Hochschule Kaiserslautern
University of Applied Sciences

Informatik und Mikrosystemtechnik
Zweibrücken

Studiengang

Medieninformatik

PO Version 2011

Bachelorarbeit

Implementierung des Prototypen einer cross-platform App zur Bustransferüberwachung

Implementation of the prototype of a cross-platform app for bus transfer monitoring

vorgelegt von

Johannes Schmitt

22. Februar 2018

Betreuung: Prof. Dr. Manfred Brill

Zweitkorrektor: Prof. Dr.-Ing. Jan Conrad

Kurzfassung

In dieser Bachelorarbeit wird eine Anwendung zum Zwecke der Überwachung von Bustransfers entwickelt. Es handelt sich um eine Client-Server Applikation, die in TypeScript realisiert ist. Die Kombination aus Apache Cordova, Angular 5 und Ionic 4 ermöglicht eine plattformübergreifende Nutzung. Zur Positionsbestimmung werden GPS und iBeacons eingesetzt. Die Kommunikation wird über eine REST-API abgewickelt.

Abstract

This bachelor thesis is about the development of an application for bustransfer surveillance. It is a client-server application realized in TypeScript. The combination of Apache Cordova, Angular 5 and Ionic 4 enables a cross-platform usage. Location tracking technologies in this app are based on GPS and iBeacon. The communication is realized by a REST-API.

Ehrenwörtliche Erklärung

Hiermit erkläre ich, **Johannes Schmitt**, geboren am **31.10.1982 in Berlin-Spandau**, ehrenwörtlich,

- dass ich meine Bachelorarbeit mit dem Titel

  Implementierung des Prototypen einer cross-platform App zur Bustransferüberwachung

  selbstständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angegebenen Hilfen benutzt habe;

- dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Zweibrücken, 22.02.2018 **[Johannes Schmitt]**

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Einleitung

1.1 Motivation

Bustracker ist eine Fallstudie im Projekt „Digitale Kommune“ der Entwicklungsagentur Rheinland-Pfalz e.V. *Bustracker* soll die Möglichkeit demonstrieren in ländlichen Gebieten den Schulweg der eigenen Kinder zu beobachten.
In vielen Gemeinden müssen Schüler oder sogar Kindergartenkinder über weite Strecken mit dem Bus zu ihrer Betreuung oder Lerninstitution fahren. In den seltensten Fällen sind noch Betreuungspersonen im Bus vorhanden. Die gängige Praxis, die Kinder durch Eltern mit dem Kfz (Kraftfahrzeug) an die Schule/den Kindergarten zu bringen, führt zunehmend zu einer Verkehrsüberlastung in den Bereichen um die Betreuungs- bzw. Lernstätte. Die Eltern versuchen, teilweise unter Ignoranz der Straßenverkehrsordnung, den Weg zum Bus oder zur Schule zu verkürzen. Damit geht eine erhöhte Gefahr für die Kinder einher, was die Eltern eigentlich durch den Transport per Kfz zu vermeiden versuchen.
Bustracker soll Eltern die Möglichkeit geben ihre Kinder guten Gewissens mit dem Bus fahren zu lassen. Plötzlich auftretende Ereignisse, wie z. B. eine Panne am Bus oder eine Straßensperrung können früher erkannt werden, da die Eltern mit *Bustracker* den Schulweg ihres Kindes verfolgen können. Dadurch kann früher reagiert und eine andere Transportmöglichkeit organisiert werden. Die Eltern wissen jederzeit wo sich ihr Kind befindet.
Bustracker implementiert verschiedene Technologien und dient zum Testen und Entwickeln des Konzeptes.

1.2 Überblick

Im Kapitel **Verwendete Technologien** wird auf die in diesem Projekt verwendeten Softwaretechnologien eingegangen. Die verwendeten Werkzeuge sind ebenfalls in diesem Kapitel beschrieben.
Das Kapitel **Organisation** beschreibt die Art und Weise wie die Entwicklung organisiert wurde. Der komplette Verlauf ist dort ebenfalls beschrieben.
Frontend erklärt den Aufbau und die Anwendung der in dieser Arbeit entwickelten App.
Backend enthält die Erklärungen und Beschreibungen zu den einzelnen Teilen der Software.
Im Kapitel **Fazit und Résumé** wird das Projekt kurz zusammengefasst. Vor- und Nachteile sowie gesammelte Erfahrungswerte werden im Kapitel **Lessons learned** besprochen. In diesem Kapitel werden ebenfalls Lösungsstrategien für die während der Entwicklung aufgetretenen Probleme dargestellt. Im Anschluss daran wird im **Ausblick** weitere Verbesserungsvorschläge oder weitere Modifikationen der App besprochen.

Kapitel 2

Verwendete Technologien

Bustracker ist in Form einer Client-Server Architektur realisiert. Die Idee Positionsdaten über mehrere Geräte zu verteilen macht es erforderlich diese Daten zentral zu halten. Bei *Bustracker* kommt ein Datenbankserver zum Einsatz. Dieser nimmt die Positionsdaten der Clients im Tracking-Modus entgegen und liefert die Daten an die Clients im Watch-Modus. Als Clients werden die Endgeräte der Nutzer bezeichnet. Die Clients kommunizieren per REST - API (REST (Representational State Transfer) API (Application Programming Interface)) mit der Datenbank. In Abbildung 2.1 ist die Struktur dargestellt.

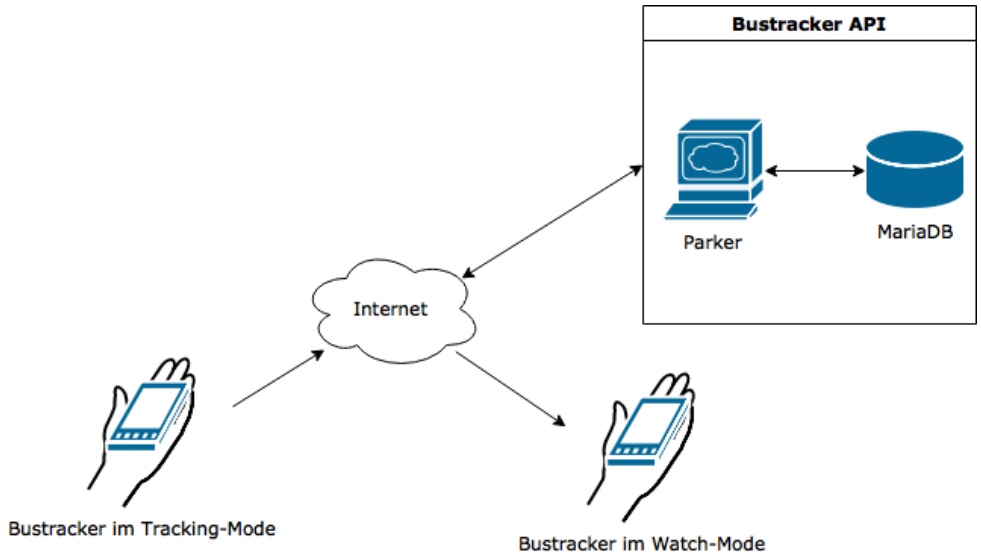

Abbildung 2.1: *Äußere Architektur von* Bustracker

Die Architektur der App *Bustracker* selbst ist in Abbildung 2.2 dargestellt. Die App ist eine „Single Page Application“. Der interaktive Teil ist als HTML (Hypertext Markup Language)-Komponente mittels <ion-app> - Tag eingebunden. An dieser Stelle wird

das „Root-Element“der Komponente geladen. Die Komponente besteht aus *Declarations*, also den Seiten und den *Providers*, diese enthalten die Programmlogik.

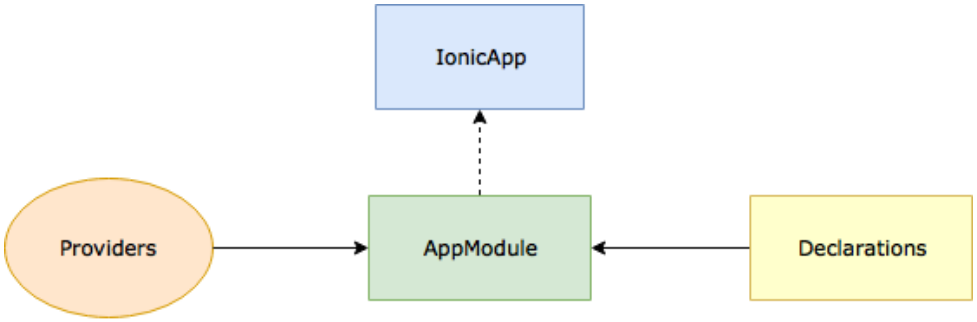

Abbildung 2.2: *Innere Architektur von* Bustracker

Um *Bustracker* auf verschiedenen Platformen einsetzen zu können, muss es vermieden werden, native Funktionen bestimmter Geräte oder Betriebsysteme direkt zu verwenden. Für diesen Prototypen wurde Angular 5, ein Framework zur Erstellung von Single Page Applications, eingesetzt. Um die nativen Gerätefunktionen, wie z. B. GPS (Global Positioning System) oder Bluetooth in *Bustracker* zu nutzen wird Cordova verwendet. Bei der Entwicklung kommt Ionic 4 als übergeordnetes Framework zum Einsatz. Ionic verwendet verschiedene Befehle und Konzepte von Angular und Cordova bereits implizit. Die Entwicklungssprache ist TypeScript, eine Spracherweiterung von Javascript, mit zusätzlicher Syntax und einer Typisierung.

2.1 Ionic 4

Die Auswahl des Frameworks begründet sich auf die Antworten folgender Fragen:

In welchen Sprachen ist der/die Entwickler erfahren?
Der Entwickler hat Erfahrung in Java und TypeScript.

Ist es eine native Anwendung oder ist sie webbasiert?
Die Anwendung soll platformübergreifend funktionieren und die nativen Funktionen des jeweiligen Endgeräts unterstützen. Die Anwendung ist grundsätzlich webbasiert, jedoch ist dies für den Nutzer nicht offensichtlich.

Welche Funktionalitäten werden benötigt und sind diese abgedeckt?
Die benötigten Funktionen sind GPS, Bluetooth, iBeacon, Festspeicher und HTTP-Kommunikation (HTTP (Hypertext Transfer Protocol)). Die Funktionen sind über die *native Plugins* abgedeckt.

Werden GUI-Elemente geboten?
Ionic bietet eine Vielzahl von GUI-Elementen für iOS und Material Design. Diese umfassen vor Allem Interaktions- und Anzeigeelemente. [3]

Sind Schnittstellen zu z. B. Datenbanken vorhanden ?
Verschiedene Schnittstellen können via Plugin nachinstalliert werden.

Ist das Framework zukunftsfähig?
Ja, das Ionic-Framework wird kontinuierlich weiterentwickelt. Dies lässt sich hervorragend am sogenannten „Pulse“des *Ionic*-Repositories ablesen. [4]

Entstehen Kosten für das Framework?
Ionic unterliegt der sogenannten MIT-Lizenz [5] und darf privat sowie kommerziell kostenlos verwendet werden.

Basierend auf den Antworten, wurde das Ionic-Framework ausgewählt. In einem vorangegangenen Praxisprojekt [6] wurde Ionic bereits erfolgreich verwendet.
Ionic 4 ist ein frei verfügbares Open-Source SDK zur Entwicklung von nativen und progressiven Web-Apps. Der besondere Fokus des Frameworks liegt auf mobilen Endgeräten. So kann eine Anwendung gleichzeitig für iOS-, Android- und Windows Phone Geräte entwickelt werden. Entwickelt werden die Apps auf Basis von HTML 5, CSS (Cascading Style Sheets) 3, Angular 5 und TypeScript. Um auf den einzelnen Geräten die integrierten Hardwarefunktionen nutzen zu können wird Apache-Cordova verwendet. Ionic-Native, ein TypeScript Wrapper für Cordova Plugins wird verwendet, um native Funktionen einfach in die Applikation einzubinden.
Das Ionic Framework steht unter der MIT-Lizenz [5], wodurch es sowohl privat als auch geschäftlich kostenlos verwendet werden kann.

2.2 Angular 5

Angular ist ein Web-Framework zur Erstellung von Single-Page Applications (SPA). Es basiert auf TypeScript. Angular verwendet das MVVM-Pattern (Model-View ViewModel). Die Komponente (Component) entspricht hier dem ViewModel, welches die UI-Logik enthält. Sie tauscht Daten mit dem Model aus und stellt Methoden und Dienste bereit. Die View wird durch Databinding an das ViewModel gebunden und ist somit einfach austauschbar. Sie wird unter Verwendung von HTML 5 und CSS 3 erzeugt. Das Model enthält alle Inhalte, die durch den Benutzer angezeigt und verändert werden können. Angular ist seit November 2017 in der Version 5 erhältlich. Das Framework ist

Open-Source und Google ist an der Entwicklung beteiligt. Unit-Tests und End-to-End Tests werden von Angular unterstützt. [7]

2.3 TypeScript

TypeScript ist eine freie Open-Source Programmiersprache, die von Microsoft entwickelt wird [8]. Bei der Programmiersprache handelt es sich um eine typisierte Obermenge von JavaScript. Sie basiert auf den ECMAScript (European Computer Manufacturers Association Script) 6 Standard und erweitert die JavaScript Sprache mit einer statischen Typisierung, wie in der Abbildung 2.3 zu sehen ist. Um die Kompatibilität zu gewährleisten, ist JavaScript Code gültiger Typescript Code. Ein Browser muss mindestens ECMAScript 5 Fähigkeiten haben, damit Typescript ausführbar ist.

