Halvard Bolli Øverlien
Dawid Staniszewski
Gisle Brandsøy Furland

Planning tool for guardrail data capture

Bachelor's thesis in Computer Science
Supervisor: Saleh Abdel-Afou Alaliyat
May 2024

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Halvard Bolli Øverlien
Dawid Staniszewski
Gisle Brandsøy Furland

Planning tool for guardrail data capture

Bachelor's thesis in Computer Science
Supervisor: Saleh Abdel-Afou Alaliyat
May 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Abstract

iSi has developed a product named inSight, which is an innovative system for discovering faults and deviations in the Norwegian road system. inSight uses a car equipped with computer-controlled cameras, lasers, infrared lights and a GPS to execute data capture. The data is then processed by tailored machine learning models to identify faults or deviations on the captured guardrails. Capture data is also not available continuously. iSi requested ideas on how a planning tool for the execution of data capture could be implemented, including continuously reported capture data.

This thesis presents a concept for a planning tool for this capture. The tool is built as a responsive web application that can be used by both planners and drivers. It focuses on how to efficiently plan and execute work orders, organized through projects. Extensive use of data streaming allows for near real-time updates, notifying drivers of critical metrics. It is also used to update the position of the vehicle in an interactive map. The guardrails can also be viewed in this map, along with their status.

The application was developed using iterative methods inspired by the SCRUM framework. The development was done in weekly iterations, with planning and retrospective meetings. In addition, there were also biweekly review meetings, where the work done was showcased to both the supervisor from NTNU, as well as the client, iSi. These reviews provided the group important feedback on the work.

The final result may help in maintaining good quality guardrails, ensuring that they are up to standard. Well maintained guardrails play an important role in reducing the extent of damage in traffic accidents.

Sammendrag

iSi har utviklet produktet inSight, som er en innovativ løsning for å avdekke feil og avvik på rekkverk i det norske vegsystemet. inSight benytter en bil utstyrt med kamera, lasere og infrarøde lys samt en GPS for å gjennomføre datafangst. Innhentede data blir deretter analysert av tilpassede maskinlæringsmodellere for å avdekke feil eller avvik på rekkverkene. De innhentede dataene er derimot ikke tilgjengelige kontinuerlig, og iSi forespurte ideer til hvordan et planleggingssystem for gjennomføringen av denne fangsten kunne være.

I denne oppgaven legges det frem et konsept for et slikt planleggingssystem. Systemet er laget som en responsiv web applikasjon som kan benyttes av både planleggere og sjåfører. Det fokuserer på å kunne effektivt koordinere arbeidsordre i prosjekter samt utføringen av disse. Systemet benytter seg av mekanismer for sanntidsdata for å kunne gi sjåfører beskjed om viktig metrikk, samt å oppdatere et interaktivt kart som viser kjøretøyets posisjon. I det samme kartet vises rekkverkene og des status.

Utviklingen av applikasjonen er gjort ved hjelp av iterative metoder, hvor de brukte metodene er inspirert av SCRUM rammeverket. Utviklingen ble gjennomført i ukentlige iterasjoner, med planlegging og retrospektive møter. Annenhver uke ble det gjort en gjennomgang av utført arbeid sammen med veileder fra NTNU og oppdragsgiver, iSi. Disse gjennomgangene gjorde det mulig å få viktige tilbakemeldinger på det utførte arbeidet.

Sluttresultatet kan være til hjelp for vedlikehold av veirekkverk, og bistå i å opprettholde en god standard. Godt vedlikeholdte veirekkverk er viktig for å redusere skadeomfanget i trafikkulykker.

Preface

This thesis presents the results from a project done in collaboration with iSi. We were tasked to develop a planning tool for their product inSight. A major motivation for choosing this project has been the ability to explore a domain and technologies which the group had no previous experience with. It also posed as an opportunity to further garner experience with previously used technologies and methodologies for developing responsive, full stack web applications. We would like to thank iSi for this challenging and exciting task. In our collaboration with them, a number of people have contributed useful input to shape the direction of the project. We would like to thank Fred Husøy, Kenneth Sylte and Børge Torvik for their input on the work we have presented. We would also like to thank our supervisor Saleh Abdel-Afou Alaliyat for providing us with advice and constructive feedback, being a reliable contact throughout the duration of the project.

Contents

List of Figures

List of Tables

Abbreviations

List of all abbreviations in alphabetic order:

- **API** Application Programming Interface
- **ARIA** Accessible Rich Internet Applications
- **CI/CD** Continuous Integration and Continuous Delivery
- **CRS** Coordinate Reference System
- **CSS** Cascading Style Sheets
- **DDL** Data Definition Language
- **DML** Data Manipulation Language
- **DNS** Domain Name System
- **DOM** Document Object Model
- **EPSG** European Petroleum Survey Group
- **ER** Entity Relationship
- **FOV** Field of view
- **GIS** Geographic Information System
- **GNSS** Global Navigation Satellite System
- **GPS** Global Positioning System
- **HTML** Hypertext Markup Language
- **HTTP** Hypertext Transfer Protocol
- **IaC** Infrastructure as Code
- **IDE** Integrated Development Environment
- **JDBC** Java Database Connectivity
- **JPA** Java Persistence API
- **JSON** JavaScript Object Notation
- **JSX** JavaScript XML

- **JTS** Java Topology Suite
- **JWT** JSON Web Token
- **NLOD** Norwegian License for Open Government Data
- **NN1954** Norway Null 1954
- **NN2000** Norway Null 2000
- **NPM** Node Package Manager
- **NPRA** Norwegian Public Roads Administration
- **NTNU** Norwegian University of Science and Technology
- **NVDB** National Road Database (Nasjonal vegdatabank)
- **OCI** Open Container Initiative
- **OS** Operating System
- **ORM** Object Relational Mapping
- **PNPM** Performant NPM
- **REST** Representational State Transfer
- **SMTP** Simple Mail Transfer Protocol
- **SQL** Structured Query Language
- **SVG** Scalable Vector Graphics
- **SRID** Spatial Reference System Identifier
- **TLS** Transport Layer Security
- **UI** User Interface
- **UN** United Nations
- **URL** Uniform Resource Locator
- **UTM** Universal Transverse Mercator
- **VoIP** Voice over Internet Protocol
- **WAI** Web Accessibility Initiative
- **WCAG** Web Content Accessibility Guidelines
- **WKT** Well-known text
- **YAML** Yet Another Markup Language / YAML Ain't Markup Language
- **XML** Extensible Markup Language

1. Introduction

This chapter introduces the client, along with their goals and requirements for the project.

1.1 Background

iSi AS has developed a product named inSight, which introduced a new innovative way of detecting deviations on guardrails in the Norwegian road network. inSight uses cars equipped with computer-controlled cameras, lasers, infrared lights and a GPS to power a digital inspection of guardrails. The car equipment captures data from guardrails, which is later processed by tailored artificial intelligence and machine learning models to identify faults and deviations. The equipment on the cars is shown in Figure 1.1.1. inSight has been developed as an innovation project in collaboration with Arvid Gjerde AS, The Norwegian Public Roads Administration, NTNU Ålesund, Møre og Romsdal County Municipality and Innovation Norway.

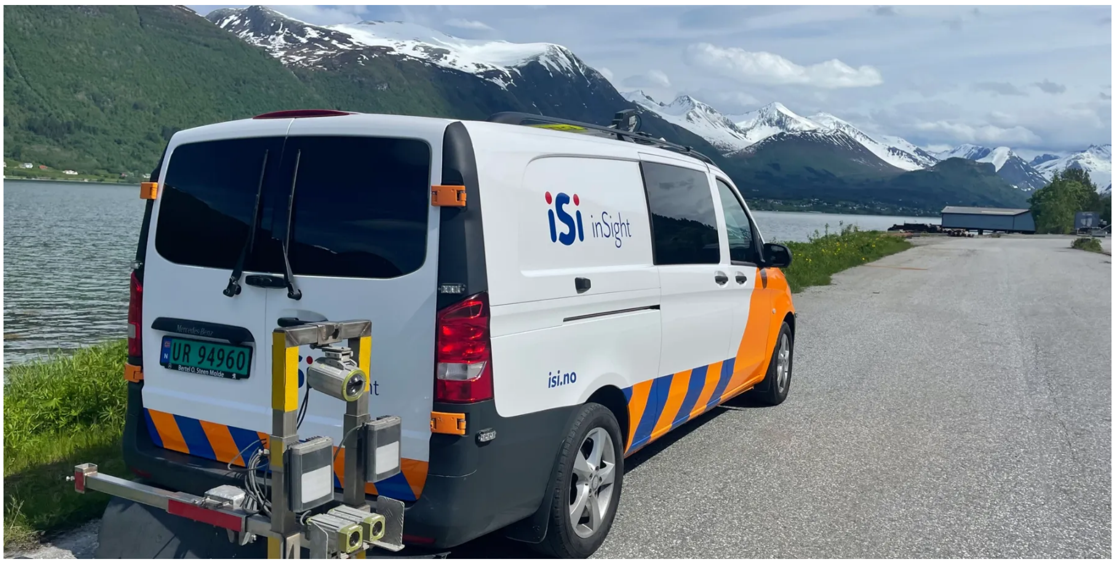


Figure 1.1.1: Car equipped with camera rig [1]

1.2 Project description

iSi already has systems in place for collecting data from the equipment on cars, but the information is not available to be presented and made use of continuously. In their task proposal, iSi requested ideas and suggestions on how to solve this challenge by transmitting data to a central service. The collected data should then be displayed to users, allowing drivers to plan their route and register what has already been captured as well as allowing planners to keep track of ongoing captures.

1.3 Motivation

When applying for this project, our primary motivation was to create an interactive and user-friendly interface that would allow both drivers and planners to efficiently plan and execute the data capture. The group found that the project both suited our previous experiences from web and application development while also allowing challenges in a domain the group had no prior exposure to.

1.4 Goals

Building on the specifications in iSi's proposal, the following goals for the result of this project were established:

- Produce an application usable on both mobile and desktop devices by using modern frontend technology
- Make the interface interactive and user-friendly, suitable for use during both planning and driving
- Display collected data in a clear and structured manner through the use of filtering and color-coding
- Update the data in close to real-time to inform users of the current status

In addition to these goals, the group defined some that they wanted to achieve through the development process:

- Gain further experience with agile development practices
- Learn about how geographic data can be visualized in modern frontend applications
- Gain experience in working with clients in an iterative method

1.5 Scope

The project had an open-ended scope, where iSi requested input from students on how a planning tool could be made for inSight. It was noted that it should ideally be possible to use the solution both on desktops in the office for planners, as well as on mobile on the drivers' phones. The open-ended scope of the project allowed the group to decide how to develop the application, both in terms of design and technology, while adjusting the product using feedback from iSi. Some of the main tasks mentioned were that the solution should integrate map data, as well as utilize data from official REST APIs such as NVDB.

1.6 Thesis structure

Chapter 2 - Theory: Describes various concepts and theoretical background for this thesis

Chapter 3 - Method: Highlights both the technology and methodology used in this project

Chapter 4 - Results: Presents the results of the project

Chapter 5 - Discussion: Discusses the challenges faced during the project

Chapter 6 - Conclusion: Concludes the work presented and notes work that can further improve the results

Chapter 7 - Societal impact: Describes the societal impact of the system and its relation to UN's sustainable development goals

2. Theory

This chapter introduces the theoretical background recommended to fully grasp concepts discussed in this thesis. Programming related concepts, technologies, principles and methodologies which set a baseline for the development of the application are described here.

2.1 Standards

This section covers the standards used and adhered to in the application.

2.1.1 Well-known text representing geometry

Well-known text, abbreviated WKT, is a standardized textual representation of geospatial objects. The standard defines how geospatial objects such as Point, LineString and Polygon can be defined in text [2]. Examples of how different geometries can be represented in WKT can be seen in Listing 1.

```
LINESTRING (6.2334 2.00083, 3.4002 4.3002)
POINT (3.3020 3.2002)
POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
```

Listing 1: Example of Line, Point and Polygon geometries in WKT

2.1.2 Coordinate reference systems

Coordinate reference systems (CRS) define how a two-dimensional planar shape relates to a three-dimensional spherical shape. These relations are used to measure locations on the surface of Earth as coordinates. Reference systems are typically given a Spatial Reference System Identifier (SRID) [3].

2.1.3 EPSG Geodetic Parameter Dataset

The EPSG Geodetic Parameter Dataset is a registry containing definitions of coordinate reference systems, coordinate transformations and other related data. GIS systems use codes from the EPSG dataset as SRIDs and related CRS data to perform transformations between reference systems [4].

2.1.4 JSON

JavaScript Object Notation, abbreviated JSON, is a data format which is both easy to read for humans and easy to process for machines [5]. JSON is inspired by the notation for object literals in the JavaScript programming language [6]. JSON is built using collections of name/value pairs and ordered lists of values [5]. Values can either be strings, numbers, lists or collections of key/value pairs. Listing 2 showcases the JSON data format with various constructs.

```
{
  "key": "value",
  "number": 1,
  "object": {
    "key": "value"
  },
  "list": [
    {
      "key": "value"
    }
  ]
}
```

Listing 2: Example of various constructs in the JSON format

2.1.5 YAML

"Yet Another Markup Language" or "YAML Ain't a Markup Language" is a human readable data serialization language optimized for writing files such as configuration settings [7]. An example of YAML is shown in Listing 3.

```
key: value
number: 1
object:
  key: value
list:
  - key: value
```

Listing 3: Example of code syntax in YAML

2.1.6 Problem Detail

Problem Detail is a proposed standard specification that defines a simple format for describing the specifics of an encountered problem. The specification defines structures for problems in both XML and JSON formats [8].

2.1.7 JSON Web Token

JSON Web Tokens, abbreviated JWTs, are JSON-based security tokens that contain a set of claims that can be signed, encrypted or both. JWTs are Base64 encoded and URL safe [9].

2.1.8 OpenAPI Specification

The OpenAPI Specification defines a formal standard for describing HTTP APIs. The format is used to communicate how an API works without requiring source code access, documentation or inspection of network traffic [10]. OpenAPI definitions can be used to generate both documentation and client code [10].

2.2 Universal design

Universal design is the concept of designing products that are usable by all people, regardless of age, size, ability or disability [11]. This section outlines principles and guidelines that can be used to ensure accessibility and universal design.

2.2.1 WCAG

Web Content Accessibility Guidelines, abbreviated WCAG, are guidelines for achieving web accessibility for people with disabilities [12]. The "WCAG 2.2" defines five main principles. The first principle, Perceivable, states that "information and user interface components must be presentable to users in ways they can perceive." The second principle, Operable, is that "user interface components and navigation must be operable." The third principle, Understandable, specifies that "information and the operation of the user interface must be understandable." The fourth principle, Robust, requires that "content must be robust enough to be interpreted reliably by a wide variety of user agents, including assistive technologies." Lastly, the principle of Conformance ensures that a website or application adheres to the aforementioned standards [13].

2.2.2 ARIA

Accessible Rich Internet Applications, abbreviated ARIA, comprise a set of roles and attributes designed to enhance the accessibility of web pages for individuals with disabilities. For example, a web browser typically assigns no inherent meaning to an empty `<div>` element. However, the addition of an ARIA attribute such as
`<div role="progressBar">`, informs the browser that this element functions as a progress bar [14]. The WAI-ARIA specification includes an Authoring Practices Guide which recommends approaches for developers to help make web application behavior accessible through utilizing such attributes [15].

2.3 Observer design pattern

The observer design pattern is a pattern that can be used to define a relationship for one way messaging between different parts of an application. The pattern consists of two components, publishers and subscribers. A publisher is responsible for notifying relevant subscribers when certain criteria is met, such as the occurrence of a user input event [16].

2.4 HTML

HTML is short for Hypertext Markup Language and is a core building block of the Web. It is used to define both the structure and meaning of content in web pages [17].

2.5 CSS

Cascading Style Sheets, abbreviated CSS, is another core building block of the Web. CSS is a stylesheet language that can be used to describe how content of web pages should be rendered on screen, paper, in speech or on other media [18].

2.6 JavaScript

JavaScript is a general purpose dynamic programming language. The language is known as the scripting language used in web browsers and allows for instance modifying the contents of a web page through APIs. In addition to running on browsers, it also runs in non-browser environments such as Node.js. The JavaScript language is multi-paradigm, supporting object-oriented, imperative and declarative styles of programming [19].

2.7 TypeScript

TypeScript is a superset of the JavaScript language that introduces additional syntax for types to the language [20]. TypeScript code does however not run in the browser like JavaScript, requiring a transpilation step where types are erased [21].

2.8 JSX

JSX is an extension of the JavaScript syntax that allows defining markup in JavaScript [22]. In contrast to the markup languages it renders to, it supports constructs like loops and conditions. The syntax extension is often used for defining components that render to HTML using frameworks [23].

2.9 React

React is a JavaScript framework for building user interfaces for both the web and native platforms. React uses the JSX syntax extension for defining interfaces in a composable manner using components [23]. React handles state updates through a Virtual DOM. State updates are made in the Virtual DOM, which a reconciler like React DOM can use to compare the difference between the Virtual DOM and the actual DOM and update accordingly [24].

2.10 Solid.js

Solid.js is a JavaScript framework for building user interfaces in the web. Similar to React, Solid.js uses the JSX syntax extension for defining interfaces. Solid.js differs from some of the other frameworks by utilizing signals for updating the interface. Signals are similar to the Observer design pattern, and allow the framework to only update the parts of a page that need to be updated. However, the process of establishing subscriptions is handled by the framework. This approach differs from frameworks like React, where a Virtual DOM is used. The framework is designed to build high-performing interfaces [25]. It is also among the top performing frameworks across multiple metrics in the js-framework-benchmark [26].

2.11 Road data

The Norwegian Public Roads Administrations maintains an open API for accessing road data, referred to as NVDB. The data is licensed under the Norwegian License for Open Government Data (NLOD) [27]. The API serves a database that contains information about the physical road system in Norway, and various object types attached to it, such as guardrails along with other metadata. The default SRID used for the Norwegian road data is EPSG SRID 5973, which is a compound CRS that combines UTM coordinates from SRID 25833 with NN2000 heights from SRID 5941.

2.11.1 Stretches

Roads are divided into systems that start and end at intersections. These systems are further divided into stretches. A stretch is represented by a series of geographical points, forming a line. The direction of the stretch is the same as the order of these points, and can be either with or against the network.

2.11.2 Guardrails

Guardrails are one of the road object types attached to the road network [27]. A guardrail is placed along or more stretches, and has a set direction and side position. The direction may be with or against the stretch. The guardrail is represented in the same way as a stretch, using a series of geographical points to form a line. In addition, some guardrails do not have their own geometry, and will then use a subset of the roads geometry as its representation instead.

2.12 Geoid

Geoid is the shape of the ocean surface under the influence by gravity. Geoid is commonly expressed as geoidal height or geoid undulation, in terms of a reference ellipsoid height. The reference ellipsoid is a shape that estimates the geoid [28]. GPS or GNSS navigation systems typically measure height in ellipsoid height. Undulation is not standardized, and different regions use varying mean sea levels as a reference [29]. NN2000 is a common nordic height reference, short for Norway Null 2000. This height reference superseded the previously used NN1954 [30]. An illustration of the relation between the geoid and the ellipsoid can be seen in Figure 2.12.1.

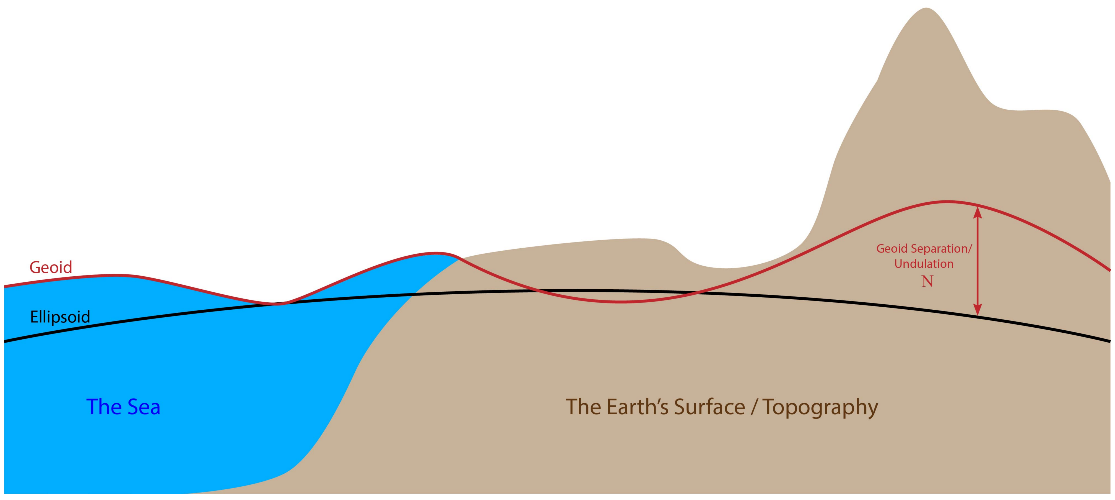


Figure 2.12.1: Geoid illustration [31]

2.13 Client-server architecture

Client-server architecture refers to a way of structuring a computer network in which remote devices (clients) request and receive service from a centralized host device (server). Client devices provide an interface through which requests can be sent to the server. Upon arrival of a request, the server will respond back to the client, providing the requested service if possible [32].

2.14 Server-sent events

Server-sent events use a one-way connection to push new data to a web page. Messages are transmitted over a HTTP connection, where the browser can treat the incoming messages as events [33].

