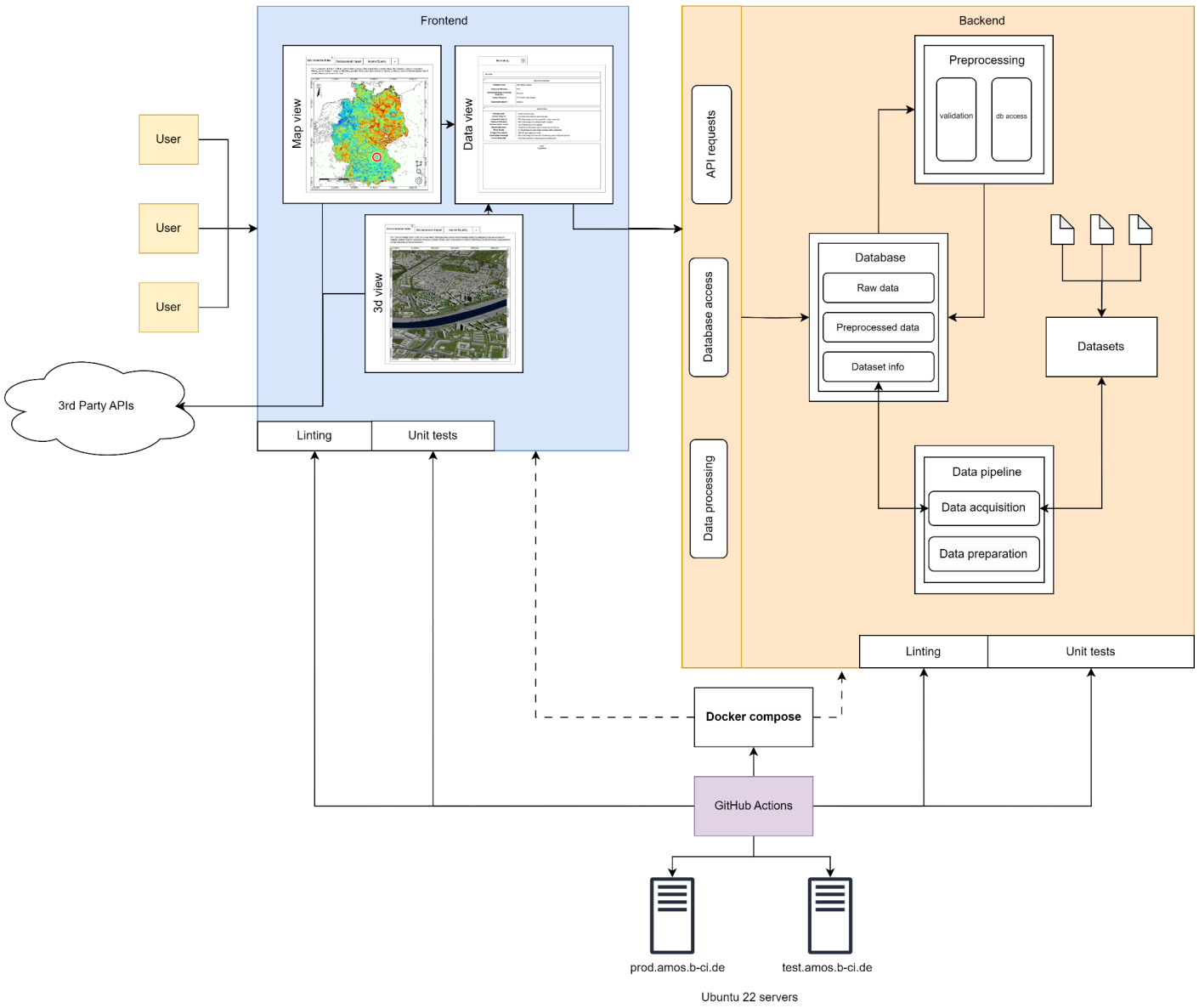
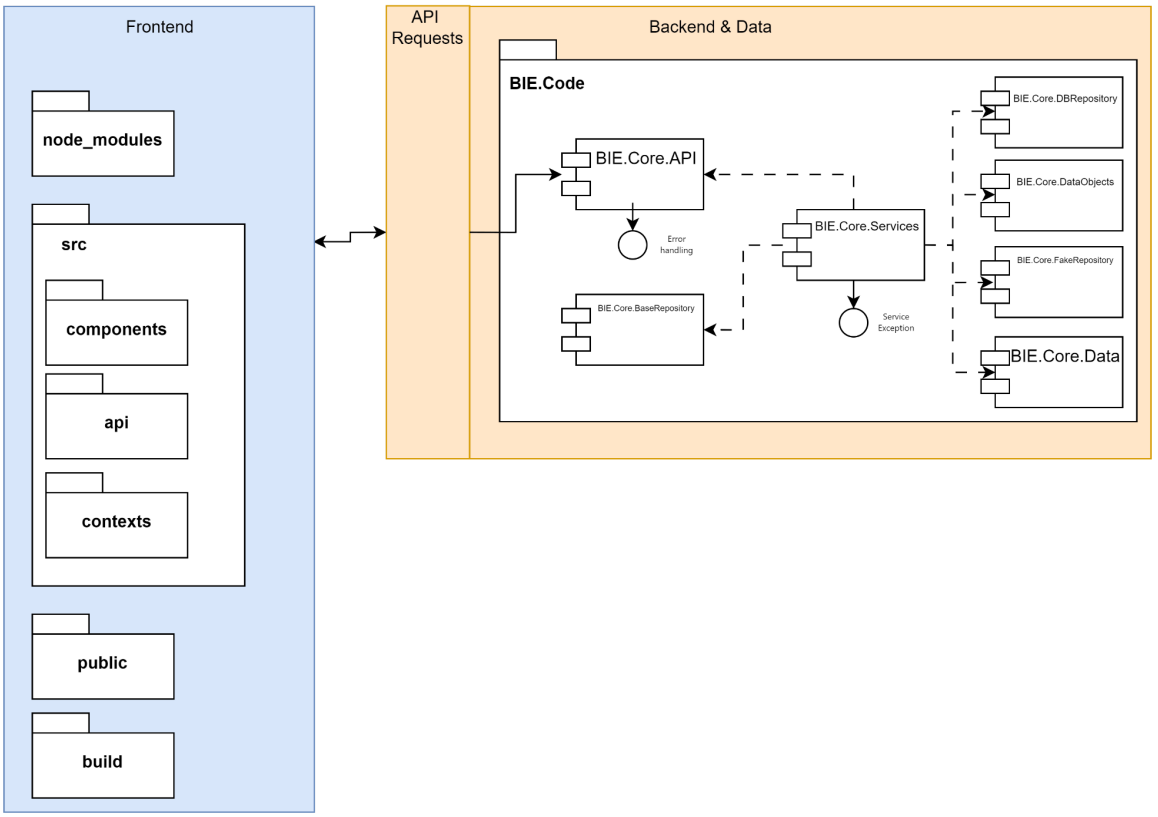
Building Information Enhancer - Software Architecture

