HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

BACHELORARBEIT

GeoConverter

Abteilung Informatik
Hochschule für Technik Rapperswil
Diego Etter, Céline Ott

Betreut von
Prof. Stefan F. Keller

Experte
Claude Eisenhut

Gegenleser
Peter Heinzmann

Frühjahrssemester 2017

Abstract

Immer mehr Geoinformationen - vor allem amtliche - werden digital und frei als Open Government Data angeboten. Technisch geschieht dies in Form eines Webdienstes, wodurch sich die Geodaten zentral verwalten lassen.
Leider unterstützen viele Planungsprogramme die Einbindung von diesen Webdiensten nicht. Was ein grosses Hindernis für Architekten, Bauingenieure und Planer darstellt, welche das Kartenmaterial für ihre Projekte und Planungen gerne beziehen würden. Dazu müssten die Geodaten in einem nutzbaren Format von den Webdiensten geladen und allenfalls konvertiert werden.

Aus diesem Grund wurde vor Jahren am Geometa Lab der HSR der GeoConverter realisiert, mit dem Vektordaten konvertiert werden können. Diese Webapplikation wird rege benutzt, insbesondere auch von Schweizer Kantonen und im HSR-Unterricht.
Die neu konzeptionierte GeoConverter Applikation bietet nun einen grösseren Funktionsumfang zur Verarbeitung der Geodaten an. Der bisherige Umfang wurde überarbeitet und mit Funktionen für Rasterdaten erweitert.
Im Rumpf der Applikation ist eine Schnittstelle für die direkte Kommunikation zu den Geodatendiensten implementiert. Zudem wurde eine Bildverarbeitungsbibliothek für die Erzeugung der verschiedenen Formate hinzugefügt.
Als Rasterdatenformat wird das Bildformat GeoTIFF angeboten, welches die Georeferenz beinhaltet. Die Vektordaten können direkt vom Webdienst bezogen oder per Datenupload in die gängigen Formate konvertiert werden.

Mit modernen Technologien wie Vue.js, Leaflet und Django umgesetzt, ist der GeoConverter gut wartbar aber auch erweiterbar, und somit für die Zukunft gerüstet.
Als Docker-Container ausgeliefert, lässt sich die GeoConverter Applikation problemlos auf einem Linux-Server installieren und betreiben.

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 geospatial data services, short WxS. The argument to use geodata services is the simplicity of this services. WxS allows easy and central management of government geodata.
A disadvantage of using WxS is that a lot of planning software does not support the integration of this sort of services. For architects, civil engineers and other clients, who would utilise the geo data, this results in a considerable obstacle. To overcome this hindrance, the geodata have to be made available as a fitting data format, loaded from the WxS.

Years ago, the Geometa Lab at HSR solved this issue by implementing the GeoConverter. This web based application is used extensively, especially by Swiss cantons and in HSR classes.
The newly designed GeoConverter application now offers a larger range of functions for processing geodata. The existing scope has been revised and extended with functions for raster data. The application itself provides an interface for the direct communication with WxS to order the geo data as well as a library to transform the data to various file formats.
For raster maps the resulting format is 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.
For vector data the application serves multiple, well known formats. The geo data can be loaded directly from a web service or transformed from an uploaded file.
The GeoConverter includes modern technologies like Vue.js, Leaflet und Django, 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 Geoinformationen - vor allem amtliche - werden digital und frei als Open Government Data angeboten. Technisch geschieht dies in Form eines Webdienstes. Solche online bereitgestellten Karten und Geodaten werden direkt ins eigene System eingebunden. Sie sind damit immer aktuell.

Leider unterstützen viele Planungsprogramme keine solchen Geodatendienste. Dies ist ein grosser Nachteil für Architekten, Bauingenieure und Planer, welche die (Raster-)Karten und Geodaten als Arbeitsmaterial gerne beziehen würden. Zudem werden die Geoinformationen nicht immer im gewünschten Format geliefert. Der Bezug der gewünschten Geoinformationen und das Konvertieren verlangen Spezialkenntnisse, die bei den Benutzern oft fehlen. Darum wurde vor Jahren am Geometa Lab der HSR der GeoConverter realisiert, mit dem Geodaten in verschiedenen Vektorformaten konvertiert werden können. Diese frei zugängliche Webapplikation wird rege benutzt, insbesondere auch von Schweizer Kantonen und im HSR-Unterricht.

Der erweiterte und vollständig überarbeitete GeoConverter soll eine nochmalige Verbesserung dieser Situation leisten.

Ziel

Mit dem Projekt GeoConverter soll eine webbasierte Plattform für den einfachen Bezug und Konvertierung von Geoinformationen geschaffen werden. Über ein übersichtliches Interface soll der Benutzer die Möglichkeit erhalten, direkt über den Geodatendienst die benötigte Karteninformation im gewünschten Format zu beziehen. Ein weiteres Feature ist die Möglichkeit, das Format von Geoinformationen zu konvertieren.

Die Applikation soll die bisherigen Basistechnologien wie die Python-Programmiersprache, das Django-Framework und die PostgreSQL-Datenbank übernehmen. Sie soll einfach zu installieren und zu warten sein. Um dies zu gewährleisten, wird der GeoConverter als Docker-Container geliefert.

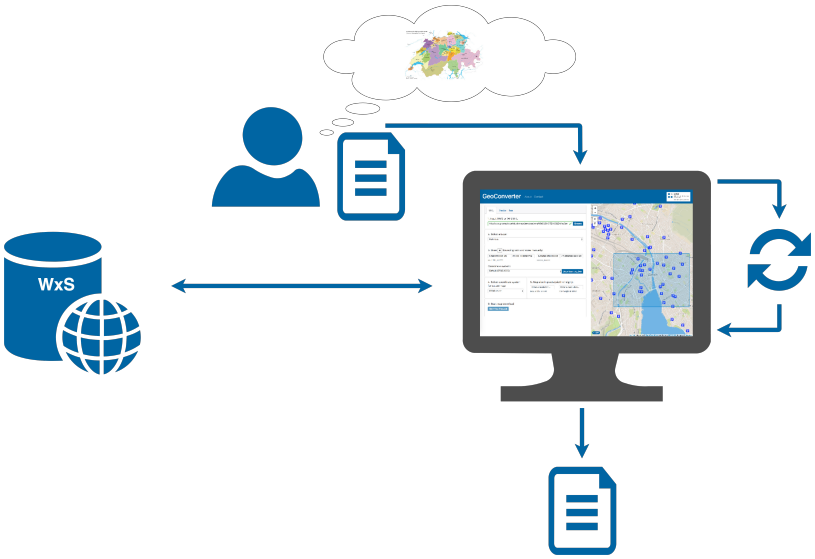


Abbildung 1: Funktionsübersicht der GeoConverter-Webapplikation (Pfeile stellen Funktionsaufrufe dar).

Ergebnisse

Das Resultat dieser Arbeit ist eine lauffähige, moderne Webapplikation mit einer übersichtlichen grafischen Benutzerschnittstelle. Der Benutzer wird schrittweise durch die Anwendung geführt. Die Geoinformationen können von «Web Map Service» (WMS) als Rasterdaten, oder von «Web Feature Service» (WFS) als Vektordaten bezogen werden. Als gängiges Rasterformat wurde «GeoTIFF» gewählt. Zudem können weiterhin Vektordateien hochgeladen werden. Diese werden über das Inputformat bestimmt und können über eine Auswahl als Zielformat frei gewählt werden.

Der GeoConverter wurde durch eine REST-Schnittstelle in zwei Teilbereiche aufgeteilt, um die Vorzüge aus verschiedenen Technologien auszuschöpfen.
Das Frontend wurde mit dem JavaScript-Framework Vue.js (Programmlogik) und dem CSS-Framework Bootstrap (Layout) umgesetzt und bietet durch das Leaflet-Plugin (Kartenansicht) eine umfassende Benutzeroberfläche an.
Für die Kommunikation mit den Geodatendiensten, die Konvertierung 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 vor allem im Geoinformationsbereich. Das Django-Framework bietet mit seinen Funktionalitäten eine gute Basis für das Grundgerüst des GeoConverters.
Die gesamte Applikation wird als Docker-Container ausgeliefert. Das vereinfacht die Installation und Wartbarkeit erheblich.

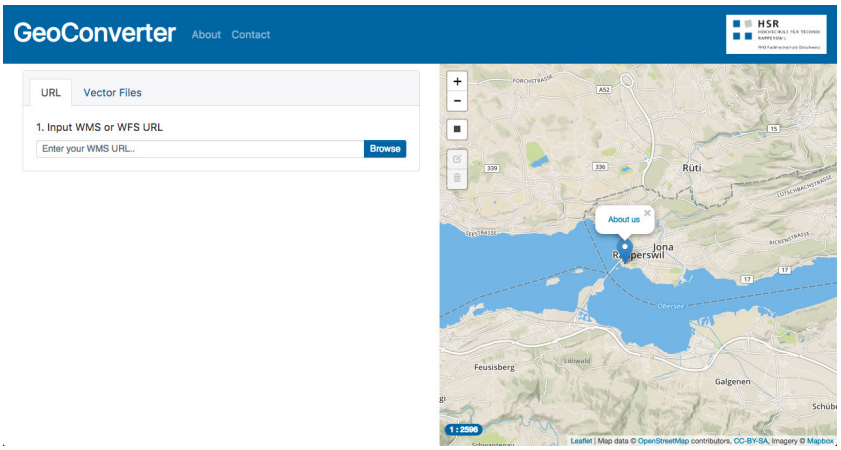


Abbildung 2: Benutzeroberfläche der GeoConverter-Webapplikation.

Ausblick

Die Funktionen innerhalb des GeoConverters bieten einige Erweiterungsmöglichkeiten an.

Webkartendienste
: Momentan wurde ausschliesslich der Kartenbezug von «Web Map Service» und «Web Feature Service» umgesetzt. Es sind aber auch andere Webkartendienste verbreitet. Darunter fallen der «Web Map Tiles Service» und/oder der «Web Coverage Service». In einem weiteren Schritt können auch diese Services eingebunden werden.

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

Logging
: Für den verbesserten Betrieb des GeoConverters könnte ein Logging umgesetzt werden. Damit könnte der Administrator den Dienst optimal warten und die Verfügbarkeit besser garantieren.

Verfügbare Geodatendienste
: Dem Benutzer könnten gängige Geodatendienste vorgeschlagen und deren Verfügbarkeit angezeigt werden.

Tutorial
: Für eine einfache Einführung könnte ein (Online-)Tutorial realisiert werden. So soll der Benutzer schrittweise in die Applikation und Funktionen eingeführt werden.

Aufgabenstellung

GeoConverter - Online-Konversion von Web Map Services ins GeoTIFF-Format (Fortsetzungsarbeit)

Bachelorarbeit Frühjahrssemester 2017, Studiengang Informatik

Hintergrund

Bauingenieure, Planungsbüros und Raumplaner benötigen für viele Projekte Planungsmaterial. Darunter fallen auch Karten welche digital und online über einen Webdienst bezogen werden können. Nicht alle Planer verfügen jedoch über das nötige Wissen um solches Kartenmaterial zu beziehen. Zudem sind viele Planungsapplikationen nicht im Stande, diese Kartendaten als Webdienste einzubinden. Es werden spezifische Datei-Formate, wie das GeoTIFF benötigt.

Ziele

Ziel ist eine Webapplikation, genannt RasterGeoConverter…

1. zum Download von Rasterkarten im GeoTIFF-Format ausgehend von einem Webservice
2. kombiniert mit dem bestehenden GeoConverter für Vektordaten
3. angeboten als Docker-Container

Hier ein kleiner Use Case:

- Benutzer wählt Datenquelle (WMS, optional WMTS)
- Benutzer wählt Ausschnitt (oder gibt Geonamen oder WGS84-Koordinate an)
- Benutzer gibt ggf. Parameter an: Format (GeoTIFF, GRID, etc.), Koordinatensystem (WGS84, UTM, LV95), Auflösung (dpi), ...
- Benutzer klickt auf „Download“ und wartet auf Downloadlink.

Aufgabenstellung

Mit dem RasterGeoConverter sollen die Benutzer GeoTiffs über einen WMS möglichst simpel beziehen können. Dies soll ohne grosse Vorkenntnisse des Webservices geschehen können:

Der RasterGeoConverter soll in einer weiteren Phase im Projekt in die bestehende Webapplikation "GeoConverter" integriert werden.

Das gesamte Produkt soll als Docker-Container angeboten werden. Zudem wird die Installation und Benutzung dokumentiert.

Deliverables:

- RasterGeoConverter bzw. GeoConverter mit RasterGeoConverter kombiniert
- Funktionierende Webapplikation (Frontend & Backend)
  - WMS unterstützt, optional: WMTS
  - als Docker-Container ausgeliefert
  - Dokumentation der Software
  - Anleitungen (wo nötig)
- Dokumentation: Gedruckter Bericht (gebunden) mit USB-Stick für Betreuer (mit sämtlichen Dokumenten, Quellen und Daten von Software und Bericht). Dazu kommt die Abgabe für den Studiengang gemäss sep. Instruktionen.

Vorgaben/Rahmenbedingungen

- Technische Anforderungen: Vue.js, Python, Django, Docker, OGC Specifications WMS/WMTS/WFS, GeoConverter (bestehendes Projekt/Repository).
- Nichtfunktionale Anforderungen: Responsive GUI, Docker-Container erstellen.

- Die Software soll Open Source sein, die Softwarelizenz ist „MIT“.
- Die SW-Engineering-Methode und Meilensteine werden mit dem Betreuer vereinbart.
- Sourcecode und Software-Dokumentation sind Englisch (inkl. Installation, keine Benutzerdokumentation, höchstens eine Online-Kurzhilfe).
- Die Software-Benutzerschnittstelle ist Englisch.
- Die Projekt-Dokumentation und -Präsentation sind auf Deutsch.
- Der Source Code, die Code-Kommentare und die Versionsverwaltung sind in Englisch.
- Die Nutzungsrechte an der Arbeit bleiben bei den Autoren, gehen aber auch an die HSR und den Betreuer über.
- Kein Video da freie Webapplikation.
- Ansonsten gelten die Rahmenbedingungen, Vorgaben und Termine des Studiengangs Informatik bzw. der HSR.

Inhalt der Dokumentation

- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Projektdokumentation
  4. Anhänge (Literaturverzeichnis, Glossar, Dateispeicher-Inhalt)
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.
- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung, Poster) gemäss www.hsr.ch und Absprache mit dem Betreuer.

Bewertungsschema

Es gelten die üblichen Regelungen zum Ablauf und zur Bewertung der Arbeit (6 Aspekte) des Studiengangs Informatik der HSR jedoch mit besonderem Gewicht auf moderne Softwareentwicklung (Tests, Continuous Integration, einfach installierbar, funktionsfähig).

Beteiligte

Diplomanden

Diego Etter und Céline Ott

Industriepartner

-

Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller (sfkeller@hsr.ch), Geometa Lab am IFS der HSR, Nicola Jordan (Mitarbeiter am IFS der HSR) sowie Vertreter von Stadt und Kanton Zürich.

Danksagungen

An dieser Stelle möchten wir uns bei all denjenigen bedanken, die uns während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Ganz besonders möchten wir Prof. Stefan F. Keller und Nicola Jordan danken, die unsere Arbeit durch ihre fachliche und persönliche Unterstützung begleitet haben.

Ein besonderer Dank gilt Stadt und Kanton Zürich für die Unterstützung dieses Projekts.

Inhaltsverzeichnis

Teil I

Technischer Report

Kapitel 1

Einführung

Immer mehr Geoinformationen - vor allem amtliche - werden digital und frei als Open Government Data angeboten. Technisch geschieht das über Geoinformationsplattformen, welche auf öffentlichen Servern betrieben werden. So können die Geoinformationen zentral verwaltet werden und allen Interessenten Zugriff gewähren. Raumplaner, Bauingenieure und andere Projektplaner sind natürlich an diesen Geoinformationen in Form von Karten, Geländeeigenschaften und Standortinformationen interessiert. Ein grosser Nachteil ist jedoch, dass die Planungsprogramme, zum Beispiel ein CAD, die Einbindung dieser Geoinformationsplattformen nicht unterstützen. Auch fehlt das nötige Wissen die Karten in das passende Format zu generieren oder konvertieren.

Für Vektordaten wurde daher vor Jahren am Geometa Lab der HSR der GeoConverter realisiert, mit dem Geodaten in verschiedenen Vektorformaten konvertiert werden können.

1.1 Vision

Mit dem Projekt GeoConverter soll eine webbasierte Plattform für den einfachen Bezug und Konvertierung von Geoinformationen geschaffen werden. Dazu gehören Raster- wie aber auch Vektordaten. Über ein übersichtliches Interface soll der Benutzer die Möglichkeit erhalten, direkt die benötigten Geoinformationen im gewünschten Format zu beziehen, aber auch bestehende Daten zu konvertieren. Das heisst, dass Rasterkartenausschnitte, über eine Kartenansicht dargestellt, ausgewählt und generiert werden können. Der bestehende GeoConverter für Vektordaten soll durch diese Webapplikation ersetzt werden. Somit sollen die bewährten Konvertierungsfunktionen in die neue Applikation integriert werden. Die Applikation soll die bisherigen Basistechnologien wie die Pythonprogrammiersprache, das Django Framework und die PostgreSQL-Datenbank übernehmen. Sie soll einfach zu installieren und warten sein. Um dies zu gewährleisten, soll das Resultat als Docker-Container geliefert und betrieben werden können.

1.2 Ziele

Die Hauptziele des Projektes sind:

- Eine funktionierende Webapplikation (Frontend und Backend)
- Rasterkarten von Web Map Service (WMS) laden
- Die Funktionen des bestehenden GeoConverters sollen integriert werden
- Der GeoConverter soll als Docker-Container angeboten werden.

Es wurden folgende optionale Ziele definiert:

- WMTS unterstützen

- Verfügbare Services vorschlagen
- Karten teilen

1.3 Rahmenbedingungen

Folgende Rahmenbedingungen wurden durch den Betreuer vorgegeben:

- Programmiersprache Python zusammen mit Django Framework
- Datenbank bevorzugt PostgreSQL
- Frontend wird mit Vue.js umgesetzt
- Wichtige nicht funktionale Anforderungen sind die HSR Corporate Identity
- responsive Graphical User Interface (GUI)
- lauffähiger Docker-Container

1.4 Vorgehen

Das Projekt wird in zwei Phasen unterteilt. In der ersten Phase wird der GeoConverter umgesetzt. Die zweite Phase wird der Integration des bestehenden GeoConverters gewidmet.
Während der Entwicklung wird der GeoConverter direkt in eine Dockerumgebung umgesetzt. So können wir garantieren, dass der veröffentlichte Stand immer problemlos in Betrieb genommen werden kann.
Wir wählen für das Projektvorgehen Rational Unified Process (RUP), kombiniert mit agilen Vorgehensweisen. Das heisst wir Implementieren die einzelnen Features in Sprintphasen und führen tägliche Standup-Meetings durch. Das Betreuerteam wird mit wöchentlichen Meetings informiert. Zudem bieten wir ihnen eine Informationsplattform, um den Projektfortschritt zu vermitteln.
Während des Projektes durchlaufen wir pro Teilbereich die RUP-Phasen. Dazu erstellen wir zuerst kleine Prototypen und Tests. Danach wird über die Constructionphase das Endprodukt entwickelt.

1.5 Stand der Technik

Der bestehende GeoConverter wurde in Python geschrieben und verwendet auch das Django-Framework. Die Benutzerschnittstelle wurde ebenfalls in Django umgesetzt.
Der RasterGeoConverter wurde während der vorhergehenden Studienarbeit erstellt und wurde im Frontend mit Angular2 und im Backend mit Python und Django entwickelt.
Obwohl die Backends beider Applikationen in Python und Django geschrieben wurden, wird nicht einfach alles übernommen, sondern neu implementiert. Somit sollen die beiden Applikationen besser in eine Applikation kombiniert werden können.
Das neue Frontend für den GeoConverter wird komplett neu implementiert, da weder vom RasterGeoConverter noch vom bestehenden GeoConverter Funktionen übernommen werden können.

Kapitel 2

Analysen

Durch die vorhergehende Studienarbeit welche sich um den RasterGeoConverter drehte, konnten einige Erfahrungen und Erkenntnisse gesammelt werden. Ziel ist es nun, diese in die neue Arbeit einzubringen und aus den gemachten Fehlern zu lernen.

2.1 Frontend

In einem ersten Schritt sollen nur die RasterGeoConverter Frontend Funktionalitäten in Vue.js umgesetzt werden. Danach sollen die Funktionalitäten des GeoConverters hinzukommen. So soll in einem iterativen Prozess der GeoConverter entstehen.

2.1.1 RasterGeoConverter

Das Frontend des RasterGeoConverters wurde während der Studienarbeit in Angular2 implementiert. Diese Entscheidung wurde durch eine Evaluation in der Studienarbeit getroffen.
Angular2 ist ein TypeScript-basiertes Webframework. Obschon Angular2 von Google mitentwickelt wurde, ist der Code Open Source. Die vielen Vorteile von TypeScript, wie die statische Typisierung, sollten der Entwicklung des RasterGeoConverters helfen. Doch leider brachte es nicht nur Vorteile, sondern auch Nachteile mit sich.

2.1.2 Leaflet

Für die Karteneinbindung im GUI wird eine JavaScript Library benötigt. Bei der Evaluation der Studienarbeit entschied man sich dabei für Leaflet. Leaflet ist eine Open Source JavaScript Bibliothek für die Darstellung von Geodaten. Die Bibliothek unterstützt WMS, Web Map Tile Service (WMTS) und weitere Formate.

Für die Verwendung von JavaScript Bibliotheken in TypeScript werden Typings benötigt. Typings ermöglichen es JavaScript-Bibliotheken mit Typannotationen zu versehen, als wären sie in typisiertem TypeScript geschrieben. (Mehr dazu im Artikel von Schulz, 2014)
Da Leaflet nur in JavaScript geschrieben wurde und die Typings dafür nur teilweise vorhanden sind, konnte nicht die komplette Funktionalität der Bibliothek verwendet werden. Dies führte zu einigen Problemen während der Studienarbeit, wodurch der Funktionsumfang des RasterGeoConverters verkleinert werden musste.

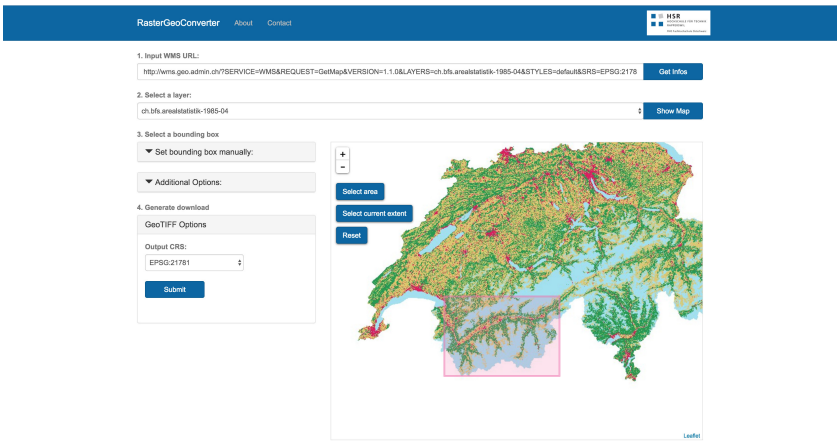


Abbildung 2.1: Bedienoberfläche RasterGeoConverter, Stand Studienarbeit

2.1.3 Schlussfolgerung

Deshalb soll in dieser Bachelorarbeit nun die komplette Funktionalität des RasterGeoConverters umgesetzt und mit den Funktionalitäten des GeoConverters erweitert werden.
Für den Erfolg des RasterGeoConverters muss die vollumfängliche Funktionalität der Leaflet Library genutzt werden können. Damit alle Funktionen der Library ohne Hindernisse verwendet werden können, soll das Frontend nun in einem JavaScript basierten Framework umgesetzt werden. Für diese Bachelorarbeit wurde deshalb das Vue.js JavaScript Framework vorgegeben.

2.1.4 Vue.js

Vue.js ist ein progressives JavaScript Framework, um User Interfaces (UIs) aufzubauen. Mit Vue.js ist es sehr einfach Projekte zu erstellen, welche auch andere JavaScript-Bibliotheken benötigen, da es entworfen wurde, um inkrementell erweitert zu werden. So kann es auch bestehenden Projekten hinzugefügt werden, um einfache Interaktivität hinzuzufügen. Die Core Library fokussiert sich hauptsächlich auf den View Layer der Applikation und ist deshalb sehr einfach mit anderen Bibliotheken zu kombinieren.

2.2 WMS

Der Webmapservice, kurz WMS, erzeugt räumlich referenzierte Karten aus geografischen Informationen. Innerhalb eines solchen Services können unterschiedliche geografische Informationen räumlich gespeichert und angeboten werden. Diese Daten können per Anfrage über das Web in unterschiedlichen Formen bezogen werden.

