HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

GEOINFO

Cloudbasiertes Geodatenmanagement mit Google Fusion Tables: Entwicklung eines Cloud-GIS Prototypen

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2012

Autoren:	**Stefan Oderbolz** (soderbol@hsr.ch) **Jürg Hunziker** (jhunzike@hsr.ch)
Betreuer:	**Prof. Stefan Keller** (sfkeller@hsr.ch)
Projektpartner:	**Marco Lehmann**, GEOINFO AG Herisau, http://www.geoinfo.ch
Experte:	**Prof. Stefan Keller** (sfkeller@hsr.ch)
Datum:	**1. Juni 2012**

Impressum und Revision

Impressum

Autoren:	Stefan Oderbolz (soderbol@hsr.ch) Jürg Hunziker (jhunzike@hsr.ch)
Dokument erstellt:	27.02.2012
Letzte Aktualisierung:	31.05.2012

Dieses Dokument wurde mit LaTeX erstellt.

Änderungsverlauf

Datum	Änderungen	Bearbeiter
27.02.2012	Dokumententwurf erstellt	Stefan Oderbolz
01.03.2012	Struktur erstellt	Jürg Hunziker
08.03.2012	Glossar hinzugefügt	Stefan Oderbolz
08.03.2012	Einleitung geschrieben	Jürg Hunziker
09.03.2012	Weitergearbeitet an Einleitung	Jürg Hunziker
19.03.2012	Beispielapplikationen dokumentiert	Jürg Hunziker
05.04.2012	Mit Dokumentation des Use Cases 1: WorldData begonnen	Jürg Hunziker
06.04.2012	Benutzerdokumentation erstellt für den Use Cases 1: WorldData	Jürg Hunziker
09.04.2012	Bildbeschriftungen zu bestehenden Grafiken hinzugefügt Klassendiagramm für Use Case 1 erstellt	Jürg Hunziker
20.04.2012	GftLib dokumentiert	Stefan Oderbolz
03.05.2012	An GftLib-Dokumentation weitergearbeitet	Stefan Oderbolz

13.05.2012	Mit UseCase 2 FixMyStreet Dokumentation begonnen	Jürg Hunziker
14.05.2012	Dokumentation Einführung in Google Fusion Tables	Stefan Oderbolz
17.05.2012	Titelseite erstellt Weitergearbeitet an Dokumentation des UseCases 2 FixMyStreet FusionTablesLayers dokumentiert	Jürg Hunziker
19.05.2012	UseCase 2 FixMyStreet dokumentiert	Jürg Hunziker
20.05.2012	Projektmanagement hinzugefügt	Stefan Oderbolz
21.05.2012	Converter-Build hinzugefügt	Stefan Oderbolz
22.05.2012	Klassendiagramm für die GftLib erstellt	Jürg Hunziker
23.05.2012	Gantt-Chart eingefügt Amazon EC2 und Jenkins Abschnitte erstellt	Stefan Oderbolz
23.05.2012	Abstract geschrieben Sprint 3 und 4 dokumentiert Impressum erstellt	Jürg Hunziker
24.05.2012	Geocodierungs-Bug beschrieben Anpassungen aus Projekt-Meeting übernommen	Jürg Hunziker
25.05.2012	PhantomJS Abschnitt erstellt Austausch mit den Google Entwicklern hinzugefügt Tabellenarten dokumentiert	Stefan Oderbolz
25.05.2012	CD Inhalt hinzugefügt Projektmonitoring geschrieben OAuth Grafiken erstellt	Jürg Hunziker
26.05.2012	Stand der Technik beschrieben Resultate der Arbeit	Stefan Oderbolz
27.05.2012	Vision/Umsetzungskonzept Schlussfolgerungen und Ausblick OAuth Abschnitt	Stefan Oderbolz
27.05.2012	Diagramme der Use Cases beschrieben FixMyStreet Masken beschrieben.	Jürg Hunziker
28.05.2012	FixMyStreet Testing beschrieben	Stefan Oderbolz
28.05.2012	Dokumentation überarbeitet	Jürg Hunziker
29.05.2012	Testabdeckung mit JSCoverage hinzugefügt	Stefan Oderbolz
31.05.2012	Dokumentation finalisiert	Stefan Oderbolz
31.05.2012	Dokumentation finalisiert	Jürg Hunziker

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

Ziel dieser Arbeit war es, das Potential der Cloud-Datenbank *Google Fusion Tables* aufzuzeigen. Dazu wurden zuerst einige Beispiele erstellt, welche die verschiedenen Features der Fusion Table verwenden. Zusätzlich wurden Anwendungsfälle im Bereich der Geo-Informationssysteme (GIS) gesucht, die sich mit Google Fusion Tables umsetzen lassen würden. Diese wurden dann als Prototypen implementiert, um den Aufwand aufzuzeigen, den es benötigt, um mit Google Fusion Tables und Google Maps eine Web-GIS-Applikation zu entwickeln.

Es wurden zwei Anwendungsfälle umgesetzt: Zum einen wurde eine Webapplikation implementiert, die es erlaubt, historische Daten von verschiedenen Ländern miteinander zu vergleichen. Das Ziel dabei war das Anzeigen von grossen Datenmengen auf einer interaktiven Webkarte.

Als zweiten Anwendungsfall wurde eine mobile Web-App entwickelt, die es dem Benutzer erlaubt, Defekte – wie zum Beispiel Strassen-Schlaglöcher – in seiner Umgebung an die zuständige Behörde zu melden. Google Fusion Table wurde dabei als Web-Datenbank zur Speicherung der gemeldeten Defekte verwendet.

Management Summary und Web-Publikation

Ausgangslage

Mit Google Fusion Tables hat Google einen Dienst lanciert, welcher einerseits eine frei zugängliche Datenbank sein soll und andererseits bei der Visualisierung der Daten helfen soll. Der Fokus liegt dabei auf geografischen Daten, welche dann beispielsweise auf einer Karte angezeigt werden. Das Ziel dieser Arbeit war es, das Potential von Google Fusion Tables abzuschätzen und dieses anhand von Use Cases aufzuzeigen. Die Use Cases wurden so gewählt, dass ein möglichst breites Spektrum abgedeckt werden konnte. Die Arbeit zeigt mögliche Szenarien auf, um Daten in Google Fusion Tables zu importieren, Daten miteinander zu kombinieren und in geeigneter Form wieder auszugeben. Als weiterer Schwerpunkte wird das Thema Kollaboration behandelt. Dabei geht es darum, dass verschiedene Personen Daten eingeben können,um diese dann als Gesamtes auszuwerten.

Als Rahmenbedingung sollten die Use Cases als Webapplikationen entwickelt werden. Dies ermöglicht einerseits eine grösstmögliche Plattformunabhängigkeit und andererseits zukunftsgerichtete Lösungen. Einer der beiden Use Cases wurde dabei als mobile Web-App gestaltet, welche auf Smartphones lauffähig ist.

Der Auftrageber dieser Arbeit ist die GEOINFO AG[1], welche massgeschneiderte GIS-Softwarelösungen für ihre Kunden entwickelt. Die GEOINFO AG ist sehr interessiert an diesem Thema, um frühzeitig zu erkennen, welche Anwendungsfälle mit Google Fusion Tables abgedeckt werden können und ob diese Dienstleitung allenfalls als Alternative zu einer konventionellen GIS-Lösung angesehen werden kann.

Bei dieser Arbeit handelt es sich um eine Studienarbeit welche innerhalb eines Semesters (14 Wochen) an der Hochschule für Technik Rapperswil (HSR) erarbeitet wurde. Insgesamt wurde die Arbeit in 4 Sprints à 3 Wochen aufgeteilt. Der erste Sprint stand ganz im Zeichen des Kennenlernens und der Einarbeitung in das API. Im zweiten Sprint wurde der erste Use Case erarbeitet und im dritten Sprint ein weiterer. Der vierte und letzte Sprint war für den Abschluss aller Arbeiten, die Dokumentation und für das Beheben von Fehlern vorgesehen.

[1] http://www.geoinfo.ch

Ergebnisse

Um die definierten Ziele zu erreichen, wurden zwei Use Cases erarbeitet, welche jeweils als Webapplikationen implementiert wurden. Der erste Use Case *WorldData* behandelt das Importieren von Daten in Google Fusion Tables. Für die Ausgabe wurde die FusionTablesLayer-Erweiterung von Google Maps verwendet. Das Ziel hierbei war es vor allem den Umgang mit grösseren Datenmengen sowie die Kombination verschiedener Datenquellen zu zeigen.

Der zweite Use Case *FixMyStreet* ist eine bekannte Idee aus Bereich der Public Participation GIS[2]. Dabei sollen Bürger einer Stadt dazu ermutigt werden, Defekte in ihrer Umgebung (Strassenlampen, Schlaglöcher, etc.) zu melden. Die zuständige Behörde kann dann die eingegangenen Meldungen abarbeiten und gegebenenfalls beheben. Um die Handhabung so einfach wie möglich zu machen, wurde dieser Use Case als mobile Web-App konzipiert, welcher plattformunabhängig auf allen gängigen Smartphones läuft.

Dank dieser zwei Use Cases und den zahlreichen Beispielanwendungen, welche während der Arbeit erstellt wurden, können wir die Stärken und Schwächen von Google Fusion Tables gut beleuchten. Dank der FusionTablesLayer lassen sich Daten sehr einfach und schnell auf einer Karte darstellen.

Google Fusion Tables als Datenbank für eine Anwendung zu verwenden ist derzeit noch sehr umständlich, gerade auch weil die Dokumentation in diesem Bereich noch unzureichend ist. Wenn jedoch bestehende Daten visualisiert werden sollen, oder jeder Benutzer seine eigenen Tabellen nutzen kann, dann zeigt der Dienst seine wahre Stärke.

Ausblick

Die Idee, strukturiert Daten in der Cloud zu speichern und auszuwerten, hat sicherlich Zukunft. Zum einen ist es für viele Anwendung ausreichend wenn sie eine solche Cloud-Datenbank verwenden, so dass nicht selbst eine Infrastruktur aufgebaut werden muss. Es ist derzeit noch gut spürbar, dass es sich bei Google Fusion Tables um ein sehr junges Projekt handelt, welches noch an einigen Stellen unvollständig ist. Schon bald wird ein neues API veröffentlicht, welches wir während dieser Arbeit bereits testen konnten. Dies ist sicher ein Schritt in die richtige Richtung.

Für grössere Anwendungen sind derzeit die Limiten nicht akzeptabel. Einerseits lassen sich nur 250MB Daten in einer Tabelle speichern und andererseits werden nur die ersten 100'000 Datensätze bei Abfragen berücksichtigt. Das SQL API bietet zwar Basisfunktionen an, jedoch kommt der Funktionsumfang noch nicht an gängige Datenbanksysteme heran.

[2] http://en.wikipedia.org/wiki/Public_participation_GIS

Inhaltsverzeichnis

Teil I.

Einleitung

1. Informationen zum Projekt

Das Ziel dieser Arbeit war das Untersuchen der Möglichkeiten, welche die Cloud-Datenbank Google Fusion Tables (GFT) bietet. Es sollen einige Prototypen für verschiedenste Anwendungsfälle im GIS-Bereich erstellt werden, welche das Potential der Datenbank aufzeigen.

1.1. Problemstellung

Die Aufgabenstellung stammt von der GEOINFO AG[1], welche massgeschneiderte GIS-Softwarelösungen für ihre Kunden entwickelt.

Für solche Unternehmen wird es nach und nach schwieriger, sich auf dem Markt zu beweisen, da bereits viele cloudbasierte GIS-Lösungen sehr günstig oder gar kostenlos erhältlich sind. Durch die zu erstellenden Prototypen soll ersichtlich gemacht werden, welche Anwendungsfälle von bestehenden proprietären GIS-Systemen bereits mit Google Fusion Tables realisierbar wären und welchen Aufwand dies darstellen würde.

1.2. Aufgabenstellung

Im Rahmen dieser Arbeit soll das Potential, aber auch die Einschränkungen von Google Fusion Tables für den Einsatzbereich eines öffentlichen Web GIS evaluiert werden. Es ist aufzuzeigen, welche der typischen Anwendungsfälle, wie sie in aktuellen Web GIS Lösungen[2] implementiert sind, auf Basis von Google Fusion Tables und Google Maps realisiert werden könnten. Eine Auswahl dieser Grundfunktionen ist anhand eines Prototypen zu implementieren.

1.3. Ziele

In der Aufgabenstellung der Arbeit wurden folgende Ziele definiert:

- Evaluation von Google Fusion Tables in Kombination mit Google Maps u.a. mit

[1] http://www.geoinfo.ch
[2] z.B. http://www.geoportal.ch oder http://www.stadtplan.stadt-zuerich.ch

Blick auf deren Funktionalität, Anwendbarkeit, Zuverlässigkeit und Performance

- Entwurf und Dokumentation einer GIS-Architektur, welche einerseits Cloud Services (am Beispiel von Fusion Tables) andererseits die Geodaten- und Serviceinfrastruktur einer Organisation integriert oder migriert.
- Analyse verschiedener Use Cases, wobei einer davon als Prototyp einer Webapplikation (voll) implementiert werden soll.
- Implementierung und Bewertung von verschiedenen cloudbasierten GIS-Prototypen unter Verwendung der Google Fusion Tables API
  - Prototyp(en) aus Use Case-Evaluation (oben). Dabei sollen v.a. auch die Geometrietypen Linestring und Polygon berücksichtigt werden.
  - Prototyp einer Datenerfassung/Verwaltung am Beispiel eines Point-of-Interest (POI) Layers mit grossen Datenmengen
- Prototyping für zukünftige (GIS-) Kollaborationsplattformen. Es soll aufgezeigt werden, wie sich bestehende Konzepte[3] verbessern lassen oder weiterentwickeln könnten.

1.4. Rahmenbedingungen

- Es gelten die Rahmenbedingungen, Vorgaben und Termine der HSR
- Die Projektabwicklung orientiert sich an einer iterativen, agilen Vorgehensweise. Als Vorgabe dient dabei Scrum, wobei bedingt durch das kleine Projektteam gewisse Vereinfachungen vorgenommen werden. Meilensteine werden bezüglich Termin und Inhalt mit dem verantwortlichen Dozenten und dem Projektpartner vereinbart.
- Die Kommunikation in der Projektgruppe, in der Dokumentation und an den Präsentationen erfolgt in Deutsch.
- Eine Prototypen-Website ist in HTML/JavaScript zu implementieren und sollte auf verschiedenen Plattformen lauffähig sein.

1.5. Vorgehen

Die Arbeit umfasst zwei Themenbereiche: Im ersten theoretischen Teil wird untersucht, inwiefern Google Fusion Tables eine Konkurrenz für bestehende proprietäre GIS-Lösungen darstellt. Im zweiten Teil sollen verschiedene Anwendungsfälle mit Google Fusion Tables

[3] z.B. http://www.mysg.ch/locations oder http://ch.tilllate.com/de/locations

als Prototypen gebaut werden, wobei einer davon als mobile Webapplikation ausprogrammiert werden soll.

1.5.1. Potential von Google Fusion Tables

Es soll aufgezeigt werden, inwiefern sich Anwendungsfälle aus bestehenden GIS-Lösungen mit Google Fusion Tables abbilden lassen.

Zu diesem Zweck sollen Use Cases erarbeitet werden, welche ein möglichst breites Spektrum abbilden können. Wichtige Themen sind dabei das Verwaltung von grösseren Datenmengen, die Verwaltung von Daten aus verschiedenen Quellen und die Kollaboration bei der Bearbeitung der Daten.

Dabei soll aufgezeigt werden, wie Daten nach Google Fusion Tables migriert werden können und anschliessend eine Weiterverarbeitung erfolgen kann. Da sich die Datenstrukturen von bestehenden Lösungen stark unterscheiden können, ist es nicht das Ziel, eine Schritt-für-Schritt Anleitung zu erstellen. Primär geht es darum, die Möglichkeiten aufzuzeigen.

Durch die theoretische und praktische Auseinandersetzung mit Google Fusion Tables soll das gegenwärtige Potential dieses Dienstes abgeschätzt werden. Diese Erkenntnis soll GIS-Lösungsprovidern einen Überblick verschaffen, inwiefern Google Fusion Tables bereits eingesetzt werden kann bzw. wo dessen Stärken und Schwächen liegen.

1.5.2. Erstellung von Prototypen

Der theoretische Teil soll dann schliesslich durch verschiedene Prototypen belegt werden. Es werden mehrere Standard-Anwendungsfälle im GIS-Bereich mittels Google Fusion Tables realisiert. Diese Prototypen sollen in Form von Webapplikationen entwickelt werden, um eine grösstmögliche Plattformunabhängigkeit zu erreichen.

Ein Anwendungsfall soll zusätzlich als vollwertige Webapplikation für Mobilgeräte ausprogrammiert werden.

1.6. Aufbau der Arbeit

Die Arbeit ist in vier Teile gegliedert. Im ersten Teil erfolgt eine Einleitung mit allgemeinen Informationen zum Projekt und dessen Umsetzung (siehe Kapitel 1 und 2). Dann folgt eine theoretische Einführung in das Thema mit einer Abschätzung des Potentials von Google Fusion Tables (siehe Kapitel 3) sowie unserer verwendeten Infrastruktur (siehe Kapitel 4). Der dritte Teil behandelt die erstellten Arbeitsresultate (siehe Kapitel 5, 6, 7 und 8) und im vierten Teil befinden sich Informationen zum Projektmanagement (siehe Kapitel 9 und 10).

Wir haben während dem Semester Sitzungen mit unserem Betreuer durchgeführt und auch unseren Auftraggeber zweimal getroffen. Dort konnten wir jeweils unseren Stand präsentieren und erhielten regelmässig Feedback zu unserem Arbeitsfortschritt.

Neben diesem Dokument umfasst diese Arbeit zahlreiche Beispielanwendungen und zwei Webapplikationen, welche im Internet verfügbar sind. Der dazugehörige Source Code ist ebenfalls frei im Internet zugänglich sowie auf der beigelegten CD zu finden.

Arbeitsresultat	**URL**
Übersichtsseite	http://gft.rdmr.ch
WorldData Use Case	http://worlddata.rdmr.ch
FixMyStreet Use Case	http://fixmystreet.rdmr.ch
Converter-Build	http://jenkins.rdmr.ch:8080/job/Convert-GIS-files/
Repository	https://github.com/odi86/GFTPrototype

Tabelle 1.1.: Übersicht aller Arbeitsresultate

2. Umsetzung

2.1. Stand der Technik

Diese Arbeit beschäftigt sich mit sogenannten Cloud-Datenbanken, mit einem Fokus auf ortsbezogene Daten. Wichtig sind dabei die verschiedenen Möglichkeiten zur Visualisierung von Daten. Dieser Abschnitt erklärt die zugehörigen Konzepte und Produkte.

2.1.1. Software-as-a-Service (SaaS) / Infrastructure-as-a-Service (IaaS)

Der Begriff *Software-as-a-Service* (SaaS) hat sich in den letzten Jahren etabliert und bezeichnet die Dienstleistung, eine Software nicht nur für einen Kunden zu entwickeln, sondern auch gleich deren Betrieb zu übernehmen. Diese gesamthafte Dienstleistung wird dann dem Kunden angeboten, so dass dieser keine eigene Infrastruktur betreiben muss. Die Cloud ist die logische Erweiterung dieses Konzepts, dabei wird der angebotene Dienst transparent auf mehreren Umgebungen und an verschiedenen Lokationen angeboten. Dies soll zum einen eine hohe Erreichbarkeit gewährleisten und zum anderen die Skalierbarkeit erleichtern.[8]

Bei Google Fusion Tables handelt es sich genau genommen um eine Infrastructure-as-a-Service (IaaS) Dienstleistung[4], da lediglich die Speicherung der Daten ausgelagert ist. Grundsätzlich würde Google aber das ganze Portfolio anbieten (Webserver, Software etc.).

2.1.2. Datenbank in der Cloud

Cloud-Datenbanken wie Google Fusion Tables schaffen das Problem der Erreichbarkeit ab. Sie sind dezentral in der Cloud gespeichert und lassen sich einfach vertikal skalieren. Dies vereinfacht den Zugang zu den Daten und ermöglicht es, diese mit anderen Personen zu teilen. Dank der bereitgestellten Infrastruktur kann sich ein Anwender der Datenbank auf das Wesentliche konzentrieren: die Daten in geeigneter Form abzulegen und auszulesen.

2.1.3. Alternative zu Google Fusion Tables

Neben der in dieser Arbeit beschriebenen Google Fusion Tables (siehe Kapitel 3) gilt es vor allem noch das Produkt CartoDB zu erwähnen. CartoDB[1] basiert auf PostGIS[2], einer PostgreSQL-Erweiterung für geografische Daten. Das Projekt wurde während unserer Arbeit, am 3. April 2012, veröffentlicht[3]. Das Projekt ist Open-Source und lässt sich somit komplett in einer eigenen Umgebung installieren[4].

Daneben bietet CartoDB aber wie Google auch die Möglichkeit, den Dienst von ihnen zu beziehen (IaaS). Obwohl die Konzepte sehr verschieden sind, sind die beiden Produkte durchaus miteinander vergleichbar[5].

2.2. Vision/Umsetzungskonzept

Um das Potential von Google Fusion Tables bewerten zu können, mussten wir uns zuerst mit dem Dienst vertraut machen. Die Idee war es, verschiedene Prototypen zu erstellen, welche jeweils verschiedene Aspekte des Produktes aufzeigen. Die Prototypen sollten sich dabei an gängige Aufgabenstellungen aus dem GIS-Bereich anlehnen.

Neben dem GIS-Aspekt war es uns wichtig, unabhängige Lösungen zu bauen, welche auf möglichst vielen Plattformen laufen. Aus diesem Grund haben wir uns einerseits für webbasierte Applikationen entschieden, andererseits in möglichst allen Bereichen mit Cloud-gestützten Diensten gearbeitet. Der Serveranteil sollte dabei so gering wie möglich sein, um keine unnötigen Abhängigkeiten und Hürden zu schaffen.

Da wir der festen Überzeugung sind, dass mobile Applikationen immer wichtiger werden, haben wir Wert darauf gelegt, dass die Prototypen mobile-fähig sind.

2.3. Resultate der Arbeit

Wir haben das gesteckte Ziel, das Potential von Google Fusion Tables abzuschätzen, sehr gut erreicht. Unser schrittweises Vorgehen hat sich ausbezahlt. Wir konnten zu Beginn die Theorie und den Praxiseinsatz sehr rasch lernen, in dem wir zahlreiche Beispielapplikationen entwickelt haben. So konnten wir uns langsam einen Überblick über die Möglichkeiten und Limitationen von GFT verschaffen.

[1] http://cartodb.com/
[2] http://postgis.refractions.net/
[3] Welcome to CartoDB 1.0 http://blog.cartodb.com/post/20403296927/welcome-to-cartodb-1-0
[4] Repository mit Installationsanleitung: https://github.com/Vizzuality/cartodb
[5] CartoDB hat einen Vergleich zwischen ihrem Produkt und Google Fusion Tables veröffentlicht. Auch wenn der Artikel stark einseitig gefärbt ist, zeigt er doch ein paar interessante Unterschiede auf: http://blog.cartodb.com/post/21264086445/comparing-fusion-tables-to-open-source-cartodb

Während der Arbeit hat es sich dann ergeben, dass wir sogenannte *Trusted Tester* für das neue API von Google wurden. Dort konnten wir neue Features ausprobieren und standen in direktem Kontakt mit den Fusion Tables Entwicklern (siehe Abschnitt 3.1.5). Gerade mit dem neuen API haben wir komplettes Neuland betreten. Die von uns geschriebene JavaScript Library bietet einen einfachen Zugang zu diesem API.

Später haben wir uns dann auf die Umsetzung der Use Cases konzentriert. Die Idee dabei war es, möglichst realistische Szenarien zu finden und dann eine mögliche Lösung mit Google Fusion Tables aufzuzeigen. Beim *WorldData* Use Case lag dabei das Gewicht auf dem Importieren und Zusammenführen von Daten aus verschiedenen Quellen sowie der Handhabung von grösseren Datenmengen (siehe Kapitel 6). Beim zweiten Use Case *FixMyStreet* stand eher die Kollaboration, der Live-Gedanke und die Nutzung von GFT als Applikations-Datenbank im Vordergrund (siehe Kapitel 7).

Ursprünglich wollten wir drei Use Cases umsetzen, wir haben uns dann aber im Verlauf der Arbeit und in Absprache mit unserem Betreuer und dem Industriepartner dazu entschieden, nur zwei Use Cases umzusetzen. Diese Entscheidung hat uns ermöglicht, einen der beiden Use Cases zu vertiefen und daraus einen vollwertigen Web-App Prototypen zu bauen.

Zusätzlich haben wir ein Tool bereitgestellt, mit dem sich zum einen Geodaten in andere Formate konvertieren lassen und zum anderen direkt Dateien als Tabellen in Google Fusion Tables importieren lassen (siehe Kapitel 8).

2.4. Schlussfolgerungen und Ausblick

Während der Evaluation der Google Fusion Tables wurden viele nützliche Features, aber auch einige Nachteile gefunden. So bietet Google mit den GFT einen kostenlosen Dienst an, welcher für kleinere Anwendungen genügend bietet.

Das SQL API ist an einigen Stellen noch unvollständig und lässt sich noch nicht mit dem einer vollwertigen Datenbank vergleichen. Die Dokumentation davon ist hingegen sehr gut und ausführlich. Falls doch Fragen auftauchen, gibt es zudem eine aktive Community, an die man sich wenden kann.

Ein grosses Plus ist die einfache Bedienung und die zahlreichen Visualisierungsmöglichkeiten, welche Google bereits anbietet. Durch die Integration in Google Maps können Daten sehr einfach auf einer Karte dargestellt werden.

Andererseits gibt es aber einige harte Limiten, welche einen daran hindern, grössere Applikationen mit Fusion Tables zu bauen. So es kann beispielsweise bei grösseren Datensammlungen problematisch sein, dass nur die ersten 100'000 Datensätze einer Tabelle für Resultat einer Abfrage berücksichtigt werden. Auch die Anzahl gleichzeitiger Zugriffe ist limitiert, sofern man dafür nicht extra bezahlen möchte.

Wenn man die Features ansieht, merkt man schnell, dass lediglich ein Typ Use Case unterstützt wird, jedoch nicht viel mehr. Es gibt zwar die Möglichkeit über Merged Tables einen `LEFT OUTER JOIN` von 2 Tabellen hinzubekommen, jedoch wäre die normale SQL `JOIN`-Syntax sehr willkommen, da sich damit beliebige Abfragen realisieren lassen.

Die OAuth Dokumentation ist zwar für den Normalfall (Zugriff auf Tabellen eines Benutzers) sehr gut dokumentiert. Das neue *Service Account*-Modell findet aber noch fast keine Erwähnung. Leider sind momentan die Berechtigungen nur sehr grob einstellbar. So gibt es lediglich einen OAuth-Scope, und dieser ermöglicht alle Schreiboperationen auf einem Account. Für einige Situationen wäre es wünschenswert, gewisse SQL-Befehle (z.B. `DELETE`) zu verbieten.

Das Web-GUI der Google Fusion Tables lässt ebenfalls noch einige Wünsche offen. Zum einen läuft es nicht sehr stabil und zum anderen ist die Bedienung nicht intuitiv. Das neue, sogenannte *experimental UI*, hat schon einige Verbesserungen gebracht, es fehlen aber einige Funktionen, welches das „alte“ UI bot. So lassen sich zum Beispiel keine Views mit einer `WHERE`-Klausel über das Web-GUI definieren.

Das neue API sieht sehr vielversprechend aus. Die Entwickler bei Google sind grundsätzlich offen für Anregungen, Kritik oder Feature Requests. Es bleibt abzuwarten, wie sich GFT und das Konkurrenzprodukt CartoDB zukünftig positionieren werden.

Teil II.

Projektdokumentation

3. Einführung in Google Fusion Tables

3.1. Was ist Google Fusion Tables?

Google Fusion Tables ist eine Datenbank-Dienst in der Cloud, welche von Google zur Verfügung gestellt wird. Die Datenbank ist spezialisiert auf das Speichern und Visualisieren von geografischen Daten. Der Dienst wurde am 10. Juni 2009 der Öffentlichkeit zugänglich gemacht[1]. Das erklärte Ziel dabei war es, die Nutzung einer Datenbank so einfach wie möglich zu machen.

