HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

STUDIENARBEIT

RasterGeoConverter

Diego Etter, Céline Ott

Betreut von

Prof. Stefan Keller

Herbstsemester 2016

Abstract

Immer mehr Karten werden digital und frei im Web angeboten. Viele amtliche Behörden, welche ihre Karten online zu Verfügung stellen wollen, haben sich für die Rasterkarten entschieden. Technisch wurde dies in Form eines Dienstes namens „Web Map Service“ umgesetzt. Dieser Service lässt sich einfach betreiben, ermöglicht den Behörden die Karten zentral zu verwalten und somit immer auf dem neuesten Stand zu halten.
Leider unterstützen viele Planungsprogramme die Einbindung des Web Map Service nicht. Ein grosses Hindernis für Architekten, Bauingenieure und Planer, welche das Kartenmaterial für ihre Projekte und Umsetzungen gerne beziehen würden. Dazu müssten die Karten in einem nutzbaren Format von den Web Map Services geladen werden.

Genau diese Lösung bietet der RasterGeoConverter. Über ein Projekt wurde iterativ eine webbasierte Lösung entwickelt, welche mit der übersichtlichen Oberfläche den Bezug der gewünschten Karte ermöglicht.
Im Rumpf der Applikation ist eine Schnittstelle für die direkte Kommunikation zu den WMS implementiert und findet sich zudem eine Bildverarbeitungsbibliothek für die Erzeugung von GeoTIFF’s.
Das GeoTIFF ist eine Erweiterung des TIFF-Formates, welche eine verlustfreie Speicherung von Bilddaten ermöglicht und zudem die Georeferenz speichern kann.
Mit modernen Technologien wie Django und Angular 2 umgesetzt, ist der RasterGeoConverter gut wartbar und ist durch die Erweiterbarkeit auch für die Zukunft gerüstet.
Die Software wird als Docker-Container ausgeliefert und ist aus diesem Grund problemlos zu installieren und kann auf einfache Weise auf einem Linux-Server betrieben werden.

Increasingly more maps are digital and freely available over the world-wide-web. Many governmental authorities who want to offer their maps online use the raster maps technology. A web service named "Web Map Service", short WMS, makes this possible. The argument to use WMS is the simplicity of this service. WMS allows easy and central management of government maps. A disadvantage of using WMS is that a lot of planning software does not support the integration of this service. For architects, civil engineers and other clients, who would utilise the maps, this results in a considerable obstacle. To overcome this hindrance, the maps have to be made available as a graphical data format, loaded from the WMS.

The solution to this problem is provided by the RasterGeoConverter. Developed in an iterative project, the RasterGeoConverter is a web based solution with a clear and well-structured user interface. It is now possible to download a map easily.
The application itself provides an interface for the direct communication with web map services to order the map as well as a library to transform the images to the popular and comprehensive GeoTIFF graphics file format. GeoTIFF is an extended version of TIFF, which is a lossless format that also includes map projection, coordinate systems and everything else necessary to establish the exact spatial reference for the file.
The RasterGeoConverter includes modern technologies like Django and Angular 2, so its future maintainability and scalability is guaranteed. The software will be deployed as a Docker container, and will therefore be easy to install and run on a Linux server.

Management Summary

Ausgangslage

Immer mehr Karten, vor allem amtliche, werden digital und frei als Open Government Data angeboten. Technisch geschieht dies in Form eines Dienstes namens «Web Map Service», kurz WMS. Über WMS-Dienste bereitgestellte Karten werden direkt ins eigene System eingebunden. Sie sind damit immer aktuell.

Leider unterstützen viele Planungsprogramme keine solche Karten-Dienste. Dies ist ein grosser Nachteil für Architekten, Bauingenieure und Planer, welche die Karten für ihre Vorhaben gerne beziehen würden. Sie setzen bevorzugt das Kartenmaterial als Grafikdatei ein, lokal und im gängigen Rasterformat GeoTIFF. Der Bezug einer Karte über einen WMS benötigt eine Internetadresse und spezifische Parameter. Dies verlangt Spezialkenntnisse, die bei Benutzer oft fehlen. Der RasterGeoConverter möchte einen Beitrag zur Verbesserung dieser Situation leisten.

Ziel

Mit dem Projekt RasterGeoConverter soll eine webbasierte Plattform für den einfachen Kartenbezug geschaffen werden. Über ein übersichtliches Interface soll der Benutzer die Möglichkeit erhalten direkt mit dem gewünschten Web Map Service zu kommunizieren. Dazu bietet der RasterGeoConverter eine Kartenansicht und Eingabefelder um einen gewünschten Ausschnitt innerhalb der Karte zu wählen. Als Resultat soll der Benutzer die Karte im GeoTIFF-Format beziehen können.

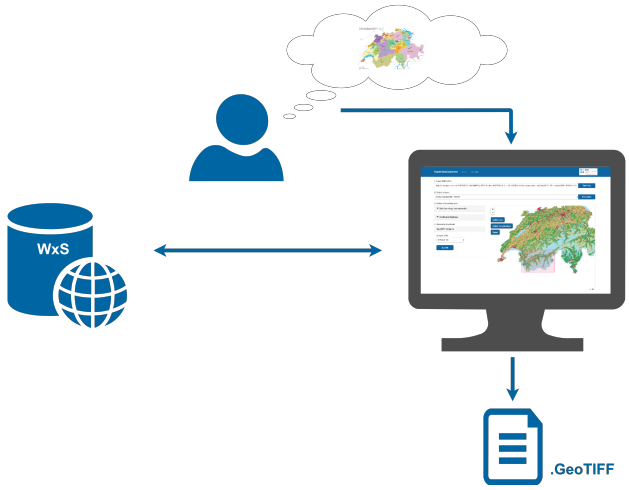


Abbildung 1: Vision RasterGeoConverter

Ergebnisse

Das Resultat dieser Arbeit ist eine lauffähige, moderne Webapplikation mit einer übersichtlichen grafischen Benutzerschnittstelle. Der Benutzer wird schrittweise durch die Anwendung geführt, beginnend mit der Eingabe der Internetadresse des WMS bis zum Download des GeoTIFFs.

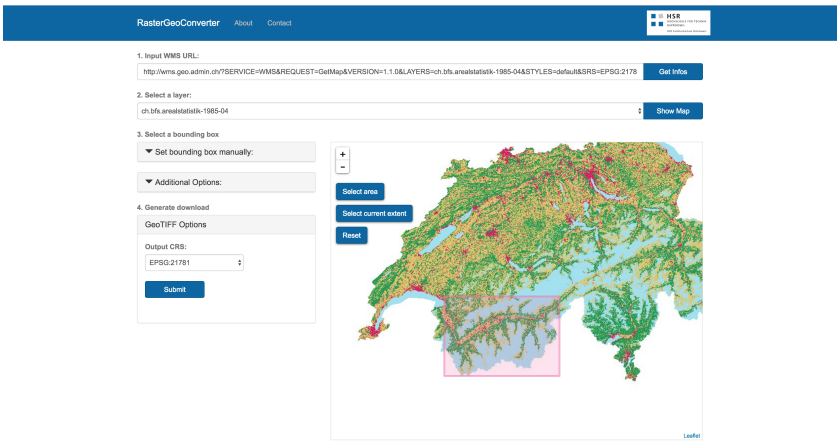


Abbildung 2: Bedienoberfläche RasterGeoConverter

Der RasterGeoConverter wurde durch eine REST-Schnittstelle in zwei Teilbereiche aufgeteilt, um die Vorzüge aus verschiedenen Technologien auszuschöpfen. Das Frontend wurde mit den Javascript-Frameworks Angular (Programmlogik) und Bootstrap (Layout) umgesetzt und bietet durch das Leaflet-Plugin (Kartenansicht) eine umfassende Bedienoberfläche an. Für die Kommunikation mit dem WMS und die Datenverwaltung wurde das Backend mit Python und dem Django-Framework realisiert. Python ist eine verbreitete Programmier- und Skriptsprache mit vielen Open Source-Bibliotheken. Das Django-Framework bietet mit seinen Funktionalitäten eine gute Basis für das Grundgerüst des RasterGeoConverters.

Die gewählten Technologien und die entsprechende Architektur bietet die Grundlage für eine gute Wartbarkeit und Erweiterbarkeit der Applikation.

Ausblick

Der RasterGeoConverter bietet im momentanen Zustand nur die Grundfunktionalitäten an. Aus Zeitgründen konnten natürlich nur die nötigsten Anforderungen umgesetzt werden. Die Ausbaumöglichkeiten sind daher vielseitig und mit den eingesetzten Technologien ohne weiteres möglich.

Weitere Web Services für Karten Momentan wurde ausschliesslich der Kartenbezug von Web Map Services (kurz WMS) umgesetzt. Es sind aber auch andere Rasterkarten-Services verbreitet. Darunter fallen der Web Map Tiles Service und/oder der Web Coverage Service. In einem weiteren Schritt können auch diese Services eingebunden werden.

Map Sharing Geladene Karten könnten im Hintergrund gespeichert und über Links geteilt werden.

Top-Ten-Liste Die viel genutzten Services könnten als Vorschlag aufgelistet werden, so dass die Benutzer über eine Auswahl direkt zum gewünschten Service finden.

Erweiterte Eingaben Der Kartenbezug wurde auf die einfachsten Eingaben beschränkt. Funktionalitäten wie Massstab, Formate und/oder eine gewünschte Auflösung des Bildes wären ebenfalls mögliche Kandidaten für die Erweiterung des RasterGeoConverters.

Zusatz Parameter Die Services bieten neben den Standardisierten Parameter auch anbieterspezifische Parameter an. Der RasterGeoConverter könnte über eine erweiterte Option diese Parameter entgegennehmen.

In bestehende Lösung einbinden Für den Bezug von Vektorkarten wurde schon eine Webapplikation umgesetzt. Der RasterGeoConverter könnte dieser Applikation hinzugefügt werden und so die Funktionalität dieser Plattform erweitern.

Aufgabenstellung

RasterGeoConverter

- Studienarbeit im Herbstsemester 2016/2017
- Autor/in: Diego Etter und Céline Ott
- Betreuer: Prof. Stefan Keller, Institut für Software, HSR
- Industriepartner: -

Ausgangslage

Bauingenieure, Planungsbüros und Raumplaner benötigen für viele Projekte Planungsmaterial. Darunter fallen auch Karten welche über einen WMS bezogen werden können. Diese bieten das Kartenmaterial als Rasterkarten an. Um die Karten effizient nutzen zu können (z.B. in CAD's) werden GeoTIFFs benötigt. Leider besitzen nicht alle Planer das nötige Wissen um solches Kartenmaterial zu beziehen. Auf dem Markt sind solche Services noch nicht vorhanden.

Aufgabenstellung

Über den RasterGeoConverter sollen die Benutzer GeoTiffs über einen WMS möglichst simple beziehen können. Dies soll ohne grosse Vorkenntnisse der WMS-API's geschehen können.

- Die Benutzer haben einen Link zu einem WMS
- Der Benutzer benötigt einen definierten Kartenausschnitt als GeoTIFF
- Über den RasterGeoConverter kann der Benutzer den Kartenausschnitt wählen, die nötigen Kriterien wählen und das Geotiff erzeugen sowie downloaden.

Ziele

- Funktionierende Webapplikation (Frontend und Backend)
- WMS unterstützt, optional WMTS und/oder WCS.

Lieferobjekte

1. Dokumentation, inkl. Textabstract (zusätzlich englisch), Management Summary (deutsch), technischer Bericht und Software Engineering-Projekt (deutsch); Anhänge (Literaturverzeichnis, CD-Inhalt).
2. Evaluation von Backend-Framework und Frontend-Framework.
3. Sowie vom Studiengang geforderten bzw. empfohlenen Lieferobjekte: Poster (nur digital), Broschüren-Abstract, optional Kurzvideo.
4. Software (englisch).

Vorgaben/Rahmenbedingungen

- Webapplikation; Server: Python, Django. Datenbank: Bevorzugt PostgreSQL. Client: -
- Daten: öffentlich zugängliche Webdienste.
- SW-Infrastruktur mit kontinuierlichem Testen; optional/empfohlen: Docker.
- Wichtige nichtfunktionale Anforderungen sind die HSR Corporate Identity (ggf. vom IFS), hohe Usability; ein responsives GUI sowie das Verarbeiten von mind. 10 Nutzern, die quasi-gleichzeitig konvertieren.

Vorgehen und Arbeitsweise: Die Studierenden wählen nach Rücksprache ein Vorgehensmodell zur Softwareentwicklung. Es gibt wöchentliche Meetings mit vorbereiteten Unterlagen; wobei Ausnahmen vereinbart werden können.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Dokumentation

Die Dokumentation ist auf Deutsch geschrieben wo nicht anders vermerkt und ist in den Lieferobjekten erwähnt.

Weitere Angaben:

- 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 gebunden (1 Ex.), inkl. einer beschrifteten CD (plus 1 Ex. für den Studiengang).
- Alle Dokumente und Quellen der erstellten Software auf CDs.

Bewertung

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).

Weitere Beteiligte

Mitarbeiter des Institut für Software der HSR. HSR-Dozent „Digitale Medien“ (Studiengang Bau). HSR-Dozent „Verarbeitung und Darstellung von digitalen Daten im Bauingenieurwesen (Studiengang Landschaftsarchitektur). GIS-Stellen von Stadt und Kanton Zürich.

Inhaltsverzeichnis

I Technischer Report 1

1 Einführung 2
- 1.1 Vision . . . 2
- 1.2 Ziele . . . 2
- 1.3 Rahmenbedingungen . . . 3
- 1.4 Vorgehen . . . 3
- 1.5 Stand der Technik . . . 3

2 Evaluation 4
- 2.1 Frontend . . . 4
  - 2.1.1 Karten Darstellung . . . 4
  - 2.1.2 Frameworks . . . 5
- 2.2 Backend . . . 10
  - 2.2.1 Programmiersprachen . . . 10
  - 2.2.2 Entscheid Programmiersprache . . . 11
  - 2.2.3 Frameworks und Bibliotheken . . . 12
  - 2.2.4 Frameworks . . . 12
  - 2.2.5 Entscheid Frameworks . . . 13
  - 2.2.6 Bibliotheken . . . 17

3 Eingesetzte Technologien 18
- 3.1 Angular 2 . . . 18
  - 3.1.1 Eingesetzte Packages . . . 18
- 3.2 Django . . . 19
  - 3.2.1 Eingesetzte Plugins . . . 19

4 Umsetzungskonzept 22
- 4.1 WxS ansprechen . . . 22
  - 4.1.1 WxS Inhalt . . . 22
  - 4.1.2 Kartenabruf . . . 22
- 4.2 GeoTIFF erzeugen . . . 22

5 Resultate, Bewertung und Ausblick 23
- 5.1 Zielerreichung . . . 23
- 5.2 Weiterentwicklung . . . 23
  - 5.2.1 WMTS . . . 24
  - 5.2.2 WCS . . . 24
  - 5.2.3 Datenhaltung . . . 24
  - 5.2.4 Karte teilen . . . 24
  - 5.2.5 Weitere Parameter . . . 25
  - 5.2.6 Vendor specific parameters . . . 25
- 5.3 Persönliche Berichte . . . 25

5.3.1 Persönlicher Bericht von Diego Etter . . . 25
5.3.2 Persönlicher Bericht von Céline Ott . . . 26

II Softwareprojekt Dokumentation 27

6 Anforderungsspezifikation 28
6.1 Funktionale Anforderungen . . . 28
6.1.1 Ziele . . . 28
6.1.2 Optionale Ziele . . . 28
6.2 Use Cases . . . 28
6.2.1 Diagramm . . . 28
6.2.2 Akteure . . . 29
6.2.3 Formuliert . . . 30
6.3 Nicht funktionale Anforderungen . . . 31
6.3.1 Sprachen . . . 31
6.3.2 Technologien . . . 31
6.3.3 Daten . . . 31
6.3.4 Design . . . 31
6.3.5 Usability . . . 31

7 Designkonzept 32
7.1 Anforderungen . . . 32
7.2 Usability . . . 32
7.3 Wireframes . . . 32
7.4 Prototyp . . . 33
7.5 Designentwicklung . . . 33
7.5.1 Weiteres Bedienelement Layerselect . . . 34
7.6 Layout . . . 35
7.7 Farbwahl . . . 35
7.8 Schriftart . . . 36
7.9 Umsetzung des Designs . . . 36

8 Software Architektur 37
8.1 Architektur . . . 37
8.1.1 Frontend . . . 38
8.1.2 Backend . . . 38
8.1.3 Data Access . . . 38
8.1.4 WxS . . . 38
8.2 Systemübersicht . . . 38
8.2.1 frontend Container . . . 38
8.2.2 backend Container . . . 39
8.2.3 DB Container . . . 39
8.2.4 GDAL Container . . . 39
8.3 Paketdiagramm . . . 40
8.3.1 Frontend . . . 40
8.3.2 Backend . . . 41
8.3.3 Paket: config . . . 43
8.3.4 Bibliothek-Pakete . . . 43

9 Implementation 44
9.1 Frontend . . . 44
9.1.1 Übersicht der Projektstruktur . . . 44
9.1.2 Applikationsstruktur . . . 44
9.1.3 Models . . . 45
9.1.4 Komponenten . . . 46
9.1.5 Service . . . 47

9.1.6 Testing . . . 48
9.1.7 Installation . . . 48
9.2 Backend . . . 49
9.2.1 Übersicht der Projektstruktur . . . 49
9.2.2 project . . . 50
9.2.3 src . . . 50
9.2.4 src.rasterconverter . . . 50
9.2.5 srs/mapservices . . . 57

10 Weiterentwicklung **61**

11 Projektmanagement **62**
11.1 Rollen und Verantwortlichkeiten . . . 62
11.1.1 Prof. Stefan Keller . . . 62
11.1.2 Nicola Jordan . . . 62
11.1.3 Céline Ott . . . 62
11.1.4 Diego Etter . . . 62
11.2 Prozessmodell . . . 62
11.2.1 RUP Phasen . . . 63
11.3 Infrastruktur . . . 63
11.3.1 Versionierung . . . 63
11.3.2 Entwicklungswerkzeuge . . . 64
11.4 Projektplan . . . 64
11.4.1 Aufwandschätzung . . . 64
11.4.2 Phasen / Iteration . . . 64
11.4.3 Meilensteine . . . 66
11.4.4 Auswertung . . . 67
11.5 Zeitplan . . . 68
11.5.1 Phasenplanung . . . 68
11.5.2 Tickets . . . 69
11.5.3 Nichterfüllte Tickets . . . 72
11.5.4 Meilensteine . . . 72
11.6 Risikoanalyse . . . 73
11.6.1 Erläuterungen . . . 73
11.6.2 Risikotabelle . . . 73
11.6.3 Risikomatrix . . . 74

III Appendix **75**

Glossar **77**

A Abgabestruktur **78**

B Eigenständigkeitserklärung **79**

Abbildungsverzeichnis

1 Vision RasterGeoConverter i
2 Bedienoberfläche RasterGeoConverter ii

2.1 Leaflet Logo (LeafletJS) 4
2.2 Mapbox Logo (Werobotics) 5
2.3 OpenLayers Logo (Geospacial Blogs) 5
2.4 Wertungstabelle Gesamt (Angular 2/Vue.js/jQuery) 6
2.5 Diagramm Gesamt (Angular 2/Vue.js/jQuery) 7
2.6 Diagramm Umfang (Angular 2/Vue.js/jQuery) 8
2.7 Diagramm Kartenintegration (Angular 2/Vue.js/jQuery) 8
2.8 Diagramm Community (Angular 2/Vue.js/jQuery) 9
2.9 Diagramm Learning (Angular 2/Vue.js/jQuery) 10
2.10 Wertungstabelle Java/Python 11
2.11 Wertungsdiagramm Gesamt Java/Python 11
2.12 Wertungsdiagramm Übersicht Java/Python 12
2.13 Wertungstabelle Gesamt (Django/Flask/Pyramid) 14
2.14 Diagramm Gesamt (Django/Flask/Pyramid) 14
2.15 Diagramm Umfang (Django/Flask/Pyramid) 15
2.16 Diagramm Community (Django/Flask/Pyramid) 15
2.17 Diagramm Learning (Django/Flask/Pyramid) 16

6.1 Use Case Diagramm 29

7.1 Erste rudimentäre Wireframes 33
7.2 Erster Wireframe Prototyp 33
7.3 Erster funktionaler Prototyp 34
7.4 Erster funktionaler Prototyp nach klick auf "Go!" 34
7.5 Zweite Variante des Prototypen 34
7.6 Prototyp mit Layerauswahl 35
7.7 Finaler Prototyp 35
7.8 HSR Farbtabelle 36

8.1 Architektur-Übersicht RasterGeoConverter 37
8.2 Verteilungsdiagramm RasterGeoConverter 39
8.3 Packetdiagramm Frontend 40
8.4 Paketdiagramm Backend 41
8.5 Paketdiagramm backend/project 41
8.6 Paketdiagramm backend/src 42

9.1 Projektstruktur Frontend 45
9.2 Applikationsstruktur Frontend 45
9.3 Models Frontend 46
9.4 URL Input Komponente 46
9.5 Layer Select Komponente 46
9.6 Map Komponente 47

9.7 Download Komponente . . . 47
9.8 Implementation Übersicht . . . 49
9.9 Klassendiagramm views . . . 51
9.10 REST-API der ServiceParamsView . . . 51
9.11 REST-API der WxsInfoView . . . 52
9.12 REST-API der WxsMapCreatorView . . . 53
9.13 REST-API der CRSTransform . . . 55
9.14 Übersicht Models . . . 56
9.15 db-handler Klassendiagramm . . . 57
9.16 map-service-provider Klassendiagramm . . . 57
9.17 map-services Klassendiagramm . . . 58
9.18 map-converter Klassendiagramm . . . 58
9.19 relation-calculator Klassendiagramm . . . 59
9.20 wms-queries Klassendiagramm . . . 60
9.21 map-service-test Klassendiagramm . . . 60

11.1 Stundendurchschnitt pro Person . . . 67
11.2 Inceptionphase . . . 68
11.3 Elaborationsphase . . . 68
11.4 Constructionsphase . . . 69
11.5 Transitionphase . . . 69
11.6 Tickets in der Inception 1 . . . 69
11.7 Tickets in der Inception 2 . . . 70
11.8 Tickets in der Elaboration 1 . . . 70
11.9 Tickets in der Elaboration 2 . . . 70
11.10 Tickets in der Construction 1 . . . 71
11.11 Tickets in der Construction 2 . . . 71
11.12 Tickets in der Construction 3 . . . 72
11.13 Nichterfüllte Tickets . . . 72
11.14 Meilensteine Ist . . . 72
11.15 Mögliche Risiken . . . 73
11.16 Risikomatrix . . . 74

Tabellenverzeichnis

11.1 Entwicklungswerkzeuge im Einsatz 64
11.2 Aufwandschätzung RasterGeoConverter 64

Listings

9.1 Konsole: Alle Packages installieren 48
9.2 Konsole: Development Server starten 48
9.3 Konsole: Projekt builden 49
9.4 Konsole: Datenbank migrieren 50
9.5 Konsole: Datenbank erzeugen 50
9.6 Konsole: Testserver ausführen 50
9.7 Inhalt apps 50
9.8 Get-Request: Resultat Parameteranfrage 52
9.9 Get-Request: Resultat Serviceinformationen 53
9.10 Get-Request: Resultat ServiceTest 54
9.11 relation-calculator: Berechnung der maximalen Auflösung 59

Teil I

Technischer Report

Kapitel 1

Einführung

Viele amtliche Behörden stellen ihre Karten online auf Web Services für Karten, kurz Web Map Service (WMS), zu Verfügung. Diese WMS können einfach eingerichtet werden und die Karten sind somit zentral verwaltet und immer auf dem aktuellsten Stand. Architekten, Bauingenieure und Planer sind natürlich für ihre Projekte an den Karten interessiert und würden diese gerne in die eigene Software importieren und nutzen. Leider unterstützen die Planungssoftwares selten die Einbindung eines WMS-Dienstes. Stattdessen werden gerne Grafikformate zurückgegriffen. Im Falle von Karteninformationen wird GeoTIFF bevorzugt.
Der Bezug eines Bildes aus dem WMS, sowie Erstellen eines GeoTIFF, ist jedoch für viele Planer ein Hindernis.

1.1 Vision

Ein Bauingenieur möchte für sein Projekt das Kartenmaterial aus dem behördlichen WMS beziehen und diese für seine Planungen nutzen. Um die Karte in seine Planungssoftware zu importieren, benutzt der Bauingenieur gerne das TIFF-Format. Er stellt sich nun die Frage: Ẅie soll ich die Karte aus dem WMS der Behörde downloaden?ünd zudem Ïch würde gerne nur diesen Ausschnitt haben, nicht die gesamte Karte.̈ Da nicht alle WMS die Karte als GeoTIFF zu Verfügung stellen, ist das natürlich die nächste Frage. Genau diesen Anforderungen möchte der RasterGeoConverter entgegen kommen. Der RasterGeoConverter soll wie die WMS über das Internet erreichbar sein. Zudem soll über ein übersichtliches Interface die Informationen der WMS geladen, und das Bild oder der Ausschnitt davon gewählt werden können.

1.2 Ziele

Die Hauptziele des Projektes sind:

- Eine funktionierende Webapplikation (Frontend und Backend)
- WMS muss unterstützt sein

Es wurde folgende optionale Ziele definiert:

- WMTS unterstützen
- WCS unterstützen
- Bild-Sharing
- Docker-Container für die einfache Installation

1.3 Rahmenbedingungen

Folgende Rahmenbedingungen wurden durch den Betreuer vorgegeben:

- Programmiersprache Python
- Datenbank bevorzugt PostgreSQL
- Wichtige nicht funktionale Anforderungen sind die HSR Corporate Identity
- responsive Graphical User Interface (GUI)
- lauffähiger Docker-Container

1.4 Vorgehen

Das Vorgehen richtete sich bei diesem Projekt vor allem nach den RUP-Phasen. In der Inception-Phase kümmerte sich das Projektteam hauptsächlich um die Einrichtung der Arbeitsumgebung, die Rahmenbedingungen und Anforderungen. Zudem wurde eine Risikoanalyse durchgeführt. In der Elaboration-Phase wurde die einzusetzende Technologie elaboriert und angeeignet. Des Weiteren wurden Funktionsprototypen erstellt als Massnahme gegen Risiken und um sich ein Bild der Funktionen zu machen. Der RasterGeoConverter wurde schlussendlich in der Construction-Phase zu seiner jetzigen Form iterativ implementiert. Während der wöchentlichen Besprechungen mit dem Betreuerteam glichen wir den Stand ab und kommunizierten das weitere Vorgehen. Schlussendlich wurde die ganze Arbeit dokumentiert.

