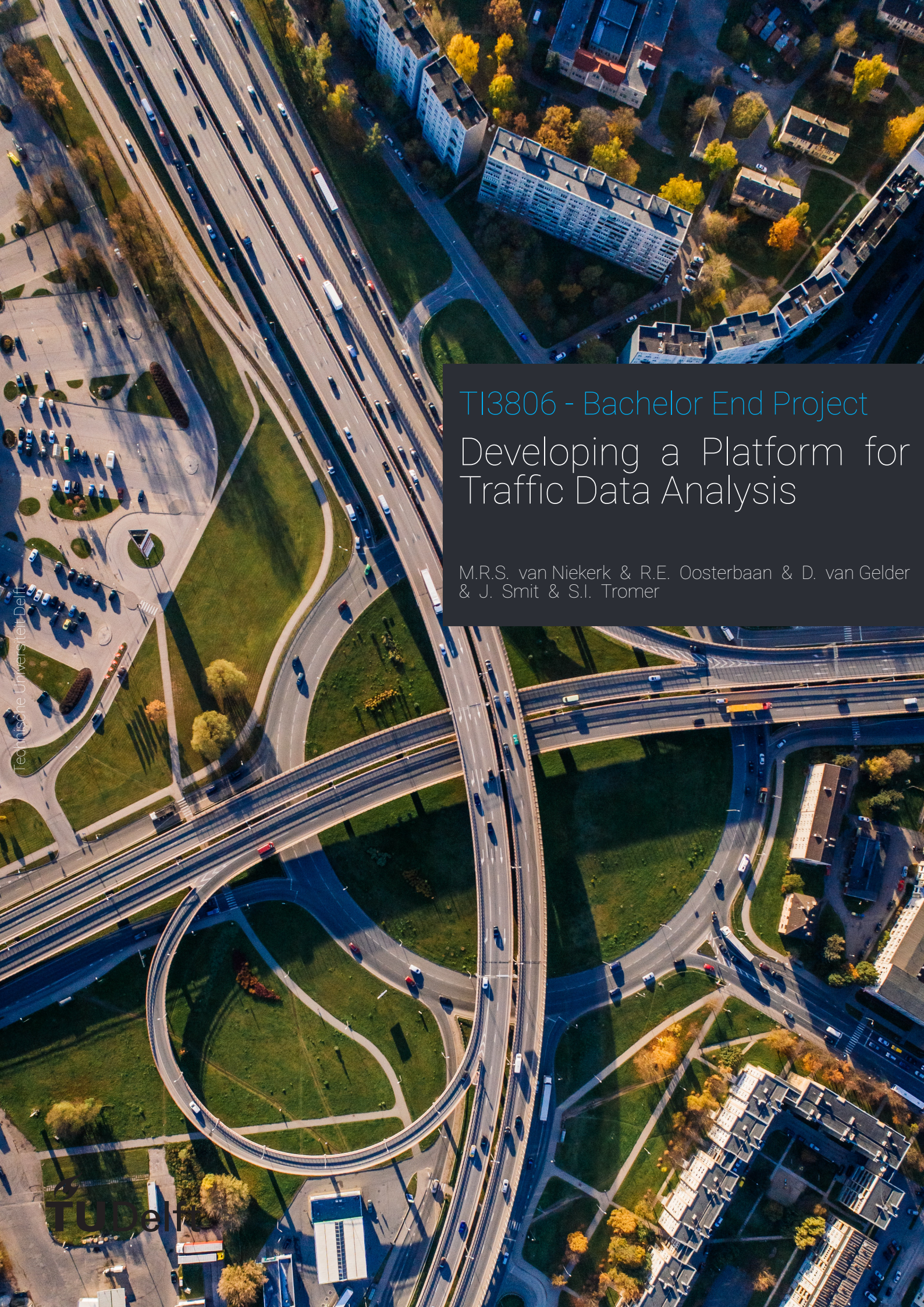


TI3806 - Bachelor End Project

Developing a Platform for Traffic Data Analysis

M.R.S. van Niekerk & R.E. Oosterbaan & D. van Gelder & J. Smit & S.I. Tromer

Technische Universiteit Delft

TU Delft

Summary

Scenwise is a business working on innovative and sophisticated solutions in the domain of traffic management. Leveraging data science and IT systems, Scenwise delivers products to institutions to facilitate efficient traffic management. In order to manage the highly complex network of infrastructure on the road network, traffic managers need to use and analyze data that is collected all across the network in order to support decision makers in management of this network. However, there is often a mismatch in expertise between traffic management experts and decision makers. Traffic management experts use highly technical visualization techniques that require significant background knowledge in the traffic management domain. In addition, the visualization techniques are spread out over a multitude of systems that do not work together. In order to bridge the knowledge gap, a product needs to be created that allows experts to extract and visualize relevant data using their traffic domain knowledge while providing intuitive and clear visualizations which are clear to both experts and non-experts. The ultimate goal of this product would be to facilitate efficient traffic management in order to improve the lives of commuters by contributing to a better organized infrastructure.

Our project group has designed and implemented a product for Scenwise that offers this solution. A web-based application has been created that retrieves and stores traffic data. The product is able to traverse the road network and provide helpful insights into the traffic network's state at either the present moment, or moments in history. The application is able to provide dynamic traffic contour plots, draw fundamental diagrams, show live traffic intensity over the entire Dutch road network and provide information related to traffic events like accidents and matrix sign states. The product is able to do all of this while providing a seamless and intuitive user interface. The system has been designed and implemented over a span of ten weeks by a group of five students. A SCRUM methodology was adopted and through careful discussion with the client and a continuous feedback loop a product was delivered that fits both the clients needs and the wider product vision that has been defined.

TI3806 - Bachelor End Project

Developing a Platform for Traffic Data Analysis

by

M.R.S. van Niekerk & R.E. Oosterbaan & D. van Gelder & J. Smit & S.I. Tromer

in partial fulfillment to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Thursday June 27th, 2019 at 14:00 PM.

Project Group:	M.R.S. van Niekerk	
	R.E. Oosterbaan	
	D. van Gelder	
	J. Smit	
	S.I. Tromer	
Assessment committee:	Ir. K. F. Chan (kinfai.chan@scenwise.com),	Scenwise BV, project client
	Ir. O. W. Visser (o.w.visser@tudelft.nl),	TU Delft, project coordinator
	Dr. A. Katsifodimos (a.katsifodimos@tudelft.nl),	TU Delft, project coach

The unredacted version of this report is confidential and cannot be made public.

An electronic version of this report is available at `http://repository.tudelft.nl/`.

Image credit cover page: Aleksejs Bermanis, published by Pexels, 2019

TU Delft

Preface

The Dutch road network is a system of connections between cities with an immense complexity. Managing traffic flow over this network requires expert knowledge in the domain of traffic management and a thorough understanding of the state of this network. Unique in its kind, the Dutch road network contains many measuring sites which constantly produce a vast amount of data. Leveraging this data stream in order to help experts manage traffic flow over the road network has been the topic of our Bachelor End Project. For ten weeks, under supervision of Scenwise, our group has worked on designing and implementing a product which provides data visualizations targeted at experts for decision support.

This report will outline the process, design and implementation details of the project. It can be used as a guideline for evaluation of our project, or as a report on the details of the structure and implementation of the final product.

We would like to thank our TU Delft coach Asterios Katsifodimos for his guidance, input and feedback throughout the entire project. We would also like to thank our product owner Kin Fai Chan from Scenwise for providing us with the enthusiastic feedback, interesting input and detailed feedback on our prototypes throughout the process.

M.R.S. van Niekerk & R.E. Oosterbaan & D. van Gelder & J. Smit & S.I. Tromer
Delft, June 2019

Contents

List of Figures

1 Introduction

1.1. Problem Analysis

Decisions in traffic management are made based on analysis of traffic data. Due to the lack of appropriate tools and the complexity of the tools that exist, it is difficult for people who are not experts in traffic management to interpret traffic data. While traffic data experts are able to judge traffic situations based on the current modelling tools, there is a knowledge gap with traffic managers/policy makers. This introduces a challenge for decision makers to understand traffic situations. Although open data exists, no clear tool exists which makes the data interpretable. A tool needs to be developed that can provide accurate traffic data between any two points on the Dutch road system in a clear and simple manner.

1.1.1. Product Vision

The goal was to develop a web-application which shows a map of the Dutch road system. The user can then pick a starting point and an end point. With these two points a route between the two points is calculated and a 'contour plot' of traffic between the two points will be shown. This 'contour plot' shows how traffic evolves over time along a route and dynamic vectors will illustrate possible congestion. The main focus of the user interface would be user friendliness. The possibility of traffic data prediction could be explored, where the application can predict future traffic conditions (average speed, traffic density, etc). The application should combine different types of visualizations of traffic data in order to support traffic managers in their analysis of the road network's state. It is crucial that decision makers within the traffic management domain can rely on their tools for accurate and clear data. Our product provides these decision makers with intuitive visualizations of this data. This could improve decision making and contribute to the efficiency of the road network for commuters.

1.1.2. Target Customers

The product will be targeting decision makers for traffic management who are not experts in traffic data analysis.

1.2. Novel and Innovative Aspects

While there are multiple projects that serve the need for traffic management analysis, there are no projects (that we know of) that combine all the features that are present in our application. A typical workflow that our application allows for goes like this: First it is possible to see in real time where congestion is occurring by looking at the real time traffic flow. Next it is possible to examine the cause by looking at incidents and construction works along the route. When the user wants to examine the congestion more closely he can look at the contour plot to see the origin, development and magnitude of the congestion. Along the road the matrix information signs are also shown so the user can examine the automated response by the traffic management system. Finally, when looking for patterns in historical data, the user can use the point cloud to examine the road congestion performance during a longer time frame.

Besides this, the way the contour plot is implemented is also unique. As far as we know, there is no other traffic management application where the contour plot is visualized using vectors that represent the cars

moving through the plot. This addition makes the contour plot much more intuitive for non-expert users.

1.3. Requirement Analysis

Requirements of the product were analyzed based on the MoSCoW method, where requirements are divided up into four categories: Must Haves (absolute necessary requirements), Should Haves (preferred requirements), Could Haves (possible extensions), Won't Haves (possible future work, not part of this project).

Must Haves

- The application shows an overview of the Dutch road network
- The user can select two points to indicate a route
- A (dynamic) contour plot of any route can be generated
- The application updates traffic data in real time
- Matrix sign information along a route can be retrieved for a given route
- Queries should not take more than a few seconds (at most 5)
- Traffic data is stored over time and a history of several hours is kept
- Application should work smoothly (no jitter) despite complex calculations on large amounts of data
- Contour plot must account for distances between measurement points on the road
- The history of road maintenance must be displayed on the map

Should Haves

- The road network already shows current traffic intensity
- Travel time on a route can be calculated based on the contour plot
- Matrix sign information data history is kept (several hours)
- Effects of matrix sign information can be measured
- The history of road maintenance must be displayed on the contour plot
- Contour plot of a route can be visualized on different points in time, based on historical data

Could Haves

- Visualization of road capacity in a point cloud can be executed based on traffic data
- The user can select more than two points for a route
- Effects of traffic flow management decisions on traffic flow can be simulated
- Researching the possibility of prediction traffic speed using a LSTM neural network

Won't Haves

- An optimal alternative route to a given route can be generated based on traffic data
- The application takes active traffic management decisions: like managing matrix sign information.

1.4. Project Planning

Table 1.1 shows the original project planning as was defined at the beginning A.5.3. Route Finding of the project. Based upon the previously formulated requirements, an estimation of workload was made and planned over the entire timeline.

Throughout the project it turned out that our final presentation would be one week earlier than originally planned. This meant that our project deadline shifted one week earlier requiring us to prioritize some features over others and discard some features. Eventually, it was decided that the research into traffic speed prediction would be limited and no implementation would be performed.

Despite a more restricted timeline, the project followed the planning appropriately.

Week	Date	Tasks	Notes
4.1	22/04 - 28/04	Orientation on technologies, product definition, analyzing possibilities for extension	
4.2	29/04 - 05/05	Setup of project backend and front end, defining API structure, writing research report	Deliverable: Research report
4.3	06/05 - 12/05	Project backend/frontend	
4.4	13/05 - 19/05	Construct RESTful API connecting layers	
4.5	20/05 - 26/05	Preparation Midterm, incorporate MSI data and road maintenance data	
4.6	27/05 - 02/06	Application will keep history of data, and history can be shown. Midterm presentation for client and supervisor, SIG code quality	
4.7	03/06 - 09/06	Research point cloud for traffic data and incorporate in application	SIG Code Quality feedback
4.8	10/06 - 16/06	Point clouds can be used to infer possible congestion later in the day	
4.9	17/06 - 23/06	Research LSTM neural network predictor for traffic speed and possibly implement it, prepare report	Second deadline SIG Code Quality
4.10	24/06 - 30/06	Finish report + info sheet and prepare final presentation	Report Deadline: 25 June (Tuesday)
4.11	01/07 - 07/07	Finish final presentation	Final Presentation: any day of this week

Table 1.1: Original Project Planning

1.5. Methodology

The project used a SCRUM based approach as follows: after every meeting with the client, requirements are set up with an estimation of its workload. From that requirement list, requirements would be selected to be worked on for the coming week. At the end of the week, the group's progress was discussed with the client and new requirements are formulated. This approach ensures flexibility of the group towards the client such that the project requirements can change throughout the project based on the group's progress. Each team member is responsible for his own assigned requirements which will be in the form of Github issues. Throughout any week, issues can be created by any group member. At the moment where requirements and issues are assigned, these new issues will be considered as well. These are typically bugs that arose throughout the week or possible new features of the project. The group is required to meet at least three times per week, once on Monday, once on Wednesday and once on Friday together with the client. Each group member is free to choose where he wants to work throughout the rest of the week as long as his responsible issues are completed at the end of the week. If this is not the case, measures can be taken by the other group members to ensure that this will be the case in the future. Should any conflicts arise within the group which can't be

resolved within the group, the group will contact the TU Delft coach in order to settle any disputes. The group intents to set up a meeting with the TU Delft coach once every two weeks to discuss the group's progress and any problems/challenges that the group encountered. However, if it is not deemed necessary to meet the meetings can be canceled to prevent the coach from attending useless meetings.

1.6. Report Structure

A thorough system design is formulated in the second chapter. This contains strategies for data stream processing/analysis and a system architecture. Component design is illustrated and inter-process communication is defined.

In the third chapter, a product description is provided containing actual implementation details of the final product. Each component's implementation is extensively described and features within the application are presented as well as the general application execution flow. Finally, implementation challenges are described that highlight the complexity of some parts of the product's implementation.

The fourth chapter discuses how we tested our application and dealt with challenges in that respect. In addition, code quality is discussed and the measures that were taken in order to ensure that the code in our application was up to industry quality standards.

The project's process is outlined in the fifth chapter, where the methodology of the project are discussed in more detail. Some challenges relating to group work are considered as well.

In the sixth chapter, our final product is evaluated according to the requirements that were formulated at the beginning of the project and according to the agreed upon methodology. In addition, the ethical implications of the product are considered and discussed.

The conclusion of our project is given in the seventh chapter, where the results of this project are summarized and final remarks are made.

Finally, the report will be concluded with some recommendations for future contributors to the product. These will include possible extensions of the project and interesting challenges that a contributor could tackle in the future.

2

System Design

2.1. Overview of Components

Based on the requirement list, we have delivered a product that provides the following functionalities to the user:

- User can select a route on a map of the road network;
- For a given route, a contour plot over a selected time range can be generated
- A point cloud (also referred to as a fundamental diagram) can be constructed over one or more days for a measurement site;
- The road network displays general traffic intensity over all roads;
- Matrix sign information is rendered along a selected route;
- Road incidents (accidents, construction works) are shown on the map with more detailed info available on the sidebar;

A detailed description of each of these components is given in the next chapter. This chapter discusses how these components connect and the design of the system as a whole.

2.2. Architecture

We define three important components in our architecture: the frontend which provides the UI for the user, the backend which parses and processes data and the database which stores data.

2.2.1. Frontend

As the frontend of the application is mainly concerned with User Interaction, design is mainly concerned with aesthetics and usability. We decided to perform all heavy computations on the backend and let the frontend be mainly concerned with rendering the received data properly in a user-friendly manner. Even though computationally load is moved to the backend, the frontend still is a computationally heavy component due the rendering of the roadmap and the contour plot. Therefore it is required that the frontend renders data efficiently in order to satisfy the fluency requirement of the UI.

The frontend is not based on any framework as the application has a relatively small UI. For added type safety we use TypeScript [37] instead of native JavaScript. We also use Sass [6] instead of native CSS. In order to render the contour plot, WebGL is used. For aesthetic design the Bootstrap [5] framework is used, as it has a familiar design for many users and is a ready-made framework with components which can be used 'straight out of the box'.

A map is a vital component of our application. There are several options for different (web) mapping services. The maps that were considered are Google Maps [13] and [redacted]. The choice came down to using proprietary or open source software. For both Google Maps and [redacted] a small demo was created to investigate the pros and cons. Developing using Google Maps proved to be cumbersome, since we

could only make a single request per day to their API. This would have limited development significantly and this basically forced us in using [redacted]. [redacted] only provides map data but no API. Fortunately [redacted] provides several similar API methods to the Google Maps APIs that work with OpenStreetmap. In addition, [redacted] provides mechanisms which allow the user to plot information on top of the map in a relatively simple and lightweight manner.

The frontend uses webpack [40] to bundle all resources to a single file. This makes the application easier to develop and decreases loading times of browsers.

2.2.2. Backend

The backend has three main responsibilities: parse the incoming data, store the relevant parsed data in a convenient format in the database and serve the stored data to the frontend. Besides these main tasks, the backend is also involved in calculating the route based on two or more locations that are sent from the frontend. We decided to use Java and the Spring framework [32] to implement backend server for three important reasons. Firstly, the original project description stated that there was already some legacy code written in Java. So writing the backend server in Java might make it possible to reuse some legacy code. Secondly, everyone in the group is already very familiar with the Java programming language. This enabled everyone to work on the backend without too much difficulty. Thirdly, we chose to use the Spring framework because this takes care of most the standard HTTP server and Database code. Making prototyping much faster since it requires much less code.