Abbildung 3.1.: Tabellenansicht von Google Fusion Tables im Web-GUI

3.1.1. Kollaboration und Nutzung von öffentlichen Daten

Der Gedanke der Cloud lässt sich abseits vom Technischen noch weiterdenken. Durch die allgemeine Verfügbarkeit der Fusion Tables Datenbank, lassen sich die dort eingetragenen

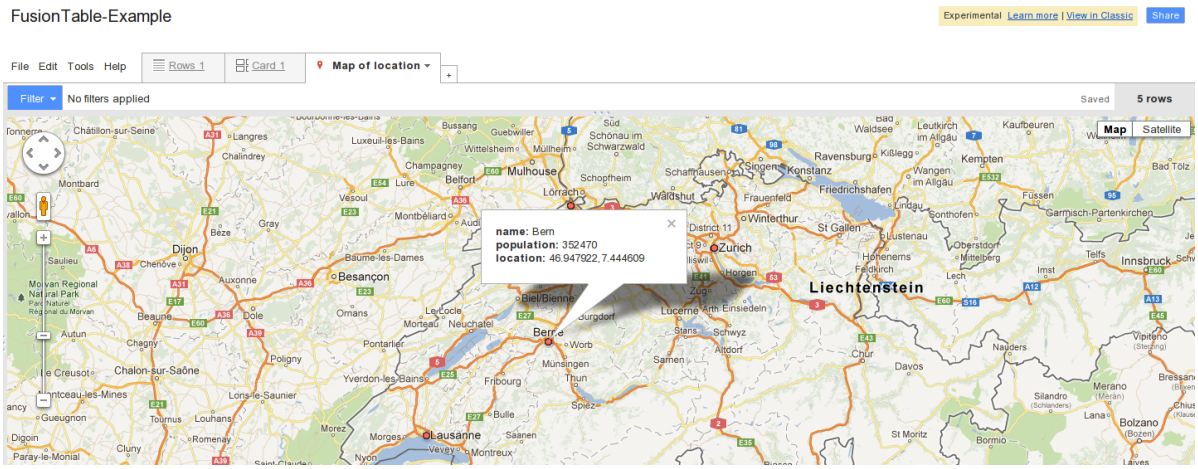


Abbildung 3.2.: Kartenansicht von Google Fusion Tables im Web-GUI

Daten mit anderen Benutzern oder gar der Öffentlichkeit teilen. Google bietet sogar die Möglichkeit öffentliche Tabellen zu durchsuchen[1].

Tabellen können zudem mit anderen Tabellen gemerged werden, wodurch die Vereinigung der Daten wiederum neue Möglichkeiten zur Nutzung der Daten ermöglicht. Eine öffentlich zugängliche Tabelle mit Länderpolygonen lässt sich so beliebig oft verwenden, um Daten mit Ländern zu verknüpfen und diese dann auf einer Karte darzustellen.

Diese passive Kollaboration ermöglicht es, auf eine breite Palette an öffentlichen bzw. veröffentlichten Daten zurückzugreifen. Via Import lassen sich auch andere bereits bestehende Daten mit Fusion Tables nutzen. Von Google werden dabei Tabellen (via Google Spreadsheet), CSV- und KML-Dateien unterstützt. Mit unserem im Kapitel 8 vorgestellten Build, lässt sich zudem eine breite Palette weiterer Dateiformate in Google Fusion Tables importieren.

Um aktiv zu kollaborieren, hat Google auch einige Features angedacht. So ist es möglich, einzelne Datensätze oder gar Zellen in der Tabelle zu kommentieren, sofern man die nötigen Berechtigungen dafür hat. Falls man Daten von mehreren Lieferanten verwalten lassen will, kann eine Partei eine Tabelle erstellen und verschiedene Views darauf erstellen. Diese werden jeweils von den verschiedenen Datenlieferanten selbst gepflegt werden. Durch entsprechend gesetzte Berechtigungen kann so jeder seine Daten zum Ganzen beitragen, ohne Zugriff auf die Daten von anderen Lieferanten zu haben. Einzig der Besitzer der Tabelle hat Berechtigungen den ganzen Datenbestand zu verändern. Google hat als Beispiel für diesen Use Case die Applikation *Flu Vaccine Finder* erstellt, welche es den Anbietern von Grippeimpfungen ermöglicht, ihre Lokale selbstständig zu erfassen und zu verwalten.[5]

[1] https://www.google.com/fusiontables/search

3.1.2. Verschiedene Tabellenarten

Google Fusion Tables unterscheidet grundsätzlich drei verschiedene Tabellenarten[2]. Jede Tabelle hat sowohl einen Namen wie auch eine numerische und *verschlüsselte* ID (`encid`). Angesprochen wird die Tabelle ausschliesslich über deren ID, wobei das neue API (siehe Abschnitt 3.4) nur noch die `encid` akzeptiert.

Base Table

Eine normale Tabelle kann sowohl über das Web-GUI, wie auch über das API erstellt werden. Eine Tabelle kann Daten entweder von einer KML- oder einer CSV-Datei importieren oder von einer bereits vorhandenen Google Spreadsheet-Datei übernehmen.

Merged Table

Merged Tables sind ein Mittel, um zwei Fusion Tables zusammenzuführen. Der Benutzer muss dabei Berechtigungen für beide Tabellen haben. Im Merge-Dialog kann dann die `JOIN`-Spalte ausgewählt werden. Sobald der Merge-Vorgang abgeschlossen ist, gibt es eine neue *virtuelle* Tabelle, welche die Daten aus beiden zugrundeliegenden Tabellen enthält. Der Merge führt dabei intern einen `LEFT OUTER JOIN` durch, wodurch alle Daten der Orginaltabelle erhalten bleiben, jedoch nur die Daten der zweiten Tabelle, welche dem `JOIN`-Kriterium entsprechen[3]. Änderungen an den zugrundeliegenden Tabellen sind sofort sichtbar.

Merged Tables haben einige Einschränkungen. So lassen sie sich bislang nicht via API erstellen, sondern aussschliesslich über das Web-GUI. `INSERT`-Befehle sind nicht möglich und `UPDATE` nur auf Attributen, die nicht am `JOIN` beteiligt sind.

Wir haben die Merged Tables in unserer erstellen Webapplikation *WorldData* verwendet (siehe Kapitel 6).

View

Mit Views können Einschränkungen von Base Tables vorgenommen werden. Dies kann einserseits die Projektion betreffen, so dass eine View nicht mehr alle Attribute der Orginaltabelle beinhaltet. Andererseits aber auch die Daten selbst, nämlich dann, wenn eine Einschränkung (`WHERE`-Klausel) definiert wurde. Letzteres ist jedoch über das Web-GUI nicht möglich.

Views sind auch ein ideales Mittel, um die Berechtigungen zu steuern. So kann z.B. der lesende Zugriff auf eine Menge von Daten gewährt werden, jedoch der schreibende Zugriff nur auf eine Untermenge, indem man eine entsprechende View erstellt. Wir haben dieses

[2] Tabellenarten in Google Fusion Tables: https://developers.google.com/fusiontables/docs/developers_guide#Exploring

[3] https://developers.google.com/fusiontables/docs/developers_guide#Terminology

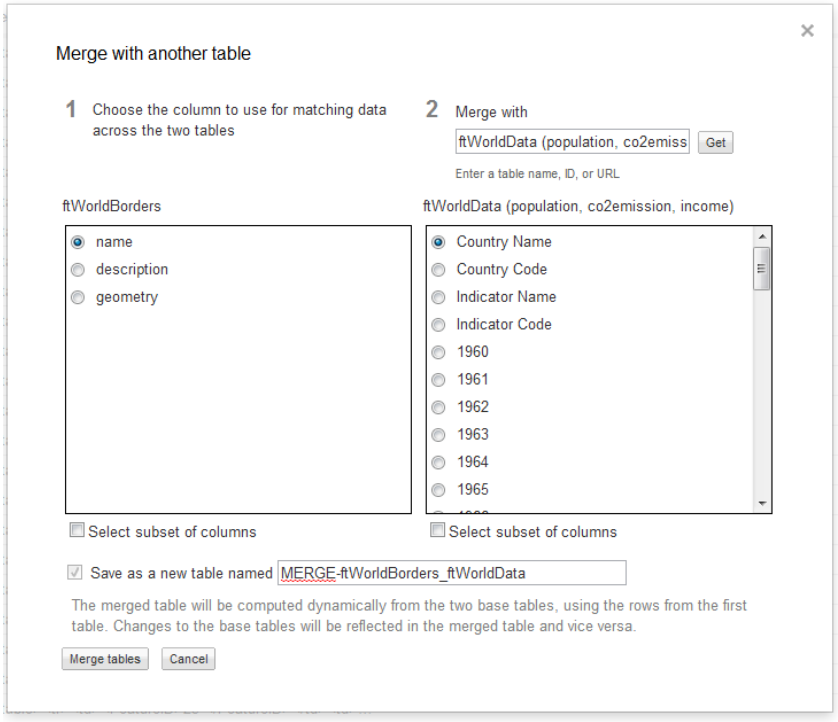


Abbildung 3.3.: Erstellen einer Merged Table über das Web-GUI

Prinzip in unserem *FixMyStreet* Use Case verwendet (siehe Kapitel 7)

Views können nur von Base Tables oder anderen Views erstellt werden, Merged Tables werden nicht unterstützt[4].

Ein Beispiel für die Verwendung von Views in Google Fusion Tables ist der *Flu Vaccine Finder*[5] von Google selbst.

3.1.3. Datentypen

Google Fusion Tables unterstützt eine Reihe von Datentypen[6]. Das Herzstück bildet dabei sicher der Typ *Location*, welcher auf vielseitige Weise geografische Daten speichern kann[7]. Jede Zelle hat eine Begrenzung von maximal 1'000'000 Zeichen.

[4] https://developers.google.com/fusiontables/docs/developers_guide#CreatingView
[5] https://developers.google.com/fusiontables/docs/articles/data_gathering
[6] http://support.google.com/fusiontables/bin/answer.py?hl=en&answer=2590990
[7] http://support.google.com/fusiontables/bin/answer.py?hl=en&answer=174680

Datentyp	Beschreibung
`Text`	Standard Datentyp, falls es sich dabei um eine URL handelt, wird diese automatisch zu einem Hyperlink oder Bild umgewandelt
`Number`	Integer- oder Dezimalzahlen
`Date/Time`	Datumswerte, nach welchen sortiert werden kann
`Location`	Geografische Daten: • Text: beliebiger Ortsangabe (z.B. Adresse oder Landesname) • Punkte (Point): Latitude/Longitude-Paar mit Leerzeichen getrennt oder KML-Punktdaten • Linien (Line): KML mit LineString-Element • Polygon: KML mit Polygon-Element Alle Daten dieses Typs werden geocodiert (siehe Abschnitt 3.5)

Tabelle 3.1.: Datentypen von Google Fusion Tables

3.1.4. Limiten

Da die Google Fusion Tables kostenlos angeboten werden, hat Google einige Limiten für die Nutzung der Datenbank festgelegt. So darf eine einzelne Tabelle maximal 250MB Speicher belegen. Zudem sind die Abfragen pro Benutzer und Tag auf 25'000 beschränkt. Auf der Karte lassen sich maximal 100'000 Elemente gleichzeitig darstellen. Allgemein werden zudem bei Abfragen nur die ersten 100'000 Datensätze der Datenbank für das Erstellen der Antwort berücksichtigt. Diese Einschränkungen können Kunden des Google Maps API Premier[8] auf Anfrage verändern. Die Preise dafür starten bei $10'000 pro Jahr und steigen je nach Bedürfnissen nach oben.[3]

Beschreibung	Limitation
Speicher pro Tabelle	250MB
Abfragen pro Benutzer und Tag	25'000
Gleichzeitig angezeigte Elemente auf Karte	100'000
Für Abfragen berücksichtigte Datensätze	erste 100'000

Tabelle 3.2.: Limitationen der Google Fusion Tables

[8] http://www.google.com/enterprise/earthmaps/maps.html

3.1.5. Austausch mit den Google Entwicklern

Da es sich bei Google Fusion Tables um ein relativ neues Projekt handelt, war uns bewusst, dass wir früher oder später vor Problemen stehen würden. Von daher war für uns klar, dass wir die beiden offiziellen Supportkanäle (Mailingliste[9] und StackOverflow[10]) aktiv nutzen werden. Zum einen ist es sehr interessant zu sehen, welche Ideen und Probleme andere Entwickler haben, zum anderen konnten wir so einen guten Kontakt zur Community pflegen.

Hauptsächlich wendeten wir uns via Mailingliste an die Google Entwickler, wo man in der Regel innerhalb eines Tages sehr gute Antworten auf Fragen bekommt. Auch ein von uns gemeldeter Bug[11] wurde sehr schnell behoben. Der Bug ist uns beim Wechsel auf das neue Trusted Tester API (siehe Abschnitt 3.4) dank unserer Unit-Tests aufgefallen. Leider gab es aber nicht für alle unsere Anfragen einfache Lösungen (siehe Abschnitt 3.5.2).

Im April fand ein Google+ Hangout (Videokonferenz) mit zwei Google Fusion Tables Entwicklern zum Trusted Tester API statt, wo Feedback und Anregungen gesammelt wurden. Da nicht viele Leute anwesend waren, konnten wir ausführlich Feedback geben und unsere Use Cases besprechen.

3.2. SQL API

Das SQL API ist eine Schnittstelle, mit welcher man mit SQL-ähnlichen Befehlen Daten aus Google Fusion Tables abfragen oder verändern kann. Sie verfügt bereits über eine grosse Palette an möglichen Befehlen[12]. Die SQL-Befehle werden als Parameter in folgender Form an das API übergeben:

https://www.googleapis.com/fusiontables/<apiVersion>/query?sql=<statement>

Lesende Zugriffe (`SELECT`, `SHOW TABLES`, `DESCRIBE`) werden dabei als `GET`-Request geschickt, schreibende Zugriffe (`CREATE`, `DROP`, `INSERT`, `UPDATE`, `DELETE`) mit der `POST`-Methode. Um Daten zu schreiben und für den Zugriff auf private Tabelle ist eine Authentifizierung (siehe Abschnitt 3.8) mit OAuth nötig.

[9] Die Mailingliste ist für allgemeine oder konzeptionelle Fragen: https://groups.google.com/forum/?fromgroups#!forum/fusion-tables-users-group

[10] Google hat alle technischen Fragen zu GFT in die bekannte Entwickler-Community StackOverflow ausgelagert: http://stackoverflow.com/questions/tagged/google-fusion-tables

[11] Bug-Report *COUNT() returns as NaN in Trusted Tester API*: http://code.google.com/p/fusion-tables/issues/detail?id=1086

[12] Befehlsreferenz: https://developers.google.com/fusiontables/docs/developers_reference

Befehl	Beschreibung
`SHOW TABLES`	Abfrage aller Tabellen des angemeldeten Benutzers
`DESCRIBE`	Bezeichnung und Datentypen aller Spalten in einer Tabelle
`CREATE TABLE`	Erstellen einer neuen Tabelle
`CREATE VIEW`	Erstellen einer View auf Grundlage einer bestehenden Tabelle
`SELECT`	Selektieren von Daten einer Tabelle
`INSERT`	Neue Zeile zu einer Tabelle hinzufügen
`UPDATE`	Daten in einer Tabelle verändern
`DELETE`	Daten aus einer Tabelle löschen
`DROP TABLE`	Löschen einer Tabelle

Tabelle 3.3.: Liste der verfügbaren SQL Befehle des SQL APIs

3.2.1. Abfragen (Queries)

Mit dem `SELECT`-Befehl des SQL API lassen sich Abfragen an Google Fusion Tables stellen. Die untenstehende Tabelle gibt eine Übersicht der Möglichkeiten.

Bereich	Beschreibung
Aggregation	Folgende Funktionen sind unterstützt: • `COUNT()` • `SUM(<column_name>)` • `AVERAGE(<column_name>)` • `MAXIMUM(<column_name>)` • `MINIMUM(<column_name>)`
Einschränkung auf Datensatz	`ROWID = <id>`

Einschränkung auf Attributen	Für Zahlen: `<column_name> <operator> <number>`, wobei `<operator>` folgende Werte haben kann: • `>, <,>=, <=, =` Für Strings: `<column_name> <operator> <string>`, wobei `<operator>` folgende Werte haben kann: • `>, <, >=, <=, =` • `LIKE` • `MATCHES` • `STARTS WITH` • `ENDS WITH` • `CONTAINS` • `CONTAINS IGNORING CASE` • `DOES NOT CONTAIN` • `NOT EQUAL TO` • `IN` Formeln oder Vergleiche mit anderen Attributen der Tabelle sind nicht unterstützt.
Einschränkung auf Anzahl Datensätze	`LIMIT <number>`, wobei `<number>` angibt, wie viele Datensätze des Resultats zurückgeliefert werden sollen. Mit `OFFSET <number>` kann der Bereich, ab welchem das Limit zählt, verändert werden.

Tabelle 3.4.: Liste der verfügbaren Abfragemöglichkeiten des SQL APIs

Um mehrere Bedingungen in einer `WHERE`-Klausel zu verwenden, können diese mit `AND` verbunden werden. Verknüpfungen mit `OR` werden hingegen nicht unterstützt.

3.2.2. Ortsbezogene Abfragen (Spatial-Queries)

Das SQL API bietet zudem eine Reihe von speziellen ortsabhängigen Abfrage-Möglichkeiten, welche in der folgenden Tabelle dokumentiert sind.

Spatial Keyword	Beschreibung
`ST_INTERSECTS( <location_column>, <geometry> )`	Liefert alle Zeilen zurück, welche sich innerhalb der definierten Geometrie `<geometry>` befinden. • Als `<location_column>` muss eine Spalte der Tabelle, vom Typ *Location*, angegeben werden. • Als `<geometry>` kann entweder ein `CIRCLE` oder ein `RECTANGLE` verwendet werden. Ein `POLYGON`-Typ fehlt derzeit noch. `ST_INTERSECTS` kann als Bedingung in der `WHERE`-Klausel des Statements verwendet werden. *Hinweis: `ST_INTERSECTS` und `ST_DISTANCE` dürfen nicht zusammen im gleichen Statement verwendet werden.*
`ST_DISTANCE( <location_column>, <coordinate> )`	Liefert die Datensätze sortiert nach der Distanz zur angegebenen Koordinate `<coordinate>` zurück. • Als `<location_column>` muss eine Spalte der Tabelle, vom Typ *Location*, angegeben werden. • Die `<coordinate>` stellt die Koordinate dar, zu welcher der Abstand gemessen werden soll. `ST_DISTANCE` kann als Bedingung in der `ORDER BY`-Klausel des Statements verwendet werden. *Hinweis: `ST_INTERSECTS` und `ST_DISTANCE` dürfen nicht zusammen im gleichen Statement verwendet werden.*
`CIRCLE( <coordinate>, <radius> )`	Wird verwendet, um einen Kreis von der angegebenen Koordinate `<coordinate>` mit dem Radius `<radius>` zu erhalten.
`RECTANGLE( <coordinate_1>, <coordinate_2> )`	Wird verwendet, um ein Rechteck mit den Ecken `<coordinate_1>` (links oben) und `<coordinate_2>` (rechts unten) zu erhalten.

Tabelle 3.5.: Liste der verfügbaren Spatial Queries des SQL APIs

3.3. Client Libraries

Google bietet Client Libraries zum API in den Sprachen PHP[13] und Python[14] an. Aufgrund der Anforderung, dass unsere Webapplikation möglichst ohne Serverteil aus-

[13] http://code.google.com/p/fusion-tables-client-php/
[14] http://code.google.com/p/fusion-tables-client-python/

kommt, waren wir auf eine Bibliothek in JavaScript angewiesen. Da es bislang noch keine solche gab, haben wir begonnen, selbst die GftLib zu entwickeln (siehe Abschnitt 3.7).

Durch die Same-Origin-Policy[15], wurden wir daran gehindert, AJAX-Requests direkt auf das Google API abzusetzen. Wir mussten eine Alternative finden, ohne eine Serverkomponente einzuführen.

Die Lösung für das Problem fanden wir in den Google Groups. Google bietet ein inoffizielles JSONP API[16] an, welches es erlaubt, AJAX-Requests auch über die eigene Domäne hinweg zu senden. Diese Methode funktionierte jedoch nur für lesende Zugriffe.

Für die schreibenden Zugriffe haben wir als Workaround einen *Relay-Dienst* geschrieben, welcher auf unserem Webserver lief. Der Dienst hat lediglich die Requests entgegengenommen und diese dann an das Google API weitergeleitet.

Als wir den Zugriff auf das Trusted Tester API (siehe Abschnitt 3.4) bekommen haben, konnten wir den Serverteil wieder entfernen. Google hat dort die Möglichkeit hinzugefügt hat, direkt mit ihrem JavaScript API Client[17] Requests ans API zu schicken. Dadurch entfällt die Same-Origin-Policy Problematik.

3.4. Trusted Tester API

Es gibt derzeit zwei verschiedene Versionen des APIs: eine öffentlich zugängliche und das sogenannte *Trusted Tester API*, welche derzeit im Beta-Stadium ist und nur ausgewählten Personen zur Verfügung steht. Als wir im Sprint 2 davon erfahren haben, haben wir uns für einen Zugang beworben, welchen wir dann auch erhalten haben.

Das Trusted Tester API bietet einige Neuerungen zur alten Schnittstelle. Es handelt sich bei diesem API um eine Vorabversion der neuen Schnittstelle, welche zukünftig ebenfalls der Öffentlichkeit zur Verfügung stehen soll. Neben dem API gibt es auch eine zugehörige Mailingsliste, auf der die Entwickler von Google Fragen beantworten und Tipps geben.

Eine grosse Neuerungen des APIs, bildet die REST-Schnittstelle. Damit ist es möglich, Tabelleninformationen abzufragen oder ganz klassisch CRUD-Operationen auf dem API auszuführen. Da wir uns bemüht haben, unsere Beispiele möglichst ohne Server-Code zu schreiben, konnten wir besonders davon profitieren, mit dem neuen API `POST`-Requests abzuschicken und so auch Schreiboperationen direkt via Browser zu unterstützen.

Alle Request an das REST-API haben die folgende Form:

[15] Die Same-Origin-Policy (SOP) ist ein Sicherheitskonzept, das es JavaScript und ActionScript nur dann erlaubt, auf Objekte einer anderen Webseite zuzugreifen, wenn sie aus derselben Quelle (Origin) stammen.[17]

[16] https://groups.google.com/forum/?hl=en&fromgroups#!topic/fusion-tables-users-group/jCiFfqCgCWM

[17] Google APIs Client Library for JavaScript: http://code.google.com/p/google-api-javascript-client/

https://www.googleapis.com/fusiontables/<apiVersion>/<resourcePath>?<parameters>

Folgende Ressourcen sind derzeit unterstützt:

- Table
- Column
- Template
- Style

Eine *Row* oder *Query* Ressource fehlt noch. Um Abfragen zu machen, muss nach wie vor das SQL API (siehe Abschnitt 3.2) verwendet werden.

Ziel	**HTTP Mapping**
Alle Ressourcen eines Typs auflisten	`GET` auf einem Ressource-Typ
Eine spezifische Ressource holen	`GET` auf einer Ressource
Eine neue Ressource einfügen (kreiert eine neue Ressource)	`POST` auf einem Ressource-Typ (mit Daten, um eine neue Ressource zu erstellen)
Aktualisieren einer bestehenden Ressource	`PUT` auf einer Ressource (mit Daten, um die Ressource zu aktualisieren)
Löschen einer Ressource	`DELETE` auf einer Ressource

Tabelle 3.6.: HTTP-Mapping des REST-APIs

3.5. Geocodierung in Google Fusion Tables

Ein grosser Vorteil der Google Fusion Tables ist die automatische Geocodierung von Standortdaten. Sobald eine neue Zeile via Web-GUI zu einer Tabelle hinzugefügt wird, werden alle Zellen vom Typ *Location* einem eindeutigen Standort auf der Karte zugewiesen. Ist dies nicht möglich, da beispielsweise eine Adresse in mehreren Orten vorkommen kann, wird die Zelle nicht geocodiert und der Text wird gelb hinterlegt. In solchen Fällen hat man die Möglichkeit den zugehörigen Ort manuell mit Hilfe einer Karte zu wählen.

name	description	population	location
Springfield	gelbe Stadt	100000	Springfield

Abbildung 3.4.: Geocodierung für Ort Springfield fehlgeschlagen

Die geocodierten Standorte werden als Metadaten in der Tabelle hinterlegt. Leider sind diese Daten über das SQL API nicht selektierbar. Möchte man die erhaltenen Daten auf

der Karte positionieren, so muss man für jede Zeile die Geocodierung manuell vornehmen, was sich negativ auf die Ladezeit der Karte auswirkt.

3.5.1. Geocoding-Dienste

Es gibt verschiedene Dienste, welche eine solche Geocodierung von Standortdaten anbieten. Die meisten davon haben aber eine Begrenzung der möglichen Anfragen pro Tag.

Anbieter	**Anfragen pro Tag**	**URL**
Google Maps Geocoding API	2500	https://developers.google.com/maps/documentation/geocoding/?hl=de
Yahoo! PlaceFinder API	50000	http://developer.yahoo.com/geo/placefinder/
MapQuest Geocoding API	keine Begrenzung	http://developer.mapquest.com/web/products/dev-services/geocoding-ws

Tabelle 3.7.: Geocoding Limitierungen

3.5.2. Geocoding von neuen Datensätzen

Ein grosses Problem stellt sich darin, dass neue Datensätze, welche über das SQL API eingefügt wurden, nicht automatisch geocodiert werden. Dies führt dazu, dass diese Daten von Abfragen, welche eine ortsbezogene Einschränkung beinhalten (siehe Abschnitt 3.2.2), nicht zurückgeliefert werden.

Eine Geocodierung dieser Daten kann nur manuell über das Fusion Tables Web-GUI gestartet werden.

3.6. Google Maps API - FusionTablesLayer

Google bietet von Haus aus bereits eine Fusion Table-Integration im Google Maps API V3 an. Damit ist es möglich Fusion Tables als eigenständige Layer direkt auf der Karte darzustellen. Die Möglichkeiten dieser Layer sind noch stark eingeschränkt, die grundlegenden Funktionalitäten für das Arbeiten mit Geodaten sind aber bereits vorhanden.

So ist es möglich, Abfragen mit `WHERE`-Conditions einzuschränken oder die Stile des Layers selbst zu bestimmen.

3.6.1. Karten-Stile

Die FusionTablesLayer bieten ein abfragebasiertes Styling der Ebene an. Damit ist es möglich, Flächen oder Linien farblich hervorzuheben oder andere Icons für Markierung zu verwenden. Zu jedem Stil kann man eine Einschränkung festlegen, welche bestimmt, ob dieser für den aktuellen Datensatz angewendet wird oder nicht. Die Einschränkung entspricht grundsätzlich einer `WHERE`-Condition in der Abfrage.

Hinweis: Passen mehrere Einschränkungen auf einen Datensatz, erhält dieser den Stil, welcher als letztes definiert wurde. Dies kann zu ungewollten Resultaten führen.

Beispiel eines Stils:

```
styles: [{
  polygonOptions: {
    fillColor: "#00FF00", // gruen
    fillOpacity: 0.3
  }
}, {
  where: "birds > 300",
  polygonOptions: {
    fillColor: "#0000FF" // blau
  }
}]
```

Code-Ausschnitt 3.1: Beispiel eines FusionTablesLayer-Stylings

In diesem Beispiel[18] werden alle Polygone der Tabelle, welche in der Spalte *birds* eine Zahl grösser als 300 eingetragen haben, *blau* eingefärbt. Die restlichen Polygone erhalten eine *grüne* Färbung mit einer Deckkraft von 30%.

Einschränkungen

Das Google Maps API hat momentan folgende Einschränkungen bezüglich den Ebenenstilen definiert.

Beschreibung	**Limitation**
Anzahl Ebenen pro Karte	5
Anzahl Ebenen, auf denen Stile definiert sein dürfen	1
Anzahl Stile auf Ebene	5

Tabelle 3.8.: Limitationen der Stile auf Fusion Table-Ebenen

[18] Quelle: https://developers.google.com/maps/documentation/javascript/layers?hl=de-DE#fusion_table_styles (Stand: 21.05.2012)

3.6.2. Heatmaps

Ein weiteres Feature der Fusion Table-Ebenen ist die Möglichkeit, die Daten der Tabelle direkt als Heatmap darzustellen. Dabei werden die Daten automatisch nach der Häufigkeit der Vorkommnisse an einem Ort anders eingefärbt. Der verwendete Farbverlauf geht dabei von *grün* (für wenig Daten) bis *rot* (für viele Daten).

Abbildung 3.5.: Daten als Heatmap mit FusionTablesLayer

Leider sind momentan die Konfigurationsmöglichkeiten der Heatmap auf ein Minimum beschränkt. Eine Legende lässt sich beispielsweise nicht anzeigen. So kann man nicht genau sagen, welche Werte nun hinter den verschiedenen Farben stecken. Zudem lassen sich diese auch nicht verändern.

3.6.3. Performance

Der grösste Vorteil der Fusion Table-Ebenen steckt aber nicht zwingend in den sichtbaren Features. Man findet ihn wohl eher darin, dass die Geocodierungen der Standort-Daten direkt aus der Tabelle gelesen werden und nicht manuell vom Client abgefragt werden müssen. Dadurch kann die Ebene komplett auf den Servern von Google aufbereitet werden. Der Client muss die erhaltenen Daten lediglich noch darstellen. Der Vorteil davon wird durch das Diagramm 3.6[19] schnell ersichtlich.

