# Austrian Flood Monitoring System

# Database Documentation

**Purpose**

This documentation aims to give insight into the currently decided and considered notions on the backend development of the web application.

**Main Tools in Use**

* Django Framework
* SQLite3
* Open Governmental Data Links
* Various Libraries for visualization

**Django Framework** will provide foundational support for the web application, which also includes the built-in user confirmation.

**SQLite3** will be the preferred RDBMS to store data and retrieve for several features of the application.

**Open Governmental Data Links** will be the main source of the real-time representation of the water-levels, danger zones and historical values to be used for the implementation of this web application’s backend.

**Visualization** will be offered by map representation of Austria with various layers to represent on one map of the country, which will visually inform the user. Also, statistical values need to be represented in chart forms. There are various libraries which can be used, some of them we discovered are:

* Pandas & GeoPandas (read data and turn into graph)
* Folium (map representation), Leaflet (additional library with JS)
* Matplotlib (statistical representation)
* Geodjango (map representation)

**Database Schema**

The database will be implemented in SQLite3. Database interactions will be managed through the Django Framework. The two main tables the web application will need are:

* User: stores user details
* Reports: stores flood-related incident reports submitted by app’s users

Additionally, to promise more flexibility for the required fields like verification, task tracking etc., the following tables might be needed:

* Votes: stores up- and downvotes for reports
* Tasks: tracks tasks assigned to emergency response agents by managers
* WaterLevels: stores real-time and historical water level data

There is still space for potential changes. However, in order to make the current state clearer, an ER Diagram is demonstrated below to show the approximate approach for now:

**A diagram of a company

Description automatically generated**

**Figure 1: ER Diagram of the Database of the Web App**

**User Roles and Permissions**

**Administrator**

* Manage user roles and oversee the platform.
* Cannot assign their own role to others.

**Emergency Response Managers**

* Verify and reject reports.
* Create and assign tasks to agents.
* Monitor real-time flood data and task completion.

**Emergency Response Agents**

* View and update assigned tasks.
* Provide feedback on task status.

**Registered Public Users**

* Submit location-based flood reports.
* Upvote or downvote other users' reports.

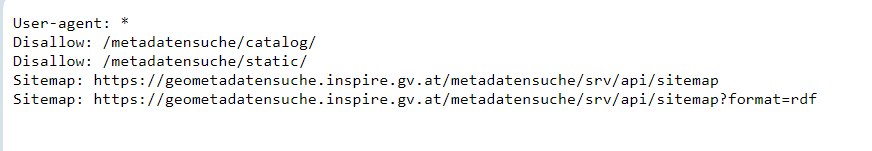
**Public Users (Unregistered)**

* View flood data but cannot submit or interact with reports.

**Back-End Implementation Details**

**Retrieving Data**

Interaction with online data sources (i.e. provided governmental links) - data has to be downloaded manually and converted outside or inside django to a data format that can be used for the application.

# Web scraping will not be possible for these websites and there is no sign of API support yet

**Parsing the Data**

After some research, it was observed that the governmental links provided contain file formats in GML and CSV. Parsing CSV files can be done using Django, but it is easier to convert gml files with the help of an external tool like QGIS to GeoJSON format. The process would begin by fetching the water level data and parsing, extracting the relevant fields to finally store it in the “WaterLevels” table.

**Map Visualization**

The web app’s system will deal with geographic data of water levels, flood zones and incident reports. This can be accomplished by using several python libraries for handling the backend implementation, the frontend could be represented by javascript libraries like Leaflet.

**Historical Analysis**

Historical data will be represented in a statistical way, meaning in charts and graphs. Therefore, libraries Pandas and Matplotlib will be utilized for analysis and visualization.

With pandas, water levels by date and station will be filtered to demonstrate the historical trends. For charting, data will be aggregated by time (e.g., monthly, yearly). Matplotlib will be used to generate series plots for historical water level data.

As a potential addition, JavaScript libraries could be used to make the graphs interactive (chart.js, plotly.js)

**Task Management and Reporting**

Creating: Emergency managers create tasks and assign them to responders. By using Django forms, tasks could be inputted, saved to the database and displayed on the map.

Once a user submits a report, the system will update the Reports table with the correlated status, and also display that report in a visual way on the report layer of the map.