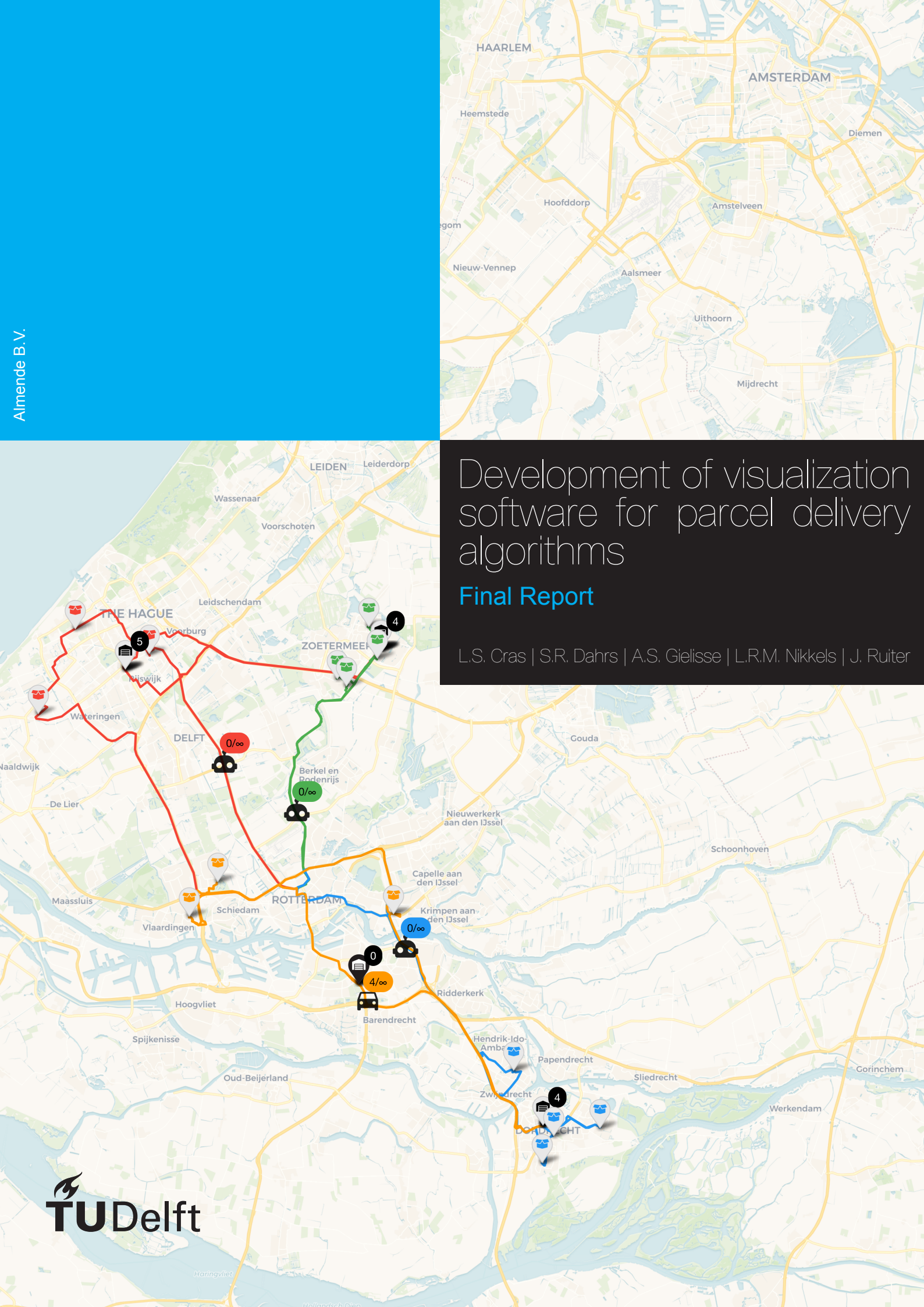
Almende B.V.



Development of visualization software for parcel delivery algorithms

Final Report

L.S. Cras | S.R. Dahrs | A.S. Gielisse | L.R.M. Nikkels | J. Ruiter

TU Delft

Development of visualization software for parcel delivery algorithms

Final Report

by

L.S. Cras | S.R. Dahrs | A.S. Gielisse | L.R.M. Nikkels | J. Ruiter

to obtain the degree of Bachelor of Science
at the Delft University of Technology.

Project duration:	April 20, 2020 – July 3, 2020	
Supervisors:	Ir. C.A. Hermans,	Almende B.V., client
	Dr. M.T.J. Spaan,	TU Delft, coach
	Ir. O.W. Visser,	TU Delft, coordinator
	Dr. H. Wang,	TU Delft, coordinator

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

TU Delft

Preface

This document is the final report for the TI3806 Bachelor End Project course offered by the TU Delft as part of the Bachelor's degree in Computer Science and Engineering. This report contains all important information concerning the 11-week project, of which two weeks were spent researching and the other nine weeks were spent actively implementing a web application for the company Almende B.V.

We would like to take the opportunity to thank everyone who guided and supported us during this project. This includes Andries Stam, Veerle van der Tas and especially Carlos Hermans from Almende B.V. who were continuously involved in research and development and provided us with feedback. A special thanks also goes out to our TU Delft supervisor, Dr. Matthijs Spaan, from the Software Technology department. His guidance and insights in the development and research phase were of great help.

L.S. Cras | *S.R. Dahrs* | *A.S. Gielisse* | *L.R.M. Nikkels* | *J. Ruiter*
Delft, June 2020

Summary

Almende B.V., a technologically innovative and research-oriented company, has been working on a new algorithm that optimizes routes for parcel delivery trucks. The algorithm contains novel features, like including the possible use of autonomous vehicles, that are at this moment in time not taken into account in existing route optimization algorithms and thus visualization applications. To this end and to get a more tangible overview of the algorithm's behavior and performance, they requested to have a customized visualization tool developed. This report describes the process and results of developing such a tool. The tool is presented as a single-page application and has been partly depicted on the cover of this document[1]. The goal of the project is to have a more clear overview of the routing algorithm's capabilities, by showing its unique features on a map and displaying statistics on the side. In addition, comparing the algorithm to existing ones should provide added insights into the (expected) benefits of the new algorithm. The main purpose of the tool developed in this project is to show insight into the workings of the algorithm and to help with enhancing and developing the algorithm. An added side-bonus is that the tool can also be used to show the performance to various groups of interested parties.

The first two weeks of the project were spent on researching the principles of data visualization and then diving into the best way to visualize the algorithm being designed by Almende B.V. Examples of questions that were asked and answered are 'what are the most important characteristics of such algorithms' and 'how can we best convey the performance of those characteristics while also highlighting why this performance is the way it is'. We also spent time looking for potential platforms and implementation details for the visualization tools such as on which library to build the frontend, etc. Answers to the questions posed above resulted in the decision to create both a map showcasing the routes, vehicles, deliveries, and more, but also a view of the various performance indicators in the form of statistics. The map can then form the basis for understanding why a certain performance indicator lands on a certain result. The most important characteristics that were decided on being shown are indicators including but not limited to cost, time and kilometers driven. These statistics are easily extendable, for whenever the need to inspect another aspect of the algorithms performance becomes necessary.

The final solution is a web application that rewrites the output of different routing algorithms to a standardized format on the server and sends this data to the frontend, where the routes are subsequently displayed in the map view and statistics are calculated for it. The code is designed according to framework standards and the product's interface is designed to be intuitive and user-friendly. While some aspects of the tool turned out to be not (yet) completely perfect in the final stages of the project, the visualization tool implements all major and most minor requested features and sheds light on the unique features of the algorithm. It can be concluded that the end product indeed solves the problems raised by Almende B.V.

[1] More visual representations of the application, including descriptions, can be found in chapter 3

Contents

1 Introduction

Efficient algorithms are essential for optimizing cost when offering a parcel delivery service. In its essence, a parcel delivery algorithm is a generalized and more elaborated variant of the traveling salesman problem, also known as a vehicle routing problem. Improvements in the efficiency of these algorithms are continuously being researched and developed, as technological developments progress.

Almende B.V. is a company that is currently working on one such innovative routing algorithm in cooperation with a master's student from the Delft University of Technology, Veerle van der Tas, and an external client, namely the algorithm is being designed as part of the I-Cave (Integrated Cooperative Automated Vehicles) research program. More specifically they are researching the benefits and applications of a fleet of autonomous vehicles in the problem space of parcel delivery systems. As part of that project, Veerle van der Tas is working in cooperation with employees from Almende B.V. on an algorithm taking into account that some roads may allow for traversal by autonomous vehicles which would eliminate human labor and with it costs for part of the delivery process in addition packages can be turned over between trucks, allowing for more efficient distribution over the course of a day.

Since the use case described above is fairly specific some of these features are not supported by existing route visualization tools, it is for this reason they requested the development of a customized application that would visualize the algorithm's output in a comprehensive manner for both development as well as showcasing of the algorithm. In chapter 2, the scope and exact definition of the problem will be discussed more extensively.

To help with testing, analyzing and developing this visualization application, the best ways to visualize the properties, behaviors and heuristics of algorithms such as the one they are creating, but also of already existing parcel delivery algorithms, were researched in the early stages of the project. The results of this research as well as research in ways to develop the tool itself can be found in Appendix D.

After the research phase, development started which includes the design and implementation of the application. Chapter 3 describes the design and the design goals that would serve as guidelines for developing the tool. This concerns both the design of the code and structure as well as the visual design of the tool itself. The technical details of the proposed solutions are discussed more comprehensively in chapter 4 about the implementation.

Something that sets this project apart from those conducted in a 'normal' setting is the fact that no in-person meetings have taken place between the team members or between the team and the client at any point during the project. This is due to the measures that were raised in response to the COVID-19 pandemic. Because of this, teamwork was limited to online meetings only, which made communication and teamwork more difficult and much scarcer than when the whole team (and client) could be together five days a week from nine to five. This also took away the experience that could be gained from working on the premises of Almende and getting to know their employees and office atmosphere. Chapter 5 discusses how this situation was handled more in depth and talks about other methodologies that were used to aid in this situation.

Aside from this, developing the visualization tool came with a number of implementation challenges. For example, during the first four weeks of developing the visualization tool, development is based on the OpenRouteService output, i.e. output of a 'standard' routing algorithm, because Almende's algorithm is still being developed. After five weeks, sample output data of the algorithm of Almende B.V.

was provided, and this proved to have quite some conceptual differences with the OpenRouteService data that had been used, which resulted in a number of implementation challenges. These challenges were eventually overcome, mostly at the cost of time and some precision, after which the tool could visualize both Almende's algorithm, and the OpenRouteService algorithm. Chapter 6 elaborates on this challenge, and on the other challenges faced during development as well as their solutions.

Finally, chapter 7 concludes the findings and end-results of this project.

2

Problem Definition and Problem Analysis

The problem definition for this project has changed slightly towards the end of the project, but was mainly defined within the first two weeks. The title of this project, which can be found in Appendix B along with the original project description, already illustrates what the problem would look like: “development of visualization software for parcel delivery algorithms.” Even though this might sound rather straight forward, there is a lot more to it; the problem definition included things like autonomous vehicles, autonomously traversable roads, the possibility of packages being handed over from one truck to another on the way, and the possibility to compare algorithms.

2.1. Visualization

Gaining insight from data that is normally extremely difficult to read for humans can be of great importance for any business decision. Visualizing parcel delivery algorithms could, therefore, not only be a useful aid in the development of such an algorithm but also in the decision making process of Almende. Logically, the idea of showing a map on which vehicles drive around would be the first step towards a gaining knowledge of how a parcel delivery algorithm behaves. This is the base of the problem definition from which there was worked towards the end-product. In the sections below, a problem definition will be provided for each of the individual problems that were ought to be addressed. It will be described in detail what information was available at the time and which problem definitions were created from that.

2.2. Purposes

The visualization has two main purposes: debug during development and presentation to customers. Both of these purposes were taken into consideration when forming the problem definition. ‘Debug during development’ means that the visualization can be used during the development of the algorithm. By seeing actual vehicles drive rather than having raw text data, it should be way more clear what the idea behind the found solution to the parcel delivery problem is. On the other hand, this visualization tool should be usable when convincing a potential customer of an algorithm that the newly developed algorithm is significantly better than the one it could replace. This further expands on the initial problem definition, namely that making a comparison between different algorithms must be possible, which will be discussed further in section 2.5.

So, in conclusion, the goal is to build a visualization tool that can convince and prove to both the developers, and the potential customers that the parcel delivery algorithm that is developed, is significantly better than another parcel delivery algorithm. That way, the developers can be sure that if they are convinced of their algorithm, they can also convince their potential clients with it through an easily understandable platform. Another benefit of integrating the visualization application with the development of the algorithm it was fine-tuned for, is the fact that the developer of the algorithm is familiar with the visualization, can possibly make alterations to it down the road, and knows all its features so that they can optimize their presentations for customers.

2.3. Autonomously Traversable Roads

As part of the I-Cave research challenge that Almende will participate in, autonomously traversable roads are a must for the visualization. The idea behind this is that not all roads within a parcel transportation network have to be driven by humans, but may be driven autonomously (without the presence of a driver) in case a given road allows this. Intuitively, taking an autonomously traversable road would be preferred over being required to have an active driver present. If a given algorithm were to make the decision to take a non-autonomously traversable road, it should be visually clear why this decision is made instead of taking a possibly nearby autonomously traversable road. Vice versa, if an algorithm were to take a large detour just to be able to drive a short distance autonomously, this would likely not be desirable. Having a good visualization on which roads can be driven autonomously plays a large role in the visualization, which is why this is one of the problems from which the final problem definition was built up. The actual decisions that were made in an attempt to solve this problem are discussed in the design chapter (3).

2.4. Package Handovers

In traditional parcel delivery systems, packages that are addressed to a destination within a certain area, are first sorted and sent to a distribution center in the vicinity of the delivery location. From there, the package is loaded on a truck which will then drive round delivering all the packages to their destinations. This inhibits same-day deliveries, even when the distance that needs to be driven is relatively small, because the truck can not receive new packages while driving. Almende is exploring a somewhat different idea. This idea is that trucks do not only pickup all their packages at the beginning, but these trucks can also receive new packages while delivering other ones by meeting up with different trucks that are also delivering packages. Whether the concept of handing over packages brings sufficient improvement in flexibility and or robustness should become apparent from the visualization tool. In either case, the actual result of whether it does is not part of the scope of this project, as the sole aim of the project is providing a good visualization of a given scenario. The problem here is: how does one visualize that a given package was handed over? Does one simply put a marker on the map? This brings lots of new problems when lots of these handovers happen, as it is very easy to lose control of the visualization. The description above is the base foundation of the problem definition for visualizing package handovers. Also for this problem definition; the actual decisions that were made in an attempt to solve this problem are discussed in the design chapter (3).

2.5. Comparison Mode

The comparison mode is one of the requirements of Almende that played a vital role in the development of this visualization tool. How would one visualize something like robustness? With robustness is meant: how prone is a given solution to small mistakes and failures like a road block or a sick driver? The problem definition for this was rather straight forward, but quite difficult to solve. Because how does one visually explain this robustness without having to know the exact works of the algorithm? The problem obviously required that two solutions should be comparable. Putting both solutions on a single map seems like an obvious choice, but after giving it more thought, this would not give the desired overview as things could be confused or get cluttered easily. Therefore using two separate maps would be the most logical solution. However, the initial problem definition stated that simplicity was of great importance, thus keeping everything on a single page were a must. Therefore the problem definition becomes more complex, because visualizing two solutions on a single page could become packed, thus not providing the desired overview. The actual decisions that were made in an attempt to solve this problem are discussed in the design chapter (3).

2.6. Requirements

With all the main features evaluated, a list of requirements was constructed in collaboration with the client. These requirements were set up according to the MoSCoW method and are listed in the research report in Appendix D. For readability purposes, the requirements are repeated in the enumeration below. Subsection 6.1 contains a second overview of all the requirements and whether they are completed or not, accompanied by a brief evaluation.

1. Must haves
   (a) The visualization program has to be accessible in a web browser. This makes it easy for the client to use it on any computer they want.
   (b) To benchmark the performance of Almende's algorithm against other similar algorithms, the final product must have the ability to visualize other algorithms that try to solve the parcel delivery problem.
   (c) The vehicles and their planned route must be visualized on a map. Handovers of packages, pick-up points and packages should be clearly visible. New orders should update the map accordingly. It must be distinguishable whether a route is for autonomous vehicles, controlled vehicles, or a hybrid.
   (d) Important metrics of the algorithm, such as cost and robustness, must be visualized in a clear and understandable way,most likely in the form of a chart. These charts are to be updated in real-time as the simulation progresses. The metrics must be broken down into granular parts.
   (e) The visualization has to display the chosen algorithm with its properties.
   (f) For the client it is valuable if they can use the code base without any problems for future development. If the product needs improvement after the project finishes, they must be able to easily work with the code.
2. Should haves
   (a) In order to present more information while keeping the map clutter-free, a clear overview of the different orders, their accompanying deliverers and the routes of these deliverers should be visible in an ordered list visible to the user.
   (b) The end-product should be able to scale up, showing a bigger area at once with-out cluttering the screen and maintaining a good, but probably less granular overview of the simulation and workings of the chosen algorithm.
   (c) In order to prevent outliers and be more sure of the end-result, the visualization tool should give users the option to per-form random simulations multiple times with the same algorithm at once, and then give the average solution.
   (d) When something happens that the user misses or wants to focus on, there should be an option to pause, or play back the simulation. The user should also be able to skip-forward if they would like to speed up the simulation.
   (e) To properly analyze Almende's algorithm,it is useful to aggregate some statistics. With those aggregations you can deduct average behavior of the algorithm.
3. Could haves
   (a) In order to convey more meaning with-out cluttering the screen, sounds can be utilized in the end product to make the user aware of certain events taking place. These sound cues should then change with different scales of the visualization.
   (b) The user can interact with the live visualization. For instance, he can manually block a road and see how the algorithm deals with that.
4. Won't haves
   (a) Seeing as showing the visualization in 3D does not add much value, except for maybe the wow-factor, and the amount of work that would go into adding this feature, it will not be implemented for the end-product.
   (b) Because the visualization contains a lot of information that needs to be presented to the user, and because this feature also requires a lot of work, without adding much value (because the product can also be viewed on a browser on a phone), a mobile application for either android, iOS or both will not be developed.

3

Design

In this chapter, design decision throughout the product are substantiated and elaborated upon. The design aspects are split into code design, which entails the structure of the code and its interfaces, (section 3.1), and the layout of the application, which encompasses the visual design choices that a user interacts with (section 3.2).

3.1. Code Design

One of the client's first requirements is to have the final product on a web server. This makes the application easily accessible at any location and on any device (although the application is optimized for pc's). When designing a web application, it is common practice to split the implementation of the application in three parts: backend code, or server-side code, which should handle heavy computations and sensitive information; frontend code, or client-side code, which should be aimed at shaping the layout of the content it received from the backend; and a database, which stores the data that is used and displayed in the backend and frontend. Figure 3.1 contains an overview of the setup for the visualization application, and each component is separately elaborated upon in this section.

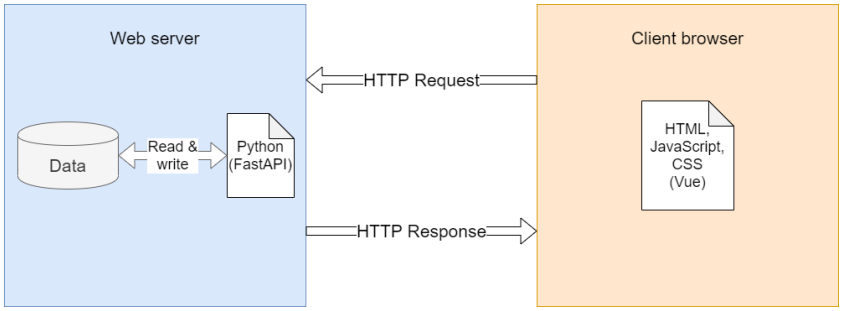


Figure 3.1: A high level diagram of the visualization application's setup.

3.1.1. Assets

It is common practice for web applications to abstract data from code as much as possible. As a consequence, databases are often stored on separate servers or hosted by external parties. For the Almende parcel delivery visualization tool there was decided not to use such a separate/external database for the following reasons: first and foremost, the data that is used for the visualization is created on-the-fly. On every input change or settings change, the underlying routing algorithm will be asked to recalculate its routes and send these to the visualization application. This means there would only be a small speedup on the initial loading of the screen, when using stored routes and statistics. This speed improvement is considered marginal, and especially cumbersome considering the extra costs of hosting a database. Nevertheless, the application still requires to have access to initial input data for the algorithms to be visualized. In the implementation this data is saved on the web server in the `./assets/` folder. This folder has a size of 6MB, 99.67% of which is taken up by a read-only GeoJSON file that contains all

potentially autonomous roads in The Netherlands. The rest of the files are input variables and settings that are used on load of the implemented algorithms for the visualization, such as delivery locations, depots, vehicles and an input scenario.

3.1.2. Backend design

The backend of the Almende parcel delivery visualization makes requests to the actual routing algorithms and translates these to a format that is interpretable by the frontend. In addition, it loads information about the implemented algorithms and (potentially) autonomous roads. Finally, it is also in charge of changing settings and generating sample delivery locations. There is a wide range of programming languages and frameworks that can be used to implement backend functions, the most commonly used languages for this are PHP, Python, Java and C++. For the Almende parcel delivery visualization algorithm it was decided to use the language Python and framework FastAPI. Elaboration on these choices and corresponding argumentation can be found in the research report (Appendix D), which contains the research that was conducted before the start of the implementation of the visualization. For readability purposes, figure 3.2 contains an overview of the final version of the backend's file structure.

The backend file structure was initially that of a standard FastAPI application: the home folder contained a file `./main.py` and `config.py` and a folder `./routes`, which contained the code executed on requests. Later, to accommodate for tests, the functional code was moved to a folder `./application` and tests were constructed in a separate folder `./tests`. Section 4.2 contains more details about the implementation of the backend.

3.1.3. Frontend design

As mentioned in section 3.1, frontend code should shape the data it receives from the backend in a clear and user-friendly manner. For the project at hand, this means the frontend should accommodate a map that displays the routes calculated by the algorithms, along with the events and special actions that occur. In addition, there should be some statistics or dashboard with metrics, and finally, some way for the user to set animation settings or input settings.