2.2.3. Database

The data persistence layer has several requirements. Some data needs to be stored once and read many times. The point measurement configuration files, for instance, are updated only once per day but are read many times. Since the point measurement configuration data only makes sense within the context of parsing data, this data is not stored in the database. The measurement data needs to be queried for a certain time interval and for a set of positions. For this to be efficient, multiple indices need to be kept. The point measurement data is updated every minute and at least a week of data needs to be stored. Initially, no columns were normalized and 8MB of data was added every minute. In a week this adds up to 80GB. While this is manageable, it would be better if the database was smaller. Using normalization we managed to improve this significantly.

Since by far the biggest part of the database is just a single table of point measurements over time, the complexity of data that the database must be able to handle is not very high. We also do not require very high write capacity, as we only need to be able to handle a few MB per minute. We do require retrieving data based on multiple requirements at once (over a time interval and over several points in space). However, almost any SQL or NoSQL database could handle these requirements. We have chosen to store our data in PostgreSQL [27], but since we make use of an Object-Relational Mapping (ORM), we could change the database with any other without much difficulty.

2.2.4. Database Structure

This section explains the details of the tables in the database.

Measurements

Since measurement data makes up the overwhelming majority of the database, it is important that this table is as small as possible. This is achieved by only storing the site id, the time, the speed and the flow. Because of limitations in our knowledge of spring, the primary key is a unique UUID. This adds 128 bits to every row, but also does not require a composite primary key that is a combination of the timestamp and site id. Every site adds a measurement, this means that about 15000 records are added every minute.

All the data about the measurements that is static is stored in the measurement sites table. Fields that are stored are the location, road number, average speed and average flow. The averages where generated from a sample of a couple of days of measurements.

Status Messages

Status Messages are lightly normalized in the database. Both the situation ID and the situation record ID are stored for every status message in the database. Other fields that are stored are the type (Accident, Construction, etc.), specific type (Road hardening, accident involving heavy lorry, etc.), start time and end time. Since most status messages stay active for a long time, every time the status messages are retrieved there is a check

if the status message already exists. This check is easy because the only thing that needs to be checked is whether the end time is set or not. This cuts down on the size of the status message tables.

Every status message can have one or more coordinates associated with it. This relation is stored as a one to many relation (where we assume that there is always at least one relation) with the `status_coords` table. The foreign key for this and all other status message tables is the situation record ID, which is guaranteed to be unique by the data source.

The lane management type has three specific fields that are also stored in the main table: number of lanes restricted, number of lanes operational and number of lanes original. Since a large portion of the status messages are of the lane management type and theses fields all take one short, there was little benefit to normalizing these fields. All other status message types have these fields set to -1.

The maintenance and construction types are the most complex types that need to be stored in the database. Therefore they have two additional tables, one for maintenance and construction specific fields and one for public comments. Every maintenance and construction message has a one to one relation to the `status_construction` table, where the expected delay, the "owner" of the construction and the expected start and end time are stored. Besides this, every maintenance and construction message can have zero or more status messages. This is a one to many relation with the `status_construction_text` table where all comments and links to relevant documents are stored.

Matrix Signs

The `msi` table in the database contains all the information about the matrix signs in the Netherlands. This includes where the matrix sign is positioned (road, carriageway, lane and km) and what is currently displayed on the matrix sign (speedlimit, lane open, lane closed, flashing, etc.). The exact location however is not in this table. The matrix signs retrieved from NDW do not contain the exact location. This means that the latitude, longitude and bearing are unknown when retrieving the matrix sign information. Fortunately NDW also provides a file that has the exact location of all the matrix signs. A parser was created to get all the matrix signs from this file and save them in a separate table called `msi_locations` that contains just the matrix sign UUID along with the latitude, longitude and bearing. Joining these two tables on the UUID enables us to get a complete picture of all the matrix signs and makes sure that the matrix signs can be plotted at the correct location on the map.

2.2.5. Communication Between Frontend and Backend

Communication between the frontend and backend components of the application will be done through a RESTful API, this ensures as much separation as possible between these components. Through HTTP requests information will be retrieved from the backend in order to be displayed on the frontend. In order to keep the computational load of the frontend low, most computation will be performed on the backend such that the frontend is only concerned with rendering the retrieved data properly.

For example: when a contour plot for a given route needs to be displayed, the frontend sends a GET request to the backend and gives the coordinates of the start- and end-point of the route as parameters. The backend will then be concerned with calculating which data points need to be retrieved and constructs the values for this contour plot. It will then return only the data required to plot the contour plot which the frontend will render in turn.

We considered using websockets to dynamically update the application using event-driven responses instead of polling the server at intervals. However, since we know that the datastream is updated at a set interval, using a polling request at an interval has a similar effect. If more intense back and forth communication would be required in the future, websockets could provide a solution.

2.3. Data Processing

To get a minimal version of the required application working, the frontend needs to be able to show the following things: A contour plot on a given route over time with matrix sign information, a map of the road network with traffic intensity, an overview of incidents and road maintenance over the network (status messages) and a point cloud for a selected measurement site. This section will consider the data sources that are used for the contour plot/traffic intensity, matrix sign information and status messages. Besides the functionality required for the minimal viable product, there are more data sources that we would like to integrate for the final product. These data sources include a history of road maintenance and an overview of matrix signs displays. This section will also touch on the data sources required for this extra functionality.

Figure 2.1: Matrix sign portal along the A12 road, retrieved from Rijkswaterstaat image database, 2019

2.3.1. Data Sources Required For the Contour Plot and Point Cloud

Speed data for every major road is determined by point measurements (a "lus"). Most "lus" measurement points measure the average speed of cars and the amount of cars over some interval, which in practice is always a minute. This measurement data is provided as an XML file every minute by NDW (Nationale Databank Wegverkeersgegevens). Besides data for the "lus" measurement point, data is also collected by bluetooth, laser, radar, license plate detection and multiple other data sources. Because of the varied data sources a separate configuration XML file is provided which is updated once per day, at 13:05 local (Amsterdam) time.

The configuration file specifies for each measurement what the source of the measurement is, how the data is aggregated and where the data is collected. The configuration file also contains information about the equipment type and the location of the measurement site. All "lus" measurement points collect data per lane of the road and some separate data on the vehicle class, where each vehicle class is in a certain size range. Since not all measurement points separate on vehicle class, we only collect aggregate data. The data for all lanes is also aggregated. Besides the speed data, the throughput ("flow") through a "lus" is also stored. We expect that this data may also be visualized in a contour plot in a way similar to the speed data.

The contour plot only needs to be able to show the average speed of cars at a certain point on the road over time. However, both the throughput and speed are required to render the point cloud for a measurement site.

2.3.2. Data Sources Required For Matrix Sign Information

The state of each matrix sign is saved and processed by the NDW as well. A separate XML file related to Matrix Sign Information data (MSI) is generated each minute, containing data of each matrix sign and properties reflecting its state. Matrix sign locations are located in the same file. In practice, matrix signs are grouped along the road on portals, shown in figure 2.1. A matrix sign portal is a group of matrix signs hanging over the road, where typically a sign hangs over each lane. For this reason, the lane number that a sign is residing over is also given in the data. However, matrix signs are not grouped per portal in the data stream so some data processing is needed in order to group them.

2.3.3. Data Sources Required For Status Messages

The status messages file provided by the NDW contains many different types of events occurring on the road network. From all these events we are only interesting in a select number of them. The two main categories are incidents (vehicle and other obstructions, road accidents, disturbances) and construction works (construction and maintenance). The data in this XML file is updated every minute. While the XML file itself is quite big (35MB uncompressed), the data in it only changes occasionally. The file only contains events that are currently ongoing. There is also another XML file that is about 1GB uncompressed that contains all ongoing and planned events. This file could be interesting for a possible extension of the application, but since there is no way to look into the future in our application, the usefulness of including it would be limited.

Due to the varied nature of the events, the structure of each event can vary wildly. There are some elements that all status messages have in common: A situation ID, a situation record ID, a location and a start and end time. A situation is a collection of situation records, while each situation record is the actual event. Some situation records can logically be grouped together, like one record that contains a construction event

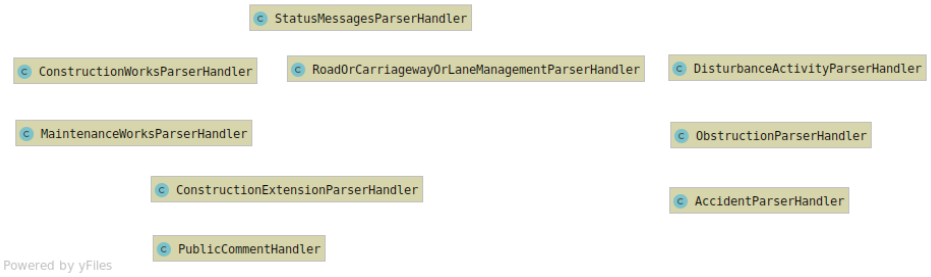


Figure 2.2: Structure of Status Messages Parser

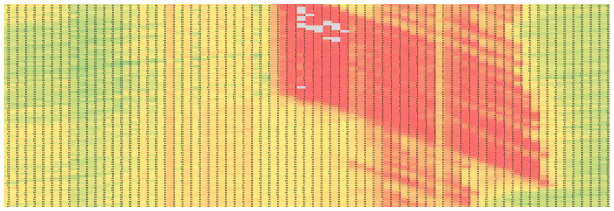


Figure 2.3: Old design time velocity plot

and another that specifies an alternate route that the traffic can take for the duration of this construction. Almost all of the time, only one situation record in a situation is of interest. Because of this, when the situation records are shown in the frontend, the id of the event is the situation id.

Because each event type has a different structure, each event type has its own parser (2.2). The `ConstructionWorks` and `MaintenanceWorks` events are similar, they share the public comment fields. In those comment fields, arbitrary textual information can be shared. Since this information is often important, it is stored and shown in the front end. Other fields that the construction and maintenance works share are average delay, start time, end time, and the government body that is ordering the construction (Zuid-Holland, Rotterdam, Rijkswaterstaat, etc.). The `RoadManagement` event is used in combination with the `ConstructionWorks` and `MaintenanceWorks` events, to detail which roads or lanes are impacted by the construction.

Disturbances, obstructions and accidents are events that are also stored as status messages. Because of their time-sensitive nature, they usually have very little information associated with them. The only specific fields that they have are concerned with the type of the incident. Accidents for example can be accidents involving lorries, heavy lorries or normal cars.

2.4. Data Analysis

Our system design also requires us to define how we want to utilize the data we retrieve. This section discusses the different types of data analysis and visualization techniques that have been researched.

2.4.1. Contour Plot

Normally traffic is visualized using a static figure of average car velocity opposed to time passed at each measurement point. These figures are similar to heatmaps. These graphs are sub optimal for understanding the flow of traffic as shown in figure 2.3. Traffic management experts find it hard to explain the meaning of these graphs to non-experts. The client needed a way to dynamically visualize traffic whilst maintaining understandability. The proposal is to use an animated vector representation as commonly used in wind visualizations [3]. The graph will represent the measurement points over a specified road segment and visualize the flow of traffic on the segment using particles. The particles will slow down when traffic is slow, but still move upwards in time. Following the particles now allows insight in traffic movement at every point in time. Our design created these vectors based on the values that the measurement sites provide as shown in figure 2.4. However, this design would pose as a too disparate visualization for current traffic management experts. Our client proposed a hybrid approach: where the vectors would function as an overlay on top of the original contour plot. An example of this is given in figure 2.5.

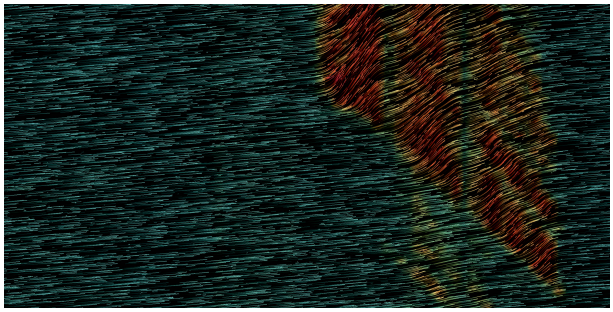


Figure 2.4: New design animated time velocity plot

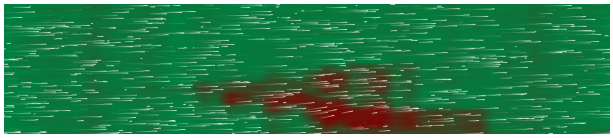


Figure 2.5: Final contour plot design with speed vectors

2.4.2. Status Messages

Other influences for traffic flow are road limitations. In order to understand traffic behaviour, it is important to know whether a congestion is caused by an overflow of traffic, or an external factor. These factors can be categorized under temporary road limitations and must also be provided to the user when analyzing the traffic. As mentioned before, the data provided by NDW contains information of active incidents and planned road works, which will be made visible to the user when necessary. Our product displays status messages on the map with varying small icons, depending on their category. When the user clicks these icons, more information is provided in a sidebar menu of the product such that the user can retrieve more details about the event taking place. This can be seen in figure 3.10. The user is also able to filter status messages on their category and sort the construction messages on their start or predicted end time.

2.4.3. Matrix Sign Information

The response of traffic velocity to road signals, in general tends towards the desired velocity. Although not as influential as active incidents, this data is still necessary to provide to the user. An important note is that there is a distinction between automatic and manual signalling. The automatic signals are designed to slow down traffic in order to guarantee safety during a congestion. The manual signals tend to be a response to active incidents or road works. The data provided does not make the distinction between automatic and manual control. However, the data is still desired when analyzing traffic and shall be incorporated in the visualizations. The design plots each matrix sign portal next to the road along the selected route, an example of this is shown in figure 2.6. However, this design would not scale when the user zooms out to a lower zoom level, this issue is discussed in the next chapter.

2.4.4. Point Cloud

Another important metric for traffic analysis is the velocity compared to the density of cars. In general a road will maintain the average velocity up until a specific point of density, after this critical point is reached the velocity degrades substantially, also called an overflow. The recovery of the throughput after an overflow is not in line with the reduction of density to a previous safe amount. The road now requires a massive reduction in density in order to climb back to the original velocity. The traffic measurement information is necessary for approximating the critical point of overflow. When this value is known it is also possible to use it for expressing actual road fullness in respect to the point of failure. The first step will be visualizing overflows over the course of a full day, the next step will be approximating the point of failure. An example plot of this traffic phenomenon is shown in figure 2.7. The visualization will allow scrubbing through the history of a road segment in order to compare how road conditions influence the maximum throughput.

Figure 2.6: Design of MSI data visualization

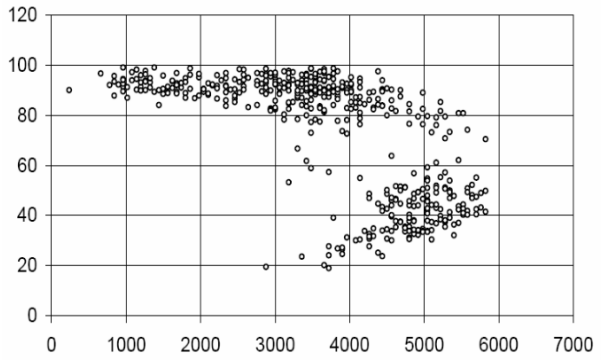


Figure 2.7: Point Cloud (fundamental diagram of traffic flow): a density velocity plot of measurements over one or more full days

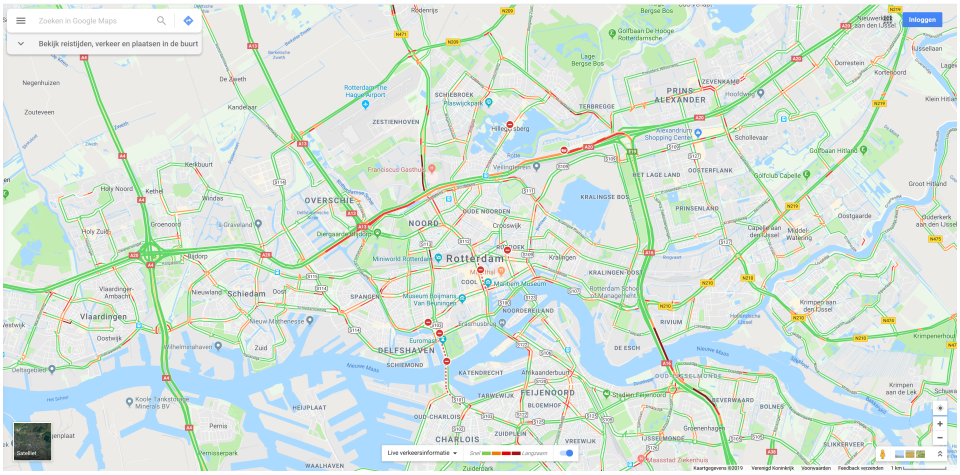


Figure 2.8: Live traffic display of Google Maps

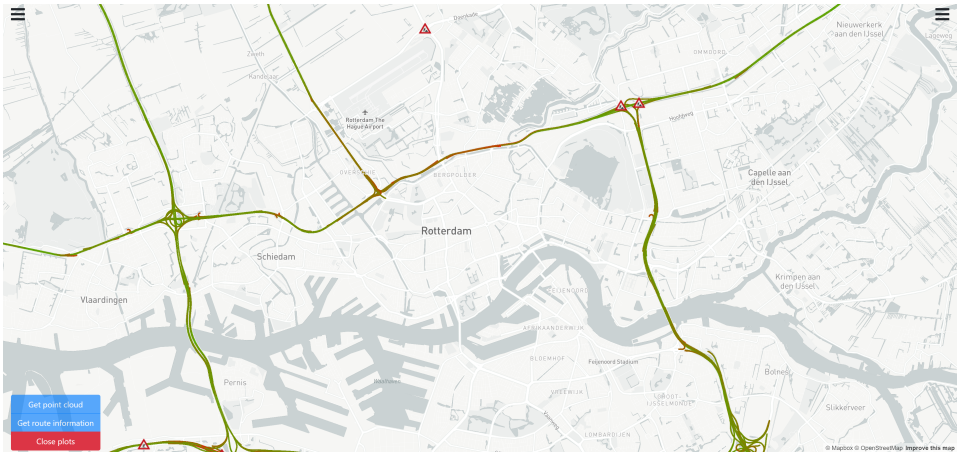


Figure 2.9: Final design of live traffic display

2.4.5. Live Traffic of the Entire Map

In order to provide context to the data that the user can visualize within the application, it is useful to have an overview of the whole road network's traffic state. The design is based on how Google Maps shows traffic intensity (figure 2.8). Essentially, an overlay is created over the road where each road segment has a color depending on the speed that cars are driving. This can point out interesting locations to retrieve more data from. As a comparison, the final design is shown in figure 2.9. Since our only data source comes from roads where measurement sites are located, there is less data to show compared to Google maps.