Die Zeit für das Rendering der Karte bleibt demnach bei der Verwendung von Fusion Table-Ebenen konstant und somit unabhängig von der Anzahl Markierungen, welche gesetzt werden müssen. Der Rechenaufwand, der für das Erstellen der JavaScript Marker-Objekte verwendet werden müsste, wird direkt von den Google Servern übernommen und das Resultat als Bild zum Client gesendet. Daraus resultiert die konstante Zeit, welche für die Anfrage zum Server und für das Senden der Antwort zum Client verwendet wird.

[19] Quelle: http://www.google.com/events/io/2011/sessions/managing-and-visualizing-your-geospatial-data-with-fusion-tables.html (Stand: 17.05.2012)

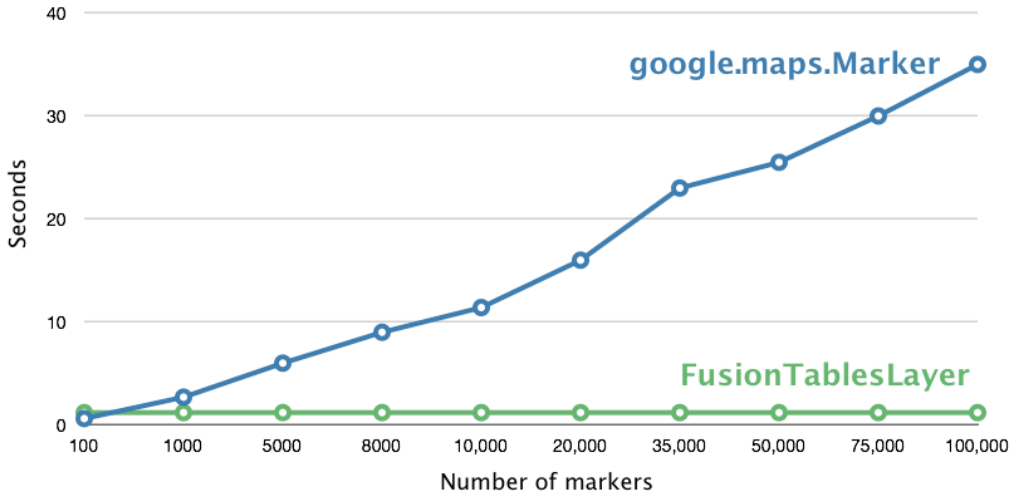


Abbildung 3.6.: FusionTablesLayer verglichen mit Markers

3.6.4. Einschränkungen

Freigabe der Tabelle

Ein grosser Nachteil der Fusion Table-Ebenen besteht darin, dass die verwendeten Fusion Tables als *öffentlich* markiert sein müssen, um diese auf einer Karte darzustellen. Dies bedeuted, dass jeder die Tabellen anzeigen oder auslesen kann. Es ist also nicht möglich, eine Tabelle mit sensiblen Daten als Fusion Table-Ebene darzustellen.

Von Google wird zur Lösung dieses Problems aber folgendes Vorgehen vorgeschlagen: Man kann für Tabellen mit sensiblen Inhalten eine View erstellen, welche lediglich die öffentlichen Spalten und Zeilen selektiert. Diese View könnte man dann als *öffentlich* markieren und in einer Fusion Table-Ebene verwenden.

Eine Ausnahme dieser Regelung bilden dabei die Maps API Premier Kunden. Diese haben die Möglichkeit eine Fusion Table als *Protectet Map Layer* freizugeben, wodurch sich diese nur in einer definierten Applikation als Ebene einbinden lässt. Die Tabelle bleibt dabei komplett privat und kann nicht ausgelesen werden.

Event-Handling auf Fusion Table-Ebenen

Bislang ist es erst möglich, den Klick-Event auf einer Fusion Table-Ebene zu behandeln. Dieser liefert standardmässig das HTML-Template des InfoWindows mit. Zudem erhält man die Daten der „angeklickten Zeile“ beziehungsweise die zugehörigen Daten des angeklickten Objekts auf der Karte. Man hat die Möglichkeit, dieses Template anzupassen bevor das InfoWindow angezeigt wird.

Weitere Events können auf einer Fusion Table-Ebene aber noch nicht behandelt werden. Es ist also nicht möglich, beispielsweise ein Objekt auf der Karte bei einem MouseOver-Event anders einzufärben oder ähnliches.

Diese fehlenden Möglichkeiten wurden schon oft von anderen FusionTables-Benutzern bei den Google-Entwicklern angefordert, welche dies ebenfalls als ein wichtiges Feature sehen, das noch implementiert werden muss.

Als Übergangslösung findet sich bereits eine Custom Library mit dem Namen *Fusion Tips*[20]. Diese legt über die Fusion Table-Ebene eine weitere transparente Ebene. Auf dieser Ebene kann man alle Events, welche das Google Maps API anbietet, behandeln. Sobald die Maus über die Position eines Elementes auf der Fusion Table fährt, sendet der Event einen Request über das SQL API an die Fusion Table und holt sich die Informationen zu der entsprechenden Zeile. So ist es möglich, ein neues Element mit der gleichen Form, aber beispielsweise einer anderen Farbe darüber zu legen.

Für den Benutzer entsteht so der Effekt, als würde sich die Farbe des gehoverten Elementes ändern.

Die Libary wird in folgendem Blog-Artikel sehr gut erklärt: http://csessig.wordpress.com/2012/02/07/multiple-layers-and-rollover-effects-for-fusion-table-maps/

Weitere Einschränkungen

Zusätzlich sind die allgemeinen Limitationen der Google Fusion Tables (siehe Tabelle 3.2) zu beachten.

3.7. Google Fusion Table JavaScript Library (GftLib)

Die Google Fusion Table JavaScript Library (GftLib) ist eine von uns entwickelte JavaScript Bibliothek, welche die Kommunikation mit dem Google Fusion Table SQL API (siehe Abschnitt 3.2) vereinfacht. Sie hilft dabei, SQL-Abfragen zu erstellen und diese per AJAX an das API zu senden. Zudem kümmert sich die Library um die Authentifizierung via OAuth, so dass auch schreibende Zugriffe oder Zugriffe auf private Tabellen möglich sind.

Die Bibliothek hat Abhängigkeiten zu jQuery[21] (für einige Hilfsfunktionen) sowie dem Google API JavaScript Client[22] (für die Authentifizierung und das Abschicken der Requests).

Die Bibliothek besteht aus zwei Klassen (GftLib und SqlBuilder). Die GftLib ist das API

[20] http://gmaps-utility-gis.googlecode.com/svn/trunk/fusiontips/docs/examples.html
[21] http://jquery.com/
[22] http://code.google.com/p/google-api-javascript-client/

zum Client und steuert dementsprechend die Requests und die Authentifizierung. Der SqlBuilder ist eine Hilfsklasse, welche es erlaubt, auf einfach Art und Weise SQL-Befehle zu generieren.

3.7.1. Verwendung

Im Code-Beispiel 3.2 sieht man die GftLib und den SqlBuilder im Einsatz. Zuerst werden alle benötigten Werte initialisiert (Zeilen 1-3), dann wird die Ausgabefunktion `printer` definiert, welche später als Callback-Funktion dient für den Request. Das heisst, dass die Funktion aufgerufen wird, sobald die Antwort der Abfrage zurückkommt.

Mit der Funktion `convertToObject()` wird die Antwort von Google Fusion Tables in eine für den Benutzer besser verständliche Form umgewandelt. Das Resultat der Fusion Tables kommt in Form eines zweidimensionalen Arrays zurück, welches für jeden Datensatz alle abgefragten Felder enthält. Die Funktion erstellt jeweils pro Datensatz ein Objekt, welches die angefragten Felder als Properties hat. So kann man mit deren Namen auf ihre Werte zugreifen (z.B. `places[i].name` oder `places[i].population`).

In diesem Beispiel wird das SQL Query direkt in der Funktion `execSelect()` erzeugt, welche dazu den SqlBuilder verwendet.

```
1  var tableId = '1LWXSMsZINyfjAKGqeS-822wi4WmlaGmmvh20Ujw';
2  var gft = new GftLib();
3  var resultList = document.getElementById("result");
4
5  var printer = function(data) {
6    var places = gft.convertToObject(data);
7
8    for (var i = 0; i < places.length; i++) {
9      var listElem = document.createElement('li');
10     listElem.innerHTML = 'Place: ' + places[i].name + ' / Population:
           ' + places[i].population;
11     resultList.appendChild(listElem);
12   }
13 }
14
15 gft.execSelect(printer, {table:tableId, fields:['name', 'population'
       ]});
```

Code-Ausschnitt 3.2: Verwendung der GftLib

3.7.2. Abhängigkeiten

Library	Version	Verwendung
jQuery	1.7.1-min	Helper-Funktionen und AJAX-Requests für den OAuthTokenService
Google APIs Client Library	ALPHA release	AJAX-Requests an das Google API und Authentifizierung

Tabelle 3.9.: Abhängigkeiten der GftLib

3.7.3. Methoden

GftLib

Methode	Beschreibung	Parameter
`execQuery( callback, query )`	Führt eine SQL-Abfrage aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `query`: SQL-Query
`execSql( callback, sql )`	Führt einen beliebigen SQL-Befehl aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `sql`: SQL-Befehl
`execSelect( callback, options )`	Führt eine `SELECT`-Abfrage aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `options`: Parameter-Objekt für `SqlBuilder.selectStmt()`
`execInsert( callback, options )`	Führt einen `INSERT`-Befehl aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `options`: Parameter-Objekt für `SqlBuilder.insertStmt()`
`execUpdate( callback, options )`	Führt einen `UPDATE`-Befehl aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `options`: Parameter-Objekt für `SqlBuilder.updateStmt()`

`execDelete( callback, options )`	Führt einen `DELETE`-Befehl aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `options`: Parameter-Objekt für `SqlBuilder.deleteStmt()`
`getTableDescription( callback, options )`	Führt eine `DESCRIBE`-Abfrage aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `options`: Parameter-Objekt für `SqlBuilder.describeStmt()`
`createView( callback, options )`	Führt einen `CREATE VIEW`-Befehl aus	• `callback`: Callback-Methode, welche nach Beendigung der Methode aufgerufen wird. • `options`: Parameter-Objekt für `SqlBuilder.createViewStmt()`
`convertToObject( gftData )`	Konvertiert das Resultat einer Abfrage in sprechende Objekte	• `gftData`: Antwort auf einen SQL Befehl von Google Fusion Tables

Tabelle 3.10.: Methoden der GftLib-Klasse

SqlBuilder

Die Methoden des SqlBuilders nehmen jeweils ein Parameter-Objekt[23] entgegen und erstellen daraus SQL-Befehle, welche dann als Input für die GftLib gebraucht werden können.

Methode	Beschreibung	Parameter
`selectStmt( options )`	Generiert ein `SELECT` Statement	`options`: Parameter-Objekt mit folgenden Feldern: • `fields`: Array von Feldernamen (Default *) • `table`: ID der Tabelle oder View • `conditions`: Array von Bedingungen für die `WHERE`-Klausel

[23] http://www.refactoring.com/catalog/introduceParameterObject.html

`insertStmt( options )`	Generiert ein `INSERT` Statement	`options`: Parameter-Objekt mit folgenden Feldern: • `fields`: Array von Feldernamen • `table`: ID der Tabelle oder View • `values`: Array von Werten (in der gleichen Reihenfolge wie `fields`)
`updateStmt( options )`	Generiert ein `UPDATE` Statement	`options`: Parameter-Objekt mit folgenden Feldern: • `fields`: Array von Feldernamen • `table`: ID der Tabelle oder View • `values`: Array von Werten (in der gleichen Reihenfolge wie `fields`) • `conditions`: Array von Bedingungen für die `WHERE`-Klausel
`deleteStmt( options )`	Generiert ein `DELETE` Statement	`options`: Parameter-Objekt mit folgenden Feldern: • `table`: ID der Tabelle oder View • `conditions`: Array von Bedingungen für die `WHERE`-Klausel
`describeStmt( options )`	Generiert ein `DESCRIBE` Statement	`options`: Parameter-Objekt mit folgenden Feldern: • `table`: ID der Tabelle oder View
`createViewStmt( options )`	Generiert ein `CREATE VIEW` Statement	`options`: Parameter-Objekt mit folgenden Feldern: • `viewName`: Name der neuen View • `query`: Ein SQL-Query, das der View zugrunde liegt

Tabelle 3.11.: Methoden der SqlBuilder-Klasse

3.7.4. Implementation

Die GftLib verwendet den SqlBuilder um aus gegebenen Input-Parametern SQL-Befehle zu generieren. Diese Befehle werden dann mit Hilfe der Google API Bibliothek (*gapi*) abgeschickt. Das Google API schickt die Resultate dann via Callback-Funktion wieder zurück an die GftLib. Aus der *jQuery*-Bibliothek verwenden wir einige Helferfunktionen für Strings sowie die Möglichkeit auf einfache Art und Weise AJAX-Requests abzuschicken.

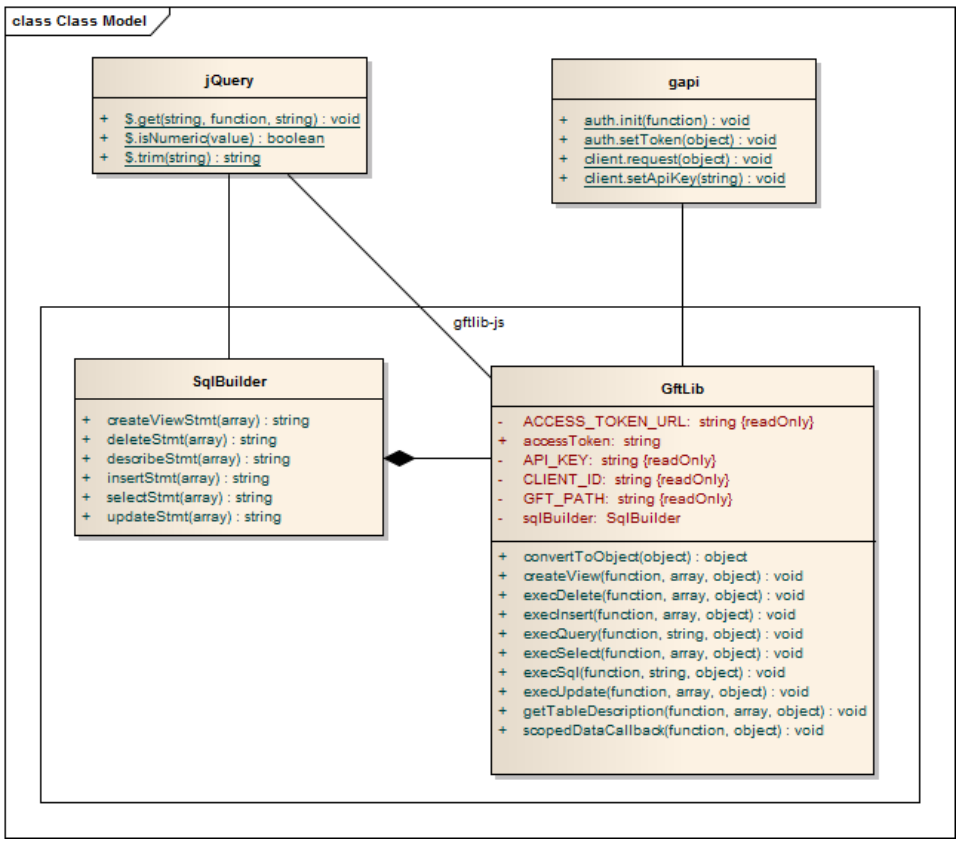


Abbildung 3.7.: GftLib Klassendiagramm

3.7.5. Testing

Mit Hilfe des Unit-Testing Frameworks QUnit[24] konnten wir für die GftLib durchgängige Tests erstellen. Die Tests wurden in JavaScript geschrieben und können deshalb direkt im Browser ausgeführt werden[25].

Wir haben insgesamt 84 Tests für die GftLib erstellt, welche jede Funktion mit verschiedenen Parametern prüft. Wir haben den *test early*-Ansatz gewählt, dabei werden

[24] http://docs.jquery.com/QUnit
[25] Produktive Tests: http://gft.rdmr.ch/test/js/

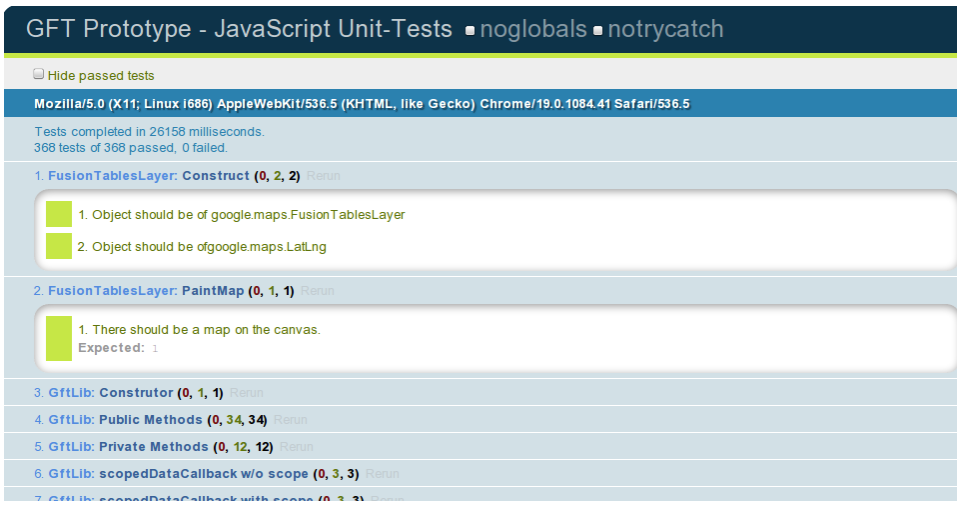


Abbildung 3.8.: QUnit Tests der GftLib

die Tests sehr zeitnah zur Erstellung einer Funktion geschrieben. Dank der guten Testabdeckung konnten wir unseren Code regelmässigen Refactorings unterziehen ohne uns darum Sorgen zu machen, bestehende Funktionalität kaputt zu machen (Regression). Es hat sich gezeigt, dass wir dadurch auch in der Lage waren, schnell auf das neue Trusted Tester API (siehe Abschnitt 3.4) zu wechseln, da wir nur sicherstellen mussten, dass unsere Tests wieder fehlerfrei durchlaufen. Dabei haben wir sogar einen Bug der Schnittstelle entdeckt, welchen wir Google gemeldet haben (siehe Abschnitt 3.1.5).

QUnit hat sich als sehr wertvoll erwiesen bei den sogenannten asynchronen Tests. Obwohl unsere Bibliothek sehr stark auf die asynchronen Aufrufe des Google APIs angewiesen ist, konnten wir trotzdem normal Tests schreiben ohne uns speziell um diesen Aspekt kümmern zu müssen.

Wenn wir einen Bug gefunden haben, haben wir diesen jeweils mit einem Unit-Test nachvollzogen, so dass dieser nicht noch ein weiteres Mal auftritt. Die Testabdeckung haben wir mit JSCoverage[26] gemessen, für die GftLib beträgt diese 95%, beim SqlBuilder sogar 100% (siehe Abbildung 3.9). Bei den fehlenden 5% handelt es sich um die Fehlerbehandlung der asynchronen Aufrufe, welcher sich nicht so einfach testen lässt.

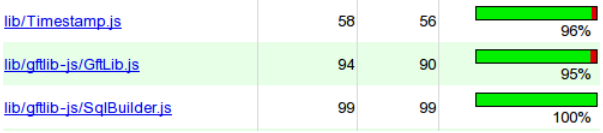


Abbildung 3.9.: Gemessene Testabdeckung mit JSCoverage

Private Methoden sind in JavaScript grundsätzlich nicht von aussen zugreifbar. Da sich ein Grossteil der Funktionalität in solchen privaten Methoden befindet, haben wir öffentliche Aliase erstellt, um Tests dieser Methoden zu ermöglichen. Die Aliase sind jeweils

[26] http://siliconforks.com/jscoverage/

mit zwei Unterstrichen gekennzeichnet; sie sind nicht teil des öffentlichen APIs und sollten nur von von Unit-Tests verwendet werden.

3.8. Authentifizierung mit OAuth

OAuth ist das Mittel der Wahl, wenn es um die Authentifizierung in Zusammenhang mit Google Fusion Tables geht. OAuth wird für den Zugriff auf private Tabellen und den schreibenden Zugriff verwendet. Das Prinzip ist sehr einfach: Eine Applikation möchte Zugriff auf die Tabellen eines Benutzers. Der Benutzer möchte aber nicht seinen Benutzernamen und sein Passwort dafür angeben, da die Applikation damit grundsätzlich Zugriff auf sein ganzes Google-Konto hätte. Es gilt also einerseits den Zugriff für die spezifische Applikation einzuschränken und gleichzeitig auch eine Authentifizierung dafür möglich zu machen.

Bei OAuth sind 3 Parteien involviert:

- Der Benutzer, welcher eine Applikation nutzen möchte
- Die Applikation die Zugriff auf eine Ressource des Benutzers haben will
- Der Anbieter, welcher Ressourcen und Authentifizierung bietet (*Service Provider*)

Sowohl der Benutzer wie auch die Applikation müssen dem Service Provider vertrauen.

3.8.1. Ablauf der Authentifizierung

Wenn der Benutzer das erste mal die gewünschte Applikation nutzen will, stellt diese fest, dass der Benutzer noch nicht authentifiziert ist. Zu diesem Zweck leitet die Applikation den Benutzer weiter zum Service Provider. Auf der Seite des Service Providers gibt der Benutzer seinen Benutzernamen und sein Passwort ein. Daraus wird ein sogenanntes *Zugriffs-Token* generiert. Dieses Token wird anschliessend der Applikation und dem Benutzer mitgeteilt, fortan dient es dazu den Benutzer zu authentifizieren.

Die Applikation kann nun im festgelegten Rahmen direkt mit dem Token beim Service Provider auf die geschützten Ressourcen zugreifen.

In der Abbildung 3.10 wird dieser Ablauf nochmals visualisiert.

3.8.2. OAuth Szenarien

Es gibt grundsätzlich verschiedene Methoden bei OAuth, je nachdem welches Ziel eine Applikation verfolgt. Auch wenn sich der in Abschnitt 3.8.1 beschriebene Ablauf immer ähnelt, gibt es Unterschiede im Detail.

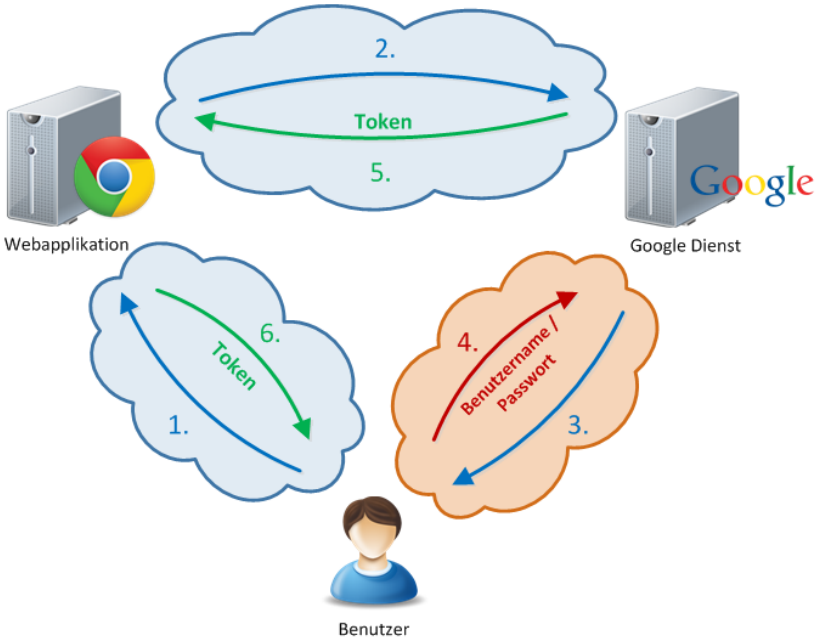


Abbildung 3.10.: Endbenutzer authentifiziert sich mit OAuth

Clientseitiges OAuth

Das clientseitige OAuth entspricht dem im Abschnitt 3.8.1 beschriebenen Ablauf. Ein Benutzer ist aktiv, führt die Authentifizierung durch und überlässt dann die weiteren Zugriffe der Applikation.

Serverseitiges OAuth

Beim serverseitigen OAuth spielt der Endbenutzer eine untergeordnete Rolle, da er an der Authentifizierung nicht beteiligt ist. Hier geht es darum, dass sich eine Applikation gegenüber einem Service Provider authentifiziert, um auf ihre eigene Dienste zuzugreifen. Dies ist z.B. dann der Fall wenn eine Applikation als Datenbank eine Cloud-Datenbank verwendet. Dieser Ablauf wird in der Abbildung 3.11 visualisiert.

3.8.3. OAuth in Google Fusion Tables

Wir standen vor der Herausforderung die Authentifizierung für die GFT nutzbar zu machen. Für den Zugriff auf Tabellen eines Benutzers funktionierte dies ausgezeichnet, wir haben dies mit einer Beispielapplikation ausprobiert. Der Ablauf dafür ist bereits ausführlich dokumentiert[27], so dass wenn die Konzepte klar sind mit Hilfe einer Client Library[28] sehr einfach eine Authentisierung erstellt werden kann.

[27] https://developers.google.com/fusiontables/docs/articles/oauthfusiontables
[28] https://developers.google.com/accounts/docs/OAuth2#libraries

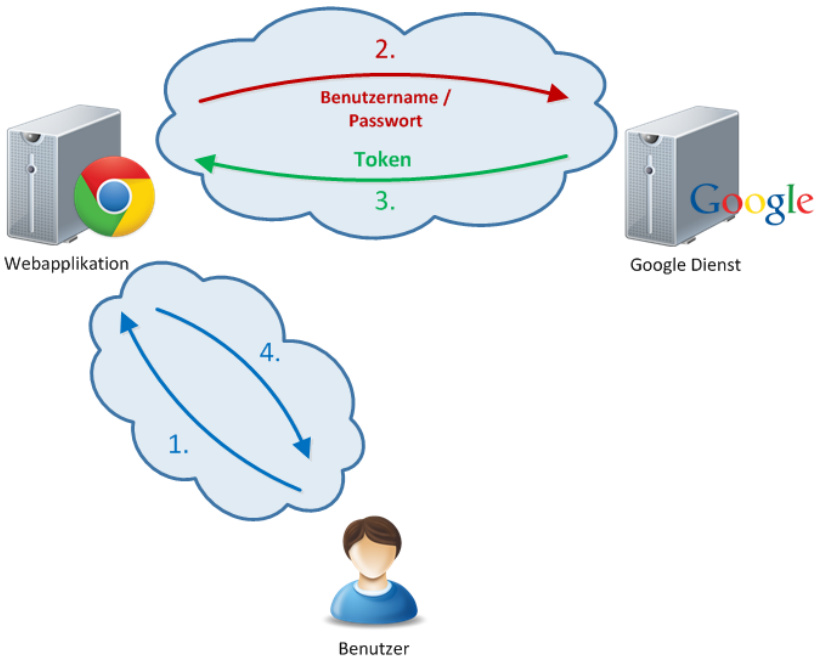


Abbildung 3.11.: Applikation verwendet einen Service Account zur Authentisierung

FixMyStreet Use Case

Für den Use Case FixMyStreet (siehe Kapitel 7) waren wir jedoch vor eine neue Hürde gestellt: Wir haben GFT explizit als Applikationsdatenbank ausgelegt, so dass die Tabelle eines Benutzers alle Daten der Applikation hält. Die Applikation soll für den Endbenutzer ohne Authentifizierung benutzbar sein.

Google stellt für diesen Fall seit März 2012 sogenannte *Service Accounts*[29] zur Verfügung. Sie sind ein Beispiel für das serverseitige OAuth (siehe Abschnitt 3.8.2).

Die Dokumentation dieses neuen Features ist leider so gut wie inexistent. Gerade auch für Google Fusion Tables fehlt ein entsprechender Service in den Client Libraries.

Die Idee bei den Service Accounts ist, dass dieser spezielle Account sich beim Service Provider anmeldet und zwar mit Hilfe eines verschlüsselten Requests. Beim erstellen des Accounts wird ein Schlüsselpaar generiert. Mit dem privaten Schlüssel können dann verschlüsselte Requests erstellt werden.

OAuth in der GftLib

Da sich für die Service Accounts die Token nur via Server erstellen lassen, haben wir einen Web Service erstellt, welcher jeweils ein gültiges Token über eine JSONP-Schnittstelle zur Verfügung stellt. Mit dem so erworbenen Token, kann sich die JavaScript Bibliothek

[29] Google Developer Blog - Service Accounts have arrived: http://googledevelopers.blogspot.com/2012/03/service-accounts-have-arrived.html

GftLib bei Google Fusion Tables authentifizieren. Durch die Berechtigungen auf einer Tabelle bzw. durch den Einsatz von Views können die Möglichkeiten, welches das Token bietet, eingeschränkt werden.