Damit diese Daten bezogen werden können, bietet der Webmapservice unterschiedliche Abfrageparameter an.

Wir richten uns in dieser Bachelorarbeit nach dem Standard des Open Geospatial Consortium (OGC) Consortium et al., 2006.

2.2.1 Anfrage-Parameter

Der WMS kann über eine HTTP-Abfrage angesteuert werden. Dazu wurden folgende Symbole reserveriert/definiert.

? : Start des Anfragestrings

& : Separator zwischen Parameter in Anfragestrings

= : Separator zwischen Name und Wert eines Parameters

, : Separator zwischen einzelen Werten in Listen-orientierten Parameter

+ : Kurze Representation für einen space-Charakter

2.2.2 Koordinatensystem

Im internationalen Standard werden zwei Klassen von Koordinatensystemen unterschieden:

1. Das Karten Koordinatensystem
2. Das Koordinatensystem des Layers für eine Bounding Box

Während einer Porträtierungsoperation konvertiert oder transformiert der WMS die Layerinformationen in das Kartenkoordinatensystem.

2.2.3 Output Formate

Der Antwort einer Webmapservice-Anfrage liegt immer eine Computerdatei bei. Diese Datei beinhaltet entweder Textdaten oder Bilddaten. Textausgaben werden in der Regel als XML geliefert. Die Bildausgabe wird als Pixelfeld mit einer fixen Grösse ausgeliefert.
Eine Übersicht über die Rasterformate findet sich auf der Seite «GDAL Raster Formats» von rouault, 2016

Jeder Server bietet mindestens eines dieser Formate an.

2.2.4 Auflösung

Für die Kartenauflösung werden die «WIDTH» und «HEIGHT» Parameter mitgegeben. Diese Parameter geben die Grösse der Karte in Pixel an.

Die Auflösung in DPI kann optional angeboten werden. Standardmässig beträgt der Wert 90 DPI («WMS vendor parameters — GeoServer 2.11.x User Manual», 2017).

2.2.5 Vendorspezifische Parameter

Je nach Anbieter werden zu den Standardparameter noch optionale Parameter angeboten. Der RasterGeoConverterteil des GeoConverters verfügt über die Möglichkeit, diese Parameter zusätzlich zu übermitteln. Jedoch liegt die Verantwortung dieser Anfrage beim Benutzer.

Wir verweisen auf den Artikel der Seite GeoServer («WMS vendor parameters — GeoServer 2.11.x User Manual», 2017).

2.2.6 Layer CRS

Ein Layerkoordinatensystem ist ein horizontales Koordinatenreferenzsystem für die geografische Information der Kartenquelle. Es ist in folgenden Entitäten ersichtlich:

- Boundingbox-Element in den Service Metadaten
- In der GetMap Anfrage
- In der GetFeatureInfo Anfrage

Jeder WMS muss mindestens ein Koordinatensystem unterstützen. Am besten sollten die gängigsten Koordinatensysteme wie CRS:84 oder EPSG:4326 unterstützt werden. Dieses Koordinatesystem ist weltweit sehr exakt und kann in die lokalen Koordinatensysteme konvertiert werden.

2.2.7 Parameter Regeln

Parameternamen sind nicht von der Gross/Kleinschreibweise abhängig im Gegensatz zu den Parameterwerten. Normalerweise sind die Parameternamen in der Grossschreibweise angegeben. Die Reihenfolge wurde aber nicht klar spezifiziert und ist abhängig je nach Service.

2.2.8 Wep Map Service Operationen

Folgende Hauptoperationen sind im Web Map Service definiert: GetCapabilities, GetMap
Optionale Operationen sind GetFeatureInfo, DecribeLayer und GetLegendGraphic.

2.2.9 GetCapabilities

GetCapabilities offeriert die Metadaten des angefragten Services.
Folgende Parameter werden in der Anfrage angeboten:

- VERSION=version [Optional] (x.y.z)
- SERVICE=WMS [Pflicht]
- REQUEST=GetCapabilities [Pflicht]
- FORMAT=MIME_type [Optional] (Standard: text/xml)
- UPDATESEQUENCE=string [Optional] (Datenstatus auf dem Server. Wird hochgezählt wenn die Daten aktualisiert werden.)

Antwort Parameter

Name und Titel Der Name ist ein String für die Machine-to-Machine Kommunikation, der Titel wird für den Benutzer angeboten.

Generelle Servicedaten Name, Titel, Online Ressource URL aber auch Abstract, Kennwortliste, Kontakt, Kosten, Verbindungsvoraussetzungen und Layerlimite in einer Anfrage

MaxWidth und MaxHeight [Optional] Limitiert die Kartengrösse in der GetMap-Anfrage. Wenn nicht angegeben, wurde kein Limit für die Anfrage gesetzt.

Layer Eigenschaften

Über Layers werden einzelne zueinander stehende Kartenelemente geordnet.

Folgende Parameter sind in den Layers zu finden:

- **Title** [Pflicht] für Benutzer lesbare Präsentation des Layers. Nur wenn der Layer einen Namen besitzt ist dieser ein Kartenlayer und kann per GetMap-Anfrage geladen werden.
- **Abstract** [Optional] Beschreibung des Kartenlayers.
- **Kennwort** [Optional] Für die Katalogisierung des Layers nutzbar.
- **EX_GeographicBoundingBox** [Pflicht] Standardboundingbox für den Layer. westBoundLongitude, eastBoundLongitude, southBoundLatitude und northBoundLatitude. Dezimal Gradwerte.
- **CRS** [Pflicht] mindestens ein CRS muss unterstützt werden.
- **BoundingBox** [Pflicht]
  Attribute: *CRS*
  , *minx, miny, maxx, maxy*(Limite der Boundingbox)
  , *resx & resy* (Optionale spatiale Auflösung)
- **Scale denominators** Kartenmassstab des Layers, kann aber je nach Anzeige variieren und ist kein genauer Indikator. Wie im Artikel des GeoServer-Forums (Rajjad, 2010) ersichtlich, ist dieser Wert kaum für Berechnungen geeignet.
- **Attribution**
- **Identifier and AuthorityURL**
- **DataURL**

Layer Attribute

Ein Layer kann folgende Attribute aufweisen:

- **queryable** Wird *GetFeatureInfo* unterstützt?
- **cascaded** Wird ein Layer von einem anderen Server bezogen, erhöht sich dieser Zähler um Eins.
- **opaque** Impliziert ob die Flächen Daten haben oder ob diese leer sein können. Karten zum Beispiel sollten einen wahren Wert haben und sind in der Regel die hintersten Layer. Layer mit Punkte und Linien haben idealerweise einen negativen Wert in diesem Attribut.
- **noSubsets** Der WMS kann nur die gesamte Karte darstellen, ansonsten sind auch Ausschnitte möglich.
- **fixedWidth** Der WMS kann den Wert nicht dynamisch anpassen falls wahr.
- **fixedHeight** Der WMS kann den Wert nicht dynamisch anpassen falls wahr.

2.2.10 GetMap

Diese Anfrage retourniert eine Karte als Output. Sofern der Service keine Karte generieren konnte, wird ein Exception zurückgegeben.

Anfrage Parameter

- **VERSION** [Pflicht] x.y.z (Bsp. 1.3.0)
- **REQUEST** [Pflicht] GetMap
- **LAYERS** [Pflicht] Kommaseparierte Layerliste
- **STYLES** [Pflicht] Kommaseparierte Styleliste
- **CRS** [Pflicht]
- **BBOX** [Pflicht] minx,miny,maxx,maxy
- **WIDTH** [Pflicht] Integer
- **HEIGHT** [Pflicht] Integer
- **FORMAT** [Optional] Outputformat
- **TRANSPARENT** [Optional] Hintergrund Transparenz (false)
- **BGCOLOR** [Optional] Hintergrundfarbe (0xffffff)
- **EXCEPTION** [Optional] Exceptionformat (xml)
- **TIME** [Optional]
- **ELEVATION** [Optional]
- **VENDOR PARAMETERS** [Optional]

2.3 Koordinatensysteme

2.3.1 EPSG

EPSG, kurz für European Petroleum Survey Group Geodesy, ist ein weltweit eindeutiger 4- bis 5-stelliger Schlüssel für Koordinatenreferenzsysteme und andere geodätische Datensätze. Die bekanntesten Koordinatensysteme sind:

- **EPSG:4326** Das World Geodetic System 1984 (WGS 84) ist ein geodätisches Referenzsystem als einheitliche Grundlage für Positionsangaben auf der Erde. Das GPS nutzt dieses Koordinatensystem.
- **EPSG:3857** Der Web Mercator ist eine Variante der Mercator Projektion. Es wird als Standard für Webkarten verwendet. Vorteil dieser Projektion ist die Vereinfachung der Koordinaten zu planaren Flächen. Dies geschieht zulasten der Genauigkeit.

2.3.2 Koordinatenachsen

Die geografischen Koordinaten werden als Länge- und Breitengrade angegeben. Im Englischen heissen diese Latitude und Longitude. Die Reihenfolge der Angaben ist nicht definiert und variiert je nach Anwendung. Auf Karten herrscht die Reihenfolge lat/lon vor. Die Angabe im lat/lon hat einen geschichtlichen Hintergrund. Aufgrund fehlender Messmethoden der Längenachse (Longitude) kam dieses Mass erst später hinzu. Daher wird auf Karten immer wieder gerne zuerst auf die Breitenachse verwiesen, bevor die Längenachse angegeben wird. Aus mathematischer und praktischer Sicht wird aber die umgekehrte Angabe bevorzugt. Also zuerst die x-Achse und danach die y-Achse, kurz Lon/Lat. Eine Auflistung von Programmen, Frameworks und deren bevorzugte Reihenfolge, ist auf der Webseite von MacWright, o.D. ersichtlich.

2.4 Auflösung der Rasterkarten

2.4.1 Hintergrund

Um bei einem WMS einen Kartenausschnitt anzufordern, wird neben der Bildeingrenzung (Bounding Box) auch die Auflösung in Pixel gefordert. Damit die Auflösung im Verhältnis zur Kartengrösse steht, müssen die Seitenverhältnisse berücksichtigt werden.

2.4.2 Problem

Die Schwierigkeit besteht darin, dass der Kartenausschnitt in den verschiedenen Koordinatensystemen nicht rechtwinklig sein muss. So sind die geografischen Punkte der Bounding Box nicht in einem Rechteck angeordnet, sondern befinden sich auf einer Kugel. Es kann also nicht mit normalen Achsen gerechnet werden.

2.4.3 Zwischenlösung

Leider konnten wir uns aus zeitlichen Gründen dem Problem nicht mehr widmen. Aus diesem Grund entschieden wir uns für die einfache Lösung, dass die Auflösung manuell angegeben werden muss. Innerhalb der Rasterkarte sind die Koordinaten der Bounding Box vorhanden und daher geografisch zuweisbar.

Kapitel 3

Eingesetzte Technologien

3.1 Python

Python ist eine universelle, interpretierte höhere Programmiersprache. Es werden mehrere Programmierparadigmen unterstützt. Dazu gehört die objektorientierte, aspektorientierte und funktionale Programmierung. Typisch für Python ist der gut lesbare und knappe Programmierstil. So wird unter anderem auf geschweifte Ausdrücke oder Semikolons verzichtet. Wegen ihrer klaren und übersichtlichen Syntax gilt Python, als einfach zu erlernen. Als Skriptsprache besitzt Python ein offenes und gemeinschaftsbasiertes Entwicklungsmodell. So findet sich diese Programmiersprache vorwiegend in akademischen Kreisen und bietet eine Vielzahl von Standardbibliotheken oder Pakete mit Teillösungen in vielen Bereichen.

3.1.1 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 des Frameworks werden für die fehlenden oder zu wenig umfassenden Funktionen stets neue Lösungen erstellt oder erweitert. Auch für den GeoConverter müssen weitere Plugins installiert werden, um gewissen Anforderungen gerecht zu werden.

3.1.2 Plugins

Django REST framework

http://www.django-rest-framework.org/
Das Frontend des GeoConverters 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 Framework Plugin entschieden. Das Plugin punktete mit umfassenden Funktionalitäten, einfacher Einbindung und guter Testbarkeit.

django-rq

https://pypi.python.org/pypi/django-rq
Django-RQ ist ein Django basiertes Plugin des Python Redis Queue. Über die Redis Queue können neue Hintergrund-Tasks gestartet und betrieben werden. Diese Tasks werden im Hintergrund abgearbeitet und sind asynchron. Der GeoConverter nutzt diese Technologie, um parallele Clientanfragen zu ermöglichen.

django-environ

https://django-environ.readthedocs.io/en/latest/
Django-Environ vereinfacht die Einstellungen der Django-Applikation, indem sie Umgebungsvariablen in eine Datei auslagern lässt.

psycopg2

https://pypi.python.org/pypi/psycopg2
Als Open Source Projekt sollte der GeoConverter 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 «psycopg2» installiert.

appdirs

https://pypi.python.org/pypi/appdirs/
Appdirs vereinfacht den Zugriff auf die Benutzerordnerstruktur. Je nach Betriebssystem werden unterschiedliche Ordnerstrukturen genutzt. Dies wird durch Appdirs vereinfacht, indem über definierte Variablen Zugriff auf die jeweiligen Ordner gewährt wird.

click

http://click.pocoo.org/5/
Click wird genutzt um Kommandofenster Oberflächen zu gestalten.

six

https://pypi.python.org/pypi/six
Kompatibilitäs-Plugin zwischen Pyhton2 und Python3.

Weitere Plugins im Einsatz

- django-filter https://django-filter.readthedocs.io/en/develop/
- markdown https://pypi.python.org/pypi/Markdown
- py https://pypi.python.org/pypi/py/
- pyparsing https://pypi.python.org/pypi/pyparsing
- redis https://pypi.python.org/pypi/redis
- requests http://docs.python-requests.org/en/master/

3.1.3 Codestyle und Testing

pytest-django

https://docs.pytest.org
https://pytest-django.readthedocs.io
Pytest wurde für die statischen Tests innerhalb von Python und Django genutzt. Diese Bibliothek unterstützt das Testen mit eingebauten Fixtures und einfache Einbindung von Testdatenbanken.

So können Transaktionen ohne das Kompromittieren der produktiven Datenbank getestet werden.

flake8

http://flake8.pycqa.org
Für einen sauberen, pythonic Codestyle wurde Flake8 eingesetzt. Dieses Tool prüft den Codestyle und Syntax von Pythoncode.

mccabe

https://pypi.python.org/pypi/mccabe
McCabe wird als Erweiterung von flake8 genutzt. Dieses Plugin wird zur Prüfung von Codekomplexität genutzt.

pyflakes

https://pypi.python.org/pypi/pyflakes
Ein passiver Prüfer von Python Programmen. Pyflake analysiert den Pythoncode und detektiert allfällige Fehler.

pycodestyle

https://pypi.python.org/pypi/pycodestyle
Pycodestyle prüft den Pythoncode nach den PEP8 Konventionen.

pytest-mock

https://pypi.python.org/pypi/pytest-mock
Mocking Plugin für die Pytest Umgebung.

3.2 Vue.js

Vue (ausgesprochen /vju:/ , wie engl. *view*) ist eine progressive JavaScript-Bibliothek um Webinterfaces aufzubauen. Im Gegensatz zu anderen monolithischen Frameworks, ist Vue.js so konzipiert, dass es schrittweise adaptiert werden kann. Dank seiner intuitiven API lässt es sich ausserdem sehr schnell erlernen.
Die Kernbibliothek konzentriert sich auf den View-Layer und ist deshalb einfach mit anderen Bibliotheken zu kombinieren. Dies ist sehr wichtig für den GeoConverter, da er auf die JavaScript-Bibliothek Leaflet für die Kartenansicht angewiesen ist.
Für den GeoConverter wird Vue.js in der Version 2.0 verwendet.

3.3 Vue.js Module

3.3.1 BootstrapVue

https://bootstrap-vue.github.io/
Die BootstrapVue-Bibliothek hilft dabei, auf einfache Weise Bootstrap 4 Komponenten in Vue.js

einzubinden. Sie beinhaltet Komponenten für die einzelnen Bootstrap Elemente. Durch die Verwendung von BootstrapVue wird kein jQuery mehr benötigt, da diese Funktionalitäten in den Vue Komponenten direkt implementiert sind.

3.3.2 Leaflet

http://leafletjs.com/
Leaflet ist eine Open Source Javascript-Bibliothek, mit der Karten dargestellt werden können. Es werden WMS, WMTS und GeoJSON sowie Bildüberlagerungen unterstützt. Durch Plug-ins können noch weitere Typen von Ebenen unterstützt werden wie KML, CSV, WKT, GPX etc.
Die Leaflet-Bibliothek wird benötigt, um den in der Applikation ausgewählten WMS-Layer darzustellen.

3.3.3 Leaflet.draw

https://github.com/Leaflet/Leaflet.draw
Mit dem Leaflet Draw Package wird das Leaflet Package erweitert. Es beinhaltet Funktionen, um geometrische Formen und Marker auf die Karte zu zeichnen. Mit dieser Erweiterung ist es möglich ein Rechteck auf die Karte zu zeichnen, welches dann als Kartenausschnitt verwendet werden kann.

3.3.4 Vuex

https://vuex.vuejs.org/
Vuex ist eine Bibliothek, welche sich um das State Management kümmert. Sie dient als zentraler Speicher für alle Komponenten in einer Vue Applikation. Durch Regeln wird sichergestellt, dass der State nur in einer vorhersehbaren Weise verändert wird.

3.3.5 vue-router

https://router.vuejs.org/
Der vue-router ist eine offiziell unterstütze Bibliothek. Mit dem vue-router ist es möglich Komponenten den «Routes» zuzuordnen.

3.3.6 vue-resource

https://github.com/pagekit/vue-resource
Dieses Plugin bietet Dienste, um HTTP-Requests zu erstellen und HTTP-Responses zu handhaben. Es kann mit XMLHttpRequest oder JSONP gearbeitet werden.

3.3.7 vue-dropzone

https://github.com/rowanwins/vue-dropzone
Die Erweiterung vue-dropzone beinhaltet eine Komponente für Fileuploads. Es können damit Files via Drag & Drop oder Dateibrowser einem Formular hinzugefügt werden.

3.4 Bootstrap

https://v4-alpha.getbootstrap.com/
Bootstrap ist das weltweit beliebteste HTML, CSS und JavaScript Framework um responsive Webapplikationen zu erstellen. Es enthält vorgestylte HTML-Elemente und vereinfacht die Oberflächengestaltung dadurch enorm. Für den GeoConverter wird Bootstrap in der Version 4 (Alpha) verwendet.

Kapitel 4

Umsetzungskonzept

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

4.1 Features Rasterteil

Das gewählte Vorgehensmodel des GeoConverters fordert iteratives Vorgehen innerhalb des Projektes. Um eine möglichst effiziente Zusammenarbeit zu gestalten, hat das Projektteam die Teilfunktionen und Komponenten in Features aufgeteilt.

4.1.1 URLParams

Über ein Inputfeld im Frontend sollen gültige MapService-URL's an das Backend gesendet werden. Das Backend extrahiert und speichert die Parameter aus der URL und sendet das Resultat als serialisierte Daten an das Frontend zurück.

4.1.2 Capabilities

Das Frontend sendet dem Backend eine gültige MapService-URL. Das Backend sendet an den Mapservice eine Metadatenanfrage. Die Metadaten vom MapService werden im Backend gespeichert. Das Frontend kann diese nun über eine Anfrage beziehen und die Parameter präsentieren.

```
- <Service>
    <Name>WMS</Name>
    <Title>SchweizMobil-Routen WMS</Title>
    <Abstract>Geodienst des GIS-ZH</Abstract>
  + <KeywordList></KeywordList>
    <OnlineResource xlink:href="http://wms.zh.ch/VelonetzZHWMS?"/>
  + <ContactInformation></ContactInformation>
    <Fees>none</Fees>
    <AccessConstraints>none</AccessConstraints>
    <MaxWidth>14336</MaxWidth>
    <MaxHeight>14336</MaxHeight>
  </Service>
+ <Capability></Capability>
</WMS_Capabilities>
```

Abbildung 4.1: Beispiel Capabilities

4.1.3 Kartenansicht

Die Kartenansicht umfasst alle nötigen Features rund um die Kartenansicht innerhalb des Frontends.

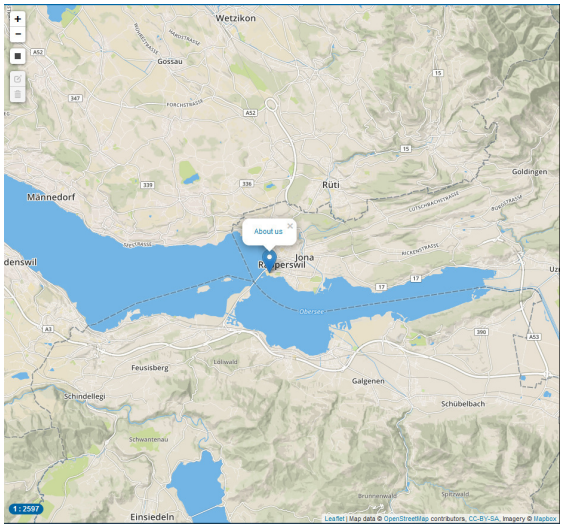


Abbildung 4.2: Kartenansicht GeoConverter mit Leaflet

WMS Ansprechen

Das Frontend kann einen WMS direkt ansprechen und die das Resultat auf der Kartenansicht darstellen.

BBox auslesen

Innerhalb der Kartenansicht kann einen Ausschnitt eingezeichnet und die Koordinatenpunkte des Feldes ausgelesen werden.

4.1.4 WMS Einstellungen

Die WMS-Einstellungen sind die Eingabemöglichkeiten, welche dem Benutzer zur Verfügung gestellt werden. Die Funktionen werden in einzelnen Features umgesetzt.

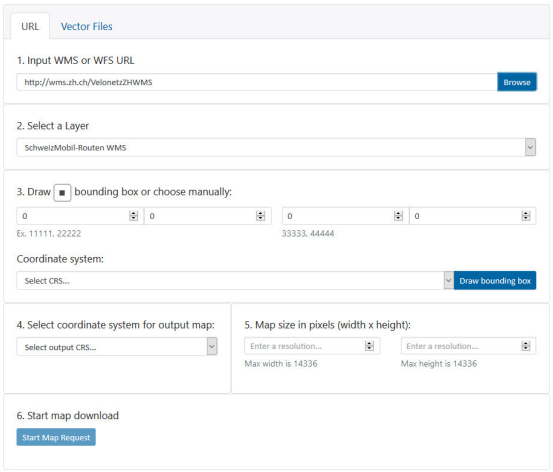


Abbildung 4.3: Einstellungsmöglichkeiten der WMS Anfragen

Update Einstellungen

Das Backend bietet die Updatefunktion für Einstellungsänderungen an. Neue Werte können direkt in die Datenbank gespeichert werden. Als Resultat werden die serialisierten Daten zurückgegeben.

Layer selection

Für die erfolgreiche Anfrage an den Webmapservice müssen, die Kartenlayer gewählt und gespeichert werden können. Dazu wird in der ersten Phase im Frontend eine einfache Möglichkeit implementiert.

4.1.5 Map Request

Wird die Kartenabfrage gestartet, werden die gespeicherten Informationen zum Kartenservice gesendet und der Kartengenerator gestartet. Die erzeugte Karte wird gespeichert und die URL in der Datenbank innerhalb des passenden Requests hinzugefügt. Das Frontend kann die URL über die Request-Abfrage beziehen.

4.1.6 Kartendownload

Über den zugehörigen Link kann die Karte auf dem Server bezogen werden. Das Frontend bietet den Link an, das Backend sichert das Kartenmaterial an der passenden Stelle.

4.1.7 Bildkonvertierung

Das Backend konvertiert über das GDAL die Karten zum gewünschten Format. Wie in der Aufgabenstellung definiert, wird hier das GeoTIFF als allgemeines Rasterformat gewählt.

4.1.8 Koordinaten Transformation

Über einen API-Aufruf sollen die Koordinaten einer Bounding-Box in ein anderes Koordinatensystem transformiert werden können.

4.2 Feature Vektorteil

4.2.1 Formatliste

Über eine API-Anfrage können alle möglichen Formate für Vektorkonvertierungen abgefragt werden.

4.2.2 Konvertierung File

Mittels der in GDAL integrierte OGR-Bibliothek kann eine Vektor-Datei in die von der Formatliste gelieferten Formate konvertiert werden.

4.2.3 Download URL

Über eine URL können von einem WFS-Server Daten bezogen und das gewünschte Format konvertiert werden.

4.2.4 Service Info

Die Auswahl des gewünschten Kartenservices kann über eine API-Anfrage durchgeführt werden. Über eine gültige URL wird ermittelt, um was für einen Service es sich handelt.

