HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

fpga.company
fpga & hardware design services

HiveTrack

Studien-/Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2018

Autor(en):	Pascal Schweizer
	Arooran Thanabalasingam
Betreuer:	Mirko Stocker
Projektpartner:	Silvio Ziegler, FPGA Company GmbH, Tann, ZH
Experte:	Leo Büttiker
Gegenleser:	Prof. Dr. Daniel Patrick Politze

Abstract

Die FPGA Company GmbH hat ein Gerät entwickelt, das periodisch Messdaten (Position, Beschleunigung, Temperatur, Feuchtigkeit) erfasst und über ein GSM Modem verschickt. Solche Geräte sollen dann in Paketen installiert und gemeinsam mit deren Inhalt verschickt werden. Dadurch kann gemessen werden, wo, wann und was während einer Lieferung passiert ist.
Ziel dieser Arbeit war es, ein System zu entwickeln, das diese Messdaten empfängt, persistiert und über ein Web-Frontend darstellt. Dafür wurden vier Komponenten entwickelt. Der Receiver, der die Messdaten empfängt und in einer Datenbank abspeichert. Die Datenbank, deren Datenmodell so aufgebaut ist, dass das Frontend mit möglichst wenigen und einfachen Abfragen die benötigten Daten auslesen kann. Das Webapp-Backend, welches die Daten aus der Datenbank liest und aufbereitet und dem Frontend eine REST API zur Verfügung stellt. Und schlussendlich noch das Webapp-Frontend, das die Positions- und Messdaten auf Karten und Diagrammen darstellt. So wird den Benutzern ermöglicht, ihre Geräte zu überwachen und deren Daten auszuwerten.

Inhalt

I Einleitung

1 Aufgabenstellung

1.1 Betreuer und Experte

Diese Arbeit wird für die FPGA Company GmbH durchgeführt und wird von Mirko Stocker, HSR, IFS, m1stocke@hsr.ch betreut.

Industriepartner:

- Silvio Ziegler, FPGA Company GmbH

1.2 Studierende

Diese Arbeit wird als **Bachelorarbeit** an der Abteilung Informatik durchgeführt von

- Pascal Schweizer, pascal.schweizer@hsr.ch

Diese Arbeit wird als **Studienarbeit** an der Abteilung Informatik durchgeführt von

- Arooran Thanabalasingam, arooran.thanabalasingam@hsr.ch

1.3 Einführung

Die FPGA Company GmbH entwickelt ein Gerät das periodisch Messdaten (Position, Beschleunigung) erfasst. Diese Daten werden über ein GSM Modem an einen Server geschickt. Dieser empfängt und persistiert die Daten und reichert sie mit zusätzlichen Informationen (Wetter) an.

1.4 Ziele der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung des Clients und Servers zur Verwaltung und Visualisierung dieser Daten. Ein bereits bestehendes System (HiveWatch) kann für den Empfang und die Speicherung wiederverwendet werden, auch das Deployment lässt sich analog dem bestehenden System wiederverwenden. Front- und Backend der Web-Anwendung sollen Sie aber neu entwickeln.

Die genauen Anforderungen werden Sie zusammen mit dem Industriepartner entwickeln. In den wöchentlichen Meetings mit dem Industriepartner und dem Betreuer werden Sie die Aufgaben der nächsten Wochen priorisieren.

Es finden wöchentliche Besprechungen mit dem Betreuer statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen, ausser der Kick-off Besprechung, sind von den Studierenden mit einer Traktandenliste vorzubereiten und zu leiten. Als erstes Traktandum soll immer der Stand des Projektes präsentiert werden (Was wurde gemacht? Was wurde erreicht? Wie viel Zeit wurde in was investiert?). Die Beschlüsse der Besprechungen sind durch die Studierenden zu protokollieren und an den Betreuer anschliessend zuzustellen, bzw. an einem allen zugänglichen Ort abzulegen.

Für die Durchführung der Arbeit ist nach dem Initialmeeting ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen (oder auch Zwischenversionen) gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Resultate.

1.5 Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente bzw. Berichtsteile sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren.

Bei einer gemischten Studien- und Bachelorarbeit muss ersichtlich sein, welche Ergebnisse von wem erzielt wurden. Für die Archivierung werden zwei Berichte abgegeben, diese können inhaltlich aber identisch sein und können zusammen am 15. Juni abgegeben werden.

1.6 Termine

Siehe auch Terminplan auf dem Skripteserver (Fachbereich/Bachelor-Arbeit_Informatik/BAI/)

Montag, den 19.02.2017	**Beginn der Arbeit**
08.06.2017	Die Studierenden geben den Abstract für die Diplomarbeitsbroschüre zur Kontrolle an ihren Betreuer/Examinator frei. Die Studierenden erhalten vorgängig vom Studiengangsekretariat die Aufforderung mit den Zugangsdaten zur Online-Erfassung des Abstracts im DAB-Tool. Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Examinator/Betreuer. Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen auf dem Skripteserver zur Verfügung.
13.06.2017	Der Betreuer/Examinator gibt das Dokument mit dem korrekten und vollständigen Abstract der Broschüre zur Weiterverarbeitung an das Studiengangsekretariat frei. Fertigstellung und Weitergabe des A0 Posters per Email bis 10.00 Uhr an das Studiengangsekretariat.
15.06.2017	**Abgabe des Berichts an den Betreuer bis 12.00 Uhr** Präsentation und Ausstellung der Bachelorarbeiten, 16 bis 20 Uhr
Bis zum 24.08.17	Mündliche BA-Prüfung

1.7 Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von 30 Stunden.

Für die Beurteilung der Bachelorarbeit ist der verantwortliche Dozent zuständig.

Gesichtspunkt	**Gewicht**
1. Organisation, Durchführung	1/6
2. Berichte (Abstract, Mgmt Summary, technischer u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/6
3. Inhalt*)	3/6
4. Mündliche Prüfung	1/6

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit präzisiert.

Die Bewertung für den SA-Studierenden wird den Vorkenntnissen entsprechend angepasst. Ebenfalls wird das Ergebnis nur bis zum Abgabetermin berücksichtigt.

Eine erfolgreiche Studienarbeit zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von 30 Stunden.

Für die Beurteilung der Studienarbeit ist der verantwortliche Dozent zuständig.

Gesichtspunkt	**Gewicht**
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, technischer u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt*)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit präzisiert.

Im Übrigen gelten die Bestimmungen der Abteilung Informatik für Studienarbeiten.

Rapperswil, den 19. Februar 2018.

Mirko Stocker

Dozent für Informatik
Institut für Software (IFS)
Hochschule für Technik Rapperswil

2 Management Summary

2.1 Ausgangslage

Die FPGA Company GmbH hat ein Gerät entwickelt, das periodisch Messdaten (Position, Beschleunigung, Temperatur, Feuchtigkeit) erfasst. Diese Geräte (Tracker) sollen in Paketen installiert und gemeinsam mit deren Inhalt verschickt werden, um so aufzeichnen zu können, wann, wo und was während der Lieferung passiert ist. Um eine Analyse der Messdaten zu ermöglichen, sollen diese über das GSM Modem der Tracker an einen Server geschickt werden, wo sie persistiert und über ein Web-Frontend darstellt werden sollen.

2.2 Vorgehen, Technologien

Im Zuge dieser Arbeit wurden vier Komponenten entwickelt:

- Der Receiver empfängt die Messdaten der Tracker, validiert diese und speichert sie in einer Datenbank ab. Er ist komplett in Scala geschrieben und basiert auf dem Play Framework.
- Da die Tracker ihre Messdaten als JSON verschicken und auch im Webapp-Frontend wieder mit JSON gearbeitet wird, wurde entschieden die dokumentenorientierte NoSQL Datenbank MongoDB für die Speicherung der Daten einzusetzen.
- Das Webapp-Backend beinhaltet eine REST API, über die das Frontend die benötigten Daten abrufen kann. Genau wie der Receiver, basiert auch das Backend auf dem Play Framework und ist in Scala geschrieben. Um die REST API vor unerlaubten Zugriffen abzusichern wurde die Authentifizierungs-Library Silhouette benutzt.
- Das Webapp-Frontend wurde mit React und Redux entwickelt. Es verwendet die Google Maps API um den Positions-Verlauf der Tracker auf einer Karte darzustellen und generiert mithilfe der JavaScript Library Highcharts interaktive Diagramme für die übrigen Messdaten.

2.3 Ergebnis

Herausgekommen ist das voll funktionsfähige System namens HiveTrack. Es ermöglicht Benutzern auf einer Übersichtskarte die aktuellen Positionen und Zustände aller ihrer Tracker auf einen Blick einzusehen oder in der Detailansicht genauere Analysen der Tracking-Daten durchzuführen. Dort können die einzelnen Lieferungen isoliert voneinander betrachtet werden. Durch Festlegen von Grenzwerten für die verschiedenen Sensoren pro Lieferung, können fehlerhafte Werte in den Diagrammen hervorgehoben und so schnell erkannt werden. Für Administratoren werden eine Benutzer- und Tracker-Verwaltung zur Verfügung gestellt, in denen unter anderem neue Tracker erfasst und den Benutzern zugewiesen werden können. Zukünftig kann das System noch erweitert werden, sodass beispielsweise Benutzer bei Grenzwertüberschreitungen direkt per E-Mail oder SMS benachrichtigt werden können.

2.4 Ausblick

Als nächstes wird von der FPGA Company GmbH ein Feldtest durchgeführt, um Feedback der Endkunden zu erhalten. Darauf basierend wird dann entschieden, welche Verbesserungen oder Erweiterungen als nächstes implementiert werden.

II Analyse & Design

3 Ausgangslage

Vor dem Start dieses Projekts waren bereits einige Komponenten vorhanden. Einerseits die von der FPGA Company GmbH entwickelten Messgeräte, in diesem Bericht *Tracker* genannt, und andererseits ein ähnliches Vorprojekt namens HiveWatch. Im HiveWatch-Projekt wird der Vorgänger der hier verwendeten Tracker dafür eingesetzt Bienenstöcke zu wägen und die Gewichtsdaten an einen Server zu senden, über den diese dann dargestellt und ausgewertet werden können. Da diese Messgeräte bis auf die unterschiedlichen eingebauten Sensortypen praktisch identisch sind und auch gewisse Anforderungen an die Darstellung der Daten im Frontend Gemeinsamkeiten aufweisen, konnten einige Konzepte und Lösungen vom HiveWatch-Projekt übernommen werden. Beim Frontend wurden aus Benutzerfreundlichkeitsgründen auch gewisse Teile des Seitenaufbaus übernommen. Der Grund dafür ist, dass beide Systeme voraussichtlich von derselben Person administriert werden und sich diese Aufgabe einfacher gestaltet, wenn die dafür benötigten Komponenten ähnlich platziert und aufgebaut sind.

4 Anforderungen

Das Hauptziel des HiveTrack Projekts war, dass die Benutzer die Positionen und Sensordaten ihrer Tracker auf einer übersichtlichen graphischen Oberfläche einsehen und auswerten können. Die verschiedenen Use Cases wurden in den ersten Wochen gemeinsam mit dem Kunden erarbeitet und später noch erweitert.
Eine Übersicht über alle Use Cases befindet sich in Abbildung 1 und eine Beschreibung der Aktoren in Listing 1.

4.1 Use Cases

UC 01: Übersichtskarte

Der Benutzer sieht auf einer Übersichtskarte die aktuellen Positionen aller seiner Tracker und die neusten Messwerte deren Sensoren.

UC 02: Detailansicht

In der Detailansicht sieht der Benutzer die Positionsänderungen vom ausgewählten Tracker auf einer Karte. Zudem sieht der Benutzer den Verlauf der verschiedenen Sensordaten wie Feuchtigkeit, Temperatur und Beschleunigung in Form von Liniendiagrammen.

UC 03: Einstellungsgruppen verwalten

Der Benutzer kann Einstellungsgruppen erstellen, bearbeiten und löschen und seine Tracker diesen Gruppen zuweisen. Darin können pro Sensor ein oberer und unterer Grenzwert und das Sendeintervall des Trackers festgelegt werden. Nach einer Änderung einer Einstellungsgruppe werden alle dieser Gruppe zugewiesenen Tracker mit den neuen Werten aktualisiert. Jeder Benutzer besitzt seine eigenen Einstellungsgruppen.

UC 04: Standard Einstellungsgruppe konfigurieren

Jeder Tracker gehört anfangs der Standard Einstellungsgruppe des jeweiligen Benutzers an. Diese soll wie die anderen Einstellungsgruppen ebenfalls konfigurierbar sein.

UC 05: Benachrichtigung

Bei Über-/Unterschreitung der festgelegten Grenzwerte wird der Benutzer automatisch mithilfe des MessageBird Dienstes benachrichtigt. Ob und über welchen Kanal (E-Mail, SMS, Anruf) diese Benachrichtigung erfolgt, ist vom Benutzer konfigurierbar.

UC 06: Alarmhistory

Alle vergangenen Unter- bzw. Überschreitungen von Grenzwerten sollen für den Benutzer jederzeit einsehbar sein.

UC 07: Start/Stopp Paketsendung

Der Benutzer startet am Anfang von einer Lieferung die Paketsendung im GUI und am Ende der Lieferung stoppt er sie wieder, so dass die Lieferung als eine isolierte Paketsendung angesehen werden kann.

UC 08: Paketsendungen bearbeiten

Paketsendungen können auch im Nachhinein noch hinzugefügt / bearbeitet werden, falls zum Beispiel vergessen wurde eine Paketsendung zu starten / stoppen.

UC 09: CSV Export

Der Benutzer kann die Messwerte der verschiedenen Sensoren seiner Tracker als CSV exportieren.

UC 10: Heatmap

Der Benutzer sieht auf einer Heatmap in welchem Umfeld sich seine Tracker in einem bestimmten Zeitraum hauptsächlich bewegt haben.

UC 11: Tracker um-/benennen

Um seine Tracker einfach identifizieren zu können, soll es dem Benutzer möglich sein jedem seiner Tracker einen beliebigen Namen zu geben.

UC 12: Tracker verwalten

Der Administrator kann neue Tracker im System erfassen und diese jeweils einem Benutzer zuweisen.

UC 13: Benutzer verwalten

Benutzer können sich nicht selbst im System registrieren, sondern erhalten vom Administrator einen Account zugewiesen. Der Administrator kann also neue Accounts erstellen und diese auch bearbeiten.

UC 14: Messdaten speichern

Die Tracker senden ihre Messdaten periodisch über ihr eingebautes *GSM* Modem zum Server. Dort werden die Daten verarbeitet und in einer Datenbank gespeichert.

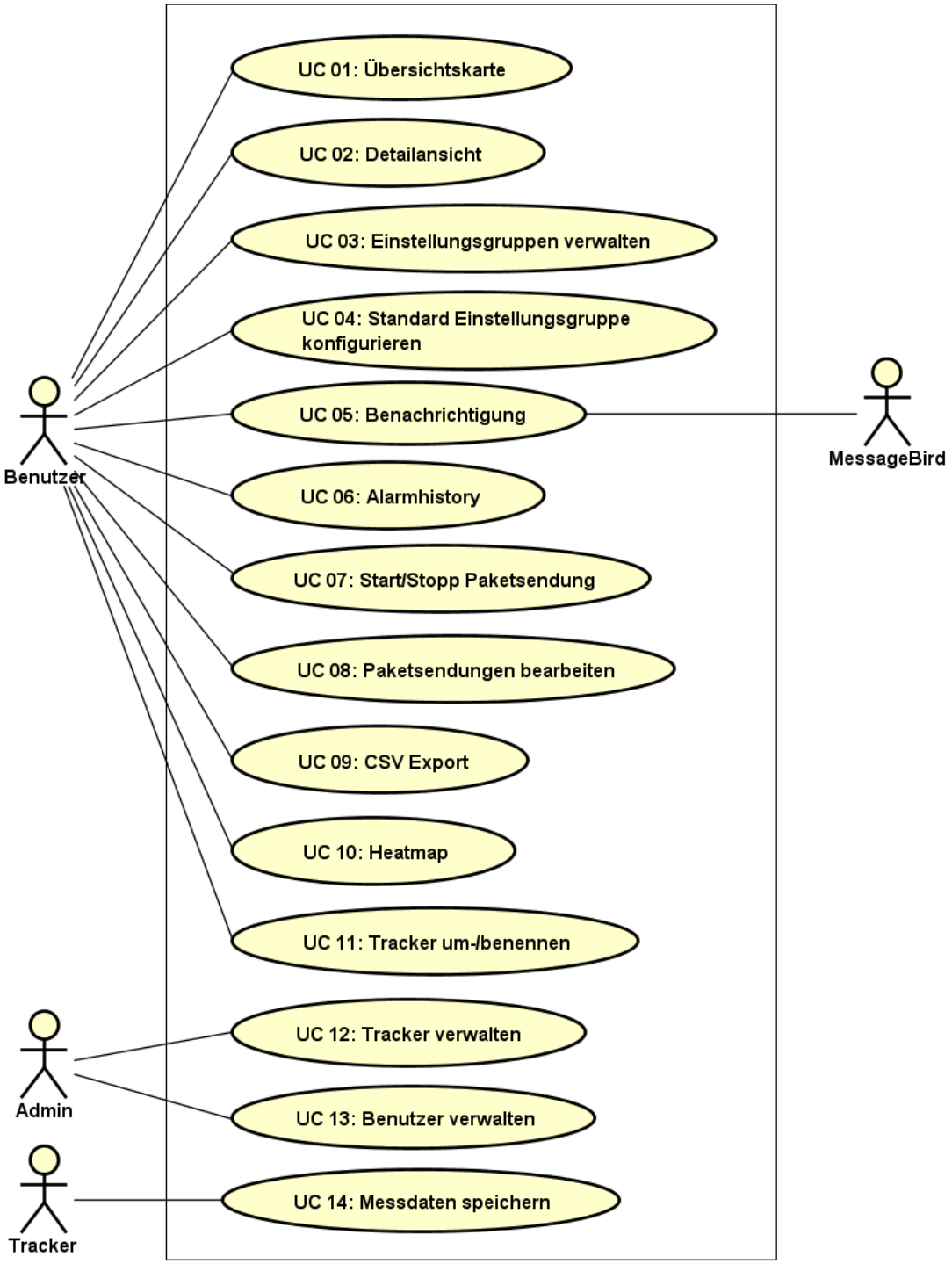

Abbildung 1 Use Case Diagramm

Aktor	Beschreibung
Benutzer	Der Benutzer ist ein Kunde der FPGA Company GmbH.
Admin	Der Administrator ist ein Mitarbeiter der FPGA Company GmbH.
Tracker	Ein von der FPGA Company GmbH entwickeltes Gerät bestückt mit verschiedensten Sensoren (Temperatur, Feuchtigkeit, Beschleunigung, …).
MessageBird	MessageBird ist ein Service welcher das Versenden von Nachrichten über Email, SMS und Voice Call ermöglicht.

Listing 1 Use Case Aktoren

4.1.1 Prioritäten

Die Use Cases wurden vom Kunden nach drei Stufen priorisiert. Dabei steht Priorität 1 für Must-have Features, ohne die das System keinen Sinn macht, Priorität 2 für wichtige und Priorität 3 für optionale Features. Die Prioritäten der einzelnen Use Cases sind in Listing 2 ersichtlich.

Use Case	Priorität
UC 01: Übersichtskarte	1
UC 02: Detailansicht	1
UC 03: Einstellungsgruppen verwalten	3
UC 04: Standard Einstellungsgruppe konfigurieren	2
UC 05: Benachrichtigung	2
UC 06: Alarmhistory	2
UC 07: Start/Stopp Paketsendung	1
UC 08: Paketsendungen bearbeiten	3
UC 09: CSV Export	3
UC 10: Heatmap	2
UC 11: Tracker um-/benennen	2
UC 12: Tracker verwalten	1
UC 13: Benutzer verwalten	1
UC 14: Messdaten speichern	1

Listing 2 Use Case Prioritäten

4.2 Nichtfunktionale Anforderungen

Neben den funktionalen gibt es auch einige nichtfunktionale Anforderungen, denen das System entsprechen soll.

Sicherheit

- Nur autorisierte Benutzer können die Daten des Systems einsehen oder verändern.
- Die Benutzer (mit Ausnahme der Administratoren) können jeweils nur die Daten der ihnen zugewiesenen Tracker einsehen und verändern.
- Nur Administratoren sind in der Lage neue Benutzer anzulegen und die Daten aller Tracking Geräte zu bearbeiten.
- Nur registrierte Geräte dürfen in der Lage sein, Sensordaten an das System zu senden.

Performance

- Das Laden der Übersichtskarte und der darauf angezeigten Tracker soll nicht länger als 2 Sekunden dauern, für bis zu 100 Tracker.
- Das Laden der Detailansicht und der Positions-/Messdaten, die in deren Karte und Charts angezeigt werden, soll nicht länger als 4 Sekunden dauern, für Paketsendungen die bis zu 5 Tage dauern. Dies gilt für das erste laden der Detailansicht, wie auch für das aktualisieren der Messdaten bei der Auswahl einer anderen Paketsendung.

Bedienbarkeit

- Die Applikation ist für Desktops ausgelegt uns soll darauf entsprechend gut bedienbar sein.

Datenpersistenz

- Die Daten der Tracking Geräte, welche beim System ankommen, müssen zuverlässig abgespeichert werden.

5 Domainmodell

Aus den, in Kapitel 4 beschriebenen, Anforderungen ist das in Abbildung 2 ersichtliche Domainmodell entstanden. Es besteht aus insgesamt sieben Komponenten.

Gerät

Die wohl wichtigste Komponente, um die sich das ganze Projekt dreht, ist das Gerät/der Tracker. Die Tracker werden über ihre *IMSI* (International Mobile Subscriber Identity) und *IMEI* (International Mobile Equipment Identity) identifiziert.

Sensor

In jeden Tracker sind verschiedene Sensoren eingebaut, die beispielsweise die Temperatur, Feuchtigkeit oder Beschleunigung in alle drei Achsen messen. Von jedem dieser Sensoren muss bekannt sein, was er misst (z.B. Temperatur) und in welcher Einheit (z.B. °C).

Messpunkt

Die von den Sensoren gemessenen Werte stellen Messpunkte dar. Jeder Messpunkt gehört genau einem Sensor an und enthält den gemessenen Wert und den genauen Zeitpunkt, wann dieser gemessen wurde.

Paketsendung

Ein Tracker kann mehrmals verwendet werden, sprich mehrmals zusammen mit einem Paket verschickt werden. Um seine Messdaten sinnvoll auswerten zu können, wurde das Konzept der Paketsendung eingeführt. Eine Paketsendung stellt die Lieferung eines Paketes inklusive eines darin enthaltenen Trackers von A nach B dar. Dabei werden jeweils der Start- und Endzeitpunkt der Lieferung und die für diese Lieferung festgelegten Grenzwerte festgehalten. Die Grenzwerte müssen pro Lieferung gespeichert werden, da sich diese von Lieferung zu Lieferung je nach Paketinhalt ändern können und die komplette Chronik der Grenzwerte abrufbar sein muss.

Einstellungsgruppe

Oft werden mehrere Lieferungen mit demselben Inhalt aber verschiedenen Trackern verschickt. Das heisst für alle diese Tracker sollen dieselben Grenzwerte gelten. Um nicht jeden einzelnen Tracker separat konfigurieren zu müssen, sollen diese unterschiedlichen Einstellungsgruppen zugewiesen werden können, über die dann alle Tracker gleichzeitig konfiguriert werden. In den Einstellungsgruppen wird abgesehen von den Grenzwerten für die einzelnen Sensoren auch ein Sendeintervall festgehalten, welches bestimmt wie oft ein Tracker seine Daten an den Server übermittelt.

Benutzer

Die Tracker und Einstellungsgruppen gehören jeweils zu einem Benutzer. Jeder Benutzer kann nur seine eigenen Tracker und Einstellungsgruppen sehen und verwalten. Die Ausnahme bilden die Administratoren, denen es möglich ist die Daten aller Tracker einzusehen. Die Administratoren sind auch die einzigen, die neue Benutzer anlegen können. Eine selbständige Registrierung ist momentan nicht vorgesehen.

Benachrichtigungskanal

Über- bzw. unterschreitet ein Messwert einen für den entsprechenden Sensor festgelegten Grenzwert, soll der Besitzer dieses Trackers benachrichtigt werden. Ob und über welchen Kanal (Anruf, E-Mail, SMS) diese Benachrichtigung erfolgen soll, kann vom Benutzer bestimmt werden.

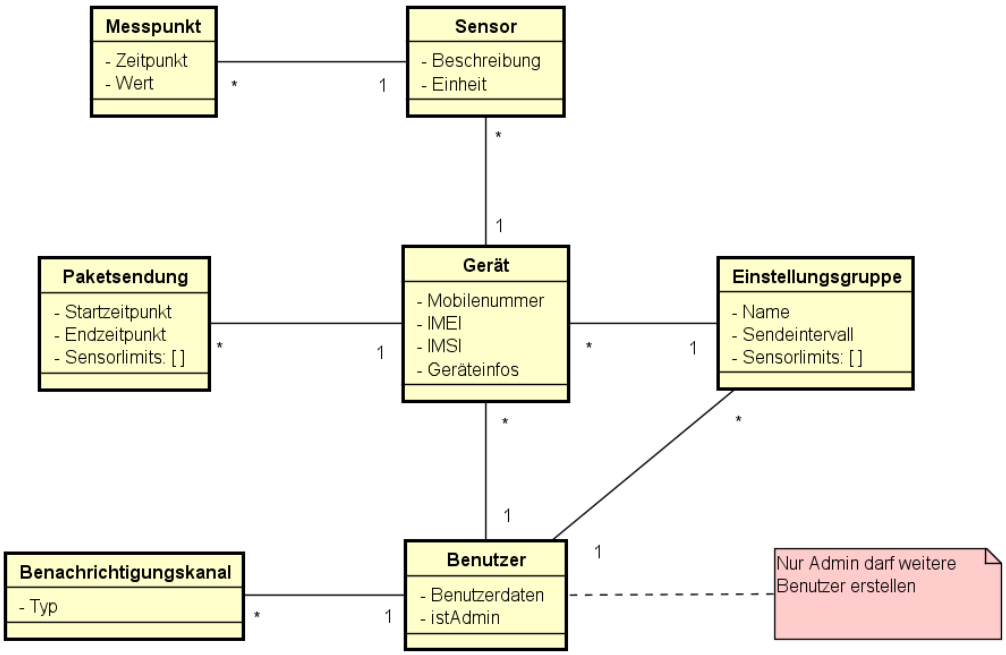

Abbildung 2 Domainmodell

6 Systemarchitektur

Wie in Abbildung 3 dargestellt, umfasst das System, den Tracker ausgenommen, insgesamt fünf Komponenten. Was deren Funktion ist, wie und warum sie so designt wurden, ist im Folgenden erläutert.

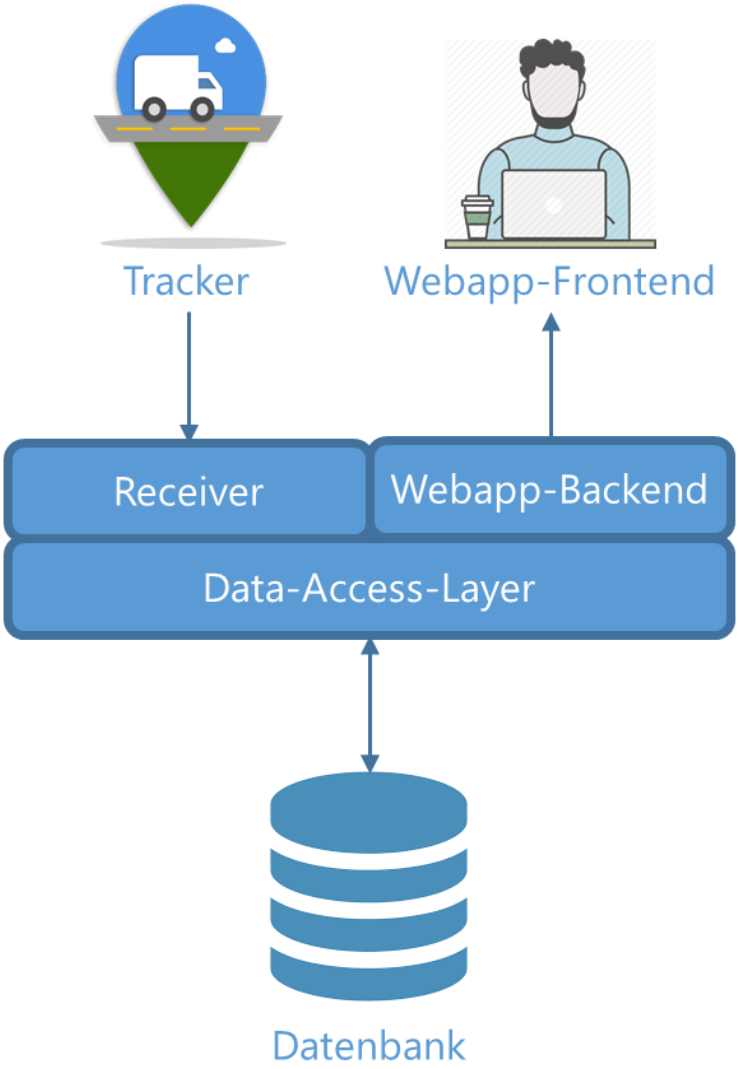

Abbildung 3 Systemarchitektur

6.1 Datenbank

Die Datenbank ist für die zentrale Ablage der erfassten Tracker-Daten und Anwendungsdaten wie Benutzerkonten zuständig. Sie wurde aus den im Folgenden erklärten Gründen mit der dokumentenorientierten NoSQL Datenbank MongoDB umgesetzt.

