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

Bachelor Thesis

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

Spring Term 2024

Authors:	Kaj Habegger & Lukas Domeisen
Advisor:	Prof. Stefan F. Keller
Project Partner:	Reto Senn - Bitforge AG
External Co-Examiner:	Claude Eisenhut
Internal Co-Examiner:	Dr. Thomas Bocek

12.06.2024

Abstract

Locating underground utility lines, such as water pipes is a complex task because they are usually hidden underground. The aim of this project is to extend and improve the existing solution from the preceding term thesis, which represents an innovative, cost-effective alternative to current solutions. Other solutions are native applications and depend on expensive hardware.

This project is a web-based augmented reality (AR) application for visualizing underground utility lines on Android tablets. It utilizes WebXR and WebGL to integrate Building Information Modeling (BIM) data, specifically Industry Foundation Classes (IFC), into a 3D environment. This involves converting IFC data to a web-optimized 3D format (glTF), using Blender with the BlenderBIM add-on. The web frameworks used for this application are Vue.js with TypeScript for the frontend and Flask for the backend. Some other key technologies of this project are Three.js, Turf.js, IfcOpenShell, Keycloak and PostgreSQL with PostGIS.

The original application has its flaws with accurate positioning of utility line models and has limited functionality. One source of the imprecise alignment of utility line models with the real world is the inaccuracy of the compass sensors used in mobile devices. Therefore, various features to correct these inaccuracies are introduced with this bachelor thesis. These features consist of manual position and rotation controls, as well as a guided compass correction. Other features include colorizing utility lines based on their type, filtering utility lines by their type and automatic loading of utility lines based on the user's location. Additionally, it's now possible to upload IFC files within the application. After uploading, the IFC files are converted to glTF in the backend and saved in the database along with the extracted metadata. In order to upload and view their IFC files, users must first authenticate themselves with either Google or GitHub. A challenge in this project was the lack of available IFC files to properly test the application. Nonetheless, the application was developed with the data available and heavy reliance on the IFC standard, which is publicly available.

Management Summary

Initial situation & approach

Locating underground utility lines, such as water 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 observed in figure 1. The aim of the preceding term thesis was to create a similar application for the web which can be used with off-the-shelf tablets and relies on a relatively low-cost, high-accuracy GNSS antenna with an integrated RTK receiver.

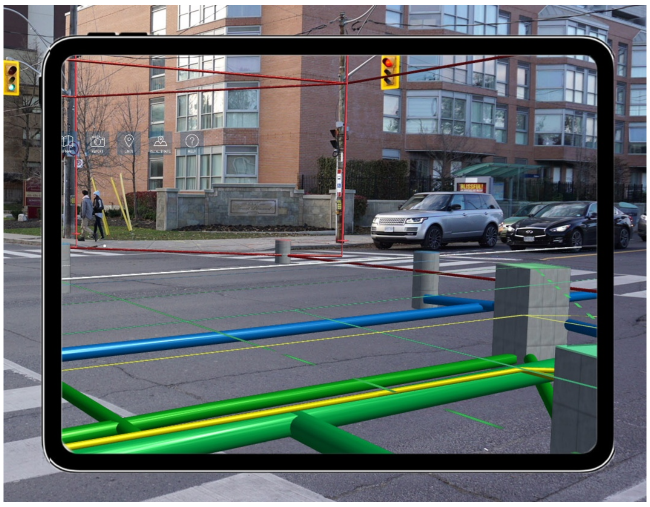


Figure 1: vGIS application [1]

The main purpose of this bachelor thesis is to extend and improve the existing application. It uses the latest web-based AR technologies, WebXR and WebGL, 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, since IFC data is not a 3D format, it can't be directly visualized in a browser. Hence, conversion into a 3D format that is usable by WebGL libraries like Three.js is required.

Starting this thesis, the team began with the detailed definition of requirements derived from the term thesis and the evaluation of possible approaches. Afterward, the implementation process started. This included the primary objectives of user authentication, the possibility to upload IFC directly in the application and enhancements of the AR viewer. Because the team wanted to avoid self-implemented user authentication, Keycloak is used as an identity and access management solution in combination with external social login providers. In order for authenticated users to upload their IFC data, the backend had to be adjusted to be able to convert IFC files to a web optimized 3D format (glTF). The conversion process is made possible by Blender with the BlenderBIM add-on. Moreover, Vue.js with TypeScript was used for the frontend and Python Flask for the backend.

Result

In the resulting application, users can log in using their Google or GitHub account. When authenticated, users can upload their IFC data. They have the option to specify the type and reference coordinates either directly in the IFC file or set custom values in the upload form. The status of the upload and conversion process is displayed in the application. After a successful upload, the utility lines can be visualized in the viewer.

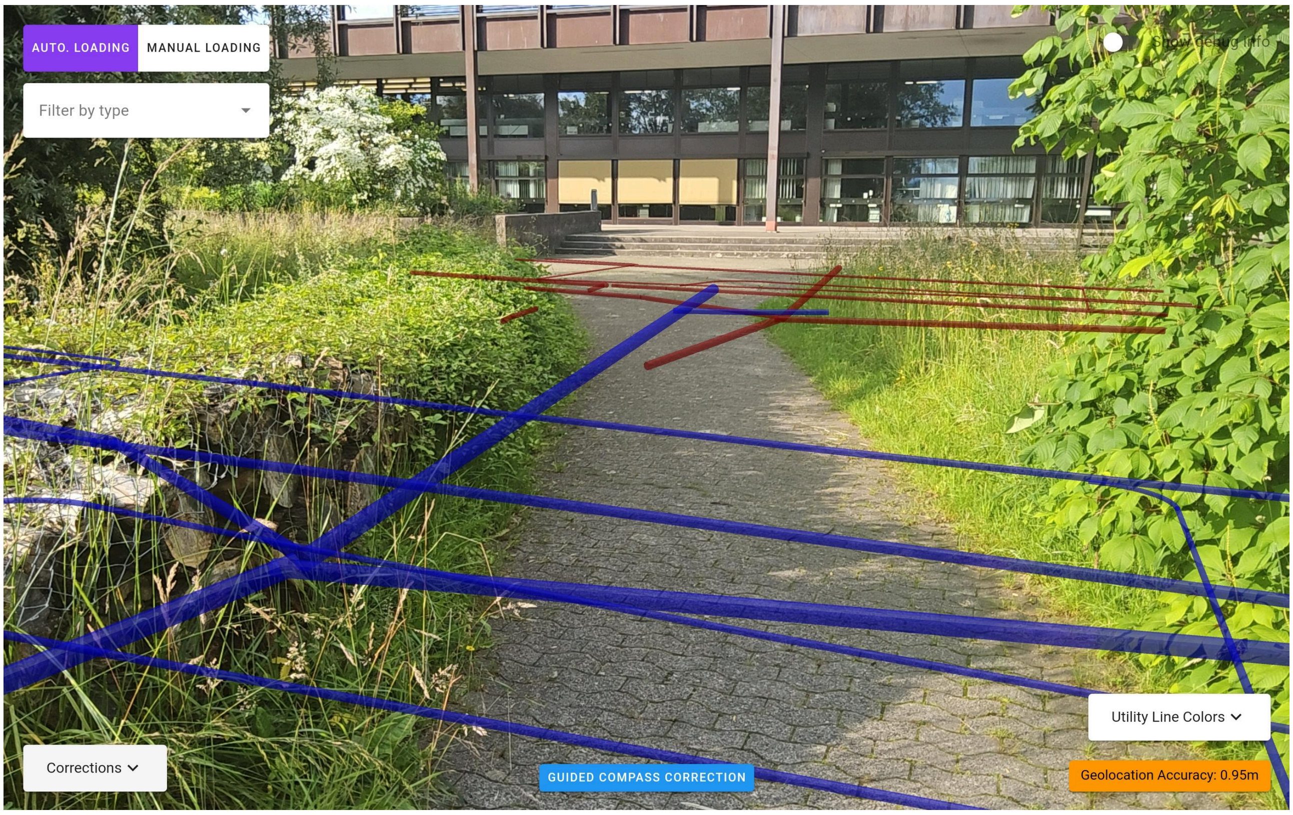
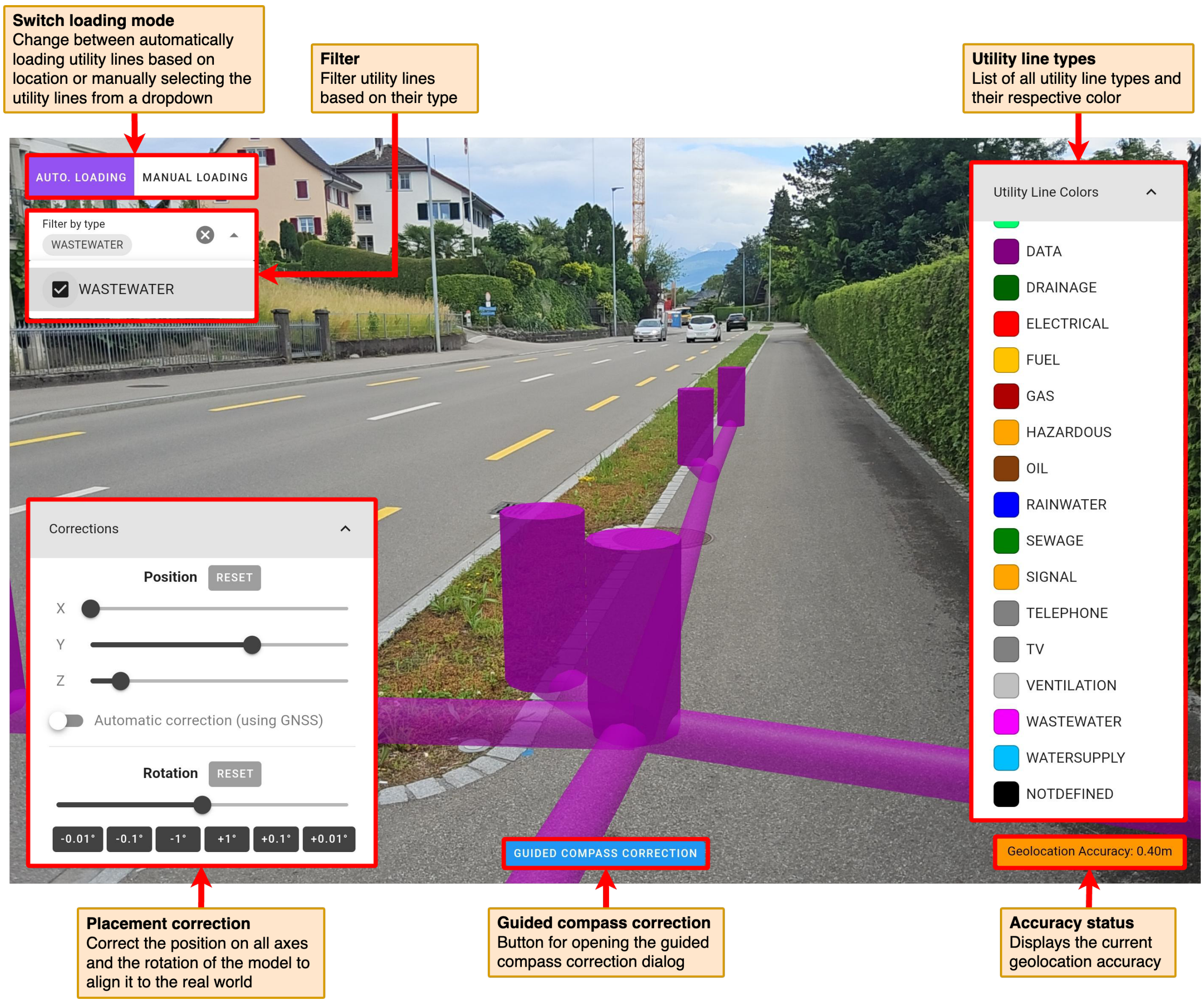


Figure 2: Two educational utility line models with different types visualized on the OST campus in Rapperswil-Jona.

Utility line models can be either loaded manually by choosing them from a dropdown menu or dynamically based on the user's position. When loading utility line models dynamically, they can be filtered by type. Utility line models are colorized based on their type. Although, there is a legend which shows the color to type mapping, users can also click on model parts to inspect their details, such as the type. The utility line models are positioned with an accuracy of less than one meter under optimal conditions. In order to improve the accuracy, users can manually change the position and rotation of the model. To get the model correctly rotated without doing it completely manually, the guided compass correction can be utilized. This instructs the user to walk in a straight line, which allows the application to calculate the compass direction based on the precise start and end points. In order to minimize the visual effect that the utility line models are above the ground, they're rendered slightly transparent.

Figure 3 shows a screenshot from the outdoor test in Stäfa with utility lines correctly aligned and colorized based on their type.


Switch loading mode
Change between automatically loading utility lines based on location or manually selecting the utility lines from a dropdown

Filter
Filter utility lines based on their type

Utility line types
List of all utility line types and their respective color

Placement correction
Correct the position on all axes and the rotation of the model to align it to the real world

Guided compass correction
Button for opening the guided compass correction dialog

Accuracy status
Displays the current geolocation accuracy

Figure 3: Stäfa screenshot with overlay descriptions

Outlook

This project comes to an end with the hand-in of this bachelor thesis. Though, there is still room for extending and improving the application.

One possible improvement is support of other BIM formats beside IFC. Improvements related to IFC processing could be the support for older IFC versions, such as 2.3. Moreover, supporting alternative ways of georeferencing in IFC files would make the application even more accessible.

Configurable colors for utility line types would also enrich the user experience. Various enhancements for processing and loading 3D data could be implemented, in order to reduce loading times. In context of the AR viewer itself, more interactive features, such as distance measurement or data modification, could be added. Allowing users to collaborate with each other by giving access to their utility line models would further increase efficiency. Finally, the effect of utility lines being underground could be improved, for example by using color gradients or displaying a grid on the ground.

Acknowledgements

In this chapter we'd like to show our gratitude to people who supported us during this bachelor thesis.

Starting with our thesis advisor Prof. Stefan Keller, who gave us valuable feedback and shared helpful ideas with us to further improve our work. We appreciated having him as our thesis advisor for both the term and the bachelor thesis.

Second, our external partner Reto Senn who was able to help us out when we weren't sure how to approach certain aspects. We want to thank him for his support.

Third, we'd like to thank everyone who provided IFC files to us, which allowed us to develop and test our application. Additionally, we are grateful to those who answered our questions about IFC with insightful responses.

Last but not least, special thanks to our friends and family for supporting us not only during this bachelor thesis but also the last three years.

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*. Furthermore, it's focused on the specific objectives and requirements of the task, conducted as a bachelor thesis during the spring term of 2024 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. Moreover, no native application should be built nor expensive AR glasses should be used. 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 goal is to further develop the existing application from the preceding term thesis and 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

- Enable interaction with virtual utility lines (such as viewing information about a utility line).
- Implement filtering of utility lines (such as showing water pipes only).
- Show utility lines only within a certain radius.
- Load utility line models dynamically based on the user's position.
- User Authentication.
- Improve positioning of utility line models in the AR session.
- Improve transformation from IFC to glTF.
- Optional: IFC data can be uploaded in the frontend. This data is automatically transformed to glTF and saved in the database.

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, RPyC, IfcOpenShell
- Database: PostgreSQL, PostGIS

Hardware

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

IFC data sources

- Bachelor thesis Jamie Maier
- Geobox AG
- IBU Institut für Bau und Umwelt
- BIM Lab OST
- Tiefbauamt Zurich

Deliverables

1. Tested software
2. Application demonstration video
3. Documentation including text-abstract, management summary and appendix
4. Brochure abstract
5. Declaration of originality

Participants

- Authors: Kaj Habegger & Lukas Domeisen
- Advisor: Prof. Stefan F. Keller
- External project partner: Reto Senn - Bitforge AG
- External Co-Examiner: Claude Eisenhut
- Internal Co-Examiner: Dr. Thomas Bocek

Prior work

This bachelor thesis builds on the preceding term thesis with the same title. The mentioned term thesis was written in autumn term 2023 at the OST - Eastern Switzerland University of Applied Sciences by Kaj Habegger and Lukas Domeisen. All developed software and source code is fully carried over to this bachelor thesis. The following sections should give a better understanding of what parts and functionality of the application are already given by the term thesis.

Frontend

The frontend was developed using Vue.js with TypeScript. Furthermore, WebXR device API was used to enable Augmented Reality (AR) sessions in the browser. The frontend in its term thesis state features a landing page which informs the user about how to use the application and the viewer which serves the AR experience to view utility lines.

API / Web server

The frontend gets the utility lines data, including their coordinates and glTF models, by using an API. This API comes with two endpoints in its term thesis state. While one endpoint delivers all names of available utility line models, another endpoint delivers one specific utility line with its respective data. The server, which acts as the API, also provides the application's frontend.

Database

The API gets its data from a PostgreSQL database which was chosen as the database management system in the term thesis [2, sec. 3.5]. The database consists of one table which holds the utility line's name, coordinates and glTF data as binary.

IFC to glTF conversion

The data of utility lines are initially given in the IFC data format which isn't a 3D format but a descriptive format. To display the utility lines within a WebXR AR session a 3D data format, such as glTF is needed. Therefore, a simple Python script was written during the term thesis to transform IFC data to glTF using Blender with the BlenderBIM extension. After conversion, the data is saved in the previously mentioned database. This script was used manually and externally, and did not run in the backend throughout the term thesis.

DevOps

For a flawless development experience a CI/CD pipeline was set up during the preceding term thesis. The pipeline consists of automatic frontend testing, building the application Docker image and deploying the application to the production server.

Part I

Technical report

Introduction

This project was initially conducted as part of the term thesis in the autumn term 2023 by Kaj Habegger & Lukas Domeisen. The term thesis can be found on https://eprints.ost.ch/id/eprint/1188/. It was continued as part of this bachelor thesis in the spring term 2024 by Kaj Habegger & Lukas Domeisen. The name of the created web application will be referred to as the chosen name “tubAR” in this thesis.

1.1 Problem definition

Locating underground utility lines, such as water tubes, is challenging because they are not visible above ground. Augmented Reality (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 come with high licensing fees.

1.2 Vision

The vision for tubAR is to provide a practical tool that overlays a digital representation of utility lines accurately onto the real world. This should be made possible by using relatively low-priced hardware such as ArduSimple’s RTK Handheld Surveyor Kit [3]. Low-priced additional hardware makes the application more accessible to everyone. Users should be able to use their existing data of utility lines infrastructure with tubAR. tubAR could simplify the management and maintenance of utility lines by providing a more intuitive and direct way of visualizing their location and layout.

1.3 Goals

- Extend the existing application, so it allows users to upload their own data of utility lines infrastructure.
- Implement user authentication so uploaded data is only visible to their owners.
- Introduce quality of life features to the application, such as allowing users to correct the positioning of models within the application’s viewer or loading models based on the user’s location.
- Colorize utility lines based on their type to make them visually differentiable from each other and allow filtering by type.

1.4 Basic conditions

The project represents a bachelor thesis, with each team member dedicating roughly 360 hours to its completion. This time commitment corresponds to 12 ECTS credits.

1.5 Approach

The approach to this project is a combination of a Scrum-like framework (simplified version of Scrum) and RUP, as described in chapter 13. The project is broken down to four phases with each phase focusing 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 process definitions and meeting schedules. However, as this project follows up on the preceding term thesis and can incorporate some of its parts, it will consume much less time than with a completely new project.

Phase 2: Elaboration

During the elaboration phase, the team focuses on defining and documenting both functional and non-functional requirements. Evaluation tasks are the main part of this phase, and they are focused on user authentication services and the application's architecture. Also, the IFC files must be analyzed to gain insight of how and if new requirements can be fulfilled.

Phase 3: Construction

The construction phase is primarily dedicated to the development of the actual application, ensuring that all specified requirements are met. As a first step, the whole architecture must be revised. Afterwards, the IFC converter must be moved to the backend and completely overhauled. To complete the construction phase, the front- and backend have to be adjusted and further developed.

Phase 4: Transition

The final phase concentrates on manual testing of the application and addressing remaining bugs. Finally, the documentation and the other deliverables have to be completed for hand-in.

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, ARUtility, ARonLine and V-Labs.

2.1.1 vGIS

vGIS is a platform specialized in visualization of geospatial data through AR. 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. Additionally, it supports integration with Esri ArcGIS and other GIS systems. They provide capabilities for real-time interaction and modification of utilities, as well as tools for measuring distances. [4] The annual licensing fee is $1250, which doesn't cover the costs of the devices themselves or optional components like LiDAR or photogrammetry technology [5].

2.1.2 ARUtility

ARUtility is a mobile application available for Android and iOS. It's used for locating and managing utility lines using AR. The key features are the ability to update and add new assets directly in the app, distance measurement and easy integration with Esri ArcGIS. [6] Their pricing is $100/month for a monthly subscription, $80/month for a quarterly subscription and $60/month for a yearly subscription. No hardware is provided and therefore must be purchased by the user. [7]

2.1.3 ARonLine

ARonLine is a mobile application developed by GISonLine for Android devices. It's used to visualize underground utilities and assets using AR. The key features of the application are GIS & CAD support and geospatial tools for analysis such as viewing attributes or measuring distances. Their pricing isn't publicly disclosed. [8]

2.1.4 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 isn't publicly disclosed, the cost of the mixed reality glasses alone starts at CHF 3,800. [9]

2.2 Disadvantages

Two primary drawbacks are evident in most of the current solutions: pricing and platform dependency.

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 Dependency

The reliance on native applications by these alternative solutions 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 antenna with an RTK receiver. 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 remarkable advancement in the field of AR-based geospatial data visualization.

Implementation concept

This project is organized on a modular basis, with each part playing a key role in fulfilling the project task. This chapter provides an overview of the project management and implementation strategy.

