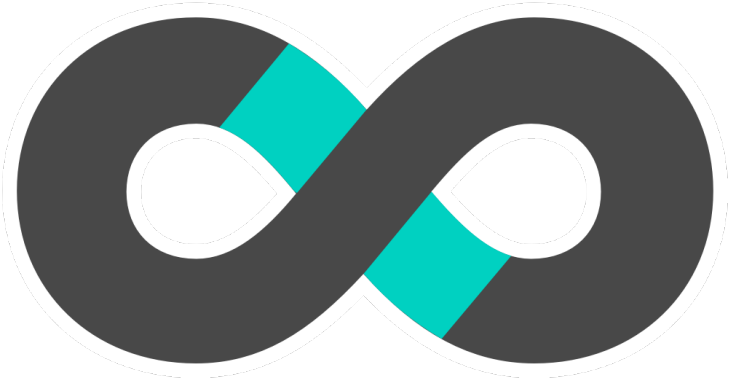
Apache Superset-Erweiterung
mit CSV-Importer und Geokodierung

Bachelorarbeit in Informatik

HSR Hochschule für Technik Rapperswil (FHO)
Herbstsemester 2019/2020

Autoren Michelle Kunz, Renato Venzin und Sascha Gschwind
Betreuer Prof. Stefan Keller
Experte Claude Eisenhut

Datum 8. Januar 2020



HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Abstract

Damit Unternehmen wettbewerbsfähig bleiben, werden Daten gesammelt und analysiert. Anschliessend werden die Resultate inklusive Visualisierungen publiziert. Dafür werden sogenannte Business-Intelligence-Tools (BI-Tools) eingesetzt. Visualisierungen können dazu verwendet werden, Zusammenhänge aufzudecken und zu verstehen. Zusätzlich dienen Visualisierungen als Basis für Entscheidungen. Apache Superset ist ein solches BI-Tool. Es ist eine der wenigen Applikationen dieser Kategorie, die Open Source ist. Das macht Superset zur kostengünstigen Alternative, vermindert vendor-lock-in und macht es erweiterbar.

Diese Arbeit erweitert die Software Apache Superset mit zwei Hauptzielen: Das erste Ziel ist, das Hochladen (upload) eigener Daten benutzerfreundlicher und moderner zu gestalten (sogenannter "CSV-Importer"). Das zweite Ziel ist die Erweiterung von Superset mit einer Geokodierungs-Funktion. Unter Geokodierung versteht man den Vorgang, aus Adressen Koordinaten zu gewinnen. Bisher mussten die Benutzer ihre Daten vorgängig über ein separates Tool selbst aufbereiten. Mit Hilfe dieser Funktion können Adressen direkt in Superset in Koordinaten transformiert werden. Damit können Benutzer nun noch mehr aus den Business-Daten holen, in dem sie zusätzlich Kartengrafiken erstellen und publizieren können.
Des Weiteren soll im Kontext dieser Arbeit Know-How über diese komplexe Software gesammelt und dokumentiert werden. Dazu gehören u. a. der Projektaufbau von Superset, die Konfiguration und die internen Strukturen. Das Know-How soll dem HSR-Institut für Software auch dazu dienen, eine eigene Instanz von Superset betreiben und erweitern zu können. Zusätzlich dienen Teile der Dokumentation dem Institut als Unterrichtsunterlagen für Weiterbildungskurse, um interessierten Personen Superset näher zu bringen.

Die oben erwähnten Ziele wurden erreicht. Sowohl der CSV-Importer als auch die Geokodierung wurden mit dem JavaScript-Framework React modernisiert. Die damit durchgeführte Pionierarbeit dient den Maintainern von Superset als Vorlage und wird ihnen helfen, weitere User-Interfaces (inkl. Eingabemasken) auf die neuesten technologischen Standards anzuheben.
Der neu designte und erweiterte CSV-Importer wirkt übersichtlicher und verständlicher. Nebst dem Hochladen der Daten in eine vordefinierte Datenbank, kann nun zwischen einer neuen SQLite- oder PostgreSQL-Datenbank ausgewählt werden. Diese neue Datenbank wird im Hintergrund mit den gegebenen Softwarebibliotheken (SQLAlchemy, Pandas) konfiguriert und erstellt.
Für die Geokodierung wurde eine eigene Eingabemaske implementiert, die im Backend auf externe Geokodierungs-Webservices von MapTiler und Google zugreift. Der Benutzer kann ein neues Koordinaten-Attribut anfügen oder ein bestehendes Attribut überschreiben lassen. Es ist auch möglich, dass nur diejenigen Adressen geokodiert werden, zu denen noch keine Koordinaten gefunden wurden. Der Vorgang der Geokodierung kann auch abgebrochen werden.
Die erstellten Anleitungen entsprechen der aktuellen Version und dem Setup von Superset und verbessern dessen bisher vernachlässigte Dokumentation.

Abstract

Companies collect, evaluate, visualize and publish data to stay competitive and create new (product) ideas. So-called Business-Intelligence Tools (BI Tools) are used for this. These visualizations can be used to uncover and understand relationships. In addition, visualizations serve as the basis basis of decisions. Apache Superset is one such BI tool. It is one of the few applications in this category that is open source. This makes Superset a cost-effective alternative, reduces vendor lock-in and makes it expandable.

This work extends the Apache Superset software with two main goals: The first goal is to make uploading your own data more user-friendly and modern (so-called “CSV importer“). The second goal is to extend Superset with a geocoding function. Geocoding is the process of obtaining coordinates from addresses. Previously, users had to prepare their data themselves beforehand using a separate tool. With the help of this function, addresses can be transformed into coordinates directly in Superset. This means that users can now get even more out of the business data by creating and publishing additional map graphics.
Furthermore, know-how about this complex software is to be collected and documented in the context of this work. It starts with the project structure of Superset, which goes through the configuration to the internal structures. The know-how should also serve the HSR Institute for Software to operate and expand its own instance of Superset. In addition, parts of the documentation serve the institute as teaching material for further training courses to bring Superset closer to interested people.

The goals mentioned above have been achieved. Both the CSV importer and the geocoding were modernized with the JavaScript framework React. The pioneering work carried out with this serves as a template for the maintainers of Superset and will help them to raise further user interfaces (including input masks) to the latest technological standards.
The newly designed and expanded CSV importer is clearer and more understandable. In addition to uploading the data to a predefined database, you can now choose between a new SQLite or PostgreSQL database. This new database is configured and created in the background with the given software libraries (SQLAlchemy, Pandas).
A separate input mask was implemented for geocoding, which accesses external geocoding web services from MapTiler and Google in the backend. The user can add a new coordinate attribute or have an existing attribute overwritten. It is also possible that only those addresses are geocoded that do not yet have coordinates. The geocoding process can also be canceled.
The instructions created correspond to the current version and setup of Superset and improve its previously neglected documentation.

Management Summary

Ausgangslage
Daten sind heutzutage eines der wichtigsten Instrumente jeglicher Unternehmungen. Sie können dazu verwendet werden ein besseres Verständnis über einen Sachverhalt zu gewinnen, sind eine wichtige Basis für Entscheidungen oder werden dazu genutzt, Prozesse zu optimieren. Doch Daten ohne Visualisierungen sind meist unverständlich und unbrauchbar.

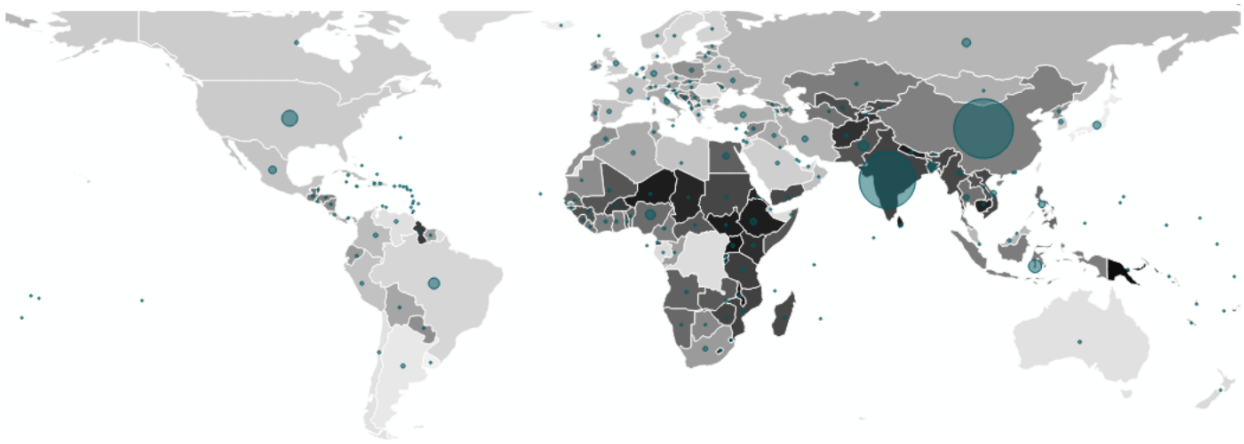


Abbildung 1: Einwohnerzahlen aller Länder als Ortsdiagrammkarte (Quelle: World Bank Data)

Hier kommen so genannte Business-Intelligence(BI)-Tools ins Spiel, die mit Hilfe moderner Technologien eine Visualisierung dieser Daten und gleichzeitig das Teilen dieser ermöglichen. So kann eine Diskussionsbasis geschaffen werden, mit Hilfe derer letztlich das Ziel erreicht werden kann.
Apache Superset ist eines der wenigen Open Source BI-Tools. Der Open Source Charakter von Superset macht es für Unternehmen sehr attraktiv, denn dadurch kann es flexibel und unabhängig bleiben. Obwohl der Funktionsumfang von Superset noch nicht mit der Konkurrenz mithalten kann, ist es bereits im Unternehmen einsetzbar. Allerdings gibt es ein signifikantes Verbesserungspotential. Viele der User-Interfaces sind veraltet und entsprechen nicht mehr den heutigen UI-Standards. Die Benutzerfreundlichkeit ist verbesserungswürdig.
Eine weitere wichtige Funktion vieler BI-Tools ist das Anreichern von Daten mit zusätzlichen Informationen. Bei geographischen Daten besitzen Unternehmungen oft nur Adressdaten, welche in der Art und Weise nicht für die Visualisierung auf einer Karte genutzt werden können. Hier kommen so genannte Geokodierungs-Dienste zum Einsatz, die basierend auf den Adressdaten Längen- und Breitengrade bestimmen. Diese Geokodierungs-Dienste reichern die Daten um zusätzliche Informationen an, welche dann vom Nutzer dazu verwendet werden können, die Adressdaten auf einer Karte zu visualisieren. Superset bietet in der aktuellen Version keine Datenanreicherung an, obwohl viele konkurrierende Produkte dies machen.

Ziele, Vorgehen und Technologien

Zu Beginn wurden ausgewählte konkurrenzierende BI-Tools analysiert und deren Funktionsumfang mit dem von Superset verglichen. Aufgrund deren Funktionalitäten haben wir die Erweiterungen für Superset gesammelt und priorisiert.

Im Kontext der Arbeit “Apache Superset Erweiterung mit CSV-Importer und Geokodierung 2019/2020“ wurden zwei Hauptziele definiert. Einerseits sollte die zentrale Funktion des CSV-Importers überarbeitet und verbessert, andererseits eine Geokodierung implementiert werden. Im ersten Teil wurde der CSV-Importer überarbeitet.

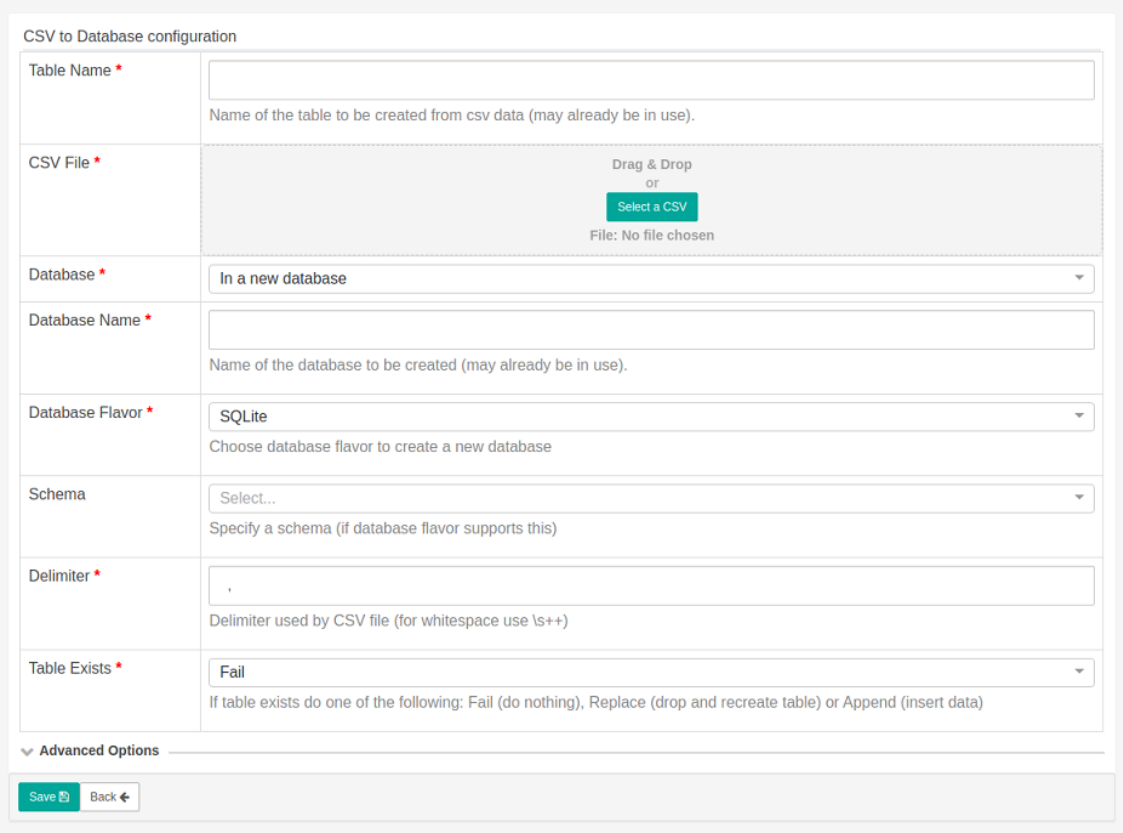


Abbildung 2: Neu designte Eingabemaske in Superset für den Import einer eigener CSV-Datei

Die Umsetzung bediente sich neuer Technologien (statt Flask neu React), wodurch Pionierarbeit geleistet wurde. Für zukünftige Anpassungen an User-Interfaces kann man die von uns geleistete Vorarbeit als Vorbild nehmen.

Gleichzeitig wurden wichtige Informationen über die optimale Konfiguration und Handhabung von Superset zusammengestellt und aufbereitet, damit das Institut für Software der HSR in der Lage ist, eine eigene Instanz anzubieten. Die gesammelten Informationen dienen ebenfalls als Hilfestellung für die Erstellung von Kursunterlagen für die kommenden Weiterbildungskurse, welche ebenfalls vom Institut für Software durchgeführt werden.

In einem zweiten Teil wurde eine Geokodierungs-Funktion implementiert. Geokodierung bedeutet, dass basierend auf einer Adresse die zugehörigen Längen- und Breitengrade bestimmt werden. Diese Koordinaten können dann dazu verwendet werden, die Adresse auf einer Karte anzuzeigen. Damit konnte eine der zentralen Datenanreicherungsfunktionen direkt in Superset angeboten werden, was sowohl für den Nutzer einen Mehrwert bietet als auch Superset konkurrenzfähiger macht.

Ergebnisse
Durch die Überarbeitung des CSV-Importer Formulars, hat sich die Benutzerfreundlichkeit verbessert. Der Benutzer sieht nun ein übersichtliches User-Interface, mit dem er seine Daten hochladen kann. Zusätzlich bietet der neue CSV-Importer dem Benutzer mehr Freiheiten, indem er eine neue Datenbank erstellen lassen kann, in die er dann seine Daten speichert. Der Wechsel in Richtung Single-Page Applikation, basierend auf einem React-Frontend bietet den Entwicklern von Superset die Möglichkeit, auch in Zukunft ansprechende User-Interfaces anzubieten.
Mit Hilfe der Geokodierungs-Funktion können neu Daten mit Adressen angereichert und für die Visualisierung aufbereitet werden, ohne vorher über Drittanbieter eine solche Aufbereitung durchzuführen.

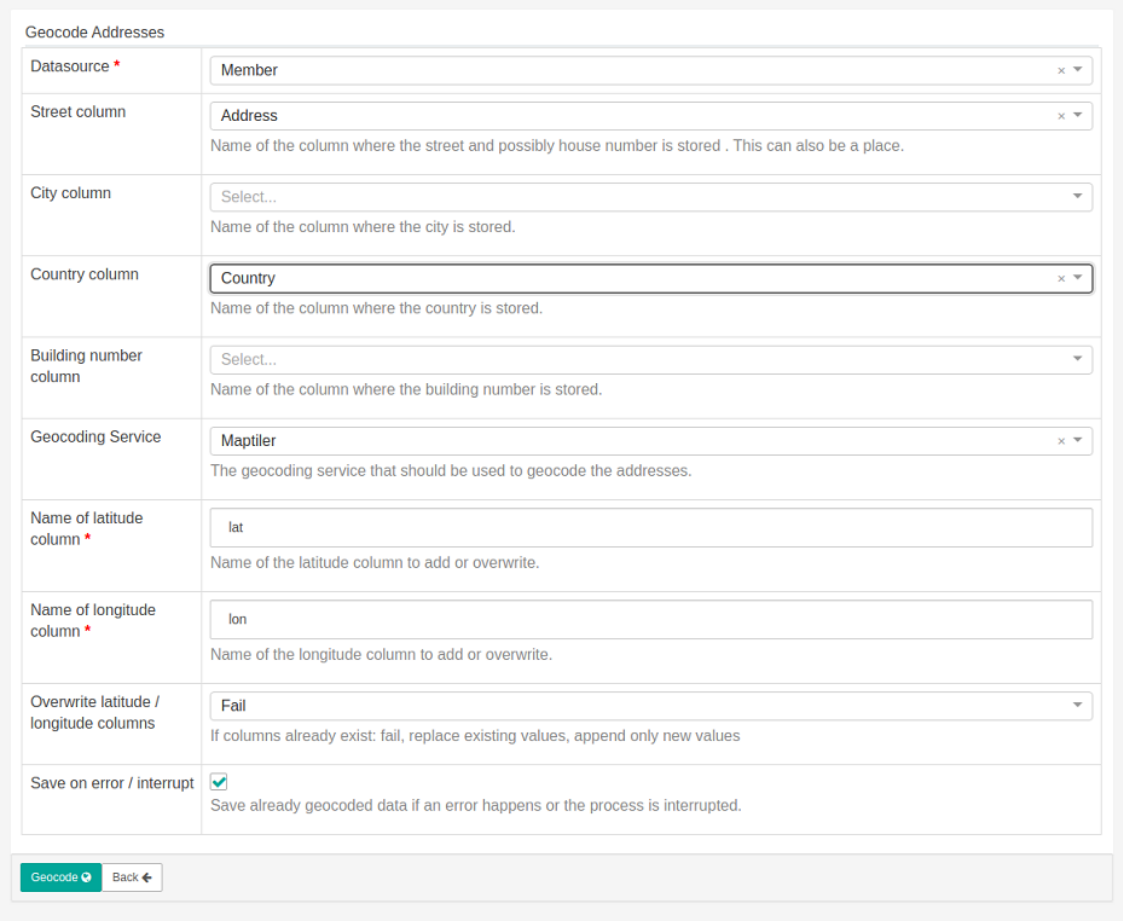


Abbildung 3: Zusätzlich implementierte Eingabemaske zur Geokodierung, d.h. zur Transformation von Adressen zu Koordinaten

Durch den Einsatz von passenden Programmier-Patterns können zukünftig, mit wenig Programmieraufwand, weitere Geokodierungs-Dienste eingebunden und genutzt werden.
Darüber hinaus wurden wichtige Erfahrungen im Zusammenhang mit der Entwicklung im Open Source Bereich gesammelt, besonders bei Superset. Diese Erfahrungen wurden in dieser Dokumentation festgehalten.

Ausblick

Die Dokumentation von Superset ist weiter ausbaufähig. Es gibt Anleitungen, wie man Superset für Weiterentwicklungen aufsetzen muss, diese sind jedoch nur auf Linux-Betriebssysteme ausgelegt. Informationen, wie das Projekt aufgebaut ist, welche Technologien im Einsatz sind oder welche Kriterien für die Weiterentwicklung in der Open Source Gemeinschaft gelten, werden kaum erwähnt. Hier haben wir bereits erste Schritte geleistet, aber es braucht noch mehr. Für zukünftige Entwickler wären solche Informationen hilfreich und würden den Rechercheaufwand reduzieren.

Superset hat noch einen langen Weg zu gehen, bis es wieder den neusten technologischen Standards entspricht. Viele weitere der User-Interfaces müssen überarbeitet und angepasst werden. Die aktuelle Implementation macht es den Entwicklern, aufgrund der starken Abhängigkeit der eingesetzten Technologie, schwierig diese Umstellung voranzutreiben.

Die Transformation bereits vorhandener Geometriedaten wäre eine wünschenswerte Funktion. Bei der Transformation von Geometriedaten geht es darum, gleichwertige Datentypen ineinander umzuwandeln. Beispielsweise kann aus einer Koordinate ein Punkt gemacht werden, da beide die identischen Informationen beinhalten.

Die von Superset zur Verfügung gestellten Grafiken benötigen stets ganz spezifische Datentypen. Dies macht sie unflexibel und der Benutzer muss seine Daten stets im Voraus aufbereiten. Hier könnte dem Benutzer einiges an Arbeit abgenommen werden, zum Beispiel durch die oben erwähnte Transformation bereits vorhandener Daten.

Die in dieser Arbeit umgesetzte Geokodierung könnte weiter ausgebaut werden. Aktuell wird jeweils der beste Treffer für die Koordinate gespeichert. Dies könnte man allerdings ausbauen, so dass dem Benutzer mehrere Treffer angezeigt werden. Vielleicht könnte es sogar so implementiert werden, dass der Benutzer selbst auf einer Karte einen Punkt markieren kann, falls alle Treffer falsch wären.

Da die Geokodierung bei vielen Adressen ein längerer Prozess sein kann, könnte auch noch ein Mailing-Dienst umgesetzt werden, der bei Beendigung des Geokodierungsprozesses den Benutzer informiert.

Auch der CSV-Importer hat noch Verbesserungspotential. Aktuell wird nur das UTF-8 Encoding angeboten, was für den Benutzer manchmal bedeutet, dass er sein CSV nochmals mit diesem Format abspeichern muss. Technisch gesehen wäre es möglich, weitere Encodings anzubieten.

Inhaltsverzeichnis

Teil I

Technischer Bericht

1 Einführung

In der heutigen Zeit ist das Sammeln von Daten für fast alle Unternehmen unerlässlich. Dadurch sind sie wettbewerbsfähig, rentabel und für die Zukunft gerüstet. Mit Hilfe von Daten lassen sich neue Erkenntnisse und Ideen kreieren und damit mehr Kunden akquirieren.[1] Das Analysieren dieser Datenflut (Big Data) geschieht mit sogenannten Business-Intelligence Tools, Werkzeuge zur Datenvisualisierung und Publikation. Sie bieten die Möglichkeit, gezielte Abfragen, Grafiken und Berichte zu erstellen. Diese basieren auf Data Mining, Prognosen, Trends, angewandten Analysen und Statistiken. Sie ermöglichen es den Planern und Entscheidungsträgern bessere Entscheidungen zu treffen, ohne dass diese technisches Wissen aufbauen müssen. Die Visualisierung der Ergebnisse dient des Weiteren als Kommunikationsmittel und ist selbst ein Werkzeug zur Analyse.[2]

Eine dieser BI Web Applikationen ist Apache Superset. Einige grosse Unternehmen wie American Express, Lyft, Twitter, Yahoo, Zalando etc. setzen auf die Funktionalitäten der Open Source Software. Sie erlaubt dem Benutzer einfach und ohne zu programmieren Daten anzuschauen, zu organisieren, zu filtern und zu teilen. Ursprünglich von Airbnb im Jahre 2015 unter dem Namen Panoramix und später Caravel entwickelt, ist Superset nun unter der Leitung der Apache Software Foundation und wird von einer grossen Entwicklergemeinschaft (im letzten Jahr ca. 54 Entwickler von Apache, Airbnb und diversen externen Firmen aus der ganzen Welt) in der Programmiersprache Python, der React-Bibliothek und mit den Flask-Frameworks weiterentwickelt.[3]

Apache Superset unterstützt eine Vielzahl an Datenbanktechnologien über die Schnittstelle und Softwarebibliothek SQLAlchemy[1]. Abfragen für Charts sind schnell erstellt und benötigen, je nach Benutzerrolle, keine Kenntnisse der Programmiersprache SQL. Die Daten können in über 50 verschieden Grafiktypen abgebildet werden. Dashboards gruppieren diese und können die Daten gegebenenfalls dynamisch filtern.

1.1 Problemstellungen und fehlende Funktionalitäten

Apache Superset hat aber auch einige Nachteile. Zum Beispiel werden geometrische Datentypen, wie Punkte, nur in bestimmten Grafiken unterstützt. Auch Adressen werden nicht automatisch in Geolokationen umgewandelt. Überdies gibt es nur eine vordefinierte Basiskartenquelle, welche sich nicht austauschen lässt. Die Overlay-Karte kann man auch nicht wählen oder ändern. Zudem ist das Importieren einer CSV-Datei in Superset mit Zusatzaufwand verbunden, sofern die Datenbankverbindung nicht vorgängig durch einen Administrator vorbereitet wurde. Des Weiteren ist der angebotene Support limitiert, da Probleme und Fehler, aufgrund des Open Source Charakters, vor allem über GitHub behandelt werden. Ausserdem ist das Aufsetzen für Weiterentwicklungen aus Softwareentwickler-Sicht mühsam. Die Entwicklung auf einem Windows System ist mit unzähligen Strapazen verbunden, da Superset komplett auf Unix ausgerichtet ist.

Mit unserer Arbeit möchten wir einige dieser Probleme und Erweiterungen angehen. Diese haben wir nach Kriterien wie Benutzerfreundlichkeit, Nützlichkeit und Machbarkeit priorisiert.

0. Aufsetzen und dokumentieren einer Software-Entwicklungsumgebung
1. Verbesserung des CSV-Importers
2. Implementation der Geokodierung von Adressen
3. Implementation der Transformation von Geometrie-Daten (Reserve)

[1] SQLAlchemy: https://www.sqlalchemy.org/

Nachfolgend sind die geplanten Features detaillierter beschrieben:

1.1.1 CSV-Importer

Um eine CSV-Datei importieren zu können, muss eine Datenbank bestehen, in welche die Daten der Datei gespeichert werden. Für diese Datenbank muss eine Datenbankverbindung eingerichtet werden und richtig konfiguriert sein, damit das Hochladen von CSV-Dateien erlaubt ist. Die Datei muss zwingend im UTF-8 Format sein, ansonsten kann es zu Problemen mit Umlauten kommen. Ausserdem werden nur Dateien mit der Endung .csv akzeptiert. Je nach darunterliegender Datenbank können auch Sonderzeichen, wie zum Beispiel Klammern, dazu führen, dass die Daten nicht importiert werden können.

1.1.2 Geokodierung

Um Adressen in einer Kartengrafik anzeigen zu können, müssen diese zuvor in Koordinaten (typischerweise in Längen- und Breitengrade) umgewandelt werden. Viele Konkurrenzprodukte ermöglichen das Geokodieren von Adressdaten oder erlauben dem Benutzer direkt in den Grafiken Adressen anzugeben. Superset bietet in der aktuellen Version keine Möglichkeit an, adressbasierte Daten zu geokodieren. Deshalb muss der Benutzer seine Daten entweder mit SQL-Befehlen nachträglich oder vor dem Einlesen in eine Datenbank, mit Hilfe eines Drittanbieter-Dienstes, mit Koordinaten anreichern. Aus Sicht des Benutzers ist dies sehr umständlich, da er einiges an Mehraufwand betreiben muss, um mit Daten Kartengrafiken erstellen zu können.

1.1.3 Transformation von Geometrie-Daten (Reserve)

Die aktuell verfügbaren Grafiktypen in Superset benötigen immer ganz spezifische Daten. Gewisse Grafiken verlangen Längen- und Breitengrade, andere wiederum Polygone und wieder andere benötigen GeoJSON. In der aktuellen Version von Superset gibt es keine Möglichkeit, in einer bestimmten Grafik einen anderen Datentyp anzugeben als derjenige, der von der Grafik explizit verlangt wird. Obschon viele dieser Datentypen miteinander verwandt sind und entsprechend zueinander transformiert werden könnten. Für einen Benutzer bedeutet das jedes Mal einen hohen Zusatzaufwand, seine Daten für die entsprechende Grafik aufzubereiten.

1.2 Vision

Die Open Source Software Apache Superset soll eine gute Alternative zu den kommerziellen BI-Tools sein und ihnen an Funktionalität und Benutzbarkeit nicht nachstehen.

