Hochschule für Technik Rapperswil
Studienarbeit

POI Tour – Personalisierter Tourenplaner für Fussgänger

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

Autor:
Fabio Scala

Betreuer:
Prof. Stefan Keller

Experte:
Prof. Stefan Keller

Rapperswil, 16. Dezember 2014 Version 1.0

Abstract

Existierende Kartendienste wie Google Maps erlauben lediglich eine Navigation von A nach B oder die manuelle Planung von Touren. Ziel dieser Arbeit ist die Entwicklung eines parametrisierbaren Routings für Fussgänger entlang bestimmten Points of Interest (POIs). Das zugrundeliegende Problem wird in der Informatik als NP-hart eingestuft und ist somit nicht in nützlicher Frist lösbar. Dieses Routing dient in einem zweiten Schritt als Grundlage für eine HTML5 Webapplikation zur Tourenplanung – POI Tour.

Als allgemeine Datenquelle wurde OpenStreetMap (OSM), ein freies Projekt für Geodaten verwendet. Zudem wurden die Open Source Routing Machine (OSRM) und pgRouting, zwei mit OSM kompatiblen Routing Engines zur Berechnung von Distanzen, in Hinblick auf Performance und Machbarkeit evaluiert. Diese Distanzen zwischen potenziellen POIs dienen als Input für einen Genetischen Algorithmus, welcher eine optimierte Tour hinsichtlich der spezifizierten Parameter generiert.

Das Resultat ist eine moderne HTML5 Webapplikation, mit deren sich Touren anhand verschiedener Eingaben (Interessen, Dauer, ...) vorschlagen lassen.

Die Erkenntnisse und das Resultat dieser Arbeit bestätigen die grundsätzliche Machbarkeit eines solchen Tourenplaners und können als Grundlage für eine komplexere Webapplikation wiederverwendet werden.

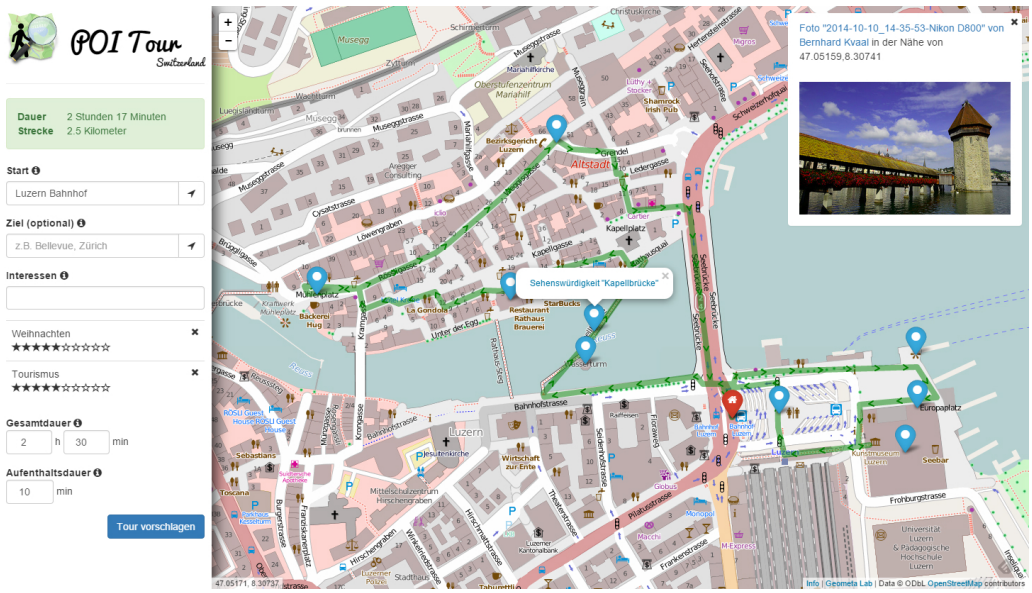
Management Summary

Ausgangslage

Eine automatische Routenplanung von Punkt A nach B ist heute mittels diversen Kartenanbietern wie beispielsweise Google Maps oder Bing wie auch durch Navigationssoftware für Autos bereits möglich. Auch die Anzeige von diversen Points of Interest (POIs) auf der Karte sowie die gezielte Suche von POIs gehört meist zur Standardfunktionalität.
Dieses Projekt (POI Tour) ist ein Versuch, genau diese zwei Aspekte zu vereinen, um so nicht nur eine Navigation von A nach B, sondern eine Tour für Fussgänger entlang wählbarer POI-Kategorien zu ermöglichen.

Ergebnisse

Das Hauptziel, eine moderne Webapplikation zur Erstellung einer individuellen Tour mittels durch den Benutzer spezifizierten Kriterien, wurde erreicht.



Dem Benutzer wird beim Navigieren auf poitour.ch eine Webapplikation gezeigt, welche stark an andere Kartenapplikationen oder der früheren Version von Google Maps erinnert und den Benutzer somit bereits abholt. Ein vor-ausgefülltes und intuitives Formular mit Hilfetexten soll dem technischen Laien den Umgang mit der Webapplikation vereinfachen. Das Formular enthält folgende Eingabemöglichkeiten zur Personalisierung der Tour:

Eingabe	Beschreibung
Start	Der Startpunkt der Tour, egal ob von Zuhause, einer Bahnstation oder dem aktuellen Standort.

Ziel (optional)	Hier gilt dasselbe wie beim Start. Wird das Ziel ausgelassen so wird ein Rundgang erstellt, das heisst zum Start zurückgekehrt.
Interessen	Eine Auswahl aus einer vielfältigen Liste von Interessen wie beispielsweise "Tourismus", "Picknick Plätze", oder ganz aktuell "Weihnachten". Der Benutzer hat zudem die Möglichkeit seine gewählten Interessen unterschiedlich zu gewichten, um die Tour noch stärker nach seinem persönlichen Geschmack zu beeinflussen.
Aufenthaltsdauer	Da kaum jemand eine Sightseeing Tour macht ohne an den diversen Orten zu verweilen oder gar Fotos zu schiessen, kann die Aufenthaltsdauer in Minuten pro POI bestimmt werden.
Gesamtdauer	Die maximale Dauer der Tour in Stunden und Minuten. Diese berücksichtigt ebenfalls die Aufenthaltsdauer und wird niemals überschritten.

Sobald eine Tour dargestellt ist, hat der Benutzer die Möglichkeit durch Klick auf einen beliebigen POI oder sonstigen Punkt der Tour, Umgebungsfotos von anderen Benutzern aus Flickr anzuzeigen. Es hat sich gezeigt, dass diese oftmals selbst durch Touristen geschossen wurden und sich somit sehr gut bei Sehenswürdigkeiten eignen.
Eine Schnittstelle erlaubt es anderen Applikationen, die Funktionalität von POI Tour für eigene Zwecke zu nutzen. Zudem steht das Projekt auf GitHub[1] allen öffentlich unter der Massachusetts Institute of Technology (MIT) Lizenz zur Verfügung und kann somit auch für kommerzielle Zwecke als Basis wiederverwendet werden.

Ausblick

Die Möglichkeiten zu einer potenziellen Weiterentwicklung von POI Tour sind vielfältig. In einem ersten Schritt sollte die Verwendbarkeit auf Mobilgeräten verbessert werden. Zudem wäre es hilfreich, wenn die Webapplikation eine textuelle Beschreibung und eine Exportmöglichkeit der Tour zur Verfügung stellt. In einem zweiten Schritt müsste der verwendete Algorithmus überdacht und eventuell angepasst werden, um komplexere und somit realistischere Szenarien zu ermöglichen. So macht es zum Beispiel selten Sinn, möglichst viele "Restaurants" besuchen zu wollen, sondern viel eher genau eines, und zwar nach einer ganz bestimmten Zeit nach Start der Tour.

[1] https://github.com/fabioscala/poitour

Aufgabenstellung

Personalisierter Tourenplaner für Fussgänger

Studienarbeit von Fabio Scala

Abteilung Informatik, Herbstsemester 2014/15

Ausgangslage

Routenplanung mit Auto-Navis und Google Maps gehört längst zum heutigen Alltag. Diese Routenplaner sind jedoch auf den motorisierten Verkehr eingeschränkt, da die zugrundeliegenden Strassen-Daten mit Autos erfasst wurden. Die Navigation für Velofahrer und Fussgänger ergibt zwangsläufig keine guten Ergebnisse. Das noch nicht so bekannte Projekt OpenStreetMap (OSM) bietet viel detailliertere Informationen auch für Velofahrer und Fussgänger und es gibt bereits auch entsprechende Applikationen (z.B. dieser Routenplaner auf OSM.ch). Dabei ist das Potential, das in diesen 'Open und Big Data' steckt noch lange nicht ausgeschöpft. Nebst den besseren Strassen und Wegen enthält OSM auch hunderte Arten von Points-of-Interests (POIs), wie beispielsweise Sehenswürdigkeiten oder Restaurants.

Aufgabenstellung

Die Idee des personalisierten Tourenplaners ist, nicht "nur" nach dem kürzesten Weg für Fussgänger und Wanderer zu suchen, sondern den - aus persönlicher Sicht - schönsten oder sonstwie optimalsten. Persönlich ist dabei die Zusammenstellung und Gewichtung von POIs, die einen interessieren.

Über ein raffiniertes GUI werden Ausgangspunkt und interessierende POIs ausgewählt. Daraus wird eine Route berechnet, z.B. ein Stadtrundgang entlang Bars oder ein Wanderweg entlang Restaurants und Brunnen. Die Herausforderung bei dieser Arbeit ist einerseits, dass die Routenberechnung performant bleibt, auch wenn nicht alle Kriterien im Voraus bekannt sind. Und andererseits ist die Gestaltung des neuartigen Frontends (User Interface Design) wichtig.

Ziele und Lieferobjekte

Ziele

- Evaluation der Routing Engine (z.B. OSRM und pgRouting)
- Erarbeiten eines Lösungsentwurfs für ein parametrisierbares Routing entlang frei ausgewählter POIs.
- Implementation des personalisierten Tourenplaners für Fussgänger (eigene Website)
- Webdienst, der auch von anderen Webapps genutzt werden kann (REST-Style) optional mit generischer textueller Wegbeschreibung

Lieferobjekte:

- Dokumentation der Erkenntnisse bei der Implementation (Dokumentation)
- Lösungsbeschreibung des personalisierten Tourenplaners
- Eigene Website sowie Webservice (API), der sich z.B. ins Kort Game integrieren lässt
- Optional: Erweiterung eines vorhandenen Routenplaners (Tourpl.ch, OSRM) durch Integration des Fussgänger-Profils
- Demo bei mind. einem potentiellen Anwender (z.B. Kort Game)
- Die vom Studiengang geforderten Lieferobjekte: Dokumentation, Management Summary, Abstract, ev. Kurzvideo)

Dazu kommen die vom Studiengang geforderten Lieferobjekte.

ausgegeben: 14.10.2014; aktualisiert: 15.12.2014

Vorgaben/Rahmenbedingungen

- V.a. Javascript (HTML5, REST), Python
- Der Student entscheidet sich nach Rücksprache mit dem Betreuer für eine SW-Entwicklungsmethodik. Die Meilensteine werden mit dem Betreuer und allfälligen Projektpartnern vereinbart.
- Source Code, Code-Kommentare und Versionsverwaltung, README (inkl. Installationsanleitung) sind in Englisch. Alles andere ist deutsch.
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs bzw. der HSR.

Inhalt der Dokumentation

- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, CD-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind (einheitliche Nummerierung).
- Die Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen (nicht ausschliesslich Wikipedia-Links auflisten).
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Eigenständigkeitserklärung, Nutzungsrechte) gemäss Vorgaben des Studiengangs und Absprache mit dem Betreuer.

Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare), inkl. je einer CD.
- Alle Dokumente und Quellen der erstellten Software auf den 3 CDs (+1 Ex. Abteilung); CD's sauber angeschrieben.

Bewertungsschema

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

- Projektorganisation (Gewichtung ca. 1/5)
- Bericht, Gliederung, Sprache (Gewichtung ca. 1/5)
- Inhalt inkl. Code (Gewichtung ca. 2/5)
- Gesamteindruck inkl. Kommunikation mit Industriepartner (Gewichtung ca. 1/5). Ein wichtiger Bestandteil der Arbeit ist, dass eine lauffähige, getestete Software abgeliefert wird (inkl. getesteter Installationsanleitung).

Beteiligte

Student

Vgl. oben.

Projektpartner

-

Betreuung HSR

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

ausgegeben: 14.10.2014; aktualisiert: 15.12.2014

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmcodeverzeichnis

Liste der Entscheidungen

Pseudocodeverzeichnis

Teil I.

Technischer Bericht

1. Einführung

Eine automatische Routenplanung von Punkt A nach B ist heute mittels diversen Kartenanbietern wie beispielsweise Google Maps oder Bing wie auch durch Navigationssoftware für Autos bereits möglich. Auch die Anzeige von diversen Points of Interest (POIs) auf der Karte sowie die gezielte Suche von POIs gehört meist zur Standardfunktionalität.
Dieses Projekt (POI Tour) ist ein Versuch, genau diese zwei Aspekte zu vereinen, um so nicht nur eine Navigation von A nach B, sondern eine Tour für Fussgänger entlang wählbarer POI-Kategorien zu ermöglichen.

1.1. Vision

Die "Vision" stammt ursprünglich von Frederick Ramm aus der OpenStreetMap (OSM) Community:

»Ich steh hier grad am Bahnhof und muss noch 40 Minuten auf meinen Zug warten – kann Kort mir einen Rund-Spaziergang von 30 Minuten vorschlagen, auf dem ich ein paar nützliche Eintragungen machen kann?« ([12])

OSM ist ein freies Projekt für Geodaten, welche Wiki-artig von jedem laufend erweitert und verbessert werden können. Kort ist eine Webapplikation für Mobilgeräte um spielerisch Daten aus OSM die als fehlerhaft oder unvollständig gelten zu verbessern. Die ursprüngliche Idee sieht also vor, dass Kort eine Tour vorschlagen soll um innert gegebener Zeit möglichst viele OSM Daten zu verbessen.

1.2. Ziele

Hauptziel dieser Arbeit war die Entwicklung einer modernen HTML5 Webapplikation, sodass der Benutzer durch möglichst wenig Eingaben, einen Vorschlag für eine Tour entlang von ihm ausgewählter Interessen erhält. Die ursprüngliche Aufgabenstellung dieser Arbeit galt dabei als Richtlinie und wurde laufend durch eine iterative Vorgehensweise zu folgenden Epics präzisiert:

- Die zwei populären Routing Engines Open Source Routing Machine (OSRM) und pgRouting im Hinblick auf Funktionalität und generelle Eignung für dieses Projekt zu evaluieren und dokumentieren.
- Entwurf eines parametrisierbaren Routing welches die spezifizierten Anforderungen erfüllt.
- Implementation einer HTML5 Webapplikation als Frontend für das obengenannte Routing.
- Eine Webservice Schnittstelle für das Routing, sodass dieses sich aus anderen Applikationen wie beispielsweise Kort nutzen lässt.
- Eine Demo für die Entwickler der Kort Webapplikation.
- Optional: Eine Erweiterung von tourpl.ch um ein Fussgänger-Routing mit OSRM.

1.3. Rahmenbedingungen

Es wurden folgende Rahmenbedingungen, welche sich lediglich auf die Implementation beziehen, durch den Betreuer vorgegeben:

- OSM als Kartenservice und Datenquelle
- Die Enhanced OpenStreetMap Database One (EOSMDBOne) als Schnittstelle zu den OSM Daten
- Python als Programmiersprache
- Deployment der Applikation muss via Web Server Gateway Interface (WSGI) möglich sein
- PostgreSQL als Datenbank
- Weitere Software die bereits in den bestehenden Technolgie-Stack des Instituts für Software IFS passt

Diese sind zugleich als Nicht-funktionale Anforderungen zu verstehen.

1.4. Vorgehen und Aufbau

Die Durchführung dieser Arbeit ist in folgenden groben Teilschritten verlaufen:

1	Einarbeitung	Durch den um eine Woche verspäteten Projektstart wurde mit der selbständigen Einarbeitung in generelle Themen wie OSM und GIS begonnen.
2	Evaluation	Zu Beginn wurden die zwei in Abschnitt 1.2 bereits erwähnten Routing Engines OSRM und pgRouting evaluiert. Diese sowie die Evaluation weiterer Technologien sind in Teil II auf Seite 12 zu finden.
3	Anforderungen	Zeitgleich wurden mögliche Anforderungen sowohl beim Betreuer wie auch durch eine externe Tourismus-Expertin von ZüriOberland Tourismus[1] aufgenommen. Die Anforderungen in Form von User Stories sind in Teil III, Abschnitt 2.1 auf Seite 41 beschrieben.
4	Algorithmus	Das zugrundeliegende NP-harte Problem wurde analysiert und nach möglichen Lösungen gesucht. Die Analyse sowie die Implementation des Prototyps sind in Teil III, Abschnitt 3.1 auf Seite 47 dokumentiert.
5	Implementation	Die Webapplikation wurde iterativ umgesetzt. Ein aktueller Stand der Applikation wurde wöchentlich mit dem Betreuer besprochen und Feedback aufgenommen und umgesetzt.
6	Optimierung	Nachdem eine gewisse Basisfunktionalität vorhanden war, wurden ein Refactoring sowie diverse kleine Optimierungen an Performance, User Interface und am Algorithmus durchgeführt.

[1] Siehe auch: http://www.zuerioberland-tourismus.ch/

2. Umsetzung

2.1. Stand der Technik

2.1.1. Bestehende Webapplikationen

Es verwundet nicht, dass es bereits andere Webapplikationen mit ähnlicher Funktionalität gibt. In diesem Abschnitt wird auf einige davon eingegangen und deren Defizite sowie den Unterschied zu den Zielen dieser Arbeit erläutert.

Fietsroutenplanner

Der "Fietsroutenplanner Zuid-Nederland"[1] in Abb. 2.1, welcher offensichtlich für die Niederlande gemacht ist, kommt dem Endprodukt dieser Arbeit, sowohl was die Funktionalität wie auch was das User Interface angeht sehr nahe. Der Velo-Routenplaner erlaubt die Auswahl diverser Routenprofile wie "schnell", "sportlich", "natürlich", etc. und die totale Distanz. Es können Start und Ziel sowie eigene Zwischenpunkte ausgewählt werden, um eine Route zu erstellen. Anders als beim Ziel dieser Arbeit, geht es bei diesem Routenplaner nicht um POIs, sondern lediglich darum eine Tour mit bestimmter Distanz zu erstellen. Was zudem auffällt, ist dass das Distanzkriterium scheinbar nicht als obere Grenze gilt, da eine Eingabe von 10 km zu Routen von über 15 km führen.
Auch eine textuelle Beschreibung sowie eine Druckansicht ist bei diesem Routenplaner möglich.

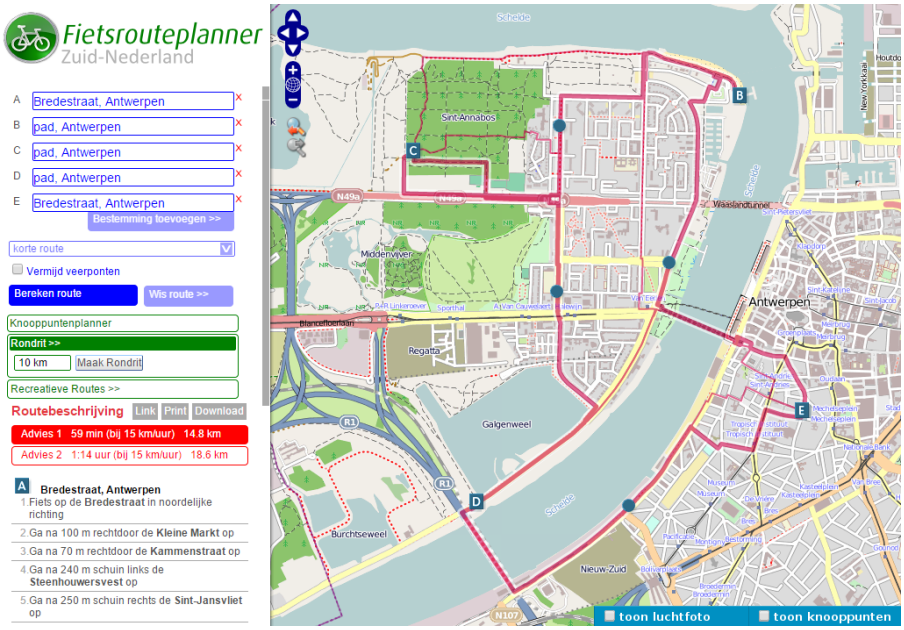
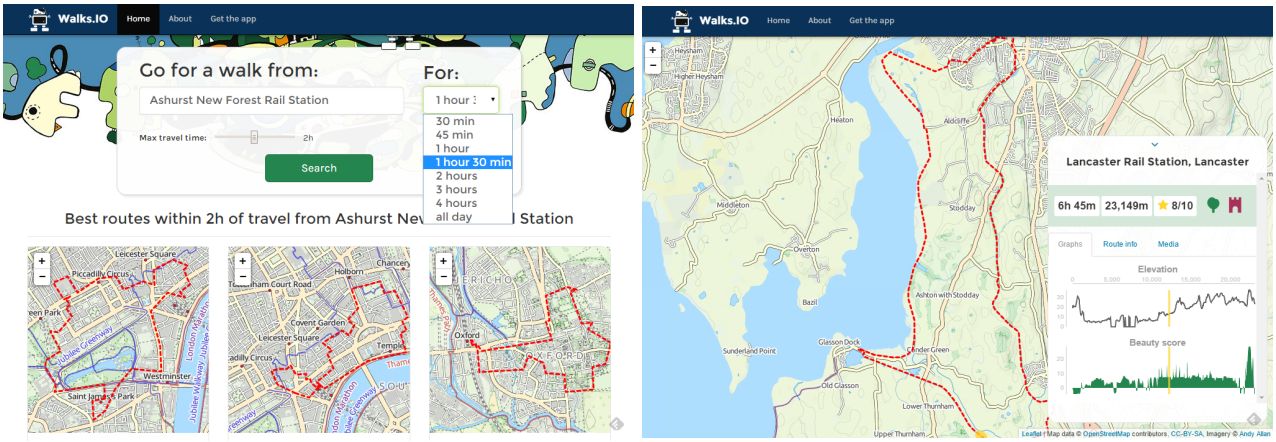


Abbildung 2.1.: Der niederländische "Fietsroutenplanner"

[1] Dt. Velo-Routenplaner Süd-Niederlande, http://fietsrouteplannerzuid.nl/

Walks.IO

Wie der Name bereits suggeriert, handelt es sich bei Walks.IO um einen Routenplaner für Fussgänger. Die Routen sind mittels Artificial Intelligence (AI) Algorithmen vorberechnet [15], was erklärt wieso nur fixe Startpunkte, nämlich ÖV-Haltestellen, zur Auswahl stehen. Walks.IO bezieht nebst OSM Daten auch weitere, staatliche Daten [15] mit in die Berechnung der Routen ein. Anhand dieser Daten werden, wie in Abb. 2.2b ersichtlich, "Beauty-Scores" für die einzelnen Streckenabschnitte erstellt.



(a) Walks.IO Routensuche

(b) Walks.IO Routenansicht

Abbildung 2.2.: Walks.IO

RouteYou

RouteYou[2] ist ein Portal, das den Benutzern erlaubt eigene Routen zu erstellen, mit Informationen und Bildern anzureichern und zu veröffentlichen. Die Qualität dieser Routen ist, da durch Menschen erstellt, relativ hoch. Diese sind jedoch fix und verlieren somit den individuellen bzw. "personalisierbaren" Aspekt.

[2] Siehe auch: http://www.routeyou.com/

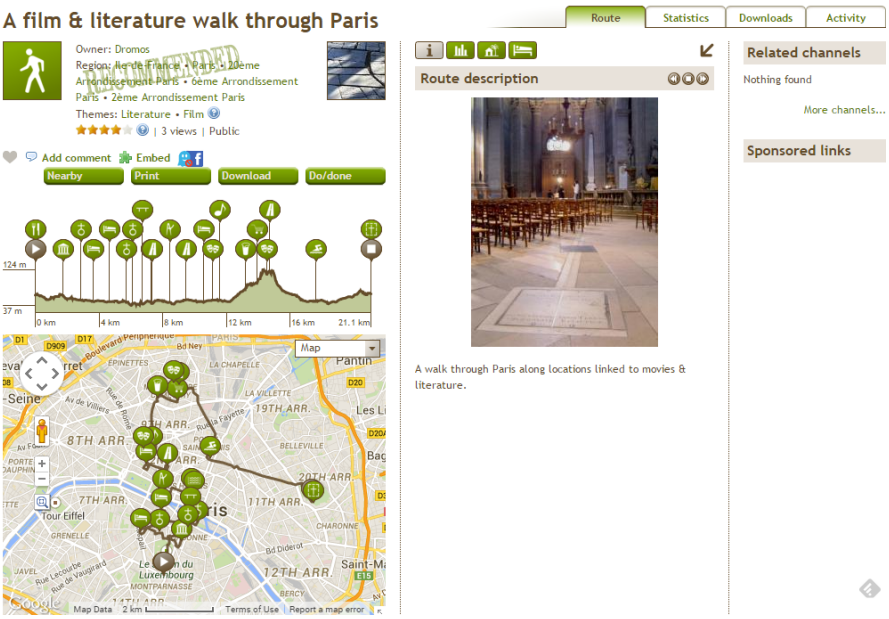


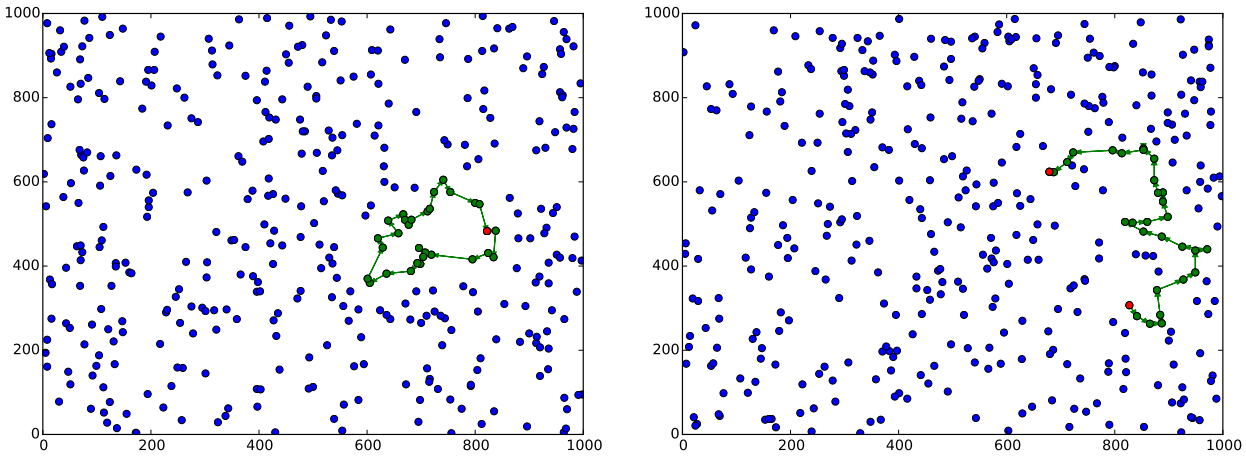
Abbildung 2.3.: RouteYou "Film & Literatur" Tour durch Paris

Fazit – Defizite

Allen in Abschnitt 2.1.1 gelisteten Lösungen fehlt die Möglichkeit, die eigenen Interessen des Benutzers miteinzubringen. Bei RouteYou ist die Wahrscheinlichkeit eine Tour mit den eigenen Interessen zu finden relativ hoch. Diese dauern jedoch möglicherweise viel länger als gewünscht oder befinden sich nicht gar nicht in der eigenen Umgebung.

2.2. Algorithmus

Da es scheinbar nur wenig bestehende Bibliotheken und öffentlich zugängliche Ressourcen im Bezug auf Routing-Probleme gibt, musste der Algorithmus selbst implementiert werden. Nachdem Simulated Annealing sich zur Formulierung des Problems als ungeeignet herausgestellt hat, wurde ein Prototyp eines Genetischen Algorithmus auf der Basis der Arbeit von Piwońska [10] mit Python und NumPy entwickelt. Da dieser, wie in Abb. 2.4 zu sehen, bereits zufriedenstellende Resultate geliefert hat, wurde darauf aufgebaut.



(a) Rundgang (b) Unterschiedlicher Start- und Endpunkt

Abbildung 2.4.: Prototyp des Genetischen Algorithmus

2.3. Webapplikation

In diesem Abschnitt wird kurz auf die wichtigsten Komponenten der Webapplikation eingegangen. Abb. 2.5 zeigt eine Gesamtübersicht von POI Tour.

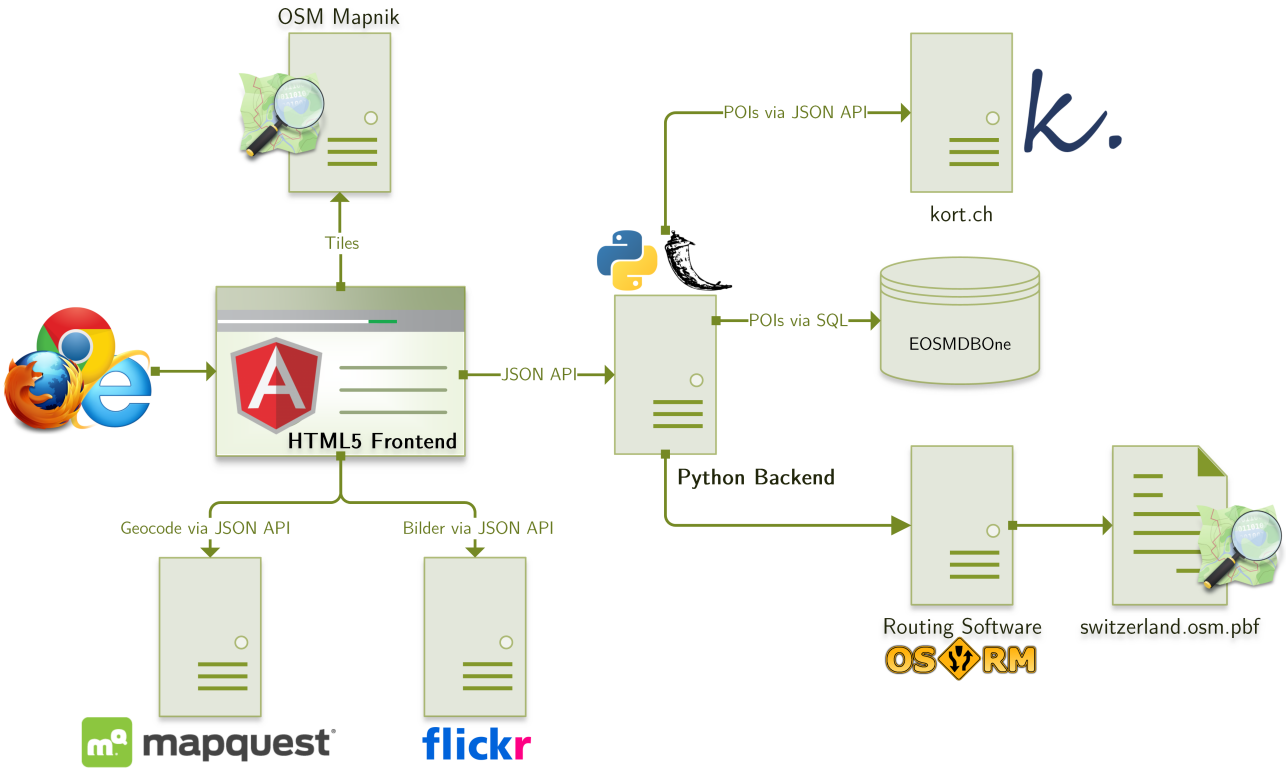


Abbildung 2.5.: Gesamtübersicht POI Tour

2.3.1. Kategorien

Die Kategorien, im User Interface als "Interessen" bezeichnet, werden mittels einer YAML Konfigurationsdatei spezifiziert. Diese werden aus der EOSMDBOne durch OSM "Tags" abgefragt. Der Bezug der Kategorien wurde absichtlich flexibel und erweiterbar gestaltet und beschränkt sich nicht nur auf OSM Daten. Die Implementation von eigenen sogenannten Kategorie-Providern ermöglicht es, weitere Kategorien und POIs zur Verfügung zu stellen. So wurde beispielsweise auch eine Kategorie "Kort-POIs" umgesetzt, welches die Punkte von offenen Kort-Missionen bezieht.

2.3.2. Routing

Da für die Distanzen zwischen den einzelnen POI nicht einfach die Luftlinie verwendet werden darf, dient OSRM als Routing Engine zur Distanzberechnung zwischen diesen Punkten. Zudem wird OSRM bei der Berechnung der schlussendlichen, auf dem User Interface angezeigten Tour eingesetzt.