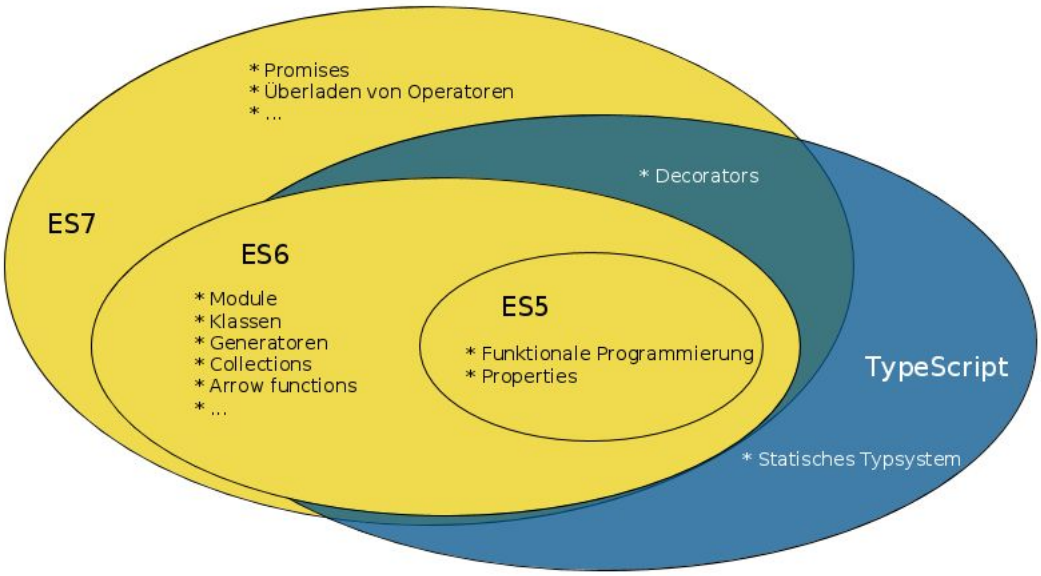

Abbildung 2.3: *TypeScript Sprach-Hierachie [1]*

2.4 Cordova

Apache Cordova ist ein Open Source Framework zur Entwicklung für mobile Plattformen. Mit Cordova ist es möglich moderne Web-Technologien, wie HTML 5, CSS 3 und JavaScript (hier TypeScript) für plattformübergreifende Entwicklung zu nutzen. Um auf die Sensoren und Daten der einzelnen Geräte zuzugreifen werden Standard APIs der einzelnen Betriebssysteme verwendet. Cordova dient als Mittelschicht zwischen Anwendung und Gerät [2]. Abbildung 2.4 zeigt den allgemeinen Aufbau der

Architektur.

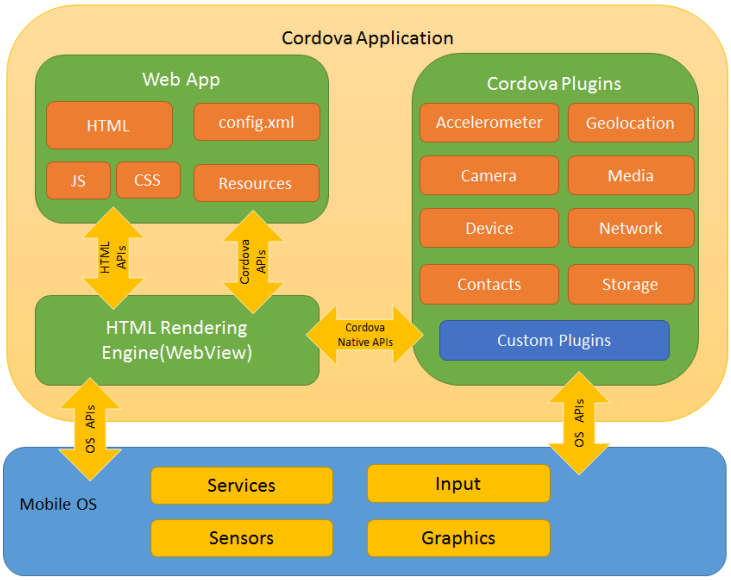

Abbildung 2.4: *Architektur von Cordova [2]*

2.5 iBeacon

Sogenannte iBeacons sind Bluetooth Low Energy Sender, die nach einem proprietärem von Apple spezifizierten Protokoll, dem iBeacon Protokoll [9], arbeiten. Die Idee hinter der Entwicklung der iBeacons war die Durchführung von Lokalisation innerhalb von Gebäuden. Beacons eignen sich aber auch für den Gebrauch im Freien, damit können unabhängig des GPS-Signals Ereignisse, wie das Erreichen oder Verlassen eines bestimmten Bereichs detektiert werden. Abbildung 2.5 zeigt eine schematische Darstellung der Technologie. Zur Identifizierung eines einzelnen Beacons werden bei *Bustracker* vier Parameter verwendet, die nachfolgend erläutert werden:

UUID

Die UUID (Universally Unique Identifier) gibt in diesem Falle die sogenannte Region an. Eine Beacon Region ist keine Region im Sinne von einem durch geografische Merkmale begrenztem Bereich. Sie wird durch die Parameter UUID, Major und Minor charakterisiert. So wie ein einzelnes Beacon durch die gleichen Parameter gekennzeichnet ist. Die physische Repräsentation einer Beacon Region ist die Reichweite aller dieser Region zugeordneten Beacons. Mehrere iBeacons können die gleiche UUID haben. Alle Beacons mit der gleichen UUID gehören zur gleichen Region. [10]

Zum Beispiel könnten 100 Beacons mit der gleichen UUID einem Busunternehmen zugeordnet sein.

Major

Der Major-Parameter unterteilt die Beacons einer UUID. Zum Beispiel könnte dies eine Linie (Strecke) des Busunternehmens sein.

Minor

Der Minor-Parameter unterteilt eine Major-Gruppe in einzelne Beacons. Dies könnte einer Haltestelle auf einer Linie entsprechen.

Identifier

Identifier ist ein String um ein Beacon zu beschreiben. Zum Beispiel könnte das hier der Name der Haltestelle sein.

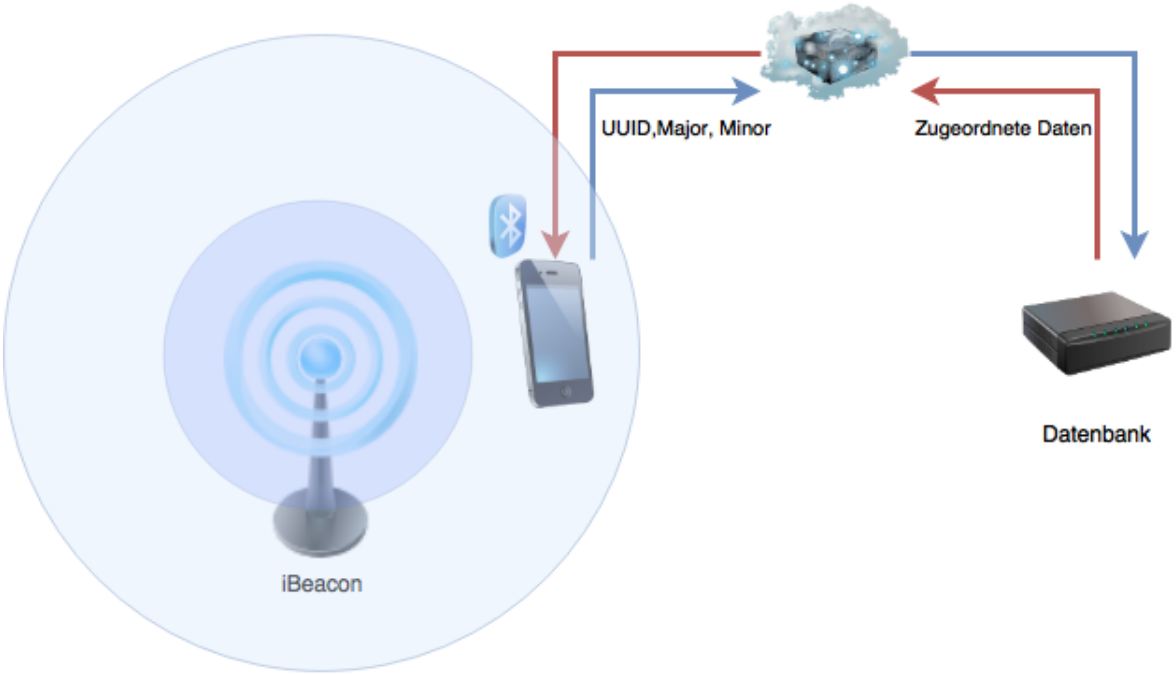

Abbildung 2.5: *Schematische Darstellung der iBeacon-Technologie*

Zur konkreten Anwendung kommt diese Technologie im iBeaconService.

2.6 REST

REST bedeutet Representational State Transfer und beschreibt eine Architektur. Eine REST API ist **statuslos**, d. h. es gibt keine Nutzersessions. Jede Anfrage erzeugt eine

neue Ressource, alle Daten werden nochmals generiert. Jede dieser Ressourcen ist über eine spezifische URI adressierbar. Ressourcen können in unterschiedlichen Repräsentationen vorliegen. Die gleiche Information kann in unterschiedlichen Formaten abgerufen werden, z. B. HTML und JSON (JavaScript Object Notation). [11]

2.7 Werkzeuge

2.7.1 WebStorm

Aus einem vorhergehenden Projekt war die integrierte Entwicklungsumgebung WebStorm bereits bekannt.
In WebStorm sind die Technologien HTML5, TypeScript, Angular und Cordova bereits integriert. Die IDE (Integrated Development Environment) lässt sich durch ein breites Angebot an Plugins erweitern. Webstorm basiert auf der IntelliJ IDEA Plattform und verwendet Konzepte, die bereits durch IDEs wie AndroidStudio bekannt sind. WebStorm ist auf JavaScript spezialisiert und bietet darauf zugeschnittene Funktionen wie z. B. Codevervollständigung, automatisches Importieren, Signaturinformationen bei Funktionsaufruf und kontextbezogene Hervorhebung von Syntax. Die Integration von Git/Github erwies sich ebenfalls als sehr hilfreich. Es wird kein weiteres Programm neben der IDE benötigt. Das Interface ist in Abbildung 2.6 zu sehen.

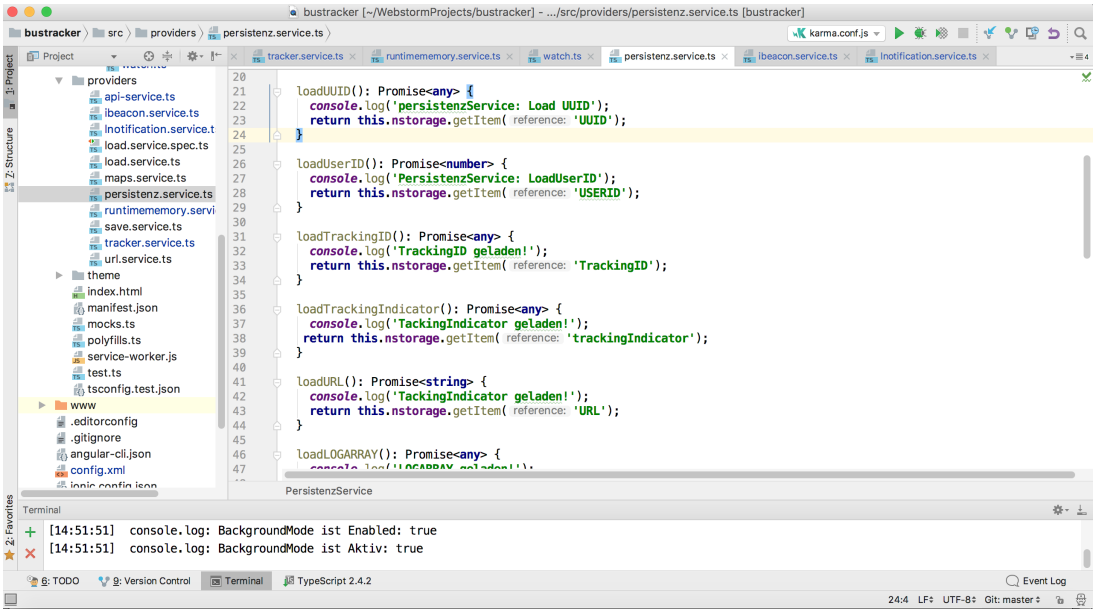

Abbildung 2.6: *Interface der WebStorm IDE*

2.7.2 compodoc

Um die Verständlichkeit und die Wartbarkeit von Software zu gewährleisten muss eine Dokumentation zum Programmcode existieren. Ein großer Teil der Dokumentation besteht aus Quellcodedokumentation. Der Dokumentationsprozess ist in der Regel aufwendig und bedeutet für die Entwickler zusätzliche Arbeit. Dies führt häufig zu einer lückenhaften und unvollständigen Dokumentation der Software. Um dem vorzubeugen gibt es Tools, die diesen Prozess vereinfachen. **Compodoc** ist ein solches Tool.
Compodoc ist kommandozeilenbasiert und erzeugt eine statische codebasierte Quellcodedokumentation von Angularanwendungen. Die Codedokumentation ist in der Struktur einer Website ausgeführt, sie umfasst eine aufbereitete Ansicht der vom Entwickler im Code eingefügten Kommentare, so dass diese nicht mehrmals geschrieben werden müssen. Es besteht die Auswahl zwischen verschiedenen Themes bzw. Templates. Die Templates sind bereits responsiv und unterstützen moderne Webtechnologien.
Diese „Website“ wird lokal gehostet und benötigt somit keinen externen Server. Eine mächtige Such-Engine namens „lunr.js“ sorgt dafür, dass die Dokumentation effizient durchsucht werden kann. Compodoc parsed den Code lediglich, es ist keine TypeScript-Kompilierung notwendig. Durch das Parsen enthält die Dokumentation alle Elemente der Anwendung automatisch. Sie sind im Inhaltsverzeichnis gelistet. Es besteht die Möglichkeit zu jeder Komponente den Quellcode direkt einzusehen. Kommentare die im JSDoc-Format [12] vorliegen, werden von Compodoc „verstanden“ und in die Codedokumentation mit aufgenommen. Zur Zeit kann Compodoc „@param, @returns, @link und @example“verarbeiten. Abbildung 2.7 zeigt den Quellcode und das Compodoc Ergebnis.

```typescript
/**
 * PersistenzService, dieser Dienst verwaltet den Native Storage des Smartphones.
 * Werte werden als key <-> Value Paare im Format STRING <-> JSONObject gespeichert.
 */
@Injectable()
export class PersistenzService {

  /**
   * Konstruktor
   *
   * @param {NativeStorage} nstorage Plugin Instanz des NativeStorage wird injeziert
   */
  constructor(private nstorage: NativeStorage) {
    console.log('Hello PersistenzProvider Provider');
  }
```

Description

PersistenzService, dieser Dienst verwaltet den Native Storage des Smartphones. Werte werden als key <-> Value Paare im Format STRING <-> JSONObject gespeichert.