Die Bedienung der Web Applikation muss sowohl für Businessanalysten, aber auch für technisch versierte Benutzer einfach sein. Wir möchten das Einlesen verschiedener Datenquellen ohne grosse Umwege und Hindernisse gestalten. Geographische Daten sollen schnell und unkompliziert verarbeitet und in den entsprechenden Grafiken individuell und korrekt dargestellt werden. Dabei darf die Performanz bei Abfragen über grosse Datenmengen nicht verringert werden.

1.3 Ziele und Unterziele

Marktforschung Bisher hatten wir wenig Kontakt mit BI-Tools. Deshalb möchten wir die Funktionsweise und deren Einsatzgebiete genauer studieren. Dabei legen wir besonderen Wert auf die Web Applikation Apache Superset.

Mit Hilfe der Marktforschung möchten wir sehen, welchen Stellenwert Apache Superset hat. Des Weiteren ermitteln wir dadurch die Funktionalitäten, welche der Software fehlen. Anhand dieser können weitere Anforderungen definiert werden.

Problemstellungen und Erweiterungen Wir möchten möglichst viele Problemstellungen von Apache Superset lösen, damit die Web Applikation mit den anderen BI-Anbietern konkurrenzieren kann. Wir wollen ausserdem den Benutzern mehr Funktionalität und Individualität bieten, um ihr Anwendererlebnis zu steigern. Komplizierte und aufwendige Prozesse, die mehrere Schritte beanspruchen, möchten wir vereinfachen.

Dokumentation Erfolgserlebnisse sowie Fehlschläge werden dokumentiert. Die Dokumentation soll anderen einen Einblick in die Herausforderungen und Lösungen der Entwicklung mit Apache Superset geben. Zudem möchten wir eine einfache und verständliche Installationsanleitung für die Inbetriebnahme der Web Applikation erstellen.

2 Stand der Technik

In diesem Kapitel zeigen wir die Funktionalitäten einiger BI-Tools, nebst Apache Superset, auf. Zudem gehen wir auf die Entwicklung der BI-Tools ein.

2.1 Apache Superset

Immer öfters hört man von der Open Source Web Applikation Apache Superset. Aufbereitete Daten können einfach analysiert, visualisiert, bzw. präsentiert und publiziert werden und das ohne besondere Programmier- oder anderweitige Kenntnisse. Die Software unterstützt eine Vielzahl an Datenbankquellen und kann auch Daten aus einer CSV-Datei verarbeiten. Darüber hinaus kann man die als Listen, Grafiken oder Karten visualisierten Daten mit anderen Personen teilen.

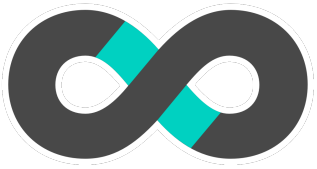


Abbildung 4: Superset Logo

2015 wurde Superset von der Firma Airbnb, während eines Hackathon, unter dem Namen Panoramix entwickelt. Den Namen haben sie sich von dem Druiden aus den Asterix-Comics abgeschaut. Panoramix kommt aus dem Griechischen und bedeutet so viel wie Weitsicht bzw. der, welcher alles sieht. Ein Jahr später benannten sie die Software zu Caravel um, nach einem portugiesischen Entdeckerschiff. Nur 6 Monate später gab es einen erneuten Namenswechsel zu Superset. Nun unter der Leitung der Firma Apache, heisst sie Apache Superset.

Superset ermöglicht es, Datensätze auf verschiedene Arten zu erforschen. Dies erfolgt durch selektieren, filtern oder darstellen, mittels einer Auswahl von ansehnlichen Grafik- und Kartentypen. Diese können in einem Dashboard mit wenigen Klicks zusammengestellt und mit anderen geteilt werden.

Die Sicherheit spielt bei Apache Superset eine grosse Rolle. Die allgemeinen Funktionalitäten können für verschiedene, selbst definierte Rollen eingeschränkt, respektive freigeschaltet werden. Jedem Benutzer kann ein Administrator eine oder mehrere dieser Rollen zuweisen. Ausserdem können einzelne Benutzer auf verschiedenen Datenbanken, Datenquellen, Dashboards und Charts berechtigt werden. Dadurch wird eine flexible Rechte-Konfiguration ermöglicht. Zusätzlich werden die Aktivitäten der Benutzer in einem Log festgehalten.[4–6]

2.2 Marktanalyse

Nachfolgend stellen wir einige der bekanntesten BI-Tools vor. Aufkommende Monitoring-Tools mit BI-Funktionen wie Kibana oder Graphana werden wir im Rahmen dieser Arbeit nicht untersuchen.

2.2.1 Chartio

Das 44-Köpfige Team von Chartio betreibt ein kommerzielles BI-Tool, welches nebst einer grossen Anzahl an Datenspeichern, besonders mit der Integration in die Cloud wirbt. Chartio ist ausschliesslich online verfügbar und beginnt bei 40$ pro Monat. Seit 2015 werden monatlich neue Releases veröffentlicht. Ihre Dokumentation ist gut aufgebaut und erklärt die Funktionen und Begrifflichkeiten zusätzlich mit Videos und Tutorials. Ausserdem gibt es einen Blog, der den Anwender auf Neuigkeiten hinweist.

CHARTIO

Abbildung 5: Chartio Logo

Besonders für technisch nicht versierte Benutzer bietet Chartio einen grossen Nutzen, weil sie ihre Daten per “Drag and Drop“ und mit vordefinierten Aggregationsfunktionen visualisieren und anpassen können. Zur Auswahl stehen die üblichen Grafiktypen wie Heat Map, Bar- oder Column-Chart, etc. Im Bereich Spatial Intelligence ist die Auswahl jedoch beschränkt. Es gibt nur zwei Diagrammtypen, die nicht mit Adressen umgehen können, dafür aber mit Ländernamen sowie Längen- und Breitengraden.

Die Charts und Dashboards lassen sich einfach exportieren und teilen, ohne dass die Empfänger bei Chartio registriert sein müssen. [2]

2.2.2 Insights for ArcGIS

ArcGIS Insights ist ein Tool der Firma esri. Die neuste Version kam im Juni 2019 heraus. Das Produkt ist für kommerzielle Zwecke kostenpflichtig, mit mindestens 1500$ pro Benutzer pro Jahr. Es gibt allerdings die Möglichkeit, die Software kostenlos zu nutzen. Dafür muss man lediglich ein öffentliches Konto erstellen. Ausserdem gibt es eine 21-tägige Testversion, die ebenfalls gratis ist.

Abbildung 6: Insights for ArcGIS Logo

Wie Chartio, bietet auch ArcGIS viele Grafiktypen für verschiedene Vorhaben an. Als Datenquellen sind sowohl Datenbanken sowie CSV- als auch Excel- und Zip-Dateien erlaubt. Zudem ist bei den Versionen Insights Enterprise und Local eine Anbindung an Geo-Datenbanken möglich. Des Weiteren können Adressen direkt geokodiert werden, um die Anzeige auf einer Karte zu ermöglichen. Die Hintergrundkarten dazu kann man im ArcGIS Onlinetool aus einer grossen Auswahl an Quellen ändern. Dieses Tool ist eng verbunden mit Insights.

Eine Hilfsbibliothek steht bei Problemen zur Verfügung. Neben Einstiegsinformationen und Übungen, um das Tool kennenzulernen, werden auch Beiträge für alltägliche Aktivitäten angeboten. [3]

2.2.3 Looker

Über 1700 Kunden sind von den analytischen Fähigkeiten von Looker überzeugt. Im Gegensatz zu den anderen BI-Tools, bietet Looker zwei Modi an: Produktion und Entwicklung. In der Entwicklung können die Abfragen und Grafiken erstellt werden. Die Produktion ist für die Business-Analysten, welche die Grafiken filtern und in Dashboards verschieben, bzw. neue erstellen können.

Abbildung 7: Looker Logo

Die Abfragen werden mit einer eigens entwickelten Sprache namens LookML (Wrapper um SQL) kreiert. Ausserdem werden sie mit Git verwaltet, so dass diese nicht während der Entwicklung im produktiven Modus zur Verfügung stehen. Auch Looker unterstützt eine Vielzahl an Datenbanken und Diagrammtypen. Zudem kann man mehrere Diagramme übereinander in einer Grafik darstellen. Unglücklicherweise sind die Geodaten-Diagramme auf Amerika und Grossbritannien ausgelegt. Für alle anderen werden lediglich die Länder, anhand ihrer Namen oder ihres Codes markiert.

Looker ist eine Multi Cloud-Lösung. Die Kosten werden jeweils individuell, anhand der Anzahl Benutzer und dem Umfang des Deployments, berechnet. Mit Hilfe der Lernprogramme, der Community, dem offiziellen Support und einer relativ grossen Dokumentation ist die Bedienung sehr einfach. [4]

[2] Chartio: https://chartio.com/
[3] Insights for ArcGIS: https://insights.arcgis.com/
[4] Looker: https://looker.com/

2.2.4 Mode

Mode ist eine komplett webbasierte Open Source Lösung, welche sowohl eine gratis als auch eine Premium Version beinhaltet. Der Preis ist, wie bei Looker, abhängig von der Anzahl Benutzer, dem Umfang des Deployments und den organisatorischen Anforderungen.

Abbildung 8: Mode Logo

Die Inbetriebnahme gestaltet sich sehr einfach. Zu beachten ist, dass beim Einrichten der Datenquellen eventuell der Zugriff von gewissen IPs erlaubt werden muss. Als Datenquellen werden diverse bekannte und weniger bekannte Hersteller unterstützt (Amazon, Oracle, MySQL, MS Azure SQL, PostgreSQL, etc.). Die resultierenden webbasierten Reports sind entsprechend einfach durch JavaScript erweiterbar. Ausserdem bietet Mode ein Skript mit diversen Visualisierungen an. Dieses ist frei verfügbar. [5] Spatial Intelligence wird nur begrenzt unterstützt. Als Geolokationsservice kann man Google Maps einbinden. Der Dienst nimmt jedoch nur separate Längen- und Breitengrade entgegen.

Der Import von Daten aus nicht datenbankspezifischen Quellen, zum Beispiel CSV oder Excel, wird nicht unterstützt. Zudem empfiehlt Mode, aus Sicherheitsgründen, auf den Quellen lediglich Lesezugriff zu erlauben.

Die Dokumentation beinhaltet diverse Seiten für den alltäglichen Gebrauch und den Einstieg in Mode. Zusätzlich gibt es ein Forum für Fragen und Vorschläge. Das Produkt wird aktiv weiterentwickelt. Durch den Open Source Aspekt gibt es mehrere Entwickler, welche ab und zu Fehler beheben oder neue Funktionen hinzufügen. [6]

2.2.5 MS Power BI

Microsoft Power BI bietet, neben einer Web- und Desktop-Version, auch eine mobile Komponente an. Die gratis Desktop-Version ist gut geeignet für Einzelpersonen. Die Pro- oder Premium-Versionen gibt es ab 10$ pro Monat.

Abbildung 9: MS Power BI Logo

Zu einer Support Dienstleistung und einer Community, führt die Software rollenspezifische Dokumentationen und Einsteigerhilfen. Der Einstieg ist sehr intuitiv. Auch die Inbetriebnahme gestaltet sich einfach, weil alles bei Microsoft betrieben und die Desktop Version direkt aus dem Microsoft Store installiert werden kann.

In MS Power BI kann man neben vielen Datenbanken auch Dateien wie CSV oder Excel importieren, welche in den verschiedensten Grafiken dargestellt werden können. Eine Geokodierung von Städte- und Ländernamen zu einem GIS-kompatiblen Datentyp ist bereits eingebaut. Geometrische Daten können jedoch nicht verarbeitet werden. Zudem bereitet die Funktion “Custom Column“ Schwierigkeiten, da sie nicht einfach zu bedienen ist.[7]

[5] Mode Skript: https://github.com/mode/alamode
[6] Mode: https://mode.com/
[7] MS Power BI: https://powerbi.microsoft.com/

2.2.6 Qlik

Abbildung 10: Qlik Logo

2019 wurde Qlik von Gartner zum 9. Mal als Leader eingestuft. [8] Qlik ist ein Produkt, welches ab 30$ pro Monat in der Cloud als “Software as a Service“ bezogen oder selbst gehostet werden kann. Es besticht durch seine grosse Vielfalt an Diagrammen, die viele andere BI-Tools nicht haben, z.B. die Kombination mehrerer Diagrammtypen. Auch die Abfrage und Zusammenstellung der Daten ist durch “Drag and Drop“ und textuell verarbeiteten Aggregationsfunktionen intuitiv und einfach. Die eingebaute künstliche Intelligenz hilft, zukünftige Daten vorherzusehen und treffendere Aussagen zu machen.

Besonders mit der Verarbeitung geographischer Daten hebt es sich von der Konkurrenz ab. Es können mehrere Basiskarten und Ebenen zur Darstellung der Daten gewählt werden. Sowohl Kontinente als auch Länder, Postleitzahlen, Städte, Dörfer, Längen- und Breitengrade werden von Qlik erkannt.

Es gibt eine umfangreiche Dokumentation, aufgeteilt in Produkte und Installationsplattformen. Die Videos helfen bei dem Verständnisaufbau. Ausserdem bietet Qlik Schulungen, Support und Consulting als zusätzliche Hilfestellungen an.

Fast jeden Monat publizieren sie eine neue Version. [9] [10]

2.2.7 R mit Shiny

Abbildung 11: Shiny Logo

Shiny ist lediglich eine Bibliothek für die Programmiersprache R. Sie erlaubt einem Anwender, auf einfache Art und Weise, Web Applikationen zu erstellen, ohne dass dieser Web-Programmierkenntnisse besitzen muss. RStudio bietet Tools und viele Angebote an, um diese Applikationen zu erweitern und zu betreiben. Da man die Visualisierungen in R programmiert, können weitere Bibliotheken dazu gezogen und dadurch der Funktionsumfang erweitert werden. Als Datenquellen werden von RStudio nur PostgreSQL und SQLite unterstützt.

R ist eine verbreitete Programmiersprache. Deshalb gibt es für diese viele Dokumentationen. Shiny stellt mit Einsteigervideos, Präsentationen, Guides und diversen Beispielen Hilfestellungen. Die Bibliothek hat 44 Entwickler auf GitHub. Allerdings sind davon lediglich 6 Entwickler aktiv. Die kostenpflichtigen Produkte RStudio Server Pro und RStudio Connect werden von RStudio weiterentwickelt. [11] [12] [13]

2.2.8 Tableau

Tableau ist eines der bekanntesten kommerziellen BI-Tools. Der erste Release erfolgte 2003. Seitdem erscheint etwa alle drei Monate eine neue Version mit zusätzlichen Funktionalitäten. Die zurzeit aktuelle Version ist 2019.3. Per Ende Jahr wird Tableau von Salesforce für über 15.7 Milliarden$ übernommen. Tableau ist eine kommerzielle Software, die im Minimum 840$ im Jahr kostet. Je nach Bereitstellungsoption (selbst hosten oder von Tableau Online hosten lassen) und je nach Anzahl und Art der Lizenzen steigen die Kosten.

[8] Gartner Magic Quadrant: https://www.qlik.com/us/gartner-magic-quadrant-business-intelligence
[9] Qlick: https://www.qlik.com/
[10] Qlick Dokumentation: https://help.qlik.com/
[11] R Dokumentation: https://docs.rstudio.com/
[12] Shiny: https://shiny.rstudio.com/
[13] R Studio: https://rstudio.com/

Die Lizenzen unterscheiden sich in den Funktionalitäten. Der “Creator“ kann alles, während der “Explorer“ nur Arbeitsmappen und Dashboards erstellen, bearbeiten und analysieren kann. Die “Viewer“ sehen lediglich die erstellten Dashboards.

Abbildung 12: Tableau Logo

Es gibt eine umfangreiche Dokumentation, die auch in verschiedenen Sprachen verfügbar ist. Die Inbetriebnahme ist dank dieser sehr einfach. Allerdings sind nicht alle Seiten übersetzt und teilweise funktionieren die Verweise nicht. Tableau lässt sich sowohl auf Windows- und Linux-Servern, in der Cloud und auf mobilen Geräten installieren. Bei Tableau Online wird der Server von Tableau selbst gehostet, deshalb kann bei diesem Produkt sofort nach der Registrierung mit der Bereitstellung der Daten begonnen werden. Ausserdem gibt es über 100 Schulungsvideos, zahlreiche Lernressourcen, Schulungen, Beratungen und einen inbegriffenen Support zu Bürozeiten, welcher zu einem Aufpreis auch 24/7 Hilfestellung leistet.

Abfragen über mehrere Excel-Datenblätter, Tabellen oder Datenquellen sind möglich. Analysen der Daten können durch Echtzeit-Visualisierungen in 24 verschiedenen Grafiken, auf einer Vielzahl von Datenbank-Infrastrukturen, getätigt werden. Dabei werden dem Anwender die dafür am geeignetsten Diagramme vorgeschlagen, welche wiederum mittels Machine Learning weiter verarbeitet, in Dashboards dargestellt oder mit anderen geteilt werden können. Des Weiteren wandelt Tableau geographische Einträge automatisch in Längen- und Breitengrade um und bietet weitere analytische Funktionen an, wie zum Beispiel Trendlinien oder Boxplots. Eine Besonderheit von Tableau ist, dass es die natürliche Sprache verarbeiten kann. Beispielsweise wird durch die Eingabe von “Durchschnitt“ die Aggregation AVG angewendet. So können auch Analysten ohne technische Kenntnisse Abfragen erstellen und filtern.[14]

2.2.9 Zoho Analytics

Zoho Analytics ist ein BI-Tool der Firma Zoho. Sie bieten zwei Versionen an: Eine Online (ab 22 Euro/Monat) und eine On-Premise (30 Euro/Monat). Für Einzelpersonen ist die Letztere kostenlos, beschränkt jedoch die Datengrösse.

Abbildung 13: Zoho Analytics Logo

Als Datenquellen werden diverse Möglichkeiten angeboten. Neben CSV, Excel, JSON, XML, HTML und Text-Dateien, werden auch Onlinespeicher (Cloud), relationale und NoSQL Datenbanken unterstützt.

Das Programm wird seit Jahren aktiv weiterentwickelt und fügte im September 2019 den Geo Heatmap Support hinzu. Zoho hat wie die anderen Tools viele Diagrammtypen zur Auswahl. Besonders im Umgang mit geographischen Daten bieten sie hilfreiche Funktionalitäten an. Zum Beispiel werden Orts- und Ländernamen automatisch in die entsprechenden Koordinaten für Karten umgewandelt.

Die Inbetriebnahme gestaltet sich als sehr einfach. Es gibt diverse Dokumente für den Einstieg und spezifische Funktionen. Zusätzlich zum üblichen User Guide gibt es auch eine Schnittstellenbeschreibung, Videos, Webinars, eBooks, eine Galerie mit Beispielen und ein Forum für den Austausch zwischen den Anwendern. [15]

[14] Tableau: https://www.tableau.com/
[15] Zoho Analytics: https://www.zoho.eu/

2.3 Von EVA zu EVAP

Täglich werden Daten verarbeitet, um aus ihnen Informationen zu gewinnen. Das Grundschema der Datenverarbeitung ist das EVA Prinzip. Dieses wurde später, als die geographischen Informationssysteme aufkamen, um die Präsentation (P) erweitert.[7]

Eingabe Die Dateneingabe erfolgte früher über Lochstreifen und -karten, bis sie von Tastatur, Maus, Touchscreen und Spracheingabe abgelöst wurden. Heutzutage geschieht die Eingabe grösstenteils über das maschinelle Einlesen automatisch, zum Beispiel aus einer Datenbank. Einige Schritte werden dennoch weiterhin manuell betrieben, wie das Befüllen einer Datenquelle.

Verarbeitung Die eigentlichen Rechenprozesse finden in diesem Bereich statt. Der Rechner nimmt die Daten und Befehle entgegen und bereitet diese mit Funktionen oder Programmen auf.

Ausgabe Zur Ausgabe gehören sowohl das Speichern als auch das Darstellen, Abspielen und Weiterleiten der Daten.

Präsentation Das Bedürfnis der Datenpräsentation kam bereits im 19. Jahrhundert auf. Zwischen 1800 und 1850 wurden die heute bekannten Diagramme Histogramme, Balken-, Kuchen-, Linien-, Zeitreihen-, Kontur-, Streudiagramme etc., sowie die erste geologische Karte erfunden. Danach folgten 3D Grafiken und die ersten GIS Systeme kamen auf. Für die räumlichen Daten setzte man damals die Rastertechnik ein, um nützliche Informationen für ein bestimmtes Problem zu gewinnen. Heute legt man oftmals mehrere Datensätze als Referenzpunkte übereinander, zum Beispiel auf einer Karte.[8] Die Fähigkeit, dreidimensionale Bilder und Objekte zu verstehen, nennt man Location Intelligence oder auch Spatial Intelligence.[9]

"*The graphic is no longer only the 'representation' of a final simplification, it is a point of departure for the discovery of these simplifications and the means for their justification. The graphic has become, by its manageability, an instrument for information processing...*" Ein Zitat des Kartographen und Theoretikers Jacques Bertin. Er sprach 1967 in seinem Buch "Sémiologie Graphique" über die Grundlage für Informationsgrafiken. Eine systematische Klassifizierung der Verwendung visueller Elemente zur Anzeige von Daten und Beziehungen. Diese Elemente bestanden aus Position, Form, Orientierung, Farbe, Textur, Wert und Grösse. [10]

Auch Edward R. Tufte prägte durch sein Buch "The Visual Display of Quantitative Information" wie man Daten für eine präzise, effektive und schnelle Analyse darstellt. Beispielsweise rät er, keine Grafik zu verwenden, wenn nicht viele Daten vorhanden sind. Ausserdem ist weniger mehr: "*Above all else show the data*". Des Weiteren erklärt er den "Lie Factor": "*The representation of numbers, as physically measured on the surface of the graphic itself, should be directly proportional to the quantities represented.*" [11]

Dieses Thema beschäftigt auch heute noch die Business-Welt. Dona M. Wong, eine ehemalige Studentin von Edward R. Tufte, beschreibt in ihrem Buch, die Dos and Don'ts der Datenpräsentation. Sie geht weniger auf die Theorie ein, sondern zeigt direkt mit Bildern auf, was gut und schlecht ist. Teilweise wiederholt sie Thesen von Edward R. Tufte, wie zum Beispiel, dass man mit Kuchendiagrammen sparsam umgehen sollte und sich ein Balkendiagramm oftmals besser als Vergleich eignet.[12]

Das Problem, welches sich daraus ergibt, ist, dass immer mehr Daten gesammelt werden. Dabei wird es schwieriger, die relevanten Informationen aus diesen zu filtern. Die oben erwähnten BI-Tools können mit grossen Datenmengen umgehen und helfen, diese zu verarbeiten, auszugeben und in geeigneten Grafiken zu präsentieren. Ausserdem setzen viele dieser Tools bereits künstliche Intelligenz ein, um die Verarbeitung zu verbessern.[9]

3 Umsetzungskonzept

In der ersten Phase lernten wir die Web Applikation Superset kennen. Wir experimentierten mit den Funktionen und machten erste Erkenntnisse über dessen Eigenschaften. Dazu analysierten wir den Code sowie die eingesetzten Technologien und Frameworks. Danach untersuchten wir weitere BI-Tools und erforschten deren Funktionalitäten. Aufgrund dessen konnten wir die Problemstellungen definieren, welche wir während dieser Arbeit umgesetzt hatten.

Die Implementation unterteilten wir in mehrere Arbeitspakete. Wir entschieden uns dafür, zuerst den CSV-Importer zu verbessern. Dies bat uns einen guten Einstieg in die Entwicklung mit Superset und React. Wir erhielten nützliche Kenntnisse, während wir die Flask-Komponenten durch React ersetzten. Ausserdem lernten wir den Umgang mit Unit- und Cypress-Tests, was wir bei der nachfolgenden Umsetzung wiederverwenden konnten.
Zuerst fokussierten wir uns auf das Erstellen einer neuen SQLite Datenbank. Diese ermöglichten wir mit der SQLAlchemy Bibliothek und Pandas. Das Hinzufügen der PostgreSQL Datenbank war durch diese Vorarbeit schnell erledigt.

Als nächstes setzten wir die Geokodierung von Adressen um. Unter Geokodierung versteht man das Zuordnen von geographischen Informationen zu Koordinaten. Dank den Erkenntnissen des CSV-Importers konnten wir viele der Anforderungen schnell umsetzen. Wir implementierten zuerst die verschiedenen API-Schnittstellen für das Abfragen der Spalten der Tabelle, die Geokodierung, das Abfragen des Fortschritts und das Abbrechen der Geokodierung. Aufgrund der eingeschränkten Nutzungsbedingungen der Services in GeoPy, wählten wir als ersten Geokodierungs-Service MapTiler. Danach kamen Google und ein Mocking-Service für die Testfälle dazu.
Für die Geokodierungs-Services mussten wir keine Bibliotheken einsetzen. Die Abfragen der Schnittstellen erfolgten über deren Request-URI. Um möglichst einfach weitere Geokodierungs-Services hinzufügen zu können, setzten wir das Factory-Method Pattern[13] ein. Wir haben dadurch eine allgemeingültige Aufrufstruktur, wobei nur wenige Methoden spezifisch auf den Geokodierungs-Service programmiert werden müssen. Alle SQL-spezifischen Befehle wurden mit der SQLAlchemy Bibliothek umgesetzt.

Während der Umsetzung des CSV-Importers eröffneten wir sogenannte Superset Improvement Proposals (SIPs). Durch diese konnten wir mit den Entwicklern von Superset über unsere Vorhaben diskutieren und Verbesserungsvorschläge einarbeiten. Aufgrund der vielen Anpassungen und der Tatsache, dass wir nach dieser Arbeit nicht mehr für Bug-Fixing zur Verfügung stehen, wurden die Pull Requests nicht angenommen. Nichtsdestotrotz wird unsere Arbeit in der HSR weiter gepflegt und verwendet.

4 Resultate

In diesem Kapitel werden die Resultate unserer Implementationen präsentiert.

4.1 Zielerreichung

Die beiden geplanten Features “Verbesserung des CSV-Importers“ und “Implementation der Geokodierung von Adressen“ wurden umgesetzt. Die Reserve-Features realisierten wir nicht, da der Umbau von Flask auf React deutlich mehr Arbeit in Anspruch nahm als ursprünglich angenommen. Allerdings könnte man den Umbau auch als eigenes Feature betrachten. Mit unserer Zielerreichung lagen wir dennoch gut im Plan.
Als Resultat dieser Arbeit, wurde ein erster Schritt in Richtung Single-Page Application mit React getätigt. Dies ist auch von den Superset-Maintainern geplant und aus ihrer Sicht notwendig. Dabei können unsere Implementierungen als Vorlage dienen.
Des Weiteren wurde mit der Geokodierung eine weitere Lücke zwischen Superset und den kommerziellen Produkten geschlossen.
Wir konnten mit unserem Code ausserdem aufzeigen, wie man gewisse Teile des bestehenden Projektes umbauen und übersichtlicher gestalten könnte. So wurden Patterns wie Dependency-Injection[14] oder Factory Method[13] verwendet, welche die Erweiterbarkeit und Testbarkeit immens verbessern. Die Implementation kann dadurch als Vorbild für weitere Umbauten und Neuentwicklungen verwendet werden.
Zu guter Letzt haben wir Informationen für Folgearbeiten gesammelt, bezüglich dem Projektsetup, der Projekterweiterung und dem Kontakt mit den Maintainern und diese entsprechend dokumentiert. Dadurch können sich Folgearbeiten an diesen orientieren und wissen, auf was geachtet werden muss, um den Code letztlich bei Superset integrieren zu können.

4.2 Vergleich zu bisherigen Lösungen

Vergleicht man Superset mit anderen Open Source Business-Intelligence-Tools, so ist es zurzeit unter den Top 10[16] und könnte seine Position mit unseren Verbesserungen weiter ausbauen. Bezieht man in diesen Vergleich allerdings auch kostenpflichtige Alternativen mit ein, so stellt man fest, dass Superset noch nicht mit deren Funktionsumfang mithalten kann. Unsere Beiträge an das Projekt sind dabei lediglich ein erster Schritt.

4.3 Ausblick: Weiterentwicklung

In diesem Abschnitt möchten wir darauf eingehen, was an Superset im Allgemeinen verbessert werden könnte.

4.3.1 Umbau zu einer Single-Page-Application

Die Zukunft von Superset liegt in einem Umbau der mittlerweile in die Jahre gekommenen Flask-Form User-Interfaces, zu einer Single-Page Application, basierend auf React und Redux. Das Backend soll zukünftig ausschliesslich aus REST-Schnittstellen bestehen und von der Single-Page Application aufgerufen werden. Dies bestätigt auch das Feedback der Maintainer von Superset. Es müssen Design-Standards spezifiziert und durchgesetzt werden. Zusätzlich benötigt es grundlegende und wiederverwendbare React-Komponenten, damit die vielen Formulare und Tabellen effizient umgestellt werden können.

[16] Top 10 Open Source BI-Tools: https://www.softwaresuggest.com/blog/best-free-open-source-bi-tools/

4.3.2 API-Dokumentation

Alle bisherigen APIs sind nicht für eine API-Dokumentation vorbereitet. Das eingesetzte Framework Flask unterstützt bereits auf YAML-Syntax basierte Docstrings, diese werden aber im gesamten Projekt nicht genutzt. Hier müsste man die Vererbungshierarchie überarbeiten, auf welcher alle APIs basieren. Zusätzlich müssten die entsprechenden “Data Contracts“ im YAML Format bei den APIs hinzugefügt werden, damit es in Zukunft eine API-Dokumentation, basierend auf OpenApi (früher Swagger genannt), geben kann.