2.15 REST

REST, short for Representational State Transfer, is an architectural pattern for web servers. The core principle of REST is that the server will respond with the representation of a resource, which is typically an HTML, XML or JSON document [34]. This representation will include the resource data, metadata and hypermedia that can be interacted with to achieve the desired state. A resource can be a document, image, service, non-virtual object, a collection of resources, and so on [35].

2.16 HTTP methods

HTTP methods are a set of standardized commands that indicate a desired action to be performed for a given resource. There are various different commands, all represented by different verbs. The verbs typically describe the intention of a request. Commonly used verbs are POST, GET, PUT, and DELETE, which are typically used to create, read, update, and delete resources, respectively [36].

2.17 Relational Databases

A relational database is a type of database that organizes data into rows and columns, which collectively form a table where the data points are related to each other. Relational databases use unique identifiers known as primary keys and foreign keys to establish different relationships between tables [37]. Relational databases use different languages to interact with them, where SQL is the most adopted language.

2.18 SQL

SQL, short for Structured Query Language, is a language commonly used to interact with relational databases. The SQL syntax includes a Data Definition Language (DDL), a Data Manipulation Language (DML), a Data Query Language (DQL) and a Data Control Language (DCL) [38]. A DDL is used to define the structure of entities within the database schema [39]. A DML on the other hand, is used to insert, update or delete records from the schema [40]. The data inserted into the defined schema can be queried using a DQL [41].

2.19 SMTP

Simple Mail Transfer Protocol, abbreviated SMTP, is a standard communication protocol used to transmit email. While the protocol allows for both sending and receiving emails, it is typically only used by clients to send email [42].

2.20 Version control

Version control is a system designed to record changes for a file or set of files, enabling the retrieval of specific versions at a later date [43].

2.21 Testing

Testing in software development is critical for ensuring reliability, functionality and security of the application. Testing can be done through several strategies, some of which are covered in this section.

2.21.1 Unit testing

Unit testing is a methodology in software testing where focus is on individual functions, methods, or classes, ensuring each unit operates as intended. This is often the first line of defense before conducting further testing [44].

2.21.2 Integration testing

Integration testing is a software testing methodology used to determine how well individually developed components, or modules of a system communicate with each other. Integration tests ensure higher test coverage by exposing system-level issues such as broken database schemas or faulty service integration. This testing methodology serves as a feedback loop throughout development, and can enhance the quality and reliability of the software [45].

2.21.3 End-to-end testing

End-to-end testing is a software testing methodology that evaluates the functionality and data flow of an application across multiple subsystems to ensure they work together from start to finish. Simply testing a single component may not be enough; instead, performing end-to-end testing verifies the application from start to finish by putting all its components together. This approach can help ensure that the application meets overall requirements and functions as intended in a real-world scenario [46].

2.22 Containerization

Containerization is a form of operating system virtualization. It packages everything needed to run an application or service into a single unit [47]. The Open Container Initiative, OCI, defines industry standards around the format of these containers. In addition to defining standards for the format of the containers, the OCI Runtime specification defines how these images can be run once unpacked to a Runtime filesystem bundle [48]. Docker is a platform that can be used to build, share and run container application [49]. containerd is an example of a runtime complying to the OCI Runtime Specification [50]. It is utilized by Docker as its runtime [51]. It is also included in some Kubernetes distributions, including K3s [52]. Kubernetes is a platform for managing containerized workloads, including both declarative configuration and automation [53].

2.23 Infrastructure as Code

Infrastructure as Code, abbreviated IaC, is a descriptive model of computing infrastructure, and can be used to provision and support infrastructure [54]. Similar to how the same source code generates the same binary, with IaC, the same source code generally generates the same environment every time it deploys [55].

2.24 Continuous Integration and Continuous Delivery

Continuous Integration is a practice where the integration of code changes from multiple contributors in a single software project is automated. It allows developers to frequently push changes to a central repository, which will trigger builds and tests to run to verify integrity [56].

Continuous delivery pipelines extend the idea of continuous integration by deploying all code changes that pass both the test and the build stage. Continuous delivery allows for defining at which interval the software should release, based on business needs [57].

2.25 SCRUM

SCRUM is a framework designed to enhance team collaboration on complex tasks through the use of short, iterative work cycles known as Sprints. By enabling teams to complete tasks in small segments, SCRUM promotes better organization and efficiency in tackling complex projects. The SCRUM framework is made up of a SCRUM team, consisting of a Product Owner, a SCRUM Master and Developers who all follow specific roles, events, and artifacts defined in the SCRUM Guide [58] [59].

2.25.1 SCRUM Master

The SCRUM Master serves as the facilitator for the development team, ensuring that the SCRUM framework is followed. The intention of this role is to eliminate roadblocks and support the team in achieving their goals efficiently [59].

2.25.2 Product Owner

The Product Owner of a SCRUM team represents the interests of the stakeholders and is responsible for managing the product backlog and prioritizing work [59].

2.25.3 Development Team

The Development Team is a group responsible for delivering a potentially releasable increments of product at the end of each Sprint. The development team works alongside the SCRUM Master and Product Owner, and has a crucial role of driving the project's progress and ensuring alignment with the stakeholders' expectations and product vision [59].

2.25.4 Sprints

A Sprint is a fixed time period containing a number of SCRUM events. These SCRUM events are structured meetings with the goal of organizing work, gathering feedback, and improving both the product and the process continuously. A Sprint typically lasts one to four weeks and aims to reach a specific goal set by the team. Prior to the Sprint, a common goal is established. Changes made to the Sprint that could endanger the Sprint goal should be avoided [59]. Figure 2.25.1 illustrates the events within a Sprint that form a cycle when executing sprints in succession.

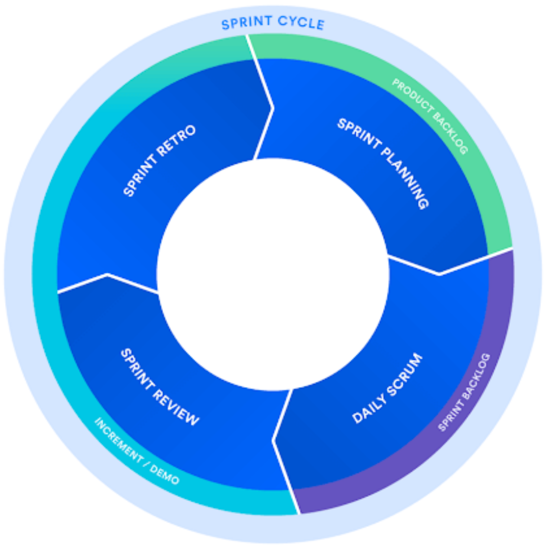


Figure 2.25.1: Sprint cycle [60]

2.25.4.1 Sprint Planning

Sprint Planning is an event that comes at the start of each sprint, with the purpose to define the work that needs to be done. During planning, the team sets goals for the Sprint, plans how the goals are going to be achieved, and distributes work to the members. The backlog of the product is used in this phase to see what work is remaining and what should be prioritized [59].

2.25.4.2 Daily stand-up

The daily stand-up is a 15-minute event held every day where developers gather to discuss their daily tasks. This meeting promotes open communication, simplifies complexities, and reduces the need for additional meetings [59].

2.25.4.3 Sprint Review

A Sprint Review is the second to last event of a Sprint. It is a meeting held between the product owner, the development team, and stakeholders if any. The purpose of a Sprint Review is to showcase what work has been done, gather feedback and identify the next steps. It ensures that everyone is up to date and makes it easier to plan future work [59].

2.25.4.4 Sprint Retrospective

A Sprint Retrospective is the final event in a Sprint. The purpose of it is to increase quality and effectiveness. During a Sprint Retrospective, the team identifies what went well, what should continue, and what needs to change, regarding individuals, interactions, processes and tools. The most impactful improvements are then expected to be addressed as soon as possible, and may even be added to the backlog for the next Sprint [59].

3. Method

This chapter covers the methods used during the project for work organization and application development.

3.1 Workflow

The work for the project was organized in one week sprints, using the SCRUM framework as an approach to organizing the work. The project consisted of four phases, where each of the phases were expected to be executed in parallel. The start of the project was however mainly used to understand the problem space, and attempt to design a user interface for the application in dialogue with the client. The four phases that were defined are Preliminary Report, Design, Development and Thesis.

3.1.1 Daily stand up

Even though daily stand ups are part of the SCRUM framework, the group ignored this activity. The group found that a planned activity for sharing what was being worked on and any eventual challenges was unnecessary as both progress and challenges naturally surfaced in other contexts. It however did not cause too much disturbance to other group members' work.

3.1.2 Sprint Planning

To plan activities for upcoming sprints, the group held non-formal meetings to discuss which tasks should be included. Throughout the project, the group maintained a backlog consisting of both bugs and unfinished user stories. These were prioritized and sorted in the backlog by importance in relation to the project, with the planned user stories inferred through the use case diagram considered most important. Smaller bugs or tasks would also be included in sprints to increase scope whenever time allowed it. The created issues did not include estimates.

3.1.3 Sprint Review

During the project, the group organized meetings with the client every two weeks. For each of these meetings, a document outlining the work done in the previous two sprints was prepared. This document was presented to the client and supervisor at the meeting, allowing them to give feedback either during or after it.

3.1.4 Sprint Retrospective

Following the end of each sprint, a retrospective meeting was held. During these meetings the group evaluated the processes and work methodology, reflecting upon the approach. The retrospective documents were made using a template in Jira, and organized in a folder in the Jira project pages.

3.1.5 Logging work

At the end of each work session, the hours spent on different tasks were logged. Using the time tracking feature in Jira, the group added work logs to each issue. To gain an extra overview in addition to issue work logs, the TimeSheet Tracking for Jira app was used.

3.1.6 Communication

Communication was a core part in the group's organizational workflow. To facilitate this, Discord, a VoIP platform that supports text chat, voice chat, screen sharing, and file sharing, was used. These features enabled a productive environment while working digitally.

Email communication and digital meetings over Teams were the main strategy for communicating with the stakeholders. In addition to these meetings, meeting notes and Sprint Review documents were shared on Confluence, which allowed for asynchronous feedback on the presented work.

3.1.7 Commit message convention

To streamline the authoring of commit messages, the group decided to follow version 1 of the Conventional Commits specification. Along with the specification, the group attached issue identifiers from Jira, allowing the GitHub integration to show which commits are tied to which issues. Combined with environment deployments in GitHub, it also allowed Jira to display whether or not an issue was published.

3.1.8 Code formatting

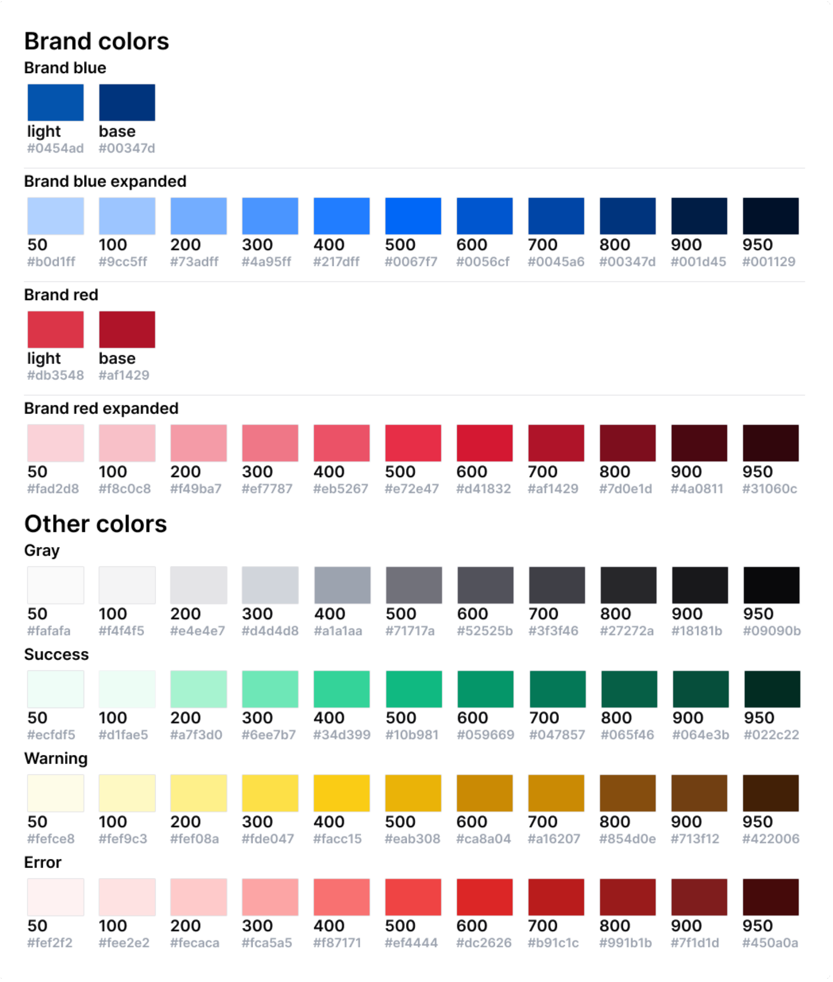
The Eclipse and Prettier code formatters were utilized during development. The Eclipse formatter was used for Java code, whilst Prettier was used for TypeScript, HTML and CSS code. They were set up to use shared configurations, which their respective editor extensions integrated into the workflow. The extensions were used to automate the process of ensuring consistent looking code.

3.2 Design

In the early phases of the project, design was prioritized. Even though it lowered in priority further down the line, it was an iterative process throughout the whole project. This section discusses the design process and the guidelines followed during the development process.

3.2.1 Design guidelines

A simplistic theme was chosen for the application to balance out the detailed look of the map. To achieve this, a minimalistic component library was used. These components were combined with standardized values for spacing, rounding, shadows and other styling options. The color scheme consists mainly of a grayscale palette taken from TailwindCSS and a generated blue one based on iSi's branding color. The colors used can be seen in the palette in Figure 3.2.1. As a measure to balance the interface with the map, it has been important to avoid overloading the application interface with excessive content. This is also in line with accessibility principles outlined in WCAG 2 about minimizing the amount of content, helping reduce cognitive overload and minimize focus loss among users [61].



Brand colors

Brand blue

light	base
#0454ad	#00347d

Brand blue expanded

50	100	200	300	400	500	600	700	800	900	950
#b0d1ff	#9cc5ff	#73adff	#4a95ff	#217dff	#0067f7	#0056cf	#0045a6	#00347d	#001d45	#001129

Brand red

light	base
#db3548	#af1429

Brand red expanded

50	100	200	300	400	500	600	700	800	900	950
#fad2d8	#f8c0c8	#f49ba7	#ef7787	#eb5267	#e72e47	#d41832	#af1429	#7d0e1d	#4a0811	#31060c

Other colors

Gray

50	100	200	300	400	500	600	700	800	900	950
#fafafa	#f4f4f5	#e4e4e7	#d4d4d8	#a1a1aa	#71717a	#52525b	#3f3f46	#27272a	#18181b	#09090b

Success

50	100	200	300	400	500	600	700	800	900	950
#ecfdf5	#d1fae5	#a7f3d0	#6ee7b7	#34d399	#10b981	#059669	#047857	#065f46	#064e3b	#022c22

Warning

50	100	200	300	400	500	600	700	800	900	950
#fefce8	#fef9c3	#fef08a	#fde047	#facc15	#eab308	#ca8a04	#a16207	#854d0e	#713f12	#422006

Error

50	100	200	300	400	500	600	700	800	900	950
#fef2f2	#fee2e2	#fecaca	#fca5a5	#f87171	#ef4444	#dc2626	#b91c1c	#991b1b	#7f1d1d	#450a0a

Figure 3.2.1: Color palette

3.2.2 Diagrams

As a tool for understanding the problem space, several diagrams were drafted. To map which use cases needed to be handled by the application, use case diagramming was utilized. An ER diagram was also used to plan the structure of data that should be persisted in the database. In addition, other one-off diagrams were made to communicate ideas related to the problem space. Diagrams were generally made using diagrams.net.

3.2.3 Wireframes

With the use cases and functionality identified, a wireframe was produced. Wireframes were made to prototype the look, feel and functionality of the application. To make these wireframes, Figma was used. A component library was added in Figma to help iterate on the UI. The mobile UI prototyped for drivers can be seen in Figure 3.2.2. Figma offered a real-time editing experience, allowing a quick and collaborative iteration on concepts.

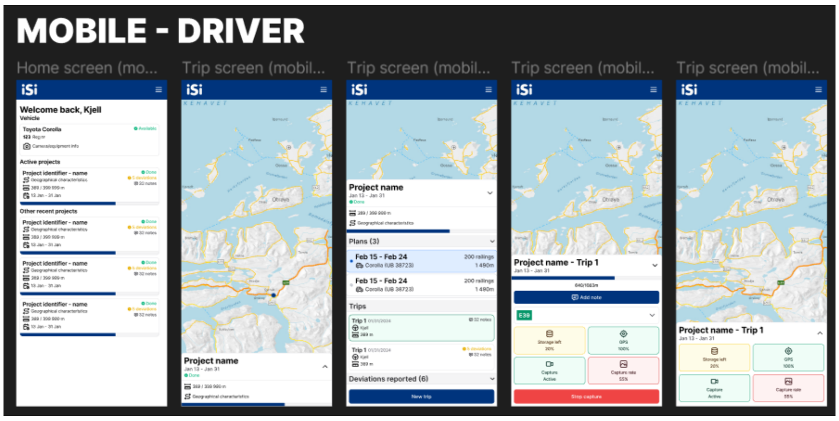


Figure 3.2.2: Figma wireframes for UI used by drivers on mobile

3.2.4 Design feedback

To ensure alignment on both design and the application workflow, we collected feedback on our wireframes from iSi. This played an important role in the project and further improved our understanding of the problem the application is trying to solve.

3.2.5 Universal design

The design of the application, while branded with iSi colors, needs to satisfy universal design requirements. To achieve this, the application was designed with the contrast ratio of colors in mind, increasing visibility. In addition, to increase the users' ability to perceive the interface, well known symbols and colors are utilized for statuses and recognizable UI elements. Furthermore, there are labels and error feedback on all forms in the application to help users avoid and correct mistakes.

3.3 Testing

This section covers the different types of testing done throughout the development process.

3.3.1 User testing

With the development of a feature-rich application, obtaining feedback on the workflow is essential. To address this, we established a demo environment where iSi could evaluate the workflow and performance of the application. This allowed them to identify and report any potential issues or inefficiencies in it.

3.3.2 Unit testing

To prevent regressions and increase the understanding of the problem, unit tests are used. Unit tests are authored using JUnit 5, with both negative and positive tests.

3.3.3 Integration testing

As another measure for verifying the understanding of problems, integration tests are used. Integration tests are authored with idempotency as a goal, to allow re-runs with predictable results. These tests are written using JUnit 5 in combination with Spring Boot Test. They verify that defined endpoints work within expectations for both read and write operations. In addition, there are persistence tests that verify that the defined entities can successfully be stored in and retrieved from the database.

3.4 Artificial intelligence tools

To explore possible or alternative solutions for programming related tasks, artificial intelligence tools like ChatGPT and GitHub Copilot were used. Copilot, and its IDE integration, were used to suggest solutions to problems, as well as to provide potential ways to complete code. In addition to this, ChatGPT was used to rephrase and spell check text.

3.5 Technologies

This section covers the various technologies chosen for this project, how they were used and why.

3.5.1 Solid.js

Solid.js is the UI framework used to develop the frontend application. UI is defined using the JSX syntax with functional components and renders to HTML in the browser. The signal primitive is used to trigger re-renders of UI elements, these signals change through events and data fetching.

3.5.2 TailwindCSS

To style the application, TailwindCSS was used. TailwindCSS is a utility CSS framework that provides a default set of utilities for design tokens such as font size, font weight, sizing and colors. The colors were renamed to semantic names, and the iSi brand colors were brought in as well. In addition to the built-in variant options like hover and focus, some extra variants were made that are bound through signals in Solid.js using Class Variance Authority.

3.5.3 Component library

While TailwindCSS provides utilities for styling the application, a set of reusable components were defined and used throughout the application. Some components were imported from solid-ui, an open-source library of components built using headless UI component libraries. solid-ui uses both the Kobalte and Corvu component libraries, and are as a result adopted in this project. These component libraries are authored following the WAI-ARIA guidelines [62] [63]. The styles of these components were adjusted to ensure consistency with the application theme. In addition to imported components, a number of components were custom made for domain specifics as well as repeated UI elements within those, often composing components from the aforementioned libraries.

3.5.4 Map rendering

To render the geographical data in a map, two different libraries were tested during development. In a meeting with iSi, Leaflet was suggested as it was the library they were most familiar with, meaning they could provide help if needed, as well as easily reuse the results of the project. The other alternative was OpenLayers. While OpenLayers uses a more object-oriented approach for their API when compared to Leaflet, it does have more functionality built-in, such as a WebGL renderer and a WKT formatter. iSi mentioned that they were familiar with this library as well.

3.5.5 Spring Boot

Spring Boot was used to develop the backend application. iSi has backend systems utilizing Java and Spring Boot already, and the group had previous experience using the framework, making it a good fit. The Spring Boot web starter was used to develop a web server using REST as the architectural pattern. Regular HTTP methods with JSON requests and responses are mainly used, but for real-time data, server-sent events were utilized. Server-sent events also emits JSON-encoded events. These events also follow the REST pattern, where each message transmitted is self-contained and independent. Spring HTTP interfaces were also used to integrate with the read API for NVDB.

