

Kort Reloaded – A Gamified App for Collecting OpenStreetMap Data

Bachelorarbeit

Abteilung Informatik
HSR Hochschule für Technik Rapperswil

Frühjahrssemester 2016

Autoren:	Marino Melchiori Dominic Mülhaupt
Betreuer:	Prof. Stefan Keller
Projektpartner:	Jürg Hunziker Stefan Oderbolz Liip AG
Experte:	Claude Eisenhut
Gegenleser:	Prof. Beat Stettler

Impressum

Autoren:	Marino Melchiori (mmelchio@hsr.ch) Dominic Mülhaupt (dmuelhau@hsr.ch)
Dokument erstellt:	10.03.2016
Letzte Aktualisierung:	28.06.2016

Dieses Dokument wurde mit L^AT_EX erstellt.

Abstract

OpenStreetMap ist ein freies Projekt, das von Benutzern aus der ganzen Welt unterstützt wird. Wege, Gebäude und viele andere geografische Daten werden weltweit in einer Datenbank erfasst und gepflegt. *OpenStreetMap* kann von jedem bearbeitet werden, besteht aus einer grossen Community und setzt auf lokales Wissen der «Mapper». Aus diesem Grund ist es nicht ausgeschlossen, dass fehlerhafte oder unvollständige Daten enthalten sind. Zur Korrektur der Daten gibt es viele verschiedene Tools, die von Experten genutzt werden können. Um eine breitere Masse anzusprechen, entstand 2012 die Web-App «KORT» im Rahmen einer Bachelorarbeit. Mit KORT kann der Benutzer Aufträge lösen, die zur Verbesserung der Daten in *OpenStreetMap* beitragen. Auf einer Kartenansicht werden die Aufträge, welche sich im Umfeld des Benutzers befinden, dargestellt. Für das Eintragen einer Lösung wird man mit Punkten (sogenannten «Koins») belohnt und kann so in der Rangliste aufsteigen. Da sich HTML5 weiterentwickelt hat, muss die Web-App abgelöst werden.

In dieser Arbeit wurde KORT als native Mobile App für *Android* und *iOS* komplett neu entwickelt. Das bestehende Backend wurde dabei praktisch unverändert weitergenutzt. Die App basiert auf dem *React Native* Framework von *Facebook*. *React Native* ist eine moderne, noch junge Technologie, welche es ermöglicht, mit *JavaScript* Apps für *Android* und *iOS* zu entwickeln. Dabei wird *JavaScript*-Code in Komponenten der jeweiligen Smartphone-Plattform übersetzt, was dem Benutzer die Erfahrung einer nativen App bietet.

Es konnten wichtige Erfahrungen mit *React Native* gesammelt werden. Dabei sind eine *Android*- und eine *iOS*-App entstanden. Die *Android*-App befindet sich noch in der Beta-Phase und wird nach Abschluss dieser Arbeit im *Google Play Store* erwartet. Die *iOS*-App ist noch in der Testphase und wird später im *Apple App Store* veröffentlicht. Weitere Informationen: www.kort.ch.

Dank

Für die Betreuung während des ganzen Projektes möchten wir uns besonders bei Prof. Stefan Keller bedanken. Bei Fragen zu *OpenStreetMap* konnte er uns während der gesamten Laufzeit der Arbeit unterstützen und er gab uns bei Problemen immer Inspirationen für eine Lösung.

Ein Dank geht auch an Jürg Hunziker und Stefan Oderbolz für die Unterstützung während des ganzen Projektes und für ihre \LaTeX -Vorlage. Sie gaben uns Tipps zum Vorgehen und haben für uns wichtige Backend-Änderungen vorgenommen. Prof. Stefan Keller, Jürg Hunziker und Stefan Oderbolz liessen uns freien Spielraum für die Suche nach einer optimalen Lösung.

Ebenfalls bedanken wir uns bei Robert Vogt und Mirco Strässle für ihre Tipps zu *React* und *React Native*.

Ein besonderer Dank geht auch an Marco Syfrig und Mario Meili für das Gegenlesen dieses Dokuments.

Aufgabenstellung

Der KORT-Client soll neu als native *Android* App zur Verfügung stehen. Das bedingt ein kompletter Rewrite des aktuellen *JavaScript*-Codes (aktuell *Sencha Touch*) mit dem [Framework React Native](#). Es soll ein Erfahrungsbericht zu *React Native* erstellt werden.

Ziele:

- Gleiche Funktionalität mit neuem Framework, damit die mobile App mit den neusten Technologien arbeitet und künftig besser wartbar ist
- Neue Erkenntnisse zu einem aktuellen Framework zur Realisierung von native mobile Apps *React Native*
- Getestete Software-Entwicklungsumgebung

Deliverables

Mindestens...

- Neuer KORT-Client als native *Android*-App
- Ablösung des Validierungsmechanismus
- Erfahrungsbericht mit Hinweisen zu Tutorials zu *React Native*
- Erweiterte Software-Entwicklungsumgebung
- Die vom Studiengang geforderten Lieferobjekte: Dokumentation, Management Summary, Abstract, Poster, Präsentation mit Stellwand, Zwischenpräsentation

Erweitert...

- KORT-Client als native *iOS*-App
- neue Funktion: Promotions anzeigen
- Kurzvideo

Die definitive Aufgabenstellung, Lieferobjekte und das Vorgehen werden am Kickoff (erste Semesterwoche) zusammen mit dem Industriepartner festgelegt. Die gemeinsam besprochene Aufgabenstellung wird ca. zwei Wochen nach Semesterbeginn aktualisiert.

Vorgaben/Rahmenbedingungen

- Die ursprüngliche KORT-App ist clientseitig in *HTML5* und *JavaScript* geschrieben.
- Serverseitig ist u.a. *PHP* und *PostgreSQL* mit der *PostGIS*-Erweiterung vorhanden.
- Als Technologien stehen *React* und *React Native* im Vordergrund.
- Es wird genügend Zeit für die Einarbeitung in die Themengebiete einberechnet.
- Die Software soll Open Source sein.
- Die SW-Engineering-Methode und Meilensteine werden mit dem Betreuer vereinbart.
- Sourcecode und Software-Dokumentation sind Englisch (inkl. Installation, keine Benutzerdokumentation, höchstens eine Online-Kurzhilfe).
- Die Software-Benutzerschnittstelle ist mind. Deutsch und Englisch.
- Die Projekt-Dokumentation und -Präsentation sind auf Deutsch.
- Der Source Code, die Code-Kommentare und die Versionsverwaltung sind in Englisch.
- Die Nutzungsrechte an der Arbeit bleiben bei den Autoren und gehen auch an die *HSR* und den Betreuer über. Die Softwarelizenz ist „MIT“.
- Ein Video gemäss den Vorgaben des Studiengangs (kann ggf. nach dem Dokumentations-Abgabetermin abgegeben werden).
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs Informatik bzw. der *HSR*.

Inhalt der Dokumentation

- Die fertige Arbeit muss folgende Inhalte haben:
 1. Abstract, Aufgabenstellung
 2. Technischer Bericht
 3. Projektdokumentation
 4. Anhänge (Literaturverzeichnis, Glossar, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.

- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Poster) gemäss www.hsr.ch und Absprache mit dem Betreuer.

Form der Dokumentation

Bericht, Dokumente und Quellen der erstellten Software gemäss Vorgaben des Studiengangs Informatik der *HSR* sowie Absprache mit dem Betreuer.

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Arbeit (6 Aspekte) des Studiengangs Informatik der *HSR* jedoch mit besonderem Gewicht auf moderne Softwareentwicklung (Tests, Continuous Integration, einfach installierbar, funktionsfähig).

Beteiligte

Diplomanden

Marino Melchiori und Dominic Mülhaupt

Industriepartner

Jürg Hunziker und Stefan Oderbolz, Liip AG Zürich

Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller

Inhaltsverzeichnis

I. Technischer Bericht	1
1. Einführung	2
1.1. Problemstellung	2
1.2. Ziele	2
1.3. Rahmenbedingungen	3
1.4. Vorgehen	3
2. Stand der Technik	5
2.1. Bestehende Lösungsansätze und Normen	5
2.2. React Native	5
2.3. OpenStreetMap OAuth	6
2.4. Kort Schnittstelle	6
3. Evaluation	7
3.1. Architektur	7
3.2. App Navigation	8
3.3. Kartendarstellung	9
3.4. OAuth Implementation	12
3.5. Internationalisierung	13
4. Resultate	14
4.1. Zielerreichung	14
4.2. Ausblick	15
4.3. Persönliche Berichte	16
II. Projektdokumentation	17
5. Anforderungsspezifikation	18
5.1. Anforderungen an die Arbeit	18
5.1.1. Muss	18
5.1.2. Soll	18
5.1.3. Kann	18
5.2. Use Cases	19
6. Technologien	20
6.1. React	20

6.2. React Native	20
6.2.1. Layout	21
6.2.2. Technische Details	21
6.2.3. Setup	22
6.2.4. Cross Platform	22
6.2.5. Community	22
6.2.6. Erfahrungen	23
7. Design	25
7.1. Datenmodell	25
7.2. Architektur	26
7.2.1. Stores	26
7.2.2. Views	27
7.2.3. Actions	27
7.2.4. Dispatcher	27
7.3. Sequenzdiagramme	29
7.3.1. Starten der App	29
7.3.2. Mission lösen	30
8. Entwicklungsumgebung	31
8.1. IDE	31
8.2. Continuous Integration	31
8.3. Projektmanagement-Tool	31
8.4. Testing	32
8.5. Code-Richtlinien	32
9. Implementation	33
9.1. Kort Backend	33
9.2. View Components	33
9.3. Libraries	34
9.3.1. Navigation	35
9.3.2. Karte	35
9.3.3. OAuth	35
10. Weiterentwicklung	37
10.1. Vorgehen	37
10.2. Weiterführende Arbeiten	39
11. Installation	41
11.1. Installation mit Source Code	41
11.2. Installation mit APK-Datei	41

III. Projektmanagement	43
12. Projektmanagement	44
12.1. Team	44
12.2. Risikomanagement	45
12.2.1. Risikoanalyse	45
12.3. Projektplan	50
12.4. Sprints	51
12.4.1. Sprint 1	51
12.4.2. Sprint 2	51
12.4.3. Sprint 3	51
12.4.4. Sprint 4	51
12.4.5. Sprint 5	52
12.4.6. Sprint 6	52
12.5. Meilensteine	52
12.5.1. MS1: Kickoff	52
12.5.2. MS2: Ende Elaboration	52
12.5.3. MS3: Evaluation der Komponenten	53
12.5.4. MS4: Zwischenpräsentation	55
12.5.5. MS5: Basis-Komponenten umgesetzt	55
12.5.6. MS6: Beta-Release mit Grundfunktionalität	57
12.5.7. MS7: Schlussabgabe	57
12.5.8. MS8: Schlusspräsentation	58
12.5.9. MS9: Release für App Store	58
13. Projektmonitoring	59
13.1. Zeitanalyse	59
13.1.1. Soll-Ist-Zeitvergleich	60
13.2. Code-Statistik	61
 IV. Anhänge	 63
Glossar	64
Literaturverzeichnis	68
Abbildungsverzeichnis	71
Tabellenverzeichnis	72
Inhalt der CD	73

Teil I.

Technischer Bericht

1. Einführung

1.1. Problemstellung

OpenStreetMap (*OSM*) beinhaltet eine sehr grosse Menge an Geodaten, welche frei zugänglich sind. Für die Pflege dieser Daten ist es daher naheliegend, auf unterstützende Software zurückzugreifen. Zu diesem Zweck gibt es eine Reihe von Applikationen, welche sich grob in zwei Kategorien einteilen lassen: Editoren und Tools zur Qualitätssicherung.

Mit den Editoren lässt sich die *OSM*-Karte direkt oder indirekt verändern und ergänzen. Die Qualitätssicherungstools zielen darauf ab, fehlende oder falsche Daten aufzuspüren. Diese werden dann entweder automatisch korrigiert oder übersichtlich dargestellt, um eine manuelle Korrektur zu ermöglichen.

Einige Tools wie *KeepRight*¹ oder *Osmose*² berechnen aus den Karten-Rohdaten die vorhandenen Fehler. Dazu werden einige Heuristiken verwendet oder einfache Plausibilitätsüberprüfung durchgeführt. Typische Fehler aus diesen Quellen sind **POIs** ohne Namen oder Strassen ohne definierte Geschwindigkeitslimiten.[21]

Zur Behebung dieser Fehler ist die **Web-App** KORT³ in Form einer Bachelorarbeit von Jürg Hunziker und Stefan Oderbolz im Herbstsemester 2012/13 entwickelt worden.[21]

KORT wurde mit dem *Sencha Touch 2 Framework* entwickelt. Da sich HTML5 weiterentwickelt hat, funktioniert die Implementation auf neuen Browsern nicht mehr sinngemäss. Aus diesem Grund entstand die Idee, die KORT-Web-App durch einen native Client zu ersetzen. Dazu bot sich *React Native* an. Diese ganz neue Technologie ermöglicht es native *iOS* und – seit Oktober 2015 – auch *Android* Apps mit *JavaScript* zu erstellen. Dadurch, dass *React Native* noch in den Kinderschuhen steckt und die Entwickler, Marino Melchiori und Dominic Mülhaupt, noch keine Erfahrung mit *JavaScript* haben, besteht hier ein gewisses Risiko – insbesondere in der Planung und der technischen Analyse.

1.2. Ziele

Ziel ist es, eine *Android* App mit der gleichen Funktionalität wie in der derzeitigen Web-App zu erstellen. Zusätzlich gibt es optionale Ziele, wie das Erstellen der *iOS*-App.

¹<http://keepright.ipax.at/>

²<http://osmose.openstreetmap.fr/map/>

³<http://play.kort.ch/>

Es entstand die Idee, den Validationsmechanismus von KORT abzulösen. Bis jetzt wurden viel mehr Missionen gelöst als Validierungen durchgeführt wurden.⁴ Das führte dazu, dass nur sehr wenige Änderungen überhaupt in der *OpenStreetMap*-Datenbank eingefügt wurden – eine gelöste Mission muss zuerst von drei Benutzern als korrekt markiert werden. Die *OSM*-Community forderte aber, dass die Antworten geprüft werden müssen. Da es scheinbar einen Trend gibt, dass Benutzer lieber Missionen lösen als zu validieren, werden dem Benutzer Validierungen nun einfach als normale Aufträge angezeigt. Sobald die abgeschickte Antwort mit der zu validierenden Antwort übereinstimmt, wird im Hintergrund eine positive Bewertung versendet.

Weitere Ziele sind:

- Das Erstellen einer *Android* App mit *React-Native* – gleiche Funktionalität mit neuem Framework, damit die mobile App mit den neusten Technologien arbeitet und künftig besser wartbar ist.
- Als Basis sollen Daten und Webdienste des *OSM*-Projekts verwendet werden.
- Es sollen neue Erkenntnisse zum aktuellen Framework (*React-Native*), zur Realisierung von native mobile Apps, gesammelt werden.
- Ein Erfahrungsbericht zu *React-Native* soll erstellt werden.
- Die Internationalisierung soll einfach umgesetzt sein.

1.3. Rahmenbedingungen

Die *React Native*-App baut auf das bestehende KORT-Backend auf. Folglich geht es bei der Entwicklung nur um das Frontend – also um den native Client. Anpassungen am Backend lagen nicht im [Aufgabenbereich](#) der Entwickler. Einen Überblick über die Architektur und das Gesamtsystem von KORT, welches bereits bestand, gibt das Kapitel 8 der Bachelorarbeit von Jürg Hunziker und Stefan Oderbolz.[21]

- Es gelten die Rahmenbedingungen, Vorgaben und Termine der HSR.
- Die Projektabwicklung orientiert sich an einer iterativen, agilen Vorgehensweise. Als Vorlage dient dabei Scrum.

1.4. Vorgehen

- Einarbeiten in *JavaScript* und *React Native* sowie den damit verbundenen Technologien.
- Einarbeiten in den Code und die Infrastruktur von KORT.

⁴rund 50 000 gelöste Missionen und 15 000 durchgeführte Validierungen (Stand 17. Juni 2016)

- Iteratives Entwickeln des Prototyps.
- Dokumentation abschliessen.

[Ziele](#) und [Resultate](#) werden im technischen Bericht erläutert. Die [Risikoanalyse](#), die [Meilensteine](#) und das [Team](#) befinden sich im Kapitel Projektmanagement. Der Source Code ist auf *GitHub*⁵ frei zugänglich.

⁵<https://github.com/kort/kort-reloaded>

2. Stand der Technik

Es gibt eine grosse Anzahl an Projekten mit dem Ziel, *OSM* zu verbessern. Sowohl Editoren für erfahrene Benutzer, als auch Tools, die auf das Finden und Beheben von Fehlern spezialisiert sind, werden angeboten. Zum Beispiel gibt es JOSM¹ (*Java OSM Editor*) als Desktop Client für *Windows*, *Mac OS X* und *Linux*. Sogar Geometrieobjekte sind damit editierbar. Daneben gibt es auch Dienste, die Fehlerdaten sammeln und öffentlich anbieten. Ein bekanntes Beispiel ist *KeepRight*². Es werden über 50 Fehlertypen angeboten. Mit diesen Werkzeugen können Daten korrigiert werden. Weitere Editoren sind auf dem *OSM-Wiki*³ zu finden.

