HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

# Kort Reloaded

# Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2013


| | |
|---|---|
| Autoren: | Carmelo Schumacher, Annrita Egli |
| Betreuer: | Professor Stefan Keller |
| Projektpartner: | bitforge AG Zürich |
| Experte: | Professor Stefan Keller |
| Gegenleser: | Professor Andreas Rinkel |

# 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

**Ausgangslage**

„Kort“ ist ein innovatives, location-based Web App für iPhone, Android und Tablets zur Verbesserung fehlender Daten in OpenStreetMap. Ein gereifter Prototyp davon wurde als Bachelorarbeit im HS 2012 an der HSR entwickelt (vgl. Website: www.kort.ch).

**Aufgabe**

Ziel dieser Arbeit war es, die durchschnittliche Anzahl Interaktionen der User mit der Kort-App zu erhöhen sowie den aktiven Spielerstamm auszuweiten. Dies sollte mittels einem erweiterten Funktionsumfang, der Integration neuer Fehlerquellen und einer verbesserten Spielprozessführung erreicht werden.

**Ergebnis**

Der Funktionsumfang der Kort-App wurde um zeitlich beschränkte, ortsbasierte Kort-Aktionen sowie ein internes, auf einem Atom-Feed basiertes Newssystem erweitert. Damit stehen den Kort-Administratoren neu Werkzeuge zur Verfügung, um das Spielverhalten gezielt anzuregen und das Gefühl der Community-Zugehörigkeit zu stärken. Die Logik zur Einbindung von Fehlern als Spielmissionen wurde überarbeitet und eine zusätzliche Daten-Quelle integriert (Datenbank EOSMDBOne am Institut). Damit können neue Fehlertypen auf generische Art und Weise erstellt und für das Spiel nutzbar gemacht werden, was die Attraktivität für den Spieler und den Nutzen für das OpenStreetMap-Projekt erhöht. Überprüfungen, die für die Güte der beantworteten Missionen verantwortlich sind, werden nun zusammen mit den Missionen auf der Spielkarte angezeigt, was den Anreiz zur Bearbeitung durch die Nutzer erhöhen soll. Zudem wurden weitere Verbesserungen und Fehlerbereinigungen umgesetzt.

# Management Summary


### Ausgangslage

„Kort“ ist ein innovatives, location-based Web App für iPhone, Android und Tablets zur Verbesserung fehlender Daten in OpenStreetMap. Ein gereifter Prototyp davon wurde als Bachelorarbeit im HS 2012 an der HSR entwickelt.

Ziel dieser Fortsetzungsarbeit war es, die durchschnittliche Anzahl Interaktionen der User mit der Kort-App zu erhöhen sowie den aktiven Spielerstamm auszuweiten. Dies sollte über einen erweiterten Funktionsumfang, neuen Fehlerquellen und einer verbesserten Spielprozessführung erreicht werden.

Die ursprüngliche Aufgabenbeschreibung sah lediglich die Implementation der Clientseite vor. Die Serverseite sollte ein Masterstudent als Projektarbeit durchführen und uns via API zur Verfügung stellen. Dieser brach jedoch seine Arbeit ab. Um das Kort-Projekt dennoch abschliessen zu können, übernahmen wir auch die Implementation der Serverseite.

### Vorgehen / Technologien

Um Störungen im laufenden Betrieb der Kort-Applikation zu vermeiden, wurde entschieden, ein Testsetting analog zur Liveversion aufzubauen und auf diesem zu entwickeln. Dazu wurde die BA_HunzikerOderbolzer-Version geforkt und eine Entwicklungsversion der Datenbank auf einem eigenen vServer der HSR aufgesetzt. Nach Projektabschluss wurde die Entwicklungsversion via Pull-Request mit der produktiven Version gemergt.

Bei den von uns verwendeten Technologien legten wir den Fokus auf OpenSource-Produkte. Anzuschaffen gab es lediglich die Webstorm IDE von JetBrains.

### Ergebnisse

Der Funktionsumfang der Kort-App wurde um zeitlich beschränkte, ortsbasierte Kort-Aktionen sowie ein internes, auf einem Atom-Feed basiertes Newssystem erweitert. Damit stehen den Kort-Administratoren neu Werkzeuge zur Verfügung, um das Spielverhalten gezielt anzuregen und das Gefühl der Community-Zugehörigkeit zu stärken. Die Logik zur Einbindung von Fehlern als Spielmissionen wurde überarbeitet und eine zusätzliche Daten-Quelle integriert (Datenbank EOSMDBOne am Institut). Damit können neue Fehlertypen auf generische Art und Weise erstellt und für das Spiel nutzbar gemacht werden, was die Attraktivität für den Spieler und den Nutzen für das OpenStreetMap-Projekt erhöht. Überprüfungen, die für die Güte der beantworteten Missionen verantwortlich sind, werden nun zusammen mit den Missionen auf der Spielkarte angezeigt, was den Anreiz zur Bearbeitung durch die Nutzer erhöhen soll. Zudem wurden weitere Verbesserungen und Fehlerbereinigungen umgesetzt.

## Ausblick

Während unserer Studienarbeit haben wir viele Ideen für die Verbesserung von Kort umsetzen können. Es sind aber nach wie vor Ideen vorhanden, was noch getan werden könnte um die Kort-App noch spannender zu gestalten.

Ein wichtiger Fortschritt wäre das automatische uploaden der von den Kort-Usern vollständig gelösten Aufgaben auf Open Street Map. Zurzeit werden die Lösungen auf der Website von Kort aufgelistet und können von dort aus von Hand auf Open Street Map übertragen werden. Da diese Lösungen von den Kort-Usern bereits überprüft wurden, ist diese manuelle Übertragung ein Aufwand, der automatisierbar wäre.

Die Kort Website[1] wurde von BA_HunzikerOderbolz auf Deutsch erstellt. Da mittlerweile sehr viele Kort-User die App in Englisch nutzen, wäre eine Übersetzung der Website zumindest auf Englisch wünschenswert.

Die Statistiken auf der Website könnten erweitert werden mit Statistiken, die aus Kort-Aktionen hervorgehen. Da die bestehenden Statistiken zum Teil verwirrend dargestellt werden, wäre eine generelle Revision der Statistikseite angebracht.

Mit dieser Revision wären auch zusätzliche Highscores denkbar. Man könnte Tabellen für die fleissigsten Kort-User in verschiedenen Zeitabschnitten erstellen, wie zum Beispiel wer letzten Monat am meisten Punkte gesammelt hat oder wer die meisten Badges gesammelt hat.

[1] http://www.kort.ch/index.html

# Inhaltsverzeichnis

# I Technischer Bericht der Arbeit

## 1. Einleitung und Übersicht

Zur Verbesserung von OpenStreetMap gibt es diverse Ansätze[2]. Einer davon ist die Kort-App, welche wir als Fortsetzungsarbeit weiterentwickeln werden.

Die Kort-App war im Herbstsemester 2012 als Bachelorarbeit von Jürg Hunziker und Stefan Oderbolz (im Folgenden BA_HunzikerOderbolz genannt) erstellt worden. Das ganze Projekt ist auf GitHub verfügbar[3]: Auf der Liveversion der Kort-App[4] sind bereits über Tausend User registriert und die meisten von ihnen haben schon Missionen auf der Kort-App durchgeführt. Die Kort-App hat einen eigenen Twitter Account[5] mit ein paar Duzend Followern womit Neuigkeiten über die App verbreitet werden können. Das Projekt der Kort-App wurde an der Diplomfeier vom März 2013 als beste Bachelorarbeit ausgezeichnet.[6]

Ziel unserer Arbeit war es, die durchschnittliche Anzahl Interaktionen der User mit der Kort-App zu erhöhen sowie den aktiven Spielerstamm auszuweiten. Dies sollte über einen erweiterten Funktionsumfang, neuen Fehlerquellen und einer verbesserten Spielprozessführung erreicht werden. Daraus wurden die folgenden Ziele abgeleitet:

- Zeitlich und räumlich begrenzte Kort-Aktionen
- News aufbauend auf Atom-Feed
- Mission und Überprüfung zusammenlegen -> erweitertes Map-Konzept
- (optional) Sneaky Peak-Funktion (Erweiterungen wie z.B. Anzeige von Missionen ausserhalb der eigenen Rayons oder Bugs beheben)
- (optional) Facebook-Login
- (optional) Relative Highscore

Die ursprüngliche Aufgabenbeschreibung sah lediglich die Implementation der Clientseite vor. Die Serverseite sollte ein Masterstudent als Projektarbeit durchführen und uns via API zur Verfügung stellen. Dieser brach jedoch seine Arbeit ab. Um das Kort-Projekt dennoch abschliessen zu können, übernahmen wir auch die Implementation der Serverseite. Daraus ergaben sich zusätzlich die folgenden optionale Ziele:

- (optional) Webservices anpassen
- (optional) Neue Fehlerquellen (EOSMDBOne)
- (optional) Neue Fehlertypen

---

[2] http://wiki.openstreetmap.org/wiki/Qualitätssicherung
[3] https://github.com/kort/kort
[4] http://play.kort.ch
[5] https://twitter.com/KortGame
[6] https://www.hsr.ch/News-Detail

## 1.1 Aufbau der Arbeit

Diese Arbeit ist in drei Teile gegliedert. Im ersten Teil ist der technische Bericht unserer Arbeit. Dieser ist in vier Kapitel gegliedert: 1. Einleitung und Übersicht (dieses Kapitel), 2. Ergebnisse unserer Arbeit, 3. Die Schlussfolgerungen und 4. Begriffsdefinitionen des gesamten Projekts.

Im zweiten Teil befindet sich die Projektdokumentation. Nach den User Szenarien in Kapitel 5 werden im 6. Kapitel die Änderungen des Kort Frontends beschrieben. Das 7. Kapitel enthält die Umsetzung des erweiterten Map-Konzepts mit den neuen Features. Im 8. Kapitel ist das Backend der App beschrieben.

Der dritte Teil beinhaltet das Projektmanagment, im 9. Kapitel die Sprints, im 10. Kapitel die Anforderungsspezifikationen, im 11. Kapitel die Rahmenbedingungen und schliesslich im 12. Kapitel die Protokolle unserer Projektmeetings.

Der Anhang besteht aus dem Literaturverzeichnis, Tabellenverzeichnis und Abbildungsverzeichnis.

Neben diesem Dokument umfasst die Arbeit die implementierte Web-App Kort. Der dazugehörige Source Code ist frei im Internet zugänglich und auf der beigelegten CD.

| Arbeitsresultat | URL |
|---|---|
| **Kort (Web-App)** | http://kort.herokuapp.com |
| **Repository** | https://github.com/kort |

# 2. Ergebnisse

Trotz Turbulenzen während des Semesters aufgrund der abgebrochenen Projektarbeit des Masterstudenten, ist es uns gelungen, alle definierten Ziele umzusetzen. Dazu gehört die Implementation der Client- sowie der Serverseite. Zu erwähnen ist auch das umfangreiche Refactoring welches wir aufgrund der Glossaränderung und unseren Erweiterungen durchführten.

# 3. Schlussfolgerungen

Der Funktionsumfang der Kort-App wurde um zeitlich beschränkte, ortsbasierte Kort-Aktionen sowie ein internes, auf einem Atom-Feed basiertes Newssystem erweitert. Damit stehen den Kort-Administratoren neu Werkzeuge zur Verfügung, um das Spielverhalten gezielt anzuregen und das Gefühl der Community-Zugehörigkeit zu stärken. Die Logik zur Einbindung von Fehlern als Spielmissionen wurde überarbeitet

und EOSMDBOne als zusätzliche Quelle integriert. Damit können neue Fehlertypen auf generische Art und Weise erstellt und für das Spiel nutzbar gemacht werden, was die Attraktivität für den Spieler und den Nutzen für das OpenStreetMap-Projekt erhöht. Überprüfungen, die für die Güte der beantworteten Missionen verantwortlich sind, werden nun zusammen mit den Missionen auf der Spielkarte angezeigt, was den Anreiz zur Bearbeitung durch die Nutzer erhöhen soll. Zudem wurden weitere Verbesserungen und Fehlerbereinigungen umgesetzt. Hierzu gehört das Refactoring, welches zu einer sauberen, übersichtlichen Codestruktur mit einheitlichen Benennungen führte.

# 4. Begriffsdefinition

**Tabelle 1 Begriffsdefinition**

| Term | Begriff | Kategorie | Beschreibung |
|---|---|---|---|
| **Badge** | Badge | Auszeichnungen | Auszeichnung, die ein *Spieler* gewinnen kann. |
| **Error DB** | Fehler-DB | SW-Komponente | Datenbankmanagementsystem mit Fehler-Daten als Teil *SW-Komponente* des *Kort-Systems* (z.B. KeepRight). |
| **Highscore** | Highscore | Rangliste | Rangliste der erreichten *Koins*, z.B. Liste der Anzahl *Koins* aller *Spieler* seit Spielbeginn (absteigend geordnet). |
| **Koin** | Koin | Belohnung | Punkte, die ein *Spieler* gewinnen kann. Das Wort ist von 'coin' (engl. Münze) abgeleitet. Das 'K' ist eine Anlehnung an Kort. |
| **Kort (mob. Web App** | Kort (mobiles Web App) | SW-Komponente | JavaScript-Applikation als Client zur *Kort Website*, realisiert in HTML5. |
| **Kort Admin** | Kort-Admin | Persona, Rolle | (kurz für Kort-Administrator) Nutzer, der die *Kort Website* mitverwaltet. |
| **Kort API** | Kort API | SW-Komponente | REST API als Schnittstelle zwischen dem *mobilen Web App* und dem Server, angeboten von einem Server. |

| | | | |
|---|---|---|---|
| **Kort Promotion** | Kort-Aktion | Belohnung | erhöht die *Koins* eines Fehlertyps oder mehrerer Fehlertypen. Eine K.A. hat einen Titel, eine Lebensdauer und eventuell eine benannte räumliche Begrenzung. |
| **Kort System** | Kort-System | SW-Komponente | Gesamtheit aller SW-Komponenten von Kort. |
| **Kort Website** | Kort Website | SW-Komponente | Webauftritt als Teil-SW-Komponente des *Kort-Systems*, realisiert als Content Management System (CMS). |
| **Mission** | Auftrag | Spieleinheit | Fehler oder fehlende Daten in der OpenStreetMap-Datenbank, die von einem *Spieler* korrigiert werden. |
| **Object** | Objekt | Punkt auf Karte | Spezielles "Objekt-von-Interesse" (engl. "Point-of-Interest", abgekürzt 'POI'). |
| **Player** | Spieler | Persona, Rolle | Synonym für Benutzer/User. |
| **Solution** | Lösung | Spieleinheit | *Auftrag*, der gelöst und mit genügender Anzahl *Überprüfungen* geprüft wurde. |
| **User** | Benutzer | Persona, Rolle | Benutzer, der hauptsächlich das *mobile Web App* Kort nutzt und eventuell die *Kort Website*. |
| **Validation** | Überprüfung | Spieleinheit | *Auftrag*, der zu prüfen ist (unabhängig vom Spieler, der den Auftrag gelöst hat). Siehe auch *Lösung*. |

Die Tabelle ist sortiert nach Begriff. Alle männlichen Begriffe, wie z.B. Spieler, gelten auch für die weibliche Form.

# II Projektdokumentation

## 5. Szenarios

### 5.1 Szenario 1, Kort-Aktion

Nachdem Andy die Kort App entdeckt hatte, benützte er sie öfters und löste einige Missionen. Seit einiger Zeit hat er aber nicht mehr an Kort gedacht und öffnete die App schon länger nicht mehr. Heute ist er erneut mit dem Zug zur Arbeit unterwegs wo er auf seinem Handy eine Notifikation entdeckt, welche ihm mitteilt, dass auf der Kort-App ab sofort eine Kort-Aktion stattfindet, bei welcher es für alle Gebäude, deren Namen eingetragen werden, für drei Tagen die doppelte Anzahl an Koins gibt. Als Andy aus dem Zug steigt öffnet er die Kort-App und sieht dort, dass es zwei Gebäude in seiner Nähe gibt, welche benannt werden sollten. Da Andy an den beiden Gebäuden vorbeiläuft um zur Arbeit zu kommen, kann er die Namen der Gebäude in die App eingeben und freut sich über die doppelte Anzahl Koins die er dafür erhält. Er nimmt sich vor, wenn er wieder zu Hause ist erneut die Kort-App zu öffnen und nach namenlosen Gebäuden Ausschau zu halten.

Ziele:

- Wiederentdeckung von Kort
- Mit Kort-Aktion mehr Koins verdienen

### 5.2 Szenario 2, Missionen nicht nur aus Umkreis anzeigen

Edi benützt die Kort-App oft und Dank seinen vielen Aktivitäten hält er sich gut in den Top Ten der Highscoreliste. Mittlerweile gibt es in weder bei ihm zu Hause noch an seinem Arbeitsort offene Missionen auf der Kort-App. Als er auf der Karte etwas hin und her scrollt, sieht er inaktive Missionen welche einige Kilometer von seinem Standort entfernt sind. Nun weiss er, dass im Dorf wo seine Schwester wohnt einige Missionen mit der Kort-App zu lösen sind. So beschliesst er, am Wochenende seine Schwester zu besuchen und im Dorf herumzuschauen, um auch diese Missionen erfüllen zu können. So wird er seine schöne Position auf der Highscoreliste beibehalten können.

Ziele:

- Weiter entfernte Missionen entdecken.
- Highscore verbessern

### 5.3 Szenario 3, Aufträge und Überprüfungen auf einer Karte / News

Seit Monika die Kort-App zum letzten Mal gebraucht hat, haben sich Änderungen ergeben. Sie entdeckt, dass es auf der Karte mehr Icons hat als bei ihrem letzten Besuch und dass einige Icons speziell gekennzeichnet sind. Da der News-Tab eine rote Notifikation aufweist, öffnet sie diese und liest, dass neu Aufträge und Überprüfungen

zusammengelegt wurden. Also wechselt sie wieder zurück zur Karte, wo sie jetzt alle offenen Punkte in ihrer Umgebung auf einen Blick sieht.

Ziele:

- News lesen
- Aufträge und Überprüfungen erledigen

# 6. Kort-Frontend

In diesem Kapitel sind die wichtigsten Elemente des Frontends dokumentiert.

Das Kapitel *2.1 Dokumentation* bildet die wichtigste Basis für die Dokumentation des Codes.

Der Abschnitt *2.2 Design* hat dieselbe Struktur wie das Kapitel *4.2 Design* der BA_HunzikerOderbolz (S. 16ff) und soll die wesentlichen strukturellen Zusammenhänge des Codes in Form von Diagrammen zeigen. Dies lässt einen direkten Vergleich der beiden Arbeiten zu, wodurch die Änderungen an der Architektur schnell ersichtlich werden.

Die darauf folgenden Kapitel bieten eine vertiefte Einsicht in die wichtigsten neuen Features.

## 6.1 Dokumentation

Wie bereits bei der Version BA_HunzikerOderbolz wurde darauf geachtet, dass der Code konsequent mit der Sencha-eigenen Dokumentationssprache JSDuck[7] annotiert ist. Das damit automatisch erzeugte API findet sich unter:

```
http://play.kort.ch/docs/Kort
```

## 6.2 Design

Die grundlegende Architektur der Applikation wird durch das MVC-Pattern des Sencha Touch 2-Frameworks und die getroffenen Architekturentscheide in der Version BA_HunzikerOderbolz vorgegeben. Im Folgenden sind die strukturellen Zusammenhänge zwischen den Controllern einerseits und Model und Stores andererseits in Form von Klassendiagrammen dokumentiert.

