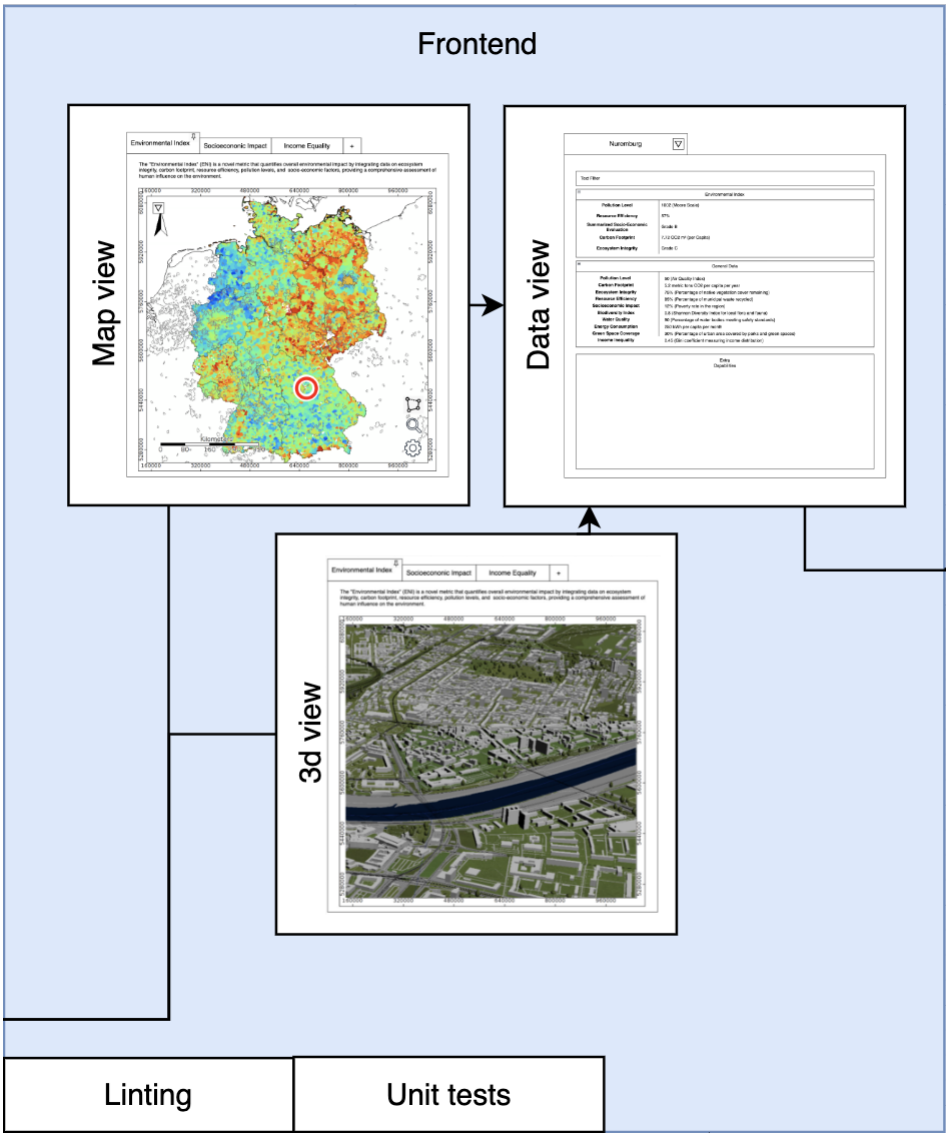
