Interactive Road Network Accident Visualization Dashboard

MerlinRoads Team - Design Group 16

J.J. te Poel, N.P. Damink, R.G.A. van der Linde,
T.J. Miedendorp de Bie, E. Rudzitis, V. Bakanas

EEMCS, TCS, University of Twente
Enschede, Netherlands

Supervisor:
Mahboobeh Zangiabady
m.zangiabady@utwente.nl

April 14, 2025

UNIVERSITY OF TWENTE.

Abstract

This report is written as part of the "Design project" course of the bachelor's degree in Technical Computer Science at the University of Twente. A dashboard called "MerlinRoads" has been developed for the department of Computer Architecture For Embedded Systems. The development of this system was motivated by the following goal: *"To develop an interactive accident visualization dashboard that leverages Python, the Dash framework, and the Dash Sylvereye road network visualization library to effectively visualize and analyze car accident statistics within the road networks of major cities or metropolitan areas"*. This report will elaborate on the design process of the developed dashboard which provides insights on accident data through road networks. It describes all phases of the project work such as making requirements, elaborating on design choices, and discussing mockups. The application is designed with a strong focus on its usability for city officials and urban planners in providing enough useful information that could be used in the decision-making process for traffic management. In the future, this dashboard tool could potentially be published by the Univerisity of Twente and Centro Geo.

Contents

Chapter 1

Introduction

Traffic accidents are a major concern worldwide, leading to severe injuries, fatalities, and economic burdens on societies. According to the World Health Organization, road traffic injuries are one of the leading causes of death worldwide, where every year the lives of approximately 1.19 million people are cut short due to traffic accidents [World Health Organization, 2023]. More than 90% of these incidents occur in low and middle-income countries.

Traditional accident analysis tools rely on static reports and historical data, which lack the ability to provide dynamic real-time insights into accident trends and patterns. Currently, this limitation hinders decision-making for traffic management authorities and urban planners. With advancements in data visualization and interactive road network frameworks, it is possible to enhance accident analysis by leveraging dynamic and real-time data representation on road maps.

In response to this challenge, the "MerlinRoads" dashboard has been developed as part of the Design Project module in the Technical Computer Science Bachelor's program at the University of Twente. The primary goal of this dashboard is to provide a powerful tool for urban planners, traffic safety analysts, and city officials to understand and address road safety issues. By visualizing car accident data on a dynamic map, users of the dashboard can identify high-risk areas and make decision-driven decisions to improve traffic safety in metropolitan regions. In addition, there is a traffic simulation functionality, where it is possible to make changes on the road networks and look at the density/congestion results of this in a city.

In Chapter 2, the general analysis of the system and its domain is examined. In Chapter 3, the requirement specification methodologies are ex-

plained together with mock-ups that are provided based on the needs of the involved stakeholders. In Chapter 4, the implementation and detailed design of the dashboard are discussed, highlighting the development and design choices. In Chapter 5, the testing approaches are discussed, together with the functionalities to be tested. Finally, in Chapter 6, the design project is evaluated and the final conclusions are highlighted.

Chapter 2

General analysis

In this chapter, the general analysis and factors influencing the development of the dashboard are discussed. By analysing the broader context of traffic safety and incident visualization, this section lays the foundation for the dashboard's design and functionality. The background will be explored, followed by the primary objectives of this project. Furthermore, we will look at the purpose, scope and applicability of the dashboard. Finally, an overview of key achievements highlights the significant milestones reached during the development of "MerlinRoads".

2.1 Background

In this section, the process of identifying the domain of the dashboard is discussed. The perspective of our client and users will be viewed on from an "outside-in" perspective [Vijverberg and Opdenakker, 2013]. This strategy will help us clarify the existing problem for road data visualization. A clear understanding of the problem and the domain will enable smooth planning and development of the dashboard.

2.1.1 Introduction to the Domain

In metropolitan areas, traffic safety is a growing concern due to rising accident rates. Road accidents are a leading cause of fatalities and injuries, making their analysis and prevention an important field of study. Traditional incident analysis methods rely on static reports and historical data, which fail to provide real-time insights into accident trends and patterns.

With current advances in data visualization and real-time analytics, it is becoming possible to integrate interactive dashboards for road safety solutions. This will improve the understanding on accident hotspots, contributing factors, and potential risk zones. This project will utilize Dash Sylvereye [Garcia-Robledo and Zangiabady, 2021], which is a WebGL-powered library that is designed to dynamically visualize road networks as nodes and edges. By integrating real-time accident data through an API, the system will enable city officials and urban planners to identify high-risk locations, monitor traffic safety trends in a city and implement targeted interventions.

2.1.2 General Knowledge of the Domain

The domain of traffic incident analysis and visualization combines multiple disciplines, like geospatial data analysis and real-time data processing. Traffic accidents occur due to various factors such as road conditions, weather, and infrastructure layout. Understanding how these factors contribute to accident rates allows for data-driven decision-making in urban planning and road safety management.

Current road safety manuals rely on textual accident reports [PIARC Road Safety, 2025]. The manual provides an outline in the road safety management process, but the interpretability and inter-activeness is still a

concern for decision makers. However, with the implementation of our dashboard, it will be possible to analyze and prevent accidents more effectively.

This project aims to bridge the gap between static analysis and real-time interactive visualization by leveraging Dash Sylvereye [Garcia-Robledo and Zangiabady, 2021]. The integration of a dashboard with accident data APIs will allow for efficient monitoring, pattern recognition, and enhanced decision-making in road safety management in the future.

2.1.3 Client, Users, and Interested Parties

The dashboard is developed for various stakeholders who will interact with it or benefit from its insights. The client of this project is the University of Twente and CentroGeo. However, the system will be used by other stakeholders. These stakeholders are categorized as primary users, indirect users, and supporting institutions. Please refer to Figure 1 in Appendix A for the visualization of all stakeholders within this domain.

a) Primary Users (Direct Interactions with the Dashboard)

- *Urban planners and city officials* – These are direct users of the dashboard who will analyze accident trends and identify high-risk areas.
- *Traffic safety analysts* – They will study accident patterns and contributing factors such as time, location, and road conditions. In addition, they are able to generate reports based on the dashboard data and provide recommendations for improving traffic management.
- *(UT) Researchers* – Academics and data scientists from academic institutions could analyze accident patterns and correlations with external factors like weather, traffic volume, and road conditions.

b) Indirect Users (Benefiting from the Insights of the Dashboard)

- *Government institutions* – Public agencies responsible for enforcing road safety laws and traffic regulations inside cities. They may utilize insights provided by traffic safety analysts.

- *Educational institutions* – Universities and research centers focusing on urban mobility and safety. They can integrate the dashboard's insights into their programs and encourage further innovations in road safety.
- *Urban companies* – Businesses involved in urban mobility, including ride-sharing companies, public transport operators, and logistics firms. They can adjust their routes based on accident-prone areas and identify high-risk zones for drivers.

c) Supporting Institutions

- *Data Providers* – Organizations like HERE Technologies and TomTom supply real-time and historical data to ensure the dashboard remains up to date.

2.1.4 Software Environment

In the first meeting with the clients, they made clear that there are certain requirements concerning technologies used. The dashboard should leverage Python, the Dash framework, the Dash Sylvereye library, Weaverlet and SUMO. Python is the primary programming language for backend logic and API interaction [Python Software Foundation, 2016]. Dash is the framework for building the interactive web dashboard and UI components through Dash Mantine Components [Plotly, 2024]. Dash Sylvereye is a specialized visualization library for rendering and analyzing road networks [Garcia-Robledo and Zangiabady, 2021]. Weaverlet is a light, server-side, component-driven framework developed at CentroGeo, used for building scalable multi-page dashboard applications [ObservatorioGeo, 2024]. SUMO is a microscopic and continuous traffic simulation tool for large networks [DLR, 2024]. For data collection, we were asked to give the possibility to upload a .csv file with historical traffic data or collect it through a real-time traffic incident API, like HERE or TomTom. This strict set of technologies provided us with a detailed roadmap of what is expected from us for the development of the dashboard.

