

Gamified Mobile App für die Verbesserung von OpenStreetMap

Bachelorarbeit

Abteilung Informatik
HSR Hochschule für Technik Rapperswil

Herbstsemester 2012/13

Autoren: **Jürg Hunziker**
Stefan Oderbolz
Betreuer: **Prof. Stefan Keller**
Projektpartner: **Reto Senn, bitforge AG Zürich**
Experte: **Claude Eisenhut**
Datum: **21. Dezember 2012**

Impressum und Revision

Impressum

Autoren:	Jürg Hunziker (jhunzike@hsr.ch) Stefan Oderbolz (soderbol@hsr.ch)
Dokument erstellt:	05.10.2012
Letzte Aktualisierung:	20.12.2012

Dieses Dokument wurde mit L^AT_EX erstellt.

Änderungsverlauf

Datum	Änderungen	Bearbeiter
05.10.2012	Dokumententwurf erstellt	Stefan Oderbolz
10.10.2012	Leaflet Komponente dokumentiert	Jürg Hunziker
01.12.2012	Abstract erstellt	Jürg Hunziker
02.12.2012	Administration dokumentiert	Jürg Hunziker
05.12.2012	Server-Setup hinzugefügt	Stefan Oderbolz
06.12.2012	Entwurf des Management Summary	Stefan Oderbolz
06.12.2012	Frontend dokumentiert	Jürg Hunziker
08.12.2012	Projektmanagement dokumentiert	Jürg Hunziker
09.12.2012	Backend dokumentiert	Stefan Oderbolz
10.12.2012	Management Summary geschrieben	Stefan Oderbolz
10.12.2012	Gamification beschrieben Konfiguration des Frontends beschrieben	Jürg Hunziker
12.12.2012	Architektur dokumentiert OAuth dokumentiert	Stefan Oderbolz
13.12.2012 - 20.12.2012	Dokumentation finalisiert	Jürg Hunziker Stefan Oderbolz

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Ort, Datum:

Ort, Datum:

Name, Unterschrift:

Name, Unterschrift:

Abstract

OpenStreetMap ist ein freies Projekt, welches jedermann ermöglicht, Kartendaten zu nutzen und zu editieren. Durch diesen öffentlichen Charakter ist es nicht ausgeschlossen, dass fehlerhafte bzw. unvollständige Daten eingetragen werden. Es gibt verschiedene Tools, die es sich zum Ziel gesetzt haben, solche Fehler zu finden und aufzubereiten. Die so aufbereiteten Daten mussten bisher anschliessend manuell verglichen und korrigiert werden.

Zur Behebung dieser Fehler ist die cross-platform [Web-App](#) KORT entwickelt worden. Diese ist in JavaScript geschrieben und basiert auf dem *Sencha Touch 2*-Framework. Im Backend kommt eine *PostgreSQL*-Datenbank zum Einsatz. Die komplette Kommunikation ist mit [REST](#)-Schnittstellen realisiert.

Mit KORT soll das Verbessern von Karten-Daten auf unterhaltsame Weise ermöglicht werden. Dem Benutzer werden dazu die *OpenStreetMap*-Fehler auf einer Karte angezeigt. Falls er die Lösung für einen dieser Fehler kennt, kann er einen entsprechenden Lösungsvorschlag abgeben. Weitere Spieler der App überprüfen daraufhin den Vorschlag auf seine Gültigkeit. Um die Benutzer zu motivieren, die App regelmässig zu verwenden, wurden zahlreiche Spiel-Elemente eingesetzt. So erhält ein Benutzer für alle Aktionen Punkte (sogenannte *Koins*), die dann in einer Rangliste zum Vergleich dargestellt werden. Zudem können Auszeichnungen für besondere Leistungen gewonnen werden. Dieser Ansatz ist bekannt unter dem Begriff [*Gamification*](#).

Management Summary

Ausgangslage

Das *OpenStreetMap*-Projekt beinhaltet eine sehr grosse Menge an Daten, welche frei zugänglich ist. 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 direkt oder indirekt die *OpenStreetMap*-Karte verändern und ergänzen. Die Qualitätssicherungstools haben sich zum Ziel gesetzt, 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 (Abbildung 0.1) berechnen aus den Karten-Rohdaten die vorhandenen Fehler. Dazu werden einige Heuristiken verwendet oder einfache Plausibilitätsprüfungen durchgeführt. Typische Fehler aus diesen Quellen sind **POIs** ohne Namen oder Straßen ohne definierte Geschwindigkeitslimiten.

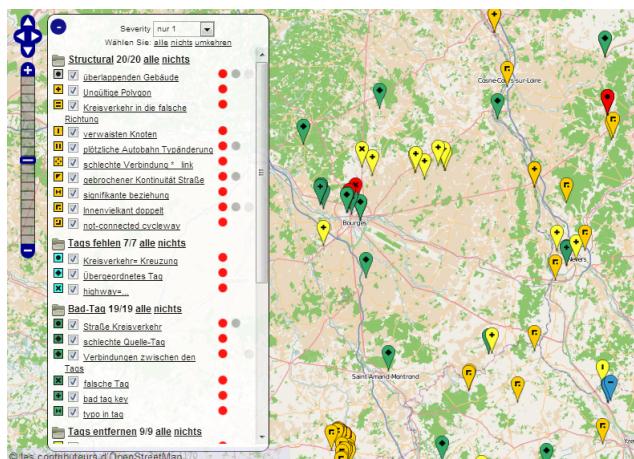


Abbildung 0.1.: Anzeige von Fehlern in Osmose

Andere Lösungen setzen darauf, dass Fehler manuell von einer Person erfasst werden. Dadurch ist es möglich, den Fehler detaillierter zu beschreiben. So gibt es Navigationsgeräte-Hersteller,

¹<http://keepright.ipax.at/>

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

welche die Daten aus *OpenStreetMap* zur Routenberechnung verwenden, und dabei ihren Benutzern ermöglichen, falsche Routen zu melden.

Falls ein **Mapper** einen Fehler melden will, so kann er diesen direkt in den Metadaten der Karte hinterlegen. Alle Objekte auf der Karte (Punkte, Wege und Relationen) können durch beliebige, sogenannte **Tags** ergänzt werden. Um auf einem Objekt Fehler zu markieren, hat sich die Community darauf geeinigt, dafür **FIXME-Tags** zu verwenden.

Quellen für derartige Fehler sind beispielsweise *OpenStreetBugs*³ oder die **FIXME-Tags** aus *KeepRight*.

Ergebnisse

Das Ziel, eine plattformübergreifende **Web-App** zu erstellen mit geeignetem Backend für die Verwaltung der Daten, wurde klar erreicht. Die App KORT bietet alle Grundfunktionalitäten, welche nötig sind, um Fehler anzuzeigen, zu lösen und gegebene Vorschläge zu validieren. Daneben sind einige Elemente der **Gamification** implementiert. So ist es beispielsweise möglich, durch das Lösen von Fehlern Punkte (sogenannte *Koins*) zu sammeln und sich über eine Highscore mit anderen Spielern zu messen. Zudem gibt es für speziellen Einsatz Auszeichnungen zu gewinnen.



Abbildung 0.2.: Auszeichnungen in KORT

Der Bereich der **Gamification** ist aber sehr offen und lässt Raum für viele weitere Konzepte. Schlussendlich ist jedoch klar, dass KORT noch ein Stück davon entfernt ist, ein „echtes“ Game zu sein.

Das Themengebiet der **Gamification** bietet viele Ansätze, um repetitive Aufgaben spannend zu gestalten. Gerade bei der Qualitätssicherung von *OpenStreetMap* fallen viele solcher Aufgaben an. Um die Wahrscheinlichkeit zu erhöhen, dass diese gelöst werden, braucht es neue Werkzeuge, die einfach zu bedienen sind und Spass machen.

Frontend

Das Frontend basiert auf dem HTML5/JavaScript Framework *Sencha Touch 2* und orientiert sich vom Look and Feel an einer iPhone-App. Die App bietet für die verschiedenen Funktionen unterschiedliche Tabs an, in welchen der Benutzer Fehler beheben oder prüfen und die

³<http://openstreetbugs.schokokeks.org/>

Rangliste oder sein Profil anschauen kann.

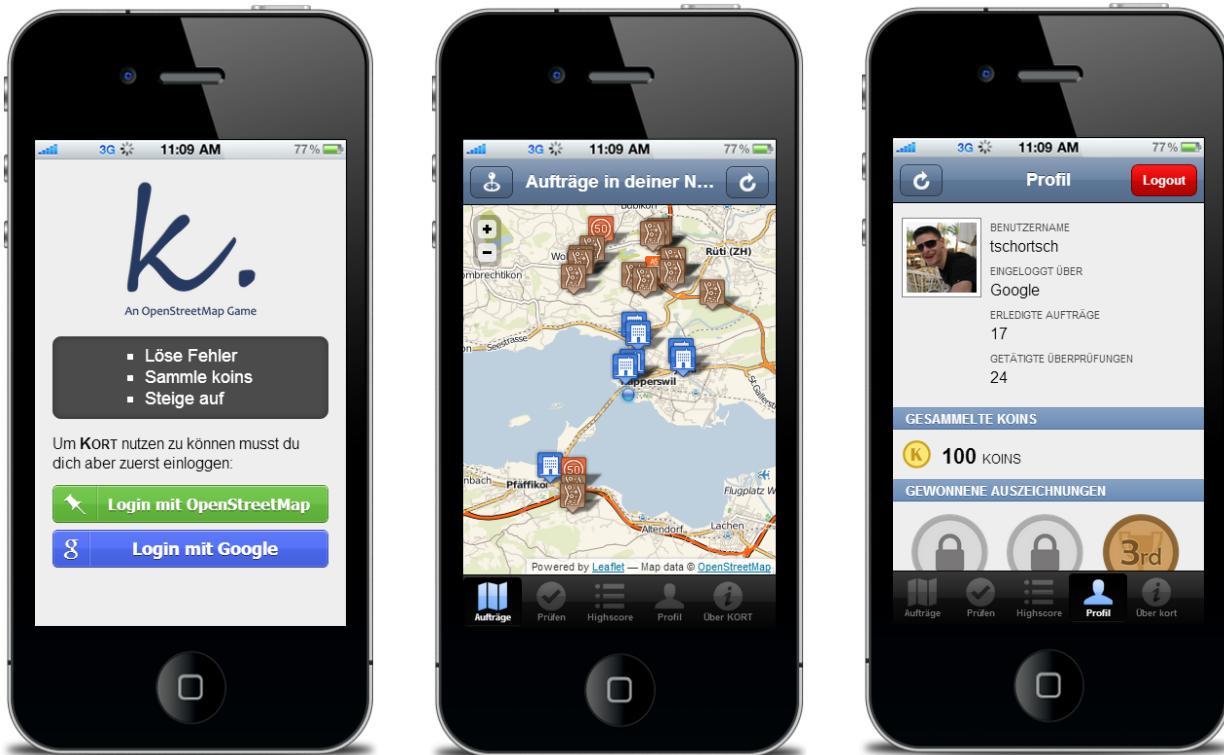


Abbildung 0.3.: KORT App

Für die Authentifizierung kommt [OAuth](#) zum Einsatz. Der Standard sorgt dafür, dass sich der Benutzer mit einem bereits bestehenden Benutzerkonto bei KORT anmelden kann. Derzeit werden dafür die [OAuth](#)-Provider *OpenStreetMap* und *Google* verwendet.

Backend

Das Backend ist in PHP geschrieben und basiert auf der Kommunikation über [REST](#)-Schnittstellen. Die eigenen Schnittstellen sind alle mit dem *Slim-Framework*⁴ erstellt. Da die Datenbank und der Webserver auf zwei verschiedenen Servern laufen, bietet auch die Datenbank eine [REST](#)-Schnittstelle an, über welche sich beliebige SQL-Abfragen absetzen lassen. Diese flexible Aufteilung ermöglicht es sehr einfach, die Systeme zu migrieren oder weitere hinzuzufügen.

⁴<http://www.slimframework.com/>

Ausblick

Die App hat grosses Potential, Personen, welche sich bislang nicht mit der Thematik *OpenStreetMap* befasst haben, für das Projekt zu begeistern.

Ganz allgemein ist es wichtig, die Robustheit und Geschwindigkeit der App zu verbessern. Gerade weil es sich um eine [Web-App](#) handelt, ist dies besonders kritisch. Im Idealfall sollte sich die App nicht von einer nativen App unterscheiden. Es ist zu prüfen, ob es sich lohnt, die App native zu builden⁵. Dies hätte den Vorteil, dass die App über die bekannten [App-Stores](#) zum User gebracht werden kann.

Beim Login wäre es wünschenswert, noch weitere [OAuth](#)-Dienste anzubieten, um so weitere Personen anzusprechen und die Akzeptanz zu steigern. Schliesslich verfügt nicht jeder über ein *OpenStreetMap*- oder ein Google-Konto. Mögliche Dienste sind beispielsweise Facebook oder Twitter, welche beide [OAuth](#) anbieten.

Um die Validierung von Lösungsvorschlägen zu verbessern, wäre es wünschenswert wenn ein Benutzer direkt einen Fotobeweis anbringen könnte. Dies hilft die Datenqualität hoch zu halten und macht eine Änderung leichter überprüfbar. Der Benutzer könnte dafür beispielsweise mit zusätzlichen Punkten belohnt werden. Technisch wäre es sicher spannend sich mit dem [Camera API](#) auseinander zu setzen.

Als zusätzliche Motivation könnten zeitlich begrenzte Aktionen durchgeführt werden. Dies soll Benutzer dazu animieren, die App immer wieder zu verwenden. Mögliche Aktionen wären beispielsweise die Konzentration auf einen Fehlertyp („*Gib allen Restaurants in deiner Umgebung einen Namen und erhalte diese Woche die spezielle Restaurant-Auszeichnung*“) oder auf eine Region („*Korrigiere jeden Tag im Dezember Fehler in Zürich und erhalte die Zürich-Silvester-Auszeichnung*“). Die Spieler könnten beispielsweise mittels Push-Notifikationen über solche Aktionen oder sonstige Neuerungen informiert werden.

Um die App längerfristig am Laufen zu halten, ist es unumgänglich, weitere Fehlertypen und -quellen einzubinden. *KeepRight* bietet zwar eine grosse Menge an Fehlerdaten, jedoch ist nur ein geringer Teil davon für unsere App nutzbar.

Die Bekanntheit der App muss durch geeignete Massnahmen gesteigert werden. Dazu gehört die Integration von Social Media-Diensten wie Facebook und Twitter. Diese kann einerseits genutzt werden, um Werbung zu machen, anderseits können Benutzer auf bereits gewonnene Auszeichnungen aufmerksam gemacht werden und so dazu animieren, diese Auszeichnungen ebenfalls zu gewinnen. Ähnlich wie man es von anderen Apps wie beispielsweise *Foursquare*⁶ oder *SBB.Connect*⁷ kennt.

Die wichtigste, noch offene Erweiterung stellt aber das noch fehlende Zurückschreiben der eingetragenen Lösungen zu *OpenStreetMap* dar. Diese Funktionalität wirkt sich stark auf die Akzeptanz der App seitens der *OpenStreetMap*-Community aus.

⁵z.B. mit Apache Cordova oder Sencha Packager

⁶<https://foursquare.com/>

⁷<http://www.sbb.ch/fahrplan/mobile-fahrplaene/mobile-apps/sbb-connect.html>

Inhaltsverzeichnis

I. Einführung	1
1. Informationen zum Projekt	2
1.1. Problemstellung	2
1.2. Aufgabenstellung	2
1.3. Ziele	3
1.4. Rahmenbedingungen	3
1.5. Aufbau der Arbeit	4
2. Umsetzung	5
2.1. Stand der Technik	5
2.2. Vision	6
2.3. Resultate der Arbeit	6
2.4. Schlussfolgerungen und Ausblick	7
3. Begriffsdefinitionen	8
3.1. Name der Applikation	8
3.2. Begriffe aus dem Spiel	8
II. Projektdokumentation	9
4. Kort	10
4.1. Analyse	11
4.1.1. User Szenarien	11
4.1.2. Paper-Prototype	13
4.2. Design	16
4.2.1. Package-Struktur	16
4.2.2. Controller-Package	17
4.2.3. Store- und Model-Package	18
4.2.4. Aufbau der Benutzeroberfläche	20
4.3. Implementation	21
4.3.1. Starten der App	21
4.3.2. Sencha Cmd	23
4.3.3. Cross-platform Unterstützung	24
4.3.4. Internationalisierung	24

4.4.	Resultate	25
4.4.1.	Maske: Aufträge	25
4.4.2.	Maske: Prüfen	26
4.4.3.	Maske: Highscore	27
4.4.4.	Maske: Profil	27
4.4.5.	Maske: Über Kort	27
4.5.	Dokumentation	28
4.6.	Bekannte Fehler	29
4.6.1.	iOS6 - Add to homescreen	29
4.6.2.	App Build	30
5.	OpenStreetMap-Daten in Sencha Touch	31
5.1.	Sencha Touch Map-Komponente	31
5.2.	Leaflet Map-Komponente	31
5.2.1.	Verfügbarkeit	32
5.2.2.	Dokumentation	33
6.	Gamification	34
6.1.	Gamification in Kort	34
6.1.1.	Sprache	34
6.1.2.	Punktesystem „Koins“	35
6.1.3.	Auszeichnungen	35
6.1.4.	Highscore	36
6.2.	Weitere mögliche Elemente	36
6.2.1.	Erste Schritte	36
6.2.2.	Zeitlich begrenzte Aktionen	36
6.2.3.	Weitere Auszeichnungen hinzufügen	36
6.2.4.	Verschiedene Highscores	37
6.2.5.	Erfahrene Spieler	37
6.2.6.	Einbinden in Apple Game Center	38
6.2.7.	Design	38
6.2.8.	Gamification von OpenStreetMap	38
7.	Fehler-Datenquellen	40
7.1.	Evaluation von geeigneten Datenquellen	40
7.1.1.	Fehler aus OpenStreetMap extrahieren	40
7.1.2.	FIXME-Tags in OpenStreetMap	41
7.1.3.	OpenStreetBugs	41
7.1.4.	KeepRight	42
7.1.5.	MapDust	42
7.1.6.	Housenumbervalidator	43
7.2.	Entscheid: Fehlerdaten von KeepRight	43
7.3.	Fehlerdaten in die Datenbank laden	44
7.3.1.	Datenformat	44
7.3.2.	Internationalisierung	45

8. Architektur	46
8.1. Bootstrapping	48
8.2. Umsysteme	48
8.2.1. KeepRight	48
8.2.2. OpenStreetMap	49
8.3. Authentifizierung	49
8.4. REST	50
9. Infrastruktur	51
9.1. Datenbankserver	51
9.1.1. Datenbank-Webservice	51
9.1.2. Redmine	52
9.2. Webserver (Heroku)	52
9.3. Deployment	53
9.3.1. Travis CI	53
9.3.2. Konfiguration über <code>.travis.yml</code>	54
9.3.3. Apache Ant	56
10. Backend	57
10.1. Implementation	57
10.1.1. Gliederung	57
10.1.2. Abhängigkeiten	58
10.2. REST-Schnittstellen	58
10.2.1. Slim Framework	58
10.2.2. Webservice: Antworten <code>/answer</code>	59
10.2.3. Webservice: Fehler <code>/bug</code>	59
10.2.4. Webservice: Highscore <code>/highscore</code>	62
10.2.5. Webservice: OpenStreetMap <code>/osm</code>	63
10.2.6. Webservice: Benutzer <code>/user</code>	64
10.2.7. Webservice: Überprüfung <code>/validation</code>	67
10.2.8. Webservice: Datenbank <code>/db</code>	69
10.3. Datenbank	74
10.3.1. Views	74
10.3.2. Applikationsschema	74
10.3.3. Schema für Fehlerquellen	74
10.3.4. Transaktionen	75
10.4. Sicherheit	75
10.4.1. Authentifizierung mit OAuth	75
10.4.2. Übertragungssicherheit	79
10.5. Testing	79
10.5.1. SimpleTest	79
10.5.2. Integrationstests der Webservices mit QUnit	81
10.6. Dokumentation	81
11. Administration	83
11.1. Hinzufügen von Fehler-Datenquellen	83

11.2. Hinzufügen von Fehlertypen	84
11.2.1. Fehlertypen	84
11.2.2. View-Typen	85
11.3. Hinzufügen von Auszeichnungen	86
11.4. Hinzufügen von OAuth Anbietern	87
11.5. Reset der Applikation	88
11.6. Frontend Konfiguration	89
III. Projektmanagement und -monitoring	90
12. Projektmanagement	91
12.1. Sprint 1	92
12.2. Sprint 2	93
12.3. Sprint 3	95
12.4. Sprint 4	96
12.5. Sprint 5	97
13. Projektmonitoring	99
13.1. Meilensteine	99
13.2. Risikomanagement	100
13.2.1. Technische Risiken	100
13.2.2. Fachliche Risiken	101
13.3. Projektverlauf	102
13.4. Arbeitsaufwand	102
13.5. Fazit	102
IV. Anhänge	104
Inhalt der CD	105
Glossar	106
Literaturverzeichnis	109
Abbildungsverzeichnis	111
Tabellenverzeichnis	113

Teil I.

Einführung

1. Informationen zum Projekt

1.1. Problemstellung

OpenStreetMap (OSM) ist ein freies Projekt, welches jedermann ermöglicht, Kartendaten anzuzeigen und zu editieren. Durch diesen öffentlichen Charakter ist es nicht ausgeschlossen, dass fehlerhafte bzw. unvollständige Daten eingetragen werden.

Aus diesem Grund entstand die Idee, mit einer mobilen Applikation eine Möglichkeit anzubieten, Fehler auf einfache Weise anzuzeigen und zu korrigieren. Um die Motivation für die Verwendung der App längerfristig aufrecht zu erhalten, soll diese mit Spiel-Elementen versehen werden.

1.2. Aufgabenstellung

Im Rahmen dieser Arbeit wird eine HTML5 [Web-App](#) entwickelt, welche es ermöglicht, unvollständige bzw. fehlerhafte Daten in der *OpenStreetMap*-Datenbank zu korrigieren oder zu vervollständigen.

Die App soll nicht als herkömmlicher Editor implementiert werden, sondern einen gewissen Gamecharakter aufweisen. Dieser zeichnet sich dadurch aus, dass die Benutzer für ihre Änderungsvorschläge belohnt werden. So können sie beispielsweise in der Rangliste (Highscore) aufsteigen oder Auszeichnungen (Badges) gewinnen. Dieses Konzept ist unter dem Stichwort [Gamification](#) bekannt. Damit ist allgemein die Anwendung von Spiel-Elementen in einem spielfremden Kontext gemeint. Es soll untersucht werden, ob und wie sich dies für *OpenStreetMap* umsetzen lässt.

Eingetragene Änderungsvorschläge können anschliessend von weiteren Benutzern kontrolliert und bewertet werden. Mehrfach validierte Änderungen sollen ins *OpenStreetMap*-Projekt zurückgeführt werden.

Mit der Integration von Social Media-Diensten (Facebook, Twitter) soll die Bekanntheit der App gefördert und die Motivation der Benutzer gesteigert werden.

1.3. Ziele

In der Aufgabenstellung der Arbeit wurden folgende Ziele definiert:

- Erstellen einer cross-platform HTML5 [Web-App](#) mit JavaScript
- Einsatz von JavaScript-[APIs](#) zur Ansteuerung von Hardwarekomponenten (z.B. GPS, Kamera)
- Es soll geprüft werden, ob die [Web-App](#) auch als native App für die Plattformen iOS und Android zur Verfügung gestellt werden kann
- Als Basis sollen Daten und Webdienste des *OpenStreetMap*-Projekts verwendet werden
 - Quellen für bekannte Fehler: *OpenStreetMap* Bug Reports, [FIXME](#)- und [TODO](#)-Tags
 - Unterstützung von [POI](#)-, Linestring- und Polygon-Objekten
- Verwendung einer vorhandenen User-Basis für die Authentifizierung ([OAuth](#))
- Integration von Social Media zum Austausch von Aktivitäten
- Konzept für Gamification von *OpenStreetMap* erarbeiten
 - Highscores / Rankings
 - Badges / Achievements
 - Aufgaben mit verschiedenen Schwierigkeitsstufen
- Verschiedene Modi
 - Erfassen von Daten, Aufnahme von Fotos (ortsbezogen)
 - Verifikation von eingegebenen Daten (ortsunabhängig)
- Das User Interface soll primär auf Deutsch erstellt werden. Es sollen jedoch Vorkehrungen getroffen werden, um Übersetzungen einfach zu ermöglichen (Internationalisierung)

1.4. Rahmenbedingungen

- Es gelten die Rahmenbedingungen, Vorgaben und Termine der HSR
- Die Projektabwicklung orientiert sich an einer iterativen, agilen Vorgehensweise. Als Vorgabe dient dabei Scrum, wobei bedingt durch das kleine Projektteam gewisse Vereinfachungen vorgenommen werden.
- Die Kommunikation in der Projektgruppe, in der Dokumentation und an den Präsentationen erfolgt auf Deutsch.

1.5. Aufbau der Arbeit

Die Arbeit ist in drei Teile gegliedert. Im ersten Teil erfolgt eine Einleitung mit allgemeinen Informationen zum Projekt und dessen Umsetzung (siehe Kapitel 1. Informationen zum Projekt, 2. Umsetzung und 3. Definitionen). Darin werden auch Begriffe erklärt, welche während der Arbeit entwickelt oder oft verwendet werden.

Der zweite Teil beinhaltet die Dokumentation der eigentlichen Arbeitsresultate (siehe Kapitel 4. Kort, 5. Leaflet Komponente, 6. Gamification, 7. Datenquellen, 8. Architektur, 9. Infrastruktur, 10. Backend und 11. Administration). Darin wird unter anderem die Implementation der [Web-App](#) beschrieben und die Infrastruktur, welche für den Betrieb notwendig ist.

Im dritten Teil befinden sich Informationen zum Projektmanagement (siehe Kapitel 12. Projektmanagement und 13. Projektmonitoring).

Neben diesem Dokument umfasst die Arbeit die implementierte [Web-App](#) KORT. Der dazugehörige Source Code ist frei im Internet zugänglich, sowie auf der beigelegten CD zu finden.

Arbeitsresultat	URL
KORT (Web-App)	http://kort.herokuapp.com
Repository	https://github.com/odi86/kort

Tabelle 1.1.: Übersicht der Arbeitsresultate

2. Umsetzung

2.1. Stand der Technik

Da es sich bei *OpenStreetMap* um ein globales Projekt handelt, gibt es zahlreiche Anstrengungen, die Karte besser zu machen. Neben konventionellen Editoren gibt es auch Tools, welche sich explizit auf das Finden und Beheben von Fehlern spezialisiert haben. Ein Beispiel dafür ist der *NoName-Layer* von Simon Poole¹. Dieser färbt Strassen, welche fälschlicherweise keinen Namen haben, rot ein. Daneben gibt es Dienste, welche Fehlerdaten sammeln und diese zur Verbesserung anbieten, wie beispielsweise *KeepRight*² oder *OpenStreetBugs*³.

Die Gemeinsamkeit dieser Werkzeuge liegt darin, dass sie vom Benutzer bereits ein gewisses Interesse an *OpenStreetMap* und am Editieren der Karte voraussetzen. Karten-Fehler sind jedoch typische Beispiele, welche sich durch die Mithilfe von möglichst vielen Personen lösen lassen (sogenanntes *Crowdsourcing*). Die Voraussetzung für ein erfolgreiches *Crowdsourcing* ist, möglichst viele Hürden abzubauen und einfach zu lösende Aufgaben bereit zu stellen. Das Konzept der *Gamification* bietet sich daher an.

Dabei werden Spiel-Elemente in eine Applikation eingebaut und dadurch die Motivation der Benutzer gesteigert, diese Applikation längerfristig zu verwenden. Es gibt bereits einige Projekte, welche sich mit der *Gamification* von *OpenStreetMap* beschäftigen⁴.

*MapRoulette*⁵ stellt dem Benutzer eine *Challenge*, welche es zu lösen gibt. Ein Beispiel einer solchen Challenge ist *Connectivity*⁶, bei welcher Strassen, die sehr nahe beieinander liegen verbunden werden sollen. Der Benutzer hat die Wahl, ob er den Fehler korrigiert, ignoriert oder als *false positive* markiert. Wenn der Benutzer einen Fehler gelöst hat, wird ihm zufällig ein weiterer Fehler angezeigt. Die Challenge ist dann fertig, wenn alle Fehler einer Kategorie behoben sind. Durch die Weiterleitung auf den nächsten Fehler entsteht ein beinahe endloses Spiel. Jede erledigte Aufgabe hat dabei den Charakter eines Levels.

Im Rahmen der *Operation Cowboy*⁷ wurden unter anderem auch mit *MapRoulette* über 2000 *Routing*-Fehler pro Tag behoben⁸.