4.3.3 Aufteilung des Codes/Refactoring

Wenn man sich einen Überblick über die Codebasis von Superset verschaffen möchte, stellt man fest, dass ein Grossteil der Logik, sich in einem Dokument befindet. In diesem wiederum, findet man den grössten Teil in der Klasse „Superset“. Zusätzlich gibt es diverse unklare oder gar versteckte Abhängigkeiten, welche durch Flask automatisch verwaltet werden.
Wir haben unsere Logik in eigene Klassen und Dateien ausgelagert. Dadurch ist das Projekt besser zu warten und das Auffinden von spezifischen Funktionen wird vereinfacht. Des Weiteren ist auch das gemeinsame Arbeiten besser möglich. Man kann einfacher aneinander vorbei arbeiten ohne diverse Merge-Konflikte auflösen zu müssen.

4.3.4 Klarere Projekt-Dokumentation und Richtlinien

Für Aussenstehende ist es sehr aufwendig, wenn man am Projekt mitarbeiten möchte. Die offizielle Entwickler-Dokumentation ist unvollständig. Ausserdem ist es sehr schwierig, Pull-Requests angenommen zu bekommen, da nur kleine Änderungen von Aussenstehenden gewünscht sind. Auch bei diesem Verbesserungsvorschlag, haben wir bereits Vorarbeit geleistet. Diese Hindernisse und Erkenntnisse haben wir dokumentiert.

4.3.5 Automatische Transformation verschiedener Geometrie Datentypen

Die in Superset vorhandenen Karten-Diagrammtypen erkennen lediglich eine Art von Geometrie. Beispielsweise muss die Datenquelle Längen-, Breitengrade oder POINT-Werte aufweisen. Der Benutzer muss sicherstellen, dass dieselben Geometriedaten[15] in verschiedenen Formen in der Datenquelle abgelegt sind, wenn er verschiedene Diagramme verwenden möchte. Das führt unweigerlich zu Problemen, sollten diese Daten angepasst werden müssen (z.B. Update-Anomalien). Ausserdem sind diese Informationen redundant, denn zwischen einer Koordinate im POINT-Format und einer Koordinate in Form von Längen- und Breitengraden, gibt es bis auf das Format keinen Unterschied. Des Weiteren ist es ein grosser Mehraufwand für den Benutzer, die Daten in die verschiedenen Formate zu transformieren.
Hier sollte man die verschiedenen Diagramme entsprechend erweitern. Ein Diagramm sollte möglichst alle Standard-Geometrietypen akzeptieren und gegebenenfalls transformieren können.
Es gibt allerdings auch einige Ausnahmen. Benötigt ein Diagramm beispielsweise Polygone, kann man nicht eine Datenquelle mit “POINT“ Werten verwenden, da dann Informationen fehlen.

Teil II

SW Projekt Dokumentation

5 Überblick

Dieses Kapitel dient dazu, einen Überblick darüber zu geben, wie wir das Projekt organisiert und aufgeteilt haben.

5.1 Projektmanagement

Als Vorgehensweise wählten wir Scrum+, welches an der HSR unterrichtet wurde. Es verbindet die Agilität von Scrum[16] mit den Meilensteinen von RUP [17]. Wir entschieden uns als Organisationstool Azure DevOps [17] einzusetzen. Dieses beinhaltet neben sogenannten Boards, welche die Organisation von Tasks und Arbeitspaketen vereinfachen, auch Pipelines für CI/CD. Diese verwendeten wir für automatische Builds unserer Dokumentation, wie auch für die Builds und das Publishing des Docker-Images unseres Superset-Forks. Des Weiteren war das Deployment des Images, auf den von der HSR zur Verfügung gestellten Linux-Server, über Azure DevOps geregelt. Zur Verwaltung unseres Superset Repository verwendeten wir GitHub[18]. GitHub bot sich deshalb an, da Travis CI [19], welches von Superset für das Testing und den Build verwendet wird, nur in Kombination mit GitHub funktioniert. Wir wollten die Entwicklungsumgebung möglichst nahe an den Superset-Technologien halten.

5.2 Projektaufteilung

Superset arbeitet mit sogenannten SIPs (Superset Improvement Proposals) [20]. Diese dienen dazu, neue Funktionalitäten zu umschreiben sowie zur Standardisierung der Kommunikation zwischen den Entscheidungsträgern von Superset und den Entwicklern. Da unsere Arbeit nicht aus einem einzigen grossen Arbeitspaket bestand, erstellten wir mehrere SIPs, damit die einzelnen Features auseinandergehalten werden konnten. Dabei beschränkten wir uns auf die zwei Hauptanforderungen:

1. CSV-Importer vereinfachen
2. Geokodierung von Adressdaten

Als Reserve planten wir, die Transformation der Geometrie-Datentypen (POINT, POLYLINE, POLYGON) auf kompatible Typen für die Superset Diagramme anzugehen.

[17] Azure DevOps: https://azure.microsoft.com/en-us/services/devops/
[18] GitHub: https://github.com/
[19] Travis CI https://travis-ci.org/
[20] SIP: https://github.com/apache/incubator-superset/issues/5602

6 Anforderungsspezifikation

Hier werden die Anforderungen an unsere Implementationen festgehalten.

6.1 Use Cases

Im nachfolgenden Diagramm werden die von uns definierten Use-Cases aufgelistet. Dabei war “CSV in bestehende Datenbank importieren“ bereits möglich. Da dies im Umfang des Use-Cases “CSV in neue Datenbank importieren“ jedoch umgebaut wurde, erwähnten wir diese Anforderung nochmals. Die unteren Use-Cases (mit transparenter/hellerer Schrift) dienten als Reserve.

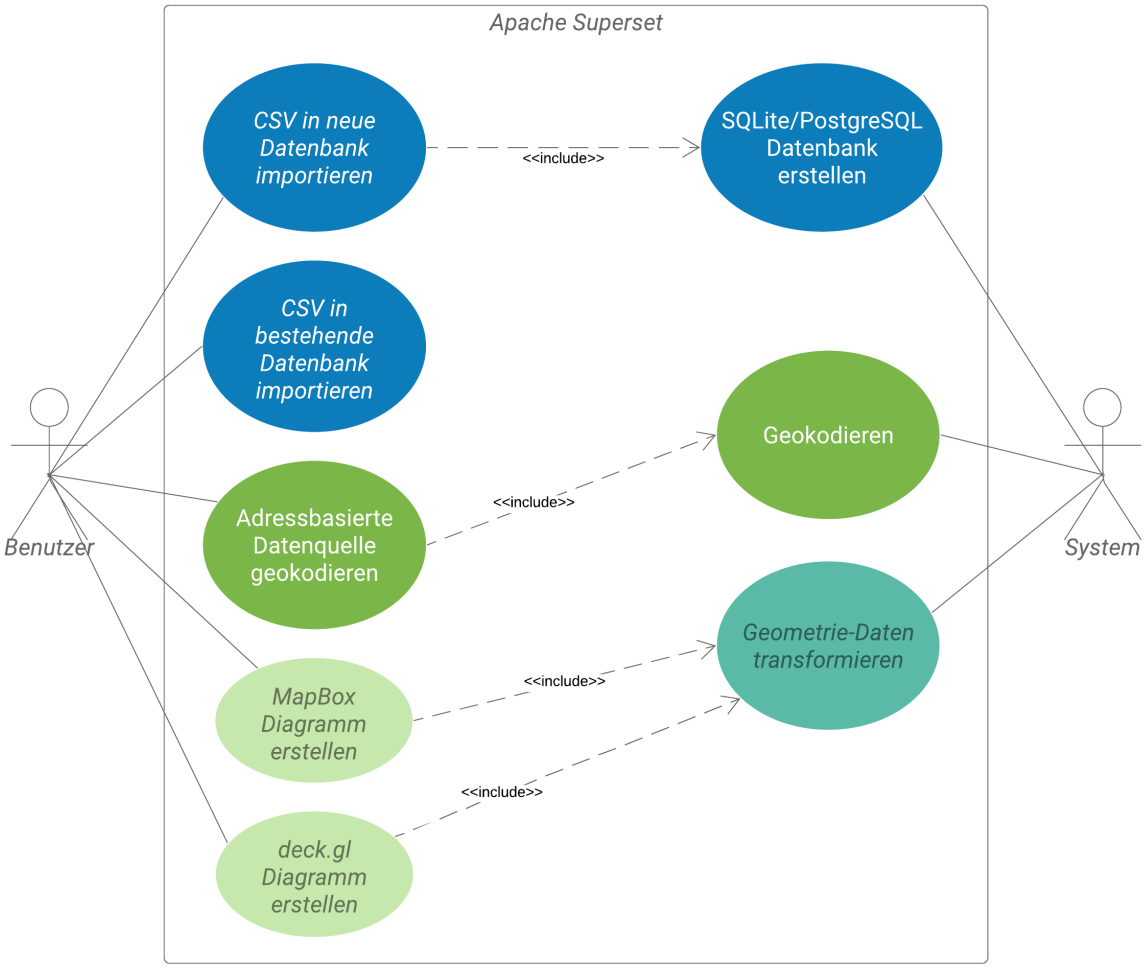


Abbildung 14: Use Case Diagramm

6.1.1 Aktoren

In der Tabelle 15 befinden sich die Aktoren, welche durch die Lösung der Problemstellungen betroffen waren. Deren Interessen werden kurz erläutert.

Aktor	Beschreibung und Interesse
Benutzer	Als Benutzer gelten diejenigen Anwender mit der Rolle Alpha oder Admin. Sie können Tabellen (Datenquellen), Dashboards und Grafiken verwalten. Dazu bewirtschaften sie Superset mit Daten und visualisieren diese für sich und andere Anwender.
System	Das System bietet verschiedene Diagramme an, mit denen Daten visualisiert werden können. Zusätzlich hat das System verschiedene Funktionalitäten zum Import und der Kontrolle von Daten, damit diese dann in den entsprechenden Diagrammen dargestellt werden können.

Abbildung 15: Use-Case Aktoren

6.1.2 CSV in neue Datenbank importieren

Aktoren: Benutzer (Admin/Alpha)

Der Benutzer kann Tabellen erstellen, auf denen dann die Grafiken und Dashboards aufbauen. Superset nennt diese auch Datasource. Er kann als Quelle anstatt eines Datenbankschemas eine CSV-Datei verwenden. Dazu lädt er die Datei hoch und konfiguriert dabei die Tabelle. Er wählt aus, dass er eine Datenbank erstellen möchte. Danach gibt er den Namen der Datenbank und der Tabelle, die Art von Datenbank, das CSV-Trennzeichen sowie weitere mögliche Einstellungen an. Die Datei zieht er per “Drag and Drop“ in die Maske oder wählt sie über den File-Upload aus, wodurch sie für das Hochladen zwischengespeichert wird.

6.1.3 SQLite/PostgreSQL Datenbank erstellen

Aktoren: System

Wird beim Hochladen einer CSV-Datei angegeben, dass diese in eine neue Datenbank importiert werden soll, wird vom System im Hintergrund automatisch eine neue Datenbank der ausgewählten Art angelegt und für den Upload verwendet.

6.1.4 CSV in bestehende Datenbank importieren

Aktoren: Benutzer (Admin/Alpha)

Der Benutzer kann beim Hochladen einer CSV-Datei auch angeben, dass er die Daten in eine bestehende Datenbank integrieren möchte. Er gibt dazu die gleichen Daten wie beim Importieren in eine neue Datenbank an, wählt aber dann eine bestehende Datenbank aus.

6.1.5 Adressbasierte Datenquelle geokodieren

Aktoren: Benutzer (Admin)

Der Benutzer kann veranlassen, dass die adressbasierten Daten seiner Datenquelle geokodiert werden. Dazu muss er die relevanten geographischen Spalten, sowie den zu verwendenden Geokodierungs-Service angeben. Dadurch kann er Diagramme wie MapBox verwenden, um zum Beispiel Städte auf der Karte zu markieren.

6.1.6 Geokodieren

Aktoren: System

Das System geokodiert die Daten in den geographischen Spalten, die der Benutzer zuvor angegeben hat, zu Koordinaten. Die Adressen müssen so weit als möglich eindeutig sein, so dass sie geokodiert werden können. Ansonsten wird das erste zurückgelieferte Resultat verwendet. Am Schluss des Vorgangs wird eine Auflistung der erhaltenen Genauigkeit der gefundenen Koordinaten angezeigt.
Doppelte Spaltennamen werden erkannt und dem Benutzer gemeldet. Dann kann dieser die Namen der Spalten ändern oder auswählen, dass die existierenden Daten überschrieben werden sollen.

6.1.7 MapBox Diagramm erstellen

Aktoren: Benutzer (Admin/Alpha)

Der Benutzer kann geographische Daten mit dem Diagramm MapBox visuell auf einer Karte darstellen. Dazu kann er die Längen- und Breitengrade oder geometrische Formen wie Punkte, Polylines oder Polygone angeben. Die Daten werden danach als Punkte auf einer Karte dargestellt.

6.1.8 deck.gl Diagramm erstellen

Aktoren: Benutzer (Admin/Alpha)

Der Benutzer kann geographische Daten mit einem der Diagramme von deck.gl visuell auf einer Karte darstellen. Dazu kann er die Längen- und Breitengrade oder geometrische Formen wie Punkte, Polylines oder Polygone angeben. Die Daten werden danach als Geometrien auf einer Karte dargestellt.

6.1.9 Geometrie-Daten transformieren

Aktoren: System

Verlangt ein Diagramm Koordinaten, erhält jedoch stattdessen geometrische Daten wie Punkte, Polygone etc., transformiert sie das System im Hintergrund zu Koordinaten. Werden geometrische Daten gefordert, transformiert das System die Längen- und Breitengrade im Hintergrund zu diesen Datentypen um.

6.2 Nicht funktionale Anforderungen

Für die nicht funktionalen Anforderungen verwenden wir die FURPS Kategorisierung.

FURPS:

- Functionality (Funktionalität)
- Usability (Benutzbarkeit)
- Reliability (Verlässlichkeit)
- Performance (Performanz)
- Supportability (Wartbarkeit)

6.2.1 Funktionalität

Wie bereits oben erwähnt sollen folgende Funktionalitäten umgesetzt werden: CSV-Dateien können in Tabellen in Superset importiert werden, ohne dass bereits eine Datenbank besteht. Ausserdem können Adressen automatisch in Koordinaten transformiert werden. Dazu können spezielle Geometrie-Datentypen (z.B. POINT) automatisch in Längen- und Breitengrade transformiert werden, welche von den Visualisierungstools verstanden werden.

6.2.2 Benutzbarkeit

Der CSV-Importer soll benutzerfreundlich gestaltet sein, so dass ein Benutzer innerhalb von 30s die gewünschte Datei importieren kann. Des Weiteren benötigt der Import einer CSV-Datei in eine bestehende Datenbank gleich viel oder weniger Klicks wie beim bisherigen Importer. Ausserdem kann ein Benutzer den CSV-Importer vornehmen, ohne irgendwelche optionalen Felder ausfüllen zu müssen. Jedoch findet er diese Optionen bei Bedarf und kann sie gegebenenfalls ausfüllen. Fehlerhafte Eingaben des Benutzers für die Geokodierung (zum Beispiel nicht vorhandene Spalten) werden verhindert. Weiter kann er wählen, ob bereits vorhandene Daten in den gewünschten geographischen Spalten überschrieben werden sollen. Überdies wird ihm der Fortschritt des Geokodierungsvorganges angezeigt. Der Geokodierungsvorgang kann, wenn gewünscht, unterbrochen werden. Das Überschreiben durch doppelte Namen der Koordinaten-Spalten wird verhindert, ausser dies wird vom Benutzer explizit ausgewählt.

6.2.3 Verlässlichkeit

Die Software soll nicht abstürzen, falls die Daten in der importierten CSV-Datei nicht übertragen werden konnten. Stattdessen wird eine informative Fehlermeldung ausgegeben. Ferner funktioniert der Upload einer korrekten CSV-Datei immer. Im Falle von Inkonsistenzen (z.B. bei verschiedenen Trennzeichen), wird der Vorgang abgebrochen, ohne den Programmablauf zu beeinträchtigen. Ausserdem werden Adressen, die nicht geokodiert werden können, weil der Geokodierungs-Service keine passenden Koordinaten fand, nicht transformiert. Bei uneindeutigen Adressen, liefert der Service das beste Ergebnis zurück. Zudem wird bei Problemen mit der Geokodierungs-API (fehlender API-Key, Service down o.ä.) eine Fehlermeldung angezeigt und der Vorgang abgebrochen. Wird während des Vorgangs das Anfragelimit für den API-Key erreicht, wird der Vorgang ordentlich abgebrochen und die bereits geokodierten Daten in die Tabelle geschrieben, falls der Benutzer diese Option angewählt hat.

6.2.4 Performanz

CSV-Dateien mit einer kleinen Anzahl an Datensätzen (bis zu 100'000 Datensätze) sollen im Durchschnitt in unter 10 Sekunden importiert werden. Hingegen CSV-Dateien mit einer grossen Anzahl an Datensätzen (über 1 Million Datensätze) sollen durchschnittlich in unter 5 Minuten importiert sein. Für das Geokodieren muss mit bis zu 2 Sekunden pro Datensatz gerechnet werden, da diese Operation sehr teuer ist und viele APIs die Anzahl Datensätze pro Sekunde beschränken. Diese Zeit soll jedoch nicht überschritten werden.

6.2.5 Wartbarkeit

Die Erweiterung der unterstützten Geokodierungs-APIs ist ohne grosse Umstellungen im Code möglich.

7 Analyse

In der Analyse begründen wir, warum wir uns für welche Technologien und Vorgehen entschieden haben. Hier behandeln wir folgende Technologien, die im Superset eingesetzt werden:

Backend

Abbildung 16: Technologien, welche im Backend eingesetzt werden

Frontend

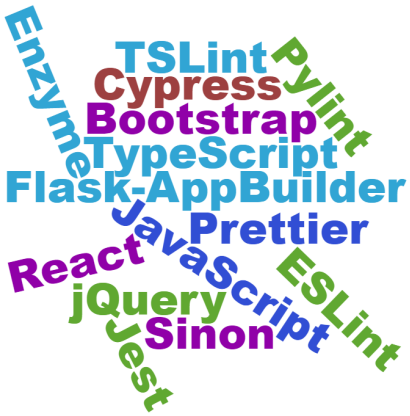


Abbildung 17: Technologien, welche im Frontend eingesetzt werden

Nachfolgend gehen wir auf diese Sprachen, Tools und Frameworks ein und erläutern den Einsatz unserer Entwicklungstools.

7.1 Backend-Programmiersprache

Apache Superset setzt Python als Backend-Programmiersprache ein. Es unterstützt die Versionen von 2.* bis 3.6. Ausserdem sollten ihre Tests auch mit 3.7 durchführbar sein. [21]

Da Superset nicht garantieren kann, dass das Projekt vollständig mit Python 3.7 kompatibel ist, wählten wir einen tieferen Release. Wir haben uns für Python 3.6.8 entschieden, weil dieser Release der letzte gewartete Feature-Release für 3.6 ist. [22]

7.2 Schnittstellen-Spezifikation

In Superset findet das Web Framework Flask-AppBuilder Verwendung. Es setzt auf Flask auf und ist zuständig für Themen wie Sicherheit, Charts, Diagramme, Schnittstellengenerierung und Weiteres. [23] Superset verwendet von diesen Themen lediglich die Sicherheit, Übersetzungen, Charts und Formulare. Anstelle von der Klasse ModelRestApi, welche die Schnittstellenbeschreibung generieren würde, erben die View-Klassen (Controller) von BaseView. [24] Mit Hilfe dieser Klasse werden die Zugriffsberechtigungen auf den Routing-Methoden definiert. Dadurch kann die Sicherheit auf die einzelnen Views heruntergebrochen werden. [25]

Eine Schnittstellen-Spezifikation für Superset gibt es nicht. Da wir uns an ihre Standards halten wollten, haben wir im Code ebenfalls keine erstellt.

7.3 Frontend-Frameworks

Auch die Frontend Entwicklung wird von Superset vorgegeben. Sehr viele Objekte wie die Diagramme und Formulare kommen aus dem Framework Flask-AppBuilder. Als Entwickler muss man diese nur noch konfigurieren, so dass sie die richtigen Daten und Felder anzeigen.

Superset verwendet zusätzlich die neuste Version (16.9.0) des Frameworks „React mit Redux“. Jedoch sind nur wenige Funktionalitäten von Superset in React programmiert. Hauptsächlich Aktionen wie etwas Neues hinzufügen, filtern, etc. Die Klassen und Funktionen für die React-Komponenten werden mit JavaScript erstellt. Einzelne Funktionen werden auch in TypeScript umgesetzt. Das Design wird anhand von Bootstrap festgelegt und der Code mit Prettier formatiert.

7.4 Erstellung von SQLite- und PostgreSQL-Datenbanken

Für die Funktionalität des CSV-Importers in eine neue Datenbank mussten wir innerhalb des Backends eine Datenbank erstellen können. In Superset wird bereits die Bibliothek SQLAlchemy verwendet, welche dieses Bedürfnis vollumfänglich abdeckt und nicht noch zusätzlich installiert werden muss. Da wir zusätzliche externe Abhängigkeiten möglichst vermeiden wollten, auch um die Annahme der Pull-Requests nicht unnötig zu erschweren, haben wir uns dazu entschieden, die bereits benutzte Bibliothek einzusetzen und keine Externe zu verwenden.

[21] Superset Python: https://superset.incubator.apache.org/installation.html#getting-started
[22] Python Release: https://www.python.org/downloads/release/python-368/
[23] Flask-AppBuilder: https://flask-appbuilder.readthedocs.io/en/latest/intro.html
[24] ModelRestApi: https://flask-appbuilder.readthedocs.io/en/latest/views.html#baseview
[25] BaseView: https://flask-appbuilder.readthedocs.io/en/latest/rest_api.html#model-rest-api

7.5 Geokodierung Python API

Für Python gibt es sehr viele APIs und Bibliotheken, die ermöglichen, Adressen in Koordinaten und wieder zurück zu transformieren. Einige davon haben wir analysiert und bewertet: [26] [27] [28] [29] [30] [31]

- **Adressraumgrösse**: Sind die Adressen zur Geokodierung auf ein Land beschränkt? Welche Adressen werden geokodiert?
- **Anzahl Services**: Wie viele Geokodierungs-Services (Provider) werden durch die API oder Bibliothek unterstützt?
- **Kosten**: Wie viel kostet die Benutzung der API bzw. die Bibliothek?
- **Letzter Release**: Wann war der letzte Release?

	Adressraumgrösse	Anzahl Services	Kosten	Letzter Release
GeoPy	alle, weltweit	25	Service abhängig	Juni 2019
Geocoder	alle, weltweit	24	Service abhängig	September 2019
OpenCage	alle, weltweit	1	50$/Monat ab 10'000 Anfragen	Juni 2019
pygeocoder	alle, weltweit	1	0.005$/Anfrage	Juni 2014
MapTiler	alle, weltweit	1	gratis bis 100'000 Anfragen/Monat	September 2019
Google	alle, weltweit	1	0.004$/Anfrage	November 2019

Zu Beginn wollten wir GeoPy als Geokodierungs-Bibliothek verwenden, da sie sehr viele der bekannten Geokodierungs-APIs bedienen kann. Wir stellten aber fest, dass viele der von GeoPy unterstützten Geokodierungs-APIs in ihren Terms of Service das Abspeichern der Koordinaten nicht erlauben. Wir haben uns deshalb dafür entschieden, eine eigene Geokodierungs-Routine zu implementieren, die man flexibel erweitern kann. Für die Geokodierungs-APIs entschieden wir uns letztlich für MapTiler und Google, da beide in ihren Terms of Service die Benutzung der Koordinaten nicht weiter einschränkten. Zusätzlich schneidet die Geokodierungs-API von Google, im Vergleich mit anderen, äusserst gut ab, womit wir auch bestmögliche Resultate garantieren können.

7.6 Guideline und Codequalität

Für lesbaren und wartbaren Code setzt Superset sogenannte Linter ein. Python-Code verwendet mypy, isort und Black, um zu prüfen, ob die Dateien Fehler aufweisen und den Python Style Guide (PEP8[32]) einhalten. Ausserdem wird die Komplexität der Klassen und Funktionen mit der McCabe-Metrik geprüft. Pylint wird ebenfalls eingesetzt.

Im Frontend wird der JavaScript Code mit ESLint geprüft, während für TypeScript TSLint verwendet wird. Beide prüfen auf markante Fehler und weisen auf mögliche Unschönheiten und Probleme hin.

[26] GeoPy: https://geopy.readthedocs.io/en/latest/
[27] Geocoder: https://geocoder.readthedocs.io/index.html
[28] OpenCage: https://opencagedata.com
[29] pygeocoder: https://bitbucket.org/xster/pygeocoder/wiki/Home
[30] MapTiler Geocoding: https://www.maptiler.com/blog/2019/09/announcing-maptiler-cloud-geocoding-api.html
[31] Google Preise: https://developers.google.com/maps/documentation/geocoding/usage-and-billing
[32] PEP8: https://www.python.org/dev/peps/pep-0008/

7.7 Test Tools

Superset setzt für Tests in Python das Tool tox ein. tox ist ein Kommandozeilen-Programm, welches die Installation mit verschiedenen Python-Environments testet, die Unit-Tests automatisiert und standardisiert, und als Continous Integeration Server interagieren kann. [33] Die Tests werden im tox mit dem Framework pytest durchgeführt. Die Unit-Tests werden in Python geschrieben und ausgeführt. [34]

Das Frontend wird mit Jest, Enzyme und Sinon getestet. Mit diesen lassen sich Testfälle modellieren und ausführen. Jest testet JavaScript, TypeScript und React, lässt sich mocken und ist sehr gut dokumentiert. In Superset wird es vor allem für die Test Coverage eingesetzt. [35] Enzyme wird für die DOM Manipulationen und Traversierungen in React verwendet. [36] Sinon ist ein Mocking-Framework für JavaScript. [37]

Integrationstests werden mit Cypress durchgeführt. Cypress simuliert die Bedienung durch einen Menschen auf der Webseite im Browser und führt so die Testfälle aus. Dabei werden die Durchführungszeiten, Netzwerkaktivitäten, Screenshots und Videos festgehalten, um die Tests besser analysieren zu können. Ausserdem lässt sich Cypress debuggen und mocken. [38]

7.8 Installationsumgebung

Windows wird von Superset grundsätzlich nicht unterstützt. Dennoch gibt es einige Hilfestellungen auf Stack Overflow und anderen Plattformen zur Installation von Superset auf einem Windows Rechner. Schlussendlich ist Superset jedoch nicht für Windows ausgelegt, was zu diversen Problemen und aufwendigen Workarounds führt, um eine Entwicklungsumgebung zu erhalten, welche dennoch nicht vollständig funktioniert. [39]

Zuerst hatten wir Superset auf einem Windows-Subsystem für Linux aufgesetzt. [40] Dies ermöglichte uns eine einfache und unkomplizierte Installation nach der Anleitung von Superset und vermied komplizierte und unschöne Workarounds. Als wir dann aber verschiedene Probleme mit dem Zugriff von PyCharm auf das Repository im Subsystem hatten und wir nicht mehr builden oder testen konnten, wechselten wir auf eine Unix VM mit Ubuntu. Auf dieser VM richteten wir unsere gesamte Entwicklungsumgebung ein. Dazu gehören die Tools PyCharm, Visual Studio Code und der Chrome Browser für das React-Debugging. Dadurch läuft alles auf ein und demselben System, welches den Anforderungen für Superset entspricht.

[33] tox: https://tox.readthedocs.io/en/latest/index.html
[34] pytest: https://docs.pytest.org/en/latest/
[35] Jest: https://jestjs.io/
[36] Enzyme: https://airbnb.io/enzyme/
[37] Sinon: https://sinonjs.org/
[38] Cypress: https://docs.cypress.io/guides/overview/why-cypress.html
[39] Unterstütze Betriebssysteme: https://superset.incubator.apache.org/installation.html#os-dependencies
[40] Windows-Subsystem für Linux: https://docs.microsoft.com/de-de/windows/wsl/about

7.9 Travis CI und GitHub

In den Artefakten von Superset findet sich eine *.travis.yml* Datei. Ihre Builds lassen sie auf dem Travis CI Service durchführen. Dabei liefert die vorhin erwähnte Datei, die Konfiguration für diese Builds. Bei einem Build werden alle Testfälle durchgeführt, was ungefähr eine Stunde dauert.

Um sicherzustellen, dass unsere Arbeit korrekt gebuildet wird und alle Unit-Tests erfolgreich durchlaufen, hatten wir den selben Service eingesetzt. Travis buildet nur Repositories, die sich auf GitHub befinden. Deshalb entschieden wir uns, den Code über GitHub zu verwalten. Da das Ausführen aller Testfälle ungefähr eine Stunde dauerte, beschlossen wir, auf das Ausführen der Tests in der lokalen Umgebung so weit als möglich zu verzichten. Diese Arbeit wollten wir dem Build Service überlassen. Lediglich die eigenen Testfälle führten wir jeweils einzeln lokal durch. Mit jedem Pull Request in den master wurden alle Tests durchgeführt und auf ihren Erfolg geprüft. Auf dem Build Server werden die Tests zudem parallelisiert durchgeführt, wodurch diese auch schneller abgeschlossen sind.