2.2 Objectives

In this section, the objectives of the design project are discussed. The objectives are structured to ensure the effective development of the "MerlinRoads" dashboard. The main objective of this design project is *to develop a dynamic and intuitive web dashboard for visualizing and analyzing road accident data.* The specific objectives include:

- **Develop an interative incident visualization dashboard** - Create a dashboard using dash and Dash Sylvereye for intuitive road network rendering.
- **Integrate real-time and historical accident data** - Provide the possibility to upload historical traffic data on the network or collect through HERE or TomTom API to provide up-to-date insights into traffic incidents.
- **Enable interactive data exploration through filters** - Provide filtering options based on factors such as time and location.
- **Simulate traffic on a road network** - Use the SUMO tool to simulate traffic on a specific road network in the dashboard.
- **Provide analytical tools and visualizations** - Provide charts, graphs, and colorings on the map to highlight accidents and hot-spots.
- **Design a user-friendly interface** - Ensure that both technical and non-technical users can navigate easily through our dashboard and understand all the features.
- **Ensure automatic data updates** - Integrate the specified API endpoints into the dashboard to ensure accurate updates.

By focusing on and achieving these main objectives during the design and development process, the project will contribute to improved traffic analysis, data-driven decision making in urban areas with complex road-networks, and urban planning strategies for accident prevention.

2.3 Purpose, Scope, and Applicability

In this section, we will look at the purpose, scope and applicability with regards to the "MerlinRoads" dashboard.

2.3.1 Purpose

The purpose of the dashboard is to provide a comprehensive and dynamic platform that enables urban planners, traffic safety analysts, and other stakeholders in the Onion Model 1 to visualize and analyze road incident data through the road network. By optionally leveraging real-time or historical accident data, the application allows for the identification of accident hotspots, emerging traffic trends, and high-risk zones. The tool aims to optimize traffic safety management and guide urban planning decisions. Ultimately, the purpose of this project is to foster a safer and more efficient road network environment through the use of data visualization and simulation technologies.

2.3.2 Scope

The scope of the dashboard is defined to establish a dynamic and interactive web dashboard that visualizes and analyzes accident data for urban traffic safety. The project will deliver a complete tool that allows users to explore accident patterns through historical or real-time data. Users can monitor traffic trends and identify high-risk zones in metropolitan areas. The dashboard will feature the following key functions:

- Visualization of road networks on an interactive map.
- Integration of real-time or historical incident data through APIs or *.csv* dataset upload.
- Ability to filter data based on various factors such as time, location, severity, incident type, etc.
- Traffic simulation using SUMO to visualize traffic patterns and assess potential accident risks.
- Analytical tools like charts for identifying accident trends over time on a road or intersection.

The project will initially focus on the global usability of our dashboard and provide insight into urban traffic safety through interactive data exploration and analysis. The traffic simulation is a side feature that we separate from the general usability of analysis.

2.3.3 Applicability

The dashboard is highly relevant to our stakeholders 2.1.3 in urban planning and traffic safety management. Urban planners use it to make informed decisions about infrastructure design and traffic safety measures. Traffic safety analysts benefit from the ability to monitor accident hotspots and observe trends over time, which helps in prioritizing helps in prioritizing interventions and improving road safety. Additionally, its broad applicability support efforts worldwide to improve traffic management.

2.4 Achievements

In this section, the achievements of the 'MerlinRoads' dashboard are shown. Upon the completion of the design project, the following key achievements were attained:

1. **Development of a dynamic, interactive dashboard** that integrates real-time and historical accident data for traffic management.
2. **Successful integration of Dash Sylvereye**, enabling the exploration of road networks and accident hot-spots on the map.
3. **Real-time and historical data integration**, allowing the dashboard to deliver up-to-data or older incident information for city officials and analysts through APIs or custom upload.
4. **Traffic simulation using SUMO**, offering insights into traffic flow and accident-prone areas on the road network.
5. **Filtering and customization**, allowing users to search for specific locations, filter on dates, types and optionally customize the settings on the dashboard.

These achievements have contributed to the overall goal of completing the predetermined objectives. With the completion of this project, the dashboard will contribute towards improving traffic safety management, enhancing urban planning, and fostering a safer and more efficient urban road network in the future.

Chapter 3

Requirements and analysis

In this chapter, the process of specifying and analyzing the requirements of the dashboard are discussed. Firstly, the process of the requirements specification is shown. This is followed by the use-case diagrams and mock-ups of the dashboard. Finally, the requirement analysis is given.

3.1 Requirements Specification

In this section, the requirement specifications are guided by Agile and Iterative Development approaches and practices. The requirements need to be properly stated. Therefore, we have applied several techniques in order to identify the requirements of the dashboard.

3.1.1 Project Management Approaches for Requirement Specification

Agile and the iterative development approaches are used to manage the project [Larman, 2004]. The requirements are specified through iterative loops, instead of being "frozen" in a major up-front specification. By refining throughout the project, we ensure that the dashboard contains all functionalities that the stakeholders need. For the project, we used the SCRUM framework, where weekly meetings with the team and the two supervisors were held to keep them updated on weekly improvements [Scrum.Org and ScrumInc, 2004]. This resulted in smaller development phases where stakeholder feedback was continuously implemented.

3.1.2 Requirements Formulation

The requirements of the dashboard are formulated according to the SMART guidelines [Mannion and Keepence, 1995]. The SMART method guides the formulation of requirements. By following these guidelines, we ensure that they are specific, measurable, acceptable, reasonable and time-bound. Due to the clear definition of the requirements, the developed functionalities can be properly tested and refined when needed.

3.1.3 Prioritization of Requirements

The SMART requirements will be prioritized based on the importance of the associated functionality. We prioritize them based on the impact for important stakeholders *(primary users)* and how it would contribute to the general purpose of improving road safety. The prioritization follows the MoSCoW method, where requirements are categorized into Must-Have, Should-Have, Could-Have and Wont-Have. MoSCow has proven to be a reliable and useful tool to prioritize requirements on your dashboard project.

3.1.4 Conclusions

The above mentioned techniques have helped us with identifying the requirements of the dashboard. The iterative Agile approach together with the weekly meetings facilitated in a structural approach of the requirement process. Refer to Appendix D for the specified requirements of the dashboard, followed by the SMART guidelines.

3.2 Use Case Diagram

In this section, the use case diagram of the functionalities on the dashboard is provided. The different use cases are provided in one large diagram that gives an overview of the whole system. Please refer to Appendix C for the use case diagram of 'MerlinRoads'.

3.3 Mock-ups

In this section, the system overview and the initial design mock-ups will be discussed. The different components of the system can be discussed based on the requirements discussed in the section above and stated in the Appendix D. These mock-ups of the dashboard serve to improve the user experience and are designed for ease of use. Keep in mind that these figures are the initial mock-ups, and a lot on the dashboard has changed through iterative improvement through feedback. In the following paragraphs, the pages and functionalities of the dashboard are identified and elaborated.

3.3.1 Dashboard Home Screen

Firstly, the user will be brought to the main dashboard page after opening the application. The main page is used for basic functionalities. On this page, the map together with the search bar and the customization components are shown. Users can go through the map and inspect the roads and intersections. When the user wants to load a specific city or street, they can search for the place with the search bar. The mock-up of the home page is visualized in Figure 3.1.