1.5 Stand der Technik

Momentan gibt es Teillösungen für Vektorkarten, welche als Webapplikation zur Verfügung stehen, an denen wir uns orientieren können. Dennoch, für die Rasterkarten gibt es momentan keine laufenden Applikationen, an denen wir uns direkt orientieren können.

Kapitel 2

Evaluation

2.1 Frontend

Die Wahl der Technologie des Frontends ist ziemlich frei. Es sollte auf das Vorwissen geachtet werden und dass die Technologie Open Source ist. In den nicht-funktionalen Anforderungen des Projekts ist auch noch festgehalten, dass auf gute Usability im Frontend geachtet werden soll. Ausserdem sollte etwas Rücksicht darauf genommen werden, dass das Frontend später weiterentwickelt werden soll.

Ein sehr wichtiger Teil des Frontends ist die Mapview, in welcher ein Kartenausschnitt ausgewählt werden kann. Es muss somit darauf geachtet werden, dass im Frontend eine Technologie verwendet wird, in welcher die Mapview integriert werden kann. Anhand dieser Anregungen wurde der Entscheid für das Frontend getroffen.

2.1.1 Karten Darstellung

Nach einer ausgiebigen Recherche wurden drei geeignete Libraries für die Mapview gefunden. Allesamt sind JavaScript-Libraries: Mapbox, Leaflet und OpenLayers.

Leaflet

Abbildung 2.1: Leaflet Logo (LeafletJS)

Leaflet ist eine Open Source Javascript-Library, mit welcher Karten dargestellt werden können. Es werden WMS, Web Map Tile Service (WMTS) und GeoJSON sowie Bildüberlagerungen unterstützt. Durch Plugins können noch weitere Typen von Ebenen unterstützt werden wie KML, CSV, WKT, GPX etc.

Lizenz: BSD-Lizenz

Mapbox

Abbildung 2.2: Mapbox Logo (Werobotics)

Mapbox ist eine Firma, welche Onlinekarten für grössere Unternehmen zur Verfügung stellt. Sie leisten einen bedeutenden Beitrag zur Entwicklung von Leaflet. Mapbox.js ist eine JavaScript-Library die Leaflet.js erweitert. Da Mapbox als Plugin von Leaflet.js implementiert ist, muss für die Verwendung dann auch Leaflet verwendet werden.

OpenLayers

Abbildung 2.3: OpenLayers Logo (Geospacial Blogs)

OpenLayers ist eine JavaScript-Library mit welcher Geodaten im Browser dargestellt werden können. Openlayers unterstützt die folgenden Geodaten: Web Feature Service (WFS) und WMS.

Lizenz: BSD-Lizenz

Fazit

Nach genauerem Betrachten der Libraries war ziemlich schnell klar, dass Leaflet.js verwendet wird. Alle drei Libraries unterstützen zwar WMS, doch nur Leaflet und Mapbox unterstützen auch noch WMTS. Da WMTS ein optionales Ziel unserer Arbeit ist, ist es wichtig, dass die Möglichkeit besteht, WMTS einzubinden. Da Mapbox.js auf Leaflet basiert, scheint es sinnvoller zu sein Leaflet zu verwenden. Ein grosser Einfluss für diesen Entscheid war auch, dass im Geometa Lab bereits Anwendungen mit Leaflet gebaut wurden und somit bereits positive Erfahrungen damit gesammelt wurden. Da unsere Applikation später auch vom Geometa Lab unterhalten wird, ist es sinnvoll eine bereits bekannte Technologie zu verwenden.

2.1.2 Frameworks

Für die Wahl des Frontend Frameworks sind keine wirklichen Vorgaben gesetzt. Da nun klar ist, mit welcher Library gearbeitet werden soll, muss das Framework natürlich mit der Library für die Kartenansicht kompatibel sein. Auch auf die Weiterentwicklung sollte natürlich Rücksicht genommen werden. Da das Backend in Python implementiert wird, wäre ein naher Gedanke gewesen das Django oder Flask Framework zu verwenden, da beide Frameworks auch eine Template Engine beinhalten. Nach den Diskussionen an den Betreuermeetings wurde klar, dass das Userinterface möglichst gut bedienbar sein soll. Für gute Bedienbarkeit gehört auch eine gute Reaktionszeit und flüssiger Webseitenlauf dazu. Damit die Website möglichst flüssig und ohne ständiges neu Laden läuft, eignen sich deshalb JavaScript Frameworks wie Vue.js und Angular 2 besser wie Django oder Flask.

Entscheidung

Den Entscheid des Frontend Frameworks haben wir aufgrund der folgenden Kriterien getroffen:

- Umfang: RESTful, Third Party Modules, Two Way Databinding, Form Validation, Installation, Testing, Performance, Benutzerinteraktion
- Kartenintegration: Beispiele, Unterstützung in Foren
- Community: Popularität, Foren, Zukunft, Tutorials
- Learning: Übersicht, Umfang

Kriterien		Kandidaten					
		Angular2		Vue.js		jQuery	
Beschreibung	**Gew[1-3]**	**Pts [1-5]**	**Result**	**Pts [1-5]**	**Result**	**Pts [1-5]**	**Result**
Umfang							
Benutzerinteraktion	3	5	15	5	15	4	12
Performance	3	5	15	4	12	4	12
Form Validation	2	4	8	3	6	2	4
Two Way Databinding	2	5	10	5	10	1	2
Installation	1	2	2	2	2	5	5
RESTful API	3	5	15	4	12	1	3
Third Party Modules	2	5	10	3	6	4	8
Testing	2	4	8	3	6	2	4
Kartenintegration							
Beispiele	2	4	8	2	4	5	10
Unterstützung in Foren	3	5	15	3	9	5	15
Community							
Popularität	2	5	10	3	6	5	10
Tutorials	2	5	10	3	6	5	10
Foren	2	5	10	4	8	5	10
Zukunft	2	5	10	4	8	4	8
Learning							
Simpel	2	2	4	2	4	4	8
Umfang	2	2	4	2	4	4	8
Resultate		**154**		**118**		**129**	

Abbildung 2.4: Wertungstabelle Gesamt (Angular 2/Vue.js/jQuery)

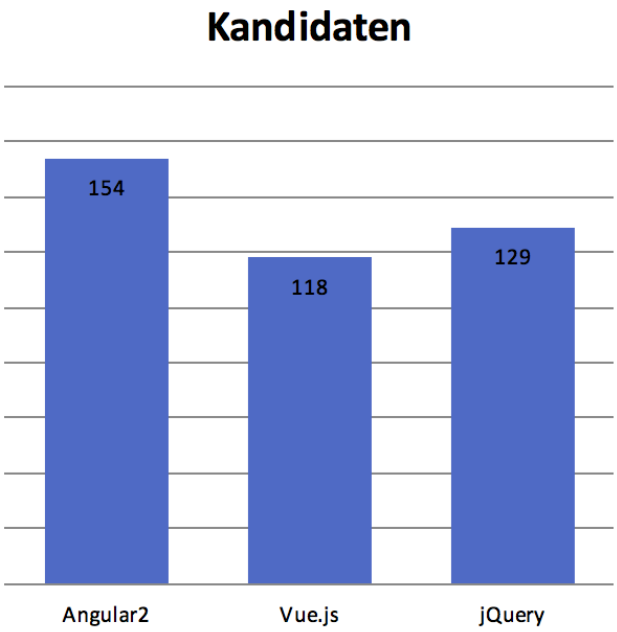


Abbildung 2.5: Diagramm Gesamt (Angular 2/Vue.js/jQuery)

Umfang

Benutzerinteraktion
: Der Benutzer soll möglichst eine flüssige Applikation haben, ohne ständig die Seite neu zu laden. Angular 2 und Vue.js erfüllen diesen Punkt speziell gut.

Performance
: Die Ladezeiten der Website sollten möglichst kurz sein, weshalb nach Benchmarks gesucht wurde. Im Blog von Stefan Krause [4] sind einige Benchmarks ersichtlich. Dabei sticht klar heraus, dass Angular 2 bessere Performance als Vue.js hat.

Form Validation
: Die Formulare müssen auf richtigen Input überprüft werden. Dabei bietet Angular 2 bereits gute Funktionalitäten. Bei jQuery muss ein zusätzliches Skript integriert werden.

Two Way Databinding
: Dadurch, dass die Daten in der ganzen Applikation immer übereinstimmen müssen, ist es wichtig, ein einfaches Two Way Databinding zu haben. Angular 2 und Vue.js bieten dies beide von Haus aus. Bei jQuery müssen diese Bindings selbst programmiert werden.

Installation
: Die erstmalige Installation ist bei jQuery klar am einfachsten, da lediglich das Script-Tag in den HTML Code integriert werden muss. Für Vue.js und Angular 2 gibt es eine CLI, welche die Installation vereinfacht. Trotz allem ist die Installation wesentlich schwieriger.

RESTful API
: Das Frontend kommuniziert mit dem Backend über die REST-Schnittstelle. Dementsprechend ist die Einbindung sehr wichtig. Angular 2 bietet die umfangreichsten Funktionen für eine REST-Schnittstelle. Wobei jQuery gar keine Funktionen für REST bietet.

Third Party Modules
: Falls die Frameworks gewisse Funktionen nicht besitzen, können diese mit Bibliotheken ergänzt werden. Angular 2 bietet hier am meisten Unterstützung.

Testing
: Die Funktionen sollen auch getestet werden können. Am einfachsten geht dies mit Angular 2, da dort bereits ein Testing integriert ist.

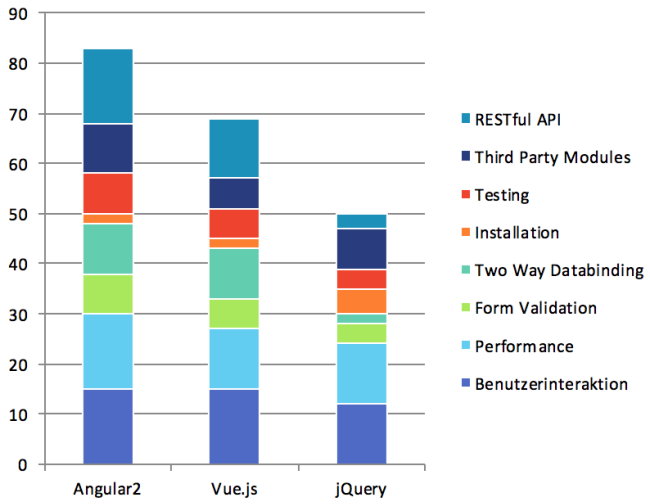


Abbildung 2.6: Diagramm Umfang (Angular 2/Vue.js/jQuery)

Kartenintegration

Beispiele
: Es ist einfacher eine korrekte Kartenintegration zu machen, wenn bereits Beispiele mit derselben Technologie bestehen. Wenn mit jQuery gearbeitet wird, kann die Kartenintegration ganz normal gemacht werden, wie es auch auf der Leaflet Seite gezeigt wird. Für die Integration in Angular 2 gibt es einige Beispiele.

Unterstützung in Foren
: Bei Problemen ist es immer hilfreich, in Foren nach Lösungen zu suchen. Es ist deshalb wichtig, dass es bereits andere Personen gibt, welche dieselbe Kombination von Kartenintegration und Framework verwendet haben.

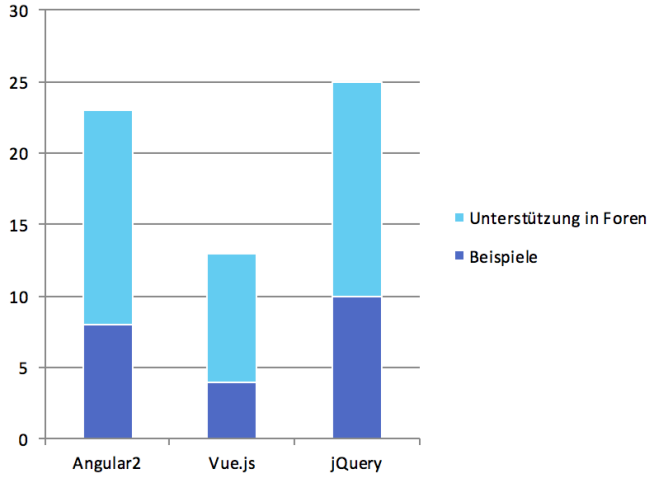


Abbildung 2.7: Diagramm Kartenintegration (Angular 2/Vue.js/jQuery)

Community

Popularität
: Bei einem populären Framework werden stetig Updates geliefert und die Community trägt aktiv zu Erneuerungen bei. Obschon der Release von Angular 2 noch nicht so lange her ist, ist es bereits sehr populär. jQuery war bereits vor Jahren schon sehr populär. Einzig Vue.js schein noch etwas ein Nischenprodukt zu sein.

Tutorials
: Damit der Einstieg in ein Framework einfacher geht, ist es wichtig, ein Tutorial dazu zu haben. Angular 2 besitzt ein sehr umfangreiches Tutorial, welches jedoch sehr einfach zu verstehen ist und einen sehr guten Eindruck des Frameworks liefert. Das Tutorial von Vue.js fällt eher etwas kurz aus. Für jQuery gibt es jede Menge Tutorials.

Foren
: Wenn man bei einem Problem nicht weiterkommt und die Dokumentation auch nicht weiterhilft, so ist es wichtig ein Forum zu haben, wo man eine ganze Community damit ansprechen kann.

Zukunft
: Da das Frontend auch in Zukunft noch weiterentwickelt werden soll, ist es wichtig, dass auch in Zukunft Unterstützung und Erneuerungen für dieses Framework erscheinen. Obwohl alle Frameworks eine grosse Community besitzen und somit bestimmt auch weiterentwickelt werden, so kann man sagen, dass Angular 2 bestimmt am meisten Zukunft hat, da es von Google unterstützt wird.

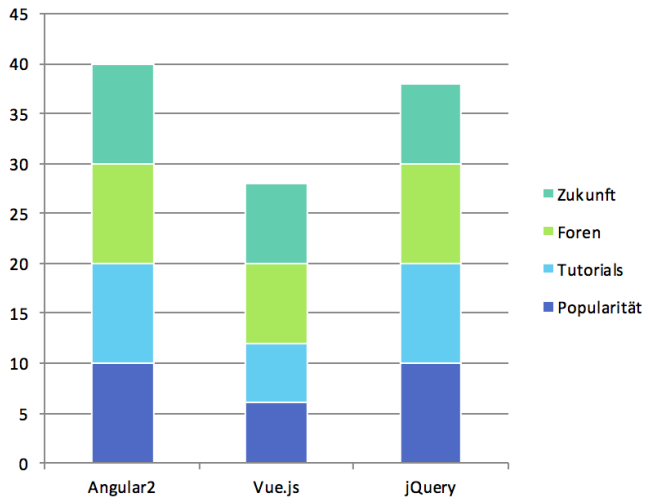


Abbildung 2.8: Diagramm Community (Angular 2/Vue.js/jQuery)

Learning

Simpel
: Sowohl Angular 2 als auch Vue.js benötigen einige Zeit, um sich einzuarbeiten. Beide Frameworks besitzen eine eigene Denkweise und in diese muss man sich zuerst einarbeiten.

Umfang
: Das Framework sollte möglichst viele Funktionen abdecken, jedoch bedeuten mehr Funktionen, aber auch dass mehr Bearbeitungszeit benötigt wird. Der Lernaufwand für Angular 2 und Vue.js ist deutlich höher als für jQuery.

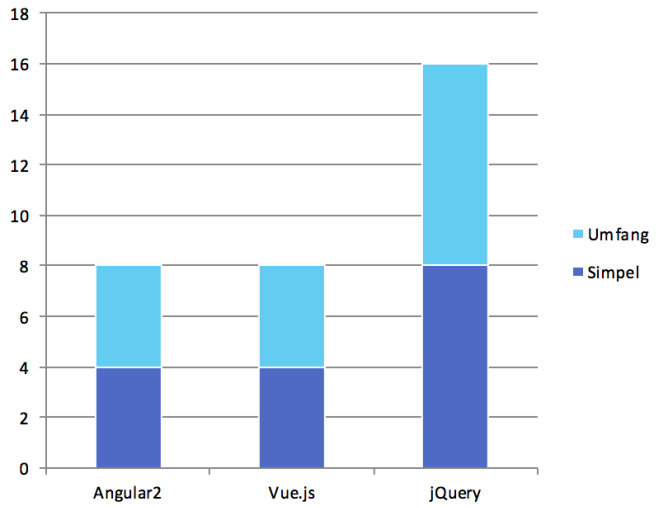


Abbildung 2.9: Diagramm Learning (Angular 2/Vue.js/jQuery)

2.2 Backend

Das Backend des RasterGeoConverters ist das Herzstück der Applikation. Alle Anfragen seitens des Clients werden im Backend bearbeitet, gespeichert und die Resultate wieder zurück ans Frontend gesendet.
Die Technologiewahl des Backends bezieht sich vor allem auf Frameworks mit Datenbank- und Webanbindung. Wenn möglich soll es eine All-in-one Toolbox mit sich bringen, welche flexibel mit neuen Funktionen erweiterbar ist. Zudem soll das Framework keine Lizenzen zu Drittanbietern benötigen und Open Source sein.

2.2.1 Programmiersprachen

Java

Java ist eine der populärsten Programmiersprachen in der Informatik. Dadurch ist die Community sehr gross und bietet viele Open-Source-Lösungen und Frameworks an. Java selbst ist eine objektorientierte und kompilierte Programmiersprache. Der kompilierte Javacode selbst wir in der Java Virtual Machine, kurz JVM, ausgeführt. Durch diese Zwischeninstanz können Java-Programme plattformunabhängig ausgeführt werden. Java selbst gilt als einfach, robust und dynamisch. Durch die vorhandenen Frameworks und Communitylösungen kann Java als einen Kandidaten für den RasterGeoConverter betrachtet werden.

Python

Python [10] ist eine skriptbasierende, hochlevel und interpreter Programmiersprache. Sie findet grosse Anwendung in den akademischen Breitengraden. Durch die stetig wachsende Community und die Open-Source-Strategie, finden sich für diese Sprache eine grosse Palette von Frameworks und Fertiglösungen zu verschiedensten Anforderungen. Python bietet also auch in der geographischen Informationswelt viele Lösungen zu Umrechnungen und Bildverarbeitung an. Als weiteres Kriterium kann die Unabhängigkeit der Plattformen betrachtet werden. Python kann auf Linux, Windows und OS X installiert und interpretiert werden.

2.2.2 Entscheid Programmiersprache

Den Entscheid der Programmiersprache haben wir aufgrund von folgenden Kriterien getroffen:

- Vorhandene Frameworks
- Vorhandene Bibliotheken
- Community
- Geodaten
- Aufwand
- Neues Lernen

Kriterien		Kandidaten			
		Java		Pyhon	
Beschreibung	Gew[1-3]	Pts [1-5]	Result	Pts [1-5]	Result
Frameworks	3	3	9	5	15
Bibliotheken	2	3	6	5	10
Einfachheit	2	3	6	5	10
Community	2	5	10	5	10
Geodaten	3	3	9	5	15
Aufwand	2	4	8	2	4
Neues Lernen	1	2	2	4	4
Resultate		50		68	

Abbildung 2.10: Wertungstabelle Java/Python

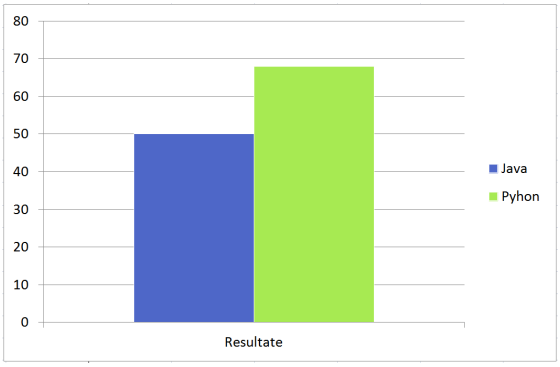


Abbildung 2.11: Wertungsdiagramm Gesamt Java/Python

Kriterien

Frameworks
: Python bietet eine grosse Palette an Frameworks, welche die wichtigsten Funktionen eines Backends in sich mitbringen. Beispiele: Django, Flask, Pyramid. Die Frameworks in Java bieten zwar ähnliche Funktionalitäten, sind aber komplexer umzusetzen und meistens fehlt es an der Flexibilität.

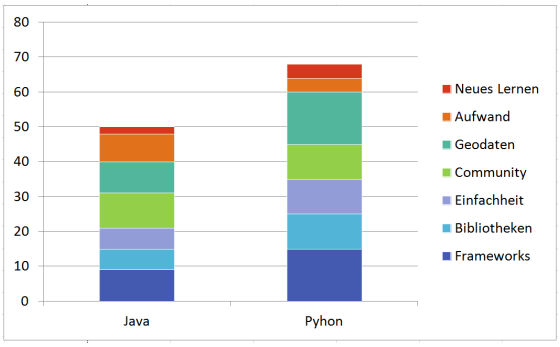


Abbildung 2.12: Wertungsdiagramm Übersicht Java/Python

Bibliotheken
: Java und Python bieten viele unterschiedliche Bibliotheken. Das Einbinden und Installieren ist aber in Python sehr simpel (pip). Zudem ist Python auf Open Source ausgelegt. So können Bibliotheken einfach erweitert werden, da diese meist als Skript eingebunden werden.

Einfachheit
: Das Aufbauen einer Software ist in Python einfacher und schneller gestaltet. Java bietet sich vor allem bei grösseren Projekten an, da die Übersicht in Python schnell ein Problem werden kann. Bei unserer Projektgrösse sollte dies jedoch kein Problem werden.

Community
: Beide Programmiersprachen besitzen eine grosse Community.

Geodaten
: Python wird in der Geowelt gerne benutzt, um Funktionalitäten umzusetzen. Für die Bildkonvertierung zum Beispiel ist GDAL die beste Bibliothek und Python bietet gewissen Funktionalitäten direkt an. Auch für Java finden sich einzelne Bibliotheken. Diese sind aber nicht so zahlreich und umfangreich wie in Python.

Aufwand
: Python muss erst gelernt werden. Java ist und bleibt die beliebteste Programmiersprache. Darum der Aufwand bei Python grösser.

Neues Lernen
: In der Studienarbeit geht es darum, neue Technologien einzusetzen und Probleme zu analysieren. Python als neue Programmiersprache ist daher interessanter.

2.2.3 Frameworks und Bibliotheken

2.2.4 Frameworks

Um den RasterGeoConverter zu realisieren, verlassen wir uns auf Frameworks, welche Grundelemente wie Datenbankanbindungen, Frontend oder Anbindungen ans Frontend und Benutzerverwaltung schon mit sich bringen. Wir wollen uns vor allem auf die Schnittstelle zwischen Benutzer und dem RasterGeoConverter und dem Bezug der Karten von den Webmapservices kümmern.

Django

Das Django Framework [2] eine eine pythonbasierte All-in-one-Lösung, welches die wichtigsten Anforderungen im Backend und auch im Frontend erfüllt. Es bietet Datenbankanbindungen,

Websecurity und auch eine View-Verwaltung fürs Frontend an. Durch die grosse Community im Hintergrund wird Django stetig gewartet, erweitert und verfügt über ein grosses Angebot von Fertiglösungen. Das Framework bietet in-house folgende Features.

- Lauffähigen Entwicklungsserver und Debugger
- Formserialisierung und Validierung
- Datenbankanbindung
- Sessionshandling
- Authentifizierung
- Integriertes Unittesting

Flask

Flask [3] ist ein schlankes pythonbasiertes Framework für die Erstellung von einfachen Apps. Dieses Framework gilt als Lightversion von Django und sollte den grossen Umfang auf das Nötigste reduzieren. Alle weiteren Anforderungen können eingebunden werden. Das Framework bietet in-house folgende Features.

- Lauffähigen Entwicklungsserver und Debugger
- Integriertes Unittesting
- REST-Integration
- Cookiebasierte Sessionen

Pyramid

Pyramid [9] gehört ebenfalls zu den bekannten Python-Frameworks für die Erstellung von Web-Apps. Pyramid wurde schlank gestaltet, um kleine Applikationen schnell zu entwickeln. Falls weitere Anforderungen hinzukommen können, diese mittels sogenannten Addons flexibel hinzugefügt werden. *Start small, finish big*, so der Leitspruch von Pyramid. Das Framework bietet in-house folgende Features.

- Uniform Resource Locator (URL) generation
- Debugging settings
- Class-based and function-based views
- Event system
- Sessionshandling
- Flexible authentication and authorization

2.2.5 Entscheid Frameworks

Den Entscheid des Frameworks haben wir aufgrund von folgenden Kriterien getroffen:

- Umfang: REST, Sessionhandling, Datenbankanbindung, Cookies, Installation, Testing, Zusatzmodule

- Community: Popularität, Foren, Zukunft
- Learning: Übersicht, Umfang

Kriterien		Kandidaten					
		Django		Flask		Pyramid	
Beschreibung	**Gew[1-3]**	**Pts [1-5]**	**Result**	**Pts [1-5]**	**Result**	**Pts [1-5]**	**Result**
Umfang							
REST	3	4	12	4	12	4	12
Sessionhandling	3	5	15	4	12	4	12
Datenbankanbindung	2	5	10	3	6	3	6
Cookies	1	5	5	4	4	4	4
Installation	1	5	5	4	4	4	4
Zusatzmodule	2	5	10	3	6	3	6
Testing	2	4	8	3	6	3	6
Community							
Popularität	2	5	10	4	8	3	6
Tutorials	2	4	8	3	6	3	6
Foren	2	5	10	4	8	3	6
Zukunft	2	5	10	4	8	3	6
Learning							
Simpel	2	3	6	4	8	4	8
Umfang	2	2	4	4	8	4	8
Resultate		**113**		**96**		**90**	

Abbildung 2.13: Wertungstabelle Gesamt (Django/Flask/Pyramid)

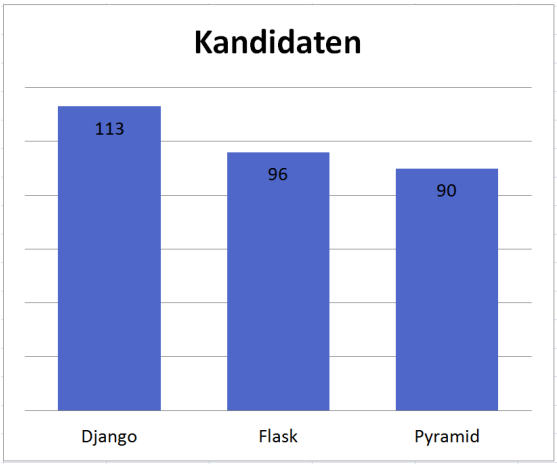


Abbildung 2.14: Diagramm Gesamt (Django/Flask/Pyramid)

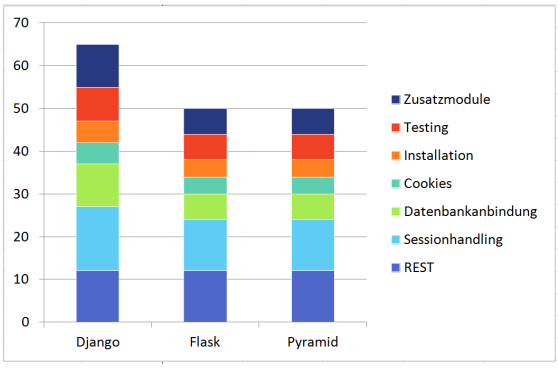


Abbildung 2.15: Diagramm Umfang (Django/Flask/Pyramid)

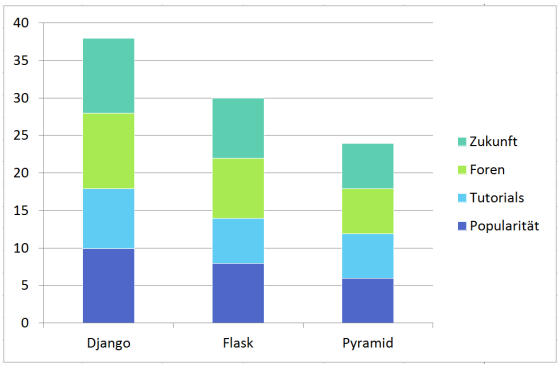


Abbildung 2.16: Diagramm Community (Django/Flask/Pyramid)

Umfang

REST
: Das Framework kommuniziert mit dem Frontend über die REST-Schnittstelle. Dementsprechend ist die Einbindung sehr wichtig. Das Django-Restframework ist bei Weitem die umfangreichste Bibliothek und erfüllt bei unserem Projekt die Anforderung am besten.