3

Product Description

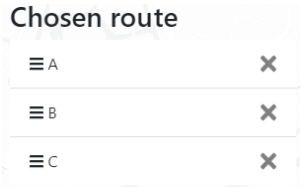
3.1. Features

This section will provide an overview of the features implemented in the developed product. These include creating routes between multiple selected points, contour plot of traffic flow in time of the selected route, point cloud that can be used to determine the maximum capacity of a road, matrix signs visualizations, road status messages and the traffic intensity of the entire Netherlands.

3.1.1. Creating Routes

One of the more crucial features is creating a route on the map. Most of the features discussed after this section heavily rely on the possibility to create a route. When the user selects two or more points on the map, a route between the chosen points is automatically created. See figure 3.1a for an example of a route. After creating a route the user is also able to reorder the points on the route and also remove points from a route. See figure 3.1b for how this works in the application. After reordering or removing points the route is again automatically updated to reflect the changes made by the user.

(a) Route between points A, B and C



(b) Reorder and remove points on a route

Figure 3.1: Visualization of selected route

3.1.2. Contour Plot

The contour plot was the most desired feature of the application by the client. The contour plot is used to provide clear and easily understandable information to not only traffic management experts, but also to people who have no experience in this field. Figure 3.2 shows a contour plot where maintenance works were ongoing. Green is used to indicate no delays on the road. The cars are driving at approximately the maximum speed allowed on the road. The colours changes to red when cars are driving slower than normal. In this case some lanes were closed. According to the contour plot, after the 5 km mark on the selected route the lanes are opened again and the traffic starts flowing again which can be seen by the colours changing from red to green almost instantly.

Contour Plot Implementation

Creating a contour plot is a multi step process that combines multiple parts of the code base. After the user has selected a route and a time window, a REST call will be made to the backend with these parameters. The backend then uses [redacted] to find all the measurement sites on the route. Using these results, the backend will query the database for all the measurements in the requested time window that have been measured

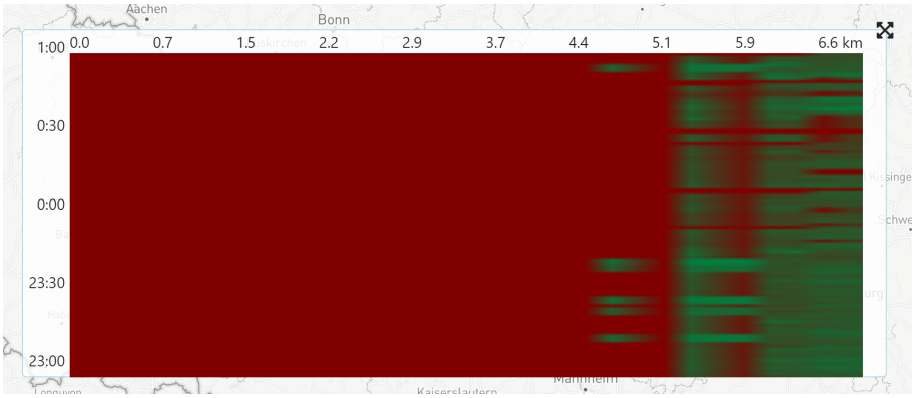


Figure 3.2: Contour plot of maintenance event

by one of the measurement sites along the route. Before these measurements can be send to the frontend, they must first be formatted into a table such that their values can be validated. The x-axis contain the measurement sites in order of the route and the y-axis contain all the measurement times in ascending order. It is possible that some of the measurement values are invalid or are missing. So the final step of the backend is to replace these missing measurements with estimations. The backend does this using a convolutional 3x3 filter on the invalid values. Invalid values will be replaced with the average value of all the other values in the filter that are not invalid. This smooths out the contour plot since the estimated value is based on a future and a past value of this measurement site and values from the measurement sites directly in front and after this measurement site. Which means that this even works if a measurement site is broken. Once all this has been done the backend will convert the data into an JSON format and send it to the frontend where it will be rendered.

Contour Plot Animation

The contour plot had to be animated as well, as stated by the requirements the animation would be based on live wind maps. These maps use small particles as an overlay with alternating velocity, density and direction. Depending on the current weather at the location of the overlay. For the Contour Plot this meant having particles that move both in the positive X and Y direction of the axes, depending on traffic speed. Such particles do come with the difficulty of being CPU intensive when rendering multiple particles, as was concluded from an earlier attempt. Due to this reason the decision was made to create custom WebGL shaders and therefore using the GPU for increased performance. Although some inspiration for the shaders was gathered from a wind map implementation [1], the bulk of the implementation was custom made. The shaders themselves are quite straightforward: upload the contour plot data as a texture to the GPU buffer, where the RGBA values are used to store directional and intensity data. Continuing by rendering particles, that query the texture containing the data for retrieving the direction and lifespan it will have. Resetting the particle is done by using a GPU based pseudorandom generator based on a fluctuating sine to determine the new location of a particle when it fades out. The colour gradient is each particle can either be a static colour or based on either horizontal or vertical velocity.

Another crucial feature of the contour plot is a mapping between a specific point on the route that is plotted on the map and that same point on the route on the contour plot. In the final product, when the user hovers over a point on the route on the map, a vertical line is drawn over the contour plot as shown in figure 3.3.

In order to do this, the location of the cursor on the route is determined initially. Since the route is a list of coordinates the closed coordinate is retrieved using a KD-tree. From that point the (Euclidean) distance to the starting point of the route is determined by summing all distances between all points that precede the highlighted point up to the starting point and this distance is normalized in a range between 0.0 and 1.0. The vertical line is then drawn over the contour plot on a separate HTML canvas which is displayed on top the contour plot on the pixels that are relatively the same distance from the left side of the plot. The point on the route is highlighted with a green dot which is removed when the user moves the mouse away from the route. However, this method of highlighting the location on the route on the contour plot is slightly inaccurate as the contour plot is only shown between the first measurement site and the last measurement site along the route. Thus the distance from the highlighted point to the starting point is not the same as the distance of

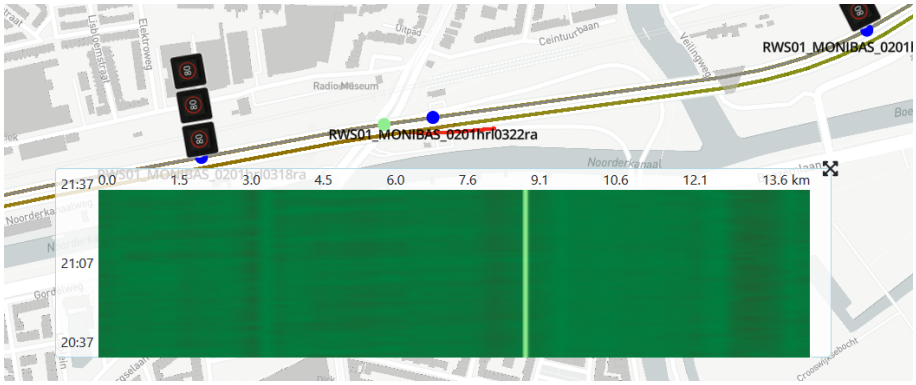


Figure 3.3: Location highlighting on contour plot, the green dot represents the location the user is hovering over

the highlighted point to the first measurement site which would be shown on the contour plot graph by the vertical line. Since the inaccuracy is minor and almost unnoticeable by the user it was decided to keep the current method to avoid an even more complex implementation.

Contour Plot Axes and Labels
The axes and their labels are generated based on the following properties: the time selected by the user, the time range selected by the user and the distances between the measurement sites along the route. The y-axis (time axis) is simply an enumeration of each half hour since the starting time of the measurements. The x-axis (distance axis) are fractions of the total distance of the measurement sites. Currently the amount of labels that are generated on the x-axis is set at 10 but this constant is mutable within the code. All labels of the axis are HTML elements within a flexbox container which are spread out to fill the entire box.

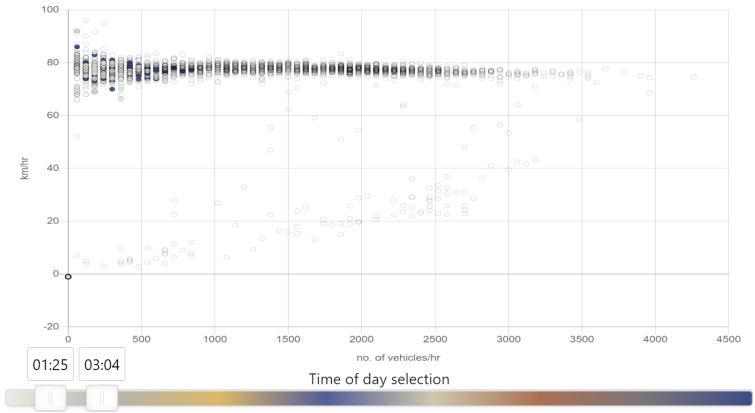
3.1.3. Point Cloud

Another coveted feature by the client was the point cloud (while the technical term is 'fundamental diagram for traffic flow', for brevity purposes we prefer to use the term point cloud). The point cloud describes the relationship between the number of vehicles per hour (intensity), number of vehicles per km (density) and the velocity of the vehicles. On the y-axis the velocity in km per hour is displayed. The x-axis is used to display the number of vehicles per hour. The slider can be used to select the time range for the point cloud. It is possible to select just an hour or even the whole day. The sliding window can be increased or decreased based on the needs of the user. An example of a point cloud is given in figure 3.4. This figure shows a point cloud generated during the night at the Kleinpolderplein interchange in Rotterdam. Using the point cloud it is possible to determine the maximum capacity of the road in this way. Using the slider it is possible to see how traffic behaves when the intensity increases. When the intensity increases, there is a point where the velocity starts to drop. Figure 3.4b shows the same location as 3.4a. In this figure the busiest time of day is picked. Using this figure it can be determined that that maximum capacity is approximately 3500 vehicles per hour. After this point the velocity starts to drop, which indicates that the road is no longer able to handle the number of vehicles.

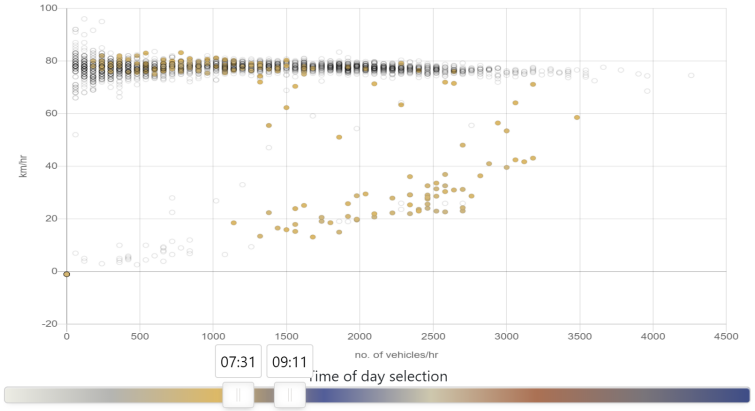
Point Cloud Implementation Details
The second visualization tool that had to be implemented was the point cloud, the design specifications are given in 2.4.4. Note that Fundamental Diagram is here called point cloud. In this chapter an implementation description will be given for the plot. The first decision made was to use Chart.js [7]. There are multiple chart frameworks available, but Chart.js is the most established. The automatic axes scaling and ability to draw many data entries made Chart.js the most suitable for this application.

As earlier described, the point cloud would need to ability to draw every measured value from one measurement point within a given time frame. For ease of use the time frame will always be from 00:00 to 23:59 of any given day. The user selects the desired days from the calender and the data from every selected date will be retrieved. Chart.js is able to quickly display all retrieves points and automatically scale the plot to a correct fit.

The ability to select a time frame within the 00:00 to 23:59 scale is fulfilled by having a slider with two handles, as can be seen in 3.4b. The distance between the handles is the length of the time frame, the position



(a) Data during the night



(b) Data during rush hour

Figure 3.4: Point cloud of Kleinpolderplein in Rotterdam

of the handles determines what time frame is active. Filtering every point in the plot on every handle update would be impossible due to the performance impact of redrawing the vertices (Points). On the other hand, only updating the fragments (Colours) has a relative low performance impact. Moving the handles will therefore only update the colours of each point rather than removing or adding them. Not only is this solution better for performance, it also makes the plot more informative due to maintaining a barely visible indication of an inactive point.

Another small feature that is difficult to implement is the difference in time zones. Although a safe assumption would be that only Dutch users would use the application, the complexity behind time zones is fully implemented for future reuse.

3.1.4. Matrix Signs

The user is also able to view the useful Matrix Sign Information (MSI) data along the chosen route. MSI data is stored in the backend as a state of a matrix sign at a time interval. Each state thus has a start time, end time, location data of the sign and a variety of properties representing the state of the sign. The states are rendered in the manner of figure 3.5.

(a) Maximum speed of 50 with a red ring

(b) Lane closed

(c) Lane closed ahead, move to the right

Figure 3.5: A selection of images showing various states of MSI data

The majority of matrix signs (under normal conditions) are blank, this data has no real value for the user. These matrix signs are discarded along the route to make the map less crowded. However if a matrix portal has a non blank sign but also has a blank sign, the blank signs are still shown. A blank sign has value if there are other signs that are not blank, such that the user can see how many lanes the road segment contains. The state of the matrix signs are represented by images of the actual state of the sign on the road. However, this is only practical at a high zoom level since at a low zoom level the signs would overlap. Therefore, three zoom levels are defined at which the representation of MSI data is different in order to keep the states comprehensible and clear for the user. In figure 3.6 the matrix signs can be seen as images. In this figure the matrix signs are displayed as can been seen while driving along the road. Every detail of the matrix sign is displayed. If a matrix sign has red dots flashing the matrix sign on the map also has flashing red dots. When the user zooms out there is not enough space alongside the road to show the matrix signs in full detail. For this reason three zoom levels have been created. Zoom level 1 shows the matrix sign in full detail as just described.

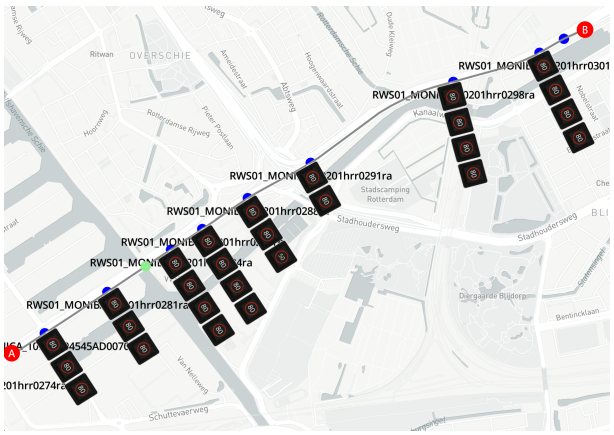


Figure 3.6: Matrix signs at zoom level 1

In zoom level 2 the matrix signs are mapped to a colour. For example if the matrix sign is blank or displays that the lane is open a green colour is used. The matrix signs of figure 3.6 all show a speed limit of 80 km/h except one that shows a speed limit of 50 km/h. As can be seen in figure 3.7 a 80 km/h speed limit is mapped to a dark blue colour and a 50 km/h speed limit is mapped to a lighter blue. All other possibilities also have their own respective colour mapping. In this way the user can still see the information that the matrix sign is displaying quickly without taking up too much space and make the map crowded with elements.

When zooming out even further than zoom level 2 the matrix sign rows are starting to overlap. So the final zoom level, zoom level 3, only shows a single colour for a row of matrix signs. The information of all the matrix signs in a single row are aggregated into a single colour. This single colour is selected based on the sign information with the highest priority. Some information is more important than others. For instance, on a portal with both a lane closed and a speed limit sign, it is more important to know that the portal has a closed lane for the user thus this colour is displayed for the entire portal. The priorities are defined as follows (from high to low): restriction end, lane closed, lane closed ahead, speed limit, lane open.

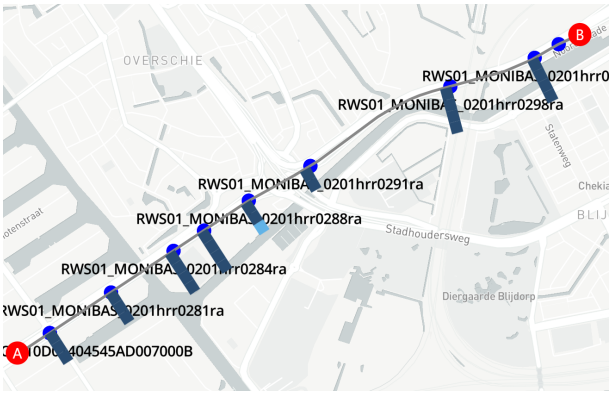


Figure 3.7: Matrix signs at zoom level 2

Figure 3.8: Matrix signs at zoom level 3

Matrix Sign Implementation Details

Matrix sign information for a sign that is received from the backend is stored in a `MatrixSignInformation` object. This object contains all properties of the sign's state at a time range, in addition to its location on the road network. When matrix sign information is retrieved along a route, all states can be grouped in layers. In the application, there exist three layers of abstraction for the states of matrix signs. First of all there exist matrix sign portals along the road. Each portal has one or more matrix signs along it so these need to be grouped together in order to reflect the actual arrangement along the road. Each portal has, as mentioned, a group of matrix signs. Finally, we consider that each sign has a history of states which are grouped as well. This abstract reasoning yields a data structure which is a three-dimensional array of matrix sign states. An illustration of this abstraction can be seen in figure 3.9.

A request to the backend for matrix sign information contains the start and end point of the route as well as a time range for which to retrieve the states of the matrix signs. The backend retrieves all matrix sign states and groups them in the structure described above and returns that data structure to the frontend.