3.5.6 Database

Spring Boot offers multiple ways to store data. In this project, PostgreSQL was used together with the PostGIS extension. The PostGIS extension extends PostgreSQL to support geospatial types, functions and indices. To communicate with the database from the backend, the approach was divided. Where data retrieval or data insertion had more complex requirements, it was opted to use the Spring Data JDBC for native SQL queries with manual row mapping logic. In simpler cases, Spring Data JPA was used as an ORM, leveraging its top-down approach for working with database entities as objects.

3.5.7 Database migrations

In order for the application to work with the database, the schema has to be created. Spring Data JPA does provide mechanisms for creating and updating this schema based off entities, but it is not able to correctly identify the intention of a change such as when renaming a field. To maintain full control over the database schema, Liquibase was used to define SQL migrations with both DDL and DML to evolve the schema. The migrations were executed in an ordered fashion, using a naming convention with a manually assigned sequence number as the prefix and a short descriptive name.

3.5.8 Geospatial types and transforms

The Java Topology Suite, abbreviated JTS, was used for geospatial types in the backend. JTS combined with Hibernate Spatial and Spring Data JPA allows entities to have properties of geospatial types that get persisted in the database. JTS also provides operations on geometries such as calculating intersections. Proj4j was used for transforming points between different reference systems. Proj4j does however not support vertical transforms.

3.5.9 Containers

To package application artifacts and run dependant technologies, containers are utilized. For development and testing, Testcontainers was made use of to allow quick verification of integration with the applications dependencies using temporary containers that are launched during the application bootstrap phase. To allow working with containers with durable storage, a docker compose file was set up, eliminating the need to recreate data between application restarts.

Application artifacts use different technologies for building. Spring Boot applications utilize Jib as a tool for building container images. The frontend, on the other hand, utilizes a Dockerfile to configure a NGINX container with static assets produced by building the application.

3.5.10 Object Storage

To support uploading images and files to a persistent store, an object storage system was used. MinIO was used, and allows testing locally as well as keeping it similar in a deployed environment. iSi already uses the object storage service from Amazon, S3, which MinIOs API is compatible with. This compatibility allows for using the same code and logic with S3 by changing configuration files.

3.5.11 Documenting APIs and code

The OpenAPI specification was used to document public facing HTTP APIs. HTTP interfaces in the client package are annotated with metadata that is used to generate a OpenAPI v3 schema for endpoints. In addition to direct annotations in the client package, authorization related logic and annotations are added in the implementation. Internal API surfaces are documented using Javadoc and JSDoc for backend and frontend respectively.

3.5.12 GitHub Actions

GitHub Actions was used to implement a CI/CD pipeline in the project. When code is pushed to the repository, a workflow run will be triggered to run tests and verify that building the application works. Deployment is run in a release workflow which is triggered by a tag being pushed to the repository. This delivery pipeline is not run as frequently to avoid noise. The workflows re-use actions built by the community for setting up tool chains such as Gradle, PNPM, kubectl and Helm.

3.5.13 Ansible

The deployment requires a host server to run on. NTNU provided a Ubuntu server that is used as a demo environment. Ansible playbooks were defined to configure the server with the necessary components. The playbooks will install or uninstall the K3s distribution of Kubernetes, along with integrations for ingress, managing certificates, domain name registration and a dashboard.

3.5.14 Kubernetes and Helm

Kubernetes is used as a container orchestrator, and runs the application in a demo environment. Kubernetes resources can be defined using IaC, in the YAML language. While maintaining raw YAML definitions is possible, in order to safely store them in version control the secrets have to be removed. To work around this, Helm is used to group these resources into a reusable package. Using Helm also allows passing values to the package, which is used to allow having secrets defined externally to the resources themselves.

3.5.15 Chrome DevTools

Chrome DevTools was used for both debugging and profiling the frontend application. The profiler can be used to record how much time is spent in specific parts of a web application, measured in milliseconds. Figure 3.5.1 showcases a recording from rendering 25.000 railings using the Canvas renderer in Leaflet.

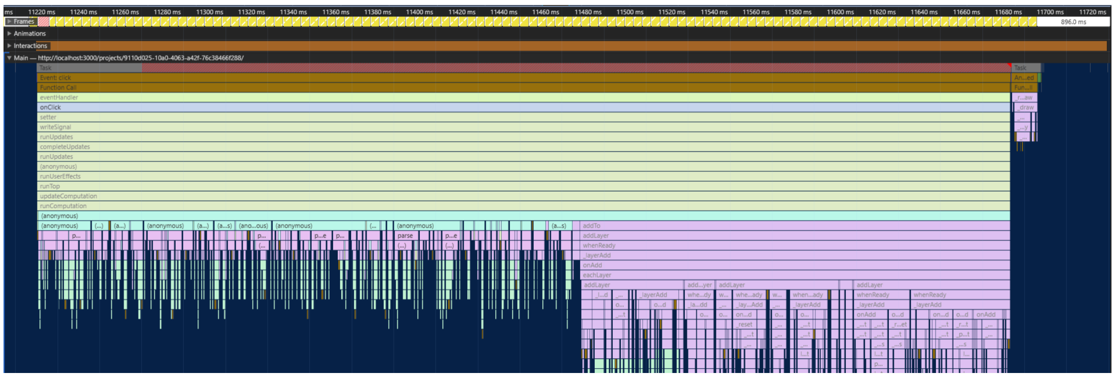
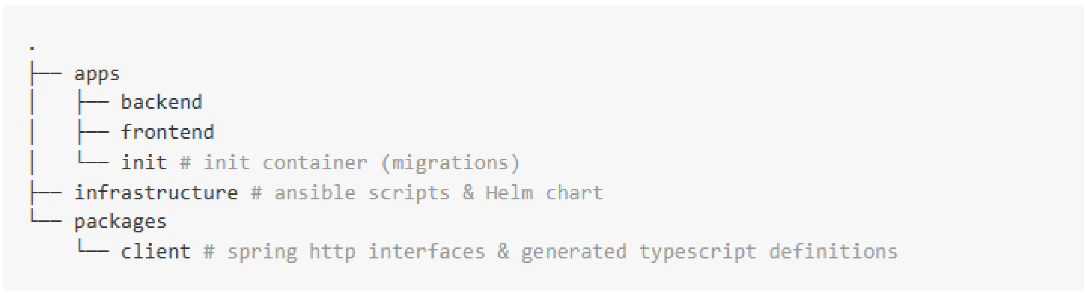


Figure 3.5.1: Recording of rendering 25.000 railings using Canvas renderer in Leaflet

3.6 Project Structure

The project is structured as a monorepository, meaning all projects or modules are part of the same repository. To structure modules within this repository, modules are categorized as infrastructure, applications or packages. Each of these have their respective folders, containing its modules. The structure is illustrated in Figure 3.6.1.

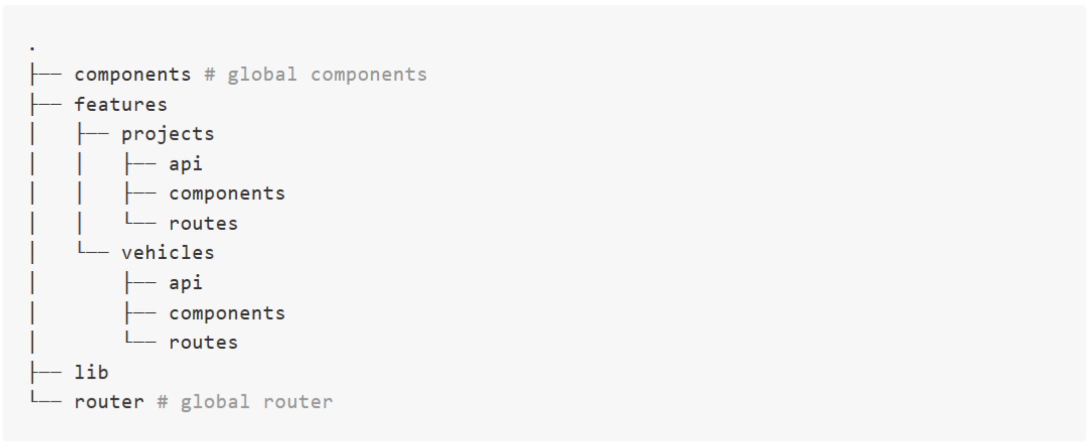


```
.
├── apps
│   ├── backend
│   ├── frontend
│   └── init # init container (migrations)
├── infrastructure # ansible scripts & Helm chart
└── packages
    └── client # spring http interfaces & generated typescript definitions
```

Figure 3.6.1: Overall project structure

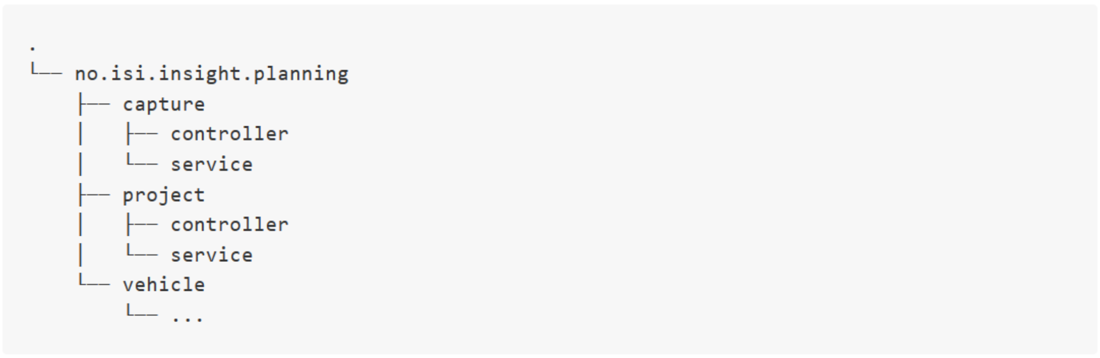
3.6.1 Module structure

In addition to structuring modules in categories, module code is also organized in a specific way. For frontend code, the organization is inspired by the Bulletproof React project, which groups code into features. The organization for backend Java code is structured similarly, using package namespaces to group related code. Figure 3.6.2 shows how the frontend code is organized, while Figure 3.6.3 shows how the backend code is organized.



```
.
├── components # global components
├── features
│   ├── projects
│   │   ├── api
│   │   ├── components
│   │   └── routes
│   └── vehicles
│       ├── api
│       ├── components
│       └── routes
├── lib
└── router # global router
```

Figure 3.6.2: Frontend code folder structure



```
.
└── no.isi.insight.planning
    ├── capture
    │   ├── controller
    │   └── service
    ├── project
    │   ├── controller
    │   └── service
    └── vehicle
        └── ...
```

Figure 3.6.3: Backend code folder structure

3.6.2 Client package

In order to separate outbound data objects and internals, the public API surface is covered in a separate package. The package contains Spring HTTP interfaces, which can be used to create API clients to interact with the backend in external systems. The HTTP interfaces define which data transfer objects are in the expected response. The definitions of these data transfer objects are also located within this package. HTTP interfaces are implemented by the backend application. In addition, the package generates TypeScript definitions based off the Java classes which can be consumed by the frontend application.

3.6.3 Application for initializing

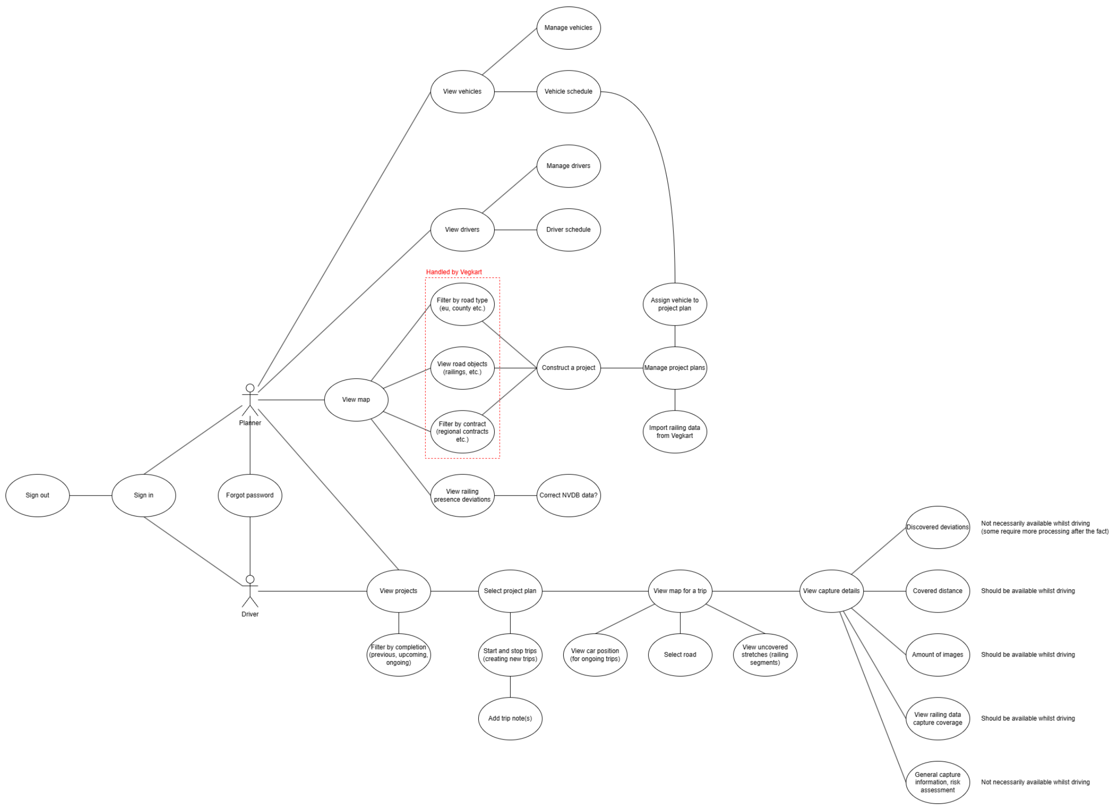
In addition to the main backend application, some of the responsibility is split out to a separate application. This application is responsible for running SQL migrations using Liquibase, executing both DDL and DML scripts to modify the database schema and its records. The application is expected to be run and successfully exit prior to starting the backend application. The application will exit once the migrations are successfully applied.

4. Results

In this chapter, the results of the project will be covered. In addition to an overview of the developed software, results such as diagrams and map rendering performance findings will be presented.

4.1 Use case diagram

To map the various uses of the system, a use case diagram has been iterated on. Figure 4.1.1 shows the end result of the use case diagram, after a number of iterations. Some of these use cases are handled by an external system, Vegkart, but are included to provide an overall view of the system's use. Additional use cases were added in later stages of the project. The diagram in full size can be found in Appendix C.



Manage vehicles
View vehicles
Vehicle schedule
Manage drivers
View drivers
Driver schedule
Handled by Vegkart
Filter by road type (eu, county etc.)
View road objects (railings, etc.)
Filter by contract (regional contracts etc.)
Assign vehicle to project plan
Construct a project
Manage project plans
Import railing data from Vegkart
Planner
View map
Sign out
Sign in
Forgot password
View railing presence deviations
Correct NVDB data?
Discovered deviations
Not necessarily available whilst driving (some require more processing after the fact)
Driver
View projects
Select project plan
View map for a trip
View capture details
Covered distance
Should be available whilst driving
Filter by completion (previous, upcoming, ongoing)
Start and stop trips (creating new trips)
View car position (for ongoing trips)
Select road
View uncovered stretches (railing segments)
Amount of images
Should be available whilst driving
Add trip note(s)
View railing data capture coverage
Should be available whilst driving
General capture information, risk assessment
Not necessarily available whilst driving

Figure 4.1.1: Final use case diagram

4.2 ER diagram

As a strategy for planning how information should be stored in the database, an entity relationship diagram was iterated on. The diagram was built iteratively, adding details to those entities that were certain and reiterating on others. Figure 4.2.1 shows the final result of the ER diagram and serves as a close representation of both the database schema and model classes in code. In addition, a full size version can be found in Appendix C.

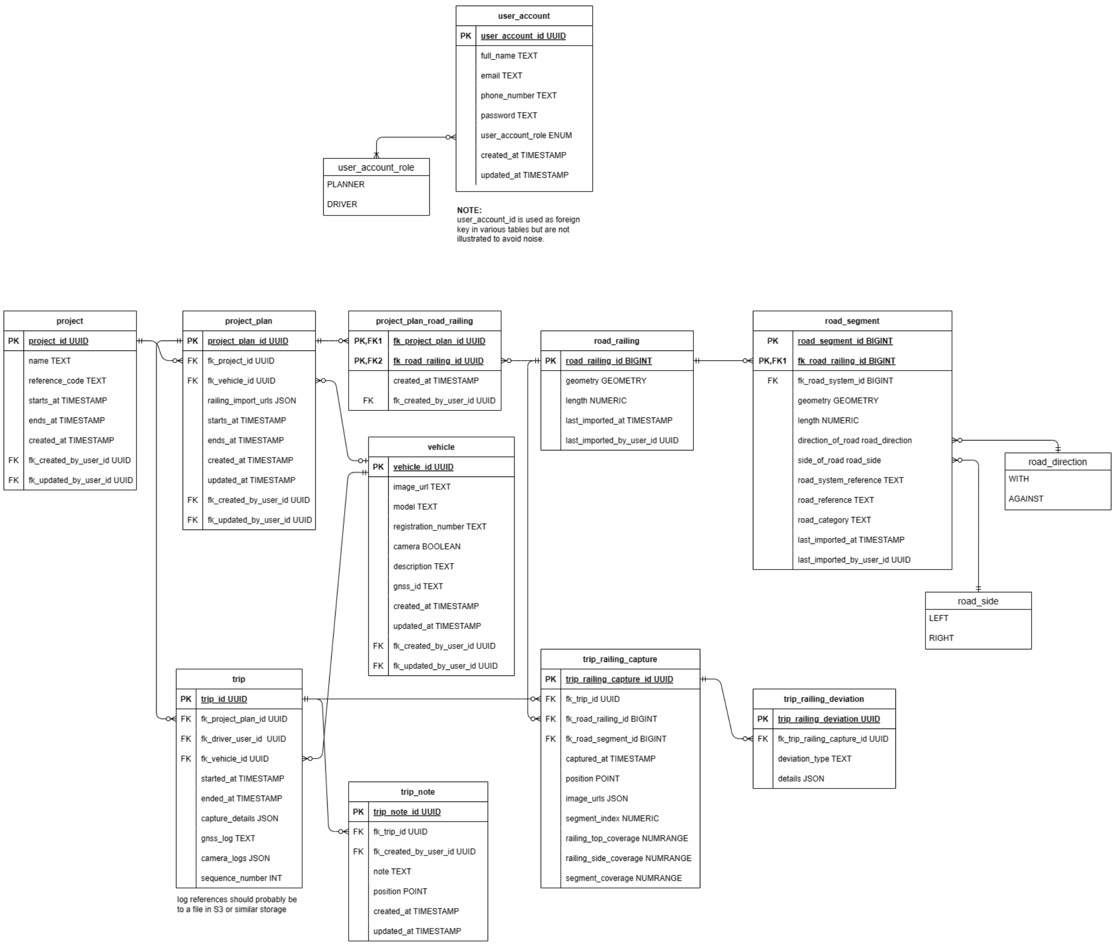


Figure 4.2.1: Final ER diagram

4.3 Transforms between vertical coordinate reference systems

The GNSS system reports height in ellipsoid height. It also reports undulation values that allows finding the Z coordinate of the cars position. However, these undulation values are not in terms of the vertical CRS used by NVDB. To solve this, a parser for reading height reference models published by The Norwegian Mapping Authority is implemented. The parser populates a grid of heights from a binary file in Gravsoft format, ported from a Python code sample from The Norwegian Mapping Authority [64]. Using a latitude and longitude pair, the undulation value for the given location can be found. Subtracting the measured ellipsoid height by the undulation from the reference model gives the correct height in the reference system represented in the file. While the parser is generic for files in the same format, it is only used to calculate NN2000 height using the ellipsoid height from the GNSS. The implementation was needed due to the lack of support for vertical transformations in Proj4j, which is used for transforming latitude and longitude values to the correct CRS. A visualization of the file's content using code adopted from the The Norwegian Mapping Authority is shown in Figure 4.3.1.