Figure 3.1: Home screen of the dashboard

3.3.2 Search Bar with Filtering

In the top left corner of the homepage, you can search for a specific place or city using the search bar. Once the search bar is clicked to look for something, the side panel bar is opened. On this side panel, users can filter for specific dates and see the accident details together with some statistics. See Figure 3.2 below to see the side panel.

Figure 3.2: Side bar after searching for a place or placing the market somewhere

3.3.3 Settings and Filter panel

On the home page, there is a settings and filter button in the top right corner. The cogwheel will open a settings panel, where it is possible to set customizations and choose different data providers (APIs).

The filter button opens a filter panel, where the user can filter on specific incidents or severity: Low, Minor, Major, Critical or All.

3.3.4 Statistics page

When there is not enough statistical information on the home page, the user can click the left corner link in Figure 3.2 to see all the accident details and metrics. On the statistics page, all the information of the street is visualized in graphs and charts, see Figure 3.3 below.

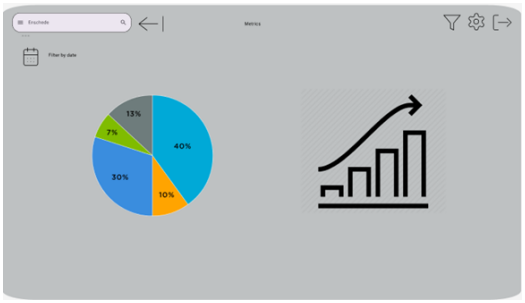

Figure 3.3: Metrics page

3.3.5 Google Street View Shortcut

Additionally, to improve the user experience, the team decided to add a Google Street View shortcut that linked to the specific location the user had searched for. With this shortcut, the user can easily get a visual image of the scene for improving the decision-making process.

3.4 Proposal

After making the mock-ups, we scheduled a proposal meeting with our supervisor and external client from CentroGeo [Garcia-Robledo and Zangiabady, 2021]. With this meeting, the intention was to receive feedback on our initial designs and iterate through them. Out of this proposal meeting, some additional requirements were formed based on the suggestions from the external supervisor.

3.4.1 Additional features

- **SUMO:** on the dashboard, there should be a traffic simulation page for the road network. For this, we will use the "Simulation of Urban Mobility", also known as SUMO [DLR, 2024]. With this additional feature, users of the dashboard will be able to simulate traffic on the road network.

- **Modularity:** Instead of making the API endpoints strict in the backend, the choice should remain in the hands of the dashboard users. Therefore, it should support multiple traffic data APIs and the customizations dashboard appearance should be customizable.

3.5 Requirement Analysis

In this section, we will build upon the analysis of the stakeholders, specified in 1. We will formulate the requirements according to the demands of the stakeholders. As the primary users of the dashboard will all have the same use cases, we will refer to them as "user". In order to fulfill these stakeholder requirements, we have identified system requirements for each of them. These are all the functionalities that the dashboard should contain.

3.5.1 Stakeholder Requirements

As we have stated, the main stakeholders of the dashboard are urban planners, city officials, traffic safety analysts, and researchers (UT).

1. As a user I want to view accidents in a certain area.
   A user can look at accidents that occurred in a certain area.

2. As a user I want to search for a specific city or street.
   A user can search for a specific city or street to see the accidents in that area.

3. As a user I want to filter the accidents based on certain characteristics.
   A user can filter accidents based on dates, the day of the week, and the time of day.

4. As a user I want to see statistics of the area that I'm looking at.
   A user can see statistics of the area they are looking at to get a quick overview of accidents in that area.

5. As a user I want to see details about a specific incident.
   A user can get more details about a specific accident by hovering over the marker on the map. This shows information like the type of accident, severity, and time it occurred.

6. As a user I want to choose the statistical graphs that I see.
   A user can pick which graphs they want to see to get the information suited for their goal.

7. As a user I want to view the traffic accident history of a specific road.
   A user can click on a road to see the accident history of that road.

8. As a user I want to change the style of the map and the markers of the accidents.

A user can customize the style of the map and change the color of a marker based on the type of accident.

9. As a user I want to export the results I found while running the application.
   A user can export the incidents currently displayed on the dashboard to upload and use the data at a later time.

10. As a user I want to authenticate securely using my credentials.
    A user can log in to the application with credentials they set up themselves.

11. As an administrator I want to be able to create a key that allows another user to create an account.
    An administrator can create a key so that a new user can create an account and set up their credentials.

12. As a user I want to be able to open Google Maps Street View of the location that I'm currently looking at.
    A user can press a button to be redirected to the Google Maps Street View of the location they are currently looking at to get a better understanding of what the location looks like.

3.5.2 System Requirements

Functional Requirements

1. The system must display markers on the map representing road incident locations

2. The system must provide a search functionality allowing users to input a location name or address of interest to navigate the map to that area

3. The system must allow users to control the map by drag and scroll gestures

4. The system must provide controls for users to filter the displayed incident types of interest

5. The system must provide controls for users to filter displayed incidents by time date range

6. The system must provide controls for users to filter displayed incidents by specific days of the week

7. The system must allow users to gain more information about individual incident marker by means of hovering over it
8. The system must allow users to select a specific road segment on the map
9. The system must display metrics (e.g. charts, statistics) associated to the selected specific road segment
10. The system must automatically check for and display new accident data received from the data source without requiring a manual page refresh
11. The system shall allow users to change the maps base tile layers
12. The system shall allow users to select which specific charts/statistics are displayed within the metrics section
13. When a specific road is selected, the system shall filter the displayed accident markers to only show those on the selected road
14. The system shall provide an option to visually represent road characteristics (e.g. coloring roads based on incident frequency or severity)
15. The system could provide functionality to export displayed incident metrics into a CSV file format
16. The system could provide a login interface for user authentication
17. The system could restrict access to the main dashboard/functionality to authenticated users
18. The system could provide an interface for designated administrators to generate unique registration keys
19. The system could allow new users to register an account using a valid registration key
20. The system could provide a button or link to open the current map view's central location in Google Maps Street View in a new tab/window

Non-Functional Requirements

1. The system must load initial accident data and markers for the default view or searched area within 5 seconds

2. The controls for filtering and map interaction should be clearly labeled and easily accessible

3. The system must accurately represent the incident data (e.g. location, severity, details, etc.) as provided by the source API

4. The systems codebase must adhere to defined coding standards and include sufficient comments and documentation to allow for easier maintenance and future development

5. The systems web interface must be compatible with the latest versions of most major web browsers (e.g. Chrome, Firefox, Safari)

6. Users performing core task of filtering incident data for the first time should be able to successfully complete it within 15 seconds without referring to help documentation

Chapter 4

Implementation and Detailed Design

In this chapter, the technicalities of the dashboard and its descriptions are provided and explained. Furthermore, the design choices of certain functionalities are identified, explained and their justification is given.

4.1 Plan of Implementation

To ensure a structured and efficient implementation of the dashboard, we followed a systematic approach that focused on maintaining consistency, readability, and quality. The following standards and tools were used in the implementation:

4.1.1 Python Coding Standards

As Python coding standards, we adhered to PEP 8 [Python Software Foundation, 2001]. This is a coding style guide for python to improve the readability and maintainability. As Python was our main (95% of the lines) coding language, we wanted to deliver consistent code that is organized and structured.

4.1.2 Merge Requests and Code Reviews

All the code changes and new features were submitted via Merge Requests (MRs). These merge requests are assigned to a reviewer before being merged into the development branch. Reviewers check the code quality and overall functionality. Comments and feedback were improved by the creator of the merge request before final approval and merging. The platform we have used for our code management is [GitLab Inc., 2025].