4.2.5 File Upload

Um Vektordaten konvertieren zu können, müssen die Daten an den GeoConverter übermittelt werden. Dazu wird im Frontend eine passende Ablage bereitgestellt. Das Backend bietet ein dementsprechendes Datenmodel an und speichert die hochgeladenen Daten lokal ab.

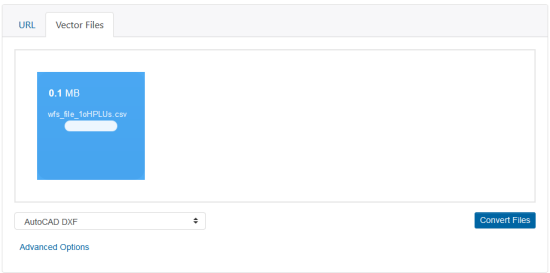


Abbildung 4.4: Einstellungsmöglichkeiten für Vektordaten-Konvertierungen

4.2.6 WFS Einstellungen

Alle möglichen Parameter werden präsentiert. Die gewählten Einstellungen werden dann an das Backend gesendet.

Abbildung 4.5: Einstellungsmöglichkeiten der WFS Anfragen

Kapitel 5

Resultate, Bewertung und Ausblick

5.1 Zielerreichung

Zusammengefasst wurden folgende Ziele in der Aufgabenstellung definiert:

- Funktionierende Webapplikation mit Frontend & Backend
- Responsives User Interface
- Download von Rasterkarten als GeoTIFF ausgehend von einem Webservice
- Funktionen des bestehenden GeoConverters werden integriert
- als Docker Container ausgeliefert

Durch die Definition einer REST-Schnittstelle konnte eine klare Trennung zwischen Frontend und Backend erreicht werden. Mit der Verwendung von Django im Backend, welches bereits viele Funktionen mitliefert, 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.

Mit dem Einsatz von Vue.js ist die Applikation mit einem progressiven JavaScript-Framework geschrieben und somit für die Zukunft gewappnet. Auch die komplette Funktion von Leaflet konnte nun genutzt werden, womit der GeoTIFF Generierung nichts mehr im Weg stand.

Mithilfe von Bootstrap wurde die ganze Applikation responsive gemacht, womit sie sich auf verschiedene Bildschirmgrössen anpasst. In Zusammenarbeit mit dem Betreuer und mithilfe der Mockups, wurde die Usability der Webseite erarbeitet und optimiert. Die daraus resultierende Webseite ist schlicht und passt sich in die Corporate Identity der HSR ein.
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 bildet eine solide Grundlage mit guter Usability.
Der RasterGeoConverter und der bestehende GeoConverter wurden in ein gemeinsames User Interface eingepasst, ohne dass der Benutzer zwischen den beiden Funktionen unterscheiden muss. Der Benutzer kann somit ohne zusätzliches Wissen die Applikation verwenden.

Somit ist es dem Benutzer möglich, aus einem Webservice eine Rasterkarte als GeoTIFF herunterzuladen oder die bewährten Funktionen des GeoConverters zu benutzen. Der Benutzer kann sowohl von einem Webservice die Daten in einem gewünschten Vektorformat herunterladen, als auch seine eigenen Dateien in ein anderes Vektorformat konvertieren.

Die Applikation wurde von Anfang an in einem Docker Container entwickelt und getestet. Somit war während der ganzen Arbeit sichergestellt, dass der GeoConverter in einem Docker Container ausgeliefert werden kann.

5.2 Weiterentwicklung

Der GeoConverter wurde über eine Bachelorarbeit entwickelt und durch die begrenzte Zeit ist auch klar, dass nicht alle Ideen umgesetzt werden konnten. Besonders im Geodatenbereich gäbe es zahlreiche Services, die noch unterstützt werden könnten. Dieser Abschnitt widmet sich insbesondere den zusätzlichen Features, mit welchen der GeoConverter erweitert werden könnte.

5.2.1 Geschätzte Zeit

Zurzeit können dem Benutzer noch keine genauen Angaben gemacht werden, wie lange die Konvertierungsaufträge dauern. Somit ist es dem Benutzer nicht ersichtlich, wie lange er auf seinen Auftrag warten muss. Dem Benutzer könnte die durchschnittliche Zeit für eine Kartenanfrage beim Start der Konvertierung angegeben werden. Die Zeit könnte pro Service, Internetgeschwindigkeit und Kartengrösse berechnet werden.

5.2.2 Logging

Damit später Auswertungen über die Nutzung gemacht werden können, könnten die Konvertierungsaufrufe in einem Log erfasst werden.

5.2.3 WMTS

Neben den Web Map Service für rasterbasierende Daten, gibt es noch den Web Map Tile Service, kurz WMTS, als Dienst für digitale Karten, welche kachelbasiert sind. Die Kacheltechnik, wie sie in diesem Service angewendet wird, funktioniert ähnlich zu einem Mosaik. Die Karten werden in mehrere Stücke 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 GeoConverter mit den spezifischen Funktionen erweitert werden.
Die Rasterfunktion könnte somit mit Anfragen an WMTS Server erweitert werden und somit das Beziehen und Bereitstellen von kachelbasierten Karten unterstützen.

5.2.4 Sharing

Der GeoConverter 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 ein Link für seinen erzeugten Download bereitgestellt werden, welchen er dann mit anderen Benutzern teilen kann.

Teil II

Softwareprojekt Dokumentation

Kapitel 6

Anforderungsspezifikation

In den Anforderungsspezifikationen werden die Ansprüche an den GeoConverter 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

Ziel dieser Arbeit ist es eine Webapplikation, genannt GeoConverter, mit den folgenden Funktionen zu erstellen:

- Download von Rasterkarten im GeoTIFF-Format, ausgehend von einem Webservice
- Konvertierung von Vektordaten in verschiedene Formate, ausgehend von einem Webservice oder einem File (Funktionalität des bestehenden GeoConverters)

6.1.2 Optionale Ziele

Die folgenden Ziele sind als optional Definiert:

- Unterstützung von WMTS

6.2 Nicht funktionale Anforderungen

Die folgenden nicht funktionalen Anforderungen wurden der Aufgabenstellung entnommen:

6.2.1 Sprachen

Das Frontend sowie der komplette Source Code soll Englisch sein.

6.2.2 Technologien

Die Anforderungen an die eingesetzten Technologien sind durch die Aufgabenstellung sowie durch den bestehenden GeoConverter vorgegeben:

- Frontend: Vue.js
- Backend: Python, Django,
- Datenbank: Bevorzugt PostgreSQL
- Auslieferung in Docker Container
- OGC Specifications zu WMS/ WMTS/ Web Feature Service (WFS) werden eingehalten

6.2.3 Qualität

Zur Sicherung der Softwarequalität wurden folgende Anforderungen bestimmt:

- Backend: PEP-8 als Coderichtlinie
- Frontend: Airbnb JavaScript Style Guide

6.2.4 Daten

- Von öffentlich zugänglichen Webdiensten
- Es werden keine Benutzerdaten gesammelt

6.2.5 Design

Das HSR Corporate Identity wird eingehalten

6.2.6 Usability

- Responsive User Interface

6.3 Szenarien

6.3.1 GeoTIFF Generierung

Immer mehr Kartenmaterial wird über Web Map Services angeboten. Da WMS jedoch nicht direkt in die Planungstools von Architekten und Planern eingebunden werden können, benötigen diese ein GeoTIFF.
Ein Benutzer besitzt eine Uniform Resource Locator (URL) eines WMS, von welchem er Kartenmaterial beziehen möchte. Er gibt die URL im GeoConverter ein **(UC01)** und bekommt eine Auswahl an Einstellungen **(UC04)**. Der Benutzer wählt, welchen Layer des WMS er benötigt, und wählt einen Kartenausschnitt auf der Karte aus **(UC05)**. Anschliessend wählt er noch, in welchem Koordinatensystem er sein GeoTIFF benötigt und welche Auflösung das GeoTIFF haben soll.
Nun kann der Benutzer die Bildgenerierung starten **(UC07)** und nach erfolgreichem Abschluss, kann der Benutzer sein GeoTIFF herunterladen **(UC10)**.

6.3.2 Datenkonvertierung

Ein Benutzer besitzt Vektordaten als Datei **(UC06)** oder eine URL zu einem WFS **(UC01)** und möchte daraus eine Datei in einem anderen, vektorbasierten Format erstellen. Hierfür wählt der Benutzer seine Datei aus und zieht sie per Drag & Drop in den Uploadbereich oder schreibt eine URL ins Eingabefeld. Nun wird das Ausgabeformat gewählt und die Datenkonvertierung **(UC07)** gestartet. Nachdem die Konvertierung beendet wurde, kann der Benutzer das Resultat als ZIP-Datei herunterladen **(UC10)**.

6.4 Use Case Diagramm

Obwohl eigentlich nur zwei grosse Userstories bestehen, haben wir die Use Cases in kleinere Pakete aufgeteilt. Mit dieser Aufteilung können wir gezielter auf die Funktionen und Interaktionsmöglichkeiten des GeoConverters eingehen.

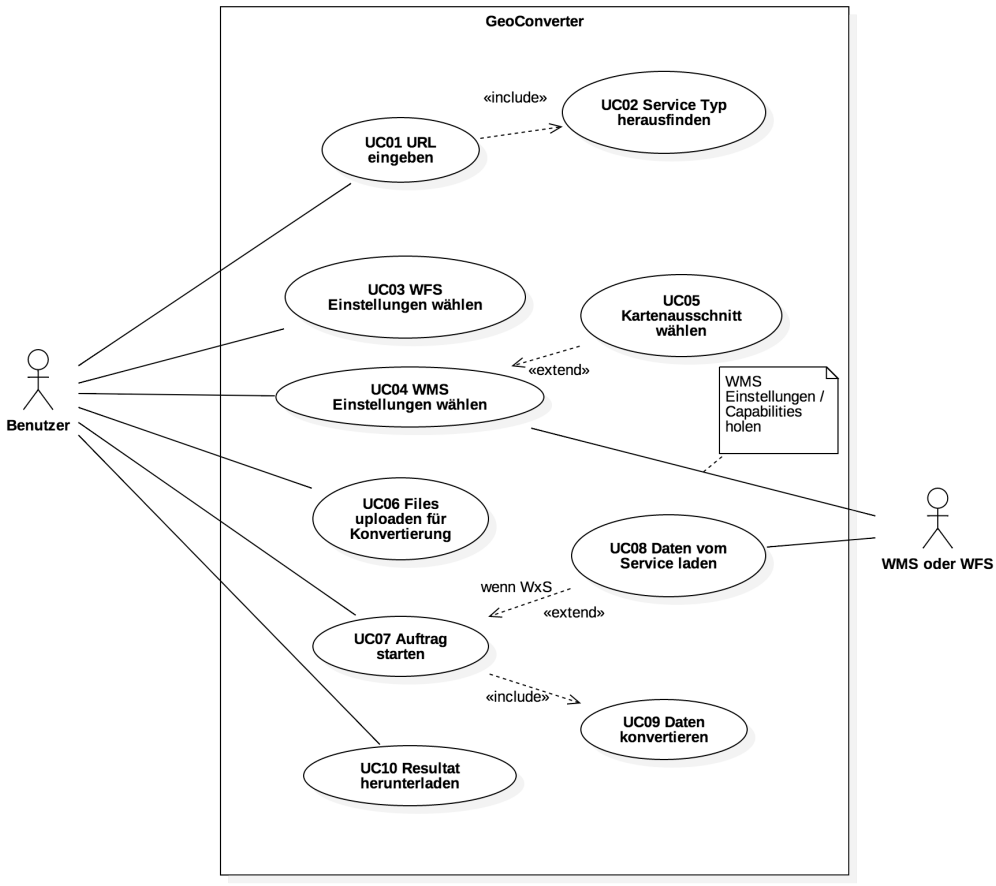


Abbildung 6.1: Use Case Diagramm

6.5 Akteure

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

6.5.1 Benutzer

Der Benutzer ist Hauptakteur des GeoConverters. Er möchte Kartenmaterial beziehen oder konvertieren. Er gibt eine URL oder eine Datei vor und startet die Konvertierungsaufträge.

6.5.2 Web Service für Karten (WxS)

Die Web Services bieten verschiedene Kartenmaterialien an und sind unterstützende Akteure. In zwei Anwendungsfällen werden die Daten über einen Web Service bezogen und anschliessend für den Benutzer in das gewünschte Format konvertiert.

6.6 Use Case Brief

Nachfolgend werden alle Use Cases im Brief Format beschrieben.

6.6.1 UC01: URL eingeben

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Der Benutzer trägt in das Eingabefeld des GeoConverters eine URL eines Web Map Service (Allgemein) (WxS) ein. Der Benutzer klickt auf «Browse». Der GeoConverter überprüft, ob die URL gültig ist, extrahiert zusätzliche Parameter aus der URL (falls vorhanden) und probiert den *Service Typ herauszufinden* (Include UC02).

6.6.2 UC02: Service Typ herausfinden

Akteure: WxS
Beteiligte Systeme: GeoConverter
Standardszenario:
Der GeoConverter fragt den Service nach seinen Metadaten an und speichert diese ab.

6.6.3 UC03: WFS Einstellungen wählen

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Der Benutzer muss das Ausgabeformat wählen und kann noch zusätzliche Einstellungen tätigen.

6.6.4 UC04: WMS Einstellungen wählen

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Der Benutzer selektiert den gewünschten Layer. Anschliessend gibt er per Koordinaten den Ausschnitt an, welchen er möchte oder *wählt einen Kartenausschnitt* (Extend UC05) auf der Karte. Nun wählt er noch das gewünschte Koordinatensystem und die Auflösung der Ausgabe.

6.6.5 UC05: Kartenausschnitt wählen

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Der Benutzer wählt das Rechteck Werkzeug aus und zeichnet damit ein Rechteck auf die Karte. Dies ist sein gewählter Kartenausschnitt.

6.6.6 UC06: Files uploaden für Konvertierung

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Der Benutzer wählt die Dateien aus und zieht sie via Drag & Drop in den Uploadbereich, oder er klickt auf den Uploadbereich und wählt die Dateien via Auswahlfenster.

6.6.7 UC07: Auftrag starten

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Der Benutzer startet die Konvertierung. Dabei werden die Daten und Einstellungen des Benutzers an den Server übermittelt. Die *Daten werden konvertiert* (Include UC09) und als Download bereitgestellt.
Alternative Szenarios:
Sollte der Benutzer den Auftrag mit einer URL gestartet haben, so wird der entsprechende Service angefragt und die *Daten vom Service geladen* (Extend UC08).

6.6.8 UC08: Daten vom Service laden

Akteure: WxS
Beteiligte Systeme: GeoConverter
Standardszenario:
Der GeoConverter ladet die entsprechenden Daten vom Service und speichert sie ab für die Weiterverarbeitung.

6.6.9 UC09: Daten konvertieren

Beteiligte Systeme: GeoConverter
Standardszenario:
Der GeoConverter konvertiert die Daten in das gewünschte Format und stellt sie als ZIP Archiv zum Download bereit.

6.6.10 UC10: Resultat herunterladen

Akteure: Benutzer
Beteiligte Systeme: GeoConverter
Standardszenario:
Dem Benutzer wird der Download als Link dargestellt. Er kann auf den Link klicken und das ZIP Archiv herunterladen.

Kapitel 7

User Interface Design

Aus der Aufgabenstellung wurde entnommen, dass das GUI responsive sein soll. Als Vorlage für das Userinterface diente der in der Studienarbeit erstellte RasterGeoConverter.

7.1 Designentwicklung

Durch die vorhergehende Studienarbeit wurde das Design bereits einmal evaluiert. Nun konnte aus den daraus gezogenen Erkenntnissen ein neues Frontend gestaltet werden. Die Grundelemente des User Interface sind dieselben wie im RasterGeoConverter geblieben, da die Funktionalität, welche die Elemente bieten müssen, immer noch dieselbe ist. Was die Anordnung und Usability angeht, so besteht Verbesserungspotenzial.

So soll nochmals auf die 10 Prinzipien von Nielsen (1995) eingegangen werden. Ein sehr wichtiger Punkt daraus ist *Visibility of system status*, was im RasterGeoConverter noch nicht so gut umgesetzt war. Im GeoConverter soll nun vermehrt darauf geachtet werden, dass dem Benutzer ersichtlich ist, wenn das System am Arbeiten ist. Auch soll auf *Aesthetic and minimalist design* geachtet werden, da das UI aus sehr vielen Bedienelementen besteht. Vor allem mit den zusätzlichen Funktionen des bestehenden GeoConverters ist es wichtig, dass das UI übersichtlich bleibt.

Um das Userinterface zu entwickeln, wurden Mockups erstellt, welche jeweils während den Sitzungen präsentiert und bewertet wurden.

7.2 Erster Entwurf

Das erste Mockup 7.1 sah dem RasterGeoConverter sehr ähnlich. In diesem ersten Entwurf wurde noch nicht auf die später hinzukommenden GeoConverter Funktionen eingegangen. Es wurde jedoch auf das Feedback der Studienarbeit RasterGeoConverter eingegangen. So wurde die Layerauswahl verkleinert, um der Karte mehr Platz zu schaffen.

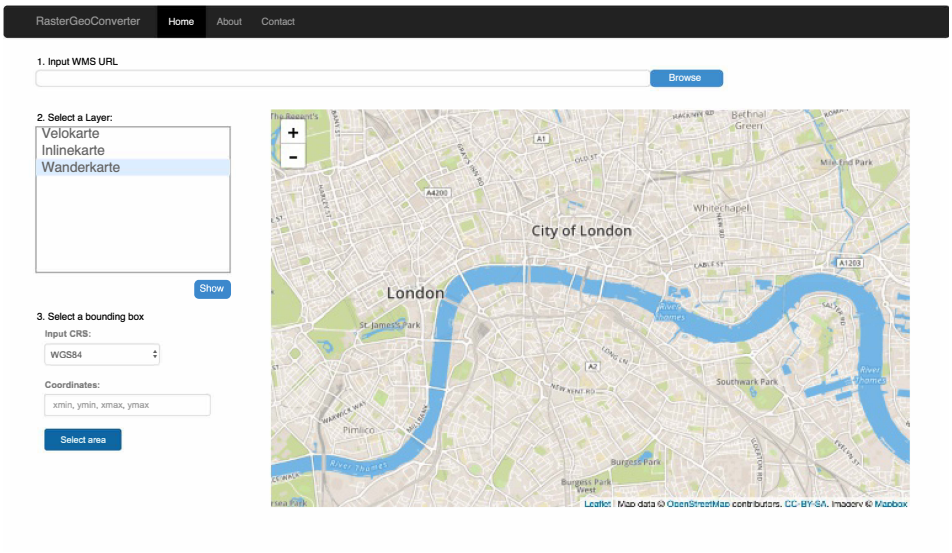


Abbildung 7.1: Erstes Mockup RasterGeoConverter (ohne GeoConverter Funktionen)

7.3 Zweiter Entwurf

Der zweite Entwurf widmete sich immer noch den RasterGeoConverter Funktionen.
Da für die Rasterkonvertierung sehr viele Einstellungen getätigt werden müssen, musste dafür mehr Platz geschaffen werden. Deshalb entstand ein Layout, welches in der Mitte geteilt ist. Dadurch wurde mehr Platz für die Einstellungen gewonnen und die Kartenansicht bekommt trotzdem noch genügend Platz.

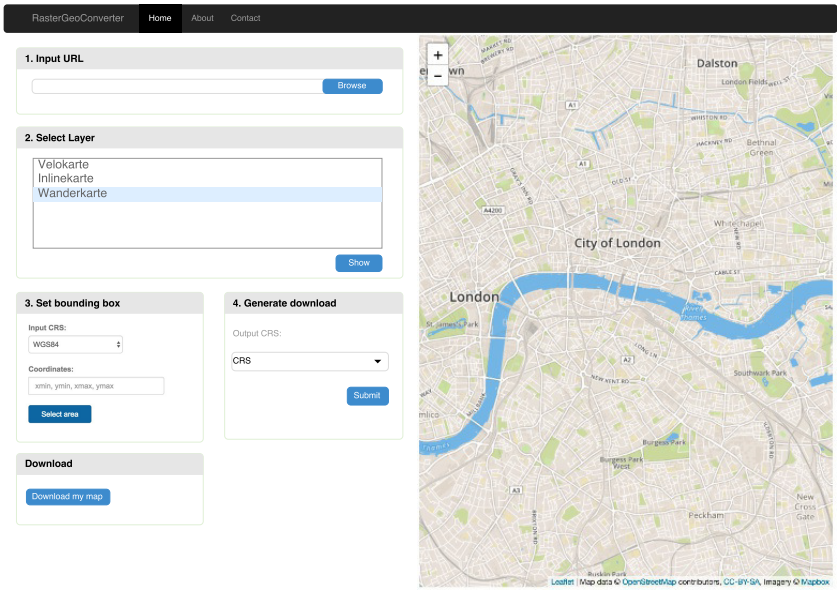


Abbildung 7.2: Zweites Mockup RasterGeoConverter (ohne Geoconverter Funktionen)

7.4 Dritter Entwurf

Da nun ein geeignetes Layout für den RasterGeoConverter gefunden wurde, müssen nun noch die GeoConverter Funktionen in das UI integriert werden. Da das Interface des bestehenden GeoConverters sich sehr vom RasterGeoConverter unterscheidet, war eine erste Idee, die beiden Applikationen über den Datentyp zu unterscheiden. So sollte es einen Menüpunkt Rasterdaten und einen Menüpunkt Vektordaten geben. Vorbild dafür war das Simple Sidebar Template (Bootstrap, o.D.)

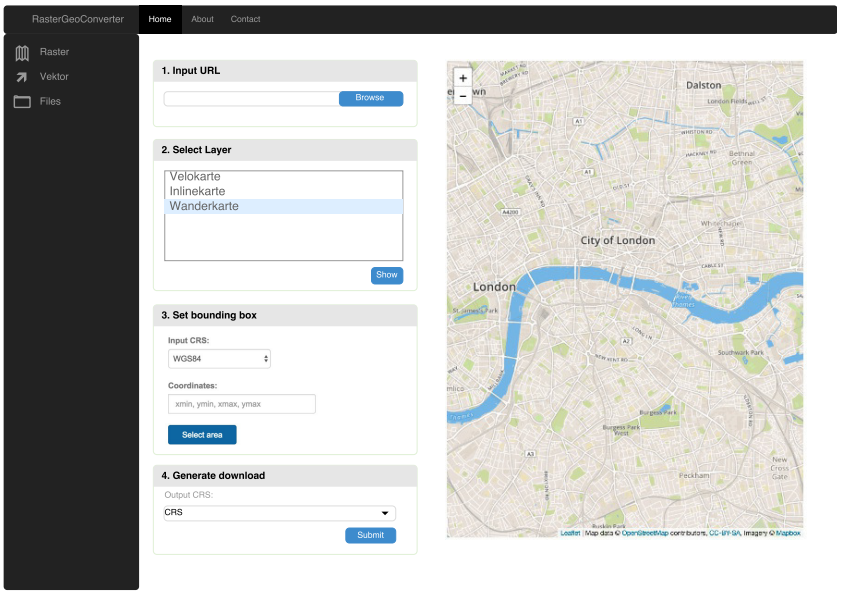


Abbildung 7.3: Mockup mit GeoConverter

Als Feedback für dieses Mockup 7.3 kam zurück, dass viele Benutzer vermutlich nicht wissen, ob es sich bei ihrem Service um Raster- oder Vektordaten handelt. Deshalb musste eine Lösung gefunden werden, in welcher die Trennung der Funktionen für den Benutzer transparent sind. Kurzum, die Funktionen müssen zusammengefasst werden.

7.5 Finaler Entwurf

Nach einer erneuten Analyse der Funktionen des GeoConverters kamen wir zum Entschluss, dass lediglich zwischen URL Input und File upload unterschieden werden muss. Dadurch ist es für den Benutzer viel einfacher, die Applikation zu benutzen. Der Benutzer muss einzig wissen, ob er eine Datei oder eine URL zu einem Service besitzt. Die Entscheidung zwischen den beiden URL-Typen WMS und WFS soll das UI für den Benutzer übernehmen. Der Benutzer muss somit auch nicht über das Wissen von Raster- und Vektordaten verfügen.

Nach diesem vereinfachten Mockup 7.4 sollte nun das GUI des GeoConverters gebaut werden.