The data is then plotted according to the zoom levels that have been defined previously, an HTML element is created for each zoom level per portal and plotted on the map as a Mapbox marker element. The elements are constructed based on the properties of the sign's state. For example, the sign may be a speed limit sign that is flashing, this requires a different image to be loaded. Matrix sign images have been created for all known possible combinations of properties. Future extensions of possible states should be added to the image library. When the zoom level changes, elements are displayed or blocked based on the zoom level such that the correct element is shown. This mechanism for showing matrix sign states puts a high computational load on the frontend as all elements are still rendered at all times even when only a single zoom level is displayed. This still works, however, when only matrix sign states are shown along a selected route as not a lot of elements are required to be plotted. On a larger scale, this mechanism would introduce performance problems.

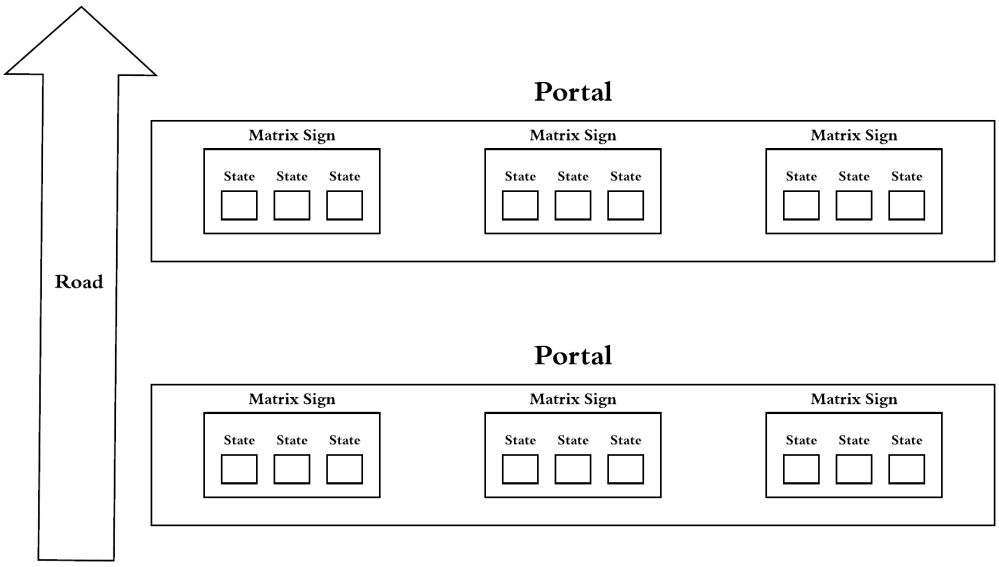
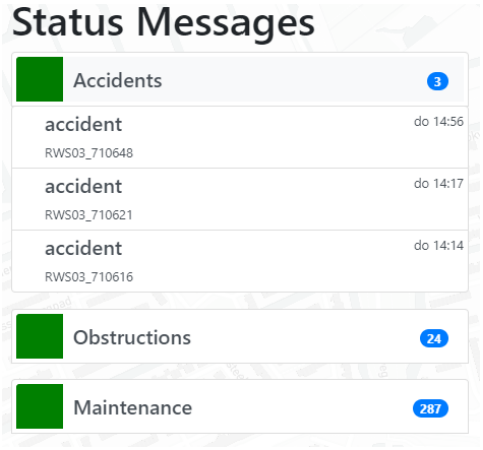


Figure 3.9: Illustration of the matrix sign information data structure as viewed along a road.

3.1.5. Status Messages

The NDW also provides information about the status of the roads. This information includes accidents, obstructions and (planned) maintenance. Each accident, obstruction and maintenance is shown on the map. An example of what an accident looks on the map is shown in figure 3.10a.

(a) Accident plotted on the road



(b) Status messages of accident

Figure 3.10: An accident and its corresponding status message

For usability reasons, not all information about the status messages is shown on the map. The user can see some information by hovering over a status message, but most data is shown on the right side bar. This bar is shown when a user clicks on a status message. On the sidebar, the user can also choose to show or hide some status messages on the map depending on the category. For example, a user that is only interested in accidents can only show accidents. Because there are many maintenance events at any given time, there are more options to filter those types of events. Users can filter on status messages by "gemeentes", "provincies" or "rijkswaterstaat". Users can also sort the maintenance status messages by start time and expected end time.

3.1.6. Live Traffic Intensity

The map also displays the traffic intensity of all the highways in the Netherlands. An example of roads near 's-Hertogenbosch is shown in figure 3.11. Green as expected indicates that the traffic is moving without any delay. Some parts of the road in 3.11 have a darker colour, which means that there is some kind of delay on the road segment. The darker the colour, the higher the traffic intensity. Dark red means that the traffic is almost standing still on the road. This feature can be toggled on or off depending on the user' preferences.

Figure 3.11: Live traffic intensity overlay

3.2. Technical Details

This section discusses the technical details of the features discussed in the previous subsection

Graph solutions for Route finding

After acquiring and storing the data streams, a common limitation was found in the data, namely the lack of proper linkage between map locations and data point locations. This bridge exists due to the usage of a multitude of different location structures in the data streams, which are incompatible with conventional maps. Our requirements go even a step further by necessitating the knowledge of the physical locations of all data inputs that are received. Requiring not only the visual location by providing a latitude and longitude but by also knowing on which physical road every point lies on, gives the possibility of locating all data points that lie on any given road.

Graph Model Alternatives

Comparing Graph models will be necessary to find the right solution for the application. The system requires the location of all measurement sites in the Netherlands in order to correctly represent their data and further fulfill the needs as described in 1.3. The main requirements for this chapter being: path finding, listing points on a path and listing matrix signs on a path. The main challenge being able to know which road is being measured by what site. The simple solution so called 'snapping', would be to find the closest road to a measurement site and use that road as reference. Although initially appearing to work, on closer inspection, said approach makes many subtle mistakes when roads cross or lie close to each other. In this chapter we will further discuss the potential solutions that were explored.

- **VILD**
  As explained in 2.3, there are a multitude of streams: measurement sites, measurement trajectories and matrix sign information. The measurement streams provide a seemingly straight forward system for mapping the measurement sites onto a road, namely VILD the Dutch implementation of Datex2, which is a graph model of all mapped roads. Path finding is then performed by running Dijkstra's algorithm over the graph. However, VILD does not contain road shapes, only nodes and edges. This would not be a problem if VILD contained many points, the shape could then be determined as straight lines between all passed nodes. Solving the lack of shapes is done by snapping the VILD points onto known roads, and using the shape associated to the snapped road for drawing paths. Which implies that instead of using the simple 'snapping' solution to map each measurement site, we now 'snap' each VILD

point. Resulting in greater complexity and equal reliability. Another drawback of using VILD is that it makes our application dependent on the Dutch road network and limits the migration ability to other countries. These facts combined with the lack of VILD definitions for matrix signs, made it so that VILD could no longer be used to suffice the requirements. There are also no available solutions for combining VILD and any map and NDW has no intention of providing any.

- [redacted]

Due to NDW transitioning all data streams to support the VILD location format and their general recommendation to use the VILD system, the initial decision was made to implement the VILD method. However after discovering the above mentioned limitations regarding path finding, road geometry, general map compatibility being non-existent and realizing that a lot of the data streams do currently not implement the VILD location format, the decision was made to transition to [redacted].

Modifications
Multiple modifications and extensions were made for [redacted] in order to make it fully compatible to all requirements, the goal was to provide all map related functionality by only requiring a self hosted graph. This meant being able to:

- Find a path between two or more points, the path must be traversable either by car or truck.
- Match all data point location to an edge in the graph, how this is performed is explained in 3.2.1
- List all data point locations that lie on the path, in order of traversal.
- List all distances between each traversed data point.
- Have physical knowledge of what data point lies before or after any data point on highways. The necessity of said knowledge further elaborated on in 3.2.2

To accomplish the aforementioned abilities, the following modifications or additions were made to [redacted]:

- **Vehicles**
  [redacted]
- **Storage**
  [redacted]
- **Optimization**
  [redacted]

The graph stores a total of 130MB of data for storing all road data from the Netherlands.

3.2.1. Graph matching

As earlier explained in 3.2 the [redacted] has no proper way of linking data points to the internal graph. This meant finding a way to correctly place every data location on a direction based edge. Important to note is that although some edges are bidirectional, all data points are onedirectional. This means that for bidirectional edges the individual point directions and order needs to be preserved. For onedirectional roads only the order of points is necessary to preserve. In this chapter an explanation will be given about how the matching of data points is achieved.

The problem as can be seen in figure 3.12, the white dot represents a measurement site located on a highway at the exact point where multiple roads cross. The simplest implementation of a 'snapping' algorithm will in this case not be able to correctly place the site. Snapping in this scenario would first find a path, and try to find all sites that lie on the calculated path. The inability does not arise from lack of directional knowledge of each site, but rather from having no knowledge of all other roads that do not lie on the calculated path. However, since [redacted] has an internal graph of all traversable roads it also has the ability to more reliably connect each site to an edge. This is also known as the Map-matching problem [9].

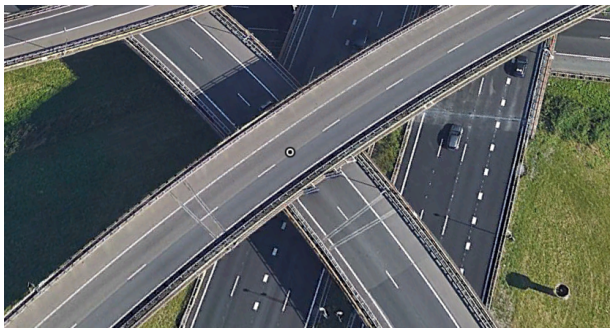


Figure 3.12: Example of a complex matching scenario, from Google Earth

Simulating GPS

NDW delivers quarterly updated shapefiles [41], these contain the longitude latitude and direction of every measurement and MSI location. The file we originally used was quickly deemed unusable due to half the locations being grossly misplaced. Therefore the longitude, latitude and direction were gathered from a combination of VILD and XML files, this resulted in a total of 13000 usable measurement locations. After a couple of weeks of relying on said method, NDW updated the shapefiles. They were now deemed better than before, resulting in a total of 18000 usable measurement locations. The given values were also more accurate and prompted a permanent switch to the file. Matching the data is performed by using the [redacted] plugin. This allows us to match GPS paths onto the graph. For every point that needs to be matched, a custom GPS path is created. Consisting of points that are generated in relation to the longitude latitude from the original point, but then translated over the direction as shown in figure 3.13. This is done using the formulas as explained by 'Vincenty solutions of geodesics on the ellipsoid' [38]. Using basic geometry is not advised due to the potential offset that can arise. This process is performed for the measure-

Figure 3.13: Example of a constructed GPS path, from Google Earth

ment and matrix sign locations, where the system is unable to place 20 locations. Although this process is executed within a couple of minutes, the storage mapping is written to a file to allow for faster startup times. The custom matching method now gives:

1. An OpenStreetMap based graph network, with the ability to use all OSM data.
2. Most data points are connected to the graph.
3. The ability to found routes and retrieve all data point and OSM data from this route.

3.2.2. Live Traffic

The NDW also delivers road trajectory speeds. In other words, the average speed it takes cars to travel over a specific segment. These segments are also delivered, however were on initial inspection seen as unreliable due to the many faulty values. Instead the decision was made to use the measurement points instead of trajectories, which also gives the ability to not only use speed but also traffic intensity for each segment. However this also limited the trajectories to highways due to them being limited in possible directions. For every point on the highway, the points that are next to approach must be found, this can be just one point or multiple in the case of intersections or exits.

Calculating Points

The calculation can be described as: for every point find all routes to every other point. Remove all paths that cross an edge that has a point on it that is not the begin or end point. What is left will be a list of paths that connect all highway point in the correct order. Important is to note that the actual algorithm uses the internal graph to use depth first search, when an edge with a point is found, the search continues only not via the found edge. The graph is also used to calculate and store the average allowed max speed. All found connection and speeds are then stored for faster startup times.
Although initially deemed unfit, the client requested the trajectories to be considered. The trajectories do cover city segments, which could provide useful information and since the framework for adding the trajectories was already present. The decision was made to also include the road trajectories as an option for displaying traffic intensity.

Retrieval

Every minute the traffic speed is received and the data in the structure is updated. The decision was made to not query the database for the traffic speeds due to high cost of the entire aggregation, instead the live data only uses the most recently gathered measurements. This removed the need for querying the database. Another issue is measurement sites that send error values. To circumvent the simplest errors a fallback mechanism is implemented that tries to find the intensity of the next or previous point in line to replace the missing point. Such a system is not in place for the trajectories due to the inability to correctly identify fallback trajectories.

3.2.3. Updating Plotted Data

The application offers the possibility to update plotted data continuously. The plotted data contains: the live traffic overlay (both normal and trajectory based), status messages and contour plot. However, this is only desired when the user is viewing current non-historical data. The user activates periodical updates of the data by pressing the "now" button next to the time selector of the contour plot. This button (shown in figure 3.14) initializes an update function which executes once every five minutes. At each interval the update function updates the time on the time selector and consequently makes the required API calls to update all plotted data mentioned before. The same functions are used to reload all the plotted data with the newly requested data. The contour plot is only rendered again if it was already rendered and active.

3.3. Implementation Challenges

3.3.1. Route Finding

The challenge with route finding was not the lack of tooling or difficulty, but rather the lack of self hosted routing engines. Most available tools are in the form of API's that have a usage limit, which is understandable but also difficult to work with during development of certain features that are currently implemented.

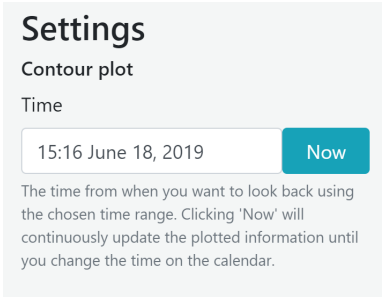


Figure 3.14: Time selector with update button for contour plot

3.3.2. Data Compatibility

One of the most noteworthy challenges is the incompatibility of the location data and any map framework other than VILD. There are multiple other online experiencing the same difficulties, but without a proper solution. Combining the incompatibility with sometimes inconsistent or blatantly wrong data delivered made solving the issue even more difficult.

3.3.3. Parsing

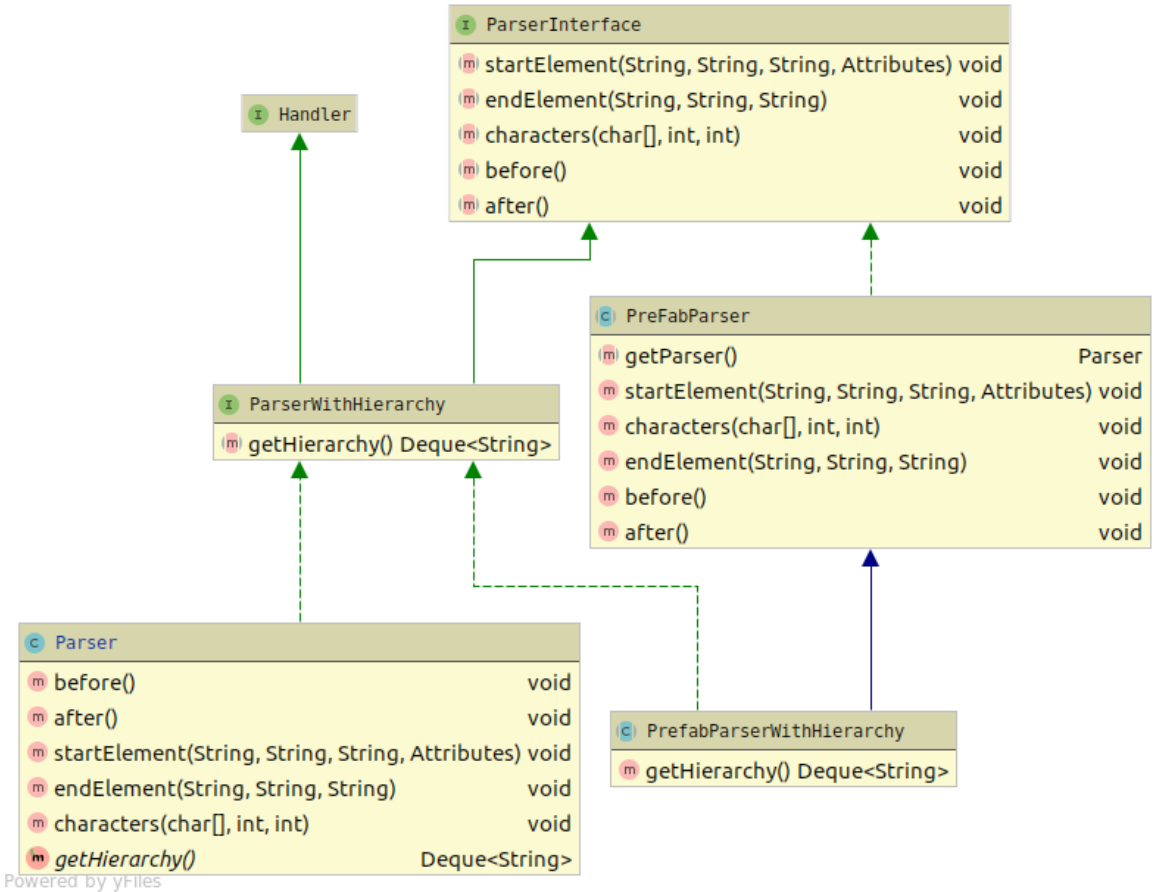


Figure 3.15: UML diagram of Parser

As the project progressed the parsing of more and more complex XML files was required. At first, only the traffic and traffic configuration XML files needed to be parsed and their information stored in the database. Later on the status messages and MSI XML files were also required in order to show and save status messages and matrix sign states. At the same time, the structure of the status message file is quite complex, since it

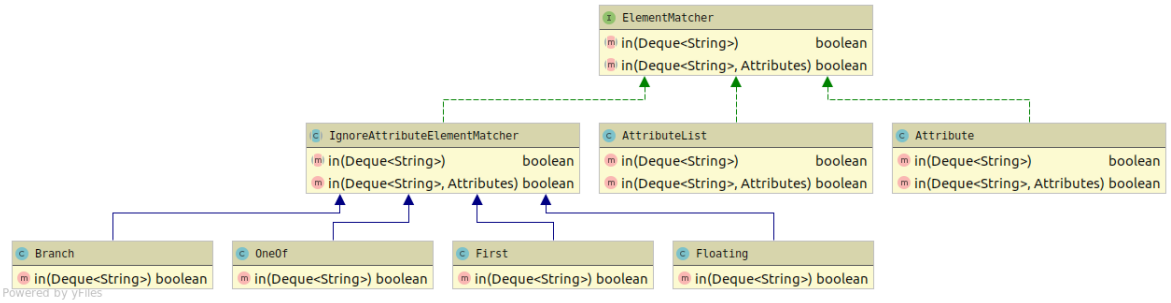


Figure 3.16: UML diagram of Matcher

needs to be able to save accidents, road closures, emergency vehicles and many other types of status messages (like warnings of wild animals on the road). While the SAX parser API [30] that we were using thus far was very fast, it is also a pain to write parsing code with. Switch statements with up to 20 cases and methods of 200 lines were not uncommon. To improve the maintainability of the code, we decided to write an XML parser library that wraps the core SAX parser that made writing XML parsers a much easier task.