4.1.3 Issue Board

After all the requirements were specified in Appendix D, issues were created for each functionality to manage tasks and feature requests. This provided transparency in the team by categorizing issues into To Do, In Progress, Review, Done. When bugs or small adjustments on the development branch occurred, an Issue was made for this and solved by one of the team members through a merge request.

4.1.4 CI/CD Pipeline

Inside our GitLab repository [GitLab Inc., 2025], we implemented a CI/CD pipeline to ensure code quality and maintainability. We used `ruff` for linting and formatting to enforce adherence to PEP 8 and docstring conventions, while selectively ignoring certain rules not applicable to our project structure. Automatic unit tests were executed using Python's built-in `unittest` framework to verify functionality and prevent regressions. This setup helps

maintain consistent code standards and ensures reliability throughout the development lifecycle.

4.2 Dashboard Overview

In this section, we will go through the final dashboard and explain the functionalities. As the scope of this project was to combine and build on existing frameworks, we did not have to design any class diagrams in general. This project focuses on the functionality of the dashboard and the requirements specified in the Appendix D.

4.2.1 Dashboard Home Page

In Figure 4.1 below, you can see the home page of 'MerlinRoads'. On this page, the initial road network map of Enschede is loaded. The user can freely move itself on the map and zoom in/out where needed. On the home page are the following functionalities: searching and filtering, looking at real-time incidents on the road network, customization through the settings panel and going to the SUMO page.

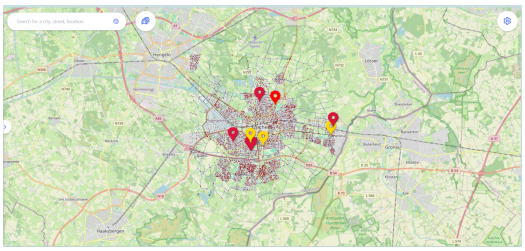

Figure 4.1: Dashboard home page

4.2.2 Search Bar and Search Side Panel

Once the search bar is clicked, the filter side panel will automatically open. When some input is typed in the search bar, there will be auto-complete and suggestions that appear under the bar. Additionally, the user of the dashboard can optionally filter the results of the incidents on date, days, and time ranges. This is especially applicable when historical incident data is uploaded on the dashboard. See Figure 4.2.

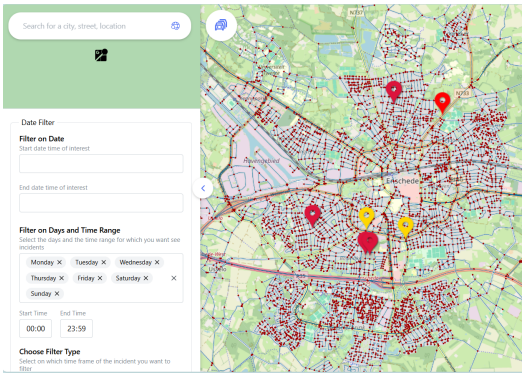

Figure 4.2: Search bar and filter side panel

4.2.3 Settings Panel

Once the settings icon in the right top corner on Figure 4.1 is clicked, the settings panel will open. On the settings panel, users have the following customization options, shown in Figure 4.3 below:

- *Default settings*: in this drop-down, the default road map range and API range are set. The road map range is the road network from Dash Sylvereye [Garcia-Robledo and Zangiabady, 2021] and the API range is for the incidents through the API or custom data.
- *API*: in this drop-down, the API endpoints are configured. We have two options for real-time incident data, which is TomTom or HERE Technologies. Alternatively, a custom .csv dataset can be uploaded.
- *Tile maps*: in this drop-down, the user can manage the map layers and tile sources. On these tile maps, the road network and incident market will be loaded. The dashboard give three default tile maps, with also the option to get a custom tile map.
- *Appearance*: in this drop-down, the user of the dashboard is completely free to customize the visual style of MerlinRoads. For the road network graph, the sizes of the intersections, roads and markers can be adjusted. Additionally, all the colors for the intersections, roads and incidents can be customized through a color picker.

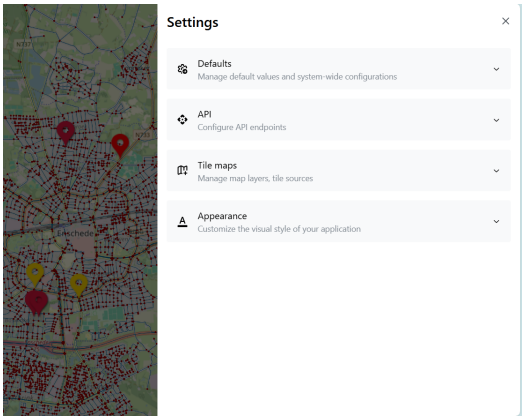

Figure 4.3: Settings panel

4.2.4 Incident Statistics Modal

Every edge on the road network graph can be clicked. When clicked, a statistics modal for potential incidents will appear. The statistics modal shows the street information, incident distribution by type in a pie chart and under it an incident timeline chart. See Figure 4.4 below.

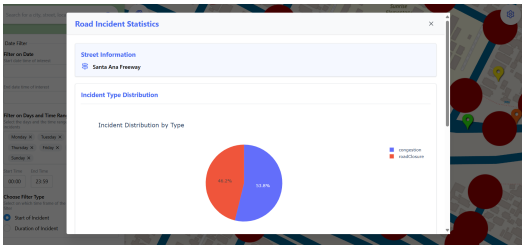

Figure 4.4: Incidents statistics modal

4.2.5 Sumo Page

When the user has done sufficient research on a specific city and would like to simulate traffic on specific roads and intersections, they can go to the SUMO page by clicking the car button next to the search bar in Figure 4.1. On the SUMO page, the incidents from the custom dataset or the API are collected and shown in the road network. The simulation settings can be changed in the settings panel and the simulation can be started. An example result of congestion measured on the roads of the network is shown in Figure 4.5 below.

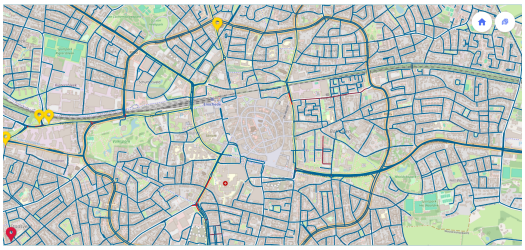

Figure 4.5: Sumo page

4.2.6 Sumo Settings

The sumo settings panel focuses on the simple usage of the traffic simulator [DLR, 2024]. On the settings panel, we have the following drop-downs, shown in Figure 4.6:

- *Simulation settings*: the user can configure the simulation parameters, where they need to set the number of vehicles and number of simulated seconds on the road network. Additionally, all incident roads can be closed for the simulation through "Add all Marked Edges". Finally, the SUMO can run.
- *Color settings*: after the simulation has run, users can adjust the color or roads based on speed, congestion, waiting time, etc. By updating the edge color, differences in congestion rates on the network will be visualized, as seen in Figure 4.5.
- *Visibility settings*: toggle to show incident markers on the road map.

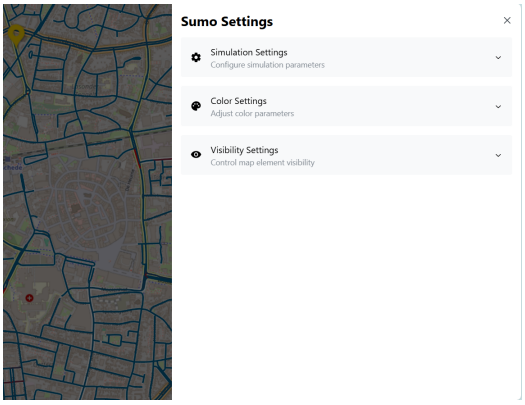

Figure 4.6: Sumo setting panel

4.2.7 Sumo Statistics Modal

Each road on the network can be clicked after the simulation has finished. A modal will appear that show the metrics of the road from the simulation. For instance, this can be used to identify congestion-sensitive roads on the network.