Die Zielgruppe dieser Werkzeuge sind Benutzer mit dem Interesse, die *OSM*-Daten zu verbessern. Um diesen Prozess effizienter zu gestalten, würde sich ein [Crowdsourcing](#)-Ansatz anbieten. Dafür muss die Zielgruppe aber erweitert werden. Ein sehr gutes Beispiel dafür ist *MapRoulette*⁴. Dem Benutzer werden Challenges präsentiert, die zum Beispiel ein Satellitenbild von einem Objekt anzeigen, das nach der Inspektion vom System einem Fussballfeld ähnlich sieht. Nun muss der Benutzer entscheiden, ob es sich tatsächlich um ein Fussballfeld handelt, oder nicht. [Gamification](#) anhand von Challenges bietet sich also sehr gut an. Die Spielelemente halten den Spieler motiviert und binden ihn an das Spiel. Einige Projekte⁵, die durch Gamification an *OSM*-Verbesserungen beitragen, sind bereits entstanden.[21]

2.1. Bestehende Lösungsansätze und Normen

Da wir die KORT-[Web-App](#) neu schreiben und es sich um eine Fortsetzungsarbeit handelt, übernehmen wir die bereits dort evaluierten Konzepte.

2.2. React Native

Aktuell, am 17.06.2016, befindet sich *React Native* in der Version 0.28. Dieses Projekt startete mit *React Native* 0.19 (veröffentlicht am 29. Januar 2016). *React Native* ist ganz neu und noch in der Entwicklungsphase. Momentan erscheint immer noch alle zwei Wochen ein neues Release mit Änderungen, die teilweise sehr nützlich und wichtig sind. Es ist aber auch schon

¹<https://josm.openstreetmap.de/>

²<http://www.keepright.at/>

³http://wiki.openstreetmap.org/wiki/Editors#Choice_of_editors/

⁴<http://wiki.openstreetmap.org/wiki/MapRoulette/>

⁵http://wiki.openstreetmap.org/wiki/Games#Gamification_of_map_contributions

vorgekommen, dass Updates ausgelassen werden mussten, da die App nicht mehr lauffähig war. Build-Probleme sind des Öfteren aufgetreten.

Die offizielle Dokumentation⁶ ist immer noch spärlich und Best Practices gibt es in vielen Bereichen keine.

2.3. OpenStreetMap OAuth

Im Gegensatz zu den anderen Social-Login-Varianten (*Google* und *Facebook*), die OAuth 2.0 anbieten, unterstützt *OSM* noch OAuth 1.0a⁷.

2.4. Kort Schnittstelle

Die KORT-[Web-App](#) nutzte eine Cookie-based Authentifizierung.^[21] Eine Session-ID wird auf der Seite des Clients in einem Cookie gespeichert. Bei jedem Request sendet der Client-Browser das Cookie an den Server und wird so wiedererkannt. Der Server geht dann davon aus, dass es sich beim Client um den Inhaber der Session-ID handelt. KORT als native App kann aber keine Session zu einem Server aufbauen. Das KORT-Backend wurde von den Projektpartnern, Stefan Oderbolz und Jürg Hunziker, dann so angepasst, dass eine Token-basierte Authentifizierung unterstützt wird. Vorerst wurde nur eine Unterstützung für eine Login-Variante über *Google* umgesetzt.

⁶<https://facebook.github.io/react-native/docs/getting-started.html>

⁷<http://wiki.openstreetmap.org/wiki/OAuth>

3. Evaluation

An einem *React Native* Meetup zu Beginn des Projekts konnten erste Entscheidungen zur Software-Entwicklungsumgebung geklärt werden. Es wurden auch Lösungskonzepte für die Darstellung der Karte besprochen.

Unter den vielen *JavaScript*-Editoren haben wir uns für *Atom* entschieden. *Facebook* hat speziell für *React*, *React Native* das *Atom*-Package *Nuclide*¹ veröffentlicht. *Atom* ist Open Source und bietet viele weitere Community-Packages.

Weitere Hinweise zur Entwicklungsumgebung und den verwendeten Werkzeugen sind im Kapitel [Projektmanagement](#) beschrieben.

3.1. Architektur

Bei der Evaluation, welches Architektur-Pattern wir anwenden würden, konnten wir uns auf [Frontend](#)-Architekturen beschränken. Dabei kamen für uns zwei Patterns in Frage: Traditionelles [MVC](#) und [Flux](#)².

¹<https://nuclide.io/>

²<https://facebook.github.io/flux/>

Variante A: Flux	
Vorteile	Nachteile
Unidirektionaler Datenfluss	Grösserer Aufwand für erstmalige Einrichtung
Kaum Kopplung – Anpassungen in späteren Arbeiten relativ problemlos möglich	Viele Callbacks nötig
Abstraktion des Applikationszustands passend für unseren Anwendungsfall	
Nutzt wichtige Konzepte von <i>React</i>	
Variante B: MVC	
Vorteile	Nachteile
Pattern ist bekannt, keine Einarbeitung nötig	

Tabelle 3.1.: Bewertung Komponente für die Navigation

Die Architektur-Entscheidung fiel zu Beginn sehr schwer, da vorgängig noch nicht genügend Erfahrungen im Umgang mit *React* gesammelt werden konnten. Entsprechend unserem [Projektplan](#) und der Grundidee, erst Erfahrungen mit für uns neuen Technologien (*JavaScript*, *React* und *React Native*) zu sammeln, haben wir uns in einem ersten Ansatz dazu entschlossen, die ersten Schritte – das Darstellen der Missionen auf der Karte – mit dem MVC-Pattern umzusetzen. Welche Auswirkungen der Einsatz von *Flux* hätte, war zu Beginn zu wenig voraussehbar.

Als die Wahl der Architektur im Zusammenhang mit [Meilenstein 5](#) neu evaluiert wurde, konnten wir leichte Vorteile in der *Flux*- gegenüber einer MVC-Architektur sehen. Einen guten Einblick in die Thematik verschafft der Artikel *Why we are doing MVC and FLUX wrong*^[1]. Wir haben uns letztlich aufgrund der oben aufgelisteten Vorteile auf *Flux* festgelegt.

3.2. App Navigation

Um die Navigation innerhalb der App zu implementieren, gab es zwei Möglichkeiten. Die erste Variante war die Umsetzung mit der von *React Native* zur Verfügung gestellten Navigator-Komponente. Damit müsste für die geplante Tab-Ansicht eine weitere [Library](#) evaluiert werden.

Als weitere Variante für die Navigation gab es die Library [react-native-router-flux](#)³. Diese Komponente basiert auf dem Navigator von *React Native* und unterstützt durch die bereits integrierte Tab-Library eine Tab-Ansicht.

³<https://github.com/aksonov/react-native-router-flux/>

Variante A: <i>React Native</i> Navigator-Komponente	
Vorteile	Nachteile
Es gibt eine Unterstützung für <i>Android</i> und <i>iOS</i> .	Die Dokumentation ist sehr knapp gehalten.
Flexibel und für einfache Use Cases gedacht.	Die Tab-Ansicht wird nicht unterstützt.
Variante B: <i>React-Native-Router-Flux</i> ⁴	
Vorteile	Nachteile
Alle Views (Scenes) sind an einem Ort deklariert. Es müssen keine Navigator-Objekte herumgereicht werden.	Stellt eine weitere Abhängigkeit an ein externes Projekt dar.
Leicht erweiterbar und wartbar. Integrierte Tab-Navigation.	Der aktuelle Navigationszustand ist nicht genau definiert.
Ein Wechsel der View ist mit einem Funktionsaufruf von überall aus möglich. Daten lassen sich dabei einfach als Parameter weitergeben.	Die Hintergrundabläufe und der Lebenszyklus sind nicht erkennbar.
Es gibt eine Unterstützung für <i>Android</i> und <i>iOS</i> .	Die Dokumentation ist sehr knapp gefasst.

Tabelle 3.2.: Bewertung Navigations-Komponente

Da *React Native* standardmässig keine Tab-Navigation anbietet, wurde *React-Native-Router-Flux* als Navigations-Variante evaluiert. Der erste Prototyp mit dieser [Library](#) erfüllte die Anforderungen. Alle Views sind an einem Ort im Code festgelegt und es lassen sich bequem weitere hinzufügen.

Im Verlaufe des Projektes sind dann aber vermehrt Fehler aufgetreten. Diese entstanden zum einen durch den Einsatz von *React Natives* experimentellen Navigator⁵, waren aber auch in der Library selbst enthalten. Im Nachhinein wäre es wahrscheinlich sinnvoller gewesen, die Navigator-Komponente zu verwenden.

3.3. Kartendarstellung

Für die Darstellung der Karte mit *React Native* sind die Varianten A bis E ausfindig gemacht worden. Diese Punkte beinhalten Libraries oder Ideen zur Umsetzung einer Kartenansicht für die App.

⁵<http://www.reactnative.com/first-look-at-react-native-navigation-experimental/>

Variante A: React Native Map Komponente

Diese Variante wird standardmässig von *React Native* als *MapView*-Komponente zur Verfügung gestellt. Die Möglichkeiten sind eingeschränkt, es lassen sich auf *Android* zum Beispiel nur normale Pins als Marker einsetzen.[10]

Variante B: Extended React Native Map Komponente

Die Extended React Native Map Komponente⁶ wird von *Facebook* anstelle der Standard-*MapView*-Komponente empfohlen.

Variante C: Mapbox GL Library

Mapbox bietet mit dieser experimentellen *React Native*-Komponente (*Mapbox GL Library*⁷) eine weitere Lösung für *iOS* und *Android*.

Variante D: Portierung von Leaflet nach React

Für *React* gibt es eine Map-Komponente namens *React-Leaflet*⁸. Diese liesse sich für *React Native* portieren. Schon in der KORT-*Web-App* wurde die *Leaflet*⁹-Library verwendet.

Variante E: Raster-Kacheln selbst darstellen

Die letzte mögliche Variante war, dass wir die benötigten Raster-Kacheln, die der Benutzer braucht, mit einer eigenen Implementation entsprechend laden und anzeigen.

⁶<https://github.com/lelandrichardson/react-native-maps>

⁷<https://libraries.io/npm/react-native-mapbox-gl>

⁸<https://github.com/PaulLeCam/react-leaflet>

⁹<http://leafletjs.com/t>

Variante A: <i>React Native</i> Map Komponente	
Vorteile	Nachteile
Komponente von <i>Facebook – React Native</i> .	Pattern Fill ist nicht implementiert und auch nicht in Planung.[3] Dadurch lassen sich keine eigenen Marker auf der Kartenansicht darstellen.
	Es lassen sich keine Map-Kacheln von einem beliebigen Service darstellen.
Variante B: Extended <i>React Native</i> Map Komponente	
Vorteile	Nachteile
Bietet alle benötigten Funktionen (eigene klickbare Marker platzieren und vieles mehr).	Nutzt die native Map API von <i>Apple iOS</i> und <i>Android</i> SDK. Ist fest mit <i>Apple</i> und <i>Google Maps</i> verbunden.
Wird von <i>Facebook</i> empfohlen.	Native Map APIs sind für ein <i>OSM</i> -Projekt aus moralischen Gründen unpassend.
Variante C: Mapbox GL Library	
Vorteile	Nachteile
Lässt sich mit Offline-Kacheln von <i>OSM2VectorTiles</i> ¹⁰ füttern (Vektor Kacheln).	Diese Library ist eine experimentelle Komponente.[26]
Es können eigene Marker-Bilder eingesetzt werden.	
Variante D: Portierung von Leaflet nach React	
Vorteile	Nachteile
	Die Darstellung ist nur in einer Web-view möglich.
Variante E: Raster-Tiles selbst darstellen	
Vorteile	Nachteile
	Bringt einen zu hohen Aufwand mit sich.

Tabelle 3.3.: Bewertung Map-Komponente

Varianten, die native Map APIs von *Google* und *Apple* verwenden, kamen für uns nicht in Frage. Wir möchten mit unserer App *OSM* Daten verbessern und möchten somit aus moralischen Aspekten auch auf diese Karte setzen.

Bei der *React Native* MapView-Komponente gab es keine Möglichkeit, Bilder auf der Karte

darzustellen und die Raster-Tiles von Hand anzuzeigen wäre schlicht zu aufwendig. Es liesse sich auch nur sehr umständlich eine schöne Map designen.

Somit sprang uns als erstes die Portierung von Leaflet für *React* ins Auge. Nach genauerem Betrachten fiel uns aber auf, dass diese Variante nur möglich ist, wenn die Karte in einer WebView-Komponente von *React Native* dargestellt wird. Das heisst, die App müsste zur Browser-Ansicht wechseln.

Als letzte Möglichkeit blieb die *Mapbox* Library. Diese konnte unsere Anforderungen trotz des experimentellen Zustands erfüllen und es war ersichtlich, dass *Mapbox* viel investiert um offene Fehler möglichst bald zu beheben. Die Kosten bei einer Anzahl von 50 000 Nutzern pro Monat sind für dieses Projekt nicht problematisch.[24]

Der Gedanke einer Offline-Unterstützung wurde auch besprochen. Nur unterstützt dies das Backend nicht und es gäbe Probleme mit dem Speicherplatzbedarf der App.

3.4. OAuth Implementation

Gebraucht wird ein Login-Dienst für *Google*-, *Facebook*- und *OSM*-Konten. Für einen *OSM*-Login muss eine eigene Lösung entwickelt werden. Für die Implementation der Authentifizierung wurden folgende zwei Möglichkeiten evaluiert:

Variante A: Auth0

*Auth0*¹¹ bietet eine Implementation für beliebige OAuth 2 Dienste.

Variante B: Open-Source-Projekte

Ein geeignetes Open Source Projekt für die *Google*-Authentifizierung wäre *react-native-google-signin*¹². Für *Facebook* bot sich *react-native-facebook-login*¹³ an. Diese Open Source Variante ist zu generell und wurde nicht in einer Tabellenansicht dargestellt.

Variante A: Auth0	
Vorteile	Nachteile
Sehr einfach Einbindung.	Nur OAuth 2 Unterstützung.[2]
Kostenlos für Open-Source-Projekte.	Backend Anpassung nötig.
	Schlecht erweiterbar mit eigenem Login für OSM.

Tabelle 3.4.: Bewertung OAuth-Komponente

¹¹<https://github.com/auth0/react-native-lock>, <https://auth0.com/>

¹²<https://github.com/devfd/react-native-google-signin>

¹³<https://github.com/magus/react-native-facebook-login>

Auth0 kam definitiv nicht in Frage, da es in nächster Zukunft nicht mit einer [OAuth 1.0a](#)-Authentifizierung, wie sie von *OSM* unterstützt wird, erweiterbar ist. Entschieden haben wir uns für das *react-native-google-signin*-Projekt. Es funktioniert auf beiden Plattformen und liefert ein Token, das vom KORT-[Backend](#) überprüft werden kann. Der Nachteil ist, dass es kein Open Source Projekt gab, welches alle gewünschten Social-Logins für *iOS* und *Android* anbietet.

Facebook wird vom KORT-Backend derzeit nicht unterstützt und das *react-native-facebook-login*-Projekt auf *GitHub* liefert auch kein Token, wie es beim *Google*-Login der Fall ist. Um dies zu behandeln wären weitere Anpassungen am Backend nötig gewesen. Wir hatten uns dazu entschieden, keine Backend-Anpassungen durchzuführen.

3.5. Internationalisierung

Die Übersetzungen, die für die Internationalisierung (I18n) der App zur Verfügung stehen, wurden in der KORT-Bachelorarbeit von 2012 mit *Transifex*¹⁴ bereits erhoben.[21] Es stehen bereits 13 vollständig und weitere 13 teilweise übersetzte Sprachen zur Verfügung. *Transifex* ist eine Plattform, bei der Projekte zur freiwilligen Übersetzung von dessen Benutzern aufgeschaltet werden können. Das heisst, dass sich ein Benutzer registriert und bei der Übersetzung von anderen Projekten beitragen kann. Die Übersetzungen sind allerdings im Java Properties Format, was eine Schwierigkeit darstellen würde.

Bei der Umsetzung der I18n müssen folgende Punkte umgesetzt werden: Auslesen des Locales aus den Geräteinformationen, entsprechendes Einsetzen der Übersetzungen und Einbinden der Übersetzungen, welche auf *Transifex* zur Verfügung stehen. Eine [Library](#), die eine *React Native* Bridge zum Auslesen des Locales, die I18n-Funktionalität zur Verfügung stellt und `.properties` Dateien entgegennimmt, gibt es leider nicht.

Es boten sich folgende Alternativen an:

1. Auslesen des Locales mit einer Library für *React Native*¹⁵ und Übersetzen anhand von Java Properties Files mit einer anderen Library¹⁶
2. Auslesen des Locales und Übersetzen mit einer Library für *React Native*¹⁷ und manuelles Umwandeln der Java Properties Files in JSON

Die 1. Variante kam für uns nicht in Frage, da *JQuery* nicht ins Projekt eingebunden werden soll. So haben wir uns für die 2. Variante entschieden, vorerst aber nur Deutsch und Englisch ins richtige Format umgewandelt. Die weiteren Sprachen werden noch nach der Abgabe übernommen.