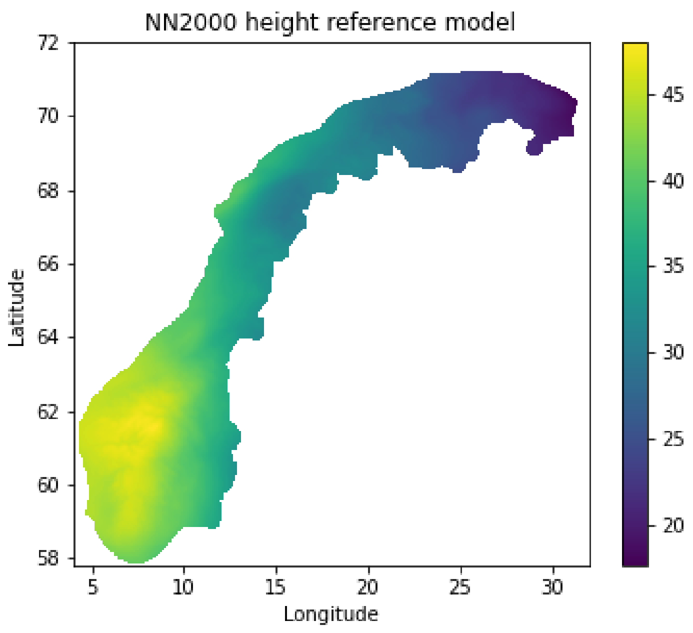


Figure 4.3.1: Visualization of NN2000 height reference values [64]

4.4 Application

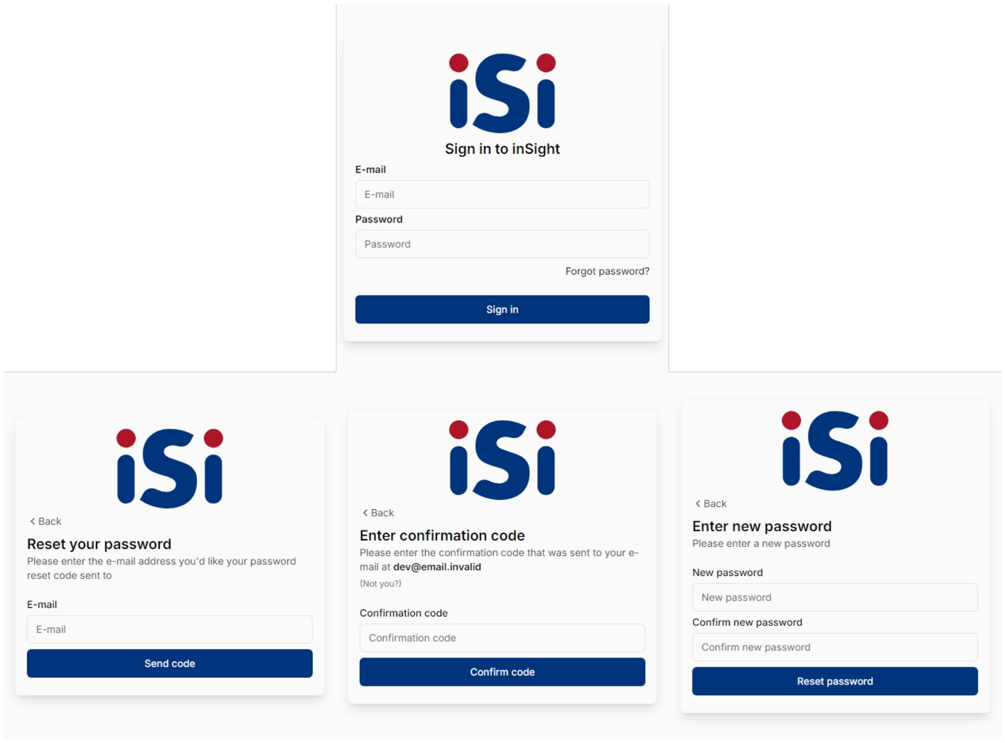
The software developed is a responsive web application which features authentication, multiple views for project, vehicle, and user management, as well as an interactive map. In this section, all features and how they relate to each other is described.

4.4.1 Authentication

The use cases include two user roles, drivers and planners. In order to differentiate between these roles, authentication and authorization was a requirement. In addition, user information is intended to be stored as audit data on various entities. To support these use cases, a user database is set up. The user account table includes hashed passwords, allowing users to log in using email and password. Successful sign ins will produce a longer lived refresh token and a shorter lived access token. The tokens can be used to identify the user, where the access token's purpose is authorization for endpoints. Refresh tokens, on the other hand, are meant to be used to retrieve new access tokens. This allows users to stay logged in unless the refresh token has expired. The tokens are JWTs and are signed using different secrets based on the type of token. Spring Security is used to handle authorization and is configured using annotations. The sign in form is shown in Figure 4.4.1.

4.4.1.1 Password resets

To allow users to change their passwords, a flow for resetting the user password is set up. The user may from the sign in screen navigate to the forgot password screen, allowing them to enter their email to receive a code for resetting their password. Confirming this code will allow the user to define a new password. The steps for resetting passwords are shown in Figure 4.4.1.


Sign in to inSight
E-mail
E-mail
Password
Password
Forgot password?
Sign in

‹ Back
Reset your password
Please enter the e-mail address you'd like your password reset code sent to
E-mail
E-mail
Send code

‹ Back
Enter confirmation code
Please enter the confirmation code that was sent to your e-mail at **dev@email.invalid**
(Not you?)
Confirmation code
Confirmation code
Confirm code

‹ Back
Enter new password
Please enter a new password
New password
New password
Confirm new password
Confirm new password
Reset password

Figure 4.4.1: Authentication workflow

4.4.2 File uploads

Certain features include the ability to upload images or files. In order to allow these uploads, a generic service for uploading files to buckets in MinIO is set up. The service enables users to upload files such as images which can be referred to in other entities. Files are stored with randomly generated names, and can be downloaded at a later stage using a reference. File uploads are handled through the backend to enable re-use of authorization logic. File uploads are enabled through the frontend using a custom-made Dropzone component, allowing users to drag and drop files into it or alternatively click it to select a file in a dialog.

4.4.3 Theme

The application supports changing its theme, while retaining iSi brand colors as accent colors. There are two themes defined, light and dark, which default to the system theme. The user can however choose to change this theme in a menu. It is intended that drivers should be able to use this application actively in the capture process, which implies that drivers may use this application in different light levels. Changing the theme of the application changes the shade of regular UI elements as well as inverts colors in the map. Scenarios where this may be useful include night work and tunnel driving. Figure 4.4.2 shows the different themes.

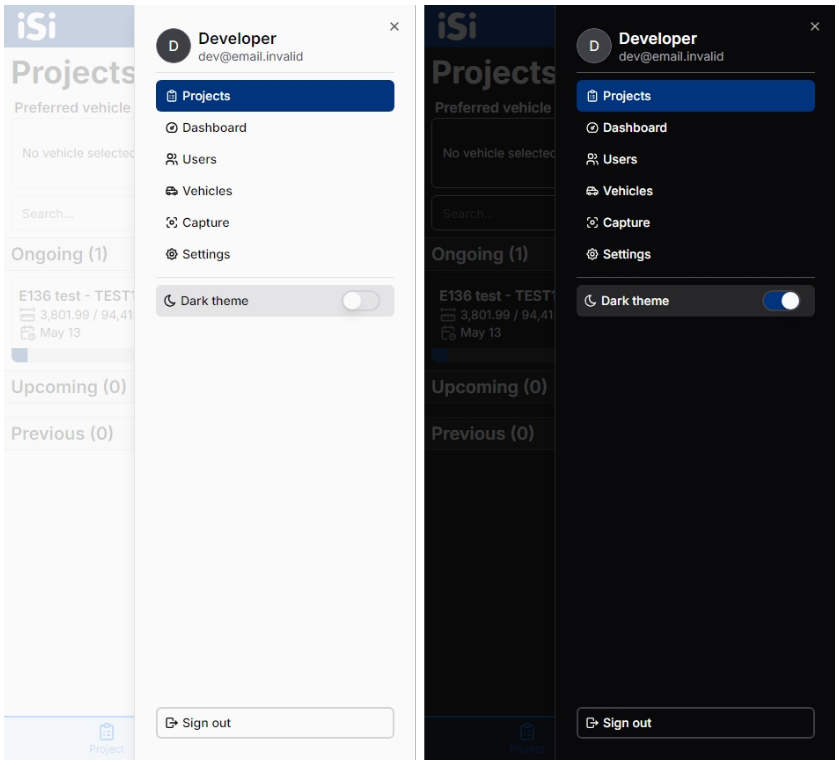
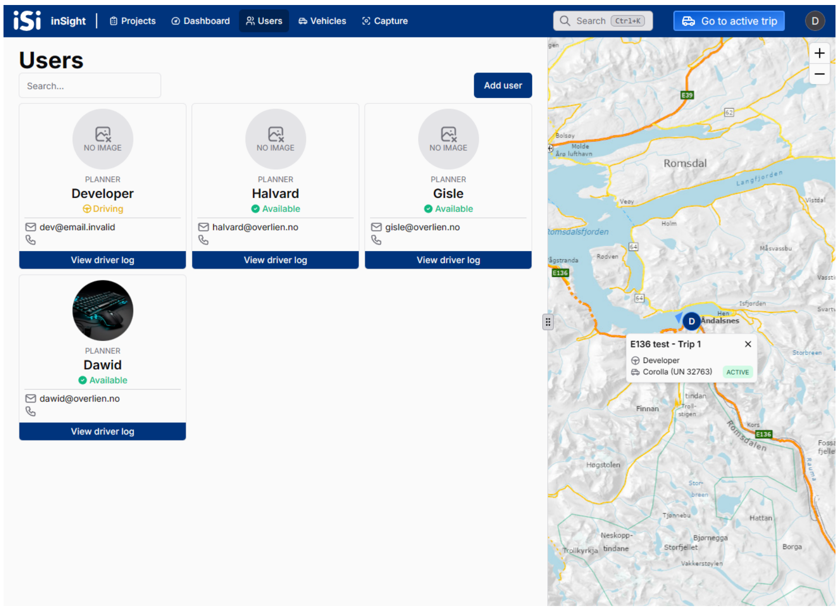


Figure 4.4.2: Toggling of dark theme

4.4.4 User management

The application includes the capability of managing user accounts. Planners can view and manage user accounts in dedicated pages listing their details as well as providing options for editing them. The position of each user is also displayed with markers in a map to the side of the user list. Additionally, a user's previous trips can be viewed in a list ordered by recency. The list includes indicators of the trips' results. Figure 4.4.3 shows how users are listed, as well as the map with position markers. Figure 4.4.4 shows the page for editing users, as well as viewing their previous trips.



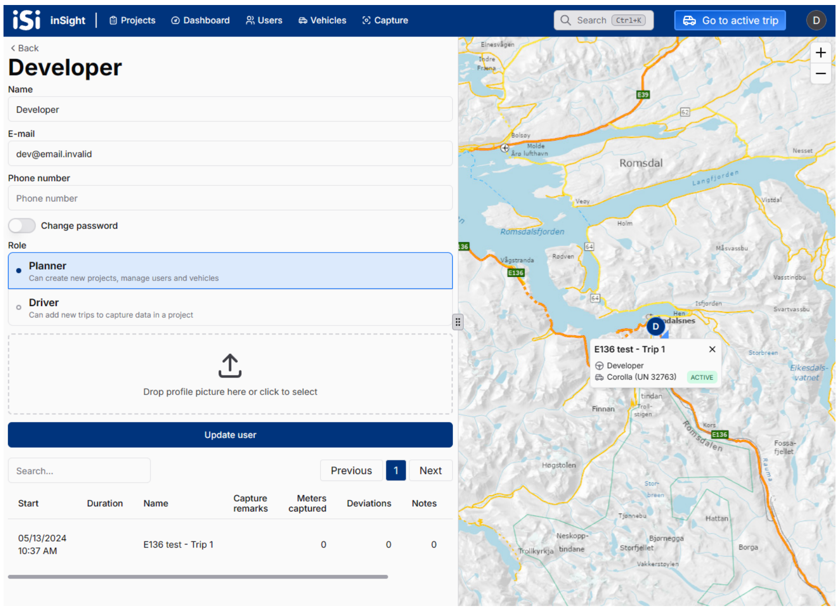
Users

Search...

NO IMAGE PLANNER Developer Driving dev@email.invalid	NO IMAGE PLANNER Halvard Available halvard@overlien.no	NO IMAGE PLANNER Gisle Available gisle@overlien.no
PLANNER Dawid Available dawid@overlien.no		

E136 test - Trip 1
Developer
Corolla (UN 32763) ACTIVE

Figure 4.4.3: User overview page



Developer

Name: Developer

E-mail: dev@email.invalid

Phone number: Phone number

Change password

Role

- Planner — Can create new projects, manage users and vehicles
- Driver — Can add new trips to capture data in a project

Drop profile picture here or click to select

Search...

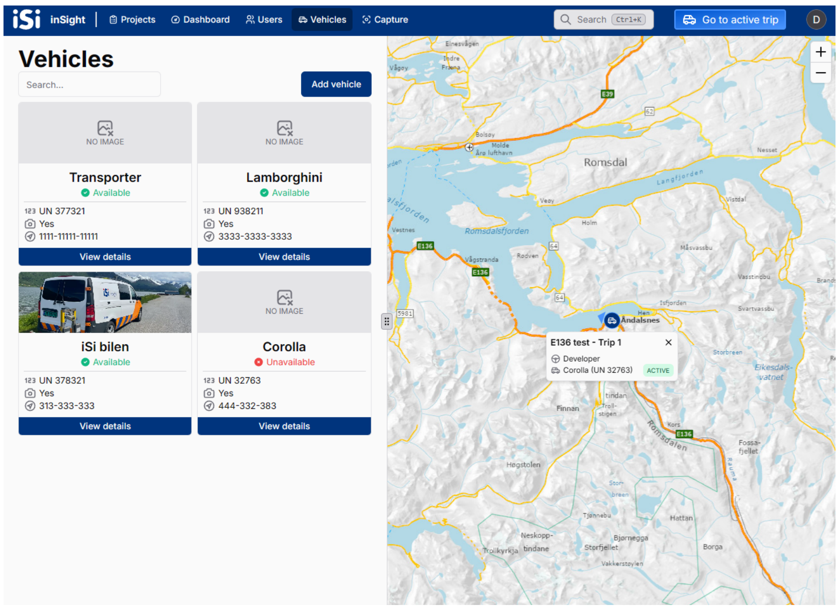
Start	Duration	Name	Capture remarks	Meters captured	Deviations	Notes
05/13/2024 10:37 AM		E136 test - Trip 1		0	0	0

E136 test - Trip 1
Developer
Corolla (UN 32763) ACTIVE

Figure 4.4.4: User details page

4.4.5 Vehicle management

Similar to managing user accounts, vehicles can be managed in the application. Planners can view and manage vehicles in dedicated pages for browsing and editing vehicle details. Vehicles can also be viewed with position markers in the map, to the side of the list. Editing vehicles and viewing the vehicles' usage in trips is also possible in a similar way to how it is done for user management. Figure 4.4.5 highlights how the vehicles are listed in the application.



Vehicles

Transporter
Available
UN 377321
Yes
1111-11111-11111

Lamborghini
Available
UN 938211
Yes
3333-3333-3333

iSi bilen
Available
UN 378321
Yes
313-333-333

Corolla
Unavailable
UN 32763
Yes
444-332-383

E136 test - Trip 1
Developer
Corolla (UN 32763) ACTIVE

Figure 4.4.5: Vehicle overview page

4.4.6 Project management

The main view in the application is the project view. It is built using a three level hierarchy consisting of projects, project plans and trips. This system, including its naming scheme, was agreed upon after discussion with iSi, as this is what they are currently familiar with. A project consists of one or several project plans, and a project plan consists of one or several trips. Project plans are multi-day work orders, which consists of a set of guardrails to capture. Trips are the work trips drivers go on to capture the guardrails assigned to a project plan.

Projects are categorized by a computed status. The statuses for a project are based on the current position in terms of their defined duration interval. A project may be defined without an end-date, but generally if the current date is between the project duration interval it is considered ongoing. Otherwise, it is either considered previous or upcoming based on the same interval. These statuses power a grouping mechanism, where lists of projects can be expanded by their status. Figure 4.4.6 shows how this grouping mechanism works for browsing projects.

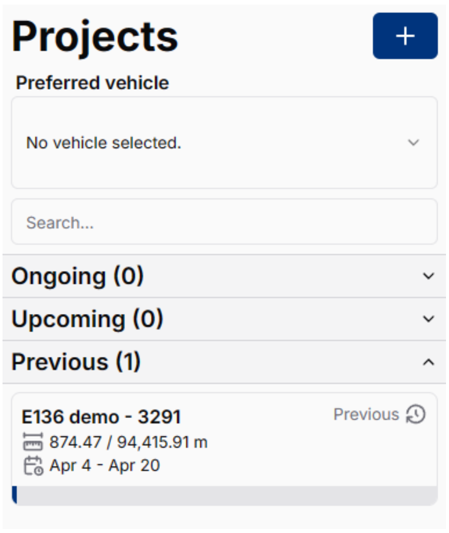
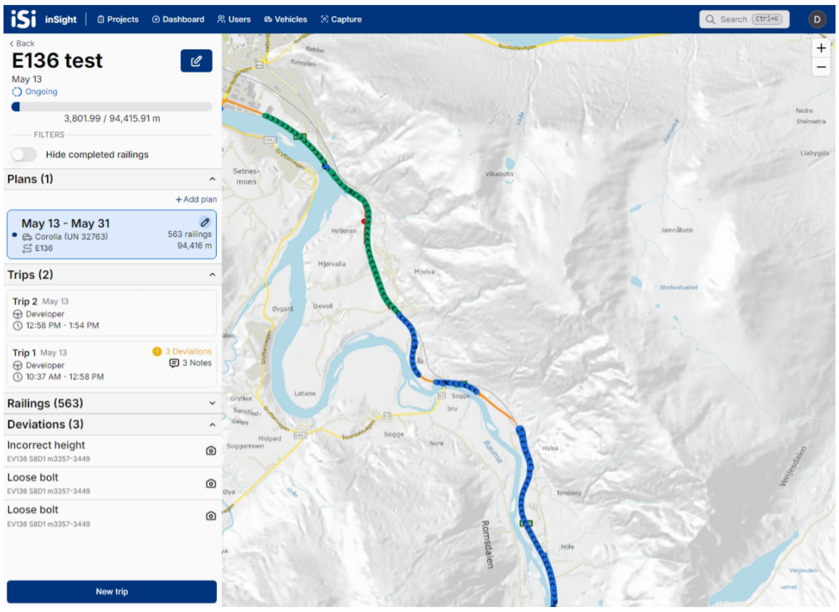

Projects
+
Preferred vehicle
No vehicle selected.
Search...
Ongoing (0)
Upcoming (0)
Previous (1)
E136 demo - 3291 Previous
874.47 / 94,415.91 m
Apr 4 - Apr 20

Figure 4.4.6: Project browsing menu with projects grouped by status

When viewing a project, its related data is displayed using a similar grouping mechanism. These groups show the project plans, trips and guardrails. Selecting one or more plan will cause both the trip and guardrail data to filter using the selected plans. The project view with this grouping mechanism, as well as with a plan selected to filter, is highlighted in Figure 4.4.7.



< Back
E136 test
May 13
Ongoing
3,801.99 / 94,415.91 m
FILTERS
Hide completed railings
Plans (1)
+Add plan
May 13 - May 31
Corolla (UN 32763) 563 railings
E136 94,416 m
Trips (2)
Trip 2 May 13
Developer
12:58 PM - 1:54 PM
Trip 1 May 13 3 Deviations
Developer 3 Notes
10:37 AM - 12:58 PM
Railings (563)
Deviations (3)
Incorrect height
EV136 S8D1 m3357-3449
Loose bolt
EV136 S8D1 m3357-3449
Loose bolt
EV136 S8D1 m3357-3449
New trip

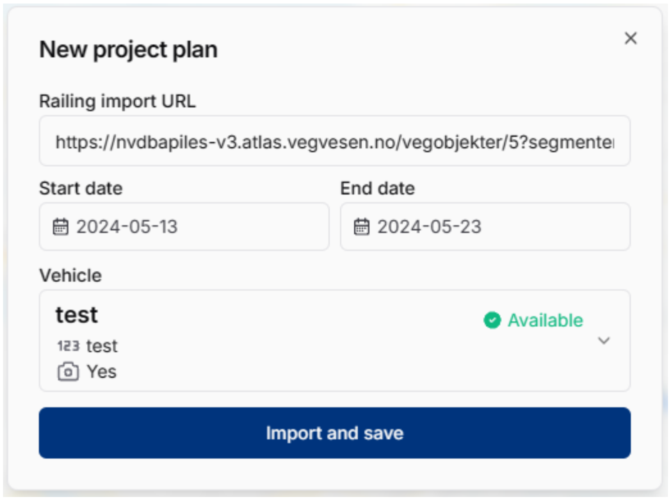
Figure 4.4.7: Frontend project view with a selected plan

4.4.7 Guardrail data import

The main requirement for the project was the planning aspect. As highlighted by the use case diagram, NVDB already provides an application where users can filter and find road objects effectively. This application is called Vegkart. Replicating the functionality already provided by Vegkart would be redundant, since it already works for this part of the planning workflow. As a result, it was decided that the planning tool should accept a reference API URL for a given selection that can be used to import data from NVDB.

These API URLs can be found by the planner in the Vegkart interface, allowing for copy and paste directly to the planning UI. A custom-made API client made using Spring HTTP interfaces uses the given reference URL to copy parameters, though with some overrides such as `inkluder=alle` to include all data.

Importing can be done both when creating a project plan and when updating it. The import will use the reference URL to import necessary guardrail data as well as attached road segments and persist it in the database. When updating a project plan, guardrails can optionally be re-imported in case of errors and replaces previously stored guardrails for a plan. Figure 4.4.8 shows the dialog for creating a project plan.


New project plan

Railing import URL: https://nvdbapiles-v3.atlas.vegvesen.no/vegobjekter/5?segmente

Start date: 2024-05-13

End date: 2024-05-23

Vehicle: test, 123 test, Yes — Available

Figure 4.4.8: Dialog shown when creating a new project plan

4.4.8 Guardrail geometry directions

Imported guardrail data from NVDB were in some cases observed to have reversed geometries. One of the goals in the project was to render the driving direction required to capture a guardrail. However, in some cases, the geometry of the guardrail was opposite to the driving direction. This reversion impacted rendering the driving direction for guardrails since it is inferred by the order of the points in the geometry. To mitigate this, an algorithm that compares points in the guardrail geometry with points from attached road segments has been tested. However, the flipped direction appeared to be random resulting in geometries not being possible to correct without deeper analysis of metadata from guardrails and its attachment to road segments. Figure 4.4.9 shows how guardrail geometries are rendered in the wrong direction when compared to the road networks direction, where a guardrail on the opposite side of the road is rendered with arrows pointing in the same direction.