Index

Methods

loadADDRESS	loadMajor	loadURL
loadFirstUse	loadTrackingCode	loadUserID
loadLat	loadTrackingID	loadUUID
loadLOGARRAY	loadTrackingIndicator	saveSingleParameter
loadLong	loadTrackMarkers	

Constructor

`constructor(nstorage: NativeStorage)`

Defined in src/providers/persistenz.service.ts:9

Konstruktor

Parameters :

Name	Type	Optional	Description
nstorage	NativeStorage		Plugin Instanz des NativeStorage wird injeziert

Abbildung 2.7: *TypeScript Quellcode und Compodoc Seite im Vergleich.*

Kapitel 3

Organisation

3.1 Workflow

Der bereits im Praxisprojekt [6] etablierte Workflow wurde zu einem großen Teil übernommen. Tägliche Treffen jedoch entfielen, da der Autor nicht in einem Team arbeitete. Anforderungen an die Software wurden als Stories in dem „Issues “-Tab des verwendeten GitHub Repositories als User Story eingetragen. Abbildung 3.1 zeigt eine solche Story.

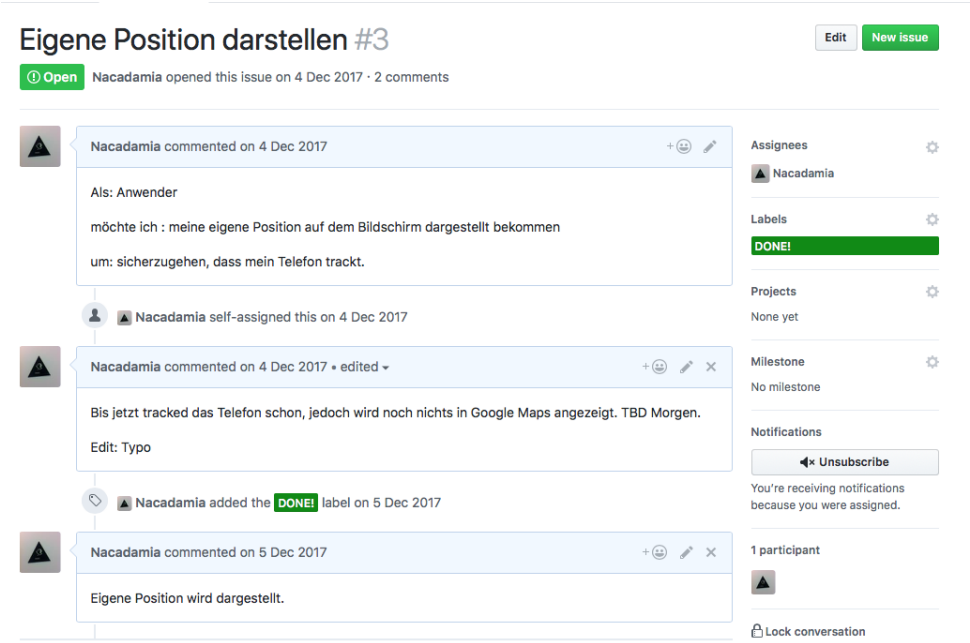

Abbildung 3.1: *Einzelne Story im Issues-Tab des* Bustracker *Repository*

Es wurden Stories aus dem in Abbildung 3.2 gezeigten IssusesTab gewählt. Nach Abschluss einer Story bzw. Implementierung eines Features, wurde diese mit dem Label

„Done“ versehen. Nach positiver Abnahme wurde die Story dann auf „closed“ gesetzt und somit geschlossen.

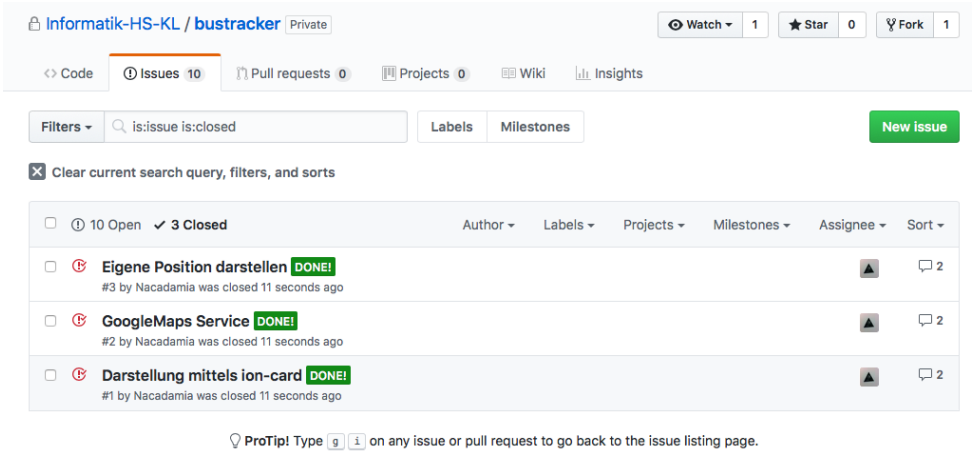

Abbildung 3.2: *Issues-Tab des Repository*

Um die Positionsdaten zu erfassen und im Anschluß wieder zu verteilen wird eine Datenbank benötigt. Die gewählte Datenbank ist eine sogenannte MariaDB [13]. Die gewählte Datenbankstruktur und die benötigten Daten sind in Abb. 3.3 dargestellt.

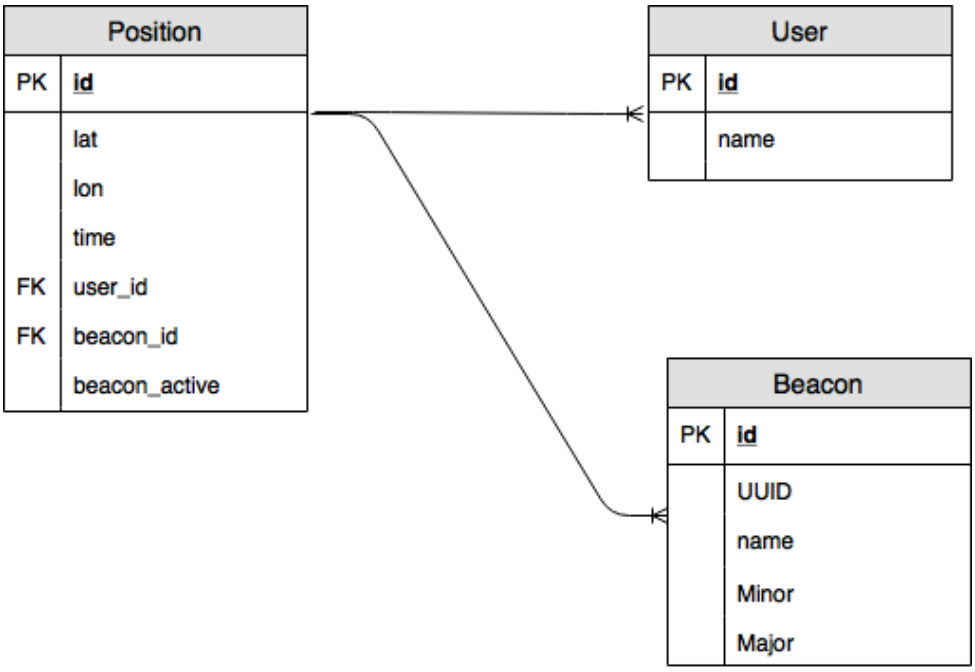

Abbildung 3.3: *ER-Diagramm der Bustracker Datenbank*

Die API-Entwicklung wurde extern durchgeführt. Die entstandene REST-API ist in [14] beschrieben.

3.2 Ablauf

Zu Beginn der Arbeit wurden die benötigten Funktionalitäten für *Bustracker* identifiziert. Diese wären zum einen die Möglichkeiten zur Positionsbestimmung CSS und iBeacons und zum anderen die Anzeige der zuvor bestimmten Position. Dies wurde mittels der Brainstorming Technik durchgeführt. (Abbildung 3.4).
Für den Prototypen kommt hier aufgrund der einfachen Implementierung und umfangreicher APIs GoogleMaps in Frage. Um den Zustand der App speichern zu können bot sich der Zugriff auf den Festspeicher des jeweiligen Endgerätes an. Um dem eigentlichen Zweck, die Mitteilung der eigenen Position an ein anderes Endgerät zu übertragen, zu entsprechen, müssen die Daten weitergeleitet werden. Dazu wird bei *Bustracker* eine REST - API eingesetzt.

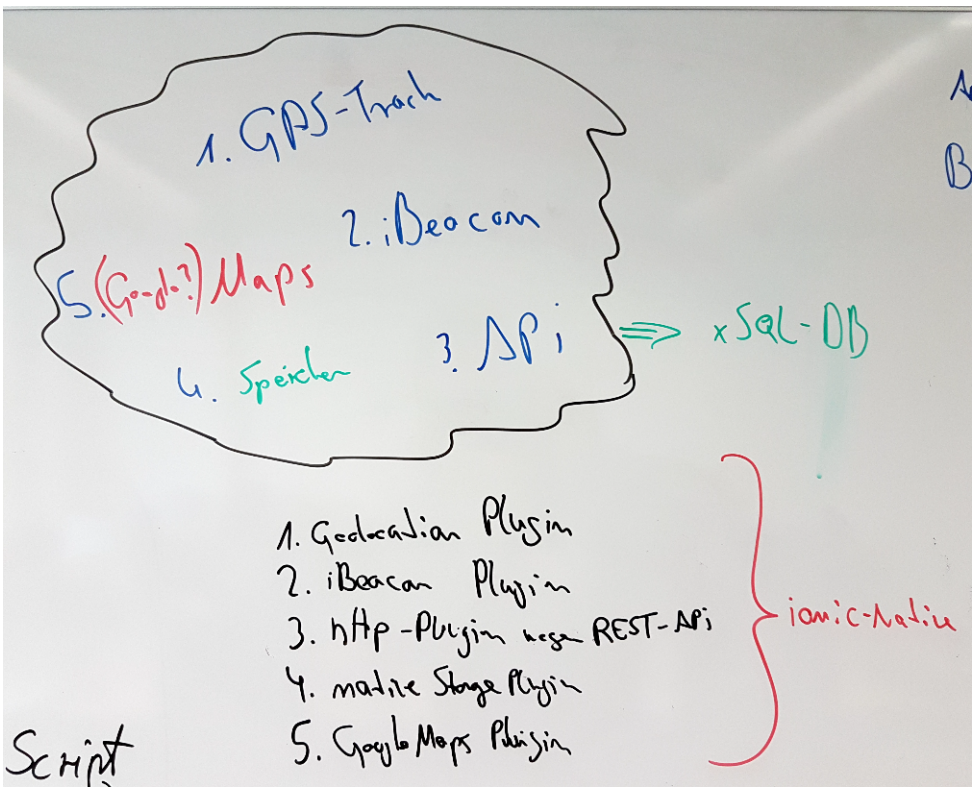

Abbildung 3.4: *Ergebnis des Brainstormings*

Ionic 4 bietet eine Reihe von Plugins an, mit denen die native Hardware des Gerätes plattformunabhängig angesprochen wird. Alle Funktionen können mittels solcher *native Plugins* abgebildet werden. Zur konkreten Implementierung siehe Kapitel Backend.
Im Anschluss an diesen Teil erfolgte die Implementierung von Teilfunktionen um mehr über das Verhalten der einzelnen Plugins herauszufinden. Parallel wurde das Ausse-

hen des GUI (Graphical User Interface) in sogenannten Scribbles skizziert. Ein Beispiel ist das Scribble der WatchPage in Abbildung 3.5.

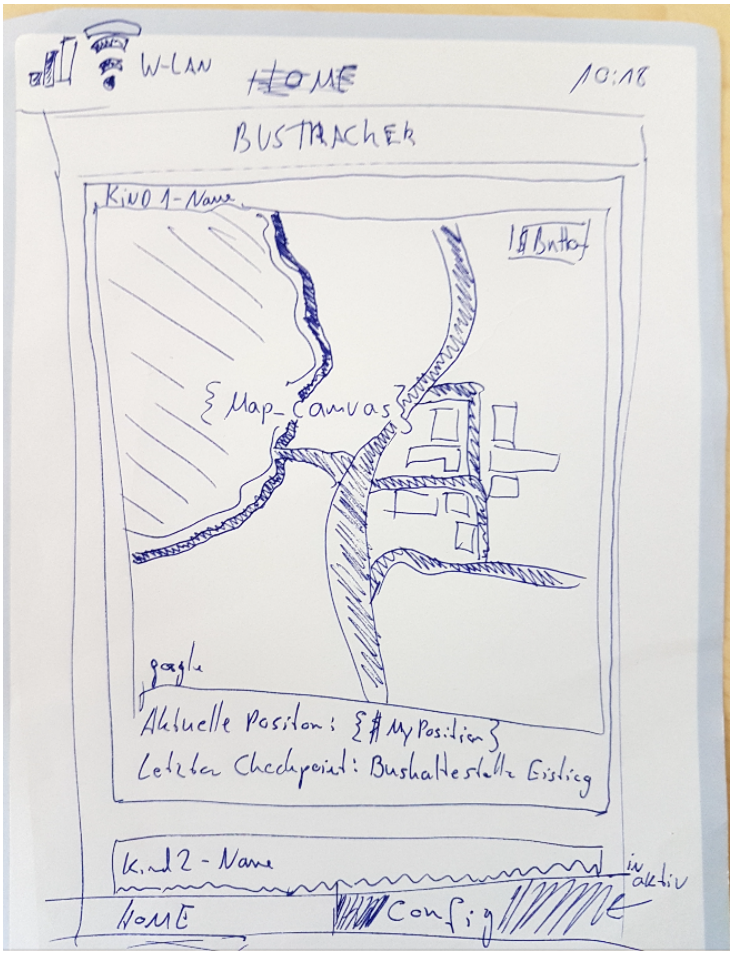

Abbildung 3.5: *Scribble der WatchPage*

Nach dem Vorbild der Scribbles wurden die Seiten gestaltet. Im Anschluss daran wurden die Komponenten ausgewählt, um die geplanten Funktionalitäten bereitzustellen. Diese Komponenten wurden eingebunden und damit der gewünschte Funktionsumfang geschaffen. Nachdem die Komponenten erfolgreich getestet wurden, erfolgte die externe Anbindung an die API. Am Ende wurde die Benutzeroberfläche einheitlich gestaltet.