¹⁴<https://www.transifex.com/>

¹⁵<https://github.com/fixd/react-native-locale>

¹⁶<https://github.com/jquery-i18n-properties/jquery-i18n-properties>

¹⁷<https://github.com/AlexanderZaytsev/react-native-i18n>

4. Resultate

Wir konnten die wichtigsten Hauptziele erreichen. KORT erfüllt nun die Grundlage aller wichtigen Anforderungen an eine moderne App. Nach dem Login wird der Benutzer wie schon bei der KORT-[Web-App](#) zur Kartenansicht weitergeleitet. Er kann auf einen Marker klicken und gelangt direkt zur Ansicht, um die gewählte Mission zu lösen. Der Zwischenschritt, dass der Benutzer zuerst gefragt wird, ob er die Lösung kennt, wurde weggelassen. Somit entfällt ein weiterer Klick, was die Spielmechanik vereinfacht. Nur noch das Marker-Icon auf der Karte verrät etwas über den Missionstyp und weckt dabei immer noch die Neugier beim Benutzer.

4.1. Zielerreichung

Erreichte Ziele:

- Eine *Android*-App mit gleicher Grundfunktionalität, wie die Web-App.
- *iOS*-App nur zu Teilen umgesetzt.
 - Nicht getestet
- Ein neuer Validierungsmechanismus
- Ein Erfahrungsbericht zu *React-Native*.
- Die Internationalisierung wurde umgesetzt.

Das [GUI](#)-Design ist noch nicht ansprechend und erfüllt nur die minimalen Ansprüche. Für eine finale Version, die veröffentlicht werden kann, muss das Design aufgebessert werden.

Zu Beginn des Projekts wurde die App für *Android* und *iOS* umgesetzt, wobei die *iOS*-App im späteren Verlauf vernachlässigt werden musste und nicht mehr explizit getestet wurde. Bei einem kurzen Test gegen Ende des Projekts mussten wir feststellen, dass der *Google* Login nicht funktioniert.

Social-Login wurde nur für *Google* realisiert.

Die Badges, welche vom [Backend](#) empfangen werden, konnten durch den neuen Validierungsmechanismus nicht mehr verwendet werden. Das Backend macht nämlich immer noch die Unterscheidung zwischen Missionen und Validierungen. Es hat zeitlich nicht mehr gereicht um die Kartenansicht beim Lösen einer Mission anzuzeigen. Die Veröffentlichung im *Google*-

Play Store ist im [Meilenstein 9](#) aufgeführt.

Wie diese zusätzlichen Arbeiten umgesetzt werden könnten wurde im Kapitel [Vorgehen](#) dokumentiert.

4.2. Ausblick

Offene Punkte und nächste geplante Arbeiten mit höherer Priorität:

- *OSM*-Login
- Finale *iOS*- und *Android*-App
- Veröffentlichung im *Apple App Store* und *Google Play Store*
- Promotions-Funktion

Das genaue Vorgehen, wie die offenen Punkte umgesetzt werden, wird im Kapitel [Vorgehen](#) beschrieben. Für die Zukunft gibt es bereits viele weitere Ideen. Eine Liste wurde im Kapitel [Weiterentwicklung](#) erstellt.

4.3. Persönliche Berichte

Dominic Mülhaupt

Nach dem [Kickoff Meeting](#) war ich sehr begeistert von KORT. Der Anfang gestaltete sich dann aber insbesondere in Bezug auf die Planung und die technische Analyse eher harzig. Ähnlich wie Marino Melchiori hatte ich kaum Kenntnisse in *JavaScript*. Mit *React* und *React Native* kamen noch zwei weitere grössere Systeme hinzu, womit es am Anfang schwer fiel, zu wissen, wo man überhaupt beginnen soll. Diesbezüglich bin ich unserem Betreuer, Prof. Stefan Keller, sehr dankbar, dass er uns die nötige Zeit gab, uns in viele der für uns neuen Technologien einzuarbeiten.

Darüber hinaus gab es auch noch mehrere, manchmal kuriose Probleme im Zusammenhang mit dem sich noch immer in Entwicklung befindenden *React Native*. Dies hat dazu geführt, dass ich oft mehrere Stunden mit Fehlersuche beschäftigt war, was manchmal etwas frustrierend war. Im späteren Verlauf der Arbeit hat sich aber ein Fluss eingestellt, wobei mich auch die verwendeten Technologien immer mehr überzeugen konnten. Das Produkt zum Zeitpunkt der [Abgabe](#) ist noch nicht vollends befriedigend, da es noch einige Punkte gibt, die sich relativ schnell umsetzen liessen. Deshalb werde ich auch gerne noch darüber hinaus am Projekt weiterarbeiten.

Ich würde mich freuen, zu einem späteren Zeitpunkt wieder einmal ein Projekt mit *React Native* durchführen zu können, da ich das Konzept sehr ansprechend finde.

Marino Melchiori

Beim Start des Projektes hatte ich keine *JavaScript*-Erfahrung. Der Einstieg in *JavaScript* war sehr schwer. Vor allem das Verstehen des Konzepts von *React* war nochmals etwas ganz Neues. Dazu kam Flexbox, das am Anfang für simple Layouts extrem aufwendig und kompliziert wirkte.

Die verwendeten Libraries erzeugten oft unerklärliche Fehler, die nach langem durchforsten von *GitHub*-Issues behoben werden konnten. Das verzögerte immer wieder die Planung und unterbrach den Fluss.

Im Verlauf der Arbeit gelang es mir aber, dank der Zusammenarbeit in unserem Team, mich gut einzuarbeiten. Das Highlight dieser Arbeit war für mich die Implementation der App mit *React Native* und die Gestaltung der Benutzeroberfläche mit *JSX*. Flexbox gefiel mir mit der Zeit immer besser, da die Wiederverwendbarkeit von Layouts sehr praktisch ist.

Trotz all den neuen und teilweise unreifen Technologien ist es uns gelungen, eine gute App-Idee neu zu entwickeln. Ich bin froh, dass ich *JavaScript*-Erfahrungen sammeln konnte. *React Native* würde ich in Zukunft sehr gerne wieder einsetzen und ich bin überzeugt, dass sich diese Technologie durchsetzen wird.

Teil II.

Projektdokumentation

5. Anforderungsspezifikation

5.1. Anforderungen an die Arbeit

Die [Autoren](#) hatten im Vorfeld der Arbeit kaum Kenntnisse zu Webtechnologien und insbesondere *JavaScript*. Insofern war der machbare Umfang des Projekts schwer absehbar. In Abstimmung mit dem [Betreuer](#) und dem [Projektpartner](#) wurde deshalb festgelegt, dass der Fokus auf dem Frontend liegt, so dass sich die Entwickler (Marino Melchiori und Dominic Mülhaupt) nicht auch noch in die Technologien des [Backends](#) einarbeiten müssen.

In der Anforderungsanalyse sind viele Aufgaben erkannt worden, die in diesem Projekt bearbeitet werden könnten. Im Rahmen dieser Arbeit ist nur ein Bruchteil davon umsetzbar. Zum einen aus zeitlichen Gründen und zum anderen, weil Anpassungen am Backend nötig wären. All diese Anforderungen sind in *Muss*, *Soll*, *Kann*, *zukünftige Arbeiten* und *abgewiesene Arbeiten* unterteilt.

5.1.1. Muss

- neuer Kort-Client als native *Android*-App
- Ablösung des Validierungsmechanismus
- Erfahrungsbericht mit Hinweisen zu Tutorials zu React Native
- die vom Studiengang geforderten Lieferobjekte: Dokumentation, Management Summary, Abstract, Poster, Präsentation mit Stellwand, Zwischenpräsentation, Schlusspräsentation

5.1.2. Soll

- Kort-Client als native *iOS*-App

5.1.3. Kann

- neue Funktion: Promotions anzeigen
- Kurzvideo (zur Instruktion und Promotion)

5.2. Use Cases

Alle Use Cases für die KORT-App sind im Kapitel 4.1.1. User Szenarien der Bachelorarbeit von Jürg Hunziker und Stefan Oderbolz beschrieben und entsprechen weiterhin den Anforderungen.[\[21\]](#) Es wurden folgende vier Szenarien beschrieben:

- Szenario 1: Zeitvertrieb an der Bushaltestelle
- Szenario 2: Validieren
- Szenario 3: Erster Kontakt zur App
- Szenario 4: Highscore-Anwärter

6. Technologien

In diesem Kapitel sind Informationen zur Funktionsweise der Technologien *React* und *React Native* dokumentiert. Das Unterkapitel zu *React Native* beschreibt ebenfalls unsere Erfahrungen und Schlüsse.

6.1. React

*React*¹ wurde im März 2013 veröffentlicht[36], ist eine Open Source *JavaScript Library* und dient zur Implementation von User Interfaces. Die View besteht aus wiederverwendbaren Components, die wiederum Components beinhalten. Was Components genau sind und wie diese verwendet wurden ist im Kapitel [View Components](#) beschrieben. *React* wird von *Facebook*, *Instagram* und von der Community entwickelt und gewartet.[37]

Für *React* wird *JSX*, welches eine HTML ähnliche Syntax nutzt, zur Erstellung der Komponenten empfohlen. So lassen sich Komponenten-Bäume direkt mit *JavaScript* erstellen. Anders formuliert können *JavaScript*-Objekte mit einer HTML-Syntax erzeugt werden. Eine Hauptkomponente gibt seine Daten per Props an die Kind-Komponenten weiter (one-way-dataflow).[11] *JSX* wird nicht zwingend benötigt.[7]

Anstatt der *DOM* nutzt *React* die sogenannte *Virtual DOM*. Wie der Begriff schon sagt, wird mit einer Abstraktion der echten DOM – also mit einer virtuellen DOM – kommuniziert. Das komplette DOM (also die Repräsentation der View vom HTML-Code) ist im lokalen Speicher abgelegt.[23] In der `render()` Methode jeder *React*-Klasse wird eine Beschreibung des DOM zurückgeliefert, die *React* mit der lokalen Kopie der DOM vergleicht. Mit einem sehr effizienten Diffing-Algorithmus berechnet *React* den Unterschied zwischen diesen Versionen des DOM und errechnet den schnellsten Weg um den Browser zu aktualisieren.[17]

6.2. React Native

Der Ansatz von *React Native*² ist *learn once – write anywhere*, das heisst, lerne eine Technologie und nutze sie für alle unterstützten Plattformen.[15] Dieses Kredo bezieht sich darauf, dass *JavaScript* sehr weit verbreitet ist und mit *React Native* nicht extra eine neue Programmiersprache erlernt werden muss um Apps für *Android* und *iOS* zu schreiben.

¹<https://facebook.github.io/react/>, <https://github.com/facebook/react>

²<https://facebook.github.io/react-native/>, <https://github.com/facebook/react-native>

Am 26. März 2015 wurde *React Native* erstmals für *iOS* veröffentlicht. Im Oktober 2015 kam *Android* dazu.[16] Seit dem Release gibt es alle zwei Wochen eine neue Version. Durch diese häufigen Änderungen konnten sich noch keine Best Practices etablieren. Auch die meisten Open Source Projekte verfolgen eigene Implementationsansätze.

Eine Desktop Unterstützung für *OSX* ist ebenfalls in Entwicklung³. Und am 13. April 2016 an der *Facebook Developer Konferenz* kündigten *Microsoft* und *Facebook* den Support für die *Universal Windows Platform (UWP)* an.[25]

6.2.1. Layout

Alle **GUI**-Komponenten befinden sich in sogenannten Containern. Ein Container wird durch eine **View**-Komponente definiert. Das Layout und die Gestaltung der Container und Komponenten wird mit *Flexbox*⁴ geregelt.

6.2.2. Technische Details

React Native nutzt einen *JavaScript*-Layer, beziehungsweise *JavaScriptCore* als Engine, um den Code auszuführen.[8] Die native Funktionen werden auf die *JavaScript*-Objekte oder Funktionen gemappt. Das Endprodukt ist also keine **Web-App** für den Browser und wird auch nicht in native Code kompiliert. Ausserdem wird der *JavaScript*-Code auf einem separaten Thread ausgeführt und nicht auf dem UI-Thread. Dadurch wirken zum Beispiel die Animationen sehr flüssig.[14]

Native Module

Damit ein *iOS* Native-Modul in *React Native* verwendet werden kann, muss das Modul das *RCTBridgeModule*-Protokoll⁵ implementieren und das Makro `RCT_EXPORT_MODULE()` enthalten. Das Protokoll dient nur dazu, das Modul in einem Array zu speichern, damit es später von der Bridge gefunden werden kann. Wenn die *JavaScript*-Seite der Bridge initialisiert ist, kann sie auf diese Daten zugreifen. Dem Makro kann auf der *JavaScript*-Seite ein optionaler Name als Parameter mitgegeben werden. Falls dieser Parameter fehlt, wird die Komponente auf *JavaScript*-Seite nach dem *Objective-C*-Klassennamen benannt. Ein Ähnliches Vorgehen gilt für *Swift*- und *Android*-Module. Genauere Hinweise sind in der *React Native*-Dokumentation unter *Native Modules* beschrieben.[13][12] Mit diesem Feature lässt sich bereits vorhandener native Code wiederverwenden.

³<https://github.com/ptmt/react-native-desktop>

⁴<https://facebook.github.io/react-native/docs/flexbox.html>. Dabei handelt es sich um einen Polyfill für die durch das *W3C* spezifizierte *Flexible Box*: <https://www.w3.org/TR/2016/CR-css-flexbox-1-20160526/>

⁵RCT ist eine Abkürzung für ReaCT

6.2.3. Setup

Die Entwicklungsumgebung lässt sich am schnellsten und einfachsten auf *OS X* einrichten. *Windows* und *Linux* sind mittlerweile ebenfalls geeignet, was zu Beginn dieser Bachelorarbeit nicht der Fall war.

Mit dem *React Native*-CLI (Command Line Interface) können neue Projekte initialisiert werden und Projekte auf einem Emulator oder einem Gerät getestet werden. Das initialisierte Projekt enthält wiederum ein *Android*- und ein *iOS*-Projekt. Diese Ordner enthalten den vorgegebenen Aufbau für Projekte der jeweiligen Plattformen. Ausserdem befinden sich die beiden Dateien `index.android.js` und `index.ios.js` im generierten Ordner. Diese werden beim Starten der App ausgeführt und verwenden weitere *JavaScript* Module, welche sich bei uns im `js` Ordner befinden.

6.2.4. Cross Platform

Auch wenn es nicht der ursprüngliche Gedanke von *React Native* war, ein Cross Platform Framework anzubieten – *React Native* hat ursprünglich nur *iOS* unterstützt –, sind die Konzepte dafür sehr ausgereift.

Zum einen gibt es *JavaScript* Module, die die native Module der beiden unterstützten Plattformen unter der selben API anbieten (`AsyncStorage`⁶ oder `ListView`⁷ sind Beispiele dafür). Zum anderen – wenn ein plattformspezifischeres Verhalten erforderlich ist – kann es sein, dass für die verschiedenen native Module auch unterschiedliche *JavaScript* Module angeboten werden (als Beispiel dafür dient die Lade-Anzeige mit `ProgressBarAndroid`⁸ und `ActivityIndicatorIOS`⁹). In diesem Fall gibt es zwei Möglichkeiten, die Module im Client Code zu verwenden. Die eine Möglichkeit ist, in einem `if`-Statement zu überprüfen, auf welcher Plattform die App läuft (`if (Platform.OS === 'ios')`) und das passende Modul einzusetzen.¹⁰ Die andere Möglichkeit ist, dass der Client Code auf zwei verschiedene Dateien aufgeteilt wird, die durch die Endung `.android.js` respektive `.ios.js` unterschieden werden.

Diese Konzepte bauen auf dem modularen Ansatz von *React* auf und bieten die Freiheit, auf elegante Weise eine plattformunabhängige App zu entwickeln.

6.2.5. Community

Die Community ist auf mehrere Portale verstreut, wobei viele wichtige Informationen in den Issues des *React Native* GitHub-Repositorys¹¹ verborgen sind. Ausserdem sind aktuell (am

⁶<https://facebook.github.io/react-native/docs/asyncstorage.html>

⁷<https://facebook.github.io/react-native/docs/listview.html>

⁸<https://facebook.github.io/react-native/docs/progressbarandroid.html>

⁹<https://facebook.github.io/react-native/docs/activityindicatorios.html>

¹⁰Es gibt auch noch weitere Varianten: <https://facebook.github.io/react-native/docs/platform-specific-code.html>

¹¹<https://github.com/facebook/react-native/issues>

16.06.2016) 741 offene Issues vorhanden – am 29.04.2016 waren es noch rund 500.

- Vorhanden ist eine öffentliche, aktive und hilfsbereite *Facebook*-Gruppe¹² mit derzeit ca. 3 500 Mitgliedern.
- Es gibt eine *Stack-Overflow*-Kategorie¹³ – leider mit nur wenigen Antworten und Lösungen.
- JS.coach¹⁴ - listet viele Open Source Projekte auf.
- Übersicht über aktuelle Artikel und Blogposts:
 - reactnative.com¹⁵
 - *React Native* Newsletter¹⁶
- Es ist ebenfalls ein aktiver Subreddit¹⁷ vorhanden.
- Das bekannte Awesome React Native¹⁸ *GitHub* Repository listet viele Artikel, Tutorials, Libraries und mehr.