3.1 Knowledge gathering

During the term thesis, significant initial knowledge was gathered, particularly regarding the tools for AR and IFC data processing. For this bachelor thesis, the knowledge gathering phase primarily includes gaining a deeper understanding of the topics covered in the term thesis. Additionally, research is conducted to explore solutions that would help achieve the goals set for this project.

3.2 Requirements specification

To determine the appropriate tools and solutions, the project requirements must be clearly defined. The requirements are divided into functional and non-functional categories. Detailed descriptions of these requirements can be found in chapter 7. These requirements serve as guidelines when building the architecture and the application itself.

3.3 Evaluation

There are several possibilities of tools or libraries that could be used for a certain requirement. Therefore, an evaluation process is necessary to select the most suitable options. Selection is based on various criteria to ensure the best tools and libraries are chosen for the implementation.

3.4 Architecture

Before implementing any new features, architectural decisions must be made. A solid and stable foundation for the application architecture was established during the term thesis, but adjustments are needed to meet the new requirements. To ensure a smooth integration of these new features, the team must reassess the architecture from the term thesis and make necessary changes. Additionally, the architecture should not become too complex nor inextensible in case of further developments.

3.5 Implementation

Based on the evaluations and the designed architecture, the application can be implemented. Improving the existing application is the core focus of this project.

3.6 Conclusion

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

Results

This chapter outlines the project's accomplishments and compares it with a competitor.

4.1 Goal achievement

The project's success is measured by comparing the actual application against its functional requirements. Below is an analysis of the implemented user stories. The team was able to implement all defined functional requirements, though US-5 not completely. Details on these user stories can be found in section 7.2.

4.1.1 US-1: User login

User login has been successfully integrated into the application utilizing Keycloak. Two social login providers, Google and GitHub, were configured for logging into the application. The website can also be used without login, in which case uploading IFC files isn't possible, and only public utility lines can be viewed. Though, public utility lines are limited to the ones the team uploaded as it's not possible for user's to set utility lines as publicly available.

4.1.2 US-2: Upload IFC data

Users are able to upload their own IFC files and view them in the viewer. The status of the upload and conversion is shown in the frontend.

4.1.3 US-3: Show utility lines only within a certain radius

The camera in the AR environment is configured to only render objects within a 50 meters radius.

4.1.4 US-4: Load utility lines dynamically based on location

When uploading an IFC file, a boundary around the entire model is calculated and saved with the model in the database. When automatic loading is enabled in the viewer, it automatically shows all utility lines where the user is within the boundary.

4.1.5 US-5: Colorize utility lines based on type

Utility lines are colorized in the viewer based on their type, and a list of all types and their respective colors can be viewed in the overlay. One story point of this requirement, allowing users to configure the colors themselves, was not implemented due to time constraints.

4.1.6 US-6: Filter utility line types

In the viewer overlay, users can select specific utility line types to display.

4.1.7 US-7: Info interaction

Clicking on a segment of a utility line model reveals its details in the overlay. The details include the location name, utility line type, name of the selected segment and bounding box size.

4.1.8 US-8: Compass correction

By using a slider, users can manually adjust the rotation of the virtual coordinate system to align it with the real world.

4.1.9 US-9: Accurate compass direction using GNSS (Guided compass correction)

To address the challenge of manually obtaining the correct compass direction, users can get an accurate compass direction using the guided compass correction. When active, this feature instructs the user to walk in a straight line for a short distance. Given the precise start and end point using the GNSS receiver, the compass direction is calculated, and the coordinate system is accordingly adjusted.

4.1.10 US-10: Movable utility line model

Despite previous correction methods and accurate location by the GNSS receiver, model positioning isn't perfect. Therefore, this additional requirement was defined and implemented. This feature allows users to move the models manually on each axis.

4.2 Comparison

This section compares tubAR to vGIS, which is currently the best and most mature alternative solution based on information from their website.

	tubAR	**vGIS**
App type	Web application	Native application (Android, iOS, HoloLens 2) [10]
Technologies	WebXR, Three.js, Vue.js, Flask, PostgreSQL, PostGIS, BlenderBIM, IfcOpenShell	Undisclosed
Key Features	• AR visualization of IFC data • Real-time interaction with models to display details • Uploading IFC data • Automatic loading based on location • Colorization based on type	• AR visualization of IFC and many other formats • Real-time interaction with models to display details • GIS integration with Esri ArcGIS and other GIS systems • Colorization based on type • Data modification and creation directly in the AR environment • Distance measurement [11][12]
Precision	Less than a meter and can be manually corrected. The receiver used in this project is up to one centimeter accurate. [3]	Horizontal: 1cm, Vertical: 2cm, Directional: +/-0.1°[10].
Price	Antenna: €399	Annual license fee: $1250 [5]

Table 4.1: tubAR and vGIS comparison

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

Outlook

This chapter provides a brief overview of potential additional functionalities for improving the application. These functionalities include:

- Supporting older IFC versions and alternative georeferencing methods of models in IFC
- Extending support for various file formats besides IFC
- More interactive controls, such as distance measurement and data correction directly in the viewer
- Allowing users to manually configure colors for utility line types
- Enabling users to collaborate with each other by giving access to their utility line models
- Improve performance of loading utility line models
- Visual enhancements of the viewer to further improve the effect of the utility lines being below the ground

Chapter 11 contains detailed descriptions of these potential features.

Part II

Project Documentation

Vision

The vision can be found in section 1.2.

Requirements specification

The diagram in figure 7.1 shows the main use cases of the system.

7.1 Use case diagram

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

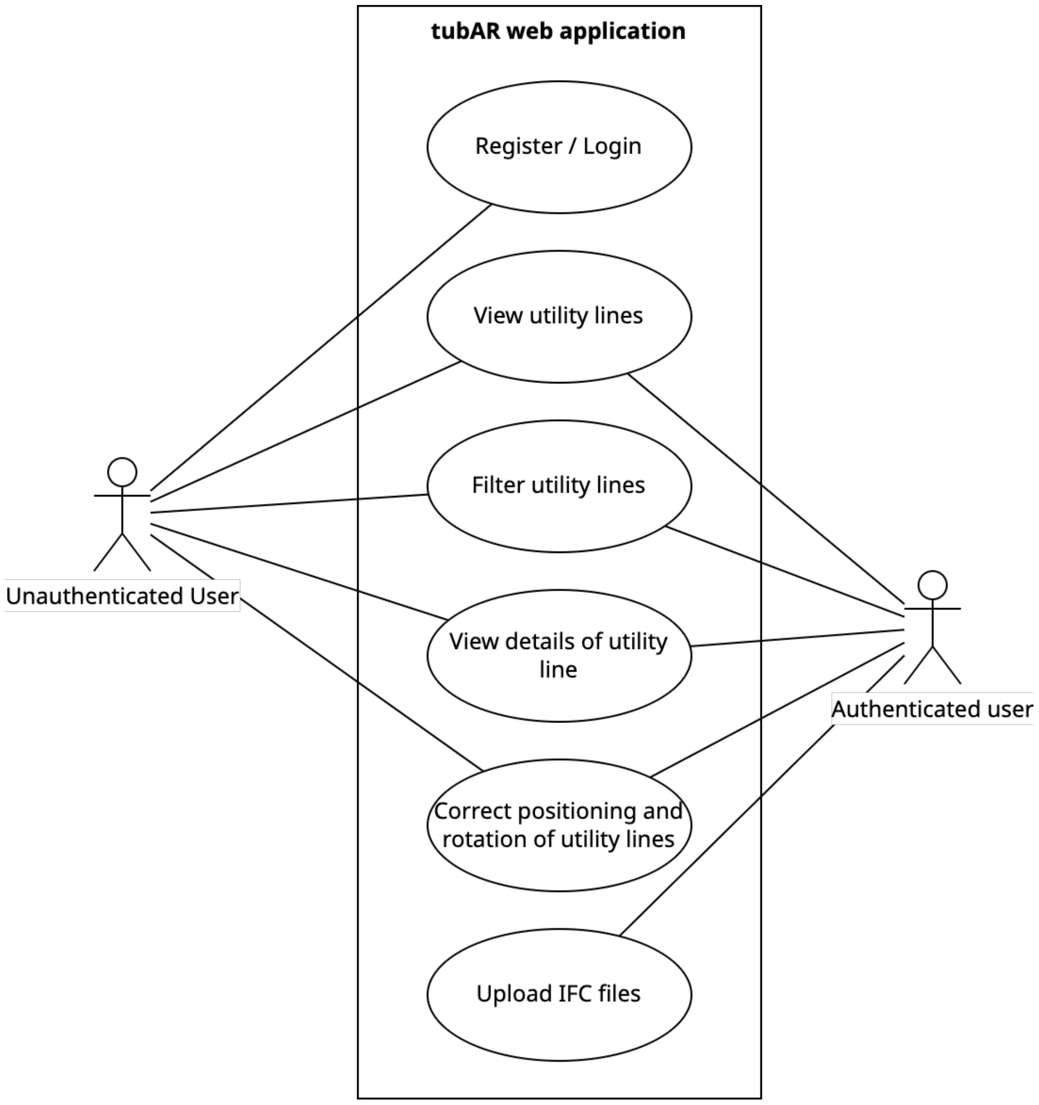


Figure 7.1: Use case diagram

Register / Login

Users can register a new account or login with an existing account.

View utility lines

Users can view utility lines in a virtual AR environment. Utility lines should be correctly positioned based on their reference coordinates.

Filter utility lines

Users can filter utility lines based on their type.

View details of utility line

Users can view more information about specific utility line segments, such as their type.

Correct positioning and rotation of utility lines

Users can adjust the positioning and rotation of utility lines to correct any inaccuracies.

Upload IFC files

Authenticated users can upload their own IFC files and view them in the AR environment.

Manage Users

Administrators can manage registered users.

7.2 Functional Requirements

The functional requirements define the specific features and functionality 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.

7.2.2 User Stories

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

A priority, as well as a rough estimation of required development effort will be assigned to each user story. The Fibonacci scale is a common estimation method 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	User login
Priority	Must have
Time estimation	8
Story points	• The user is able to log in to the application with either Google or GitHub. • The user has to authenticate themselves to upload IFC files and gain access to their data (Uploaded IFC files).

Table 7.1: User login (US-1)

ID	US-2
Subject	Upload IFC data
Priority	Could have
Time estimation	13
Story points	• The user wants to be able to upload their own IFC data. • The user wants to be able to see their uploaded IFC data in the viewer. • The user wants to be able to set the type and reference coordinates for the uploaded IFC file.

Table 7.2: User login (US-2)

Viewer

ID	US-3
Subject	Show utility lines only within a certain radius
Priority	Could have
Time estimation	13
Story points	• The user wants to see utility lines within a reasonable radius.

Table 7.3: User login (US-3)

ID	US-4
Subject	Load utility lines dynamically based on location
Priority	Must have
Time estimation	8
Story points	• The user wants to automatically see the utility lines based on their location.

Table 7.4: User login (US-4)

ID	US-5
Subject	Colorize utility lines based on type
Priority	Could have
Time estimation	13
Story points	• The user wants to be able to determine the utility line types based on their color. • The user wants to see which color belongs to which utility line type. • The user wants to be able to set the color for each utility line type.

Table 7.5: User login (US-5)

Overlay

ID	US-6
Subject	Filter utility line types
Priority	Must have
Time estimation	8
Story points	• The user wants to be able to filter utility lines by type.

Table 7.6: Filter utility line types (US-6)

ID	US-7
Subject	Info interaction
Priority	Should have
Time estimation	13
Story points	• The user wants to be able to access additional information of specific utility line segments by tapping on them.

Table 7.7: Info interaction (US-7)

ID	US-8
Subject	Compass correction
Priority	Must have
Time estimation	5
Story points	• The user wants to be able to manually correct the orientation of the model.

Table 7.8: Compass correction (US-8)

ID	US-9
Subject	Accurate compass direction using GNSS
Priority	Should have
Time estimation	5
Story points	• The user wants to be able to accurately determine the device's orientation by using the coordinates from the GNSS antenna.

Table 7.9: Compass correction (US-9)

ID	US-10
Subject	Movable utility line model
Priority	Should have
Time estimation	5
Story points	• The user wants to be able to manually move loaded utility line models in order to correct their position.
Notes	This is an additional feature and was not part of the requirements defined by the task definition.

Table 7.10: Movable utility line model (US-10)

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.

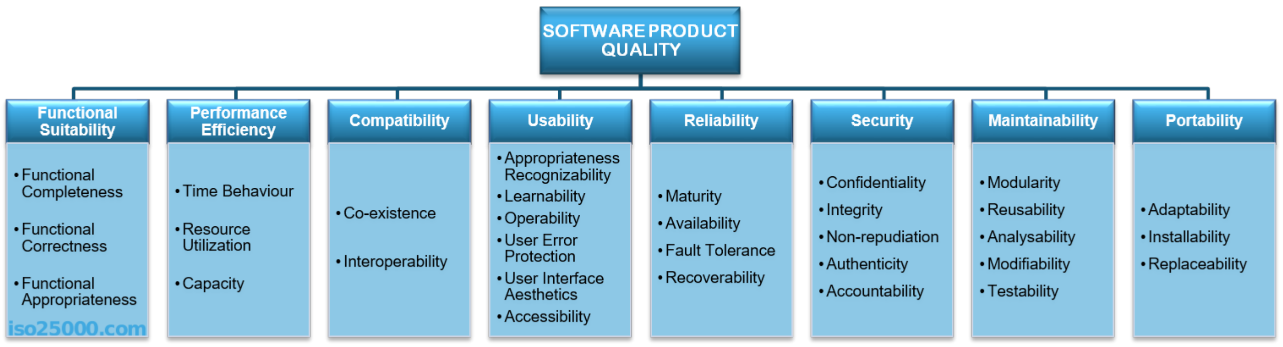


Figure 7.2: ISO/IEC 25010 quality characteristics [13]

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 isn’t 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 team is responsible to check if this NFR is fulfilled during performance testing.

Table 7.11: 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.
Fulfillment check	The team is responsible to check if this NFR is fulfilled during performance testing. This can be displayed with Three.js’ developer tools.

Table 7.12: 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.
Fulfillment check	The team is responsible to check if this NFR is fulfilled during performance testing. This can be checked in the Android settings.

Table 7.13: 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 125.
Fulfillment check	The team is responsible to check if this NFR is fulfilled during acceptance testing.

Table 7.14: 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 the responsible team member has to check if this NFR is fulfilled.

Table 7.15: Vuetify usage (NFR-5)

7.3.6 Security

ID	NFR-6
Subject	Don’t allow unauthorized access to data
Requirement	Confidentiality / Authenticity
Priority	High
Measures	• An OAuth service will be used to achieve this.
Fulfillment check	To reach Done state for Tasks/Issues the responsible team member has to check if this NFR is fulfilled.

Table 7.16: Don’t allow unauthorized access to data (NFR-6)

ID	NFR-7
Subject	Log user actions
Requirement	Non-repudiation / Accountability
Priority	High
Measures	• User actions are logged with their user ID and a timestamp in the back-end.
Fulfillment check	To reach Done state for Tasks/Issues the responsible team member has to check if this NFR is fulfilled.

Table 7.17: Log user actions (NFR-7)

7.3.7 Maintainability

ID	NFR-8
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 the responsible team member has to check if this NFR is fulfilled.

Table 7.18: Usage of Vue.js (NFR-8)

ID	NFR-9
Subject	Unit tests
Requirement	Testability
Priority	High
Measures	• Test coverage must be at least 80% for frontend, backend server and backend converter. • This will be checked in the CI/CD pipeline for the frontend and manually for the backend components. • For each implemented feature unit tests must be implemented by the responsible developer, if reasonable.
Fulfillment check	To reach Done state for Tasks/Issues the responsible team member has to check if this NFR is fulfilled.

Table 7.19: Unit Tests (NFR-9)

ID	NFR-10
Subject	Logging backend
Requirement	Analyzability
Priority	Low
Measures	• All errors and warnings in the backend are logged.
Fulfillment check	To reach Done state for Tasks/Issues the responsible team member has to check if this NFR is fulfilled.

Table 7.20: Logging backend (NFR-10)

7.3.8 Portability

ID	NFR-11
Subject	Dockerization of web application
Requirement	Installability
Priority	High
Measures	• The application will be deployed using Docker images to ensure device independence.
Fulfillment check	To reach Done state for Tasks/Issues the responsible team member has to check if this NFR is fulfilled.

Table 7.21: Dockerization of web application (NFR-11)

Analysis & Evaluation

This chapter provides an overview of the domain model, including detailed descriptions of the elements. Furthermore, it contains a summary of the database model and evaluations of potential technologies for the project.

8.1 Domain Analysis

The domain model in figure 8.1 outlines key entities, attributes, and their relationships within this project's problem domain.

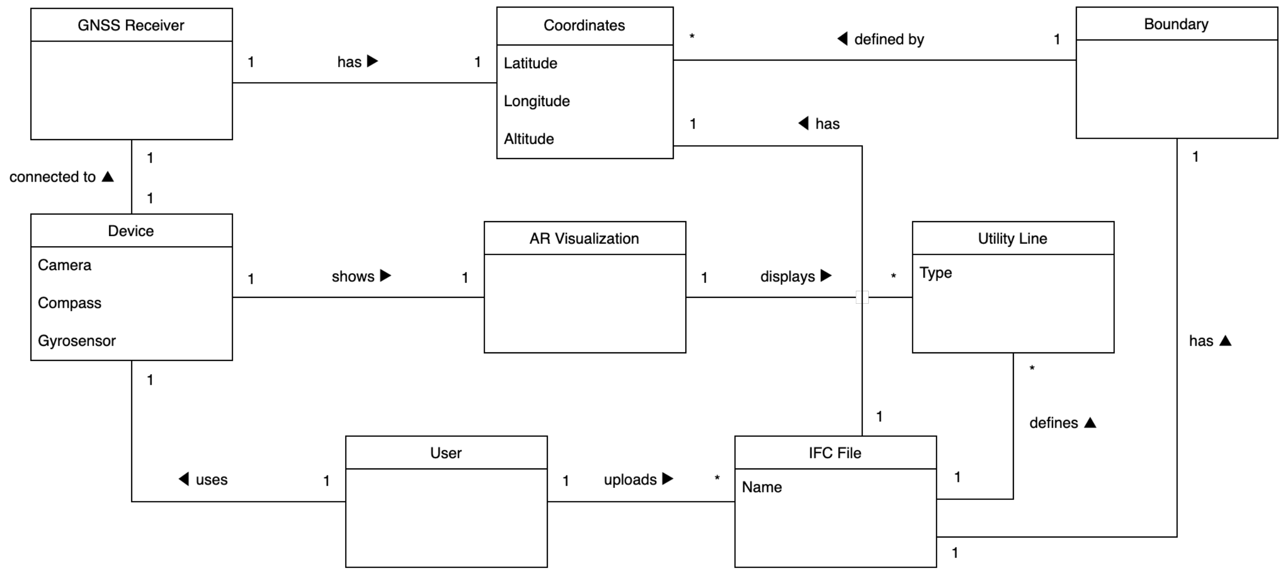


Figure 8.1: Domain model

8.2 Object catalog

The object catalog provides a detailed description of the various elements in the domain model and their relationships shown in figure 8.1.

8.2.1 AR visualization

AR visualization refers to the graphical representation of utility lines within the AR environment of the application. It involves the process of correctly displaying the utility lines in an optimized 3D format.

8.2.2 Boundary

The boundary is calculated from coordinates of utility line segments.

8.2.3 Coordinates

Coordinates are defined by latitude, longitude and altitude using WGS 84. It's used to accurately position the utility lines within the application's AR space, ensuring that the AR visualization aligns correctly with the real-world coordinate system.

8.2.4 Device

The device, typically a tablet, runs the web application and renders the AR visualization of the utility lines. It interfaces with the GNSS receiver and other built-in sensors used for accurate real-time positioning in the AR space.

8.2.5 GNSS Receiver

The GNSS receiver with built-in RTK capabilities is connected to the device and provides high-precision location data. This real-time positioning with high accuracy is crucial for aligning the 3D representations of utility lines with their physical counterparts in the real world. Furthermore, the receiver ensures that the AR visualization is accurate and responsive to the movements of the device.

8.2.6 IFC File

The IFC file contains detailed information about the utility lines, including their type, reference coordinates and dimensions. It serves as the blueprint for generating the 3D representations of the utility lines.

8.2.7 User

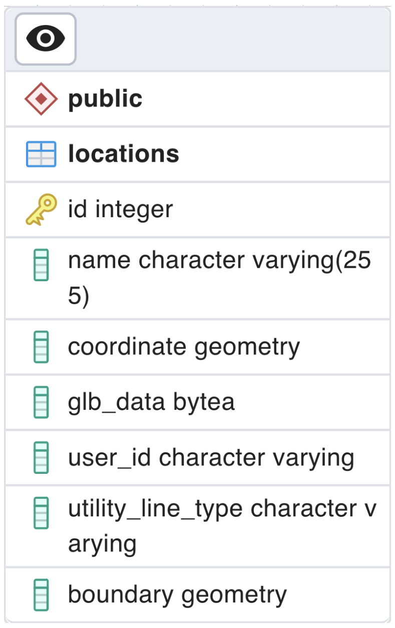
The user interacts with the application through the device. They can view and analyze the utility lines in AR and upload IFC files.

8.2.8 Utility line

Each utility line is a 3D representation of a physical utility line, defined by the IFC file.