Client-side code for web applications usually exists of HTML, CSS and JavaScript. There are numerous JavaScript frameworks that can be used to simplify and modularize this code structure. The research report in Appendix D contains a presentation of different framework options and their advantages and disadvantages. From this research it was concluded that Vue[1] would be used as framework for the frontend.

The parcel delivery visualization is a single-page application, which means all desired components should be visible in a single view and the page should not have to be reloaded to refresh data/components. With Vue, one can create so-called Vue Components, which are blocks of code that can be a mix of HTML, JavaScript and CSS. By passing parameters to these components, they can be reused with different data, hereby minimizing code duplication and size. It is common practice to place such a Vue Component in a separate file. In a single-page application, this leads to a dependency structure where there is one main file that in turn contains the main component, and these can then each be broken down into smaller components. The component dependency structure of the frontend is displayed in Figure 3.3. As can be seen in the figure, the first-level components correspond to the main features that should be available in the application: a map, statistics, settings and a table with algorithm properties, and they are combined into one application by the MainFrame component. The next section will elaborate upon the visual representation of these components.

3.2. Application layout

The main purpose of the final product is to visualize the routes chosen by a routing algorithm, therefore the map view is the center of the design and the component that draws one's attention when landing on the page. To display the other features of the application, two collapsible sidebars are added. The rightmost sidebar contains statistics regarding the displayed algorithm(s). It has multiple tabs, including the statistics of the current run, boxplot statistics for the selected algorithm(s) and, when the application is in comparison mode, comparison statistics. The sidebar on the left can be seen as the control panel of the application, here the user can manage algorithm settings, e.g. which algorithm to display, and

[1] `https://vuejs.org/`

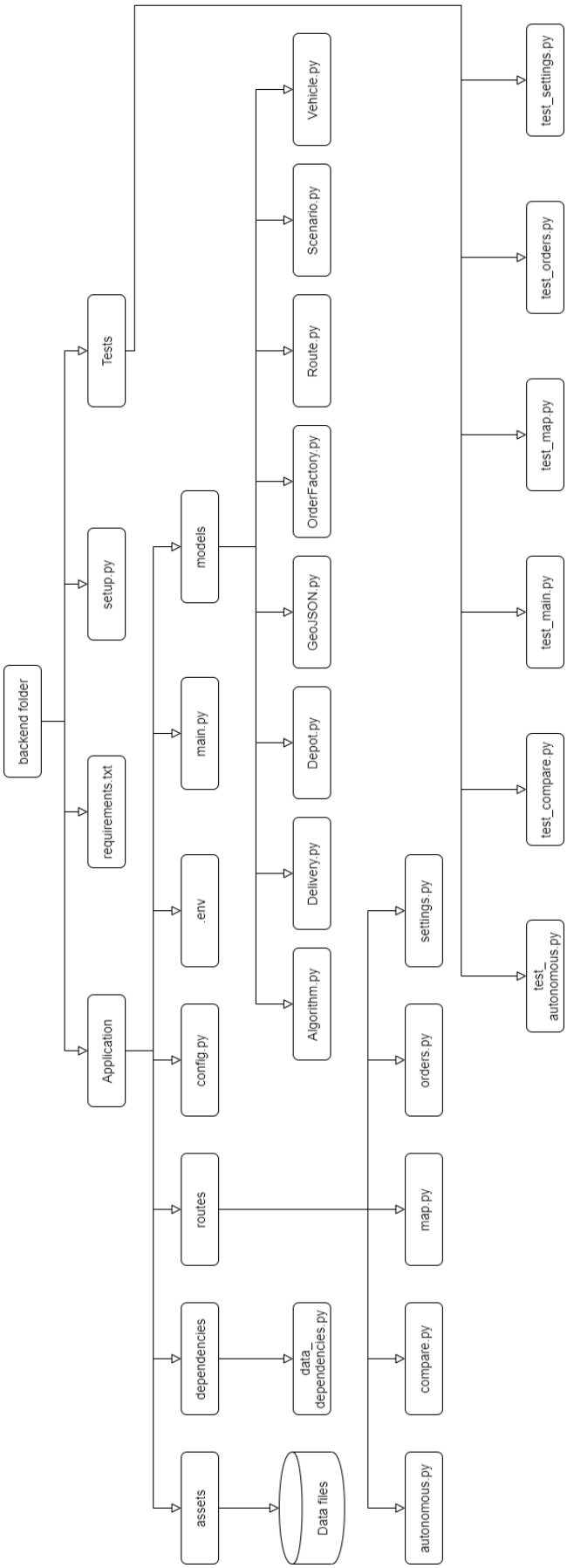


Figure 3.2: The file structure of the visualization application's backend (excluding external dependencies)

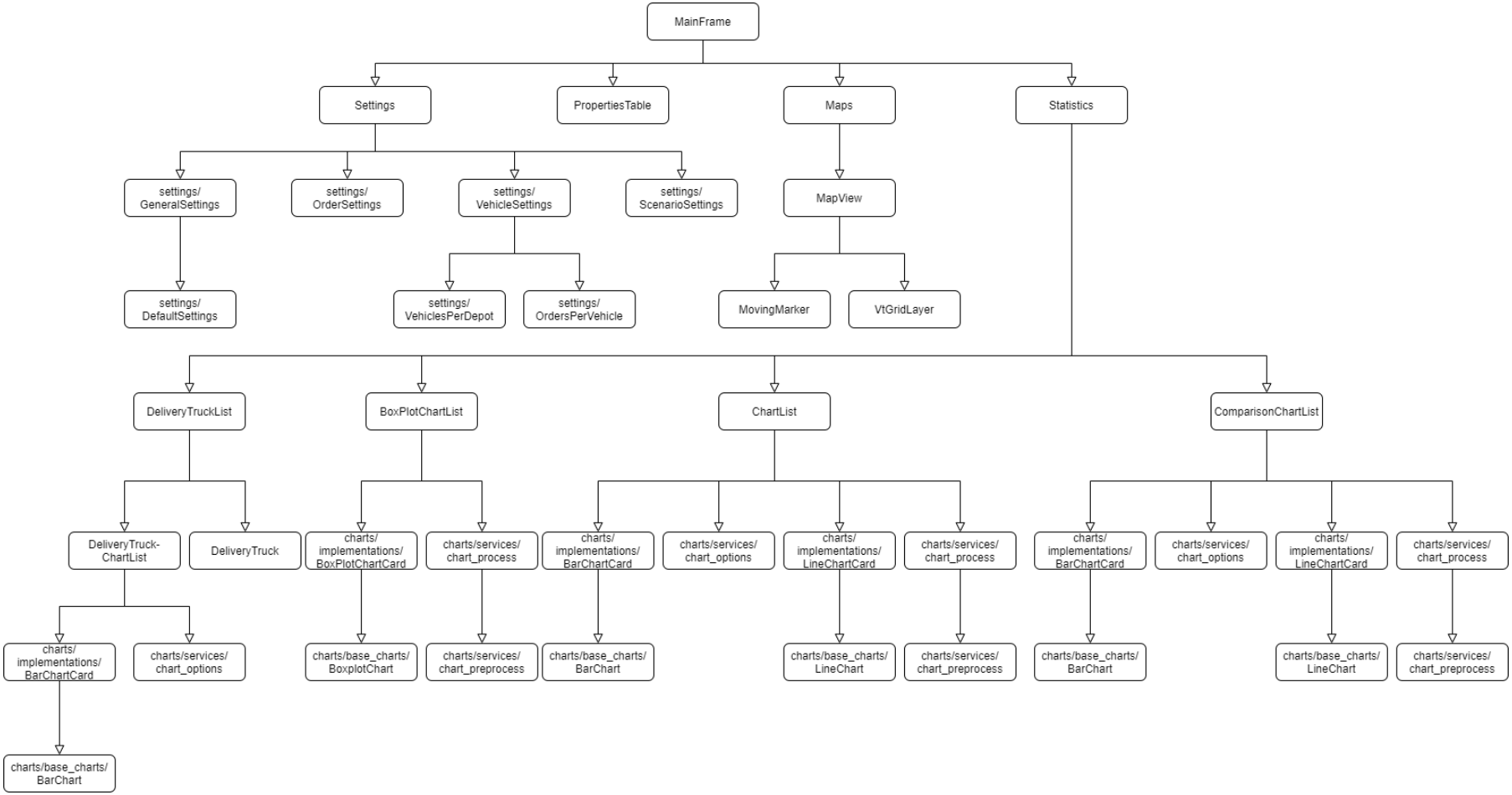


Figure 3.3: The component dependency structure of the visualization application's frontend (excluding external dependencies)

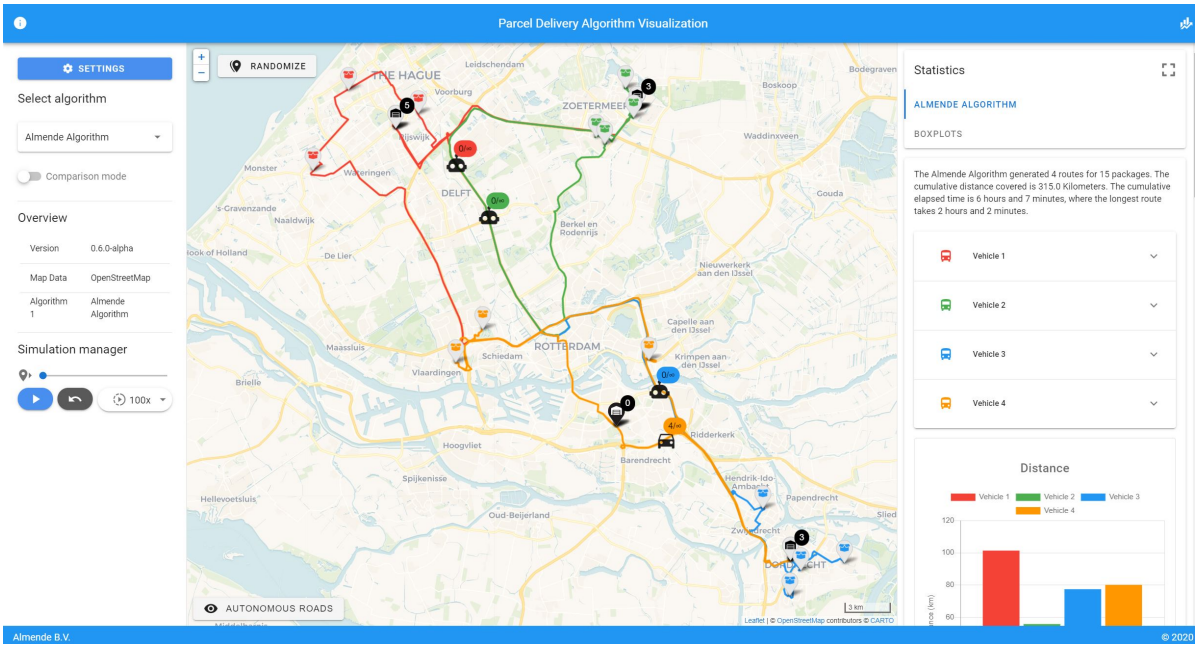


Figure 3.4: The layout of the landing page of the Almende parcel delivery algorithm visualization application.

animation playback settings, e.g. the speed of the vehicles. The following sections will dive deeper into the specifics of each of the features described above. For readability, a screenshot of the landing page is included in Figure 3.4.

3.2.1. Map

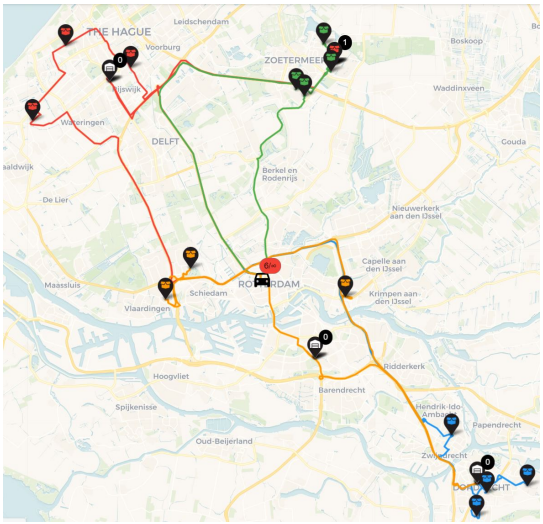
The four main elements on the map are routes, delivery locations, depot locations, and moving vehicles. Next to that, there is the double map view, also referred to as 'comparison mode', and there are some special features specifically relevant to the novel Almende routing algorithm.

Basics

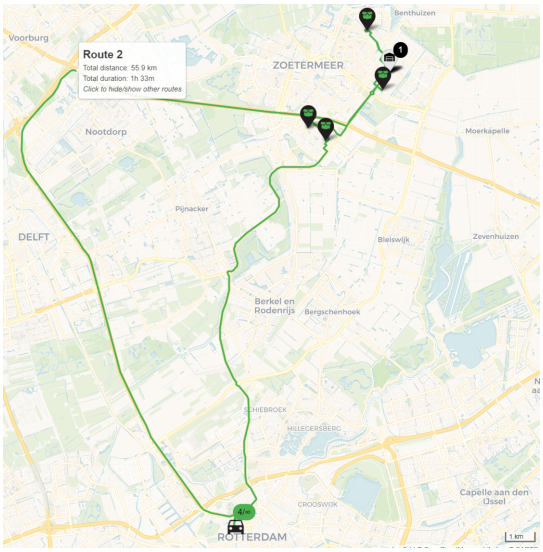
Each route is identified by a color and a number, which corresponds directly with the color and number of the vehicle that drives the route. The tooltip of a route displays a very brief summary of the route: its id, the total distance and total time. The routes' colors are the identifiers for the vehicles throughout the application, especially used to easily relate statistics with the routes on the map. The number of different colors grows linearly with the number of vehicles/routes on the map. The initial colors of the routes are primary colors, next secondary colors, and finally other colors. When choosing the colors, it was deemed important to make sure all colors appear to be of equal 'weight', i.e. their brightness and distinction is roughly equal to each other. Overall, the colors are meant to each stand out against the bland, pastel map background. A single route and collection of routes can be seen in Figure 3.5.

Delivery locations are displayed as markers on specific points on the map. They each have a unique id and their color corresponds to the color of the vehicle that eventually drops off the package. Each delivery point also has a tooltip with brief summary: id, distance traveled from its pickup location, and time elapsed since pickup. To visualize the status of the delivery, the background of the marker is either white, which means it is yet to be delivered, or black, which means it has been delivered already. A delivery layout before and after delivery can be seen in Figure 3.6 and the tooltip in Figure 3.7a.

Depot locations are displayed as markers as well. They have a different icon in the marker than the deliveries and their color does not relate to a vehicle/route, since multiple vehicles can use the same depot. At the top right of the depot marker is a badge that displays the number of parcels present at that depot. When this number is more than 0, the depot marker is 'active', which is visualized by having the background white and icon black. When the depot has no parcels stored, the background is black and icon white. At the tooltip, when hovering over a depot marker, it displays the ids of the parcels present at that depot. A depot layout when active and inactive can be found in Figure 3.8 and the tooltip in Figure 3.7b.

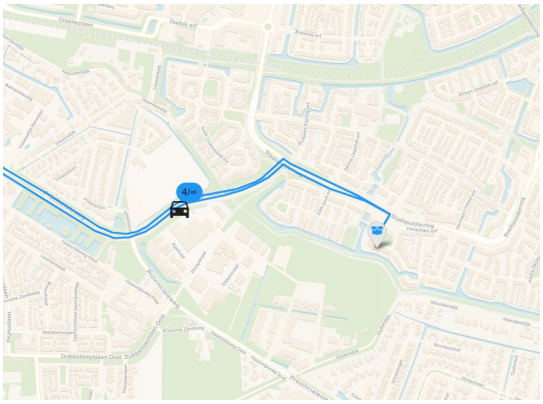


(a) The layout of a multitude of routes: a full map view. (Situation: after delivery.)

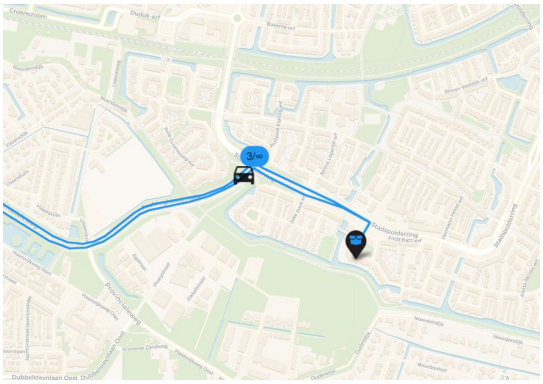


(b) The layout of a single route + tooltip. (Situation: after delivery.)

Figure 3.5: The layout of the map view.

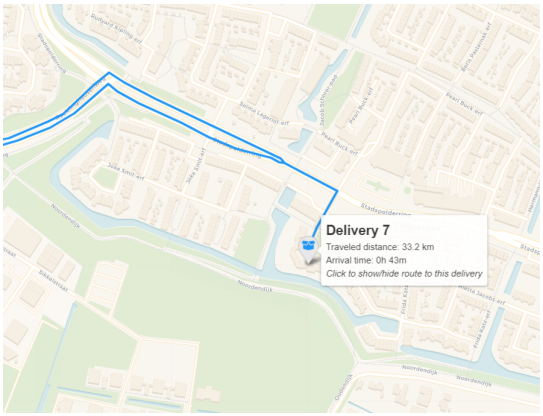


(a) The layout of a delivery marker before the package has been delivered.

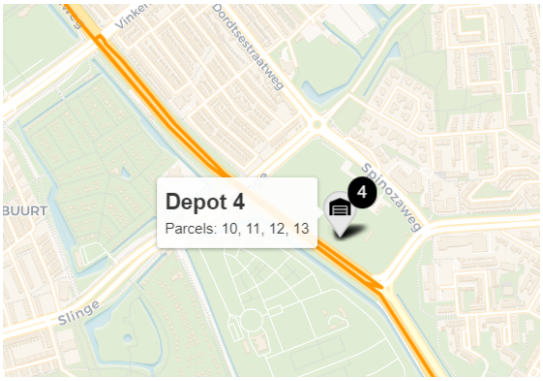


(b) The layout of a delivery marker after the package has been delivered.

Figure 3.6: The layout of a delivery marker before and after delivery.

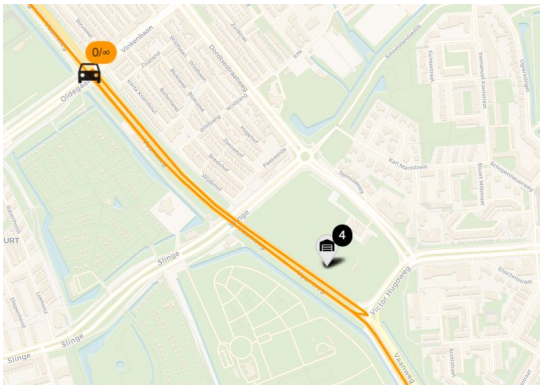


(a) The layout of a single delivery + tooltip.

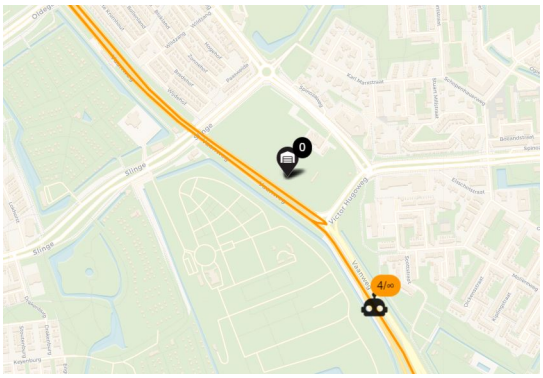


(b) The layout of a single depot + tooltip.

Figure 3.7: The layout of a delivery marker and deopt marker with tooltip.

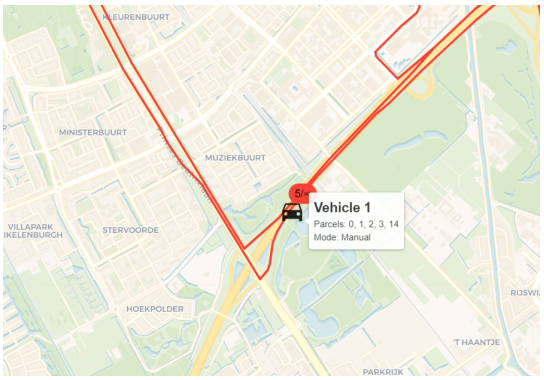


(a) The layout of a depot marker before a vehicle picks up all of its parcels.

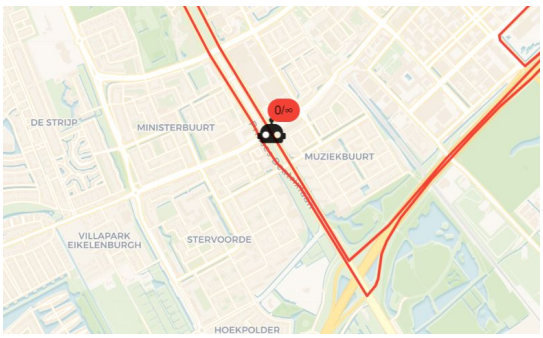


(b) The layout of a depot marker after a vehicle picks up all of its parcels. (There was also an occurrence of a mode change at the depot.)

Figure 3.8: The layout of a depot before and after pickup.



(a) The layout of a single vehicle in manual mode + tooltip.



(b) The layout of a single vehicle in autonomous mode.

Figure 3.9: The layout of a vehicle before and after mode change.