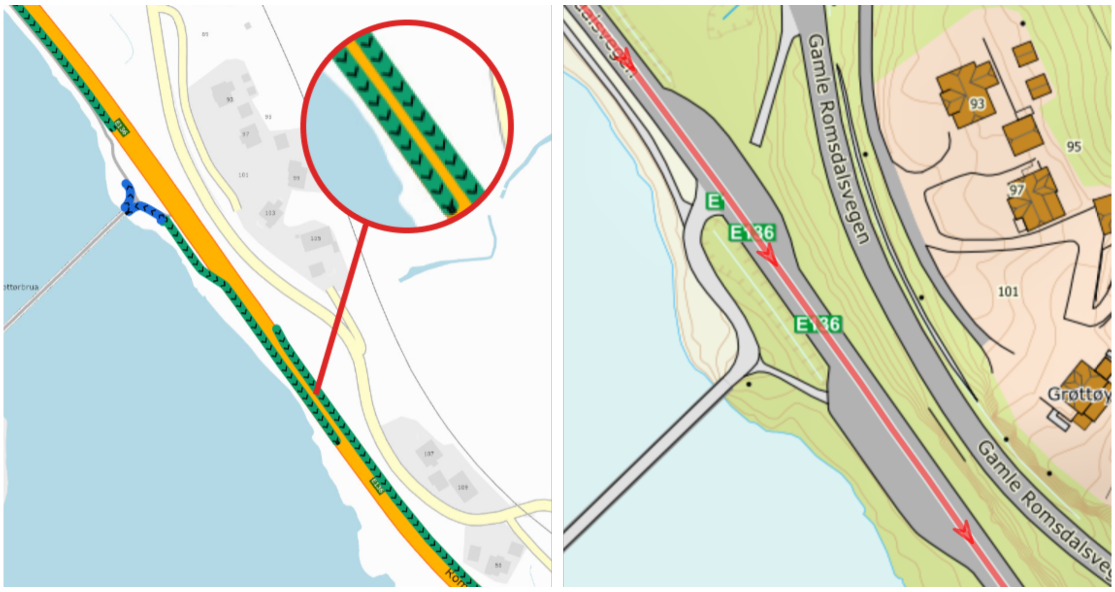


Figure 4.4.9: Comparison of railing geometry directions with road network direction

4.4.9 Processing capture logs

The data capture process produces logs that need to be parsed and processed in order to allow displaying capture progress. The GNSS and each of the cameras produce their own separate log files. The condition for considering a meter of a guardrail's segments captured is the existence of an image in the correct direction. As a result there is a need to merge these log files into a format that contains information about all these parts to proceed with matching the capture to railings. The log files are processed into an intermediary structure containing information gathered from the different logs. Figure 4.4.10 shows the logic behind processing these capture logs into the intermediary structure.

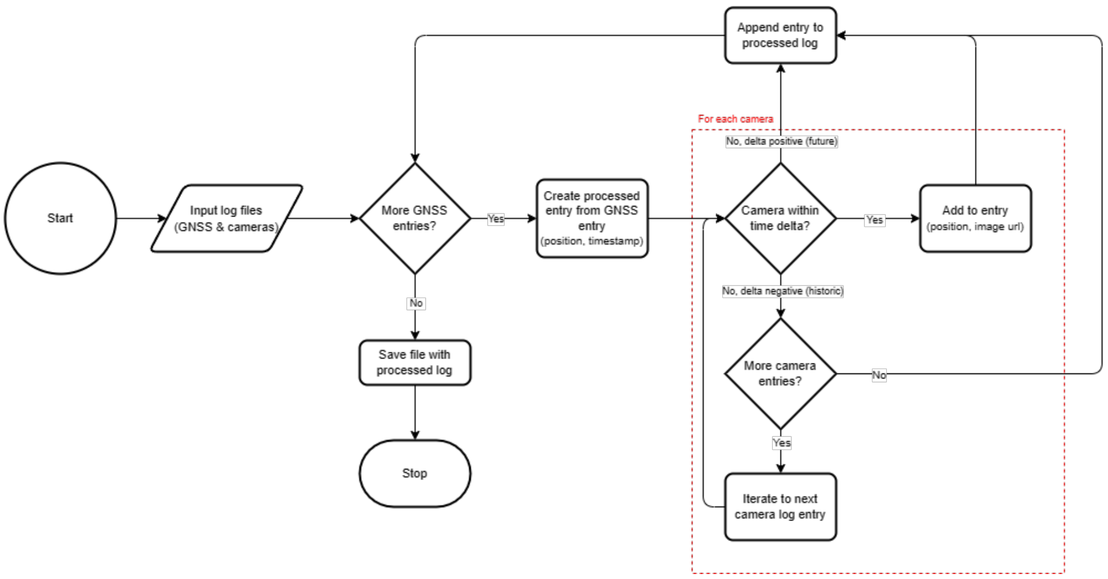
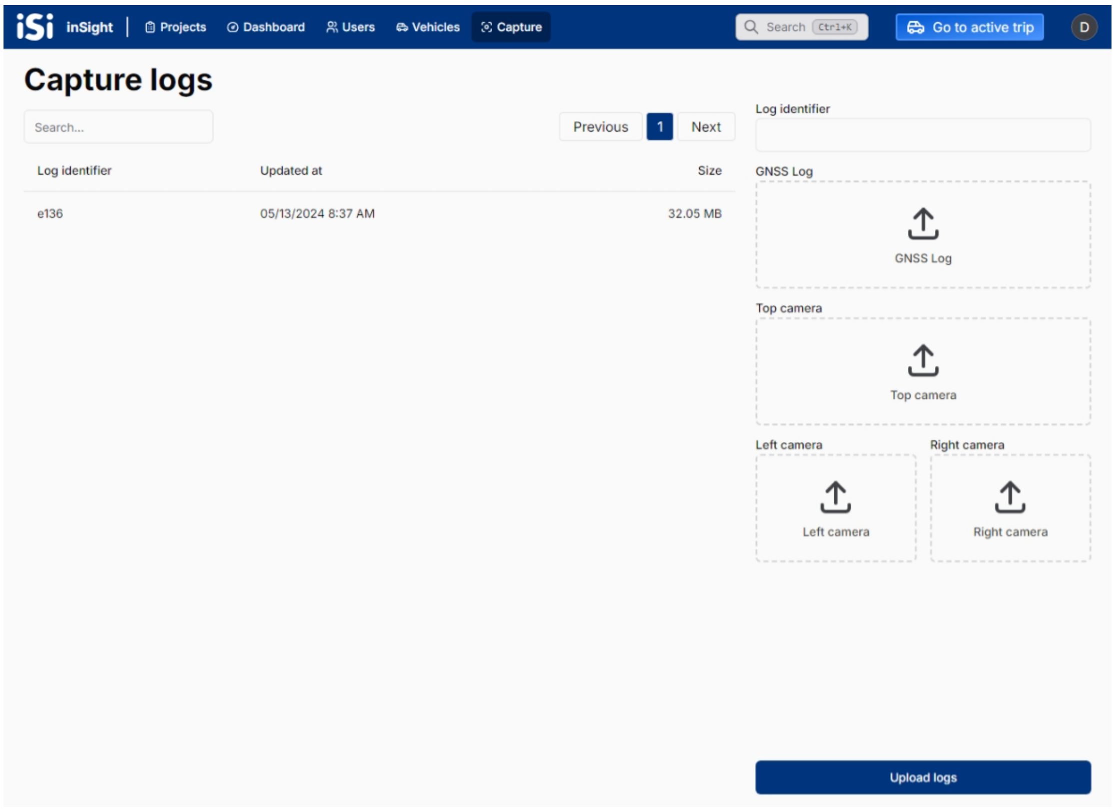

Start
Input log files (GNSS & cameras)
More GNSS entries?
Yes
No
Create processed entry from GNSS entry (position, timestamp)
Save file with processed log
Stop
Append entry to processed log
For each camera
No, delta positive (future)
Camera within time delta?
Yes
Add to entry (position, image url)
No, delta negative (historic)
More camera entries?
No
Yes
Iterate to next camera log entry

Figure 4.4.10: Flow chart displaying the logic used when processing capture logs

4.4.9.1 Uploading capture logs

To simplify the process of receiving feedback from iSi, a feature in the web application that allows for uploading of capture logs is implemented. This feature allows users with the necessary GPS and camera logs to upload necessary data to simulate a replay. When uploading a new capture log, users encounter four designated file inputs for submission of necessary files. This feature was essential for establishing a complete workflow loop within the application, making it possible for iSi to test and provide feedback. A page for uploading and viewing uploaded capture logs is shown in Figure 4.4.11.



Log identifier	Updated at	Size
e136	05/13/2024 8:37 AM	32.05 MB

Figure 4.4.11: Page for uploading and viewing uploaded capture logs

4.4.10 Trips and replays of capture logs

Trips and the execution of them are a core part of the capture process. In order to verify the concept for executing these without integrating with the car, a replay system is introduced. This replay system is intended to be replaced by an in-car solution, which can give the driver real-time feedback on critical metrics. Drivers are able to choose which of the processed capture logs to use as a replay for the trip. The form in the dialog will default to their set preferred vehicle, which can be selected in the project overview, displayed when logging in. The driver can optionally change this vehicle without overriding the preference. The dialog a driver is shown when starting a trip is shown in Figure 4.4.12.

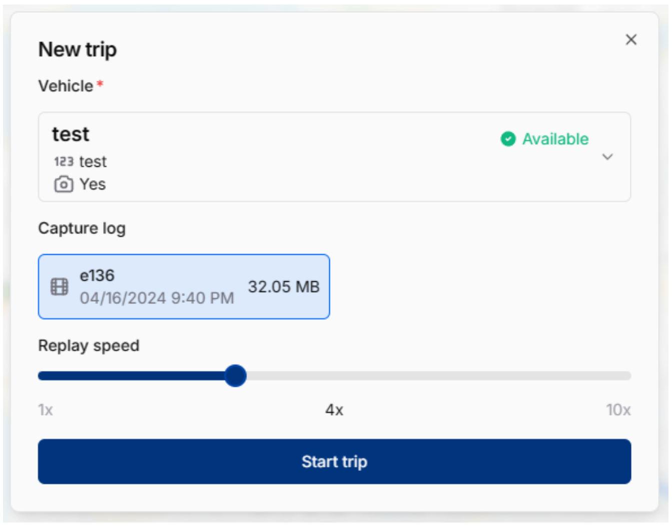


Figure 4.4.12: Dialog shown when starting a new trip

The replay system steps through the processed logs in a configurable speed. The stepped through log entries are emitted as events. A service managing these replays subscribes to these events and forwards details from the capture to the frontend using server-sent events. The forwarded events inform the frontend about the latest position in the capture, along with additional metrics, such as the GPS signal, and the amount of images and meters covered. The frontend updates a signal containing the latest capture details allowing both the map and indicators to re-render with this information. The indicators in Figure 4.4.13 are those updated by the latest capture details, in addition to position updates to the vehicle marker.

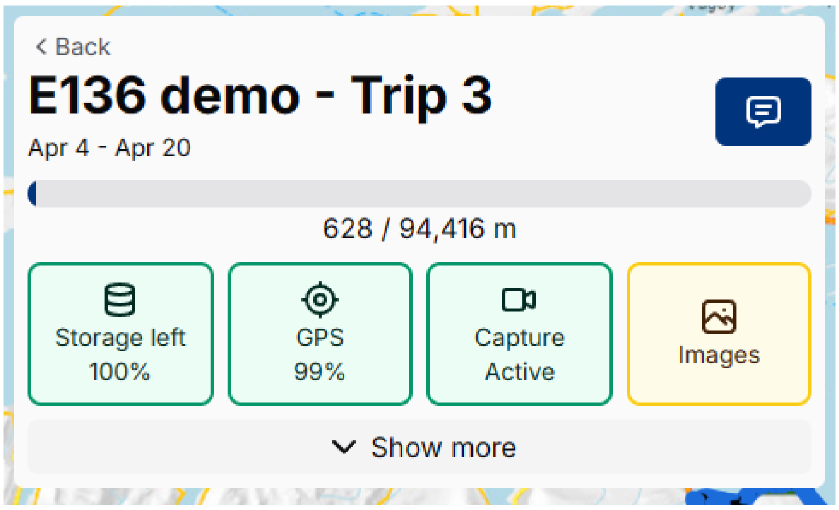
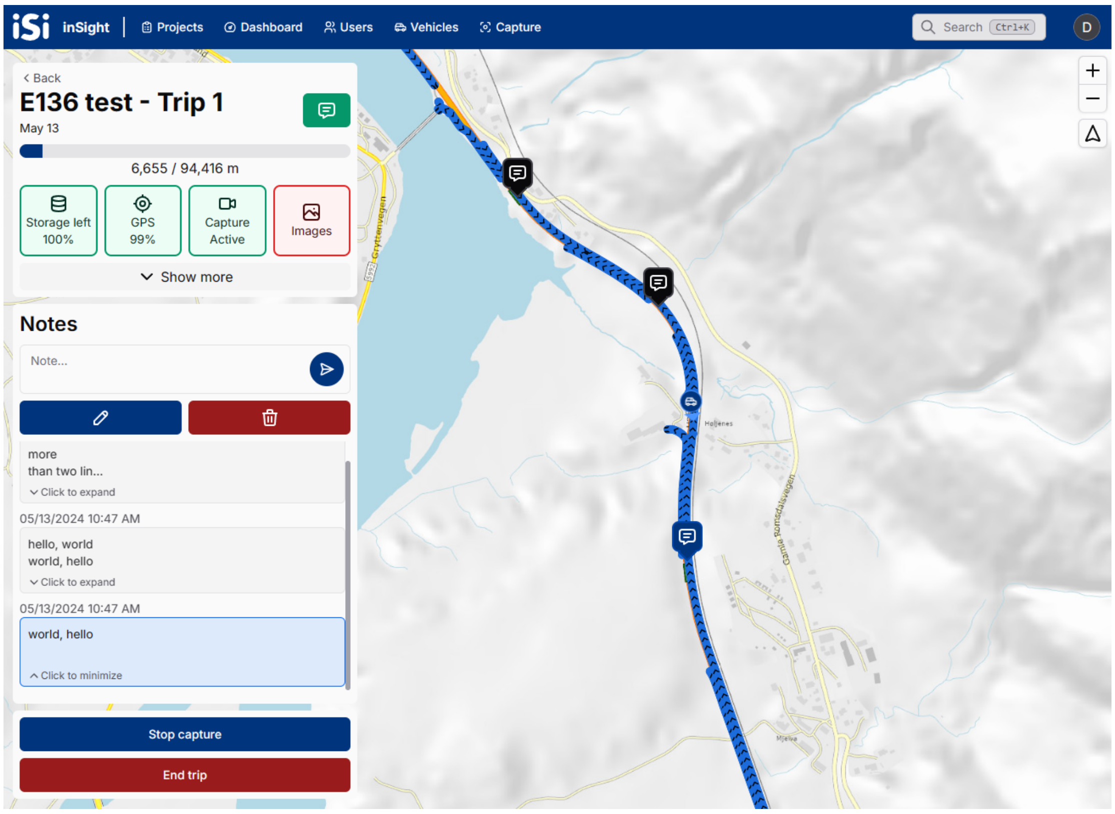


Figure 4.4.13: Indicators displayed in the trip view

In addition to these events being forwarded to subscribed clients, they are also emitted internally. Capture events are emitted at the same rate internally, allowing other parts of the system to subscribe to changes in position for ongoing replays. There is also an internal API available for getting the latest capture details. This API makes it possible to store the latest details from a trip's capture, making it accessible after a trip has ended.

4.4.11 Trip notes

As part of the trip execution, drivers have the ability to add notes. In cases of events occurring during a trip, drivers can add a note to it describing the event. Drivers may also edit and delete notes after creation. Using the latest position from the replay, the notes are displayed in the map with synced select interactions. This allows the user to press a note in the map, resulting in it being highlighted in the list or vice versa. Figure 4.4.14 shows the selection of a trip notes marker, with the appropriate list element also highlighted.



more
than two lin...
Click to expand

05/13/2024 10:47 AM
hello, world
world, hello
Click to expand

05/13/2024 10:47 AM
world, hello
Click to minimize

Figure 4.4.14: Trip note markers and selection of them

4.4.12 Vehicle and user status

Both vehicles and users have statuses. These statuses are inferred from their assignment to ongoing events. Vehicles may be assigned to a plan, leading to its availability changing in that time period. Users, on the other hand, only change their status based on whether or not they are currently involved in an active trip. When users are on a trip, a button for navigating to the active trip is shown in the header. This button is only displayed when the user is not in the current trips view. The button is shown in Figure 4.4.15.

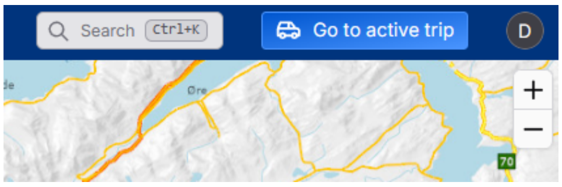


Figure 4.4.15: Button displayed when a user is outside their active trip

4.4.13 Matching capture data to guardrails

Capture data has to be matched to the planned set of guardrails. In order to visualize and track what has been captured, a process for matching captured data to guardrails is necessary. The application uses log entries in the aforementioned intermediary structure in a procedure that compares the data to the guardrails previously imported. The procedure checks whether or not a shape representing the camera's FOV intersects a railing using JTS, Figure 4.4.16 highlights how this matching works. This works for guardrails that have their own geometry, however for those who do not, special processing is needed. The procedure returns a match result containing the side of which it was matched on, which is inferred in two ways. For railings with their own geometry, the camera FOV side is used. However, for those without own geometry the guardrail placement metadata is used in combination with the driving direction according to the nearest road segment.

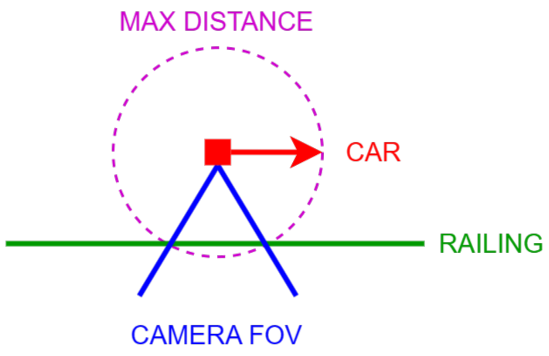


Figure 4.4.16: Diagram displaying how guardrails are matched to captures

Trigonometry is used to infer the driving direction of the vehicle with respect to the road segment. The matching process finds two nearby points from the segment, and uses it as a vector. Using this vector, it will compare the angle of the vector to the angle of the heading direction of the car as reported by the GNSS system. If these two angles are within 180 degrees of each other, they are considered to be in the same direction. Figure 4.4.17 shows how this inference works graphically.

The matching process produces coverage metrics. Matching the railing will yield two length coverage ranges for the railing, as well as one length coverage range for the segment. As parts of the railings may be unreachable, progress is measured in terms of the road segments. These ranges are approximate start and end indexes along the length of the geometries. For segments a range is generally 1 meter long, with the exception of starts and ends of geometries. While for a railing the coverage is the start and end index of the intersecting lines from the camera FOV shape.

A 2D intersection may not be enough to identify railing matches in all scenarios. The elevation of a car should generally be within a given delta to eliminate potentially matching railings at the wrong elevation when crossing overpasses or bridges. To mitigate this, the elevation of a car is compared to the nearest segment. Since the car elevation is measured in ellipsoid height by the GNSS, it is transformed into the NN2000 vertical CRS used in SRID 5973 using a height reference model provided by The Norwegian Mapping Authority using the approach outlined in Section 4.3.

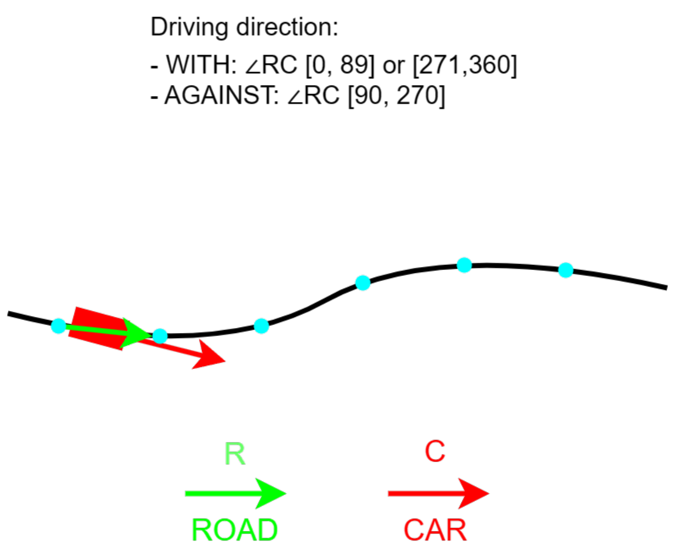


Figure 4.4.17: Diagram displaying the comparison of the cars vector and the roads vector

4.4.14 Calculating guardrail capture grade

The capture data produced by the matching process is used to determine the capture grade of guardrails. Capture grade is measured by the sum of all merged segment coverage ranges compared to the length of the segments attached to a railing. Figure 4.4.18 displays how the ranges are merged. The road segments attached to a railing are the stretches that are expected to be driven, which is especially the case in scenarios where the distance to a railing is too large to capture it from the road. The completion grade is used to set railing status to one of TODO, ERROR or OK as shown in Figure 4.4.19. The threshold for considering a guardrail captured, or OK, is 95%, where anything less than 95% but greater than zero is considered ERROR.