8.3 Database model

The database consists of only one table. This is adequate for the scope of this project and its resulting application.



public

locations

id integer

name character varying(25
5)

coordinate geometry

glb_data bytea

user_id character varying

utility_line_type character v
arying

boundary geometry

Figure 8.2: Database model

8.4 Evaluation

This section evaluates different solutions for user authentication and data transmission between the Flask server and the converter, both running in seperate Docker containers. Keycloak is chosen for authentication due to its open-source nature and ease of integration. For data transmission between the Docker containers, an RPC server with RPyC is selected to keep a clean and simple architecture.

8.4.1 User authentication

Given the new requirement for user authentication, this section evaluates various possible solutions.

Acceptance criteria

- Setup/Integration effort
- Provides needed functionality
- Security
- Cost

Keycloak

Keycloak is an open-source and cost-free solution, offering a wide range of features including support for OAuth 2.0 and many social identity providers [14]. Keycloak is designed to be self-hosted, which would not create additional costs for the project as the server is provided by the OST. The deployment of Keycloak can be done with Docker, reducing setup efforts, since Docker is already set up and in use for other components of this project.

Auth0

Auth0 is a commercial alternative to Keycloak, offering a robust authentication service across various application types. It operates on a model that provides a limited free tier up to 7'500 active users, after which a paid subscription is necessary [15]. Although, this project may not exceed the free tier limit, it's still a factor to consider. Unlike Keycloak, Auth0 hosts its services themselves, making authentication dependent on their authentication servers.

Own implementation

Considering an own implementation of user authentication, particularly a direct implementation of the OAuth2/OpenID Connect flow, comes with significant challenges. While it allows for maximum customization, it introduces considerable security risks and demands a high level of effort to develop and maintain. Additionally, opting for a self-built system means all user management aspects, including registrations and session management, would need to be handled internally. Since user authentication isn't the main focus of this project, this option is clearly out of scope.

Conclusion

The team concluded that Keycloak is the most suitable authentication solution for this project. Keycloak's open-source nature, combined with its comprehensive features and Docker integration, provides robust user authentication without significant cost or complexity while keeping a high level of security.

8.4.2 Data transmission between Flask server and converter

Given the requirement for the user to be able to upload their own IFC data and view them in the viewer, the IFC converter must be moved into the backend. Everything in the backend is running in Docker containers. Following this principle, the converter which utilizes Blender to convert IFC files to glTF, should be running in a container too. It was decided not to run the converter in the same container as the Flask server due to modularity and better error resistance in case of a failure. Since the Flask server gets the IFC data from the frontend, it must somehow be able to communicate with the converter container. Hence, two possible solutions for this problem are evaluated in this section.

Acceptance criteria

- Setup/Integration effort
- Modularity

API with Flask

The first possibility would be to run another Flask server but in the same container as the converter. This server exposes an API endpoint, which could be used by the backend to forward IFC data that was received from the frontend. This approach is easy to implement, and no additional Docker container would be necessary besides the one already needed for the converter. However, a major drawback would be that it would run a Flask application besides the converter and the Blender application. Additionally, running a Flask application in a container which isn't directly exposed to the internet is a bit exaggerated.

Message queue with RabbitMQ

The second possibility would be to introduce another container which runs RabbitMQ. RabbitMQ is a reliable and mature messaging and streaming broker. [16] It provides many options to define how messages go from the publisher to one or many consumers. These messages can contain a variety of information types such as plain text, complex structures or even data. In order to work with RabbitMQ, an implementation of one of its supported protocols is needed. As no additional programming language should be introduced to this project, an implementation for Python would be pleasant. Hence, Pika, a pure Python implementation of the AMQP 0-9-1 protocol, comes in quite handy [17]. The idea of this approach is that the Flask server, which communicates with the frontend, sends the received IFC data to RabbitMQ in a queue and the converter container listens on that queue. If a new message appears in the queue the converter fetches the message containing the IFC data, further processes it and stores it in the database. This approach is slightly more difficult than the API with Flask to implement, because the team has no experience with RabbitMQ. Furthermore, this approach would introduce an additional Docker container where the queue runs. Nevertheless, modularity would be kept this way, because for example, the converter could be changed any time to another implementation without bothering the Flask server, which communicates with the frontend.

What is AMQP 0-9-1?

AMQP 0-9-1 (Advanced Message Queuing Protocol) is a messaging protocol that enables conforming client applications to communicate with conforming messaging middleware brokers. The AMQP 0-9-1 Model has the following view of the world: messages are published to exchanges, which are often compared to post offices or mailboxes. Exchanges then distribute message copies to queues using rules called bindings. Then the broker either deliver messages to consumers subscribed to queues, or consumers fetch/pull messages from queues on demand [18].

RPC Server with RPyC

A third option would be to build a simple RPC server which also runs the converter itself. Remote Python Call (RPyC) is a Python library for symmetrical remote procedure calls [19]. Additionally, RPyC offers asynchronous remote procedure calls and registering callbacks for completed procedures. These two features drastically simplify the implementation process, while keeping the architecture tidy. When the client sends an IFC file to the Flask server it would create a task for the IFC file and initiate a remote procedure call to the container which runs the converter. The converter container starts processing the data upon its reception. As soon as the converter converted the IFC file and stored it in the database, it would invoke the callback function which was previously handed over by the caller. With this callback function, the task can be set to completed or failed. The client can check the conversion status of their file via an API endpoint.

Because all remote procedure calls are asynchronous, the Flask server completes its task as soon as the remote procedure call is performed.

Conclusion

Since the team has the goal to develop and maintain a clean architecture but also seeks for simplicity, the RPC server with RPyC will be implemented. Furthermore, this approach has the least integration effort and simultaneously keeps a high modularity, as desired by the acceptance criteria.

Design

This chapter describes the architecture of tubAR, providing an overview of the various parts, such as the front- and backend. It also includes sequence diagrams to visually represent some key processes within the application.

9.1 Architecture

This section outlines the architectural framework and structural elements of tubAR. It includes a detailed explanation of the system's architecture using C4 diagrams and an analysis of the source code structure for both the front- 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 https://c4model.com/.

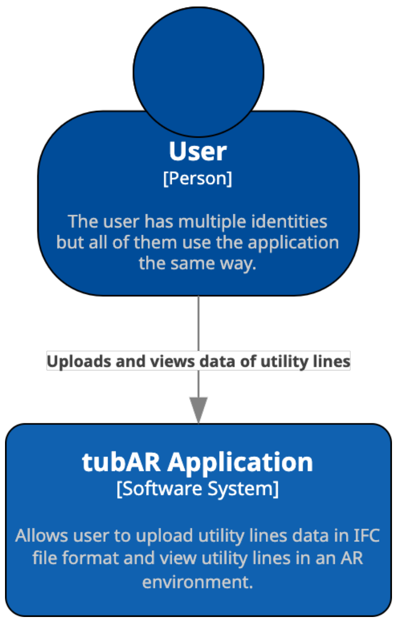

User
[Person]
The user has multiple identities but all of them use the application the same way.
Uploads and views data of utility lines
tubAR Application
[Software System]
Allows user to upload utility lines data in IFC file format and view utility lines in an AR environment.

Figure 9.1: System context diagram

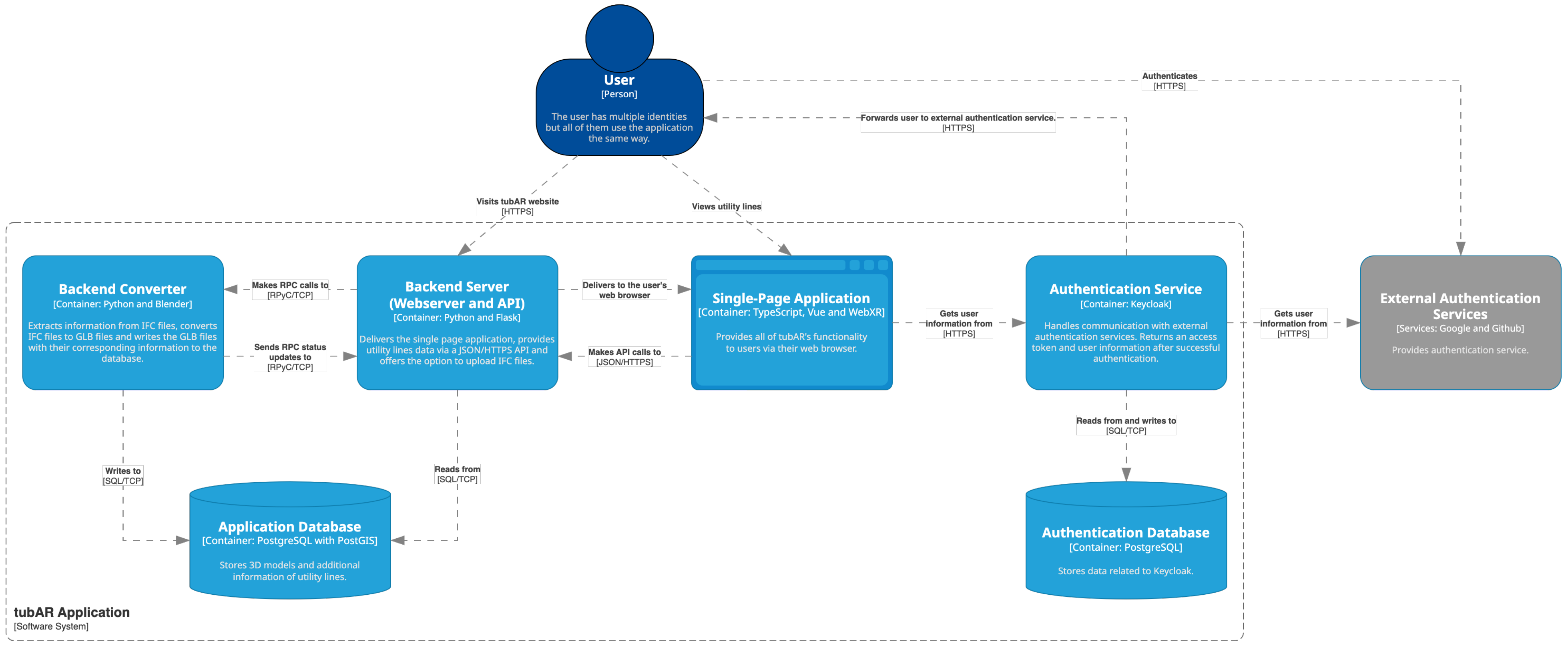

User
[Person]
The user has multiple identities but all of them use the application the same way.
Authenticates
[HTTPS]
Forwards user to external authentication service.
[HTTPS]
Visits tubAR website
[HTTPS]
Views utility lines
Backend Converter
[Container: Python and Blender]
Extracts information from IFC files, converts IFC files to GLB files and writes the GLB files with their corresponding information to the database.
Makes RPC calls to
[RPyC/TCP]
Sends RPC status updates to
[RPyC/TCP]
Backend Server (Webserver and API)
[Container: Python and Flask]
Delivers the single page application, provides utility lines data via a JSON/HTTPS API and offers the option to upload IFC files.
Delivers to the user's web browser
Makes API calls to
[JSON/HTTPS]
Single-Page Application
[Container: TypeScript, Vue and WebXR]
Provides all of tubAR's functionality to users via their web browser.
Gets user information from
[HTTPS]
Authentication Service
[Container: Keycloak]
Handles communication with external authentication services. Returns an access token and user information after successful authentication.
Gets user information from
[HTTPS]
External Authentication Services
[Services: Google and Github]
Provides authentication service.
Writes to
[SQL/TCP]
Reads from
[SQL/TCP]
Application Database
[Container: PostgreSQL with PostGIS]
Stores 3D models and additional information of utility lines.
Reads from and writes to
[SQL/TCP]
Authentication Database
[Container: PostgreSQL]
Stores data related to Keycloak.
tubAR Application
[Software System]

Figure 9.2: Container diagram

9.1.2 Source code structure

The top-level directory structure of the frontend, backend server and backend converter is depicted in figure 9.3.

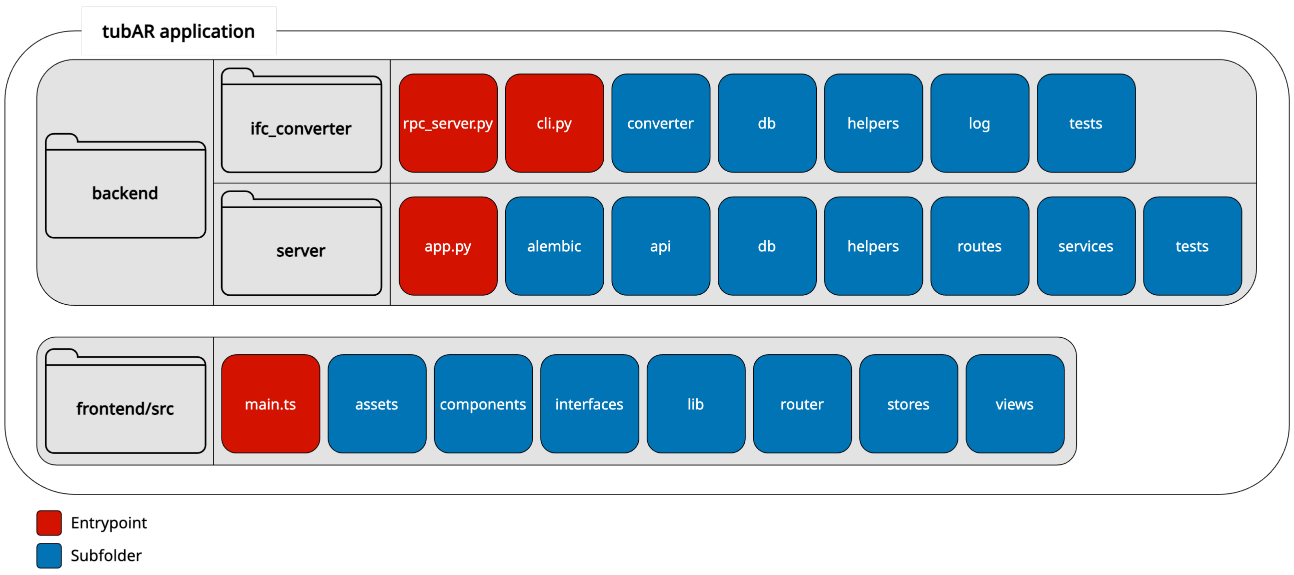


Figure 9.3: Source code structure

- **backend/ifc_converter**:
  - **rpc_server.py**: Contains the RPC server code, such as the configuration and the only exposed remote function, which is utilized by the backend server.
  - **cli.py**: Contains the CLI code, such as the available commands.
  - **converter**: Contains Python files for all converter related things, such as the metadata extractor and the code which is executed through the Blender Python API.
  - **db**: Contains Python files which are needed for database operations, such as creating utility line location entries.
  - **helpers**: Contains the Python files which provide several helping functions for the other components.
  - **log**: Contains the logger.
  - **tests**: Contains Pytest unit tests.
- **backend/server**:
  - **app.py**: Contains the code and configuration of the Flask application.
  - **alembic**: Contains the code for database migrations. If any database modification is needed, this can be done here.
  - **api**: Contains the Python files which define the API endpoints. The Flask application loads these files on startup.
  - **db**: Contains Python files which are needed for database operations, such as getting utility line location entries.
  - **helpers**: Contains the Python files which provide several helping functions for the other components.
  - **routes**: Contains the Python file which defines the only routes besides the API endpoint which returns the frontend. The Flask application loads the file on startup.
  - **services**: Contains the Python files which provide several helping functions for the other components.

  - **tests**: Contains Pytest unit tests.

- **frontend**:
  - **main.ts**: Serves as the entry point for the Vue.js frontend application.
  - **assets**: Contains the base CSS file, which is empty for this application.
  - **components**: Contains Vue.js components which define parts of the application.
  - **interfaces**: Contains interfaces to abstract objects such as coordinates.
  - **lib**: Contains TypeScript files which define general frontend logic, such communication with the API or the authentication service. Also, the logic for the viewer is defined here.
  - **router**: Contains the Vue.js router to enable navigation between the different views.
  - **stores**: Contains the TypeScript file which defines a store object for the correction used by the viewer.
  - **views**: Contains Vue.js components defining the different views of the website.

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.

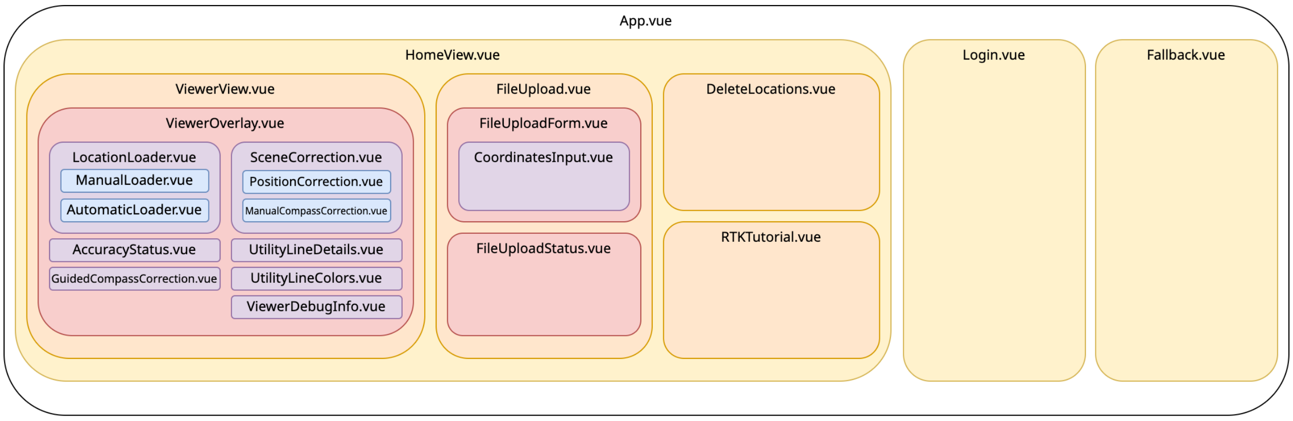


Figure 9.4: Components hierarchy diagram

- **App.vue**: Root component which defines the entry point for Vue.js.
- **HomeView.vue**: Contains the description of the application and is basically the main view of the web application.
- **ViewerView.vue**: Contains the overlay and a button to start a WebXR session. Utilizes the UtilityLineViewer class (described in figure 9.5) for visualizing utility lines.
- **ViewerOverlay.vue**: Acts as the WebXR session overlay. Includes various UI elements, such as controls to correct positioning of utility line models or the current GNSS accuracy.
- **FileUpload.vue**: Contains a form for uploading IFC data. Additionally, it contains a status view, which shows the status of the current upload.
- **DeleteLocations.vue**: Contains the view where users can delete their utility line models.
- **RTKTutorial.vue**: Contains a tutorial for the RTK setup.
- **Login.vue**: Contains the login view where users can choose which service they want to use to log in.
- **Fallback.vue**: Defines a fallback view in case a route does not exist. Contains a link to the home view.

Dependency diagram

Besides the Vue.js 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.

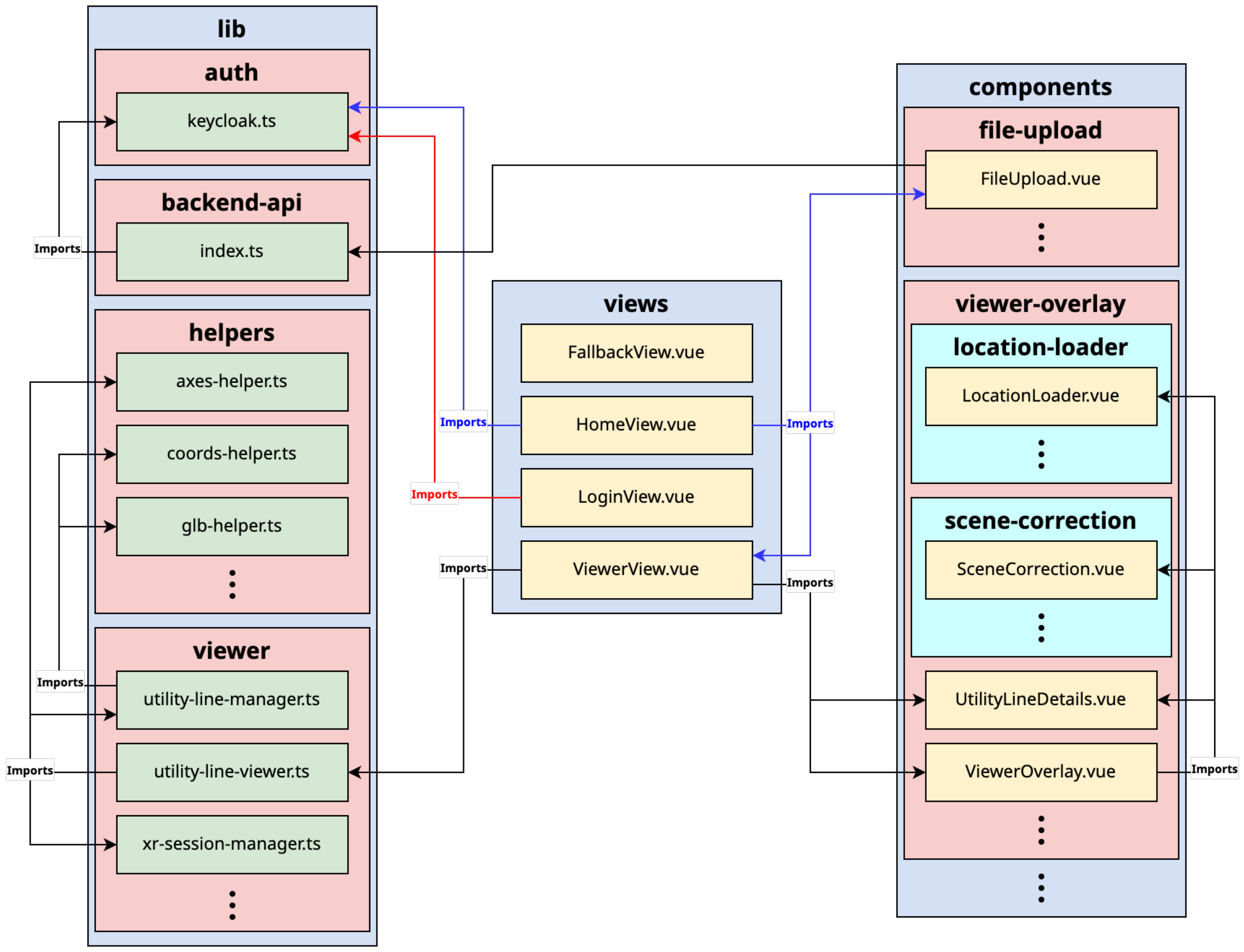


Figure 9.5: Frontend dependency diagram

- **lib**:
  - **auth**: Contains the keycloak.ts file which is used to handle operations related to authentication.
  - **backend-api**: The TypeScript file in this folder contains the code to communicate with the API.
  - **helpers**:
    - **axes-helper.ts**: Generates colored arrows for each axis, which is mainly used for debugging purposes.
    - **coords-helper.ts**: Calculates the relative distance between two WGS 84 coordinates.
    - **glb-helper.ts**: Handles GLB file loading, so they can be used in Three.js.
  - **viewer**:
    - **utility-line-manager.ts**: Defines the UtilityLineManager class. Used for loading utility lines as GLB and adding it to the scene.
    - **utility-line-viewer.ts**: Defines the UtilityLineViewer class. Uses the UtilityLineManager and XRSessionManager to correctly update and place utility line models.

