HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

senozon
understanding mobility

Bachelorarbeit, Abteilung Informatik

Verkehrsmodell-Fallstudien-Editor

Hochschule für Technik Rapperswil

Frühlingssemester 2016
17. Juni 2016

Autoren:	Fabian Keller & Dominik Heeb
Betreuer:	Prof. Dr. Luc Bläser
Projektpartner:	Dr. Marcel Rieser, Senozon AG
Arbeitsperiode:	22.02.2016 - 17.06.2016
Arbeitsumfang:	360 Stunden, 12 ECTS pro Student

Inhaltsverzeichnis

Kapitel 1

Abstract

Das Unternehmen Senozon AG ist immer wieder mit Fragestellungen wie „Was passiert, wenn diese Strasse gesperrt wird?“ oder „Was wäre, wenn diese Strasse eine andere Geschwindigkeitsbeschränkung hätte?“ konfrontiert. Um diese Fragestellungen zu beantworten, unterhält die Senozon AG ein Modell, das sowohl das Strassennetz als auch privater und öffentlicher Verkehr beinhaltet. Mit Hilfe des Open Source Simulations Framework MATSim kann dieses Modell simuliert werden.
Die Arbeit „Verkehrsmodell-Fallstudien-Editor“ behandelt die Entwicklung und Implementation einer Web Applikation, die das interaktive Bearbeiten von Daten eines solchen Verkehrsmodells innerhalb einer Karte ermöglicht. Die grosse Herausforderung dabei ist vor allem der effiziente Umgang mit der grossen Datenmenge. Dafür wird der QuadTile Algorithmus von OpenStreetMap [Ope16b] eingesetzt. Weitere Optimierungen auf Software- wie auch Datenbankseite sind ein wichtiger Teil dieser Arbeit.
Die vollendete Implementation stellt eine performante Web Applikation dar, die im Design der Senozon AG erscheint. Änderungen an Strassen können vorgenommen und in einer Datenbank persistiert werden. Dazu wurde ein Konzept entwickelt, welches ermöglicht, die Änderungen abzuspeichern ohne das Stammnetzwerk der Senozon AG anzupassen.

Kapitel 2

Einführung

2.1 Senozon AG

Diese Bachelorarbeit wird in Zusammenarbeit mit dem Unternehmen Senozon AG durchgeführt. Die Senozon AG ist ein international tätiges Technologieunternehmen auf dem Gebiet der Standortplanung und -bewertung, Verkehr- und Infrastrukturplanung sowie Mobilitätsforschung. Dafür unterhält die Senozon AG verschiedene Verkehrsmodelle, die im Wesentlichen aus einem Strassennetz, ÖV-Fahrplan und Personendaten bestehen, wobei letztere die Reiseaktivität der Personen beschreiben.

2.2 MATSim Framework

Die Senozon AG verwendet für die Simulation der verschiedenen Verkehrsmodelle das Open Source Framework MATSim.
MATSim ist ein modular aufgebautes Tool zur agentenbasierten Mobilitätssimulation, welches besonders für grosse Szenarien mit Millionen von Agenten (Personen) geeignet ist. Zudem kann sowohl privater als auch öffentlicher Verkehr (Bus, Zug, Tram usw.) simuliert werden.
Die Senozon AG stellt zusammen mit der ETH Zürich, der TU Berlin und vielen anderen Universitäten die Weiterentwicklung und Qualitätssicherung von MATSim sicher.

Kapitel 3

Problemstellung

Immer wieder kommen Kunden der Senozon AG mit Fragestellungen wie „Was passiert, wenn diese Strasse gesperrt wird?“ oder „Was wäre, wenn ein Neubaugebiet mit 5000 Einwohnern gebaut wird?“. Bislang waren diese Fragestellungen mit viel Handarbeit verbunden, um das Modell anzupassen. Dies insbesondere, weil die Simulationsdaten von MATSim XML-basiert und meist sehr gross sind.
Derzeit kann lediglich die Senozon AG selbst die Änderungen an den Verkehrsmodellen vornehmen, weil dies nur mit dem erforderlichen Know-How und der entsprechenden Erfahrung möglich ist. Das Entwickeln einer Web Applikation soll diese Funktion zukünftig auch dem Kunden auf eine einfache Art und Weise zur Verfügung stellen. Dies würde die Arbeit der Senozon AG deutlich erleichtern, weil dann die Kunden selbst oder mit Unterstützung von Senozon AG ihre Daten für die Simulation aufbereiten können. Ziel dieser Arbeit ist es, die Konzepte und wesentlichsten Risiken einer solche Web Applikation, die das Bearbeiten eines Verkehrsmodells ermöglicht, zu analysieren. Zudem wurde ein Prototyp erstellt, der die erkannten Risiken beseitigt und die wichtigsten Konzepte umsetzt.

3.1 Simulationsdaten

Die Simulationssoftware MATSim arbeitet mit dateibasierten Simulationsdaten, d.h. alle benötigten Daten für die Simulation werden in XML Dateien zur Verfügung gestellt. Diese XML Dateien können bis zu mehreren Gigabytes gross werden, wenn es sich um grössere Netzwerke, wie z.B. das Netz der gesamten Schweiz oder Deutschland, handelt. Das Verkehrsmodell-Netzwerk, welches der relevante Teil der Daten für unsere Arbeit darstellt, besteht aus mehreren Nodes und Links. Wie aus Abbildung 3.1 ersichtlich ist, setzt sich eine Strasse aus zwei Nodes, welche die Endpunkte darstellen und einem Link, der die Verbindung zwischen diesen beiden Nodes aufzeigt, zusammen.

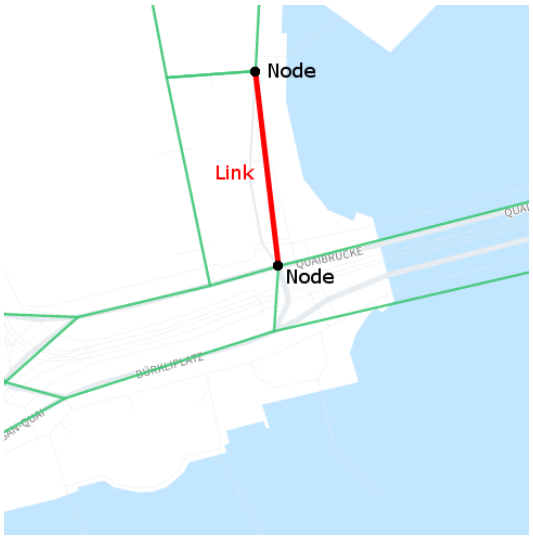


Abbildung 3.1: Beispiel Strasse: Nodes mit Link verbunden

Ein Link stellt somit eine klassische Strasse aus dem Verkehrsnetz dar. Im Verkehrsmodell wird oft eine Strasse in mehrere Links unterteilt, da für jeden Abschnitt eigene Eigenschaften gelten. Wie in Abbilung 3.2 ersichtlich, werden diese Eigenschaften zusammen mit den IDs der beiden Nodes des Links direkt auf dem Link gespeichert. Die genaue Position dieser Strasse erfahren wir somit durch die X und Y Koordinaten der beiden „from“- und „to“-Nodes.

```
<node id="1" x="680827" y="4825897"/>
<node id="2" x="680841" y="4825447"/>
<link id="3"
        from="1"
        to="2"
        length="450"
        freespeed="13.9"
        capacity="1964.0"
        permlanes="2.0"
        oneway="1"
        modes="car"
/>
```

Abbildung 3.2: XML Beispieldaten einer Strasse

Für die Bearbeitung dieser Daten verwendet die Senozon AG zur Zeit eine Client Applikation, die direkt mit den XML Dateien arbeitet. Dies führt dazu, dass diese Applikation sehr arbeitsspeicherintensiv ist, um trotzdem performant zu sein. Daher ist diese Applikation nicht für den Einsatz beim Kunden vor Ort geeignet.

3.2 Änderungsmanagement

Die bisherigen Änderungen an einem Verkehrsmodell wurden direkt an den vorhandenen Stammdaten vorgenommen. Dies führt dazu, dass die Stammdaten redundant in verschiedenen Verkehrsmodellen vorhanden sind. Dadurch müssen bei Änderungen der Stammdaten immer wieder alle Verkehrsmodelle überarbeitet werden.
In der Web Applikation sollen alle Benutzer mit denselben Stammdaten arbeiten können und nur die Änderungen individuell abgespeichert werden. Dadurch wird der benötigte Speicherplatz deutlich verringert und auch die Änderung der Stammdaten stellt somit kein Problem mehr dar.

Kapitel 4

Anforderungen & Risiken

4.1 Anforderungen

Im Rahmen dieser Bachelorarbeit wurden folgende funktionalen und nicht funktionalen Anforderungen von der Senozon AG an die Arbeit gestellt.

4.1.1 Funktionale Anforderungen

Der entwickelte Prototyp soll insbesondere aufzeigen, wie und mit welchen Konzepten gewisse Funktionalitäten dieser Web Applikation implementiert werden könnten. Dafür wurden folgende funktionalen Anforderungen an das Frontend sowie das Backend gestellt:

- Die Applikation soll das aktuelle Verkehrsmodell mit Hilfe einer Karte darstellen.
- Der Benutzer hat die Möglichkeit, ein Change Set zu laden, löschen und erstellen.
- Der Benutzer kann ein Attribut einer Strasse bearbeiten und die Änderung persistent speichern.
- Der Benutzer hat die Möglichkeit, über ein Formular neue Stammdaten zu importieren.
- Der Benutzer kann die Richtung einer Strasse ändern.

4.1.2 Nicht-funktionale Anforderungen

Zusätzlich zu den funktionalen Anforderungen ist es bei diesen grossen Datenmengen sehr wichtig, die nicht-funktionalen Anforderungen zu erfüllen. Folgende nicht-funktionalen Anforderungen haben uns vor besondere Herausforderungen gestellt:

- Performance: Effiziente Kommunikation zwischen Frontend und Backend für die Übertragung von Netzwerkdaten.
- Performance: Eine Abfrage inkl. Filterung von Netzwerkdaten auf der Datenbank muss unter einer Sekunde beantwortet werden können.

- Bedienbarkeit: Das User Interface soll einfach und möglichst ohne Tutorial bedienbar sein.
- Aussehen: Das User Interface soll im Design der Senozon AG erscheinen.
- Skalierbarkeit: Die Web Applikation soll skalierbar sein, damit auch grössere Mengen an Daten bewältigt werden können.

4.2 Risiken

In einer ersten Phase der Bachelorarbeit wurde eine Analyse der bestehenden Risiken in diesem Projekt durchgeführt. Während der gesamten Arbeit war das primäre Ziel, diese Risiken vollständig zu eliminieren.