2.3.3. Frontend

Das HTML5 Frontend wurde mit AngularJS umgesetzt. AngularJS ist ein populäres MVVM JavaScript Framework für Single Page Applications (SPAs). Als weitere Hauptkomponente dient Leaflet, eine JavaScript Bibliothek zur Anzeige von Karten.

2.3.4. Backend

Die Serverseite wurde mit Python (vorgegeben) und Flask, einem schlichten aber dennoch sehr erweiterbaren Webframework umgesetzt.

3. Resultate und Ausblick

Es wurden alle in Abschnitt 1.2 definierten Muss-Ziele der Aufgabenstellung erreicht. Es ist ein relativ gut gelungener Prototyp, in Abb. 3.1 ersichtlich, in Form einer HTML5 Webapplikation entstanden.

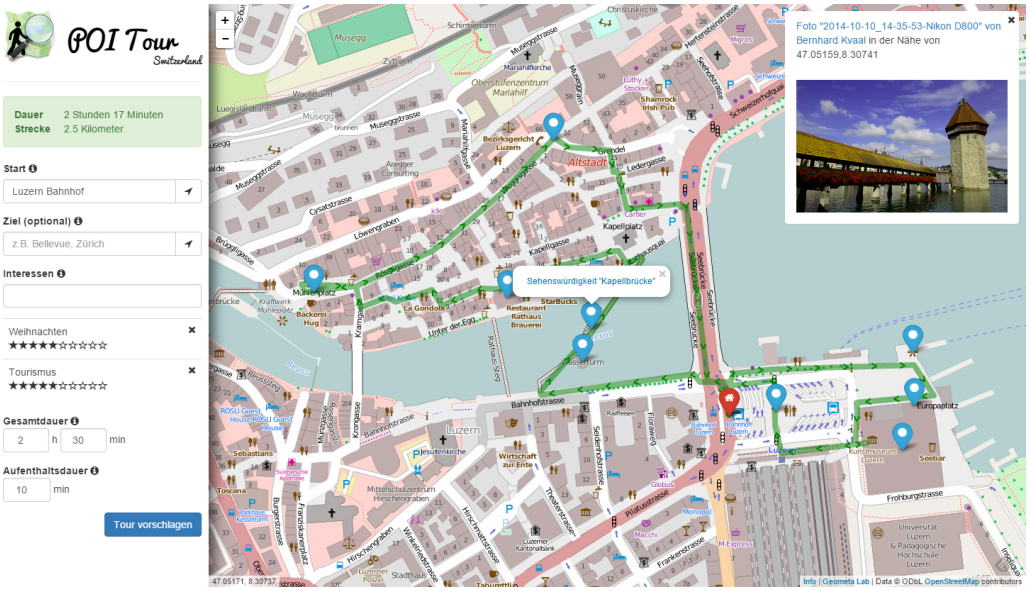


Abbildung 3.1.: Endresultat "POI Tour"

Dem Benutzer wird beim Start ein fast vollständig vor-ausgefülltes Formular präsentiert. Sobald der Startpunkt via Texteingabe, Lokalisierung des Browsers oder Kontextmenu der Karte gesetzt wird, kann POI Tour bereits eine Tour vorschlagen. Eine animierte Ansicht der Tour soll zu einer besseren Übersicht der Laufrichtung führen, ohne die es bei einer längeren Tour mit vielen Kreuzungspunkten oder demselben Retourweg zu Verwirrungen führt.
Sobald eine Tour dargestellt ist, hat der Benutzer die Möglichkeit durch Klick auf einen beliebigen POI oder sonstigen Punkt der Tour, Umgebungsfotos aus Flickr[1] anzuzeigen.
Die optionale Integration des OSRM Fussgänger Profils in tourpl.ch wurde vernachlässigt, dafür konnten mehr der spezifizierten Anforderungen an die Webapplikation umgesetzt werden.
Die Möglichkeiten zu einer potenziellen Weiterentwicklung von POI Tour sind vielfältig. In einem ersten Schritt sollte die Verwendbarkeit auf Mobilgeräten verbessert werden. Zudem wäre es hilfreich, wenn die Webapplikation eine textuelle Beschreibung und eine Exportmöglichkeit analog dem "Fietsroutenplanner" in Abschnitt 2.1.1 zur Verfügung stellt. In einem zweiten Schritt müsste der verwendete Algorithmus überdacht und eventuell angepasst werden, um komplexere und somit realistischere Szenarien zu ermöglichen. So macht es zum Beispiel selten Sinn, möglichst viele "Restaurants" besuchen zu wollen, sondern viel eher genau eines, und zwar nach einer ganz bestimmten Zeit nach Start der Tour.
Weitere Optimierungs- und Weiterentwicklungsmöglichkeiten sind in Teil III, Abschnitt 5.2 auf Seite 72 beschrieben.

[1] Siehe auch: https://www.flickr.com/

3.1. Persönlicher Bericht

Trotz fehlendem Know-how in vielen technischen, algorithmischen und diversen anderen Bereichen der Arbeit, konnte ich durch den von Prof. Stefan Keller offerierten Spielraum bei der Umsetzung stets motiviert bleiben. Glücklicherweise waren mir die durch die Randbedingungen der Arbeit vorgegebenen Programmiersprachen bereits bekannt, wodurch ich mehr Zeit in die Analyse und der Findung eines passenden Algorithmus investieren konnte.
Aus diesem Grund waren die durch diese Vertiefungsarbeit erworbenen ECTS die wohl lehrreichsten während des bisherigen Studiums an der Hochschule für Technik Rapperswil (HSR).

3.2. Dank

In erster Linie möchte ich mich bei meinem Betreuer, Prof. Stefan Keller, für die Unterstützung sowie den obengenannten Spielraum während der Arbeit bedanken. Ein besonderer Dank gilt zudem auch meiner Freundin, Seline Ziegler, die als direkte Testbenutzerin für jegliche Fragen zur Verfügung stand und mich während der gesamten Arbeit seelisch unterstützt hat.

Teil II.

Evaluation Technologien

1. Methodik

Bei der Evaluation von Technologien, insbesondere wenn es sich um Open Source Software handelt, gibt es je nach Programmiersprache und Problem dutzende Möglichkeiten aus denen es auszuwählen gilt. Um die für das Problem und das Projekt am besten geeignete Lösung zu finden ist es deshalb wichtig, eine sorgfältige Evaluation inklusive Bewertung beruhend auf vordefinierten Kriterien durchzuführen und diese am Schluss zur vergleichen um so die geeignete Lösung zu finden. Dabei wurde folgende in Abb. 1.1 aufgezeigte Vorgehensweise [17] gewählt.

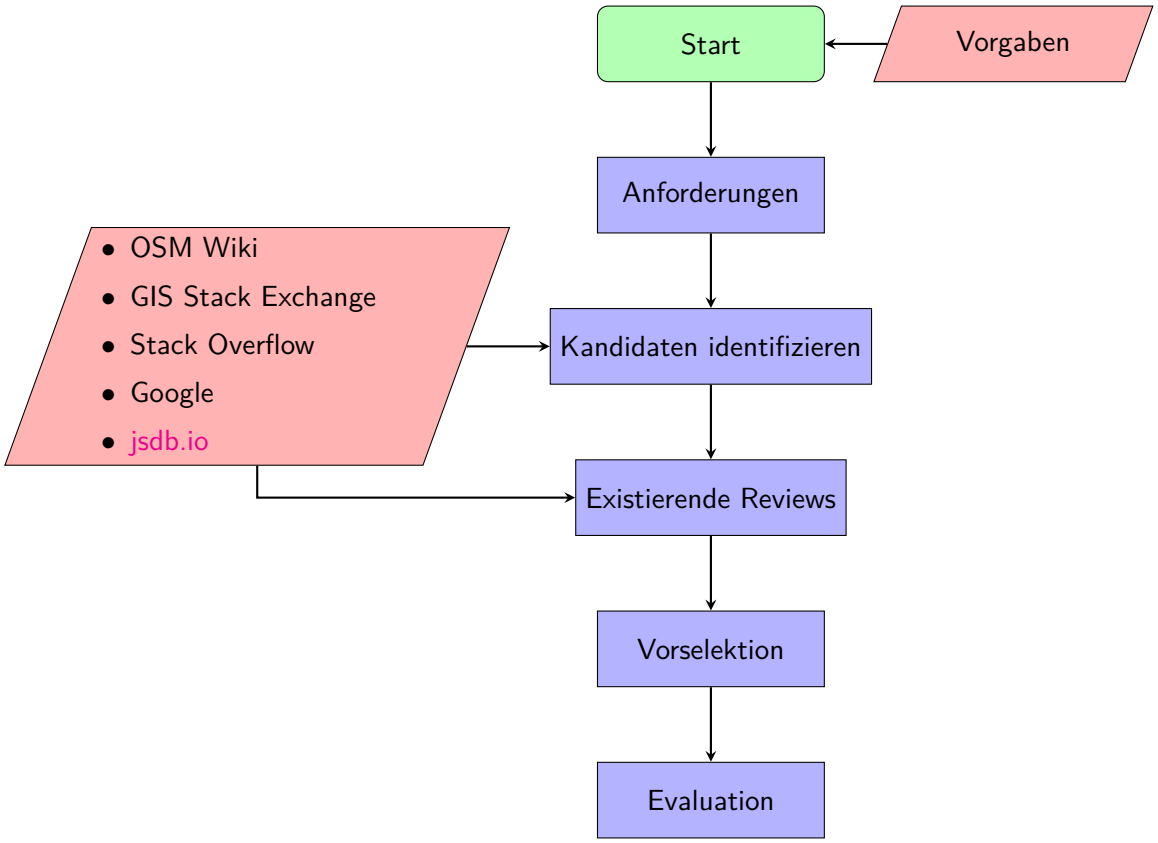


Abbildung 1.1.: Vorgehen bei der Evaluation

1. In einem ersten Schritt werden die Anforderungen an das Projekt bzw. der Applikation geklärt.
2. Es findet eine Recherche statt wobei möglichst viele Kandidaten die potenziell in Frage kommen könnten gesammelt werden.
3. Durch existierende Reviews wird über Vor- und Nachteile sowie offensichtliche Mankos der Lösungen informiert.
4. Aufgrund der gesammelten Informationen wird nun eine grobe Vorselektion durchgeführt um die Anzahl Kandidaten zu reduzieren.

5. Durch eine tiefe Recherche und Selbstevaluation der übrig gebliebenen Kandidaten werden diese anhand der relevanten Bewertungskriterien Bewertet und ausgewählt.

Dabei ist zu berücksichtigen, dass nicht alle Bewertungskriterien für das Projekt gleich relevant bzw. gewichtet sind und deshalb jedes Kriterium eine Gewichtung erhält. Die Summe aus der Multiplikation von Bewertungen und Gewichten ergibt die Endpunktzahl für die jeweiligen Kandidaten. Aufgrund der offensichtlichen K.O. Kriterien wurde auf eine solche abschliessende Gegenüberstellung der Kandidaten verzichtet.

2. Routing Engine

In diesem Kapitel gilt es die zwei vorgegebenen, mit OSM kompatiblen Routing Engines

- OSRM und
- pgRouting,

im Hinblick auf die in Tabelle 2.2 aufgeführten Kriterien zu evaluieren. Die Routing Engine ist der Kern der zu entwickelnden Applikation und wird somit detailliert im Hinblick auf Machbarkeit (Funktionale Anforderungen) und Performance (nicht-funktionale Anforderung) evaluiert.

Kriterium	Beschreibung	Gewichtung
Funktionalität	Wie sehr eignet sich die Routing Engine zur Lösung der Aufgabenstellung. Ist die Aufgabenstellung damit zu bewältigen und mit welchem (Mehr-)Aufwand gegenüber der anderen Lösung.	★★★★★
Features	Wie viel *zusätzliche* Funktionalität und Flexibilität bietet die Engine im Vergleich zur anderen.	★★★☆☆
Performance	Wie schnell wir diejenige Funktionalität die beide Engines gemeinsam besitzen, ausgeführt. Ist in einer Real-Time Applikation wie dieser kritisch.	★★★★☆
Wartbarkeit	Wie gut ist die Dokumentation. Wie verbreitet ist die Lösung, um wenn nötig, Personal mit entsprechendem Know-how anzustellen oder Hilfe auf Internetplattformen zu suchen.	★★☆☆☆
Benutzbarkeit (Entwickler)	Wie lange dauert die Einarbeitung und wie schnell bzw. reibungslos lässt sich danach damit Entwickeln.	★★★★☆

Tabelle 2.2.: Bewertungskriterien Routing Engine

Die Installation und Evaluation der Engines erfolgte dabei auf einer Virtuellen Maschine mit Ubuntu Server 14.04.1 LTS 64-bit als Gastsystem. VirtualBox lief auf einem Windows 8.1 Host mit einem Core i7-2640M und folgenden Performance-relevanten VirtualBox Einstellungen:

Einstellung	Wert
Base Memory	4096 MB
Processors	2 CPUs
Execution Cap	100%
PAE/NX	Enabled
VT-x/AMT-V	Enabled
Nested Paging	Enabled
Video Memory	12 MB
3D Acceleration	Disabled
2D Video Acceleration	Disabled

Tabelle 2.3.: VirutalBox Einstellungen

2.1. Kandidat OSRM

2.1.1. Installation [5, Art. Building-on-Ubuntu]

1. Toolchain

   Die nötigen Abhängigkeiten und Compiler-Pakete installieren.

```
fscala@ubuntu:~$ sudo apt-get install build-essential git cmake pkg-config libprotoc-dev
 ↪  libprotobuf8 protobuf-compiler libprotobuf-dev libosmpbf-dev libpng12-dev libbz2-dev
 ↪  libstxxl-dev libstxxl-doc libstxxl1 libxml2-dev libzip-dev libboost-all-dev lua5.1
 ↪  liblua5.1-0-dev libluabind-dev libluajit-5.1-dev libtbb-dev
# ...
Setting up libtbb-dev (4.2~20130725-1.1ubuntu1) ...
Setting up protobuf-compiler (2.5.0-9ubuntu1) ...
Processing triggers for libc-bin (2.19-0ubuntu6) ...
# ...
```

2. Software

   2.1. Den aktuellen OSRM Source aus dem GitHub Repository klonen.

   2.2. Das Fussweg Routing-Profil von routing.osm.ch herunterladen.

```
fscala@ubuntu:~$ git clone https://github.com/Project-OSRM/osrm-backend.git
Cloning into 'osrm-backend'...
remote: Counting objects: 21811, done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 21811 (delta 10), reused 0 (delta 0)
Receiving objects: 100% (21811/21811), 8.96 MiB | 1.44 MiB/s, done.
Resolving deltas: 100% (13029/13029), done.
Checking connectivity... done.

fscala@ubuntu:~/osrm-backend$ cd osrm-backend

fscala@ubuntu:~/osrm-backend$ git clone https://github.com/sosm/cbf-routing-profiles.git
Cloning into 'cbf-routing-profiles'...
remote: Counting objects: 121, done.
```

```
remote: Total 121 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (121/121), 29.40 KiB | 0 bytes/s, done.
Resolving deltas: 100% (73/73), done.
Checking connectivity... done.
```

3. OSRM kompilieren

```
fscala@ubuntu:~/osrm-backend$ mkdir build
fscala@ubuntu:~/osrm-backend$ cd build
fscala@ubuntu:~/osrm-backend/build$ cmake ..
# ...
-- Configuring done
-- Generating done
-- Build files have been written to: /home/fscala/osrm-backend/build


fscala@ubuntu:~/osrm-backend/build$ make
# ...
[100%] Built target osrm-routed
```

4. Kartendaten vorbereiten
   4.1. Kartendaten der Schweiz herunterladen
   4.2. Kartendaten mittels `osrm-extract` extrahieren

```
fscala@ubuntu:~/osrm-backend/build$ wget
 ↪  http://download.geofabrik.de/europe/switzerland-latest.osm.pbf


fscala@ubuntu:~/osrm-backend/build$
 ↪  LUA_PATH="/home/fscala/osrm-backend/cbf-routing-profiles/lib/?.lua" ./osrm-extract
 ↪  switzerland-latest.osm.pbf --profile=../cbf-routing-profiles/foot-city.lua
[info] Input file: switzerland-latest.osm.pbf
[info] Profile: foot-city.lua
[info] Threads: 2
[info] Using script ../cbf-routing-profiles/foot-city.lua
# ...
[info] Processed 6498342 nodes and 6793805 edges
[info] extraction finished after 145.783s
[info] To prepare the data for routing, run: ./osrm-prepare switzerland-latest.osrm
```

5. Index-Strukturen für OSRM mittels `osrm-prepare` generieren

```
fscala@ubuntu:~/osrm-backend/build$
 ↪  LUA_PATH="/home/fscala/osrm-backend/cbf-routing-profiles/lib/?.lua" ./osrm-prepare
 ↪  switzerland-latest.osrm --profile=../cbf-routing-profiles/foot-city.lua
[info] Input file: switzerland-latest.osrm
[info] Restrictions file: switzerland-latest.osrm.restrictions
[info] Profile: foot-city.lua
# ...
```

```
[info] Preprocessing : 2124.6 seconds
[info] Expansion  : 261271 nodes/sec and 102736 edges/sec
[info] Contraction: 1220.64 nodes/sec and 8011.79 edges/sec
[info] finished preprocessing
```

Achtung

Es ist wichtig genügend Arbeitsspeicher zu besitzen, da es sonst bereits bei kleinen Karten wie "nur" der Schweiz zu einem Abbruch der Verarbeitung mit `osrm-extract` kommt. Sowohl bei 1024 MB wie auch 2048 MB kam es wie zum forcierten Beenden des Prozesses:

```
fscala@ubuntu:~/osrm-backend/build$
 ↪  LUA_PATH="/home/fscala/osrm-backend/cbf-routing-profiles/lib/?.lua" ./osrm-extract
 ↪  switzerland-latest.osm.pbf --profile=../cbf-routing-profiles/foot-city.lua
[info] Input file: switzerland-latest.osm.pbf
[info] Profile: foot-city.lua
[info] Threads: 2
[info] Using script ../cbf-routing-profiles/foot-city.lua
# ...
[info] Parsing in progress..
Killed

fscala@ubuntu:~$ dmesg
[   99.182198] [ 1216]  1000  1216   832288   488258    1517   257385             0
 ↪  osrm-extract
[   99.182200] Out of memory: Kill process 1216 (osrm-extract) score 965 or sacrifice
 ↪  child
[   99.182377] Killed process 1216 (osrm-extract) total-vm:3329152kB, anon-rss:1953032kB,
 ↪  file-rss:0kB
```

Erst bei 4096 MB konnte das nur 213 MB grosse Kartenmaterial der Schweiz (`switzerland-latest.osm`) extrahiert werden.

6. OSRM starten

```
fscala@ubuntu:~/osrm-backend/build$ ./osrm-routed switzerland-latest.osrm
[warn] ./osrm-routed could not be locked to RAM
[info] starting up engines, v0.4.3, compiled at Sep 26 2014, 14:12:41
[info] HSGR file:       "switzerland-latest.osrm.hsgr"
[info] loading graph data
[info] loading graph from switzerland-latest.osrm.hsgr
[info] number_of_nodes: 2555263, number_of_edges: 16771672
[info] loaded 2555263 nodes and 16771672 edges
# ...
[info] http 1.1 compression handled by zlib version 1.2.8
[info] running and waiting for requests
```

7. Test

   Folgende Beispielanfrage mit den Koordinaten der HSR im Browser absetzen um sicherzustellen, das OSRM läuft und ansprechbar ist.

   http://127.0.0.1:5000/locate?loc=47.223461,8.817441

Produziert folgende Ausgabe im zuvor gestarteten `osrm-routed` Prozess:

```
[info] 27-09-2014 21:37:04 10.0.2.2 - Mozilla/5.0 (Windows NT 6.3; WOW64)
 ↪  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
 ↪  /locate?loc=47.223461,8.817441
[info] 27-09-2014 21:37:04 10.0.2.2 - Mozilla/5.0 (Windows NT 6.3; WOW64)
 ↪  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
 ↪  /favicon.ico
```

und folgende Antwort im Browser:

```
{"mapped_coordinate":[47.223457,8.817451],"status":0}
```

2.1.2. Funktionalität und Features

Die Kernfunktionalität von OSRM beschränkt sich hauptsächlich auf die nachfolgend beschriebenen vier Funktionen [5, Art. Server-api].

Nearest Node (locate)

Lokalisierung des nächsten OSM Nodes der angegebenen Koordinaten.
Beispiel-Aufruf mit den Koordinaten des HSR Gebäudes:
http://127.0.0.1:5000/locate?loc=47.223461,8.817441
Führt zu folgender Antwort:

```
{
  "mapped_coordinate": [
  47.223457,
  8.817451
  ],
  "status": 0
}
```

Nearest Point (nearest)

Lokalisierung der nächsten Strasse bzw. dessen zugehöriger Punkt.
Beispiel mit Koordinaten des HSR Gebäudes:
http://127.0.0.1:5000/nearest?loc=47.223461,8.817441
Führt zu folgender Antwort:

```
{
  "name": "",
  "mapped_coordinate": [
  47.223454,
  8.817449
  ],
  "status": 0
}
```

Routing (viaroute)

Routing zwischen zwei oder mehreren (via) Punkten inkl. Instruktionen, Distanzen und weitere, aus der Server API[1] entnehmbaren Details der Route.
Beispielaufruf für ein Routing zwischen der Pestalozziwiese und dem Starbucks im Central, Zürich:
http://127.0.0.1:5000/viaroute?loc=47.375524,8.538751&loc=47.376576,8.544044&instructions=true
Führt zu folgender (verkürzter) Antwort:


```
{
  "hint_data": {
    "locations": [
    "YBMPAGYTDwAAAAAAMAAAABsAAAAAAAAAAAAAAP____945NICbkqCAAAAAAA",
    "-k0RAAJOEQAAAAAACwAAAAAAAABuAAAAAAAAAJrQFQDL6NICFV-CAAcAAAA"
    ],
    "checksum": 0
  },
  "route_name": [
  "",
  "Bahnhofstrasse"
  ],
  "via_indices": [
  0,
  31
  ],
  "found_alternative": false,
  "route_summary": {
    "end_point": "",
    "start_point": "",
    "total_time": 162,
    "total_distance": 539
  },
  "via_points": [
  [
  47.375481,
  8.538734
  ],
  [
  47.376587,
  8.544021
  ]
  ],
  "route_instructions": [
    // Instruktionen...
  ],
  "route_geometry": "ofqjyA{edhO|AwKcMeFiDwAmDyAac@iQcDoAcYeLf@wC‘@cCpAaHYgQBaU}@qHTaYh
   ↪  @kQHgVNeMLmOtB_j@rBaJhAa@vAkElDcB]_C}@_Ei@eCgAqFIoGl@eC‘AwD",
"status_message": "Found route between points",
"status": 0
}
```


[1] https://github.com/Project-OSRM/osrm-backend/wiki/Server-api

Distanzmatrix (table)

Seit Mitte 2014 kann OSRM auch Distanzmatrizen berechnen. Dadurch entfallen hauptsächlich HTTP Request-Overheads sowie detaillierte Berechnungen. Der Nachteil dabei sind die fehlenden Metadaten, OSRM gibt lediglich eine Matrix mit Ganzzahl-Zeiten zurück.

2.1.3. Performance

Die Dauer der einzelnen in Abschnitt 2.1.1 genannten Aufbereitungsschritte wurde mit `/usr/bin/time`, der erweiterten GNU Version von `time` gemessen. Dier hier relevante Zeit die es mit anderen Routing Engines zu vergleichen gilt sind die 35 Minuten beim `osrm-prepare`.

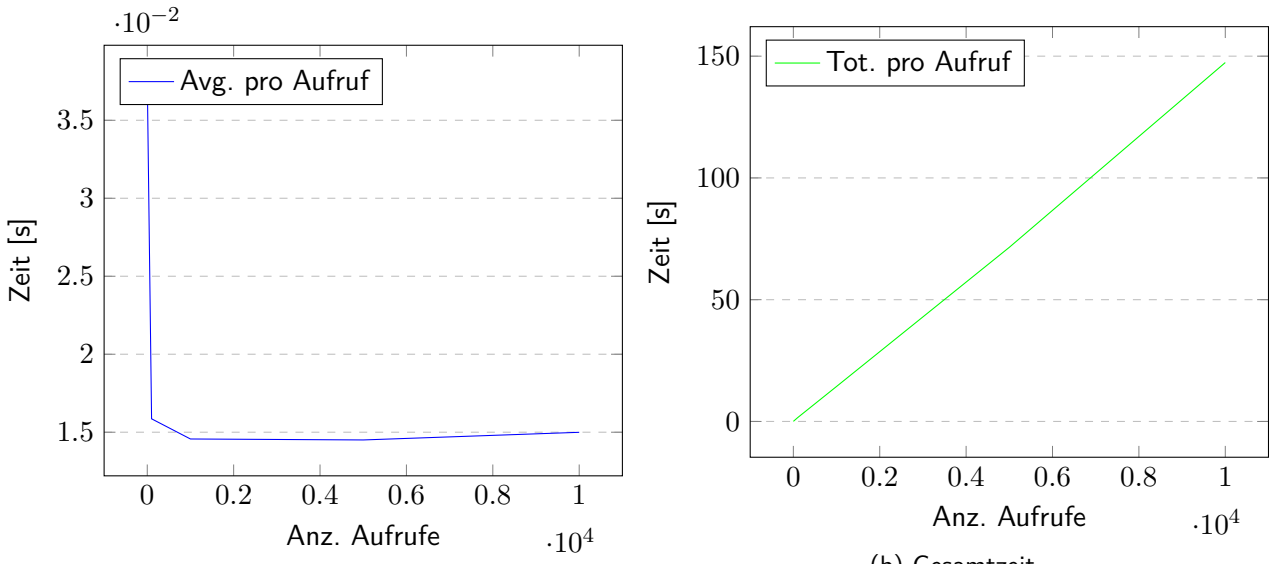
osrm-extract

```
217.87 user
9.77 system
2:25.83 elapsed
```

osrm-prepare

```
4153.85 user
34.86 system
35:24.61 elapsed
```

Um die Performance der `viaroute` Aufrufe zu messen wurde ein Python-Script geschrieben und mit diversen Parametern ausgeführt. Das Python-Script verwendet bei jedem `viaroute` Aufruf zwei zufällige Koordinatenpaare in der Umgebung von Zürich um ein 1:1 Caching der Anfragen zu verhindern. Zwischen den Aufrufen wurde der OSRM Server jeweils neu gestartet um weitere Caching-Effekte zu vermeiden.
In Abb. 2.1a sieht man die Durchschnittszeiten von jeweils 1, 100, 1000 und 10'000 Aufrufen. Es ist klar zu erkennen, dass bei nur einem Aufruf pro Session und somit gleichzeitig beim ersten Aufruf überhaupt, ein gewisser Initialisierungs-Overhead besteht. Bei den übrigen blieb die Zeit relativ konstant und wurde nicht etwa schneller (Caching) oder langsamer.
Abb. 2.1b bestätigt die vorherige Vermutung, dass sich die Performance linear verhält.



(a) Durchschnittszeit

(b) Gesamtzeit

Abbildung 2.1.: Performance sequenzieller `viaroute` Aufrufe

Auch die Anzahl nicht erfolgreicher Routenberechnungen scheint sich wie in Abb. 2.2 zu sehen ist, linear zu Verhalten und hängt mit grösster Wahrscheinlichkeit von dem Kartenmaterial sowie den zufällig gewählten Start- und Endpunkten ab.

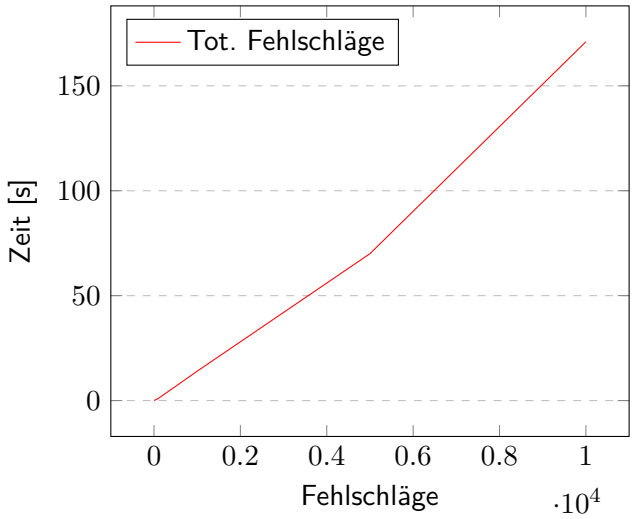
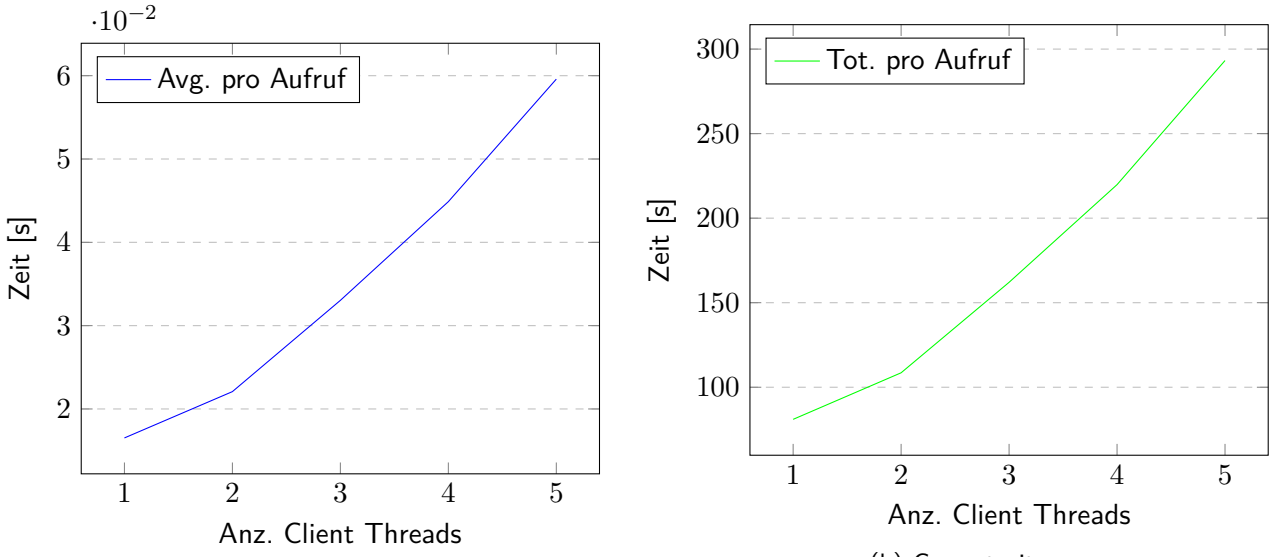


Abbildung 2.2.: Fehlgeschlagene `viaroute` Aufrufe

Da wir nun wissen, dass sich die Performance linear verhält, fixieren wir die Anzahl Aufrufe auf 5'000 und variieren die Anzahl gleichzeitiger Zugriffe mittels Threading im Python-Script. Sofort ist sowohl in Abb. 2.3a wie auch in Abb. 2.3b ersichtlich, dass die Performance trotz gleichbleibender Gesamtanfragen mit steigender Anzahl paralleler Anfragen degradiert.



(a) Durchschnittszeit

(b) Gesamtzeit

Abbildung 2.3.: Performance paralleler `viaroute` Aufrufe

2.1.4. Bewertung

Kriterium	Bewertung	Bemerkung
Funktionalität	★★★★★	Da mit tourpl.ch bereits eine relativ ähnliche Lösung existiert, die ebenfalls auf dem TSP Problem basiert, ist eine grundsätzliche Machbarkeit nahezu gewährleistet.
Features	★★☆☆☆	OSRM bietet grundsätzlich nur statisches Routing.
Performance	★★★★★	Mit einer durchschnittlichen End-to-End Zeit von 60 ms bei 5 simultanen Anfragen Clients in einer virtuellen Maschine ist OSRM auch unter erhöhter Last sehr schnell.
Wartbarkeit	★★★☆☆	Hinter OSRM steckt lediglich eine kleine Community. Es gibt zur Zeit auch keine bekannten Bücher, in denen OSRM erwähnt wird[2].
Benutzbarkeit (Entwickler)	★★★★★	OSRM ist auf Ubuntu relativ einfach zu installieren. Auch die Verwendung der API ist aufgrund deren Überschaubarkeit simpel und effizient.