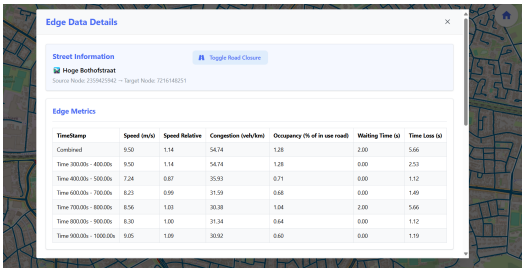

Figure 4.7: Sumo statistics modal

4.3 Design Choices

Throughout the design development process of the dashboard, several choices have been made in consultation with the whole project group and our supervisors. These choices serve a general purpose of improving the end product, where we will elaborate on their justification.

4.3.1 System Look and Feel

The first design choice concerns the system's look and feel. For this dashboard, it is very important that the usage is intuitive and resembles a map dashboard. Therefore, we have decided to stay in line with Google Maps and use simplistic lucide icons on the buttons. Additionally, we decided to make use of colors (e.g. indigo) that do not feel overwhelming.

4.3.2 Filtering on Search

As a city official or traffic analyst, you most likely want to look for certain patterns in incidents. Therefore, we believe that the filtering on date, days and time is an important part of the search mechanism. As this functionality is not really applicable for real-time data through the APIs, we still believe that it will bring value when custom incident data is uploaded.

4.3.3 Order on Settings Panel

We have discussed the order appearance of the setting dropdown during the peer reviews with the other project groups. As we first had the appearance dropdown with all the color pickers appear first, we received feedback that it could feel overwhelming. Additionally, the customization is a setting that in general would not primarily be used. Therefore, we decided to order the parts of the settings panel based on importance of usage, as shown in Figure 4.3.

4.3.4 Statistics Modal instead of a Page

In the development process of MerlinRoads, we came to the discovery that the API endpoints did not provide any historical data. As a result, the daily incidents on the road map were with a maximum of approximately two on the same road. As a result, the statistics page on itself was not applicable anymore, and could potentially cause confusion for the user. Therefore, we

decided to create a statistics modal for each edge, that displays the pie- and line-chart or, empty when an edge is clicked that contains no data.

4.3.5 Separation of Search with Sumo

During the design and development process, we decided to separate the search functionality with the traffic simulator. This is based on the assumption that the users of MerlinRoads will first search and filter for their specific locations. When they are interested in simulating specific roads or areas, they can decide to go to the SUMO page. This provides a more simplistic and clearer page to simulate traffic on the road network.

4.3.6 Edge and Node vs. Road and Intersection

With our stakeholders kept in mind, we decided to remove the theoretical term of Edges and Nodes from our dashboard. This can potentially cause confusion for city officials and traffic analysts, as they may have not heard these terms before. Therefore, we use "Roads" and "Intersections" throughout the whole application to improve the user experience.

4.3.7 Session Management

In our Dash application, we use Flask sessions for managing user-specific settings across multiple interactions and page loads. Dash is built on top of Flask, allowing seamless integration with Flask's session management system.

Flask sessions store data on the client side using signed cookies. This means that most of the user configuration, preferences, and temporary state are saved in the browser itself, ensuring a lightweight and scalable server-side architecture. Despite residing on the client side, Flask cryptographically signs the data to prevent tampering, ensuring integrity and security.

We manage the session settings and behavior in the ***settings.py*** file. This includes the session key, timeout configurations, and any default values that need to be initialized. Through this configuration, we ensure consistency in user experience, preserve state across callbacks, and provide a secure method to store and retrieve per-user data.

This client-side session approach is particularly suitable for applications with moderate session data requirements and enhances performance by reducing the need for a centralized session store.

Chapter 5

Testing The Dashboard

In this chapter, the testing approach and test results are provided for the dashboard. The different functionalities which are tested are indicated.

Lastly, the test results are provided of all the tests. The design has been changed or bugs have been found in the code. They are solved based on the results of all these tests together.

5.1 Testing Approach

The dashboard project will focus on providing unit tests that will cover most of the code paths in the project source. External dependencies of the frameworks and libraries will not be extensively tested. In order to test 'MerlinRoads', we apply three testing strategies. These are:

1. Unit testing
2. Integration testing
3. Usability testing

While the first two strategies focus on testing the code and the 'back-end' of the system, the usability tests focuses more on the end users. In the following subsections, these three testing strategies are discussed.

5.1.1 Unit Testing

Unit testing for the MerlinRoads dashboard focuses on testing individual components of the system to ensure correctness, reliability, and edge case coverage. The Python unittest framework is used extensively to verify isolated functionality in the codebase.

In total, 128 unit tests have been written. These cover all major modules, including:

- SUMO simulation module
- API interaction handlers (TomTom, HERE, and CSV uploads)
- Settings management
- Utility functions (e.g., date filtering, incident parsing, formatting)
- ZIndex Manager
- ID Manager

The goal of the unit testing effort is to maximize code coverage, including:

1. Validating function outputs for a wide range of inputs
2. Ensuring correct function call sequences and side effects
3. Checking data consistency and fallback behavior under edge cases

All unit tests are included in the CI/CD pipeline and must pass before a merge request is approved, ensuring consistent code quality.

Results

The tests successfully helped identify and resolve several bugs, such as:

- Incorrect incident severity parsing from certain CSV formats
- ZIndex inconsistencies when rendering overlapping map elements
- State leakage in utility functions causing unexpected side effects

By thoroughly testing the core functionality in isolation, we reduced the likelihood of downstream integration issues and improved overall system reliability.

5.1.2 Integration Testing

Integration testing targets the correct interaction between modules that function independently under unit testing. Even when individual units are correct, the integration of these components may cause unforeseen issues. This is especially true in complex flows involving data updates, visualization rendering, and backend/frontend synchronization.

Our approach treats the dashboard as a black-box system, interacting with it via its inputs (e.g., APIs, settings, file uploads) and inspecting outputs (e.g., updated map visuals, statistics modals). Key integration scenarios include:

- Switching between live APIs and user-uploaded datasets
- Simultaneous interaction with map filtering, appearance settings, and the statistics modal
- SUMO simulation execution after setting incident-related parameters

Results

The main issue identified through integration testing was:

- **Visualization bugs** occurring when switching between APIs and then uploading historical datasets. These led to marker overlap and incorrect coloring, which were resolved by resetting the internal state between source changes.

Integration testing ensured that the dashboard works cohesively across all interconnected features, especially under common user workflows.

5.1.3 Usability Testing

For the usability testing, we mainly focus on the following heuristics for the user interface of the dashboard:

- **Visibility of dashboard status:** the users should always be informed by the system about its current status and get feedback through notifications.
- **Consistency and standards in the interface:** it should be clear to the user that certain terms and words mean exactly the same thing on different pages/panels.
- **Aesthetic and minimalist design:** the dashboard should not contain information which is irrelevant for the use cases and may be redundant. Every extra snippet of functionalities or information competes with the relevant features and diminishes their relative visibility.

The user interface design can only be accepted when it meets the expectations above. The masters of the feedback and acceptance are the users/supervisors of the system. Therefore, the dashboard will only be accepted when it satisfies the demands of these users.

