OST
Eastern Switzerland
University of Applied Sciences

3D-Visualization of Utility Lines in the Browser Using Augmented Reality on Tablets

Term Project

Department of Computer Science
OST - University of Applied Sciences
Campus Rapperswil-Jona

Autumn Term 2023

Authors: Kaj Habegger & Lukas Domeisen
Advisor: Prof. Stefan F. Keller
Partner: Bitforge AG

Abstract

Locating underground utility lines such as water tubes or gas pipes is a complex task due to the fact that they are mostly hidden. The aim of this project is to create a solution which represents an innovative, cost-effective alternative to current solutions. Current solutions are dependent on expensive hardware and native applications.

This thesis introduces a web-based augmented reality (AR) application for visualizing underground utility lines on Android tablets. Thereby, addressing the challenge of locating such lines. It utilizes WebXR and WebGL to integrate Building Information Modeling (BIM) data, specifically Industry Foundation Classes (IFC), into a 3D web environment. This process involves converting IFC data for compatibility with WebGL, using Blender with the BlenderBIM add-on. The application's frontend was developed using Vue.js with Typescript, and the backend with Python Flask. Besides the already mentioned technologies, the application utilizes PostgreSQL, PostGIS, Three.js, Turf.js and Vuetify.

A key aspect of this project was the application's ability to approximate the positioning of utility lines. Using the device's geolocation and the lines' coordinates, the relative distance is calculated and applied in the virtual space. Additionally, the virtual space has to be aligned with the real world, which is achieved by rotating the virtual coordinate system based on the device's compass. This thesis identifies a challenge in aligning the virtual and real-world coordinate systems, due to the compass inaccuracy on mobile devices. These compass inaccuracies lead to a wrongly rotated virtual coordinate system and therefore imprecise placement of the utility lines.

Management Summary

Initial situation & approach

Locating underground utility lines such as water tubes or gas pipes is a complex task due to the fact that they are usually hidden below the ground. A solution that enables visualizing these lines in their actual environment could significantly simplify the localization process. Augmented reality (AR) offers a promising approach to display these hidden utility lines in the real-world context. There are already a few existing solutions, but all of them are dependent on costly hardware and license fees. A screenshot of an existing solution (vGIS) can be seen in figure 1. The aim was to create a similar application for the web which is simple in use and relies on a relatively low-cost external GNSS device.

This project uses the latest web-based AR technologies to visualize underground utility lines. The utility lines are defined by Industry Foundation Classes (IFC) data, which is a standard for Building Information Modeling (BIM) data. However, IFC data isn't directly compatible with 3D web-models. Hence, a conversion into a format that is usable by WebGL libraries like Three.js is needed.

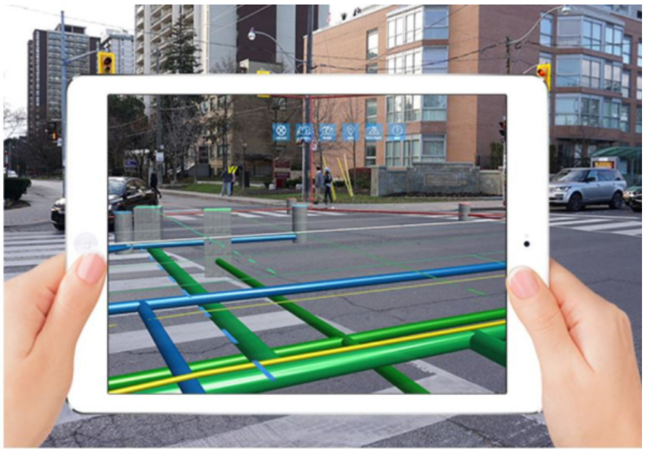


Figure 1: Task definition
Source: https://www.vgis.io/2019/01/09/kick-off-2019-gis-season-with-esri-federal-gis/

The development process began with a comprehensive review of existing technologies and potential approaches to create such an application. Following this, a prototype was developed in order to evaluate the feasibility of these technologies. This lead to the conclusion that WebXR, a standardized API, is the most convenient way to create AR and VR web experiences. To convert IFC files to an optimized 3D format for the web (glTF), Blender with the BlenderBIM add-on was utilized. The actual application was then implemented using the previously discussed technologies. Vue.js with Typescript was used for the frontend and Python Flask for the backend. The application and its development process heavily relies on real IFC data. Consequently, numerous people were approached to gather IFC data from them.

Result

The resulting application is able to approximately position the utility line models. Positioning of the models is calculated based on the device's geolocation and the coordinates of the utility lines. In figure 2 it can be observed, how the application was tested by manually defining the coordinates of the model. The coordinates were set to a point on the OST campus in Rapperswil. The visible model was created only for educational use and does not reflect real utility lines.

By using the educational model, many issues could be recognized and fixed. Afterwards, a field test took place in Stäfa, in order to test the application's behavior in a real-world scenario. The utility line in figure 3 is almost perfectly aligned with the street, which indicates a good accuracy in the application's mapping capabilities in real world scenarios.

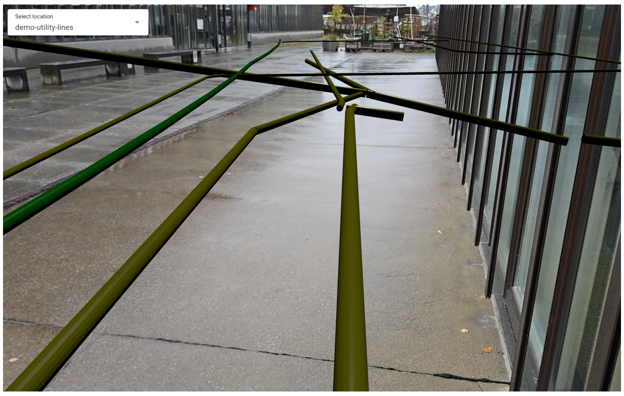


Figure 2: Demo test screenshot

Also, it can be observed in figure 3, that the positioning still lacks precision. This can be explained due to the fact that the virtual coordinate system must be perfectly aligned to the real-world. Otherwise, even a small inaccuracy in degree can lead to a notable misplacement, especially in large-scale models. Aligning the virtual coordinate system is done using on the compass. However, this is not accurate enough as the compass on mobile devices has a low accuracy and is easily deflected by nearby magnetic fields.

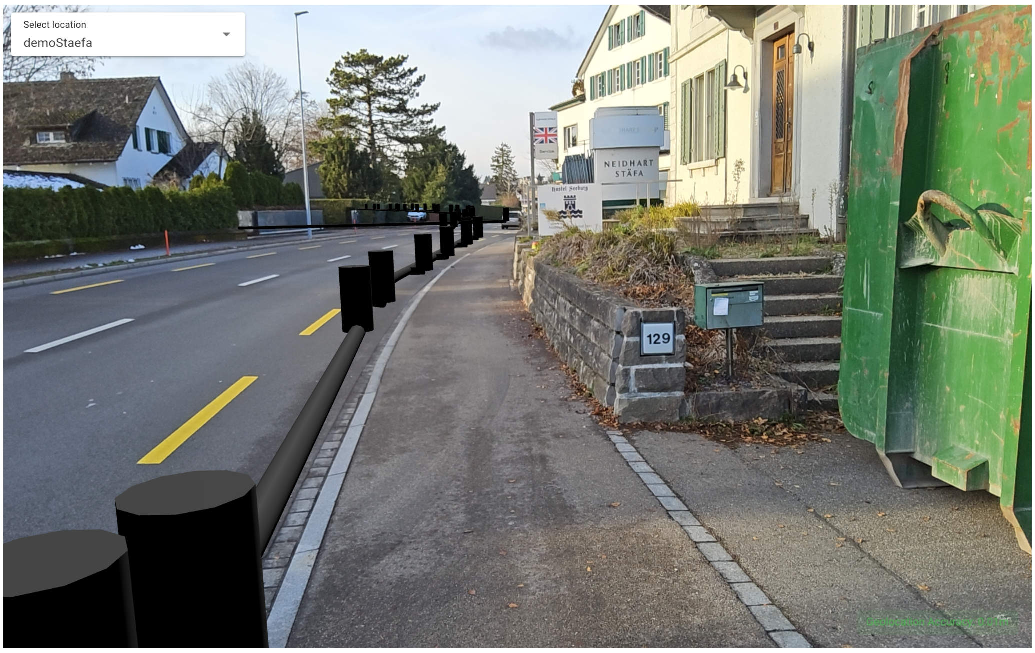


Figure 3: Stäfa field test screenshot

Outlook

There are many additional features that could be implemented into the application to enhance user experience and functionality. On top of that, the alignment issue must be addressed to achieve a more precise solution in terms of positioning. Therefore, this project will be continued as a bachelor thesis.

The additional features include interaction with the loaded utility lines, such as displaying information about specific pipes. Another yet missing feature is filtering and colorizing utility lines based on their types for a better overview. Currently, the user has to select the utility lines from a dropdown menu. Thus, it would improve the user experience when utility lines are dynamically loaded based on the user's position. Another feature which is currently missing is user authentication for the application as not all data should be visible to the public.

Contents

Task Definition

This chapter outlines the task details for the project titled “3D-Visualization of Utility Lines in the Browser using Augmented Reality on Tablets”. It is focused on the specific objectives and requirements of the task, conducted as a term project during the autumn term of 2023 in the Bachelor’s program in Computer Science.

Description

This project focuses on the application of Augmented Reality (AR) for visualizing utility lines such as water, electricity etc. on an Android tablet. The integration of AR technology aims to improve the management of underground utility lines, which are often hard to picture in a real-world context. The main goal is to gain experience with the latest technologies in the web (WebXR), data processing and hardware (GNSS). The primary users of this application will be professionals in architecture and the construction industry.

Tasks

- Develop a web application for Android tablets as a showcase.
- Processing of IFC data, primarily in version 4.3, to an optimized 3D format for web.
- Use GNSS technology for precise geolocation accuracy.

Technologies

Overview of the software and hardware technologies that were used for the project.

Software

- Frontend: TypeScript, Vue.js, WebXR, Three.js
- Backend: Python, Flask, Blender with BlenderBIM add-on (for converting IFC data to glTF)
- Database: PostgreSQL, PostGIS

Hardware

- Galaxy Tab S9+
- RTK Handheld Surveyor Kit (GNSS antenna)

Deliverables

1. Documentation including text-abstract, management summary and appendix
2. Brochure abstract
3. Declaration of originality

Participants

- Authors: Kaj Habegger & Lukas Domeisen
- Advisor: Prof. Stefan F. Keller
- External partner: Bitforge AG

Part I

Technical report

Introduction

This project was conducted as part of the term thesis in the autumn term 2023 by Kaj Habegger & Lukas Domeisen. The name of the realized web application will be referred to as “tubAR” in the following documentation, as this name was chosen by the team.

1.1 Problem definition

Locating underground utility lines such as water tubes or gas pipes presents a challenge because they are not visible above ground. AR allows for their visualization and would greatly ease the process of locating these lines. With the recent advancements in AR technology, particularly in web-based applications, there is an opportunity to apply these technologies in new areas. Current solutions available on the market rely on native applications, expensive hardware and high licensing fees.

1.2 Vision

The vision for tubAR is to provide a practical tool that overlays a digital representation of utility lines onto the real world. This should be made possible by using relatively low-priced hardware such as ArduSimple’s RTK Handheld Surveyor Kit. Low-priced additional hardware would make it more accessible to everyone. Such an application could improve the management and maintenance of these lines by providing a more intuitive and direct way of visualizing their location and layout.

1.3 Goals

- Develop a web application that integrates AR for displaying 3D models of underground utility lines.
- Create 3D models optimized for web-based visualization based on IFC data.
- Implement accurate geolocation features using GNSS receiver data to position the models in a virtual environment corresponding to their real-world locations.

1.4 Basic conditions

The project represents a term thesis, with each team member dedicating roughly 240 hours to its completion. This time commitment corresponds to 8 ECTS credits.

1.5 Approach

The approach to this project is based on SCRUM+, which is a combination of SCRUM and RUP. The project is broken down to four phases which focus on different aspects.

Phase 1: Inception

The inception phase lays the groundwork for the project. It involves setting up essential tools for issue tracking, time management, and version control. This phase also embraces risk evaluation and the establishment of project management protocols, including role assignments, process definitions and meeting schedules.

Phase 2: Elaboration

During the elaboration phase, the team focuses on defining and documenting both functional and non-functional requirements. This stage includes developing a prototype to explore AR capabilities, geolocation data handling and IFC file processing. Additionally, it involves preparing the CI/CD pipelines for the actual application.

Phase 3: Construction

The construction phase is primarily dedicated to the development of the actual application, ensuring that all specified requirements are met. The initial focus is on developing a stable backend for IFC file management, followed by the frontend implementation which contains the user interface and AR functionalities.

Phase 4: Transition

The final phase concentrates on completing the documentation and possibly addressing small remaining issues in the web application.

Current State Of The Art

This chapter examines the current landscape of AR technology for geospatial data visualization, focusing on existing solutions and the benefits of this project.

2.1 Existing solutions

This section provides a concise overview of the existing solutions, more specifically vGIS and V-Labs.

2.1.1 vGIS

vGIS is a platform specialized in visualization of geospatial data through augmented reality. Even though vGIS requires a native application, they support various devices such as Android devices, iPhones/iPads and Microsoft HoloLens 2. vGIS is versatile in its data format support, notably including IFC among many others. Furthermore, they provide capabilities for real-time interaction and modification of utilities, as well as tools for measuring distances. However, the annual licensing fee is $1250, which doesn't cover the costs of the devices themselves or optional components like LiDAR or photogrammetry technology.

2.1.2 V-Labs

Based in Switzerland, V-Labs is another service in the field of AR for geospatial data visualization. They also offer extended capabilities such as distance measurement and modification of GIS data. V-Labs focuses on mixed reality glasses rather than tablet based solutions. Their custom designed headset integrates essential hardware like a GNSS antenna with mixed reality glasses, providing a comprehensive solution. Although, their pricing is not publicly disclosed, the cost of the mixed reality glasses alone starts at CHF 3,800.

2.2 Disadvantages

Two primary drawbacks are evident in the current solutions: pricing and platform compatibility.

2.2.1 Pricing

High costs are a significant barrier in the adoption of AR technologies for geospatial data visualization. This is likely attributed to the costly hardware employed by solutions like vGIS and V-Labs and high development costs for native applications.

2.2.2 Platform Compatibility

The reliance on native applications by both vGIS and V-Labs introduces challenges in supporting a diverse range of platforms, limiting their accessibility and adaptability.

2.3 Benefits of tubAR

tubAR could be a promising alternative, leveraging the capabilities of WebXR to offer a cross-platform, web-based solution. This approach significantly increases the potential user base by eliminating the need for purchasing specialized hardware, except for the GNSS receiver device. Furthermore, the avoidance of expensive hardware like mixed-reality glasses makes tubAR a more cost-effective solution. This combination of accessibility, affordability and technological innovation positions tubAR as a significant advancement in the field of AR-based geospatial data visualization.

Evaluation

In this chapter several libraries and tools are evaluated. This is necessary to exclude ineligible libraries and tools for this project in advance. In addition, a prototype was developed to provide a more in-depth evaluation of certain technologies.

3.1 Test framework

This section evaluates and selects appropriate testing frameworks for the backend (Flask) and frontend (Vue.js) of the project.

3.1.1 Acceptance criteria

- Experience by team members with the framework
- Setup effort
- Provides needed functionality
- Ability to produce reports (JUnit report for GitLab and coverage report)
- Suitability for used technology (Flask / Vue.js)

3.1.2 Backend (Flask)

For the backend testing, the two frameworks which the team member have the most experience with, Unittest and Pytest, were evaluated.

Unittest

Unittest is the Python integrated test library. Therefore, it comes with close to zero setup effort. As the contemplated testing scenario for this project doesn't require any special testing functionality, Unittest fulfills this criterion. This framework has no built-in feature for generating test reports such as JUnit reports. The reporting functionality is limited to console output only. Unittest works totally fine with a Flask application. Sadly, this testing framework comes with a lot of boilerplate code and is therefore often avoided by developers.[1]

Pytest

Pytest is a flexible and extensible testing framework with a big plugin ecosystem. All team members are familiar with Pytest. Similar to Unittest, Pytest comes with almost no setup effort, but it needs to be manually installed from the Python Package Index (PyPi). Pytest fulfills the criterion of needed functionality such as Unittest does. Reports such as JUnit reports can be generated by Pytest using the pytest-cov plugin. The official Flask documentation uses Pytest. Therefore, Pytest is optimal for a Flask application

Conclusion

In conclusion, using Pytest is the best choice for this project, mainly because it's used in the Flask documentation which will make it much easier to integrate into this project. Additionally, only Pytest comes with a built-in reporting function to JUnit which is required by GitLab for analyzing the test results.

3.1.3 Frontend (Vue.js)

Since the build tool Vite is used to create the Vue.js application, the testing framework Vitest will be used. Also, all team members worked with Vitest before and it's recommended by the developers of Vue.js.[2] Overall, Vitest fulfills all given acceptance criteria.

3.2 AR library

Evaluation of AR libraries, mainly AR.js and WebXR.

3.2.1 Acceptance criteria

- Setup effort
- Adaptability to the project's needs
- Actively maintained
- State of documentation

3.2.2 AR.js

AR.js is a lightweight library for augmented reality on the web, which includes features like image tracking, location based AR and marker tracking.[3] Because it's a library it must be installed separately via node package manager (npm). Therefore, the setup effort for AR.js is moderate. AR.js already has location-based augmented reality integrated, which fits perfectly for this project as the final application will be dependent on the exact location. Though, it's quite probable that it won't be good enough since the external project partner already used it, and it didn't perform any good. AR.js is developed by a community and relies on time available by the community members to maintain the library as well as its documentation. It seems that the library and its documentation is not regularly updated as the last important change happened about four months ago as of today.[4]

3.2.3 WebXR

WebXR, which is also known as WebXR device API, is part of the web standard. It's a specification defined by the World Wide Web Consortium (W3C), the immersive Web Community Group and the Immersive Web Working Group. The Community Group works on the proposals in the incubation period and the Working Group defines the final web specifications to be implemented by the browsers.[5] This standard provides access to input and output capabilities commonly associated with augmented reality devices. The WebXR device API enables augmented reality applications on the web by allowing pages to detect if augmented reality capabilities are available, querying these capabilities and much more.[6] There is only one downside, which is that it would require manually transforming the GNSS geolocation data into local coordinates inside the AR space, because location-based AR is not supported yet. WebXR is actively maintained by the Immersive Web Working Group.[7] Furthermore, the standard is well documented.[8]

3.2.4 CesiumJS

It was wrongly assumed that CesiumJS is another AR candidate to build the fundament for this project's application. Though, CesiumJS is a library for mapping 3D data (such as globes and maps) but doesn't include any features for AR, so it doesn't fit the project's needs.[9]

3.2.5 Conclusion

Both AR.js and WebXR were tested in the prototype. WebXR will be used for the final application as it seems to be more stable than AR.js. A huge advantage is that WebXR device API is included in the web standard which makes it a lot easier to use. Also, all team members consider WebXR to be more future-proof.

3.3 3D data format

Given the project's focus on 3D data on the web, choosing the appropriate data format was a straightforward decision.

3.3.1 Acceptance criteria

- Optimized for web usage
- Compatibility with AR library

3.3.2 glTF

There is a vast number of 3D data formats. Most of them are not ideal for web application purposes. But there is one unofficial industry standard for 3D on the web, which is glTF.[10][11][12]

Graphics Library Transmission Format in its abbreviation glTF was explicitly developed for 3D on the web by the Khronos Group and is also maintained by them.