Datenformat

Die Tracker versenden ihre Messdaten im *JSON*-Format, welches auch im Frontend benutzt wird. Um die Daten nicht beim Speichern und Auslesen aus der Datenbank extra ummappen zu müssen, wurde entschieden eine dokumentenorientierte Datenbank einzusetzen.

Impedance Mismatch

Ein weiterer Grund dafür, eine dokumentenorientierte Datenbank einzusetzen, ist die "Impedance Mismatch" Problematik. Diese bezeichnet den Unterschied zwischen der Struktur von objektorientierten und Datenbank Daten [1]. Bei SQL-Datenbanken ist der "Impedance Mismatch" gross, weil sich relationale Datenmodelle nicht auf natürliche Weise auf objektorientierte Datenmodelle mappen lassen. Dadurch ergibt sich für Anwendungsentwickler ein hoher kognitiver Overhead. Aber bei dokumentenorientierten Datenbanken ist der "Impedance Mismatch" klein, denn objektorientierte und dokumentenorientierte Datenmodelle ähneln sich sehr. Entsprechend ist auch der kognitive Overhead für Anwendungsentwickler niedrig [2].

Datenbanktreiber

Das Play Framework ist von Grund auf vollkommen asynchron aufgebaut. Deshalb wäre es ideal, wenn auch die Datenbankzugriffe asynchron wären. Die meisten Datenbanktreiber sind aber synchron und entsprechend sind die Datenbankzugriffe mit diesen Treibern blockierend. Es gibt jedoch eine bemerkenswerte Ausnahme namens ReactiveMongo, welcher asynchron mit MongoDB kommuniziert [3].

6.1.1 Datenmodell

In Abbildung 4 werden die Strukturen der Dokumente und ihre Relationen dargestellt. Die Verwendungszwecke der jeweiligen Dokumente ist genauer in Kapitel 5 erläutert. Ein Sternchen vor einem Feldnamen bedeutet, dass es ein erforderliches Feld ist. Die anderen Felder können null gesetzt werden oder undefined sein. Das Datenmodell wurde so entworfen, dass das Frontend möglichst wenige Requests durchführen muss.

Namenskonvention

Da der Datentausch über JSON erfolgt und es keine offizielle Namenskonvention für JSON gibt, wurden folgende Kandidaten evaluiert [4]:

- Pascal Case: SettingGroup
- Camel Case: settingGroup
- Snake Case: setting_group
- Kebab Case: setting-group

Weil bei Scala und JavaScript bereits Camel Case als Namenskonvention verwendet wird, wurde entschieden auch für das Datenmodell Camel Case zu verwenden.

Feldtypen

Wenn der Inhalt von einem Feld in "Quotes" verpackt ist, dann ist das Feld ein String. Dies ist auch bei BSONObjectId der Fall. Alle Zahlenwerte sind Integers, abgesehen von folgenden Ausnahmen.

Die folgenden Felder enthalten Long Werte:

- start
- end
- timestamp
- lastUpdate
- created

Die folgenden Felder enthalten Floating Points:

- upperLimit
- lowerLimit
- value
- lat
- lng

Reading Value

Die Positionsdaten werden gleich wie Messpunkte gehandhabt. Daher kann das Feld "value" vom Reading-Dokument entweder ein Double Wert oder ein Positions-Objekt sein. Falls es ein Positions-Objekt ist, dann sieht es wie in Listing 3 aus. Ansonsten sieht es wie in Abbildung 4 aus.

```
{
  "_id": "BSONObjectId",
  "timestamp": 1522145303536,
  "value": {
    "lat": 47.223673,
    "lng": 8.817406
  },
  "sensorId": 100,
  "deviceId": "BSONObjectId",
  "shipmentId": "BSONObjectId"
}
```

Listing 3 Location Reading

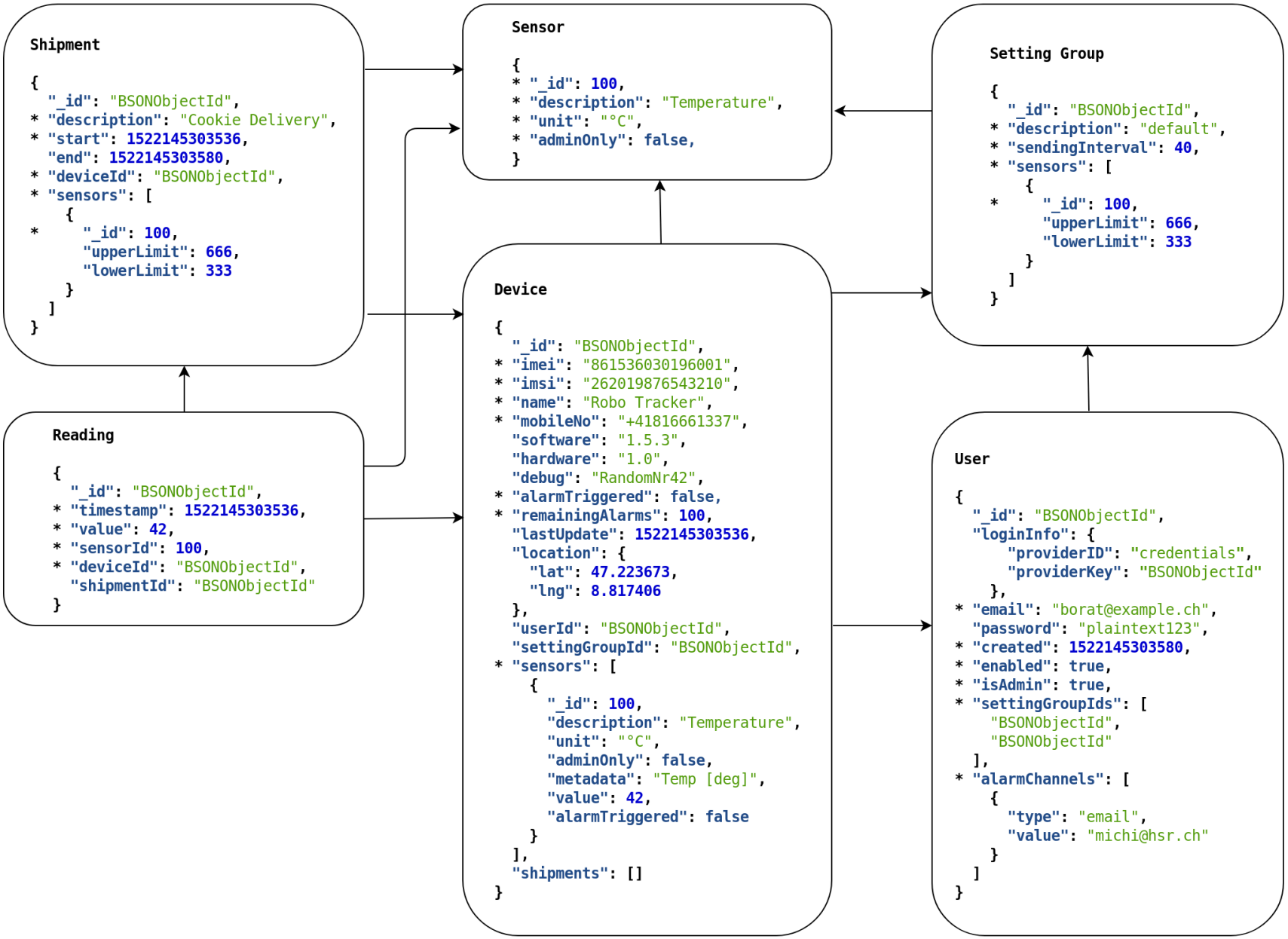

Abbildung 4 Datenmodell

6.2 Receiver

Der Receiver ist für das Empfangen, Validieren und Abspeichern von Tracker-Daten zuständig. Dass der Receiver mit Play Framework umgesetzt werden muss, wurde bereits von Betreuer festgelegt, weil man im HiveWatch-Projekt bereits gute Erfahrungen damit gemacht hatte.
Ursprünglich wollte man den Receiver vom HiveWatch-Projekt übernehmen, da die Tracker die gleichen Daten zusendet. Aus verschiedenen Gründen kann man den Receiver aber nicht einfach übernehmen. Ein Grund ist, dass ein anderes Datenbanksystem und ein anderes Datenmodell verwendet werden. Dazu hat dieses Projekt das Konzept von Paketsendungen (Shipments), welches das HiveWatch-Projekt nicht kennt. Dieses erfordert eine spezielle Aufbereitung der Daten. Deshalb war es erforderlich, einen eigenen Receiver zu implementieren.
Der Receiver ist in Scala geschrieben und verwendet MacWire Dependency Injection, welche unter 6.3.1 Dependency Injection genauer beschrieben wird. Da das Play Framework und die Libraries ReactiveMongo, MacWire und Cats auch im Webapp-Backend verwendet werden, werden diese unter 6.3.2 Technologien genauer beschrieben.

6.2.1 Input-Daten

Die Input-Daten müssen der in Listing 4 gezeigten Struktur entsprechen.

Header:
```
Content-Type: application/json;
```

Body:
```
{
  "id": "IMSI/IMEI",
  "software": "0.45",
  "hardware": "Revision 42",
  "latitude": "47.2261837",
  longitude": "8.8183046",
  "debug": "random facts",
  "101": {
    "metadata": "Temperature [deg]",
    "alarm": "triggered / 30",
    "data": [
      {
        "time": "34050",
        "value": "21.833"
      },
      {
        "time": "15360",
        "value": "20.001"
      }
    ]
  }
}
```

Listing 4 Beispiel Input-Daten

Generelle Infos

Im HTTP Header muss "application/json" als Content-Type eingetragen sein. Die Identifikation der Tracker geschieht über das Feld "id", welches aus IMSI, Forward-Slash und IMEI besteht. Die Felder "software" und "hardware" enthalten jeweils die aktuelle Software bzw. Hardware Version des Trackers. Das Feld "debug" enthält Debugging Information für die FPGA Company GmbH. Die Positionsdaten erhaltet man über die Felder "latitude" (Breitengrad) und "longitude" (Längengrad).

Sensor

Jedes Sensor Objekt hat als Key seine Sensor-ID, z.B. "101", eingetragen. Diese muss ein Integer sein. Das Sensor Objekt muss mindestens die Felder "metadata" und "data" enthalten.

Das Feld "metadata" sollte eine beschreibende Information über den Sensor enthalten. Idealerweise sollte es in folgendem Format aufgebaut sein:

"Sensorname [Einheit]"

Das Feld "data" besteht aus einem Array von Messobjekten. Dieses Messobjekt ist wiederum ein Paar aus "time" und "value". Das Feld "time" enthält die relative Zeit in Millisekunden in Bezug auf den Sendezeitpunkt der Daten. Die Zeitangabe muss ein gültiger Long Wert sein. Das Feld "value" enthält den effektiven Messwert zum angegeben Zeitpunkt. Dieser muss ein gültiger Double Wert sein.

6.2.2 Paketdiagramm

In Abbildung 5 wird das Paketdiagramm des Receivers dargestellt. Es gibt nur einen Contoller, welcher für das Empfangen der Input-Daten zuständig ist. Die Klasse BootstrapLoader ist für die Dependency Injection verantwortlich. Der Data-Access-Layer (common) ist im Kapitel 6.4 genauer beschrieben.

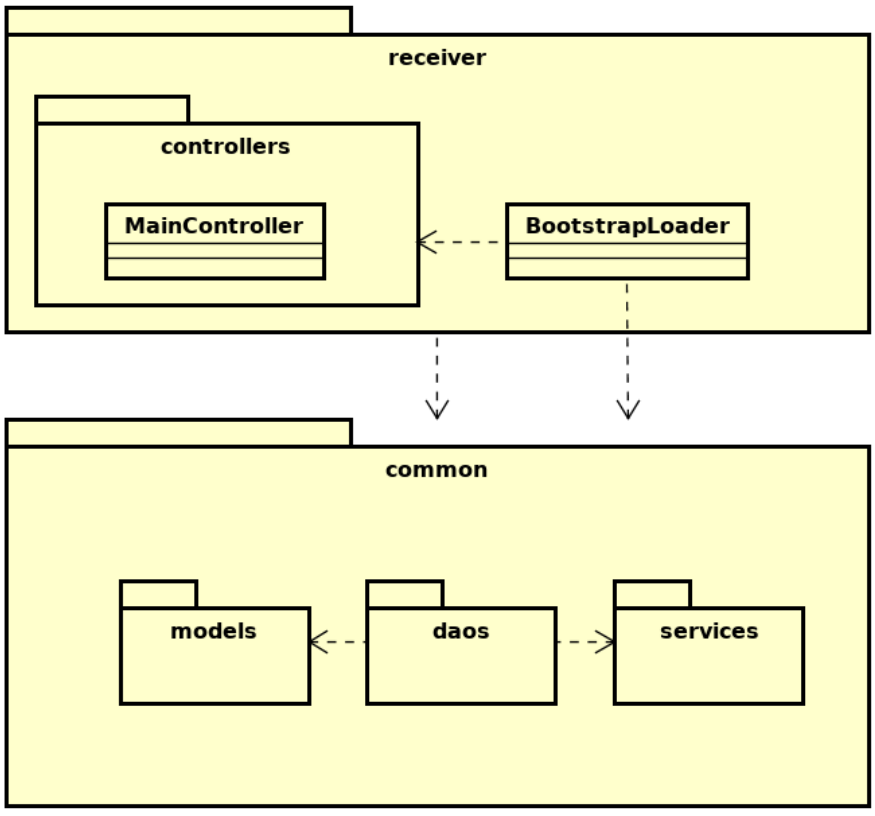

Abbildung 5 Paketdiagramm Receiver

6.3 Webapp-Backend

Das Webapp-Backend bietet eine *REST* API für das Frontend an. Es ist sozusagen das Rückgrat vom Frontend, da es die Daten für das Frontend aufbereitet. Ausserdem überprüft das Backend bei jedem Request, ob die Authentifizierung und Autorisierung für den Benutzer stimmt.
Wie beim Receiver war auch hier vorgegeben, dass das Webapp-Backend mit dem Play Framework umzusetzen ist und es ist ebenfalls in Scala geschrieben.

6.3.1 Dependency Injection

Wenn man das Starter Projekt von Play verwendet, wird es standardmässig mit Guice von Google ausgeliefert. Tatsächlich ist Play aber Dependency Injection agnostisch. Oder anders gesagt, man ist nicht gezwungen eine bestimmte Dependency Injection Technik zu verwenden, sondern kann frei wählen.
Es gibt zwei Arten von Dependency Injection (DI). Die erste ist Compile-Time DI, welche während der Kompilierung die Komponenten injiziert. Die zweite ist Run-Time DI, welche zur Laufzeit die Komponenten injiziert. Der Vorteil von Compile-Time DI ist, dass man Fehler frühzeitig beim Kompilieren bemerkt. Dafür muss man sehr viel Code per Hand schreiben. Der Vorzug bei Run-Time DI ist, dass es einfacher zu konfigurieren ist und nur minimalen Boilerplate Code benötigt. Der Nachteil dabei ist aber, dass man Fehler erst bei der Nutzung der Komponenten bemerken würde. Da für uns eine stabile Applikation einen höheren Stellenwert hat, wurde entschieden Compile-Time DI zu verwenden.
Es wurden drei Compile-Time DI Techniken evaluiert. Bei der ersten handelt es sich um Manuelle DI. Dabei müssen die Komponenten selber per Hand konstruiert und anschliessend miteinander verdrahtet werden. Die zweite Technik heisst Cake Pattern. Dort müssen die Komponenten weiterhin manuell konstruiert werden. Die Verdrahtung der Komponenten wird aber vom Scala Compiler übernommen. Trotzdem gibt es am Schluss mehr Boilerplate Code als bei Manueller DI. Die dritte Technik heisst MacWire. MacWire ist eine Library, welche mit Macros die Klassen konstruiert und die benötigten Komponenten verdrahtet. Sie funktioniert gleich wie Manuelle DI, die Handarbeit wird aber von den Macros übernommen. Deshalb wurde entschieden MacWire DI zu nutzen [5].

6.3.2 Technologien

Play Framework

Das Play Framework ermöglicht die Entwicklung von Web Anwendungen in Java oder Scala. Es basiert auf einer leichten, zustandslosen, webfreundlichen Architektur. Play verwendet ein vollständig asynchrones Modell, das auf Akka aufgebaut ist [6]. Da einer der Entwickler bereits Erfahrung in der Entwicklung mit Scala hatte, wurden Receiver, Webapp-Backend und Data-Access-Layer in Scala geschrieben.

ReactiveMongo

ReactiveMongo ist ein Scala Treiber, der nicht-blockierende und asynchrone I/O-Operationen für MongoDB bietet [7]. Er wird verwendet, da er dieselbe Design-Philosophie namens Reactive wie das Play Framework anstrebt.

MacWire

MacWire ist eine leichte und nicht-intrusive Dependency Injection Library für Scala. Durch die Macros von MacWire entfällt der handgeschriebene Code für die Konstruktion und Verdrahtung der Komponenten [8]. Dadurch sieht der Code auch eleganter aus. Weitere Vorteile von MacWire sind, dass es Compile-Time Verifikation anbietet und man eine ähnliche Flexibilität wie mit Google's Guice hat.

Silhouette

Silhouette ist eine Authentifizierungs-Library für Play Framework Anwendungen, die mehrere Authentifizierungsmethoden wie OAuth1, OAuth2, OpenID, CAS, Credentials, Basic Authentication oder eigene Authentifizierungsmethoden unterstützt [9]. Sie wurde von Betreuer empfohlen, da dieser bereits gute Erfahrungen mit Silhouette gemacht hatte. Um keine Abhängigkeiten zu anderen Identity Providern zu haben, wurde entschieden das Credentials Verfahren zu verwenden.

Swagger

Die OpenAPI Spezifikation (OAS), früher als Swagger Spezifikation bezeichnet, ist ein Standard für die Definition von RESTful Schnittstellen. Die OAS ermöglicht es Entwicklern, technologie-unabhängige Schnittstellen zu entwerfen und zu dokumentieren [10].
Da die offizielle Swagger Library für Play mit Java Annotations arbeitet und diese den Code unleserlich machen können, wurde entschieden stattdessen die Swagger Library von iHeartReadio einzusetzen. Diese generiert anhand der Route-Files, den darin enthaltenen Kommentaren und den Modell-Klassen die Spezifikation [11].

Cats

Cats und Scalaz sind Libraries, welche Abstraktionen für die Funktionale Programmierung in der Programmiersprache Scala bieten. Weil Cats eine bessere Dokumentation bietet und auch modularer aufgebaut ist, wurde entschieden Cats zu verwenden [12]. Die Cats Library wird vor allem benötigt, um Option und Future in einer For-Comprehension zu kombinieren, da man standardmässig in einer For-Comprehension nur Option oder nur Future verwenden kann [13].

Akka Quartz Scheduler

Quartz ist eine Job Scheduling Library für Java [14]. Akka-Quartz-Scheduler von Enragedginger nutzt die Konzepte des Quartz Scheduling Systems, um ein robustes und zuverlässiges Scheduling für Akka zu ermöglichen [15]. Es wird benötigt, um die AuthTokens von Silhouette zu bereinigen.

ScalaTest

ScalaTest ist das flexibelste und beliebteste Testing Tool im Scala-Ökosystem, womit man Scala und Java Code testen kann. Durch die umfassende Integration in Tools wie JUnit, Maven, sbt, Mockito, Eclipse und IntelliJ erleichtert ScalaTest die Durchführung von Tests [16]. Mithilfe von ScalaTest wurden Unit Tests und Integrationstests geschrieben.

Mockito

Mockito ist ein Mocking Framework für Java. Es ermöglicht die bequeme Erstellung von Mocks von Objekten zu Testzwecken [17]. Mit Mockito wird die Datenbank Verbindung gemockt, so dass die Tests auch auf dem CI Server ohne echte Datenbank laufen können.

6.3.3 REST Endpunkte

Grundsätzlich gelten hier auch die üblichen REST API Konventionen. Bei einem GET auf eine Ressourcengruppe wird ein Array von Objekten zurückgegeben und bei einem GET auf eine Ressource wird ein einzelnes Objekt zurückgegeben. Um eine Ressource zu erstellen, muss ein POST und das ganze Objekt an eine Ressourcengruppe gesendet werden. Um eine bestehende Ressource zu aktualisieren, muss ein PUT und das ganze Objekt an eine bestimmte Ressource gesendet werden. Um eine bestehende Ressource zu entfernen, muss ein DELETE auf der Ressource durchgeführt werden [18].
Normalerweise werden Ressourcen und Ressourcengruppen anhand der Namensgebung unterschieden. Meistens haben Ressourcen am Schluss eine lange ID Nummer und Ressourcengruppen haben am Schluss eine Substantive, welche im Plural ist. Um die Unterscheidung klarer zu machen, wurde, wie in Listing 5 gezeigt, bei Ressourcengruppen immer ein abschliessender

Slash angefügt, welcher bei Ressourcen nie vorhanden ist. Dieses Modell entspricht auch dem Verhalten des Unix Dateisystems, wo die Verzeichnisse mit einem abschliessenden Slash gekennzeichnet werden. So ist es auch in der ursprünglichen Dissertation von REST definiert [19].

```
/api/v1/resource
/api/v1/resourcegroup/
```

Listing 5 Unterscheidung von Ressourcen & Ressourcengruppen

Bei Requests muss der Body jeweils leer sein, oder valides JSON enthalten. Ist ein Request erfolgreich, wird immer die betroffene Ressource, bzw. ein Array der betroffenen Ressourcen, in Form von JSON zurückgegeben. Falls die Struktur vom erwarteten Body nicht übereinstimmt, wird der Fehlercode "400 Bad Request" zurückgegeben. Abgesehen vom Login Endpunkt benötigen alle Endpunkte ein gültiges Session-Token. Ist das Session-Token nicht mehr gültig, oder gar nicht vorhanden, wird der Fehlercode "401 Unauthorized" zurückgegeben. Beginnt ein Endpunkt mit Präfix "/me", dann erhält man über diesen Endpunkt benutzerspezifische Objekte. Die Swagger-Dokumentation der Endpunkte ist unter https://hivetrack.ch/swagger einsehbar. Dafür sind Adminrechte erforderlich.

Einstellungsgruppe / SettingGroup

GET /api/v1/me/settingGroups/

Dieser Endpunkt liefert die persönlichen SettingGroups.

Antwortcodes

200	ok

GET /api/v1/me/settingGroups/{id}

Dieser Endpunkt liefert die SettingGroup mit der angegebenen ID. Um darauf zuzugreifen, muss man entweder der Besitzer der SettingGroup oder Administrator sein. Falls die notwendige Berechtigung fehlt oder die angeforderte SettingGroup nicht existiert, wird 404 zurückgegeben.

Antwortcodes

200	ok
404	not found

PUT /api/v1/me/settingGroups/{id}

Über diesen Endpunkt kann man eine SettingGroup aktualisieren. Dafür muss man der Besitzer der SettingGroup sein. Falls die notwendige Berechtigung fehlt oder die SettingGroup nicht existiert, wird 404 zurückgegeben.

Antwortcodes

200	ok
400	invalid setting group format
404	not found

Sensor

GET /api/v1/sensors/

Dieser Endpunkt liefert alle Sensoren.

Antwortcodes

200	ok

POST /api/v1/sensors/

Über diesen Endpunkt lassen sich neue Sensoren erstellen. Dafür werden Adminrechte benötigt.

Antwortcodes

201	created
400	invalid sensor format
403	forbidden
409	id already exists
500	error writing in DB

PUT /api/v1/sensors/{id}

Über diesen Endpunkt kann ein existierender Sensor aktualisiert werden. Hierzu werden Adminrechte benötigt. Zusätzlich zur globalen Sensorliste werden auch die Sensoren in allen Device-Objekten aktualisiert.

Antwortcodes

200	ok
400	invalid sensor format
403	forbidden
404	not found
500	error updating devices in DB

Paketsendung / Shipment

POST /api/v1/me/shipments/

Über diesen Endpunkt kann ein neues Shipment erstellt werden. Dabei werden alle "alarmTriggered" Felder auf dem dazugehörigen Device auf false zurückgesetzt. Zusätzlich wird die aktuelle Position des Devices als neues Location Reading für das neue Shipment eingefügt. Falls man nicht der Besitzer des Devices ist, wird 403 zurückgegeben.

Antwortcodes

201	created
400	invalid shipment format
403	forbidden
500	error writing in DB

PUT /api/v1/me/shipments/{id}

Über diesen Endpunkt lässt sich ein existierendes Shipment aktualisieren. Falls man nicht der Besitzer des dazugehörigen Devices ist, wird 403 zurückgegeben.

Antwortcodes

200	ok
400	invalid shipment format
403	forbidden
404	not found

Messpunkt / Reading

GET /api/v1/me/shipments/{shipmentId}/readings/

Dieser Endpunkt liefert alle Readings eines bestimmten Shipments. Dafür muss man entweder der Besitzer des dazugehörigen Devices sein oder Adminrechte haben. Falls die notwendige Berechtigung fehlt oder das angeforderte Shipment nicht existiert, wird 404 zurückgegeben.

Antwortcodes

200	ok
404	not found

Tracker / Device

Die meisten Device-Endpoints enthalten den Query Parameter "withShipments". Mit diesem kann eingestellt werden, ob die Devices mit oder ohne Shipments geliefert werden sollen. Standardmässig werden die Devices ohne Shipments geliefert.

GET /api/v1/me/devices/?withShipments=false

Dieser Endpunkt liefert die persönlichen Devices.

Antwortcodes

200	ok

GET /api/v1/me/devices/{id}?withShipments=false

Dieser Endpunkt liefert das Device mit der angegebenen ID. Hierzu muss man entweder der Besitzer des Devices oder Administrator sein. Falls die notwendige Berechtigung fehlt oder das angeforderte Device nicht existiert, wird 404 zurückgegeben.

Antwortcodes

200	ok
404	not found

GET /api/v1/devices/?withShipments=false

Dieser Endpunkt liefert alle registrierten Devices. Hierzu werden Adminrechte benötigt.

Antwortcodes

200	ok
403	forbidden

POST /api/v1/devices/

Über diesen Endpunkt können mehrere neue Devices gleichzeitig erstellt werden. Dazu werden Adminrechte benötigt. Der Request Body muss dabei wie in Listing 6 gezeigt aufgebaut sein.

```
[
  {
  * "imei": "string",
  * "imsi": "string",
  * "mobileNo": "string",
  * "name": "string"
  }
]
```

Listing 6 Modell InsertDevice

Antwortcodes

201	created
400	invalid insert device format
403	forbidden
500	error writing in DB

PATCH /api/v1/devices/{id}

Über diesen Endpunkt lässt sich ein existierendes Device partiell aktualisieren. Als Admin können IMSI, IMEI, Name, UserId und SettingGroupId geändert werden. Als normale Benutzer lässt sich nur der Name ändern. Es werden nur gesetzte Felder aktualisiert. Oder anders gesagt, falls ein Feld null oder undefined ist, dann wird dieses Feld nicht aktualisiert. Beim Aktualisieren der UserId muss auch die SettingGroupId mitgesendet werden.

Antwortcodes

200	ok
400	invalid patch device format
403	forbidden
404	not found

Benutzer / User

GET /api/v1/users/

Dieser Endpunkt liefert alle Users. Dazu werden Adminrechte benötigt.

Antwortcodes

200	ok
403	forbidden

POST /api/v1/users/

Über diesen Endpunkt können neue User erstellt werden. Hierzu werden Adminrechte benötigt. Falls die E-Mail Adresse bereits existiert, wird 409 zurückgegeben. Der Request Body muss wie in Listing 7 gezeigt aufgebaut sein.

```
{
* "email": "user@hsr.ch",
* "password": "StrongPassword?42",
  "isAdmin": false,
  "enabled": true
}
```

Listing 7 Modell RegisterData

Antwortcodes

201	created
400	invalid register format
403	forbidden
409	email already exists

PUT /api/v1/users/{id}

Über diesen Endpunkt kann ein existierender User aktualisiert werden. Als Admin darf man alle User aktualisieren und als normaler Benutzer nur seine eigenen Daten. Nur als Admin können die Attribute "enabled" und "isAdmin" verändert werden. Falls die E-Mail Adresse aktualisiert wird, diese aber bereits existiert, wird 409 zurückgegeben.