The core of the library is the `Parser` class, which can be seen in figure 3.15. The Parser class itself extends the `DefaultHandler`, the SAX parser parsing entrypoint. The idea is that the user can create a `Matcher` (see figure 3.16) that matches an XML node based on some criterion and a `Handler`, which is an anonymous function that runs when the matcher matches an XML node. With this matcher and handler, the user can easily retrieve only the part of the XML that they want in just one pass over the data stream. The matcher has access to the current XML hierarchy, not only the current XML element. In this way it is possible to create more complex matchers that only match element A if it is a descendant of element B. This is especially useful since it is common for the XML files that we need to parse to have a child element that has the same name as the parent element.

To ease the parsing of complex XML files, an extra feature that is available are child parsers. When a matcher matches an element and it has a child parser attached, the child parser takes over execution from the parent parser and executes its own parsing code until the element that triggered the parser is exited again. This feature makes it possible to write a "location" parser and a "time" parser that only run when in the relevant part of the XML document.

These features were more than enough enough to rewrite all parsers except for the status message parser. The structure of this XML file was so complex that it needed some extra features. An attribute (not an element) at the beginning of a status message decides the type of the status message. Every type of status message has some unique elements, that all need to be handled in a different way. Depending on the Status Message type a specific child parser needs to take over execution. For this the `ElementMatcher` interface gets an extra parameter: the attributes of the element. This enabled a matcher to match the attributes of an element, besides just the name of the element itself. Some of the child parsers also had shared functionality that could not be delegated to a common child parser. For this purpose, sibling parsers were added. These parsers run unconditionally alongside the main parser.

For convenience, some extra classes (see image 3.15) were added. The `Parser` is meant to be built by a `ParserBuilder`, which has a convenient (builder pattern) syntax for creating parsers. The `PrefabParser` allows the `ParserBuilder` to create a parser class instead of just an object. The parser needs to be created on the object constructor of the `PrefabParser` and later it needs to be called with the `getParser` method. This allows the parser to be its own class that can keep track of its own state. This is a useful feature for more complex parsers.

All these abstractions decreased the size and complexity of all the parsers dramatically. Size decreased on average by about 70% and almost all branches were eliminated.

3.3.4. Domain Knowledge

Another challenge that was more complicated than expected, was the domain knowledge required for the application to work. While the client was very forthcoming with information about traffic, a lot of details were not obvious. For example, when detecting the average speed along a road segment, the name of some measurement points is prefixed by `MONIBAS`. Other measurement points have the prefix `MONICA`. It turned

out that MONIBAS and MONICA measurement points give information about the same physical measurement points. The only difference is that MONIBAS points are post processed to improve accuracy.

3.3.5. Map Performance

Using MapBox to draw all measurement points found on a single route seemed to be a non-issue performance wise. Drawing all 6000 paths retrieved from the live traffic, all coloured according to the intensity was deemed problematic. DeckGL [16] appeared to be a suitable solution for our use case, since it also comes with MapBox integration. Their site demonstrates 126.000 paths without problems.

Since the MSI data that is displayed is one of the main performance bottlenecks DeckGL could provide a solution for plotting MSI data. The current implementation renders three HTML elements simultaneously impacting the performance for large sets of data. In fact, an attempt to plot *all* MSI data on the map would crash the application while this may be desirable in some cases. However, migrating this implementation to use DeckGL was out of scope for our project so this remains one of the performance bottlenecks.

3.3.6. Database Performance

One of the goals of this project was to be able to store roughly 4 weeks of data. Making it possible for the user to be able to visualize any data from this history. Since there are 15,380 different measurement site, each producing a new measurements every minute, the database has to store 682.940.160 measurement records. Due to the large amount of records in the table it is very important that the table is indexed correctly. If this not the case database might need preform a table scan, which will significantly hurt the performance of the application. Thus the most expensive queries within the application are now examined.

Contour Plot

The contour plot needs all the measurements from a varying amount of measurement sites in a fixed window of either 1, 2 or 4 hours. Each measurement site has its own unique ID, which means that the performance of the query can be improved by indexing these site IDs. The query also has a range part, in which it checks if the measurement is in its time window. This part of the query is rather expensive but its performance can be improved by indexing the time of the measurements in descending order. Which significantly reduces the amount of measurements that need to be scanned. Combining both these indices significantly improves the performance of this query as long as all the data is gathered in a single query. Naively, in one of the earlier versions, the application created a query for each of the measurement sites and queried the database for their measurements in parallel. Initially this approach was faster, however performing the queries in parallel proved to be a bottleneck as the amount of records in the database grew over time. That is why the final version of the application performs only a single query with an `OR` condition for each measurement site, as suggested by Asterios Katsifodimos. This makes the formatting of the Contour Plot a bit harder, in the sense that the application needs more complicated code to format the results from the database in the desired format. However this is more than worth it since the back end of the application is now able to respond with a contour plot in within 8 seconds.

Point Cloud

The point cloud plot needs all the measurements from a fixed site in a varying amount of time ranges, for example different days of the week. This query also needs to find measurements from a specific site, so indexing the measurement site IDs will also improve the performance of this query. However the effect of this index is far smaller on this query compared to the Contour Plot query, since the point cloud plot query has a varying amount of windows. Which means that this query needs to check multiple range conditions, which are much more expensive. So even though the database has indices on both the measurement site ID and the measurement time in descending order, the query still took more than 30 seconds.After further examination of the data we realized that the measurement times are actually discrete values, since they don't take seconds or milliseconds into account and thus increase incrementally every minute. So we used this property to transform the range queries into `WHERE IS` queries. This already decreased the query time to roughly 15 seconds. However, due to the large amount of different `WHERE` conditions, one for every minute in the time ranges, the query was still not performing optimally. So finally we tried to limit the amount of `WHERE` conditions by sampling a fixed amount of measurement times from the different time ranges. By limiting the amount of samples to roughly 5,000, the point cloud plot still looks exactly the same to the human eye as the original plot. While the application is able to compose the point cloud plot in roughly 11 seconds for a measurement site, which is a significant improvement.

4

Code Quality & Testing

4.1. Code Quality

This section discusses the different tools that have been used to ensure the code quality.

4.1.1. Continuous Integration

For our project we used Travis CI [36] as our Continuous Integration (CI) tool. This CI was used to verify if both the frontend and backend compiled, adhered to the code quality standards and that both their test cases still passed. If this was not the case, the programmer that introduced the change that broke the build was notified automatically. The CI also prevents developers from merging broken builds into the 'master' branch by only allowing developers to merge with the master branch if all the verification steps pass.

4.1.2. TypeScript

Initially JavaScript was used as the scripting language for the frontend, it turned out to not scale well with the increasing codebase of the project. Most code was cluttered into a few files and the code was unreadable. Therefore, halfway throughout the project significant effort was made to convert the JavaScript codebase to Typescript [37]. Typescript introduced static-typing and Object-Oriented design to the project. This greatly improved readability of the code and split code over separate modules. Thanks to the static typing of TypeScript, it was now also possible to use the CI to verify if all the types in the project still matched after changes have been made.

4.1.3. ESLint

To further ensure code quality in the frontend, ESLint [25] was included into the project. ESLint provides some basic rules to ensure code quality. It is also able to automatically fix the most basic quality issues like: incorrect indentation, missing semicolons, etc. It can also run static analysis such that a CI build will fail if there are quality issues in the code. For a significantly sized project like ours, this can contribute significantly to the quality of the codebase as group members have an automatic feedback tool on their code quality.

4.1.4. Checkstyle

The code quality of the backend was checked by the CI using CheckStyle [8], PMD [26] and FindBugs [12]. These tools ensures that everyone adheres to the same writing style based on a set of predefined rules. Originally we did not use these tools, however as the project progressed we saw that everyone had their own way of writing Java code, which made the project quite chaotic. So this made us decide to add CheckStyle, PMD and FindBugs to our project.

4.2. Testing

This section will discuss how we have tested the application and the results of these tests.

4.2.1. Frontend Testing

After we added TypeScript to the project described in section 4.1.2, we were now also able to add tests to the frontend. In the beginning this was hard to setup. We encountered many problems setting up the test environment. The main problem was that it is hard to emulate the DOM model in a test. For example: packages such as Mapbox will send requests to their APIs to retrieve data, but this is not possible using an emulated DOM model. As a consequence classes that use Mapbox were difficult to test. That is why we would recommend adding UI tests using a real browser (see section 8.1.1). The classes that live in more isolation, such as a utility class, are completely tested since these methods do not rely on a DOM model and have no external dependencies. In the end we still managed to create 55 tests. This is a lot since several classes depend on Mapbox which are not possible to (unit) test.

TS-Mocha is used in the project as a testing framework [21]. TS-Mocha is wrapper around Mocha which is JavaScript testing framework [20]. TS-Mocha is necessary, because we added TypeScript to our project.

4.2.2. Backend Testing

The backend of the application has been tested using JUnit5 [35]. The test suits mainly consists of unit tests but there are also some integration and smoke tests. For example, the test suite uses an in memory embedded database to test the migration schema of the database as a smoke test. This embedded database is also used for many integration tests between the application and the database. The test suite also has a smoke test that makes sure that the server has been configured correctly by starting up the entire application.

4.2.3. Coverage

As can be seen in figure 4.1 the backend of the application has a branch coverage of 90%. In this test coverage report we excluded the auto-generated code from Lombok [28] since they already have an excellent test suite for their own code. The Spring server configuration classes have also been excluded since these classes contain many branches that can never be triggered but are needed for the boilerplate code and thus create a misleading metric.

4.3. SIG Report

On the 11th of June we received the feedback report from SIG. According to this report we scored 3.1 stars on their maintainability model, which means that the code quality of the application is around market average. They provided us with 3 points of improvement. The following sections discuss each of these points and how we tried to improve these points. The complete report can be found in appendix C.

4.3.1. Unit Test-code

One of the observations that SIG made was that there were very few test cases in the uploaded code. This observation was completely right since we had written very little test code up to that point. The reason for this was that up to that point we were mainly exploring how specific functionalities could be implemented. Due to this exploration and often unclear functional requirements we often had to delete or rewrite large parts of our codebase. To keep exploration as fast as possible, we decided we would only write test code for the parts with clear requirements and a valid solution. In addition, setting up testing on the frontend was a big challenge due to compatibility issues between the many frameworks.

Sadly, we were still exploring possible solutions for a large part of the code we uploaded to SIG, so that is the reason why the amount of test code was so smalled. However, in the weeks following the upload, the exploration was finished and we clearly defined the functionality of the different parts of the codebase. With these definitions we were able to significantly improve the amount of automated test cases as can be seen in the test coverage report in 4.2.3. Overall, we conclude that this feedback had been implemented properly.

4.3.2. Unit Interfacing

Another observation that SIG has made was that the application had a relatively high percentage of functions with more than industry average amount of parameters. This mainly occurs in the frontend part of the application. In JavaScript it is very tempting to pass a large amount of parameters instead of creating new classes.

scenwise

scenwise

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
nl.tudelft.scenwise.pathfinding.locationLinks		96%		88%	12	65	10	203	1	16	0	1
nl.tudelft.scenwise.parser.matcher		94%		85%	6	34	0	43	0	14	0	7
nl.tudelft.scenwise.roadMeasurements.retrieval		86%		86%	11	78	28	204	8	59	2	9
nl.tudelft.scenwise.pathfinding.graphhopper.osm		41%		72%	9	18	36	64	6	9	0	1
nl.tudelft.scenwise.pathfinding.locationMapping		90%		92%	5	40	22	179	2	18	0	3
nl.tudelft.scenwise.intensity		87%		89%	3	22	18	98	0	8	0	3
nl.tudelft.scenwise.config		61%		25%	9	14	13	29	7	12	0	3
nl.tudelft.scenwise.pathfinding.graphhopper		75%		88%	4	23	11	56	2	14	0	3
nl.tudelft.scenwise.statusMessages.retrieval		86%		84%	8	75	33	221	6	68	1	12
nl.tudelft.scenwise.roadMeasurements.persistence.custom		80%		50%	2	8	3	23	0	6	0	2
nl.tudelft.scenwise.measurementSites.persistence		60%		50%	2	4	2	6	0	2	0	1
nl.tudelft.scenwise.parser		100%		97%	1	57	0	113	0	38	0	5
nl.tudelft.scenwise.msi.persistence		100%		96%	1	22	0	28	0	7	0	2
nl.tudelft.scenwise.pathfinding.locationMapping.roadSegments		89%		95%	9	31	12	73	8	21	0	3
nl.tudelft.scenwise.pathfinding		89%		92%	1	12	9	61	0	5	0	2
nl.tudelft.scenwise.roadMeasurements.pointCloud		100%		91%	1	16	0	32	0	10	0	5
nl.tudelft.scenwise.roadMeasurements.persistence		94%		75%	1	8	1	21	0	6	0	4
nl.tudelft.scenwise.roadMeasurements.contourPlot.gathering		100%		100%	0	41	0	95	0	24	0	5
nl.tudelft.scenwise.msi.controllers		86%		100%	3	24	12	72	3	14	1	6
nl.tudelft.scenwise.msi.retrieval		85%		100%	4	31	13	70	4	27	1	3
nl.tudelft.scenwise.pathfinding.data		91%		100%	0	8	6	40	0	5	0	3
nl.tudelft.scenwise.utils		100%		100%	0	9	0	17	0	6	0	1
nl.tudelft.scenwise.roadMeasurements.contourPlot		100%		100%	0	6	0	11	0	3	0	2
nl.tudelft.scenwise.pathfinding.locationMapping.roadLocations		100%		100%	0	5	0	14	0	4	0	3
nl.tudelft.scenwise.config.persistence		0%		n/a	7	7	30	30	7	7	1	1
nl.tudelft.scenwise.measurementSites		21%		n/a	2	3	9	11	2	3	1	2
nl.tudelft.scenwise.intensity.retrieval		82%		n/a	3	17	9	45	3	17	1	3
nl.tudelft.scenwise.statusMessages.controllers		0%		n/a	3	3	4	4	3	3	1	1
nl.tudelft.scenwise.pathfinding.locationMapping.trajectSpeeds		68%		n/a	1	4	5	15	1	4	0	1
nl.tudelft.scenwise.intensity.controllers		0%		n/a	3	3	3	3	3	3	1	1
nl.tudelft.scenwise.roadMeasurements.road		0%		n/a	1	1	2	2	1	1	1	1
nl.tudelft.scenwise.errors		86%		n/a	1	6	2	12	1	6	0	3
nl.tudelft.scenwise		37%		n/a	1	2	2	3	1	2	0	1
nl.tudelft.scenwise.config.view		0%		n/a	1	1	1	1	1	1	1	1
nl.tudelft.scenwise.config.swagger		96%		n/a	1	2	1	7	1	2	0	1
nl.tudelft.scenwise.roadMeasurements.contourPlot.rest		100%		n/a	0	4	0	16	0	4	0	1
nl.tudelft.scenwise.config.security		100%		n/a	0	2	0	14	0	2	0	1
nl.tudelft.scenwise.roadMeasurements.pointCloud.rest		100%		n/a	0	2	0	5	0	2	0	1
Total	1,163 of 8,683	86%	50 of 506	90%	116	708	297	1,941	71	453	12	108

Figure 4.1: Test coverage report of the backend

TypeScript on the other hand, encourages developers to encapsulate behaviour into new classes. As mentioned before in section 4.1.2 we migrated from a JavaScript based frontend to a TypeScript based frontend. During this migration we tried to address this issue as much as possible. However some artifacts from the JavaScript origin still exist and remain hard to solve.

4.3.3. Unit Size

SIG also observed that we had a relatively high percentage of code which unit length is longer than industry average. This usually happens when specific parts of the code have more than one responsibility. Reviewing the code that SIG referenced led us to realize this issue was indeed apparent. For example: our parser had long methods which made it hard to understand. This is why large parts of the code were refactored by splitting up long methods into shorter private methods and why some classes that had too many responsibilities were split up into different classes. For example: the parser has been refactored into a builder and delegation pattern distributed over different classes. With this refactoring the parser has not only become much easier to understand but also more customizable and testable.

5 Process

5.1. Workflow

The group used an Agile workflow during the project. This meant that there was a general road map for the project but the daily and weekly planning were allowed to vary based on the current issues. The group did this by keeping a prioritized list of issues and tasks that needed to be resolved. Every time a developer encountered a problem he would add a new task that described the problem to the list. The group also received a lot of feedback during our weekly meetings with the client which resulted into a lot of additional tasks which were all added to this list. Every Friday the group went over this list and assigned the tasks to persons best suited to this task.

To ensure that the code was of sufficient quality, every pull request on GitHub had to be reviewed by at least one developer before it could be merged into master. We also agreed that pull requests were to be reviewed by someone who had knowledge of the code. This made sure that no breaking changes were accepted in master. Also several checks were required to pass as mentioned in section 4.1.1.

5.2. Meetings

As a group, meetings were frequent since most days everyone worked together at TU Delft. Meetings with the client occurred weekly and meetings with the coach infrequently (only during important parts of the projects).

5.2.1. Group Meetings

Official group meetings were held three times per week, on Monday afternoon, Wednesday morning and Friday after the meeting with the client. During these meetings, we discussed progress and planned and subdivided the work for the next couple of days. Besides the official meetings, joined group working sessions were held frequently, so if someone had a question, it was very easy to get an answer immediately.

5.2.2. Communication With Client and Coach

Every Friday, we had a meeting with our client, Kin Fai Chan. Besides his engagement with and enthusiasm about the project, he could also provide us with very useful feedback due to his extensive knowledge about the problem that we tried to solve. Besides the weekly meetings, there was very frequent contact through email, such that any unclarities could be resolved before the meeting. Because of the many moments of feedback, we are confident that we delivered a product that the client is happy with.