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

²<http://www.keepright.at/>

³<http://openstreetbugs.schokokeks.org/>

⁴http://wiki.openstreetmap.org/wiki/Games#Gamification_of_map_contributions

⁵<http://wiki.openstreetmap.org/wiki/MapRoulette>

⁶<https://oegeo.wordpress.com/2012/10/29/new-maproulette-challenge-connectivity-bugs/>

⁷http://wiki.openstreetmap.org/wiki/DE:Operation_Cowboy

⁸<https://twitter.com/opcowboy/status/272438199769501696>

2.2. Vision

Die [Web-App](#) KORT bietet dem Benutzer eine einfache Oberfläche, Fehler zu lokalisieren. Das Zielpublikum hat keinerlei Vorwissen über Karten oder *OpenStreetMap*. Da die App als Spiel konzipiert ist, werden dem Benutzer kleine, einfach zu lösende Aufgaben gestellt. Diese Aufgaben beziehen sich auf Fehler in den Karten-Daten.

Für das Lösen solcher Aufgaben gewinnt der Benutzer Punkte und kann so in der Highscore aufsteigen. Für besondere Leistungen werden ihm dabei auch Auszeichnungen verliehen. Dieser Mix sorgt dafür, dass der Benutzer die App immer wieder öffnet, um weitere Korrekturen vorzunehmen.

Als Alternative zum Beantworten von Fragen, hat der Benutzer die Möglichkeit, die Antworten von anderen Spielern zu validieren. Dazu soll er die gegebenen Antworten als richtig oder falsch markieren. Erreicht eine Antwort genügend Stimmen, welche deren Richtigkeit bestätigen, gilt diese als abgeschlossen und kann anschliessend als Korrektur an *OpenStreetMap* gesendet werden.

Durch die Implementation als cross-platform [Web-App](#), kann diese auf allen gängigen mobilen Betriebssystemen verwendet werden.

2.3. Resultate der Arbeit

Wir konnten fast alle gesetzten Ziele erreichen. KORT erfüllt alle Anforderungen an eine moderne [Web-App](#). Nach dem Login über [OAuth](#) werden dem Benutzer Fehler in seiner Umgebung auf der Karte angezeigt.

Um das Spiel zu starten, kann der Benutzer eine beliebige Aktionen durchführen. Wenn er beispielsweise einen Fehler auswählt, indem er auf dessen Markierung tippt, wird er gefragt, ob er die Lösung für den Fehler kennt. Dadurch soll die Neugier des Spielers geweckt werden.

Die Spielmechanik ist denkbar einfach, so dass das Spiel auch nur sekunden-, aber auch minutenlang gespielt werden kann. Der Benutzer erhält sofort ein Feedback und kann verfolgen, wie er sich gegenüber seinen Mitspielern verbessert. Eine zusätzliche Motivation wird über Auszeichnungen geschaffen.

Wir konnten einige Gamificationkonzepte direkt in der App umsetzen. Um das Spiel für möglichst viele Benutzer attraktiv zu machen, muss aber noch ein detaillierteres Konzept ausgearbeitet werden. Gerade mit den Badges lassen sich verschiedene Spielertypen ansprechen. Auch Schwierigkeitsstufen oder zusätzliche Berechtigungen für erfahrene Benutzer können den längerfristigen Erfolg der Applikation erhöhen.

Auf der technischen Seite haben wir erfolgreich ein System entwickelt, welches stets mit neuen Fehlern „gefüttert“ wird. Die Architektur ist so flexibel gewählt, dass sich beinahe beliebig Komponenten hinzufügen oder entfernen lassen. Dies stellt sicher, dass zukünftig auch weitere Fehlerdatenquellen in KORT integriert werden können.

2.4. Schlussfolgerungen und Ausblick

Einige Punkte aus der Aufgabenstellung sind noch offen. Wir hatten das Ziel, dass Benutzern neben einer textuellen Antwort auch noch ein Bild hochzuladen können, quasi als Beweis für ihre eingetragene Fehlerkorrektur. Dabei sind wir jedoch an einer technischen Limite des verwendeten Frontend-Frameworks *Sencha Touch 2* gescheitert. Des weiteren sollten soziale Medien wie Facebook und Twitter integriert werden, um eine breitere Öffentlichkeit zu erreichen. Im Verlaufe der Arbeit haben sich die Prioritäten diesbezüglich aber geändert.

Als letzter offener Punkt bleibt noch das Zurückschicken der Korrekturen zu *OpenStreetMap*. Da wir damit beschäftigt waren unser eigenes System fertig zu stellen, konnten wir dies schlussendlich aus Zeitgründen nicht mehr implementieren.

Die App in der jetzigen Form ist also in sich geschlossen. Die getätigten Korrekturen werden in der eigenen Datenbank abgelegt, jedoch noch nicht zu *OpenStreetMap* zurückgesendet.

Wir mussten in dieser Arbeit feststellen, dass zuerst viele Grundfunktionen implementiert werden müssen, auf denen dann Weiterentwicklungen stattfinden können. Ursprünglich haben wir uns erhofft, tiefer in die Thematik der [Gamification](#) einzusteigen und die App als „echtes“ Game zu gestalten.

Wenn unsere App trotzdem mithelfen kann, einzelne Benutzer für das [Mappen](#) zu begeistern, haben wir unser Ziel jedoch mehr als erreicht.

3. Begriffsdefinitionen

Für diese Arbeit wurden einige neue Begriffe verwendet und Konzepte definiert. Dieses Kapitel soll kurz die wichtigsten Begriffe von KORT erklären. Hierbei handelt es sich um fachliches Vokabular, welches in dieser Arbeit verwendet wird. Alle anderen, eher technischen Begriffe, befinden sich im Glossar (siehe Teil IV).

3.1. Name der Applikation

Beim Begriff KORT handelt es sich um einen skandinavischen Ausdruck für *Karte*. Obwohl dieser Begriff sehr allgemein ist, scheint er im Zusammenhang mit Applikationen noch nicht häufig verwendet worden zu sein. Der Name ist kurz und prägnant, was für eine App ideal ist.

Falls die App zukünftig auch im skandinavischen Raum verwendet wird, muss eine Namensänderung allfällig in Betracht gezogen werden.

3.2. Begriffe aus dem Spiel

Kategorie	Begriff	Beschreibung
Spieleinheit	Auftrag	Aufträge sind Fehler auf der Karte, welche von einem Benutzer korrigiert werden.
Belohnung	Koins	Punkte, welcher ein Benutzer gewinnen kann, werden <i>Koins</i> genannt. Das Wort ist vom Englischen <i>coin</i> (Münze) abgeleitet. Das „K“ ist dabei eine Anlehnung an den Spielnamen KORT.
Auszeichnungen	Badge	Auszeichnungen, die ein Benutzer gewinnen kann, werden <i>Badge</i> genannt.
Rangliste	Highscore	Rangliste aller Spieler, sortiert nach Anzahl <i>Koins</i> .
Punkt auf der Karte	Objekt	Ein speziell ausgezeichneter Ort (POI) auf der Karte, zu dem Informationen fehlen.

Tabelle 3.1.: Begriffe aus KORT

Teil II.

Projektdokumentation

4. Kort



<http://kort.herokuapp.com/>



4.1. Analyse

4.1.1. User Szenarien

Um die Anforderungen an die App genauer abschätzen zu können, wurden im Vorfeld der Implementierung einige Szenarien erstellt.

Szenario 1: Zeitvertreib an der Bushaltestelle

Simon muss 15 Minuten an der Bushaltestelle auf den nächsten Bus warten. Um sich die Wartezeit zu verkürzen, nimmt er sein Smartphone hervor und startet die KORT-Applikation.

Die App hat ihn bereits lokalisiert und zeigt ihm offene *OpenStreetMap*-Aufträge in der Nähe, wahlweise als Liste oder auf der Karte an. Für die Aufträge werden Simon verschiedene Belohnungen angeboten, abhängig vom Schwierigkeitsgrad der Aufgabe.

Simon entscheidet sich für einen Auftrag mit mittlerer Belohnung. Die App zeigt ihm den Weg zum Auftrag und erklärt, was zu tun ist. Beim Auftrag handelt es sich um einen fehlenden Strassennamen.

Als Simon vor Ort ist, findet er sehr schnell ein Strassenschild. Er gibt den Namen in der App ein wofür er bereits 10 Punkte erhält. Da er zusätzlich sogar noch ein Foto hochlädt, bekommt er weitere 5 Punkte.

Der Auftrag ist somit für ihn erledigt, was ihm entsprechend von der App mitgeteilt wird. Danach verschwindet der Auftrag aus seiner Auftragsliste und von der Karte.

Ziele

- Zeitvertreib
- Punkte sammeln
- Daten verbessern

Szenario 2: Validieren

Andy sitzt im Zug und langweilt sich. Er öffnet die KORT-App im Indoor-Modus und sieht sich die Liste der zu validierenden Lösungsvorschläge an. Er sortiert die Liste, um diejenigen Einträge zu sehen, welche nur noch einen Review benötigen, um an *OpenStreetMap* gesendet werden zu können.

Er öffnet einen Vorschlag für einen fehlenden Strassennamen in der Umgebung. Er kennt zwar die Gegend ist sich aber nicht 100% sicher ob der Name stimmt. Um sicherzugehen, öffnet er das angehängte Bild. Das Bild zeigt ein Strassenschild, welches mit dem eingegebenen Namen übereinstimmt Andy bestätigt dann die Änderung und schliesst den Bug damit ab.

Ziele

- Schnell und einfach geeignete Einträge zum Validieren finden
- Qualität der Änderungen sicherstellen

Szenario 3: Erster Kontakt zur App

Über den Kurznachrichtendienst Twitter sieht Monika, dass ihre Kollegin gerade das erste mal KORT gestartet hat. Als sie auf den Link klickt öffnet sich ihr Browser und die App wird angezeigt. Da es sich um ihren ersten Besuch auf der Seite handelt, wird ihr kurz erklärt um was es geht.

Danach sieht sie die Karte mit den vorhandenen Fehlereinträgen. Bevor sie einen Eintrag bearbeiten kann, muss sie sich anmelden. Sie wählt dazu einen Benutzernamen und wird anschliessend auf die Seite von Google weitergeleitet, um sich anzumelden. Nach dem erfolgreichen Login wird Monika zurück zur KORT-App geleitet, wo sie beginnen kann, die ersten Aufträge zu erfüllen.

Ziele

- Direkt ersichtlich, was die App kann
- Schneller Einstieg
- Einfache Anmeldung (keine Registrierung!)
- Benutzerführung durch die Funktionen

Szenario 4: Highscore-Anwärter

Edi benutzt schon seit einiger Zeit die KORT-App und hat in seinem Revier bereits den zweiten Platz der Highscore erreicht. Seine Platzierung überprüft er regelmässig in der App.

Heute möchte er endlich die Spitze erklimmen und den „Leader“-Badge erhalten. Dazu hat er sich einige Aufträge ausgesucht, welche er der Reihe nach bearbeiten will. Bei jedem Auftrag sieht Edi, wie viele Punkte er sammeln kann.

Nachdem er den 5. Auftrag erfolgreich erledigt hat, erhält er eine Benachrichtigung, dass er den „Leader“-Badge erhalten hat. Auch in der Rangliste steht Edi nun zuoberst.

Ziele

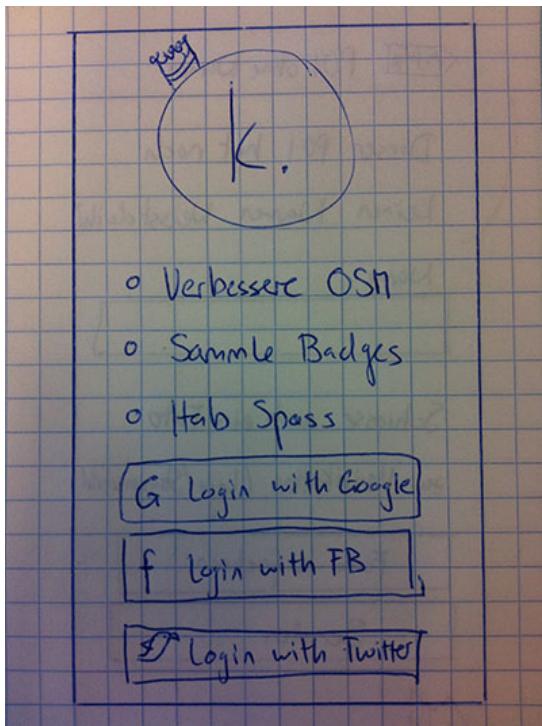
- Einfach mehrere Aufträge nacheinander ausführen
- Badges sammeln
- Highscore anzeigen
- Erster Rang erreichen

4.1.2. Paper-Prototype

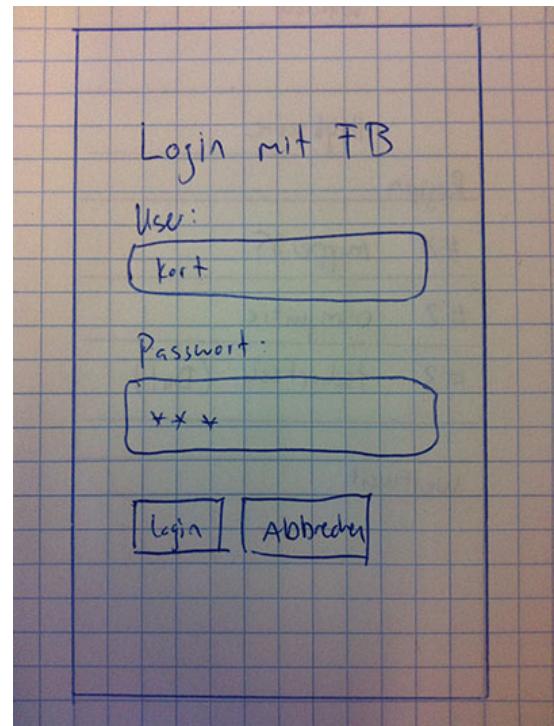
Aus den gesetzten Projektzielen und den Anforderungen der erstellten Szenarien (siehe Abschnitt 4.1.1) wurde vor der Implementation der Oberfläche ein Paper-Prototype des GUI-Designs erstellt. Der Prototype besteht aus vier Hauptmasken und einem Overlay für den Login.

Overlay: Login

Beim ersten Starten der [Web-App](#) erhält man die Möglichkeit sich über verschiedene Dienste anzumelden. Mit einem Klick auf den jeweiligen Anbieter, wird man zu diesem weitergeleitet und kann sich dort anmelden.



(1) Login - Anbieterauswahl



(2) Login - Loginformular des jeweiligen Anbieters

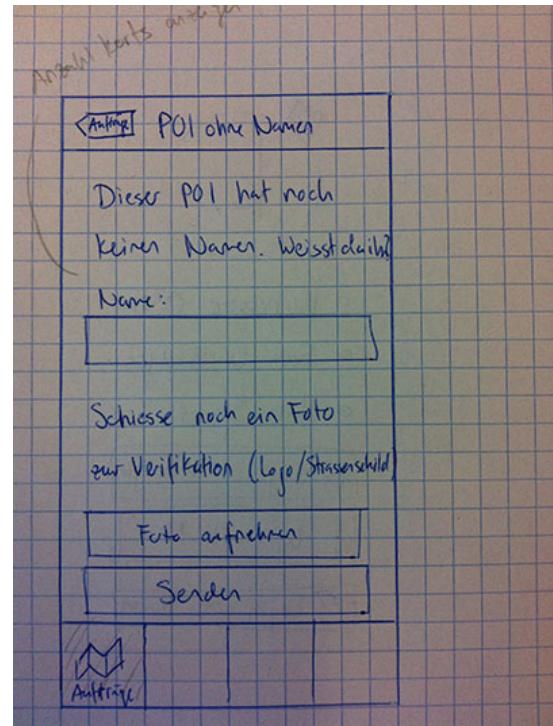
Maske: Aufträge

Nachdem sich der Benutzer erfolgreich angemeldet hat, erscheint die Maske mit den Aufträgen. Darauf werden die vorhandenen Fehler auf einer Karte angezeigt. Es werden jeweils nur die Fehler angezeigt, welche sich in unmittelbarer Nähe des eigenen Standorts befinden. Die Fehler werden mit einer Markierung auf der Karte dargestellt.

Durch Anklicken einer solchen Markierung öffnet sich die Detailansicht des Fehlers, wo sich der Fehler direkt beheben lässt. Neben der Möglichkeit, einen Lösungstext einzugeben, soll es auch möglich sein, ein Beweis-Foto hochzuladen. Mit einem Klick auf den Senden-Knopf schliesst sich die Detailansicht und man gelangt zur Karte mit den Aufträgen zurück.



(3) Aufträge - Karte mit Fehlern

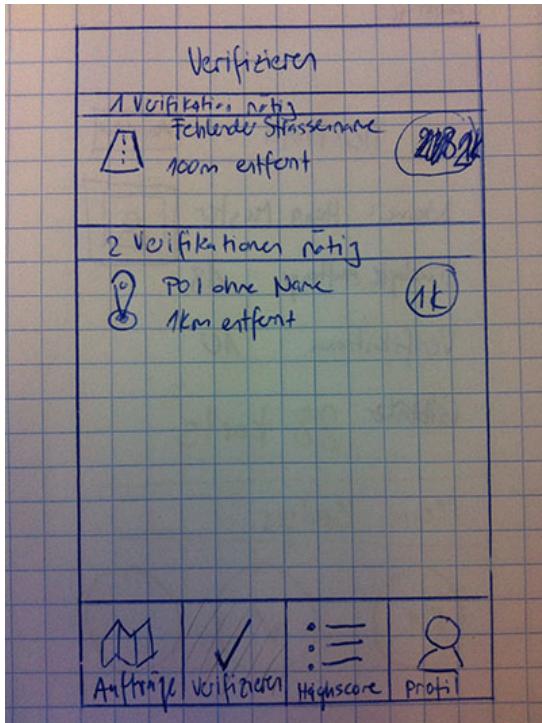


(4) Aufträge - Detailansicht eines Fehlers

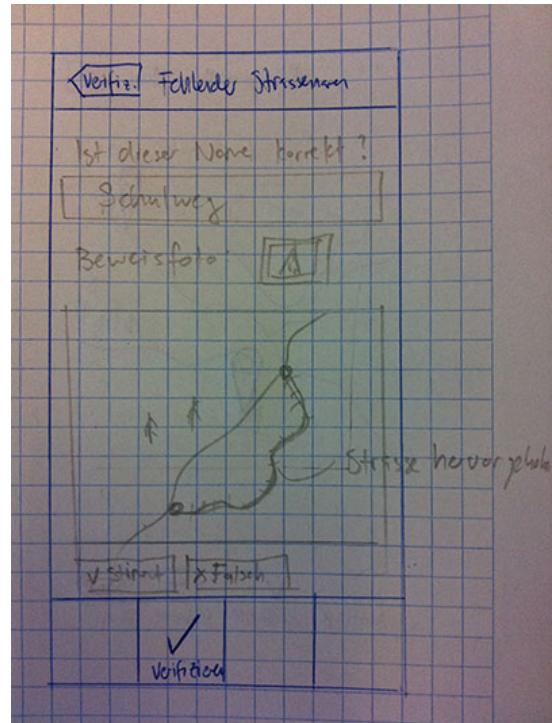
Maske: Verifizieren

Auf der Verifikationsmaske werden die bereits gelösten Fehler in der Nähe angezeigt. Sie sind gruppiert nach Anzahl nötigen Verifikationen, um sie an *OpenStreetMap* zurückzusenden.

Per Klick auf einen Eintrag öffnet sich die Verifikationsmaske. Darin wird der Fehlerlösungs-text und das Beweis-Foto angezeigt. Zusätzlich wird das betroffene *OpenStreetMap*-Objekt auf einer Karte angezeigt. Man hat die Möglichkeit, die Problemlösung als *Korrekt* oder *Falsch* zu bewerten.



(5) Verifizieren - Liste mit Fehlerlösungen



(6) Verifizieren - Detailansicht einer Fehlerlösung

Maske: Highscore

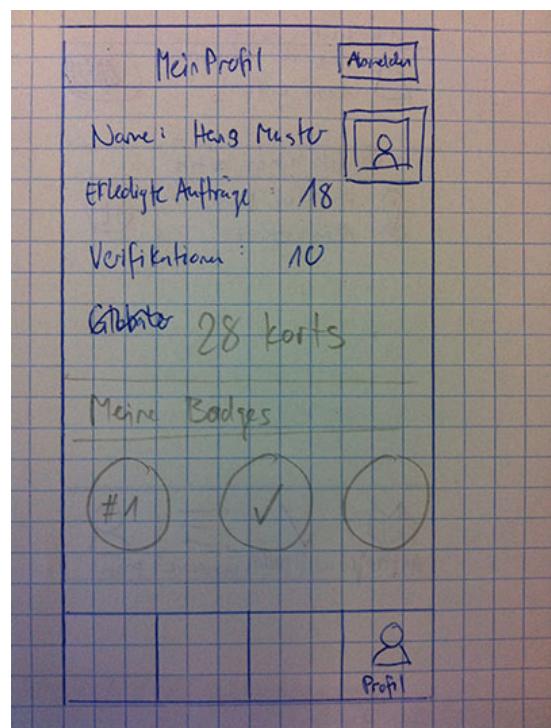
In der Highscore-Maske kann man sich mit anderen Spielern messen. Man sieht seine eigene Platzierung und die der anderen Spieler. Es werden Highscores für verschiedene Kategorien (z.B. regional, weltweit) angezeigt.

Maske: Profil

Im Profil werden die Eckdaten des eigenen Benutzers angezeigt. Dazu gehören die Anzahl der gelösten Aufträge und die Anzahl der getätigten Verifikationen. Zusätzlich werden die Gesamtanzahl der gesammelten Punkte und die gewonnenen Badges ausgewiesen. Auf der Profil-Maske kann sich der Benutzer von der App abzumelden.



(7) Highscore



(8) Profil

4.2. Design

4.2.1. Package-Struktur

Die Package-Struktur wird vom *Sencha Touch 2*-Framework vorgegeben. Sie entspricht grundsätzlich dem **MVC**-Layout. Speziell daran ist das Konzept der *Stores*, welche einen beliebigen Datenspeicher abstrahieren. Stores sind an ein *Model* gebunden, welches die Struktur der gespeicherten Daten vorgibt.

Zusätzlich sind sie über einen *Proxy* mit der Datenquelle verbunden. Dabei kann es sich beispielsweise um einen REST-Webservice handeln oder den **Local Storage** des Browsers.

In Abbildung 4.1 wird die Package-Struktur von KORT mit deren Abhängigkeiten gezeigt.

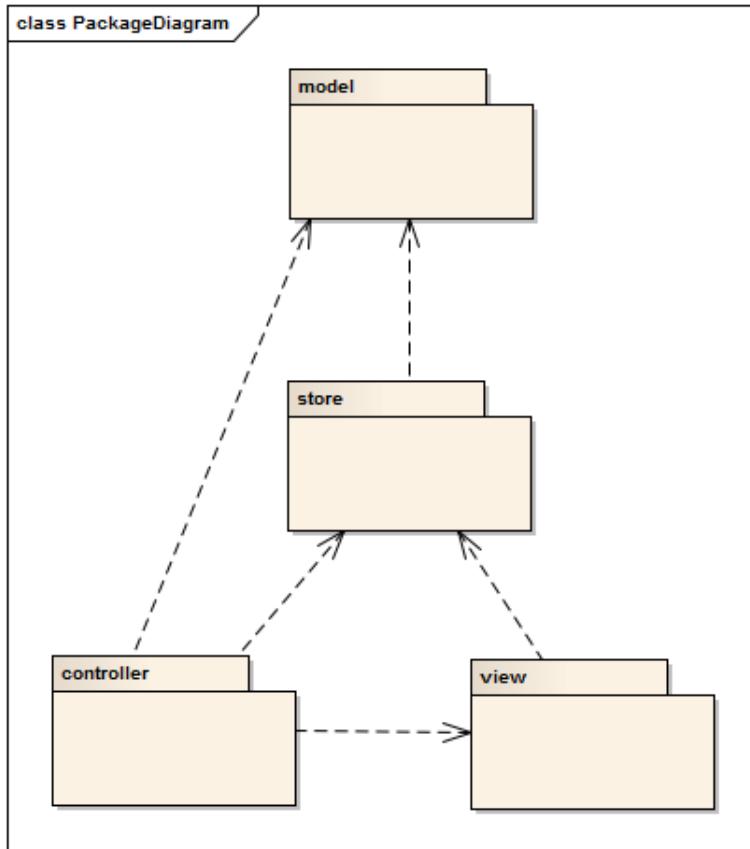


Abbildung 4.1.: Package-Struktur

4.2.2. Controller-Package

Die App ist so gestaltet, dass jede Hauptansicht einen eigenen Controller besitzt. Dieser ist für die Steuerung der Benutzeroberfläche zuständig.

Die verschiedenen Controller lassen sich grob in drei Klassen einteilen. Zunächst gibt es Controller für die einzelnen Masken der Applikation (siehe Abbildung 4.2 → *main tab controllers*). Zusätzlich haben einige Tabs eine Detailansicht, welche ebenfalls von einem eigenen Controller gesteuert wird (siehe Abbildung 4.2 → *detail controllers*). Zuletzt gibt es eigenständige Controller für die Overlay-Komponenten, welche die gesamte Oberfläche der App verdecken (siehe Abbildung 4.2 → *overlay controllers*).

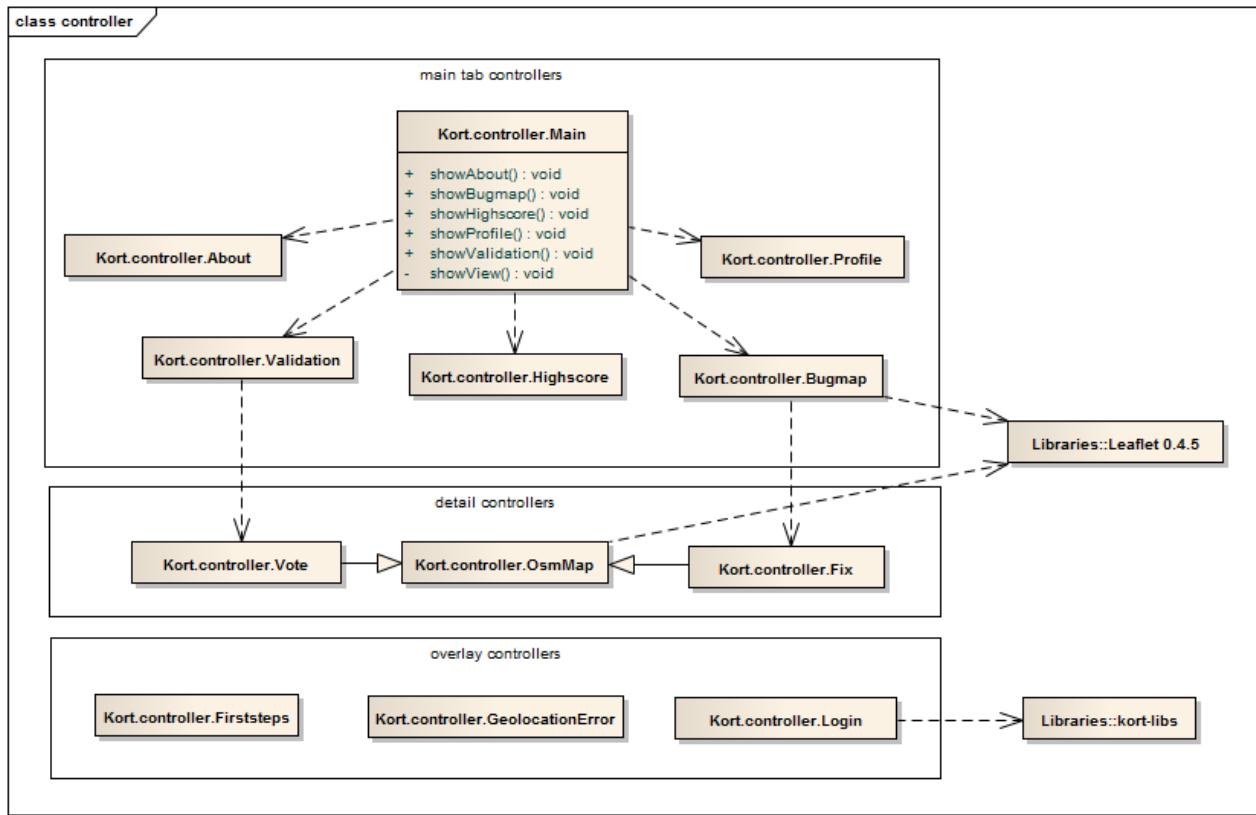


Abbildung 4.2.: Klassendiagramm der Controller

4.2.3. Store- und Model-Package

Die *Stores* bilden die Datenspeicher in einer *Sencha Touch* Applikation. Über ein *Model* wird die jeweilige Struktur der gespeicherten Daten festgelegt. Stores sind zudem über einen *Proxy* mit der Datenquelle verbunden. Im Falle von KORT wurden dafür ausschliesslich REST-Ressourcen verwendet (siehe Abbildung 4.3).