Kapitel 4

Frontend

4.1 Aufbau

Die App gliedert sich in folgende Bereiche:

Watch Bereich
: Im Watch Bereich lädt die App regelmäßig Daten des zu beobachtenden Nutzers von einem Server.

Track Bereich
: Im Track Bereich führt die App regelmäßig Positionsbestimmungen durch und wartet auf das Entdecken bestimmter iBeacons durch das Gerät. So erkennt die App Checkpoints, in der Regel Haltestellen oder Ziele.

Konfiguration
: In der Konfiguration wird festgelegt, welcher Nutzer beobachtet wird. In Zukunft wird diese Konfiguration komplexer und umfangreicher werden.

4.2 Funktionsweise

Bei erstmaligem Start werden die sogenannten Permissions abgefragt. Der Nutzer muss der App den Zugriff auf Festspeicher, Bluetooth und GPS gestatten. Dann gelangt der Nutzer auf die HomePage. Hier wählt der Nutzer aus, ob er seinen Standort mitteilen oder ein anderes Gerät verfolgen möchte. Nur von der Homepage kommt der Nutzer auf die ConfigurationPage. Dort stellt der Nutzer seine eigene ID (die UserID), sowie die ID des zu beobachtenden Geräts (trackingID) ein, anschließend kehrt er auf die HomePage zurück. Abbildung 4.1 zeigt die App in Form eines Ablaufdiagramms.

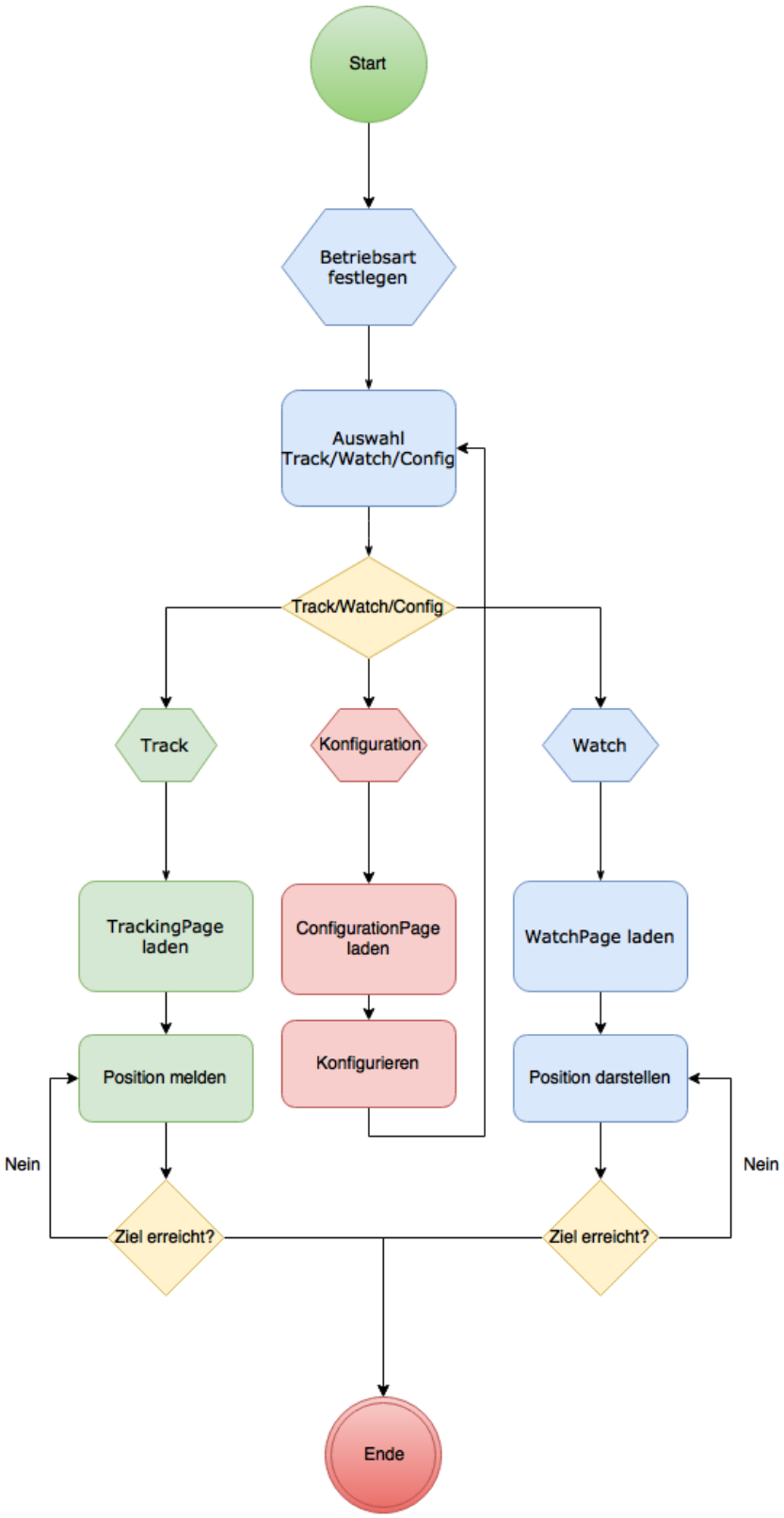

Abbildung 4.1: *Ablaufdiagramm* Bustracker

4.2.1 API

Das Akronym API steht für *Application Programming Interface* und bedeutet Programmierschnittstelle. Sie definiert in welcher Form Daten, in diesem Fall serverseitig, angenommen und bereitgestellt werden. Sie stellt definierte Funktionen für den Datentransfer bereit. Ein großer Vorteil einer API besteht darin, dass sich die interne Datenverwaltung der API ändern kann, ohne die Schnittstelle zu beeinflussen. Abbildung 4.2 zeigt den Kommunikationsablauf mit *Bustracker*. Die API ist nicht an die jeweilige Anwendung gebunden. Weitere Anwendungen könnten die API verwenden, um Daten bereitzustellen oder abzurufen. So wäre es denkbar, Skills für Sprachassistenten zu entwickeln, die auf die API Daten zurückgreifen.

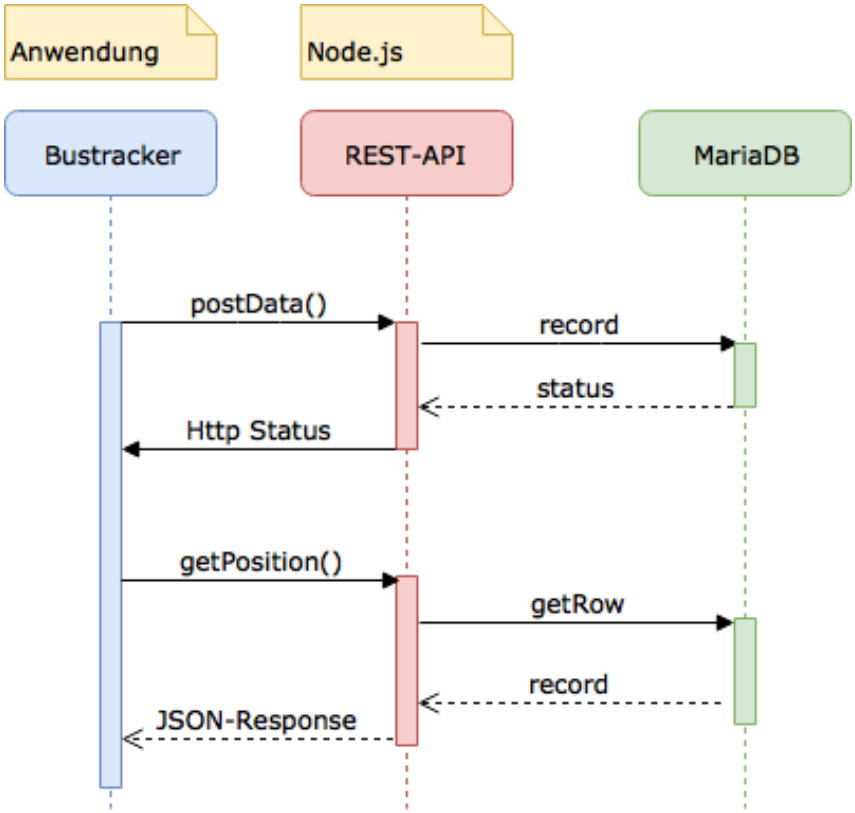

Abbildung 4.2: Bustracker *API Sequenzdiagramm*

Bustracker kommuniziert mit einer REST - API (siehe Kapitel 2.6), dies ist eine gängige Form für APIs. Zum Zeitpunkt der Entwicklung befand sich der Server innerhalb des hochschulinternen Netzwerks und war von außen nur per VPN (Virtual Private Network) zu erreichen. In einer Produktivumgebung wird der Server direkt erreichbar sein, dann mit einer Nutzerauthentifizierung.

4.2.2 Speicherstruktur

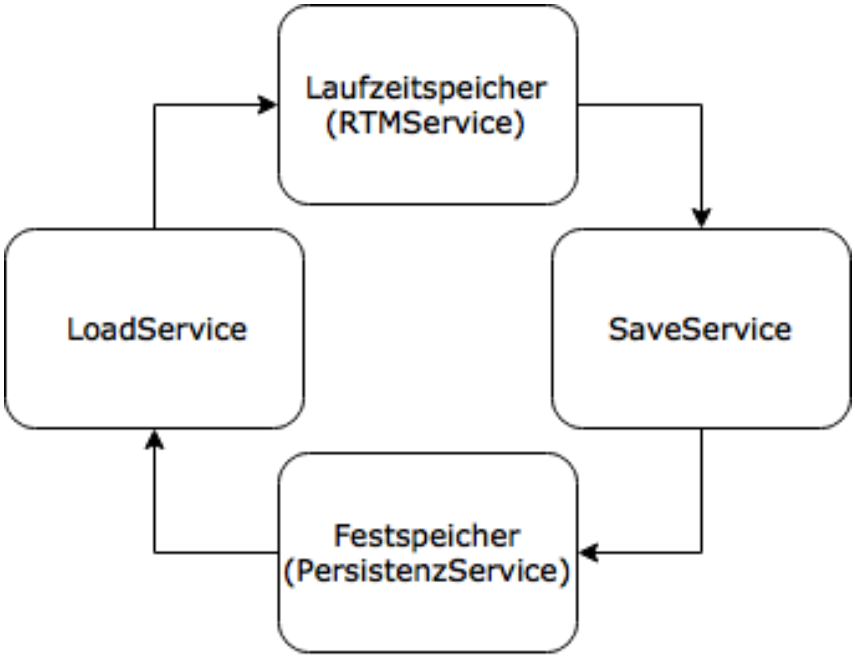

Abbildung 4.3: Bustracker *Speicherstruktur mit Datenflussrichtung*

Die Struktur des Speichers (siehe Abb. 4.3) wurde so gewählt, dass der Persistenz-Service keine anderen Injectables instantiiert. Der LoadService und der SaveService instantiieren den PersistenzService, der den Festspeicher des Geräts verwaltet. Load- und SaveService werden nur vom RTMService instantiiert. Daraus ergibt sich eine Struktur, die unanfällig für Zirkelbezüge ist. Um einen Zirkelbezug, wie in Kapitel 6 aus dem Praxisbericht [6] dargestellt zu vermeiden, wurde die Speicherstruktur direkt so ausgestaltet, dass sich keine Injectables gegenseitig instantiieren.

Kapitel 5

Backend

5.1 HomePage

Die Homepage erscheint zum Start von *Bustracker*, auf dieser Seite wird die Funktion gewählt. Je nach Auswahl wird die entsprechende Page der App geladen.

5.1.1 HTML

Die HTML-Seite in Abb. 5.1 zeigt zwei Buttons, der obere dient dazu das Tracking zu aktivieren und führt auf die TrackingPage. Der untere Button dient dazu, ein anderes Gerät zu verfolgen und führt auf die WatchPage. Am oberen rechten Rand befindet sich ein „Zahnrad“, bei einem Klick darauf erscheint die ConfigurationPage.

5.1.2 CSS

Listing 5.1 zeigt das Stylesheet für die Homepage. Die Anweisungen dienen dazu, den Seitenheader in dunkelblau (Hex: 0D47A1) und die Schrift Weiß (Hex: FFFFFF) zu färben. Der Seitenhintergrund wird mittels der Zeilen 3 - 5 in einem helleren Blau dargestellt (Hex: 1C4AFF).

5.1.3 TypeScript

Die TypeScript Datei **home.ts** enthält diejenigen Methoden, die nach Betätigen des jeweiligen Buttons die neue Seite aufrufen. Diese sind im Listing 5.2 zu sehen.

```
page-home {
  .scroll-content {
    background-color: #1C4AFF;
  }
  .toolbar-background {
    background-color: #0D47A1;
  }
  .toolbar-title {
    color: white;
  }
  .icon-only {
    background-color: #0D47A1;
  }
  .icon{
      color:white;
    }
  .icon-only{
    background-color: #0D47A1;
    color: #0D47A1;
    }
  .bar-buttons{
    background-color: #0D47A1;
    color: #0D47A1;
  }
}
```

Listing 5.1: *Stylesheet für die Homepage*

```
 goTracking(){
    this.navCtrl.push(TrackingPage).then(()=> console.log('TrackingPage
    aufgerufen'));
  }

  goWatch(){
    this.navCtrl.push(WatchPage).then(()=> console.log('WatchPage
    aufgerufen'));
  }

  goConfig(){
    this.navCtrl.push(ConfigPage).then( ()=> console.log('ConfigPage
    aufgerufen'));
  }
```

Listing 5.2: *Methoden zur Navigation*

```
1 <ion-item>
2    <ion-label color="primary" fixed> Eigene ID</ion-label>
3    <ion-input type="number" [(ngModel)]="this.rtm.USER_ID" placeholder="
     Hier ihre eigene ID"></ion-input>
4  </ion-item>
```

Listing 5.3: *Input mittels Databinding*

5.2 ConfigurationPage

Die Konfigurationsseite (siehe Abb. 5.1) bietet die Möglichkeit die eigene NutzerID einzustellen. Weiterhin kann die ID des zu trackenden Nutzers eingestellt werden. Die API-Aufrufe erfolgen auf der Tracking Seite mit der eigenen ID. Auf der Watchseite wird die ID des zu trackenden Nutzers verwendet.

