stellen, dass es sich bei den vom Server generierten GFTS-Daten um valide Daten des GTFS-Standards handelt, wurden die generierten Daten mit dem von Google zur Verfügung gestellten Open Source Tool *FeedValidator*[4] geprüft. Das Ergebnis wies nach, dass die erstellen GTFS-Daten konform mit dem GTFS-Standard sind.

Zusammenfassend kann die Aussage getroffen werden, dass es sich bei den vom Client erfassten und vom Server generierten Daten um gültige und valide Daten handelt, welche im produktiven Umfeld eingesetzt werden können.

6.2 Einsetzbarkeit

Die Client-App, sowie die Serverkomponente wurden ausreichend umgesetzt, um unter Einschränkungen bereits aktiv zum Sammeln von Daten eingesetzt werden zu können.

[4] FeedValidator Git-Repository - https://github.com/google/transitfeed/wiki/FeedValidator (Datum des Zugriffs: 08. Juli 2016)

Allerdings ist es zum aktuellen Stand noch nicht möglich Routen zusammenzuführen. Verschiedene Teilrouten einer Route werden als separate Routen erkannt und nicht zu einer vollständigen Route verbunden. Das gleiche Problem besteht bei Stationen die ein Haltepunkt für verschiedene Routen sind. Das führt dazu, dass Stationen redundant erfasst und als unterschiedliche Stationen behandelt werden.

Des Weiteren können erfasste Daten weder bearbeitet noch gelöscht werden. Der Funktionsumfang beschränkt sich hierbei ausschließlich auf die Informationsanzeige von Stationen und aufgezeichneten Routen.

6.3 Datenverkehr

Gemäß den in Abschnitt 3.3 definierten nicht-funktionalen Anforderungen sollen die zu übertragenden Daten und die Latenz möglichst gering sein. Die Metainformationen wurden durch die Nutzung von JSON als Übertragungsformat stark reduziert. Ein Ansatz zur weiteren Reduzierung wurde in Abschnitt 4.3.1.6 näher erläutert.

Routen die mit dem *Rekorder* zu Testzwecken aufgezeichnet wurden, wiesen keine unüblichen Datengrößen auf. Die Schnittstellen des Servers und die durch Vergrößerungsstufen eingegrenzte Abfragen des Clients lassen keine Übertragung von größeren Datenmengen im Megabyte-Bereich zu. Eine Ausnahme bildet hierbei das Herunterladen von Stationsdaten bei Aktivierung des Offline-Modus.

Tabelle 18 zeigt am Beispiel einer aufgenommenen Route, bestehend aus 8 Stationen, die dazu erzeugten Datenmengen im JSON-Format beim Übertragen zum Applikationsserver.

Tabelle 18: Beispiel Datenmengen einer Route mit 8 Stationen im JSON-Format

Typ	**Menge**	**Datenmenge in Bytes**
Stops	8	3.620
Stoptimes	8	2.620
Trip	1	454
Route	1	303
Shapes	81	19.550
Calendar	1	289
	Summe	26.836 (26,836 KB)

Eine Aussage über die Latenz lässt sich zum gegenwärtigen Zeitpunkt nicht machen, da Verbindungen mit dem Server ausschließlich in einer Testumgebung mit optimalen Bedingungen stattgefunden haben.

6.4 Schwachstellen

Neben den in Abschnitt 6.1 genannten Schwächen bezüglich der Genauigkeit der Positionsbestimmung auf unterirdischen Strecken, liegt die größte Schwäche in der benötigten Akkuleistung während des Aufzeichnens mit dem *Rekorder*. Um die Position möglichst genau bestimmen zu können, wird die Einstellung mit der höchsten Genauigkeit und gleichzeitig dem höchsten Akkuverbrauch verwendet. Moderne Akkus werden immer leistungsfähiger, allerdings kann sich der Verbrauch bei längeren Routen deutlich bemerkbar machen.

7 Zusammenfassung

Die von der moovel Group GmbH geforderte Client-Server-Lösung wurde erfolgreich konzipiert und in einer ersten Version als Prototyp realisiert.

Ausgehend von der Zielsetzung in Abschnitt 1.2 wurden die Anforderungen an das Projekt ermittelt, um anschließend ein technisch versiertes Konzept zu entwickeln. Hierfür war es nötig sich zunächst in die Standards von Kartenformaten einzuarbeiten, um ein Verständnis für die Funktionsweise von Kartendiensten zu erlangen. Im Anschluss war es während der Konzipierung möglich verschiedene Kartendienste anhand ihrer Eigenschaften zu vergleichen und zu evaluieren, welcher Dienst die nötigen Voraussetzungen erfüllt. Das GTFS-Format dient als Grundlage und Zielformat, welches es zu erreichen gilt. Daher war es nötig sich intensiv mit dem Format auseinanderzusetzen. Bereits existierende GTFS-Editoren wurden untersucht, um festzustellen, ob ein bereits existierender Editor die Zielstellung und die in Abschnitt 1.3 definierten Anforderungen erfüllt. Keiner der Editoren eignete sich und es konnte mit der Konzeption einer eigenen Lösung begonnen werden. Als Plattform für den mobilen Client konnte das Android-Betriebssystem als am geeignetsten festgestellt werden. Darauf aufbauend wurden vorhandene Technologien eruiert und anhand der gewonnenen Informationen konnten geeignete Frameworks und Strategien für die Realisierung der Android-App gefunden werden. Serverseitig wurde neben der Einrichtung eines MySQL-Servers entsprechende Schnittstellen für die Kommunikation mit dem Client definiert und ein einheitliches Übertragungsformat und Struktur festgelegt und umgesetzt.