Risiko 1: Grosse Datenmenge

Durch die grosse Datenmenge besteht die Gefahr, dass sehr viel Zeit in die Performance Optimierung investiert werden muss. Sollte dieses Risiko eintreffen, wird es sehr schwierig werden, einen voll funktionsfähigen Editor zu entwickeln.
Eine Massnahme, um dieses Risiko möglichst klein zu halten, ist es, die grosse Datenmenge bereits bei der Evaluation der Konzepte des Projektes zu berücksichtigen. Zusätzlich ist es sehr wichtig, möglichst früh die Performance des Prototyps mit grossen Datenmengen zu testen. Dadurch kann sehr schnell eingeschätzt werden, ob dieses Risiko weiterhin besteht oder bereits durch dafür vorgesehene Konzepte beseitigt werden konnte.
Dieses Risiko ist nicht eingetreten. Durch den Einsatz von verschiedenen Konzepten, wie z.B. des QuadTile Algorithmus [Ope16b] oder verschiedenen Optimierungen auf Seite der Software und der Datenbank, konnte eine gute bis sehr gute Performance erreicht werden.

Risiko 2: Aufwändige Backend Implementierung

Die Konzeptionierung und Implementation dieser Applikation konzentriert sich hauptsächlich auf das User Interface. Das Backend soll stark vereinfacht angenommen werden. Jedoch besteht das Risiko, dass ohne komplexeres Backend die grossen Datenmengen auf dem User Interface nicht effizient dargestellt werden können.
Als Massnahme wurde möglichst früh mit grossen Datenmengen gearbeitet, sodass der Umfang der Optimierungen, die auf der Seite des Backends vorgenommen werden müssen, zum Vorschein kam. Es stelle sich heraus, dass dieser Umfang um einiges grösser war als erwartet. Dies bedeutete einen deutlich höheren Aufwand in der Implementation des Backends, denn ohne des effizienten Umgangs mit den Daten im Backend ist der effektive Nutzen dieser Web Applikation nur sehr klein.

Kapitel 5

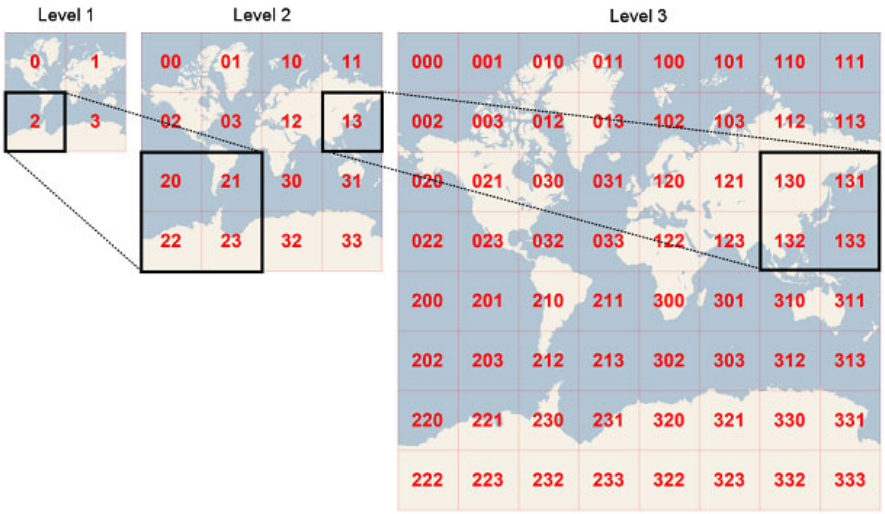
Konzepte & Architektur

5.1 Konzepte

Für die Bewältigung der Anforderungen an die Softwarelösung wurden Konzepte entwickelt, welche Einfluss auf die einzelnen Komponenten und die Architektur haben. Die Softwarelösung soll, wie unter Kapitel 4 Anforderungen & Risiken beschrieben, die Möglichkeit bieten, Simulationsdaten auf einer Karte darzustellen, sowie auch zu bearbeiten. Wichtig ist dabei, dass die Bearbeitung keinen Einfluss auf die Stammdaten hat. Die dafür benötigten Konzepte werden in diesem Kapitel beschrieben.

5.1.1 Daten QuadTiles

Der Umgang mit der grossen Datenmenge (ca. 4 Millionen Datensätze für die Schweiz) ist eine grosse Herausforderung dieser Arbeit. Um diese grosse Datenmenge zu bewältigen, wurde ein bewährtes Verfahren für die Kartendarstellung verwendet. Dies ist der QuadTile Algorithmus von OpenStreetMap [Ope16b]. Dabei werden die Daten nicht als Gesamtes geliefert, sondern über Teilbereiche, sogenannte Tiles, angefragt. Ein Tile ist ein Quadrat, welches einen vordefinierten Bereich der Welt abdeckt. Dies ermöglicht es, nur einzelne Bereiche (Tiles), die zur Zeit angezeigt werden, und nicht immer den kompletten Datenstamm zu laden. Ein weiteres Konzept in Verbindung mit dem QuadTile Algorithmus wird unter Kapitel 5.1.2 Preprocessing und Bewertung der Daten beschrieben. Dabei werden die Daten für jede Zoomstufe bewertet und lediglich bei Relevanz mitgeliefert. Durch diese beiden Konzepte kann die Datenmenge, die von der Website verarbeitet werden muss, stark reduziert werden.



Quelle: https://msdn.microsoft.com/en-us/library/bb259689.aspx

Abbildung 5.1: Fixe Aufteilung der Welt in Bereiche (Tiles)

Um von einem Datensatz (Link oder Node) auf das Tile zu schliessen, in welchem dieser sichtbar ist, wird ein Schlüssel (QuadKey) berechnet. Der QuadKey benennt eindeutig das kleinstmögliche Tile, in welchem der komplette Datensatz ersichtlich ist. Der QuadKey setzt sich aus der Nummerierung der Teilbereiche zusammen (vgl. Abbildung 5.1). Dabei wird die Welt in der äussersten Zoomstufe in 4 Bereiche aufgeteilt, die von 0 bis 3 durchnummeriert werden. Für die nächste Stufe wird jeder Bereich aus Level 1 erneut in 4 Bereiche aufgeteilt. Diese neuen Teilbereiche werden dann nummeriert. Dabei wird die Nummerierung des vorhergehenden Bereichs als Präfix verwendet und wieder von 0 bis 3 durchnummeriert. Dadurch kann jeder Bereich auf jeder Zommstufe eindeutig adressiert werden. Dieser Algorithmus bietet ebenfalls die Möglichkeit, mit Hilfe des QuadKeys alle darunterliegenden Bereiche zu finden, weil dieser QuadKey ein Präfix aller QuadKeys ist, die unter diesem Bereich liegen. Dieser Ansatz wird für die Filterung der Daten verwendet.

5.1.2 Preprocessing und Bewertung der Daten

Ein wichtiges Ziel der Aufbereitung der Daten ist das Verringern der Zugriffszeiten auf die Datenbank. Diese Aufbereitung findet direkt beim Import statt. Durch diesen Vorgang werden bewusst Redundanzen in das Datenmodell eingeführt. Diese Redundanzen erlauben den Zugriff auf Daten ohne teure JOIN-Statements oder SQL-Funktionen aufzurufen. Auf die Berechnungen und Aufbereitungen, die beim Import des Datenmodells angewendet werden, wird genauer im Kapitel 6.2 Tiling der Daten eingegangen.

5.1.3 Changeset

Der Benutzer kann auf dem Verkehrsmodell-Fallstudien-Editor Änderungen vornehmen und anschliessend persistieren. Dafür wurde ein Konzept entwickelt, das es erlaubt, Änderungen an den Daten abzuspeichern ohne die Stammdaten direkt anzupassen. Arbeitet der Benutzer direkt mit den Stammdaten, hat dies zur Folge, dass für jeden Benutzer die gesamten Stammdaten separat abgelegt werden müssen. Dies ist eine unnötige Redundanz der Daten und kostet viel Speicherplatz.
Um dem entgegen zu wirken, wird das Konzept Changeset verwendet. Dabei werden die Stammdaten nur einmalig abgelegt und von jedem Benutzer verwendet. Die Struktur eines Changesets stellt ein Abbild der Struktur der Stammdaten dar. Führt ein Benutzer eine Änderung an den Daten durch, wird lediglich die Differenz zu den Stammdaten in diesem Changeset abgelegt. Somit ist es möglich, wie in Abbilung 5.2 ersichtlich, das Changeset eines Benutzers als Differenz über die Stammdaten zu legen und somit den aktuellen Datensatz zu erhalten. Sollten Änderungen an den Stammdaten gemacht werden, sind diese direkt bei allen Benutzern ersichtlich, sofern die Änderungen nicht durch das Changeset überdeckt werden.

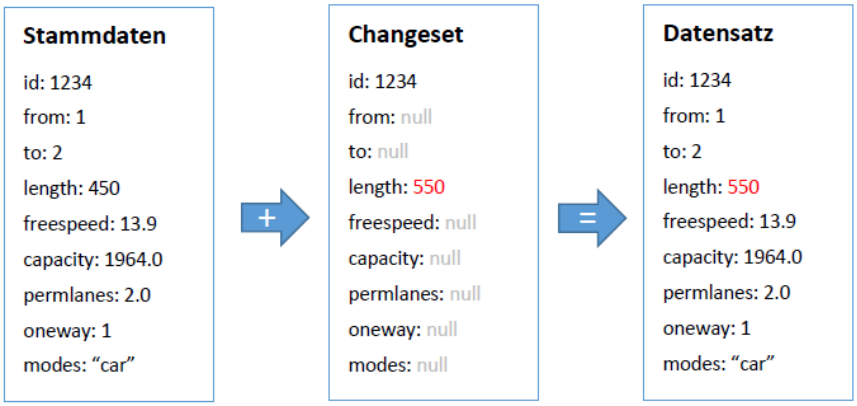


Abbildung 5.2: Changeset Beispiel

5.1.4 User Interface - Konzept

Das User Interface ist eine zentrale Komponente des Verkehrsmodell-Fallstudien-Editors und benötigt daher ein einfaches und intuitives Konzept, das ohne grosse Einarbeitungszeit zu bedienen ist. Für dieses Konzept diente Google Maps [Goo16b] bezüglich der Darstellung der Karte sowie auch der ID Editor [Ope16a] von OpenStreetMap bezüglich den Bearbeitungsmöglichkeiten als Inspirationsquelle für das User Handling. Beide Web Applikationen besitzen ein durchdachtes Design, an dessen Bedienung sich die Benutzer gewöhnt sind. In den folgenden Kapiteln folgt eine Analyse dieser beiden Web Applikationen sowie das daraus resultierende Detail Konzept.

