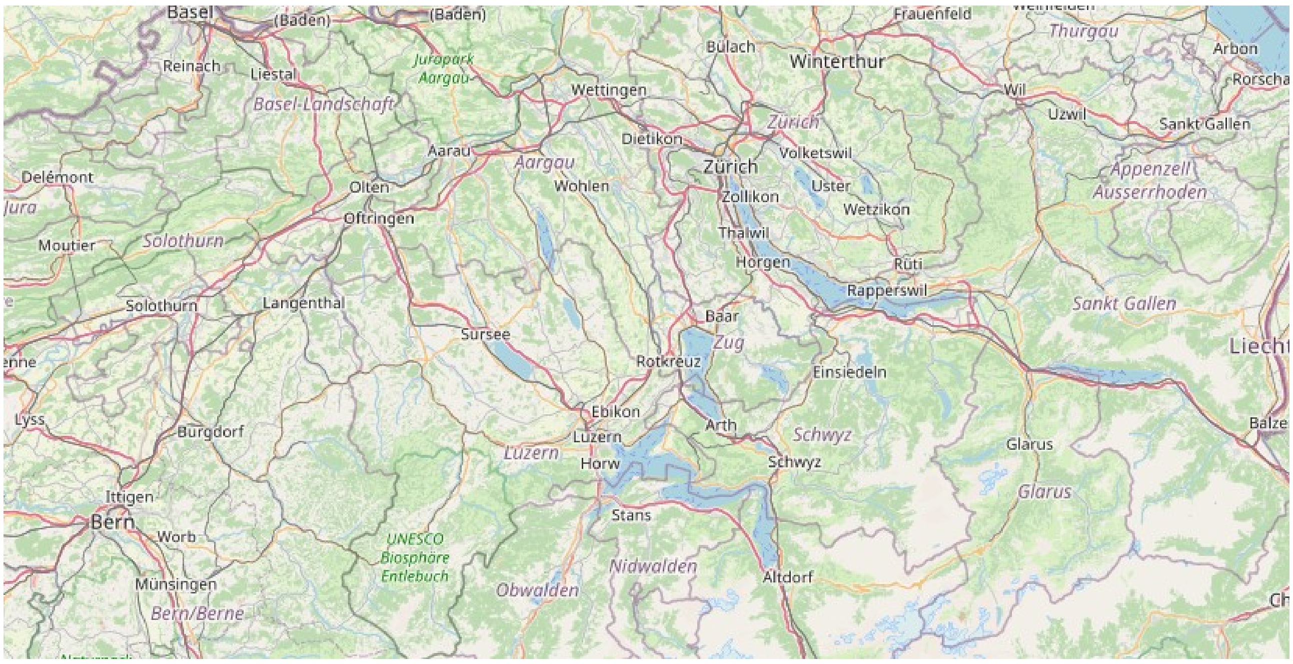
OSM Monitoring Tool

Samuel Lemmenmeier, Tim Wisotzki

Examiner: Prof. Stefan F. Keller



Bachelor Thesis - FS 2022 OST in collaboration with Schutz & Rettung Zurich created: March 14, 2022

last revision: June 17, 2022

titlepicture: card extract from OSM

Samuel Lemmenmeier, Tim Wisotzki

Bachelor Thesis, Spring Semester 2022 at OST

Abstract

The search & rescue organization Schutz & Rettung Zurich are taking different resources into account, when preparing an operation. One of these resources is OpenStreetMap which is openly licensed. Therefore Schutz & Rettung Zurich has to ensure the correctness of the analysed data. To efficiently track and monitor the data integrity a tool is needed to list and filter changesets in Switzerland. Changesets are grouped modifications with a time-stamp whenever the data of OSM is edited.

Schutz & Rettung Zurich requested a tool to track and monitor changesets with the possibility to filter for specific features, called tags, directly on the underlying data of OSM.

"OSM Monitoring Tool" is a full stack web application and consists of three parts: The first part is the front-end (Javascript, Quasar) which enables the user to interact with the application. The second part is a PostgreSQL database for storing all the required data such as application specific information, as well as the complete history of the changeset data of OSM of Switzerland (tables changesets, users) and its underlying objects (tables nodes, ways, relations). The database gets updated with the newest modifications on a regular basis. The third part of the application is the business layer (Python, Django). The business layer is responsible for handling all requests from the front-end, processing the data and gathering the necessary information from the database. For an easy deployment every part of the application runs in separate Docker container and the entire application can be started with a couple commands.

Schutz & Rettung Zurich has announced that their data curators will be using OpenStreetMap in their daily work in the near future.

Management summary

Initial situation

The search & rescue organization Schutz & Rettung Zurich are taking different resources into account, when preparing an operation. One of these resources is OpenStreetMap which is openly licensed and an alternative to the well-known Google Maps.
For planning a rescue mission SRZ has to rely on the correctness of the underlying data of OSM. To efficiently track and monitor data changes a tool is needed to list and filter changesets in Switzerland. Changesets are grouped modifications with a time-stamp whenever the data of OSM is edited.

There exist many quality management tools in OSM but none of these are able to filter for specific features, called tags, directly on the data of OSM.[1] This is a major requirement and is genuinely needed by SRZ during their daily work of maintaining the integrity of their material. Some tools already in use by SRZ are "Targeted Monitoring Tool" and OSMCha, as well as many others. The main focus of our bachelor thesis is on a rewrite of the TMT without the dependency of OSMCha and adding the required tags filter.

To do a BT at the end of the bachelor studies at the OST is the last step to finish the studies and to earn the bachelor degree in computer science. Besides the mandatory necessity to write a BT it is a huge opportunity to apply and show what we have learned in the past few years. A huge motivation were also to gain some insights in the fields of big data and OSM by contributing to a real world problem. Finally it is nice to know that the final product is requested by a third party and the codebase will be available open-source.

Figure 1: Control room of Schutz & Rettung Zurich, from where all emergency operations are planned, monitored and supported.

[1]While finishing the Bachelor Thesis an article reported OSMCha implemented recently exactly that highly requested feature (Willie Marcel 2022).

Approach

First we analysed existing tools, like one which has been created in a previous term thesis at OST as well as other popular tools like OpenStreetMap Changeset Analyzer. In order to deliver a well-rounded product, we drew inspiration from these tools and were able to incorporate some ideas into the final project. After considering all the available information, we started with a new greenfield implementation. The main focus was to create a tool which fulfills the requirements, has an extendable architecture and consists of state-of-the-art technologies.

First we spend some time to get into the subject of big data and OSM. At the start of the BT we needed a lot of time to familiarize with the technologies to build OMT. Internally in the team we worked in a agile way with the help of scrum. This was a huge advantage because some requirements regarding the functionality changed over time and so we were easily able to adopt to it.

Technology

Our project "OSM Monitoring Tool" consists of a full stack web application and is split into three parts: The first part is the front-end (Javascript, Quasar) which enables the user to interact with the application. The second part is a PostgreSQL database for storing all the required data such as application specific information, as well as the complete history of the changeset data of OSM of Switzerland (tables changesets, users) and its underlying objects (tables nodes, ways, relations). The database gets updated with the newest modifications on a regular basis. The third part of the application is the business layer (Python, Django). The business layer is responsible for handling all requests from the front-end, processing the data and gathering the necessary information from the database. The provided API is created with Django Ninja which is a fast Django REST Framework. For an easy deployment every part of the application runs in separate Docker container and the entire application can be started with a couple commands.

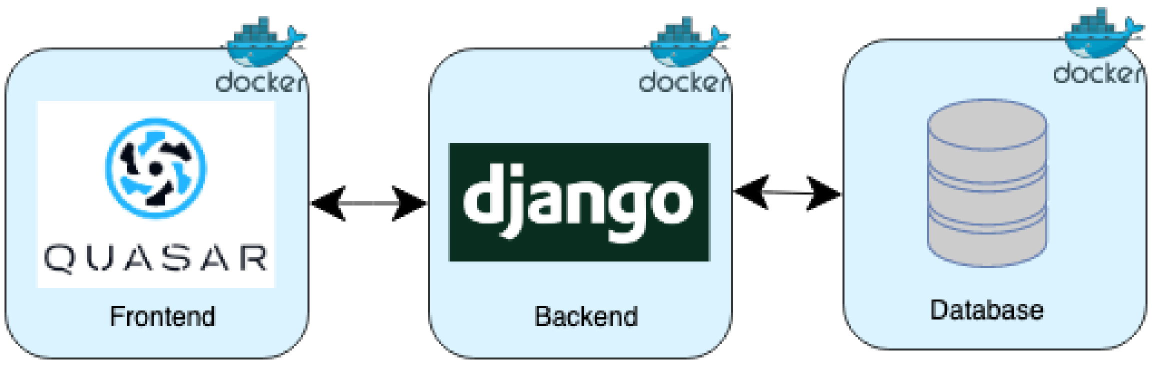
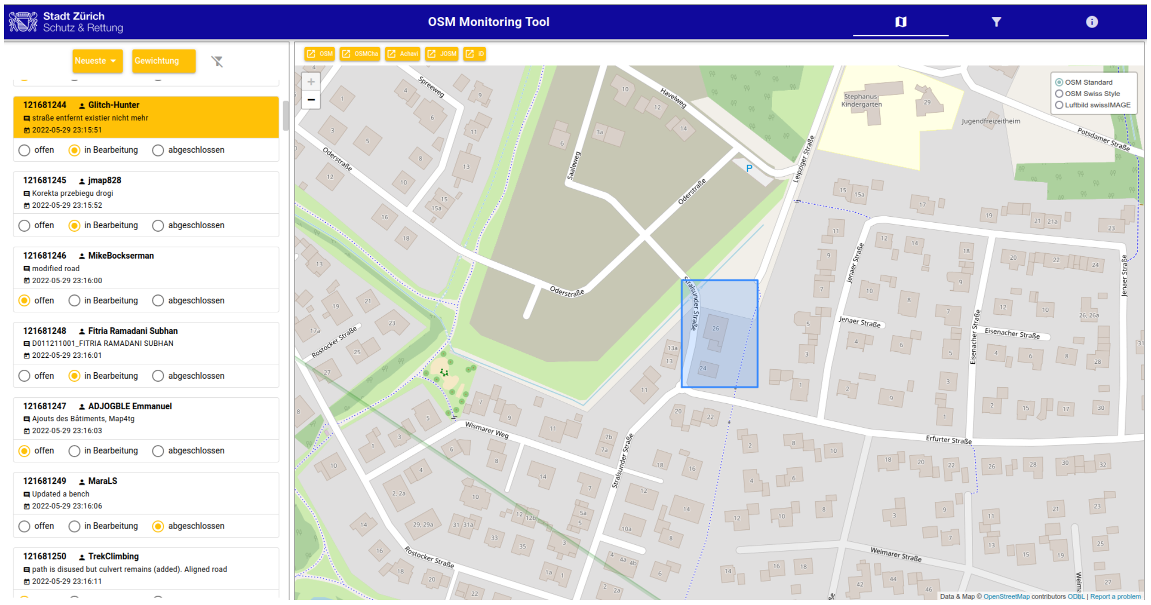


Figure 2: An overview of the application architecture, consisting of a front- and back-end as well as a database, all in Docker.

Result

As a result of our work, we have developed an application called "OSM Monitoring Tool", which allows monitoring changes in OSM in user defined ways. It consists of a web application with a map in the main window and links to some well-known external editors and tools. In a panel to the left a list of changesets is being displayed with their processing status (open, in process, closed) and which can be sorted chronologically or according to priority. A user can create named custom filters. A filter typically consists of a list of tags. Other filter criteria are user name, creation/modification date, processing status, and geographic location, which can be defined by drawing a free-form geometry on a map. One of the unique functions of our OMT is, that it acts on the underlying OSM objects - not only on the changesets. This allows a detailed filtering and increases the usability significantly. It is open source and the web design (color, logo) of the front-end can be easily customized.

The front-end is clearly laid out to simplify as requested from SRZ. One of the main requirements is being independent from OSMCha, this allows to have more control over the map and what will be displayed on it. In general the code structure is build in a modular way to allow further implementations of functionality. The SRZ has announced that their data curators will be using OMT in their daily work in the near future.



121681244 Glitch-Hunter
straße entfernt existier nicht mehr
2022-05-29 23:15:51
offen [✓ in Bearbeitung] abgeschlossen

121681245 jmap828
Korekta przebiegu drogi
2022-05-29 23:15:52
offen [✓ in Bearbeitung] abgeschlossen

121681246 MikeBockserman
modified road
2022-05-29 23:16:00
[✓ offen] in Bearbeitung abgeschlossen

121681248 Fitria Ramadani Subhan
D011211001_FITRIA RAMADANI SUBHAN
2022-05-29 23:16:01
offen [✓ in Bearbeitung] abgeschlossen

121681247 ADJOGBLE Emmanuel
Ajouts des Bâtiments, Map4tg
2022-05-29 23:16:03
[✓ offen] in Bearbeitung abgeschlossen

121681249 MaraLS
Updated a bench
2022-05-29 23:16:06
offen in Bearbeitung [✓ abgeschlossen]

121681250 TrekClimbing
path is disused but culvert remains (added). Aligned road
2022-05-29 23:16:11

Figure 3: An example of our application in use, which shows some of the different components and functions it supports.

Outlook

During the project we had to tackle a lot of hurdles, but all solved minor and major problems led to a bigger learning process. We learned new technologies, working on a bigger project as well as a lot about time management. It was a great experience to do such a project from start to end.

The code is build in a modular way and can therefor be easily extended. Possible improvements in the future could comprise more filter settings or displaying more information about a changeset on the map. Of course, as with every software project, the overall robustness and speed of the code could also be further improved.

Contents

Abstract **2**

Management summary **3**
- Initial situation . . . 3
- Approach . . . 4
- Technology . . . 4
- Result . . . 4
- Outlook . . . 5

Task description (translated to English) **8**
- Rough problem description . . . 8
  - Main "technical" properties: . . . 8
  - Must-haves at start of project: . . . 8
  - Optional: . . . 8
  - Changes regarding the must-haves during the project: . . . 8
  - Further requirements: . . . 8
  - Existing tools for changesets: . . . 8
  - Existing papers: . . . 8

***** Part I technical report** **10**
- Introduction . . . 10
- Task setting and vision . . . 10
- Goals . . . 10
- Approach . . . 10
- State of the art (existing solutions) . . . 10
  - OSMCha . . . 11
- Design . . . 14
  - Architecture . . . 14
  - C4 model system context . . . 15
  - C4 model container . . . 15
  - C4 model component . . . 16
  - C4 model code, classes, object and functions . . . 18
- Implementation . . . 21
  - Approach of implementation . . . 21
  - Main technologies used . . . 22
  - Front-end layer . . . 22
  - Business layer . . . 22
  - Database layer . . . 22
  - Sequence diagram . . . 23
  - UI design . . . 24
  - API provided by back-end . . . 25
  - Testing . . . 31
- Results . . . 31
  - Completeness . . . 32
  - Comparison . . . 32

***** Part II project-documentation** **33**
- Introduction . . . 33

Vision . . . 33
Requirement specification . . . 33
Functional requirements . . . 33
Non-functional requirements . . . 34
General conditions, limitations . . . 34
No external dependencies . . . 34
Time . . . 35
Technologies . . . 35
Knowledge . . . 35
Use cases . . . 36
Use case diagram . . . 36
Use cases detailed description . . . 36
Organisation . . . 37
Stakeholders . . . 37
Roles and responsibilities . . . 37
Work approach . . . 37
Milestones . . . 38
Project phases . . . 38
Risks . . . 39
Results and further development . . . 40
Further development . . . 40
Possible approach . . . 40
Personal reviews . . . 41
Acknowledgements and thanks . . . 41

Software documentation **42**
Installation . . . 42
Some examples . . . 42
Component overview and description . . . 45
Warnings . . . 50

Appendix A **51**
Glossary . . . 51
Links . . . 51

Appendix B **52**
Table of figures . . . 52
Figure sources . . . 52
Table of tables . . . 53
Bibliography . . . 53
Other sources . . . 53
Other documents . . . 53
Final time report . . . 53
Architecture drafts . . . 57
API as JSON . . . 58
Meeting protocols . . . 66

Task description (translated to English)

Goal Rewrite of TMT but independent of OSMCha.

Rough problem description

Functionality of TMT but without any special dependency e.g. OSMCha

Main "technical" properties:

1. Parsing and persisting relevant OSM data like changes and changesets
2. Notify users in case of changes on watched objects
3. Register of observable objects

Must-haves at start of project:

- Define changesets "watcher"
  - restrictable on area of map
  - restrictable on tags, predefined categories for better UX
  - restrictable on single nodes/ways
  - saving user settings
  - sharing (or public/private)
- Login via OSM
- Notifications (mail or alternative)

Optional:

- Selecting predefined regions for observation (on start of application or via environment variable), e.g. Switzerland, canton of Zurich, Germany, etc; restricts the size of the DB and allows using OMT in e.g. Africa.
- "History Slider" for changesets (jump function to display changes on changesets).

Changes regarding the must-haves during the project:

- No user management and no notifications needed
- Optional ranking with QRank
- Range based date filter

Further requirements:

- GUI: Responsive: especially desktop but also mobile tablet (if applicable mouse and touch)

Existing tools for changesets:

- https://github.com/ToeBee/ChangesetMD
- https://github.com/zhm/osmchanges-postgres

Existing papers:

- SA Nadine Sennhauser and Denis Nauli, Herbstsemester 2020/2021, SABA-HS202-Slot-2-Aufgabenstellung and https://eprints.ost.ch/916/ .

- Targeted Monitoring Tool (TMT). Webapp https://srzedi.srz.borsnet.ch/ . Repo https://github.com/Schutz-Rettung-Zurich/srz-edi . This implementation , does only work correct if user-names are "watched". Selecting tags as a filter criterion works not correctly, it does return more changesets than expected.
- "Project of the Month (PotM) CH": Example https://potm.osm.ch/superset/dashboard/17/ , Repo "OSM History DB of Switzerland" (osmhistorydb-ch) https://github.com/sosm/osmhistorydb-ch

*** Part I technical report

Introduction

The following text documents the bachelor thesis "OSM Monitoring Tool".

Part one of the documentation contains the technical report, which focuses mainly on the implementation and surrounding technologies. For more general aspects of the project you can refer to Part II.

Task setting and vision

As described in part twos chapter Vision the main task of the project was to build a well working web application that is tailored specifically for the needs of monitoring changes in OSM, without any major dependencies.

Goals

The main goal of the project was build a web application that can be used to monitor the constantly evolving map-data from OSM to verify its correctness. The initial requirements and goals are stated in the task description, which are described again in the following paragraph:

The main goal of the project is to redesign the existing TMT tool without any external dependencies. Throughout the existing functionality should be maintained and where ever possible improved to better suit the workflow at SRZ and increase the usability. Furthermore the application should be supplemented with additional features that can enhance the experience.

Approach

As described in part II the project was structured into different phases, during which we worked in a sprint by sprint basis. All decisions were done as a group, during regular meetings. Additionally an exchange with our advisors confirmed major points during the project.

In the early stages of the project the main focus was on researching all necessary information in order to make well informed decisions about how to approach the development. During that process we also started to form a rough overview about the architecture and its different components. With more research then the individual components were planned in more detail and decisions about technologies and interfaces were formed.

Starting with the development the different components were build as individual prototypes. This was mainly done to gather some experience with the technologies and would not be necessary when there is already some familiarity with them. Continuously the different components were connected and containerized until a basic prototype was achieved.

Finally the prototype was further extended in order to improve its functionality and meet all requirements.

State of the art (existing solutions)

There are already a couple different solutions available. Some have more, some have less features and the target problem is slightly different for all of them.

The closest to our OMT is certainly the previous work of the TMT which is currently deployed and running. Since this project is a direct reimplementation of the TMT, there are by design many similarities. However the TMT has some significant drawbacks which makes it not fully usable for its current purpose:

- TMT is not able to filter for tags on nodes, ways or relations.
- TMT dependents on OSMCha.

The best solution out there is OSMCha, which offers a wide range of possibilities, but still has its limitations. Since it is the closest to our end goal the following chapter will take a closer look at it.

Besides two solutions above there is also the "project of the month" with which does not have the same goal, but shares some similarities regarding the DB.

OSMCha[2]

What is OSMCha

OSMCha stands for OSM Changeset Analyzer and is an existing web tool for validating changesets. It is designed to analyze and review data changes to OSM. Changes in OSM are collected in changesets and after setting some filter criteria a selection of them are displayed to the user.

What is possible with OSMCha

Validating changesets and it's underlying data. The general workflow with OSMCha will be regarding changesets: Filter -> Select -> View -> Validate

General

- login with OSM account
- view changeset list with many order options
- set filters for filtering changeset list and save/share filters
- view results for a changeset on map
- different hyperlinks to view changeset, OSM elements etc on OSM or other webpages
- create/modify account settings, filters, mapping team, trusted user-list, user watch-list etc.