Die bereitgestellte Client-Server-Lösung kann von der moovel Group GmbH als Basis für eine zukünftige weiterführende Realisierung verwendet werden. Über die App erfasste Daten können in den eigenen Produkten verwendet werden, um vorhandene Datenbestände zu erweitern.

Entwickler hätten, mit der freien Zurverfügungstellung der Fahrplandaten, die Möglichkeit diese in ihren bestehenden Anwendungen zu verwenden oder neue Anwendungen auf Basis von GTFS-Daten zu entwickeln.

7.1 Ausblick

Das Projekt soll nach Abschluss dieser Bachelorarbeit weitergeführt und in vollem Umfang fertiggestellt werden. Dazu zählt die Möglichkeit bestehende Stationen und Routen zu bearbeiten oder zu löschen. Neben dem *Rekorder* soll es weitere Eingabemöglichkeiten geben Routen und Stationen anzulegen. Nutzer sollen in der Lage sein Fotos bei Stationen zu hinterlegen und sich untereinander in einer Community auszutauschen. Durch Gamifizierung sollen Nutzer zusätzlich motiviert werden Fahrplandaten zu erfassen. Das Projekt bietet Potenzial weltweit eingesetzt zu werden und einen großen Beitrag in der Open Data Community zu leisten.

Quellenverzeichnis

Bildquellen

Catalonia (2011): Raster / Vector Image Models: Comparison of Raster and Vector Methods. http://www.catalonia.org/cartografia/Clase_03/Raster_Vector.html (Datum des Zugriffs: 14. Mai 2016)

Google Developers (2016): Google Maps Android API: Kachelüberlagerungen. https://developers.google.com/maps/documentation/android-api/tileoverlay#coordinates (Datum des Zugriffs: 13. Mai 2016).

Google I/O (2013): Quelle für Anhang E. https://www.youtube.com/watch?v=Bte_GHuxUGc (Datum des Zugriffs: 01. Juni 2016)

Realm.io (2014): Introducing Realm. https://realm.io/news/introducing-realm/ (Datum des Zugriffs: 23. Juni 2016)

StatCounter Global (2016): Quelle für Abbildung 4 und Abbildung 5. http://gs.statcounter.com/ (Datum des Zugriffs: 14. Mai 2016)

Literaturverzeichnis

Android Developers (2016a): Storage Options http://developer.android.com/guide/topics/data/data-storage.html (Datum des Zugriffs: 11. Mai 2016)

Android Developers (2016b): Projects Overview https://developer.android.com/studio/projects/index.html (Datum des Zugriffs: 27. Mai 2016)

Android Developers (2016c): Location Strategies https://developer.android.com/guide/topics/location/strategies.html (Datum des Zugriffs: 01. Juni 2016)

Android Developers (2016d): Service https://developer.android.com/reference/android/app/Service.html (Datum des Zugriffs: 05. Juli 2016)

Bennett, J. (2010): OpenStreetMap: Be your own Cartographer. Birmingham: Packt Publishing Ldt., ISBN: 978-1-847197-50-4.

Bibiana McHugh (2013): Pioneering Open Data Standards: The GTFS Story. http://beyondtransparency.org/chapters/part-2/pioneering-open-data-standards-the-gtfs-story/ (Datum des Zugriffs: 03. Mai 2016).

Blömeke, M. (2014): GPSMAP 62 + 64 Handbuch: Einfach Touren planen und navigieren. Norderstedt: Books on Demand, ISBN: 978-3-7347-3607-0.

Bounding Box (2015): Boundary Bounding Box for Countries, States, Counties, and Zipcodes: Lulu Press, Inc., ISBN: 978-1-329-79160-2.

Calvo A. (2015): Beginning Android Wearables: With Android Wear and Google Glass SDKs. New York: Springer Science+Business Media New York. ISBN: 978-1-484205-18-1.

CNN (2012): Seven ways mobile phones have changed lives in Africa http://edition.cnn.com/2012/09/13/world/africa/mobile-phones-change-africa/index.html (Datum des Zugriffs: 03. Mai 2016)

Davis, B.E. (2001): GIS: A Visual Approach. 2. Auflage. Boston: Cengage Learning, ISBN: 978-0766827646.

Dickmann, F. (2004): Einsatzmöglichkeiten neuer Informationstechnologien für die Aufbereitung und Vermittlung geographischer Informationen: Das Beispiel kartengestützte Online-Systeme. Göttingen: Goltze, ISBN: 978-3-884-52112-0.