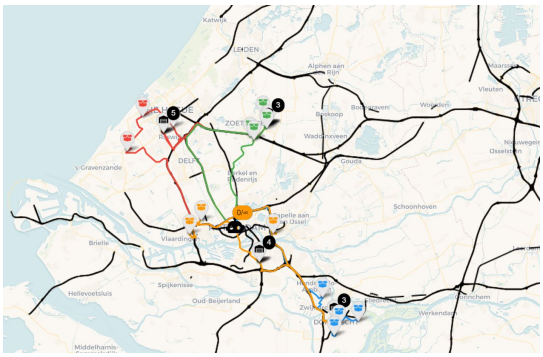
The vehicles that are moving over the route lines are the animated part of the visualization. Their movement can be managed in the control panel (subsection 3.2.3). While the vehicle icons are always black, the icon itself may change during playback: it is either a car symbol (for obvious reasons) or a robot icon to symbolize that the vehicle drives autonomously. Like the depot, the vehicle has a badge on its top right corner, that indicates the current storage inside the vehicle along with its maximum storage capacity. This badge is also used to indicate which vehicle it is, by having its background in the corresponding color. When hovering over the vehicle, the tooltip displays the vehicle's id, which parcels are in its storage space, and the current mode ('Autonomous' vs 'Manual'). A vehicle closeup and layout when driving autonomously and manually can be found in Figure 3.9.

Novel features

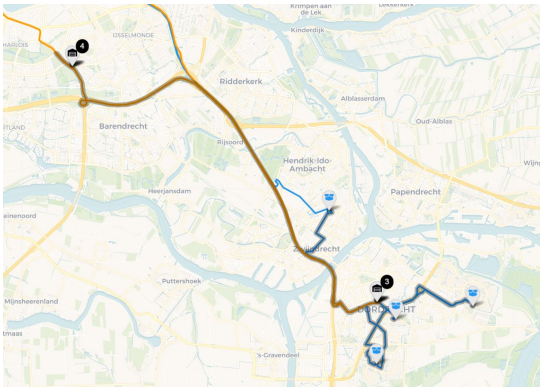
The visual entities described above are entities that are a part of most routing algorithms. However, the algorithm designed by Almende comes with a number of novel features that make it unique. These features are the possible presence of autonomous vehicles, possible occurrence of handovers of packages, and the inclusion of (forced or unforced) delays.

As mentioned in the section above, to accommodate for the autonomous vehicles, the visualization changes the vehicle icon to a robot to indicate autonomous driving. The mode changes are likely to occur at the location of a different stop (i.e. a depot or delivery point) and the change of vehicle icon is considered clear enough in these cases, but if the mode change would occur on its own, there would be a mode change marker at that point on the map that indicates where the change happens. As aforementioned, the layout of a vehicle when driving autonomously and manually can be found in Figure 3.9. Additionally, there is a button on the map view that loads an overlay of all roads in The Netherlands that have speed limit 80 km/h or higher, which should give an indication of which roads

could be traversed by autonomous vehicles (in the near future). The layout of this overlay can be found in Figure 3.10a.



(a) The layout of the overlay of roads possibly passable by autonomous vehicles. The roads marked are Dutch roads with speed limit 80 km/h or more.



(b) The layout of the overlay of a package's traveled route. Situation: the yellow vehicle picks the delivery up at the depot on the left. It brings it to the depot at the bottom (handover). There, the blue vehicle picks it up and takes it along until it reaches its delivery location and drops it off.

Figure 3.10: The layout of the highways overlay and the delivery route overlay.

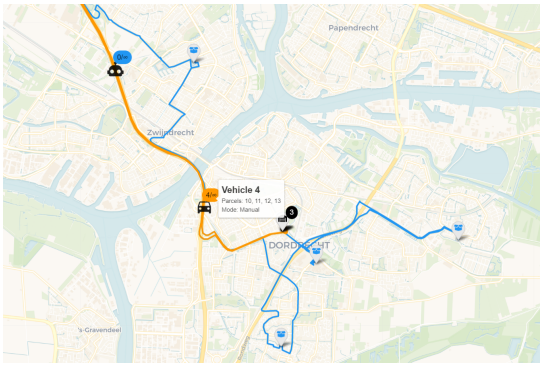
Another new feature is the possibility of package handovers. More specifically, a package can be picked up at a depot by vehicle 1 and then handed over to another vehicle 2, which delivers it to its final destination. During the handover, the package could be staying at some location for a while until the other vehicle picks it up, or it may happen instantaneously. To visualize this feature, the counters of the vehicles and their storage are updated accordingly during a handover. Additionally, when a package is at a stop location (depot or delivery), the marker at that location will be updated with a +1 in a badge at the top right corner. Depots always have such a badge, delivery markers will only have one in such a situation. In case the handover takes place at a different location, this location will be marked with a special handover marker, and this marker will be updated with a badge when a package is present at that location. For an example handover event, refer to Figure 3.11. Another function to help visualize the occurrence of handovers is the display of the route that a single delivery takes. This single route is visualized as a partly transparent, thick black overlay on top of the existing colorful routes. This route appears when one clicks on a delivery and it should maintain all other information while showing that it may or may not have traveled over different colored routes. An example can be found in Figure 3.10b.

As for the inclusion of vehicle delays, these are simply represented by having the moving marker that represents the vehicle stand still for the duration of the delay. This duration is accordingly scaled with the speed factor that is set in the simulation manager.

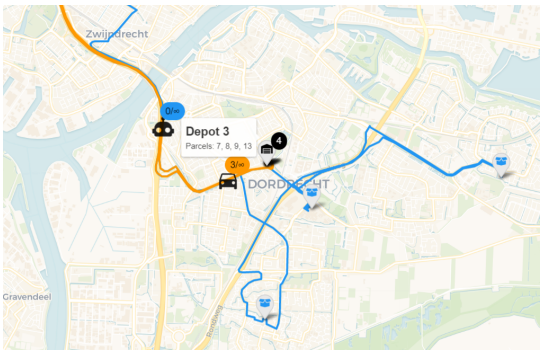
Comparison mode

Since the goal of the visualization should be to verify the efficiency and robustness of the routing algorithms, the possibility to visually compare routing decisions between different algorithms or scenarios is valuable. For this reason a 'comparison mode' is implemented. There had been some debate on what this comparison mode should look like, there were multiple options: Display both algorithms in one map view, with one color representing one algorithm; Let users simply open a new tab in their browser with a separate instance of the whole application; Or display two separate map views side by side in the same application. Having users open a new tab was decided to be undesirable because it makes it hard to visualize comparison statistics and makes the application less complete as a whole. Eventually, it was decided to use the split map view, because it would allow keeping the most details about both algorithms without becoming too cluttered. It splits the map view in two and loads a second map view with the desired second algorithm displayed. Since the application is meant for scientific investigation into the algorithm's behavior, the assumption was made that users would have access to a larger screen to use the application on, and so the double map view would not be too crowded.

The comparison mode shows the two maps side by side and allows to zoom in and out and pan each map view separately. The animation of the vehicles is still managed via the single control panel

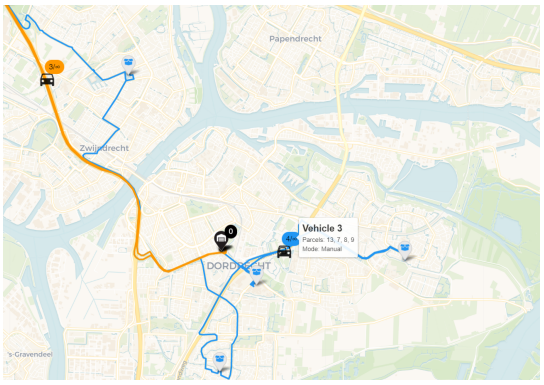


(a) The layout of a package handover before the handover takes place. Parcel 13 is now in the storage of vehicle 4 and will be handed over at depot 3.

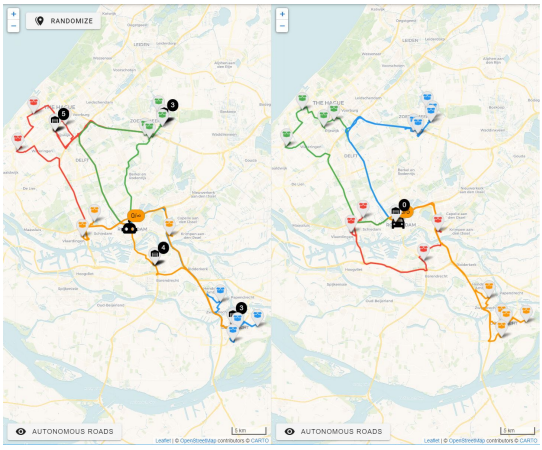


(b) The layout of a package handover when vehicle 4 has handed over the parcel and the parcel is waiting for pickup at the depot. Parcel 13 is now at depot 3.

Figure 3.11: The layout of a handover before and after vehicle 1 hands it over.



(a) The layout of a package handover when vehicle 2 has picked up the parcel from the depot. Parcel 13 is now in the storage of vehicle 3.



(b) The layout of comparison mode on load. Left: Almende algorithm. Right: OpenRouteService.

Figure 3.12: The layout of a handover after vehicle 2 picks it up and the layout of comparison mode on load.

and the animation playback (start, pause, fast forward) has effect on both map views simultaneously. The layout of the map view in comparison mode can be seen in Figure 3.12 and Figure 3.13.

3.2.2. Statistics

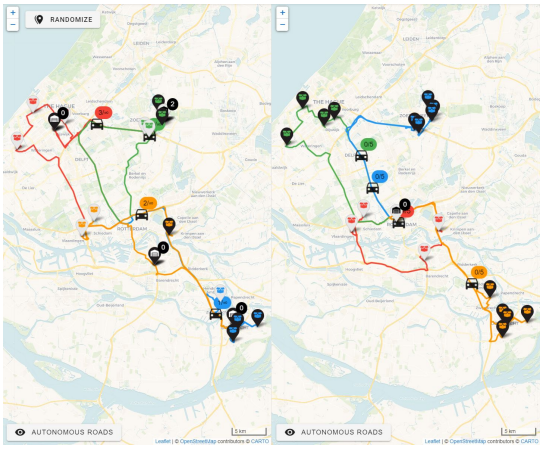
To get insights into the metrics of routing algorithms, statistics are of great help. They can visualize divergences, metrics on a timeline and cumulative and aggregated results. For the Almende parcel delivery algorithm visualization, these statistics are displayed in the bar on the right of the application. Beneath the tabs navigation menu, a small text summarizes the behavior of the algorithm displayed. Below are the charts, ordered in different categories, that will be described below.

Vehicle statistics

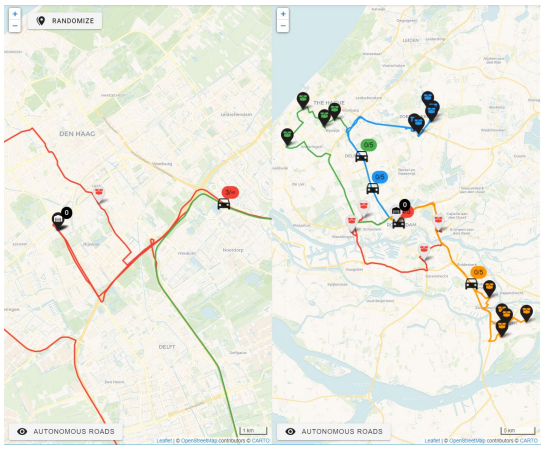
The first thing a user sees on the statistics bar is an overview of the vehicles at play. This also functions to identify the colors of the vehicles on the map view and associate the correct number. The list is expandable and opens a view with details about the vehicle. These details include a short description of the route taken by this vehicle and a bar chart that shows the time elapsed between each delivery. The layout of the drop-down menu when it is closed and when it is expanded is visible in Figure 3.14.

Algorithm statistics

Below the vehicles menu are all the other charts that display information about the currently displayed routes in the map view. All metrics are calculated separately per vehicle, so the user can easily detect



(a) The layout of comparison mode during simultaneous animation playback. Left: Almende algorithm. Right: OpenRouteService.



(b) The layout of comparison mode with single zoom. Left: Almende algorithm. Right: OpenRouteService.

Figure 3.13: The layout of comparison mode in different scenarios.

outliers or unexpected behavior and relate it to the right location and timing on the map. There are three bar charts available: one for the total distances, one for the stop counts and one for the total costs. Additionally, there are three line charts that the display the packages delivered, the distance and the costs over time. The layout of the charts can be found in Figure 3.15, 3.16 and 3.17.

Box plots

In addition to the statistics about the current routing schedule, there is also the possibility to review average, aggregated results for the algorithm displayed. To visualize this, box plots are put to use. These charts can be found in a separate tab, because they are not directly related to the routes on the main map view. The layout of these box plots can be found in Figure 3.18 and Figure 3.19a.

Comparison statistics

When the application is in comparison mode, a user logically wants to compare the two algorithms or scenarios also when it comes to statistics. To accommodate for this feature, the second algorithm also gets a dedicated tab with the same statistics as the first algorithm had in single mode. The user would also like to see the statistics side-by-side and so an extra tab appears in the tab menu, that says 'compare algorithms'. When this tab is selected, the user has access to six charts that accumulate the data of both algorithms displayed. The charts visualize the packages delivered over time and distance over time in line charts and the total values for distance, time, cost, and idle time in bar charts. The layout of these charts can be seen in Figure 3.21 and 3.22.

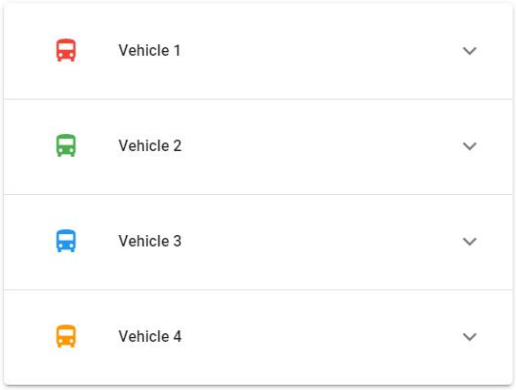
Additionally, the charts in the box plot tab are extended with a second data pair, which also allows to compare the aggregated data on the two algorithms as well. The layout of the box plot tab in comparison mode can be found in Figure 3.20 and 3.19b.

Full screen statistics

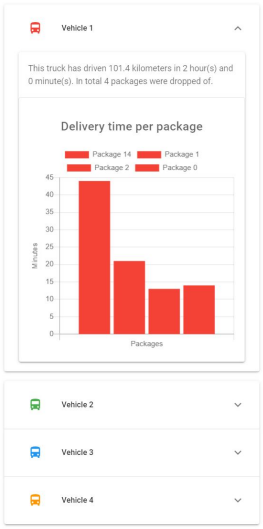
Finally, to allow users to focus solely on the statistics, and to view them in larger format, the statistics panel can be expanded to a full screen format. The full screen version maintains all the same charts, and has one additional full width bar chart. This new chart visualizes the time it takes each vehicle to deliver its n^{th} package. The layout of the full screen statistics can be found in Figure 3.23 and 3.24.

3.2.3. Control panel

Lastly, there are a number of variables and settings used throughout the application that a user would want to manage. To fulfill this need, the left panel of the application serves as a control panel, where one can manage the algorithm they would like to see, the input settings of this algorithm and the animation playback of the map view.



(a) The layout of the vehicle statistics menu when it is collapsed.



(b) The layout of the vehicle statistics menu when it is expanded.

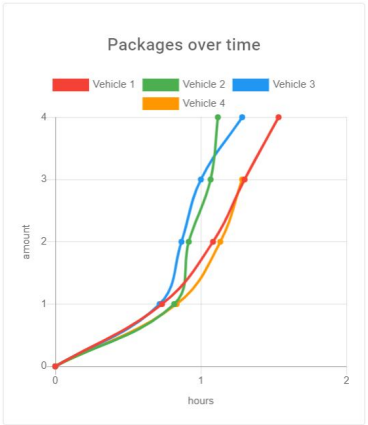
Figure 3.14: The vehicle statistics.

(a) The layout of the bar chart that displays the distance per vehicle.

(b) The layout of the bar chart that displays the number of stops per vehicle.

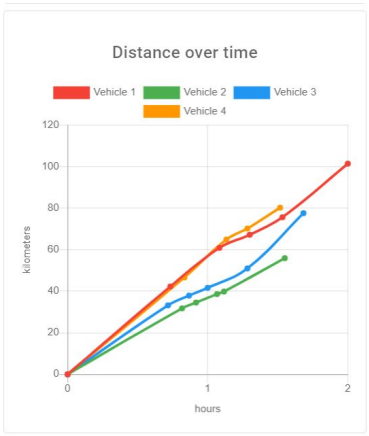
Figure 3.15: The algorithm specific statistics.

(a) The layout of the bar chart that displays the costs per vehicle.

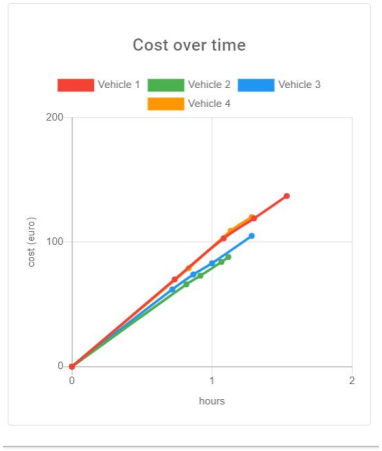


(b) The layout of the line chart that displays the packages delivered over time per vehicle.

Figure 3.16: The algorithm specific statistics.

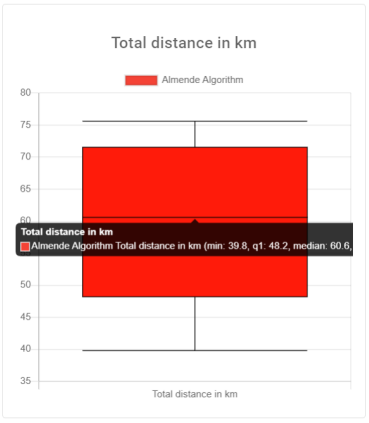


(a) The layout of the line chart that displays distance over time per vehicle.

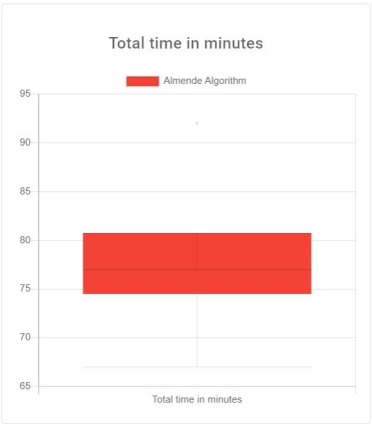


(b) The layout of the line chart that displays the costs over time per vehicle.

Figure 3.17: The algorithm specific statistics.

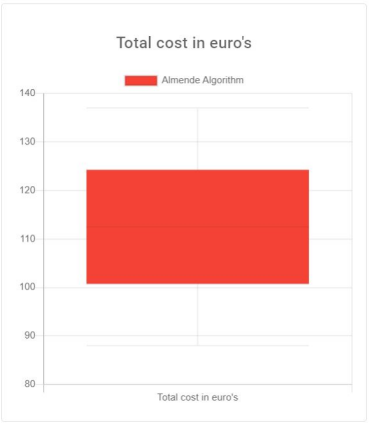


(a) The layout of the box plot that displays the total distance traveled of a routing schedule, with active tooltip.

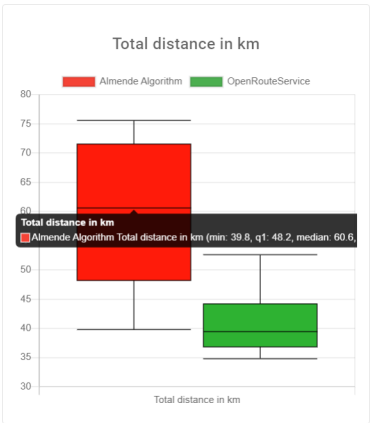


(b) The layout of the box plot that displays the total travel time of a routing schedule.

Figure 3.18: The box plots.



(a) The layout of the box plot that displays the total costs of a routing schedule.



(b) The layout of the box plot that displays the total distance traveled of a routing schedule in comparison mode, with active tooltip.

Figure 3.19: The box plots in single mode and comparison mode.

(a) The layout of the box plot that displays the total travel time of a routing schedule in comparison mode.

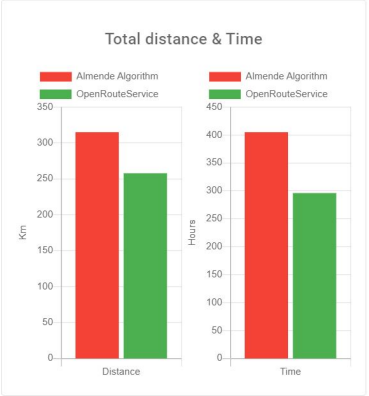
(b) The layout of the box plot that displays the total costs of a routing schedule in comparison mode.

Figure 3.20: The box plots in comparison mode.

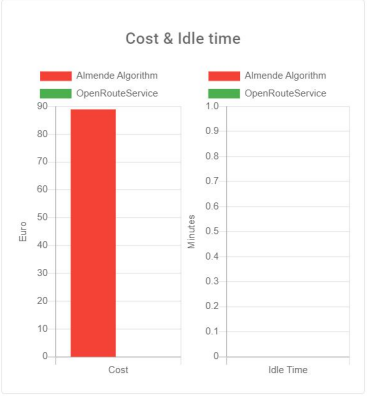
(a) The layout of the line chart that compares the packages delivered over time between selected algorithms.

(b) The layout of the line chart that compares the traveled distance over time between selected algorithms.

Figure 3.21: The comparison statistics.



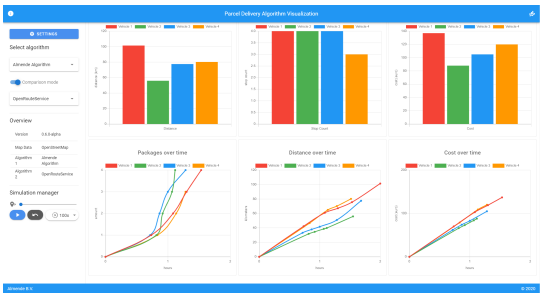
(a) The layout of the bar chart that compares the total distance and time between selected algorithms.



(b) The layout of the bar chart that compares the costs and idle time between selected algorithms.

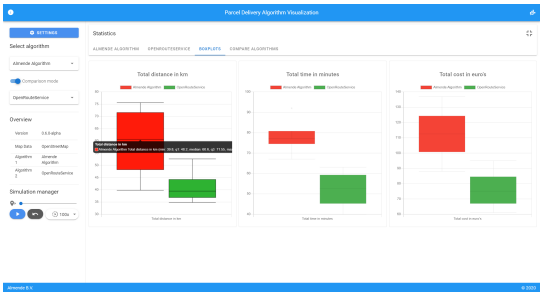
Figure 3.22: The comparison statistics.