7.10 Docker

Um unsere entwickelten Arbeitspakete in einem Release unseren Stakeholdern vorführen zu können, verwendeten wir Docker.

Superset ist so konzipiert, dass es hoch verfügbar und skalierbar ist. Es läuft sowohl in der Cloud als auch auf Containern. Die Installation von Superset in einem Docker Container ist sehr einfach und gut nachvollziehbar. Durch die Verwendung von Docker ist die Reproduzierbarkeit gewährleistet und das Laufzeitverhalten auf allen Systemen gleich. Ausserdem müssen auf dem Betriebssystem des Servers keine Abhängigkeiten, wie beispielsweise pip, installiert werden.

7.11 Virtueller Linux Server

Mit jedem Pull Request in den master, wurde bei uns automatisch das Deployment auf einem Docker Container ausgelöst. Da die Linux Container des Dockers in Windows (LCOW) frühstens in der Version 1709 Preview funktionsfähig sind und die HSR lediglich virtuelle Windows Server der Version 1604 anbietet, installierten wir die Docker-Engine auf einem virtuellen Linux Server. [41]

[41] LCOW: https://docs.microsoft.com/en-us/virtualization/windowscontainers/deploy-containers/linux-containers

7.12 Sequenzdiagramme

Um uns einen Überblick zu verschaffen, wie wir unsere Features implementieren könnten, bezogen auf die Interaktionen zwischen Front- und Backend, erstellten wir Sequenzdiagramme. Diese sollten uns dabei helfen, uns Gedanken zu machen, wo welche Funktionalitäten Sinn machen. Des Weiteren sind diese Diagramme nützlich als übersichtliche Darstellung, wie wir die Implementation dieser Features im Groben geplant hatten.

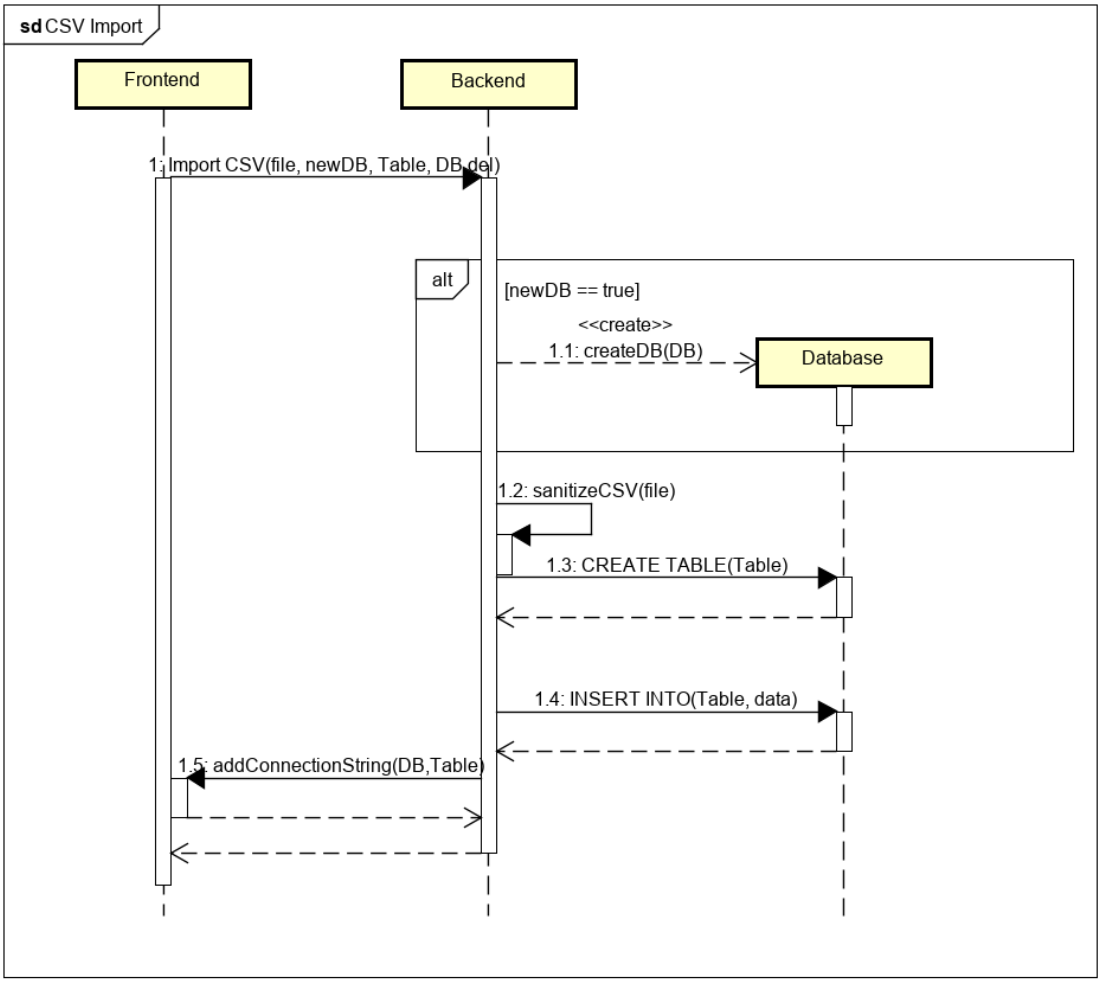


Abbildung 18: Sequenzdiagramm für den Import eines CSVs mit Fallunterscheidung ob neue DB

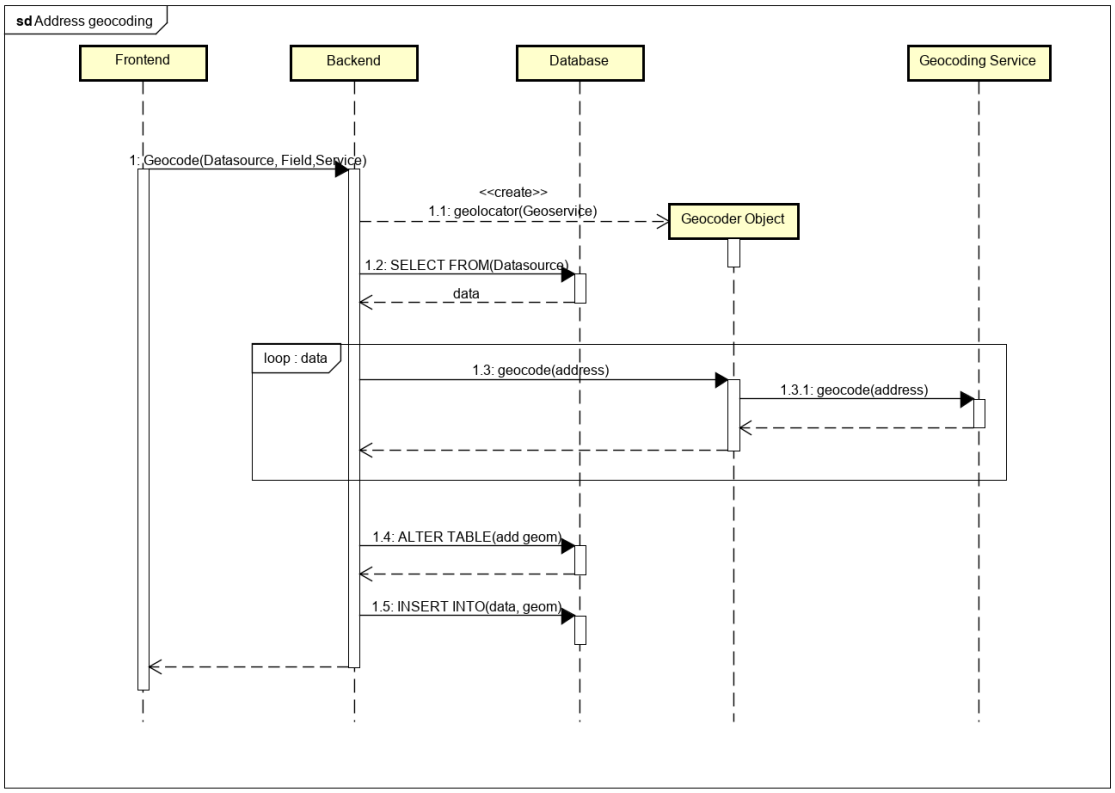


Abbildung 19: Sequenzdiagramm für die Geokodierung der Adressen

Das nachfolgende Diagramm zeigt die Funktionalität, welche wir als Reserve eingeplant hatten.

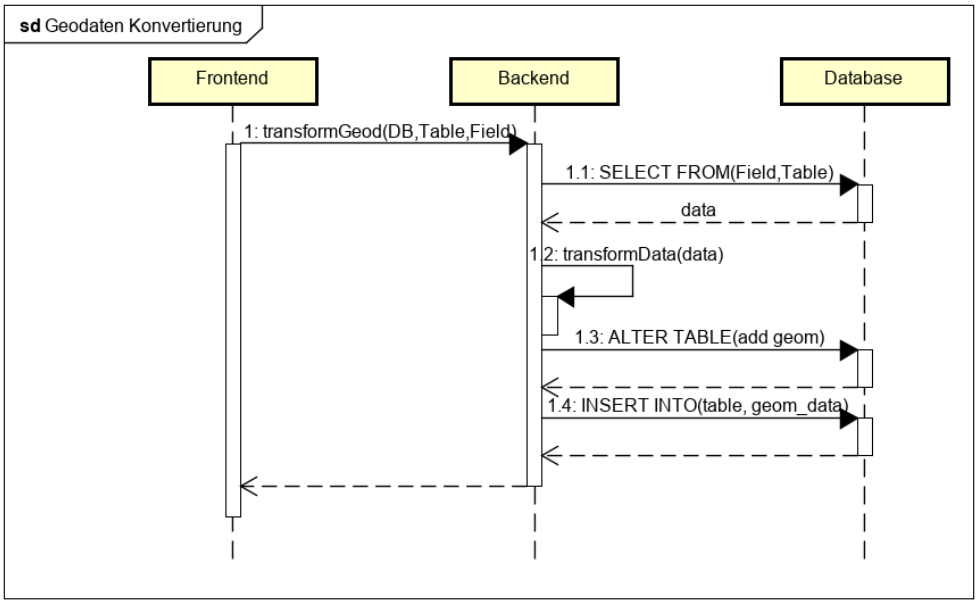


Abbildung 20: Sequenzdiagramm für die Transformation der Geometrie-Daten (z.B. von POINT zu Längen- und Breitengraden)

7.13 GUI-Entwürfe

Nachfolgend werden die GUI-Entwürfe gezeigt und beschrieben. Diese wurden mit Mockflow [42] skizziert. Ziel der Entwürfe war es, eine Übersicht über die Umsetzung der geplanten Features zu geben und wie deren Benutzung aussehen könnte.

7.13.1 CSV-Importer

Das GUI lehnt sich am bereits vorhandenen CSV-Importer an. Es werden allerdings nur die allerwichtigsten Felder angezeigt.

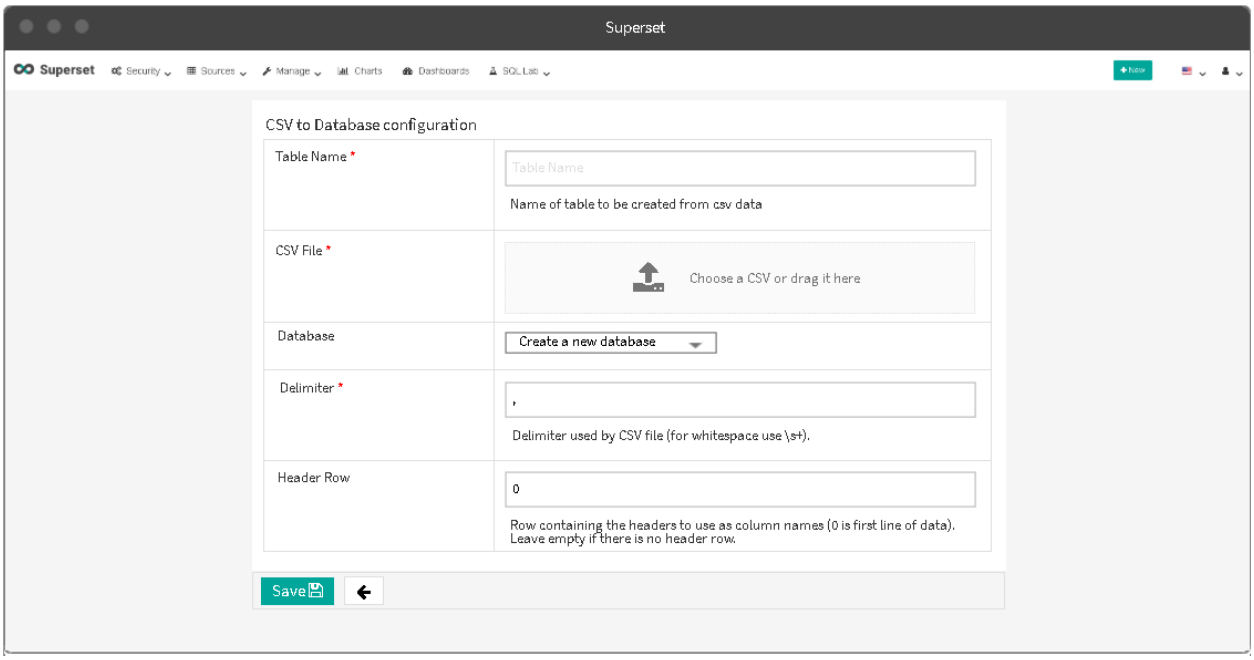


Abbildung 21: GUI-Entwurf des CSV-Importers

Table Name Der Benutzer kann den Namen der Tabelle eintragen, die dann als Datenquelle in Superset verfügbar ist. Es handelt sich hierbei um ein Pflichtfeld, das zwingend angegeben werden muss.

CSV File Im grauen Bereich kann der Benutzer die CSV-Datei auswählen. Neu kann er dies auch mit Drag and Drop machen. Klickt er in den grauen Bereich öffnet sich das Datei-Suchfenster. Die Anzeige verändert sich leicht, wenn der Benutzer eine Datei ausgewählt hat, um ihm zu signalisieren, dass die Auswahl erfolgreich war. Er kann jederzeit eine andere Datei auswählen, was in diesem GUI-Entwurf allerdings nicht sichtbar ist.

Database Bei diesem Selektionsfeld kann der Benutzer wählen, ob die Tabelle in eine neue Datenbank oder in eine bereits bestehende eingefügt werden soll. Die Standardauswahl ist, die Tabelle in eine neue Datenbank hochzuladen.

Delimiter In diesem Feld kann der Benutzer angeben, welches Trennzeichen in der CSV-Datei verwendet wurde. Standardmässig wird das Komma verwendet.

Header Row Die Angabe in welcher Zeile die Header stehen, kann in diesem Feld getätigt werden. Standardmässig wird die erste Zeile mit Index 0 verwendet.

[42] Mockflow: https://mockflow.com

7.13.2 Geokodierung

Für die Geokodierung wird ein eigenes Formular angeboten, bei dem man die nötigen Einstellungen treffen kann, damit die Geokodierung dann auch erfolgreich durchgeführt werden kann.

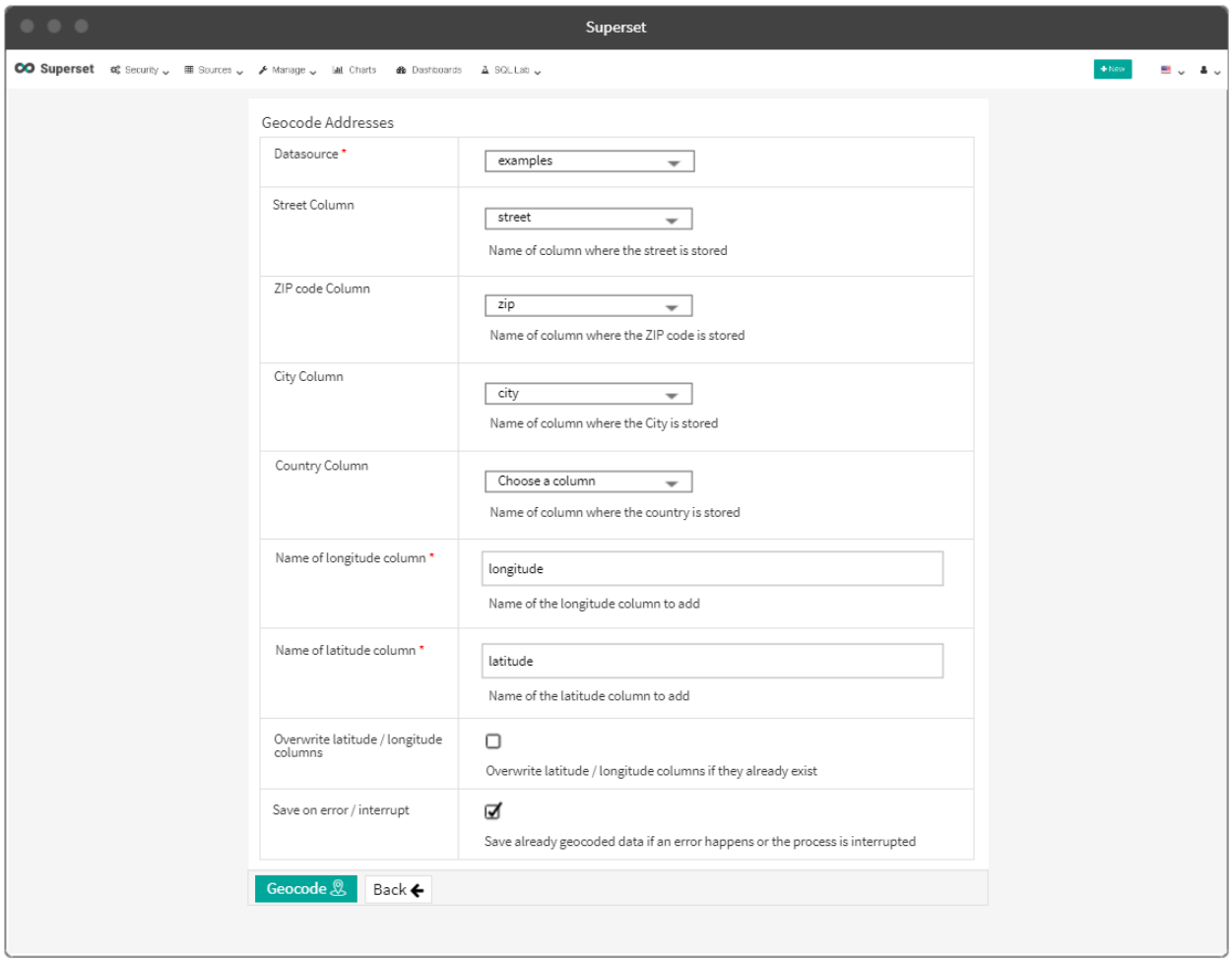


Abbildung 22: GUI-Entwurf des Formulars für die Geokodierung

Datasource In diesem Feld muss der Benutzer angeben, welche Tabelle er geokodieren möchte. Es handelt sich hierbei um ein Pflichtfeld.

Street Column Hier kann der Benutzer den Spaltennamen der Spalte, in der die Strasse abgespeichert ist, angeben.

City Column Der Spaltenname der Spalte, in der die Stadt abgespeichert ist, kann vom Benutzer in dieser Zeile eingegeben werden.

Country Column Will der Benutzer für das Geokodieren Länderangaben verwenden, so kann er in dieser Zeile den Spaltenname der Spalte angeben, in der das Land abgespeichert ist.

Name of longitude column Der Benutzer muss hier den Namen der neu einzufügenden Spalte für den Längengrad angeben. Es handelt sich hierbei um ein Pflichtfeld mit dem automatischen Vorschlag lon.

Name of latitude column Bei der letzten Zeile muss der Benutzer den Namen der neu einzufügenden Spalte für den Breitengrad angeben. Es handelt sich hierbei um ein Pflichtfeld mit dem automatischen Vorschlag lat.

Nicht sichtbar Falls der Benutzer weder Strasse, Stadt noch Land angibt und versucht, die Geokodierung anzustossen, wird er über eine Fehlermeldung informiert, dass mindestens eines dieser Felder angegeben werden muss.

7.14 Benutzerrollen und Rechte im Superset

Die Sicherheit wird mit dem Framework Flask-AppBuilder umgesetzt. Dabei wird automatisch für jede View- und jede API-Klasse ein eigenes Recht erstellt.

7.14.1 Rollen

Superset erstellt zu jeder Installation Benutzerrollen. Es wird empfohlen, diese nicht anzupassen. Stattdessen soll man neue Rollen anlegen und diese mit den vordefinierten kombinieren. Im Folgenden werden die Rollen kurz erläutert:

Admin Die Rolle Admin besitzt sämtliche Rechte und kann dadurch alles. Sie definiert die Administratoren der Superset-Instanz.

Alpha Mit der Rolle Alpha hat der Benutzer ähnliche Rechte wie der Administrator. Er kann jedoch anderen Benutzern keine Rechte erteilen. Der Benutzer erhält Zugriff auf sämtliche Datenbanken sowie Datenquellen und kann diese dementsprechend bearbeiten, erstellen und löschen. Allerdings sieht er nur seine eigenen Dashboards und Diagramme sowie diejenigen, welche public sind. Diese kann er auch ändern sowie Neue erstellen.

Gamma Ein Benutzer mit der Rolle Gamma sieht standardmässig keine Dashboards bzw. Diagramme. Diese Rechte müssen dem Benutzer separat zugewiesen werden.

sql_lab Mit dieser Rolle kann der Benutzer auf das SQL Lab zugreifen und Abfragen erstellen, bzw. anpassen.

Public Einem Benutzer, welcher sich nicht eingeloggt hat, wird die Rolle Public zugewiesen. Er sieht wie der Gamma nichts, solange man ihn nicht auf Datenquellen, Dashboards und Diagramme freischaltet.

Granter Die Granter-Rolle ermöglicht es einem Benutzer, die Rechte in den Rollen anzupassen.

Pro Gruppe, die auf die gleichen Datenbanken oder Datenquellen zugreifen müssen, sollte eine eigene Rolle definiert werden. Es empfiehlt sich den Zugriff auf die Datenbank, statt die einzelnen Datenquellen, zu definieren.

7.14.2 Rechte

In Superset gibt es 93 generierte Rechte. Die Rechte werden teilweise vererbt.

Erhält ein Benutzer zum Beispiel Zugriff auf eine Datenbank, so erhält er automatisch auch auf deren Datenquellen, Dashboards und Charts Zugriff. Diese sieht er aber nur, wenn er direkt über die URL darauf zugreift, in der Liste tauchen sie nicht auf. Ein anderes Beispiel ist, dass wenn man einem Benutzer ein Dashboard freigibt, so werden ihm auch dessen Charts freigegeben.

Um in Superset die Charts eines Dashboards, von welchem die URL bekannt ist, sehen zu können, werden fünf Rechte benötigt. Der Benutzer kann über die URL die Diagramme sehen, welche mit ihm geteilt wurden. Anderweitig kann er nicht auf das Dashboard zugreifen. Nachfolgend das Minimum an Rechten, die ein Benutzer dafür besitzen muss:

datasource access on [*xxx*].[*yyy*] Dieses Recht ermöglicht es, auf die Datenquelle *yyy* der Datenbank *xxx* zuzugreifen. Ausserdem erhält er Zugriff auf die Dashboards dieser Datenquelle und auf die Diagramme in diesen Dashboards.

can dashboard on Superset Damit der Benutzer das Dashboard sehen kann, benötigt er den Zugriff auf die Dashboards in Superset.

can explore json on Superset Die Diagramme arbeiten mit JSON. Ohne dieses Recht sieht er den Inhalt der Diagramme nicht.

can favstar on Superset In Superset werden die Daten aus dem Cache geladen. Damit auf diesen zugegriffen werden kann, benötigt man dieses Recht.

can list on CssTemplateAsyncModelView Dieses Recht wird benötigt, damit keine Fehlermeldung bezüglich der CSS-Templates angezeigt wird.

Soll der Benutzer die freigegebenen Dashboards in seinem Dashboard aufgelistet sehen können, so werden zwei weitere allgemeine Rechte benötigt:

can list on DashboardModelViewAsync Mit diesem Recht sieht der Benutzer auf seinem Dashboard alle für ihn freigegebenen Dashboards.

menu access on Dashboards Dieses Recht blendet im Menu den Punkt “Dashboard“ ein. Der Benutzer kann dadurch von jeder Seite zurück auf das Dashboard navigieren.

Alle anderen “Views“ können auf die gleiche Weise wie folgt freigegeben werden:

menu access on *xxx* Dieses Recht erlaubt dem Benutzer im Menu den Punkt *xxx* zu sehen und auszuwählen. Die darunterliegenden Punkte sind einzeln, auf dieselbe Art freizugeben.

can add on *yyy* Der Benutzer kann neue Objekte von *yyy* anlegen.

can edit on *yyy* Mit diesem Recht kann er die Objekte von *yyy* bearbeiten.

can list on *yyy* Alle Objekte von *yyy* in Superset werden mit diesem Recht angezeigt.

Zwischendurch gibt es weitere Rechte, die zusätzlich nötig sind, wie zum Beispiel “can new on Dashboard“. Für diese gibt es allerdings keine allgemeinen Regeln.

8 Architektur

In diesem Kapitel werden die Strukturierung der Dateien und Codefragmente erläutert.

8.1 Backend

Wir wollten eine moderne Umsetzung unserer Backend-Komponenten realisieren und entschieden uns deshalb dafür, das Backend RESTful zu implementieren. Das Backend setzt sich aus verschiedenen REST-Schnittstellen, den eigentlichen APIs, zusammen. Da Superset keine klare Schichtentrennung hat, haben wir versucht, für unsere Komponenten eine solche einzuführen. Die APIs greifen, sofern sinnvoll, über einen Business-Layer (eigentliche Logik) mit Hilfe von SQLAlchemy (Data-Access-Layer) auf die Daten zu. Für die Datenübertragung an den Client wird durchgehend das JSON-Format verwendet.

8.1.1 CSV-Importer

Einen grossen Teil der Logik des ursprünglichen CSV-Importers konnten wir wiederverwenden. Der wesentliche Unterschied zur bisherigen Implementation ist das Transferieren der Daten. Sie werden nicht wie in Flask direkt als Form, sondern, aufgrund der neuen Architektur, von der API als Dictionary an die Methode übergeben. Des Weiteren änderte sich das Auslesen der Parameter, da diese als String aus dem POST-Request gelesen werden. Ausserdem werden die Datenbanken, die “allow csv upload“ gesetzt haben, als DTO Objekte an das Frontend übergeben.

Der Ablauf im Backend des CSV-Importers gestaltet sich folgendermassen: Als erstes werden der Name der CSV-Datei und der Tabelle überprüft. Der Dateiname darf keine Sonderzeichen (z.B. LFI) enthalten. Diese werden entfernt. Sollte der Name danach leer sein, wird eine Meldung an das Frontend gesendet und der Vorgang abgebrochen. Der Tabellenname muss in Superset eindeutig sein, ansonsten kann es zu Konflikten kommen. Dies wird ebenfalls dem Benutzer gemeldet.

Danach wird die Datenbank je nach Use Case geladen oder erstellt:

Bestehender Use-Case „import into existing database“
Im neuen Prozess wird die Datenbank nicht mehr im Form als Objekt übermittelt, sondern lediglich die Datenbank-ID. Im Backend wird mit dieser das eigentliche Datenbankobjekt aus der Session geladen.

Neuer Use-Case „import into new database“
Beim neuen Use-Case kann der Benutzer wählen, ob er eine neue SQLite- oder PostgreSQL-Datenbank erstellt haben möchte. Die Fallunterscheidung, ob eine neue Datenbank erstellt werden soll, erfolgt durch die Datenbank ID. Falls das Frontend die ID “-1“ sendet, wird eine neue Datenbank erstellt, welche den benutzerdefinierten Namen erhält. Defaultmässig wird der CSV-Dateiname verwendet. Wie bereits bei der Datei werden auch beim Datenbanknamen die Sonderzeichen entfernt und bei leerem Namen der Benutzer informiert. Danach wird die Datenbank erstellt. Je nach gewählter Datenbankart wird die SQLAlchemy-URL anders zusammengesetzt und weitere Konfigurationen verlangt. Sollte in diesem Schritt etwas schief gehen, wird die soeben erstellte Datenbank wieder entfernt.

Nachdem die Datenbank erstellt oder geladen wurde, wird die Tabelle für den CSV-Importer generiert. Um den Inhalt der CSV-Datei in die Tabelle zu füllen, wird die Datei anschliessend gespeichert. Das Befüllen übernimmt die Superset Klasse BaseEngineSpec. Die dazugehörige Methode haben wir von Superset übernommen und angepasst.

Bei jedem Schritt könnten Fehler auftreten. Diese werden von unseren eigens kreierten Exceptions abgefangen und mit einer verständlichen Meldung an den Benutzer weitergeleitet. Für die Kommunikation mit dem Frontend im Fehlerfall werden “json_response“ Objekte verwendet. Die Information, dass der Import erfolgreich war, wird wie bis anhin mit einer Flash-Message dargestellt, welche nach der Weiterleitung auf die Tabellenliste automatisch erscheint.