DIE WELT (2014): Das sind die zehn schnellsten Züge der Welt. http://www.welt.de/reise/article133803025/Das-sind-die-zehn-schnellsten-Zuege-der-Welt.html (Datum des Zugriffs: 01. Juni 2016)

Google (2016): Legal Notices for Google Maps/Google Earth and Google Maps/Google Earth APIs http://www.google.com/intl/en_ALL/help/legalnotices_maps.html (Datum des Zugriffs: 20. Juli 2016)

Google Developers (2016): What is GTFS? https://developers.google.com/transit/gtfs/ (Datum des Zugriffs: 05. Mai 2016)

Google I/O (2013): Google I/O 2013 - Beyond the Blue Dot: New Features in Android Location. https://www.youtube.com/watch?v=Bte_GHuxUGc (Datum des Zugriffs: 01. Juni 2016)

Hecker T. (2016): Besprechung über den Umfang der Realisierung. Persönliches Interview, geführt vom Verfasser. Stuttgart, 23. Juni 2016.

ITWissen (2016): Framework http://www.itwissen.info/definition/lexikon/Framework-framework.html (Datum des Zugriffs: 08. Juli 2016)

JSON (2016): Introducing JSON http://www.json.org/ (Datum des Zugriffs: 23. Mai 2016)

Kaufmann, S. (2014): Opening Public Transit Data in Germany. Ulm: Universität Ulm.

Liqiu, M., Zipf, A., Winter S. (2008): Map-based Mobile Services: Design, Interaction and Usability. Berlin Heidelberg: Springer-Verlag, ISBN: 978-3-540-37109-0.

Mapbox (2016a): How web maps work
https://www.mapbox.com/help/how-web-maps-work/ (Datum des Zugriffs: 19. Mai 2016)

Mapbox (2016b): Offline maps with Mapbox Mobile
https://www.mapbox.com/help/mobile-offline/ (Datum des Zugriffs: 20. Mai 2016)

Massé, M. (2012): REST API Design Rulebook. Sebastopol: O'Reilly Media, Inc., ISBN: 978-1-449-31050-9.

Moovel (2016): Die moovel Group.
https://www.moovel.com/de/DE/company (Datum des Zugriffs: 03. Mai 2016)

ODbL (2016): Open Database License (ODbL) v1.0
http://opendatacommons.org/licenses/odbl/1-0/ (Datum des Zugriffs: 20. Juli 2016)

Realm.io (2016): A better mobile database means better apps.
https://realm.io/ (Datum des Zugriffs: 23. Juni 2016)

Schneider, G., Vecellio S. (2011): ICT-Systemabgrenzung, Anforderungsspezifikation und Evaluation: Grundlagen zur Systemanalyse und –beschaffung mit Beispielen, Fragen und Antworten. Zürich: Compendio Bildungsmedien AG, ISBN:978-3-7155-9563-4.

Statista (2015a): Welche Location bzw. Verkehrsmittel-App besitzen Sie?
http://de.statista.com/statistik/daten/studie/452968/umfrage/besitz-von-location-und-verkehrsmittel-apps-nach-altersgruppen-in-oesterreich/
(Datum des Zugriffs: 03. Mai 2016).

Statista (2015b): Anzahl der Smartphone-Nutzer in Deutschland in den Jahren 2009 bis 2015 (in Millionen)
http://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/ (Datum des Zugriffs: 03. Mai 2016)

Svennerberg, G. (2010): Beginning Google Maps API 3. New York: Springer Science+Business Media, ISBN: 978-1-4302-2803-5.

VBB (2015): VBB GTFS – Stand 16. Dezember 2015.
https://transitfeeds.com/p/verkehrsverbund-berlin-brandenburg/213/latest (Datum des Zugriffs 08. Juli 2016)

Vogel, O., Arnold, I., Chughtai, A., Ihler, E., Kehrer, T., Mehlig, U., Zdun, U. (2009): Software-Architektur: Grundlagen – Konzepte – Praxis. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, ISBN: 978-3-8274-1933-0.

Wikipedia (2016): MySQL.
https://de.wikipedia.org/wiki/MySQL (Datum des Zugriffs 08. Juli 2016)

Wilson M., **Hughes-Croucher T.** (2012): Einführung in Node.js. Köln: O'Reilly Verlag GmbH & Co. KG. ISBN: 978-3-86899-797-2.

Yavuz, H. (2011): Crowdsourcing: Eine systematische Litaturanalyse. Hamburg: Diplomica Verlag GmbH. ISBN: 978-3-8428-6556-3.

Glossar

API: Eine Programmierschnittstelle die anderen Anwendungen eine Anbindung an ein System ermöglicht

App: Beschreibt eine Anwendung auf einem mobilen Endgerät

Bounding Box: Ein quadratischer Bereich definiert durch zwei Geographische Koordinaten, welche jeweils durch ein Breitengrad und einem Längengrad definiert sind (vgl. Bounding Box, 2015)

Crowdsourcing: Beschreibt die Auslagerung von Aufgaben an eine Gruppe von Freiwilligen, die dadurch einen direkten Mehrwert für sich selbst oder Anerkennung erhalten (vgl. Yavuz, 2011, S.1)

Framework: Ein Programmiergerüst „das dem Programmierer den Entwicklungsrahmen für seine Anwendungsprogrammierung zur Verfügung stellt“ (ITWissen, 2016)

Gamifizierung: Beschreibt die Methode, triviale und monotone Anwendungen durch spieltypische Elemente interessant zu gestalten