Sessionhandling
: Der RasterGeoConverter soll in der Lage eine Map-Sharing-Funktion anzubieten. Dazu ist es wichtig, dass ein Sessionhandling im Framework eingebunden ist. Dies ist bei allen drei Frameworks vorhanden.

Datenbankanbindung
: Der RasterGeoKonverter soll die gängigsten Web Map Service (Allgemein) (WxS) erfassen, speichern und auswerten können. Zudem ist es möglich, dass in naher Zukunft Benutzereingaben gespeichert und als Vorschlag präsentiert werden sollen. Um das zu ermöglichen, muss eine Datenbankanbindung vorhanden sein. Django bietet von Haus aus eine umfangreiche Anbindungsmöglichkeit an. Die anderen Frameworks müssen erweitert werden.

Cookies
: Für die Benutzeridentifikation sollen Cookies benutzt werden. Alle drei Frameworks erfüllen diese Anforderung.

Installation
: Das Framework soll einfach und ohne grossen Zusatzbibliotheken installierbar sein. Alle Frameworks können über den Python Pip-Installer bezogen werden. Da Django die meisten Funktionalitäten integriert hat, fallen hier Zusatzaufwände weg.

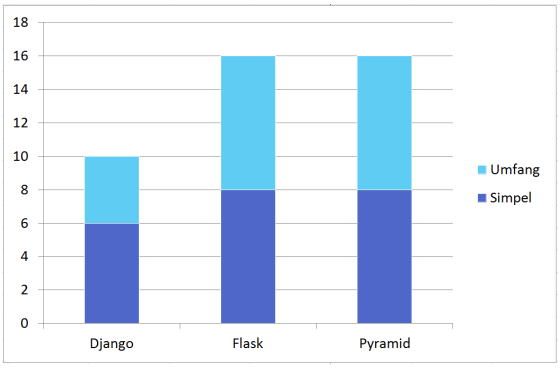


Abbildung 2.17: Diagramm Learning (Django/Flask/Pyramid)

Zusatzmodule
: Falls die Frameworks gewisse Funktionen nicht besitzen, können diese mit Bibliotheken ergänzt werden. Django besitzt hier die grösste Unterstützung.

Testing
: Die Funktionen sollen über Tests geprüft werden können. Alle Frameworks sind testbar. Django verfügt aber über eine grössere Vielfalt.

Community

Popularität
: Ein populäres Framework wird stetig erweitert und korrigiert. Zudem erstellt die Community neue Bibliotheken, Funktionen und aber auch Lösungen. Hier besitzt Django einen klaren Vorsprung. Da Flask und Pyramid Neulinge sind, ist dementsprechend die Community bei diesen Frameworks noch klein. Die meisten Lösungsansätze sind auf Django abgeschnitten.

Tutorials
: Um die Funktionalitäten des Frameworks zu erlernen sind Tutorials der leichteste Weg. Django besitzt ein umfangreiches, aber auch komplexes Tutorial. Flask und Pyramid haben nur die Grundlegendste Funktionen in einem Tutorial behandelt.

Foren
: Nicht alle Problemstellungen können mit einer Bibliothek gelöst werden. Und wenn die Tutorials keine Antworten auf Fragen haben, so sind die Foren der beste Freund des Programmierers. Hier besitzt Django einen ganz klaren Vorsprung.

Zukunft
: Der RasterGeoConverter soll über längere Zeit funktionieren und allenfalls auch erweitert werden. Darum sollte ein Framework auch in Zukunft eingesetzt werden, sodass der Support sichergestellt ist. Django ist das älteste Framework, aber durch die tragende Community und die stetige Weiterentwicklung glauben wir, dass Django auch in Zukunft eingesetzt werden wird.

Learning

Simpel
: Damit das Framework gelernt werden kann, sollten die Funktionen einfach zu erfassen sein. Oder aber die Beschreibungen und Tutorial erklären es mit treffenden Beispielen. Django besitzt einen ungeheuren Umfang, zudem sind die Funktionen nicht immer so übersichtlich. Die Tutorials oder Erklärungen sind zum Teil komplex oder fassen nicht alles an

einem Ort zusammen. Da sind Flask und Pyramid durch ihren simplen aufbau besser zu erlernen.

Umfang
: Das Framework sollte wenn möglich viele Funktionen selbst abdecken. Mehr Funktionen bedeuten aber auch, dass mehr Erklärungen nötig sind. Der Umfang von Django ist riesig. Es bedarf zwar nicht viel Zeit, um eine einfache App zu programmieren. Dennoch soll die App möglichst effiziente Funktionen aufweisen und auch übersichtlich sein, bedarf es eine längere Lernzeit, als es bei Flask und Pyramid nötig ist.

2.2.6 Bibliotheken

GDAL

`http://www.gdal.org/`
Die Geospatiale-Daten-Abstractions-Bibliothek, kurz GDAL, ist eine Übersetzungsbibliothek für raster- und vektorgeospatiale Datenformate. Sie wird unter der Open Source Lizenz von der Open Source Geospatial Foundation betrieben. Traditionell wurde GDAL vor allem für die Rasterdaten benutzt. Mit OGR als Erweiterung ist es aber auch möglich, Vektordaten zu transformieren. Diese Bibliothek kann über verschiedene Schnittstellen angesprochen werden und ist durch ihre Einzigartigkeit auch unsere erste Wahl für die Erzeugung von GeoTIFF's.

OwsLib

`https://geopython.github.io/OWSLib/`
Die OWSLib ist eine Open Source Pythonbibliothek für die Kommunikation mit Open Geospatial Consortum Webservices Schnittstellen. Damit können Webservices für Karten direkt angesprochen werden und die verfügbaren Informationen extrahiert werden. Dazu gehören verfügbare Parameter und aber auch das Kartenmaterial. OWSLib wird durch die Community stetig erweitert. Als ein Pythonpaket kann die Funktionalität bei Gebrauch auch direkt erweitert oder geändert werden. Für unser Projekt erfüllt diese Anbindung alle nötigen Voraussetzungen und wurde daher eingebunden.

Kapitel 3

Eingesetzte Technologien

3.1 Angular 2

`https://www.angular.io/`
Angular 2 ist ein clientseitiges JavaScript-Webframework zur Erstellung von Single-page-Webappliaktionen. Es basiert dabei auf TypeScript. Das Framework ist Open-Source und wird von Google entwickelt.
Die Struktur einer Angular 2 Webanwendung basiert dabei auf Komponenten. Eine Komponente ist eine Kombination aus HTML-Template und einer Komponenten Klasse welche das HTML-Template kontrolliert.
Angular Anwendungen und Angular selbst ist abhängig von Features und Funktionalitäten, die von einer Vielzahl von Drittanbieter-Paketen kommen. Nun folgen die wichtigsten eingesetzten Packages.

3.1.1 Eingesetzte Packages

Übersicht:

- @angular/http
- @angular/router
- @types/leaflet
- angular-cli
- bootstrap
- leaflet
- leaflet-draw
- rxjs

@types/leaflet

In diesem Package befinden sich die Typings für die Leaflet Library. Damit Leaflet im TypeScript Code verwendet werden kann, müssen Typings dazu zur Verfügung stehen.

Angular CLI

Angular CLI ist ein Command Line Interface für Angular 2. Es generiert eine Applikation, welche bereits out of the box funktioniert. Auch können weitere Komponenten sehr einfach generiert werden. Auch beinhaltet Angular CLI gleich einen Webpack Server, womit die Applikation für Testzwecke laufen gelassen werden kann.

Bootstrap

Mit Hilfe von Bootstrap wurde dafür gesorgt, dass die Website Responsive ist und auch ein gewisses Grundstyling besitzt.

Leaflet

Das Leaflet Package liefert den Leaflet-Sourcecode, welcher für die Kartenintegration benötigt wird.

Leaflet Draw

Mit dem Leaflet Draw Package wird das Leaflet Package erweitert. Das Leaflet Draw Package beinhaltet die Funktionalitäten um Auswahlen auf der Karte zu zeichnen und die Koordinaten davon zu erhalten.

RxJS - Reactive Extensions

Durch das RxJS Package wird unter anderem die Verwendung von Observables in Angular ermöglicht.

3.2 Django

`https://www.djangoproject.com/`
Django ist ein high-level Python Webframework für die schnelle und trotzdem umfassende Erstellung von Webapplikationen.
Wie in der Evaluation schon angesprochen, erfüllt Django die meisten Anforderungen für das Backend. Durch die grosse Popularität dieses Frameworks werden für die fehlenden oder zu wenig umfassenden Funktionen stets neue Lösungen erstellt oder erweitert. Auch für den RasterGeo-Converter müssen weitere Plugins installiert werden, um gewissen Anforderungen gerecht zu werden.

3.2.1 Eingesetzte Plugins

Übersicht:

- click
- django-cors-headers
- django-environ
- djangorestframework

- first
- owslib
- pip-tools
- psycopg2
- pyproj
- python-dateutil
- pytz
- PyYAML
- requests
- shapely

django-cors-headers

`https://pypi.python.org/pypi/django-cors-headers/1.1.0`
Durch die Aufteilung in zwei Komponenten im RasterGeoConverter müssen wir in unserem Projekt den Cross Origin Resource Sharing, kurz CORS, konfigurieren. Im Web ist durch die Same-Origin-Policy der Zugriff auf andere Quellen durch eine aufgerufene Webapplikation untersagt. Um dem entgegen zu wirken, wurde der CORS-Mechanismus entwickelt. Damit nun Django dieses Feature unterstützt und den Zugriff durch das Frontend zulässt, musste das "django-cors-headersPlugin installiert werden.

djangorestframework

`http://www.django-rest-framework.org/`
Das Frontend des RasterGeoConverters kommuniziert über die REST-Schnittstelle mit dem Django-Backend. Für die umfassende Anbindung und aber auch brauchbare Umsetzung haben wir uns für das "django-rest-frameworkPlugin entschieden. Das Plugin punktete mit umfassenden Funktionalitäten, einfache Einbindung und gute Testbarkeit.

psycopg2

`https://pypi.python.org/pypi/psycopg2`
Als Open Source Projekt sollte der RasterGeoConverter auch eine Datenbank nutzen, welche die Anforderungen im laufenden Betrieb erfüllt und ebenfalls die Open Source Standards erfüllt. PostgreSQL war daher die logische Wahl. Für die Anbindung von Django wurde das Plugin "psycopg2ïnstalliert.

PyYAML

`https://pypi.python.org/pypi/pyaml`
Für das Einlesen von Testdaten oder Konfigurationen wurde Yaml gewählt. Yaml ist eine vereinfachte Auszeichnungssprache zur Datenserialisierung. Es ist an XML angelehnt jedoch simpler gestaltet und durch den Menschen einfach lesbar. Für die Einbindung wurde die Python-Bibliothek PyYAML verwendet.

shapely

`https://pypi.python.org/pypi/Shapely`
Für das Umrechnen von Koordinatensystemen haben wir die Shapely-Bibliothek integriert. Shapely wurde für die Manipulation und Analysen von geometrischen Objekten im kartesischen System entwickelt.

Kapitel 4

Umsetzungskonzept

Das Umsetzungskonzept im technischen Bericht soll eine allgemeine Übersicht der Features und ihrer Umsetzung des RasterGeoConverters gewähren. Die detaillierte Umsetzung innerhalb der Software ist in der Softwaredokumentation zu entnehmen.

4.1 WxS ansprechen

Die Webservices für Karten liefern alle Dateninformationen zu den von Ihnen verwalteten Karten im XML-Format. Für den Abruf einer gewünschten Karte muss eine passende URL erzeugt werden.

4.1.1 WxS Inhalt

Der Abruf aller möglichen Parameter eines WxS wird über das Backend realisiert. Die OWSLib bietet dafür die nötigen Funktionen. Das Backend kann so über die Bibliothek alle nötigen Parameter extrahieren, filtern und dem Frontend zu Verfügung stellen.

4.1.2 Kartenabruf

Für einen erfolgreichen Kartenabruf benötigt der WxS mehrere Parameter. Je nach Kartentyp können diese unterschiedlich ausfallen. Für eine einfache Kartenansicht übernimmt das Frontend selbst diese Aufgabe. Das Kartenplugin greift auf die verfügbaren Karteninformationen des WxS zurück. Damit nun ein Bild im gewünschten Format generiert werden kann, greift der RasterGeoConverter zu den verfügbaren Funktionen im Backend zurück. Im Backend wird die nötige URL zusammengestellt und die daraus resultierende Karte heruntergeladen.

4.2 GeoTIFF erzeugen

Nicht alle Webservices für Karten bieten GeoTIFF's als mögliches Bildformat an. Aus diesem Grund wird mit der Hilfe von GDAL eine solche Funktion im Backend zur Verfügung gestellt. Das Backend speichert ein Bild von einem Webservice in einem passenden Format ab und transformiert dieses mit der passenden GDAL-Funktion zu einem GeoTIFF.

Kapitel 5

Resultate, Bewertung und Ausblick

5.1 Zielerreichung

Zusammengefasst wurden folgende Ziele in der Aufgabenstellung definiert:

- Webapplikation bestehend aus Frontend und Backend
- Responsive Website mit guter Usability
- Benutzer können einen Kartenausschnitt auswählen, die nötigen Kriterien setzen und als GeoTIFF downloaden

Durch die Definition einer REST-Schnittstelle konnten wir eine klare Trennung in Frontend und Backend erstellen. Der Einsatz von Angular 2 hat uns dabei geholfen eine flüssig laufende Oberfläche zu erstellen, was sehr benutzerfreundlich ist. Mit Hilfe von Bootstrap konnte auch sichergestellt werden, dass die Website sich responsiv verhaltet auf verschiedenen Bildschirmgrössen.
In Zusammenarbeit mit dem Betreuer hatte sich die Website anhand des Prototyps mehrfach verändert und optimiert. Daraus folgte als Resultat eine schlichte Website welche sich in das Corporate Identity der HSR einpasst. Durch den flüssigen Ablauf ist dem Benutzer immer klar, was gerade geschieht auf der Website und es ist auch klar ersichtlich wenn auf eine Operation gewartet werden muss. Dies zeugt von guter Usability.
Der Benutzer kann einen Kartenausschnitt auswählen und davon ein GeoTIFF erzeugen lassen welches dann zum Download zur Verfügung gestellt wird.
Durch die Verwendung von Django im Backend konnte auf einfache Weise eine REST-Schnittstelle erstellt werden welche mit den Funktionen im Backend interagiert. Sehr viele aufwendige Rechenaufgaben wurden dadurch dem Backend überlassen. Darunter auch die GeoTIFF Generierung.

5.2 Weiterentwicklung

Der RasterGeoConverter wurde über eine Studienarbeit entwickelt und durch die begrenzte Zeit ist auch klar, dass nicht alle Ziele erreicht werden konnten. Dieses Kapitel widmet sich insbesondere auf die fehlenden Features, welche den RasterGeoConverter in der jetzigen Version fehlen oder noch nicht gänzlich umgesetzt werden konnten.

5.2.1 WMTS

Neben den WebMapService für rasterbasierende, digitale Karten gibt es noch den WebMapTile-Service, kurz WMTS, als Dienst für digitale Karten welche kachelbasiert sind. Die Kachel-Technik, wie es in diesem Service angewendet wird, funktioniert ähnlich zu einem Mosaik. Die Karten werden in mehreren Stücken geteilt. So müssen Karten nicht als Ganzes übertragen werden, sondern werden für einen Abruf zusammengesetzt. Der WMTS wurde zwar WMS-ähnlich aufgebaut, benötigt jedoch zusätzliche oder unterschiedliche Parameter als der WMS. Für die erfolgreiche Einbindung muss der RasterGeoConverter mit den spezifischen Funktionen erweitert werden.

5.2.2 WCS

Der WebConverageService, kurz WCS, ist ein weiterer Schnittstellenstandard für die webbasierte Abfrage von digitalen Geoinformationen. Im Gegensatz zum Web Map Service, der Geo-Daten als Kartenbilder liefert, stellt der WCS verfügbare Daten zusammen mit ihren detaillierten Metadaten bereit und definiert eine reiche Syntax für Anfragen auf diesen Daten und Metadaten. Insbesondere werden Daten mit ihrer vollen Semantik zurückgegeben ausgeliefert. Im Gegensatz zu den lediglich für Menschen geeigneten Bildern des WMS lassen sich WCS-Daten deshalb auch maschinell weiter auswerten.

Der WCS benötigt ein weiteres Spektrum an Parameter wie zum Beispiel raum- und/oder zeitvariierende Angaben. Der RasterGeoConverter kann mit den nötigen Funktionen erweitert werden, sollte WCS erfolgreich eingebunden werden.

5.2.3 Datenhaltung

Der RasterGeoConverter extrahiert aus WxS-URL die nötigen Parameter und kommuniziert für die Kartenanfrage mit den jeweiligen Services und Server. Aus diesen Anfragen könnten die Adressen und die Anzahl Anfragen in Statistiken gespeichert werden. Mit diesen Daten könnten Listen mit den meistbenutzten Services erstellt werden, um die Benutzer bei der Suche nach dem passenden Service zu unterstützen. Zudem wäre es so auch möglich, den momentanen Status eines Services darzustellen. Zum Beispiel die Verfügbarkeit oder die Bestimmungen.

In der momentanen Version werden zwar Einträge in die Datenbank übernommen, aber die Daten werden nicht weiter ausgewertet. Dies könnte sich mit einer entsprechenden Erweiterung ändern.

5.2.4 Karte teilen

Der RasterGeoConverter lädt regelmässig Karten von den Services und generiert GeoTIFF's. Dies produziert Datenverkehr zwischen dem Server und den Services. Und da diese Karten zum Teil eine immense Datengrösse haben, lohnt es sich, wenn diese Karten unter den interessierten Benutzern geteilt werden könnten.

Dazu könnte dem Benutzer die Möglichkeit geboten werden, seine geladene Karte zu teilen. Der RasterGeoConverter würde die geladene Karte in den Sharing-Ordner platzieren und dem Benutzer den Schlüssel in Form eines Linkes übertragen und/oder einen Eintrag in eine Ladeliste einfügen. Über diesen Link kann dann die Zielkarte direkt geladen und gespeichert werden, ohne eine weitere Anfrage starten zu müssen.

5.2.5 Weitere Parameter

Die Web Services für Karten besitzen standardisierte Parameter für die Abfrage der gewünschten Karten. Je nach Service werden aber auch optionale Parameter unterstützt. In der momentanen Version des RasterGeoConverters werden ausschliesslich die Standardparameter unterstützt. Es benötigt also eine Erweiterung der Funktionen, sollte die optionalen Parameter ebenfalls berücksichtigt werden können.

5.2.6 Vendor specific parameters

Die sogenannten Vendorspecific-Parameter sind individuelle Parameter, welche von keinem Standard erfasst wurden und von einzelnen Services angeboten werden. Momentan gibt es in der jetzigen Version des RasterGeoConverters keine Möglichkeit diese Parameter zu übertragen. Ein möglicher Lösungsansatz wäre es dazu, über eine Inputfunktion diese Parameter individuell an den Service zu übertragen. Da diese Parameter nicht überprüft werden können, wäre die Verantwortung beim Benutzer.

5.3 Persönliche Berichte

5.3.1 Persönlicher Bericht von Diego Etter

Das Projekt RasterGeoConverter war eine interessante und herausfordernde Studienarbeit. Der grosse Spielraum im Bezug der Umsetzung, die vielseitigen Anforderungen und die Evaluation der einzusetzenden Technologien erforderten ein umsichtiges Projektmanagement. Obwohl die Projektplanung sehr umsichtig gewählt wurde und wir stets im Zeitplan waren, mussten wir feststellen, dass wir nicht allen Risiken die nötige Aufmerksamkeit geschenkt haben. So brachte die Evaluation der richtigen Technologie und auch das Erlernen einer neuen Sprache einen grossen Zeitaufwand mit. Zeit, die wir auf für die Umsetzung des Projekts hätten brauchen können. Ich lege stets einen hohen Massstab an meine Leistung und bin nicht so schnell zufrieden mit dem Ergebnis, das ich liefere. Und obwohl wir einiges erreichen konnten und der RasterGeoConverter an sich funktionierte, war ich etwas enttäuscht über das Resultat. Gerne hätte ich auch einige optionale Ziele erreichen wollen, oder die Software in einem ausgefeilteren Zustand abgegeben. Aber, es ist eine Studienarbeit und die Zeit war sehr begrenzt. Das dürfen wir hier nicht vergessen.

Mit Python und Django konnte ich Neuland im Bereich Programmiersprache und Frameworks betreten. Und wie bei jeder neuen Sprache musste ich auch hier meine Sporen abverdienen und lernen mit den Eigenheiten, Vorgaben und Schwierigkeiten umzugehen. Leider konnte ich mir durch die knappe Zeit keine weiteren nötigen Technologien so richtig aneignen. Docker und auch die nötigen Servertechnologien, um den RasterGeoConverter erfolgreich auszuführen, konnte ich nur oberflächlich behandeln. Gerne hätte ich mich auch hier weiter vertieft.

Ich empfand indes die Zusammenarbeit mit Céline Ott als Projektpartner unserem Betreuerteam Stefan Keller und Nicola Jordan als sehr bereichernd und angenehm. Vor allem die Freiheiten im Bezug des Projektmanagements haben mich positiv herausgefordert und motiviert die Aufgaben nach bestem Wissen und Gewissen zu meistern.

Meine wichtigsten Erkenntnissen aus dem Projekt waren:

Mehr Gewicht auf die Risikoanalyse setzen und die Risiken stets im Auge behalten.

Die Dokumentation in die Projektarbeit miteinbeziehen.

Laufende Issues direkt angehen und die Teamarbeit darauf abstimmen.

5.3.2 Persönlicher Bericht von Céline Ott

Die Thematik von Rasterkarten interessierte mich von Anfang an, weshalb wir uns auch auf diese Arbeit beworben hatten. Zu Beginn rechnete ich damit, dieses Projekt mit Python umzusetzen. In dieser Programmiersprache hatte ich bis dahin noch keine Erfahrungen gemacht.

Während der Evaluation kristallisierte sich heraus, dass die Applikation aus zwei Teilen, einem Frontend und einem Backend, bestehen wird. Von Anfang an präferierte ich den Frontend-Teil da ich bereits mehrere Webseiten und Webapplikationen erstellt hatte. Da für das Frontend keine Vorgaben bestanden, konnte ich die Technologie relativ frei wählen. Nach einer Evaluation der für mich spannendsten Varianten kam ich zum Entschluss, Angular 2 zu wählen. Da ich bis anhin nur in Angular 1 gearbeitet hatte, musste ich die Technologie zuerst noch erlernen. Es war für mich sehr motivierend eine neue Technologie zu lernen, welche bestimmt eine Zukunft hat. Jedoch brachte dies auch einige Risiken mit sich, welche wir leider nicht genügen abgeschätzt hatten. So dauerte die Einführung in die neue Technologie länger als zuvor angenommen.

Nachdem der Prototyp in Angular 2 erstellt worden war, traf ich bereits auf die ersten Hürden. Es bereitete mir einige Schwierigkeiten die native JavaScript Library Leaflet in die TypeScript Umgebung von Angular 2 einzubinden. Dafür musste ich Typings verwenden. Während der Entwicklung benötigte ich immer mehr Funktionen der Leaflet Library und kam somit an die Grenzen der Typings. So musste ich die bestehenden Typings erweitern, damit ich alle benötigten Funktionen verwenden konnte. Es war sehr lehrreich ein solches Typing zu erweitern.
Während dieser Arbeit konnte ich sehr viel Neues lernen, was mich sehr freute.

Die Zusammenarbeit mit Diego Etter und dem Betreuerteam funktionierte sehr gut. Unsere Organisation im Team war sehr gut und so waren wir auch immer im Zeitplan.

Diese Arbeit war eine tolle Erfahrung für mich und ich werde auch in Zukunft bestimmt wieder mit Angular 2 arbeiten.

Teil II

Softwareprojekt Dokumentation

Kapitel 6

Anforderungsspezifikation

In den Anforderungsspezifikationen werden die Ansprüche an den RasterGeoConverter genauer erläutert. Dazu gehören funktionale wie auch nicht funktionale Anforderungen. Anhand der Use Cases konnten die Anforderungen noch genauer beschrieben werden.

6.1 Funktionale Anforderungen

Die Funktionalen Anforderungen wurden in Ziele und optionale Ziele unterteilt.

6.1.1 Ziele

- Funktionierende Webapplikation (Frontend und Backend)
- Webapplikation unterstützt WMS
- Kartenausschnitt ist wählbar
- Karte kann als GeoTIFF heruntergeladen werden

6.1.2 Optionale Ziele

- Webapplikation unterstützt WMTS und/oder WCS

6.2 Use Cases

6.2.1 Diagramm

Über die Aufgabenstellung konnten wir die nötigen Anwendungsfälle erfassen können.