    - **xr-session-manager.ts**: Defines the XRSessionManager class. Handles parts related to the WebXR session such as starting the session, rendering the scene, object selection, etc.
- **views**: All views are already explained in figure 9.4.
- **components**: All components are already explained in figure 9.4.

9.1.4 Backend

The backend server (Flask application) and the backend converter do not follow any specific patterns, but common structures for similar projects. Though, the factory pattern is utilized for creating the Flask application instance. This allows to have multiple instances with different configurations, which is very useful for testing.

Dependency diagram - Backend server

The files of the backend server and their relationship are illustrated in figure 9.6.

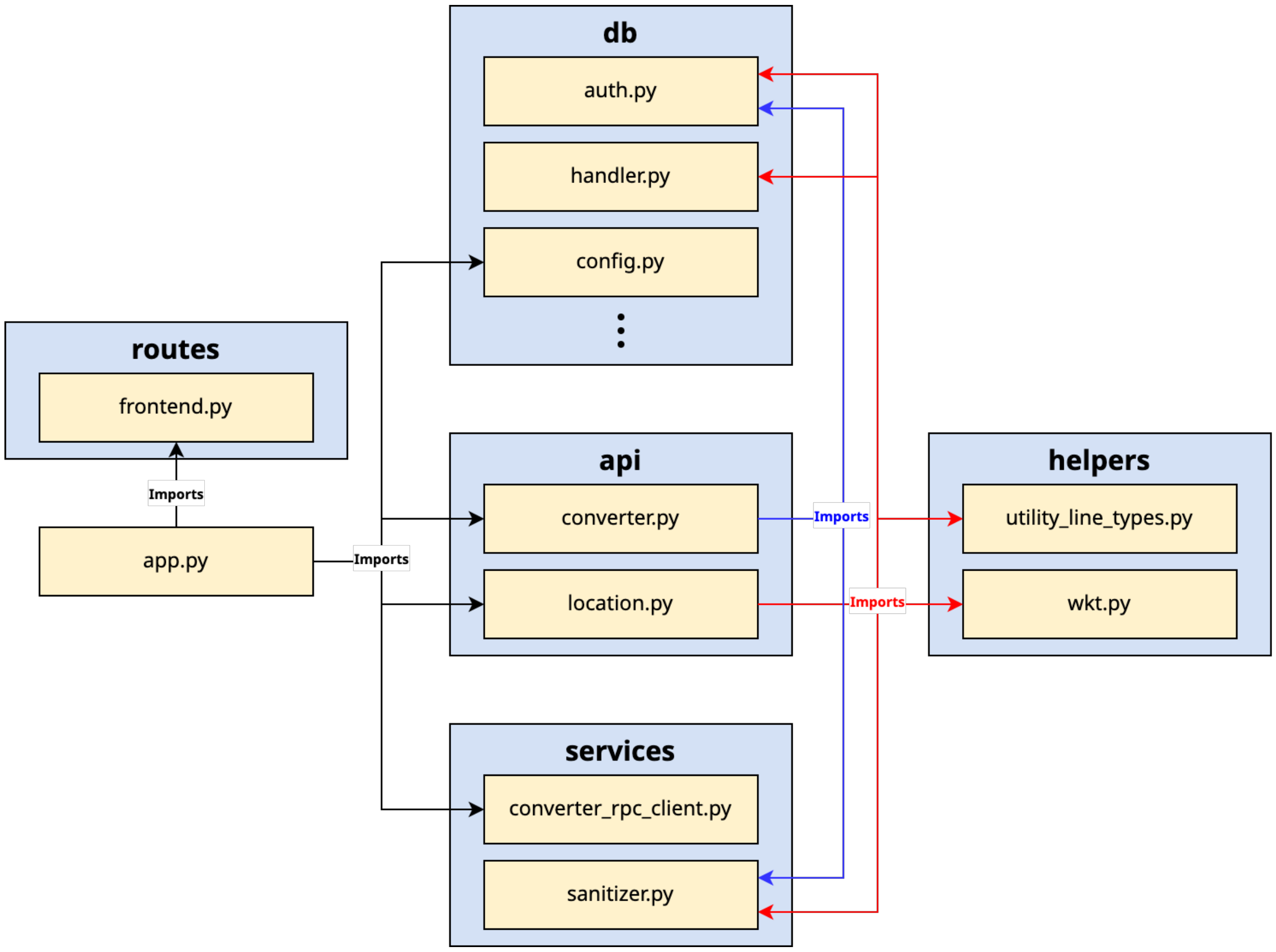


Figure 9.6: Backend server dependency diagram

- **app.py**: This is the entry point of the Flask application.
- **routes**: The frontend.py file in this module contains the only non API route, which returns the frontend. This is a catch-all route which is invoked when a request matches no other endpoint.
- **api**: Both Python files in this module define API endpoints used by the frontend.
- **services**: The converter_rpc_client.py file contains the code to communicate with the RPC server and sanitizer.py provides methods to sanitize user input.
- **db**: The Python files in this module provide functions to write to and read from the database. The functionality is implemented with the ORM-Framework SQLAlchemy.
- **helpers**: The Python files in this module provide helper functions.

Dependency diagram - Backend converter

The backend converter runs in a separate container than the backend server.

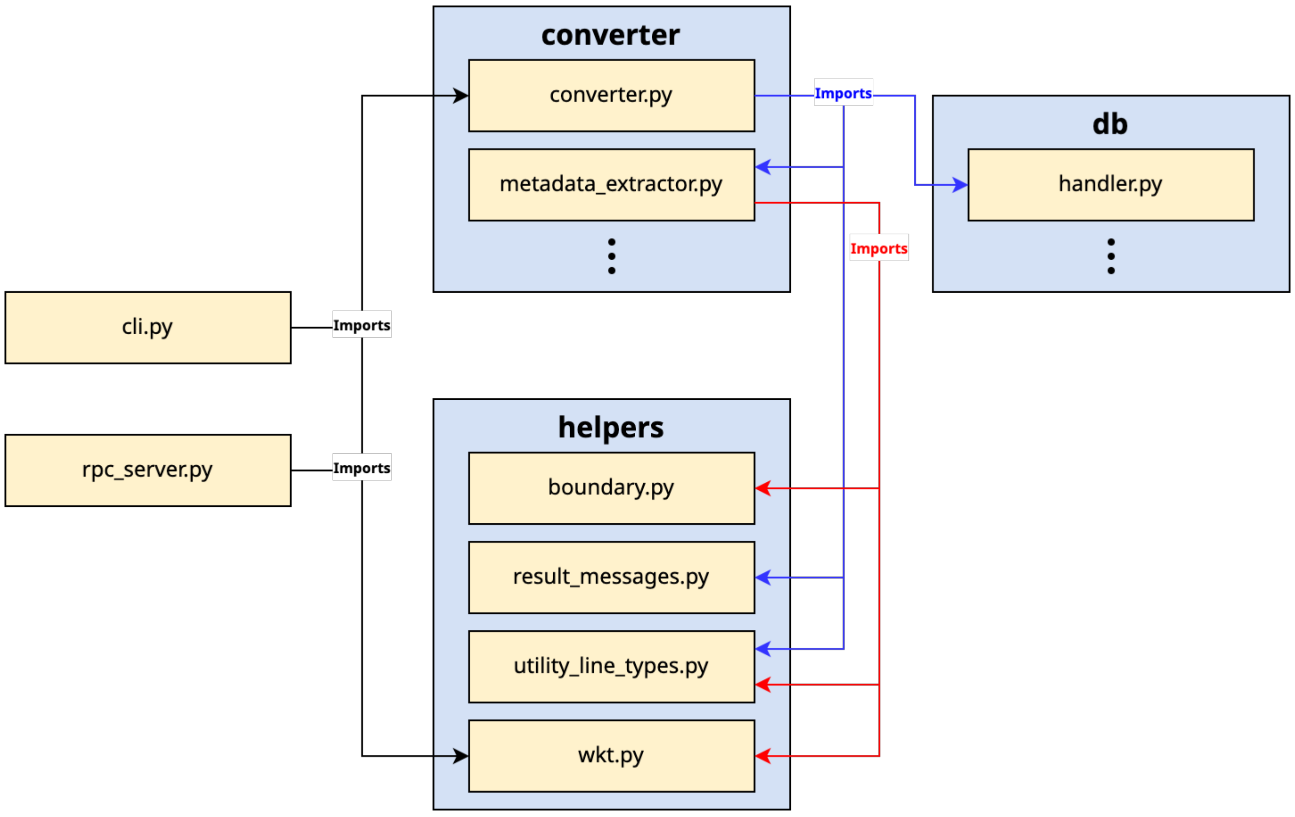


Figure 9.7: IFC converter dependency diagram

- **rpc_server.py**: This Python file contains the RPC server code which uses the converter.
- **cli.py**: This Python file contains the CLI code which uses the converter.
- **converter**: The Python files in this module contain the logic related to the whole process of converting IFC files to GLB files up to writing everything to the database.
- **helpers**: The Python files in this module provide helper functions. For example boundary.py is used to calculate the boundaries of utility line models.
- **db**: The Python files in this module provide functions to write to the database. The functionality is implemented with the ORM-Framework SQLAlchemy.

9.1.5 Keycloak

As mentioned before, Keycloak was integrated into the application for managing user authentication. Since the architecture is already based on Docker, the Keycloak Docker image was used to achieve this [20].

Configuration

Setting up Keycloak required multiple configurations in the admin interface. By default, Keycloak comes with a “Master” realm which should only be used internally and not for external applications. For the application, a new realm named “tubar” was created, and within this realm a new client with the same name was created. The client configuration includes essential parameters such as the website’s home URL and the URLs permitted for redirecting users after logging in.

Identity providers

The application is configured to support user authentication via Google and GitHub. To achieve this, new OAuth applications were created with both Google and GitHub, obtaining unique client IDs and client secrets to register in Keycloak. Additionally, attribute mappings were configured to automatically map the attributes from the external providers to the attributes of the Keycloak user.

Authentication flow

Figure 9.8 visually represents the authentication process, showing the sequence of interactions between the user, the application, the identity providers and Keycloak.

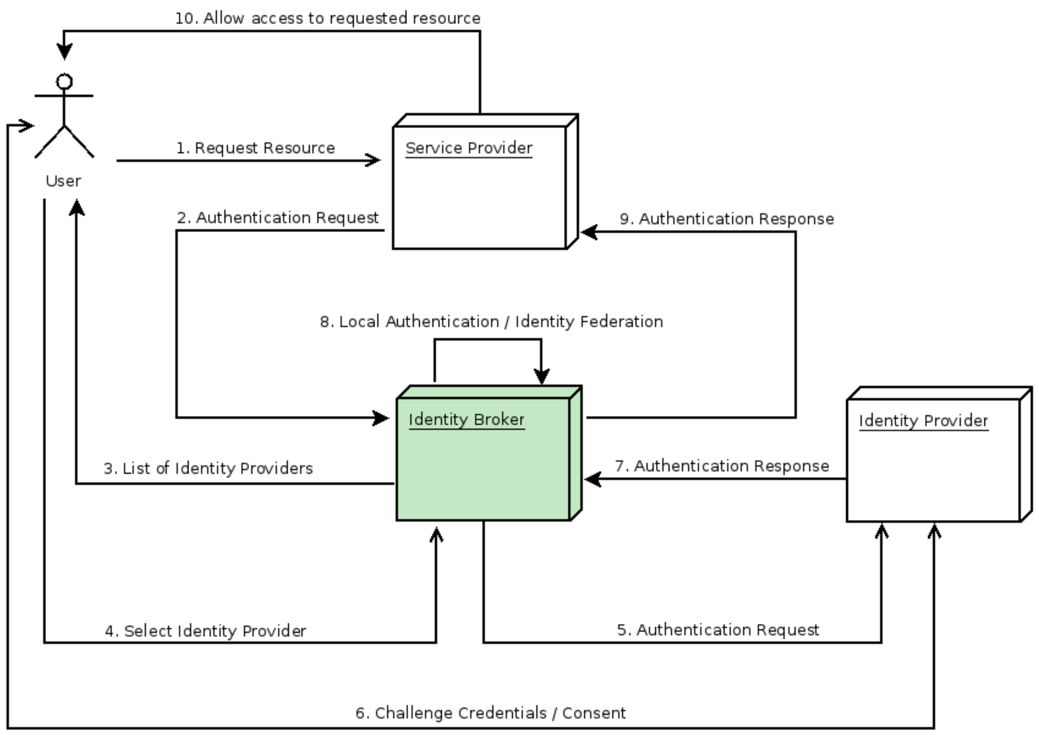

10. Allow access to requested resource
User
1. Request Resource
Service Provider
2. Authentication Request
9. Authentication Response
8. Local Authentication / Identity Federation
Identity Broker
Identity Provider
3. List of Identity Providers
7. Authentication Response
4. Select Identity Provider
5. Authentication Request
6. Challenge Credentials / Consent

Figure 9.8: Keycloak identity provider flow [21]

9.2 Sequence Diagrams

Sequence diagrams illustrate the interactions between different parts of a system in a specific order. This section shows two of the most important processes in the application where both the front- and backend are involved.

9.2.1 Visualizing and filtering utility lines

The process of dynamically loading utility lines based the device's location and filtering a specific type is illustrated in figure 9.9.

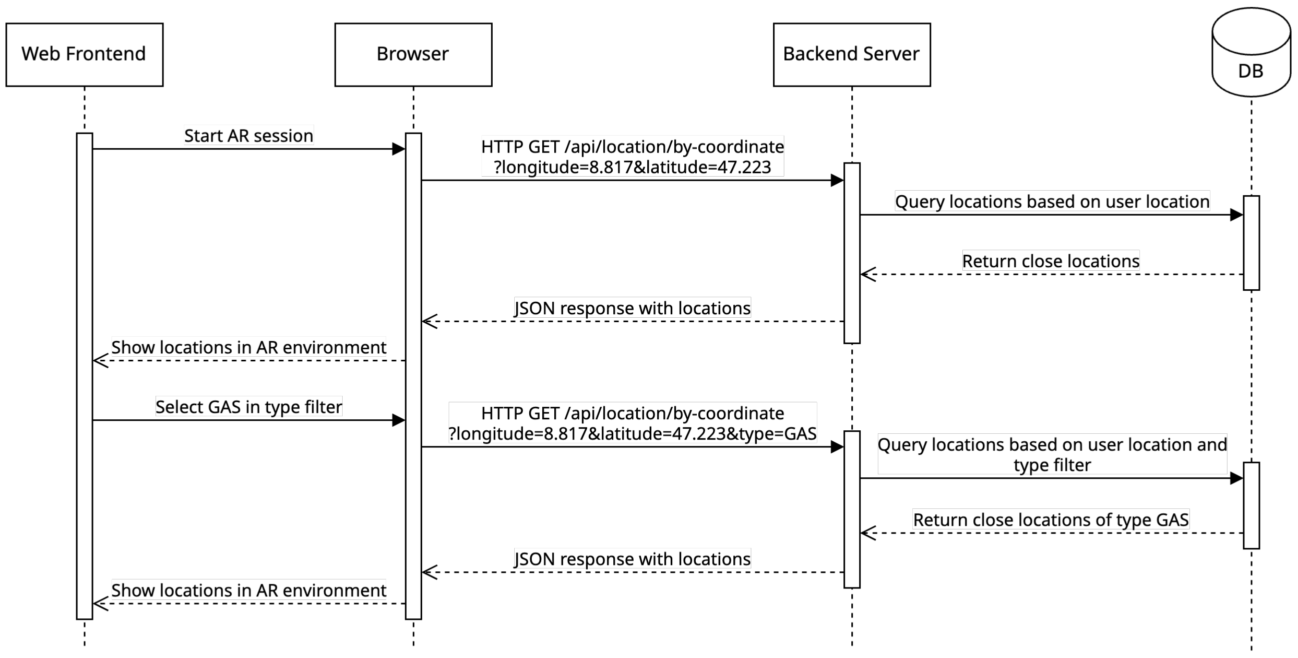

Web Frontend
Browser
Backend Server
DB
Start AR session
HTTP GET /api/location/by-coordinate ?longitude=8.817&latitude=47.223
Query locations based on user location
Return close locations
JSON response with locations
Show locations in AR environment
Select GAS in type filter
HTTP GET /api/location/by-coordinate ?longitude=8.817&latitude=47.223&type=GAS
Query locations based on user location and type filter
Return close locations of type GAS
JSON response with locations
Show locations in AR environment

Figure 9.9: Visualizing utility lines sequence diagram

9.2.2 IFC file upload

Figure 9.10 shows the process of uploading an IFC file as an authenticated user.

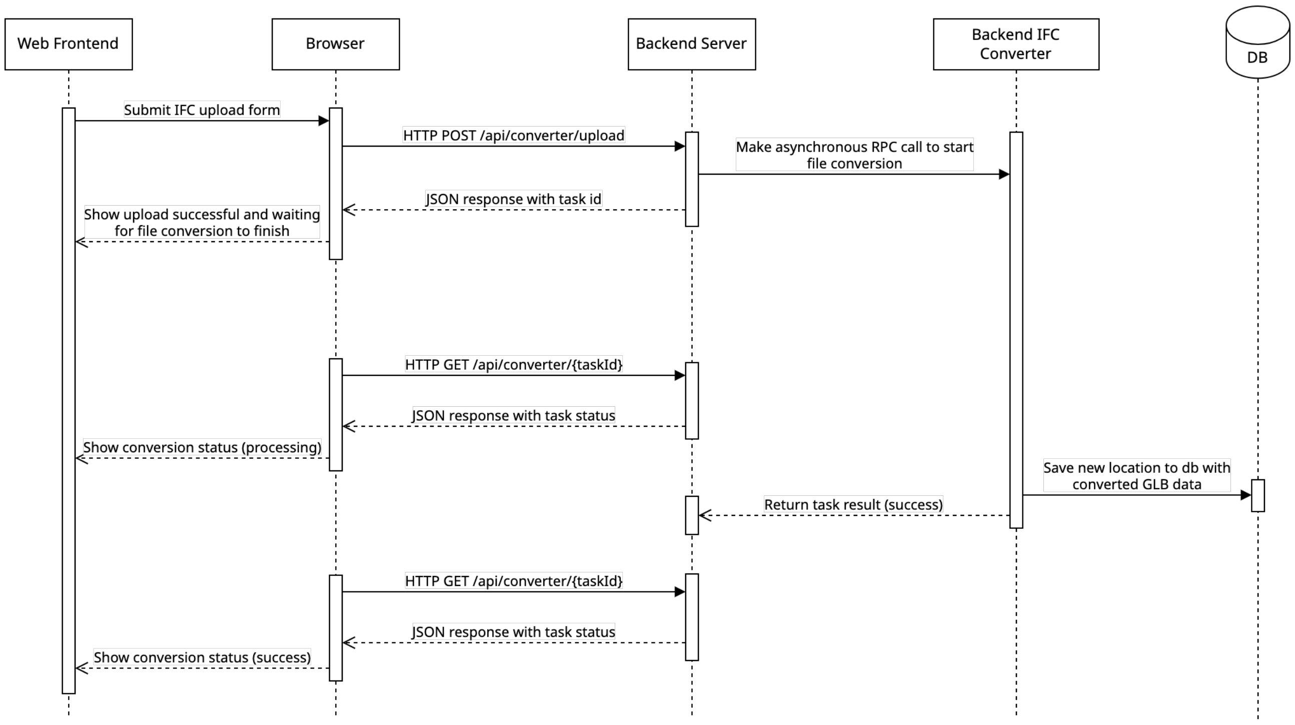

Web Frontend
Browser
Backend Server
Backend IFC Converter
DB
Submit IFC upload form
HTTP POST /api/converter/upload
Make asynchronous RPC call to start file conversion
JSON response with task id
Show upload successful and waiting for file conversion to finish
HTTP GET /api/converter/{taskId}
JSON response with task status
Show conversion status (processing)
Save new location to db with converted GLB data
Return task result (success)
HTTP GET /api/converter/{taskId}
JSON response with task status
Show conversion status (success)

Figure 9.10: IFC file upload sequence diagram

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 section 12.4.