Google Maps

Google Maps ist die am weitesten verbreitete Web Anwendung in Bezug auf die Benutzung einer Karte im Browser. Der Benutzer ist sich an das Handling von Google Maps gewöhnt und kann mit Systemen, die ähnlich aufgebaut sind, ohne Probleme arbeiten. Der grösste Vorteil von Google Maps ist die intuitive Bedienung, gekoppelt an ein einfaches, aufgeräumtes und leichtes Design.

Abbildung 5.3: Analyse Google Maps

Die Abbildung 5.3 zeigt die Auswertung der Analyse von Google Maps. Die Karte selbst erhält sehr viel Platz und alle Bedienelemente (Controls) werden am Randbereich der Karte angeordnet. Dadurch ist sichergestellt, dass sich der Benutzer auf eine Kernaufgabe konzentriert und sich nicht ablenken lässt. Bei schlechteren Beispielen, auf denen die Karte nur wenig Platz erhält, fühlt sich der Benutzer sehr schnell eingegrenzt und muss deutlich mehr Aufwand ins Zoomen und das seitliche Bewegen investieren. Dies kann einen Benutzer stören und ihn vom schnellen Erledigen der Aufgabe abhalten. Das User Interface passt sich laufend an die Nutzung des Benutzers an. Hat der Benutzer einen Ort gefunden und möchte eine Route berechnen, wird ihm neu diesbezüglich ein Menü eingeblendet. Dadurch wird zwar der Karte Platz genommen, jedoch liegt nun das Hauptaugenmerk auf dem Planen der Route. Zusätzlich werden zum Einsparen von Platz bei den Bedienelementen meist kein Text sondern Symbole eingesetzt.

Ergebnisse Analyse

Folgende Punkte fliessen in das Design des Verkehrsmodell-Fallstudien-Editors mit ein:

- Der Karte viel Platz einräumen.
- Menüs, welche nicht dem aktuellen Use Case entsprechen, ausblenden.
- Viele Symbole und wenig Text verwenden.

ID Editor

Der ID Editor von OpenStreetMap ist der bekannteste Editor für OpenStreetMap. Er bietet die Möglichkeit, Daten von OpenStreetMap direkt auf einer Karte zu bearbeiten. Die Änderungen werden dann als Changeset an OpenStreetMap übertragen und freigeschaltet. Die Funktionalität des Verkehrsmodell-Fallstudien-Editors ähnelt sehr stark den Funktionen des ID Editors. Aufgrund dessen wurde das User Interface des ID Editors ebenfalls analysiert.

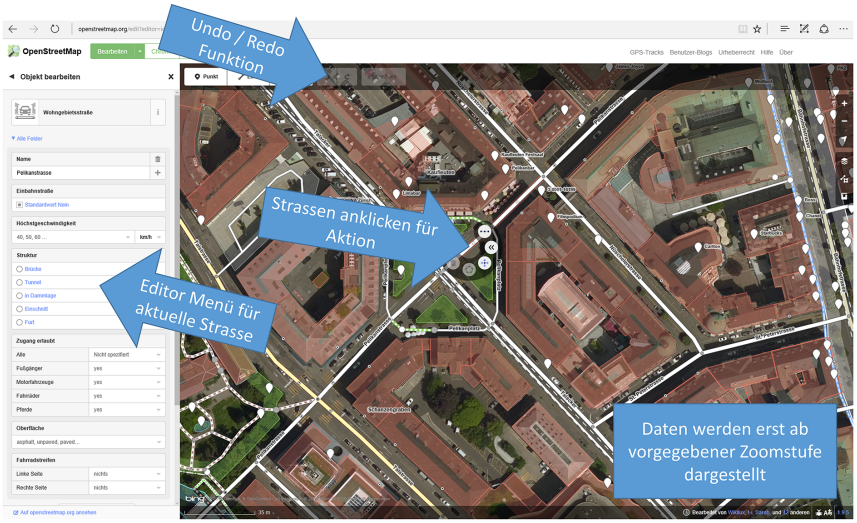


Abbildung 5.4: Analyse ID Editor

Die Abbildung 5.4 zeigt die Auswertung der Analyse des ID Editors. Eine Kerneigenschaft des User Interfaces ist es, dass die Daten erst ab einer gewissen Zoomstufe angezeigt werden. Dies hauptsächlich, weil der ID Editor eine deutlich grössere Datenmenge verarbeiten muss. Diese Eigenschaft ermöglicht es dem ID Editor, die Datenmenge, die zur selben Zeit dargestellt werden muss, auf ein Minimum zu reduzieren. Das Bearbeitungsmenü (Editor Menü) ist so aufgebaut, dass es nach dem Auswählen eines Elements die dazugehörigen Daten des Elements zur Bearbeitung anzeigt. Die Tatsache, dass die Karte dabei immer ersichtlich bleibt, ermöglicht eine leichte und schnelle Bedienung für den Benutzer. Eine weitere wichtige Funktion des ID Editors ist die Undo- / Redo-Funktion. Änderungen an Elementen können einfach rückgängig gemacht werden oder wiederhergestellt werden.

Ergebnisse Analyse

Folgende Punkte fliessen in das Design des Verkehrsmodell-Fallstudien-Editors mit ein:

- Karte bleibt während der Bearbeitung von Strassenattributen ersichtlich.
- Bearbeitung der Strassen durch Auswahl (anklicken).
- Undo- / Redo-Funktion.

Detail Konzept

Aus den Erkenntnissen der Analyse von Google Maps und dem ID Editor wurde folgendes Konzept für das User Interface des Verkehrsmodell-Fallstudien-Editors entwickelt.

Abbildung 5.5: Wireframe Editor

Angelehnt an Google Maps, wird das User Interface eine Karte besitzen, die den gesamten Platz im Browser ausfüllt. Die Bedienelemente in der linken oberen Ecke werden als selbsterklärende Symbole dargestellt und sind während der gesamten Benutzung der Applikation ersichtlich. Menüs, die zur Zeit nicht gebraucht werden, sind wie in Abbildung 5.5 dargestellt, ausgeblendet. Dies bietet dem Benutzer viel Platz für das Suchen und Auswählen einer für ihn relevanten Strasse. Durch das Auswählen einer Strasse wird ein Menü von der rechten Seite her eingeblendet, womit das Bearbeiten der Parameter dieser Strasse ermöglicht wird (vgl. Abbildung 5.6).

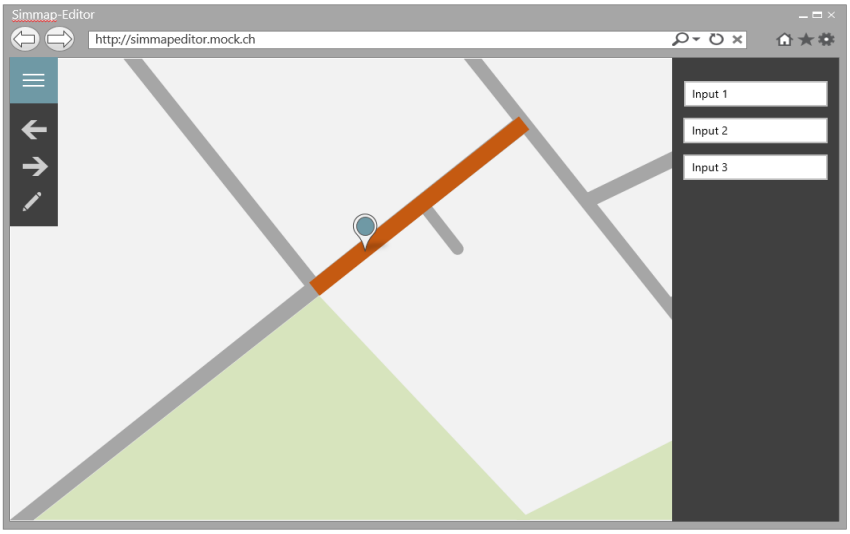


Abbildung 5.6: Wireframe Strassenattribute bearbeiten

Um die Wegführung einer Strasse zu bearbeiten, kann der Benutzer in einen Bearbeitungsmodus wechseln. In diesem Modus werden zusätzlich zu den Links auch die Nodes angezeigt, die ansonsten aus Performancegründen nicht geladen werden. Der Benutzer kann eine Strasse auswählen und deren Führung mit Hilfe eines Menüs ändern.

Abbildung 5.7: Wireframe Strassenführung bearbeiten

Ein weiteres, naheliegendes Konzept für die Bearbeitung der Strassenführung, wäre ein Drag & Drop Verfahren. Dieses Konzept wäre für den Benutzer am einfachsten, ist jedoch sehr zeitintensiv in der Implementation. Aus Zeitgründen musste dadurch auf Drag & Drop verzichtet werden.

5.2 Architektur

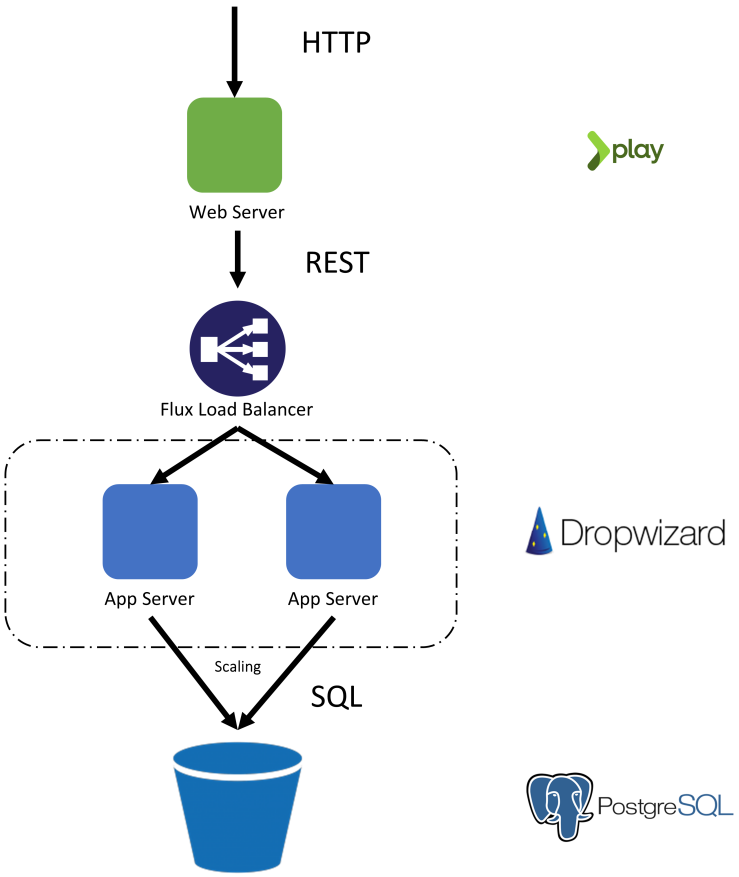


Abbildung 5.8: Tier des Verkehrsmodell-Fallstudien-Editors

Aufgrund der Anforderungen bezüglich Performance und Antwortgeschwindigkeit an diese Arbeit, musste die Architektur für die Softwarelösung des Verkehrsmodell-Fallstudien-Editors skalierbar sein. Es muss daher eine Entkopplung der Komponenten durchgeführt werden, die es erlaubt, die Last auf mehrere Server zu verteilen.
Die Softwarelösung des Verkehrsmodell-Fallstudien-Editors ist als 3-Tier Applikation aufgebaut. Der Frontend-Tier ist eine Web Applikation, umgesetzt mit dem Play Framework. Der Backend-Tier wird als REST Service (Maturity Level 2 [Fow16]) mit Dropwizard entwickelt. Auf diesen Tier wird über einen Flux Load Balancer zugegriffen, der für die Skalierung verantwortlich ist. Dies erlaubt eine parallele Ausführung mehrerer Anfragen. Der letzte Tier, der Daten-Tier, wird durch eine PostgreSQL Datenbank bereitgestellt.