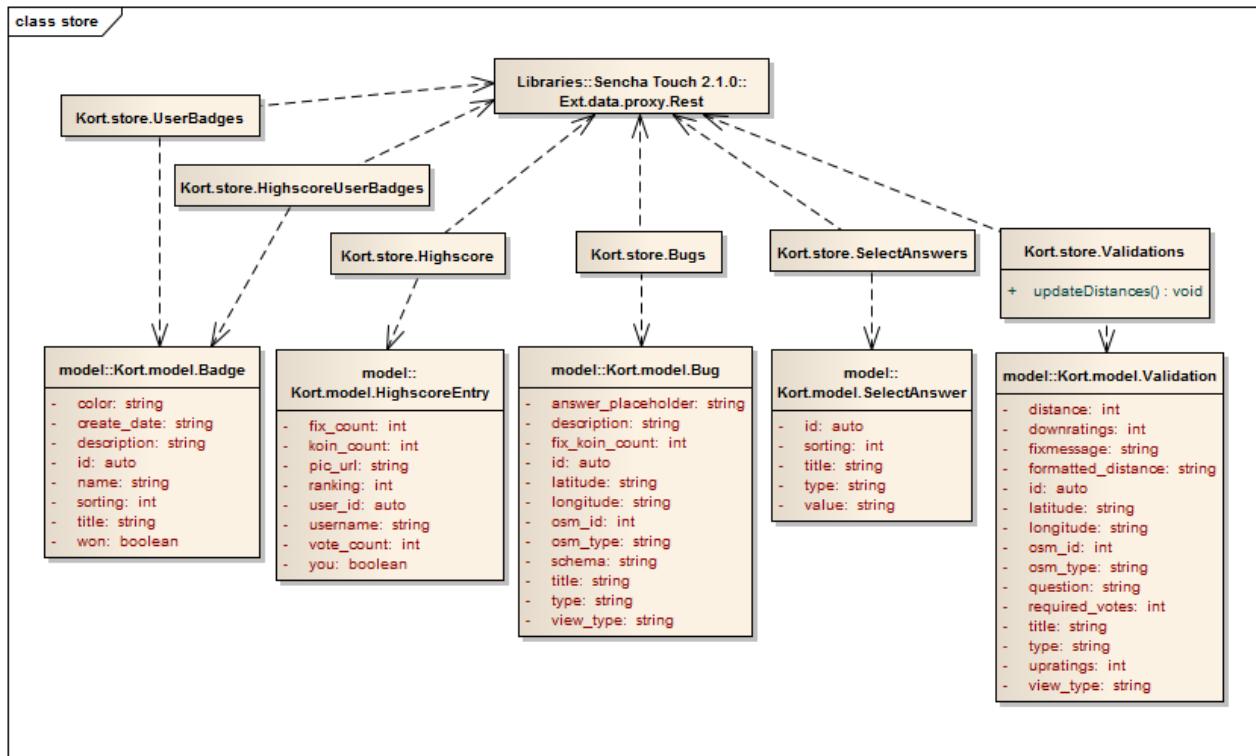


Abbildung 4.3.: Klassendiagramm der Stores

Models ohne Store

Falls die Applikation jeweils nur eine Instanz eines *Models* verwendet, können diese ohne einen Store direkt mit einem Proxy verbunden werden. In KORT wurde dies für den eingeloggten Benutzer (User) sowie für die zu sendenden Datenpakete wie der Fehler-Lösung (Fix) und der Validierung (Vote) verwendet (siehe Abbildung 4.4).

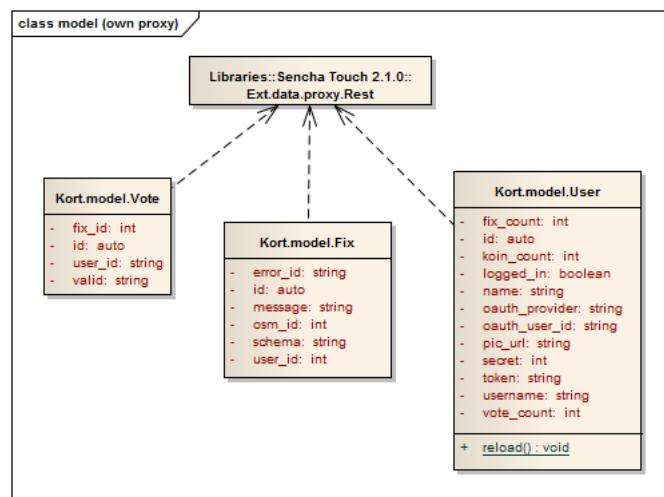


Abbildung 4.4.: Klassendiagramm der Models ohne Stores

Speichern der Logininformationen

Die Logininformationen des Benutzers werden im [Local Storage](#) des Browsers abgelegt (siehe Abbildung 4.5). Der genaue Ablauf wird in Abschnitt [10.4.1](#) beschrieben.

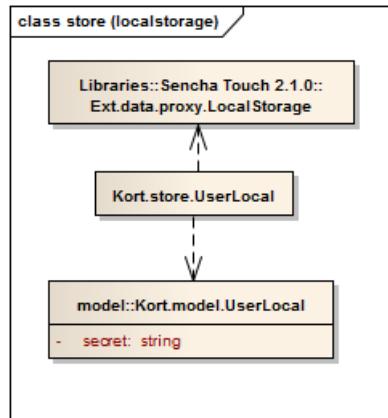


Abbildung 4.5.: Speichern der Logininformationen im Local Storage

4.2.4. Aufbau der Benutzeroberfläche

Jede Maske der Applikation befindet sich in einer eigenen View-Klasse. Diese verschiedenen View-Klassen werden in der Hauptklasse `Kort.view.Main` inkludiert und angezeigt (siehe Abbildung 4.6).

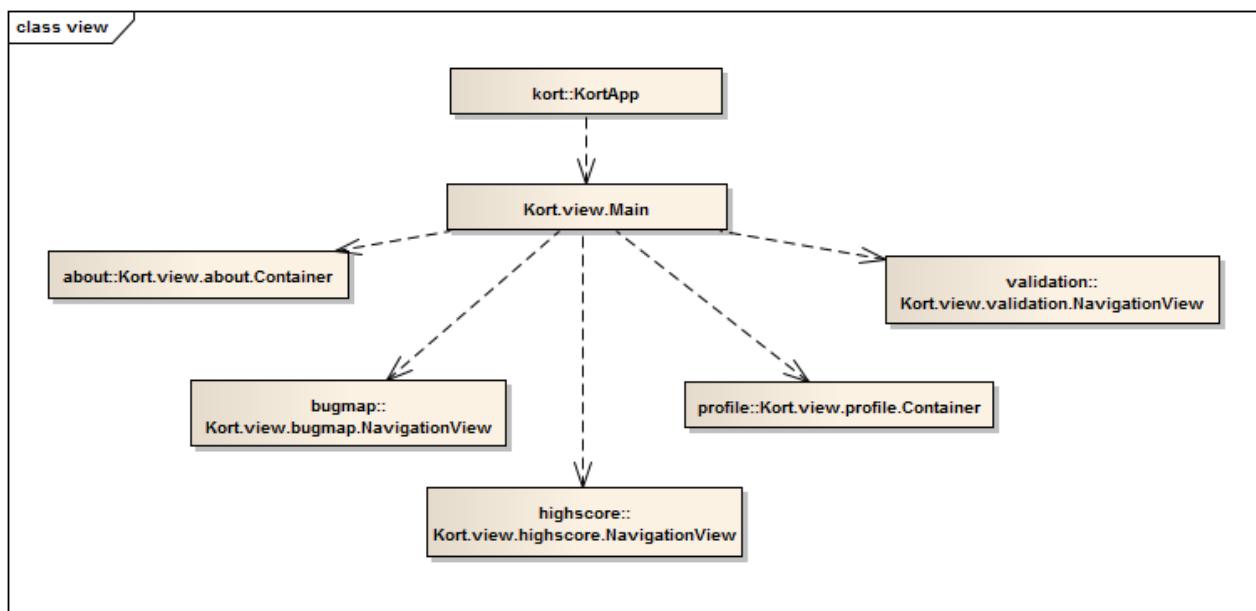


Abbildung 4.6.: Aufbau der Benutzeroberfläche

4.3. Implementation

KORT wurde mit dem *Sencha Touch 2*-Framework¹ erstellt. Zusätzlich wurden verschiedene Libraries und Community-Plugins eingesetzt, welche in der Tabelle 4.1 beschrieben sind.

*Leaflet*² wurde eingesetzt, um *OpenStreetMap*-Daten auf der Karte anzuzeigen. Zusätzlich wurde das *leaflet-osm*³-Plugin für Leaflet verwendet, um *OpenStreetMap*-Objekte als XML-Code direkt auf der Karte darzustellen.

Für die Einbindung der Leaflet-Karte in Sencha Touch wurde das Plugin *Ext.ux.LeafletMap* eingesetzt, welches ebenfalls während dieser Arbeit entwickelt wurde (siehe Abschnitt 5).

Für die Internationalisierung des Frontends wurde das *Ext.i18n.Bundle-touch*⁴-Plugin für Sencha Touch eingesetzt.

Library	Version	Verwendung
Sencha Touch 2	2.1.0	Framework zur Erstellung von mobilen Web-Apps
Sencha Cmd 2	3.0.0.250	Build-Tool von Sencha
Leaflet	0.4.5	Anzeige von <i>OpenStreetMap</i> -Daten auf der Karte
leaflet-osm	Git Revision 38665cc6c0	Leaflet-Plugin zur Anzeige von <i>OpenStreetMap</i> -Objekten auf der Karte
Ext.ux.LeafletMap	1.0.1	Sencha Touch-Plugin zur Einbindung einer Leaflet-Karte in Sencha Touch
Ext.i18n.Bundle-touch	Git Revision b4a0beaeb8	Sencha Touch-Plugin zur Internationalisierung der Oberfläche

Tabelle 4.1.: Abhängigkeiten im Frontend

4.3.1. Starten der App

Beim Starten der App wird durch mehrere Entscheidungen festgelegt, welche Maske dem Benutzer angezeigt wird. In Abbildung 4.7 ist der Ablauf grob dargestellt.

Schlussendlich wird dem Benutzer eine der folgenden Masken angezeigt:

- **Hauptmaske:** Falls der Benutzer bereit ist, die App zu benutzen.
- **Login Maske:** Falls der Benutzer noch nicht eingeloggt ist.
- **Erste Schritte Maske:** Falls der Benutzer noch keinen Benutzernamen gewählt hat.

¹<http://www.sencha.com/products/touch/>

²<http://leafletjs.com/>

³<https://github.com/jfirebaugh/leaflet-osm>

⁴<https://github.com/elmasse/Ext.i18n.Bundle-touch/tree/css-content>

- **Geolocation Fehler Maske:** Wenn keine Geolocation-Informationen verfügbar sind.

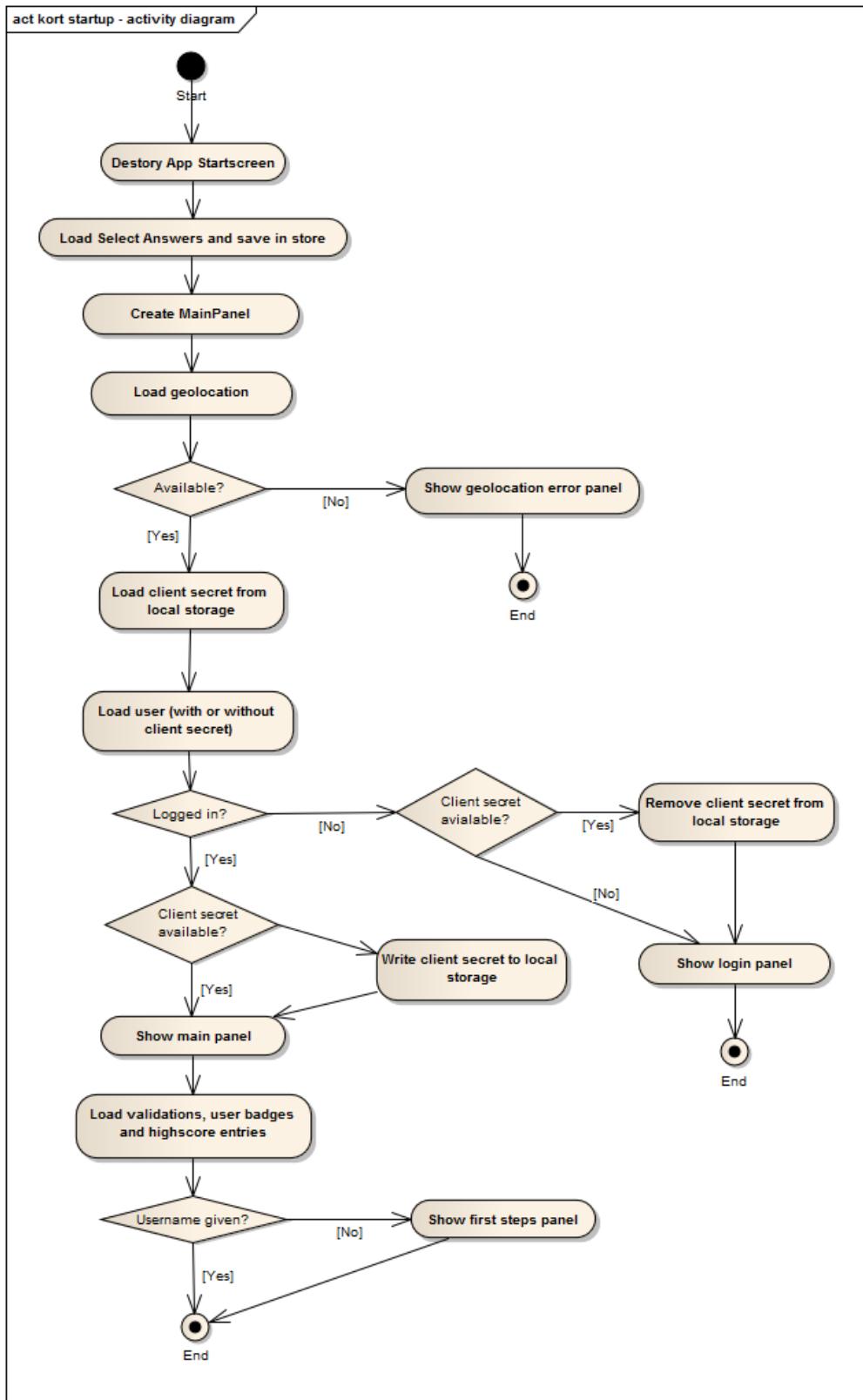


Abbildung 4.7.: Ablauf beim Starten der App

4.3.2. Sencha Cmd

Das Grundgerüst der App wurde komplett mit dem Sencha-eigenen Build-Tool *Sencha Cmd*⁵ generiert. Dieses bietet verschiedene Build-Möglichkeiten an:

Build	Beschreibung
testing	Bei diesem Build werden die JavaScript-Quelldateien in einer Datei zusammengefasst.
production	Der Production-Build entspricht grundsätzlich dem Testing-Build. Zusätzlich werden aber die JavaScript-Quelldateien komprimiert.
package/native	Mit diesen Builds lassen sich native Apps für die verschiedenen mobilen Betriebssysteme generieren.

Tabelle 4.2.: Build-Möglichkeiten mit Sencha Cmd

Durch den Einsatz von *Sencha Cmd* ist es möglich, mit geringen Mehraufwand, eine native App zu generieren und diese in den verschiedenen [App-Stores](#) anzubieten.

Konfiguration

Der Buildprozess wird in der Datei `app.json` konfiguriert. Die wichtigsten Einstellungen sind in Tabelle 4.3 beschrieben.

Property	Beschreibung
"js"	Hier können die JavaScript-Quelldateien, welche von der App verwendet werden eingetragen werden. Diese werden im Buildprozess automatisch in den <code><head></code> -Bereich des <code>index.html</code> -Files geschrieben.
"css"	Ähnlich wie das "js"-Property nur für CSS-Ressourcen.
"resources"	Hier können weitere Ressourcen wie Grafiken oder verwendete Libraries angegeben werden, welche beim Buildprozess ebenfalls in das Zielverzeichnis kopiert werden.

Tabelle 4.3.: Konfiguration von Sencha Cmd (`app.json`)

Zusätzlich ist in der Datei `.sencha/app/sencha.cfg` der Classpath der App definiert. Dieser wird für das Finden der abhängigen Ressourcen der App verwendet.

Weitere Informationen zur Konfiguration findet man im entsprechendem Guide der Sencha Docs⁶.

⁵<http://www.sencha.com/products/sencha-cmd>

⁶http://docs.sencha.com/touch/2-1/#!/guide/command_app

App Build starten

Der Build kann von der Konsole aus gestartet werden. Dazu muss in das *Root-Verzeichnis* von KORT gewechselt und folgender Befehl eingegeben werden:

```
sencha app build <environment>
```

Die erstellte App wird dann in dem entsprechenden Verzeichnis abgelegt:

```
/build/Kort/<environment>
```

4.3.3. Cross-platform Unterstützung

Grundsätzlich unterstützt KORT alle Plattformen (bzw. Browser), welche vom Sencha Touch 2 Framework unterstützt werden. Eine Liste davon findet man auf der Produkt-Homepage von Sencha Touch⁷.

Trotzdem kann es bei einigen Geräten aufgrund unterschiedlicher Browsersversionen zu Anzeigeproblemen kommen.

Getestet wurde die App mit folgenden Geräten:

- Smartphones
 - Apple iPhone 4, 4s
 - Samsung Galaxy S2, S3
 - HTC Desire S
- Tablets
 - Apple iPad 3
 - Asus Nexus 7
 - Asus Transformer

4.3.4. Internationalisierung

Die Oberfläche von KORT wurde bereits für eine mögliche Übersetzung vorbereitet. Dazu wurde das Plugin *Ext.i18n.Bundle-touch*⁸ für Sencha Touch eingesetzt. Dieses ermöglicht es, einzelne Texte in externen Sprach-Property-Dateien auszulagern.

Bei KORT befinden sich diese Files im Verzeichnis `resources/i18n/` und müssen nach folgendem Schema benannt werden: `Kort_<locale>.props`. Die verwendete Sprache wird in

⁷<http://www.sencha.com/products/touch/features/>

⁸<https://github.com/elmasse/Ext.i18n.Bundle-touch>

der Funktion `prepareI18n()` der Datei `app.js` festgelegt.

```
1 prepareI18n: function() {
2   Ext.i18n.Bundle.configure({
3     bundle: 'Kort',
4     language: 'de-CH',
5     path: 'resources/i18n',
6     noCache: true
7   );
8 }
```

Code-Ausschnitt 4.1: kort - Sprache definieren

In dieser Funktion kann über das `language`-Property die Sprache konfiguriert werden. Falls das Property weggelassen wird, wird die aktuelle Spracheinstellung des Browsers verwendet. Wird dabei die entsprechende Datei nicht gefunden, verwendet das Plugin die Datei `Kort.props`.

4.4. Resultate

KORT besteht aus fünf verschiedenen Hauptmasken. Diese sind in der Applikation über die Tabs im unteren Bereich erreichbar.

4.4.1. Maske: Aufträge

In dieser Maske werden dem Benutzer alle noch nicht gelösten Fehler in seiner Umgebung als Markierungen auf der Karte angezeigt (siehe Abbildung 4.8). Die Fehleranzahl ist dabei auf die 25 nächstgelegenen Fehler limitiert.

Beim Klick auf einen Fehler wird der Benutzer gefragt, ob er diesen auch wirklich lösen kann. Bestätigt er, wird ihm der Fehler im Detail angezeigt. In dieser Detailansicht wird ihm zudem je nach Fehlertyp ein Text- oder ein Auswahlfeld angezeigt, in welchem er die Antwort eingeben bzw. auswählen kann.

Zusätzlich gibt es die Möglichkeit, sich den Fehler nochmals auf der Karte anzuzeigen zu lassen. Dieser wird dabei als Geometrie-Objekt angezeigt. Eine Strasse wird dabei beispielsweise als Linie oder ein Gelände als Polygon dargestellt.



Abbildung 4.8.: Maske: Aufträge

4.4.2. Maske: Prüfen

In der *Prüfen*-Maske werden dem Benutzer die Lösungen angezeigt, welche noch zu überprüfen sind (siehe Abbildung 4.9). Diese sind dabei nach der Anzahl noch nötiger Überprüfungen gruppiert. So soll erreicht werden, dass Lösungen, die schon bald an *OpenStreetMap* zurückgesendet werden können, bevorzugt behandelt werden. In der Liste werden maximal 25 Überprüfungen angezeigt.

Sobald der Benutzer einen Eintrag auswählt, wird ihm neben der eingetragenen Lösung auch der Fehler auf der Karte angezeigt. Er kann nun beurteilen, ob diese Lösung korrekt oder falsch ist.



Abbildung 4.9.: Maske: Prüfen

4.4.3. Maske: Highscore

In der Highscore werden die Benutzer nach Anzahl gewonnener Punkte (sog. *Koins*) sortiert (siehe Abbildung 4.10). Es werden jeweils die ersten zehn Platzierungen angezeigt.

Wird ein Eintrag in der Highscore-Liste ausgewählt, so kann man sich das Profil des jeweiligen Benutzers anschauen.

4.4.4. Maske: Profil

Im Profil findet man eine Zusammenfassung seiner persönlichen Spielaktivitäten. Man sieht die Gesamtanzahl der gesammelten *Koins* und eine Übersicht der gewonnenen Auszeichnungen (siehe Abbildung 4.10).

Auf der Profilseite lassen sich die Auszeichnungen auch in Grossformat anzeigen.

4.4.5. Maske: Über Kort

Auf der *Über Kort*-Seite werden allgemeine Informationen zur Applikation angezeigt (siehe Abbildung 4.10).

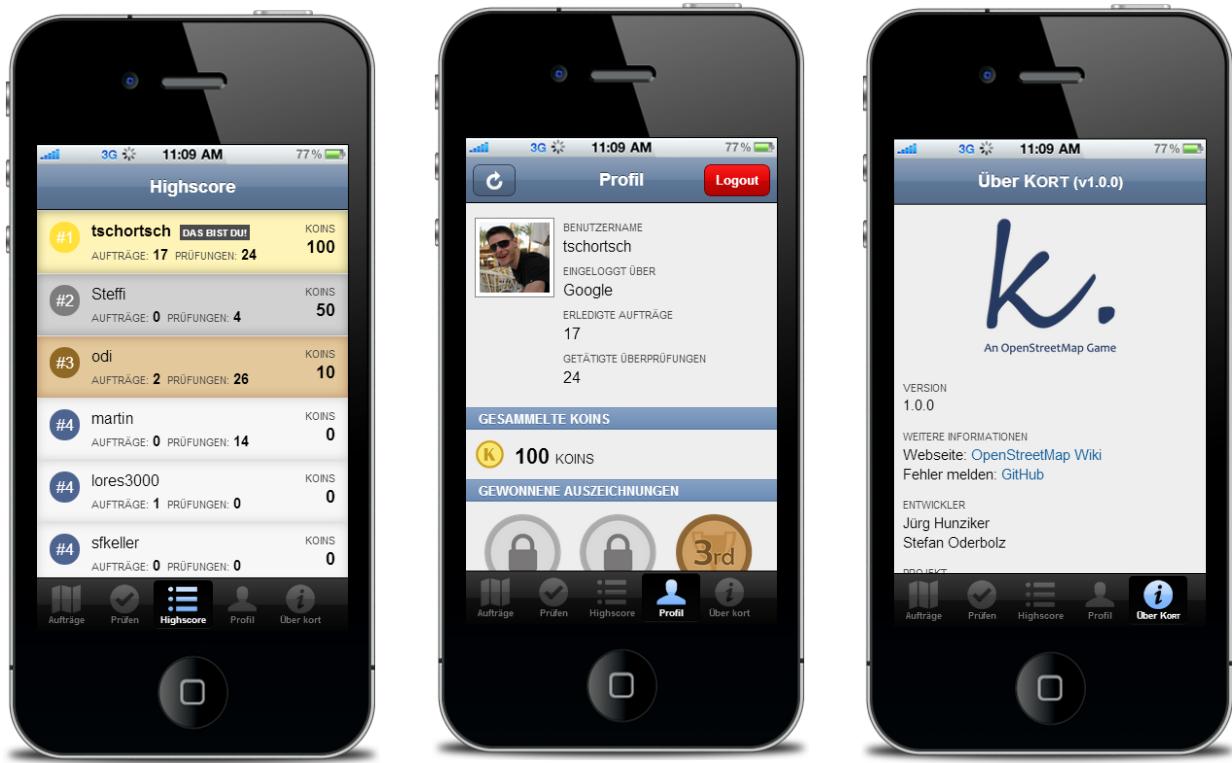


Abbildung 4.10.: Masken: Highscore / Profil / Über KORT

4.5. Dokumentation

Das Frontend von KORT ist durchgängig mit der Sencha-eigenen Dokumentationssprache *JSDuck*⁹ dokumentiert. Die Dokumentation findet sich unter: <http://kort.herokuapp.com/docs/Kort>.

Abbildung 4.11.: Frontend Dokumentation mit JSDuck

⁹<https://github.com/senchalabs/jsduck>

4.6. Bekannte Fehler

4.6.1. iOS6 - Add to homescreen

Eine sehr nützliche Funktionalität, welche Apple im Mobile Safari-Browser anbietet ist die *Add to homescreen*-Funktion.



Abbildung 4.12.: iOS Add to homescreen-Funktion

Dadurch wird ein App-ähnliches Bookmark der aktuellen Webseite auf dem Homescreen erstellt. Dieser erhält ein hinterlegtes Icon und einen Titel. Beim Starten der App erscheint ein Splashscreen, welchen man ebenfalls in der Webseite definieren kann. Zudem öffnet sich der Browser ohne jegliche Toolbars wie Adress- oder Navigationsleiste.

Leider befindet sich in iOS6 ein Bug, welcher den Zugriff auf die Geolocation verhindert, wenn die [Web-App](#) vom Homescreen aus gestartet wird. Auf StackOverflow¹⁰ wird der Bug genauer beschrieben.

Workaround

Im Sencha Forum¹¹ wird als Workaround vorgeschlagen, die Generierung des `apple-mobile-web-app-capable` Metatags in der Sencha Touch Library zu deaktivieren (siehe Code-Ausschnitte 4.2 und 4.3).

¹⁰ <http://stackoverflow.com/questions/12503815/ios-6-breaks-geolocation-in-webapps-apple-mobile-web-app-capable>

Code-Ausschnitt 4.2: Metatag, welcher iOS6 Bug hervorruft

¹¹ <http://www.sencha.com/forum/showthread.php?246317-2.1.0-RC1-Save-to-home-screen-Geolocation-not-working>

```
1 //meta('apple-mobile-web-app-capable', 'yes');
```

Code-Ausschnitt 4.3: iOS6 Bug Workaround

Das Deaktivieren dieser Zeile hat aber zur Folge, dass der native Rahmen des Browsers (Adressleiste, Navigationsleiste) wieder angezeigt wird. Dies ist zwar unschön, löst aber das Problem mit dem Zugriff auf die Geolocation.

4.6.2. App Build

Wie in Abschnitt 4.3.2 beschrieben, basiert die App vollständig auf dem Sencha-eigenen Build-Tool *Sencha Cmd*. Darin sind aber noch einige Bugs vorhanden.

Bei KORT besteht dabei ein Problem bei der fest eingebauten Komprimierung der JavaScript-Sourcen. Während diesem Prozess werden lokale Variablennamen mit einzelnen Buchstaben abgekürzt, um die Dateigröße zu reduzieren. Dabei treten Konflikte mit der *Leaflet*-Library auf, welche den Buchstaben *L* als Namespace verwendet.

Workaround

Um diese Problem zu umgehen, mussten wir das Build-Skript von *Sencha Cmd* minimal anpassen. So mussten wir die Zeile, welche den *Microloader* komprimiert, auskommentieren (siehe Code-Ausschnitt 4.4). Diese befindet sich in folgender Datei:

/<Sencha Cmd Verzeichnis>/plugins/touch/current/app-build.js Zeile 362

```
1 processIndex = function () {
2     [...]
3     compressor = new ClosureCompressor();
4     microloader = (environment == 'production'
5         ? 'production'
6         : 'testing') +
7         '.js';
8     _logger.debug("using microloader : {}", microloader);
9     content = readFileContent(joinPath(sdk, "microloader", microloader));
10    //content = compressor.compress(content);
11    remotes = [
12        '<script type="text/javascript">' +
13            content + 'Ext.blink(' +
14                (environment == 'production' ? jsonEncode({
15                    id: config.id
16                }) : appJson) + ')',
17        '</script>',
18    ];
19    [...]
20};
```

Code-Ausschnitt 4.4: Sencha Cmd Workaround

5. OpenStreetMap-Daten in Sencha Touch

5.1. Sencha Touch Map-Komponente

Das Sencha Touch 2-Framework bietet zur Darstellung einer Karte lediglich eine Google Maps-Komponente an. Diese ist stark auf das Google Maps API ausgerichtet und kann deshalb nicht für andere Kartendaten verwendet werden.