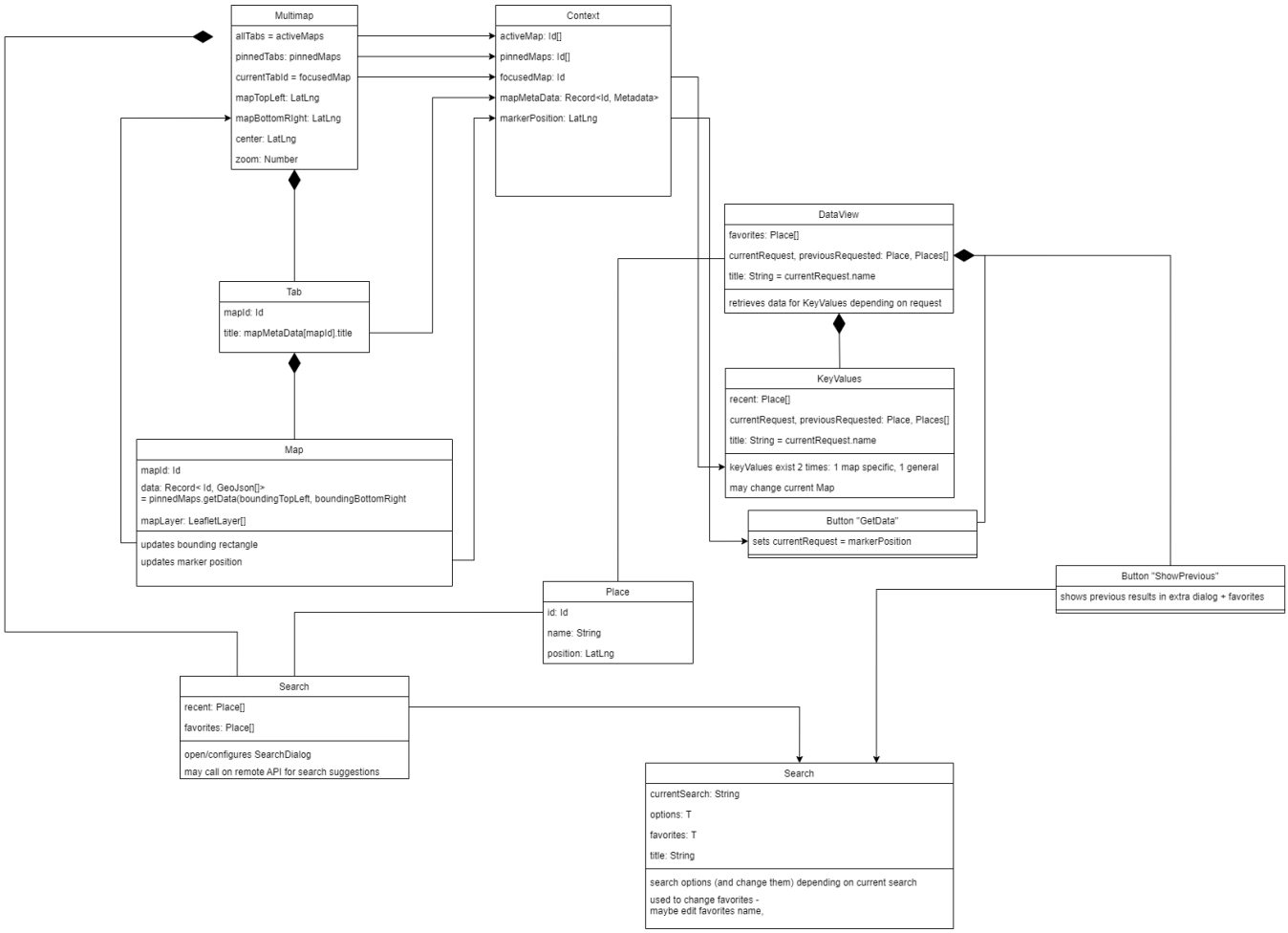
Very basic components