The interface design are tested through 'supervisor based testing'. The supervisor and creator of Dash Sylvereye will be asked to perform a series of tasks and navigate through the functionalities of the dashboard. During the cognitive walkthrough, the supervisor was asked to say what came to mind and receive feedback from her. The interface passes the test when our supervisor can intuitively navigate through the dashboard and use all the features.

Results

The usability tests have provided a considerable amount of feedback which have served to enhance the system design for the end user. Our supervisor was asked to fulfill a variety of user tasks listed below:

1. Move through the map and zoom in/out on certain places.
2. Search for a specific location through the search bar.
3. Customize the settings of the dashboard in the panel.
4. Upload a custom dataset with historical incident data.

5. Use the search filter with dates to look for time-specific incidents on the uploaded dataset.

6. Go to the SUMO page on a specific city.

7. Set the simulation settings and run the traffic simulator on the road network.

As a result of these test scenarios, we received feedback on the usability of the system. For example, indications on loading states would give proper status updates to the user. Additionally, more notifications that help the user navigate through the functionalities were applicable. Usability issues that are solved by means of the test scenarios are the following:

1. It was not intuitive to manually open the filter panel when pressing the search bar.
   Solution: *The filter side panel now automatically opens when the search bar is pressed.*

2. When switching between APIs, there was no feedback to the user when it was updating the data.
   Solution: *Set a loading state and provide a notification pop-up after successfully updating.*

3. When the simulation is started, it is unclear how long it takes before the simulation is finished and visible on the map.
   Solution: *Set a loading state on the "Run simulation" button and close the settings panel when finished.*

The usability tests serve to enhance the heuristics and improve the user experience on the dashboard.

5.1.4 Schedule

An iterative agile approach has been used throughout the development of the system. Unit tests were made each time as a new feature was added to the dashboard. Additionally, some existing tests have been changed according to the requirement changes. The unit tests had to pass before the merge request could be approved, referring to the pipeline in 4.1.4. Integration tests were done each time a new feature was added as well, though this process is more informal when it comes to testing. Finally, usability testing was used towards the final three weeks of the project, receiving feedback to optimize user experience. Since this strategy is used to tweak the interface, it is not imperative to do while implementing a feature.

Chapter 6

Conclusions

In this final chapter, the general conclusions of this design project are discussed. We will reflect on the past 10 weeks, looking at the planning, responsibilities, team work, requirements evaluation and limitations of the dashboard. Furthermore, the development of the application comes to an end during this quartile, however the usage and future planning will then start. Therefore, a planning for the future scope of this project is of utmost importance.

6.1 Planning

To ensure a smooth execution during the 10 weeks of our project, we decided to keep a structured plan where we separate the global components of a Design Project *(Proposal, Testing, Backend, Documenting)*. In the first weeks, we focused on the design and proposal. This transitioned into the development and testing phase. The final 2 weeks mainly focused on documenting and reporting to ensure that we deliver an optional and complete end product. For the planning visualized in a Gantt Chart, see Appendix B.

Every week, a meeting with the client and supervisor was scheduled to discuss the progress of the dashboard and to request feedback about certain requirements.

At the later phases of the design project, the deliverables needed to be finalized and handed-in. The deadlines of these deliverables were set in advance together with our supervisor, ensuring a strict and consistent planning. The overall planning was respected throughout the 10 weeks which ensured an equally distributed workload and high quality of the final documents.

6.2 Responsibilities

The team was composed of students with different interests and skill-sets, meaning that the project was well divided into different tasks and responsibilities. Responsibilities were assigned based on interests and strengths. Moreover, due to a history of working well together on past projects, the responsibilities were quite straightforward from the start. The assigned responsibilities of the team members are listed below:

- **Ruben:** Lead Designer and Documentation Specialist
- **Ernests:** Front-End Developer and Report Contributer
- **Tiko:** Product Manager and Communications Lead
- **Jacco:** Chief SUMO Wrangler, Back-End Developer and Testing Coordinator
- **Niek:** Back-End Developer and Technical Writer
- **Vytautas:** Scrum Master, Requirement Analyst and Back-End Developer

Note that most of the responsibilities were a group effort, meaning that every single team member contributed to it.

6.3 Team Evaluation

A consistent work attitude led to a disciplined development process. From the early stages of planning to the final implementation and report writing, our team maintained a strong sense of cooperation and commitment. While individual roles were assigned based on preference and skills, everybody was prepared to step in and assist each other beyond their designated tasks.

Ultimately, the success of the "MerlinRoads" project was driven by a strong group dynamic, where the project not only was productive but also enjoyable.

6.4 System Specifications

For the development and testing of the dashboard, most of the project members utilized the LENOVO 21AAS27Y00, an x64-based PC system. The processor used was the 11th Gen Intel(R) Core(TM) i7-11800H, running at 2.30GHz with 8 physical cores and 16 logical processors.

It is important to acknowledge this hardware specification, as the relatively high processing power may not reflect the performance capabilities of an average end-user's device. As a result, the dashboard's responsiveness and overall performance could vary when deployed on systems with lower computational resources. It is important to take this into account for future optimizations and deployment considerations.

6.5 Requirements Evaluation

This section evaluates the final dashboard against the functional and non-functional requirements specified in Section 3.5.2. The requirements are grouped for clarity, and their implementation status is indicated (not implemented, partially implemented, fully implemented).

6.5.1 Functional requirements

- **Visualization and interaction of the map (FR 1, FR 2, FR 3, FR 8):**
  - Requirements: Displaying incident markers on a map, providing search functionality, allowing map control (drag/scroll), enabling selection of specific road segments
  - Status: Fully implemented
- **Incident filtering (FR 4, FR 5, FR 6):**
  - Requirements: Filtering displayed incidents by type (implicit via the data source), time/date range, and specific days of the week.
  - Status: Fully implemented
- **Incident information and metrics (FR 7, FR 8, FR 9):**
  - Requirements: Gaining more information about individual incidents (by hovering incident markers), displaying metrics associated with a selected road segment.
  - Status: Fully implemented
- **Map appearance (FR 11, FR 14):**
  - Requirements: Allowing users to change map base tile layers to predefined and custom, providing options to visually represent road characteristics (e.g., coloring based on frequency/severity).
  - Status: Fully implemented
- **Displayed metrics selection (FR 12):**
  - Requirements: Allowing users to choose between and select specific charts and statistics to be displayed within the metrics section.
  - Status: Not implemented
  - Comments: The incidents statistics modal displays incident distribution by type as a pie chart and incidents per day as a timeline, which were thought to be the most useful and informing statistics, thus introducing selection would be redundant. A similar justification is applied with regard to the sumo statistics modal.

- **Advanced Filtering and Display (FR 13):**
  - Requirements: Filtering displayed accident markers to only show those on the selected road.
  - Status: Not implemented
  - Comments: In the main dashboard and sumo page, selecting a road leads to the statistics modal appearing that displays informative metrics related to that selected segment. Differentiating between the users intent of filtering accident markers and displaying the statistics would require introduction of additional workarounds, which in turn would instigate additional (unneeded) usage complexity for the end-user.
- **Data Management and Export (FR 10, FR 15):**
  - Requirements: Automatically checking for and displaying new accident data without manual refresh, providing functionality to export displayed metrics to CSV.
  - Status: Not implemented
  - Comments: While the source APIs update frequently (approx. every 1-2 minutes), automatic real-time refresh was deemed not as critical for the primary use case of traffic incident analysis. Analysts typically focus on patterns over time or specific historical datasets, rather than requiring instantaneous updates. Users needing the most current up to date information can achieve this through a manual page refresh or by re-initiating a search. Moreover, this design choice conserves API usage (reducing potential operational costs). CSV export functionality was de-prioritized in favor of implementing other core functionalities.
- **User Authentication and Access Control (FR 16, FR 17, FR 18, FR 19):**
  - Requirements: Providing a login interface, restricting access, allowing admin key generation, enabling user registration with a key.
  - Status: Not implemented
  - Comments: The entire user authentication system was a 'could' have requirement, and ultimately was scoped out during development due to time constraints and importance of core functionalities.