Runtime Components Diagram



Code Components Diagram



Technology stack

Backend

- C# - backend programming language
- .NET - backend framework
- SQL Database - main database to use, with other databases possible in the future, such as MongoDB and others.

Frontend

- Typescript - frontend programming language
- React - frontend framework
- Vite - build tool for faster and leaner development experience for modern web projects
- Nginx (soon to be changed to Apache HTTP Server) - web server for serving frontend files
- Material UI - components library for easier development of frontend

DevOps (CI/CD Pipeline)

- Docker - deployment of both frontend and backend as containers
    - Docker Compose - management of multiple Docker images
- GitHub Actions - CI/CD automatic functions on push/pull requests
    - Eslint - frontend linting
    - Backend linting - TBD
    - Unit tests - frontend & backend TBD
- Hetzner Hosting Ubuntu Servers - test and production environments

Textual explanation of the diagrams and choices

The architecture of our Building Information System is designed to efficiently handle and process geospatial and environmental data to support energy savings and building sustainability analysis, with a focus on scalability, reliability, and ease of use. While the industry partner gave no strict requirements for the architecture, we plan to develop the architecture iteratively in order to support the vast, heterogeneous data sources we should handle, with more datasets and their types to be ingested and supported in the future.

Diagrams Explanation

The runtime components diagram

- The diagram represents the architecture of the product, containing both frontend and backend components. The frontend offers users different views like maps, data, and 3d interfaces, with the map and 3D views connecting to external 3rd party APIs like OpenStreetMap API. Users can interact with the product, selecting and exploring different datasets by connecting to the Frontend Container. Then, via API requests, the connection is made to the backend which propagates the requests through various routes and reads the data from the database. Since the main priority of this project was to have a data lake, the preprocessing of data sources will mainly happen on backend side and will mostly happen statically, thus being pre-prepared for the user to read. The data lake itself consists of the main SQL database. The diverse datasets will be digested by the data pipeline component and stored in the database, with the possibility of processing them by the preprocessing component later on. While currently we have only one database, we are planning to include more to increase the support of diverse datasets.
- The system also has the CI/CD pipeline used for the development setup, involving Docker for container building and management, GitHub actions for building and testing automation (including running the build docker-compose, linting, and test actions), and finally, two distinct Ubuntu 22 servers provided by industry partner for testing and production environments (one again, with the code deployed automatically by the GitHub action).

The code components

- Similar to the runtime diagram, the project is split into two parts, the backend and the frontend. For the frontend, we are using the React application folder hierarchy, with assets located in the `public` folder, libraries, and packages downloaded by the npm tool in the `node_modules` folder, `build` folder for the app files after running the build, and finally, all code files located in the `src` folder. The `src` folder is then split into three subfolders, `components` for React components, `api` for the typescript files related to API access and requests, and the `contexts` folder for storing high-level React contexts. More sub-folders might be added in the future.
- For the backend, we are using Visual Studio IDE, thus the components are split into several VS projects. This can be also seen in the form of folder division, with one folder for each of the VS projects, inside the main `BIE.Code` folder. Additional functions are also annotated in the form of white circles, such as Error handling.

Technologies Stacks Rationale

- **Backend**
  - As per the requirements from our industry partner, C# was chosen as the BE language of choice. With all backend SDs familiar with Microsoft's .NET, we have decided to use it as the development platform of choice.
  - For the Databases, the only industry partner requirement was to include an SQL database (if possible PostGIS) with other databases possible to include later. With the goal of developing the Data Lake, we might include other databases later during the project.
- **Frontend**
  - Typescript programming language for the Frontend was a technology requirement from our industry partner, but the framework was left for us to choose. During team formation, most of the SDs were familiar with the React Framework, thus making it the framework of choice for the Frontend. Additionally, the chosen technologies ensure a fast, reliable, and scalable web application and a development environment that supports rapid iteration and robust deployment of new features.
  - For the faster and easier development of the project, we have selected the Vite builder tool and the Material UI for faster development of the Frontend UI. Both choices were made based on the recommendations from Frontend SDs from their own experience.
  - Finally, for the web server of choice we have selected nginx docker image (again, due to familiarity with it), however, after the last industry partner meeting, a requirement to change to Apache HTTP Server was made, thus the web server will be changed in the following sprint.
- **DevOps (CI/CD Pipeline)**
  - Due to the nature of this project and the focus on Agile software development, we have chosen to implement an automatic CI/CD pipeline, including GitHub Actions (with repo already on GitHub, perfect integration could be achieved), Docker containers (requirement from the industry partner) and the deployment to the Hetzner hosting services (provided by free by the industry partner).
  - In order to automatically test our software, we plan to integrate linting and unit test checks, with tools still to be determined. The only exception is eslint library for the FE, which already shipped by using the Vite tool.
  - Finally, with a single CI/CD developer in our team, we decided to use their experience and recommendations for choosing the CI/CD tools.