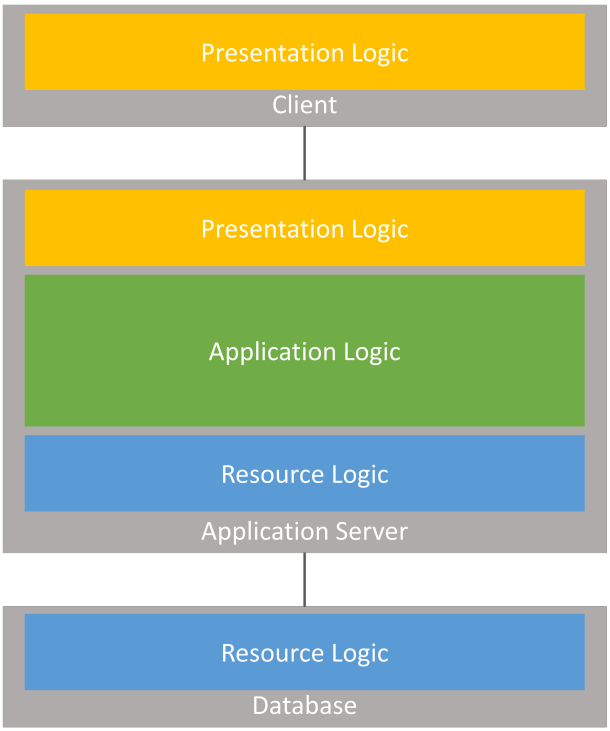


Abbildung 5.9: Tier- / Layeraufteilung

Die Aufteilung der Presentation Logic auf den Client-Tier sowie den Backend-Tier erlaubt es, die Last der Datendarstellung auf die verschiedenen Komponenten zu verteilen. Die Vorbereitung und Aufbereitung der Daten wird vom Backend-Tier übernommen. Die Daten werden anschliessend in einem für den Client lesbaren GeoJSON [Geo16a] Format an den Client-Tier übertragen, der dann das Rendering dieser Daten übernimmt. In Abbildung 5.9 ist die gesamte Tier- und Layeraufteilung ersichtlich.

5.3 SimMapEditor

Abbildung 5.10: SimMapEditor

Der SimMapEditor ist die Frontend-Komponente der Architektur und basiert auf dem Play Framework. Der Client-Tier ist eine in Javascript entwickelte Software. Für die Abfrage der Daten, die auf der Karte dargestellt werden, greift sie asynchron über Ajax auf den Backend-Tier zu. Das Rendering der Daten übernimmt ebenfalls der SimMapEditor. Er selbst besitzt keine eigene Business Logik, da diese komplett auf den Backend-Tier

ausgelagert und dadurch nicht im Play Framework implementiert ist. Der SimMapEditor ist dafür zuständig, das User Interface Konzept (vgl. Kapitel 5.1.4 User Interface - Konzept) sowie auch das Tiling der Daten (vgl. Kapitel 5.1.1 Daten QuadTiles) zu implementieren und somit die Daten für die Karte vom Backend-Tier in Bereichen anzufordern.

5.4 SimMapService

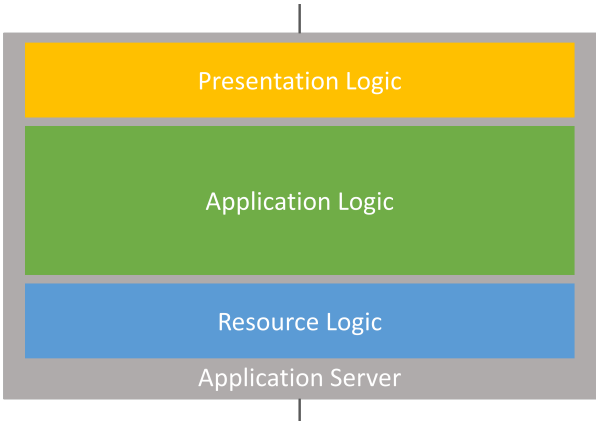


Abbildung 5.11: SimMapService

Der SimMapService stellt den Backend-Tier der Softwarelösung dar und ist in einer 3-Layer Architektur implementiert. Er wird als REST Service (Maturity Level 2 [Fow16]) entwickelt. Um dem Maturity Level 2 zu entsprechen, werden die Anfragen und Antworten des Services mit HTTP Standard Methoden und Codes ausgestattet. Mit den Codes wird dem Client angezeigt, welchen Status die Antwort besitzt. Beispielsweise wird bei einer erfolgreichen Anfrage, in der keine Daten in der Antwort enthalten sind, der HTTP Status Code „204 - No Content“ zurückgesendet. Somit weiss der Client, dass die Anfrage erfolgreich war, jedoch keine Daten im Body enthalten sind. Für das Entgegennehmen, Verarbeiten und Aufbauen der Antworten mit den korrekten Codes ist die Presentation Logic Schicht des SimMapServices verantwortlich.
Die Application Logic Schicht des SimMapServices ist für die Durchführung der Aufbereitung (vgl. Kapitel 5.1.2 Preprocessing und Bewertung der Daten) zuständig. Zusätzlich lädt die Application Logic Schicht je nach Zoomstufe und Koordinaten Daten aus der Resource Logic Schicht aus und bereitet diese auf. Anschliessend werden diese Daten an die Presentation Logic Schicht für das Weiterleiten an den Client weitergegeben.
Die Resource Logic Schicht bietet der Application Logic Schicht die Möglichkeit, auf die Daten in der Datenbank zuzugreifen. Dabei ist es die primäre Aufgabe dieser Schicht, Datenbankabfragen zu erstellen und durchzuführen.

5.5 Datenbank

Aufgrund der grossen Datenmenge, welche ein Verkehrsmodell umfassen kann, ist die Datenbank ein essentieller Teil der Architektur. Das Datenmodell wurde auf die Trennung der Stammdaten, von den Daten der Changesets (vgl. Kapitel 5.1.3 Changeset) ausgelegt.

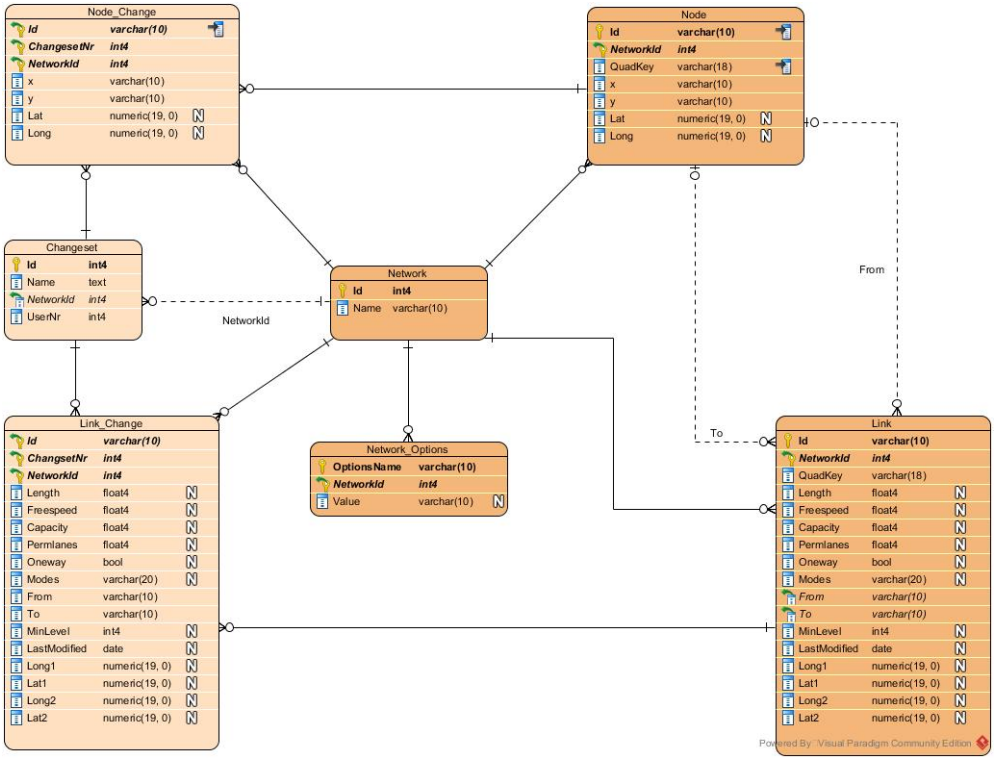


Abbildung 5.12: ERP der Datenbank

Die zentrale Komponente des Datenbankschemas, wie in Abbildung 5.12 ersichtlich, ist das Netzwerk. Das Netzwerk wird aus dem XML eingelesen, das über den Service importiert wurde. Jedes Netzwerk enthält sowohl Links als auch Nodes, die zusammen die Strassen des Verkehrsmodells definieren. In der Tabelle Network_Options werden generelle Eigenschaften des Netzwerks aus dem XML gespeichert.
Die Tabellen Node_Change und Link_Change stellen ein Abbild der Tabellen Node und Link dar. In diesen beiden Tabellen sowie auch der Tabelle Changeset werden die Daten eines Changesets abgespeichert. Dabei ist wichtig, dass lediglich die Differenz zu den originalen Datensätzen aus den Tabellen Link und Node abgespeichert werden. Die restlichen Spalten werden auf null gesetzt.
Das Datenmodell enthält bewusst einige Redundanzen. Die Koordinaten der Nodes werden aus Performancegründen sowohl direkt auf dem Node als auch in den dazugehörigen Links abgespeichert. Im Kapitel 6.4.4 Datenbank Redundanz werden die Gründe dazu genauer erläutert.

Kapitel 6

Implementation

Für die Implementation dieser Bachelorarbeit wurden die im vorherigen Kapitel 5 Konzepte & Architektur beschrieben Konzepte angewendet.