5.2.1 HTML

Auf der Configuration Page sind zwei Inputs zu sehen. Der erste ist dazu gedacht, die eigene NutzerID einzugeben. Der zweite Input dient zum Eingeben des zu beobachtenden Nutzers. Es existiert eine API Beschränkung. Die IDs müssen existieren, sonst führt dies zu Fehlern bei der Abfrage. In Listing 5.3 ist die dynamische Datenbindung zu sehen. Der Nutzer manipuliert den Input, direkt nach der Manipulation wird der Wert in den RTMService gespeichert. Beim Betreten der Seite wird die jeweilige ID direkt aus dem RTMService geladen.

5.2.2 CSS

Listing 5.4 zeigt das Stylesheet für die Homepage. Die Anweisungen dienen dazu, den Seitenheader in dunkelblau (Hex: 0D47A1) und die Schrift Weiß (Hex: FFFFFF) zu färben. Die Zeilen 12 - 15 dienen dazu, den „Zurück“-Pfeil in Weiß einzufärben. Der Seitenhintergrund wird mittels der Zeilen 3 - 5 in einem helleren Blau dargestellt (Hex: 1C4AFF).

5.2.3 TypeScript

In der TypeScript Datei wird der RTMService im Konstruktor instantiiert um das sogenannte *two-way-databinding* zu ermöglichen. Siehe Listing 5.5

```
page-konfiguration {

.scroll-content {
background-color: #1C4AFF;
}
.toolbar-background {
background-color: #0D47A1;
}
.toolbar-title {
color: white;
}
.bar-button-default-md, .bar-button-clear-md-default, .
    bar-button-md-default
{
color: white;
}
}
```

Listing 5.4: *Stylesheet für die ConfigurationPage*

```
 constructor(public navCtrl: NavController, public navParams: NavParams,
   private rtm: RTMProvider) {}
```

Listing 5.5: *Konstruktor ConfigurationPage*

5.3 TrackingPage

Die TrackingPage (Abb. 5.1) zeigt die Eigene Position als LatLong Koordinate und als Adresse an. Zusätzlich wird der letzte Checkpoint angezeigt. Über einen Schalter kann der Nutzer das Tracking aktivieren oder deaktivieren.

5.3.1 HTML

Die HTML-Seite der TrackingPage zeigt oben einen Schalter zum Aktivieren oder Deaktivieren des Trackings. Darunter befindet sich ein Feld mit dem Längen- und Breitengrad, sowie die, sich aus dieser Koordinate ergebende Adresse. Die Daten werden via Databinding aus dem RTMService bezogen. Listing 5.6

```
<ion-item>
 {{this.rtm._config.lat}}:Latitude {{this.rtm._config.lng}}:Longitude
 Strasse: {{this.rtm.address.thoroughfare}} Nr: {{this.rtm.address.
    subThoroughfare}}
  PLZ: {{this.rtm.address.postalCode}} Ort: {{this.rtm.address.locality}}
</ion-item>
```

Listing 5.6: *Adresse via Databinding*

```
1  page-tracking {
2  .scroll-content {
3  background-color: #1C4AFF;
4  }
5  .toolbar-background {
6  background-color: #0D47A1;
7  }
8  .toolbar-title {
9  color: white;
10 }
11 .bar-button-default-md, .bar-button-clear-md-default, .
       bar-button-md-default
12 {
13 color: white;
14 }
15 }
```

Listing 5.7: *Stylesheet für die TrackingPage*

```
1  ionViewDidLoad() {
2      console.log('ionViewDidLoad TrackingPage');
3      (this.rtm.trackingIndicator.valueOf()== true) ? this.trackerService.
       startTracking() : this.trackerService.stopTracking();
4      console.log(this.rtm.trackingIndicator.valueOf());
5      this.beaconService.initBeacon();
6  }
```

Listing 5.8: *ionViewDidLoad()-Methode der TrackingPage*

5.3.2 CSS

Listing 5.7 zeigt das Stylesheet für die Homepage. Die Anweisungen dienen dazu, den Seitenheader in dunkelblau (Hex: 0D47A1) und die Schrift Weiß (Hex: FFFFFF) zu färben. Die Zeilen 12 - 15 dienen dazu, den „Zurück“-Pfeil in Weiß einzufärben. Der Seitenhintergrund wird mittels der Zeilen 3 - 5 in einem helleren Blau dargestellt (Hex: 1C4AFF).

5.3.3 TypeScript

Die TrackingPage instantiiert den RTMService, den TrackerService und den iBeacon-Service, um auf Werte zugreifen zu können, sowie Methoden der Dienste zu verwenden.
Nachdem die Seite vollständig geladen ist, wird die Funktion *ionViewDidLoad()* ausgeführt. Innerhalb dieser Funktion wird der Status des „trackingIndicators “abgefragt, basierend darauf wird das Tracking gestartet oder gestoppt. Dazu kommt eine If-Abfrage in Form eines bedingten ternären Operators zum Einsatz [37]. Siehe Listing 5.8.

Im Anschluss wird der iBeaconService gestartet. Beim Verlassen der Seite wird automatisch der Inhalt vom RTMService auf den Festspeicher übertragen.

5.4 WatchPage

Die WatchPage zeigt die Position des beobachteten Geräts auf einer Karte, als LatLong-Koordinate und als Adresse an.

5.4.1 HTML

Die WatchPage (Abb. 5.1) zeigt eine GoogleMaps-Karte mit Positionsinformationen des zu beobachtenden Nutzers an. Die Karte stellt der MapsService bereit. Sie wird an das *div* mit dem Namen „map_canvas“ gebunden. Zusätzlich ist die aktuelle Adresse der Position und der letzte Checkpoint zu sehen. Adresse und Checkpoint werden hier analog zur TrackingPage vom RTMService zur Verfügung gestellt und angezeigt.

5.4.2 CSS

Die Google Karte benötigt einen transparenten Hintergrund und ein benanntes . Hier wird die Karte in einer sogenannten ion-card angezeigt, einem Template des Ionic-Frameworks. Mittels CSS wird diese Card als transparent gekennzeichnet. Listing 5.9 zeigt die Style-Konfiguration für das und die Klasse „transparent-card“. Zusätzlich sind die Stylingoptionen für sichtbare CSS-Elemente eingetragen, analog zu den anderen Seiten um das Erscheinungsbild möglichst einheitlich zu gestalten.

5.4.3 TypeScript

Die WatchPage verfügt, analog zur TrackingPage, über eine *ionViewDidLoad()*-Methode. In dieser wird der MapsService initialisiert. Die Seite bietet verschiedene Methoden zum Abrufen und Zeichnen der Position des gewünschten Teilnehmers. Listing 5.10 zeigt die Methode zum Abrufen der letzten Position eines Teilnehmers.

```
page-watch {
  #map_canvas {
    height: 90%;
  }
  .transparent-card{
    background-color: transparent;
  }
  .scroll-content {
  background-color: #1C4AFF;
  }
  .toolbar-background {
  background-color: #0D47A1;
  }
  .toolbar-title {
  color: white;
  }
  .bar-button-default-md, .bar-button-clear-md-default, .
   bar-button-md-default
  {
  color: white;
  }
}
```

Listing 5.9: *Transparentes Canvas und Styling der WatchPage*

```
getSinglePos(){
  this.apiService.getLastLocationById(this.rtm.trackingID).then(
    (data)=> {
    this.rtm.lat = data.lat;
    this.rtm.long = data.lon;
    this.trackerService.geoCode();
  }).catch((err)=> {console.log('Fehler getSinglePos: ' + JSON.stringify(
   err))});
}
```

Listing 5.10: *Methode zum Abruf der letzten Position*

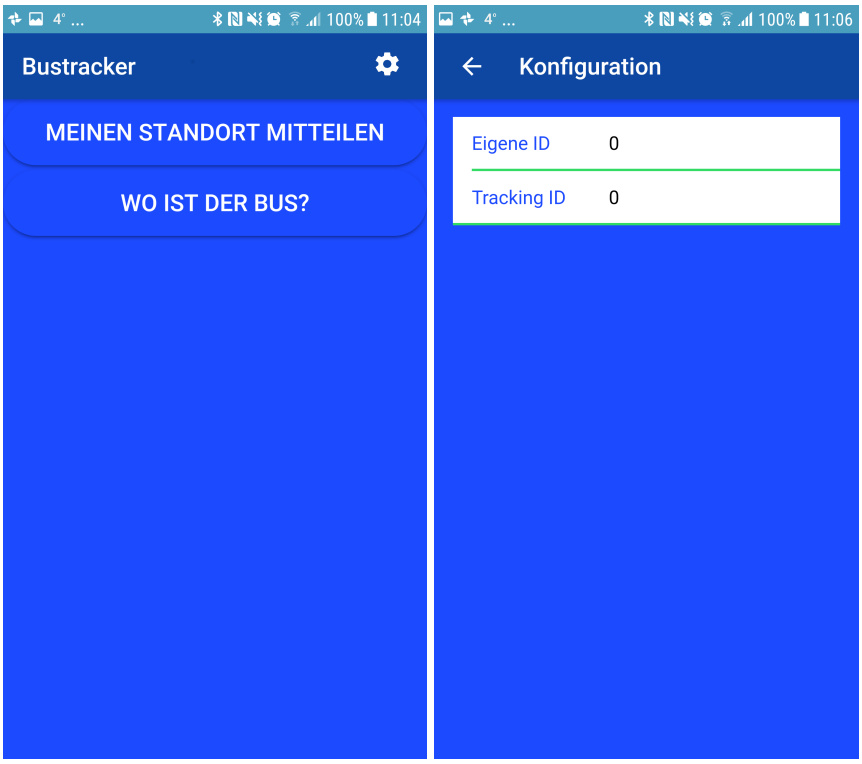

Abbildung 5.1: *Home- und Configpage von* Bustracker

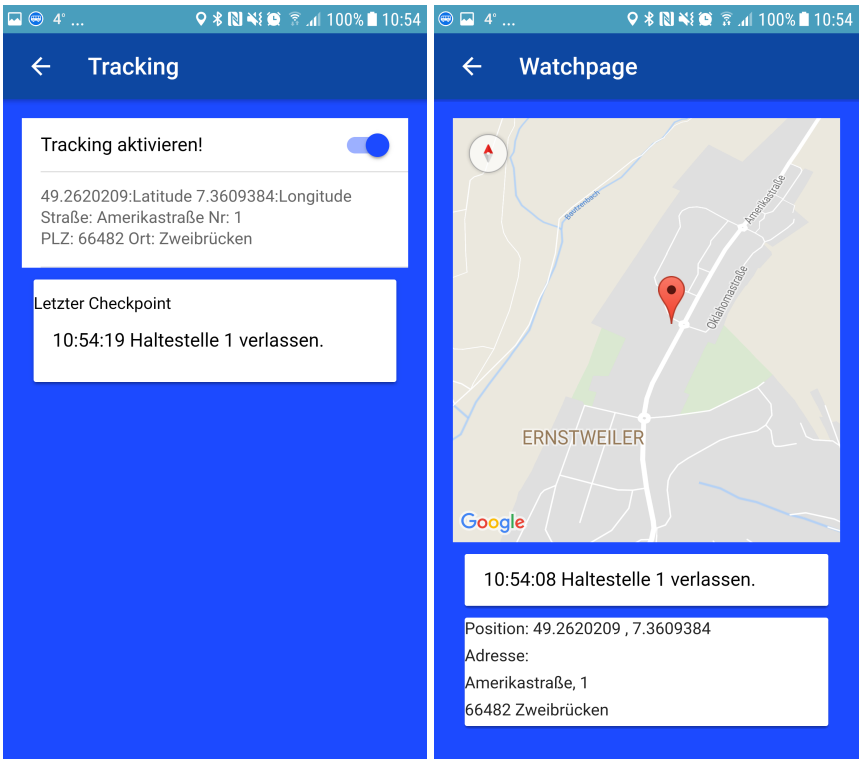

Abbildung 5.2: *Tracking- und Watchpage von* Bustracker

```
getAllDataById(id: number) {
    console.log('GET URL: ' + this.rtm.URL + '/' + id);
    this.http.get(this.rtm.URL + '/' + this.rtm.trackingID, {headers: this.
    headers}).subscribe(
       (data) => {
         data['result'].forEach((itr) => {
           this.rtm.trackPoints.push(itr);
           console.log(JSON.stringify(itr))
         });
         this.apiCallID = data;
         },
       (err) => {
         console.log('Fehler beim get-Request' + JSON.stringify(err));
       });
    }
```

Listing 5.11: *Rückgabe aller gültigen Positionsdaten der jeweiligen TrackingID*

5.5 APIService

Der API Service übernimmt die Kommunikation mit der REST-API von *Bustracker*.

5.5.1 Verwendete Plugins

@angular/common/http [15]

5.5.2 Arbeitsweise

Der Service stellt eine **POST** Methode bereit, mit der die Daten des Mobilgeräts an die API gesendet werden. Diese Funktion kommt beim Tracking 5.13 zum Einsatz. Das Abrufen der Daten geschieht mittels verschiedener **GET** Methoden. Es können alle gespeicherten oder die letzte gespeicherte Position eines mittels **ID** festgelegten Teilnehmers abgerufen werden. Listing 5.11.

5.6 iBeaconService

Der iBeaconService verwaltet Events, die durch das Betreten oder Verlassen des Sendebereichs eines iBeacons ausgelöst werden.

5.6.1 Verwendete Plugins

@ionic-native/ibeacon [16]

5.6.2 Arbeitsweise

Bustracker hört auf zwei Ereignisse, das Betreten und Verlassen des Sendebereichs eines Beacons. Die Beacons sind zwar aktive Sender, jedoch ändern diese ihr Verhalten nicht wenn ein Ereignis eintritt. Vereinfacht kann man sagen, iBeacons transferieren ortsbezogene Ereignisse in eine Art, die durch Software erfassbar ist und somit auf diese Ereignisse reagiert werden kann. Befindet sich ein iBeacon in Reichweite des Gerätes, so wird dieses erkannt und mit bekannten iBeacons verglichen. Im Erfolgsfall, also das Beacon ist der Software bekannt, wird eine Benachrichtigung auf dem Gerät angezeigt, die Checkpointanzeige aktualisiert und der Checkpoint der API mitgeteilt. Beim Verlassen des Sendebereichs wird ebenfalls eine Meldung (siehe Local Notification Service) angezeigt, der Checkpoint Status (siehe WatchPage) aktualisiert und die API (siehe APIService) wird informiert.