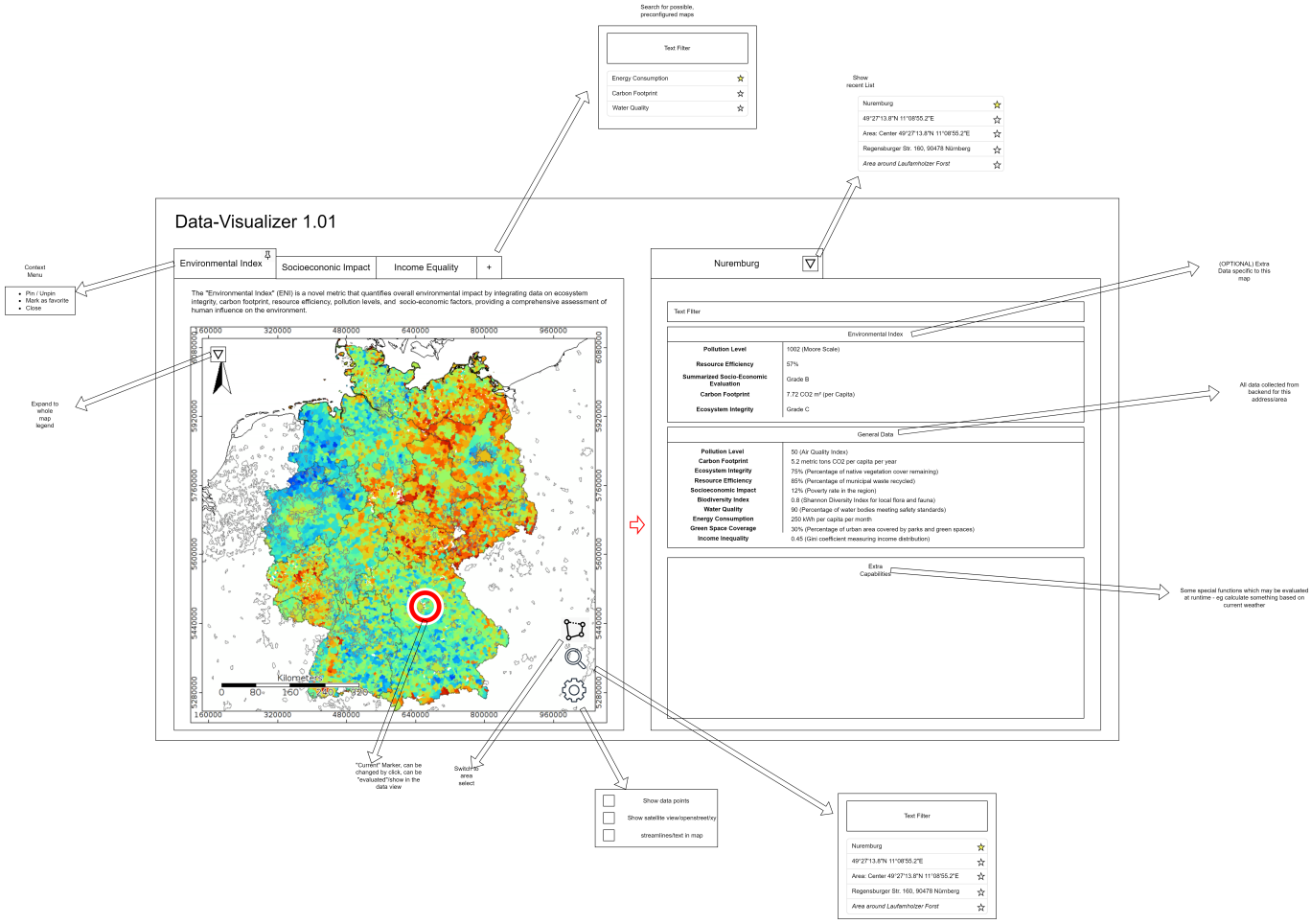


System Design (technology dependent)


Multimap
allTabs = activeMaps
pinnedTabs: pinnedMaps
currentTabId = focusedMap
mapTopLeft: LatLng
mapBottomRight: LatLng
center: LatLng
zoom: Number
Context
activeMap: Id[]
pinnedMaps: Id[]
focusedMap: Id
mapMetaData: Record<Id, Metadata>
markerPosition: LatLng
DataView
favorites: Place[]
currentRequest, previousRequested: Place, Places[]
title: String = currentRequest.name
retrieves data for KeyValues depending on request
Tab
mapId: Id
title: mapMetaData[mapId].title
KeyValues
recent: Place[]
currentRequest, previousRequested: Place, Places[]
title: String = currentRequest.name
keyValues exist 2 times: 1 map specific, 1 general
may change current Map
Map
mapId: Id
data: Record< Id, GeoJson[]>
= pinnedMaps.getData(boundingTopLeft, boundingBottomRight
mapLayer: LeafletLayer[]
updates bounding rectangle
updates marker position
Button "GetData"
sets currentRequest = markerPosition
Button "ShowPrevious"
shows previous results in extra dialog + favorites
Place
id: Id
name: String
position: LatLng
Search
recent: Place[]
favorites: Place[]
open/configures SearchDialog
may call on remote API for search suggestions
Search
currentSearch: String
options: T
favorites: T
title: String
search options (and change them) depending on current search
used to change favorites -
maybe edit favorites name,