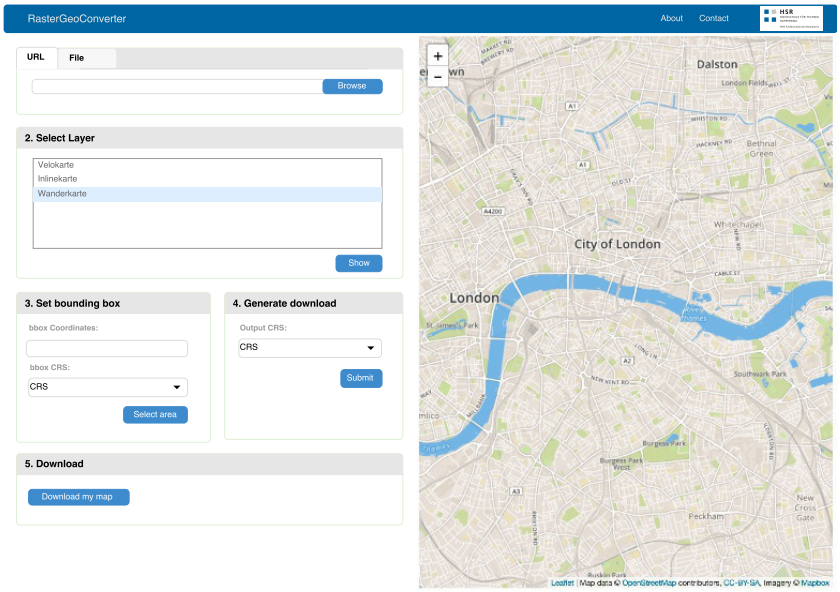


Abbildung 7.4: Finales Mockup mit GeoConverter

Kapitel 8

Software Architektur

8.1 Systemübersicht

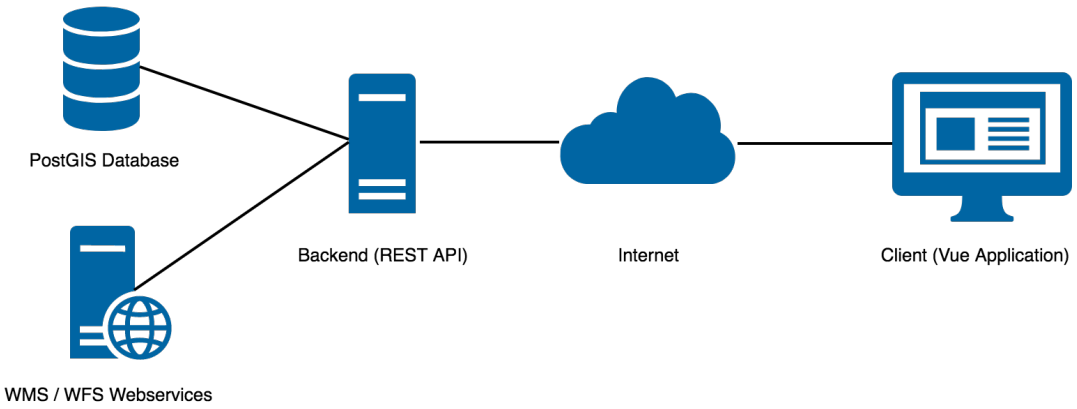


Abbildung 8.1: Systemübersicht

Die Systemübersicht gibt einen Überblick über die verschiedenen Komponenten des GeoConverters. In den Unterkapiteln werden die einzelnen Komponenten genauer beschrieben.

8.2 Sequenzdiagramme

Um den Ablauf der Anfrage zu visualisieren, wurden Sequenzdiagramme erstellt. In diesen Diagrammen wird das erfolgreiche Szenario beschrieben. Wir haben uns bewusst nicht auf die Fehlerszenarien konzentriert, da diese vor allem in Verbindung mit den Geodatenservices stehen. Sobald der Service nicht antwortet oder die Daten nicht verfügbar sind, wird der Anfrageprozess unterbrochen und der Benutzer informiert.

8.2.1 Teilbereiche

Um die Abläufe möglichst umfassend aufzuzeigen, haben wir folgende Teilbereiche identifiziert:

- 1.1 URL
- 1.2 File
- 2.1 WMS

- 2.2 WFS

Als mögliche Eingangsdaten seitens des Benutzers wurden URL's und Dateien definiert. Diese Daten sind der Ausgangspunkt einer jeder Anfrage. Je nach dem, welchen Service diese Eingabedaten ansprechen, wird der WMS- oder WFS-Ablauf aufgerufen.

8.2.2 URL

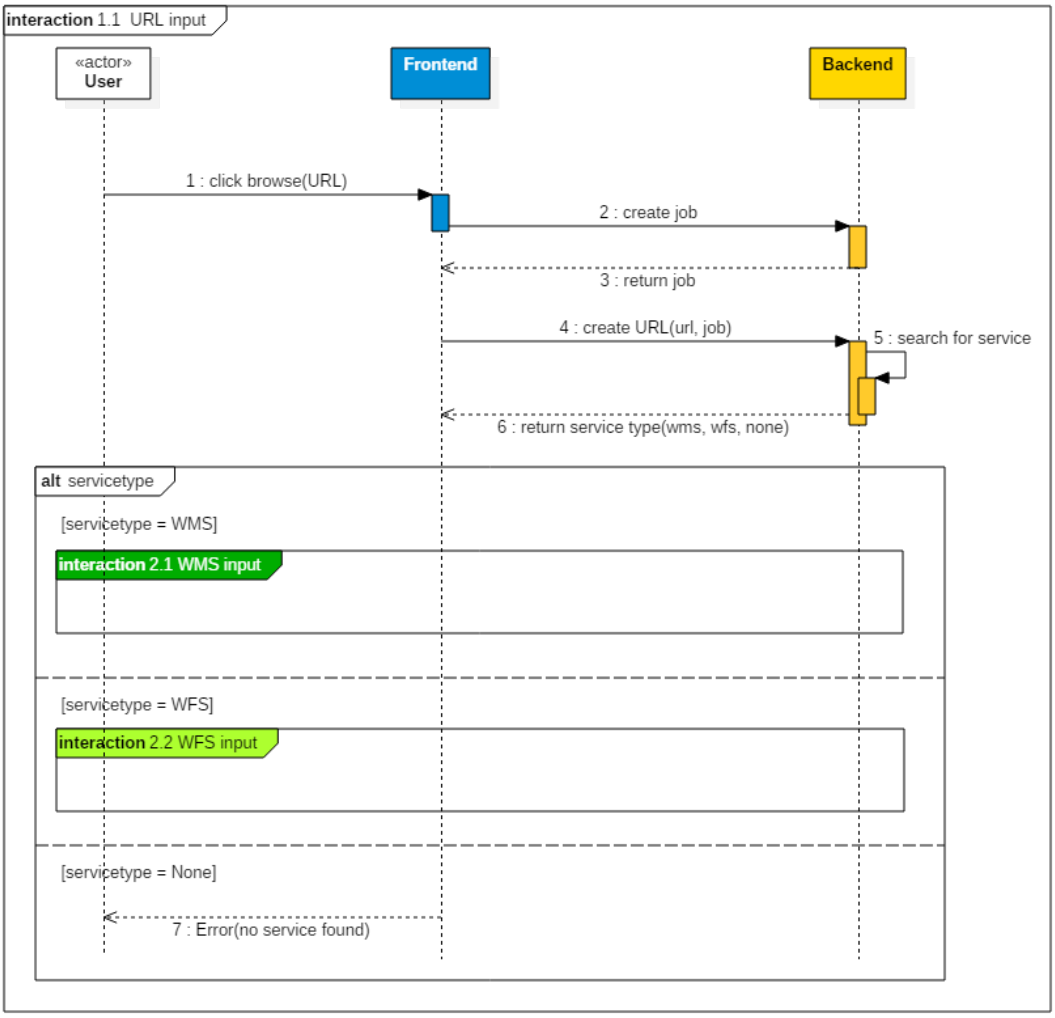


Abbildung 8.2: Sequenzdiagramm URL Input

Ablauf

Der Benutzer sendet eine URL an den GeoConverter. Dadurch wird ein neuer Job, also ein Auftragscontainer, erstellt. An diesem Job wird nun die URL als neuer Auftrag hinzugefügt und zur Servicesuche übergeben. Der GeoConverter ermittelt den passenden Geodatenservice. Drei Szenarien können dadurch ausgelöst werden:

1. WMS Anfrage
2. WFS Anfrage

3. Kein passender Service gefunden

8.2.3 File

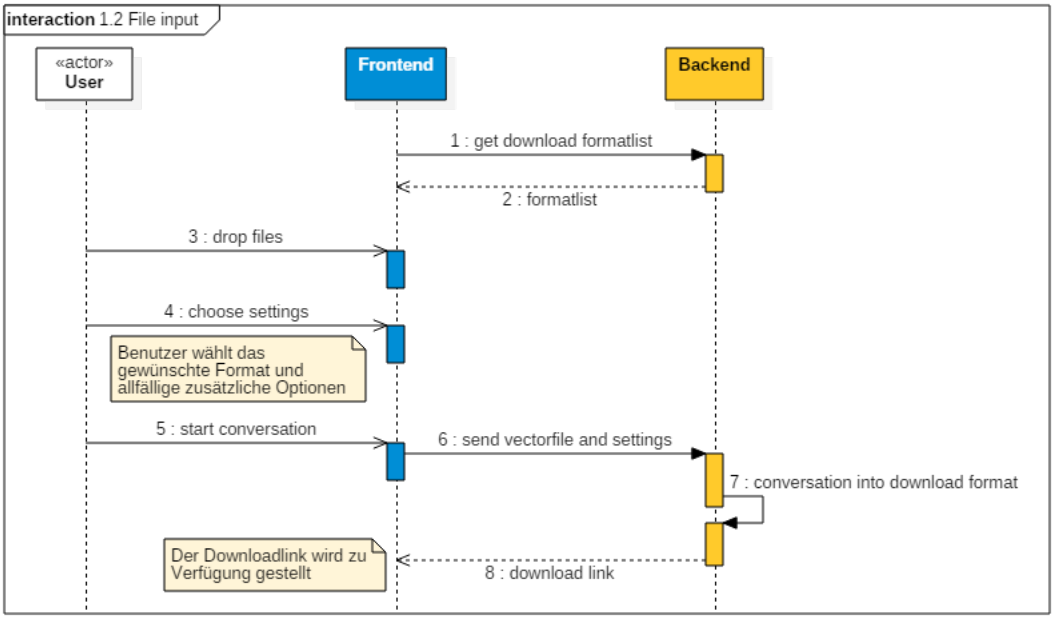


Abbildung 8.3: Sequenzdiagramm File Input

Ablauf

Der Benutzer möchte Geodaten konvertieren lassen. Innerhalb des GeoConverters werden die unterstützten Formate bereitgestellt und dem Benutzer präsentiert. Die zu konvertierenden Daten und Einstellungen werden dem GeoConverter übergeben. Nach der Freigabe der Geodaten werden diese konvertiert und bereitgestellt. Über den generierten Downloadlink kann das Resultat geladen werden.

8.2.4 WMS

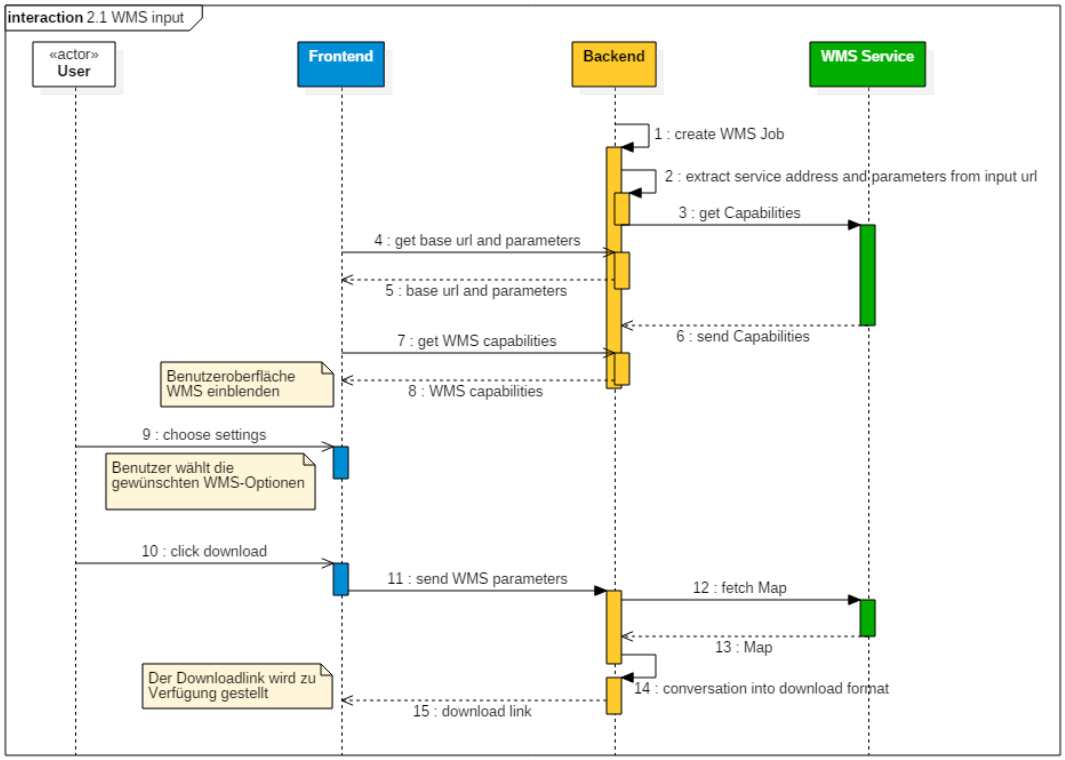


Abbildung 8.4: Sequenzdiagramm WMS Input

Ablauf

Der Ablauf zum Bezug von Rasterkarten aus einem WMS beginnt mit der Erstellung des WMS-Auftrages. Hier werden die Service bezogenen Parameter aus der URL extrahiert und gespeichert. Zudem werden die Metadaten (Capabilities) aus dem Service geladen. Sobald alle Daten verfügbar sind, werden dem Benutzer die Einstellmöglichkeiten präsentiert. Der Benutzer wählt die gewünschten Parameter aus und startet den Auftrag. Nun lädt der GeoConverter die Raster-Karte vom WMS und konvertiert sie in das passende Format. Über den generierten Downloadlink kann das Resultat bezogen werden.

8.2.5 WFS

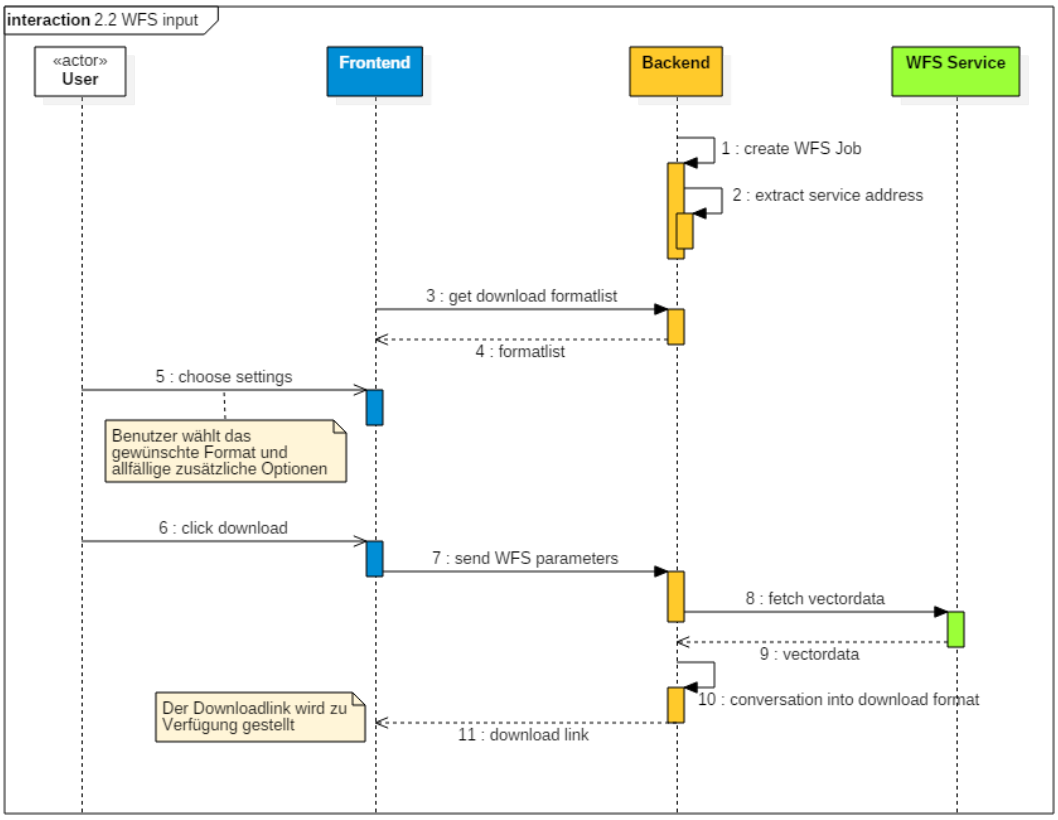


Abbildung 8.5: Sequenzdiagramm WFS Input

Ablauf

Der Ablauf zum Bezug von Vektordaten aus einem WFS beginnt mit der Erstellung des WFS-Auftrages. Die Serviceadresse wird extrahiert und gespeichert. Sobald alle Daten verfügbar sind, werden dem Benutzer die Einstellmöglichkeiten präsentiert. Der Benutzer wählt die gewünschten Parameter aus und startet den Auftrag. Nun lädt der GeoConverter die Vektordaten vom WFS und konvertiert diese in das passende Format. Über den generierten Downloadlink kann das Resultat bezogen werden.

8.3 Layerdiagramm

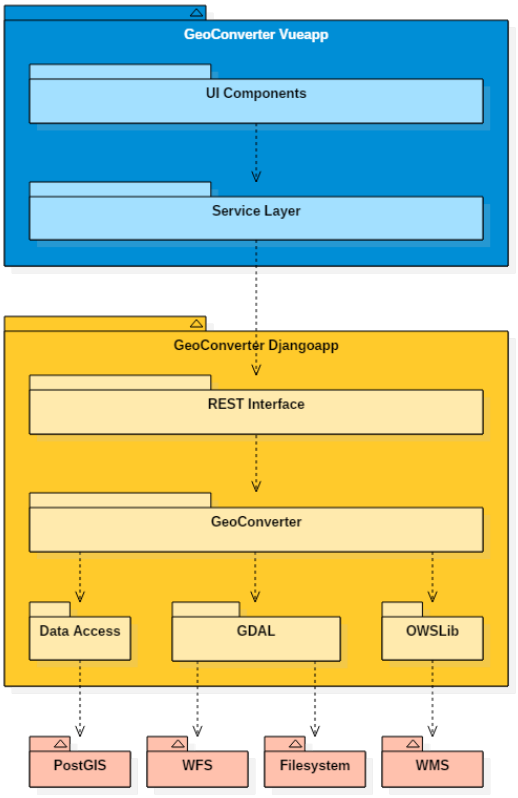


Abbildung 8.6: Layerdiagramm GeoConverter

Der GeoConverter wurde in drei Schichten unterteilt.

Vue-Applikation das Frontend des GeoConverters

Django-Applikation der Service- und Logikbereich des GeoConverters

Datenquelle und externe Services

8.4 Vue-Applikation

8.4.1 Vuex Datenfluss

Für die Applikationsstruktur des Frontends wurde das State Management Pattern von Vuex eingesetzt. Dieses Pattern wurde von Flux, Redux und The Elm Architecture inspiriert. Der Unterschied von Vuex zu diesen Patterns ist, dass es nicht nur ein Pattern ist, sondern auch eine ganze Funktionsbibliothek, welche auf Vue.js zugeschnitten ist.

Die folgenden Erklärungen wurden zu grossen Teilen aus der Vuex Dokumentation (Vue.js, 2017) abgeleitet.
Eine Komponente verfügt über einen «State» (Zustand) welcher die Daten beinhaltet, eine «View» welche die Daten anzeigt und «Actions» welche die Möglichkeiten sind, wie sich der «State» als Reaktion auf Benutzereingaben ändern könnte. «One-way data flow» ist ein wichtiges Konzept,

welches das «State Management Pattern» von Vuex umsetzt. Dieses Konzept ist in einer einzelnen Vue Komponente auch ohne Vuex einfach umzusetzen, da jede Komponente über einen eigenen «State» verfügt welchen sie mit Aktionen ändern kann (vgl. Abbildung 8.7).

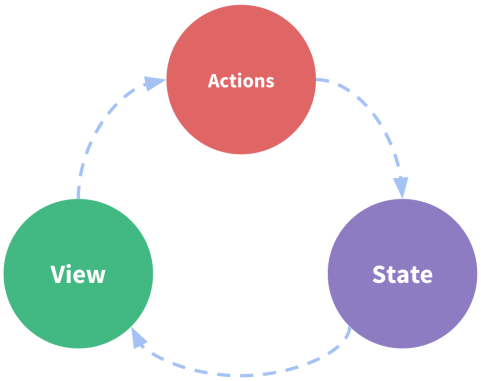


Abbildung 8.7: One-way data flow (Quelle: https://vuex.vuejs.org/)

Das «one-way data flow» Konzept ist jedoch nicht mehr so einfach anzuwenden, sobald mehrere Komponenten einen gemeinsamen Zustand teilen. Dabei können sowohl mehrere «Views» vom selben Zustand abhängen, als auch Aktionen aus verschiedenen «Views» den selben Zustand ändern.
Vuex bietet hierfür eine gute Lösung. Vuex holt den «State» aus den Komponenten heraus und verwaltet ihn in einem globalen Singleton. Nun kann jede Komponente auf den «State» zugreifen oder Aktionen auslösen, um diesen zu ändern.

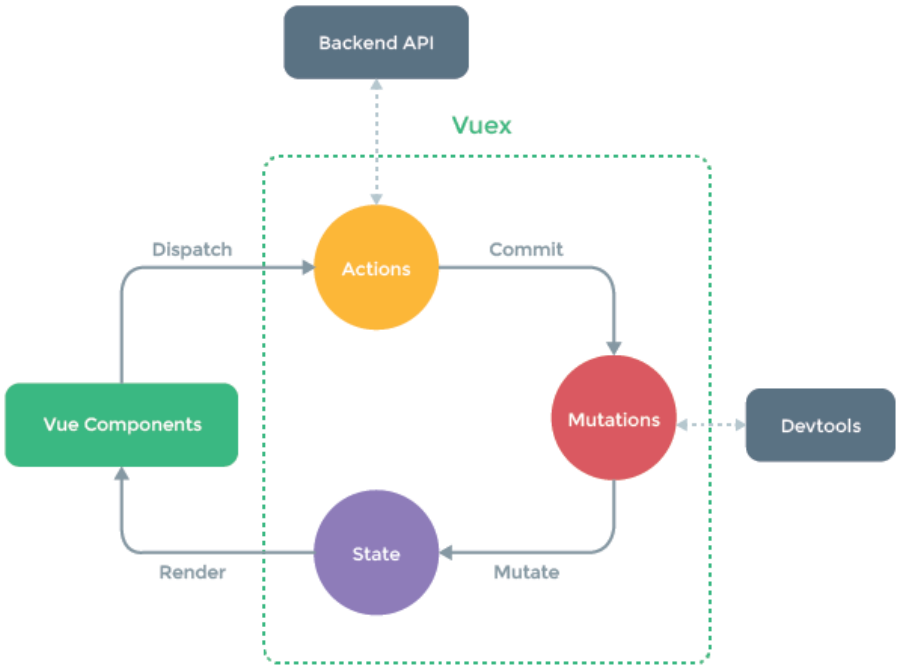


Abbildung 8.8: Unidirektionaler Datenfluss in einer Vuex Applikationsarchitektur (Quelle: https://vuex.vuejs.org/)

Wenn aus einer «View» (Vue Komponente) ein Ereignis ausgelöst wird, beispielsweise ein Buttonklick, muss die «View» eine «Action» auslösen. Die «Action» gibt ihre Parameter an eine Mutation weiter, welche dann die Daten im «State» mutiert. Falls sich Daten im «State» geändert haben, welche eine «View» betreffen, wird die «View» automatisch neu gerendert um die neuen Daten darzustellen.

8.4.2 Architektur

Da es sich beim Frontend um eine Webapplikation handelt, werden Directories und keine Namespaces o.ä. zur Gliederung eingesetzt. Jedes Package im Diagramm 8.9 entspricht einem Directory in der Projektstruktur.