Antwortcodes

200	ok
400	invalid user format
403	forbidden
404	not found
409	email already exists

Authentisierung

GET /api/v1/authUser

Wenn man ein gültiges Session-Token hat, erhält man über diesen Endpunkt das User-Objekt.

Antwortcodes

200	ok
401	Unauthorized

POST /api/v1/session

Über diesen Endpunkt kann man eine Session erstellen, sprich sich einloggen. Falls die E-Mail oder das Passwort nicht übereinstimmt, wird 401 zurückgegeben. Wenn der Benutzer deaktiviert ist, wird 423 zurückgegeben. Der Request Body muss wie in Listing 8 gezeigt aufgebaut sein.

```
{
* "email": "user@hsr.ch",
* "password": "StrongPassword?42",
* "rememberMe": false
}
```

Listing 8 Modell LoginData

Antwortcodes

201	created
400	invalid login format
401	wrong credentials
423	user is disabled

DELETE /api/v1/session

Über diesem Endpunkt kann man seine Session löschen, sprich sich ausloggen.

Antwortcodes

200	ok

6.3.4 Paketdiagramm

In Abbildung 6 wird das Paketdiagramm vom Webapp-Backend aufgezeigt. Der HomeController ist für die Auslieferung des Frontends zuständig. Die restlichen Controllers sind für das Handling der jeweiligen Datenmodelle verantwortlich. Im Paket utils sind Hilfsklassen für die Authentifizierung enthalten. Die Klasse index im Paket views ist ein Scala Template für Generierung der HTML Seite. Die Klasse BootstrapLoader ist für die Dependency Injection verantwortlich. Der Data-Access-Layer (common) ist im Kapitel 6.4 genauer beschrieben.

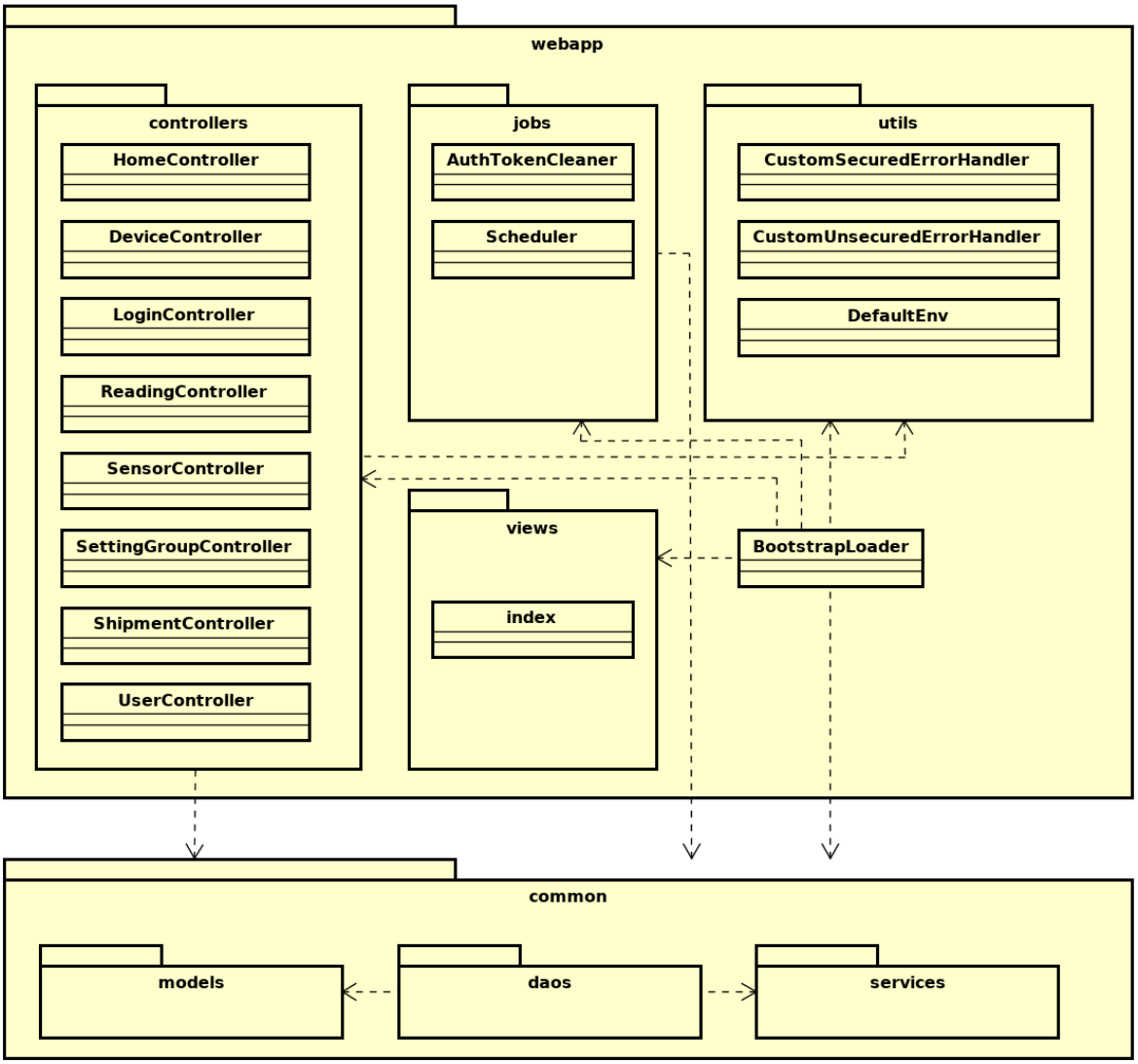

Abbildung 6 Paketdiagramm Webapp-Backend

6.4 Data-Access-Layer

Um das Don't Repeat Yourself (DRY) Prinzip nicht zu verletzten, wird der Data-Access-Layer vom Receiver und Webapp-Backend als gemeinsame Komponente gepflegt. Da man zukünftig allenfalls weitere Gemeinsamkeiten teilen könnte, heisst die Komponente im Quellcode "common".

6.4.1 DAO Design Pattern [20]

Das grundsätzliche Problem ist, dass von Zeit zu Zeit die Zugriffsmechanismen auf der Persistierungsebene ändern können, oder sogar auf eine andere Art von Persistierung gewechselt werden kann.
Die Lösung dieses Problems ist die Verwendung eines Data Access Objects (DAO). Durch dieses wird der gesamte Zugriff auf die Datenquelle abstrahiert und gekapselt. Das DAO verwaltet die Verbindung zur Datenquelle, um Daten zu erhalten und zu speichern.

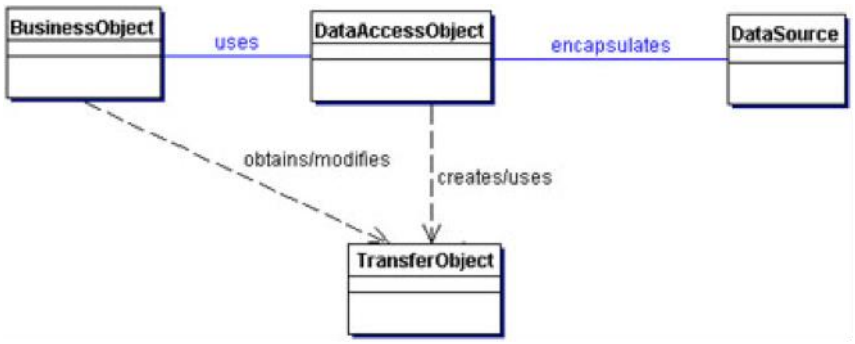

Abbildung 7 Klassendiagramm DAO

BusinessObject

Das BusinessObject repräsentiert das Objekt, das Zugriff auf die Datenquelle benötigt, um Daten zu erhalten und zu speichern. In unserem Fall sind die Controllers die BusinessObjects.

DataAccessObject

Das DataAccessObject abstrahiert die zugrundeliegende Datenzugriffs-Implementierung für das BusinessObject, um Zugriffe auf die Datenquelle zu ermöglichen. Das BusinessObject delegiert ausserdem Vorgänge zum Laden und Speichern von Daten an das DataAccessObject. In unserem Fall sind die Repositories die DataAccessObjects.

DataSource

Dies stellt eine Datenquellen-Implementierung dar, was beispielsweise eine Datenbank sein kann. In unserem Fall ist der DbService die DataSource.

TransferObject

Das TransferObject wird als Datenträger verwendet um Daten zwischen Business- und DataAccessObject auszutauschen. In unserem Fall sind die Models die TransferObjects.

6.4.2 Paketdiagramm

In Abbildung 8 wird das Paketdiagramm vom Data-Access-Layer dargestellt. Im Diagramm sind abstrakte Klassen und Traits *kursiv* geschrieben. DbService ist das Interface für Datenbankzugriffe und MongoService ist die konkrete Implementation für Datenbankzugriffe. Key Namen von Dokumenten sind als Konstanten in DbKeys enthalten. GlobalFormats enthält generelle Formate, welche für das Mapping von Scala Objekten zu JSON oder umgekehrt zuständig sind.

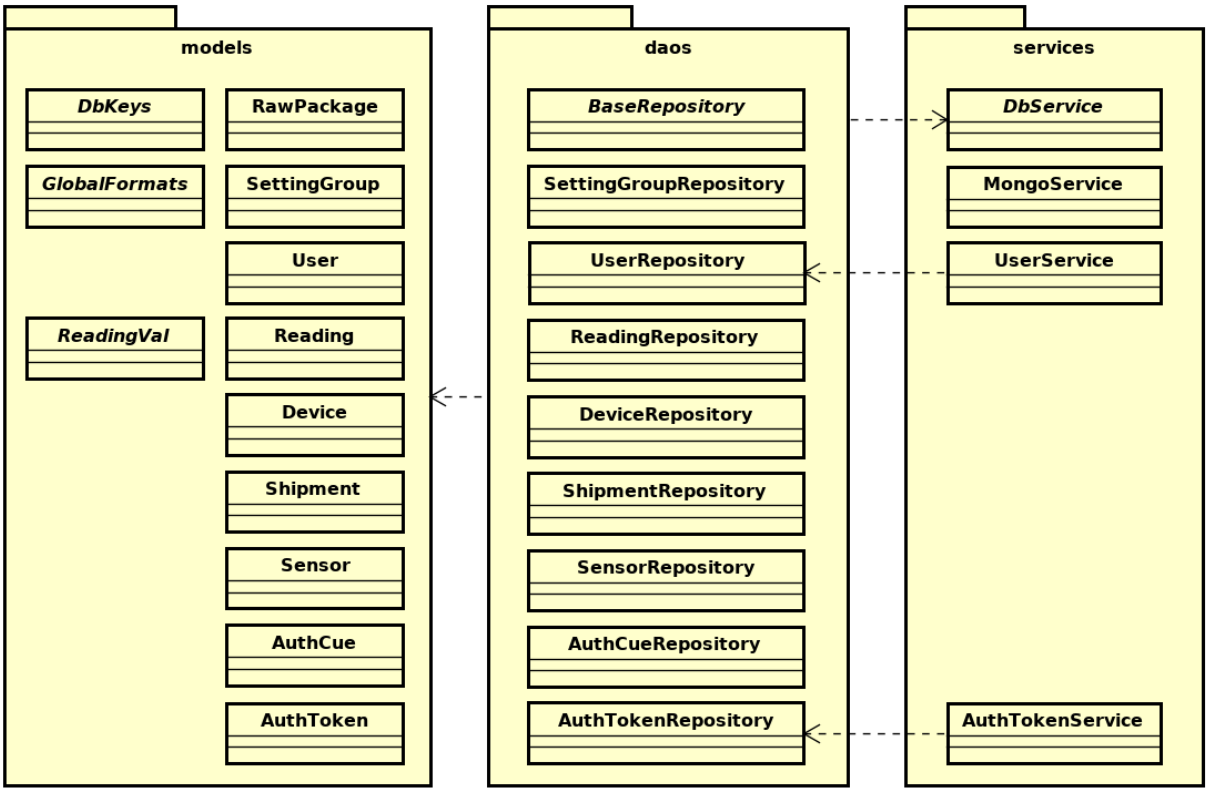

Abbildung 8 Paketdiagram Data-Access-Layer

6.5 Webapp-Frontend

Das Webapp-Frontend ist für die Darstellung der Tracker-Daten zuständig und erhält diese über die REST API des Backends.

6.5.1 Technologien

Dass das Frontend mit React-Redux umzusetzen ist, wurde bereits vor dem Start des Projekts vom Betreuer festgelegt.

React

React ist eine JavaScript Library für die Erstellung von User Interfaces [21]. Sie ermöglicht es das User Interface in einzelne wiederverwendbare Komponenten aufzuteilen. Jede dieser Komponenten kann aus weiteren Komponenten bestehen und verwaltet jeweils ihren eigenen State.

Redux

Redux ist wie React eine JavaScript Library [22]. Mit ihr wird das State-Management um einen globalen Store erweitert. Daten, für die sich mehrere Komponenten interessieren, können also anstatt in den einzelnen Komponenten im globalen Store verwaltet werden, auf den alle Komponenten Zugriff haben. Für die Verwaltung dieses Stores definiert Redux den in Abbildung 9 dargestellten Datenfluss:

1. Die UI-Komponenten lesen vom Store die für sie interessanten Daten und stellen diese dar.
2. Durch Benutzerinteraktionen oder andere Trigger werden von den UI-Komponenten Actions ausgelöst.
3. Eine Aktion besteht jeweils aus einem Typen, der beschreibt um was für eine Aktion es sich handelt, und den mit der Aktion assoziierten Daten. Ein Beispiel für eine Aktion wäre das Umbenennen eines Trackers, welche den neuen Namen des Trackers enthält.
4. Diese Actions werden dann von den verschiedenen Reducern verarbeitet. Hierbei ist es üblich, dass pro Klasse des Datenmodells ein Reducer existiert. In unserem Fall wäre also ein Tracker-Reducer sinnvoll. Jeder Reducer reagiert immer nur auf die für ihn bestimmten Actions. Es ist auch möglich, dass mehrere Reducer auf die gleiche Aktion reagieren.
5. Die Reducer aktualisieren dann basierend auf den Actions und den in ihnen enthaltenen Daten einen Teil des Stores.
6. Daraufhin werden die von den Änderungen betroffenen UI-Komponenten benachrichtigt und aktualisiert, sodass wieder der aktuelle Zustand des Stores dargestellt wird.

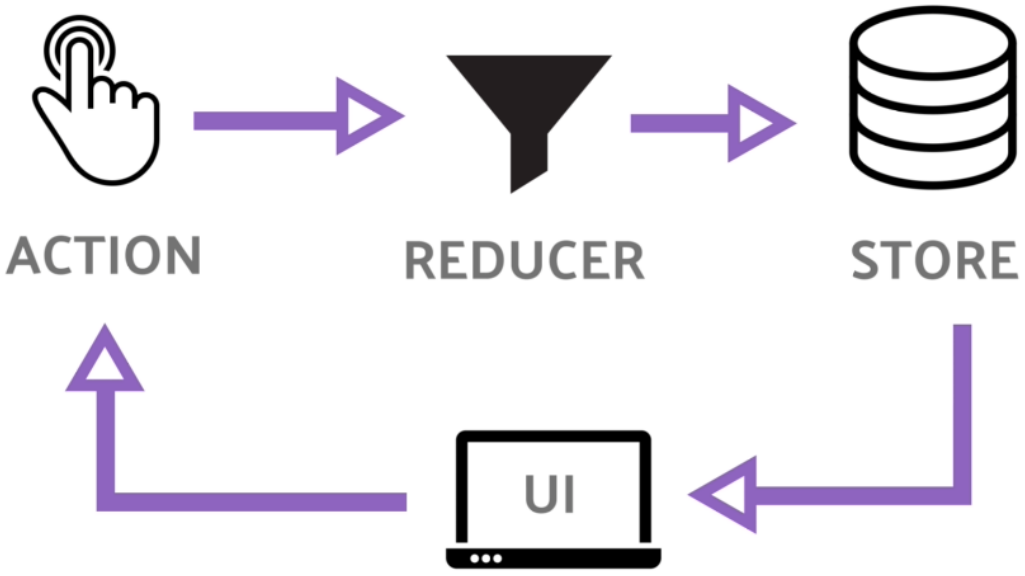

Abbildung 9 Redux Datenfluss

Abgesehen von React und Redux wurden noch die folgenden JavaScript Libraries eingesetzt.

React Redux

Die React Redux Library bildet das Bindeglied zwischen React und Redux [23]. Mithilfe ihrer `connect()`-Funktion können React Komponenten Daten aus dem Redux Store laden und Redux Actions absenden.

Highcharts

Mithilfe von Highcharts können Daten in den verschiedensten Chart-Formen dargestellt werden [24]. In unserem Fall wurden aber nur relativ einfache Linien-Charts für die Darstellung der Sensordaten benötigt. Die Highcharts Library wird hauptsächlich deshalb verwendet, weil sie bereits mit grossem Erfolg im HiveWatch-Projekt eingesetzt wurde.

Highcharts React

Dies ist die offizielle React Wrapper Komponente für Highcharts [25]. Sie erleichtert die Verwendung der Highcharts Library in React.

Material-UI

Material-UI [26] bietet fertig gestylte React Komponenten, die Google's Material Design [27] implementieren. Durch die Verwendung dieser Komponenten sind nur minimale Anpassungen am Styling notwendig, was das UI-Design stark erleichtert.

React Google Maps

Die React Google Maps Library bietet React Wrapper Komponenten für die Google Maps API [28]. Sie werden verwendet um beispielsweise die Übersichtskarte und die darauf befindlichen Positions-Marker für die Tracker anzuzeigen.

React Router

React Router ermöglicht die Verwaltung der verschiedenen Frontend-Routen mithilfe von einfachen React Komponenten [29].

Redux Auth Wrapper

Mittels des Redux Auth Wrappers wird die Authentisierung und Autorisierung gehandhabt [30]. Über ihn kann festgelegt werden ob ein Benutzer auf eine React Komponente zugreifen darf und wohin er weitergeleitet werden soll, wenn dies nicht der Fall ist.

Axios

Axios ist ein *Promise* basierter HTTP Client für den Browser, der verwendet wird um *XMLHttpRequests* an die Backend API zu senden [31].

6.5.2 Routen

Eine Beschreibung der Frontend-Routen ist in Listing 9 zu finden.

Route	Beschreibung
/	Die Übersichtskarte, die dem Benutzer einen Überblick über alle seine Tracker bietet.
/device/[id]	Die Detailansicht eines einzelnen Trackers, auf der der komplette Verlauf der Positions- und Messdaten gezeigt wird.
/trackers	Eine Liste aller Tracker, die nur Administratoren zugänglich ist. Hier werden auch Informationen wie die Software- und Hardware-Version des Trackers angezeigt, von denen normale Benutzer nichts wissen.
/settings	Die Einstellungsseite besteht aus mehreren Tabs, die alle quasi eine eigene Seite darstellen. Darunter befinden sich die Konfiguration der Einstellungsgruppen, die Benutzerverwaltung, die Einstellungen für den eigenen Account und die Registrierung von neuen Trackern.
/login	Das Login, auf das man immer weitergeleitet werden soll, falls man nicht bereits eigeloggt ist.
/swagger	Eine nur für Administratoren zugängliche Route, in der die mit Swagger erstellte Backend API Dokumentation mittels Swagger UI [32] dargestellt wird.

Listing 9 Frontend-Routen

6.5.3 Wireframes

Um dem Kunden früh etwas Handfestes präsentieren zu können und eine solide Diskussionsgrundlage zu schaffen, wurden für die wichtigsten Komponenten Wireframes erstellt. Sie widerspiegeln nicht den aktuellen Stand, da sie nach der Kreierung nicht mehr aktualisiert wurden. Da die Applikation vorerst nur auf Englisch zur Verfügung gestellt werden soll, wurde dies auch bei den Wireframes übernommen.

Übersichtskarte / Overview Map

Die Übersichtskarte, siehe Abbildung 10, soll als Startseite für die Applikation fungieren und dem Benutzer einen schnellen Überblick über alle seine Tracker und deren Zustand ermöglichen.

- Im Mittelpunkt steht die Karte, auf der die aktuellen Positionen der Tracker anhand von Positions-Markern abgebildet sind. Die Marker zeigen anhand ihrer Farbe auch gleich den Zustand des Trackers, bzw. des Paketes, an. Blau bedeutet, dass alles in Ordnung ist. Rot bedeutet, dass während der aktuellen oder der letzten Paketsendung, je nachdem ob der Tracker momentan unterwegs ist oder nicht, mindestens ein Wert ausserhalb der konfigurierten Grenzen gemessen wurde. So kann der Benutzer auf einen Blick erkennen, wie viele seiner verschickten Pakete potentiell defekte Ware enthalten.
- Bewegt man den Cursor über einen Marker werden verschiedene Infos wie Name, IMSI, IMEI und die aktuellen Sensorwerte des Trackers angezeigt.
- Rechts oberhalb der Karte befindet sich ein aufklappbarer Filter. Mit diesem können die Tracker zum Beispiel anhand ihres Zustands gefiltert werden. Dies kann sehr praktisch sein, falls man duzende von Trackern besitzt und die Karte voll von Markern ist.
- Unterhalb der Karte sind noch alle Tracker mit ihrem Namen aufgelistet. Dort kann der Benutzer über die Start- und Stopp-Buttons seine Paketsendungen für die jeweiligen Tracker starten und stoppen. Ist ein Button deaktiviert, bzw. ausgegraut, bedeutet das, dass bereits eine laufende Paketsendung existiert, bzw. eben nicht existiert.
- Durch Klicken eines Markers oder Listeneintrags erreicht man die Detailansicht des entsprechenden Trackers.

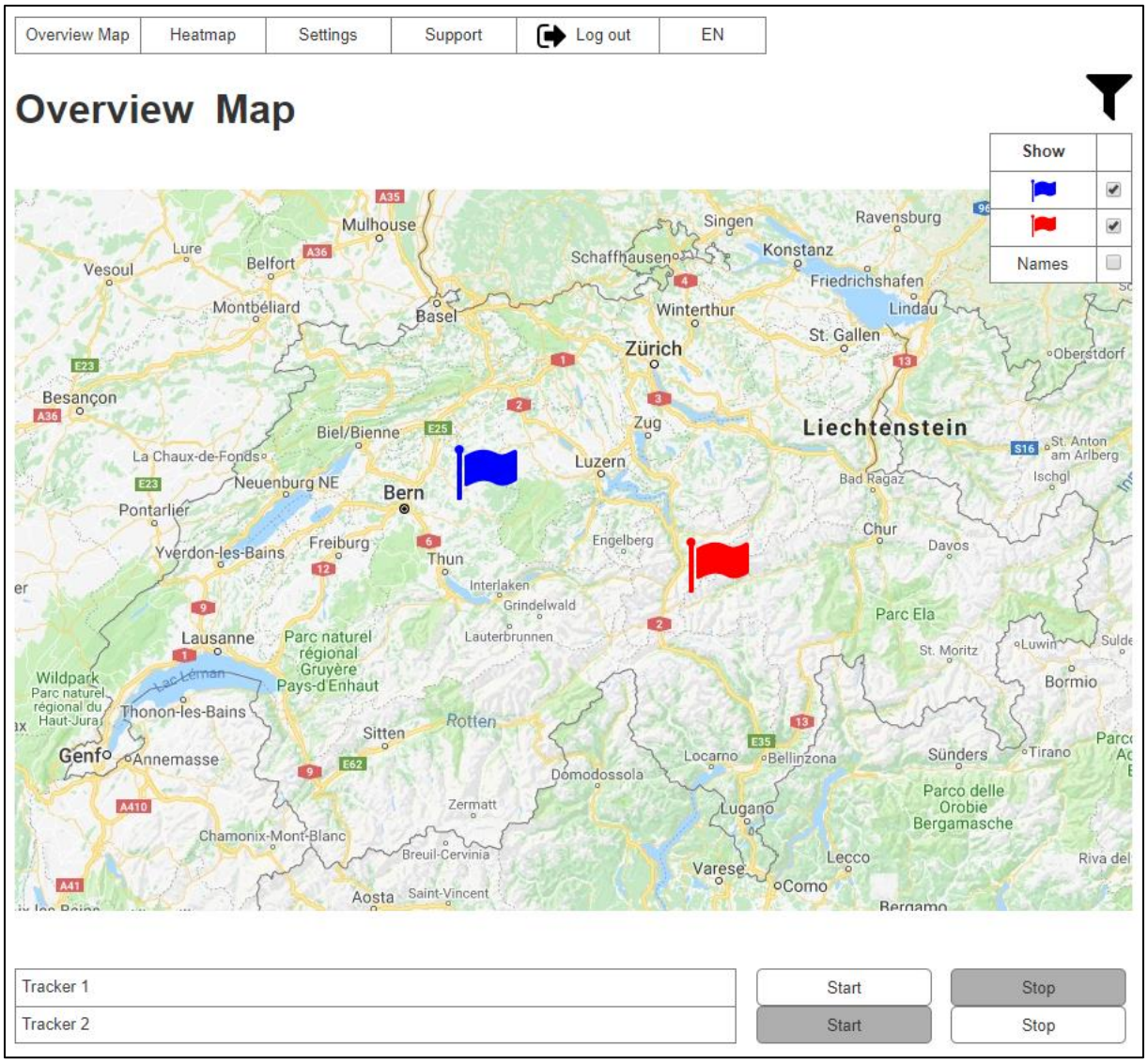

Abbildung 10 Wireframe – Übersichtskarte

Detailansicht / Tracker Details

In der Detailansicht, siehe Abbildung 11, soll der Benutzer den Positionsverlauf und den Verlauf der Sensormesswerte während der einzelnen Lieferungen (engl. Shipments) analysieren können.

- Für die Darstellung des Positionsverlaufs wird auch hier wieder eine Karte verwendet.
- Unterhalb der Karte wird pro Sensor ein Linien-Chart mit den entsprechenden Messwerten gezeigt. Falls für einen Sensor Grenzwerte konfiguriert wurden, werden diese ebenfalls im Chart als horizontale Linien angezeigt und die genauen Grenzwerte rechts neben dem Chart aufgeführt. Die Werte, die ausserhalb dieser Grenzen liegen, werden im Chart rot markiert.
- Mithilfe eines Sliders, im Wireframe Timeline genannt, kann der Positions-Marker auf der Karte, wie auch die Markierungen auf den Charts, bewegt werden. So ist einfach ersichtlich, wann und wo welche Werte gemessen wurden.
- Oberhalb der Karte befindet sich ein Dropdown-Feld, über das auch auf die älteren Lieferungen und deren Messdaten zugegriffen werden kann.
- Analog zur Tracker-Liste in der Übersichtskarte können auch in der Detailansicht Lieferungen gestartet oder gestoppt werden.

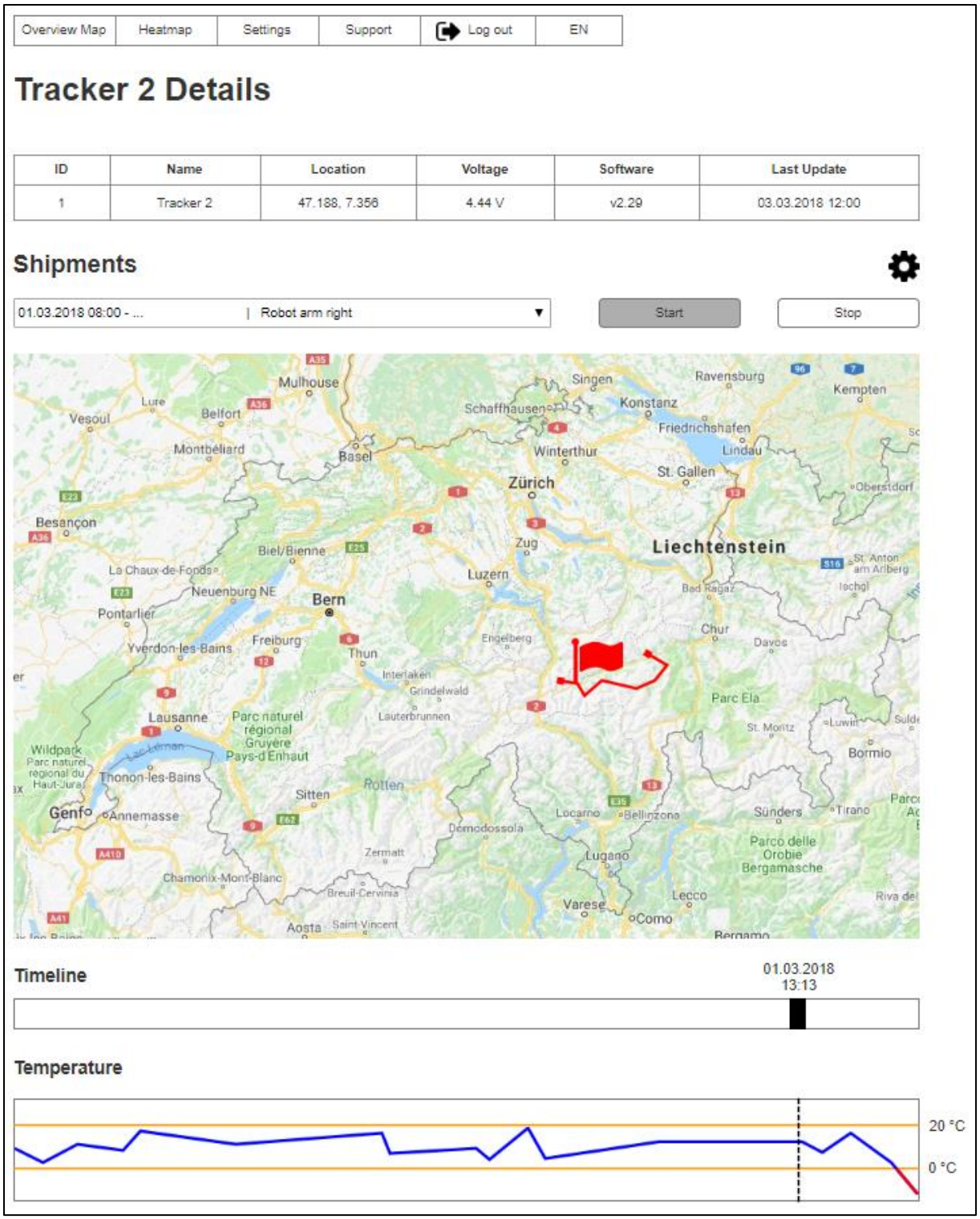

Abbildung 11 Wireframe – Detailansicht

Einstellungen / Settings

Die Einstellungen bestehen aus mehreren Tabs:

- Benutzerverwaltung
- Profil
- Tracker Einstellungen / Einstellungsgruppen
- Benachrichtigungsoptionen
- Tracker Registrierung

Da aber die meisten dieser Tabs auch im HiveWatch System präsent sind und deren Design grösstenteils übernommen werden kann, wurde nur für die Tracker Einstellungen ein Wireframe angefertigt, siehe Abbildung 12.