UI Concept

Initial Concept with the following wireframe as a future vision:



How to build the Backend

Deployment

- when running the main command `npm run deploy` the Backend api container is build and deployed.
- to deploy the [[Data Pipeline]] navigate to the `/backend` directory and run `docker compose up -d` to start the docker containers for the datapipeline project and the database.

Development

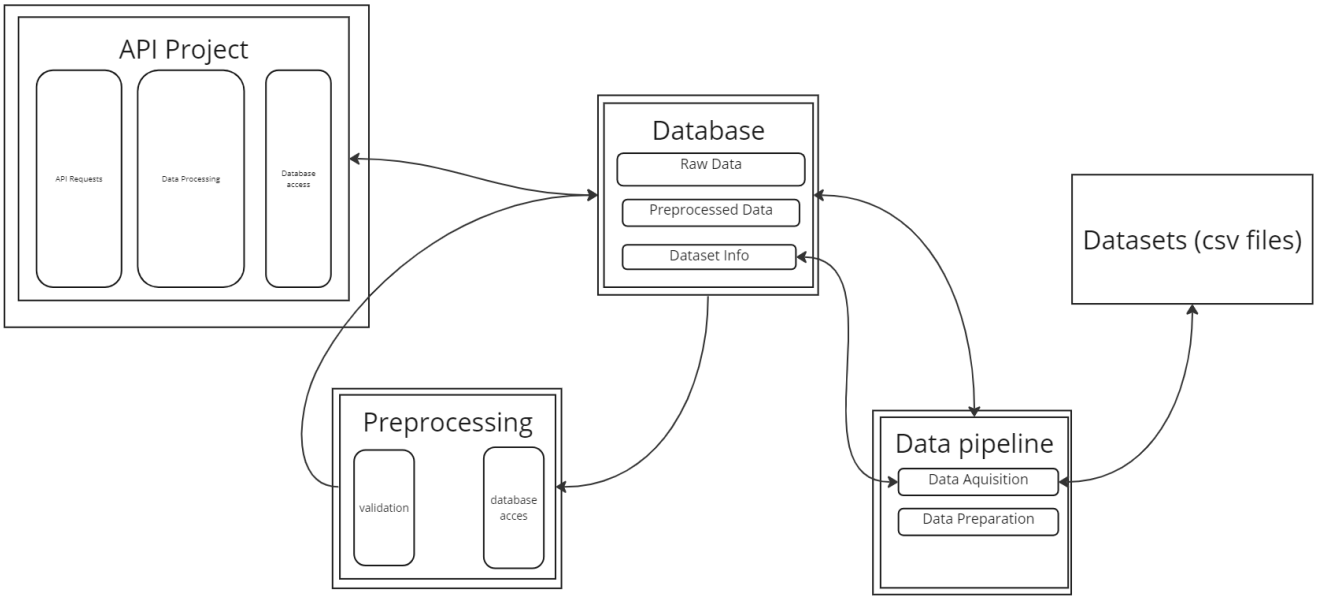
- Do develop both the [[Data Pipeline]] and the Backend Api you need Visual Studio installed. The solutions can be built and tested from within Visual Studio.
- Please look at the Configuration details given in [[Data Pipeline]] to be able to run the Datapipeline Project.

Backend Architecture overview

The infrastructure of the backend is split into four components:

- API Project
- Database

- Preprocessing
- Data pipeline



API Project:

The API project receives the requests from the frontend. Based on the request the data is read from the database. This data is processed and send to the frontend.

Database:

The database stores the data from the csv-source files, the pre-processed data and information about the datasets.

Preprocessing:

The preprocessing reads the raw data from the database. Does some calculations and filtering, and saves the new data in the database.