In dem nachfolgenden Diagramm werden die von uns geänderten und erstellen Klassen bzw. Methoden dargestellt. Die grauen Komponenten stellen die verwendeten Bibliotheken dar, während die Grünen die Superset-eigenen Objekte symbolisieren.

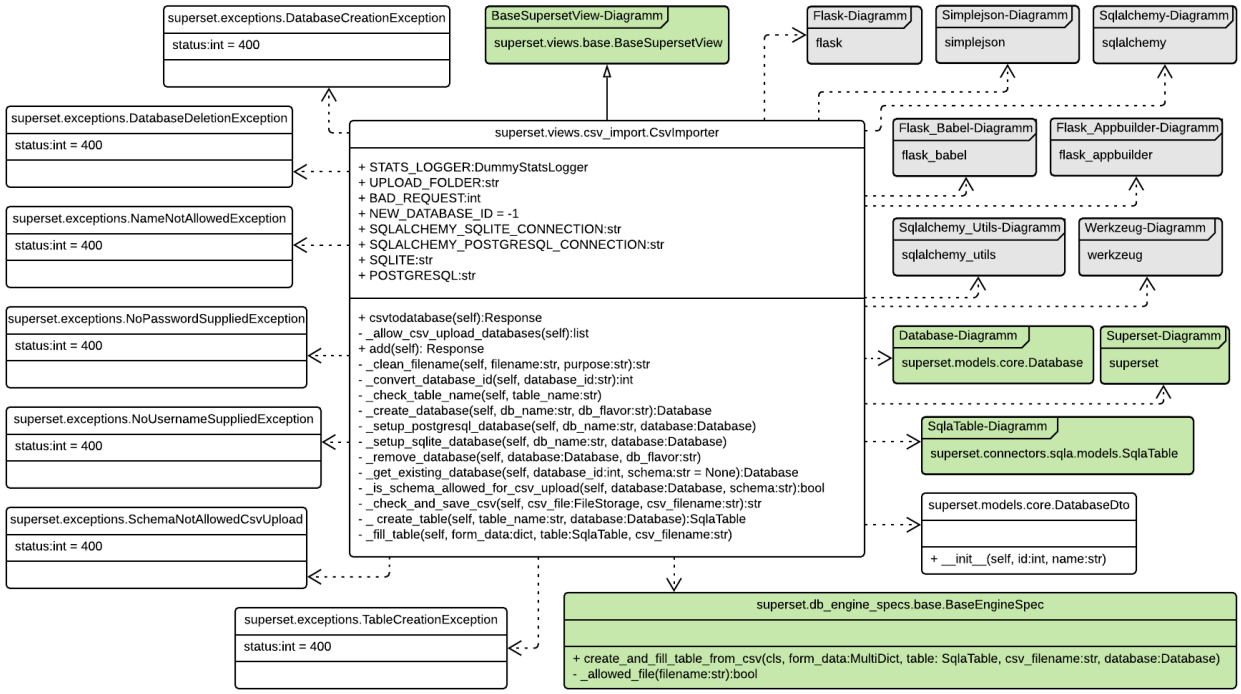


Abbildung 23: Vereinfachtes Klassendiagramm, zugeschnitten auf den CSV-Importer

Schnittstellenbeschreibung
Beim CSV-Importer gibt es zwei Schnittstellen. Die erste Schnittstelle, erreichbar unter “csvimporter/csvtodatabase“, liefert das HTML an den Client und entspricht dem Frontend. Die zweite Schnittstelle (“csvimporter/csvtodatabase/add“) wird beim Senden des Formulars aufgerufen und entspricht somit dem Backend.

Abbildung 24: CSV-Importer Schnittstellen-Übersicht

Die Schnittstelle für das Frontend (“csvimporter/csvtodatabase“) benötigt keine Parameter. Die Backend-Schnittstelle “csvimporter/csvtodatabase/add“ hingegen benötigt diverse Parameter. Diese entsprechen den im Eingabeformular vorhandenen Einstellungen. Vom Backend können daraufhin drei verschiedene Antworten zurückgegeben werden. Bei erfolgreichem API-Aufruf wird ein “200“er Status Code retourniert. Tritt ein Fehler auf, wird je nach Fehlerart eine “400“er, respektive eine “500“er Response zurückgegeben. Die Error-Responses enthalten die Fehlermeldungen, welche dann im Frontend dargestellt werden.

8.1.2 Geokodierung

Die Geokodierung ist eine komplett neue Funktionalität, weshalb wir uns weniger in bestehende Strukturen einhängen mussten. Ziel war es, bereits in einer Tabelle vorhandene Daten zu geokodieren und diese Daten in der Tabelle abzuspeichern. Die Logik hatten wir bereits zu Beginn in verschiedene unabhängige Methoden unterteilt, um die Arbeit besser aufteilen zu können. Hierbei haben sich vier Hauptschritte herauskristallisiert.

- Daten aus den angegebenen Spalten Auslesen
- Neue Spalten für Längen- und Breitengrade hinzufügen, falls gewünscht
- Datensätze geokodieren
- Geokodierte Werte in die Tabelle schreiben

Vorbereitungen
Damit der Benutzer weiss, welche Tabellen und Daten verfügbar sind und um Tippfehler zu vermeiden, stellen wir die Tabellen und deren Spalten im Frontend als Select-Auswahllisten dar. Dabei kann der Benutzer bis zu vier Spalten auswählen, welche benutzt werden sollen, um möglichst genaue Daten von der Geokodierungs-Schnittstelle zu erhalten. Dem Benutzer werden nur diejenigen Tabellen angezeigt, welche "allow DML" auf ihrer Datenbank gesetzt haben. Ohne diese Eigenschaft können wir die geokodierten Daten nicht zu der Tabelle hinzufügen.

Daten aus angegebenen Spalten auslesen
Im ersten Schritt der Geokodierung werden die zu kodierenden Daten aus der gewählten Tabelle gelesen. Dabei werden nur die angegebenen Spalten ausgelesen. Diese müssen zuvor überprüft werden, da der Request abgefangen und insofern manipuliert werden konnte.

Neue Spalten für Längen- und Breitengrade hinzufügen, falls gewünscht
Danach werden die Spalten für die Längen- und Breitengrade erstellt. Diese werden vor der eigentlichen Kodierung erstellt, da diese die teurere Operation ist. Die geokodierten Daten könnten beispielsweise nicht gespeichert werden und der ganze Vorgang müsste man wiederholen. Dies würde nicht nur lange dauern, sondern könnte, je nach API-Key, gegen dessen Limit zählen, was wir möglichst vermeiden möchten.

Datensätze geokodieren
Dies ist der zeitaufwendigste Schritt, da jeder einzelne Datensatz an eine API gesendet und die Antwort gespeichert werden muss. Das grösste Problem hierbei war, dass diverse Anbieter der Geokodierungs-Services nicht wollten, dass man die Datensätze "in Massen" verarbeitet und noch weniger, dass die Daten gespeichert werden. Hier haben wir uns, aufgrund der guten Beziehungen des Arbeitsbetreuers zu den Verantwortlichen von MapTiler, dafür entschieden, als erstes MapTiler zu implementieren. Obwohl auch MapTiler diese Einschränkungen in den AGBs festgehalten hat. Die zuerst geplante Implementierung mit GeoPy war nicht möglich, da GeoPy MapTiler nicht unterstützte.
Dieser Teil der Implementation war erst als asynchrone Methode angedacht. Schlussendlich haben wir sie synchron implementiert, so dass das Backend für diesen Vorgang blockiert ist. Der Grund dafür war, dass wir uns nicht mit Asynchronität in Python auskannten. Superset bietet Konfigurationsmöglichkeiten für das Framework Celery an, welches jedoch weder auf unserer Instanz noch auf der des Arbeitsbetreuers eingerichtet wurde. In Abklärung mit dem Arbeitsbetreuer und auch aus zeitlichen Gründen, haben wir uns gegen die Verwendung von Celery entschieden.
Die verbliebene Zeit investierten wir in einen zweiten Geokodierungs-Service und in eine Teil-Geokodierung. Wir entschieden uns aus verschiedenen Gründen für Google. Unter anderem hat Google keine Einschränkungen zur Nutzung ihres Geokodierungs-Services, ihre Schnittstelle ist sehr zuverlässig und kann mit vielen Adressen, Orten und anderen geographischen Merkmalen umgehen. Des Weiteren ist Google bei vielen sehr beliebt und bekannt.
Die Teil-Geokodierung implementierten wir aus der Überlegung heraus, dass ein Benutzer bereits einen Teil seiner Daten geokodieren konnte und die restlichen ebenfalls kodieren möchte. Dabei sollen aber nur diejenigen an die externe Schnittstelle gesendet werden, welche noch keine Koordinaten besitzen.

Geokodierte Datensätze in die Tabelle schreiben
Als Letztes werden die geokodierten Daten in die Tabelle niedergeschrieben, so dass diese für visuelle Darstellungen auf den Superset-Karten benutzt werden können, ohne dass die dazugehörigen Adressen jedes Mal neu geokodiert werden müssen.

Optionen und weitere Eigenschaften
Falls der Benutzer den Geokodierungsvorgang vorzeitig abbrechen möchte, zum Beispiel weil er merkt, dass eine falsche Spalte ausgewählt wurde, gibt es ein Interrupt-Flag, welches den Vorgang unterbricht. Dieses wird vor jedem Datensatz, der verarbeitet werden soll, geprüft.
Des Weiteren gibt es eine “saveOnInterrupt “ Variable, welche bei einem Unterbruch der Kodierung zum Zuge kommt. Sie entscheidet, ob bei einem schwerwiegenden Fehler oder einem Interrupt die bereits ermittelten Koordinaten gespeichert werden oder ob alles verworfen werden soll. Tritt zum Beispiel ein schwerwiegender Fehler nach 3/4 der verarbeiteten Datensätze ein, so dass der Vorgang abgebrochen werden muss, wurde die Arbeit nicht umsonst gemacht und die Daten trotzdem gespeichert, falls der Benutzer dies wünscht.

In dem nachfolgenden Diagramm werden die von uns geänderten und erstellen Klassen bzw. Methoden dargestellt. Die grauen Komponenten stellen die verwendeten Bibliotheken dar, während die Grünen die Superset-eigenen Objekte symbolisieren.

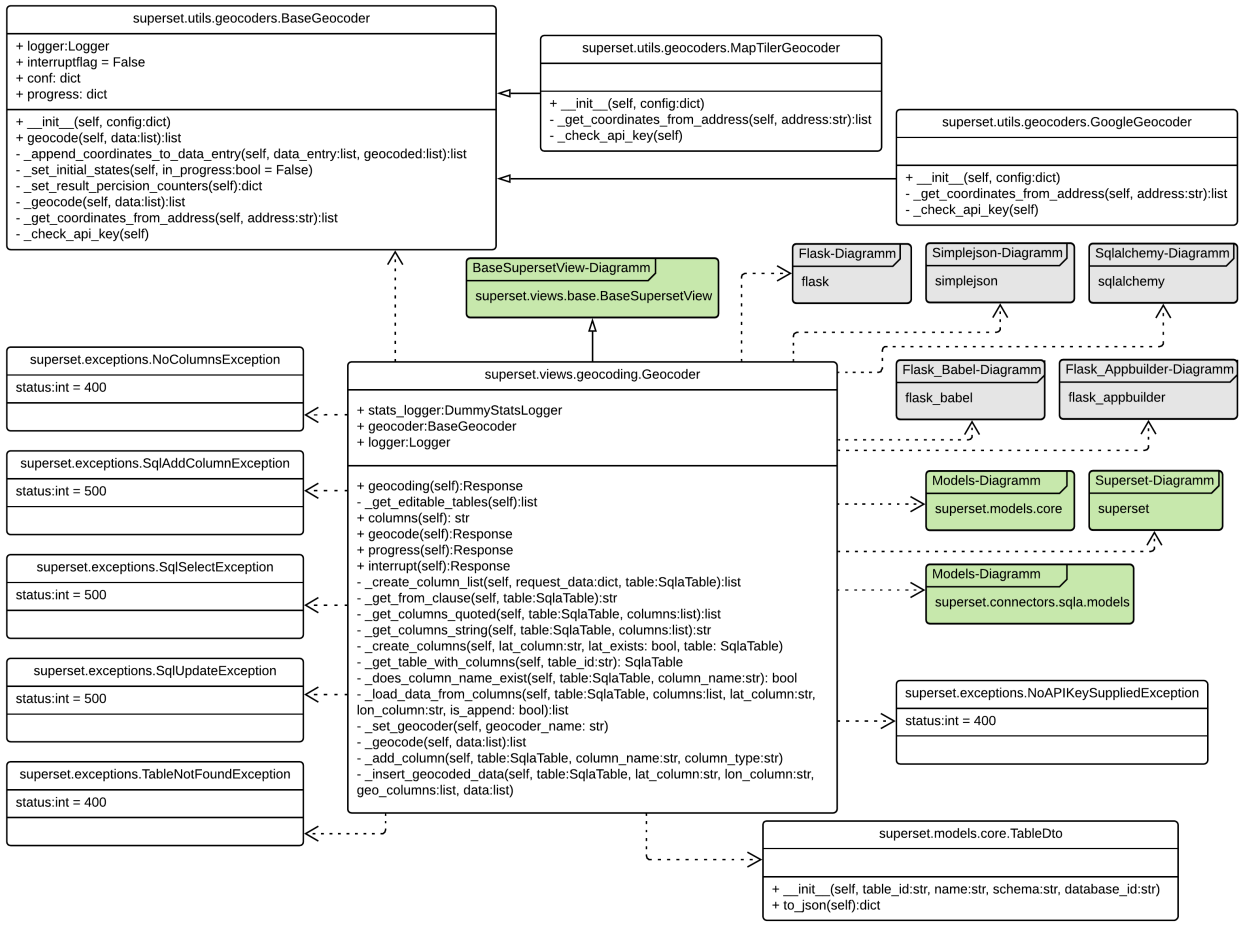


Abbildung 25: Vereinfachtes Klassendiagramm, zugeschnitten auf die Geokodierung

Schnittstellenbeschreibung
Bei der Geokodierung gibt es fünf verschiedene Schnittstellen.

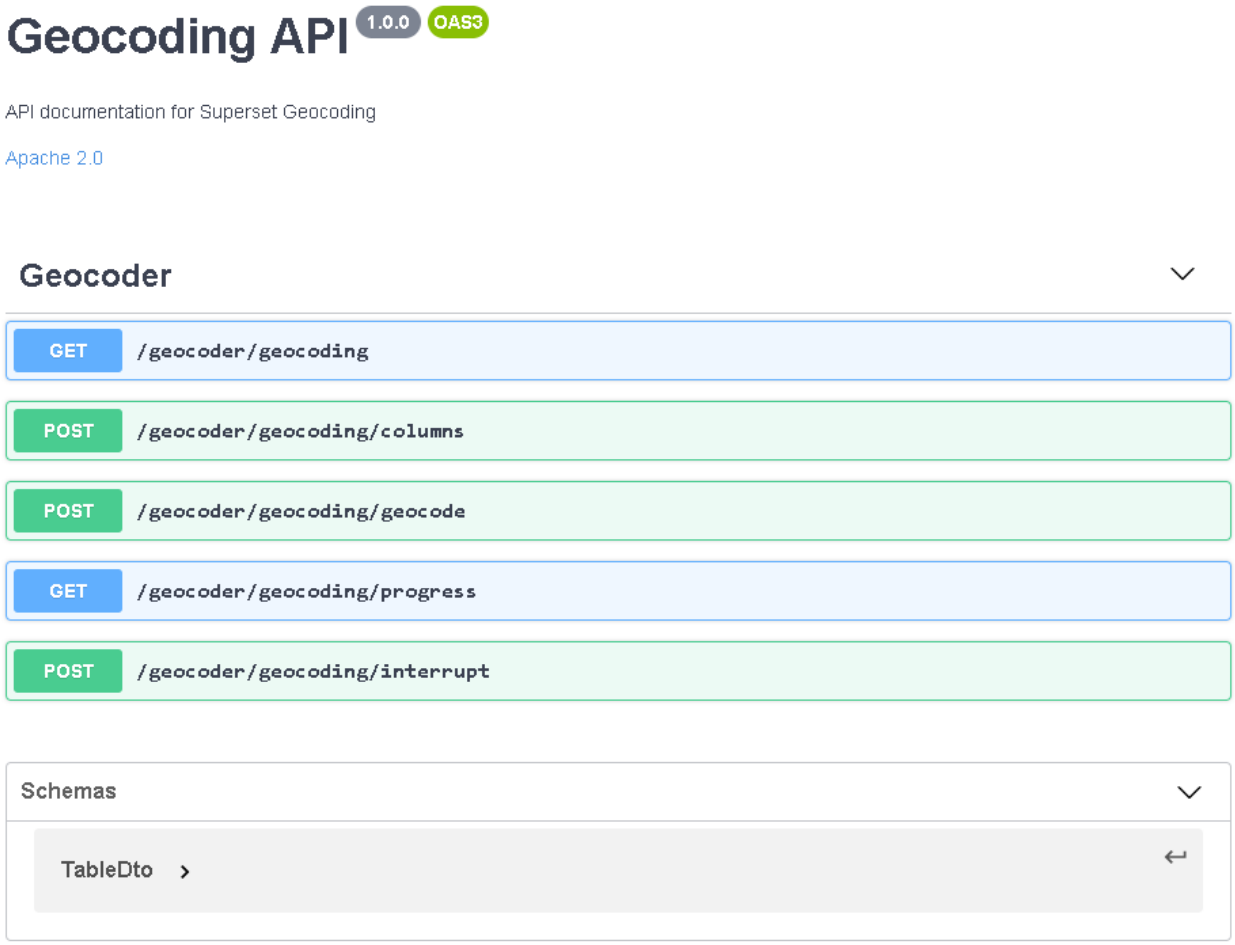


Abbildung 26: Geokodierung Schnittstellen-Übersicht

Die erste Schnittstelle, erreichbar unter „/geocoder/geocode“, liefert das HTML an den Client und entspricht dem Frontend. Die „/geocoder/geocoding/columns“ Schnittstelle ist dazu da, dass bei der Auswahl einer Datenquelle die verschiedenen Select-Eingaben mit den richtigen Spalten befüllt werden. Dadurch kann der Benutzer aus den bestehenden Spalten auswählen und muss nicht wissen, wie diese genau heissen. Des Weiteren werden Tippfehler durch den Benutzer vermieden. Die wichtigste Schnittstelle ist die „/geocoder/geocoding/geocode“ Schnittstelle. Sie macht den Hauptteil der Arbeit und ist für das eigentliche Geokodieren zuständig.
Um dem Benutzer auch anzeigen zu können, wo die Geokodierung gerade steht, gibt es die „/geocoder/geocoding/progress“ Schnittstelle. Sie teilt dem Benutzer einerseits mit, ob eine Geokodierung stattfindet, andererseits wird der aktuelle Fortschritt inklusive einer Anzeige, wie viele Adressen erfolgreich geokodiert werden konnten, zurückgegeben.
Zu guter Letzt gibt es noch eine „/geocoder/geocoding/interrupt“ Schnittstelle. Mit Hilfe dieser API kann der Benutzer den Geokodierungsprozess jederzeit unterbrechen. Je nach Einstellungen im Formular werden die bis dahin geokodierten Daten daraufhin gespeichert oder verworfen.

8.2 Frontend

Auch im Frontend wollten wir eine moderne Umsetzung realisieren. Ein Grossteil der UI-Komponenten von Superset wurden mit dem Flask-AppBuilder umgesetzt. Aufgrund der Unflexibilität und unscharfen Trennung von Front- und Backend der Flask-Formulare, entschieden wir uns, auf das Framework Flask-AppBuilder für die Frontendentwicklung zu verzichten. Dieses wurde stattdessen mit React umgesetzt. Dank React wurden wir in der Auswahl der HTML-Elemente nicht eingeschränkt und konnten eine ordentliche Trennung zwischen Front- und Backend erzielen.

8.2.1 Konfiguration von Flask und React

Das Diagramm 27 zeigt den Zusammenhang von Flask und React am Beispiel des CSV-Importers und wird in diesem Paragraphen erläutert.

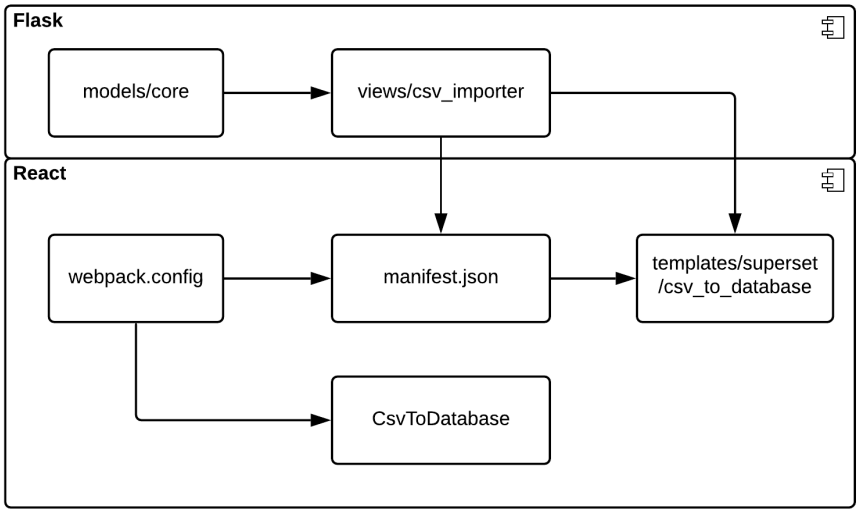


Abbildung 27: Frontend Verbindungen und Zusammenhänge

Da in Superset nicht ausschliesslich React für das Frontend verwendet wird, musste unsere Komponente erst einmal mit Flask verbunden werden. Superset bündelt die ganzen React-Komponenten mit Hilfe von Webpack[43] in statische “Assets“. Speziell ist bei Superset, dass aus Webpack nicht direkt das HTML generiert wird, sondern lediglich die statischen Assets und eine Manifest-Datei, welche dann von Flask dazu verwendet wird, ein HTML zu generieren. Zuerst haben wir einen Endpoint im Flask registriert, der auf die URL “/csvimporter/csvtodatabase“ reagiert und die Methode “csvtodatabase“ ausführt. Die Methode “csvtodatabase“ im csv_import.py wird beim Aufruf des CSV-Importers ausgeführt, weil Flask die passende @expose-Annotation zur URL bei dieser findet. Sie liefert nicht nur die verfügbaren CSV-Datenbanken aus der gleichnamigen Modelklasse an das Frontend zurück, sondern definiert auch, welches Template angezeigt werden soll und konfiguriert dieses.

[43] Webpack: https://webpack.js.org/

Listing 1: Frontend Route

```
@has_access
@expose("/csvtodatabase")
def csvtodatabase(self):
  session = db.session()
  Database = models.Database
  databases =
     session.query(Database).filter_by(allow_csv_upload=True).all()
  databases_json = [models.DatabaseDto(-1, "In a new database")]
  for database in databases:
    databases_json.append(models.DatabaseDto(database.id,
       database.name))

  bootstrap_data = {
    "databases": databases_json,
    "common": self.common_bootstrap_payload(),
  }

  if request.args.get("json") == "true":
    return json_success(
      json.dumps(bootstrap_data, default=lambda x: x.__dict__)
    )

  return self.render_template(
    "superset/csv_to_database.html",
    entry="csvToDatabase",
    standalone_mode=False,
    title="CSV to Database configuration",
    bootstrap_data=json.dumps(bootstrap_data, default=lambda x:
       x.__dict__),
  )
```

Dabei ist es wichtig, dass der “entry“ -Punkt mit demjenigen im webpack.config übereinstimmt. Anhand dessen wird nämlich bestimmt, welche React-Komponenten in das Template eingefügt werden sollen.

Listing 2: Auszug webpack.config.js

```
entry: {
  theme: path.join(APP_DIR, '/src/theme.js'),
  preamble: PREAMBLE,
  addSlice: addPreamble('/src/addSlice/index.jsx'),
  explore: addPreamble('/src/explore/index.jsx'),
  dashboard: addPreamble('/src/dashboard/index.jsx'),
  sqllab: addPreamble('/src/SqlLab/index.jsx'),
  welcome: addPreamble('/src/welcome/index.jsx'),
  profile: addPreamble('/src/profile/index.jsx'),
  csvToDatabase: addPreamble('/src/CsvToDatabase/index.jsx'),
  showSavedQuery: [path.join(APP_DIR,
     '/src/showSavedQuery/index.jsx')],
},
```

React generiert mit Hilfe von Webpack, anhand der webpack.config, aus der React-Komponente CsvToDatabase eine JavaScript-Datei und das manifest.json, in welchem die Files hinterlegt sind. Die “csvtodatabase“ -Methode fügt die erstellten JavaScript-Dateien anschliessend in das HTML Template ein.

Listing 3: HTML Template für CSV-Importer

```
{% extends "superset/basic.html" %}

{% block body %}
<div id="app" data-bootstrap="{{ bootstrap_data }}" />
{% endblock %}
```

Das “div“ mit der id “app“ ist hierbei sehr wichtig, denn die React-Komponenten werden bei diesem HTML Element als sogenannte “Children“ eingefügt und damit dargestellt. Über das “extends“ kommen sowohl die Navigation als auch das CSRF-Token.

8.2.2 Kommunikation zwischen Front- und Backend

Für die Kommunikation zwischen Front- und Backend setzten wir das Package “superset-ui/connection“ [44] ein. Dieses macht die eigentlichen Anfragen und verarbeitet die Antworten. Damit die Komponente frei von der konkreten Abfrage-Logik ist, setzten wir im React das Redux-Framework ein. Dies hat den erfreulichen Nebeneffekt, dass sich Redux sehr gut debuggen lässt. Der Redux-Flow ist im nachfolgenden Diagramm 28 aufgezeigt.

[44] Superset-UI/Connection: https://github.com/apache-superset/superset-ui/tree/master/packages/superset-ui-connection

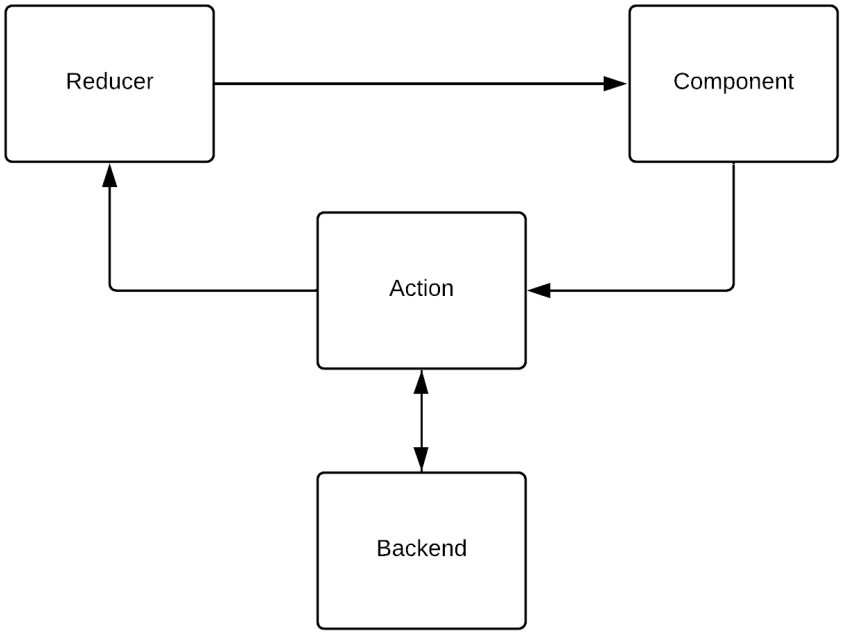


Abbildung 28: Redux Flow

Die Komponente selbst (in diesem Fall die CsvToDatabase-Komponente) kennt lediglich die “Action“, die es ausführen möchte. Die “Action“ ist so implementiert, dass diese mit Hilfe des “superset-ui/connection“ Packages die Abfrage an den Server macht und je nach Antwort eine weitere Aktion, mit einem spezifischen Namen versendet. Der “Reducer“ hört ständig mit und prüft, ob eine spezifische “Action“ versendet wurde. Ist dies der Fall, verarbeitet er das Resultat der Aktion und informiert die Komponente, dass neue Daten vorhanden sind. Konkret bedeutet dies im Fehlerfall, dass eine Fehlermeldung aus dem Request ausgelesen und in der Komponente “StatusMessages“ dargestellt wird. Im Erfolgsfall wird eine leere Meldung versendet und eine Umleitung auf die Tabellenliste gemacht.