6.2.6. Erfahrungen

Eine Stärke von *React Native* ist die Plattformunabhängigkeit. Wenn keine spezifischen *Android*- oder *iOS*-Komponenten verwendet werden, kann der Code für beide Plattformen genutzt werden.

Die grössten Hürden von *React Native* sind das Erlernen von *React* und *Flexbox*. Denn *Flexbox* für *React Native* unterscheidet sich in vielen Details vom *Flexbox* für Webseiten. Einerseits ist das Grundkonzept für das Layout bei vielen verschachtelten Komponenten schwer zu verstehen. Selbst Die Umsetzung von ganz simplen Views ist am Anfang schwer und frustrierend. Andererseits wirkt es, nachdem viele Stunden in das Lernen investiert wurden, doch konsistent und praktisch. Dazu kommt, dass durch *Flexbox* die definierten Styles gut wiederverwendbar sind. Das Gleiche gilt für jegliche Komponenten, die auch gut durch Inspiration aus anderen Open Source Projekten erstellt werden können. Durch die wachsende Community gibt es immer mehr solcher Projekte, Pull-Requests und Beiträge an *React Native* selber. Das macht *React Native* langfristig gesehen zu einer immer besseren und wichtigeren Technologie.

Die Umstellung von der native Entwicklung zu *React Native* ist Anfangs auch schwer. Vor allem wenn Fehler auf *React Native*-Seite existieren, die bei der native Entwicklung nicht durch unschöne Workarounds übersteuert werden müssten. Mit der Zeit zeigen sich aber die gros-

¹²<https://www.facebook.com/groups/react.native.community/>

¹³<http://stackoverflow.com/questions/tagged/react-native>

¹⁴<https://js.coach/>

¹⁵<http://www.reactnative.com/>

¹⁶<http://reactnative.cc/>

¹⁷<https://www.reddit.com/r/reactnative>

¹⁸<https://github.com/jondot/awesome-react-native>

sen Vorteile von *React-JavaScript*. Jede einzelne Komponente wird allein durch ihren State kontrolliert. Je nach State kann die Darstellung der Component durch die eigene `render()` Methode kontrolliert und verändert werden.

Ein angenehmes Feature für die Entwicklung ist das Live- und das Hot-Reload Feature. Nur beim ersten Ausführen der App muss ein Build erstellt werden. Dank dem gleichzeitigen Starten des [Packagers](#) kann neu geschriebener *JavaScript*-Code direkt in die auf dem Emulator oder dem Smartphone laufende App geladen werden.

Der App-Showcase der offiziellen *React Native*-Dokumentation wächst stetig¹⁹. Wir gehen davon aus, dass *React Native* zusammen mit *Xamarin*²⁰ in Zukunft eine wichtige Rolle in der mobilen Cross-Platform-Entwicklung einnehmen wird.

¹⁹<https://facebook.github.io/react-native/showcase.html>

²⁰<https://www.xamarin.com/>

7. Design

7.1. Datenmodell

Die Model Logik ist weitestgehend im [Backend](#) implementiert, welches die Daten über eine [REST](#)-Schnittstelle zur Verfügung stellt. Die über die Schnittstelle zur Verfügung gestellten Informationen sind also Ausgangspunkt für die von uns verwendbaren Domain Klassen. Es waren kleinere Anpassungen nötig, etwa um den neuen Validierungsmechanismus zu unterstützen. Da die empfangenen Daten fast ausschliesslich als Repräsentation von Informationen dienen und keine eigene Logik implementieren, wurden sie als [Datentransferobjekte](#) umgesetzt.

Das folgende Datenmodell repräsentiert die [Datentransferobjekte](#) und ist nicht als semantisches Modell zu verstehen.

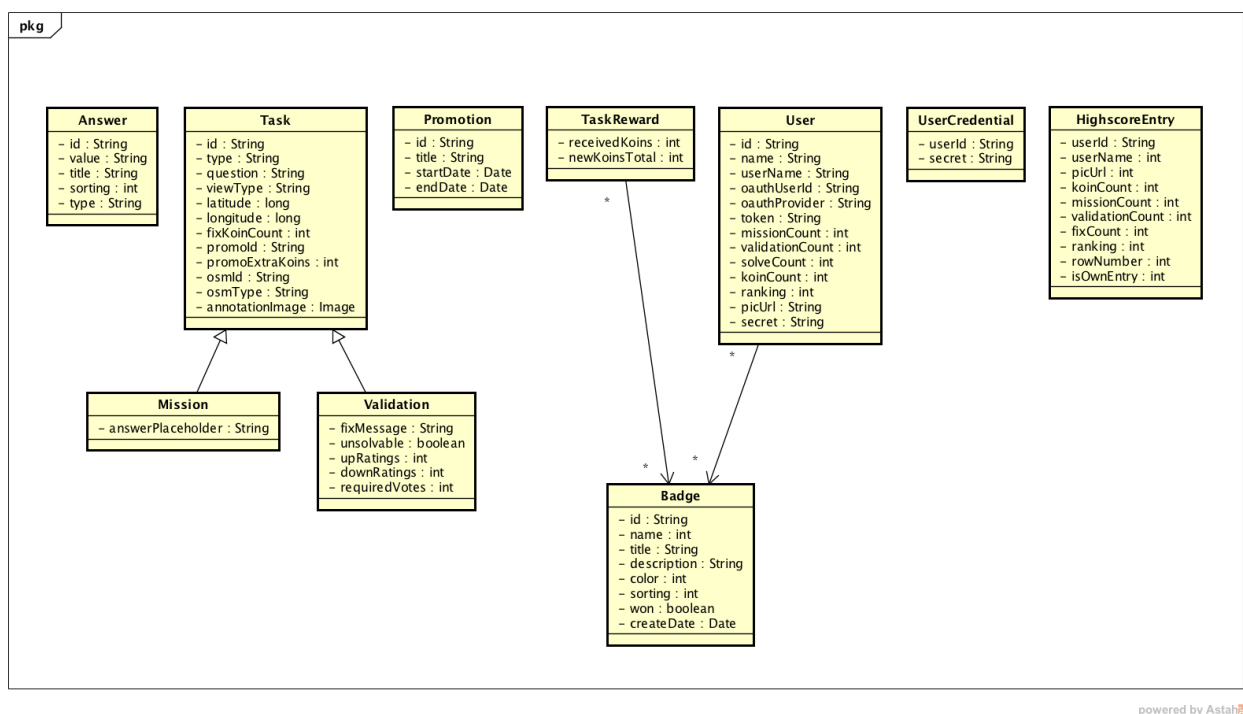


Abbildung 7.1.: Modellierung der DTOs

7.2. Architektur

Für unsere Applikationsarchitektur haben wir das *Flux*¹ Architekturpattern von *Facebook* eingesetzt.

Die folgenden Erklärungen wurden zu grossen Teilen aus der *Flux* Dokumentation[6] abgeleitet. Zu den nachfolgend beschriebenen Konzepten der *Flux*-Architektur mussten für unsere Realisierung keine Anpassungen gemacht werden. Die Beschreibungen entsprechen also unserer Implementation.

Flux ist eine Architektur, die in *Frontend*-Applikationen eingesetzt wird. Ein unidirektionaler Datenfluss ist das Grundprinzip, auf welchem *Flux* aufbaut, und in welchem es sich von *MVC*-Architekturen unterscheidet. Das Pattern beschreibt folgende vier Hauptbestandteile: die Stores, die Views (oder Controller-Views), die Actions und der Dispatcher. Ein Beispiel, wie die Architektur in der Abbildung 7.2 umgesetzt wurde, ist im Sequenzdiagramm zum *Lösen einer Mission* beschrieben.

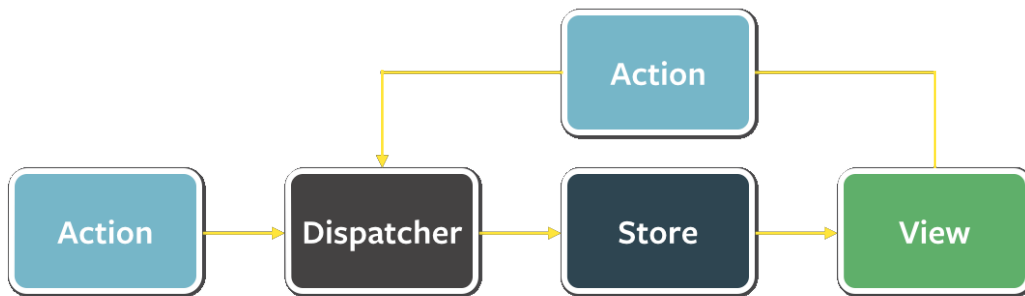


Abbildung 7.2.: Idee der *Flux*-Architektur

7.2.1. Stores

Stores enthalten den Applikationszustand und die Applikationslogik. Verglichen mit dem *MVC*-Pattern entsprechen sie am ehesten dem Model. Sie unterscheiden sich aber insofern, dass sie nicht unbedingt einzelne Modellklassen repräsentieren sollen, sondern domänenspezifische Aufgaben übernehmen. Da KORT keine besonders komplexe Domäne enthält, ist diese Unterscheidung allerdings nicht von grosser Bedeutung. Ein Beispiel für diese Einteilung nach Domäne und nicht nach Modell ist in der Aufteilung von *UserStore* und *AuthenticationStore* zu finden. Stores sind als *Singletons* umgesetzt. Sie registrieren sich beim Dispatcher um Updates zu erhalten und ihren Zustand entsprechend anzupassen. Views wiederum können sich bei den Stores registrieren um neue Informationen zu erhalten, können die Stores aber nicht direkt anweisen, sich zu aktualisieren.

¹<https://facebook.github.io/flux/>

7.2.2. Views

Der Applikationsanwender kommuniziert seine Absichten über die View. Deshalb gilt die View in *Flux* als Auslöser für neue Aktionen. In *React* kann zwischen *Views* und *Controller-Views* unterschieden werden.

Controller-Views finden sich an der Spitze der View-Hierarchie. Sie warten auf Updates der Stores und reichen die Daten entlang der Kette ihrer untergeordneten *Views* weiter.

Diese untergeordneten *Views* wiederum reagieren auf Zustandsänderungen indem ihre `render()` Methode neu aufgerufen wird. Somit bleiben *View* Komponenten modular austauschbar, da sie unabhängig von ihrem Kontext eingesetzt werden können.

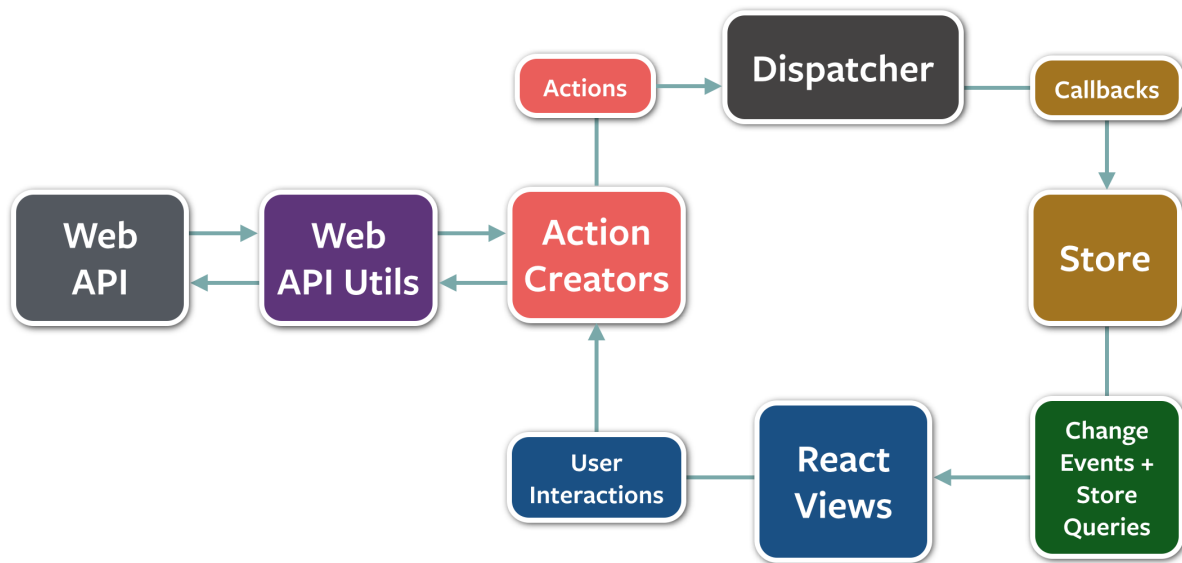
7.2.3. Actions

Actions sind Helfermethoden, welche im Dispatcher ein Ereignis und somit in den Stores ein Update auslösen. Sie werden ausschliesslich durch Views ausgelöst, da diese für den Kontrollmechanismus der Applikation zuständig sind. Ausnahmen wären hier denkbar, waren aber nicht nötig. Beispielsweise könnte der `LocationStore` eine Action auslösen wenn er eine neue Position erkannt hat oder der Server wenn ein Update an die Applikation gesendet werden soll.

Ausserdem sollten Daten, wenn diese für ein Update nötig sind, bereits durch die Actions an den Dispatcher mitgeliefert werden. Aufrufe an die [REST](#)-Schnittstelle des [Backends](#) werden also über die Actions ausgelöst. Somit kann sichergestellt werden, dass verschiedene Stores, welche auf dieselbe Action reagieren, denselben [API](#)-Aufruf mehrmals ausführen. In Abbildung [7.3](#) ist dies ersichtlich.

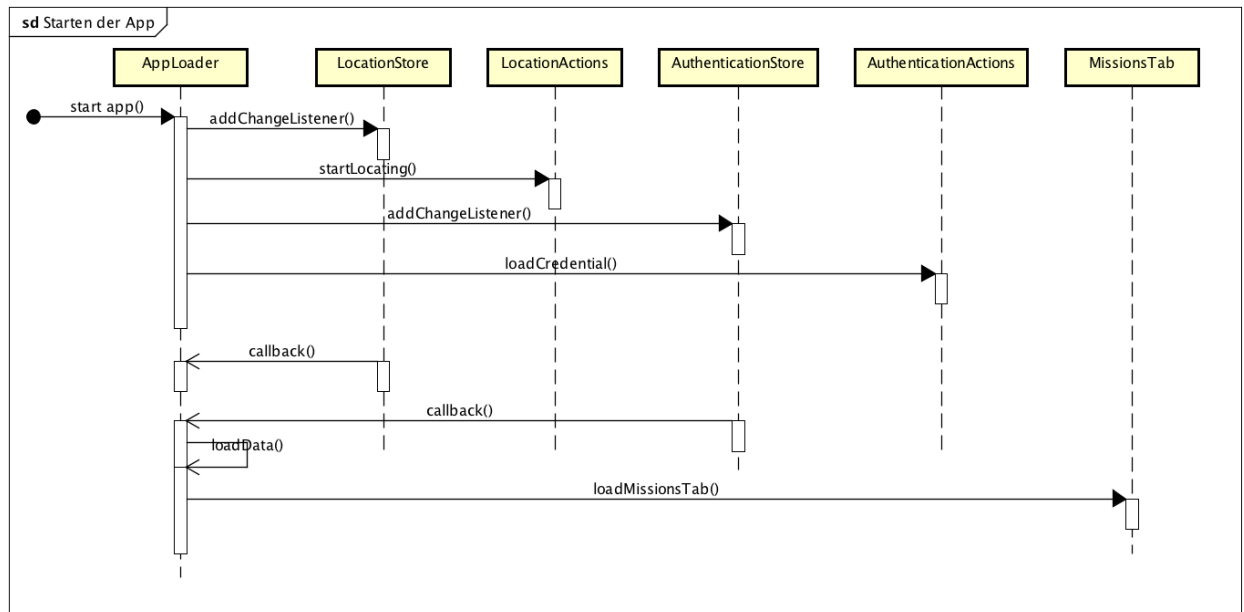
7.2.4. Dispatcher

Der Dispatcher ist der zentrale Knotenpunkt, durch den der gesamte Datenfluss der Applikation koordiniert wird. Seine einzige Aufgabe ist die Verteilung der Actions an die Stores. Er enthält also keine intelligente Logik, sondern ist grundsätzlich ein Register von Callbacks.

Abbildung 7.3.: Vollständiges *Flux*-Diagramm[5]

7.3. Sequenzdiagramme

7.3.1. Starten der App



powered by Astah

Abbildung 7.4.: Bevor Daten geladen und Informationen dargestellt werden können, muss der Benutzer autorisiert und lokalisiert sein.