Filter options for changesets

- basic
  - date range
  - keyword in changeset comment
  - location by polygon drawing or predefined rudimentary polygon boundaries, a positive result is generated when the search area intersects with the bounding box of the changeset
  - bounding box location coordinates and size
- flagged
  - flag boolean or reason
- review
  - status from verifying a changeset
  - review date
  - by reviewer
  - tags of a changeset
- user & teams
  - user-name or user-id
  - trusted/watch users, mapping team search options

[2]Wiki OSMCha (2022); Webpage OSMCha (2022)

- changeset details
  - by changeset id
  - any metadata not settable by other filter options but only regarding the changeset data
  - created range
  - of OSM elements created/modifies/deleted
  - of comments
  - source imagery used
  - specific editor based: Null island edits, Edit count based search, Using multiple filters on OSMCha
  - order by can be set for specific filter

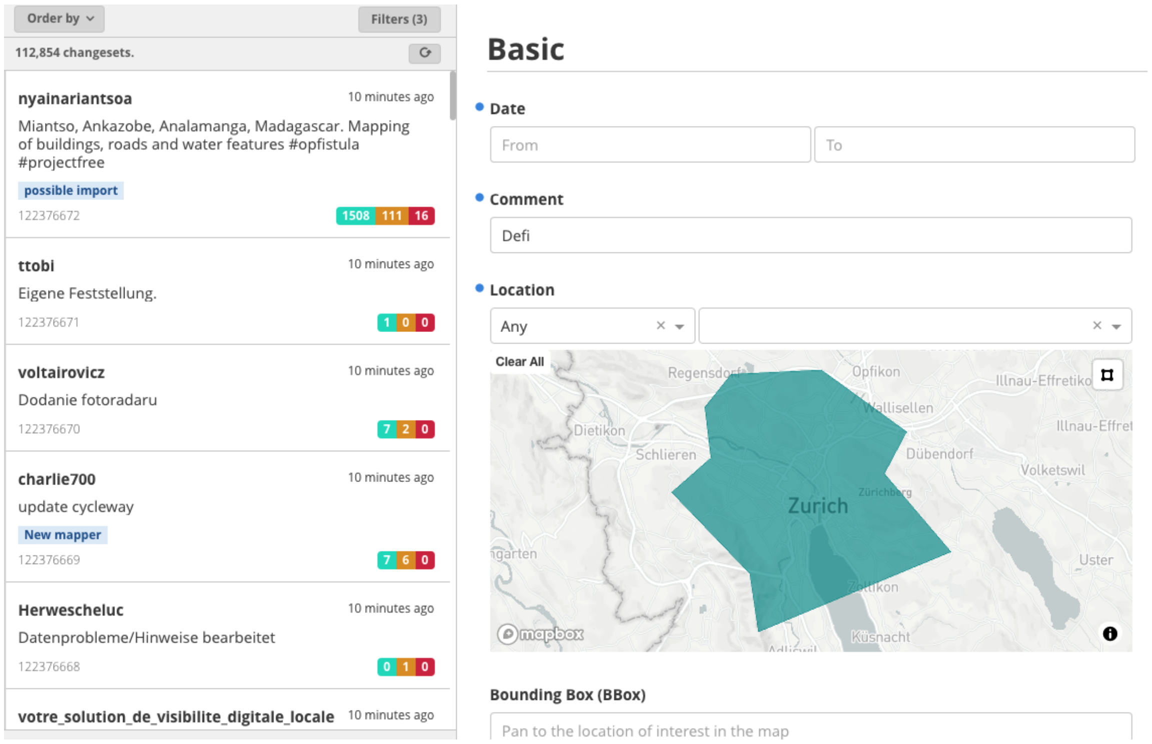
Options for single selected changeset

- view changes of changeset on map
- view changeset details: user, time-tamp, # of created/modified/deleted OSM element, changeset comment, other metadata
- view flagged OSM elements or flag them like: node, way, relation
- view tag changes: all tags altered (in this changeset) regarding OSM elements (key=value) and in which version
- view geometry changes on map
- view other features: created/modified/deleted OSM elements
- start discussion with user for that changeset
- view user details: also tag user as trusted or put on watch list
- control map view for that changeset: on/off added/modified/deleted OSM elements or on/off OSM elements
- control map view: different map styles
- verify changeset: good/bad

Is OSMCha always reliable

Not always does OSMCha return a valid result for the desired filter settings even if it works sometimes. The error message suggests checking filter settings or network connection. In such cases no or the default list of changesets are returned. Such an example is illustrated below.

The search on OSMCha with filter settings: keyword "Defi" and Zurich as the search area should return only changesets containing the search string "Defi" inside it's comment and the bounding box should intersect with the search area. This is obviously not the case because none of the changesets in the list on the image below does contain "Defi" inside it's comment. And a valid result could be expected because e.g. changeset with id: 122352053 would fulfill the search criteria.



Order by Filters (3)

112,854 changesets.

nyainariantsoa 10 minutes ago
Miantso, Ankazobe, Analamanga, Madagascar. Mapping of buildings, roads and water features #opfistula #projectfree
possible import
122376672 1508 111 16

ttobi 10 minutes ago
Eigene Feststellung.
122376671 1 0 0

voltairovicz 10 minutes ago
Dodanie fotoradaru
122376670 7 2 0

charlie700 10 minutes ago
update cycleway
New mapper
122376669 7 6 0

Herwescheluc 10 minutes ago
Datenprobleme/Hinweise bearbeitet
122376668 0 1 0

votre_solution_de_visibilite_digitale_locale 10 minutes ago

Basic

Date: From / To

Comment: Defi

Location: Any

Bounding Box (BBox): Pan to the location of interest in the map

Figure 4: Tested OSMCha with the following filter settings: keyword "Defi" inside the comment and Zurich as the search area. Result list does not contain the requested changesets.

What can OSMCha not do

At the start of this project OSMCha was not able to filter for tags belonging to OSM elements like node, ways or relations. But recently OSMCha implemented this highly requested feature (Willie Marcel 2022).

Design

Architecture

The architecture of OMT follows a monolithic 3-layer-architecture, where each layer is deployed in a separate Docker container. Since each layer runs in a separate Docker container the communication between them is protocol dependent.

The 3 layers are:

- front-end (client layer)
- back-end (business layer)
- database (database layer)

This 3-layer-architecture yields some advantages like:

- each layer has a certain purpose
- each layer could get scaled independently of the others
- each layer could get replaced

This advantages are generated through a loose coupling between the layers and a tight cohesion inside each layer (Hansruedi Tremp 2021).

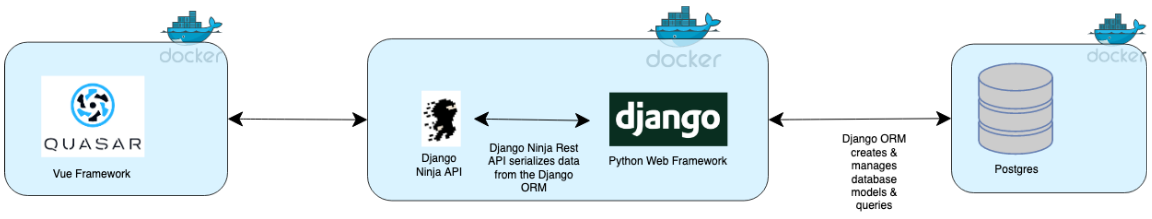

docker
QUASAR
Vue Framework
docker
Django Ninja API
Django Ninja Rest API serializes data from the Django ORM
django
Python Web Framework
Django ORM creates & manages database models & queries
docker
Postgres

Figure 5: This simple architecture overview shows the 3 layers and the communication between them.

C4 model system context

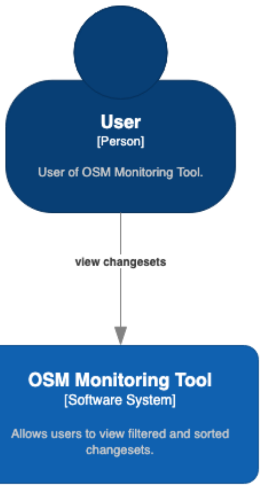

User
[Person]
User of OSM Monitoring Tool.
view changesets
OSM Monitoring Tool
[Software System]
Allows users to view filtered and sorted changesets.

Figure 6: C4 Model Level 1

C4 model container

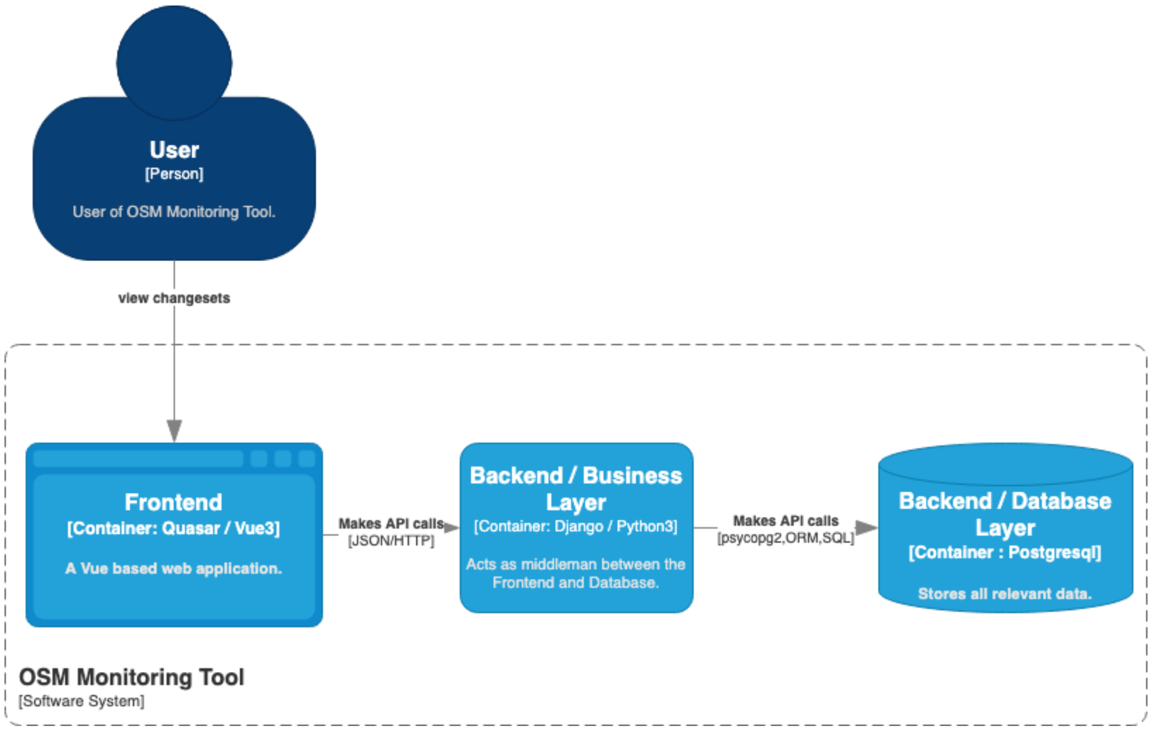


Figure 7: C4 model level 2

C4 model component

Front-end container

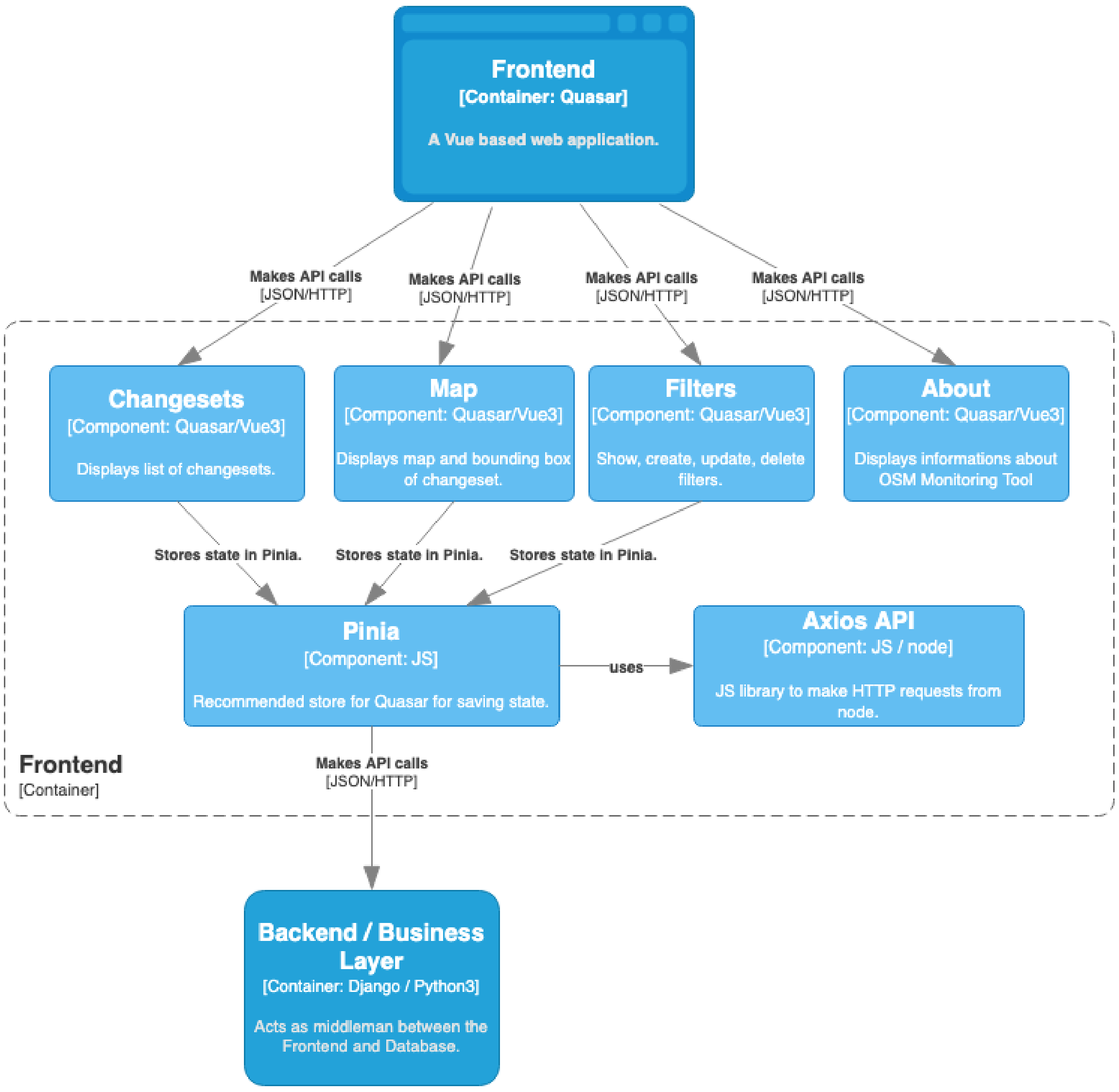


Figure 8: C4 model level 3 front-end

Backend container

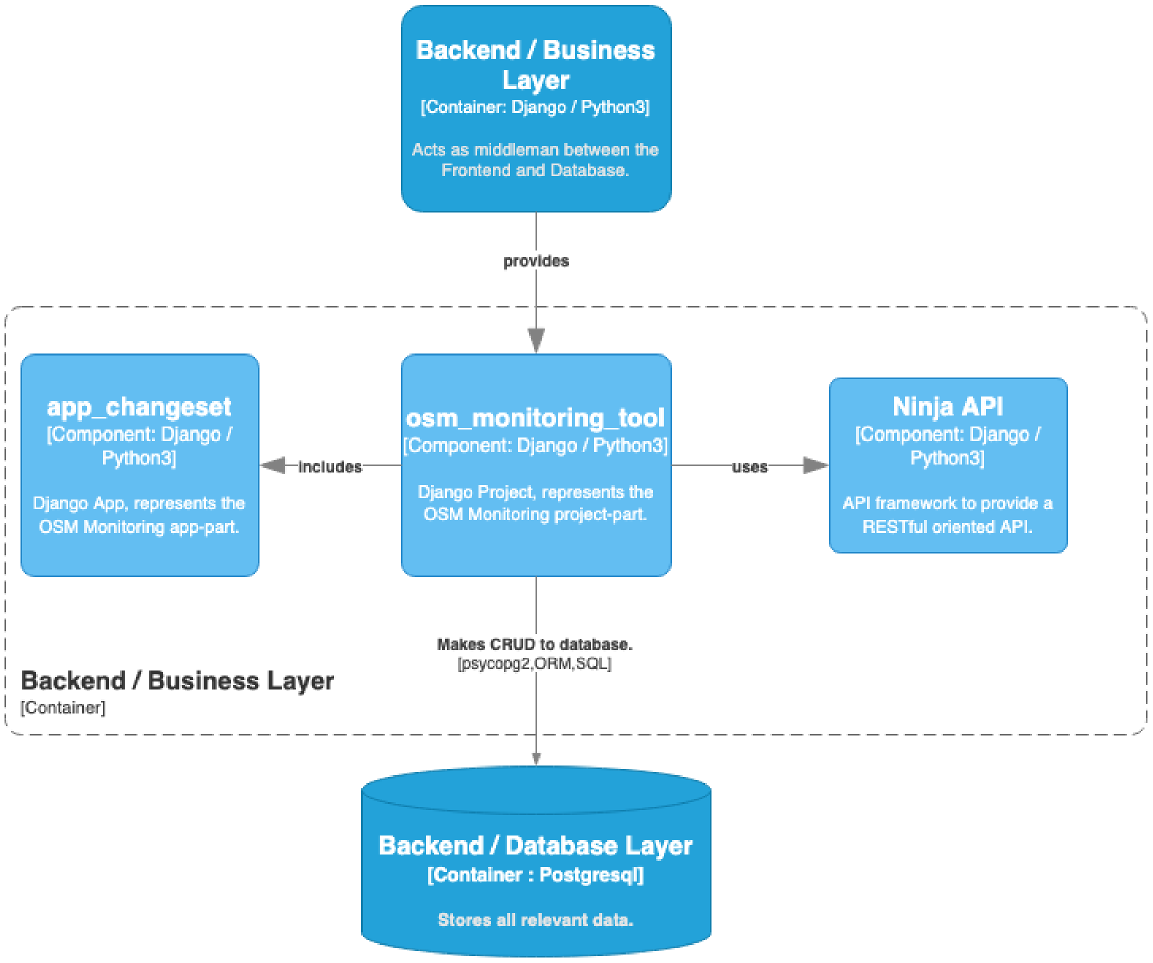

Backend / Business Layer
[Container: Django / Python3]
Acts as middleman between the Frontend and Database.
provides
app_changeset
[Component: Django / Python3]
Django App, represents the OSM Monitoring app-part.
includes
osm_monitoring_tool
[Component: Django / Python3]
Django Project, represents the OSM Monitoring project-part.
uses
Ninja API
[Component: Django / Python3]
API framework to provide a RESTful oriented API.
Makes CRUD to database.
[psycopg2,ORM,SQL]
Backend / Business Layer
[Container]
Backend / Database Layer
[Container : Postgresql]
Stores all relevant data.

Figure 9: C4 model level 3 back-end / business layer

Database container

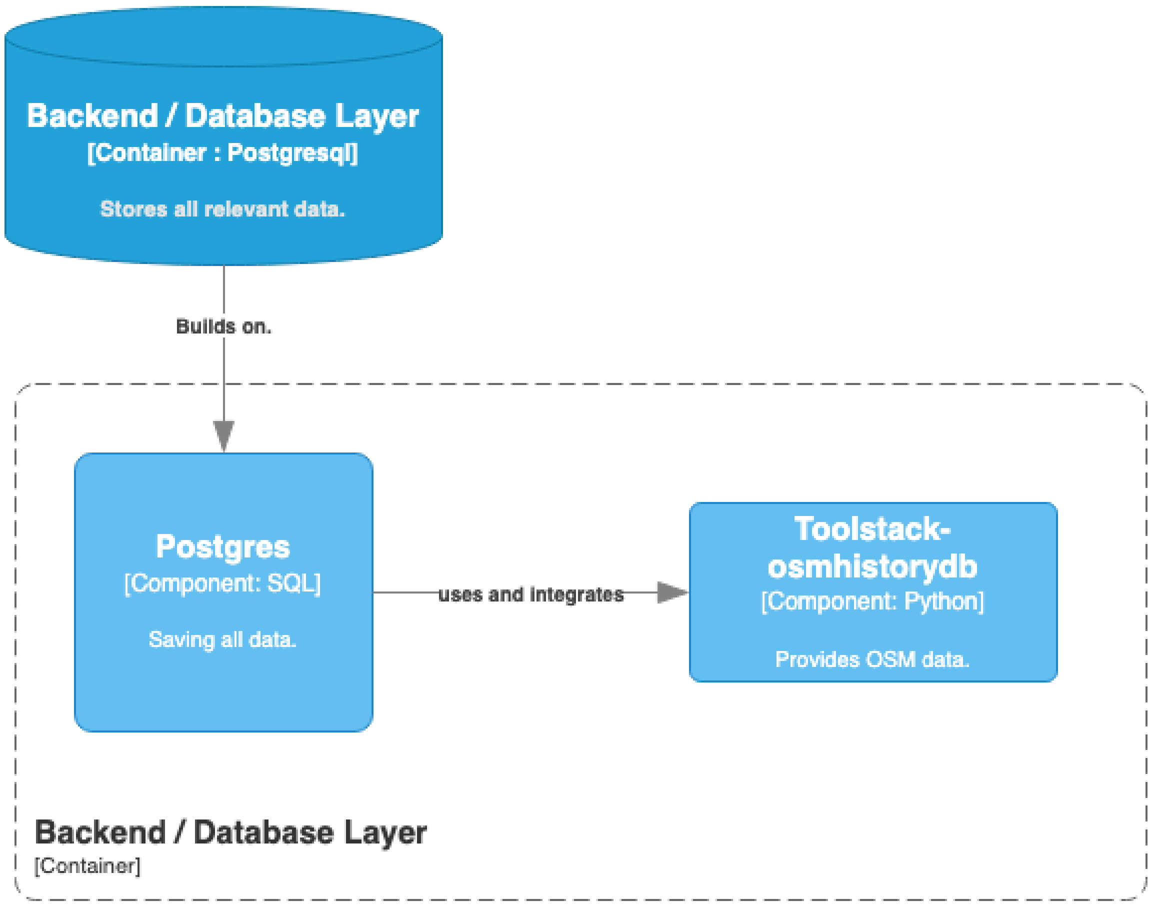


Figure 10: C4 model level 3 back-end / database layer

C4 model code, classes, object and functions

The code and the structure of each layer is described in a separate subchapter. Each subchapter has a different structure because each layer is build with a different technology and needs therefor a different treatment.

Front-end

The front-end layer is build with Quasar which uses Vue3 underneath (Quasar 2022). This layer contains many JS functions and vue components. The most relevant ones are listed below. A Vue3 application consists of components and each component has it's own lifecycle (Vue3 Lifecycle Hooks 2022). This allows building applications in a quite modular manner.

The MainLayout component dictates the overall visual structure of OMT. Inside the MainLayout are different vue components embedded like the ChangesetComponent, MapComponent, FilterComponent and the AboutComponent. In OMT every vue component is a SFC (Vue3 SFC 2022).