> **Official glTF description from the Khronos Group**
>
> glTF is a royality-free specification for the efficient transmission and loading of 3D scenes and models by engines and applications. glTF minimizes the size of 3D assets, and the runtime processing needed to unpack and use them. glTF defines an extensible, publishing format that streamlines authoring workflows and interactive services by enabling the interoperable use of 3D content across the industry.[13]

glTF offers two possible file extensions .gltf (JSON/ASCII) and .glb (binary). The binary variant is used for this project as it bundles all required data including materials in one file. Whereas the JSON format is separated into multiple files on export. Additionally, the transmission of the data can be further enhanced by using the binary format. The only drawback of using GLB is that using a 3D software is mandatory to inspect the file.

The glTF format is compatible with both AR.js and WebXR. Therefore, it fulfills all given criteria.

3.4 IFC to glTF transformation

The IFC.js library and the BlenderBIM add-on were tested and evaluated in the prototype. IFC.js does not actually transform IFC data into glTF, but directly transforms the data into so-called fragments, which are readable by Three.js.[14] As IFC.js was tested before the decision to use glTF was made, it's still discussed in this section.

3.4.1 Acceptance criteria

- Transformation quality
- Flexibility in configuration
- Actively maintained

3.4.2 IFC.js

IFC.js, or more specific, IFC.js Components is a library which includes various BIM tools based on Three.js such as loading IFC files as 3D models or reading and writing IFC files. While testing this library in the prototype there were various issues with the IFC files available to the team. Some files could not be loaded at all because they were too big, other files could be loaded, but the visualization was erroneous, or no lines were visible at all. Only one file seemed to be displayed correctly, but the browser console still showed an error. IFC.js Components is very flexible in configuration but rather complex to understand without much experience in the field. The library seems to be well maintained, but still having a lot of bugs.[15]

3.4.3 Blender with BlenderBIM

BlenderBIM is an add-on for Blender which allows converting IFC files to glTF. This add-on performed significantly better than IFC.js and loaded the IFC models correctly. While this was tested manually by using Blender and loading and exporting the IFC files, this could be automated using the Blender Python API. As the IFC files are converted into glTF directly within Blender, it's also possible to manipulate the glTF model within Blender. Meaning that parts of the model which aren't needed could easily be deleted, for example. Hence, this approach allows much flexibility in configuration. BlenderBIM is part of the IfcOpenShell project and is actively maintained.[16]

3.4.4 Conclusion

Since IFC.js performed poorly during testing and is still in pre-alpha status, BlenderBIM will be used for the final web application. On top of that, Blender with BlenderBIM is logically the only correct decision as out of Blender and IFC.js, only BlenderBIM supports glTF output.

3.5 glTF data storage

The backend should deliver the glTF data to the frontend, requiring a suitable data storage for it. As a precondition, the database management system is already set to PostgreSQL. PostgreSQL is the only well known database management system to all project team members. That's why it's the only option without loosing much time on learning a new database management system.

The glTF files usually contains utility line data from a whole or a large part of a municipality. There are two possible approaches to realize this data storage where each basically handles the glTF files differently.

3.5.1 Acceptance criteria

- Time needed for implementation
- Complexity

3.5.2 Approach 1: Split glTF files in smaller pieces

The first but more complex approach is to split a glTF model into smaller objects, where each object contains only one pipe, for example. Each object must have an absolute coordinate property, as each tube, drain, etc. would be delivered as a single entity to the frontend. This means the distance from each smaller object to the reference point of the parent object must be calculated. Hence, it would be extremely complex and time-consuming to write a tool which does this.

Still, the advantages would be that this approach enables spatial queries on the database which allows the frontend to only request data it really needs. Like tubes within 20 meters, for example. Furthermore, the performance of the frontend would probably be better than in the second approach because the frontend has fewer data to compute.

Whereas the disadvantages are that it's really challenging to create such a tool and would consume a lot of time. Additionally, it would be sort of a blind flight for the team as everything would have to be self-engineered.

3.5.3 Approach 2: Save whole glTF file as one record

As a second approach, the glTF files could be stored as GLB and stored within a PostgreSQL binary field. To be able to search for a specific glTF file it would be mandatory to have a second column where the name of the municipality or the area is saved as a string. The respective glTF file could then be requested from the frontend using a dropdown with all the available areas and municipalities.

This approach would be easier to implement and less time-consuming compared to the first approach.

The disadvantages are that the frontend will probably have a worse performance than in the first approach, as it usually has to compute more data than it needs.

3.5.4 Conclusion

The first approach goes beyond the scope of this project. Furthermore, the first approach would also add additional complexity to the frontend and to the API. In conclusion, the second approach better suits this project, because time is limited. Also, the focus of this project lies on the web application rather than on the storage of the data.

3.6 Prototype

The prototype was built in order to further explore and test certain technologies that are less familiar to the team. At some parts of the planned application there were multiple candidate technologies. For example, it had to be evaluated if WebXR or AR.js should be used for the realization of the AR environment. Furthermore, the prototype was utilized to set up Docker images, Docker compositions and the GitLab pipeline.

3.6.1 Contents

This section describes the different technologies which were tested in the prototype.

IFC.js

As described in the IFC.js evaluation, it can be used to directly load IFC data as 3D models on the web.

IFC to glTF conversion

Because IFC.js is in pre-alpha state it's likely that it won't perform well. As an alternative, the IFC files would first be converted to an optimized 3D format in the backend and then served to the frontend. More specifically, it would be converted to glTF, which is the recommended format when using Three.js.[17]

AR Visualization

For visualizing the utility lines in augmented reality, AR.js and WebXR are tested in the prototype. AR.js comes with integrated location-based augmented reality, which aligns well with our project, given the final application relies on the exact location. The WebXR device API could also be used for the project. It's designed to provide a robust set of AR/VR features and good integration with device hardware. Being a standard API, it's directly integrated into the browser, ensuring better performance and future-proofing. However, using WebXR requires manually transforming the GNSS geolocation data into local coordinates inside the AR space, because location-based based AR is not supported.

Database

It's planned to use PostgreSQL with the PostGIS extension to be able to store coordinates easily. Furthermore, it must be checked what's the best way of storing glTF data.

Docker Images

To get the application running multiple Docker images are needed. Two images were planned. One image for the application and a second image for the PostgreSQL database.

GitLab Repository and CI/CD pipeline

The prototype uses a single repository. This repository includes tests for both the frontend and backend.

The CI/CD pipeline of the prototype repository consists of three stages:

1. Build the frontend
2. Build the application docker image
3. Deploy the application image to the project server

Deployment

The prototype also includes setting up a CI/CD pipeline to automatically deploy the application onto the project server. This process should be completed within the GitLab pipeline.

3.6.2 Findings

This section serves as a summary of the technical choices and justifications based on prototype testing.

IFC.js

While testing this library in the prototype there were various issues observed with the available IFC files. Some files could not be loaded at all because they were too big, other files could be loaded, but the visualization was erroneous, or no lines were visible at all. Only one file seemed to be displayed correctly, but the browser console still showed an error.

IFC to glTF conversion

Regarding the conversion of IFC to glTF, there are two solution that were taken into consideration: IfcOpenShell and BlenderBIM. At the time of the prototype creation, it was required to convert the IFC files to glTF (not GLB), because of this it was decided to use BlenderBIM for the conversion, as IfcOpenShell only supports converting to GLB and not glTF.
BlenderBIM is an add-on for blender, as mentioned it allows the conversion of IFC to glTF/GLB, additionally it serves as an IFC viewer in Blender. Using BlenderBIM performed significantly better than using IFC.js and converted almost all IFC models correctly. While this was tested manually by using Blender and loading and exporting the IFC files, this could be automated using the Blender Python API. Given these findings, this approach will be used instead of IFC.js.

AR Visualization

While testing AR.js and WebXR in the prototype, AR.js had some issues such as jittery 3D models and incorrect position/rotation of the 3D models based on their geolocation. AR.js also has an incomplete documentation and a limited API, which would make it more difficult for future development when more complicated features should be implemented.
The performance of WebXR was much better and the AR experience ran smoother. Just like AR.js, the positioning of the 3D models showed some inaccuracies. However, since the positioning logic for WebXR is manually implemented by the team, it can be easily modified and improved

in the actual application. For WebXR to work, it's required that the connection uses HTTPS. Consequently, it must be ensured that the hosting server has valid SSL/TLS certificates installed.

Database

As planned, PostgreSQL with the PostGIS extension is used as the database management system. PostGIS is necessary to be able to efficiently store coordinates of 3D utility line models within one column. The 3D data is stored in a column of type binary as the glTF data is stored in its binary format GLB. SQLAlchemy is used to enable communication between the Flask web server and the database. Alembic was also integrated and tested for database migrations. Although, migrations might be unnecessary in the initial phase of application development it will probably pay off if the application is further developed after this term project.

> **What is SQLAlchemy and Alembic?**
>
> SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.[18]
>
> Alembic is a database migrations tool written by the author of SQLAlchemy.[19] It was built to be used with SQLAlchemy.

Docker Images

It turned out that three images are needed to be able to run the prototype:

1. **Application**: Consists of a Flask/Gunicorn web server serving the Vue application and handling API calls.
2. **nginx**: Uses the latest stable docker image from docker hub. Acts as a reverse proxy in front of the web server as recommended by the Flask creators.[20]
3. **PostgreSQL**: Uses the PostGIS 16-3.4 docker image from docker hub. This image offers PostgreSQL with the PostGIS extension preinstalled.

> **What is Gunicorn?**
>
> Flask is a WSGI (Web Server Gateway Interface) application. A WSGI server is used to run the application, converting incoming HTTP requests to the standard WSGI environ, and converting outgoing WSGI responses to HTTP responses.[20] Gunicorn is a commonly used WSGI server and was used for the prototype application.

GitLab Repositories and CI/CD pipeline

Initially it was planned to use two repositories for the actual application, where one repository is used for the backend and the other one for the frontend. While building the prototype it became clear that no team member has enough experience to set up GitLab pipelines and creating Docker images with multiple repositories in a reasonable amount of time. Hence, a monolithic repository is used for the actual application.

The actual configured prototype pipeline looks the following:

1. Build the frontend (Vue.js application) and store the built application in artifacts.
2. Build the application Docker image and push it to GitLab's container registry.
3. Deploy the application by composing three Docker containers (application, Nginx and PostgreSQL). The complete deployment process can be found below.

For the actual application the pipeline has one additional stage for testing the frontend.

Deployment

To automatically deploy the application to the project server within the pipeline an additional Gitlab runner is necessary. This runner runs directly in the shell of the project server as a service by a dedicated user. This user is only allowed to run the GitLab runner process and Docker commands. Additionally, the user is restricted in file system access.

Upon approaching of the deployment job in the CI/CD pipeline the GitLab runner process clones the repository and then runs Docker compose on the production docker compose YAML.

Implementation concept

This project is organized on a modular basis. Each part plays a key role in fulfilling the project task. With this implementation concept, a rough overview over the project management is given.

4.1 Knowledge gathering

Initially, information has to be gathered about what tools are available to build a AR web application. Besides the AR tooling, it has to be evaluated how IFC data could be converted into a 3D model. For this project the team got a tablet and an additional RTK receiver device, which also has to be set up and tested.

4.2 Requirements specification

To further evaluate which tools and solutions should be used, requirements must be specified. Requirements are divided into non-functional and functional requirements. Further description about these requirements can be found in the requirements specification chapter.

These requirements serve as guidelines when building the architecture and even the application itself. Additionally, during evaluation and prototyping it's possible to granularly choose the tools, which offer the flexibility to realize the specified requirements.

4.3 Evaluation & prototype creation

As described in previous chapters there are multiple possibilities of tools or libraries in each part of the application. Therefore, the best tools and libraries available have to be evaluated. The best evaluation technique is building a prototype where everything can be tested in practice.

4.4 Architecture

Before implementing the application, architectural decisions have to be made. Along with chosen libraries and tools, the initial architecture builds the fundament for the application. An architecture must be designed which is neither too complex nor inextensible in case of further developments.

Luckily, a rough sketch of the architecture is derivable from the prototype. Which means the team must not only rely on theoretical information but also practical knowledge from the prototype.

4.5 Implementation

On the basis of the evaluations and the designed architecture, the application can be implemented. The implemented application builds the core part of this project.

4.6 Conclusion

Approaching the end of the project time, an overall conclusion will be taken. This part compares expectations with effective experience and discusses further development ideas.

Results

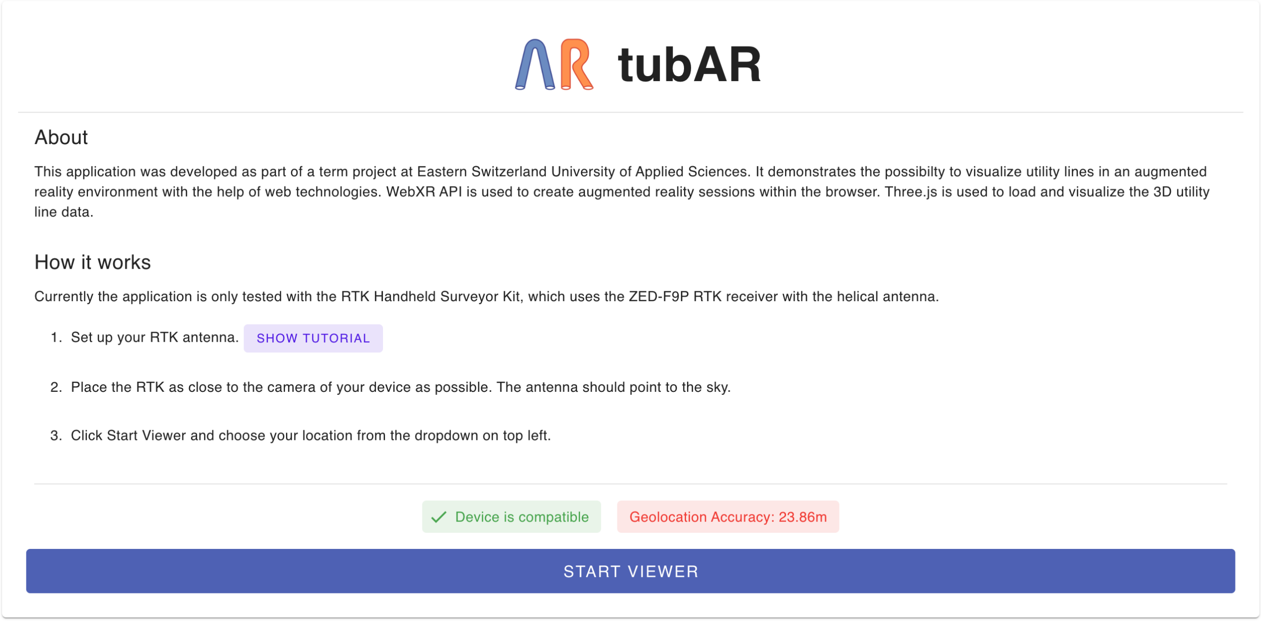
This chapter outlines the project's accomplishments and compares them with those of a competitor. It also provides a brief overview of potential additional functionalities, which will be approached during the bachelor thesis.

5.1 Goal achievement

The project's success was measured by comparing the actual application against its functional requirements. Below is an analysis of the implemented user stories, followed by the remaining unimplemented features. Details on these user stories can be found in functional requirements.

5.1.1 US-1: Start screen

The starting screen has been fully implemented and contains essential information about the application and its usage.



About

This application was developed as part of a term project at Eastern Switzerland University of Applied Sciences. It demonstrates the possibilty to visualize utility lines in an augmented reality environment with the help of web technologies. WebXR API is used to create augmented reality sessions within the browser. Three.js is used to load and visualize the 3D utility line data.

How it works

Currently the application is only tested with the RTK Handheld Surveyor Kit, which uses the ZED-F9P RTK receiver with the helical antenna.

1. Set up your RTK antenna. SHOW TUTORIAL
2. Place the RTK as close to the camera of your device as possible. The antenna should point to the sky.
3. Click Start Viewer and choose your location from the dropdown on top left.

✓ Device is compatible

Geolocation Accuracy: 23.86m

Figure 5.1: tubAR start screen

5.1.2 US-4: Accurate depth display

Utility lines are correctly placed below ground, accurately representing their depth. Future improvements include adding transparency to the utility lines for a better depth effect.

5.1.3 US-5: Accurate positioning of utility lines

The positioning of utility lines is somewhat accurate. Challenges with geo-alignment and compass accuracy, as discussed in the challenges chapter, are the primary causes. This topic will be further evaluated in the continuation of this project.

5.1.4 US-6: Rendering lines as 3D objects

Utilizing Three.js and WebXR, the utility lines are rendered as 3D objects in the AR environment and behave accordingly to movement of the device.

5.1.5 US-11: Display status of GNSS accuracy

A GNSS accuracy status can be seen on the starting screen as well as in the AR session. Though, while inside the AR session, the location accuracy badge may not be clearly visible at times due to its transparency. This will also be improved in the project continuation.

5.1.6 US-12: Location selection

While in the AR session, a dropdown menu with all locations is placed in the top left corner. After selecting the desired location, it gets loaded from the backend and displayed in the AR environment.

5.1.7 Unimplemented features

While all essential ("Must have") user stories were implemented, US-5 requires further development. Due to time constraints and lack of experience with augmented reality by the team, the following user stories were not completed in this term thesis:

- **Should have**
  - US-2: Onboard guidance
  - US-10: Calibration guidance
- **Could have**
  - US-3: User login
  - US-7: Interactive controls
  - US-8: Filter utility line types
  - US-9: Info interaction

5.2 Comparison to vGIS

This section compares the developed application, tubAR, with its primary competitor, vGIS. The comparison highlights key aspects and differences of each solution.

	tubAR	**vGIS**
App type	Web application	Native application, supported platforms: Android, iOS, HoloLens 2. [21]
Technologies	WebXR, Vue.js, Three.js, Flask, PostgreSQL, PostGIS, BlenderBIM	Undisclosed
Functionality	Augmented reality visualization of IFC data.	AR visualization of IFC (and many other formats), real-time interaction with models, data modification and creation, distance measurement. [21]
Precision	Can't be precisely determined due to alignment issues, can range from a few centimeters to a few meters depending on various factors. The antenna used in this project is up to one centimeter accurate.[22]	Horizontal: 1cm, Vertical: 2cm, Directional: +/-0.1°. [21]
Price	Antenna: €399	Annual license fee: $1250

Table 5.1: tubAR and vGIS comparison

In summary, the comparison between tubAR and vGIS highlights tubAR's affordability and web-based functionality, but also it's shortcomings when it gets to more advanced features and precision.

5.3 Additional features

Besides the unimplemented features, the team has come up with the following additional features to be implemented in the bachelor thesis:

- Only display utility lines that are inside a certain radius of the device
- Dynamically load models based on the user's geolocation
- Improved conversion of IFC to glTF
- Optionally: Allow users to upload and visualize IFC data directly on the web-app

Details about these potential features are elaborated in chapter 12.

Part II

Project Documentation

Vision

The vision can be found here.

Requirements specification

This chapter outlines the requirements towards the term thesis, use cases, as well as functional and non-functional requirements of the software solution, serving as a guide for development.

7.1 Use cases

Use cases describe how a system interacts with its environment in order to achieve a specific goal. For the system of this project, there's a user which interacts with tubAR web application.

7.1.1 Use case diagram

The following diagram shows the main use cases of the system.