7.3.2. Mission lösen

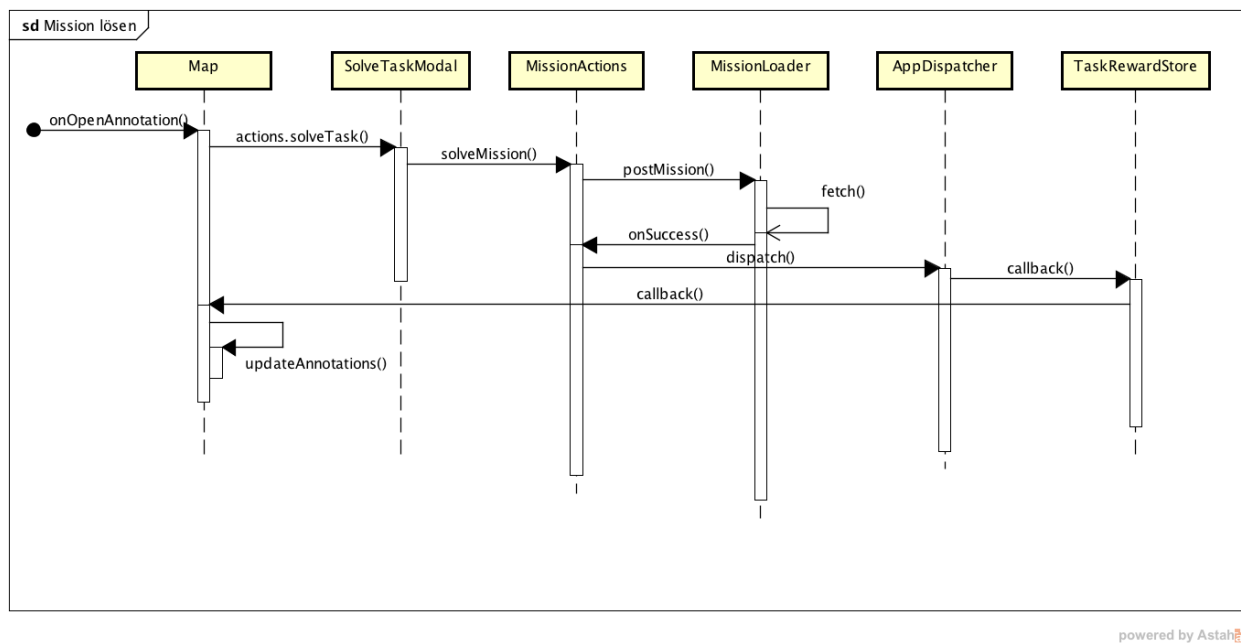


Abbildung 7.5.: Lösen einer Mission und Anzeigen der Belohnung

8. Entwicklungsumgebung

8.1. IDE

Die Funktionalitäten und Features der App wurden alle mit *Atom* implementiert. Für das Debugging waren die native IDEs *Android Studio* und *Xcode* aber besser geeignet. Das Debuggen in der *Atom-IDE* oder im *Google Chrome* Browser war fehlerbehaftet. Die native IDEs wurden des Weiteren verwendet um projektspezifische Konfigurationen – etwa das Einbinden von Libraries oder Build-Konfigurationen – vorzunehmen. Ansonsten wurden die native IDEs nur für das Einfügen von statischen Bildern mit passenden Auflösungen für entsprechende Displays genutzt.

8.2. Continuous Integration

Als Versionsverwaltungssystem wurde *git*¹ zusammen mit dem Online-Dienst *GitHub*² verwendet: <https://github.com/kort/kort-reloaded>. Dabei haben wir folgendes Branching Model eingesetzt: Der master Branch wird nur für Releases verwendet. Für die Entwicklung wurde der develop Branch genutzt, wobei jedes Feature und jeder Fix eines Bugs einen eigenen Branch erhielt, welcher nach Fertigstellung wieder in den develop Branch gemerged wurde.

Für die *Continuous Integration* (CI) nutzen wir den freien Dienst von *Travis-CI*³. Dieses Setup lässt sich bequem in Verbindung mit *GitHub* nutzen. Dabei wird bei jeder Neuerung auf dem master und develop Branch über *Travis-CI* ein neuer Build erstellt. Bei jedem Build werden die Tests durchlaufen und der Code auf die Einhaltung der *Code-Richtlinien* überprüft.

Die Konfigurationsdatei für *Travis-CI* (`.travis.yml`) befindet sich im *GitHub*-Repository von KORT.

8.3. Projektmanagement-Tool

Als Projektmanagement-Tool wurde *Redmine* verwendet. Weiterführende Links zum Redmine-Projekt, das für die Ticketerfassung verwendet wurde, sind im Kapitel *Projektmanagement* dokumentiert.

¹<https://git-scm.com/>

²<https://github.com>

³<https://travis-ci.org/>

8.4. Testing

Fürs Testing wurden [Unit Tests](#), [Integration Tests](#) und [Funktionalen Tests](#) evaluiert. Letztlich konnten – zum Zeitpunkt der Abgabe – lediglich die Unit Tests umgesetzt werden. Integration Tests wären für die `data` Klassen wünschenswert gewesen, konnten aber aus Zeitgründen nicht umgesetzt werden. Automated UI Tests sind mit *React Native* möglich⁴, sind aber sehr aufwendig einzurichten. Für die Unit Tests wurde *Jest*⁵ eingesetzt. *Jest* ist ein *JavaScript* Testing-[Framework](#) und wird zum Beispiel von *Facebook* zum Testen von *React*-Applikationen verwendet.

Eine besondere Eigenschaft von *Jest* ist, dass standardmässig für alle Module automatisch [Mocks](#) bereitgestellt werden. Somit wird verhindert, dass aus Versehen das Verhalten anderer Module getestet wird. Leider gab es in diesem Zusammenhang öfters Probleme beim Mocken vom *Node* Modulen.

8.5. Code-Richtlinien

Um Code-Richtlinien festzulegen und deren Einhaltung zu prüfen, wurde *ESLint*⁶ eingesetzt. Dadurch wird die Lesbarkeit und Wartbarkeit des Codes erhöht. Die Konfiguration von *ESLint* stammt von *Airbnb*⁷. Darüber hinaus wurden Plugins für *React*⁸ und *React Native*⁹ eingesetzt.

Die Konfiguration findet sich in der Datei `.eslintrc.json`.

⁴Einen guten Überblick bietet diese Seite: <http://testdroid.com/tech/testing-react-native-apps-on-android-and-ios>

⁵<https://facebook.github.io/jest/>

⁶<http://eslint.org/>

⁷<https://github.com/airbnb/javascript/tree/master/packages/eslint-config-airbnb>

⁸<https://github.com/yannickcr/eslint-plugin-react>

⁹<https://github.com/Intellicode/eslint-plugin-react-native>

9. Implementation

Dieses Kapitel beschreibt die Einrichtung und Implementation von einigen der verwendeten [Libraries](#) sowie weiteren Komponenten.

9.1. Kort Backend

Das KORT-Backend existierte bereits und wurde von den Entwicklern nicht mehr abgeändert. Das Backend konnte von uns über die [REST](#)-Schnittstelle angesprochen werden. Dieses ist im Kapitel 10.2. REST-Schnittstellen der Bachelorarbeit von Jürg Hunziker und Stefan Oderbolz dokumentiert.^[21]

9.2. View Components

Components entsprechen den [Views](#) der *Flux*-Architektur. Component Klassen erweitern `React.Component`¹ und implementieren mindestens die `render()` Methode². In der `render()` Methode wird in [JSX](#) Syntax definiert, wie die Component dargestellt werden soll.

Eine Component kann andere Components wiederverwenden, Dadurch wird sowohl ein hierarchischer Aufbau der View, als auch ein modularer Einsatz von Components ermöglicht.

Ein weiteres Konzept, das von Components realisiert wird, ist die Unterscheidung von Property und State. Eine Property repräsentiert eine sich nicht ändernde Eigenschaft einer Component, während der State den aktuellen Zustand ausdrückt. Properties werden vom Owner³ gesetzt und können durch den Ownee als `propTypes` deklariert werden. Wird eine State Variable neu gesetzt, führt dies zu einem erneuten Aufruf der `render()` Methode.

Components dürfen den Zustand – der in den Stores gehalten wird – nie direkt manipulieren, sondern müssen dies über Actions auslösen. In den Callbacks, welche sie bei den Stores registriert haben, können sie dann die Getter aufrufen um den neuen Zustand zu erhalten. Components sollen nicht mehr über den Applikationszustand wissen als für ihre Ansicht nötig ist. Die Logik sollte also so weit möglich in den Stores umgesetzt sein.

¹<https://facebook.github.io/react/docs/component-api.html>

²<https://facebook.github.io/react/docs/component-specs.html>

³Die Owner-Ownee-Beziehung wird unter *Ownership* beschrieben: <https://facebook.github.io/react/docs/multiple-components.html>

9.3. Libraries

Damit die entsprechenden [Libraries](#) genutzt werden können, mussten sie mit dem Node-Package-Manager (npm) heruntergeladen werden. Mit dem `npm install`-Befehl wurden alle aufgelisteten Abhängigkeiten in der `package.json`-Datei installiert und im Ordner `node-modules` gespeichert. Wenn eine Library eine Bridge zu nativem Code verwendet, muss sie zusätzlich im *Android*- und *iOS*-Projekt gelinkt werden. Dann konnten die Libraries in einer *JavaScript*-Komponente importiert und verwendet werden.

Nachfolgend sind die wichtigsten Libraries aufgeführt, die von uns verwendet wurden.

Library	Version	Verwendung
flux	2.1.1	Nutzung des Dispatchers
react	15.0.2.	Entwicklung vom User Interface
react-native	0.26.3	Entwicklung von native Apps mit <i>React</i>
react-native-google-signin	0.6.0	Native Authentifizierung mit Google OAuth
react-native-i18n	0.0.8	Internationalisierung der Benutzeroberfläche
react-native-mapbox-gl	4.1.1	Darstellung der Karte
react-native-router-flux	3.26.15	Navigation im User Interface

Tabelle 9.1.: Abhängigkeiten im *JavaScript*-Code

Library	Version	Verwendung
babel-eslint	6.0.4	Compiler für die verschiedenen Libraries und <i>ES2015</i>
babel-jest	12.1.0	
babel-polyfill	6.9.0	
babel-preset-es2015	6.6.0	
babel-preset-react	6.5.0	
babel-preset-react-native-stage-0	1.0.1	
eslint	2.11.1	Einhaltung der Code-Richtlinien
eslint-config-airbnb	6.2.0	
eslint-plugin-react	4.3.0	
eslint-plugin-react-native	1.0.0	
jest	12.1.1	Testing

Tabelle 9.2.: Abhängigkeiten für die Entwicklung

Die vollständige Übersicht aller Abhängigkeiten finden sich in der Datei `package.json`.

9.3.1. Navigation

Die Navigation ist in der Component App deklariert. Alle Scenes werden mit einem Key deklariert und falls nötig mit Optionen für eine Tab-Ansicht erweitert. Die definierten Scenes werden als Kind-Komponenten einem Router übergeben. Jede Scene kann mit einem Funktionsaufruf

- die Properties aktualisieren.
- sich selber schliessen.
- eine andere Scene mit dem Key und mit optionalen Properties, als Parameter, aufrufen.

Die [API](#) der Navigations-Komponente ist auf der entsprechenden *GitHub*-Seite⁴ dokumentiert.

9.3.2. Karte

Um die Map-Komponente von *Mapbox* zu nutzen und ein Access-Token zu erhalten, muss auf der *Mapbox*-Webseite⁵ ein Benutzerkonto angelegt werden.

Damit die Marker an der richtigen Position angezeigt werden, wurden der *Mapbox*-Komponente ein Array mit Annotations als Parameter übergeben. Das Annotation-Array wird fortlaufend mit allen Missionen vom [Backend](#) aktualisiert. Dabei werden nur Missionen in der Umgebung des Benutzers geladen. Eine Annotation enthält die Koordinaten und die ID einer Mission. Es war von der *Mapbox-API* her nicht möglich, einer Annotation direkt die Mission als Objekt mitzugeben. Bei einem Klick auf einen Marker wird die Funktion `onOpenAnnotation` aufgerufen. Diese Funktion öffnet wiederum eine Scene, die es dem Benutzer erlaubt, die gewählte Mission zu lösen. Die *Mapbox-API* ist auf der entsprechenden *Github*-Seite⁶ dokumentiert.

9.3.3. OAuth

Für die Authentifizierung über *Google* wurde eine Open Source [Library](#) evaluiert (Kapitel Evaluation [OAuth Implementation](#)). Damit die App sich mit der *Google-API* verbinden kann, muss eine `google-services.json`-Datei auf der *Google*-Webseite⁷ generiert werden. Diese Datei verknüpft die *Google-OAuth*-Konfiguration mit einem eindeutigen Identifier für die App (Bundle ID und Signing Key).

Nach dem Login des Benutzers erhält die App über diese [Library](#) das Benutzer-Token von *Google*. Dieses Token wird dann von der App dem KORT-[Backend](#) mitgeteilt und von dort aus

⁴<https://github.com/aksonov/react-native-router-flux>

⁵<https://www.mapbox.com/>

⁶<https://github.com/mapbox/react-native-mapbox-gl>

⁷<https://developers.google.com/cloud-messaging/android/client>

bei *Google* überprüft. Wie die Authentifizierung auf der [Backend](#)-Seite weiter verläuft ist im Kapitel 10.4.1. der Bachelorarbeit von Jürg Hunziker und Stefan Oderbolz beschrieben.[\[21\]](#)

10. Weiterentwicklung

KORT hat grosses Potenzial für weiterführende Arbeiten. Diese umfassen vor allem zwei Bereiche:

- Wie kann KORT besser dazu beitragen, *OSM* Daten zu verbessern?
- Und wie kann der Benutzer mithilfe von Konzepten der Gamification weiter motiviert werden, zur Datenpflege beizutragen?

Wir haben uns Gedanken dazu gemacht und hier zusammengefasst, wie KORT weiter optimiert werden könnte. Das Unterkapitel [Vorgehen](#) enthält Arbeiten, die beim derzeitigen Stand noch verbessert werden müssen. Weiterhin sind Ideen aufgelistet, die Anpassungen am Backend erfordern.

10.1. Vorgehen

OpenStreetMap Login

Ein sehr wichtiges Feature, das bisher noch nicht umsetzbar war, ist der *OSM*-Login. Neben den Änderungen am Backend (damit dieses Feature überhaupt brauchbar ist) gibt es folgende Schritte zu erledigen:

1. App bei *OSM* mit Callback-URL registrieren
 - Als Callback-URL könnte zum Beispiel 'http://www.kort.ch/' verwendet werden.
2. *OSM*-Request-Token-URL aufrufen um das OAuth-Token und das Token-Secret zu erhalten
3. *OSM*-Authorize-URL mit dem OAuth-Token aufrufen, damit sich der Benutzer auf der *OSM*-Seite einloggen kann
4. WebView-Component öffnen um auf die Callback-URL zu warten
5. Wenn sich der Benutzer eingeloggt hat, wird er zur Callback-Seite mit dem OAuth-Token und dem OAuth-Verifier in der URL umgeleitet.
6. WebView schliessen
7. Request *OSM*-Access-Token-URL

8. Dem Backend das Token zur Überprüfung senden

Dieses Vorgehen wurde noch nicht getestet und dient nur als Idee zur möglichen Umsetzung. Die korrekten Request-URLs können aus der Dokumentation im *OSM-Wiki*¹ entnommen werden.

Als Nächstes müsste für den **Gamification**-Ansatz das Design weiter verbessert werden. Dafür wäre zuerst eine Änderung am Backend geplant um die Validierungen endgültig abzuschaffen, indem sie als normale Missionen gezählt werden. Dann wäre es auch wieder möglich, korrekte Badges für den Missionen-Counter anzuzeigen.

Durch den Einsatz von Farben oder einem Hintergrundbild beim Login-Screen würde das Design einen Spieler besser ansprechen. Passend dazu könnten die verwendeten Bilder und Icons einheitlich gestaltet und erneuert werden.

Meldungen, die dem Benutzer die Anzahl gewonnener Coins anzeigen, erscheinen momentan im Vollbildmodus. Das liegt an einem Fehler der eingesetzten Navigations-Komponente, die keine transparenten Hintergründe zulässt. In Zukunft könnte das aber noch behoben werden. Dann wäre es eleganter, Meldungen in einem Fenster zu zeigen.

GUI-Arbeiten

- Textfeld zur Anpassung des Benutzernamens anbieten.
- Layout und Design aufbessern.
- Tab-Icons für *iOS* einfügen.

Bis die Funktionalität der **Web-App** komplett umgesetzt ist, fehlen noch zwei Features. Diese wären die zweite Highscore-Ansicht und die Karte, die beim Lösen einer Mission angezeigt wird. Die zweite Highscore-Ansicht zeigt den eingeloggten Benutzer mit seinen direkten Konkurrenten, die vor und nach ihm platziert sind. Wenn sich der Benutzer in der Ansicht zum Lösen einer Mission befindet, sollte in einem weiteren Tab eine Karte, die das Objekt der Mission hervorhebt, angezeigt werden.

Testing

Gerne hätten wir noch funktionale Tests mit den **Components** durchgeführt, doch die Priorität dafür war zu tief und die Zeit zu knapp. Dazu haben wir für die Zukunft die Enzyme²-Testing-API (für **Component**-Tests) und das Mocha³-**Framework** (um die Tests laufen zu lassen) evaluiert.