[7] https://github.com/senchalabs/jsduck

### 6.2.1 Controller-Package

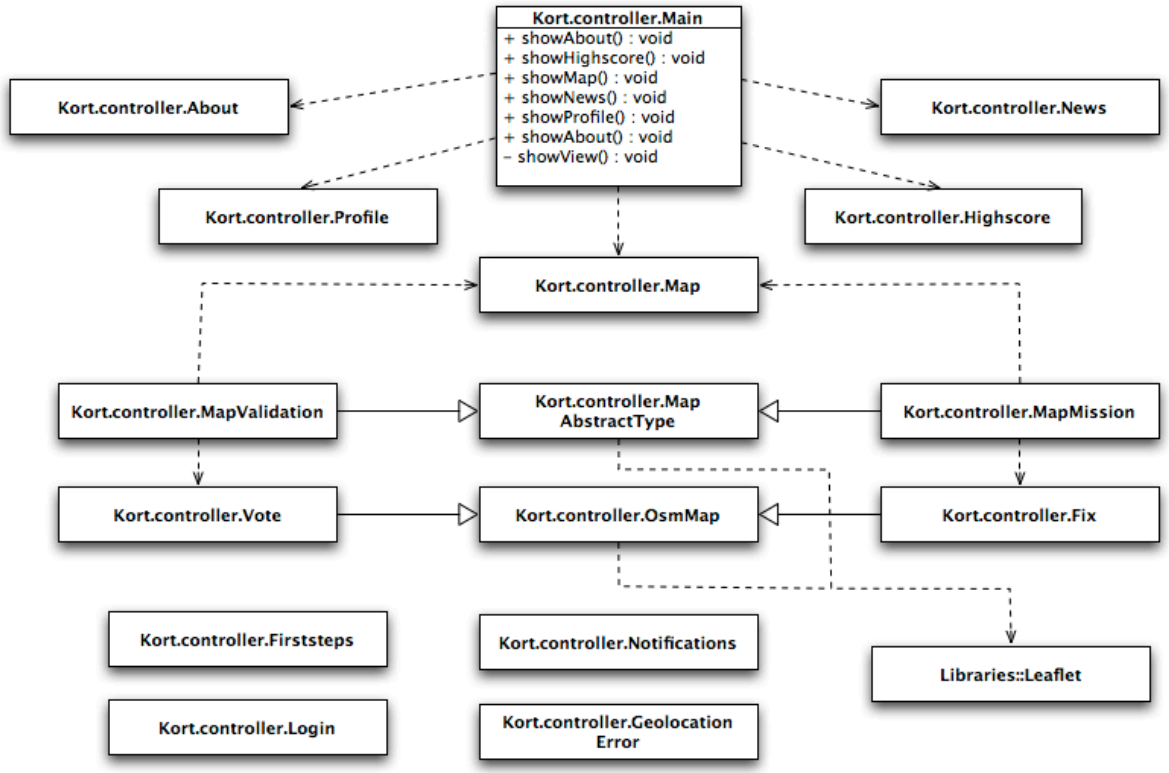



Abbildung 1 Controller Package

### 6.2.2 Model-Package mit remote Stores

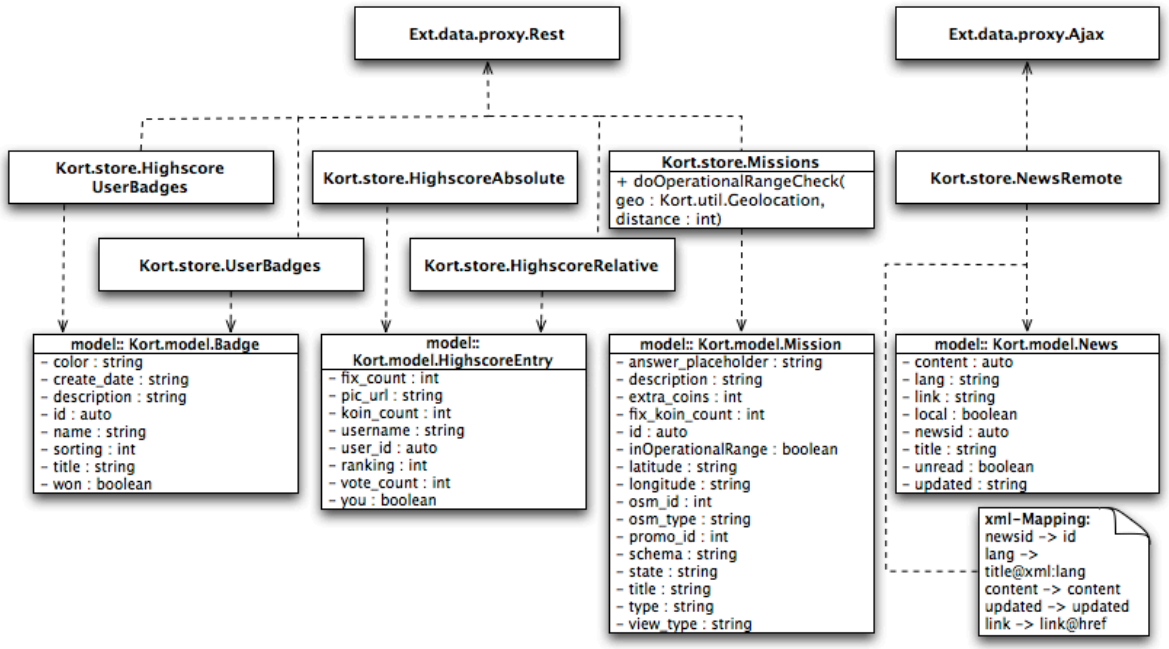



Abbildung 2 Model-Package mit remote Stores I

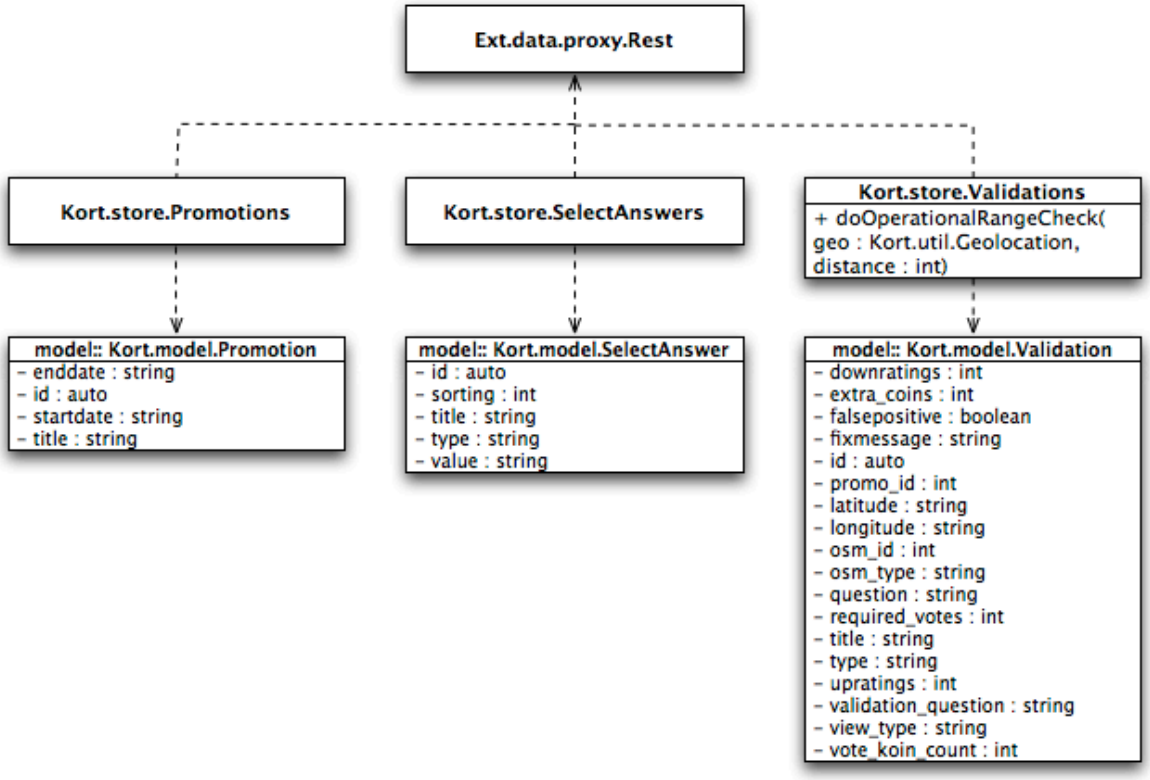



**Abbildung 3 Model-Package mit remote Stores II**

### 6.2.3 Model-Package mit local Stores

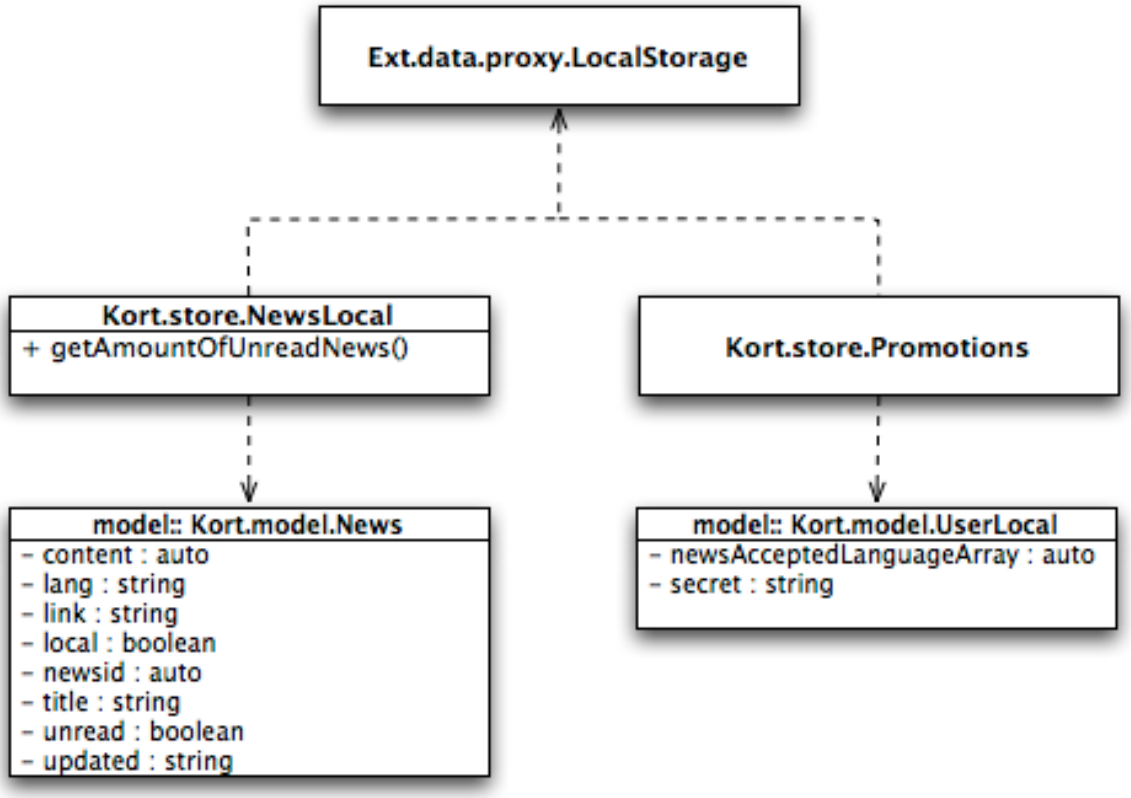



Abbildung 4 Model Package mit local Stores

### 6.2.4 Model-Package ohne Stores

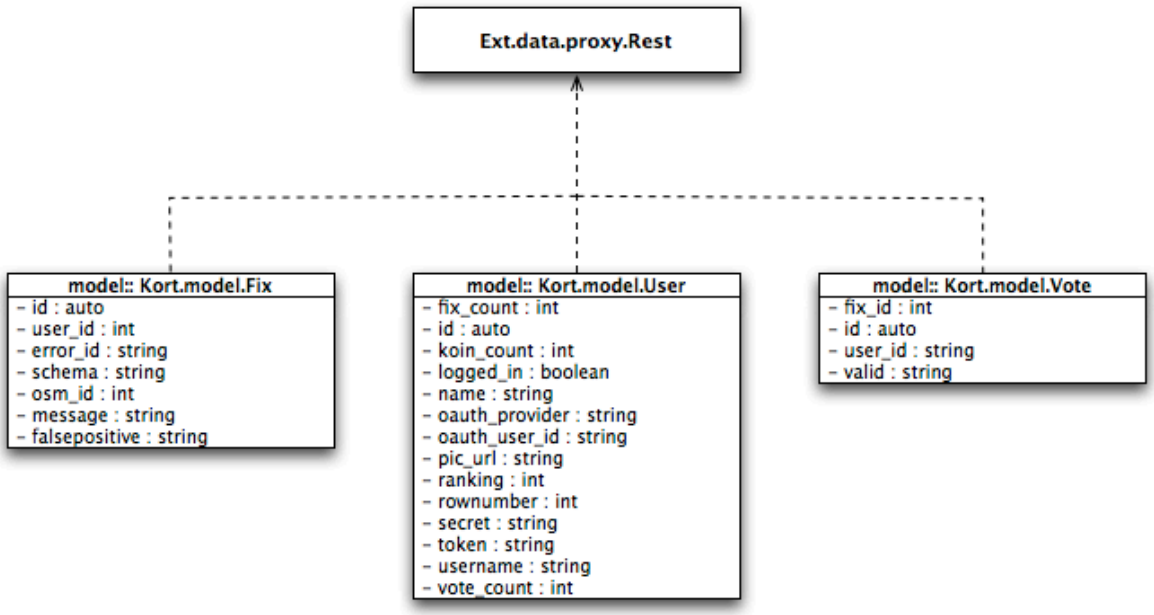



Abbildung 5 Model Package ohne Stores

### 6.2.5 Übersicht Libraries

[BA_HunzikerOderbolz S.33]

**Tabelle 2 Übersicht Libraries**

| 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 OpenStreetMap-Daten auf der Karte |
| **leaflet-osm** | Git Revision 38665cc6c0 | Leaflet-Plugin zur Anzeige von OpenStreetMap - 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 |

# 7. Erweitertes Map-Konzept

In der Kort-Version BA_HunzikerOderbolz wurden lediglich Missionen auf der Leaflet-Karte (Tab 'Map') dargestellt. Validierungen wurden in einem separaten Tab mittels Listenansicht präsentiert. Im Rahmen dieser Erweiterungsarbeit wurden diese zwei Tabs zusammengelegt, so dass nun sowohl Missionen als auch Validierungen gemeinsam auf der Leaflet-Karte dargestellt werden. Der ursprüngliche Tab 'Check' wurde dadurch obsolet. Zusätzlich kann neu einer Mission und einer Validierung auch noch eine Kort-Aktion zugeordnet werden (siehe Kapitel 4. Kort-Aktion). Damit die unterschiedlichen Elemente - in der Leaflet-Terminologie Markers genannt - voneinander unterscheidbar sind, musste das Map-Konzept vollständig überarbeitet werden. Die notwendigen Anpassungen betrafen sowohl den grafischen, wie auch den logischen Teil der Applikation.

## 7.1 Übersicht Architektur

Die Grundidee der angepassten Map-Logik kann am besten anhand der Beziehungen der neuen Kontrollerklassen beschrieben werden:

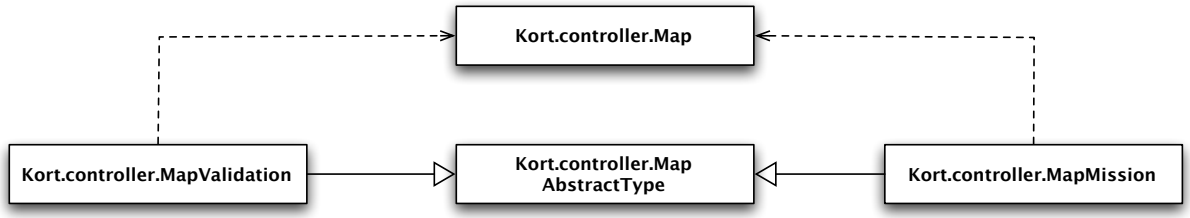



**Abbildung 6 Beziehungen der Kontrollerklassen**

Der Kort.controller.Map fungiert dabei als Master-Controller. Er kapselt über entsprechende API-Funktionen die Interaktion mit dem Leaflet-Plugin und verarbeitet die Events der Hauptnavigationselemente des Map-Tabs.

Klassen, die sogenannte Markers auf die Leaflet-Karte platzieren möchten, erben von dem als abstrakte Klasse konzipierten Kort.controller.MapAbstractType. Diese stellt die Grundfunktion für das Zeichnen von Leaflet-Markers zur Verfügung und definiert den grundlegenden Updateprozess der abstrakt deklarierten Datenstores und die Benachrichtigung des Mainkontrollers über den gegenwärtigen Updatestatus. Auch hat er die für die Interaktion zwischen dem Benutzer und den Markers auf der Karte nötigen Listener registriert und delegiert die Events an die konkreten Kinderklassen.

Kort.controller.MapValidation und Kort.controller.MapMission definieren lediglich die konkrete Logik, was bei einem Klick auf den entsprechenden Markertyp geschehen soll und definieren einen konkreten Store, über den Missions- bzw. Validierungsdaten geladen werden sollen.

Um die Latenz der Applikation möglichst kurz zu halten, wurde darauf geachtet, dass jegliche Kommunikation mit dem Server asynchron und parallel ausgeführt wird. Eine blockierende Lademaske wurde lediglich beim Applikationsstart implementiert, um zu verhindern, dass der Benutzer zu Beginn auf eine leere Map trifft. Weitere Ladevorgänge, wie sie z.B. in Zusammenhang mit der Sneaky-Peak-Funktion (siehe *Kapitel 7.7 Sneaky-Peak*) auftreten, werden nicht blockierend ausgeführt und lediglich ein diskreter Ladeindikator in der rechten oberen Ecke der Applikation informiert den Benutzer über den aktuellen Ladestatus.

Das Zusammenspiel zwischen den einzelnen Komponenten, die für die Map zuständig sind, funktioniert über eine relativ komplexe Eventlogik. Im Anhang dieser Arbeit finden sich ausführliche Sequenzdiagramme, die die Interaktion zwischen den beteiligten Instanzen zeigen.

## 7.2 MapMarker-States

Grundsätzlich kann ein Fehlertyp neu in den folgenden vier verschiedene States auftreten:

**Tabelle 3 Die vier States der Fehlertypen**

| State | Bedeutung |
|---|---|
| **missionState** | Ein Missionsobjekt eines Fehlertyps |
| **missionPromotionState** | Ein Missionsobjekt eines Fehlertyps, für das zudem eine Kort-Aktion definiert ist |
| **validationState** | Ein Validierungsobjekt eines Fehlertyps |
| **validationPromotionState** | Ein Validierungsobjekt eines Fehlertyps, für das zudem eine Kort-Aktion definiert ist |

Zusätzlich gibt es noch den State *inactive*, der zusätzlich zu einem bestehenden Status gesetzt werden kann. Dieser zeigt an, dass sich das betreffende Map-Objekt ausserhalb des definierten Bearbeitungsradius befindet und somit nur betrachtet, nicht aber gelöst werden kann.

Die States werden über das pseudo-Enum 'mapMarkerState' im Config File (Kort.util.Config) definiert.