6.1 XML Daten Import

Um mit den grossen Datenmengen in den vorhandenen XML Dateien effizient zu arbeiten, müssen diese Daten in eine Datenbank importiert werden. Dafür wurde ein Import Formular im Web Interface integriert, das im Hintergrund die Daten an einen Web Service sendet, der für den Import zuständig ist. Die Daten werden direkt beim Import aufbereitet, sodass die Abfrage der Daten schneller ist.

6.2 Tiling der Daten

Während des Imports werden zwei verschiedene Vorberechnungen durchgeführt. Mit Hilfe dieser Aufbereitung ist es möglich, die Daten effizienter zu filtern. Dadurch wird die Antwortzeit einer Anfrage deutlich verkürzt.

6.2.1 QuadKey

Die erste Vorbereitung ist die Berechnung des QuadKeys. Wie in Kapitel 5.1.1 Daten QuadTiles beschrieben, können mit Hilfe des QuadKeys einzelne Bereiche der Welt eindeutig identifiziert werden. Wenn nun eine Anfrage an den Server gesendet wird, in der die Daten eines spezifischen QuadTiles angefordert werden, müssen die Links ebenfalls einen QuadKey besitzen, um danach zu filtern. Dafür werden die QuadKeys der beiden Nodes berechnet, die zu einem Link gehören. Der gemeinsame Präfix stellt nun den kleinstmöglichen QuadKey dar, in den der gesamte Link hineinpasst.
Wie in Abbildung 6.1 bei Link 2 ersichtlich, kann eine Strasse in einem QuadTile ersichtlich sein, ohne dass sich beide Nodes in diesem Quadtile befinden. In diesem Beispiel hat Link 1 einen QuadKey von 010 und Link 2 von 01. Damit nun der Link 2 bei einer

Abfrage der Daten des QuadTiles 010 auch mitgeliefert wird, müssen alle Strassen, die einen QuadKey besitzen, der einen Prefix des angeforderten QuadKeys darstellt, auch mitgeliefert werden. D.h. dass nun für die Abfrage des QuadTiles 010 alle Strassen, die den QuadKey 010, 01, 0 oder sogar einen leeren QuadKey („“) besitzen mitgeliefert werden.

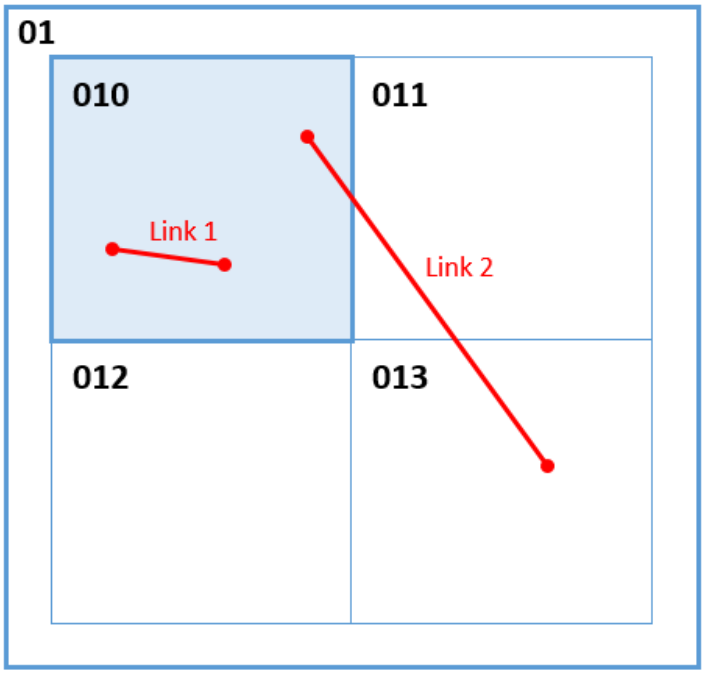


Abbildung 6.1: Beispiel QuadTile
QuadKey Link 1: 010
QuadKey Link 2: 01

Der QuadKey ist eine sehr gute Technik um die Daten auf der Karte zu adressieren und dadurch eine Filterung der Daten durchzuführen. Zudem stellt diese Technik sicher, dass alle in einem gewissen QuadTile ersichtlichen Links sicher mitgeliefert werden.
Ein Nachteil dieser Technik ist es, dass bei höheren Zoomlevels Daten mitgeliefert werden, die bei einem vorherigen QuadTile bereits mitgeliefert wurden. Das User Interface verhindert zwar das neue Zeichnen von Duplikaten, jedoch werden dadurch Datensätze übertragen, die bereits gezeichnet sind und dadurch überflüssig sind. In den für diese Arbeit zur Verfügung stehenden Modellen hält sich die Menge von Links, die einen kleinen Präfix besitzen in Grenzen. Dadurch ebenfalls die Menge an Links, die mehrfach übertragen werden oder ausserhalb des sichtbaren Bereichs gezeichnet werden.

6.2.2 MinLevel

Zusätzlich zu dem QuadKey wird beim Import für jeden Link der Mindest-Level (MinLevel) berechnet. Denn nur mit dem QuadKey würden alle Strassen in der aktuell angezeigten Bounding Box geladen werden, auch die eher kleineren Nebenstrassen. Aus diesem Grund wird beim Import ein MinLevel berechnet, der besagt, ab welchem Zoom Level in der Karte diese Strasse geladen wird. Dadurch ist es möglich, eine Strasse nach ihrer Wichtigkeit zu beurteilen und erst ab einem bestimmten Zoomlevel anzuzeigen.
Bei der Berechnung dieses Mindest-Levels wird in erster Linie auf die maximale Geschwindigkeit geachtet. Die Geschwindigkeit sagt sehr viel über die Wichtigkeit der Strasse selbst aus. Eine Strasse mit einer hohen maximalen Geschwindigkeit, wie z.B. eine Autobahn, wird in diesem System als wichtig erachtet und dadurch bereits auf einem niedrigeren Zoomlevel angezeigt. Beim Level 10 und 14 wurden noch weitere Kriterien verwendet, um eine feinere Skalierung zu erhalten.
Der MinLevel einer Strasse wird wie folgt berechnet:

Level 10	=>	Geschwindigkeit > 30 m/s und Anzahl Spuren > 2
Level 11	=>	Geschwindigkeit > 30 m/s
Level 12	=>	Geschwindigkeit > 23 m/s
Level 13	=>	Geschwindigkeit > 14 m/s
Level 14	=>	Geschwindigkeit > 13 m/s und Kapazität >= 4000
Level 15	=>	Geschwindigkeit > 13 m/s
Level 16	=>	Der Rest

Alternativ zu den verwendeten Kriterien könnte die Berechnung des MinLevels auch mit Hilfe der Kapazität und zur feineren Abstufung der Anzahl Spuren der einzelnen Strassen durchgeführt werden. Dieses Verfahren hat jedoch eine unregelmässigere Verteilung auf die einzelnen Zoomlevels zur Folge. Aus diesem Grund wurde das Verfahren mit der Geschwindigkeit gewählt.

6.3 User Interface - Implementation

Wie in Kapitel 5.1.4 User Interface - Konzept beschrieben, muss das User Interface des Verkehrsmodell-Fallstudien-Editors einigen Anforderungen entsprechen. Dieses User Interface wird als Javascript Single Page Anwendung konzipiert und implementiert. Dies hat zum Vorteil, dass der Benutzer die Seite nie wechselt, sondern alle benötigten Elemente laufend nachgeladen werden. Dadurch erhält der Benutzer eine flüssigere User Experience. Die Web Applikation wird in eine Main View (Haupt-Anzeige) und mehrere Partial Views (Teil-Anzeigen) aufgeteilt. Die Partial Views werden im Play Framework als Route erfasst und mittels Javascript angefordert. Anschliessend werden diese Partial Views in einen Container geladen. Die Abbildung 6.2 zeigt den Aufbau des User Interface mit einer eingezeichneten Partial View.

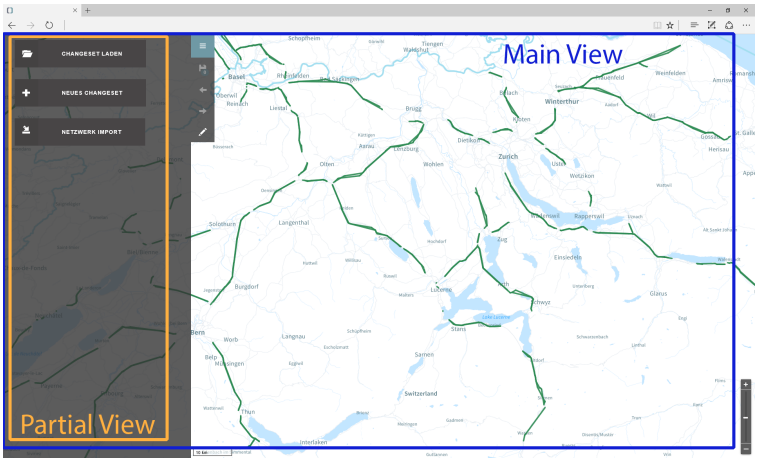


Abbildung 6.2: User Interface mit eingezeichneter Partial View

Wie in Abbildung 6.2 ersichtlich, wurde der Karte viel Platz eingeräumt. Für die Darstellung der Karte wurde das Framework Leaflet [Lea16] sowie d3.js [D316] verwendet. In der Kombination erlauben diese Frameworks die Erstellung einer Karte mit beliebigem Kartenmaterial im Hintergrund. Für die Karte wurde ein eigenes Kartendesign über Mapbox[Map16] erstellt. Um zusätzlich zu der Karte eine Schicht mit den Daten aus dem Verkehrsmodell darzustellen, musste eine Erweiterung für Leaflet (TileLayer GeoJSON [Geo16b]) verwendet werden. Diese Erweiterung ermöglicht es, eine Schicht aufzubauen, welche die Daten für die Darstellung von einer URL nach dem OpenStreetMap Schema (x, y , zoom) anfordert und dabei GeoJSON als Antwortformat erwartet. Diese Library wurde gemäss den Anforderungen des Verkehrsmodell-Fallstudien-Editors erweitert. Mit Hilfe dieser Libray konnte das Konzept des Tiling der Daten (vgl. Kapitel 5.1.1 Daten QuadTiles) umgesetzt werden. Für die Interaktionen mit den einzelnen Editoren wird AngularJS [Goo16a] Version 1.5 verwendet. AngularJS ermöglicht das Binding eines Models an Datenfelder und dadurch auch das Bearbeiten dieser Models. Z.B. wird beim Anklicken einer Strasse das Model der Strasse an das Edit-Menü gebunden und kann direkt bearbeitet werden (vgl. Abbildung 6.3).

