Francisco M. A. Monteiro

FIELD SERVICE OPTIMIZATION FRONTEND
RELATÓRIO FINAL

1 2 9 0 UNIVERSIDADE Ð COIMBRA

1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Francisco Miguel Almeida Monteiro

FIELD SERVICE OPTIMIZATION FRONTEND
RELATÓRIO FINAL

VOLUME 1

Dissertação no âmbito do Mestrado de Engenharia Informática, Engenharia de Software orientada pelo Professor Nuno Antunes e apresentada à Faculdade de Ciências e Tecnologia/Departamento de Engenharia Informática

Junho de 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Field Service Optimization Frontend

Final Report

Francisco Miguel Almeida Monteiro

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering advised by Prof. Nuno Antunes and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

June 2021

1 2 9 0

UNIVERSIDADE Đ COIMBRA

This page is intentionally left blank.

Abstract

More and more companies are looking for ways to maximize their profits while saving time. A large part of the expenses of these companies are related to field operations, such as transportation of resources. A work plan defines the work to be done by employees. Most of the time these work plans are not built in the most smart and efficient way.

The goal of this work is to build a web application in which customers can interact in order to obtain optimized work plans. The quality of these work plans can be evaluated by a set of metrics (e.g., total distance). By doing so, they can minimize the costs of this type of operations. We also hope to achieve some business generalization. In other words, we want this tool to be able to adapt to different types of businesses.

In order to accomplish this we elaborated a state of the art which allowed us to gather more information about this type of tools. Furthermore, we were able to define the high-level functional requirements of our tool. Having these as a starting point, we designed an architecture. Hereafter, we started the implementation process.

We used Agile as the methodology methodology. This process took about 4 months, distributed across 7 sprints. Throughout this time, the frameworks used were Django for the backend and React for the frontend.

Then, we had to validate our solution. In order to do this we did three types of tests: unit testing, integration testing and functional testing. Since the usability is extremely important in this internship context, we also applied a usability checklist.

As a result, we ended with a web application that is simple to use and that allows the end user to carry out the desired tasks. In the end, all the main objectives of this curricular internship were achieved. We built an interface that allows its users to minimize operating costs. In addition, there is also the possibility of being adapted to different types of businesses in the future.

Keywords

Route Optimization; Fleet Management; Route Planning

This page is intentionally left blank.

Resumo

Cada vez mais empresas estão à procura de formas de maximizar os seus lucros e em simultâneo poupar o seu tempo. A grande maioria das despesas destas empresas está relacionada com as operações no terreno, como por exemplo o transporte de recursos. Um plano de trabalho define o conjunto de tarefas a ser feito pelos funcionários. Na maioria das vezes, estes planos de trabalho não são construídos da maneira mais inteligente e eficiente.

O objetivo deste trabalho é construir uma aplicação web na qual os clientes possam interagir de forma a obter planos de trabalho otimizados. A qualidade destes planos de trabalho pode ser avaliada por um conjunto de métricas (por exemplo: distância total). Consequentemente, podemos minimizar os custos deste tipo de operações. Esperamos ainda conseguir obter alguma generalização. Ou seja, queremos que esta ferramenta seja capaz de se adaptar a diferentes tipos de negócios.

Para isso elaborámos um estado da arte que nos permitiu reunir mais informações sobre este tipo de ferramentas. Além disso, fomos capazes de definir os requisitos funcionais de alto nível da nossa ferramenta. Tendo isso como ponto de partida, construímos uma arquitetura. Consequentemente, iniciámos o processo de implementação.

A metodologia de desenvolvimento utilizada foi Agile. O desenvolvimento durou cerca de 4 meses, distribuídos em 7 sprints. Durante esse tempo, os frameworks utilizados foram Django para o backend e React para o frontend.

Posto isto, tivemos que validar a solução. Para o fazer, recorremos a três tipos de testes: testes de unidade, testes de integração e testes funcionais. Para além disso, como a usabilidade é extremamente importante no contexto deste estágio, foi ainda aplicada uma checklist de usabilidade.

Como resultado, terminámos com uma aplicação web simples de utilizar e que permite ao utilizador final realizar as tarefas desejadas. No final, todos os objetivos principais deste estágio curricular foram alcançados. Construímos uma interface que permite aos seus utilizadores minimizar os custos operacionais. Além disso, existe também a possibilidade de ser adaptada a diferentes tipos de negócios no futuro.

Palavras-Chave

Otimização de Rotas; Gestão de Frota; Planeamento de Rotas

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

ASRs Architecturally Significant Requirements. 3

COD Cash On Delivery. 11

DOM Document Object Model. 22, 23

ETA Estimated Time of Arrival. 18

FSM Field Service Management. 1, 3, 5, 21

IPN Instituto Pedro Nunes. 1, 50

JSX JavaScript XML. 23

NFRs Non-Functional Requirements. 27, 31, 37

PM Project Manager. 53

QA Quality Attributes. 26, 31

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Nowadays, more and more companies are increasingly trying to save resources in the management of their road transport operations. In order to do this, Field Service Management (FSM) solutions are used. FSM is a software solution that assists companies in the management of field employees and resources (e.g., vehicles) [6]. The main purpose of these tools is to get the job done in the best possible way, thus saving money.

These costs take into account many factors (e.g., fuel, tolls and vehicle wear) and for large companies these expenses can reach astronomical values. For this reason, it is critical to be able to maximize the resources available to each company.

Although there are many solutions on the market that help to overcome this problem, they are not the right ones. Each business is different from the next, therefore the result is not always the expected. With this in mind, many companies are looking for a solution tailored and adapted to their business.

In addition, many of these tools are not appealing to the user. Sometimes the results are not presented in the best way and as such they can become difficult to interpret.

In this internship, we try to address these problems. In order to do this we intend to create a tool in the form of a web application. The main objective is to help Sentilant's customers to optimize their work plans in an easy way, improving their business value. At the same time we want this tool to be able to adapt to different business types.

1.1 Context

This internship took place at a company named Sentilant, located on the Instituto Pedro Nunes (IPN), Coimbra. One of Sentilant's main focuses is the development of customized FSM systems for its customers.

Although the company has an office at IPN, this internship was done remotely. The reason for this is, at the time of writing, we are in the middle of the COVID-19 pandemic.