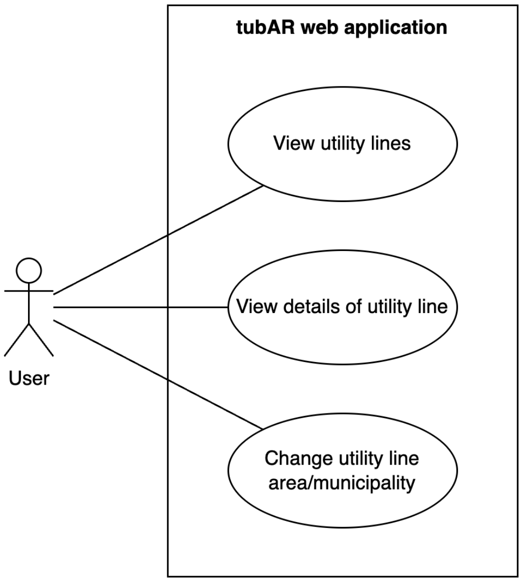

tubAR web application
View utility lines
View details of utility line
Change utility line area/municipality
User

Figure 7.1: Use case diagram

View utility lines

Visualization of the utility lines in an AR world. The utility lines should be displayed under the ground and correctly positioned based on their coordinates.

View details of utility line

Displaying more information about one specific utility line, such as their type.

Change utility line area/municipality

Changing the utility lines to different areas or municipalities based on the user's selection.

7.2 Functional Requirements

The functional requirements define the specific features and functionality that the software solution must provide.

7.2.1 Actors

The final product is meant to be used by construction workers, landscape gardeners, telecommunication professionals, etc. As the intended usage of the app by all these users is the same, there will be only one actor referenced in the following user stories: the user. The goal of the user is to be able to visualize utility lines directly in their environment through their tablet.

7.2.2 User Stories

Actions that a user wants to perform in the web application are defined by user stories. They are marked with US-X (X corresponding to the number of the story).

All user stories will be given a priority of development, as well as rough estimation of development effort needed. The Fibonacci scale is a common measure in the scrum process and reflects the relative effort of a task in the project. Additionally, the MoSCoW method will be used to prioritize the user stories.

Entry point

ID	US-1
Subject	Start screen
Priority	Must have
Time estimation	3
Story points	• When the web application is loaded the user wants to have an overview about the web-application. Which included what it does and how it works. • The user wants to know if their device is compatible with the web application. • The user wants to know if their location is accurate enough.

Table 7.1: Start screen (US-1)

ID	US-2
Subject	Onboard guidance
Priority	Should have
Time estimation	13
Story points	• To get started with the augmented reality experience, the user wants to know what they can do with the web application. • The user wants to know how each feature can be used.

Table 7.2: Onboard guidance (US-2)

ID	US-3
Subject	User login
Priority	Could have
Time estimation	8
Story points	• The user has to authenticate themself to gain access to the web application and its data.

Table 7.3: User login (US-3)

Viewer

ID	US-4
Subject	Accurate depth display
Priority	Must have
Time estimation	5
Story points	• The user wants an appropriate depth display of the utility lines. Such that the displayed utility lines look like they are below the ground. • The user wants the utility lines to be correctly displayed respectively to their height.

Table 7.4: Accurate depth display (US-4)

ID	US-5
Subject	Accurate positioning of utility lines
Priority	Must have
Time estimation	5
Story points	• The user wants the utility lines to be accurately positioned within their AR experience. • The user wants to be able to comprehend the position of the utility lines.

Table 7.5: Accurate positioning of utility lines (US-5)

ID	US-6
Subject	Rendering utility lines as 3D objects
Priority	Must have
Time estimation	13
Story points	• Inside the viewer, the user wants to see the utility lines as 3D objects.

Table 7.6: Rendering utility lines as 3D objects (US-6)

Overlay

ID	US-7
Subject	Interactive controls
Priority	Could have
Time estimation	5
Story points	• The user wants to be able to zoom.

Table 7.7: Interactive controls (US-7)

ID	US-8
Subject	Filter utility line types
Priority	Could have
Time estimation	5
Story points	• The user wants to be able to filter by line types directly inside the viewer. • The user only wants to see the utility lines they filtered for.

Table 7.8: Filter utility line types (US-8)

ID	US-9
Subject	Info interaction
Priority	Could have
Time estimation	8
Story points	• The user wants to be able to access additional information of the utility lines by tapping on them.

Table 7.9: Info interaction (US-9)

ID	US-10
Subject	Calibration guidance
Priority	Should have
Time estimation	5
Story points	• If the web application needs recalibration or initial calibration to display utility lines correctly the user wants to be informed how they should proceed. • The user wants to know how their GNSS accuracy can be increased.

Table 7.10: Calibration guidance (US-10)

ID	US-11
Subject	Display status of GNSS accuracy
Priority	Should have
Time estimation	3
Story points	• The user wants to stay informed about their GNSS accuracy.

Table 7.11: Display status of GNSS accuracy (US-11)

ID	US-12
Subject	Location selection
Priority	Must have
Time estimation	3
Story points	• The user wants to be able to select the location for which the utility lines should be displayed. • The user wants to see a visual indication as soon as the utility lines are successfully loaded and displayed.

Table 7.12: Location selection (US-12)

7.3 Non-Functional Requirements

The non-functional requirements outline the characteristics and constraints that the software solution must satisfy, such as performance, security and scalability.

7.3.1 Categories

The eight categories defined in ISO/IEC 25010 are used to categorize each non-functional requirement.

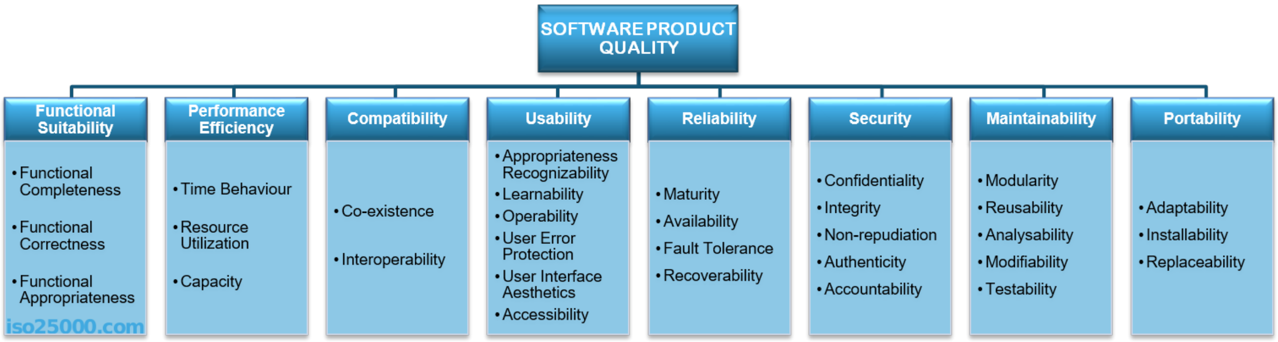


Figure 7.2: ISO/IEC 25010 quality characteristics
Source: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

7.3.2 Fulfillment-Check Procedure

Each NFR has a “Fulfillment Check” row which describes who has the responsibility to check if the NFR is fulfilled and how this is checked, if it is not already mentioned in “Measures”.

7.3.3 Performance and Efficiency

ID	NFR-1
Subject	3D data loading
Requirement	Time behavior
Priority	Medium
Measures	• Loading and displaying the 3D data should not exceed 3 seconds.
Fulfillment check	The architect is responsible to check if this NFR is fulfilled.

Table 7.13: 3D data loading (NFR-1)

ID	NFR-2
Subject	Frames per second
Requirement	Resource utilization
Priority	High
Measures	• During runtime of the viewer at least 30 fps should be achieved at average. • Starting with the alpha version of the application this will be checked for every major version (beta, final). • This can be displayed with Three.js’ developer tools.
Fulfillment check	The architect is responsible to check if this NFR is fulfilled.

Table 7.14: Frames per second (NFR-2)

ID	NFR-3
Subject	RAM usage
Requirement	Resource Utilization
Priority	Low
Measures	• During runtime of the application not more than 2 GB of RAM is used. • Starting with the alpha version of the application this will be checked for every major version (beta, final). • This can be checked in the Android settings.
Fulfillment check	The architect is responsible to check if this NFR is fulfilled.

Table 7.15: RAM usage (NFR-3)

7.3.4 Compatibility

ID	NFR-4
Subject	Supported Browser
Requirement	Compatibility
Priority	High
Measures	• Only Chrome will be supported as browser starting from its major version 117.
Fulfillment check	The architect is responsible to check if this NFR is fulfilled.

Table 7.16: Supported Browser (NFR-4)

7.3.5 Usability

ID	NFR-5
Subject	Vuetify usage
Requirement	User Interface Aesthetics
Priority	Low
Measures	• To guarantee an easy-to-use and consistent UI, Vuetify will be used.
Fulfillment check	To reach Done state for Tasks/Issues each responsible developer has to check if this NFR is fulfilled.

Table 7.17: Vuetify usage (NFR-5)

7.3.6 Security

ID	NFR-6
Subject	Don't allow unauthorized access to data
Requirement	Confidentiality / Authenticity
Priority	High (If user authentication is realized, this NFR must be fulfilled.)
Measures	• An OAuth service will be used to achieve this.
Fulfillment check	The architect is responsible to check if this NFR is fulfilled.

Table 7.18: Don't allow unauthorized access to data (NFR-6)

7.3.7 Maintainability

ID	NFR-7
Subject	Usage of Vue.js
Requirement	Reusability / Modifiability
Priority	High
Measures	• To achieve reusability of components, Vue.js is used.
Fulfillment check	To reach Done state for tasks / issues each responsible developer has to check if this NFR is fulfilled.

Table 7.19: Usage of Vue.js (NFR-7)

ID	NFR-8
Subject	Unit tests
Requirement	Testability
Priority	High
Measures	• Test coverage must be 80% at a minimum for TypeScript and Python files. • This will be checked via CI/CD pipeline in GitLab. • For each implemented feature unit tests must be provided by the responsible developer.
Fulfillment check	To reach Done state for tasks / issues each responsible developer has to check if this NFR is fulfilled.

Table 7.20: Unit Tests (NFR-8)

ID	NFR-9
Subject	Logging backend
Requirement	Analyzability
Priority	Low
Measures	• All errors and warnings of the backend are logged. • Flask includes a logger.
Fulfillment check	To reach Done state for tasks / issues each responsible developer has to check if this NFR is fulfilled.

Table 7.21: Logging backend (NFR-9)

7.3.8 Portability

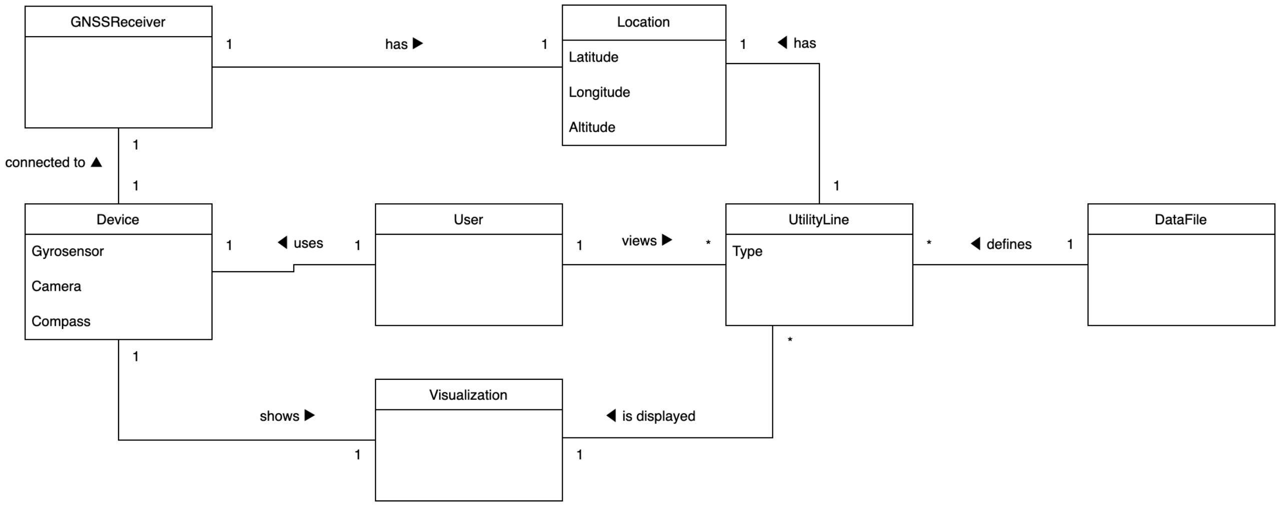
ID	NFR-10
Subject	Dockerization of web application
Requirement	Installability
Priority	High
Measures	• The application will be deployed using a Docker image to ensure device independence.
Fulfillment check	To reach Done state for tasks / issues each responsible developer has to check if this NFR is fulfilled.

Table 7.22: Dockerization of web application (NFR-10)

Analysis

The “Analysis” chapter provides an overview of the domain model, including detailed descriptions of the elements. Furthermore, it contains a summary of the database model.

8.1 Domain Analysis



8.2 Object catalog

The object catalog provides a detailed description of the various elements in the domain model and their relationships.

8.2.1 Device

The device, typically a tablet, runs the web application and renders the augmented reality visualization of the *UtilityLines*. It interfaces with the *GNSSReceiver* and other built-in sensors for accurate real-time positioning.

8.2.2 Location

Location is defined by WGS 84 coordinates. It is used to accurately position the *UtilityLines* within the application’s virtual space, ensuring that the AR visualization aligns correctly with the real-world coordinate system.

8.2.3 DataFile

The *DataFile* is in IFC format and contains detailed information about the *UtilityLines*, including their types, dimensions and materials. It serves as the blueprint for generating the 3D representations of the *UtilityLines*.

8.2.4 UtilityLine

Each *UtilityLine* is a 3D representation of a physical utility line, defined by the *DataFile*. The location attribute indicates the reference position for the *UtilityLine*.

8.2.5 GNSSReceiver

The *GNSSReceiver*, connected to the Device, provides high-precision location data. This real-time positioning is crucial for aligning the 3D representations of *UtilityLines* with their physical counterparts in the real world. The *GNSSReceiver* ensures that the AR visualization is accurate and responsive to the movements of the Device.

8.2.6 User

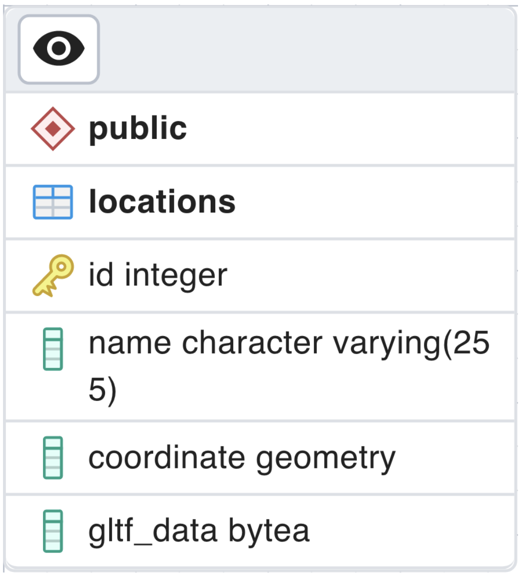
The *User* interacts with the application through the Device. They can view and analyze the *UtilityLines* in AR.

8.2.7 Visualization

Visualization refers to the graphical representation of *UtilityLines* within the augmented reality environment of the application. It involves the process of correctly displaying the *UtilityLines* in an optimized 3D format.

8.3 Database model

As described in evaluation of the glTF data storage approach 2 the database consists only of one table. This is adequate for the scope of this project and its resulting application.



public

locations

id integer

name character varying(255)

coordinate geometry

gltf_data bytea

Figure 8.1: Database model

Design

This chapter details the architecture of the application providing an overview of the various parts such as frontend and backend. It also includes sequence diagrams to visually represent the key processes of the project.

9.1 Architecture

This section outlines the architectural framework and structural elements of the project. It includes a detailed explanation of the system's architecture using C4 diagrams and an analysis of the source code structure for both the frontend and backend components.

9.1.1 C4 diagrams

C4 modeling leads to a model which is easy to understand, but still provides all important information about an architecture. Not all existing C4 diagrams are used to describe this application. The system context diagram and the container diagram are sufficient to describe tubAR. More about C4 modeling can be read on c4model.com.

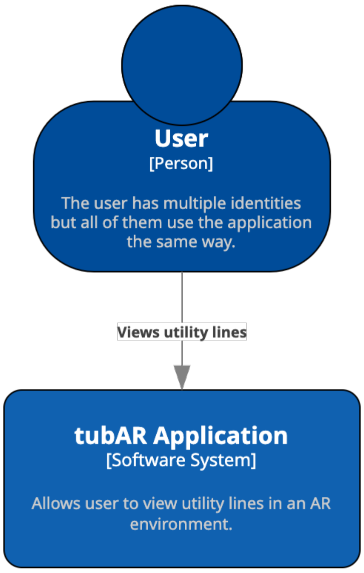

User
[Person]
The user has multiple identities but all of them use the application the same way.
Views utility lines
tubAR Application
[Software System]
Allows user to view utility lines in an AR environment.