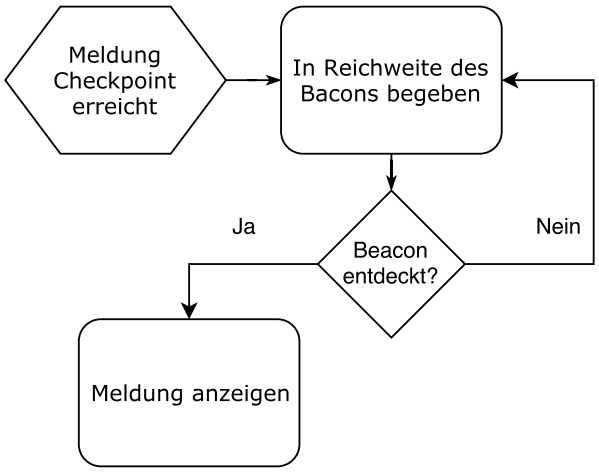

Abbildung 5.3: *Einsatz eines iBeacons als Flussdiagramm*

Der iBeaconService stellt die Funktionen zum „Hören“auf iBeacons bereit. Zuerst wird ein Delegat [17] für den nativen LocationManager erzeugt. Im Anschluss wird sich bei den benötigten EventListenern des Delegaten registriert. Die EventListener sind als Observeables ausgeführt, somit ist es einfach Zustandsänderungen zu detektieren. Listing 5.12 zeigt eine solche Registrierung und die im Erfolgsfall (also hier das Verlassen einer Region) auszuführenden Funktionsaufrufe.
Die zu entdeckenden Regionen müssen mitgeteilt werden. Dazu verwendet man eine Methode der lokalen Instanz des Plugins. Diese Methode *startMonitoringForRegion(beacon : BeaconRegion)* ist als Promise ausgeführt, da sie asynchron ist. Es ist möglich, den Erfolgs- oder Fehlerfall abzufragen, so dass es eine Mitteilung gibt, wenn die Region nicht beobachtet werden kann.

```
delegate.didExitRegion()
      .subscribe(
      (data) =>{
          this.notificationService.updateWaypoint(data.region.identifier);
        console.log('Gebiet verlassen: ', data.region.identifier);}
      );
```

Listing 5.12: *Registrieren beim didExitRegion-EventListener*

```
let beacon1 = this.iBeacon.BeaconRegion('Haltestelle 1' , this.rtm.UUID,
    10001, 10002);
.
.
.
this.iBeacon.startMonitoringForRegion(beacon1)
       .then(
         (data) => console.log('Native layer received the request to
    monitoring' + JSON.stringify(data)),
          error => console.error('Native layer failed to begin monitoring: '
    , JSON.stringify(error))
       );
```

Listing 5.13: *Übergabe einer Region an das iBeacon Objekt.*

5.7 Local Notification Service

Der Local Notification Service dient dazu Statusinformationen über die jeweiligen Checkpoints als Mitteilung auf dem Bildschirm oder im Notificationcenter eines Smartphone anzuzeigen.

5.7.1 Verwendete Plugins

@ionic-native/local-notification [18]

5.7.2 Arbeitsweise

Die Nachricht enthält die Uhrzeit und den Checkpoint sowie den Status erreicht bzw. verlassen. Abbildung 5.4 zeigt eine solche Notification.

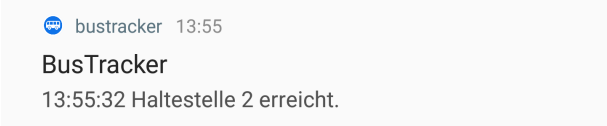

Abbildung 5.4: *Benachrichtigung im Notification Center*

```
1  sendWaypointNotification(identifier: string ){
2      let text: string = '';
3      text = this.rtm.timeHelper(); switch(identifier){
4        case 'Haltestelle 1':{ text = text + ' Haltestelle 1 erreicht.';
5                               this.rtm.LOGARRAY.push(text);
6                               break;}
7        case 'Haltestelle 2':{ text = text + ' Haltestelle 2 erreicht.';
8                               this.rtm.LOGARRAY.push(text);
9                               break;}
10       case 'Haltestelle 3':{ text = text + ' Haltestelle 3 erreicht.';
11                              this.rtm.LOGARRAY.push(text);
12                              break; }
13       default: {text = 'Ein Beacon wurde nicht zugeordnet.'};
14
15     }
16
17     this.notification.schedule({id: 1, title:'BusTracker', text: text, led:
       '#FF0000'});
18   }
```

Listing 5.14: *Erzeugen einer Notification bei Erreichen einer Haltestelle*

Das Erzeugen einer Notification ist in Listing 5.14 dargestellt. Basierend auf dem Namen des iBeacons wird die Nachricht generiert. Die *.schedule()*-Methode lässt die Nachricht im Notification Center des Telefons erscheinen.
Listing 5.15 zeigt den Aktualisierungsprozess der Notification. Basierend auf dem Beaconnamen wird die Notification aktualisiert.

5.8 LoadService

Der LoadService dient zum Laden gespeicherter Werte vom Festspeicher des Telefons in den Laufzeitspeicher RTMService.

5.8.1 Verwendete Plugins

keine

5.8.2 Arbeitsweise

Die Methoden des LoadService rufen Methoden des PersistenzService auf. Die Rückgabewerte des LoadService sind als Promises ausgeführt, da die Festspeicherzugriffe asynchron erfolgen.
Listing 5.16 dient als Beispiel und zeigt die Methode um die UserID zu laden. Sie ruft eine Methode des PersistenzService auf, wartet auf das Ergebnis und gibt dieses dann

```
updateWaypoint(identifier : string){
    let text: string = '';
    text = this.rtm.timeHelper();
    switch(identifier){
      case 'Haltestelle 1':{ text =  text + ' Haltestelle 1 verlassen.';
                             this.rtm.LOGARRAY.push(text);
                             break;}
      case 'Haltestelle 2':{ text =  text + ' Haltestelle 2 verlassen.';
                             this.rtm.LOGARRAY.push(text);
                             break;}
      case 'Haltestelle 3':{ text =  text + ' Haltestelle 3 verlassen.';
                             this.rtm.LOGARRAY.push(text);
                             break;}
     default: {text = 'Update, kein gueltiger case' };
    }
    this.notification.update({id: 1, text: text, led:'#FF0000'});
  }
```

Listing 5.15: *Update einer Notification bei Verlassen einer Haltestelle*

```
loadUserID(): Promise<number>{
    console.log('LoadService: LoadUSERID');
    return this.persist.loadUserID();
  }

```

Listing 5.16: *Laden der UserID*

zurück.

5.9 SaveService

Der SaveService hat die Aufgabe, Daten und Werte aus dem Laufzeitspeicher RTMService in den Festspeicher PersistenzService zu überführen.

5.9.1 Verwendete Plugins

keine

5.9.2 Arbeitsweise

Es wird eine im PersistenzSerivce 5.10 vorhandene generische Speichermethode mit Übergabeparametern aufgerufen. Listing 5.17 zeigt den Aufruf. Die Methode des 5.10 ist generisch, so müssen als Argumente ein Label vom Typ String und ein Parameter, hier vom Typ number übergeben werden.

```
saveUserID(uid: number){
    console.log('SaveService: UserID');
    this.persistenzService.saveSingleParameter('USERID', uid);
  }
```

Listing 5.17: *Speichern der UserID*

```
loadUserID(): Promise<number> {
    console.log('PersistenzService: LoadUserID');
    return this.nstorage.getItem('USERID');
  }
```

Listing 5.18: *Laden der UserID vom Native Storage*

5.10 PersistenzService

Der PersistenzService verwaltet den Festspeicher des Geräts und bietet dafür Lade- und Speichermethoden an.

5.10.1 Verwendete Plugins

@ionic-native/native-storage [19]

5.10.2 Arbeitsweise

Der Speicher wird „Native Storage “genannt. Es wird der native Speicher des jeweiligen Geräts verwendet. Das Speichern erfolgt via Key und Value. Soll nun ein Wert geladen werden, so wird die entsprechende *Load*-Methode mit dem Key aufgerufen. Im Beispiel 5.18 wird die UserID vom Festpeicher geladen. Der Zugriff erfolgt asynchron. Die UserID wird als Promise zurückgegeben.
Soll ein Wert gespeichert werden, so kommt die generische Speichermethode zum Einsatz. Hier wird schon wie im SaveService Referenz und Wert übergeben. Die Methode *setItem* ist als Promise ausgeführt, um der Asynchronität Rechnung zu tragen.

5.11 RTMService

Der Laufzeitspeicher enthält die Daten und Werte, die zur Laufzeit benötigt werden.

5.11.1 Verwendete Plugins

@ionic-native/native-geocoder [20]

```
saveSingleParameter(ref:string, value:any) {
    console.log('PersistenzService: save');
    this.nstorage.setItem(ref, value).then(() =>
      console.log(ref + 'gespeichert!'), (err) => {
      console.log('Fehler beim Speichern von ' + ref + '' + JSON.stringify(
    err));
     })
  }
```

Listing 5.19: *Generische Speichermethode für Native Storage*

```
save(){
    console.log('RTM SaveMethode aufgerufen.')
    this.saveService.saveTrackingIndicator(this.trackingIndicator);
    this.saveService.saveURL(this.URL);
    this.saveService.saveLat(this.lat);
    this.saveService.saveLong(this.long);
    this.saveService.saveADDRESS(this.address);
    this.saveService.saveTrackMarkers(this.trackMarkers);
    this.saveService.saveUserID(this.USER_ID);
    this.saveService.saveTrackingCode(this.trackingCode);
    this.saveService.saveLOGARRAY(this.LOGARRAY);
    this.saveService.saveTrackingID(this.trackingID);
    this.saveService.saveUUID(this.UUID);
  }
```

Listing 5.20: *Methode zum Persistieren des Appzustandes beim Beenden*

5.11.2 Arbeitsweise

Der RTMService nimmt Werte von verschiedenen Services entgegen und stellt sie anderen Services zur Verfügung. Alle Dienste die Daten anliefern oder benötigen, instanzieren den RTMService. Die Services, die Speicher- oder Lademethoden benötigen, rufen diese ebenfalls über den RTMService auf.

5.12 MapsService

Der MapsService dient zur Darstellung der Eigen- und Fremdposition auf einer Karte. Hier wird auch der zurückgelegte Weg des Beobachteten angezeigt.

5.12.1 Verwendete Plugins

@ionic-native/google-maps [21]

```
this._map = GoogleMaps.create('map_canvas', {
 camera: {
   target: {
     lat: 49.26204150113853,
     lng: 7.36005587579744
   },
  zoom: 15,
  tilt: 30
  }
});
```

Listing 5.21: *Instantiierung GoogleMap*

5.12.2 Arbeitsweise

Der MapsService bietet unter Verwendung der GoogleMaps API die Möglichkeit Dinge auf einer Karte darstellen zu lassen. Der MapsService erzeugt eine Karte, die zuvor durch Optionen parametriert wird. In Listing 5.21 ist die Instantiierung der Karte dargestellt.
Die hier angegebenen Optionen sind *map_canvas*, der Name des in dem die Karte erscheinen soll, und die Position der Kamera. Die Kameraposition wird hier als *target* bezeichnet und enthält einen Längen- sowie einen Breitengrad als Koordinate. Weitere Kameraoptionen sind *zoom*, die Größe des sichtbaren Ausschnitts und *tilt* die Neigung. Nachdem die Karte geladen wurde, wird mittels der *.one*-Methode einmalig auf ein Event gewartet. In dieser Situation wird auf das **MAP_READY**-Event gewartet. Die *.one*-Methode ist als Promise ausgeführt. Wenn die Karte bereit ist, wird das *mapReady*-Flag auf true gesetzt und die eigene Position abgefragt. Basierend auf den Positionsdaten wird mit „*return this._map.addMarker() ..* “ eine Markierung auf die Karte gesetzt. Diese Markierung muss ebenfalls parametriert werden.
Die Optionen sind:

title Titel bzw. Name des Markers.

snippet Zusatztext, der nach dem Klicken auf den Marker angezeigt wird.

position Position des Markers als LatLong-Koordinate.

animation Animation, mit der der Marker an der in *position* angegebenen Stelle auftaucht.

Der dazugehörige Programmcode befindet sich in Listing 5.22.

```
this._map.one(GoogleMapsEvent.MAP_READY).then(() => {
this._mapReady = true;
this._map.getMyLocation().then((location: MyLocation) =>{
return this._map.addMarker({
title: 'Sie sind hier!',
snippet: 'Ihre aktuelle Position',
position: location.latLng,
animation: GoogleMapsAnimation.BOUNCE
})})
});
}
```

Listing 5.22: *Automatische Darstellung der eigenen Position*

5.13 TrackerService

Der Location Tracker stellt die aktuelle GPS Position des Telefons bereit und überträgt diese an den RTMService. Das Ermitteln der GPS Position ist nicht immer gleich, es wird unterschieden ob sich die Anwendung im Vorder- oder Hintergrund befindet.

5.13.1 Verwendete Plugins

@ionic-native/BackgroundGeolocation [22]
@ionic-native/Geolocation [23]

5.13.2 Arbeitsweise

Sobald der Nutzer auf der Homepage **Tracking** auswählt, wird die Positionserfassung mittels *startTracking()* gestartet. Nachdem eine Positionsfeststellung erfolgt ist, wird die Position mithilfe des APIService an die API weitergegeben.
Beim Aufruf der Seite wird der iBeaconService mittels der Methode **initBeacon()** aktiviert und iBeacons werden nun erkannt.