4. Infrastruktur

4.1. Server in der Cloud mit Amazon EC2

Die Prämisse war von Anfang an klar: möglichst alle Dienste welche wir für diese Studienarbeit brauchen, sollen aus der *Cloud* kommen. Da liegt es natürlich Nahe, dies auch auf die Infrastruktur anzuwenden. Dank dem Amazon Free Usage Tier[1] konnten wir kostenlos Server bei Amazon EC2 aufsetzen.

Gemäss dem Cloud-Ansatz, sind Ressourcen relativ kostengünstig, weshalb wir für jede Aufgabe einen separaten Server verwendet haben. So haben wir auch die Kopplung reduziert und können einzelne Dienste sehr einfach austauschen.

Insgesamt haben wir 3 Server verwendet:

- Build- und Deploymentserver: Build und Hosting der Sourcen
- PhantomJS-Server: Ausführung der JavaScript-Tests
- Redmine-Server: Hosting der Projektmanagement-Software

Den Build- und Deploymentserver und den PhantomJS-Server haben wir selbst aufgesetzt. Den Redmine-Server mussten wir lediglich konfigurieren. Dies weil wir dafür auf einem bestehenden Image aufsetzen konnten, auf welchem Redmine schon installiert war. Das Open-Source Projekt BitNami[2] erstellt für zahlreiche Projekte solche Images. Dadurch entfällt ein Grossteil des Installationsaufwandes.

4.2. Unit-Testing mit PhantomJS und QUnit

Da wir hauptsächlich JavaScript als Programmiersprache verwendet haben, stellte sich bald einmal die Frage, wie sich unser Code sinnvoll testen lässt. Da wir bereits aus einem anderen Projekt sehr gute Erfahrungen mit QUnit[3] gesammelt haben, wollten wir darauf aufbauen und haben damit unseren Code geprüft.

Da es sich auch bei unserem Testcode um JavaScript handelt, sind wir immer auf einen

[1] http://aws.amazon.com/free/
[2] http://bitnami.org/
[3] QUnit ist ein Unit Testing Framework von jQuery: http://docs.jquery.com/QUnit

Browser angewiesen, welcher entsprechend die Tests ausführt. Für einen automatisierten Build ist dies unzureichend, da dieser nicht direkt die Tests ausführen kann.

Nach einigem Suchen sind wir schliesslich auf das Projekt PhantomJS[4] aufmerksam geworden. Dabei handelt es sich um einen headless Browser, welcher ohne grafische Darstellung auskommt. Damit konnten wir dann die URL unserer Tests[5] ansteuern und das Resultat in Jenkins auswerten[6].

Es hat sich dann noch als Schwierigkeit ergeben, dass wir den Output von QUnit in eine für Jenkins verständliche Form bringen mussten. Dazu haben wir den Testrunner `run_qunit.js` erweitert und mit zusätzlichen Ausgabemöglichkeiten ausgestattet. Alle nötigen Schritte haben wir schliesslich als Anleitung in einem Blogpost[7] veröffentlicht. Wir haben auch schon Feedback[8] dazu erhalten und konnten so unsere Lösung weiter verbessern.

4.3. Continuous Integration mit dem Jenkins Build-Server

Bei Jenkins CI[9] handelt es sich um ein Open Source Projekt, welches Unterstützung bietet für die Projektautomatisierung. Es unterstützt zahlreiche Repositories und Build-Mechanismen. Wir haben auch einige Plugins aktiviert, u.a. die GitHub- und Redmine-Integration oder der Logfile-Parser.

4.3.1. Logfile-Analyse mit dem Log Parser Plugin

Das Log Parser Plugin[10] ermöglicht es auf einfache Art und Weise Probleme mit dem Build festzustellen. Für den Build kann es zum Teil sehr schwierig sein, den Status des Builds zu bestimmen. Der Build kann erkennen ob ein aufgerufener Befehl fehlgeschlagen hat oder ob die ausgeführten Tests erfolgreich durchlaufen. Wenn jedoch ein aufgerufenes Programm eine Warnung oder einen Fehler anzeigt, kann der Build das nicht selbst erkennen. Hier kommt der Logparser ins Spiel. Man kann Muster definieren für Dinge die auf eine Warnung oder einen Fehler hindeuten. Diese Logik erlaubt es dem Build viel detailliertere Aussagen über die laufende Installation zu machen.

[4] http://phantomjs.org/
[5] http://gft.rdmr.ch/test/js/
[6] http://phantomjs.rdmr.ch/gftprototype.php
[7] http://www.readmore.ch/post/18940470535
[8] https://github.com/odi86/GFTPrototype/issues/1
[9] http://jenkins-ci.org/
[10] http://wiki.hudson-ci.org/display/HUDSON/Log+Parser+Plugin

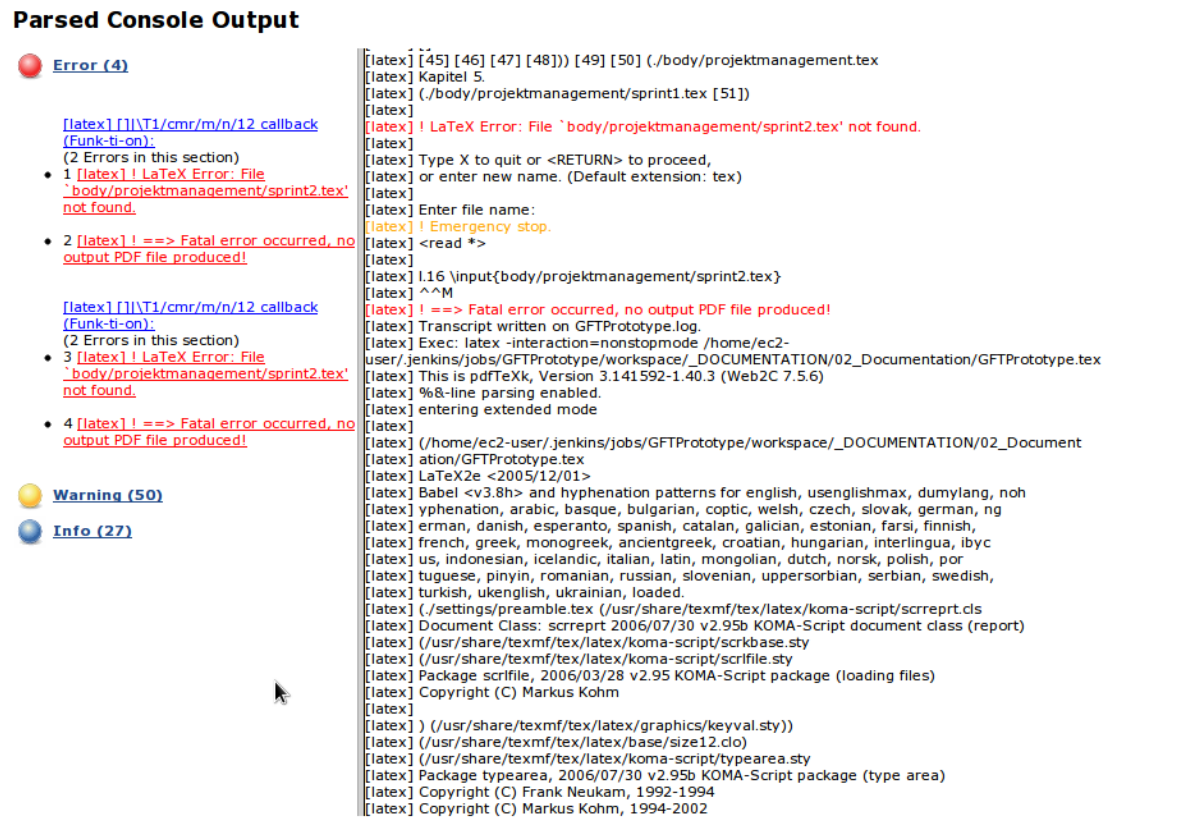


Abbildung 4.1.: Ansicht des Logfiles mit dem Log Parser Plugin

4.3.2. Trigger (Auslöser)

Wir haben unseren Buildserver so getriggert, dass dieser ausgelöst wird, wenn eine Änderung auf GitHub gepusht wurde. Dazu haben wir das GitHub Plugin[11] auf Jenkins installiert, welches sich einfach im Backend installieren lässt. Im GitHub-Repository mussten wir dann noch sogenannte Web Hooks[12] einrichten, so dass GitHub jeweils Jenkins benachrichtigt, sobald Änderungen eintreffen.

4.3.3. Build Steuerung mit Ant

Zur Build-Steuerung verwenden wir Ant[13]. Ant ist vor allem aus der Java-Welt bekannt, lässt sich aber auch gut in andere Umgebungen einbinden. Der Build wir in einer XML-Datei spezifiziert. Einzelne Schritte darin werden *Ant Target* genannt und diese sollten grundsätzlich von einander unabhängig sein. Dies ermöglicht es auch nur Teilschritte

[11] https://wiki.jenkins-ci.org/display/JENKINS/Github+Plugin
[12] http://help.github.com/post-receive-hooks/
[13] http://ant.apache.org/

eines Builds auszuführen. Es lassen sich aber auch Abhängigkeiten zwischen Targets definieren, welche dann der Reihe nach abgearbeitet werden.

Hier zum Beispiel unser `build`-Target, welches alle anderen Targets als Abhängigkeit hat:

```
<target name="build" depends="test,documentation,generate_css,deploy,
    test-js">
  <echo message="BUILD FINISHED!"/>
</target>
```

4.3.4. Testausführung

Zur Darstellung der Testresultate bauen wir auf dem JUnit XML-Interpreter auf, welcher in Jenkins bereits integriert ist. Dazu ist es lediglich notwendig alle Testresultate in das JUnit XML-Format zu bringen, die ganzen Auswertungen, Grafiken etc. bekommt man dann quasi „gratis“ dazu.

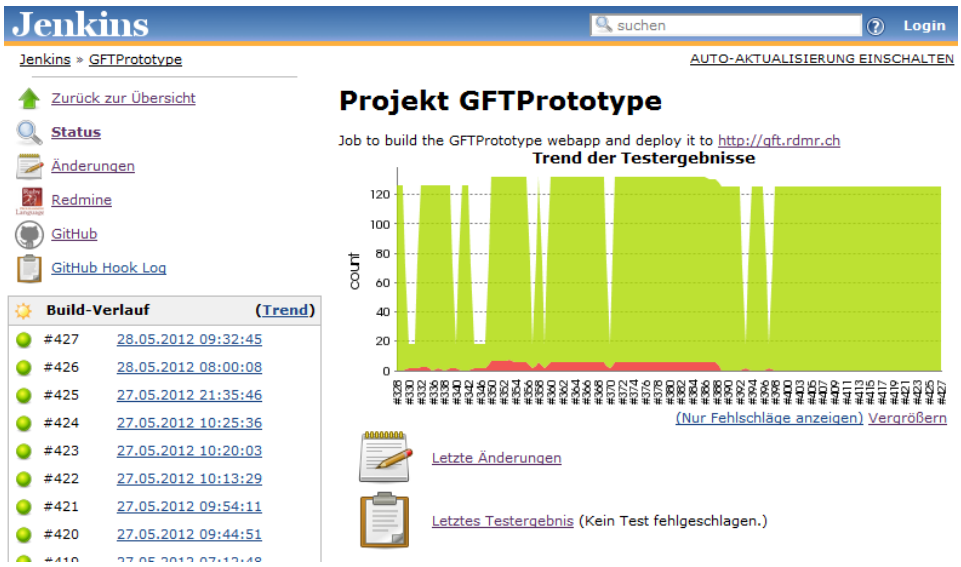


Abbildung 4.2.: Diagramm der Testresultate im Verlaufe der Zeit

Teil III.

Implementation

5. Beispielapplikationen

Um die verschiedenen Features der Google Fusion Tables kennenzulernen, erstellten wir zu Beginn der Arbeit einige kleine Beispielapplikationen, welche diese verwenden.

Eine Übersicht über alle erstellen Beispiele findet man auf folgender Übersichtsseite: http://gft.rdmr.ch/

Daten ausgeben

URL:	http://gft.rdmr.ch/examples/js/data-print/
Beschreibung:	Daten mittels GftLib von FusionTable selektieren und textuell ausgeben.

Daten selektieren

URL:	http://gft.rdmr.ch/examples/js/data-select/
Beschreibung:	Queries eingeben und Resultat tabellarisch darstellen

Daten selektieren mit ortsbezogener Einschränkung

URL:	http://gft.rdmr.ch/examples/js/spatialquery-condition/
Beschreibung:	Daten mittels GftLib von FusionTable selektieren und textuell ausgeben. Die Daten werden zusätzlich mit dem Spatial-Query `ST_INTERSECTS` (siehe Abschnitt 3.2.2) eingeschränkt.

Daten selektieren mit ortsbezogener Sortierung

URL:	http://gft.rdmr.ch/examples/js/spatialquery-order/
Beschreibung:	Daten mittels GftLib von FusionTable selektieren und textuell ausgeben. Die zurückgelieferten Daten werden bei der Abfrage mit dem Spatial-Query `ST_DISTANCE` (siehe Abschnitt 3.2.2) sortiert.

Google Maps: Daten auf Karte anzeigen

URL: http://gft.rdmr.ch/examples/js/gmap-rawdata/

Beschreibung: Daten mittels GftLib von FusionTable selektieren und auf Karte anzeigen.

Google Maps: Daten auf Karte anzeigen mit manueller Geocodierung

URL: http://gft.rdmr.ch/examples/js/gmap-geocoding/

Beschreibung: Daten mittels GftLib von FusionTable selektieren und auf Karte anzeigen. Um die Markierungen zu positionieren wird der Geocoding-Service des Google Maps API verwendet.

Google Maps: Fusion Table-Ebene

URL: http://gft.rdmr.ch/examples/js/gmap-fusiontableslayer/

Beschreibung: Anzeigen der Daten aus zwei FusionTables via FusionTablesLayer.

Google Maps: Fusion Table-Ebene Stile

URL: http://gft.rdmr.ch/examples/js/gmap-fusiontableslayer-clickstyle/

Beschreibung: Anzeigen der Daten aus einer FusionTable via FusionTablesLayer mit manuell konfiguriertem Stil (siehe Abschnitt 3.6.1). Sobald auf eine angezeigte Fläche geklickt wird, ändert sich deren Farbe.

Google Maps: Dynamische Fusion Table-Ebene

URL: http://gft.rdmr.ch/examples/js/gmap-dynamic-fusiontableslayer/

Beschreibung: Anzeigen der Daten aus zwei FusionTables via FusionTablesLayer. Per Slider lassen sich zusätzlich die zurückgegebenen Daten nach der Anzahl an Einwohnern einschränken.

Google Maps: Fusion Table-Ebene mit ortsbezogener Einschränkung

URL:	http://gft.rdmr.ch/examples/js/gmap-spatialquery/
Beschreibung:	Anzeigen der Daten aus einer FusionTable via FusionTablesLayer. Es werden nur diejenigen Daten angezeigt, welche im Radius der auf der Karte positionierten Markierung liegen. Die Markierung lässt sich per Drag&Drop auf der Karte verschieben. Zusätzlich lässt sich der Radius per Slider definieren.

Google Charts: Daten mit Diagrammen visualisieren

URL:	http://gft.rdmr.ch/examples/js/gchart-fusiontable/
Beschreibung:	Visualisieren der Daten aus einer FusionTable mit Diagrammen. Dazu wird das Google Chart Tool API verwendet.

Einfügen von Daten

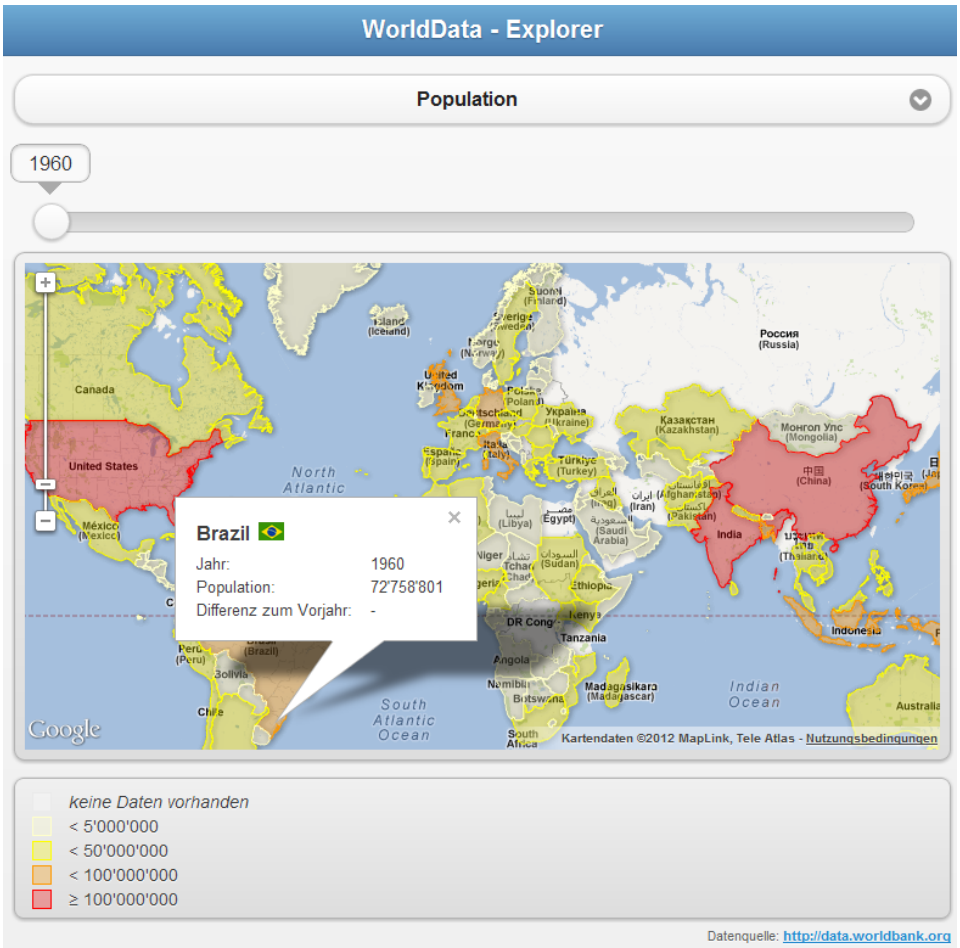
URL:	http://gft.rdmr.ch/examples/js/oauth-login/
Beschreibung:	Einfügen von Daten in eine FusionTable. Dazu muss zuerst ein Zugriffs-Token via OAuth angefordert werden. *Hinweis: Dieses Beispiel funktioniert nur mit einem Google Account.*

Verwendung einer Fusion Table-Ebene in Sencha Touch 2

URL:	http://gft.rdmr.ch/examples/js/senchatouch-fusiontableslayer/
Beschreibung:	Anzeige einer FusionTablesLayer in einer Sencha Touch 2 Applikation.

6. Use Case 1: WorldData

http://worlddata.rdmr.ch/



6.1. Einführung

Im ersten Use Case geht es hauptsächlich um die Anzeige grosser Datenmengen auf der Karte. Dazu importieren wir bestehende Datenbestände in die Google Fusion Tables und visualisieren diese mittels FusionTablesLayer des Google Maps API (siehe Abschnitt 3.6) auf der Karte.

6.1.1. Ziel

Es sollen verschiedene historische Länderdaten auf einer Weltkarte angezeigt werden. Die Daten sind pro Jahr und Thema unterteilt. Über eine Zeitachse soll es möglich sein die Daten der verschiedenen Jahre zu selektieren. Eine solche Darstellung kann beispielsweise dabei helfen Zusammenhänge zwischen verschiedenen Themenbereichen zu finden.

6.1.2. Vorgehen

Um die Daten pro Land zu visualisieren, wurden zuerst die Landesgrenzen als Geometrie-Datensätze in eine Fusion Table importiert. In einer anderen Tabelle wurden dann die historischen Daten, unterteilt nach Land und Jahr, geladen. Aus diese beiden Tabellen entstand dann per Merge-Funktion (siehe Abschnitt 3.1.2) eine sogenannte Merged Table, welche die Daten der beiden zugrundeliegenden Tabellen kombiniert. Diese neue Tabelle konnte dann mittels FusionTablesLayer auf der Karte dargestellt werden.

Die Idee des Use Cases stammt von der Webapplikation *Public Data*[1] von Google. Darin lassen sich bereits öffentliche Daten aus verschiedensten Quellen über das Google Chart API[2] in verschiedenen Diagrammen darstellen.

6.2. Analyse

6.2.1. Datenquellen

Als Datenquelle für diesen Use Case wurde der Datenkatalog der Weltbank[3] verwendet. Darin finden sich länderspezifische Daten aufgeteilt in über 7000 Themenbereiche. Diese lassen sich als XML- oder Excel-Datei herunterladen. Für unseren Import verwendeten wir die Excel-Dateien, welche wir vorbereitet und als CSV-Dateien gespeichert haben. Diese konnten wir anschliessend in Google Fusion Tables importieren.

Die Landesgrenzen wurden als KML-Datei von einer inoffiziellen Google Earth Library-

[1] http://www.google.ch/publicdata/
[2] https://developers.google.com/chart/
[3] http://data.worldbank.org/

Webseite[4] bezogen.

6.2.2. UseCases

Folgende Anwendungsfälle sollen von der Applikation abgedeckt werden.

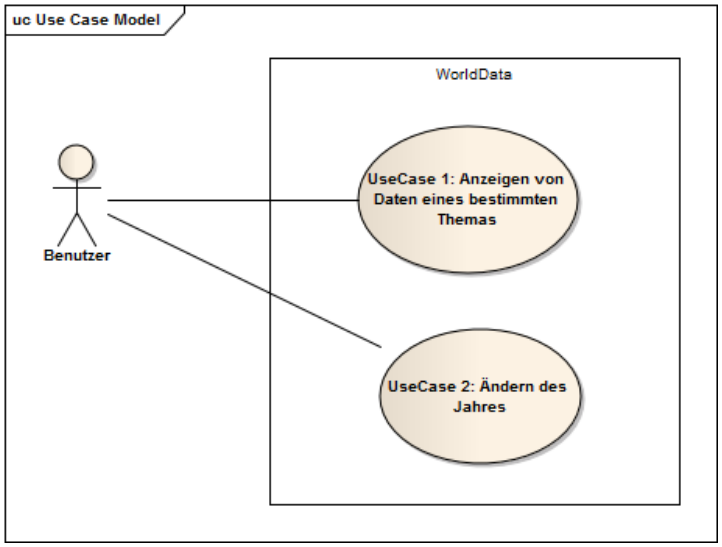


Abbildung 6.1.: WorldData: UseCase Modell

Use Case 1: Anzeigen von Daten eines bestimmten Themas

Primary Actor	Benutzer
Stakeholders and Interests	Benutzer: Möchte Daten zu einem bestimmten Thema auf der Karte anzeigen
Preconditions	Webapplikation ist gestartet
Success Guarantee (Postconditions)	• Daten des gewählten Themas werden auf der Karte angezeigt • Legende zu den Daten wird angezeigt
Main Success Scenario	1. Benutzer wählt das Thema aus
Frequency of Occurrence	Tritt sehr häufig auf, da eine beliebige Anzahl von Benutzern die Webapplikation bedienen können

[4] http://www.gelib.com/world-borders.htm

Use Case 2: Ändern des Jahres

Primary Actor	Benutzer
Stakeholders and Interests	Benutzer: Möchte Daten eines anderen Jahres anzeigen
Preconditions	• Webapplikation ist gestartet • Thema ist ausgewählt
Success Guarantee (Postconditions)	• Daten des gewählten Jahres werden auf der Karte angezeigt
Main Success Scenario	1. Benutzer wählt das gewünschte Jahr aus
Alternative Flows	1a. Falls für das gewählte Jahr keine Daten vorhanden sind, ist dies ebenfalls auf der Karte ersichtlich
Frequency of Occurrence	Tritt sehr häufig auf, da eine beliebige Anzahl von Benutzern die Webapplikation bedienen können

6.3. Design

6.3.1. Datenbankschema

Wie bereits im Abschnitt 6.1.2 beschrieben, verwendet die Applikation die Daten der Merged Table *MERGE-ftWorldBorders_ftWorldData* (siehe Abschnitt 3.1.2). Diese beinhaltet pro Thema und Land eine Zeile mit Daten. Diese Daten sind wiederum pro Jahr in einer Spalte abgelegt. Dieses Schema ist 1:1 von den Daten der Weltbank übernommen. Durch einen `JOIN` über den Namen des Landes, werden die Geometriedaten dazugenommen.

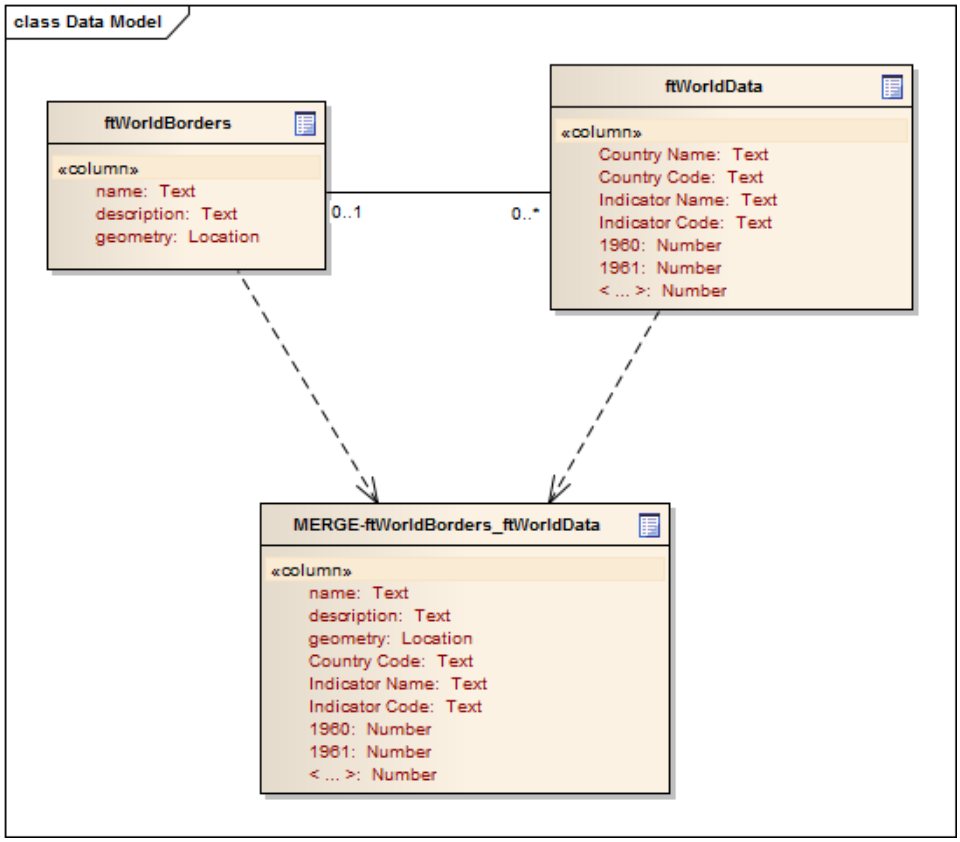


Abbildung 6.2.: WorldData: Datenbankschema

6.3.2. Aufbau der Applikation

Die Applikation hat eine *View* welche die Anzeige der Daten regelt. Diese wiederum wird von einem *Controller* gesteuert. Beide verwenden Konfigurationsparameter, welche in der *Config*-Klasse gesetzt werden können. Zusätzlich verwenden beide Klassen Helfermethoden aus der *Helper*-Klasse.

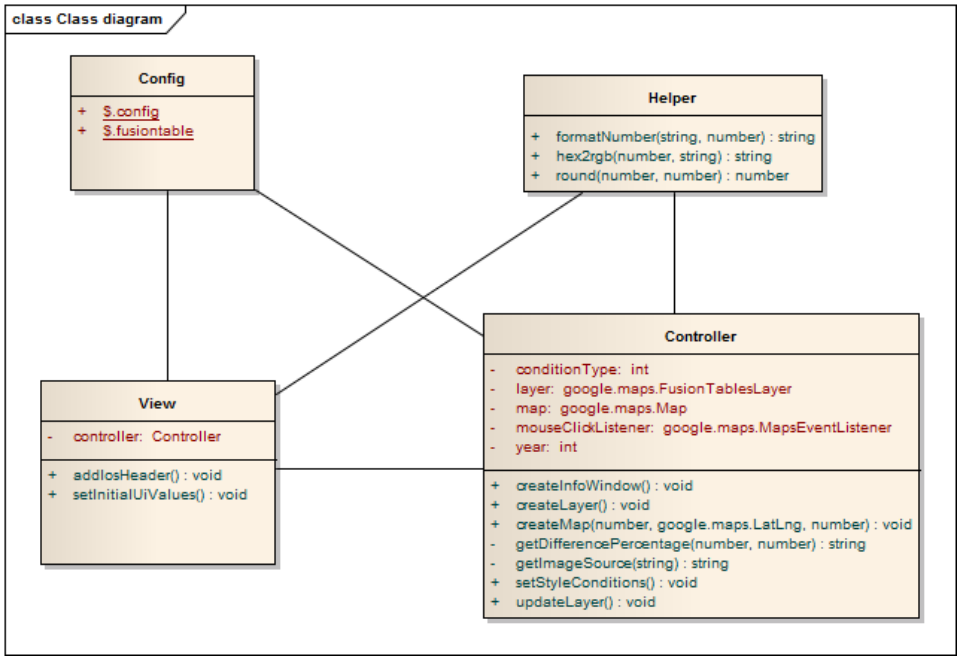


Abbildung 6.3.: WorldData: Klassendiagramm

6.4. Implementation

Die Applikation wurde als Webapplikation implementiert. Für die GUI-Elemente wurde das Javascript Framework jQuery Mobile[5] verwendet. Dieses bietet eine sehr breite Plattform-Unterstützung.