Um trotzdem Daten von *OpenStreetMap* verwenden zu können, mussten wir eine neue Sencha Touch Map-Komponente erstellen, welche eine für diesen Zweck vorgesehene Library verwendet.

5.2. Leaflet Map-Komponente

Zur Darstellung der OSM-Daten verwendeten wir zuerst die *OpenLayers*¹-Library. Leider mussten wir nach einiger Zeit feststellen, dass diese für unsere Zwecke zu komplex und überladen ist.

Wir haben deshalb eine Komponente erstellt, welche die *Leaflet*²-Library zur Darstellung der Karte verwendet. Leaflet ist eine moderne, leichtgewichtige Karten-Library. Sie wurde speziell für den Einsatz auf mobilen Geräten konzipiert. Zusätzlich ist sie sehr gut dokumentiert und lässt sich einfach bedienen.

¹<http://openlayers.org/>

²<http://leafletjs.com/>

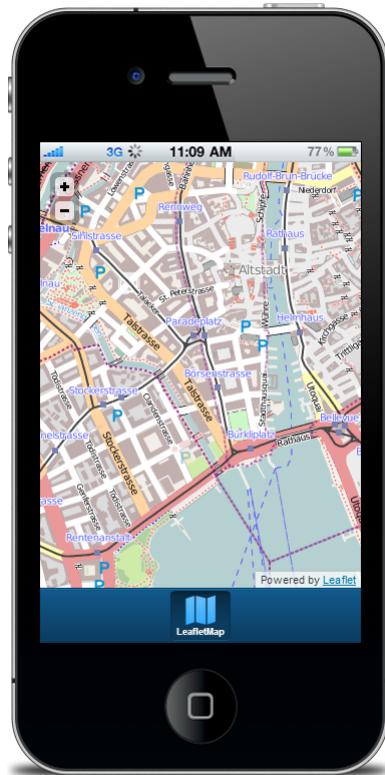


Abbildung 5.1.: Ext.ux.LeafletMap-Komponente in Sencha Touch

5.2.1. Verfügbarkeit

Unsere Sencha Touch Komponente war zuletzt so ausgereift, dass wir uns entschieden haben, diese für die Allgemeinheit zugänglich zu machen. Wir veröffentlichten sie deshalb unter dem Namen *Ext.ux.LeafletMap* im offiziellen Sencha Market³. Sie ist verfügbar unter: <https://market.sencha.com/users/162/extensions/177>.

Ort	URL
Sencha Market	https://market.sencha.com/users/162/extensions/177
GitHub	https://github.com/tschortsch/Ext.ux.LeafletMap

Tabelle 5.1.: Ext.ux.LeafletMap Verfügbarkeit

³<http://market.sencha.com/>

The screenshot shows the Sencha Market website. At the top, there's a dark header with the Sencha market logo, a search bar, and a button to 'List your Extension'. Below the header, the URL 'Home / Components' is visible. The main content area features a large green title 'Ext.ux.LeafletMap' followed by the author 'by Jürg Hunziker - Version 1.0.0'. A 'Plugins' category button is shown. A brief description states: 'Sencha Touch 2.x component which wraps a Leaflet map.' Below this, 'Products: Sencha Touch 2.1 and Sencha Touch 2.0' and 'Browsers: Android (mobile), Safari (desktop), Chrome (desktop) and iOS (mobile)' are listed. A prominent green 'Download' button is on the right. To the left, there's a thumbnail image of a map showing parking spots (labeled 'P') and a link to see more examples. On the right side, there are sections for 'Readme', 'License' (MIT), and 'Version History' (Version 1.0.0). The 'Readme' section contains a brief description of the component and its compatibility.

Abbildung 5.2.: Leaflet Map-Komponente im Sencha Market

5.2.2. Dokumentation

Die Komponente ist durchgängig mit der Sencha-eigenen JavaScript-Dokumentationssprache *JSDuck*⁴ dokumentiert. Die Dokumentation befindet sich unter: <http://kort.herokuapp.com/docs/Ext.ux.LeafletMap>.

⁴<https://github.com/senchalabs/jsduck>

6. Gamification

Unter **Gamification** versteht man die Verwendung von spieltypischen Elementen in einem spielfremden Kontext. Die bekanntesten Elemente dabei sind die Punktevergabe für das Lösen von Aufgaben, die Verleihung von Badges für besonders engagierte Einsätze oder das Bereitstellen einer Highscore zum Vergleich mit anderen Benutzern.

Während unserer Arbeit hatten wir verschiedene Partner, welche uns bei der **Gamification** unserer Applikation unterstützen. So ist unserer Industriepartner Reto Senn Mitinhaber der Firma bitforge AG¹, welche sich auf Games im mobilen Umfeld spezialisiert hat. Er konnte uns vor und während der Entwicklung viele wertvolle Tipps zur Verbesserung des Spielgefühls geben.

6.1. Gamification in Kort

6.1.1. Sprache

Gamification findet sich nicht nur in schönen Grafiken und Spiel-Elementen. Sie beginnt bereits bei der Sprachwahl von Texten. Beim Lesen der Texte sollte eine gewisse Spannung aufgebaut werden. Dadurch erhöht sich die Motivation, die anstehende Aktion durchzuführen.

In KORT findet sich dafür ein Beispiel in der Button-Bezeichnung bei der Eingabe des eigenen Benutzernamens. Zu Beginn war dieser mit „App starten“ beschriftet. Wir haben ihn aber später mit dem Text „Mission beginnen!“ ersetzt. Weiter wurde der Button grün eingefärbt, um ihn zusätzlich vom Text und vom Hintergrund abzuheben.



Abbildung 6.1.: Gamification - Sprache

Neben der gewählten Ausdrucksweise sollte die Sprache dem Zielpublikum angepasst werden. Wir mussten dafür viele Begriffe aus dem **Mapping**-Vokabular für unsere breite Zielgruppe mit allgemeineren Wörtern ersetzen.

¹<http://bitforge.ch/>

6.1.2. Punktesystem „Koins“

Eines der wichtigsten Elemente im Spiel bildet das Punktesystem. Dieses findet sich in beinahe allen Aktionen der App wieder. So gewinnt ein Spieler für gelöste Aufgaben oder getätigte Prüfungen eine gewisse Anzahl an sogenannten *Koins*. Mit der Anzahl *Koins* kann er sich wiederum in der Highscore mit den anderen Spieler messen.

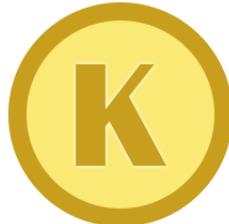


Abbildung 6.2.: Gamification - Koins

6.1.3. Auszeichnungen

Zusätzlich zu den *Koins* kann ein Spieler Auszeichnungen gewinnen. Diese erhält er durch besondere Leistungen. In KORT sind momentan folgende Auszeichnungen implementiert:

Auszeichnungstyp	Beschreibung
Anfänger	Eine Auszeichnungen für das Lösen des 1. Auftrags und eine für das Überprüfen der 1. Antwort.
Platzierung	Drei Auszeichnungen für das Erreichen des 1., 2. und 3. Rangs in der Highscore.
Aufträge	Drei Auszeichnungen für das Lösen von 10, 50 und 100 Auträgen.
Prüfungen	Drei Auszeichnungen für 10, 10 und 1000 geprüfte Antworten.

Tabelle 6.1.: Auszeichnungen in KORT



Abbildung 6.3.: Gamification - Badges

Das Hinzufügen von zusätzlichen Auszeichnungen wird in Abschnitt [11.3](#) beschrieben.

6.1.4. Highscore

Über die Highscore haben die Benutzer der App die Möglichkeit sich mit den anderen Spielern zu vergleichen. Dazu werden sie nach Anzahl gewonnener *Koins* in einer Rangliste eingestuft.

6.2. Weitere mögliche Elemente

Neben den verwendeten Gamification-Elementen in KORT gibt es noch eine Vielzahl weiterer Elemente, welche sich für diesen Anwendungszweck eignen würden. Diese konnten während der Arbeit aber nicht implementiert werden.

Daneben gibt es aber auch noch Elemente, welche zwar für *OpenStreetMap* als Projekt interessant wären, sich jedoch nicht mit KORT umsetzen lassen.

Diese werden in der Folge genauer beschrieben.

6.2.1. Erste Schritte

Um den Einstieg in die Verwendung der App weiter zu vereinfachen, wäre es sinnvoll, beim ersten Start eine kurze Einführung anzuzeigen. So könnte man dem Benutzer für die einzelnen Masken jeweils Tipps einblenden oder gar eine geführte Tour durch die App und deren Möglichkeiten anbieten.

Wenn der Benutzer nicht weiß, welche Möglichkeiten er hat, kann dies dazu führen, dass er schnell wieder aufgibt oder nicht das volle Potential der App ausschöpfen kann.

6.2.2. Zeitlich begrenzte Aktionen

Durch das Einführen von zeitlich begrenzten Aktionen kann man Benutzer dazu motivieren, die App über einen längeren Zeitraum zu verwenden. So könnte man Tage definieren, an denen man die doppelte Anzahl an Punkten gewinnt. Zusätzlich könnte man Aktionen² starten, bei denen man spezielle Auszeichnungen gewinnen kann.

Um den Benutzer dazu zu animieren, die App erneut zu starten, könnte man per Push-Meldungen auf aktuelle Aktionen, Updates oder Ereignisse aufmerksam machen.

6.2.3. Weitere Auszeichnungen hinzufügen

Bei der Auswahl an verfügbaren Auszeichnungen sollte man darauf achten, dass es für jeden Spielertyp geeignete Auszeichnungen zu gewinnen gibt. Die verschiedenen Typen haben eine ganz andere Herangehensweise und müssen deshalb auch mit verschiedenen Belohnungen belohnt werden.

²Beispiel einer zeitlich begrenzten Aktion in *OpenStreetMap*: Das *big baseball project 2011* http://wiki.openstreetmap.org/wiki/Big_baseball_project_2011

gen motiviert werden. Eine Einteilung von verschiedenen Spielertypen wird auf *Gamasutra*³ beschrieben.

Es gibt bereits eine Liste von möglichen Badges im Wiki von OpenStreetMap⁴.

6.2.4. Verschiedene Highscores

Durch das Bereitstellen von verschiedenen Highscores (Bsp. Regional, Nach Fehlertyp), gibt man allen Benutzern die Chance, irgendwo den ersten Platz zu erreichen. Dadurch verhindert man eine mögliche Demotivation beim Vergleich mit Langzeitspielern, welche bereits eine grosse Anzahl an Punkten gesammelt haben.

6.2.5. Erfahrene Spieler

Man könnte als Belohnung für viele gelöste Fehler die Berechtigungen des Benutzers erhöhen. So könnte beispielsweise seine Stimme bei einer Überprüfung einer Lösung doppelt zählen. Dieses Prinzip wird auch von der Frage/Antwort-Plattform *StackOverflow*⁵ angewendet.

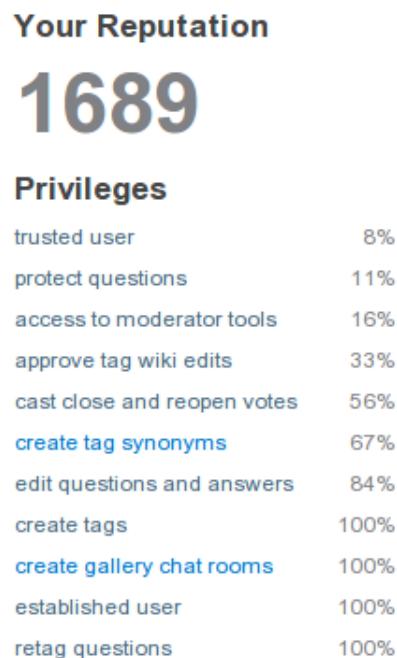


Abbildung 6.4.: Zusätzliche Berechtigungen bei StackOverflow

Um einen erfahrenen Spieler nicht zu unterfordern, könnte man ihm beim Erreichen einer gewissen Punktzahl, Fehler anzeigen, welche schwieriger zu lösen sind. Wichtig ist es dabei,

³http://www.gamasutra.com/view/feature/6474/personality_and_play_styles_a_.php

⁴<https://wiki.openstreetmap.org/wiki/Badges>

⁵<http://stackoverflow.com/>

dass das Spiel nicht plötzlich vorbei ist. Auch der Spieler, der bereits die meisten Punkte hat, muss noch eine Motivation haben, die App weiter zu verwenden.

6.2.6. Einbinden in Apple Game Center

Apple bietet mit dem *Game Center* einen zentralen Ort an, Punkte und Auszeichnungen von Game-Apps zu speichern. Dadurch können sich Spieler direkt mit anderen Spielern und Kollegen, die ebenfalls diese App verwenden, vergleichen. Dadurch, dass das *Game Center* bereits eine grosse Community an Spielern aufweist, wäre es von Vorteil, die App darin einzubinden. Leider besteht dabei die Einschränkung, dass man lediglich iOS-Games für das Game Center anmelden kann.



Abbildung 6.5.: Apple Game Center

6.2.7. Design

Spiele habe viele Design-Eigenheiten, welche sich von reinen Business-Applikationen abheben. Dies kann durch geeignete Farben und UI-Elemente realisiert werden. Typischerweise wird ein Benutzer durch die Applikation geführt, so dass ihm immer klar ist, wie es weiter geht.

6.2.8. Gamification von OpenStreetMap

In der *OpenStreetMap* Community gibt es bereits einige Ideen um Spiele-Konzepte für das Projekt zu verwenden⁶.

An der State Of The Map Konferenz 2011 (SotM 2011) hat Martijn van Exel einen Vortrag zum Thema *Can gaming concepts help make OpenStreetMap better?*⁷ gehalten. Er beschreibt darin eines der Hauptprobleme von *OpenStreetMap*, dass es neuen Benutzern schwerfällt die *erste Meile* zurückzulegen. Viele neue Benutzer haben Angst davor etwas falsch zu machen und müssen sich zuerst mit den vielfältigen Möglichkeiten des Karten-Editors vertraut machen. Die Statistik der Benutzer von *OpenStreetMap* bestätigt dieses Bild: fast zwei Drittel der Benutzer hat noch nie Änderungen vorgenommen⁸.

Martijn van Exel schlägt deshalb vor, dass das Recht die Karte zu editieren gestaffelt werden

⁶z.B. von Prof. Stefan Keller initiierte Diskussion auf der Geowanking-Mailingliste: http://geowanking.org/pipermail/geowanking_geowanking.org/2012-September/thread.html#26302

⁷http://wiki.openstreetmap.org/wiki/SotM_2011_session:_Insert_Coin_To_Play

⁸<http://osmstats.altogetherlost.com/>

soll. So soll sich ein Benutzer das Recht „verdienen“ müssen, komplexere Änderungen an der Karte vorzunehmen.

Dadurch kann der Spieltrieb des Menschen geweckt werden, um sich weiterzuentwickeln. Nebenbei lernt der Benutzer den Umgang mit dem Karteneditor und fühlt sich zunehmend sicherer überhaupt Änderungen vorzunehmen.

7. Fehler-Datenquellen

Die Grundlage unserer [Web-App](#) bilden die Fehlerdaten, welche korrigiert werden sollen. Deshalb musste zuerst eine geeignete Quelle für Fehlerdaten von *OpenStreetMap* gefunden werden.

7.1. Evaluation von geeigneten Datenquellen

Durch die Anforderungen an die Applikation ergaben sich folgende Bedingungen, welche die Fehlerdaten erfüllen müssen:

- Die Fehler müssen auch für Nicht-[Mapper](#) lösbar sein.
- Sie müssen auf Deutsch übersetzbare sein. Die Fehlerbeschreibungen sollten somit einem gegebenen Schema folgen und keinen Freitext beinhalten.
- Die Fehler müssen sich einfach in einem UI abbilden lassen.
- Die Fehler sollen sich nur auf Metadaten beziehen und nicht auf Geometrieobjekte (Strasse, Gebäude, usw.), da wir keinen vollwertigen Karten-Editor anbieten können.
- Die Fehler sollen sich immer nur auf einen [Tag](#) des betroffenen *OpenStreetMap*-Objekts beziehen. Dadurch vereinfacht sich das Zurückschreiben der Daten zu *OpenStreetMap*.

Wir untersuchten mehrere mögliche Fehlerdatenquellen im Hinblick auf ihre Qualität.

7.1.1. Fehler aus OpenStreetMap extrahieren

Da die Daten von *OpenStreetMap* frei verfügbar sind, steht es jedem offen, diese Daten zu nutzen und selbst zu verarbeiten. Es wäre somit eine Möglichkeit, sich selbst daran zu versuchen, Fehler mit geeigneten Regeln zu finden und aus den Daten zu extrahieren.

URL	http://www.openstreetmap.org
Erfasser	Backend von KORT
API verfügbar?	Ja (REST API) http://wiki.openstreetmap.org/wiki/API_v0.6
Lösung zurückschreiben	Lösung könnte via API an <i>OpenStreetMap</i> gesendet werden.
Eignung	schlecht Das Finden von Fehlern ist komplex, da es sehr viele Ausnahmen zu beachten gibt. Dies würde den Rahmen dieser Arbeit sprengen.

Tabelle 7.1.: Fehler aus OpenStreetMap

7.1.2. **FIXME-Tags in OpenStreetMap**

Fehlerhafte Objekte in *OpenStreetMap* können von Benutzern direkt mit einem **FIXME-Tag** versehen werden, wenn sie einen Fehler aufweisen. Darin wird der eigentliche Fehler beschrieben.

URL	http://wiki.openstreetmap.org/wiki/DE:Key:fixme
Erfasser	<i>OpenStreetMap</i> -Benutzer
API verfügbar?	Ja (REST API) http://wiki.openstreetmap.org/wiki/API_v0.6
Lösung zurückschreiben	Lösung könnte via API an <i>OpenStreetMap</i> gesendet werden.
Eignung	schlecht Durch die manuelle Erfassung der Daten, ist es für uns nicht möglich, automatisiert ein GUI für die Fehlerbehebung zu erstellen.

Tabelle 7.2.: FIXME-Tags in OpenStreetMap

7.1.3. **OpenStreetBugs**

Mit der Applikation *OpenStreetBugs* ist es für die Benutzer möglich, Fehler in den Karten-daten als Bug zu erfassen.

URL	http://openstreetbugs.schokokeks.org/
Erfasser	OpenStreetBugs-Benutzer
API verfügbar?	Ja (REST API oder Fehlerdaten als Dump-File downloadbar) http://wiki.openstreetmap.org/wiki/OpenStreetBugs/API_0.6
Lösung zurückschreiben	Lösung müsste an <i>OpenStreetMap</i> gesendet werden. Zusätzlich müsste der Bug auf OpenStreetBugs als gelöst markiert werden.
Eignung	schlecht Die erfassten Daten sind Freitext. Sie eignen sich daher nicht für die automatisierte Erstellung eines GUIs.

Tabelle 7.3.: OpenStreetBugs

7.1.4. KeepRight

KeepRight ist ein Dienst, welcher jeweils über Nacht Fehler in OSM-Objekten sucht und in einer eigenen Datenbank ablegt.

URL	http://keepright.ipax.at
Erfasser	Automatisiert durch nächtlichen Job
API verfügbar?	Ja (Fehlerdaten werden als Dump-File zum Download angeboten) http://keepright.ipax.at/interfacing.php
Lösung zurückschreiben	Lösung müssten lediglich an <i>OpenStreetMap</i> gesendet werden. <i>KeepRight</i> baut seine Fehlerdatenbank jeweils über Nacht aus den <i>OpenStreetMap</i> -Daten neu auf.
Eignung	gut Dadurch, dass die Fehlerdaten automatisiert erstellt werden, ist es gut möglich, für jeweilige Fehlertypen eine geeignete Maske für deren Behebung zur Verfügung zu stellen.

Tabelle 7.4.: KeepRight

7.1.5. MapDust

MapDust ist ein Bug-Tracker der Firma skobbler¹. Er ist *OpenStreetBugs* (siehe Abschnitt 7.1.3) sehr ähnlich, konzentriert sich aber mehr auf Fehler beim [Routing](#).

¹<http://www.skobbler.de/>

URL	http://www.mapdust.com/
Erfasser	Benutzer von skobbler
API verfügbar?	Ja (Dump-File der Fehler sowie API) siehe http://wiki.openstreetmap.org/wiki/MapDust
Lösung zurückschreiben	Korrekturen können direkt an <i>OpenStreetMap</i> geschickt werden. Zusätzlich muss der Bug bei MapDust geschlossen werden.
Eignung	mittel Die Qualität der Fehler lässt sehr zu wünschen übrig, da sie meist von unerfahrenen Benutzern erstellt werden. Positiv ist, dass es eine Vielzahl von Fehlern gibt und auch ein API angeboten wird.

Tabelle 7.5.: MapDust

7.1.6. Housenumbervalidator

Der *Housenumbervalidator* ist eine Quelle, welche die *OpenStreetMap*-Daten automatisiert auf fehlerhafte Adressen prüft. Die Fehler reichen von falschen Strassennamen bis zu Duplikaten von Hausnummern in der gleichen Straße.

URL	http://gulp21.bplaced.net/osm/housenumbervalidator/
Erfasser	Berechnet aus <i>OpenStreetMap</i>
API verfügbar?	Nein, aber der Quellcode ist frei verfügbar.
Lösung zurückschreiben	Korrekturen können direkt an <i>OpenStreetMap</i> geschickt werden.
Eignung	mittel Es ist schade, dass es kein API des Housenumbervalidators gibt. Die Fehler sind zum Teil schwierig zu verstehen für Laien, was die Erstellung eines UIs schwierig macht. Zudem sind die Daten auf Deutschland und Österreich begrenzt.

Tabelle 7.6.: Housenumbervalidator

7.2. Entscheid: Fehlerdaten von KeepRight

Durch die gute Eignung der Fehlerdaten vom *KeepRight*-Dienst, haben wir uns dazu entschieden, deren Daten als Basis für unsere Fehler zu verwenden. Der Dienst liefert zuverlässig Daten in hoher Qualität. Die Fehlermeldungen sind klar und alle Fehler sind bereits kategorisiert, was es einfach macht, diese für die grafische Darstellung zu unterscheiden.

Ein grosses Plus ist auch, dass sich die Daten auf *OpenStreetMap* abstützen, so dass eine Korrektur nur an einer Stelle vorgenommen werden muss.

7.3. Fehlerdaten in die Datenbank laden

Um stets aktuelle Daten für die App anzuzeigen, wird der Dump von *KeepRight* jede Nacht mit einem Shellskript in die Datenbank geladen. Das Shellskript befindet sich unter:

`/server/database/keepright_setup.sh`

Mittels Cronjob wird folgender Befehl aufgerufen:

```
$ setup_keepright_db.sh -o osm -n osm_bugs -s keepright
```

Parameter	Bedeutung
<code>-o osm</code>	Datenbankbenutzer welcher das Schema besitzen soll (<i>Owner</i>)
<code>-n osm_bugs</code>	Name der Datenbank
<code>-s keepright</code>	Name des Schemas

Tabelle 7.7.: Parameter für `setup_keepright.sh`

7.3.1. Datenformat

Auf der Webseite von *KeepRight* ist ersichtlich², welche Daten im Dump enthalten sind.

Feld	Bedeutung
<code>schema</code>	Schemabezeichner welcher die Region gemäss Planet-Karte angibt
<code>error_id</code>	Fehler-ID pro Schema
<code>error_type</code>	Typisierung des Fehlers
<code>error_name</code>	Name des Fehlertyps
<code>object_type</code>	<i>OpenStreetMap</i> -Typ des Objekts (Node , Way oder Relation)
<code>object_id</code>	<i>OpenStreetMap</i> -ID des Objekts
<code>state</code>	Status des Fehlers (new, reopened, ignore_temporarily, ignore)
<code>msgid</code>	Fehlermeldung mit Platzhaltern (\$1 - \$5)
<code>txt1</code> <code>txt2</code> <code>txt3</code> <code>txt4</code> <code>txt5</code>	Texte für Platzhalter in msgid

²<http://www.keepright.at/interfacing.php>

<code>first_occurrence</code>	Zeitpunkt, an dem dieser Fehler das <i>erste</i> Mal entdeckt wurde
<code>last_checked</code>	Zeitpunkt, an dem dieser Fehler das <i>letzte</i> Mal entdeckt wurde
<code>object_timestamp</code>	Letzter Änderungszeitpunkt am Objekt
<code>user_name</code>	Benutzer, welcher das Objekt zuletzt verändert hat
<code>lat</code> <code>lon</code>	Längen- und Breitengrad des Fehlers
<code>comment</code>	Kommentar zum Fehler
<code>comment_timestamp</code>	Zeitpunkt des Kommentars zum Fehler

Tabelle 7.8.: Datenformat der KeepRight-Daten

Diese Daten sind in der Tabelle `keepright.errors` enthalten. Ein Subset davon wird für die Applikation benötigt.

7.3.2. Internationalisierung

Eine Anforderung an KORT war es, ein deutsches UI anzubieten und daneben die App so weit vorzubereiten, dass sich diese internationalisieren lässt. Dies schliesst selbstverständlich die Texte aus der Datenbank ein. Alle Fehlerdaten werden ohne Übersetzung aus der Datenbank gelesen und anschliessend mit den Werten aus den Datenbank-spezifischen Properties-Files, versehen. Die Dateien befinden sich im Verzeichnis `resources/i18n/`.

8. Architektur

Das Gesamtsystem setzt sich aus insgesamt vier Komponenten zusammen: der Datenbank, dem Webserver, der Fehlerquelle und dem Zielsystem von *OpenStreetMap*. Die einzelnen Komponenten sind über REST-Schnittstellen miteinander verbunden. Dabei sind das Zielsystem (*OpenStreetMap*) und die Fehlerquelle (*KeepRight*, siehe Kapitel 7) Fremdsysteme, bei welchen die Schnittstellen gegeben waren. Unsere eigenen Server haben wir entsprechend angepasst und auch via REST zugänglich gemacht.

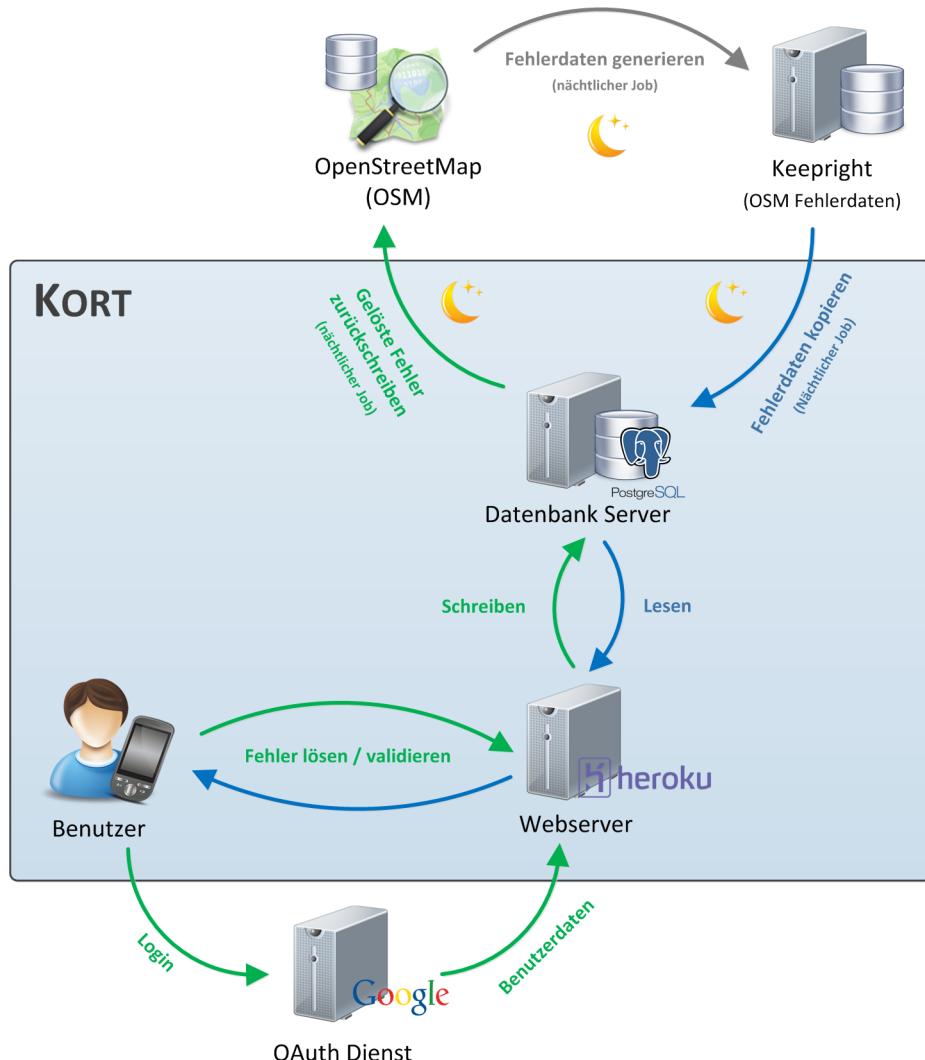


Abbildung 8.1.: Übersicht des Gesamtsystems

Das Backend besteht aus zwei logisch und derzeit auch physisch getrennten Servern. Der Webserver liefert die [Web-App](#) aus und ist auch der einzige Kommunikationspartner für das Frontend. Dies ist zum einen eine architektonische, zum anderen eine technische Entscheidung.