¹<http://wiki.openstreetmap.org/wiki/OAuth>

²<http://airbnb.io/enzyme/>

³<https://github.com/mochajs/mocha>

10.2. Weiterführende Arbeiten

Punkte, die KORT attraktiver machen würden:

- KORT zusätzlich als Web-App – umgesetzt mit *React* – anbieten
- Benutzerlogin mit weiteren OAuth-Diensten erweitern (z. B. *Twitter*, *GitHub*)
- Gamification
 - Von gesammelten *Koins* abhängige Levels einführen (z. B. bestimmte Fehlertypen erst ab fortgeschrittenem Level anzeigen, Avatars, Levelbezeichnungen)
 - Verschiedene Schwierigkeitsstufen
 - Einbindung in *Game Center* der jeweiligen Plattform
 - Weitere Badges einführen (viele Ideen finden sich hier: <https://wiki.openstreetmap.org/wiki/Badges>, z. B. auch Badges für Spielertypen)
 - Verschiedene Highscores anzeigen (z. B. zeitlich oder Regional begrenzt, nach Fehlertypen kategorisiert, schnellste Aufsteiger)
 - Zusätzliche Berechtigungen für erfahrene Benutzer (für den langfristigen Erfolg)
- Neue realistische Fehlertypen⁴
 - Hausnummern einfügen
 - Stockwerk-Anzahl einfügen
 - Einbahnstrassen erfassen
 - Öffnungszeiten von öffentlichen Gebäuden festhalten
- Weniger realistische Fehlertypen
 - Kreisel erfassen
 - Bushaltestellen von *DIDOK*⁵ erfassen
- Erkennen von Benutzern, die nicht sorgfältig validieren⁶
- Ausführliche Statistiken für individuelle Benutzer⁷
- Aufträge aus *wheelmap*⁸ einfügen

⁴<https://github.com/kort/kort/issues/81>

⁵<https://didok.osm.ch/>

⁶<https://github.com/kort/kort/issues/7>

⁷<https://github.com/kort/kort/issues/71>

⁸<http://wheelmap.org>

- Offline-Fähigkeit (offline Maps für *React Native* wären erforderlich)
- Wenn Aufträge z. B. drei Mal nicht gelöst werden können, soll eine *OSM*-Notiz generiert werden

Unrealistische Arbeiten

Folgende Punkte wurden in einem Meeting mit den Entwicklern und dem Betreuer (Prof. Stefan Keller) besprochen. Sie sind für KORT als weniger geeignet empfunden worden.

- Erweitern der Verifizierung mit der Möglichkeit, ein Foto als Beweis hochzuladen
 - *Begründung:* Aspekte des Datenschutzes bergen ein gewisses Risiko. Benutzer müssten für das Hochladen von Bildern zusätzliche Bedingungen akzeptieren.
- standortunabhängige Aufgaben lösen (Gefahr von Couch Mapping)
 - *Begründung:* Es ist ein Anliegen der *OSM* Community, dass die Mapper vor Ort sein sollen um Aufträge zu lösen.

11. Installation

In diesem Kapitel werden zwei Varianten beschrieben, wie die App installiert werden kann.

11.1. Installation mit Source Code

Um die App anhand des Source Codes zu installieren, muss die *React Native* Entwicklungsumgebung aufgesetzt sein. Wie diese eingerichtet wird, ist in der Getting-Started-Anleitung, der Onlinedokumentation von *React Native*¹, erklärt. Diese Anleitung erklärt Schritt für Schritt den Ablauf von der Einrichtung der Entwicklungsumgebung auf einem *Mac*-, *Linux*- oder *Windows*-Rechner (für *iOS* und *Android*) bis zum Starten der App.

Hier muss noch beachtet werden, dass die `SecretConfig.js`-Datei mit folgenden Werten ergänzt werden sollte. Der Dateipfad lautet: `/js/constants/SecretConfig.js`

- Mapbox Access Token
 - Dieses Token kann auf der Mapbox-Webseite² erstellt werden.
- Google Client ID
 - Wie die Google Client ID erhalten wird, ist in der Anleitung des *GitHub*-Projekts der Google-Signin-Library erklärt.³
 - Wichtig ist, dass anhand dieser Anleitung auch die `google-services.json`-Datei neu generiert und in das Projekt eingefügt werden muss.

11.2. Installation mit APK-Datei

Damit die *Android*-App von der bereitgestellten APK-Datei auf der CD installiert werden kann, muss das Smartphone mit dem Computer per USB-Kabel verbunden sein und erkannt werden.

1. Die APK-Datei in einen Ordner auf dem Gerätespeicher kopieren.

¹<https://facebook.github.io/react-native/docs/getting-started.html>

²<https://www.mapbox.com/help/create-api-access-token/>

³<https://www.mapbox.com/help/create-api-access-token/>

2. Die APK-Datei auf dem Smartphone auffinden und anklicken.
3. Dem App-Herausgeber vertrauen und Installation abschliessen.

Teil III.

Projektmanagement

12. Projektmanagement

Hier sind die Angaben zum Projekt, zu den eingesetzten Entwicklungswerkzeugen und zur eingesetzten Software zu finden.

- Website: <http://www.kort.ch/> (<http://play.kort.ch/>)
- Source Code: <https://github.com/kort/kort-reloaded/>
- Dokumentation: <https://github.com/kort/kort-reloaded-docu>
- Projektmanagement: Redmine (sinv-56059.edu.hsr.ch/redmine/projects/ba-kort/)
- Issues: <http://sinv-56059.edu.hsr.ch/redmine/projects/ba-kort/issues> (Backlog: im Wiki auf Redmine)

12.1. Team

- *Betreuer*: Prof. Stefan F. Keller
- *Projektpartner*: Liip AG, Limmatstrasse 183, CH-8005 Zürich
 - Jürg Hunziker
 - Stefan Oderbolz
- *Experte*: Claude Eisenhut
- *Gegenleser*: Prof. Beat Stettler

Autoren

Diese Arbeit wird als Bachelorarbeit an der Abteilung Informatik durchgeführt von

- Marino Melchiori
- Dominic Mülhaupt

Aufgabenbereiche

Jeweils zu Beginn eines Sprints wurden die Zuständigkeiten aufgeteilt, abgearbeitet und am Ende besprochen.

Einige Kernpunkte der Aufgabenbereiche von Marino Melchiori waren:

- OAuth
- GUI
- Map

Die Aufgabenbereiche von Dominic Mülhaupt waren:

- Daten-Layer
- Stores
- *Flux*
- Setup (*GitHub*, *Travis-CI*)

12.2. Risikomanagement

Da KORT bereits in einer vorhergehenden Bachelorarbeit erfolgreich umgesetzt werden konnte und Anklang gefunden hat, wurden bereits viele Risiken abgedeckt.

Dennoch wurde das Risikomanagement nicht vernachlässigt. Alle bekannten Risiken sind gesammelt aufgelistet und wurden nach jedem Meilenstein neu evaluiert.

12.2.1. Risikoanalyse

Am Anfang der Bachelorarbeit wurden folgende Risiken identifiziert:

R01: Mangelnde Erfahrung mit <i>JavaScript</i>, <i>React</i> und <i>React Native</i>	
Beschreibung	Wirkt sich negativ auf Design und Programmcode aus.
Schadenspotential	60h
Eintrittswahrsch.	65 %
Auswirkung	Da die Entwickler sich mit den zugrundeliegenden Programmierkonzepten vertraut machen während sie bereits planen und Entscheidungen treffen, teilweise sogar schon programmieren müssen, kann und wird es vorkommen, dass gewisse Entscheidungen schlecht getroffen und gewisse Konzepte nicht sauber umgesetzt werden. Dies kann zu Verspätungen oder unsauberem Programmcode führen.
Vorbeugung	Die Entwickler haben mit dem Betreuer (Prof. Stefan Keller) besprochen, dass das Minimum Viable Product der Bachelorarbeit eine Android App mit der gleichen Funktionalität, wie sie bereits im ursprünglichen Kort Game umgesetzt wurde, sein wird. Im Rahmen dieser Arbeit blieb aber schlicht keine Zeit, um sich umfassend in die Technologien einzuarbeiten, bevor die restliche Arbeit angepackt ist. Aus diesem Grund wird vor allem zu Beginn vermehrt auf Pair Programming gesetzt. Die Entwickler haben sich auch in regelmässigen Abständen die Zeit genommen, ein Code Review durchzuführen. Ausserdem ist es wichtig, dass bei Schwierigkeiten nicht zu lange gezögert wird, den Kontakt zu Experten im jeweiligen Bereich zu suchen.
Massnahmen beim Eintreffen	Komplexe Features vereinfachen und so gestalten, dass sie leicht erweiterbar sind.

Tabelle 12.1.: Risiko R01

R02: Es existiert keine passende Map Library für <i>React Native</i>	
Schadenspotential	70h
Eintrittswahrsch.	30 %
Auswirkung	Es müsste statt <i>React Native</i> ein alternatives mobiles Framework gefunden werden, welches die <i>OSM</i> -Daten anzeigen kann.
Vorbeugung	Zu Beginn des Projektes muss eine <i>React Native</i> Prototyp-Applikation implementiert werden, welche <i>OSM</i> -Daten auf der Karte darstellt.
Massnahmen beim Eintreffen	Auf eine aufwendigere Variante der Kartendarstellung, die im Kapitel Evaluation evaluiert wurde, zurückgreifen.

Tabelle 12.2.: Risiko R02

R03: <i>KeepRight</i> stellt den Dienst ein	
Schadenspotential	70h
Eintrittswahrsch.	10 %
Auswirkung	Die Anzahl der zur Verfügung stehenden Missionen würde stark eingeschränkt werden. Ausserdem wären diese auf die Schweiz beschränkt. Ein anderer Dienst (z. B. <i>Osmose</i>) müsste eingesetzt werden, was vor allem Änderungen im Backend erfordern würde.
Vorbeugung	Bereits früh im Projekt wird das Thema mit dem Projekt-partner besprochen, um festzustellen, ob dieser bereit wäre, sich dieser Problematik anzunehmen.
Massnahmen beim Eintreffen	Ausarbeitung einer neuen Aufgabenstellung mit dem Betreuer und den Projektpartnern (Jürg Hunziker und Stefan Oderbolz), die Änderungen am Backend beinhaltet.

Tabelle 12.3.: Risiko R03

R04: <i>React Native</i> ist noch nicht ausgereift genug für <i>Android</i>	
Schadenspotential	20h
Eintrittswahrsch.	30 %
Auswirkung	<i>Android</i> wird erst seit Oktober 2015 durch <i>React Native</i> unterstützt. Einige Features von Komponenten stehen nur für <i>iOS</i> zur Verfügung.
Vorbeugung	Bevor mit der Implementation begonnen wurde, sind die mangelnden Funktionalitäten (z. B. im Testing), so weit möglich, analysiert. Grundsätzlich sollte es möglich sein, die App für <i>Android</i> umzusetzen, da bereits komplexere Apps damit erstellt wurden. Wahrscheinlich wird es vorkommen, dass Workarounds nötig sein werden. Im schlimmsten Fall müsste während der Entwicklung auf <i>iOS</i> umgestellt werden, was grundsätzlich möglich ist.
Massnahmen beim Eintreffen	Ausarbeitung einer neuen Aufgabenstellung mit dem Betreuer, die eine <i>iOS</i> -Version bevorzugt.

Tabelle 12.4.: Risiko R04

R05: Mangelnde Erfahrung mit der flux-Architektur	
Beschreibung	Wirkt sich negativ auf Design und Programmcode aus.
Schadenspotential	30h
Eintrittswahrsch.	40 %
Auswirkung	Da die Konzepte der flux-Architektur neu erarbeitet werden müssen und flux in der Community als eher komplex eingestuft wird, kann die Entwicklung mehr Zeit beanspruchen.
Vorbeugung	Die korrekte Umsetzung wird bereits vor dem Start der Implementation so weit wie möglich analysiert. Dabei wurden die Themengebiete zum Einlesen aufgeteilt und schlussendlich besprochen und erklärt.
Massnahmen beim Eintreffen	Rücksprache mit dem IFS-Team, das an einem <i>React</i> -Projekt arbeitet.

Tabelle 12.5.: Risiko R05

R06: Mangelnde Erfahrung mit OAuth	
Beschreibung	Keiner der Entwickler ist mit OAuth vertraut. Dadurch, dass sich die Implementation der Authentifizierung bei einem native Client zu einer Implementation einer Web-App unterscheidet (Token- vs. Session-based), wird mehr Zeit beansprucht.
Schadenspotential	40h
Eintrittswahrsch.	60 %
Auswirkung	Die OSM-Community würde nur sehr ungern auf einen entsprechenden Login verzichten.
Vorbeugung	Mehr Aufwand in die Evaluation stecken.
Massnahmen beim Eintreffen	Rücksprache mit dem Betreuer.

Tabelle 12.6.: Risiko R06

Risikoanalyse MS2: Ende Elaboration – 29.03.2016

R01: Mangelnde Erfahrung mit JavaScript, React und React Native: Die Eintrittswahrscheinlichkeit konnte auf 50 % gesenkt werden.

R03: KeepRight stellt den Dienst ein: Wurde nach Absprache mit den Projektpartnern (Jürg Hunziker und Stefan Oderbolz) ganz eliminiert.

Risikoanalyse MS3: Evaluation der Komponenten – 08.04.2016

R02: Es existiert keine passende Map Library für React Native: Die Wahrscheinlichkeit des Eintretens konnte nach erfolgreicher Implementation auf 20 % gesenkt werden. Dadurch, dass noch keine Missionen gelöst wurden, galt das Risiko noch nicht als behoben.

Risikoanalyse MS4: Zwischenpräsentation – 22.04.2016

Das Eintreten von **R02** konnte weiterhin auf 10 % gesenkt werden. Die Marker waren klickbar und sie enthielten alle nötigen Informationen für das Lösen einer Mission.

Risikoanalyse MS5: Basis-Komponenten umgesetzt – 20.05.2016

R02 wurde eliminiert, **R01** auf 20 % gesenkt und **R05: Mangelnde Erfahrung mit der flux-Architektur** als tief eingeschätzt (10 %).

Risikoanalyse MS6: Beta-Release mit Grundfunktionalität – 06.06.2016

R05 wurde ebenfalls eliminiert – die flux-Architektur war stabil und eine Mission konnte erfolgreich gelöst werden.

Risikoanalyse MS7: Schlussabgabe – 17.06.2016

R04: *React Native* ist noch nicht ausgereift genug für *Android*: Wurde dank einer lauffähigen Version auf 10 % gesenkt.

R06: Mangelnde Erfahrung mit OAuth: Dies ist ein weiterhin bestehendes Risiko – die OSM-Authentifizierung konnte leider nicht umgesetzt werden.

12.3. Projektplan

Eine grobe Planung wurde in der zweiten Woche nach Projektbeginn festgehalten. Der erste Prototyp für die Demo an der Zwischenpräsentation war am 11.04.2016 geplant. Am 16.05. war dann das erste Release, mit dem Missionen gelöst werden können, vorgesehen. Um bei der [Abgabe der Bachelorarbeit](#) die App veröffentlichen zu können, entstand der folgende Projektplan.

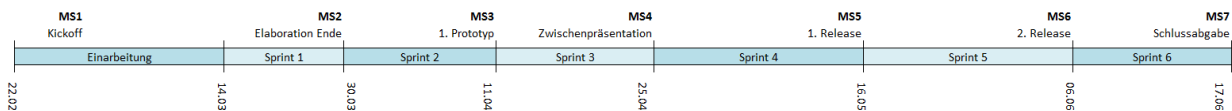


Abbildung 12.1.: Erste Grobfassung des Projektplans als Zeitstrahl dargestellt

Beim dritten Sprint wurde festgestellt, dass die Planung zu optimistisch und dass eine Veröffentlichung zum geplanten Zeitpunkt unrealistisch war. Die Planung wurde neu besprochen und in einem neuen [Zeitstrahl](#) festgehalten. Es gab Schwierigkeiten bei der Evaluation der Architektur und der Libraries. Die Libraries und Komponenten wurden in einzelnen Projekten getestet. Das hat uns zu einem besseren Verständnis für *React Native* und damit zu neuen Erkenntnissen für die [Architekturentscheidung](#) verholfen.

Erst beim [Meilenstein 5](#) konnte eine Version mit den definitiv evaluierten Komponenten entstehen. Beim achten Meeting (29.04.2016) wurde mit unserem Betreuer, Herr Prof. Stefan Keller, besprochen, dass wir das Projekt nach der Abgabe gerne weiterführen würden. Somit sollte es dann auch möglich sein, die App zu veröffentlichen.

Bis zum Ende vom [Meilenstein 6](#) entstand ein erstes Release, eines Prototyps mit der geplanten Grundfunktionalität. Bei der Abgabe ist das Projekt in einem Zustand, bei dem sich neue Features zügig implementieren lassen. Dieser aktuelle Stand zu Beginn des Projekts für den [Meilenstein 5](#) geplant gewesen.

Somit ist dieser neue Projektplan entstanden.