6.4.1. Systemanforderungen

Da es sich um eine Webapplikation handelt, ist es möglich, die Applikation auf beinahe allen Geräten, welche über einem Browser verfügen, zu starten. Einschränkungen gibt es nur in den unterstützten Browsern. Eine Liste davon findet sich auf der Webseite des jQuery Mobile Frameworks: http://jquerymobile.com/blog/2012/04/06/jquery-mobile-1-1-0-rc2/#platforms

[5] http://jquerymobile.com/

6.4.2. Abhängigkeiten

Library	Version	Verwendung
jQuery Mobile	1.1.0	GUI-Elemente
jQuery	1.7.1	Basis für den Gebrauch von jQuery Mobile
Google Maps API	V3	Karte mit FusionTablesLayer
Cubiq - Add to home screen	2.0	Popup welches auf allen iOS Geräten erscheint

Tabelle 6.3.: WorldData: Abhängigkeiten

6.4.3. Quellcode-Struktur

Datei	Beschreibung
`images/`	Bilder der Applikation
`js/`	Hier befindet sich die eigentliche Implementation der Applikation
`js/Config.js`	Konfiguration der Applikation
`js/Controller.js`	Controller der Applikation
`js/Helper.js`	Helper-Funktionen welche von der Applikation verwendet werden
`js/View.js`	Steuert die Anzeige der Applikation
`lib/`	Von der Applikation verwendete Libraries
`styles/`	CSS-Styles der Applikation
`index.html`	Startseite der Applikation

Tabelle 6.4.: WorldData: Quellcode-Struktur

6.5. Testing

Die Applikationslogik dieses Use Cases befindet sich im Controller und zu kleinen Teilen im Helper. Für diese beiden Komponenten haben wir insgesamt zwölf Tests erstellt[6]. Zusätzlich gibt es noch zwei Tests für die Konfiguration, diese dienen vor allem dazu, Änderungen der Konfiguration zu erkennen.

[6] Ausführbare Tests: http://gft.rdmr.ch/test/js/?filter=WorldData

Die Anzeige der Daten basiert auf dem FusionTablesLayer des Google Maps API, weshalb wir dort keine eigene Logik haben und diese somit auch nicht testen müssen.

6.6. Resultate

Das Resultat dieses Use Cases ist eine Webapplikation, welche es dem Benutzer erlaubt länderspezifische Daten zu verschiedenen Themen auf der Karte darzustellen.

6.6.1. Features

- Auswahl des Themas über Selectbox
- Wahl des Jahres über Slider
- Weitere Informationen zu den angezeigten Daten per Klick auf das gewünschte Land
- Applikation ist dank Verwendung des jQuery Mobile Frameworks ebenfalls auch für Mobilgeräte optimiert

6.6.2. Screenshots

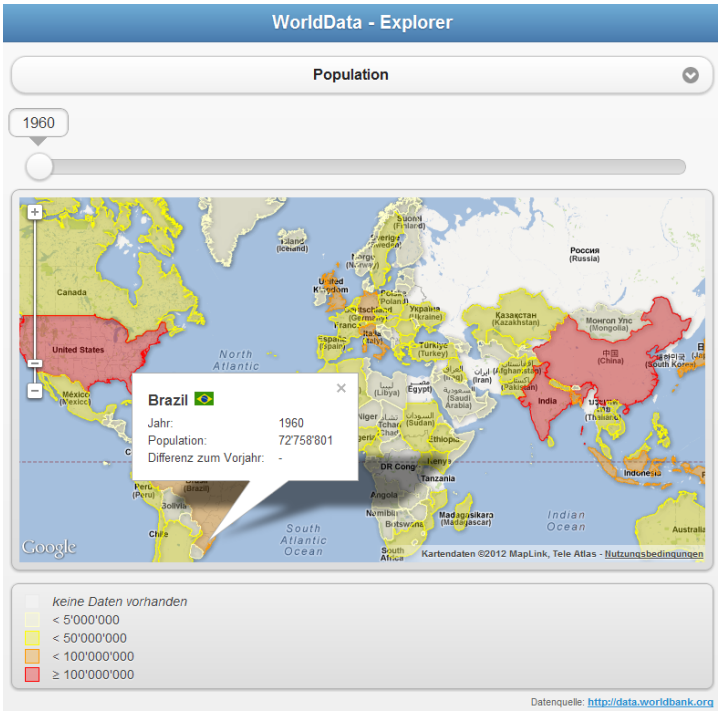


Abbildung 6.4.: WorldData: Population im Jahr 1960

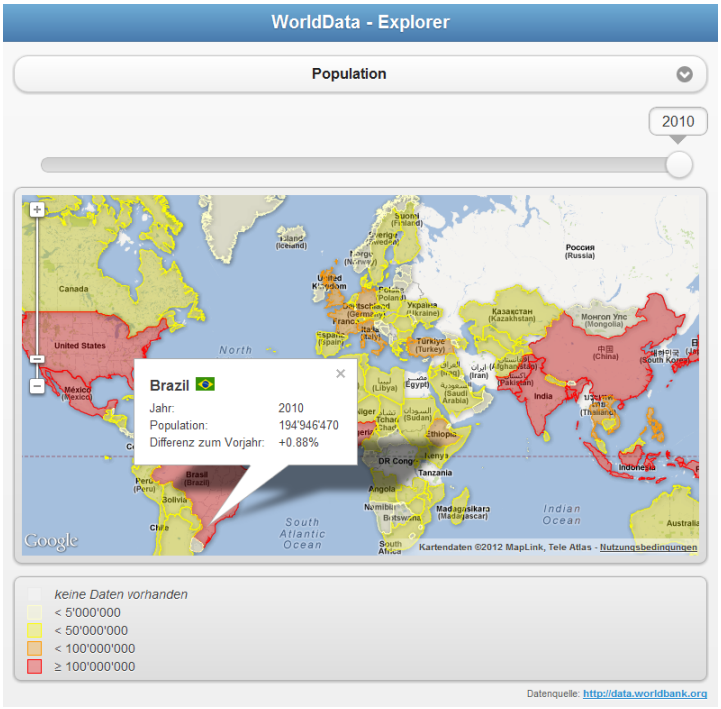


Abbildung 6.5.: WorldData: Population im Jahr 2010

6.6.3. Erkenntnisse zur Fusion Table-Ebene

Die Fusion Table-Ebene von Google Maps API eignet sich sehr gut zur Darstellung einer grossen Anzahl von Daten. Leider hindern die Einschränkungen betreffend Styles (siehe Abschnitt 3.6.1) die individuelle Gestaltung der Ebene stark.

Wie wir feststellen mussten, sind diese 5 Stile sehr schnell aufgebraucht. Einer davon fällt meistens für den Standard-Stil weg. Dieser wird angewendet, wenn die entsprechende Zeile aus der Tabelle auf keine Bedingung der restlichen Stile passt. So bleiben lediglich noch 4 Stile übrig für alle Elemente die man speziell hervorheben möchte.

Sobald man dann auch noch verschiedene Geometrie-Typen in der Tabelle abgelegt hat (Punkt, Linie, Fläche), muss man sich gut überlegen, für welche Elemente man wirklich einen speziellen Stil anwenden will.

6.7. Weiterentwicklung

Da es sich bei der Applikation lediglich um einen Prototypen handelt, bietet diese natürlich noch ein grosses Potential zur Weiterentwicklung. Hier eine Auflistung möglicher Features, welche noch implementiert werden könnten:

- Weitere Themen in die Datenbank importieren

- Auswahl der Länder für welche Daten angezeigt werden sollen
- Anzeige einer Rangliste der 10 Länder mit den höchsten Werten des gewählten Themas
- Direkter Vergleich verschiedener Länder

6.8. Benutzerdokumentation

6.8.1. Importieren der Daten in Google Fusion Tables

Landesgrenzen

Die Landesgrenzen liegen als KML-Datei vor. Diese beinhaltet alle Länder mit ihren Grenzen definiert als Polygone.

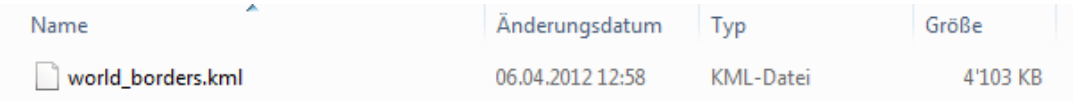
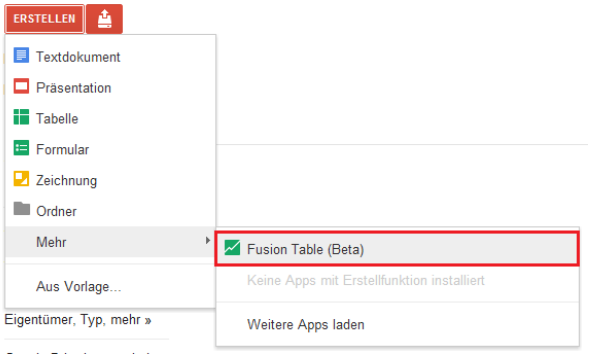


Abbildung 6.6.: WorldData: Landesgrenzen liegen als KML-Datei vor

1. Google Drive[7] öffnen und sich mit seinem Google Account einloggen
2. Erstellen > Mehr > Fusion Table (Beta)



3. Es öffnet sich die neue Fusion Table mit dem Dialog zum Importieren von bestehenden Daten
4. Im Tab *From this computer* die lokal gespeicherte Datei auswählen > Next
5. Die Daten werden automatisch in passende Spalten eingeteilt
6. Abschliessend muss der Tabelle noch einen Namen gegeben werden
7. Mit *Finish* werden die Daten dann importiert

[7] https://drive.google.com/

Die Tabelle sollte nun folgendermassen aussehen:

Abbildung 6.7.: WorldData: Landesgrenzen erfolgreich als Fusion Table importiert

Daten

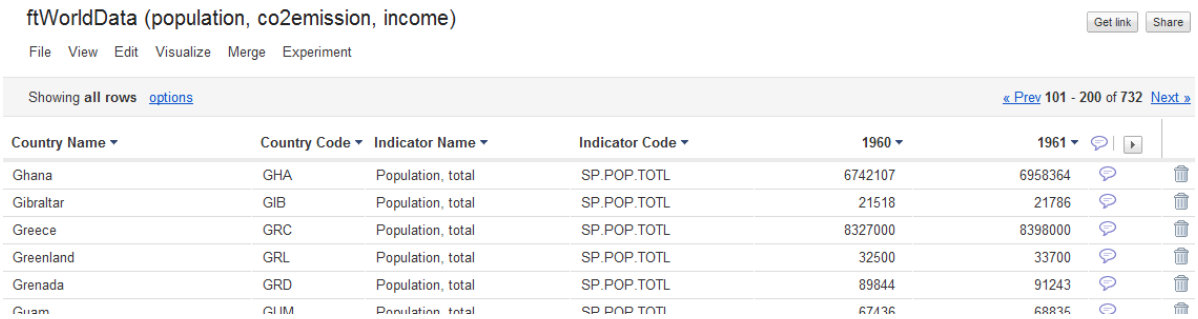
Die Daten liegen als CSV-Datei vor. Diese beinhaltet folgende Spalten:

- Country Name
- Country Code
- Indicator Name
- Indicator Code
- 1960
- 1961
- ...

Abbildung 6.8.: WorldData: Daten liegen als CSV-Datei vor

Das Vorgehen für den Import der Daten ist dasselbe wie bei den Landesgrenzen (siehe Abschnitt 6.8.1).

Nach dem Import der Daten sollte die Tabelle folgendermassen aussehen:


ftWorldData (population, co2emission, income)

Get link Share

File View Edit Visualize Merge Experiment

Showing all rows options « Prev 101 - 200 of 732 Next »

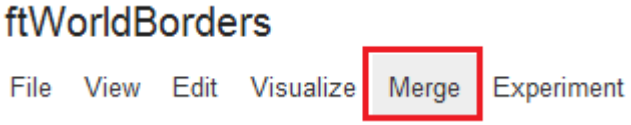
Country Name	Country Code	Indicator Name	Indicator Code	1960	1961
Ghana	GHA	Population, total	SP.POP.TOTL	6742107	6958364
Gibraltar	GIB	Population, total	SP.POP.TOTL	21518	21786
Greece	GRC	Population, total	SP.POP.TOTL	8327000	8398000
Greenland	GRL	Population, total	SP.POP.TOTL	32500	33700
Grenada	GRD	Population, total	SP.POP.TOTL	89844	91243

Abbildung 6.9.: WorldData: Daten erfolgreich als Fusion Table importiert

Tabellen mergen

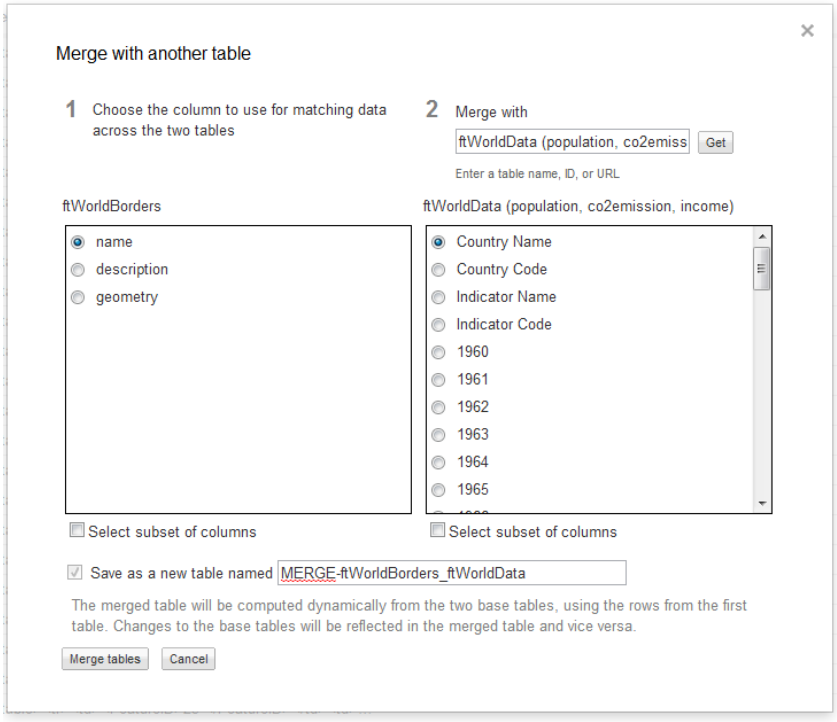
Sind beide Fusion Tables erstellt müssen diese zusammengefügt werden, um sie schlussendlich als einen einzelnen Layer auf der Karte darzustellen. Dazu bietet Google Fusion Tables die *Merge*-Funktion an.

1. Die Tabelle mit den Landesgrenzen öffnen
2. Im Menü *Merge* auswählen


ftWorldBorders

File View Edit Visualize Merge Experiment

3. Es öffnet sich ein Popup, welches durch den Vorgang führt
4. In der 1. Spalte wählt man die Spalte aus, über welche die beiden Tabellen verbunden werden sollen. In unserem Fall ist dies die Spalte mit den Namen der Länder.
5. In der 2. Spalte muss man zuerst die andere Tabelle auswählen und dann ebenfalls die Spalte, in welcher die Namen der Länder gespeichert sind.
6. Schlussendlich muss der neuen Tabelle ein Namen gegeben werden
7. Mit einem Klick auf *Merge tables* wird die neue Tabelle mit den zusammengefügten Daten erstellt.



Die zusammengeführte Tabelle sollte nun folgendermassen aussehen:

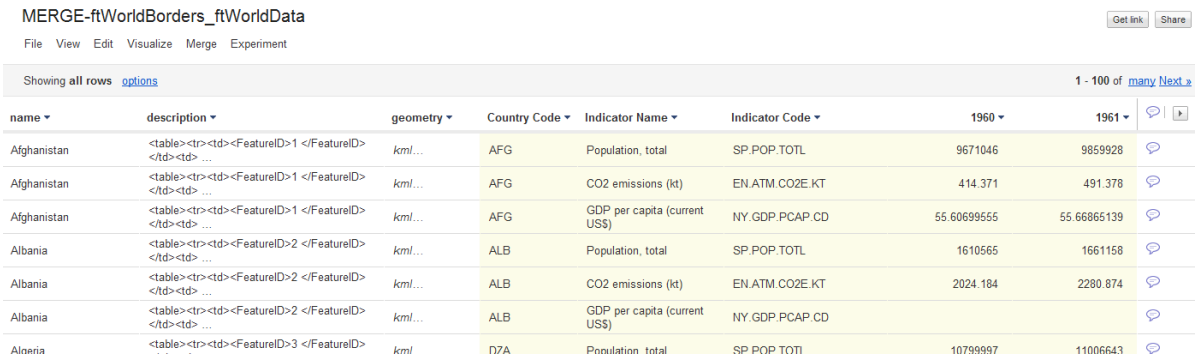


Abbildung 6.10.: WorldData: Merge der Landesgrenzen- und Daten-Tabelle

Merge-Tabelle für die Verwendung mit FusionTablesLayer vorbereiten

Um die Tabelle nun als FusionTablesLayer verwenden zu können, muss diese als *öffentlich* markiert werden. Dazu klickt man bei der geöffneten Tabelle auf den *Share*-Button in der linken oberen Ecke.

Abbildung 6.11.: WorldData: Freigabe der Merge-Tabelle

Im Dialogfenster wählt man unter *Visibility options* den Eintrag *Public* aus.

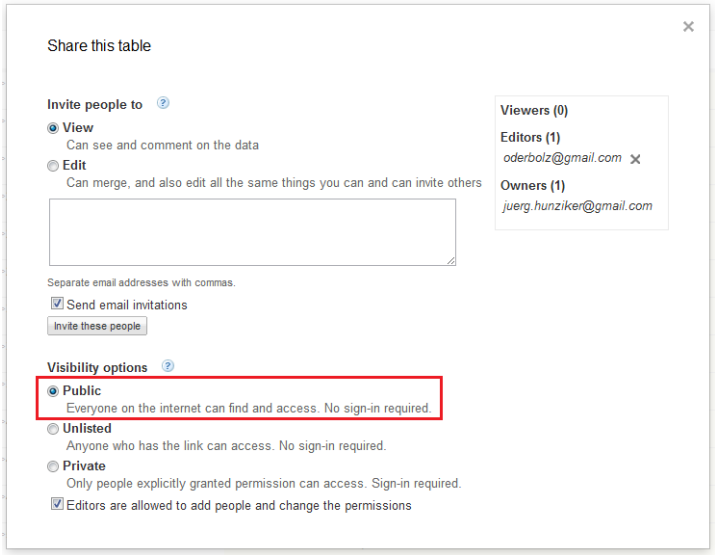


Abbildung 6.12.: WorldData: Freigabeeinstellungen der Merge-Tabelle

Als letzten Schritt muss man sich noch die eindeutige ID der Tabelle merken. Dazu wählt man im Menü *File* > *About* und kopiert sich die angezeigte *Encrypted ID* im geöffneten Dialog.

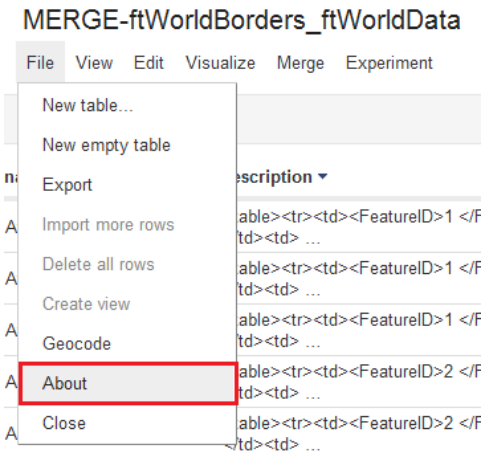


Abbildung 6.13.: WorldData: ID der Merge-Tabelle finden

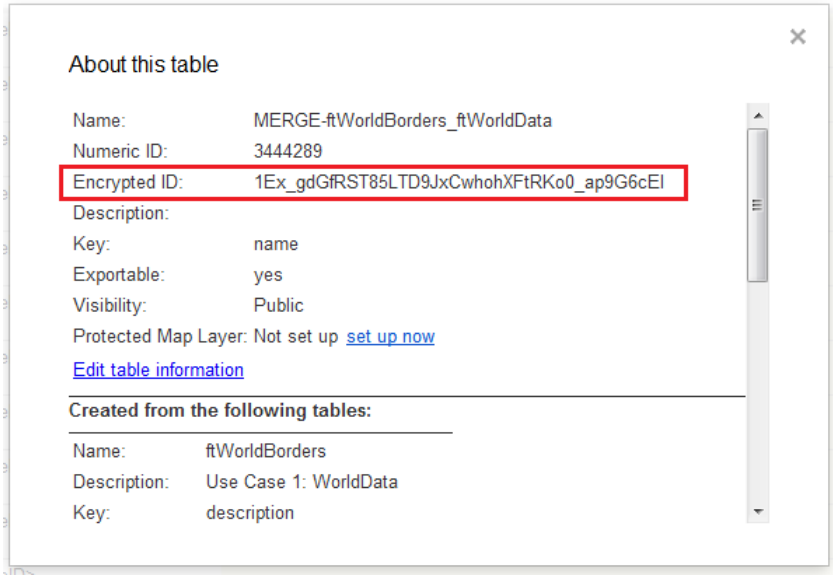


Abbildung 6.14.: WorldData: Dialog mit ID der Merge-Tabelle

6.8.2. Konfiguration der Applikation

Um die eben erstellte Fusion Table nun in der WorldData-Applikation zu verwenden muss das Konfigurations-File der Applikation folgendermassen angepasst werden.

/js/Config.js

Parameter	Typ	Beschreibung
`$.fusiontable.id`	`string`	Öffnetliche ID der zu verwendenden Fusion Table
`$.fusiontable.field`	`string`	Spaltenname welcher die Landesgrenzen speichert *(ACHTUNG: Gross-/Kleinschreibung relevant)*
`$.fusiontable.typeField`	`string`	Spaltenname aus welchem die Themen gelesen werden sollen *(ACHTUNG: Gross-/Kleinschreibung relevant)*

`$.fusiontable.types`	-	Konfiguration der verschiedenen Themen. Für jedes Thema muss ein neuer Block nach folgendem Schema erstellt werden:
		`'<TYPE>': {`
		`  name: '<TYPE.name>',`
		`  styleBoundaries: {`
		`  low: <TYPE.styleBoundaries.low>,`
		`  medium: <TYPE.styleBoundaries.medium>,`
		`  high: <TYPE.styleBoundaries.high>`
		`  }`
		`}`

`TYPE`	`string`	ID des Themas
`TYPE.name`	`string`	Bezeichnung des Themas
`TYPE.styleBoundaries.low`	`int`	Obere Grenze für tiefe Werte
`TYPE.styleBoundaries.medium`	`int`	Obere Grenze für mittlere Werte
`TYPE.styleBoundaries.high`	`int`	Obere Grenze für hohe Werte

Tabelle 6.5.: WorldData: Konfigurationsparameter

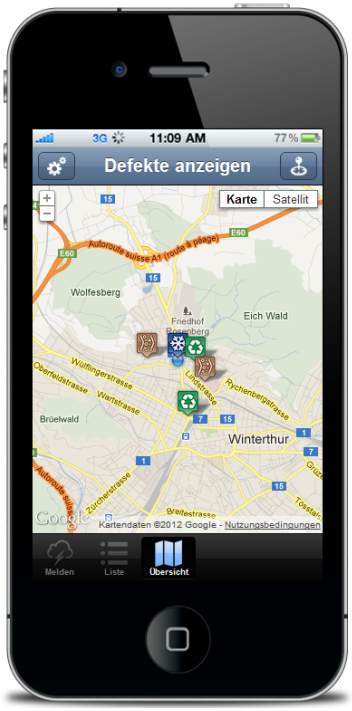
6.8.3. Starten der Applikation

Beim Starten der Applikation sollten nun alle Themen der angegebenen Fusion Table in das Auswahlfeld *Ebene auswählen...* geladen werden.

Abbildung 6.15.: WorldData: Auswahlliste der Themen

7. Use Case 2: FixMyStreet

http://fixmystreet.rdmr.ch/



7.1. Einführung

7.1.1. Idee

FixMyStreet ist ein Konzept, welches bereits in verschiedenen Städten bzw. Ländern umgesetzt wurde. Beispiele dafür sind Deutschland[1] oder England[2]. Beide Beispiele bieten neben der Webseite auch native Apps für iOS und Android an.

Die Idee ist so einfach wie auch genial: Man ermöglicht dem Bürger, per Webseite oder App, Defekte in seiner Umgebung (kaputte Strassenlampen, Schlaglöcher, usw.) direkt der zuständigen Behörde zu melden. Diese kann dann die erhaltenen Meldungen überprüfen und wenn nötig beheben. So können teure Kontrollfahrten auf ein Minimum reduziert werden.

7.1.2. Ziel

Das Ziel dieses Use Cases war die Erstellung einer Web-App, welche genau dieses Konzept umsetzt. Die Benutzer sollen die Möglichkeit haben Defekte in ihrer Umgebung zu melden.

Google Fusion Table soll dazu als Datenbank verwendet werden, in der die Defekte abgelegt werden. Natürlich sollen auch einige GIS-Features der Fusion Table verwendet werden, um beispielsweise nur die Defekte im aktuell sichtbaren Bereich der Karte zu laden.

7.2. Analyse

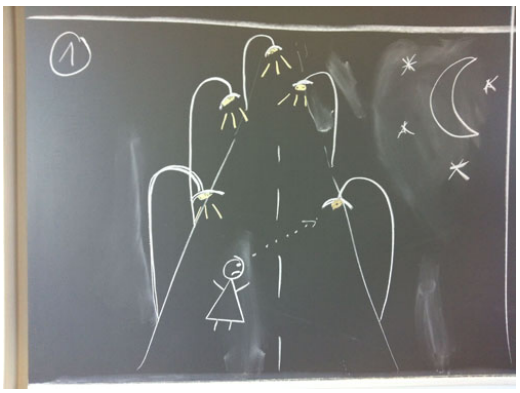
7.2.1. Storyboard

Zur Visualisierung der FixMyStreet-Idee wurde vorausgehend ein Storyboard erstellt.

1. Claudia entdeckt auf ihrem Heimweg eine defekte Strassenlampe.
2. Sie öffnet die *FixMyStreet* Web-App auf ihrem Handy und meldet den Standort der defekten Strassenlampe.
3. Am nächsten Tag überprüft der Werkshofleiter Franz die neuen gemeldeten Fälle im System und findet den Eintrag von Claudia.
4. Er macht sich auf den Weg und repariert die defekte Strassenlampe.

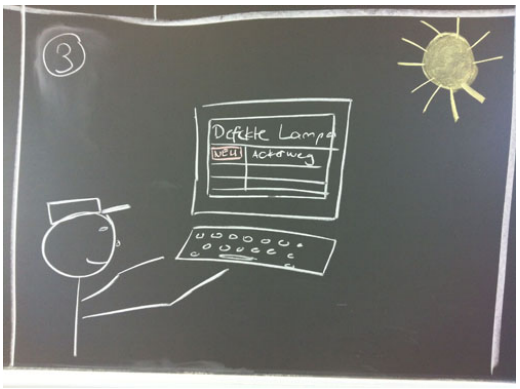
[1] http://de.seeclickfix.com/
[2] http://www.fixmystreet.com/

5. Am Abend darauf stellt Claudia erfreut fest, dass die Strassenlampe bereits repariert wurde.

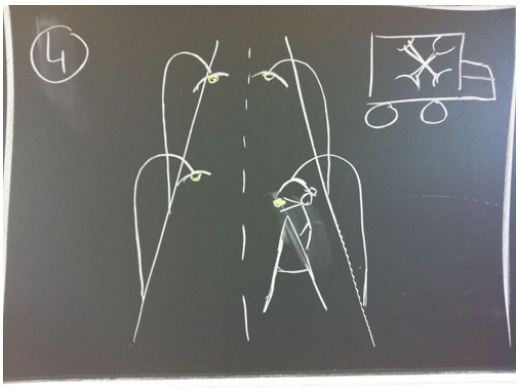


(1) Defekte Strassenlampe entdeckt

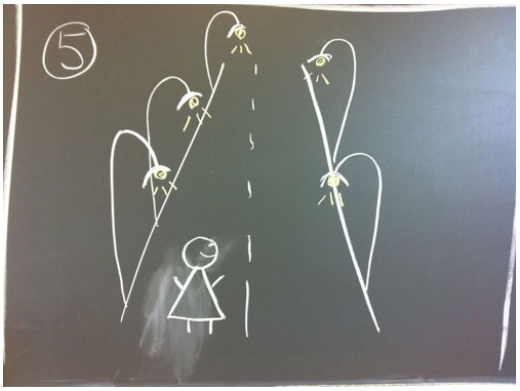
(2) Defekt melden



(3) Neue Meldungen überprüfen



(4) Defekt reparieren



(5) Defekt repariert

7.2.2. Use Cases

Die *FixMyStreet* Web-App soll folgende Use Cases abdecken.