## 7.3 Icon-Konzept

Für die Kort-Aktion Icons fertigten wir zuerst verschiedene Entwürfe an. Dabei mussten wir auch an den übernächsten Sprint denken, bei welchem wir die Missionen und Überprüfungen in einen Reiter zusammenlegen. Auch dort wird ein Icon gebraucht, welches speziell gekennzeichnet ist, nämlich als Überprüfung. Ist eine Überprüfung gleichzeitig eine Kort-Aktion, so müssen beide Zeichen auf dem Icon ersichtlich sein. Die Erstellung dieser Icons forderte unsere ganze Kreativität und Designkünste.

Die Darstellung der Icons stellte uns auch in diesem Sprint vor eine Herausforderung. Die Icons sollten konsistent sein, also jeder Fehlertyp sollte immer dasselbe Icon haben, es muss jedoch zwischen Aufgabe und Überprüfung unterschieden werden können und ersichtlich sein, wenn es sich um eine Kort-Aktion handelt (aus Sprint 2). Wichtig ist auch die generische Einbindung in die Karte. Schlussendlich ist uns aber eine visuell ansprechende und funktionale Gestaltung der Icons gelungen.

Inaktive Items welche durch die Sneaky Peak Funktion angezeigt werden, sind schwarzweiss.

| Mission | Kort-Aktion | Überprüfung | Kort-Aktion und Überprüfung |
|---|---|---|---|

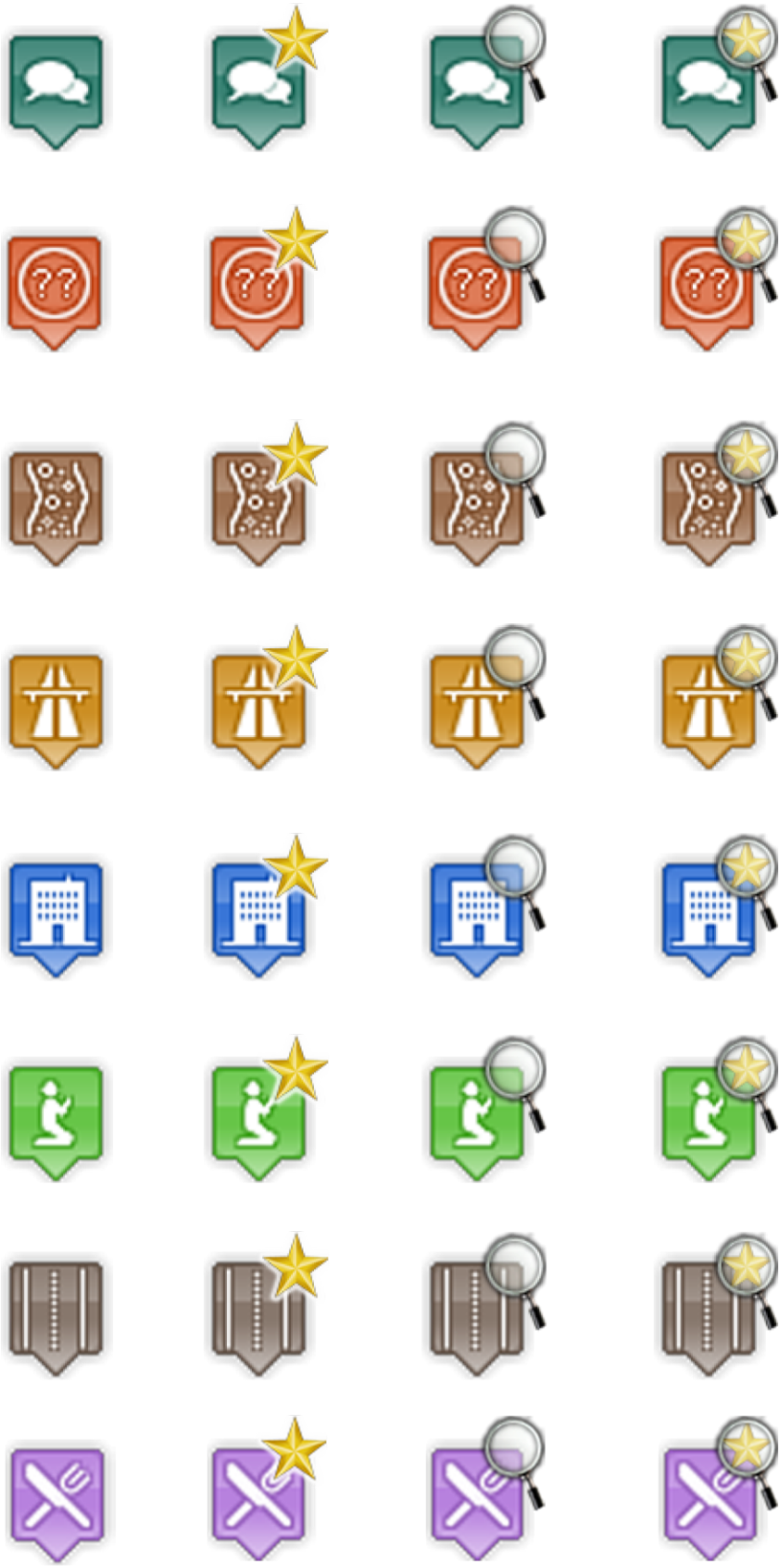


Abbildung 7 Icons mit Kennzeichnung der verschiedenen Status

## 7.4 Layers-Control

Durch die Zusammenlegung von Auftrag und Überprüfung werden dem User unter Umständen sehr viele offene Aufgaben angezeigt. Damit der User trotzdem die Möglichkeit hat, sich nur Überprüfungen oder nur Aufträge anzuzeigen, erstellten wir zwei verschiedene Layer, welche der User mittels Checkboxes ein- oder ausblenden kann. Dazu fügten wir den Leaflet-Markern der Typen Auftrag und Überprüfung einzelne Layer hinzu, deren Sichtbarkeit über ein Layer-Control gesteuert werden kann. Damit können nun wahlweise nur Aufträge, Überprüfungen oder beide Typen auf der Karte angezeigt werden.

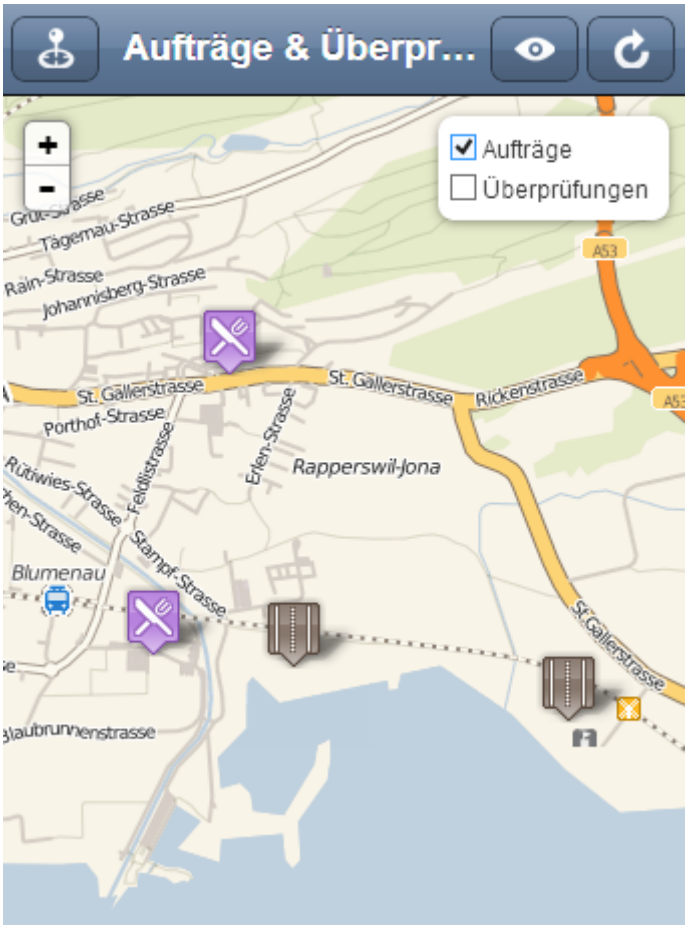



**Abbildung 8 Checkboxes oben rechts um einzelne Layer ein- oder auszublenden**

Wir dachten darüber nach, ein zusätzliches Feature einzubauen, bei welchem alle unterschiedliche Typen von Aufträgen auf dem Layer-Control zu- und weggeschaltet werden könnten. Von dieser Variante sahen wir aber aus Platzgründen ab. Die Anzeige der Auswahl dieser Layer würde einen grossen Teil von kleinen Bildschirmen verdecken.

## 7.5 Neue Masken/Workflows

Auf der bisherigen Kort-App waren die Überprüfungen in einem separaten Tab, die User mussten also aktiv dorthin wechseln um Überprüfungen zu tätigen. Wir änderten den

Workflow bei einer Überprüfung so ab, dass er nun ähnlich wie bei einem Auftrag abläuft.

**Workflow bis anhin:**

Tab Prüfen auswählen -> Liste mit Überprüfungen wird angezeigt.

Kartenansicht auswählen -> Karte mit Überprüfungen wird angezeigt.

Icon anklicken -> Antwortscreen erscheint, wo man ja/nein auswählen kann -> Frage beantworten.

Nach dem Beantworten sieht der User wie viele Koins er soeben für diese Überprüfung erhalten hat.

**Workflow neu mit ValidationMessageBox.js:**

Kort-App starten - Karte mit Aufträgen und Überprüfungen wird angezeigt.

Auf Überprüfungsicon klicken -> ValidationMessageBox.js erscheint mit der Information über die Anzahl Koins, die der User für die Beantwortung erhält.

Auf Beantworten klicken -> Antwortscreen erscheint, wo man ja/nein auswählen kann -> Frage beantworten.

## 7.6 Kort-Aktion

Das Ziel einer Kort-Aktion ist es, die User durch zeitlich und örtlich beschränkte besondere Belohnungen zum Gebrauch der Kort-App zu animieren. In der BA_HunzikerOderbolz wurde dieses Feature folgendermassen beschrieben: „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“*).“

### 7.6.1 Definitionen und Restriktionen

Eine Kort-Aktion besteht aus den folgenden Attributen:

**Tabelle 4 Attribute der Kort-Aktionen**

| Attribut-Name | Beschreibung |
|---|---|
| **Titel** | Jede Kort-Aktion hat einen Titel. Dieser darf aufgrund des beschränkten Platzes der Designvorgabe nicht länger als 40 Zeichen lang sein. |
| **Start** | Eine Kort-Aktion hat einen definierten Startzeitpunkt (Datum und Uhrzeit). |

| | |
|---|---|
| **Ende** | Eine Kort-Aktion hat einen definierten Endzeitpunkt (Datum und Uhrzeit). |
| **Zugeordnete/er Fehlertyp/en** | Eine Kort-Aktion wird einem oder mehreren Fehlertypen zugeordnet. |
| **Regionale Beschränkung** | Eine Kort-Aktion ist räumlich beschränkt. Die Beschränkung wird über ein Multipolygon definiert. |

Eine Mission oder eine Validierung wird demnach einer Kort-Aktion zugeordnet, wenn

a. der Zeitpunkt der clientseitigen Abfrage im zeitlichen Intervall [Start,Ende] der Aktion liegt und
b. der Fehlertyp der Mission/Validierung mit einem in der Aktion definierten Fehlertyp übereinstimmt und
c. der geometrische Punkt der Mission/Validierung innerhalb des für die Aktion definierten Multipolygons liegt

In der Beziehung einer Kort-Aktion mit einem Fehlertyp wird die zusätzliche Anzahl Koins der Missionen und Validierungen festgelegt, die für diese Kort-Aktion gelten:

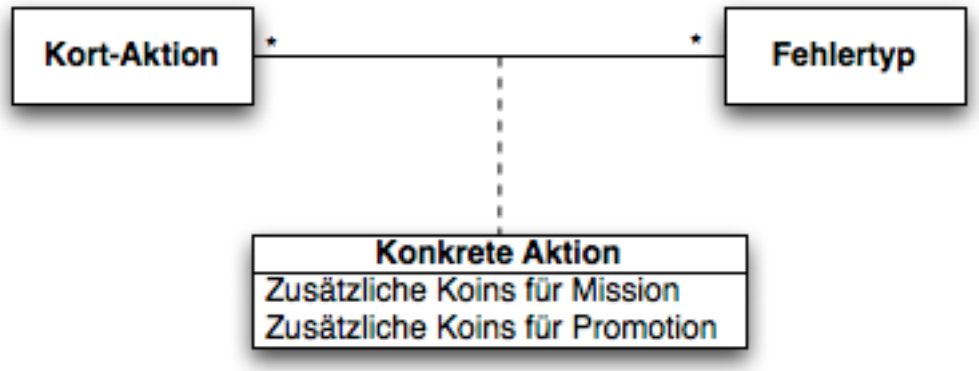



**Abbildung 9 Beziehung Kort-Aktion zu Fehlertyp**

### 7.6.2 Restriktionen

Durch die im vorherigen Abschnitt definierten Eigenschaften für Kort-Aktionen ist es prinzipiell möglich, dass auf eine konkrete Mission/Validierung zwei Kort-Aktionen kommen. Dies ist aufgrund der gewählten Architektur Lösung nicht zulässig und führt zu undefiniertem Verhalten der Applikation. Es wurde definiert, dass der Administrator für die Einhaltung dieser Beschränkung zuständig ist.

### 7.6.3 Neue Masken/Workflows

Beim Auswählen eines Icons einer Kort Aktion, erscheint die PromotionMessageBox.js. Dort wird angezeigt, wie viele extra Punkte für das Lösen dieser Aufgabe vergeben werden. Daneben befindet sich ein Info-Button. Im Hintergrund befindet sich ein orangener Kreis. Wenn ein User den Info-Button drückt, rückt der Kreis in den

Vordergrund und zeigt die Informationen wie Zusatzpunkte und Zeitdauer der Kort-Aktion an.

Im Kort-Aktion-Kreis wird der Titel der Kort-Aktion angezeigt. Dieser Titel sollte eigentlich in verschiedenen Sprachen vorhanden sein, für die verschiedenen Einstellungen der Browser. Wir beschlossen jedoch, dass den Kort-Aktionen fixe Titel vergeben werden sollten und zwar in der Sprache, welche in der Region gesprochen wird, wo die Aktion stattfindet.


**Abbildung 10 MissionMessageBox einer Kort-Aktion mit Infobutton**


**Abbildung 11 ValidationMessageBox mit einer Kort-Aktion mit Infobutton**

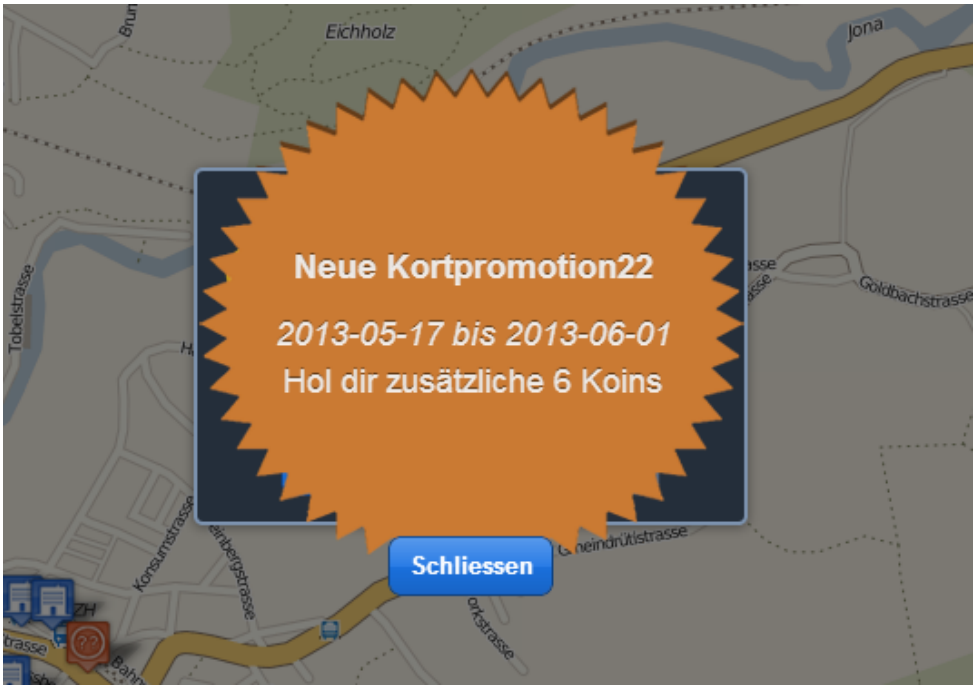



**Abbildung 12 Information über die Kort-Aktion**

### 7.6.4 Neue Kort-Aktion erstellen

Solange noch kein Webinterface für die Andiminstration der Kort-Aktionen zur Verfügung steht, können neue Kort-Aktionen direkt über die Datenbank erstellt werden.

Dazu wird zuerst eine neue Kort-Aktion definiert und danach einem oder mehreren Fehlertyp/en zugeordnet:

**1. Neue Kort-Promotion in Tabelle *kort.promotion* einfügen:**

```
INSERT INTO kort.promotion(title,lang,startdate,enddate,region,geom)
VALUES (

      'FOSSGIS-Promotion',
      'de_DE',
      TIMESTAMPTZ '2013-06-16 00:00:00+02,
      TIMESTAMPTZ '2013-06-18 00:00:00+02',
      'Deutschschweizer Grenze',
      public.ST_GeomFromText('MULTIPOLYGON(((7.844238 48.107431,
      5.185547 45.95115, 9.360352 45.58329, 10.964355 47.100045,
      7.844238 48.107431)))', 4326)
);
```

**2. Bestehende Kort-Promotionen mit Fehlertyp/en verknüpfen:**

Um eine Kort-Aktion mit einem Fehlertyp zu verknüpfen, wird dazu ein entsprechender Eintrag in der Hilfstabelle *kort.promo2mission* eingefügt. Diese Tabelle bildet die n:m Beziehung zwischen Aktion und Fehlertyp ab und stellt die Beziehung über die Fremdschlüssel auf die id der Tabelle kort.Promotion und den error_typ der Tabelle

kort.error_type her. Zudem werden auf dieser Stufe die aus der Beziehung abgeleiteten Attribute mission_extra_coins und validation_extra_coins definiert. Um beispielseweise eine Aktion mit der angenommenen id=22 mit dem Fehlertyp 'missing_cuisine ' zu verknüpfen, wird der folgende Eintrag gemacht:

```
INSERT INTO kort.promo2mission(promo_id, error_type,
mission_extra_coins, validation_extra_coins)
VALUES (

      22,
      'missing_cuisine',
      10,
       5
);
```

## 7.7 Sneaky Peak

Da wir annehmen, dass die Situation wie in Szenario 2 in Kapitel 5.2 beschrieben öfters vorkommen wird, setzten wir uns die Implementation der Sneaky Peak Funktion zum Ziel. Unsere Idee war, dem User die Möglichzeit zu geben, dass er sehen kann wo Aufträge und Überprüfungen offen sind, auch ausserhalb seines Rayons.