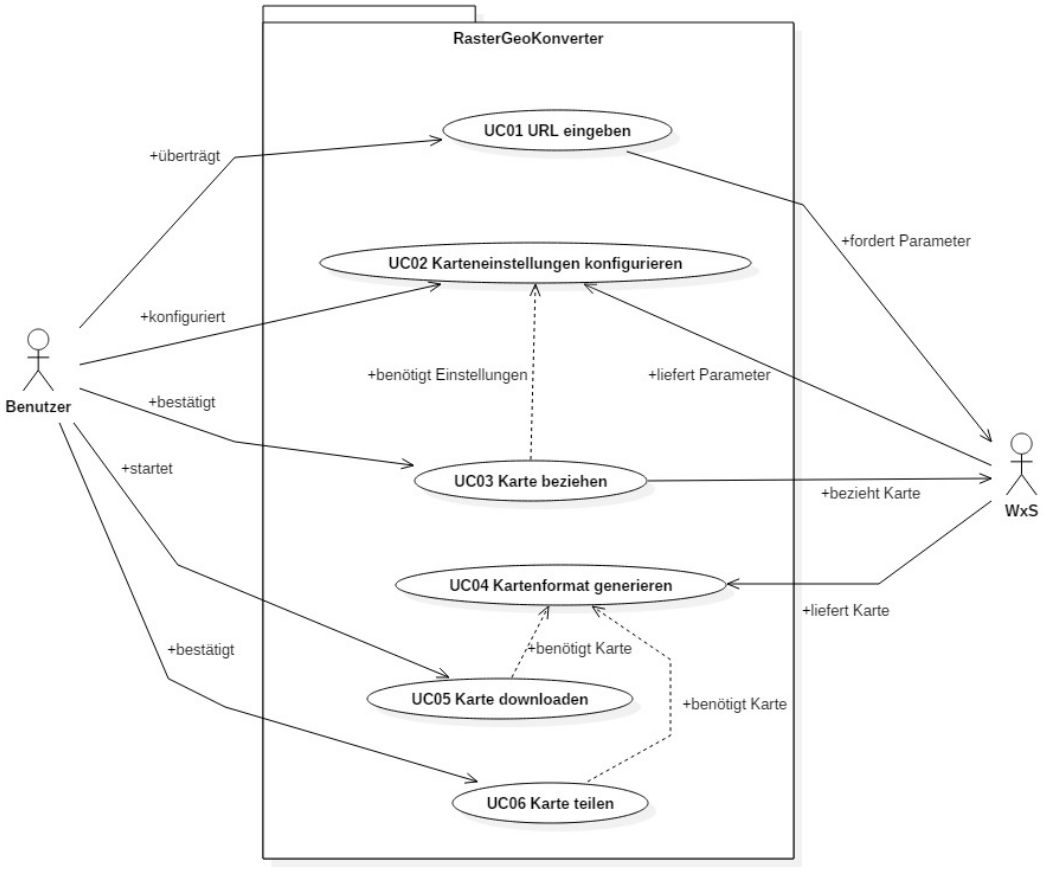


Abbildung 6.1: Use Case Diagramm

6.2.2 Akteure

Wir haben über die Identifikation der nötigen Anwendungsfälle zwei Akteure definieren können.

- Benutzer
- Web-Service für Karten (WxS)

Benutzer

Der Benutzer ist unser Hauptakteur des RasterGeoConverters. Der Benutzer möchte über eine Url einen Karten-Web-Service ansprechen, die Karte anwählen und beziehen.

Web-Service für Karten

Der Karten-Web-Service bietet eine Auswahl an verschiedene Karten an. Um diese zu beziehen, benötigt der Service vordefinierte Anfrageparameter. Der Service kann auch Beschränkungen definieren, wie zum Beispiel maximale Auflösungen oder Authentifizierung.

6.2.3 Formuliert

UC01: URL eingeben

Akteure: Benutzer
Beteiligte Systeme: RasterGeoConverter
Beschreibung:
Der Benutzer überträgt eine URL eines WMS in den RasterGeoKonverter. Der RasterGeoConverter überprüft, ob die URL gültig ist und erstellt eine Verbindung zum WxS. Falls die Verbindung erfolgreich war, fragt der RasterGeoKonverter nach den Parametern.

UC02: Anfrage konfigurieren

Akteure: Benutzer
Beteiligte Systeme: RasterGeoConverter, WxS
Beschreibung:
Der Benutzer wählt die vom WMS gegebenen Werte aus. Für neue Werte, welche von den konfigurierten Werten abhängig sind, wird der WxS fortlaufen angefragt.

UC03: Karte beziehen

Akteure: Benutzer
Beteiligte Systeme: RasterGeoConverter, WxS
Beschreibung:
Voraussetzung ist, dass gültige Karteneinstellungen gewählt wurden. Die gewählten Werte werden zum WxS übermittelt und eine Verbindung zum Wms erstellt.

UC04: Bildformat generieren

Akteure: Benutzer
Beteiligte Systeme: RasterGeoConverter, WxS
Beschreibung:
Der RasterGeoConverte formatiert nach dem Erhalt der Karte das Bild zum gewünschten Format um und speichert das bezugsbereite Bild.

UC05: Karte downloaden

Akteure: Benutzer
Beteiligte Systeme: RasterGeoConverter
Beschreibung:
Voraussetzung ist, dass die Karte erfolgreich geladen wurde. Der Benutzer lädt die Karte zu einem von ihm gewählten Ort, sobald er informiert wurde, dass die Karte bereitgestellt ist.

UC06: Karte teilen

Akteure: Benutzer
Beteiligte Systeme: RasterGeoConverter
Beschreibung:
Voraussetzung ist, dass die Karte erfolgreich geladen wurde. Der Benutzer bestätigt, dass er die Karte teilen möchte. Der RasterGeoKonverter erzeugt einen öffentlichen Pfad und übermittelt diesen an den Benutzer. Der Pfad ermöglicht, weiteren Benutzern die gleiche Karte zu beziehen.

6.3 Nicht funktionale Anforderungen

Die folgenden nicht funktionalen Anforderungen wurden der Aufgabenstellung entnommen:

6.3.1 Sprachen

- Frontend in Deutsch
- Sourcecode in Englisch

6.3.2 Technologien

- Frontend: keine Vorgaben
- Backend: Python
- Datenbank: Bevorzugt PostgreSQL
- Optional: Container Technologie von Docker

6.3.3 Daten

- Von öffentlich zugänglichen Webdiensten

6.3.4 Design

- Das HSR Corporate Identity soll eingehalten werden

6.3.5 Usability

- Responsives User Interface
- Verarbeiten von mindestens 10 Benutzern

Kapitel 7

Designkonzept

7.1 Anforderungen

Aus der Aufgabenstellung konnten die folgenden Anforderungen an das Userinterface gestellt werden:

- Das HSR Corporate Identity soll eingehalten sein
- Es soll auf eine hohe Usability geachtet werden
- Die Website soll responsiv sein und sich an verschiedene Bildschirmgrössen anpassen. Wobei die Zielgruppe aber Desktop User sind.

7.2 Usability

Um auf gute Usability zu achten, gibt es sehr viele Ansätze. Jakob Nielsen ist eine der wohl berühmtesten Personen im Bereich der Benutzerfreundlichkeit. Seine 10 Prinzipien zu Interaktionsdesign [7] sind sehr gute Faustregeln, welche für eine gute Benutzerfreundlichkeit unbedingt beachtet werden sollten. Für diese Arbeit werden deshalb unter anderem diese 10 Prinzipien angewendet.

7.3 Wireframes

In einem ersten Schritt wurden Wireframes von Hand gezeichnet. Dabei war die grundlegende Idee die verschiedenen User Interaktionsmöglichkeiten passend anzuordnen, sodass ein übersichtliches GUI entsteht. Ein wichtiger Aspekt dabei war auch herauszufinden welche Interaktionsmöglichkeiten benötigt werden, und wie die Informationshierarchie sein soll. Um einen ersten Eindruck zu bekommen, wurden auch viele bereits bestehende Webapplikationen angeschaut, welche ebenfalls mit einer Kartendarstellung arbeiten.
In der Abbildung 7.1 ist ein erster, sehr rudimentärer Ansatz des GUIs ersichtlich. Dies wurde benötigt, um eine erste Vorstellung der Anordnung zu bekommen. Anhand dieser Wireframes konnte dann ein erster Prototyp in einem Prototyping Tool erstellt werden.

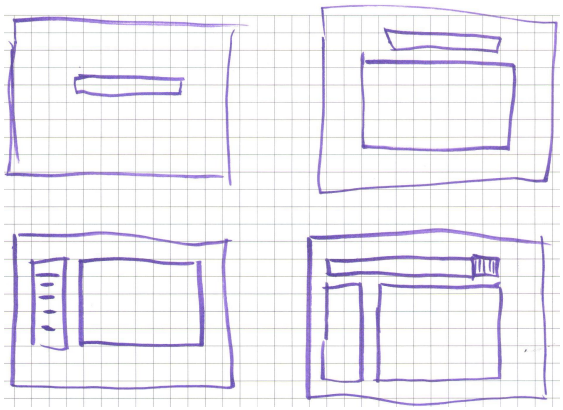


Abbildung 7.1: Erste rudimentäre Wireframes

7.4 Prototyp

In Abbildung 7.2 ist die aller erste Version des GUI-Prototyps ersichtlich. Dabei wurde das Augenmerk hauptsächlich auf das erste Bedienelement, die WMS URL Eingabe, gelegt. Wichtig dabei war eine herausstehende Platzierung, sodass dieses Bedienelement klar als erste Interaktionsmöglichkeit hervorgeht.
Deshalb soll auch auf ein minimalistisches Design Wert gelegt werden, wie es in den 10 Heuristiken von Nielson [7] unter dem Punkt Äesthetic and minimalist designërwähnt ist. Dieser Punkt besagt, dass jede zusätzliche Interaktionsmöglichkeit mit der im Dialog vorhandenen Interaktionsmöglichkeit konkurriert. Deshalb ist die WMS URL Eingabe sehr minimalistisch und hervorstechend gehalten.

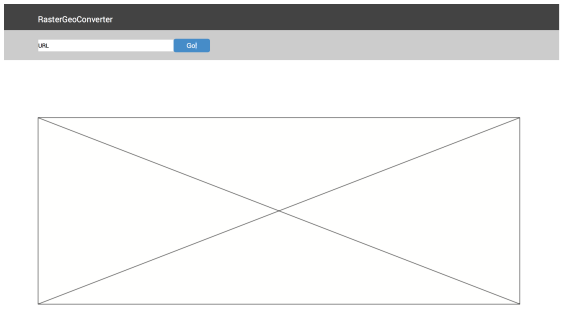


Abbildung 7.2: Erster Wireframe Prototyp

7.5 Designentwicklung

In einem nächsten Schritt haben wir einen klickbaren Prototypen erstellt. Dies half uns bereits früh zu erkennen, wo unterschiedliche Vorstellungen zwischen uns und dem Auftraggeber lagen. In Abbildung 7.3 ist der Startscreen der Applikation ersichtlich. Erst wenn eine WMS URL eingegeben wurde und auf den Button "Go!"geklickt wurde, wird das restliche Interface angezeigt. In Abbildung 7.4 ist ersichtlich, wie das Userinterface aussieht nach dem Klick auf den Button "Go!". Diese hierarchische Benutzerführung soll einen Ablauf in die Applikation bringen. Denn ohne URL können alle weiteren Schritte nicht verwendet werden. Deshalb ist es wichtig, dem Benutzer einen Ablauf zu signalisieren.

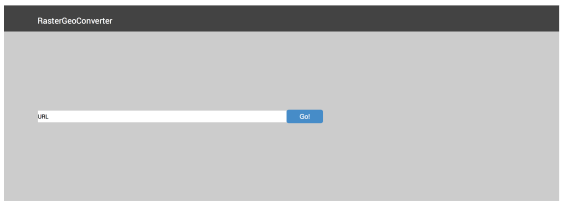


Abbildung 7.3: Erster funktionaler Prototyp

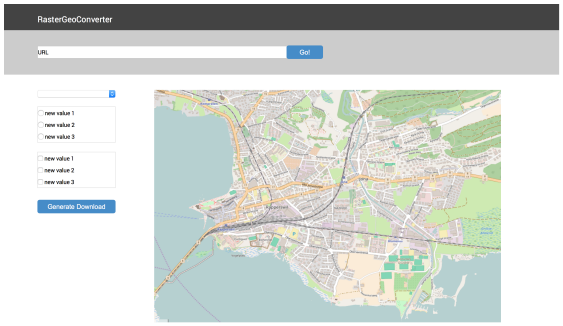


Abbildung 7.4: Erster funktionaler Prototyp nach klick auf "Go!"

In einer zweiten Variante des Prototypen, wie sie die Abbildung 7.5 zeigt, wurden die Bedienelemente oberhalb der Karte platziert, damit der Kartenausschnitt grösser dargestellt werden kann. Durch die Platzierung der Bedienelemente oberhalb rutscht die Kartendarstellung nach unten und es muss gescrollt werden, damit die ganze Karte ersichtlich ist. Dies ist ein Umstand für den User, da nicht alle Bedienelemente auf einer Seite ersichtlich sind. Aus diesem Grund wurde diese Version wieder verworfen.

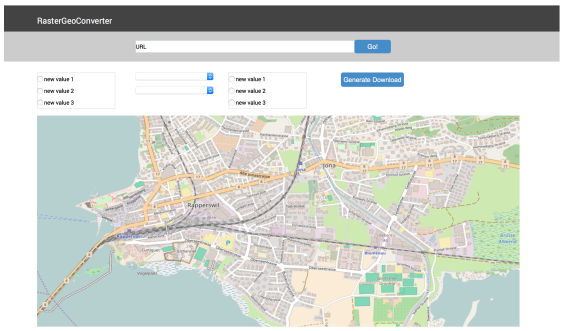


Abbildung 7.5: Zweite Variante des Prototypen

7.5.1 Weiteres Bedienelement Layerselect

Nach weiteren Recherchen bezüglich WMS und Anwendungsszenarien wurde klar, dass ein zusätzliches, wichtiges Bedienelement die Layer-Auswahl ist. Es gibt WMS mit nur wenigen Layers, aber auch WMS mit bis zu 100 Layers.
In Abbildung 7.6 ist deshalb der Prototyp mit einem Auswahlfeld für die Layers erweitert worden. Dadurch sollte es übersichtlicher sein, einen Layer auszuwählen.

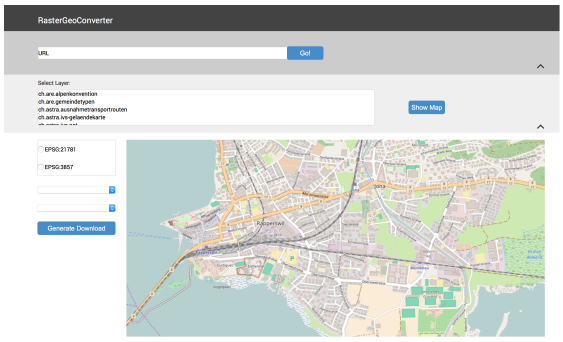


Abbildung 7.6: Prototyp mit Layerauswahl

Nach weiterem Feedback kam jedoch heraus, dass die Auswahlbox zu gross ist und somit wieder viel Platz für die Karte verloren geht. Ausserdem ist das Scrollen in einer solchen Box eher mühsam. Deshalb wurde der Prototyp mit einem einfachen Select-Feld erweitert. In Abbildung 7.7 ist nun der letzte Prototyp ersichtlich, welcher bereits sehr ähnlich wie das resultierende GUI aussieht.

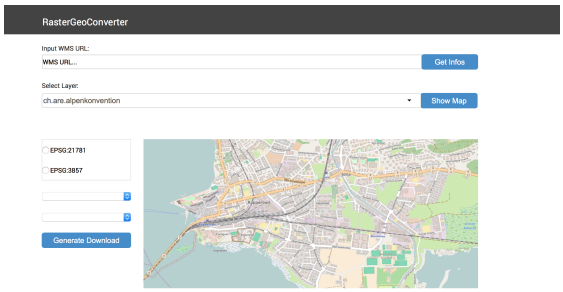


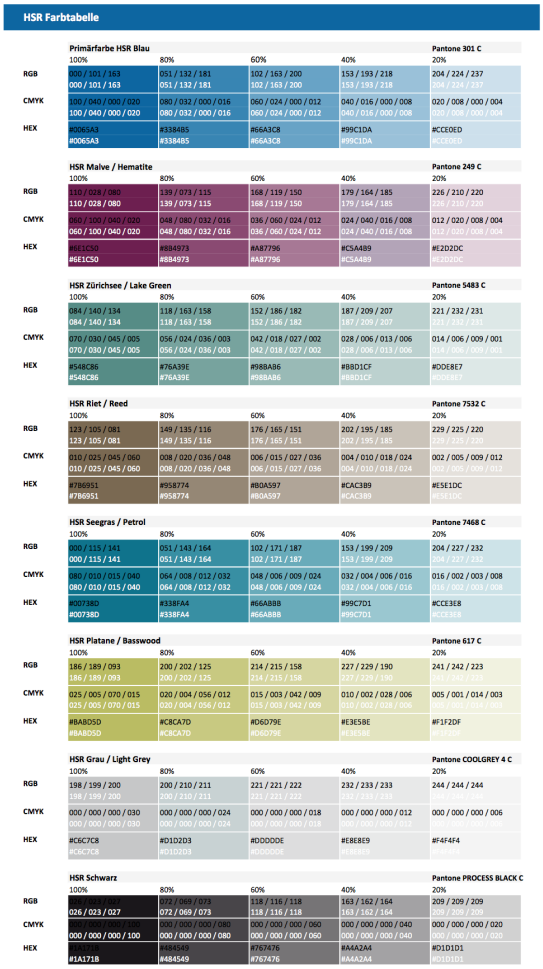
Abbildung 7.7: Finaler Prototyp

7.6 Layout

Für das Layout wurde ein Twitter Bootstrap verwendet. Durch die Verwendung von Bootstrap ist auch gleich sichergestellt, dass die Website Responsive ist. Durch das Layout von Bootstrap in Spalten werden bei schmalen Bildschirmen die Spalten einfach untereinander dargestellt. Zusätzlich wird mit Bootstrap bereits ein gewisses Grundstyling geliefert, dadurch ist das spätere stylen von Bedienelementen wesentlich einfacher. Die Navigation, welche von Boostrap mitkommt, ist auch sehr ähnlich wie in anderen Applikationen des Geometa Labs wie das OSMaxx. Ausserdem gehört zum Layout das HSR Logo, welches auf der Seite vorhanden sein muss. Dieses haben wir wie bei OSMaxx rechts platziert, damit ein gewisser Einheitslook erreicht wird.

7.7 Farbwahl

Für die Farbwahl soll das HSR Corporate Identity eingehalten werden. Zudem soll sich die Applikation in die restliche Applikationslandschaft des Geometa Lab einpassen. Für die Navigationsleiste wird deshalb das HSR Blau verwendet, wie es auch bei anderen Applikationen der Fall ist. Auch für wichtige Aktionsbuttons wird das HSR Blau verwendet. Ansonsten soll die Applikation ohne viele Farben auskommen. Somit soll ein möglichst HSR konformes Look and Feel erzielt werden.



HSR Farbtabelle

	Primärfarbe HSR Blau				Pantone 301 C
	100%	80%	60%	40%	20%
RGB	000 / 101 / 163 000 / 101 / 163	051 / 132 / 181 051 / 132 / 181	102 / 163 / 200 102 / 163 / 200	153 / 193 / 218 153 / 193 / 218	204 / 224 / 237 204 / 224 / 237
CMYK	100 / 040 / 000 / 020 100 / 040 / 000 / 020	080 / 032 / 000 / 016 080 / 032 / 000 / 016	060 / 024 / 000 / 012 060 / 024 / 000 / 012	040 / 016 / 000 / 008 040 / 016 / 000 / 008	020 / 008 / 000 / 004 020 / 008 / 000 / 004
HEX	#0065A3 #0065A3	#3384B5 #3384B5	#66A3C8 #66A3C8	#99C1DA #99C1DA	#CCE0ED #CCE0ED

	HSR Malve / Hematite				Pantone 249 C
	100%	80%	60%	40%	20%
RGB	110 / 028 / 080 110 / 028 / 080	139 / 073 / 115 139 / 073 / 115	168 / 119 / 150 168 / 119 / 150	179 / 164 / 185 179 / 164 / 185	226 / 210 / 220 226 / 210 / 220
CMYK	060 / 100 / 040 / 020 060 / 100 / 040 / 020	048 / 080 / 032 / 016 048 / 080 / 032 / 016	036 / 060 / 024 / 012 036 / 060 / 024 / 012	024 / 040 / 016 / 008 024 / 040 / 016 / 008	012 / 020 / 008 / 004 012 / 020 / 008 / 004
HEX	#6E1C50 #6E1C50	#8B4973 #8B4973	#AB7796 #AB7796	#C5A4B9 #C5A4B9	#E2D2DC #E2D2DC

	HSR Zürichsee / Lake Green				Pantone 5483 C
	100%	80%	60%	40%	20%
RGB	084 / 140 / 134 084 / 140 / 134	118 / 163 / 158 118 / 163 / 158	152 / 186 / 182 152 / 186 / 182	187 / 209 / 207 187 / 209 / 207	221 / 232 / 231 221 / 232 / 231
CMYK	070 / 030 / 045 / 005 070 / 030 / 045 / 005	056 / 024 / 036 / 003 056 / 024 / 036 / 003	042 / 018 / 027 / 002 042 / 018 / 027 / 002	028 / 006 / 013 / 006 028 / 006 / 013 / 006	014 / 006 / 009 / 001 014 / 006 / 009 / 001
HEX	#548C86 #548C86	#76A39E #76A39E	#98BAB6 #98BAB6	#BBD1CF #BBD1CF	#DDE8E7 #DDE8E7

	HSR Riet / Reed				Pantone 7532 C
	100%	80%	60%	40%	20%
RGB	123 / 105 / 081 123 / 105 / 081	149 / 135 / 116 149 / 135 / 116	176 / 165 / 151 176 / 165 / 151	202 / 195 / 185 202 / 195 / 185	229 / 225 / 220 229 / 225 / 220
CMYK	010 / 025 / 045 / 060 010 / 025 / 045 / 060	008 / 020 / 036 / 048 008 / 020 / 036 / 048	006 / 015 / 027 / 036 006 / 015 / 027 / 036	004 / 010 / 018 / 024 004 / 010 / 018 / 024	002 / 005 / 009 / 012 002 / 005 / 009 / 012
HEX	#7B6951 #7B6951	#958774 #958774	#B0A597 #B0A597	#CAC3B9 #CAC3B9	#E5E1DC #E5E1DC

	HSR Seegras / Petrol				Pantone 7468 C
	100%	80%	60%	40%	20%
RGB	000 / 115 / 141 000 / 115 / 141	051 / 143 / 164 051 / 143 / 164	102 / 171 / 187 102 / 171 / 187	153 / 199 / 209 153 / 199 / 209	204 / 227 / 232 204 / 227 / 232
CMYK	080 / 010 / 015 / 040 080 / 010 / 015 / 040	064 / 008 / 012 / 032 064 / 008 / 012 / 032	048 / 006 / 009 / 024 048 / 006 / 009 / 024	032 / 004 / 006 / 016 032 / 004 / 006 / 016	016 / 002 / 003 / 008 016 / 002 / 003 / 008
HEX	#00738D #00738D	#338FA4 #338FA4	#66ABBB #66ABBB	#99C7D1 #99C7D1	#CCE3E8 #CCE3E8

	HSR Platane / Basswood				Pantone 617 C
	100%	80%	60%	40%	20%
RGB	186 / 189 / 093 186 / 189 / 093	200 / 202 / 125 200 / 202 / 125	214 / 215 / 158 214 / 215 / 158	227 / 229 / 190 227 / 229 / 190	241 / 242 / 223 241 / 242 / 223
CMYK	025 / 005 / 070 / 015 025 / 005 / 070 / 015	020 / 004 / 056 / 012 020 / 004 / 056 / 012	015 / 003 / 042 / 009 015 / 003 / 042 / 009	010 / 002 / 028 / 006 010 / 002 / 028 / 006	005 / 001 / 014 / 003 005 / 001 / 014 / 003
HEX	#BABD5D #BABD5D	#C8CA7D #C8CA7D	#D6D79E #D6D79E	#E3E5BE #E3E5BE	#F1F2DF #F1F2DF

	HSR Grau / Light Grey				Pantone COOLGREY 4 C
	100%	80%	60%	40%	20%
RGB	198 / 199 / 200 198 / 199 / 200	200 / 210 / 211 200 / 210 / 211	221 / 221 / 222 221 / 221 / 222	232 / 233 / 233 232 / 233 / 233	244 / 244 / 244 244 / 244 / 244
CMYK	000 / 000 / 000 / 030 000 / 000 / 000 / 030	000 / 000 / 000 / 024 000 / 000 / 000 / 024	000 / 000 / 000 / 018 000 / 000 / 000 / 018	000 / 000 / 000 / 012 000 / 000 / 000 / 012	000 / 000 / 000 / 006 000 / 000 / 000 / 006
HEX	#C6C7C8 #C6C7C8	#D1D2D3 #D1D2D3	#DDDDDE #DDDDDE	#E8E8E9 #E8E8E9	#F4F4F4 #F4F4F4

	HSR Schwarz				Pantone PROCESS BLACK C
	100%	80%	60%	40%	20%
RGB	026 / 023 / 027 026 / 023 / 027	072 / 069 / 073 072 / 069 / 073	118 / 116 / 118 118 / 116 / 118	163 / 162 / 164 163 / 162 / 164	209 / 209 / 209 209 / 209 / 209
CMYK	000 / 000 / 000 / 100 000 / 000 / 000 / 100	000 / 000 / 000 / 080 000 / 000 / 000 / 080	000 / 000 / 000 / 060 000 / 000 / 000 / 060	000 / 000 / 000 / 040 000 / 000 / 000 / 040	000 / 000 / 000 / 020 000 / 000 / 000 / 020
HEX	#1A171B #1A171B	#484549 #484549	#767476 #767476	#A4A2A4 #A4A2A4	#D1D1D1 #D1D1D1

Abbildung 7.8: HSR Farbtabelle

7.8 Schriftart

Auch die Schriftart wird durch das HSR Corporate Design vorgegeben. Für Websiten dürfen nur die Schriften Frutiger oder Arial verwendet werden. Da für Frutiger zusätzliche Lizenzen benötigt werden, wird für diese Studienarbeit mit Arial gearbeitet.