Figure 9.1: System context diagram

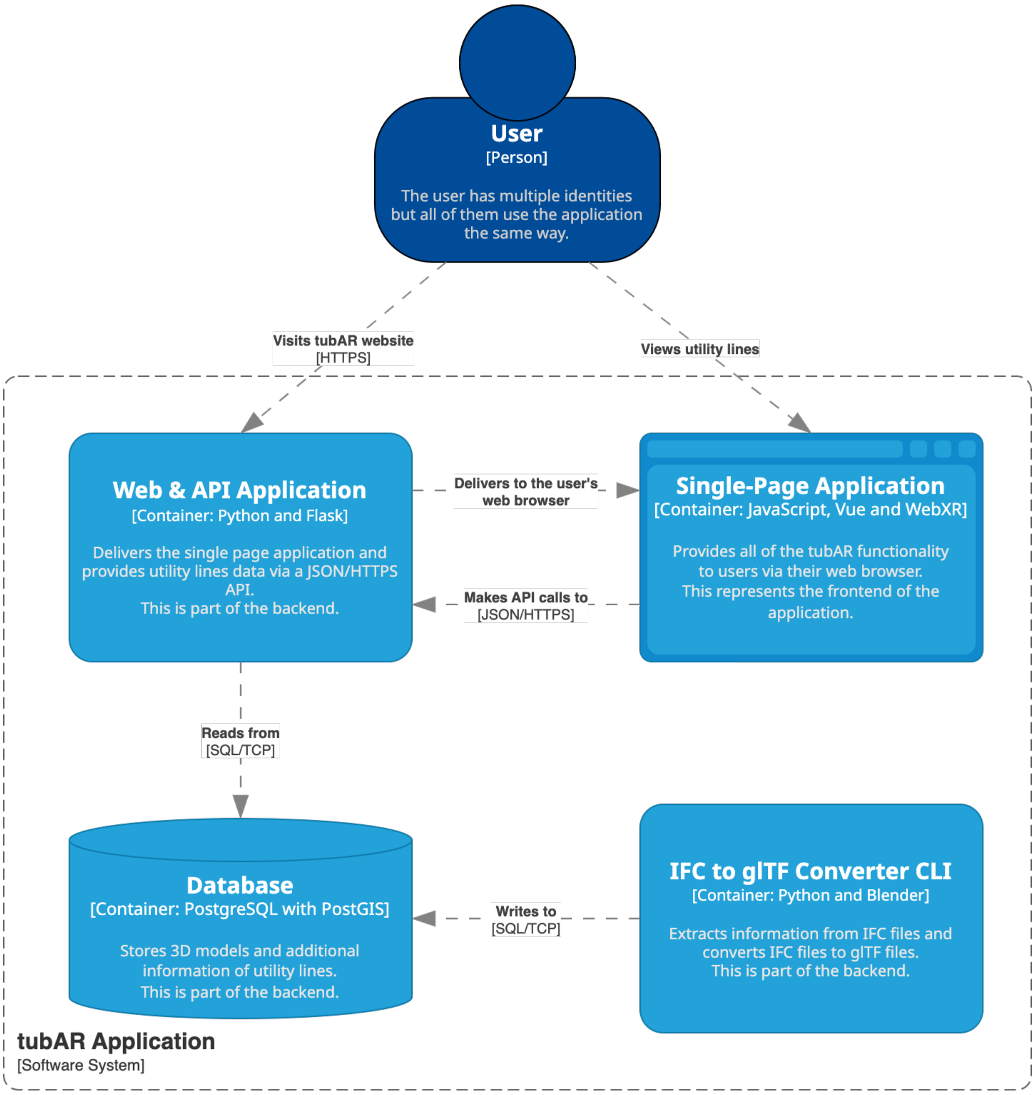

User
[Person]
The user has multiple identities but all of them use the application the same way.
Visits tubAR website
[HTTPS]
Views utility lines
Web & API Application
[Container: Python and Flask]
Delivers the single page application and provides utility lines data via a JSON/HTTPS API.
This is part of the backend.
Delivers to the user's web browser
Makes API calls to
[JSON/HTTPS]
Single-Page Application
[Container: JavaScript, Vue and WebXR]
Provides all of the tubAR functionality to users via their web browser.
This represents the frontend of the application.
Reads from
[SQL/TCP]
Database
[Container: PostgreSQL with PostGIS]
Stores 3D models and additional information of utility lines.
This is part of the backend.
Writes to
[SQL/TCP]
IFC to glTF Converter CLI
[Container: Python and Blender]
Extracts information from IFC files and converts IFC files to glTF files.
This is part of the backend.
tubAR Application
[Software System]

Figure 9.2: Container diagram

9.1.2 Source code structure

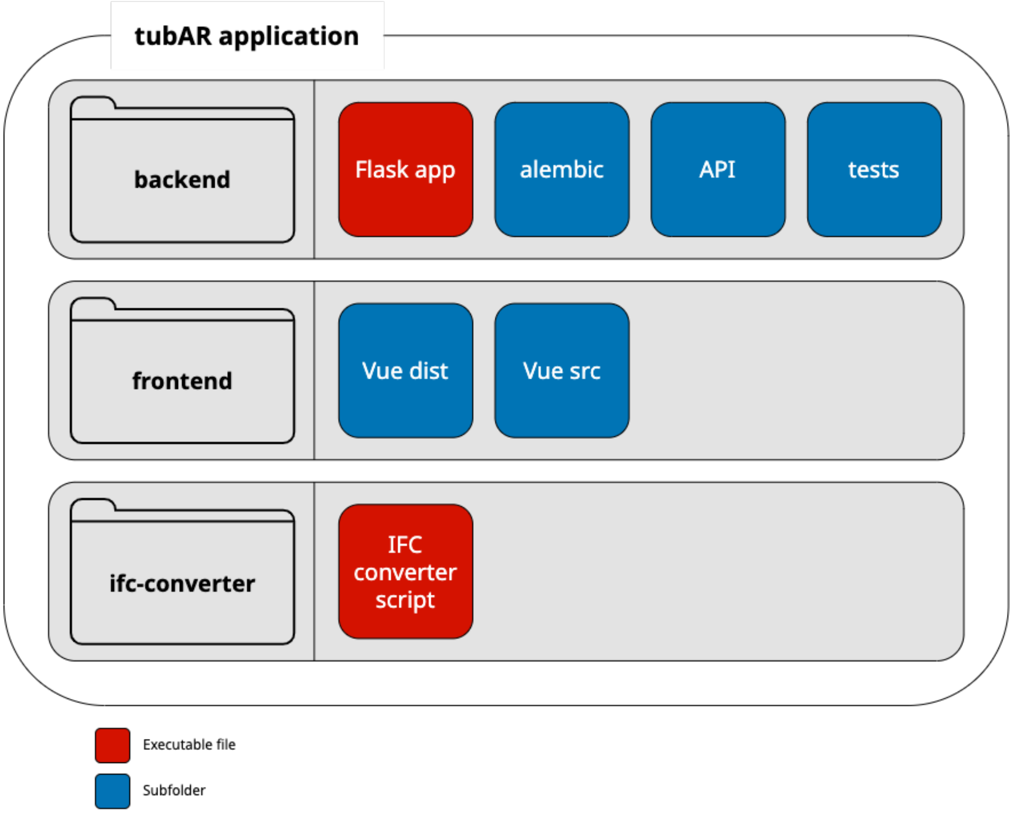


Figure 9.3: Source code structure

- backend:
  - **Flask app**: Holds the executable Flask application.
  - **alembic**: Contains the code for database migrations. If any table must be altered or a new table must be added this can be done here.
  - **API**: Consists of the API models and the routes. The Flask app loads this content on startup.
  - **tests**: Includes Pytest tests for the backend.
- frontend:
  - **Vue dist**: Contains the built frontend application, which consists only of TypeScript, HTML and CSS. This is statically served from the Flask app.
  - **Vue src**: The Vue.js source code lies in this folder.
- ifc-converter:
  - **IFC converter script**: Holds the executable script.

9.1.3 Frontend

The frontend employs the Model-View-ViewModel (MVVM) pattern. Vue.js primarily implements the ViewModel, connecting JavaScript objects (Model) with the DOM (View) through two-way bindings. The Model and View are mainly defined by the developer.

Components hierarchy

Figure 9.4 illustrates the hierarchy of the Vue.js components within the application.

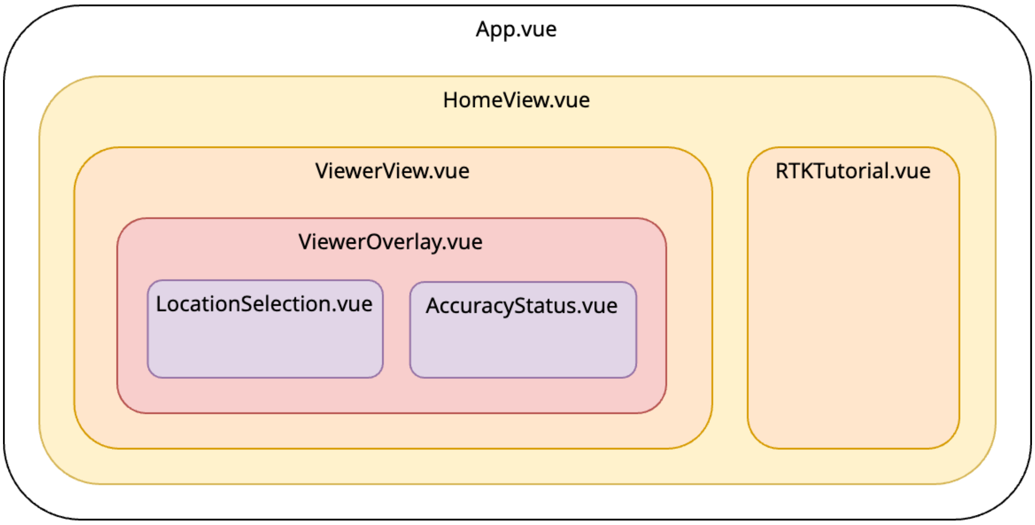


Figure 9.4: Components hierarchy diagram

- **App.vue**: Root component which defines the entry point for Vue.js.
- **HomeView.vue**: Contains the description of the application, a tutorial for the RTK setup (RTKTutorial.vue) and the ViewerView component.
- **ViewerView.vue**: Contains the overlay and a button for starting the WebXR session. Utilizes the UtilityLineViewer class (described in module diagram) for visualizing utility lines.
- **ViewerOverlay.vue**: Acts as the WebXR session overlay. Includes a location dropdown and the current geolocation accuracy status.

Module diagram

Besides the Vue components, additional TypeScript files have been implemented for handling parts of the application logic. Most relevant files and their relationship are illustrated in figure 9.5.

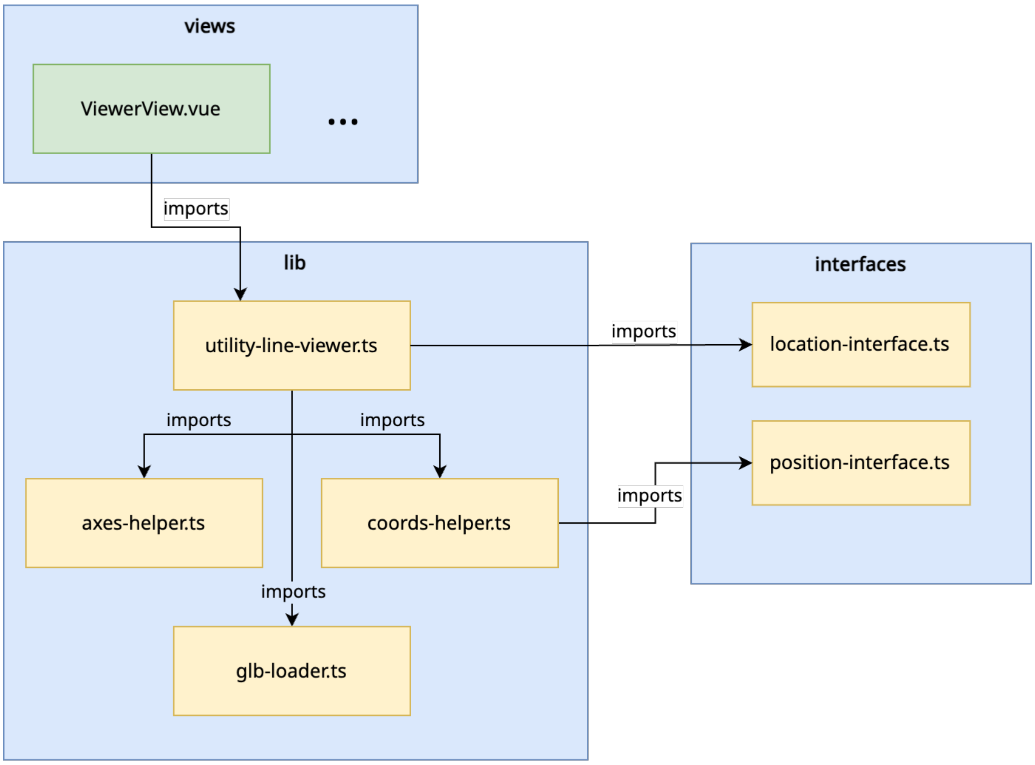


Figure 9.5: Frontend module diagram

- **utility-line-viewer.ts**: Manages AR visualization and positioning of utility lines in WebXR.
- **axes-helper.ts**: Generates colored arrows for each axis, mainly used for debugging purposes.
- **coords-helper.ts**: Calculates the relative distance between two WGS 84 coordinates.
- **glb-loader.ts**: Converts GLB strings to blobs, in order to load the model with Three.js.
- **location-interface.ts**: Defines the Location type with name, coordinate and gltf_data.
- **position-interface.ts**: Defines the Position type with latitude, longitude and altitude.

9.1.4 Backend

The backend is designed using the Model-View-Controller (MVC) pattern, though Flask itself doesn't specify a pattern. In the context of this REST API, the view is represented by JSON data. Besides MVC, the factory pattern is utilized for creating the application instance. This allows to have multiple instances with different configurations, which is very useful for testing.

Module diagram

The files of the Flask app and their relationship are illustrated in figure 9.6.

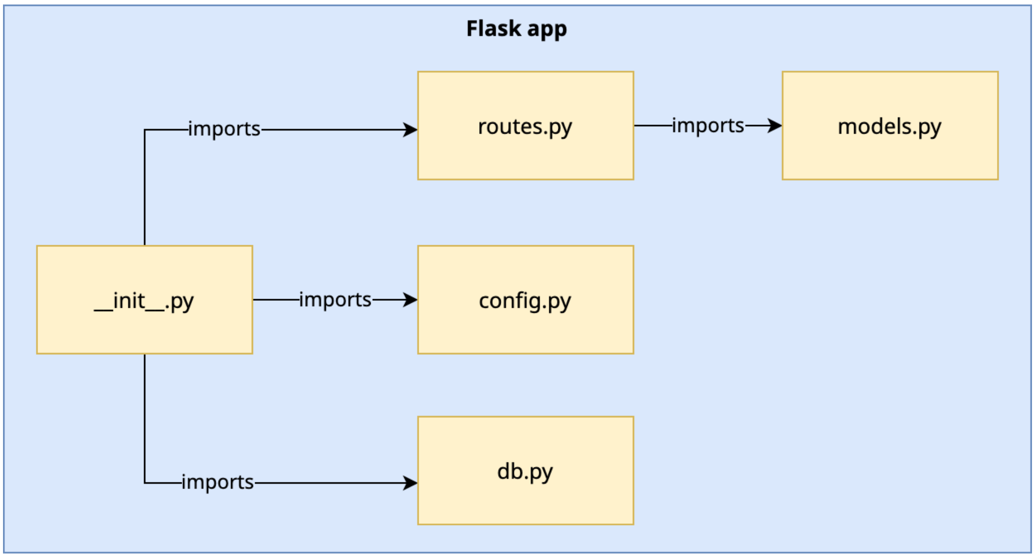


Figure 9.6: Backend module diagram

- **__init__.py**: Implements the factory method for creating an instance of the application.
- **routes.py**: Maps routes to functions which return JSON data. The fallback route returns the frontend.
- **models.py**: Defines the mapping of the database tables.
- **config.py**: Retrieves and sets the database URI from an environment variable.
- **db.py**: Creates an instance of the ORM-Framework SQLAlchemy.

9.2 Sequence Diagram

Sequence diagrams illustrate the interactions between different parts of a system in a specific order. The two main processes for the project are: Usage of the web application and conversion of IFC to GLB.

9.2.1 Web application

The complete process for loading and visualizing utility lines in the AR session is illustrated in the following diagram.

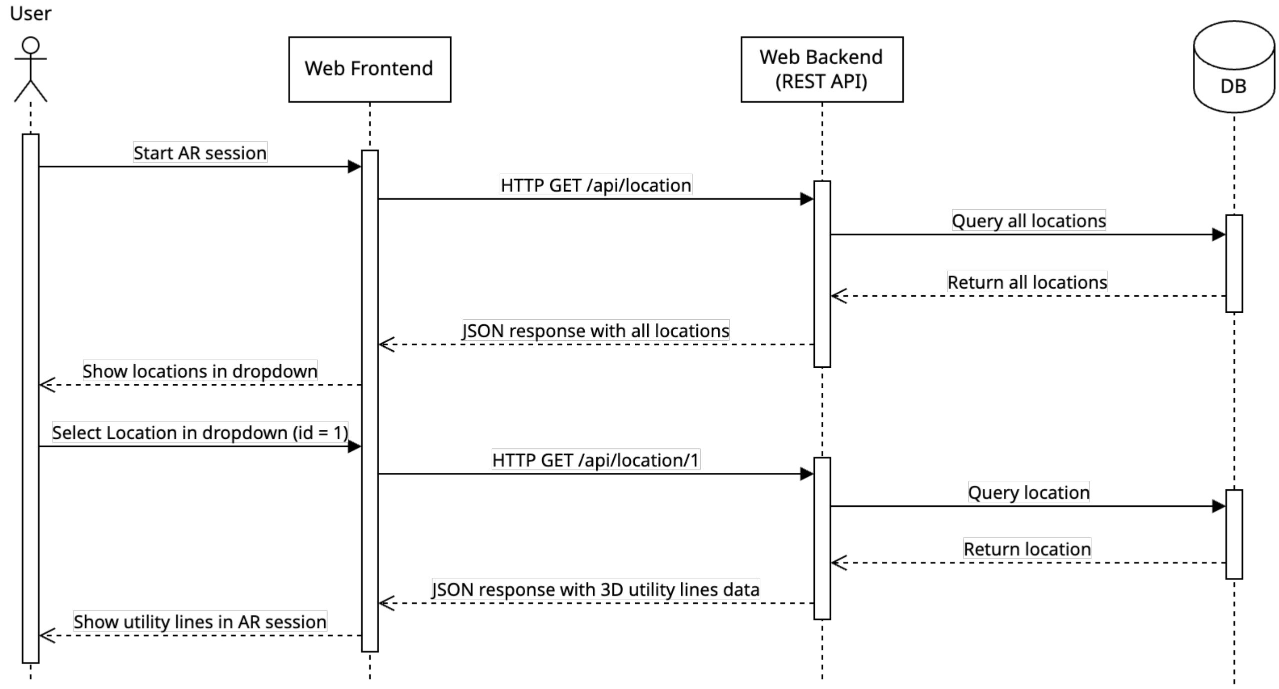


Figure 9.7: Web application sequence diagram

9.2.2 IFC converter script

The process of how the IFC converter script works, can be seen in the diagram below.

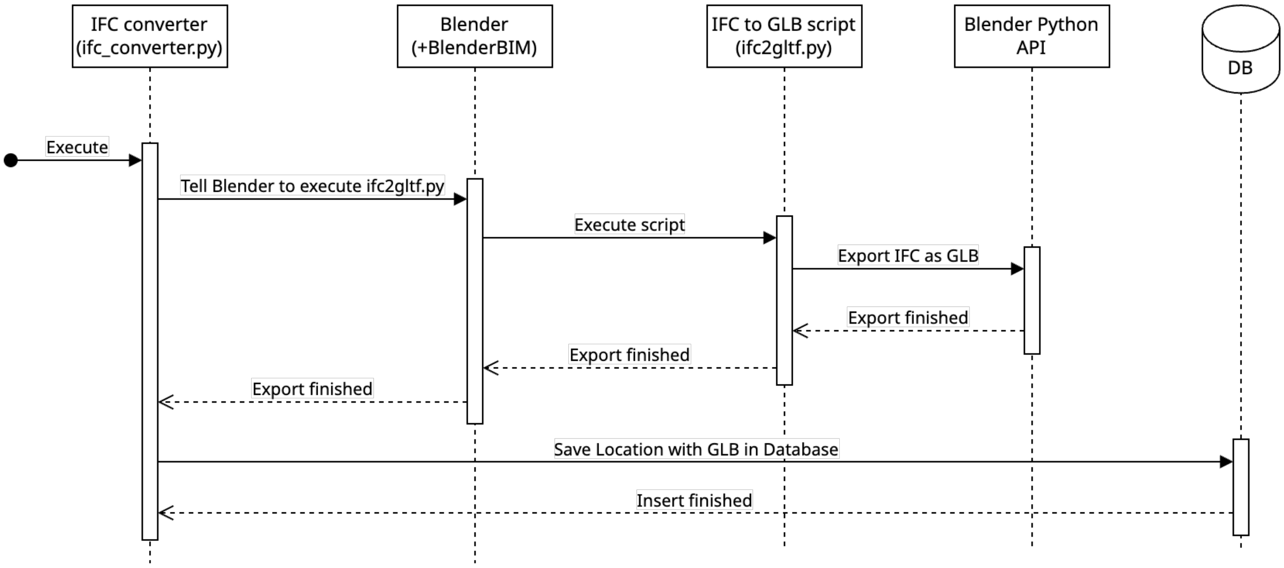


Figure 9.8: IFC converter script sequence diagram

The ifc_convert.py script can't access the Blender Python API directly because Blender has its own python environment which contains additional packages and different versions of packages. Hence, the conversion of IFC to GLB (which requires the Blender Python API) is implemented in a separate file that gets called by Blender with the corresponding environment.