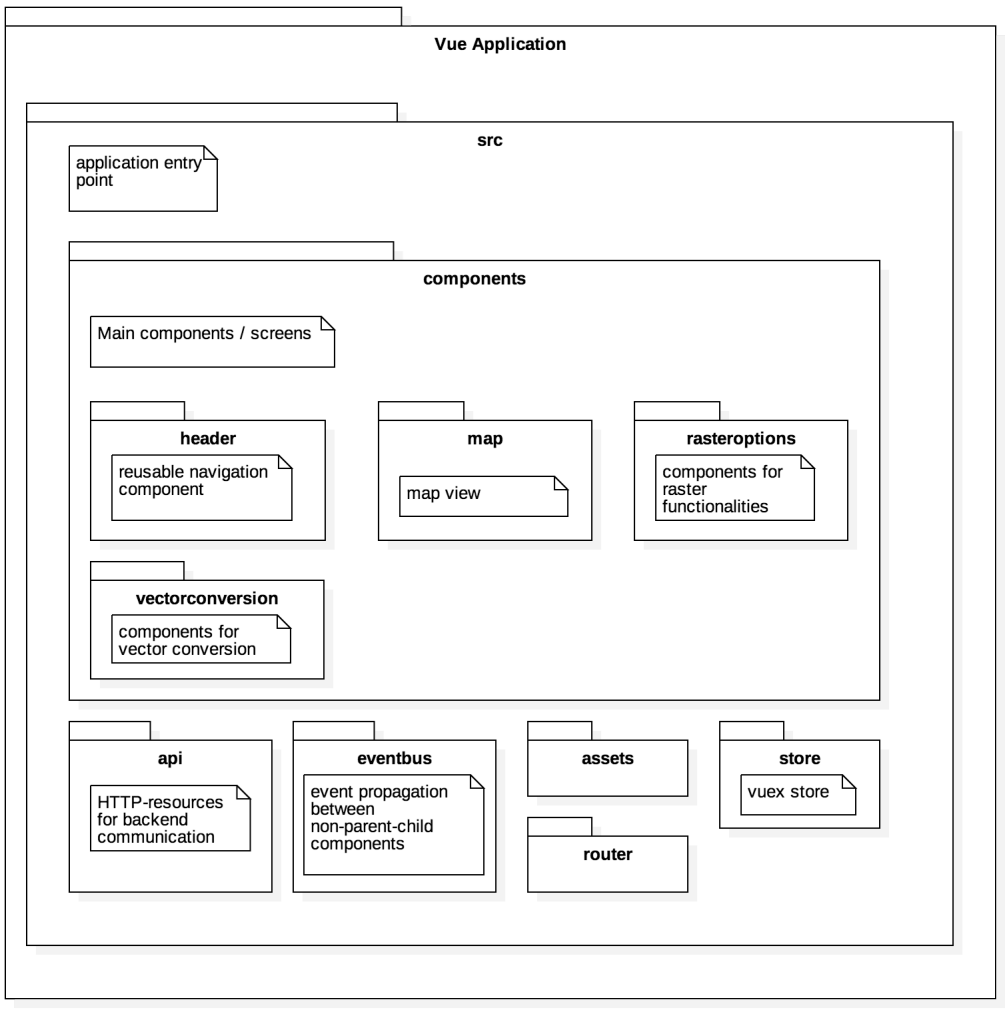


Abbildung 8.9: Projektstruktur Frontend

Api

Der API Client wird benötigt, um auf die API des Backends zuzugreifen. Er ist generisch gehalten, indem sich darin eine Vuex-Store Factory befindet. Mit dieser Factory können für alle API Ressourcen eigene Vuex Stores erstellt werden.

Assets

In den Assets werden Bilder und statische Dateien abgelegt.

Components

Die Komponenten bilden zusammen die jeweiligen Seiten der Applikation. Sie sind in Pakete aufgeteilt, welche die einzelnen Funktionalitäten/Seiten der Applikation abgrenzen. Auf der obersten Ebene befinden sich die Hauptkomponenten.

Eventbus

Der Eventbus wird für die Eventpropagation und Kommunikation zwischen Komponenten verwendet, welche nicht in einem Eltern-Kind Verhältnis stehen.

Router

Der Router verwaltet das ganze Applikationsrouting und stellt somit sicher, dass die richtigen Komponenten jeweils geladen werden.

Store

Für jeden Applikationsbereich der einen «State» benötigt, wird ein Store angelegt. Dieser wird aus der generischen Ressource erzeugt.

8.5 Softwareverteilung

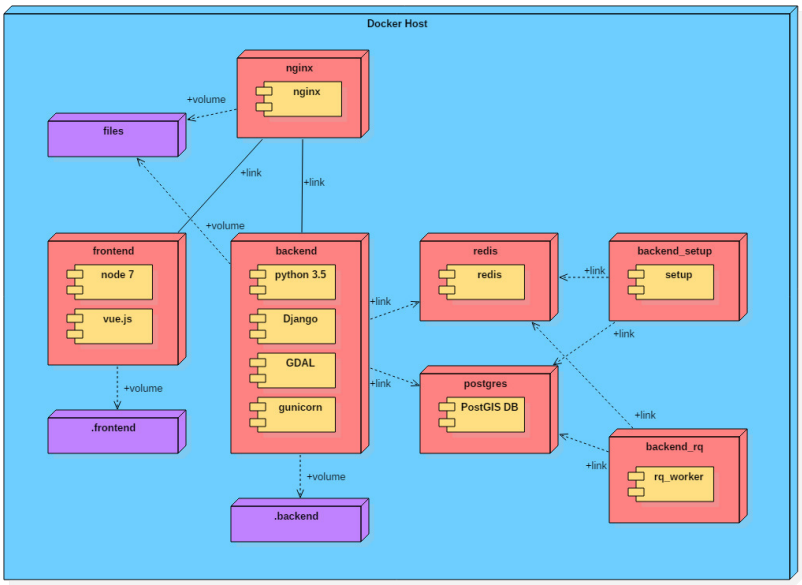


Abbildung 8.10: Verteilungsdiagramm Docker

Der GeoConverter wird als Docker-Container-Gruppe veröffentlicht, installiert und betrieben. Die einzelnen Systemteile werden in eigenständigen Docker-Containern betrieben und sind über ein internes Netzwerk verbunden.

8.5.1 frontend

Der Frontend-Container beinhaltet die Vueapplikation. Über ein angebundenes Volume kann das Frontend Systemdaten teilen.

8.5.2 backend

Innerhalb des Backend-Containers wird die Djangoapplikation betrieben.
Um die Geodaten-Konvertierungen zu ermöglichen, wurde das GDAL hinzugefügt. Über ein angebundenes Volume kann das Backend auf Systemdaten zugreifen und Geodaten speichern.

8.5.3 nginx

Damit alle Teilbereiche zusammengefügt werden können, wird ein Webserver benötigt. Diese Aufgabe übernimmt nginx. Über ein Template werden alle nötigen Zugriffspunkte und Netzverbindungen definiert. Der Webserver stellt zudem den Zugriff auf die gespeicherten Daten sicher.

8.5.4 redis

Der Redis-Container stellt die Auftragswarteschlange bereit. Die Backend-Anfragen werden in diese Warteschlage gestellt und warten auf einen freien Arbeiterthread. So wird das Backend entlastet und garantiert das Bearbeiten paralleler Anfragen.

8.5.5 backend rq

In diesem Container werden die Prozessthreads betrieben. Alle Aufträge werden über diesen Container abgewickelt.

8.5.6 postgres

Als Datenbank wird die GIS-Variante von Postgres eingesetzt.

8.5.7 backend setup

Der Setup-Container bereitet den Backend-Container vor. Über diesen Container wird die Django-Umgebung eingerichtet und stellt sicher, dass der Backend-Container reibungslos startet und betrieben werden kann.

Kapitel 9

Implementation

Der GeoConverter wurde konzeptionell in zwei Komponenten aufgeteilt. Die Vueapplikation bildet das Frontend und die Djangoapplikation das Backend.

Die beiden Komponenten kommunizieren über eine REST-Schnittstelle. Innerhalb des Kapitels Implementation werden die grundlegenden Funktionen erklärt. Das Kapitel wurde wie folgt aufgeteilt:

- Vue-Applikation
- REST API
- Django-Applikation

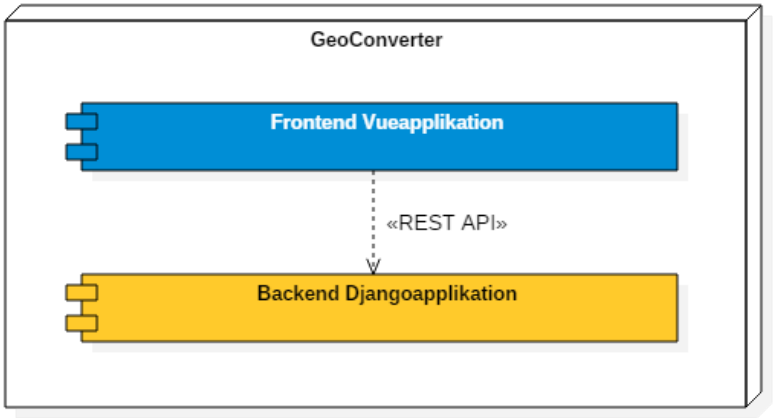


Abbildung 9.1: GeoConverter Komponentenaufteilung

9.1 Vue-Applikation

Das Frontend des GeoConverters wurde in Vue.js implementiert. Durch die moderne Technologie ist es auch für die Zukunft gewappnet.

Ein wichtiges Konzept von Vue ist die Aufteilung in Komponenten. Das Komponentensystem ermöglicht es, Grossanwendungen aus kleinen, eigenständigen Komponenten zu bauen.

Mit den *.vue* Single-File-Komponenten ist Template, Logik und Styling in einem File kombiniert. Diese Single-File-Komponenten sind möglich durch Build-Tools wie Webpack oder Browserify. Build-Tools verarbeiten die Applikation und bauen rekursiv einen Abhängigkeitsgraphen auf, der

jedes Modul enthält, das die Applikation benötigt, und verpackt dann alle diese Module in eine kleine Anzahl von Bündeln - oft nur eines -, die vom Browser geladen werden.

Anhand des Komponentenbaums in Abbildung 9.2 ist ersichtlich, welche Komponenten aus welchen Komponenten bestehen. Bei allen Komponenten im GeoConverter handelt es sich um Single-File-Komponenten. Mehrere Komponenten zusammen bilden die verschiedenen Views des GeoConverters.

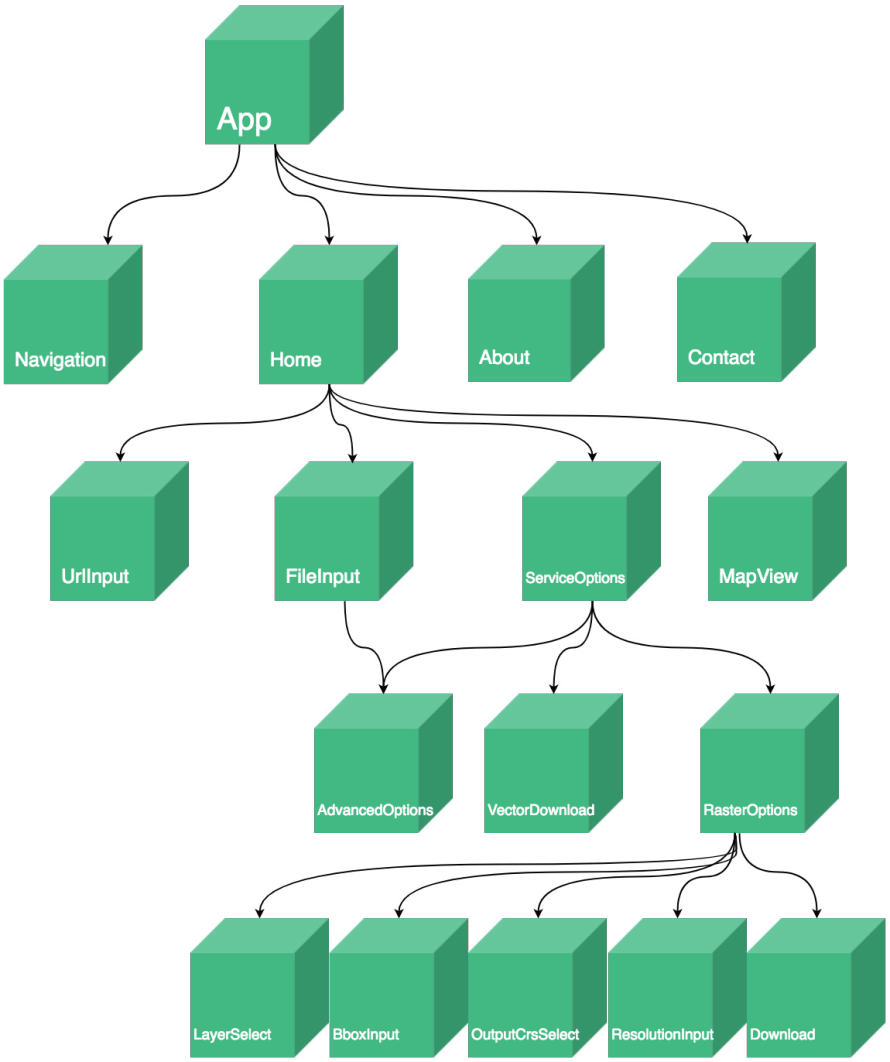


Abbildung 9.2: Komponentenbaum Frontend

Zwei wichtige Elemente der Vueapplikation sind der API Teil und der Vuex Store.

9.1.1 API

Im Directory *api* sind die Zugriffe an die Backend-API geregelt. Mithilfe von *vue-resource* werden im **APIClient** alle HTTP Requests ans Backend getätigt. Dieser unterstützt alle üblichen HTTP-Request Methoden.

VuexResource ist die Basisklasse für alle Vuex Ressourcen. Durch diese Abstraktion ist es sehr einfach einen neuen Vuex Store zu erzeugen. Die Klasse verwendet den APIClient, um die Daten des Stores mit der Backend API auszutauschen. Es können somit Daten vom Backend geladen werden oder Änderungen zurück ans Backend gesendet werden.

Mit dem Ordner *api* ist die ganze Kommunikation mit dem Backend an einem zentralen Punkt geregelt.

9.1.2 Store

Im Directory *store* befindet sich der globale Vuex Store. Dabei werden für die verschiedenen URLs der Backend API einzelne Stores / Ressourcen erstellt. Somit ist jeder Pfad des Backends über eine Ressource zugreifbar. Im folgenden Code ist ersichtlich, welche Routen auf welche Ressourcen gemapped werden.

```
export default new Vuex.Store({
  actions,
  mutations: {
    updateRequest(state, request) {
      state.MapLoad.item = request;
    },
  },
  modules: {
    TransformCrs: createVuexResource('utils/crstransform'),
    Job: createVuexResource('job'),
    Url: createVuexResource('url'),
    WmsUserSettings: UserInputResource, // former MapRequest
    WmsCapabilities: createVuexResource('wms/capabilities'),
    WfsUserSettings: createVuexResource('wfs/userinput'),
    VectorFileFormats: createVuexResource('vector/format'),
  },
  strict: true,
});
```

Listing 9.1: Store-Implementation

Zudem beinhaltet das Directory noch globale Actions, welche über mehrere Stores Aktionen ausführen. Diese Aktionen müssen über keine Ressource aufgerufen werden, sondern können über den globalen Store aufgerufen werden.

9.1.3 RasterOptions

Die Komponente RasterOptions beinhaltet alle Einstellungen/Komponenten, welche für die Erzeugung eines Rasterbildes benötigt werden. Die einzelnen Komponenten teilen sich einen gemeinsamen Store (*WmsUserSettings*) um die Zustände zu sichern.

Dies bedeutet, dass die einzelnen Inputfelder als Model den gemeinsamen *WmsUserSettings* State verwenden. Im folgenden Code ist ersichtlich, wie das Select-Field als Model das Property *outCrs* verwendet. Das Property outCrs befindet sich jedoch nicht im lokalen State, sondern wird mit einem Getter direkt vom WmsUserSettings Store gelesen. Wird der Wert des Selects geändert, wird der Setter aufgerufen, welcher eine Mutation auf dem gemeinsamen Store ausführt.

```
    <select class="form-control form-control-sm" id="crsselect" v-model="outCrs" >
      <option selected disabled value="null">Select output CRS...</option>
      <option  v-for="crs in allCrs" v-bind:value="crs">EPSG:{{ crs }}</option>
    </select>
```

Listing 9.2: Komponente OutputCrsSelect (Template-Teil)

```
  export default {
    name: 'outputcrsselect',
    data() {
      return {
      };
    },
    computed: {
      ...mapState({
```

```
9           allCrs: state => state.WmsUserSettings.item.layer.crs,
10        }),
11        outCrs: {
12          get() {
13            if (this.$store.state.WmsUserSettings.item.out_crs) {
14              return this.$store.state.WmsUserSettings.item.out_crs;
15            }
16            return null;
17          },
18          set(crs) {
19            this.$store.commit('WmsUserSettings/updateOutCrs', Number(crs));
20          },
21        },
22      },
23    };
```

Listing 9.3: Komponente OutputCrsSelect (JavaScript-Teil)

9.1.4 MapView

Mit Leaflet und Leaflet.draw wurde die Kartenansicht implementiert. Die Karte wird in jedem Anwendungsfall des GeoConverters angezeigt. Jedoch wird sie nur in Verbindung mit einer WMS URL auch wirklich benötigt.

Damit besser ersichtlich ist, wo man sich auf der Welt befindet, wird ein BaseLayer eingebunden. Dieser basiert auf OpenStreetMap Daten. Als Standardansicht 9.3 wird der Standort der HSR angezeigt, um auf den Entwicklungsort des GeoConverters hinzuweisen.

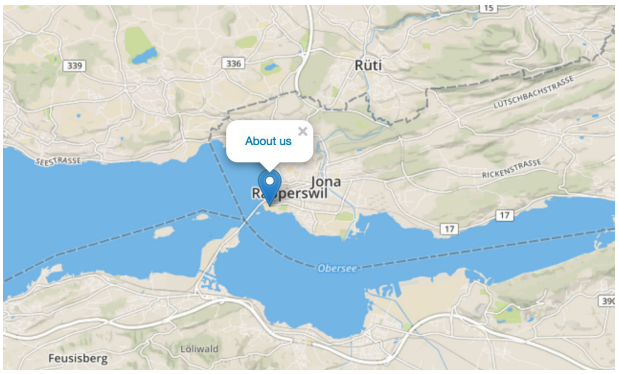


Abbildung 9.3: Baselayer mit HSR Standortmarkierung

Wenn eine WMS URL in den GeoConverter eingefügt wird, wird der HSR Standort auf der Karte entfernt und ein Layer des WMS eingeblendet. Der WMS Layer wird transparent gesetzt, falls er dies unterstützt und über den Baselayer gelegt.

Abbildung 9.4: Baselayer mit überlagertem Veloweg-Layer

9.2 REST API

Die Kommunikation zwischen dem Frontend und Backend wurde mit einer REST-Schnittstelle realisiert. Über definierte Methoden kann das Frontend dem Backend Daten übertragen und die Resultate beziehen. Dieser Abschnitt widmet sich der Umsetzung der von uns entwickelten API.

Die API für den GeoConverter wird gemäss den Spezifikationen und Anforderungen in verschiedene Teilbereiche unterteilt.

- **Job** [REST: /api/job/] Jobsteuerung für die Ausführung der Benutzeraufträge
- **URL** [REST: /api/url/] Erfassung einer WxS-URL um die passende Funktion zu ermitteln
- **WFS** [REST: /api/wfs/] Zugriff auf die WFS-Aufträge
- **WMS** [REST: /api/wms/] Zugriff auf die WMS-Aufträge
- **Raster** [REST: /api/raster/] Rasterspezifische Funktionen
- **Vector** [REST: /api/vector/] Vektorspezifische Funktionen
- **Util** [REST: /api/util/] Unterstützende Funktionen wir Koordinatentransformationen

9.2.1 Job

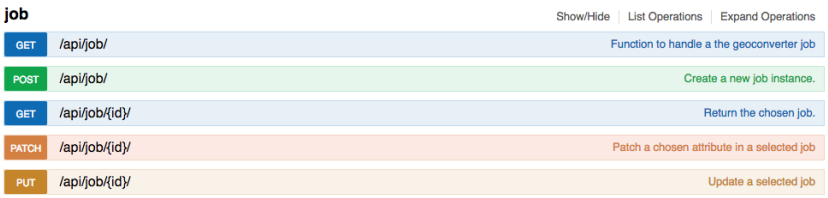


Abbildung 9.5: API Übersicht Job

Die Job-API dient zur Registrierung von neuen Aufträgen. Alle Bezugs- und Konvertierungsaufträge werden an einen Job gekettet. Sobald alle Aufträge erfasst wurden, können diese über die Job-API gestartet werden. Die Resultate werden innerhalb des Jobs zusammengefasst angeboten.

Job erfassen

Um neue Aufträge zu registrieren, muss zuerst ein neuer Job angefordert werden. Über die API wird ein neuer, leerer Job erstellt. Das geschieht über einen leeren «POST». Dadurch wird in der Datenbank ein neuer Job gespeichert und die nötige ID wird zu Verfügung gestellt.

```
{
  "id": 1,
  "status": "init",
  "message": null,
  "output": null,
  "wxs_url": [],
  "raster_file": [],
  "vector_file": [],
  "wfs_user_input": [],
  "wms_user_input": []
}
```

Listing 9.4: JSON eines neuen Job's

Job starten

Um einen Job und die dazugehörigen Aufträge zu starten, wird der Status über die Job-API auf «ready» gesetzt. Dies geschieht mit einer «PATCH»-Anfrage vom Frontend.

```
{
  "id": 1,
  "status": "ready",
  "message": null,
  "output": null,
  "wxs_url": [ 1 ],
  "raster_file": [],
  "vector_file": [],
  "wfs_user_input": [],
  "wms_user_input": [ 1 ]
}
```

Listing 9.5: JSON Job starten

Job Resultate beziehen

Sobald alle Aufträge innerhalb eines Jobs durch das Backend beendet wurden, werden alle Resultate zusammengefasst im Job dargeboten. Das Frontend kann nun die Daten über ein «GET» beziehen. Der Output ist eine gültige URL für den Datenbezug.

```
{
  "id": 1,
  "status": "finished",
  "message": null,
  "output": "2017_06_01_08-02-23.zip",
  "wxs_url": [ 1 ],
  "raster_file": [ 1 ],
  "vector_file": [],
  "wfs_user_input": [],
  "wms_user_input": [ 1 ]
}
```

Listing 9.6: JSON beendeter Job

9.2.2 URL

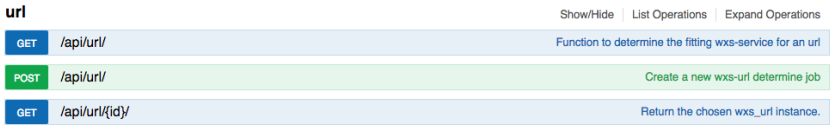


Abbildung 9.6: API Übersicht URL

URL erfassen

Um einen Webservice zu erfassen, muss dessen URL an einem Job gespeichert werden. Dazu wird ein existierender Job benötigt. Im Backend wird dann ermittelt, um welchen Service es sich handelt. Sobald die Anfrage beendet wurde, wird der Status aktualisiert. Das Frontend kann über diesen Datensatz den zugehörigen Service ausfindig machen, oder wird informiert, dass kein passender Service gefunden wurde.

```
{
  "id": 1,
  "request_url": "http://wms.zh.ch/VelonetzZHWMS,
  "linked_job": 1,
  "wms": null,
  "wfs": null,
  "status": "init",
  "message": null
}
```

Listing 9.7: JSON URL erfassen

9.2.3 WFS

Unter WFS befinden sich die erfassten WFS-Webadressen und gewählten Benutzereinstellungen. Mit den Benutzerdaten werden Anfragen an die WFS-Server gesendet und die Vektordaten konvertiert.

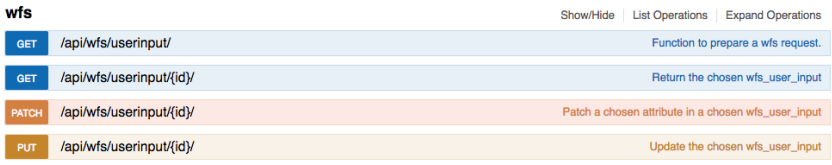


Abbildung 9.7: API Übersicht WFS

Benutzereingaben

Zur Übersicht und Bearbeitung der Benutzerdaten wird der Bereich Userinput bereitgestellt. Das Frontend kann über einfache «PATCH»-Anfragen die jeweiligen Parameter übergeben.

```
{
  "id": 1,
  "status": "init",
  "message": null,
  "data": null,
  "request_url": "http://maps.zh.ch/wfs/TBAWFS",
  "destination_format": "",
  "source_epsg": 4326,
  "target_epsg": 4326,
```

```
  "simplify": null,
  "advanced": null
}
```

Listing 9.8: JSON WFS Userinput

9.2.4 WMS

Unter WMS befinden sich die erfassten WMS-Webadressen und gewählten Benutzereinstellungen. Mit den Benutzerdaten werden Anfragen an die WMS-Server gesendet.