Since the market for FSM systems is very competitive, Sentilant seeks to find ways to innovate. As such, one of the things to consider is the tool with which clients interact directly. This tool must be appealing, easy and simple to use. In addition, it must also be able to adapt to changes that may arise.

Sentilant solves different types of FSM problems. However, there is one that matters the most in the context of this internship, the **Fuel Problem**. Currently Sentilant is working on a solution for this problem. In addition, it also has a client in this area.

The Fuel Problem consists of optimizing the transport of hazardous materials (e.g., fuels) inside tank trucks. Each truck, contains a cistern that is divided in multiple compartments. This compartments are distributed across different parts of the cistern. As such, the stability of the vehicle must be taken into account. For example, if only the compartments in the rear of the vehicle are filled, the vehicle may become unstable during a climb due to the weight exerted. At the same time, we also want to maximize the amount of materials the truck can transport. There are still other factors to take into account. The transport of these types of substances usually require specific licenses for each driver. In addition, some routes are quite extensive and require more than one driver. As such, the final tool must address all these details.

Although we only talk about the fuel problem, this application will also have to adapt to other types of problems.

For example, one of the other problems for which the company presents a solution is the **airport problem**. This problem consists in obtaining a work plan for the transport of multiple passengers between their homes and the airport.

There is an ambition for the application to be able respond to different types of businesses. Each potential customer has their own "problem", which basically represents the business process they are trying to optimize. While doing this, we need to find a way in which the application works properly for all kinds of problems.

We want to have a web application that serves all the customers. In order to achieve this, we need to build a multi tenant application. For consistency reasons, let's assume each customer is a tenant.

For example, lets assume we have two different tenants, with two different business types. They also have two different links to access this web application. At the same time, the information about each business is isolated and only accessible to the responsible tenant.

- **Client A** - has a fuel transport business - `http://clienta.fsm.local`
- **Client B** - has an airport transport business - `http://clientb.fsm.local`

The data model also has to be built in such a way that is able to adapt to different types of problems.

In addition, we also need to make sure our tool communicates with external systems (e.g., solvers). If the tool is not able to communicate with the solvers, it cannot obtain the solution. This is critical for the correct functioning of the application.

These solvers are a piece of software quite complex and difficult to understand. As such, how these work is totally out of the context of this internship. The solver used throughout this internship was built by a Sentilant member, during his internship at the company. Section 2.1 describes how it is used.

Finally, if possible, we will still have to address how we are going to integrate our solution with the current tools used by Sentilant. The main example of this is **DrivianTasks**, Sentilant's current solution. DrivianTasks is also a web application.

1.2 Internship objectives

The main objective of this internship is the creation of a web application that can later stand out from other optimization tools of route planning existing in the market. In other words, the main focus of this work is to develop a web application in which customers can interact in order to obtain their optimized work plans. We can divide this work in the following sub-objectives:

- Develop a web application that allows the introduction of tasks and certain resources, in order to obtain an optimized plan for execution, taking into account the resources previously indicated
- Build a simple-to-use web interface that makes it possible to parameterize and manage the optimization and later view the results obtained in a concise way

1.3 Document structure

The rest of the present document is divided into the following chapters:

- **Chapter 2 - State of the art:** We start by providing a little background on the topic of FSM. In addition, an analysis and comparison is made on the current solutions on the market. Finally, we make a comparison to choose one of two tools to be used in the development phase.
- **Chapter 3 - Requirements:** Defines the Architecturally Significant Requirements (ASRs) of our tool, adopting an high-level overview. These include: functional requirements, technical & business constraints and quality attributes.
- **Chapter 4 - Tool Overview:** A design of mockups was carried out. Additionally, a preliminary high-level architecture is presented.
- **Chapter 5 - Validation:** Checks if we built the right product. Contains all the tests made and also a usability checklist.
- **Chapter 6 - Internship Planning:** Illustrates the development methodology. Furthermore, internship planning for the first and second semester is analyzed.
- **Chapter 7 - Conclusion:** Summarizes the work done throughout the internship. Future work is also discussed.

This page is intentionally left blank.

Chapter 2

State of the art

Instead of "diving" directly into the construction and development of our application it is a good practice to first do an in-depth search on the topic. The main objective of this chapter is to elaborate a state of the art that allow us to gather some information on the work to be done.

However, since we are not developing our work in a scientific scope, but in a curricular internship context we have some minor differences. Most of the time in this phase, reading articles and papers is the way to go. Instead, we intend to build a web-based application to be used by potential clients. So, the approach to be followed consists of analyzing the tools currently on the market, which serve the same purpose as the system we intend to build.

This work not only allows us to have an idea of the approaches that others have used to deal with similar problems, it also allows us to have an idea of the possible competition in the market. Furthermore, when doing this market analysis, we can observe the weaknesses of the competitor's tools and aim to build a more complete application for our clients.

In Section 2.1 we can find the Background knowledge needed to understand some aspects of this document. Section 2.2 lists some tools on the market that have similar objectives to the tool to be built. In Section 2.3 these tools are placed side by side in order to be compared. Since we have to choose one tool specialized in frontend web development, in Section 2.4 we put React and Angular side by side.

2.1 Background

Before we begin to structure the problem towards the solution, there are some concepts and terms that need to be introduced. As mentioned earlier, one of the main goals of a Field Service Management (FSM) system is to optimize routes. Some questions may arise to the reader. What does this mean? How are we going to do this?

Route Optimization is the process of calculating the best possible route while minimizing its costs [7]. It is not always this linear, because we must consider external factors such as fuel price, traffic, tolls, among others. The process of calculating the most optimal route, taking into account all the constraints imposed, is done by a piece of software called **solver**. Although the solvers are quite important in what concerns the quality of the solution, as mentioned earlier, they are not within the scope of this internship. Instead, we focus on the looks and functionalities of our tool.

Analogously, we can think of a solver as a car. We can know how to drive a car, but not how it works internally. Basically, all we need to know is how to use this component.

As mentioned before, a solver is a piece of software that receives a list of resources (drivers and vehicles) and a list of tasks to be optimized. Returns a list of optimized routes with tasks. In addition, it also returns a set of metrics about each route, as well as general metrics. If it is not possible to optimize certain tasks, we also receive that information and the reasons for doing so. In order to use the solver, we need to make a **HTTP POST** request where we pass all the information in a **JSON** object. The response, an optimized work plan, is also a JSON object.