GPS: Ein globales Positionbestimmungssystem

GTFS: Ein international standardisiertes Format von Google zur Bereitstellung von Fahrplänen und geografischen Informationen

HTTP: Ein Protokoll zur Kommunikation zwischen Servern und Clients

JSON: Ein kompaktes textbasierendes Format zum Datenaustausch zwischen verschiedenen Systemen

SQLite: Ein weit verbreitetes relationales Datenbanksystem. Die SQLite-Bibliothek lässt sich direkt in eine Software integrieren, sodass ein Zugriff auf die Datenbank ohne zusätzliche Treiber oder einen Datenbankserver möglich ist

MySQL: Ein weit verbreitetes relationales Datenbankverwaltungssystem. Es ist als Open-Source-Software sowie als

	kommerzielle Enterprise Version für verschiedene Betriebssysteme verfügbar (vgl. Wikipedia, 2016)
Open Data:	Beschreibt die freie Verfügbarkeit und Nutzung von Daten
Open Source:	Bezeichnet Software deren Quellcode öffentlich ist und von Dritten eingesehen werden kann
Software Development Kit:	Eine Zusammenstellung von Werkzeugen mit dem Ziel der Erstellung von Software

Anhang

Anhangsverzeichnis

Anhang A: GTFS ER-Model

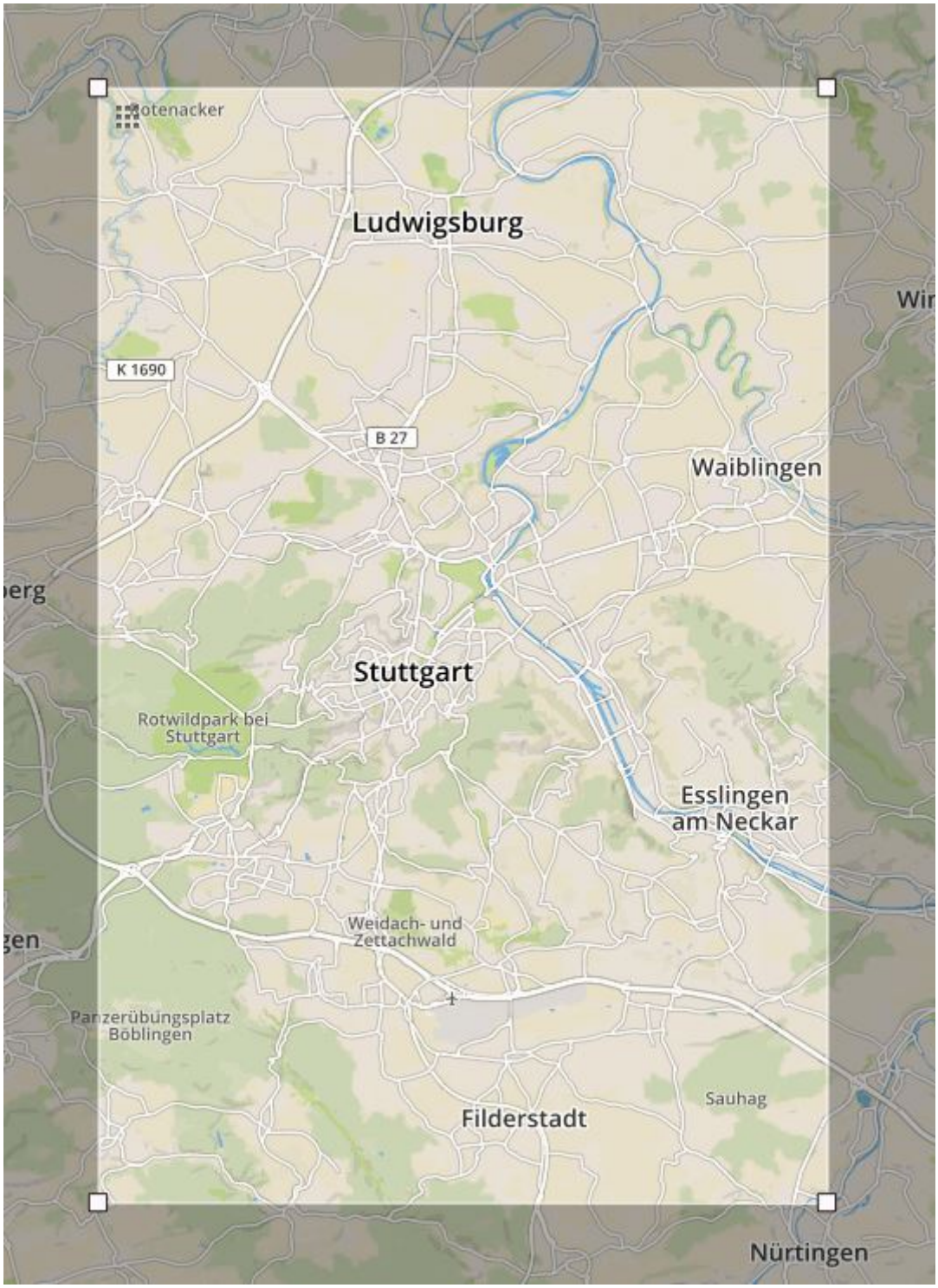
Anhang B: Kartendienste

Tabelle 19: Kartendienste im Vergleich

Kartendienste	Offline-Modus	Kachelformat	Plattform Support	Eigene Styles	Offene Daten	Kommerziell	Besonderheiten	Gesamt-punkte