Hintergrund

Wird *Bustracker* in den Hintergrund geschickt, ist das Hintergrundtracking aktiv. Dies geschieht regelmäßig bei der Nutzung einer anderen App oder beim Deaktivieren des Bildschirms. Das backgroundGeolocation-Plugin benötigt eine Konfiguration. Diese wird im Code direkt vor dem Aufruf der eigentlichen Methode erzeugt.
Diese Konfiguration wird an die *.configure()* Methode übergeben, diese liefert ein Observeable vom Typ *BackgroundGeolocationResponse* zurück.

```
let config = {
            desiredAccuracy: 0,
            stationaryRadius: 20,
            distanceFilter: 10,
            debug: false,
            interval: 1000,
            startForeground: false
            stopOnTerminate: true,
        };
```

Listing 5.23: *Konfiguration Backgroundtracking*

Das Backgroundtracking wird mittels der *.configure()* Methode konfiguriert und muss explizit durch die *.start()* Methode aktiviert werden. Die *.start()* Methode gibt ein Promise zurück, ob das Starten geglückt ist. Im Erfolgsfall, kann mit *.then(()=>)* weiterer Code ausgeführt werden, der auf den erfolgreichen Start wartet. Der Fehlerfall kann mittels *.catch((err)=>)* abgefangen werden.
Die BackgroundGeolocationResponse wird im angegebenen Intervall (Millisekunden) aktualisiert. Bei *Bustracker* werden aktuell Längen- und Breitengrad als Datum im Latlong-Format verwendet. Diese können vom *location*-Objekt mittels Punktoperator gewonnen werden. Die Verwendung wird in Listing 5.24 gezeigt. Jede Aktualisierung stellt einen Erfolgsfall dar. Ebenfalls im Listing 5.24 zu sehen ist der Code, der im Erfolgsfall ausgeführt wird. Um eine ChangeDetection zu garantieren, wird dieser Code in einer Kopie der aktuellen Angular Zone ausgeführt. Weiterhin werden Längen- und Breitengrad in die entsprechenden Variablen des Laufzeitspeichers 5.11 geschrieben.
Die Variable **bodydata** wird aus aktuellen Daten erzeugt und bei jedem neuen „Fix“an die Methode *postData()* des API-Service 5.5 Parameter übergeben. Dies sorgt dafür, dass jede Position an den Server gesendet wird. Im Anschluss wird die neue Position dazu verwendet um eine „menschenlesbare “Adresse zu erzeugen. Diese wird auf Tracking- und WatchPage angezeigt.
Die iOS Plattform benötigt noch eine Mitteilung über das Ende des Backgroundtracking, siehe Listing 5.25.

Vordergrund

Sollte sich *Bustracker* im Vordergrund befinden, wird Foregroundtracking benötigt. Analog zum Backgroundtracking wird eine Konfigurationsvariable verwendet, siehe Listing 5.26.
Die Optionen werden beim Aufruf der *watchPosition* Methode übergeben. Die Ausgabe wird mittels *filter((p: any) => p.code === undefined)* so eingestellt, dass nur gültige Werte ausgegeben werden, siehe Listing 5.27.

```
this.backgroundGeolocation.configure(config).subscribe((location) => {

      console.log('BackgroundGeolocation:  ' + location.latitude + ',' +
   location.longitude);

      this.zone.run(() => {
        this.lat = location.latitude;
        this.rtm.lat = location.latitude;
        let bodydata = {id:0, lat: this.rtm.lat, lon: this.rtm.long, time:
   Date.now(), user_id: this.rtm.USER_ID, beacon_id: 0, beacon_active: true
   };
        this.apiCall.postData(bodydata);
        console.log('LAT BGTR: '+this.lat);
        this.lng = location.longitude;
        this.rtm.long = location.longitude;
        console.log(' LNG BGTR: '+ this.lng)
        this.geoCode();
      });
      if(this.platform.is('ios'))
      {
        this.backgroundGeolocation.finish().then(
          ()=> console.log('location-tracker.service:backgroundGeolocation.
   configure: ios finish = ok'));
      }
    }, (err) => {

      console.log(err);

    });
```

Listing 5.24: *Aktivierung Backgroundtracking*

```
if(this.platform.is('ios'))
            {
            this.backgroundGeolocation.finish().then(()=>
            console.log('(...)ios finish = ok'));
            }
```

Listing 5.25: *finish()-Methode für iOS*

```
let options = {
            frequency: 1000,
            enableHighAccuracy: true
        };
```

Listing 5.26: *Konfiguration Foregroundtracking*

```
1         this.watch = this.geolocation.watchPosition(options).filter((p: any
      ) => p.code === undefined)
2             .subscribe((position: Geoposition) => {
3 // Run update inside of Angular's zone
4                 this.zone.run(() => {
5                    this.lat = position.coords.latitude;
6           this.rtm.lat = position.coords.latitude;
7           console.log('FOREGRNDLAT: '+ this.lat);
8           this.lng = position.coords.longitude;
9           this.rtm.long = position.coords.longitude;
10          console.log(' FOREGRNDLONG:'+ this.lng);
11          this.geoCode();
12                });
13
14            });
```

Listing 5.27: *.watch()-Methode*

```
1 stopTracking() {
2             console.log('location-tracker.service.stopTracking');
3             this.backgroundGeolocation.stop();
4             console.log('location-tracker.service.stopped by function call
      stopTracking()');
5             this.watch.unsubscribe('location-tracking.unsubscribed');
6           }
```

Listing 5.28: *.stopTracking()-Methode*

Beim Betreten der TrackingPage wird der *trackingIndicator* abgefragt. Basierend auf dem Wert (true oder false) wird das Tracking mittels *.startTracking()* aktiviert. Es existiert eine *.stopTracking()* Methode, diese beendet das Backgroundtracking und beendet die *subscripton* auf der *.watchPosition()* Methode. Diese Methode wird aber nur benötigt, sollte das Tracking zur Laufzeit beendet werden. Beim Beenden von *Bustracker* wird das Tracking automatisch beendet. Dieses Feature ist in der aktuellen Version des *backgroundGeolocation*-Plugins eingeführt worden. In der Vergangenheit musste die *.stopTracking()*-Methode beim Beenden der App zwingend ausgeführt werden. Listing 5.28 zeigt die *.stopTracking()*-Methode.

5.13.3 URLService

Der URLService wird zum Zeitpunkt der Erstellung dieser Arbeit nicht genutzt.

5.13.4 Verwendete Plugins

keine

5.13.5 Arbeitsweise

In der Struktur ist der URLService vorgesehen, um in Zukunft dynamisch URLs für API-Zugriffe zu erzeugen.

Kapitel 6

Fazit und Résumé

Die Fallstudie *Bustracker* wurde mit folgendem Ziel begonnen: „*Mit Hilfe von mobilen Geräten realisieren wir eine Tracking-Möglichkeit für Kinder, die durch einen Bustransport in ländlichen Regionen auf dem Weg zum oder vom Kindergarten, Grundschule oder auch weiterführende Schule befinden.*“
Zu diesem Zwecke wurde eine plattformübergreifend lauffähige Applikation für Mobilgeräte vollständig neu entwickelt. Zur Positionsbestimmung werden GPS und iBeacons verwendet. Die Applikation, ausgeführt als Single Page Application, zeigt die Machbarkeit und kann in Zukunft unter realen Bedingungen getestet werden.
Bustracker existiert als Prototyp. Es ist nun möglich, ein Kind auf seinem Weg zur oder von der Betreuung zu beobachten. Die Kombination aus GPS und iBeacons hat sich als zweckmäßig und vorteilhaft erwiesen.

6.1 Lessons learned

In diesem Abschnitt werden die im Laufe der Entwicklung aufgetretenen Probleme sowie deren Lösungen diskutiert.

6.1.1 Marker eignen sich nicht zum Anzeigen von Wegen

Zu Beginn der Entwicklung wurde der Ansatz verfolgt, einzelne Marker in Reihe anzuordnen und so den Fahrweg nachzuvollziehen. Mit Hilfe der *makeTrack(LatLng)*-Methode wurde bei jeder neuen Positionsbestimmung ein Marker auf der Map erzeugt (Listing 6.1). Einige wenige Marker lassen sich ohne Performanceverlust beim scrollen darstellen. Steigt die Zahl der Marker jedoch an, so kommt es zu einer ruckelnden Darstellung. Die Performance der Karte beim Scrollen innerhalb und außerhalb der Karte nahm stark ab, bis hin zur Unzumutbarkeit für den Nutzer.

```
1  makeTrack(latlong:LatLng){
2    let moptions= {
3    title:'Strecke',
4   snippet: 'Hier faehrt der Bus!',
5   position: latlong,
6   animation: GoogleMapsAnimation.BOUNCE
7  }
```

Listing 6.1: *Erstellen einzelner Marker*

Als erste Gegenmaßnahme wurde die Darstellung der Marker analog zum Gameloop Konzept verwendet. In ihrer Veröffentlichung ***Real Time Game Loop Models for Single-Player Computer Games*** [24] schreiben die Autoren in Kapitel 2.1, dass die einfachste Game Loop aus den Schritten „User Input“, dem„Update“ und dem „Rendern“ besteht. In *Bustracker* wäre der User Input die neue Position, das Update wäre das aktualisierte Array und das Rendern der Aufruf einer Methode zum Darstellen einer Gruppe von Markern, eines sogenannten Clusters. Eine Testimplementierung zeigte bereits zu Beginn, dass hierdurch kein Performancegewinn zu erzielen ist.
Als Lösung wurde der Weg über die Polyline gewählt. Die Polyline ist eine Linie, welche die sie definierenden Punkte verbindet. Das Zeichen einer Linie beansprucht wesentlich weniger Ressourcen als das Zeichnen einer Grafik in Form eines Markers für jeden einzelnen Punkt. Listing 6.2 zeigt die Verwendung der Polyline.

```
1 makePolyFromArray(array: any[]){
2 this._map.addPolyline({
3 points: array,
4 'color' : '#AA00FF',
5 'width': 10,
6 'geodesic': true
7 });}
```

Listing 6.2: *Zeichnen einer Polyline auf einer Google Map*

Die Punkte werden als Array vom Typ LatLng[] übegeben.
Weitere Parameter sind:

color Farbe der Linie in Hexadezimalschreibweise

width Breite der Linie in Pixel

geodesic Falls true gewählt ist, passt sich die Linie der Erdkrümmung an.

```
1  snapToRoad(dinger: LatLng[]): Promise<LatLng[]> {
2    console.log('Hello SnapToRoad');
3    let prom = new Promise<LatLng[]>((resolve, reject) => {
4    let payload: string = '';
5    let ret: LatLng[] = [];
6    for (let i: number = 1; i <= 100; i++) {
7      payload = payload + dinger[i].lat + ',' + dinger[i].lng;
8      if (i < 100)
9      payload = payload + '\|';
10   }
11     this.http.get('https://roads.googleapis.com/v1/snapToRoads?path=' +
       payload
12     + '&interpolate=true&key=<Der API-Key>').subscribe((data) => {
13     data['snappedPoints'].forEach((itr) => {
14     ret.push(new LatLng(itr.location.latitude, itr.location.longitude));
15   });
16       resolve(ret);
17     }), (err) => {
18         reject('Fehler beim SnapToRoad' + JSON.stringify(err));
19       };
20    });
21     console.log('Bye SnapToRoad');
22     return prom;
23   }
24 }
```

Listing 6.3: *Verwendung der RoadsAPI durch* Bustracker

6.1.2 Polyline wurde nicht gezeichnet, maximal 100 Punkte sind erlaubt

Um den zurückgelegten Pfad vom beobachteten Nutzer an den Straßenverlauf anzupassen, wird die Google Maps Roads API [25] verwendet. Diese nimmt eine Menge von LatLong Koordinaten entgegen und interpoliert, wenn gewählt, die Strecke mit zusätzlichen Punkten. Diese Punkte werden im JSON-Format zurückgegeben und können zur Darstellung einer Linie auf der Karte verwendet werden.
Listing 6.3 zeigt die Methode, die eine Menge von LatLong-Koordianten an die Google Maps Roads API übergibt und die Antwort in Form eines Promise zurückgibt. Die ersten Versuche, die Koordinaten aus der *Bustracker*-API zu verwenden, führten nicht zum Erfolg. Nach Einfügen von Konsolenausgaben und teilweisem Auskommentieren von Code wurde ersichtlich, dass der Fehler im Bereich des GET-Requests lag. Um das „err " Objekt auf der Konsole lesbar auszugeben, wird es mit Hilfe der *JSON.stringify()*-Methode in einen String umgewandelt. Aus der Fehlermeldung ging hervor, dass die Anzahl der an die Google Maps Road API übergebenen Punkte zu hoch ist, maximal 100 sind erlaubt. Um diesem Umstand Rechnung zu tragen wurden die Zeilen 6 -10

```
1 $ ionic cordova plugin add cordova-plugin-googlemaps --variable
    API_KEY_FOR_ANDROID="YOUR_ANDROID_API_KEY_IS_HERE" --variable
    API_KEY_FOR_IOS="YOUR_IOS_API_KEY_IS_HERE"
2 $ npm install --save @ionic-native/google-maps
```

Listing 6.4: *CLI Befehle - Installation Google Maps Plugin*

aus dem Listing 6.3 eingefügt. Diese Schleife sorgt nun dafür, dass nur die letzten 100 Elemente an die Google Maps API geschickt werden. Die Strecke wird sukzessive in Abschnitten gezeichnet.

6.1.3 API-KEY für native Android und iOS Anwendungen

Bei der Installation des Google Maps Plugins [21] muss im Konsolenbefehl (Listing 6.4) der API-Key für die jeweilige Plattform angegeben werden.
Dieser Key ist über die Google Developer Konsole [26] einzurichten und zu beziehen. Googles Dokumentation zu Maps enthält eine ausführliche Anleitung zum Erzeugen von Keys. [27].

6.2 Ausblick

Die Anwendung wird kontinuierlich weiterentwickelt und neue Features werden implementiert. An dieser Stelle wird ein Überblick über weitere Entwicklungsmöglichkeiten von *Bustracker* gegeben.

6.2.1 Umstellung auf freies Kartenmaterial

Um in Zukunft weniger abhängig von Google und deren APIs zu sein, soll die Anwendung auf freies Kartenmaterial umgestellt werden. Das Kartenmaterial soll von OSM (Open Street Maps) bezogen werden [28]. Das Anpassen an den Straßenverlauf soll mittels der OSRM (Open Street Routing Machine) [29] durchgeführt werden.

6.2.2 Berechung der Ankuftszeit am Ziel

Ein geplantes Feature ist die Berechnung der Ankunftszeit an einem Ziel. Dem Beobachter soll mitgeteilt werden, zu welcher Zeit die Ankunft am Ziel erwartet wird. Zunächst soll dies mit Hilfe der Google Distance API gelöst werden. In einer späteren Iteration soll dies ebenfalls unter Verwendung der OSRM durchgeführt werden.

6.2.3 Nutzerauthentifizierung

Zukünftig soll eine Authentifizierung durchgeführt werden, um nur befugte Beobachter für ein bestimmtes Gerät zuzulassen. Verschiedene Lösungen sind denkbar, z. B. das Generieren eines einzigartigen Schlüssels für jeden einzelnen Trackingvorgang. Denkbar ist auch eine Konfiguration einer Art Familie, in der einzelne Geräte einem Beobachter zugewiesen werden können.