Data pipeline:

The data pipeline reads the data from a given csv-source file. This data gets parsed and filtered, and is the stored in the database.

Microsoft SQL Server provides robust support for geospatial data, allowing you to store, query, and analyze geographic and geometric data. Here are some of the key features and capabilities of SQL Server's geospatial functionality:

Key Geospatial Features in SQL Server

Spatial Data Types:

Geometry:

This type stores data that represents the Euclidean (flat) coordinate system. It is useful for applications that do not need to account for the curvature of the Earth.

Geography:

This type stores data that represents the Earth's curved surface, using the WGS 84 coordinate system. It is suitable for applications that need to handle real-world geography.

Spatial Methods:

SQL Server includes a variety of methods to perform operations on spatial data, such as:

STDistance(): Measures the shortest distance between two spatial objects. STIntersects(): Determines if two spatial objects intersect. STUnion(): Returns a geometry instance representing the union of two geometry instances. STArea(): Calculates the area of a spatial object. STLength(): Measures the length of a spatial object. STBuffer(): Creates a buffer area around a spatial object.

Spatial Indexing:

SQL Server supports spatial indexes, which significantly improve the performance of spatial queries. You can create spatial indexes on both geometry and geography data types to speed up spatial searches and operations. Integration with Other SQL Server Features:

Spatial data can be integrated with other SQL Server features, such as Full-Text Search, Reporting Services, and Analysis Services, to provide comprehensive solutions for data analysis and reporting.

Spatial Reference Identifiers (SRIDs):

Both geometry and geography data types support SRIDs, which identify the coordinate system and spatial reference system used by the spatial data. Examples of Using Geospatial Features

Creating and Storing Spatial Data:

```
CREATE TABLE Locations ( Id INT PRIMARY KEY, Name NVARCHAR(100), Location GEOGRAPHY );

INSERT INTO Locations (Id, Name, Location) VALUES (1, 'Central Park',
geography::STGeomFromText('POINT(-73.968285 40.785091)', 4326));
```

Querying Spatial Data:

```
SELECT Name FROM Locations WHERE
Location.STDistance(geography::STGeomFromText('POINT(-73.981923 40.768053)', 4326)) < 1000;
Creating a Spatial Index:

CREATE SPATIAL INDEX SIndx_Location ON Locations(Location);
```

Using Spatial Methods:

```
SELECT Location.STAsText() AS WKT, Location.STArea() AS Area FROM Locations;
```

Practical Applications

Mapping and GIS: SQL Server's geospatial features are used in Geographic Information Systems (GIS) for mapping and spatial analysis. Location-Based Services: Applications can use these features to provide location-based services, such as finding the nearest store or tracking assets.