10.3 Browser compatibility

According to the WebXR documentation, the primary mobile browsers that support WebXR are Chrome, Opera and Samsung Internet. However, testing WebXR on these browsers revealed that AR sessions only work on Chrome. Chrome utilizes ARCore to provide the AR functionality on supported Android devices. [22] [23] [24]

10.4 Challenges

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

10.4.1 Accurate model placement

Assuming the reference coordinates of the model are correct, accurate model placement depends on the geolocation accuracy of the device and the correct alignment of the virtual coordinate space with the real world. Using a GNSS antenna with integrated RTK receiver, geolocation accuracy can reach up to 1 cm under optimal conditions. Aligning the virtual space to the real world basically means aligning the Z-axis to true north. To achieve precise alignment, the team implemented manual rotation controls and a guided compass correction, which allows for more accurate alignment adjustments than using the device's compass. Despite these improvements, field-tests with the application revealed that model placement was still not perfect. To address this, the team came up with an additional feature defined by US-10, which allows users to manually correct the model's position on each axis.

10.4.2 Utility lines underground visualization

Another challenge was the visualization of utility lines, which often appeared to be above the ground rather than below it. To mitigate this effect, the utility lines are rendered slightly transparent, and the lighting was adjusted to use a directional light from above. A direct comparison is shown in figures 10.1 and 10.2 from the term thesis and the bachelor thesis, respectively. Although, the new visualization is an improvement, the lines still appear somewhat above the ground. This issue comes from the limitations of AR technology itself, which overlays virtual objects onto the camera's view.

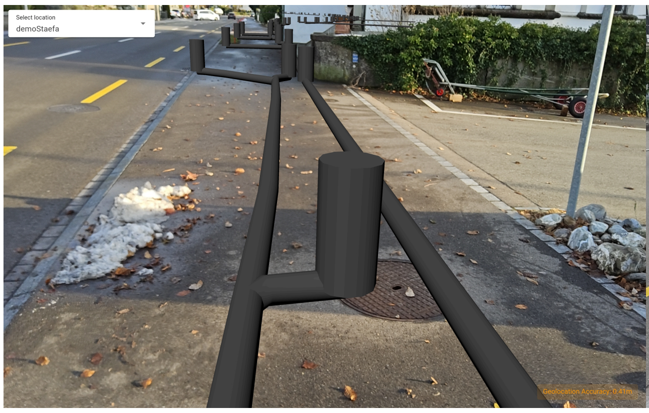


Figure 10.1: Old visualization

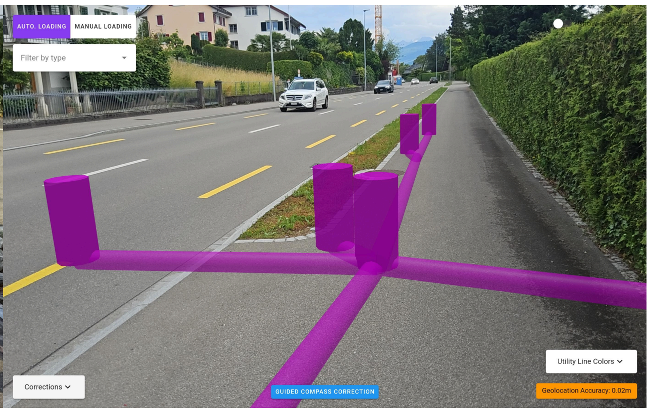


Figure 10.2: New visualization

Results & further development

In addition to chapter 4, this chapter should give a deeper insight into the results as well as the further development of the application.

11.1 Results

The user login feature was integrated into the application using Keycloak. This feature allows users to authenticate themselves using one of the social login providers Google or GitHub. In order to integrate Keycloak into the application, the frontend uses keycloak-js and the backend python-keycloak. The application also supports anonymous usage, though with limited functionality where IFC file uploads are disabled, and only public utility lines can be viewed.

Another key aspect was the ability for users to upload and visualize their own IFC data. The IFC upload feature is a 2-step process consisting of the upload itself and the conversion of IFC to GLB. The frontend indicates the status of the upload and conversion process, providing immediate feedback.

The application is able to dynamically load utility lines based on the user's location. This is made possible by the boundary which is stored in the database with every location entry. When the user is inside this boundary, the utility lines get visualized. By default, utility lines of all types are displayed. By using the filter in the overlay, users can show only specific types which they're interested in.

Another goal of the project was to allow users to configure colors for different utility line types, this aspect was not realized due to time constraints. However, the application does automatically colorize utility lines based on their type and users can view a list of all types and their corresponding colors in the overlay. These colors are currently hard-coded in the backend.

To ensure that users only see relevant parts of the utility lines, the visibility range of the camera is limited to 50 meters. Objects outside this radius are not rendered.

Individual parts of the utility line models are interactive, allowing users to click on them and display detailed information. This includes the location name, the type, the name of the selected object and the bounding box size.

Manual compass correction was implemented using a slider, enabling users to manually adjust the rotation of the virtual coordinate system to align it accurately with the real world. To further enhance the correct alignment of the virtual coordinate system, a guided compass correction using GNSS was implemented. This feature can be activated in the overlay. After starting the guided correction process, the user is instructed to walk in a straight line while also holding the tablet as straight as possible. When the user decides to finish the process, the compass direction is calculated based on the start and end position and the virtual coordinate system is adjusted accordingly. The further the user walks the more accurate this will get.

The team noticed during outdoor testing that even with these mentioned correction methods, the positioning of models isn't perfect. Therefore, an additional feature was implemented which allows users to manually move utility line models on each axis. This flexibility ensures that users can achieve precise alignment of the models. As part of this feature, a switch was also implemented which can enable or disable the automatic position correction of utility line models. This is useful to stop the model from jumping around while trying to align it to some anchor point such as a manhole.

While the application lacks some precision in displaying utility line models, it establishes a robust foundation for future enhancements and development. The backend API provides a swagger API documentation, making it easier to understand the usage of the API.

11.2 Further development

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

11.2.1 Support for older IFC versions & alternative georeferencing methods

One possible area for improvement is the support for older IFC versions such as 2.3. This is important because these versions are still quite common, as multiple individuals which the team contacted for IFC 4.3 files mentioned they only have files of version 2.3. This improvement also includes integrating alternative georeferencing methods for the models, as the currently implemented method in the application uses IFC fields which got introduced in version 4. Additionally, the team learned during this thesis that even in version 4, georeferencing is done in different ways.

11.2.2 Support for other file formats

Another significant enhancement would be the support of other formats than just IFC. This expansion would make the application more accessible to various industry standards.

11.2.3 Advanced interactive controls

Introducing more interactive controls in the viewer, such as distance measurement and data correction could considerably improve the workflow of managing utility line data. Distance measurement tools would enable users to quickly get distances and use them for planning and analysis. Data correction features would allow users to make real-time adjustments to the models, ensuring greater accuracy.

11.2.4 Utility line types color configuration

Allowing users to manually configure colors for utility line types is another potential improvement. This feature would enable users to set the colorization based on their specific standards.

11.2.5 Collaboration

Currently, users can only see their uploaded utility lines and public utility lines. Enabling collaboration by allowing users to share access to their utility line models would enable teams to work together much easier. This could greatly improve efficiency in team-based projects.

11.2.6 Improve performance

Improving the performance of loading utility line models is another important area for development. Optimizing the loading process would enhance the application's responsiveness and speed, providing a smoother and more efficient user experience. This improvement is particularly crucial for handling large files with complex models, ensuring that the application remains stable.

11.2.7 Improve underground visual effect of utility lines

As mentioned in section 10.4.2, the utility lines appear to be somewhat above the ground. Concrete ideas to minimize this visual effect would have to be evaluated. Possible approaches include using color gradients, displaying a grid on the ground or experimenting with highlights and shadows.

Quality Measures

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

12.1 Quality Assessment Tools

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

12.1.1 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 can be found on https://github.com/cmhughes/latexindent.pl. 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.

12.1.2 Linter

For automatically checking that the application's 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 TypeScript code. For integration into VS Code, the extension "ESLint" has to be installed.

Pylint

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

12.1.3 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 Jira is used and all tasks are defined via issues, the following Definition of Done for issues are defined.

- **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 TypeScript and Python files
  - Complies with defined coding standards
  - No Bugs or Code Smells
  - Build Pipeline passed
- **Documentation**
  - All necessary items have been documented

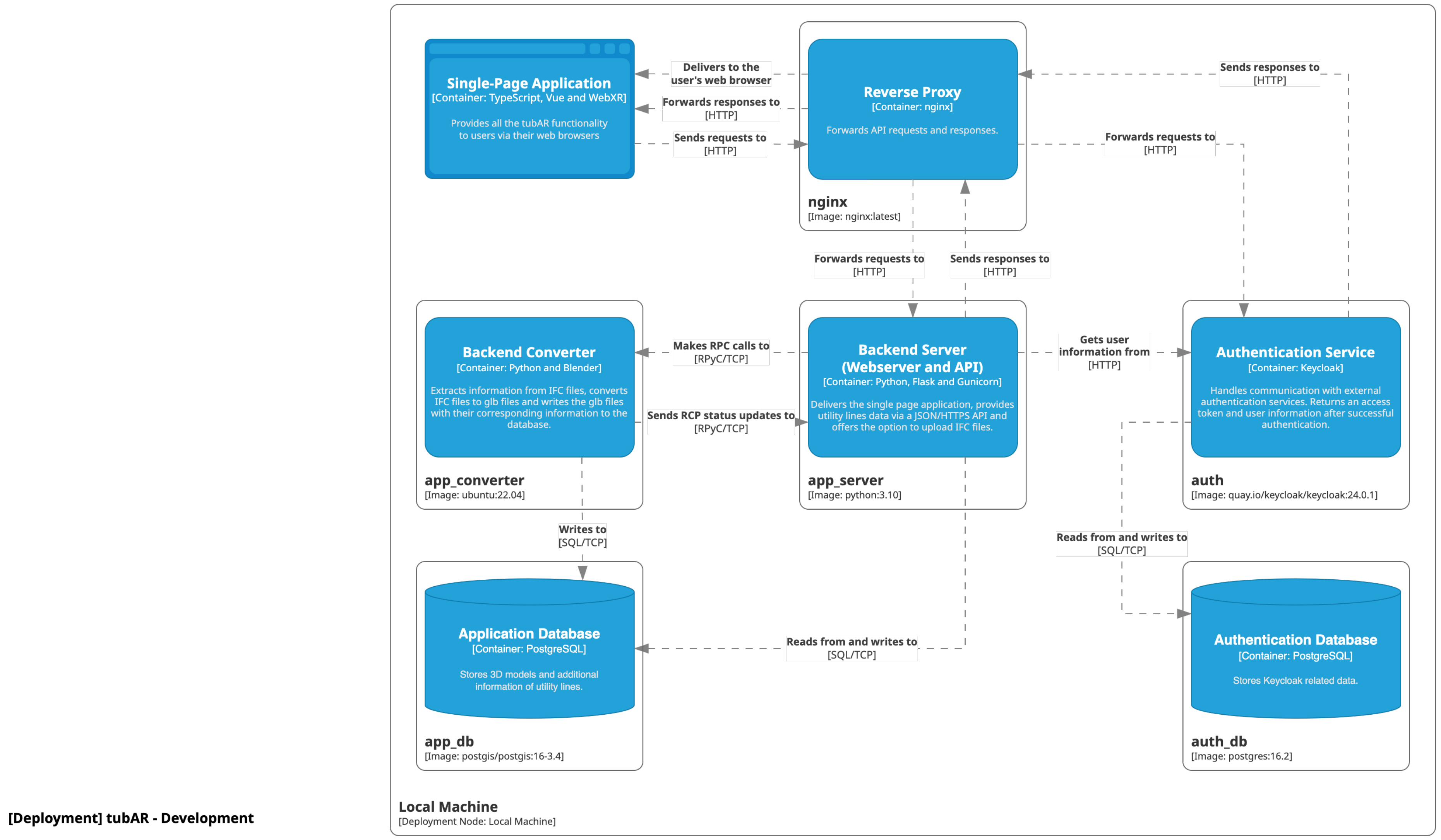
12.1.4 Git-Branching & Merges

To avoid mistakes, it's 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 team member has implemented their feature and the definition of done is fulfilled (except the peer code review criterion), they 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.

12.2 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.

12.2.1 Development



[Deployment] tubAR - Development
Single-Page Application
[Container: TypeScript, 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]
Sends requests to [HTTP]
Reverse Proxy
[Container: nginx]
Forwards API requests and responses.
nginx
[Image: nginx:latest]
Sends responses to [HTTP]
Forwards requests to [HTTP]
Forwards requests to [HTTP]
Sends responses to [HTTP]
Backend Converter
[Container: Python and Blender]
Extracts information from IFC files, converts IFC files to glb files and writes the glb files with their corresponding information to the database.
app_converter
[Image: ubuntu:22.04]
Makes RPC calls to [RPyC/TCP]
Sends RCP status updates to [RPyC/TCP]
Backend Server (Webserver and API)
[Container: Python, Flask and Gunicorn]
Delivers the single page application, provides utility lines data via a JSON/HTTPS API and offers the option to upload IFC files.
app_server
[Image: python:3.10]
Gets user information from [HTTP]
Authentication Service
[Container: Keycloak]
Handles communication with external authentication services. Returns an access token and user information after successful authentication.
auth
[Image: quay.io/keycloak/keycloak:24.0.1]
Writes to [SQL/TCP]
Reads from and writes to [SQL/TCP]
Reads from and writes to [SQL/TCP]
Application Database
[Container: PostgreSQL]
Stores 3D models and additional information of utility lines.
app_db
[Image: postgis/postgis:16-3.4]
Authentication Database
[Container: PostgreSQL]
Stores Keycloak related data.
auth_db
[Image: postgres:16.2]
Local Machine
[Deployment Node: Local Machine]

Figure 12.1: Development environment described as C4 deployment diagram

12.2.2 Production

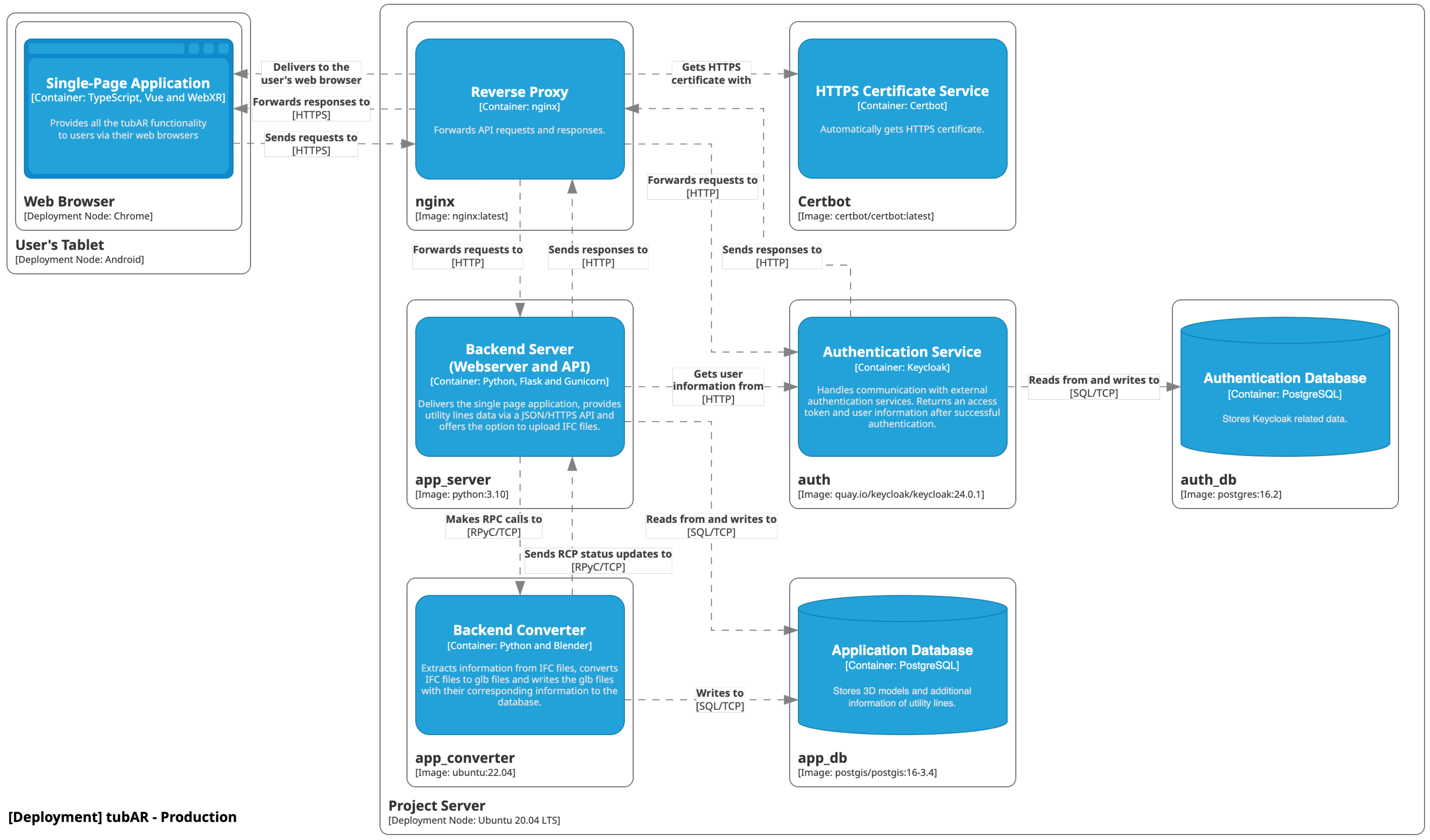

Single-Page Application
[Container: TypeScript, Vue and WebXR]
Provides all the tubAR functionality to users via their web browsers
Web Browser
[Deployment Node: Chrome]
User's Tablet
[Deployment Node: Android]
Delivers to the user's web browser
Forwards responses to [HTTPS]
Sends requests to [HTTPS]
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]
Forwards requests to [HTTP]
Sends responses to [HTTP]
Sends responses to [HTTP]
Backend Server (Webserver and API)
[Container: Python, Flask and Gunicorn]
Delivers the single page application, provides utility lines data via a JSON/HTTPS API and offers the option to upload IFC files.
app_server
[Image: python:3.10]
Gets user information from [HTTP]
Authentication Service
[Container: Keycloak]
Handles communication with external authentication services. Returns an access token and user information after successful authentication.
auth
[Image: quay.io/keycloak/keycloak:24.0.1]
Reads from and writes to [SQL/TCP]
Authentication Database
[Container: PostgreSQL]
Stores Keycloak related data.
auth_db
[Image: postgres:16.2]
Makes RPC calls to [RPyC/TCP]
Sends RCP status updates to [RPyC/TCP]
Reads from and writes to [SQL/TCP]
Backend Converter
[Container: Python and Blender]
Extracts information from IFC files, converts IFC files to glb files and writes the glb files with their corresponding information to the database.
app_converter
[Image: ubuntu:22.04]
Writes to [SQL/TCP]
Application Database
[Container: PostgreSQL]
Stores 3D models and additional information of utility lines.
app_db
[Image: postgis/postgis:16-3.4]
Project Server
[Deployment Node: Ubuntu 20.04 LTS]
[Deployment] tubAR - Production

Figure 12.2: Production environment described as C4 deployment diagram

12.3 CI/CD

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

12.3.1 Workflow

The team uses the workflow described below to achieve continuous integration and deployment. it's 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.

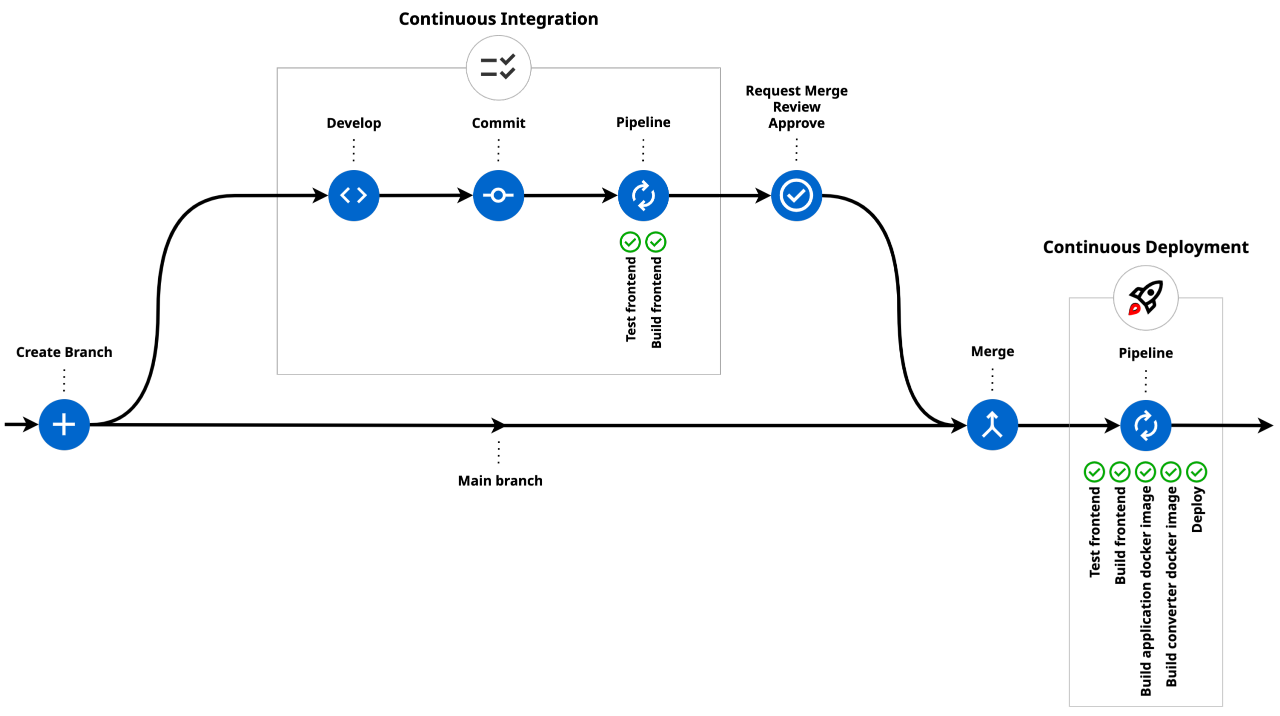
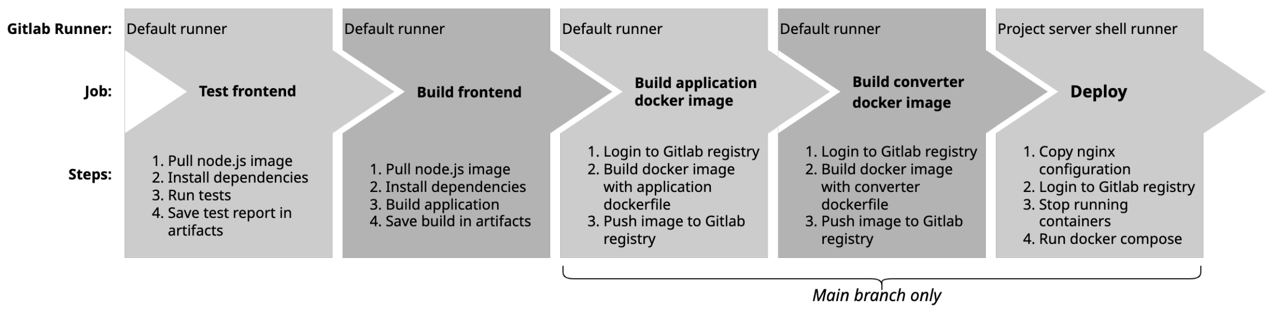


Figure 12.3: Workflow

12.3.2 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 figure 12.2.