6.2.4 Sparchassistentenunterstützung

Es wäre denkbar, die Datenbankeinträge des Servers mittels sogenannter Skills für Sprachassistenten abzurufen. Die rein auditive Darstellung könnte die momentane geocodierte Position bzw. den jeweiligen Checkpoint enthalten.

6.3 Fazit

Bustracker ist ein Beispiel für eine Anwendung im Rahmen der Digitalisierung des ländlichen Raumes. Zurzeit ist die Anwendung im Status eines Prototyps. Durch kontinuierliche Weiterentwicklung, vor allem das Einarbeiten der im Ausblick genannten Features und Anpassungen an zukünftig evaluierten Nutzungsszenarien, wird *Bustracker* zu einer höheren Sicherheit auf dem Schulweg beitragen.

Literaturverzeichnis

[1] ROGGE, Sven: *Quelle Abbildung 2.3 TypeScript Sprach-Hirachie - Letzter Aufruf: 15.02.2018.* – https://blog.flavia-it.de/ecmascript-6-und-TypeScript-auf-in-die-neue-javascript-generation/

[2] FOUNDATION, The Apache S.: *Cordova Dokumentation - Letzter Aufruf: 15.02.2018.* – https://cordova.apache.org/docs/en/latest/guide/overview/

[3] IONIC: *Ionic Framework UI-Elemente - Letzter Aufruf: 15.02.2018.* – https://ionicframework.com/docs/components/#overview

[4] *Ionic Framework Pulse Ansicht - Letzter Aufruf: 15.02.2018.* – https://github.com/ionic-team/ionic/pulse/monthly

[5] *MIT Lizenz - Letzter Aufruf: 15.02.2018.* – https://opensource.org/licenses/MIT

[6] SCHMITT, Johannes: *GisboAlarm - Entwicklung einer cross-platform App mittels Ionic-Framework, Praxisprojekt, Hochschule Kaiserslautern, Zweibrücken.* 2017

[7] *Angular Website - Letzter Aufruf: 15.02.2018.* – https://angular.io

[8] CORPORATION, Microsoft: *TypeScript Dokumentation - Letzter Aufruf: 15.02.2018.* https://www.typescriptlang.org

[9] IONIC: *Apple iBeacon Spezifikation - Letzter Aufruf: 15.02.2018.* – https://developer.apple.com/ibeacon/

[10] IONIC: *Beacon Region Erklärung - Letzter Aufruf: 15.02.2018.* – https://community.estimote.com/hc/en-us/articles/203776266-What-is-a-beacon-region-

[11] ABTS, Dietmar: *REST-basierte Web Services mit JAX-RS.* Wiesbaden : Springer Fachmedien Wiesbaden, 2015. – 277–330 S. http://dx.doi.org/10.1007/978-3-658-09921-3_10. http://dx.doi.org/10.1007/978-3-658-09921-3_10. – ISBN 978–3–658–09921–3

[12] PROJECT, JSDoc 3.: *JSdoc Spezifikation - Letzter Aufruf: 15.02.2018.* – http://usejsdoc.org

[13] MARIADB, Corporation: *MariaDB Dokumentation - Letzter Aufruf: 15.02.2018.* https://mariadb.com/kb/en/library/documentation/

[14] KALWEIT, Fabian ; SCHMITT, Johannes: *API Dokumentation Bustracker, Hochschule Kaiserslautern, Zweibrücken*, 2018

[15] *Angualr HTTPClientModule - Letzter Aufruf: 15.02.2018*. 2018. – https://angular.io/guide/http

[16] IONIC: *iBeacon Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. 2017. – https://ionicframework.com/docs/native/ibeacon/

[17] *Kapitel* Entwurfs-, Architektur- und Integrationsmuster. In: SCHATTEN, Alexander ; DEMOLSKY, Markus ; WINKLER, Dietmar ; BIFFL, Stefan ; GOSTISCHA-FRANTA, Erik ; ÖSTREICHER, Thomas: Heidelberg : Spektrum Akademischer Verlag, 2010. – ISBN 978–3–8274–2487–7, 229–300

[18] IONIC: *Local Notification Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/local-notifications/

[19] IONIC: *NativeStorage Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/native-storage/

[20] IONIC: *Native GeocoderPlugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/native-geocoder/

[21] IONIC: *Google Maps Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/google-maps/

[22] IONIC: *IonicBackground-Geolocation Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. 2017. – https://ionicframework.com/docs/native/background-geolocation/

[23] IONIC: *Ionic Geolocation Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/geolocation/

[24] VALENTE, Luis ; CONCI, Aura ; FEIJÓ, Bruno: Real time game loop models for single-player computer games

[25] *Google Maps Roads API Dokumentation - Letzter Aufruf: 15.02.2018*. – https://developers.google.com/maps/documentation/roads/intro?hl=de

[26] INC., Google: *Google Developer Konsole - Letzter Aufruf: 15.02.2018*. – https://console.developers.google.com/apis/dashboard

[27] INC., Google: *Google Developer Konsole - Schlüsselanforderung - Letzter Aufruf: 15.02.2018*. – https://developers.google.com/maps/documentation/javascript/get-api-key?hl=de

[28] FOUNDATION, OpenStreetMap: *Open Street Maps FAQ- Letzter Aufruf: 15.02.2018*. – https://www.openstreetmap.de/faq.html#wie_daten_nutzen

[29] PROJECT, Open Source Routing M.: *Open Source Routing Machine - Letzter Aufruf: 15.02.2018*. – http://project-osrm.org/

[30] GOOGLE, Inc: *Angular Dokumentation - Letzter Aufruf: 15.02.2018*. – https://angular.io/docs/ts/latest/

[31] WEISSE, Bengt: *Ionic Tutorial - Letzter Aufruf: 15.02.2018*. – https://angularjs.de/artikel/ionic2-tutorial-deutsch

[32] IONIC: *Ionic Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/v2/

[33] SCHMITT ; SANDOZ ; LÊ: *GisboMobile - ein cross-platform Prototyp mit Ionic-Framework, Studienprojekt, Hochschule Kaiserslautern*, 2017

[34] IONIC: *Toast Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/toast/

[35] IONIC: *BackgroundMode Plugin Dokumentation - Letzter Aufruf: 15.02.2018*. – https://ionicframework.com/docs/native/background-mode/

[36] HÖLLER, Christoph: *Angular Das umfassende Handbuch*. Rheinwerk Computing, 2017. – ISBN 978–3–8362–3914–1

[37] *"Bedingter (ternärer) Operator (?:) (JavaScript) MSDN JavaScript Dokumentation - Letzter Aufruf: 15.02.2018*. – https://msdn.microsoft.com/de-de/library/be21c7hw(v=vs.94).aspx

Abkürzungsverzeichnis

API	Application Programming Interface
CSS	Cascading Style Sheets
ECMAScript	European Computer Manufacturers Association Script
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
Kfz	Kraftfahrzeug
OSM	Open Street Maps
OSRM	Open Street Routing Machine
REST	Representational State Transfer
UUID	Universally Unique Identifier
VPN	Virtual Private Network

Anhang A

Anhang

1. Bustracker API Dokumentation
2. CD mit Quellcode und Latex-Dateien

API Dokumentation Bustracker

Hochschule Kaiserslautern IMST
Fabian Kalweit,
Johannes Schmitt

Abstract

Die Bustracker API ist eine RestAPI für die Anwendung Bustracker. Die API stellt Funktionen zum Speichern und Laden von Positionsdaten, Benutzerdaten und Daten über die eingesetzten IBeacons bereit.

Diese Dokumentation beschreibt den allgemeinen Aufbau der Bustracker API und genauer die einzelnen Methoden.

Contents

1 Allgemeiner Aufbau

Die RestAPI wurde mit TypeScript und NodeJs umgesetzt. Die Daten werden in einer MariaDb gehalten und über TypeScript-MYSQL abgefragt. In Abbildung 1 ist die Kommunikation der drei Elemente dargestellt.

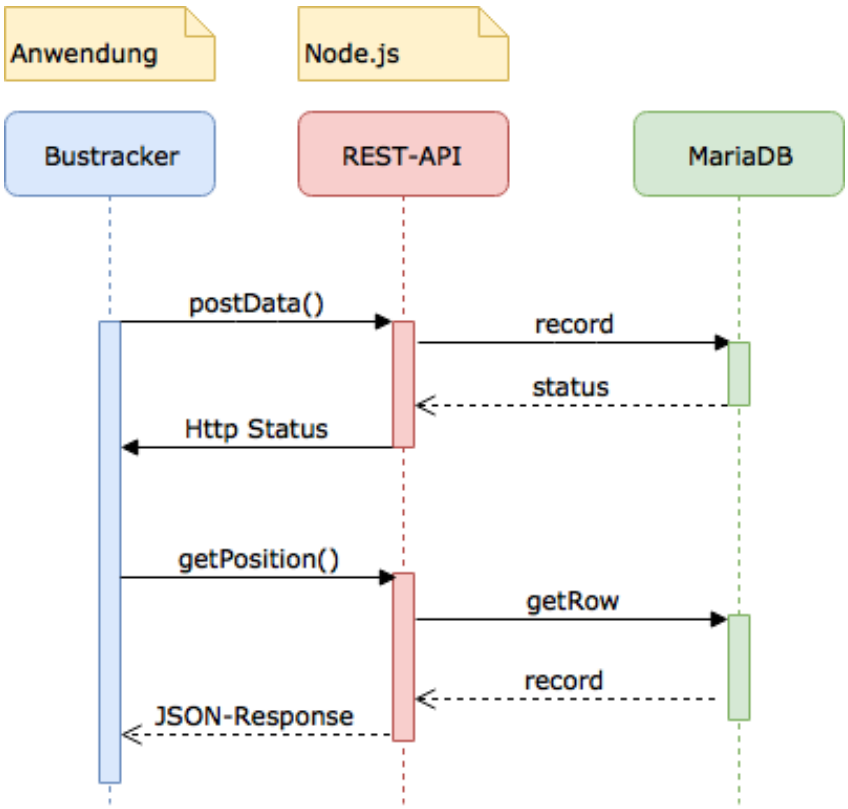

Figure 1: Kommunikation mit der Datenbank und dem Bustracker

1.1 Abhängikeiten der RestAPI

Alle Abhängigkeiten werden mit npm installiert.

devDependencies:

typescript : TypeScript Paket. Sollte global installiert werden.

mocha : Testframework für TypeScript

chai und chai-http : Erweiterte Asserts dür TypeScript.

types/mysql : MYSQL-Types für TypeScript

gulp und gulp-typescript : Ermöglicht das Erstellen von Scripts zum Beispiel zum Erstellen der Anwendung.

ts-node : Anbindung von TypeScript an NodeJS

Dependencies:

express : Express ist ein einfaches und flexibles Node.js-Framework von Webanwendungen, das zahlreiche leistungsfähige Features und Funktionen für Webanwendungen und mobile Anwendungen bereitstellt. Mithilfe unzähliger HTTP-Dienstprogrammmethoden und Middlewarefunktionen gestaltet sich das Erstellen einer leistungsfähigen

API schnell und einfach.
http://expressjs.com/de/

morgan : HTTP request logger middleware for node.js

type-sql : Ein typischerer Querybuilder für SQL erstellt mit TypeScript. Es werden Postgres und MySQL unterstützt.
https://github.com/ggmod/type-sql

2 Bustracker API

Hier werden die von außen sichtbaren Klassen der Bustracker API beschreiben.

2.1 PositionRouter

Der **Position Router** stellt Methoden zum Laden und Speichern von Positionsdaten zur Verfügung. Im Folgenden sind die einzelnen Methoden beschreiben:

getPositionsg : Gibt alle Positionen in der Datenbank zurück. Wird über die Adresse GET **root/api/v1/positions** aufgerufen.
Bei Erfolg gibt die Methode den Status 200, die Nachricht 'Success' und die Daten zurück.
Bei einem internen Fehlschlag gibt die Methode 500, die Nachricht 'Database error' und die Fehlernachricht des DBService.

getUserPosition : Gibt alle Positionen eines bestimmten Benutzers zurück. Wird über die Adresse GET **root/api/v1/positions/:id** aufgerufen. Bei Erfolg gibt die Methode den Status 200, die Nachricht 'Success' und die Daten zurück.
Bei einem internen Fehlschlag gibt die Methode 500, die Nachricht 'Database error' und die Fehlernachricht des DBService.
Wird eine nicht lesbare oder nicht vorhandene User-ID mitgegeben, gibt die Methode den Status 404 un die nachricht 'No position found with the given id.' zurück.

getLastUserPosition : Gibt die letzten x Positionen eines bestimmten Benutzers zurück. Wird über die Adresse GET **root/api/v1/positions/last** aufgerufen.
Parameter: userID und limit →
root/api/v1/positions/last?id=n&limit=x
Bei Erfolg gibt die Methode den Status 200, die Nachricht 'Success' und die Daten zurück.
Bei einem internen Fehlschlag gibt die Methode 500, die Nachricht 'Database error' und die Fehlernachricht des DBService.
Wird eine nicht lesbare oder nicht vorhandene User-ID mitgegeben, gibt die Methode den Status 404 un die nachricht 'No position found with the given id.' zurück.

postPosition : Speichert einen Positionsdatensatz in der Datenbank. Wird über POST **root/api/v1/positions** aufgerufen.
Bei Erfolg gibt die Methode den Status 200, die Nachricht 'Success' zurück.
Bei einem internen Fehlschlag gibt die Methode 500, die Nachricht 'Database error' und die Fehlernachricht des DBService.
Sind die Werte für lat oder lon gleich null, gibt die Methode den Status 400 un die nachricht 'lat or lon = 0.' zurück.

2.2 UserRouter

Der **User Router** stellt Methoden zum Laden und Speichern von Benutzerdaten zur Verfügung. Im Folgenden sind die einzelnen Methoden beschreiben:

getUser : Gibt alle User in der Datenbank zurück. Wird über die Adresse GET **root/api/v1/users** aufgerufen.
Bei Erfolg gibt die Methode den Status 200, die Nachricht 'Success' und die Daten zurück.
Bei einem internen Fehlschlag gibt die Methode 500, die Nachricht 'Database error' und die Fehlernachricht des DBService.