Furthermore, there are some terms we must have in mind throughout this document. We have 3 main entities that need to be defined:

- **Tasks**: Sometimes referred as orders. They can be of many types (e.g., delivery and pickup). They represent the work that needs to be done, normally has a deadline associated. For example: "Package A must be delivered at Place B before Friday, 10:00 AM". They are usually under many restrictions. For example, they may require special vehicles or drivers.

- **Vehicles**: Means of transport used to perform these tasks. They have several characteristics, such as load capacities (e.g., volume and weight).

- **Drivers**: Person responsible for the vehicle and for carrying out the tasks. It also has a certain type of characteristics, such as skills.

Additionally, we need to define what a **work plan** is. The work plan is where we can find all the information regarding all the work to be done in a specific time window. It has information on the tasks to be performed and the routes to follow in order to do them. It also has a set of metrics (e.g., total time, total distance and total costs) in order to assess its quality. The work plan contains the optimized solution that is presented to the user, whether it is displayed in a map or in some other way. Figure 2.1 illustrates a generic work plan.

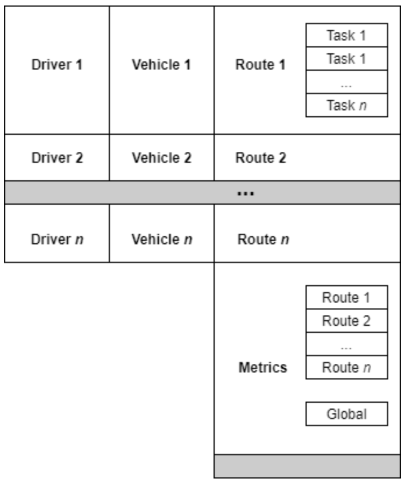


Figure 2.1: Generic Work Plan (adapted from Tailored Field-Service Optimization [1])

2.2 Existing tools

Before jumping into the market analysis, no restrictions were imposed on the type of tools to be analyzed. The only thing to consider is that they should have the main functionalities of the application to be built within the scope of this internship. With this is mind, we proceed to search all kinds of tools (free, paid, web-based, desktop, among others).

Although we did not impose any restrictions in relation to the type of platform on which the application runs, all that we found were web-based tools. Just like ours will be, with one or two exceptions. In addition, it should be mentioned that many of the tools found had integration with a mobile app, mostly for the field end-users, in this case drivers.

As soon as we started looking for route optimization tools, we were immediately overwhelmed by the amount of results found when searching for tools with similar goals. As such, we ended up finding more than 30 tools that perhaps made sense to analyze with more detail. However, doing this would be a very time-consuming job. In order to solve this problem, I explained the situation to my advisor in the company. In consequence, we agreed that the best thing to do was to make a shortlist with a few tools to analyze with more detail. At this point, the problem arose as to how we could choose the best tools that would make this shortlist.

A superficial (high-level) analysis was performed on all the candidate tools. Those that stood out are part of the shortlist. Some are interesting and for that reason are mentioned in the honorable mentions in Subsection 2.2.6. The rest of the tools were ignored and discarded.

With that done, we ended up with 5 tools on this list. They will be analyzed in more detail in the following subsections. Finally, the analysis of each tool will be divided into **5 components: brief explanation of how the application works**, **main features**, **size of the problems it can optimize**, **weaknesses** and **strengths**.

2.2.1 Routific

Routific is a cloud web-based route optimization software platform that helps delivery businesses plan their routes more efficiently. The company responsible for this product appeared in 2012 and since it was founded based on this solution, both the company and the tool share the same name. Compared to the rest of the tools on this list, Routific is probably the simplest. Not only from the point of view of use, but also in terms of configuration. Its simplicity is one of the main reasons why this tool was chosen. Regarding the price, this tool is paid and has several subscription tiers.

The first step to start using this platform is to create an account at **www.routific.com**. Once done, we can proceed to authenticate with the email and password previously registered. The first time we enter the application, we are asked to create a project. Each project created is saved in the cloud and has certain associated characteristics such as drivers and tasks. In addition, it still has a name (to make it easier to distinguish) and a date. This tool also allows us to view the list of previously saved optimized plans. As such, we can return to use recurring plans. In addition, in this list we can compare our optimizations in general, since we are presented with a certain set of metrics. Within these metrics we have: number of drivers, number of tasks (assigned and unassigned), total distance and total working time. This can be seen as a kind of comparison since it grants us a general idea of the quality of the plans.

Once the project is created, we are presented with a main dashboard. In order to start optimizing tasks, first we have to configure our drivers and tasks. In the case of the drivers, there are 3 ways to do it. The first one, which is the simplest, we just need to introduce the starting address for the designated driver. In the second one, in addition to the starting address, we can also add several additional parameters such as working hours, capacity of the vehicle, lunch break, among others. Last, but not least, we can also import our drivers using properly formatted .csv or .xls files.

Regarding the tasks, as with the drivers, we also have 3 ways to create them. The simplest way is to just enter a name for the stop and its address. Alternatively, we can create more detailed tasks, with parameters such as duration, customer email, delivery notes and so on. Finally, we can also import a .csv or .xls file with all tasks, as long as it is in the correct format.

Once that is done, we can finally proceed to the optimization process on the main dashboard. We just have to select the tasks and drivers we want to use and optimize our plan. Still within the main dashboard, we can observe our tasks and drivers distributed across the map. From the moment we optimize the plan, the routes are also displayed on the map. Additionally, at the bottom of our dashboard, we also have a timeline where it shows the assigned drivers and their tasks. On this timeline, we can also get an idea of some metrics of the plan such as total distance, working time, tasks scheduled, among others. Figure 2.2 illustrates Routific dashboard.