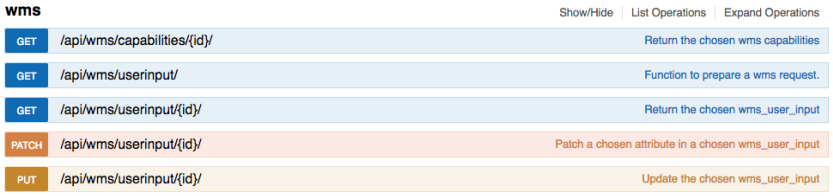


Abbildung 9.8: API Übersicht WMS

Benutzereingaben

Zur Übersicht und Bearbeitung der Benutzerdaten wird der Bereich Userinput bereitgestellt. Das Frontend kann über einfache «PATCH»-Anfragen die jeweiligen Parameter übergeben.

```
{
  "id": 1,
  "request_url": "http://wms.zh.ch/VelonetzZHWMS",
  "service": "WMS",
  "version": "1.3.0",
  "layer": {
    "name": "VelonetzZHWMS"
  },
  "bbox": {
    "type": "MultiPoint",
    "coordinates":
    [
      [
        680000,
        243000
      ],
      [
        696931,
        255698
      ]
    ]
  },
  "crs_bbox": 21781,
  "out_crs": 21781,
  "res_width": 800,
  "res_height": 600,
  "img_format": "image/tiff",
  "service_meta": 1
}
```

Listing 9.9: JSON WMS Userinput

9.2.5 Raster

Der API-Bereich «Raster» listet alle gespeicherten Rasterdateien zur Konvertierung auf.

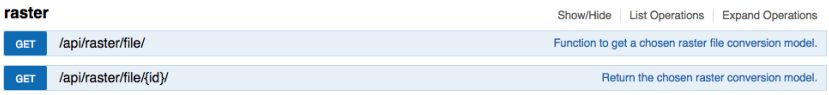


Abbildung 9.9: API Übersicht Raster

File

Auflistung aller Rasterdateien für die Konvertierung. Zu finden ist hier insbesondere das gewünschte Format und Koordinatensystem.

```
{
  "id": 1,
  "status": "finished",
  "message": null,
  "data": "raster_file_7hc83aD.tiff",
  "destination_format": "image/tiff",
  "source_epsg": 4326,
  "target_epsg": 4326
}
```

Listing 9.10: JSON beendeter Rasterauftrag

9.2.6 Vector

Der API-Bereich «Vector» listet alle gespeicherten Vektordateien zur Konvertierung auf.

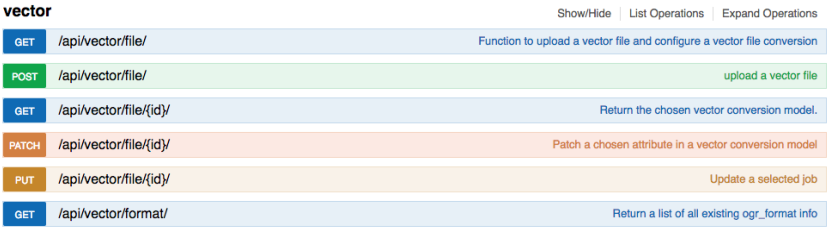


Abbildung 9.10: API Übersicht Vector

File

Auflistung aller Rasterdateien für die Konvertierung. Zu finden ist hier insbesondere das gewünschte Format und Koordinatensystem.

```
{
  "id": 1,
  "status": "finished",
  "message": null,
  "data": "/vector_file_xRfnL9D.dxf",
  "destination_format": "DXF",
  "source_epsg": 4326,
  "target_epsg": 4326,
  "simplify": null,
  "advanced": null
}
```

Listing 9.11: JSON beendeter Vektorauftrag

Format

Alle unterstützten Formate von Vektordaten werden unter diesem API-Aufruf aufgelistet. Die dazu nötigen Parameter werden in einem JSON-Objekt mitgeliefert.

```
{
  "id": 1,
  "name": "AutoCAD DXF",
  "ogr_name": "DXF",
  "file_extension": "dxf",
  "output_type": "file"
},
{
  "id": 2,
  "name": "Comma Separated Value (CSV)",
  "ogr_name": "CSV",
  "file_extension": "csv",
  "output_type": "file"
},
```

Listing 9.12: JSON unterstützte Ogrformate

9.2.7 Util

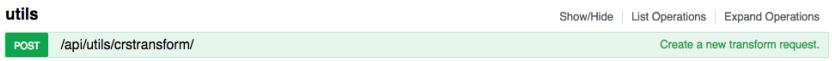


Abbildung 9.11: API Übersicht Utils

CRSTransform

CRSTransform ermöglicht das Transformieren von Bounding Boxes in ein anderes Koordinatensystem. Dazu wird über eine «POST»-Anfrage eine Boundig Box, das Ausgangskoordinatensystem und das Zielkoordinatensystem erfasst. Nach der Registrierung startet das Backend mit der Konvertierung und liefert die Resultate über den Datensatz aus.

```
{
  "id": 1,
  "bbox": {
    "type": "MultiPoint",
    "coordinates": [
      [
        680000,
        243000
      ],
      [
        696931,
        255698
      ]
    ]
  },
  "crs": 21781,
  "geo_bbox": {
    "type": "MultiPoint",
    "coordinates": [
      [
        8.497077527332271,
        47.33301034975646
      ],
      [
        8.723800515619033,
        47.444932218745066
      ]
```

```
28      ]
29      },
30    "bbox_out": {
31      "type": "MultiPoint",
32      "coordinates": [
33        [
34          8.49707752733227,
35          47.33301034975645
36        ],
37        [
38          8.723800515619029,
39          47.44493221874505
40        ]
41      ]
42    },
43    "crs_out": 4326
44  }
```

Listing 9.13: JSON CRSTransform

9.3 Django-Applikation

9.3.1 Projektstruktur

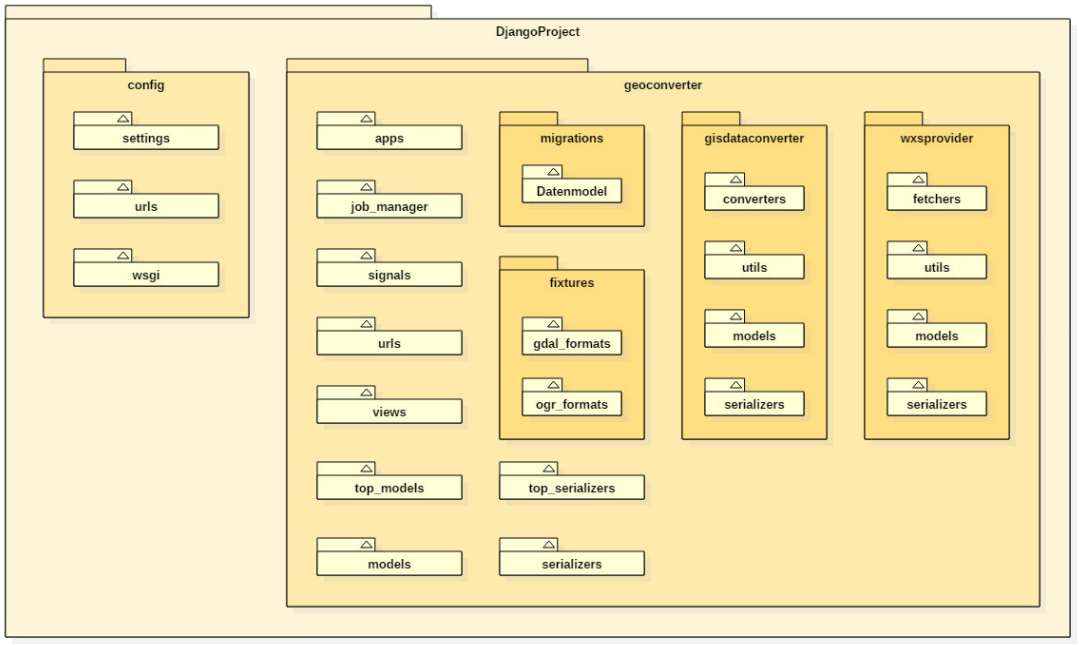


Abbildung 9.12: Projektstruktur in der Djangoapplikation

Die Djangoapplikation wurden im Wesentlichen in zwei Teilbereichen aufgebaut. Der Webkartendienstprovider ist für den Datenbezug aus den WMS und WFS zuständig. Für die Konvertierung der Geoinformationen wurde der GISdatenkonverter programmiert. Damit die einzelnen Aufträge gesammelt und ausgeführt werden, wurde die Jobmechanik umgesetzt. Die einzelnen Aufträge werden über Arbeiterthreads abgearbeitet. Gestartet werden diese über sogenannte Signale, welche während der Datengenerierung ausgelöst werden.

9.3.2 Datenmodel

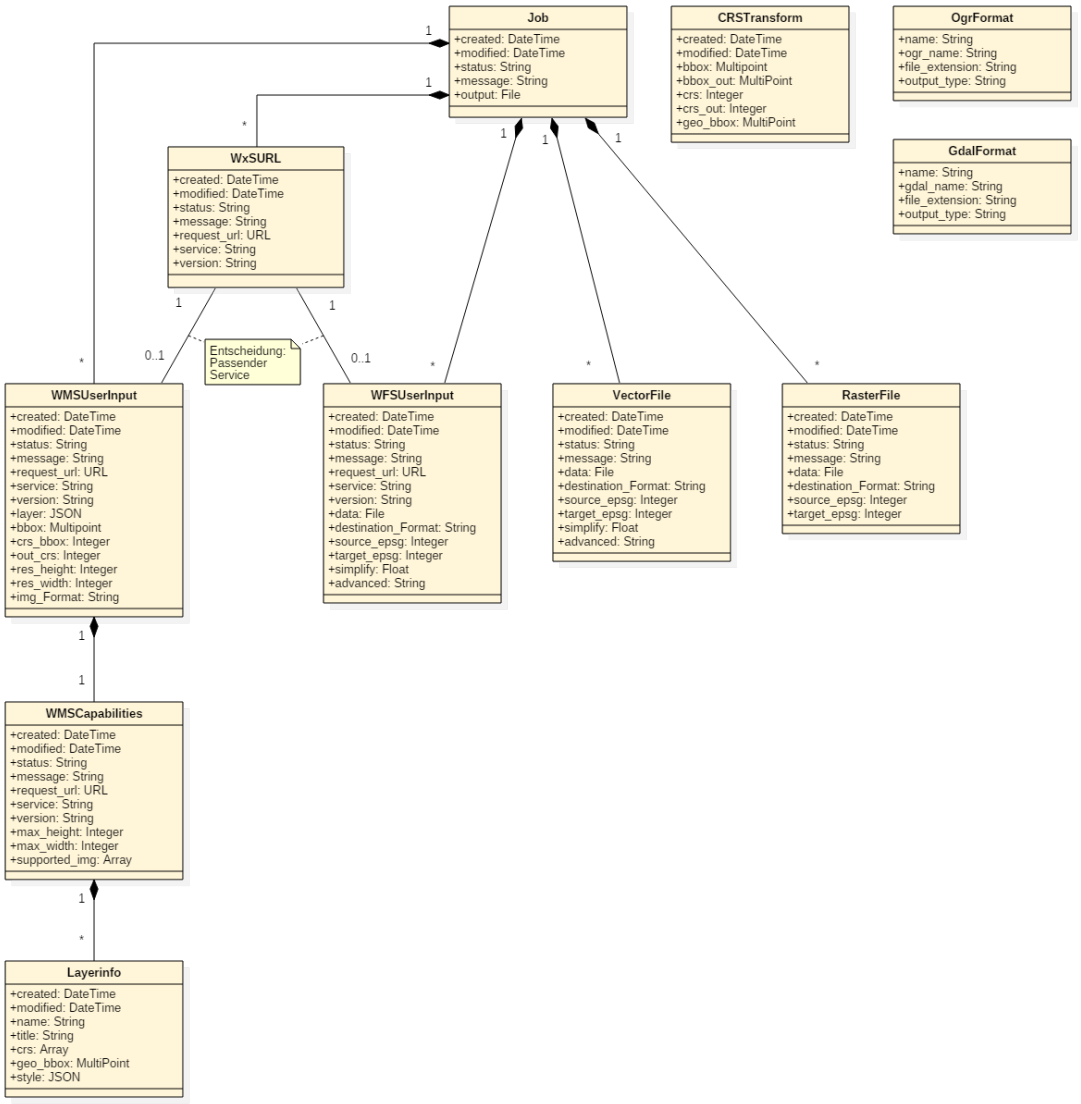


Abbildung 9.13: API Übersicht Raster

Job

Als zentraler Zugriffs- und Ausgangspunkt der Aufträge wird das Jobmodel genutzt. Jede Anfrage erzeugt im ersten Schritt einen neuen Job. Alle Aufträge werden dem Job angefügt. Sobald alle Einstellungen gegeben wurden und die Aufträge für die Ausführung bereit sind, wird der Job gestartet. Werden die Daten bereinigt, können diese über den Job gesucht und entfernt werden.

WxS-URL

Als Bindeglied zu den Geoinformation-Webservices wurde die WxS-URL erstellt. Mit diesem Model können die nicht zugeordneten URLs erfasst werden, bis ermittelt wurde, welcher Geoinformations-

Webservice der URL entspricht. Sobald der passende Service gefunden wurde, werden die entsprechenden Eingabenfelder erstellt und über die WxSURL zugänglich gemacht.

WMS-Userinput

Der WMS-Userinput-Datensatz bietet die nötigen Datenfelder für eine Anfrage an einen Webmapservice. Sobald eine URL einem WMS zugeordnet und das Datenmodell erstellt wurde, werden die Servicemetadaten (Capabilities) angefragt. Diese werden dem WMSUserInput-Modell angefügt und dienen zur Information der möglichen und nötigen Parameter für die erfolgreiche Datenanfrage an den Service. Allgemeine Servicemetadaten finden sich im WMS-Capabilities-Datensatz, Geoinformationen in Layerinfo-Datensätze. Die resultierende Ausgabeinformation wird in einem Raster-File-Model gespeichert.

WFS-Userinput

Das WFS-Userinput-Datensatz wird zur Erfassung der Webfeatureservices genutzt. Anders als die Anfrage über den WMS, benötigt die WFS-Funktion keine Metadaten. Zudem werden die Ausgabeinformationen direkt im Datensatz gespeichert, da der Bezug der Daten direkt im richtigen Format durchgeführt werden kann.

Vector-File

Alle zur Konvertierung bereitgestellten Vektordaten werden in einem Vector-File-Datensatz gespeichert. Zusammen mit den gewünschten Zielinformationen werden die Konvertierungsaufträge gesteuert und über dieses Modell ausgegeben.

Raster-File

Alle zur Konvertierung bereitgestellten Rasterdaten werden in einem Raster-File-Datensatz gespeichert. Zusammen mit den gewünschten Zielinformationen werden die Konvertierungsaufträge gesteuert und über dieses Modell ausgegeben.

CRS-Transform

Um Koordinaten in verschiedene Koordinatensysteme zu konvertieren, können innerhalb der Djangoapplikation Konvertierungsaufträge durchgeführt werden. Das CRSTransform-Model nimmt die Eingabedaten entgegen. Das Resultat findet sich nach der Konvertierung im selben Model.

Geodaten Formate

Die unterstützten Geoinformationsformate werden über die Geodaten Formate Modelle ausgegeben.

9.3.3 Auftragsabläufe

Innerhalb der Djangoapplikation werden die einzelnen Aufträge als Job zusammengefasst ausgeführt. Das Flussdiagramm zeigt den Ablauf für die Erfassung eines Job's und dessen Aufträge sowie die Abarbeitung der Teilschritte, auf.

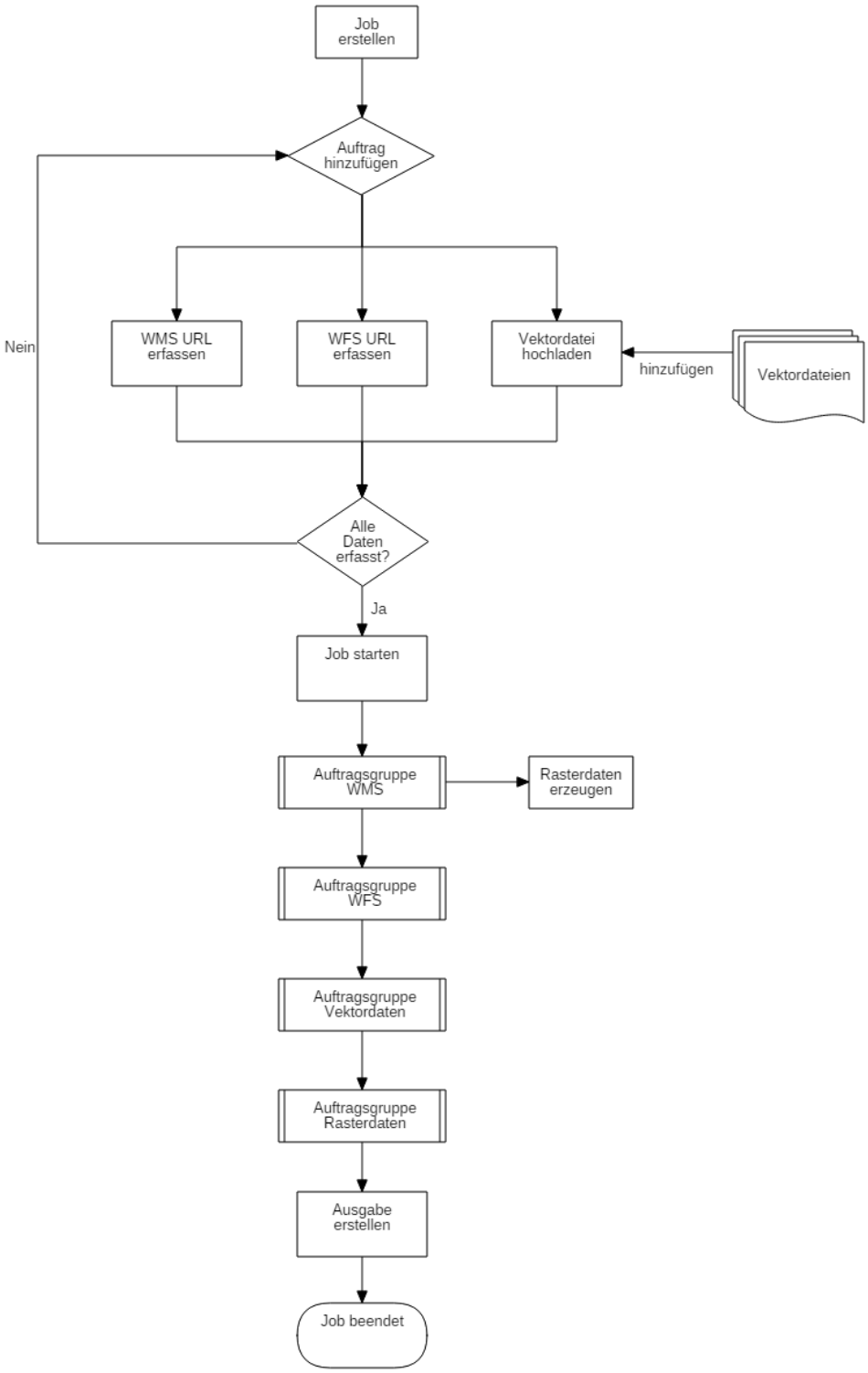


Abbildung 9.14: Flussdiagramm Ablauf Job's

Innerhalb der Auftragsgruppe werden alle erfassten Aufträge nacheinander durchgeführt.

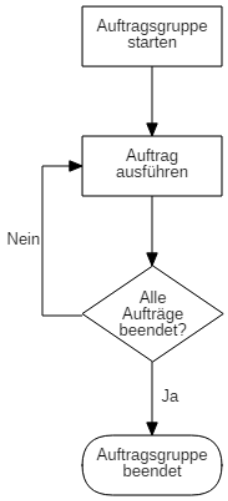


Abbildung 9.15: Flussdiagramm Ablauf Aufträge

9.3.4 Signale

Signale werden über das Redis-Plugin zur Verfügung gestellt. Über Signale können verschiedene Events registriert werden. Diese Events umfassen insbesondere die Änderung innerhalb der Datenbank, wenn zum Beispiel ein neuer Datensatz erstellt oder geändert wurde.

```
1 @receiver(post_save, sender=JobModel)
2 def clean_old_job_signal(sender, instance, created, **kwargs):
3     if created:
4         django_rq.enqueue(clean_all_old_job)
```

Listing 9.14: Signalfunktion Job's bereinigen

9.3.5 Job erstellen

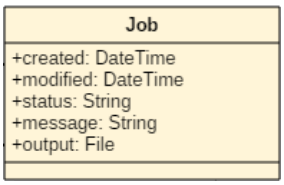


Abbildung 9.16: Job Datenmodell

Ein Job kann über das REST-Interface erstellt werden. Dazu werden keine Initialdaten benötigt. Beim Erstellen des Jobs wird die Erstellungszeit erfasst. Wichtig bei dieser Operation ist einzig die Job-ID für das Anfügen der Aufträge.

9.3.6 Job starten

Signal

Sobald alle Aufträge erfasst wurden, wird über das Job-Model die Ausführung ausgelöst. Gestartet wird der Job sobald der Status auf «ready» gesetzt wurde. Die Funktion für die Ausführung

wird dann in die Warteschlange gestellt und ausgeführt sobald ein «Worker» bereitgestellt wurde.

```
@receiver(post_save, sender=JobModel)
def start_job_signal(sender, instance, **kwargs):
    if instance.status == 'ready':
        django_rq.enqueue(start_job, instance)
```

Listing 9.15: Signalfunktion Job starten

Aufträge ausführen

Über die Funktion «start-job» wird über die angefügten Aufträge iteriert und diese mit der jeweiligen Funktion ausgeführt. Sobald alle Aufträge bearbeitet wurden, werden die Ausgabedaten in einem Zip-Archiv gesammelt und als Ausgabedatei (output) dem Jobmodel angefügt.

```
def start_job(job):
    job.status = STATUS_PENDING
    job.save()
    # load all wms maps
    try:
        for wms in WMSUserInput.objects.filter(linked_job=job):
            execute_wms_download(wms)
        # convert all wfs url files
        for wfs in WFSUserInput.objects.filter(linked_job=job):
            execute_wfs_url_conversation(wfs)
        # convert all vector files
        for vector in VectorFile.objects.filter(linked_job=job):
            execute_vector_conversation(vector)
        # convert all raster files
        for raster in RasterFile.objects.filter(linked_job=job):
            execute_raster_conversation(raster)
        # finalize output
        generate_job_archive(job)
        job.status = STATUS_FINISHED
    except Exception as e:
        job.status = STATUS_ERROR
        job.message = str(e)
    finally:
        job.save()
        return job
```

Listing 9.16: Funktion Job's ausführen

Damit ein fehlgeschlagener Auftrag nicht zum Abbruch des gesamten Ausführungsprozesses führt, wurde eine Wrapper-Funktion erstellt. So können alle fehlgeschlagenen Aufträge abgefangen und der Status ergänzt werden. Hier als Beispiel die Funktion «execute-wms-download».

```
def execute_wms_download(wms):
    try:
        fetchers.fetch_wms_data(wms)
    except Exception as e:
        wms.status = STATUS_ERROR
        wms.message = str(e)
    finally:
        wms.save()
```

Listing 9.17: Starte WMS download

Die Funktion «execute-wms-download» startet den Bezug der Rasterkarte mit den gespeicherten Eingabedaten. Damit allfällige Fehler den Job nicht unterbrechen, werden diese abgefangen und die Fehlermeldung gespeichert.

9.3.7 Rasterkarten downloaden

WMS Download durchführen

Im ersten Schritt werden alle Parameter aus den WMS-Daten extrahiert und für den Bezug vorbereitet. Alle Parameter werden als Map zugänglich gemacht. Für den erfolgreichen Bezug wird

einen verfügbaren Layer, eine Bounding Box, das dazugehörige Koordinatensystem, die Auflösung und das Format der Rasterkarte benötigt. Sofern die Anfrage an den WMS erfolgreich war, wird die resultierende Karte innerhalb eines neuen RasterFiles gespeichert. Schlussendlich wird der Status aktualisiert und alle Änderungen gespeichert.

Parameter Extrahieren

Damit der WMS Download erfolgreich durchgeführt werden kann, müssen alle nötigen Parameter in der richtigen Form vorhanden sein. Während der Parameterextraktion werden die Parameter in die gewünschte Form gebracht und als Map zurückgegeben. Konvertierte Parameter:

Bounding Box Die gewünschte Bounding Box (BBox) wird aus der gespeicherten Bounding Box und dem Ausgangskoordinatensystem erzeugt. Dazu wird ein neuer Multipoint erstellt und dieser zum Ausgangsresultat transformiert.

Koordinatensystem Das Koordinatesystem wird für die Anfrage richtig formatiert.