Important source files and their functions are described below. Of course each of these components uses in addition the provided lifecycle hooks of Vue3. The components are build with the Composition Api (Vue3 Composition API 2022).

Main layout

Is responsible for the main structure of the webpage, like the header bar and the position of the main content. It also acts like a mount point for the other Quasar pages respectively Quasar components.

Changeset page

This Quasar page contains the Vue ChangesetComponent. This component is responsible for representation of the list of changesets. It also handles user interactions regarding a changeset. Some important functions inside this component are:

- setup: Initial function of the vue lifecycle, responsible for the setup. All the other functions and variables are defined and called inside the setup function.
- updateStatus: Resets the status of an individual changeset.
- deactivateAllFilters: resets all attributes of the applied filter.
- loadData: Does trigger the API call for loading the changesets.
- clickingOnChangeset: saves all relevant data of a changeset inside the changeset Pinia store.
- sort: Does contain the sorting logic for the sort functionality of the changeset list.
- onBeforeUnmount: Helps to clean up and reset some variables.

Map page

This Quasar page contains the Vue MapComponent. The map inside this component is responsible for displaying the bounding box of a changeset. This component is also one of the key parts of the whole application. Some important functions inside this component are:

- setup
- onMounted: Used to mount and create the map when the component is loaded for the first time.
- onBeforeUpdate: Does call createGeojson if the user clicks on a changeset.
- createGeojson: Does create the geojson of the bounding box of a changeset.

Filter page

This Quasar page contains the Vue FilterComponent. This component contains all the logic related with the filers. Some important functions inside this component are:

- setup
- editFilter: Gets called whenever a user clicks on the edit icon of a filter.
- saveFilter: Is responsible for initiating the saving process for a changeset.
- removeFilter: Deletes a filter.
- clickingOnFilter: A filter gets applied.
- checkDate: Checks the validity of a date while the user types or selects a date in the corresponding filter section.
- onMounted: Handles the initial mounting of the map, the layers and the drawing tools.
- onBeforeUnmount
- tagsToString: This is a helper function it helps to pre-process the tags into a certain string format. It does recognize different input formats because pattern matching is in place due to regex.

About page

This Quasar page contains the Vue AboutComponent.

Error not found page

Will be displayed if a default page is not found. This should not happen when everything works like intended.

Map

The maps are created with the Leaflet library. This is an open-source JS library and well suited for OMT (Leaflet 2022). To bring the drawing tools for drawing, editing and deleting a rectangle or polygon on the map the Leaflet Draw API is used (Leaflet Draw 2022).

Pinia

Pinia is used to save the state of components. Whenever data has to be send to the DB a certain procedure takes place. First the data gets stored temporarily inside the responsible Pinia store and from inside the store an API call gets triggered via so called actions. Whenever Pinia is used many stores can be created (Pinia 2022).

Important stores implemented are:

- filters: This store saves all current filters. Whenever a filter changes, gets defined or deleted, an action inside the store triggers the corresponding API call to update the DB.
- filterObject: This store saves the attributes of the applied filter. It does trigger the loading of filtered changesets.
- changeset: All relevant data of a clicked changeset are stored temporarily inside this store.

Business layer

The business layer is a Django project (osm_monitoring_tool) which contains a Django app (app app_changeset).

Django does provide an API for the communication between Django and Quasar. This API is documented here with Swagger. Django has some classes for the models and some for the schemas. The models are in the Django application and the schemas and provided API is in the Django application defined.

The model classes are:

- ChangesetModel: Represents a changeset from the DB. Does contain for each field of in the DB a corresponding attribute.
- FilterModel: Represents a filter from the DB

This classes are used to capture and check the loaded changesets or filters object from the DB.

The schema classes are:

- ChangesetSchema
- FilterAppliedSchema
- FilterInSchema
- FilterOutSchema
- StatusInSchema

This classes are used in the provided Django Ninja API. With this schemas Django does automatically check if the right attributes with the correct types are received or send by Django.

Database layer

It is a PostgreSQL DB.

Table 1: The table below shows all relevant relations stored in the osmhistory DB.

Schema	Name	Type	Owner
public	app_changeset_filtermodel	table	hello_django
public	nodes	table	hello_django
public	osm_changeset	table	hello_django
public	osm_changeset_comment	table	hello_django
public	osm_changeset_state	table	hello_django
public	relations	table	hello_django
public	users	table	hello_django
public	ways	table	hello_django

Implementation

Approach of implementation

After familiarized with the subject of our BT it was time to choose the right technology for the right problem. Definitely not an easy task. But we were kind of lucky because our tutor and the IFS at OST had some preferences.

First we familiarized ourself with Django and PostgreSQL by running them locally and independent. After that we connected these two layers successfully and later on we dockerized them as well. Of course this was not quite as easy and consumed some time.

After that we added Quasar in a separate Docker container to the project. After again having tried it locally and independent first.

The general procedure was to always have a working prototype and extend it little by little until a cut through all layers was accomplished.

At the beginning only some test data was stored in the DB. This provided a good initial setup since it was easy to verify, that the entire data was loaded correctly. After the initial setup a few real changesets were added to the DB. A major achievement was having a running and working DB with all changesets and the complete history of all nodes, ways and relations from Switzerland. This was possible by integrating osmhistorydb-ch into our project. Unfortunately this was not as easy as it looked in the beginning:

The entire toolstack was build on Python 2, which is no longer supported nor recommended. With that many of the dependencies and libraries had outdated versions as well. In order to use the setup we first had to convert it to Python 3, which in itself was not that difficult. However not all of the dependencies were available for Python 3, which made the entire upgrade of the toolstack quite challenging. Additionally it was designed to be run directly on the system, not inside of a container, which posed even more difficulties during the integration. Having a running DB with all the necessary data was quite demanding and time consuming.

Main technologies used

Since each layer runs in a separate Docker container each layer can and is build with a different technology. For that reason it makes sense to inspect each layer in more detail. All containers get instantiated from a so called image which itself is build from a Dockerfile. For that reason the main components in the Dockerfile are mentioned in the corresponding layer below.

Front-end layer

Image for the container is based on a node.js image. Besides that Quasar and NGINX are installed too.

Inside the front-end following settings, libraries and technologies are used:

- Quasar
  - Vue3 with Composition Api
  - Webpack for having PWA support
  - Babel as the JS compiler
  - Leaflet for integrating a map
  - Leaflet Draw for having draw support on the map
  - Axios: Promise based HTTP client for the browser and node.js
  - Pinia: for state management
- Poetry for package dependency management

Business layer

The main component of this container is a Python image. It is combined with many libraries and Poetry for package management. A script checks if the DB is already running and if this is the case it starts the setup for a successful communication.

Inside the business layer following settings, libraries and technologies are used:

- Django
  - ORM of Django: Simplifies access to DB
  - Django Ninja API
    - helps to serialize result of query
  - GEOSGeometry: Simplifies working with geometry objects of the PostGis extension
  - HStoreField: Extension for saving the tags in the DB
  - psycopg2: For DB communication
  - osmviews: For QRanking

This layer defines both the API for the front-end and the SQL query for requesting data from the DB.The query is comprised of different parts to dynamically gather all relevant data. The extend of the query is dictated by the number and types of the attributes of an applied filter. Finally the query joins and filters the DB tables and returns the result to the business layer. Django then transforms the result list into a further processable format and finally sends it to the front-end.

Database layer

The basis of this container is a PostgresQL image with the integrated PostGis and hstore extension. PostGis enables the DB to handle geographic and geometry related objects. An extra script defines the DB properties and is the starting point to load all OSM relevant data into to DB.

The DB setup is inspired by osmhistorydb-ch. One of the main projects depending on this DB setup is Project of the Month as well as many others. To use this setup some additions, updates and major changes to it, regarding dependencies and versions had to be done.

Sequence diagram

This subchapter visualizes the data flow between the layer from a rough perspective. Whenever a request is generated from the user, the front-end will send the request to the business layer. The business layer checks the data structure of the payload and handles it appropriately. After that the request will be redirected to the database layer. The response from the database is checked again for data structure in the business layer before it is returned to the front-end. This procedure is valid for any kind of request the user can generate in the front-end. For that reason two possible requests are visualized in the figures below.

A major use case is saving or editing a filter. The sequence diagram below shows exactly that procedure.

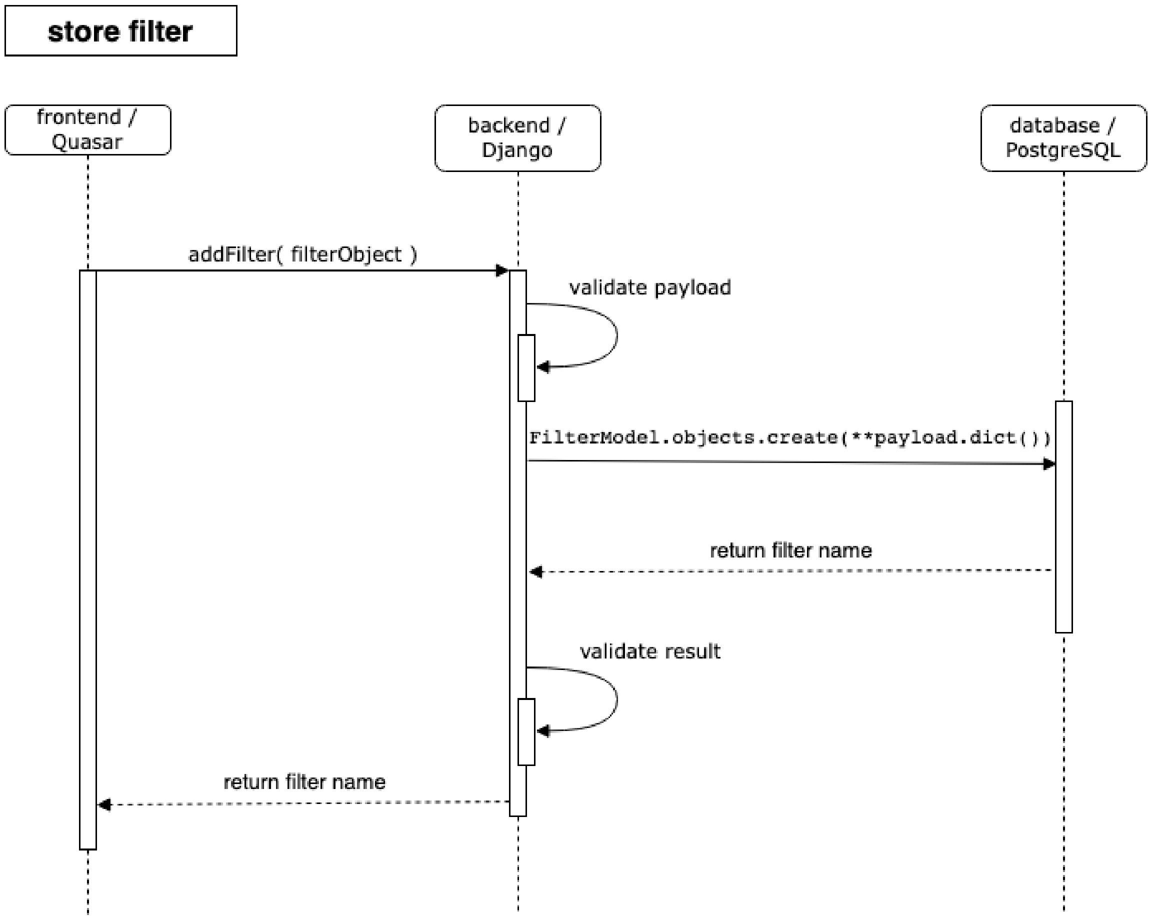


Figure 11: Sequence diagram of saving a filter

Another major use case which results from the one above, is applying a filter. The sequence diagram below shows how filtered changesets are requested.

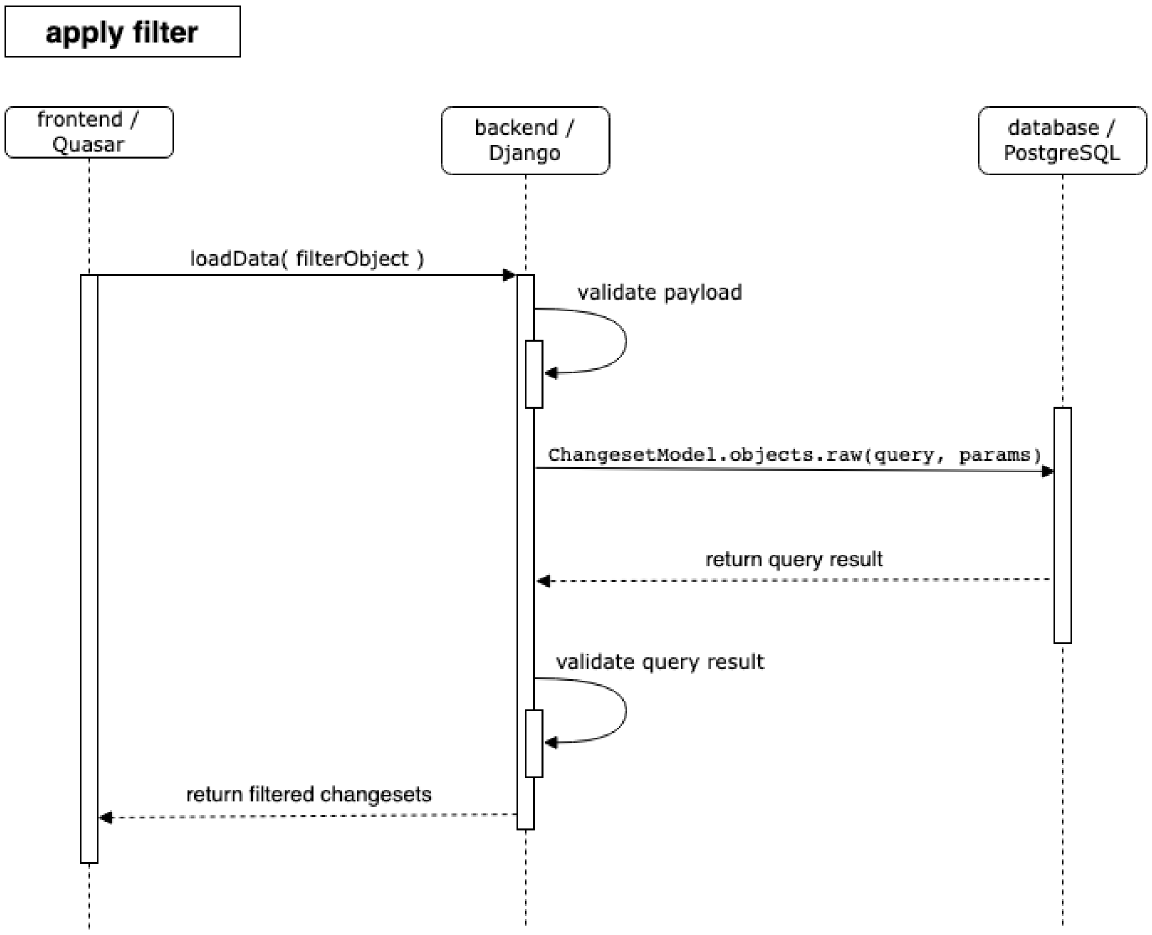


Figure 12: sequence diagram of filtering changesets

UI design

One of the more important aspects for the usability of any service is the interface between user and the program. Our goal was to create a user experience that is as intuitive as possible. Additionally it should be very simple and efficient to perform any tasks, which means that a balance between a clean, uncluttered layout and an accessibility of a few clicks for all features needs to be found.

To achieve the best result in this regard the user-interface is inspired by the existing TMT, as well as OSMCha. This insures that operators that are already familiar with either one don't need to get used to a new layout and immediately can start working. Of course the layout was further refined in order to provide an elegant, but strait-forward design.

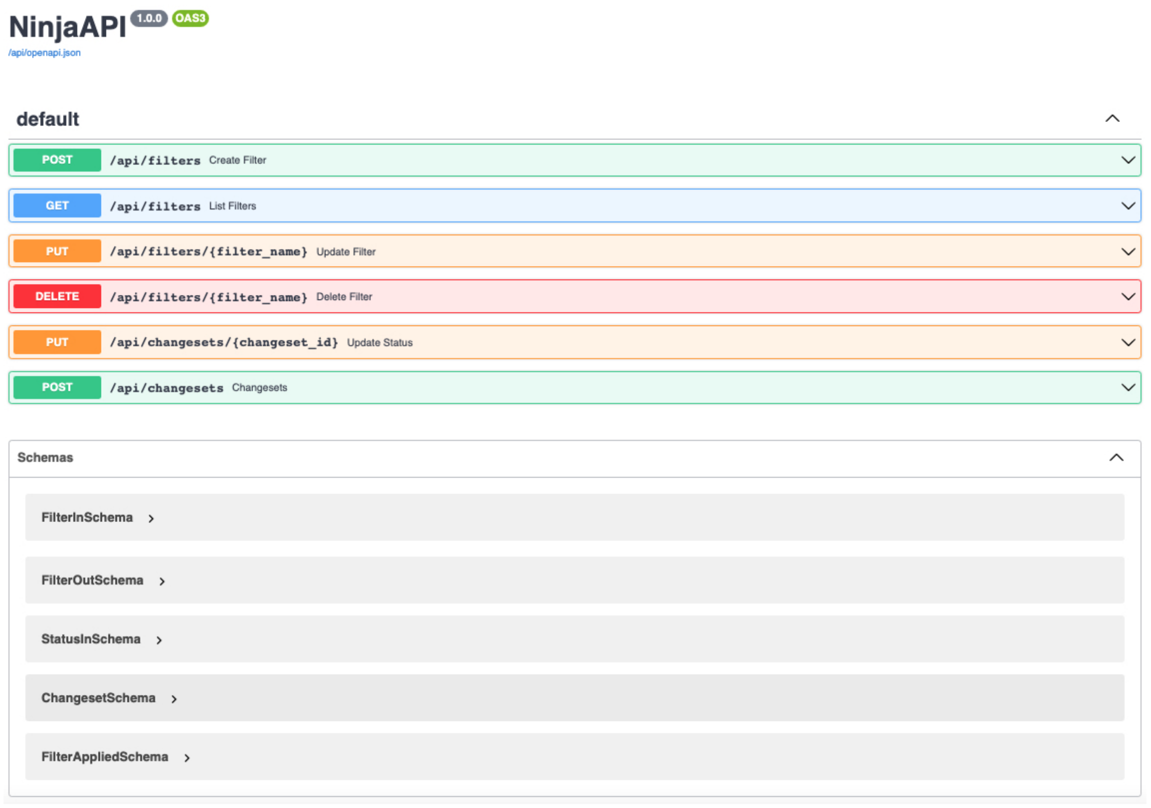
Simplicity is key to reach the goals above. This is reflected thought the interface. The colors are kept modest and are inspired by the SRZ. Through their contrast important parts are easy to spot when highlighted. An example is the selected changeset, which clearly stands out of the list, without being to intrusive. The entire list of changesets is always shown, since it's the key part of the application (for mobile devices with smaller screens it can be hidden), and important for the user to see the current list at a glance, regardless of the current page. With the pages the logical functions are separated, which keeps the interface focused. Still there is a connection over the common changeset area, so the user always knows what is going on.

Because of the simple structure we used icons to mark most fields and buttons. This avoids unnecessary distractions through text and makes it easier for people that might not speak the language. Together the interface should be intuitive for a wide range of users. The layout is also responsive in order to cater to a wide range of screens.

API provided by back-end

The API provided by the back-end is created with the Django Ninja API framework (Django Ninja 2022). API description in JSON is in the appendix.