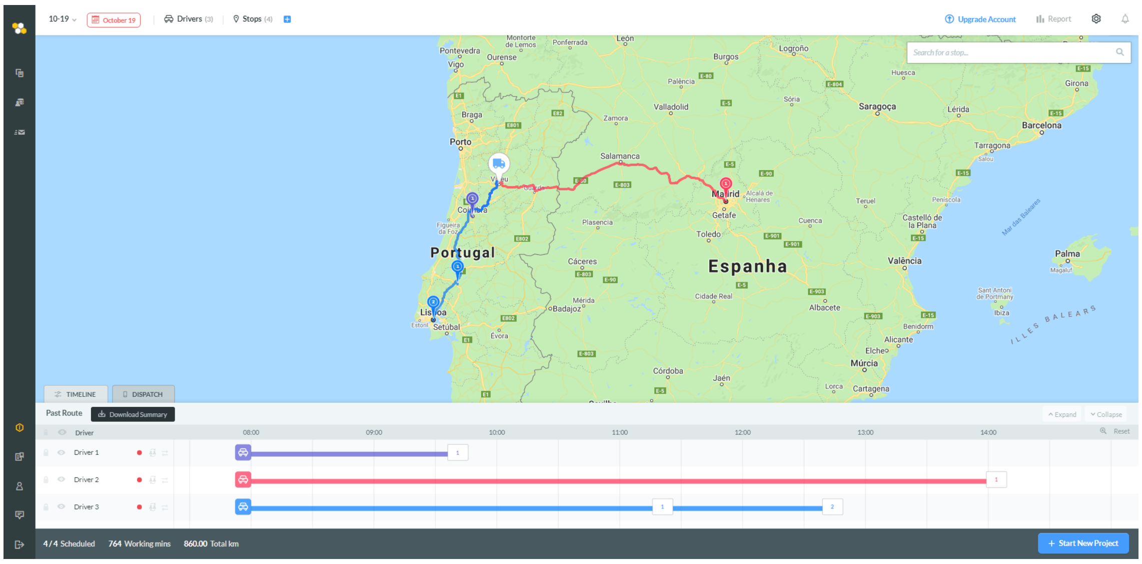


Figure 2.2: Routific - Dashboard

It is also important to mention that this tool has a configuration menu that concerns each project. In this menu, we can define a number of restrictions and parameters that can directly affect our plan. We can choose metrics related to distances (miles or kilometers), select expected traffic levels as well as some settings that can be applied in general to all tasks. In addition, we can also specify some general configurations that concern drivers. Such as working overtime and additionally balancing routes so that all drivers have the same workload.

With regard to features, this platform also has integration with a mobile app. This allows us to send the routes to each one of the drivers as soon as the plan is optimized, as long as they have the mobile application installed. Then, the drivers can configure their navigation system taking into account the tasks assigned. Another feature is the existence of end of the day reports. This way we can have a general idea of how the work day went in view of the plan used.

When it comes to the size of the problems that Routific can handle we were not able to find that much information. When consulting the price list (https://dev.routific.com/pricing), we found out that the tool supports an unlimited number of vehicles and that it allows up to 2000 tasks (also known as visits) per optimization

In order to conclude the analysis of Routific, we will summarize its strengths as well as its weaknesses. The ease of configuration and use of this platform was seen as one of its strengths. In addition, it is still quite trivial to interpret the results of the optimized plan, both on the map and on the timeline table. Finally, the speed with which we can obtain a plan, from the beginning of the configuration to the end of the optimization.

On the other hand, the fact that this tool is quite simple, also has its drawbacks. One of these weaknesses is the fact that we are very restricted when it comes to possible configurations. The tasks that can be defined are quite simple with few metrics and dimensions and as such they often do not allow to simulate real-world problems. Finally, there is also no distinction between drivers and vehicles. There are no vehicles in this tool, it is as if they were part of the driver's entity. Once again, we are faced with a new constraint that does not occur in the real world.

2.2.2 OptimoRoute

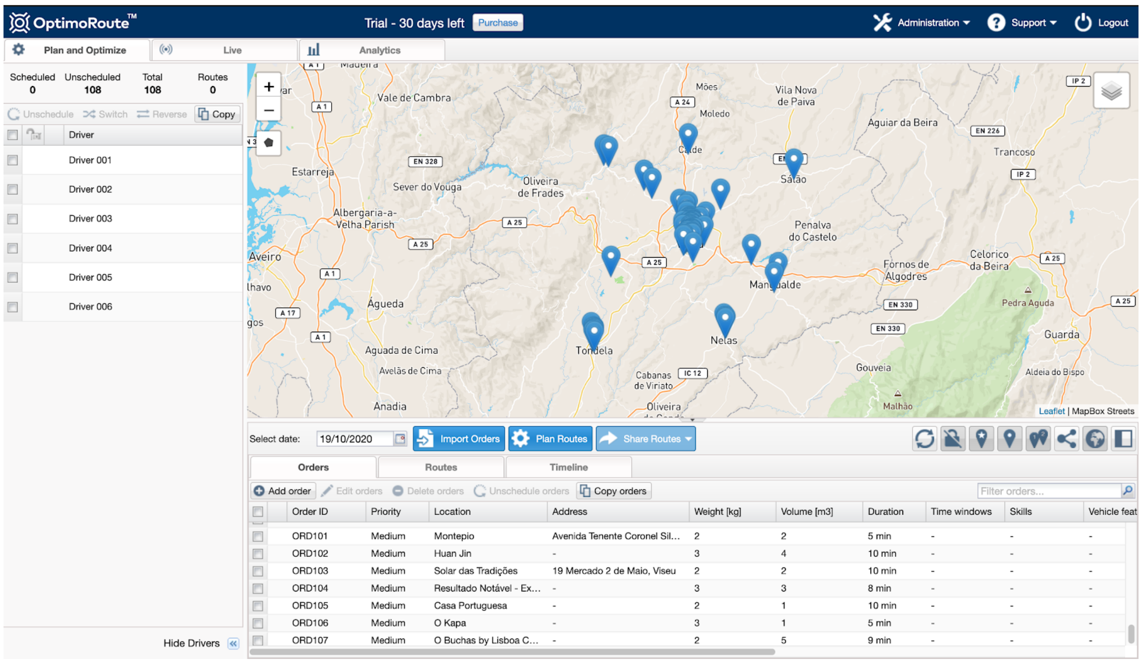
OptimoRoute is a cloud-web based software which focus on route optimization and schedules for delivery and field service businesses. It was founded in 2012 in Palo Alto, US, and just like Routific, both the company and the product have the same name. This tool, although simple, manages to give a wide choice of possible configurations to the end-user. Regarding the price, it also has a business model based on subscription plans. However, we are able to subscribe to a 30-day free trial.