Gitlab Runner:	Default runner	Default runner	Default runner	Default runner	Project server shell runner
Job:	**Test frontend**	**Build frontend**	**Build application docker image**	**Build converter docker image**	**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. Login to Gitlab registry 2. Build docker image with converter dockerfile 3. Push image to Gitlab registry	1. Copy nginx configuration 2. Login to Gitlab registry 3. Stop running containers 4. Run docker compose

Figure 12.4: Gitlab pipeline

12.4 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.

12.4.1 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

These tests make 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

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. Information about the event can be found on https://www.meetup.com/rapperswil-frontend-best-practices-meetup-group/events/292300468/.

Performance Testing

These tests make sure that the application behaves correctly under different circumstances.

- Execution: Manual
- Time of Execution: At the end of each sprint during the Construction phase

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

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

12.4.2 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. Not every line of code is part of the coverage report. The team excluded files that only use parts of external libraries such as Vuetify due to the fact that this logic already tested by their respective developers. Furthermore, source code which deals with complex WebXR or Three.js logic is also not included into the coverage reports, because such code units are barely testable. For the backend tests, the coverage report must be generated locally, as it requires a testing database and the effort of integrating this into the CI/CD pipeline isn't worth it at the moment.

12.4.3 Tooling

For testing the Python code, Pytest will be used. Vitest and Vue Test Utils will be used for the TypeScript code.

12.5 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.

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 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 experience in backend 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 February 20th 2024 and will end on June 14th 2024, 5:00PM with the final project submission. Each of us is required to spend approximately 24 hours per week working on this project, which makes 48 hours a week and 720 hours for the whole project. Besides the time consumed by the implementation and documentation, meetings also count as work time. All work time is tracked in the respective issues in Jira.

13.1.3 Cost

As this is a university project, there is no budget which can be expressed in terms of money. However, the costs can be expressed as the earlier mentioned 720 hours, that the team is allowed to use 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

13.1.5 Hosting of the application

Throughout the bachelor thesis, the backend of the application was running on a server provided by OST. The provided server was a virtual machine running Ubuntu 20.04 LTS. It had a 2 core CPU assigned to it and came with 4 GB of RAM and 50 GB of disk space configured. There was no dedicated GPU available. tubAR is at the time of this bacherlor thesis' hand-in available on https://srbsci-26.ost.ch/.

13.2 Responsibilities

Usually roles would be assigned in a Scrum project. The team found this to be unnecessary, because it only consists of two people. Nevertheless, both team members carry the responsibilities below:

- Check that the project plan is being followed and that the status of work is on track.
- Track work time in Jira.
- All documents are handed in before according deadlines.
- Adhere to the defined quality measures.
- Maintain the product backlog and reevaluate time estimations.
- Keep the risks up-to-date.
- Implement software / infrastructure according to project plan and product backlog.
- Define and maintain the CI/CD processes.
- Define overall structure of the system.

13.3 Processes and meetings

To achieve agile project management, a Scrum-like framework combined with RUP is used to define processes in this project.

13.3.1 Product and 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: Monday
- Total amount of sprints: 8 (first sprint is only one week)

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 team member takes care of which sprint backlog items and define time estimations.

Further information about planning:

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

Weekly (meetings)

The team does weekly meetings, so called weeklys, to catch up with each other's progress. Of course, urgent matters should be communicated immediately through the defined communication channels.

Further information about weeklys:

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

Review

Each sprint ends with 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: Monday 1:00 PM
- Duration: 30m

Retrospective

Each sprint ends with a retrospective.

Retrospective procedure: The team defined a quality measure checklist, which needs to be inspected during the sprint retrospective. If any item isn't 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?
- 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: Monday 1:30 PM
- Duration: 30m

13.3.3 Advisor meetings

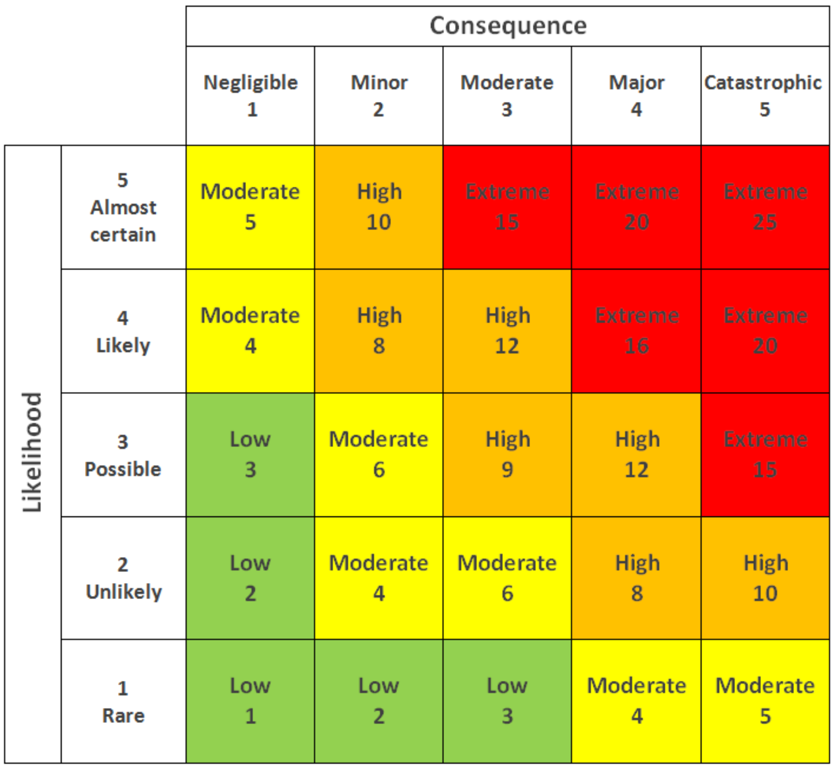
- Frequency: Once every two weeks
- Location: Room 8.261
- Day/Time: Tuesday 3:00 PM
- Duration: 1h
- Additional attendants: Prof. Stefan F. Keller

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 | 6 (Likelihood: 2, Consequence: 3)

Description

All team members have gained some experience in development of AR applications during the term project which took place in the previous semester. Still, it's quite a new field to all team members. Therefore, it's possible that some features aren't implemented optimally.

Mitigation

Every implementation of an AR feature must be planned in the best way possible in advance to balance out the lack of experience.

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

Description

Since the team members have already worked on multiple projects together, including the preceding term thesis, the risk of bad team work is small but existent.

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

it's difficult to perfectly define all requirements in a software project right from the start, especially when the team has little experience with the technology.

Mitigation

Usage of agile (Scrum-like) methodology 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 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) | 8 (Likelihood: 2, Consequence: 4)

Description

A lot of features and improvements are planned to be implemented within this bachelor thesis. It's possible that the available time is not enough to complete all requirements. Furthermore, it's possible that not every team member has the amount of time available which should be used to work on the project (24 hours per week).

Mitigation

All team members should try to estimate their available time for each sprint. If the recommended amount of time per week cannot be met, it should be discussed with the team. Also, when a feature or improvement takes a lot more time to implement than planned, it should be communicated as early as possible so that the time plan can be adjusted accordingly.

13.4.7 Testing taking up too much time | 6 (Likelihood: 2, 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 application with real data, it's required to physically go to the respective coordinates.

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 complexity | 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. Hence, it's necessary that the IFC data is converted into a 3D format first.

Mitigation

Blender with the BlenderBIM add-on is used to convert the IFC files to glTF files. Furthermore, Blender offers an API which can be used through Python scripts. This helps simplifying the task by automating the conversion of the IFC files. Though, it's still time consuming and not our primary task of this bachelor thesis.

13.4.10 Inconsistent IFC data structure | 8 (Likelihood: 4, Consequence: 2)

Description

Inconsistencies in IFC files, such as varying fields for utility lines and missing reference coordinates. It's also not clear which fields in an IFC file are usually used for the type of utility line, due to little IFC files at disposal.

Mitigation

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

13.4.11 Too few IFC test data | 12 (Likelihood: 4, Consequence: 3)

Description

The team has only access to a handful of IFC test files. This leads to potential problems developing the application because edge cases or other things related to IFC files could be overlooked. Additionally, with such a small amount of IFC test files manual application tests are difficult to conduct properly.

Mitigation

More IFC data should be gathered to be able to check whether the application can handle most files correctly.

13.4.12 Changes History

Risk	**Value change**	**Reason for change**	**Date**
Inconsistent IFC data structure	$12 \rightarrow 20$	After analyzing the currently available IFC files, it turned out that only two files can be used to develop and test the application effectively. That's too little, hence the likelihood rises.	18.03.2024
Inconsistent IFC data structure	$20 \rightarrow 16$	By sticking to the standards the present IFC inconsistencies can be handled. Hence, if an IFC file doesn't comply to the standards tubAR can't smoothly process the data.	01.04.2023
Inconsistent IFC data structure	$16 \rightarrow 8$	There are a lot of inconsistent or missing parts within IFC files, but tubAR will just work with what's defined in the standard for IFC 4.3. Therefore, the consequence can be lowered.	16.04.2023
Testing taking up too much time	$9 \rightarrow 6$	Most of the defined user stories are now fulfilled which means there is more time to test the application than expected.	29.04.2023

Table 13.1: Risk changes history

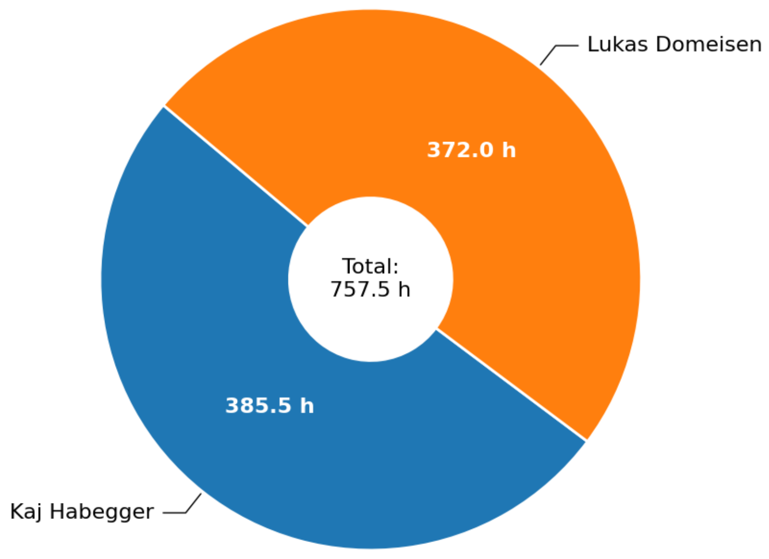
13.5 Long-term Plan

Timeplan		Sprint	Sprint 7	Sprint 8		Sprint 9		Sprint 10		Sprint 11		Sprint 12		Sprint 13		Sprint 14	
		SW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Inception	Prepare basic documentation structure	*Estimate*	🟨														
		Actual	🟩														
	Setup issue tracking / time tracking (Jira)	*Estimate*	🟨														
		Actual	🟩														
	Create long term plan	*Estimate*	🟨														
		Actual	🟩														
	Reconfigure SA infrastructure to BA requirements	*Estimate*	🟨														
		Actual	🟩														
	Define roles, processes, resources and meetings	*Estimate*	🟨														
		Actual	🟩														
	Risk-Management	*Estimate*	🟨														
		Actual	🟩														
Elaboration	Requirement-Analysis	*Estimate*	🟨	🟨													
		Actual	🟩														
	Evaluate user authentication services	*Estimate*		🟨	🟨												
		Actual		🟩	🟩												
	Reevaluate architecture and design	*Estimate*		🟨	🟨												
		Actual		🟩	🟩												
	Analyze IFC data for new requirements	*Estimate*		🟨	🟨												
		Actual		🟩	🟩												
Construction	IFC converter (Improve conversion, run on backend server)	*Estimate*				🟨	🟨	🟨	🟨								
		Actual			🟩			🟩	🟩	🟩							
	Backend (user authentication, database adjustments, file upload)	*Estimate*						🟨	🟨	🟨	🟨	🟨	🟨				
		Actual				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
	Frontend (user authentication, UI adjustments)	*Estimate*						🟨	🟨	🟨	🟨	🟨	🟨				
		Actual				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
Transition	Define and execute manual tests	*Estimate*												🟨	🟨		
		Actual										🟩		🟩	🟩		
	Fix bugs and re-test	*Estimate*													🟨	🟨	
		Actual												🟩	🟩	🟩	
	Project reflection	*Estimate*														🟨	
		Actual														🟩	🟩
	Finalize documentation	*Estimate*															🟨
		Actual												🟩	🟩	🟩	🟩
Legend	🟨 Estimate 🟩 Actual	SW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Sprint	Sprint 7	Sprint 8		Sprint 9		Sprint 10		Sprint 11		Sprint 12		Sprint 13		Sprint 14	

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 result was calculated for the Python code: `Your code has been rated at 7.94/10`. It's important to note that some linter warnings, which were identified as false positives, have been ignored. Most of the linting issues that remain in the code are too long lines, which come from the swagger API models and the converter CLI arguments definitions. The team decided to leave these lines unchanged, as they're easier to read in their current format.

14.2.2 Frontend

Eslint has been configured specifically to adhere to the recommended rules for TypeScript and Vue.js 3. The frontend codebase does not contain any linting issues.

Software Documentation

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

15.1 Technology-stack

Technology	Version
Axios	1.6.8
Keycloak JS	24.0.2
Pinia	2.1.7
three.js	0.158.0
Turf.js	6.5.0
TypeScript	5.2.2
Vite	4.5.0
Vitest	0.34.6
Vue.js	3.4.26
Vuetify	3.3.23
WebXR API	N/A

Table 15.1: Technologies frontend

Technology	Version
Python	3.10
Alembic	1.12.1
Certbot	2.10.0
Click	8.1.7
Flask	3.0.0
Flask-SQLAlchemy	3.1.1
Flask-restx	1.3.0
GeoAlchemy2	0.14.2
Gunicorn	21.2.0
Keycloak	24.0.1
nginx	1.25.4
Pytest	7.4.3
SQLAlchemy	2.0.29
RPyC	6.0.0
PostgreSQL	16.2
PostGIS	3.4.2
Shapely	2.0.4

Table 15.2: Technologies backend

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
Visual Studio Code	Code Editor	1.90.0
pgAdmin 4	Database management	7.0

15.3 Installation

15.3.1 Real-Time Kinematic positioning (RTK) receiver

Utility lines need to be displayed precisely in the AR environment. Therefore, it's necessary that the device which is running the AR application can access precise localization. As common mobile devices can only deliver a location accuracy of approximately five meters, it's mandatory to use an external RTK capable receiver.

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 [25].

With both the satellites and the base station combined it's possible to get a location with a preciseness of up to less than a centimeter. The RTK receiver used for this project is ArduSimple's RTK Handheld Surveyor Kit [3].

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. Ensure that the kit's 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 tubAR, 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 available Mount Points hosted by rtk2go, a list can be found on:
http://www.rtk2go.com:2101/SNIP::STATUS
Especially useful if a Mount Point outside of Switzerland is needed.

15.4 Upload of IFC files

To upload IFC files via tubAR, the following requirements have to be met:

- User is logged in.
- IFC file is of version 4 or higher.
- IFC file size is less than 20 MB.
- Reference coordinates either defined through the form or within the IFC file.
- A name for the uploaded data.

The utility line type is optional. Though, when the utility line type is neither given by the form nor by the IFC file, it will be stored as undefined and therefore displayed black in the viewer.

After the upload, one of the following messages is displayed:

- Converter run was successful
- tubAR only supports IFC files using version 4 or above
- Conversion from IFC to GLB failed
- Reference coordinates were neither given by user nor by IFC file
- Saving the utility line data in database failed

Part III

Appendix

Appendix A: Deliverables

Documentation

There exists a printed and a digital version (PDF) of this document. The printed version and the digital version are handed in to Prof. Stefan Keller. The digital version is also submitted to the AVT tool of the OST. All versions were handed in on time by the team.

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.

Poster for bachelor exhibition

The poster for the bachelor exhibition must be ready at the time of the exhibition build up. This poster is demanded by the OST and was handed in two days before exhibition for printing.

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.

Source code repositories

- Application: https://gitlab.ost.ch/ba-ar-werkleitungen/ba-application
- Documentation: https://gitlab.ost.ch/ba-ar-werkleitungen/ba-documentation

Only Kaj Habegger, Lukas Domeisen, Prof. Stefan Keller, Thomas Bocek, Claude Eisenhut and Reto Senn can currently access the repositories.

The application's backend is only tested to run on macOS and Ubuntu. Furthermore, it only runs seamlessly on x86 64-bit based systems.

Application

At the time of hand-in, the application is publicly accessible via https://srbsci-26.ost.ch/.

Appendix B: Glossary and list of abbreviations

Term	Description
BIM	Abbreviation for Building Information Modeling. it's the process of modeling physical places in a digital manner. BIM data is a file or multiple files that consist of such data.
Esri ArcGIS	ArcGIS is a family of client, server and online geographic information system (GIS) software developed and maintained by Esri [26].
FR	Abbreviation for Functional Requirement. Specifies a specific feature or functionality a software solution must provide.
GLB	Abbreviation for binary file format of glTF data.
glTF	Graphics Library Transmission Format is a 3D scene and model file format.
GIS	Abbreviation for Geographic Information System. Further information about the definition can be found on https://www.esri.com/en-us/what-is-gis/overview.
GNSS	Abbreviation for Global Navigation Satellite System. Refers to globally available systems that use satellites for positioning. The most prominent is probably GPS.
IAR	Abbreviation for Integer Ambiguity Resolution. The openrtk documentation gives further information about this [27].
IFC	Abbreviation for Industry Foundation Classes. it's a data schema to exchange CAD data, or more specifically BIM data. it's descriptive only and does not contain 3D data directly. The Wikipedia entry gives further information about the schema [28].
KasmVNC	KasmVNC provides remote web-based access to a Desktop or application [29].
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 [30].
MVC	Model-View-Controller is a pattern used for software architecture.

MVVM	Model-View-View Model is a pattern used for software architecture.
NFR	Abbreviation for Non-Functional Requirement. Specifies a characteristic or constraint that a software solution must satisfy.
OAuth	OAuth, also known as OAuth 2.0 and stands for ”Open Authorization”, is a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user [31]. It basically allows logging in to tubAR with social logins, such as Google and GitHub.
RPC	Remote Procedure Call (RPC) is a protocol that enables a program to execute a procedure or subroutine on a remote server as if it were a local call, abstracting the complexities of the network communication.
RTK	Abbreviation for Real-Time Kinematic positioning. The article from u-blox gives a clear understanding of this technology [3].
RUP	Abbreviation for Rational Unified Process. it's a software development framework using iterations as its base concept. The Wikipedia entry gives further information about the framework [32].
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 which can be found on https://www.scrum.org/.
tubAR	This is the name of the application which was further developed within this thesis.
UTM	Universal Transverse Mercator (UTM) is a map projection system for assigning coordinates to locations on the surface of the earth [33]. Due to its grid-based system, it allows calculating differences of two coordinates in meters with relatively small distortions.
Vuetify	A component framework for Vue.js, which simplifies building user interfaces.
WebXR	WebXR, also known as WebXR device API, is part of the web standard. It provides access to input and output capabilities commonly associated with AR devices. Hence, WebXR enables AR applications on the web by allowing pages to detect if AR capabilities are available, querying these capabilities and much more. [34]
WGS 84	World Geodetic System, whereas version 84 is the current version.

Table 17.1: Glossary

Appendix C: Test protocols

In this appendix all application testing protocols are shown.