Tabelle 2.5.: Bewertung von OSRM

2.2. Kandidat pgRouting

2.2.1. Installation

1. PostgreSQL installieren

[2] Suche bei Amazon

```
fscala@ubuntu:~$ sudo apt-get install postgresql postgresql-client
# ...
* Starting PostgreSQL 9.3 database server
 ↪  [ OK ]
Setting up postgresql (9.3+154) ...
Processing triggers for libc-bin (2.19-0ubuntu6) ...
# ...
```

2. PostgreSQL Administrator (Benutzer: postgres) Passwort setzen

```
fscala@ubuntu:~$ sudo -u postgres psql postgres
psql (9.3.5)
Type "help" for help.
```

```
postgres=# \password postgres
Enter new password:
Enter it again:

postgres-# \q
```

3. PostGIS und pgRouting Erweiterungen für PostgreSQL installieren
   - pgRouting Repository hinzufügen

```
fscala@ubuntu:~$ sudo add-apt-repository ppa:georepublic/pgrouting
# ...
OK

fscala@ubuntu:~$ sudo apt-get update
# ...
Fetched 521 kB in 12s (40.8 kB/s)
Reading package lists... Done
```

   - Pakete installieren

```
fscala@ubuntu:~$ sudo apt-get install postgresql-contrib postgis
 ↪  postgresql-9.3-postgis-2.1 postgresql-9.3-pgrouting
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
# ...
```

4. Neue Datenbank und Benutzer einrichten

```
fscala@ubuntu:~$ psql -U postgres -h localhost
Password for user postgres:
psql (9.3.5)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.
```

```
postgres=# CREATE ROLE fscala WITH LOGIN ENCRYPTED PASSWORD 'fscala';
CREATE ROLE

postgres=# CREATE DATABASE pgrouting_eval WITH OWNER fscala ENCODING 'UTF8';
CREATE DATABASE


postgres=# \c pgrouting_eval
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
You are now connected to database "pgrouting_eval" as user "postgres".


pgrouting_eval=# CREATE EXTENSION postgis;
CREATE EXTENSION

pgrouting_eval=# CREATE EXTENSION pgrouting;
CREATE EXTENSION
pgrouting_eval=# \q
```

5. osm2pgrouting installieren

```
fscala@ubuntu:~$ sudo apt-get install libpq-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
# ...
Processing triggers for libc-bin (2.19-0ubuntu6) ...

fscala@ubuntu:~$ git clone https://github.com/pgRouting/osm2pgrouting.git
Cloning into 'osm2pgrouting'...
# ...
Checking connectivity... done.

fscala@ubuntu:~$ cd osm2pgrouting

fscala@ubuntu:~/osm2pgrouting$ cmake -H. -Bbuild
-- The C compiler identification is GNU 4.8.2
# -- ...
-- Configuring done
-- Generating done
-- Build files have been written to: /home/fscala/osm2pgrouting/build

fscala@ubuntu:~/osm2pgrouting$ cd build/

fscala@ubuntu:~/osm2pgrouting/build$ make
```

```
Scanning dependencies of target osm2pgrouting
[  7%] Building CXX object CMakeFiles/osm2pgrouting.dir/src/Class.cpp.o
# [...%] ...
[100%] Building CXX object CMakeFiles/osm2pgrouting.dir/src/math_functions.cpp.o
Linking CXX executable osm2pgrouting
[100%] Built target osm2pgrouting

fscala@ubuntu:~/osm2pgrouting/build$ sudo make install
[100%] Built target osm2pgrouting
Install the project...
-- Install configuration: ""
-- Installing: /usr/share/bin/osm2pgrouting
-- Installing: ...
# -- ...
```

6. OSM Kartendaten importieren
   - Kartendaten der Schweiz herunterladen und mit Osmosis entpacken

```
fscala@ubuntu:~/osm2pgrouting/build$ wget
 ↪  http://download.geofabrik.de/europe/switzerland-latest.osm.pbf

fscala@ubuntu:~/osm2pgrouting/build$ sudo apt-get install osmosis
Reading package lists... Done
# ...
done.

fscala@ubuntu:~/osm2pgrouting/build$ osmosis --read-pbf switzerland-latest.osm.pbf
 ↪  --write-xml switzerland-latest.osm
Oct 01, 2014 11:07:42 PM org.openstreetmap.osmosis.core.Osmosis run
INFO: Osmosis Version 0.40.1
# ...
INFO: Total execution time: 150845 milliseconds.
```

   - Kartendaten in die pgRouting Datenbank importieren

```
fscala@ubuntu:~/osm2pgrouting/build$ /usr/bin/time ./osm2pgrouting -file
 ↪  switzerland-latest.osm -conf ../mapconfig.xml -dbname pgrouting_eval -user
 ↪  postgres -passwd fscala -clean
host=127.0.0.1 user=postgres dbname=pgrouting_eval port=5432 password=fscala
connection success
Trying to load config file ../mapconfig.xml
Trying to parse config
Trying to load data
Trying to parse data
unknown maxspeed value: CH:urban
# unknown maxspeed value: ...
Split ways
Dropping tables...
Creating tables...
Nodes table created
2create ways failed:
Types table created
```

```
Way_tag table created
Relations table created
Relation_ways table created
Classes table created
Adding tag types and classes to database...
Adding relations to database...
Adding nodes to database...
Adding ways to database...
Creating topology...
NOTICE:  PROCESSING:
NOTICE:  pgr_createTopology('ways', 1e-05, 'the_geom', 'gid', 'source', 'target',
 ↪  'true')
NOTICE:  Performing checks, pelase wait .....
NOTICE:  Creating Topology, Please wait...
NOTICE:  1000 edges processed
# NOTICE:  ... edges processed
NOTICE:  -------------> TOPOLOGY CREATED FOR  1659689 edges
NOTICE:  Rows with NULL geometry or NULL id: 0
NOTICE:  Vertices table for table public.ways is: public.ways_vertices_pgr
NOTICE:  ----------------------------------------------
Create Topology success
size of streets: 773031
size of splitted ways : 1659689
finished
```

Actung!

Der osm2pgrouting Importer benötigte beim Kartenmaterial der Schweiz, welches als XML Datei ungefähr 4 GB gross ist, etwa diesselbe Menge and Arbeitsspeicher. Die für die virtuelle Maschine konfigurierten 4096 MB wurden zu 95% gefüllt.

Es kann zudem zu folgender Meldung beim Import kommen:

```
Creating topology...
NOTICE:  PROCESSING:
NOTICE:  pgr_createTopology('ways', 1e-05, 'the_geom', 'gid', 'source', 'target',
 ↪  'true')
NOTICE:  Performing checks, pelase wait .....
NOTICE:  -------> ways not found
```

Die Daten wurden zwar importiert, doch die Topologie, welche für die Abfragen benötigt wird konnte nicht erstellt werden. In diesem Fall sollte der Import mit dem `postgres` Benutzer ausgeführt werden[3].

2.2.2. Funktionalität und Features

Die Routing Features von pgRouting sind sehr umfangreich und gehen weit über das, was für simples Routing benötigt wird hinaus. Es unterstützt diverse "Shortest Path" Algorithmen, analytische Funktionen sowie

[3] Siehe auch:
https://github.com/pgRouting/osm2pgrouting/issues/32 und
https://github.com/pgRouting/osm2pgrouting/issues/28

isochronsiche Flächenberechnungen.

Funktion	Beschreibung
Analytisch	
`pgr_analyzeGraph`	Analysiert die Daten und findet Sackgassen, Lücken, isolierte Segmente, Überlappungen sowie "Ring Geometrien".
`pgr_analyzeOneway`	Findet potenzielle Probleme bei Einbahnstrassen.
Routing	
`pgr_apspJohnson`	All Pairs Shortest Path, Johnson's Algorithm Siehe `pgr_apspWarshall` mit leicht angepassten Randbedingungen.
`pgr_apspWarshall`	All Pairs Shortest Path, Floyd-Warshall Algorithm Kürzester Pfad der durch alles Nodes geht. Könnte sich eignen um indem man pro Iteration ein POI in der nähe hinzufügt bis die gewünschte Zeit erreicht wird. Löst den TSP Spezialfall nicht und garantiert nicht, dass möglichst viele besucht werden.
`pgr_astar`	Shortest Path A* Kürzester Pfad zwischen zwei Nodes. Modifizierte Version des Dijkstra Algorithmus.
`pgr_bdAstar`	Bi-directional A* Shortest Path Dasselbe wie `pgr_bdAstar`. Diese Version sucht von beiden Seiten und terminiert in der wenn sie sich treffen.
`pgr_bdDijkstra`	Bi-directional Dijkstra Shortest Path Dasselbe wie `pgr_dijkstra`. Diese Version sucht von beiden Seiten und terminiert in der wenn sie sich treffen.
`pgr_dijkstra`	Shortest Path Dijkstra Kürzester Pfad zwischen zwei Nodes. Der wahrscheinlich am häufigsten verwendete pgRouting Algortihmus.
`pgr_kDijkstra`	Mutliple destination Shortest Path Dijkstra Dasselbe wie `pgr_dijkstra` jedoch mit einem Start- und meheren Endpunkten. Es werden alle Routen zwischen dem Start- und den verschiedenen Endpunkten berechnet. Kann evtl. zu einem Performance-Gewinn führen gegenüber einzelnen Abfragen.
`pgr_ksp`	K-Shortest Path Berechnet mehrere (k) kürzeste Pfade zwischen den zwei Nodes. Hilfreich wenn man etwas wie eine alternative Route benötigt.
`pgr_tsp`	TSP Verwendet Simulated Annealing um das TSP zu lösen.
`pgr_trsp`	Turn Restriction Shortest Path (TRSP) Basiert ebenfalls auf Dijkstra, kann jedoch mit Abbiege- und Wendeveboten umgehen um die Route zu berechnen.
Isochronische Flächen	
`pgr_drivingDistance`	Driving Distance Berechnet alle von einem Punkt aus erreichbaren Nodes innerhalb einer bestimmten Distanz und bildet somit eine Fläche. Wird verwendet um z.B. Einzugs- und Versorgungsgebiete zu bestimmen (z.B. welche Filiale ist zuständig für welches Gebiet)

Tabelle 2.7.: pgRouting Kernfunktionalität

In Tabelle 2.7 sind alle relevanten[4] Funktionen von pgRouting aufgelistet.
Um die korrekte Funktionsfähigkeit der Routing-Resultate zu verifizieren wurde QGIS (Quantum GIS) verwendet, ein mächtiges Tool welches Layer aus diversen Datenquellen inklusive PostGIS darstellen kann. Eine Beispielroute in Abschnitt 2.2.3 verwendeten Abfrage ist in Abb. 2.4 erischtlich.

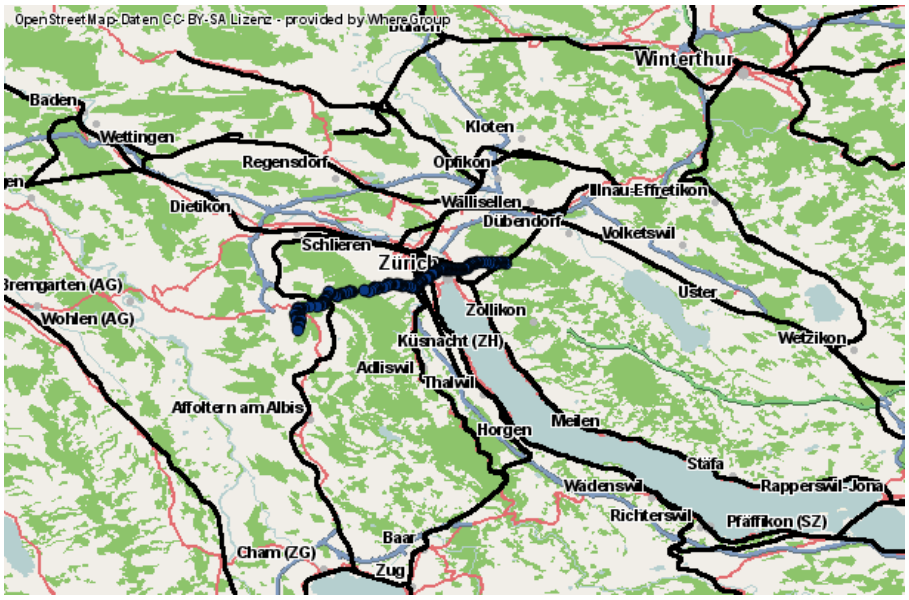


Abbildung 2.4.: pgRouting Testroute in QGIS

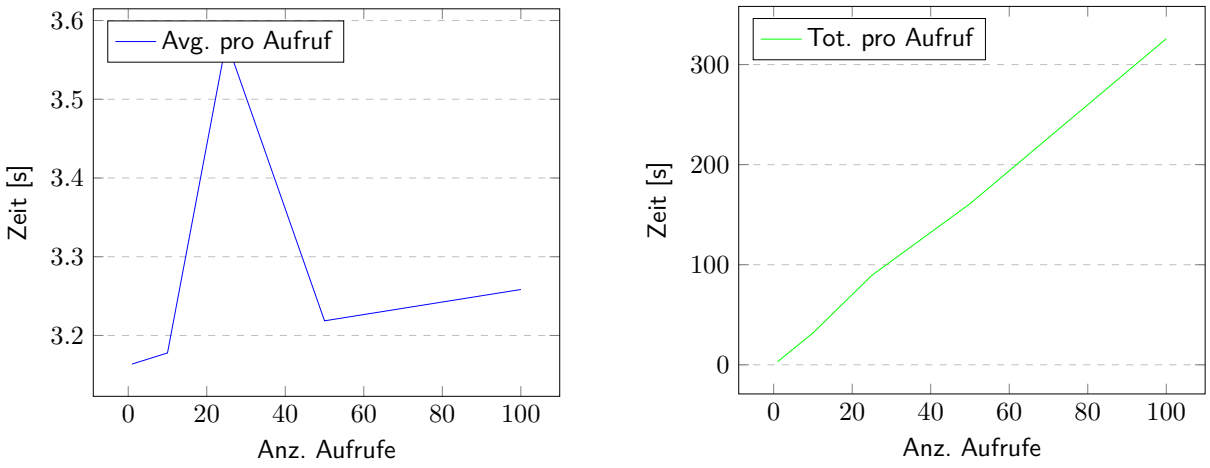
2.2.3. Performance

Auch bei pgRouting wurde die Zeit für den Import-Prozess gemessen. Erstaunlicherweise dauerte diese mit über einer Stunde mehr als doppelt so lange wie bei OSRM, obwohl letzteres noch optimiert und indexiert. Dies ist vermutlich auf den Overhead seitens PostgreSQL sowie die Erstellung der Topologie zurückzuführen.

```
249.05 user
180.29 system
1:11:52 elapsed
```

Eine für SQL-Abfragen angepasste Version des in Abschnitt 2.1.3 verwendeten Python-Scripts wurde für die Performancemessung der pgRouting Funktion `pgr_dijkstra` eingesetzt. Die Koordinaten wurden nach wie vor zufällig aus derselben Region Zürich gewählt. Auch hier wurde der Server, in diesem Fall der PostgreSQL Server, zwischen den Tests neu gestartet.
In Abb. 2.5 sieht man die Durchschnittszeiten von jeweils 1, 10, 25, 50 sowie 100 Aufrufen. Da die Routenberechnung um ein vielfaches langsamer als bei OSRM zu sein scheint, wurde aus Zeitgründen nicht mit mehr Samples, wie etwa bei OSRM mit 10'000, gemessen, und es entsteht somit ein Bias durch die zufälligen Koordinatenpaare. Diese statistische Ungenauigkeit ist sehr gut durch den Ausschlag in Abb. 2.5a zu erkennen.
Insgesamt scheint sich jedoch auch bei pgRouting die Performance linear zu Verhalten, was wiederrum in Abb. 2.5b zu sehen ist.
Bemerkenswert ist ebenfalls, dass bei pgRouting alle Abfragen zu einem Resultat geführt haben, nicht etwa wie bei OSRM.

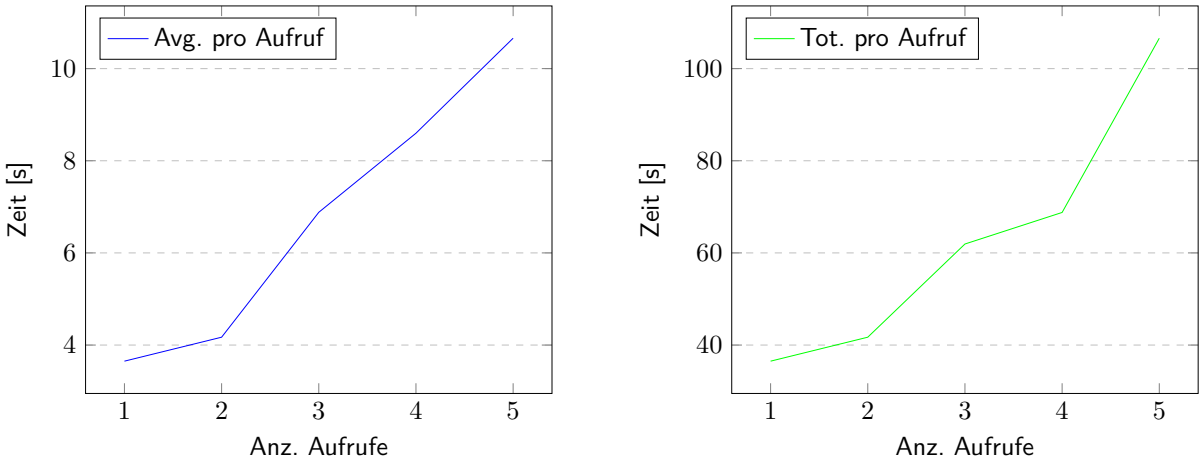
[4] Wartungs- und Initialisierungsfunktionen ausgeschlossen



(a) Durchschnittszeit (b) Gesamtzeit

Abbildung 2.5.: Performance sequenzieller `pgr_dijkstra` Aufrufe

Wir fixieren genau gleich wie bei OSRM in Abschnitt 2.1.3 die Anzahl Aufrufe, in diesem Fall aufgrund der bisher schlechten Performance auf nur 10 und führen diese parallel aus, um zu sehen wie sich pgRouting unter Last verhält. Genau gleich wie bei OSRM degradiert die Performance trotz gleichbleibender Aufrufe mit einem Fakor zwischen 3-4 bei nur 5 gleichzeitigen Clients.



(a) Durchschnittszeit (b) Gesamtzeit

Abbildung 2.6.: Performance paralleler `pgr_dijkstra` Aufrufe

Optimierungen

Laut Martin Stypinski, einem HSR Stundenten der sich beruflich mit pgRouting beschäftigt, ist der Flaschenhals die übergabe der Daten an die Boost C++ Library. Um dies zu optimieren soll nur ein Subset an `ways` (Edges) die für das Routing zur Verfügung stehen mit einer Vorselektion limitiert werden. Dies führt wie nachfolgend zu sehen ist, tatsächlich zu einer durchschnittlichen Zeit pro Aufruf von nur noch etwa 500 ms, was um Faktor 6 schneller als zuvor ist. Weitere Performanceoptimierungen in diesem Bereich sind mit höchster Wahrscheinlichkeit durch eine für Fussgänger optimierte `mapconfig.xml` beim osm2pgrouting zu erreichen, dabei würden dann alle für Fussgänger unerreichbaren Edges wie Autobahnen und weitere Strassen entfallen.

```
fscala@ubuntu:~$ ./pgrouting_perftest.py 100
Samples: 100
Threads: 1
(Thread, avgTime, totTime, samples, failures): 0, 0.477237, 25.770781, 100, 46
Total AVG time: 0.477237s
Total time: 25.770781s
Total failures: 46
```

Was jedoch in diesem vereinfachten Testfall ebenfalls auffällt, ist dass nun plötzlich sehr viele (46/100) Abfragen fehlschlagen. Dies hat vermutlich den Grund, dass pgRouting nicht mehr auf Strassen die ausserhalb der selektierten Bounding-Box liegen, zurückgreifen kann. In Programmcode 2.1 ist die optimierte Abfrage zu sehen. Der erste an `pgr_dijkstra` übergebene Parameter stellt dabei die Abfrage der Vorselektion dar. In diesem Beispiel werden nur diejenigen `ways` in betracht gezogen, welche innerhalb der Bounding-Box unserer zufällig gewählten Start- und Endpunkte liegt.

Programmcode 2.1: Optimierte `pgr_dijkstra` Abfrage

```
SELECT id, the_geom FROM
  pgr_dijkstra(
    'SELECT gid as id, source, target, reverse_cost as cost FROM ways WHERE the_geom &&
     ↪  ST_MakeEnvelope(8.402566, 47.316579, 8.8771673, 47.428183, 4326);',
    pgr_pointtoid(ST_setSRID(ST_MakePoint(8.75419706987, 47.3998922807), 4326), 0.1,
     ↪  'ways_vertices_pgr', 4326)::Integer,
    pgr_pointtoid(ST_setSRID(ST_MakePoint(8.68231039364, 47.4012570193), 4326), 0.1,
     ↪  'ways_vertices_pgr', 4326)::Integer,
    false,
    false
  ) as route
JOIN ways_vertices_pgr ON route.id1 = ways_vertices_pgr.id;
```

2.2.4. Bewertung

Kriterium	Bewertung	Bemerkung
Funktionalität	★★★★☆	Bietet die Mindestfunktionalität im Bezug auf Routing und erfüllt somit diese Anforderung.
Features	★★★★☆	Im Vergleich zu OSRM bietet pgRouting viel mehr Funktionalität. Den Abzug bekommt es für die fehlende VRP Funktionalität, welche sich für die Aufgabenstellung als nützlich hätte erweisen können.
Performance	☆☆☆☆☆	Mit einer Durchschnittlichen End-to-End Zeit von 10 s bei 5 simultanen Anfragen ist pgRouting definitiv zu langsam für eine Real-Time Applikation. Insbesondere wenn man bedenkt, dass abhängig vom Algorithmus bis zu $\frac{nPOI(nPOI-1)}{2}$ Abfragen nötig sein werden.
Wartbarkeit	★★★☆☆	Obschon pgRouting eine grössere Community hat und gegen aussen als sehr reifes Projekt aussieht, war der Import der Daten ein sehr fehleranfälliger Prozess. Auch Martin Stypinski als externer Experte berichtete von Migrationsproblemen zwischen Versionen sowie Berechtigungsproblemen[5] beim PostgreSQL Benutzer.
Benutzbarkeit (Entwickler)	★★☆☆☆	Es ist ohne grössere Einarbeitung möglich simplere Abfragen zu erstellen. Doch alles was darüber hinaus geht inklusive Performance-Optimierungen braucht tieferes Verständnis. Zudem ist ein "out of the box" Routing optimiert für Fussgänger nicht vorhanden und müsste erst erarbeitet werden.

Tabelle 2.9.: Bewertung von pgRouting

2.3. Fazit

Aufgrund der inakzeptablen Performance von pgRouting wurde auf eine direkte Gegenüberstellung der zwei Kandidaten verzichtet und wie folgt entschieden:

Entscheid 2.1: Routing Engine (OSRM)

Im Kontext der Evaluation einer für die Aufgabenstellung passenden Routing Engine wurde, um eine akzeptable Real-Time Performance zu erreichen für OSRM und gegen pgRouting entschieden und wir akzeptieren, dass OSRM weniger Flexibilität, Funktionalität und evt. Support liefert.

[5] Abfragen müssen als Superuser ausgeführt werden

3. Weitere Software

In diesem Kapitel wird kurz auf die Evaluation und Entscheide bezüglich anderer eingesetzter Software Komponenten und Frameworks eingegangen. Da es sich hierbei nicht mehr um die Machbarkeit der Aufgabenstellung dreht, wurde nicht so detailliert und mit so vielen Bewertungskriterien evaluiert wie bei der Routing Engine.

3.1. Backend (Server)

Da der Einsatz von Python als Programmiersprache für das Backend durch den Auftraggeber als Nichtfunktionale Anforderung definiert wurde, schränkt sich die Vorselektion der in Frage kommenden Frameworks auf die populärsten Python Webframeworks ein. Die Vorselektion wurde auf nachfolgend beschriebenen populären Python Frameworks festgelegt.
Es wurden diverse Reviews sowie die "Quickstart" Dokumentationen der jeweiligen Frameworks konsultiert. Wie in Tabelle 3.2 ersichtlich, ist auch hier wieder eine möglichst kurze Einarbeitungszeit, vor allem auch aufgrund der kurzen Dauer der Studienarbeit sehr wichtig.

Kriterium	Beschreibung	Gewichtung
Funktionalität	Wie gut eignet sich das Framework zur Implementation der Aufgabenstellung. Ist die Aufgabenstellung damit zu bewältigen und mit welchem (Mehr-)Aufwand gegenüber der anderen Frameworks.	★★★☆☆
Wartbarkeit	Wie gut ist die Dokumentation. Wie verbreitet ist das Framework, um wenn nötig, Personal mit entsprechendem Know-how anzustellen oder Hilfe auf Internetplattformen zu suchen. Wer steckt hinter dem Framework und wie gross sind die Chancen, dass dieses nicht mehr weitergeführt wird.	★★☆☆☆
Benutzbarkeit (Entwickler)	Wie lange dauert die Einarbeitung und wie schnell bzw. reibungslos lässt sich danach damit Entwickeln.	★★★★★

Tabelle 3.2.: Bewertungskriterien des Backend Frameworks

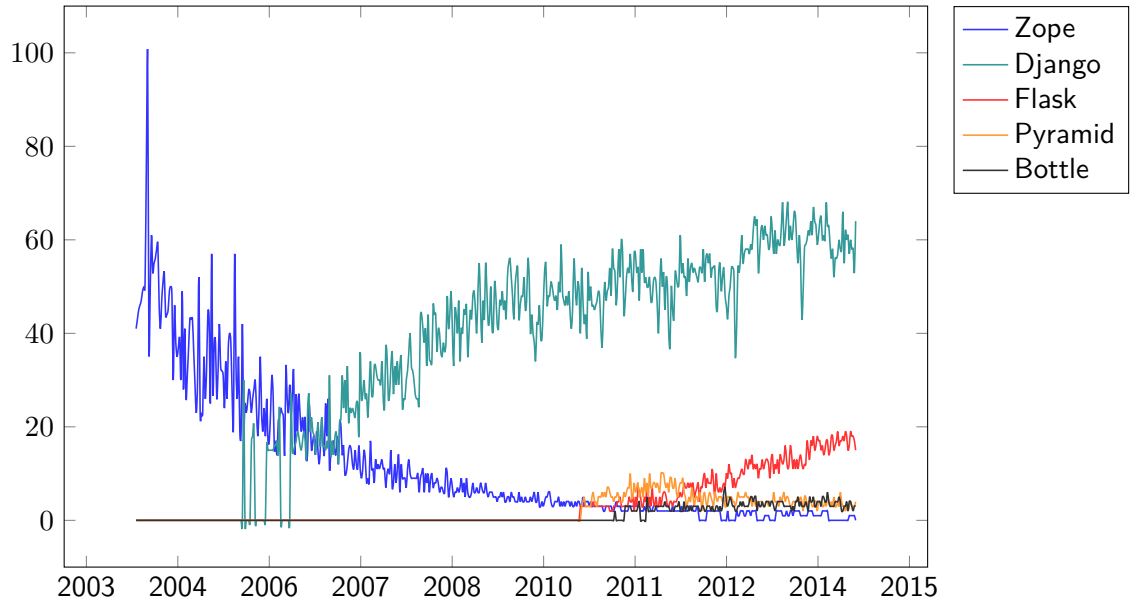


Abbildung 3.1.: Python Webframework Trends
(Datenquelle: *Google Trends* [7])

3.1.1. Zope

Zope ist das Urgestein der Python Webframeworks und wird hier nur noch aus Vollständigkeitsgründen genannt. Plone, ein heute noch bekanntes Python Content Management System (CMS) basiert auf Zope. Die komplexe Architektur und Verwendung von Zope sowie Templates die CMS-artig in der Zope Datenbank (ZODB) leben machen Zope zu einem altmodischen und unattraktiven Kandidaten für neue Software-Projekte. Die Trendanalyse in Abb. 3.1 zeigt ebenfalls den stetigen Rückgang des einst populärsten Python Webframework.

3.1.2. Django

Django ist ein Frameworks welches auf dem klassichen Model-View-Controller (MVC) Prinzip basiert und heutzutage das mit Abstand bekannteste Python Webframework. Django gehört mit folgender Funktionalität klar zu den "full-stack" Frameworks:

- Templating/Views
- Models / Eigenes Object-relational Mapping (ORM) Framework
- Create-Update-Delete (CRUD) Generator
- Caching
- Internationalisierung (I18N)
- User- und Session-Handling
- Security Features
  - Authentication & Authorization
  - SQL-Injection

  - Cross-Site-Scripting
  - Cross-Site-Request-Forgery
  - ...

Durch die Verbreitung gibt es eine Vielzahl an Plugins sowie eine grosse und stabile Community.

3.1.3. Pyramid

Pyramid ist der Nachfolger von Pylons und gehört ebenfalls zu den eher funktionsreicheren Frameworks, jedoch mit einem minimalistischen Ansatz. Es bietet weit weniger Funktionalität als Django und findet häufig Verwendung beim "Bau" eigener Frameworks wie z.B. CMS' (Kotti[1], Nive[2], ...).

- Scaffolding
- Templating/Views
- ORM via SQLAlchemy oder ZODB
- Session Handling
- I18N und Lokalisierung (L10N)
- Authentication & Authorization
- Testing

Pyramid wird aufgrund der komplexen Architektur und Verwendung von Teilen aus Zope nicht weiter verfolgt.

3.1.4. Flask

Flask ist eines der bekannteren Python Mikro-Webframeworks, wie unschwer in Abb. 3.1 zu erkennen ist.

- Templating via Jinja2
- Session Handling
- ORM via SQLAlchemy
- Modularität via Flask "Blueprints"

3.1.5. Bottle

Bottle ist ebenfalls ein Mikro-Webframework welches aus einem einzigen Modul besteht und im Gegensatz zu Flask keine weiteren Abhängigkeiten besitzt. Im Standalone[3] Minimalbeispiel in Programmcode 3.1, entnommen aus der Dokumentation, ist die Simplizität, die auch etwas an Flask erinnert zu erkennen.

[1] http://kotti.pylonsproject.org/
[2] http://cms.nive.co/
[3] Es wird nichts weiter benötigt, das Beispiel ist komplett lauffähig

Programmcode 3.1: Bottle Minimalbeispiel

```
from bottle import route, run, template

@route('/hello/:name')
def index(name='World'):
  return template('<b>Hello {{name}}</b>!', name=name)

run(host='localhost', port=8080)
```

Bottle wird aufgrund der grossen Überschneidung des Funktionsumfangs mit Flask und der kleinen Community, was sich durch weniger Plugins und Online Support äussert, nicht weiter verfolgt.

3.1.6. Fazit

Nach einer kurzen Evaluationsphase standen sich Flask als simples Mikroframework mit vielen Erweiterungsmöglichkeiten via Plugins und Django, welches von Haus aus fast alles selbst mitbringt, gegenüber. Da beide Frameworks sich für die simplen Anforderungen der Studienarbeit gut eignen und eine starke Community besitzen, wurde auf eine detaillierte Evaluation verzichtet und lediglich das Kriterium der Lernkurve bzw. Verwendung aus Entwicklersicht betrachtet:

Entscheid 3.1: Backend Framework (Flask)

Im Kontext der Evaluation eines für die Aufgabenstellung passenden Python Backend-Frameworks wurde, um die Anforderungen der Studienarbeit (DB-Zugriffe, Webservices, Basis-Funktionalität Webserver) zu erreichen für Flask und gegen Django und andere entschieden und wir akzeptieren, dass erweiterte Funktionalität (ORM, Caching, ...) nur über Plugins verfügbar ist.

3.2. Webfrontend (Client)

In diesem Abschnitt geht es um eine kurze Evaluation des JavaScript Frameworks, welches für die Webseite und evtl. einer künftigen mobile Webapplikation eingesetzt werden soll. Im heutigen HTML5 Zeitalter geprägt von SPAs ist der Einsatz von JavaScript kaum wegzudenken. In den letzten Jahren konnte sich vor allem jQuery als Bibliothek zur DOM Manipulation durchsetzen und gilt heute als de-facto Standard in diesem Bereich. Doch für alle anderen Zwecke wie Routing, Architektur, Templating etc. gibt es diverse Optionen. Aufgrund der hohen Anzahl "ein-Mann"[4] Frameworks und Libraries in der JavaScript Welt, ist der Einsatz solcher mit einem gewissen Risiko verbunden und sollten wenn möglich vermieden werden.