Implementation & Testing

10.1 Implementation

This section usually contains some source code with according explanation. But it is not allowed to have source code in the documentation for the e-prints portal. Therefore, this section remains empty.

10.2 Automated & manual testing

Information about testing can be found in test strategy.

10.3 Challenges

This chapter summarizes noteworthy challenges the team encountered during this project.

10.3.1 Geo-alignment problem

The most difficult challenge is the so called geo-alignment problem. On initialization of a WebXR session, a virtual environment with its own coordinate system gets created. This coordinate system looks as follows; the negative Z-axis is pointing in the user's view direction and the positive X-axis to the right of the user. The user moves within this virtual environment. As tubAR aims to visualize objects of the real world at their exact position, it is necessary that both the real and virtual coordinate system are aligned. Otherwise, the objects are displayed at a wrong position within the virtual environment and don't match their real world position.

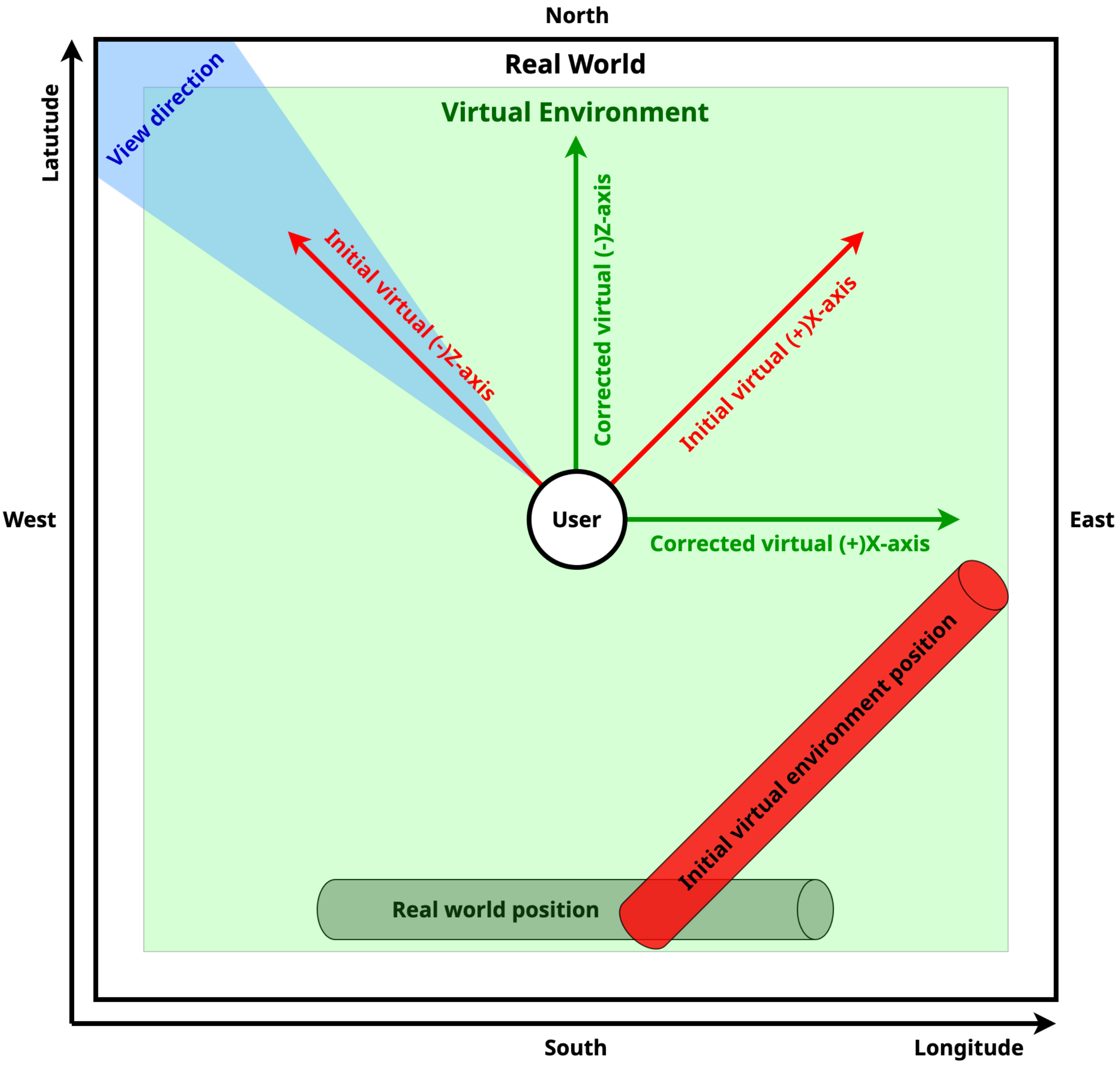

North
Real World
Virtual Environment
View direction
Latutude
Initial virtual (-)Z-axis
Corrected virtual (-)Z-axis
Initial virtual (+)X-axis
West
User
East
Corrected virtual (+)X-axis
Initial virtual environment position
Real world position
South
Longitude

Figure 10.1: Geo-alignment problem visualized

Figure 10.1 should give a clearer understanding of the problem. The real world coordinate system is the outer box, while the green inner box represents the virtual environment. There are two kinds of virtual environment axes shown, whereas the red ones show the axes set on initialization by WebXR. Furthermore, it is shown where the tube is placed in the real world and where it is placed according to the initial axes in the virtual environment. At this point it gets clear what the issue is. The negative Z-axis and the positive X-axis must point to the north and east accordingly, so that the utility line is in its correct position.

When looking at this graphic, solving this problem seems simple, because rotating the virtual environment by the current compass heading (which is 315 degree) in a counter-clockwise direction, would correspond to the corrected axes, which are green. In fact, it isn't simple as it is not possible to edit the so-called WebXR reference space because they are read-only. A convenient way would be if it was possible to tell WebXR that it should try to align its reference space to the real world coordinate system on initialization. This exact feature was requested multiple times by other developers but sadly not implemented in the WebXR API standard yet.[23][24]

Because of the described situation, another solution is needed. To get into further details, something else has to be explained first. The virtual environment actually consists of two parts. One is the WebXR reference space and the second one is the Three.js scene which lives inside the reference space. This Three.js scene is responsible for rendering all objects, whereas WebXR is basically only handling the movement of the user. Therefore, the approach was taken to try rotating the Three.js scene only, so that it is aligned with the real world coordinate system.

Throughout dummy testing, this lastly taken approach worked fine. During final testing in Stäfa it turned out, that this approach doesn't work properly when moving around and updating the utility lines' position because WebXR still handles the moving offsets incorrectly. This is the case because the reference space does not align to the real world coordinate system and forwards the movement incorrectly to Three.js. Unfortunately this was first noticed during final testing, which meant the team wasn't able to tackle this issue within scope of this term project. As a temporary workaround, the user can point the device to north when starting the WebXR session which would then initialize the reference space correctly aligned to the real world.

10.3.2 Compass problem

In the context of the geo-alignment problem outlined earlier, obtaining the device's current compass heading is required for aligning the virtual environment with the real world. However, the team encountered significant challenges due to the inaccuracy of the compass in mobile devices, such as the tablet used in this project. These compasses are highly sensitive to nearby magnetic fields and occasionally point to a completely wrong direction when not first calibrated.

Although there are methods to calibrate the compass for enhanced accuracy, these measures proved to be not accurate enough for the purpose of this application.[25] As a result, it's planned to evaluate alternative techniques in the continuation of this project. Currently, the team has identified two potential solutions, which could be used independently or in combination:

- Implementing a slider for manual adjustments to achieve a precise compass heading.
- Using the GNSS antenna to track movement in a straight line over a certain distance, then calculating the compass direction based on the starting and ending coordinates.

Even a small inaccuracy like 1 degree can lead to significant displacement of the model, especially if the model is large-scale. Therefore, splitting large-scale models into smaller models would mitigate the problem as well.

10.3.3 Gap between camera and GNSS receiver

Because of the previously mentioned challenges such as the geo-alignment problem and the compass issue, the application is currently not able to place the utility lines accurately to the centimeter. Still, there is a problem which will probably be observable as soon as the utility lines can be placed accurately. The complication is the fact that there's a gap between the camera and the GNSS receiver on the back of the tablet. This gap leads to a slight static offset between the position received from the GNSS receiver and the camera. In the figure below the problem can be observed. Because the camera is the actual point zero of the augmented reality session and not the GNSS receiver, the offset will also be observable in the session. Hence, there will always be a little position discrepancy between the virtual utility lines displayed and the actual utility lines with the current solution. When the team reaches the centimeter accuracy within the bachelor thesis, this problem has to be tackled as well.

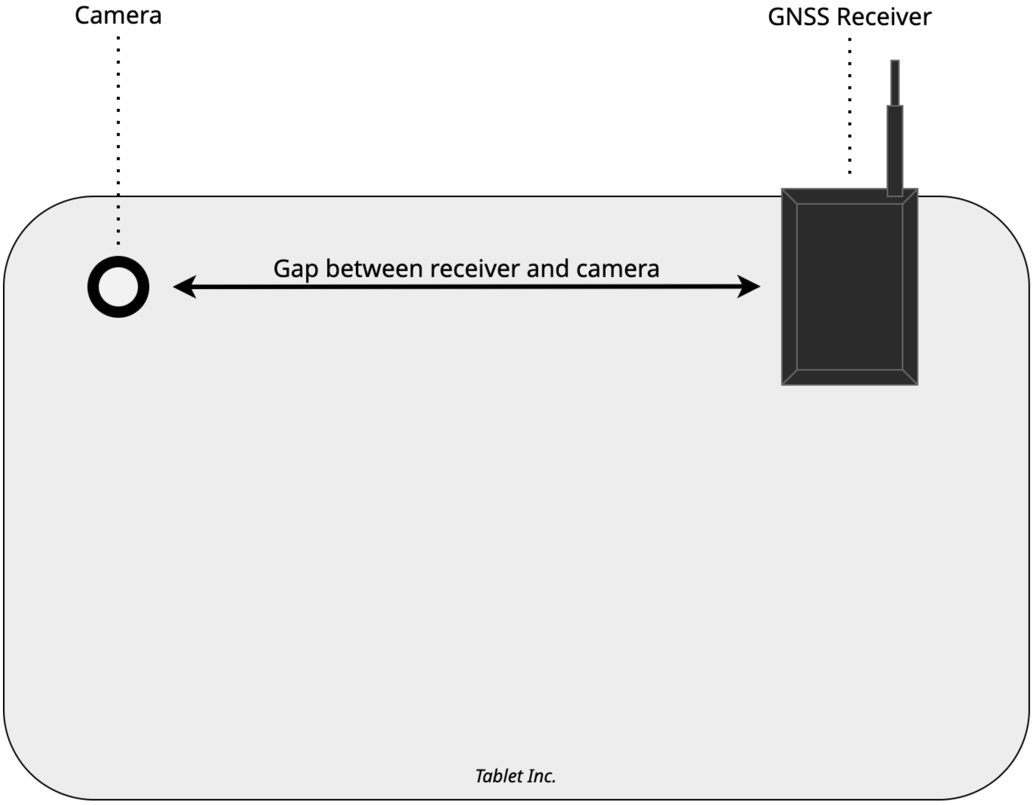


Figure 10.2: Gap between camera and GNSS receiver visualized

Quality Measures

This chapter defines the quality measures implemented to ensure a high standard of the application. It encompasses guidelines, tools, and workflows designed to maintain and enhance the quality of the application.

11.1 Quality Assessment Tools

The section begins by introducing the tools used for quality assessment, such as a LaTeX Formatter (latexindent.pl) for maintaining consistency in documentation and linters (ESLint and Pylint) for ensuring code quality in TypeScript and Python. Furthermore, various guidelines regarding the documentation, code and Git are described.

11.1.1 LaTeX Formatter

Latexindent.pl is a Perl script designed to enhance the appearance and organization of LaTeX code by adding horizontal leading spaces for improved readability.
Source on GitHub
This formatting tool makes it easier to maintain and understand the LaTeX code structure. To keep the documentation consistent, all team members use latexindent.pl as a tool to format the source code.

11.1.2 Linter

For automatically checking that our code conforms with the respective coding guidelines of the language we will use linters for our TypeScript and Python code.

ESLint

ESLint will be used for our TypeScript code. For integration into VS Code, the extension “ESLint” has to be installed.

Pylint

Pylint will be used for our Python code. For integration into VS Code, the extensions “Python” and “Pylint” must be installed.

11.1.3 Guidelines

This section outlines the documentation and coding guidelines adopted by the team for maintaining consistent quality in both writing and programming practices.

Documentation guidelines

The guidelines for the documentation are already given by this LaTeX template. No additional guidelines have been defined.

Code guidelines

Our team complies with the guidelines given by ESLint and Pylint. All warnings must be resolved before pushing code to production.

Definition of Done

Since we work with Jira and all tasks are defined via issues, we define the following Definition of Done for issues.

- **Acceptance criteria**
  - Complies with Functional- and Non-Functional Requirements
  - All defined sub-tasks of the issue have been solved
  - The feature is ready to demonstrate
  - Peer code review by the other team member
- **Quality**
  - Unit Test Coverage is over 80% over all JavaScript and Python files
  - Complies with defined coding standards
  - No Bugs or Code Smells
  - Build Pipeline passed
- **Documentation**
  - All necessary items have been documented

11.1.4 Git-Branching & Merges

To avoid mistakes, it is forbidden to push anything directly to the main branch. When working on an issue, a new branch starting with the respective Jira issue ID must be created. After a developer has implemented his feature and the definition of done is fulfilled (except the peer code review criterion), the developer can create a merge request on GitLab and assign a reviewer. The reviewer has to review the changes and check if it complies with the definition of done. If everything is OK, the merge request will be approved by the reviewer.

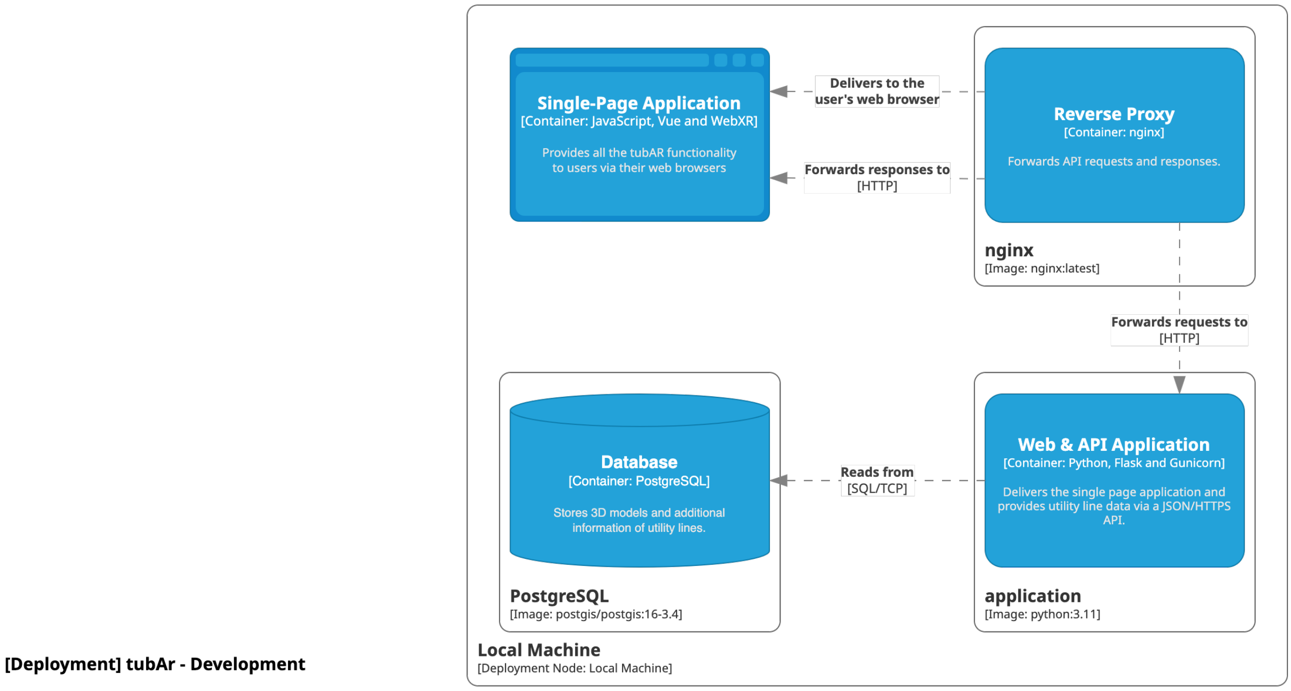
11.1.5 Sprint Retrospective

To regularly reevaluate the quality of the project, each sprint ends with a retrospective meeting.

11.2 Environments

Two Docker compose files were created, one for development and a second one for production. These files define two almost identical compositions of Docker containers. This setup ensures that the application can be developed in a production-like environment. Furthermore, the usage of Docker allows device independent development as always the same setup is given.