Acceptance tests

Functional requirements

Test-Nr.	FR	Preconditions	Description	Steps	Expected Result	Actual Result	Status
AT-1	US-1		User can register using his Google account	1. Open login page: https://srbsci-26.ost.ch/login 2. Login with Google account	Correct name is displayed on the home page	Correct name is displayed on the home page	Succeeded
AT-2	US-1		User can register using his GitHub account	1. Open login page: https://srbsci-26.ost.ch/login 2. Login with GitHub account	Correct name is displayed on the home page	Correct name is displayed on the home page	Succeeded
AT-3	US-1	- User has registered using his Google account - User has already uploaded an IFC file	The user has to authenticate themself to gain access to their data (Uploaded IFC files).	1. Open login page: https://srbsci-26.ost.ch/login 2. Login with Google account 3. Start viewer, change to manual location selection 4. Open location selection	Location selection contains location uploaded by the user	Location selection contains location uploaded by the user	Succeeded
AT-4	US-1	- User has registered using his GitHub account - User has already uploaded an IFC file	The user has to authenticate themself to gain access to their data (Uploaded IFC files).	1. Open login page: https://srbsci-26.ost.ch/login 2. Login with GitHub account 3. Start viewer, change to manual location selection 4. Open location selection	Location selection contains location uploaded by the user	Location selection contains location uploaded by the user	Succeeded
AT-5	US-1	- User has registered using his Google or GitHub account - User has already uploaded an IFC file	The user has to authenticate themself to gain access to their data (Uploaded IFC files).	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer, change to manual location selection 3. Open location selection	Location selection doesn't contain location uploaded by the user	Location selection doesn't contain location uploaded by the user	Succeeded
AT-6	US-2		User has to be authenticated to upload IFC data	1. Open home page: https://srbsci-26.ost.ch	"Upload IFC" button is not visible	"Upload IFC" button is not visible	Succeeded
AT-7	US-2		Authenticated user can upload IFC data	1. Open home page: https://srbsci-26.ost.ch 2. Click "Upload IFC" button 3. Fill out the form including all optional fields and upload a IFC 4.3 file 4. Submit form	The status page show that the IFC file was uploaded and processed successfully	The status page show that the IFC file was uploaded and processed successfully	Succeeded
AT-8	US-2		Can't upload IFC file with version below 4	1. Open home page: https://srbsci-26.ost.ch 2. Click "Upload IFC" button 3. Fill out the form including all optional fields and upload a IFC file below version 4 4. Submit form	The status page shows an error message that only IFC version 4+ is supported	The status page shows an error message that only IFC version 4+ is supported	Succeeded
AT-9	US-2		Reference coordinates of model must be defined in the file itself or in the upload form	1. Open home page: https://srbsci-26.ost.ch 2. Click "Upload IFC" button 3. Fill out the form without custom reference coordinates and upload a IFC file that doesn't contain reference coordinates 4. Submit form	The status page shows an error message that no reference coordinates are given	The status page shows an error message that no reference coordinates are given	Succeeded
AT-10	US-3	- User has already uploaded an IFC file which contains utility lines	User can only see utility lines within a radius 50 meters	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Change to manual location selection 4. Select location 5. Stand directly on a utility line 6. Walk away in a straight line	After walking about 50 meters away from the utility line, it's not visible anymore	After walking about 50 meters away from the utility line, it's not visible anymore	Succeeded
AT-11	US-4	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	Utility lines are automatically loaded and displayed based on the user's position	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Stay within the utility lines boundary	The utility lines are visible in the viewer	The utility lines are visible in the viewer	Succeeded
AT-12	US-4	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	Utility lines are automatically loaded and displayed based on the user's position	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Stay outside the utility lines boundary	The utility lines are not visible in the viewer	The utility lines are not visible in the viewer	Succeeded

AT-13	US-5	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines - The utility line type is set to gas	Utility lines are colorized based on their type	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer	The utility lines are yellow	The utility lines are yellow	Succeeded
AT-14	US-5		User can set cusotm colors for utility line types				Not implemented
AT-15	US-6	- User has already uploaded two IFC files which contains utility lines - The user is located at the utility lines - One utility line type is set to gas and the other to water	User can filter utility lines based on their type	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Change the type filter to gas only	The water utility line types are not visible anymore	The water utility line types are not visible anymore	Succeeded
AT-16	US-7	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	User can click on utility lines to see their details	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Click on a part of the utility lines	The overlay displays details of the selected part	The overlay displays details of the selected part	Succeeded
AT-17	US-8	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	User can manually correct the rotation of the utility lines	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Click on corrections in the bottom left of the overlay 4. Use the slider and the buttons for the rotation	The model is rotated	The model is rotated	Succeeded
AT-18	US-9	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	User can automatically calibrate the rotation to the correct compass direction using GNSS	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Click on the "Guided compass correction" button 4. Follow the instructions	The model is correctly aligned to north	The model is correctly aligned to north	Succeeded
AT-19	US-10	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	User can manually correct the position of the utility lines	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Click on corrections in the bottom left of the overlay 4. Use the sliders to change the position of the utility lines	The position of the utility lines is adjusted	The position of the utility lines is adjusted	Succeeded
AT-20	US-10	- User has already uploaded an IFC file which contains utility lines - The user is located at the utility lines	User can disable automatic positioning of the utility lines	1. Open home page: https://srbsci-26.ost.ch 2. Start viewer 3. Click on corrections in the bottom left of the overlay 4. Disable the switch "Automatic correction"	The utility lines position does not get corrected when the user moves around	The utility lines position does not get corrected when the user moves around	Succeeded

Non-functional requirements

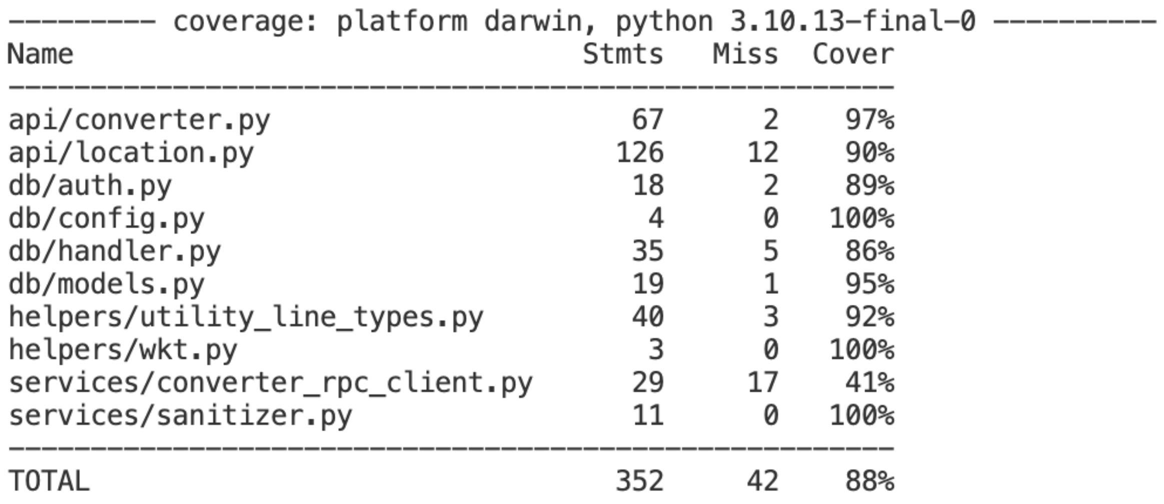
Test-Nr.	NFR	Description	Steps	Expected Result	Actual Result	Status
AT-21	NFR-4	Google Chrome major version 125 and newer is supported	Install Google Chrome with major version 125, go trough acceptance tests and check the results	The results of the acceptance tests are the same	The results of the acceptance tests are the same	Succeeded
AT-22	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-23	NFR-6	OAuth is used for authentication	Check Keycloak configuration	OAuth services are configured	OAuth services are configured	Succeeded
AT-24	NFR-7	User actions in the backend are logged	Check code for user actions (file upload)	Actions are logged with user id	Actions are logged with user id	Succeeded
AT-25	NFR-8	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-26	NFR-9	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	Succeeded
AT-27	NFR-9	Test coverage for backend server is at least 80%	Run backend server tests and generate coverage report	Backend server tests have a coverage of 80% or more	Coverage is 88%	Succeeded
AT-28	NFR-9	Test coverage for backend converter is at least 80%	Run backend converter tests and generate coverage report	Backend converter tests have a coverage of 80% or more	Coverage is 90%	Succeeded
AT-29	NFR-10	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-30	NFR-11	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

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	Average RAM usage is 500MB and maximum RAM usage is 1.5GB	Succeeded

Unit tests

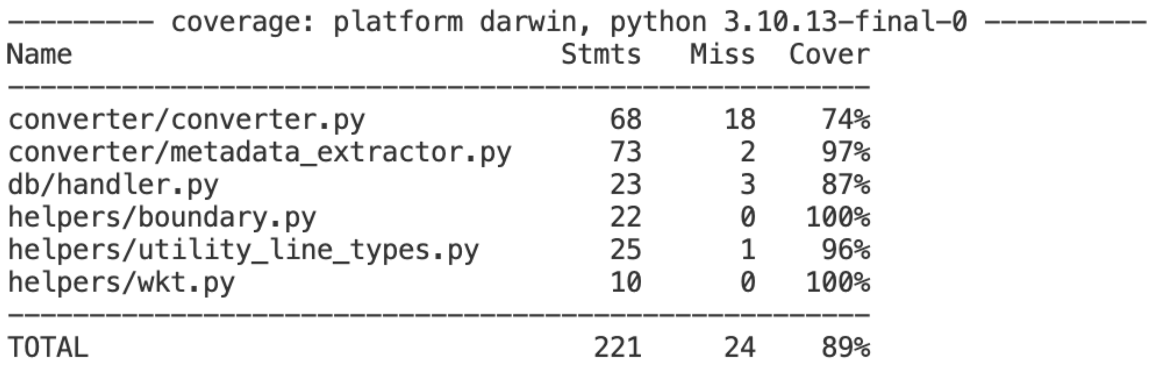
Backend - server



```
--------- coverage: platform darwin, python 3.10.13-final-0 ----------
Name                                 Stmts   Miss  Cover
--------------------------------------------------------
api/converter.py                        67      2    97%
api/location.py                        126     12    90%
db/auth.py                              18      2    89%
db/config.py                             4      0   100%
db/handler.py                           35      5    86%
db/models.py                            19      1    95%
helpers/utility_line_types.py           40      3    92%
helpers/wkt.py                           3      0   100%
services/converter_rpc_client.py        29     17    41%
services/sanitizer.py                   11      0   100%
--------------------------------------------------------
TOTAL                                  352     42    88%
```

Figure 18.1: Backend server coverage report

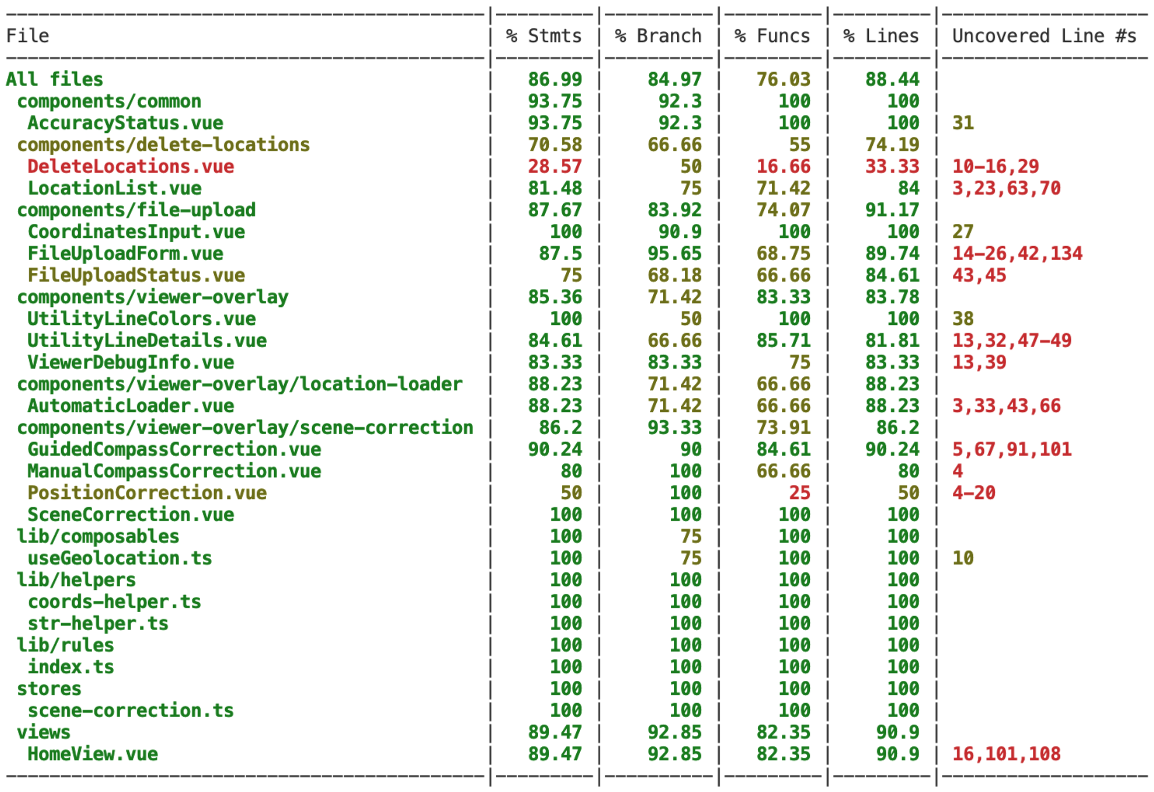
Backend - IFC converter



```
--------- coverage: platform darwin, python 3.10.13-final-0 ----------
Name                                Stmts   Miss  Cover
-------------------------------------------------------
converter/converter.py                 68     18    74%
converter/metadata_extractor.py        73      2    97%
db/handler.py                          23      3    87%
helpers/boundary.py                    22      0   100%
helpers/utility_line_types.py          25      1    96%
helpers/wkt.py                         10      0   100%
-------------------------------------------------------
TOTAL                                 221     24    89%
```

Figure 18.2: Backend IFC converter coverage report

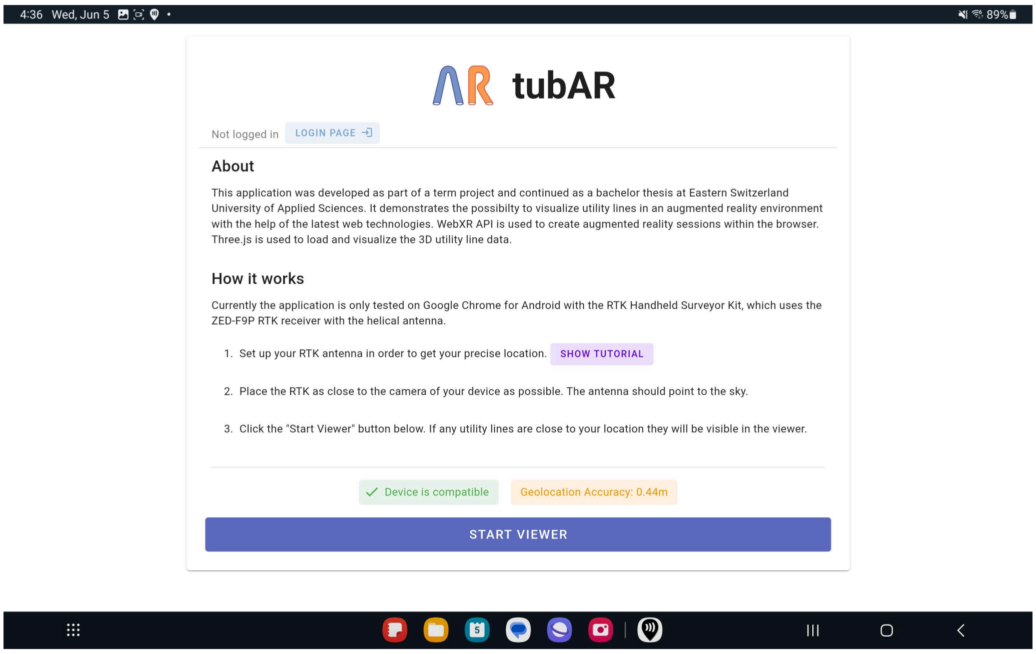
Frontend



```
--------------------------------------------|---------|----------|---------|---------|-------------------
File                                        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------------------------------|---------|----------|---------|---------|-------------------
All files                                   |   86.99 |    84.97 |   76.03 |   88.44 |
 components/common                          |   93.75 |     92.3 |     100 |     100 |
  AccuracyStatus.vue                        |   93.75 |     92.3 |     100 |     100 | 31
 components/delete-locations                |   70.58 |    66.66 |      55 |   74.19 |
  DeleteLocations.vue                       |   28.57 |       50 |   16.66 |   33.33 | 10-16,29
  LocationList.vue                          |   81.48 |       75 |   71.42 |      84 | 3,23,63,70
 components/file-upload                     |   87.67 |    83.92 |   74.07 |   91.17 |
  CoordinatesInput.vue                      |     100 |     90.9 |     100 |     100 | 27
  FileUploadForm.vue                        |    87.5 |    95.65 |   68.75 |   89.74 | 14-26,42,134
  FileUploadStatus.vue                      |      75 |    68.18 |   66.66 |   84.61 | 43,45
 components/viewer-overlay                  |   85.36 |    71.42 |   83.33 |   83.78 |
  UtilityLineColors.vue                     |     100 |       50 |     100 |     100 | 38
  UtilityLineDetails.vue                    |   84.61 |    66.66 |   85.71 |   81.81 | 13,32,47-49
  ViewerDebugInfo.vue                       |   83.33 |    83.33 |      75 |   83.33 | 13,39
 components/viewer-overlay/location-loader  |   88.23 |    71.42 |   66.66 |   88.23 |
  AutomaticLoader.vue                       |   88.23 |    71.42 |   66.66 |   88.23 | 3,33,43,66
 components/viewer-overlay/scene-correction |    86.2 |    93.33 |   73.91 |    86.2 |
  GuidedCompassCorrection.vue               |   90.24 |       90 |   84.61 |   90.24 | 5,67,91,101
  ManualCompassCorrection.vue               |      80 |      100 |   66.66 |      80 | 4
  PositionCorrection.vue                    |      50 |      100 |      25 |      50 | 4-20
  SceneCorrection.vue                       |     100 |      100 |     100 |     100 |
 lib/composables                            |     100 |       75 |     100 |     100 |
  useGeolocation.ts                         |     100 |       75 |     100 |     100 | 10
 lib/helpers                                |     100 |      100 |     100 |     100 |
  coords-helper.ts                          |     100 |      100 |     100 |     100 |
  str-helper.ts                             |     100 |      100 |     100 |     100 |
 lib/rules                                  |     100 |      100 |     100 |     100 |
  index.ts                                  |     100 |      100 |     100 |     100 |
 stores                                     |     100 |      100 |     100 |     100 |
  scene-correction.ts                       |     100 |      100 |     100 |     100 |
 views                                      |   89.47 |    92.85 |   82.35 |    90.9 |
  HomeView.vue                              |   89.47 |    92.85 |   82.35 |    90.9 | 16,101,108
--------------------------------------------|---------|----------|---------|---------|-------------------
```

Figure 18.3: Frontend coverage report

Appendix D: Screenshots



4:36 Wed, Jun 5

89%

tubAR

Not logged in

About

This application was developed as part of a term project and continued as a bachelor thesis at Eastern Switzerland University of Applied Sciences. It demonstrates the possibilty to visualize utility lines in an augmented reality environment with the help of the latest 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

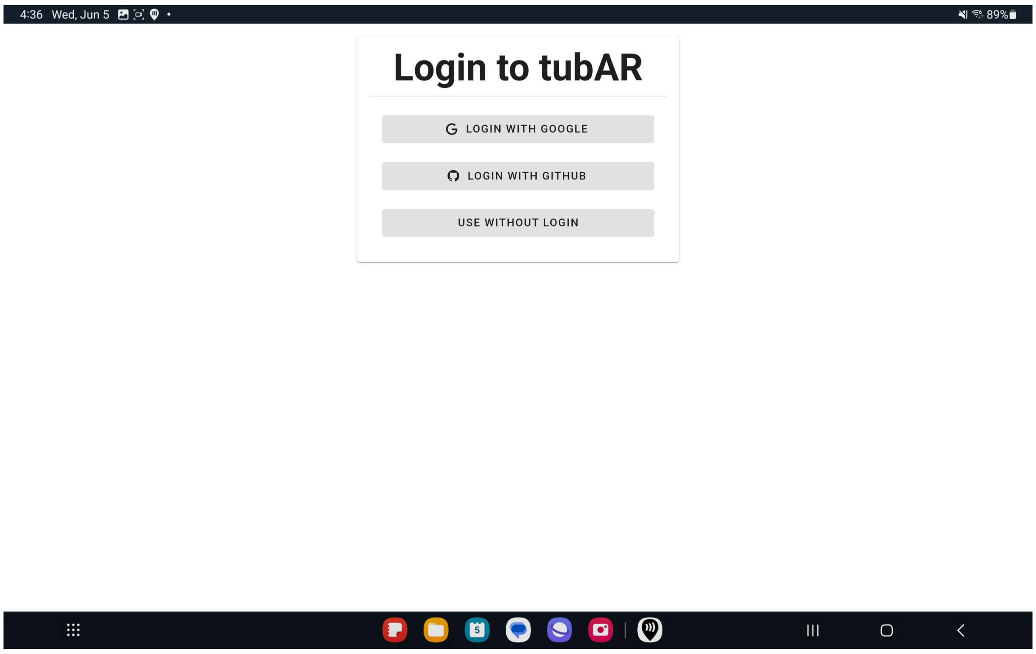
Currently the application is only tested on Google Chrome for Android with the RTK Handheld Surveyor Kit, which uses the ZED-F9P RTK receiver with the helical antenna.