7.9 Umsetzung des Designs

Es ist klar ersichtlich, dass die Applikation zu den HSR-Applikationen gehört, auch wenn gewisse Elemente etwas an Bootstrap erinnern.

Kapitel 8

Software Architektur

8.1 Architektur

Der RasterGeoConverter wurde konzeptionell in drei Schichten unterteilt.

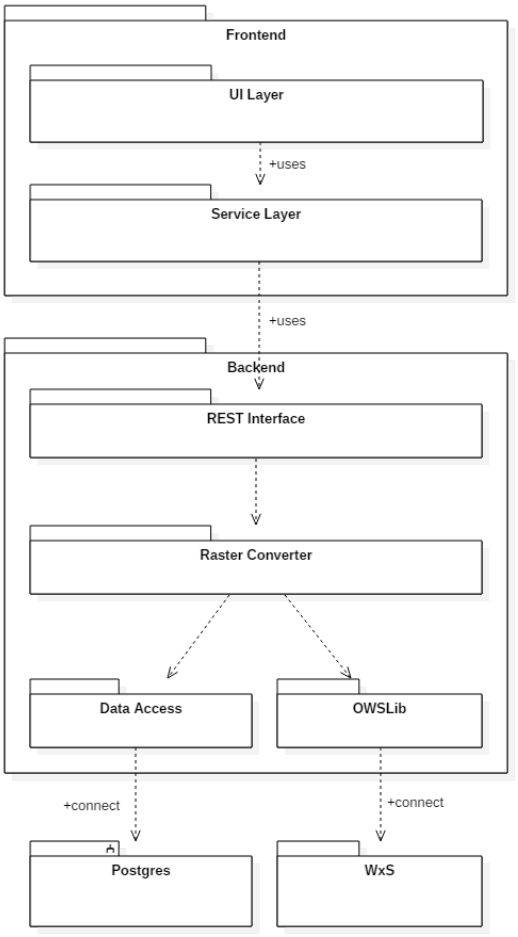


Abbildung 8.1: Architektur-Übersicht RasterGeoConverter

8.1.1 Frontend

Das Frontend besteht aus dem User Interface (UI) Layer, welcher die verschiedenen Komponenten des User Interfaces beinhalten. Zu diesen Komponenten gehört unter anderem auch die Kartenansicht, welche durch das Plugin Leaflet ermöglicht wird. Leaflet bezieht die Kartenmaterialien zur Ansicht selbstständig. Der UI Layer bezieht seine Daten über den Service Layer.
Der Service Layer verbindet dann das Frontend direkt zum REST-API des Backends. Im Service Layer werden alle Anfragen ans Backend gehandelt und die Daten encoded/decoded.

8.1.2 Backend

Das Backend erfüllt folgende Anforderungen

- Ansprechen der WxS-Anbieter
- Kartenbilder laden
- Kartenbilder transformieren
- Koordinatensystem umrechnen
- WxS-Parameter persistieren

Das Backend bietet für die Ansteuerung eine REST-Schnittstelle an. Über diese Schnittstelle wird die Raster Converter App angesprochen, welche dann über die OWS-Lib die Map-Services anspricht und die Karten lädt. Zudem verfügt der Raster Converter über einen Datenzugang für den Datenbankzugriff. Somit können Informationen persistiert werden.

8.1.3 Data Access

Der Data Access Layer ist für die Persistierung der Daten zuständig. Beim RasterGeoConverter wurde das über eine Datenbank realisiert.

8.1.4 WxS

Die Web Services für Karten sind öffentliche Server, welche Kartenmaterialien anbieten. Über Parameter in der Url können die nötigen Anfragen gestartet werden.

8.2 Systemübersicht

Der RasterGeoConverter wird über Docker-Container ausgeführt. Dazu wurden die einzelnen Systeme in eigene Container verpackt. Im Verteilungsdiagramm RasterGeoConverter sind die einzelnen Komponenten und ihre Abhängigkeiten ersichtlicht.

8.2.1 frontend Container

Im Frontend-Container werden alle nötigen Komponenten für das Frontend installiert und ausgeführt. Das Frontend interagiert mit dem Backend über eine REST-Schnittstelle. Für den Fileaustausch wird in der Serverumgebung ein gemeinsamer Ordner definiert, welcher direkt über das Netzwerk abrufbar ist.

8.2.2 backend Container

Im Backend-Container wird der Django-Server ausgeführt. Das RasterGeoConverter-Backend besitzt eine Datenbankanbindung zu einem DB-Container. Wir benutzen als Datenbank Postgres. Für die erfolgreiche Transformierung der Kartenbilder benutzt das Backend eine GDAL Partition, welche in einem externen Container mitgeliefert wird. Über einen Media-Ordner kann das Backend die geladenen Bilder für das Frontend bereitstellen.

8.2.3 DB Container

Im Datenbank Container wird eine Postgres-Datenbank eingebunden. Diese Datenbank persistiert Informationen seitens des Backends.

8.2.4 GDAL Container

Der GDAL Container liefert ein betriebsbereites GDAL.

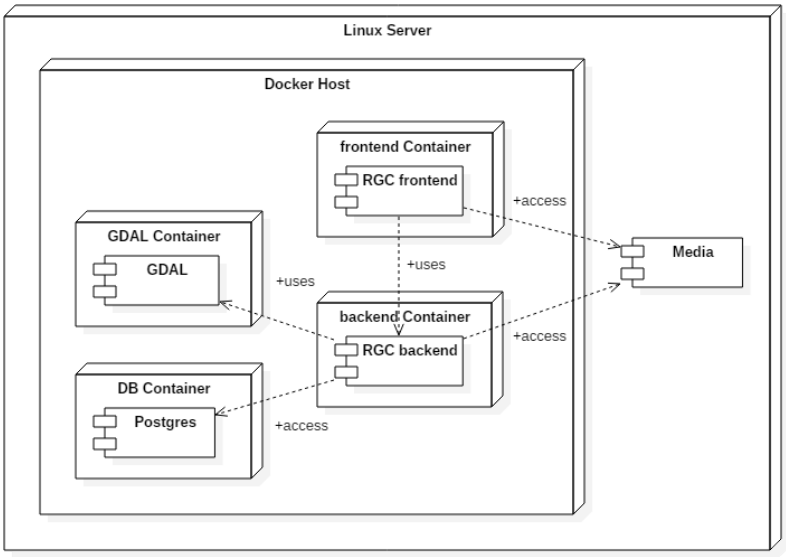


Abbildung 8.2: Verteilungsdiagramm RasterGeoConverter

8.3 Paketdiagramm

8.3.1 Frontend

Im Packetdiagramm 8.3 ist der Aufbau des Frontends ersichtlich.

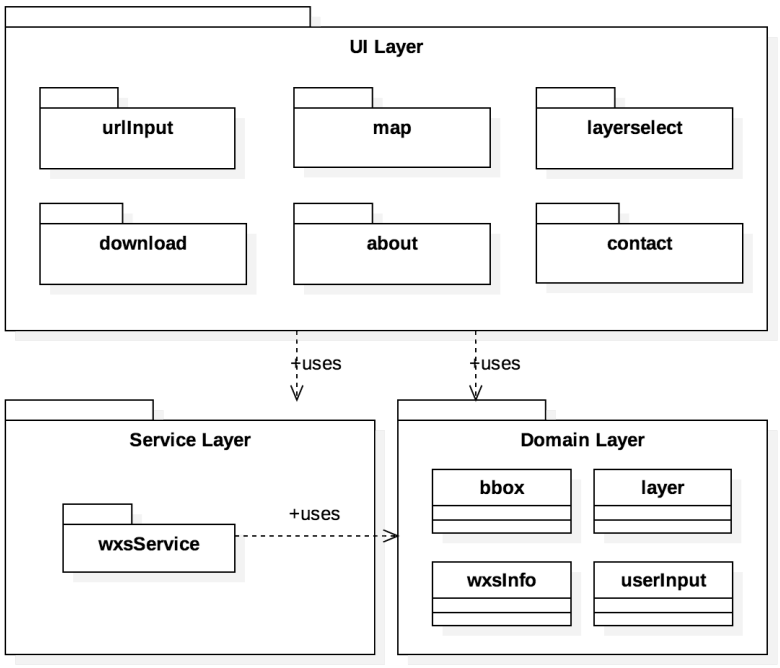


Abbildung 8.3: Packetdiagramm Frontend

UI Layer

In diesem Paket befinden sich die einzelnen Komponenten für das User Interface. Die Komponente urlInput ist für das Entgegennehmen der WMS URL verantwortlich und holt über den wxsService die Informationen vom Backend.
In der Komponente layerselect werden aus den Daten vom Backend die Layers eines WMS ausgelesen und für den User zur Auswahl gestellt.
Die map Komponente stellt die Karte dar. Zudem stellt diese Komponente die Möglichkeit die Koordinaten manuell einzugeben zur Verfügung.
Die download Komponente nimmt die userInputs entgegen und beinhaltet den Download Mechanismus. Dabei wird auch wieder über den wxsService die Kartengenerierung im Backend angestossen. Auch der Download-Link wird in dieser Komponente dargestellt.
In der Komponente about befindet sich die About-Seite.
Die Komponente contact beinhaltet die Contact-Seite.

Service Layer

In diesem Packet befindet sich der wxsService, welcher alle Requests an das Backend handelt. Der wxsService arbeitet eng mit dem Domain Layer zusammen welcher die einzelnen Models beinhaltet. Bei einer Anfrage an das Backend werden die Antwortdaten jeweils in das passende Model gefasst und erst dann an die UI Komponente zurückgegeben.

8.3.2 Backend

Das Paketdiagramm, Abbildung 8.4, zeigt den inneren Aufbau des Backends.

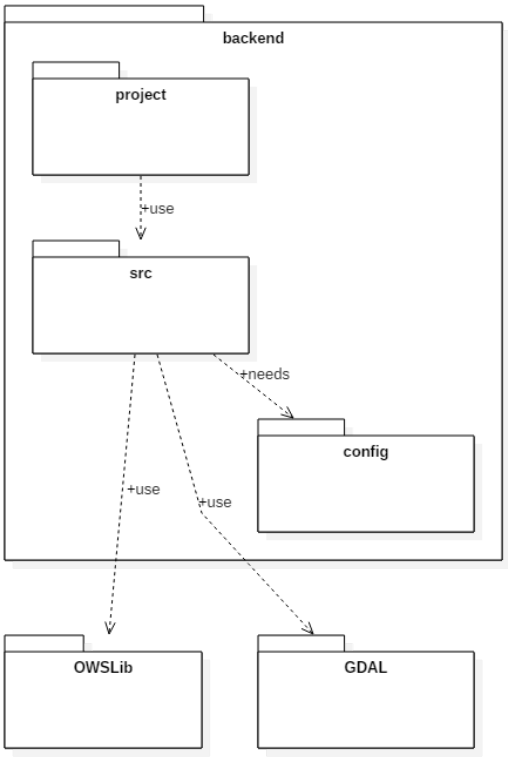


Abbildung 8.4: Paketdiagramm Backend

Paket: project

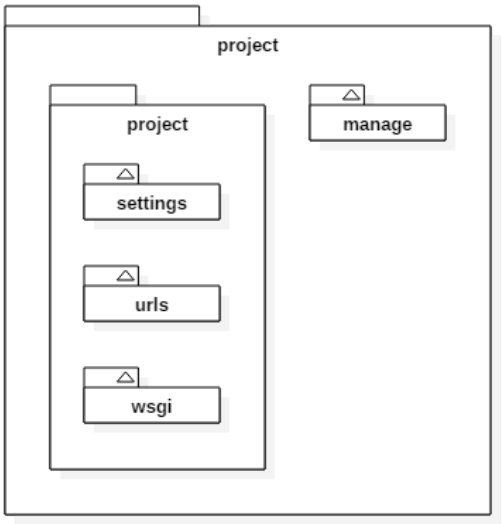


Abbildung 8.5: Paketdiagramm backend/project

Innerhalb des project-Pakets, Abbildung 8.5, sind alle Django-Kern-Komponenten platziert. Die Komponente "manage"bietet Zugang zu den Django internen Funktionen. Dazu gehören Datenbankerstellung sowie einen Debug-Server. Im Auslieferungszustand wird "manage"für die Ausführung nicht benötigt.
Im project/project-Ordner finden sich die *Settings*, *Urls* und *wsgi*. In den *Settings* sind alle Einstellungen für die erfolgreiche Ausführung des Serverprozesses zu finden.
In den *Urls* wurden alle Pfade für die REST-Schnittstelle definiert.
Und der *wsgi* beherbergt die Einstiegskonfiguratione für den externen Serverprozess. In unserem Falle Unicorn.

Paket: src

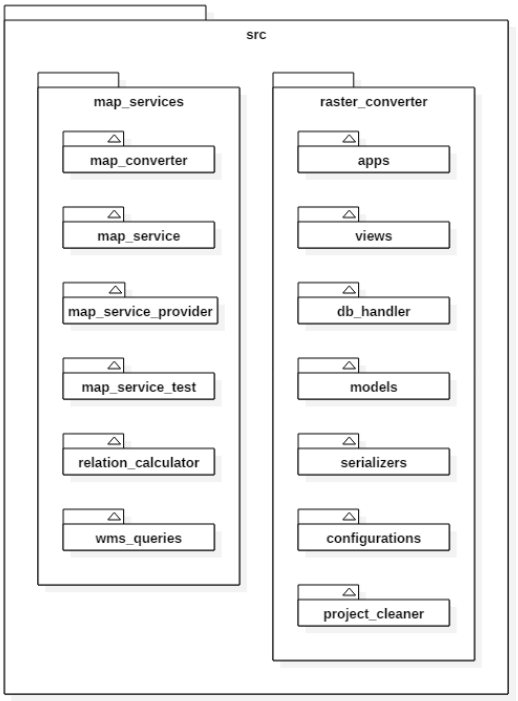


Abbildung 8.6: Paketdiagramm backend/src

Im *src*-Paket, Abbildung 8.6, befinden sich das Backend und seine interne Funktionen.
src wurde in zwei weitere Pakete unterteilt.

raster-converter
: Die RasterGeoConverter-Backend-App ist in diesem Paket vorzufinden. Hier werden alle Funktionen für die REST-Schnittstelle angeboten, die Datenmodelle-Serialisierer und die Konfigurationskomponenten wurden hier platziert.

map-services
: Alle Funktionen für den erfolgreichen Bezug von Karteninformationen, Transformationen und Umrechnungen sind innerhalb dieses Pakets vereinigt.

Mehr dazu ist im Kapitel *Implementation* zu finden.

8.3.3 Paket: config

Im Paket *configuration* sind alle laufenden Konfigurationen für den Betrieb des RasterGeoConverter-Backends zu finden. Die Konfigurationsparameter sind innerhalb von yaml-Dateien gespeichert.

8.3.4 Bibliothek-Pakete

Das Backend importiert die Pakete *OWSLib* und *GDAL*. Diese Bibliotheken liefern die nötigen Funktionen für den Abruf von Karteninformationen und Transformationen.

Kapitel 9

Implementation

In diesem Kapitel wird die Implementation des RasterGeoConverters beschrieben. Wie in der Projektzuteilung und auch in der Architektur ersichtlich, wird dieses Kapitel in zwei Teilbereiche unterteilt.

- Frontend
- Backend

9.1 Frontend

Das Frontend des RasterGeoConverters wurde in Angular 2 implementiert und ist somit mit einer modernen Technologie erstellt worden, wodurch es auch bei einer späteren Weiterentwicklung noch aktuell ist. Es soll dem User das Ansprechen eines WMS vereinfachen und gibt schwierigere Aufgaben ans Backend weiter.

9.1.1 Übersicht der Projektstruktur

In der Abbildung 9.1 ist ersichtlich, wie die Projektstruktur im Frontend aussieht. Dies ist eine typische Struktur, wie sie in einem Angularprojekt vorzufinden ist. Auf der obersten Ebene befinden sich die Konfigurationsfiles. Dazu gehören unter anderem das *package.json* und das *angular-cli.json* File. Mithilfe des *package.json* können die Packages mit der jeweiligen Version definiert werden, damit jeder Build reproduziert werden kann. Im *angular-cli.json* befinden sich Konfigurationen von Angular CLI, mit welchem der Webpack Server gestartet werden kann. Durch Webpack kann ganz einfach ein Developement Server gestartet werden, auf welchem das Projekt getestet werden kann.
Der Ordner *dist* wird generiert, wenn die App gebuildet wird. Er beinhaltet den minifizierten Code, welcher für die Distribution geeignet ist. Im Ordner *src* befindet sich die komplette Applikationsstruktur.

9.1.2 Applikationsstruktur

Die Applikationsstruktur wurde grundsätzlich von Angular CLI übernommen, entspricht jedoch auch den Style Guidelines von Angular 2 [1]. Unter *@types* befinden sich die Typings, welche erweitert werden mussten. Im Ordner *app* befindet sich die komplette Applikationslogik, wie dies

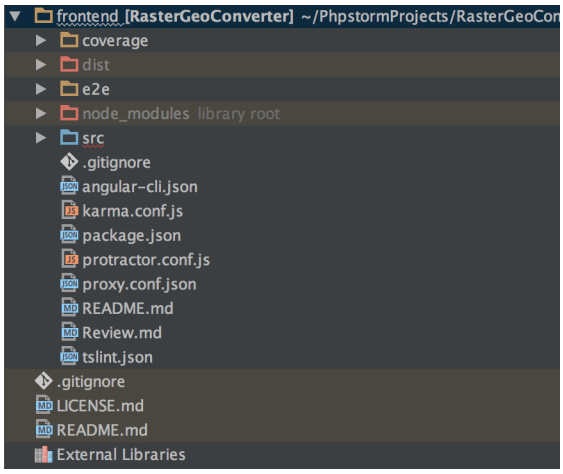


Abbildung 9.1: Projektstruktur Frontend

für Angular 2 Projekte empfohlen ist. Alle statischen Dateien, welche sich auf der Website befinden, sind im Ordner *assets* zu finden. Die Bilder befinden sich im Ordner *images*. Des Weiteren befinden sich auf dieser Ebene das Favicon und die Index-Seite.

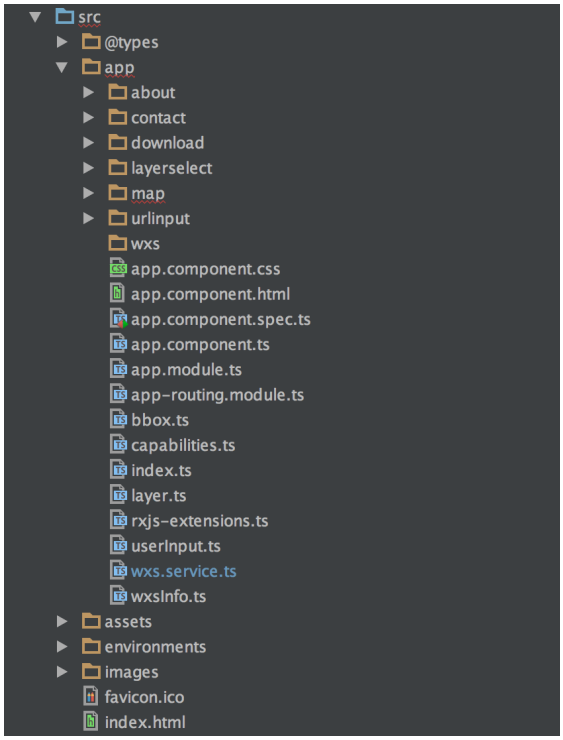


Abbildung 9.2: Applikationsstruktur Frontend

9.1.3 Models

Im Ordner *app* befinden sich unter anderem die Domain Models. Wie im Diagramm 9.3 ersichtlich ist, verwendet das Frontend vier Klassen. Diese Klassen werden für die Daten welche vom Backend kommen verwendet. Die Klasse *userInput* ist für die Haltung der vom User gewählten Einstellungen. In der Klasse *wxsInfo* werden die Informationen eines WxS gehalten.

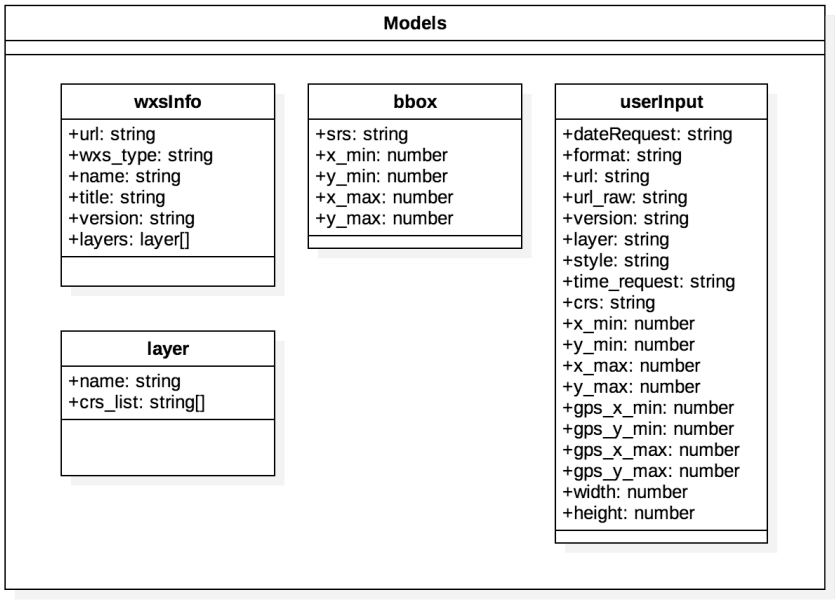


Abbildung 9.3: Models Frontend

9.1.4 Komponenten

Bei einem sauberen Aufbau einer Angular 2 App, erstellt man sogenannte Komponenten, welche über ein HTML-Tag geladen werden. Diese Komponenten sind zuständig für einen Teil der View und dessen Interaktionen.

urlinput

Die Komponente *urlinput* ist für das Inputfeld der WMS URL zuständig. Bei erfolgreicher WxS-Info Abfrage, wird die nächste Komponente, *layerselect*, geladen.

Abbildung 9.4: URL Input Komponente

layerselect

In dieser Komponente werden alle Layers einer WxS-Info Abfrage in ein Select-Feld geladen. Sobald ein Layer gewählt wird, wird die Komponente *map* geladen.

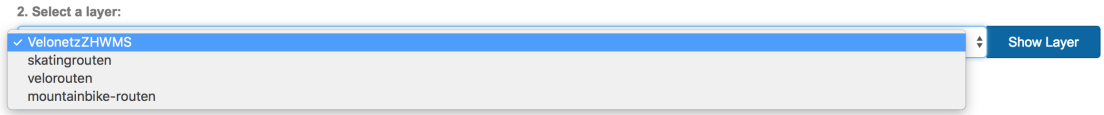


Abbildung 9.5: Layer Select Komponente

map

Diese Komponente beinhaltet die Kartenansicht und die manuelle Koordinateneingabe. Damit die Karte korrekt geladen wird, muss diese nach dem Konstruktor in der *ngOnInit()* Methode aufgerufen werden.

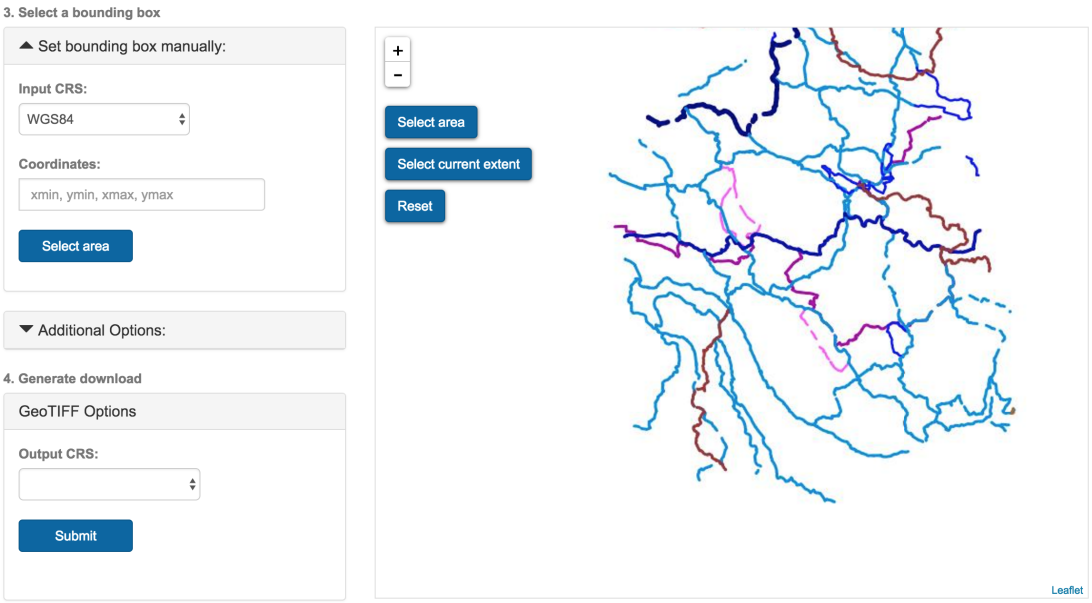


Abbildung 9.6: Map Komponente

download

Die *download* Komponente wird von der *map* Komponente aufgerufen, sobald eine Karte generiert wird.

Abbildung 9.7: Download Komponente

about

Die About-Seite ist in dieser Komponente implementiert.

contact

Die Contact-Seite ist in dieser Komponente implementiert.

9.1.5 Service

Der Service *wxs.service* ist für alle Anfragen ans Backend zuständig. Die einzelnen Komponenten rufen den globalen Service auf, um eine Anfrage ans Backend zu starten. Die Anfragen wurden

mit Observables implementiert. Sobald die Daten vom Backend eintreffen, werden sie in das entsprechende Model abgefüllt und an die Komponente zurück gegeben.

getUserinput()
: schickt die vom User eingegebene URL ans Backend um die vom User eingegebenen zusätzlichen Parameter herauszufiltern und füllt die Antwort in das *userInput* Model ab.

getWxsInfo()
: fragt das Backend nach den kompletten WxS-Informationen an und füllt diese anschliessend in ein *wxsInfo* Model ab.

getDownload()
: stösst die Generation des GeoTIFFs im Backend an und bekommt den Download-Pfad des generierten Bildes zurück.

transformCoordinates()
: lässt das Backend Koordinaten umwandeln und füllt diese anschliessend in ein *bbox* Model ab.

9.1.6 Testing

Da es in einem Frontend sehr schwierig ist sinnvolle Tests zu schreiben und auch der Wandel der Oberfläche sehr schnell vor sich geht, wurden keine Unit Tests geschrieben. Dafür wurde nach jedem Build das folgende Szenario durchgespielt, um das Frontend zu testen.