Here there is also the need to create a user account. Once that is done, we can authenticate ourselves in the website in order to start the optimization process.

The first time we enter the application we have to fill in some general information about our business. First, we need to enter the address of our main depot. In a nutshell, a depot is like a warehouse where we can store large quantities of goods. In this case, the depot will be the place from where the drivers will depart. Then, we need to choose our type of business (food delivery, retail & distribution, among others). Finally, we define the number of drivers we have and their working hours. Additionally, we can define break times for these. Once that is done, the initial configuration is finished and the program generates a certain number of tasks so that we can start exploring the application.

As with other applications, OptimoRoute is composed of a main dashboard. In this dashboard, in the center, we have a map where our tasks are arranged. At the bottom, we have an interactive list of our orders, routes and a timeline of our plan. On the left side of our screen, we can see our drivers with some metrics just like their orders and their working time. Additionally, we can also see the number of tasks scheduled and unscheduled and the total number of routes generated.

Figure 2.3 illustrates OptimoRoute dashboard.



Order ID	Priority	Location	Address	Weight [kg]	Volume [m3]	Duration	Time windows	Skills	Vehicle feat
ORD101	Medium	Montepio	Avenida Tenente Coronel Sil...	2	2	5 min	-	-	-
ORD102	Medium	Huan Jin	-	3	4	10 min	-	-	-
ORD103	Medium	Solar das Tradições	19 Mercado 2 de Maio, Viseu	2	2	10 min	-	-	-
ORD104	Medium	Resultado Notável - Ex...	-	3	3	8 min	-	-	-
ORD105	Medium	Casa Portuguesa	-	2	1	10 min	-	-	-
ORD106	Medium	O Kapa	-	3	1	5 min	-	-	-
ORD107	Medium	O Buchas by Lisboa C...	-	2	5	9 min	-	-	-

Figure 2.3: OptimoRoute - Dashboard

This application also has a configurable settings menu. Here we can define parameters like how the routes should be balanced. For example, we can decide to use all available drivers and balance routes between them. Alternatively, we can opt to just balance the routes, which can lead to unused drivers. Additionally we can also select the balancing factor which can be either the working time or the number of orders per driver.

When compared to other applications, with regard to the number of features and possibility of configurations, OptimoRoute can be considered very complete. We can configure several parameters related to entities like drivers, tasks, locations, vehicles, among others. One of the most interesting features is the possibility of customizing our entire vehicle fleet. We can define types of vehicles, such as "refrigerated vehicles", "vehicles for transporting dangerous goods" and so on. Furthermore, unlike Routific, there is a clear distinction here between the entities of the driver and the vehicle. Just like it should be, since it is what happens in real world problems.

We could continue to list more features, however we will not do so. Since later on we will place the tools side by side and we will take this into account.

When it comes to the size of the problems that OptimoRoute can handle in the optimization process we were only able to find information about the number of tasks. This tool can handle up to 750 tasks per optimization.

The ease of configuration and use of this platform can be seen as one of its strengths. It is extremely easy to setup and to start using. Besides that, the optimization is done very quickly. It also has a wide range of possible configurations and constraints that can be used in the process. Last but not least, it is quite simple and easy to interpret the results of the optimized plan. At the same time, it also allows us to be aware of the quality of the plan, if we analyze the metrics calculated alongside the plan.

Finally, we went looking for weaknesses. The tool does not allow us to have more than one depot. Therefore, we are restricted in a way that all the drivers must have the same starting point. This does not always happen in real world problems.

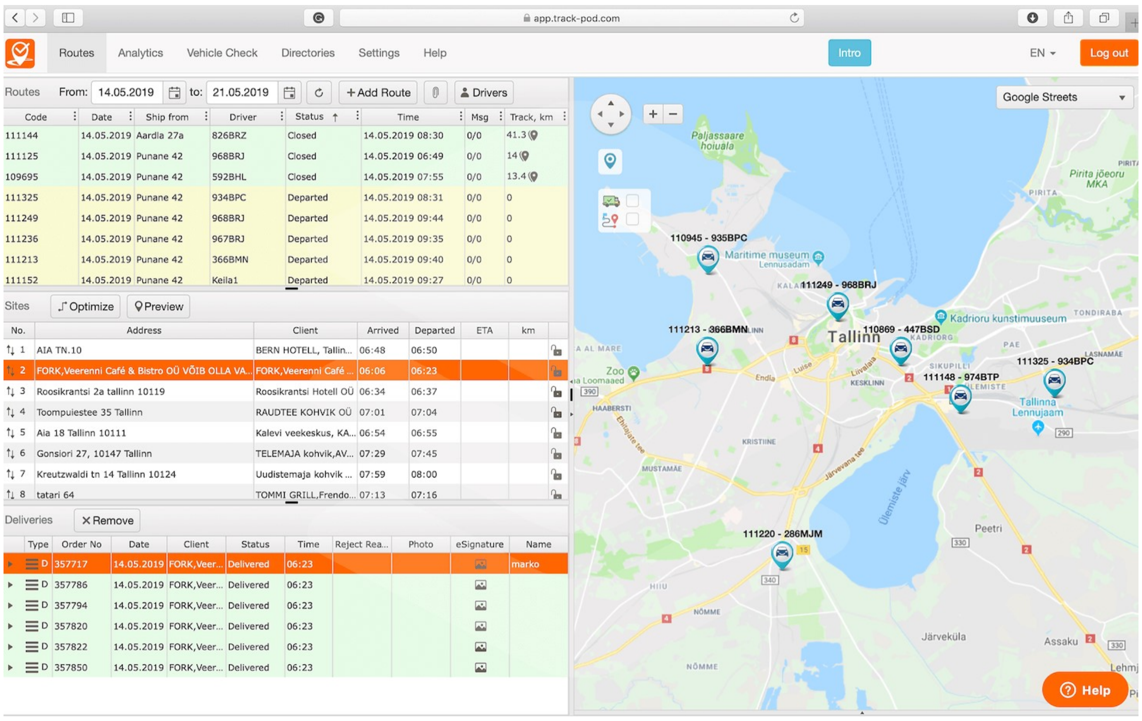
2.2.3 Track-POD

Track-POD is a cloud-based delivery management system specialized in route optimization. It is web-based, just like the previous tools. In addition, it is also paid, based on subscription plans. The company that supports this tool was established in 2016 in Vilnius, Lithuania.