Wir entschlossen uns ein Limit zu setzen, so dass immer nur höchstens je 25 inaktive Missionen und Überprüfungen angezeigt werden. Diese sind auf dem Zentrum der Karte wo der User hin gescrollt hat. Damit diese Icons nicht die ganze Zeit vom Server geladen werden müssen, haben wir eingestellt, dass die Icons erst zu laden beginnen wenn der User zwei Sekunden ohne zu Scrollen auf einem Ausschnitt der Karte war. Im Aktualisierungsbutton wird angezeigt, wenn die Funktion am Laden ist. Diese „inaktiven“ Missionen und Überprüfungen werden mit einem schwarz-weissen Icon gekennzeichnet. Das Icon zeigt ebenfalls an, ob es sich um eine Mission oder Überprüfung handelt und ob es zu einer Kort-Aktion gehört. Die Sneaky Peak Funktion kann mit dem entsprechenden Button ein- und ausgeschaltet werden. Tippt der User ein inaktives Icon an, so wird ihm eine Messagebox angezeigt mit der offenen Frage zu diesem Icon. Der User bekommt hier aber keine Möglichkeit diese Frage zu beantworten, er kann nur die Messagebox wieder schliessen.

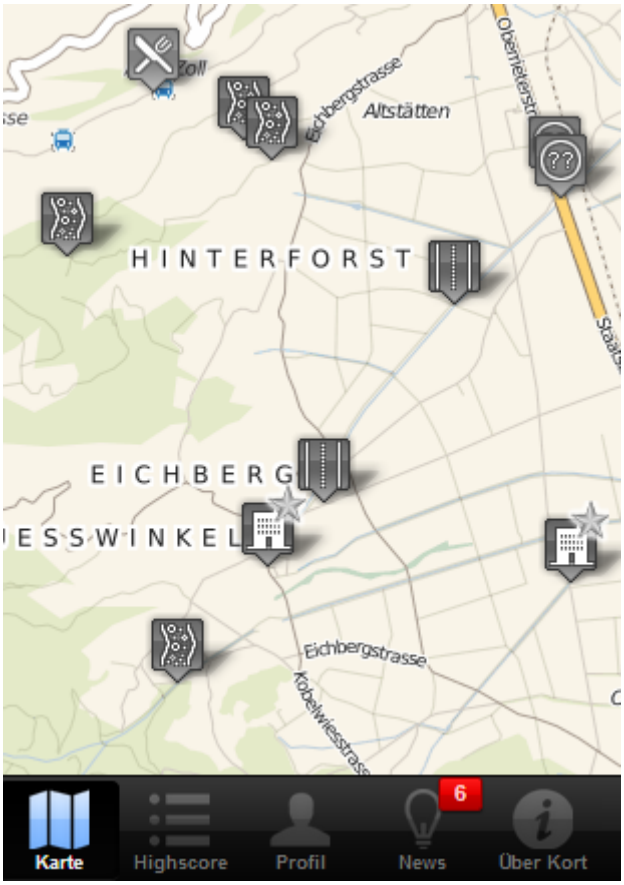



**Abbildung 13 Inaktive Icons Dank der Sneaky Peak Funktion**


**Abbildung 14 Messagebox einer inaktiven Missing-Cuisine-Mission**

## 7.8 Highscore

Neu stehen zwei Arten für die Anzeige der Highscores zur Verfügung.

### 7.8.1 Absolute Highscore

Dies ist die Ansicht wie sie auch in der bisherigen App bekannt war. Die besten 10 User werden dort aufgelistet, an elfter Stelle steht der Rang des aktuellen Benutzers mit seiner Punktzahl.

### 7.8.2 Relative Highscore

Wir vom User der Tab “Relativ” ausgewählt, sieht er seine Umgebung der Highscoreliste, dort kann er sehen wie viele Punkte er benötigt um in der Rangliste emporzuklettern.

## 7.9 News

Wir bauten einen neuen Tab in die App: “News“. In diesem Tab gibt es zwei verschiedene Views: Öffnet der User den Tab so sieht er eine Liste, wo alle News aufgelistet sind. Beim Auswählen eines Newseintrags erscheint die Detailview mit Titel, Erscheinungsdatum und Beschreibung.


Abbildung 15 News List


Abbildung 16 News Detailview

Die Nachrichten erscheinen auf der Newslist sortiert nach Publikationsdatum, die ungelesenen Nachrichten in fetter Schrift und mit einem Stern gekennzeichnet. Sobald der Newseintrag in der Detail-View angeschaut wurde, erscheint er im normalen Font.

Für die Präsentation der News wurde ein minimaler RSS-Atom-Reader implementiert. Die Lösung beruht auf zwei Stores, dem NewsRemote-Store, der mittels XML-Reader den

Atom-Feed parst, und dem NewsLocal-Store, der die News aus dem RemoteStore synchronisiert und einen persistenten lokalen State davon anlegt. Die Anzeige als Liste basiert auf den Daten des NewsLocal-Stores. Ausgehend vom Model-Feld 'unread' werden die zwei States 'unread'=true und 'unread'=false unterschieden. Wird auf der news.List ein Listenelement angeklickt, so wird über den News-Controller die darin definierte Funktion `onNewsListItemTap` aufgerufen, die den State des betroffenen Records auf 'unread'=false setzt, den Store synchronisiert und danach neu lädt. Die DetailView wird zum entsprechenden Record gepusht. In der Folge wird die List aktualisiert und der BadgeText dynamisch angepasst. Mit dem Back-Button kann wieder zur news.List zurückgekehrt werden.

Die Datenhoheit wurde an den Atom-Feed übertragen. Das heisst, wenn ein Admin die Atom-Feed Entries löscht, werden diese auch aus dem NewsLocal-Store entfernt.

Per Default werden dem User alle News in allen Sprachen angezeigt. Dies ist momentan sinnvoll, da momentan alle News in Englisch abgefasst sein werden. Im News-Tab gibt es einen Settings-Button, mit welchem der User das Einstellungspanel öffnet, wo ausgewählt werden kann, von welchen Sprachen er die News angezeigt bekommen möchte. Diese Einstellungen werden im Localstorage im UserLocal-Model gespeichert, was als Filter auf den NewsLocal-Store wirkt.


Abbildung 17SettingsPanel für Sprachauswahl

### 7.9.1 Atom-Feed

Die Atom-Feed-URL kann in Kort.util.Config definiert werden:

```
newsAtomFeedUrl: './resources/stores/news_default.xml'
```

Das Sprachattribut `xml:lang` wird im Titel der Entry definiert.

Beispiel von zwei Newseinträgen in verschiedenen Sprachen:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">

    <title>Kort Newstitle></title>
    <link href="http://kort.ch/"/>
    <updated>2013-12-13T18:30:02Z</updated>
    <author>
        <name>Kort Team</name>
    </author>
```

```
        <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>

        <entry>
            <title xml:lang="en">New Promotion In Rheintal</title>
            <link href="http://example.org/2003/12/13/atom03"/>
            <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa7a</id>
            <updated>2013-04-16T18:30:02Z</updated>
            <content type="html">Different english text</content>
        </entry>
        <entry>
            <title xml:lang="fr">Nouvelle promotion dans la
Romandie</title>
            <link href="http://example.org/2003/12/13/atom03"/>
            <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa8a</id>
            <updated>2013-04-16T18:30:02Z</updated>
            <content type="html">Un texte en français</content>
        </entry>

    </feed>
```

### 7.9.2 Notifikation

Damit den Usern das Lesen einer Newsnachricht nicht aufgezwungen wird, haben wir uns dazu entschlossen, dem News Tab einen Badgetext als Notifikation hinzuzufügen. So sieht der User ob es neue News hat und es steht ihm frei, ob er die News in der Newsliste anschauen will.

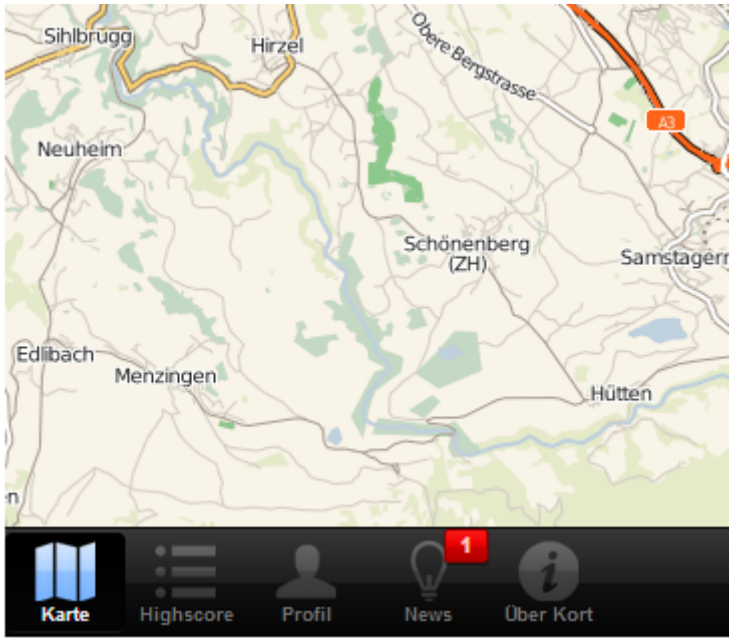



**Abbildung 18 : Notifikation neuer News durch einen Badgetext**

Die Notifikation haben wir mit der Funktion

```
updateNewsBadgeText: function() {
      this.getTabPanel().getTabBar().getComponent(3)
            .setBadgeText(Ext.getStore('NewsLocal')
                  .getAmountOfUnreadNews());
}
```

im Notification-Controller eingefügt.

Der BadgeText für den News-Tab-Button wird dynamisch anhand der ungelesenen News im NewsLocal-Store mit der Funktion `getAmountOfUnreadNews` bestimmt. Werden im NewsLocal-Store Modelstates verändert oder kommen neue News-Items aus dem NewsRemote-Store hinzu, wird systemweit der Event 'newsupdated' gesendet. Der Controller Notifications.js hört auf dieses Event und setzt den BadgeText auf den aktuellen Wert (über die Hilfsfunktion `'getAmountOfUnreadNews'`, die direkt innerhalb des NewsLocal-Store implementiert ist). Sobald der User dieses neue Item in der Detailansicht angeschaut hat, wird der BadgeText entsprechend angepasst. So sieht der User immer, wie viele ungelesene News noch vorhanden sind. Die News werden nach Datum sortiert. Die News müssen und können vom User nicht gelöscht werden. Da wir der Meinung sind, dass veraltete News schlechte News sind, stellen wir dem Admin die Anforderung, nicht mehr aktuelle News zu löschen.

### 7.9.3 Permalink

Wird ein User mittels einer News über eine Kort-Aktion informiert, wird in der Detailansicht ein Link sein, welcher den User auf die Map führt und zwar zum Mittelpunkt der Kort-Aktion. So kann der User die Gegend auf der Karte erkunden gehen und sieht, sofern er die Sneaky Peak Funktion (siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** Sneaky Peak) aktiviert, anhand der inaktiven Icons mit Stern wo sich die Kort-Aktionen befinden. Dazu mussten wir einen Permalink implementieren. Ein Beispiel für einen Permalink steht hier: http://kort.herokuapp.com/#map/permalink?lat=51.212345&lng=8.45647654. Mit „lat“ und „lng“ werden die Koordinaten des Punkts angegeben. Der Link kann auch um einen Zoomlevel ergänzt werden. Der Default Zoomlevel ist 15. Will der User einen anderen Zoomlevel, kann er dem Link ein &z=[zoomlevel] anfügen. Zum Beispiel: http://kort.herokuapp.com/#map/permalink?lat=48.5275538&lng=7.9756324&z=10

## 7.10 Facebook-Login

Als zusätzliches Feature erledigten wir während diesem Sprint die Implementation des Facebook oAuth. Dies war ein Wunsch von Kort-Usern[8] und wurde bereits in der Bachelorarbeit[9] als wünschenswertes Feature erwähnt. Dazu passten wir zuerst auf dem Startscreen das GUI an, indem wir unter den Google-Loginbutton den Facebook-Loginbutton hinzugefügten. Danach konnten wir der Anleitung auf Facebook[10] folgen

[8] https://github.com/kort/kort/issues

[9] https://github.com/kort/kort-docu/blob/master/_DOCUMENTATION/ba-kort-jhunzike_soderbol.pdf

[10] https://developers.facebook.com

um Facebook als weiteren oAuth2.0 Provider aufzunehmen. Nun können sich die Kortuser auch mittels ihres Facebook-Account bei der Kort-App anmelden.

# 8. Kort-Backend

## 8.1 Webservices

Als plattformunabhängiges Kommunikationsparadigma zwischen den verteilten Instanzen wurde REST eingesetzt. Die REST-Webservices wurden mit dem PHP Microframework Slim erstellt.

Die Struktur der Webservices musste gegenüber der Version BA_HunzikerOderbolz nur leicht erweitert werden (siehe nachfolgendes Kapitel 8.1.1 Übersicht). Als Basis für die Auflistung der Webservices wurde wo möglich die Ausführungen aus der Bachelorarbeit BA_HunzikerOderbolz '10.2 REST-Schnittstellen' übernommen.

Komplexere Anpassungen ergaben sich jedoch in Zusammenhang mit den Kort-Aktionen an den Hilfsmethoden der Klassen BugHandler, ValidationHandler, FixHandler und VoteHandler, über die Daten mittels Datenbankaufrufe abgerufen und aufbereitet werden. Umfangreiche Kommentare zu den veränderten SQL-Queries sind direkt innerhalb der entsprechenden Klassen im Verzeichnis server/php/Webservice/ zu finden.

### 8.1.1 Übersicht

**Client <--> Webserver (Heroku)**

**Tabelle 5 Übersicht Kommunikation von Client und Webserver**

| Typ | Pfad | Zweck | Δ BA_HunzikerOderbolz |
|---|---|---|---|
| **GET** | answers/<type> | Alle Antworten eines Fehlertyps <type> | |
| **GET** | highscore/absolute | Benutzer sortiert nach Anzahl Koins plus - falls noch nicht im Antwortset enthalten - der aufrufende Benutzer | VORHER highscore/ |
| **GET** | highscore/relative | Benutzer sortiert nach Anzahl Koins | NEU |
| **GET** | mission/position/ <lat>,<lng> | Limitierte # Missionen nearest neighbor-Sortiert nach dem Punkt(<lat>,<lng>) | VORHER /bug/position/ <lat>,<lng> + ERGÄNZTE RÜCKGABE |
| **POST** | mission/fix | Lösung senden | VORHER /bug/fix |
| **GET** | osm/<type>/<id> | OSM Objekt vom Typ {node,line} und Id | - |
| **GET** | promotion/ | Alle Kort-Aktionen | NEU |

| | | | |
|---|---|---|---|
| **GET** | user/<secret> | Benutzerdaten von User mit <secret>laden, falls Match mit Datenbank (<secret> optional) | - |
| **GET** | user/<id>/badges | Badge des Users mit <id> laden | - |
| **GET** | user/<id>/logout | User mit <id> ausloggen | - |
| **PUT** | user/<id> | Benutzerdaten von User mit <id> modifizieren | - |
| **GET** | validation/position/ <lat>,<lng> | Limitierte # Überprüfungen neighbor-Sortiert nach dem Punkt(<lat>,<lng>) | ERGÄNZTE RÜCKGABE |
| **POST** | validation/vote | Überprüfung senden | - |

### Webserver (Heroku) <--> Datenbankserver (sinv-56055.edu.hsr.ch)

**Tabelle 6 Übersicht Kommunikation von Webserver und Datenbankserver**

| Typ | Pfad | Zweck | Δ BA_HunzikerOderbolz |
|---|---|---|---|
| **GET** | db/<table>/<fields> | Daten von Postgres-Datenbank abrufen | - |
| **POST** | db/<table>/<fields> | Daten in Postgres-Datenbank einfügen | - |
| **PUT** | db/<table>/<fields> | Daten in Postgres-Datenbank modifizieren | - |
| **POST** | db/transaction | Generische Transaktion auf Postgres-Datenbank ausführen | - |

## 8.1.2 Antworten /answer

[BA_HunzikerOderbolz S.59]

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

**Tabelle 7 Webservice Antworten (GET /answer)**

| | |
|---|---|
| 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 | |

**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"
      },
      { ... }              }
   ]
}
```

### 8.1.3 Highscore /highscore

[nach BA_HunzikerOderbolz S.62]

Über den Highscore-Webservice können die Benutzer nach Anzahl Koins geladen werden. Beim absoluten Highscore wird - im Unterschied zum relativen Highscore - der aufrufende Benutzer zusätzlich angefügt (falls dieser nicht bereits im Antwortset enthalten ist).

### Absoluter Highscore laden

**Tabelle 8 Webserver Highscore (GET /highscore/absolute)**

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

### Relativer Highscore laden

**Tabelle 9 Webserver Highscore (GET /highscore/relative)**

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

### Beispiele:

```
GET http://kort.herokuapp.com/server/webservices/highscore/
absolute?limit=10

GET http://kort.herokuapp.com/server/webservices/highscore/
relative?limit=10
```

### Antwort:

```
{
   "return":[
      {
         "user_id":"3",
```

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


### 8.1.4 Auftrag /mission

[nach BA_HunzikerOderbolz S.59ff]

Der Webservice /mission liefert eine limitierte Anzahl Missionen, nearest-neighbor-Sortiert in Bezug auf einen mitgegeben Positionspunkt zurück.
Zusätzlich kann über diesen Webservice eine Lösung zu einem Fehler eingetragen werden.

#### Missionen laden

**Tabelle 10 Webservice Auftrag (GET /mission/position/<lat>,<lng>)**

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

**Beispiel:**

```
GET http://kort.herokuapp.com/server/webservices/mission/
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...
         "latitude":"47.0995850000000000",
         "longitude":"8.0979140000000000",
         "view_type":"select",
         "answer_placeholder":"Typ",
         "fix_koin_count":"5",
         "txt1":"",
         "txt2":"",
         "txt3":"",
         "txt4":"",
         "txt5":"",
         "promo_id":"22",
         "extra_coins":"6",
      },
      { ... }         ..
   ]
}
```

### Lösung senden

**Tabelle 11 Webservice Auftrag (POST /mission/ fix)**

| | | |
|---|---|---|
| URL | http://kort.herokuapp.com/server/webservices/mission/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 | |

**Beispiel:**

```
POST http://kort.herokuapp.com/server/webservices/mission/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"
}
```

### 8.1.5 OpenStreetMap /osm

[BA_HunzikerOderbolz S.59]

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[11] weiter und sendet das Resultat an die Webapplikation zurück.

#### OpenStreetMap Objekt laden

**Tabelle 12 Webservice OpenStreetMap (GET /osm/<type>/<id>)**

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

**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/">
```

[11] http://wiki.openstreetmap.org/wiki/API_v0.6

```
   <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>
```

### 8.1.6 Kort-Aktion /promotion

Der Webservice /promotion wird verwendet, um alle Kort-Aktionen abzurufen.

**Kort-Promotionen laden**

**Tabelle 13 Webservice Kort-Aktion (GET /promotion)**

| | | |
|---|---|---|
| URL | http://kort.herokuapp.com/server/webservices/promotion | |
| Methode | GET | |
| Parameter | - | |
| Antwort | 200 OK | Daten konnten erfolgreich geladen werden. |
| Antworttyp | JSON | |

**Beispiel:**

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

**Antwort:**

```
{
   "return":[
      {
         "id":"22",
         "title":"Neue Kortpromotion22",
         "startdate":"28.05.2013",
         "enddate":"01.06.2013"
      },
      { ... }          ...       }
   ]
```

}

### 8.1.7 Benutzer /user

[BA_HunzikerOderbolz S.64ff]

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**

**Tabelle 14 Webservice Benutzer (GET /user/<secret>)**

| | | |
|---|---|---|
| URL | http://kort.herokuapp.com/server/webservices/user/[<secret>]<br><br><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 | |

**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\/..."
      "logged_in":true
   }
}
```


### Badges eines Benutzers laden

**Tabelle 15 Webservice Benutzer (GET / user/<id>/badges )**

| | | |
|---|---|---|
| URL | http://kort.herokuapp.com/server/webservices/user/<id>/badges<br><br><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 | |

**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 ...",
```

```
        "color":"#FFFBCB",
        "sorting":"110",
        "won":true,
        "create_date":"13.12.2012 18:56"
    },
    { ... }          ...      }
]
```


### Logout

**Tabelle 16 Benutzer (GET / user/<id>/logout )**

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

### Beispiel:

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

### Antwort:

```
Congratulations! You've now officially logged out!
```

### Benutzerdaten ändern

**Tabelle 17 Benutzer (PUT / user/<id> )**

| | |
|---|---|
| URL | http://kort.herokuapp.com/server/webservices/user/[<id>]<br><br><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 | – | |

**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/... ",
   "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":"tschortsch",
   "oauth_user_id":"user@oauth.com",
   "secret":"secret"
}
```

### 8.1.8 Überprüfung /validation

[nach BA_HunzikerOderbolz S.67ff]

Der Webservice /validation liefert eine limitierte Anzahl Überprüfungen, nearest-neighbor-Sortiert in Bezug auf einen mitgegeben Positionspunkt zurück.