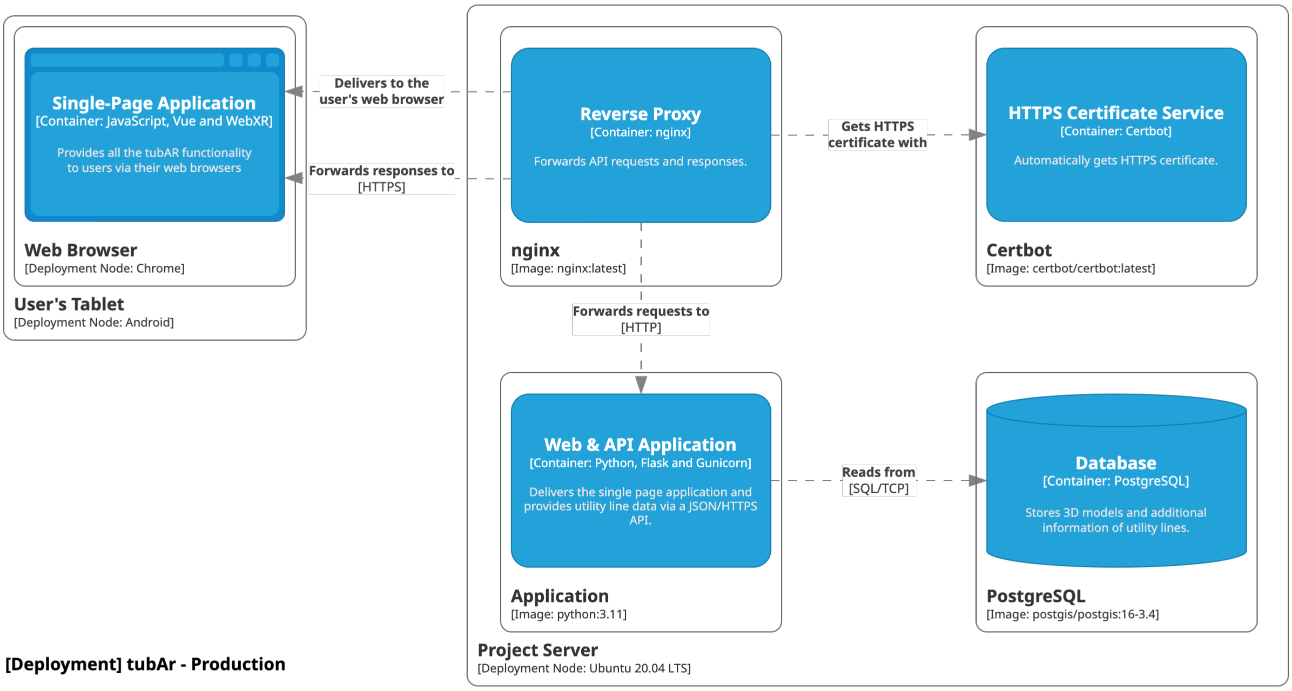
11.2.1 Development



Single-Page Application
[Container: JavaScript, Vue and WebXR]
Provides all the tubAR functionality to users via their web browsers
Delivers to the user's web browser
Forwards responses to [HTTP]
Reverse Proxy
[Container: nginx]
Forwards API requests and responses.
nginx
[Image: nginx:latest]
Forwards requests to [HTTP]
Database
[Container: PostgreSQL]
Stores 3D models and additional information of utility lines.
PostgreSQL
[Image: postgis/postgis:16-3.4]
Reads from [SQL/TCP]
Web & API Application
[Container: Python, Flask and Gunicorn]
Delivers the single page application and provides utility line data via a JSON/HTTPS API.
application
[Image: python:3.11]
Local Machine
[Deployment Node: Local Machine]
[Deployment] tubAr - Development

Figure 11.1: Development environment described as C4 deployment diagram

11.2.2 Production



Single-Page Application
[Container: JavaScript, Vue and WebXR]
Provides all the tubAR functionality to users via their web browsers
Web Browser
[Deployment Node: Chrome]
User's Tablet
[Deployment Node: Android]
Delivers to the user's web browser
Forwards responses to [HTTPS]
Reverse Proxy
[Container: nginx]
Forwards API requests and responses.
nginx
[Image: nginx:latest]
Gets HTTPS certificate with
HTTPS Certificate Service
[Container: Certbot]
Automatically gets HTTPS certificate.
Certbot
[Image: certbot/certbot:latest]
Forwards requests to [HTTP]
Web & API Application
[Container: Python, Flask and Gunicorn]
Delivers the single page application and provides utility line data via a JSON/HTTPS API.
Application
[Image: python:3.11]
Reads from [SQL/TCP]
Database
[Container: PostgreSQL]
Stores 3D models and additional information of utility lines.
PostgreSQL
[Image: postgis/postgis:16-3.4]
Project Server
[Deployment Node: Ubuntu 20.04 LTS]
[Deployment] tubAr - Production

Figure 11.2: Production environment described as C4 deployment diagram

11.3 CI/CD

This section should give a better understanding of how CI/CD was integrated into this project.

11.3.1 Workflow

The team uses the workflow described below to achieve continuous integration and deployment. It is also important to note that the pipeline acts different on development branches and the main branch. On the main branch, the pipeline additionally includes building Docker images and deploying the application to the project server. This means, only commits (stable versions) directly on the main branch will be deployed.

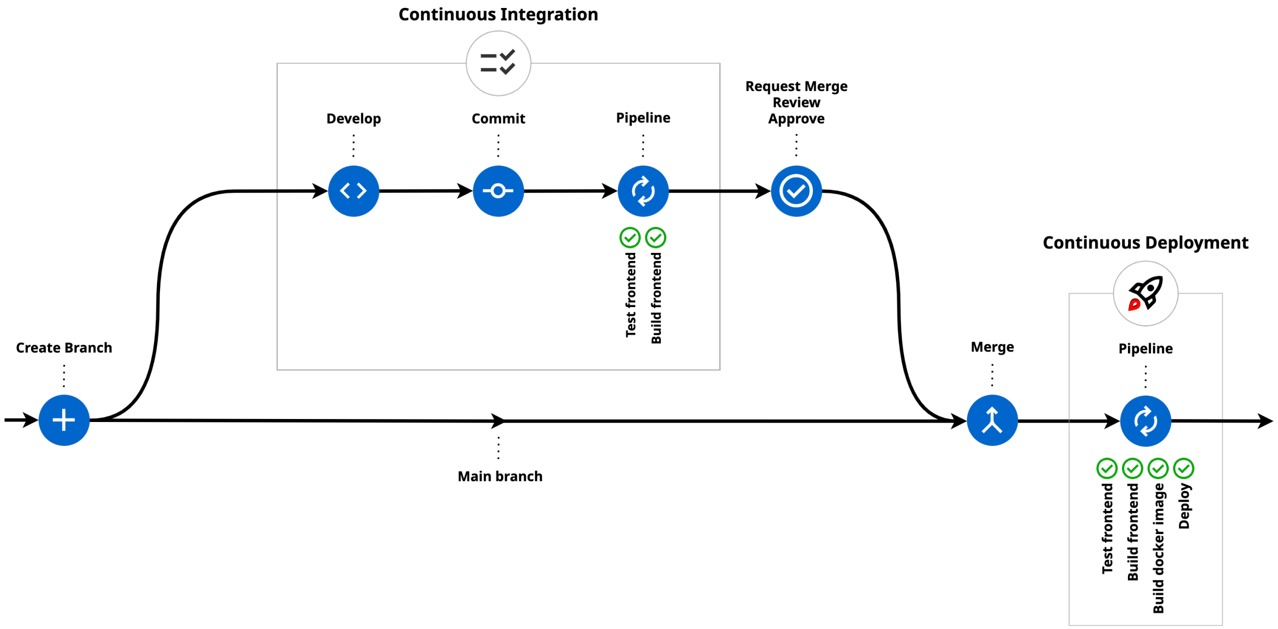
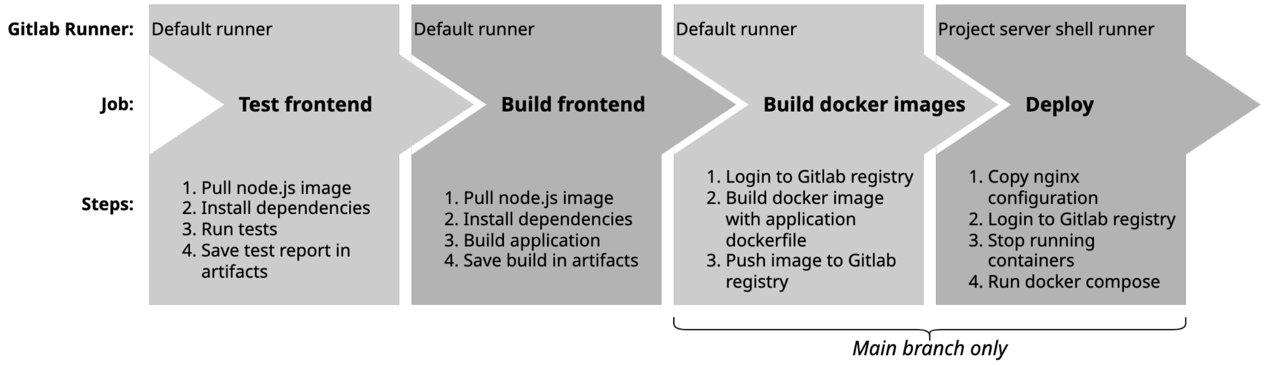


Figure 11.3: Workflow

11.3.2 Gitlab Pipeline

The pipeline only includes testing the frontend, because the backend is tested locally before merging due to database dependency. How the production environment is constructed after deployment can be found in production environment.



Gitlab Runner:	Default runner	Default runner	Default runner	Project server shell runner
Job:	**Test frontend**	**Build frontend**	**Build docker images**	**Deploy**
Steps:	1. Pull node.js image 2. Install dependencies 3. Run tests 4. Save test report in artifacts	1. Pull node.js image 2. Install dependencies 3. Build application 4. Save build in artifacts	1. Login to Gitlab registry 2. Build docker image with application dockerfile 3. Push image to Gitlab registry	1. Copy nginx configuration 2. Login to Gitlab registry 3. Stop running containers 4. Run docker compose

Main branch only

Figure 11.4: Gitlab pipeline

11.4 Test strategy

Testing is an important part of every software project, requiring a well-defined test strategy. This section describes the types of testing and how they are executed in the project.

11.4.1 Types of Testing

Overview over the different types of testing:

- Unit Testing
- UI Testing
- Performance Testing
- Acceptance Testing

Each category has a type of execution, when the tests are executed and a definition of who is responsible to create the kind of tests.

Unit Testing

Testing of individual units of code, such as methods, to ensure they are working as expected.

- Execution: Automated
- Time of Execution: With every build
- Responsibility: Each developer for his units of code

F.I.R.S.T principles: Each unit test has to conform with these principles:

- **Fast:** Unit tests should be fast, meaning they should execute quickly so that they can be run frequently during development.
- **Independent:** Unit tests should be independent of one another, meaning that the outcome of one test should not affect the outcome of another. This helps ensure that each test is testing a specific and isolated unit of code.
- **Repeatable:** Unit tests should be repeatable, meaning that they should produce the same result every time they are run. This helps to detect faulty changes in the code.
- **Self-validating:** Unit tests should be self-validating, meaning that they should be able to automatically determine if they have passed or failed without human intervention. This helps ensure that tests can be run as part of a continuous integration process.
- **Timely:** Unit tests should be written in a timely manner, meaning that they should be written before or directly after the code they are testing is implemented. This helps ensure that the code is written with testability in mind and can help catch bugs earlier in the development process.

AAA pattern: Each unit test will follow a three-step process:

- **Arrange:** During the Arrange phase, the test prepares the necessary prerequisites for the test. This involves creating any required objects, initializing variables, and configuring any dependencies that the code under test relies on.
- **Act:** During the Act phase, the test executes the action being tested. This could involve calling a specific method or interacting with an object.

- **Assert:** In this phase, the test verifies that the action performed in the Act phase has produced the expected result. This involves comparing the actual result of the action with the expected result and failing the test if the two do not match.

UI Testing

Makes sure that the user interface is working properly and all components function as expected.

- Execution: Manual
- Time of Execution: At the end of each sprint during the Construction phase
- Responsibility: Kaj Habegger & Lukas Domeisen

While it's usually possible to automate UI testing, we have decided to do manual testing for this project since it would be very hard to check if 3D objects are correctly displayed. This was also mentioned multiple times at the event "Frontend Best Practices 23 – 3D-Web", which the team attended.

Performance Testing

Make sure the application behaves correctly under different circumstances.

- Execution: Manual
- Time of Execution: At the end of each sprint during the Construction phase
- Responsibility: Kaj Habegger & Lukas Domeisen

Google Chrome includes various development tools for testing the performance of websites.

Acceptance Testing

These tests ensure that the defined functional and non-functional requirements are met.

- Execution: Manual & Automated
- Time of Execution: Before the final release
- Responsibility: Kaj Habegger & Lukas Domeisen

To ensure that all defined requirements are met, we will use a combination of manual and automated testing. While non-automatable requirements will be verified manually, automatable requirements will be integrated into unit tests and checked automatically.

11.4.2 Test Coverage

The GitLab CI/CD pipeline will run tests and create a coverage report for the TypeScript codebase. In order for the pipeline to succeed, no test can fail. For the backend tests, this must be done locally, as it requires a testing database and the effort of integrating this into the CI/CD pipeline is not worth it at the moment.

11.4.3 Tooling

For testing the python code, Pytest will be used. Vitest will be used for the TypeScript code.

11.5 Communication Tools

Microsoft Teams serves as the primary communication platform for online meetings and messaging. Additionally, files which are not required to be part of a Git repository are shared via Teams.

Results & further development

In addition to chapter 5, this chapter focuses on a deeper overview of the results as well as the further development of the application.

12.1 Results

This project lead to a working application, which, as of now, is not ready for production use. The reason for this is that the models of utility lines are not accurately positioned. Nonetheless, the application is reachable via https://srbsci-161.ost.ch/ and can be tested either in Stäfa with real data or on the OST campus in Rapperswil with an educational model.

Throughout this project, a lot of IFC data was gathered. Not every IFC file is meant to be publicly available. A user authentication would be necessary to protect data which should not be available to everyone through the live application. Sadly, there was not enough time left within the scope of this project to implement user authentication. Therefore, not all data at disposal to the team is available through the live application. The application was still tested with all data at disposal.

The frontend comes with a welcome screen which displays a guide of how to set up everything needed for the application to run properly. Furthermore, it features the utility lines viewer which builds the core of tubAR. The whole AR session happens within the utility line viewer as soon as everything is set up and the start button is pressed.

The PostgreSQL database mainly stores the 3D models of the utility lines. By using a script which is only accessible to the team members, additional models can be added to the database. This script converts IFC files to glTF and writes the models' reference coordinate, name and glTF data to the database.

Besides serving the frontend, the backend offers two API endpoints for requesting all available model names and requesting a specific model's data.

Although, the application lacks precision in displaying the models of utility lines, it now builds a solid fundament for further development.

12.2 Further development

In this section, the features already discussed in the results of the technical report are explained in more detail.

User authentication

As mentioned before in results, there is no user authentication available as of now. It is planned to set up user authentication within the bachelor thesis.

Authenticated users will only be allowed to access data they created and data which is accessible by everyone such as educational models of utility lines. The team hasn't decided yet which kind of user authentication should be adopted. This decision will take place in the early phases of the bachelor thesis. Still, the realization shouldn't take up too much time and should remain simple in its implementation. Reason for this is that the focus should stay on the features of the utility lines viewer rather than other things.

Improve model positioning

The application in its current state isn't able to position models of utility lines precisely. Therefore, it is mandatory for the team to highly improve this aspect in the future.

The team has several ideas of how to improve the positioning situation. Firstly, the application heavily relies on the compass to correctly align the virtual coordinate system, but the compass is rather bad in accuracy. This could be improved in two possible ways as discussed in the compass problem. For additional potential solutions, the team will conduct research throughout the course of the bachelor thesis.

Interaction with utility lines

Interacting with utility lines within the viewer was already an idea in the beginning of the term thesis. Furthermore, it was defined as a requirement in US-9. This feature would state an interesting possibility to make the application richer in its functionality.

Besides the feature's potential it probably consumes a lot of time in implementation. In order to display information about utility lines, the information has to be extracted somewhere first. The information could be extracted directly from the IFC file, or it could be entered manually when the data is added to the database. Either way, the task of implementing this feature is time intensive and needs careful planning. Hence, the implementation is postponed to the scope of the bachelor thesis.

Filtering utility lines

Another rather interesting feature which was already meant to be implemented within the term thesis is filtering utility lines by their type. This feature was defined as a requirement in US-8. This couldn't be implemented as well due to time reasons.

Similarly to the interaction with utility lines, this feature requires information extraction from the IFC files or manual appending of the information about what type a utility line has. Therefore, some more analysis of the IFC files is needed. Implementing this could be done simultaneously with the interaction feature.

Dynamic loading of utility lines based on user location

In the application's current state, the location or in other words the model which should be loaded must be manually selected. This can get really confusing when a lot of models are available in the selection dropdown. Instead, the models should be loaded dynamically based on where the user is currently located.

The contemporary idea to implement this is that the backend gets changed so that it only delivers the closest or some of the closest models available to the user's position. This can be achieved by using the capabilities of PostGIS as the reference point of each model already gets stored in the database.

Render models only up to a certain radius

Some models currently available to the team are huge in their size. This leads to a strange or inappropriate visualization of the model in the far distance within the viewer. Furthermore, that visualization issue happens due to the fact that WebXR and Three.js in combination are obviously not intended to render objects which are far away.

A solution would be to cut the models into multiple smaller models. Whereas, all of them need to be tagged with an absolute coordinate and then stored separately in the database. The API could then, again with the power of PostGIS, only request so-called sub-models of the complete models which are needed. This would probably also improve the performance of the AR session as a little side effect.

Improve IFC to glTF conversion

The script which is momentarily used for converting IFC to glTF lacks sophistication in terms of extracting information available in IFC files and generating the glTF data. Sometimes an IFC file also consists of not only utility lines data but also ground data for example. In that case, the script should be able to automatically detect and remove the ground as it is not needed for this application.

It is planned to drastically improve the script in the scope of the bachelor thesis. These planned improvements include that the script can extract information such as type, location and data to be omitted from IFC files. Additionally, the script should be able to add colorization to the glTF data based on the utility lines' type.

Also, the team considers adding a graphical interface for uploading IFC data in the frontend. This interface should communicate with the script through the backend and be able to persist the corresponding data in the database.

Project management

This chapter provides an overview of the methodologies, tools and practices used to effectively manage and coordinate the project. It outlines the team structure, roles and responsibilities, as well as the risk management.

13.1 Resources

This section details the resources available for the project, including team, time, cost and the software tools utilized for specific tasks.

13.1.1 Team

- Kaj Habegger
- Lukas Domeisen

The team consists of two students at the OST - Eastern Switzerland University of Applied Sciences. Lukas has extended knowledge in web development, while Kaj has basic experience in web development. Both of us have already gathered a great theoretical and practical understanding of project management.

13.1.2 Time

The project started with the kickoff meeting on September 19th 2023 and will end on December 22nd 2023, 5:00PM with the final project submission. Each of us is required to spend approximately 17 hours per week working on this project, which makes 34 hours a week and 442 hours for the whole project. Besides the time consumed by the implementation and documentation, meetings also count as work time.

13.1.3 Cost

As this is a university project, we do not have a budget which can be expressed in terms of money. However, our costs can be expressed as the earlier mentioned 442 hours, that we can use for this project.

13.1.4 Tooling

- Code editor: Visual Studio Code
- Version control: Git (GitLab hosted by the OST)
- Issue tracking: Jira
- Documentation: $\LaTeX$
- File sharing: Teams

- Communication: Teams and Jira

13.2 Roles

Roles related to Scrum but also general roles were defined for the project.

- Project Manager: Kaj Habegger
- Scrum Master: Lukas Domeisen
- Developer Team: Both
- Product Owner: Both
- Backend-Engineers: Both
- Frontend-Engineers: Both
- DevOps-Engineer: Kaj Habegger
- Architects: Lukas Domeisen

13.2.1 Responsibilities

- Project Manager:
  - Checks that the project plan is being followed and that the status of work is on track.
  - Maintains an environment where the team can innovate.
  - Maintains time tracking for the project.
  - Generating project related reports for documentation and review.
  - Is responsible that all documents are handed in before according deadlines.
- Scrum Master:
  - Guides the development team through executing the Scrum framework correctly.
  - Plans and leads all Scrum related events / meetings (Sprint Planning, Weekly Meetings and Sprint Retrospective).
  - Maintains the product backlog and reevaluate time estimations.
  - Keeps the risks up-to-date.
- Developer Team:
  - Implements software / infrastructure according to project plan and product backlog.
- DevOps-Engineer:
  - Introduces processes, tools, and methodologies to balance needs throughout the software development life cycle, from coding and deployment, to maintenance and updates.
  - Defines and maintains the CI/CD processes.
  - Maintains the infrastructure for Issue- / Time Tracking.
- Architects:
  - Defines overall structure of the system.
  - Establishes and enforces design principles and standards
  - Documents the architecture.
  - Decides over technologies.

13.3 Processes and Meetings

To achieve an agile project management, Scrum+ (Scrum + RUP) is used to define processes in this project.

13.3.1 Product & Sprint backlogs

Both the product and sprint backlogs are maintained in Jira. Items from the product backlog are refined into issues during sprint planning.