- Die Tracker und ihre Einstellungsgruppen werden in Tabellenform dargestellt. Dabei bezeichnet jede Einstellungsgruppe eine Tabellensektion, in der sich alle Tracker dieser Gruppe befinden. Über einen Button in Form eines Zahnrads neben jeder Einstellungsgruppe, können diese konfiguriert, sprich die Grenzwerte für die einzelnen Sensoren festgelegt werden.
- Mithilfe der Buttons unterhalb der Tabelle kann der Benutzer neue Einstellungsgruppen erstellen und seine Änderungen speichern, bzw. verwerfen.
- Die Zuweisung der Tracker zu den verschiedenen Gruppen geschieht per Drag and Drop eines Tabelleneintrags in die entsprechende Sektion.

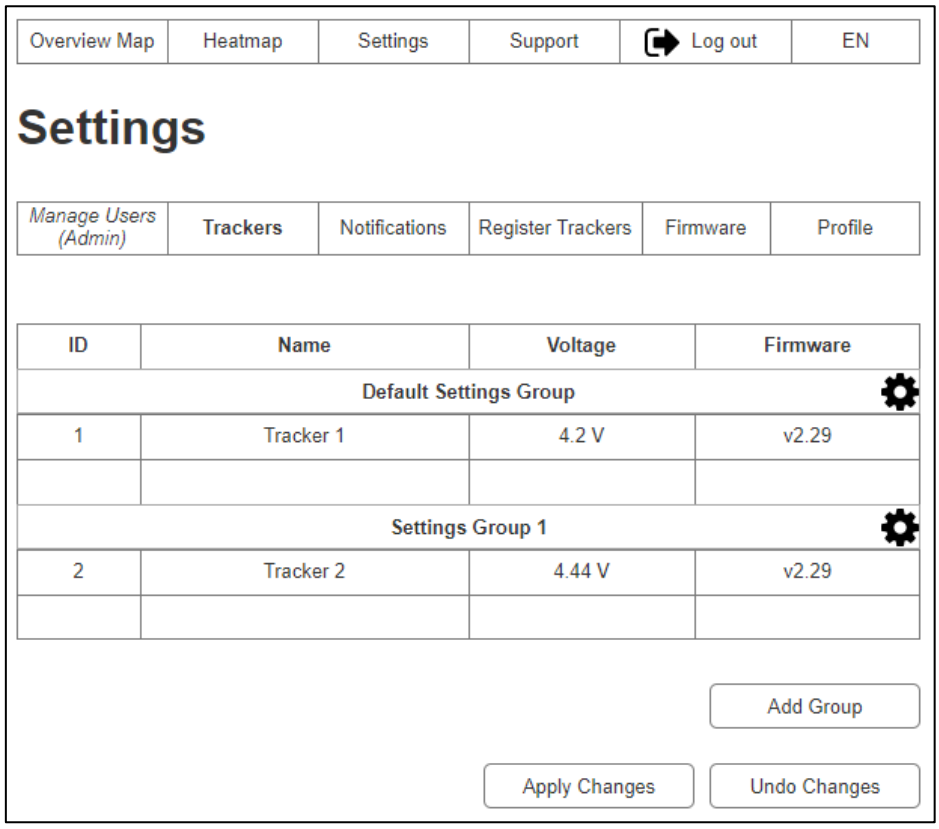

Abbildung 12 Wireframe – Einstellungen

6.5.4 React Komponenten

Eine Übersicht über alle React Komponenten und wie diese ineinander Verschachtelt sind, ist in Abbildung 13 ersichtlich.

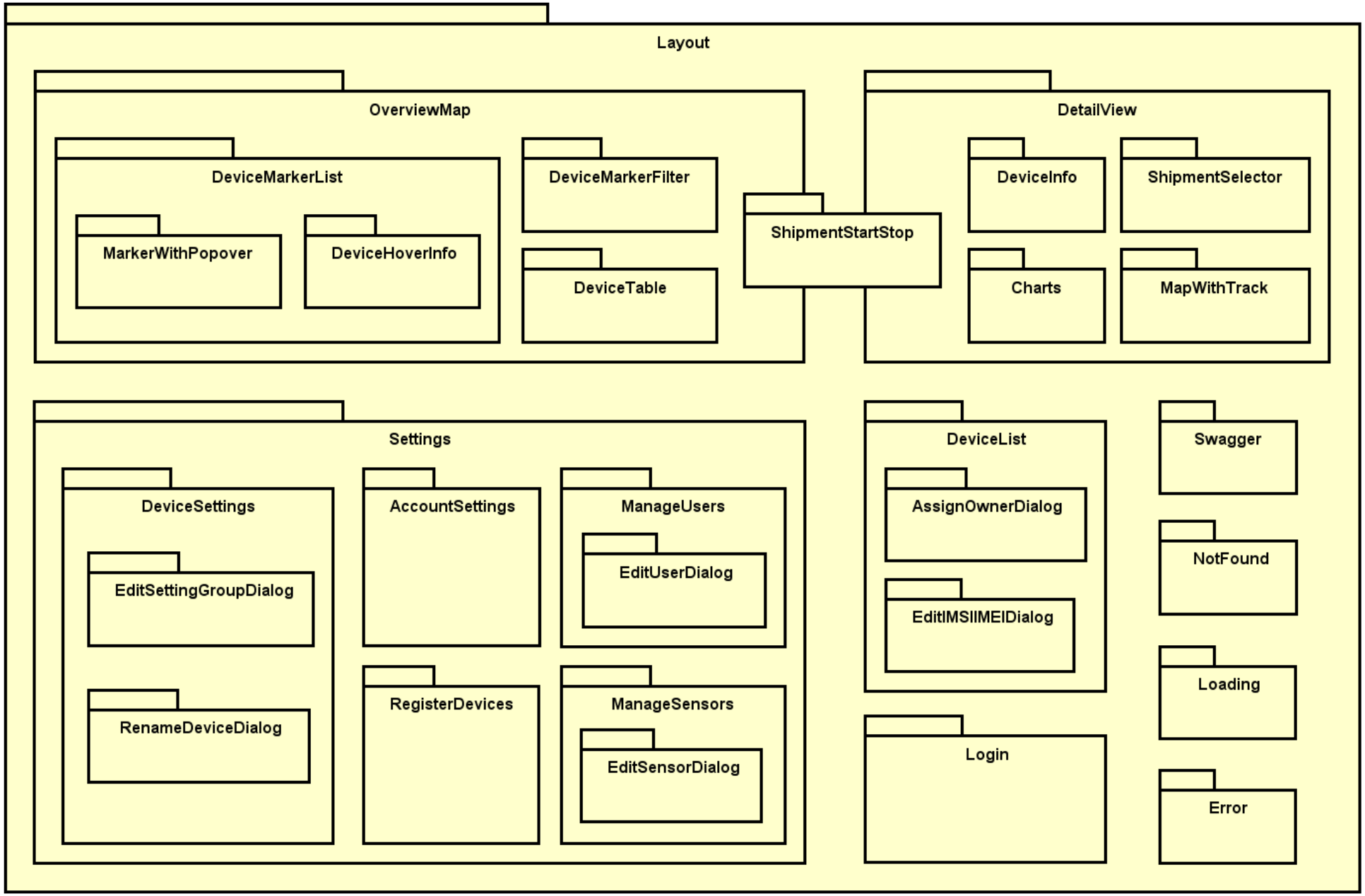

Abbildung 13 Webapp-Frontend React Komponenten

7 Deployment

In Abbildung 14 wird das Deployment Diagramm dargestellt. Jede Software Komponente läuft als eigener Docker Container. Aktuell ist die Verbindung zu MessageBird noch nicht implementiert. Es war aber angedacht, dass der Receiver die eingehenden Daten überprüft und einen Alarm auslöst, falls Werte ausserhalb der definierten Grenzen liegen.
Das Frontend ist nur über den Port 443, sprich eine sichere HTTPS-Verbindung erreichbar. Dies wird ermöglicht, indem Nginx als Reverse-Proxy vor die WebApp geschaltet wurde. Grundsätzlich würde auch Play HTTPS unterstützen. Es gibt aber kein Plug-In von LetsEncrypt für Play, womit die Zertifikate erstellt werden. Da LetsEncrypt Zertifikate eine kurze Laufzeit haben, werden diese mittels zwei Hilfscontainern automatisch verlängert.
Der Receiver ist über den Port 4000 mit einer normalen HTTP-Verbindung erreichbar.
Weil der Datenbank Container keine Ports gegen aussen öffnet, muss man im selben Docker-Netzwerk wie der Container sein, um auf die Datenbank zuzugreifen. Deshalb ist dafür keine zusätzliche Authentifizierung nötig mehr.

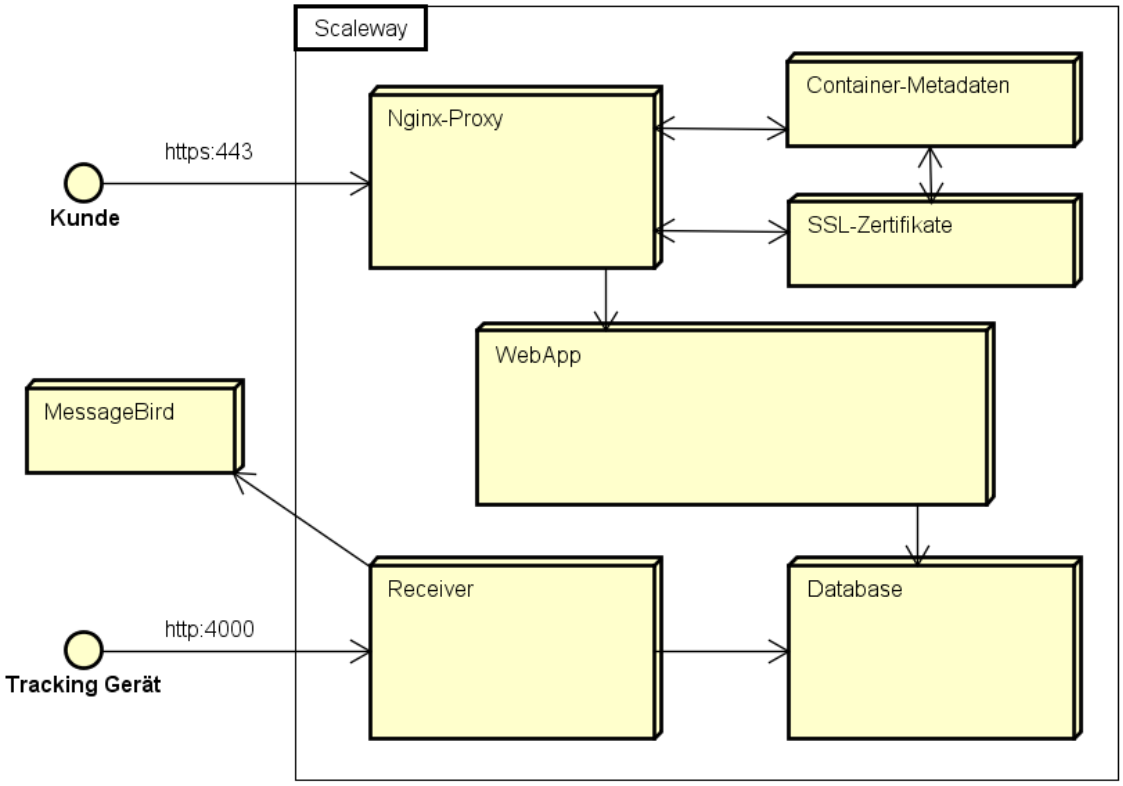

Abbildung 14 Deployment Diagramm

8 Continuous Integration

Der CI Server wurde mit Jenkins auf einer HSR VM Instanz aufgesetzt und ist unter folgender Adresse erreichbar:

- http://sinv-56050.edu.hsr.ch

8.1 Voraussetzungen

Für den Build Prozess wird folgende Software mit zugehörigen Versionen vorausgesetzt:

- git 2.17.0
- npm 5.6.0
- docker 1.13.1
- sbt 1.1.1

8.2 Ablauf

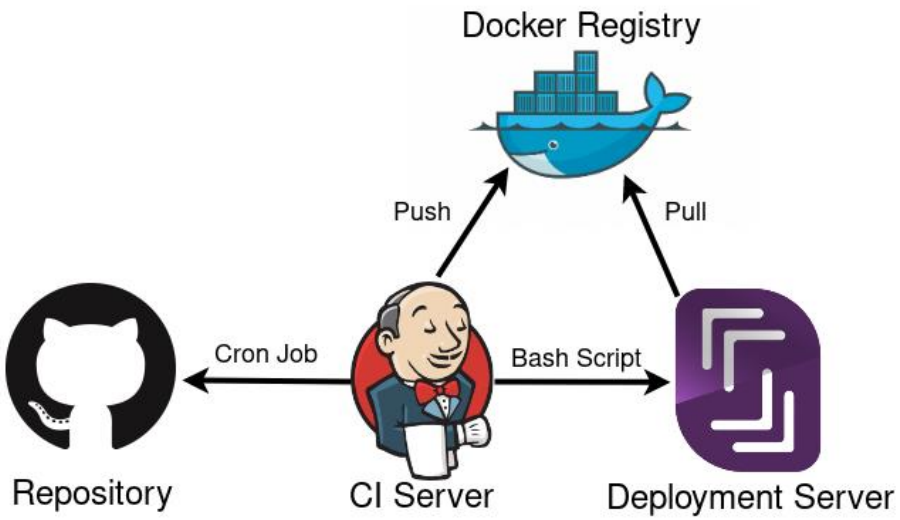

Abbildung 15 Ablauf Contiuous Integration

Der Build-Trigger ist ein Cron-Job, welcher alle 5 Minuten überprüft, ob es eine Änderung auf dem Master Branch des Github Repository gibt. Wenn es eine Änderung gibt, dann startet Jenkins den Build Prozess, welcher wie folgt aussieht.

Testing & Coverage Reporting

```
$ sbt clean coverage test coverageReport coverageAggregate
```

Zuerst werden alle Build Files entfernt. Danach werden alle Projekte getestet und jeweils ein Coverage Bericht erstellt. Der Befehl "coverageAggregate" fasst die Coverage Berichte der verschiedenen Projekte zu einem Bericht zusammen.

Building Webapp

```
$ sbt “project webapp” docker:publish
```

Der Projekt Webapp wird kompiliert und zu Jar Files verpackt. Nachher wird ein Docker Image erstellt, welches anschliessend auf eine vordefinierten Docker Registry gepusht wird.

Building Receiver

```
$ sbt “project receiver” docker:publish
```

Wie beim Webapp-Projekt werden zuerst Jar Files generiert und darauffolgend ein Docker Image erstellt, welches anschliessend auf die vordefinierte Docker Registry gepusht wird.

Deployment auf Scaleway

```
$ ssh -o StrictHostKeyChecking=no -i /home/user/id_rsa \
  root@SERVER_IP 'bash -s' < /var/jenkins_home/deploy_scaleway.sh
```

Standardmässig wird beim Verbindungsaufbau mit einem SSH Client die Identität des Hosts geprüft. Um die Identitätsprüfung zu unterdrücken, wird das Argument "StrictHostKeyChecking=no" übergeben [33]. Mit der Option "-i" kann man das für den Verbindungsaufbau benötigte Private Key File angeben. Die Option "-s" ermöglicht, dass die Befehle vom Standard Input gelesen werden [34]. Das deploy_scaleway.sh Shell Script fährt als erstes die laufende Docker Compose Instanz herunter. Danach werden die neusten Docker Images heruntergeladen und die alten Images abgeräumt. Schliesslich wird die Docker Compose Instanz wieder gestartet.

Aufräumen der alten Images

```
$ if [ "$(docker images|grep "<none>"|awk '{print $3}')" != "" ]; \
   then docker rmi $(docker images|grep "<none>"|awk '{print $3}'); \
  fi
```

Mit jedem neuen Build Prozess wird ein neues Docker Image erstellt. Die alten Docker Images werden aber nur untagged und nicht automatisch abgeräumt. Da der Speicher auf HSR VM knapp ist, müssen nach jedem Build Prozess die untagged Images entfernt werden. Da es vorkommen kann, dass es keine Images abzuräumen gibt, wurde der Befehl in eine If-Bedienung eingepackt. Ansonsten würde der Build Prozess fehlschlagen, wenn es keine untagged Images gäbe, weil der Befehl "docker rmi" mindestens ein Argument erwartet.

8.3 Docker in Docker

Da während des Build Prozesses Docker Images erstellt werden, braucht es auf der Jenkins Umgebung auch eine Docker Engine. Unser Jenkins läuft als Docker Container, das heisst, es wird eine Docker Engine in einem Docker Container benötigt. Es gibt zwei Möglichkeiten, wie man das bewerkstelligen kann.
Die erste Variante heisst Docker-in-Docker (DinD). Dabei wird eine vollständige Docker Engine innerhalb eines Docker Containers installiert. DinD ist vor allem nützlich, wenn die Images vor dem Host System verborgen und isoliert werden sollen.
Die zweite Variante heisst Docker-outside-of-Docker (DooD). Dort wird das Socket und das Binary der Docker Engine des Host Systems auf den Jenkins Container gemappt. Der Vorteil gegenüber DinD liegt darin, dass es möglich ist die Images und den Cache des Host Systems wiederzuverwenden [35].
Da der Speicher auf der HSR VM bereits knapp war, wurde entschieden DooD einzusetzen.
Um Docker's Socket und Binary zu mappen, muss der Jenkins Container wie folgt gestartet werden:

```
$ docker run -d -v /root/jdata:/var/jenkins_home -p 80:8080 \
      -v /var/run/docker.sock:/var/run/docker.sock          \
      -v $(which docker):/usr/bin/docker                    \
      jenkins/jenkins:lts
```

Um Docker innerhalb Jenkins auszuführen, muss der User Jenkins der Gruppe Docker hinzugefügt werden. Da die Docker Images von SBT erstellt werden, muss die Klasse Others auf das Docker Socket zugreifen können. Deshalb sind folgende Änderungen notwendig, welche Root Rechte erfordern:

```
$ docker exec -it -u root jenkins_container bash
# groupadd docker
# usermod -aG docker jenkins
# chmod 777 /var/run/docker.sock
```

8.4 Docker Garbage

Immer wenn ein neues Image mit derselben Bezeichnung und Version, wie ein bereits existierendes Image, erstellt, bzw. gepullt wird, dann wird das alte Image untagged. Die alten Images werden nur untagged und nicht sofort gelöscht, da sie noch in Gebrauch sein könnten. Untaggen bedeutet, dass die Version des Images auf <none> gesetzt wird.
Es gibt drei Instanzen, wo untagged Images generiert werden. Die erste Instanz ist Jenkins, weil da die neuen Docker Images erstellt werden. Die zweite Instanz ist die Docker Registry, dort werden die Images aufbewahrt. Die dritte Instanz ist Scaleway, wo die Images gepullt werden.
Bei Jenkins und Scaleway kann man die untagged Images mit folgendem Befehl abräumen:

```
$ docker rmi $(docker images|grep "<none>"|awk '{print $3}')
```

Die Docker Registry benutzt einen Garbage Collector. Daher kann man die Images nicht sofort löschen, sondern nur für die Löschung markieren. Da der Zugriff auf die Docker Registry eher etwas umständlich ist, wurde entschieden den Registry Container periodisch von Hand zu killen und neu zu erstellen. So werden alle Images, welche sich im Container befanden, entfernt.

III Implementation

9 Datenbank

9.1 Die Auswirkungen von Key Namen

Wenn man davor nur mit relationalen Datenbank gearbeitet hat, dann wird man sich bemühen sinnvolle und ausdrucksvolle Namen für Spalten zu vergeben. Aber dabei macht man sich keine Gedanken darüber, wie viel Speicher die Spaltennamen am Schluss beanspruchen werden. In einem relationalen Datenbanksystem werden die Spaltennamen nur einmal abgespeichert, meistens in der sogenannten zentralen Metadaten Tabelle. In einem schemafreien Datenbanksystem wie MongoDB hat die Wahl der Key Namen jedoch erhebliche Auswirkungen auf die endgültige Grösse einer Collection. Das liegt daran, dass die Key Namen in jedem Dokument gespeichert werden.
Bei der empirischen Untersuchung von Christopher Maier wurden 1.6 Milliarden Dokumente einmal mit langen Keys wie in Listing 10 und einmal mit kurzen Keys wie in Listing 11 abgespeichert. Die Gesamtgrösse mit langen Keys lag etwa bei 243 GB und das Gesamtgrösse mit kurzen Keys bei etwa 183 GB. Durch die Verwendung von kurzen Keys konnte also eine Einsparung von 60 GB Speicher erzielt werden [36].

```
{
  "sequence": "AHAHSPGPGSAVKLPAPHSVGKSALR",
  "location": {
    "chromosome": "19",
    "strand": "-",
    "begin": "51067007",
    "end": "51067085"
   }
}
```

Listing 10 Beispiel Dokument mit langen Keys

```
{
  "s":"AHAHSPGPGSAVKLPAPHSVGKSALR",
  "l": {
    "c":"19",
    "s":"-",
    "b":"51067007",
    "e":"51067085"
  }
}
```

Listing 11 Beispiel Dokument mit kurzen Keys

9.2 Datenbankmodell

In Abbildung 16 werden die tatsächlichen Strukturen der Dokumente dargestellt und die Relationen zwischen den Dokumenten aufgezeigt. Der Name einer Collection ist im jeweiligen Kasten oben links angegeben. Falls die Key Namen nicht selbsterklärend sind, kann das Datenmodell in Abbildung 4 zur Unterstützung genommen werden. Bei erforderlichen Feldern steht jeweils ein Sternchen vor dem Feldnamen. Die anderen Felder sind optional.
Zahlenwerte, welche nicht in einem speziellen Wrapper sind, sind automatisch Double Werte. Dies ist vor allem wichtig, wenn man von Hand per direktem Datenbankzugriff Daten einfügen oder ändern will. Denn MongoDB interpretiert alle Zahlenwerte ohne Wrapper immer als Double. Auch wenn diese keine Nachkommastellen haben. Zum Beispiel muss ein Integer also immer im NumberInt-Wrapper eingefügt werden.
Da die Länge der Keys bei MongoDB einen Einfluss auf den Speicher hat, was genauer in Kapitel 9.1 beschrieben ist, wurde darauf geachtet die Keys möglichst kurz und prägnant zu wählen. Bei den Collections readings und shipments wird eine hohe Anzahl an Datensätzen erwartet. Dementsprechend sind diese Keys bedingungslos kurz gehalten. Bei den anderen Collections gibt es wenige etwas längere Keys. Dies ist es nicht tragisch, da bei diesen Collections nur eine geringe Anzahl von Datensätzen erwartet wird.
Im Gegensatz zum Datenmodell gibt es zwei weitere Collections namens auth.cues und auth.tokens, welche für die Authentifizierung benötigt werden. Zudem fehlt das Feld "password" in der users Collection. Das liegt daran, dass die Passwörter gehasht in der Collection auth.cues gespeichert werden.

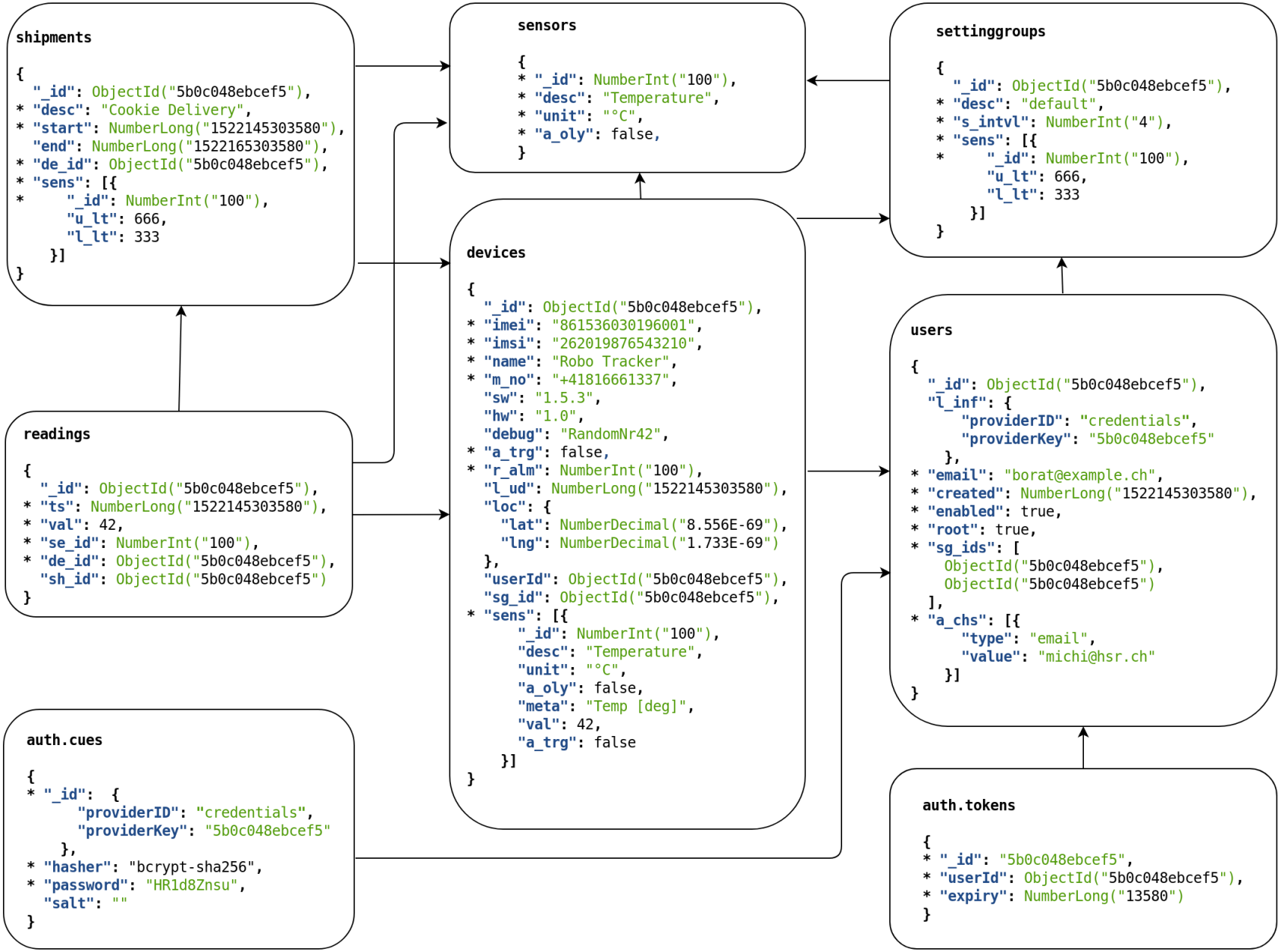

Abbildung 16 Datenbankmodell

10 Receiver

Der Receiver kann die Tracker-Daten sowohl auf der Route "/data", als auch "/" empfangen. Ursprünglich war nur die Route "/" angedacht. Der Test Tracker sendet die Daten aber auf die Route "/data". Daher werden aktuell beide Routen unterstützt.