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

#### Überprüfungen laden

**Tabelle 18 Webservice Überprüfung (GET / validation/position/<lat>,<lng>)**

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

**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 ...",
            "latitude":"47.3370456000000000",
            "longitude":"8.5207856000000000",
            "upratings":"0",
            "downratings":"0",
            "required_validations":"3"
             "promo_id":"0",
            "extra_coins":"3"
        },
        { ... }          ...
    ]
}
```


### Überprüfung eintragen

**Tabelle 19 Webservice Überprüfung (POST / validation/vote)**

| | | |
|---|---|---|
| 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 | Der Benutzer ist nicht korrekt eingeloggt und kann somit keine Daten an den Server senden. |
| Antworttyp | JSON | |

**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"
}
```

### 8.1.9 Datenbank /db

[BA_HunzikerOderbolz S.69ff]

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

**Tabelle 20 Webservice Datenbank (GET / db/<table>/<fields>)**

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

**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

**Tabelle 21 Webservice Datenbank (PUT / db/<table>/<fields>)**

| | | |
|---|---|---|
| URL | http://kort.rdmr.ch/webservices/db/<table>/<fields><br><br><table> Datenbank-Tabellenname (inkl. Schema)<br><fields> Komma-separierte Liste von Feldern der Tabelle | |
| Methode | PUT | |
| Parameter | key API-Key<br>where WHERE-Klausel der Abfrage (Einschränkung)<br>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 | |

**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

**Tabelle 22 Webservice Datenbank (POST / db/<table>/<fields>)**

| | | |
|---|---|---|
| URL | http://kort.rdmr.ch/webservices/db/<table>/<fields><br><br><table> Datenbank-Tabellenname (inkl. Schema)<br><fields> Komma-separierte Liste von Feldern der Tabelle | |
| Methode | POST | |
| Parameter | key API-Key<br>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 | |

**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

**Tabelle 23 Webservice Datenbank (POST / db/transaction)**

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

**Beispiel:**

```
POST http://kort.rdmr.ch/webservices/db/transaction
{
   [   [
      {
         "type":"SQL",
         "sql" : "select vote_koin_count from kort.valida..."
      },
      {
         "type":"UPDATE",
         "fields":"username",
         "table":"kort.user",
         "where":"user_id=7",
         "returnFields":"user_id ,username",
         "data":{
```

```
            "username":"newUsername"
         }
      }
   ]
}
```

**Antwort:**

```
{
   [
      {
         [
            {
               "vote_koin_count":5
            },
            { ...}
         ]
      },
      {
         "user_id":7,
         "username:"newUsername"
      }
   ]
}
```

## 8.2 Fehlerquellen

Grundlage aller Kort-Missionen sind Fehler in den OpenStreetMap-Kartendaten. Im Rahmen der Bachelorarbeit BA_HunzikerOderbolz wurden verschiedene Quellen für aufbereitete Fehlerdaten evaluiert (siehe BA_HunzikerOderbolz 'Evaluation von geeigneten Datenquellen') und Keepright[12] aufgrund der Güte der Daten und gutem API-Zugriff ausgewählt. Aus den umfangreichen Keepright-Daten wurden diejenigen Fehlertypen lokalisiert, dies sich aufgrund ihrer wohldefinierten Struktur automatisiert bearbeiten lassen (siehe Kapitel 8.2.4 Übersicht Fehlertypen).

Die Verfügbarkeit von noch ungelösten Fehlern ist essentiell für den Spielfluss von Kort. Da sich nur ein Subset der Keeprightdaten zur Aufbereitung von Kort-Missionen eignet und die Anzahl Fehler pro Typ natürlich begrenzt ist, wurde der Wunsch nach neuen Fehlerquellen laut, aus denen sich neue Fehlertypen ableiten lassen.

Als gewichtiger Teil dieser Arbeit wurde EOSMDBOne[13] als weitere Fehlerquelle an die Applikation angebunden. Im Unterschied zu Keepright lassen sich damit Fehlertypen aus einer booleschen Kombination von OSM-Attributen generieren (weiter Informationen finden sich in den Folgekapiteln Übersicht Fehlerquellen, Übersicht Fehlertypen und Neuer Fehlertyp hinzufügen).

### 8.2.1 Anforderungen an Fehlerquellen

Damit Fehlertypen von verschiedenen Fehlerquellen reibungslos integriert werden können, muss eine gemeinsame Struktur für den generischen Fehlertyp definiert werden, die von der Fehlerquelle bedient werden kann.

Es gelten die folgenden grundsätzlichen Anforderungen an eine Fehlerquelle [BA_HunzikerOderbolz, S.83]:

- 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

Per Konvention wurde zudem festgelegt, dass jede Fehlerquelle über ein eigenes Datenbankschema integriert wird.

In der Applikationslogik werden die Fehler aller Fehlerquellen schlussendlich in der View kort.all_errors zusammengezogen. Die Felder der Fehlerquellen müssen sich

[12] http://keepright.ipax.at

[13] http://giswiki.hsr.ch/EOSMDBOne

demnach auf die in der View definierten Felder abbilden lassen. Die Definition dieser View lautet wie folgt:

**Tabelle 24 Felder der View kort.all_errors**

| Feldname | Feldtyp | Bedeutung |
|---|---|---|
| source | character varying(20) | Bezeichner der Fehlerquelle (siehe Kapitel 8.2.2 Übersicht Fehlerquellen) |
| error_id | bigint | ID des Fehlers, innerhalb eines Schema eindeutig |
| schema | character varying(6) | Zur Unterteilung der Fehlerdaten in Bereiche. |
| error_type_id | integer | Id des Kort-Fehlertyps (siehe Kapitel 8.2.4 Übersicht Fehlertypen) |
| osm_id | bigint | Global eindeutige OSM-ID |
| osm_type | keepright.osm_type | OSM-Typ des Fehlers ('node' \| 'way' \| 'relation') |
| description | text | Fehlerbeschreib (nur deskriptiv, keine Bedeutung im Code) |
| latitude | numeric | Integer, der Dividiert durch den Faktor 10^7 den effektiven Breitengrad ergibt (damit keine Fliesskommazahlen in DB) |
| longitude | numeric | Integer, der Dividiert durch den Faktor 10^7 den effektiven Längengrad ergibt (damit keine Fliesskommazahlen in DB) |
| geom | public.geometry(Point,4326) | Postgis Geometrie-Objekt (Point, SRID: 4326) |
| txt1 | text | Texte für Platzhalter in msgid |
| txt2 | text | Texte für Platzhalter in msgid |
| txt3 | text | Texte für Platzhalter in msgid |
| txt4 | text | Texte für Platzhalter in msgid |
| txt5 | text | Texte für Platzhalter in msgid |

Anmerkung zur View kort.all_errors:

- Das Tripel (error_id, schema, osm_id) kennzeichnet einen Fehler applikationsglobal eindeutig.
- Die Platzhalter txt1-txt5 können in der Funktion von Variablen im für den User sichtbaren Fehlertext eingebunden werden. Am Beispiel des Fehlertyps "missing_cuisine" wie folgt: Welche Art der Küche hat '$1'? $1 wird dann bei der Ausgabe mit dem Text aus txt1 ersetzt, z.B. 'Restaurant Geeren'.

### 8.2.2 Übersicht Fehlerquellen

Im Rahmen dieser Arbeit wurde in Ergänzung zur bestehenden Fehlerquelle 'Keepright' der Version BA_HunzikerOderbolz, EOSMDBOne als zusätzliche Fehlerquelle integriert. Um Interferenzen bei den Fehlertypen verschiedener Fehlerquellen zu vermeiden, mussten verbindliche Bereiche für die Fehlertypen-Attribute 'error_type_id' und 'schema' definiert werden.

**Tabelle 25 Übersicht Fehlerquellen**

| Bezeichner | DB-Schema | Intervall error_type_id | Intervall schema | Region der Fehler | Art der Quelle |
|---|---|---|---|---|---|
| **Keepright**[14] | keepright | ( 0, 1000 ] | ( 0, 1000 ] | Global | CSV |
| **EOSMDBOne**[15] | osm_errors | ( 1000, 1100 ] | ( 1000, 1100 ] | Schweiz | SQL |

### 8.2.3 Update Fehlerquellen

Die Fehler werden täglich mittels Cronjob-getimten Shellskriptaufrufen von den entfernten Quellen abgerufen. Da im Gegensatz zur Version BA_HunzikerOderbolz neu mehrere Fehlerquellen abgerufen werden können und komplexere Konsolidierungsvorgänge nötig wurden (siehe**Fehler! Verweisquelle konnte nicht gefunden werden.** Kapitel 8.2.2 Übersicht Fehlerquellen), wurde ein Skript geschrieben, das den Updatevorgang koordiniert. Dieses findet sich unter:

```
/server/database/update_db.sh
```

Der entsprechende Cronjob Aufruf lautet wie folgt:

```
$ update_db.sh -o osm -n osm_bugs
```

**Tabelle 26 Parameter Update-Shellskript**

| Parameter | Bedeutung |
|---|---|
| **-o osm** | Datenbankbenutzer welcher das Schema besitzen soll (Owner) |
| **-n osm_bugs** | Name der Datenbank |

Damit werden die Fehler aus allen Datenquellen im Schema all_errors in der Tabelle errors konsolidiert und die nötigen Indexe darüber aufgebaut.

[14] http://keepright.ipax.at

[15] http://giswiki.hsr.ch/EOSMDBOne

### 8.2.4 Übersicht Fehlertypen

Im Rahmen dieser Arbeit wurde ein neuer Fehlertyp für die neue Fehlerquelle EOSMDBOne (siehe Kapitel 8.2.2 Übersicht Fehlerquellen) erstellt. Dieser stellt zugleich den Prototyp für weitere Fehlertypen dieser Fehlerquelle dar (siehe Kapitel 8.2.5 Neuer Fehlertyp hinzufügen).

**Tabelle 27 Übersicht Fehlertypen**

| type | error_ type_id | Quelle | Beschreibung |
|---|---|---|---|
| **way_wo_tags** | 71 | Keepright | Typ des Wegs unbekannt |
| **motorway_ref** | 90 | Keepright | Strasse ohne Namen |
| **religion** | 100 | Keepright | Kultstätte/Kirche ohne Religion |
| **poi_name** | 110 | Keepright | Objekt ohne Namen |
| **missing_maxspeed** | 300 | Keepright | Fehlendes Tempolimit |
| **missing_track_type** | 390 | Keepright | Typ des Wegs unbekannt |
| **language_unknown** | 360 | Keepright | Sprache des Namens unbekannt |
| **missing_cuisine** | 1001 | EOSMDBOne | Restaurant ohne definierten Küchentyp |

### 8.2.5 Neuer Fehlertyp hinzufügen

Für das tägliche Synchronisieren der Fehler mit den externen Fehlerquellen und die Darstellung der Missionen im Client sind viele verschiedene Instanzen der Kort-Applikation zuständig. Um einen neuen Fehlertyp hinzuzufügen, müssen zurzeit Anpassungen von Hand an all diesen beteiligten Instanzen durchgeführt werden. In Zukunft sollte dies mittels eines entsprechenden Admin-Interfaces möglich sein.

**Neuer Keepright-Fehlertyp hinzufügen:**

*Schritt: Fehleridentifikatorstring definieren:*

Instanzen des neuen Fehlertyps müssen im Keepright-Resultateset (siehe Keepright 'Table layout'[16]) eindeutig identifiziert werden können (String-Match). Eine Zeile des Keepright-Resultatesets wird in die Datenbank geschrieben, falls der Fehlerindikatorstring in der entsprechenden Zeile vorkommt.

[16] http://keepright.ipax.at/interfacing.php

Bis anhin wurden nur Fehler ausgewählt, die sich eindeutig über das Feld error_name identifizieren lassen.

Schritt: Fehleridentifikator auf Datenbankserver eintragen:

Die Liste der Fehlerindikatoren muss um den neuen Fehleridentifikatorstring erweitert werden (neue Zeile in txt-Dokument):

```
/server/database/whitelist_errors.txt
```

Beim nächtlichen Updateprozess wird bei einem Matching die entsprechende Zeile in die Tabelle errors des Schemas all_errors geschrieben.

Schritt: Gemeinsame Schritte für alle Fehlertypen:

Siehe Abschnitt weiter unten ' Gemeinsame Schritte für alle Fehlertypen'

**Neuer EOSMDBOne-Fehlertyp hinzufügen:**

Schritt: Schema und error_type_id definieren

Für neue Fehlertypen der Fehlerquelle EOSMDBOne muss das Feld schema und error_type_id selber bestimmt werden. Dabei kann ein beliebiger Wert aus dem entsprechenden Intervall (siehe Kapitel 8.2.2 Übersicht Fehlerquellen) genommen werden, der nicht bereits durch einen anderen Fehlertyp belegt ist.

Schritt: Fehlerquery definieren:

Für Fehler aus der EOSMDBOne wird für jeden Fehlertyp eine eigene Query fomuliert, die mittels dblink-Extension Daten direkt vom EOSMDBOne-Server zieht.

Dadurch können über die WHERE-Clause OSM-Attribute nach dem booleschen Prinzip beliebig zu einem Fehlertyp kombiniert werden.

Der Aufbau einer solchen Query ist weitestgehend immer gleich. Daher kann jeweils eine bestehende Query als Vorlage für eine neue Fehlerquery genommen werden. Die Queries befinden sich im Verzeichnis:

```
 /server/database/osm_errors/queries/
```

Nachfolgend die exemplarische Query für den Fehlertyp 'missing_cuisine'. Für eine neue Query müssen lediglich die Werte 'schema' und 'error_type_id' analog zum vorangegangenen Schritt geändert werden, sowie die WHERE-Clause:

```
INSERT INTO osm_errors.errors (error_id, schema, error_type_id,
error_name, object_id, object_type, msgid, lat, lon, geom, txt1)

SELECT osmq.osm_id, 1001, 1001, 'missing cuisine', osmq.osm_id,
osmq.osm_type, 'Error from EOSMDBOne', osmq.lat, osmq.lon,
osmq.geom, osmq.txt1
```

```
FROM osm_errors.dblink('dbname=gis_db port=8080 host=152.96.80.44
user=readonly',

     'SELECT CASE gtype='pt' WHEN TRUE THEN 'node' ELSE 'way' END AS
osm_type, ceiling(ST_Y(ST_Transform(ST_Centroid(way),4326))::numeric
* 10000000) AS lat,
ceiling(ST_X(ST_Transform(ST_Centroid(way),4326))::numeric *
10000000) AS lon, ST_Transform(ST_Centroid(way),4326) AS geom,
osm_id, name AS txt1

                FROM osm_poi

                WHERE

                osm_id>0

                AND tags @> hstore('amenity','restaurant')

                AND (hstore("tags")->'cuisine') IS NULL

                AND (hstore("tags")->'name') IS NOT NULL

                LIMIT 10000') AS osmq(osm_type
keepright.osm_type,lat integer,lon integer,geom
public.geometry(Point,4326), osm_id int8, txt1 text)
```

Schritt: Neue Fehlerquery zum EOSMDBOne-Update-Shellskript hinzufügen

Die im vorangegangen Schritt neu erstellte Fehlerquery muss noch dem EOSMDBOne-Update-Shellskript hinzugefügt werde. Dabei wird das File

```
/server/database/setup_osm_errors_db.sh
```

unter # Load osm_errors data um den Query-SQL-Aufruf ergänzt. Am Beispiel der missing_cuisine-Fehlerquery lautet der Eintrag wie folgt:

```
psql -d $DB_NAME -f
$DIR/osm_errors/queries/osm_errors_missingcuisine.sql
```

**Schritt: Gemeinsame Schritte für alle Fehlertypen:**

Schritt: Neuer Eintrag in kort.error_type

In der Tabelle kort.error_type wird ein "blanker" Fehler um die spieltypischen Attribute wie z.B. die Anzahl Koins für diesen Fehlertyp erweitert. Auch werden dort die i18n-Keys für die verschiedenen Texte der Userinteraktionen (clientseitig) für einen Fehlertyp hinterlegt. Die Tabelle kort.error_type muss um einen neuen Eintrag ergänzt werden, so dass die Zuordnung zwischen all_errors.erros und kort.error_type über die error_type_id gemacht werden kann.

**Tabelle 28 Feldnamen von kort.error_type**

| Feldname | Feldtyp | Beispielwerte für den Error-Typ 'Objekt ohne Namen' |
|---|---|---|
| error_type_id | integer | 110 |
| type | character varying(20) | poi_name |
| description | character varying(255) | error_type.title.poi_wo_name |
| view_type | character varying(50) | text |
| answer_placeholder | character varying(100) | error_type.placeholder.name |
| vote_question | character varying(255) | vote.question.name |
| vote_koin_count | integer | 5 |
| fix_koin_count | integer | 15 |
| required_votes | integer | 3 |
| osm_tag | character varying(100) | name |
| bug_question | character varying(255) | bug.question.poi |

**Schritt: Keys in den KortDb-i18n-Files eintragen:**

In den i18n-Property-Files müssen die in kort.error_type neu definierten Keys eingetragen werden. Am Beispiel des Error-Typ's 'Objekt ohne Namen' und der Sprache Englisch (KortDb_en.props) betrifft das die folgenden Zeilen:

```
error_type.title.poi_wo_name=Object without a name

error_type.placeholder.name=Name

vote.question.name=Is this the name of this $1?

bug.question.poi=What's this $1 called?
```

Schritt: Neue Icons erstellen:

Abschliessend müssen Icons für alle States des neuen Fehlertyps in folgendem Verzeichnis erstellt werden:

```
/resources/images/marker_icons/
```

Zum Zeitpunkt dieser Arbeit gibt es vier verschiedene States pro Mission und ebenfalls vier verschiedene States für die Validierung. Zudem müssen die Icons jeweils in zwei verschiedenen Auflösungen zur Verfügung stehen. Dies ergibt eine totale Anzahl von 16 Icons pro Fehlertyp (siehe $$$ Iconkonzept). Die folgenden Icons müssen pro Fehlertyp zur Verfügung stehen, wobei die Variable $type mit dem entsprechenden Eintrag des Feldes 'type' aus kort.error_type ersetzt werden muss:

```
$type_mission.png
$type_mission@2x.png
$type_missioninactive.png
$type_missioninactive@2x.png
$type_missionpromotion.png
$type_missionpromotion@2x.png
$type_missionpromotioninactive.png
$type_missionpromotioninactive@2x.png
$type_validation.png
$type_validation@2x.png
$type_validationinactive.png
$type_validationinactive@2x.png
$type_validationpromotion.png
$type_validationpromotion@2x.png
$type_validationpromotioninactive.png
$type_validationpromotioninactive@2x.png
```

Als Template für die Icons stehen die folgenden Photoshop-Dokumente zur Verfügung:

```
_DESIGN/marker_icon_template.psd
_DESIGN/marker_icon_template@X2.psd
```

## 8.3 Architektur

[Komplett aus BA_HunzikerOderbolz S.46ff]

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 Fehlerquellen Fremdsysteme, bei welchen die Schnittstellen gegeben waren. Unsere eigenen Server haben wir entsprechend angepasst und auch via REST zugänglich gemacht.