Listing 4: Upload CSV Action

```
import { SupersetClient } from '@superset-ui/connection';
import getClientErrorObject from 'src/utils/getClientErrorObject';

export const UPLOAD_CSV_SUCCESS = 'UPLOAD_CSV_SUCCESS';
export const UPLOAD_CSV_FAILURE = 'UPLOAD_CSV_FAILURE';

export function uploadCsv(data) {
  return dispatch =>
    SupersetClient.post({
      endpoint: '/superset/csvtodatabase/add',
      postPayload: { ...data },
      stringify: false,
    })
    .then(() => {
      dispatch({ type: UPLOAD_CSV_SUCCESS, message: '' });
      window.open('/tablemodelview/list/', '_self');
    })
    .catch((response) => {
      getClientErrorObject(response).then((error) => {
      dispatch({ type: UPLOAD_CSV_FAILURE, message: error.error });
    });
  });
}
```

Listing 5: Upload CSV Reducer

```
import * as actions from '../actions/csvToDatabase';

export default function csvToDatabaseReducer(state = {}, action) {
  if (action.type === actions.UPLOAD_CSV_SUCCESS) {
    return Object.assign({}, state, {
      uploadStatus: { message: action.message, timestamp: Date.now()
          },
    });
  }
  if (action.type === actions.UPLOAD_CSV_FAILURE) {
    return Object.assign({}, state, {
      uploadStatus: { message: action.message, timestamp: Date.now()
          },
    });
  }
  return state;
}
```

8.2.3 Übersetzung

Für die Übersetzungen wird das Package “superset-ui/translation“ verwendet. Superset verwendet Babel [45] für die Übersetzungen. Man braucht lediglich die jeweiligen Texte, die man übersetzt haben möchte, in einen Funktionsaufruf zu wrappen und zur Laufzeit werden diese dann automatisch übersetzt, sofern die entsprechende Übersetzung hinterlegt ist. Ansonsten wird der Text so zurückgegeben, wie er im Code eingegeben wurde.

8.2.4 CSV-Importer

Der CSV-Importer wurde komplett neugestaltet. Die ursprüngliche Ansicht, welche mit Flask-AppBuilder erstellt wurde, ersetzten wir durch eine neue React-Komponente.

Komponenten
Wir legten grossen Wert darauf, dass die Erscheinung dem ursprünglichen Formular ähnlich sieht. Dabei wurden sowohl neue als auch bestehende Komponenten in Superset verwendet. Das Diagramm 29 zeigt den Einsatz der Komponenten für den CSV-Importer.

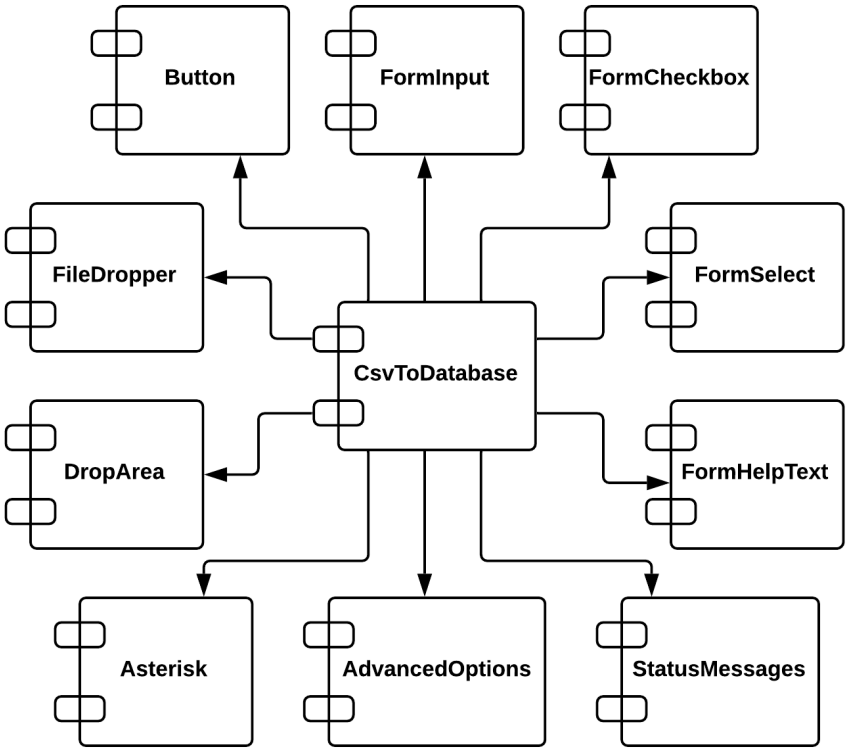


Abbildung 29: React-Komponentendiagramm vom CSV-Importer

Die Hauptkomponente ist “CsvToDatabase“ . In dieser werden die Informationen für die Erstellung einer neuen Tabelle sowie die CSV-Datei mitgegeben.

[45] Babel: http://babel.pocoo.org/en/latest/

Das Formular beinhaltet dieselben Felder wie der ehemalige CSV-Importer. Allerdings wurde die Reihenfolge angepasst, so dass die zwingenden Felder weiter oben dargestellt und die Optionalen hinter einer “Advanced Options“ Sektion versteckt werden. Dadurch wirkt das Formular ruhiger sowie übersichtlicher und ist für den Benutzer einfacher zu bedienen.

Das Auswählen der CSV-Datei wurde durch eine Drag and Drop-Komponente ersetzt. Es ist weiterhin möglich, über einen Klick auf diese, den bisher genutzten Datei-Dialog zu öffnen. Zusätzlich kann der Anwender seine Dateien direkt in den Browser ziehen. Dafür erstellten wir die Komponenten “FileDropper“ für den Dialog und “DropArea“ für das Drag and Drop. Die Komponenten sind bewusst getrennt, damit man nicht zu viel Logik miteinander vermischt und verschieden gestaltete Drag and Drop-Bereiche implementieren kann. Die Komponente ist auf maximale Browser-Kompatibilität ausgerichtet. Für diejenigen Browser, die den Drag and Drop nicht unterstützen, wird ein Fallback auf einen normalen Datei-Input gemacht.

Für die verschiedenen Input-Typen wurden eigene Komponenten erstellt, damit diese so gut wie möglich wiederverwendet werden können. Der “FormInput“, die “FormCheckbox“ sowie das “FormSelect“ bestehen alle aus dem jeweiligen Input-Feld und dem optionalen “FormHelpText“, welcher unterhalb der Komponente dargestellt werden kann. Um die Darstellung einheitlich zu halten, wurden für die FormCheckbox sowie den FormSelect die bereits vorhandenen Komponenten “Checkbox“ und “Select“ verwendet (nicht in 29 dargestellt).

Auch das rote Asterisk hinter den zwingenden Feldern wurde in eine eigene Komponente ausgelagert und dadurch wiederverwendbar gemacht.

Alternativer, verworfener Ansatz
Da Superset bisher alle ihre Eingabeformulare mit dem Flask-AppBuilder generiert hat, wollten wir dies zu Beginn auch so umsetzen. Flask-AppBuilder verwendet dazu die so genannten “WTForms“, eine Art HTML-Template Format, welches für das Generieren von Forms verwendet werden kann.

Obwohl man bei den WTForms auch eigene Komponenten erstellen kann, merkten wir schnell, dass diese sehr eingeschränkt sind. Bei der Drag and Drop-Komponente stiessen wir an die Grenzen des Möglichen, da so eine Komponente einiges an JavaScript-Code benötigt, um korrekt zu funktionieren. Ausserdem sollte man bei einem Drag and Drop stets einen Fallback implementieren, da man nicht davon ausgehen kann, dass alle Benutzer die neuesten Browser benutzen. So etwas war mit WTForms nicht möglich.

Ein weiteres grosses Problem der Flask-AppBuilder-Variante, war das Aufteilen der Implementation. Das Flask-Formular fungiert gleichzeitig als Frontend- und Backend-Logik. Wir hatten bereits am Anfang dutzende Merge-Konflikte, die wir auflösen mussten. Es war nahezu unmöglich die Arbeit sinnvoll untereinander aufzuteilen, da alle Änderungen in der gleichen Datei und Klasse gemacht werden mussten.

Wir hatten also die Wahl: Auf das Drag and Drop verzichten, oder einen grösseren Umbau vorzunehmen, um eine zeitgemässe und moderne Implementation zu garantieren. Wir haben uns letztlich gegen eine Implementation mit dem Flask-AppBuilder als Frontend-Framework und für einen Umbau auf React als Frontend und Flask (API) als Backend entschieden.

8.2.5 Geokodierung

Bei der Geokodierung handelte es sich um eine neue Funktionalität, welche von Grund auf implementiert werden musste. Wir konnten einige Komponenten des CSV-Importers wiederverwenden, mussten aber auch Neue hinzufügen.

Komponenten
Wie bereits beim CSV-Importer legten wir auch beim Geokodierer viel Wert darauf, dass die Erscheinung den bisher vorhandenen Formularen ähnlich sieht.

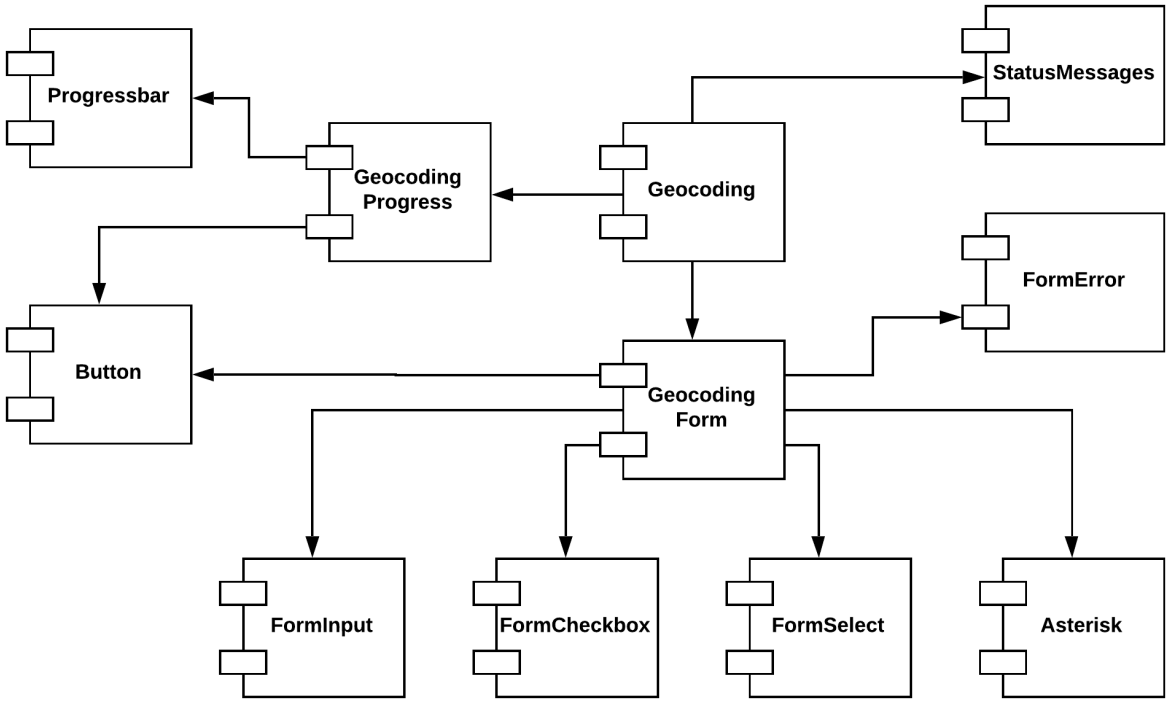


Abbildung 30: React-Komponentendiagramm der Geokodierung

Im Diagramm 30 werden die Zusammenhänge der React-Komponenten aufgezeigt. Im Zentrum steht die “Geocoding“ Komponente, welche verschiedene Teilkomponenten verwendet. Diese besteht hauptsächlich aus einem Formular und einer Fortschrittsanzeige. Wenn eine Geokodierung stattfindet, wird die Fortschrittsseite dargestellt, sonst das Formular. Die Fortschrittsseite entspricht der “GeocodingProgress“ Komponente und besteht hauptsächlich aus einem Fortschrittsbalken, einer Confidence-Anzeige (Relevanz der Koordinaten) und einem Abbrechen-Button, mit dem man die Geokodierung stoppen kann. Das Formular besteht aus den Eingabekomponenten “FormInput“, “FormCheckbox“, “FormSelect“ und “Asterisk“, welche bereits beim CSV-Importer implementiert wurden. Für die Darstellung der Fehlermeldungen und die Formvalidierung wurde eine neue Komponente “StatusMessages“ gebaut.

9 Implementation

In den folgenden Unterkapiteln gehen wir auf die Implementationsdetails unserer Funktionalitäten ein. Dabei wollen wir auf Codefragmente und die dahinterliegenden Entscheidungen eingehen.

9.1 Backend

Hier werden die Details des Backends erläutert und die wichtigsten Code-Ausschnitte aufgezeigt.

9.1.1 CSV-Importer

Der CSV-Importer war bisher mit Flask implementiert. Das Frontend war abhängig vom Backend und umgekehrt. Um diese Abhängigkeit aufzulösen, mussten wir im Backend einiges umbauen und teilweise neu implementieren.

Neue Datenbanken erstellen
In einem ersten Schritt werden die Formulardaten entgegengenommen und verarbeitet. Zuerst wird geprüft, ob eine neue Datenbank erstellt werden muss. Ist dies der Fall, wird je nach Einstellung eine SQLite- oder PostgreSQL-Datenbank angelegt. Alle weiteren Schritte finden danach auf dieser Datenbank statt. Die Methode für das Erstellen der Datenbank wurde unabhängig vom Datenbanktyp gehalten:

Listing 6: csv_import.py _create_database

```
def _create_database(self, db_name: str, db_flavor: str) ->
    Tuple[Database, str]:

  database = SQLAInterface(Database).obj()
  database.database_name = db_name
  database.allow_csv_upload = True

  try:
    if db_flavor == POSTGRESQL:
      uri = self._setup_postgresql_database(db_name, database)
    else:
      uri = self._setup_sqlite_database(db_name, database)
    db.session.add(database)
    db.session.commit()
    return database, uri
  except DatabaseAlreadyExistException as e:
    raise DatabaseAlreadyExistException(e.args[0], e)
  except IntegrityError as e:
    raise DatabaseCreationException("Error when trying to create
       Database", e)
  except Exception as e:
    raise DatabaseCreationException(
    "An unknown error occurred trying to create the database.", e)
```

Die Datenbank wird mittels SQLAlchemy erstellt und für CSV-Importers präpariert. Die SQLAlchemy URI und die weiteren Einstellungen werden in den dafür vorgesehenen Methoden gesetzt. Sobald die Datenbank fertig initialisiert wurde, wird sie gespeichert.
Die "_create_database" Methode gibt zwei Parameter zurück: Die erstellte Datenbank und die URI zum Löschen der Datenbank, falls beim Erstellen, bzw. beim Befüllen der Tabelle ein Fehler auftritt und die Datenbank wieder entfernt werden soll.
Beim Erstellen der Datenbank können einige Probleme auftreten. Diese werden mit unseren eigenen Exceptions abgefangen und mit Hilfe von verständlichen Beschreibungen dem Benutzer gemeldet.

Multidatabase Support beim Erstellen einer neuen Datenbank
Zu Beginn fokussierten wir uns auf das Erstellen einer SQLite-Datenbank, da diese keine Konfigurationen oder Vorinstallationen benötigt und sie deshalb überall erstellt werden kann. Erst danach implementierten wir auch PostgreSQL als Datenbanktyp.

Wir hielten die Methode, welche die Datenbank erstellt, generisch. Die spezifischen Einstellungen für SQLite und PostgreSQL lagerten wir in Methoden aus:

Listing 7: csv_import.py _setup_postgresql_database

```
def _setup_postgresql_database(self, db_name: str, database:
    Database) -> str:
  postgresql_user = conf["POSTGRESQL_USERNAME"]
  if not postgresql_user:
    raise NoUsernameSuppliedException(
      "No username supplied for PostgreSQL", None
    )
  postgresql_password = conf["POSTGRESQL_PASSWORD"]
  if not postgresql_password:
    raise NoPasswordSuppliedException(
      "No password supplied for PostgreSQL", None
    )

  url = (
    SQLALCHEMY_POSTGRESQL_CONNECTION
    + postgresql_user
    + ":"
    + postgresql_password
    + "@localhost/"
    + db_name
  )
  enurl = (
    "postgresql://"
    + postgresql_user
    + ":"
    + "XXXXXXXXXX"
    + "@localhost/"
    + db_name
  )

  engine = sqlalchemy.create_engine(url)
```

```
  if not sqlalchemy_utils.database_exists(engine.url):
    sqlalchemy_utils.create_database(engine.url)
  else:
    raise DatabaseAlreadyExistException(
      f"The database {db_name} already exists", None
    )

  database.sqlalchemy_uri = enurl
  database.password = postgresql_password
  return url
```

Listing 8: csv_import.py _setup_sqlite_database

```
def _setup_sqlite_database(self, db_name: str, database: Database)
   -> str:
  db_path = os.getcwd() + "/" + db_name + ".db"
  if os.path.isfile(db_path):
    message = f"Database file for {db_name} already exists, please
       choose a different name"
    raise DatabaseAlreadyExistException(message, None)
  database.sqlalchemy_uri = SQLALCHEMY_SQLITE_CONNECTION + db_path
  return db_path
```

Zu sehen ist, dass für SQLite- und PostgreSQL-Verbindungen unterschiedliche Einstellungen getätigt werden müssen. Während bei SQLite lediglich die Datenbank als Datei angelegt werden muss, verlangt eine PostgreSQL-Datenbank eine unverschlüsselte URL für die SQL Engine und eine verschlüsselte URL für SQLAlchemy. Der Grund, weshalb wir eine verschlüsselte und eine unverschlüsselte URL verwenden müssen, ist SQLAlchemy. Die Bibliothek erwartet eine verschlüsselte URL und setzt danach das Passwort in die XXX Platzhalter. Das Passwort und der Benutzername für PostgreSQL werden aus der Konfiguration ausgelesen. Jede Datenbank wird dadurch mit dem gleichen Benutzer angelegt.
Beide Methoden geben eine URI zurück, welche im Fehlerfall verwendet wird, um die Datenbank wieder zu entfernen. SQLite gibt den Pfad der Datenbankdatei zurück. PostgreSQL hat als Rückgabeparameter die URL der SQLAlchemy Engine gesetzt.
Bei beiden Datenbanktypen könnte die Exception “DatabaseAlreadyExistException“ auftreten. Allerdings aus unterschiedlichen Prüfungen. Bei PostgreSQL wird die unverschlüsselte SQLAlchemy URL geprüft und bei SQLite wird kontrolliert, ob die Datenbankdatei bereits existiert. Zusätzlich können bei PostgreSQL Fehler auftreten, wenn die PostgreSQL-Benutzerdaten nicht in der Konfiguration hinterlegt wurden.

Da jeder Datenbanktyp andere spezifische Einstellungen benötigt, müssten für weitere Datenbanktypen zusätzliche Methoden erstellt werden. Dies führt leider dazu, dass keine generische Implementation für das Erstellen von Datenbanken möglich ist. Mit den Patterns “Dependency-Injection“[14] und “Factory-Method“[13] könnte die Implementation allerdings noch verbessert werden. Aus zeitlichen Gründen konnten diese Verbesserungen nicht mehr umgesetzt werden.

9.1.2 Geokodierung

Geokodierung der Adressen
Die Geokodierung wurde für zwei Geokodierungs-Services umgesetzt: MapTiler und Google. Damit weitere Services hinzugefügt werden können, wurde die Implementierung mit dem Factory-Method Pattern[13] umgesetzt: Die Klasse BaseGeocoder bildet das Grundgerüst und beinhaltet alle API-unabhängigen Methoden. Die spezifischen Methoden werden von den jeweiligen Geokodierungs-Service-Klassen überschrieben. Dadurch bleiben sie schlank und einfach. Die allgemeine Logik für das Geokodieren, Ermitteln des Fortschritts, Abbrechen und Ermitteln der Relevanz (Genauigkeit) werden somit nur einmal in der Basisklasse implementiert.

Die “check_api_key“-Methode prüft, ob in der Konfiguration der entsprechende API-Key des Services hinterlegt ist. Ohne diesen Key können keine Adressen geokodiert werden.
Die andere Methode, “_get_coordinates_from_address“, sendet die Adressen an den Service, welcher dann die entsprechenden Koordinaten und die Relevanz (Genauigkeit) zurück liefert. Der Rückgabewert muss immer gleich aufgebaut sein, so dass die aufrufende Methode aus der Klasse “BaseGeocoder“ die Daten weiterverarbeiten kann: Der erste Rückgabewert ist eine Liste mit zwei Einträgen, dem Breitengrad als erstes Element und dem Längengrad als Zweites. Als zweiter Rückgabewert wird die Relevanz der gefundenen Koordinate zurückgegeben.

Listing 9: MapTilerGeocoder

```
class MapTilerGeocoder(BaseGeocoder):

    def __init__(self, config: dict):
        BaseGeocoder.__init__(self, config)

    def _get_coordinates_from_address(self, address: str):
        base_url = "https://api.maptiler.com/geocoding/"
        response = get(
            base_url + address + ".json?key=" +
               self.conf["MAPTILER_API_KEY"]
        )
        decoded_data = json.loads(response.content.decode())
        features = decoded_data["features"]
        if features:
            feature = features[0]
            center = feature["center"]
            relevance = feature["relevance"]
            if len(center) > 0:
                return [
                    [center[1], center[0]], relevance,
                ]  # Make sure format is [lat, lon]
        return None

    def check_api_key(self):
        if not self.conf["MAPTILER_API_KEY"]:
            raise NoAPIKeySuppliedException("No API Key for MapTiler
               was supplied")
```

Listing 10: GoogleGeocoder

```
class GoogleGeocoder(BaseGeocoder):

  def __init__(self, config: dict):
    BaseGeocoder.__init__(self, config)

  def _get_coordinates_from_address(self, address: str):
    base_url = "https://maps.googleapis.com/maps/api/geocode/"
    response = get(
      base_url + "json?address=" + address + "&key=" +
         self.conf["GOOGLE_API_KEY"]
    )
    decoded_data = json.loads(response.content.decode())
    results = decoded_data.get("results")
    status = decoded_data.get("status")

    if status == "OK" and results and results[0]:
      geometry = results[0].get("geometry")
      location_type = geometry.get("location_type")
      relevance = 1.0
      if (
        location_type == "RANGE_INTERPOLATED"
        or location_type == "APPROXIMATE"
        or location_type == "GEOMETRIC_CENTER"
      ):
        relevance = 0.5
      location = geometry.get("location")
      lat = location.get("lat")
      lon = location.get("lng")
      return [[lat, lon], relevance] or None
    elif (
      status == "OVER_DAILY_LIMIT"
      or status == "OVER_QUERY_LIMIT"
      or status == "REQUEST_DENIED"
      or status == "INVALID_REQUEST"
    ):
      raise RequestException(status)
      return None

  def check_api_key(self):
    if not self.conf["GOOGLE_API_KEY"]:
    raise NoAPIKeySuppliedException("No API Key for Google was
       supplied")
```

Fortschrittsanzeige
Die Fortschrittsanzeige wurde mit Hilfe eines Dictionaries umgesetzt. Alle relevanten Informationen zum Fortschritt, wie zum Beispiel die verarbeiteten Datensätze in Prozent oder die erfolgreich geokodierten Daten, werden darin gespeichert.

Listing 11: geocoders.py Initialisierung Fortschrittsanzeige

```
self.progress["success_counter"] = 0
self.progress["doubt_counter"] = 0
self.progress["failed_counter"] = 0
self.progress["is_in_progress"] = in_progress
self.progress["progress"] = 0
```

Die geokodierten Daten werden als Listen von den Services zurückgegeben. Darin enthalten sind, wie bereits oben erwähnt, die Koordinaten (Längen- und Breitengrade) und die Relevanz, wie eindeutig die Koordinaten berechnet wurden. Die Schwellwerte der Relevanz für “Sicher“, “Unsicher“ und “Fehlerhaft“ wurden wie folgt festgelegt:

- alles grösser als 80% gilt als “Sicher“
- zwischen 50% und 80% gilt als “Unsicher“
- alles kleiner als 50% gilt als “Fehlerhaft“

Die geokodierten Daten werden dann zu den jeweiligen Tabelleneinträgen hinzugefügt. Dabei wird auch gleichzeitig der Fortschritt aktualisiert.

Listing 12: geocoders.py _append_cords_to_data_entry

```
def _append_cords_to_data_entry(self, data_entry: list, geocoded:
   list):
    coordinates = geocoded[0]
    relevance = geocoded[1]
    if relevance > 0.8:
        self.progress["success_counter"] += 1
    elif relevance >= 0.5:
       self.progress["doubt_counter"] += 1
    else:
       self.progress["failed_counter"] += 1
   data_entry.append(str(coordinates[0]))
   data_entry.append(str(coordinates[1]))
   return data_entry
```

Die Variable data_entry enthält die geographischen Daten des Datensatzes, den wir aus der Tabelle gelesen haben. Zu diesem fügen wir die Koordinaten hinzu. Dadurch können wir beim Einfügen der Koordinaten in die Tabelle die SQL-Abfrage leichter zusammenstellen, da wir die WHERE- und SET-Parameter in einem Listeneintrag zusammen haben und direkt darauf zugreifen können.

Rechte
Die Geokodierung wurde so gestaltet, dass nur Administratoren diese Funktionalität sehen und verwenden können. Die initiale Rechtevergabe für die verschiedenen Rollen wird im security.py festgelegt. Dabei erbt der Administrator alle Berechtigungen, welche die anderen Rollen besitzen. Folgende Rechte werden im security.py definiert: Menü-Ansichten und API-Zugriffe.

Der Menü-Eintrag der Geokodierung wurde der Liste “ADMIN_ONLY_VIEW_MENUS“ hinzugefügt. In die Liste “ADMIN_ONLY_PERMISSIONS“ wurden die API-Zugriffe der Geokodierung eingetragen.

Listing 13: security.py

```
ADMIN_ONLY_VIEW_MENUS = {
  "AccessRequestsModelView",
  "Manage",
  "SQL Lab",
  "Queries",
  "Refresh Druid Metadata",
  "ResetPasswordView",
  "RoleModelView",
  "Security",
  "Geocode Addresses",
} | USER_MODEL_VIEWS

ADMIN_ONLY_PERMISSIONS = {
  "can_sql_json",
  "can_override_role_permissions",
  "can_sync_druid_source",
  "can_override_role_permissions",
  "can_approve",
  "can_update_role",
  "can_geocoding",
  "can_geocode",
  "can_columns",
  "can_progress",
  "can_progress",
  "can_interrupt",
}
```

9.2 Frontend

In diesem Kapitel wird die Umsetzung des Frontends dokumentiert. Es werden die verschiedenen React-Komponenten genauer beschrieben und deren Implementation aufgezeigt.

9.2.1 Übergreifende Komponenten

Form-Inputs
Für die Implementation der Formulare wurden verschiedene Eingabefelder benötigt. Es handelt sich hierbei um ein Eingabefeld für Text, ein Eingabefeld für Zahlen, eine Checkbox und einen Select. Alle diese Komponenten wurden nach dem gleichen Schema implementiert, welches hier am Beispiel des Eingabefeldes “FormInput“ genauer erläutert wird.

Listing 14: FormInput.jsx Props

```
const propTypes = {
  name: PropTypes.string.isRequired,
  type: PropTypes.string.isRequired,
  placeholder: PropTypes.string,
  value: PropTypes.any,
  onChange: PropTypes.func,
  required: PropTypes.bool,
  disabled: PropTypes.bool,
  helpText: PropTypes.string,
};
```

Zuerst wurden die Parameter definiert. Diese können je nach Eingabetyp etwas anders sein. Es gibt aber immer die Option, das Eingabefeld zwingend zu machen (“required“), oder es zu deaktivieren (“disabled“). Dazu gibt es bei jedem Eingabetyp einen optionalen Hilfetext (“helpText“), der jeweils unterhalb des Eingabefeldes eingeblendet wird. Die “render“ Methode prüft, ob ein Hilfetext dargestellt werden soll und übergibt dem HTML-Eingabefeld die Parameter.

Listing 15: FormInput.jsx render

```
render() {
  const {
  name,
  type,
  placeholder,
  value,
  onChange,
  required,
  helpText,
  } = this.props;
  const help = helpText && <FormHelpText helpText={helpText} />;
  return (
    <>
      <input
        className="form-control"
        type={type}
        id={name}
        name={name}
        placeholder={placeholder}
        value={value}
        onChange={onChange}
        required={required}
        disabled={disabled}
      />
      {help}
    </>
  );
}
```

Statusmeldungen
Für die Statusmeldungen wurde eine eigene React-Komponente implementiert, die mit Hilfe des Redux-Stores die Meldungen vom Backend speichert. So kann sichergestellt werden, dass das gleiche Message-Verhalten wie bei den Flask-Forms auch im React weiterhin besteht. Für das Hinzufügen und Entfernen von Statusmeldungen wurden so genannte “Actions“ definiert, welche dann wiederum von React-Komponenten sowie anderen “Actions“ verwendet werden können (allgemein als “dispatching“ bezeichnet). Ebenfalls gibt es einen so genannten “Reducer“ für das Abarbeiten der zwei neuen “Actions“, welcher den Typ der aufgerufenen Aktion überprüft und gegebenenfalls darauf reagiert. In unserem Fall speichert der Reducer entweder eine neue Statusmeldung in den Store oder entfernt eine vorhandene aus diesem. Die “Reducer“ funktionieren also wie eine Art Middleware.