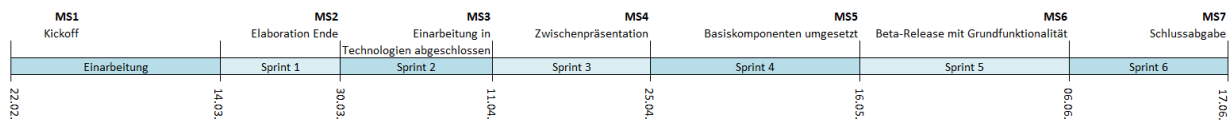


Abbildung 12.2.: Überarbeiteter Projektplan als Zeitstrahl dargestellt

12.4. Sprints

Durch die Anwendung von Scrum-Ansätzen wurden Sprints geplant. Eine Sprint-Periode dauerte immer bis zum Ende eines Meilensteins. Zu Beginn eines Sprints wurde das konkrete Vorgehen geplant und am jeweiligen Ende sind die Schwerpunkte dokumentiert worden.

12.4.1. Sprint 1

14.03.2016 bis 30.03.2016

Zum Schwerpunkt in diesem Sprint gehörte die Einarbeitung in die verwendeten Technologien und die Einarbeitung in das bestehende KORT-Projekt.

12.4.2. Sprint 2

30.03.2016 bis 11.04.2016

Im zweiten Sprint wurden die benötigten Libraries evaluiert und getestet. Die Details und die Begründungen zu den Entscheidungen wurden im [Kapitel Evaluation](#) festgehalten.

12.4.3. Sprint 3

11.04.2016 bis 25.04.2016

Neben der Vorbereitung der Zwischenpräsentation wurden die Anforderungen aktualisiert. Das Ersetzen der Validierung im Frontend kam neu dazu. Zukünftige Arbeiten und Ideen wurden ebenfalls gesammelt und im [Kapitel Realistische Arbeiten](#) dokumentiert.

12.4.4. Sprint 4

25.04.2016 bis 16.05.2016

Am Anfang des 4. Sprints wurde abgeklärt, ob ein Webprototyp mit *React* umsetzbar ist. Dies ist durchaus vorstellbar, allerdings nicht im Rahmen dieser Arbeit. Am Ende des Sprints wurde festgelegt, was getestet werden muss.

12.4.5. Sprint 5

16.05.2016 bis 06.06.2016

Während diesem Sprint fand das Meeting mit den Projektpartnern (Jürg Hunziker und Stefan Oderbolz) statt, um die Authentifizierung abzuklären. Es ging darum, neu zu evaluieren, wie das Login in der App umgesetzt wird und was dies für Änderungen am Backend mit sich bringen würde. Am Ende von Sprint 5 wurde ein Code Freeze bis zum Abgabetermin eingeleitet.

12.4.6. Sprint 6

06.06.2016 bis 17.06.2016

Die Validierungen konnten im Frontend anhand von Anpassung der Businesslogik abgeschafft werden. Ausserdem konnte die wichtige Internationalisierung für die Sprachen Englisch und Deutsch umgesetzt werden.

12.5. Meilensteine

In diesem Projekt wurde die Planung auf neun Meilensteine (MS) aufgeteilt. Die Meilensteine [MS 8](#) und [MS 9](#) finden nach der offiziellen Schlussabgabe statt und sind deswegen noch ohne Resultate dokumentiert.

12.5.1. MS1: Kickoff

Fällig am 25.02.2016

Resultate

- Kickoff Meeting bei Liip mit den Projektpartnern (Jürg Hunziker und Stefan Oderbolz) und dem Betreuer (Prof. Stefan Keller)

12.5.2. MS2: Ende Elaboration

Fällig am 29.03.2016

Resultate

- Infrastruktur aufgesetzt
 - Datenbank
 - [Continuous Integration](#)

- *UserVoice*
- Redmine
- Installationsskripte
- Dokumentation aufgesetzt
- Ausgangslage definiert und dokumentiert
- Anforderungsspezifikation erarbeitet
- Risikomanagement analysiert und dokumentiert
- Projektplan erarbeitet
- Map Komponente ausgewählt und eingesetzt
- Aufgabenstellung erarbeitet
- Testspezifikation erarbeitet
- Einarbeitung in die KORT-[Web-App](#)
- [GUI](#)-Mockups

Erledigte Arbeiten

Eine erste Version der Aufgabenstellung konnte erarbeitet werden. Die Dokumentation wurde eingeleitet. Es gelang uns einen automatisierten Build der App aus dem *GitHub*-Sourcecode mit *Travis CI* aufzusetzen. Für die Map-Komponente wurde die MapBox GL [Library](#)¹ mit den Vektor Daten von [osm2vectortiles](#)² evaluiert und getestet. Nebenbei konnten wichtige Erfahrungen in den verwendeten Technologien (*JavaScript*, *React* und *React Native*) gemacht werden. Zusätzlich wurde ein für *Android* angepasstes Design entworfen und Grundkonzepte der Architektur erarbeitet. Die Architektur ist noch nicht final. Sie erleichtert uns aber den Einstieg beim Programmieren. Die Infrastruktur ist soweit aufgesetzt.

Probleme

Für die detaillierte Erarbeitung der Testspezifikation fehlte noch die nötige Erfahrung in *React Native*.

12.5.3. MS3: Evaluation der Komponenten

Fällig am 08.04.2016

¹<https://libraries.io/npm/react-native-mapbox-gl>

²<http://osm2vectortiles.org/>

Resultate

- Einarbeitung in *JavaScript*, *React* und *React Native*
- *Android* Prototyp
 - Tab-Navigation
 - Darstellung der Karte
 - Evaluation der Architektur und Implementation des Grundgerüsts
 - Missionen auf Map anzeigen
 - Authentifizierung mit [OAuth](#) evaluiert
- *Travis-CI* Konfiguration aktualisieren

Erledigte Arbeiten

Die Entwicklungsumgebung wurde optimiert. Das *Travis*-Konfigurationsfile prüft den Code nun mit *ESLint*³ (*JavaScript* linter, prüft Styleguidelines) und *flow*⁴ (static type checker).

Für die Tab-Navigation konnte eine Demo mit *react-native-router-flux*⁵ umgesetzt werden. Diese muss noch in der KORT App implementiert werden.

[OAuth](#) (nur clientseitig, ohne Backend Kommunikation) wurde evaluiert und konnte dann anhand einer Demo getestet werden – allerdings nur mit *Facebook* und *Google*.

Die Darstellung der Karte aus [MS2: Ende Elaboration](#) wurde leicht ausgebaut. Neu wird nun der Standort des Benutzers ermittelt.

Die Missionen konnten im Umkreis von fünf Kilometern geladen werden.

Probleme

Schwierigkeiten traten vor allem im Zusammenhang mit *React Native* auf. Oft gab es Build-Fehler bei gleicher Code-Basis. Die Fehlerbehandlung hat uns enorm viel Zeit gekostet und es war schwer, im Internet Hilfestellungen zu erhalten. Da *React Native* alle zwei Wochen ein Update erhält, sind die Dokumentationen oder die Diskussionen im Internet teilweise schon wieder veraltet.

Im Austausch mit anderen *React Native* Entwicklern haben wir dasselbe Feedback erhalten. Der Prototyp konnte nicht wie geplant fertiggestellt werden und einige Arbeiten mussten auf den nächsten Meilenstein ausgelagert werden.

Die Authentifizierung mit dem Backend konnte noch nicht umgesetzt werden.

³<http://eslint.org/>

⁴<http://flowtype.org/>

⁵<https://github.com/aksonov/react-native-router-flux>

12.5.4. MS4: Zwischenpräsentation

Fällig am 22.04.2016

Resultate

- *Mapbox* Prototyp fertig
 - Missionen mit Marker auf Karte dargestellt
 - Tab-Navigation implementiert
- Zwischenpräsentation (Dauer ca. 30 Minuten)
 - Aufgabenstellung, Problembesprechung
 - IST-Situation
 - geplantes Resultat
 - Beschlussprotokoll für den [Betreuer](#)

Erledigte Arbeiten

Der Prototyp enthielt die Karte in einer Tab-Ansicht. An den jeweiligen Positionen der geladenen Missionen konnten Marker eingefügt werden. Durch einen Klick auf einen Marker konnte der Titel der Mission erfolgreich in die Konsole geloggt werden.

Leider konnte noch immer keine Lösung für die Implementation eines Logins mit [OAuth 1.0a](#) (*OSM*) gefunden werden. Dieses Feature noch zur Abgabe zu liefern wäre unrealistisch. Deswegen wurde es auf den [Meilenstein 9: Release für App Store](#) verschoben.

Probleme

Weiterhin traten Schwierigkeiten mit willkürlichen Fehlern der *React Native* App auf. Aus diesen Gründen konnte die Navigation und die Struktur der App nicht abschliessend umgesetzt werden. Dass der Benutzer nach dem Login automatisch zur Tab-Ansicht weitergeleitet wurde, hatte aufgrund eines Fehlers in der Navigations-[Library](#) nicht geklappt. Zwischenzeitlich wurde aus diesem Grund das Erhalten des Login-Tokens nach der Authentifizierung über *Google* und *Facebook* in einer separaten App getestet.

12.5.5. MS5: Basis-Komponenten umgesetzt

Fällig am 20.05.2016

Resultate

- Native Location Tracking implementiert
- **GUI** fertig entworfen
- Testing-**Framework** aufgesetzt
- Architektur-Entscheid und -Implementation
- Token basierte *Google*-Authentifizierung im Frontend umgesetzt, nachdem das Backend vom Projektpartner, Stefan Oderbolz und Jürg Hunziker, dafür angepasst wurde
- KORT-Datenbank für Tests lokal aufgesetzt
- KORT ist *iOS*-fähig
- *React-Web-App* evaluiert
- Validierungen vom Backend als Missionen laden
- Kurzvideo-Konzept besprochen

Erledigte Arbeiten

Wir haben die Architekturvarianten evaluiert und uns für die *Flux*-Architektur entschieden. Daraufhin wurde das bestehende – noch sehr schlanke – Grundgerüst, welches mit dem MVC-Pattern umgesetzt wurde, durch *Flux* ersetzt.

Unterdessen konnte das Grundgerüst des **GUI** grösstenteils fertiggestellt werden. Es fehlte noch die dynamische Behandlung von gewissen GUI-Komponenten. Zum Beispiel die Unterscheidung, ob beim Lösen einer Mission ein Picker zum Auswählen der Antwort gerendert wird, oder ob ein Text-Input-Feld angeboten wird.

KORT konnte erfolgreich auf *iOS* getestet werden.

Probleme

Weiterhin war es nicht möglich mit der Router-Flux-Navigationskomponente eine Weiterleitung nach erfolgreichem Login zur Kartenansicht umzusetzen. Hierbei handelte es sich um den bekannten Fehler dieser Komponente. Das verzögerte leider das geplante Ziel: eine Mission zu lösen.

Die KORT-Datenbank konnte nur bei einem Entwickler richtig aufgesetzt werden. Es gab Probleme mit der Installation der Scripts auf *OS X*. Aus diesem Grund wurde für Tests nach Absprache mit dem Projektpartner (Jürg Hunziker), das Development-KORT-Backend genutzt.

12.5.6. MS6: Beta-Release mit Grundfunktionalität

Fällig am 06.06.2016

Resultate

- dynamische [GUI](#) fertiggestellt
- Login Weiterleitung
- Login-Logik mit Local Storage
- Testing abgeschlossen
- Missionen und Validierungen sind lösbar
- Internationalisierung umgesetzt

Erledigte Arbeiten

Durch eine zusätzliche App-Loader-View konnte das Problem der Weiterleitung nach dem Login behoben werden. Diese Komponente lädt nun alles im Voraus und erst danach wird je nach Zustand, ob der Benutzer eingeloggt ist oder nicht, die entsprechende Ansicht angezeigt. Nach und nach wurden die Platzhalter der GUI mit dynamischen Daten der Businesslogik ersetzt. Die GUI ist nun dynamisch und zeigt je nach Zustand einer **Component** die entsprechende Oberfläche.

Schlussendlich konnten wir erfolgreich Missionen und Validierungen lösen.

Probleme

Im [Backend](#) sind Fehler aufgetreten wenn Validierungen negativ beantwortet wurden.

12.5.7. MS7: Schlussabgabe

Fällig am 17.06.2016

Resultate

- Gebundene, vollständige Dokumentation eingereicht
- CD mit Abgabe eingereicht
- Abstract eingereicht
- Poster eingereicht

12.5.8. MS8: Schlusspräsentation

Fällig am 29.06.2016

Ziele

- Aktualisierte Dokumentation
- Design Verbesserungen
- Präsentation fertiggestellt
- Handouts ausgedruckt

Resultate

- Dokumentation aktualisiert und neue CD gebrannt
- *iOS* App getestet

Probleme

Bei der *iOS* App funktioniert der Login mit *Google* noch nicht. Es liegt ein Problem mit dem erhaltenen Token von *Google* vor.

12.5.9. MS9: Release für App Store

Fällig am 05.07.2016

Ziele

- *Android* App im *Google Play* Store veröffentlicht

13. Projektmonitoring

13.1. Zeitanalyse

Für eine Bachelorarbeit werden 12 ECTS-Punkte vergeben, wobei ein Punkt einem Aufwand von 30 Stunden entspricht. In einem Team von zwei Entwicklern, entspricht dies 360 Stunden Aufwand pro Person (siehe Tabelle [13.1](#)).

Person	Aufwand
Marino Melchiori	461.5 h
Dominic Mülhaupt	467 h

Tabelle 13.1.: Arbeitsaufwand pro Person

Wenn die gleich folgenden Tabellen [13.2](#) und [13.3](#) verglichen werden, ist die Arbeitsaufteilung sehr gut erkennbar. Dies ist in den Unterschieden des Aufwandes in den Aktivitäten Analyse, Implementation, Dokumentation und Testing erkennbar. Bei neuen Erfahrungen haben sich die Entwickler ausgetauscht und waren so immer auf dem aktuellen Stand.

Marino Melchiori	
Aktivität	Aufwand
Analyse & Design	106.00 h
Implementation	125.00 h
Dokumentation	141.00 h
Requirements	30.75 h
Deployment	16.25 h
Allgemein	16.75 h
Projektmanagement	25.75 h

Tabelle 13.2.: Aufwand pro Aktivität (Marino Melchiori)

Dominic Mülhaupt	
Aktivität	Aufwand
Analyse & Design	49.50 h
Implementation	191.75 h
Dokumentation	66.25 h
Requirements	26.00 h
Deployment	7.00 h
Allgemein	52.50 h
Projektmanagement	52.00 h
Testing	22.00 h

Tabelle 13.3.: Aufwand pro Aktivität (Dominic Mülhaupt)

13.1.1. Soll-Ist-Zeitvergleich

Die Tabelle [13.4](#) zeigt, wie das Projekt kategorisiert wurde. In der KORT *Projekt*-Kategorie ging es vor allem um die Implementation der Architektur und die Verwendung und Einrichtung von *React Native*. Die Kategorien *Login*, *Missionen*, *Profil* und *Highscore* enthalten die Implementation des [GUI](#). *Map* und *Internationalisierung* sind Kategorien, bei denen es um die Installation der entsprechend verwendeten [Libraries](#) ging. In der Kategorie *Technologien* wurde die Einarbeitungszeit gebucht. Die *Qualitätssicherung* beinhaltet das Testing und die Kategorie *Allgemein* die Evaluation von verwendeten Konzepten und [Libraries](#).

Die Schätzung fand jeweils bei der Erstellung eines Tickets in einer Kategorie statt.

Die Differenz der *Kort Projekt*-Kategorie lässt sich durch die häufigen Build-Fehler erklären. Das Suchen nach Lösungen im Internet war sehr zeitaufwendig.

Bei der Kategorie *Missionen* wurden 15 Stunden zu viel geschätzt. In diesem Fall setzten wir vermehrt auf [Pair Programming](#), da es sich dort um eine der ersten umgesetzten Funktionen handelte. Somit konnte kostbare Zeit, die für das Refactoring geplant war, eingespart werden. Später folgende Features der Kategorien *Login*, *Profil* und *Highscore* wurden viel besser geschätzt.

Schlussendlich wurden 21 Stunden zu viel geschätzt, was bei einem Arbeitstag von 8 Stunden etwa 2.5 Arbeitstagen entspricht. Gerade für ein Projekt, bei dem es viele Unbekannten gab, da noch keine vorgängigen Erfahrungen mit den Technologien gesammelt wurden, ist es eher untypisch, dass die Soll-Ist-Differenz so klein ist. Wenn man die Arbeitspakete, für die zu wenig und diejenigen, für die zu viel Zeit eingeschätzt wurde, gruppiert ist die Differenz deutlich höher. Es ist eher Zufall, dass diese sich beinahe ausgleichen.