- Das Frontend muss sich nicht darum kümmern, woher es welchen Dienst bezieht
- Die Same-Origin-Policy[14] einiger Server lässt keine direkte Kommunikation zwischen *fremdem* JavaScript und dem Server zu

Der Webserver ist somit der Dreh- und Angelpunkt der Applikation, jegliche Informationen von und zum Frontend durchläuft diese zentrale Komponente.

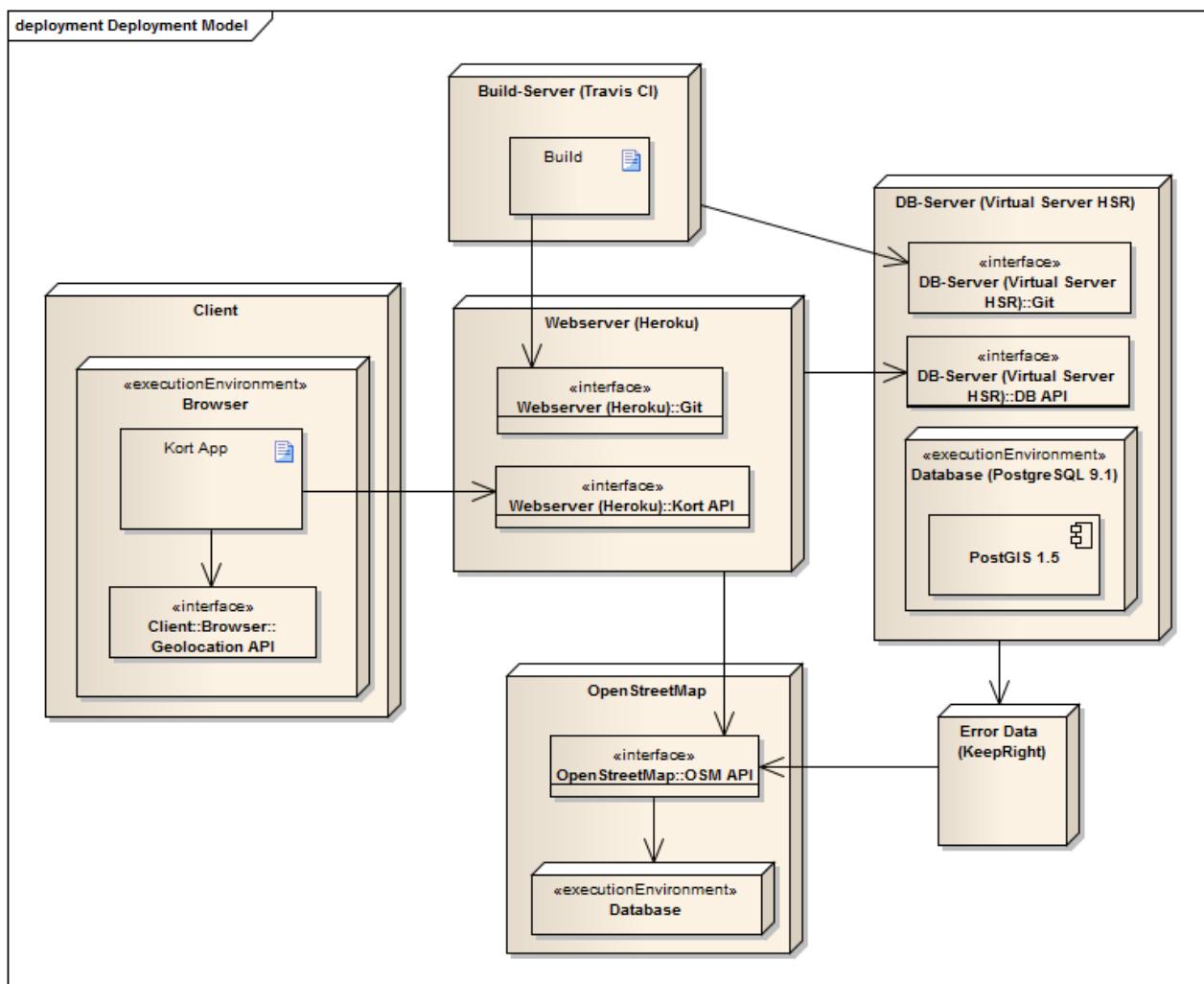


Abbildung 8.2.: Deployment-Diagramm

8.1. Bootstrapping

Das [Bootstrapping](#) ist ein sehr wichtiges Grundprinzip dieser Arbeit. Es besagt, dass sich das System selbst aufbauen kann. Konkret haben wir darauf geachtet, dass für alle Aktionen Skripte oder Build-Schritte vorhanden sind, um diese später nachzuvollziehen. Das Ziel ist es, ein System zu entwickeln, dass sich möglichst einfach wieder aufbauen lässt.

Auf der Seite des Datenbankservers geschieht dies so, dass die Datenbank jede Nacht neu aufgebaut wird. Zum einen werden dabei die neuesten Fehlerdaten von *KeepRight* geladen, zum anderen alle Änderungen an der Datenbank nachvollzogen. Dies hat den grossen Vorteil, dass stets ein konsistenter Zustand anzutreffen ist.

Eine Ausnahme bilden dabei die laufenden Daten der Applikation (d.h. Userdaten, Lösungsvorschläge etc.), welche nicht jede Nacht zurückgesetzt werden. Dies hat aber eher praktische als technische Gründe.

Auf der Seite des Webservers ist das [Bootstrapping](#) noch stärker anzutreffen, da bei jedem Build das System komplett neu aufgebaut wird. Somit müssen alle Informationen in Skripten hinterlegt sein, da sie sonst schlicht nicht zur Anwendung kommen.

8.2. Umsysteme

Zu unserem System gehören auch die Fehlerdatenquelle *KeepRight* sowie *OpenStreetMap* dazu. Unsere Applikation interagiert direkt mit diesen beiden Systemen, ohne eine Kontrollfunktion zu übernehmen. Es handelt sich somit um sogenannte Fremdsysteme.

Bei der Gestaltung der Systemlandschaft war es uns wichtig, diese so flexibel und offen wie möglich zu halten. Weitere Systeme sollen sich einfach einbinden lassen. Um dieses Ziel zu erreichen, haben wir darauf geachtet keine festen Verbindungen zwischen den Systemen zu schaffen. Solche lassen sich später nur schwer wieder entfernen oder ändern.

Ein wichtiger Eckpfeiler ist es auch, die gesamte Kommunikation über [REST](#)-Schnittstellen zu realisieren. Dadurch lassen sich Dienste sowohl orts-, wie auch technologie-unabhängig voneinander betreiben.

8.2.1. KeepRight

Die Verbindung zu *KeepRight* ist sehr lose, da das angebotene [API](#) lediglich ein täglich generierter Dump von Fehlerdaten ist. Diese Daten werden jede Nacht heruntergeladen und in unsere Datenbank integriert.

Weitere Fehlerquellen lassen sich einfach einbinden (siehe Abschnitt [11.1](#)), da innerhalb der Applikation nur über eine View auf die Daten zugegriffen wird.

8.2.2. OpenStreetMap

Das System benötigt drei Schnittstellen zu *OpenStreetMap*, wovon in dieser Arbeit zwei realisiert sind.

OpenStreetMap liefert die genauen Informationen über ein geografisches Objekt, um dieses auf einer Karte zeichnen zu können. Dies verwenden wir, um Details zu einem Fehler anzuzeigen. Wenn beispielsweise der Name einer Strasse erfasst werden soll, kann sich der Benutzer vorher über die Detail-Karte vergewissern, welche Strasse genau gemeint ist.

Der zweite Dienst, den wir beanspruchen, ist die Authentifizierung über [OAuth](#). So kann das *OpenStreetMap*-Benutzerkonto direkt verwendet werden, um KORT zu spielen.

Um den Kreis zu schliessen, sollen Korrekturen an *OpenStreetMap* zurückgeschickt werden. Dieses Themengebiet ist sehr heikel, da die *OpenStreetMap*-Community technischen Benutzern sehr kritisch gegenübersteht¹. Es gilt die Grundregel, dass alle Änderungen über persönliche Benutzerkonten gemacht werden müssen.

Dies hat vor allem damit zu tun, dass Änderungen dadurch leichter nachvollziehbar sind. Solche Einschränkungen können jedoch auch ein Hindernis sein. In begründeten Fällen wird deshalb auch eine Ausnahme zugelassen. Beispielsweise hat die *WheelMap*-Applikation² offiziell die Erlaubnis, mit einem technischen Benutzer Änderungen zu erfassen³.

Dies war unter anderem auch der Grund, weshalb wir unser Projekt am OSM-Stammtisch in Zürich vorgestellt haben⁴. Das Feedback war durchaus positiv, jedoch gab es starke Vorbehalte gegenüber unserem Vorhaben, die korrigierten und validierten Lösungen automatisiert an *OpenStreetMap* zu senden.

Das Thema ist also sowohl technisch und organisatorisch als komplex einzustufen. Dies war uns von Anfang an klar, weshalb wir diesen Punkt auch in unserem Risikomanagement (siehe Abschnitt 13.2) behandelt haben. Schlussendlich hat uns die Zeit gefehlt, dieses Feature zu implementieren, weshalb KORT derzeit ein geschlossenes System ist, welches Lösungsvorschläge für Fehlerdaten sammelt.

8.3. Authentifizierung

Bei jeder Applikation, welche Daten für verschiedene Nutzer speichert, stellt sich die Frage, wie sich die User anmelden sollen. Die wichtigsten Kriterien dabei sind:

- Sicherheit
- Einfachheit

¹<http://lists.openstreetmap.org/pipermail/talk/2011-May/058396.html>

²<http://wheelmap.org/>

³<http://wiki.openstreetmap.org/wiki/Talk:Wheelmap>

⁴http://wiki.openstreetmap.org/wiki/DE:Switzerland:Z%C3%BCrich/OSM-Treffen#36._OSM-Stammtisch

- Zielpublikum
- Benötigte Zusatzdienste

Auf diese Kriterien hin, war für uns sofort klar, dass wir keine eigene Benutzerverwaltung machen wollten. Ein App, das primär eine Ablenkung für einige Minuten sein soll, kann es sich nicht leisten, den Benutzer durch die Eingabe von Anmeldedaten abzuschrecken.

Daneben hat ein Benutzer bereits sehr viele Benutzerkonten bei verschiedenen Diensten. Daher ist es auch naheliegend, diese Daten wiederzuverwenden. Der Benutzer erspart sich dadurch ein weiteres Login, an das er sich erinnern muss.

[OAuth](#) ist mittlerweile ein gängiger Standard, um Applikationen Zugriff auf fremde Ressourcen zu geben, ohne direkt den Benutzernamen und das Passwort preiszugeben. Auch *OpenStreetMap* unterstützt [OAuth](#) und so fiel die Entscheidung leicht, dieses Protokoll zu verwenden.

In unserem Fall ist die Resource jedoch „nur“ die Benutzerinformationen. Das Protokoll basiert darauf, dass sowohl die Applikation, sowie der Benutzer dem jeweiligen [OAuth](#)-Anbieter vertraut.

Vom authentisierten Benutzer werden dann die Benutzerinformationen (Name, ID, E-Mail) in unsere Datenbank übernommen. Jeder Benutzer erhält daraufhin ein sogenanntes *secret*, welches ihm fortan erlaubt, ohne erneuten Login die App zu nutzen. Eine genaue Erklärung des Login-Ablaufs wird in Abschnitt [10.4.1](#) beschrieben.

8.4. REST

[REST](#) ist die Idealvorstellung von plattformunabhängigen Schnittstellen. So lassen sich alle möglichen Entitäten als *Ressourcen* darstellen, welche über eine der vier Grundfunktionalitäten (CRUD[11] - **C**reate, **R**ead, **U**pdate, **D**elete) manipuliert werden können.

Jede grössere Applikation kann von dieser Flexibilität profitieren. Sie ermöglicht es, dass Funktionalitäten mit beliebigen Technologien und Programmiersprachen entwickelt werden können. Dies ist besonders gut umsetzbar, wenn die Applikation bereits in Betrieb ist. So können einzelne Komponenten neu entwickelt und in das bestehende System integriert werden, ohne eine nennenswerte Downtime.

Beim [Marshalling](#) der Daten mag es effizientere Möglichkeiten geben als JSON[1] und XML[12]. Wenn die Datenmengen gering sind, fällt dies jedoch nicht stark ins Gewicht. Sobald Binärdaten übertragen werden müssen, könnte es sinnvoll sein, einige Komponenten über andere Schnittstellen anzusprechen.

In unserem System spielt das keine Rolle, da unsere Requests durchschnittlich 10KB gross sind.

9. Infrastruktur

9.1. Datenbankserver

Beim Datenbankserver handelt es sich um einen virtuellen Server, den uns die HSR Hochschule für Technik Rapperswil für die Dauer dieser Arbeit zur Verfügung gestellt hat. Dieser Server enthält die Installation der KORT-Datenbank sowie das Projektmanagementtool Redmine. Letzteres ist die einzige kritische Anwendung auf dem Server, da sich der Rest über entsprechende Installationskripts sehr einfach und schnell neu aufbauen lässt.

Die Installation der benötigten Software kann mit dem Ubuntu Standard-Mechanismus `apt-get install` durchgeführt werden.

Name	sinv-56055.edu.hsr.ch
DNS CNAME	kort.rdmr.ch
Art des Servers	Virtueller Server
Betriebssystem	Ubuntu 12.04 (LTS)
Zugriff	Root-Zugriff via SSH
Installierte Software	PostgreSQL 9.1, PostGIS 2.0, Redmine 2.1, MySQL 5.5, Apache Websever mit PHP 5.4
Pfade	Repository: /home/odi/kort Redmine: /home/redmine/redmine-2.1.0

Tabelle 9.1.: Datenbankserver

9.1.1. Datenbank-Webservice

Der Zugang auf die Datenbank von aussen läuft ausschliesslich über den REST-Webservice. Für den Betrieb dieses Dienstes ist eine Apache Webserver Installation mit PHP-Unterstützung (mindestens PHP 5.3) erforderlich.

Für den Betrieb des Webservices ist das KORT-Repository auf dem Server geklont. Anschliessend muss noch ein Symlink angelegt werden, um das Skript korrekt aufrufen zu können:

```
$ ln -s /home/odi/kort/server/webservices /var/www/webservices
```

Der Webservice wird in Abschnitt [10.2.8](#) genauer beschrieben.

9.1.2. Redmine

Die Installation von Redmine verlangt es, einen neuen Benutzer `redmine` auf dem Server anzulegen. Danach muss die Software nur noch auf den Server kopiert werden und der Installationsanleitung gefolgt werden. Diese ist in der Datei `/server/redmine/redmine_install.md` zu finden.

Backup

Die Daten von Redmine werden über zwei Skripts täglich gesichert. Das Skript `redmine_backup.sh` kümmert sich darum, alle Backup-Daten täglich auf dem virtuellen Server zu sammeln und an einem zentralen Ort zu speichern.

Mit dem zweiten Skript `sync_backup.sh` kann dann das zuvor erstellte Backup auf beliebige andere Systeme verteilt werden. In unserem Fall haben wir das Skript auf unseren Laptops eingerichtet und so täglich die aktuellen Daten gesichert.

9.2. Webserver (Heroku)

Bei *Heroku*¹ handelt es sich um einen kostenlosen Dienst, welcher eine Deploymentumgebung für verschiedene Plattformen anbietet. Der Dienst hat eine Schnittstelle, über welche sich automatisiert Applikationen erstellen lassen. Die Datenübertragung läuft dann über [Git](#).

Art des Servers	Server in der Cloud
Betriebssystem	Ubuntu
Zugriff	Daten via Git , Befehle via Kommandozeilen-API (<i>Heroku-Toolbelt</i>)
Installierte Software	Apache, KORT Web-App

Tabelle 9.2.: Server bei Heroku

Die Entscheidung, den Webserver von *Heroku* zu wählen, ist dadurch begründet, dass uns dies die grösstmögliche Freiheit bietet. *Heroku* bietet bereits eine hervorragende [API](#), welche sich über das Kommandozeilen-Toolset *Heroku-Toolbelt* steuern lässt. Dies erlaubt es, beliebig viele Applikationen automatisiert zu erstellen. Auch für den Betrieb bietet das [API](#) viele Möglichkeiten zur Fernwartung an (SSH-Zugang, Logs, Prozessorauslastung).

¹<http://www.heroku.com/>

9.3. Deployment

Am Deployment der Applikation sind mehrere Systeme beteiligt. Alle Änderungen werden von den Entwicklern via *Git* zu *GitHub*² übertragen. Auf *GitHub* gibt es sogenannte *Hooks*, die man aktivieren kann. Dabei handelt es sich um weitere Aktionen, welche durch verschiedene Ereignisse ausgelöst werden können. In unserem Fall haben wir einen *post-commit Hook* aktiviert, welcher dem *Continuous Integration* Dienst *Travis-CI*³ Bescheid gibt, wenn neue Änderungen auf *GitHub* eingetroffen sind.

Nach einem erfolgreichen Build werden die Resultate an *Heroku* gesendet. Zusätzlich wird der Datenbankserver über die Änderungen notifiziert, worauf dieser sich selbst aktualisiert.

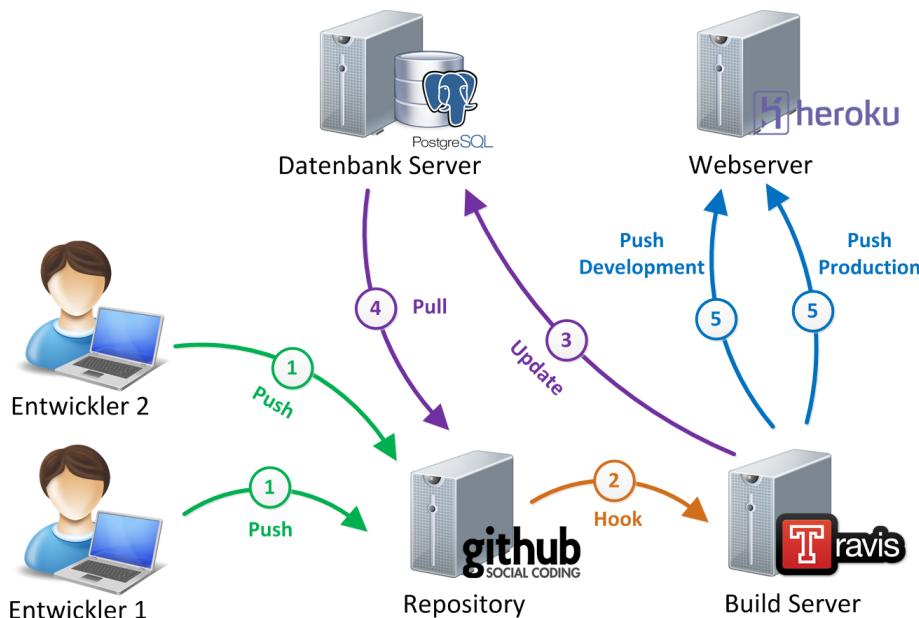


Abbildung 9.1.: Entwicklungsprozess von KORT

9.3.1. Travis CI

Auf *Travis* läuft der Build, welcher durch die Konfigurationsdatei `.travis.yml` gesteuert ist. Darin lassen sich die Schritte sowie die Umgebung für Builds definieren. Für jede Umgebung wird ein separater Build ausgelöst. Somit lassen sich bequem verschiedene Versionen mit unterschiedlichen Umgebungen testen. Daraus entsteht eine sogenannte Build-Matrix, welche bei jedem Build durchlaufen wird (siehe Tabelle 9.3).

²<http://github.com>

³<http://travis-ci.org>

	Test	Produktion
PHP 5.3	Build und Test	Build und Test
PHP 5.4	Build, Test und Deployment auf http://kort-dev.herokuapp.com	Build, Test und Deployment auf http://kort.herokuapp.com

Tabelle 9.3.: Build-Matrix von Travis CI

Ein Travis-Build läuft immer in einer neuen virtuellen Umgebung, so dass strikt nach dem Prinzip des *Bootstrappings* vorgegangen werden muss. Das bedeutet, es muss möglich sein, die Applikation ohne Vorkenntnisse zu installieren. Dies hilft, den Installationsprozess genau festzuhalten.

Am Ende des Build-Vorgangs wird die [Web-App](#) schliesslich zu Heroku übertragen.

9.3.2. Konfiguration über `.travis.yml`

Die `.travis.yml` Datei ist die Konfigurationsdatei von *Travis CI*. Darin wird die Build-Matrix festgelegt sowie die einzelnen Schritte des Builds definiert.

Vor dem eigentlichen Build wird die Umgebung aufgesetzt und die benötigte Software installiert (Zeilen 19-34 in Code-Ausschnitt [9.1](#)).

Build-Matrix

Die Build-Matrix bestimmt, welche Builds mit welchen Parametern ausgeführt werden. Dabei wird gemäss definierten Sprachversionen (Zeilen 3-5 im Code-Ausschnitt [9.1](#)) und Umgebungsvariablen (Zeilen 15-17) ein Build ausgeführt. Daneben gibt es noch eine Reihe von globalen Umgebungsvariablen, welche in allen Builds verwendet werden. In unserem Fall werden also vier Builds ausgeführt, mit jeweils PHP 5.3 und PHP 5.4 für die Test- und die Produktionsumgebung.

Verschlüsselte Umgebungsvariablen

Die Einträge mit *secure* sind verschlüsselte Einträge, welche bei der Ausführung des Builds wieder entschlüsselt werden⁴. Auf diese Art lassen sich geheime Informationen wie API-Schlüssel schützen.

In unserem `.travis.yml` kommt dies zweimal zum Einsatz:

1. Heroku API-Key, mit welchem sich beliebige Aktionen auf Heroku durchführen lassen
2. KORT DB API-Key, welcher den Datenbank-Webservice (siehe Abschnitt [10.2.8](#)) vor fremden Zugriffen schützt

⁴<http://about.travis-ci.org/docs/user/build-configuration/#Secure-environment-variables>

```
1 language: php
2
3 php:
4   - 5.3
5   - 5.4
6
7 env:
8   global:
9     - CI_HOME=`pwd`/odi86/kort
10    - DEPLOY="true"
11    - BUILD_DIR=`pwd`/build_heroku
12    - secure: "EIn+Rm70xX80KygBRBCaST0qFGcBMWu5kHdKqSx0mHRJYjxuMOJpKV+
rnyep\n+RsyFDx2Z9yKlqRRS4cpZh7M6wwC63EV46+7
aWtzzTjnbMZfVzLQA9EmaEU4\
nYMsKGtpQk2mhvaNKd3UbEpD10Zq74NnAY0zipx0102UymcFnZEc="
13    - secure: "c0mMBP4Uyk1RC0nfetZsX/NV4GhMBT+
AntUmlWxXS5Rj2yrNcmNc7320gNm5\nnCLSVRYyk7/8feyUEMznWrUn/62
htZp0tEBAWtXg86dgIZgH4HPy912pKuSsH\
nxZTHgjUJI7J0uyLG4ID9D5maVLE35UWag/NEtcRVy5QXLZ0rsOM="
14 matrix:
15   - TARGET_ENV="dev"
16   - TARGET_ENV="prod"
17
18 before_script:
19   - gem install sass
20   - gem install compass
21   - gem install jsduck
22   - wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh >/dev/
null 2>&1
23   - sudo npm install -g grunt >/dev/null 2>&1
24   - pear install PHP_CodeSniffer && phpenv rehash
25   - sudo pear channel-discover pear.survivethedeepend.com
26   - sudo pear channel-discover hamcrest.googlecode.com/svn/pear
27   - sudo pear channel-discover pear.phpdoc.org
28   - sudo pear install --alldeps deepend/Mockery
29   - sudo pear install phpdoc/phpDocumentor-alpha && phpenv rehash
30   - sudo apt-get install graphviz
31   - export PATH="$PATH:$CI_HOME:/usr/local/heroku/bin"
32   - curl http://kort.rdmr.ch/webservices/update/git
33   - mv $CI_HOME/server/php/WebService/Database/DbConfig.example.php $_
CI_HOME/server/php/WebService/Database/DbConfig.php
34
35 script: ant -f build_kort.xml build
36
37 after_script: bash $CI_HOME/server/heroku/heroku.sh
```

Code-Ausschnitt 9.1: Die Travis CI Konfigurationsdatei .travis.yml

9.3.3. Apache Ant

Das Build-Skript ist mit Apache Ant geschrieben. Es beinhaltet alle Targets und Kombinationen davon, um einen kompletten oder teilweisen Build durchzuführen.

Derzeit werden bei einem vollständigen Build folgende Schritte durchlaufen:

1. Aus den SCSS-Quelldateien wird CSS generiert
2. Die JavaScript- und PHP-Dateien werden auf ihren Code Style geprüft
3. Anschliessend wird die Code-Dokumentation generiert
4. Zum Schluss werden noch die Unit Tests ausgeführt

GruntJS

Die JavaScript spezifischen Build-Aufgaben werden von Ant an *GruntJS*⁵ abgegeben. Dieses Tool hat wiederum sein eigenes Konfigurationsfile `grunt.js`. Darin sind Tasks beschrieben, welche auf den Code angewendet werden sollen.

In unserem Fall brauchen wir Grunt für zwei Aufgaben: JavaScript-Tests ausführen und JSHint auf sämtlichen JavaScript Quellcode anwenden.

Bei *JSHint*⁶ handelt es sich um ein Linting-Werkzeug, welches mit diversen Optionen konfiguriert werden kann. Das Ziel ist, dass der Code einheitlich wird und ein gewisser Code-Standard eingehalten wird. Der Code wird dadurch besser lesbar und weniger fehleranfällig.

Die Unit Tests sind mit *QUnit*⁷ erstellt und werden mit dem headless Browser *PhantomJS*⁸ durchgeführt. Dies ermöglicht es, auf einfache Art und Weise Frontend-Tests während dem Build durchzuführen.

PHP_CodeSniffer (PHPCS)

Der *PHP_CodeSniffer*⁹ ist ein Linting-Tool für PHP. Unser Code verwendet den Standard PSR-2¹⁰, welcher noch mit einigen Regeln zu PHPDoc-Kommentaren angereichert wurde.

Dadurch ist sichergestellt, dass der Code sauber ist und keine unerwarteten Seiteneffekte auftreten. PHPCS lässt sich direkt in die Entwicklungsumgebung integrieren, so dass direkt beim Schreiben des Codes der Style überprüft werden kann.

⁵<http://gruntjs.com/>

⁶<http://www.jshint.com/docs/>

⁷<http://qunitjs.com/>

⁸<http://phantomjs.org/>

⁹<http://pear.php.net/manual/en/package.php.php-codesniffer.php>

¹⁰<https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md>

10. Backend

10.1. Implementation

Das Backend besteht neben der Datenbank und den REST-Schnittstellen vor allem aus PHP-Code. Der meiste Code entfällt auf das Handling von Webservice-Anfragen und die Authentifizierung mit OAuth.

10.1.1. Gliederung

Das Backend befindet sich im Verzeichnis `server/` im Repository. Das Backend teilt sich auf verschiedene Unterordner auf (siehe Tabelle 10.1).

Order	Inhalt
<code>database/</code>	SQL und Shell-Skripts für die Erstellung der Datenbank
<code>heroku/</code>	Shell-Skripte für das Deployment auf Heroku
<code>oauth2callback/</code>	Callback-Handler der verschiedenen OAuth-Dienste
<code>php/</code>	PHP-Klassen für das Backend
<code>redmine/</code>	Skripts und Anleitung für Redmine
<code>ssh_pub_keys/</code>	Öffentliche SSH-Schlüssel für das Deployment
<code>webservices/</code>	REST-Ressourcen (Endpunkte der Schnittstellen)

Tabelle 10.1.: Gliederung des Backends

Speziell zu erwähnen sind dabei die PHP-Klassen, welche die Logik des Backends abbilden. Sie sind über Namespaces aufgeteilt und liefern die Logik für alle anderen Teile des Backends. Um dies zu ermöglichen, gibt es die `ClassLoader`-Klasse¹. Falls irgendwo eine PHP-Klasse gebraucht wird, muss nur diese Klasse geladen werden, diese wiederum kümmert sich darum, alle abhängigen Dateien nachzuladen.

Für die Klassen gibt es eine separate Dokumentation (siehe Abschnitt 10.6).