[4] Es wird hauptsächlich von einem einzigen Entwickler gewartet

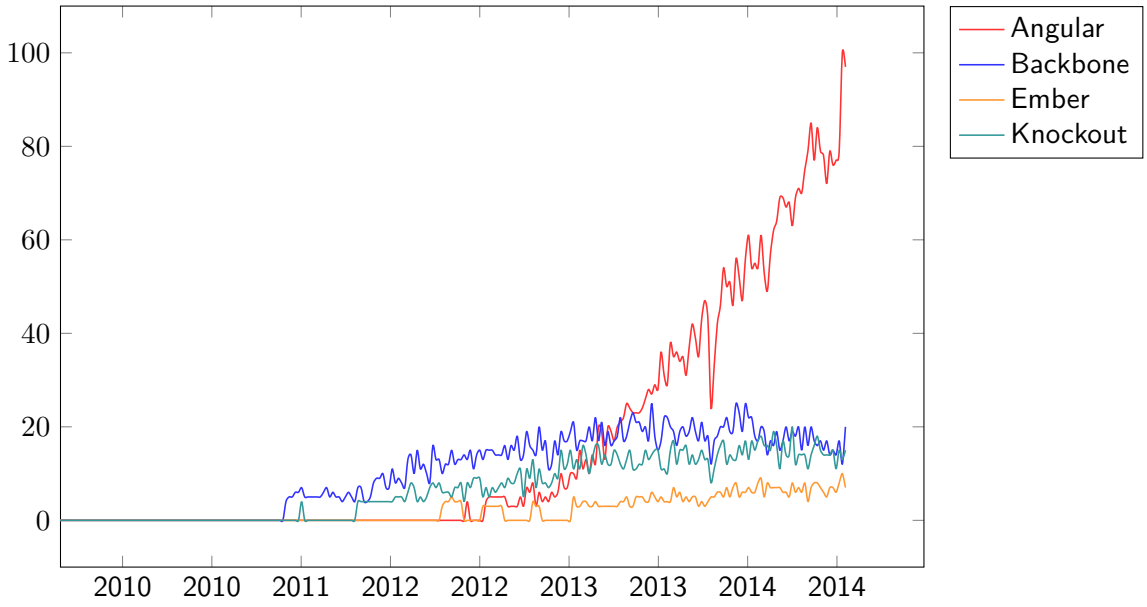


Abbildung 3.2.: JavaScript Framework Trends
(Datenquelle: *Google Trends* [7])

Die Trendanalyse in Abb. 3.2 zeigt die heute am häufigsten verwendeten JavaScript MVC Frameworks [11]. Es ist klar ersichtlich, dass AngularJS spätestens Ende 2013 die Konkurrenz endgültig hinter sich gelassen hat, was durch die Kennzahlen in Tabelle 3.4 bestätigt wird. Durch die Modularität von AngularJS und der möglichkeit, eigene Module in Form von Services oder Direktiven[5] zur Verfügung zu Stellen ist die Anzahl Plugins gegenüber den anderen Frameworks nahezu explodiert.

Framework	Contributors	Commits	Releases	Branches	Plugins
Angular	1045	5990	111	12	2582
Backbone	236	2667	21	2	510
Ember	412	7496	90	35	245
Knockout	43	1241	33	18	84

Tabelle 3.4.: Kennzahlen JavaScript Frameworks

Aus diesen Gründen wurde folgende Entscheidung getroffen:

Entscheid 3.2: Frontend Framework (AngularJS)

Im Kontext der Evaluation eines für die Aufgabenstellung passenden Frontend JavaScript Frameworks wurde, um eine rapide Entwicklung mit geringen Support-Risiken zu erreichen für AngularJS und gegen Ember, Knockout, Backbone und andere entschieden und wir akzeptieren, dass der Einsatz mit gewissen Einarbeitungsaufwänden verbunden ist.

Als zweite Hauptkomponente im Frontend wird eine JavaScript Bibliothek zur Darstellung von Karten benötigt. Auf die Evaluation zwischen Leaflet und dessen Konkurrent OpenLayers wurde verzichtet:

[5] Erweiterung der HTML Syntax, so gibt es zum Beispiel eine <leaflet></leaflet> Direktive

Entscheid 3.3: Bibliothek zur Darstellung von Karten (Leaflet)

Im Kontext der Auswahl einer JavaScript Bibliothek zur Darstellung von Karten wurde, um solche Karten mit möglichst wenig Aufwand darzustellen für Leaflet und gegen OpenLayers entschieden und wir akzeptieren, dass ausser Feedback durch Dritte keine Evaluation stattgefunden hat und der Einsatz somit mit Risiken verbunden ist.

Teil III.

Projektdokumentation

1. Vision

Die Vision von POI Tour und deren Herkunft ist in Teil I, Abschnitt 1.1 auf Seite 2 beschrieben.

2. Anforderungen

In diesem Kapitel sind die Anforderungen an die Applikation in Form von Scrum Epics und User Stories sowie weitere, nicht-funktionale Anforderungen dokumentiert.

2.1. User Stories

Die User Stories wurden aufgrund der zunächst unbekannten Anforderungen iterativ aufgenommen. In Abb. 2.1 sind die drei Epics als Übersicht in Form eines Use-Case Diagramms dargestellt.

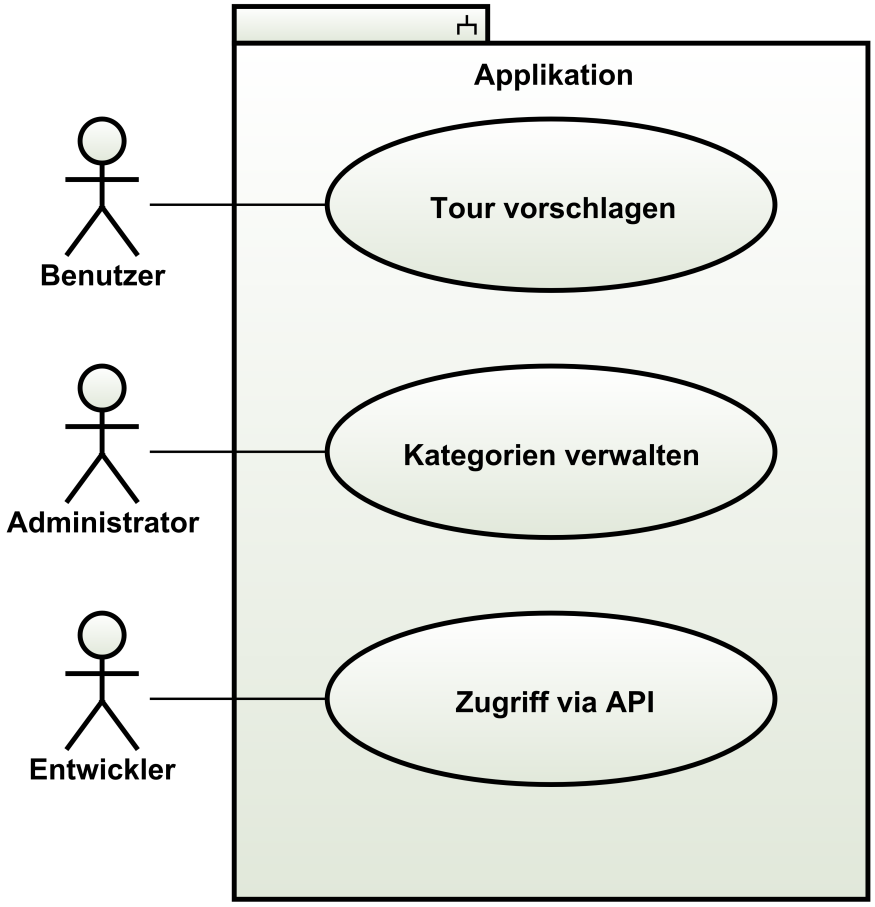


Abbildung 2.1.: Übersicht Epics

2.1.1. Rollen

In Tabelle 2.2 sind die für das Projekt relevanten Rollen bzw. Akteure aufgelistet. Diese werden in den nachfolgenden Abschnitten referenziert.

Rolle	Beschreibung
Benutzer	Ist diejenige Person welche an der Verwendung der Webapplikation für eigene Zwecke interessiert ist. Kann beispielsweise ein Tourist sein der die Sehenswürdigkeiten Luzerns entdecken will, oder ein Einwohner der spontan einen Spaziergang vor hat.
Entwickler	Die Entwickler der Applikation. Während der Laufzeit der Studienarbeit ist dies Fabio Scala.
Drittentwickler	Andere Entwickler welche die Funktionalität der Applikation via API nutzen wollen.
Kort Benutzer	Sind diejenigen, die das Kort Game verwenden und an der Vervollständigung von OSM Daten interessiert sind.
Administrator	Der Applikationsverantwortliche, welche die Applikation betreibt und wartet.

Tabelle 2.2.: Rollen/Akteure

2.1.2. Webservice

Die konkreten Anforderungen Seitens des Product Owners (Betreuer) beschränken sich auf der in Abschnitt 1.1 beschriebenen Vision für das Kort Game. Aus dieser Vision ergibt sich folgendes Epic 1.

Epic 1: Webservice / API

Als Drittentwickler, will ich die Funktionalität der Software über eine REST API ansprechen können, sodass ich diese in meiner eigenen Software (z.B. Kort) nutzen kann.

Die Schnittstelle basierend auf dem HTTP Protokoll (Webservice) soll sich mit den Anforderungen von Kort verwenden lassen. Der Webservice soll anhand von

- Startpunkt
- Endpunkt
- Zwischenstopps
- Reisezeit
- Zeit pro Zwischenstopp

eine Tour in einem generischen und/oder standardisierten Format zurückgeben, sodass diese von der Kort Applikation weiterverarbeitet werden kann.

2.1.3. Fussgänger-Tour erstellen

Epic 2: Fussgänger-Tour erstellen

Als ein Benutzer, will ich eine Fussgänger-Tour erstellen lassen können, sodass ich in gegebener Zeit möglichst viele für mich interessante POIs besuchen kann.

Akzeptanzkriterien
Gegeben
- ein nach dem Jahre 2011 verkaufter handelsüblicher PC,
- einen modernen Webbrowser (Internet Explorer 10 oder neuer und alle Evergreen[a] Browser),
- einer Leistungsstarken Server-Infrastruktur (solange das Hinzufügen von Prozessoren oder Arbeitsspeicher zu einer Verbesserung führt),

wenn ich eine Tour mit einer maximalen Dauer von 6 h und höchstens 400 in Frage kommende (umliegende) POIs erstellen will, dann soll die Applikation nach maximal 15 s eine Tour auf meinem Bildschirm anzeigen.

[a] Browser die sich automatisch auf dem neusten Stand halten

Durch ein kurzes Interview[1] sowie Diskussionen mit Drittpersonen[2], wurden die Anforderungen in Absprache mit dem Betreuer wie folgt präzisiert:

User Story 2.1: Start- und Ende bestimmen

Als ein Benutzer, will ich sowohl den Start- wie auch den Endpunkt (Ziel) meiner Tour bestimmen, sodass ich mich am Ende meines Spaziergangs am gewünschten Ort befinde.

Akzeptanzkriterien
Gegeben
- den Startpunkt A,
- den Endpunkt B,

wenn ich die Tour berechnen lasse, dann soll diese bei einem speziell markierten Punkt A starten und über ausgewählte POIs zu Punkt B führen.

User Story 2.2: Sofortige Validierung der Eingaben

Als ein Benutzer, will ich meine Eingaben sofort validieren lassen, sodass ich diese gleich korrigieren kann und nicht erneut über alle falschen Eingaben gehen muss.

Akzeptanzkriterien
Gegeben ein Eingabefeld mit bestimmten Validierungskriterien, wenn ich die Eingabe für das bestimmte Feld beendet habe, dann soll sofort ersichtlich sein, ob diese gültig ist.

[1] Siehe Anhang B auf Seite 104 mit einer Tourismus-Expertin von ZüriOberland Tourismus
[2] Mitstudenten, Freunde und Bekannte, ...

User Story 2.3: Anzeige von Start und Ende

Als ein Benutzer, will ich nach der Eingabe von Start oder Ende diese Punkte sofort auf der Karte sehen können, sodass ich sicher sein kann, dass die korrekten Orte verwendet werden.

Akzeptanzkriterien
Gegeben ein Eingabefeld für Start oder Ende, wenn ich die Eingabe für das bestimmte Feld beendet habe, dann soll der für die Eingabe gefundene Ort auf der Karte dargestellt werden.

User Story 2.4: Aufenthaltsdauer bestimmen

Als ein Benutzer, will ich die Aufenthaltsdauer pro POI bestimmen können, sodass ich mir Zeit nehmen kann um diese zu besichtigen.

Akzeptanzkriterien
Gegeben ein Eingabefeld für die Aufenthaltsdauer, wenn ich die Aufenthaltsdauer verändere und die Tour berechnen lasse, dann soll die Tour abhängig von der Aufenthaltsdauer länger oder kürzer werden.

User Story 2.5: Interessen bestimmen

Als ein Benutzer, will ich die für mich interessanten POIs auswählen können, sodass ich eine für mich personalisierte Tour erhalte.

Akzeptanzkriterien
Gegeben eine Eingabemöglichkeit für Interessenskategorien / POI-Kategorien wenn ich Interessen hinzufüge oder entferne, dann soll die Tour nur durch jene Interessenspunkte gehen.

User Story 2.6: Nahegelegene Sehenswürdigkeiten integrieren

Als ein Benutzer, will ich die Möglichkeit haben weitere Sehenswürdigkeiten welche die Tour nicht beeinträchtigen (auf dem Weg liegen) in die Tour einfliessen zu lassen, sodass ich optional auch weitere, mir nicht zwingend interessante POIs anschauen kann.

User Story 2.7: Einzelne Viapunkte festlegen

Als ein Benutzer, will ich der Tour auch eigene fixe Punkte hinzufügen können, sodass ich nicht durch die Eingabe via vorhandenen Interessenskategorien eingeschränkt werde.

User Story 2.8: Anzeige in Frage kommender POIs

Als ein Benutzer, will ich alle für meine Eingaben in Frage kommenden POIs sehen, sodass ich die Kriterien noch vor der Berechnung der Tour anpassen kann.

Akzeptanzkriterien
Gegeben die Webseite mit einer Eingabemaske und einer Karte, wenn
- ich eine Kategorie hinzufüge/entferne,
- ich die Dauer oder Aufenthaltsdauer ändere,
- ich den Startpunkt ändere,

dann sollen auf der Karte alle in Frage kommenden POIs angezeigt werden.

User Story 2.9: Anzeige von Bildern

Als ein Benutzer, will ich Bilder der in der Tour enthaltenen POIs sehen, sodass ich Entscheiden kann, ob ich diesen POIs besichtigen möchte.

Akzeptanzkriterien
Gegeben die vorgeschlagene und angezeigte Tour mit den POIs, wenn
- ich mit der Maus über ein POI fahre,

dann sollen mir Bilder des POI gezeigt werden.

2.1.4. Kategorien verwalten

Epic 3: Verwaltung der POIs

Als ein Administrator, will ich die Liste der von den Benutzern auswählbaren POIs verwalten/erweitern können, sodass den Benutzern und der Applikation laufend mehr POIs zur Auswahl stehen.

User Story 3.1: Verwaltung durch OSM Tags

Als ein Administrator, will ich Kategorien durch Angabe von ein oder mehreren (konjunkt oder disjunkt vereinigt) OSM Tags konfigurativ verwalten können, sodass ich OSM POIs selbst und ohne Entwicklungsaufwand hinzufügen kann.

Akzeptanzkriterien
Gegeben eine für Menschen gut lesbare Konfigurationsdatei, wenn
- ich Kategorien hinzufüge/entferne,
- die Applikation neu starte,

dann sollen die aktuellen Kategorien dem Benutzer angezeigt werden.

User Story 3.2: Programmatische Erweiterung durch Plugins

Als ein Entwickler, will ich Kategorien programmatisch hinzufügen können, sodass ich zusätzliche Kategorien aus anderen Quellen (z.B. Kort API) hinzufügen kann.

Akzeptanzkriterien
Gegeben den Applikationscode und einem speziellen Verzeichnis für Plugins, wenn
- ich ein Plugin mit vordefinierten Schnittstellen programmiere,
- ich die Applikation neu starte,

dann soll die Applikation zusätzliche Kategorien aus dem Plugin beziehen und anzeigen.

2.2. Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen der einzelnen User Stories sind in deren Akzeptanzkriterien enthalten. In diesem Abschnitt werden zusätzliche, aus der Aufgabenstellung entnommene nicht-funktionale Anforderungen, ergänzt durch sinnvolle Anforderungen moderner Software-Entwicklung beschrieben.

2.2.1. Technologien

Die Anforderungen an die eingesetzten Technologien[3] ist teilweise durch die Aufgabenstellung sowie durch die bereits existierende GIS Umgebung an der HSR wie folgt eingeschränkt:

- OSM als Kartenservice und Datenquelle
- EOSMDBOne als Schnittstelle zu den OSM Daten
- Python als Programmiersprache
- Deployment der Applikation muss via WSGI möglich sein
- PostgreSQL als Datenbank

Die Wahl der Frameworks, sowohl Server- wie auch Clientseitig ist dem Entwickler überlassen. Der Betreuer dieser Studienarbeit empfiehlt jedoch die Verwendung von Flask.

2.2.2. Qualität

Zur Sicherung der Softwarequalität wurden folgende drei Anforderungen durch den Betreuer bestimmt:

- Einsatz von automatisierten Unit Tests
- Kommentieren des Codes in einem Sphinx[4] kompatiblen Format
- PEP-8 als Codierrichtlinie, besonders die Verwendung von Leerzeichen zur Einrückung

[3] Programmiersprachen, Frameworks, Libraries, Services, ...
[4] Siehe auch: http://sphinx-doc.org

3. Analyse

3.1. Algorithmus

Das grundsätzliche Problem, eine Tour mit *möglichst vielen* POIs innert gegebener Zeit zu besuchen und zum Startpunkt (oder wahlweise zu einem anderen) als Endpunkt zurückzukehren weist gewisse Parallelen zum Travelling Salesman Problem (TSP) auf. Dieses ist NP-hart und somit mit dem heutigen Wissensstand nicht innerhalb nützlicher Frist lösbar. Das TSP Problem wird häufig mit Simulated Annealing angegangen um eine näherungsweise gute Lösung zu finden.
Weitere Probleme, welche der Aufgabestellung teilweise näher kommen als TSP sind unter den folgenden Namen bekannt:

- Selective Travelling Salesman Problem (STSP)
- Orienteering Problem (OP)
- TSP with Profits (TSPwP)
- Deadline-TSP (DTSP)
- Vehicle Routing Problem (VRP)

Der schlussendlich verwendete Algorithmus löst das obengenannte Selective Travelling Salesman Problem (STSP) und ist in Abschnitt 3.1.2 beschrieben.

3.1.1. Simulated Annealing

Simulated Annealing ist ein probabilistisches Verfahren und basiert auf der Idee der Langsamen Abkühlung, welche beim Härten von Metallen Einsatz findet. Dabei ist es zwingend nötig eine Energiefunktion, welches die "Fitness" der Lösung angibt zu definieren. Zudem ist es wichtig zu jeder Lösung eine "Nachbarlösung" generieren zu können. Eine Nachbarlösung ist so definiert, dass sie sich möglichst wenig von der vorherigen unterscheidet. So ist es möglich durch erzeugen von Nachbarlösungen in lokale Minimas zu gelangen. Um nicht in lokale, meist nicht optimalen Minimas stecken zu bleiben, wird mit einer bestimmten Wahrscheinlichkeit die bessere Lösung verworfen und weitergesucht. Diese Wahrscheinlichkeit hängt üblicherweise von der aktuellen Temperatur ab und sinkt mit zunehmend besserer Fitness der Lösungen.
Beim Standard-TSP werden jeweils alle Punkte besucht. Ein Erstellen einer Nachbarlösung ist somit sehr einfach, man tauscht lediglich zwei aufeinanderfolgende Punkte der Tour miteinander. In unserem Fall stellt sich die Frage wie ein solcher Nachbar zustande kommt. Das Problem liegt in der Variabilität von sowohl Anzahl Punkte wie auch deren Reihenfolge. Um eine gute Lösung zu finden müssten beide Parameter bei einem Nachbarn modifiziert werden, da jedoch das Hinzufügen oder Löschen von Punkten zu einer komplett anderen Fitness führen, wurde Simulated Annealing als Lösungsverfahren nicht weiter analysiert.

3.1.2. Genetischer Algorithmus

Genetische Algorithmen, oftmals Evloutionäre Algorithmen genannt sind heuristische Optimierungsverfahren welche an die Evolution von Lebewesen angelehnt sind. Genetische Algorithmen weisen immer eine ähnliche Struktur auf, diese in ist Abb. 3.1 aufgezeigt.

Die Arbeit von Piwońska [10] beschreibt einen genetischen Algorithmus für das Selective Travelling Salesman Problem (STSP). Die einzelnen Phasen des Algorithmus werden in den folgenden Abschnitten beschrieben.

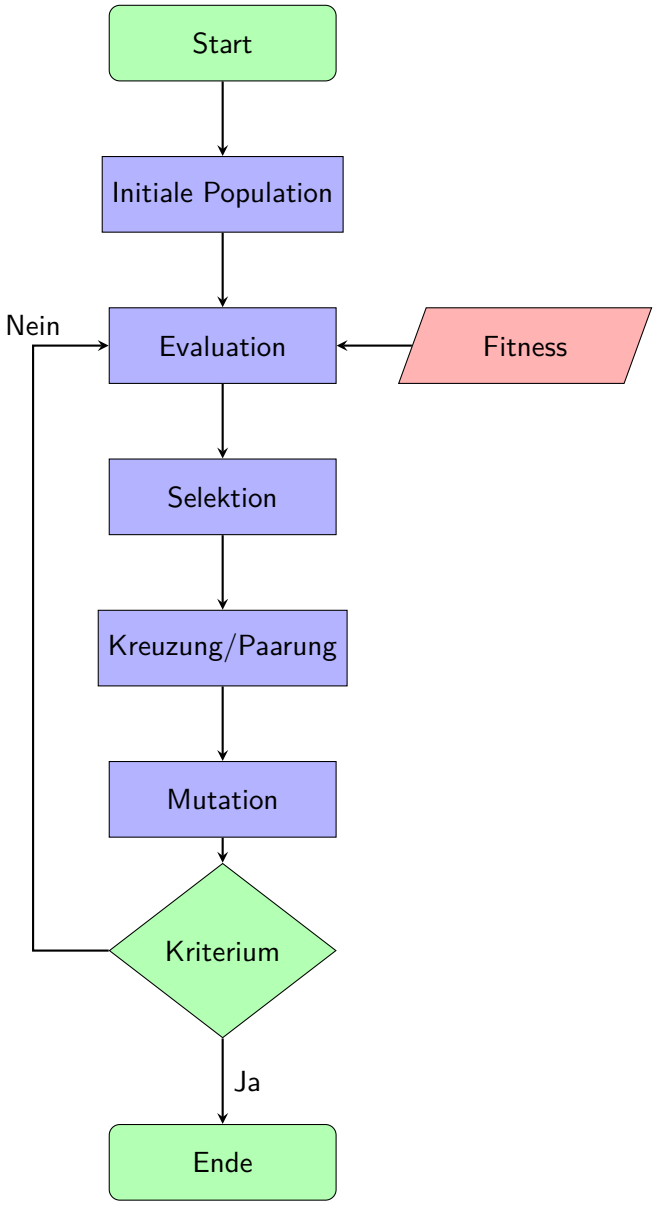


Abbildung 3.1.: Allgemeiner Ablauf genetischer Algorithmen

Begriff	Beschreibung
Gen	Ein Gen ist ein Teil des Individuums. Im Falle einer Tour entspricht ein einzelner Punkt[1] einem Gen.
Individuum	Ein Individuum besteht aus einer Reihe von Genen und entspricht genau einer Lösung. Eine Tour in Form einer Liste von sortierten Punkten ist ein Individuum und besitzt eine bestimmte Fitness.
Fitness	Die Fitness der Individuen sagt aus wie "gut" bzw. "fit" dieser ist. Die Fitness lässt sich berechnen und dient zum Vergleich mit anderen Individuen.
Population	Die Population ist die Gesamtmenge an zurzeit vorhandenen Individuen. Die Anzahl Individuen wird anfangs gesetzt und ändert während der Laufzeit des Algorithmus nicht.
Evaluation	Bei der Evaluation wird die Fitness des einzelnen Individuen berechnet.
Selektion	Die Fittesten überleben und kommen in die neue Population (die Nachkommen) indem sie die das vorherige Individuum ersetzen.
Kreuzung	Zwei Individuen werden gepaart um ein neues Individuum zu erzeugen. Sind die entstandenen fitter als die Eltern, so ersetzen sie diese.
Mutation	Bei der Mutation wird üblicherweise das Individuum mit einer gewissen Wahrscheinlichkeit auf eine wohldefinierte Weise manipuliert. Dies kann beispielsweise durch hinzufügen oder löschen von Genen geschehen.

Tabelle 3.2.: Verwendete Terminologie beim Genetischen Algorithmus

Initiale Population

Zu Beginn wird eine Menge von N Touren mit einer möglichst guten Fitness generiert. Bei der Variante STSP, das heisst bei gleichem Start- und Endpunkt werden die Touren wie in Pseudocode 3.1 beschrieben erzeugt. Kurz gefasst wird jeweils ein zufälliger Punkt zur Tour hinzugefügt bis maximal die Hälfte der Zeit erreicht wurde, danach wird derselbe Weg zurück zum Start verwendet. Die Symmetrien aufgrund der gespiegelten Tour und die Duplikate dieser initialen Tour werden durch die darauf folgenden Schritte sehr schnell eliminiert [10].

[1] Die Zwischenstopps (POIs) wie auch Start- und Endpunkt

Pseudocode 3.1: Initiales Individuum

$Hier \leftarrow Startpunkt$
$Individuum \leftarrow [Hier]$ ▷ Liste von Punkten bildet die Tour
$TotKosten \leftarrow 0$ ▷ Dauer der Tour
$i \leftarrow 1$
while 1 **do**
  $Kandidat \leftarrow$ Zufälliger Punkt ausser $Hier$ oder $Startpunkt$
  $Kosten \leftarrow$ Kosten von $Hier$ nach $Kandidat$
  if $Kosten \leq \frac{Maximalkosten}{2}$ **then**
    $Individuum[i] \leftarrow Kandidat$
    $TotKosten \leftarrow TotKosten + Kosten$
    $i \leftarrow i + 1$
  else
    break
  end if
end while
$Individuum \leftarrow Individuum + \text{REVERSED}(Individuum)$[1,-1] ▷ Gespiegelt zurück zum Start

Die Arbeit von Piwońska beschreibt lediglich das Szenario wo Start- und Endpunkt äquivalent sind. Für den Fall, dass diese sich Unterscheiden wurde eine eigene Methodik zur Generierung der initialen Population entwickelt und unterscheidet sich im wesentlichen darin, dass die Tour so lange mit zufälligen Punkten erweitert wird, bis es nicht mehr möglich ist innert der gegebenen Zeit zum Ende zurückzukehren.

Selektion

Bei der Selektion wird über alle Individuen iteriert und mit dem fittesten aus zufälligen T[2] Individuen der Population ersetzt. Je grösser T gewählt wird, desto schneller konvergiert der Algorithmus, hat jedoch mehr Mühe aus lokalen Minimas zu kommen.

Pseudocode 3.2: Selektion

$Population \leftarrow$ Liste von Individuen
for all $Individuum \in Population$ **do**
  $Kandidaten \leftarrow$ Zufällige T aus $Population$
  $Kandidat \leftarrow$ Fittester aus $Kandidaten$
  Ersetze $Invididuum$ in der $Population$ mit $Kandidat$
end for

Kreuzung

Bei der Kreuzung wird pärchenweise über die Population iteriert. Es wird geschaut ob die Individuen gemeinsame Gene besitzen und ein zufälliges dieser ausgewählt. An diesem Kreuzungspunkt werden die hinteren Teile der Individuen vertauscht. Erfüllen die neuen Individuen die Kriterien einer gültigen Lösung, so ersetzen sie die Eltern in der Population.

[2] Die sog. "Turniergrösse"

Pseudocode 3.3: Kreuzung

for all Zufällige 2 Individuen $(Individuum1, Individuum2) \in Population$ **do**
  $GemeinsameGene \leftarrow$ Alle Punkte die $Individuum1, Individuum2$ gemeinsam haben
  $Kreuzungsgen \leftarrow$ Zufälliger Punkt aus $GemeinsameGene$
  $Kind1 \leftarrow Individuum1$ bis $Kreuzungsgen + Individuum2$ ab $Kreuzungsgen$
  $Kind2 \leftarrow Individuum2$ bis $Kreuzungsgen + Individuum1$ ab $Kreuzungsgen$
  if Kosten $Kind1 \leq Maximalkosten$ **then**
    Ersetze $Individuum1$ durch $Kind1$ in der $Population$
  end if
  if Kosten $Kind2 \leq Maximalkosten$ **then**
    Ersetze $Individuum2$ durch $Kind2$ in der $Population$
  end if
end for

Mutation

Bei der Mutation werden zunächst doppelte Gene entfernt. Danach wird ein zufälliges Gen gelöscht. Zum Schluss werden so lange Gene (und somit Punkte der Tour) hinzugefügt wie es die Maximalkosten nicht überschreitet hat. Dies erfolgt dadurch, dass die in Frage kommenden Gene in Fitness-maximierender[3] Reihenfolge hinzugefügt werden.

Pseudocode 3.4: Mutation

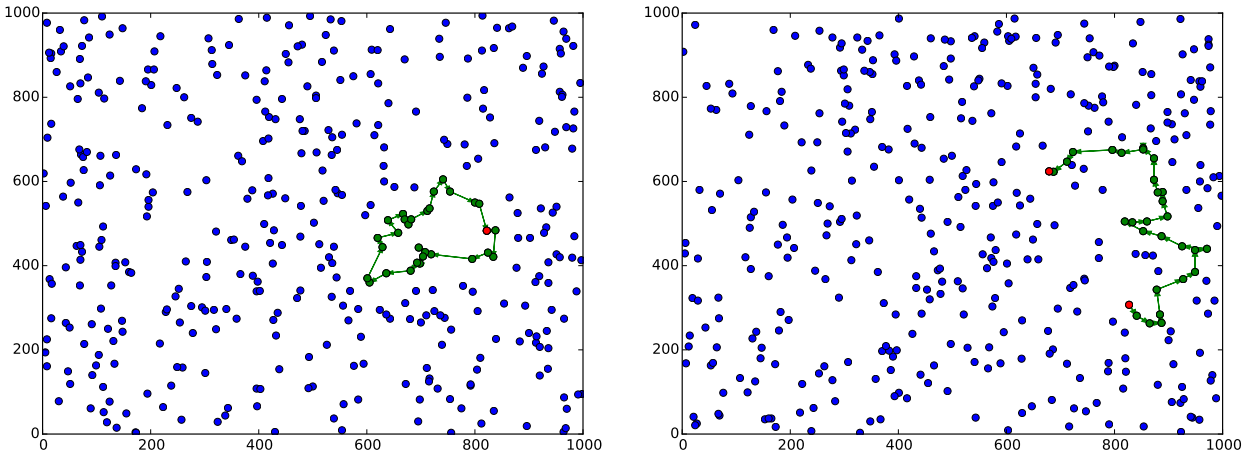
for all $Individuum \in Population$ **do**
  $Individuum \leftarrow Individuum$ ohne doppelte Gene
  $Individuum \leftarrow Individuum$ mit einem zufällig gelöschten Gen (ausser $Startpunkt$ oder $Endpunkt$)
  $Gen \leftarrow$ Zufälliges Gen aus $Individuum$
  $Punkte \leftarrow$ Alle Punkte profitmaximierend sortiert
  for all $Punkt \in Punkte$ **do**
    $Kandidat \leftarrow$ Kopie von $Individuum$
    Füge $Punkt$ nach Gen in $Kandidat$ ein
    if Kosten $Kandidat \leq Maxinmalkosten$ **then**
      Ersetze $Individuum$ in $Population$ mit $Kandidat$
    else
      break
    end if
  end for
end for

Prototyp

Um die Machbarkeit der Aufgabenstellung mit dem genetischen Algorithmus von Piwońska zu prüfen, wurde zunächst ein "loser" Prototyp mit kartesischen Koordinaten implementiert und in Hinsicht Performance und

[3] Mit dem geringsten Abstand vom zufälligen Punkt, sodass möglichst viele hinzugefügt werden können

Fitness der Lösungen evaluiert. Der Algorithmus verwendet NumPy[4], die de-facto Standardbibliothek für mathematische Berechnungen mit Python. Die Ergebnisse des Prototyps sind in Abb. 3.2 ersichtlich.



(a) Von A nach A (b) Von A nach B

Abbildung 3.2.: Prototyp des Genetischen Algorithmus

Die Ergebnisse des Prototyps waren sehr zufriedenstellen, weshalb für dieses entschieden und darauf aufgebaut wurde:

Entscheid 3.1: Algorithmus (Genetischer Algorithmus für STSP)

Im Kontext des NP-harten Routingproblems wurde aus Zeitgründen der Genetische Algorithmus aus der Arbeit von Piwońska verwendet. Simulated Annealing sowie weitere Ansätze wurden nicht weiter verfolgt. Wir akzeptieren, dass es eventuell andere, besser geeignete Algorithmen zur Lösung des Problems gäbe.

3.2. Routing Engine

Aufgrund der Notwendigkeit einer Distanzmatrix für alle Punkte, ist die Performance der darunterliegenden Routing Engine entscheidend. Bei nur 100 in Frage kommenden Punkten liegt die Anzahl Anfragen bereits bei $\frac{n(n-1)}{2} = 4950$ einzelnen Distanzabfragen. Die detaillierte Evaluation der Routing Engine sowie der Entscheid (OSRM) ist in Teil II, Kapitel 2 auf Seite 14 beschrieben.