Test Szenario

1. Submit Button klicken ohne WMS URL
2. WMS URL eingeben und submitten
3. Layer auswählen und anzeigen lassen.
4. In der Karte zoomen, ein Rechteck von Hand zeichnen und wieder löschen
5. Optional - Ab dem Zeitpunkt, an dem die manuelle Eingabe möglich war: Manuelle Eingabe der Koordinaten
6. GeoTIFF Download anstossen (Submit)
7. GeoTIFF herunterladen

9.1.7 Installation

Um das Frontend starten zu können, muss Node.js installiert sein. Wenn dies installiert ist, können mit Hilfe von Node Package Manager (NPM) die benötigten Packages der Applikation installiert werden. Anschliessend kann der Development Server gestartet werden.

Mit diesem Befehl können alle Packages installiert werden.

```
npm install
```

Listing 9.1: Konsole: Alle Packages installieren

Mit diesem Befehl kann der Development Server mit den Proxykonfigurationen gestartet werden.

```
ng serve --proxy-config proxy.conf.json
```

Listing 9.2: Konsole: Development Server starten

Mit diesem Befehl kann das Projekt gebuildet werden. Die erstellten Artifakte werden in den Ordner */dist* gespeichert.

```
ng build
```

Listing 9.3: Konsole: Projekt builden

9.2 Backend

Das RasterGeoConverter-Backend ist für die Datengenerierung, -haltung und Kommunikation mit den Webservices für Karten zuständig. Es wurde wie in den vorherigen Kapiteln schon erwähnt, vollkommen in Python und dem Framework Django umgesetzt.

9.2.1 Übersicht der Projektstruktur

In der Ansicht 9.8 ist ersichtlich, wie die Projektstruktur des Backends aufgebaut ist. Es wurde unterteilt in *project* und *src*. Der *project*-Ordner enthält alle nötigen Konfigurationen und Einstellungen für das Verwalten der Django-App. Der *src*-Ordner enthält die eigentliche Backend-Logik des RasterGeoConverters.

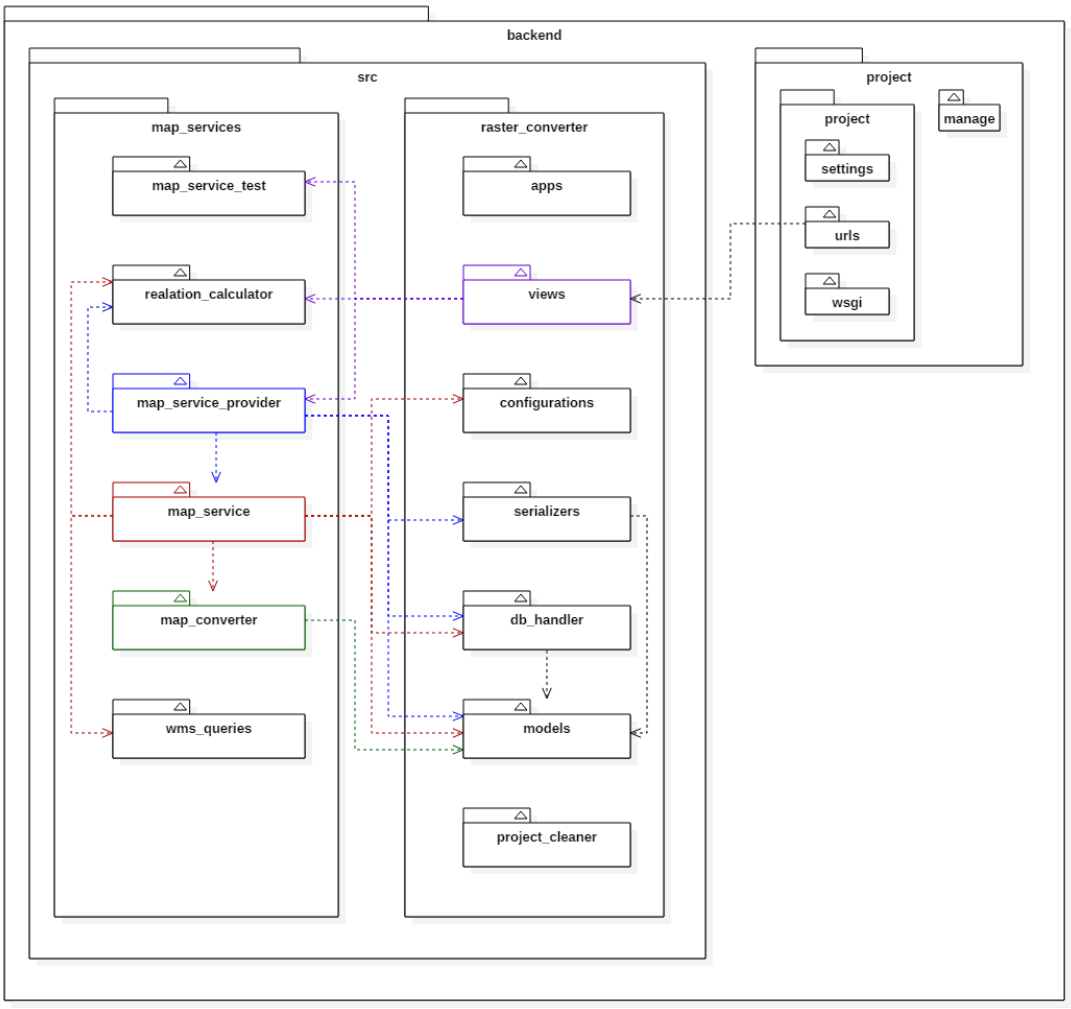


Abbildung 9.8: Implementation Übersicht

9.2.2 project

Das *manage.py*-Modul ist für die Django-Umgebung die Command-line-Ansteuerung. Damit kann die Strukturumgebung aufgebaut und verwaltet werden. Dazu nutzt das *manage* die Einstellungen unter *settings.py*. Von Haus aus liefert Django einen Debug-Server mit, um die Applikation lokal zu testen. Weitere Funktionen sind unter anderem die Erzeugung und Verwaltung der Datenbank, welche in den Settings definiert wurden.

```
python manage.py makemigrations
```

Listing 9.4: Konsole: Datenbank migrieren

Mit diesem Befehl erzeugt Django aus den Models eine neue Datenbankmigration oder ändert die vorhandenen Daten.

```
python manage.py migrate
```

Listing 9.5: Konsole: Datenbank erzeugen

Über den Befehl *migrate* wird die Datenbank erzeugt oder geändert. Enthaltene Daten werden auf das neue Model geändert.

```
python manage.py runserver
```

Listing 9.6: Konsole: Testserver ausführen

Damit wird der mitgelieferte Django-Server gestartet. Mit diesem Serve kann die Applikation lokal getestet werden.

9.2.3 src

In *src* ist die eigentliche Backend-Logik enthalten. Die Funktionen wurden hier in zwei Bereiche unterteilt:

Der raster-converter ist das Kernstück der Applikation und enthält alle nötigen Ansprechspunkte, Modelle, Serialisierer und Konfigurationen für die Ausführung der App.

Der Bereich map-services enthält die Logik für das Ansprechen und Kommunizieren der Webservices.

9.2.4 src.rasterconverter

apps

Diese Klasse ist einzig für die Benennung der *raster-converter*-App zuständig. Diese wird der AppConfig hinzugefügt, indem die Klasse von der AppConfig erbt.

```
class RasterConverterConfig(AppConfig):
    name = 'raster_converter'
```

Listing 9.7: Inhalt apps

views

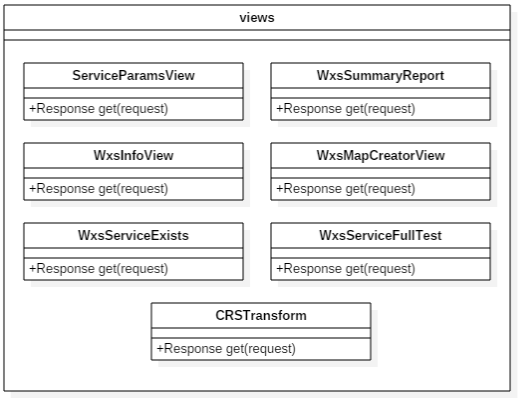


Abbildung 9.9: Klassendiagramm views

views ist das Herzstück des ServiceInterfaces. Hier wurde die REST-Schnittstelle umgesetzt. Dazu werden einzelne Klassen für die API-Anfragen zu Verfügung gestellt.

views.ServiceParamsView

Eine Anfrage an diese Schnittstelle extrahiert alle gültigen Parameter aus einer WxS-Url und soll dem Frontend eine gültige Url sowie die gewählten Parameter liefern.

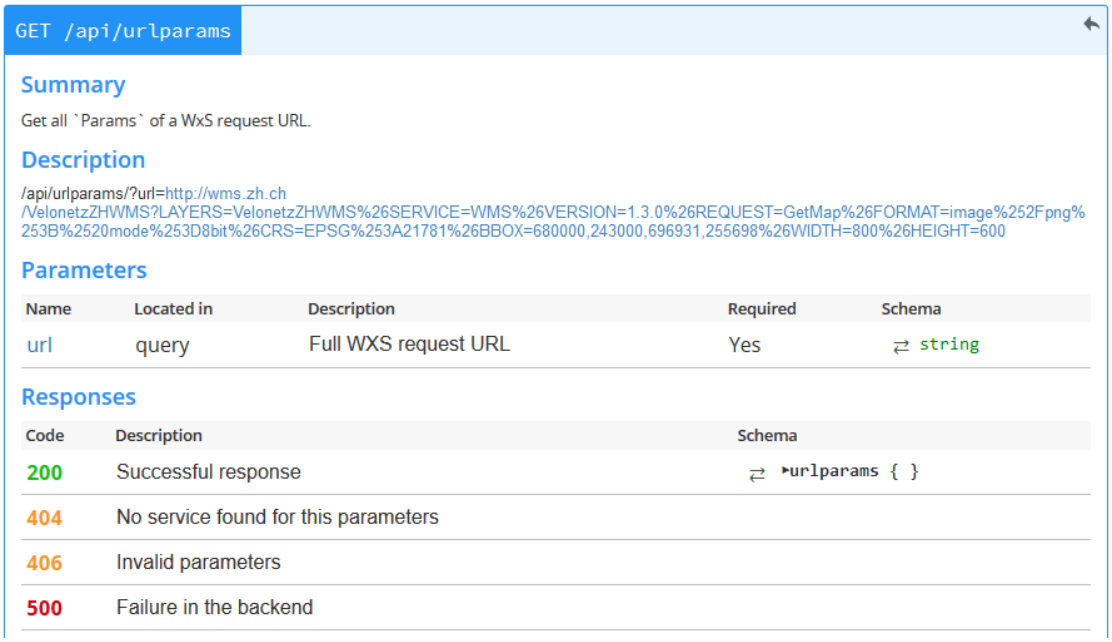


Abbildung 9.10: REST-API der ServiceParamsView

```
"id": 2,
"date_request": "2016-12-17T12:40:30.609201Z",
"user_key": "",
"url_raw": "(Siehe Description)",
"url": "http://wms.zh.ch/VelonetzZHWMS",
"version": "1.3.0",
"layer": "VelonetzZHWMS",
"format": "image/png",
"srs": "EPSG:21781",
"x_min": 680000.0,
"y_min": 243000.0,
"x_max": 696931.0,
"y_max": 255698.0,
"gps_x_min": 8.49707752733227,
"gps_y_min": 47.33301034975646,
"gps_x_max": 8.72380051561903,
"gps_y_max": 47.44493221874505,
"width": 800,
"height": 600,
"style": "",
"transparent": "False",
"bgcolor": "0xFFFFFF",
"exceptions": "",
"time": "",
"elevation": "",
"vendor_params": ""
```


Listing 9.8: Get-Request: Resultat Parameteranfrage

views.WxsInfoView

Die *WxsInfoView* liefert über eine gültige Url alle unterstützten Parameter eines Webservices an das Frontend. So kann das Frontend dem Benutzer die Auswahl der Parameter präsentieren.

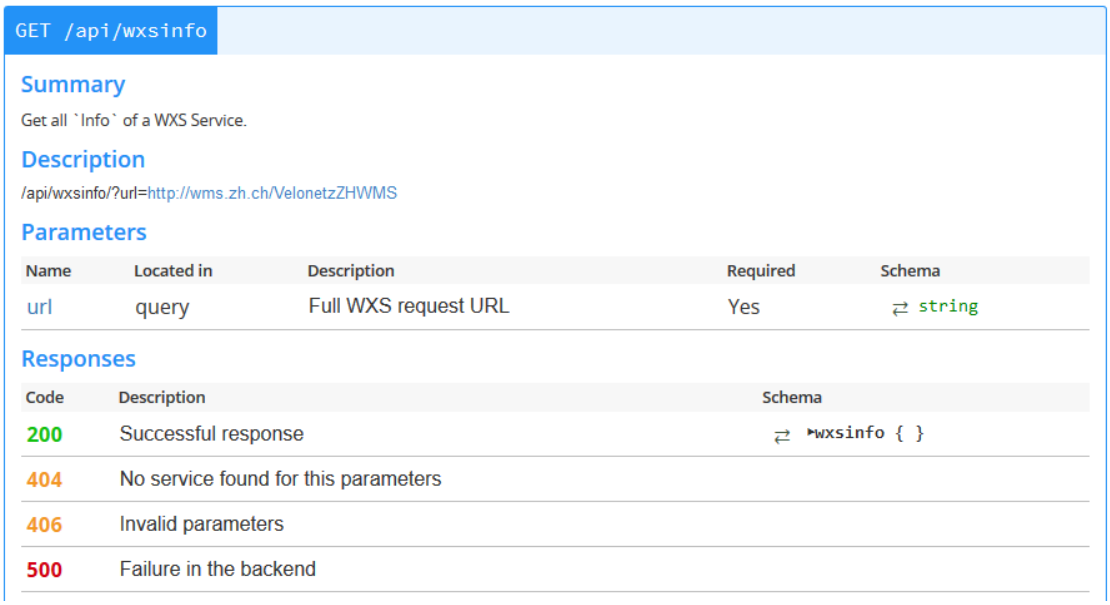


Abbildung 9.11: REST-API der WxsInfoView

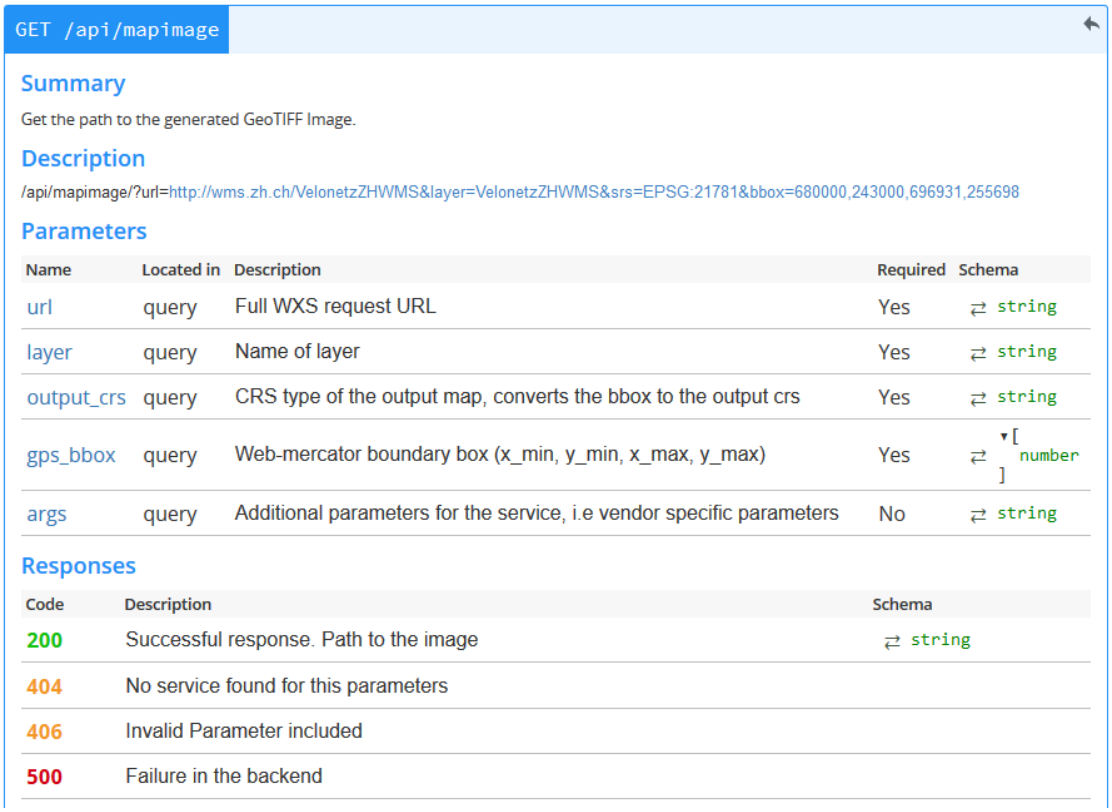
```
"url": "http://wms.zh.ch/VelonetzZHWMS",
    "wxs_type": "WMS",
    "title": "Velo-, Skating- und Mountainbikerouten",
    "version": "1.3.0",
    "layers": [
        {
            "name": "VelonetzZHWMS",
            "crs_list": [
                {
                    "x_min": 8.15722,
                    "y_min": 47.1417,
                    "x_max": 9.03748,
                    "y_max": 47.7124,
                    "srs": "WGS84"
                },...
```


Listing 9.9: Get-Request: Resultat Serviceinformationen

Die Layers wurden aus Platzgründen verkürzt.

views.WxsMapCreatorView

Die *WxsMapCreatorView* erzeugt über eine gültige Url und den nötigen Parameter eine Kartenanfrage an den gewünschten Service. Die geladene Karte wird dann automatisch in ein GeoTIFF tranformiert und in den Mediaordner gespeichert. Als Rückgabewert wird der Download-Link geliefert.



GET /api/mapimage

Summary

Get the path to the generated GeoTIFF Image.

Description

/api/mapimage/?url=http://wms.zh.ch/VelonetzZHWMS&layer=VelonetzZHWMS&srs=EPSG:21781&bbox=680000,243000,696931,255698

Parameters

Name	Located in	Description	Required	Schema
url	query	Full WXS request URL	Yes	string
layer	query	Name of layer	Yes	string
output_crs	query	CRS type of the output map, converts the bbox to the output crs	Yes	string
gps_bbox	query	Web-mercator boundary box (x_min, y_min, x_max, y_max)	Yes	[number]
args	query	Additional parameters for the service, i.e vendor specific parameters	No	string

Responses

Code	Description	Schema
200	Successful response. Path to the image	string
404	No service found for this parameters	
406	Invalid Parameter included	
500	Failure in the backend	

Abbildung 9.12: REST-API der WxsMapCreatorView

views.WxsServiceExists

Der *WxsServiceExists* startet eine Testreihe, welche über die wxs-data.yml konfiguriert wurde. Damit werden alle erfassten Webservices geprüft, ob diese existieren.


```
{
   "id": 55,
   "TestCase": [
       {
           "id": 99,
           "wxstype": "None",
           "name": "Swiss Admin Server",
           "url": "http://wms.geo.admin.ch/",
           "version": "1.3.0",
           "service_exists": true,
           "map_path": "None",
           "test_layer": "None",
           "test_srs": "",
           "x_min": 0.0,
           "y_min": 0.0,
           "x_max": 0.0,
           "y_max": 0.0,
           "wxstestsummary": 55
       },
```


Listing 9.10: Get-Request: Resultat ServiceTest

Dieser Output wird aus einer solchen Anfrage generiert. *WxsServiceFullTest* generiert zusätzlich noch eine Karte um zu prüfen, ob diese Operation unterstützt wird.

views.CRSTransform

Über CRSTransform kann eine BBox von einem Koordinatensystem in ein anderes konvertiert werden. Dazu wird die Start-BBox mit dem Koordinatensystem, sowie das Ziel-Koordinatensystem benötigt.

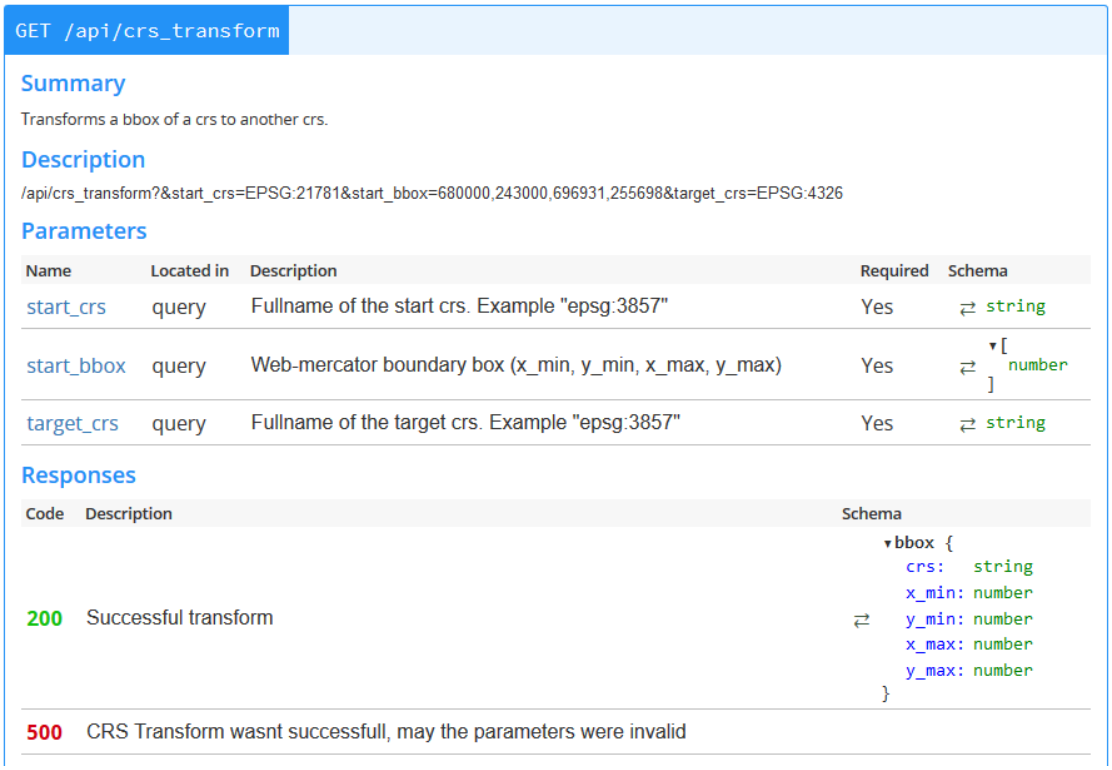


Abbildung 9.13: REST-API der CRSTransform

models und serializers

Die *models* sind bereitgestellte Datenmodelle für die Datenhaltung, sei es zum Übertragen über die REST-Schnittstelle und/oder Persistierung in der Datenbank. Mittels den *serializers* werden die Daten in JSON-Files zur Übertragung serialisiert.

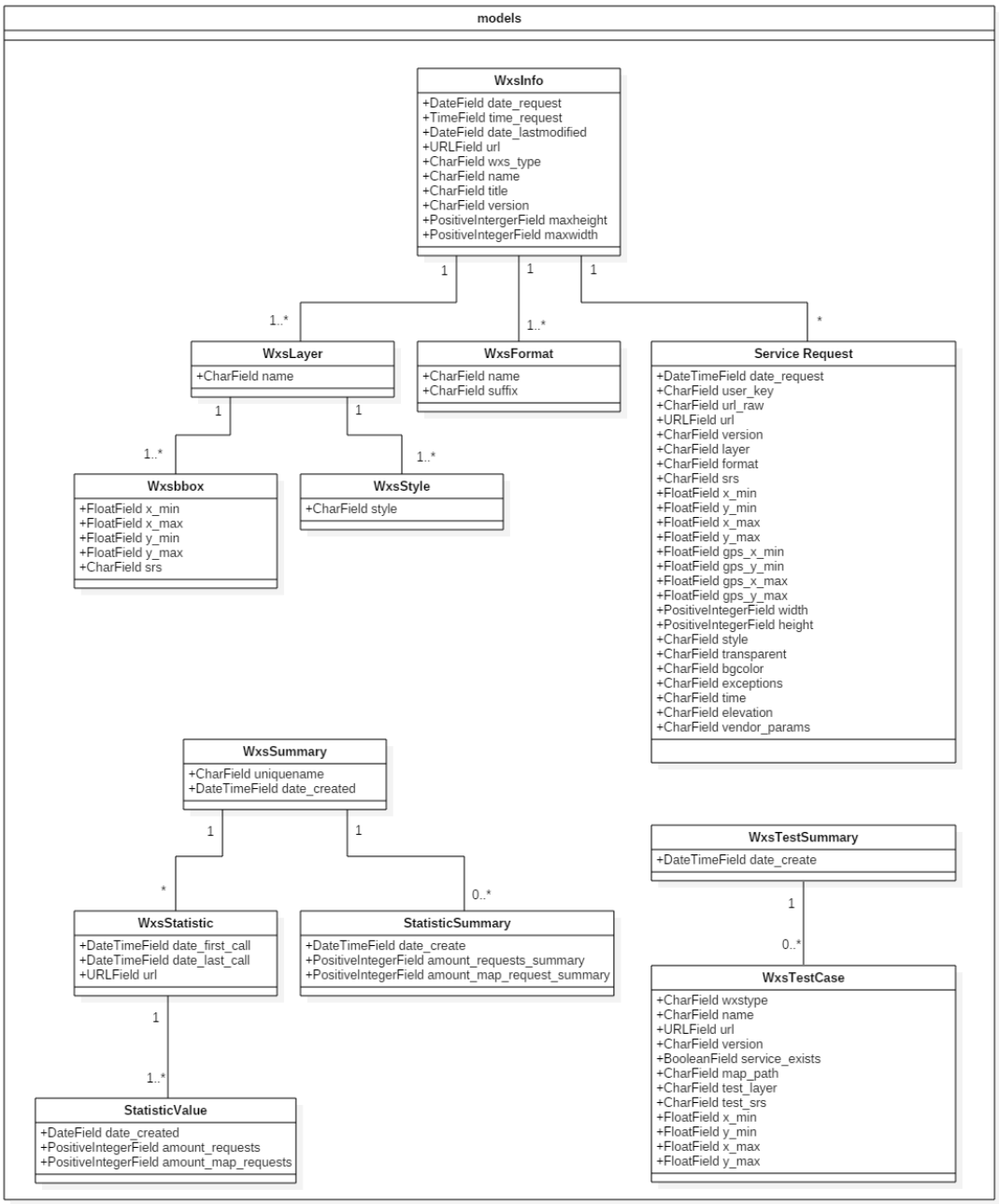


Abbildung 9.14: Übersicht Models

db-handler

Der db-handler ist zuständig für die Verwaltung der Datenbank und liefert Funktionen für die Datenpersistierung.