Because we encountered no big road blocks during the project, the meetings with Asterios Katsifodimos were relatively sparse. At the start of the project, he ensured that the project was challenging enough for a bachelor end project, and in the middle of the project he provided valuable insight on the improvement of the database efficiency.

5.3. Intergroup Challenges

Our group has done several projects with the same team members before the start of this project. Since we worked together on previous projects we were comfortable with working together with each other. This benefitted us greatly since we had no startup issues and we could basically start right away.

The main challenges in the team were the quality checks in the code. We had to balance quality versus effort and sometimes this caused some small frustrations in the team. Some team members preferred to spend more time improving the quality of the product, but other team members found the code to be sufficient already and wanted to allocate time more pragmatically. By having frequent discussions about the workflow, these issues could be resolved. In the end this ensured that we had less technical debt, but still implemented the required features with a better than sufficient quality.

6 Evaluation

Our project is evaluated based on two main aspects: the fulfillment of the requirements by the product and the evaluation of the project according to the methodology and process. In addition, the ethical implications of our project will be considered as well.

6.1. Evaluation According to Requirements

Together with the client, a list of requirements was constructed outlining the various requirements that the final product needed to satisfy. When this list was constructed, it was considered that this list of requirements would change throughout the project albeit in written or spoken form. As with any software development project, project demands and product requirements change based on progress or evolving insights into the problem. Such was also the case within our project. Although we satisfy almost all requirements of the basic features, new features have been added as well and some requirements abandoned due to reconsidered priorities.

An overview of how our product satisfies the requirements defined at the beginning of the project is given in table 6.1. There are some requirements not fulfilled or which have a "partial" satisfaction label.

6.1.1. Partially Fulfilled Requirements

There are two requirements which have been partially fulfilled by the final product. The first of which is the requirement that queries should not take more than 5 seconds. This has been partially fulfilled as not all queries are able to have such a quick response. Table 6.2 shows the average response times for typical queries.

Both the contour plot and the point cloud lack a quick response time despite significant optimizations in the backend. However, the features which do require a quick response time from the server, as these are executed without an explicit user command, do satisfy this requirement. These are: find a route, updating status messages and live traffic data.

The other requirement that was partially satisfied is the requirement that effects of matrix sign information can be measured. While no separate feature has been implemented that explicitly measures effects of matrix sign information on traffic flow. The user is visually able to identify the effects of matrix sign information on traffic flow based on the matrix sign history and the corresponding contour plot.

6.1.2. Unfulfilled Requirements

While initially the aim of every project is to fulfill every requirement, it often occurs that some requirements are not met, whether it be because of time constraints or unexpected difficulties in implementation. The unfulfilled requirements within our project were mainly discarded requirements; requirements that were ignored resulting from discussion with the client about prioritization of some features over others.

The first unfulfilled requirement is the requirement that travel time can be calculated based on measurement data from the contour plot. After discussion with our client we concluded that this functionality is not suitable for our target audience. Traffic management experts are rarely interested in total travel time between points on the map but rather in delay times at certain road segments. Implementing a travel time estimator would require significantly more data about the road network such as maximum speed limits and the imple-

Category	Requirement	Fulfilled
Must Have	The application shows an overview of the Dutch road network	Yes
Must Have	The user can select two points to indicate a route	Yes
Must Have	A (dynamic) contour plot of any route can be generated	Yes
Must Have	The application updates traffic data in real time	Yes
Must Have	Queries should not take more than a few seconds (at most 5)	Partially
Must Have	Traffic data is stored over time and a history of several hours is kept	Yes
Must Have	Application should work smoothly (no jitter) despite complex calculations on large amounts of data	Yes
Must Have	Contour plot must account for distances between measurement points on the road	Yes
Must Have	The history of road maintenance must be displayed on the map	Yes
Should Have	The road network already shows current traffic intensity	Yes
Should Have	Travel time on a route can be calculated based on the contour plot	No
Should Have	Matrix sign history is kept (several hours)	Yes
Should Have	Effects of matrix sign information can be measured	Partially
Should Have	The history of road maintenance must be displayed on the contour plot	No
Should Have	Contour plot of a route can be visualized on different points in time, based on historical data	Yes
Could Have	Visualization of road capacity in a point cloud can be executed based on traffic data	Yes
Could Have	The user can select more than two points for a route	Yes
Could Have	Effects of traffic flow management decisions on traffic flow can be simulated	No
Could Have	Researching the possibility of prediction traffic speed using a LSTM neural network	No

Table 6.1: Overview of requirement satisfaction of the final product

mentation would be complex enough to become its own project. Therefore this feature was scrapped from the project and this allowed the group to devote more time to other more important features.

Another unfulfilled requirement is the requirement that road maintenance data can be displayed on top of the contour plot. Unfortunately, the implementation of the status messages side bar within the application turned out to be more complex than expected. This led us to abandon this feature. We would still encourage future developers to implement this feature as the required data is available.

The final two unfulfilled requirements relate to simulation and prediction of traffic conditions. This was out of scope of our project, but because of personal interest we decided to explore the possibility. Our client advised us to prioritize other features over these features as these were more interesting for the product. Some time was spent on researching the possibility of using a neural network to predict traffic speed on measurement sites and the results looked promising. However, as this was out of scope for the project, no more time was invested in this feature.

6.1.3. Overall Satisfaction

The final product clearly satisfies the list of requirements that was composed at the beginning of the project. The requirements that were partially satisfied or completely unfulfilled were discarded in discussion with the client.

In section 1.1, the problem to be solved was analyzed and a vision for the final product was constructed. We are confident that we have delivered a product that successfully tackles this problem and fits the vision

Query	Average Response Time
Data for Contour Plot (A4 Delft to The Hague, 1h)	8.3s
Matrix Sign States (A4 Delft to The Hague, 1h)	0.35s
Finding a Route Between Two Points (A4 Delft to The Hague)	0.15s
Current Status Messages	0.77s
Live Traffic Data	2.72s
Point Cloud (full day)	11.03s

Table 6.2: Overview of average response times for querying server

that was formulated at the beginning of the project. This is not only reflected in the satisfaction of the requirement list but also the enthusiasm of the client throughout our demo's. Several experts connected to our client have seen demo's of our product and felt comfortable using it. The product therefore fits within the vision that was formulated, that it would allow traffic managers to access traffic data in a intuitive manner and support decision makers in the domain of traffic management.

6.2. Evaluation of Methodology and Process

The project required the group to work completely autonomous and placed all responsibility for delivering a working product upon the group members. Leveraging previous project experience in SCRUM, the group adopted this methodology at the beginning of the project. By having sufficient meetings, as described in section 5.2, group members were able to efficiently allocate tasks and distribute an appropriate workload over all group members such that each group member contributed equally to the final product.

There were little conflicts within the group and the conflicts that arose were mainly revolved around approaches to application design and implementation as described in section 5.3. In all cases, these conflicts were resolved through joined discussions with all group members.

The SCRUM methodology came to good use by using Github issues as a backlog. Having weekly meetings internally with the group and meetings with the client gave us the opportunity to have a continuous feedback loop on our progress and the tasks that each group member was working on. This system also allowed every group member to keep active track of what each other group member was doing, without the need to meet together. Therefore, it was possible for all group members to work on the project from home throughout the week. In practice, however, joined group work sessions were preferred.

Overall, the group functioned efficiently and fairly and the group agrees that the project was successful in terms of methodology and process.

6.2.1. Group Member Responsibilities

Throughout the project it became apparent that group members preferred to specialize in certain areas of the project. While some spent more time working on the frontend and UI, others preferred to spend their time on improving the backend. This led to a distribution of responsibilities, where these group members had special knowledge and kept oversight over certain parts of the project. These responsibilities were distributed as follows:

Stephan Tromer - Frontend, backend, matrix signs, map, UI
Matthijs van Niekerk - Backend, frontend, database, parsing, status messages
Robin Oosterbaan - Backend, route finding, graph model, live traffic, contour plot, point cloud
Jordi Smit - Backend, database, API, matrix signs, contour plot
Daniël van Gelder - Frontend, matrix signs, contour plot

6.2.2. Insights

Even though the group is satisfied with the result of the project, there are some things that we would have done differently throughout the project when looking back at the project.

Route Finding
The measurement sites along a route were originally found using a graph model provided by NDW called VILD. While our client discouraged us to use this approach, we still pursued it, because the NDW documentation recommended this approach and it seemed fairly simple to implement. The alternative approach that was initially recommended to us by the client also seemed to be not as robust. However, due to missing data and questionable design decisions the VILD data is both incomplete and very difficult to work with due to the many instances of missing data and exceptions. This is to be expected because the road network in the Netherlands is also highly complex. While the implementation using VILD worked, it was not perfect and it proved difficult to integrate with the other parts of the code that also needed to be plotted along a trajectory (like the MSI data). Therefore we moved to the method that was originally recommended by the client. While this method performed better, it was still not perfect. To improve the performance, we later reimplemented the code using the graph matching mechanism described in section 3.2.1.

TypeScript
The frontend was originally written in JavaScript and this turned out to cause a substantial amount of maintainability issues. All code became clustered in only a handful of files, with unreadable code. While the complexity of our application design and implementation increased, so did the issues in code maintainability. Therefore we realized that the codebase in the frontend needed to be refactored. A solution for us was to convert the frontend codebase to TypeScript [37]. This is also discussed in section 4.1.2. However, this conversion took significant effort as a lot of functions needed to be rewritten. In hindsight, we therefore conclude that using TypeScript from the beginning on would have been a better decision. The major takeaway from this insight is that large projects require a strictly structured frontend. This needs to be taken into account during the system design phase.

XML parsing
The parsing of XML documents was constantly difficult, since we decided to use a low level SAX parser. When the status messages were implemented in the project, the parsing complexity ballooned out of control. This is because the status message XML structure is highly complex. To solve this issue, we created a parsing library that wraps the low level SAX parser. In our opinion, not many XML parsing libraries that are available for the JVM would have fit our use case. The XML parser needs to store only a small amount of the fields that are present in the original XML file. The parser should preferably also be a streaming API, that streams the XML document once and does not keep the whole file in memory at any point in time. If it was clear from the beginning that parsing would have been such an issue, we would have spent some time creating some reusable parsing code. This would have avoided the need to rewrite all parsing code near the end of the project.

6.3. Evaluation of Ethical Implications
To evaluate the ethical implications of the project, two aspects are considered: the possibility that privacy-sensitive data is stored and the possibility that the functionalities of the product can be used for any unethical purposes.

6.3.1. Use of Privacy-Sensitive Data
Since there exist no user authentication within our application, no user information is stored or accessed. No cookies are stored on the clients machine either. The only data that is stored is extracted from the datastream of NDW. NDW provides an anonymized dataset of cars, which makes it impossible to track a specific person or even a vehicle. The privacy statement of NDW assures that all data is anonymized and no personal identity-related data is stored [39]

6.3.2. Misuse of Product Functionalities
While our product uses publicly available data, the resulting processing and analysis of this data needs to be evaluated on whether it could possibly negatively impact individuals. There are several features in our application which could reveal possible illegal behaviour of individuals. For instance: data derived from the

plots could indicate areas with where speeding occurs frequently. It is highly disputable whether this use of the features is unethical as no privacy-sensitive data is associated with the measurements and no person is directly affected by this use of the product. We personally do not consider this unethical, since no single person is affected (privacy is ensured) and the data used is publicly available.

No significant uses have been identified that could be considered unethical. We took great care into ensuring that our application does not have any negative ethical implications. Since we are exclusively working with publicly available data and our application contains no prediction, classification or decision making mechanisms, the possibilities of unethical use are extremely limited.

7

Conclusion

Scenwise's goal is to make the analysis of traffic data easier for both traffic management experts and non-experts. Up to this point, Scenwise had multiple ideas for tackling this problem but none of their products implemented all their ideas into a single application that worked for the entire Dutch road network. The newly created prototype is able to move Scenwise much closer towards this goal, while being able to handle all the traffic data in the Netherlands. Both the point cloud and the contour plot features make the comprehensibility of the data significantly easier for both experts and non-experts. Especially the flow animation within the contour plot, which is something that has never been done in this domain significantly contributes towards this goal. Besides the visualization tools, this project also produced some other useful tools and domain knowledge that can be used by Scenwise in future projects. Especially the produced graph model (section 3.2) and parser library for the NDW data stream (section 3.3.3) will most likely prove to be quite useful.

This project proofed to be a good learning experience. Not only did this project teach us how to deal with the technical issues related to creating an industry-ready application but it also taught us how to work and communicate both with an external client and within a group. Also learning all the requirement traffic management domain knowledge from our client and our visit to Rijkswaterstaat made this a unique Bachelor End Project.

As indicated in section 6.1 all the critical requirements of the project have been met. Scenwise has already shown the application to some traffic management experts and they all indicate that the application is quite intuitive and user friendly. That is why Scenwise is currently planning to deploy the application as a fully working service after some additional security features have been added. Which indicates that the project was a success.

8

Recommendations

8.1. Improvements

This section discusses some of the smaller changes that some future developer might want to implement to improve the application.

8.1.1. UI Testing

As described in chapter 4.2.1 creating frontend tests was more difficult than expected. To further improve the frontend testing suite, it is recommended to create functional tests using, for example, Selenium [31]. By using Selenium it is possible to create UI tests that make sure the UI works as intended. During the project it was not yet possible to include UI tests, since the UI was frequently changing. The tests would have to be rewritten each time a small change was made in the UI. We decided it was better to focus on implementing features first before creating functional tests.

8.1.2. Point Cloud Speedup

Currently when retrieving the data for the point cloud all the data is retrieved. As a result it takes a while to download all the data to the user's computer. This could be improved by only downloading the data that is required based on the selected window. The disadvantage of this method is that each time the user uses the slider there is a small delay in showing the updated point cloud.

8.1.3. Missing Data Convention

In the final week of the project we learned that it is traffic management convention to show missing data on a contour plot with a gray colour. However the current implementation of the contour plot removes the missing data using a convolutional filter. So in order to adhere to this convention the following changes should be made. Firstly, the convolutional filter should be removed and instead the REST response should indicate which values are missing. Secondly, the contour plot on the front end should draw areas with missing values gray. Finally, the hardest part of this change is to make sure that the arrow flow of the Contour Plot handles these missing values correctly. By convention NDW returns -1 if the measurement is invalid. This is troublesome for the arrow flow since this would result into flows getting stuck in specific areas and leaving other areas completely empty. So a possible solution for this problem must be found before this change can be implemented.

8.1.4. Filtering Faulty Data

In practice, it turned out that there is a substantial amount of measurement sites which constantly broadcast faulty data. While there are mechanisms in place as mentioned previously to handle faulty data, it could be an improvement these faulty measurements before they are stored in the database. Another possible solution for could be to store the names of these faulty sites separately in order to provide feedback to NDW about these faulty measurement sites.

8.2. Extensions

This section discusses some of the larger changes that some future developer might want to implement to improve the application.

8.2.1. Security

Before this product can be taken into production some additional security features must be implemented. At this moment, no security features have been implemented, since the project was originally only meant as a proof of concept. Thus the following security features must be implemented:

Firstly, currently anyone that knows the IP address of the server can access the application. So some method of authentication and authorization must be implemented to ensure only that only authorized people can access the application. It will probably be easiest to implement this feature using Spring Security [33], since the application has been built using Spring Boot [32]. The application only has read REST endpoints. Which means that the authorization model will be relatively simple and can most likely be handled with simple authentication only.

Secondly, the database has been created with the default user settings. So before deploying it to production this must be changed to prevent unauthorized access through the exposed port. Another possible solution is to let Docker-compose [24] not expose the port. The application will still have access to the database internally while the database will not be exposed to the outside world.

8.2.2. Database Backups

Currently the database runs within a Docker container [11] on the same server as the application. A disadvantage of this approach is that it is rather difficult to create a backup of the database, which might be desirable in a production environment. If this is the case, further research should be performed into possible solutions to this problem. One possible but pricey solution might be to migrate to a database service such as AWS [2] or Azure [19].

8.2.3. Traffic Speed Prediction

The literature contains multiple papers that discuss how Deep Learning can be used to predict sequences of future speed and flow measurements based on past observations [18, 17]. The literature proposes multiple approaches to create these types predictions. One of the suggested approaches in the literature is to use a Long Short-Term Neural Network, to capture nonlinear traffic dynamic from remote microwave sensors to predict short-term traffic speed [18]. This approach is very interesting since all the visualizations in this project are all based on the same type of data from remote microwave sensors. These predictions can be used to extend the visualization with data from the predicted situation. Another approach suggested in the literature is to use a Deep Convolutional Neural Network that predicts segment-wide traffic speed [17]. The interesting part of this approach is that it uses a contour plot in image format as input for its prediction of the future state of the traffic flow.

During the project we were able to successfully train such a Neural Network using an Encoder-Decoder LSTM architecture [15] on a small sub set from the measurement sites in the data set. However due to time constraints, this area of research was not pursued further since it wasn't a requirement of the client.

A future researcher might want to research if such a network could be trained on a much larger data set containing all the measurement site in the Netherlands. Such a network could than be used to extend the contour and point cloud plots with additional data points. This network could also be used to predict the future effects of car accidents that just happened or to predict the formation of congestion in general.

8.2.4. Rendering MSI Data For a Region

One possible extension of the project brought up by the client was to have the functionality that the user can select a region for which all MSI data along every road within the region is rendered. This feature was discussed to be implemented with the client, however due to time constraints was unable to be implemented before the project deadline. Two main challenges exist in order to implement this feature: rendering the vast amount of data while preserving performance and efficiently retrieving all data in a short amount of time.

Because the three zoom levels are currently rendered simultaneously in the current implementation for plotting MSI data, this would dramatically decrease performance on a regional level. An option could be to render only one zoom level, as this feature would only be used for monitoring on a regional level so deeper zoom levels are not required any longer. If this still has a big performance impact, more optimized rendering

mechanisms and libraries could be considered apart from Mapbox, for example: DeckGL [16]. Functions exist to render the zoom levels separately within the code already so with little adaptation the retrieved MSI data could be plotted using the existing codebase.