13.3.2 Sprint

Sprint information:

- Sprint duration: 2 weeks
- Sprint start: Tuesday
- Total amount of sprints: 6 (Sprint 0 excluded)

Planning

Each sprint is started by planning.

Planning procedure:

1. Discuss what should be achieved during this sprint.
2. Evaluate which product backlog items should be put into the spring backlog.
3. Discuss which developer takes care of which sprint backlog items.

Further information about planning:

- Frequency: Once every two weeks
- Location: Teams
- Day/Time: Tuesday 2:00 PM
- Duration: 1h

Weekly (Meetings)

The team does weekly meetings, so called weeklys, to catch up with each other’s progress.

Further information about weeklys:

- Frequency: Once per week
- Location: Teams
- Day/Time: Tuesday 2:00 PM
- Duration: 30m

Review

Each sprint-ending is initiated by a review.

Review procedure:

1. Each developer presents what they worked on during the sprint.
2. Progress evaluation of the project.
3. Evaluate and apply environment changes (risk analysis, product backlog adjustments, etc.)

Further information about review:

- Frequency: Once every two weeks
- Location: Teams
- Day/Time: Tuesday 1:00 PM
- Duration: 30m

Retrospective

Each sprint ends by a retrospective.

Retrospective procedure: The team defined a quality measure checklist, which needs to be inspected during the sprint retrospective. If any item is not to the team's satisfaction, the team will define measures for improving the quality in the next sprint.

The quality measures checklist includes following items:

- Documentation quality:
  - Is the documentation correctly formatted?
  - Are there any known grammatical errors?
  - Is anything missing or in need of improvement?
- Time tracking evaluation:
  - Has everyone tracked their time correctly?
  - How good are the estimates for tasks?
  - Did both team members spend about the same amount of time on the work?
- Risk reevaluation:
  - Are there any risks that can be further mitigated?
  - Did the team notice any new risks?
- Code quality:
  - Is the code quality to the team's satisfaction?
  - Are there bugs, vulnerabilities of code smells that need to be taken care of?
- Git / Branches cleanup:
  - Are there any leftover branches that need to be deleted?
  - Did the team encounter any problems with Git?

Further information about retrospective:

- Frequency: Once every two weeks

- Location: Teams
- Day/Time: Tuesday 1:30 PM
- Duration: 30m

13.3.3 Advisor Meetings

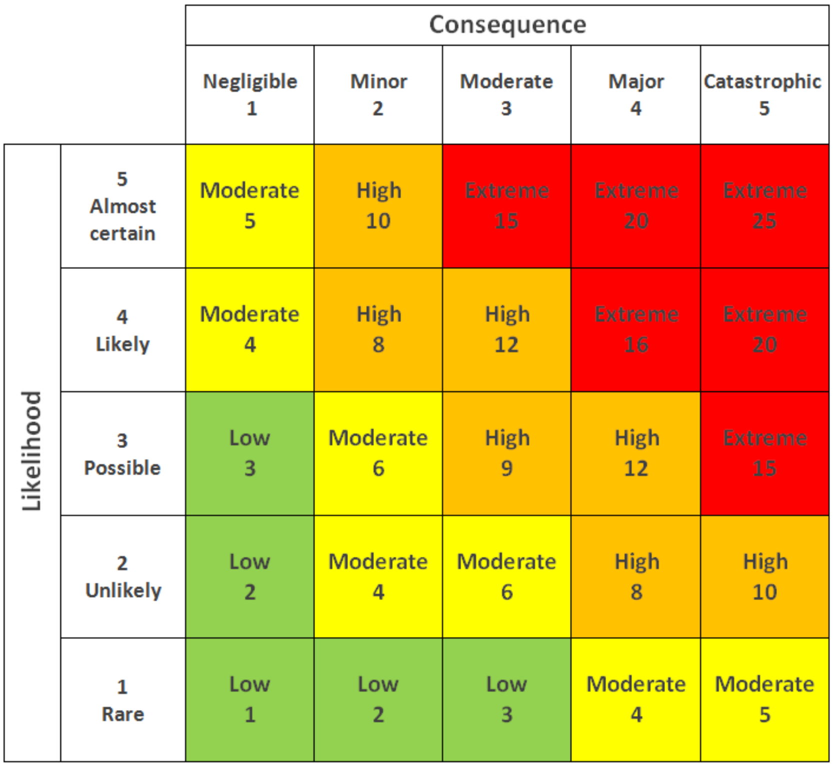
- Frequency: Once per week
- Location: Room 8.261
- Day/Time: Wednesday 3:15 PM
- Duration: 1h
- Additional Attendants: Prof. Stefan F. Keller

13.4 Risk Management

Risk management is a critical process that involves identifying potential risks and developing strategies to mitigate or eliminate them.

13.4.1 Risk categorization method

Risks in the project were categorized and assigned values using the risk matrix below to facilitate their rating. The respective values can be found right next to the title of the risk.



		Consequence				
		Negligible 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
Likelihood	5 Almost certain	Moderate 5	High 10	Extreme 15	Extreme 20	Extreme 25
	4 Likely	Moderate 4	High 8	High 12	Extreme 16	Extreme 20
	3 Possible	Low 3	Moderate 6	High 9	High 12	Extreme 15
	2 Unlikely	Low 2	Moderate 4	Moderate 6	High 8	High 10
	1 Rare	Low 1	Low 2	Low 3	Moderate 4	Moderate 5

Figure 13.1: Risk Matrix

13.4.2 Lack of experience with AR | 9 (Likelihood: 3, Consequence: 3)

Description

All team members have almost no experience with development of AR solutions which makes it quite a big risk.

Mitigation

All team members have completed an AR tutorial to gain some basic knowledge. Additionally, a prototype will be created to try out some of the things which will be needed in the actual web app.

13.4.3 Working as a Team | 2 (Likelihood: 1, Consequence: 2)

Description

Since the team members have already worked on multiple projects together, this risk is very minimal.

Mitigation

Respect the opinions of each other. Try the best to meet deadlines of the tasks, and if not possible let the other team member know as soon as possible.

13.4.4 Specification of requirements/features is inaccurate | 4 (Likelihood: 2, Consequence: 2)

Description

At the start of the project it's hard to perfectly define all requirements as it's unclear how the application will look like in the end.

Mitigation

Usage of agile methodology SCRUM+ for the project. This assures that the requirements will be regularly adjusted in the sprint planning and sprint review meetings.

13.4.5 Jira limitations | 2 (Likelihood: 1, Consequence: 2)

Description

Jira is used for the project tracking and the team has already noticed some limitations, such as creating reports for spent time.

Mitigation

There are many Jira plugins to resolve such limitations.

13.4.6 Time-Management (not enough time) | 12 (Likelihood: 3, Consequence: 4)

Description

It's very likely that the team won't always be able to work the recommended amount of time on the project due to other courses taking up a lot of time.

Mitigation

All team members should try to estimate their available time for each sprint. If the recommended amount of time (17h) per week cannot be met, it should be discussed with the team.

13.4.7 Testing taking up too much time | 12 (Likelihood: 4, Consequence: 3)

Description

Testing can be a very time-consuming task and is often overlooked when estimating effort of features. Especially for this project, in order to manually test the web app with real data we have to physically go the respective coordinates and check their correctness.

Mitigation

Make sure to always include enough time for testing when estimating an issue. To make sure that not too much time is needed for testing, at first only unit tests will be implemented for the respective issue. At a later stage in the project, if the time allows it, more tests will be added such as: integration tests, functional tests, system tests.

13.4.8 Absences / Illness | 10 (Likelihood: 5, Consequence: 2)

Description

Absences and illness could affect the project, due to someone missing in a meeting or not being able to meet the deadline for a task.

Mitigation

Absences and illness should be communicated with the team as early as possible.

13.4.9 IFC data processing | 6 (Likelihood: 2, Consequence: 3)

Description

The given data is in IFC format, which cannot be directly displayed as 3D objects using a 3D web library and has to be converted into a supported 3D format.

Mitigation

Evaluate various possibilities and libraries for converting IFC data.

13.4.10 Inconsistent IFC data structure | 12 (Likelihood: 3, Consequence: 4)

Description

Inconsistencies in IFC files, such as varying fields for utility lines and missing reference coordinates.

Mitigation

More IFC data will be gathered to get an idea of the various inconsistencies and adapt our application accordingly.

13.4.11 Changes History

Risk	Value change	Reason for change	Date
Jira limitations	6 → 2	Necessary extensions have been installed in Jira.	10.10.2023
IFC data processing	12 → 9	Meeting with the external project partner helped to gain more knowledge about IFC. IFC libraries have been evaluated.	10.10.2023
Time-Management	16 → 20	Limited time for the project due to other courses.	24.10.2023
New: Inconsistent IFC data structure	20	Description	24.10.2023
IFC data processing	9 → 6	Script for processing and converting IFC data has been implemented.	07.11.2023
Inconsistent IFC data structure	20 → 12	Inconsistencies have been discussed with Prof. Stefan F. Kellerand more IFC 4.3 data has been gathered.	07.11.2023

Lack of experience with AR	12 → 9	Various AR related issues have been implemented.	21.11.2023
Specification of requirements/features is inaccurate	6 → 4	Unclear features/requirements have been discussed.	21.11.2023
Testing taking up too much time	9 → 12	Issues taking more time than expected.	21.11.2023
Time-management (not enough time)	20 → 12	Reduced scope in this project, because it will be continued as bachelor thesis.	05.12.2023

Table 13.1: Risk changes history

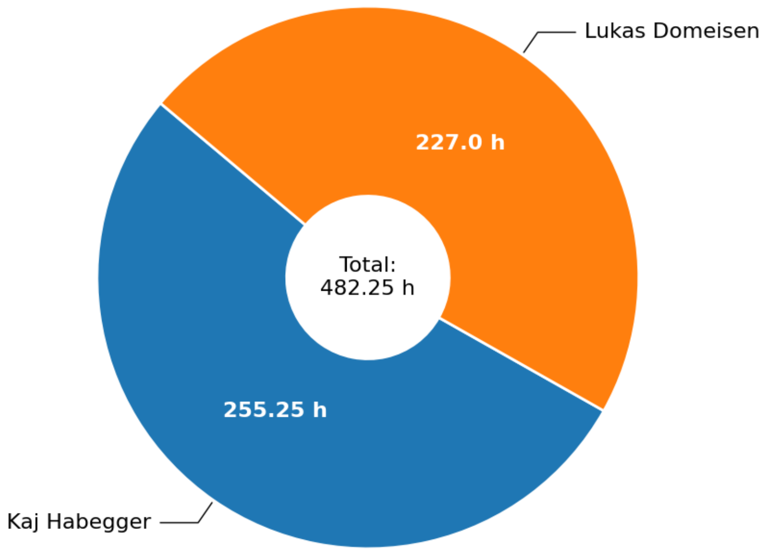
13.5 Long-term Plan

Timeplan		Sprint	Sprint 0	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6	
		SW	1	2	3	4	5	6	7	8	9	10	11	12	13
Inception	Prepare basic documentation structure	*Estimate*	🟧												
		Actual	🟩												
	Setup issue tracking / time tracking (Jira)	*Estimate*	🟧												
		Actual	🟩												
	Setup tablet / GNSS receiver	*Estimate*	🟧												
		Actual	🟩												
	Create long term plan	*Estimate*	🟧												
		Actual	🟩												
	Prepare build-pipelines	*Estimate*	🟧	🟧											
		Actual	🟩	🟩											
	Define roles, processes, resources and meetings	*Estimate*		🟧											
		Actual		🟩											
	Risk-Management	*Estimate*		🟧											
		Actual		🟩											
Elaboration	Requirement-Analysis	*Estimate*			🟧										
		Actual			🟩										
	Evaluate 3D-libraries, test-libraries and IFC-transformation-libraries	*Estimate*			🟧										
		Actual			🟩										
	Domain-model	*Estimate*			🟧 **M1**										
		Actual			🟩										
	Setup build-pipelines	*Estimate*				🟧	🟧								
		Actual					🟩	🟩	🟩						
	Create prototype	*Estimate*				🟧	🟧 **M2**								
		Actual				🟩	🟩	🟩							
Construction	Backend (Transform IFC data to glTF)	*Estimate*						🟧	🟧	🟧	🟧				
		Actual						🟩	🟩	🟩					
	Backend (Implement endpoint for data requests incl. database)	*Estimate*						🟧	🟧	🟧	🟧 **M3**				
		Actual						🟩	🟩	🟩	🟩				
	Frontend (Visualize data)	*Estimate*								🟧	🟧	🟧	🟧 **M4**		
		Actual								🟩	🟩	🟩	🟩		
Transition	Define and execute manual tests	*Estimate*												🟧	
		Actual												🟩	
	Fix bugs and re-test	*Estimate*												🟧	
		Actual												🟩	
	Project reflection	*Estimate*													🟧
		Actual													🟩
	Finalize documentation	*Estimate*													🟧 **M5**
		Actual												🟩	🟩
Legend	● Estimate ● Actual ● Milestones	SW	1	3	4	5	6	7	8	9	10	11	12	13	14
		Sprint	Sprint 0	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6	

Project Monitoring

This chapter includes a diagram that illustrates the hours spent by the team as well as code statistics.

14.1 Time tracking

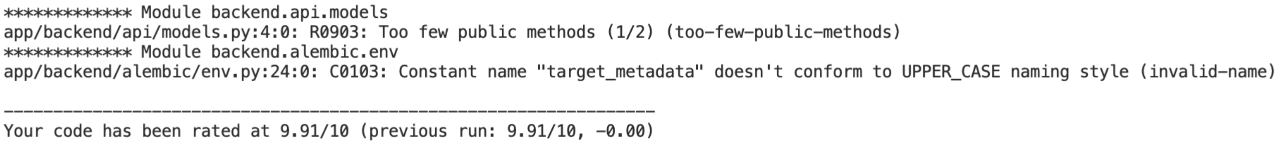


14.2 Code statistics

This section offers an analysis of the project's codebase, serving as an indicator of code quality. The analysis utilizes the designated linters for each programming language, namely Pylint for Python and ESLint for TypeScript.

14.2.1 Backend

As mentioned in quality measures, Pylint is used to ensure the quality of the code, which can also be used to analyze the code and create a report. Pylint evaluates the code and calculates a score out of 10, which reflects the quantity and severity of the issues found. The following report was generated for our Python code. It is important to note that we have selectively ignored certain linter warnings, which were identified as false positives:



```
************* Module backend.api.models
app/backend/api/models.py:4:0: R0903: Too few public methods (1/2) (too-few-public-methods)
************* Module backend.alembic.env
app/backend/alembic/env.py:24:0: C0103: Constant name "target_metadata" doesn't conform to UPPER_CASE naming style (invalid-name)

------------------------------------------------------------------
Your code has been rated at 9.91/10 (previous run: 9.91/10, -0.00)
```

Figure 14.1: Code statistics report by Pylint

14.2.2 Frontend

For the frontend we employ Eslint, configured specifically to adhere to the recommended rules for TypeScript and Vue.js 3. Currently our frontend codebase does not contain any linting issues.

Software Documentation

This chapter summarizes what technologies and tools where used to build the application. Furthermore, it describes some preconditions which have to be met to be able to run the application.

15.1 Technology-stack

Technology	Version
Python	3.11
Flask	3.0.0
SQLAlchemy	2.0.23
Alembic	1.12.1
Gunicorn	21.2.0
TypeScript	5.2.0
Vue.js	3.3.4
Vuetify	3.3.21
WebXR API	N/A
Three.js	0.158.0
Turf.js	6.5.0
PostgreSQL	16.1
nginx	3.18
Certbot	2.8.0

15.2 Tool-stack

Tool	Usage	Version
Blender + BlenderBIM add-on	IFC visualization and visualizing / editing glTF	4.0.2
Open IFC Viewer	IFC visualization	24.9.0
BIMvision	IFC visualization	2.75.5
gltf editor	glTF visualization and editing	online
glTF Viewer	glTF visualization	online

15.3 Installation

15.3.1 RTK receiver

Utility lines need to be displayed precisely in the augmented reality environment. Therefore, it is necessary that the device which is running the augmented reality application can access precise localization. As common mobile devices can only deliver a location accuracy of two to five meters, it is mandatory to use an external RTK capable receiver. RTK is the short form of Real-Time Kinematic positioning.

RTK is a technique used to improve the accuracy of a standalone GNSS receiver. In its simplest form, an RTK solution makes use of a single reference station in proximity to the user receiver. As the reference station is in a surveyed position, it can estimate the errors for each received GNSS signal. After error corrections have been communicated to the user receiver, Integer Ambiguity Resolution (IAR) takes place. This principle works best if the distance between the user and the reference station is reasonably short. When the distance between the user and the reference station grows too large, the atmospheric conditions at the two positions can differ. This may cause from unsuccessful IAR. A typical guideline for max distance can be 25 km [26].

With both the satellites and the base station combined it is possible to get a location with a preciseness of up to less than a centimeter. The RTK receiver used for this project is RTK Handheld Surveyor Kit from ArduSimple.

RTK Handheld Surveyor Kit setup process / test process

There are other ways to set up the kit, but our recommendation is as follows, because we tested it that way. First check if the receiver works fine by following the steps provided below:

1. Set Android language to English.
2. Download and install SW Maps from the Google Play Store.
3. Open SW Maps, click on the SW Maps icon and tap on *USB Serial GPS*.
4. Under *Devices*, you should see **FT232R USB UART**.
5. Set *Baud Rate* to **115’200** and Instrument Model to **u-blox RTK**.
6. Click *CONNECT* button and grant permission if asked.
7. Go back to the menu and tap *NTRIP Connection*.
8. Enter the following details:
   - Address: **rtk2go.com**
   - Port: **2101**
   - Mount Point: **NEAR-Swiss** (See below for alternative Mount Points)
   - User Name: ***your email*** (you will be informed via this e-mail if you got banned by the service for some reason)
   - Password: **none**
9. Click *CONNECT* button and a live data stream will be displayed.
10. Assure that the kit is placed in a location with good view of the sky.
11. It usually takes around 10 seconds until the receiver reaches a low precision error.

To set the receiver up for other applications including the augmented reality application of this project, please follow the instructions below:

1. Download and install the GNSS Master application.
2. Set the GNSS Master application as the mock location application in Android developer settings.
3. Configure GNSS Receiver in GNSS Master application as follows:
   - Mode: **USB Serial**
   - Baud Rate: **115’200**
   - Choose receiver as USB device.
4. Configure NTRIP in GNSS Master application as follows:
   - Address: **rtk2go.com**
   - Port: **2101**
   - Mount Point: **NEAR-Swiss** (See below for alternative Mount Points)
   - User Name: ***your email*** (you will be informed via this e-mail if you got banned by the service for some reason)
   - Password: **none**

> **Alternative Mount Points**
>
> For all Mount Points hosted by rtk2go, a list can be found here. Especially useful if a Mount Point outside of Switzerland is needed.

Part III

Appendix

Appendix A: Deliverables

16.1 Documentation

There exists only digital versions of this document. This document is handed-in as a PDF to Prof. Stefan Keller and the AVT tool of the OST.

16.2 Brochure abstract

The brochure abstract is an additional abstract which must be entered on https://abstract.rj.ost.ch/. This abstract is demanded by the OST and was handed-in on time by the team.

16.3 Signed documents

OST requires a declaration of independence, a declaration of consent for publication on e-prints and an agreement on copyrights and rights of use for each thesis. These signed documents were submitted to the relevant entities on time.