This are the API endpoints:


NinjaAPI 1.0.0 OAS3
/api/openapi.json

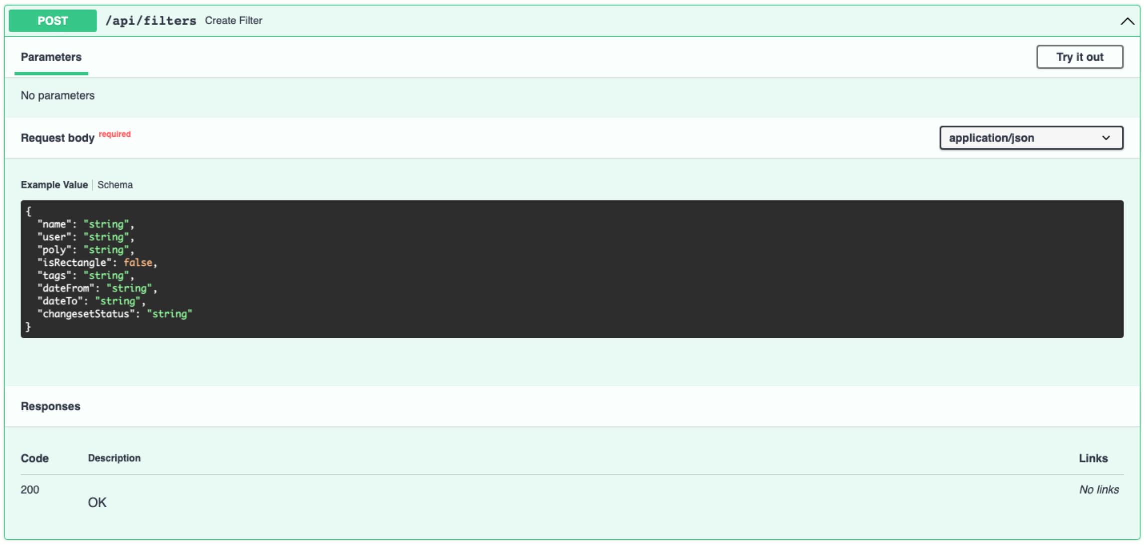
default

POST /api/filters Create Filter
GET /api/filters List Filters
PUT /api/filters/{filter_name} Update Filter
DELETE /api/filters/{filter_name} Delete Filter
PUT /api/changesets/{changeset_id} Update Status
POST /api/changesets Changesets

Schemas

FilterInSchema
FilterOutSchema
StatusInSchema
ChangesetSchema
FilterAppliedSchema

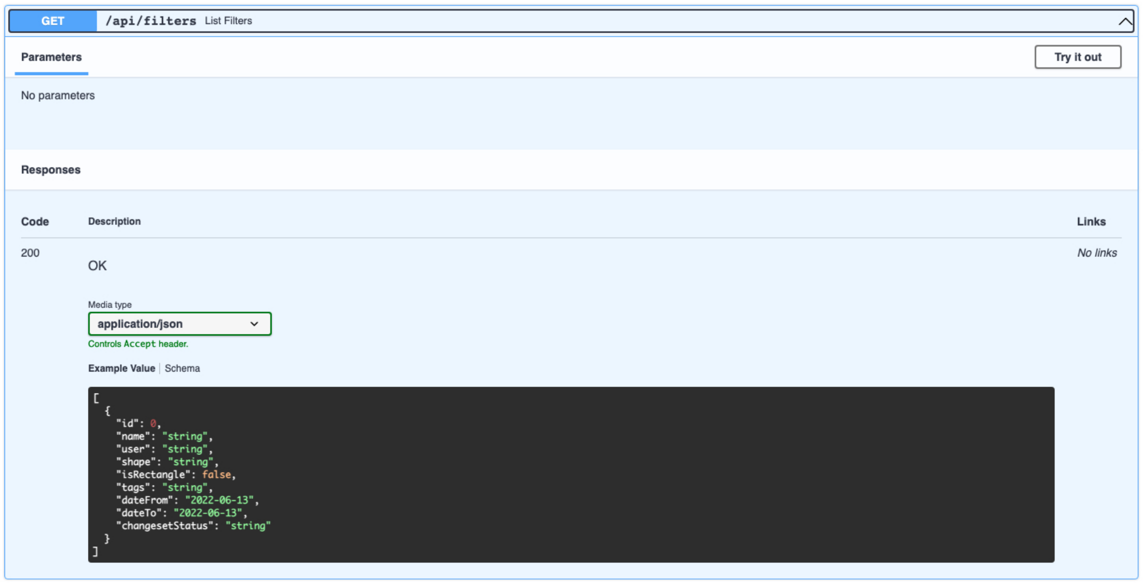
Figure 13: API overview



```
{
  "name": "string",
  "user": "string",
  "poly": "string",
  "isRectangle": false,
  "tags": "string",
  "dateFrom": "string",
  "dateTo": "string",
  "changesetStatus": "string"
}
```

Code	Description	Links
200	OK	No links

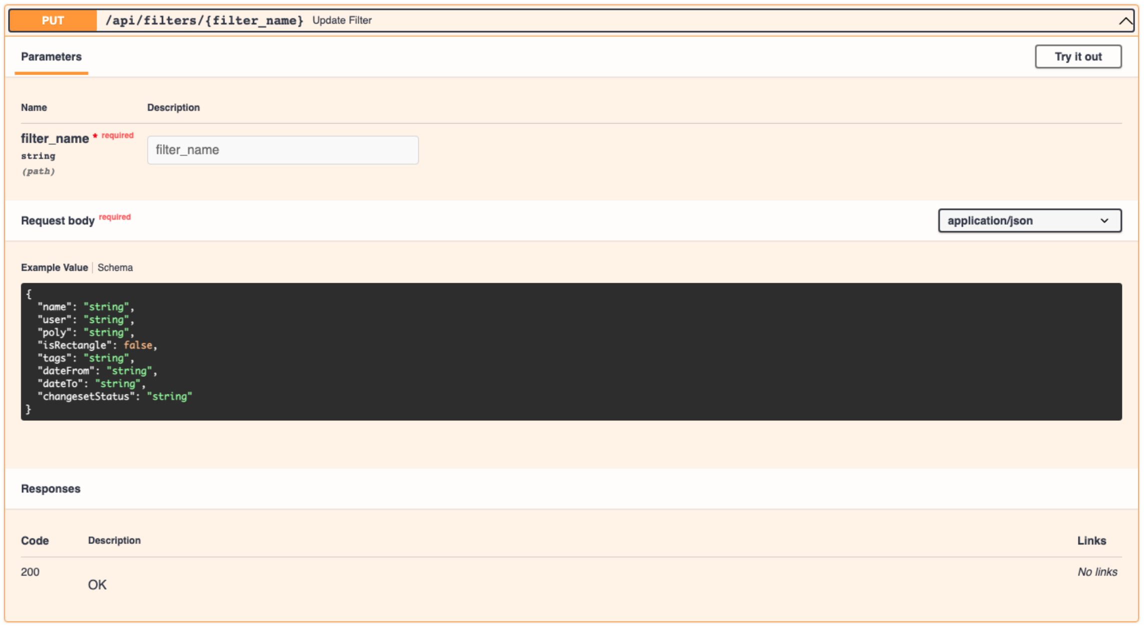
Figure 14: API poast /api/filters



Code	Description	Links
200	OK	No links

```
[
  {
    "id": 0,
    "name": "string",
    "user": "string",
    "shape": "string",
    "isRectangle": false,
    "tags": "string",
    "dateFrom": "2022-06-13",
    "dateTo": "2022-06-13",
    "changesetStatus": "string"
  }
]
```

Figure 15: API get /api/filters


PUT /api/filters/{filter_name} Update Filter

Parameters

Name	Description
filter_name * required string (path)	filter_name

Request body required application/json

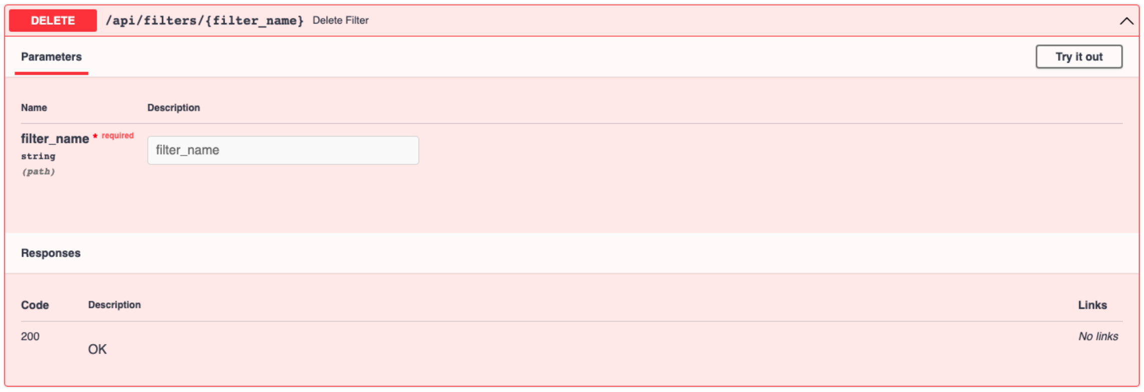
Example Value | Schema

```
{
  "name": "string",
  "user": "string",
  "poly": "string",
  "isRectangle": false,
  "tags": "string",
  "dateFrom": "string",
  "dateTo": "string",
  "changesetStatus": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

Figure 16: API put /api/filters/{filter-name}


DELETE /api/filters/{filter_name} Delete Filter

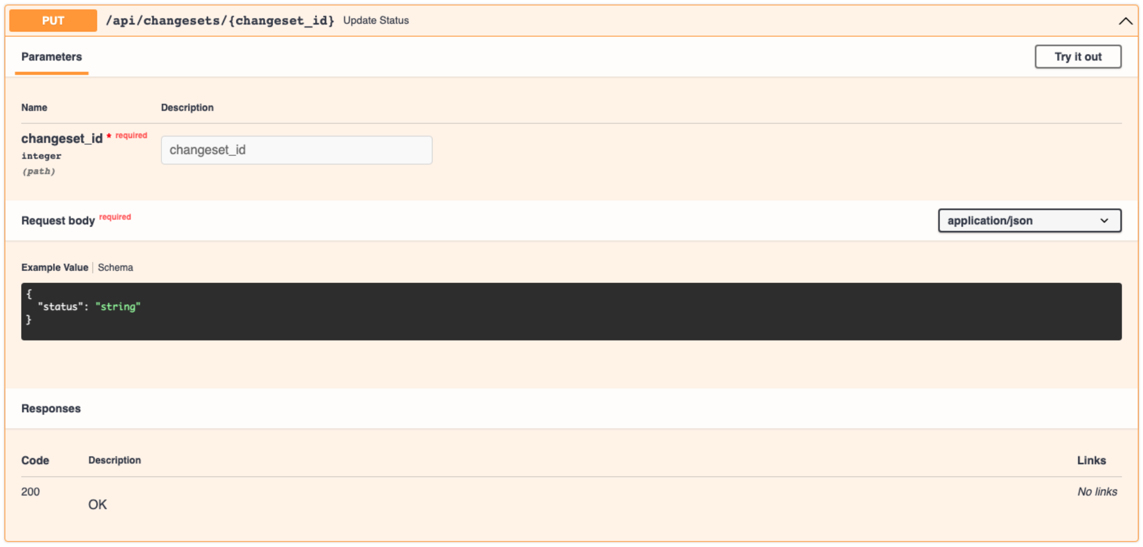
Parameters

Name	Description
filter_name * required string (path)	filter_name

Responses

Code	Description	Links
200	OK	No links

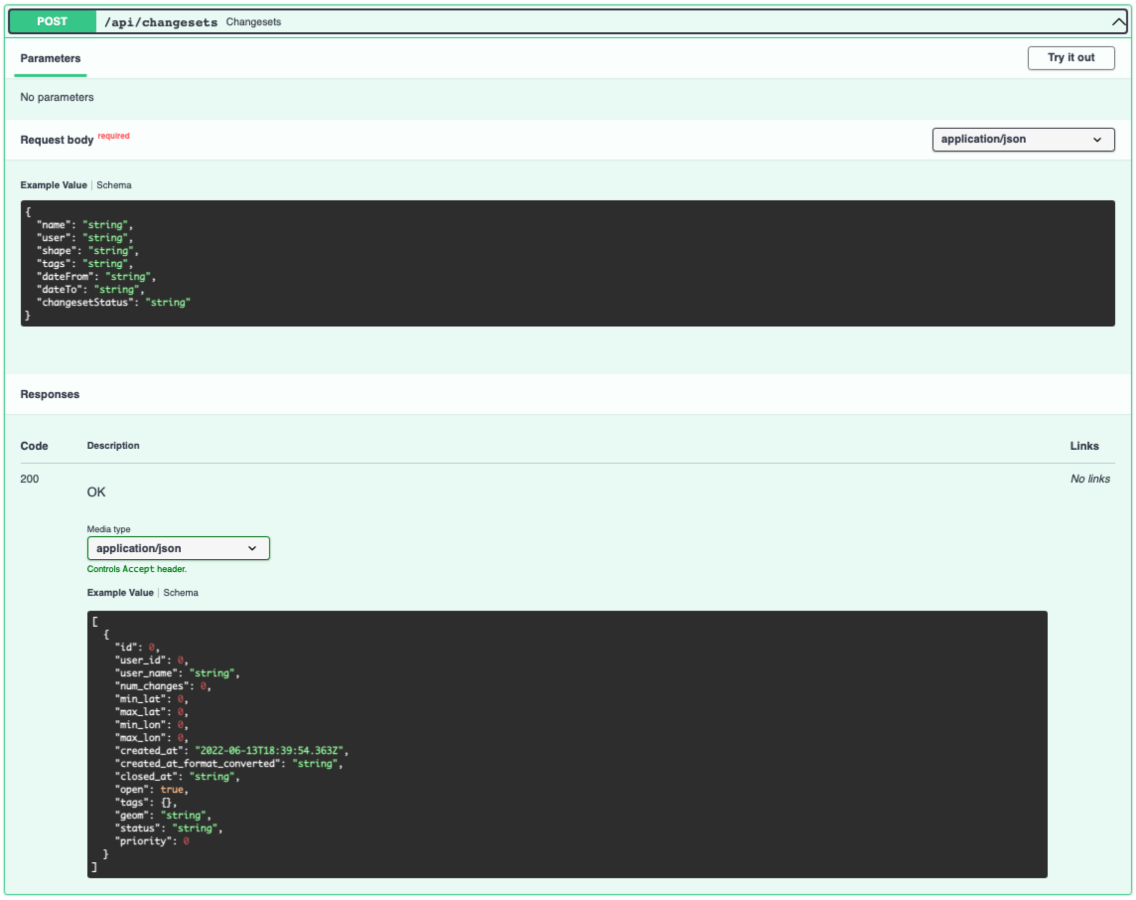
Figure 17: API delete /api/filters/{filter-name}



```
{
  "status": "string"
}
```

Code	Description	Links
200	OK	No links

Figure 18: API put /api/changesets/{changeset-id}



No parameters

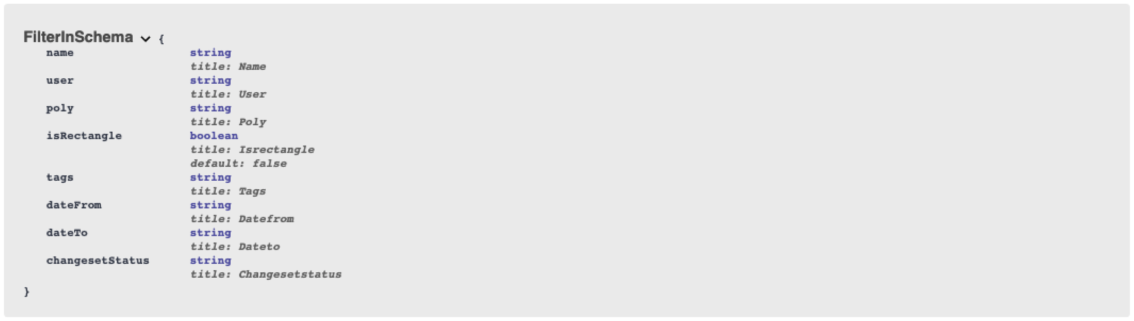
```
{
  "name": "string",
  "user": "string",
  "shape": "string",
  "tags": "string",
  "dateFrom": "string",
  "dateTo": "string",
  "changesetStatus": "string"
}
```

200 OK — No links

```
[
  {
    "id": 0,
    "user_id": 0,
    "user_name": "string",
    "num_changes": 0,
    "min_lat": 0,
    "max_lat": 0,
    "min_lon": 0,
    "max_lon": 0,
    "created_at": "2022-06-13T18:39:54.363Z",
    "created_at_format_converted": "string",
    "closed_at": "string",
    "open": true,
    "tags": {},
    "geom": "string",
    "status": "string",
    "priority": 0
  }
]
```

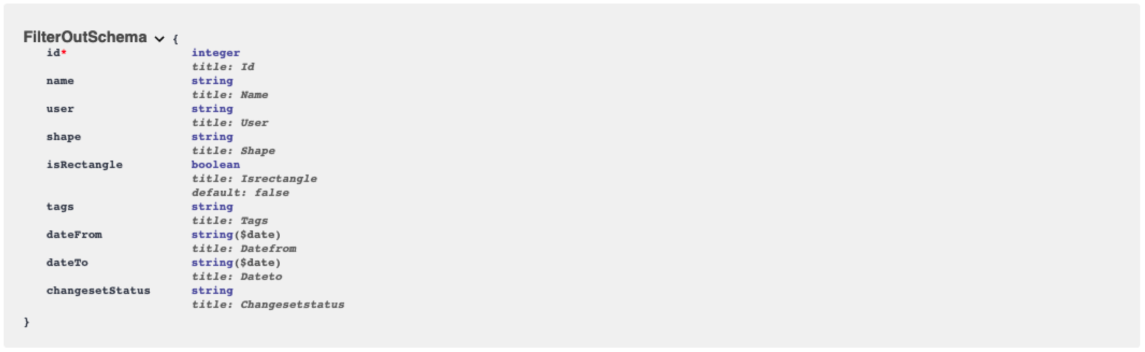
Figure 19: API post /api/changesets

This are the schemas used by the API endpoints to validate the payloads of the requests and response:



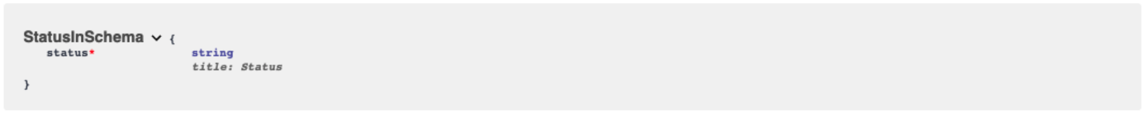
```
FilterInSchema {
    name                string
                        title: Name
    user                string
                        title: User
    poly                string
                        title: Poly
    isRectangle         boolean
                        title: Isrectangle
                        default: false
    tags                string
                        title: Tags
    dateFrom            string
                        title: Datefrom
    dateTo              string
                        title: Dateto
    changesetStatus     string
                        title: Changesetstatus
}
```

Figure 20: API filter in schema



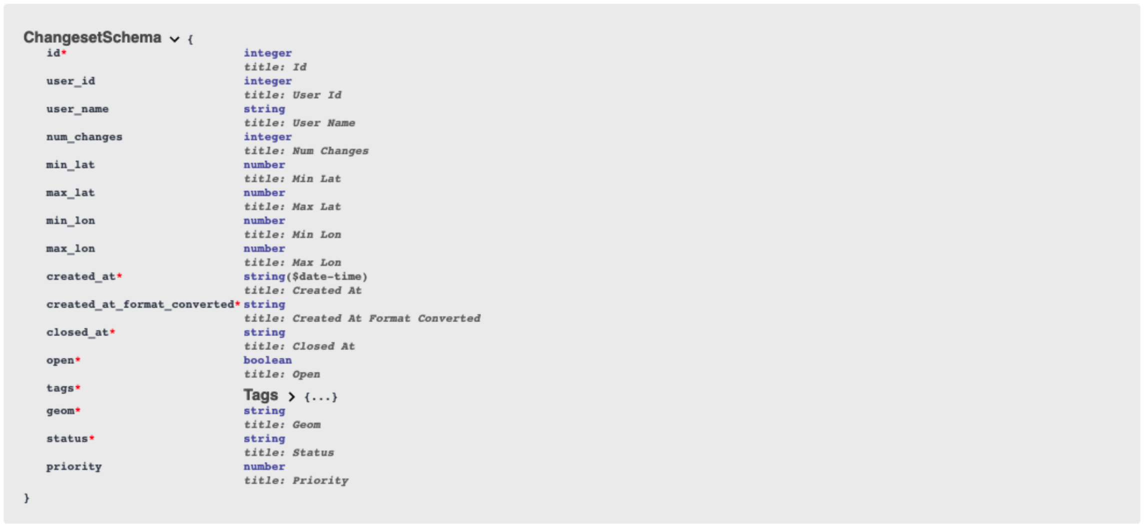
```
FilterOutSchema ⌄ {
   id*                  integer
                        title: Id
   name                 string
                        title: Name
   user                 string
                        title: User
   shape                string
                        title: Shape
   isRectangle          boolean
                        title: Isrectangle
                        default: false
   tags                 string
                        title: Tags
   dateFrom             string($date)
                        title: Datefrom
   dateTo               string($date)
                        title: Dateto
   changesetStatus      string
                        title: Changesetstatus
}
```

Figure 21: API filter out schema



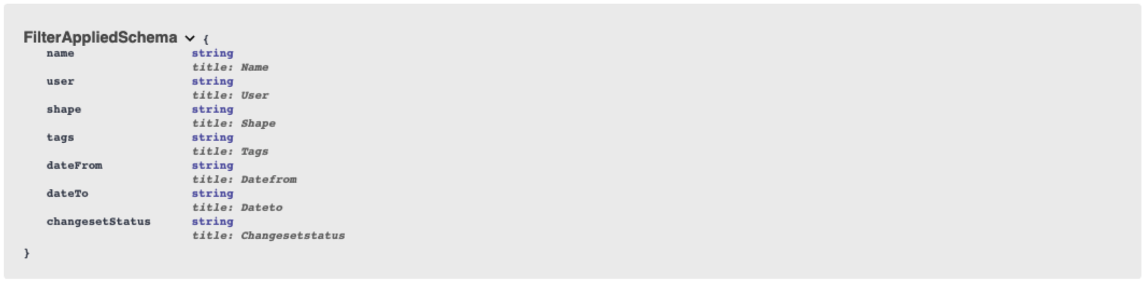
```
StatusInSchema ⌄ {
   status*              string
                        title: Status
}
```

Figure 22: API status in schema



```
ChangesetSchema ⌄ {
   id*                          integer
                                title: Id
   user_id                      integer
                                title: User Id
   user_name                    string
                                title: User Name
   num_changes                  integer
                                title: Num Changes
   min_lat                      number
                                title: Min Lat
   max_lat                      number
                                title: Max Lat
   min_lon                      number
                                title: Min Lon
   max_lon                      number
                                title: Max Lon
   created_at*                  string($date-time)
                                title: Created At
   created_at_format_converted* string
                                title: Created At Format Converted
   closed_at*                   string
                                title: Closed At
   open*                        boolean
                                title: Open
   tags*                        Tags > {...}
   geom*                        string
                                title: Geom
   status*                      string
                                title: Status
   priority                     number
                                title: Priority
}
```

Figure 23: API changeset schema