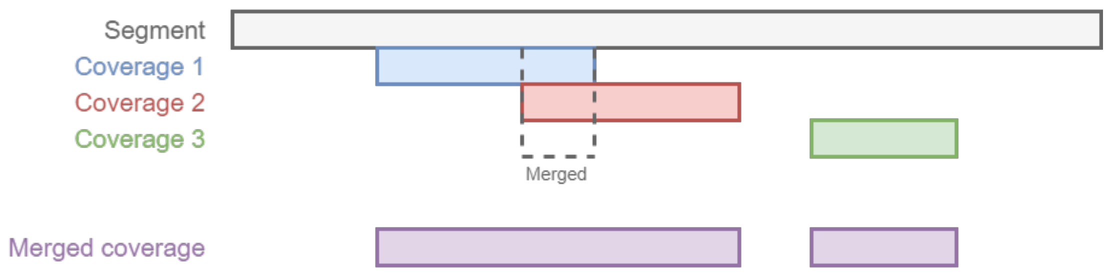


Figure 4.4.18: Merging of segment coverage ranges

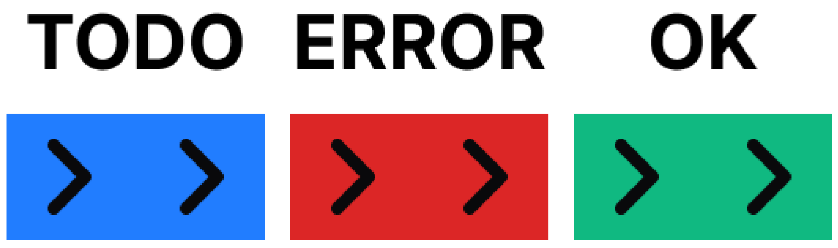


Figure 4.4.19: Guardrail variants based on completion grade

4.4.15 Interactive map for viewing guardrails

To allow both drivers and planners to get an overview of the task at hand, an interactive map for viewing guardrails to be captured is implemented. The guardrails include hover effects to emphasize the ability to click them to show a popup where its details are shown. Figure 4.4.20 shows how the railings are rendered in the frontend, along with its popup shown on press.

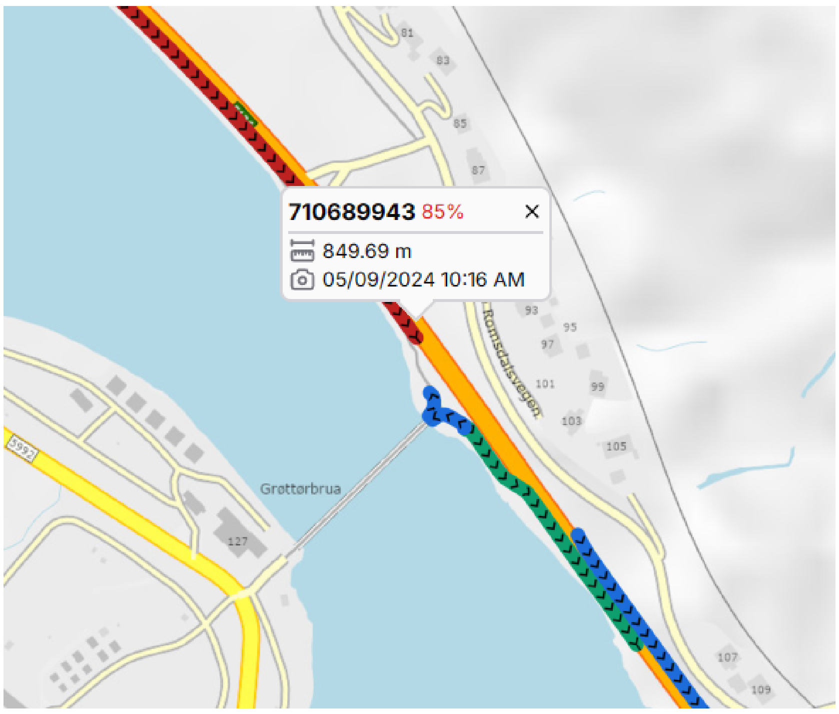


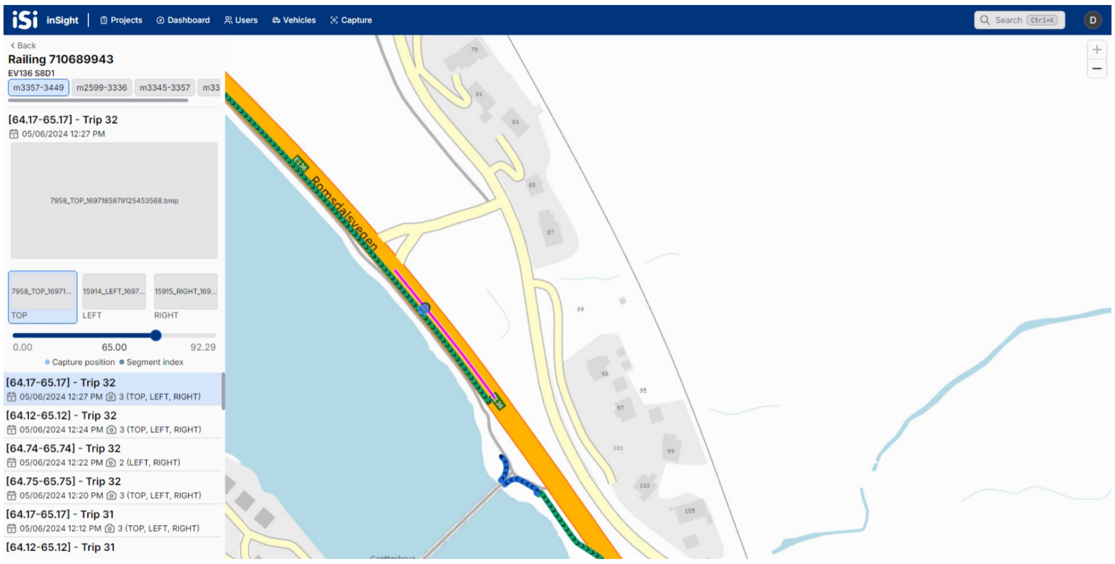
Figure 4.4.20: Rendering of guardrails in the map where one is clicked

4.4.16 Viewing images captured of guardrails

After a capture has been stored, the images from that capture can be viewed in the frontend application. Due to the lack of actual image data, the previews are placeholders with the names of the files from the given location instead. The images are divided into the stretches related to guardrails, and users can use a slider to find images from an approximate location along that stretch as shown in Figure 4.4.21. This is also shown in the video in Appendix F.

4.4.17 Position updates for users and vehicles

A service for positions is implemented by listening to capture events. Capture events emitted by the replay system is subscribed to by a position service, which holds the latest position for both vehicles and users. These positions are keyed separately for vehicles and users, allowing defining a subscription to specific keys. The positions stored in the service are subscribed to using server-sent events in both the vehicle and driver pages, as well as the dashboard. The state of this service is kept in-memory, not saved to persistent storage. Various uses of the position updates can be seen in the video in Appendix F.



Railing 710689943
EV136 S8D1
m3357-3449 m2599-3336 m3345-3357 m33

[64.17-65.17] - Trip 32
05/06/2024 12:27 PM
7958_TOP_1697185879125453568.bmp
7958_TOP_16971... 15914_LEFT_1697... 15915_RIGHT_169...
0.00 65.00 92.29

[64.17-65.17] - Trip 32
05/06/2024 12:27 PM 3 (TOP, LEFT, RIGHT)
[64.12-65.12] - Trip 32
05/06/2024 12:24 PM 3 (TOP, LEFT, RIGHT)
[64.74-65.74] - Trip 32
05/06/2024 12:22 PM 2 (LEFT, RIGHT)
[64.75-65.75] - Trip 32
05/06/2024 12:20 PM 3 (TOP, LEFT, RIGHT)
[64.17-65.17] - Trip 31
05/06/2024 12:12 PM 3 (TOP, LEFT, RIGHT)
[64.12-65.12] - Trip 31

Figure 4.4.21: Screenshot from the UI displaying how images of guardrails can be viewed

4.4.18 Follow mode for in-vehicle viewing

Maintaining a good overview of remaining guardrails while driving can be challenging. To avoid requiring the driver to continuously update the maps center and zoom, a mode where the map view centers around the cars position is developed. In addition to zooming, the mode will rotate the map, ensuring the heading direction of the vehicle points upwards in the map. How this mode works can be seen in the video in Appendix F.

4.4.19 Responsive layouts

To support usage on mobile and tablet devices as well as desktops, layout shifts are configured with various screen size breakpoints. The breakpoints collapse and expand content in a number of ways to make content presentable for the current screen size. In some scenario,s separate presentations are made, such as the mobile navigation menu shown in Figure 4.4.22.

In addition to layout adjustments for supporting smaller viewports, resizable panels are added to some pages. Both the user and driver pages utilize resizable panels, allowing planners to customize the layout to increase or decrease size of the map by dragging a handle between panels. Figure 4.4.23 shows the user page configured with different layouts using the resizable panel handle. The configured sizes are persisted locally in the browser, and restored on page load. The behaviour of responsive layouts is showcased in the video in Appendix F.

4.4.20 State management

The URL and browser storage APIs are used for state in the frontend application. For both navigation state and filtering options such as hiding guardrails, the browser URL is used. In addition, preferences such as the preferred vehicle, use browser storage APIs to allow persisting these preferences across application loads. As a result, a user can refresh the page and resume with the same settings. An exception to this is form state, which is intentionally in-memory and not recoverable between refreshes.

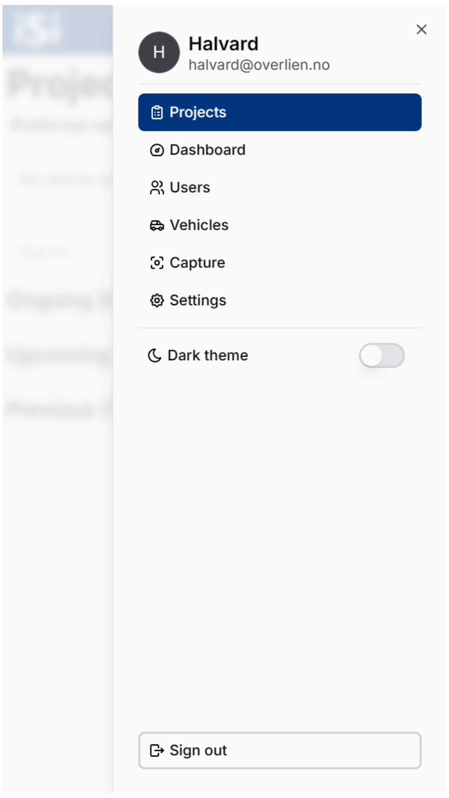


Figure 4.4.22: Navigation menu for small or mobile viewports

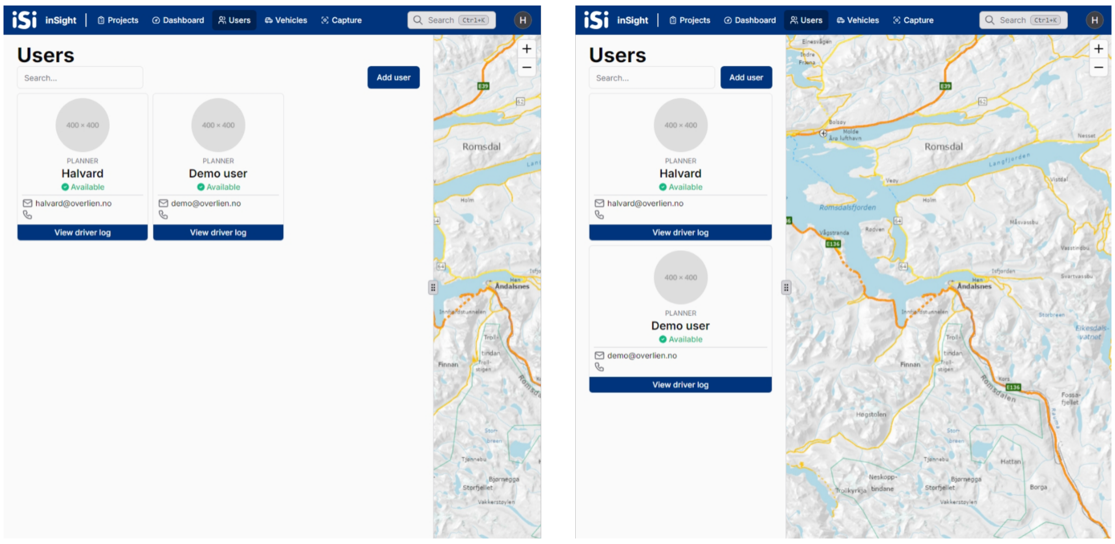


Figure 4.4.23: Responsive layout used to configure the users page with different map widths

4.4.21 Reporting deviations

For iSi to be able to track deviations in the same system, an API has been prepared for reporting deviations to the application. The API is designed to be simple, with the goal of illustrating how it can be integrated with the matched railing capture data. The deviations reported can be expanded in the project details view, as shown in Figure 4.4.24. The stretch in which the deviation was discovered is displayed, along with the type. These deviations can be clicked to navigate to the capture where the deviation was identified, the page discussed in Subsection 4.4.16.

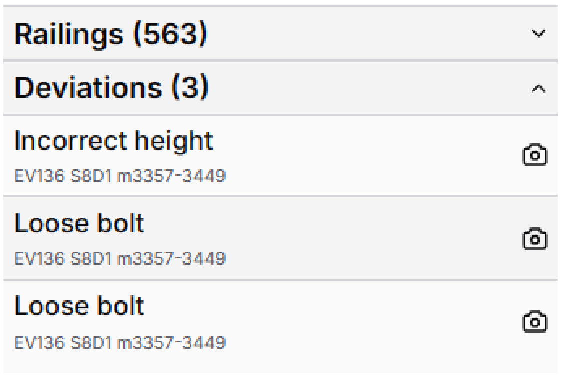

Railings (563)
Deviations (3)
Incorrect height
EV136 S8D1 m3357-3449
Loose bolt
EV136 S8D1 m3357-3449
Loose bolt
EV136 S8D1 m3357-3449

Figure 4.4.24: List of deviations displayed after they have been reported in the UI

4.4.22 Dashboard

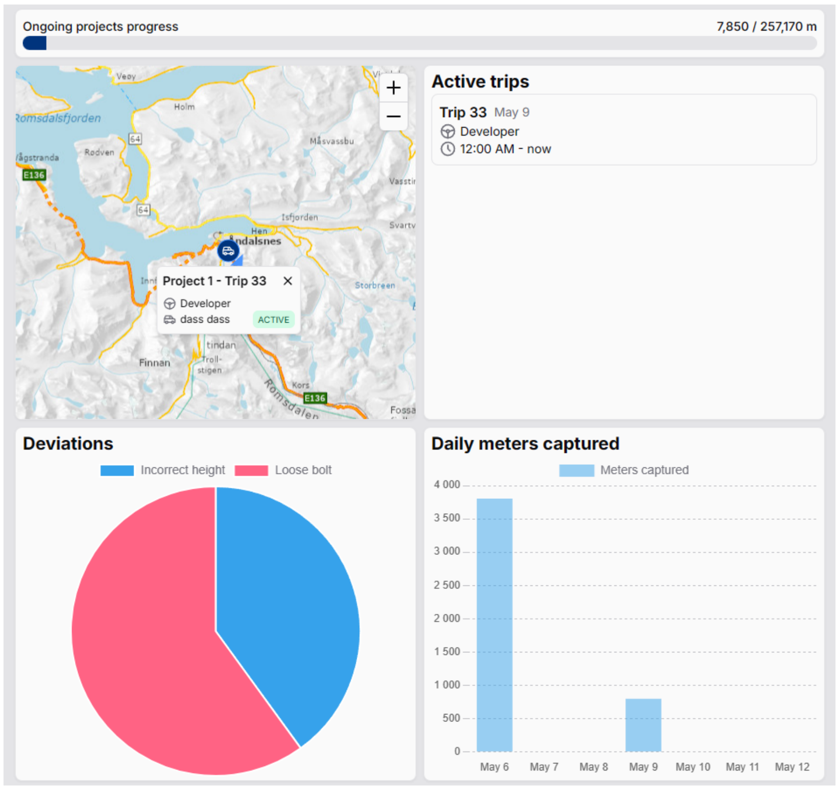
To allow planners to gain a quick overview of the current work status, a dashboard has been prototyped. The dashboard displays an aggregate progress for all ongoing projects, as well as the latest positions of vehicles and active trips. In addition to this, deviations reported the last week as well as meters captured by day is presented in graphs. Figure 4.4.25 shows the dashboard highlighting these features.

4.4.23 Global search

The number of railings may be large, thus finding a specific railing may be challenging. In order to simplify the process of finding information with knowledge of identifiers for either road segments or road railings, users can search in a menu that queries across multiple entities. The same search menu can be used to search for users, projects and vehicles. The search is an SQL query that returns a polymorphic JSON structure using a discriminated union. Figure 4.4.26 shows how results are displayed when searching using this menu. The results shown can be navigated to using both mouse and keyboard, where clicking or pressing enter navigates to the selected search result.

4.4.24 Internationalization

Though not a requirement, strategies for internationalization are used. Text values in the frontend refer to keys in a translation file. To format numeric values, the JavaScript Intl global object is used, using the configured locale. To format date values, locale-specific formatters are used in day.js. These functions are exposed through a re-usable hook.



Ongoing projects progress 7,850 / 257,170 m

Active trips

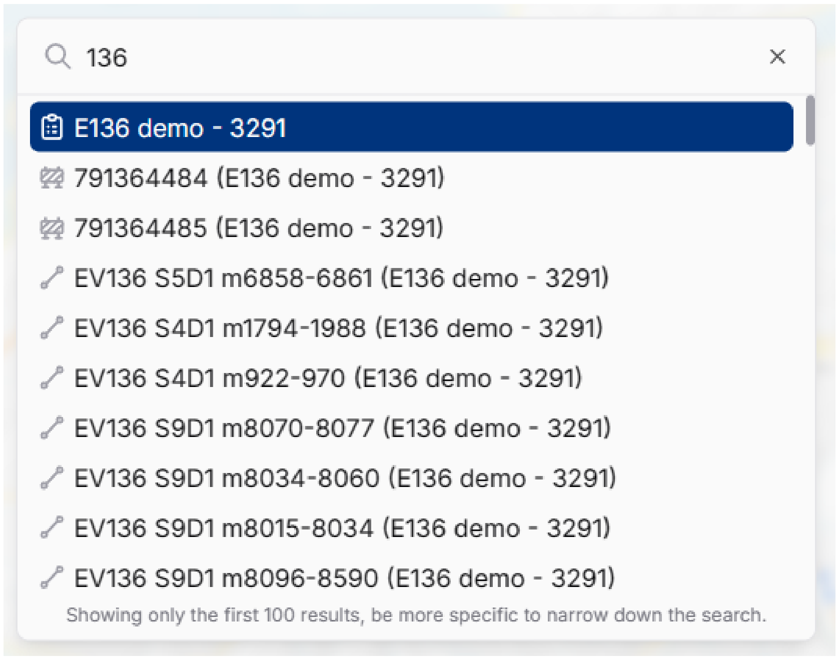
Trip 33 May 9
Developer
12:00 AM - now

Project 1 - Trip 33
Developer
dass dass ACTIVE

Deviations

Daily meters captured

Figure 4.4.25: Dashboard UI with widgets displaying key information



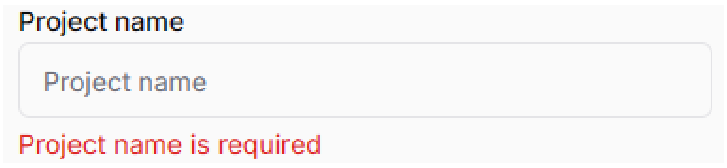
136

E136 demo - 3291
791364484 (E136 demo - 3291)
791364485 (E136 demo - 3291)
EV136 S5D1 m6858-6861 (E136 demo - 3291)
EV136 S4D1 m1794-1988 (E136 demo - 3291)
EV136 S4D1 m922-970 (E136 demo - 3291)
EV136 S9D1 m8070-8077 (E136 demo - 3291)
EV136 S9D1 m8034-8060 (E136 demo - 3291)
EV136 S9D1 m8015-8034 (E136 demo - 3291)
EV136 S9D1 m8096-8590 (E136 demo - 3291)
Showing only the first 100 results, be more specific to narrow down the search.

Figure 4.4.26: Frontend UI for global search

4.4.25 Input validation

To prevent illegal input data from being stored, multiple approaches are used. Both the frontend and the backend code utilize input validation. As a measure to provide the user with quick feedback on the validity of forms, the frontend validates forms using Zod in combination with error labels located near the user input. An example can be seen in Figure 4.4.27. In addition to the frontend validation, the backend will also validate the request from the frontend. As a third measure to validate, database constraints are utilized. However, in normal use, these constraints are not visible due to the workflow of the application.