3.3. Webservice Schnittstelle

Aufgrund der Verwendung von AngularJS und somit auch XHR Aufrufen für den Datenaustausch erübrigt sich ein spezieller Webservice welches die Benutzerfunktionalität abbildet. Ein spezieller Webservice gemäss Epic 1, welches keine Kategorien zur Findung von POIs sondern einzelne Punkte übergibt muss dennoch implementiert werden.

[4] Siehe auch: http://www.numpy.org/

3.3.1. Parameter

Ausser den Kategorien wird von denselben Parametern wie im User Interface ausgegangen, diese wären:

- Startpunkt
- Endpunkt (optional)
- Liste von Punkten
- Dauer in Sekunden
- Aufenthaltsdauer pro POI in Sekunden

Die Koordinaten von Start- und Endpunkt sowie der übrigen Punkte sind in SRID 4326 zu übergeben.

3.3.2. Resultat

Als Format für die geographischen Strukturen[5] in der JSON Antwort bietet sich GeoJSON[6] an. Ein offener Standard welches bereits in diversen existierenden Applikationen wie beispielsweise Nominatim[7] Verwendung findet.

3.4. User Interface

Beim Human Computer Interaction and Design (HCID) wurde auf ein möglichst schlankes und simples User Interface gesetzt. Ein erstes Mockup, was in Abb. 3.3 zu sehen ist enthielt noch eine Navigationsleiste sowie Schieberegler zur Gewichtung der POIs. Diese Elemente wurden nach einer knappen Befragung[8] einer Tourismus-Expertin sowie Rücksprache mit dem Betreuer entfernt, woraus ein noch schlichteres Design wie in Abb. 3.3 ersichtlich, entstanden ist.
Da zu einem späteren Zeitpunkt die Anforderung der Gewichtung erneut aufgekommen ist, enthält die schlussendliche Lösung in Teil I, Abb. 3.1 auf Seite 9 erneut ein Element zur Gewichtung.

[5] Punkte, Pfade, ...
[6] Siehe auch: http://geojson.org/geojson-spec.html
[7] Ein Geocoding Service für OSM
[8] Anhang B auf Seite 104

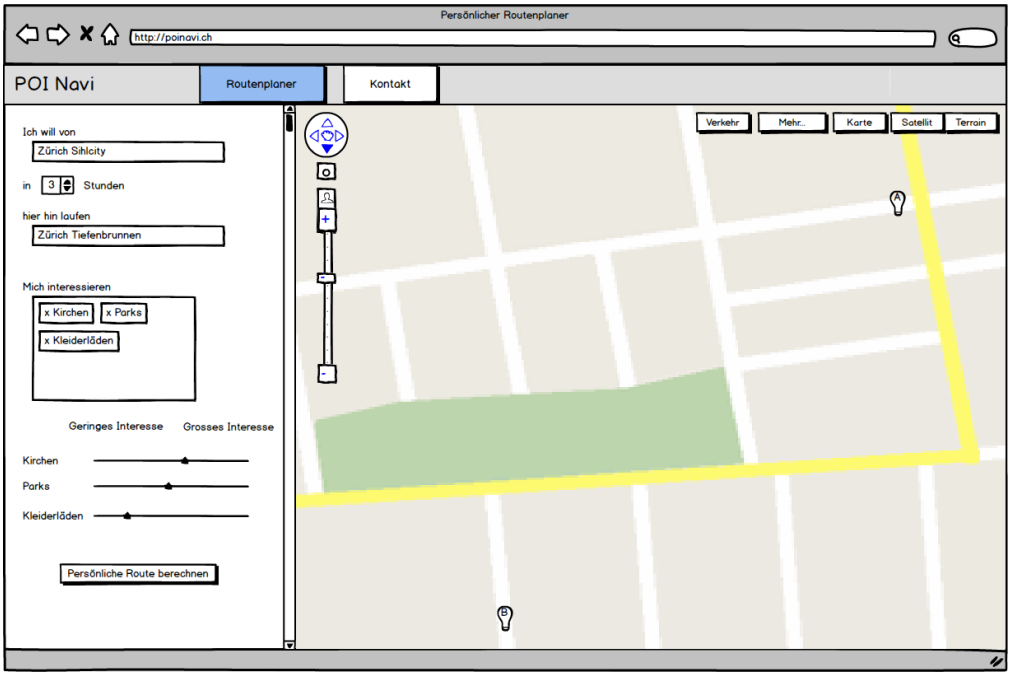


Abbildung 3.3.: Erste Skizze für Interview mit Tourismus-Expertin

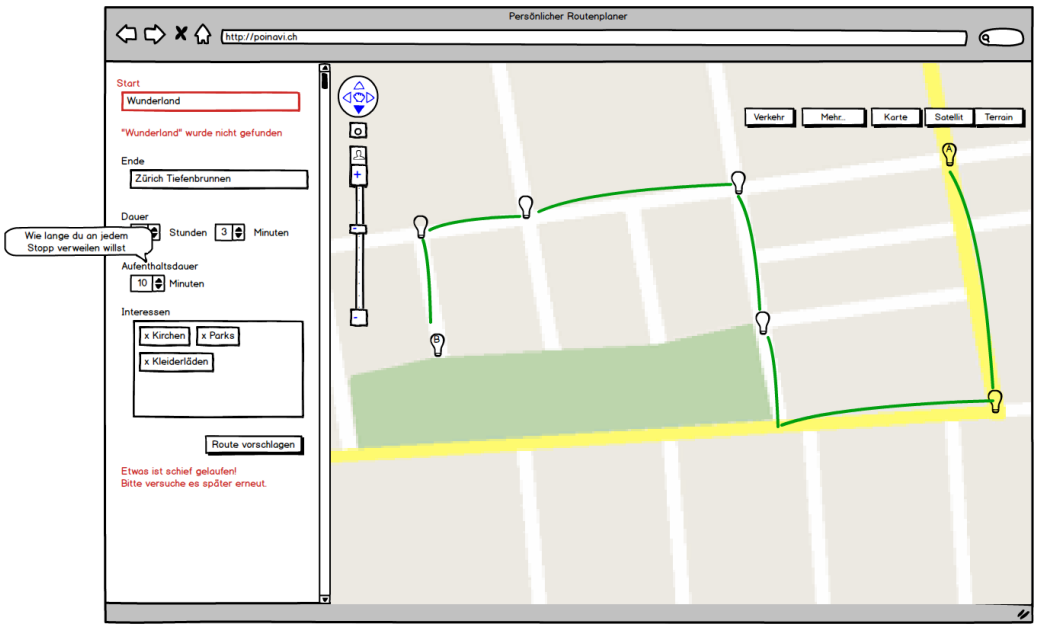


Abbildung 3.4.: Zweite Skizze nach Interview und Besprechung mit Product Owner

4. Design und Implementation

4.1. Architektur

In den folgenden Abbildungen 4.1 bis 4.3 wird die Grundstruktur von POI Tour aufgezeigt.

4.1.1. Komponenten

Abb. 4.1 zeigt eine Gesamtübersicht der Komponenten und verwendeten externen APIs. Die Pfeile sind als Aufrufe oder umgekehrte Datenflüsse[1] zu deuten.

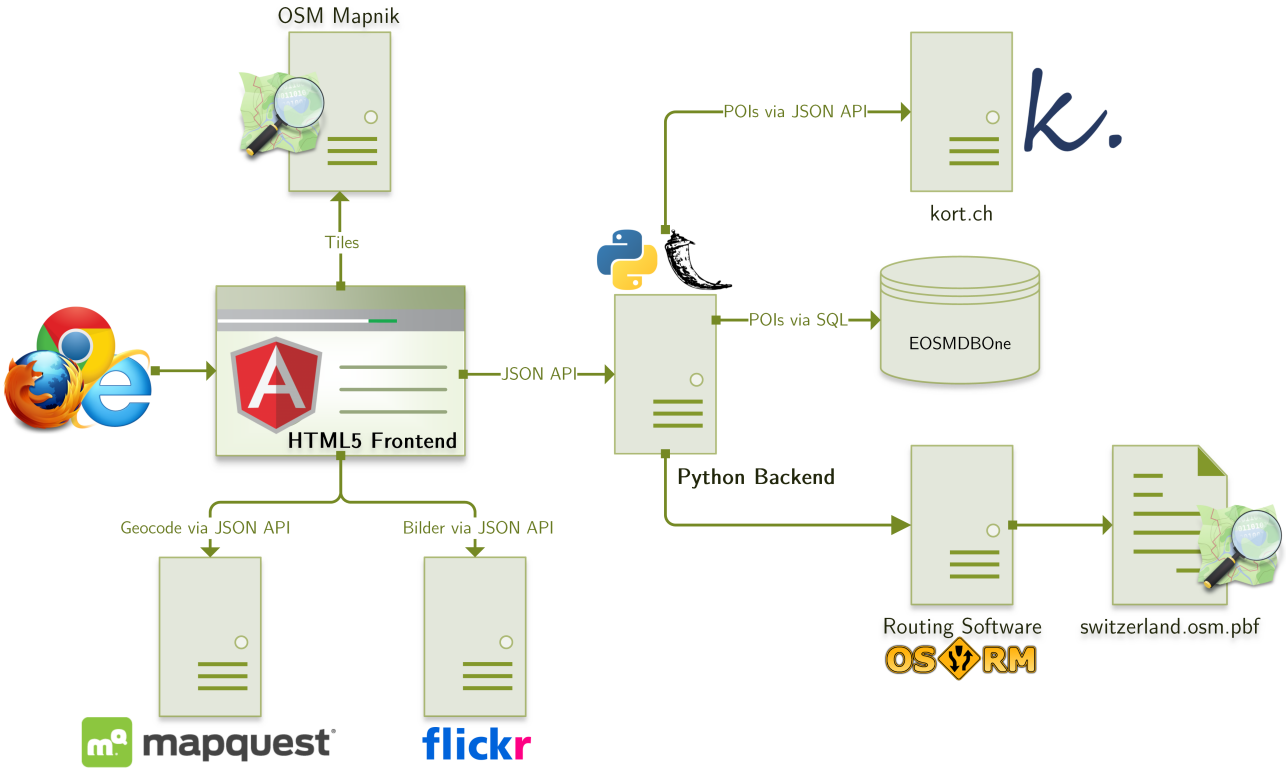


Abbildung 4.1.: Gesamtübersicht Komponenten und Datenquellen

[1] Die Daten fliessen in die entgegengesetzte Richtung der Pfeile

4.1.2. Klassen

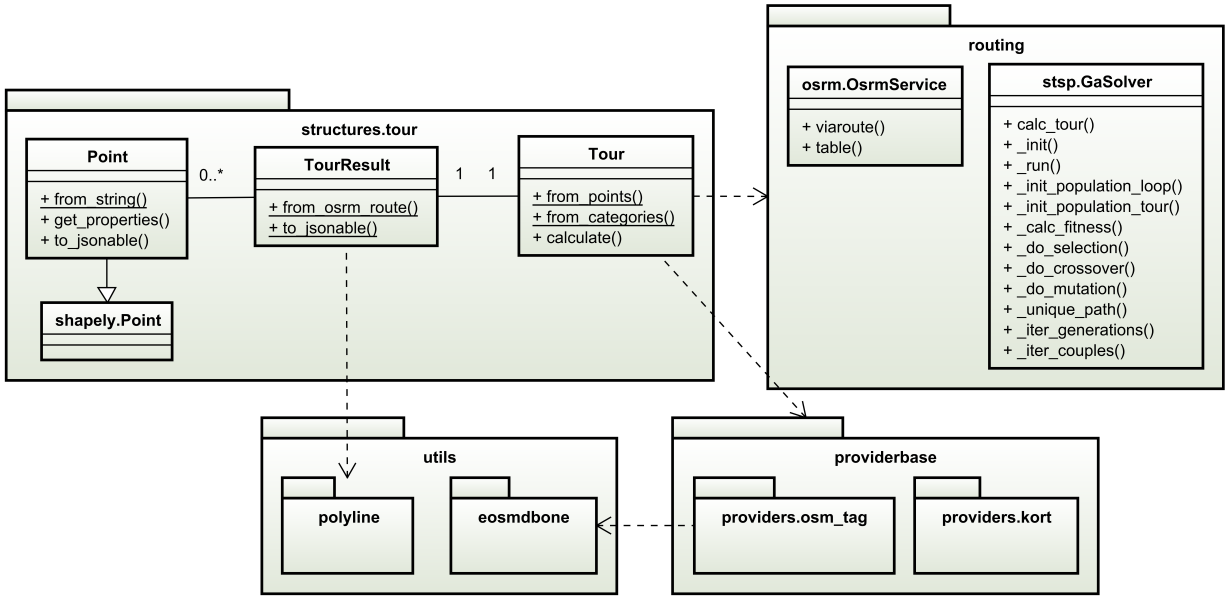


Abbildung 4.2.: Grobübersicht der Python Packages und Klassen

4.1.3. Sequenzdiagramm

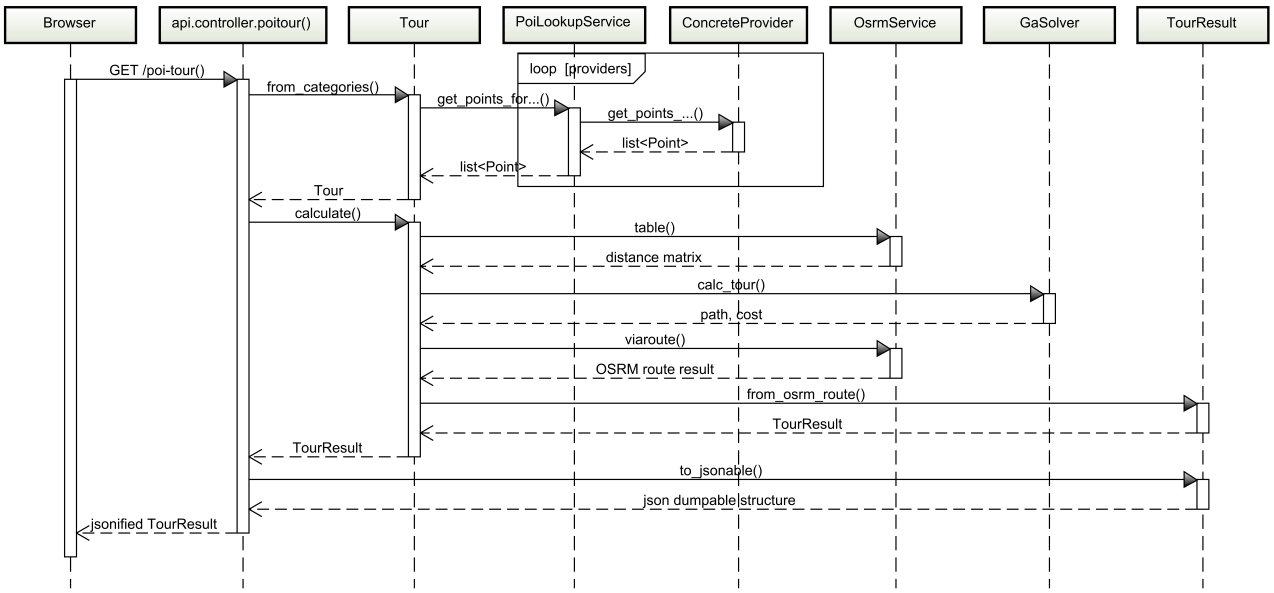


Abbildung 4.3.: Grobübersicht der Aufrufe einer Berechnung

4.2. OSRM

OSRM ist die Routing Engine, welche für das Routing zwischen den einzelnen POIs verwendet wird. Die detaillierte Evaluation der Routing Engine ist in Teil II, Kapitel 2 auf Seite 14 zu finden. OSRM wird einerseits

vor dem eigentlichen Algorithmus, zur Berechnung der Distanzen zwischen den einzelnen POIs (Distanzmatrix[2]) eingesetzt. Andererseits wird, sobald die Abfolge der POIs durch den Genetischen Algorithmus bestimmt wurde, eine Tour entlang der POIs via OSRM abgefragt.

4.2.1. Routing Profil

OSRM ist eine extem performante Routing Engine. Dessen Performance liegt in der Vorverarbeitung der OSM Daten anhand eines in Lua programmierten Routing Profils, welches unter anderem die Geschwindigkeiten diverser Streckenabschnitte (Autobahn, Fussweg, ...) festlegt.
Mit routing.osm.ch existiert bereits ein gutes Fussgänger Routing Profil für die Schweiz[3]. Das Profil ist so eingestellt, dass Geschwindigkeiten im Bereich 7 km/h–12 km/h verwendet werden um Präferenzen[4] zu erzwingen:

»Main gotcha: the profile misuses speed to get appropriate route preferences. The frontend therefore ignores the times it gets from OSRM and simply computes its own using the distance and a fixed speed.« ([6])

Da die Distanzmatrix von OSRM sinnvollerweise Zeiten und keine echten Distanzen zurückgibt, ist das Profil zur Lösung der Aufgabenstellung nicht zweckmässig. Die Alternative wären einzelne Routing Aufrufe an OSRM, welche auch die Distanzen beinhalten. Solche Routenberechnungen sind jedoch viel teurer als die Distanzmatrix, dazu kommen die $\frac{n(n-1)}{2}$ einzelnen HTTP Aufrufe zur Vervollständigung der Distanzmatrix[5], was auf jeden Fall zu Vermeiden ist.
POI Tour verwendet als Basis dennoch das Routing Profil `foot-city.lua` von routing.osm.ch, jedoch mit einer fixen Geschwindigkeit von 1 m/s[6].

4.2.2. Patch

Die Distanzmatrix Funktion von OSRM akzeptiert zum jetzigen Zeitpunkt nur höchstens 100 Punkte als Eingabe. Es gibt jedoch keinen Grund wieso dieser nicht angehoben werden soll [3] und zu dessen Konfigurierbarkeit ist bereits ein Pull-Request auf GitHub pendent.
Diese Limite kann vor dem Kompilieren von OSRM in `osrm-backend/Plugins/DistanceTablePlugin.h` in der Funktion `HandleRequest` gesetzt werden.

4.3. Kategorie-Provider

Die POI-Kategorien sowie die konkreten darunterliegenden POIs werden über sogenannte Kategorie-Provider zur Verfügung gestellt. Dies erlaubt eine beliebige Erweiterung von Kategorien und POIs durch Implementation der Provider Schnittstelle. Die in Abb. 4.4 ersichtliche Klasse "PoiProvider" gilt als Abstrakte Superklasse für alle Provider und hat eine eigene Python-Metaklasse. Die Metaklasse registriert automatisch alle konkreten Provider Subklassen die von "PoiProvider" erben und dient so als simpler Plugin-Mechanismus.

[2] Reisezeit in 1/10 s und nicht in etwa Meter
[3] Siehe auch: https://github.com/sosm/cbf-routing-profiles
[4] Ein Fussgänger sollte beispielsweise nicht die Autobahn nehmen
[5] Die Diagonale sowie alles unterhalb entfallen
[6] 3.6 km/h, Schrittgeschwindigkeit

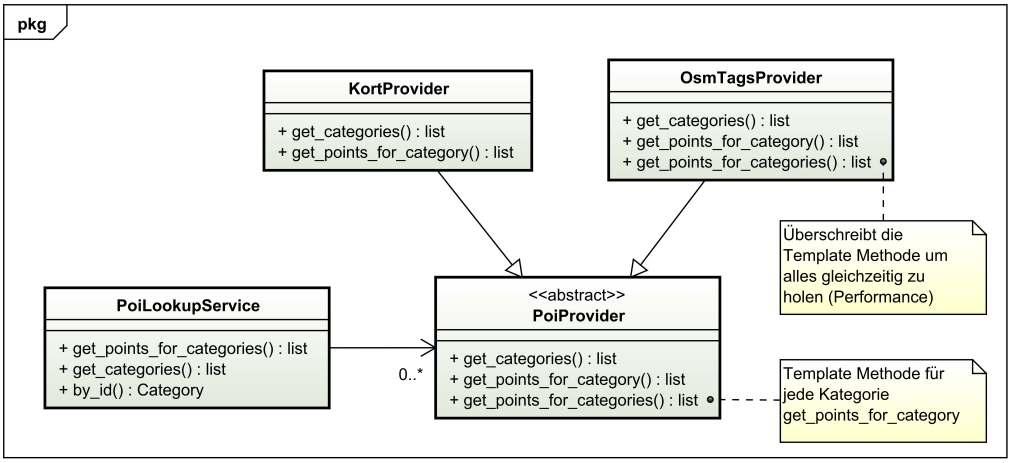


Abbildung 4.4.: Hauptklassen der Kategorie-Provider

Alle Provider erhalten zusätzlich zum Startpunkt weitere Informationen wie die maximale Distanz (Radius) die in gegebener Zeit besucht werden kann. Da es nicht möglichst ist innert gegebener Zeit und Geschwindigkeit eine weitere Strecke als die direkte Luftlinie zurückzulegen, sollten die von den Providern zurückgegebenen POIs wie in Abb. 4.5 ersichtlich limitiert werden. Dies ist nötig um den Algorithmus nicht mit unerreichbaren POIs zu belasten.

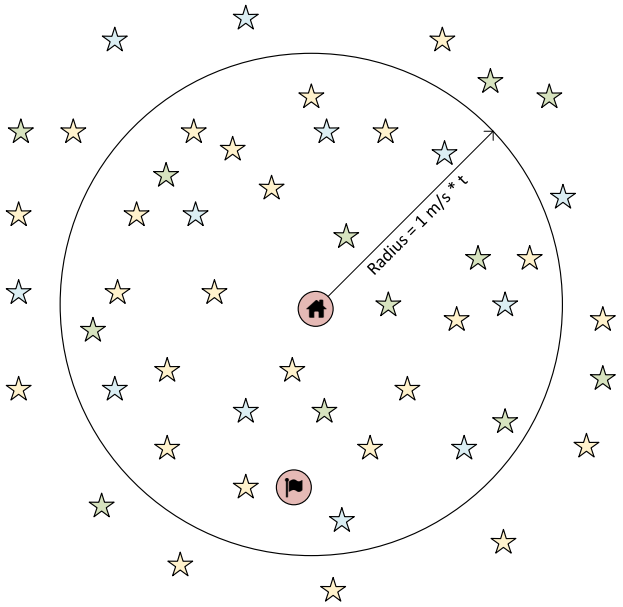


Abbildung 4.5.: Vorselektion der POIs

4.3.1. OSM POIs

Der grösste Teil der POIs werden durch die Enhanced OpenStreetMap Database One (EOSMDBOne) Datenbank des Geometa Labs bezogen. Die EOSMDBOne ist eine PostgreSQL Datenbank welche in einem ersten Schritt die OSM Daten mittels osm2pgsql in PostGIS Geometrien konvertiert und in einem zweiten Post-Processing Schritt weitere Tabellen und Views zur Verfügung stellt [9].
Diese POIs bzw. deren Kategorien werden in der YAML Konfiguration mittels

- einer ID als Zeichenkette,
- einem Namen (Einzahl),
- einem Anzeigenamen (Mehrzahl),
- einer Beschreibung (zurzeit im UI nicht verwendet),
- eine Liste von Tags
- und eine Liste von Unterkategorien

konfiguriert. In den folgenden Abschnitten werden die zwei wichtigsten Konfigurationsattribute "tags" und "includes" erläutert.

Entscheid 4.1: Konfigurationsformat (YAML)

Im Kontext der Konfiguration von Kategorie-Providern bzw. POIs aus der EOSMDBOne wurde, um die manuelle Wartung für Menschen zu erleichtern für YAML und gegen XML oder JSON entschieden und akzeptieren, dass YAML weit weniger verbreitet ist.

Tags

Das "tags" Attribut ist der Kern der darunterliegenden SQL Abfrage und ist eine Liste von Key-Value Paaren (Dictionary). Alle Tags die sich im gleichen Dictionary befinden werden konjunkt (AND) vereinigt und müssen somit beide erfüllt sein. Sind mehrere Dictionaries vorhanden, so werden die einzelnen diskjunkt (OR) vereinigt. Die Tilde vor einem Wert gilt als Komplement (NOT).

Programmcode 4.1: Beispielkonfiguration POI Kategorie

```
OsmTagsProvider:
   - id: attraction
   name: Sehenswürdigkeit
   display_name: Sehenswürdigkeiten
   description: null
   tags:
   - tourism: attraction
   attraction: ~animal
   - tourism: 'yes'
```

Die Tags der YAML Konfiguration im Programmcode 4.1 wird nach dem Parsen zu folgender Python Datenstruktur:

Programmcode 4.2: Beispielkonfiguration POI Kategorie in Python

```
[
   {'tourism': 'attraction', 'attraction': '~animal'},
   {'tourism': 'yes'}
]
```

Diese wiederum zum Prädikat "Alle POIs welche die Tags (tourism = attraction UND NICHT attraction = animal) ODER (tourism = yes)"

Unterkategorien

Die Kategorien können auch geschachtelt werden, das heisst, dass die Kategorie "Weihnachten" automatisch die POIs der Unterkategorien "Weihnachtsmarkt" und "Weihnachtsbaum" beinhaltet. Dies ist deshalb sinnvoll, weil sich so die einzelnen POIs noch individuell einordnen lassen und nicht alle nur "Weihnachts-POI" heissen. Zudem lassen sich so die Kategorien wiederverwenden und man kann einerseits "Hotel", "Hostel", "Jugendherberge", ... aber auch nur "Unterkünfte" zur Auswahl anbieten.
Programmcode 4.3 zeigt das Beispiel mit der Kategorie "Weihnachten", wobei die Unterkategorien in diesem Fall durch Auslassen des `display_name` Attributs nicht noch separat im User Interface angeboten werden.

Programmcode 4.3: Beispielkonfiguration POI Unterkategorien

```
OsmTagsProvider:
   - id: xmas_tree
   name: Weihnachtsbaum
   description: null
   tags:
      - 'xmas:feature': tree

   - id: xmas_market
   name: Weihnachtsmarkt
   description: null
   tags:
      - 'xmas:feature': market

   - id: xmas
   name: Weihnachts-POI
   display_name: Weihnachten
   description: null
   includes:
      - xmas_market
      - xmas_tree
```

Abfrageoptimierung

Grundsätzlich sind alle OSM POIs über die in Programmcode 4.4 definierte View der EOSMDBOne verfügbar.

Programmcode 4.4: Definition der View `osm_poi`

```
CREATE OR REPLACE VIEW osm_poi AS
SELECT osm_point.osm_id, osm_point.name, osm_point.tags, 'pt'::text AS gtype,
 ↪  osm_point.osm_version, osm_point.way
FROM osm_point
UNION
SELECT osm_polygon.osm_id, osm_polygon.name, osm_polygon.tags, 'po'::text AS gtype,
 ↪  osm_polygon.osm_version, st_centroid(osm_polygon.way) AS way
FROM osm_polygon;
```

Bei vielen selektierten Kategorien und dennoch wenigen POIs dauern einzelne Abfragen pro Kategorie auf die View `osm_poi` der EOSMDBOne sehr lange[7], was die EOSMDBOne zum Flaschenhals der Berechnung macht.

[7] Über 7 Sekunden bei einer Tour die schlussendlich 10 POIs enthält

Eine Analyse mit `EXPLAIN ANALYZE` der vereinfachten Abfrage, ersichtlich in Abb. 4.6, zeigt, dass die Performance vor allem durch die Verwendung von `UNION` und dem damit Verbundenen Uniqueness-Constraint negativ beeinträchtigt wird.

Programmcode 4.5: Simple `osm_poi` Abfrage mit UNION

```
SELECT osm_point.osm_id FROM osm_point
UNION
SELECT osm_polygon.osm_id FROM osm_polygon
LIMIT 10;
-- Total runtime: 45000 ms
```

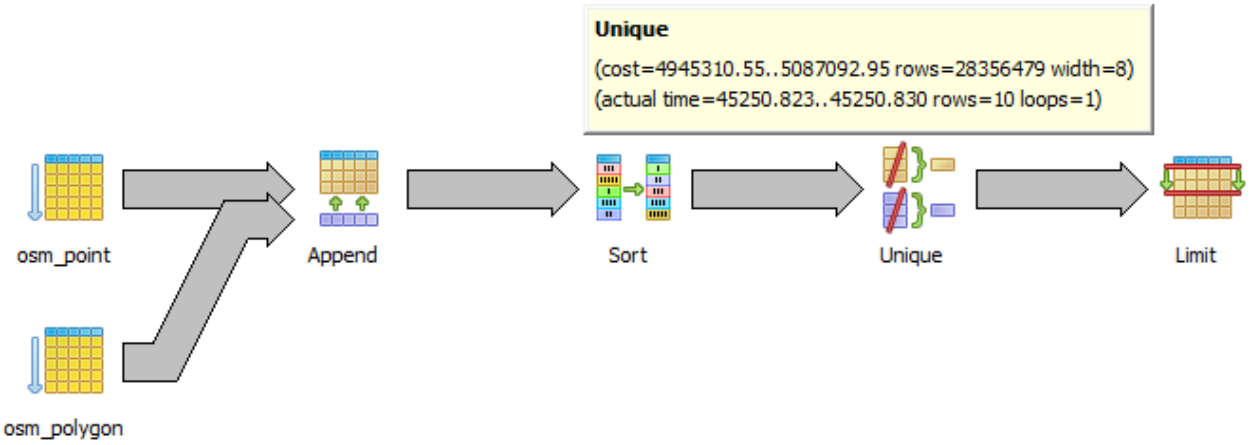


Abbildung 4.6.: EXPLAIN ANALYZE von Programmcode 4.5

Die Abfrage in Programmcode 4.6 bzw. dessen Ausführungsplan in Abb. 4.7, bei dem `UNION` durch `UNION ALL` ersetzt wurde, beendet mit Faktor 2800 schneller als die vorherige Abfrage.

Programmcode 4.6: Simple `osm_poi` Abfrage mit UNION

```
SELECT osm_point.osm_id FROM osm_point
UNION ALL
SELECT osm_polygon.osm_id FROM osm_polygon
LIMIT 10;
-- Total runtime: 16ms
```

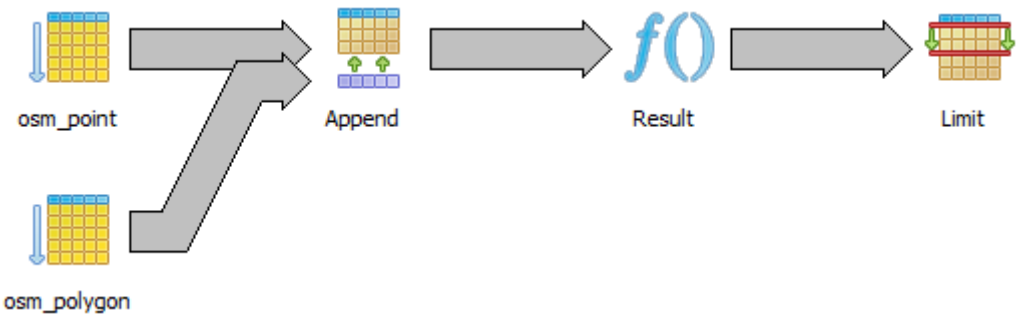


Abbildung 4.7.: EXPLAIN ANALYZE von Programmcode 4.6

Um diese Schwäche zu beseitigen wurde eine eigene Abfrage mit `UNION ALL` verwendet. Zudem ist es für die Applikation nötig zu wissen, aus welcher Kategorie der POI entstammt. Die simpelste Lösung dafür wären mehrere Abfragen und so jede Resultatmenge der entsprechenden Kategorie zuordnen. Auch dies konnte durch eine Common Table Expression (CTE) Abfrage elegant gelöst werden. Schlussendlich bleibt unabhängig von der Anzahl selektierten Kategorien eine einzige SQL Abfrage übrig. Die Form dieser Abfrage ist in Programmcode 4.7 ersichtlich und ist für das obengenannte Szenario um Faktor fünf schneller als vor der Optimierung.

Programmcode 4.7: Optimierte CTE Abfrage ohne `osm_poi`

```
WITH poi_cte AS (
   SELECT -- ...
      ST_X (st_transform(osm_point.way, 4326)) AS lon,
      ST_Y(st_transform(osm_point.way, 4326)) AS lat,
      TRUE AS is_point
   FROM osm_point
   WHERE ST_DWithin(osm_point.way, st_transform(st_geomfromwkb(%(START)s, 4326), 900913),
    ↪  %(RADIUS)s)
   UNION ALL
   SELECT -- ...
      ST_X(ST_Centroid(ST_Transform(osm_polygon.way, 4326))) AS lon,
      ST_Y(ST_Centroid(ST_Transform(osm_polygon.way, 4326))) AS lat,
      FALSE AS is_point
   FROM osm_polygon
   WHERE ST_DWithin(osm_point.way, st_transform(st_geomfromwkb(%(START)s, 4326), 900913),
    ↪  %(RADIUS)s)
)
SELECT -- ...
   %(CATEGORY_ID)s AS c_id
FROM poi_cte
WHERE poi_cte.tags @> -- ...
   AND -- ...

UNION ALL

SELECT -- ...
   %(OTHER_CATEGORY_ID)s AS c_id
FROM poi_cte
WHERE poi_cte.tags @> -- ...

UNION ALL
```

```
-- ...
```

4.3.2. Kort POIs

Zur Erfüllung der Aufgabenstellung sowie der ursprünglichen Vision aus Abschnitt 1.1 dient ein separater Kort Kategorie-Provider. Dieser Provider stellt nur eine einzige "Kort POIs" Kategorie zur Verfügung und bezieht diese via Kort Webservice [13, 8]. Dabei wurde die in Tabelle 4.2 beschriebene Methode [13, S. 40] verwendet. Die Parameter `lat`, `lng` und `radius` wurden dabei wieder so gewählt wie in Abb. 4.5 auf Seite 58.