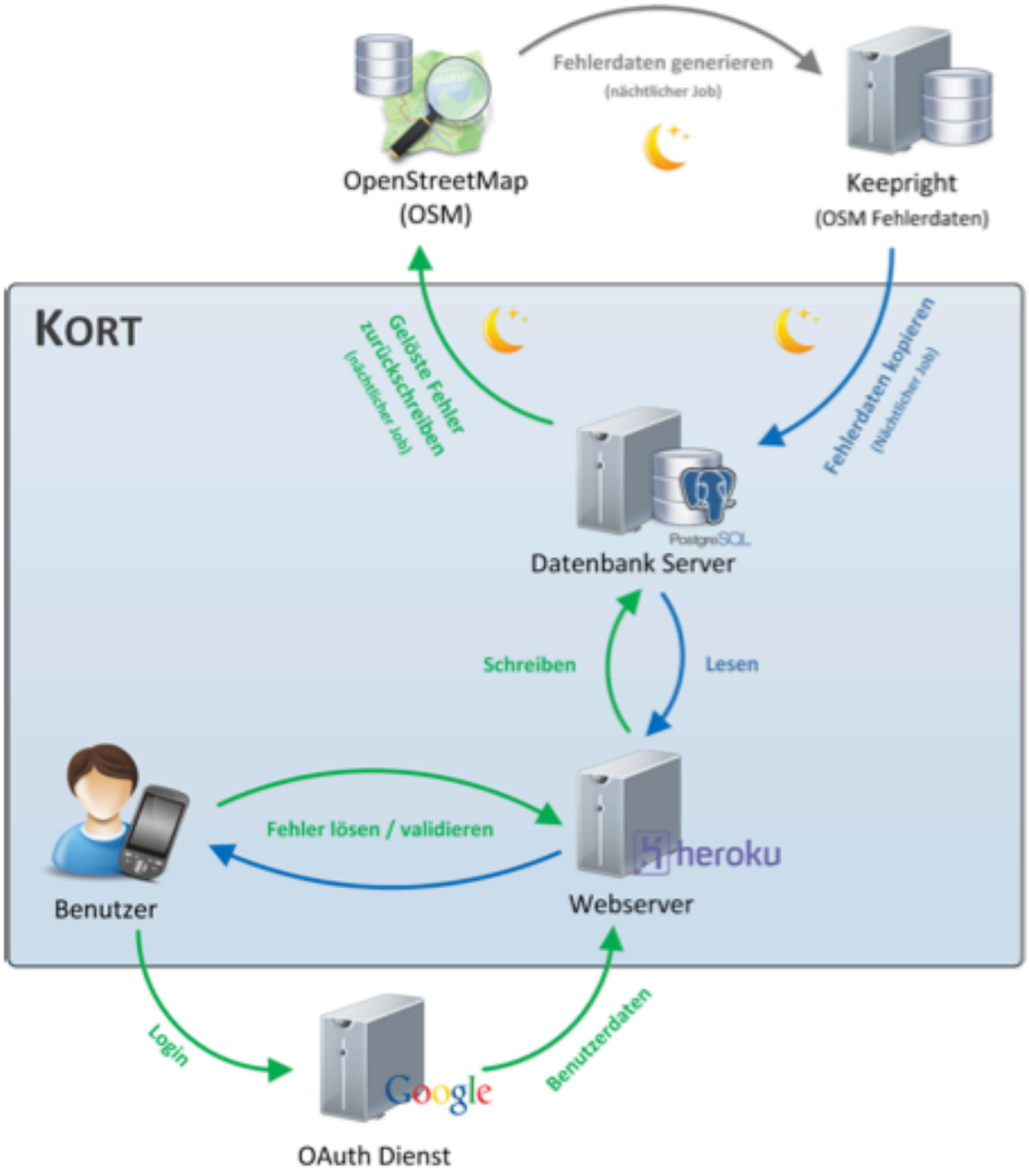



**Abbildung 19 Übersicht des Gesamtsystems**

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[13] 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.

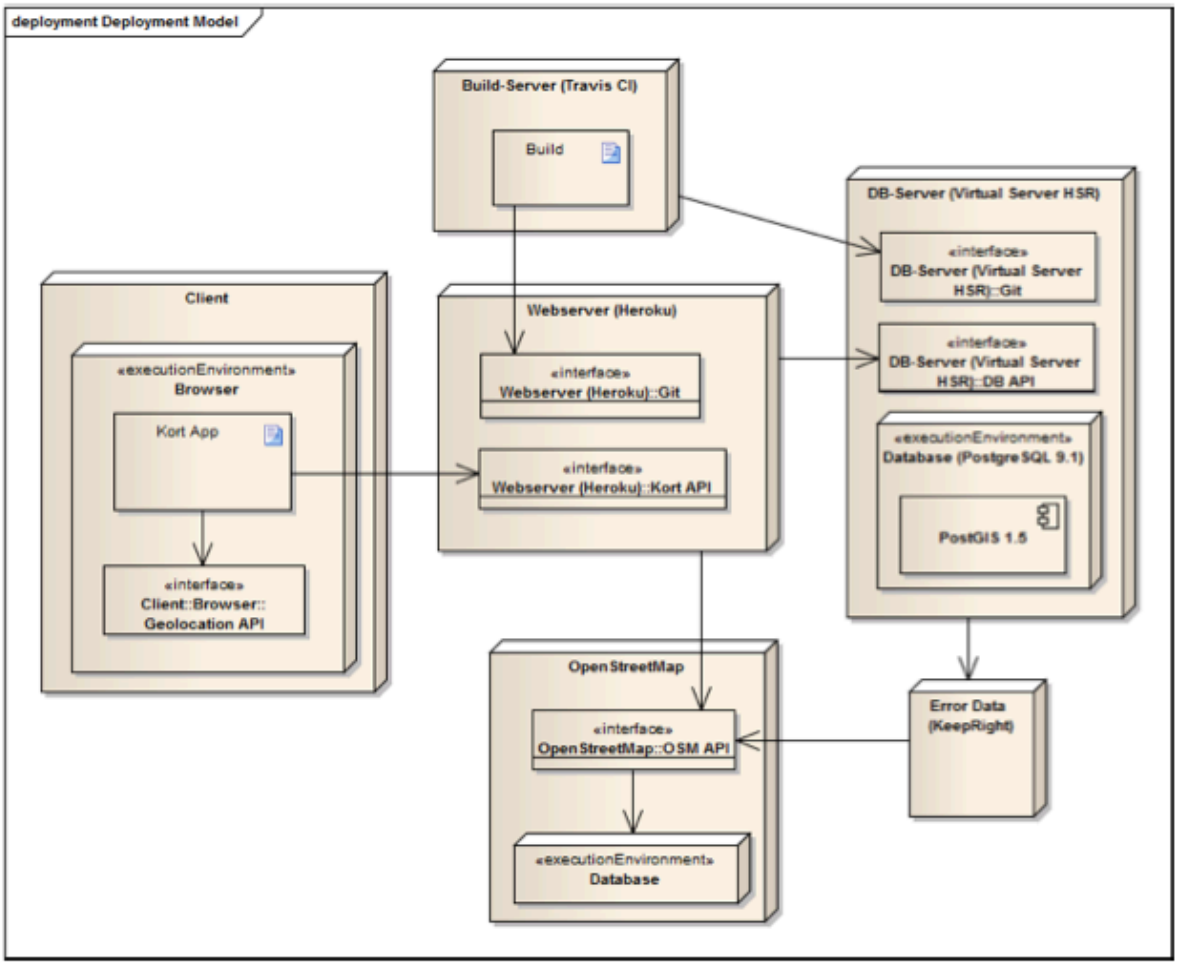



**Abbildung 20 Deployment-Diagramm**

Anmerkung Kort Reloaded: Error Data noch um zusätzliche Fehlerquelle (EOSMDBOne) ergänzt.

## 8.4 Infrastruktur

[Komplett aus BA_HunzikerOderbolz S.51ff]

### 8.4.1 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.

**Tabelle 29 Datenbankserver**

| | |
|---|---|
| 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 Webserver mit PHP 5.4 |
| Pfade | Repository: /home/odi/kort |

### 8.4.2 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 Webservice 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 Webservices genauer beschrieben.

### 8.4.3 Webserver (Heroku)

Bei Heroku[17] handelt es sich um einen kostenlosen Dienst, welcher eine Deploymentumgebung für verschiedenste Plattformen anbietet. Der Dienst hat eine Schnittstelle, über welche sich automatisiert Applikationen erstellen lassen. Die Datenübertragung läuft dann über Git.

**Tabelle 30 Server bei Heroku**

| | |
|---|---|
| Art des Servers | Server in der Cloud |
| Betriebssystem | Ubuntu |
| Zugriff | Daten via Git, Befehle via Kommandozeilen-API (Heroku- Toolbelt) |
| Installierte Software | Apache, Kort Web-App |

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).

### 8.4.4 Deployment

Am Deployment der Applikation sind mehrere Systeme beteiligt. Alle Änderungen werden von den Entwicklern via Git zu GitHub[18] 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[19] 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.

[17] http://www.heroku.com/
[18] http://github.com
[19] http://travis-ci.org

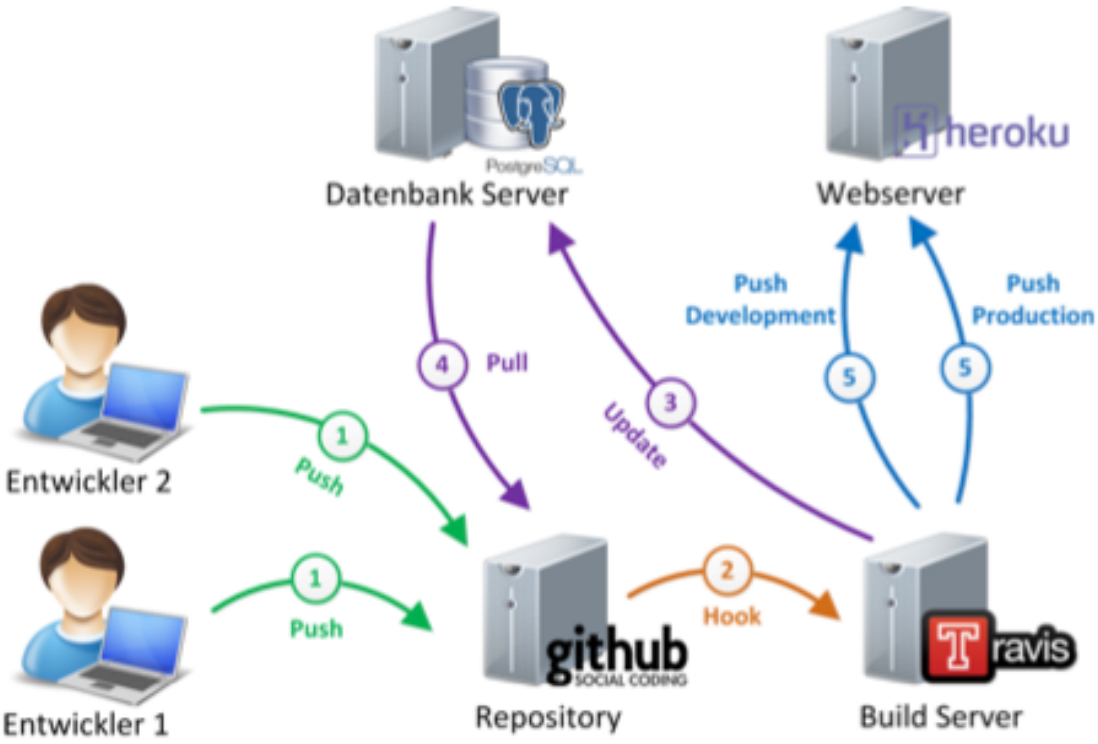



Abbildung 21 Entwicklungsprozess von Kort

### 8.4.5 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.

Tabelle 31 Build-Matrix von Travis CI

| | 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 |

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.

### 8.4.6 Konfiguration über .travis.yml

Die .travis.yml Datei ist die Konfigurationsdatei von Travis CI. Darin wird die Build-Matrix festgelegt sowie die 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).

Build-Matrix

Die Build-Matrix bestimmt, welche Builds mit welchen Parametern ausgeführt werden. Da- bei wird gemäss definierten Sprachversionen (Zeilen 3-5 im Code-Ausschnitt) 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 werden[20]. 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 Webservices) vor fremden Zugriffen schützt

[20] http://about.travis-ci.org/docs/user/build-configuration/#Secure-environment-variables

```
language: php

php:
  - 5.3
  - 5.4

env:
  global:
    - CI_HOME=`pwd`/odi86/kort
    - DEPLOY="true"
    - BUILD_DIR=`pwd`/build_heroku
    - secure: "EIn+Rm7OxX8OKygBRBCaSTOqFGcBMWu5kHdKqSxOmHRJYjxuMOJpKV+
       rnyep\n+RsyFDx2Z9yKlqRRS4cpZh7M6wwC63EV46+7
       aWtzzTjnbMZfVzLQA9EmaEU4\
       nYMsKGtpQk2mhvaNKd3UbEpDlOZq74NnAYOzipx0l02UymcFnZEc="
    - secure: "cOmMBP4UyklRCOnfeTZsX/NV4GhMBT+
       AntUmlWxXS5Rj2yrNcmNc7320gNm5\nCLSVRYyk7/8feyUEMznWrUn/62
       htZp0tEBAWtXg86dgIZgH4HPy9l2pKuSsH\
       nxZTHgjUJI7JOuyLG4ID9D5maVLE35UWag/NEtcRVy5QXLZOrs0M="
  matrix:
    - TARGET_ENV="dev"
    - TARGET_ENV="prod"

before_script:
  - gem install sass
  - gem install compass
  - gem install jsduck
  - wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh >/dev/
     null 2>&1
  - sudo npm install -g grunt >/dev/null 2>&1
  - pear install PHP_CodeSniffer && phpenv rehash
  - sudo pear channel-discover pear.survivethedeepend.com
  - sudo pear channel-discover hamcrest.googlecode.com/svn/pear
  - sudo pear channel-discover pear.phpdoc.org
  - sudo pear install --alldeps deepend/Mockery
  - sudo pear install phpdoc/phpDocumentor-alpha && phpenv rehash
  - sudo apt-get install graphviz
  - export PATH="$PATH:$CI_HOME:/usr/local/heroku/bin"
  - curl http://kort.rdmr.ch/webservices/update/git
  - mv $CI_HOME/server/php/Webservice/Database/DbConfig.example.php $
     CI_HOME/server/php/Webservice/Database/DbConfig.php

script: ant -f build_kort.xml build

after_script: bash $CI_HOME/server/heroku/heroku.sh
```

**Abbildung 22 Codeausschnitt aus Travis CI Konfigurationsdatei .travis.yml**

# III Projektmanagement

## 9. Sprints

Die Aufgabenstellung unserer Studienarbeit ist gut in kleinere Aufgaben einzuteilen. Somit war uns sofort klar, dass wir die agile Projektmethodik Scrum anwenden werden und aus den einzelnen Aufgaben die Sprints hervorgehen.

### 9.1 Sprint 1

21. Februar bis 8. März

#### 9.1.1 Ziele

Installation des bestehenden Projekts für eine funktionierende Programmierumgebung.

Meeting mit dem bisherigen Kort Entwicklungsteam mit Projektübergabe

#### 9.1.2 Hauptaufgaben / Fokussierung im Sprint

- Präzisierung der Aufgabenstellung
- Festlegung der Projektmanagementmethode
- Ausarbeitung Projektplan
- Projektübernahme BA-Team
- GitHub Projektkonto einrichten, forken von kort/kort
- Einrichtung der Entwicklungsumgebung, Installation des geforkten Kort-Projektes lokal und auf dem virtuellen Server der HSR. Studium des bestehenden Projektcodes.
- Einarbeitung in die verwendeten Frameworks wie Sencha Touch[21] und Leaflet[22].

#### 9.1.3 Termine

21.02.2013 Kickoff-Meeting.

Zu diesem Zeitpunkt war die Aufgabenstellung für unsere SA noch nicht genau definiert. Es waren aber viele Ideen vorhanden wie die Kort-App erweitert werden könnte. Neue Ideen wie die Kort-App verbessert werden könnte, waren und sind auch auf GitHub[23] zu finden.

Aus dieser Sammlung von Verbesserungsideen werden wir die genauen obligatorischen und optionalen Aufgaben unserer SA bei einer nächsten Sitzung spezifizieren.

---

[21] http://docs.sencha.com/touch/2.1.0/
[22] http://leafletjs.com/
[23] https://github.com/kort/kort/issues?page=1&state=open

01.03.2013 Projektmeeting mit BA-Team

Zu diesem Zeitpunkt trafen wir bereits auf erste Probleme mit der Installation des Projekts. Da waren wir froh, dass mit Jürg Hunziker und Stefan Oderbolz ein Termin geplant war für die Projektübergabe. Stefan Oderbolz bestätigte uns, dass zurzeit kein Setup des Projekts möglich ist und versprach uns, die entsprechenden Installationsfiles zu korrigieren und anzupassen. Ein paar Tage später war es schliesslich möglich das Projekt zu installieren und somit konnten wir auch den virtuellen Server aufsetzen.

### 9.1.4 Erledigte Arbeiten

Forken des kort/kort https://github.com/kort/kort Projekts auf GitHub in das Projekt kort/faustos https://github.com/faustos/kort?source=cc

Nachdem die Aufgabenstellung genau spezifiziert wurde, konnten wir den Projektplans erstellen. siehe Kapitel 7.4

Da wir beide bis zu diesem Zeitpunkt noch nie mit Sencha Touch gearbeitet hatten, galt es die Dokumentation zu studieren und uns in das Framework einzuarbeiten. Sencha Touch hat eine klar strukturierte Dokumentation und gute Tutorials auch für Beginner, so konnten wir uns bald vorstellen, was bei dieser Arbeit etwa auf uns zukommen wird.

Erwerben der Entwicklungsumgebung PhpStorm 6.0 bzw. WebStorm 6.0.1 von JetBrains. Dies ist ein gutes Tool für die Entwicklung mit Sencha Touch da es Autocomplete für JavaScript, HTML5, SASS und CSS anbietet. Git wird ebenfalls unterstützt.

### 9.1.5 Probleme

Am Anfang der Arbeit unterschätzten wir den Aufwand für eine Übernahme eines bereits existierenden Projekts. Dazu kam, dass die Installationsdateien anfangs nicht korrekt waren. So benötigten wir viel mehr Zeit für das Aufsetzen der Entwicklungsumgebung als wir geplant hatten.

## 9.2 Sprint 2

8. März bis 22. März

Kort-Aktionen sollen die Kort-App spannender machen.