¹<http://kort.herokuapp.com/docs/Kort-backend/classes/Kort.ClassLoader.html>

10.1.2. Abhangigkeiten

Library	Version	Verwendung
Slim	2.1.0	Micro-Framework fur die Implementation von REST-Schnittstellen
Google APIs Client Library	0.6.0	PHP-Library fur Google APIs
oauth-php	175	OAuth Library fur PHP
Ant-Contrib	1.0b3	Erweiterte Tasks fur Apache Ant

Tabelle 10.2.: Abhangigkeiten im Backend

10.2. REST-Schnittstellen

Die Entscheidung, REST-Schnittstellen zu verwenden, haben wir schnell gefasst. Zum einen ist damit eine einheitliche Schnittstelle im gesamten System vorhanden, so dass immer klar ist uber welchen Kanal eine Kommunikation stattfindet.

Zum anderen sind REST-Schnittstellen fur Webapplikation sehr einfach zu verwenden, da entsprechende Bibliotheksfunktionen bereits vorhanden sind. Schlussendlich ist entscheidend, dass REST-Schnittstellen ein grosses Mass an Plattformunabhangigkeit bieten und so die raumliche Teilung der Systeme erleichtern.

10.2.1. Slim Framework

Fur die Erstellung der REST-Webservices wurde das PHP Microframework *Slim*² verwendet. Das Framework ist sehr klein und beinhaltet nur das Notigste, um die Webservices zu implementieren. Bereits mit wenigen Zeilen Code lsst sich ein Webservice erstellen (siehe Code-Ausschnitt 10.1).

```
1 <?php
2 $app = new \Slim\Slim();
3 $app->get('/hello/:name', function ($name) {
4     echo "Hello, $name";
5 });
6 $app->run();
```

Code-Ausschnitt 10.1: Beispiel REST-Webservice in Slim

²<http://www.slimframework.com/>

10.2.2. Webservice: Antworten /answer

Bei einigen Fehlertypen wird eine Auswahl an möglichen Antworten vorgegeben. Um diese Antworten vorzuladen, wird der /answer-Webservice verwendet. Dieser liefert alle Antworten der verschiedenen Fehlertypen zurück.

Antworten laden

URL	http://kort.herokuapp.com/server/webservices/answer [/<type>]	
	<type> (optional) Antworten auf Typ beschränken	
Methode	GET	
Parameter	-	
Antwort	200 OK	Daten konnten erfolgreich geladen werden.
Antworttyp	JSON	

Tabelle 10.3.: Webservice Antworten (GET /answer)

Beispiel:

GET http://kort.herokuapp.com/server/webservices/answer/missing_track_type

Antwort:

```
{  
  "return": [  
    {  
      "id": "1",  
      "value": "grade1",  
      "title": "Asphalt, Beton oder Pflastersteine",  
      "sorting": "110",  
      "type": "missing_track_type"  
    },  
    { ... }  
  ]  
}
```

10.2.3. Webservice: Fehler /bug

Die Fehler werden über den /bug-Webservice geladen. Diesem muss die aktuelle Position mitgeliefert werden. Dadurch ist es möglich, lediglich die Fehler in der Umgebung zu laden.

Zusätzlich kann über diesen Webservice eine Lösung zu einem Fehler eingetragen werden.

Fehler laden

URL	<code>http://kort.herokuapp.com/server/webservices/bug/position/<lat>,<lng></code>	
	<p><code><lat></code> Latitude der aktuellen Position <code><lng></code> Longitude der aktuellen Position</p>	
Methode	GET	
Parameter	limit Maximale Anzahl der zu ladenden Fehler radius Radius in dem sich die Fehler befinden müssen	
Antwort	200 OK	Daten konnten erfolgreich geladen werden.
Antworttyp	JSON	

Tabelle 10.4.: Webservice Fehler (GET /bug)

Beispiel:

```
GET http://kort.herokuapp.com/server/webservices/bug/position/47.1,8.1?
limit=1&radius=5000
```

Antwort:

```
{
  "return": [
    {
      "id": "32621371",
      "schema": "95",
      "type": "missing_track_type",
      "osm_id": "119068810",
      "osm_type": "way",
      "title": "Typ des Wegs unbekannt",
      "description": "Um welchen Weg-Typ handelt es sich hier?",
      "latitude": "47.099585000000000000",
      "longitude": "8.097914000000000000",
      "view_type": "select",
      "answer_placeholder": "Typ",
      "fix_koin_count": "5",
      "txt1": "",
      "txt2": "",
      "txt3": "",
      "txt4": "",
      "txt5": ""
    },
    { ... }
  ]
}
```

Lösung senden

URL	http://kort.herokuapp.com/server/webservices/bug/fix	
Methode	POST	
Parameter	Die zu sendende Antwort muss als JSON-Objekt im Body gesendet werden.	
Antwort	200 OK	Die Lösung konnte erfolgreich gesendet werden. Als Antwort werden die erspielten Punkte und Auszeichnungen zurückgeliefert.
	403 Forbidden	Der Benutzer ist nicht korrekt eingeloggt und kann somit keine Daten an den Server senden.
	400 Bad request	Das gesendete JSON ist nicht valide oder es gab einen Fehler beim Schreiben der Daten in die Datenbank.
Antworttyp	JSON	

Tabelle 10.5.: Webservice Fehler (POST /bug/fix)

Beispiel:

```
POST http://kort.herokuapp.com/server/webservices/bug/fix
```

```
{
  "id": "ext-record-230",
  "user_id": 3,
  "error_id": "28704192",
  "schema": "95",
  "osm_id": 1611867263,
  "message": "McDonalds"
}
```

Antwort:

```
{
  "badges": [
    {
      "name": "highscore_place_1"
    }
  ],
  "koin_count_new": "15",
  "koin_count_total": "55"
}
```

10.2.4. Webservice: Highscore /highscore

Über den /highscore-Webservice können die Benutzer sortiert nach Anzahl *Koins* geladen werden.

Highscore laden

URL	http://kort.herokuapp.com/server/webservices/highscore	
Methode	GET	
Parameter	limit Maximale Anzahl der Benutzer	
Antwort	200 OK	Daten konnten erfolgreich geladen werden.
Antworttyp	JSON	

Tabelle 10.6.: Webservice Antworten (GET /highscore)

Beispiel:

GET <http://kort.herokuapp.com/server/webservices/highscore?limit=10>

Antwort:

```
{
  "return": [
    {
      "user_id": "3",
      "username": "tschortsch",
      "koin_count": "140",
      "fix_count": "12",
      "vote_count": "4",
      "ranking": "1",
      "you": true
    },
    { ... }
  ]
}
```

10.2.5. Webservice: OpenStreetMap /osm

Um *OpenStreetMap*-Objekte auf der Karte anzuzeigen, werden über den /osm-Webservice die entsprechenden OSM-Daten geladen. Der Webservice leitet den Request an das *OSM API*³ weiter und sendet das Resultat an die Webapplikation zurück.

OpenStreetMap Objekt laden

URL	<code>http://kort.herokuapp.com/server/webservices/osm/<type>/<id></code>	
	<type> OSM-Objekttyp <id> ID des OSM-Objekts	
Methode	GET	
Parameter	-	
Antwort	200 OK	Daten konnten erfolgreich geladen werden.
Antworttyp	XML	

Tabelle 10.7.: Webservice OpenStreetMap (GET /osm)

Beispiel:

GET `http://kort.herokuapp.com/server/webservices/osm/node/1658024260`

Antwort:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap server" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/odbl/1-0/">
  <node id="1658024260" version="1" changeset="10861664" lat="47.5114378" lon="8.5443127" user="pfrauenf" uid="479871" visible="true" timestamp="2012-03-03T20:05:48Z">
    <tag k="amenity" v="fast_food"/>
  </node>
</osm>
```

³http://wiki.openstreetmap.org/wiki/API_v0.6

10.2.6. Webservice: Benutzer /user

Der /user-Webservice dient zur Authentifizierung des Benutzers. Über ihn können sich die Benutzer an- und abmelden. Zudem werden die Benutzerdaten darüber geladen.

Benutzerdaten laden

URL	<code>http://kort.herokuapp.com/server/webservices/user/[<secret>]</code>	
	<secret> (optional) User Secret wird gesendet falls der Benutzer bereits eingeloggt ist.	
Methode	GET	
Parameter	-	
Antwort	200 OK	Daten konnten erfolgreich geladen werden. Der Webservice liefert die Benutzerdaten zurück.
Antworttyp	JSON	

Tabelle 10.8.: Webservice Benutzer (GET /user)

Beispiel:

```
GET http://kort.herokuapp.com/server/webservices/user
```

Antwort:

```
{
  "return": {
    "id": "3",
    "name": "J\u00fcrg Hunziker",
    "username": "tschortsch",
    "oauth_user_id": "email@host.com",
    "oauth_provider": "Google",
    "token": null,
    "fix_count": "2",
    "vote_count": "4",
    "koin_count": "40",
    "secret": "secret",
    "pic_url": "http://www.gravatar.com/avatar/secret?s=200&d=mm&r=r",
    "logged_in": true
  }
}
```

Badges eines Benutzers laden

URL	<a href="http://kort.herokuapp.com/server/webservices/user/<id>/badges">http://kort.herokuapp.com/server/webservices/user/<id>/badges	
	<id> ID des Benutzers	
Methode	GET	
Parameter	-	
Antwort	200 OK	Daten konnten erfolgreich geladen werden. Der Webservice liefert alle Badges zurück mit der Angabe, ob der Benutzer ihn gewonnen hat oder nicht.
Antworttyp	JSON	

Tabelle 10.9.: Webservice Benutzer (GET /user/<id>/badges)

Beispiel:

```
GET http://kort.herokuapp.com/server/webservices/user/3/badges
```

Antwort:

```
{
  "return": [
    {
      "id": "1",
      "name": "highscore_place_1",
      "title": "1. Rang",
      "description": "Erster Rang in der Highscore erreicht.",
      "color": "#FFFBCB",
      "sorting": "110",
      "won": true,
      "create_date": "13.12.2012 18:56"
    },
    { ... }
  ]
}
```

Logout

URL	<code>http://kort.herokuapp.com/server/webservices/user/<id>/logout</code>	
	<id> ID des Benutzers	
Methode	GET	
Parameter	-	
Antwort	200 OK	Der Benutzer wurde erfolgreich ausgeloggt.
Antworttyp	Text	

Tabelle 10.10.: Webservice Benutzer (GET /user/<id>/logout)

Beispiel:

```
GET http://kort.herokuapp.com/server/webservices/user/<id>/logout
```

Antwort:

Congratulations! You've now officially logged out!

Benutzerdaten ändern

URL	<code>http://kort.herokuapp.com/server/webservices/user/[<id>]</code>	
	<id> ID des Benutzers	
Methode	PUT	
Parameter	Die neuen Benutzerdaten müssen als JSON-Objekt im Body gesendet werden.	
Antwort	200 OK	Der Benutzer wurde erfolgreich aktualisiert.
Antworttyp	-	

Tabelle 10.11.: Webservice Benutzer (PUT /user)

Beispiel:

```
PUT http://kort.herokuapp.com/server/webservices/user/3
```

```
{
  "logged_in":true,
  "id":"3",
  "username":"tschortsch",
  "oauth_user_id":"user@oauth.com",
  "oauth_provider":"Google",
```

```
"pic_url": "http://www.gravatar.com/avatar/1234?s=200&d=mm&r=r",
"name": "J\u00fcrg Hunziker",
"token": null,
"fix_count": 4,
"vote_count": 7,
"koin_count": 85,
"secret": "secret"
}
```

Antwort:

```
{
  "user_id": "3",
  "name": "J\u00fcrg Hunziker",
  "username": "tschartsch",
  "oauth_user_id": "user@oauth.com",
  "secret": "secret"
}
```

10.2.7. Webservice: Überprüfung /validation

Die eingetragenen Lösungen werden über den /validation-Webservice geladen. Diesem muss die aktuelle Position mitgeliefert werden, um lediglich die Lösungen in der Umgebung zu laden.

Zusätzlich kann über diesen Webservice eine Überprüfung zu einer Lösung gesendet werden.

Lösungen laden

URL	<code>http://kort.herokuapp.com/server/webservices/validation/position/<lat>,<lng></code>	
	<lat> Latitude der aktuellen Position <lng> Longitude der aktuellen Position	
Methode	GET	
Parameter	limit Maximale Anzahl der zu ladenden Lösungen radius Radius, in dem sich die Lösungen befinden müssen	
Antwort	200 OK	Die Lösungen konnten erfolgreich geladen werden.
Antworttyp	JSON	

Tabelle 10.12.: Webservice Überprüfung (GET /validation)

Beispiel:

```
GET http://kort.herokuapp.com/server/webservices/validation/position/47.1,8.1?
limit=1&radius=5000
```

Antwort:

```
{
  "return": [
    {
      "id": "186",
      "type": "missing_maxspeed",
      "view_type": "number",
      "fix_user_id": "1",
      "osm_id": "110725957",
      "osm_type": "way",
      "title": "Fehlendes Tempolimit",
      "fixmessage": "50",
      "question": "Darf man auf dieser Strasse mit dieser Geschwindigkeit fahren?",
      "latitude": "47.337045600000000000",
      "longitude": "8.520785600000000000",
      "upratings": "0",
      "downratings": "0",
      "required_validations": "3"
    },
    { ... }
  ]
}
```

Überprüfung eintragen

URL	http://kort.herokuapp.com/server/webservices/validation/vote	
Methode	POST	
Parameter	Die zu sendende Überprüfung muss als JSON-Objekt im Body gesendet werden.	
Antwort	200 OK	Die Überprüfung konnte erfolgreich eingetragen werden. Als Antwort werden die erspielten Punkte und Auszeichnungen zurückgeliefert.
	403 Forbidden	Der Benutzer ist nicht korrekt eingeloggt und kann somit keine Daten an den Server senden.
	400 Bad request	Das gesendete JSON ist nicht valide oder es gab einen Fehler beim Schreiben der Daten in die Datenbank.
Antworttyp	JSON	

Tabelle 10.13.: Webservice Überprüfung (POST /validation/vote)

Beispiel:

```
POST http://kort.herokuapp.com/server/webservices/validation/vote

{
  "id": "ext-record-177",
  "fix_id": 151,
  "user_id": "3",
  "valid": "true"
}
```

Antwort:

```
{
  "badges": [
    {
      "name": "vote_count_10"
    }
  ],
  "koin_count_new": "5",
  "koin_count_total": "95"
}
```

10.2.8. Webservice: Datenbank /db

Die Kommunikation vom Webserver zum Datenbankserver geschieht über den /db-Webservice. Als zugehörige Ressource kann eine Tabelle und deren Felder angegeben werden.

Alternativ können eine Reihe von SQL-Queries in einer Transaktion auf der Datenbank laufen gelassen werden. Dazu gibt es die spezielle *transaction*-Ressource.

Da dieser Webservice direkt die Daten auf der Datenbank verändern kann, ist er durch einen API-Key vor fremden Zugriffen geschützt. Der API-Key muss sowohl auf dem Webserver, wie auch auf dem Datenbankserver in der Umgebungsvariable KORT_DB_API_KEY definiert sein.

SELECT-Abfrage

URL	<code>http://kort.rdmr.ch/webservices/db/<table>/<fields></code>	
	<table> Datenbank-Tabellenname (inkl. Schema) <fields> Komma-separierte Liste von Feldern der Tabelle	
Methode	GET	
Parameter	key API-Key where WHERE-Klausel der Abfrage (Einschränkung) orderby ORDER BY-Klausel der Abfrage (Sortierung) limit Maximale Anzahl Records	
Antwort	200 OK	Die Datenbankabfrage war erfolgreich.
	403 Forbidden	Der API-Key ist nicht korrekt.
Antworttyp	JSON	

Tabelle 10.14.: Webservice Datenbank (GET /db)

Beispiel:

```
GET http://kort.rdmr.ch/webservices/db/kort.answer/id,value,title?
where=type='missing_track_type'
```

Antwort:

```
{
  [
    {
      "id": "1",
      "value": "grade1",
      "title": "Asphalt, Beton oder Pflastersteine"
    },
    { ... }
  ]
}
```

UPDATE-Abfrage

URL	<code>http://kort.rdmr.ch/webservices/db/<table>/<fields></code>	
	<table> Datenbank-Tabellenname (inkl. Schema) <fields> Komma-separierte Liste von Feldern der Tabelle	
Methode	PUT	
Parameter	key API-Key where WHERE-Klausel der Abfrage (Einschränkung) return Komma-separierte Felder welche als Antwort geschicht werden	
Antwort	200 OK	Die Datenbankabfrage war erfolgreich.
	403 Forbidden	Der API-Key ist nicht korrekt.
Antworttyp	JSON	

Tabelle 10.15.: Webservice Datenbank (PUT /db)

Beispiel:

```
PUT http://kort.rdmr.ch/webservices/db/kort.user/user_id,username?  

where=user_id=3&return=user_id,username'  
  

{  

  "user_id":3,  

  "username":"testuser"  
  

}
```

Antwort:

```
{  

  [  

    {  

      "id": "3",  

      "username": "testuser"  

    }  

  ]  

}
```

INSERT-Abfrage

URL	<code>http://kort.rdmr.ch/webservices/db/<table>/<fields></code>	
	<table> Datenbank-Tabellenname (inkl. Schema) <fields> Komma-separierte Liste von Feldern der Tabelle	
Methode	POST	
Parameter	key API-Key return Komma-separierte Felder welche als Antwort geschickt werden	
Antwort	200 OK	Die Datenbankabfrage war erfolgreich.
	403 Forbidden	Der API-Key ist nicht korrekt.
Antworttyp	JSON	

Tabelle 10.16.: Webservice Datenbank (POST /db)

Beispiel:

```
POST http://kort.rdmr.ch/webservices/db/kort.fix/user_id,error_id,osm_id,  
schema,message?return=fix_id'
```

```
{
  "user_id":5,
  "error_id":8983274,
  "osm_id":212342,
  "schema":"95",
  "message":"grade1"

}
```

Antwort:

```
{
  [
    {
      "fix_id":43
    }
  ]
}
```

Transaction

URL	http://kort.rdmr.ch/webservices/db/transaction	
Methode	POST	
Parameter	key API-Key	
Antwort	200 OK	Die Transaktion war erfolgreich.
	403 Forbidden	Der API-Key ist nicht korrekt.
	404 Not Found	Eine oder mehrere Abfragen der Transaktion waren fehlerhaft.
Antworttyp	JSON	

Tabelle 10.17.: Webservice Datenbank (POST /db/transaction)

Beispiel:

```
POST http://kort.rdmr.ch/webservices/db/transaction
```

```
{
  [
    {
      "type": "SQL",
      "sql": "select vote_koin_count from kort.validations where required_validations > 1"
    },
    {
      "type": "UPDATE",
      "fields": "username",
      "table": "kort.user",
      "where": "user_id=7",
      "returnFields": "user_id,username",
      "data": {
        "username": "newUsername"
      }
    }
  ]
}
```

Antwort:

```
{
  [
    {
      [
        [
          {
            "vote_koin_count": 5
          },
          {
            "vote_koin_count": 10
          }
        ]
      ]
    }
  ]
}
```