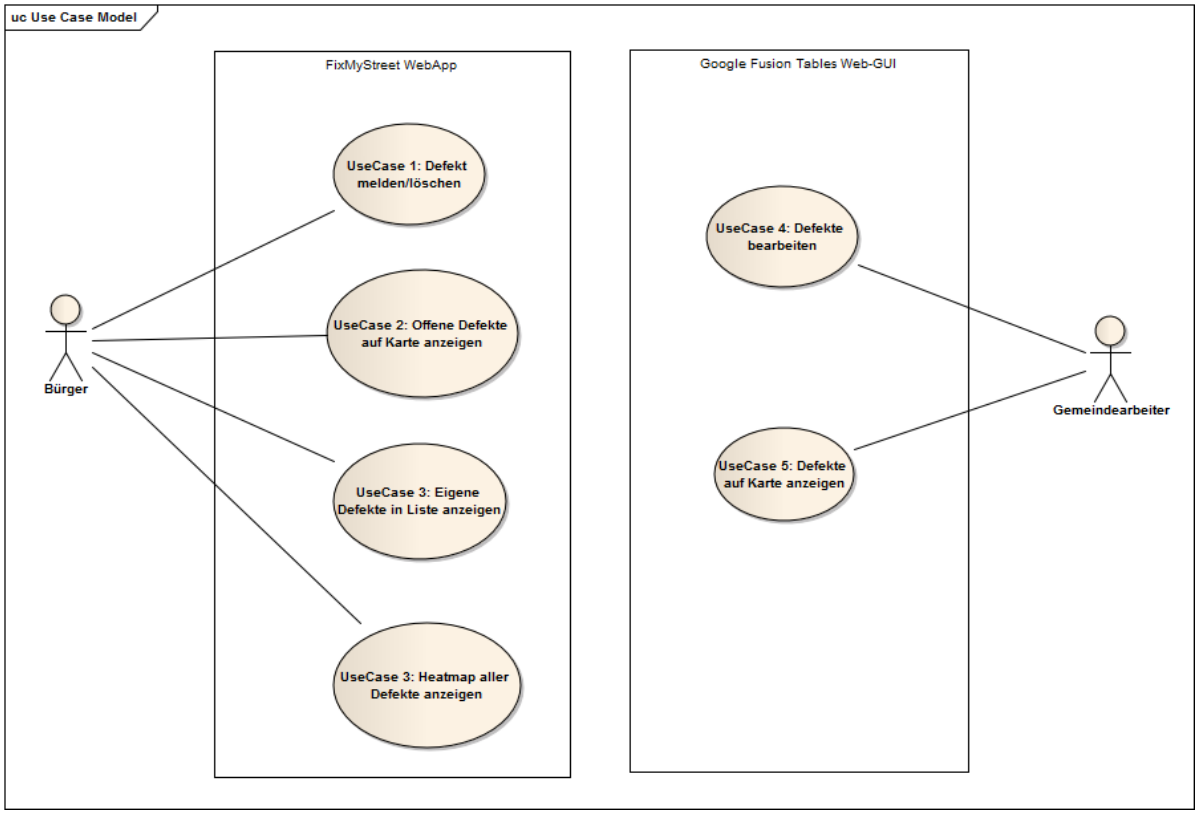


Abbildung 7.1.: FixMyStreet: UseCase Modell

Use Case 1a: Defekt melden

Primary Actor	Bürger
Stakeholders and Interests	Bürger: Möchte einen entdeckten Defekt melden
Preconditions	Web-App ist gestartet
Success Guarantee (Postconditions)	• Neuer Defekt ist in Datenbank gespeichert • Defekt ist in der Liste und auf der Karte ersichtlich • Melde-Maske wurde in den Ursprungszustand zurückgesetzt (kein Defekttyp ausgewählt, Markierung wieder auf die aktuelle Position verschoben)

Main Success Scenario	1. Bürger wählt Defekttypen aus 2. Bürger verschiebt Markierung auf Karte zur Position des Defekts 3. Bürger sendet den Defekt ab 4. Bürger bestätigt die Kontrollfrage, ob der defekt tatsächlich gesendet werden soll
Alternative Flows	2a. Markierung muss nicht zwangsläufig verschoben werden. Sie zeigt beim Starten der Web-App auf die aktuelle Position 2b. Falls die Web-App nicht auf die aktuelle Position zugreifen kann, zeigt die Markierung auf einen vorkonfigurierten Ort
Frequency of Occurrence	Tritt sehr häufig auf, da eine beliebige Anzahl von Bürgern Defekte melden kann

Use Case 1b: Defekt löschen

Primary Actor	Bürger
Stakeholders and Interests	Bürger: Möchte einen bereits gemeldeten Defekt löschen
Preconditions	• Web-App ist gestartet • Die Listenansicht wurde geöffnet • Es wurde bereits ein Defekt gemeldet
Success Guarantee (Postconditions)	• Der Defekt wurde aus der Datenbank gelöscht • Defekt ist nicht mehr auf der Liste und auf der Karte ersichtlich
Main Success Scenario	1. Bürger markiert den zu löschenden Defekt in der Liste 2. Bürger wählt den Befehl *Löschen*
Alternative Flows	2a. Falls der Status des Defektes bereits von einem Gemeindearbeiter geändert wurde (in beispielsweise *In Bearbeitung* oder in *Erledigt*), ist es für den Bürger nicht mehr möglich den Defekt zu löschen
Frequency of Occurrence	Tritt sehr häuft auf, da es für jeden Bürger, welcher bereits einen Defekt gemeldet hat, möglich ist seine eigenen Defekte wieder zu löschen

Use Case 2: Noch nicht behobene Defekte auf Karte anzeigen

Der Bürger öffnet die Übersicht. Darin werden ihm alle noch nicht behobene Defekte grafisch auf der Karte angezeigt. Die Daten werden laufend aktualisiert, damit er immer einen aktuellen Stand der gemeldeten Defekte sieht.

Use Case 3: Heatmap aller Defekte anzeigen

Der Bürger öffnet wiederum die Übersicht. Darin hat er die Möglichkeit die Defekte als Heatmap darstellen zu lassen. Dabei werden ihm Gebiete in denen viele Defekte gemeldet wurden farblich stärker markiert als Gebiete in denen kaum Defekte gemeldet wurden. Diese Ansicht wird ebenfalls laufend aktualisiert.

Use Case 4: Defekte bearbeiten

Der Gemeindearbeiter öffnet die FixMyStreet Fusion Table. Darin findet er eine Liste mit allen gemeldeten Defekten. Diese kann er direkt bearbeiten.

Hinweis: Dieser Use Case wird vom gegebenen Google Fusion Tables Web-GUI abgedeckt und wird nicht in der FixMyStreet-App realisiert.

Use Case 5: Defekte auf Karte anzeigen

Der Gemeindearbeiter öffnet die FixMyStreet Fusion Table. Er hat darin die Möglichkeit alle Defekte auf der Karte anzuzeigen.

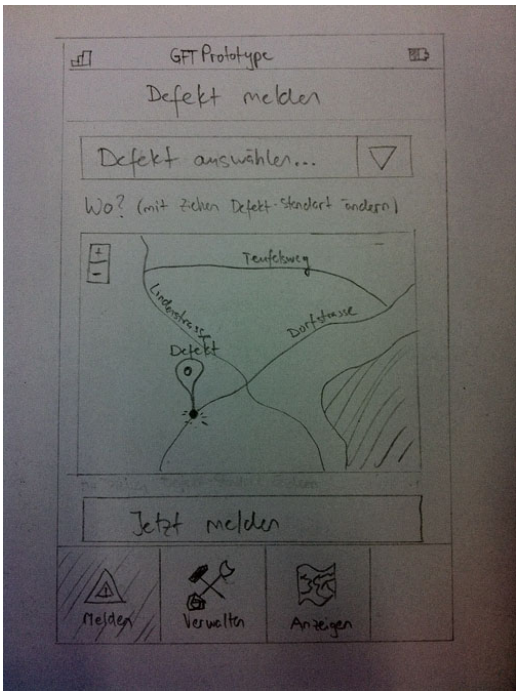
Hinweis: Dieser Use Case wird vom gegebenen Google Fusion Tables Web-GUI abgedeckt und wird nicht in der FixMyStreet-App realisiert.

7.2.3. Paper-Prototype

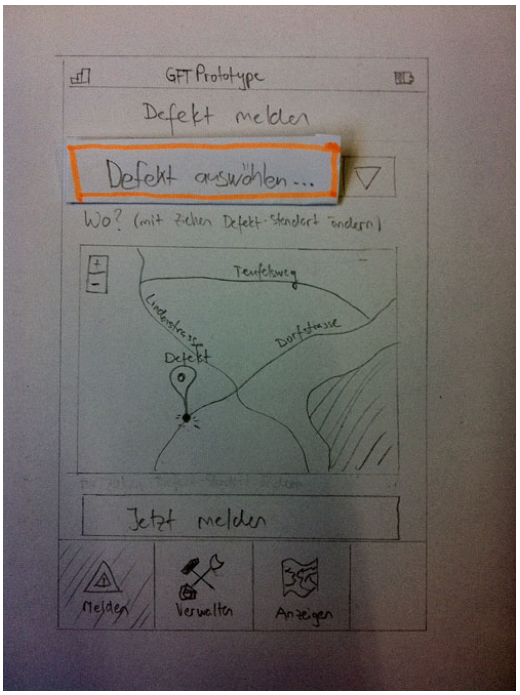
Vor der Implementation der Oberfläche wurde ein Paper-Prototype des GUI-Designs erstellt. Dieses wurde von verschiedenen Personen getestet. Der Prototype besteht aus drei verschiedenen Hauptmasken.

Maske: Defekt melden

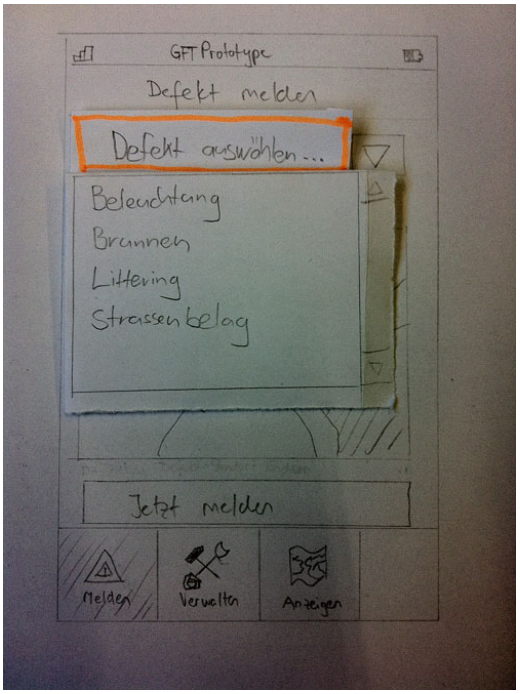
Auf dieser Maske können Defekte gemeldet werden. Dazu lässt sich zuerst der Defekt-Typ wählen. Danach kann man auf der Karte den Standort des Defekts auswählen indem man die Defekt-Markierung darauf verschiebt. Abschliessend lässt sich die Meldung absenden.



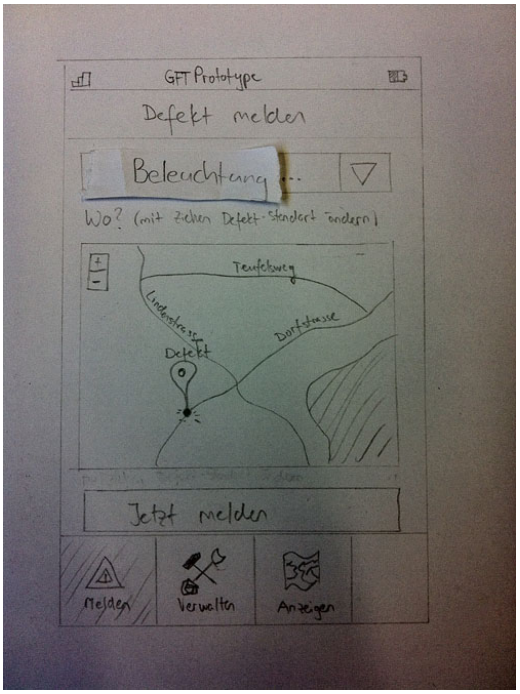
(1) Defekt melden - Übersicht



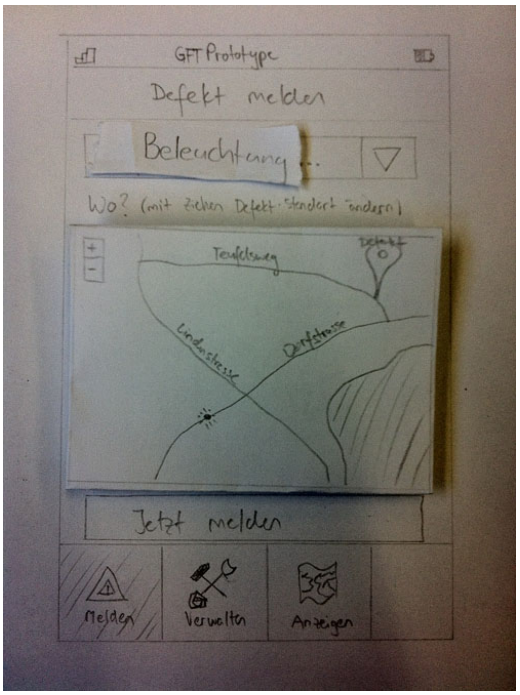
(2) Fehleranzeige beim Absenden ohne Defekttyp-Auswahl



(3) Defekttyp-Auswahl aufgeklappt



(4) Defekttyp ausgewählt

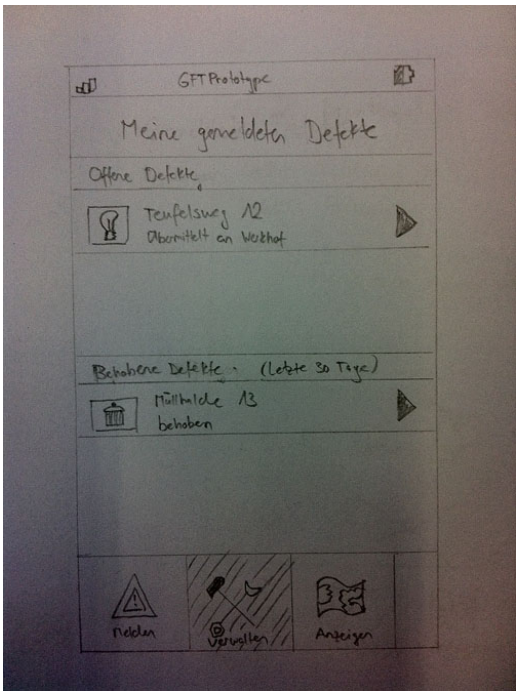


(5) Defekt-Markierung auf Karte verschoben

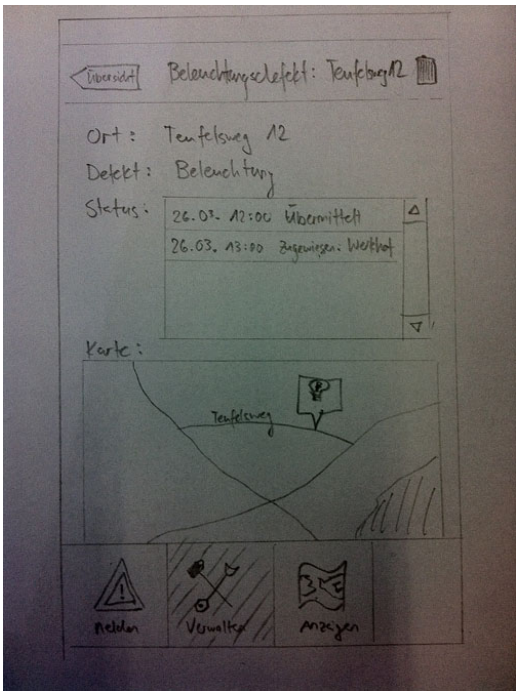
(6) Defekt erfolgreich gemeldet

Maske: Defekte verwalten

In dieser Maske kann man seine bereits gemeldeten Defekte verwalten. Diese werden gruppiert in *Offene Defekte* und *Behobene Defekte*. Zu jedem Defekt wird der aktuelle Status angezeigt. Mit einem Klick auf einen Defekt erreicht man dessen Detailanzeige.



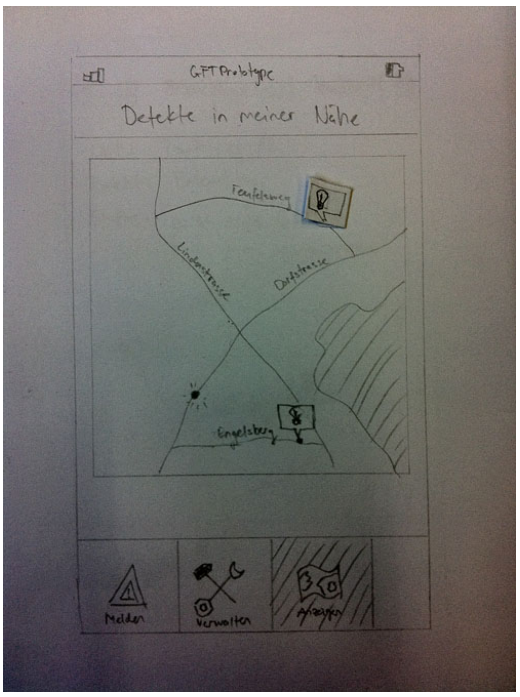
(7) Defekte verwalten - Übersicht



(8) Detailansicht eines gemeldeten Defekts

Maske: Defekte anzeigen

Auf dieser Maske werden alle gemeldeten Defekte in der Nähe angezeigt. Mit einem Klick auf eine Defekt-Markierung erhält man zusätzliche Information zu diesem Defekt.



(9) Anzeigen - Übersicht

7.3. Design

7.3.1. Datenbankschema

Die Applikation verwendet als Haupttabelle die Fusion Table *ftFixMyStreet*[3]. Davon wurden zwei Views erstellt:

- *ftFixMyStreet_Read*[4]: In dieser View sind alle gemeldeten Defekte sichtbar, die Applikation hat aber nur Lesezugriff darauf.
- *ftFixMyStreet_Write*[5]: In dieser View sind lediglich die Defekte vorhanden deren Status auf *Neu* gesetzt ist. Die Applikation kann darin neue Defekte abspeichern und diese auch wieder löschen.

[3] https://www.google.com/fusiontables/DataSource?docid=1ggQAh0WF7J7myI27_Pv4anl0wBJQ7ERt4W5E6QQ
[4] https://www.google.com/fusiontables/DataSource?docid=1no3_lJ0CCazZVN6rlAY8vrtf9ejoz_xo0e7a9cY
[5] https://www.google.com/fusiontables/DataSource?docid=1EX1S20fZmhetpLuWf_i-Hi5qfx1412a3TbRV1Ac (private Tabelle)

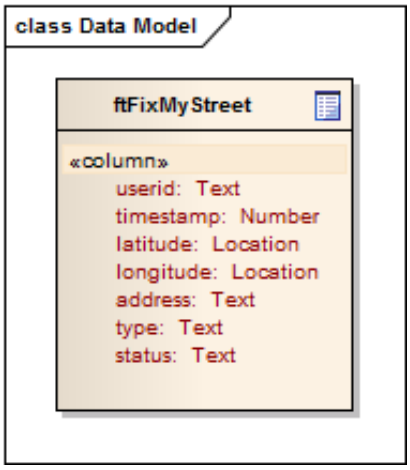


Abbildung 7.2.: FixMyStreet: Datenbankschema

7.3.2. Package-Struktur

Die Package-Struktur wird vom Sencha Touch Framework 2 vorgegebenen. Speziell daran ist das Konzept der *Stores*, welche einen beliebigen Datenspeicher abstrahieren. An einen Store wird ein *Model* gebunden, welches die Struktur der beinhaltenden Daten vorgibt. Zudem können Stores über einen *Proxy* an fremde Datenquellen gebunden werden. Einen solchen Proxy haben wir zur Anbindung der Google Fusion Tables an unsere Applikation entwickelt (siehe Abschnitt 7.4.4).

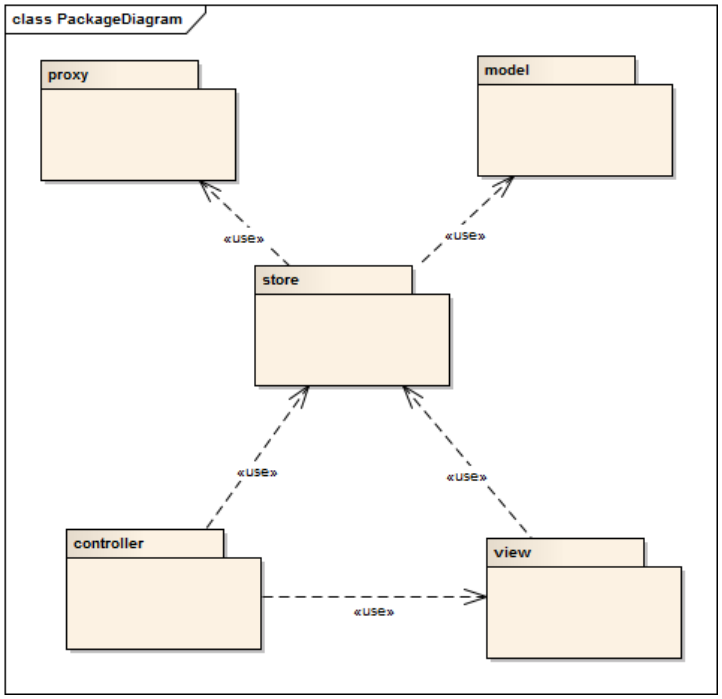


Abbildung 7.3.: FixMyStreet: Package-Diagramm

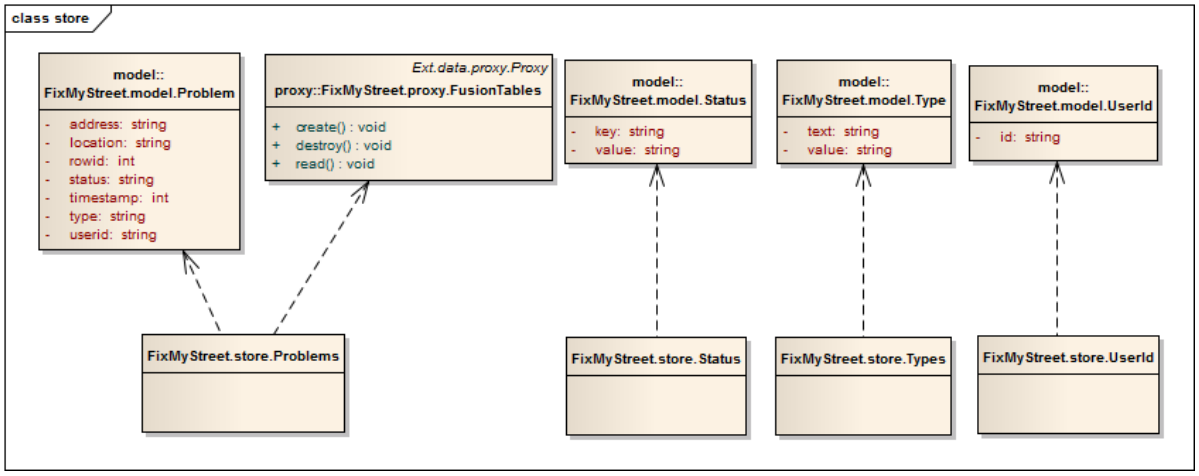


Abbildung 7.4.: FixMyStreet: Stores Klassendiagramm

7.3.3. Aufbau der Benutzeroberfläche

Wie in Abbildung 7.5 ersichtlich, besteht die Applikation aus den drei verschiedenen Hauptmasken *Report*, *List* und *Map*. Diese werden über den *MainContainer* in einem Tabpanel dargestellt.

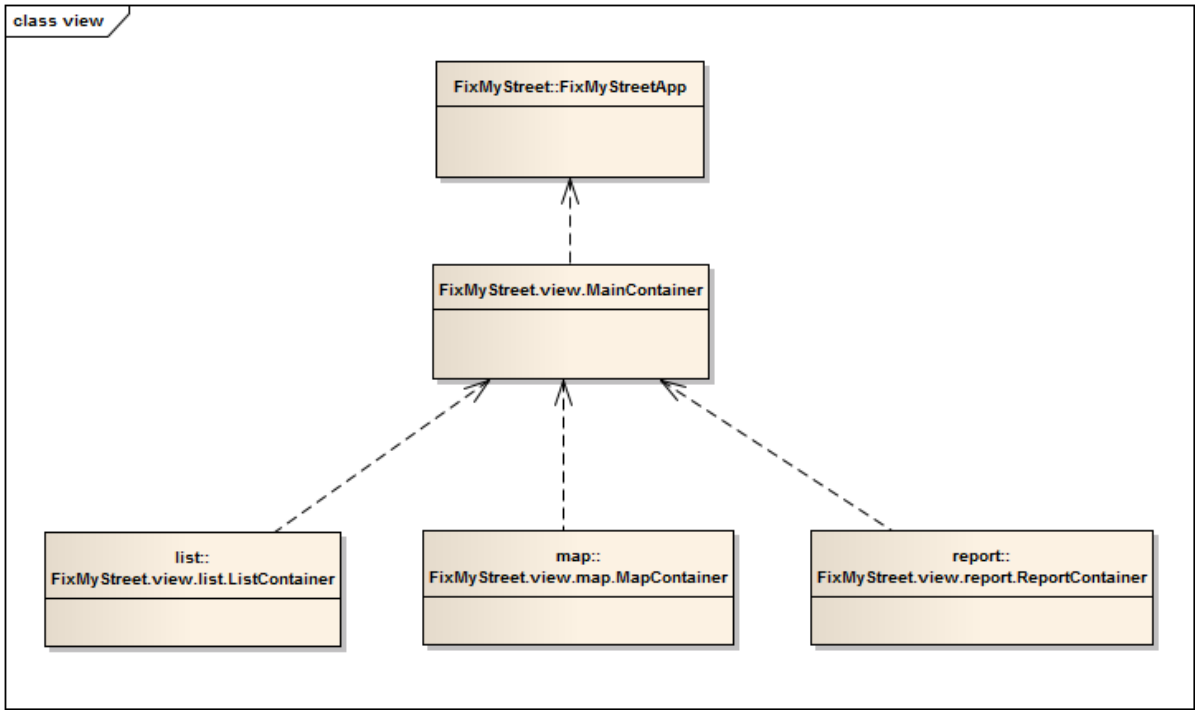


Abbildung 7.5.: FixMyStreet: View Klassendiagramm

7.3.4. Zusammenspiel der Controller

Die Applikation wird auf oberster Ebene vom *Main*-Controller gesteuert. Dieser ist dafür zuständig, die korrekten Masken anzuzeigen. Die einzelnen Masken werden dann von eigenen Controllern gesteuert.