Project name

Project name

Project name is required

Figure 4.4.27: Sample validation error displayed in frontend when submitting forms

4.4.26 Error handling

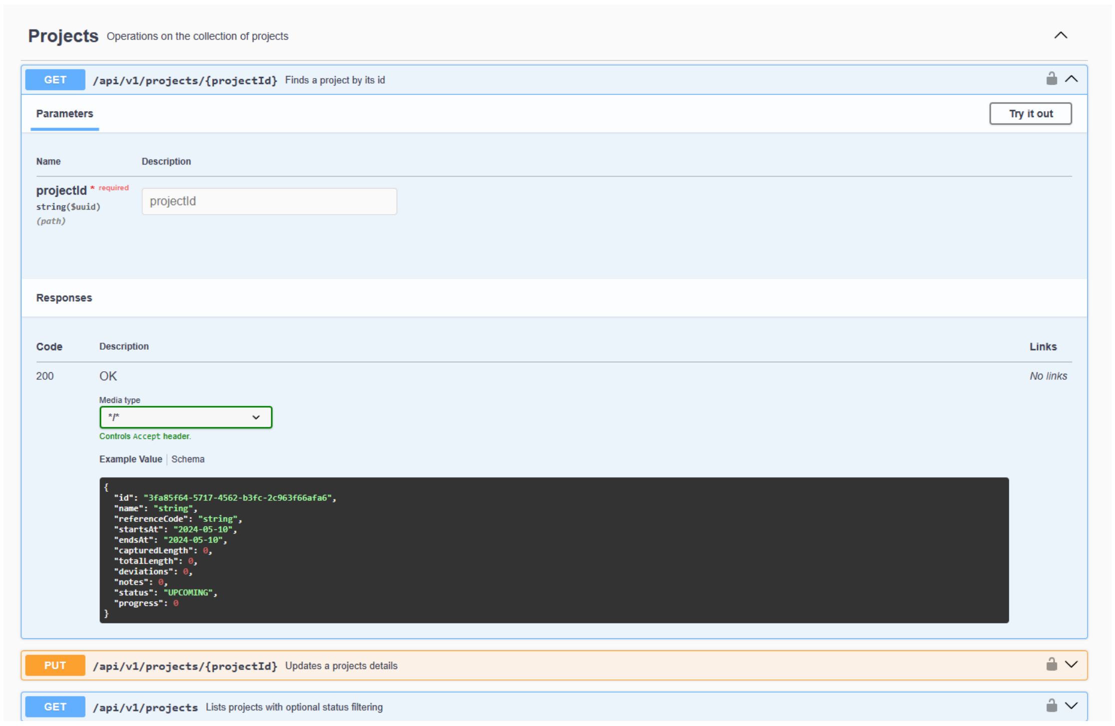
The application aims to inform users of why something has failed. While the aforementioned form validation provides error labels in the forms, errors are also handled in the backend. Backend errors are formatted using the Problem Detail specification and help the consumer identify why their request has failed. An example of a Problem Detail response is shown in Listing 4.

```
{
  "type": "about:blank",
  "title": "Bad Request",
  "status": 400,
  "detail": "Request contains 2 validation error(s)",
  "instance": "/api/v1/projects",
  "errors": {
    "fields": ["name", "referenceCode"],
    "reasons": {
      "name": [
        {
          "message": "must not be blank"
        }
      ],
      "referenceCode": [
        {
          "message": "must not be blank"
        }
      ]
    }
  }
}
```

Listing 4: Example of Problem Detail response when request validation fails

4.4.27 API documentation

The application documents APIs in various ways. For internals, JSDoc is used for frontend code, while Javadoc is used in the backend. In addition to this, API endpoints in the REST API is documented using OpenAPI v3. The OpenAPI schema can be browsed using Swagger UI, which is available at `/api/swagger-ui.html`. Figure 4.4.28 shows how the defined collections and operations on them are rendered in the documentation.


Projects Operations on the collection of projects

GET `/api/v1/projects/{projectId}` Finds a project by its id

Parameters

Try it out

Name	Description
projectId * required `string($uuid)` *(path)*	projectId

Responses

Code	Description	Links
200	OK	*No links*

Media type: */*

Controls `Accept` header.

Example Value | Schema

```
{
  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "name": "string",
  "referenceCode": "string",
  "startsAt": "2024-05-10",
  "endsAt": "2024-05-10",
  "capturedLength": 0,
  "totalLength": 0,
  "deviations": 0,
  "notes": 0,
  "status": "UPCOMING",
  "progress": 0
}
```

PUT `/api/v1/projects/{projectId}` Updates a projects details

GET `/api/v1/projects` Lists projects with optional status filtering

Figure 4.4.28: API docs rendered using Swagger

4.5 Application deployment

As part of the thesis, a server was set up. This server is used as a demo environment where the application runs. In order to deploy the application a pipeline is set up in GitHub Actions. Creating a new tag in Git will trigger the pipeline. A Gradle task is defined to help align versions across the Gradle and PNPM workspaces, the task can be executed by running `./gradlew updateVersion -DnewVersion="1.0.0"`. The pipeline re-uses the CI workflow to run all tests, early exiting or failing the pipeline to prevent regressions in code covered by tests. The pipeline will publish container images to the GitHub Container Registry if both the tests and the build succeeds, proceeding to deploy a new version to the demo environment set up. The steps in the workflow for deploying new versions is shown in Figure 4.5.1. The environment runs a single node Kubernetes cluster using the K3s distribution. The cluster is installed and configured using Ansible. The deployment uses a Helm chart to template the resources, allowing for re-use across multiple environments in multi-stage deployments. An overview of the resources that are part of the Helm chart can be seen in Figure 4.5.2. Services are routed through a NGINX Ingress reverse proxy configured to use certificates issued by Let's Encrypt for TLS. These certificates are managed using cert-manager. The application hostnames are also registered in DNS using ExternalDNS to integrate with the used provider.

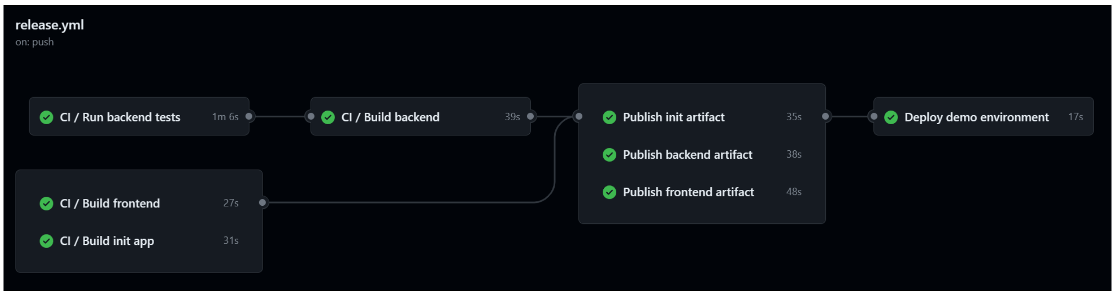

release.yml
on: push
CI / Run backend tests 1m 6s
CI / Build backend 39s
Publish init artifact 35s
Publish backend artifact 38s
Publish frontend artifact 48s
Deploy demo environment 17s
CI / Build frontend 27s
CI / Build init app 31s

Figure 4.5.1: GitHub Actions deployment workflow

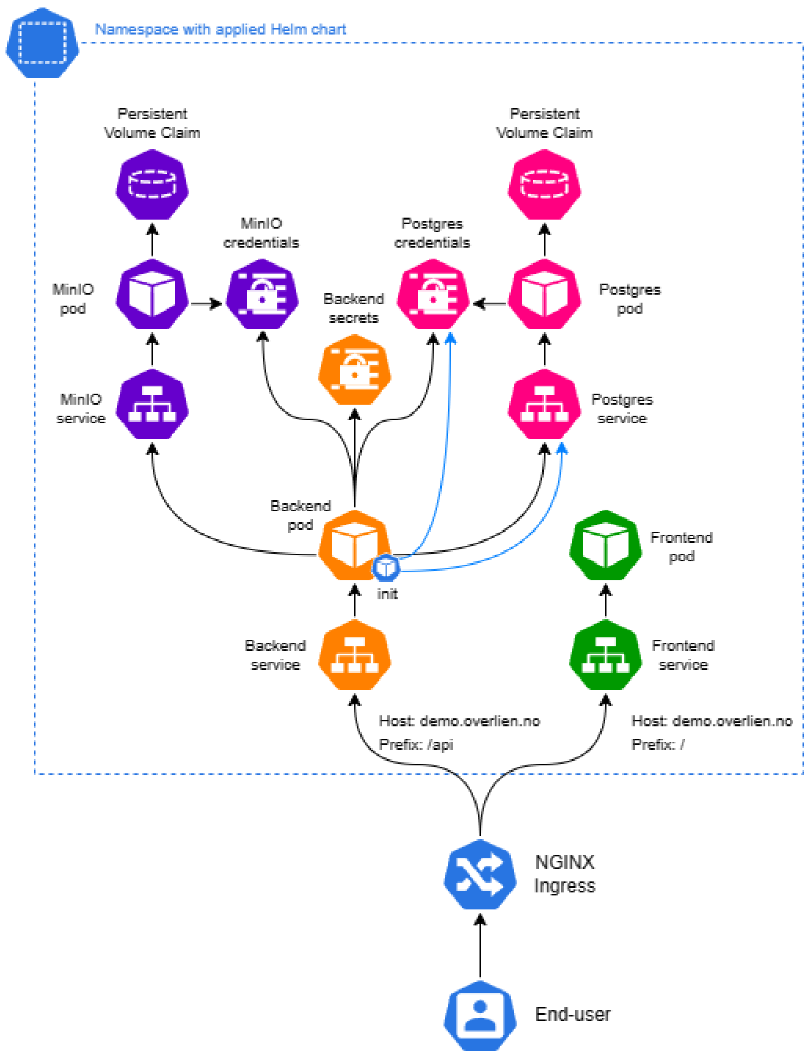


Figure 4.5.2: Diagram showing all resources defined by the Helm chart

Using the same containers, the application can be configured to run in other environments as well. The containers are set up with configurable environment variables, which allows configuring the service for alternative deployments, such as using Docker locally or other container runtimes. Which environment variables, and whether or not they are required, is noted in the project's README file.

4.6 Map rendering performance

During development of the application, performance problems were identified when attempting to add directions to guardrails in Leaflet. Initially the map implementation used Leaflet as a library, but it had no clear API for rendering guardrail directions without introducing third-party extensions. Adding the Leaflet SVG TextPath extension to allow rendering arrows along the guardrails geometry impacted performance to a unusable state for larger sample sizes. To help determine whether or not to continue using Leaflet for rendering maps, the performance was compared with OpenLayers. Table 4.6.1 shows the results from profiling the performance of the libraries using different implementations for rendering guardrails. Only the OpenLayers WebGL and Leaflet SVG TextPath implementations included rendering of guardrail directions. Using these results, while also considering other conveniences such as the built-in WKT parsing in OpenLayers ultimately ended with the decision to switch libraries.

Sample size	**1000**	**2500**	**5000**	**10.000**	**25.000**	**50.000**
Leaflet SVG TextPath	1200ms	4000ms	–	–	–	–
Leaflet SVG	55ms	120ms	235ms	550ms	1200ms	2400ms
Leaflet Canvas	25ms	60ms	100ms	215ms	500ms	1150ms
OpenLayers Canvas	25ms	38ms	68ms	140ms	385ms	950ms
OpenLayers WebGL	30ms	45ms	*missing*	135ms	275ms	600ms

Table 4.6.1: Guardrail rendering performance comparison

5. Discussion

Throughout the development process, some challenges were faced with both the technologies used and implementations. This chapter covers these challenges and the considerations made facing them.

5.1 Preliminary project plan

At the beginning of the project, a preliminary project plan was written. Since the group only had a general idea of what iSi was requesting at this point in time, not many specifics were included in it. As a result, only four generic phases of the project were defined. However, this didn't seem to impact the end result, as meetings with iSi helped garnering a better understanding of the project scope.

5.2 Planning workflow

As stated in Subsection 3.1.2, tasks were not estimated explicitly. As a result, work would sometimes be unevenly distributed. This happened even though there was a joint effort made when planning to attempt to balance this workload. This can likely be attributed to the combination of working with an unknown domain and new technologies, as well as varying levels of experience within the group.

Additionally, user stories and tasks could sometimes lack clear descriptions or criteria causing confusion in the development process. However, the group found that these details would get solved by internal discussions, and didn't heavily impact the work or cause delays.

5.3 Design choices

In the design of the application, in-car use by drivers has been an important consideration. A complex map combined with a cluttered interface could visually overwhelm the driver, potentially increasing cognitive load and causing distractions. As mentioned in Subsection 3.2.1, a simplistic style that avoids unnecessary content was important to ensure a positive user experience and reduce the risk of introducing disturbances. Drivers may also see themselves in dark environments or environments where light levels change, such as going in and out of tunnels. This resulted in adding a dark theme, allowing users to change colors to less disturbing ones. However, when integrating the application with an in-car system, adding automatic transitioning of colors to less disturbing ones should be considered.

5.4 Technology choices

Technologies chosen in this project have been picked with the scope of the project in mind. The project required learning about an unknown domain, including geospatial data which the group had little prior experience with. This impacted the choice of technologies, ultimately resulting in choosing technologies where prior experience could be taken advantage of. For the backend, this meant utilizing Spring Boot and Java. The backend uses a PostGIS database, which extends PostgreSQL with geospatial functionality, allowing building further experience using relational databases for persistent application data. For the frontend, TypeScript and Solid.js were chosen. The prior frontend experience was using the React framework, but Solid.js has similar APIs and utilizes the same JSX syntax for component definitions. Though there are some differences, the learning experience of the framework hasn't been considered a time sink.

5.5 Client package and data transfer objects

As described in Subsection 3.6.2, a separate package for API definitions was set up. This included data transfer objects, and generation of them. This helped in avoiding mismatching types and properties in these data transfer objects. It also made a clear surface for developers to read when implementing frontend logic interacting with the backend due to the lack of implementation detail of APIs. One drawback was that it required manual generation by executing a Gradle task, a chore that sometimes would be forgotten.

5.6 Module structure

During development, it has sometimes been unclear whether or not some things should be divided into separate features. The structure described in Subsection 3.6.1 established rough guidelines, but some features such as "trip" and "project" are interconnected. Deciding where to place logic would sometimes be a challenge in these scenarios. It was however decided that this could be refactored at a later stage. Additionally, boundaries of these features are not well defined as features do not explicitly define which parts of the features are to be consumed by other features.

5.7 Custom API client for NVDB

The backend is written using Spring Boot 3, which has migrated to the jakarta package [65]. Due to this migration, errors were encountered when attempting to use the NVDB API client. The original NVDB API client still uses the javax package, resulting in incompatibility with Spring Boot 3 [66]. As a result, some time was spent creating a minimal API client that covers the API surface required by the project. It did however require us to reinvent some of the logic already provided by their official API client, such as iterating through pages of results. The lack of knowledge of the APIs nullability also caused a number of errors when processing the results. Nullability is however not covered in the official API client either, meaning it would be a problem nevertheless. In attempts to cover most edge cases, full imports of guardrail data in NVDB were tested. There were however some problems identified at a later stage due to the data changing.

5.8 Comparing capture to guardrails

A core part of the project was identifying whether or not guardrails were captured. The process for identifying whether or not a car has captured a guardrail relies on correctly being able to tell the driving direction in the road system, as well as the side the guardrail is on. Some imported data were however identified to have the opposite direction. Strategies for identifying flaws, and correcting them, was a time consuming process due to the complexity of imported data.

With the current implementation there also will not be coverage metrics where guardrails deviate from the roads geometry. These scenarios will cause a lower completion grade, potentially outside the bounds of considering the guardrail complete. An alternative approach would be to only match against the road segments a car is expected to drive, but this would require being able to properly identify the direction of the road in which a car should drive and take images based off both segment and guardrail metadata, essentially ignoring the guardrails own geometry.

5.9 Map problems

One of the goals for the project was to create an efficient way in which drivers could see guardrails, allowing them to plan their driving route. It was brought up that seeing the driving direction required to capture a railing would be helpful. In order to accomplish this, multiple approaches were prototyped. However, rendering the direction of the geometries would ultimately end up harming the performance of the map rendering when utilizing Leaflet to render the map. It was decided to draft a complementing implementation using OpenLayers instead, which includes APIs for rendering geometries as vectors using WebGL. As highlighted in Section 4.6, rendering performance when including guardrail direction is significantly better when using the OpenLayers WebGL integration, especially for larger sample sizes.

There were also other problems that arose using Leaflet. For example, when making the functionality for having a vehicle move around on the map, it was noticed there was no simple way of rotating the marker. This resulted in some time being spent making a workaround approach, which ultimately did not work as intended. Another problem arose when using the built-in Leaflet popups to show guardrail details. The popups had non-configurable CSS styling, and attempting to remove said styling interfered with coordinate transformation calculations. This made the popup show up far away from the clicked location, at random. When evaluating similar functionality in OpenLayers, no such problems were identified, so the map renderer was switched.

5.10 Stateless backend considerations

The replay system and its server-sent events use stateful data. The state of a replay is only present in memory for a backend instance, this state is then lost in cases of crashes or reboots. A result of the state being lost on restart is also that the backend container, in its current state, cannot be horizontally scaled to multiple replicas without introducing potential bugs. If users are to be routed to an instance without the expected state, users may see differing results in the UI. The replay system is however only meant as a substitute for integrating with the cars system.

5.11 Ingest performance

Capture tables may see benefit from performance optimizations such as table partitioning. The table for matched railing captures is an ever growing dataset, and could potentially grow to the point where query performance is degraded. The dataset will grow linearly with capture interval and railing length as variables. Though problems have not arisen during testing, due the nature of the ingested data it is to be expected that this gradually evolves into one as the persisted data grows.

5.12 Internationalization

Though not a requirement, internationalization strategies are used. The use of a translation file allowed easy translation of string enum values from the backend, such as project status, into human readable text. The type of translation keys are inferred and can as a result give build time errors in cases where a new status is added in the backend, but has not been translated in the frontend. In addition, the use of locale-specific formatters for dates and numbers also allowed proper formatting without introducing string manipulation or custom logic. Currently, the implementation does not allow changing the locale, however only requires configuring a new locale key and adding a translation file for it.

5.13 Identity considerations

Using a identity provider can be beneficial as opposed to implementing custom authentication and authorization logic. Some time during the project has been spent implementing identity management. Identity providers such as Keycloak could have been used to reduce the time spent on identity tasks as it already provides functionality for login, password resets and more. However, the custom implementation allowed making decisions that might not have been possible otherwise. For instance, due to the lack of ability to define headers in the EventSource browser API, using the Authorization header with a JWT token does not work without introducing extensions to the API. To support authorization for these backend endpoints, cookies are used for storing the users JWT token. Cookies are automatically included in the request from the frontend, reducing the amount of authentication logic needed in the frontend application. This is also the case for protecting uploaded static assets, such as images for vehicles or users. Utilizing the built-in image element in HTML doesn't allow customizing request headers. Request parameters could be used as an alternative, but would require custom processing of those in these scenarios, requiring extra functionality for authorizing such requests.

5.14 Feedback

By presenting work in review meetings discussed in Subsection 3.1.3, the group received feedback throughout the entirety of the project. One alignment which happened early on was in the naming scheme of entities. There was a difference in the understanding of how work orders should be structured, thus some adjustments had to be made in both the diagrams and the application. Generally, the concepts were similar, but with a difference in naming. Additionally, the progress calculation was discussed. It was decided that the progress calculations should be in terms of the guardrails road segments, since it is the most accurate representation of what can be done. Other than this, it seemed like iSi were satisfied with the development, with no other significant adjustments being made.

6. Conclusion

The goal of the project was to develop a planning tool for the capture process in inSight. Development of the tool was done using iterative methods and a number of technologies that enable the end result. The result is a responsive web application that allows for both planning and executing the capture process. The application design focuses on being minimalist to reduce cognitive overload when used by drivers. The application integrates with NVDB to import guardrail data for work orders. It utilizes server-sent events to power a frontend containing an interactive map where the car's position is displayed. The guardrails and their coverage are also displayed in the same map, giving both planners and drivers an overview of the task at hand. The user interface is customizable and responsive, allowing users to change both theme and sizes of panels in the UI. Application components are containerized and can be deployed using container technologies. The application covers goals defined initially, with all user stories implemented. Though there are some flaws and potential adjustments that can be made, especially in regards to rendering guardrail directions and capture grade calculations, it seems the end result satisfies iSi's initial request.

6.1 Future work

The requirements for the project focuses on the planning aspects of the process for capturing railing data. As a result, the application lacks some functionality to make it a complete system for handling the entire data capture process. This section covers identified work that has to be done to make it a more complete solution.

6.1.1 Integrating with the cars system

The application currently utilizes a replay system for simulating the process of capturing the data. This system has to be replaced with an in-car system that can give the driver continuous feedback while also reporting progress to a central service when internet connectivity is available.

6.1.2 Guardrail matching improvements

Though the approach highlighted in Subsection 4.4.13 works in most cases, there is work that would make the process more accurate. The matching process uses the GNSS position as the reference, but this may not be accurate with the placement of the equipment on the car. In addition, the FOV of each camera and the max distance of matching needs to be aligned with the specifications of the equipment.

Additionally, the matching process may see performance degradation if the number of railings within a plan is large. The matcher currently uses a linear scan for finding potential matches, which has worked fine for the problem sizes used in testing. Utilizing a R-tree structure to index railings may improve performance when identifying potential matches. A sort-tile recursive R-tree implementation is available in JTS, and could potentially be used to increase throughput.

6.1.3 Guardrail direction fixes

The application uses arrows to show the driving direction required to capture guardrails. Though, as mentioned in Section 5.8, the guardrail metadata in NVDB doesn't appear to provide us with a good understanding of its direction in terms of the required driving direction. Guardrail data has a placement along one or more road segments, where each of these segments are a stretch of the road network. Each of these components have their own direction, making it confusing to infer the driving direction required. Since the arrows rotate with the order of points in the geometry, incorrect inference results in them facing the opposite direction to the one required to successfully capture the railing.