```
        { ... }
    ],
},
{
  "user_id":7,
  "username":"newUsername"
}
]
```

10.3. Datenbank

Die Datenbank ist in verschiedenen Schemas organisiert. Das Schema `kort` beinhaltet die Daten und Views für die Applikation.

10.3.1. Views

Als Grundregel verwenden wir immer Views als [API](#), gegen das wir programmieren. So lassen sich Änderungen daran sehr leicht einbringen, ohne dafür eine Tabelle anzupassen.

Jedes Model (siehe Abschnitt 4.2.3) in der [Web-App](#) hat seine eigene View in der Datenbank, welche genau die Daten liefert, welche es braucht. Dies reduziert den Code für das Frontend und verschiebt die Datenlogik in die Datenbank.

10.3.2. Applikationsschema

Um die Applikationslogik möglichst von der Datenbank unabhängig zu halten, wird vom Backend-Code aus, lediglich auf die Views aus dem Schema `kort` zugegriffen. Dadurch können weitere Schemas unabhängig voneinander hinzugefügt oder entfernt werden. Auch Änderungen innerhalb des Schemas bedeuten keinen Bruch des View-[APIs](#).

10.3.3. Schema für Fehlerquellen

Jede Fehlerquelle sollte ein eigenes Schema besitzen. Jegliche Fehlerquellen-spezifischen Daten, Funktionen oder Types sind dann in diesem Schema gespeichert. Dadurch lassen sich diese beliebig hinzufügen und entfernen.

In unserem Fall kommt uns das zugute, da jede Nacht das *KeepRight* Schema gelöscht wird und anschliessend mit den neuen Fehlerdaten neu angelegt wird.

So ist sichergestellt, dass alle zugehörigen Informationen beieinander sind. Lediglich in der View `kort.all_errors` werden die Fehlerdaten der verschiedenen Schemas zusammengezogen. Die Schemas für Fehlerquellen gehören somit nicht zum [API](#) der Datenbank.

10.3.4. Transaktionen

Um die Konsistenz und Integrität der Daten zu wahren, war es eine wichtige Anforderung Transaktionen auf der Datenbank durchführen zu können. So sollten beispielsweise einem Benutzer *Koins* gutgeschrieben werden, wenn er einen Lösungsvorschlag ablieferiert.

Dadurch, dass die Datenbank über eine REST-Schnittstelle angesprochen wird, könnte man auf die Idee kommen, mehrere Datenbankabfragen mit aufeinanderfolgenden Requests zu realisieren. Dadurch könnte die Datenbank aber zwischen den einzelnen Requests in einen nicht-definierten Zustand geraten.

Um dies zu verhindern, ist es notwendig, Transaktionen über eine separate REST-Ressource zu abstrahieren. Clients können dazu ihre Abfragen in einem einzigen Request schicken. Diese werden dann in einer einzigen Datenbank-Transaktion ausgeführt (siehe Abschnitt 10.2.8). Falls eine Abfrage schief läuft, werden automatisch die Änderungen der Transaktion rückgängig gemacht (*Alles oder nichts*).

10.4. Sicherheit

10.4.1. Authentifizierung mit OAuth

Die Idee hinter OAuth ist bestechend einfach: Eine Applikation möchte auf Dienste im Namen eines Benutzers zugreifen. Der Benutzer möchte der Applikation sein Passwort jedoch nicht preisgeben, sondern lediglich einige wenige Ressourcen freigeben. Da sowohl der Benutzer als auch die Applikation dem Anbieter der Ressource vertrauen, kann der Benutzer seinen Anbieter anweisen, der Applikation Zugriff zu gewähren. Der Anbieter erstellt dazu ein sogenanntes *Access Token*, welches er der Applikation zur Verfügung stellt.

Dieses Prinzip hat auf den ersten Blick nichts mit einem Login zu tun, jedoch kann das Vertrauensverhältnis des Benutzers zum Anbieter und der Applikation zum Anbieter dazu genutzt werden, einen Benutzer einzuloggen, ohne von ihm ein Passwort zu verlangen. Die Ressource beinhaltet in diesem Fall die verfügbaren Benutzerinformationen wie Namen, E-Mail Adresse oder ein Profilfoto.

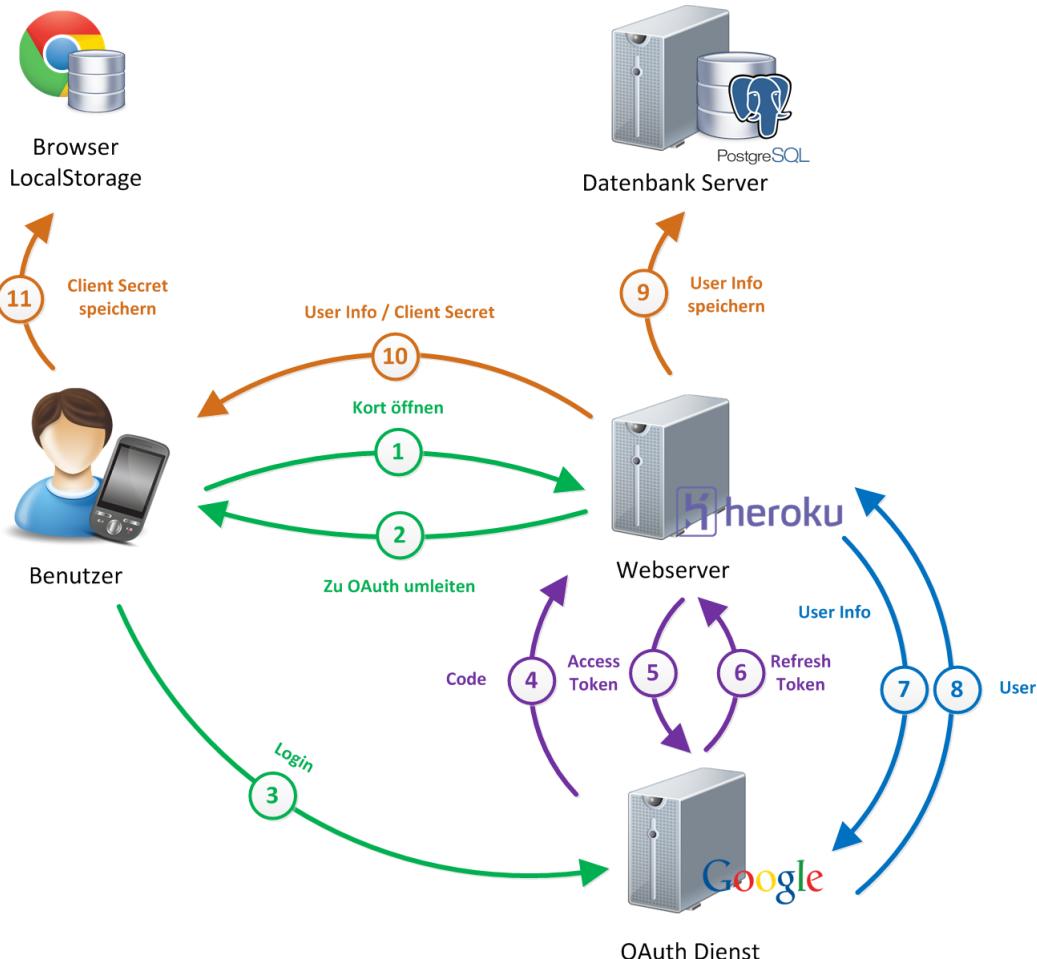


Abbildung 10.1.: Erster Login des Benutzers mit OAuth

In unserem Fall ist KORT die Applikation und Google oder OpenStreetMap der Anbieter der Benutzer-Ressource. Um dem Benutzer zu ersparen, sich bei jedem Besuch erneut via OAuth anzumelden, wird auf dem Server ein *Secret* generiert. Dieses wird lokal beim Benutzer gespeichert (siehe Abbildung 10.1) und ermöglicht es dem Benutzer beim nächsten Login direkt dieses *Secret* zu übermitteln, um Zugriff auf die Applikation zu erhalten (siehe Abbildung 10.2). Diese Übertragung sollte wenn möglich über SSL/TSL erfolgen.

Wenn der Benutzer erfolgreich eingeloggt wurde, wird dies direkt in der Session des Benutzers gespeichert. Auf die entsprechenden Werte wird dann bei sicherheitskritischen Abfragen zurückgegriffen. So ist sichergestellt, dass ein Benutzer lediglich seine eigenen Daten manipulieren kann.

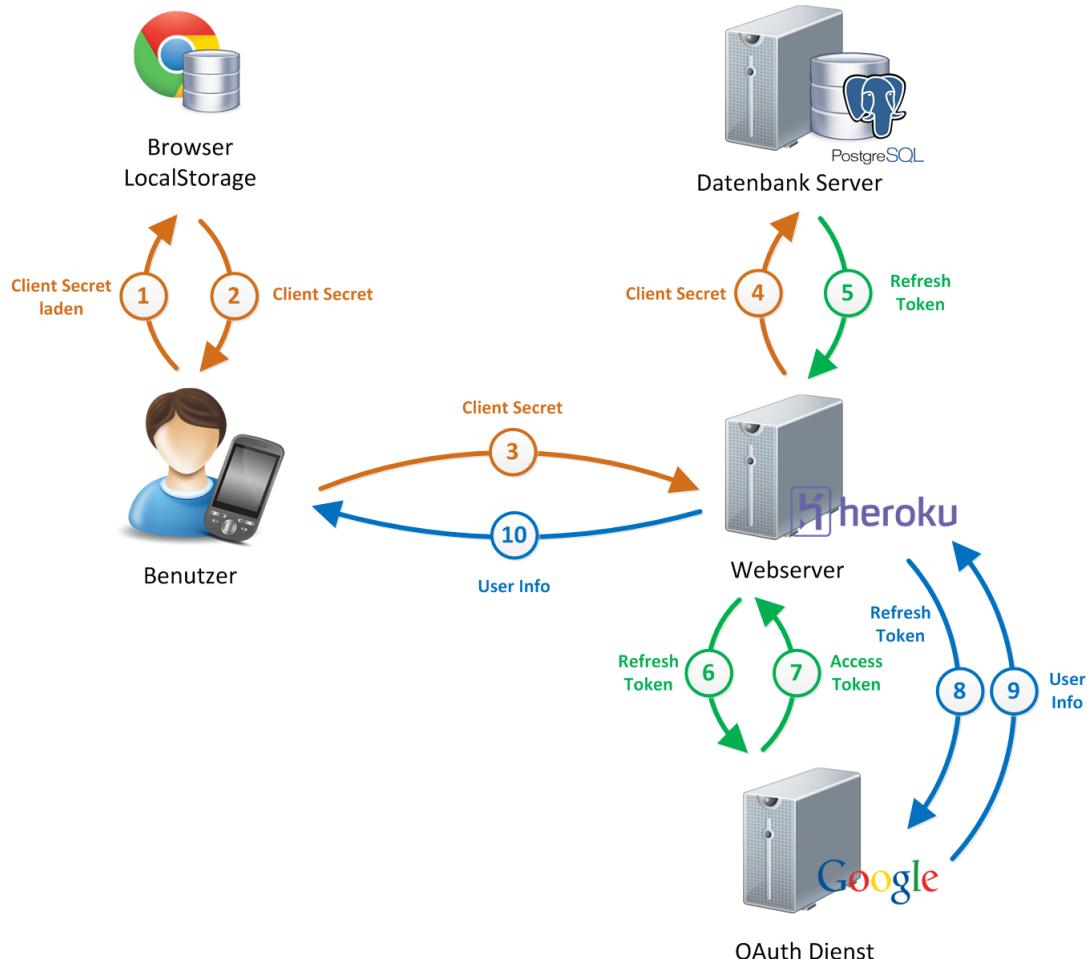


Abbildung 10.2.: Wiedererkennen des Benutzers

Registrierung der Applikation bei Google

Um den Google OAuth Dienst zu nutzen, muss diese zuerst registriert werden (siehe Abbildung 10.3). Ein Benutzer, der sich einloggen möchte, wird von KORT zu Google weitergeleitet. Von dort wird er nach erfolgter Authentifizierung wieder zurückgeleitet. Dieses „Zurückleiten“ wird auch als *Callback* bezeichnet. Bei der Registrierung der Applikation müssen die gültigen Werte für den Callback definiert werden.

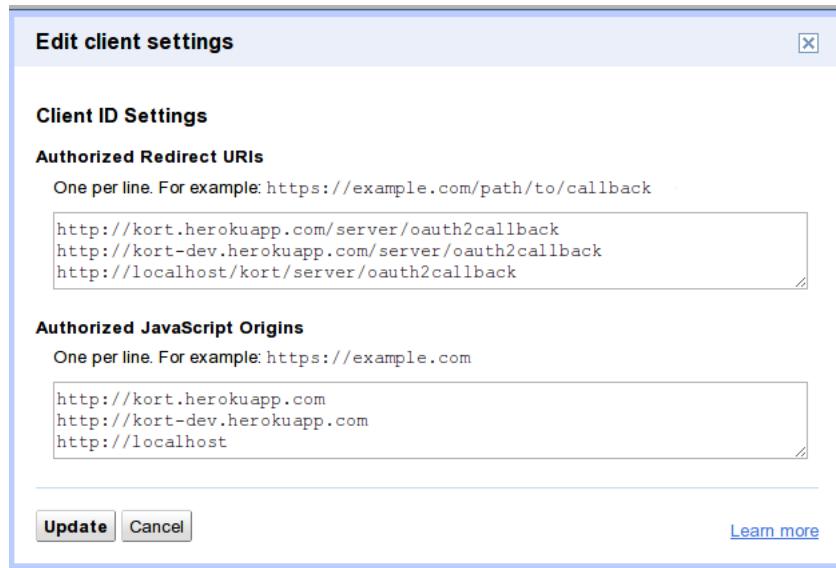


Abbildung 10.3.: OAuth Einstellungen bei Google

Registrierung der Applikation bei OpenStreetMap

Die Registrierung bei *OpenStreetMap* ist sehr ähnlich wie bei Google. Leider bietet OSM aber keine Möglichkeit, bei einer Registrierung, mehrere URLs anzugeben. Somit mussten wir für alle unsere Umgebungen (lokal, Entwicklung und Produktion) eine separate Registrierung vornehmen (siehe Abbildung 10.4).

Anwendungsname	Ausgestellt am	Aktion
kort	2012-12-16 21:54:56 UTC	Widerrufen!
kort-dev	2012-12-16 21:23:23 UTC	Widerrufen!
kort-localhost	2012-12-16 16:17:18 UTC	Widerrufen!
Potlatch 2	2012-06-27 07:40:32 UTC	Widerrufen!

Abbildung 10.4.: OAuth Einstellungen bei OpenStreetMap

10.4.2. Übertragungssicherheit

Die App in der jetzigen Form bietet einige Angriffspunkte. Die Übertragung über HTTP ermöglicht es einem Angreifer, Nachrichten mitzuhören. Damit kann er theoretisch die Daten, welche von und zu der Applikation gesendet werden, abfangen und ändern.

Dies ist aus unserer Sicht aber kein grosses Problem, da es sich bei den Applikationsdaten weder um geheime noch besonders schützenswerte Informationen handelt.

Das grösste Risiko ist, dass es unerlaubte Zugriffe auf die Datenbank gibt. Dazu müsste die Kommunikation zwischen dem Webserver und dem Datenbankserver abgefangen werden. Der in jedem Request enthaltene **API-Key** für die Datenbank läuft derzeit nicht ab.

Abhilfe würde hier eine Umstellung auf HTTPS (SSL/TLS) schaffen. Dies erschwert das Mitlesen der Daten erheblich. Des weiteren könnte die Kommunikation zwischen den Servern mit einem zeitlich generierten Token verbessert werden. So können auch einfach Replay-Attacken verhindert werden.

Der Fokus dieser Arbeit lag aber nicht darauf, die Server-Server oder Server-Client Kommunikation sicher zu gestalten. Für den produktiven Einsatz der Applikation müssen in diesem Bereich aber noch Verbesserungen erfolgen.

10.5. Testing

10.5.1. SimpleTest

*SimpleTest*⁴ ist ein einfaches aber dennoch sehr flexibles Unit Testing Framework. Die bestehende Funktionalität lässt sich sehr einfach erweitern und auf die eigenen Bedürfnisse anpassen. Aus diesem Grund sind im Namespace **TestHelper**⁵ einige Klassen entstanden als Erweiterung von SimpleTest. So ist es beispielsweise möglich, die Tests sowohl im Browser, wie auch auf der Kommandozeile auszuführen.

Die Testresultate der Entwicklungsumgebung befinden sich unter <http://kort-dev.herokuapp.com/test>.

⁴<http://simpleset.org/>

⁵<http://kort.herokuapp.com/docs/Kort-backend/namespaces/TestHelper.html>

kort - TestHttpHelper

Pass: kort - TestHttpHelper->testConstruct->Value [Object: of Helper\HttpHelper] should be type [Helper\HttpHelper] at [/home/odi/hsr/kort/server/php/Helper/Test/TestHttpHelper.php line 27]
Pass: kort - TestHttpHelper->testGet->HTTP GET should not return empty page at [/home/odi/hsr/kort/server/php/Helper/Test/TestHttpHelper.php line 16]

1/1 test cases complete: 2 passes, 0 fails and 0 exceptions.

kort - TestPostGisSqlHelper

Pass: kort - TestPostGisSqlHelper->testGetLatLngGeom->Equal expectation [String: ST_SetSRID(ST_Point(8,47),4326)] at [/home/odi/hsr/kort/server/php/Helper/Test/TestPostGisSqlHelper.php line 16]

1/1 test cases complete: 1 passes, 0 fails and 0 exceptions.

kort - TestSecretGenerator

Pass: kort - TestSecretGenerator->testConstruct->Value [Object: of Helper\SecretGenerator] should be type [Helper\SecretGenerator] at [/home/odi/hsr/kort/server/php/Helper/Test/TestSecretGenerator.php line 27]
Pass: kort - TestSecretGenerator->testGetSecret-> at [/home/odi/hsr/kort/server/php/Helper/Test/TestSecretGenerator.php line 16]

1/1 test cases complete: 2 passes, 0 fails and 0 exceptions.

kort - TestSlimHelper

Pass: kort - TestSlimHelper->testConstruct->Value [Object: of Helper\SlimHelper] should be type [Helper\SlimHelper] at [/home/odi/hsr/kort/server/php/Helper/Test/TestSlimHelper.php line 32]
{"return": {"test": "data"}}

1/1 test cases complete: 1 passes, 0 fails and 0 exceptions.

[kort - TestHttpHelper](#)

Abbildung 10.5.: Testresultate von SimpleTest im Browser

Der derzeitige Build ist so konfiguriert, dass er fehlschlägt, wenn die Tests nicht bestanden werden. Dies stellt sicher, dass sich nur stabile Versionen auf unserem Webserver befinden.

10.5.2. Integrationstests der Webservices mit QUnit

Neben den klassischen Unit Tests, waren für die verschiedenen REST-Webservices auch noch Integrationstest nötig. Im Rahmen solcher Tests wird via *QUnit* (welches wir für das JavaScript Testing benötigen) eine Anfrage an die betreffende Ressource gestellt. Deren Antwort lässt sich dann im Unit Test leicht prüfen.

Die JavaScript Tests befinden sich auf der Entwicklungsumgebung unter <http://kort-dev.herokuapp.com/test/client/>.

The screenshot shows a browser window with a dark header bar containing the text "kort - JavaScript Unit-Tests" and three checkboxes: "noglobals", "notrycatch", and "notrycatch". Below the header is a red navigation bar with a "Hide passed tests" button. The main content area has a blue header bar with the text "Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.57 Safari/537.1". Underneath is a grey bar stating "Tests completed in 96 milliseconds. 14 tests of 15 passed, 1 failed." The test list consists of 10 items, each with a color-coded status bar and a "Rerun" link:

- 1. **kort-Availability: Includes (SKIPPED)** (0, 0, 0) Rerun
- 2. **kort-UrlLib: Constructor** (0, 2, 2) Rerun
- 3. **kort-UrlLib: getCurrentUrl** (0, 1, 1) Rerun
- 4. **kort-UrlLib: getCurrentUrl with anchor** (0, 1, 1) Rerun
- 5. **kort-UrlLib: getUrlParams** (0, 6, 6) Rerun
- 6. **kort-UrlLib: getUrlParams without values** (0, 3, 3) Rerun
- 7. **kort-UrlLib: getAppUrl** (0, 1, 1) Rerun
- 8. **kort-AnswerWebservice: root (SKIPPED)** (0, 0, 0) Rerun
- 9. **kort-AnswerWebservice: missing_track_type (SKIPPED)** (0, 0, 0) Rerun
- 10. **kort-AnswerWebservice: not existsing type** (1, 0, 1) Rerun

The last item, "10. kort-AnswerWebservice: not existsing type", is highlighted with a red background. A red callout box points to the first error in the list: "1. not existing answer type should return 404." It provides details about the expected result (404), the actual result (200), the difference (404 200), and the source code location (at http://localhost/kort/test/client/webservices/TestAnswerWebservice.js:53:9).

Abbildung 10.6.: Testresultate der Integrationstests im Browser

10.6. Dokumentation

Das Backend von KORT ist durchgängig mit der PHP-Dokumentationssprache *PHPDoc*⁶ dokumentiert⁷. Um sicherzustellen, dass die PHPDoc-Kommentare stets aktuell sind, haben wir deren Schema in unseren Code-Standard integriert. Der Code-Standard wird automatisch überprüft und Abweichungen davon dem Benutzer angezeigt. Die ganze Dokumentation wird jeweils beim builden neu generiert.

Grundsätzlich stellt die Dokumentation das öffentliche API des Backends dar. Alle privaten

⁶<http://www.phpdoc.org/>

⁷Die Dokumentation findet sich unter: <http://kort.herokuapp.com/docs/Kort-backend>

Funktionen sind aber ebenfalls dokumentiert, um einen einfachen Einstieg in die Weiterentwicklung der Applikation zu ermöglichen.

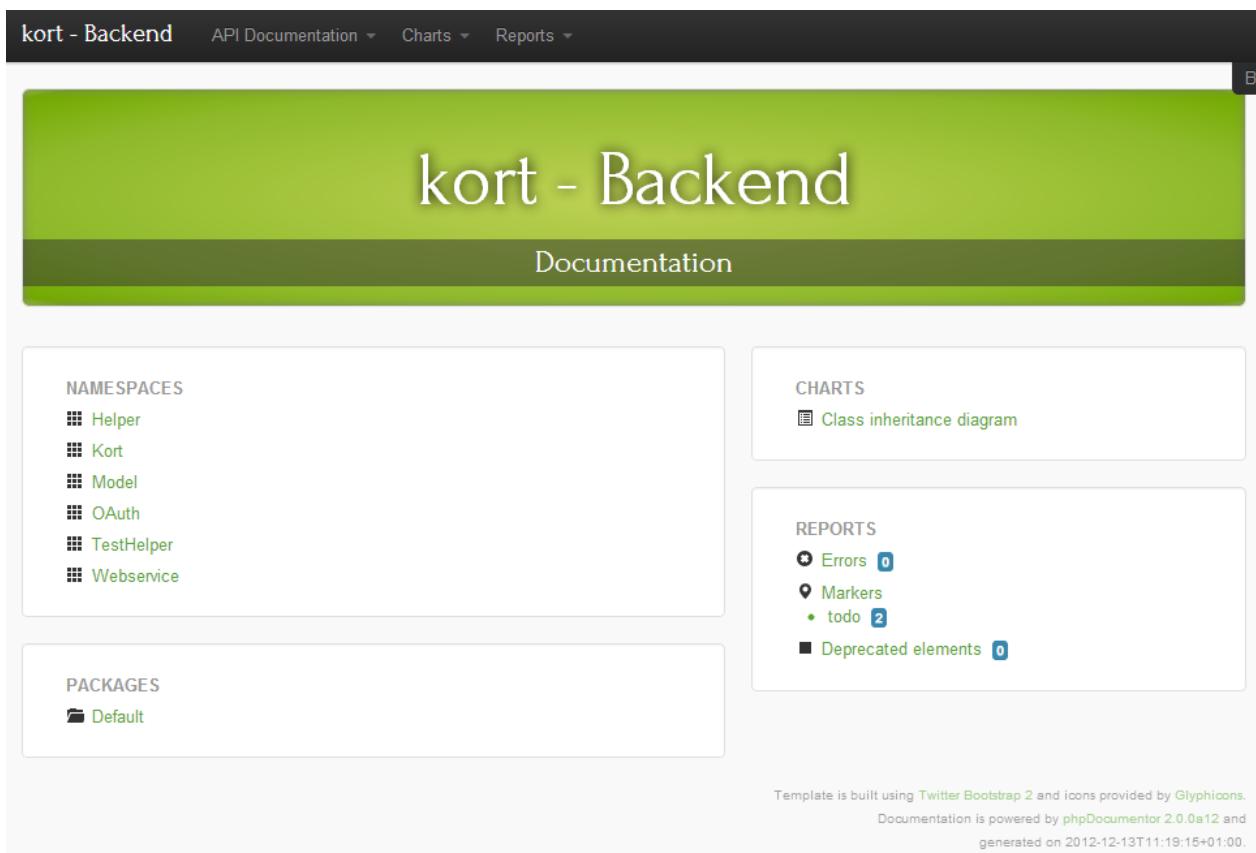


Abbildung 10.7.: KORT Backend Dokumentation mit PHPDoc

11. Administration

11.1. Hinzufügen von Fehler-Datenquellen

Die Applikation benutzt die View `kort.all_errors` (siehe Abbildung 11.1) als Schnittstelle zu allen Fehlerdaten. Wenn neue Fehlerquellen an das System angeschlossen werden sollen, dann muss entsprechend diese View angepasst werden. Die aktuelle View ist auf die Fehlerquelle *KeepRight* (siehe Kapitel 7.4) angepasst.

Es werden einige Anforderungen an eine neue Fehlerquelle gestellt:

- Ein Fehler muss sich konkret auf ein *OpenStreetMap*-Objekt beziehen ([Node](#), [Way](#) oder [Relation](#))
- Für die Darstellung auf der Karte muss eine einzelne Koordinate angegeben werden können (keine Flächen oder Wege)
- Zu einem Fehler gibt es eine aussagekräftige Fehlermeldung

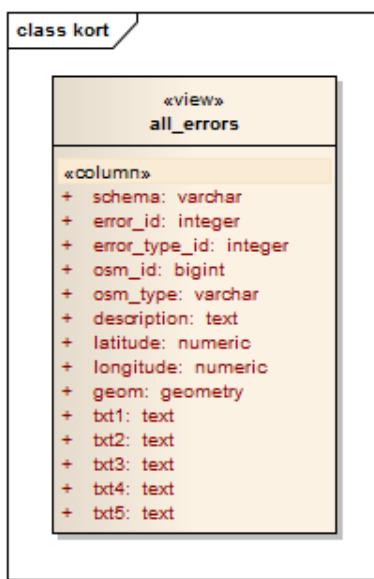


Abbildung 11.1.: Die View `kort.all_errors`

Die Fehler einer neuen Quelle müssen auf die bestehenden Fehlertypen gemappt werden. Falls dies nicht möglich ist oder es sich um neue Arten von Fehlern handelt, müssen zusätzliche Fehlertypen eingeführt werden (siehe Abschnitt 11.2).

Daneben müssen für alle neuen Fehlertypen Texte erfasst und allenfalls auch internationalisiert werden (siehe Abschnitt [7.3.2](#)).

11.2. Hinzufügen von Fehlertypen

Um neue Fehlertypen hinzuzufügen müssen sowohl technische als auch fachliche Anpassungen vorgenommen werden. Wichtig dabei ist es, den Endbenutzer nicht aus den Augen zu verlieren. Alle aufgeführten Fehler sollten sich mit wenigen Eingaben lösen lassen. Die bisher implementierten Fehlertypen benötigen alle nur eine einzige Eingabe. Für neue Typen ist zu prüfen, ob diese sich bereits mit den bestehenden Views abbilden lassen (siehe Kapitel [11.2.2](#)). Ist dies nicht der Fall, so muss eine neue View erstellt werden.

11.2.1. Fehlertypen

In der Tabelle [11.1](#) sind die in der App bereits implementierten Fehlertypen beschrieben.

FehlerTyp	Anzahl vorhandener Fehler (Stand 20.12.2012)
Typ des Wegs unbekannt	1'271'632
Fehlendes Tempolimit	463'819
Sprache des Namens unbekannt	111'565
Objekt ohne Namen	92'080
Strasse ohne Namen	74'864
Kultstätte/Kirche ohne Religion	15'288
Autobahn ohne Bezeichner	1'504
Total	2'030'752

Tabelle 11.1.: Fehlertypen in KORT

Diese Typen sind in der Datenbanktabelle `kort.error_type` (siehe Abbildung [11.2](#)) definiert. Im Feld `type` dieser Tabelle befindet sich die eindeutige Identifikation eines Typs. Diese wird beispielsweise für die Anzeige eines passenden Marker-Icons (auf der Karte) und der Verbindung zu einem passenden View-Typ (siehe Abschnitt [11.2.2](#)) verwendet.

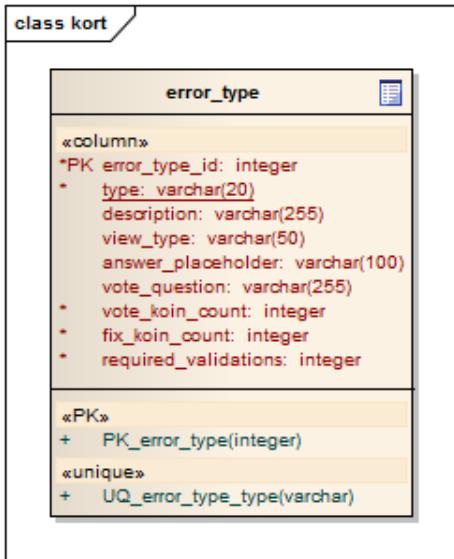


Abbildung 11.2.: Die Tabelle kort.error_type

Erstellen eines neuen Fehlertyps

Ein neuer Fehlertyp muss in der Tabelle `kort.error_type` (siehe Abbildung 11.2) hinzugefügt werden. Sobald Fehler als `type` den neuen Typs eingetragen haben, wird dieser auch angezeigt. Jeder Fehlertyp hat ein zugehöriges Marker-Icon, welches beispielsweise auf der Karte angezeigt wird. Die Bilddatei hat den gleichen Namen wie das Attribut `type` und befindet sich im Ordner `resources/images/marker_icons/<type>.png`.

11.2.2. View-Typen

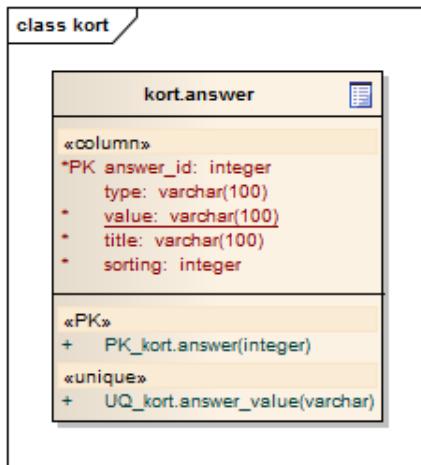
All diese Typen werden dann wiederum einem spezifischen View-Typen zugeordnet. Dieser bestimmt, wie das Formular zum Lösen des Fehlers in der Benutzeroberfläche aussieht.

In Tabelle 11.2 sind die bereits vorhandenen View-Typen beschrieben.

Typ	Beschreibung
text	Rendert ein Text-Eingabefeld
number	Rendert ein Zahl-Eingabefeld
select	Rendert eine Select-Box mit vorgefüllten Werten aus der Datenbank

Tabelle 11.2.: View-Typen in KORT

Wird als View-Typ `select` gewählt, müssen in der Tabelle `kort.answer` (siehe Abbildung 11.3) die möglichen Antworten eingetragen werden. Darin kann der eigentliche Wert des *OpenStreetMap*-Tags und eine passende Bezeichnung hinterlegt werden. Als `type` muss der jeweilige Typen-Bezeichner gewählt werden.

Abbildung 11.3.: Die Tabelle `kort.answer`

Erstellen eines neuen View-Typen

Um einen neuen View-Typen zu definieren muss wie in Code-Ausschnitt 11.1 beschrieben die Unterscheidung in der Klasse `Kort.view.bugmap.fix.Form` mit dem den neuen Typen ergänzt werden.

```

1 createFixField: function(bug) {
2     [...]
3
4     if(bug.get('view_type') === '<Neuer View-Typ>') {
5         fixField = Ext.create('<Neue View-Klasse>', fieldConfig);
6     } else {
7         ...
8     }
9
10    [...]
11 }
```

Code-Ausschnitt 11.1: Unterscheidung der View-Typen in `Kort.view.bugmap.fix.Form`

11.3. Hinzufügen von Auszeichnungen

Die bereits vorhandenen Auszeichnungen sind in Tabelle 6.1 beschrieben. Um weitere Auszeichnungen hinzuzufügen, muss folgendermassen vorgegangen werden:

1. Es muss ein neuer Badge in der Tabelle `kort.badge` (siehe Abbildung 11.4) erstellt werden
2. Zusätzlich muss eine Regel für das Gewinnen des Badges in der Methode `updateBadges()`¹ der Klasse `RewardHandler` hinzugefügt werden.

¹http://kort.herokuapp.com/docs/Kort-backend/classes/Webservice.RewardHandler.html#method_updateBadges

3. Für das Frontend muss ein Bild erstellt werden, welches dem Namen (Tabellenattribut `name`) des Badges entspricht. Dieses Bild muss in folgendem Pfad gespeichert werden `/resources/images/badges/<badgename>.png`.



Abbildung 11.4.: Die Tabelle kort.badge

Sind alle Punkte abgearbeitet, ist der Badge im Frontend ersichtlich und kann von den Benutzern gewonnen werden.

11.4. Hinzufügen von OAuth Anbietern

Derzeit ist der Login über die [OAuth](#)-Dienste von *OpenStreetMap* und *Google* möglich. Um zukünftig weitere Login-Dienste anbieten zu können, müssen einige Schritte beachtet werden. Der Dienst muss folgende Anforderungen erfüllen:

- Unterstützung für OAuth 1.0, 1.0a oder 2.0
- Offline-Zugang, d.h. es müssen Aktionen ohne den Benutzer möglich sein
- Der Dienst muss eine Registrierung der Applikation ermöglichen

Wenn ein Dienst diese Anforderung erfüllt, muss die Applikation registriert werden.

Im Backend kann eine entsprechende Subklasse von `AbstractOAuthCallback`² erstellt werden. Eine so erstellte OAuthCallback-Klasse nimmt dann einen Callback des entsprechenden Anbieters entgegen und authentifiziert den Benutzer. Das zugehörige Callback-Script wird im Order `/server/oauth2callback` gespeichert.

Schlussendlich muss das Frontend noch angepasst werden. Dazu gehört, dass der neue Dienst in der Konfiguration (siehe Kapitel 11.6) eingetragen werden muss. Um einen neuen Login-Button anzuzeigen, muss dieser auf dem Login-Panel hinzugefügt werden. Die zugehörige Logik befindet sich im Login-Controller

²<http://kort.herokuapp.com/docs/Kort-backend/classes/OAuth.AbstractOAuthCallback.html>

Die Dateien befinden sich unter:

- Login-Panel: /app/view/overlay/login/Panel.js.
- Login-Controller: /app/controller/Login.js



Abbildung 11.5.: Login-Buttons in KORT

11.5. Reset der Applikation

Um die Applikation zurückzusetzen, kann die Stored Procedure `reset_kort()` verwendet werden. Die Prozedur löscht alle laufenden Daten aus der Datenbank, so dass wieder der Ursprungszustand hergestellt ist:

- Die Punkte (*Koins*) aller Benutzer werden auf 0 zurückgesetzt
- Die Badges aller Benutzer werden entfernt
- Alle Lösungsvorschläge von Benutzern werden entfernt
- Alle Überprüfungen von Lösungsvorschlägen werden entfernt

Der Aufruf der Prozedur sieht man im Code-Ausschnitt 11.2. Die Prozedur liefert `true` wenn alle Aktionen erfolgreich ausgeführt werden konnten, ansonsten `false`.