```
FilterAppliedSchema ⌄ {
   name                string
                       title: Name
   user                string
                       title: User
   shape               string
                       title: Shape
   tags                string
                       title: Tags
   dateFrom            string
                       title: Datefrom
   dateTo              string
                       title: Dateto
   changesetStatus     string
                       title: Changesetstatus
}
```

Figure 24: API filter applied schema

Testing

In every software project, testing is an important part of the development process. But like the variety of different solutions that can be implemented, there is a number of different approaches to testing. From very specific unit tests, all the way to fully-featured acceptance tests, an appropriate mix of test-cases and scenarios needs to be chosen in order to achieve the best balance between verification and costs.

Contrary to many other applications this project does not contain a lot logic components, but the main difficulties are in the processing of the large data amounts. To verify the correct processing multiple tests were done at distinct stages of the development.

Because of the limited business logic and major data flow between the different application layers, we opted to mainly perform manual testing. Especially for bigger integration test the effort to create automated tests would have outweighed their benefit. The majority of tests were however conducted on multiple instances and by different people to insure, that no human error resulted in misleading results.

Whenever new components were introduced extensive integration tests were performed. Additionally the individual parts were analysed according to their respective tasks.

As an additional safety layer we defined in- and output schemas in Django, which always acts as an intermediate level between the database and the front-end. Therefor the flow of data is checked for correct types every time it passes through.

Results

The result of the project is a full-stack, single page web-application replacing the existing TMT tool. A detailed description of the OSM monitoring tool can be found in the software documentation.

For thoughts on further development and useful extensions compare results and further development in part II.

Completeness

Analysing the requirements and comparing them to the finished result the success is judged critically:

- Maintain functionality of TMT tool: Or project
- Remove any special dependencies: In our OMT we do not have any major dependencies. There are some libraries like leaflet, which are used, but the big external tools, mainly OSMCha und mapbox are removed.
- Filter changesets by region: A bounding box or a more specific polygon can be defined and will refine the number of changeset displayed. Since a lot of data is checked this can take a little longer if many changesets need to be verified.
- Filter changesets by tags: Changesets can be filtered by tags by defining a key-value pair. It is also possible to filter by multiple tags at the same time.
- Filter changesets by nodes/ways: The new application is able to filter tags on the included objects of any changeset, which includes nodes, ways and relations. When multiple tags on a high number of changesets is filtered the process might take a moment.
- Filter by date: It works well to filter the changesets according to date.
- Track progress: Every changeset has a status assigned which can be used to monitor if it has not been looked at (open), is "in progress" or already finished.
- Responsive GUI: The user interface of the OMT is able to handle a wide range of display sizes. All contents are adapted responsively. Only when the screen-area get too narrow (for example for smartphones) the application is no longer optimised and will at some point lose its elegance.

From the nice to have requirements a couple were implemented as well:

- Introduce ranking: The Qrank system from IFS was integrated into the application to sort the changesets by their assigned priority.
- Predefined regions for observation: The application is not able to do this distinction.
- History slider: The application is not able to do this, but an external Page like OSM can be used to see history data.
- Add users + notifications: During the project this was determined not to be that great of a benefit and was therefor not implemented.

Comparison

This chapter compares our application to the previous solutions. It highlights the most relevant similarities and differences.

Our project is by design quite similar to the TMT, since that was one of the given requirements. However we were able to provide provide a more targeted selection of functionality and at least in our opinion a more sleek interface. - TMT uses OSMCha for a majority of their features, which was removed as a dependency from our project. Without the external dependencies there is much more freedom of what to display. - Through our own solution the features are much more targeted and specific for the use-cases of SRZ compared to OSMCha in the TMT - Contrary to the TMT in our application it is possible to filter by tags. It is even possible to filter by tags on the internal objects of a changeset, which is a key distinguishing factor. - OSMCha uses mapbox as a dependency, which is also no longer in the application. The map part is done with the leaflet library instead. - The database is its own instance, which allows the operator to independently control their data.

*** Part II project-documentation

Introduction

One of our goals was to work on a project that will have some sort of real world use and therefor benefit society or a group of people. With this project that is certainly the case. Not only does it directly help with a specific problem at SRZ, but will indirectly help many more people by helping to maintain the open-source card material from OSM. Additionally there is the benefit of optimising the work at the emergency dispatch which frees resources for other important tasks.

The project is also quite interesting from a more technical side, enabling us to work with all parts of a modern web application. Developing it as a greenfield project a lot of great changes for creativity, skill and learning are guaranteed.

This part of the documentation focuses on the project itself, not the produced product. It contains the overall approach of the project and all general aspects that are not part of the technical report.

Vision

The main goal of this project was to provide a software solution that supports data curators with their job of monitoring changes on OSM. In order to best assist with their task the application should be simple to use and provide an versatile and flexible set of features. Currently there are no solutions available that ideally allow a user to ascertain all changes while being able to filter them according to a specific set of attributes. Our vision is to enable users to inspect all changes for specific targets in a smooth and transparent way.

Requirement specification

In pursuance of the projects vision the following requirements were determined. They are split into functional and non-functional ones. Furthermore they are separated by their importance into necessary and additional ones.

Since the project was developed with an agile approach the requirements were continuously adapted, while to project progressed. Through discussions with the involved stakeholders and especially the representatives from SRZ the actual usage and needs were further analysed, which resulted in an adaptation of the requirements.

Functional requirements

Functional requirements specify what the product should be able to do. The following points represent the requirements set during the initial task description as well as later additions during the course of the project.

- Maintain functionality of TMT tool: All features and the basic layout and structure of the existing TMT tool should persist after the re-implementation of the solution.
- Remove any special dependencies: The dependence on external sources should be removed, mainly the use of OSMCha. Together with the first point this is the core task of the project and therefore an important requirement for the project.
- Filter changesets by region: This builds a dominant urge for the SRZ and their usage of the tool.
- Filter changesets by tags: Also a key value for the usability the sorting by tags should be possible.

- Filter changesets by nodes/ways: It should be somehow possible to not only filter depending on attributes on the changeset itself, but it's contained objects like nodes and ways.
- Filter by date: Again the selection of a fitting time frame for which to get changesets returned is a request in order to improve the ability of the tool.
- Track progress: It should be possible to keep track of the progress done and what changesets are processed.
- Responsive GUI: The user interface should be responsive to support a wide range of devices.

Nice to haves

The following requirements are not compulsory but it would still be nice if they could be implemented in the application:

- Introduce ranking: Some sort of ranking function in order to sort the changesets by priority.
- Predefined regions for observation: It would be nice if a predefined area (i.e. Switzerland) could be selected at the start of the application which would then only contain changes from the selected region.
- History slider for changesets: This would enable the curators to directly see a history of any changed objects in the application, which would allow them to more easily track any modifications and their validity.
- Add users + notifications: Currently not of major concern, but that would allow a more personalized experience and possibly an automated monitoring that only requires user interaction if certain rules are triggered.

Non-functional requirements

Non-functional requirements are requirements that define how the product should do its tasks. They usually apply to the overall project and are generally applicable.

Performance is one of the most popular non-functional requirements for obvious reasons. If the application is to underpowered it lowers its usability significantly. Unfortunately performance is very hard to quantify, since many factors are involved, starting from the hardware it is deployed on to the amount of data and users involved.

As previously mentioned the usability is an other important requirement. Being rather subjective to a certain degree it is still critical that for the targeted audience the handling of most functions should be possible in an easy and intuitive fashion.

A key feature of long-term successful applications is that they will have be taken care of. To allow this to take place in an officiant and simple matter, it is important that the program has a good maintainability. Additionally it should be possible to extend the application effectively.

General conditions, limitations

From the initial requirements and surroundings of the project the following limitations were given:

No external dependencies

No external dependencies should be present in the project, which can lead to additional design and implementation efforts, but insures that the project can be maintained without relying on third parties.

Time

The project has a strict time limit posed by a deadline. In addition the time budget during the project was limited too, since the number of developers was fixed and the time commitment was limited to a part-time involvement, limited by the regulations of the thesis.

Technologies

Although there were not hard regulations on the utilized technologies it was strongly suggested to work with already present technologies within the department and advisory team. Additionally the customer had to be familiar with the development as well, which limited the number of choices significantly.

Knowledge

Contrary to the others the existing knowledge is a little harder to define and changes the most throughout the project. But the lack of experience with different aspects of the project certainly limits the progress and scope of it. Depending on the amount and complexity of new technologies and skills necessary, the acquisition of the necessary knowledge can take a significant amount of resources.

Use cases

Use case diagram

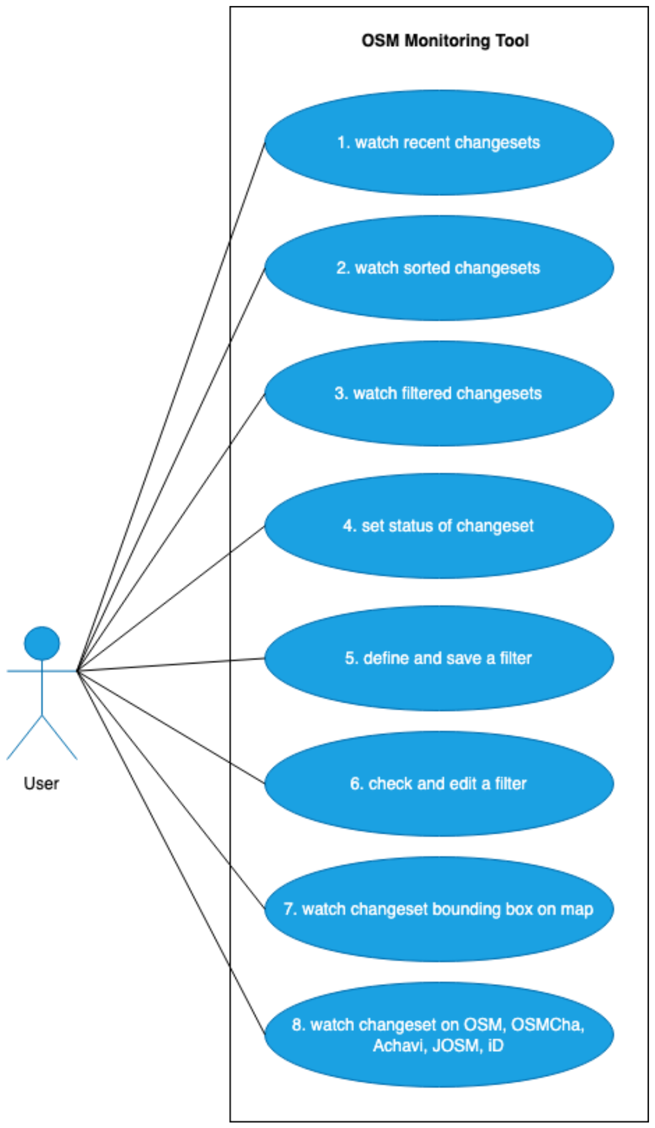

OSM Monitoring Tool
1. watch recent changesets
2. watch sorted changesets
3. watch filtered changesets
4. set status of changeset
5. define and save a filter
6. check and edit a filter
7. watch changeset bounding box on map
8. watch changeset on OSM, OSMCha, Achavi, JOSM, iD
User

Figure 25: Use case diagram

Use cases detailed description

1. Watch list of the most recent changesets with status “open” or “in progress”
2. Sort the list on date, emphasis, status and watch it
3. Filter all changesets on specific filter criteria like OSM user, tags, data, region
   - see if a filter is selected
   - unselect all filters
   - date from or to or in combination possible

    - multiple tags possible
    - 3 different map layers
    - region definable as rectangle or polygon on map
    - shape is editable
4. Set new status on an individual changeset
5. Define, save or delete a filter
6. Check or edit a filter
    - change filter in general e.g.
        - add tags
        - redraw or modify shape
        - reset a filter property
7. Watch boundary box of a changeset on map
    - also with zooming map
    - or changing map layer between OSM Default, OSM Swiss Style, Satellite swissImage
8. Watch changeset data on specific editor by clicking OSM, OSMCha, Achavi, JOSM, iD

Organisation

Stakeholders

The project stakeholders consists of the team members, advisors and external partner.

- Team members: Tim Wisotzki & Samuel Lemmenmeier
- Tutor: Nicola Jordan
- Advisor: Prof. Stefan Keller, Institut für Software OST
- External partner: Christian Nüssli, Schutz & Rettung Zurich

Roles and responsibilities

The team consists of two equal members, therefore it was decided not to assign any fixed roles. All major decisions are made in a group decisions. Individual tasks are defined on a sprint by sprint bases and assigned to best fit the current needs.

Work approach

We choose to work in an agile approach. As we were already familiar with the concept and have successfully used it before we used a combination of the Rational Unified Process as well as Scrum.

According to the RUP we defined the project phases, which are listed in the chapter of the same name. They were set to to get an overall structure and were not strictly followed. The more specific planning for each week was done in the sprint planing. We used a sprint duration of a week, for which the sprint-backlog was set in the opening meeting which also acted as primary base to discuss general project organisations. Since the work only happened on a part time schedule we did not do daily scrums, nevertheless we keep a fairly regular meeting schedule of at least a couple times per week in order to stay up to date and exchange any noteworthy information.

Tools

As described in the first part we used different technologies during the development. Since part of the project goals was the deployment via docker containers we already utilised this to create a mostly operating independent running environment for the project. Regarding development tools we adapted a command-line focused approach which allowed a quite flexible environment for both developers. We both used vim as primary editor and different browsers and their dev-tools.

For version control and code collaboration we used git and the OSTs internal instance of gitlab. Meetings were held mostly via teams and occasionally in person.

Milestones

The project contains of different landmark events. These mark important turns in the advancement of the project.

Initially during the exploration of the technologies the connection of the different application parts pose smaller wayposts. Being able to send data from one to the other one is an important step towards a proper prototype.

Milestone I

The first major milestone is the connection of all application parts with the ability to communicate with each other. This proves the rough concept and builds the base for all following steps.

Milestone II

The next and most important milestone is the complete prototype. At this stage all primary functions are at least partially working. After this break through moment the focus shifts on improving the different aspects and features.

Milestone III

The last milestone is concluded with the final application. By then all developed features will be working properly and the requirements are fulfilled.

Project phases

- Inception: 21.2.2022 - 27.2.2022
- Elaboration: 28.2.2022 - 13.3.2022
- Construction: 14.3.2022 - 28.5.2022
- Transition: 29.5.2022 - 11.6.2022

Weekly overview

We created the following weekly plan at the beginning of the project in order to keep a rough schedule throughout the project. Through our agile approach none of those dates (except the final deadline) are fixed though and the actual weekly plan will be determined during the weekly sprint meetings.

- 1st Week - getting into the project
- 2nd Week - project-plan, define rough architecture components (06.03.)
- 3rd Week - rough architecture
- 4th Week - prototype of architecture + starting a first prototype (20.03.)
- 5th Week - work out details of architecture + first prototype (27.03.)
- 6th Week - implementation
- 7th Week - implementation
- 8th Week - implementation
- 9th Week - implementation MVP
- 10th Week - implementation MVP
- 11th Week - implementation
- 13th Week - final touches + documentation
- 14th Week - buffer
- 15th Week - finish project documentation

Some other relevant dates for the duration of the program are the following:

- spring-holidays no work (11-18.04)
- planned end of coding in order and switch to documentation (end of may)
- deadline to hand in project (17.06.)

Risks

As with every project there are a number of risks. Over the course of the project those risks will be reduced continuously because of more knowledge, proven decisions and finished parts.

Table 2: risks at the beginning of the project

risk description	prob.	damage	risk	risk aversion
mistakes in architecture	0.2	100h	20	reviews
unforeseen knowledge gaps	0.05	20h	1	detailed planing
hidden complexities	0.1	50h	5	plan well
software outages	0.5	10h	5	have alternatives
API change	0.1	30h	3	-