Google Maps	Ja	Vektor / Raster	Web, Android, iOS	Ja	Nein	Ja	Offline-Modus exklusiv für Google Maps App	7
Punkte	1	2	3	1	0	0		
Apple Maps	Nein	Vektor / Raster	iOS	Nein	Nein	Nein		3
Punkte	0	2	1	0	0	0		
Bing Maps	Nein	Vektor / Raster	Web	Nein	Nein	Ja		3
Punkte	0	2	1	0	0	0		
OpenStreetMap	Ja	Raster	Web, Android, iOS	Nein	Ja	Nein		7
Punkte	1	1	3	0	1	1		
HERE Maps	Ja	Vektor / Raster	Web, Android, iOS	Ja	Nein	Ja	Die ersten 90 Tage kostenfrei	7
Punkte	1	2	3	1	0	0		
Mapbox	Ja	Vektor / Raster	Web, Android, iOS	Ja	Ja	Ja	Kostenfrei bis 50.000 Zugriffe pro Monat	8
Punkte	1	2	3	1	1	0		
MapForge	Ja	Vektor	Android	Ja	Ja	Nein	Kein Online-Modus	7
Punkte	1	2	1	1	1	1		
Mapzen	Nein	Vektor	Web, Android	Ja	Ja	Nein		7
Punkte	0	2	2	1	1	1		

Für mehr Informationen zur Punktevergabe siehe Abschnitt 4.2.4.1

Anhang C: Beispiel offline Kartenausschnitt



Quelle: https://www.mapbox.com/labs/offline-estimator/

Minimale Vergrößerungsstufe: 0

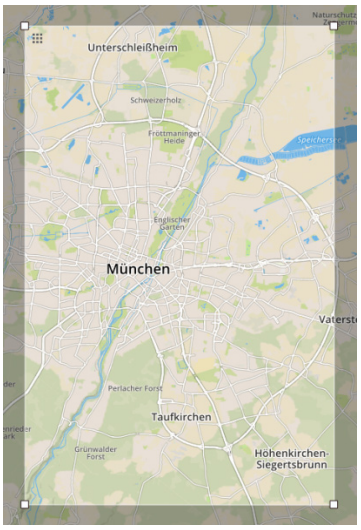
Maximale Vergrößerungsstufe: 16

Aktuelle Vergrößerungsstufe: 10

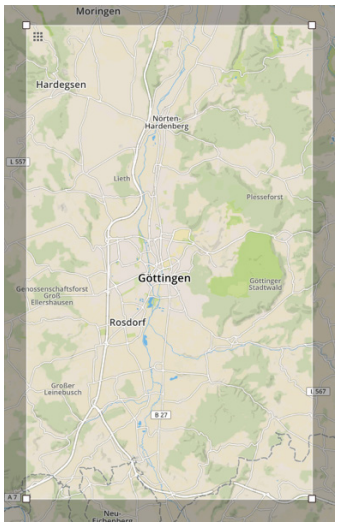
Kachelanzahl: 5992

Anhang D: Toleranz Kachelanzahlschätzung

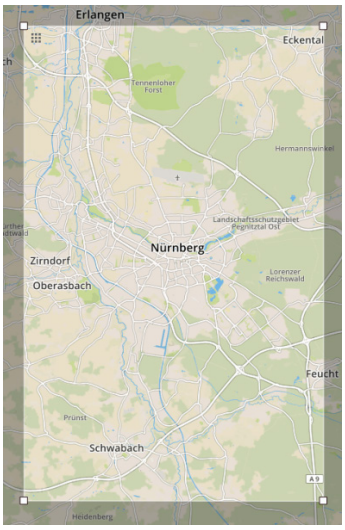
Folgende Kartenausschnitte wurden mit dem *Offline-Estimator* von Mapbox, für eine Toleranzschätzung der Kachelanzahl, selektiert:



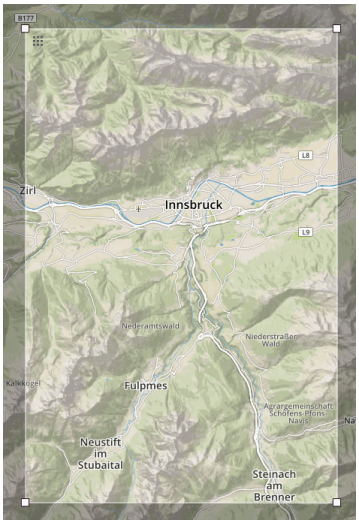
Kachelschätzung
5815



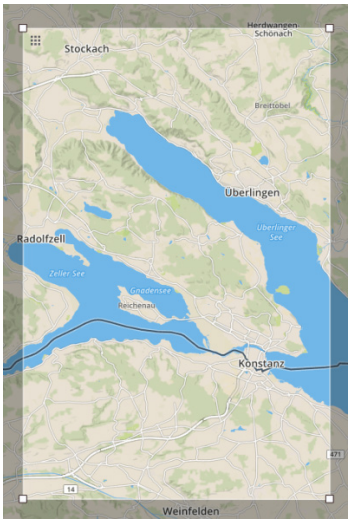
Kachelschätzung
6279



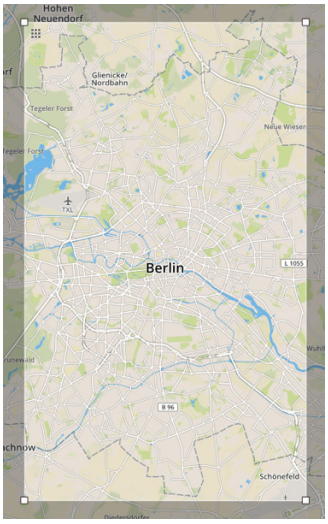
Kachelschätzung
5834



Kachelschätzung
5641



Kachelschätzung
5663



Kachelschätzung
6394

Quelle: https://www.mapbox.com/labs/offline-estimator/

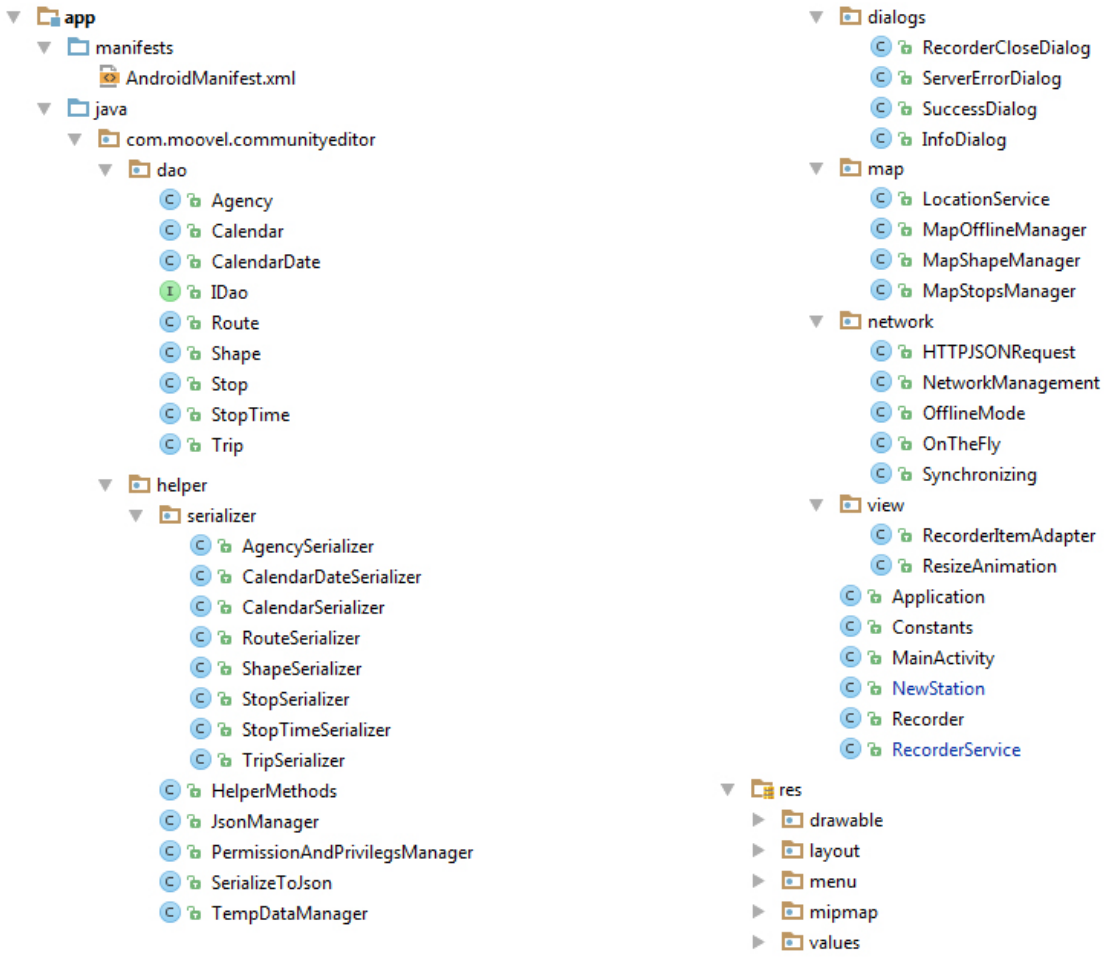
Alle Kartenausschnitte wurden unter folgenden Einstellungen aufgenommen:

Minimale Vergrößerungsstufe: 0

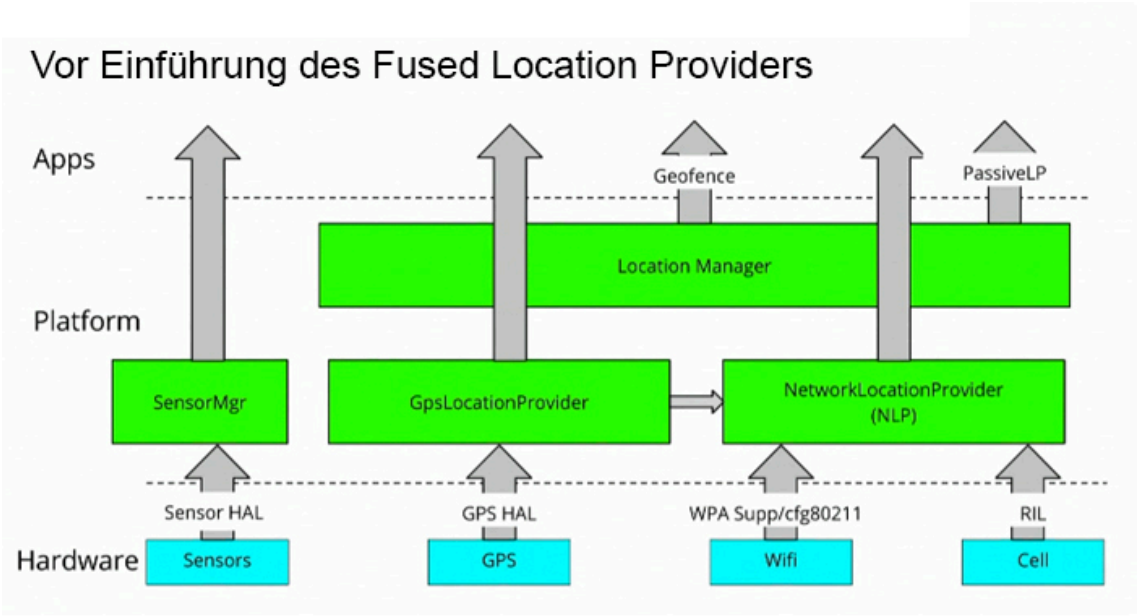
Maximale Vergrößerungsstufe: 16

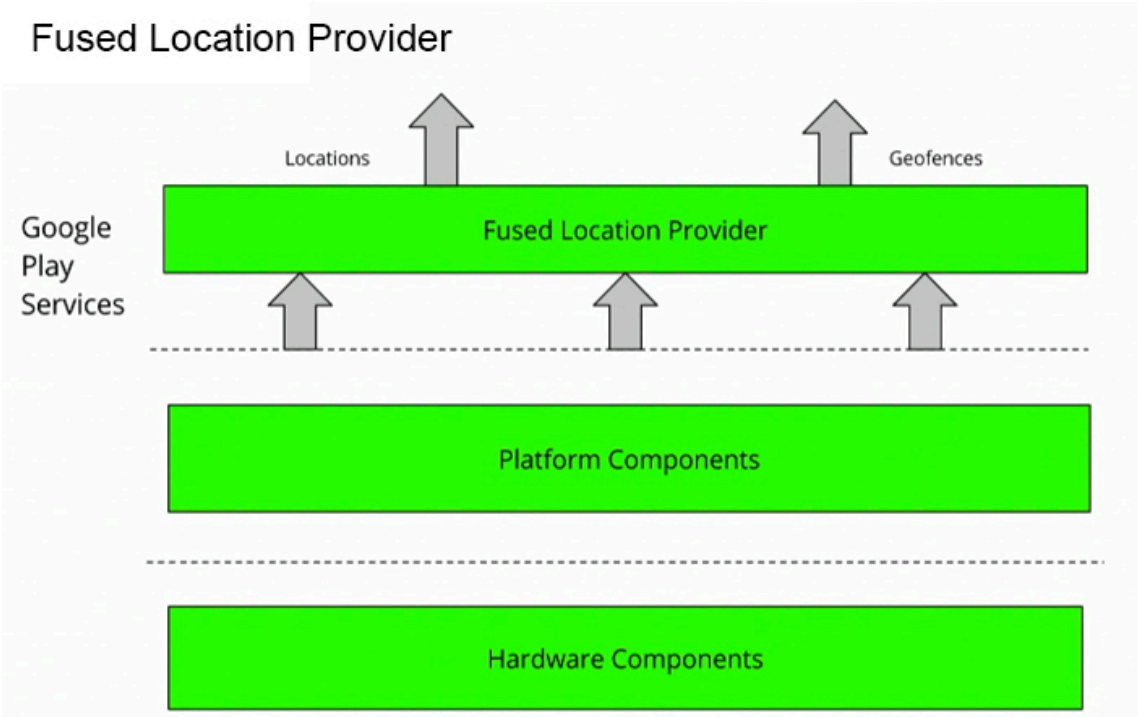
Aktuelle Vergrößerungsstufe: 10

Anhang E: Verzeichnisstruktur der Client-App



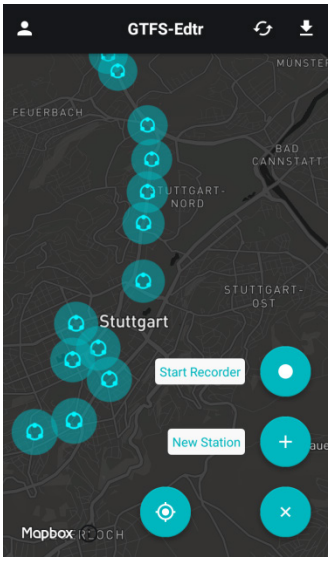
Anhang F: Vergleich Location API und Fused Location Provider