In order to start using the application, the first thing we need to do is create a user account. The first time we authenticate ourselves, we come across a window where we need to enter the address of our depot or alternatively our office. Afterwards, we configure our first driver. Some of the characteristics of each driver are: name, vehicle, phone number, location where he ships from, and so on. Additionally, when a driver is being created, we can define also a username and password. This is for him to be able to access his work plan through the mobile application developed to be used in conjunction with this tool. That said, we are in a position to start adding tasks. Each task can be one of these 4 types: Delivery, Collection, Pickup & Delivery and Pickup-Hub-Delivery. A task has associated some fields such as a client, address, date, contact, type of goods. Additionally it also has information about where it ships from and details about the cargo (e.g., weight, volume, quantity and so on). We can also specify a Cash On Delivery (COD) value in each order.

Regarding the main dashboard, we can consider this more complex than the other tools analyzed so far. The dashboard divides the screen vertically into two halves. On the right half, we have the map. On the map we can see the disposition of our drivers as well as our deliveries. Since we do not have the routes represented on the map, it becomes a little difficult to interpret our plan. On the left half, we have three tables which are the main reason why this dashboard can be more complex. The first table has all the routes and the second one has all the sites for deliveries. The last table, the third one, has all the deliveries to be done.

Figure 2.4 illustrates Track-POD dashboard.



Routes From: 14.05.2019 to: 21.05.2019 + Add Route Drivers

Code	Date	Ship from	Driver	Status	Time	Msg	Track, km
111144	14.05.2019	Aardla 27a	826BRZ	Closed	14.05.2019 08:30	0/0	41.3
111125	14.05.2019	Punane 42	968BRJ	Closed	14.05.2019 06:49	0/0	14
109695	14.05.2019	Punane 42	592BHL	Closed	14.05.2019 07:55	0/0	13.4
111325	14.05.2019	Punane 42	934BPC	Departed	14.05.2019 08:31	0/0	0
111249	14.05.2019	Punane 42	968BRJ	Departed	14.05.2019 09:44	0/0	0
111236	14.05.2019	Punane 42	967BRJ	Departed	14.05.2019 09:35	0/0	0
111213	14.05.2019	Punane 42	366BMN	Departed	14.05.2019 09:40	0/0	0
111152	14.05.2019	Punane 42	Keila1	Departed	14.05.2019 09:27	0/0	0

Sites Optimize Preview

No.	Address	Client	Arrived	Departed	ETA	km
1	AIA TN.10	BERN HOTELL, Tallin...	06:48	06:50		
2	FORK,Veerenni Café & Bistro OÜ VÕIB OLLA VA...	FORK,Veerenni Café ...	06:06	06:23		
3	Roosikrantsi 2a tallinn 10119	Roosikrantsi Hotell OÜ	06:34	06:37		
4	Toompuiestee 35 Tallinn	RAUDTEE KOHVIK OÜ	07:01	07:04		
5	Aia 18 Tallinn 10111	Kalevi veekeskus, KA...	06:54	06:55		
6	Gonsiori 27, 10147 Tallinn	TELEMAJA kohvik,AV...	07:29	07:45		
7	Kreutzwaldi tn 14 Tallinn 10124	Uudistemaja kohvik ...	07:59	08:00		
8	tatari 64	TOMMI GRILL,Frendo...	07:13	07:16		

Deliveries × Remove

Type	Order No	Date	Client	Status	Time	Reject Rea...	Photo	eSignature	Name
D	357717	14.05.2019	FORK,Veer...	Delivered	06:23				marko
D	357786	14.05.2019	FORK,Veer...	Delivered	06:23				
D	357794	14.05.2019	FORK,Veer...	Delivered	06:23				
D	357820	14.05.2019	FORK,Veer...	Delivered	06:23				
D	357822	14.05.2019	FORK,Veer...	Delivered	06:23				
D	357850	14.05.2019	FORK,Veer...	Delivered	06:23				

Figure 2.4: Track-POD - Dashboard (from Track-POD website (www.track-pod.com))

Concerning the size of the problems that this tool can handle, we were unable to find any information.

In short, the vast number of features and functionalities at our disposal can be seen as a major strength of this application. Although we were unable to test many of them since they are not part of the free trial. In addition, it is also worth mentioning how quickly the optimization is done.

However, there are also a few weaknesses to point out. First, due to its complexity, this tool is significantly more complicated to configure and use when compared to the others. In a certain way, we can say that the learning curve is high. When we were exploring the tool we had a few struggles in order to do the first optimization. Lastly, the optimized plane is also significantly complicated to interpret. This is mainly due to the way the dashboard is built. The high number of tables full of information becomes overwhelming to the end-user.

2.2.4 eLogii

eLogii is a cloud-based delivery management software solution for route planning and route optimization. Just like the previous tools it is also web-based. eLogii is part of the Brisqq Group which was founded in 2015 in London, UK. This tool is paid for and uses subscription plans that can be paid monthly or annually.

As always, we first need to create a user account in order to be able to access the application. The first time we authenticate ourselves, we are presented with a quick setup tutorial. That said, the first step is to add our first depot, similar to what happened with other tools. To do this, we just need to give our depot a name and specify its address. It should be noted that in this tool we can have more than one depot. The second step, is to set up some dimensions, which are defined by their name and unit. This dimensions are used both when defining vehicles and their capacities, and creating new deliveries and their size. For example, we can define a dimension called "Weight" with the unit "Kg". The third step on our setup tutorial is to establish some skills for our soon to be drivers. Drivers skills allows us to define which drivers can be assigned to which tasks. Afterwards, we can also set up vehicle capabilities. Vehicle capabilities allows us to define which vehicles can be used for which tasks. For instance, we can have a refrigerated vehicle to transport specific goods. Next, we are presented with a menu with the optimization settings (e.g., return to depot at end of route or allow driver overtime tolerance). Subsequently, we are required to define at least a vehicle type before we can start to insert our fleet into the system. Vehicle types serve to represent vehicles with similar characteristics. For example, we can create a "Light vehicle" type that has a "Weight" dimension with a capacity of 3500 kg. Alternatively, we can create a "Heavy vehicle" type for larger weights. This allows us to make our problems much more realistic, just like we want them. With that done, we can then start creating our vehicles. To do this we just need to enter a name and the type of vehicle. That said, all that remains is to create the drivers. In order to do so, we must insert his name, mobile number and a username and password. With this credentials, the driver can authenticate in the mobile app, which is integrated with the system, and access the tasks assigned to him. Additionally, when creating a driver, we can still assign him a vehicle, schedule and skills. Finally, all that remains is to create the tasks. We have two types of tasks: single delivery or pickup & delivery. A single delivery is a delivery where the starting point is the depot and the ending point is the recipient. On the other hand, pickup & delivery consists of a delivery in which we will collect the order at a point A that later will be delivered at a point B. Once the tasks are created, we are able to finish the introduction tutorial and optimize the plan.