Auflösung Die Auflösung wird als Tupel benötigt

Bildformat Anfrage Das passendste Bildformat wird ermittelt.

```
1  def get_wms_params(map_request):
2
3      supported_img = map_request.service_meta.supported_img
4      bbox = MultiPoint(
5          map_request.bbox[0],
6          map_request.bbox[1],
7          srid=map_request.crs_bbox
8      ).transform(map_request.out_crs, clone=True)
9      params = dict()
10     params[WMS_PARAM_URL] = map_request.request_url
11     params[WMS_PARAM_VERSION] = map_request.version
12     params[WMS_PARAM_LAYER] = map_request.layer.get(WMS_LAYER_NAME, EMPTY_STRING)
13     params[WMS_PARAM_STYLES] = []
14     params[WMS_PARAM_BBOX] = [bbox[0][0], bbox[0][1], bbox[1][0], bbox[1][1]]
15     params[WMS_PARAM_CRS] = '{}:{}'.format(CRS_EPSG_NAME, str(map_request.out_crs))
16     params[WMS_PARAM_SIZE] = (
17             map_request.res_width,
18             map_request.res_height
19         )
20     params[WMS_PARAM_FORMAT_OUTPUT] = choose_best_case_img_format(
21         map_request.img_format,
22         supported_img
23     )
24     params[WMS_PARAM_FORMAT_REQUESTED] = map_request.img_format
25     return params
```

Listing 9.18: Extrahiere WMS Parameter

9.3.8 Vektordaten downloaden

ogr2ogr

Ogr2ogr ist die GDAL-Funktion für die Konvertierung von Vektordaten in ein anderes Datenformat. Ausgeführt wird diese Funktion über einen Shell-Command. Als Referenzwerk beziehen wir uns auf die offizielle GDAL-Dokumentation «Ogr2ogr Referenz», o.D. Innerhalb des GeoConverters wird diese Funktion für die Konvertierung von Vektordaten genutzt. Diese werden entweder über eine URL als Webserviceinformation oder per Vektordatei an Ogr2ogr übergeben.

WFS Daten beziehen und konvertieren

Anders als bei den Rasterdaten können die Vektordaten direkt von den Geoinformationsservern bezogen und konvertiert werden. Dies geschieht über das GDAL. Die Konvertierung wird über

den GDAL-Dienst «ogr2ogr» ausgeführt. Die Funktion «convert-wfs-data» erstellt dazu den passenden Shell-Command-Aufruf.
Folgende Parameter werden unterstützt:

path_input Pfad zur URL. In diesem Fall wird «WFS:» vorangestellt. Damit wird ein direkter WFS-URL-Aufruf signalisiert.

path_output Pfad zum Ausgabepunkt

destination_format Das gewünschte Format der Ausgabedatei

source_epsg Optionales Argument für das Eingangskoordinatensystem

target_epsg Optionales Argument für das Ausgangskoordinatensystem

simplify Optionales Argument für die Vereinfachung der Wegpunkte.

advanced Optionale Argumente, welche mit eigener Verantwortung hinzugefügt werden können.

```
def convert_wfs_data(wfs_info):
    wfs_info.status = STATUS_PENDING
    wfs_info.save()
    wfs_url = wfs_info.request_url
    suffix = get_ogr_file_extension(wfs_info.destination_format)
    temp_folder = tempfile.mkdtemp()
    try:
        args = get_ogr2ogr_args(path_input='{}{}'.format('WFS:', wfs_url),
                                path_output=path_combine(
                                    temp_folder,
                                    default_name_wfs,
                                    suffix
                                ),
                                destination_format=wfs_info.destination_format,
                                source_epsg=wfs_info.source_epsg,
                                target_epsg=wfs_info.target_epsg,
                                simplify=wfs_info.simplify,
                                advanced=wfs_info.advanced)

        output = utils.execute_shell(args)
        if output.get(STATUS_ERROR, None):
            wfs_info.message = 'Args:{}\nErrormsg:{}'.format(str(args), output[STATUS_ERROR])
        else:
            wfs_info.message = str(args)
        create_output_file(wfs_info.data, temp_folder)
        wfs_info.status = STATUS_FINISHED
    except Exception:
        wfs_info.status = STATUS_ERROR
    finally:
        wfs_info.save()
        shutil.rmtree(temp_folder)
```

Listing 9.19: Konvertiere WFS Daten von URL

9.3.9 Vektordaten Konversation

Allfällig hochgeladene Vektordaten werden zur passenden Auftragsgruppe zugeteilt.

Vorbereitung und Datenspeicherung

Wie beim Bezug der Vektordaten von einem Webserver, wird bei der Datenkonvertierung der GDAL-Dienst «ogr2ogr» aufgerufen. Der Unterschied besteht jedoch, dass die Eingangsdaten physisch als Datei übergeben werden.

9.3.10 Rasterdaten Konversation

Rasterdaten entstehen in der aktuellen Version des GeoConverters über den Bezug von Webservices. Sobald ein Rasterkarte geladen wurde, wird das Resultat zu einem Rasterdaten-Auftrag

hinzugefügt. Damit soll garantiert werden, dass das Ausgangsformat ein GeoTIFF ist.

Vorbereitung und Datenspeicherung

Die Konversation der Rasterkarten findet über GDAL statt. Die Funktion «convert-raster-data» bereitet die Konversation vor und speichert den Output in der Datenablage.
Vorgang:

1. Eingangsdaten laden
2. Suffix der Ausgangsdaten beziehen
3. Temporärer Ausgangspfad erstellen
4. Rasterdaten konvertieren
5. Ausgangsdaten speichern
6. Status aktualisieren
7. Temporärer Pfad entfernen

```
def convert_raster_data(raster_file):
    raster_file.status = STATUS_PENDING
    raster_file.save()
    src_file = raster_file.data
    suffix = utils.get_format_suffix(raster_file.destination_format)
    temp_folder = tempfile.mkdtemp()
    output_file = path_combine(temp_folder, default_name_raster, suffix)
    try:
        open(output_file, 'a').close()
        utils.translate_img(src_filename=src_file.path,
                            dst_filename=output_file,
                            dst_format=raster_file.destination_format
                            )
        raster_file.data.delete()
        create_output_file(raster_file.data, temp_folder)
        raster_file.status = STATUS_FINISHED
    except Exception:
        raster_file.status = STATUS_ERROR
    finally:
        raster_file.save()
        shutil.rmtree(temp_folder)
```

Listing 9.20: Konvertiere Rasterdaten

GDAL ansteuern

Die Konversationsfunktion steuert das GDAL an. Um Rasterdaten zu konvertieren wird der passende Treiber mit dem gewünschten Ausgangsformat geladen. Danach werden die Ausgangsdaten erstellt.

```
def translate_img(src_filename, dst_filename, dst_format):
    from osgeo import gdal
    src_ds = gdal.Open(src_filename)
    format = format_map.get(get_format_suffix(dst_format), 'GTiff')
    driver = gdal.GetDriverByName(format)
    driver.CreateCopy(dst_filename, src_ds, 0)
```

Listing 9.21: GDAL Aufruf

9.3.11 Ausgangsdaten

Nach der Beendigung der Auftragsgruppen werden alle Resultate in einem Archiv gesammelt und im Job gespeichert.

```
def generate_job_archive(job):
    t = tempfile.NamedTemporaryFile()
    job.output.save(get_archive_name(), File(t))
    target_zip = zipfile.ZipFile(job.output.path, 'w')
    try:
        for wfs in WFSUserInput.objects.filter(linked_job=job):
            try_write_to_zip(target_zip, wfs.data.path, wfs.data.name)
        for vector in VectorFile.objects.filter(linked_job=job):
            try_write_to_zip(target_zip, vector.data.path, vector.data.name)
        for raster in RasterFile.objects.filter(linked_job=job):
            try_write_to_zip(target_zip, raster.data.path, raster.data.name)
    finally:
        t.close()
        job.save()
        target_zip.close()
        return job
```

Listing 9.22: Generiere Archiv aller Auftragsresultate

9.4 Testing

9.4.1 Manuelle Tests

Um die Anfrage an die verschiedenen Server zu testen wurden manuelle Testanfragen gestartet. Folgende WMS Server wurden getestet:

- ArcGIS Mapserver
- GeoServer
- MapProxy
- QGIS
- UMN Mapserver

Als Testergebnis wurden folgende Kriterien bewertet:

- Server erreichbar
- Service identifizierbar
- Capabilities geladen
- Kartenansicht zeigt WMS-Daten

Ergebnisse WMS

Insgesamt wurden 227 Server getestet.

14 Stück waren nicht erreichbar

52 Capabilities-Anfragen waren nicht erfolgreich

52 GetMap-Anfragen waren nicht erfolgreich

Das heisst, 175 von 227 WMS-Server konnten mit dem GeoConverter angefragt werden.
In Prozent sind das 77,1%.

Natürlich ist das Resultat von der Internetverbindung und dem Serverzustand abhängig. Daher kann das Resultat variieren.

Ergebnisse WFS

Insgesamt wurden 201 Server getestet.

15 Stück waren nicht erreichbar

159 Stück konnten nicht als WFS identifiziert werden

Das heisst, 42 von 201 WFS-Server konnten mit dem GeoConverter angefragt werden.
In Prozent sind das 20.1%.

Natürlich ist das Resultat von der Internetverbindung und dem Serverzustand abhängig. Daher kann das Resultat variieren.

Fazit

Die Tests zeigen, dass der GeoConverter bei der Identifizierung der WFS entweder noch Schwierigkeiten hat, oder das Konzept der Serviceerkennung muss überarbeitet werden. Allenfalls sollte der Benutzer selbst entscheiden können, was für einen Service erwartet wird.

9.4.2 Backend

Pytest

Innerhalb der Djangoapplikation wurden statische Tests erstellt. Als Testframework wurde Pytest benutzt.

Syntax und Codestyle

Um die Syntax und Codestyle zu testen, wurde flake8 eingesetzt. Über das Makefile können alle Tests ausgeführt werden.

9.4.3 Frontend

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. Zu jeder Aktion wurde auch die erwartete Reaktion (in kursiv) angegeben.

Test Szenario WMS

1. Submit Button klicken ohne WMS URL — *Es wird eine Fehlermeldung angezeigt.*
2. WMS URL eingeben und submitten — *Die WMS Einstellungen werden geladen.*
3. Layer auswählen — *Der gewählte Layer wird auf der Karte angezeigt.*
4. In der Karte zoomen, ein Rechteck von Hand zeichnen und wieder löschen. Anschliessend nochmals Rechteck einzeichnen. — *Die Koordinaten der Bbox werden in den Einstellungen aktualisiert.*
5. Optional - Ab dem Zeitpunkt, an dem die manuelle Eingabe möglich war: Manuelle Eingabe der Koordinaten — *Die manuelle Eingabe wird auf die Karte gezeichnet.*
6. Output CRS wählen — *Das Ausgabe CRS wird im Store aktualisiert.*

7. Auflösung in etwa dem Rechteck Verhältnis entsprechend eingetragen — *Auflösung wird in Sotre gespeichert.*
8. Download anstossen (Submit) — *Auftragsstatus und Details werden angezeigt.*
9. GeoTIFF herunterladen — *GeoTIFF lässt sich öffnen und beinhaltet die gewählten Daten.*

Test Szenario WFS

1. WFS URL eingeben und submitten — *Die WFS Einstellungen werden geladen.*
2. Beliebiges Format auswählen — *Die Einstellung wird im Store gespeichert.*
3. Simplify Parameter angeben — *Die Einstellung wird im Store gespeichert.*
4. Konvertierung anstossen (Convert) — *Wartesymbol wird angezeigt.*
5. Zip herunterladen — *Zip beinhaltet das gewünschte Format. Datei lässt sich öffnen.*

Test Szenario File upload

1. File in Dropzone ziehen — *File wird in Dropzone mit Name angezeigt.*
2. Beliebiges Format auswählen — *Die Einstellung wird im Store gespeichert.*
3. Simplify Parameter angeben — *Die Einstellung wird im Store gespeichert.*
4. Konvertierung anstossen (Convert) — *Die Files werden aus der Dropzone hochgeladen und anschliessend entfernt. Wartesymbol wird angezeigt.*
5. Zip herunterladen — *Zip beinhaltet das gewünschte Format. Datei lässt sich öffnen.*

End to End Tests

Mit Nightwatch wurden zwei einfache End-to-End Tests erstellt. Da die Applikation sehr viele abhängige Elemente hat, war es sehr schwierig Tests zu schreiben, welche durch die ganze Applikation gehen.
Die End-to-End Tests wurden deshalb eher für UI Tests verwendet. So wird zum Beispiel überprüft, ob die Navigation funktioniert und die richtigen Seiten geladen werden.
Ausserdem wurde ein WMS-Test erstellt, welcher eine URL eingibt und testet, ob anschliessend die Einstellungen geladen werden.

Die Tests können mit folgendem Befehl gestartet werden.

```
npm run e2e
```

Wichtig: Für den WMS-Test muss das Frontend und Backend auf Port 8081 laufen.

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 GIS, Programmiersprachen und Frameworks konnte er das Projektteam aktiv unterstützen. In der zweiten Hälfte des Projektes übernahm Nicola Jordan aktiv die Betreuerrolle.

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 GeoConverters.

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 Backends des GeoConverters.

11.2 Prozessmodell

Im Allgemeinen richtet sich das Projekt nach dem RUP-Prinzip. Damit aber die einzelnen Phasen möglichst flexibel und auf neue Informationen gestaltet werden können, wird unser Projektablauf mit agiler Planung ergänzt.

11.2.1 RUP

Der Rational Unified Process, kurz RUP, ist ein Vorgehensmodell für die Softwareentwicklung. RUP wird in vier Phasen unterteilt, wobei jede Phase einer anderen Projektthematik gewidmet ist. Ziel dieser Phasen ist das Einhalten des zeitlichen Rahmens. Wir haben uns für dieses Vorgehensmodell entschieden, da die Bachelorarbeit an einem festgesetzten Zeitpunkt endet.

Projektplanung

Die Projektplanung für den GeoConverter wird durch die verschiedenen RUP-Phasen aufgeteilt. Aufgrund der Teilbereiche innerhalb des Projektes, definierte das Projektteam zwei Hauptphasen, welche einzeln in die RUP-Phasen aufgeteilt wurden. Die erste Hauptphase dient zur Vorbereitung des Projektes und soll den Rasterkarten-Bereich als Resultat liefern. In der zweiten Hauptphase soll der Vektorbereich hinzugefügt werden und der GeoConverter im ganzen als Resultat geliefert werden.

Inception

Die Inception wird in diesem Projekt sehr kurz gehalten. Sie dient zur Einrichtung der Arbeitsumgebung und des Projektmanagements.

11.2.2 Elaboration RasterGeoConverter

Die Elaboration des RasterGeoConverters soll einen Durchstich der verschiedenen Layers als Resultat liefern. Aufgrund vorangegangener Analysen, konnten wir uns direkt um die Lösungserarbeitung kümmern. Dieses Vorgehen lieferte uns schnelle und aussagekräftige Resultate. So konnten wir die Risiken minimieren und klar bezeugen, dass der erfolgreiche Ausgang des Projektes erreichbar ist.

Construction RasterGeoConverter

Die Constructionphase soll eine lauffähige Applikation mit dem umgesetzten RasterGeoConverter als Resultat liefern.

Transition/Reserve

Die Transitionphase soll uns als Reserve für unvorhergesehene Ereignisse dienen. Zudem findet während dieser Zeit die Zwischenpräsentation statt, welche dem Betreuer und Gegenleser unseren bisherigen Arbeitsfortschritt nachweisen soll.

Elaboration GeoConverter

Die zweite Hauptphase beginnt erneut mit einer Elaboration. Hier sollen alle nötigen Änderungen, Erweiterungen und Risiken ermittelt werden, um den Vektorbereich in die bestehende Applikation hinzuzufügen.

Construction GeoConverter

In der Constructionphase des GeoConverters wird die Zielapplikation implementiert.

Transition

Die Transitionphase dient zur Beendigung des Projektes. Eine saubere Abgabe von Code und Dokumentation soll mit dieser Phase garantiert werden.

11.2.3 Agiler Teil

Obwohl RUP nach dem Wasserfall-Prinzip aufgebaut ist, also nach definierten Phasen und Termine, erweitern wir das Vorgehensmodell mit agilen Elementen.

Standup Meeting

Jeder Arbeitstag beginnt mit dem Standup Meeting. Inhalt des Meetings ist:

- Was wurde erledigt
- Rahmenpunkte oder Issues
- Tagesziele

Wir haben uns auf eine Gesamtdauer von 15 Minuten pro Meeting geeinigt.

Construction sprints

Die einzelnen Funktionen sollen über Sprints umgesetzt werden.

Reviews

Neue Features oder Funktionen werden nach einem Review in den Develop-Branch gemerged.

11.3 Infrastruktur

11.3.1 Versionierung

Die Projektversionierung findet mit der Versionskontrolle «Git» statt. Git ist ein Open Source-Versionskontrollsystem, kurz VCS. Eine genauere Umschreibung von Git findet sich im Artikel von Denker und Srecec (2015).

11.3.2 GitFlow

Für eine übersichtliche Zusammenarbeit und Versionierung innerhalb des Projekts orientiert sich das Projektteam nach dem «GitFlow»-Prinzip. GitFlow nutzt die vorhandene Git-Mechanik und gibt einen klaren Versionierungsprozess vor, welcher von vielen Entwicklungs- und Versionierungswerkzeugen unterstützt wird. Das Projektteam richtet sich nach dem Artikel von Driessen, 2010 und der Kurzfassung im Artikel von Kummer, o.D.

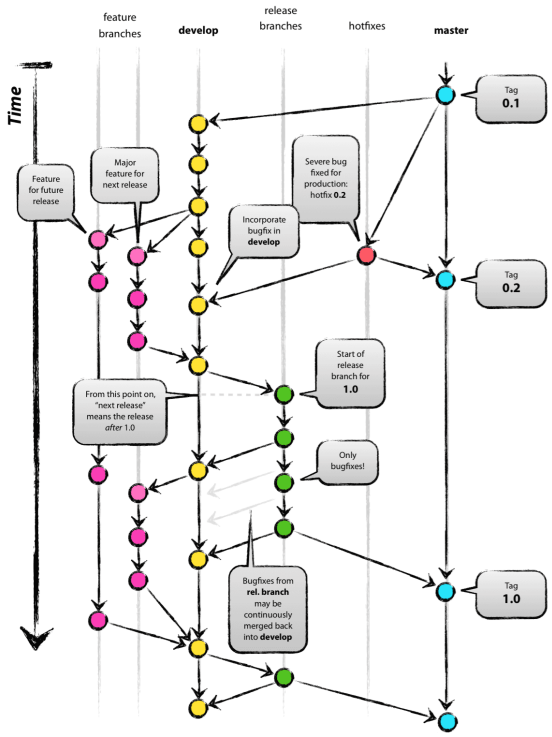


Abbildung 11.1: GitFlow Modell [Quelle: Artikel Driessen, 2010]

Im Git-Flow wird zwischen folgenden Zuständen unterschieden:

- Master
- Develop
- Feature
- Release
- Hotfix

Im *Master*-Zweig werden die Produktreleases versioniert. Diese Versionen sind entweder Einstiegspunkte innerhalb eines Projektes oder werden in der Produktion eingesetzt. Hingegen ist der *Develop*-Zweig der Ausgangspunkt während der Entwicklung. Es ist die gemeinsame Basis um einzelne *Features* hinzuzufügen und parallel zu arbeiten. Innerhalb eines *Features* können die einzelnen Entwicklungsschritte durchgeführt werden. Jedes Projektmitglied erstellt für einen Entwicklungsschritt einen neuen Featurezweig. Es ist auch möglich, während des Entwicklungsprozesses *Features* zu kombinieren. Sobald ein Feature fertiggestellt wurde, wird dieser zurück in den *Develop*-Branch eingefügt. Für einen nächsten *Release* wird ein neuer Zweig erstellt. Es werden fortlaufend Features hinzugefügt, bis der gewünschte Stand erstellt wurde. Während dem Release können Verbesserungen durchgeführt werden. Am Schluss, sobald alle Tests positiv und Anforderungen erfüllt sind, kann der Release in den *Master*-Zweig eingefügt werden.

Feature zu Develop

Das Feature wird im laufenden Sprint benötigt. Alle nötigen Funktionen und Anforderungen innerhalb eines neuen Features sind erfüllt. Die Tests sind alle positiv und decken alle neuen Funktionen komplett ab. Innerhalb eines UI wird dies über eine Funktionskontrolle garantiert. Es wurde ein Review durchgeführt.

Bugfixes

Sofern Fehler auftreten, werden diese innerhalb des aktiven Branches korrigiert. Die einzige Ausnahme ist der Masterzweig. Hier wird ein Hotfix erstellt.

Develop zu Release

Ein neuer Release wird innerhalb des Projektteams definiert. Er umfasst alle im Zeitraum definierten und erledigten Features.

Release zu Master

Alle definierten Funktionen sind vorhanden und wurden erfolgreich getestet. Ein Releasereview innerhalb des Projektteams wurde durchgeführt. Die Projektmitglieder haben gemeinsam zugestimmt.

Hotfix zu Master

Der Fehler wurde erfolgreich behoben und die Funktionen wurden getestet. Ein Review wurde durchgeführt und die Projektmitglieder haben gemeinsam zugestimmt.

Dokumentation

Alle Dokumente der Studienarbeit werden über Github in einem privaten Repository versioniert und gespeichert. Zugriff erhalten alle Beteiligten des Projektes.
https://github.com/celineellenanna/GeoConverterDoc.git

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 am Projekt Beteiligten.
https://github.com/celineellenanna/GeoConverter.git

11.3.3 Entwicklungswerkzeuge

Zweck	Tool	Version
IDE Backend	PyCharm	2017.1
IDE Frontend	Webstorm	2017.1
Versionierung	Git	2.9.0
Repository	Github	-
Versionierung GUI	GitKraken	2.6.0
Projektmanagement	Redmine	3.2.1
Dokumentation	LaTex	-

Tabelle 11.1: Entwicklungswerkzeuge im Einsatz

11.4 Projektplan

11.4.1 Aufwandschätzung

Projektdauer	16.5 Wochen
Zeit pro Mitarbeiter	ca. 21 Stunden pro Woche
Gesamtaufwand	720 Stunden
Projektstart	20. Februar 2017
Projektende	16. Juni 2017

Tabelle 11.2: Aufwandschätzung RasterGeoConverter

11.4.2 Phasen / Iteration

11.4.3 Meilensteine

M1 Architektur

Erreicht am: 27.03.2017
Folgende Ziele wurden erfüllt:

- Elabortationsprototyp durch alle Layers
- Mockups für das Frontend wurden erstellt.
- Projektstruktur aufgebaut
- Aufgabenstellung

Das Hauptziel dieses Meilensteins war das Definieren der Rahmenbedingung innerhalb des Projektes. Das Projektteam muss die Aufgabe verstanden haben. Zudem soll der Stand der Software jede Schicht ansprechen können. Dazu gehören das Frontend, das Backend und der Kartenservice. Innerhalb des Projektteams wurden die Zusammenarbeit, die Versionierung und der Informationsaustausch klar definiert.

M2 RasterGeoConverter

Erreicht am: 01.05.2017
Folgende Ziele wurden erfüllt:

- URL einlesen
- Inputfelder für erfolgreichen Aufruf anbieten
- Kartenansicht
- Kartenbezug von WMS
- Formattransformation
- Download

Der Meilenstein RasterGeoConverter konnte mit einer Verspätung von 14 Tagen erfüllt werden. Folgende Faktoren führten zu der Verspätung:

- Die Vorbereitungszeit der Zwischenpräsentation war grösser als geplant.
- Die Implementation des Vuex Management Pattern war aufwendiger als geplant.

- Das Datenmodell im Backend wurde überarbeitet.

Das Hauptziel des Meilensteins war der erfolgreiche Release eines funktionierenden RasterGeoConverter's. Über einen URL-Input sollte einen Service angefragt werden können. Die Metadaten werden dann im gleichen Schritt vom Server bezogen und wenn vorhanden dem Benutzer bereitgestellt. Alle nötigen Eingabefelder werden angezeigt und die Daten werden bei der Kartenanfrage übermittelt. Am Schluss soll die Karte geladen und als Download bereitgestellt werden. Der Abschluss des Meilensteins verzögerte sich aber um zwei Wochen.