```
1 select reset_kort();  
2  
3 reset_kort  
4 -----  
5 t  
6 (1 row)
```

Code-Ausschnitt 11.2: Aufruf von `reset_kort()`, um die Applikation zurückzusetzen

11.6. Frontend Konfiguration

Die Konfiguration des Frontends befindet sich in der Datei `/app/util/Config.js`. Darin sind alle Parameter definiert, welche die Applikation für den Betrieb benötigt.

In der KORT-Dokumentation sind alle Parameter detailliert beschrieben:

<http://kort.herokuapp.com/docs/Kort/#!/api/Kort.util.Config>

Teil III.

Projektmanagement und -monitoring

12. Projektmanagement

Wir verwendeten die agile Projektmethodik *Scrum*¹ und arbeiteten dabei während 14 Wochen. Die Dauer eines Sprints haben wir auf 2 Wochen festgelegt. In den ersten beiden Wochen waren wir mit dem Erstellen der Aufgabenstellung beschäftigt. Für die restlichen 12 Wochen ergeben sich dadurch 6 Sprints.

Aufwand pro Person

Für diese Bachelorarbeit werden 12 ECTS-Punkte vergeben, wobei 1 ECTS-Punkt 30 Stunden Arbeitsaufwand bedeuten. Dies ergibt 360 Stunden pro Person, was etwas mehr als 3 Arbeitstage pro Woche entspricht.

Anpassung der Sprintdauer

Wir mussten während des Verlaufs des Projektes feststellen, dass die Sprintdauer von 2 Wochen etwas zu kurz ist und haben deshalb die letzten 6 Wochen mit 2 Sprints à 3 Wochen ersetzt. Insgesamt wurden im Projekt 5 Sprints durchgeführt.

Storypoints pro Sprint

Als Ausgangslage haben wir 12 Storypoints pro Sprint angenommen, wobei 1 Storypoint ungefähr einem Arbeitstag entspricht. Dieser Wert hat sich gut bewährt und wir mussten ihn lediglich bei der Anpassungen der Sprintdauer auf 17 Punkte erhöhen.

¹<http://www.scrum.org>

12.1. Sprint 1

1. Oktober 2012 bis 14. Oktober 2012

Alle Informationen zum Sprint 1 sind auch in unserem Wiki zu finden: http://kort.rdmr.ch/redmine/projects/kort/wiki/Sprint_1

Hauptaufgaben / Fokussierung im Sprint

- Aufsetzen der Infrastruktur (Repository, Projektmanagement, Entwicklungsumgebung, Server)
 - Einarbeitung in die Thematik
 - Gamification
 - Highscore API
 - OpenStreetMap
 - OpenLayers
 - allenfalls weitere APIs
 - Wrapper für *OpenStreetMap* in Sencha Touch
 - Projektsetup
 - GitHub² als Repository
 - Heroku³ fürs Hosting
 - Travis⁴ für Continuous Integration (CI)
 - Redmine⁵ für Projektmanagement/Wiki/Bugtracker
 - LATEX⁶ für Dokumentation
 - Scrum⁷ als Methodik

Ziele

- *OpenStreetMap* genauer kennenlernen (Daten, Struktur, API)

²<http://www.github.com>

³<http://www.heroku.com>

⁴<http://travis-ci.org>

⁵<http://www.redmine.org>

⁶<http://www.latex-project.org>

⁷<http://de.wikipedia.org/wiki/Scrum>

- Integration in Sencha sicherstellen

Abgabe / Deliverables

- Übersicht zum Projekt auf Redmine (Sprints, Issues/Tasks, generelle Infos)
- Projekt-Setup mit GitHub, Heroku und Jenkins
- Lauffähiger Prototyp mit *OpenStreetMap* in Sencha Touch

Erledigte Arbeiten

Im ersten Sprint konnten wir bereits einiges erledigen. So gelang uns ein automatisierter Build der App inklusive Deployment zu Heroku. Außerdem haben wir eine Sencha Touch Komponente für OpenLayers erstellt, um *OpenStreetMap* Daten in der App darstellen zu können. Ebenfalls gelang es uns mit dem Plugin *Ext.i18n.bundle-touch* die Internationalisierung der App vorzubereiten.

Probleme

Leider mussten wir feststellen, dass OpenLayers Library ein nicht mehr zeitgemäßes API aufweist. Es gilt deshalb im zweiten Sprint Alternativen dafür zu finden.

12.2. Sprint 2

15. Oktober 2012 bis 28. Oktober 2012

Alle Informationen zum Sprint 2 sind auch in unserem Wiki zu finden: http://kort.rdmr.ch/redmine/projects/kort/wiki/Sprint_2

Hauptaufgaben / Fokussierung im Sprint

- Alternative für OpenLayers finden
- Aufbau der Fehler-Datenbank
- Aufsetzen der REST-APIs für DB-Zugriff
 - REST-API auf Heroku für Sencha-App
 - REST-API auf DB-Server für Zugriff von Heroku
- Requirements
 - User Szenarien
 - Paper Prototype

Ziele

- Roundtrip von Datenbank → Heroku → [Web-App](#) und zurück (Daten lesen und schreiben)
- Anforderungen klar machen für Ausrichtung der App
 - Was ist möglich?
 - Was soll tatsächlich abgebildet werden?
- Technische Hürden im Middletier überwinden, um sich auf Business-Logik und UI zu konzentrieren

Abgabe / Deliverables

- Lauffähiger Prototyp
 - Anzeigen von Bugs
 - Eintragen von Daten in DB
- User Szenarien
- Paper Prototyp

Erledigte Arbeiten

Während des zweiten Sprints haben wir verschiedene Fehler-Datenquellen evaluiert. Wir entschieden uns für den Einsatz der *KeepRight*-Daten, da diese sehr gut unseren Anforderungen entsprechen. Sie werden automatisiert generiert, was es uns ermöglicht, für die verschiedenen Typen, auch automatisiert eine Lösungsmaske anzuzeigen. Zudem handelt es sich nur um Fehler in Tags der OpenStreetMap-Objekte.

Ebenfalls konnten wir, als Alternative zu OpenLayers, eine Sencha Touch Komponente erstellen, welche *Leaflet* verwendet, um OpenStreetMap-Daten anzuzeigen. Die Library besitzt ein gutes [API](#) und wurde speziell für die Verwendung im mobilen Umfeld erstellt.

Schlussendlich konnten wir einen kompletten Roundtrip der Fehler-Daten von unserer Datenbank in die App und das Senden der Lösung wieder zurück in die Datenbank realisieren.

Probleme

Aus Zeitgründen konnten wir das Zurückschreiben der Daten zu *OpenStreetMap* noch nicht erledigen.

12.3. Sprint 3

29. Oktober 2012 bis 11. November 2012

Alle Informationen zum Sprint 3 sind auch in unserem Wiki zu finden: http://kort.rdmr.ch/redmine/projects/kort/wiki/Sprint_3

Hauptaufgaben / Fokussierung im Sprint

- Backend
 - Automatisierung der Fehlerdatenbank
 - OAuth-Login bei Google/Facebook/Twitter etc.
- Frontend
 - Detail-Seiten für verschiedene Bug-Typen
 - Profil-Seite

Ziele

- Welche Bug-Typen eignen sich für unsere App?
- Bug-Detailmasken implementiert
- OAuth Login und Logout

Abgabe / Deliverables

- Lauffähiger Prototyp
 - Anzeigen von Bug-Details (mind. 2 verschiedene Typen)
 - Profile-Seite
 - Login/Logout

Erledigte Arbeiten

In diesem Sprint konzentrierten wir uns auf die Handhabung der verschiedenen Fehlertypen. Wir mussten zuerst herausfinden, welche Typen sich für unsere App eignen. Für diese mussten wir dann passende Lösungsmasken im Frontend erstellen.

Das zweite grosse Ziel war der Login bzw. Logout über OAuth. Dafür konnten wir ebenfalls eine erste Lösung erstellen.

Probleme

Der Benutzer wird derzeit noch nicht in der Datenbank persistiert. Der Login wird momentan noch nicht der Session gespeichert.

12.4. Sprint 4

12. November 2012 bis 2. Dezember 2012

Alle Informationen zum Sprint 4 sind auch in unserem Wiki zu finden: http://kort.rdmr.ch/redmine/projects/kort/wiki/Sprint_4

Hauptaufgaben / Fokussierung im Sprint

- Backend
 - Datenbank-Setup
 - OAuth-Handling auf Server (Refresh-Token)
- Frontend
 - Validations-Maske erstellen mit Fix-Einträgen, welche zu verifizieren sind
 - Highscore-Masken
 - Abschluss Bug Detailmasken

Ziele

- Validations-Maske implementiert
- Highscore-Maske implementiert
- Komplettes Datenbank-Setup
- User-Handling (Client, Server, Persistenz, Refresh-Token)

Abgabe / Deliverables

- Lauffähiger Prototyp
 - Validations-Maske zeigt zu validierenden Lösungsvorschläge an
 - Highscore (global) kann angezeigt werden
 - Datenbank mit vollständigem Schema (bootstrapped)

Erledigte Arbeiten

Zu Beginn des Sprints mussten wir die Datenbank nach unserem definierten Schema aufbauen. Dadurch konnten wir mit der Implementation der weiteren Masken der App (Validation, Highscore) starten. Zusätzlich haben wir begonnen die Benutzeranmeldungen in der Datenbank zu persistieren. Während dieses Sprints nahmen wir noch am OSM Stammtisch⁸ teil und konnten dort unsere App der [Mapping](#)-Community präsentieren. Wir erhielten spannende und wichtige Hinweise für die weitere Entwicklung unserer App.

Probleme

Leider konnten wir die Persistierung der Benutzer noch nicht komplett abschliessen. Wir werden dies im nächsten Sprint angehen.

12.5. Sprint 5

3. Dezember 2012 bis 21. Dezember 2012

Alle Informationen zum Sprint 5 sind auch in unserem Wiki zu finden: http://kort.rdmr.ch/redmine/projects/kort/wiki/Sprint_5

Hauptaufgaben / Fokussierung im Sprint

- Fertigstellen der begonnenen Funktionalitäten
- Fehlerbehebungen
- Production-Build erstellen
- Dokumentation
 - Code-Dokumentation
 - Projektdokumentation
 - A0 Poster erstellen

Ziele

- Abschliessen der begonnenen Funktionalitäten
 - Gutschreiben der Koins / Badges
 - Zurückschreiben der Daten zu OSM

⁸http://wiki.openstreetmap.org/wiki/DE:Switzerland:Z%C3%BCrich/OSM-Treffen#36._OSM-Stammtisch

- OAuth mit OpenStreetMap
- Finalisieren der Dokumentation

Abgabe / Deliverables

- Finale Version der App
- A0 Poster / Abstract / Management Summary
- Finale Version der Dokumentation

Erledigte Arbeiten

Im letzten Sprint lag der Fokus bei der Finalisierung der Applikation. So mussten wir Funktionalitäten, welche noch nicht vollständig implementiert waren, fertigstellen und bestehende Fehler beheben.

An zusätzlichen Funktionen haben wir den Login über OpenStreetMap implementiert, sowie das Setzen des eigenen Benutzernamens auf der Profilseite.

Ebenfalls galt ein grosser Teil der Zeitplanung der Dokumentation des Projektes.

Probleme

Wir mussten aus Zeitgründen auf das Zurückschreiben der Korrekturen zu *OpenStreetMap* verzichten.

13. Projektmonitoring

13.1. Meilensteine

Nachdem wir die Hauptaktivitäten des Projekts abschätzen konnten, haben wir eine Liste von Meilensteinen definiert. Diese beschreiben konkrete Ziele welche während dieser Arbeit zu erreichen sind.

Wir haben uns jeweils einige Meilensteine für einen Sprint zum Ziel gesetzt und haben über die Wiki-Seite unseren Fortschritt festgehalten.

Somit gab es neben dem rein agilen Sprint-Planning auch noch funktionale Eckpfeiler.

Meilenstein 1: Aufbereiten von Fehlerdaten

- ✓ Geeignete Fehler finden
- ✓ Fehler in Datenbank speichern
- ✓ Fehler filtern
- ✓ Fehler zugänglich machen

Meilenstein 2: Lesen von Fehlerdaten

- ✓ Zugriff auf Datenbank aus App
- ✓ Persistieren der Daten in App

Meilenstein 3: Kartenanzeige von Fehlern

- ✓ Geeignete Darstellung der Fehler auf der Karte finden
- ✓ Fehler auf der Karte anzeigen
- ✓ Location-based Karte

Meilenstein 4: OAuth

- ✓ Login möglich über OAuth
- ✓ User wird in App persistiert
- ✓ Logout möglich
- ✓ Login mit OSM-OAuth

Meilenstein 5: Schreiben von Fehlerbehebungen

- ✓ Eingaben von Benutzer in DB speichern
- ✓ Unterscheidung von Fehlertypen

Meilenstein 6: Validieren von Änderungen

- ✓ UI zum Validieren von Fehlerbehebungen
- ✓ Implementation eines Thresholds für Validierung

Meilenstein 7: Daten zu OSM schicken

- ✗ Fehlerlösungen einheitlich via API an OSM senden
- ✗ Prüfung, ob eine Änderung zulässig ist

Meilenstein 8: Gamification-Elemente

- ✓ Geeignete Gamification-Elemente für OSM finden
- ✓ Elemente in App umsetzen

13.2. Risikomanagement

Für das Projekt wurden folgende Risiken identifiziert:

13.2.1. Technische Risiken

OpenStreetMap-Daten können nicht in einer Sencha Touch-Applikation angezeigt werden

Auswirkung	Es müsste ein alternatives mobiles Framework gefunden werden, welches mit <i>OpenStreetMap</i> -Daten umgehen kann.
Wahrscheinlichkeit	tief
Massnahme zur Verhinderung	Zu Beginn des Projektes muss eine Sencha Touch Prototyp-Applikation implementiert werden, welche <i>OpenStreetMap</i> -Daten auf der Karte darstellt.

Keine geeignete Fehlerdatenquelle vorhanden

Auswirkung	Es müsste eine Möglichkeit gefunden werden, vorhandene Fehlerdaten so abzuändern, dass sie für den Einsatz in der App verwendet werden können.
Wahrscheinlichkeit	mittel
Massnahme zur Verhinderung	Es muss eine Evaluation von bestehenden Fehlerdatenquellen durchgeführt werden.

Schwierigkeiten mit OAuth

Auswirkung	OAuth ist als Protokoll bekannt, welches Schwierigkeiten bereiten kann. Falls sich die Anforderungen nicht umsetzen lassen, muss eine alternative Lösung gefunden werden für den Login.
Wahrscheinlichkeit	mittel
Massnahme zur Verhinderung	Es muss genügend Zeit für OAuth eingeplant werden.

13.2.2. Fachliche Risiken

OpenStreetMap erlaubt keinen allgemeinen Benutzer zum Zurückschreiben der Fehlerbehebungen

Auswirkung	Es müsste eine Möglichkeit gefunden werden, die Fehlerbehebungen trotzdem in <i>OpenStreetMap</i> einpflegen zu können.
Wahrscheinlichkeit	mittel
Massnahme zur Verhinderung	<i>OpenStreetMap</i> -Community muss von der Idee hinter KORT überzeugt werden.

Zu wenig Erfahrung mit Game-Design

Auswirkung	Die App hat keinen Game-Charakter oder wird vom Benutzer nicht als solches erkannt.
Wahrscheinlichkeit	gross
Massnahme zur Verhinderung	Unser Industriepartner hat viel Erfahrung mit der Entwicklung von Games und kann uns mit Ratschlägen unterstützen. Daneben haben wir versucht, Designer zu involvieren welche uns unterstützen können.

13.3. Projektverlauf

Das Projekt bestand aus 5 Sprints, wobei wir ursprünglich 6 geplant haben. Gegen Ende des Projekts mussten wir uns eingestehen, dass der Overhead für zweiwöchige Sprints zu gross ist, deshalb haben wir die letzten beiden Sprints auf 3 Wochen erweitert.

Dank unserem Projektmanagement-Tool *Redmine*, hatten wir eine gute Übersicht über das Projekt. Jeweils am Ende eines Sprints haben wir besprochen, wie wir weiterfahren sollen und entsprechend den nächsten Sprint geplant. Durch diese Iterationen war es uns möglich, schnell ein System zu erstellen, welches den aktuellen Bedürfnissen entspricht.

Abschliessend ist zu sagen, dass das Projekt gut verlaufen ist und wir grundsätzlich alle unsere Ziele erreichen konnten (siehe Abschnitt [13.5](#)). Daneben gab es auch Raum, kreative Ideen auszuprobieren.

13.4. Arbeitsaufwand

Wie im Kapitel [12](#) beschrieben, war der vom Modul vorgegebene Aufwand pro Person auf *360 Stunden* festgelegt. Leider haben wir beide diese Vorgabe leicht überschritten (siehe Tabelle [13.1](#)).

Person	Aufwand
Jürg Hunziker	379h
Stefan Oderbolz	394h

Tabelle 13.1.: Arbeitsaufwand pro Person

13.5. Fazit

Die entwickelte [Web-App](#) erfüllt alle Erwartungen an eine moderne Applikation. Ein Benutzer kann mit der App die gewünschten Aufgaben (Fehler korrigieren und validieren) durchführen und wird durch einige Gamification-Konzepte animiert, die App weiter zu verwenden.

In der knapp bemessenen Zeit ist es uns aber nicht gelungen ein vollwertiges Spiel zu entwickeln. Dazu müsste noch mehr Wert auf Details gelegt werden, so dass ein abgerundetes Spielerlebnis entsteht. Schlussendlich haben wir uns mehr auf die Grundfunktionalität der [Web-App](#) konzentriert.

Mit Hilfe der [Continuous Integration](#) hatten wir stets ein lauffähiges System. Dieses wurde bei jeder Änderung automatisiert neu gebaut und auf alle Systeme ausgerollt.

Dank der agilen Vorgehensweise konnten wir schnell auf Änderungen in den Anforderungen oder Probleme reagieren, welche sich bei einem solchen Projekt zwangsläufig ergeben. Wir waren uns zu Beginn noch nicht bewusst, welche Funktionalitäten wichtig sind und welche

weniger. Der ursprüngliche Plan war, so schnell wie möglich den Roundtrip durch alle Systeme zu erfolgreich durchführen zu können. Als wir aber erkannten, dass wir es mit vielen verschiedenen Schnittstellen zu tun haben, sind wir von diesem Plan wieder abgekommen und haben uns dazu entschlossen, zuerst unsere App zu stabilisieren und dann eine Schnittstelle nach der anderen anzuschauen.

Die Arbeit hat uns viele spannende Einblicke in *OpenStreetMap* und die Thematik der [Gamification](#) ermöglicht. Das *OpenStreetMap*-Umfeld ist sehr interessant und die intensive Arbeit mit Fehlerdaten hat uns gezeigt wie einzigartig dieses Projekt ist.

Wir hoffen, dass auch unsere App ihren Teil dazu beiträgt neue Benutzer fürs [Mappen](#) zu begeistern.

Teil IV.

Anhänge

Inhalt der CD

Pfad	Beschreibung
.sencha/	Konfiguration für Sencha Cmd
_DESIGN/	Grafik-Rohdaten
_DOCUMENTATION/	Dokumentation der Arbeit
_DOCUMENTATION/ba-kort-jhunzike_soderbol.pdf	Dokumentation der Arbeit im PDF-Format
app/	KORT Frontend
build/Kort/production/	KORT Production Build (komprimierte JavaScript Sourcелефies)
docs/	Generierte Code-Dokumentationen (KORT Frontend, KORT Backend, Ext.ux.LeafletMap)
examples/	Frontend Prototypen
i18n/	Internationalisierungs-Plugin für Sencha Touch
jsduck/	Konfiguration zur Generierung der JSDuck Code-Dokumentation
lib/	Verwendete Library-Pakete
patch/	Sencha Touch Bugfixes
resources/	Ressourcen, welche von KORT verwendet werden (CSS, Bilder, Sprach-Property-Files)
server/	KORT Backend
test/	Tests der Applikation
touch/	Sencha Touch 2 Library
ux/	Sencha Touch Erweiterungs-Komponenten
.travis.yml	Konfiguration für Travis CI
app.js	Einstiegspunkt des KORT Frontends
app.json	Sencha Cmd Konfiguration von KORT
index.html	Startseite

Glossar

API

Application Programming Interface, Schnittstelle für die Programmierung. [3](#), [41–43](#), [48](#), [52](#), [54](#), [58](#), [69–74](#), [79](#), [81](#), [92–94](#)

App-Store

Ein App-Store ist eine Verkaufsplattform eines Betriebssystemherstellers für Smartphones, beispielsweise Google Play für Android, Apple App Store für iOS. [vii](#), [23](#)

Bootstrapping

Bootstrapping wird der Prozess genannt, der auf einem einfachen System ein komplexeres System aktiviert[[8](#)]. Dadurch wird das System ermächtigt, sich selbst zu starten. Deshalb wird Bootstrapping auch Lösung für das Henne-Ei-Problem genannt. [48](#), [54](#)

Camera API

Das Camera API ist eine Spezifikation vom W3C[[7](#)], welches den Zugriff auf die Kamera und die Bilder eines Endgerätes beschreibt. [vii](#)

Cloud

Als Cloud oder Cloud-Computing (*Wolke*) bezeichnet man die Gesamtheit aller Dienste, welche ortsunabhängig im Internet angeboten werden. Dies können zum Beispiel Datenspeicher, Server oder Datenbanken oder schlicht Rechenleistung sein. Der grosse Vorteil der Cloud ist, dass sie sehr leicht skalierbar ist, so dass man die Leistungen dynamisch an den Bedarf angepassen kann[[9](#)]. [52](#)

Continuous Integration

Unter dem Begriff *Continuous Integration*[[4](#)] beschreibt die Idee, dass Änderungen an einer Software schnell eingebracht werden sollen. Dazu zählt, dass diese in einem Versionsverwaltungs-Tool eingetragen und durch automatisierte Tests geprüft werden. [53](#), [102](#)

Crowdsourcing

Unter Crowdsourcing versteht man die Auslagerung von traditionell internen Teilaufgaben an eine Menge von freiwilligen Usern im Internet[[10](#)]. [5](#)

Gamification

Unter Gamification versteht man das Hinzufügen von Spiel-Elementen in einem nicht spieltypischen Kontext[[6](#)]. [iii](#), [v](#), [2](#), [5](#), [7](#), [34](#), [36](#), [92](#), [103](#)

Git

Git ist ein verteiltes Versionsverwaltungssystem für Dateien. Ursprünglich wurde es für die Entwicklung des Linux Kernels kreiert. [52](#), [53](#)

Local Storage

Beim Local Storage handelt es sich um einen Key-Value Speicher des Browsers in welchen Daten einer Webseite persistiert werden können. [17](#), [20](#)

Mapper

Personen welche auf OpenStreetMap die Karten ergänzen und pflegen, nennen sich selbst *Mapper*. [v](#), [7](#), [34](#), [40](#), [97](#), [103](#)

Marshalling

Als Marshalling wird der Vorgang bezeichnet, bei dem strukturierte oder elementare Daten in ein Format umgewandelt werden, welches sich für die Übertragung eignet[[13](#)]. Auf der Empfängerseite geschieht dann ein entsprechendes Unmarshalling um wieder die ursprünglichen Daten zu bekommen. Gängige Formate sind JSON oder XML. [50](#)

Microloader

Der Microloader ist dafür zuständig, beim Starten einer Sencha Touch Applikation alle verwendeten Ressourcen (wie JavaScript- oder CSS-Files) zu laden. [30](#)

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)[[2](#)]. [16](#)

Node

Ein Node ist das grundlegendste Objekt in OpenStreetMap, es wird durch seine Attribute (genannt [Tags](#)) genauer bestimmt. Nodes können Teil eines [Way](#) sein. [44](#), [83](#), [108](#)

OAuth

OAuth ist ein offenes Protokoll, das eine standardisierte, sichere API-Autorisierung erlaubt[[5](#)]. [vi](#), [vii](#), [3](#), [6](#), [49](#), [50](#), [57](#), [58](#), [75–77](#), [87](#)

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. [iv](#), [3](#), [8](#)

Relation

Eine Relation stellt eine Verbindung zwischen verschiedenen OSM-Objekten dar. Relationen werden meist dazu gebraucht, um übergeordnete Beziehungen darzustellen.

Beispielsweise können alle Bushaltestellen einer Buslinie über eine Relation miteinander verknüpft sein. [44](#), [83](#)

REST

Representational State Transfer[\[3\]](#) ist ein Programmierparadigma, welches besagt, dass sich der Zustand einer Webapplikation als Ressource in Form einer URL beschreiben lässt. Auf eine solche Ressourcen können folgende Befehle angewendet werden: GET, POST, PUT, PATCH, DELETE, HEAD und OPTIONS. HTTP ist ein Protokoll welches REST implementiert. [iii](#), [vi](#), [17](#), [18](#), [41](#), [42](#), [46](#), [48](#), [50](#), [51](#), [57](#), [58](#), [75](#), [81](#)

Routing

Routing beschreibt den Vorgang einen optimalen Weg zwischen zwei oder mehr Punkten zu finden. Eine so gefundene Strecke wird oft als *Route* bezeichnet. [5](#), [42](#)

Tag

Ein Tag bezeichnet ein Attribut eines [Nodes](#). Es besteht aus einem Namen und einem Wert. Ein [Node](#) kann beliebig viele Tags beinhalten. [v](#), [3](#), [40](#), [41](#), [107](#)

Way

Ein Weg oder eine Strasse wird in OpenStreetMap als *Way* bezeichnet. Es handelt sich dabei um eine Serie von miteinander verbundenen Nodes. [44](#), [83](#), [107](#)

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, die über einen in das Betriebssystem integrierten Browser aus dem Internet geladen und so ohne Installation auf dem mobilen Endgerät genutzt werden können[\[15\]](#).
[iii](#), [v](#), [vii](#), [2–4](#), [6](#), [13](#), [21](#), [29](#), [40](#), [47](#), [52](#), [54](#), [74](#), [94](#), [102](#)

Literaturverzeichnis

- [1] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON), 2006. [Online; Stand 12. Dezember 2012]. URL: <http://tools.ietf.org/html/rfc4627>.
- [2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (the Gang Of Four). *Design Patterns: Elements of Reusable Object-Oriented Software*. AddisonWesley Professional, 1994.
- [3] 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>.
- [4] Martin Fowler. Continuous Integration, 2006. [Online; Stand 9. Dezember 2012]. URL: <http://www.martinfowler.com/articles/continuousIntegration.html>.
- [5] Ed D. Hardt. The OAuth 2.0 Authorization Framework, 2012. [Online; Stand 6. Dezember 2012]. URL: <http://tools.ietf.org/html/rfc6749>.
- [6] Kevin Werbach, Dan Hunter. *For the Win: How Game Thinking Can Revolutionize Your Business*. Wharton Digital Press, 2012.
- [7] World Wide Web Consortium (W3C). Camera API Specification, 2009. [Online; Stand 10. Dezember 2012]. URL: <http://lists.w3.org/Archives/Public/public-device-apis/2009Apr/att-0001/camera.html>.
- [8] Wikipedia. Bootstrapping (Informatik) — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 6. Dezember 2012]. URL: [http://de.wikipedia.org/w/index.php?title=Bootstrapping_\(Informatik\)&oldid=102848493](http://de.wikipedia.org/w/index.php?title=Bootstrapping_(Informatik)&oldid=102848493).
- [9] Wikipedia. Cloud-Computing — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 10. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Cloud-Computing&oldid=103606763>.
- [10] Wikipedia. Crowdsourcing — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 19. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Crowdsourcing&oldid=111553592>.
- [11] Wikipedia. CRUD — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 11. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=CRUD&oldid=103876970>.

- [12] Wikipedia. Extensible Markup Language — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 20. Dezember 2012]. URL: http://de.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=111516810.
- [13] Wikipedia. Marshalling — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 19. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Marshalling&oldid=102490056>.
- [14] Wikipedia. Same-Origin-Policy — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 6. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Same-Origin-Policy&oldid=96343774>.
- [15] Wikipedia. Webapp — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 6. Dezember 2012]. URL: <http://de.wikipedia.org/w/index.php?title=Webapp&oldid=102886692>.

Abbildungsverzeichnis

0.1.	Anzeige von Fehlern in Osmose	iv
0.2.	Auszeichnungen in KORT	v
0.3.	KORT App	vi
4.1.	Package-Struktur	17
4.2.	Klassendiagramm der Controller	18
4.3.	Klassendiagramm der Stores	19
4.4.	Klassendiagramm der Models ohne Stores	19
4.5.	Speichern der Logininformationen im Local Storage	20
4.6.	Aufbau der Benutzeroberfläche	20
4.7.	Ablauf beim Starten der App	22
4.8.	Maske: Aufträge	26
4.9.	Maske: Prüfen	27
4.10.	Masken: Highscore / Profil / Über KORT	28
4.11.	Frontend Dokumentation mit JSDuck	28
4.12.	iOS Add to homescreen-Funktion	29
5.1.	Ext.ux.LeafletMap-Komponente in Sencha Touch	32
5.2.	Leaflet Map-Komponente im Sencha Market	33
6.1.	Gamification - Sprache	34
6.2.	Gamification - Koins	35
6.3.	Gamification - Badges	35
6.4.	Zusätzliche Berechtigungen bei StackOverflow	37

6.5. Apple Game Center	38
8.1. Übersicht des Gesamtsystems	46
8.2. Deployment-Diagramm	47
9.1. Entwicklungsprozess von KORT	53
10.1. Erster Login des Benutzers mit OAuth	76
10.2. Wiedererkennen des Benutzers	77
10.3. OAuth Einstellungen bei Google	78
10.4. OAuth Einstellungen bei OpenStreetMap	78
10.5. Testresultate von SimpleTest im Browser	80
10.6. Testresultate der Integrationstests im Browser	81
10.7. KORT Backend Dokumentation mit PHPDoc	82
11.1. Die View <code>kort.all_errors</code>	83
11.2. Die Tabelle <code>kort.error_type</code>	85
11.3. Die Tabelle <code>kort.answer</code>	86
11.4. Die Tabelle <code>kort.badge</code>	87
11.5. Login-Buttons in KORT	88

Tabellenverzeichnis

1.1. Übersicht der Arbeitsresultate	4
3.1. Begriffe aus KORT	8
4.1. Abhängigkeiten im Frontend	21
4.2. Build-Möglichkeiten mit Sencha Cmd	23
4.3. Konfiguration von Sencha Cmd (app.json)	23
5.1. Ext.ux.LeafletMap Verfügbarkeit	32
6.1. Auszeichnungen in KORT	35
7.1. Fehler aus OpenStreetMap	41
7.2. FIXME-Tags in OpenStreetMap	41
7.3. OpenStreetBugs	42
7.4. KeepRight	42
7.5. MapDust	43
7.6. Housenumbervalidator	43
7.7. Parameter für setup_keepright.sh	44
7.8. Datenformat der KeepRight-Daten	45
9.1. Datenbankserver	51
9.2. Server bei Heroku	52
9.3. Build-Matrix von Travis CI	54
10.1. Gliederung des Backends	57

10.2. Abhängigkeiten im Backend	58
10.3. Webservice Antworten (GET /answer)	59
10.4. Webservice Fehler (GET /bug)	60
10.5. Webservice Fehler (POST /bug/fix)	61
10.6. Webservice Antworten (GET /highscore)	62
10.7. Webservice OpenStreetMap (GET /osm)	63
10.8. Webservice Benutzer (GET /user)	64
10.9. Webservice Benutzer (GET /user/<id>/badges)	65
10.10. Webservice Benutzer (GET /user/<id>/logout)	66
10.11. Webservice Benutzer (PUT /user)	66
10.12. Webservice Überprüfung (GET /validation)	67
10.13. Webservice Überprüfung (POST /validation/vote)	68
10.14. Webservice Datenbank (GET /db)	70
10.15. Webservice Datenbank (PUT /db)	71
10.16. Webservice Datenbank (POST /db)	72
10.17. Webservice Datenbank (POST /db/transaction)	73
11.1. Fehlertypen in KORT	84
11.2. View-Typen in KORT	85
13.1. Arbeitsaufwand pro Person	102