10.1 Ablauf

Die eingehenden Daten werden bereits so aufbereitet, dass das Frontend möglichst wenige Requests durchführen muss. In Abbildung 17 wird der ganze Ablauf in Form eines Flussdiagramms dargestellt.

Validierung

Die Input-Werte werden beim Parsen auch gleich validiert, um zu verhindern, dass sich fehlerhafte Werte ins System einschleichen und später Fehler verursachen. Wie im Abschnitt 6.2.1 bereits aufgezeigt, sind Zahlenwerte in "Quotes" eingepackt, sprich sie werden als Strings gesendet. Daher wird beim Parsen darauf geachtet, dass die Zahlenwerte gültige Scala Werte sind. Ausserdem wird bei Timestamps sichergestellt, dass sie zwischen dem Jahr 2000 und 3000 liegen.

Frühzeitige Antwort

Die Tracker erwarten als Antwort auf ihre Datensendungen immer den Statuscode "200 OK". Erhalten sie diese Antwort nicht innerhalb von drei Minuten, senden sie die Daten erneut. Beim HiveWatch-System gab es selten Vorfälle, bei denen der Server die Antwort zu spät zurücksendete, weil die Speicherung der Daten zu lange dauerte. Um dies zu vermeiden, sendet dieser Receiver eine frühzeitige Antwort nach der Validierung der Daten, da nach diesem Punkt die Wahrscheinlichkeit dass ein Fehler auftritt extrem niedrig ist.
Nach der Validierung wird noch überprüft, ob der Tracker bereits in der Datenbank registriert ist, da nur Daten von registrierten Trackern angenommen werden. Falls entweder die Validierung fehlschlägt, oder der Tracker nicht registriert ist, wird der Statuscode "400 BadRequest" zurückgesendet. Sind beide Bedienungen erfüllt, wird "200 OK" zurückgesendet.

Speicherung

Nach der Validierung werden alle Shipments geladen, welche im Zeitraum der Input-Daten liegen. Dann werden zwei parallel laufende Prozesse gestartet.
Der erste Prozess ist die Erstellung der Readings und Alarmierung zuständig. Er erstellt pro Messpunkt ein Reading und überprüft, ob der Wert innerhalb der vom Shipment vorgegebenen Grenzwerte ist. Falls ein oder mehrere Werte ausserhalb seiner Grenzwerte liegen, wird ein Alarm ausgelöst. Dieser besteht momentan aus einer einfachen Konsolenausgabe, kann aber später mit einem Messaging-Dienst erweitert werden.
Der zweite Prozess ist für die Aktualisierung des Device-Objekts zuständig. Er überprüft zuerst, ob im vorherigen Schritt mindestens ein Shipment geladen wurde. Ist dies nicht der Fall, werden nur Grundinformationen wie Software, Hardware, und Debug aktualisiert. Existiert ein laufendes Shipment, wird eine komplette Aktualisierung durchgeführt. Dies hat den Grund, dass das Device-Objekt nicht mit Daten aktualisiert werden soll, die nicht innerhalb eines Shipments liegen, da diese Daten für den Benutzer nicht interessant sind. Bei der kompletten Aktualisierung werden auch die Sensoren mit ihren neusten Werten aktualisiert. Existiert ein Sensor noch nicht im Device-Objekt, wird versucht, diesen Sensor anhand seiner ID in der globalen Sensorliste zu finden. Gelingt dies, wird die dort vorhandene Beschreibung und Einheit verwendet. Wenn nicht, werden diese Felder anhand der Metadaten des Sensors generiert und dieser neue Sensor auch der globalen Liste hinzugefügt.

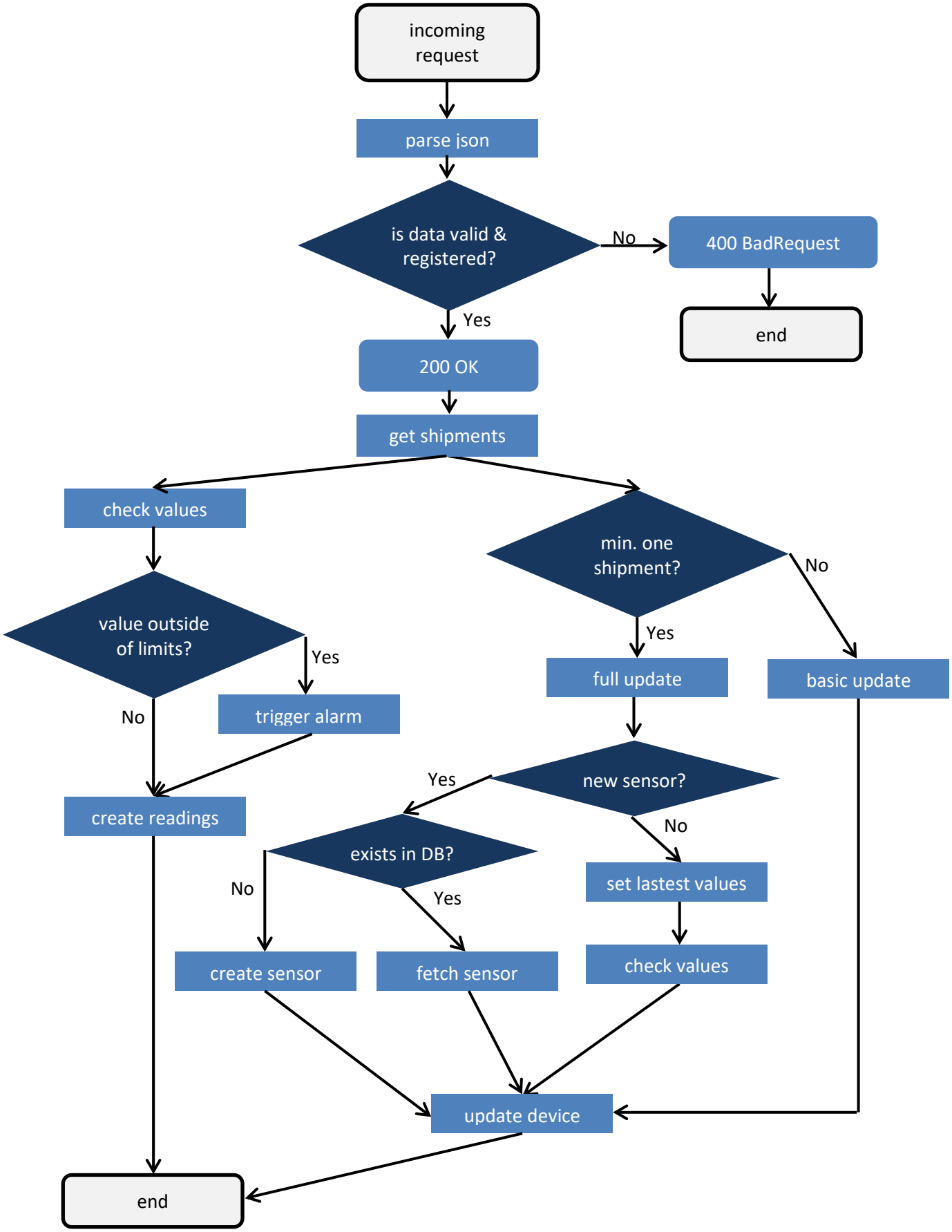

Abbildung 17 Flussdiagramm Receiver

11 Webapp-Backend

Das ganze Authentifizierungszusammenspiel zwischen Frontend und Backend wurde von der silhouette-play-react-seed[1] Vorlage übernommen. Dort wird jede Response im Wrapper APIResponse verpackt. Da dieser Wrapper für ein kleines Projekt wie HiveTrack überflüssig war, wurde er entfernt.

11.1 Postcondition

Einmal tauchte im Frontend ein Fehler auf, weil vom Backend eine Antwort in Form von JSON erwartet wurde, welches aber Text lieferte, da die Antwort aus einem einfachen String bestand. Deshalb einigte man sich darauf, dass das Backend alle Antworten immer als JSON liefern muss. Bei Sonderfällen, wie einfachen Strings, muss man darauf achten, dass diese erst nach einer expliziten Umwandlung, siehe Listing 12, als JSON gesendet werden.

```
Forbidden                              // content-type: text/html
Forbidden("forbidden")                 // content-type: text/html
Forbidden(Json.toJson("forbidden")) // content-type: application/json
```

Listing 12 Forbidden as JSON

11.2 CSRF – Token

Bei Play 2.6 ist der CSRF-Check standardmässig aktiviert. Bei Requests mit einer anderen Methode als GET, HEAD oder OPTIONS, die Cookies enthalten, wird jeweils das CSRF-Token überprüft [37]. Dieses muss als Header, wie in Abbildung 18 gezeigt, mitgesendet werden.

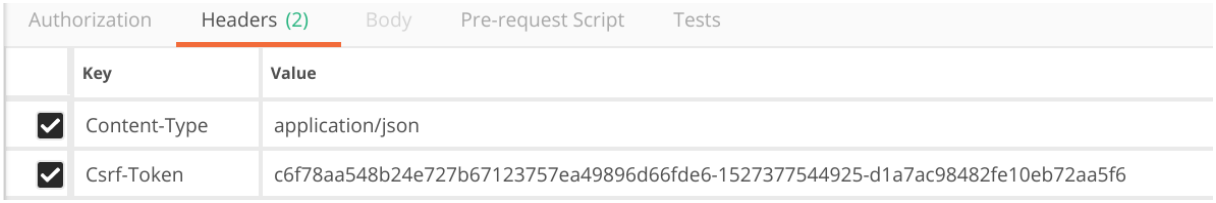

Abbildung 18 Postman Csrf-Token

11.3 AuthToken

Das AuthToken-Konzept wurde ebenfalls von der silhouette-play-react-seed Vorlage übernommen. Diese Tokens können dafür verwendet werden, neue Benutzer ihre E-Mail Adresse bestätigen zu lassen, um ihren Account zu aktivieren. Man würde also den Benutzern eine E-Mail mit einem Aktivierungs-Link, der das Token enthält, senden.
Da momentan aber alle Accounts noch von Administratoren erstellt werden und sich Benutzer gar nicht selbständig registrieren können, werden diese Tokens noch nicht verwendet. Da sich dies in Zukunft aber ändern kann, wurden die ganzen AuthToken-Klassen noch beibehalten.

11.4 AuthCue

Die Klasse AuthCue wird benötigt, um nicht gemeinsam mit dem User-Objekt auch das geheime Passwort an das Frontend zurückzusenden. Stattdessen wird das User-Objekt mit einem AuthCue assoziiert, welcher nicht mitgesendet wird und das Passwort enthält.

[1] https://github.com/setusoft/silhouette-play-react-seed

11.5 Routes-Dateien

Da entschieden wurde die API mit dem Swagger Plug-In von iHeartRadio zu dokumentieren, wurden alle zusätzlichen Angaben für die OpenAPI Spezifikation in den Routes-Dateien festgehalten. Weil man mit der Zeit bei einer einzelnen Routes-Datei die Übersicht verlieren würde, wurde diese in mehrere Routes-Dateien unterteilt. Dank dieser Unterteilung wird auch bei der OpenAPI Spezifikation eine entsprechende Gruppierung gemacht.
Nun muss aber darauf geachtet werden, dass bei jeder Routes-Datei die Asset-Route mit gleichem Inhalt angegeben werden muss. Falls diese bei einer der Routes-Dateien fehlt, funktioniert sie auch bei keiner der anderen Routes-Dateien mehr.
Etwas komisch ist, dass die Routes-Dateien jeweils mit ".routes" enden müssen, die Referenzierung dieser Dateien von der Haupt-Datei aus aber mit einem grossen R, also ".Routes" geschrieben werden muss.

12 Data-Access-Layer

12.1 Implizites Schema

Da MongoDB eingesetzt wird, gibt es kein Schema auf der Datenbank-Ebene. Dennoch haben schemalose Strukturen immer noch ein implizites Schema. Jeder Code, der die Daten manipuliert, muss einige Annahmen über die Datenstruktur treffen, beispielsweise die Namen von Feldern. Alle Daten, die nicht zu diesem impliziten Schema passen, werden nicht richtig manipuliert, was zu Fehlern führt [38]. Da es sich bei dieser Datenbank nicht um eine Integrationsdatenbank, auf welche von verschiedenen Abteilungen zugegriffen wird, sondern um eine isolierte Applikationsdatenbank handelt, ist die schemalose Struktur der Datenbank bedenkenlos.
Grundsätzlich sollten alle direkten Datenmanipulationen über MongoShell oder andere Tools vermieden werden, da die Daten so rasch und unbemerkt in einen Zustand gebracht werden können, welcher nicht mehr dem impliziten Schema entspricht. Später würde der Scala Code eine Exception werfen, weil entweder ein benötigtes Feld nicht gefunden werden kann oder der Datentyp von einem Feld nicht übereinstimmt. Wenn eine direkte Datenmanipulation durchgeführt wird, muss also immer darauf geachtet werden, dass die Feldnamen und Datentypen mit dem Datenbankmodell übereinstimmen.
Muss ein neues Feld für ein Objekt eingeführt werden, kann dies auf zwei Arten geschehen. Entweder man fügt dieses Feld in alle bestehenden Objekte ein, sodass alle Objekte wieder demselben Schema entsprechen. Oder man verpackt dieses neue Feld im Scala Code in den Typ Option und macht es so optional.

12.2 JSONCollection vs BSONCollection

Der ReactiveMongo-Treiber bietet mehrere Möglichkeiten auf die Collections der Datenbank zuzugreifen, welche als *BSON* gespeichert werden. Entweder in Form von JSON oder BSON.
Wird mit JSONCollections gearbeitet, dann müssen Parameter für Operationen als JSON übergeben werden und als Rückgabewert kommt auch JSON zurück. Die ganze Umwandlung von BSON zu JSON wird von ReactiveMongo übernommen. Wie in Abbildung 19 aufgezeigt, gibt es für JSONCollections nur einen Umwandlungsschritt beim Auslesen der Daten.
Wird mit BSONCollections gearbeitet, dann müssen Parameter für Operationen als BSON übergeben werden und als Rückgabewert werden Scala Objekte zurückgegeben. Die Umwandlung von BSON zu Scala Objekten wird mittels generierten Serializern/Deserializern ermöglicht. Da das Frontend aber JSON erwartet, müssen die Scala Objekte noch in JSON umgewandelt werden. Wie in Abbildung 20 dargestellt, gibt es für BSONCollections gesamthaft zwei Umwandlungsschritte beim Auslesen der Daten.

Abbildung 19 JSONCollection auslesen

Abbildung 20 BSONCollection auslesen

Da beim Arbeiten mit BSONCollections also ein Umwandlungsschritt mehr benötigt wird, wurde angenommen, dass das Auslesen von Daten als JSONCollection schneller ist. Um die Annahme zu bestätigen, wurde eine empirische Untersuchung durchgeführt. Dafür wurden 50‘000 gespeicherte Dummy-Objekte einmal als JSONCollection ausgelesen und einmal als BSONCollection ausgelesen und zu JSON umgewandelt. Die Zeit wurde in Nanosekunden gemessen. Wie in Listing 13 aufgezeigt, ist BSONCollection überraschenderweise etwa 3-Mal schneller als JSONCollection.

Durchgang	JSONCollection [ns]	BSONCollection [ns]
1.	2250413122	771045175
2.	2221809295	719528277
3.	2217861511	775460038
4.	2174103893	692610856
5.	2390114195	909560321
6.	2196846715	719223724
Mittelwert	**2241858122**	**764571399**

Listing 13 Vergleich JSONCollection vs. BSONCollection

Fazit

Trotz eines zusätzlichen Umwandlungsschrittes ist die BSONCollection-Variante immer noch 3-Mal schneller als die JSONCollection-Variante. Dies kann daran liegen, dass die Umwandlung von BSON zu JSON bei ReactiveMongo suboptimal gelöst wurde. Abgesehen von der besseren Performance, bietet BSON auch noch feingranularere Typen als JSON. Daher wurde entschieden BSONCollections zu verwenden.

12.3 ReactiveMongo Annotations

@Key

Mit dieser Annotation können andere Feldnamen für die Datenbank gesetzt werden, welche von den Feldnamen im Scala Code abweichen. Dadurch ist es möglich sinnvolle und beschreibende Namen für die Kommunikation mit dem Frontend zu wählen und gleichzeitig speichereffiziente Namen innerhalb der Datenbank zu nutzen.

@Ignore

Mit dieser Annotation markierte Felder werden nicht in die Datenbank gespeichert. Dies wurde für bei solchen Feldern eingesetzt, deren Inhalte vom Backend generiert und nur für die Kommunikation mit dem Frontend benötigt werden.

@Flatten

Wie in Listing 14 gezeigt, können mit dieser Annotation die Felder von verschachtelten Dokumenten in das äussere Dokument verschoben werden. Dies wird nur einmal beim Modell AuthCue verwendet.

```
case class Range (start: Int, end: Int)

case class LabelledRange (
  name: String,
  @Flatten range: Range
)

// Flattened
BSONDocument("name" -> "foo", "start" -> 0, "end" -> 1)
```

Listing 14 Beispiel @Flatten

13 Webapp-Frontend

Das Frontend besteht aus mittlerweile 32 React Komponenten und einigen weiteren Hilfsfunktionen, welche in der in Abbildung 21 gezeigten Ordnerstruktur organisiert sind.

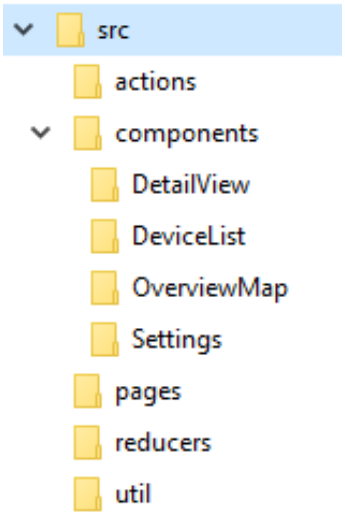

Abbildung 21
Webapp-Frontend Ordnerstruktur

Pages

Pages (dt. Seiten) sind grössere React Komponenten, die jeweils eine ganze Seite, wie beispielsweise die Übersichtskarte oder Detailansicht, darstellen. Sie sind dafür verantwortlich die benötigten Daten vom Backend in den Redux Store zu laden und die entsprechenden Teile des Stores an ihre Subkomponenten weiterzugeben. Weil die Subkomponenten nicht direkt auf den Store zugreifen, sondern ihre Daten von einer Überkomponente erhalten, müssen sie nichts von der Existenz des Stores oder Redux wissen. Sie sind "dumme", einfache React Komponenten, die basierend auf ihren Parametern eine GUI Komponente rendern. Diese Entkopplung macht die Subkomponenten extrem wiederverwendbar.

Components

Pro Seite existiert im Components-Ordner ein Unterordner, der die jeweiligen Subkomponenten der Seite enthält. Falls eine Subkomponente auf mehreren Seiten verwendet wird, wird diese direkt im Components-Ordner abgelegt.

Actions

In diesem Ordner befinden sich alle Redux Actions. Sie sind pro Datenmodell-Objekt in einer separaten Datei zusammengefasst. Die einzige Ausnahme sind die Shipments (Paketsendungen), deren Actions mit denen der Devices (Tracker) zusammengefasst wurden, da die Paketsendungen jeweils direkt mit den Trackern geladen werden. Der Grund dafür hat mit der Effizienz der Applikation zu tun.
Lädt man mehrere Tracker und all deren Paketsendungen über zwei separate Backend Abfragen, erhält man vom Backend auch zwei separate Listen. Das Problem dabei ist, dass die Paketsendungen irgendwie den Trackern zugeordnet werden müssen. Dies geschieht über die Tracker-ID, welche in den Paketsendungen abgelegt ist. Das bedeutet, dass man jedes Mal, wenn man alle Paketsendungen eines bestimmten Trackers benötigt, alle Paketsendungen nach dieser ID durchsuchen muss, was sehr ineffizient ist.
Deshalb entschieden wir, das "withShipments"-Flag bei den Devices-Endpoints der REST API einzuführen. Über dieses Flag wird entschieden, ob nur der/die Tracker, oder auch ihre Paketsendungen geladen werden sollen. So können diese bereits im Backend direkt in die Tracker-Objekte eingefügt werden. Dadurch ist kein ständiges Durchsuchen von Paketsendungen mehr nötig, da in jedem Tracker seine Paketsendungen bereits enthalten sind.

Reducer

Wie bei den Actions gibt es auch hier pro Datenmodell-Objekt einen Reducer. Wieder mit Ausnahme der Shipments. Bemerkenswert ist auch die Struktur der Readings (Messwerte), die sich von der Struktur der Readings in der Datenbank unterscheidet. Dies hat mit einem ähnlichen Problem wie bei den Trackern und Paketsendungen zu tun. Anstatt um Tracker und zugehörige Paketsendungen, handelt es sich aber um Sensoren und zugehörige Messwerte.
Hier wurde ein etwas anderer Lösungsansatz angewendet. Und zwar geschieht im Backend das in Listing 15 gezeigte Ummapping der Messwerte. Dieses ermöglicht es die Messwerte nach Sensoren getrennt zu betrachten, ohne alle Messwerte durchsuchen zu müssen. Das ist hier noch wichtiger als bei den Paketsendungen, da es sich um eine um ein Vielfaches grössere Datenmenge handelt.

```
[
  {
    "_id": "BSONObjectId",
    "timestamp": 1522145303536,
    "value": 42,
    "sensorId": 100
  },
  {
    "_id": "BSONObjectId",
    "timestamp": 1522145313536,
    "value": 1337,
    "sensorId": 200
  },
  ...
]
```

```
{
  100: [  //sensorId
    {
      "timestamp": 1522145303536,
      "value": 42
    },
    ...
  ],
  200: [  //sensorId
    {
      "timestamp": 1522145303536,
      "value": 1337
    },
    ...
  ],
  ...
}
```

Listing 15 Ummapping der Messwerte

Util

In diesem Ordner befinden sich diverse Hilfsfunktionen, wie zum Beispiel die Funktion `getLocationName(location)`, die mithilfe der Google Maps Geocoding API [39] geografische Koordinaten in lesbare Adressen umwandelt.

13.1 Funktionalität

Ein Grossteil der im Frontend umgesetzten Funktionalität ist bereits zusammen mit den Wireframes in Kapitel 6.5.3 dokumentiert. Hier werden dementsprechend nur die Neuerungen oder Änderungen aufgeführt, die nicht mit den Wireframes übereinstimmen, welche nach ihrer Erstellung nicht mehr aktualisiert wurden.

Übersichtskarte / Overview Map

Die Übersichtskarte, siehe Abbildung 22, wurde bis auf ein paar kleinere Änderungen komplett nach dem Wireframe gebaut.

- Administratoren sehen hier nicht nur ihre eigenen, sondern alle im System registrierten Tracker.
- Tracker, bei denen während der letzten Paketsendung alles in Ordnung war, werden nicht blau, sondern grün dargestellt.
- Anstatt dass der Benutzer durch einen Klick auf einen Listeneintrag unterhalb der Karte die Detailansicht des entsprechenden Trackers öffnen kann, wurde dafür ein separater Button "Show Details" in die Liste eingefügt. Dies hat den Grund, dass das Öffnen der Detailansicht zu versteckt war und für neue Benutzer nicht auf den ersten Blick klar war, dass überhaupt eine Detailansicht existiert.
- Ein Klick auf einen Listeneintrag löst neu eine Hervorhebung des entsprechenden Positions-Markers auf der Karte aus. Diese Hervorhebung geschieht durch eine Vergrösserung des Positions-Markers.
- Zusätzlich zum Marker-Filter rechts oberhalb der Karte wurden Checkboxen in die Tracker-Liste eingefügt, durch die Tracker nicht nur gruppenweise, sondern auch einzeln herausgefiltert werden können.

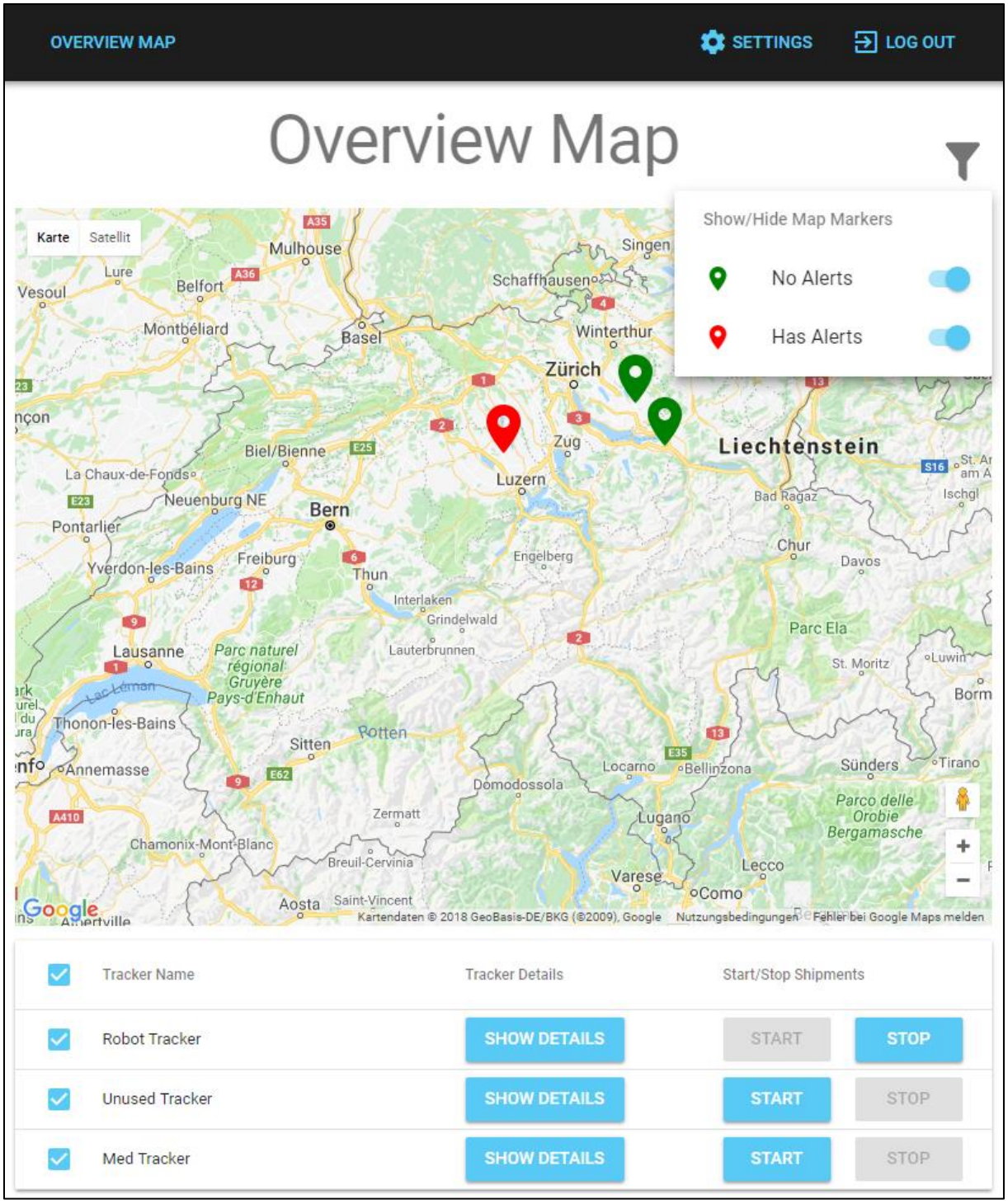

Abbildung 22 Übersichtskarte

Detailansicht / Tracker Details

Bei der Detailansicht, siehe Abbildung 25, gab es hauptsächlich Anpassungen und Erweiterungen in Zusammenhang mit den Charts und dem Starten von Paketsendungen.

- Wird der Start-Button geklickt, öffnet sich das in Abbildung 23 gezeigte Popup, über das der Paketsendung nun auch eine Beschreibung hinzugefügt werden kann.