(a) The layout of full screen mode of the statistics for a single algorithm: top half.

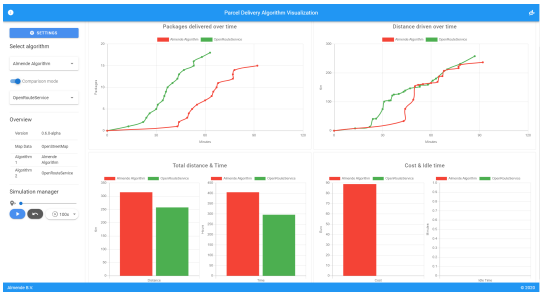


(b) The layout of full screen mode of the statistics for a single algorithm: bottom half.

Figure 3.23: The statistics in full screen mode.



(a) The layout of full screen mode of the statistics box plots in comparison mode.



(b) The layout of full screen mode of the statistics for comparison mode.

Figure 3.24: The statistics in full screen mode.

Algorithm selection and overview

The parcel delivery algorithm visualization application supports two routing algorithms: the generic OpenRouteService Optimization algorithm, and the newly developed Almende algorithm. In a simple drop-down select menu, a user can select the algorithm they would like to see.

Directly below this menu, the user can switch to comparison mode with a simple switch that triggers the appearance of a second algorithm drop-down menu. Once the user selects which algorithm they would like to compare the first with, the whole layout reloads. The algorithm selection functions can be seen in Figure 3.25.

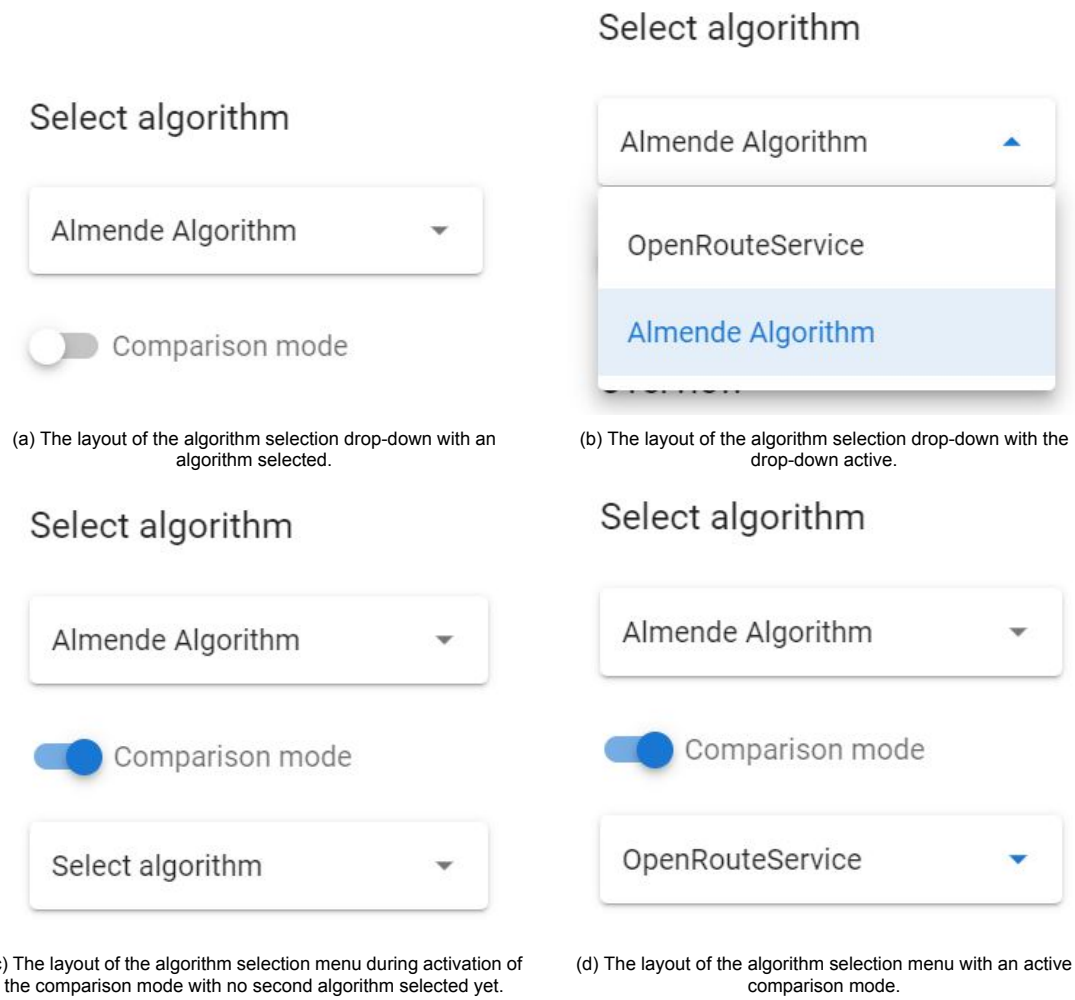
To avoid any confusion on which algorithms are being displayed, there is a small properties table in the control panel that displays the names of the algorithms. There is also a row that provides information about the application itself, indicating the current working version. The layout of this table can be found in Figure 3.26a.

Settings

At the very top of the control panel there is a ‘Settings’ button. Clicking this button opens a dialog in which all settings that are not of immediate necessity reside. The settings are divided into categories and these appear as tabs at the top of the dialog. The categories are: general settings, order settings, and one or two algorithm specific settings, depending on whether the application is in comparison mode.

The general settings include a single setting: the default algorithm to use on page load. This is the only setting that makes a permanent change to the data on the server, all the other settings are just used in session. The general settings tab can be seen in Figure 3.26b.

The next tab contains the order settings. With the sliders, users can set the number of orders to generate, and with what radius they should be rendered around the depot(s). Once these settings are saved, users can see them working when they randomize the delivery points. Randomization of delivery points is a feature that is only supported for the OpenRouteService algorithm. At the moment of development, the input format of the Almende algorithm was not known yet, and later it was not feasible to adapt the implementation to make it work for the Almende algorithm as well, since the format deviates greatly from that of OpenRouteService and it was not considered a priority at that time. Concretely,



(a) The layout of the algorithm selection drop-down with an algorithm selected.

(b) The layout of the algorithm selection drop-down with the drop-down active.

(c) The layout of the algorithm selection menu during activation of the comparison mode with no second algorithm selected yet.

(d) The layout of the algorithm selection menu with an active comparison mode.

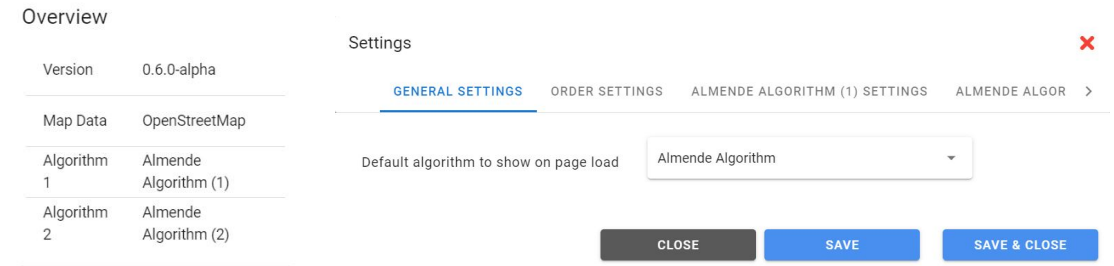
Figure 3.25: The layout of the algorithm selection menu.

this means that the part of the input that determines the locations of delivery points cannot be altered for the Almende algorithm. However, other input settings work for both algorithms, as described in the next paragraph. The order settings tab can be seen in Figure 3.27.

Aside from the global settings, there are also settings that are specific to each supported algorithm and their own input formats. To account for their differences and to not lose valuable input aspects, each algorithm has its own menu. Moreover, when an algorithm is in comparison mode with itself, each instance gets its own settings tab, so that a user can alter the settings of one instance, like the vehicle capacity or driver costs, and compare the effects that this has on the routes and statistics with the unaltered version. The layout of the settings for the Almende algorithms can be found in Figure 3.31 and that for the OpenRouteService algorithm can be found in Figure 3.29 en 3.30.

Simulation manager

The last control item in the panel is the simulation manager. With this simple menu the vehicle animation on the map can be managed. The slider on top moves along with the animation to show the current position in time. The head of the slider can also be dragged to a point in time to move the vehicles on the map to this time position. To start and pause the animation, a user simply clicks on the blue dual play/pause button. The gray arrow button in the middle can be used to reset the vehicles to their start positions. Lastly, the speed of the animation playback can be managed with the speed multitude drop-down selector. The options in the drop-down are all factors of the true speed that the vehicle would have on the route and range between 1 and 600 times the actual speed. The default is set to 100 times the actual speed. The layout of the simulation manager can be seen in Figure 3.28.



(a) The layout of the properties table when the application is in comparison mode with two instances of the Almende algorithm.

(b) The layout of the general settings tab in the settings popup dialog.

Figure 3.26: The properties table and the general settings.

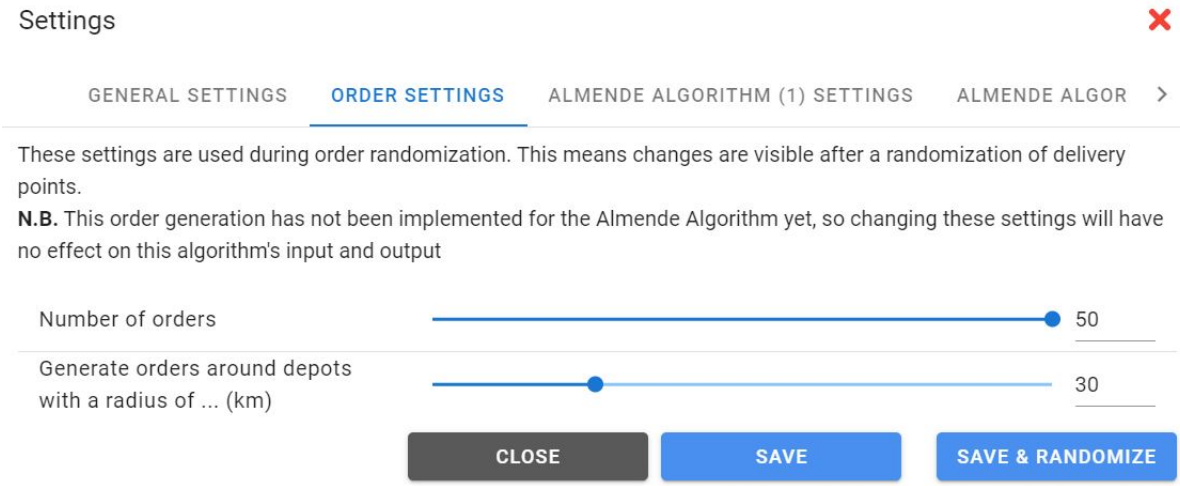
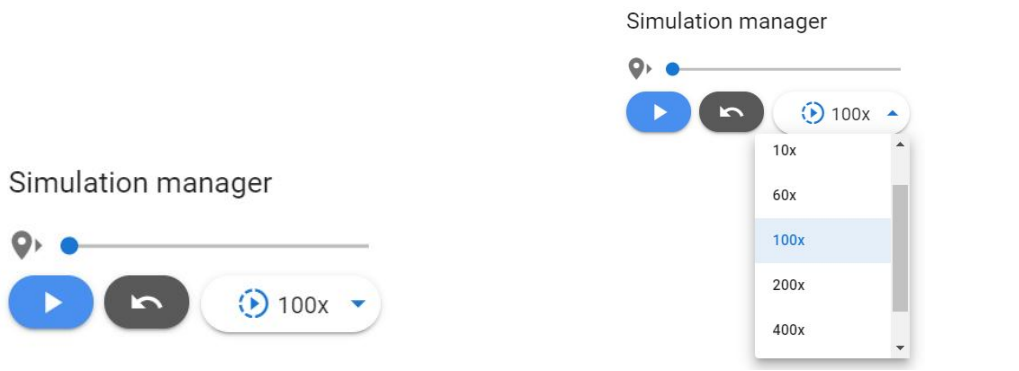


Figure 3.27: The layout of the order settings tab in the settings popup dialog.



(a) The defeault layout of the simulation manager.

(b) The layout of the simulation manager without and with the speed factor drop-down selector active.

Figure 3.28: The simulation manager layout.

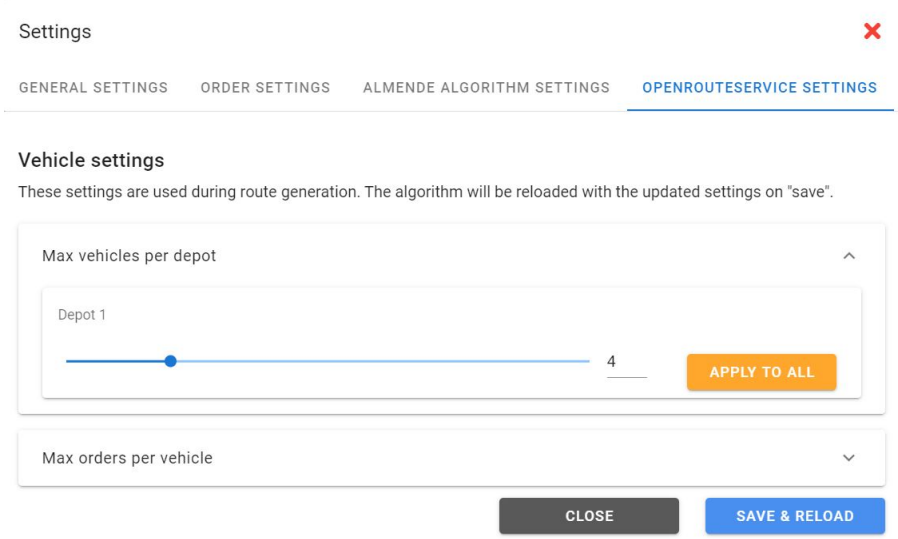


Figure 3.29: The layout of the OpenRouteService input settings tab in the settings popup dialog, with the vehicles per depot drop-down menu active.

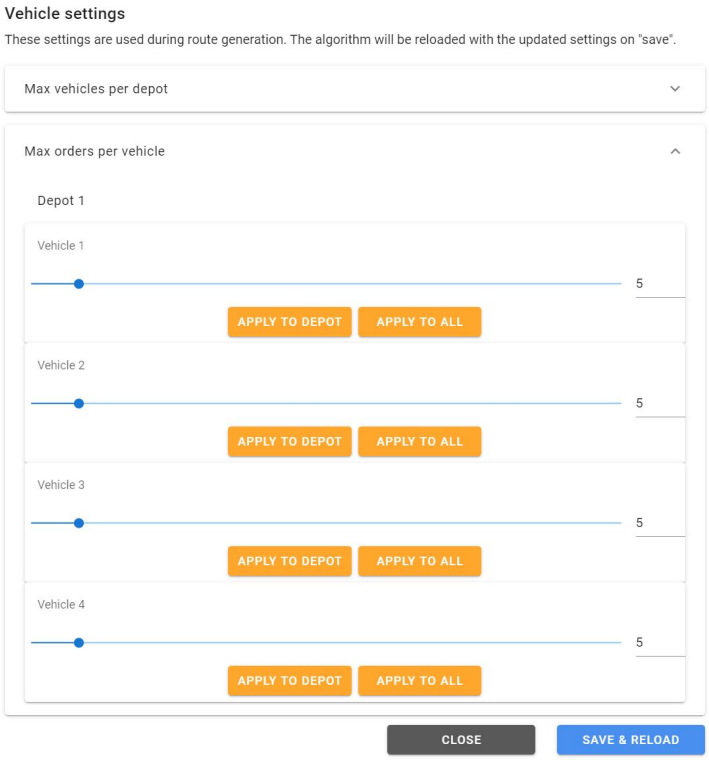


Figure 3.30: The layout of the OpenRouteService input settings tab in the settings popup dialog, with the order per vehicle (i.e. vehicle capacity) drop-down menu active.

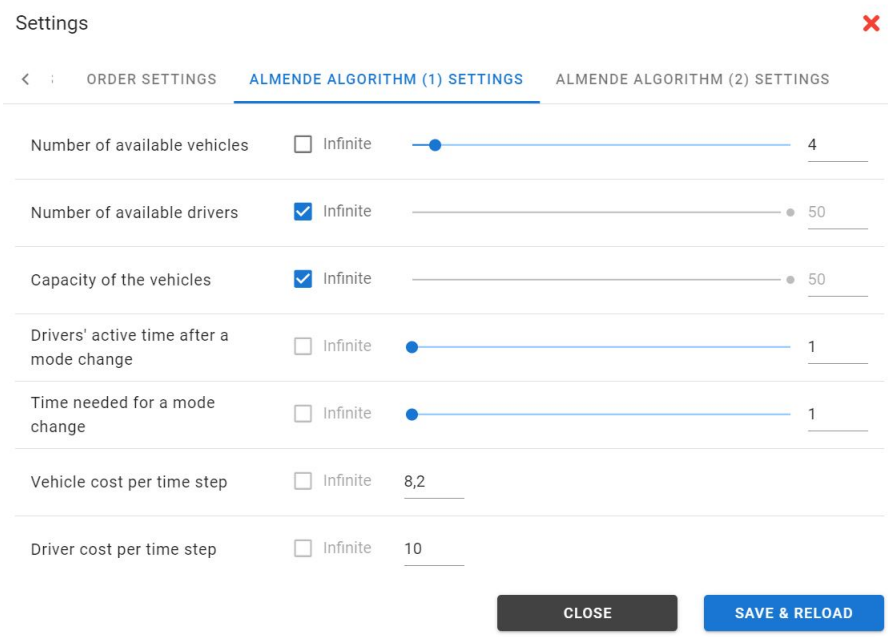


Figure 3.31: The layout of the Almende algorithm input settings tab in the settings popup dialog, when the application is in comparison mode with two instances of the Almende algorithm.

4

Implementation

In this chapter, the process of implementation is outlined, as well as the various challenges that arose during the project. In section 4.1, the implementation details of the frontend part of the application are highlighted. In section 4.2, the backend implementation of the application is explained further. Lastly, in section 4.3 the use of APIs is discussed.

4.1. Frontend

The frontend of the application is the most important because the purpose of this tool is to visualize. Therefore, a lot of attention was paid to this area in order to make sure that it conveyed the necessary information in a clear and understandable manner.

4.1.1. Map

From the research, it was concluded that the best way to approach showing the map with the corresponding visualizations was by utilizing a python package called Folium on the backend. Folium was chosen because it works seamlessly with the data that is processed and output on the backend. At first, this seemed to work great, but over the course of the first week of development, limitations in how the map was to be rendered in the frontend became apparent. Namely, Folium had a method which created a leaflet map for the user. But sending the HTML code from the backend to the frontend was not great code design and Folium had no other ways of sending its data to a frontend map. Because of this, the group decided to switch from Folium to a GeoJSON centered backend and a leaflet library designed to be used with Vue, the framework used to build the frontend. GeoJSON is a specific JSON standard used to denote information in a geographic context.

Markers

After the choice was made to use Leaflet for the map part of the visualization, a way to show the locations of depots and delivery points on the map was needed. In Leaflet, points can be shown on the map in the way of markers. But because these markers needed to be declared in the html beforehand, and the delivery points were received on the fly from the backend this had to be done differently. Luckily, Leaflet allows for a user to define a layer that visualizes a GeoJSON object directly. In this layer, points defined in the GeoJSON, as opposed to lines, are automatically rendered as markers. These points can also be dynamically styled with the help of some special functions, that were especially badly documented however. Both delivery points and depots were rendered on the map in this manner. When hovering a certain delivery or depot, extra information is displayed in a tooltip, such as the time it takes before the package is delivered, and how many kilometers have been driven up until that point.

Routes

Routes, much like the markers discussed above, are also rendered directly in the GeoJSON layer of the map. Because routes can be multiple kilometers long and most roads are anything but straight, poly-lines are used, that is a line with (potentially) very many intermediate points that the line goes through on its way to the end-point. These routes are then colored according to the vehicle they belong

to. After the basic implementation of showing the routes had been finished, a feature was added to be able to only show a certain route when clicking on that route. This is helpful when the screen is cluttered and the user wants to focus on a single delivery route. Lastly, after a demo was shown to the client in a later stage, he requested a feature to be able to see the route that was driven for a specific delivery. Because of hand-overs, the package to be delivered at that place could have been handed over between delivery trucks along the way, therefore this route might span several routes. To calculate this, the frontend alone was not capable enough because it was unaware of packages and their routes. Instead the routes are analyzed on the backend and the corresponding joined route is then sent to the frontend on activation of this feature.

Another feature that has to do with visualizing routes is the overlay toggle that shows which roads can be driven autonomously and which roads are not. Initially, in the absence of input with concrete autonomously traversable roads, this was implemented by generating a very large GeoJSON file that contains all roads in The Netherlands with a speed limit of 80km/h and over. The frontend requests this file when toggled on and displays the roads as an overlay. The reason for showing highways as possibly autonomously traversable roads was because the client had indicated in the initial phases that the autonomous vehicles were likely to be deployed on larger roads. After the implementation, when the format of the output was handed over by Almende, the feature lost some of its value because the output required the visualization tool to find routes between the delivery points. Therefore, the route chosen by the visualization tool is not guaranteed to be the same route that was used to generate the output. If the road chosen by Almende's algorithm was designated to be autonomously driven, it cannot be guaranteed that the chosen road is the same, and that the chosen road from the visualization tool is also autonomously-enabled. However, this does mean that the overlay can be used to provide an indication of which routes were actually taken by the autonomous vehicles. More about the computed routes is described in the next paragraph about the vehicles.

Vehicles

Another feature on the map requested by Almende, is the possibility to see in which order the parcels are delivered. Although one could visit all markers on a certain route one by one to see if the traveled distance increased since the previous marker, it was decided to go for the less cluttered way of implementing a simulation manager. This manager controls some special markers on the map, namely moving markers. Like normal Leaflet markers they appear on top of the map, however the `Leaflet.MovingMarker`[1] plugin is able to follow a poly-line. And because routes are composed of poly-lines, the markers, depicted as vehicles, can move on the roads. The simulation manager consists of a play/pause and reset button, a speed selector and a time slider; they control the vehicles depicted on the map.