In the main dashboard, the results are arranged in two components. As usual, we have a map in the center with the routes of each driver drawn. Below, we have the traditional task timeline, also referring to drivers. We also have the percentage of load that each one takes, as well as the percentage of working time. Finally, we are presented with some general metrics about the plan (e.g., total distance, total duration and average driver utilization percentage).

Figure 2.5 illustrates elogii map, and Figure 2.6 illustrates elogii timeline.

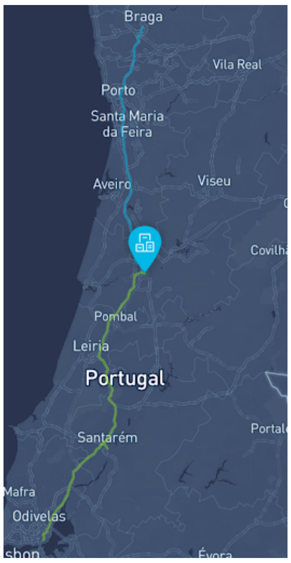


Figure 2.5: Elogii - Dashboard - Map

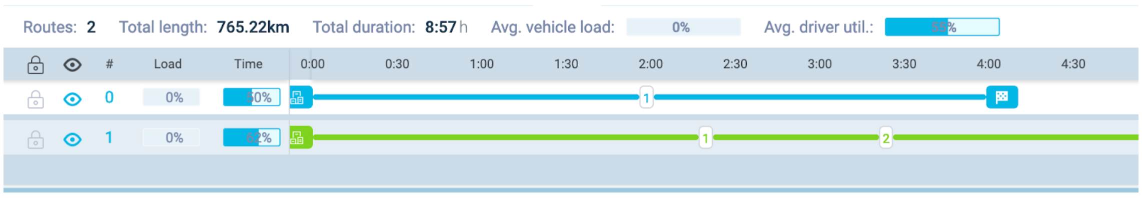


Figure 2.6: Elogii - Dashboard - Timeline

Concerning the size of the problems, eLogii can handle a number of unlimited tasks. In relation to the number of vehicles, it supports more than 1000.

Regarding the strengths of this tool, we can highlight a few. It is worth mentioning the ease of configuration and usage of this tool. The learning curve is quite low. This can be easily explained by the high quality of the introduction tutorial. Besides that, the speed with which the optimization is done is also quite high. Finally, the dashboard is very appealing and well built which allows for an easy interpretation of results.

About the weaknesses, we did not find much. We just considered that the tool could be more complete in terms of possible configurations.

2.2.5 ElasticRoute

ElasticRoute is a web-based route planning and optimization solution. Like the other tools, it also has its infrastructure in the cloud. The responsible company was founded in 2017 in Singapore, SG. This tool is free, but has more complete subscription plans that are paid monthly or annually.

The first time we use this tool, we are presented with a menu with an extensive menu with the general settings. We must insert our location and respective timezone. This menu is very complete and has a lot of possible settings. Once we have configured the settings, we can begin to enter the information about our business. The first step is to insert a depot, giving it a name and its address. It should be noted that we can add more than one depot. Now, we can start to define our vehicle fleet. As with eLogii, we have to define a vehicle type first (e.g., truck). A vehicle can have more that one type. That said, we are able to create our vehicles. A vehicle has a name and a type and in addition can also have a start and end depot. As expected, we must also define its capacity. In order to do this, we have 3 dimensions: weight, volume and seating. Regarding the drivers, we can define break time windows and availability schedules. As with Routific, in ElasticRoute there is no clear distinction here between drivers and vehicles. With the vehicles defined and initial configurations done, we can start to introduce the tasks. We can then calculate our optimized work plan.

In order to interpret the results, we also have a kind of dashboard. This dashboard is similar to the ones we have seen so far, despite this there are some differences. It consists of a map with the routes and a table where we can see the stops. As such, it does not have a timeline as the others tools usually have. In a certain way, we can say that this dashboard is not so good when compared to the others.

Figure 2.7 illustrates ElasticRoute dashboard.