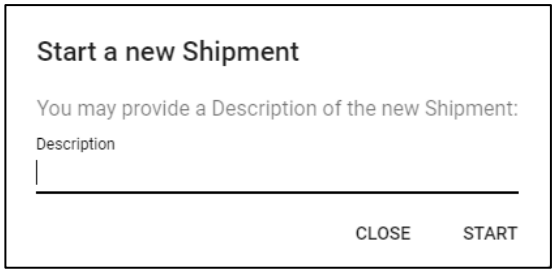

Abbildung 23 Start Shipment Popup

- Leider konnten die Ober- und Untergrenzen für die Sensorwerte nicht als horizontale Linien im Chart angezeigt werden. Die rote Markierung der fehlerhaften Werte konnte aber durch definieren von Chart-Zonen [40] umgesetzt werden.
- Anstatt des separaten Timeline-Sliders, über den der Positions-Marker auf der Karte und die vertikalen Markierungen in den Charts verschoben werden können, passiert dies durch bewegen des Cursors über die Charts. Dabei ist es egal, über welches Chart der Cursor bewegt wird, da alle Charts synchronisiert sind, sprich sich alle Markierungen gleichzeitig verschieben.
- Pro Chart wird der genaue Wert, der momentan markiert ist, rechts oben gemeinsam mit dessen Einheit angezeigt.
- Die Charts besitzen eine Zoom-Funktion, mithilfe derer auf einen kürzeren Zeitbereich (minimal 1 Minute) hereingezoomt werden kann. Dies ermöglicht eine wesentlich genauere Analyse der Messwerte. Die Zoom-Funktion ist ebenfalls für alle Charts synchronisiert. Wird in ein Chart hereingezoomt, geschieht dies auch bei den anderen. Wie die Anwendung des Zooms funktioniert, ist in Abbildung 24 dargestellt.

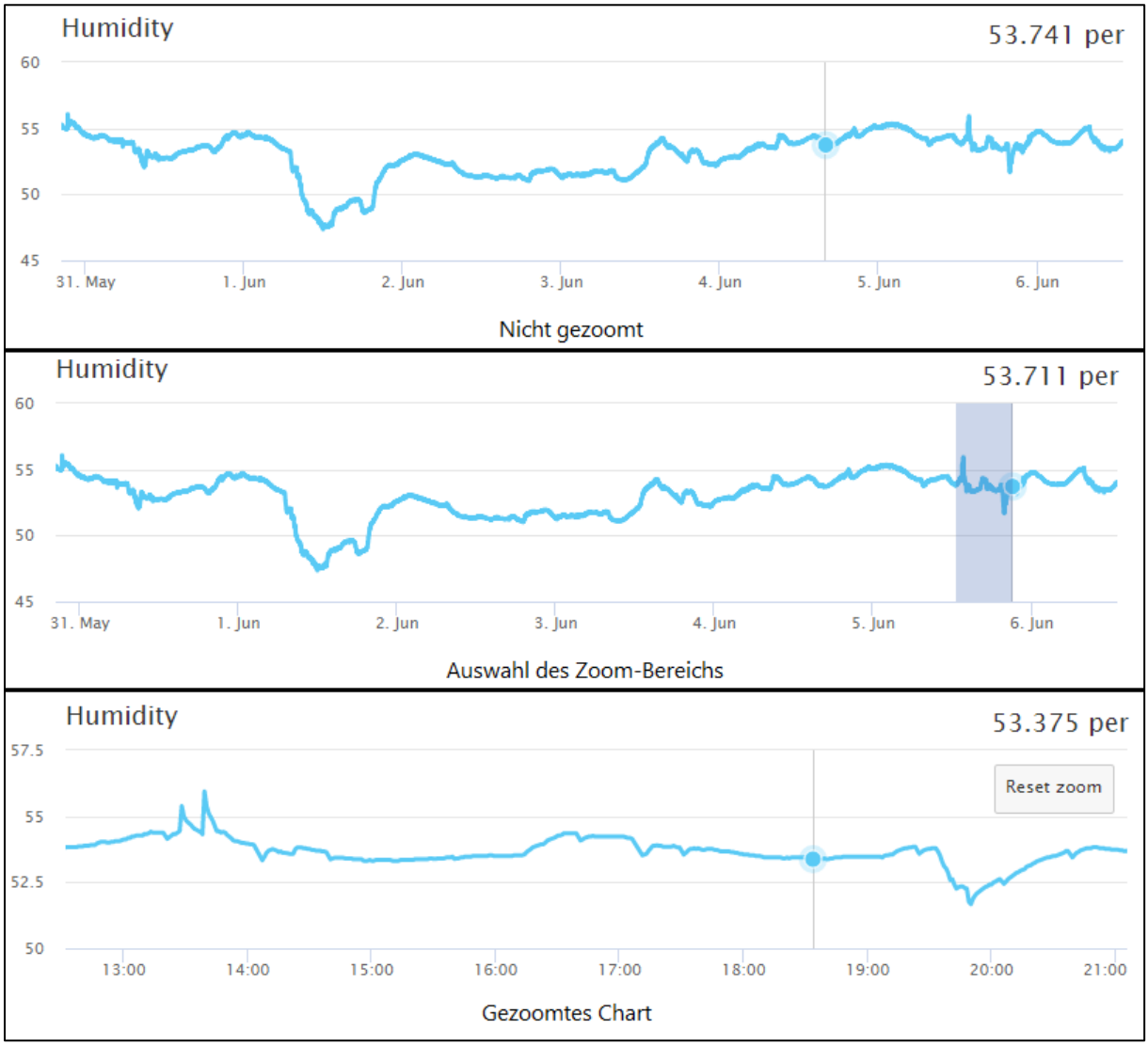

Abbildung 24 Chart Zoom

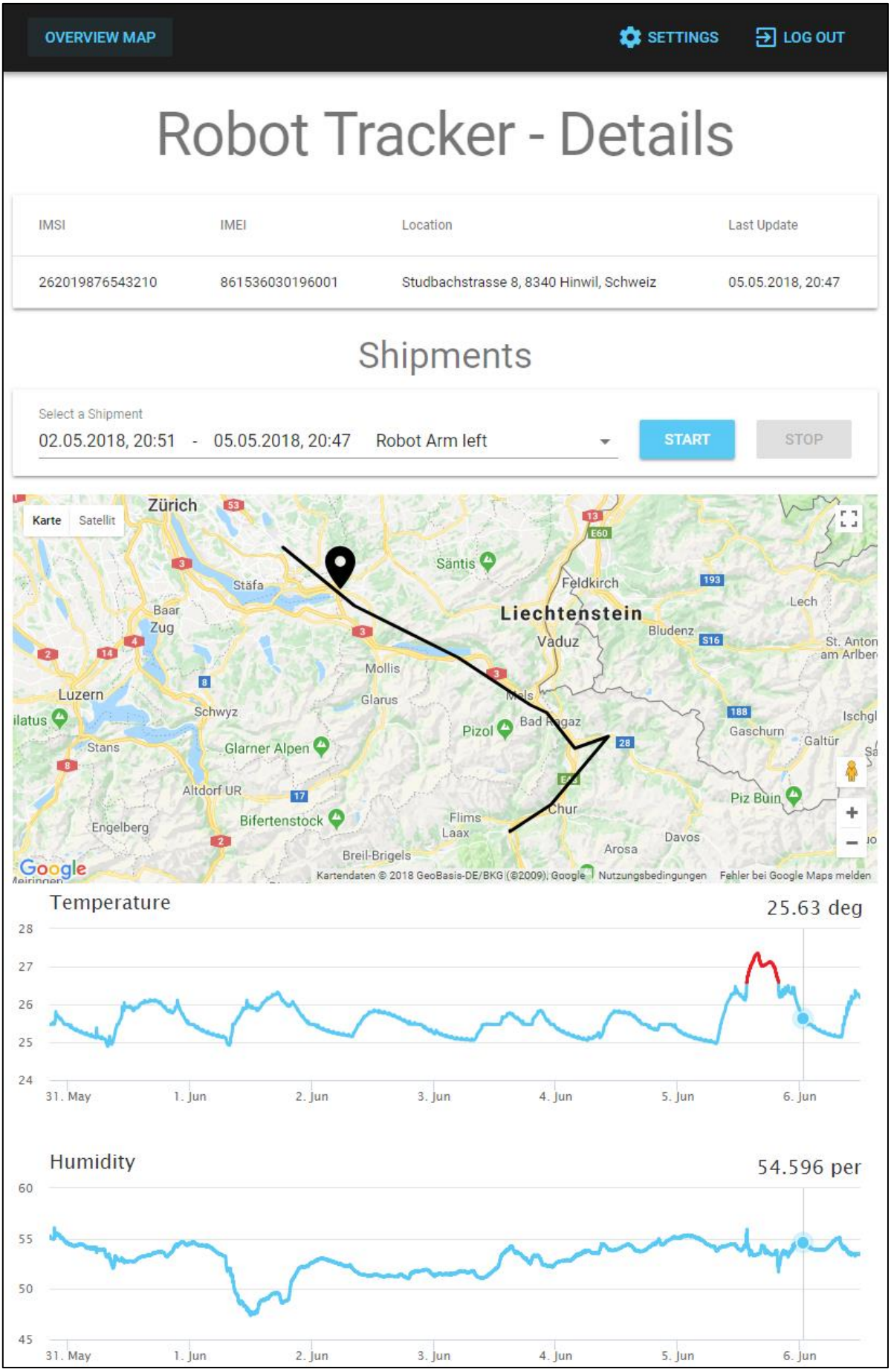

Abbildung 25 Detailansicht

Tracker Liste / Trackers

Für die Tracker Liste, siehe Abbildung 26, wurde kein Wireframe erstellt, da diese in einer sehr ähnlichen Form bereits im HiveWatch-Projekt vorhanden ist und von dort Grösstenteils übernommen werden konnte. Sie ist nur für Administratoren zugänglich und zeigt alle Informationen für alle im System registrierten Tracker an. Die Liste bietet Administratoren zwei Verwaltungsfunktionen.

- IMSI und IMEI der Tracker kann hier angepasst werden. Dies wird benötigt um die nahtlose Ersetzung von defekten Trackern zu ermöglichen. Anstatt den neuen Ersatz-Tracker zusätzlich zu registrieren, können IMSI und IMEI des defekten Trackers durch die des Ersatz-Trackers ersetzt werden. So merkt der Benutzer gar nichts von der Ersetzung und kann weiterarbeiten, als hätte er immer denselben Tracker verwendet.
- Neu registrierte Tracker können hier einem Benutzer zugewiesen werden.

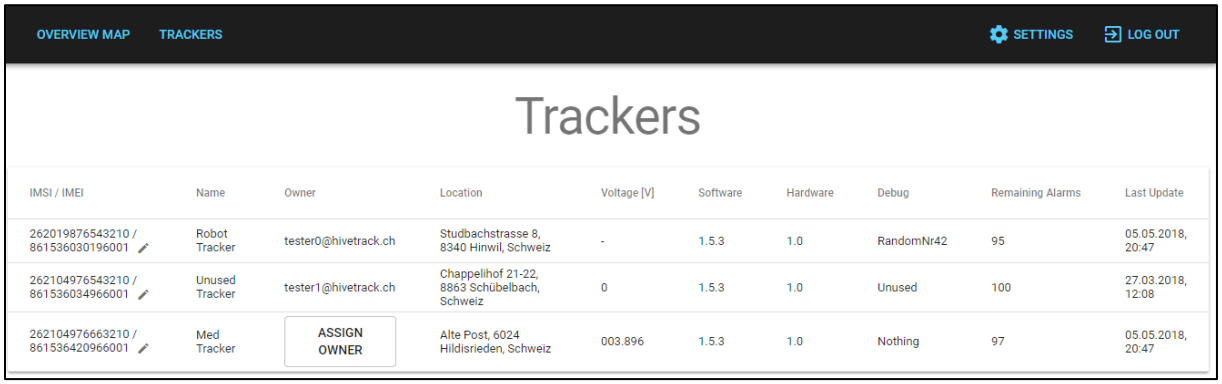

Abbildung 26 Tracker Liste

Einstellungen / Settings

Die Einstellungen umfassen mittlerweile fünf Tabs, von denen drei nur für Administratoren sichtbar sind.

1. Im ersten Tab, siehe Abbildung 27, kann der Benutzer seine Tracker und Einstellungsgruppen verwalten. Anders als auf der Übersichtskarte sehen Administratoren hier nur ihre eigenen Tracker und Einstellungsgruppen. Neu hinzugekommen ist die Möglichkeit für Benutzer ihre Tracker umzubenennen.
2. Der zweite Tab enthält eine einfache Accountverwaltung, in der Benutzer ihre Login Informationen, sprich E-Mail und Passwort, ändern können.
3. Im dritten Tab, siehe Abbildung 28, befindet sich die Benutzerverwaltung für Administratoren. Diese können ebenfalls E-Mails und Passwörter von sämtlichen Benutzern anpassen, Administratorenrechte vergeben, Benutzer de-/aktivieren und neue Benutzer erstellen. Ist ein Benutzer deaktiviert, bedeutet dies, dass er sich nicht mehr einloggen kann.
4. Der vierte Tab dient Administratoren dazu, neue Tracker zu registrieren. Dies wurde analog zum HiveWatch-System mit einer mehrzeiligen Textbox gelöst. In dieser können pro Zeile jeweils IMSI, IMEI und Mobile-Nr. eines neuen Trackers durch Kommas getrennt eingetragen werden. Durch Klicken des "Registrieren"-Buttons wird der Textbox-Inhalt nach dem erwähnten Schema geparst und die einzelnen Tracker im System erfasst. Der Grund für diese eher simple Lösung liegt darin, dass der Administrator üblicherweise eine Liste von zu erfassenden Trackern hat, die er dann nur noch in das Textfeld einfügen muss.
5. Der letzte Tab, siehe Abbildung 29, enthält die Sensorverwaltung. Hier können Administratoren jeder Sensor-ID eine Beschreibung und eine Einheit zuweisen. Diese werden dann für die Beschriftung der Charts in der Detailansicht verwendet. Zusätzlich gibt es pro Sensor ein "Admin Only"-Flag. Dieses bedeutet, dass für die entsprechenden Sensoren kein Chart angezeigt werden soll, da die Messwerte nur für Administratoren bestimmt sind, was beispielsweise bei der Chip Temperatur der Tracker der Fall ist. Diese Sensoren werden stattdessen in der Tracker Liste aufgeführt.

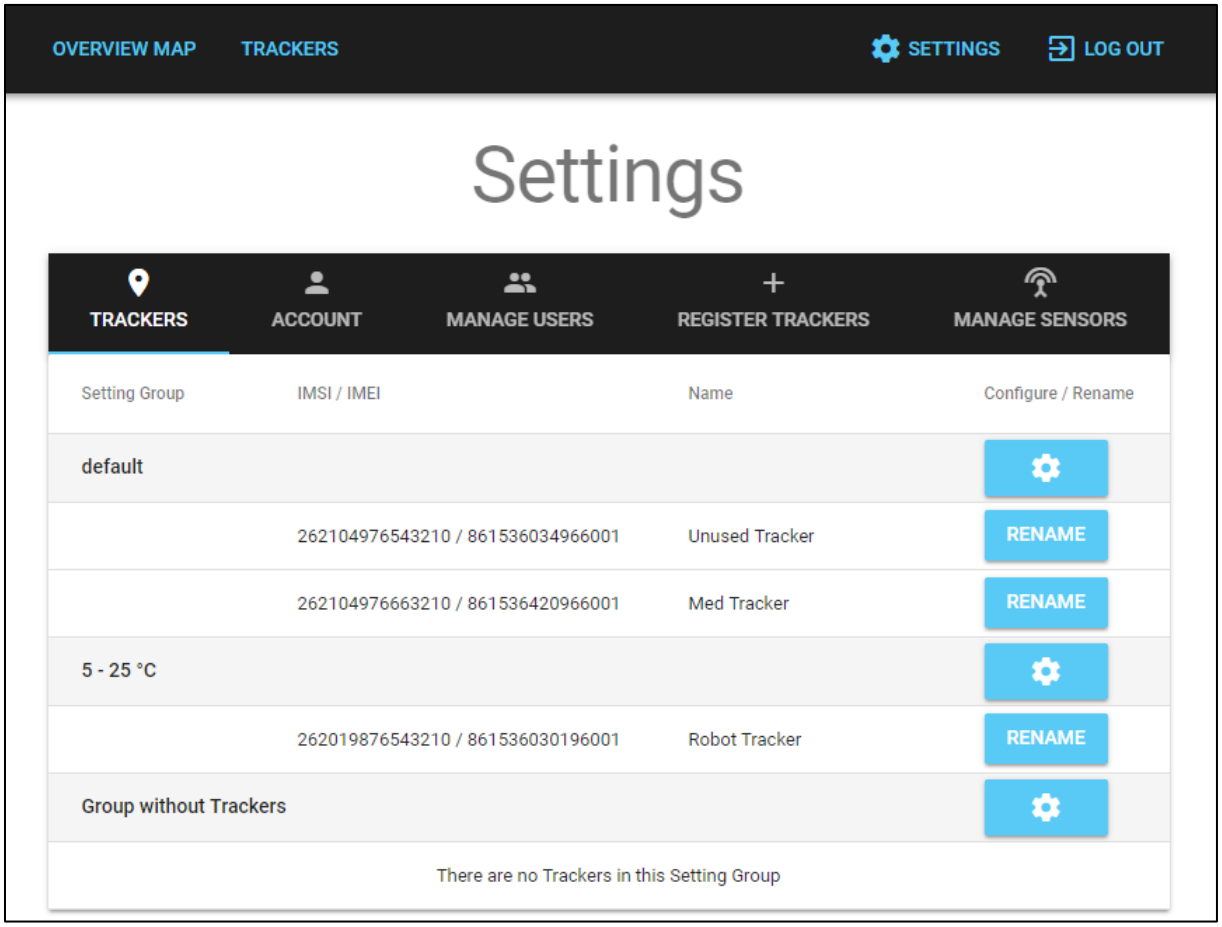

Abbildung 27 Tracker Einstellungen

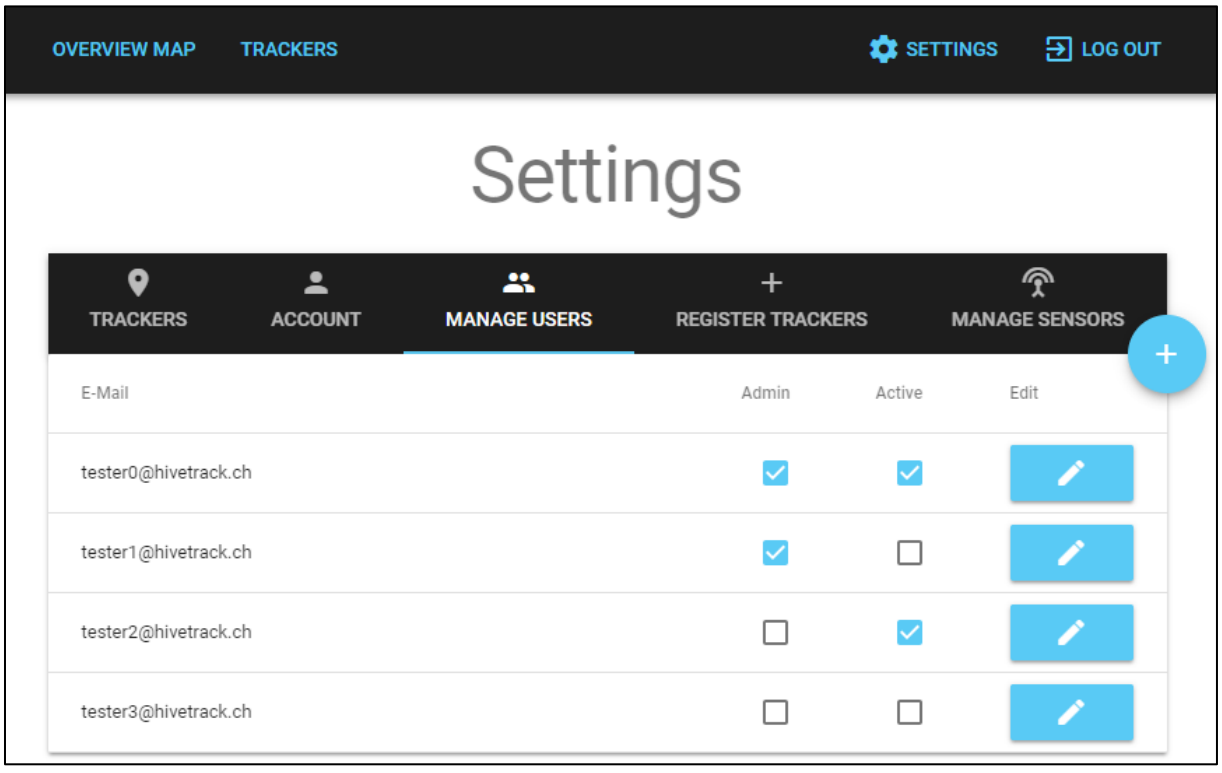

Abbildung 28 Benutzerverwaltung

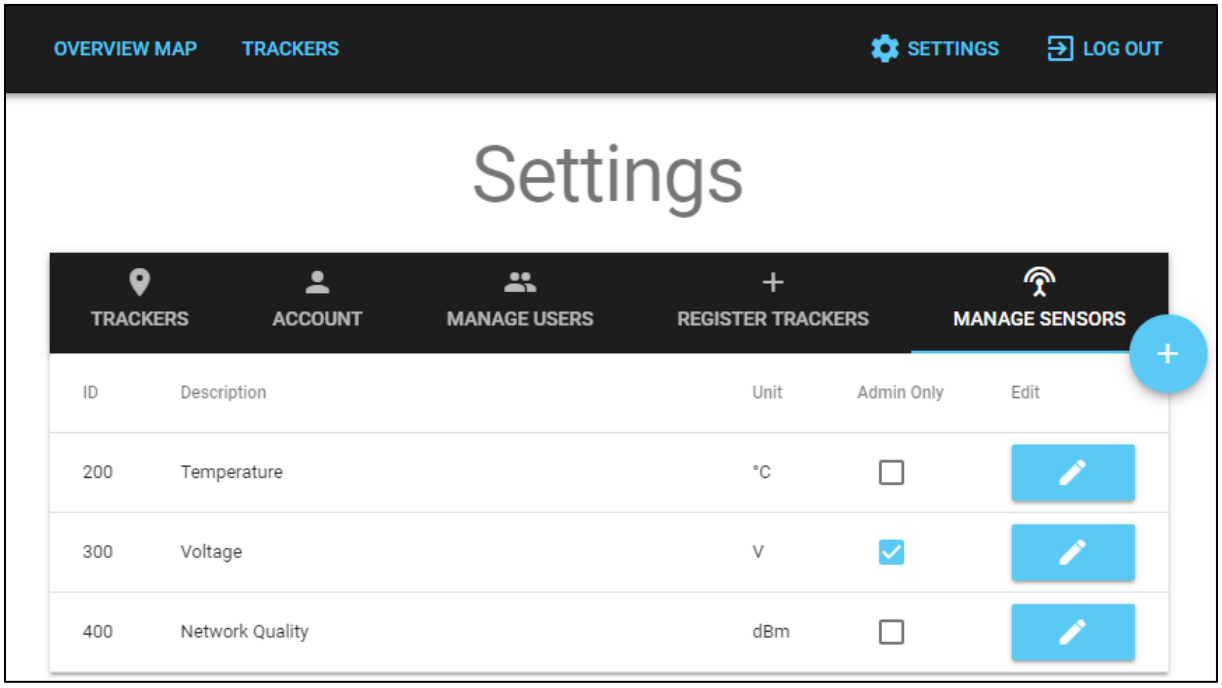

Abbildung 29 Sensorverwaltung

13.2 Immutability in JavaScript

Bei der Verwendung von React und Redux gibt es drei Orte, an denen State verwaltet wird:

1. Das State-Objekt jeder React Komponente.
2. Die Props (Properties) jeder React Komponente, welche wie Parameter von Funktionen angesehen werden können.
3. Der globale Redux Store.

Wie in den Guidelines von React [41] [42] und Redux [43] beschrieben, soll jeglicher State als read-only behandelt werden, sprich unveränderbar (immutable) sein. Die einzige Art State zu "verändern", ist ihn zu klonen und dann die Kopie zu verändern. Immutability und JavaScript sind aber zwei Begriffe, die nicht wirklich Hand in Hand gehen, da JavaScript keine Konstanten Objekte kennt und beinahe alles verändert und überschrieben werden kann. Es liegt also beim Entwickler, darauf zu Achten, nie irgendeine Art von State zu verändern.
Beim State-Objekt der React Komponenten ist dies relativ einfach, da man von React Unterstützung erhält. Mithilfe der `setState(...)`-Methode kann angegeben werden, was sich am State ändern soll. Diese klont dann den aktuellen State und ersetzt wo nötig die alten Werte durch die neuen.
Bei den Props und dem Redux Store muss dies selbst getan werden. Ein Objekt zu klonen und bestimmte Werte darin zu ersetzen geht am besten mit der `Object.assign(...)`-Methode oder der Destrukturierenden Zuweisung [44]. Dabei muss aber bedacht werden, dass diese beiden Varianten jeweils nur eine *Shallow Copy* des Objekts erstellen. Das heisst, dass Referenzen, die im Objekt enthalten sind, immer noch auf dieselben Objekte oder Arrays zeigen werden. Soll solch ein verschachteltes Objekt oder Array verändert werden, muss dieses ebenfalls zuerst geklont werden. Ein Beispiel dafür ist in Listing 16 ersichtlich.
Beim Verändern von Arrays können einige der von JavaScript definierten Array-Funktionen verwendet werden. Dabei muss aber immer darauf geachtet werden, dass diese Funktionen nicht die Ursprungs-Arrays verändern, sondern eine Kopie erstellen und diese verändern. Einige nützliche Funktionen sind:

- `Array.concat(...)` um Elemente einem Array hinzuzufügen
- `Array.filter(...)` um Elemente aus einem Array zu entfernen
- `Array.map(...)` um Elemente eines Arrays zu verändern

Beschreibung	**Operation**	**Ergebnis**

Original Objekt	`const outer = {`
	`  name: "outer",`
	`  inner: {`
	`  name: "inner",`
	`  nr: 42`
	`  }`
	`}`
		`outer = {`
		`  name: "outer",`
		`  inner: {`
		`  name: "inner",`
		`  nr: 42`
		`  }`
		`}`

Inneres Objekt klonen und verändern (Original bleibt unverändert)	`const innerClone = {`
	`  ...outer.inner,`
	`  name: "inner clone"`
	`}`
		`outer = {`
		`  name: "outer",`
		`  inner: {`
		`  name: "inner",`
		`  nr: 42`
		`  }`
		`}`
		
		`innerClone = {`
		`  name: "inner clone",`
		`  nr: 42`
		`}`

Äusseres Objekt klonen und Kopie des inneren Objekts einfügen (Original bleibt unverändert)	`const outerClone = {`
	`  ...outer,`
	`  inner: innerClone`
	`}`
		`outer = {`
		`  name: "outer",`
		`  inner: {`
		`  name: "inner",`
		`  nr: 42`
		`  }`
		`}`
		
		`innerClone = {`
		`  name: "inner clone",`
		`  nr: 42`
		`}`
		
		`outerClone = {`
		`  name: "outer",`
		`  inner: {`
		`  name: "inner clone",`
		`  nr: 42`
		`  }`
		`}`

Listing 16 Verschachtelte Objekte klonen und verändern

14 Testing

14.1 Unit Tests

Die Unit Tests wurden mithilfe von ScalaTest und Mockito geschrieben. Mit Mockito wird unter anderem auch die Datenbankverbindung gemockt, wodurch alle Tests ohne Datenbank ausgeführt werden können.

Timestamp Problematik

Beim Receiver wird intensiv mit Timestamps gearbeitet, was sich als eine grosse Herausforderung für das Schreiben der Tests herausstellte. Der Grund dafür ist, dass die vom Receiver und den Unit Tests generierten Timestamps immer ein paar Millisekunden voneinander abweichen, was einen Vergleich schwierig macht. Deshalb wurde zuerst versucht einen eigenen Equality Comparator zur Verfügung zu stellen, welcher einen sinnvollen Toleranzwert enthält. Doch auch mit diesem eigenen Equality Comparator, schlugen gewisse Tests manchmal fehl.
Nach einer Internet-Recherche wurde in einer Stackoverflow Diskussion eine bessere Möglichkeit gefunden. Die Timestamps werden mittels der JodaTime Library erstellt. JodaTime bietet die Möglichkeit an, systemweit die Zeit zu fixieren [45]. Dies wir nun genutzt, um zu Beginn der Tests die Zeit einzufrieren und so statische Timestamps zu erhalten, welche verglichen werden können.

14.2 Usability Test

Als die Übersichtskarte, Detailansicht und Benutzerverwaltung fertiggestellt waren, wurde mit zwei vom Projekt unabhängigen Personen ein Usability Test durchgeführt. Dabei wurden zwei möglichst unterschiedliche Personen gewählt. Eine arbeitet als Informatiker und hat tagtäglich mit der Bedienung von verschiedensten Programmen zu tun. Die andere kommt nicht aus einem technischen Umfeld und hat dementsprechend weniger Routine, wenn es um die Bedienung von Webseiten geht.
Im Folgenden werden die Erkenntnisse und Massnahmen dieses Tests kurz zusammengefasst. Die kompletten Testprotokolle sind in den Anhängen C.1 und C.2 zu finden.

14.2.1 Testablauf

Der Test ist wie folgt abgelaufen:

- Jede Testperson erhielt eine kurze Einführung, worum es bei der Software geht und was für eine Rolle sie bei diesem Test einnimmt.
- Dann sollte jede Testperson selbständig neun Aufgaben mithilfe der Software lösen.
- Währenddessen beobachtete einer der Entwickler die Testperson und notierte, ob sie die Software wie angedacht bedient, oder nicht.