- **External Integration (FR 20):**
  - Requirements: Providing a button/link to open the current map view's location in Google Maps Street View.
  - Status: Fully implemented

6.5.2 Non-functional requirements

- **Performance (NFR 1):**
  - Requirements: Load initial data/markers within 5 seconds.
  - Status: Fully implemented
- **Usability & Accessibility (NFR 2):**
  - Requirements: Intuitive layout, clear labeling using tooltips and instinctive icons, adhering to general usability principles.
  - Status: Fully implemented
- **Data accuracy (NFR 3):**
  - Requirements: The dashboard accurately reflects the location, severity and details provided by the connected APIs or user uploaded data.
  - Status: Fully implemented
- **Maintainability (NFR 4):**
  - Requirements: Compatibility with latest versions of major browsers.
  - Status: Fully implemented (with limitations)
  - Comments: The dashboard was primarily developed and tested on recent version of Chrome and Firefox. While expected to work on Safari, comprehensive testing across all browsers and versions was not performed.
- **Usability Tasks (NFR 6):**
  - Requirements: Completing filtering task within 15 seconds the first time.
  - Status: Satisfied

6.6 Limitations of the System

6.6.1 Data source dependencies and granularity

The accuracy of the incident views are entirely dependent on the quality, update frequency, and coverage provided by the selected external APIs (HERE/TomTom). The system itself can not correct or control inaccuracies in the source data. Furthermore, historical analysis relies solely on end-user uploaded datasets, because of lack of historical data provided by the aforementioned APIs. Additionally, supported upload data is confined to a specific format and depends on the users providing this correctly formatted data.

6.6.2 SUMO simulation scope

The SUMO framework integration provides a simulation based on user-defined settings, and not on a real-time traffic. Therefore its accuracy and usefulness is highly dependent on road network data quality and ultimately the parameters (network and simulation configuration) chosen by the end-user.

6.6.3 Performance on extremely large datasets

While the system was design with performance in mind (NFR 1), the systems behavior under 'extreme' load caused by a large custom dataset upload and visualization (for instance, visualizing many years of nationwide incident data) has not been explicitly tested and could potentially lead to slower response times.

6.7 Future Scope of the Project

Because of the limited time that we had for this project, we had to prioritize the functionalities that we wanted to implement. Therefore, there were some things we didn't have time to implement, but these can be implemented in the future.

6.7.1 Using City Boundaries Instead of Fixed Ranges

Currently, our dashboard loads roads and API markers within a rectangular area centered on the coordinates of the searched place or street. Although this approach works, it would be more effective to load data based on the actual city boundaries. For example, when searching for "Enschede," the system could automatically load all relevant data within the official boundaries of Enschede. This can be achieved using an API such as [Geoapify, 2025], which provides the geographic boundary coordinates (latitude and longitude) of cities.

6.7.2 Choose which metrics you want to see

As one of our initial requirements, we stated that we wanted the user to be able to select with metrics they wanted to see. At the moment, we have only implemented two graphs. So in the future we could make more graphs and let the user chose which ones they want to see.

6.7.3 Export Data

Another requirement we defined, but did not implement, is the option to export the real-time data that is currently displayed on the dashboard. That way, the user can import this data back into the application when (s)he wants to continue with that data.

6.7.4 Advanced Filtering

A nice addition to the current filtering could be that the dashboard only shows markers on the streets that you have selected.

6.7.5 Automatic Updates

In our current implementation the real-time data provided by the API's is only updated when you search for a location. It would be a neat addition

to the application if the data updates automatically when you have searched for a location.

6.7.6 Authentication

Authentication and access control were not implemented in this project, as they were deemed out of scope by our supervisor. However, these features are crucial for any real-world deployment. If the application were to be launched, a secure login system with access restrictions and key registration managed by an administrator would need to be developed.

Acknowledgment

We thank the EEMCS department at the University of Twente for their guidance and support. Special thanks to our instructors for providing the resources to successfully complete this assignment.

Glossary

- Dash: A python framework developed by Plotly for building interactive web-application based dashboards
- Dash Mantine Components: A user interface library that provides common components for Dash applications
- Dash Sylvereye: A specialized library built for Dash for visualization and analysis of large road networks
- Edge: In the context of road network visualization, an edge represents a road segment connecting two intersections
- Endpoint: A specific URL or address which can be accessed to perform a particular function or retrieve specific data
- FR: Functional requirement. Specifies what the system should do
- Linting: The process of using an automated tool to analyze source code for potential errors, style inconsistencies and coding standards violations
- NFR: Non-functional requirement. Specifies quality attributes
- Node: In the context of road network visualization, a node represents an intersection on the road network
- PEP 8: Python Enhancement Proposal 8. Documents that propose coding style guides for Python
- Session management: The process of maintaining user state information across multiple (lengthy) interactions with a web application
- SUMO: An open source traffic simulation package designed to handle large road networks

- Weaverlet: A server-side, component-driven framework developed at CentroGeo for building scalable multi-page dashboard applications, used as part of the underlying structure for MerlinRoads.

Bibliography

[Alexander, 2004] Alexander, I. F. (2004). A better t-characterising the stakeholders. In *CASE Workshops*, pages 215–223.

[DLR, 2024] DLR (2024). Sumo documentation. Accessed: 2024-03-26.

[Garcia-Robledo and Zangiabady, 2021] Garcia-Robledo, A. and Zangiabady, M. (2021). Dash sylvereye: a webgl-powered library for dashboard-driven visualization of large street networks. *arXiv.org*.

[Geoapify, 2025] Geoapify (2025). Geoapify boundaries API.

[GitLab Inc., 2025] GitLab Inc. (2025). GitLab: The DevSecOps Platform. Accessed: 2025-04-02.

[Larman, 2004] Larman, C. (2004). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, Boston.

[Mannion and Keepence, 1995] Mannion, M. and Keepence, B. (1995). Smart requirements. *ACM SIGSOFT Software Engineering Notes*, 20(2):42–47.

[ObservatorioGeo, 2024] ObservatorioGeo (2024). Weaverlet. Accessed: 2024-03-26.

[PIARC Road Safety, 2025] PIARC Road Safety (2025). Introduction to road safety. Accessed: March 16, 2025.

[Plotly, 2024] Plotly (2024). Dash documentation. Accessed: 2024-03-26.

[Python Software Foundation, 2001] Python Software Foundation (2001). PEP 8 – Style Guide for Python Code. Accessed: 2025-04-02.

[Python Software Foundation, 2016] Python Software Foundation (2016). Python software foundation. Online; accessed 2024-03-26.

[Scrum.Org and ScrumInc, 2004] Scrum.Org and ScrumInc (2004). The scrum guide. `https://www.scrum.org/resources/scrum-guide`. [Online; accessed DATE].

[Vijverberg and Opdenakker, 2013] Vijverberg, A. M. M. and Opdenakker, R. J. G. (2013). *Vitale organisatiestrategie: een ontdekkingstocht naar waarde, positie en identiteit*. Kluwer.

[World Health Organization, 2023] World Health Organization (2023). Road traffic injuries.