In the development process of this feature, it was discovered that the output of Almende's algorithm only contains the destinations, rather than also the routes. Like the autonomous roads in the previous paragraph, the computed routes by the visualization tool therefore might not be the same as intended by Almende. After a discussion with the client, this turned out to not be a problem, as the routes can't differ that much.

During a meeting with the client, it turned out to be useful to see if a vehicle has visited a location already. In order to visualize that, the moving marker has to know where on the route it is located. The harder part of this implementation is to know where on its route the vehicle visits the marker in question, as the locations to visit are close to the route but not on a road. As a workaround it was decided to check multiple times per second if the vehicle is close to a marker. If that is the case, the corresponding marker will change its color.

Because Almende's algorithm works with hand-overs and these can not always be perfected such that there is no idle or waiting time for one of the trucks, either the receiving or giving party will probably have to wait a short amount of time for the other party to arrive, adding the ability for the tool to be able to show the car standing still was, due to initial implementation, more difficult than it appears on the surface. This is due to the fact that the vehicles are animated from start to finish and stopping the animation, although possible would mess up the timing in some edge cases. Luckily, the implementation choice also had another way of fitting in the stops that was less error prone. Namely by adding stations along the way in the animation.

[1] `https://github.com/ewoken/Leaflet.MovingMarker`

4.1.2. Statistics

One of the main features of the application, next to the map view, is the ability to look at statistics concerning the algorithm(s) displayed. Every statistic chart is made with the library `vue-chartjs`[2], which is a wrapper for Chart.js in Vue. This library supports a multitude of different charts, like bar charts, line charts, scatter charts and pie charts, and allows for easy extensibility. In the final product, only bar charts and line charts are used, which also came forward as the most reliable types of charts in the research report (Appendix D).

To get started, the base charts were loaded from vue-chartjs into corresponding Vue components (i.e. LineChart.vue and BarChart.vue), so that they could be called at multiple locations in the code. The chart data and chart layout options are both adjustable per chart because they are passed along as props.

Initially, the component structure was as follows: In the main statistics component, the different compartments are combined, such as the list of vehicles and the list of charts. In the list of charts component, a new chart file was made for every chart, to perform the necessary statistical computations and call the correct base chart with this data.

After doing the first SIG submission and finding out that such a structure results in a lot of (unnecessary) code duplication, the details of which are described in Appendix C, this approach was altered. In the final version, all option generation methods are contained in the same JavaScript file, to reduce code duplication on subroutines, and the data generation methods are spread over two JavaScript files. Moreover, for each base chart type, a parent component is made with additional styling in the shape of a Vue card. In the statistics list components, these chart cards are called and the correct data and options are calculated via the imported JavaScript files, and then passed along to the card components as props.

4.2. Backend

The backend is completely written in Python and developed with the use of the FastAPI web framework. A couple features of this framework are really useful. For instance, it uses model classes for input and output validation. Another feature that is convenient, is the automatic API documentation generation. It looks at the method definitions, which include the input and output model classes, and generates documentation according to the OpenAPI standards.

The backend is entirely responsible for calculating and optimizing routes. It is set up to be an API that is completely stateless. The motivation for this is that it would be relatively simple to connect a different frontend, for example a mobile app, to the backend.

4.2.1. Structure

The file structure of the backend was initially that of a standard FastAPI application: the home folder contained a file `main.py`, `config.py`, `requirements.txt` and a folder `routes`. The file `requirements.txt` contains a list of packages that should be installed in order to modify and run the application. The `config` file contains the settings of the application, which in this case translates to reading from the protected environment file which contains, for example, the value of the API key for the OpenRouteService API. The `routes` folder contains the code that is executed on client requests (section 4.3 gives a more detailed description). The `main.py` file ties the application together and determines the middleware (such as the allowed origins and headers). The Pydantic models and data dependency methods, as described in section 4.3, are contained in the separate folders `models` and `dependencies`, respectively.

Later, to accommodate automated testing with pythons's pytest library, the entire backend application is moved to a new folder `application`. Next to that, a new folder `tests` is created in which the automated tests are put. Additionally, a new file `setup.py` is made to setup and validate the structure of the application. The final structure allows to run the tests by running `pytest` and the application by running `python application/main.py`.

[2] `https://vue-chartjs.org/`

4.3. API

To interact with the frontend, it is necessary to send data back and forth, which is where the API comes into play. The API is structured as follows. There are multiple endpoints, also called routes. These endpoints are grouped in functional categories: autonomous, compare, map, orders, settings. These groups of endpoints each have their own file residing in the `routes` directory, and are referenced in the main file to create some structure in the codebase. The separate endpoint files get their own URL prefix, which is equal to their category as enumerated above.

Also, each endpoint specifies and validates its input format using a Pydantic model. If the input does not meet the desired format, a HTTP 422 Unprocessable Entity error is thrown automatically by FastAPI. Also the output format is specified using a Pydantic model class, and is also automatically validated before the response is sent. All these model classes are grouped categorically in separate files that are located in the `models` directory of the backend.

Apart from data validation in interaction with the frontend, there is also data validation for reading and writing input/data files. The python library `pandas` is used for reading and writing to file. After a read, the tuples are cast to instances of a Pydantic model. To avoid code duplication and avoid defining file locations multiple times, the backend is supplemented with single file `dependencies/data_dependencies.py` that contains these methods that can read from data in the `assets` folder.

4.3.1. Map routing

For calculating and optimizing routes between locations, a copy of the OpenRouteService API is hosted locally and used. OpenRouteService is an open source routing service that consumes free geopgrahpic data from Openstreetmap. It is used to simply calculate routes between points, but also as a comparison algorithm using the optimization feature in the public API. OpenRouteService can output it's results in multiple formats. Since the frontend uses Leaflet, which can visualize GeoJSON data, OpenRouteService is instructed to output the data in GeoJSON format.

For optimizing routes, given a set of vehicles and deliveries, the public OpenRouteService API with personalized API key is used. However, calculating routes using a set of coordinates, is part of the open source OpenRouteService code. There is a Docker container instance of OpenRouteService running on the server maintained by Almende. The advantages of having this instance are that, contrary to the API, there is no set limit of requests that can be performed, and that the performance is better since the OpenRouteService instance and the actual visualization application run on the same machine.

5

Process Evaluation and Recommendations

A development process, especially when it involves multiple developers and stakeholders, is not necessarily self-evident. This chapter will elaborate upon how this process came to be for this project and how it influenced the collaboration and product. The current unique situation, where a pandemic dominates the world and influences every aspect of normal life, also has effect on the development process of this project. Section 5.1 will explain how this was handled. The tools and standardized development methods used will be further explained in section 5.2, and section 5.3 will elaborate on how code quality was maintained throughout development.

5.1. COVID-19 response

To put things into perspective, it should be clarified that this project was conducted from April 20 until July 1 2020. A couple of weeks before the start of the project, it became apparent that the virus COVID-19 was forming a real threat to the world and people's individual health and it was officially declared to be a pandemic. As a consequence, The Netherlands (and many other countries) went into lock-down, which in practice meant that people should not be leaving their house unless they absolutely need to.

For this particular project, this meant that all work that had to be done for the completion of the project took place at everyone's home. If the pandemic would not have been in place for the duration of the project, the project members would have worked at the office of Almende B.V. during workdays from nine to five, as is common for full-time internships. While none of this has any influence on the content of the project, one could imagine that this entirely different work method certainly influenced the process, and likely the final result as well.

5.1.1. Collaboration within the team

The initial phases of a project are usually the most contact-intensive, because they require a lot of brainstorming, getting familiar with the project's content and everyone's work ethic and simply getting to know the people involved personally. Of the five students that took part in this project, only two of them were already familiar with each other and the rest were strangers. Usually you get to know people very well when you have to work with them every day, but this is less so when you never meet in person or never spend more than an hour a day talking. To get through the initial exploratory stages of the project as best as possible, it was decided early on to hold daily online meetings in the morning to update each other on everyone's progress and, if necessary, setbacks or uncertainties. This had the added benefit of making everyone familiar with each other and perhaps also of reassuring everyone that all team members were putting in the required effort.

In hindsight, this was probably a good decision and gave everyone still enough freedom to organize their own day while at the same time keeping everyone in the same lane of thought. After a few weeks of maintaining this strategy, the team decided that they were well attuned to each other and everyone had found their place in the group, and so the daily meetings organically became redundant and were

abolished. Instead, the team would communicate primarily via WhatsApp and would still meet online if there were 'severe' issues, and also before meetings with the TU Delft supervisor or client.

Overall, although the team members were not acquainted beforehand, everyone proved to be motivated and nice to work with during the course of the project. This made collaboration, even under such special circumstances and even though it may not have resulted in very deep personal connections, a pleasant experience.

5.1.2. Collaboration with the client

Before going into technical details, it may be worth noting that the team members, on an individual and personal level, may not have experienced the practicalities of this project in the way that they imagined beforehand. Part of the appeal of such a project is getting a taste of what working at a real company feels like, by going there every day and meeting the people who work there. Nonetheless, experiencing the project as it was, came with a whole lot of different, meaningful lessons.

Throughout the project, there have been two ways of communication with the client. The first is weekly meetings (usually on Friday) in which a demo was provided with the newly added features of that sprint. This was also a moment where difficulties were discussed and the client would indicate their expectations for the upcoming week(s). Oftentimes, a meeting like that would result in a number of issues that would need to be followed up, because the client's contact person, Carlos, would need to discuss certain matters with other company staff before giving a definite answer. For such situations, the communication channel that was used for the follow-up information was Slack.

Even though the combination of these two communication platforms was pleasant to work with and the Almende B.V. representatives were always quick to respond, helpful and positively involved, there are some aspects in the project where communication could have been better (chapter 6 elaborates on these aspects). From the team, there was perhaps not a sufficiently investigative attitude from the start and some assumptions were made that in hindsight should have been taken up for confirmation. This is unfortunate, especially considering that the team does not necessarily regard themselves as shy or hesitant by nature. Being present at the office of Almende and having daily interactions with the project supervisors there, however informal these interactions may be, would perhaps have resulted in a smoother and better aligned process.

These thoughts are of course based on speculation, and the main thing to take away is that communication is key in any project and, especially if there may in the future be more cases of projects that rely purely on online communication, it is recommended to take initiative and not hesitate to begin a conversation on what may seem as a small issue or problem for later on.

5.2. Methods

There are many different frameworks, tools and ideas on how projects should be carried out, especially projects that concern software development. Since section 5.1 focused on communication aspects of the process, this section explores the tools that have been used to track progress and develop the product.

5.2.1. Scrum

For Computer Science and Engineering students at TU Delft, scrum is a well-known methodology for software development projects. It follows the agile approach, and is known to be especially valuable for projects in small teams with a relatively small scope. It was therefore not a hard decision at the start of the project that the scrum method would be used here as well.

Following this approach, the workload, at that stage defined as requirements but translated to 'epics' to use proper scrum terms, was divided into weekly (Monday up until Friday) sprints. After having agreed with the client upon this roadmap, the team would divide the high-level requirements that were scheduled for that week into small, manageable pieces of work (referred to as issues) on Mondays. These were then divided among the team members such that everyone was tasked with an equal workload.

This method worked well for this project for a number of reasons. First of all, as mentioned in section 5.1, all forms of communication took place online, so in this situation where there is not a lot of social control or interaction, one could easily lose track of what team members are up to. By dividing the tasks in such small portions and being able to verify the progress of issues (explained in

subsection 5.2.2), everyone has a clear overview of the project and its progress at all times.

Additionally, this project concerns the development of a web application, which has the benefit that many team members had prior knowledge (at least from studies, but also from work experience) on the programming languages that were used and their accompanying protocols. That, in combination with the fact that in the early stages the development challenge lied more in the volume of tasks than their difficulty, makes working in small incremental steps very rewarding and effective. This is because most contributions are immediately visible in the product's design and because someone else could easily build on top of a newly added feature because it is quite intuitive. For a project like this one, using scrum has proven to be effective and nice to work with and so it is certainly recommended for future software development projects.

5.2.2. Gitlab

Scrum for software development would not be able to work without a proper version control system. To accommodate for this need, it was decided to use Git through the community edition of GitLab, which one of the team members could host on a server for free. GitLab comes with numerous built-in features, next to version control, that helped the development process of the product.

To ensure code quality and to prevent having failing or unsafe versions of the product online, a number of agreements were made among the team members in the first week of development, when it comes to branching off and merging. Firstly, the master branch should always contains a safe, stand-alone working version. Therefore, a new branch by the name of sprint-*n* was made at the start of every sprint. Moreover, for every issue that a team member was going to take upon, they had to make a new branch, sourcing from this sprint-*n* branch according to GitLab's naming conventions that link issues to branches. To set the code for an issue on the according sprint-*n* branch, a merge request had to be made and had to be reviewed and approved by at least two group members before being allowed to merge. By the end of a sprint, the branch sprint-*n* would be released to the master and the cycle would repeat.

GitLab has a dedicated section to keep track of issues (see subsection 5.2.1). Grateful use was made of it for the duration of this project. GitLab offers the option to create labels for issues (and merge requests), which was used for putting the issue in different categories. Every issue would be marked with one label to set its progress state (ranging from 'sprint backlog', 'in progress', 'testing' and 'review needed' to 'done'), one label for its relative priority (1, 2, 3, 4 or 5) and one label to set the type ('backend feature', 'bug', 'enhancement', 'frontend feature', 'release' and 'testing'). Additionally, issues are assigned a person responsible and a due date/milestone. The progress labels were used as pillars in GitLab's board view of the issues, and so it was always easy to see the sprint's progress and tasks at a glance.

5.3. Quality assurance

Several measures were in place for the duration of this project, that would ensure that the code maintained a high degree of quality. This includes an appropriate way of testing and the static verification of code quality.

5.3.1. Tests

User tests

The final product is a visualization application, meaning that the development relies heavily on visual elements, and, therefore, on the shaping of the application and data in the frontend. With Yarn and Uvicorn (in combination with a virtual environment), that were used for package management and the development of frontend and backend, respectively, each team member was able to run the web application on their own computer. When these static servers are active and a change is made in the code, the corresponding server forces a reload of the component that was altered in the browser that runs the application, and so the changes are immediately visible. It comes therefore as no surprise that user tests are the primary type of testing for this project.

User testing, or usability testing, is a type of testing where a user performs tasks on the application to be tested, which should give insights into the application's behavior and its intuitiveness. After making an alteration in the frontend code base, a developer immediately looks at the result in the application, and so it is convenient and easy to verify the correctness of the code. Since it is easy to develop

a tunnel vision on a specific element or task, reviewers of merge requests always applied user tests on the branch to merge as well, hereby testing a combination of features to make sure they were not broken. Throughout the development phase, this testing method has proved to pay off and work well. It is, however, strictly recommended to indeed have other people than the developer perform these user tests as well (instead of performing only static code tests), because sometimes changes in code have undetected side-effects or behave unexpectedly on other people's devices, due to differences in operating systems or absent dependencies.

Automated tests

Automated testing for web applications is not commonly thought to have a lot of added value. In the initial development phases, a number of tests were developed to verify frontend features, but these tests would, for example, verify the existence of components in the browser, which could just as easily be seen when simply opening the application. Frontend testing was therefore abolished after a few weeks.

Testing backend code with automated tests is considered to be more valuable. The backend in its current state serves only to parse routing algorithm output to GeoJSON. This means that any bugs in the result would be visible in the frontend or in the application. In addition, all input and output, both with data files and HTTP requests, is validated with Pydantic, as mentioned in section 4.2. For this reason, extensive testing was omitted up to this point. A testing framework including small tests can be found in the folder `tests` in which tests can be written for when a developer wants to extend the application.

5.3.2. Static code quality

Code review

For this project, it is likely that the client will build further upon the final product and will perhaps modify it if the routing algorithm interfaces change. It is therefore desirable that the code is easy to read, understandable and intuitive. Due to the team members' background in Computer Science, it is to be expected that they are to write code of a certain quality standard. When looking at the code without running it, this entails, for example, the complexity of methods, method/file sizes, maintenance of interfaces, and minimized duplication.

To verify whether these code qualifiers were kept in mind during development, this was agreed to be the second important reviewing aspect (next to user testing, see subsection 5.3.1) when reviewing merge requests. As aforementioned, at least two team members should have reviewed a code submission before merging it and with this agreement in place, the code that was going to be included in the final product was assured to be of good quality.

Software Improvement Group

As part of the TI3806 Bachelor End Project course, the code base has to be uploaded twice to the Software Improvement Group, which performs quantitative static code quality checking. The results and evaluations of these submissions are enclosed in Appendix C.

6

Product Evaluation and Recommendations

The algorithm that needs to be visualized contains numerous unique and novel features. It follows from this that when visualizing these features, one finds themselves also in uncharted waters. Although exciting and motivating, it can be challenging and requires some improvising at times. The combination of this and the fact that for many project group members this was (one of) the first time(s) that they were making a fully applied product for a real business, results into a steep personal learning curve for the duration of the course. In some cases, one can only tell in hindsight whether an earlier choice was correct or not. This chapter will elaborate on some of these choices. Subsection 6.1 evaluates whether the requirements that were set upfront were feasible. Subsection 6.2 evaluates aspects that could have gone better or may be improved during the design of the application. Finally, section 6.3 will dive into recommendations for deployment and possible future developments.

6.1. Evaluation of requirements

The requirements as they were defined in the project plan and research report, are evaluated briefly in Table 6.1. Some features require more explanations, these will be provided in section 6.2.

Table 6.1: A brief description of each requirement and whether it has been implemented and, if necessary, why not.

Category	**Requirement**	**Completed?**	**Discussion**
Must have	The visualisation program has to be accessible in a web browser. This makes it easy for the client to use it on any computer they want.	Yes	The initial setup of frontend and backend was a quick, smooth process. Moreover, a docker file was created for the final product, which made it east to upload it onto the server as well.
Must have	To benchmark the performance of Almende's algorithm against other similar algorithms, the final product must have the ability to visualize other algorithms which try to solve the parcel delivery problem.	Yes	OpenRouteService was used from the start of the development as alternative routing algorithm. Perhaps more such algorithms could have been supported.

Table 6.1: A brief description of each requirement and whether it has been implemented and, if necessary, why not.

Category	Requirement	Completed?	Discussion
Must have	The vehicles and their planned route must be visualized on a map. Handovers of packages, pick-up points and packages should be clearly visible. New orders should update the map accordingly. It must be distinguishable whether a route is for autonomous vehicles, controlled vehicles, or a hybrid.	Mostly	Most of the features in this requirement have been implemented. The visualization of autonomous/manual routes turned out slightly different than planned, and is now visualized by the icon of the vehicle at a certain point on the route. Also, all deliveries are visible from the first moment. Issues with timing are further described in section 6.2.
Must have	Important metrics of the algorithm, such as cost and robustness, must be visualized in a clear and understandable way, most likely in the form of a chart. These charts are to be updated in real-time as the simulation progresses. The metrics must be broken down into granular parts.	Mostly	Most requirements are fulfilled. However, the charts are not updated in sync with the simulation. In the development phase, it was decided, in co-determination with the client, that the amount of work and complexity did not weigh up against the benefits, considering there were already line charts that displayed the metrics over time. This decision is elaborated upon in section 6.2.
Must have	Display of chosen algorithm with properties	Yes	The visualized algorithm's name is displayed in a small properties table in the control panel.
Must have	For the client it is valuable if they can use the code base without any problems for future development. If the product needs improvement after the project finishes, they must be able to easily work with the code.	Yes	Throughout the project, it was made sure that all code was properly documented. In addition, due to FastAPI's and Vue's setup and coding practices, the code is broken up into manageable pieces by files/components. Also, frontend and backend are written in commonly known programming languages; JavaScript/HTML/CSS and Python, respectively.
Should have	In order to present more information while keeping the map clutter free, a clear overview of the different orders, their accompanying deliverers and the routes of these deliverers should be visible in an ordered list visible to the user.	No	During the development phase of the product, after the must have requirements were implemented, this requirement was left aside because the map view, in combination with the simulation and all the tooltips, provided a clear enough image of the events. A list would not be able to parallel this quality overview.
Should have	The end-product should be able to scale up, showing a bigger area at once without cluttering the screen and maintaining a good, but probably less granular overview of the simulation and workings of the chosen algorithm.	Yes	On page load, the map view focuses on the bounding box of the outmost edges of all routes, meaning all the routes should be initially in the screen. The user can choose to zoom in and the map markers resize accordingly. In addition, if the map view is zoomed out far, the map markers disappear, so that a user can still distinguish routes without the map being too cluttered.

Table 6.1: A brief description of each requirement and whether it has been implemented and, if necessary, why not.

Category	Requirement	Completed?	Discussion
Should have	In order to prevent outliers and be more sure of the end-result, the visualization tool should give users the option to perform random simulations multiple times with the same algorithm at once, and then give the average solution.	Partly	The application supports the display of aggregated statistics in the form of box plots. However, the Almende algorithm cannot be run at this point, and so there is not valid data inside these box plots in the final product. Section 6.2 contains some more information on this subject.
Should have	When something happens that the user misses or wants to focus on, there should be an option to pause, or play back the simulation. The user should also be able to skip-forward if they would like to speed up the simulation.	Yes	The control panel of the application allows the user to pause, play and reset the simulation. There is also a time slider that a user can use to rewind or fast forward. Additionally, the user can set the speed of the simulation playback. Section 6.2 contains some more notes on the simulation implementation.
Should have	To properly analyze Almende's algorithm, it is useful to aggregate some statistics. With those aggregations you can deduct average behavior of the algorithm.	Partly	The application supports the display of aggregated statistics in the form of box plots. However, the Almende algorithm cannot be run at this point, and so there is not valid data inside these box plots in the final product. Section 6.2 contains some more information on this subject.
Could have	In order to convey more meaning without cluttering the screen, sounds can be utilized in the endproduct to make the user aware of certain events taking place. These sound cues should then change with different scales of the visualization.	No	Towards the final stages of development, many high-end features that were already in place for the OpenRouteService algorithm, still had to be implemented for the Almende algorithm, once their output was received. This meant that there was not enough time to implement extra features and this specific feature was also not considered important enough to bench other features' developments.
Could have	The user can interact with the live visualization. For instance, he can manually block a road and see how the algorithm deals with that.	No	This feature turned out to be infeasible in multiple ways. Aside from the fact that the Almende algorithm was not ready for deployment and that there was no time to implement such a feature. There is the added issue with relating the routes provided by Almende (without coordinates) with points on the map. There is no way of knowing that, once a user blocks a road on the map, the removal of this edge would have any effect on the output because there is no way of knowing whether it was ever really in the solution. Additionally, even if it would potentially calculate a new route, this would not be visible on the map, because OpenRouteService provides the routes displayed on the map, not Almende directly. Section 6.2 elaborates upon this topic.