URL	http://kort.ch/server/webservices/mission/position/
Methode	GET
Parameter	`limit`: Maximale Anzahl der zu ladenden Fehler `radius`: Radius in dem sich die Fehler befinden müssen
Antwort	200 OK – Daten konnten erfolgreich geladen werden.
Antworttyp	JSON

Tabelle 4.2.: Verwendete Kort Webservice Methode

4.4. JSON API

Für das HTML5 Frontend wurde lediglich eine JSON API zur Verfügung gestellt. Dies ermöglicht auch die Anbindung an POI Tour aus anderen, zukünftigen Applikationen.

4.4.1. Konfiguration – GET /config

Gibt alle für den HTML5 Client notwendigen Konfigurationsdaten sowie initiale Daten zurück.

Beispielantwort

Programmcode 4.8: Beispielantwort GET /config

```
{
  "categories": [
  {
    "description": null,
    "display_name": "\u00d6ffentliche Toiletten",
    "id": "toilet"
  },
  {
    "description": null,
    "display_name": "Picknick Pl\u00e4tze & Tische",
    "id": "picnic"
  },
  {
    "description": null,
```

```
    "display_name": "B\u00e4che",
    "id": "stream"
  },
  {
    "description": null,
    "display_name": "Sportpl\u00e4tze",
    "id": "pitch"
  },
  // Weitere Kategorien...
  ],
  "config": {
    "API_URL": "/api/v1.0/poi-tour",
    "ATTRIBUTION": "\n<a href=\"#/about\">Info</a> |\n<a target=\"_blank\"
     ↪  href=\"http://www.hsr.ch/geometalab\">Geometa Lab</a> |\nData &copy; ODbL <a
     ↪  target=\"_blank\" href=\"http://openstreetmap.org/copyright\">OpenStreetMap</a>
     ↪  contributors",
    "CATEGORIES": [
    "xmas",
    "tourism"
    ],
    "DEFAULT_WEIGHT": "5",
    "END": "",
    "FLICKR_API_KEY": "28362dfe07480dcca1b2f7e7b1b87c8d",
    "HOURS": 2,
    "LATLNG_DECIMAL_PRECISION": "5",
    "MINUTES": 30,
    "NOMINATIM_URL":
     ↪  "http://open.mapquestapi.com/nominatim/v1/search?countrycodes=ch&format=json&
     ↪  addressdetails=1&q={{query}}",
    "START": "",
    "STAY_TIME": 10,
    "WEIGHTS": [
    5,
    5
    ]
  }
}
```


4.4.2. POI Tour – GET /poi-tour

Der Kern von POI Tour. Berechnet eine mögliche Tour und maximiert dabei die Anzahl besuchter POIs abhängig von den Parametern.

Parameter

`start`	Startpunkt als lat,lon Zeichenkette (z.B. 47.322, 8.341)
`end`	Ziel als lat,lon Zeichenkette analog `start`
`time_s`	Die maximale Dauer (Gesamtdauer) der Tour in Sekunden
`stay_time_s`	Die gewünschte Aufenthaltsdauer pro POIs
`categories`	Eine kommaseparierte Liste von Kategorien (Interessen) die besucht werden sollen
`weights`	Eine kommaseparierte Liste von Gewichten pro Kategorie um Präferenzen auszudrücken. Diese sollten von der Reihenfolge her den Kategorien entsprechen.

Beispielanfrage

GET <apiurl>/poi-tour?categories=attraction,fountain&end=47.378058,8.5398226&start=47.378058,8.5398226&stay_time_s=900&time_s=9000&weights=5,5 HTTP/1.1

Beispielantwort 200 OK

Programmcode 4.9: Beispielantwort GET /poi-tour


```
{
  "distance_m": 2660,
  "path": {
    "coordinates": [
    [8.539781, 47.3779789],
    [8.540326, 47.377767],
    // Weitere Koordinaten...
    ],
    "type": "LineString"
  },
  "points": {
    "features": [
    {
      "geometry": {
        "coordinates": [8.5398226, 47.378058],
        "type": "Point"
      },
      "id": "start",
      "properties": {},
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [8.5396406615385, 47.3746913163422],
        "type": "Point"
      },
      "id": null,
      "properties": {
        "description": "",
        "name": "Brunnen ohne Namen",
        "url": "http://www.openstreetmap.org/node/693318521"
      },
      "type": "Feature"
    },

    // Weitere Punkte...

    {
      "geometry": {
        "coordinates": [8.5398226, 47.378058],
        "type": "Point"
      },
      "id": "start",
      "properties": {},
      "type": "Feature"
    }
    ],
    "type": "FeatureCollection"
  },
```

```
  "time_s": 2790,
  "time_total_s": 8954
}
```

4.4.3. Selective Travelling Salesman – GET /tour

API Funktion zur direkten Verwendung des STSP Algorithmus anhand eigener Punkte. Wird von POI Tour selbst nicht verwendet.

Parameter

`start`	Startpunkt als lat,lon Zeichenkette (z.B. 47.322, 8.341)
`end`	Ziel als lat,lon Zeichenkette analog `start`
`time_s`	Die maximale Dauer (Gesamtdauer) der Tour in Sekunden
`stay_time_s`	Die gewünschte Aufenthaltsdauer pro POIs
`point`	Eine Liste von lat,lon Zeichenketten als mögliche Punkte die besucht werden sollen. Die Liste entsteht durch die Mehrfachverwendung des Parameters `point`

Beispielanfrage

GET `<apiurl>/tour?start=47.376921,8.539824&end=47.376921,8.539824&time_s=10000&stay_time_s=300&point=47.372677,8.535189&point=47.374508,8.539288&point=47.371268,8.538708&point=47.367751,8.512744&point=47.376223,8.541862` HTTP/1.1

Beispielantwort 200 OK

Die GeoJSON Property "id" entspricht dem Index der übergebenen Liste von Punkten.

Programmcode 4.10: Beispielantwort GET /poi-tour

```
{
  "distance_m": 6822,
  "path": {
    "coordinates": [
    [8.539787, 47.376928],
    [8.539902, 47.377126],
    // Weitere Koordinaten...
    ],
    "type": "LineString"
  },
  "points": {
    "features": [
    {
      "geometry": {
        "coordinates": [8.539824, 47.376921],
        "type": "Point"
      },
      "id": "start",
      "properties": {},
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [8.541862, 47.376223],
```

```
        "type": "Point"
      },
      "id": 4,
      "properties": {},
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [8.539288, 47.374508],
        "type": "Point"
      },
      "id": 1,
      "properties": {},
      "type": "Feature"
    },
    // Weitere Punkte...
    {
      "geometry": {
        "coordinates": [8.539824, 47.376921],
        "type": "Point"
      },
      "id": "start",
      "properties": {},
      "type": "Feature"
    }
    ],
    "type": "FeatureCollection"
  },
  "time_s": 7818,
  "time_total_s": 9018
}
```


4.5. Berechnung / Algorithmus

Der Prototyp des Genetischen Algorithmus aus Abschnitt 3.1.2 auf Seite 51 wurde weitestgehend übernommen und die Parameter mittels "Trial and Error" optimiert. Zudem wurden konfigurierbare Mechanismen zur Steuerung der Lösungsqualität und maximaler Ausführungsdauer implementiert. Diese Konfigurationsparameter sind in Abschnitt 1.3.2 auf Seite 100 beschrieben.

4.5.1. Fitness und Gewichtung

Bei der Selektion der besten Lösungen sowie als Abbruchkriterium ist es notwendig, die einzelnen Lösungen qualitativ beurteilen zu können. Dies wird bei AI- und Optimierungsverfahren häufig Fitness genannt und wird in diesem Fall wie folgt Berechnet:

$$[(\sum_{i=0}^{AnzPOIs} Gewicht_i) + AnzPOIs] \cdot MaxZeit - Zeit$$

Abbildung 4.8.: Fitness der Lösungen

Abb. 4.8 ergibt, wenn man die Summe der Gewichte ignoriert, eine Zahl, die gleich sortiert wie eine lexikographische Sortierung nach Anzahl POIs in der Lösung (aufsteigend) und der Gesamtzeit (absteigend). Durch Einbringen der einzelnen Gewichte der POIs bei der Multiplikation führen POIs mit einem höheren Gewicht zu einer deutlich besseren Fitness.
Eine weitere Stelle im Algortihmus, wo die Gewichtung eine Rolle spielt, ist beim Einfügen von neuen POIs an einer zufälligen Stelle der Tour. Dies ist in Pseudocode 3.4 auf Seite 51 ersichtlich, wo die einzufügenden Punkte erst profitmaximierend sortiert werden. Im Falle von keiner Gewichtung, was im User Interface einer identischen Gewichtung aller Kategorien entspricht, wird lediglich nach Distanz vom vorherigen POI aus sortiert. Ist hingegen eine Gewichtung vorhanden werden die Kandidaten erst aufsteigend nach Gewicht dann absteigend nach Distanz sortiert. Die Unterscheidung ob die Gewichte identisch sind und somit weniger sortiert werden muss, kann bei den typisch vielen Iterationen von solchen Optimierungsverfahren einen Unterschied von mehreren Sekunden ausmachen.

4.5.2. Performance

Die Performance des Algorithmus ist Dank der Verwendung von NumPy sowie eine bereits im Vorhinein performancebewusste[8] Entwicklung und Verzicht auf übermässige Abstraktion[9] bei durchschnittlicher Verwendung[10] relativ gut. Das Akzeptanzkriterium von Epic 2 auf Seite 43 wird erfüllt. Die Qualität und somit indirekt die Dauer der Ausführung kann mittels Konfigurationsparametern gesteuert werden.

4.6. User Interface

Wie bereits aus Teil II, Abschnitt 3.2 auf Seite 36 hervorgeht, ist das Frontend mit AngularJS und Leaflet implementiert.
Abb. 4.9 zeigt den groben Aufbau der Seite und einigen Angular Controllern bzw. Direktiven. Die Applikation ist als SPA entwickelt. Der Webserver stellt nur die statische Applikation[11] zur Verfügung. Diese lädt alles andere wie Konfiguration und Daten via XHR nach.

[8] Keine "premature optimization" aber auch keine unnötigen Operationen, Loops, etc.
[9] You Ain't Gonna Neet It (YAGNI)
[10] Eine vierstündige Tour in der Stadt Zürich mit drei bis vier Kategorien
[11] HTML, JavaScript, CSS, Bilder, ...

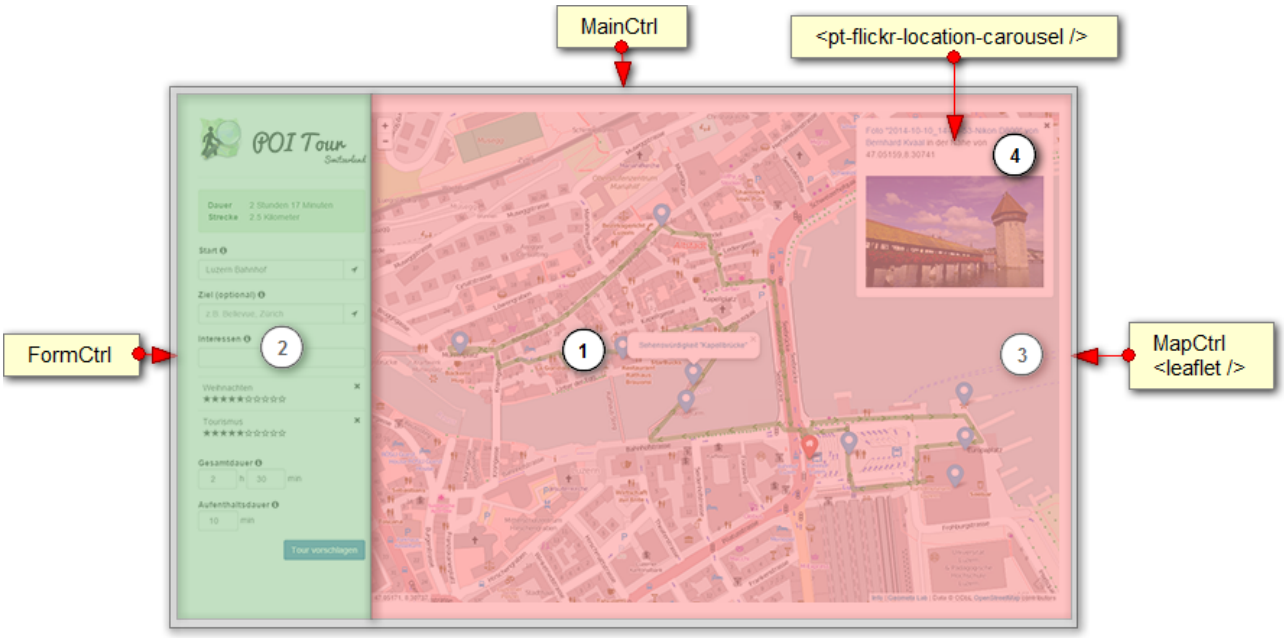


Abbildung 4.9.: Übersicht Aufbau Frontend

4.6.1. Geocoding (Nominatim)

Als Geocoding versteht man die Auflösung eines Namens oder einer Adresse zu bestimmten geographischen Koordinaten. Diese Auflösung findet direkt im Frontend mittels Nominatim statt, einem Geocoding Webservice für OSM. Standardmässig ist der Nominatim Server von MapQuest[12] konfiguriert, da dieser in dessen Nutzungsbedingungen mehr Anfragen pro Zeit erlaubt.

4.6.2. Fotos (Flickr)

Zur Anzeige von Fotos der POIs wurde die API[13] von Flickr verwendet. Eine Radius-Abfrage in der Umgebung von 20 m um den POI mit der Standard-Sortierung[14] bringt gute Ergebnisse. Weitere Experimente mit Sortierung nach Distanz, Interesse oder Relevanz führten zu schlechteren Ergebnissen oder nicht relevanten Fotos. Auch werden Portrait-Fotos herausgefiltert, da diese oft zu Personenfotos führt. Die bessere Alternative wäre Panoramio von Google, welche jedoch nur in Kombination mit Google Maps verwendet werden darf und somit ausgeschlossen ist. Auch auf die Integration von Mapillary, einer offenen Google Street View Alternative, wurde verzichtet. Das Projekt ist noch relativ neu und bietet deshalb nur sehr wenig Fotos.

4.7. Testing

In den nachfolgenden Abschnitten 4.7.1 und 4.7.2 wird beschrieben, wie und womit die Applikation getestet wird.

[12] http://open.mapquestapi.com/nominatim/
[13] `flickr.photos.search`, siehe auch: https://www.flickr.com/services/api/flickr.photos.search.html
[14] Datum, neuste zuerst

4.7.1. Backend

Für das Backend wurden Python Unit Tests[15] für folgende kritische Komponenten erstellt:

- Der Genetische Algorithmus für das Selective Travelling Salesman Problem (STSP)
- Die Pluggable Kategorien-Provider
- Die Helper Module mit der Business Logik für die Tourenberechnung

Diese Tests sind Projektverzeichnis unter `backend/tests` zu finden und lassen sich individuell als Python Modul aber oder mit dem externen Testrunner namens Nose ausführen. Tests welche die Datenbank benötigen und somit nicht auf dem Continuous Integration (CI) ausgeführt werden sollen können mit dem Decorator `@requires_db` annotiert werden.

```
test_get_categories (tests.providerbase_tests.ProviderTestCase) ... ok
test_get_points_for_category (tests.providerbase_tests.ProviderTestCase) ... ok
test_registration (tests.providerbase_tests.ProviderTestCase) ... ok
test_calc_fitness (tests.stsp_tests.GeneticAlgorithmTestCase) ... ok
Overall test ... ok
test_init_tour (tests.stsp_tests.GeneticAlgorithmTestCase) ... ok
test_init_tsp (tests.stsp_tests.GeneticAlgorithmTestCase) ... ok
test_max_runtime (tests.stsp_tests.GeneticAlgorithmTestCase) ... ok
test_unique_path (tests.stsp_tests.GeneticAlgorithmTestCase) ... ok
test_from_string (tests.tour_tests.PointTestCase) ... ok
test_jsonable (tests.tour_tests.PointTestCase) ... ok
test_point (tests.tour_tests.PointTestCase) ... ok
test_jsonable (tests.tour_tests.TourResultTestCase) ... ok
test_calculate (tests.tour_tests.TourTestCase) ... ok
test_from_categories (tests.tour_tests.TourTestCase) ... ok
test_from_points (tests.tour_tests.TourTestCase) ... ok
```

4.7.2. Frontend

Unit Tests

AngularJS erlaubt es, die Applikation in lose gekoppelten Komponenten zu strukturieren, welche untereinander nur via Dependency Injection eine Objektreferenz erhalten können. Deshalb ist es möglich Unit Tests für einzelne Komponenten[16] zu schreiben und alle Abhängigkeiten zu "Mocken". Direktiven und Services werden hauptsächlich durch Unit Tests, Controller hingegen mit E2E Tests getestet.
Die Unit Tests sind im Verzeichnis `frontend/test/spec` zu finden.

Ent-to-End Tests

End-to-End (E2E) Tests, oftmals auch Integrationstests genannt, testen nicht nur eine einzelne, sondern gleich mehrere, bestenfalls alle Komponenten einer Applikation. Wie der Name End-to-End bereits suggeriert wird zwar aktiv am einen Ende (Frontend) getestet, diese reichen jedoch bis ans andere Ende (meist die Datenbank/Datenquelle) der Applikation. Dies geschieht durch den Einsatz von Protractor, einem UI Testing Framework speziell für AngularJS, ebenfalls automatisiert. Protractor exponiert eine simple DSL für das darunterliegende Selenium bzw. die WebDriver API und kümmert sich um die Synchronisation wie beispielsweise das Abwarten eines XHR Requests.

[15] Python Standard Library: `unittest` Modul
[16] AngularJS Services, Direktiven und Controller

Der Nachteil von solchen Tests ist die Notwendigkeit der gesamten, laufenden Applikation sowie einem Browser. Aus diesem Grund werden diese nicht mittels CI Server ausgeführt.
Die E2E Tests sind im Verzeichnis `frontend/test/spec_e2e` zu finden.

5. Resultate und Ausblick

5.1. Resultate

Das Resultat dieser Studienarbeit ist in Teil I, Kapitel 3 auf Seite 9 beschrieben.
Dabei ist zu erwähnen, dass die User Stories 2.6 bis 2.8 nicht weiter verfolgt oder durch den Porduct Owner mit niedriger Priorität eingestuft wurden.
Die User Story 2.6 "Nahegelegene Sehenswürdigkeiten integrieren" wurde als nicht sinnvoll erachtet. Die Tour wird ohnehin bereits bezüglich der Anzahl POIs optimiert. Ein solches Verhalten kann also wahlweise durch zusätzliche Auswahl der Kategorie "Sehenswürdigkeiten" erreicht werden.

5.2. Weiterentwicklung

Die erste Version von POI Tour hat die Grundsätzliche Machbarkeit einer solchen Touren-Applikation sowie eine mögliche Implementation davon aufgezeigt. Dennoch gibt es zahlreiche weitere Ideen sowie kleine Verbesserungen welche nicht mehr für den Scope dieser Arbeit gereicht haben.

5.2.1. Szenarien

Weiterentwicklung durch Autor

Fabio Scala entwickelt nach der kommenden Bachelorarbeit punktuell an POI Tour weiter. Je nach Interesse und Verfügbarkeit von F. Scala ist dies ein realistisches Szenario.

Weiterentwickluing an der HSR

Eine Weiterentwicklung von POI Tour an der HSR durch andere Studenten im Rahmen einer Studien- oder Bachelorarbeit. Für dieses Szenario ist jedoch eine konkrete Vorstellung und ein Projektscope der richtigen Grösse notwendig.

Fork auf GitHub

Das Projekt ist öffentlich und unter der Massachusetts Institute of Technology (MIT) Lizenz auf GitHub verfügbar. Eine dritte Partei oder ein interessierter Entwickler hat somit die Möglichkeit das Projekt selbständig weiterzuentwickeln oder als Basis für eine andere Applikation zu verwenden.

5.2.2. Kleine Verbesserungen

OSRM Profil

Das jetzige Profil verwendet fixe Geschwindigkeiten, wodurch viele der Vorteile des Profils von routing.osm.ch verloren gehen. Sobald OSRM eine Distanztabelle mit Distanzen statt Zeiten bietet, sollte wieder auf das unveränderte Profil umgestellt werden.

Performance
Die Performance des Algorithmus ist bereits akzeptabel, dennoch wäre eine Optimierung durch Profiling sinnvoll. Häufig werden Algorithmen auch mit Cython optimiert um nativen Maschinencode zu generieren.

Zeit im UI
Die Routing (`viaroute`) Funktion von OSRM verwendet zur Zeit nicht denselben Algorithmus zur Bestimmung des nächsten Streckenabschnittes wie bei der Distanztabelle. Aus diesem Grund gibt es zwischen dem Resultat von `viaroute` und `table` eine kleine Differenz in der Zeit. Um dem entgegenzuwirken wird im User Interface das Total aus dem Genetischen Algorithmus angezeigt. Diese Differenz ist nur vorübergehend und wird in einer späteren Version von OSRM behoben.

Error Handling
POI Tour macht im User Interface keine Unterscheidung, ob bei der Berechnung einer Tour keine POIs in der Umgebung gefunden wurden oder ob lediglich Start und Ziel zu weit auseinander liegen. Eine präzisere Fehlermeldung wäre angebracht.

Kompakteres UI
Das Formular wächst mit der Anzahl ausgewählter Kategorien in die Vertikale. Je nach Anzahl oder grösse des Bildschirms muss gescrollt werden. Eine Platzersparnis kann durch Anzeige der Kategorien auf nur einer Zeile, beispielsweise durch die Verringerung der Gewichtungsmöglichkeit auf 1–5 erreicht werden.

Andere Tiles
Zusätzlich zu den jetzigen Tiles von OSM Mapnik, könnten weitere Styles wie beispielsweise der Swiss Style, MapBox, MapQuest, Google etc. im User Interface zur Auswahl stehen.

Mobile Kompatibilität
Die Verwendung von POI Tour ist auf den kleinen Bildschirmen von Smartphones kaum möglich. Ein möglicher "Quick-Fix" wäre der Einsatz eines "Swipe-Menus" wie auf nativen Android Geräten üblich. Als mögliche Implementation bietet sich Snap.js[1] an.

5.2.3. Neue Funktionalität

Mobile Applikation
Der grösste der Teil der Internetnutzer verwenden zum surfen Mobilgeräte. Aus diesem Grund würde eine Webapplikation speziell für Mobilgeräte oder gar eine native Implementation POI Tour attraktiver machen. Um die Wiederverwendbarkeit der bestehenden Codebasis zu erhöhen bieten sich diverse AngularJS Lösungen wie Mobile Angular UI[2] oder Ionic[3] an.

Komplexe Bedingungen
POI Tour versucht jeweils die Anzahl der besuchten POIs in einer Tour zu maximieren. Doch bei vielen der jetzigen Kategorien wie beispielsweise "Restaurants" macht dies kaum Sinn. Aus diesem Grund wäre die Implementation von diversen "komplexen Constraints" pro Kategorie sinnvoll. Der Benutzer möchte während der Tour nur an einem Restaurant vorbeikommen und zwar wenn möglich etwa zwei Stunden nachdem er vom Start losgelaufen ist. Für eine solche Funktionalität wären massive Anpassungen am Algorithmus wie auch am User Interface nötig.

[1] Siehe auch: https://github.com/jakiestfu/Snap.js/ bzw. http://jtrussell.github.io/angular-snap.js/
[2] Siehe auch: http://mobileangularui.com/
[3] Siehe auch: http://ionicframework.com/

Einzelne POIs auslassen
Dem Benutzer wird eine Tour vorgeschlagen mit einem POI, den er bereits kennt oder er auf keinen Fall besichtigen möchte. Dazu könne man eine Funktion, beispielsweise im bereits existierenden Kontextmenu, implementieren um einzelne POIs auszuschliessen.

Eigene POIs
Der Benutzer ist zwar mit der vorgeschlagenen Tour sowie den vorhandenen Kategorien von POI Tour zufrieden, doch er muss zwingend noch an einem ganz bestimmten Ort vorbei, möglicherweise um etwas abzuholen. Aus diesem Grund soll es möglich sein, einzelne Viapunkte der Tour hinzuzufügen. Auch dies wäre im User Interface mit dem Kontextmenu lösbar.

Weitere POI Provider
OSM bietet bereits sehr viele POIs an. Dennoch wären zusätzliche POIs aus anderen Quellen wünschenswert. Auch spezielle Kategorien analog den Kort POIs, wie beispielsweise Geocaches, wären denkbar.

POIs sofort anzeigen
Je nach Ortschaft sind sehr wenig wenn überhaupt POIs vorhanden.Deshalb sollen alle in Frage kommenden POIs der ausgewählten Kategorien und unter Berücksichtigung der Gesamtdauer, auf der Karte angezeigt werden.

Exportmöglichkeit
Der Benutzer soll die Möglichkeit haben, die Tour zu exportieren, um Unterwegs darauf zurückgreifen zu können. Dies ist auch deshalb sinnvoll, weil sich die durch POI Tour vorgeschlagenen Touren auch bei gleichen Eingabeparametern stark unterscheiden können und sich eventuell nicht reproduzieren lassen.

Textuelle Beschreibung
Der Benutzer soll die Möglichkeit haben, eine textuelle Beschreibung der Tour analog routing.osm.ch darzustellen.

Teil IV.

Projektmanagement

1. Vorgehensmodell

Als Vorgehensmodell für die Studienarbeit (SA) wurde Scrum verwendet. Scrum ist ein agiles, iteratives Modell, welches sich aufgrund dessen Einfachheit und nicht besonders strikter Vorgaben besonders für kleinere, gut überschaubare Projekte eignet. Scrum basiert wie viele andere agile Projektmanagementmethoden auf den vier Grundsätzen die im Agilen Manifest [2] beschrieben sind:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Wobei der Punkt »Working software over comprehensive documentation« aufgrund der Tatsache, dass es sich um eine akademische Arbeit handelt, vernachlässigt wird.

1.1. Rollen

Rolle	Beschreibung
Product Owner	Der Product Owner bestimmt die Anforderungen (User Stories) und priorisiert diese. Die Priorität gibt an, welche User Stories es im nächsten Sprint zu erledigt gilt.
Team	Das Team besteht aus allen für das Projekt tätige Entwickler.
Scrum Master	Der Scrum Master kümmert sich um die fortlaufende Kontrolle des Fortschritts sowie Optimierungsmöglichkeiten. In der Praxis ist er zudem für die Leitung der Daily Scrum Meetings, welche in diesem Projekt wegfallen, zuständig.

Tabelle 1.2.: Scrum Projektrollen

1.2. Artefakte

Artefakt	Beschreibung
Product Backlog	Der Product Backlog enthält alle noch nicht abgeschlossenen User Stories des Projekts. Vor Beginn des neuen Sprints werden die noch offenen User Stories vom Product Owner repriorisiert und dessen Aufwand vom Team, in diesem Falle von einem einzigen Entwickler, geschätzt.
Sprint Backlog	Der Sprint Backlog enthält alle für einen bestimmten Sprint ausgewählte User Stories. Das Team setzt die Quantität fest und der Product Owner bestimmt den Inhalt und somit welche User Stories in den nächsten Sprint einfliessen.
Burndown Charts	Mittels Jira werden laufend die Ist- und Sollstunden des Sprints addiert und daraus ein regressives Diagramm erstellt. Dies soll aufzeigen wie man zeitlich steht um mögliche Zeitprobleme zu vermeiden.

Tabelle 1.4.: Scrum Artefakte

1.3. Sprints

Die Iterationen, bei Scrum Sprint genannt und in Abb. 1.1 dargestellt, dauerten aufgrund der kleinen Teamgrösse von nur einem Entwickler jeweils drei Wochen. So konnte, wie bei Scrum üblich, zum Ende jedes Sprints ein Deliverable, sei es eine lauffähige Applikation oder ein anderes Resultat, zur Verfügung gestellt werden.

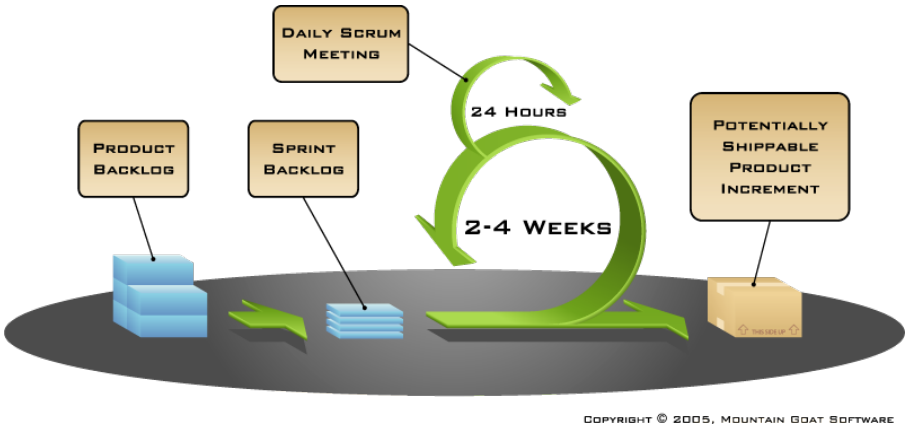


Abbildung 1.1.: Scrum Sprint [14]

1.4. User Stories

Eine User Story ist eine in der Sprache des Benutzers formulierte Anforderung an das System, woraus für ihn ein bestimmter Mehrwert entsteht. Ein Benutzer ist nicht zwingend der Benutzer der Applikation, sondern eine Rolle und kann somit auch ein Entwickler, Administrator, etc. sein. Da User Stories immer dieselbe Struktur besitzen, wurden zu Beginn des Projekts Story Card Schablonen wie in Abb. 1.2 und 1.3 eingesetzt.

SA HS2014 – Personalisierter Routenplaner für Fussgänger Hochschule für Technik Rapperswil

☐ Epic / ☐ User Story

Name [PN— _____] Priorität: 1 2 3

_____ Schätzung: _____h

Als ein (Rolle)

_____ ☐ Entwickler ☐ Benutzer

, will ich (Ziel)

_____

_____

_____

_____

_____

, sodass (Nutzen)

_____

_____

_____

_____

_____

Fabio Scala (fscala)

Abbildung 1.2.: Vorderseite einer Story Card

SA HS2014 – Personalisierter Routenplaner für Fussgänger Hochschule für Technik Rapperswil

Akzeptanzkriterien

(Usability, Funktionalität, Error Handling, Performance, NFRs)

Gegeben (Bedingung/Kontext)

_____

_____

_____

, wenn (Ereignis/Verhalten)

1. _____

2. _____

3. _____

4. _____

5. _____

, dann (Ergebnis)

1. _____

2. _____

3. _____

4. _____

5. _____

Fabio Scala (fscala)

Abbildung 1.3.: Rückseite einer Storycard

Aufgrund der zunächst unbekannten Anforderungen an das Projekt wurden zu Beginn nur Epics erfasst. Epics sind grobgranulare User Stories, welche danach in einzelne User Stories unterteilt werden. Im Gegensatz

zur User Story sind Epics Sprint-übergreifend. Die Schätzung der User Stories erfolgte nicht wie üblich mit einer virtuellen Einheit (z.B. Story Points), sondern in Stunden. Die virtuelle Einheit eignet sich für ein Projekt mit einer solch kurzen Dauer wenig, da die Anzahl machbarer Story Points pro Iteration[1] sich erst über mehrere Iterationen hinweg einstellt.

[1] Auch Velocity (Geschwindigkeit) genannt

2. Rollen und Verantwortlichkeiten

2.1. Prof. Stefan Keller

Prof. S. Keller, Mitarbeiter des Institut für Software (IFS) und Dozent an der HSR übernimmt im Projekt eine Doppelrolle:

- Als Experte bzw. Betreuer übernimmt er eine die Aufsicht und Bewertung der SA
- und als fiktiver Kunde die Kontaktperson zur Aufnahme der Anforderungen (Product Owner).

2.2. Fabio Scala

Fabio Scala, Teilzeitstudent an der HSR und Banking Software Engineer bei Swisscom IT Serivces übernimmt als einziger alle restlichen Rollen im Projekt:

- Das Entwicklungsteam
- Scrum Master

3. Risiken

Für die Abschätzung der Projektrisiken wurde auf existierende [1] Risikochecklisten gesetzt um möglichst keine potenziellen Risiken bei der Analyse auszulassen. Die sehr umfangreiche Risikoliste von Wallace und Keil [16] wurde mit den bekannten Top Zehn Software-Projektrisiken aus dem Jahre 1989 [4] sowie mit eigenen, Projektspezifischen Risiken ergänzt. Es erscheint wichtig, auch für das Projekt nicht relevante Risiken in der Liste beizubehalten, um später nachzuvollziehen, dass auch diese analysiert und nicht etwa ignoriert oder gar vergessen wurden.