This brings us to the next problem which is retrieving the large amount of data efficiently. While MSI data in general is lightweight, retrieving it on a regional scale can be cumbersome. This is due to that on a regional scale, all MSI data within bounds of four coordinates (the window) need to be retrieved, as opposed to the current implementation which retrieves MSI data along a route. Retrieving data based on geolocations has not been researched within the current project so it is hard to estimate whether this is computationally feasible. However, databases do offer geolocation features which allow queries to run more optimally using locations.

Bibliography

[1] Vladimir Agafonkin. *How I built a wind map with WebGL*. July 2017. URL: https://blog.mapbox.com/how-i-built-a-wind-map-with-webgl-b63022b5537f (visited on 06/15/2019).

[2] *Amazon Web Services (AWS) - Cloud Computing Services*. 2019. URL: https://aws.amazon.com/ (visited on 06/20/2019).

[3] Cameron Beccario. *a global map of wind, weather, and ocean conditions*. URL: https://earth.nullschool.net/ (visited on 06/18/2019).

[4] *Blade*. 2019. URL: https://github.com/lets-blade/blade (visited on 06/19/2019).

[5] *Bootstrap*. 2019. URL: https://getbootstrap.com/ (visited on 06/19/2019).

[6] Hampton Catlin et al. *Syntactically Awesome Style Sheets*. 2019. URL: https://sass-lang.com/ (visited on 06/19/2019).

[7] Chartjs. *Chart.js*. URL: https://www.chartjs.org/docs/latest/ (visited on 06/18/2019).

[8] *Checkstyle 8.21*. URL: http://checkstyle.sourceforge.net/ (visited on 06/19/2019).

[9] Wikipedia contributors. *"Map matching — Wikipedia, The Free Encyclopedia"*. 2019. URL: https://en.wikipedia.org/w/index.php?title=Map_matching&oldid=898376941 (visited on 06/17/2019).

[10] *Elastic*. 2019. URL: https://www.elastic.co/ (visited on 06/19/2019).

[11] *Enterprise Container Platform*. 2019. URL: https://www.docker.com/ (visited on 06/20/2019).

[12] *Findbugs*. 2019. URL: http://findbugs.sourceforge.net/ (visited on 06/20/2019).

[13] *Google Maps*. URL: https://maps.google.com/ (visited on 06/21/2019).

[14] *Grails*. 2019. URL: https://grails.org/ (visited on 06/19/2019).

[15] *How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption*. 2019. URL: https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/ (visited on 06/18/2019).

[16] *Large-scale WebGL-powered Data Visualization*. URL: https://deck.gl/#/ (visited on 06/19/2019).

[17] Xiaolei Ma et al. "Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction". In: *Sensors* 17.4 (2017), p. 818.

[18] Xiaolei Ma et al. "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data". In: *Transportation Research Part C: Emerging Technologies* 54 (2015), pp. 187–197.

[19] *Microsoft Azure Cloud Computing Platform and Services*. 2019. URL: https://azure.microsoft.com/ (visited on 06/17/2019).

[20] Mocha. *Mocha*. 2019. URL: https://mochajs.org/ (visited on 06/18/2019).

[21] TS-Mocha. *TS-Mocha*. 2019. URL: https://www.npmjs.com/package/ts-mocha (visited on 06/18/2019).

[22] *MongoDb*. 2019. URL: https://www.mongodb.com/ (visited on 06/19/2019).

[23] *NDW*. 2019. URL: http://opendata.ndw.nu/ (visited on 06/19/2019).

[24] *Overview of Docker Compose*. 2019. URL: https://docs.docker.com/compose/overview/ (visited on 06/20/2019).

[25] *Pluggable JavaScript linter*. URL: https://eslint.org/ (visited on 06/21/2019).

[26] *PMD*. 2019. URL: https://pmd.github.io/ (visited on 06/18/2019).

[27] *PostgreSQL*. 2019. URL: https://www.postgresql.org/ (visited on 06/21/2019).

[28] *Project Lombok*. 2019. URL: https://projectlombok.org/ (visited on 06/19/2019).

[29] *React JS*. 2019. URL: https://reactjs.org/ (visited on 06/20/2019).

[30] *SAX parsing in Java*. URL: https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html (visited on 06/21/2019).

[31] Selenium. *Selenium*. 2019. URL: https://www.seleniumhq.org/ (visited on 06/18/2019).

[32] *Spring Projects*. 2019. URL: https://spring.io/projects/spring-boot (visited on 06/20/2019).

[33] *Spring Projects*. 2019. URL: https://spring.io/projects/spring-security (visited on 06/20/2019).

[34] *SQLite*. URL: https://www.sqlite.org/index.html (visited on 06/21/2019).

[35] *The new major version of the programmer-friendly testing framework for Java*. 2019. URL: https://junit.org/junit5/ (visited on 06/19/2019).

[36] *TRAVIS CI*. 2019. URL: https://travis-ci.org/ (visited on 06/20/2019).

[37] *TypeScript*. 2019. URL: https://www.typescriptlang.org/ (visited on 06/18/2019).

[38] Chris Veness. *Movable Type Scripts*. URL: http://www.movable-type.co.uk/scripts/latlong-vincenty.html#direct (visited on 06/21/2019).

[39] L. Verbeek. *Privacy Statement NDW Databank*. 2018. URL: http://www.ndw.nu/downloaddocument/834642063e0ab6c3bc55ffed54e494c4/Privacyreglement%5C%20NDW.pdf (visited on 06/18/2019).

[40] *webpack*. 2019. URL: https://webpack.js.org/ (visited on 06/19/2019).

[41] Wikipedia contributors. *Shapefile — Wikipedia, The Free Encyclopedia*. 2019. URL: https://en.wikipedia.org/w/index.php?title=Shapefile&oldid=901339602 (visited on 06/17/2019).

[42] Evan You. *Vue JS*. 2019. URL: https://vuejs.org/ (visited on 06/20/2019).

A

Research Report

A.1. Problem Analysis

A.1.1. Problem Definition

Decisions in traffic management are made based on analysis of traffic data. Due to the lack of appropriate tools and the complexity of the tools that exist, it is difficult for people who are not experts in traffic management to interpret traffic data. While traffic data experts are able to judge traffic situations based on the current modelling tools, there is a knowledge gap with traffic managers/policy makers. This introduces a challenge for decision makers to understand traffic situations. Although open data exists, no clear tool exists which makes the data interpretable to novices. A tool needs to be developed that can provide accurate traffic data between any two points on the Dutch road system in a clear and simple manner.

A.1.2. Product Vision

The goal is to develop a web application which shows a map of the Dutch road system. The user can then pick a starting point and an end point, a route between the two points is calculated and a 'contour plot' of traffic between the two points will be shown. This 'contour plot' shows how traffic evolves over time along a route, dynamic vectors will illustrate possible congestion. The main focus of the user interface will be user friendliness. The possibility of traffic data prediction can be explored, where the application can predict future traffic conditions (average speed, traffic density, etc).

A.1.3. Target Customers

The product is expected to have a broad potential target audience, but will be specifically targeting decision makers for traffic management who are not experts in traffic data analysis.

A.2. Requirement Analysis

Requirements of the product will be analysed based on the MoSCoW method, where requirements are divided up into four categories: Must Haves (absolute necessary requirements), Should Haves (preferred requirements), Could Haves (possible extensions), Won't Haves (possible future work, not part of this project).

Must Haves

- The application shows an overview of the Dutch road network
- The user can select two points to indicate a route
- A (dynamic) contour plot of any route can be generated
- The application updates traffic data in real time
- Matrix sign information along a route can be retrieved for a given route
- Queries should not take more than a few seconds (at most 5)

- Traffic data is stored over time and a history of several hours is kept
- Application should work smoothly (no jitter) despite complex calculations on large amounts of data
- Contour plot must account for distances between measurement points on the road
- The history of road maintenance must be displayed on the map

Should Haves

- The road network already shows current traffic intensity
- Travel time on a route can be calculated based on the contour plot
- Matrix sign history is kept (several hours)
- Effects of matrix sign information can be measured
- The history of road maintenance must be displayed on the contour plot
- Contour plot of a route can be visualized on different points in time, based on historical data

Could Haves

- Visualization of road capacity in a point cloud can be executed based on traffic data
- The user can select more than two points for a route
- Effects of traffic flow management decisions on traffic flow can be simulated
- Researching the possibility of prediction traffic speed using a LSTM neural network

Won't Haves

- An optimal alternative route to a given route can be generated based on traffic data
- The application takes active traffic management decisions: like managing matrix sign information.

A.3. Project Planning

Week	Date	Tasks	Notes
4.1	22/04 - 28/04	Orientation on technologies, product definition, analyzing possibilities for extension	
4.2	29/04 - 05/05	Setup of project backend and front end, defining API structure, writing research report	Deliverable: Research report
4.3	06/05 - 12/05	Project backend/frontend	
4.4	13/05 - 19/05	Construct RESTful API connecting layers	
4.5	20/05 - 26/05	Preparation Midterm, incorporate MSI data and road maintenance data	
4.6	27/05 - 02/06	Application will keep history of data, and history can be shown. Midterm presentation for client and supervisor, SIG code quality	
4.7	03/06 - 09/06	Research point cloud for traffic data and incorporate in application	SIG Code Quality feedback
4.8	10/06 - 16/06	Point clouds can be used to infer possible congestion later in the day	
4.9	17/06 - 23/06	Research LSTM neural network predictor for traffic speed and possibly implement it, prepare report	Second deadline SIG Code Quality
4.10	24/06 - 30/06	Finish report + info sheet and prepare final presentation	Report Deadline: 25 June (Tuesday)
4.11	01/07 - 07/07	Finish final presentation	Final Presentation: any day of this week

A.4. Methodology

The project will use a SCRUM based approach. After every meeting with the client, requirements are set up with an estimation of its workload. From that requirement list, requirements will be selected to be worked on for the coming week. At the end of the week, the group's progress is discussed with the client and new requirements are formulated. This approach ensures flexibility of the group towards the client such that the project requirements can change throughout the project based on the group's progress. Each team member is responsible for his own assigned requirements which will be in the form of Github issues. Throughout any week, issues can be created by any group member. At the moment where requirements and issues are assigned, these new issues will be considered as well. These are typically bugs that arose throughout the week or possible new features of the project. The group is required to meet at least twice per week, once on Monday and once on Friday together with the client. Each group member is free to choose where he wants to work throughout the rest of the week as long as his responsible issues are completed at the end of the week. If this is not the case, measures can be taken by the other group members to ensure that this will be the case in the future. Should any conflicts arise within the group which can't be resolved within the group, the group will contact the TU Delft coach in order to settle any disputes. The group intents to set up a meeting with the TU Delft coach once every two weeks to discuss the group's progress and any problems/challenges that the group encountered. However, if it is not deemed necessary to meet the meetings can be canceled to prevent the coach from attending useless meetings.

A.5. Technology Choice

In this section, we will go into the technology choice and alternatives that we considered regarding some of the most crucial parts of the application.

A.5.1. Frontend Framework

During the system design phase, several frameworks were considered to be used in the frontend like React [29] and Vue [42]. While these frameworks offer very convenient mechanisms for building a web application, there is a lot of setup involved. We realized that our product would need to visualize data but would not need to do complex computing or execute any other advanced frontend behaviour. We concluded therefore that no frontend framework was needed, as the setup required would not benefit the development of the eventual product. In addition, our product only displays a single page, whereas typical web application have multiple pages to display. It is in these types of scenarios where frameworks provide the best benefits, but this is not applicable to our case.

A.5.2. Backend Framework

During the planning phase, we decided to use Java as the backend programming language, since most of the developers already have a lot of experience with this programming language. This only left us with the choice of the backend framework by choosing a framework such as Spring Boot [32], Blade [4], Grails [14], etc. Not using a web framework wasn't an option since this would mean that we would have to waste a lot of time on reinventing the wheel. Our main requirements for this framework were that it should be easy to use, have a minimum amount of boiler plate and most importantly some of the developers already should have some experience with this framework. Eventually we decided to choose the Spring Boot Framework [32] due to the following reasons. Firstly, One of the developers already had some experience with this framework. Secondly, this framework removes quite a lot of boiler plate with their annotation based approach. Finally, the original project description stated that there was already some legacy code written with this framework.

A.5.3. Parsing

Since all the data is provided by XML files, the choice of XML parser is also important. The XML parser had to satisfy a couple of requirements. Every minute, multiple XML documents that are about 100 MB in size need to be parsed. This requires a decently fast XML parser that preferably does not keep the whole XML file in memory at once. One the other hand, the complexity of the files that need to be parsed is quite high. Most of the XML file is irrelevant, only some fields need to be converted to objects in the application and later stored in the persistence layer.

The two main choices that seem to be available for Java are SAX parsers [30] and DOM parsers. SAX parsers use a streaming API that is push based. That is, the parser controls the application thread and the parser handler can only accept invocations from the actual parser. The parsing is also generally quite fast. This type of API does require some kind of state machine and switch statements that tend to get very complex when the XML doucment gets more complex. DOM parsers on the other hand keep the whole document in memory as an object tree at once. This enables flexibility but requires a very large memory footprint and a has a fairly slow parsing speed. Because we were not confident that the XML doucments could be comfortably be held in memory, we decided to opt for the SAX parser.

Later on it turned out that the complexity of the SAX parsing code was too high. A parsing library that wraps the SAX parser was written to combat this. This custom parser gave us both speed and maintainability at the same time.

A.5.4. Database

The persistence layer of our application had to satisfy a couple of requirements. The main data source, the point measurement data, has a write load of about 16000 new records per minute. This data also needs to be queried every time the application is loaded (for the live traffic intensity) and every time a contourplot is generated. The live traffic intensity needs to retrieve all data from a certain timestamp (the last measurement that is complete) while the contourplot needs to retrieve the data from the last couple of hours for a given number of measurement points. This requires indexes (or something similar) on at least the time and location ID to enable fast retrieval of this data. At most, about 600 million measurement records will be stored. This size is achieved after a month of data collection. Any older records are removed from the database. The database that we decide to use needs to be able to insert a decent amount of records, search through a very large number of records efficiently based on two different attributes at the same time and be able to store half a billion records as efficiently as possible.

Besides the high intensity part of the database, there is some other data that needs to be stored. Status Messages are reasonably complex, since they can be one of 4 different (main) types that all have their specific attributes. For this, some kind of normalizing capabilities or even inheritance would be beneficial.

Other requirements that played a role are familiarity and integration with backend language and framework, java and Spring. The databases that we were familiar with were SQLite [34], PostgreSQL [27], ElasticSearch [10] and MongoDB [22]. ElasticSearch was discarded as a possibility since the unique advantage that it brings, fast and extensive full text search, could not be utilized in this application. SQLite was also discarded since it would have an advantage if the database needed to be embedded or stored on the end users computer. Since both of those were not the case PostgreSQL seemed the superior choice. The choice between MongoDB and PostgreSQL was relatively arbitrary. MongoDB would have no impedance mismatch with Java, this would probably make development easier. PostgreSQL on the other hand would add more satisfy to the persistence layer, due to the more advanced type system and the more sane default settings. PostgreSQL could also easily integrate with Spring, using Hibernate. Finally, PostgreSQL allows for much more complicated queries. This would give more flexibility if the project requirements change after the data model was decided. This is a weakness of MongoDB, since there is often only one efficient way to query the data. Because of these advantages, PostgreSQL was the database technology that we chose in the end.

A.6. Data Processing

To get a minimal viable product (MVP) working, the frontend needs to be able to show two things: A contour plot of business on a given route over time and a map of the road network. This section will consider the data sources that are used for the contour plot. Besides the functionality required for the minimal viable product, there are more data sources that we would like to integrate for the final product. These data sources include a history of road maintenance and an overview of matrix signs displays. This section will also touch on the data sources required for this extra functionality.

A.6.1. Data Sources We Use for the Contour Plot

The contour plot needs to be able to show the average speed of cars at a certain point on the road over time. Speed data for every major road is determined by point measurements (a "lus"). Most "lus" measurement points measure the average speed of cars and the amount of cars over some interval, which in practice is always a minute. This measurement data is provided as an XML file every minute by NDW (Nationale Databank Wegverkeersgegevens) [23]. Besides data for the "lus" measurement point, data is also collected by bluetooth, laser, radar, license plate detection and multiple other data sources. Because of the varied data sources a separate configuration XML file is provided which is updated once per day, at 13:05 local (Amsterdam) time. The configuration file specifies for each measurement what the source of the measurement is, how the data is aggregated and where the data is collected. All "lus" measurement points collect data per lane of the road and some separate data on the vehicle class, where each vehicle class is in a certain size range. Since not all measurement points separate on vehicle class, we only collect aggregate data. For the MVP it is also not necessary to separate data per lane, so the speed of all lanes is averaged before it is sent to the frontend. However, to enable future extension the speed data is stored by lane. Besides the speed data, the throughput ("flow") through a "lus" is also stored. We expect that this data may also be visualized in a contour plot in a way similar to the speed data.

A.6.2. Finding the Position of the Point Measurements

One difficulty present in the data is finding the position of the point measurements. There are two approaches that work, using coordinates or using VILD (Verkeersinformatie Locatie Database). In our approach we use a synthesis of both. Besides the configuration file that is updated daily there exist two more data sources. One is a collection of "shapefiles" that is updated every couple of months. A shapefile is a file format that is used for storing and displaying coordinate data. Most point measurement locations have a coordinate linked to it. This can be used for an approximation of the location of a measurement.

However, these coordinates alone are not enough to reconstruct the ordering of the point measurements along a road. For this we use VILD. VILD is a database by Rijkswaterstaat with point locations (like intersections), line locations (like roads), and shape locations (like provinces). VILD point locations can be linked to each other in a graph, spanning all the major roads in the Netherlands. Each point measurement is linked to a VILD point, in a many to one relationship. The offset of each point measurement to the VILD point is also known as well as the side of the road the measurement is taken on. By combining these data points, it becomes possible to reconstruct the ordering of the point measurements along the road. The VILD data-set is also a cheap way of generating routes. Since the point form a graph, the shortest route of any point to any other point can be calculated using Dijkstra's shortest path algorithm.