Listing 16: StatusMessage Actions

```
export const ADD_STATUS_MESSAGE = 'ADD_STATUS_MESSAGE';
export const REMOVE_STATUS_MESSAGE = 'REMOVE_STATUS_MESSAGE';

export const STATUS_TYPE = {
  INFO: 0,
  WARNING: 1,
  ERROR: 2,
};

export function addStatusMessage(message, statusType) {
  return dispatch => dispatch({ type: ADD_STATUS_MESSAGE, status: {
     message, statusType } });
}

export function removeStatusMessage(status) {
  return dispatch => dispatch({ type: REMOVE_STATUS_MESSAGE, status
     });
}
```

Listing 17: StatusMessage Reducer

```
import { addToArr, removeFromArr } from 'src/reduxUtils';
import * as actions from '../actions/statusMessages';

export default function statusMessageReducer(state = {}, action) {
  if (action.type === actions.ADD_STATUS_MESSAGE) {
    const statusMessages = state.statusMessages ?
       state.statusMessages.slice() : [];
    const newState = Object.assign({}, state, { statusMessages });
    return addToArr(newState, 'statusMessages', action.status);
  }
  if (action.type === actions.REMOVE_STATUS_MESSAGE) {
    return removeFromArr(state, 'statusMessages', action.status);
  }
  return state;
}
```

Für die eigentliche Darstellung der nun gespeicherten Statusmeldungen wurde eine neue Komponente erstellt. Diese beobachtet den Redux-Store und reagiert auf eintreffende, respektive entfernte Meldungen. Sobald sich die Liste der Statusmeldungen verändert, rendert die Komponente neu. Dies führt dazu, dass immer die richtigen Statusmeldungen angezeigt werden.

Listing 18: StatusMessage render

```
render() {
  const { statusMessages } = this.props;
  let elements = <></>;
  if (statusMessages && statusMessages.statusMessages) {
    elements = statusMessages.statusMessages.map((status) => {
      switch (status.statusType) {
        case Actions.STATUS_TYPE.INFO:
          return this.renderInfoContainer(status);
        case Actions.STATUS_TYPE.WARNING:
          return this.renderWarningContainer(status);
        case Actions.STATUS_TYPE.ERROR:
          return this.renderErrorContainer(status);
        default:
          return <></>;
      }
    });
  }
  return <div className="alert-container">{elements}</div>;
}
```

9.2.2 CSV-Importer

Eingabeformular

Das neu erstellte Eingabeformular wurde mit React umgesetzt und orientiert sich stark an den tabellenartigen Flask-Formularen. Damit soll sichergestellt werden, dass der Benutzer weiterhin ein vertrautes UI vor sich hat. Viele der Einstellungen wurden in einen “Advanced“-Tab verschoben, mit dem Ziel, den Benutzer nicht mit den vielen Einstellungsmöglichkeiten zu überfordern.

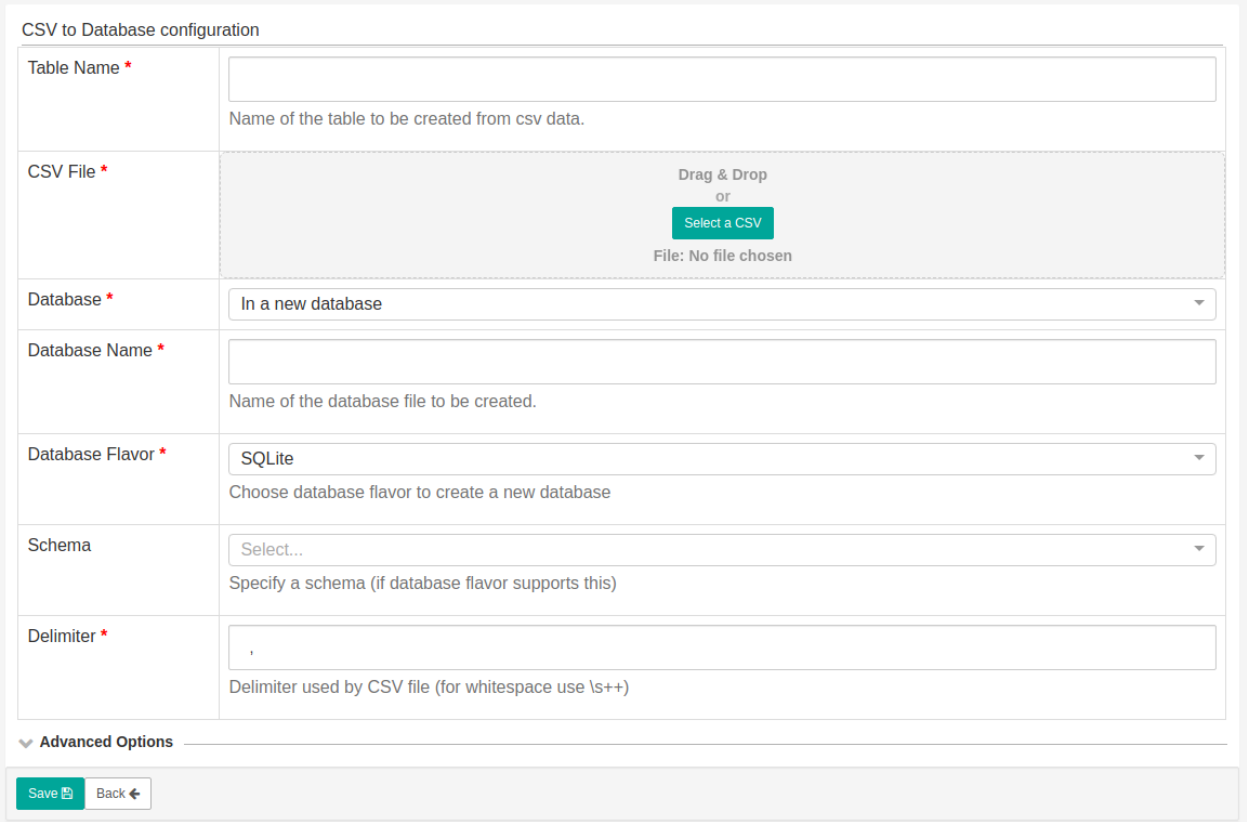


Abbildung 31: CSV-Importer Formular

Drag and Drop

Für den Drag and Drop wurde eine eigene React-Komponente gebaut. Diese setzt sich aus zwei Teilkomponenten zusammen, dem eigentlichen Input-Feld (“FileDropper“) und einem Drag and Drop Bereich (“DropArea“), welcher mit dem Input-Feld verbunden ist. Da es immer noch Browser gibt, die Drag and Drop nicht unterstützen (Beispiel: Safari), musste die Drag and Drop Komponente flexibel gestaltet werden, so dass es einen sinnvollen Fallback gibt. Für den Fallback wurde die vorherige Ansicht, wie sie im alten CSV-Importer Formular verwendet wurde, implementiert.

Listing 19: Drag and Drop Browsersupport-Check

```
export function supportsDragAndDrop() {
  const div = document.createElement('div');
  return ('ondragover' in div && 'ondrop' in div);
}
```

Um zu überprüfen, ob der Browser Drag and Drop unterstützt, wird ein HTML-Element erstellt und geprüft, ob dieses die Events “ondragstart“ und “ondrop“ kennt. Basierend auf dieser Methode werden den Komponenten Klassennamen zugewiesen, wobei mit etwas CSS die relevanten Komponenten ein- respektive ausgeblendet werden.

Listing 20: FileDropper render

```
render() {
  const {
  isRequired,
  allowedMimeTypes,
  disableClick,
  allowMultipleSelection,
  className,
  fileInputClassName,
  } = this.props;
  return (
    <div
      className={className}
      onClick={() => !disableClick && this.fileRef.click()}
      onDragOver={event => event.preventDefault()}
      onDrop={(event) => {
        this.handleFileInputChanged(event.dataTransfer.files);
        event.preventDefault();
      }}
    >
    {this.props.children}
    <input
      required={isRequired}
      type="file"
      id="file"
      accept={allowedMimeTypes && allowedMimeTypes.join(',')}
      multiple={allowMultipleSelection || false}
      ref={element => this.setFileRef(element)}
      className={supportsDragAndDrop() ? `filedropper-input ${
        fileInputClassName || ''
        }` : fileInputClassName}
      onChange={this.fileInputChanged}
    />
    </div>
  );
}
```

Die DropArea ist lediglich für die Darstellung relevant, hat aber nichts mit der eigentlichen Drag and Drop Funktionalität zu tun. Dies ist in der render-Methode des FileDroppers zu sehen. Die DropArea wird ihr als Child übergeben und innerhalb eines Containers gerendert.

Da es viele mögliche Einsatzszenarien für eine Drag and Drop Komponente gibt, wollten wir diese auch flexibel gestalten, so dass die Wiederverwendbarkeit gegeben ist. Wir ermöglichten es, die erlaubten Dateitypen (MIME Types), die maximale Dateigrösse und das Hochladen von mehreren Dateien einstellen zu können.

Listing 21: FileDropper MIME Typen

```
allowedMimeType(file) {
  const allowedMimeTypes = this.props.allowedMimeTypes || [];
  return allowedMimeTypes.length > 0
    ? allowedMimeTypes.indexOf(file.type) > -1
    : true;
}
```

Listing 22: FileDropper checkFile

```
checkFile(file, accept, reject) {
  if (!file) {
    return;
  }

  if (!this.allowedMimeType(file)) {
    reject({
      status: ErrorStatus.MimeTypeDisallowed,
      msg: `File type '${file.type}' is not allowed (${
        this.props.allowedMimeTypes
      }).`,
      fileName: file.name,
    });
    return;
  }

  if (this.props.maxSize && file.size > this.props.maxSize) {
    reject({
      status: ErrorStatus.FileSizeExceeded,
      msg: `File size (${file.size}) exceeds maximum file size (${
        this.props.maxSize
      }).`,
      fileName: file.name,
    });
    return;
  }

  accept(file);
}
```

Listing 23: FileDropper Change Handler

```
handleFileInputChanged(fileList) {
  if (!fileList) {
    return;
  }

  const files = [];
  const errors = [];

  for (let i = 0; i < fileList.length; i++) {
    this.checkFile(fileList.item(i), f => files.push(f), e =>
        errors.push(e));
  }

  if (errors.length > 0 && this.props.onError) {
    this.props.onError(errors);
  }

  if (files.length > 0) {
    this.props.onFileSelected(files);
  }
}
```

9.2.3 Geokodierung

Eingabeformular
Das Eingabeformular für die Geokodierung wurde mit den gleichen Komponenten implementiert, die bereits beim CSV-Importer verwendet wurden. Um Eingabefehler seitens des Benutzers zu reduzieren, werden die Auswahlmöglichkeiten weitestgehend vom Backend geladen und dem Benutzer in Form von so genannten “Selects“ angezeigt. Zusätzlich wird die Auswahl der verschiedenen Spalten deaktiviert, wenn noch keine Datenquelle ausgewählt wurde.

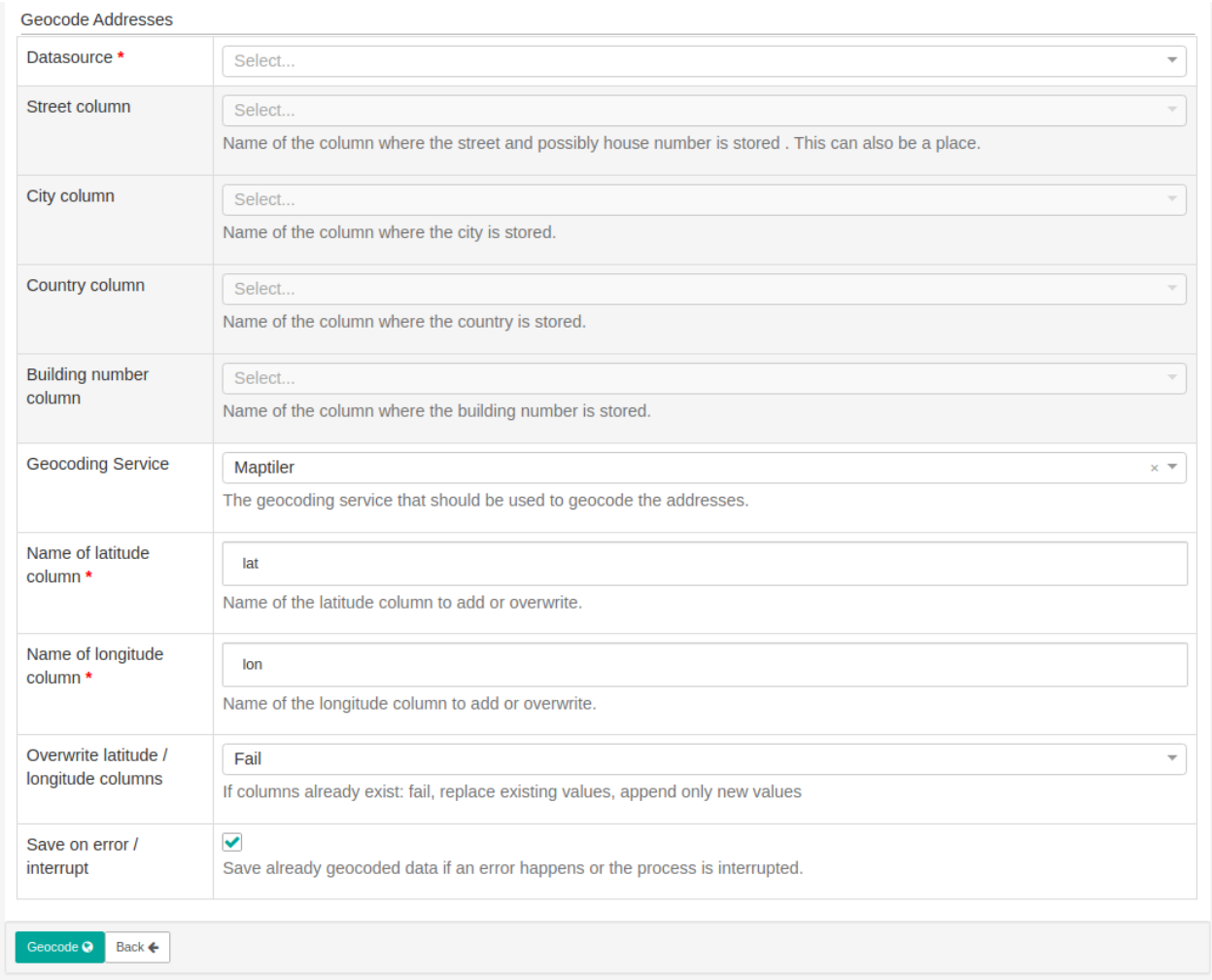


Abbildung 32: Geokodierungs-Formular

Formvalidierung

Beim Geokodierungs-Formular müssen einige Eingaben des Benutzers validiert werden. Die meisten falschen Benutzereingaben konnten durch ein geschicktes UI-Layout bereits im Vorfeld verhindert werden, z.B. durch Deaktivierung der Eingabekomponente, wenn zuerst andere Angaben getätigt werden müssen. Bei der Angabe der Spalten der adressbasierten Daten, wie der Strasse, dem Ort oder dem Land, mussten wir allerdings eine Validierung durchführen, da es dem Benutzer selbst überlassen ist, welche dieser Felder er verwenden möchte. Die Validierung wurde einfach implementiert, in dem man vor dem Übermitteln des Formulars die Eingaben überprüft und dem Benutzer entsprechend eine Fehlermeldung anzeigt, die ihm genau beschreibt, was zu tun ist.

Fortschrittsanzeige

Die Fortschrittsanzeige wurde so umgesetzt, dass in regelmässigen Abständen ein Request an das Backend gesendet wird. Dieses antwortet mit einem JSON, welches sowohl den Fortschritt als auch die Relevanz der Geolokationen enthält. Die gefundenen, respektive nicht gefundenen Geolokationen sind in drei Kategorien aufgeteilt: “Sicher“, “Unsicher“ und “Fehlerhaft“. Das Frontend zeigt basierend auf diesen Daten einen Fortschrittsbalken sowie eine Liste, basierend auf diesen drei Kategorien, an. Zusätzlich enthält die Seite einen Button, mit dem die Geokodierung abgebrochen werden kann. Da diese im Backend stattfindet wird dazu ebenfalls ein Request an dieses gesendet.

Abbildung 33: Geokodierung Fortschrittsanzeige

Cypress Test

Auch für die Geokodierung wollten wir Integration Tests mit Cypress erstellen, welche den gesamten Ablauf der Geokodierung testen. Allerdings ist das Geokodierungs-Frontend sehr reaktiv gestaltet. So werden die Auswahlfelder für die einzelnen Spalten erst aktiviert, wenn eine Tabelle ausgewählt wurde. Mit Cypress konnten wir diese Selektion mit anschliessender Wartezeit nicht umsetzen. Entweder konnten wir die Selektion nicht setzen oder die Spalten wurden auch nach 30 Sekunden Wartezeit vom Backend nicht zurück gesendet.
Aufgrund des Projektaufbaus konnten wir die Tests nicht lokal ausführen, was das Debugging unmöglich machte. Wir konnten die Tests lediglich auf dem Travis Server ausführen lassen, mit einen Build. Durch die langen Ausführungszeiten von Cypress, blockierten die Versuche die anderen Builds.
Über die Unit-Tests konnten wir die Geokodierung durchgängig testen. Alle Komponenten, vom Progress bis zum Interrupt werden abgedeckt.
Wir haben deshalb auf die Integration Tests für die Geokodierung verzichtet, weil der zusätzliche Arbeitsaufwand, den geringen Mehrwert der Tests, nicht gerechtfertigt hätte.

10 Infrastruktur

Eine gute Infrastruktur ist die Basis eines guten Projektes. Deshalb haben wir schon früh viel Zeit investiert, unsere Infrastruktur zu planen und einzurichten. Im Abschnitt 7 sind wir bereits darauf eingegangen, weshalb wir uns für gewisse Technologien entschieden haben. In diesem Kapitel werden wir im Detail unsere Client- und Server-Infrastruktur beschreiben sowie auf unsere Continuous Integration (CI) und Continuous Deployment (CD) eingehen.

10.1 Client-Infrastruktur

Für das Entwickeln auf unseren Windows- Laptops, setzten wir eine Linux-VM (Ubuntu 18.04 LTS[46]) ein. Auf dieser VM lag das lokale Repository, die Python-IDE PyCharm [47], der Quelltext-Editor Visual Studio Code [48] und der Chrome Browser [49]. Um das Frontend effizient bearbeiten und debuggen zu können, wurden ausserdem diverse Extensions installiert:

Visual Studio Code	Chrome
JSLint	React DevTools
TSLint	Redux DevTools
Prettier	
GitLens	

Abbildung 34: VSCode & Chrome Extensions

Über PyCharm konnte innerhalb der VM sowohl das Backend als auch das Frontend gestartet werden. Für das Debuggen des Backends wurde ebenfalls PyCharm verwendet. Das Frontend wurde im Browser direkt über die Entwicklertools, respektive die installierten Extensions debugged.
Für die Dokumentation des Projekts wurde LaTeX [50] verwendet. Da jeder von uns andere Vorlieben bezüglich LaTeX Bearbeitungstools hatte, war jeder Entwickler selbst dafür verantwortlich, dass auf seinem Rechner die entsprechenden Tools installiert waren.

10.2 Server-Infrastruktur

Für die Server-Infrastruktur setzten wir einige Cloud-Dienste sowie einen eigenen Linux-Server ein. In diesem Kapitel werden die einzelnen Komponenten im Detail beschrieben, wohingegen im Kapitel 10.3 auf das Zusammenspiel dieser Komponenten eingegangen wird.

10.2.1 Azure DevOps

Für unsere Organisation, das Sprint-Planning, die Zeiterfassung, Notizen im Wiki-Format, die Dokumentationsablage sowie einige Build und Deployment Pipelines setzten wir Azure DevOps [51] ein. Azure DevOps ermöglichte es, eine agile Planung durchzuführen und gab uns die Möglichkeit, den Überblick über unsere offenen Tasks sowie das Zeitmanagement zu behalten.
Ausserdem bot Azure DevOps verschiedene Pipelines an, welche wir für das CI/CD benutzen konnten.

[46] Ubuntu 18.04 LTS: http://releases.ubuntu.com/18.04/
[47] PyCharm: https://www.jetbrains.com/pycharm/
[48] VS Code: https://code.visualstudio.com/
[49] Chrome: https://www.google.com/intl/de/chrome/
[50] LaTeX: https://www.latex-project.org/
[51] Azure DevOps: https://azure.microsoft.com/en-us/services/devops/

10.2.2 GitHub

GitHub [52] verwendeten wir für die Ablage unseres Code-Repository. GitHub kann sowohl von Travis-CI als auch von Azure DevOps angesteuert werden, was es uns auf einfache Art und Weise ermöglichte, GitHub in unsere CI/CD Pipelines zu integrieren.

10.2.3 Travis-CI

Da Superset selbst Travis-CI [53] nutzt, um den Code automatisch zu builden (CI) und sicherzustellen, dass bei einem Pull-Request alles in Ordnung ist, setzten auch wir diesen Cloud-Dienst dafür ein. Travis-CI kann mit GitHub verknüpft und über YAML Dateien konfiguriert werden. Travis-CI buildet dann den Code auf ihren eigenen Servern und kommuniziert das Resultat zurück an GitHub.

10.2.4 Dockerhub

Das Deployment basierte auf Docker-Images, welche auf Dockerhub [54] liegen. Für unsere Arbeit wurden drei verschiedene Images verwendet. Das erste war Redis [55], welches vor allem als Cache und Message-Broker, innerhalb der Docker-Infrastruktur verwendet wurde. Das zweite war die PostgreSQL-[56]Datenbank und zu guter Letzt unser Fork von Superset.

10.2.5 Linux-Server

Für das Hosting setzten wir einen eigenen Linux-Server ein, welcher von der HSR zur Verfügung gestellt wurde. Auf dem Server war ein Azure Pipelines Agent [57] sowie Docker [58] installiert. Über den Azure Pipelines Agent wurde die Kommunikation mit unserem Azure DevOps sichergestellt. Über Docker wurden die verschiedenen Docker-Images gehostet, wobei das Superset Docker Image eine Schnittstelle nach aussen zur Verfügung stellte, über welche die Anwendung erreicht und genutzt werden konnte.

10.3 Continuous Integration & Continuous Deployment

Unser Projekt war in zwei Repositories aufgeteilt: Code und Dokumentation. Das Dokumentations-Repository sowie dessen Build- und Deployment-Pipelines lagen im Azure DevOps. Das Code-Repository lag auf GitHub, wobei hier die Continuous Integration von Travis-CI übernommen wurde. Das Deployment wurde wiederum über Azure DevOps gemacht.

[52] GitHub: https://github.com/
[53] Travis-CI: https://docs.travis-ci.com/user/for-beginners/
[54] Dockerhub: https://hub.docker.com/
[55] Redis: https://redis.io/
[56] PostgreSQL: https://www.postgresql.org/
[57] Azure Pipelines Agent: https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops
[58] Docker: https://www.docker.com/

10.3.1 Code

Jeder Pull-Request stiess einen Build bei der Travis-CI an, welche das Repository buildete, die Unit-Tests durchführte und letztlich das Resultat an GitHub zurücklieferte. Jede Änderung am master-Branch löste das Deployment aus. Zuerst wurde im Azure DevOps das Superset-Image erstellt und mit dem Tag 'master' auf Dockerhub hochgeladen. War dies erfolgreich, wurde die zweite Phase eingeleitet. Azure DevOps verband sich über den Azure Pipelines Agent, welcher sich auf dem Linux Server befand, auf den Server. Dort wurden zuerst alle Docker-Instanzen beendet. Waren diese heruntergefahren, wurden die neusten Images von Dockerhub heruntergeladen und die Container erneut hochgefahren. Am Schluss dieser Phase war die Anwendung wieder verfügbar und konnte benutzt werden. Das nachfolgende Diagramm 35 zeigt diese Zusammenhänge nochmals auf.

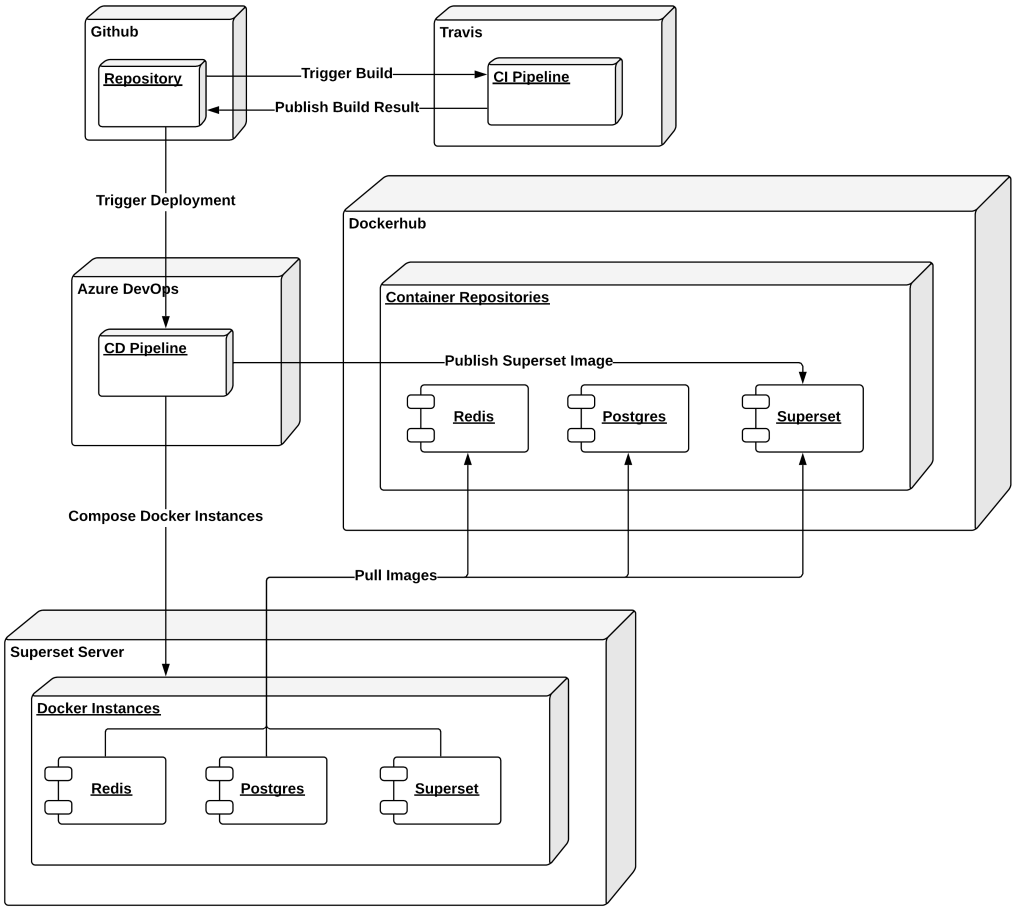


Abbildung 35: Deployment-Diagramm Code

10.3.2 Dokumentation

Bei dem Dokumentations-Repository lief es ähnlich ab. Jeder Pull-Request stiess erstmal einen Build an, welcher das Resultat wiederum in den Pull-Request schrieb. Erst wenn dieser Build erfolgreich war, konnte der Pull-Request gemerged werden. Gleichzeitig wurden die vom Build generierten PDFs als Artifacts zur Verfügung gestellt.

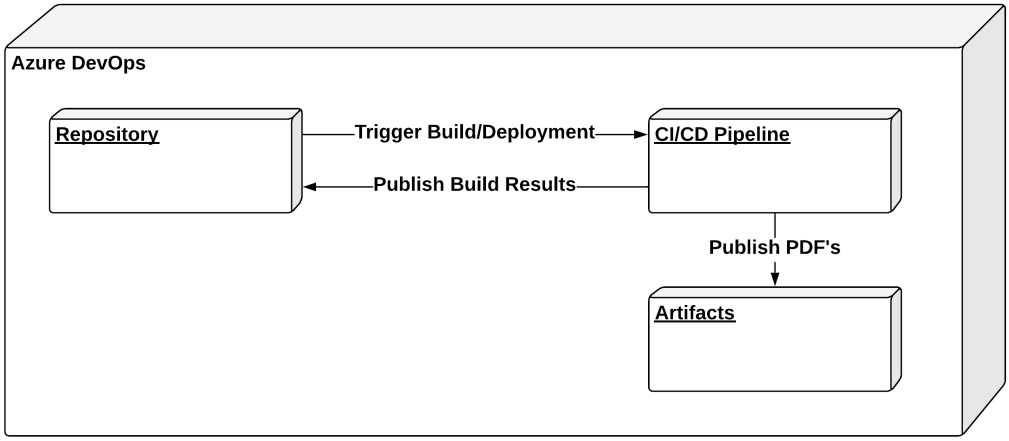


Abbildung 36: Deployment-Diagramm Dokumentation

11 Resultate und Weiterentwicklung

Das Ergebnis und weitere Verbesserungen unserer Arbeit werden im nachfolgenden erläutert.

11.1 Resultate

In diesem Abschnitt gehen wir auf die Resultate der Umsetzung ein.

11.1.1 CSV-Importer

Mit unserer CSV-Importer-Überarbeitung haben wir die Grundlage für die Modernisierung der User-Interfaces gelegt. Mit Hilfe unserer wiederverwendbaren React-Komponenten, können die vielen verschiedenen Flask-Formulare umgebaut werden. Die neu hinzugefügte REST-Schnittstelle im Backend empfängt die Daten des Frontends und verarbeitet diese. Die neue Schnittstelle wurde von der mittlerweile sehr grossen Datei “core.py“ getrennt. Dank unseren Verbesserungen kann das Frontend vom Backend getrennt werden, was das gemeinsame Entwickeln an einem Feature vereinfacht. Zudem erleichterte es uns die Erweiterung der Unit-Tests. Wir konnten zeigen, dass man durch die Trennung des Frontends vom Backend und mit ein wenig Zusatzaufwand robustere Software bauen kann.