14.2.2 Erkenntnisse

Grundsätzlich ist die Bedienung der Software intuitiv. Beide Testpersonen konnten alle Aufgaben ohne grössere Probleme lösen. Dabei blieben jedoch ein paar Features unentdeckt und eine Information wurde fehlinterpretiert.

- Auf der Übersichtskarte benutzte keine der Testpersonen die Filter. Dies hat aber auch mit den Testdaten zu tun, welche nur eine kleine Anzahl von Trackern enthielt. Deshalb war das Filtern der Tracker auch gar nicht nötig.
- In der Detailansicht hat keine der Testpersonen die Zoom-Funktion der Charts entdeckt. Diese hätte aber bei einer Aufgabe, in der ein bestimmter Wert abgelesen werden musste, geholfen.
- In der Hover-Info von Positions-Markern auf der Übersichtskarte werden bei Trackern mit fehlerhaften Werten während der letzten Paketsendung die entsprechenden Sensoren rot eingefärbt. Wie in Abbildung 30 gezeigt, betrifft diese Einfärbung jeweils den Sensor-Namen, wie auch den daneben angezeigten, aktuellen Messwert. Dies wurde von beiden Testpersonen so interpretiert, dass die rot eingefärbten Werte jeweils fehlerhafte Werte sind, die ausserhalb der definierten Grenzwerte liegen. Das ist aber nicht zwingend der Fall, da die Einfärbung unabhängig des aktuellen Messwertes gemacht wird.
- Bis zum Test war man sich uneinig darüber, ob die Start- und Stopp-Buttons auf der Übersichtskarte UND der Detailansicht benötigt werden. Da von den Testpersonen die Buttons an beiden Orten verwendet wurden, hat man sich darauf geeinigt dies so zu belassen.

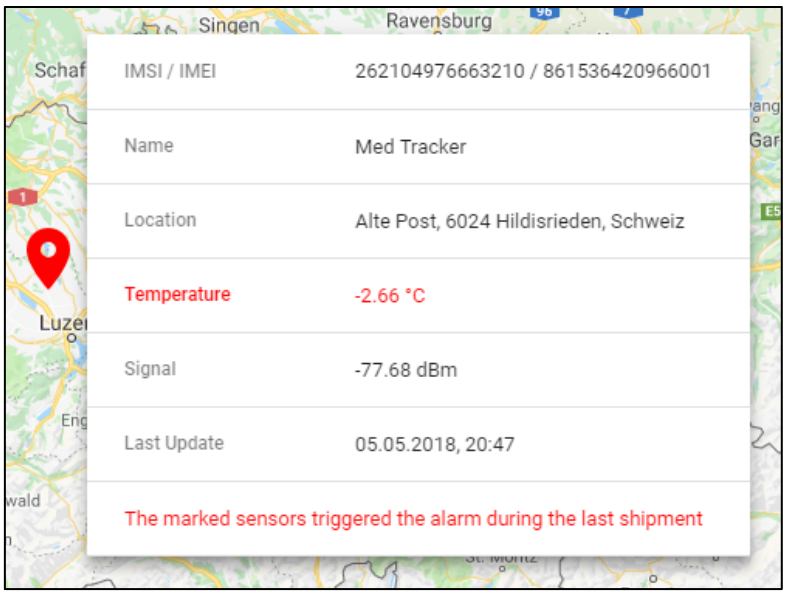

Abbildung 30 Hover-Info von Positions-Markern (alt)

14.2.3 Massnahmen

Basierend auf den Erkenntnissen des Tests wurden einige Massnahmen definiert und teilweise auch umgesetzt.

- Um die Zoom-Funktion der Charts klarer zu machen gibt es viele Varianten. Eine permanente Textbox, die beschreibt wie der Chart-Zoom funktioniert, war aber unerwünscht. Auch ein kurzes Tutorial, das beim ersten Login gezeigt wird, ist nicht optimal. Denn die meisten Benutzer sind von solchen Tutorials oftmals überfordert und genervt, da sie noch gar nichts von der Applikation gesehen haben. Am besten wäre deshalb eine kurze Beschreibung des Zooms, die auftaucht nachdem der Benutzer beispielsweise 10-Mal die Detailansicht besucht, aber den Zoom noch nicht benutzt hat. Dies ist jedoch recht aufwändig und wurde deshalb noch nicht umgesetzt.
- Das Problem mit der Hover-Info war relativ leicht zu lösen. Wie in Abbildung 31 gezeigt, werden die aktuellen Messwerte nur noch dann rot eingefärbt, wenn sie auch wirklich ausserhalb der definierten Grenzwerte liegen. Zusätzlich wurde die Fehlermeldung unterhalb der Infos angepasst.

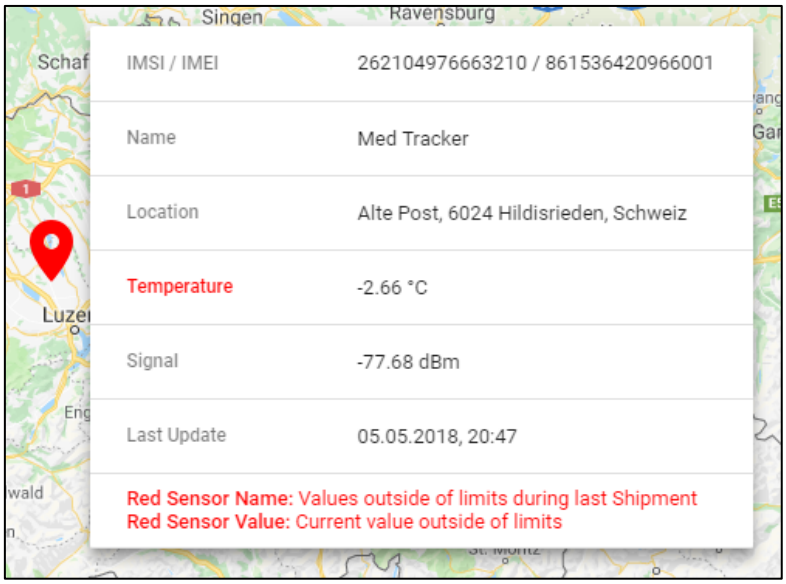

Abbildung 31 Hover-Info von Positions-Markern (neu)

14.3 Systemtest

Am Ende der Implementations-Phase wurde ein umfassender Systemtest durchgeführt. Dafür wurden pro umgesetzten Use Case ein oder mehrere Test Cases definiert. Diese wurden dann mit fixen Testdaten, die für jeden Test Case zurückgesetzt wurden, durchgeführt. Da aber nicht bei allen Test Cases direkt im Frontend alle Auswirkungen ersichtlich sind, wurde der Test als White-Box-Test durchgeführt und jeweils auch die Kommunikation zwischen Back- und Frontend und die Datenbank beobachtet.
Da alle neuen Features, bevor sie in das Live-System deployed wurden, immer zuerst von beiden Entwicklern auf ihren lokalen Systemen getestet wurden, ist das Ergebnis des Systemtests, dass alle Test Cases wie Erwartet funktionieren, wenig überraschend.
Das komplette Protokoll des Systemtests und eine Beschreibung der verwendeten Testdaten ist im Anhang C.3 zu finden.

IV Ergebnisse

15 Umsetzung der Funktionalen Anforderungen

Von den 14 Use Cases konnten insgesamt 9 umgesetzt werden. Um welche Use Cases es sich dabei handelt und wo die entsprechende Funktionalität umgesetzt wurde, ist in Listing 17 ersichtlich.

Use Case	Umgesetzt?	Wo?
UC 01: Übersichtskarte	✓	Übersichtskarte
UC 02: Detailansicht	✓	Detailansicht
UC 03: Einstellungsgruppen verwalten	✗	
UC 04: Standard Einstellungsgruppe konfigurieren	✓	Einstellungen, Tab 1
UC 05: Benachrichtigung	✗	
UC 06: Alarmhistory	✓	Detailansicht, Charts
UC 07: Start/Stopp Paketsendung	✓	Übersichtskarte / Detailansicht
UC 08: Paketsendungen bearbeiten	✗	
UC 09: CSV Export	✗	
UC 10: Heatmap	✗	
UC 11: Tracker um-/benennen	✓	Einstellungen, Tab 1
UC 12: Tracker verwalten	✓	Tracker Liste
UC 13: Benutzer verwalten	✓	Einstellungen, Tab 3
UC 14: Messdaten speichern	✓	Receiver

Listing 17 Umsetzung der Use Cases

16 Umsetzung der Nichtfunktionalen Anforderungen

Von den acht nichtfunktionalen Anforderungen konnten alle Umgesetzt werden.

Sicherheit

Die vier Sicherheitsanforderungen konnten problemlos mithilfe der Silhouette Library umgesetzt werden. Jeder Benutzer kann nur seine eigenen Daten einsehen und nur Administratoren haben Zugriff auf alle Tracker und können neue Benutzer anlegen.

Performance

Das Laden der Übersichtskarte dauert auch mit ein paar hundert Trackern weniger als eine Sekunde und erfüllt somit die Anforderung, für bis zu 100 Tracker in maximal 2 Sekunden geladen zu sein. Auch die Detailansicht erfüllt die Anforderung, für Paketsendungen die bis zu 5 Tage dauern in 4 Sekunden geladen zu sein. Dies dauert jedoch 3 bis 3.5 Sekunden und ist somit schon nahe am Limit. Hier muss beobachtet werden, wie das System in der Praxis eingesetzt wird und wie lange Paketsendungen normalerweise dauern. Basierend auf den daraus gewonnenen Erkenntnissen kann die Detailansicht dann optimiert werden.

Bedienbarkeit

Wie gewünscht wurde die Applikation für Desktops entwickelt und kann in den meistbenutzten Browsern (Google Chrome, Mozilla Firefox, Microsoft Edge) problemlos dargestellt und bedient werden. Auch wenn das Design nicht für Tablets und Smartphones optimiert wurde, ist die komplette Funktionalität auch auf diesen Geräten nutzbar.

Datenpersistenz

Ein Test-Tracker sendet bereits seit einigen Tagen Messdaten an den Receiver, welcher diese ohne Probleme verarbeitet und gespeichert hat.

17 Ausblick

Abgesehen von den in Kapitel 15 erwähnten Use Cases, die noch nicht umgesetzt werden konnten, gibt es noch eine Reihe weiterer Verbesserungs- und Ausbaumöglichkeiten.

17.1 Bekannte Probleme

Es existiert noch eine bekannter Bug im System, der aufgrund seiner niedrigen Priorität noch nicht gefixt wurde. Er hat mit der Hover-Info auf der Übersichtskarte zu tun, die erscheint, wenn man den Cursor über einen der Positions-Marker bewegt. Aktiviert man den Vollbild-Modus der Google Map, erscheint diese Hover-Info nur noch in Google Chrome, aber nicht in Mozilla Firefox oder Microsoft Edge. Das liegt daran, dass jeder Browser den Vollbild-Modus anders handhabt. Da die Hover-Info aber ohne Vollbild-Modus in allen Browsern problemlos funktioniert und dieser voraussichtlich nicht sehr oft Verwendung finden wird, wurde entschieden den Bug noch nicht zu fixen und die Zeit für wichtigeres zu nutzen.

17.2 Erweiterungen

Bevor irgendwelche Erweiterungen entwickelt werden, will die FPGA Company GmbH das System in einem grösseren Feldversuch testen und Feedback der Endkunden einholen. Es wurden aber dennoch einige mögliche Erweiterungen besprochen.
Falls beispielsweise die Paketsendungen oft mehrere Wochen dauern und die Detailansicht, wie in Kapitel 16 beschrieben, zu lange braucht um die Messdaten vom Server zu laden, kann diese auf verschiedene Arten optimiert werden. Eine Variante, die bereits im HiveWatch-System Einsatz findet, ist, vorerst nur die stündlichen Spitzenwerte und erst beim hereinzoomen weitere Werte zu laden.
Abgesehen von solchen Performance-Optimierungen gibt es noch Verbesserungsmöglichkeiten bei der Bedienungsfreundlichkeit. Zum Beispiel können aktuell die Formulare nicht mit der Enter-Taste abgeschickt werden.
Es wurde auch noch ein neues Feature für die Detailansicht besprochen. Momentan kann dort durch bewegen des Cursors über eines der Charts die Position des Trackers zu diesem Zeitpunkt auf der Karte angezeigt werden. Zusätzlich dazu soll beim hereinzoomen in ein Chart die Strecke, die der Tracker während des ausgewählten Zeitraums zurückgelegt hat, auf der Karte farblich markiert werden. Dies wäre relativ einfach durch Zeichnen einer zweiten Linie in einer anderen Farbe umzusetzen.
Hier ist aber nur ein Bruchteil der beinahe endlosen Ausbaumöglichkeiten beschrieben. Was in der Praxis wirklich benötigt wird, muss der Feldversuch zeigen.

V Glossar

Begriff	Beschreibung
BSON	MongoDB repräsentiert JSON-Dokumente im binärcodierten Format namens BSON. BSON erweitert JSON um zusätzliche Datentypen und geordnete Felder.
GSM	Das Global System for Mobile Communications ist ein Mobilfunkstandard für volldigitale Mobilfunknetze.
IMEI	Die International Mobile Station Equipment Identity ist eine eindeutige 15-stellige Seriennummer, anhand derer jedes GSM-Endgerät weltweit eindeutig identifiziert werden kann.
IMSI	Die International Mobile Subscriber Identity dient in Mobilfunknetzen der eindeutigen Identifizierung der Netzteilnehmer und wird jeweils auf der SIM-Karte gespeichert.
JSON	Die JavaScript Object Notation ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen.
Promise	Ein Promise ist ein JavaScript Objekt das den Rückgabewert einer noch nicht abgeschlossenen asynchronen Operation darstellt.
REST / RESTful	Representational State Transfer bezeichnet ein Programmierparadigma für verteilte Systeme, insbesondere für Webservices.
Shallow Copy	Eine Kopie eines Objekts, bei dem jegliche kopierte Referenzen immer noch auf dieselben Objekte zeigen.
Tracker	Ein von der FPGA Company GmbH entwickeltes Gerät bestückt mit verschiedenen Sensoren (Temperatur, Feuchtigkeit, Beschleunigung, ...).
XMLHttpRequest	XMLHttpRequest ist ein JavaScript Objekt, das von Microsoft entwickelt und von Mozilla, Apple, und Google übernommen wurde. Es bietet einen einfachen Weg, Daten von einem URL zu erhalten.

VI Verweise

[1] P. Haack , „The meaning of "Impedance Mismatch".," 15 Juni 2004. [Online]. Available: https://haacked.com/archive/2004/06/15/impedance-mismatch.aspx/. [Zugriff am 3. Juni 2018].

[2] M. Fowler, „GOTO 2012 • Introduction to NoSQL • Martin Fowler," 19 Februar 2013. [Online]. Available: https://www.youtube.com/watch?v=qI_g07C_Q5I. [Zugriff am 3. Juni 2018].

[3] Zengularity, Lightbend, „ThreadPools - 2.6.x," [Online]. Available: https://www.playframework.com/documentation/2.6.x/ThreadPools. [Zugriff am 3. Juni 2018].

[4] StaxMan, „JSON Naming Convention," 9 Juni 2011. [Online]. Available: https://stackoverflow.com/questions/5543490/json-naming-convention. [Zugriff am 4. Juni 2018].

[5] P. Krzysztof, „Dependency injection in Play Framework using Scala," 7 April 2016. [Online]. Available: https://www.schibsted.pl/blog/dependency-injection-play-framework-scala/. [Zugriff am 3. Juni 2018].

[6] Zengularity, Lightbend, „Play Framework - Build Modern & Scalable Web Apps with Java and Scala," [Online]. Available: https://www.playframework.com/. [Zugriff am 3. Juni 2018].

[7] ReactiveMongo, „ReactiveMongo for Play Framework," [Online]. Available: https://github.com/ReactiveMongo/Play-ReactiveMongo. [Zugriff am 3. Juni 2018].

[8] A. Warski, „Lightweight and Nonintrusive Scala Dependency Injection Library," [Online]. Available: https://github.com/adamw/macwire. [Zugriff am 3. Juni 2018].

[9] Mohiva, „Introduction," [Online]. Available: https://www.silhouette.rocks/docs/introduction. [Zugriff am 3. Juni 2018].

[10] SmartBear Software, „OpenAPI Specification and Swagger," [Online]. Available: https://swagger.io/solutions/getting-started-with-oas/. [Zugriff am 4. Juni 2018].

[11] iHeartRadio, „Swagger API spec generator for Play," [Online]. Available: https://github.com/iheartradio/play-swagger. [Zugriff am 4. Juni 2018].

[12] The Cats Maintainers, „Cats," [Online]. Available: https://typelevel.org/cats/. [Zugriff am 4. Juni 2018].

[13] The Cats Maintainers, „OptionT," [Online]. Available: https://typelevel.org/cats/datatypes/optiont.html. [Zugriff am 4. Juni 2018].

[14] Terracotta, Inc., „Quartz Enterprise Job Scheduler," [Online]. Available: http://www.quartz-scheduler.org/. [Zugriff am 4. Juni 2018].

[15] S. Hopper, „akka-quartz-scheduler," [Online]. Available: https://github.com/enragedginger/akka-quartz-scheduler. [Zugriff am 4. Juni 2018].

[16] B. Venners und Artima, „ScalaTest," [Online]. Available: http://www.scalatest.org/. [Zugriff am 4. Juni 2018].

[17] J. Tecson, „FAQ," 1 Februar 2017. [Online]. Available: https://github.com/mockito/mockito/wiki/FAQ. [Zugriff am 4. Juni 2018].

[18] V. Sahni, „Best Practices for Designing a Pragmatic RESTful API," 12 August 2015. [Online]. Available: https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api. [Zugriff am 5. Juni 2018].

[19] M. York, „Trailing slash in RESTful API," 13 Februar 2013. [Online]. Available: https://softwareengineering.stackexchange.com/questions/186959/trailing-slash-in-restful-api. [Zugriff am 7. Juni 2018].

[20] Sun Microsystems, Inc., „Core J2EE Patterns - Data Access Object," 29 März 2013. [Online]. Available: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html. [Zugriff am 3. Juni 2018].

[21] Facebook Inc., „React - A JavaScript library for building user interfaces," [Online]. Available: https://reactjs.org/. [Zugriff am 4. Juni 2018].

[22] „Redux," [Online]. Available: https://redux.js.org/. [Zugriff am 4. Juni 2018].

[23] „Usage with React - Redux," [Online]. Available: https://redux.js.org/basics/usage-with-react. [Zugriff am 4. Juni 2018].

[24] Highsoft, „Interactive JavaScript charts for your webpage | Highcharts," [Online]. Available: https://www.highcharts.com/. [Zugriff am 4. Juni 2018].

[25] Highsoft, „highcharts/highcharts-react," [Online]. Available: https://github.com/highcharts/highcharts-react. [Zugriff am 4. Juni 2018].

[26] H. Nguyen, „Material-UI," [Online]. Available: https://material-ui.com/. [Zugriff am 4. Juni 2018].

[27] Google, „Material Design," [Online]. Available: https://material.io/. [Zugriff am 4. Juni 2018].

[28] T. Chen, „tomchentw/react-google-maps," [Online]. Available: https://github.com/tomchentw/react-google-maps. [Zugriff am 4. Juni 2018].

[29] React Training, „ReactTraining/react-router," [Online]. Available: https://github.com/ReactTraining/react-router/tree/master/packages/react-router-dom. [Zugriff am 4. Juni 2018].

[30] M. Russell, „mjrussell/redux-auth-wrapper," [Online]. Available: https://github.com/mjrussell/redux-auth-wrapper. [Zugriff am 4. Juni 2018].

[31] axios, „axios/axios," [Online]. Available: https://github.com/axios/axios. [Zugriff am 4. Juni 2018].

[32] SmartBear, „Swagger UI," [Online]. Available: https://swagger.io/tools/swagger-ui/. [Zugriff am 4. Juni 2018].

[33] ShellHacks, „HowTo: Disable SSH Host Key Checking," 27 Dezember 2016. [Online]. Available: https://www.shellhacks.com/disable-ssh-host-key-checking/. [Zugriff am 4. Juni 2018].

[34] Will, „What does bash -s do?," 14 Mai 2016. [Online]. Available: https://stackoverflow.com/questions/37224634/what-does-bash-s-do. [Zugriff am 4. Juni 2018].

[35] A. Mouat, „Running Docker in Jenkins (in Docker)," 11 März 2015. [Online]. Available: http://container-solutions.com/running-docker-in-jenkins-in-docker/. [Zugriff am 4. Juni 2018].

[36] C. Maier, „The Importance of MongoDB Key Names," 22 Mai 2011. [Online]. Available: http://christophermaier.name/2011/05/22/MongoDB-key-names/. [Zugriff am 5. Juni 2018].

[37] Zengularity; Lightbend, „Protecting against Cross Site Request Forgery," [Online]. Available: https://www.playframework.com/documentation/2.6.x/ScalaCsrf. [Zugriff am 11. Juni 2018].

[38] M. Fowler, „Schemaless Data Structures," 7 Januar 2013. [Online]. Available: https://martinfowler.com/articles/schemaless/fallback.html. [Zugriff am 8. Juni 2018].

[39] Google, „Google Maps Geocoding API," [Online]. Available: https://developers.google.com/maps/documentation/geocoding/start. [Zugriff am 4. Juni 2018].

[40] Highsoft, „Highcharts Zones," [Online]. Available: https://www.highcharts.com/docs/chart-concepts/series#zones. [Zugriff am 6. Juni 2018].

[41] Facebook Inc., „State and Lifecycle - React," [Online]. Available: https://reactjs.org/docs/state-and-lifecycle.html#using-state-correctly. [Zugriff am 10. Juni 2018].

[42] Facebook Inc., „Components and Props - React," [Online]. Available: https://reactjs.org/docs/components-and-props.html#props-are-read-only. [Zugriff am 10. Juni 2018].

[43] „Three Principles - Redux," [Online]. Available: https://redux.js.org/introduction/three-principles#state-is-read-only. [Zugriff am 10. Juni 2018].

[44] Mozilla, „Destrukturierende Zuweisung - JavaScript," [Online]. Available: https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Operators/Destrukturierende_Zuweisung. [Zugriff am 9. Juni 2018].

[45] L. Pireyn, „Time dependent unit tests," 11 April 2011. [Online]. Available: https://stackoverflow.com/questions/5622194/time-dependent-unit-tests. [Zugriff am 11. Juni 2018].

VII Abbildungsverzeichnis

VIII Listingverzeichnis

IX Anhang

Anhang A: Quick Start Guide

Ein kurzer Leitfaden für die Benutzer von HiveTrack.

Übersichtskarte / Overview Map

- Verschaffen Sie sich einen Überblick über die aktuellen Positionen all ihrer Tracker mittels der Karte. Rote Positions-Marker bedeuten, dass bei der letzten (oder aktuell laufenden) Paketsendung mindestens ein Messwert ausserhalb der festgelegten Limits gemessen wurde. Grüne Positions-Marker stehen für Tracker, bei denen alles in Ordnung war.
- Bewegen Sie den Cursor über einen Positions-Marker um die aktuellen Messwerte zu sehen.
- Blenden Sie mithilfe des Filters rechts oberhalb der Karte einen Teil der Tracker ein/aus.
- Blenden Sie durch die Checkboxen in der Liste unterhalb der Karte einzelne Tracker ein/aus.
- Heben Sie durch klicken eines Listeneintrags den entsprechenden Positions-Marker hervor.
- Starten/Stoppen Sie Paketsendungen über die entsprechenden Buttons in der Liste. Fügen Sie den Paketsendungen beim Starten eine Beschreibung hinzu, um sie später einfach wiederzuerkennen.
- Öffnen Sie durch Klicken auf einen Positions-Marker oder den entsprechenden Button in der Liste die Detailansicht eines Trackers.

Detailansicht / Tracker Details

- Wählen Sie eine der Paketsendungen des Trackers aus, um dann deren Daten zu analysieren.
- Starten Sie eine neue Paketsendung, oder stoppen Sie die ausgewählte über die entsprechenden Buttons. Fügen Sie den Paketsendungen beim Starten eine Beschreibung hinzu, um sie später einfach wiederzuerkennen.
- Verfolgen Sie auf der Karte den Weg des Trackers während der Paketsendung.
- Analysieren Sie die Messwerte des Trackers mithilfe der Diagramme unterhalb der Karte. Rot markierte Werte befinden sich ausserhalb der für diese Paketsendung festgelegten Limits.
- Bewegen Sie den Cursor über ein Diagramm, um den genauen Messwert und die ungefähre Position zu einem bestimmten Zeitpunkt zu sehen.
- Zoomen Sie in ein Diagramm hinein, um grössere Datenmengen einfacher auswerten zu können. Zum Zoomen halten Sie die linke Maustaste gedrückt und ziehen den Cursor über das Diagramm, um den Bereich auszuwählen, in den hereingezoomt werden soll. Um den Zoom zurückzusetzen, klicken Sie auf den "Reset Zoom"-Button, der auf dem Diagramm erscheint.

Einstellungen / Settings

Tab 1: Trackers

- Konfigurieren Sie hier über den Button mit dem Zahnrad-Symbol die Limits für alle Ihre Tracker gleichzeitig. Die konfigurierten Limits werden für alle folgenden Paketsendungen verwendet und können jederzeit wieder geändert oder entfernt werden.
- Geben Sie Ihren Trackern mittels des "Rename"-Buttons einen Namen, um sie später einfach identifizieren zu können.

Tab 2: Account

- Ändern Sie hier ihre Anmeldedaten (E-Mail & Passwort) für HiveTrack.

Anhang B: Projektplan

Projekt: HiveTrack

Projektplan

Arooran Thanabalasingam
Pascal Schweizer

1 Projekt Übersicht

Die FPGA Company GmbH entwickelt ein Gerät das periodisch Messdaten (Position, Feuchtigkeit, Temperatur, Beschleunigung) erfasst. Diese Daten werden über ein GSM Modem an einen Server geschickt. Dieser empfängt und persistiert die Daten und reichert sie mit zusätzlichen Informationen (Wetter) an.

Nun sollen Client und Server zur Verwaltung und Visualisierung dieser Daten entwickelt werden. Ein bereits bestehendes System (HiveWatch) kann für den Empfang und die Speicherung wiederverwendet werden, auch das Deployment lässt sich analog dem bestehenden System wiederverwenden. Front- und Backend der Web-Anwendung sollen aber neu entwickelt werden.

1.1 Zweck und Ziel

- Der Benutzer kann auf einer Übersichtskarte alle seine im Einsatz befindlichen Geräte auf einen Blick sehen.
- Pro Gerät kann der Verlauf der verschiedenen Sensordaten (Position, Feuchtigkeit, Temperatur, Beschleunigung) in einer Detailansicht eingesehen werden.
- Bei Über-/Unterschreitung von festgelegten Grenzwerten kann der Benutzer benachrichtigt werden.

1.2 Lieferumfang

- Software
- Dokumentation
- Projektplan
- Testprotokolle
- Sitzungsprotokolle
- Zeiterfassung und Zeitauswertung
- Persönliche Berichte
- Erklärung zur eigenständigen Durchführung der Arbeit
- Bedienungsanleitung

1.3 Annahmen und Einschränkungen

Durch die Festsetzung des Arbeitspensums von 240, bzw. 360 Stunden pro Projektmitarbeiter ist es möglich, dass die Software nicht rechtzeitig fertig wird. Um dem vorzubeugen wird die Zeitplanung überwacht und wenn nötig zusätzlicher Aufwand in einem sinnvollen Mass betrieben. Sollte dies nicht ausreichen, werden die Anforderungen frühzeitig angepasst. Falls wir früher fertig werden, können noch optionale Erweiterungen implementiert werden.

2 Projektorganisation

Das Projektteam besteht aus vier Personen. Dem Dozenten, der für die Betreuung des Projektes zuständig ist, dem Product Owner und den beiden Studenten/Programmierern.

2.1 Organisationsstruktur

Mirko Stocker	Silvio Ziegler	Arooran Thanabalasingam	Pascal Schweizer
• Betreuer	• Product Owner	• Programmierer	• Programmierer

3 Management Abläufe

3.1 Besprechungen

In wöchentlichen Meetings, an denen nach Möglichkeit alle vier Projektmitglieder teilnehmen, werden jeweils die Ergebnisse der letzten Woche präsentiert und die Arbeiten der darauffolgenden Woche priorisiert. Diese Meetings finden jeweils am Mittwoch um 15:00 statt.