implementation difficulties	0.5	40h	20	initial familiarization with tools
sickness project member	0.1	20h	2	-
misinterpretation of requirements	0.4	50h	20	early and continuous feedback
loss of work	0.01	100h	1	backups

During the project we were able to minimise most of the risks, although some cannot be reduced. For example software problems of external services cannot be influenced by us. One of those instances is gitlab, which had to be used, even though the instance at OST is not the most reliable and we experienced multiple outages. Since we had anticipated those interruptions we were still able to collaborate and share code over a separate repository.

Results and further development

The results of the project can be found in part I results and the software documentation.

Further development

As with every project there is always more work to be done. The following list of things would be options of what to approach next.

Smaller improvements

Some of the smaller improvements that could be done are the following: - Show the last time the changesets were loaded. This would give the user an indication of how current his page is. - Include the tags of deleted osm-elements. Since OSM does not retain them for deleted objects it would be necessary to analyse the history of the individual contained objects in order to regenerate its tags. However it would make the search for tags complete. - Optimise the database queries. Especially if a lot of data has to be processed the requests on the database tend not to be that fast. By optimising the queries a little more some improvement might be possible. - Add production settings to the application. Currently only development settings are available. - Limit the map extract to Switzerland when the Satellite map is chosen. Because the gathered data for the areal view is only available for Switzerland. - Add support for small devices like smartphones, even it the usage on such a small screen probably isn't that efficient and most work will be done on a regular desktop machine.

Major additions

Besides the smaller improvements, there are some additional features that would extend the application nicely: - as mentioned in the comparison to the requirements adding users is not a major factor at the moment, but would certainly open the door for some interesting further possibilities. A login via OAuth with an OSM account would be the ideal - The exact last working state could be retained and returned to the next time the application is opened. - Custom filters could be kept private or shared with other users. - Some sort of notification system for example in case an accordingly modified filter detects a change that fulfills the necessary criteria a notification can be sent to the user or a group of users. - Especially if the software would be utilized in other areas as well, a switch at the start or simply during the installation would be nice, where the area that should be tracked can be determined. This would reduce the size of the database while still providing the option of an international deployment. However some features like the satellite map provider would have to be changed dynamically which might not be trivial in all regions. - Similarly it would be nice to introduce some internationalisation in order to support more languages besides German. - Introduce more information about the changeset directly in the application, like data about the history of the current objects would allow user to make decisions about the changeset faster. - Even more advanced filtering of the changesets for example with a full-text search would further extend the options of limiting the number of entires to the most relevant. This might also include a search function for the already filtered list of changesets. - To make the application faster and reduce the capacity needed it would be nice to reduce the size of the database. This might require more data to be loaded externally and certainly requires some major changes in the entire changeset processing, but would pose an interesting and efficient change to the current state.

Possible approach

The options above are in no particular order. Here we quickly discuss some possible approaches. Of course they might vary depending on the motivation with which the project is extended. It should also be noted that the best approach strongly depends on the future needs of a potential client.

If no particular features are urgently requested, it would make most sense to first improve the current features. This can consists of making them more robust, improving performance or to extend them with new options.

When after this there are still resources available a new feature or functionality can be implemented. The list in above is not prioritised in any way and it has to be determined for the specific occasion which addition is the most useful to develop.

Personal reviews

Samuel

To do such an extended project is a valuable experience. During the software development process I acquired and strengthen some skills which will be applicable in my future occupation as a software developer. I liked it a lot to develop and being part of a process from start to end. A really nice side effect were having the possibility to dive into new areas like OpenStreetMap. Learning and applying new technologies like Quasar and Django as well as using Docker in an extended way is really precious. It was not always easy and sometimes even a bit frustrating but even more satisfying and rewarding at the end having a working product.

Tim

The bachelor thesis is a final opportunity to showcase our acquired skills and knowledge. Besides that I wanted to work on a project, that has some sort of bigger purpose, providing some form of benefit beyond the conclusion of the project. With the project we were able to do all of those goals were fulfilled, which confirms the subject choice.

From the beginning of the project, the main task was quite clear, but since it was developed from scratch there were still many unknowns ahead. This allowed us to work on all major development stages, which was great to gather more experience and knowledge along the way. The fact that we mainly used previously unknown technologies posed some difficulties, forcing us to spend some time getting to know them, but overall was a great way to extend our abilities. Our step-by-step approach and regular meetings proved to be successful and worked well for our small team. It also allowed both to be involved in all aspects of the project without loosing efficiency. An other benefit was the ability to mutually support each other, particularly when things did not work as planned or were frustrating.

The project consistent of a wide range of challenges and experiences: From changing requirements, to outdated dependencies, to entirely new demands, the project always had a surprise ready. However we were able to deal with all of them, producing a working project, that satisfy the goals and learned a lot in the process.

Acknowledgements and thanks

We would like to say thanks to our advisors and other people that supported us along the way:

- Prof. Stefan Keller
- Nicola Jordan
- Christian Nüssli and his team at SRZ
- Lukas Buchli

Software documentation

The following chapters are designed to give some insight to the final application and provide an overview over its components. They also include some guidance and suggestions about the intended usage, as well as present the different options available for the end-user.

Installation

The installation process can be split into three parts. During the preparation it is insured that all necessary requirements are met. Then the actual setup takes place, which is followed by some final touches.

Prerequisites

In order to run the application successfully a system with sufficient power and capacity is required. Further the following programs need to be available:

- the docker container engine
- docker-compose for orchestrating the container management

It should also be insured, that there are no existing docker images or containers that will interfere with the new setup. Similarly none of the necessary ports should be in use already. Finally the system needs a running internet connection during the installation and usage.

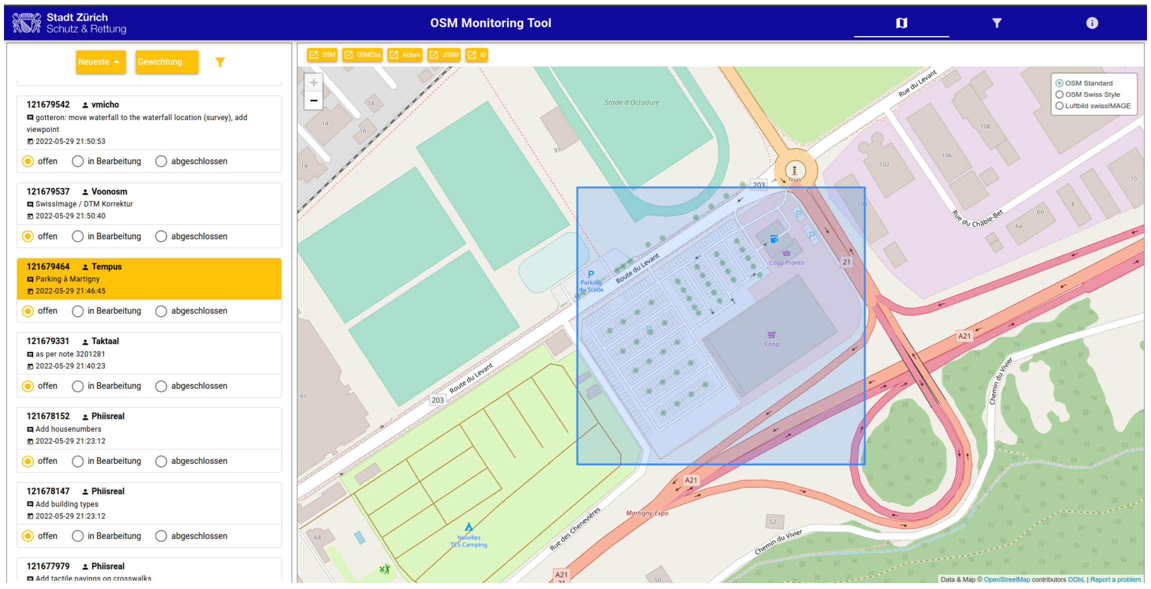
Run installation

1. pull repo
2. download osm-data
3. download osh into directory
4. copy files to database directory in the project as `changesets.osm.bz2` and `switzerland-internal.osh.pbf` respectively.
5. update sequence file (see here)
6. `docker-compose up --build`
7. enter container `docker exec -it prototype_db_1 /bin/bash`
8. execute script `/app/scripts/init.sh`
9. add `*/10 * * * * bash cd /app/osmhistorydb-ch/OSM_Objects/ && ./insert_expanded.sh` to chrontab
10. restart docker-containers

Some examples

Here some use cases are shown, for more detailed information refer to component overview and description

Analyse basic changeset



121679542 vmicho
gotteron: move waterfall to the waterfall location (survey), add viewpoint
2022-05-29 21:50:53
[✓ offen] in Bearbeitung abgeschlossen

121679537 Voonosm
SwissImage / DTM Korrektur
2022-05-29 21:50:40
[✓ offen] in Bearbeitung abgeschlossen

121679464 Tempus
Parking à Martigny
2022-05-29 21:46:45
[✓ offen] in Bearbeitung abgeschlossen

121679331 Taktaal
as per note 3201281
2022-05-29 21:40:23
[✓ offen] in Bearbeitung abgeschlossen

121678152 Phiisreal
Add housenumbers
2022-05-29 21:23:12
[✓ offen] in Bearbeitung abgeschlossen

121678147 Phiisreal
Add building types
2022-05-29 21:23:12
[✓ offen] in Bearbeitung abgeschlossen

121677979 Phiisreal
Add tactile pavings on crosswalks

Figure 26: application after a changeset was selected

On the left side of the screen you can see the list of changesets. The currently selected one is highlighted in yellow. On the changeset you can see its id, followed by the user that created it. Below that the changeset comment is displayed, below which is the created date is shown. Just underneath the yellow part you can see and change the current processing status.

Towards the right side of the screen a map is visible. It contains the bounding box of the selected changeset. In order to get more information about the changeset and to see its history you can open it in one of the following external pages. The links are located above the map on the right-hand side. As an example the changeset with id: 122380948 is used. The link for JOSM requires that JOSM is running on the same machine from where the request originates. This are the links:

- OSM
- OSMCHa
- Achavi
- JOSM
- id

Define simple filter

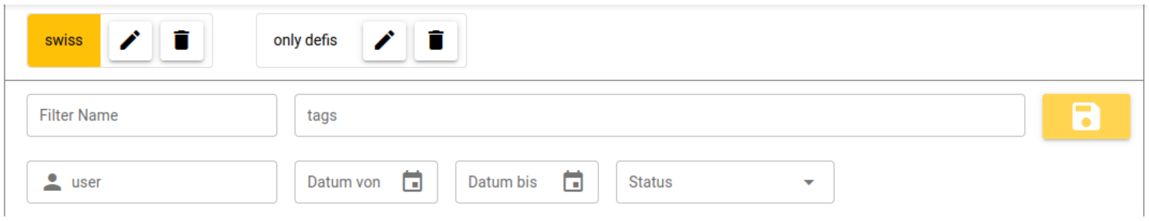


Figure 27: extract from filter page with "swiss" filter selected

Since there are many changesets created on a daily basis, it can be useful to filter the results in order to only see a relevant selection. To access the filter page you need to use the filter button on the top right-hand side of the screen. There you can create a new filter, by entering a filter name and selecting one or multiple of the following attributes:

- user-name (only shows changesets created by the entered user)
- tags (compare define filter tags
- date (only shows changesets in the selected timespan)
- status (only shows changesets with the selected status)
- region (compare define geo-filter)

After saving the filter it can then be applied by clicking on the filter name on the top of the filter page. This will automatically trigger the changesets to be filtered accordingly. To select a different filter you can simply choose the next one out of the list, or create a new filter. To remove any selected filter and get back to the unfiltered list you can press the little filter icon just above the list of changesets, slightly to the right of the sorting buttons. Filters can not only be created and deleted but also be edited.

Define filter tags

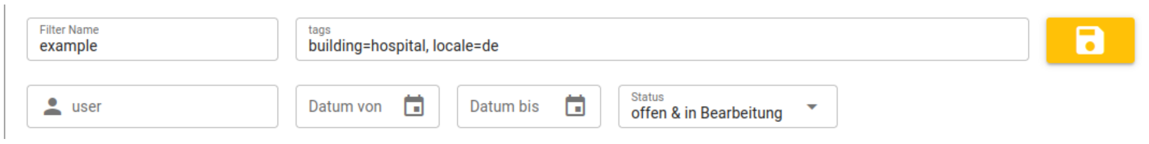

Filter Name: example
tags: building=hospital, locale=de
Status: offen & in Bearbeitung

Figure 28: extract from filter page showing multiple tags set in filter

To define a more specific filter the tag field can be used. You can enter a custom key-value pair against which the tags on the changeset will be compared. Additionally the tags will also be checked on the objects included in the changeset, like on a modified street. The list of changesets is then filtered so only changesets that are tagged accordingly or contain objects that correspond to the tag are displayed. Further more it is possible to enter multiple tags in order to further narrow down the results. This enables a powerful and quite targeted selection of the desired changesets.

Define geo-filter

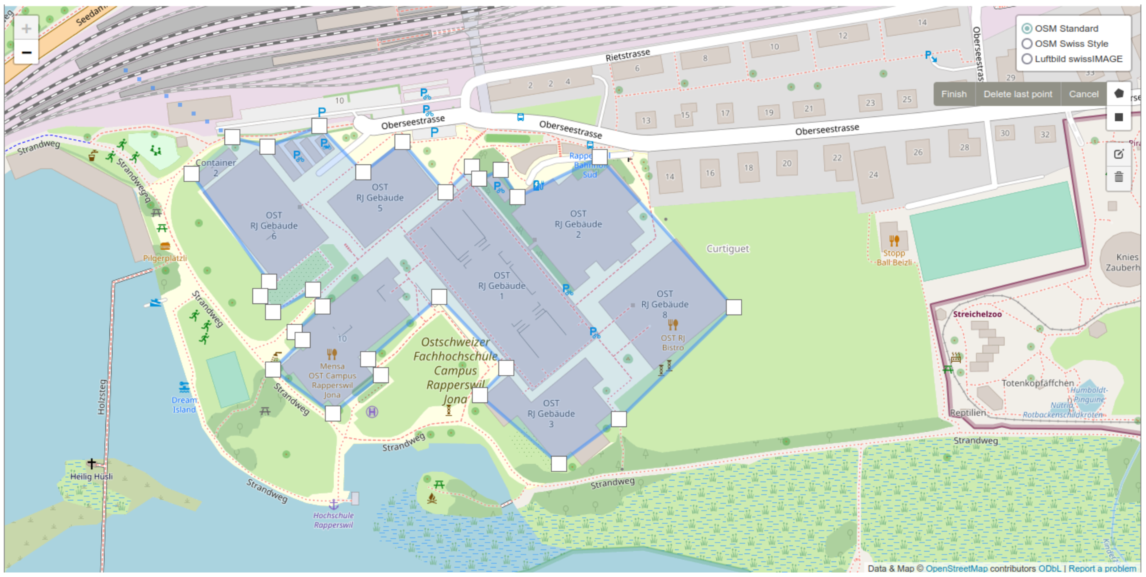


Figure 29: define a geographic filter with a custom polygon

The application also contains the option to define geographic filters. This allows the user to specify an area on the map, with which the bounding box of the changesets needs to intersect.

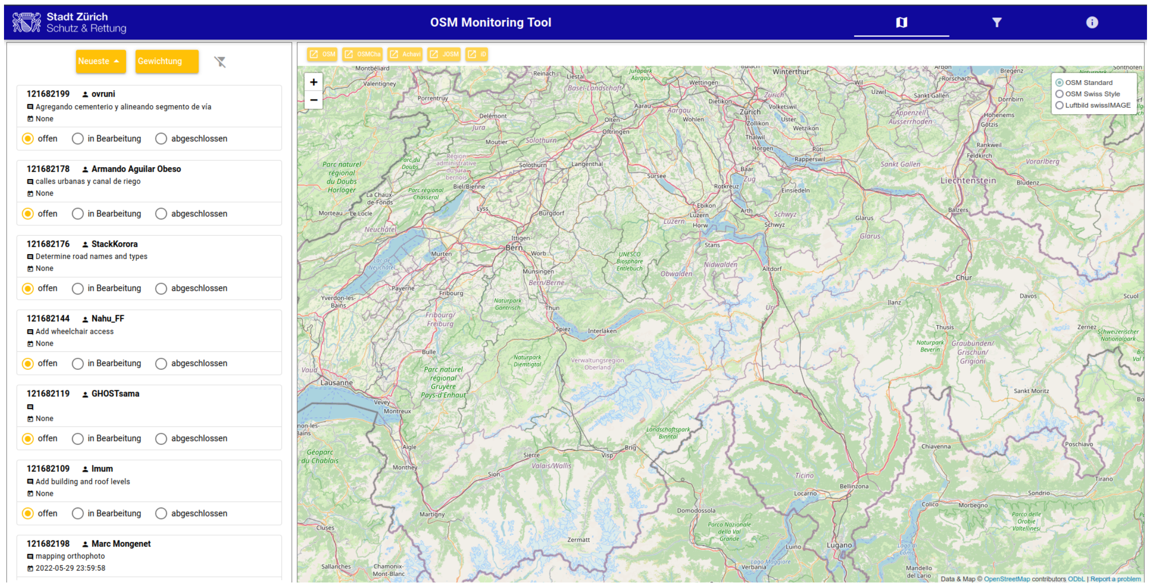
There are two options to select an area. The first is to draw a simple rectangle onto the map, which is great for quickly selecting a rough area to track. In case it is required to track a more oddly shaped region it is possible to draw a polygon which can take any shape and therefor allows a very flexible selection. Both approaches can still be edited later on in case the relevant area changes or needs to be further refined.

Depending on the deployed resources the geographic filters might take a moment to load, since the amount of data verified might be rather large.

Component overview and description

The application can be split into different parts, which are explained in the following chapters.

Overview front-end



121682199 ovruni
Agregando cementerio y alineando segmento de vía
None
[✓ offen] in Bearbeitung abgeschlossen

121682178 Armando Aguilar Obeso
calles urbanas y canal de riego
None
[✓ offen] in Bearbeitung abgeschlossen

121682176 StackKorora
Determine road names and types
None
[✓ offen] in Bearbeitung abgeschlossen

121682144 Nahu_FF
Add wheelchair access
None
[✓ offen] in Bearbeitung abgeschlossen

121682119 GHOSTsama
None
[✓ offen] in Bearbeitung abgeschlossen

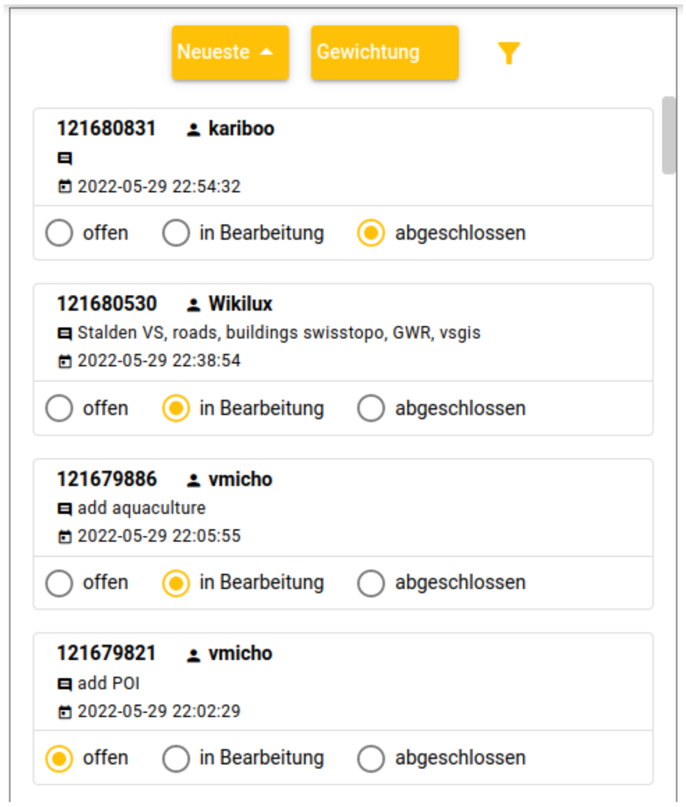
121682109 lmum
Add building and roof levels
None
[✓ offen] in Bearbeitung abgeschlossen

121682198 Marc Mongenet
mapping orthophoto
2022-05-29 23:59:58

Figure 30: overview of the entire application after startup

The front-end is separated into three major areas. Firstly there is the header bar, which also includes the buttons for page navigation on its right-hand side. Below the central header-bar the page is split into the list of changesets on the left and a changeable area on the right. The later contains either the default map, the filter, or an about page. To Navigate between the different sites the buttons in the header can be used. If a changeset on the left is selected the application switches automatically to the map page, regardless of the previously displayed page.

Changesets



121680831 kariboo
2022-05-29 22:54:32
offen | in Bearbeitung | [✓ abgeschlossen]

121680530 Wikilux
Stalden VS, roads, buildings swisstopo, GWR, vsgis
2022-05-29 22:38:54
offen | [✓ in Bearbeitung] | abgeschlossen

121679886 vmicho
add aquaculture
2022-05-29 22:05:55
offen | [✓ in Bearbeitung] | abgeschlossen

121679821 vmicho
add POI
2022-05-29 22:02:29
[✓ offen] | in Bearbeitung | abgeschlossen

Figure 31: overview over the changeset-list

On the left side of the screen there is the list of current changesets. Per default all not yet done changesets are displayed in chronological order. At the top of the list there are three buttons. The first two can be used to sort the displayed changesets according to ascending or descending order based on their date or priority. Next is an icon, that indicates if any filters are currently active. It can also be used to remove any filters to get back to the initial list. For more information about filters refer to the filter page.

The changesets itself show their id, the user that created it, the date and the comment set during its creation. Information about the location can be found on the map page. Below the information of each changeset, there are three buttons to select the current processing status of the changeset. Initially all changesets are set to “open”, which means they have not been processed yet. When a curator starts analysing one of them, the status can be set to “in progress”, so other users don’t start to work on the same ones. When all processing is one the changeset can be set to “done”. The status can be changed again, but the next time the changesets are reloaded (i.e. when a new filter is selected) all finished changesets will be removed from the list.

Header bar

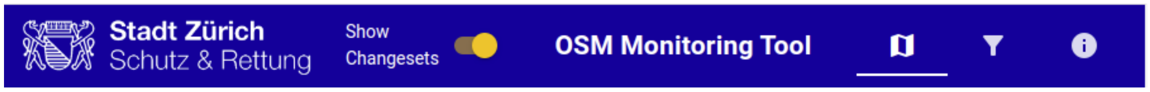


Figure 32: header bar on a mobile device (with toggle switch) with the map page selected

The header builds the top of the application and houses the navigation buttons on the right-hand side. In case a mobile device is used to access the application, it also houses a toggle switch, that allows the user to hide the list of changesets. This setting is intended to provide an option to use the space more efficiently if needed on smaller displays.

Map page (default)

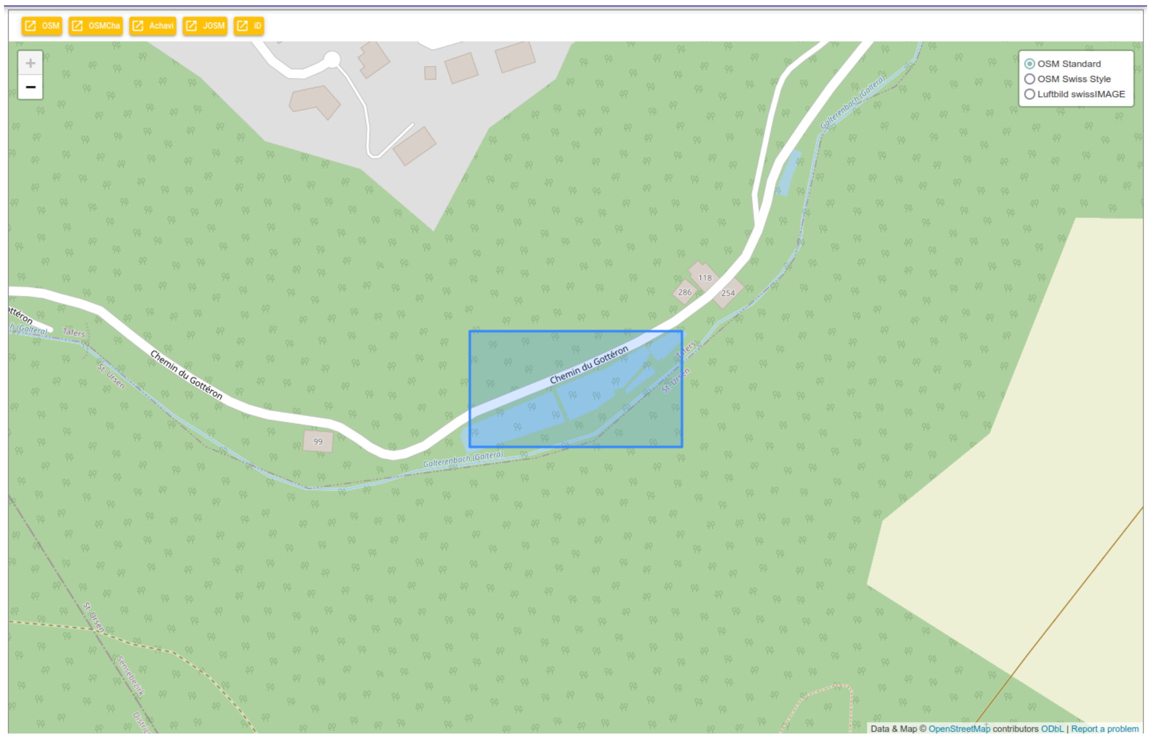
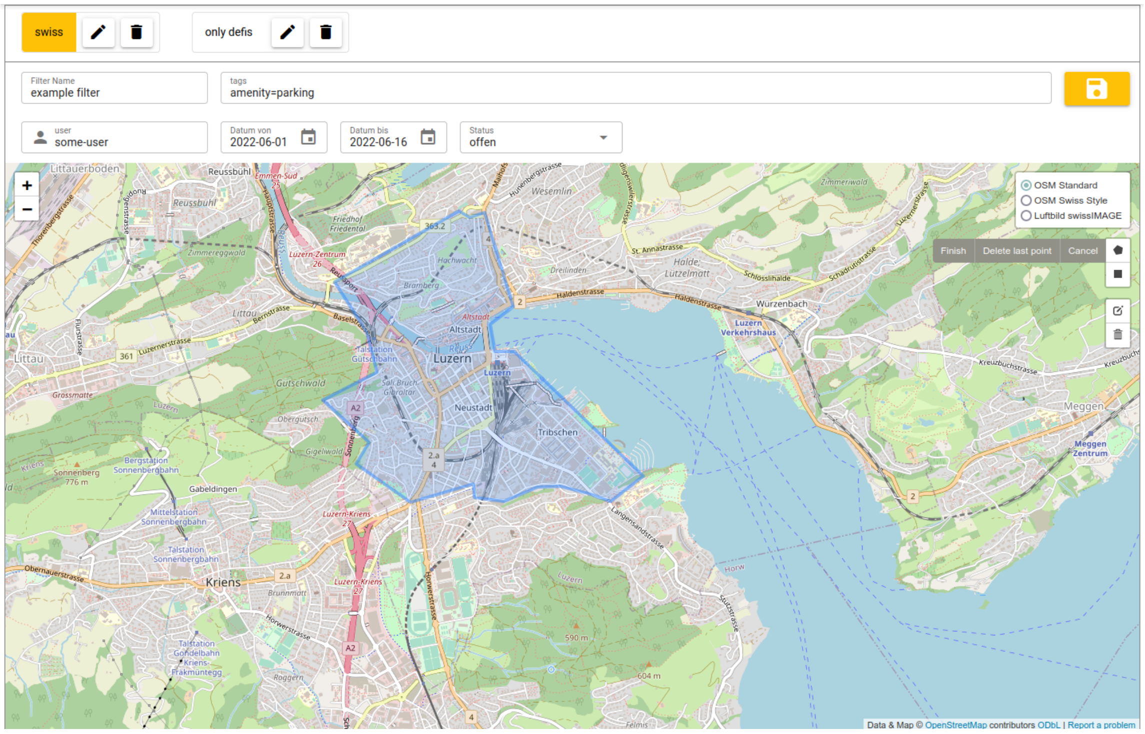


Figure 33: map page, showing the area of an example changeset

The main part of the website is the map page, which as the name predicts prominently features a big map. When any changeset is selected the map page is automatically loaded and the area of the selected changeset is marked on the map. This can range from a single point to a larger area, depending on the objects included in the changeset. Just above the map there are different quick-links that allow the user to directly switch to an external service, where the current changeset and map settings are preserved.

Of course it is possible to move and zoom the provided map in order to get more information if necessary. Additionally it is possible to change the provided card material between the standard OSM map, the Swiss version of OSM as well as a satellite view. For more information about the card sources and information a correlating copyright notice is automatically updated in the right bottom corner.

Filter page



Filter Name: example filter

tags: amenity=parking

user: some-user

Datum von: 2022-06-01

Datum bis: 2022-06-16

Status: offen

Figure 34: filter page, during the creation of a new filter with multiple options