1. Set up your RTK antenna in order to get your precise location. 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 the "Start Viewer" button below. If any utility lines are close to your location they will be visible in the viewer.

Device is compatible

Geolocation Accuracy: 0.44m

Figure 19.1: Screenshot showing the home view of tubAR

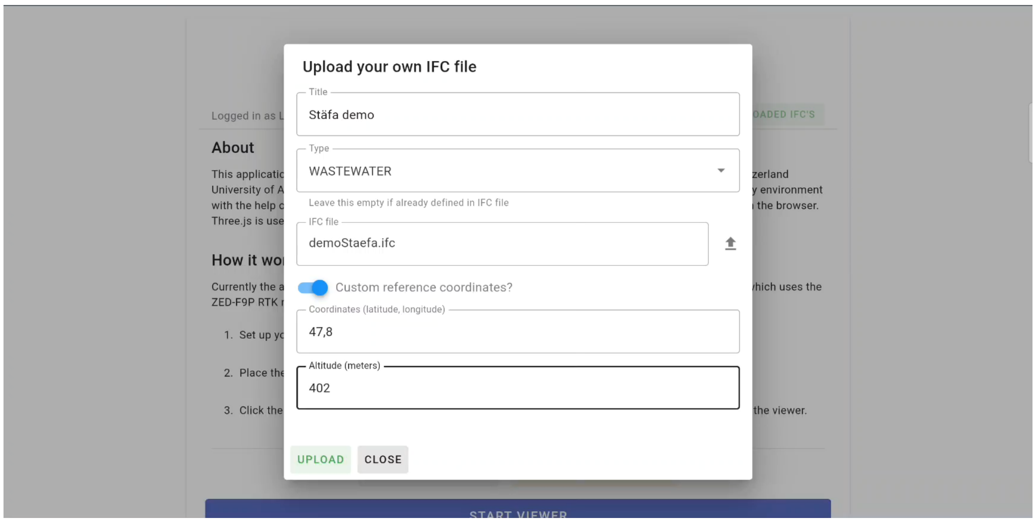


4:36 Wed, Jun 5

89%

Login to tubAR

Figure 19.2: Screenshot showing the login view of tubAR


Upload your own IFC file

Title: Stäfa demo

Type: WASTEWATER

Leave this empty if already defined in IFC file

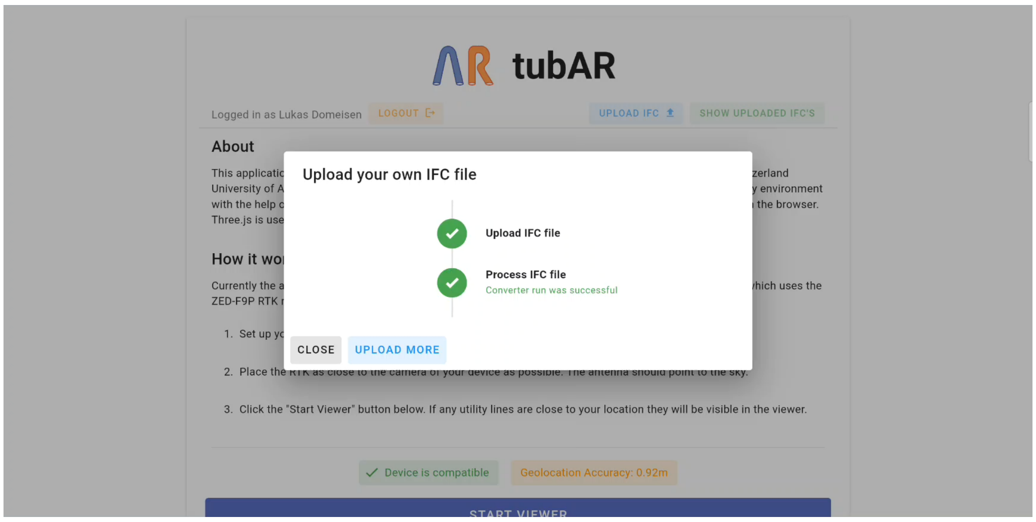
IFC file: demoStaefa.ifc

Custom reference coordinates?

Coordinates (latitude, longitude): 47,8

Altitude (meters): 402

Figure 19.3: Screenshot showing the upload form of tubAR


Upload your own IFC file

Upload IFC file

Process IFC file

Converter run was successful

Logged in as Lukas Domeisen

2. Place the RTK as close to the camera of your device as possible. The antenna should point to the sky.

3. Click the "Start Viewer" button below. If any utility lines are close to your location they will be visible in the viewer.

Device is compatible

Geolocation Accuracy: 0.92m

Figure 19.4: Screenshot showing the upload and conversion status of tubAR

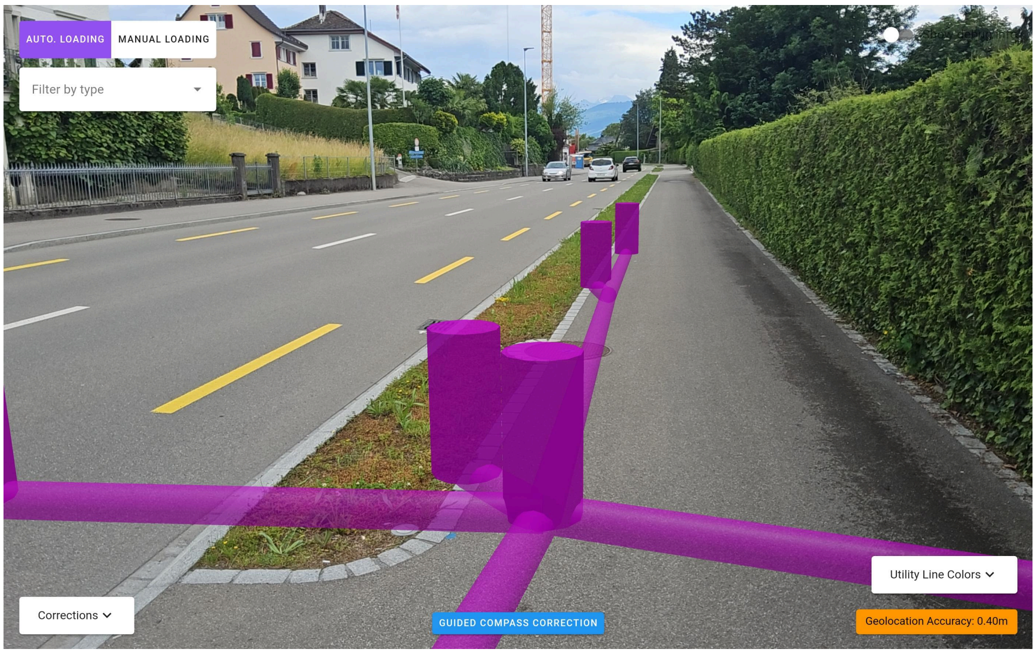
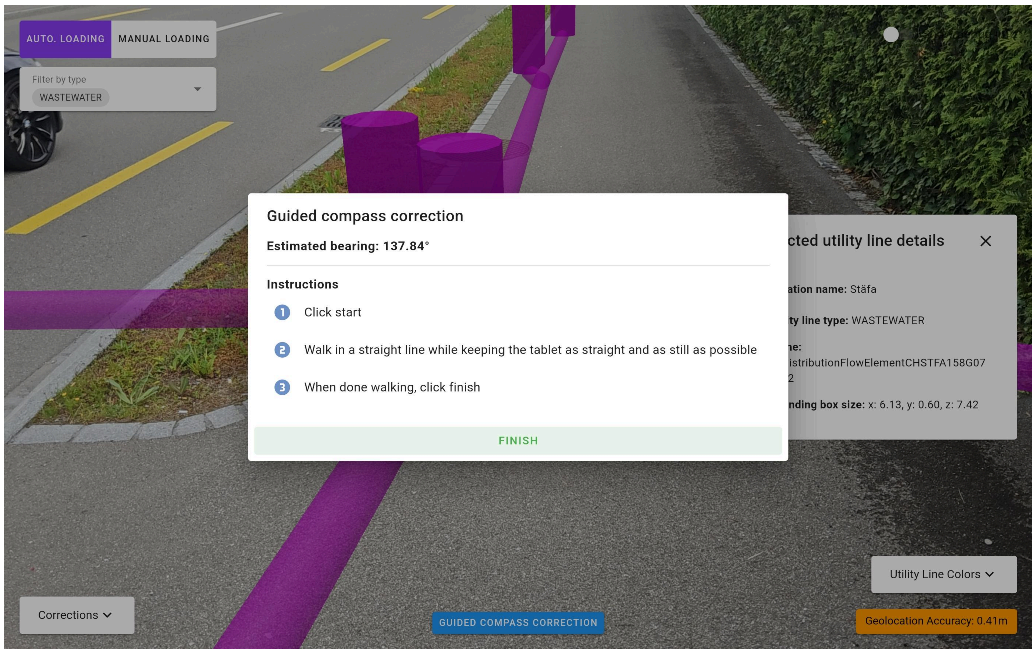


Figure 19.5: Screenshot showing the viewer with utility lines from Stäfa in tubAR



Guided compass correction

Estimated bearing: 137.84°

Instructions

1. Click start
2. Walk in a straight line while keeping the tablet as straight and as still as possible
3. When done walking, click finish

cted utility line details

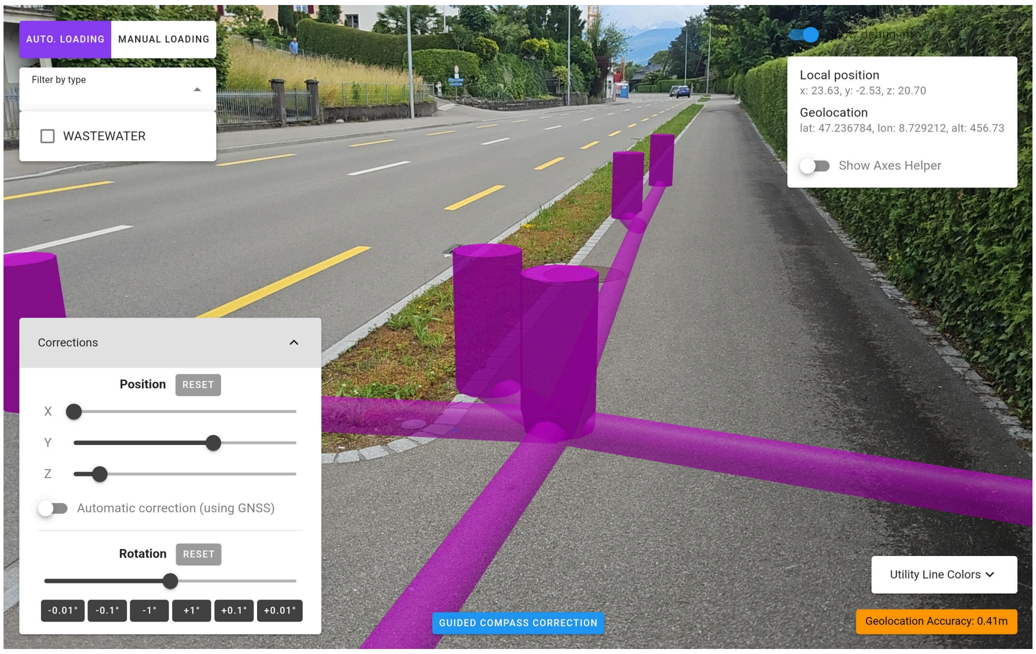
ation name: Stäfa

ty line type: WASTEWATER

ne:
istributionFlowElementCHSTFA158G07
2

nding box size: x: 6.13, y: 0.60, z: 7.42

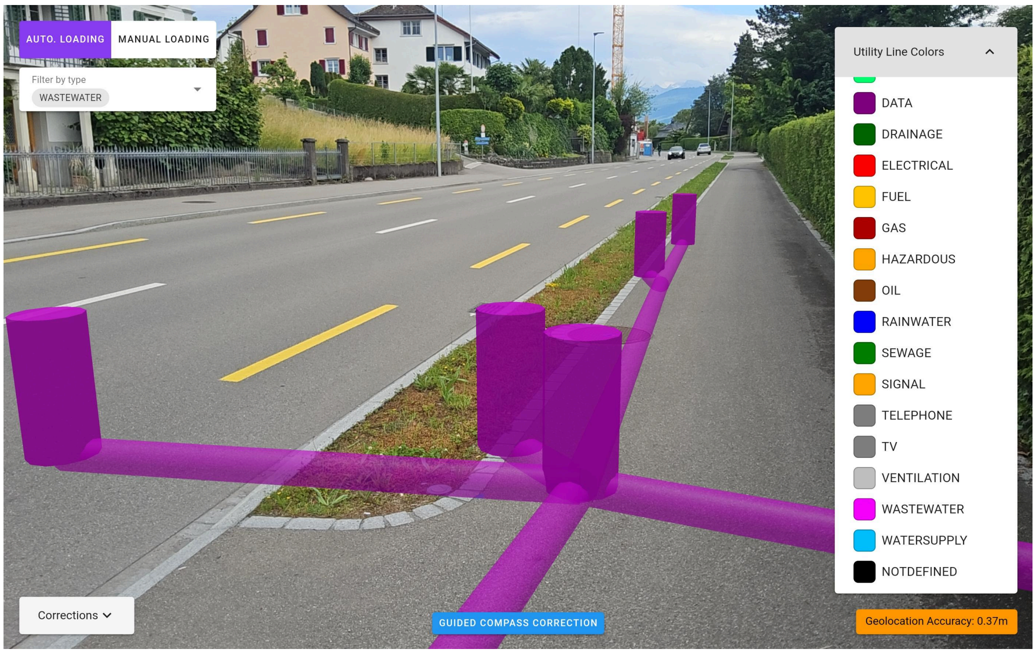
Figure 19.6: Screenshot showing the guided compass correction of tubAR



Local position
x: 23.63, y: -2.53, z: 20.70
Geolocation
lat: 47.236784, lon: 8.729212, alt: 456.73

Geolocation Accuracy: 0.41m

Figure 19.7: Screenshot showing the correction controls in the viewer overlay of tubAR



🟪	DATA
🟩	DRAINAGE
🟥	ELECTRICAL
🟨	FUEL
🟥	GAS
🟧	HAZARDOUS
🟫	OIL
🟦	RAINWATER
🟩	SEWAGE
🟧	SIGNAL
⬜	TELEPHONE
⬜	TV
⬜	VENTILATION
🟪	WASTEWATER
🟦	WATERSUPPLY
⬛	NOTDEFINED

Geolocation Accuracy: 0.37m

Figure 19.8: Screenshot showing the color to type legend in the viewer overlay of tubAR

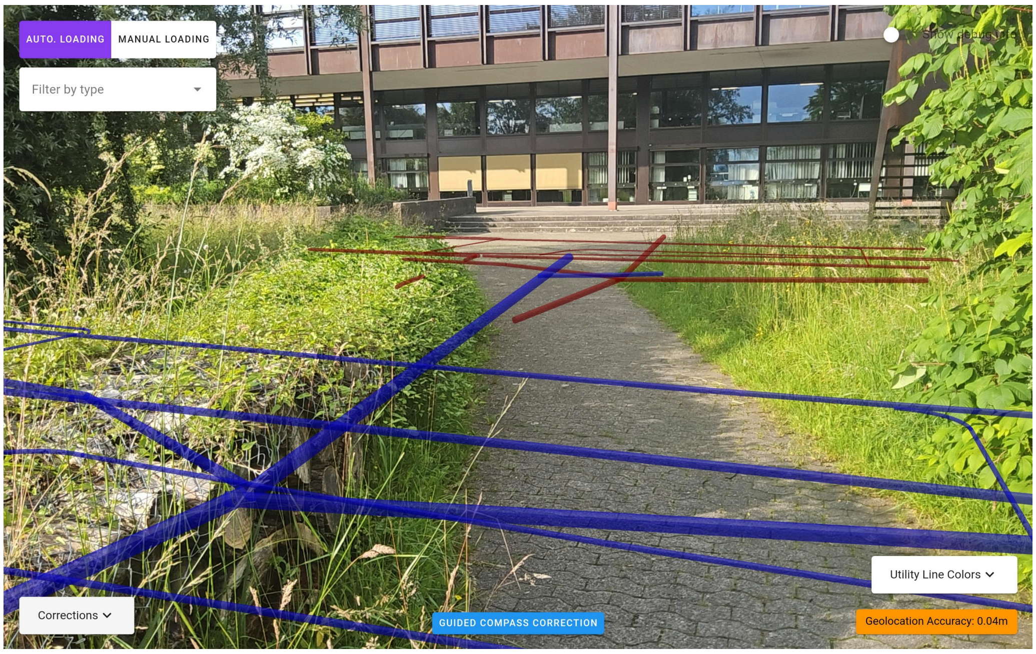


Figure 19.9: Screenshot showing the viewer with two different types of utility lines in tubAR

Appendix E: Bibliography

[1] vGIS. *vGIS marketing illustration*. June 12, 2024. URL: https://www.vgis.io/wp-content/uploads/2022/01/vGIS-augmented-reality-ar-for-GIS-utilities-sue-esri-arcgis-high-accuracy-engineering-grade-home.jpg.

[2] Kaj Habegger and Lukas Domeisen. *3D-Visualization of Utility Lines in the Browser Using Augmented Reality on Tablets*. May 16, 2024. URL: https://eprints.ost.ch/id/eprint/1188/.

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

[4] vGIS. *vGIS website*. May 28, 2024. URL: https://www.vgis.io/.

[5] vGIS. *vGIS pricing*. May 28, 2024. URL: https://www.vgis.io/ar-mr-vr-gis-vgis-pricing/.

[6] ARUtility. *ARUtility website*. May 28, 2024. URL: https://www.arutility.com/.

[7] ARUtility. *ARUtility pricing*. May 28, 2024. URL: https://www.arutility.com/pricing.

[8] ARonLine. *ARonLine website*. May 28, 2024. URL: https://gisonline.co/aronline/.

[9] V-Labs. *V-Labs website*. May 28, 2024. URL: https://www.v-labs.ch/.

[10] vGIS. *vGIS technical details*. June 2, 2024. URL: https://www.vgis.io/technical-specification-vgis-high-accuracy-survey-grade-augmented-reality-ar-for-bim-gis-arcgis-esri/.

[11] vGIS. *vGIS Features*. June 2, 2024. URL: https://www.vgis.io/vgis-utilities-features-high-accuracy-survey-grade-augmented-reality-ar-bim-gis/.

[12] vGIS. *Why vGIS?* June 2, 2024. URL: https://www.vgis.io/why-vgis-utilities-high-accuracy-survey-grade-augmented-reality-ar-bim-gis-2/.

[13] ISO/IEC 25000. *ISO/IEC 25010*. Oct. 1, 2023. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

[14] Keycloak. *Keycloak homepage*. Mar. 17, 2024. URL: https://www.keycloak.org/.

[15] Okta. *Auth0 homepage*. Mar. 17, 2024. URL: https://auth0.com/.

[16] Broadcom. *RabbitMQ homepage*. Mar. 27, 2024. URL: https://www.rabbitmq.com/.

[17] Tony Garnock-Jones et al. *Introduction to Pika*. Mar. 27, 2024. URL: https://pika.readthedocs.io/en/stable/.

[18] Broadcom. *AMQP 0-9-1 Model Explained*. Mar. 27, 2024. URL: https://www.rabbitmq.com/tutorials/amqp-concepts.

[19] Tomer Filiba. *RPyC - Transparent, Symmetric Distributed Computing*. Mar. 27, 2024. URL: https://rpyc.readthedocs.io/en/latest/#.

[20] Red Hat Inc. *Keycloak Container image*. June 12, 2024. URL: https://quay.io/repository/keycloak/keycloak?tab=info.

[21] Keycloak. *Keycloak identity provider flow*. Apr. 10, 2024. URL: https://www.keycloak.org/docs/latest/server_admin/#_identity_broker_overview.

[22] MDN. *WebXR browser compatibility*. May 26, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API#browser_compatibility.

[23] W3C Immersive Web Working and Community Groups. *WebXR browser compatibility*. May 26, 2024. URL: https://immersiveweb.dev/#supporttable.

[24] Google. *ARCore supported devices*. May 26, 2024. URL: https://developers.google.com/ar/devices.

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

[26] Wikipedia contributors. *ArcGIS*. June 10, 2024. URL: https://en.wikipedia.org/wiki/ArcGIS.

[27] Aceinna Inc. *Integer Ambiguity Resolution*. June 10, 2024. URL: https://openrtk.readthedocs.io/en/latest/algorithms/ambiguityfix.html.

[28] Wikipedia contributors. *Industry Foundation Classes*. June 10, 2024. URL: https://en.wikipedia.org/wiki/Industry_Foundation_Classes.

[29] Kasm Technologies. *KasmVNC - Linux Web Remote Desktop*. June 9, 2024. URL: https://github.com/kasmtech/KasmVNC.

[30] Wikipedia contributors. *MoSCoW method*. June 10, 2024. URL: https://en.wikipedia.org/wiki/MoSCoW_method.

[31] Okta inc. *What is OAuth 2.0?* June 9, 2024. URL: https://auth0.com/intro-to-iam/what-is-oauth-2.

[32] Wikipedia contributors. *Rational unified process*. June 10, 2024. URL: https://en.wikipedia.org/wiki/Rational_unified_process.

[33] Wikipedia contributors. *Universal Transverse Mercator coordinate system*. June 9, 2024. URL: https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system.

[34] toji et al. *WebXR Device API Explained*. June 9, 2024. URL: https://github.com/immersive-web/webxr/blob/master/explainer.md#what-is-webxr.

Appendix F: List of figures

Appendix G: List of tables