3.2 Zeitliche Planung

Da dieses Projekt eine gemischte Semester- und Bachelorarbeit ist, stehen den beiden Programmierern jeweils 240 bzw. 360 Stunden zur Verfügung. Das heisst insgesamt 600 Mannarbeitsstunden. Das Projekt läuft vom 19.02.2018 bis 15.06.2018. Dies sind 17 Wochen, wovon sich die ersten 15 noch während der Unterrichtszeit und die letzten 2 in den Semesterferien befinden. Aus diesem Grund wird während der letzten 2 Wochen mehr am Projekt gearbeitet, was sich auch in der Zeitplanung wiederspiegelt.
Die 17 Semesterwochen, welche jeweils von Montag bis Sonntag dauern, werden in 16 Projektwochen aufgeteilt. Die Projektwochen starten und enden jeweils am Mittwoch um 15:00, da zu dieser Zeit auch die wöchentlichen Meetings stattfinden. Es sind nur 16 Projektwochen, da die erste und letzte etwas länger sind, was daran liegt, dass die offiziellen Projektstart, bzw. Enddaten auf einen Montag, bzw. Freitag fallen.

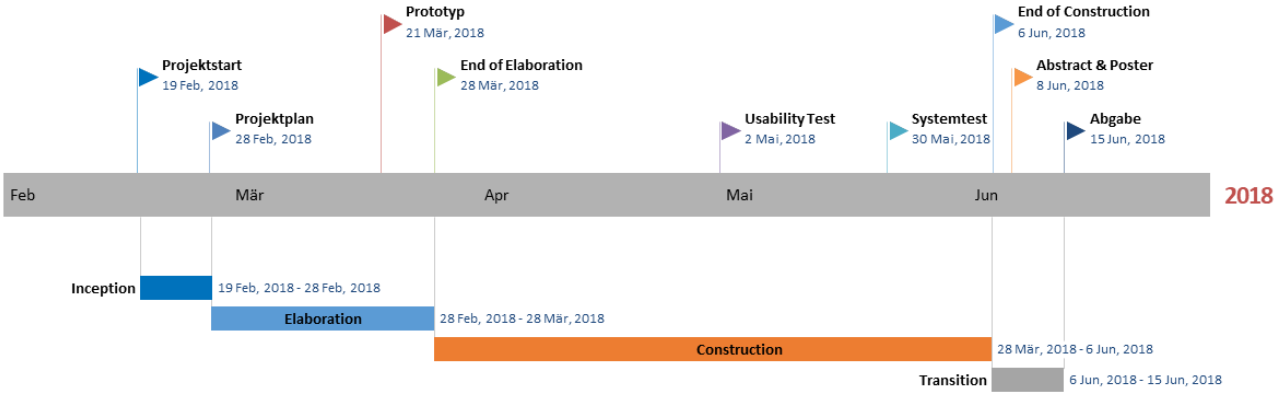

Abbildung 1: Zeitplanung – 27.02.2018

3.2.1 Meilensteine

KW	PW	Phase	Meilenstein	Datum	Beschreibung
8	1	**Inception**	Projektstart	19.02.2018	Projektstart
			Projektplan	28.02.2018	Erste Version des Projektplans Risikoanalyse Meilensteinplanung
9	2	**Elaboration**			
10	3				
11	4		Prototyp	21.03.2018	Architektur Prototyp
12	5		End of Elaboration	28.03.2018	Use Cases brief (wichtige casual) Wireframes Datenstruktur / Domain Model Deployment Model Nicht-Funktionale Anforderungen
13	6	**Construction**			
14	7				
15	8				
16	9				
17	10		Usability Test	02.05.2018	Usability Test
18	11				
19	12				
20	13				
21	14		Systemtest	30.05.2018	Systemtest
22	15		End of Construction	06.06.2018	Code Freeze
23	16	**Transition**	Abstract & Poster	08.06.2018	Online Abstract & A0 Poster
24			Abgabe	15.06.2018	Projektabgabe

Legende
KW – Kalenderwoche (Mo – So)
PW – Projektwoche (Mi – Mi, Ausnahmen: erste & letzte PW)

4 Risikomanagement

4.1 Risiken

Projekt		HiveTrack					
Erstellt am		22.02.2018					
Gewichteter Schaden		42					
Nr	**Titel**	**Beschreibung**	**max. Schaden [h]**	**Eintritts-wahrscheinlichkeit**	**Gewichteter Schaden**	**Vorbeugung**	**Verhalten beim Eintreten**
R1	Komplikationen bei der Implementation	Unvorhergesehene Komplikationen bei der Implementation treten auf	44	30%	13.2	Genügend Zeitpuffer für Arbeitspakete einplanen, Regelmässige Risikoanalyse	Frühzeitig melden, Zeitplan anpassen, Alternativlösung suchen
R2	CI-Server Automatische Builds	Es entstehen Probleme beim Build-Prozess auf dem CI-Server	20	15%	3	Bewährte Lösungsansätze des Vorprojekts (HiveWatch) als Beispiel verwenden	Mitglieder des Vorprojekts fragen
R3	Dokumentation	Erstellte Arbeiten werden nicht gut genug dokumentiert oder Detailierungsgrad ist nicht ausreichend	28	20%	5.6	Alles verständlich und zeitnah dokumentieren und kommunizieren	Sensibilisierung der Projektmitglieder für Dokumentation
R4	Anforderungen	Stark wechselnde Anforderungen	28	5%	1.4	Genaue Definition während der Elaboration	Änderungen einschätzen, überprüfen und eventuell vornehmen
R5	Ausfall im Projektteam	Ein Projektmitglied fällt aufgrund Krankheit/Unfall aus	40	10%	4	Zeitpuffer einplanen, Kein Bungee-Jumping ;D	Flexibel planen, sodass Arbeitspakete auch verzögert werden können, Verlorene Zeit im Nachhinein teilweise aufholen
R6	Zeitliche Fehleinschätzung	Die benötigte Zeit für Arbeitspakete wird stark unterschätzt	32	35%	11.2	Benötigte Zeit eher zu hoch schätzen, Zeitpuffer einplanen	Scope verkleinern, Mehrarbeit
R7	Know-How	Fehlendes Know-How in eingesetzten Technologien (v.a. React)	24	15%	3.6	Know-How mittels Tutorials vertiefen	Know-How aufbauen
Summe			**216**		**42**		

4.2 Umgang mit Risiken

Bei allen Arbeitspaketen wird die Dauer absichtlich etwas zu hoch eingeschätzt, um bei Problemen möglichst genug Zeit für dessen Lösung zu haben.
Falls Probleme auftreten, werden diese in GitHub erfasst. Wenn möglich werden sie noch im selben Sprint gelöst und ansonsten auf den nächsten verschoben.

5 Infrastruktur

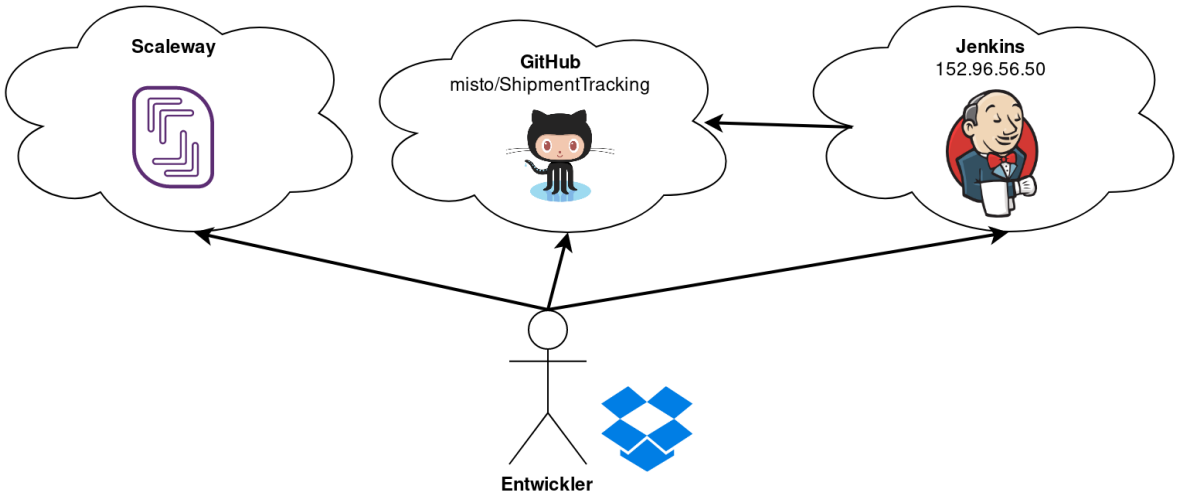

Abbildung 2: Infrastruktur Aufbau

IntelliJ IDEA IDE von JetBrains welche Scala und React unterstützt, was primär für die Entwicklung verwendet wird.

Dropbox Auf Dropbox befindet sich unsere Dokumentenablage. Dropbox bietet uns Versionskontrolle und eine Änderungshistory.

Office Suite Wir verwenden Office für unsere Dokumentationen, da kein Projektmitglied Erfahrungen mit LaTEX hat und dies somit viel Overhead bedeuten würde.

Astah Für die Erstellung von Diagrammen verwenden wir die Software Astah, welche bei den Projektmitgliedern bereits bestens bekannt ist.

Axure RP Axure RP erlaubt es uns auf einfache Weise nützliche clickable Wireframes zu erstellen. Es wird anderen Tools bevorzugt, da es bereits in vorherigen Projekten eingesetzt wurde und deshalb bereits bekannt ist.

Git & GitHub Git dient uns als Versionskontrollsystem. Das Repository für den CI Server befindet sich auf GitHub, wo auch die Arbeitspakete/Issues verwaltet werden.

Waffle.io Waffle.io stellt die auf GitHub erfassten Issues auf einem übersichtlichen Board dar, was deren Verwaltung erleichtert und auch zu Besprechungszwecken verwendet werden kann.

Jenkins Der CI Server ermöglicht einen automatisierten Buildprozess bei jeder Änderung der Codebasis, inklusive Ausführung der automatisierten Testfälle.

Toggl Für die Zeiterfassung und Auswertung verwenden wir das online Tool Toggl.

6 Qualitätsmassnahmen

In diesem Kapitel wird definiert, durch welche Massnahmen und Tools die Qualität des Projektes sichergestellt wird.

6.1 Dokumentation

Die Dokumente werden auf einem freigegebenen Dropbox Ordner „SABA“ gespeichert. Dropbox bietet uns kostenfrei die Funktion veränderte oder gelöschte Dateien bis auf 30 Tage zurück wiederherzustellen.

Die Qualität wird dadurch erreicht, dass wir alle dieselben Vorlagen verwenden und somit einen einheitlichen Auftritt gegenüber dem Kunden haben.

Falls Änderungen an bereits bestehenden Dokumentationen getätigt werden müssen, ist das ganze Team zu informieren. So ist sichergestellt, dass alle immer nach dem aktuellen Stand der Dinge implementieren.

6.2 Entwicklung

Der Sourcecode wird auf einem privaten Repository auf GitHub zentral gespeichert.

GitHub Repository	**https://github.com/misto/ShipmentTracking**

6.2.1 Vorgehen

Unser Entwicklungsprozess sieht grob gegliedert folgendermassen aus:

1. Erstellen eines Branches pro Userstory (Branchname entspricht Nummer der Userstory)
2. Implementieren der Userstory, inkl. Unit- und Integrationstests
3. Branch mit neuester Master-Version updaten und Pull-Request erstellen
4. Review des Codes (inkl. Tests) durch das zugewiesene Teammitglied
5. Besprechung von allfälligen Diskrepanzen
6. Wenn nötig: Code korrigieren und zurück zu Schritt 4

6.2.2 Unit Testing

Unit-Tests müssen grundsätzlich für alle wichtigen Funktionen erstellt werden. Die Tests werden einerseits manuell von den Entwicklern vor einem Commit gestartet, da nur funktionierender, getesteter Code committet werden soll und andererseits mit jedem Build automatisch ausgeführt. Dadurch werden Fehler frühzeitig erkannt und können behoben werden.

6.2.3 Code Reviews

Jedes Issue wird in GitHub einem Teammitglied als Entwickler zugewiesen. Das jeweils andere Teammitglied übernimmt die Rolle des Reviewers und kontrolliert den Code bevor dieser in den Master Branch gemerged wird. Die gefundenen Diskrepanzen werden zwischen den betreffenden Teammitgliedern besprochen und allfällige Verbesserungen daraus definiert und umgesetzt.

6.3 Testen

6.3.1 Systemtest

Zum Schluss des Projektes wird das komplette System mit einer Reihe von Systemtests überprüft. Diese werden in einem Testprotokoll festgehalten.

6.3.2 Usability Test

Nach der Implementation des GUI und der wichtigsten Funktionen werden aussenstehende Personen zu einem Usability Test hinzugezogen. Diese sollen anhand von Testfällen die Software intuitiv bedienen können. Dabei werden Sie vom Projektteam beobachtet. So werden neue Erkenntnisse über die Usability der Software gewonnen. Der Test ist erfolgreich wenn alle Personen die Testfälle erfolgreich und ohne grossen Suchaufwand abschliessen.

6.4 Definition of Done

- Der Code ist
    - fertiggestellt
    - getestet
    - im Versionierungssystem eingespielt
- Das Code Review wurde durchgeführt und akzeptiert

Anhang C: Testprotokolle

C.1 Usability Test Protokoll – Giovanni

1 Angaben zur Durchführung

Usability Test Objekt:	Build #70
Tester:	Giovanni
Beobachter:	Pascal Schweizer
Datum:	24.05.18

2 Einführung

Herzlich Willkommen zu unserem Usability Test, oder für nicht Informatiker einfach Benutzerfreundlichkeitstest. Wir danken dir schon im Voraus für die aufgebrachte Zeit. Durch diesen Test wollen wir auf keinen Fall dein Können oder sogar dich persönlich testen – Du kannst absolut nichts falsch machen.
Wir wollen mit diesem Test unsere Software unter die Lupe nehmen und allfällige Schwierigkeiten in der Benutzerführung beziehungsweise dem Design aufdecken.

2.1 Testablauf

Der Test wird wie folgt ablaufen:

- Du erhältst eine kurze Einführung, worum es bei der Software geht und was für eine Rolle du bei diesem Test einnimmst.
- In jedem Test Case findest du eine Aufgabe, die du selbstständig in unserer Software lösen sollst.
- Löse bitte die Aufgaben der Reihe nach, da gewisse Test Cases aufeinander aufbauen.
- Während dem Lösen der Aufgaben bitten wir dich, immer alle Gedankengänge laut zu denken, damit unser Beobachter Notizen machen kann.

3 Test Cases

3.1 Test Case 1: Potenziell defekte Pakete finden

Aufgabe
Finde heraus, bei wie vielen Trackern bei der letzten Paketsendung (Shipment) etwas schief gelaufen ist, sprich mindestens ein Sensorwert ausserhalb der definierten Limits lag.
Notizen des Beobachters •

3.2 Test Case 2: Sensoren mit fehlerhaften Werten finden

Aufgabe
Wähle einen der Tracker aus Test Case 1 und untersuche bei welchen Sensoren Werte ausserhalb der Limits aufgezeichnet wurden.
Notizen des Beobachters • Nicht klar, dass die Werte in der Hover-Info auf der Übersichtskarte die aktuellen Werte sind. • Hat die rot markierten Werte als fehlerhafte Werte interpretiert.

3.3 Test Case 3: Genaue Fehlerwerte und Positionen finden

Aufgabe
Überprüfe beim in Test Case 2 gewählten Tracker folgendes: • Was sind die genauen Werte, die ausserhalb der Limits liegen • Wann wurden sie aufgezeichnet/gemessen • Wo ungefähr befand sich der Tracker/das Paket zu diesem Zeitpunkt
Notizen des Beobachters • Die Zoom-Funktion der Charts wurde nicht entdeckt, was das Ablesen der genauen Werte erschwert hat.

3.4 Test Case 4: Aktuelle Position von bestimmtem Tracker finden

Aufgabe
Finde heraus, wo sich der Tracker namens „Med Tracker“ momentan befindet und ob während seiner letzten Paketsendung alles in Ordnung war.
Notizen des Beobachters • Highlighting der Tracker auf der Übersichtskarte durch zufälliges Klicken entdeckt.

3.5 Test Case 5: Paketsendung (Shipment) starten

Aufgabe
Wähle einen Tracker, der momentan nicht unterwegs ist, und starte eine neue Paketsendung (Shipment).
Notizen des Beobachters • Hat die Start-/Stopp-Buttons auf der Übersichtskarte verwendet.

3.6 Test Case 6: Paketsendung – Infos finden und stoppen

Aufgabe
Wähle einen Tracker, der momentan unterwegs ist (nicht den, den du bei Test Case 4 losgeschickt hast), finde heraus, was verschickt wird und stoppe die Paketsendung.
Notizen des Beobachters •

3.7 Test Case 7: Tracker – Paketsendungsverlauf untersuchen

Aufgabe
Wähle einen Tracker und untersuche seinen Paketsendungsverlauf, sprich wann und was mit den Paketen, in denen dieser Tracker montiert war, verschickt wurde.
Notizen des Beobachters •

3.8 Test Case 8: Eigene Accountdaten bearbeiten

Aufgabe
Ändere deine Login E-Mail und setze ein neues Passwort.
Notizen des Beobachters •

3.9 Test Case 9: Andere Benutzer verwalten

Aufgabe
Tobe dich in der Benutzerverwaltung aus. Erstelle und bearbeite Accounts nach Belieben (du kannst nichts kaputt machen).
Notizen des Beobachters • Wollte die Checkboxen in der Liste anklicken, da diese nicht das Disabled-Styling haben, sondern nur deaktiviert sind.

C.2 Usability Test Protokoll – Brigitte

1 Angaben zur Durchführung

Usability Test Objekt:	Build #70
Tester:	Brigitte
Beobachter:	Pascal Schweizer
Datum:	24.05.18

2 Einführung

Herzlich Willkommen zu unserem Usability Test, oder für nicht Informatiker einfach Benutzerfreundlichkeitstest. Wir danken dir schon im Voraus für die aufgebrachte Zeit. Durch diesen Test wollen wir auf keinen Fall dein Können oder sogar dich persönlich testen – Du kannst absolut nichts falsch machen.
Wir wollen mit diesem Test unsere Software unter die Lupe nehmen und allfällige Schwierigkeiten in der Benutzerführung beziehungsweise dem Design aufdecken.

2.1 Testablauf

Der Test wird wie folgt ablaufen:

- Du erhältst eine kurze Einführung, worum es bei der Software geht und was für eine Rolle du bei diesem Test einnimmst.
- In jedem Test Case findest du eine Aufgabe, die du selbstständig in unserer Software lösen sollst.
- Löse bitte die Aufgaben der Reihe nach, da gewisse Test Cases aufeinander aufbauen.
- Während dem Lösen der Aufgaben bitten wir dich, immer alle Gedankengänge laut zu denken, damit unser Beobachter Notizen machen kann.

3 Test Cases

3.1 Test Case 1: Potenziell defekte Pakete finden

Aufgabe
Finde heraus, bei wie vielen Trackern bei der letzten Paketsendung (Shipment) etwas schief gelaufen ist, sprich mindestens ein Sensorwert ausserhalb der definierten Limits lag.
Notizen des Beobachters •

3.2 Test Case 2: Sensoren mit fehlerhaften Werten finden

Aufgabe
Wähle einen der Tracker aus Test Case 1 und untersuche bei welchen Sensoren Werte ausserhalb der Limits aufgezeichnet wurden.
Notizen des Beobachters • Hat angenommen, dass der rot markierte Wert ausserhalb der Limits liegt, obwohl dieser eigentlich in Ordnung ist.

3.3 Test Case 3: Genaue Fehlerwerte und Positionen finden

Aufgabe
Überprüfe beim in Test Case 2 gewählten Tracker folgendes: • Was sind die genauen Werte, die ausserhalb der Limits liegen • Wann wurden sie aufgezeichnet/gemessen • Wo ungefähr befand sich der Tracker/das Paket zu diesem Zeitpunkt
Notizen des Beobachters • Hat die Zoom-Funktion der Charts nicht entdeckt.

3.4 Test Case 4: Aktuelle Position von bestimmtem Tracker finden

Aufgabe
Finde heraus, wo sich der Tracker namens „Med Tracker“ momentan befindet und ob während seiner letzten Paketsendung alles in Ordnung war.
Notizen des Beobachters •

3.5 Test Case 5: Paketsendung (Shipment) starten

Aufgabe
Wähle einen Tracker, der momentan nicht unterwegs ist, und starte eine neue Paketsendung (Shipment).
Notizen des Beobachters • Hat die Start-/Stopp-Buttons auf der Übersichtskarte übersehen und diejenigen in der Detailansicht genutzt.

3.6 Test Case 6: Paketsendung – Infos finden und stoppen

Aufgabe
Wähle einen Tracker, der momentan unterwegs ist (nicht den, den du bei Test Case 4 losgeschickt hast), finde heraus, was verschickt wird und stoppe die Paketsendung.
Notizen des Beobachters •

3.7 Test Case 7: Tracker – Paketsendungsverlauf untersuchen

Aufgabe
Wähle einen Tracker und untersuche seinen Paketsendungsverlauf, sprich wann und was mit den Paketen, in denen dieser Tracker montiert war, verschickt wurde.
Notizen des Beobachters •

3.8 Test Case 8: Eigene Accountdaten bearbeiten

Aufgabe
Ändere deine Login E-Mail und setze ein neues Passwort.
Notizen des Beobachters •

3.9 Test Case 9: Andere Benutzer verwalten

Aufgabe
Tobe dich in der Benutzerverwaltung aus. Erstelle und bearbeite Accounts nach Belieben (du kannst nichts kaputt machen).
Notizen des Beobachters •

C.3 Systemtest Protokoll

1 Angaben zur Durchführung

Systemtest Objekt: Build #100
Testumgebung: Windows 10, Google Chrome Version 67.0
Test-Typ: White-Box
Datum: 31.05.18

2 Testdaten

- 3 Tracker mit je 8 Sensoren
    - Sensoren
        - Beschleunigung X-, Y-, Z-Achse
        - Temperatur
        - Feuchtigkeit
        - Chip Temperatur (Admin Only)
        - Batteriestand (Admin Only)
        - Verbindungsqualität (Admin Only)
    - Messwerte
        - Von den beiden Test-Trackern des HiveWatch-Systems geladen
    - Tracker 1
        - 3 Shipments, alle abgeschlossen
        - Alle Shipments mit Grenzwertüberschreitungen
    - Tracker 2
        - 2 Shipments, eines noch unterwegs
        - Keine Grenzwerte festgelegt → keine Überschreitungen
    - Tracker 3
        - Keine Shipments
        - Keine aktuelle Position
- 2 Benutzer
    - Benutzer 1
        - Normaler Benutzer
        - Besitzt Tracker 1
    - Benutzer 2
        - Administrator
        - Besitzt Tracker 2

3 Test Cases

Use Case	Test Case	Erwartetes Ergebnis	Bestanden?	Bemerkungen
UC 01: Übersichtskarte	TC 01: Laden der Übersichtskarte als Benutzer 1	1 Positions-Marker (rot), 1 Tracker in der Liste	✓	
	TC 02: Laden der Übersichtskarte als Benutzer 2	2 Positions-Marker (1x grün, 1x rot), 3 Tracker in der Liste	✓	
	TC 03: Aktivieren und Deaktivieren des grünen Marker-Filters	Grüner Marker wird versteckt und wieder angezeigt	✓	
	TC 04: Aktivieren und Deaktivieren des roten Marker-Filters	Roter Marker wird versteckt und wieder angezeigt	✓	
	TC 05: Ausblenden und Anzeigen von Tracker 1 über seine Checkbox in der Liste	Marker von Tracker 1 wird versteckt und wieder angezeigt	✓	
	TC 06: Ausblenden und Anzeigen von allen Trackern über die Listen-Checkbox	Alle Marker werden versteckt und wieder angezeigt, Status aller Checkboxen ändert zu unchecked und wieder zu checked	✓	
UC 02: Detailansicht	TC 07: Laden der Detailansicht für Tracker 1 als Benutzer 1	Letztes Shipment ist ausgewählt, Tracker-Infos, Karte mit Track & 5 Diagramme mit Messwerten werden angezeigt, Grenzwertüberschreitungen sind im Chart rot gekennzeichnet	✓	
	TC 08: Laden der Detailansicht für Tracker 2 als Benutzer 1	Fehlermeldung erscheint: Tracker konnte nicht geladen werden, leere Detailansicht wird angezeigt	✓	
	TC 09: Laden der Detailansicht für Tracker 2 als Benutzer 2	Letztes Shipment ist ausgewählt, Tracker-Infos, Karte mit Track & 5 Diagramme mit Messwerten werden angezeigt	✓	
	TC 10: Laden der Detailansicht für Tracker 3 als Benutzer 2	Shipment-Selector zeigt an, dass keine Shipments vorhanden sind, Tracker-Infos, leere Karte und leere Charts werden angezeigt	✓	
	TC 11: Altes Shipment von Tracker	Start- und Stopp-Button werden deaktiviert,	✓	

	2 auswählen	Karte und Charts werden aktualisiert		
	TC 12: Bewegen des Cursors über eines der Charts	Markierung auf der Karte und allen Charts wird angezeigt und bewegt	✓	
	TC 13: Hereinzoomen in eines der Charts	Alle Charts werden auf denselben Zeitbereich gezoomt, ein Button zum Zurücksetzen des Zooms wird angezeigt	✓	
	TC 14: Zoom eines Charts zurücksetzen	Zoom von allen Charts wird zurückgesetzt	✓	
UC 03: Einstellungsgruppen verwalten	Use Case nicht implementiert	-	-	
UC 04: Standard Einstellungsgruppe konfigurieren	TC 15: Grenzwerte der Gruppe verändern und neues Shipment starten	Neues Shipment übernimmt neue Grenzwerte	✓	
UC 05: Benachrichtigung	Use Case nicht implementiert	-	-	
UC 06: Alarmhistory	TC 16: Altes Shipment von Tracker 1 auswählen	Grenzwertüberschreitungen sind im Chart rot gekennzeichnet		
UC 07: Start/Stopp Paketsendung	TC 17: Shipment für Tracker 1 in Übersichtskarte starten	Start-Button wird deaktiviert, Stopp-Button wird aktiviert, Positions-Marker ändert auf grün	✓	
	TC 18: Shipment für Tracker 2 in Übersichtskarte stoppen	Start-Button wird aktiviert, Stopp-Button wird deaktiviert	✓	
	TC 19: Shipment für Tracker 1 in Detailansicht starten	Start-Button wird deaktiviert, Stopp-Button wird aktiviert, neues Shipment wird ausgewählt	✓	
	TC 20: Shipment für Tracker 2 in Detailansicht stoppen	Start-Button wird aktiviert, Stopp-Button wird deaktiviert	✓	
UC 08: Paketsendungen bearbeiten	Use Case nicht implementiert	-	-	
UC 09: CSV Export	Use Case nicht implementiert	-	-	
UC 10: Heatmap	Use Case nicht implementiert	-	-	
UC 11: Tracker um-/benennen	TC 21: Tracker 1 umbenennen	Der neue Name wird auf der Übersichtskarte, Detailansicht und Tracker Liste angezeigt	✓	

UC 12: Tracker verwalten	TC 22: Laden der Tracker Liste als Benutzer 1	Umleitung auf Übersichtskarte	✓	
	TC 23: Laden der Tracker Liste als Benutzer 2	Liste mit 3 Trackern wird angezeigt	✓	
	TC 24: Einzelnen neuen Tracker erfassen	Neuer Tracker erscheint in der Tracker Liste für Admins	✓	
	TC 25: Mehrere neue Tracker gleichzeitig erfassen	Alle neuen Tracker erscheinen in der Tracker Liste für Admins	✓	
	TC 26: Tracker 3 Benutzer 1 zuweisen	Benutzer 1 wird als Besitzer in der Tracker Liste angezeigt, Benutzer 1 sieht den Tracker in der Liste auf der Übersichtskarte	✓	
UC 13: Benutzer verwalten	TC 27: Neuen Benutzer (aktiviert) erstellen	Neuer Benutzer erscheint in der Benutzer Liste, Neuer Benutzer kann sich anmelden	✓	
	TC 28: Benutzer 1 deaktivieren	Fehlermeldung erscheint bei der Anmeldung von Benutzer 1: Anmeldung nicht möglich	✓	
	TC 29: Benutzer 1 zu Admin ändern	Benutzer 1 erhält Zugriff auf alle Tracker	✓	
	TC 30: E-Mail und Passwort von Benutzer 1 ändern	Benutzer 1 kann sich nur mit den neuen Daten anmelden	✓	
	TC 31: Benutzer 2 zu normalem Benutzer ändern	Benutzer 2 verliert sofort Zugriff auf alle Admin-Tools und die anderen Tracker	✓	
UC 14: Messdaten speichern	TC 32: Neue Messdaten für Tracker 2 an Receiver senden	Neue Messdaten werden in der Detailansicht von Tracker 2 im aktuellen Shipment angezeigt	✓	
	TC 33: Messdaten für nicht registrierten Tracker an Receiver senden	Receiver meldet "400 Bad Request" zurück, keine Daten werden erfasst	✓	