The filter page allows the user to filter the results in order to only see the relevant changesets at any given time. There is a variety of different attributes that can be used and combined in order to get the desired specificity:

User-name filters according to any entered user-name. Tags (compare define filter tags), include key-value pairs that will be tested on the changeset itself as well as all its containing objects, which makes it a very valuable tool for filtering. It is also possible to list multiple tags in order to narrow down the selection even more. A date-range lets you select changesets from a limited time-frame. The Status is per default set to show open and in progress changesets, but can be set to exclusively show one or the other. Finally the geographic filter (compare define geo-filter) allows the selection of specific regions. These can be defined as a simple rectangle or quite advanced polygons to truly select any fitting shape.

Tags are set as a key-value pair. These can be entered in different notations which can be freely chosen. Below are some examples of possible notations:

- `key=value`
- `key:value`
- `key1=value1; key2=value2`
- `key1:value1, key2:value2`

After saving the filter it can then be applied by clicking on the filter name on the top of the filter page. This will automatically trigger the changesets to be filtered accordingly. To select a different filter you can simply choose the next one out of the list, or create a new filter. To remove any selected filter and get back to the unfiltered list you can press the little filter icon just above the list of changesets, slightly to the right of the sorting buttons. Filters can not only be created and deleted but also be edited.

About page

The about page includes some information about the project and involved parties.

Warnings

The application includes a couple warning and error messages. They are displayed in yellow and red respectively at the top and center of the page.

- “Changesets können nicht geladen werden!”
  - The application can not load the chagnesets.
  - Most likely there will be a network connection issue.
  - Or a problem in the database.
- “Dieses Changeset hat keine Koordinaten!”
  - The changeset does not contain any geography information.
  - Probably there was an error during the initial loading of the changeset and there are already no information in the database.
- “Längere Filterzeit möglich!”
  - This message is displayed if a geographic region is defined in the corresponding filter. Such a query has to handle a large amount of data.
- “Status konnte nicht gespeichert werden!”
  - Most likely there will be a network connection issue.
  - Or a problem in the database.

Appendix A

Glossary

Table 3: Glossary

abbreviation	meaning
API	Application Programming Interface
APP	Application
BA / BT	Bachelor thesis
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DB	Database
GIS	Geographic Information System
HTML	Hyper Text Markup Language
HW	Hardware
IFS	Institut for Software
JS	Java Script
OMT	Open Street Map Monitoring Tool
OPE	OSM-PostgreSQL-Experiments
ORM	Object-relational mapping
OSM	Open Street Map
PWA	Progressive Web Application
REST	Representational state transfer
RUP	Rational Unified Process
SA / TT	Term thesis
SFC	Single-File Component
SQL	Structured Query Language
SRZ	Schutz & Rettung Zurich
SW	Software
TMT	Targeted Monitoring Tool
UX	User Experience
VCS	Version Control System

Links

- Task description
- Code repository
- OST Wiki

Appendix B

Table of figures

1 Control room of Schutz & Rettung Zurich, from where all emergency operations are planned, monitored and supported. 3
2 An overview of the application architecture, consisting of a front- and back-end as well as a database, all in Docker. 4
3 An example of our application in use, which shows some of the different components and functions it supports. 5
4 test 13
5 This simple architecture overview shows the 3 layers and the communication between them. 14
6 C4 Model Level 1 15
7 C4 model level 2 15
8 C4 model level 3 front-end 16
9 C4 model level 3 back-end / business layer 17
10 C4 model level 3 back-end / database layer 18
11 Sequence diagram of saving a filter 23
12 sequence diagram of filtering changesets 24
13 API overview 25
14 API poast /api/filters 26
15 API get /api/filters 26
16 API put /api/filters/{filter-name} 27
17 API delete /api/filters/{filter-name} 27
18 API put /api/changesets/{changeset-id} 28
19 API post /api/changesets 29
20 API filter in schema 29
21 API filter out schema 30
22 API status in schema 30
23 API changeset schema 30
24 API filter applied schema 31
25 Use case diagram 36
26 application after a changeset was selected 43
27 extract from filter page with "swiss" filter selected 44
28 extract from filter page showing multiple tags set in filter 44
29 define a geographic filter with a custom polygon 45
30 overview of the entire application after startup 46
31 overview over the changeset-list 47
32 header bar on a mobile device (with toggle switch) with the map page selected . . . 48
33 map page, showing the area of an example changeset 48
34 filter page, during the creation of a new filter with multiple options 49

Figure sources

- titlepicture: openstreetmap.org (last accessed 14 June 2022)
- figure 1: gvz.ch (last accessed 14 June 2022)
- figure 4: osmcha.org (last accessed 14 June 2022)

Table of tables

1 The table below shows all relevant relations stored in the osmhistory DB. 21
2 risks at the beginning of the project 39
3 Glossary 51

Bibliography

Django Ninja. 2022. "Django Ninja - Fast Django REST Framework." https://django-ninja.rest-framework.com/.

Hansruedi Tremp. 2021. *Architekturen Verteilter Softwaresysteme*. Springer.

Leaflet. 2022. "Leaflet Documentation." https://leafletjs.com/reference.html.

Leaflet Draw. 2022. "Leaflet Draw API Reference." https://leaflet.github.io/Leaflet.draw/docs/leaflet-draw-latest.html.

Pinia. 2022. "Pinia Documentation." https://pinia.vuejs.org/introduction.html.

Quasar. 2022. "Quasar Framework." https://quasar.dev/.

Vue3 Composition API. 2022. "Vue3 Composition API Setup." https://vuejs.org/api/composition-api-setup.html.

Vue3 Lifecycle Hooks. 2022. "Vue3 Lifecycle Hooks." https://vuejs.org/guide/essentials/lifecycle.html.

Vue3 SFC. 2022. "Vue3 Single File Component." https://vuejs.org/guide/scaling-up/sfc.html.

Webpage OSMCha. 2022. "OSMCha." https://osmcha.org/.

Wiki OSMCha. 2022. "OSMCha." https://wiki.openstreetmap.org/wiki/OSMCha.

Willie Marcel. 2022. "An Even More Powerful OSMCha." https://developmentseed.org/blog/2022-05-31-more-powerful-osmcha.

Other sources

- OpenStreetMap
- Quasar
- Django

Other documents

Below are different artifacts and reports that were generated during the project or summarize its course:

Final time report

Time statistics

- **total estimate**:
- **total spent**: 96d 50m
- **spent**: 96d 50m
- **samuel.lemmenmeier**: 48d 1h 25m
- **tim.wisotzki**: 47d 7h 25m

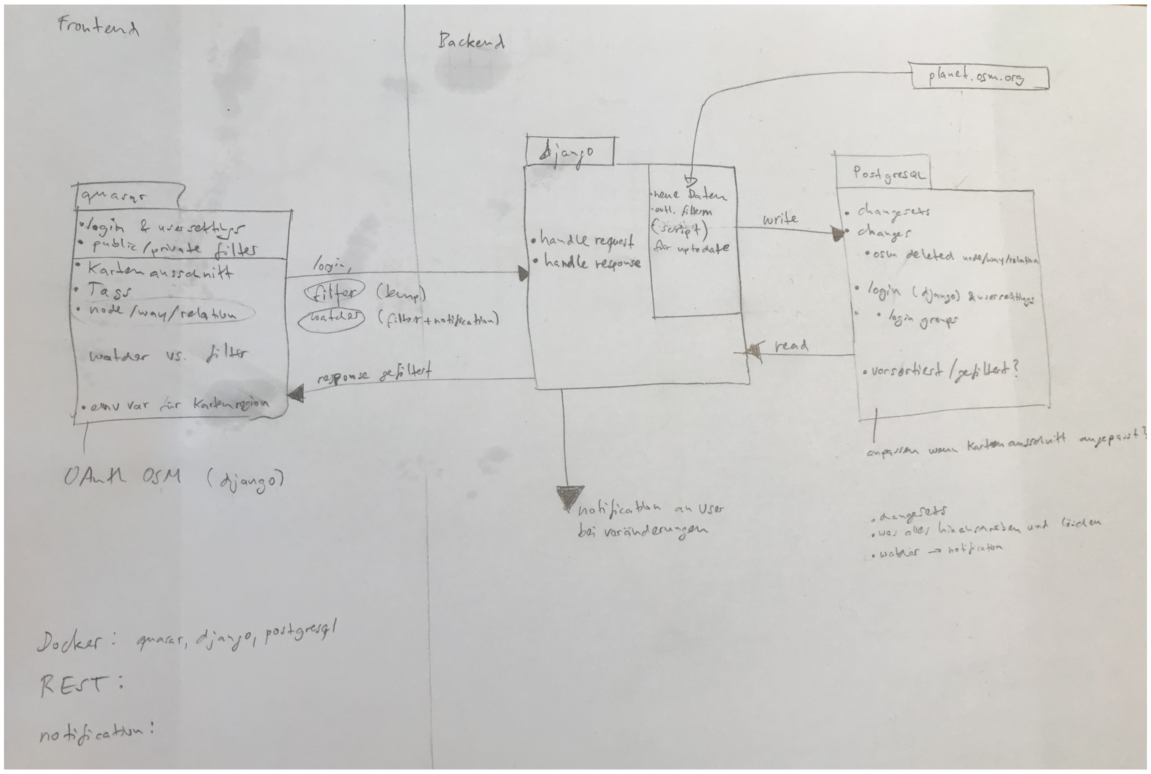
Issues

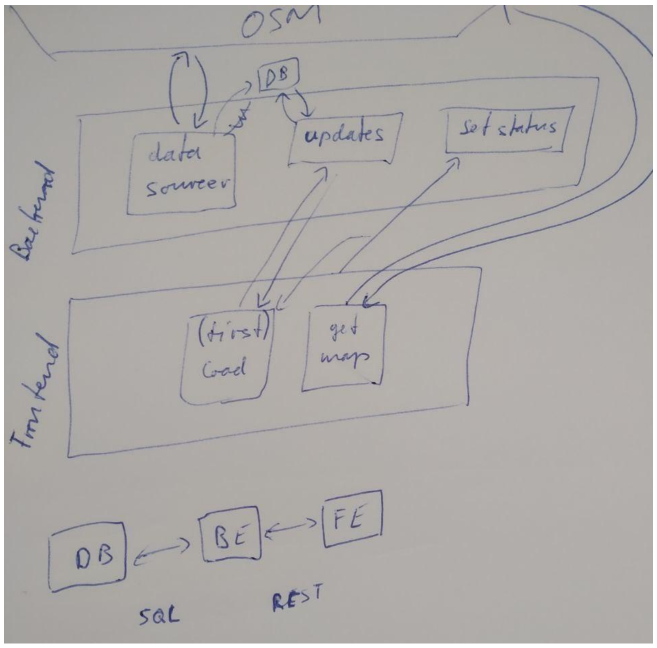
iid	title	spent	total estimate	time tim.wisotzki	time samuel.lemmenmeier
94	fix & test db	7h		7h	
93	doc summaries	7h		3h	4h
92	meeting sprint16	1d 4h		6h	6h
91	documentation setup	7h		7h	
90	check ; wherever possible	30m			30m
89	account page raus	30m			30m
87	check and finnish links	2h			2h
85	abstand bei changesets (margin in div)	15m			15m
84	evtl. fuer kleinere bildschirme anpassen	2h			2h
82	litzens	30m		30m	
81	evtl. about seite stylen	30m		30m	
80	about seite (teams, ba-osm-monitoring-tool team) link hinzufuegen	15m		15m	
79	code aufräumen	2h 30m		1h 30m	1h
78	filter status	3h			3h
77	frontend: map-attribution-links	5h			5h
76	meeting spring15	5h		2h 30m	2h 30m
75	changeset status	2d 6h		2d 3h	3h
73	DOKU requirements, todos, use-cases	1h 30m		30m	1h
72	meeting spring14	6h		3h	3h
70	filter shape query	6h			6h
69	development filter	1d 1h		2h	7h
67	filter datum	4h			4h
66	development - map	1d 4h			1d 4h
65	meeting spring13	1h 40m		50m	50m
64	meeting sprint12	1d		4h	4h
63	layout	6h		2h	4h
61	tag filter	1d		2h	6h
60	cron job	3h		3h	
59	rating system einbauen	4h			4h
58	meeting sprint11	1d 4h		6h	6h
57	improve db	2d 7h		2d 7h	
56	documentation	12d 1h		7d 5h	4d 4h
55	meeting sprint10	1d 40m		4h 20m	4h 20m
54	prototype - map - filter	4d 7h			4d 7h
53	meeting sprint9	7h 30m		3h 45m	3h 45m
52	meeting sprint8	4h 30m		2h 15m	2h 15m
51	prototype - quasar	1d 4h			1d 4h

iid	title	spent	total estimate	time tim.wisotzki	time samuel.lemmenmeier
50	meeting sprint7	1d 1h		4h 30m	4h 30m
49	prototype - quasar - map	2d 6h			2d 6h
48	prototype - django	2d 7h		2h	2d 5h
47	prototype - docker - allgemein	2d 1h		1d 7h	2h
46	prototype - allgemein	1d 5h		3h	1d 2h
45	prototype - quasar	3d 4h		1d 2h	2d 2h
44	prototype - ninja api	6h			6h
43	prototype - docker - quasar	4h			4h
42	prototype - docker - postgres	7d		6d 4h	4h
41	meeting sprint6	1d 3h		5h 30m	5h 30m
40	osmcha app testen, code anschauen und DOKU	1d 1h		4h	5h
38	Zwischenpräsentation mit Laurent Metzger	1h		30m	30m
37	db-laden	2d 4h		2d 4h	
36	django_rest-api einarbeiten	4h			4h
35	meeting sprint 5	1d 1h		4h	5h
34	prototype erstellen	1d 5h 30m		5h	1d 30m
33	osmhistorydb einarbeiten	6h 30m		5h	1h 30m
32	DOKU Generelles	2h 15m		1h 15m	1h
31	db schema überlegen	5h		3h	2h
30	meeting sprint 4	4h 40m		2h 20m	2h 20m
29	python einarbeiten	3h 30m			3h 30m
27	daten ziehen mit django	1d 2h			1d 2h
26	architektur mit docker erstellen	1d 30m		1d 30m	
25	meeting sprint 3	3h 40m		1h 50m	1h 50m
24	quasar einarbeiten	1d 7h		6h	1d 1h
23	repo sharen	15m		15m	
22	meeting sprint 2	4h 10m		1h 50m	2h 20m
21	general administration sprint 1	1h		1h	
20	overview links	30m			30m
19	time-tracking vorbereiten	1h			1h
18	grobe architektur festlegen	1d 15m		6h 15m	2h
17	Server bestellen	15m		15m	
16	DOKU Risiko	30m		30m	
15	meeting sprint 1	4h 30m		2h 15m	2h 15m

iid	title	spent	total estimate	time tim.wisotzki	time samuel.lemmenmeier
14	Docker einarbeiten	2d 30m		7h	1d 1h 30m
13	Leitfaden BA, Struktur BA beachten	2h 15m		1h	1h 15m
12	DOKU funktionsweise projekt beschreiben	4h		4h	
11	DOKU aufgabenstellung ueberpruefen	1h		30m	30m
10	DOKU dokumentation vorbereiten	1h 30m			1h 30m
9	django einarbeiten	1d 30m		3h	5h 30m
8	postgres DB einarbeiten	7h		3h	4h
7	OSM, changeset, tagging etc. anschauen	3d 1h 30m		1d 7h	1d 2h 30m
6	DOKU groben projektplan erstellen	1h		1h	
5	DOKU Requirements aufschreiben	30m		30m	
4	einarbeiten bestehende arbeit (alte SA)	2d 7h 30m		1d 7h	1d 30m
3	DOKU projekt vorgehensweise / sprints bestimmen	30m		30m	
2	Frontend recherchieren / festlegen	2h		2h	
1	Gitlab einrichten	15m		15m	

Architecture drafts





API as JSON

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "NinjaAPI",
    "version": "1.0.0",
    "description": ""
  },
  "paths": {
    "/api/filters": {
      "post": {
        "operationId": "osm_monitoring_tool_api_create_filter",
        "summary": "Create Filter",
        "parameters": [],
        "responses": {
          "200": {
            "description": "OK"
          }
        },
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/FilterInSchema"
              }
            }
          },
          "required": true
        }
      },
      "get": {
        "operationId": "osm_monitoring_tool_api_list_filters",
        "summary": "List Filters",
        "parameters": [],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "title": "Response",
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/FilterOutSchema"
                  }
                }
              }
            }
          }
        }
      }
    },
    "/api/filters/{filter_name}": {
      "put": {
        "operationId": "osm_monitoring_tool_api_update_filter",
        "summary": "Update Filter",
        "parameters": [
          {
            "in": "path",
            "name": "filter_name",
            "schema": {
              "title": "Filter Name",
              "type": "string"
            },
            "required": true
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          }
        },
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/FilterInSchema"
              }
            }
          },
          "required": true
        }
      },
      "delete": {
        "operationId": "osm_monitoring_tool_api_delete_filter",
        "summary": "Delete Filter",
        "parameters": [
          {
            "in": "path",
            "name": "filter_name",
            "schema": {
              "title": "Filter Name",
              "type": "string"
            },
            "required": true
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          }
        }
      }
    },
    "/api/changesets/{changeset_id}": {
      "put": {
        "operationId": "osm_monitoring_tool_api_update_status",
        "summary": "Update Status",
        "parameters": [
          {
            "in": "path",
            "name": "changeset_id",
            "schema": {
              "title": "Changeset Id",
              "type": "integer"
            },
            "required": true
          }
        ],
        "responses": {
          "200": {
            "description": "OK"
          }
        },
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/StatusInSchema"
              }
            }
          },
          "required": true
        }
      }
    },
    "/api/changesets": {
      "post": {
        "operationId": "osm_monitoring_tool_api_changesets",
        "summary": "Changesets",
        "parameters": [],
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "title": "Response",
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ChangesetSchema"
                  }
                }
              }
            }
          }
        },
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/FilterAppliedSchema"
              }
            }
          },
          "required": true
        }
      }
    }
  },
  "components": {
    "schemas": {
      "FilterInSchema": {
        "title": "FilterInSchema",
        "type": "object",
        "properties": {
          "name": {
            "title": "Name",
            "type": "string"
          },
          "user": {
            "title": "User",
            "type": "string"
          },
          "poly": {
            "title": "Poly",
            "type": "string"
          },
          "isRectangle": {
            "title": "Isrectangle",
            "default": false,
            "type": "boolean"
          },
          "tags": {
            "title": "Tags",
            "type": "string"
          },
          "dateFrom": {
            "title": "Datefrom",
          "type": "string"
        },
        "dateTo": {
          "title": "Dateto",
          "type": "string"
        },
        "changesetStatus": {
          "title": "Changesetstatus",
          "type": "string"
        }
      }
    },
    "FilterOutSchema": {
      "title": "FilterOutSchema",
      "type": "object",
      "properties": {
        "id": {
          "title": "Id",
          "type": "integer"
        },
        "name": {
          "title": "Name",
          "type": "string"
        },
        "user": {
          "title": "User",
          "type": "string"
        },
        "shape": {
          "title": "Shape",
          "type": "string"
        },
        "isRectangle": {
          "title": "Isrectangle",
          "default": false,
          "type": "boolean"
        },
        "tags": {
          "title": "Tags",
          "type": "string"
        },
        "dateFrom": {
          "title": "Datefrom",
          "type": "string",
          "format": "date"
        },
        "dateTo": {
          "title": "Dateto",
          "type": "string",
          "format": "date"
          },
          "changesetStatus": {
            "title": "Changesetstatus",
            "type": "string"
          }
        },
        "required": [
          "id"
        ]
      },
      "StatusInSchema": {
        "title": "StatusInSchema",
        "type": "object",
        "properties": {
          "status": {
            "title": "Status",
            "type": "string"
          }
        },
        "required": [
          "status"
        ]
      },
      "ChangesetSchema": {
        "title": "ChangesetSchema",
        "type": "object",
        "properties": {
          "id": {
            "title": "Id",
            "type": "integer"
          },
          "user_id": {
            "title": "User Id",
            "type": "integer"
          },
          "user_name": {
            "title": "User Name",
            "type": "string"
          },
          "num_changes": {
            "title": "Num Changes",
            "type": "integer"
          },
          "min_lat": {
            "title": "Min Lat",
            "type": "number"
          },
          "max_lat": {
            "title": "Max Lat",
            "type": "number"
```

```
    },
    "min_lon": {
      "title": "Min Lon",
      "type": "number"
    },
    "max_lon": {
      "title": "Max Lon",
      "type": "number"
    },
    "created_at": {
      "title": "Created At",
      "type": "string",
      "format": "date-time"
    },
    "created_at_format_converted": {
      "title": "Created At Format Converted",
      "type": "string"
    },
    "closed_at": {
      "title": "Closed At",
      "type": "string"
    },
    "open": {
      "title": "Open",
      "type": "boolean"
    },
    "tags": {
      "title": "Tags",
      "type": "object"
    },
    "geom": {
      "title": "Geom",
      "type": "string"
    },
    "status": {
      "title": "Status",
      "type": "string"
    },
    "priority": {
      "title": "Priority",
      "type": "number"
    }
  },
  "required": [
    "id",
    "created_at",
    "created_at_format_converted",
    "closed_at",
    "open",
    "tags",
          "geom",
          "status"
        ]
      },
      "FilterAppliedSchema": {
        "title": "FilterAppliedSchema",
        "type": "object",
        "properties": {
          "name": {
            "title": "Name",
            "type": "string"
          },
          "user": {
            "title": "User",
            "type": "string"
          },
          "shape": {
            "title": "Shape",
            "type": "string"
          },
          "tags": {
            "title": "Tags",
            "type": "string"
          },
          "dateFrom": {
            "title": "Datefrom",
            "type": "string"
          },
          "dateTo": {
            "title": "Dateto",
            "type": "string"
          },
          "changesetStatus": {
            "title": "Changesetstatus",
            "type": "string"
          }
        }
      }
    }
  }
}
```