## 9.3 Sprint 3

22. März bis 19. April

### 9.3.1 Ziele

Wenn die Kort-App geladen wird, sieht der User sofort alle Aufträge und Überprüfungen. Er kann anhand der Markierung der Icons erkennen worum es sich an der betreffenden Stelle handelt. Es gibt keinen Überprüfungs-Tab mehr, der bis anhin extra ausgewählt werden musste um sich Überprüfungen anzuzeigen. Wenn der User will, kann er aber

einzelne Layer ausblenden, er kann sich nur Aufträge oder nur Überprüfungen anzeigen lassen. Wie wir dies implementierten ist im Kapitel 7 erklärt.

## 9.4 Sprint 4

19. April bis 3. Mai

### 9.4.1 Ziele

Anzeige von News ausgehend von einem RSS/Atom-Feed, z.B. als HTML5-Notification. Damit werden zum Beispiel die Kort-Aktionen aus Sprint 2 angekündigt. Wie wir dieses Ziel erreichten ist in Kapitel 7.9 zu finden.

Refactoring

Sneaky Peak Funktion implementieren

## 9.5 Sprint 5

3. Mai bis 17. Mai

### 9.5.1 Ziele

Anzeige der Highscoreliste so dass der User seinen Rang und seine direkten Vorgänger und Nachfolger sieht.

Bestehende Unit Tests fixen und erweitern, neue Unit Tests hinzufügen.

Nach Projektplanänderung vom 16. Mai (siehe Kapitel 2.1 “Änderung der Aufgabenstellung”):

Bereitstellung des PHP-Servers

### 9.5.2 Sprintplan Änderung

Am Anfang dieses Sprints wurde bereits klar, dass der Server von Michael Wolski nicht rechtzeitig bereit sein wird und wir die Webservices selber schreiben müssen, sofern wir wollen dass unsere Arbeit auch livegeschaltet werden kann. Somit wurde zu diesem Zeitpunkt eine Änderung der Planung nötig. Mit der Anpassung des Servers wurde ein grosser zusätzlicher Task in das Projekt eingefügt, somit waren während diesem Sprint viele zusätzliche Arbeitsstunden nötig.

## 9.6 Sprint 6

17. Mai bis 31. Mai

### 9.6.1 Ziele

Liveschaltung

Dokumentation fertig stellen

# 10. Anforderungsspezifikation

- Zeitlich begrenzte Aktionen/Promotions (neues Attribut: "Teil von Aktion X...") durchführen und Spielemechanik parametrisieren ("Wer erhält für was wie viel..."; Fehlertypen servergesteuert anbieten).
- Mission und Überprüfung zusammenlegen (bei Überprüfungen direkt Kartenausschnitt anzeigen).
- Anzeige von News ausgehend von einem RSS/Atom-Feed, z.B. als HTML5-Notification.
- Optional: Highscore: Anzeige der Nachbarn in der Rangliste.
- Optional: Erweiterungen wie z.B. Anzeige von Missionen ausserhalb des eigenen Rayons oder Bugs beheben.
- Erweiterung der Server-Komponente (PHP) um Aktionen/Kampagnen und (optional) zusätzliche Fehlertypen (Domain-Logik, Datenbank, Webservices) (Nachtrag Arbeitswoche 11)

# 11. Rahmenbedingungen

- Integration mit der Kort-Server-Applikation, d.h. lose Zusammenarbeit mit Michael Wolski, (Student MSE bis Arbeitswoche 11, dann selbständig) vor allem über gemeinsame Webservices.

  - Eingesetzte Client-Technologien sind HTML5.
  - Die Studenten entscheiden sich nach Rücksprache mit dem Betreuer für eine SW-Entwicklungsmethodik. Die Meilensteine werden mit dem Betreuer und allfälligen Projektpartnern vereinbart.
  - Code, Kommentare und Versionsverwaltung sind in Englisch. Der Code ist grundsätzlich mehrsprachig. Alles andere ist Deutsch.
  - Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs beziehungsweise der HSR.

# 12. Sitzungsprotokolle

## 12.1Kickoff-Meeting

21. Februar 2013, 14:30 Uhr im IFS

Die Aufgabenstellung für die SA ist noch nicht genau definiert. Stefan Keller zeigt uns einige Dinge die an der Kort App verbessert werden könnten.

### 12.1.1Inputs von Stefan Keller:

- Missionen einfügen, bei welchen es zum Beispiel für eine bestimmte Zeit doppelte Anzahl an Koins zu gewinnen gibt. Diese brauchen ein spezielles Look & Feel.
- Neue Kategorien von Aufgaben hinzufügen.

- Die Rubriken "Aufträge" und "Prüfen" zusammenlegen zu einem Tab.
- Weitere Inputs sowie ein Ausblick sind auf dem Feedback vom GitHub zu finden: https://github.com/kort/kort/issues?page=1&state=open
- Wir werden nur clientseitig arbeiten, da der Server vom Masterstudenten Michael Wolski bearbeitet wird. Eine seiner Aufgabe wird die Einführung von Tages- und Wochenhighscores sein. Die Spezifikationen für unsere Erweiterungen können wir ihm jeweils mitteilen.

### 12.1.2Vorgehen bis zum nächsten Meilenstein/Sprint:

- Einarbeitung in Sencha Touch
- Server aufsetzen
- GitHub Projektkonto einrichten (kort/kort)
- Projektvorgehen bestimmen
- Die Aufgaben für die SA werden im Wiki von Stefan Keller gesammelt und bei einer nächsten Sitzung spezifiziert. Diese Spezifikationen mit der Unterteilung in obligatorisch und optional werden wir später unterschreiben müssen.

## 12.2Projektmeeting

28. Februar 2013, 15 Uhr im IFS

Features Ideen:

- Highscoreeinstellungen: lokal - global, ewige - aktuelle
- Anzeige des Rankingverlaufs
- Wenn es eine spezielle Punkteaktion gibt, wird dies mittels einer Push-Nachricht angezeigt.
- Es werden auch Aufträge angezeigt, die sich ausserhalb des definierten Radius des Userstandorts befinden. Diese kann der User zwar anschauen, aber nicht ausführen, bevor er sich nicht in dessen Nähe befindet.

Verschiedenes:

- Projektvorgehen = Scrum. Wir werden Sprints von jeweils zwei Wochen definieren.
- Vor den Sitzungen schicken wir an Stefan Keller eine Termineinladung. Dort definieren wir die Traktanden und geben einen Verweis auf das Redmine, wo allfällige Probleme ersichtlich sind.

## 12.3Projektmeeting mit BA-Team

1. März 2013, 14 Uhr am Kaffeetisch des IFS

Anwesend: Stefan Keller, Stefan Oderbolz, Jürg Hunziker, Michael Wolski, Carmelo Schumacher, Annrita Egli

### 12.3.1Übergabe des Projekts

Zurzeit ist kein Setup des Projekts möglich.

Stefan Oderbolz wird die nötigen Files für das Setup bis am Montag, 11. März anpassen.

Es wird beschlossen, dass Stefan Oderbolz und Jürg Hunziker während des Semesters keine Änderungen am Code vornehmen werden, aber sie werden weiterhin bei den Communities zu der Kort App aktiv sein.

## 12.4Projektmeeting

7. März 2013, 15:15 Uhr im IFS

Festlegen der Aufgabenstellung für die Studienarbeit

http://wiki.hsr.ch/StefanKeller/SAEgliSchumacherAufgabenstellung

## 12.5Projektmeeting mit Michael Wolski

7. März 2013 in der Cafeteria

### 12.5.1Beschlüsse:

Kampagne:

Betrifft REST-Schnittstelle "bug"

JSON muss um 2 Felder erweitert werden:

- campaignID
- campaignExtraCoins

## 12.6Projektmeeting

14. März 2013 im IFS

### 12.6.1Inputs von SKE

- Wenn keine Prüfungen angezeigt werden können, kommt die Meldung "Bravo, du hast alle Überprüfungen abgeschlossen". Dies ist ein Bug. Da im Laufe des Projekts die Missionen und Prüfungen zusammengelegt werden, wird diese Anzeige sowieso nicht mehr erscheinen. Dafür ist zu überlegen, was genau angezeigt werden soll, wenn beim gegenwärtigen Standort des Users weder Missionen noch Prüfungen zu erledigen sind.
- Die Aufgabenstellung von MWO wurde bereits definiert und unterschrieben und ist hier zu finden:
  http://wiki.hsr.ch/StefanKeller/PA2VPWolskiAufgabenstellung
- SKE schlägt vor, bei Oderbolz/Hunziker nachzufragen, ob sie die Icons in grösserer Auflösung besitzen. Er betont aber, momentan noch keine Zeit in grössere Icons zu investieren.

## 12.7Projektmeeting

21. März 2013 im IFS

### 12.7.1Aufgabenstellung festlegen und ausdrucken.

Übersetzungen

Das ehemalige Kort-Team hat die Sprach-Files jeweils bei Transifex.com hochgeladen, wo diese in die verschiedenen Sprachen übersetzt werden. Dort sieht man jeweils wie weit die einzelnen Sprachen übersetzt wurden. Wir entscheiden darüber, ab wann wir die Files in das Projekt einfügen werden. Die Vorgabe für unsere SA lautet, dass wir das File auf Englisch fertig stellen und danach bei Tranisfex.com hochladen. Dafür benötigen wir entsprechenden Zugang zum Kort-Projekt.

Poster

In der Aufgabenstellung steht, dass ein Poster über das Projekt zu erstellen ist. SKE betont aber, dass dies nur als Übung für die Bachelorarbeit gedacht ist und nicht ausgedruckt werden muss. Ein PDF genügt.

CDs & Präsentation

Es sind 3 CDs abzugeben. Jeweils ein Exemplar für Oliver Rehmann (fürs Archiv), eines für SKE (Betreuer) und eines für einen Gegenleser.
Es gibt keine Präsentation der SA.

## 12.8Projektmeeting

4. April 2013 im IFS

### 12.8.1Traktanden

Was wurde erreicht:

- Sprint 2 ist fast beendet, Sprint 3 begonnen.
- Layer kann ausgewählt werden um nur Missionen oder Prüfungen anzuzeigen
- Mission und Prüfungen wurden zusammengelegt

Frage:

Der Titel der Kort-Aktion wird ja angezeigt wenn man die Info zur Kort-Aktion öffnet. Im Moment ist dieser hardcodiert. Da die Kort App aber mehrere Sprachen anbietet, stellt sich die Frage

- ob wir Standardtitel für die Aktionen anbieten?
- oder die Aktionen nur bei der entsprechenden Spracheinstellung verfügbar sind?
- oder eine Aktion erst gestartet werden kann, wenn der Titel in alle verfügbaren Sprachen übersetzt wurde?

SKE erläutert, dass es nur wenige Personen gibt, die eine Kort-Aktion erstellen können. Diese Aktionsleiter steuern die Aktionen auf einem administrativen Backend. SKE schlägt vor, dass der Titel der Kort-Aktion jeweils auf Englisch angezeigt werden soll,

sofern der Titel nicht in der im Browser eingestellter Sprache existiert. In der DB bräuchte es den Eintrag "title" nicht.

Was muss gemacht werden:

- Um Sprint 2 abzuschliessen muss Server bereit sein
- Layer generischer gestalten
- Sprint 3 dokumentieren und fertig stellen

Reto Senn von bitforge Zürich hat sich anerboten, das Projekt Kort zu betreuen. Dies bedeutet, dass wir ihn ansprechen können, sollten Fragen auftauchen, insbesondere zum GUI.

## 12.9Projektmeeting

10. April 2013 im IFS

### 12.9.1Protokoll

Beschlüsse zu Kort-Aktionen und deren Internationalisierung

- Das Backend wird auf Englisch zur Verfügung gestellt.
- Kort-Aktionen haben einen einheitlichen Titel, die Sprache wird der Region angepasst.
- Bei Überschneidungen von mehreren Kort-Aktionen auf denselben Bug trifft der Webservice die Entscheidung, welche Kort-Aktion dass beim jeweiligen Bug gilt.
- Da in absehbarer Zeit nicht mehr als ein Duzend Kort-Aktionen aufs Mal aktiv sein werden, werden diese beim Starten der App geladen.

Inputs zu News (Sprint 4)

- Feed Reader beim Starten der App, wo die letzten drei News angezeigt werden, egal wie alt diese sind.
- Auf dem Webinterface werden News mittels Atom Feed angezeigt.
- Jedes Newsitem wird in einer Sprache verfasst.

Inputs zu Highscoreanzeige (Sprint 5)

- In der Kort-App werden die globalen Highscores angezeigt.
- Weitere Highscores werden im Webinterface angezeigt.
- Anzeigen wenn keine weiteren Einträge vorhanden sind.
- MWO hatte die Idee, dass man bei den Highscores nach einem User suchen kann. Usernamen sind jedoch nicht eindeutig.

## 12.10 Projektmeeting

18. April 2013 15 Uhr im IFS 15 Uhr

### 12.10.1 Traktanden

Was wurde gemacht:

- Sprint 3 von unserer Sicht her abgeschlossen.
- Beschlüsse betreffend Sprint 2 & 3 von letzter Sitzung umgesetzt.
- Sprint 4 Gedanken zur Architektur von News, Notifikation in News Tab anstelle Ladescreen.

Fragen:

- Dokumentation: In welchem Umfang kann auf die Bachelorarbeit von Hunziker/Oderbolz verwiesen werden?
- Wie sieht der Projektplan von MWO momentan aus? Wurden Anpassungen gemacht?
- Hat sich Odi betreffend Server gemeldet?

Was wird gemacht:

- RSS Feed umsetzen als Reader.

### 12.10.2 Protokoll

Zu Sprint 4:

- Spezifikation Newsanzeige: Es gibt nur einen Feed. Dieser zeigt laufenden Kort-Aktionen an mit Sprachangabe.
- Die Sprache der News gibt Ausschlag über die Region. Die Region wird im Titel angegeben.

Allgemeines:

- MWO wird den Server bis am 26. April auf Python umstellen.
- SKE und MWO haben bemerkt, dass es nur einen Fehlertypen gibt, der noch nicht implementiert ist => Missing Website
- Die Dokumentation von der SA-Kort soll eigenständig lesbar sein. Wo nötig können Abschnitte aus der Kort-Bachelorarbeit kopiert werden mit entsprechendem Verweis.

## 12.11 Projektmeeting

24. April im IFS

### 12.11.1 Traktanden

Was wurde gemacht

- Sprint 4: News Tab mit Newsliste & Detailansicht
- Der News Tab hat einen Settingsbutton wo die Sprachen ausgewählt werden können. Als Default Wert werden alle News angezeigt.
- Idee: News, in denen Promotionen angekündigt werden, haben einen Link zur Karte mit dem Mittelpunkt der Promotion. Dort sieht man inaktive Items.
- Überlegung: Wenn die News veraltet sind, werden sie vom Admin gelöscht.

Fragen

- Stand von Odi ?

- Stand von Wolski? (vor allem betreffend Highscore-Sprint)
- Mail vom Sonntag => Unklarheiten besprechen

Was muss gemacht werden

- Sprint 4 abschliessen

### 12.11.2 Protokoll

- Bei den Spracheinstellungen fehlt noch: alle auswählen / keine auswählen, Titel hinzufügen
- Link in News einfügen mit lat/lon
- Das Glossar sollte gefestigt sein, also kann mit Refactoring begonnen werden
- Odi hat sich am 18.4 zum letzten Mal gemeldet

## 12.12 Projektmeeting

2. Mai im IFS

### 12.12.1 Traktanden

Was wurde gemacht

- Umfangreiches Code-Refactoring.
- Sneaky Peak - Funktionalität eingebaut
- Karten URL Sprung
- Look & Feel von News überarbeitet

Fragen

- Es konnte verifiziert werden, dass die Datenbank tatsächlich wie von Odi vermutet das Bottleneck ist. Es wurde der Index neu aufgebaut und VACUUM durchgeführt - ohne Erfolg. Haben Sie eine Idee, woran es liegen könnte bzw. würden wir gerne das Vorgehen zur Analyse / Fehlersuche mit Ihnen besprechen. Es ist insofern komisch, dass beide Datenbanken in etwa die gleiche Datenmengen und Operationen verarbeiten müssen.
- Zugangsdaten Transifex und Heroku (!Dringend!). Ich habe Stefan Oderbolz schon vor einiger Zeit diesbezüglich angefragt und noch keine Antwort erhalten. Wir benötigen die Zugangsdaten bis morgen Freitag 11:00 Uhr. Können Sie Odi diesbezüglich kontaktieren?

Was wird gemacht

- Sprint 4 abschliessen
- Termin von CSC und MWO in Zürich am Freitagnachmittag, 3. Mai:
    - Integration der Webservices auf lokalen Rechnern
    - Systemtests und AdHoc-Behebung allfälliger Probleme/Erweiterungen
    - Commit auf GitHub

### 12.12.2 Protokoll

- Der Link mit dem Sprung auf eine bestimmte latitude/longitude auf der Map heisst "Permalink"
- SKE schlägt folgenden Link vor um den Index der DB zu analysieren: http://explain.depesz.com Damit sollten wir auch die Explains vergleichen können von unserer DB und dieser auf dem Liveserver.
- Um unabhängig von den Heroku Zugangsdaten zu sein schlägt SKE vor, einen separaten Heroku-Testserver einzurichten
- Es sind noch Unit Tests zu schreiben.

## 12.13 Projektmeeting

Mittwoch, 8. Mai, 16 Uhr im IFS

### 12.13.1 Traktanden

Was wurde gemacht

- Termin vom letzten Freitag wurde von MWO abgesagt. Vorschlag für weiteres Vorgehen:

Mail von CSC an MWO: „Ich würde folgendes vorschlagen: Wir erstellen einen neuen Branch auf faustos/kort und nennen diese python-integration. Dort pushst du deine Arbeit, sobald du fertig bist (idealerweise heute). Wir müssen uns ja sowieso noch Gedanken zum Bootstrapping machen. Evtl. hast du die Python-Libraries aufgelistet, die du benötigst und was sonst noch zu beachten ist bzw. inwiefern build_kort.xml und INSTALL.md angepasst werden muss? In Zukunft sollte die Installation ja automatisch erfolgen."

- Sequenzdiagramme erstellt für Doku
- Einarbeitung in QUnit Tests
- Codedokumentation nach Mozilla JS Guideline und JSHint
- Begonnen Aufbau Testsetup (Travis, Heroku) -> Ziel: Deployment realitätsnah testen

Fragen

- Weiteres Vorgeben/ Terminierung Webservices (MWO), Transifex (Odi), Heroku (Odi)

Was wird gemacht

- Highscore-Sprint

### 12.13.2 Protokoll

- MWO hat auch auf Mail von SKE nicht reagiert. Somit müssen wir damit rechnen, dass der Server nicht rechtzeitig bereit sein wird. Vorschlag von SKE: Wir lassen die App auf dem PHP Server und CSC wird diesen selber so erweitern, dass er an unsere Änderungen angepasst ist. CSC ist damit einverstanden und es wird beschlossen, damit frühestens am 9. Mai mittags zu beginnen. Es kann dabei alles

mit Files realisiert werden. SKE kann bei Fragen zu den Polygons bei den Promotions helfen. Die bereits mit MWO besprochenen Spezifikationen der DB-Struktur für den Server wird SKE per Mail an CSC mitteilen. Infolge dieser Mehrarbeit, werden der Highscore-Sprint und das A0 Poster gestrichen.
- Reto Senn passt der Termin vom 16. Mai nicht, wir machen vorerst keinen Termin ab aus Zeitgründen.
- Odi hat gesagt, dass er CSC zum Transifex hinzugefügt hat, dies scheint aber nicht geklappt zu haben. SKE schreibt ein weiteres Mail an Odi.