6.2. Evaluation of design decisions

6.2.1. Real time statistics updates

There is no real benefit in using real time statistics, when you also have access to statistics over time. Implementing this feature would add a lot of complexity and global variables to maintain, without much added value. As stated in the requirements on statistics, the provided charts are granular and calculated per vehicle, and oftentimes per event, so a user should easily be able to trace a 'moment' in the statistics back to that same moment in the simulation.

6.2.2. Unexpected Almende output

A big challenge in the development phase of the product arose when the Almende output format and a sample output file were made available. At that point in time, the development phase had passed the halfway mark and the product started to come together, when run with the OpenRouteService algorithm. Up until that point it was always assumed, perhaps naively, that the Almende output would be very similar to that of OpenRouteService. However, it was quite different in its approach and much more oriented towards time steps, and less towards location. This would turn out to be a problem, both in the frontend and backend.

In the frontend, it became clear that in the Almende output there was no indicator of the vehicle location in between stops, and thus no route layout. The result was a collection of straight lines between markers that took no notion of the underlying map structure. This is quite an essential part of the visualization, so this needed to be fixed with a workaround.

The backend was an issue because OpenRouteService passes along a number of aggregated and calculated values that are gratefully passed along to and used in the frontend. Most of these aggregated values relied on distance, but also on past locations. It was therefore unfortunate that the Almende output (at that point) passed no indicator of distance in its output. To account for this (and the routing issue in the frontend), all intermediate segments between stops were passed to the OpenRouteService directions' API. This way, the coordinate values of the routes were now known and the distances of every segment. These could be aggregated into total distance values (per route).

It must be stressed that this is a workaround, and not an ideal solution, because there is no way of telling whether the routes that are displayed on screen are the actual chosen routes by the Almende algorithm. This relates to the issue described in the last row of Table 6.1. This visualization of the routes makes live interaction with the algorithm (as a possible future extension) very difficult to achieve because such an interaction would very often rely on the vehicles' locations and routes.

The issues as described above were a bit time-consuming. By the time the basics were fixed, there was also still the need to implement support for the added features that are unique for the Almende algorithm, like handovers and mode changes. It was perhaps an assessment error on the project group members' part that these events could easily be displayed with markers on the map. This was eventually not the case because more often than not, these newly added features would take place simultaneously/at the same location, so markers would simply overlap. To resolve this issue, the added events were made clear with the help of the animation, which is described in the next section. This, in turn, was time consuming as well. This meant that the last few weeks of development were spent making the basic visualization features work for the Almende algorithm and making the special Almende algorithm features work in the existing visualization. As a result, there was little time for perfecting other aspects of the final product and working more on 'Should have' and 'Could have' requirements. In hindsight, there should have been more communication with the client about the features and formats of the Almende algorithm early on, because this would have saved time and stress in the end. This is certainly one of the biggest learning/improvement points of this project and it will definitely be born in mind when working on other projects later on.

6.2.3. Animation playback

Following the GeoJSON format that was used for OpenRouteService and interacts well with Leaflet, the solutions provided by Almende were also translated to GeoJSON features. This meant that all events and locations (deliveries and depots, but also: handovers and mode changes) were passed along as 'Points' on the map. This way of encoding the output data is very location-oriented.

It was decided towards the end of the project, that the Almende-specific map features should be made visible in the animation. To make this work with already chosen and implemented moving ve-

hicles, there was no other way but to track the location of the moving marker and see whether it had passed a certain feature on the map. If this was indeed the case, it would fire an event like the change of the icon of the vehicle from a car to a robot, to indicate autonomous driving.

This implementation is considered a workaround and would probably not be the preferred option in hindsight. This feature was suggested by Almende in a later meeting, which is why we did not take any such feature in account up front. However, the feature is rather obvious, so arguing that we should have seen it coming is probably justified. The current solution should not be notable for most previews, however, it comes with minor shortcomings as we will discuss. This is due to the fact that the animation is re-enacting the traversal of the route and firing actions as it passes them, instead of traversing the events in the output. The big difference is that in the former implementation, a vehicle could possible 'miss' a delivery, due to perhaps a lag in the animation, or the delivery marker not being close enough to the route overlay (the latter is the biggest problem, which could simply be solved by keeping delivery points preferably close to, or exactly on a route). If this would be the case, the parcel counter of the vehicle would not be updated and the delivery wouldn't be marked 'delivered', while in the actual output this was the case.

It should be clear that these inconsistencies are undesirable. Unfortunately, this implementation in combination with the simulation manager's reset and rewind/fast forward options is not entirely predictable, as one might expect. Given the limitations of the options at the time, it all works as good as it possibly could.

In hindsight, and with the knowledge of the present, that specifically being that there would be a lot interaction with the moving marker, a different type of simulation might be a better choice. Perhaps one that would be more time-oriented, rather than location-oriented. However, choosing the time-oriented simulation would have its shortcomings as well when it comes to locations. The main (unsolvable) problem here is that delivery points are not actual locations on the route. No matter how you look at it, some kind of prediction has to be made as to which package belongs to which location on the route. One could argue that this is more of a OSM-related issue, as their delivery route output contains a list which does not contain the location you originally were planning the route for. Section 6.3 elaborates on this topic.

6.2.4. Aggregated statistics

Combining multiple runs of a given algorithm in a single overview is a nice feature to have, so that you can see the spread of your performance. Having a single well performing run, while others could on-average be terrible, is not valuable. One such option to visualize multiple runs of a given algorithm is done using boxplots. These boxplots are used on statistics that we already calculate for single algorithms; for example their total distance, total time and total costs. One feature that we do not yet support is the input of multiple runs for a single algorithm, since this would not at all be possible to visualize on the route map. However, the functionality to accept such an input exists in the boxplot overview. We then had a boxplot with a single statistic, which does not show anything at all. To temporarily resolve this issue, we took the single measurement and randomly added a value within a given range to test the boxplot functionality. This was repeated to obtain a random data distribution. So, it should be noted that the actual information in the current boxplots for multiple runs is not sensible at all; it is just a proof of concept. However, boxplots were also use for aggregate statistics within a single run. An example of this is the distance that delivery trucks drive. One could easily put all those distances of that single run into a boxplot to show the spread of the distances that the trucks travel. This is were the boxplots were also used and this implementation is complete and working as it should.

6.2.5. Additional features

Even though some requirements are not completely fulfilled, there are some aspects of the final product that are not present in the requirements, but are yet still added to the application, because the client thought of it during the development phase, or because team members decided themselves that it would have a lot of added value to include it.

The first and biggest feature is the inclusion of input settings. When looking at the list of requirements the notion of input is not mentioned anywhere. During developments and test runs, however, it was thought to be a nice added feature to be able to tweak the input settings, to run different scenarios and to evaluate the effects of changes.

Directly related to this, is the possibility to randomize delivery locations. With this feature, all algo-

rithm settings remain the same, they are just run on a different scenario. This was also considered a beneficial feature, especially to review the robustness of the algorithms under different inputs.

6.3. Recommendations

In the eleven weeks of this project, a lot of the Almende's requirements can be considered as done. Though, as time was limited, some features could be added to the visualization tool and some features could be improved. These features are discussed below.

6.3.1. Visited markers

A part of the simulation manager is the option to see when a vehicle has visited a marker (which represents a depot or delivery location). As remarked in subsection 4.1.1, the MovingMarker (which represents a vehicle) is not able to see whether a marker has been visited. This has to do with the fact that the marker is not exactly on the route, but only close to a route. The workaround to still be able to see if a marker is visited, has the drawback that at a high simulation speed, sometimes a marker is not noticed as visited. Also, when the time slider of the simulation manager is used to move to a previous point in time, the markers are not yet able to change into the unvisited state again. This could potentially be solved by linking a marker to a certain percentage of the route. As the MovingMarker does keep track of the fraction of the route that is visited, the markers would in that case be able to see if the vehicle already passed the percentage of the route it is located at.

6.3.2. Number of parcels

Besides the option to mark locations as visited, the simulation manager keeps track of the number of parcels in a vehicle and depot. Also this option lacks the ability to notice changes in time, caused by the time slider. As explained in the previous paragraph about 'Visited markers', also this problem could be solved by linking the amount of parcels in vehicles and depots to the percentage of the route that the vehicle is located at.

6.3.3. Autonomous roads

Another feature that could be improved, is the possibility to see where on the map autonomous roads are located. In the current implementation of Almende's algorithm, the vehicles are informed at which location they should change their mode (to either autonomous or manual). Initially however, it was assumed that roads would be marked as autonomous-enabled or not. For that reason, a button to mark all dutch highways was implemented. In order to give this button a more relevant function again, the parcel delivery algorithm should pass around all roads that are considered as autonomous.

6.3.4. Delivery point generation

In the final product, there is a button that randomizes delivery input points and reloads the algorithm with the new delivery points. This feature can be valuable to verify the behavior of algorithms in different scenarios. However, due to time issues, this functionality is only available for the OpenRouteService algorithm, and not for Almende. However, the Almende algorithm uses a csv file with nodes as input, and so it should be fairly easy to change delivery points to random nodes. It is recommended to implement this feature to make the application more complete and because it should be a small effort.

6.3.5. Could haves

As described in Table 6.1, there was not enough time to work on the could haves. They might however contain useful suggestions for further development, so they are repeated here.

When a visual representation of the algorithm alone is not structured enough anymore, it could be useful to make use of sounds when newsworthy events on the map happen.

The second could have is the option to interact with the live visualization. Blocking a road on the map, for instance, should make the vehicle find a new route. Or stopping a vehicle should cause the parcels to be handed over to another vehicle, so they can still be delivered. As explained before, this is not possible with the current version of Almende's algorithm, but it might be useful to change this in the future.

7

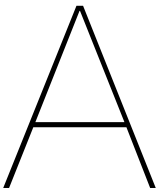
Conclusion

At the beginning of the project, Almende B.V. wished to have an application that would increase the value of their developing routing algorithm by making it more tangible and understandable for both developers and customers. This would be achieved by creating a visualization application that displays the algorithm's behavior and statistics. This visualization application is unique in the fact that it is fine-tuned for the hypothetical situation where the vehicle fleet is expanded with autonomous vehicles and in that it allows the comparison of such a situation to existing 'traditional' routing algorithms.

The main feature of the final product is its ability to display routes on a map, which includes delivery points and depots, but also a simulation where the different vehicles at play move over the routes and interact with events on the map. In addition, it displays multiple statistics with respect to cost, time and distance in a side-bar overview or dashboard. All these features are also accessible for two algorithms at the same time, to compare their strengths and weaknesses. Finally, the application allows interaction with the displayed routing algorithm (when it is deployed) by changing input settings.

In the later stages of development, it turned out to be hard to adapt certain implementations to the routing algorithm output provided by Almende B.V. The effect of this is mostly visible in the map view, and specifically the animation playback. While it works good by default, the final implementation can be unstable when used in combination with high playback speeds or the rewind/fast forward function. This is mostly due to the short time that was available for implementing this option, and so this is not something that cannot be improved further down the road, especially when the algorithm that is currently being developed by Almende B.V. is in its final, stable stages of development as well.

At the moment of writing, the web application is running on a public server and parts of the application have already been used by Almende B.V. to provide their client with more insight into the algorithm that is visualized. It can be concluded that, while the application still may need to be tweaked or revised in some aspects, the final product is in fact a user-friendly routing algorithm visualization application that includes all the main requirements.



Infosheet

The info sheet can be found on the next page.

Development of visualization software for parcel delivery algorithms

Name of the client organization: Almende B.V.
Date of the final presentation: July 1, 2020

Description

Almende B.V. is a research-oriented innovative company that is in a collaboration with I-Cave (Integrated Cooperative Automated VEhicles) to create an algorithm that schedules routes for parcel deliveries, in which novel features such as autonomous vehicles and package handovers are applied. The client has asked for an application that visualizes these novel aspects and the algorithm's robustness.

To get an idea of the available options and existing relevant solutions, the initial research focused on currently available (traditional) routing visualization applications, principles of data visualization to work by, and the available platforms for map visualizations and web applications.

In hindsight, the biggest challenge of building the application was the fact that the input and output of the novel algorithm diverged greatly from that of traditional scheduling algorithms and that these formats were only known to us in a late stadium of development.

Given the Covid-19 regulations that were in place for the duration of this project, all communication took place online. In the first few weeks we organized short, daily meetings to update each other, talk through issues and get to know each other as well. Next to that, we used the Scrum methodology to keep track of weekly deadlines and everyone's assigned issues to maintain the planning.

The final product is a web application that is optimized for large screens, but accessible on many different platforms and devices. The application relies mostly on the front-end (i.e. User Interface), which was tested continuously through user tests. Roughly 80% of the initial requirements have been fulfilled.

To extend the application in width and provide more comparison options, developers could add support for more alternative routing algorithms. To extend the application in depth, it would be especially interesting to be able to interact with the algorithm, by, for example blocking a road during playback and observing its effect on the routes. This would, however, first require the algorithm to be able to run live.

Team members

Luca Cras
Interests: Front-end design and development, Machine learning.
Contributions: Map, Vehicles, Autonomous roads, Design, Code Quality.

Simon Dahrs
Interests: Artificial intelligence, autonomous vehicles, backend development
Contributions: Map, Project setup, Deployment, Code Quality

Sander Gielisse
Interests: Backend development, statistics, automation, deep learning
Contributions: Initial design, properties box, charts, vehicle position calculations and checkpoint support, backend chart calculations

Lucile Nikkels
Interests: Programming, allround web development, user interaction
Contributions: Support Almende output, randomize orders, settings, track parcels in animation

Jesse Ruiter
Interests: Frontend development, human-computer interaction
Contributions: Support OpenRouteService, markers, vehicles, simulation manager

All team members contributed to preparing the research report, final report and the final project presentation.
Everyone took on a different role every week, ranging from scrum master, lead developer, lead tester and contact person to secretary.

Client:	Ir. C.A. Hermans	Almende B.V.
Coach:	Dr. M.T.J. Spaan	TU Delft (Algorithmics, Software Technology, EEMCS)
Contact:	J. Ruiter	`jesse.ruiter@hotmail.com`

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

B

Project Description

In this appendix the original project description is given. The content was copied from Project Forum. The project description or goal has not been altered since the start of the project.

Development of visualization software for parcel delivery algorithms

At Almende, our goal is to improve self-organization to better cope with an increasingly complicating world. Right now, we are specifically involved in a project investigating the benefits and applications of an autonomous vehicle fleet in parcel delivery.

One of our tasks in this projects is to illustrate the uses and implications of using autonomous vehicles, and particularly how it affects profit and complexity of the deliveries.

Currently, with a masters student, an algorithm is being developed for a use case where some roads may not be traversed by autonomous vehicles, and therefore we consider driver pickup and package handovers, which make this algorithm novel.

In the proposed bachelors thesis, the students are to develop an application that visualizes the benefit of this newly developed algorithm compared to currently used delivery scheduling algorithms. An example of how this should be done, is by looking at the saved costs and increased complexity, but also at stability of the solution whenever a driver is unavailable, and therefore the solution needs to change last-minute.

How this visualization is precisely implemented, is left for the project, and should be discussed along the way, as we are sure the students can come up with some creative and innovative visualization methods. We only necessarily require that this method is sufficiently adaptable to the algorithm, as it is not developed yet. I.e. the output of the algorithm should be fed into the visualization, and it should be able to do this for outputs for multiple algorithms for comparison.

Hence, the students are free to decide the framework they want to use for this visualization app. In any case, it has to be compatible with the (to be) developed algorithm and codebases on our end.

Almende B.V.

Our vision is that the ‘commons’ offered by modern technology will be better used when individuals participate to the ‘global brain’ by applying the principles of self-organization. Our mission is then to create innovative ICT solutions to empower human beings to better organize their lives in an increasingly complex world.

C

SIG feedback

Part of the TI3806 Bachelor End Project course by the TU Delft is the twofold submission of the code base to Software Improvement Group B.V. (SIG). This is a company that performs static code review to assess the code's maintainability based on the following properties: volume, duplication, unit size, unit complexity, unit interfacing, module coupling, component balance and component independence. The results of the first submission should be treated as feedback and the properties with low scores should be looked into for improvements. The second SIG evaluation than assesses whether improvements were made.

C.1. Feedback

After the first submission in week 6, the scores of the code for the different properties was as depicted in Figure C.1 below:

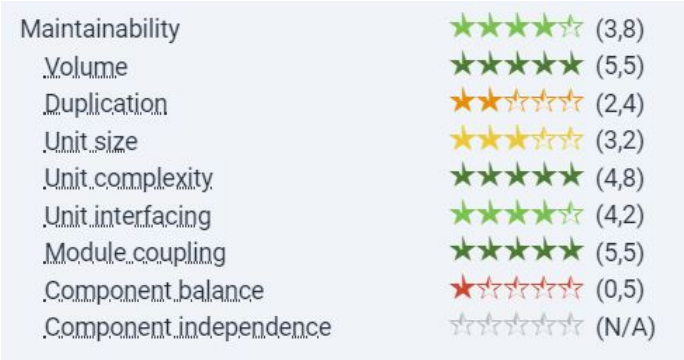


Figure C.1: Code maintainability scores provided by Software Improvement Group B.V. after the first submission.

Accompanying this code was a list of refactoring suggestions. The most important aspects of this are duplication and unit size, which reflects in the scores as well. In figure C.1, the property component balance has quite a low score, but this is not recommended to refactor, since it requires having to restructure and rewrite a big part of the whole code base, which is not desirable in late stages of development.

C.1.1. Duplication

Roughly 85% of all duplication errors originated from files that instantiated charts. This is due to the fact that at that point, a new file was created for every chart. This was done to pass along the right chart options and calculate the correct data values. However, this can definitely be rewritten so that all chart instances can be made with the same method. Given the amount of charts and their occasional complexity, this can be a lot of extra work and time, but the improvement is worth the effort.

C.1.2. Unit size

The unit size errors are more diverse and complex. For this category too, many errors are due to charts files (about 50%), however, the chart library that is used for this project (vue-chart-js) requires large data structures with lots of variables to initialize charts, which makes it hard to avoid writing large methods or structures.

The GeoJSON generation method in the back-end also scores poorly when it comes to unit size. This is due to a similar issue. The GeoJSON object is a very large structure, and during generation one needs to keep track of a number of variables. There will be an attempt at splitting the method up into more smaller methods, but it may be difficult due to the necessary incremental computations to gather aggregated values.

Finally, the MainFrame vue component has a too large data component. This will be hard to fix, since the MainFrame is used to tie the entire front-end application together and keep track of all its aspects. To resolve this issue, new components might have to be introduced and this may not be feasible.

C.2. Results

Two weeks after the first submission, the second submission had to handed in. The updated scores after this submission had been graded are depicted in Figure C.2 below:

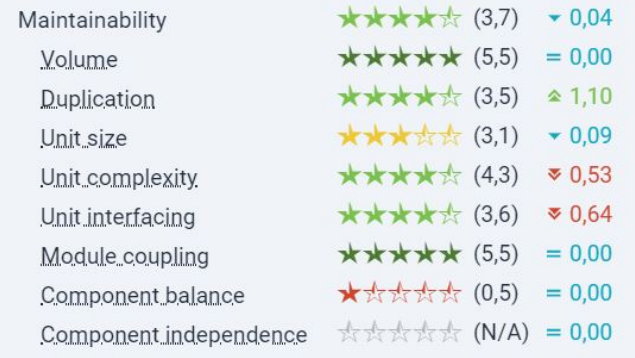


Figure C.2: Code maintainability scores provided by Software Improvement Group B.V. after the second submission.

Unfortunately, it is clear that, according to this feedback, there has not been a great maintainability improvement over the course of these two weeks. Subsection C.2.1 elaborates on the fact that improvements have been made, they are not as visible in the results as hoped. Subsection C.2.2 evaluates the feedback of the most changed properties.

C.2.1. Complexities

Initially, there would only have been one week between the first and last submission. If this were the case, the adaptations that were made after the first feedback would have been reflected more strongly in the second feedback. However, this was eventually not the case, and the number of last-minute adjustments and time pressure of the final sprint prevailed in week 8. The following is an explanation for how the code developed as it did.

As thoroughly explained in section 6.2, a large part of the code had to be adapted to the Almende algorithm's output in the last weeks of development. To be more precise, up until week 6 (the first SIG submissions) no work had been done on the Almende algorithm's output. Then, this quickly had to implemented, which turned out to be harder than expected. In agreement with the client, week 8 (the second SIG submission) was set to be the last sprint in which major modifications were to happen. As a result, there was a significant amount of time pressure on the final weeks, during which a lot of (complex) code modifications took place.

To summarize, this means that even though the feedback of the first SIG submission was taken up well and corrections and enhancements were made after this, they go slightly unnoticed after the addition of newly added complex code and workarounds.

C.2.2. Evaluation

In this section, the code quality properties that have undergone the biggest changes (or were expected to) are listed and details are given on how the feedback has been processed.

C.2.2.1. Duplication

Code duplication has improved greatly, from the first to the second submission. As mentioned in subsection C.1.1, the initial duplication errors originated mostly from instantiating chart, which was determined to be changed by replacing the old methods for new, generic methods. This plan was put into action and all chart files were removed and replaced by components that take computed data as input and shape this into a chart. The data for every chart is computed in new JavaScript files `chart_preprocess.js` and `chart_process.js`. The data structures for chart options are generated in a new file `chart_options.js`. The remaining duplication errors still mostly originate from chart files, but this level of duplication is unavoidable when using a library that requires input in a certain format.

C.2.2.2. Unit size

Unit size was the second lowest score according to the first feedback and the lowest score in the second feedback. Subsection C.1.2 already hints at the possible complications with resolving unit size issues for data structures in the frontend and the GeoJSON in the backend. During feedback processing, the issues around creating a GeoJSON structure in `map.py` were resolved. Even the creation of a GeoJSON for the Almende algorithm, which requires a very large amount of aggregate values due to its different structure, was managed to stay relatively small, at a small cost in unit interfacing and complexity.