Meeting protocols

Meeting notes 09.02.22

```
Kickoff arbeit

 - Leitfaden von Claudia Furrer beachten
 - Tipps in pdf von Ihnen sehen: SABAPAMA_Tipps_15d.pdf
    -> vorher gedanken ueber kapitel machen

 - Fragen gerne sofort (Teams ist bevorzugt) (gerne immer gleich)

OSM:
 - vorherige + aenderungen = neu (changesets) ist sehr wichtig
 - tagging system (ebenfalls relevant)

Technologien:
 - docker muss eingesetzt werden
 - offen mit Python oder Go
 - Gitlab / Gitlab-CI

aufgabe:
 - tool das feststellt wenn aenderungen passieren (aufgrund changeset)
    - transparent abspeichern (nutzerbasiert oder geshared)
 - frontend (mit Python/Django / anderes moeglich (Nicola mit Vue.js bevorzugt))


Admin:
 - alte arbeit
 - andere tools
 - was sind changesets / tagging
 - technologie festlegen (django, vue, ...)
 - docu art
 - sitzungsrythmus bestimmen
 - repo etc. aufsetzen
 - aufgabenstellung ergaenzen falls inputs noetig
 - postgres
 - frontend bestimmen


OSM account erstellen
 - borsnet (bestehendes tool), etwas mit rumspielen


keller im hintergrund dabei -> nicola betreut
von aussen gab es viele positive intressensmeldungen (Berlin stammtisch /
   NRW verkehrsverbund)

schutz und rettung hat software von Mentz GmbH -> taeglich daten aus OSM abziehen
        und in Infosystem
    verarbeiten (dann von S&R genutzt)
 - routing von einsatzfahrzeugen werden aufgrund von OSM routing dispached
 - Unfallort kann eingegeben werden

bei fragen zum backend steht nicola zur verfuegung

SRZ Kloten anschauen gehen

Kompletter rewrite von bosnet.ch (bestehende dependency von OSMCha rauswerfen)

dashboard potm.osm.ch

Filter definieren
 - Name, Tags (amenity=hospital), Kanton,
--> zurzeit kann es nur nach user filtern

Geonick (Prof. Keller) useraccount


MVP -> filtern nach user+tags als "Gruene wiese"

Filterfreigabe

Technische Fragen sicher an Nicola

Ideen:
    alle changesets zusammenfassen und in DB -> dann querien

anderes Team mit aehnlichem Technologie-stack (evtl. austausch)

osm-data-pipeline


Aufgabe:
 - Datenbank erstellen mit jeweils aktuellen (changesets)
 - von Webapp (zuweisung User auf abgeschlossen)
 - update DB with changes

Login mit Django und OAuth (gleicher user wie OSM)

ein prozess laedt aenderungn
der zweite updated map


dokumentation englisch/deutsch egal (60-100 seiten umfang)
 -> im pdf stehen tips zur dokumentation (muessen nicht alle benutzt werden)
 -> ca. 50% der Zeit geht fuer Dokumentation drauf


OST erwartet Abstrakt:
    1) Abstract gemaess eprints / monitoring (eprints.ost.ch/id/eprint/916)
    2) Broschuerenabstract (management-summary gekuerzt)

Management Summary (3 Seiten mit 2-3 Grafiken (ausgangslagen fuer Broschueschen-
    abstract)


In mitte von Arbeit gibt es review von (anderem Professor)
Schlusspresentation (muendlich) (Experte: Claude Eisenhut (extern))



Es gibt eine fixe anzahl stunden, alles andere ist nicht so relevant

Lukas Buchli Ibuchli (bei fragen zu OSM History DB / Pipeline)


Erstes meeting: 14:00 - 15:00
```

meeting notes 04.03.22

```
technologies that will probably be used
 - docker compose
 - quasar
 - postgres

things to do next
 - schweizer datenbank mit user anschauen
 - werden daten aufbereitet, resp. sind diese besser nutzbar als offizielle
 - grobe architektur document erstellen
 - tag von node bestimmen

next meeting
 - Mo:  11:00
```

meeting notes 11.03.22

```
## what we did:
 - setup gitlab, OSM accounts, etc.
 - general project preparation (read requirements, etc.)
 - get to know technologies (OSM, django, quasar, etc.)
 - rough project outline
 - prepare documentation

## questions
 - what are the drawbacks of OSMCha and why should it be removed as a
    dependencies?
 - according to chapter 4.2 of the "Leitfaden fuer Bachelor und Studienarbeiten"
    the Task-description should include an overview about the scope as well as
    the start
    and end dates, which are currently missing.
 - which db for changesets? osmhistorydb-ch just for the beginning?
       planet.osm.org?
 - which tags should be available as filter, maybe mapping needed?
 - elements related to changeset
 - is it possible to get an account for the existing application?
 - what other tools can be used or should it be from scratch?


## notes
was sind die naechste schritte

wie weit sind wir gekommen


vor dem meeting (1-2 h, oder 1 tag vorher)
 - grober stand
 - protokoll (was haben wir geschaft, wo stehen wir, was wollen wir erreichen)

hauptziel:
 - rewrite ohne OSMCha
 - genaue richtung danach ist offen (wir entscheiden was wollen wir, was macht
   sinn)
 - alle "Must" sind zwingend
   - fuer development hat nicola je noch einen test-server wo nur mail konfi-
   guriert werden muss
 - probleme OSMCha diverse:
   - uralte django version (python eingeschraenkt)
   - ideen klauen, aber nicht projekt selbst
 - kein Django projekt verwenden
 - Muss sicher mit docker laufen
 -

 - bei problemen koennen wir uns immer an nicola wenden

 - sicher ziemlich allgemein filterbar -> groesseres interesse an (Spitaeler etc.)
   - einige dinge vordefiniert + selbst definierbar

 - es gibt sicher tools die gewisse sachen machen -> nicht alles selber schreiben
 - basiert auf osmchange (minutely updates, etc.) nicht changesets

 - hauptanwendungsfall ist schweiz
   - moeglich umzuwandeln auf anderes
   - [schweiz](https://planet.osm.ch/replication/)
   - sicher configurierbar

 - repo mit Nicola teilen
 - wichtigste [tags](https://github.com/Schutz-Rettung-Zurich/srz-edi/blob/main/
       critical_nodes.md)

 - bestehendes [tool](https://srzedi.srz.borsnet.ch/)
 - wenn wir neu schreiben, dass es gut beschrieben ist wie man es instaliert etc.
    - minimal dokumentiert in README


 - [C4](https://c4model.com/) -> zum beschreiben vom groben ins feine
    - muss nicht genutzt werden, aber evtl. hilfreich

 - frontend + backend im gleichen
    - doku mit drinn ist gut

 - koennen uns jederzeit bei ihm melden
 - architektur"durchstich" moeglichst schnell
    -> schnell anfangen mit coden
 - grundsaetzlich sind wir verantwortlich, er stoppt falls es gar nicht passt
    - wenn noetig greift er korrigierend ein
    -

 - 13:00 Uhr meeting am Donnerstag
 - fragen + protokoll im Teams channel

TODO:
 - repo teilen
 - grobes protokoll
 - projektplan ausarbeiten
 - wenn moeglich alles klein schreiben, nicht gross, kleinschreibung mischen
```

meeting notes 24.03.22

```
## current work
 - prototype


## problems
 - zusammenfuegen den komponenten
 - lokal funktioniert einiges
 - das zusa


 - lokal funktioniert einiges, aber zusammenfuegen ist nicht ganz einfach
 - verschiedene komponenten
 - viele kleine sachen
 -


 - aktueller stand jeweils auf branch machen
 -
Django fragen an Nicola (hat entsprechendes wissen)
```

```
das production dockerfile vom project ist etwas ueberkomplizierter

--> vorzugsweise mit django env -> vereinfacht das anapssen etc.

empfehlung alles mit docker zu machen -> sonst laeuft man spaeter in config-sachen


## Quasar
 - webpack ist sicher gut (default)
 - packet manager ist voellig egal (npm)
 - store ist nicht relevant aber default ist sicher nicht so schlecht

 - axios ist gut, nicola nutzt sonst fetch (ist nicht so relevant)
    - axios ist grundsaetzlich gut und etwas vereinfachtes AJAX (etwas verschoen-
        erte variante von AJAX)
    - es gibt vorlagen fuer login, CSRF token, etc. fuer axios online
 - Django-REST weggekommmen -> Django Ninja rest framework ist evtl. intuitiver
    - decoding encoding etc. ist wahrscheinlich mit ninja einfacher
    - "twelve factor" anschauen (was soll es machen, was soll es nicht machen)


 - Zwischenpraesentation mit Gegenleser Laurent Metzger noetig
    --> bilateral einen termin ausmachen (7.-8. SW)
    - stand arbeiten und aufgabenstellung praesentieren
    - keine folien notwendig (aber moeglich)

OSMChar nochmal genauer anschauen ob es moeglich waere dies anzuwenden

fuer 144 ist die kantonsgrenze wichtig

erwartet abstraktere definition was als filterkriterum angegeben werden kann
    (zurzeit implizit via gui definiert)


next meeting at tuesday at 11:00 -> Samuel laedt ein "BA Lemmenmeier Wisotzki"
     serie bis 9.Juni)
```

meeting notes 31.03.22

```
 - fuer naechste meetings -> kleine Demo (besonders wenn Keller anwesend)

naechste schritte:
 - DB ganz fuellen
 - Karte einfuegen
 - environments / variabeln erstellen

caddy file / nginx als reverse proxy fuer ganzes programm
 - fuer moment im moment nicht wichtig, solange funktionalitaet funktioniert
 - Nicola kann fuer dies etwas bereitstellen
```

```
alpine enhaelt ein package nicht, was zu problemen fuehren kann
 - bei Python mit Python:3.9 (macht nichts wenn es groesser ist)
 - fuer alles andere ist es egal

viel zeit fuer docker verwendet:
 - evtl. Nicola drueberschauen
 - beim ersten aufstarten daten reinziehen
 - einen separaten container um aktuell zu halten
 - evtl. Beispeiel von Nicola

 - bei datenbank kann zumindest im dev. auch manuell angefasst werden


 - Nicola & Lukas upgraden den [toolstack](https://github.com/sosm/osmhistorydb-
    ch) auf python3


 - Map mit "Leavelett"?? als "Gruene Wiese" projekt verwenden
    - Map mit OSMcha kann verwendet werden, aber darf keine logos drin haben.
    - Kein MapBox logo drin -> soll austauschbar sein mit eigenen karten (bspw.
        mit swisstopo karten)


 - schrittweise vorgehen:
 - die karte soll auf den jeweiligen kartenbereich anzeigen
    - bounding box anzeigen
    - "Goudy" die OSM Elemente anzeigen
 - als standard: swisstopo und osm karten bereitstellen
 - mit link auf openstreetmap (mit richtigen koordinaten)


 - overpass api
 - Kein Django-leaflet nicht anschauen -> bringt nichts
 - anschauen [vue-leaflet](https://github.com/vue-leaflet/vue-leaflet) fuer karte


yaml referenzen
&referenz

<<: \*referenz
--> vgl. script in chat



[openlayers](openlayers.org) oder leaflet


Lukas fragen bezueglich dump von HistoryDB -> egal wie alt, aber dass struktur
    passt
```

```
[overpass](overpass-turbo.osm.ch)
```

meeting notes 07.04.22

```
meeting with Chrisitan Nuessli and Michael Schmid from SRZ


hauptziel:
 - osm-history from project of the month auf etablierten stand bringen
 - zusaetzlich frontend mit entsprechenden filtern

christian nuessli:
 - applikations verantwortlich in SRZ notrufzentrale (Zurich, schwyz,
   schaffhausen)
 - qualitaetsmonitoring tool zusammen mit michael schmid verantwortlich

osm-applikation is kleiner teil von tools

michael schmid:
 - fuer geodaten zustaenden
 - gebaeude etc. register zustaendig
 - hausadresse ist am wichtigsten in einsatz
 - spital, schulen etc. werden aus osm bezogen

spezialfaelle werden noch nicht gut abgedeckt
 -> ziel: wissen wann altersheim oder spital geaendert werden


## Aufgabenstellung neu
Ziel ist die Überwachung von Änderungen an der OSM-Datenbank in einer Region
(Bbox, Polygon) auf Grundlage einer Liste geänderter Tags. Diese Filter werden
einem Datenkuratoren zugeordnet. Eine Änderung hat einen Status zugeordnet
(erledigt) und verlinkt auf OSM (bzw. Editoren). Beispiele: Überwachung der
Fahrradinfrastruktur in einer Region. Auflistung aller Änderungen an Objekten,
die den Schlüssel highway=cycleway (und andere) enthalten. Überwachen von Bäumen
in einer Region. Auflistung von Änderungen an Objekten, bei denen "natural=tree"
geändert wurde (die gefällt werden konnten und daher jetzt als natural=tree_stump
gekennzeichnet sind).


## current pain-points SRZ
 - datensetz werden nicht regional genug angezeigt
 - eingegrenzt auf Kantone

    --> changesets die zu grossflaechig sind

- wenn filter auf region gesetzt werden, sollen diese angewant werden
-> moeglichst nur hochqualitative changesets anzeigen
```

```
Q-rank ranking (von extern) die liste priorisieren

Fuer einsatzplanung wird [rescuetrack](https://apps.rescuetrack.com/de-de/)
verwendet



Staten
 - offen
 - in Bearbeitung
 - abgeschlossen -> aus liste entfernen

Einstuffung
 - Hoch / Mittel --> ist im Moment unklar wie genau und wird nicht gebraucht

bei verdaechtigen:
 - link zu OSM
 - link JOSM
 - link ID


-> tool sollte ausgeben was geaendert hat -> sonst muss selbst recherchiert werden


hauptfokus:
 - kantonsfilter sollen funktionieren
 - evtl. auch fuer Staete
```

meeting notes 22.04.22

```
## leaflet
http://leaflet.github.io/Leaflet.draw/docs/leaflet-draw-latest.html
leaflet-draw library um drawing tools und Polygon zeichnen einzubinden
- .js in import
- .css als style importieren (global in app.vue zum testen) aber nicht als style
    scoped
leaflet-toolbar & leaflet-draw-toolbar werden nicht gebraucht, kann man selber
    machen (draw start event)


## db
komplette historydb innerhalb docker ist noch in Arbeit. Ziel: so schnell wie
möglich fertig!
- queries auf vorhandener historydb Instanz ausprobieren z.B. alle Spitäler in ZH

## Metzger
hat sich noch nicht gemeldet, wir versuchen aber einen Termin zu finden

## Aufgabenstellung
```

```
siehe wiki vom IFS, ist gültig so

## Frontend
Grobstruktur wie bis jetzt ok
- grundsätzlich an vorhandener SA und OsmCha orientieren

### Features
- logo srz
- sortieren von changesets: neuste, priorität QRanking (code snippet kommt noch),
    status

### map page
- link in map auf osm spezifische Koordinaten
- Kartenauswahl: OSM.ch, OSM.org Standard, Satellit Swisstopo
- links unterhalb Karte: OSM, OSMCha, Achavi, JOSM, iD

### notifications (error etc)
- via notify quasar plugin lösen

### responsive
Fokus ist nicht mobile tauglich, wichtiger ist Grundfunktionalität
- changeset Liste als z.B. drawer ab tablet Grösse
```

meeting notes 28.04.22

```
 - wichtig, die arbeit ist wesentlicher bestandteil der Arbeit
    --> sollte nicht auf der strecke bleiben
    - links zu papers
    - auf was basiert es

 - schauen wieviel zeit braucht es fuer doku -> wichtig


 - manual steps during installation are ok

 - Frontend:
    - karte sollte moeglichst ganze seite fuellen
    - auswahl kartenbaterial als "OSM Standart" + "Luftbild"
    - Raender rausnehmen
    - API fuer SwissTopo wird von Herr Keller bereitgestellt
    - oben Logo + Text in Header anzeigen
    - Buttons auf Deutsch und in "CamelCase" -> "Karte, Filter, Benutzer" /
        "Offen, in Bearbeitung, Fertig"
    - alles auf Deutsch (wie semesterarbeit)

 - Changesets evtl. etwas schmaler machen -> changeset kommentar laenge median
    auswerten
    - 2 Zeilen danach mit ...

 - Development mailserver siehe message nicola:
```

```
# Django Settings:
# EMAIL SETTINGS
EMAIL_CONFIG = env.email(
'DJANGO_EMAIL_URL', default='smtp://user:password@localhost:25'
)
vars().update(EMAIL_CONFIG)
# im env file oder docker environment:
# https://django-environ.readthedocs.io/en/latest/types.html
DJANGO_EMAIL_URL: smtp://user:password@maildev:1025
maildev:
image: djfarrelly/maildev
command: >-
bin/maildev --base-pathname /maildev -w 8001 -s 1025 --incoming-user
user --incoming-pass password
ports:
- 8001
localhost:8001/maildev
```

meeting notes 05.05.22

```
offizielles logo vgl. [link](https://github.com/Schutz-Rettung-Zurich/srz-ed    i/
blob/main/logo_SRZ.png)

evtl. Seite hinzufuegen mit akredirungen
 - arbeit zusammenarbeit mit OST und SRZ
 - namen Tim, Samuel
 - evlt. betreuer etc.

 frontend favicon anpassen -> von alter arbeit
  -> eigene loesung ist auch ok

 evtl. direkt Sascha fragen -> Nicola vermittelt

 documentation auch machen
  -> bericht ist etwa 35 - 40%
  -> umsetzung ist auch teilweise im bericht enthalten

 --> laut Keller ist docu und app etwa gleichgewichtet

buttons wieder nach rechts
alle floats auf 7 stellen runden (oder kuerzer)
name von karten (osm standard, satelit, etc.)
kanten wechsel in standardkarte
```

Meeting notes 12.05.22

```
## Nuessli war anwesend in ersten 20min
```

```
### Demo und Anmerkungen

- weitere moegliche Filter:
    - vordefinierte bounding boxen
    - range datum oder oder z.B. last 3 month

    - sharen filter oder filter für alle gültig
    - filter evtl global anstatt sharen

- Internationalisierung: lieber deutsch
- name tool: Osm Monitoring

### Weiteres Meeting ohne Nuessli
- footer: nicht nötig wenn scrollbar weg ist
- border weg von allen pages

- changeset: id (fett) mit/ohne icon und user (fett) mit icon
             icon comment
             icon datum
             evtl. sources, hashtag (aber noch nicht implementieren)

### fragen meeting
- geodjango: Ist gute Wahl und noetig fuer postgis mit django
    gdal: version 3.9 nehmen oder 3.10 und dann ubuntu jammy base image gdal
- wie koordinaten speichern: polygon type bez. geography wenn moeglich sonst
    geometry

- status Problem:
    - Extra Tabelle mit update Funktion
```

Meeting notes 19.05.22

```
## Demo und Anmerkungen
- aktiver Filter anzeigen
- Kartenmaterial besprochen
- status ist in Vorbereitung
- 3 Pixel bei unterem Rand ist dies korrekt
- copyright für alle 3 Karten abhängig von Kartenmaterial neben SRZ link

- 3 Namen für Karten und links sowie copyright links etc bekommen wir noch über!

- icon material Design, Copyright und Datenherkunft erwähnen, mindestens in Doku

## Protokoll durchgegangen

- was wir gemacht haben
- Frage wurde bereits in Demo besprochen
- Nächste Schritte: was wir in den nächsten 1-2 Wochen vorhaben
    - status
    - ranking
```

```
## Allgemeines
### TODO's für nächstes Meeting
- Liste für gute/mögliche Erweiterungen gewünscht
- Inhaltsverzeichnis der Doku
- Liste mit must-haves und nice-to-have

### Mögliche Erweiterungen
- datum range
- bounding box vordefiniert
- Ranking
```

Meeting notes 02.06.22

```
 - many small things and improvements


 - grundsaetzlich ist es ein



TODO:
 - compare to [changesetMD](https://github.com/geometalab/ChangesetMD/tree/
    python3-compat)


Nicola macht code review (dienstag)
    -> aktuellen stand auf main branch


Kartenmaterial
 - standard, swiss-style ok
 - luftbuild -> unten mit Swisstopo nicht cogis
 --> keller schickt korrekte attribution

srz in attribution ganz weglassen


evtl. Status button raus nehmen
 - alaternativ nur offene oder nur in bearbeitung anzeigen



das rauszoomen beim sateliten ist egal



abgabe:
 - 1x ausgedruckt (ohne spezielle bindung)
 - poststempel am 17. reicht
```

```
am 9. ist das letzte meeting

 - 1. management summary mit 3 bildern
 - 2. abstract in tool (zusammengefasst) (broschueren abstract)
 - 3. abstract aus arbeit als txt an herr spielmann fuer eprints
        (wissentschaftlicher abstract)



aufgabenstellung ist angepasst
```

Meeting notes 09.06.22

```
## code review
 - python code looks generally quite good

### Review 09.6.22
    Python:
    * jinja2 template satt string concatenation fur raw query
    * auto_generated_models.py scheint ungultig, wird das nicht verwendet
    (`db_table = 'spatial_ref_sys# python manage.py inspectdb`?
    Quasar:
    * pinia -> API-Interaction dorthin verschieben (https://pinia.vuejs.org/core-
    concepts/actions.html#actions)
    * Zu grobgranulare components, Untercomponents sinnvoller
    * Stile-Mix (zB imports mit `;` und ohne)
    * Magic strings, zB const api = axios.create({ baseURL: 'http://localhost:
    8000/api/' })
     * Das ist fur spateres deployen sehr schlecht, da es nie localhost sein
     wird. Besser: `{ baseURL: '/api/' }`.
    Kleinigkeiten
    * Schreibfehler: Contributers -> Contributors


## other notes
 - the code can also be improved after the project is handed in (documentation
    is fix)
 - the abstract should be started very soon -> needs to be final by the 13th


## software improvements
 - titel: "OSM Monitoring Tool" in bold
 - trennstrich ueber status buttons noch rausnehmen


 - evtl. noch filtern nach substrings in comment
```