16.4 Source code repositories

- Prototype: https://gitlab.ost.ch/sa-ar-werkleitungen/sa-prototype
- Application: https://gitlab.ost.ch/sa-ar-werkleitungen/sa-application
- Documentation: https://gitlab.ost.ch/sa-ar-werkleitungen/sa-documentation

Only Kaj Habegger, Lukas Domeisen and Prof. Stefan Keller can currently access the repositories.

16.5 Application

The application is accessible via https://srbsci-161.ost.ch/ to everyone.

Appendix B: Glossary and list of abbreviations

Term	Description
tubAR	This is the name of the application or the product which was created within this thesis.
IAR	Abbreviation for Integer Ambiguity Resolution. The openrtk documentation gives further information about this.
RTK	Abbreviation for Real-Time Kinematic positioning. The article from u-blox gives a clear understanding of this technology.
GLB	Abbreviation for Binary file format for glTF data.
GNSS	Abbreviation for Global Navigation Satellite System. Every global available system which uses satellites for positioning is a GNSS. Most prominent is probably GPS.
BIM	Abbreviation for Building Information Modeling. Is the process of modelling physical places in a digital manner. BIM data is a file or multiple files which consists of such data.
IFC	Abbreviation for Industry Foundation Classes. Is a data schema to exchange CAD data. It is descriptive only and does not contain 3D data directly. The Wikipedia entry giver further information about the schema.
GIS	Abbreviation for Geographic Information System. This Esri page gives further information about the definition.
FR	Abbreviation for Functional Requirement. Specifies a specific feature or functionality a software solution must provide.
NFR	Abbreviation for Non-Functional Requirement. Specifies a characteristic or constraint that a software solution must satisfy.
MoSCoW method	Is a prioritization method. M stands for Must-have, S for Should-have, C for Could-have and W for will not have. The Wikipedia entry gives further information about the technique.
Vuetify	A component framework for Vue.js, which simplifies building user interfaces.
WGS 84	World Geodetic System, whereas version 84 is the current version.

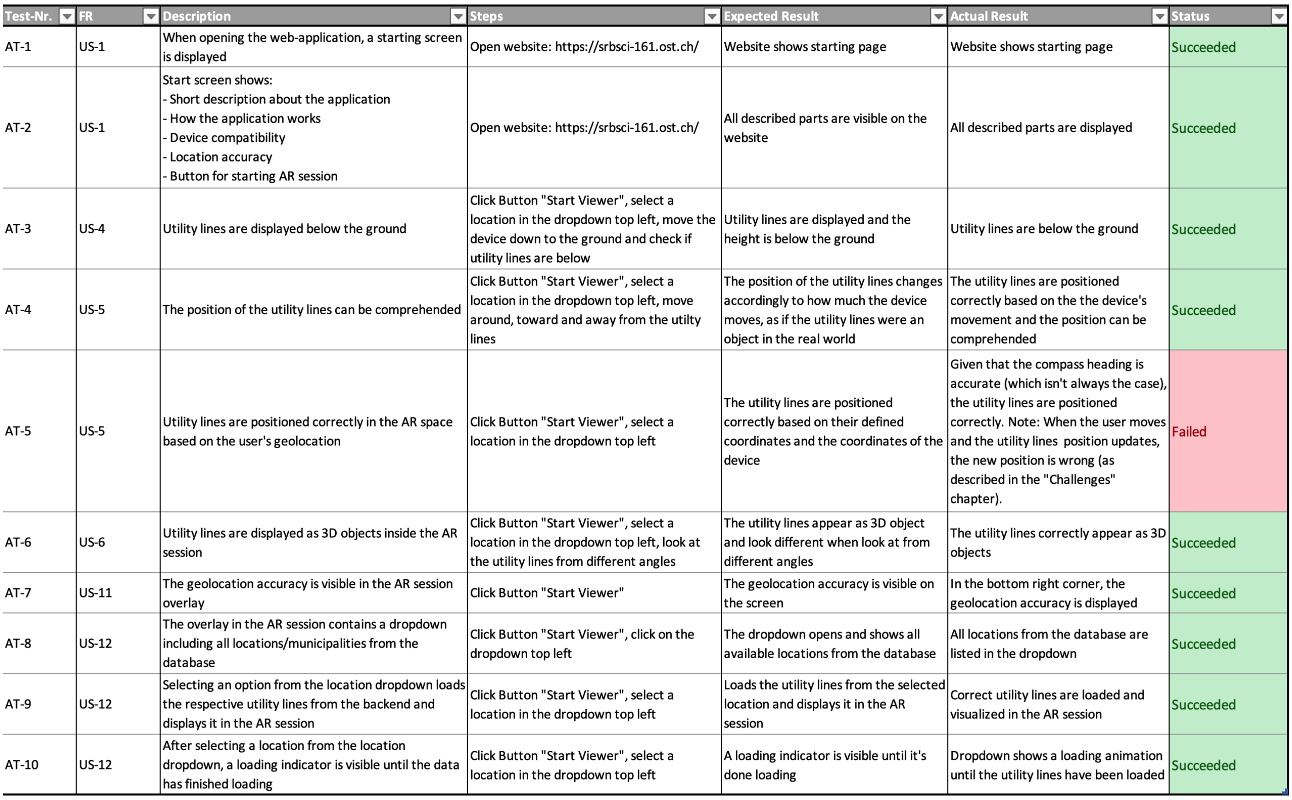
glTF	Graphics Library Transmission Format is a 3D scene and model file format.
MVVM	Model-View-View Model is a pattern used for software architecture.
MVC	Model-View-Controller is a pattern used for software architecture.
Real world environment	When referring to real world environment the explicit real world is meant. It is often used in combination with coordinate system, which in that case means the global coordinate system WGS 84.
Virtual environment	When referring to virtual environment a virtual environment on the table is meant, such as the AR environment, the reference space from WebXR or the Three.js scene. It is often used in combination with coordinate system, which means the coordinate system within this virtual space.
Reference space	WebXR has a reference space, which represents a virtual space where the user is moving in while a XR session is running.
Scrum	Scrum is a framework for team collaboration. At its core is agility, which represents the aim of it. For further information the official Scrum website can be consulted.
RUP	Abbreviation for Rational Unified Process. It is a software development framework using iterations as its base concept. The Wikipedia entry gives further information about the framework.

Table 17.1: Glossary

Appendix C: Test protocols

Acceptance tests

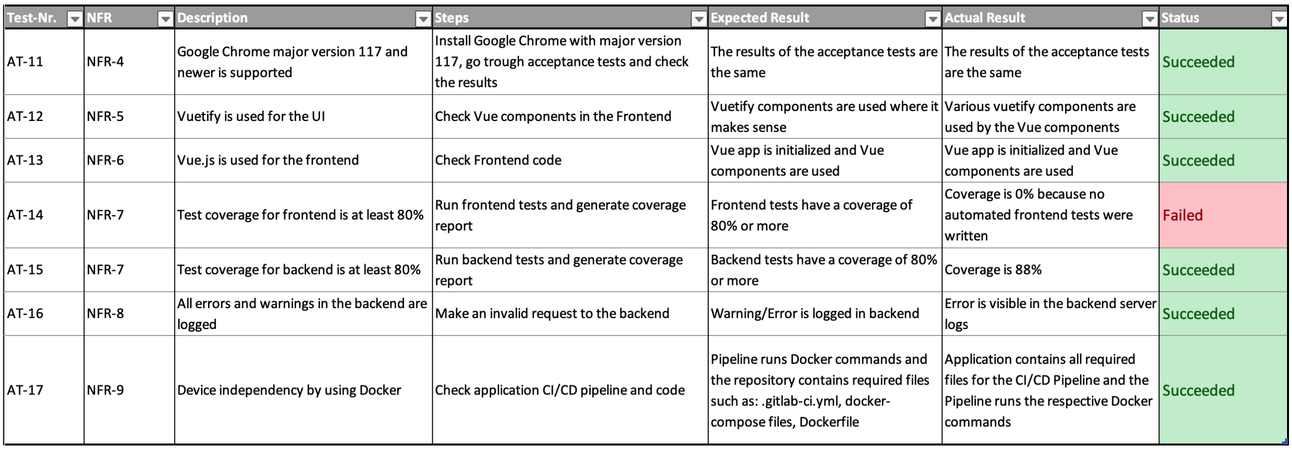
Functional requirements



Test-Nr.	FR	Description	Steps	Expected Result	Actual Result	Status
AT-1	US-1	When opening the web-application, a starting screen is displayed	Open website: https://srbsci-161.ost.ch/	Website shows starting page	Website shows starting page	Succeeded
AT-2	US-1	Start screen shows: - Short description about the application - How the application works - Device compatibility - Location accuracy - Button for starting AR session	Open website: https://srbsci-161.ost.ch/	All described parts are visible on the website	All described parts are displayed	Succeeded
AT-3	US-4	Utility lines are displayed below the ground	Click Button "Start Viewer", select a location in the dropdown top left, move the device down to the ground and check if utility lines are below	Utility lines are displayed and the height is below the ground	Utility lines are below the ground	Succeeded
AT-4	US-5	The position of the utility lines can be comprehended	Click Button "Start Viewer", select a location in the dropdown top left, move around, toward and away from the utilty lines	The position of the utility lines changes accordingly to how much the device moves, as if the utility lines were an object in the real world	The utility lines are positioned correctly based on the the device's movement and the position can be comprehended	Succeeded
AT-5	US-5	Utility lines are positioned correctly in the AR space based on the user's geolocation	Click Button "Start Viewer", select a location in the dropdown top left	The utility lines are positioned correctly based on their defined coordinates and the coordinates of the device	Given that the compass heading is accurate (which isn't always the case), the utility lines are positioned correctly. Note: When the user moves and the utility lines position updates, the new position is wrong (as described in the "Challenges" chapter).	Failed
AT-6	US-6	Utility lines are displayed as 3D objects inside the AR session	Click Button "Start Viewer", select a location in the dropdown top left, look at the utility lines from different angles	The utility lines appear as 3D object and look different when look at from different angles	The utility lines correctly appear as 3D objects	Succeeded
AT-7	US-11	The geolocation accuracy is visible in the AR session overlay	Click Button "Start Viewer"	The geolocation accuracy is visible on the screen	In the bottom right corner, the geolocation accuracy is displayed	Succeeded
AT-8	US-12	The overlay in the AR session contains a dropdown including all locations/municipalities from the database	Click Button "Start Viewer", click on the dropdown top left	The dropdown opens and shows all available locations from the database	All locations from the database are listed in the dropdown	Succeeded
AT-9	US-12	Selecting an option from the location dropdown loads the respective utility lines from the backend and displays it in the AR session	Click Button "Start Viewer", select a location in the dropdown top left	Loads the utility lines from the selected location and displays it in the AR session	Correct utility lines are loaded and visualized in the AR session	Succeeded
AT-10	US-12	After selecting a location from the location dropdown, a loading indicator is visible until the data has finished loading	Click Button "Start Viewer", select a location in the dropdown top left	A loading indicator is visible until it's done loading	Dropdown shows a loading animation until the utility lines have been loaded	Succeeded

Figure 18.1: Acceptance test protocol (functional requirements)

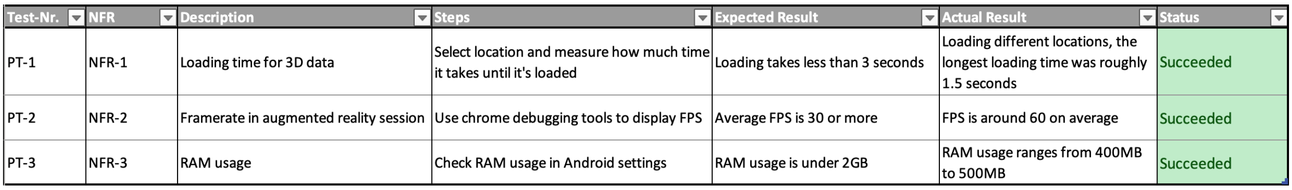
Non-functional requirements



Test-Nr.	NFR	Description	Steps	Expected Result	Actual Result	Status
AT-11	NFR-4	Google Chrome major version 117 and newer is supported	Install Google Chrome with major version 117, go trough acceptance tests and check the results	The results of the acceptance tests are the same	The results of the acceptance tests are the same	Succeeded
AT-12	NFR-5	Vuetify is used for the UI	Check Vue components in the Frontend	Vuetify components are used where it makes sense	Various vuetify components are used by the Vue components	Succeeded
AT-13	NFR-6	Vue.js is used for the frontend	Check Frontend code	Vue app is initialized and Vue components are used	Vue app is initialized and Vue components are used	Succeeded
AT-14	NFR-7	Test coverage for frontend is at least 80%	Run frontend tests and generate coverage report	Frontend tests have a coverage of 80% or more	Coverage is 0% because no automated frontend tests were written	Failed
AT-15	NFR-7	Test coverage for backend is at least 80%	Run backend tests and generate coverage report	Backend tests have a coverage of 80% or more	Coverage is 88%	Succeeded
AT-16	NFR-8	All errors and warnings in the backend are logged	Make an invalid request to the backend	Warning/Error is logged in backend	Error is visible in the backend server logs	Succeeded
AT-17	NFR-9	Device independency by using Docker	Check application CI/CD pipeline and code	Pipeline runs Docker commands and the repository contains required files such as: .gitlab-ci.yml, docker-compose files, Dockerfile	Application contains all required files for the CI/CD Pipeline and the Pipeline runs the respective Docker commands	Succeeded

Figure 18.2: Acceptance test protocol (non-functional requirements)

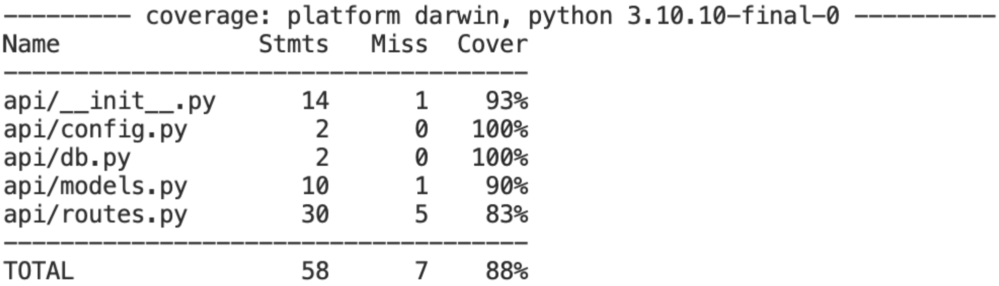
Performance tests



Test-Nr.	NFR	Description	Steps	Expected Result	Actual Result	Status
PT-1	NFR-1	Loading time for 3D data	Select location and measure how much time it takes until it's loaded	Loading takes less than 3 seconds	Loading different locations, the longest loading time was roughly 1.5 seconds	Succeeded
PT-2	NFR-2	Framerate in augmented reality session	Use chrome debugging tools to display FPS	Average FPS is 30 or more	FPS is around 60 on average	Succeeded
PT-3	NFR-3	RAM usage	Check RAM usage in Android settings	RAM usage is under 2GB	RAM usage ranges from 400MB to 500MB	Succeeded

Figure 18.3: Performance test protocol

Backend test coverage



```
--------- coverage: platform darwin, python 3.10.10-final-0 ----------
Name                Stmts   Miss  Cover
---------------------------------------
api/__init__.py        14      1    93%
api/config.py           2      0   100%
api/db.py               2      0   100%
api/models.py          10      1    90%
api/routes.py          30      5    83%
---------------------------------------
TOTAL                  58      7    88%
```

Figure 18.4: Backend test coverage report

Appendix D: Bibliography

[1] arnab roy chowdhury arnab roy. *Best Python Testing Frameworks*. en. Apr. 2022. URL: https://medium.com/@arnabroyy/best-python-testing-frameworks-bb7ab1b3d366 (visited on 12/10/2023).

[2] Vue.js. *Testing*. Dec. 13, 2023. URL: https://vuejs.org/guide/scaling-up/testing.

[3] nicolocarpignoli et al. *AR.js - Augmented Reality on the Web*. Dec. 10, 2023. URL: https://ar-js-org.github.io/AR.js-Docs/.

[4] kalwalt et al. *AR.js*. Dec. 10, 2023. URL: https://github.com/AR-js-org/AR.js.

[5] Wikipedia contributors. *WebXR — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-December-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=WebXR&oldid=1188510753.

[6] toji et al. *WebXR Device API Explained*. Dec. 10, 2023. URL: https://github.com/immersive-web/webxr/blob/master/explainer.md#what-is-webxr.

[7] Immersive Web Working Group. *Immersive Web Working Group - Publications*. Dec. 10, 2023. URL: https://www.w3.org/groups/wg/immersive-web/publications/.

[8] Mozilla. *WebXR Device API*. Dec. 10, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API.

[9] Cesium GS. *The Cesium Platform*. Dec. 10, 2023. URL: https://cesium.com/platform/.

[10] Adobe. *3D file types*. Dec. 3, 2023. URL: https://www.adobe.com/products/substance3d/discover/3d-files-formats.html.

[11] Mariliis Retter. *The Hitchhiker's guide to understanding digital 3D model file formats*. July 20, 2023. URL: https://www.alpha3d.io/3d-file-formats/.

[12] Sonia Schlechter. *Essential Guide to 3D File Formats*. May 5, 2020. URL: https://www.marxentlabs.com/3d-file-formats/.

[13] Khronos Group. *What is glTF?* Nov. 25, 2023. URL: https://www.khronos.org/gltf/.

[14] That Open Company. *FragmentIfcLoader*. Dec. 13, 2023. URL: https://docs.thatopen.com/Tutorials/FragmentIfcLoader.

[15] agviegas et al. *Open BIM Components*. Dec. 10, 2023. URL: https://github.com/IFCjs/components.

[16] IfcOpenShell volunteers. *IfcOpenShell*. Dec. 10, 2023. URL: https://github.com/IfcOpenShell/IfcOpenShell/tree/main.

[17] Three.js. *FAQ*. Nov. 1, 2023. URL: https://threejs.org/docs/#manual/en/introduction/FAQ.

[18] Michael Bayer. *The Python SQL Toolkit and Object Relational Mapper*. Dec. 20, 2023. URL: https://www.sqlalchemy.org/.

[19] Michael Bayer et al. *alembic*. Dec. 20, 2023. URL: https://github.com/sqlalchemy/alembic.

[20] Pallets. *Deploying to Production*. Nov. 24, 2023. URL: https://flask.palletsprojects.com/en/3.0.x/deploying/.

[21] vGIS. *vGIS Technical Details*. URL: https://www.vgis.io/technical-specification-vgis-high-accuracy-survey-grade-augmented-reality-ar-for-bim-gis-arcgis-esri/.

[22] Ardusimple. *RTK Handheld Surveyor Kit*. URL: https://www.ardusimple.com/product/rtk-handheld-surveyor-kit/.

[23] blairmacintyre. *Geospatial Anchors for WebXR*. Dec. 9, 2023. URL: https://github.com/MozillaReality/webxr-geospatial.

[24] rjksmith et al. *Geo-alignment*. Dec. 9, 2023. URL: https://github.com/immersive-web/geo-alignment.

[25] Google. *Compass calibration*. URL: https://support.google.com/maps/answer/2839911?co=GENIE.Platform%3DAndroid&hl=en#zippy=%2Ccalibrate-your-phone-or-tablet.

[26] u-blox AG. *Real-Time Kinematic (RTK)*. Nov. 24, 2022. URL: https://www.u-blox.com/en/technologies/rtk-real-time-kinematic.

Appendix E: List of figures

Appendix F: List of tables