M3 Prototyp GeoConverter

Erreicht am: 29.05.2017
Folgende Ziele wurden erfüllt:

- Vektordaten hochladen
- Vektordaten von Web laden
- Vektordaten konvertieren
- Jobsteuerung
- Refactoring nach Codereview

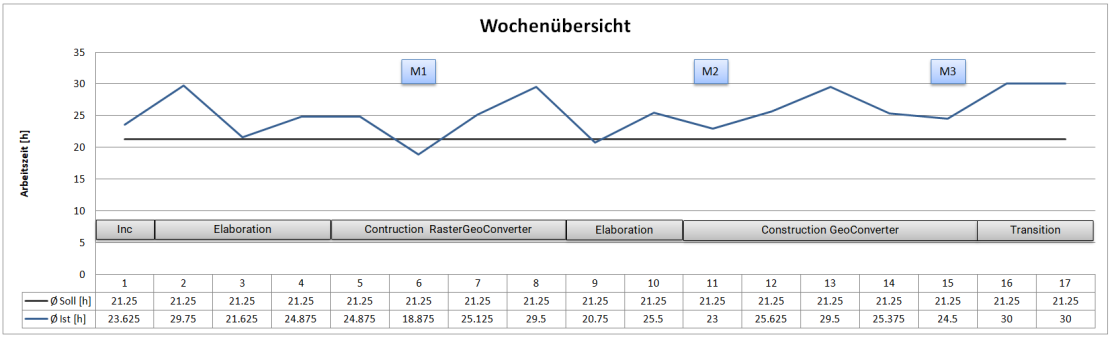
Der Meilenstein Prototyp GeoConverter konnte mit einer Verspätung von 14 Tage erfüllt werden. Folgende Faktoren führten zu der Verspätung:

- Durch das Aufwendige Refactoring der Architektur verzögerte sich der Fortschritt
- Die Architektur im Backend und Frontend wurde überarbeitet.
- Aktualisierung der REST-Schnittstelle

Das Hauptziel des Meilensteins war der erfolgreiche Release mit allen Funktionen des GeoConverters. Über einen URL-Input sollten alle Services, also WMS und WFS, angefragt werden können. Der innere Aufbau des GeoConverters soll dem Endprodukt entsprechen. Zudem sollen die gängigen WMS Services getestet werden können. Trotz der Verspätung konnten wir das Projekt erfolgreich abschliessen.

11.4.4 Auswertung

Wochenübersicht



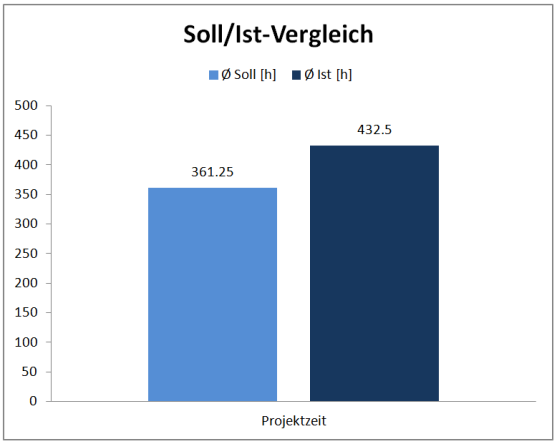
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Ø Soll [h]	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25	21.25
Ø Ist [h]	23.625	29.75	21.625	24.875	24.875	18.875	25.125	29.5	20.75	25.5	23	25.625	29.5	25.375	24.5	30	30

Abbildung 11.2: Stundendurchschnitt pro Person

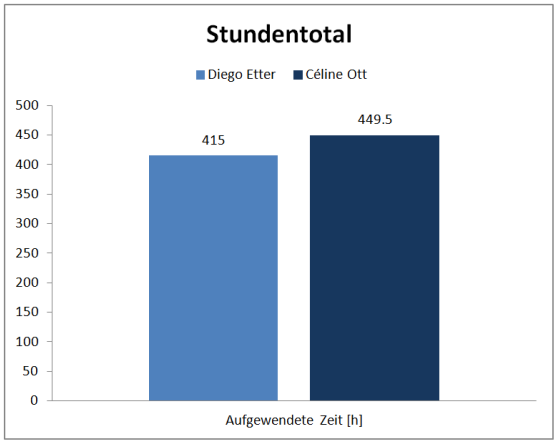
Die Wochenübersicht zeigt klar auf, dass der durchschnittliche Zeitaufwand für das GeoConverter Projekt höher war als vorgesehen. Ein höherer Zeitaufwand wurde vorwiegend während dem

Erreichen der Meilensteine betrieben. Bis zur Woche 6 wurde der RasterGeoConverter Prototyp erstellt. In der Woche 8 wurden die Vorbereitungen für die Zwischenpräsentation getroffen. Eine grosse Belastung fand zudem in der Woche 10 für die Fertigstellung des RasterGeoConverters statt. Nach dem Codereview wurde die Architektur überarbeitet und der GeoConverter implementiert. Dies ist zwischen Woche 11 bis 15 ersichtlich. Der grösste Aufwand bietet jedoch die Abschlussdokumentation.

Statistiken



(a) Soll/Ist-Vergleich Projektzeit



(b) Stundentotal Einzelperson

Wie schon in der Wochenübersicht abzusehen, war der durchschnittliche Zeitaufwand höher als geplant. Zwischen den Projektmitgliedern gab es ebenfalls grosse Zeitunterschiede. Diese lassen sich dadurch erklären, dass Céline Ott einen grösseren Lernaufwand im noch unbekannten Vue.js betreiben musste. Zudem unterzog sie das Frontend mehreren Refactorings aus Architekturgründen. Die Python und Django-Vorkenntnisse, welche Diego Etter aus der Studienarbeit ziehen konnte, erleichterten die Planung und Implementation des Backends erheblich.

11.5 Zeitplan

11.5.1 Phase RasterGeoConverter

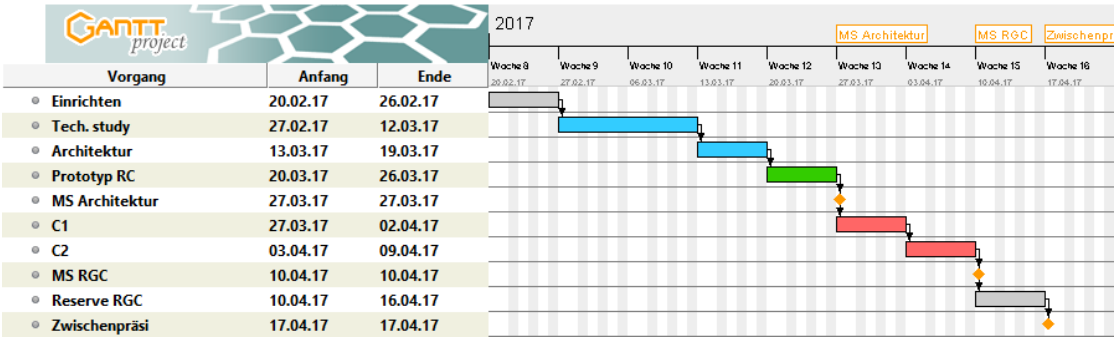


Abbildung 11.3: Phase RasterGeoConverter

Der Inhalt der ersten Phase ist die Umsetzung des RasterGeoConverters.

Einrichten

Mit dem Einrichten beginnt die Vorbereitung der Arbeitsumgebung. Das gesamte Projektmanagement soll in dieser Phase aufgebaut werden.

Vorstudie

Die eingesetzte Technologie soll effizient und ohne grösseres Risiko eingesetzt werden können. Dazu wurden zwei Wochen geplant. Das Projektteam möchte so auch ein grosses Risiko minimieren.

Architektur

Die Grundarchitektur des RasterGeoConverters wir innerhalb einer Woche geplant. Ein besonderes Augenmerk wurde auf die Kommunikation zwischen Front- und Backend gelegt.

Prototyp

Mit dem RasterGeoConverter Prototyp soll ein Durchstich bis zum WMS gelingen.

Implementation

Der RasterGeoConverter wird über zwei Wochen hinweg komplett implementiert. Ziel ist es, einen funktionierenden Pre-Release zu schaffen.

Reserve

Um allfällige Issues oder Verzögerungen abzufangen, haben wir einen Puffer im Phasenplan eingebaut.

Zwischenpräsentation

Das Finale der RasterGeoConverter-Phase ist die Zwischenpräsentation dem Betreuer und Gegenleser gegenüber.

11.5.2 Phase GeoConverter

Der finale GeoConverter soll in dieser Phase erstellt werden.

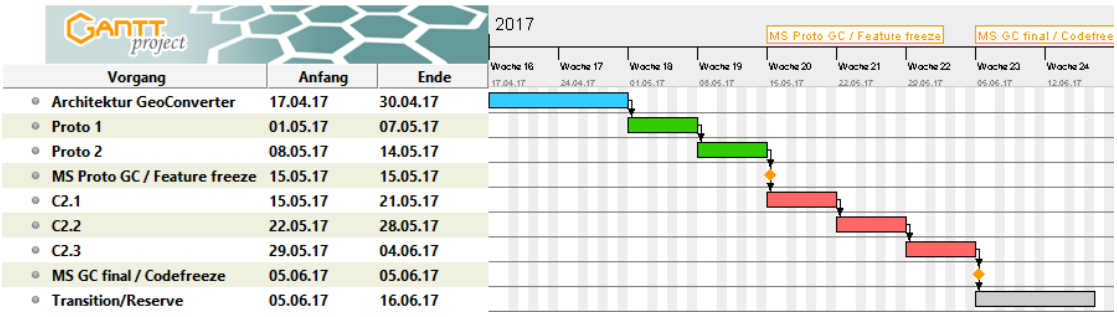


Abbildung 11.4: Phase GeoConverter

Planung GeoConverter

Damit die Vektordaten erfolgreich geladen und konvertiert werden können, wird die bestehende Architektur erweitert. Zudem kann diese Phase für Codereviews und Refactorings genutzt werden.

Prototypen

Die einzelnen Vektorfunktionen sollen über Prototypen umgesetzt werden. Ziel ist es, die Funktionen des bestehenden GeoConverters nacheinander zu rekonstruieren und zu testen.

Implementation

Über drei Wochen hinweg werden die Funktionen des GeoConverters implementiert. Das Ziel ist es, den finalen Release mit allen gesetzten Zielfunktionen zu erstellen.

Abgabe erstellen

Die Abgabe des GeoConverters wird über zwei Wochen hinweg vorbereitet. Dazu gehören die Installationsanleitung, die Dokumentation, alle Formalitäten seitens der HSR und natürlich die Abgabestruktur auf dem Übergabe Medium an den Betreuer. Die Abgabe wird mit der Projektvorstellung am 16.06.2017 finalisiert.

11.5.3 Projektablauf

Der Projektaublauf wurde über das Redmine erfasst und dokumentiert.

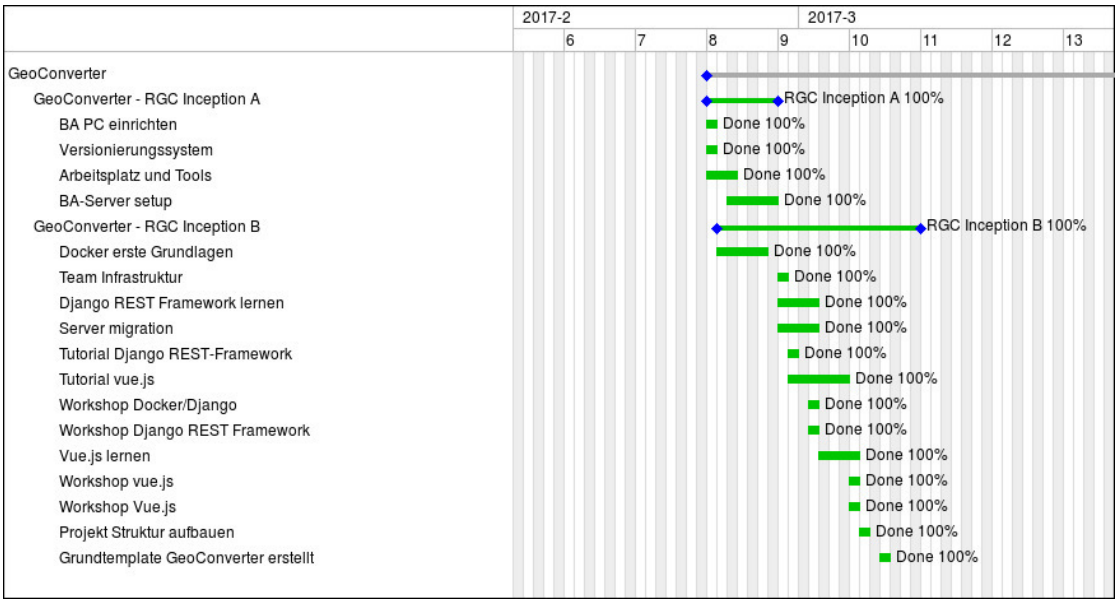


Abbildung 11.5: Vorbereitungsphase des Projekts

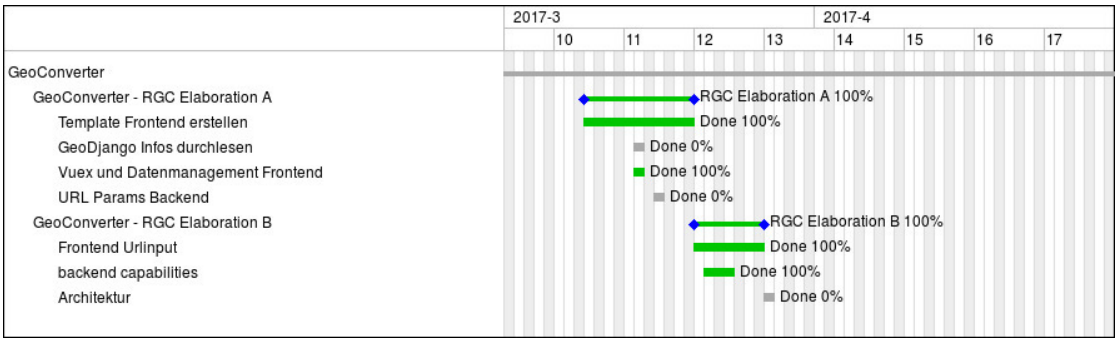


Abbildung 11.6: Architekturphase

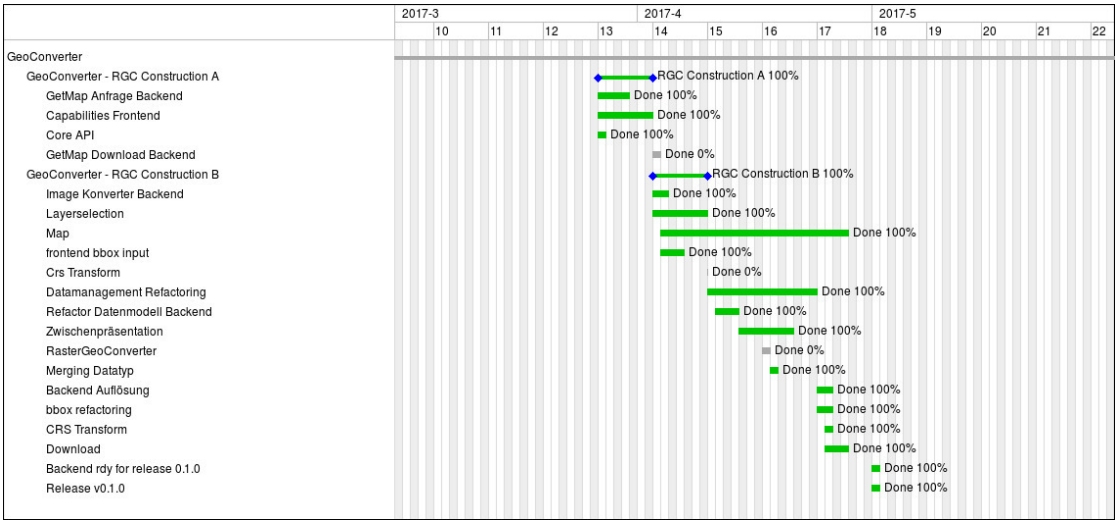


Abbildung 11.7: Implementierung RasterGeoConverter

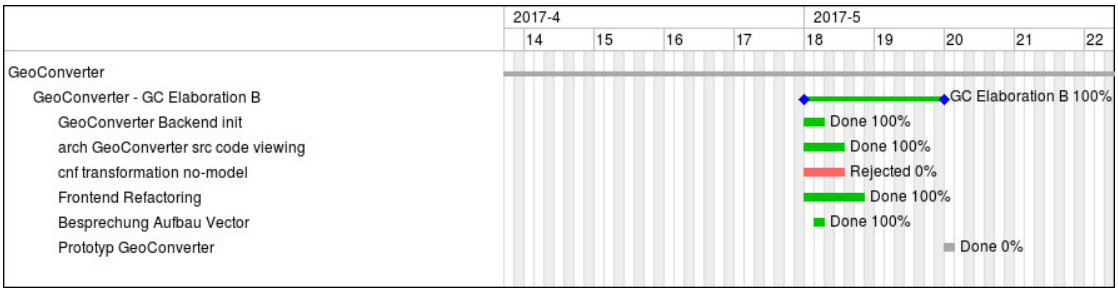


Abbildung 11.8: Vorbereitung GeoConverter

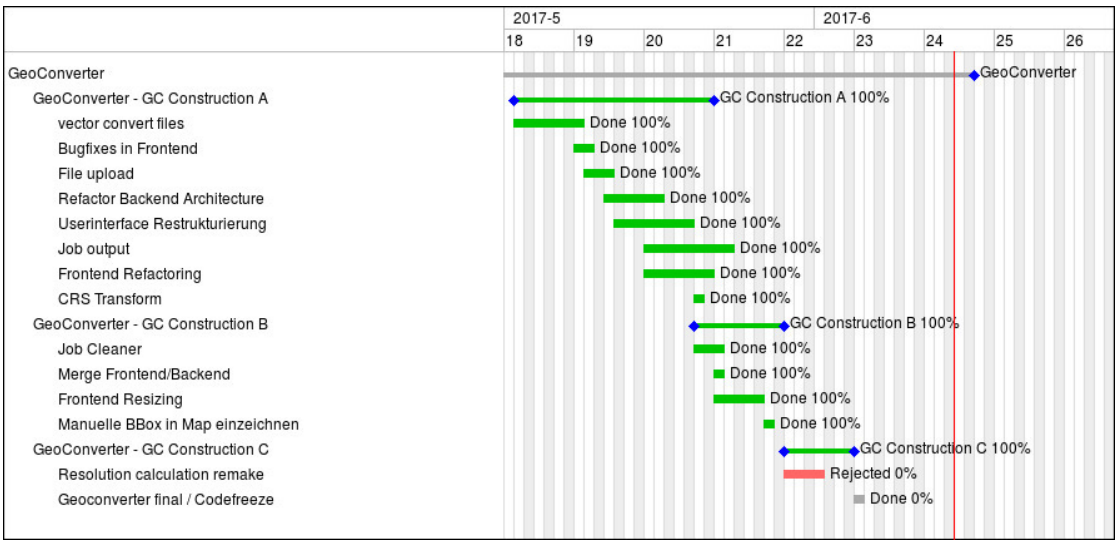


Abbildung 11.9: Implementierung GeoConverter

11.6 Risikoanalyse

11.6.1 Erläuterung

In jedem grösseren Projekt sind Risiken und Chancen vorhanden. Risiken sind uneinschätzbare Problemstellungen oder Situationen. Verzögerungen, Umplanungen oder sogar Projektabbrüche können mögliche Szenarien sein. 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.

11.6.2 Projekt

Die Risikoanalyse betrifft weniger die Applikation und die damit verbundenen Risiken, sondern umfasst die Bearbeitung der Bachelorarbeit und deren gelingen. So wurden vor allem die Szenarien gewählt und gewertet welche den Projektablauf negativ beeinflussen könnten. Mitglied bezogene Szenarien wurden bewusst ausgelassen. Beispielsweise der Ausfall durch Krankheit oder Projektabbruch einer Person. Wir haben uns ausschliesslich auf die technische Umsetzung und den damit verbundenen Aufwänden gewidmet.

Beide Werte zusammengerechnet ergeben dann einen gewichteten Schaden in Stunden. Die Zeit aller möglichen Schäden wurde im Zeitplan mit eingerechnet.

11.6.3 Risikotabelle

Parameter

Um ein Risiko zu bewerten und allfällige Kosten zu berücksichtigen musste ein passender Massstab geschaffen werden. In unserem Projekt ist unsere Hauptressource die Zeit, also beziehen sich die Risikokosten auf diesen Faktor.

Zwei Parameter wurden definiert:

Möglicher Schaden in Stunden

Eintrittswahrscheinlichkeit in Prozent

Nr	Titel	Beschreibung	max. Schaden [h]	Eintritt %	Gewichteter Schaden [h]	Vorbeugung	Verhalten beim Eintreten
R1	Vue.js lernen	Der Lernaufwand ist grösser als geplant	24	50%	12	- Workshop mit Nicola - Webtutorial - Kontinuierliche Weiterbildung	Der Zeitplan muss angepasst und die Issues im Verlauf des Projektes behoben werden
R2	Docker lernen	Der Lernaufwand ist grösser als geplant	24	50%	12	- Workshop mit Nicola - Webtutorial - Kontinuierliche Weiterbildung	Der Zeitplan muss angepasst und die Issues im Verlauf des Projektes behoben werden
R3	Converter kombinieren [Frontend]	- Funktionen können nicht direkt übernommen werden - Berechnungen im Frontend komplex	12	25%	3	- Aufbau des Converters schon in der Entwicklungsphase berücksichtigen	Der Zeitplan muss angepasst und die Issues im Verlauf des Projektes behoben werden
R4	Converter kombinieren [Backend]	Der RasterGeoConverter und der bestehende GeoCoverter lassen sich nur mit grossem Aufwand kombinieren	48	50%	24	'- Aufbau des Converters schon in der Entwicklungsphase berücksichtigen	Der Zeitplan muss angepasst und die Issues im Verlauf des Projektes behoben werden
R5	GDAL	GDAL lässt sich nicht oder nur mit grossem Aufwand integrieren	24	20%	4.8	- Über Docker direkt einbinden - passende Tests schreiben	Issue erfassen und mit Betreuer klären. Evt. Konvertierung entfernen.
R6	WxS Standard nicht vollständig umgesetzt. [Backend]	Die nötigen Schnittstellen werden nicht komplett oder mangelhaft angesprochen/implementiert	24	25%	6	Die Implementation soll die vom Standard definierten Parameter berücksichtigen	Pendenz abschätzen und allenfalls per Feature nachimplementieren. Evt. Weglassen und dokumentieren.
R7	Mapviewer WxS-Unterstützung	Der gewählte Mapviewer kann den WxS nicht ansprechen und keine Karte laden.	48	25%	12	- Abklären welche Technologien unterstützt werden. - Issues mit Betreuer besprechen	Der Service könnte exklusiv sein oder das Backend liefert das nötige Bildmaterial.

Abbildung 11.10: Implementierung GeoConverter

11.6.4 Risikomatrix

Achsen und Parameter

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.

Eintrittswahrscheinlichkeit, 1 - 5 Punkte

Schadensmass, 1 - 5 Punkte

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.

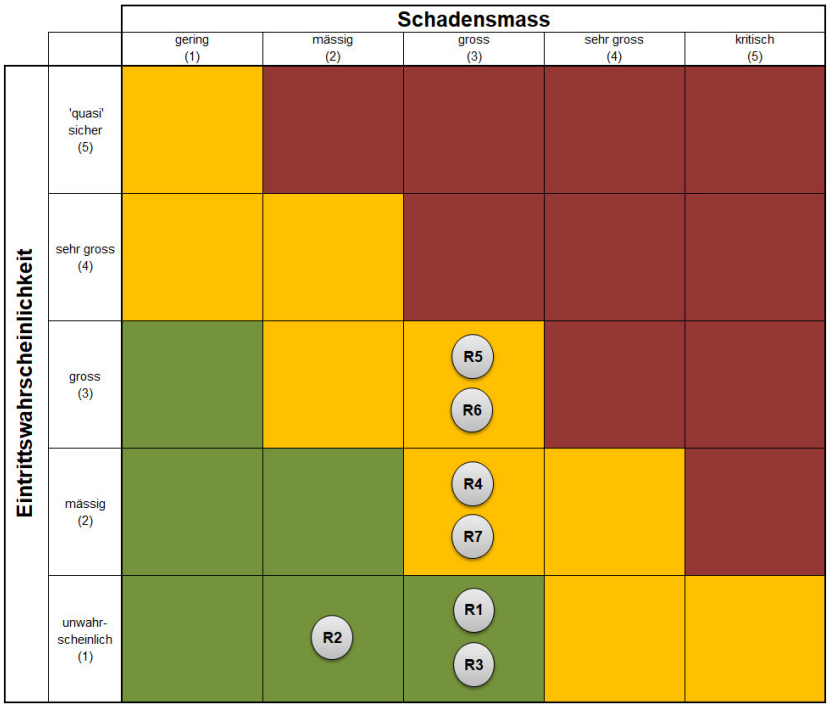


Abbildung 11.11: Implementierung GeoConverter

11.6.5 Fazit

Die Erfahrungen aus der Studienarbeit haben uns gelehrt, ein besonderes Augenmerk auf das Erlernen der eingesetzten Technologien zu haben. Daher identifizierten wir diese Punkte als grosses Risiko. Als Massnahme nahmen wir an einem Workshop teil, um uns mit Docker, Django und Vue.js auseinanderzusetzen. Natürlich konnten wir dadurch nicht allen Schwierigkeiten zuvorkommen. Dennoch, wir haben davon profitiert und lösten allfällige Knackpunkte schneller als vorher. Nicht zuletzt war dies elementar für das Erreichen unserer Projektziele. Für uns war die Risikoplanung vor allem in der Startphase sehr wichtig. Denn wir wollten Massnahmen treffen, welche es uns ermöglicht gar nicht erst ein Risiko aufkommen zu lassen.