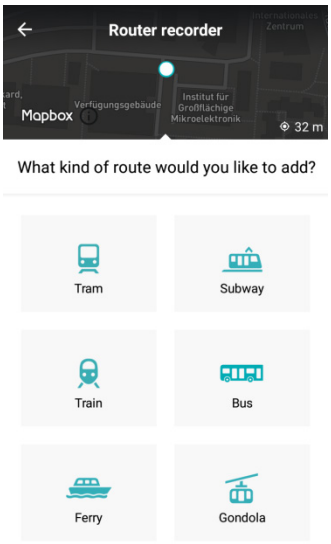
Quelle: Google I/O, 2013

Anhang G: Rekorder Ablaufplan



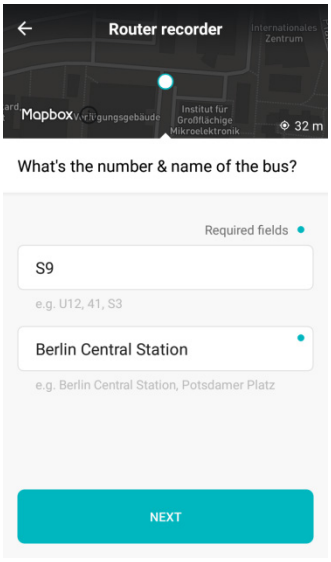
Schritt 1: Rekorder starten

Über das Menü in der rechten unteren Ecke lässt sich der *Rekorder* starten.



Schritt 2: Verkehrsmittel auswählen

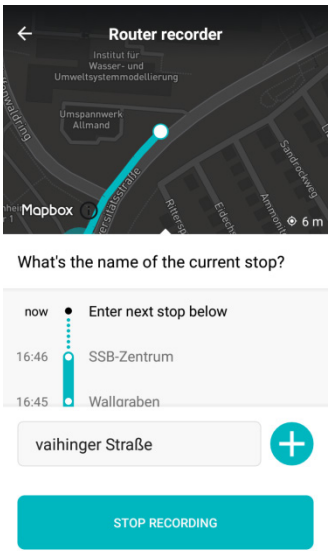
Der Nutzer erhält verschiedene öffentliche Verkehrsmittel zur Auswahl. Das gewählte Fahrzeug wird der Route zugewiesen. Einer Route kann nur ein Verkehrsmittel zugewiesen werden.



Schritt 3: Routenname und Richtung definieren

Bei diesem Eingabefenster kann der Nutzer den Namen und die Richtung der Route nennen. Die Richtung der Route ist hierbei verpflichtend anzugeben. Der Routenname dagegen ist optional.

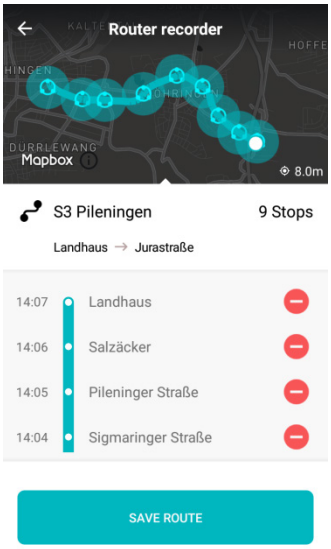
Beispiele: S9 – Berlin Central Station



Schritt 4: Eingabe von Stationen

In dieser Ansicht kann der Nutzer nun jedes Mal wenn er an einer Station an kommt, den Namen der Station eintragen. Durch den „+“-Button wird die Station in der Auflistung darüber aufgenommen und die Station auf der Karte hinzugefügt. Während der Aufnahme wird zusätzlich die zurückgelegt Strecke in Form einer Linie auf der Karte gezeichnet. Sobald die erste Station eingetragen ist beginnt die Aufzeichnung. Zu diesem Zeitpunkt ist es nicht mehr möglich zurück zu den vorherigen Ansichten zu gehen und die Daten der Route zu ändern.

Über den Button „stop recording“ beendet der Nutzer die Aufzeichnung und gelangt zu einer Übersicht über die aufgezeichnete Route.



Schritt 5: Route bestätigen

In der letzten Ansicht erhält der Nutzer eine Übersicht über die während der Aufzeichnung aufgenommenen Daten und erfassten Stationen. Er hat die Möglichkeit die Daten zu prüfen und fehlerhaft eingegebenen Stationen zu bearbeiten. Über den „save route“-Button kann der Nutzer die aufgenommenen Daten speichern und den *Rekorder* beenden.

Sollte der *Rekorder* oder die App unerwarteter Weise beendet werden sind die Daten nicht verloren. Der Nutzer erhält beim Start der App eine Meldung darüber, dass es noch ungesicherte Daten gibt. Er kann an dieser Stelle entscheiden ob die Daten gesichert werden oder verworfen werden.

Anhang H: Datenträger

Der beigefügte Datenträger enthält die folgenden Daten:

- Dieses Dokument als PDF-Datei
- Der im Rahmen der Bachelorarbeit erstellte Quellcode
- Generierte APK-Datei der Android App
- Beispiele für Server-Antworten- und Anfragen der einzelnen Schnittstellen