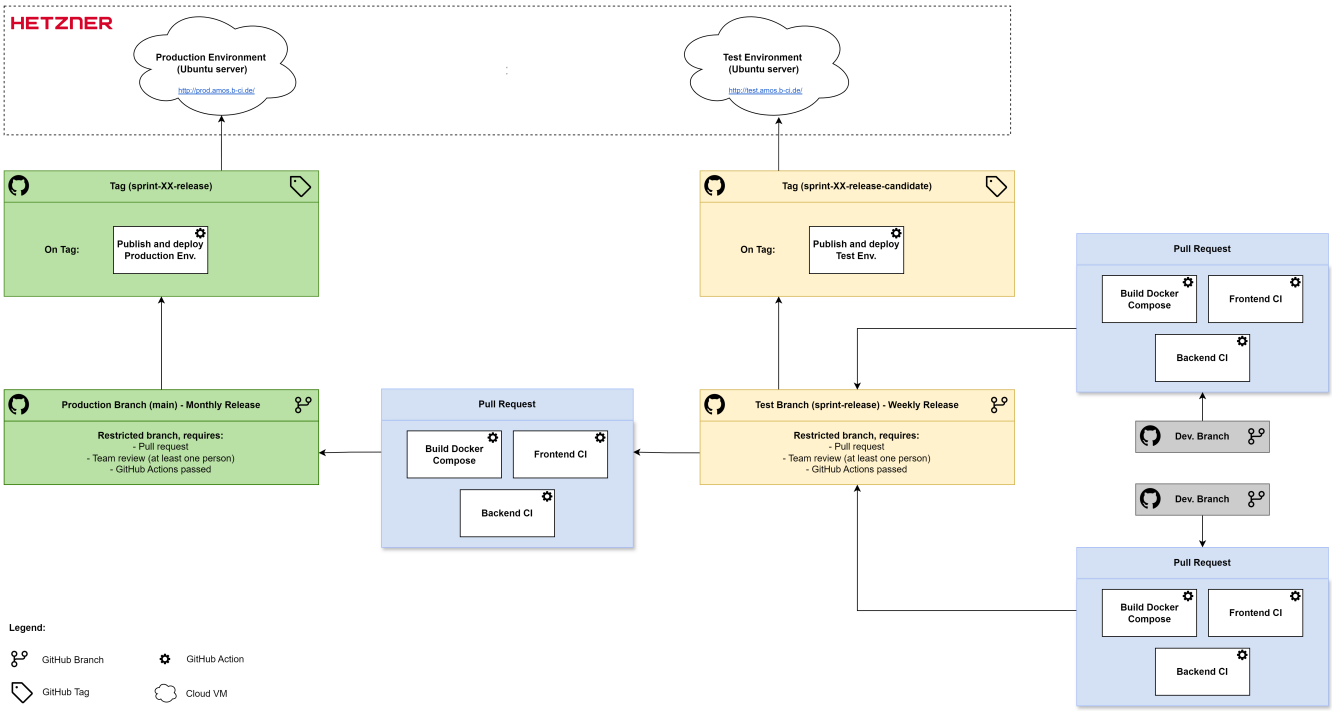
Urban Planning and Environmental Science: Geospatial data is crucial for urban planning, environmental monitoring, and resource management.

Conclusion

SQL Server's geospatial capabilities provide powerful tools for handling spatial data. By leveraging these features, developers and data scientists can create sophisticated applications that require spatial data processing and analysis.

CI-CD Pipeline Architecture

To follow the agile development principles (thus making the deployment quicker, easier, and fully automatic) we have developed a CI/CD pipeline visible in the diagram below.



GitHub Actions

We are taking full advantage of the following GitHub Actions workflows:

- Pull requests to the restricted branches (`main` and `sprint-release`):
  - Frontend CI - Runs linting and test checks for the frontend service.
  - Backend CI - Runs linting, formatting, and test checks for the backend service.
  - Build Docker Compose - Test if the Build of the whole Docker Compose file works.
- On new `sprint-XX-release` or `sprint-xx-release-candidate` tags:

- Publish and deploy Production Env. - Build and publish the newest Docker images and deploy them to the production environment Ubuntu server.
- Publish and deploy Test Env. - Build and publish the newest Docker images and deploy them to the test environment Ubuntu server.

🚀 Automatic Deployment

For the deployment of our service (both production and testing), we are using Ubuntu VMs hosted by the Hetzner provider. When a deploy action is started, all Docker Images are built and published to the `ghcr.io` container images repository of the `amosproj/amos2024ss04-building-information-enhancer` project. Then, an SSH connection is made, and the appropriate server pulls and deploys the newest container images using the Docker Compose file. The servers are restricted to only allow incoming connections for HTTP, HTTPS, and SSH protocols.

🖌️ Sprint Release Candidate

During each sprint development, new features are created on separate development branches and pull-requested to the `sprint-release` branch. This branch is restricted to only accepting pull requests, including the need for all passed actions and at least a single peer review.

Additionally, the Release Manager creates a new `release-candidate` tag, automatically starting the aforementioned deploy workflow to the test environment. Therefore, the `sprint-release` branch contains the newest and latest changes to the product and can be viewed here and is tagged as a `sprint-XX-release-candidate` each sprint.

💻 Production Release

The production system is located in the `main` branch. Similarly to the release candidate, when a new `sprint-xx-release` tag is created, an appropriate workflow is called and the newest code is deployed to the production server. Again, the `main` branch is restricted to only accepting pull requests, including the need for all passed actions and at least a single peer review. The production system can be viewed here.