In fact, all unit size issues that were not contained in a data object, but in functional code, that were pointed out by the first SIG feedback are processed and not recurring in the problematic files of the second feedback. However, the number of violations due to data objects being too large has increased in the second feedback. These data structures are static JSON objects that are defined in a Vue component instead of a separate JSON file, while if they were not, they would not have formed a problem. Even though having these data structures is not desirable, minimizing their size is not a priority compared to keeping functional code small and simple.

The biggest code violations in unit size comes from the file `MovingMarker.vue`. Unfortunately, this is due to issues described in subsection C.2.1 and section 6.2, where it is explained that with the implementation of the Almende algorithm, a lot of new features had to contained in the animation (moving markers). In hindsight, the `MovingMarker.vue` file could have been split up in separate JavaScript and Vue files to reduce its individual file's size.

C.2.2.3. Unit complexity

While unit complexity received a high score in the first feedback, this score is slightly decreased in the second feedback. The feedback of the first submission provided 0 violations to improve. The second feedback indicates that the function `iterate_actions` in the file `map.py` violates the unit complexity quality. This is one of the functions that was added to accommodate the creation of the Almende algorithm GeoJSON object. This method contains a very large if-else statement inside a for-loop. The use of this structure was unavoidable, due to the format of the provided output file to turn in to a GeoJSON object. This is because all relevant event are contained in a list of actions, which should be traversed chronologically to maintain a log of past waypoints and route stops, and there are 6 different types of actions. The if-else structure functions as a type of switch statement over the action type, in the absence of a proper switch statement in Python. Implementing this function in another way would likely result in one or more functions that would feel counter-intuitive and would perhaps be more 'complex' for the client to understand and continue building upon.

C.2.2.4. Unit interfacing

After the first submission, unit interfacing scored a 4.2, which gave not too much reason for modifications. Nevertheless, four out of the five methods that were marked as violations, did not recur in the second submission's results because they had been altered. Unfortunately, in return, there were two new violations in the second submission's feedback: the function `iterate_actions` and the function `add_delivery` in the file `map.py`. These functions were both added to accommodate the conversion

from the Almende output to GeoJSON. As mentioned in the previous sections, a lot of variables had to be maintained in order to gather the same aggregate values as OpenRouteService. This fact makes it hard to optimize unit size, unit complexity and unit interfacing all at once. When splitting up a large method to decrease size and complexity, there is a trade-off in the form of unit interfacing, because it requires passing many values to the submethods. Overall, the trade-off is weighed out quite fairly and in neither of the three categories are the violations very serious, i.e. a high value according to SIG.

One method that was the top violation in the first and second SIG results is the method `generateChartOptionsLine` in the file `chart_options.js` (in the final version). Once again, one can speak of a trade-off between unit interfacing and unit size, but also duplication. In order to standardize the chart option generation methods and reduce duplication, a single method was made per chart type that would generate the options data object based on the parameters supplied. As a consequence, and due to the chart plugin requiring numerous variables, this method needed to suffer in number of parameters. Just like in the first example, this trade-off is well justified and the violation is, again, not very serious.

C.3. Conclusion

The process of incorporating the initial feedback provided by SIG came with some obstacles, but by balancing a number of trade-offs, the final result is up to the standards that can be expected. The product received a maintainability score of 3.7, which is a good score, especially considering the circumstances explained in subsection C.2.1, and the fact that static code quality evaluation, while very worthy, sometimes does not show the complete picture. Sometimes it may be more valuable to sacrifice a tiny bit in one area to gain a lot in other, when also considering code readability and extensibility. Overall, the project team has confidence in the final product and its code quality and it is satisfying to have that reflected in the final maintainability grade.

D

Research Report

In this appendix the content of the Research Report is given. Originally the Research Report also contained a cover page, a table of content and a bibliography. These components are omitted because they are redundant or already included in another part of this final report.

D.1. Introduction

In this chapter, the project will be introduced. The goal of the project is to develop and deploy a visualization application that will allow the client, Almende B.V., to gain more insight into the workings and benefits of a new algorithm that they are designing. This is a parcel delivery algorithm that focuses on improving the cost and efficiency through the use of handovers and autonomous vehicles, while also maintaining robustness against a variety of perturbations. To understand the performance of this algorithm, however, a tool is required to visualize all the data and output of the algorithm in question such that the information can be presented in a less abstract way. This also enables Almende to compare similar algorithms against each other on a variety of metrics. The same visualization tool might then also be used for another project that they are working on.

Visualization of parcel delivery systems can be done in a variety of ways, most importantly through the use of a map. Packages, drivers and roads can be visualized on this map in a realistic manner and choices made by the simulated algorithm can clearly be seen. Another important tool in visualization is the use of charts to enable the presentation of key statistics in a brief and clear way.

In this report, the best means to visualize the algorithm that Almende is developing will be researched and the best options to develop the software required will be weighed up. We will do this considering the requirements and wishes set out by the client.

D.2. Problem analysis

The company Almende is developing an algorithm for a use case where some roads may or may not be traversed by autonomous vehicles. Therefore driver pickups and package handovers are considered, which make this algorithm novel. The purpose of this project is to develop an application that visualizes the benefits of this newly developed algorithm compared to currently used delivery scheduling algorithms. As the output data of the parcel delivery algorithm may be hard to analyze or present to customers, visualization plays a critical role in the development of this parcel delivery algorithm.

The features listed below are so-called ‘must-have’ and ‘should-have’ features requested by the client.

- The visualization is accessible in a web browser.
- The visualization can be used to compare different parcel delivery algorithms.
- The visualization shows the routes of delivery vehicles, the vehicle type, packages, package handovers, pick-up points and new orders.

- Important metrics of the algorithm are visualized in some type of chart form.
- Algorithm properties are displayed for the algorithm's comprehensibility.
- The code for the visualization algorithm is easily adaptable.
- A detailed overview of orders and routes is available in chart or list form.
- The visualization is scalable to different numbers of orders and geographical areas.
- The visualization provides the option to perform random simulation for the same algorithm.
- The visualization allows playback and speed control.
- The visualization algorithm accumulates statistical data on the parcel delivery algorithm.

From the list above, a number of research questions regarding the visualization problem can be deduced. First of all, when thinking about the visualization end-product, one could ask themselves how different entities and events should be portrayed such that the visualization is effective and user-friendly to all user types? Also, and more concretely, what are suitable charts or graphics to visualize algorithm metrics, properties and statistics? When looking at the problem from a practical developer's perspective, it would be beneficial to look into suitable platforms on which to build the visualization, and compare their characteristics with respect to for example scalability. Perhaps there are existing visualization algorithms that are created for similar projects. It is interesting to look into how they measure their effectiveness, cost and robustness, and also with what platforms and tools they did their development. Finally, to live up to the requirements, some in-depth analysis of the parcel delivery system of Almende will need to be done. In this analysis it should be investigated how their algorithm differs from other parcel delivery systems, how this can be emphasized in a visualization, and by what metrics the robustness of this new algorithm can be measured.

To get insight into these problems and subjects and to find possible ways of dealing with them, it is beneficial to conduct the necessary research in these areas. With the help of this research, well-informed preliminary design decisions can be made, as well as educated decision on the spot during the development process. The research on these subjects can be found in the next chapters.

D.3. Data visualization

This chapter will elaborate on different aspects of data visualization. These include its definition, which can be found in the first section, the principles of designing data visualizations in the second section, and the project-specific case of data visualization on maps can be found in the third section.

D.3.1. Definitions

The definitions of data visualization according to different sources is as follows:

- "Data visualization is the graphic representation of data. It involves producing images that communicate relationships among the represented data to viewers of the images. This communication is achieved through the use of a systematic mapping between graphic marks and data values in the creation of the visualization. This mapping establishes how data values will be represented visually, determining how and to what extent a property of a graphic mark, such as size or color, will change to reflect changes in the value of a datum." (Wikipedia, 2020)
- "We can look at data but we cannot really see it without the context of relationships that help us compare and contrast them effectively with other values. To derive understanding from data, we need to see it represented in a different, visual form. This is the act of data representation. [...] Representation choices concern the form in which your data will be visually portrayed." (Kirk, 2016)
- "The role of visualization systems is to provide visual representations of datasets that help people carry out tasks more effectively. A visualisation should save time, have a clear purpose, include only the relevant content and encode data/information appropriately." (Munzner, 2014)

By comparing these definitions and their most important aspects, data visualization has been defined, and will be applied in this report, as follows: Data visualization concerns the mapping of data attributes that are relevant to the user into a visual shape or form that reflects the datum's true value and is comprehensible and pleasant to work with as a user.

D.3.2. Principles of data visualization

In order to create and define a 'good' data visualization, we can use the following three questions. Each question forces a designer to think about relevant aspects of the visualization (Munzner, 2014).

What is being visualized?

According to Senay and Ignatius (1990), one of the most important issues in data visualization is choosing the right mapping of data attributes into a visual. This can be more difficult than it sounds, because there are usually hundreds of possible mappings which may lead to different visualization technique designs. Selecting and creating the most effective design among all the alternatives for a given situation usually requires considerable knowledge and creativity on the part of the visualization technique designer. So, a designer needs to identify and justify its mappings, possibly by showing alternative (bad) solutions or writing textual motivation to show that there has been thought of a number of different angles and why the chosen option is the best. The concrete answer to the question of this paragraph should then be what the major data types, attributes and their classifications are. (Munzner, 2014)

Why is it being visualized?

Every aspect of a data visualization should have a purpose. One should be able to answer the questions "why do the users need this?" and "what do the users need to be able to do with it?" (Munzner, 2014). As stated by Kirk (2016), as well, this means that data visualizations should not contain unnecessary details, because the user does not need it. Also, data that is necessary for context, but not a key aspect should be represented accordingly; conversely put, if something looks significant, it should be, otherwise the visual is misleading. Kirk (2016), has identified numerous more audience qualifiers and aspects to consider when making a data visualization: the user's level of knowledge on the subject; their interest in the subject; their level of engagement with the subject matter; their knowledge of the visualization type (e.g. graph, pie chart, map); and the time and pressure they are under to obtain the knowledge in the visualization.

In the use case where there is a single audience type, this type can be put on the scale of each qualifier, and the visualization should be created by the rules of this scaling. In the use case where the visualization needs to be used by different types of audiences, a designer would have to think about the differences between these user types and how to make the visualization user-friendly for each. In the example of the parcel delivery algorithm, the algorithm creator probably wants to see their algorithm act in detail, visualizing every step it takes to verify its behavior. On the other hand, a customer looking to invest in the algorithm wants to see it work on a large scale, with indicators of the overall, maybe average, benefits of the algorithm. To make both audience types happy, a designer has to introduce a level of scalability on, probably, each of the audience qualifiers identified by Kirk (2016). How this will be achieved, is described in subsection D.6.1.

How can it be visualized?

At this point it is already established *what* data aspects we want to portray and *why* the user wants to see this data. Next up is to think about how these ideas will take form. To start, the data has to be encoded in a certain way. According to Munzner (2014), this can be done by either arranging the data in a meaningful way, or by mapping the data to another form of representation. Encoding data in the right way is important because it determines how a user decodes the data and thus how a user 'sees' the data. From this perspective, the user will draw their conclusions, so a designer should aim to create an encoding that minimizes the decoding error. A designer has to understand that some visual channels are more effective for some data types than others, and some data has a natural mapping that our brains expect given certain types of data. The effectiveness of different data representations, quantified by decoding error, has been researched in a crowdsourced study by Heer and Bostock (2010). Their findings included that charts that use position (i.e. different forms of bar charts) perform the best, charts that use angles (i.e. pie charts) perform second best and charts that use areas (i.e. bubble diagrams) perform the worst. In addition, they found that to minimize decoding error, chart heights should be more

than 40 pixels high and should include grid lines. Increasing chart height above 80 pixels, however, has limited beneficial effect.

Encoding data Data encoding can be achieved through arranging, mapping and a combination of the two. When arranging, you could choose to arrange the data in, for example, graph plots or put them as points on a map. When mapping qualitative data (e.g. different types of vehicles) to a visual aspect, this can be achieved by applying different hues of color or different textures or shapes. When mapping quantitative data (e.g. the number of parcel orders) to a visual aspect, one could use saturation, luminance or transparency of a color, or the position, size or angle of a pointer, or through some motion, for example in a certain direction, with a certain speed, or with some frequency.

To continue, when applying one or more of these techniques, there are a number of issues that a designer should be aware of. Firstly, the so-called pop-out effect. This has to do with the fact that humans subconsciously scan an image in a matter of milliseconds. In this time, they can immediately identify objects that stand out if they are different from the rest. For instance, in a scatter plot with ten blue dots and one red dot, the attention is immediately drawn to the red dot. A designer that wants to make use of this should be aware that the pop-out effect does not work for all visual differences between objects; color works the best for this. This effect can also happen unintentionally, hereby perhaps drawing unnecessary attention to some data value that in fact has an equally important meaning as that of other colors. To resolve such an issue, a designer should think about using a tested color map, like the HSL linear L rainbow palette. It was designed by Kindlmann et al. (2002), and is widely used, for example in `matplotlib`.

Another thing to keep in mind when working with colors is that in many use cases it is hard to determine semantically sound colors to represent a data value. Therefore it can easily become unclear which color belongs to which value, especially when there are many different values. A designer should think about how to make the translation from color to value easy (or represent the value with a shape instead), so the user does not have to repeatedly look it up in the legend. This reduces the decoding time and the cognitive load on the user. One final consideration is that as a rule, 2D visuals are better than 3D visuals. The visual system of humans is not very good at perceiving information in 'depth' and so, while it may look appealing, 3D data visualization almost always lead to a higher decoding error. (Munzner, 2014)

D.3.3. Data visualization on maps

While the principles of the previous section are applicable to all fields and methods of data representation, it is interesting to look further into some considerations about visualizing data on maps, since this is a large part of the project on visualizing a parcel delivery algorithm. Some basic, yet important guidelines for map visualizations are as follows. First of all, one should create or use an easily understandable basic map vocabulary: title, legend, baselayer, marker, popup, zoom level, polygon, polyline and source should all be present, and as self-explanatory as possible. Secondly, one should add source credits and bylines to build credibility and accountability (Wong, 2010). Thirdly, one should use colors to logically organize the data. Avoid random colors and color combinations from the opposite side of the color wheel (by Isaac Newton). Use contrasting colors to call attention to important data (Knaflic, 2015). This remark is in line with the findings in subsection D.3.2. Perhaps the notes on the use of colors are even more significant when using maps, because the map is essentially the data's 'background' and so the data's message should be clear even through the distractions of the basemap's colors. This leads to the following point of advice: one should choose the basemap wisely. Think about whether it is desirable that the basemap contains a lot of information or not. Basemaps may use designs or colors that can be distracting to the user. A designer should think about the minimum number of elements required in the basemap to bring across the message of the data (Dougherty and Ilyankou, 2020).

The next chapter will dive deeper into map visualization and the pros and cons of different map APIs.

D.4. Visualization platforms

In this chapter different visualization platforms are discussed. The first section is about the requirements of the platforms in our project. Secondly, an overview of potential platform providers is given. The last

section gives insight in the reasons to choose certain platforms over others.

D.4.1. Considerations

From the previous chapters it should be clear now that the goal of this project is to visualize parcel delivery algorithms. In order to do that in the best possible way, there are several conditions to the platforms that will be used.

In the first place, the visualization should be scalable in a way that both usual and autonomous deliveries can be visualized. One of the reasons why the parcel delivery algorithm of Almende is better than existing ones, is because it takes into account the restrictions and possibilities of autonomous vehicles. Therefore this distinction between vehicles should also be visible on a map.

Another way of interpreting scalability in our project is the desired possibility to visualize multiple pickup & delivery algorithms. In this way the tradeoffs of different algorithms can be shown. This is an important criterion, as one of the goals of this project is to be able to visually show to a customer why the new developed algorithm of Almende is better than existing ones.

A third requirement would be the ability to visualize everything in an uncluttered way. One can imagine that it is doable to show several parcels that have to be delivered by one driver. However, an algorithm could also output loads of parcels, that all have to be visible on a map. In order to keep an overview, an application like Google Maps will probably cluster a large number of elements. Though this will make it harder to see the working of an algorithm.

The last requirement on the platforms that are going to form the base of our visualization tool, is that they are time and cost efficient. As the application might need to visualize outputted algorithm data real-time, the platforms should not be the bottleneck in this visualization. For that reason, the tools that are picked in the next section are filtered by their ability to handle large amounts of data with the least amount of overhead.

D.4.2. Platform providers

In order to successfully develop an application that visualizes parcel delivery algorithms, several existing platforms will be used. In this section the most suitable platforms for both map data visualization and web application platforms will be considered.

D.4.2.1. Map data visualization

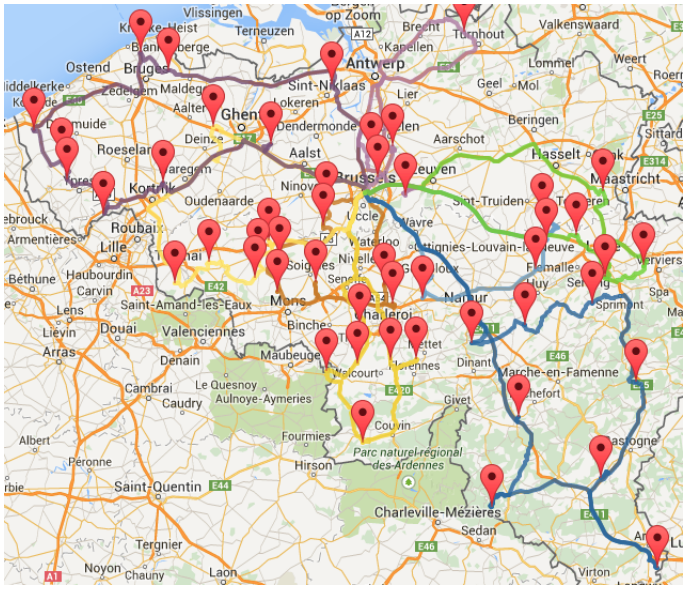
A large part of the comparison between parcel delivery algorithms will have to be shown on a map. Our search for map providers has shown that there are big differences in e.g. costs or adaptability. This results of that search are written below.

Google Maps (`https://cloud.google.com/maps-platform`)
When thinking about visualizing parcel delivery algorithms, one of the first things that comes to mind is Google Maps. An existing tool that makes routes, using Google Maps, is OptaPlanner[1]. We will not go into the details on how the underlying algorithm works, as the algorithm should be completely interchangeable for our use case. In Figure D.1a, a map of Belgium is used as an example. In this map, all depot locations are shown by red markers while all the routes taken by each individual truck are shown in different colors.

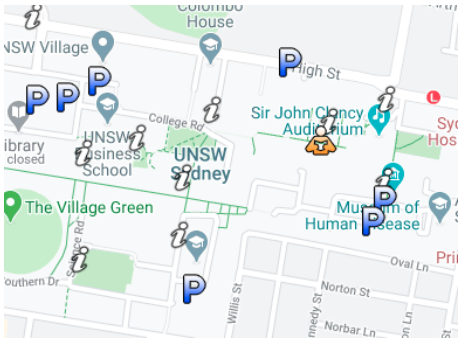
Even though this example only uses inbuilt markers, Google Maps has the feature to allow custom markers to be placed, as shown in Figure D.1b. This makes Google Maps a great candidate for our map visualization in case we were to work on real world scenarios, as we would be free to mark routes, delivery/pickup locations and swapping locations exactly in the way that is desired by the product owner. However, building things like multiple different panels and statistics that differ based on the selection is something that would be very hard when using a ready for use package like Google Maps. However, it is important to note that Google Maps is not free of charge. In case a completely proprietary visualization tool is desired, using third party non-free data suppliers will not be possible.

OpenStreetMap (`https://openstreetmap.org`)
Another well-known platform for maps is OpenStreetMap (OSM). A huge advantage of this platform over e.g. Google Maps is that it is open source. Not only in terms of costs this is beneficial, but also

[1] `https://www.optaplanner.org`



(a) Route visualization as done by OptaPlanner from:
https://optaplanner.org/blog/2015/03/10/VisualizingVehicleRoutingWithLeafletAndGoogleMaps.html



(b) Custom markers as done by Google from:
https://developers.google.com/maps/documentation/javascript/custom-markers

Figure D.1: Visualization in Google Maps

in terms of keeping management over the visualized data on the map. Besides, we can guaranty the performance of our application, instead of being dependent of the service of a platform provider. OpenStreetMap has multiple plugins to visualize markers and lines on a map. OpenLayers and Leaflet are two examples of JavaScript libraries that would be very useful to display delivery vehicles and destinations. A routing plugin like OpenRouteservice could be used on top of OSM to compute the best route from the parcel distribution center to the destination.

MapBox (https://mapbox.com)
Just like Google Maps, MapBox offers a platform for developers to create applications that solve problems with maps, data and spatial analysis. The maps are based on data of OpenStreetMap. An advantage Mapbox is that it is a cheaper alternative than Google Maps. But because it is used less often, there are less examples available to see its capabilities.

Carto (https://carto.com)
The last map data visualization platform is Carto. Like the ones above, this platform offers an API to turn location data into a clear overview on a map. It offers a free year to try (limited) options of the platform. For the time of our project this could work, but in the long run it could be one of the most expensive options. Another drawback of Carto is the limited amount of available examples, as, like MapBox, it isn't used that often for public projects.

D.4.2.2. Web front-end

For any integrated, complicated and modern web application, it is advisable and common practice to make use of a JavaScript framework to support the development process. One of the requirements of the visualization for the parcel delivery algorithm is to have it run in a web browser, therefore the visualization application will be developed in JavaScript, using a JavaScript framework. Different frameworks come with different features and compatibility, which is why their pros and cons, with regard to the visualization problem at hand, are researched below.