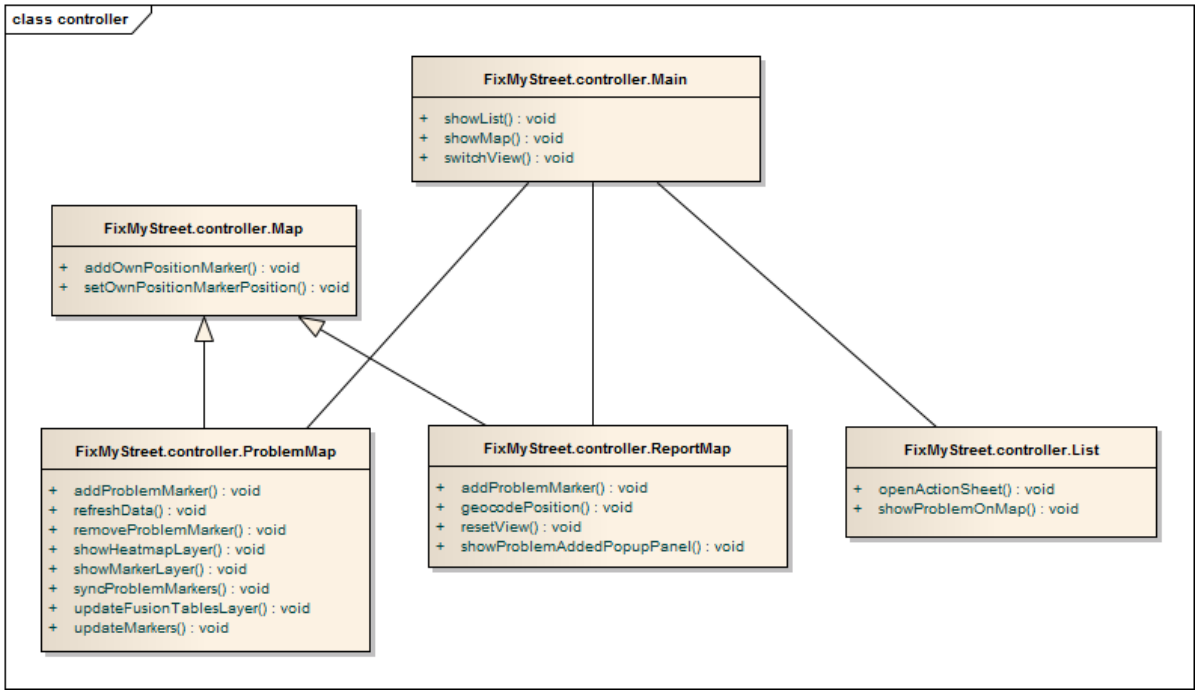


Abbildung 7.6.: FixMyStreet: Controller Klassendiagramm

7.3.5. Deployment

Die Applikation läuft direkt im Browser des Benutzers. Darin greift die *FusionTablesProxy*-Klasse (siehe Abschnitt 7.4.4) auf die GftLib (siehe Abschnitt 3.7) zu, welche die Verbindung zur Fusion Table erstellt. Dazu wird beim ersten Zugriff ein Zugriffstoken über den Web Service *OAuthTokenService* geholt. Danach kann mit diesem Token auf die Views *ftFixMyStreet_Read* und *ftFixMyStreet_Write* zugegriffen werden.

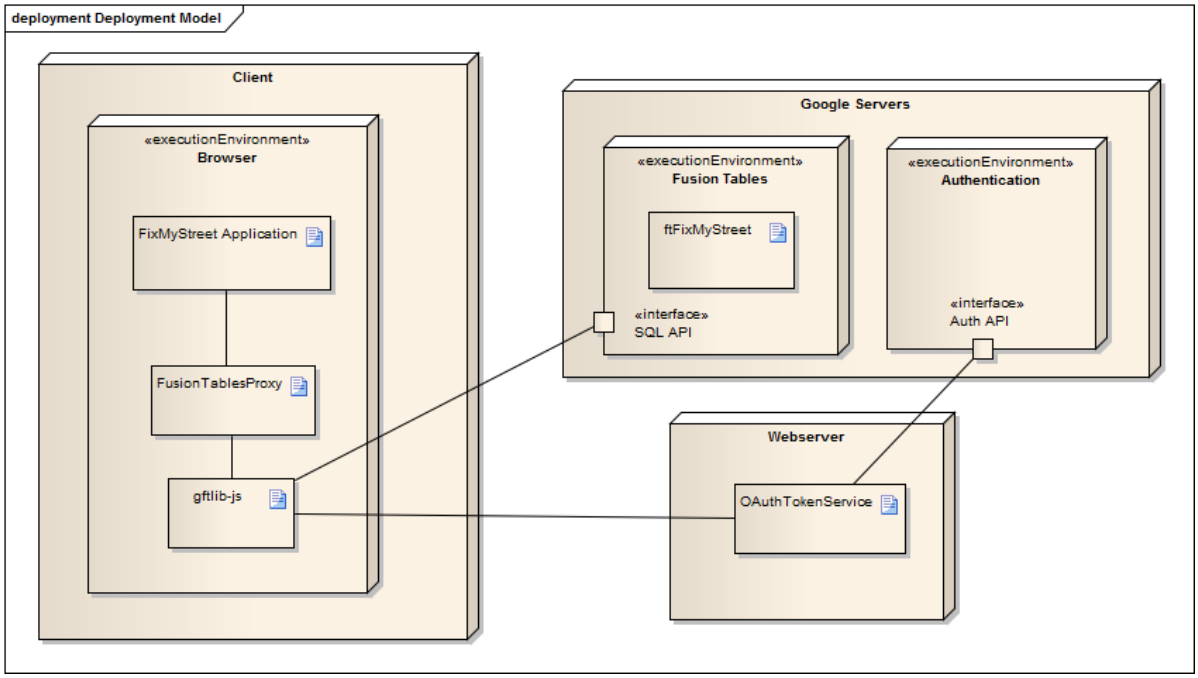


Abbildung 7.7.: FixMyStreet: Deploymentdiagramm

7.3.6. Starten der Applikation

Der Benutzer startet die Applikation mit dem Zugriff auf die `index`-Seite, welche die `launch()`-Funktion der Applikation aufruft. Danach werden folgende Schritte durchlaufen:

1. Laden der verschiedenen Status-Typen
2. Laden der Defekt-Typen
3. Laden der UserId aus dem LocalStorage
4. Falls noch keine UserId vorhanden ist, wird diese generiert und in den LocalStorage geschrieben (siehe Abschnitt 7.4.5)
5. Laden der bereits gemeldeten Defekte des Benutzers aus der Fusion Table
6. Aktueller Standort des Benutzers wird ausgelesen
7. Laden der Benutzeroberfläche

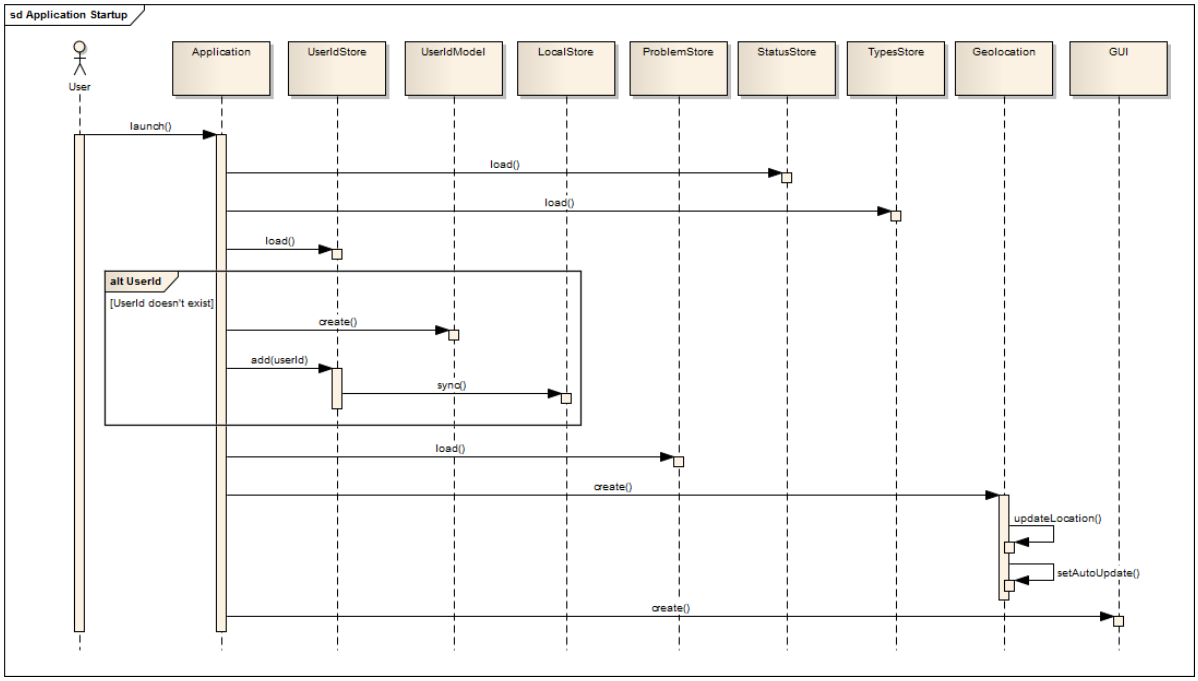


Abbildung 7.8.: FixMyStreet: Starten der Applikation

7.4. Implementation

Die Web-App basiert auf dem Sencha Touch 2 Framework[6]. Dieses bietet eine Basis zur Erstellung von mobilen Web-Apps mit JavaScript. Das Framework bietet die Möglichkeit eine Applikation nach dem MVC-Pattern aufzubauen.

Dazu verwendet Sencha Touch unter anderem das Konzept der Component-Queries[7]. Damit lassen sich View-Komponenten über verschiedene Attribute referenzieren. So kann man beispielsweise über deren eindeutige ID eine Referenz auf das Objekt erstellen.

In den Controller-Klassen werden genau über dieses Konzept Referenzen auf die View-Objekte erstellt und darauf zugegriffen. Damit ist es in fast allen Fällen möglich, die View-Klassen frei von Businesslogik zu halten.

Beispiel einer Component-Query

Der Code-Ausschnitt 7.1[8] zeigt einen Teil eines Controllers. Dieser referenziert im Property `nav` via Component-Query die View-Komponente mit der ID `mainNav`.

[6] http://www.sencha.com/products/touch/
[7] http://docs.sencha.com/touch/2-0/#!/api/Ext.ComponentQuery
[8] Quelle: http://docs.sencha.com/touch/2-0/#!/api/Ext.app.Controller (Stand: 24.05.2012)

```
Ext.define('MyApp.controller.Main', {
  extend: 'Ext.app.Controller',

  config: {
    refs: {
      nav: '#mainNav'
    }
  }

  ...
});
```

Code-Ausschnitt 7.1: Beispiel eines Component-Queries in Sencha Touch 2

7.4.1. Systemanforderungen

Das Sencha Touch 2 Framework unterstützt alle WebKit-fähigen Browser:

- Desktop:
  - Chrome
  - Opera
  - Safari
- Mobile:
  - iOS
  - Android
  - Blackberry

Eine vollständige Liste der unterstützten Geräte findet man auf der Seite http://www.sencha.com/products/touch/features/ unter *Sencha Touch 2 Device Support*.

7.4.2. Abhängigkeiten

Library	Version	Verwendung
Sencha Touch 2	2.0.1	Framework zur Erstellung von mobilen Web-Apps
GftLib	1.0	Kommunikation mit dem Google Fusion Tables via SQL API
jQuery	1.7.1	Wird von der GftLib verwendet
Google Maps API	V3	Anzeige der Karten

Tabelle 7.3.: FixMyStreet: Abhängigkeiten

7.4.3. Quellcode-Struktur

Datei	Beschreibung
`app/app.js`	Startet die Applikation
`app/controller/List.js`	Controller der *List*-View
`app/controller/Main.js`	Haupt-Controller der Applikation (Steuert die einzelnen Ansichten)
`app/controller/Map.js`	Basis-Controller der *Report* und *Map*-View
`app/controller/ProblemMap.js`	Controller der *Map*-View
`app/controller/ReportMap.js`	Controller der *Report*-View
`app/model/*.js`	Daten-Modelle der Applikation (Defekt, Typ, ...)
`app/plugin/PullRefresh.js`	PullRefresh Plugin der Liste
`app/proxy/FusionTables.js`	Proxy zur Anbindung der Google Fusion Table an die Applikation
`app/store/*.js`	Daten-Speicher der Applikation (Defekte, Typen, ...)
`app/utli/Config.js`	Konfiguration der Applikation
`app/util/Geolocation.js`	Steuert Zugriff auf aktuelle Position des Geräts
`app/view/*.js`	View-Klassen der Applikation
`resources/images/`	Bilder der Applikation
`resources/styles/`	CSS-Styles der Applikation
`index.html`	Einstiegspunkt der Applikation (inkludiert `app/app.js`, welches die Applikation startet)

Tabelle 7.4.: FixMyStreet: Quellcode-Struktur

7.4.4. Anbindung an Google Fusion Table

Das Sencha Touch 2 Framework verwendet für den Zugriff auf fremde Datenquellen das Konzept der *Proxies*. Ein Proxy verbindet einen internen Applikations-Store mit der eigentlichen Datenquelle.

Es gibt bereits vorgefertigte Proxies, welche die meisten Anwendungsfälle abdecken. So ist es möglich über einen LocalStorage-Proxy mit dem internen Browserspeicher (Web Storage[9]) zu kommunizieren. Zudem existieren Proxies für AJAX-Serveranfragen oder JSONP-Anfragen.

Für den Zugriff auf die Fusion Table als Datenbank mussten wir aber einen eigenen Proxy schreiben. Dieser verwendet unsere selbst geschriebene JavaScript Library GftLib (siehe Abschnitt 3.7), um über das SQL API mit der Fusion Table zu kommunizieren. Der Proxy implementiert lediglich die in unserem Use Case verwendeten CRUD-Operationen `Create`, `Read` und `Delete`. Ein Änderung (`Update`) von gemeldeten Defekten ist über die Web-App nicht vorgesehen und wurde deshalb auch nicht implementiert. Die Operationen sind jeweils in den Methoden `create()`, `read()` und `destory()` des Proxies abgebildet.

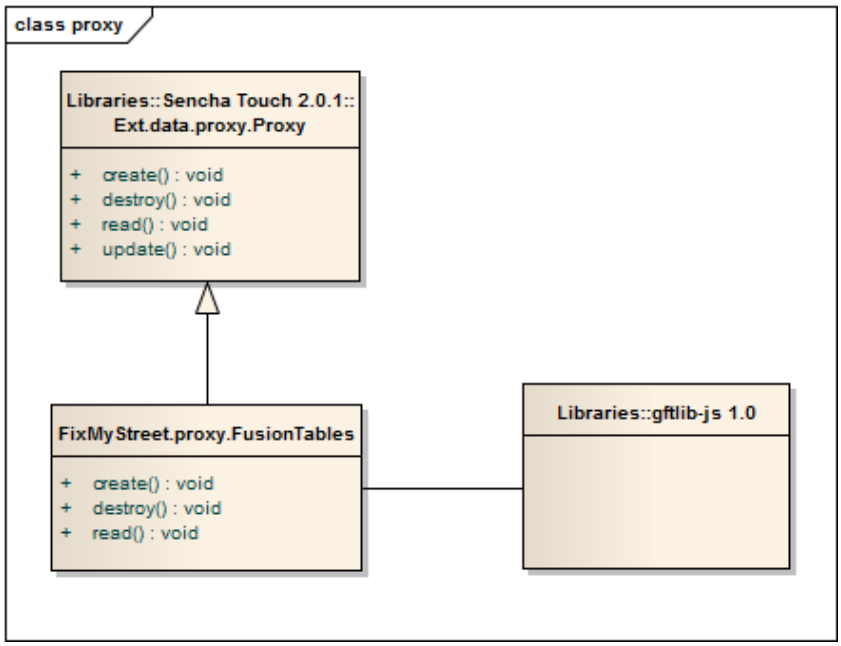


Abbildung 7.9.: FixMyStreet: Aufbau des FusionTablesLayer-Proxies

7.4.5. Wiedererkennung des Benutzers

Um die Hemmschwelle zur Benutzung der App möglichst klein zu halten, wollten wir möglichst darauf verzichten, dass sich die Benutzer zuerst registrieren müssen um diese

[9] http://de.wikipedia.org/wiki/Web_Storage

zu verwenden. Trotzdem soll die App die Möglichkeit haben einen Benutzer wiederzuerkennen, um ihm seine bereits gemeldeten Defekte anzeigen zu können.

Wir generieren dazu beim ersten Starten der Applikation eine *Version 1 UUID* (sequenziell) gemäss RFC 4122[10]. Diese legen wir im *Web Storage* des Browsers ab. Die gemeldeten Defekte beinhalten dann jeweils die UUID und können so den Benutzern zugewiesen werden.

Ein Problem ergibt sich aber aus diesem Vorgehen: Sobald ein Benutzer den Cache des Browsers löscht oder die App von einem anderen Gerät aus startet, erhält er eine neue UUID. Seine bereits gemeldeten Defekte kann er somit nur auf dem Gerät sehen, mit welchem er sie gemeldet hat. Dadurch können verwaiste Datensätze in der Datenbank entstehen, welche keinem Benutzer mehr zugeordnet werden können. In unserem Anwendungsfall ist dies aber nicht weiter schlimm, da die gemeldeten Fälle von der zuständigen Behörde direkt via Fusion Tables GUI bearbeitet werden. Darin spielt die Benutzerzuordnung keine Rolle.

7.4.6. Automatisches Aktualisieren der Übersichtsmaske

Eine weitere Anforderung war es eine möglichst aktuelle Ansicht aller gemeldeten Fälle zu bieten. Wir haben dazu in der Übersichtsmaske einen Polling-Mechanismus implementiert. Dieses aktualisiert alle 30 Sekunden die Markierungen auf der Karte indem es eine Anfrage an die Fusion Table sendet. Gelöschte und erledigte Defekte werden entfernt und Neue hinzugefügt.

Die Karte aktualisiert sich zudem jedes Mal, wenn die Übersichtsmaske aufgerufen wird.

7.4.7. Nur Defekte im sichtbaren Bereich laden

Um nicht zu viele Markierungen gleichzeitig zu laden, was sich negativ auf die Performance der Applikation auswirken würde, laden wir auf der Übersichtsmaske nur diejenigen Defekte, welche im aktuell sichtbaren Bereich der Karte liegen. Sobald die Karte verschoben oder der Zoom verändert wird, werden die Defekte für den neuen Bereich nachgeladen.

Dazu haben wir vorerst das Spatial-Query `ST_INTERSECTS` des SQL APIs (siehe Abschnitt 3.2.2) verwendet. Als Grenze geben wir diesem als Parameter ein Rechteck bestehend aus zwei Ecken der momentan angezeigten Karte mit.

Aufgrund der Einschränkung, dass neue Datensätze nicht geocodiert werden (siehe Abschnitt 3.5.2), mussten wir das Spatial-Query aber wieder entfernen und durch eine eigene `WHERE`-Bedingung ersetzen. Ansonsten würden neu gemeldete Defekte solange nicht auf der Karte angezeigt, bis die Datensätze über das Fusion Tables Web-GUI manuell

[10] http://tools.ietf.org/html/rfc4122

geocodiert werden.

7.4.8. Heatmap-Ansicht

Auf der Übersichtsmaske hat man die Möglichkeit alle gemeldeten Defekte als Heatmap darzustellen. Dazu haben wir das eine Fusion Table-Ebene über die Karte gelegt und dessen Heatmap-Feature (siehe Abschnitt 3.5) verwendet.

7.5. Testing

Das verwendete Framework Sencha Touch 2 machte das Testing der Applikation nicht ganz einfach. Obwohl das Framework den Code sehr stark nach MVC gliedert, haben die einzelnen Teile eine starke Kopplung an das Framework. Dies ist auch der Grund, weshalb wir die Applikation nur punktuell getestet haben[11].

Zum einen konnten wir prüfen ob alle Teile der Applikation korrekt geladen werden. Dies hilft dafür immer einen konsistenten Stand an Komponenten zu haben, welche so auch zusammen funktionieren können. Zum anderen prüfen wir einige Konfigurationen. Dabei geht es vor allem darum um Änderungen daran zu erkennen.

Der Use Case basiert zu einem grossen Teil auf unserer Bibliothek GftLib, welche wir für den Zugriff auf die Fusion Tables Datenbank verwenden. Die Bibliothek ist mit ausführlichen Test ausgestattet (siehe Abschnitt 3.7.5). Wir haben schliesslich noch versucht den Proxy (siehe Abschnitt 7.4.4) zu testen, sind jedoch an den Abhängigkeiten zum Framework gescheitert.

7.6. Resultate

Das Resultat dieses Use Cases ist eine voll ausgereifte Web-App.

7.6.1. Maske: Defekt melden

Dies ist die Hauptmaske der App. Sie wird geöffnet sobald der Benutzer die Applikation startet. Darauf kann der Benutzer Defekte an beliebigen Orten melden.

Features

- Defekt-Markierung kann per *Drag&Drop* oder per *Klick* auf der Karte verschoben werden

[11] Ausführbare Tests: http://gft.rdmr.ch/test/js/?filter=FixMyStreet

- Zentrierung der Karte und Positionierung der Defekt-Markierung auf die eigene Position

- Schritt für Schritt Anleitung

Screenshots

(1) Defekt melden

(2) Bestätigungsmeldung

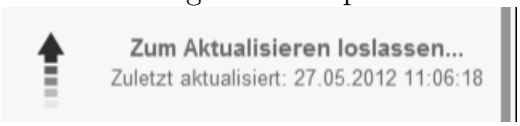
(3) Defekt erfolgreich gemeldet

7.6.2. Maske: Meine Defekte

In dieser Ansicht werden dem Benutzer alle bereits von ihm gemeldeten Defekte aufgelistet. Die Defekte sind nach ihrem aktuellen Status (*Neu*, *In Bearbeitung* und *Erledigt*) gruppiert.

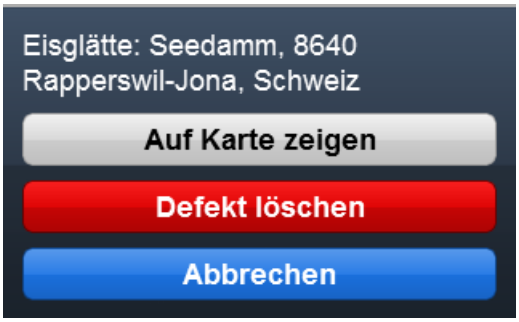
Features

- Aktualisierung der Liste per Pull-Down-Geste



- Durchsuchen der Liste

- Öffnen des Kontextmenüs zu einem Eintrag per *Swipe*- oder *Tap-And-Hold*-Geste



Screenshots

(4) Liste mit gemeldeten Defekten

(5) Aktualisiern der Liste

(6) Kontextmenü eines Defekts

7.6.3. Maske: Defekte anzeigen

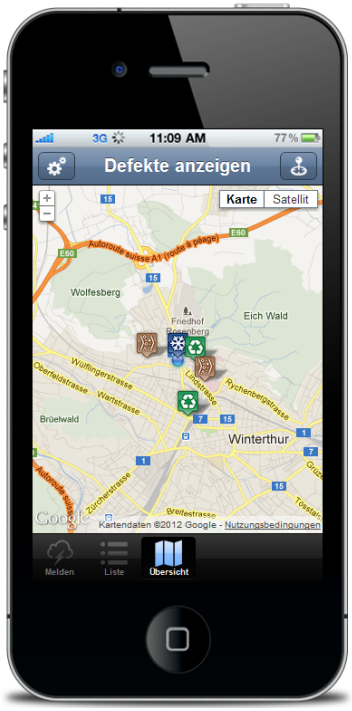
Hier werden dem Benutzer alle noch nicht behobenen Defekte auf der Karte angezeigt.

Features

- Zentrieren der Karte auf die eigene Position
  
- Filtern nach Defekt-Typ

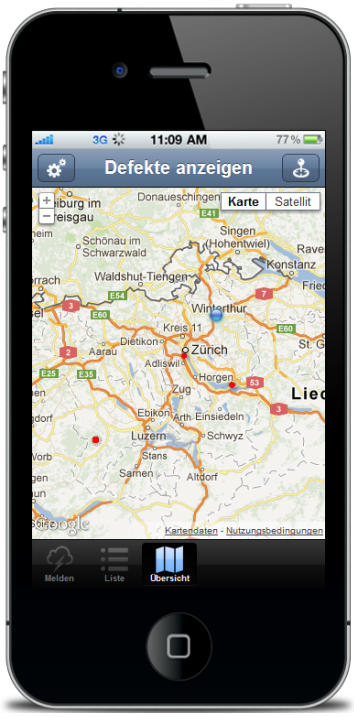
- Anzeige der Defekte als Heatmap

Screenshots



(7) Defekte anzeigen

(8) Einstellungen



(9) Defekte als Heatmap anzeigen

7.6.4. Probleme bei der Verwendung von Google Fusion Tables als Datenbank

Wir hatten einige Probleme bei der Verwendung einer Fusion Table als Datenbank der Applikation. Ein Hauptproblem war klar das Fehlen eines `GRANT`-Mechanismus, wie man ihn von vielen anderen Datenbanksystemen kennt. Die gegebenen Möglichkeiten für die Vergabe von Lese- oder Schreibmöglichkeiten würden für den Gebrauch in einer produktiven Applikation kaum ausreichen.

Zusätzlich konnten wir die GIS-Features, welche Google Fusion Tables anbieten nur minimal nutzen. So hinderte uns beispielsweise die fehlende Geocodierung von neu eingefügten Datensätzen daran Spatial-Queries zu verwenden (siehe Abschnitt 3.5.2).

Zudem haben wir bewusst auf die Verwendung einer Fusion Table-Ebene (siehe Abschnitt 3.6) für die Übersichtsmaske verzichtet, da wir dadurch die Möglichkeit verloren hätten Custom-Markers auf der Karte hinzuzufügen. Dies hätte die Usability der App stark verschlechtert, da man nicht mehr direkt den Typen des markierten Defekts erkannt hätte.

Alles in allem waren die Nachteile bei der Verwendung von Google Fusion Tables als Datenbank wahrscheinlich grösser als deren Vorteile.

7.7. Weiterentwicklung

Auch dieser Use Case hat noch ein grosses Ausbaupotential. So wäre es beispielsweise sinnvoll, wenn man bei der Meldung des Defekts noch ein Foto hinzufügen könnte. Damit könnte man noch genauer zeigen, was wirklich defekt ist. Zudem würde es auch der zuständigen Behörde einfacher fallen „Spassmeldungen“ von echten Defekten zu unterscheiden. Zusätzlich wäre es hilfreich, wenn man die Möglichkeit hätte in einem Beschreibungsfeld eine zusätzliche Problembeschreibung anzugeben.

Weiter müssten natürlich auch die Stammdaten (Defekttypen, Status) in einzelne Tabellen ausgelagert werden. Diese sind momentan statisch im Code verankert. Ein Problem würde sich dabei aber stellen: Das Web-GUI der Fusion Table unterstützt momentan keine Auswahllisten von Daten aus einer fremden Tabelle. Man würde deshalb lediglich die Fremdschlüssel der Stammdaten sehen.

Ein weiteres Feature wäre eine Detailansicht der gemeldeten Defekte. Darin liesse sich der Status-Log anzeigen, um genau nachzuverfolgen, was bereits zur Behebung des angezeigten Defekts unternommen wurde.

Zudem wäre es sinnvoll, wenn die App auch offline nutzbar wäre. Dadurch könnte man Defekte auch an Orten melden, welche über keine ausreichende Mobilnetzabdeckung verfügen.

8. Konvertierung von GIS-Daten

8.1. Idee

Für die Konvertierung von GIS-Daten hatten wir die Idee, dies mit Hilfe eines Builds zu machen. Das heisst ein Skript auf unserem Build-Server Jenkins (siehe Abschnitt 4.3) laufen zu lassen. Dadurch fiel die Programmierung eines Web-GUIs weg, da dieses bereits von Jenkins generiert wird. Auch Sicherheitsaspekte wie die bereits vorhandene Benutzerverwaltung spielten eine Rolle.

Der Converter-Build war zu Beginn ein sehr einfaches Hilfsmittel um Testdaten in Google Fusion Tables zu laden. Das Problem war, dass GFT von sich aus nur den Import von Google Spreadsheet, CSV- und KML-Dateien zu lässt, andere Formate sind nicht unterstützt. Viele frei verfügbare GIS-Daten sind jedoch in anderen Formaten verfügbar, so dass eine Konvertierung zwangläufig notwendig ist. Zuerst haben wir jeweils den GeoConverter der HSR[1] für diesen Zweck verwendet. Der Prozess war allerdings etwas mühsam, da die konvertierte Datei dann jeweils noch manuell in eine Fusion Table importiert werden musste.

Der Build war schlussendlich eine einfache Lösung, um schnell beliebige Dateien zu konvertieren oder in GFT zu importieren.

8.2. Implementation

8.2.1. Technischer Hintergrund

Die Konvertierung basiert auf der Open Source Bibliothek GDAL/OGR[2].

Technisch gesehen ist der Converter-Build ein einfaches Ant-Skript welches das Kommandozeilen Tool ogr2ogr[3] abstrahiert. Dieses Tool wird auf dem Server ausgeführt mit den entsprechend eingegebenen Parametern.

[1] http://geoconverter.hsr.ch/
[2] http://www.gdal.org/ogr
[3] http://www.gdal.org/ogr2ogr.html

8.2.2. Features

Da der Build auf ogr2ogr basiert, sind die Limitation durch dieses Tool gegeben. Es wird eine Vielzahl an GIS-Formaten unterstützt[4]. Speziell zu erwähnen ist dabei die Unterstützung für Google Fusion Tables, da wir damit die Möglichkeit bekommen haben, Daten direkt in eine Fusion Table zu importieren[5].