Appendices

A Onion Model

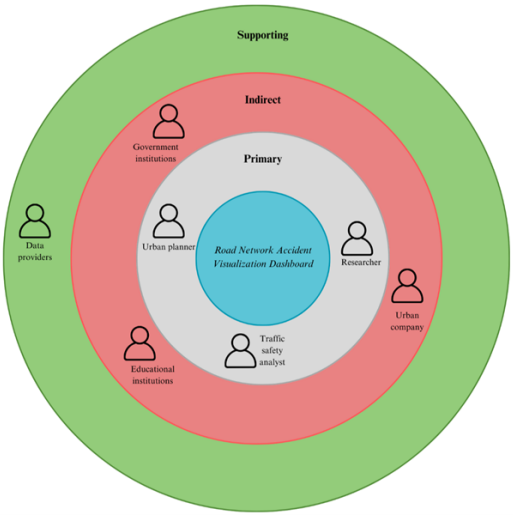

Figure 1: Stakeholder onion model [Alexander, 2004]

B Gantt Chart

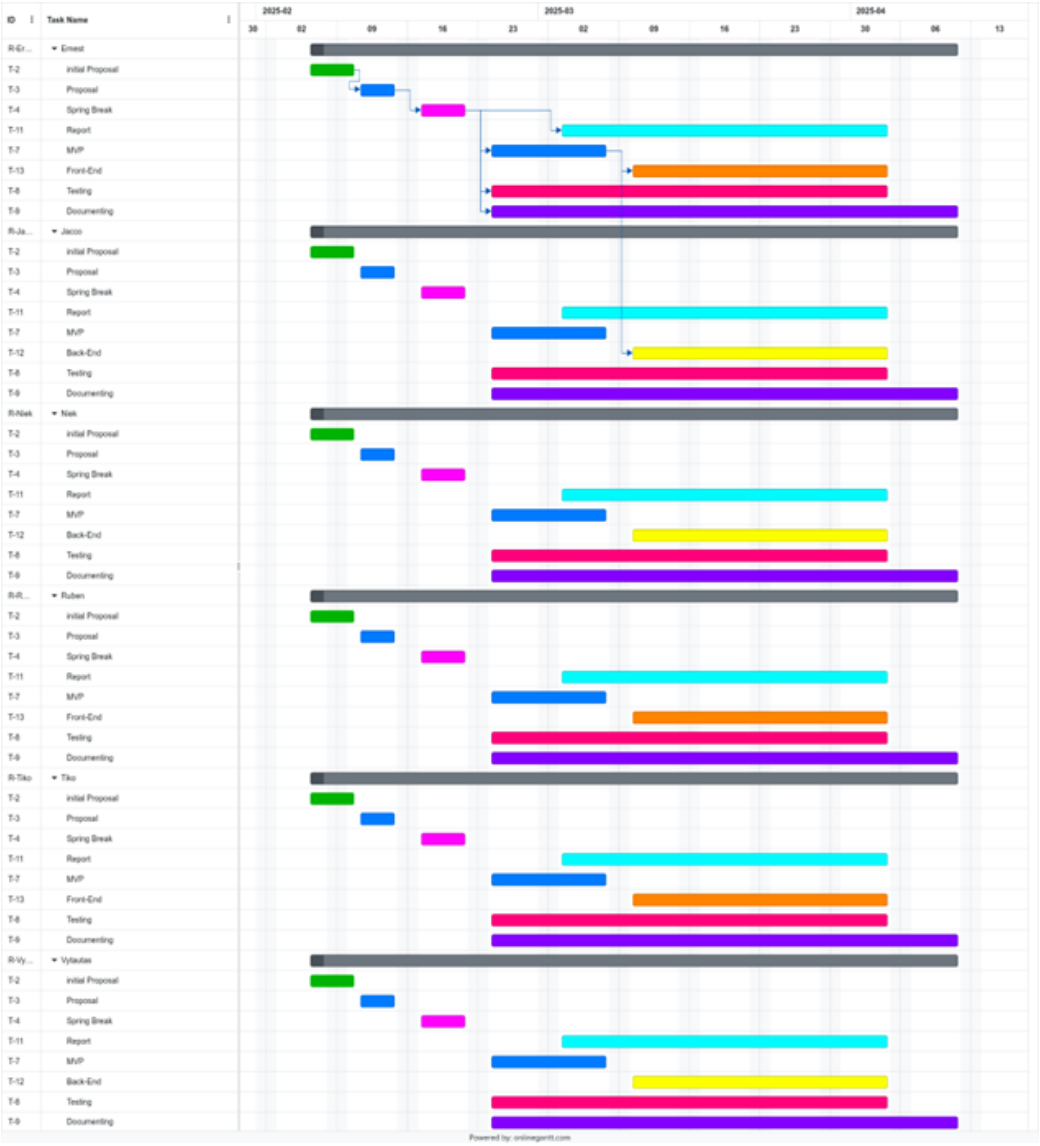

Figure 2: Gantt Chart

C Use Case Diagram

Figure 3: Use Case Diagram

D Requirement Specification

A. Must-have
1. As a user, I want to view road accidents in a specifically chosen area.
1.1. The system must have a map which shows road accidents.
1.2. The system must have a search option to view a specific location on the map.
2. As a user, I want to be able to interact with the map.
2.1. The system's map must be zoom-able by scrolling.
2.2. The system's map must be navigable by dragging.
3. As a user, I want to filter the accidents on the map based on different characteristics.
3.1. The system must support filtering accidents on their severity (Low, Minor, Major, Critical, All).
3.2. The system must support filtering accidents by specific dates.
3.3. The system must support filtering accidents by time of day (e.g. Morning, Afternoon, Evening, Night, All).
4. As a user, I want to be able to see different types of metrics for accidents in a specific area to support data-driven decision-making.
4.1. The system should have a section visualizing accident trends, comparisons, and statistics for a chosen area.
4.2. The system's graphs and charts are interactive (e.g. zoomable, filterable, real-time updates, etc).
4.3. The system's charts and graphs must update dynamically based on user-applied filters (e.g. timeframe, location, severity).
5. As a user, I want to see road accidents in real-time.
5.1. The system must automatically update the page with new data from the API when new accident data is present.
6. As a user, I want to be able to get more information about specific accidents which occurred.
6.1. The system must give information about each specific accident (e.g. type, conditions, severity, description, etc.).

Table 1: Must-have Features

B. Should-have
1. As a user, I want to be able to customize the metric view.
1.1. The system should allow the user to hide/show the metrics tab.
1.2. The system should allow the user to cherry-pick metrics of interest.
2. As a user, I want to be able to focus on a specific road to view all traffic accident history of this road.
2.1. The system's map should allow users to focus on specific roads, only displaying accidents that occurred on this specific road.
2.2. The system should only show statistics related to the specific road when the user is focused on this specific road.
3. As a user, I want to be able to customize the style of the map, so that I can adjust its appearance to my preferences or specific use cases.
3.1. The system should have an option to display the map in a different view layer (e.g. standard view, traffic flow view, satellite view, etc.).
3.2. The system should have an option to color roads (edges) with respect to certain characteristics (e.g. number of accidents, etc.).

Table 2: Should-have Features

C. Could-have
1. As a user, I want to be able to export the results I found.
1.1. The system can export accident metrics in a CSV format.
2. As a user, I want to authenticate securely using my credentials.
2.1. The system should have a login form.
2.2. The system should restrict access to the dashboard for users who have not logged in.
3. As an admin, I want to be able to create a key that allows another user to create an account.
3.1. The system should allow an admin to generate a unique registration key.
3.2. The system should allow the user to create an account with a valid key.
4. As a user, I want to be able to open a Google Maps Street View of the current location, so that I can get a real-world visual context of the area.
4.1. The system should have a button that redirects users to a Google Maps Street View of the current location.

Table 3: Could-have Features