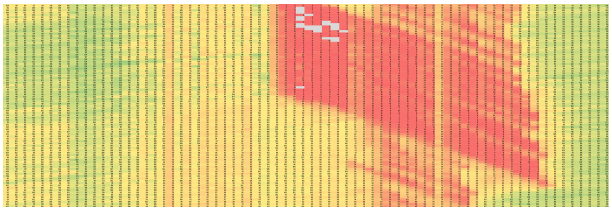


Figure A.1: Old time velocity plot

A.6.3. Processing and Storing the Data

Since we are only interested in "lus" data, we can immediately ignore a large part of the data. This is good because the configuration file is 218MB in size when uncompressed. The measurement data that is provided every minute is also quite big at about 42MB uncompressed. Finally, the data structure used in the configuration and measurement files is very unwieldy to use, since it provides for a much more general case than we are interested in. The combination of those realizations implies that the parser that reads those files needs to be able to handle large files efficiently, not store unnecessary data and provide the output in a nicer format. Because of these requirements we chose a SAX parser, which is a parser conforming to a standard where each piece of XML is parsed sequentially, instead of a parser that operates on the document on a whole (a DOM parser). A DOM parser would in our case have been much slower, and would likely have led to memory issues. The total overhead of parsing now amount to about 600ms per minute and 2000ms on start-up, which is well within our performance budget.

After processing the measurement data, it is aggregated in a database. For every measurement, the location, time and lane is stored. This means that the data contains many fields that are duplicated many times in the database. Since high read and write throughput is required, the data can not be normalized too much. At the current data model, 8MB per minute of new data is stored in the database. Some normalization may be required to keep the size of the database to an acceptable state.

A.7. Data Analysis

This section discusses the different types of data analysis and visualization techniques that have been researched.

A.7.1. Road display

A map is a vital component of our application. There are several options for different (web) mapping services. The maps that were considered are Google Maps [13] and [redacted]. The choice basically came down to proprietary or open source software. For both Google Maps and [redacted] a small demo was created to investigate the pros and cons. Developing using Google Maps proved to be cumbersome, since we could only make 1 request per day to their API. This limited developing significantly and this basically forced us in using [redacted]. [redacted] only provides map data but no API. Fortunately [redacted] provides several similar API methods to the Google Maps APIs that work with [redacted].

It would also be possible to self host the map APIs. However at the moment this is not crucial to implement for the project. This could be a complete project on its own and is out of scope for this project.

A.7.2. Traffic Flow Contour Plot

Normally traffic is visualized using a static figure of average car velocity opposed to time passed at each measurement point. These graphs are sub optimal for understanding the flow of traffic as shown in figure A.1. The customer needed a way to dynamically visualize traffic whilst maintaining understandability. The proposal is to use an animated vector representation as commonly used in wind visualizations [3]. The graph will represent the measurement points over a specified road segment and visualize the flow of traffic on the segment using particles. The particles will slow down when traffic is slow, but still move upwards in time. Following the particles now allows insight in traffic movement at every point in time.

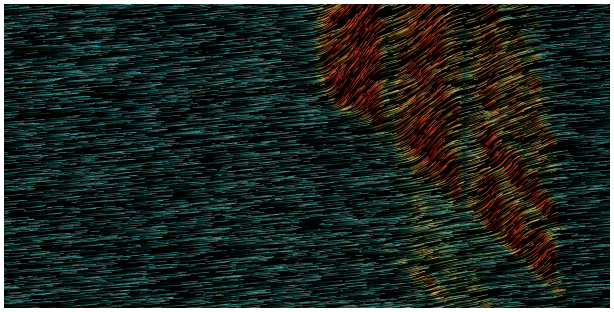


Figure A.2: New animated time velocity plot

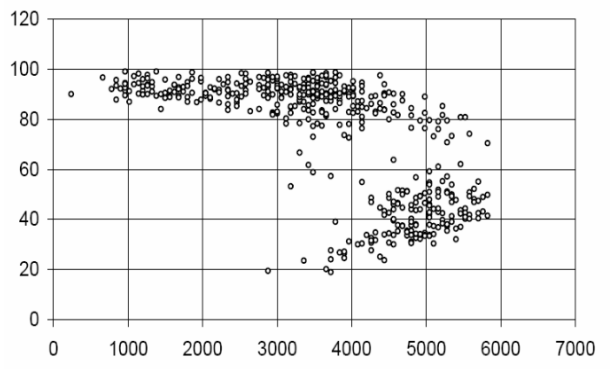


Figure A.3: Density velocity plot

A.7.3. Road Work & Incidents

Other influences for traffic flow are road limitations. In order to understand traffic behaviour, it is important to know whether a congestion is caused by an overflow of traffic, or an external factor. These factors can be categorized under temporary road limitations and must also be provided to the user when analyzing the traffic. The data provided by NDW contains information of active incidents and planned road works, which will be made visible to the user when necessary.

A.7.4. Matrix Signs

The response of traffic velocity to road signals, in general tends towards the desired velocity. Although not as influential as active incidents, this data is still necessary to provide to the user. An important note is that there is a distinction between automatic and manual signaling. The automatic signals are designed to slow down traffic in order to guarantee safety during a congestion. The manual signals tend to be a response to active incidents or road works. The data provided does not make the distinction between automatic and manual control. However, the data is still desired when analyzing traffic and shall be incorporated in the visualizations.

A.7.5. Road Capacity and Recovery

Another important metric for traffic analysis is the velocity compared to the density of cars. In general a road will maintain the average velocity up until a specific point of density, after this critical point is reached the velocity degrades substantially, also called an overflow. The recovery of the throughput after an overflow is not in line with the reduction of density to a previous safe amount. The road now requires a massive reduction in density in order to climb back to the original velocity. The traffic measurement information is necessary for approximating the critical point of overflow. When this value is known it is also possible to use it for expressing actual road fullness in respect to the point of failure. The first step will be visualizing overflows over the course of a full day, the next step will be approximating the point of failure. An example plot of this traffic phenomenon is shown in A.3. The visualization will allow scrubbing through the history of a road segment in order to compare how road conditions influence the maximum throughput.

A.7.6. Predicting Speed

An additional extension of the project is the prediction of a sequence of future speed and capacity usage of specific road segments. The literature proposes multiple approaches to create these types predictions. One of the suggested approaches in the literature is to use a Long Short-Term Neural Network, to capture nonlinear traffic dynamic from remote microwave sensors to predict short-term traffic speed [18]. This approach is very interesting since all the visualizations in this project are all based on the same type of data from remote microwave sensors. These predictions can be used to extend the visualization with data from the predicted situation. An other approach suggested in the literature is to use a Deep Convolutional Neural Network predicts segment-wide traffic speed [17]. The interesting part of this approach is that it uses a contour plot in image format as input for its prediction of the future state of the traffic flow. Thus this approach can be used to extend the contour plot visualization with predictions of the future.

A.8. Architecture

This section discusses the different architecture choices that have been researched and the resulting choices that have been made based on this research.

A.8.1. Database

The data persistence layer has several requirements. Some data needs to be stored once and read many times. Two examples of this type of data is the VILD database and the point measurement configuration files. The VILD database is updated every year, and the point measurement configuration every day. So the VILD database can be seen as a completely static resource that is just included in the project. This data is "lightly" relational, in the sense that it is normally stored in a relational database. But the database consists of only one table so the only relations are the graph connecting VILD points to other VILD points, which means that the only "foreign" keys are keys pointing to other rows of the same table. Since relational databases are not particularly good at recursive queries we just pull all the data from the database when needed. So the storage medium of this data does not matter a great deal.

The point measurements data and point measurement configuration data is updated more frequently. Since the point measurement configuration data only makes sense within the context of parsing data, this data is not stored in the database. The measurement data needs to be queried for a certain time interval and for a set of positions. For this to be efficient, multiple indexes need to be kept. The point measurement data is updated every minute and at least a week of data needs to be stored. At the moment no columns are normalized and 8MB of data is added every minute. In a week this adds up to 80GB. While this is manageable, it would be better if the database is smaller. Possible options are compression, and (partial) normalization. Just storing the strings of each data column (like the "lane") as id's instead of duplicating the string every time could save a considerable amount of space without effecting performance.

Since by far the biggest part of the database is just a single table of point measurements over time, the complexity of data that the database must be able to handle is not very high. We also do not require very high write capacity, as we only need to be able to handle 8MB per minute. We do require retrieving data based on multiple requirements at once (over a time interval and over several points in space). However, almost any SQL or NoSQL database could handle these requirements. For the moment, we have chosen to store our data in PostgreSQL, but since we make use of an ORM, we could change the database with any other without much difficulty.

A.8.2. Frontend

As the frontend of the application is mainly concerned with User Interaction, design is mainly concerned with aesthetics and usability. We decided, with respect to computational load, to outsource all heavy computations to the backend and that the frontend is mainly concerned with rendering the received data properly in a user-friendly manner. Even though computationally load is moved to the backend, the frontend still is a computationally heavy component due the the rendering of the roadmap and the contourplot. Therefore it is required that the frontend renders data efficiently in order to satisfy the fluency requirement of the UI.

The frontend is not based on any framework as the application has a relatively small UI, thus we rely on native JavaScript, HTML and CSS. However, Node.js is used to communicate with the backend through HTTP requests. In addition, the roadmap is retrieved using the MapBox library and its API. In order to render the contourplot, WebGL is used. For aesthetic design the Bootstrap framework is used, as it has a familiar design for many users and is a ready-made framework with components which can be used 'straight out of the box'.

A.8.3. Backend

The main responsibility of the backend is to implement the data processing pipeline and to allow the frontend to access the data that has been processed. It would have been possible to put these separate responsibilities on different servers, however because the data processing pipeline is relatively simple we decided to combine them into a single backend server. We decided to use Java and the Spring framework to implement backend server due to 3 important reasons. Firstly, the original project description stated that there was already some legacy code written in Java using the Spring framework. So writing the backend server in Java might make it possible to reuse some legacy code. Secondly, everyone in the group is already very familiar with the Java programming language.Thirdly, we chose to use the Spring framework because this takes care of most the standard HTTP server and Database code. Making prototyping much faster since it requires much less code.

A.8.4. Communication between Frontend and Backend

Communication between the frontend and backend components of the application will be done through a RESTful API, this ensures true separation of these components. Through HTTP requests information will be retrieved from the backend in order to be displayed on the frontend. In order to keep the computational load of the frontend low, most computation will be performed on the backend such that the frontend is only concerned with rendering the retrieved data properly.

For example: when a contourplot for a given route needs to be displayed, the frontend sends a GET request to the backend and gives the coordinates of the start- and end-point of the route as parameters. The backend will then be concerned with calculating with data points need to be retrieved and constructs the values for this contourplot. It will then return only the data required to plot the contourplot which the frontend will render in turn.

Since information will be updated periodically, the possibility of using websockets is being considered. This allows the application to receive event-driven responses without the need to poll the server. However, it is given that the required data updates on a known interval thus periodically polling the server could be a similarly effective solution.

B

Mid Term SIG Report

Beste,

Hierbij ontvang je onze evaluatie van de door jou opgestuurde code. De evaluatie bevat een aantal aanbevelingen die meegenomen kunnen worden tijdens het vervolg van het project. Bij de evaluatie van de tweede upload kijken we in hoeverre de onderhoudbaarheid is verbeterd, en of de feedback is geaddresseerd. Deze evaluatie heeft als doel om studenten bewuster te maken van de onderhoudbaarheid van hun code, en dient niet gebruikt te worden voor andere doeleinden.
Let tijdens het bekijken van de feedback op het volgende:

- Het is belangrijk om de feedback in de context van de huidige onderhoudbaarheid te zien. Als een project al relatief hoog scoort zijn de genoemde punten niet 'fout', maar aankopingspunten om een nog hogere score te behalen. In veel gevallen zullen dit marginale verbeteringen zijn, grote verbeteringen zijn immers moeilijk te behalen als de code al goed onderhoudbaar is.

- Voor de herkenning van testcode maken we gebruik van geautomatiseerde detectie. Dit werkt voor de gangbare technologieën en frameworks, maar het zou kunnen dat we jullie testcode hebben gemist. Laat het in dat geval vooral weten, maar we vragen hier ook om begrip dat het voor ons niet te doen is om voor elk groepje handmatig te kijken of er nog ergens testcode zit.

- Hetzelfde geldt voor libraries: als er voldaan wordt aan gangbare conventies worden die automatisch niet meegenomen tijdens de analyse, maar ook hier is het mogelijk dat we iets gemist hebben.

Mochten er nog vragen of opmerkingen zijn dan horen we dat graag.

Met vriendelijke groet, Dennis Bijlsma
[Feedback]

De code van het systeem scoort 3.1 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Interfacing en Unit Size vanwege de lagere deelscores als mogelijke verbeterpunten.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Dit kan worden opgelost door parameter-objecten te introduceren, waarbij een aantal logischerwijs bij elkaar horende parameters in een nieuw object wordt ondergebracht. Dit geldt ook voor constructors met een groot aantal parameters, dit kan een reden zijn om de datastructuur op te splitsen in een aantal datastructuren. Als een constructor bijvoorbeeld acht parameters heeft die logischerwijs in twee groepen van vier parameters bestaan, is het logisch om twee nieuwe objecten te introduceren.

Voorbeelden in jullie project:

- $MeasurementLocations.insertTwoWayMeasurementLocation(Integer,int,int,String,double)$
- $contourplot-gl.js:VectorVisualiser.reshapeArray$
- $contourplot-gl.js:VectorVisualiser.reshapeArray$

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:

- $MatrixParserHandler.startElement(String,String,String,Attributes)$
- $StatusMessagesParserHandler.startElement(String,String,String,Attributes)$
- $MeasurementReader.read(File,File)$

Als laatste nog de opmerking dat er nauwelijks (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen. Op lange termijn maakt de aanwezigheid van unit tests je code ook flexibeler, omdat aanpassingen kunnen worden doorgevoerd zonder de stabiliteit in gevaar te brengen.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

C

End Term SIG Report

Beste,

Hierbij ontvang je de resultaten van de hermeting van de door jou opgestuurde code. In de hermeting hebben we met name gekeken naar of/hoe de aanbevelingen van de vorige evaluatie zijn geimplementeerd. Ook deze hermeting heeft het doel om studenten bewuster te maken van de onderhoudbaarheid van hun code en dient niet gebruikt te worden voor andere doeleinden.

Mochten er nog vragen of opmerkingen zijn dan hoor ik het graag.

Met vriendelijke groet, Dennis Bijlsma

[Feedback] In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen. We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien. Op allebei de gebieden zien we een duidelijke en structurele verbetering, ook in de nieuwe code. Ook is het goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven. Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

D

Project Description in BEPSys

D.1. Company Background

Scenwise B.V. is a company with a lots of experience in Data Science and smart mobility. We work together with our partners to develop innovative software for:

- traffic management (e.g. automatic incident detection, response plans, etc.)
- data science & visualisation. (e.g. traffic monitoring, data fusion, Big Data)

Our customers are the Dutch Highway Agency (Rijkswaterstaat), Nationale Databank Wegverkeersgegevens (NDW), provinces, large cities, ITS system suppliers, Feyenoord stadium and recently also the city of Edmonton in Canada.

D.2. Project description

For different projects, we have developed separate tools to process and analyse different types of traffic data, for visualisation, and for generating new smart mobility information. Most of the tools are currently used by trained internal personals. Some tools are depolyed to our customers but need enhancement. Scenwise has increasing number of data science projects, for example: developing innovative Data Fusion, Big Data solutions for NDW, software for incidents detection for Rijkswaterstaat and datafusion module for the city of Amsterdam.

For these reasons, we intend to develop a new platform which will integrate and replace the current tools and at the time, make it possible for us to incorporate more new data sources and speed up our development of new functionalities. Within this project, a team of students will work together to conduct research, make design decisions, develop the application inclusive testing. The backend should be a new big data platform using different data sources. The front-end should be web-based using modern graphical interfaces which different real-time graphics to analyse the traffic conditions. This is an innovative project with a lots of technical challenges.

We are looking for enthousiastic students with skills and interest in database techniques, data processing, visualization techniques and algoritmes who are willing to take the challenge of developing a new platform to meet the upcoming requirements of our customers.

D.3. Techniques

Scenwise will provide detailed information for making design decisions and guidance in making both the technical designs and the graphical interfaces designs. Setting up test scenario's for software acceptance is also a part of the project. Since this is a new development, there are enough rooms for the project team to make design decisions. If the project team is familiar with Java Spring (back-end) or Angular 6 (front-end), then they may re-use part of the current source code.

D.4. Other information

The students will receive internship compensation and travel allowance conform market prices.

E

Info sheet

Title of the project: Developing a Platform for Traffic Data Analysis
Name of the client organization: Scenwise
Date of the final presentation: June 27th at 14:00

Project description: Scenwise has developed several tools to process and analyse traffic data. However most of the tools are outdated, not as intuitive or not complete enough. This project focused on combining several data sources including matrix signs, traffic flow and incidents on the road and present the data in a intuitive manner, so that not only traffic experts can interpret the data and plots.

Challenge: The challenging part of the project was creating a graph model which was needed to map the road measurement points to a road as well as finding out what the best method is to visualize traffic data.

Research: The research focused on how to convey technical traffic information to people that are not traffic experts, but do have to make decisions about traffic management.

Process: During the project the SCRUM methodology was used. For each week a goal was set for the team. The work was divided based on the respective expertise of each team member.

Product: The product created is a map where users can create routes and based on this generate traffic visualizations. The map also shows accidents, maintenance as well as matrix signs.

Outlook: The client is satisfied with the delivered product and has the intention to further develop the product.

Team roles

Stephan Tromer

- Interests: Programming, web development and entrepreneurship
- Responsibilities: Frontend, backend, matrix signs, map, UI

Matthijs van Niekerk

- Interests: Programming languages, databases and NLP
- Responsibilities: Backend, frontend, database, parsing, status messages

Robin Oosterbaan

- Interests: Machine learning, hardware and graphics
- Responsibilities: Backend, route finding, graph model, live traffic, contour plot, point cloud

Jordi Smit

- Interests: Machine learning and web development
- Responsibilities: Backend, database, API, matrix signs, contour plot

Daniël van Gelder

- Interests: Data science, machine learning and automation
- Responsibilities: Frontend, matrix signs, contour plot

We all contributed to testing, preparing the report and the final project presentation.

Contact

Client Kin Fai Chan - Scenwise

TU Coach Asterios Katsifodimos - EEMCS, Web Information Systems, TU Delft

Contact person Stephan Tromer - stephan@stephantromer.nl

The final report for this project can be found at: `http://repository.tudelft.nl`