Durch die Anpassungen im Formular, erhält der Benutzer einen schnelleren Überblick über die geforderten Daten. Mit nur vier Klicks und einem Drag and Drop kann er eine CSV-Datei in eine Tabelle importieren. Die zusätzlichen optionalen Felder wurden elegant hinter einem aufklappbaren Menü versteckt. Dass der Benutzer, ohne technisches Wissen, eine neue Datenbank für seine importierten Daten erstellen kann, verbessert die Benutzerfreundlichkeit weiter. Dabei kann er zwischen SQLite und PostgreSQL wählen. Ausserdem werden seine Eingaben früher überprüft und bei Fehler bessere und verständlichere Fehlermeldungen angezeigt als im bisherigen CSV-Importer Formular.

11.1.2 Geokodierung

Mit dieser neuen Funktionalität wird Superset konkurrenzfähiger, gegenüber den anderen BI-Tools. Der Benutzer kann seine Daten direkt in Superset aufbereiten, ohne dass er vorher einen externen Service in Anspruch nehmen muss.

In Anlehnung an den CSV-Importer wurde auch die Geokodierung in eine eigene Datei ausgelagert und das Frontend vom Backend getrennt. Viele der React-Komponenten konnten vom CSV-Importer wiederverwendet werden.
Auch die Umsetzung der eigentlichen Adressgeokodierung wurde in eine separate Utils-Datei verschoben. Die Wartbarkeit wurde mit Hilfe des Factory-Method Pattern[13] sichergestellt. Das Erweitern weiterer Geokodierungs-Services hat sich dadurch vereinfacht. Bei der Implementierung der Eingabemaske achteten wir darauf, dass wir dem Benutzer viele Freiheiten und Entscheidungsmöglichkeiten geben. Dank der Auswahlhilfen in der Eingabe-Maske werden fehlerhafte oder ungültige Eingaben verhindert. Ausserdem werden verständliche Fehlermeldungen dem Benutzer angezeigt. Nicht nur die Select-Listen steigern die Benutzerfreundlichkeit, sondern auch die kontinuierliche Anzeige des Fortschritts. Die Fortschrittsanzeige zeigt an, wie viele Daten bereits geokodiert wurden und deren Relevanz (die Koordinate stimmt zu 80-100% mit der Adresse überein, der Service ist sich zu mehr als 50% sicher die richtige Koordinate gefunden zu haben oder weniger). Der Benutzer kann den Prozess jederzeit beenden und seine Daten oder Einstellungen gegebenenfalls anpassen. Ausserdem kann er entscheiden, ob die bereits geokodierten Daten gespeichert werden sollen oder ob der nächste Vorgang alle Daten erneut geokodieren soll. Zu guter Letzt wurden die Ausführungsrechte der Geokodierung auf den Administrator beschränkt.

11.2 Weiterentwicklung

In diesem Abschnitt erläutern wir die Verbesserungsmöglichkeiten zu unserer Umsetzung.

11.2.1 CSV-Importer

Sobald eine offizielle Design-Spezifikation von Superset definiert und veröffentlicht wird, kann das Design des CSV-Importers entsprechend überarbeitet werden. Die bisher genutzte Tabellen-basierte Darstellung ist nicht mehr modern und sollte in Zukunft abgelöst werden. Ein weiteres Verbesserungspotential ist die grosse Anzahl der Einstellungsmöglichkeiten. Es wäre empfehlenswert, den CSV-Importer anders zu lösen. Dabei sollte es das Ziel sein, dass der Benutzer möglichst wenige Eingaben tätigen muss. Anstelle des Formulars könnte es eine Art “Prozess“ sein, der den Benutzer über mehrere Seiten leitet. Als Vorbild könnte man den Installationsprozess eines Windows-Programmes nehmen, bei dem es einen Schnelldurchlauf und einen Experten-Modus gibt.
Als weitere Verbesserung könnte man verschiedene Encodings der CSV-Dateien anbieten. Aus zeitlichen Gründen konnten wir das bisher unterstützte Encoding von UTF-8 nicht erweitern. Ausserdem wäre die Erweiterung für weitere Datenbank-Typen besser mit dem Factory-Method Pattern[13] gelöst. Des Weiteren müsste man, aus sicherheitstechnischen Gründen, die Aufrufe mit SQLAlchemy besser gegen SQL-Injections schützen. Zurzeit könnte der Request an das Backend abgefangen und manipuliert werden. Aufgrund anderer, höher priorisierter Anforderungen unseres Auftraggebers und der Tatsache, dass Superset unternehmensintern eingesetzt wird, setzten wir diese Massnahmen nicht um.

11.2.2 Geokodierung

Gegenwärtig werden die API-Aufrufe synchron ausgeführt. Das heisst, es kann jeweils nur ein Benutzer die Geokodierung ausführen. Mit dem Framework Celery könnten mehrere Benutzer gleichzeitig ihre Adressen geokodieren.
Ausserdem könnte der Benutzer am Ende der Geokodierung per E-Mail informiert werden, dass diese beendet wurde und die Koordinaten nun zur Verfügung stehen. Dadurch könnte der Benutzer die Superset-Seite verlassen und wüsste trotzdem, wann seine Daten fertig geokodiert wurden.
Eine weitere Massnahme zur Steigerung der Benutzerfreundlichkeit wäre, bei Unsicherheiten dem Benutzer eine Auswahl möglicher Treffer zu präsentieren, bei denen er dann eine Koordinate auswählen oder sogar eine eigene setzen könnte. Damit könnte der Benutzer selbst die korrekte Adresse wählen, was die Datenqualität wesentlich verbessern würde. Als Zusatz könnte man die Adressen auf einer Karte anzeigen, anstelle der textuellen Anzeige. Das würde dem Benutzer die Auswahl noch mehr erleichtern.
Unsere Geokodierung unterstützt nur zwei von mehr als 25 verschiedenen Geokodierungs-Services. Durch die Einbindung von GeoPy und der Konfiguration verschiedener API-Keys, könnten weitere Services verwendet und dem Benutzer mehr Entscheidungsfreiheiten ermöglicht werden. Dabei ist aber zu beachten, dass viele der Geokodierungs-Services das Abspeichern der Resultate nicht erlauben. Hier sollten die Nutzungsbedingungen des jeweiligen Anbieters beachtet werden.

12 Projektmanagement

Zur Verwaltung und Planung der Arbeitspakete, Sprints, Zeitbuchungen und der Dokumentation wird Azure DevOps von Microsoft eingesetzt. Dieser Service unterstützt die agile Methode Scrum, von der wir gewisse Aspekte verwendeten. Die aufgewendete Zeit wird direkt auf den Arbeitspaketen verbucht und automatisch in den Timesheets festgehalten. Die Abstufung wird 15 Minuten genau erfasst.

12.1 Entwicklung

Der von uns eingesetzte Build Server Travis CI kann nur von GitHub Repositories lesen und diese builden. Deshalb liegt das Repository für den Source Code in einem öffentlichen GitHub Repository. Wir erstellten pro neue Funktionalität einen Branch, der dann für alle Entwickler gesperrt wurde. Änderungen wurden über Pull Requests in den abgezweigten Branch eingefügt, welche zuvor von einem anderen Teammitglied überprüft und kommentiert wurden. Dadurch stellten wir sicher, dass der Code der Definition of Done entspricht, Fehlerquellen gesenkt und gleichzeitig der Wissensaustausch gefördert wurde. Der Source Code wurde mit jedem Pull Request automatisch gebuildet, getestet und publiziert. Infolgedessen steigerte sich die Entwicklerproduktivität und repetitive Arbeiten wurden automatisiert. Die Branchnamen für die Pull Requests in der Dokumentation lauteten folgendermassen: “userstory/xxx“. Diejenigen im Code nannten wir “_csv-importer/xxx“ oder “_geocoding/xxx“.

Mittels Pylint, Black, mypy und isort wurden sowohl die Codequalität und Supersetstandards eingehalten als auch Fehler vorgebeugt. Zusätzlich wurden zu jedem Arbeitspaket Unit-Tests erstellt, um die Funktionsweise der Implementierung zu gewährleisten.

Definition of Done Ein Arbeitspaket musste vor dem Abschluss folgende Kriterien erfüllen:

- Code weist keine syntaktischen Fehler auf
- Code ist nach den Python Style Guidelines von Superset formatiert
- Unit-Test zu dem Arbeitspaket wurden erstellt
- Alle Unit-Tests sind erfolgreich
- Dokumentation wurde entsprechend erstellt oder erweitert
- Pull Request wurde von einem anderen Teammitglied überprüft und akzeptiert
- Build war erfolgreich
- Stakeholder hat das Arbeitspaket gesehen und sein Einverständnis gegeben

12.2 Rollen und Verantwortlichkeiten

Wir entschieden uns dafür, keinen festen Projektleiter festzulegen. Planungen und Entscheidungen wurden jeweils im Team getätigt. Die Verantwortlichkeiten legten wir in 5 Teilbereiche fest:

- Frontend: Sascha Gschwind
- Backend: Michelle Kunz, Renato Venzin
- CI/CD: Sascha Gschwind
- Dokumentation: Michelle Kunz
- Qualitätssicherung: Renato Venzin, Michelle Kunz

12.3 Prozessmodell

Als Prozessmodell setzten wir eine Kombination aus Rational Unified Process (RUP) und Scrum ein (genannt Scrum+). Durch RUP konnten wir das Projekt in mehrere Phasen unterteilen und koordinieren, während Scrum uns eine hohe Flexibilität durch adaptives Planen ermöglicht hat und in Meetings eine hohe Transparenz bat. Zudem erlaubte die zeitnahe Inkrementation den einfacheren Umgang mit kurzfristigen Problemen und neuen Anforderungen.

12.4 Planung

Die Planung wurde bei Änderungen im Projekt laufend nachgeführt. Nachstehend werden die einzelnen Phasen genauer erläutert.

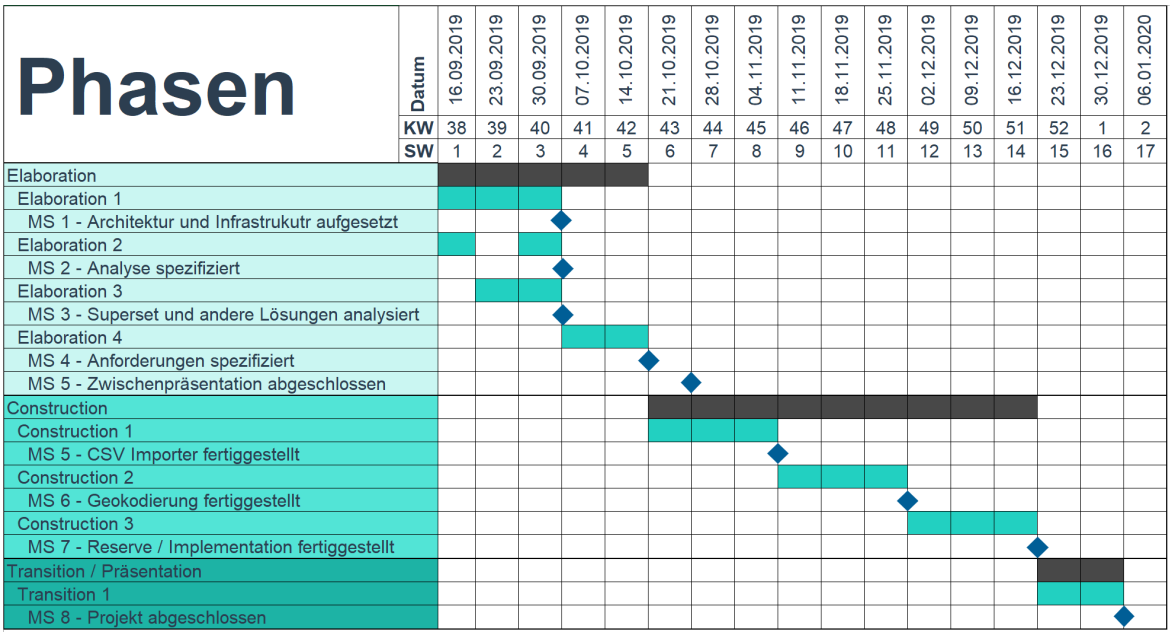


Abbildung 37: Projektplan mit den einzelnen RUP-Phasen

12.4.1 Inception

Der Projektantrag und die -vision reichte unser Betreuer Herr Stefan Keller bereits vor Beginn dieser Arbeit ein. Deshalb werden wir hier nicht weiter auf diese Phase eingehen.

12.4.2 Elaboration

Die Elaboration Phase dauerte fünf Wochen. Während dieser Phase setzten wir die Architektur und Infrastruktur auf, um Erweiterungen am Apache Superset vornehmen zu können und diese gegen die Tool-Chain von Superset zu testen. Ausserdem analysierten wir die Webapplikation Apache Superset und ähnliche Lösungen und verglichen diese. Anschliessend vertieften wir die Anforderungen an Superset und hielten diese fest.

In dieser Phase präsentierten wir zudem die Zwischenpräsentation der Bachelorarbeit für den Gegenleser. Allfällige Fragen und Unklarheiten konnten sogleich analysiert und bearbeitet werden.

12.4.3 Construction

In der neunwöchigen Construction Phase wurden die Anforderungen implementiert und getestet. Die zwei geplanten Features wurden jeweils nach drei Wochen Entwicklungszeit released und vorgestellt. Die letzten drei Wochen wurden als Reserve für Bugfixing und Refactorings eingeplant.

12.4.4 Transition

Die Dokumentation und das Abstract wurden in den letzten Wochen in den finalen Zustand versetzt. Die Bachelor-Präsentation und das Plakat für die Vorstellung der Bachelor Themen wurden ebenfalls in dieser Zeit vorbereitet. Mit der Abgabe der Arbeit wurde diese Phase beendet.

12.5 Stakeholder

Wir haben 4 Stakeholder identifiziert. Diese stehen im folgenden Interesse/Einfluss Verhältnis:

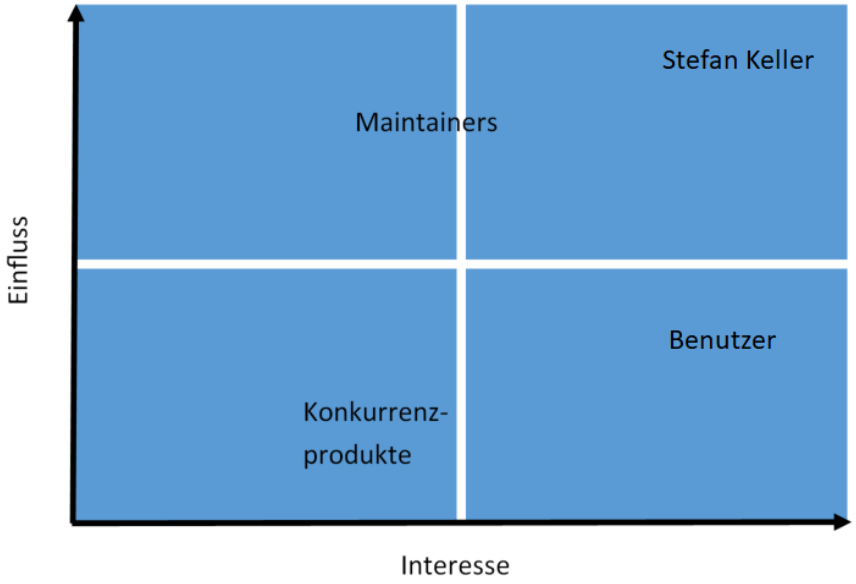


Abbildung 38: Stakeholder

Stefan Keller Herr Keller ist gleichzeitig auch Product Owner und dementsprechend einflussreich und am Projekt interessiert.

Maintainers Diese Gruppe ist an zusätzlichen Features und Bugfixes interessiert, aber nicht speziell an unserem Projekt. Sie sind jedoch auch sehr einflussreich, da Sie die Pull-Requests auch ablehnen können, wodurch das Feature nicht in den Superset master integriert wird.

Benutzer Zu den Benutzern gehören zum Beispiel die Institutsmitarbeiter der HSR, welche Superset verwenden. Sie haben einen gewissen Einfluss, bezüglich den umzusetzenden Features, die sie sich wünschen.

Konkurrenten Die Konkurrenten sind nicht besonders interessiert an unserem Projekt. Sie haben lediglich Interesse an den Neuerungen ihrer Konkurrenz. Da wir lediglich zwei neue Funktionalitäten hinzufügen, hält sich das Interesse in Grenzen.

12.6 Risikoanalyse

Bei der Risikoanalyse geht es um die technischen Risiken, die im Laufe des Projekts eintreten könnten. Allgemeine Risiken wie Hardware Ausfall, Ausfall von Git oder Git-Komponenten, Krankheit eines Teammitglieds etc. werden nicht beschrieben. Diese projektspezifischen Risiken können den Projektverlauf erheblich beeinflussen.

Nr.	Titel	max. Schaden	Eintritts-wahrscheinlichkeit	Gewichteter Schaden
R1	Unklare Anforderungen	8h	20%	1.6
R2	Unterschätzung der Komplexität	16h	40%	6.4
R3	Feature nicht erwünscht	24h	5%	1.2
R4	Verspätete Fehlererkennung	6h	30%	1.8
R5	Keine Dokumentation vorhanden	24h	70%	16.8

Abbildung 39: Risiko Analyse

Den Risiken aus 39 kann vorgebeugt werden. Die dazugehörigen Massnahmen sind in den folgenden Abschnitten festgehalten.

12.6.1 Unklare Anforderungen

Beschreibung Unsere Anforderungen entsprechen nicht den Kriterien von SMART.

Vorbeugung Die Anforderungen werden anhand der SMART Kriterien definiert.

Verhalten beim Eintreten Wir werden die Anforderungen mit unserem Product Owner anschauen und klären, wie wir diese formulieren sollen.

12.6.2 Unterschätzung der Komplexität

Beschreibung Die Features stellen sich als komplexer heraus als ursprünglich gedacht.

Vorbeugung Die Elaboration Phase wird ernstgenommen und zur Sicherheit in der Construction Reservezeit eingeplant.

Verhalten beim Eintreten Wir werden die Reserve aufbrauchen und weitere geplante Implementationen neu priorisieren, verschieben oder streichen.

12.6.3 Feature nicht erwünscht

Beschreibung Die Superset Maintainer lehnen einen Pull-Request ab, weil sie das Feature nicht benötigen oder dieses bereits umgesetzt wurde.

Vorbeugung Die SIPs werden frühzeitig aufgegeben, so dass sie Rückmeldungen geben und Fragen zu unseren Features stellen können. Ausserdem werden wir jedes Feature in einem separaten Branch erstellen und nur diesen in ihren master integrieren.

Verhalten beim Eintreten Wir werden die geplanten Implementationen trotzdem durchführen, allerdings ohne diese in ihren master zu mergen. Es wird eine neue Version von Superset geben, die nur die HSR verwendet.

12.6.4 Verspätete Fehlererkennung

Beschreibung Man testet neue Features nicht und merkt erst spät, dass man an einem Ort Fehler eingebaut hat.

Vorbeugung Die Testfälle werden jeweils bei der Umsetzung der Features implementiert.

Verhalten beim Eintreten Wir werden die Fehler beheben und sogleich Tests schreiben, um sicherzustellen, dass der Fehler endgültig eliminiert wurde.

12.6.5 Keine Dokumentation vorhanden

Beschreibung Superset bietet keine geeignete Dokumentation an, in der erklärt wird, warum sie was gewählt haben, bzw. wie etwas funktioniert.

Vorbeugung Im Internet wird nach Dokumentationen von Apache, Airbnb und anderen Personen zu Superset gesucht.

Verhalten beim Eintreten Wir stellen Fragen, welche die Weiterentwicklung betreffen in den Slack-Channel. Alles andere müssen wir uns selbst erarbeiten.

Dies ergibt die folgende Risiko-Matrix:

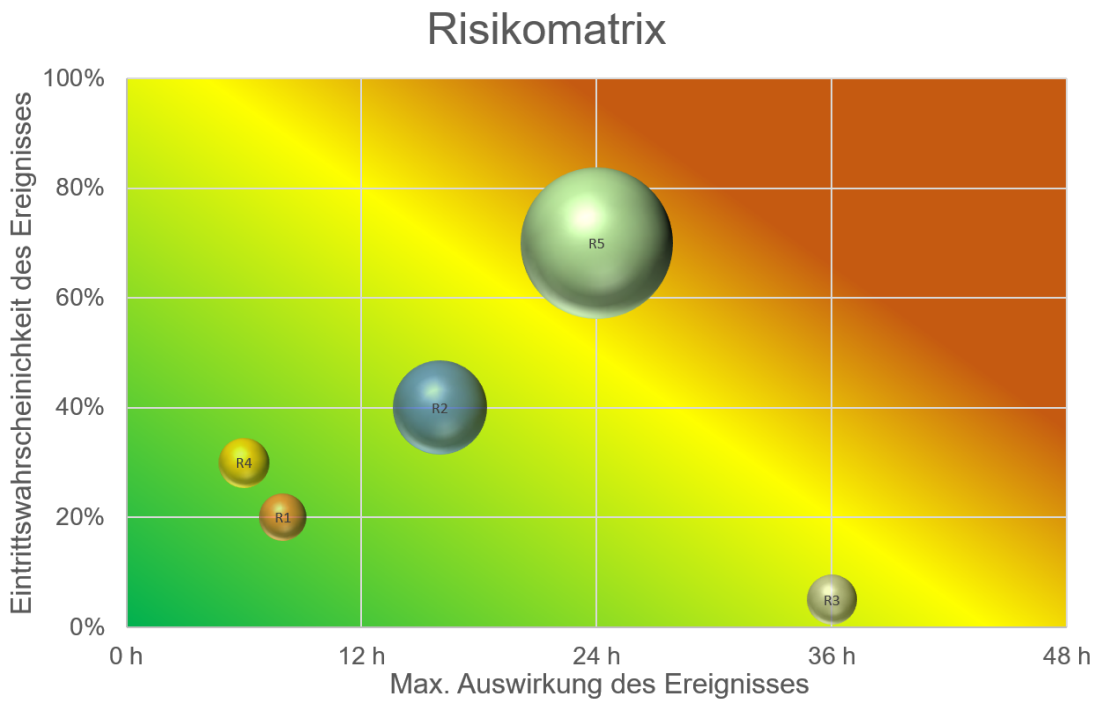


Abbildung 40: Risiko Matrix zur vorhergehenden Risikoanalyse

Es ergeben sich daraus zwei Risiken, welche gefährlich sein könnten: R3 (Feature nicht erwünscht) und R5 (Keine Dokumentation vorhanden). Um R3 vorzubeugen versuchen wir so früh wie möglich die SIPs aufzugeben. R5 ist abhängig von anderen Personen. Hat niemand Dokumentationen zu Superset verfasst, sind wir auf die Hilfe der Slack-Community von Superset angewiesen.

Teil III

Anhänge

Literatur

[1] General Networks. *What is Big Data? And Why Is It Important to Me?* 31. März 2015. URL: https://gennet.com/big-data/big-data-important/ (besucht am 21. 10. 2019).

[2] Tableau. *What is business intelligence? Your guide to BI and why it matters*. 2019. URL: https://www.tableau.com/learn/articles/business-intelligence (besucht am 21. 10. 2019).

[3] Susana Santos. *Apache Superset Open Source BI: almost the alternative to Tableau*. 17. Sep. 2018. URL: https://www.xpand-it.com/2018/09/17/apache-superset-open-source-bi/ (besucht am 21. 10. 2019).

[4] InData Labs. *Superset: benefits and limitations of the open source data visualization tool by Airbnb*. 17. Mai 2017. URL: https://medium.com/@InDataLabs/superset-benefits-and-limitations-of-the-open-source-data-visualization-tool-by-airbnb-8dc8ac81efa9 (besucht am 21. 10. 2019).

[5] Maxime Beauchemin. *Superset: Airbnb's data exploration platform*. 30. März 2016. URL: https://medium.com/airbnb-engineering/caravel-airbnb-s-data-exploration-platform-15a72aa610e5 (besucht am 21. 10. 2019).

[6] Maxime Beauchemin. *The history and anatomy of Apache Superset*. 2019. URL: https://www.datacouncil.ai/hubfs/DataEngConf/Data%20Council/Slides%20SF%2019/The%20history%20and%20anatomy%20of%20Apache%20Superset.pdf (besucht am 21. 10. 2019).

[7] IT-Service24. *EVA Prinzip Definition & Begriffserklärung*. 2019. URL: https://www.it-service24.com/lexikon/e/eva-prinzip/ (besucht am 21. 10. 2019).

[8] Michael Friendly. *A brief history of data visualization*. Springer, 2008.

[9] OmniSci. *Location Intelligence*. 2019. URL: https://www.omnisci.com/learn/resources/technical-glossary/location-intelligence (besucht am 21. 10. 2019).

[10] Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. Bd. 1. Esri Press, CA, 2010.

[11] Edward R. Tufte. *The visual display of quantitative information*. Bd. 2. Graphics press Cheshire, CT, 2001.

[12] Dona M. Wong. *The Wall Street Journal guide to information graphics: The dos and don'ts of presenting data, facts, and figures*. WW Norton, 2010.

[13] Martin Fowler u. a. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2009.

[14] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: https://martinfowler.com/articles/injection.html.

[15] Microsoft (several Contributors). *Spatial Data Types Overview*. 2016. URL: https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-data-types-overview?view=sql-server-ver15.

[16] Ken Schwaber und Jeff Sutherland. *The Scrum Guide*. 1. Nov. 2017. URL: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100 (besucht am 21. 10. 2019).

[17] Philippe Kruchten. *Rational Unified Process*. Techn. Ber. Rational, 2001. URL: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf?mhsrc=ibmsearch_a&mhq=rup (besucht am 21. 10. 2019).

Abbildungsverzeichnis

Listings

Glossar

BI Kurzform für Business-Intelligence. 2–6, 8–11, 65

Business-Intelligence Ein Business Intelligence-Werkzeug (BI-Tool) ist ein datengesteuertes Entscheidungsunterstützungssystem, das Datenerfassung, Datenspeicherung und Wissensmanagement mit Analyse kombiniert, um Input für den Entscheidungsprozess zu liefern. Der Begriff stammt aus dem Jahr 1989; davor waren viele seiner Merkmale Teil von Führungsinformationssystemen. Business Intelligence legt den Schwerpunkt auf die Analyse grosser Datenmengen über Fakten und Abläufe[**Negash2008**]. 1–3, 12, 76

Chart Ein Chart ist ein Diagramm, dass verschiedene Typen annehmen kann. Früher nannte Superset die Charts 'Slices'. Im Code wird noch heute Slice anstelle von Chart verwendet . 2, 5, 6, 21, 29, 30

Dashboard Eine Seite, die den Status der Metriken und Key Performance Indikatoren visuell darstellen. 2, 5, 6, 9, 17, 29, 30

Definition of Done Akzeptanzkriterien, die erfüllt sein müssen, bevor eine User Story abgeschlossen werden darf. 67

DML Data Manipulation Language: Sprache, welche die Abänderung von Daten in einer Datenbank erlaubt, bei SQL wären dies z.B. die Statements: UPDATE, INSERT und DELETE. . 34

DTO Data transfer object. Ein Objekt, welches dazu dient, Daten zwischen zwei Endpunkten zu transportieren, dadurch können verschiedene nötige Informationen gebündelt werden was die Anzahl an Requests senkt und so Zeit spart. 31

LFI Eine Schwachstelle, bei welcher ein Angreifer auf eine Datei zugreifen kann, auf die er nicht zugreifen können sollte. Dadurch könnte z.B. der Inhalt verschiedenster Dateien ausgelesen werden. Dies ist unter Umständen möglich, wenn Benutzereingaben nicht überprüft werden. Beispiel: „../../../etc/passwd“. . 31

Lint Eine Software zur statischen Code-Analyse. 22

Location Intelligence Synonym für Spatial Intelligence. Siehe Spatial Intelligence. 10

Overlay-Karte Eine Overlay-Karte ist eine Karte (üblicherweise eine Vektorkarte), die man über eine Rasterkarte legt. Ein Beispiel für eine Overlay-Karte wäre eine Satellitenkarte. 2

Pull Request Eine Methodik in Azure DevOps für die Zusammenarbeit in git. 11, 24, 67

RESTful Der Begriff RESTful basiert auf dem Programmierparadigma REST, was für Representational State Transfer steht. REST stellt eine einfache Alternative zu ähnlichen Verfahren wie SOAP oder WSDL dar. Damit eine Schnittstelle als REST-Schnittstelle bezeichnet werden kann muss sie einige Bedingungen erfüllen: Adressierbarkeit, Zustandslosigkeit, einheitliche Schnittstelle sowie Entkopplung von Ressourcen und deren Repräsentation (HATEOAS - Hypermedia as the Engine of Application State). . 31

Spatial Intelligence Die Fähigkeit, dreidimensionale Bilder und Objekte zu verstehen, nennt man Location Intelligence, auch Spatial Intelligence genannt.[9]. 6, 7, 10, 76

Windows-Subsystem für Linux Ein Feature von Windows 10, mit dem man native Linux-Kommandozeilen Tools direkt auf Windows ausführen kann. 23