## 12.14 Projektmeeting

16. Mai im IFS

### 12.14.1 Traktanden

- Hilfe bei Erweiterung des Servers

### 12.14.2 Protokoll

Anpassung Projektdefinition

Da MWO seine Projektarbeit abgebrochen hat, wurden Anpassungen nötig was unsere Studienarbeit betrifft:

- Aufgabenstellung:
  - Veränderte Anzeige der Highscore ist neu eine optionale Aufgabe
  - Neu hinzugefügt wurde: Erweiterung der Server-Komponente (PHP) um Aktionen/Promotions und optional zusätzliche Fehlertypen (Domain-Logik, Datenbank, Webservices)

Allgemeines

- SKE stellt die Frage in den Raum, warum bei manchen Überprüfungen 3 und bei anderen 5 positive Rückmeldungen nötig sind. Und was passiert mit der Mission, wenn zu viele negative Rückmeldungen erfasst wurden?
- SKE bittet uns einen Issue zu erstellen, welcher folgenden Bug anspricht: Mit Chrome kann auf einem bestimmten Tablet die Position nur mit GPS gefunden werden. Da dies länger dauert, führt dies zu einen Timeout der App

## 12.15 Projektmeeting

22. Mai im IFS

### 12.15.1 Traktanden

- Rollout-Plan erstellen. MissionID-Type-Problem besprechen. Migrations-Skripte durchsprechen. Wer macht wann was mit Stefan und Odi koordinieren. Abhängigkeiten eruieren.
- Neue Fehlerquelle EOSMDBOne eingebunden. Neue DB-Architektur reviewen. Updateprozess reviewen.

- Cuisine-Type Answers festlegen / besprechen (Anregung: [http://wiki.openstreetmap.org/wiki/Key:cuisine] und Michelin Cuisine Types).
- Problem klären: osm-Webservice bei neuer Fehlerquelle EOSMDBOne: von gtype (table osm_poi) muss auf den osm_type geschlossen werden können. Ansatz: Falls gtype=pt => node sonst way. Erforderlich, damit osm ajax call der Form http://www.openstreetmap.org/api/0.6/node/271223885 korrekt gemacht wird (relevant für Tab Map auf Mission-Fix)
- Wieso Website als zusätzlicher Keepright-Fehlertyp ungeeignet ist.

### 12.15.2 Protokoll

- In der Dokumentation soll beschrieben sein, wie ein neuer Fehlertyp auf Kort hinzugefügt werden kann und wie man eine Kort-Aktion startet. Ebenfalls in die Dokumentation unter "Ausblick" sollen folgende Punkte aufgezählt werden:
  - Upload der Lösungen auf OSM
  - Geplant: 12. Juni Start von erster Kort-Aktion in Zusammenhang mit der Fossgiss Konferenz.
  - Verständlichere Statistiken auf der Website
- CSC bereitet mit Odi das Rollout vor, dieses findet am Dienstag, 28. Mai ab 16 Uhr statt. Vorher muss vom Server ein Backup erstellt werden, dies führt jeweils Oliver Rehmann durch und muss mit ihm abgemacht werden.
- Zurzeit stellt Keepright keine Daten mehr zur Verfügung. SKE schreibt diesbezüglich ein Mail an Keepright.
- Nach einer Kort-Aktion wäre es schön eine Highscoretabelle speziell für diese Aktion zu haben.
- Ebenfalls wünschenswert wäre, in der Zeit wo die DB auf Kort neu aufgesetzt wird dies den Usern anzuzeigen.
- Festlegen der Antworttypen vom Fehlertyp "Restaurant ohne Küchenangabe".
- Stefan Keller plant ab dem 12. Juni eine erste Kort-Aktion zu starten. An diesem Datum findet die Fossgiss Konferenz[24] statt, wo der Start dieser Aktion verkündet werden kann.

## 12.16 Projektabgabe

Freitag, 31. Mai 17 Uhr

[24] http://www.fossgis.de/konferenz/2013/

# Literaturverzeichnis

**Bachelorarbeit HSR**

[1] J Hunziker, S. Oderbolz, "Gamified Mobile App für die Verbesserung von OpenStreetMap", Dezember 2012

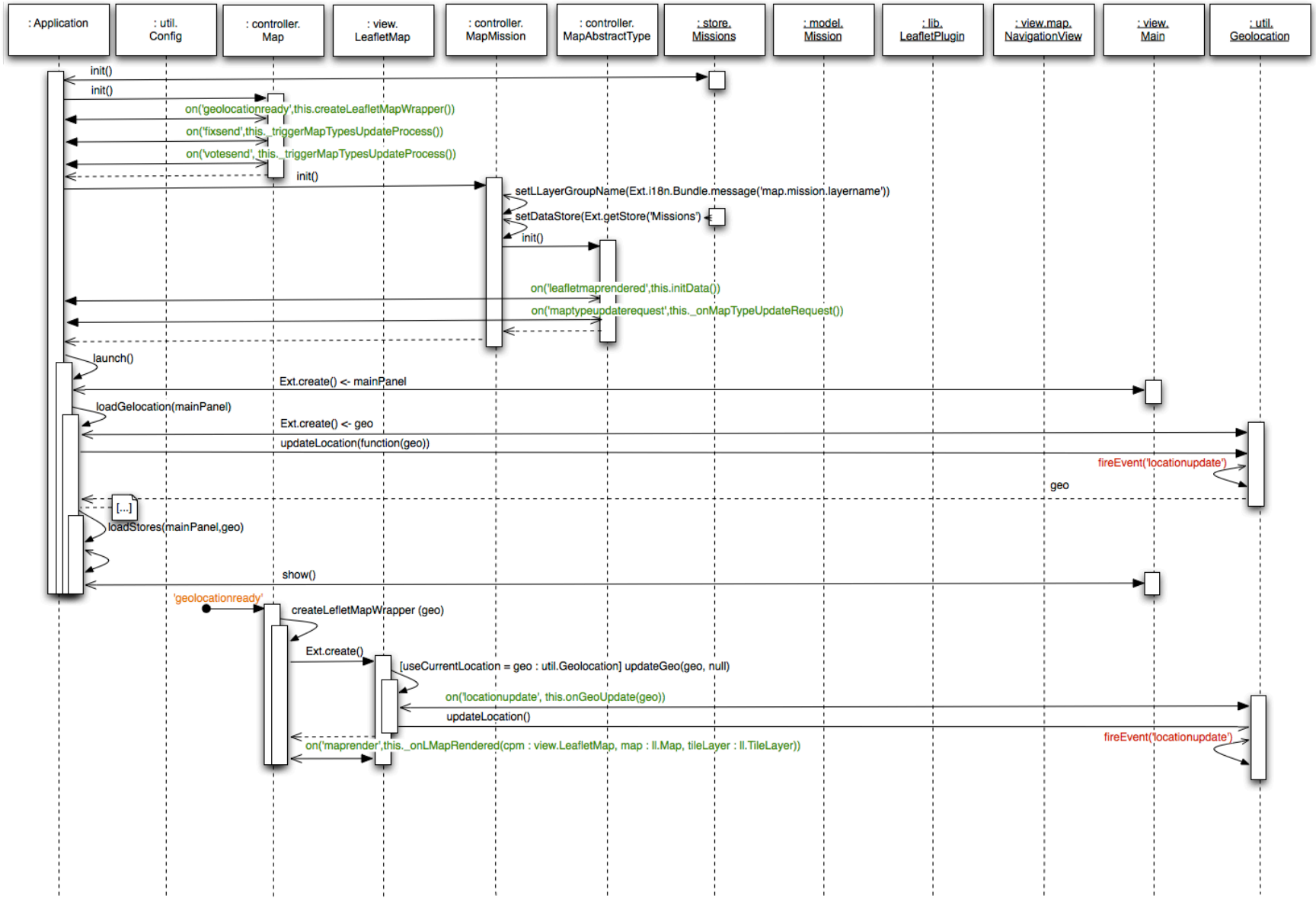
**Bücher**

[2] John E. Clark, Bryan P. Johnson, " Build web applications for Apple iOS and Google Android touchscreen devices with this first HTML5 mobile framework", Verlag: Packt Publishing, Februar 2012

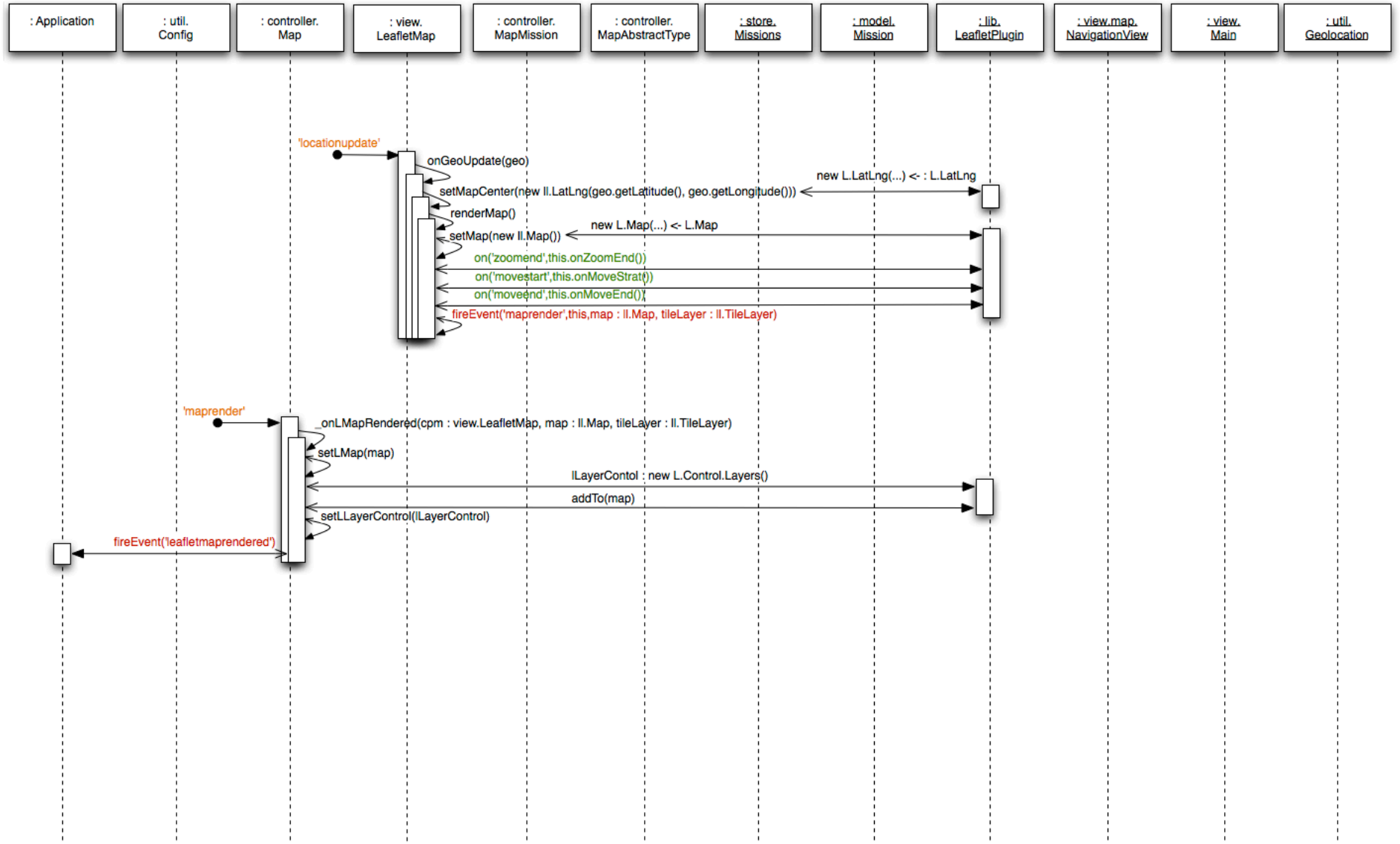
# Tabellenverzeichnis

# Abbildungsverzeichnis

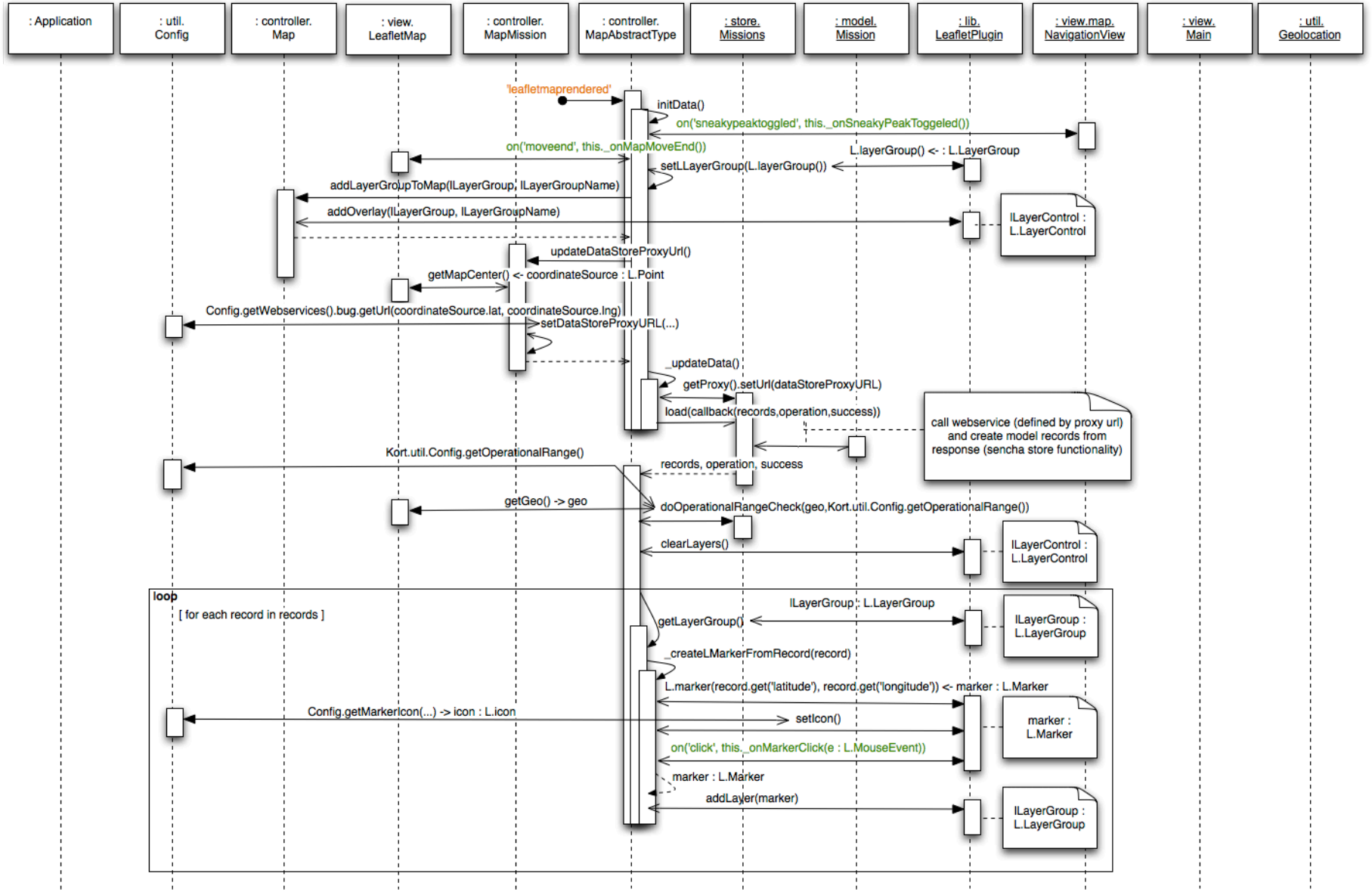
## Sequenzdiagramm Starten der Applikation I



## Sequenzdiagramm Starten der Applikation II



## Sequenzdiagramm Starten der Applikation III



# Kort Reloaded - Gamified Mobile App für die Verbesserung von OpenStreetMap

**Studienarbeit von Annrita Egli und Carmelo Schumacher**

**Abteilung Informatik, Frühjahrssemester 2013**

## Ausgangslage

"Kort" ist innovatives, location-based Web App für iPhone, Android und Tablets zur Verbesserung fehlender Daten in OpenStreetMap. Dies ist eine Fortsetzungsarbeit.

## Aufgabenstellung

Im Rahmen dieser Arbeit soll der gereifte Prototyp weiterentwickelt werden. Die Zielgruppe ist unverändert.

### Aufgaben

Verschiedene Erweiterungen und Refactorings:

- Zeitlich begrenzte Aktionen/Kampagnen (neues Attribut: "Teil von Aktion X...") durchführen und Spielemechanik parametrisieren ("Wer erhält für was wie viel..."; Fehlertypen servergesteuert anbieten).
- Mission und Überprüfung zusammenlegen (bei Überprüfungen direkt Kartenausschnitt anzeigen).
- Anzeige von News ausgehend von einem RSS/Atom-Feed, z.B. als HTML5-Notification.
- Optional: Highscore: Anzeige der Nachbarn in der Rangliste.
- Optional: Erweiterungen wie z.B. Anzeige von Missionen ausserhalb des eigenen Rayons oder Bugs beheben.
- Erweiterung der Server-Komponente (PHP) um Aktionen/Kampagnen und (optional) zusätzliche Fehlertypen (Domain-Logik, Datenbank, Webservices) (Nachtrag Arbeitswoche 11)

## Vorgaben/Rahmenbedingungen

- Integration mit der Kort-Server-Applikation, d.h. lose Zusammenarbeit mit Michael Wolski, (Student MSE bis Arbeitswoche 11, dann selbständig) v.a. über gemeinsame Webservices.
- Eingesetzte Client-Technologien sind HTML5.
- Die Studenten entscheiden sich nach Rücksprache mit dem Betreuer für eine SW-Entwicklungsmethodik. Die Meilensteine werden mit dem Betreuer und allfälligen Projektpartnern vereinbart.
- Code, Kommentare und Versionsverwaltung sind in Englisch. Der Code ist grundsätzlich mehrsprachig. Alles andere ist deutsch.
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs bzw. der HSR.

### Inhalt der Dokumentation

- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind (einheitliche Nummerierung).

- Die Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen (nicht ausschliesslich Wikipedia-Links auflisten).
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, kein Poster) gemäss Vorgaben des Studiengangs und gemäss Absprache mit dem Betreuer.

### Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare) und in Ordner (1 Exemplar „kopierfähig" in losen, gelochten Blättern).
- Alle Dokumente und Quellen der erstellten Software auf CD; CD's sauber angeschrieben (3 Ex.).

### Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Studienarbeit des Studiengangs Informatik mit besonderem Gewicht auf moderne Softwareentwicklung wie folgt:

- Projektorganisation (Gewichtung ca. 1/5)
- Bericht, Gliederung, Sprache (Gewichtung ca. 1/5)
- Inhalt inkl. Code (Gewichtung ca. 2/5)
- Gesamteindruck inkl. Kommunikation mit Industriepartner (Gewichtung ca. 1/5)

## Beteiligte

### Diplomanden/innen

Annrita Egli
Carmelo Schumacher

### Projektpartner

Fa. Bitforge (bei Bedarf).

### Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller, IFS-HSR (sfkeller@hsr.ch)

S. Keller, 16.5.2013

A. Egli, 16.5.2013

16.5.2013