6.1.4 Ingest performance

As noted in Section 5.11, ingested data is ever growing in a linear fashion. There are multiple approaches that can improve the performance of ingested data, including table partitioning. The TimescaleDB extension of Postgres can be adopted in order to automate table partitioning strategies and implement real-time aggregates of coverage metrics.

6.1.5 Identity provider integration

As a measure to avoid having to handle the added complexity of managing identities, a identity provider could be integrated. Though, as noted in Section 5.13, it may not be as flexible. However, MinIO could be configured to authorize users using a OAuth2 compliant identity provider such as Keycloak. This could eliminate the need for forwarding requests through the backend as described in 4.4.2 while accomplishing the same.

6.1.6 Split layout for guardrails

It has been discovered that listing guardrails in groups similar to projects may cause too much content in the left panel. This leaves little space for viewing railings together with plans and trips. An alternative layout for projects has been prototyped, where railings and its deviations is placed in a right aligned side menu in addition to the left one. The prototyped layout can be seen in Figure 6.1.1. This could make the list more usable, and could potentially include the view discussed in Subsection 4.4.16. It could also be combined with the resizable panels, allowing users to change the size and possibly collapsing panels when not needed.

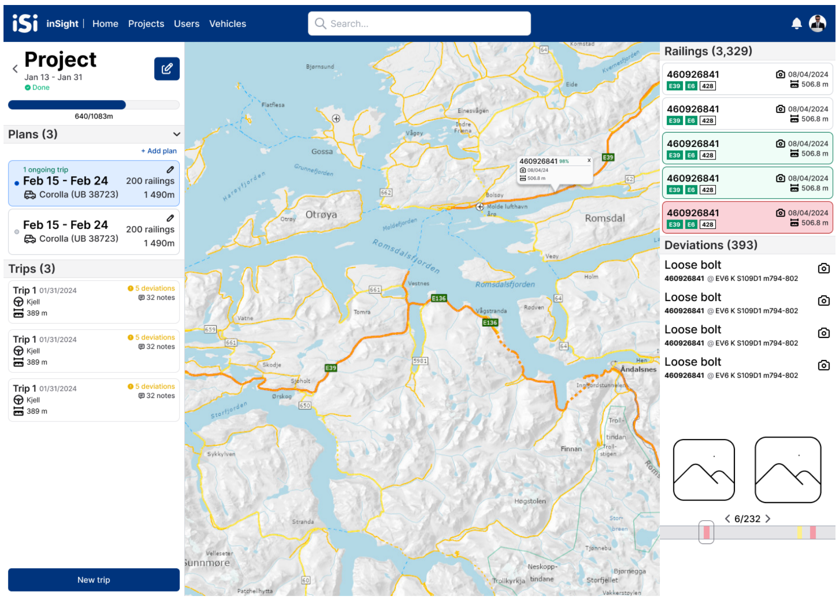


Figure 6.1.1: Prototype split layout for viewing guardrails and deviations

6.1.7 Other layout improvements

There are also other use cases which could benefit from layout improvements. For instance, gaining an overview of a user or vehicles involvement in work may not be easy in a table layout. This may be improved by including schedule calendars or timelines, where involvement is shown. This is also the case for projects, where grouping by status may not be sufficient for finding them efficiently.

6.1.8 User preferences

To enhance the accessibility of the application, a page for setting user preferences can be introduced. User preferences may for example include editing of user details, setting the language, or choosing an alternative background map.

7. Societal Impact

This chapters covers ways in which our application may impact society. The main focus here is on road safety, with some broader effects which stem from it.

7.1 The importance of guardrail maintenance

The purpose of guardrails is to minimize the damages caused by traffic accidents. Therefore, maintaining them by ensuring that no deviations are present and that they continue to adhere to the standards set by the Norwegian Public Roads Administration increases road safety. According to reports by the NPRA's Accident Analysis Group, there is a sizeable amount of traffic accidents where done damages could have been less had the guardrails been of expected quality [67]. Table 7.1.1 shows that 6 percent of fatal accidents in years 2005 - 2022 had its damage extent affected by guardrail quality. While the amount of fatal traffic accidents has been decreasing, the impact of guardrail faults has stayed about the same, with a slight upwards trend in recent years.

Year	**Amount of fatal accidents in total**	**Percentage of affected fatal accidents**
2005	202	6
2006	228	6
2007	208	3
2008	237	6
2009	186	6
2010	190	5
2011	158	3
2012	139	6
2013	170	8
2014	135	6
2015	102	3
2016	128	6
2017	102	9
2018	100	10
2019	100	11
2020	89	6
2021	76	14
2022	105	5
2005 - 2022	**2655**	**6**

Table 7.1.1: Impact of guardrail quality on fatal traffic accidents

An application providing clear overviews for routine maintenance and streamlined tools for planning this maintenance may help bring this percentage down. Proactive maintenance could help prevent major accidents, meaning less casualties, less vehicle damages, and less time and resources spent on repairs. There may even be a slight reduction of cases where roads get closed off due to accidents, causing detours and complicating transportation.

7.2 Ethical aspects

Real-time position updates of workers raises some ethical concerns. The application allows for real-time viewing of vehicle positions while captures are ongoing. However, in its current state these positions are only reported with ongoing captures. To access this information it is required that users are authenticated with the service. Additionally, the service is configured to use TLS encryption.

The system contains references to images that could potentially identify people. When performing data capture, the equipment takes images with the intention of capturing guardrails. However, some captured images may include people, allowing identification. This raises a privacy concern, but image data is handled externally to this system, moving it outside of the scope of this project. Furthermore, access to these references is provided by using the same logic for authorization and encryption as other data.

7.3 Contribution to the UN's Sustainable Development Goals

UN's goal number 3 is to ensure healthy lives and promote well-being for all at all ages. As stated in Section 7.1, efficient planning of guardrail maintenance may increase the effectiveness of guardrails. This could potentially reduce the severity of road accidents, directly relating to target 3.6 [68], which aims to halve the number of global deaths and injuries. Additionally this may also impact target 8 in goal 8 [69]. This target is to protect labour rights and promote safe and secure working environments for all workers. A significant portion of the population have roads as their main working environment [70].

Goal 12 in UN's Sustainable Development Goals aims to ensure sustainable consumption and production patterns. Target 12.2 is to by 2030, achieve sustainable management and efficient use of natural resources [71]. Through efficient planning and near real time updates on critical metrics and system status, the system may help reduce the time it takes for drivers to notice flaws in the capture. This reduces the risk of having to re-capture segments where faults occurred, ultimately reducing energy consumption by the vehicle.

References

[1] *iSi inSight Digital Rekkverkskontroll - Første Leveranser Gjennomført | iSi AS*. URL: https://isi.no/project/isiinsight_leveranser/ (visited on 05/10/2024).

[2] *Well-Known Text Representation of Geometry*. In: *Wikipedia*. Sept. 27, 2023. URL: https://en.wikipedia.org/w/index.php?title=Well-known_text_representation_of_geometry&oldid=1177433825 (visited on 04/08/2024).

[3] *Spatial Reference System*. In: *Wikipedia*. Feb. 9, 2024. URL: https://en.wikipedia.org/w/index.php?title=Spatial_reference_system&oldid=1205529676 (visited on 04/08/2024).

[4] *EPSG Geodetic Parameter Dataset*. In: *Wikipedia*. Jan. 16, 2024. URL: https://en.wikipedia.org/w/index.php?title=EPSG_Geodetic_Parameter_Dataset&oldid=1196127532 (visited on 04/08/2024).

[5] *JSON*. URL: https://www.json.org/json-en.html (visited on 03/25/2024).

[6] Patrick Charollais. "ECMA-404, 2nd Edition, December 2017". In: (2017).

[7] *YAML Ain't Markup Language (YAML™) Revision 1.2.2*. URL: https://yaml.org/spec/1.2.2/ (visited on 05/20/2024).

[8] Mark Nottingham, Erik Wilde, and Sanjay Dalal. *Problem Details for HTTP APIs*. Request for Comments RFC 9457. Internet Engineering Task Force, July 2023. 16 pp. DOI: 10.17487/RFC9457. URL: https://datatracker.ietf.org/doc/rfc9457 (visited on 03/25/2024).

[9] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. Request for Comments RFC 7519. Internet Engineering Task Force, May 2015. 30 pp. DOI: 10.17487/RFC7519. URL: https://datatracker.ietf.org/doc/rfc7519 (visited on 03/25/2024).

[10] *OpenAPI Specification v3.1.0 | Introduction, Definitions, & More*. URL: https://spec.openapis.org/oas/latest.html (visited on 05/02/2024).

[11] *About Universal Design*. Centre for Excellence in Universal Design. URL: https://universaldesign.ie/about-universal-design (visited on 05/02/2024).

[12] W3C Web Accessibility Initiative (WAI). *WCAG 2 Overview*. Web Accessibility Initiative (WAI). URL: https://www.w3.org/WAI/standards-guidelines/wcag/ (visited on 05/02/2024).

[13] *Web Content Accessibility Guidelines (WCAG) 2.2*. URL: https://www.w3.org/TR/WCAG22/ (visited on 05/02/2024).

[14] *ARIA - Accessibility | MDN*. Apr. 17, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA (visited on 05/02/2024).

[15] W3C Web Accessibility Initiative (WAI). *WAI-ARIA Overview*. Web Accessibility Initiative (WAI). URL: https://www.w3.org/WAI/standards-guidelines/aria/ (visited on 05/09/2024).

[16] *Observer*. URL: https://refactoring.guru/design-patterns/observer (visited on 05/02/2024).

[17] *HTML: HyperText Markup Language | MDN*. Mar. 5, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (visited on 03/19/2024).

[18] *CSS: Cascading Style Sheets | MDN*. Mar. 5, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (visited on 03/19/2024).

[19] *JavaScript | MDN*. Mar. 5, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 03/19/2024).

[20] *TypeScript homepage*. URL: https://www.typescriptlang.org/ (visited on 03/19/2024).

[21] *TypeScript Handbook - The Basics*. URL: https://www.typescriptlang.org/docs/handbook/2/basic-types.html (visited on 03/19/2024).

[22] *JSX Specification Draft*. Aug. 4, 2022. URL: https://github.com/facebook/jsx (visited on 03/19/2024).

[23] *React*. URL: https://react.dev/ (visited on 03/19/2024).

[24] *Virtual DOM and Internals – React*. URL: https://legacy.reactjs.org/docs/faq-internals.html (visited on 03/19/2024).

[25] *Solid Docs Home*. URL: https://docs.solidjs.com/ (visited on 03/19/2024).

[26] *JS Framework Benchmark 2024 Chrome 122*. URL: https://krausest.github.io/js-framework-benchmark/2024/table_chrome_122.0.6261.69.html (visited on 03/19/2024).

[27] *Hva er Nasjonal vegdatabank (NVDB)*. Statens vegvesen. URL: https://www.vegvesen.no/fag/teknologi/nasjonal-vegdatabank/hva-er-nasjonal-vegdatabank/ (visited on 04/08/2024).

[28] *Earth Ellipsoid*. In: *Wikipedia*. Apr. 14, 2024. URL: https://en.wikipedia.org/w/index.php?title=Earth_ellipsoid&oldid=1218967848 (visited on 05/20/2024).

[29] *Geoid*. In: *Wikipedia*. Apr. 18, 2024. URL: https://en.wikipedia.org/w/index.php?title=Geoid&oldid=1219635836 (visited on 05/03/2024).

[30] Lars Mæhlum. *NN2000*. In: *Store norske leksikon*. Jan. 25, 2023. URL: https://snl.no/NN2000 (visited on 05/03/2024).

[31] *What Is the Geoid? | Virtual Surveyor*. Support Portal. URL: https://support.virtual-surveyor.com/support/solutions/articles/1000261346-what-is-the-geoid- (visited on 05/08/2024).

[32] *Client-Server Architecture | Definition, Characteristics, & Advantages | Britannica.* Apr. 26, 2024. URL: https://www.britannica.com/technology/client-server-architecture (visited on 05/07/2024).

[33] *Server-Sent Events - Web APIs | MDN.* Mar. 6, 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (visited on 04/01/2024).

[34] *REST.* In: *Wikipedia.* May 6, 2024. URL: https://en.wikipedia.org/w/index.php?title=REST&oldid=1222499122 (visited on 05/07/2024).

[35] *What Is REST?* REST API Tutorial. Dec. 12, 2023. URL: https://restfulapi.net/ (visited on 05/07/2024).

[36] *HTTP Request Methods - HTTP | MDN.* Apr. 10, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (visited on 04/15/2024).

[37] *What is a relational database? | IBM.* URL: https://www.ibm.com/topics/relational-databases (visited on 03/19/2024).

[38] *SQL.* In: *Wikipedia.* May 11, 2024. URL: https://en.wikipedia.org/w/index.php?title=SQL&oldid=1223298674 (visited on 05/20/2024).

[39] *Data Definition Language.* In: *Wikipedia.* May 13, 2024. URL: https://en.wikipedia.org/w/index.php?title=Data_definition_language&oldid=1223657397 (visited on 05/20/2024).

[40] *Data Manipulation Language.* In: *Wikipedia.* Dec. 14, 2023. URL: https://en.wikipedia.org/w/index.php?title=Data_manipulation_language&oldid=1189909275 (visited on 05/20/2024).

[41] *Data Query Language.* In: *Wikipedia.* Dec. 2, 2023. URL: https://en.wikipedia.org/w/index.php?title=Data_query_language&oldid=1188028005 (visited on 05/20/2024).

[42] *Simple Mail Transfer Protocol.* In: *Wikipedia.* Apr. 23, 2024. URL: https://en.wikipedia.org/w/index.php?title=Simple_Mail_Transfer_Protocol&oldid=1220388312 (visited on 05/03/2024).

[43] *Git - About Version Control.* URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control (visited on 03/19/2024).

[44] *Unit Testing.* In: *Wikipedia.* Mar. 12, 2024. URL: https://en.wikipedia.org/w/index.php?title=Unit_testing&oldid=1213385244 (visited on 03/19/2024).

[45] *Integration Testing - Microsoft Solutions Playbook.* URL: https://playbook.microsoft.com/code-with-engineering/automated-testing/integration-testing/#why-integration-testing (visited on 03/19/2024).

[46] *E2E Testing - Microsoft Solutions Playbook.* URL: https://playbook.microsoft.com/code-with-engineering/automated-testing/e2e-testing/#resources (visited on 03/19/2024).

[47] IBM Cloud Team. *Containers vs. Virtual Machines (VMs): What's the Difference?* IBM Blog. Apr. 9, 2021. URL: https://www.ibm.com/blog/containers-vs-vms (visited on 05/07/2024).

[48] *About the Open Container Initiative - Open Container Initiative*. URL: https://opencontainers.org/about/overview/ (visited on 05/10/2024).

[49] *Docker: Accelerated Container Application Development*. May 10, 2022. URL: https://www.docker.com/ (visited on 05/10/2024).

[50] *Containerd/Containerd: An Open and Reliable Container Runtime*. URL: https://github.com/containerd/containerd/tree/main (visited on 05/10/2024).

[51] *Containerd vs. Docker | Docker*. Mar. 27, 2024. URL: https://www.docker.com/blog/containerd-vs-docker/ (visited on 05/10/2024).

[52] *K3s - Lightweight Kubernetes | K3s*. May 6, 2024. URL: https://docs.k3s.io/ (visited on 05/10/2024).

[53] *Overview | Kubernetes.io*. URL: https://kubernetes.io/docs/concepts/overview/ (visited on 05/10/2024).

[54] *What Is Infrastructure as Code? - IaC Explained - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/iac/ (visited on 04/15/2024).

[55] mijacobs. *What Is Infrastructure as Code (IaC)? - Azure DevOps*. Nov. 28, 2022. URL: https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code (visited on 04/15/2024).

[56] Atlassian. *What Is Continuous Integration*. Atlassian. URL: https://www.atlassian.com/continuous-delivery/continuous-integration (visited on 04/15/2024).

[57] Atlassian. *Continuous Integration vs. Delivery vs. Deployment*. Atlassian. URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment (visited on 04/15/2024).

[58] *What Is Scrum? | Scrum.Org*. URL: https://www.scrum.org/resources/what-scrum-module (visited on 04/01/2024).

[59] *Scrum Guide | Scrum Guides*. URL: https://scrumguides.org/scrum-guide.html (visited on 04/01/2024).

[60] *What Is Scrum and How to Get Started*. URL: https://www.atlassian.com/agile/scrum (visited on 05/15/2024).

[61] W3C Web Accessibility Initiative (WAI). *Avoid Too Much Content*. Web Accessibility Initiative (WAI). May 7, 2024. URL: https://www.w3.org/WAI/WCAG2/supplemental/patterns/o5p03-manageable-quantity/ (visited on 05/10/2024).

[62] *Introduction to Corvu*. corvu. URL: https://corvu.dev/docs/ (visited on 05/09/2024).

[63] *Introduction – Kobalte*. URL: https://kobalte.dev/docs/core/overview/introduction (visited on 05/09/2024).

[64] *Python-kode som les inn bin-filer*. Kartverket.no. Feb. 20, 2023. URL: `https://kartverket.no/api-og-data/separasjonsmodellar/python-kode-som-les-inn-bin-filer` (visited on 05/09/2024).

[65] *Preparing for Spring Boot 3.0*. Preparing for Spring Boot 3.0. URL: `https://spring.io/blog/2022/05/24/preparing-for-spring-boot-3-0` (visited on 04/01/2024).

[66] *Manglende Støtte for EE9 · Issue #107 · Nvdb-Vegdata/Nvdb-Api-Client*. URL: `https://github.com/nvdb-vegdata/nvdb-api-client/issues/107` (visited on 04/01/2024).

[67] *Dybdeanalyser av dødsulykker – UAG*. Statens vegvesen. URL: `https://www.vegvesen.no/fag/fokusomrader/trafikksikkerhet/ulykkesdata/analyse-av-dodsulykker-uag/` (visited on 05/15/2024).

[68] *Goal 3 | Department of Economic and Social Affairs*. URL: `https://sdgs.un.org/goals/goal3#targets_and_indicators` (visited on 05/03/2024).

[69] *Goal 8 | Department of Economic and Social Affairs*. URL: `https://sdgs.un.org/goals/goal8#targets_and_indicators` (visited on 05/03/2024).

[70] *Almost 29 Transport Workers per 1 000 People in the EU*. URL: `https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20210923-2` (visited on 05/03/2024).

[71] *Goal 12 | Department of Economic and Social Affairs*. URL: `https://sdgs.un.org/goals/goal12#targets_and_indicators` (visited on 05/03/2024).

Appendices

A - GitHub repository

All code and technical documentation referred to in this document is uploaded to a GitHub repository. Explanations on how to start or deploy the application can be found in the README-file.

GitHub repository link

`https://github.com/triosnok/isi-planning-tool`

B - Impact of guardrail quality on fatal traffic accidents

Data compiled from analysis reports of fatal traffic accidents on a national basis published by NPRA's Accident Analysis Group. The table below shows traffic accident fatality in relation to guardrails.
The reports can be found here:
`https://www.vegvesen.no/fag/fokusomrader/trafikksikkerhet/ulykkesdata/analyse-av-dodsulykker-uag/`

Year	**Amount of fatal traffic accidents in total**	**Percentage of fatal traffic accidents where extent of damage was affected by guardrail faults**
2005	202	6
2006	228	6
2007	208	3
2008	237	6
2009	186	6
2010	190	5
2011	158	3
2012	139	6
2013	170	8
2014	135	6
2015	102	3
2016	128	6
2017	102	9
2018	100	10
2019	100	11
2020	89	6
2021	76	14
2022	105	5
2005 - 2022	**2655**	**6**

C - Full size diagrams

Some of the diagrams in this thesis is available in their original format on GitHub.

Note that they may be more ideal to view in dark-mode, to view it you can import it in diagrams.net. Some shapes have white color, resulting in them potentially appearing invisible when viewing in-browser.

ER diagram

`https://github.com/triosnok/isi-planning-tool/blob/main/docs/diagrams/erdiagram.drawio.svg`

Use case diagram

`https://github.com/triosnok/isi-planning-tool/blob/main/docs/diagrams/use-cases.drawio.svg`

Architecture diagram

`https://github.com/triosnok/isi-planning-tool/blob/main/docs/diagrams/architecture.drawio.svg`

Log processing flow chart

`https://github.com/triosnok/isi-planning-tool/blob/main/docs/diagrams/log-processing.drawio.svg`

D - Map rendering performance profiling

Profiling results can be found in a zip file, prefixed by their type and suffixed by sample size. One of the results is missing for the OpenLayers WebGL is missing, but hasn't been needed to draw a conclusion in the decision.

Prefix	**Implementation**
leaflet-direction	Leaflet SVG TextPath
leaflet-svg	Leaflet SVG
leaflet-canvas	Leaflet Canvas
ol-canvas	OpenLayers Canvas
ol	OpenLayers WebGL

The zip file is available on Google Drive.
`https://drive.google.com/file/d/1-v9lw4n9CLRBVBY_LofLnyq7Mx2DwPuA/view?usp=sharing`

E - Wireframes

The Figma wireframes used to prototype the look and feel of the application can be found here: `https://www.figma.com/file/uKKi6cVPcAo7b5SVhel6wD/Planning-tool?type=design&node-id=0%3A1&mode=design&t=8VdIVfexb8UOyjEv-1`

F - Video

Some functionality may be hard to replicate without access to appropriate data. To visualize the results, a video highlighting major features is made. This video can be found on Google Drive. `https://drive.google.com/file/d/1vUZ1JPS3I1SKjbXx-b8k592imTpz0-Pl/view?usp=drive_link`