Für die anderen Formate wird jeweils das konvertierte File wieder zum Download angeboten.

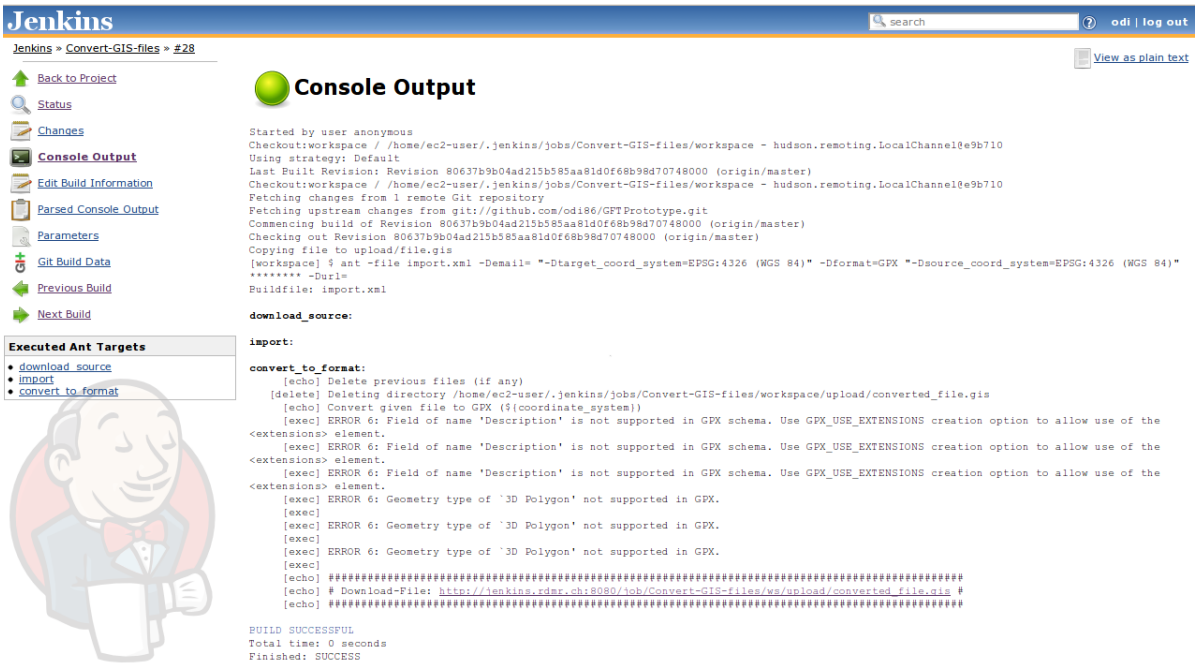


Abbildung 8.1.: Konvertierung abgeschlossen

Als Quelldatei kann man wahlweise eine Datei hochladen oder eine URL auf eine Datei im Internet angeben. Weiter lässt sich das gewünschte Dateiformat und eine allfällige Koordinatenkonvertierung einstellen. Falls als Ausgabeformat *GFT* gewählt wurde, muss man noch seine Angaben zu seinem Google-Konto angeben.

[4] http://www.gdal.org/ogr/ogr_formats.html
[5] http://www.gdal.org/ogr/drv_gft.html

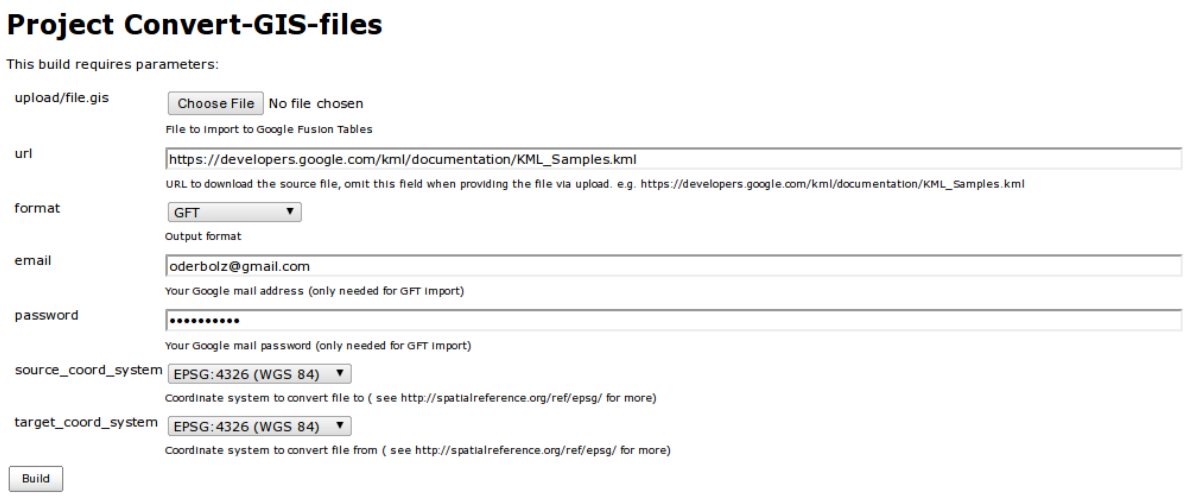


Abbildung 8.2.: Optionen für den Converter-Build

8.3. Installation

Die Voraussetzungen für den Converter-Build sind zunächst eine Jenkins-Instanz, auf welcher das Plugin *Parametetrized Build*[6] installiert ist. Damit lassen sich an den Build via Web GUI Parameter übergeben. Grundsätzlich können damit generische Build-Jobs erstellt werden.

Das Herzstück des Builds ist die GDAL/OGR Bibliothek welche sich aus dem Quellcode kompilieren lässt. Wichtig ist dabei, dass die Option `--with-curl` angegeben wird, da ansonsten der GFT-Import nicht funktioniert.

```
1 $ ./configure --with-curl
2 $ make
3 # make install
```

Code-Ausschnitt 8.1: Kompilierung der GDAL/OGR Bibliothek mit cURL

Für die Koordinaten-Konvertierung muss zusätzlich noch die Bibliothek Proj.4[7] installiert werden.

[6] https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Build
[7] http://trac.osgeo.org/proj/

Teil IV.

Projektmanagement und -monitoring

9. Projektmanagement

Wir verwendeten die agile Projektmethologie *Scrum*[1] und arbeiteten dabei während 12 Wochen (4 Sprints à 3 Wochen pro Sprint). Für diese Studienarbeit werden 8 ECTS-Punkte vergeben, wobei 1 ECTS-Punkt 30 Stunden Arbeitsaufwand bedeuten. Dies ergibt 240 Stunden pro Person, was etwas mehr als 2 Arbeitstagen pro Woche entspricht.

Als Ausgangslage haben wir 12 Storypoints pro Sprint angenommen, wobei 1 Storypoint ungefähr einem Arbeitstag entspricht. Bis zum Schluss hat sich dieser Wert nicht merklich verändert.

9.1. Kickoff

Das Projekts wurde mit einem Kickoff-Meeting zusammen mit dem Betreuer und dem Industriepartner am 23. Februar 2012 gestartet. Darin wurden nebst der Aufgabe auch alle Formalitäten der Arbeit besprochen. Es wurde fest gelegt, dass neben dem theoretischen Teil der Arbeit über Google Fusion Tables auch drei Anwendungsfälle aus dem GIS-Bereich umgesetzt werden sollen.

9.2. Sprint 1

Der erste Sprint stand ganz im Zeichen des Einarbeitens. Es ging uns darum mit dem Google Fusion Tables API bekannt zu werden. Daneben musste noch unsere Infrastruktur eingerichtet werden. Dazu zählt das Repository, unser Projektmanagement-Tool und der Build-Server.

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

Hauptaufgaben / Fokussierung im Sprint

- Aufsetzen der Infrastruktur (Repository, Projektmanagement, Entwicklungsumgebung, Server)
- Einarbeitung in die Thematik

[1] http://www.scrum.org

  - GIS
  - Google Fusion Tables (GFT)
  - allenfalls weitere APIs

- Erarbeitung eines ersten Roundtrips um Daten von und zu Google Fusion Tables zu schicken/empfangen
- Projektsetup
  - GitHub[2] als Repository
  - LaTex[3] für Dokumentation
  - Jenkins [4] für Continuous Integration (CI)
  - Redmine [5] für Projektmanagement/Wiki/Bugtracker
  - Scrum als Methodik

Ziele

- Google Fusion Table API kennen lernen, Potential abschätzen können
- Erster Prototyp mit GFT bauen (Roundtrip mit CRUD-Operationen)

Abgabe / Deliverables

Wir sind im ersten Sprint gut vorangekommen und konnten mit zahlreichen aufeiander aufbauenden Beispielen lernen wie das API funktioniert und welche Möglichkeiten es bietet: Abfragen erstellen, Zugriff via API, Zugriff via Google Maps Layer (FusionTablesLayer).

- Repository, Build-Server und Projekmanagement-Tool aufgesetzt (siehe Kapitel 4)
- Lauffähiger Prototyp mit Unit-Test für CRUD-Operationen
- Zahlreiche Beispiele um die Funktionsweise des APIs zu testen

Probleme

Es ist uns nicht gelungen den Roundtrip zu erstellen, d.h. Daten vom Benutzer in Fusion Tables zu speichern und diese wieder abzufragen. Wie sich herausgestellt hat, müssen

[2] http://www.github.com
[3] http://www.latex-project.org
[4] http://jenkins-ci.org
[5] http://www.redmine.org

schreibende Zugriffe autorisiert sein. Dazu empfiehlt Google OAuth zu benutzen. Diese Thematik war schlicht zu gross um im ersten Sprint anzuschauen. Die schreibenden Zugriffe sind deshalb Ziel für den 2. Sprint geworden.

9.3. Sprint 2

Im zweiten Sprint gab es zwei Schwerpunkte: Zum einen mussten wir uns langsam Gedanken machen, welche Use Cases wir mit den Google Fusion Tables abdecken wollen. Aus diesen Use Cases sollen dann unabhängige Applikationen entstehen, welche dann das Potential des Produktes aufzeigen sollen. Zum anderen gab es noch ein wichtiges technischen Thema, nämlich die Schreiboperationen. Dazu waren einige Grundlagen von OAuth nötig, so dass wir dann die ganze Bandbreite der Schnittstelle nutzen konnten.

Als Nebenthema mussten wir uns noch um den Import von verschiedenen GIS Dateien in Fusion Tables kümmern. Zum einen ist dies ein relevantes Thema um eine Migration zu ermöglichen, zum anderen sind viele Daten in beliebigen Formate verfügbar, welche wir natürlich gern als Testdaten nutzen möchten.

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

Hauptaufgaben / Fokussierung im Sprint

- Use Cases erarbeiten
- GIS Daten in GFT importieren
- Informationen über das *Trusted Tester API* sammeln
- Schreiboperationen (INSERT/UPDATE/DELETE) auf Fusion Tables durchführen können (Authentifizierung mit OAuth notwendig)

Ziele

- Finden von WebGIS Use Cases (1 „grosser“ und 2 „kleine“ Use Cases)
- Implementation eines kleinen Use Cases
- Geo-Daten importieren und verknüpfen
  - KML importieren
  - Converter einsetzen, dann importieren
  - Verschiedene Fusion Tables joinen/mergen

- Trusted Tester API
  - Zugriff erhalten
  - API testen

Abgabe / Deliverables

Als ersten umzusetzenden Use Case haben wir uns für *WorldData* (siehe Kapitel 6) entschieden. Zudem haben wir einen Build-Job (siehe Kapitel 8) erstellt, welcher es uns erlaubt auf einfache Art und Weise GIS Daten zu importieren.

Für OAuth haben wir ein Beispiel entwickelt, welches es einem Benutzer ermöglicht seine Tabellen für unsere Applikation freizugeben, so dass dann Schreiboperationen möglich werden.

- Erster umgesetzter Use Case *WorldData*
- Import-Verfahren für GIS Daten
- Erweiterte GftLib (Schreiboperationen, Authentifizierung mit OAuth)

Probleme

Der Converter-Build war noch nicht sehr ausgereift, so war es noch nicht möglich das Koordinationsystem zu wechseln und überhaupt ein anderes Format zu wählen als GFT.

Bei der Authentifizierung gab es noch einige Probleme um diese für die Library nutzbar zu machen. Für den Fall Google Fusion Tables als Backend-Datenbank zu verwenden und nicht lediglich für den Zugriff auf Fusion Tables eines Benutzers, gab es noch keine Lösung.

Aus zeitlichen Gründen haben wir beim Review des zweiten Sprints zusammen mit dem Betreuer und dem Industriepartner entschlossen, dass wir nur noch einen weiteren „grossen“ Use Case implementieren werden.

9.4. Sprint 3

Im dritten Sprint war das Hauptziel die Umsetzung des zweiten Use Cases *FixMyStreet* als mobile Web-App. Dies beinhaltete unter anderem auch die Anbindung des Sencha Touch 2 Frameworks an die Google Fusion Table. Zusätzlich wollten wir noch einen Converter-Build erstellen, mit welchem es möglich ist verschiedenste GIS-Formate in eine Google Fusion Table zu importieren oder in andere Formate zu konvertieren.

Alle Informationen zum Sprint 3 sind auch in unserem Wiki zu finden: http://redmine.

rdmr.ch/redmine/projects/gftprototype/wiki/Sprint_3

Hauptaufgaben / Fokussierung im Sprint

- Zweiter Use Case *FixMyStreet* implementieren
- Converter-Build erstellen

Ziele

- Lauffähige App für *FixMyStreet* Use Case
  - Eintragen von *Defekten*
  - Anzeigen der Defekte auf Karte
  - Daten von GFT lesen / in GFT schreiben

Abgabe / Deliverables

- Lauffähige Version der *FixMyStreet* Web-App
- Converter-Build (GIS Formate konvertieren bzw. in GFT importieren)

Am Ende dieses Sprints hatten wir eine lauffähige Version der *FixMyStreet* Web-App (Kapitel 7). Diese beinhaltete die Anbindung einer Google Fusion Table als Datenbank. Es sind aber noch einige kleine Bugs vorhanden, welche im nächsten Sprint korrigiert werden müssen.

Probleme

In der App sind noch einige Bugs in Bezug auf das Lesen und Schreiben der Daten aus einer Fusion Table vorhanden. Es scheint als gäbe es Probleme mit der internen (in App) und externen ID (in Fusion Talbe) der Datensätze. Zudem wurde bei verschiedenen Tests noch einige Usability-Probleme festgestellt, welche noch behoben werden müssen.

9.5. Sprint 4

Im letzten Sprint ging es noch um das Finalisieren aller Arbeiten. Dies beinhaltete die Behebung der im Sprint 3 gefundenen Probleme in der *FixMyStreet* Web-App. Der grösste Teil der Zeit wurde aber für das Schreiben der Dokumentation eingeplant.

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

Hauptaufgaben / Fokussierung im Sprint

- Finalisierung aller Arbeiten
  - Fertiggestellung des Use Cases *FixMyStreet*
  - Abschluss der Dokumentation
  - Erstellung des Posters A0

Ziele

- Finalisierung des *FixMyStreet* Use Cases
  - Bugs / Usability-Probleme beheben
  - Live-Daten anzeigen
  - Berechtigungen für Benutzer
- Finalisierung Dokumentation der Ergebnisse
  - Converter-Build
  - Use Cases / Examples
  - Google Fusion Tables allgemein

Abgabe / Deliverables

- Fertige Dokumentation
- A0 Poster
- Vollständige Arbeit (Source Code usw.)

Nach diesem Sprint ist die Abgabe der kompletten Arbeit fällig. Dies bedeutet, dass zu diesem Zeitpunkt alle Use Cases lauffähig sind und die Dokumentation abschlossen ist.

10. Projektmonitoring

10.1. Projektverlauf

In Abbildung 10.1 sind alle erledigten Stories mit Start- und Endpunkt auf der Zeitachse abgebildet. Leider werden die Sprints etwas verfälscht dargestellt, da wir einzelne bereits begonnene Stories aus Zeitgründen von einem Sprint in den nächsten verschoben haben (Stories #25, #113 und #186). Diese werden von Redmine dann über mehrere Sprints hinweg dargestellt, was natürlich auch den Sprint künstlich „verlängert“.

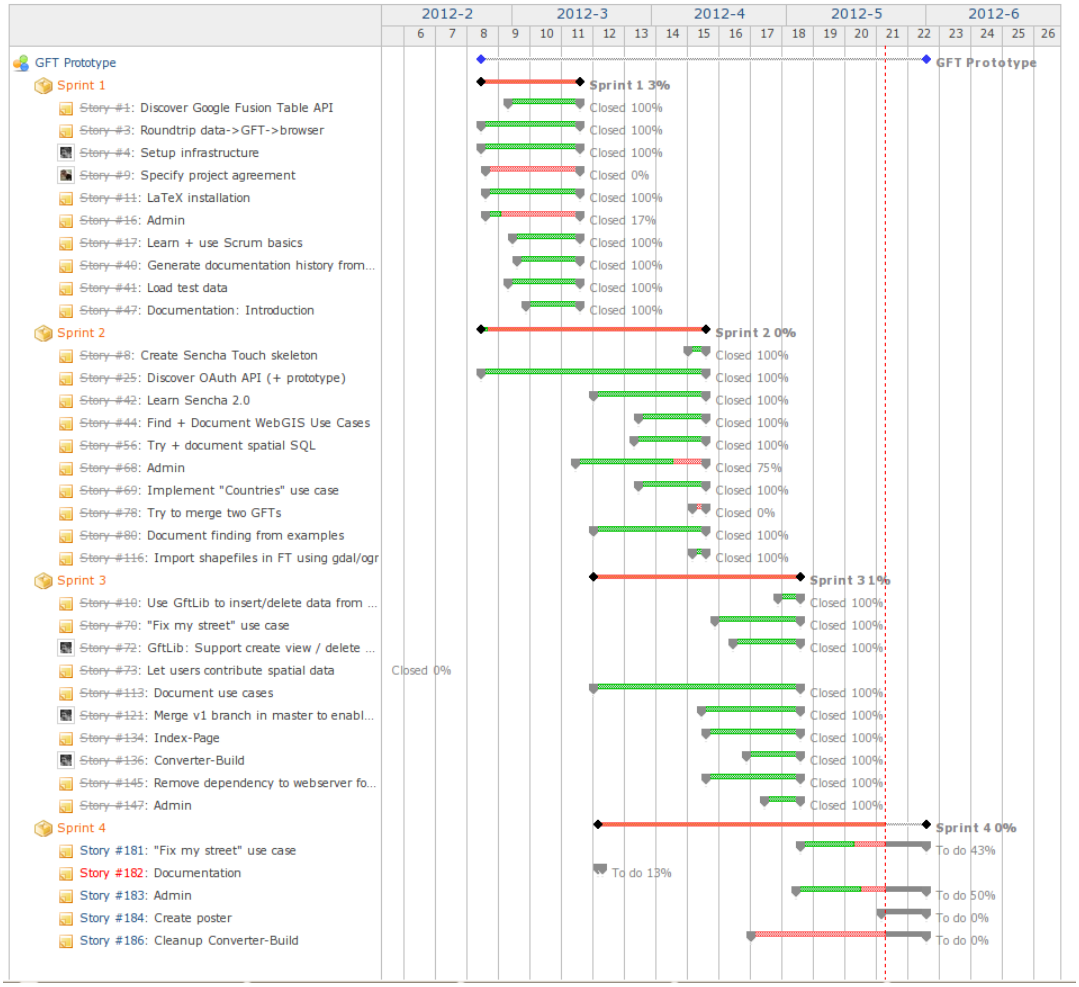


Abbildung 10.1.: Gantt-Diagramm des Projektverlaufs

10.2. Arbeitsaufwand

Wie schon im Kapitel 9 beschrieben, war der vom Modul vorgegebene Aufwand pro Person auf *240 Stunden* festgelegt. Wie in der Tabelle 10.1 ersichtlich haben wir diese Vorgabe beide leicht überschritten.

Person	**Aufwand**
Stefan Oderbolz	255h
Jürg Hunziker	264h

Tabelle 10.1.: Arbeitsaufwand pro Person

10.3. Fazit

Wir haben bezüglich Projektmanagement in diesem Projekt gute Erfahrungen mit Scrum sammeln können. Zuvor kannten wir diese Methodik lediglich aus der Theorie. Jetzt konnten wir die Stärken und Schwächen selbst kennenlernen.

Es ist klar, dass wir Scrum für unsere Bedürfnisse etwas anpassen mussten, gerade weil wir nur zwei Personen im Projektteam waren. Es hat sich bewährt jeweils Stories für neue Features im Backlog zu erfassen, wenn diese gerade aufgetaucht sind. Bei der Planung eines neuen Sprints konnten wir so die bestehenden Stories mit Schätzungen versehen und diese dann auf den Sprint planen.

Dank unserem Tooling mit Redmine hatten wir stets den Überblick über unser Projekt und die noch offenen Stories bzw. deren Tasks. Darin konnten wir auch gleich unsere Zeiterfassung unterbringen, indem wir unsere Aufwände auf die Tasks buchten. Das Projekt war hingegen fast zu kurz, um von den Stories Points zu profitieren. Wir haben über alle 4 Sprints hinweg mit durchschnittlich 12 Points pro Sprint gerechnet.

Teil V.

Anhänge

Inhalt der CD

Verzeichnis	**Beschreibung**
`SenchaDesignerProject/`	Beispielapplikation welche mit dem Sencha Designer wurde
`_DESIGN/`	Grafik-Rohdaten
`_DOCUMENTATION/`	Dokumentation der Arbeit
`_DOCUMENTATION/studienarbeit-gft-soderbol_jhunzike.pdf`	Kompilierte Dokumentation im PDF-Format
`classes/`	PHP Serverklassen
`examples/`	Beispielapplikationen
`lib/`	Verwendete Library-Pakete
`resources/`	Ressourcen, welche auf der Indexseite verwendet werden (CSS, Bilder)
`services/`	PHP Webservices
`test/`	Tests der Use Cases und Libraries
`usecases/`	Implementierte Use Cases (WorldData, FixMyStreet)
`usecases/fixmystreet/`	Use Case FixMyStreet
`usecases/worlddata/`	Use Case WorldData
`index.php`	Indexseite mit Auflistung aller erstellen Applikationen

Glossar

AJAX
Asynchronous JavaScript and XML beschreibt die Möglichkeit via JavaScript Daten von einem Server nachzuladen[7]. 20, 26, 28, 31, 79

Ant Target
Das Buildautomatisierungs-Tool Ant nennt die einzelnen Schritte eines Builds *Target*. Targets sind eine Sammlung von semantisch zusammengehörigen Tasks. Sie können Abhängigkeiten zu anderen Targets aufweisen, welche dann als Vorbedingung zuerst ausgeführt werden[2]. 39

API
Application Programming Interface, Schnittstelle für die Programmierung. v, 3, 8, 9, 13, 15–23, 25, 26, 28, 31–33, 43, 44, 46, 51–53, 78–80, 91–94, 108

Cloud
Als Cloud oder Cloud-Computing (*Wolke*) bezeichnet man die Gesamtheit aller Dienste, welche ortsunabhängig im Internet angeboten werden. Dies können zum Beispiel Datenspeicher, Server oder Datenbanken oder schlicht Rechenleistung sein. Der grosse Vorteil der Cloud ist, dass sie sehr leicht skalierbar ist, so dass man die Leistungen dynamisch an den Bedarf angepassen kann[8]. vi, 2, 3, 6, 7, 11, 34, 37

CRUD
Das Akronym CRUD beschreibt die 4 Standardoperationen einer Datenbank: **C**reate, **R**ead, **U**pdate, **D**elete[9]. 20, 79, 92

CSV
Comma-separated values[10]. 12, 13, 46, 55, 87

Geocodierung
Bei der Geocodierung wird eine Zeichenkette (Adresse, Namen) einer geografischen Position zugewiesen. 21, 22, 86

GIS
Geoinformationssysteme[12]. iv–vi, 2–4, 7, 63, 86–88, 91–95

headless Browser

Bei einem *headless Browser* handelt es sich um einen Browser, welcher ohne grafische Benutzeroberfläche auskommt. Häufig werden sie dazu verwendet, um Serverjobs, welche auf den Browser angewiesen sind, auszuführen. 38

IaaS
Infrastructure as a Service[4]. 7

JSONP
JavaScript Object Notation with Padding[13] beschreibt die Möglichkeit über Domaingrenzen hinweg Daten zu laden. Es ist eine möglich Lösung zu der Same-Origin-Policy Problematik[17]. 20, 35, 79

KML
Keyhole Markup Language[6]. 12, 13, 46, 54, 87, 93

Merge
Bei einem *Merge* werden 2 Dinge zusammengeführt, im Fall von Google Fusion Tables können 2 Tabellen mit einem *Merge* zu einer sogenannten *Merged Table* zusammengeführt werden. 13, 46, 56

MVC
MVC ist ein Architekturmuster aus der Software-Entwicklung, die Abkürzung steht für **M**odel **V**iew **C**ontroller[14]. 76, 81

OAuth
OAuth ist ein offenes Protokoll, das eine standardisierte, sichere API-Autorisierung erlaubt[15]. 9, 16, 26, 33–35, 44, 93, 94, 106

REST
Representational State Transfer[16] ist ein Programmierparadigma, welches besagt, dass sich der Zustand einer Webapplikation als Ressource in Form einer URL beschreiben lässt. Auf eine solche Ressourcen können folgende Befehle angewendet werden: `GET`, `POST`, `PUT`,`PATCH`, `DELETE`, `HEAD` und `OPTIONS`. HTTP ist ein Protokoll welches REST implementiert. 20

SaaS
Software as a Service[18]. 6

UUID
Universally Unique Identifier ist ein Standard für eindeutige Identifikatoren[19]. 80

Web-App

Der Begriff Web-App (von der englischen Kurzform für web application), bezeichnet im allgemeinen Sprachgebrauch Apps für mobile Endgeräte wie Smartphones und Tablet-Computer, die über einen in das Betriebssystem integrierten Browser aus dem Internet geladen und so ohne Installation auf dem mobilen Endgerät genutzt werden können[20]. iv–vi, 8, 63, 65, 66, 76, 78, 79, 81, 94, 95

XML

Extensible Markup Language[11]. 39, 40, 46

Literaturverzeichnis

[1] Alon Halevy, Rebecca Shapley. Google Fusion Tables, 2009. [Online; Stand 19. April 2012]. URL: http://googleresearch.blogspot.com/2009/06/google-fusion-tables.html.

[2] Apache Software Foundation. Apache Ant 1.8.2 Manual — Target, 2012. [Online; Stand 30. Mai 2012]. URL: http://ant.apache.org/manual/targets.html.

[3] Google Developers. Google Fusion Tables API — Developer's Guide, 2012. [Online; Stand 14. Mai 2012]. URL: https://developers.google.com/fusiontables/docs/developers_guide#Geo.

[4] Frank Hardisty and John A. Dutton Sterling Quinn. What is Cloud Computing?, 2012. [Online; Stand 24. Mai 2012]. URL: https://www.e-education.psu.edu/cloudGIS/node/91.

[5] Kathryn Hurley. Data Gathering with Fusion Tables, 2011. [Online; Stand 19. April 2012]. URL: https://developers.google.com/fusiontables/docs/articles/data_gathering.

[6] Wikipedia. Keyhole Markup Language — Wikipedia, Die freie Enzyklopädie, 2011. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Keyhole_Markup_Language&oldid=97341373.

[7] Wikipedia. Ajax (Programmierung) — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Ajax_(Programmierung)&oldid=102964155.

[8] Wikipedia. Cloud-Computing — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 24. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Cloud-Computing&oldid=103606763.

[9] Wikipedia. CRUD — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=CRUD&oldid=103876970.

[10] Wikipedia. CSV (Dateiformat) — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=CSV_(Dateiformat)&oldid=103809247.

[11] Wikipedia. Extensible Markup Language — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=103137038.

[12] Wikipedia. Geoinformationssystem — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Geoinformationssystem&oldid=102409551.

[13] Wikipedia. JavaScript Object Notation — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=103522442.

[14] Wikipedia. Model View Controller — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Model_View_Controller&oldid=103366388.

[15] Wikipedia. OAuth — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=OAuth&oldid=102475591.

[16] Wikipedia. Representational State Transfer — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 20. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=103295993.

[17] Wikipedia. Same-Origin-Policy — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 18. März 2012]. URL: http://de.wikipedia.org/w/index.php?title=Same-Origin-Policy&oldid=96343774.

[18] Wikipedia. Software as a Service — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Software_as_a_Service&oldid=103794812.

[19] Wikipedia. Universally Unique Identifier — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Universally_Unique_Identifier&oldid=103312771.

[20] Wikipedia. Webapp — Wikipedia, Die freie Enzyklopädie, 2012. [Online; Stand 30. Mai 2012]. URL: http://de.wikipedia.org/w/index.php?title=Webapp&oldid=102886692.

Abbildungsverzeichnis

Tabellenverzeichnis