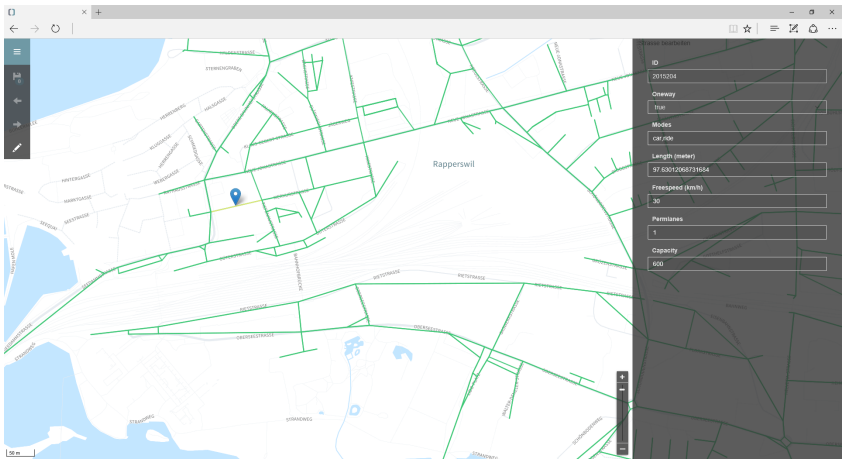


Abbildung 6.3: User Interface mit geöffnetem Editor für Strassenattribute

Um die Wegführung einer Strasse zu bearbeiten, kann eine Strasse angeklickt werden. Dabei öffnet sich ein Bearbeitungsfenster, in dem der Start und Endpunkt der Strasse mittels Klick auf den entsprechenden Node bearbeitet werden kann (vgl. Abbildung 6.4). Nach der Bearbeitung wird die Strasse neu gezeichnet und hervorgehoben.

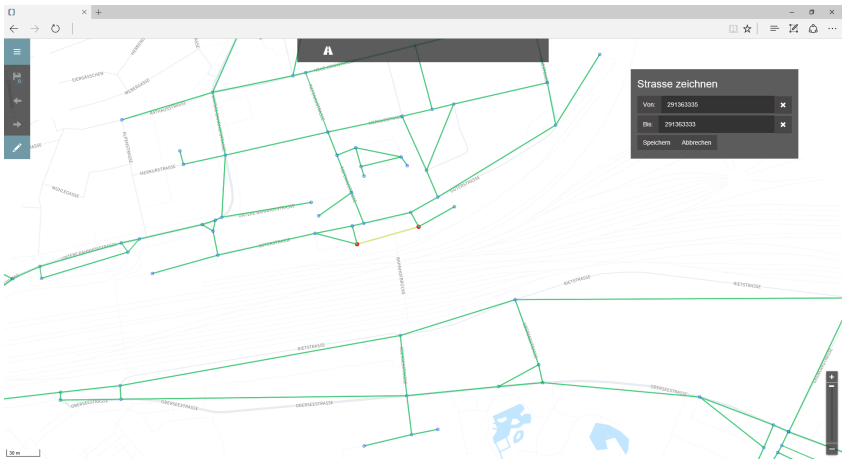


Abbildung 6.4: User Interface mit geöffnetem Editor für Strassenführung

Änderungen, welche über den Attribut- oder Strassenführungseditor erstellt werden, werden auf einem Undo-Stack abgelegt. Dadurch können Änderungen über eine Undo-Funktion wieder rückgängig gemacht werden und über ein Redo wiederhergestellt werden. Das Changeset sowie der Undo- und Redo-Stack sind im Session Storage des Browsers abgelegt. Dadurch gehen die Änderungen nicht verloren, solange die Browser Session noch offen ist. Wird der Browser geschlossen, gehen die Änderungen durch die Löschung der Session verloren. Der Benutzer kann seine Änderungen speichern, so dass sie in der Datenbank persistiert werden. Der Vorteil des Session Storage liegt darin, dass ein Benutzer die Seite neu laden kann oder ein Tab schliessen kann, ohne dass seine lokalen Änderungen verloren gehen.

6.4 Performance Optimierungen

Die Performance ist ein sehr wichtiger Faktor in dieser Arbeit. Ohne starke Performance kann die grosse Datenmenge in diesem Projekt nicht effizient dargestellt werden. Aufgrund dessen wurde einiges an Zeit in die Optimierung investiert. Zwei wichtige Faktoren haben grossen Einfluss auf die Performance dieser Web Applikation. Zum einen ist dies die Antwortzeit der einzelnen Abfragen der Daten. Diese Antwortzeit konnte durch mehrere Optimierungen, die in diesem Kapitel genauer beschrieben werden, deutlich verbessert werden. Dadurch konnten die nicht-funktionalen Anforderungen aus dem Kapitel 4.1.2 Nicht-funktionale Anforderungen eingehalten werden.
Ein weiterer grosser Faktor ist das Rendering der Daten im Browser. In höheren Zoomlevels werden bis zu mehreren Tausend Links und Nodes im Browser dargestellt. Der Browser benötigt für das Rendering dennoch eine gewisse Zeit, auf die der Entwickler keinen grossen Einfluss hat. Es kann lediglich darauf geachtet werden, dass keine redundante Informationen vom Browser gerendert werden müssen.

6.4.1 Caching

Für die Anzeige der Strassen und Nodes werden sehr viele Daten übertragen. Damit die Daten bei mehrmaligem Anfragen nicht jedes Mal übertragen werden müssen, wird clientseitiges Caching eingesetzt.
Der Server gibt dem Client vor, wie er die Daten zwischenspeichern soll. Dafür wird bei jeder Antwort ein maximales Alter der Daten gesetzt. Wie in Abbildung 6.5 ersichtlich, speichert der Client die Daten in seinen lokalen Zwischenspeicher (Cache) und verwendet diese, solange das maximale Alter von einer Stunde noch nicht erreicht ist. Dies macht Sinn, weil sich die Stammdaten des Verkehrsmodells nur sehr selten ändern. Erst nach Ablauf dieser Stunde sendet der Browser erneut eine Anfrage an der Server.

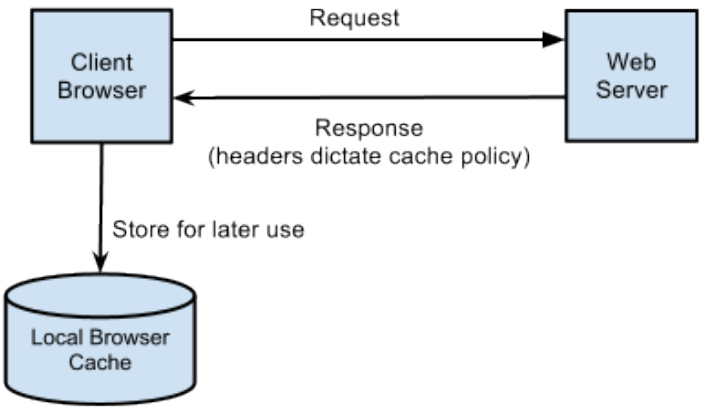


Abbildung 6.5: Browser Zwischenspeicher Request (Cache) [Her16]

Nun liegt es am Server zu prüfen, ob die Daten, die sich im Cache des Clients befinden, noch aktuell sind. Dafür wird mit einem HTTP Entity Tag gearbeitet [Wik16]. Dieser setzt sich aus der x- und y-Koordinate, dem Zoom Level sowie aus dem Zeitstempel der letzten Bearbeitung der Daten zusammen. Durch diese unterschiedlichen Informationen ist der Entity Tag für jeden QuadTile eindeutig. Durch das zusätzliche Verwenden des Zeitstempels ist auch sichergestellt, dass sich der Entity Tag unterscheidet, sobald eine Änderung in den angeforderten Stammdaten auf dem Server vorliegt.
Jede Antwort, die zuvor an den Client gesendet wurde, wird mit diesem HTTP Entity Tag versehen. Beim erneuten Anfragen der Daten, wird dieser Entity Tag vom Client mitgesendet und mit dem berechneten, aktuellen Entity Tag der Daten auf dem Server verglichen. Wie in Abbildung 6.6 ersichtlich, wird vom Server der HTTP Status 304 (Not Modified) [W3.16] gesendet, falls diese identisch sind. Dadurch weiss der Client, dass seine Daten noch aktuell sind. Wurden die Stammdaten auf dem Server geändert, stimmen die Entity Tags aufgrund des Zeitstempels der letzten Bearbeitung nicht mehr überein und der Server sendet die neuen Daten an den Client. Diese Daten werden dann vom Client wieder für eine Stunde zwischengespeichert und anschliessend beginnt der Prozess von vorne. Dieses Caching kann zu einer Inkonsistenz der Daten im Cache und dem Server führen. Jedoch wurde dieses Risiko bewusst eingegangen weil ein Update der Stammdaten nur sehr selten vorkommt. Zudem kann ein Update der Stammdaten zeitlich so gelegt werden, dass die Benutzer davon nicht betroffen sind.

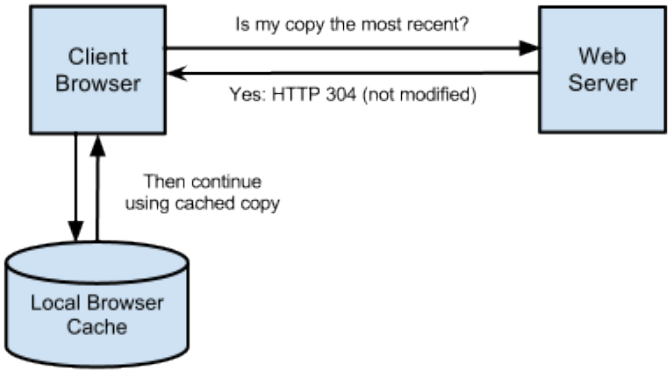


Abbildung 6.6: HTTP Entity Tag Request [Her16]

Durch dieses Verfahren ist sichergestellt, dass das Übertragen von Daten möglichst reduziert wird und dadurch Performance gewonnen wird. Das Beantworten eines Requests, der noch den selben Entity Tag verwendet und somit im Cache noch aktuell ist, benötigt ca. 50ms. Hingegen das erneute Senden würde ca. 200ms bis 300ms benötigen. Dadurch ist das Beantworten um den Faktor vier bis sechs schneller. Diese gewonnene Performance wird erst beim längeren Verwenden der Applikation bemerkbar, sobald bereits geladene Daten nicht mehr erneut geladen werden müssen.