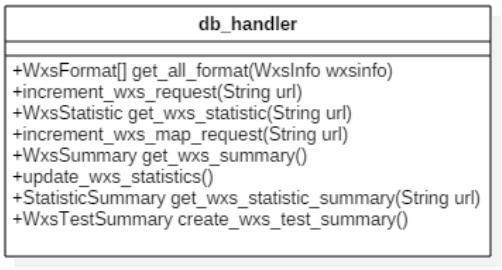


Abbildung 9.15: db-handler Klassendiagramm

configurations

In dieser Klasse befinden Funktionen für die Konfigurationsparameter. Alle Werte für die Konfiguration sind aus der yaml-Datei *config/rgc-config.yml*.

9.2.5 srs/mapservices

map-service-provider

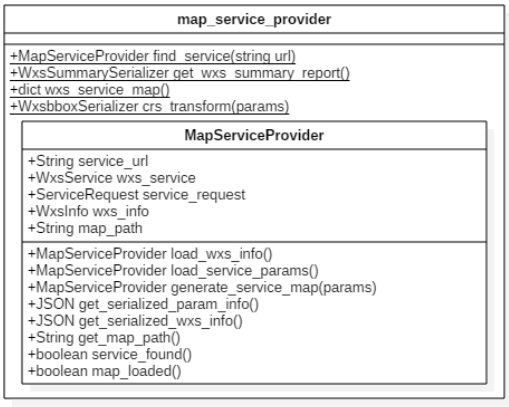


Abbildung 9.16: map-service-provider Klassendiagramm

Der *map-service-provider* wurde nach dem Builder-Pattern gestaltet und erlaubt es so für die verschiedenen Anfragen den richtigen MapService zu finden und die richtige Ausführfunktion zu wählen. Dazu werden die allgemeinen Funktionen zu Verfügung gestellt. Über find-servcie(url) wird der WxS-Service ausgewählt und der Provider erstellt, sowie zurückgegeben.

load-wxs-info(): Sucht und speichert die WxS-Parameter

load-service-params(): Extrahiert und speichert die im Request befindlichen Parameter

über die get-serialized-Funktionen können die Daten serialisiert werden

get-map-path: Gibt den Pfad der Karte als downloadbare URL zurück.

service-found: Falls ein Service auf eine Url gefunden wurde, gibt diese Funktion *true* zurück

map-loaded: Wenn eine Karte erfolgreich geladen wurde und ein Pfad vorhanden ist, gibt diese Funktion *true* zurück.

map-services

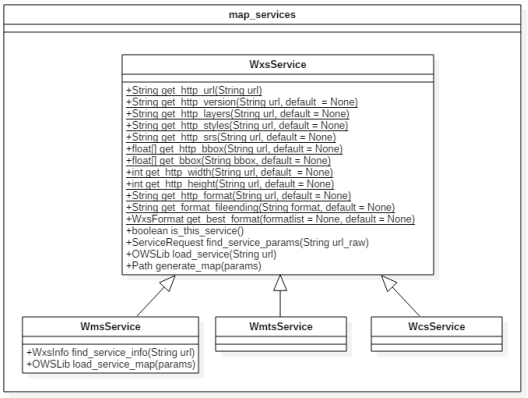


Abbildung 9.17: map-services Klassendiagramm

Map-services beinhaltet alle Funktionen für eine erfolgreiche Anfrage an den richtigen WxS. Dazu werden Funktionen für die Parameter Extraktion aus einer Url zu Verfügung gestellt, sowie Klassen für die Kommunikation mit den WxS. Alle WxS-Klassen erben von der ÏnterfaceKlasse WxsService. WxsService liefert alle Funktionen, welche für den erfolgreichen Aufruf nötig sind und von den Service-Klassen implementiert werden müssen.

Damit die Parameter aus der Url extrahiert werden können, wurden Funktionen mit regulären Ausdrücken in WxsService erstellt. Falls diese sich für einen WxS unterscheiden, wird diese Funktion in der abgeleiteten Klasse überschrieben.

Der RasterGeoConverter wurde in der aktuellen Version für den WMS implementiert. Dazu wurde die Klasse *WmsService* erstellt. Diese Klasse greift dazu auf die Funktionen der OWSLib zu.

map-converter

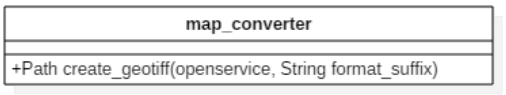


Abbildung 9.18: map-converter Klassendiagramm

Der *map-converter* ist für die Erzeugung des GeoTIFF's zuständig. Dazu bietet diese Klasse die nötigen Funktionen an, welche über GDAL die Karte in das richtige Format umwandelt.

relation-calculator

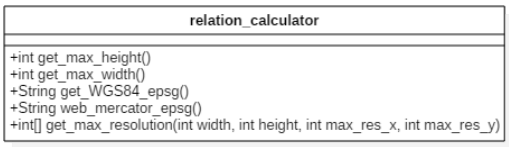


Abbildung 9.19: relation-calculator Klassendiagramm

Der *relation-calculator* bietet Funktionen für die allgemeinen Umrechnungen an. Unter anderem gehören dazu:

- get-max-height: Berechnet die grösstmögliche Höhe für Auflösung
- get-max-width: Berechnet die grösstmögliche Breite für die Auflösung
- get-WGS84: Gibt die EPSG-Nr für das GPS-Koordinatensystem zurück
- web-mercator-epsg: Gibt die EPSG-Nr für das web-mercator-Koodinatensystem zurück
- get-max-resolution: Berechnet die grösstmögliche Auflösung einer Karte. Dazu wird die Höhe und Breite des Bildes aus der BBox genommen mithilfe der Auflösung verrechnet. Die Auflösung der Zielkarte ist dann der Output dieser Funktion

Die Berechnung für die optimale Auflösung wird über diesen Algorithmus durchgeführt

```
def get_max_resolution(width, height, max_res_width, max_res_height):
    assert width > 0
    assert height > 0
    assert max_res_width > 0
    assert max_res_height > 0

    if width > height:
        res_height = height * max_res_width / width
        res_width = max_res_width
    else:
        res_height = max_res_height
        res_width = max_res_height * width / height

    if res_height > max_res_height:
        res_width = res_width * max_res_height/ res_height
        res_height = max_res_height

    if res_width > max_res_width:
        res_height = res_height * max_res_width / res_width
        res_width = max_res_width

    return res_width, res_height
```

Listing 9.11: relation-calculator: Berechnung der maximalen Auflösung

Im ersten Teil wird geprüft, ob alle Parameter vorhanden sind. Zweitens, die Verhältnisse der Auflösung wird zum Verhältnis der Seitengrössen angepasst. Drittens, die Auflösung wird angepasst, sobald eine Seite grösser als das Maximum sein sollte.

wms-queries

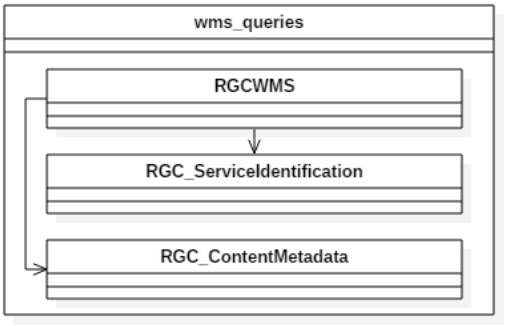


Abbildung 9.20: wms-queries Klassendiagramm

Die *wms-queries*-Klasse wurde aus der OWSLib abgeleitet. Die bisherige Funktion wurde ergänzt um die maximale Auflösung der Services abzufragen. Ansonsten bietet diese Klasse alle nötigen Funktionen für die Service-Abfrage und Kartengenerierung.

map-service-test

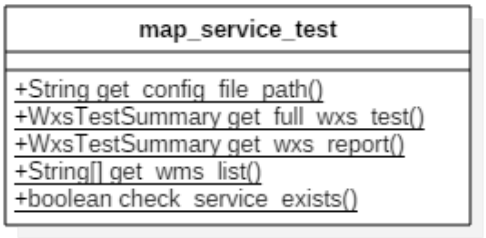


Abbildung 9.21: map-service-test Klassendiagramm

Die *map-service-test*-Klasse enthaltet Funktionen für die Ausführung der Service-Tests. Es testet alle Services, welche im Konfigurations-Yaml erfasst wurden.

Kapitel 10

Weiterentwicklung

Dieses Kapitel wurde schon im Kapitel 5 unter der Sektion *Weiterentwicklung* abgehandelt.

Kapitel 11

Projektmanagement

11.1 Rollen und Verantwortlichkeiten

11.1.1 Prof. Stefan Keller

Prof. Stefan Keller, Mitarbeiter des Instituts für Software (IFS) und Dozent an der HSR. Als Projektbetreuer und Interessensvertreter ist er für die Rahmenbedingungen und den reibungslosen Ablauf des Projektes verantwortlich.

11.1.2 Nicola Jordan

Nicola Jordan arbeitet im Institut für Software (IFS). Er war Teil des Betreuerteams. Mit seinem grossen Wissen über Programmiersprachen und Frameworks konnte er das Projektteam aktiv unterstützen.

11.1.3 Céline Ott

Céline Ott ist Vollzeitstudentin an der Hochschule für Technik Rapperswil. Sie übernimmt im Entwicklerteam die Verantwortung für das Frontend des RasterGeoConverters.

11.1.4 Diego Etter

Diego Etter ist Teilzeitstudent an der HSR und arbeitet parallel zum Studium als Werkstudent in der Siemens AG am Standort Wallisellen. Als Teil des Entwicklerteams übernimmt er die Verantwortung für die Entwicklung des RasterGeoConverter-Backends.

11.2 Prozessmodell

Für die Projektabwicklung richten wir uns nach dem Rational Unified Process (RUP) Vorgehensmodell für Softwareentwicklung. RUP unterteilt das Projekt in vier Phasen: Inception, Elaboration, Construction, und Transition.
Diese Phasen finden sich auch in unserer Projektplanung wieder. Der Grund unserer Wahl dieses Vorgehensmodell liegt insbesondere am festen Zeitplan, sowie den bisherigen gesammelten Erfahrungen im Software-Engineering-Projekt.

11.2.1 RUP Phasen

Inception

Diese erste Konzeptionsphase dient dem Ausformulieren einer Vision, eines klaren Zieles sowie der Erstellung eines rudimentären Anwendungsfallmodelles, das die wesentliche Funktionalität beschreibt sowie einer tentativen/provisorischen Architektur. Darüber hinaus werden die wesentlichsten Risiken identifiziert und die Ausarbeitungsphase geplant. Sie resultiert im Lifecycle Objective Milestone.

Elaboration

In dieser Phase werden der Architekturprototyp sowie eine detaillierte Beschreibung für etwa 80 Prozent der Anwendungsfälle ausgearbeitet. Hier erfolgt die Planung der Konstruktionsphase.

Construction

Nachdem die Architektur ausgearbeitet wurde, konzentriert sich diese Phase auf die Entwicklung und das Testen des Produktes. Hier entsteht die erste lauffähige Version der Software.

Transition

Übergabephase und Auslieferung der Software.

11.3 Infrastruktur

11.3.1 Versionierung

Dokumentation

Alle Dokumente der Studienarbeit werden im git der HSR versioniert und gespeichert. Zugriff erhalten alle Beteiligten des Projektes. `https://git.hsr.ch/git/SA_MSP`

Source Code

Der Sourcecode des Frontends sowie des Backends wird über das Github versioniert und gespeichert. Dazu wurde ein privates Repository erzeugt. Zugriff erhalten alle Beteiligten am Projekt. `https://github.com/celineellenanna/RasterGeoConverter`

11.3.2 Entwicklungswerkzeuge

Zweck	Tool	Version
IDE Backend	PyCharm	2016.2.3
IDE Frontend	PhpStorm	2016.2.2
Versionierung	Git	2.9.0
Projektmanagement	Redmine	3.2.1
Dokumentation	LaTex	-

Tabelle 11.1: Entwicklungswerkzeuge im Einsatz

11.4 Projektplan

11.4.1 Aufwandschätzung

Projektdauer	14 Wochen
Zeit pro Mitarbeiter	17 Stunden pro Woche
Gesamtaufwand	480 Stunden
Projektstart	19. September 2016
Projektende	20. Dezember 2016

Tabelle 11.2: Aufwandschätzung RasterGeoConverter

11.4.2 Phasen / Iteration

Inception 1

Zeitrahmen der Phase: 20. September - 27. September
Inhalt der Phase

- Kickoff
- Arbeitsumgebung einrichten
- Projektmanagement erstellen

Inception 2

Zeitrahmen der Phase: 28. September - 04. Oktober
Inhalt der Phase

- Technologien sammeln
- Risikoplanung
- Zeitplan final

Elaboration 1

Zeitrahmen der Phase: 05. Oktober - 18. Oktober
Inhalt der Phase

- Use Cases

- GUI Skizze
- Allgemeine Anforderungen

Elaboration 2

Zeitrahmen der Phase: 19. Oktober - 01. November
Inhalt der Phase

- Prototypen erstellen
- Schnittstellen erstellen
- Construction planen

Construction 1

Zeitrahmen der Phase: 02. November - 15. November
Inhalt der Phase

- User Interface erstellen und testen
- Backend erstellen
- WMS ansprechen

Construction 2

Zeitrahmen der Phase: 16. Oktober - 29. November
Inhalt der Phase

- UI mit Backend verbinden
- WMS testing
- Download einrichten

Construction 3

Zeitrahmen der Phase: 30. November - 13. Dezember
Inhalt der Phase

- Reserve
- RasterGeoConverter fertigstellen
- Video und Anleitung erstellen

Transition

Zeitrahmen der Phase: 14. Dezember - 23. Dezember
Inhalt der Phase

- Dokumentation fertigstellen
- RasterGeoConverter übergeben

11.4.3 Meilensteine

M1 Techwahl

Termin: 27. Oktober 2016
Beschreibung: Eingesetzte Technologien definiert
Artefakte:

- Elaboration
- Dokumentation

M2 End of Elaboration

Termin: 01. November 2016
Beschreibung: Die Elaboration wurde beendet
Artefakte:

- Prototypen für die kritischen Funktionen
- Dokumentation Use Cases
- Dokumentation Schnittstellen
- Dokumentation Aufbau RasterGeoConverter

M3 Basic-Operations

Termin: 22. November 2016
Beschreibung: Die Grundfunktionen wurden umgesetzt
Artefakte:

- Funktionstüchtige App
- Anfragen an WxS funktionieren
- Download möglich

M4 Feature Freeze

Termin: 01. Dezember 2016
Beschreibung: Die definierten Features werden nicht mehr erweitert und es werden keine weiteren Features integriert
Artefakte:

- Protokoll Betreuermeeting

M5 Code Freeze

Termin: 13. Dezember 2016
Beschreibung: Ende der Implementation. Es kommen keine weiteren Codeteile hinzu
Artefakte:

- Protokoll Betreuermeeting

M6 Abgabe

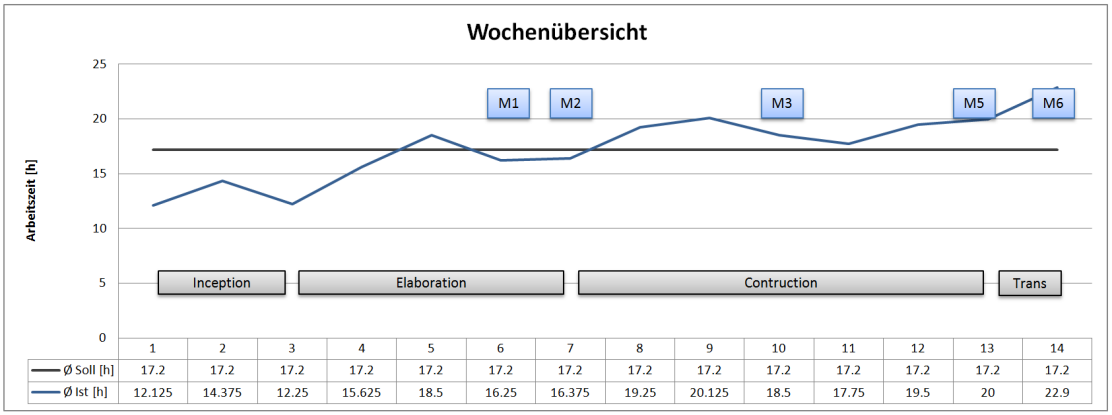
Termin: 23. Dezember 2016
Beschreibung: Abgabe der Studienarbeit
Artefakte

Dokumentation

RasterGeoConverter im DockerContainer

11.4.4 Auswertung

Wochenübersicht

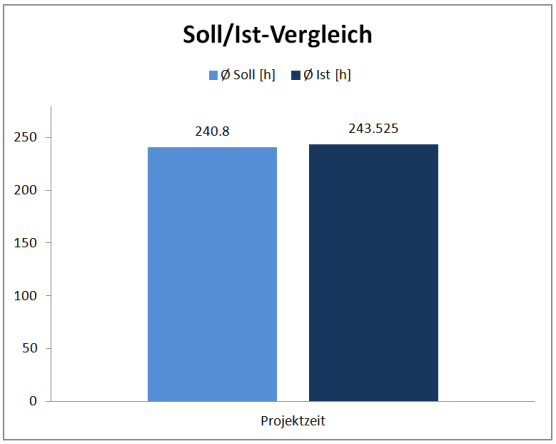


	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Ø Soll [h]	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2	17.2
Ø Ist [h]	12.125	14.375	12.25	15.625	18.5	16.25	16.375	19.25	20.125	18.5	17.75	19.5	20	22.9

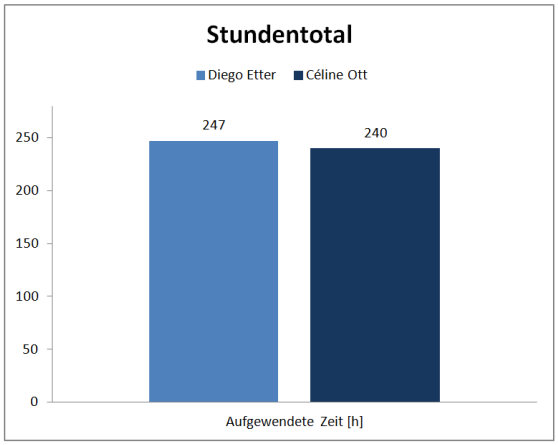
Abbildung 11.1: Stundendurchschnitt pro Person

Aus der Grafik ist ersichtlich, dass in der Anfangsphasen viele Abklärungen für die Durchführung des Projektes nötig waren. Zuerst mussten wir ermitteln, welche Technologien eingesetzt werden sollten und mussten uns diese aneignen. Zudem war der Aufwand im Studium höher, sodass wir auch Zeit für die Module aufwenden mussten. Im späteren Verlauf konnten wir uns dann mehr und mehr auf die Studienarbeit konzentrieren.

Statistiken



(a) Soll/Ist-Vergleich Projektzeit



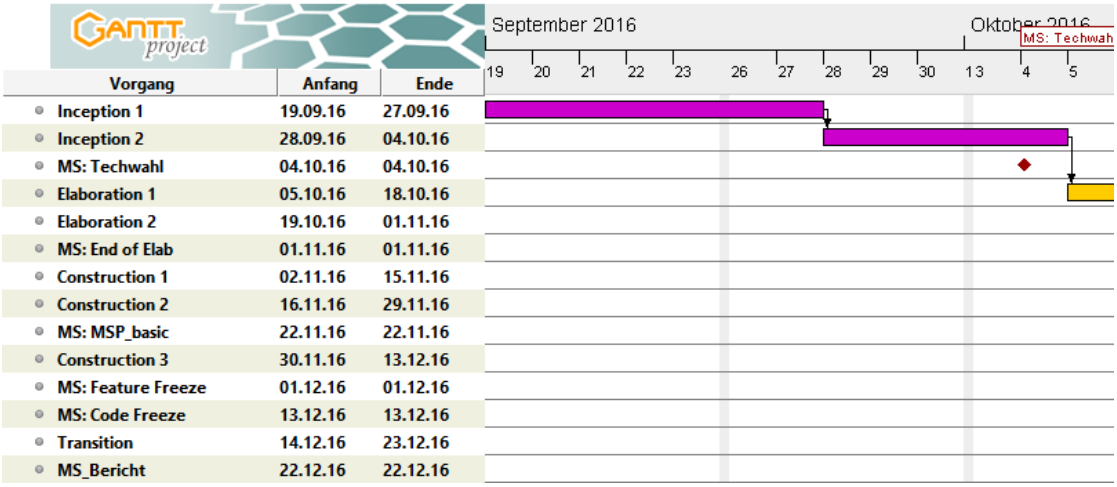
(b) Stundentotal Einzelperson

Die Zeitauswertung zeigt auf, dass wir die durch die Einhaltung des Zeitplanes die angestrebten Zeiten gut einhalten konnten. Der Unterschied zwischen den Projektmitgliedern ist dadurch zu erklären, dass Céline Ott zusätzlichen Aufwand in den Studienmodulen betreiben musste. Diego Etter konnte durch seinen schlanken Stundenplan mehr Stunden für die Studienarbeit aufwenden. Die Arbeitsaufteilung war aber nach den Einschätzungen des Projektteams fair.

11.5 Zeitplan

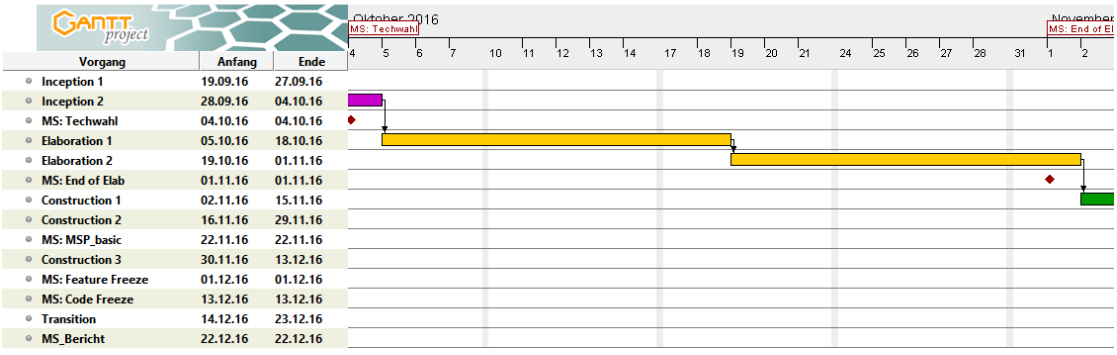
11.5.1 Phasenplanung

Zu Beginn haben wir die Phasen innerhalb des Projektes geplant und die Meilensteine gesetzt. Die einzelnen Tickets wurden dann über das Betreuermeeting und Teammeetings erstellt.



Vorgang	Anfang	Ende
Inception 1	19.09.16	27.09.16
Inception 2	28.09.16	04.10.16
MS: Techwahl	04.10.16	04.10.16
Elaboration 1	05.10.16	18.10.16
Elaboration 2	19.10.16	01.11.16
MS: End of Elab	01.11.16	01.11.16
Construction 1	02.11.16	15.11.16
Construction 2	16.11.16	29.11.16
MS: MSP_basic	22.11.16	22.11.16
Construction 3	30.11.16	13.12.16
MS: Feature Freeze	01.12.16	01.12.16
MS: Code Freeze	13.12.16	13.12.16
Transition	14.12.16	23.12.16
MS_Bericht	22.12.16	22.12.16

Abbildung 11.2: Inceptionphase



Vorgang	Anfang	Ende
Inception 1	19.09.16	27.09.16
Inception 2	28.09.16	04.10.16
MS: Techwahl	04.10.16	04.10.16
Elaboration 1	05.10.16	18.10.16
Elaboration 2	19.10.16	01.11.16
MS: End of Elab	01.11.16	01.11.16
Construction 1	02.11.16	15.11.16
Construction 2	16.11.16	29.11.16
MS: MSP_basic	22.11.16	22.11.16
Construction 3	30.11.16	13.12.16
MS: Feature Freeze	01.12.16	01.12.16
MS: Code Freeze	13.12.16	13.12.16
Transition	14.12.16	23.12.16
MS_Bericht	22.12.16	22.12.16

Abbildung 11.3: Elaborationsphase

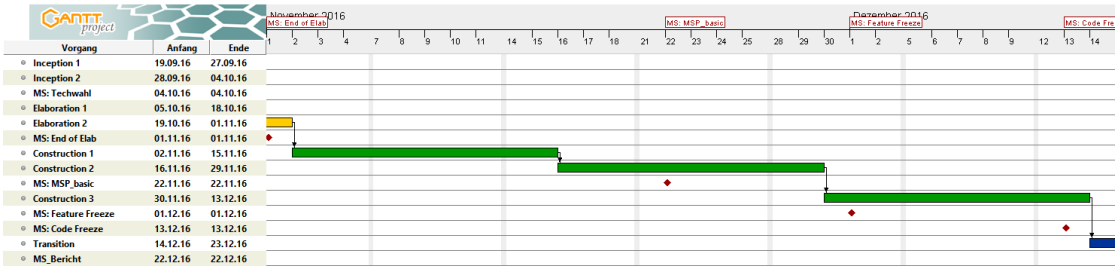


Abbildung 11.4: Constructionsphase

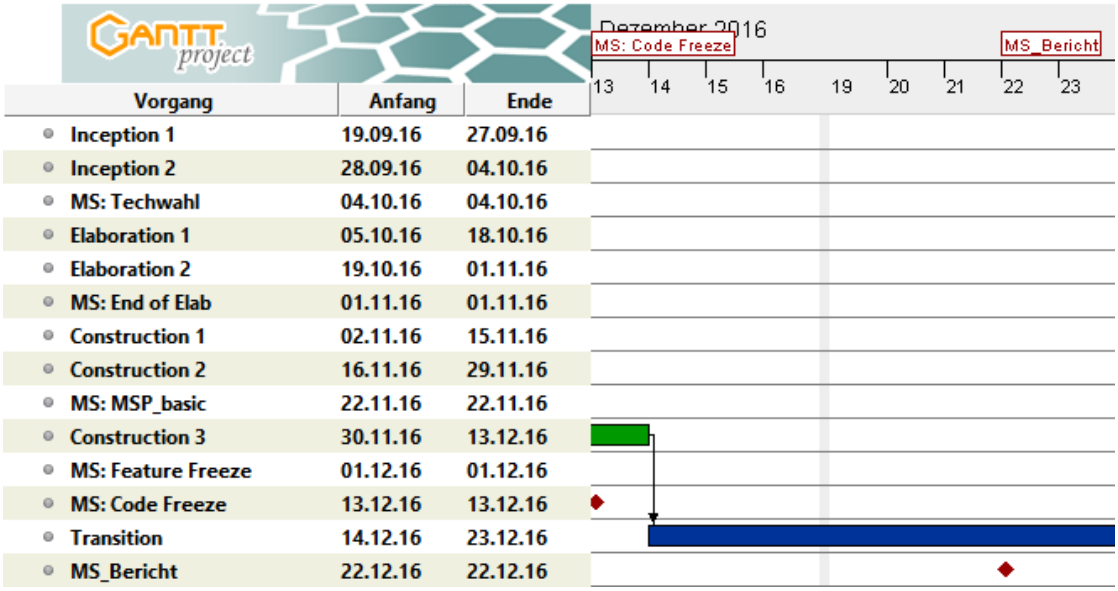


Abbildung 11.5: Transitionphase

11.5.2 Tickets

Die einzelnen Aufgaben haben wir im Redmine über das Ticketsystem erfasst und so unsere Stundenabrechnung realisiert. In den nachfolgenden Graphen sind die einzelnen Tickets je Phase ersichtlich.