3.1. Initiale Risikoanalyse

Bis zum Ende des ersten Sprints wurde eine Risikoanalyse des Projekts durchgeführt. Tabelle 3.1 zeigt eine erste ausgefüllte Risikoanalyse. Sowohl für den Schadenspotenzial wie auch für die Eintrittswahrscheinlichkeit wurde eine Skala von eins bis sechs gewählt, mit der Ausnahme von komplett ausschliessbaren oder nicht auf dieses Projekt übertragbaren Risiken, die mit null (keinem Risiko) bewertet wurden. In den nachfolgenden Abschnitten wird näher auf die besonders kritischen Risiken und deren Massnahmen eingegangen.

ID	Kategorie	Risiko	Schadens-potenzial	Eintritts-wahrschei-nlichkeit	Schaden	Präventionsmassnahme	Massnahme bei Eintritt
R01	Mensch	~~Teammitglied sträubt sich gegen Änderungen~~	-	0	0	-	-
R02		~~Konflikte im Team~~	-	0	0	-	-
R03		Negative Einstellung im Team	3	1	3	Frühzeitig bei Projektaufsicht (Prof. S. Keller) melden	-
R04		Fehlende Motivation im Team	3	1	3	Frühzeitig bei Projektaufsicht (Prof. S. Keller) melden	-
R05		~~Fehlende Kooperation im Team~~	-	0	0	-	-
R06	Anforderungen	Ständig ändernde Anforderungen	1	2	2	Agile Vorgehensweise. Häufige Projektsitzungen mit dem Kunden	Agile Vorgehensweise erlaubt Anpassungen und Repriorisierung der Anforderungen
R07		Anforderungen ungenügend erfasst	1	3	3	Agile Vorgehensweise. Häufige Projektsitzungen mit dem Kunden	Agile Vorgehensweise erlaubt Anpassungen und Repriorisierung der Anforderungen
R08		Unklare Anforderungen	2	3	6	Agile Vorgehensweise. Häufige Projektsitzungen mit dem Kunden	Agile Vorgehensweise erlaubt Anpassungen und Repriorisierung der Anforderungen
R09		Falsche Anforderungen (Umsetzung der falschen Funktionalität)	3	1	3	Agile Vorgehensweise. Am Ende jedes Sprints wird das lauffähige Produkt mit dem Kunden besprochen.	Change Request als neue User Story in den Product Backlog aufnehmen
R10	Komplexität	Verwendung neuer Technologien	5	0	0	Es werden keine neuen (bleeding edge, neu auf dem Markt) Technologien verwendet	-
R11		Hohe technische Komplexität	4	2	8	Machbarkeit frühzeitig abschätzen	Einbezug externer Experten (z.B. aus dem IFS)
R12		Verwendung unreifer Technologien	5	2	10	Evaluation der Technologien: Verwendung etablierter Technologien.	Technologiewechsel
R13		Verwendung noch nie zuvor eingesetzter Technologie	2	4	8	Evaluation der Technologien: Einsatz von Technologie mit steiler Lernkurve	Einbezug externer Experten
R15	Planung & Kontrolle	Fehlende Projektmanagement Plattform	4	1	4	Verwendung von Atlassian JIRA, was sich in der Praxis mehrfach bewährt hat	Ausweichen auf Excel Spreadsheets in der Cloud
R16		Projektfortschritt wird ungenügend überwacht	4	1	4	Atlassian JIRA bietet umfangreiche Auswertungsmöglichkeiten. Tägliche und Wöchentliche Auswertungen planen.	-
R17		Benötigte Ressourcen werden falsch abgeschätzt (oder: Ressourcen fix, Aufwand wird falsch abgeschätzt)	3	2	6	Burndown Charts zur frühzeitigen Erkennung	Verschieben von User Stories in den Product Backlog. Repriorisierung mit dem Kunden.
R18		Ungenügende Projektplanung			0		
R19		Meilensteine ungenügend definiert	2	3	6	Umfang der Sprints zu Beginn provisorisch festlegen	Nach Bedarf User Stories repriorisieren und in andere Sprints verschieben
~~R20~~		~~Unerfahrene Projektmanager~~	-	0	0	-	-
R21		Ungenügende Kommunikation	3	1	3	Wöchentliche Projektsitzungen. In einem Team wären Daily Standup-Meetings üblich.	Ergänzung durch E-Mail Verkehr.
R22		Gold-plating	3	2	6	Der Kunde setzt die Prioritäten vor jedem Sprint neu.	-
R23		Personalausfall (Krankheit, Unfall, ...)	4	2	8	Höhere Gewalt, keinen Einfluss bei Einzelarbeit.	Projektscope reduzieren
R24	Team	~~Unerfahrene Teammitglieder~~	5	1	0	Siehe projektspezifische Risiken (Einarbeitung Technologien)	-
R25		Fehlendes Know-how im Team	4	2	8	Siehe projektspezifische Risiken (Einarbeitung Technologien)	Einbezug externer Experten (z.B. aus dem IFS)
R26	Organisation	Organisatorische Anpassungen (z.B. Betreuer oder Ansprechperson des Kunden ändert)	5	1	5	Höhere Gewalt, keinen Einfluss.	-
~~R27~~		~~Negativer Einfluss durch Corporate-Politik~~	-	0	0	-	-
~~R28~~		~~Instabile Unternehmensorganisation~~	-	0	0	-	-
~~R29~~		~~Firmenorganisation wird umstrukturiert~~	-	0	0	-	-
R30	Projektspezifisch	Ausfall Infrastruktur des Entwicklers (Notebook)	3	2	6	Trotz Einzelarbeit möglichst Oft die commits auf den Server pushen.	Notebook neu aufsetzen. Verlust von etwa 6h wird selbst getragen.
R31		Einarbeitung in Python erfordert mehr Zeit als angenommen	3	0	0	-	-
R32		Einarbeitung in verwendetes Webfrontend-Framework erfordert mehr Zeit als angenommen	3	2	6	Evaluation zu Beginn des Projekts	Einbezug externer Experten (z.B. aus dem IFS)
R33		Einarbeitung in verwendetes Backend-Framework erfordert mehr Zeit als angenommen	3	2	6	Evaluation zu Beginn des Projekts	Einbezug externer Experten (z.B. aus dem IFS)
R34		Der BitBucket Git-Server fällt temporär aus	1	1	1	-	Warten bis behoben. Nicht kritisch das Einzelarbeit.
R35		Die Atlassian JIRA OnDemand Instanz fällt temporär aus	2	1	2	Skizze der User Stories sind auf Papier vorhanden.	Warten bis behoben. Nicht kritisch das Einzelarbeit.
R36		Es treten technische Hürden auf, die nicht vorhergesehen wurden und nicht zu bewältigen sind	5	1	5	Evaluation zu Beginn des Projekts.	Einbezug externer Experten (z.B. aus dem IFS)

Tabelle 3.1.: Erste Risikoanalyse

3.1.1. R11 – Hohe technische Komplexität

Die Aufgabenstellung, aus vielen möglichen Punkten eine optimale Tour zu erstellen, stellt eine gewisse Herausforderung in Sachen Algorithmik und technischer Umsetzung dar. Aus diesem Grund soll die Machbarkeit möglichst früh abgeschätzt werden.
Bei Hindernissen bezüglich der des Algorithmus wird bei Dozenten[1] der HSR angefragt.

3.1.2. R12, R25, R32, R33 – Verwendung unreifer/ungeeigneter Technologien

Vor allem bei Open Source Software (Frameworks, Libraries, etc.) liegt oftmals eine breite Auswahl an ähnlichen Optionen vor. Es gilt, sich für das Projekt am geeignetsten zu evaluieren um spätere Überraschungen zu verhindern.

3.1.3. R13 – Verwendung fremder Technologie

Zur Komplexität der Aufgabenstellung kommt noch ein technisches Umfeld mit vielen diversen Technologien[2] hinzu in die es sich davor erst einzuarbeiten gilt.
Bei technischen Fragen wird bei Prof. Stefan Keller oder erfahrenen Mitarbeitern des IFS angefragt.

3.1.4. R23 – Personalausfall

Fabio Scala fällt über längere Zeit aus (erkrankt, verunfallt, . . .). Dieses Risiko kann aufgrund der Einzelarbeit nicht ausgeschlossen werden. Die Eintrittswahrscheinlichkeit ist dafür auch entsprechend gering.

3.2. Neubewertung der Risiken nach Sprint 3

Bis auf R23 (Abschnitt 3.1.4, Personalausfall) konnten alle kritischen Risiken augeschlossen werden.

[1] z.B. Prof. Dr. Ruedi Stoop, Dozent "Artificial Intelligence"
Prof. Dr. Andreas Müller, Dozent "Mathematik für Informatiker"
[2] Flask, AngularJS, OSRM, pgRouting, OSM, PostGIS, . . .

4. Entwicklungsumgebung

Abb. 4.1 zeigt eine Übersicht der Entwicklungsumgebung.

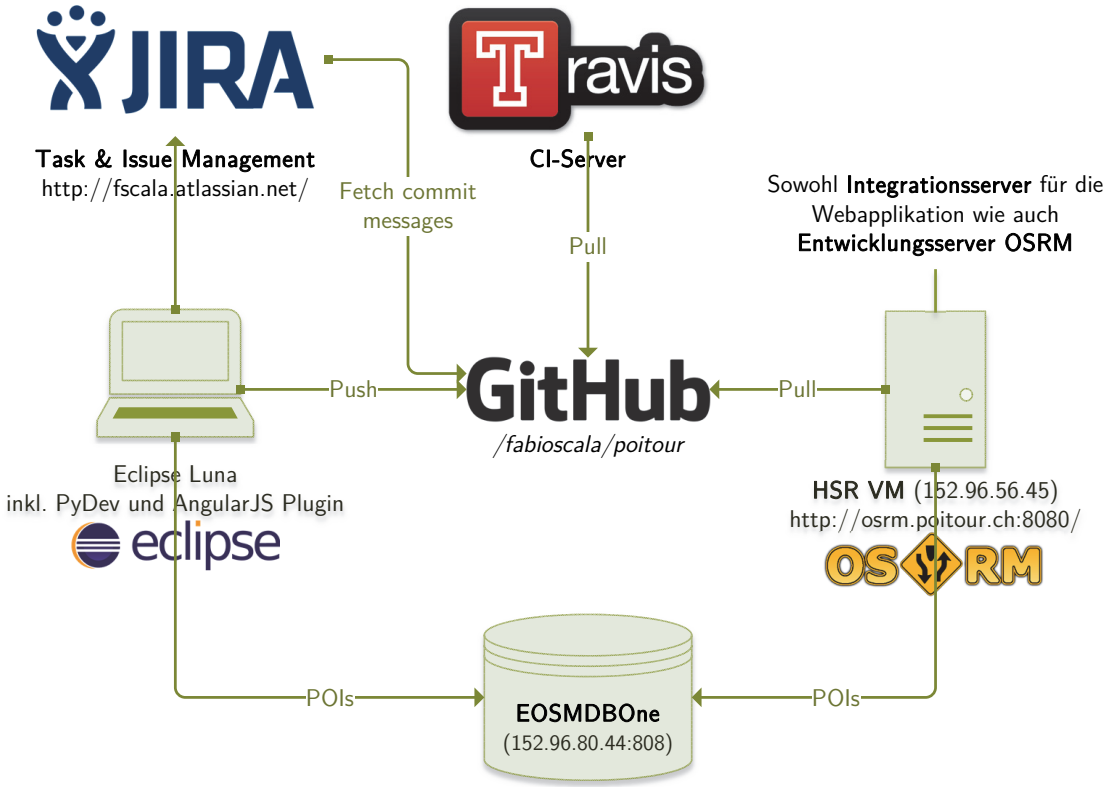


Abbildung 4.1.: Entwicklungsumgebung

4.1. IDE

Als IDE wurde Eclipse Luna mit zusätzlichen Plugins für Python[1] und AngularJS verwendet.

4.2. SCM

Als Source Control Management (SCM) wurden private GitHub Repositories[2] verwendet. Frontend und Backend wurden dabei in separaten Submodulen gepflegt.

[1] PyDev
[2] Ist nun nach Umbenennung und erstem Release im öffentlichen Repository `fabioscala/poitour` verfügbar

4.3. Projektmanagement

Der kostenpflichtige OnDemand Service von Atlassian Jira inklusive Agile Plugin wurde zur Verwaltung und Planung der Epics, User Stories und Tasks verwendet. Jira ermöglicht via intuitivem User Interface eine flexible Planung der Sprints sowie Überwachung mittels Burndown Charts und Reporting der Aufwände wie in den Abb. 4.2 und 4.3 gezeigt.

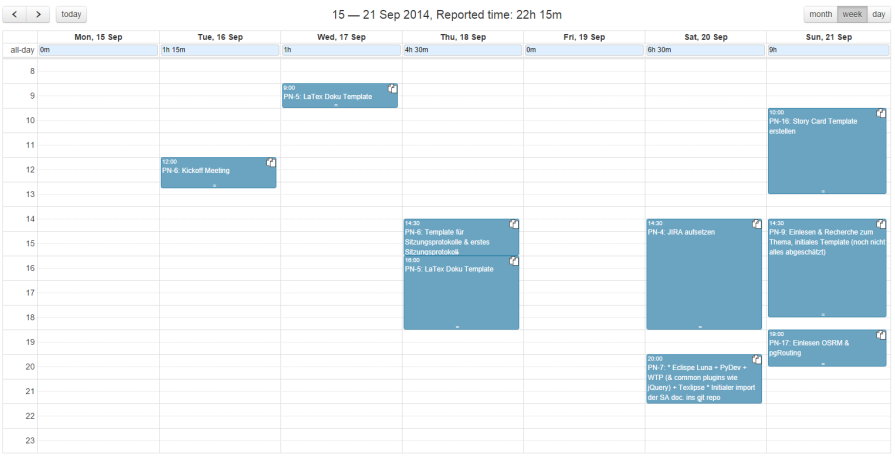
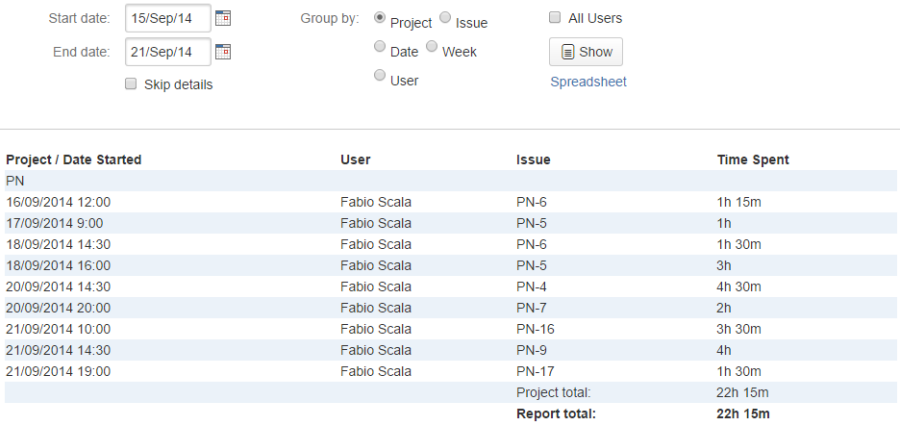


Abbildung 4.2.: Buchung der Aufwände in Jira



Project / Date Started	User	Issue	Time Spent
PN			
16/09/2014 12:00	Fabio Scala	PN-6	1h 15m
17/09/2014 9:00	Fabio Scala	PN-5	1h
18/09/2014 14:30	Fabio Scala	PN-6	1h 30m
18/09/2014 16:00	Fabio Scala	PN-5	3h
20/09/2014 14:30	Fabio Scala	PN-4	4h 30m
20/09/2014 20:00	Fabio Scala	PN-7	2h
21/09/2014 10:00	Fabio Scala	PN-16	3h 30m
21/09/2014 14:30	Fabio Scala	PN-9	4h
21/09/2014 19:00	Fabio Scala	PN-17	1h 30m
		Project total:	22h 15m
		Report total:	**22h 15m**

Abbildung 4.3.: Reporting der Aufwände in Jira

4.4. HSR VM

Da das Kompilieren von OSRM unter Windows gegenüber Linux mit erheblichem Mehraufwand verbunden ist und die Verwendung einer lokalen virtuellen Maschine viel Ressourcen[3] beansprucht, wurde die OSRM Instanz der Integrationsumgebung[4] zugleich für die Entwicklung benutzt. Dies kommt mit dem Nachteil, dass Anpassungen an der OSRM Konfiguration, welche mit derjenigen der Applikation abgestimmt sein müssen, nur zeitgleich mit einem Integrations-Release geschehen dürfen.

[3] Arbeitsspeicher, Rechenkapazität, ...
[4] Demo-Server

5. Qualitätsmanagement

5.1. Tests

Um eine angemessene Qualität der Software zu gewährleisten wird auf automatisiertes Testing gesetzt. Diese Tests werden von einem externen CI Anbieter namens Travis-CI[1] bei jedem Push von Quellcode in das GitHub Repository ausgeführt und bei Änderungen des Status die Entwickler via E-Mail wie in Abb. 5.1 gezeigt, benachrichtigt. Die dazu eingesetzten Testing-Frameworks sind im entsprechenden Abschnitt 4.7 auf Seite 69 in Teil III beschrieben.

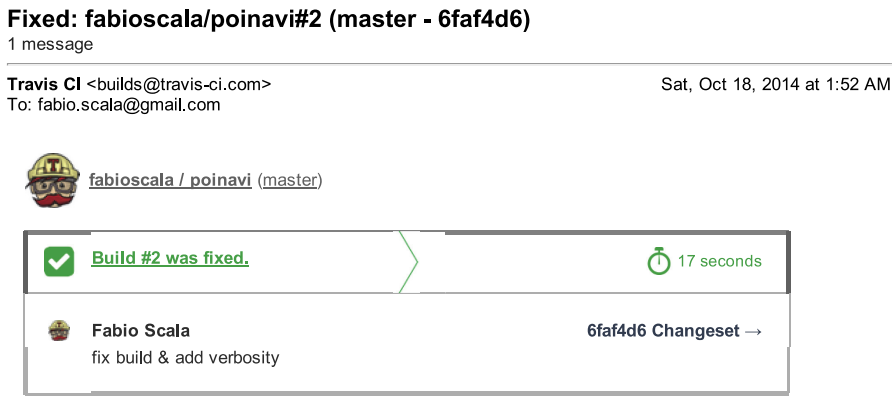


Abbildung 5.1.: Travis-CI E-Mail Benachrichtigung

5.2. Coding-Richtlinien

Um eine einheitliche Formatierung und Stil des Quellcodes zu erlangen wurden für alle primären Programmiersprachen des Projekts (Python, JavaScript) Coding Style Guidelines festgelegt.

5.2.1. Python

Es wurden die in der Python Community weit verbreiteten Python Enhancement Proposal 8 (PEP 8) Richtlinien angewendet mit einziger Ausnahme der Zeilenlänge, die bei 160 statt nur 80 Zeichen angesetzt wurde. Die PEP 8 Regeln werden durch die Entwicklungsumgebung sowie Flake8[2] erzwungen. Flake8 wird zudem auch im Travis-CI Build aufgerufen und generiert entsprechenden Warnungen bei nicht Einhalten. Ein Beispiel-Output einer solchen Prüfung sieht folgendermassen aus:

[1] Siehe auch: https://travis-ci.org
[2] Siehe auch: http://flake8.readthedocs.org

```
[1;33m[WARN] [0;0m flake8: app\main\controller.py:45:16: E703 statement ends with a semicolon
[1;33m[WARN] [0;0m flake8: app\main\controller.py:45:17: W292 no newline at end of file
[1;33m[WARN] [0;0m flake8: app\main\__init__.py:5:25: W292 no newline at end of file
[1;33m[WARN] [0;0m flake8: app\structures\route.py:72:56: W291 trailing whitespace
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:9:1: W293 blank line contains whitespace
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:18:1: W293 blank line contains whitespace
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:24:1: E303 too many blank lines (3)
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:25:52: E231 missing whitespace after ','
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:25:63: E231 missing whitespace after ','
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:27:1: W293 blank line contains whitespace
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:29:5: E265 block comment should start with
↪  '# '
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:29:5: E303 too many blank lines (2)
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:30:1: W293 blank line contains whitespace
[1;33m[WARN] [0;0m flake8: app\utils\eosmdbone.py:30:1: W391 blank line at end of file
------------------------------------------------------------
[1;31mBUILD FAILED - flake8 found 43 warning(s)
```

5.2.2. JavaScript

Bei JavaScript wurde an die Richtlinien von Google[3] angelehnt mit einigen Ausnahmen und Erweiterungen bezüglich Styling. Diese wurden grundsätzlich durch ein speziell konfiguriertes Profil im automatischen Formatierer der Entwicklungsumgebung durchgesetzt und sind im Projektverzeichnis unter `frontend/ide/eclipse/javascript_formatter.xml` abgelegt. Auch hier wurden Best Practices mittels JSHint[4] durch den CI Server überprüft:

```
app/scripts/controllers/map.js
line 94   col 6    Missing semicolon.
line 177  col 49   Expected '===' and instead saw '=='.
line 307  col 41   'popupMarkup' is already defined.
line 310  col 38   'popupMarkup' used out of scope.
line 322  col 25   ['photos'] is better written in dot notation.
line 324  col 32   ['photos'] is better written in dot notation.
```

[3] Siehe auch: http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml
[4] Siehe auch: http://jshint.com/

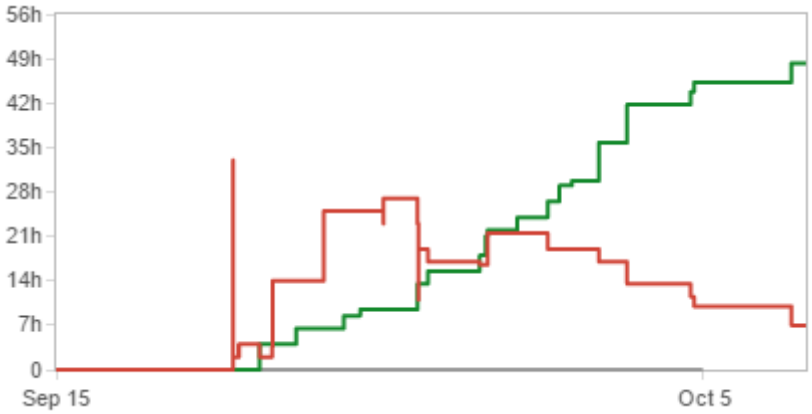
6. Sprints

6.1. Sprint 1

6.1.1. Summary

Sprint 1 *(Analog RUP Inception/Elaboration)*	
Periode	15.09.2014–05.10.2014
Stunden Soll	51.4 h
Stunden Plan	–
Stunden Ist	75.5 h

Im ersten Sprint, welches bereits bei der Projektaufgleisung begonnen hat, war der "Scope Change" aufgrund der zu Beginn noch nicht aufgegleisten/existierenden Tasks sehr hoch.



Burndown Chart Sprint 1

6.1.2. Ziele

- Verstehen der Aufgabenstellung
- Projekt Initialisierung (SCM, Dokumentation, . . .)
- Aufgleisung der Projekts
- Erste Anforderungen sammeln (Requirements Engineering)
- Erste Risikoanalyse

6.1.3. Abgeschlossen

Folgende High-level (ohne Subtasks) Tasks wurden während des ersten Sprints erstellt und sogleich abgeschlossen.

JIRA-Key	Summary
PN-1	Projektinfrastruktur einrichten
PN-6	Projektsitzungen Sprint 1
PN-9	Initiale Risikoanalyse
PN-14	Evaluation Routing Engine
PN-16	Story Cards erstellen und drucken
PN-26	Organisatorische Aufwände Sprint 1
PN-32	Projektdokumentation Sprint 1

6.1.4. Probleme

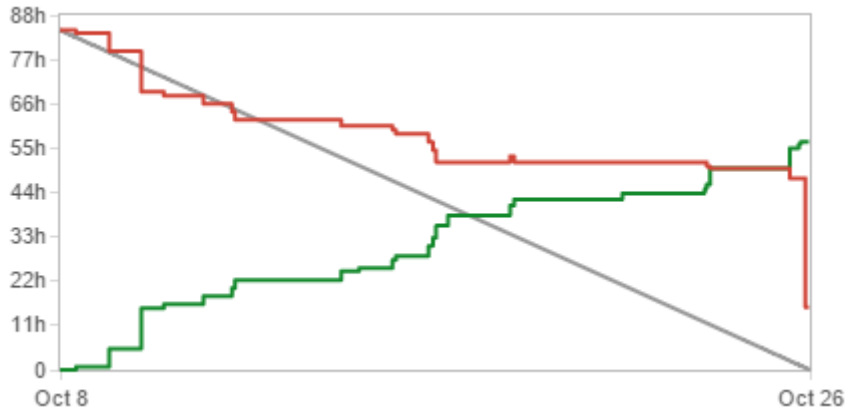
Es beschäftigt die Frage nach einem effizienten Algorithmus um das "Kort TSP" Problem zu lösen.

6.2. Sprint 2

6.2.1. Summary

Sprint 2 *(Analog RUP Elaboration/Construction)*	
Periode	06.09.2014–26.10.2014
Stunden Soll	51.4 h
Stunden Plan	78 h
Stunden Ist	71.75 h

Im zweiten Sprint waren laut Burndown Chart etwa 85 h fällig. Hierbei hat sich jedoch ein Fehler bei einem Task eingeschlichen, wo zunächst das Parent-Issue und danach auch noch die einzelnen Sub-Tasks geschätzt wurden und in der Grafik summiert visualisiert werden. Dieser Fehler sieht man auch gegen Ende, als das Issue geschlossen wurde und die verbleibende Zeit plötzlich absinkt.
Der Grund weshalb die grüne Kurve (gebuchte Zeit) gegen Ende steigt, jedoch die verbleibende Zeit des Sprints (rot) gleich bleibt, ist weil in dieser Zeit bereits an Issues des kommenden Sprints gearbeitet wurde, ohne den Scope des jetzigen verändern zu wollen.



Burndown Chart Sprint 2

6.2.2. Ziele

- Einrichten der Entwicklungs- und Integrationsumgebung inkl. CI
- Requirements Engineering Teil 2
- Prototypen eines Routing Algorithmus der das Problem zufriedenstellen löst

6.2.3. Abgeschlossen

Folgende High-level (ohne Subtasks) Tasks wurden im zweiten Sprint abgeschlossen.

JIRA-Key	Summary
PN-12	Mini-Evaluation Python Framework
PN-13	Mini-Evaluation Frontend Framework
PN-20	Projektsitzungen Sprint 2
PN-27	Organisatorische Aufwände Sprint 2
PN-33	Projektdokumentation Sprint 2
PN-39	Entwicklungsumgebung aufsetzen (Webserver inkl. Frameworks/Boilerplates)
PN-40	Integrationsumgebung aufsetzen (HSR Server mit CI)
PN-41	Prototypen für "Kort TSP"

6.2.4. Probleme

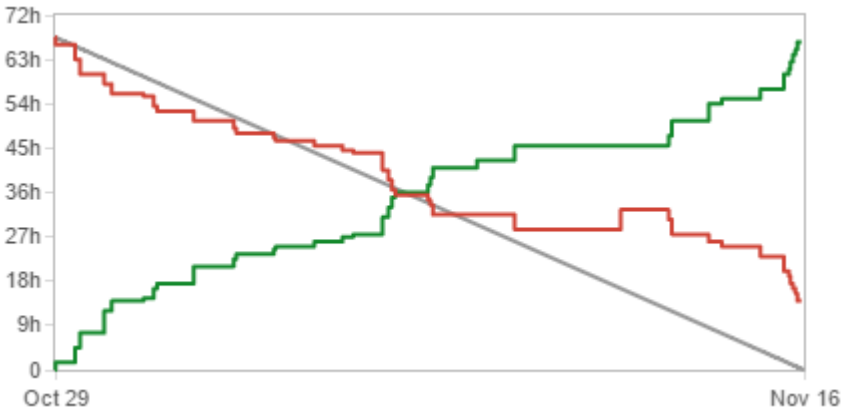
- Es wurde ein Prototyp basierend auf einem genetischen Algorithmus für das STSP implementiert, doch die benötigte Zeit für ein gutes Resultat bei einer langen Tour ist nicht mehr akzeptabel.
- OSRM unterstützt zwar neuerdings Distanztabellen bis 100 Punkte (kann im C++ Quellcode auch gepatcht werden, ein Issue für eine Parametrisierbarkeit existiert bereits) doch diese gibt nur Distanzen in Sekunden zurück. Das Routing-Profil *foot-city.lua* von routing.osm.ch missbraucht jedoch die Geschwindigkeitseinstellungen für um Präferenzen zu erzwingen.

6.3. Sprint 3

6.3.1. Summary

Sprint 3 *(Analog RUP Construction)*	
Periode	27.10.2014–16.11.2014
Stunden Soll	51.4 h
Stunden Plan	67.5 h
Stunden Ist	66.5 h

Die geplante sowie gearbeitete Zeit stimmen zwar überein, doch verbunden mit der verbleibenden Zeit im Burndown Chart sagt dies aus, dass einige Tasks länger gedauert, andere dafür weniger Zeit beansprucht haben oder nicht fertig geworden sind. In der Tat sind Dokumentation und Unit Tests nicht ganz auf dem Stand der abgeschlossenen User Stories. Diese fliessen somit in den nächsten Sprint als separate Tasks ein.



Burndown Chart Sprint 3

6.3.2. Ziele

- Stabiles erstes Release, das heisst:
  - Umsetzen aller Sprint 3 User Stories
  - Keine offensichtlichen Fehler
  - Wenn ein Fehler auftritt eine Benutzerfreundliche Nachricht anzeigen
  - Unit Tests
  - Inline Dokumentation
  - Deployment auf Demo Server
- Dokumentation

6.3.3. Abgeschlossen

Folgende High-level (ohne Subtasks) Tasks wurden im dritten Sprint abgeschlossen.

JIRA-Key	Summary
PN-11	Neubewertung Risiken nach 8 Wochen
PN-21	Projektsitzungen Sprint 3
PN-28	Organisatorische Aufwände Sprint 3
PN-34	Projektdokumentation Sprint 3
PN-44	Prototyp GUI
PN-45	Abfrage mittels eigenen Punkten
PN-51	Start- und Endpunkt sofort visualisieren
PN-52	Kategorie auswählen
PN-53	Sofortige Validierung der Eingaben
PN-54	Abfrage mittels Kategorien
PN-55	Route berechnen
PN-56	Verweildauer bestimmen

6.3.4. Probleme

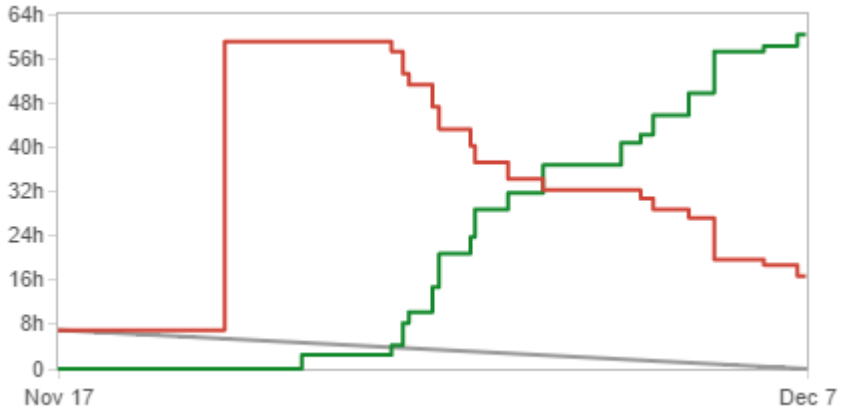
Nach wie vor die Distanztabellenfunktion von OSRM in Kombination mit dem Schweizer `foot-city.lua` Profil. Als vorübergehende Lösung wurden die Geschwindigkeiten des Profils auf konstante 3.6 km/s gesetzt, was in bisherigen Tests zu keinen offensichtlichen Routing-Problemen geführt hat.

6.4. Sprint 4

6.4.1. Summary

Sprint 4 *(Analog RUP Construction)*	
Periode	17.11.2014–07.12.2014
Stunden Soll	51.4 h
Stunden Plan	61 h
Stunden Ist	72.5 h

Es wurde neu ein Task "Andere Aufwände (Bugfixing, Feedback)" mit eingeplant. Doch dieser hat auch gleich 5x soviel Zeit beansprucht wie geschätzt. Glücklicherweise wurden diese Aufwände durch andere, mit zu vielen Stunden geschätzte Tasks, kompensiert, sodass lediglich etwa 10 "Überstunden" geleistet werden mussten. Der "Scope Change" im Burndown Chart erklärt sich dadurch, dass die Inhalte des Sprints erst bei der Projektsitzung Mitte Woche festgelegt wurden.