6.4.2 Erneutes Zeichnen

Die verwendete Library „Leaflet GeoJSON Tile Layer“ [Geo16b], die für das Abfragen der Daten in Tiles verantwortlich ist, wurde standardmässig so entwickelt, dass die Daten auf jedem Zoom Level neu gezeichnet werden. Im Rahmen dieser Arbeit wurde die Funktionalität erweitert, so dass die Strassen beim Hineinzoomen bestehen bleiben und lediglich die neuen Strassen dazu gezeichnet werden. Ebenfalls wurde die Library so angepasst, dass beim Herauszoomen alle Strassen, die auf einem höheren Zoomlevel gezeichnet wurden, wieder entfernt werden. Dafür werden alle Schlüssel der gezeichneten Strassen mit dem dazugehörigen Zoomlevel, auf dem sie gezeichnet wurden, in eine Map< Zoomlevel, List<Schlüssel> > abgelegt. Dadurch kann die Library beim Herauszoomen alle Schlüssel, die zu einem Zoomlevel gehören, löschen.

6.4.3 Datenbank Index

Der Verkehrsmodell-Fallstudien-Editor arbeitet mit einer grossen Datenmenge, die in einer Datenbank gehalten wird. Die Abfrage einer Teilmenge der Daten benötigte bei einer Datenmenge von 4 Millionen Datensätzen bis zu drei Sekunden. Diese Antwortzeit ist deutlich zu lange und eine Auswertung des Ausführungsplans zeigte, dass die Indizes, die auf die einzelnen Zeilen gesetzt sind, nicht verwendet wurden.
Die Abfrage der Strassen beinhaltet eine LIKE Abfrage von SQL, die das Suchen von Strassen mit QuadKeys ermöglicht, die den QuadKey des aktuellen Tiles als Prefix besitzen. Speziell für SQL Abfragen mit einem LIKE Operator, bietet PostgreSQL eine Operator Klasse „varchar_pattern_ops“, die zusätzlich auf den Index des QuadKeys gesetzt werden kann. Dadurch wird der Index speziell für solche LIKE Abfragen vorbereitet und deutlich schneller. Nach dem Hinzufügen dieser Operator Klasse war der Zugriff auf die Daten zehn Mal so schnell und wir erreichten eine Zugriffszeit zwischen 100ms und 300ms.

6.4.4 Datenbank Redundanz

Die Daten in der Datenbank sind in eine Tabelle „Links“ und eine Tabelle „Nodes“ aufgeteilt. Bei der Abfrage werden sowohl die Parameter des Links als auch die Koordinaten der dazugehörigen Nodes benötigt. Dafür wurde zu Beginn ein klassischer SQL JOIN verwendet, um die beiden Tabellen miteinander zu verknüpfen. Durch eine erneute Analyse des Ausführungsplans wurde auch hier ersichtlich, dass dieser JOIN ca. die Hälfte der Ausführungszeit in Anspruch nahm. Um diese Zeit einzusparen, werden nun die Koordinaten der beiden Nodes direkt beim Import der Daten in die Tabelle der Links mit abgespeichert. Dadurch sind die Koordinaten der Nodes zwar redundant vorhanden, jedoch beschleunigt dies die Abfrage um den Faktor 2. Daher wurde die Redundanz der Daten in Kauf genommen.

Kapitel 7

Resultate

7.1 Performance

Wie in Kapitel 4.1.2 Nicht-funktionale Anforderungen beschrieben, ist die Performance eine der wichtigsten nicht-funktionalen Anforderungen. Dies hatte zur Folge, dass besonders viel Zeit in deren Optimierung investiert wurde. In Kapitel 6.4 Performance Optimierungen wird genauer auf die durchgeführten Optimierungen eingegangen.
Grundsätzlich gab es drei verschiedene Stufen von Performance, die erreicht wurden. In diesem Kapitel werden diese drei Stufen genauer analysiert und beschrieben.

Stufe 1

In der ersten Stufe, zu Beginn des Projektes, wurde das gesamte Verkehrsmodell-Netzwerk mittels einer einzigen Abfrage beim Öffnen der Applikation geladen. Dabei wurde mit dem öffentlichen Netzwerk von Santiago de Chile gearbeitet. Dies beinhaltet ca. 60'000 Datensätze. Gemäss Tests benötigte diese Abfrage im Durchschnitt ca. 6 Sekunden. Mittels dieser Vorgehensweise wäre es unmöglich gewesen, ein Netzwerk wie die Schweiz mit ca. 4 Millionen Datensätzen in dieser Applikation zu laden und anzuzeigen.

Stufe 2

In der nächsten Stufe wurde der QuadTile Algorithmus von OpenStreetMap eingeführt. Dadurch wird nicht mehr das gesamte Verkehrsmodell-Netzwerk geladen, sondern die Daten in Bereiche aufgeteilt. In dieser Stufe war es bereits möglich mit den 4 Millionen Datensätzen aus dem Verkehrsmodell der Schweiz zu arbeiten. Obwohl die Antwortzeiten immer noch deutlich zu hoch waren, konnte das Anzeigen des Schweizer Verkehrsmodells gemeistert werden.
Folgende Zugriffszeiten wurden gemessen:

Zoomstufen 10 - 14	=>	1 - 3 Sekunden
Zoomstufen 14 - 18	=>	3 - 10 Sekunden

Stufe 3

In der dritten Stufe wurden besonders im Backend sehr viele Optimierungen vorgenommen. Sei dies das Client-seitige Caching, das Verhindern vom erneuten Zeichnen einer Strasse beim Wechsel der Zoomstufe, der Einsatz einer speziellen Operator Klasse für den Index oder die absichtliche Redundanz in dem Datenmodell. Genauere Beschreibungen der einzelnen Optimierungen sind in Kapitel 6.4 Performance Optimierungen zu finden. Alle diese Optimierungen zusammen führten dazu, dass die Applikation eine deutlich höhere Performance aufwies und dadurch die Bearbeitung des Schweizer Verkehrsmodells kein Problem mehr darstellte.
Folgende Zugriffszeiten wurden gemessen:

Zoomstufen 10 - 14	=>	50 - 100 Millisekunden
Zoomstufen 14 - 18	=>	100 - 300 Millisekunden

Resultat

Die daraus resultierte Performance erfüllt die Anforderungen und ermöglicht ein benutzerfreundliches Bearbeiten des Schweizer Verkehrsmodells. Dennoch entsprechen die geladenen Links und Nodes einer sehr grosse Datenmenge, zum Teil mehr als 10'000 Datensätze pro Anzeige. Das Rendering dieser Datenmenge benötigt eine gewisse Zeit und verzögert dadurch das Darstellen der Daten auf der Karte. Auf die Geschwindigkeit des Renderings des Browsers konnte jedoch kein Einfluss genommen werden.

7.2 Rückblick Technologien

Leaflet

Für die Darstellung der Karte wird die Library Leaflet in Verbindung mit d3.js verwendet. Ein Vorteil von Leaflet liegt darin, verschiedene Schichten dynamisch zu erzeugen und übereinander anzuordnen. Zusätzlich können Daten im GeoJSON Format interpretiert werden und als SVG Geometrien dargestellt werden. Ein Nachteil ist die fehlende Optimierung für Touchscreens wodurch die Verwendung auf Mobilen Geräten erschwert wird. Zudem fehlt Leaflet die Möglichkeit GeoJSON Daten in Form des Tile-Systems zu beziehen. Diese Funktionalität wurde dann durch das Plugin Leaflet GeoJSON Tile Layer hinzugefügt.

Leaflet GeoJSON Tile Layer

Leaflet GeoJSON Tile Layer ist ein Plugin für Leaflet, das die Möglichkeit bietet Daten in Form des Tile-Systems von einem Service zu beziehen. Diese Daten werden anschliessend von Leaflet interpretiert und als SVG Elemente gerendert. Eine klare Stärke dieses Plugins ist die einfache Verwendung. Dieses Plugin besitzt auch einige Schwächen in der Implementation. Beispielsweise werden bei einer Antwort mit dem HTTP Status Code 204 - No content Daten erwartet was jedoch dem Standard widerspricht. Hierfür und für das Erkennen von Duplikate wurde das Plugin angepasst.

AngularJS

AngularJS wurde verwendet um die Daten der Formulare, sowie alle Menüs zu verwalten. Eine grosse Stärke dieses Frameworks ist die umfangreiche Funktionalität. Eine dieser Funktionen ist die Modularisierung, die das Aufteilen des Codes in Module ermöglicht. Jedoch bringt diese Modularisierung auch Schwierigkeiten mit sich. Jedes Modul beinhaltet Methoden und Daten die sich im Scope des jeweiligen Moduls befinden. Damit zwei Module interagieren können muss der Scope geteilt werden. Diese Umsetzung war aufgrund der fehlenden oder mangelhaften Dokumentation sehr umständlich.

Play Framework

Das Play Framework bildet die Basis des SimMapEditors. Damit wurden die Views der Single Page Applikation erstellt. Eine Stärke des Play Frameworks ist sicherlich der eingebaute Netty Server der ein einfaches Deployment erlaubt. Dadurch kann die Applikation direkt als eigenständige Web Applikation gestartet werden und muss nicht in einen Web Server geladen werden. Über die funktionalen Stärken und Schwächen kann keine fundierte Aussage getroffen werden, da für die Logik hauptsächlich Javascript verwendet wurde.

Dropwizard

Dropwizard stellt ein Werkzeugkoffer mit verschiedenen Libraries zur Implementation von REST Service zur Verfügung. Zum einen enthält sie JAX-RS mit Jersey für die REST Funktionalitäten, JSON Jackson für die Bearbeitung von JSON Objekten und ein Netty Server für das eigenständige Ausführen der Applikation. Eine grosse Stärke ist sicherlich die einfache Anwendung. Ohne grossen Aufwand kann ein lauffähiger Webservice erstellt und deployed werden. Zudem ermöglichen die beiden Libraries JAX-RS in Verbindung mit Jersey die Implementation eines REST Services gemäss den Richardson Maturity Levels. Im Rahmen dieses Projektes wurde z.B. der Maturity Level 2 umgesetzt.