Angular (https://angular.io)
Angular is an open source JavaScript framework operated by Google. One of Angular's key selling points is its support for Single Page Applications (SPA's). The visualization for the parcel delivery system, as most visualizations, will appear on a single page and update on the same page, which

means it can be categorized as an SPA. Another beneficial feature is that it has two-way data binding implemented, which, in practice, means that when a value in the data store updates, the UI updates as well. On the downside, Angular is said to lack somewhat in modularity and flexibility. Additionally, Angular requires its users to always specify the controllers of view-models. Finally, Angular was one of the earliest developed JavaScript frameworks, hereby establishing a great benchmark and proving to be reliable, but this also leads to Angular being relatively harder to learn and more heavyweight than some of its alternatives. (Wohlgethan, 2018)

Vue.js (`https://vuejs.org`)
Just like Angular, Vue.js has great support for the creation of Single Page Applications. This is due to the fact that Components and Views are smaller interactive parts of an app that can be easily integrated into the existing infrastructure, with no negative effect on the entire application. This also allows for easy code reusability. While offering powerful data-binding as well, Vue.js is considerably more lightweight than Angular and provides users with more flexibility (Boczkowski and Pańczyk, 2020). Vue.js is also known to be intuitive and quite easy to learn. One of its disadvantages is the fact that it is a relatively new framework that is evolving quite fast, which could possibly make it hard to find documentation or solutions to coding problems. Additionally, there are developers that have suggested that Vue.js provides too much flexibility for working with complex projects, although this will likely not be a problem with the parcel delivery algorithm visualization project. (Wohlgethan, 2018)

React (`https://reactjs.org`)
React is characterized by its support for dynamic user interfaces with high incoming traffic. It works well for such cases because while other frameworks update the entire DOM on any update, React updates the virtual DOM, with which it is possible to update only the changed value. This makes updates really quick and enables developers to work with UI objects faster and apply changes in real-time. Additionally, just like Vue.js, React allows the reuse of code components (Boczkowski and Pańczyk, 2020). On the downside, React is continuously developing and changing features. This requires programmers to keep up or dive deep into the matter to look up functionalities that may have changed over time. A problem that comes with this is poor documentation; React does not have the reputation that its creators take the time to write proper instructions, which can be cumbersome especially when learning for the first time (Duvander and Romhagen, 2019).

D.4.2.3. Web back-end
Of course, the application will need a server to perform the calculations required for the visualization. This also shields the user from knowing the inner workings of the tool. For the language that will be used in the back-end, Python has been chosen. There are a few reasons for this choice. Namely, Python has very strong libraries that will be very useful for data manipulation and calculations. Python is also very easy to use and pretty fast. Almende also has experience with Python in-house, therefore it will be easier for them to adapt and amend the application later on. Having chosen python, this leaves us with the choice of framework. There are three frameworks in particular that will be looked at. Django, Flask & FastAPI.

Django (`https://djangoproject.com`)
Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Django also makes most choices required for the developer to start working. This enforces a very strict standard but is very useful for very large projects where a good structure in the code can save the developer a lot of work. Django has many plug-ins developed by its community providing extra features, such as web-shop functionality. This will most likely not be necessary but is worthy of a mention.

Flask (`https://flask.palletsprojects.com/`)
Flask takes on a different approach to the python web framework. Flask is a micro web framework, meaning that it is very lightweight compared to Django and it lets the developer make all the choices required to start development. It takes a more hands-off approach but is also a lot more appropriate for smaller projects where the developer doesn't need all the functionality that Django comes with out

of the box. Flask is therefore also a lot easier to start with and learn, than Django which has a high learning curve but also a lot of power.

FastAPI (`https://fastapi.tiangolo.com/`)
The strength of FastAPI is that it strictly defines the input and output data. It uses Pydantic[2] to do so. The main advantage of that is that you can auto-generate API documentation that conforms to the OpenAPI standard. The second advantage of it is that input and output validation is done automatically by the framework. The final advantage is that you can use auto-generated client code. This makes it easy to build multiple clients for the same API. On top of this, FastAPI takes a lot of cues from the likes of Flask and is therefore also, as it name also states, very fast but also very light weight.

D.4.3. Motivation

Now that the considerations and potential platform providers are described, this section will present the motivation why we chose for certain platforms.

Map data visualization
After having seen four platforms to visualize map data (in subsubsection D.4.2.1), it is time to conclude which of these platforms will be used in our project. It was not very hard to make the decision to use OpenStreetMap for our project. To display the OpenStreetMap data in the front-end, we will be using a JavaScript library called Leaflet, which is lightweight and has all the mapping capabilities that will be needed, such as markers and road highlighting. Leaflet works well with Vue.js, because it is also written in JavaScript. It will also work well with our back-end, because a Python library called Folium has good interaction with Leaflet, enabling us to do data manipulation and route planning in Python and then mapping this to the Leaflet map. OpenStreetMap has several more advantages as well. In the first place this has to do with the large amount of examples of similar projects. As none of the group members are familiar with data visualization on maps, it is very useful to be able to have a look at examples and learn how other developers used the platform. Secondly, in contrast with the other platforms, OpenStreetMap is open source. That provides the advantages that it is free and the data is kept in-house. Also due to the platform being open source, there have been developed a lot of open source 'plugins', which for example makes it possible for us to use Leaflet to make the development easier. So we are not limited to the functions that are given by a platform provider like Google Maps. Also OpenStreetMap is not an API, and therefore its service does not determine the performance of our application, as we run the version ourselves with the amount of resources we deem necessary with less delay. Lastly, we want our findings to be easily reproducible and using an open source platform makes this much easier.

Web front-end
When comparing Angular, Vue.js and React as possible JavaScript Frameworks for the project at hand, two important aspects must be thought of: that of the user and that of the designer. Firstly, the user aspect: is the resulting UI from using a framework fast, user-friendly and does it provide good interfaces for data visualization (i.e. charts, graphs and animations)? React, due to its virtual DOM, definitely has the fastest user interface. However, the parcel delivery algorithm does not necessarily need this super fast interactivity because it does not have to deal with huge amounts of user input. When it comes to user-friendliness and data visualization, all frameworks have the ability to present equally presentable results. All three frameworks have built-in data visualization tools and, in addition, there are many data visualization libraries that are compatible with all these frameworks. Concluding, the user aspect comparative factors are not decisive on which JavaScript framework to choose, which means the designer aspect will be the deciding factor.

When it comes to the designer aspect of working with each of the aforementioned frameworks, one specifically has to look at its flexibility, its size, the amount of documentation available and the amount of experience of the designer or how difficult it is to learn (Duvander and Romhagen, 2019). Concluding from the findings in subsubsection D.4.2.2, Vue.js is the most flexible framework of the three, due to it allowing code reusability and allowing the designer to define the application structure. When considering size, Vue.js is the most lightweight framework. When it comes to documentation,

[2] `https://pydantic-docs.helpmanual.io/`

Angular has the best amenities, due to its long existence and it being owned by Google. Finally, when looking at the required knowledge level for the three frameworks, it looks like Vue.js is the easiest to learn, due to it being intuitive and relatively small. Additionally, one of the project members of this visualization project has experience working with Vue.js, which is considered a big advantage, given that the project is off a short duration and so it is not desirable to spend a large amount of time on setting up the framework and learning about it. When weighing up all these factors, it can be concluded that Vue.js is the most suitable for this project and thus it is decided to work with Vue.js as JavaScript framework.

Web back-end

After having taken a look at the different popular frameworks for the back-end of the web application. The advantages and disadvantages of each become clear. Ultimately, the choice has been made to go with FastAPI for the back-end. In this section, the reasons for this choice will be presented. But let's start with why Django was not picked. The main reason for this is that it is too bulky and not really suitable for this project. Django is better for bigger projects with huge amounts of requirements and functionality. It comes with too many presets that are not necessary. After eliminating Django, the choice was then between Flask and FastAPI. We chose for FastAPI, because it has many of the advantages of Flask and more. It allows fir client code generation and documentation generation via the OpenAPI standard, which will enable us to develop quicker and focus on the front-end and calculations part more.

D.5. Related work

In this chapter, it will be researched what related work has been done and in which way this visualization project could improve on existing products to come up with a result that suffices the requirements as provided in the requirements section. It is surprising how many pickup and delivery algorithm papers are published, without having any visuals to back their findings. Mostly, numbers and statistical data are used to support the claims.

An exception to this is the research done by Hoen et al. (2004). Even though this research paper does not have any visuals in it, it contains a clear description of how visuals were used. They used a panel based overview, where each panel represents different statistics or visuals. The main panel in the middle contained the underlying grid (road map) with the depots and trucks. Since this research by Hoen et al. (2004) was done for logistic management in transportation, no such thing as customer delivery locations are shown on the map. If this overview were to be adopted in this project, the map would also include delivery locations, as well as the locations where two trucks swap packages, as this was also one of the features of the underlying algorithm.

In this same research, the side panels of the overview are used to display information about the currently selected truck or depot. This would apply perfectly to our use case; being able to select a single pickup location, parcel truck, swap location or delivery location and show any related statistics, measurements and other data for a single given selection could be useful when trying to analyze the given situation.

Using different colors for the route driven by each truck would also help to improve the visualization of the complexity of the underlying algorithm. Being able to show visually that a given truck drove a lot less using a given new algorithm compared to a given existing algorithm, could be a valuable property of our visualization tool when presenting the newly developed algorithm to a client.

This research by Hoen et al. (2004) goes out from three main principles for visualisation, which would very well apply to our visualizations as well. This is why the following will be the foundation of the development of our visualization: "Present all information on a single graphical interface, provide the user with the ability to easily navigate through the simulation, with complete information and intermediate results. The information given in the visualization should be palatable: it can be understood without delving in the underlying complex semantics of the model."

D.6. Visualizing the parcel delivery algorithm

In this chapter, some extra choices for visualization are made that are specific to the use case of the algorithm that is being developed at Almende. Also, the research findings of the previous chapters will

be weighted up against each other and used to find a fitting solution for this parcel delivery algorithm visualization.

D.6.1. Scalability

To elaborate on the different user types mentioned in subsection D.3.2 (the algorithm developer and the potential investor), it would be helpful to dive into these two user types for the parcel delivery algorithm by Almende. The initial client is the person who is developing this algorithm. They want to be able to see the steps that it takes and the decisions that it makes, especially with regards to its novel aspects, like autonomous vehicles and package handovers. To classify this user under the scale made by Kirk (2016), the user has extensive knowledge on the subject, is extremely engaged with it, has probably moderate experience with maps and charts and they have plenty of time to investigate the visualization. Based on this, the visualization may, and probably should, contain a lot of technical details on what is displayed. It also should be on quite a low level: this user is probably most interested in watching two up to five vehicles operate in detail.

Looking at the other user type, a possible investor or buyer of the algorithm, this view changes a bit. Their knowledge of the underlying algorithm is superficial and how it operates is not a primary interest of them. Their knowledge of maps and charts is probably moderate as well and, finally, they want to see quickly, without a lot of trouble, whether this algorithm is beneficial in terms of cost. Clearly, this user is more interested in the overall metrics. Showing this user that the algorithm is very cost-efficient and robust under all circumstances has more positive influence than showing that parcels may or may not be handed over between vehicles.

To tie this all together, the visualization should be scalable on two fronts. Firstly, the map visualization. To make both users satisfied, one could implement the visualization of the details, like different vehicle-types and handovers, to be shown when the user is zoomed in and has some maximum number of vehicles on its screen. When the user zooms out, the vehicles that are close together should be clustered and represented as a number, hereby presenting a clear map overview and moving the focus more to the dashboard. The dashboard is the second front on which a level of scalability should be introduced. A broad range of data visualization should be visible in the dashboard (for the algorithm developer) but there should also be the option to fold each visualization, hereby making the dashboard uncluttered and focussing on the metrics that are important to the customer.

D.6.2. Visualization of autonomous vehicles

There are a few complications to the standard visualization that arise when autonomous vehicles are to be considered. The first of which is differentiating the vehicles shown on the map to inform the user of which car is of what type. This, however, is not that hard to do and the way that it will be done is by giving the cars a different look based on what type it is. The second complication ties in to the fact that these autonomous vehicles will not be able to drive on certain roads.

The first question that then needs to be asked is how are these roads chosen. Because there aren't roads that are especially designated, by the government for example, to be forbidden for autonomous cars. But, it is well known that self-driving cars function more easily on more predictable roads, such as highways, and much less reliably on small, curvy roads without much guidance in terms of stripes and guidance. Which road will and will not be driven by autonomous vehicles depends mostly on the algorithm of Almende, but for other algorithms and for testing purposes, highways and larger streets are the only streets that autonomous vehicles can drive on.

The second question that then needs to be answered is how are these different roads visualized and distinguishable from each other. To differentiate various kinds of roads, roads that allow self-driving cars will be thicker and of a different color than the roads where autonomous cars cannot drive. However, thickness should be used with caution since a large number of thick colored roads could result in a blur rather than a nice overview.

D.6.3. Robustness visualization

The robustness visualization involves a difficult part of our research and decisions. Even though robustness feels like something that should be perfectly visualized graphically, finding the correct visual representations is yet another challenge. Robustness is defined as a metric that describes how prone an algorithm is to sudden changes. A solution could be very optimal, but become a terribly performing one in case of a single change. In such a case the robustness of a given solution would be bad or

low. This is not at all desired for parcel delivery, as it is guaranteed to have changes while the parcel delivery trucks are already on its way. These changes can include road blocks, broken vehicles, sick drivers or traffic jams. A way to view robustness is to run the algorithm both with and without problems simultaneously. This would give a visualization on how well the given algorithm is able to correct itself and still be as close to optimal as possible in case of a problem. Something else that would be interesting to test is to compare a simulation in which there are problems, with a simulation that has the same problems, but runs a different underlying algorithm for finding the solution. Doing it this way, our tool could be used to visualize how the newly found algorithm copes with unexpected problems compared to an existing algorithm, hoping to create a feel of stronger robustness of the new algorithm when presenting it to a customer.

Because both options described above are desired, a possible solution for us would be to simply allow the simultaneous visualization of two algorithms. Rather than launching the tool with a single file, launching the tool with two (or more) files would be the simplest solution. The same map will be used, but trucks could be colored differently depending on which algorithm they belong to. The same goes for graphs; rather than showing a single line, two lines could be shown each in a different color depending on the algorithm they belong to.

Having an exact metric for robustness seems like a difficult task, but an example of how this could maybe be done is calculating the ratio between the optimality of the solution with problems and the optimality of the solution without problems. However, now it becomes difficult to describe optimality; is there an exact metric that determines how optimal a solution is? Having this would require knowing the optimal solution, which is unlikely to be easily determinable due to the computational complexity of the problem. Alternatively, metrics like total kilometers driven or average package delivery time could be used instead of the optimality of the problem, which both somewhat describe how optimal a given solution is. The exact metrics that will be involved here, is yet open for discussion and depends solely on what is desired by our client.

D.6.4. Algorithms

Even though the visualization described in this project is primarily aimed at visualizing the ‘parcel delivery algorithm’ being developed by Almende, it is also a requirement that the visualization can run different types of routing algorithms. Additionally, the algorithm made by Almende is still under construction and is therefore not ready to be used in testing and development of the visualization tool. Therefore, the visualization algorithm will be tested with alternative routing algorithms throughout the project.

There are several approaches to tackle the parcel delivery problem. Such as greedy algorithms or even easier randomness. Solving this problem requires a graph with nodes and edges of the street networks, however. Luckily, this data can be retrieved from OpenStreetMap with the help of several Python libraries that can be used in the back-end. One such library is OSMnx which allows the retrieval of geospacial data from OpenStreetMap (Boeing (2017)). The algorithm of choice can then be run and the result can be mapped via the Leaflet library on the OpenStreetMap.

At first, however, an open source API will be used. This API enables the visualization to be tested earlier because it does all the work of choosing and implementing an algorithm with the map data of OSM. This API is called OpenRouteService.

Another choice for an API to be used in the beginning was the ‘Graphhopper API’, more specifically, the Route Optimization API by Graphhopper. The motivation for using this API would primarily be that it has extensive documentation and has good customer support. The drawback and eventual reason for not being picked is the fact that it is not a free service such as OpenRouteService. On the other hand, Graphhopper also has a few open-source libraries which at first glance seem appropriate as well and which are free. But on closer inspection, these libraries do not offer the full functionality that is required, such as working with the OpenStreetMap data out of the box when solving the optimization problem.

D.7. Conclusion

During problem analysis, it was shown that, in order to arrive at an optimal parcel delivery visualization tool that fits the product requirements, numerous design aspects and decisions should be thought of. When it comes to data visualization design in general, it has been found that it is essential to keep in mind what a user should take away from a visualization. The design process that follows should be

aimed at optimizing the clarity of this message with the help of a combination of visual effects that are known to support it.

In order to arrive at such a sound visualization, developers can make use of multiple existing tools and platforms, which should make the development process more efficient. The parcel delivery algorithm visualization requires the visualization of some type of map on which routes can be displayed, as well as a dashbourd-like interface with metrics and charts. It is therefore considered wise to make use of a map data visualization platform and a web application (JavaScript) framework. These platforms should allow scalability and clarity and should be cost efficient. After comparing different platform providers, as well as looking into examples of similar routing visualization projects, always with the parcel delivery visualization in mind, the chosen map visualization is OpenStreetMap and the JavaScript framework is Vue.js.

To deal with the requirements for the project that make it novel and unique, a few more design aspects have been analyzed. These have to do with the inclusion of autonomous vehicles in Almende's algorithm and the request for visualizing robustness measures. Autonomous vehicles will appear on the map with icons that differ from 'normal' cars, and roads traversable by autonomous vehicles will appear on the map with different colours than 'normal' roads. To show the robustness of the underlying algorithm, it has been found useful to allow simultaneous visualizations of algorithms. Such a feature could be useful for both comparing an algorithm with itself (with the inclusion of some limiting factor) and with other algorithms. In addition, robustness and optimality metrics shall be displayed in the dashboard.

All these findings should form a coherent image of what the parcel delivery algorithm visualization should look like and should describe roughly how this visualization tool will be implemented. Additionally, when in the development phase of the project, the conducted research should form a basis on which to rely when in doubt about certain design decision.

Glossary

COVID-19 An infectious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). As of 22 June 2020, more than 8.91 million cases have been reported across 188 countries and territories. As a result, many countries, including The Netherlands, have taken lock-down measures, that include the prevention of working outside the house and hosting face-to-face meetings. 1, 29

GeoJSON An open standard format, designed for representing simple geographical features, along with their non-spatial attributes. It is based on the JavaScript Object Notation. The features include points, line strings, polygons, and multi-part collections of these features. GeoJSON is compatible with a multitude of map libraries, including Leaflet. 7, 25, 26, 28, 32, 36, 46–48

Git A distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers and its goals include speed, data integrity, and support for distributed, non-linear workflows. 31

GitLab A web-based DevOps lifecycle tool that provides a Git-repository manager providing wiki, issue-tracking and continuous integration/continuous deployment pipeline features, using an open-source license. 31

MoSCoW A prioritzation method that groups requirements into the following groups: Must haves, Should haves, Could haves and Won't haves. 4

OpenRouteService (ORS) An open source service that is capable of computing optimal routes. For our project it is hosted at Almende's own server to compute directions between the markers. In addition, the API of ORS is called to compute a solution to the vehicle routing problem, as an alternative to Almende's own algorithm. 1, 2, 20, 21, 27, 28, 33, 35, 36, 38, 48

pandemic A disease that is occurring over a wide geographic area of the world and affecting an exceptionally high proportion of the population. 1, 29

Pydantic A data validation and settings manager for Python, that uses type annotations. You can create a subclass of Pydantic's BaseModel class, to build a model by which you can validate data. 27, 28, 32

scrum An agile framework for developing, delivering, and sustaining complex products, with an emphasis on software development. 30, 31

single-page application (SPA) A web application or website that interacts with the web browser by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages. A SPA loads what it needs per click, updates the target, and leaves the rest of the page untouched. v, 8

Slack A business communication platform that offers many IRC-style features, including persistent chat rooms organized by topic, private groups, and direct messaging. 30

Uvicorn An ASGI (Asynchronous Server Gateway Interface) server for Python/FastAPI applications. 31

Yarn A relatively new package manager that replaces the existing workflow for the npm client while remaining compatible with the npm registry. It operates fast, secure and reliable, and offers the option of static file serving. 31

Bibliography

Krzysztof Boczkowski and Beata Pańczyk. Comparison of the performance of tools for creating a spa application interface-react and vue. js. *Journal of Computer Sciences Institute*, 14:73–77, 2020.

Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 07 2017. doi: 10.1016/j.compenvurbsys.2017.05.004.

Jack Dougherty and Ilya Ilyankou. *Data Visualization for All: Tell your data story with free and easy-to-learn tools*. GitBooks, 2020.

Jacob Duvander and Oliver Romhagen. *What affects the choice of a JavaScript framework: Interviews with developers*. Jönköping University, 2019.

Jeffrey Heer and Michael Bostock. Crowdsourcing graphical perception: Using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 203–212. Association for Computing Machinery, 2010. doi: 10.1145/1753326.1753357.

Pieter Jan't Hoen, Girish Redekar, Valentin Robu, and Johannes A. La Poutré. Simulation and visualization of a market-based model for logistics management in transportation. In *Autonomous Agents and Multiagent Systems, International Joint Conference on*, volume 4, pages 1218–1219. IEEE Computer Society, 2004. doi: 10.1109/AAMAS.2004.10261.

Gordon Kindlmann, Erik Reinhard, and Sarah Creem. Face-based luminance matching for perceptual colormap generation. In *Proceedings of the Conference on Visualization '02*, pages 299–306. IEEE Computer Society, 2002.

Andy Kirk. *Data Visualisation: A Handbook for Data Driven Design*. Sage Publications Ltd., 2016.

Cole Nussbaumer Knaflic. *Storytelling with Data: A Data Visualization Guide for Business Professionals*. John Wiley & Sons, 2015.

Tamara Munzner. *Visualization Analysis and Design*. CRC press, 2014.

Hikmet Senay and Eve Ignatius. Rules and principles of scientific data visualization, 1990. URL `https://www.cs.rit.edu/usr/local/pub/ncs/hypervis/percept/visrules.htm`. [Online; accessed 21-April-2020].

Wikipedia. Data visualization, 2020. URL `https://en.wikipedia.org/w/index.php?title=Data_visualization&oldid=951917255`. [Online; accessed 22-April-2020].

Eric Wohlgethan. *Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js*. Hamburg University of Applied Sciences, 2018.

Dona M Wong. *The Wall Street Journal Guide to Information Graphics*. W.W. Norton & Company, 2010.