Soll-Ist-Vergleich vom Gesamtaufwand der Kategorien			
Kategorie	Soll-Aufwand	Ist-Aufwand	Differenz
Kort Projekt	71.75 h	88.25 h	+16.50 h
Login	91.50 h	92.50 h	+1.00 h
Map	47.75 h	45.50 h	-2.25 h
Missionen	59.75 h	44.75 h	-15.00 h
Highscore	11.50 h	12.70 h	+1,20 h
Profil	18.00 h	16.25 h	-1.75 h
Technologien	182.50 h	187.55 h	+0.05 h
Gamification und Design	7.25 h	4.75 h	-2.50 h
Qualitätssicherung	36.50 h	30.00 h	-6.50 h
Internationalisierung	4.50 h	4.25 h	-0.25 h
Dokumentation	204.00 h	198.25 h	-5.75 h
Allgemein	152.75 h	144.50 h	-8.25 h
Meeting	61.75 h	59.25 h	-2.50 h
Total	975.00 h	926.50 h	-21.00 h

Tabelle 13.4.: Soll-Ist-Vergleich: Gesamtaufwand pro Kategorie

13.2. Code-Statistik

Zur groben Einschätzung der Grösse vom Projekt wurde in der Tabelle 13.5 die Gesamtanzahl der *JavaScript* Codezeilen (ohne Code-Kommentare) gezählt. Beim Linken von [Libraries](#) gab es auch Anpassungen im *Android*- und *iOS*-Projekt. Der Umfang dieser Änderungen wurde nicht dokumentiert, da es sich vor allem um Anpassungen an vorhandenen Dateien handelte.

Sprache	Zeilen
<i>JavaScript</i>	3913

Tabelle 13.5.: Dateien und Codezeilen

Die Tabelle 13.6 zeigt die Anzahl an Codezeilen in einem Package. Die Packages **Actions**, **Data**, **Dispatcher**, **DTO** und **Stores** sind von der verwendeten Architektur gegeben. **Constants** beinhaltet Konstanten, IDs und sonstige fixe Parameter. Im Package **Components** befinden sich die **GUI**-Komponenten. Die **Shared Components** enthalten Buttons und weitere Komponenten, die oft wiederverwendet wurden.

Package	Zeilen
<i>Components</i>	1042
<i>Shared Components</i>	463
Total Components	1511
<i>Actions</i>	267
<i>Data</i>	513
<i>Dispatcher ohne Tests</i>	2
<i>Dispatcher Tests</i>	43
<i>DTOs ohne Tests</i>	199
<i>DTO Tests</i>	69
<i>Stores ohne Tests</i>	408
<i>Store Tests</i>	466
Total Logik ohne Tests	1389
<i>Constants</i>	441

Tabelle 13.6.: Codezeilen pro Package

Teil IV.

Anhänge

Glossar

API

Application Programming Interface, Schnittstelle für die Programmierung[21]. 27, 35

Backend

Als Backend wird das auf dem Server laufenden Programm bezeichnet. ii, 13, 14, 18, 25, 27, 35, 36, 57

Continuous Integration

Der Begriff *Continuous Integration*[19] beschreibt die Idee, dass Änderungen an einer Software schnell eingebracht werden sollen. Dazu zählt, dass diese in einem Tool zur Versionsverwaltung eingetragen und durch automatisierte Tests geprüft werden[21]. 31, 52

Crowdsourcing

Unter Crowdsourcing versteht man die Auslagerung von traditionell internen Teilaufgaben an eine Menge von freiwilligen Usern im Internet[27]. 5

DOM

DOM bedeutet Document Object Model und ist eine Spezifikation für den Zugriff auf HTML-Dokumente. Anhand des DOMs kann ein Computerprogramm den Inhalt der HTML-Elemente dynamisch verändern. Für einen Webentwickler wäre das der HTML-Code.[29]. 20

DTO

Ein Datentransferobjekt ist ein Objekt, das nur zur Übertragung von Daten oder Informationen dient und keine logischen Komponenten enthält. 25

Framework

Ein Framework ist noch kein fertiges Programm. Es ist ein Rahmen (ein Gerüst) welches der Programmierer verwenden kann um sein eigenes, persönliches Programm zu erstellen[30]. iv, 2, 32, 38, 47, 56

Frontend

Als Frontend wird das beim Client laufende Programm bezeichnet. [7](#), [26](#)

Funktionaler Test

Funktionale Tests überprüfen das Gesamtverhalten des Systems. Mit einem automatisierten UI Testing Tool werden verschiedene Szenarien durchlaufen, welche aus den Use Cases abgeleitet werden können. Funktionale Tests werden im Vergleich zu [Unit Tests](#) und [Integration Tests](#) eher spärlich eingesetzt. [32](#)

Gamification

Unter Gamification versteht man das Hinzufügen von Spielelementen in einen nicht spieltypischen Kontext.[\[22\]](#). [5](#), [38](#), [39](#)

GUI

GUI ist eine Abkürzung für graphical user interface und bedeutet grafische Benutzeroberfläche[\[31\]](#). [14](#), [21](#), [45](#), [53](#), [56](#), [57](#), [60](#), [61](#)

IDE

IDE bedeutet Integrierte Entwicklungsumgebung – auf Englisch integrated development environment[\[32\]](#). [31](#)

Integration Test

In Integration Tests wird die Zusammenarbeit von Systemteilen getestet. [32](#), [65](#)

JSX

JSX ist eine *JavaScript*-Syntax-Erweiterung die ähnlich wie XML aussieht[\[9\]](#). [20](#), [33](#)

Library

Eine Library (Programmbibliothek) bezeichnet in der Programmierung eine Sammlung von Unterprogrammen, die Lösungswege anbieten. Bibliotheken laufen im Unterschied zu Programmen nicht eigenständig, sondern sie enthalten Hilfsmodule, die angefordert werden können[\[35\]](#). [8](#), [9](#), [11](#), [13](#), [20](#), [33–35](#), [53](#), [55](#), [60](#), [61](#)

Mapper

Personen welche auf OpenStreetMap die Karten ergänzen und pflegen, nennen sich selbst *Mapper*[\[21\]](#). [ii](#)

Minimum Viable Product

Das Minimum Viable Product ist ein Produkt, welches gerade die Kernfunktionalität aufweist, die nötig ist um es zu veröffentlichen[33]. 46

Mock

Mock-Objekte werden in diversen Tests verwendet um das Verhalten eines anderen Objekts für spezifische Fälle zu simulieren ohne es tatsächlich zu implementieren. 32, 67

MVC

Der englischsprachige Begriff *model view controller* (MVC) ist ein Muster zur Strukturierung von Software-Entwicklung in die drei Einheiten Datenmodell (model), Präsentation (view) und Programmsteuerung (controller)[4]. 7, 8, 26

OAuth

OAuth ist ein offenes Protokoll, das eine standardisierte, sichere API-Autorisierung erlaubt[20]. 12, 13, 39, 45, 49, 54, 55

Packager

Der Packager kompiliert und bundelt den JavaScript Code für das Smartphone während der Entwicklungsphase und dem Testen auf dem Gerät. 24

Pair Programming

Bei Pair Programming handelt es sich um eine Arbeitsweise, bei der zwei Programmierer an einem Rechner arbeiten. Dabei ist ein Programmierer damit beschäftigt, den Code zu schreiben, während der andere über die Problemstellungen nachdenkt, den geschriebenen Code kontrolliert und Probleme, die ihm dabei auffallen, sofort anspricht[34]. 46, 60

POI

Abkürzung für Point of Interest. Dies ist ein allgemeiner Begriff für einen Ort mit irgendeiner Bedeutung, sei es eine Schule, Kirche, Bushaltestelle oder sonst etwas von besonderem Interesse[21]. 2

REST

Representational State Transfer[18] ist ein Programmierparadigma, welches besagt, dass sich der Zustand einer Webapplikation als Ressource in Form einer URL beschreiben lässt. Auf solche Ressourcen können folgende Befehle angewendet werden: GET, POST, PUT, PATCH, DELETE, HEAD und OPTIONS. HTTP ist ein Protokoll welches REST implementiert[21]. 25, 27, 33

Singleton

Das Singleton beschreibt ein Entwurfsmuster, wonach sichergestellt wird, dass von einer Klasse immer höchstens ein Objekt existiert, welches üblicherweise global verfügbar ist[38]. 26

Unit Test

Unit Tests werden geschrieben um kleine Teile des Codes – üblicherweise Module – zu testen. Da wirklich nur das Verhalten des entsprechenden Moduls getestet werden soll, müssen Abhängigkeiten dieses Moduls durch [Mocks](#) ersetzt werden. 32, 65

Virtual DOM

Das virtual DOM ist eine Abstraktion des DOM. Es ist eine lokale Kopie des DOM[23]. 20

Web-App

Der Begriff Web-App (von der englischen Kurzform für web application) bezeichnet im allgemeinen Sprachgebrauch Apps für mobile Endgeräte wie Smartphones und Tablet-Computer. Diese Apps laufen auf dem im Betriebssystem integrierten Browser, werden aus dem Internet geladen und können so ohne Installation auf dem mobilen Endgerät genutzt werden[28]. ii, 2, 5, 6, 10, 14, 21, 38, 49, 53, 56

Literaturverzeichnis

- [1] Christian Alfoni. Why we are doing MVC and FLUX wrong, 2015. [Online; Stand 15. Juni 2016]. URL: http://www.christianalfoni.com/articles/2015_08_02_Why-we-are-doing-MVC-and-FLUX-wrong.
- [2] Auth0. Authenticate React Native – iOS with Generic OAuth2 Provider, 2016. [Online; Stand 11. Juni 2016]. URL: <https://auth0.com/authenticate/react-native-android/oauth2>.
- [3] Jason Brown. Create a map with react-art, 2015. [Online; Stand 11. Juni 2016]. URL: <http://browniefed.com/blog/create-a-map-with-react-art/>.
- [4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (the Gang Of Four). *Design Patterns: Elements of Reusable Object-Oriented Software*. AddisonWesley Professional, 1994.
- [5] Facebook. flux, 2015. [Online; Stand 17. Juni 2016]. URL: <https://www.npmjs.com/package/flux>.
- [6] Facebook. Flux | Application Architecture for Building User Interfaces, 2015. [Online; Stand 14. Juni 2016]. URL: <https://facebook.github.io/flux/docs/overview.html>.
- [7] Facebook. Displaying Data. JSX Syntax, 2016. [Online; Stand 15. Juni 2016]. URL: <https://facebook.github.io/react/docs/displaying-data.html#jsx-syntax>.
- [8] Facebook. JavaScript Environment. JavaScript Runtime, 2016. [Online; Stand 15. Juni 2016]. URL: <http://facebook.github.io/react-native/releases/0.27/docs/javascript-environment.html#javascript-runtime>.
- [9] Facebook. Jsx in depth, 2016. [Online; Stand 13. Juni 2016]. URL: <https://facebook.github.io/react/docs/jsx-in-depth.html>.
- [10] Facebook. Mapview, 2016. [Online; Stand 12. Juni 2016]. URL: <http://facebook.github.io/react-native/releases/0.27/docs/mapview.html#annotations>.
- [11] Facebook. Multiple Components. Data Flow, 2016. [Online; Stand 15. Juni 2016]. URL: <https://facebook.github.io/react/docs/multiple-components.html#data-flow>.
- [12] Facebook. Native Modules. Android, 2016. [Online; Stand 15. Juni 2016]. URL: <http://facebook.github.io/react-native/releases/0.27/docs/native-modules-android.html>.

- [13] Facebook. Native Modules. iOS, 2016. [Online; Stand 15. Juni 2016]. URL: <http://facebook.github.io/react-native/releases/0.27/docs/native-modules-ios.html#native-modules>.
- [14] Facebook. Performance. JavaScript frame rate, 2016. [Online; Stand 15. Juni 2016]. URL: <https://facebook.github.io/react-native/docs/performance.html#javascript-frame-rate>.
- [15] Facebook. React native, 2016. [Online; Stand 13. Juni 2016]. URL: <https://facebook.github.io/react-native/>.
- [16] Facebook. React Native, 2016. [Online; Stand 15. Juni 2016]. URL: <https://code.facebook.com/projects/450791118411445/react-native/>.
- [17] Facebook. The virtual dom, 2016. [Online; Stand 12. Juni 2016]. URL: <https://facebook.github.io/react/docs/working-with-the-browser.html#the-virtual-dom>.
- [18] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. URL: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
- [19] Martin Fowler. Continuous Integration, 2006. [Online; Stand 9. Dezember 2012]. URL: <http://www.martinfowler.com/articles/continuousIntegration.html>.
- [20] Ed D. Hardt. The OAuth 2.0 Authorization Framework, 2012. [Online; Stand 6. Dezember 2012]. URL: <http://tools.ietf.org/html/rfc6749>.
- [21] Stefan Oderbolz Jürg Hunziker. Gamified Mobile App für die Verbesserung von OpenStreetMap. HSR Hochschule für Technik Rapperswil, 2012.
- [22] Kevin Werbach, Dan Hunter. *For the Win: How Game Thinking Can Revolutionize Your Business*. Wharton Digital Press, 2012.
- [23] Bartosz Krajka. The difference between Virtual DOM and DOM, 2015. [Online; Stand 12. Juni 2016]. URL: <http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/>.
- [24] Mapbox. Plans & pricing, 2016. [Online; Stand 11. Juni 2016]. URL: <https://www.mapbox.com/pricing/>.
- [25] Microsoft. React Native on the Universal Windows Platform, 2016. [Online; Stand 15. Juni 2016]. URL: <https://blogs.windows.com/buildingapps/2016/04/13/react-native-on-the-universal-windows-platform/>.
- [26] Bobby Sudekum. React native mapbox gl, 2015. [Online; Stand 11. Juni 2016]. URL: <https://github.com/mapbox/react-native-mapbox-gl#react-native-mapbox-gl>.
- [27] Wikipedia. Crowdsourcing – Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 19. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Crowdsourcing&oldid=111553592>.

- [28] Wikipedia. Webapp – Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 6. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Webapp&oldid=102886692>.
- [29] Wikipedia. Document Object Model – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 12. Juni 2016]. URL: https://de.wikipedia.org/w/index.php?title=Document_Object_Model&oldid=151802596.
- [30] Wikipedia. Framework – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 10. Juni 2016]. URL: <https://de.wikipedia.org/w/index.php?title=Framework&oldid=153658347>.
- [31] Wikipedia. Grafische Benutzeroberfläche – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 12. Juni 2016]. URL: https://de.wikipedia.org/w/index.php?title=Grafische_Benutzeroberfl%C3%A4che&oldid=154152657.
- [32] Wikipedia. Integrierte Entwicklungsumgebung – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 14. Juni 2016]. URL: https://de.wikipedia.org/w/index.php?title=Integrierte_Entwicklungsumgebung&oldid=153872017.
- [33] Wikipedia. Minimum viable product - Wikipedia, the free encyclopedia, 2016. [Online; Stand 17. März 2016]. URL: https://en.wikipedia.org/w/index.php?title=Minimum_viable_product&oldid=706716563.
- [34] Wikipedia. Paarprogrammierung – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 17. März 2016]. URL: <https://de.wikipedia.org/w/index.php?title=Paarprogrammierung&oldid=150084864>.
- [35] Wikipedia. Programmbibliothek – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 12. Juni 2016]. URL: <https://de.wikipedia.org/w/index.php?title=Programmbibliothek&oldid=152936522>.
- [36] Wikipedia. React (JavaScript library) – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 15. Juni 2016]. URL: [https://en.wikipedia.org/w/index.php?title=React_\(JavaScript_library\)&oldid=725306667](https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=725306667).
- [37] Wikipedia. React (JavaScript library) – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 12. Juni 2016]. URL: [https://en.wikipedia.org/w/index.php?title=React_\(JavaScript_library\)&oldid=723818681](https://en.wikipedia.org/w/index.php?title=React_(JavaScript_library)&oldid=723818681).
- [38] Wikipedia. Singleton (Entwurfsmuster) – Wikipedia, Die freie Enzyklopädie, 2016. [Online; Stand 14. Juni 2016]. URL: [https://de.wikipedia.org/w/index.php?title=Singleton_\(Entwurfsmuster\)&oldid=154168600](https://de.wikipedia.org/w/index.php?title=Singleton_(Entwurfsmuster)&oldid=154168600).

Abbildungsverzeichnis

7.1. Modellierung der DTOs	25
7.2. Idee der <i>Flux</i> -Architektur	26
7.3. Vollständiges <i>Flux</i> -Diagramm[5]	28
7.4. Bevor Daten geladen und Informationen dargestellt werden können, muss der Benutzer autorisiert und lokalisiert sein.	29
7.5. Lösen einer Mission und Anzeigen der Belohnung	30
12.1. Erste Grobfassung des Projektplans als Zeitstrahl dargestellt	50
12.2. Überarbeiteter Projektplan als Zeitstrahl dargestellt	51

Tabellenverzeichnis

3.1. Bewertung Komponente für die Navigation	8
3.2. Bewertung Navigations-Komponente	9
3.3. Bewertung Map-Komponente	11
3.4. Bewertung OAuth-Komponente	12
9.1. Abhängigkeiten im <i>JavaScript</i> -Code	34
9.2. Abhängigkeiten für die Entwicklung	34
12.1. Risiko R01	46
12.2. Risiko R02	47
12.3. Risiko R03	47
12.4. Risiko R04	48
12.5. Risiko R05	48
12.6. Risiko R06	49
13.1. Arbeitsaufwand pro Person	59
13.2. Aufwand pro Aktivität (Marino Melchiori)	59
13.3. Aufwand pro Aktivität (Dominic Mülhaupt)	60
13.4. Soll-Ist-Vergleich: Gesamtaufwand pro Kategorie	61
13.5. Dateien und Codezeilen	61
13.6. Codezeilen pro Package	62

Inhalt der CD