Burndown Chart Sprint 4

6.4.2. Ziele

- Feature Freeze
  - Anbindung an Kort
  - Gewichtung der POIs implementieren
  - Feedback des "Kunden" (Prof. Stefan Keller) sowie dritter (Mitstunden, Bekannte, Mitarbeiter, ...) einfliessen lassen.
  - Refactoring einiger AngularJS Module

6.4.3. Abgeschlossen

Folgende High-level (ohne Subtasks) Tasks wurden im vierten Sprint abgeschlossen.

JIRA-Key	Summary
PN-22	Projektsitzungen Sprint 4
PN-29	Organisatorische Aufwände Sprint 4
PN-35	Projektdokumentation Sprint 4
PN-57	Fotos anzeigen
PN-71	Kort Provider
PN-73	Gewichtung POIs
PN-81	Andere Aufwände (Bugfixing, Feedback, etc.)

6.4.4. Probleme

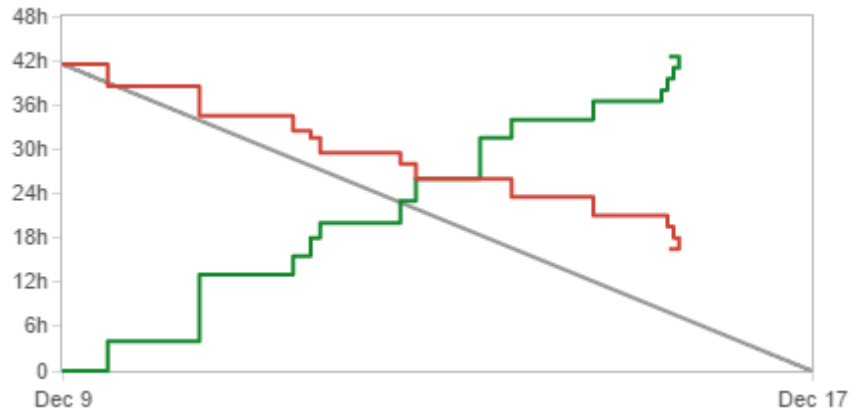
Die Performance der Abfragen auf die EOSMDBOne `osm_poi` View wurde nach der Schachtelung der Kategorien zum Flaschenhals. Dies konnte durch Bündelung der Abfragen zur einer einzigen CTE sowie Verzicht auf die Verwendung von `osm_poi` gelöst werden.

6.5. Sprint 5

6.5.1. Summary

Sprint 5 *(Analog RUP Transition)*	
Periode	08.12.2014–17.12.2014
Stunden Soll	34.3 h
Stunden Plan	39 h
Stunden Ist	42 h

Im letzten Sprint ging es nur noch darum offensichtliche Bugs zu beseitigen und den abzugebenden Bericht zu vervollständigen.



Burndown Chart Sprint 5 kurz vor Ende

6.5.2. Ziele

- Letzte Bugfixes
- Bestehende Dokumentationsteile fertigstellen
- Ausstehende Dokumentationsteile erstellen
  - Teil I – Technischer Bericht
  - Teil V – Softwaredokumentation
  - Management Summary
  - Abstract
  - A0 Poster erstellen
  - CDs erstellen
- Programmcode auf GitHub veröffentlichen

6.5.3. Abgeschlossen

Folgende High-level (ohne Subtasks) Tasks wurden im letzten Sprint abgeschlossen.

JIRA-Key	Summary
PN-23	Projektsitzungen Sprint 5
PN-30	Organisatorische Aufwände Sprint 5
PN-36	Projektdokumentation Sprint 5
PN-50	Dokumentation finalisieren
PN-92	Bugfixing
PN-93	Andere Aufwände

6.5.4. Probleme

Die Zeit für den Abschluss des Projekts inkl. der Dokumentation ist aufgrund diverser anderen Abgaben und der etwas verfrühten Abgabe doch etwas knapp geworden. Nächstes Mal sollten ganze zwei Wochen, also wirklich 14 d eingeplant werden.

6.6. Total und Fazit

Total	
Periode	15.09.2014–19.12.2014
Stunden Soll	240 h
Stunden Ist	328 h

Der erhöhte Gesamtaufwand von 328 h lässt wie folgt erklären:

- Experimentelles Thema
- Keine Erfahrung bezüglich den eingesetzten Technologien. Dieser Aufwand wurde von F. Scala gerne getragen.
- Keine Erfahrung mit LaTeX. Auch dieser Aufwand wurde von F. Scala in Kauf genommen.
- Lange Sitzungen
- Meetings und Kommunikation mit Drittparteien

Teil V.

Softwaredokumentation

1. Installation

Dieses Kapitel erläutert wie die einzelnen Komponenten von POI Tour zu installieren und konfigurieren sind, sodass die gesamte Applikation lauffähig wird.
Alle nötigen Komponenten sind im zentralen GitHub Repository `fabioscala/poitour` als Submodule enthalten. Es empfiehlt sich im Rahmen dieser Installationsanleitung den `hsr-sa` Tag via Git zu Klonen oder diesen alternativ aus der beigefügten CD unter `/3_SRC/poitour` zu beziehen.

```
$ git clone --recursive -b hsr-sa https://github.com/fabioscala/poitour.git
```

Hinter dem Proxy der HSR empfiehlt es zudem alle SSH URLs durch Git mit HTTP ersetzen zu lassen, das es sonst zu fehlern kommen kann.

```
$ git config --global url."https://github.com/".insteadOf git@github.com:
$ git config --global url."https://".insteadOf git://
```

Findet die Installation auf einem Ubuntu Server[1] mit mindestens 4096 MB statt, so kann folgendes Installationsskript verwendet werden um alle Komponenten einzurichten:

```
user@ubuntu:~/poitour$ sudo ./install.sh
```

Dabei werden lediglich die einzelnen in Abschnitte 1.1 bis 1.3 genannten Installationsskripte aufgerufen.

1.1. OSRM

Die Installation von OSRM ist bereits in Teil II, Abschnitt 2.1.1 auf Seite 15 beschrieben und wird hier nicht wiederholt. Zur Kompatibilität sollte jedoch der im POI Tour Repository vorhandene Fork verwendet werden. Zudem sollte das angepasste Profil `poitour/osrm/profiles/foot-city.lua` und nicht das originale von routing.osm.ch verwendet werden.
Als Alternative kann das Installationsskript `poitour/osrm/install.sh` verwendet werden. Bei Fehlern oder Problemen kann der kommentierte Inhalt des Skripts oder die obengenannte Installationsanleitung konsultiert werden.

1.2. Frontend

Eine bereits kompilierte[2] Version des Frontends mit allen Abhängigkeiten ist auf der CD unter `/4_DIST/frontend` abgelegt. Falls man diesen Build-Prozess selbst durchführen möchte, kann auch hier das entsprechende `poitour/frontend/install.sh` verwendet werden. Weitere Details zu diesem Prozess wie auch zur Entwicklung sind in Kapitel 2 beschrieben.

[1] Getestet auf Ubuntu Server 14.04.1 LTS 64-bit
[2] JavaScript und HTML Dateien konkateniert und komprimiert

1.3. Backend

Auch die Serverseite lässt sich reibungslos mit dem Installationsskript `poitour/backend/install.sh` installieren. Wobei das Skript automatisch eine Konfigurationsdatei anhand von `poitour/backend/sample_config.ini` erstellt und den Benutzer nach den Details und Credentials der Datenbank fragt:

```
user@ubuntu:~/poitour/backend$ sudo ./install.sh
Enter EOSDMDBOne host and port (e.g. localhost:5432): 127.0.0.1:5432
Enter EOSDMDBOne database name: gis_db
Enter EOSDMDBOne database username: fscala
Enter fscala's password: TheAnswerIs42
Do you want to start the backend in dev mode on 0.0.0.0:5000 now? [Y/n] y * Running on
 ↪  http://0.0.0.0:5000/
* Restarting with reloader
```

1.3.1. WSGI

Um die Applikation mittels WSGI auf einem Apache Webserver zu installieren kann die WSGI-Schnitstelle in `poitour/bachend/app.wsgi` mit folgendem Minimalbeispiel eine Apache Konfiguration verwendet werden:

```
<VirtualHost *:80>
  ServerName beta.poitour.ch
  WSGIScriptAlias / /var/www/poitour/backend/app.wsgi
  <Directory /var/www/poitour/backend/app/>
    WSGIProcessGroup app
    WSGIApplicationGroup %{GLOBAL}
    Order deny,allow
    Allow from all
  </Directory>
</VirtualHost>
```

1.3.2. Konfiguration

Die obengenannte Konfigurationsdatei enthält alle für sowohl Backend wie auch Frontend relevanten Konfigurationsoptionen. Diese werden in den folgenden Abschnitten erläutert.

Sektion `[GENERAL]`

`CONFIG_TYPE`	Welche Konfigurationsklasse aus `poitour/backend/config.py` verwendet werden soll. Mögliche Werte sind `development`, `testing`, `integration` und `production` wobei die letzteren zwei ein komplett kompiliertes Frontend benötigen.
`OSM_BASE_URL`	Die URL für OpenStreetMap. Wird für die Links zu den OSM Nodes und Ways verwendet.

Sektion `[ROUTING]`

WALKING_SPEED_KM_H	Die Geschwindigkeit in km/h (Fussgänger) welche für die Berechnungen und Abfragen verwendet werden soll. Sollte mit der Geschwindigkeit des OSRM Profils übereinstimmen.

Sektion `[CLIENT]`

Alle in dieser Sektion enthaltenen Daten (auch eigene, hier nicht genannte) werden an das Frontend übermittelt und stehen dort über den AngularJS Service `appConfig` zur Verfügung.

HOURS	Die standardmässig hinterlegte Stundenzahl bei der Gesamtdauer im User Interface.
MINUTES	Die standardmässig hinterlegte Minutenzahl bei der Gesamtdauer im User Interface.
START	Der standardmässig hinterlegte Startpunkt im User Interface.
END	Der standardmässig hinterlegte Endpunkt (Ziel) im User Interface
STAY_TIME	Die standardmässig hinterlegte Aufenthaltszeit im User Interface
CATEGORIES	Die standardmässig hinterlegten Interessen im User Interface als kommaseparierte Liste von Kategorie IDs und Gewichten (z.B. `xmas:5,tourism:5`).
DEFAULT_WEIGHT	Das Gewicht welches bei durch den Benutzter hinzugefügten Interessen verwendet werden soll.
LATLNG_DECIMAL_PRECISION	Mit welcher Präzision die Koordinaten im User Interface dargestellt werden soll. Dies findet beispielsweise Verwendung bei der Auswahl von Start- und Ziel mittels dem Kontextmenu.
API_URL	Die relative API URL zur Berechnung der Tour.
NOMINATIM_URL	Die URL für den Nominatim Geocoding Service. Dabei dient `{{query}}` als Platzhalter für die den eingegebenen Suchstring.
FLICKR_API_KEY	Der API Schlüssel für die Flickr Foto-API.
ATTRIBUTION	Einen beliebigen HTML String für die Leaflet Attribution, welche unten rechts auf der Karte erscheint.

Sektion `[EOSMDBOne]`

DB_NAME	Name der EOSMDBOne Postgres Datenbank.
DB_USER	Name des Postgres Benutzers der für die Abfragen verwendet wird.
DB_PASSWORD	Password des Postgres Benutzer in `DB_USER`.
DB_HOST	Hostname und Port als `host:port` des Postgres Datenbankservers der die EOSMDBOne enthält.

Sektion `[OSRM]`

BASE_URL	Die URL des OSRM Servers.
CORRECTION_FACTOR	Korrekturfaktor für die von OSRM zurückgegebenen Zeiten in der Distanzmatrix. Diese werden mit diesem Faktor multipliziert. Nützlich falls das OSRM Profil nicht mit derselben Geschwindigkeit wie WALKING_SPEED_KM_H aus der Sektion [GENERAL] konfiguriert ist.

Sektion [GA]

Die Konfigurationsoptionen in dieser Sektion beziehen sich auf den in der Applikation verwendeten Genetischen Algorithmus und haben direkten Einfluss auf Laufzeit und Qualität der Touren. Die Einstellungen aus der Musterkonfiguration haben sich für Touren bis zu 7 Stunden einer POI-reichen Umgebung wie Zürich bewährt.

POPULATION_SIZE	Die Gesamtgrösse der Population (d.h. Lösungen) mit denen der Algorithmus laufen soll. Je höher die Zahl desto mehr unterschiedliche Touren entstehen pro Generation. Eine höhere Anzahl kann zu besseren Resultaten jedoch auch zu einer erheblich längeren Laufzeit führen.
TOURNAMENT_SIZE	Die Anzahl Lösungen die pro Generation während der Selektion miteinander verglichen werden sollen. Eine höhere Zahl lässt den Algorithmus schneller konvergieren (kürzere Laufzeit) jedoch zu potenziell schlechteren Lösungen, da die Wahrscheinlichkeit in lokalen Minimas stecken zu bleiben steigt.
MIN_GENERATIONS	Die minimale Anzahl zu durchlaufender Iterationen bevor irgendwelche Abbruchkriterien geprüft werden.
MAX_GENERATIONS	Nach dieser Anzahl Iterationen wird der terminiert Algorithmus unabhängig von der Qualität der Lösungen.
TERMINATION_THRESHOLD	Prozentuale Verbesserung die in den letzten 5 Iterationen mindestens stattgefunden haben muss um weiterzulaufen. Ein Wert von 0.01 entspricht $1\,\%$.
MAX_RUNTIME_MS	Die maximale Laufzeit des Algorithmus in Millisekunden. Nach dieser Zeit terminiert dieser unabhängig von der Qualität der Lösungen.

2. Entwicklung

2.1. Frontend

Das AngularJS Frontend wurde via Scaffolding mittels Yeoman[1] und dessen AngularJS Generators[2] erstellt. Dieses sollte wenn möglich auch weiterhin verwendet werden um die Angular Best-Practices einzuhalten und Tipparbeit zu ersparen. Die nötige Entwicklungsumgebung[3] ist bei erfolgreichem Ausführen von `poitour/frontend/install.sh` ebenfalls installiert.

2.1.1. Hinzufügen von Drittkomponenten

Das Hinzufügen von weiteren Komponenten und Libraries geschieht mittels Bower und Grunt:

```
$ bower install  angular-loading-bar --save # Installieren und im bower.json referenzieren
$ grunt wiredep # Fügt die nötigen Dependencies automatisch dem index.html hinzu
```

2.1.2. Testing

Die Unit Tests sind so konfiguriert, dass diese in Firefox (aufgrund Travis-CI) ausgeführt werden. Dies kann unter `poitour/frontend/test/karma.conf.js` umgestellt werden.

```
$ grunt test # Unit Tests ausführen
```

Die Protractor (End-to-End GUI Testing) werden hingegen in Chrome ausgeführt. Dies kann unter `poitour/frontend/test/protractor.conf.js` umgestellt oder zur gleichzeitigen Ausführung auf mehreren Browsern konfiguriert werden.

```
$ grunt test:e2e # Unit Tests ausführen
```

2.1.3. Build / Deployment

In einer produktiven Umgebung will man alle JavaScript, HTML und CSS Dateien konkatenieren und komprimieren um die Ladezeiten und den Traffic gering zu halten. Dies geschieht ebenfalls mittels Grunt Task:

```
$ grunt build
```

[1] http://yeoman.io/
[2] https://github.com/yeoman/generator-angular
[3] node.js, npm, Yeoman, Grunt und Bower

2.2. Backend

Die nötigen Python Packages sind alle in den Dateien `poitour/backend/requirements.txt` und `poitour/backend/requirements_dev.txt` aufgelistet und lassen sich via PIP installieren:

```
user@ubuntu:~/poitour/backend$ pip install -r requirements.txt
user@ubuntu:~/poitour/backend$ pip install -r requirements_dev.txt
```

Für diejenigen Packages die sich nicht ohne Mehraufwand kompilieren lassen können diese via apt (z.B. `$ apt-get install python-scipy`) oder auf Windows von Christoph Gohlkes Webseite[4]

[4] http://www.lfd.uci.edu/~gohlke/pythonlibs/

A. Inhalt der CD

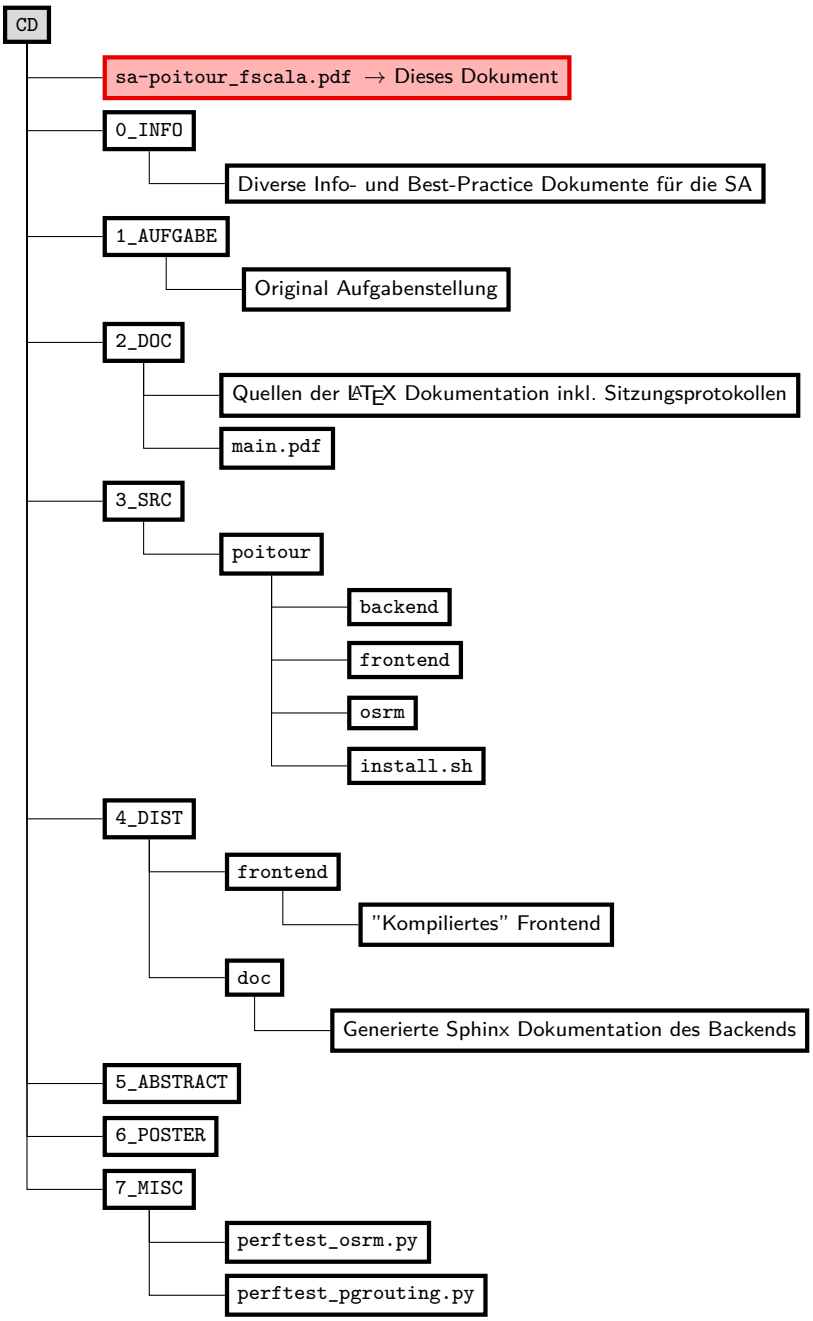


Abbildung A.1.: Inhalte der beigefügten CD

B. Mailverkehr mit ZüriOberland Tourismus

Fragen bezüglich Fussgänger-Routenplaner

Daniela Waser <daniela.waser@zuerioberland.ch> Tue, Oct 21, 2014 at 9:52 AM
To: Fabio Scala <fabio.scala@gmail.com>

Grüezi Herr Scala

Herzlichen Dank für Ihre Ausführungen. Nachfolgend finden Sie meine Einschätzungen dazu:

Wie sinnvoll fänden Sie eine solche Applikation? Besteht überhaupt ein Bedarf?

Soweit ich weiss existiert bis anhin keine vergleichbare App. Ein Versuch ist es also durchaus wert! Ich könnte mir gut vorstellen, dass es insbesondere in den Städten gut ankommen könnte, da dort die Sehenswürdigkeiten auf rel kleinem Raum eng zusammen liegen. Für die ganze Region wird es eher schwierig. Aber wie gesagt – wer nicht wagt, der nicht gewinnt!

Nach welchen individuellen Kriterien würde eine solche Routenplanung Sinn machen?

(z.B. POI Kategorien in bestimmter Zeit abklappern, evtl. noch eigene Routenpunkte?)

- POI: das heisst Leistungsträger von Zürioberland Tourismus, Sehenswürdigkeiten (zB historische Gebäude, Aussichtspunkte), Feuerstellen, …
- Verfügbare Zeit
- Für mich ist noch wichtig, dass nicht nur der schnellste Weg gewählt wird, sondern dass dieser für den Gast auch attraktiv ist. Also dass er zB auf Wanderwegen gehen kann, statt einfach nur der Strasse entlang zu laufen. Vielleicht könnte man das noch durch Sterne-Kategorien (im Sinne von „Genuss-Sterne“ zur Auswahl geben? Also wenn der Gast bei der Auswahl 1 Stern wählt will er möglichst schnell von A nach B. Wenn er jedoch 5 Sterne wählt, dann möchte er nicht primär schnell am Ziel angelangen, sondern eine schöne Route gehen.
- Auch wenn ein Punkt nicht vom Gast ausgewählt wurde, sollte ihm dieser angezeigt werden, wenn er ohnehin daran vorbeikommt. Ein Beispiel: Wandert er von C nach D und kommt unterwegs zufällig an einem schönen Wasserfall vorbei (ohne dies ausgewählt zu haben), dann erscheint der trotzdem auf der App.
- Auch öV Haltestellen, PP, Mobility PP etc müssten gekennzeichnet sein.

User Interface

- Wichtig sind Bilder! Die müssen zwingend eingebunden werden, dass sich der Gast vorab entscheiden kann, ob er das sehen will oder nicht. Allenfalls könnte man hier gleich die Google Panoramino Bilder einbinden?
- Möglichst einfach! Den Regler „Geringes/grosses Interesse“ würde ich weglassen. Oder für was braucht’s den? Der Gast hat ja schon gesagt, was ihn interessiert.
- Fahrplan öV einbinden: es muss einfach möglich sein, sich die Informationen zur An-/Abreise zu beschaffen. zB durch die Einbindung durch die ÖV Haltestellen in die Karte. Allenfalls will der Gast ja auch mit unterschiedlichen Verkehrsmitteln (zu Fuss und ÖV)
- Informationen zur Wegbeschaffenheit: Es könnte ja sein, dass ein Rollstuhlfahrer/Inliner die Strecke abfahren möchte: Dann ist die Wegbeschaffenheit entscheidend. Auch wenn ein Radfahrer

die App nutzen möchte müssen Informationen zu Fahrverboten/Velowegen etc vorhanden sein.

- Entweder braucht der Gast die Karte ODER die Kriterien-Auswahl. Das würde man dann über einen App-Knopf wie „Einstellungen“ regeln? Aber hier kenne ich mich zu wenig aus…

- „Mich interessieren“: kann der Gast aus einer Liste auswählen oder gibt er alle Begriffe selbst ein? Letztere Variante: Wie stellen Sie sicher, dass es seinen POI dann auch tatsächlich findet (Schreibfehler usw)

- In welchen Sprachen gibt’s die App?

Haben Sie noch weitere Verbesserungsvorschläge?

- Meine wichtigsten Überlegungen habe ich oben ausgeführt. Das ist für den Moment alles.

- Vielleicht könnten Sie sich noch bei Herrn Schad von der Hochschlue Luzern – Wirtschaft melden. Er kennt sich in der Thematik Mobilität sehr gut aus und unterrichtet dies auch im Studiengang „Tourismus und Mobilität“.

- Eventuell wäre es noch sinnvoll, das User Interface mit einem Profi anzuschauen.

Inwiefern nutzen Ihre Kunden das "mobile" Angebot bzw. würde man eine solche Route Zuhause am PC planen oder doch eher "ad-hoc" mit dem Smartphone

- .Hierzu liegen uns leider keine genauen Zahlen vor.

- Wir stellen jedoch fest, dass sich die Gäste zu Hause (rel kurzfristig) informieren, was sie unternehmen möchten. Dann gehen sie raus in die Region und besuchen die vorher ausgesuchten „Stationen“. Bei einer solchen Anwendung wäre es aber zwingend, dass die Route mobil verfügbar ist. Die Gäste kennen die generierte Route ja nicht und müssen diese entsprechend vor sich haben, um sie ablaufen zu können.

- Ich bezweifle jedoch, dass der Gast in die Region fährt (ohne jegliche Vorabklärungen) und dann sein Programm ad-hoc zusammenstellt. So flexibel sind die dann auch nicht…☺

Ich hoffe, Ihnen mit meinen Ausführungen weitergeholfen zu haben. Falls ich mich unklar ausgedrückt habe oder weitere Fragen auftauchen, dürfen Sie sich gerne wieder bei mir melden.

Ich wünsche gutes Gelingen bei Ihrer Studienarbeit!

Freundliche Grüsse

Daniela Waser

Marketing Tourismus

C. Eigenständigkeitserklärung

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.
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:

Rapperswil, 16. Dezember 2014

Name, Unterschrift:

Fabio Scala

D. Sitzungsprotokolle

Die Sitzungsprotokolle wurden in dieser Ausgabe der Arbeit ausgelassen und sind nur auf der vollen, auf der CD ablegten Version, einzusehen.

Literatur

[1] Tharwon Arnuphaptrairong. *Top Ten Lists of Software Project Risks. Evidence from the Literature Survey*. English. organization. März 2011. URL: http://www.uio.no/studier/emner/matnat/ifi/INF5181/h14/pensumliste/microsoft-word---iaeng-top-ten-lists-of-software-project-risk1---imecs2011_pp732-737.pdf (besucht am 21. 09. 2014).

[2] Mike Beedle u. a. *Manifesto for Agile Software Development*. 2001. URL: http://agilemanifesto.org/ (besucht am 18. 09. 2014).

[3] OSRM Community Benutzer. *Distance table plugin, set maximum size of input before truncation from cmake*. 22. Okt. 2014. URL: https://github.com/Project-OSRM/osrm-backend/pull/1236.

[4] Barry W. Boehm. *Software Risk Management*. English. IEEE Computer Society Press, 1. Aug. 1989. 450 S. ISBN: 978-0818689062.

[5] OSRM Community. *OSRM Backend Wiki*. English. Project OSRM. URL: https://github.com/Project-OSRM/osrm-backend/wiki (besucht am 25. 09. 2014).

[6] Sarah Hoffmann. *OSRM-Talk Mailing Liste. Foot profile*. English. 29. Apr. 2014. URL: https://www.mail-archive.com/osrm-talk@openstreetmap.org/msg00455.html (besucht am 10/2014).

[7] Google Inc. *Google Trends*. Data. Google Inc. URL: http://www.google.com/trends/.

[8] Stefan Keller. Persönliche Kommunikation mit Prof. S. Keller.

[9] GISpunkt Wiki Mitwirkende. *EOSMDBOne*. URL: http://giswiki.hsr.ch/EOSMDBOne.

[10] A Piwońska. »An improved genetic algorithm for solving the Selective Travelling Salesman Problem on a road network«. In: *Zeszyty Naukowe Politechniki Białostockiej. Informatyka* (2011), S. 59–70. URL: http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BPB1-0051-0005.

[11] Sebastian Porto. *A Comparison of Angular, Backbone, CanJS and Ember*. English. 12. Apr. 2013. URL: http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/ (besucht am 03. 10. 2014).

[12] Frederik Ramm. Persönliche Kommunikation mit Prof. S. Keller. 4. Mai 2014.

[13] Carmelo Schumacher und Egli Annrita. »Kort Reloaded«. Studienarbeit. Institut für Software, 2013. URL: http://eprints.hsr.ch/296/1/SA_Kort-Reloaded_aegli_cschumac_FS2013_Dokumentation.pdf (besucht am 11/2014).

[14] *Scrum Illustration*. 2005. URL: http://www.mountaingoatsoftware.com/agile/scrum/images (besucht am 18. 09. 2014).

[15] Demeter Sztanko und Patu Tifinger. *Walks.IO – About*. 2013. URL: http://walks.io/about.html (besucht am 12/2014).

[16] Linda Wallace und Mark Keil. »Software Project Risks and Their Effect on Outcomes«. In: *Commun. ACM* 47.4 (Apr. 2004), S. 68–73. ISSN: 0001-0782. DOI: 10.1145/975817.975819. URL: http://doi.acm.org/10.1145/975817.975819.

[17] David A Wheeler. *How to evaluate open source software/free software (OSS/FS) programs*. English. Aug. 2011. URL: http://www.dwheeler.com/oss_fs_eval.html.

Glossar und Abkürzungsverzeichnis

AI Artificial Intelligence.

Boost C++ Library ist ein Sammlung freier C++ Bibliotheken für diverse Zwecke).

http://www.boost.org/

CI Continuous Integration.

CMS Content Management System.

CRUD Create-Update-Delete.

CTE Common Table Expression.

EOSMDBOne Enhanced OpenStreetMap Database One.

http://giswiki.hsr.ch/EOSMDBOne

GIS Geographic information system.

HCID Human Computer Interaction and Design.

HSR Hochschule für Technik Rapperswil.

http://hsr.ch/

I18N Internationalisierung.

IFS Institut für Software.

http://www.ifs.hsr.ch/

L10N Lokalisierung.

Lua Eine vielseitige, plattformunabhängige Skriptsprache.

http://www.lua.org/

MIT Massachusetts Institute of Technology. Die MIT Lizenz, welche aus dem MIT stammt, erlaubt die Wiederverwendung des Programmcodes sowohl für andere offene aber auch nicht offene bzw. kommerzielle Software.

http://opensource.org/licenses/MIT

MVC Model-View-Controller.

ORM Object-relational Mapping.

OSM OpenStreetMap ist ein Projekt, welches open source geografische Daten bereitstellt.

http://www.openstreetmap.org/

osm2pgrouting ist ein Tool um OpenStreeMap Kartendaten in eine pgRouting Datenbank zu importieren.

http://pgrouting.org/docs/tools/osm2pgrouting.html

osm2pgsql ist ein Tool um OSM Daten in eine PogstreSQL PostGIS Datenbank zu importieren.

http://wiki.openstreetmap.org/wiki/Osm2pgsql

Osmosis ist ein Tool um OpenStreeMap Kartendaten zu manipulieren (packen/entpacken, Ausschnitte extrahieren, ...).

https://github.com/openstreetmap/osmosis

OSRM Open Source Routing Machine eine high-performance, in C++ implementierte Routing Engine zur Ermittlung der kürzesten Route zwischen zwei geografischen Punkten.

http://project-osrm.org/

PEP 8 Python Enhancement Proposal 8 definiert die Coding Konventionen bzw. Style Guidelines für sämtlichen Code der Python Standard Library.

https://www.python.org/dev/peps/pep-0008/

pgRouting ist eine PostGIS Erweiterung für PostgreSQL welche raumbezogene Routing Funktionalität zur Verfügung stellt.

http://pgrouting.org/

POI Point of Interest (dt. Ort von Interesse).

PostGIS ist eine PostgreSQL Erweiterung welche geographische Funktionalität und Objekte zur Verfügung stellt um beispielsweise Ortsabfragen zu ermöglichen.

http://postgis.net/

PostgreSQL ist ein freies, objektrelationales Datenbankmanagementsystem.

http://www.postgresql.org/

SA Studienarbeit.

SCM Source Control Management.

Simulated Annealing ist ein porbabilistisches Optimierungsverfahren welches auf der Grundidee der "langsamen Abkühlung" beim Härten von Metallen basiert).

SPA Single Page Applications sind Webapplikationen, dessen HTML nur zu Beginn geladen wird und danach nur noch via API mit dem Server kommuniziert.

SQL Structured Query Language. Eine deskriptive Sprache zur Formulierung von Abfragen aus Datenbanken.

STSP Selective Travelling Salesman Problem.

TSP Travelling Salesman Problem.

VRP Vehicle Routing Problem.

WSGI Web Server Gateway Interface (WSGI) ist ein Python Standard welches definiert wie ein Webserver wie z.B. Apache mit der Applikation kommunizieren soll.

http://wsgi.readthedocs.org/

XML Extensible Markup Language ist eine deskriptive Sprache für hierarchische Daten.

YAGNI "You Ain't Gonna Neet It" ist ein Prinzip aus der agilen Softwareentwicklung und besagt, dass keine Funktionalität implementiert werden soll, bis diese gebraucht wird.

YAML YAML Ain't Markup Language ist eine beschreibende Sprache für Daten welche aufgrund dessen Lesbarkeit oft für Konfigurationsdaten verwendet wird.

http://www.yaml.org/