Allgemein

Im Rahmen dieses Projektes wurde folgende Technologien verwendet:

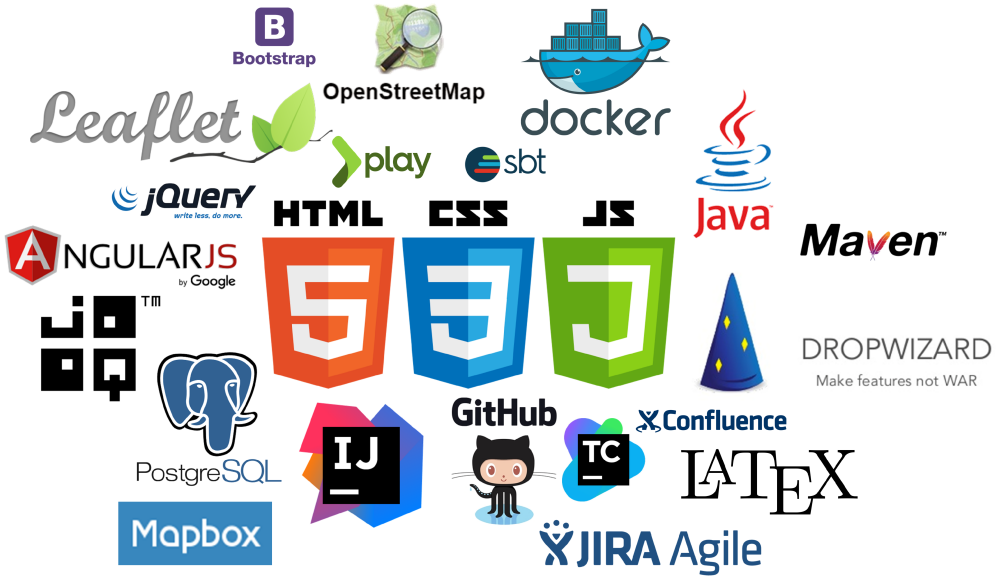


Abbildung 7.1: Technologien Cloud

Kapitel 8

Schlussfolgerung

8.1 Erkenntnisse

Die Arbeit „Verkehrsmodell-Fallstudien-Editor“ war eine sehr umfangreiche Arbeit mit vielen interessanten Herausforderungen. Sie beinhaltete die Bearbeitung von vielen verschiedenen neuen Themengebieten, in denen das Projektteam noch nicht sehr viel Erfahrung hatte. Dazu gehörten beispielsweise die Kartendarstellung im Browser, das Anzeigen von Strassen auf dieser Karte oder der effiziente Umgang mit grossen Datenmengen. Der Fokus dieser Arbeit lag zu Beginn auf dem Frontend. Das Backend durfte vereinfacht angenommen werden. Schnell wurde jedoch klar, dass ohne komplexeres Backend das effiziente Darstellen des Frontends nicht möglich ist. Aufgrund dessen wurde entschieden, dass trotzdem mehr Zeit in das Backend investiert wird. Diese Entscheidung erst ermöglichte das Entwickeln eines performanten Verkehrsmodell-Fallstudien-Editors. Das Backend vereinfacht anzunehmen machte zu Beginn Sinn. Jedoch wurde bereits bei ersten Tests ersichtlich, dass das Frontend Performanceprobleme aufwies, dies obwohl erst mit einer mittleren Datenmenge (ca. 60’000 Datensätze) gearbeitet wurde. Das vereinfachte Backend stellte den kompletten Datenstamm eines Verkehrsmodells für das Frontend zur Verfügung. Dies war dann bei einer grösseren Datenmenge wie z.B. der Schweiz (ca. 4 Millionen Datensätze) undenkbar. Daher wurde der Ansatz des QuadTile Algorithmus von OpenStreetMap [Ope16b] verwendet, der die Bereiche einer Karte aufteilt und eindeutig identifizierbar macht. Die Verwendung dieses Algorithmus setzt eine gewisse Komplexität des Backends voraus. Der Mehraufwand dieser Komplexität führte zu einer Veränderung des Funktionsumfangs dieser Arbeit. Das Feature „Neue Strasse zeichnen“ konnte auf Grund dieser Veränderung nur vereinfacht umgesetzt werden.
Zudem war das Rendern der unterschiedlichen Browser ein wichtiges Thema. Google Chrome kann besser mit grossen Datenmengen umgehen als z.B. Mozilla Firefox oder der Microsoft Edge Browser. Der Unterschied war zu Beginn in der Benutzung der Applikation deutlich zu sehen. Damit aber die Web Applikation auf allen Browsern genutzt werden kann, wurden Tests auf allen Browser durchgeführt und mit Hilfe des QuadTile Algorithmus die Datenmenge für das Rendern so klein wie möglich gehalten.

8.2 Backlog

Der Umfang dieser Arbeit beschränkte sich aus zeitlichen Gründen auf einen gewissen Umfang. Folgende Themen können bei einer Weiterentwicklung dieses Projektes noch umgesetzt werden.

Netzwerk-ID automatisch hochzählen

Zur Zeit ist es lediglich möglich, ein einziges Verkehrsmodell zu importieren. Im Rahmen dieses Projektes reichte dies vollkommen. Wird ein zusätzliches Modell importiert, wird für dieses Netzwerk wieder die ID 1 vergeben und alle Daten zu einem grossen Netzwerk zusammengeschlossen. Dies stellt kein Problem dar, solange sich die IDs der Nodes und Links der beiden Netzwerke unterscheiden.

Changeset basierend auf bestehendem Changeset

Zur Zeit kann ein leeres Changeset erstellt oder ein bestehendes Changeset bearbeitet werden, jedoch nicht ein neues Changeset basierend auf einem bestehendem Changeset erstellt werden. Dafür müsste lediglich das Changeset mit allen Link_Changes und Node_Changes kopiert werden und eine neue ID vergeben werden. Dies wäre eine nützliche Funktion für die Weiterentwicklung dieses Projekts.

Export Funktion von XML

Die Daten können über ein Formular in der Web Applikation importiert werden. Jedoch wurde die Funktionalität, die bearbeiteten Daten wieder zu exportieren, noch nicht implementiert. Das Datenmodell wurde so vorbereitet, dass alle Daten noch vorhanden sind und für den Export verwendet werden können.

Neue Strasse / Node zeichen

Das Zeichnen einer neuen Strasse oder eines neuen Nodes war zu Beginn ebenfalls Teil dieser Arbeit. Jedoch musste aus zeitlichen Gründen darauf verzichtet werden. Das Datenmodell ist auch hierfür vorbereitet. Um dies in Zukunft zu implementieren muss berücksichtigt werden, dass zur Zeit für jede empfangene Strasse geprüft wird, ob ein Eintrag im Changeset vorhanden ist. Ist dies der Fall, wird der Datensatz aus dem Changeset verwendet. Ansonsten wird der normale Datensatz gezeichnet. Eine neue Strasse ist jedoch noch nicht in den Stammdaten vorhanden und wird somit nie geprüft. Dadurch würde eine neue Strasse mit der jetzigen Logik nie gezeichnet werden. Diese Logik müsste auf der Seite des Clients so angepasst werden, dass neue Strassen im Changeset am Ende noch zusätzlich gezeichnet werden.

8.3 Persönliches Fazit

8.3.1 Dominik Heeb

Die Bachelorarbeit Verkehrsmodell-Fallstudien-Editor war von Anfang bis Schluss eine sehr spannende und vielseitige Arbeit. Das Themengebiet über die Implementation von Maps mit interaktiven Daten, war für uns komplettes Neuland. Im Team konnten wir uns jedoch sehr schnell in das Thema einleben. Die Implementation eines REST Services, welcher in einer skalierbaren Infrastruktur gehalten wird, war eines der Highlights der Arbeit für mich. Dabei konnte ich das erste Mal in einem Projekt mit Docker arbeiten, was für mich eine sehr gute und spannende Technologie darstellt. Weiter war es überaus spannend mit Datenmengen im Millionenbereich zu arbeiten und die Probleme kennenzulernen welche diese mit sich bringen. Das Team, eingeschlossen der Projektpartner Senozon AG, hat sehr gut harmoniert. Durch eine enge und koordinierte Zusammenarbeit, konnten alle Hauptrisiken in kürzester Zeit bewältigt werden. Persönlich konnte ich sehr viel von diesem Projekt für meine Zukunft profitieren.

8.3.2 Fabian Keller

Die Bachelorarbeit „Verkehrsmodell-Fallstudien-Editor“ war für mich eine sehr positive Erfahrung. Zu Beginn war die Thematik für beide Teammitgleider ziemlich neu und daher eine grosse Herausforderung. Dazu gehörten beispielsweise die Kartendarstellung im Browser, das Anzeigen von Strassen auf dieser Karte oder der effiziente Umgang mit grossen Datenmengen. Jedoch konnte wir uns durch grossen Einsatz und hohe Motivation ziemlich schnell in die Thematik einarbeiten und die grössten Risiken in kurzer Zeit beseitigen. Ein Highlight dieser Arbeit war sicherlich die erreichte Performance in der Darstellung der grossen Datenmenge auf der Karte. Zudem gefällt mir das User Interface Konzept der Web Applikation sehr. Durch die Vielseitigkeit dieser Arbeit konnte ich persönlich sehr viel profitieren und werde sicherlich in meiner zukünftigen Tätigkeit als Software Engineer einiges wiederverwenden können. Die Zusammenarbeit im Team sowie auch die Zusammenarbeit mit dem Projektpartner Senozon AG war einwandfrei und hat mir sehr viel Spass gemacht.

Abbildungsverzeichnis

Glossar

Dropwizard

Das Framework Dropwizard.io erlaubt eine einfache Implementation von REST-Full Web Services. Dropwizard verbindet verschiedene Technologien wie JAX-RS mit Jersey, Jackson, Netty Server usw., um ein Paket für den Entwickler zu erstellen, welches einfach implementiert und deployed werden kann.
(`http://www.dropwizard.io/`).

Play Framework

Das Play Framework ist ein Opensource Projekt mit dem Ziel, ein einfaches und leichtgewichtiges MVC Framework zu bieten. Play wird mit einem Netty Server ausgeliefert und muss daher nicht installiert werden (`https://www.playframework.com`).

PostgresSQL

PostgreSQL ist eine relationale Opensource Datenbank (`http://www.postgresql.org/`).

SQL LIKE Operator

Der SQL LIKE Operator wird für das Suchen eines spezifischen Patterns in einer Spalte verwendet (`http://www.w3schools.com/sql/sql_like.asp`).

Literaturverzeichnis

[D316] D3. Data-Driven Documents, 2016. https://d3js.org/.

[Fow16] Martin Fowler. Richardson Maturity Model, 2016. http://martinfowler.com/articles/richardsonMaturityModel.html.

[Geo16a] GeoJSON. GeoJSON, 2016. http://geojson.org/.

[Geo16b] TileLayer GeoJSON. Leaflet GeoJSON Tile Layer, 2016. https://github.com/glenrobertson/leaflet-tilelayer-geojson.

[Goo16a] Google. Angular JS, 2016. https://angularjs.org/.

[Goo16b] Google. Google Maps, 2016. https://www.google.ch/maps.

[Her16] Heroku. Increacing Application Performance with HTTP Cache Headers, 2016. https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers#conditional-requests.

[Lea16] Leaflet. Leaflet JS, 2016. http://leafletjs.com/.

[Map16] Mapbox. Mapbox, 2016. https://www.mapbox.com/.

[Ope16a] OpenStreetMap. ID Editor, 2016. http://ideditor.com/.

[Ope16b] OpenStreetMap. Quadtiles, 2016. http://wiki.openstreetmap.org/wiki/QuadTiles.

[W3.16] W3.org. Status Code Definition, 2016. https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

[Wik16] Wikipedia. HTTP ETag, 2016. https://de.wikipedia.org/wiki/HTTP_ETag.