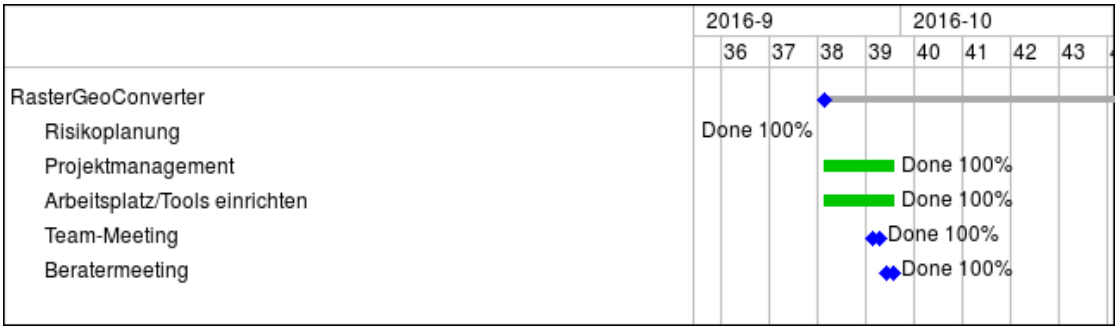


Abbildung 11.6: Tickets in der Inception 1

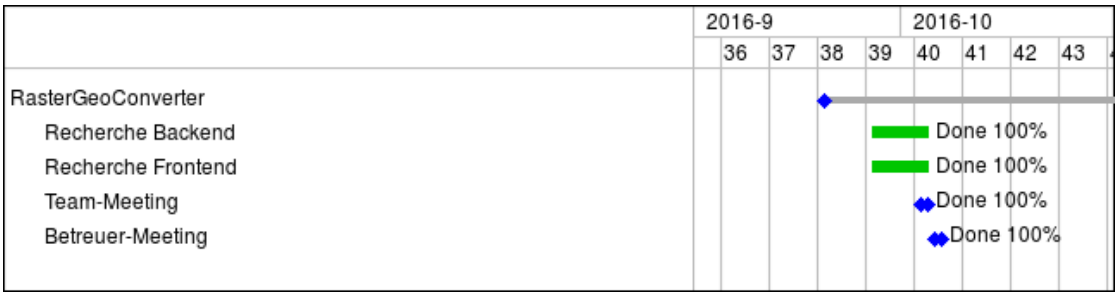


Abbildung 11.7: Tickets in der Inception 2

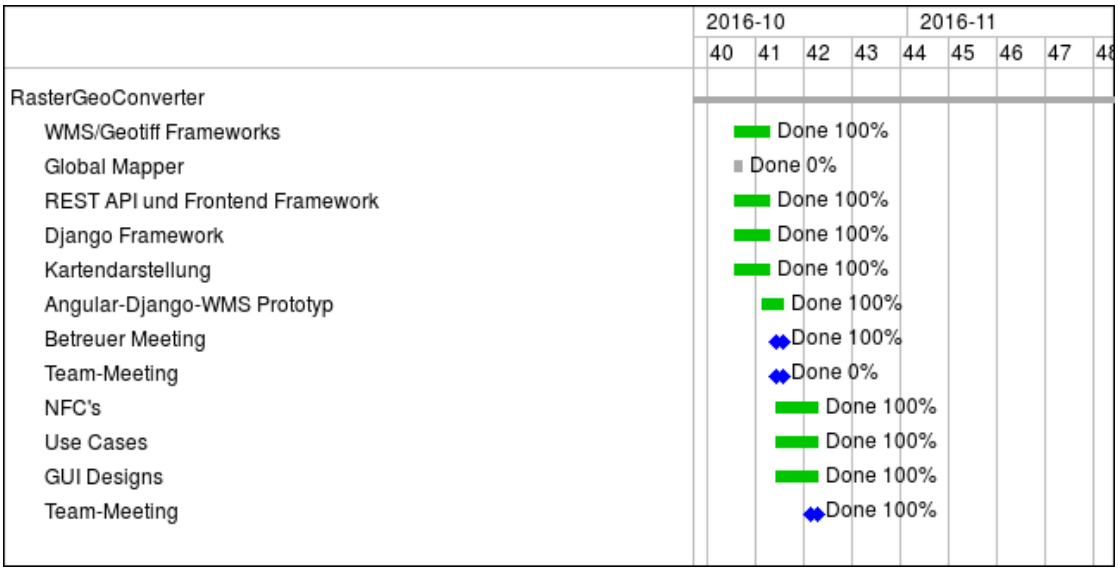


Abbildung 11.8: Tickets in der Elaboration 1

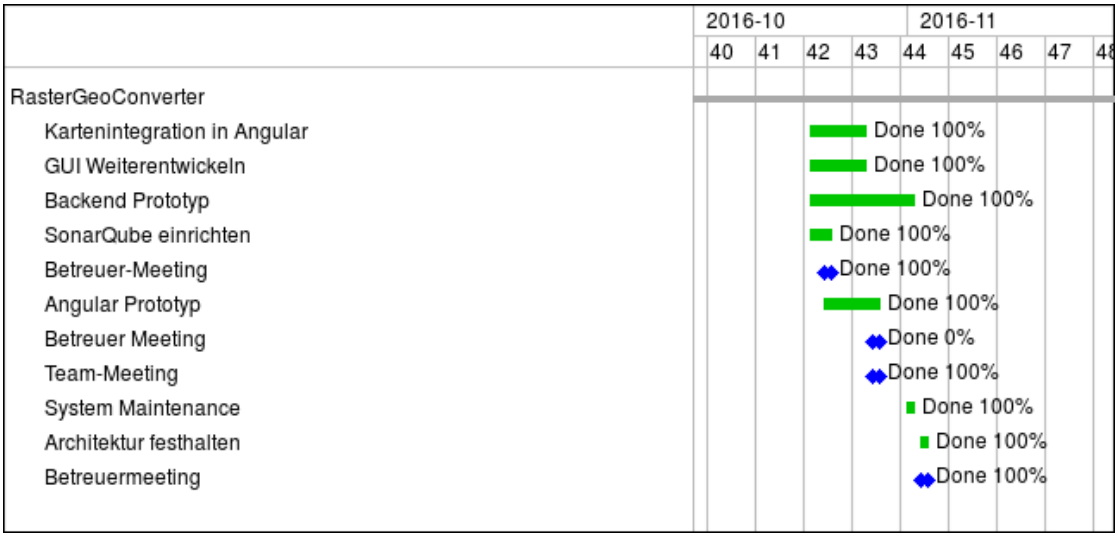


Abbildung 11.9: Tickets in der Elaboration 2

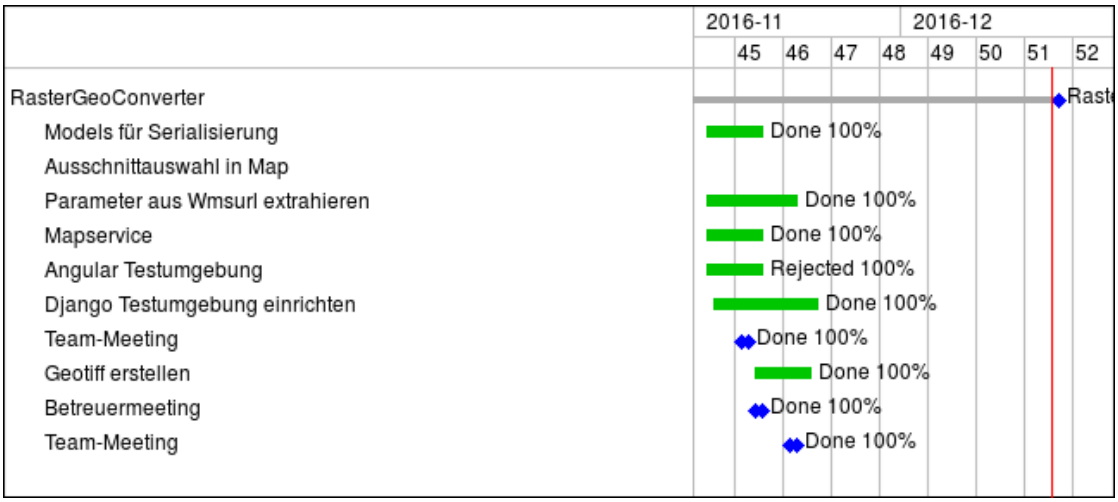


Abbildung 11.10: Tickets in der Construction 1

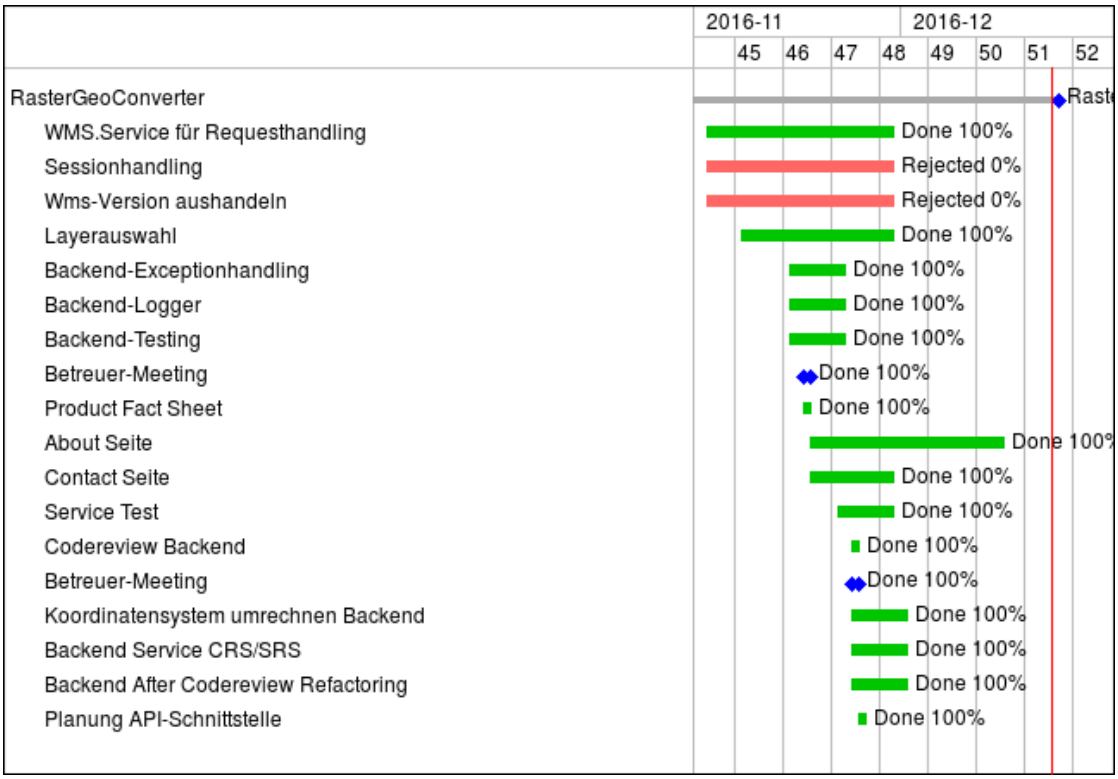


Abbildung 11.11: Tickets in der Construction 2

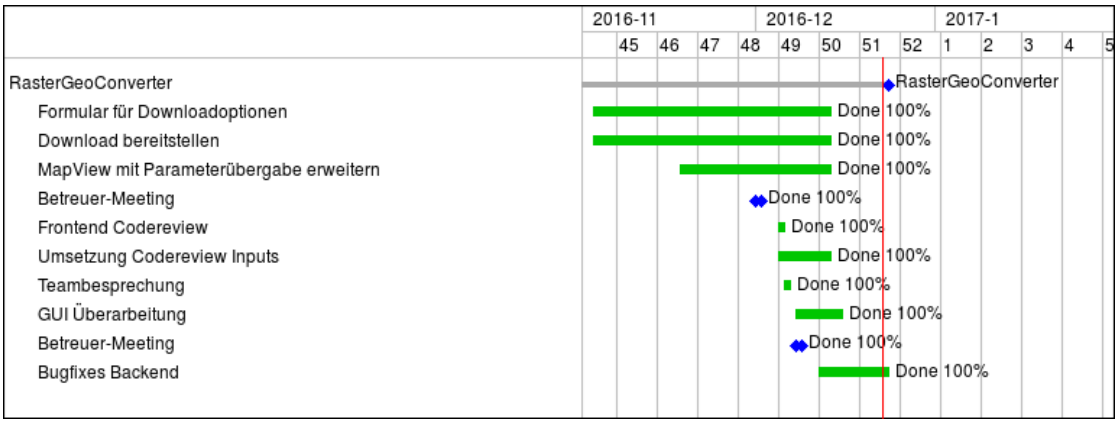


Abbildung 11.12: Tickets in der Construction 3

11.5.3 Nichterfüllte Tickets

Im folgenden Ganttdiagramm ist ersichtlich, dass wir nicht alle Aufgaben erfüllen konnten. Die optionalen Aufgaben konnten wir aus Zeitgründen nicht realisieren. Diese sind im Kapitel 5 unter *Weiterentwicklung* zu finden.

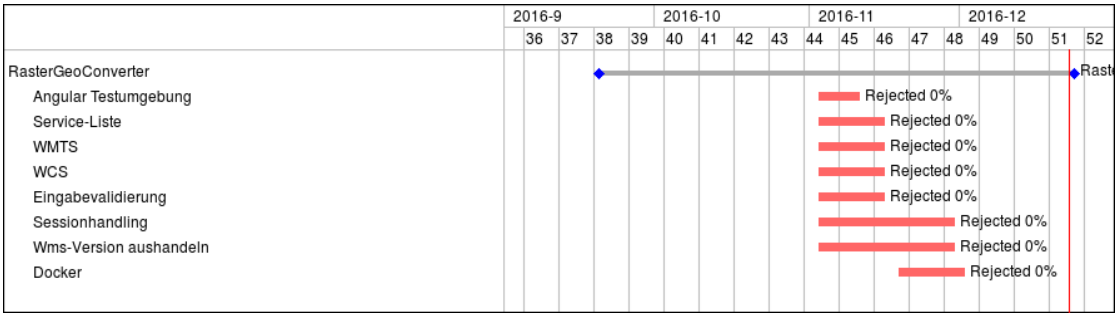


Abbildung 11.13: Nichterfüllte Tickets

11.5.4 Meilensteine

Die Meilensteine haben wir alle erreicht. Einzig der Meilenstein: Techwahl mussten wir um 2 Wochen verschieben, da die Evaluation länger gedauert hatte als wir zuerst angenommen haben. Die Evaluation wurde auch bei den Risiken aufgeführt.

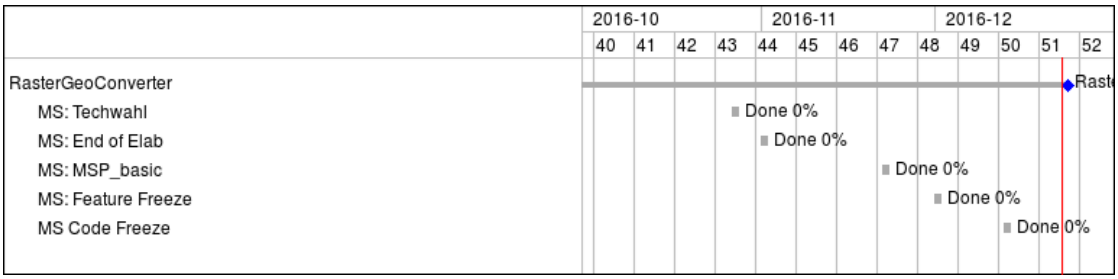


Abbildung 11.14: Meilensteine Ist

11.6 Risikoanalyse

In jedem grösseren Projekt sind Risiken und Chancen vorhanden. Risiken sind uneinschätzbare Problemstellungen oder Situationen. Verzögerungen, Umplanungen oder sogar Projektabbrüche können mögliche Szenarien sein, wenn Risiken eintreffen. Chancen sind Möglichkeiten einen Mehrgewinn in einem Projekt zu generieren. Als Beispiel könnten Erfahrungswerte, Softwarebibliotheken oder aber passende Technologien das Projekt positiv beeinflussen. Damit nun unser Projektaufwand geschätzt werden konnte, führten wir eine Risikoanalyse durch. So konnten wir die kritischen Bereiche erfassen und Massnahmen definieren, um diese zu umgehen oder beim Eintreffen des Risikos zu reagieren.

11.6.1 Erläuterungen

Bei unserer Risikoanalysen haben richten wir uns an zwei Parameter:

- Möglicher Schaden in Stunden
- Eintrittswahrscheinlichkeit in Prozent

Beide Werte zusammen ergeben dann einen gewichteten Schaden in Stunden. Dieser sollte in den Zeitplan hinzugerechnet werden.

11.6.2 Risikotabelle

Nr	Titel	Beschreibung	max. Schaden [h]	Eintritt %	Gewichteter Schaden [h]	Vorbeugung	Verhalten beim Eintreten
R1	Kartenmaterialbezug	Der Bezug des Kartenmaterials erweist sich als sehr umständlich. Service einbinden benötigt sehr viel Zeit.	24	10%	2.4	Informationen in Inceptionphase zusammensuchen um Komplikationen vorzubeugen.	WMS Priorisieren und mit Betreuer absprechen.
R2	GeoTIFF Download	Der GeoTIFF download benötigt sehr viele Ressourcen, was unseren Server überfordert.	16	20%	3.2	Mechanismus Überarbeiten, Alternativen definieren; z.B. über Links,Externe Tools ausmachen	Abschätzung der Alternativen und Entscheid
R3	Kompatibilitätsproblem e	Verknüpfung von Frontend und Backend funktioniert nicht/ ist erschwert.	32	5%	1.6	Schnittstellen definieren, Prototyping, Frameworks finden	Gemeinsame Fehlersuche, Krisenmanagement
R4	WMS-Standard nicht richtig implementiert	Der Lieferant konfigurierte den WMS nicht richtig. Die Daten können nur erschwert oder gar nicht bezogen werden	32	20%	6.4	Programmarchitektur auf validierung anpassen.	Benutzer informieren und Prozess abbrechen.
R5	Einarbeitung für Backend	Für das Backend wird eine neue Programmiersprache sowie Frameworks eingesetzt. Der Aufwand für das Erlernen ist gross	32	50%	16	Die Frameworks werden evaluiert.	Die Sprache und Frameworks müssen sauber erlernt werden. Der dazugehörende Aufwand ist Teil des Projektes.
R6	Einarbeitung für Frontend	Für das Frontend wird Javascript, sowie passende Frameworks eingesetzt. Der Aufwand für das Erlernen ist gross	32	50%	16	Die Scriptsprache und Frameworks werden evaluiert.	Die Sprache und Frameworks müssen sauber erlernt werden. Der dazugehörende Aufwand ist Teil des Projektes.
R7	Usability	Der Rahmen der Usability ist offen. Der Umfang wurde noch nicht klar abgesteckt, d.h. der Aufwand ist nicht absehbar.	32	25%	8	- Mit dem Betreuer den Umfang klar definieren - Feste Zeiten für die Mockups eintragen	Betreuer über den zusätzlichen Zeitaufwand informieren und hinweise, dass die Zielerreichung in Gefahr sein könnte
Summe			**200**		**53.6**		

Abbildung 11.15: Mögliche Risiken

11.6.3 Risikomatrix

Die Risikomatrix ermöglicht eine gute Einschätzung der Risiken. Es ist ersichtlich welche Risiken genauer betrachtet werden müssen. Um die Risikotabelle zu erstellen mussten zwei Werte errechnet werden.

1. Eintrittwahrscheinlichkeit 1 - 5
2. Schadensmass 1 - 5

Die Eintrittswahrscheinlichkeit ist aus der Risikotabelle direkt ersichtlich. Für das Schadensmass mussten wir einen maximalen Stundenwert definieren. Dieser haben wir aus der Summe des gewichteten Schadens definiert, da der gewichtete Schaden die gesamte Reserve für das Risikomanagement ist.

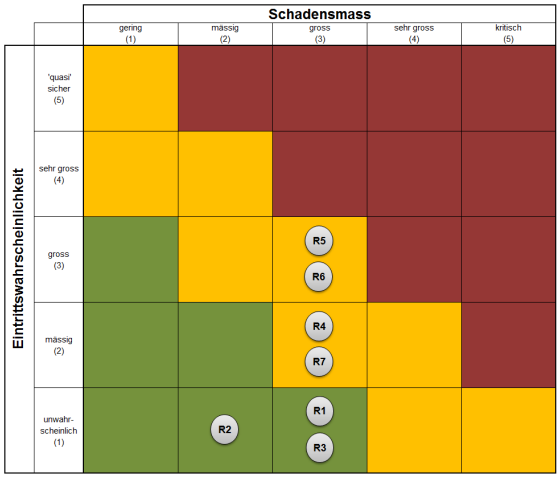


Abbildung 11.16: Risikomatrix

Fazit

Aus der Risikomatrix ist ersichtlich, dass vor allem die Einarbeitung die verschiedenen Technologien ein grosses Risiko waren. Es war nicht einfach abzuschätzen, wie viel Zeit wir tatsächlich für die Einarbeitung benötigen. Zudem könnten die gewählten Technologien für die Anforderungen des Projektes falsch gewählt worden sein. Tatsächlich zeigte sich während des Projekts, dass die Einarbeitungszeit sehr gross war. Das heisst, das Risiko war eingetroffen und wir mussten die mehr Zeit investieren, als wir überhaupt eingeplant hatten. Daraus erfolgte, dass der RasterGeoConverter nicht den Stand erreicht hatte, den wir erhofft hatten. Nur die Basisfunktionen konnten umgesetzt werden. Die optionalen Ziele mussten wir auf einen späteren Zeitpunkt verlegen. Diese Ziele sind im Kapitel 5 unter *Weiterentwicklung* zu finden.

Teil III

Appendix

Literatur

[1] Angular2. *Angular2 Style Guide*. URL: https://angular.io/docs/ts/latest/guide/style-guide.html#!#file-tree (besucht am 15. 12. 2016).

[2] Django. *Django Documentation*. URL: https://docs.djangoproject.com/en/1.10/ (besucht am 22. 12. 2016).

[3] Flask. *Flask Documentation*. URL: http://flask.pocoo.org/docs/0.12/ (besucht am 22. 12. 2016).

[4] Stefan Krause. *JS web frameworks benchmark – Round 3*. URL: http://www.stefankrause.net/wp/?p=301 (besucht am 06. 12. 2016).

[5] LeafletJS. *Logo Leaflet*. URL: http://leafletjs.com/docs/images/logo.png (besucht am 23. 10. 2016).

[6] Mapbox. *Logo Mapbox*. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/b/b4/Mapbox_Logo.svg/1160px-Mapbox_Logo.svg.png (besucht am 23. 10. 2016).

[7] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (besucht am 06. 12. 2016).

[8] OpenLayers. *Logo OpenLayers*. URL: http://geospatial.blogs.com/.a/6a00d83476%20d35153ef017d3dc4b31c970c-600wi (besucht am 23. 10. 2016).

[9] Pyramid. *Pyramid Overview*. URL: https://trypyramid.com/ (besucht am 22. 12. 2016).

[10] Python. *Python Documentation*. URL: https://docs.python.org/3/ (besucht am 22. 12. 2016).

Glossar

Angular 2 Angular2 ist ein Open Source JavaScript Framework von Google. Angular2 liefert fast alles, was man braucht um ein kompliziertes Webfrontend oder eine mobile Anwendung zu bauen, von schnellem Rendering über leistungsstarke Templates bis zum Datenmanagement bietet Angular2 fast alles.. 5, 7–9, 18, 19, 23, 26

Django Python Framework für die Entwicklung von Frontend/Backend-Lösungen. All-in-one Werkzeugkiste mit integrierten Funktionalitäten wie Datenbank, Admintools und Websicherheit. 5, 11–13, 15, 16, 19, 23

Geometa Lab Das Geometa Lab gehört zum Institut für Software (IFS) der HSR. Im Geometa Lab werden unter anderem geowissenschaftliche Anwendungen entwickelt.. 5

GUI Graphical User Interface. 3

NPM Node Package Manager. 48

OSMaxx Geodaten des einzigartigen und weltweiten OpenStreetMap-Projekts aufbereitet für Kartenvisualisierung und Analyse in GIS. Schneidet OpenStreetMap-Daten aus (exract, excerpt), bereitet sie zu Geodaten auf und gibt sie als GIS-Fileformate ab.. 35

Python Python ist eine interpretierte, objektorientierte, high-level Programmiersprache. Es besteht eine weite Verbreitung in den akademischen Anwendungen.. 3, 5, 10, 11, 19, 25, 26

Rasterkarten Rasterkarten sind vergleichbar mit einer eingescannten klassischen Papierkarte. Sie bestehen aus einer einzigen zweidimensionalen Ebene und enthalten nur die dort aufgedruckten Informationen, egal wie stark man hinein- oder herauszoomt. Sie werden denn auch beim Hineinzoomen etwas verpixelt, beim Herauszoomen „laufen sie zu“, werden also wegen der gleich bleibenden Informationsdichte etwas unübersichtlich.. 3, 26

UI User Interface. 38, 40

URL Uniform Resource Locator. 13, 22

WFS Web Feature Service. 5

WMS Web Map Service. 2, 4, 5

WMTS Web Map Tile Service. 4, 5

WxS Web Map Service (Allgemein). 15, 45

Anhang A

Abgabestruktur