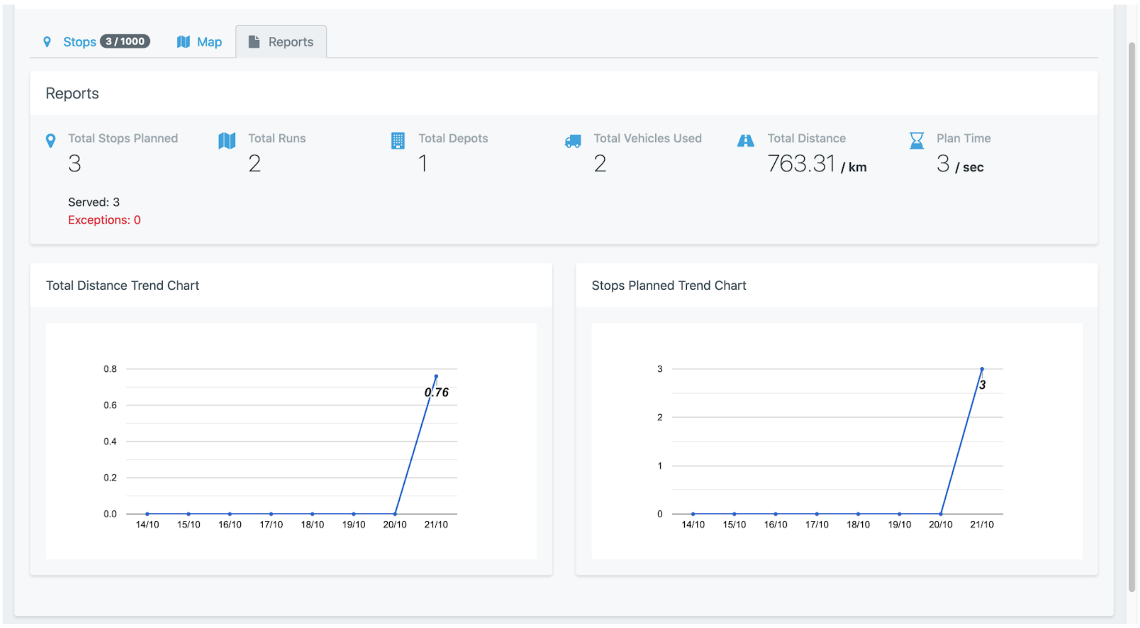


Figure 2.7: ElasticRoute - Dashboard

Concerning the size of the problems this tool can deal with, the information found is quite explicit. We can route up to an unlimited number of vehicles with a maximum of a 1000 stops per plan. The number of plans and re-planes we can do is also unlimited.

This tool, despite its construction quality, still has its strengths. In this case, it is just one. The large number of possible configurations, perhaps the largest analyzed so far. This is perhaps the main reason why we analyzed this tool in depth and why it is a part of this short list.

Regarding the weaknesses, we managed to point out a few. The tool is not well structured and as such it is not intuitive to use. The way the dashboard is found is one of the main reasons, it becomes very difficult to interpret the results. Furthermore, it is also quite difficult to navigate within the application. In addition, as mentioned earlier, there is no distinction between the driver and the vehicle. This should not happen.

2.2.6 Honorable mentions

Although they are not on the short list, there are still some tools worth mentioning. As such, this subsection serves to list some of these tools.

- **Less Platform**: cloud-native and web-based logistics optimization and management platform. The speed of optimization is one of its main strengths. However, the dashboard is a little overwhelming and clumsy.
- **onfleet**: delivery management software. Apparently quite appealing in the world of FSM systems, won some awards in 2020. Good amount of functionalities. On the other hand, can be considered quite complex.
- **GSMTasks**: web-based route planning & optimization software. Reasonable number of features and easy to use. It has been around for a few years (since 2000), so it is a little outdated. In addition, it also appears to have some technical problems.

2.3 Comparison

Placing these tools side by side and comparing them directly is not feasible. Although in the end they are similar, they were built for different purposes, with different end goals. As such, in order to be able to compare these tools efficiently, we need to find comparison points. In this approach, we create tables taking into account the characteristics of each one the tools.

In the following two subsections we have two tables in order to represent this vectors. The first table takes into account the general general aspects of each. The second one focuses on more technical and functional aspects.

2.3.1 General comparisons

Before analyzing the technical aspects of each tool, we should compare them in a more general context. Features such as price, have to be taken into account. For instance, we can have two similar tools with identical prices. However, if we analyze with more detail, one can be much more complete than the other. This subsection serves to compare our tools, regarding this type of characteristics that do not fit into a technical aspect.

- **Price**: Starting price of the tool
- **Free trial**: If the tool provides a free trial. It is a good way to attract potential customers.
- **Owner**: Company responsible for the tool
- **Configuration difficulty**: The difficulty to configure the application in order to start using it
- **Difficulty of use**: The difficulty to use the application once its configured

- **Premium features**: Can we buy new features inside the tool?
- **Tutorial**: Existence of a beginners tutorial inside the application

	Routific	OptimoRoute	Track-POD	eLogii	ElasticRoute
Price	Starting 39$ per vehicle/month	Starting 19$ per driver/month	Starting 35$ per driver/month	Starting 159$/month	Free
Free trial	✓	✓	✓	✓	✓
Owner	Routific	OptimoRoute	Track-POD	eLogii	Detrack Systems
Configuration difficulty	Simple	Simple	Elevated	Relatively simple	Moderate
Difficulty of use	Simple	Simple	Elevated	Simple	Moderate
Premium features	✓	✓	✓	✓	✓
Tutorial	✓	✓	✗	✓	✗

Table 2.1: General aspects comparison

By looking at Table 2.1, we can draw some conclusions. Regarding pricing (according to salesforce.com) we have some differences. There are 4 different business models here. Routific charges per vehicle per month. OptimoRoute and Track-POD use another approach and charge by driver. Meanwhile eLogii charges per month. ElasticRoutes despite being free, only supports one vehicle in this plan. However, for 39$ per month we can activate a more complete subscription plan.

In relation to the difficulty of configuration and use, it is important to note that the Track-POD has an elevated difficulty. This factor can make a difference to the end-user.

When it comes to the tutorial, Track-POD and ElasticRoute do not have it. Finally, all the tools have free trial and premium features.

2.3.2 Features and technical comparisons

Since we have finished comparing the tools in general terms, we can now proceed to do a comparison in technical terms by analyzing the features. In order to do that, we thought that was a good idea to divide this subsection in 4 parts, one for each component.

As such, we ended up with a component that concerns the most general features of the application. In addition, we also have a part for each one of the most important entities discussed so far: **tasks**, **drivers** and **vehicles**. This subsection serves that purpose.

The metrics that follow are features that most of the analyzed applications have in general. These are also features that can make sense to be considered for our solution. Furthermore, it is still important to analyze them for reasons of consistency. For example, it does not make sense to compare tools, side by side, that are built for different types of devices (e.g., mobile vs desktop).

General

- **Web based**: Is the tool web based? Can it be used on a web-browser?
- **Cloud based**: If the tool is cloud based. Is the tool hosted on the cloud?
- **Mobile integration**: If the tool provides a complementary mobile application (e.g., for drivers to use)
- **Multi user**: Can the tool be accessed by multiple users?

- **Optimization**: Type of optimizations the tool can do
- **Live tracking**: Does the tool provide a feature in order to see the current position of a vehicle?
- **ETA**: Does it provide a Estimated Time of Arrival (ETA)?
- **Multi depot**: Can we have more than one depot?
- **Optimization time**: Does it provide information about the optimization time?
- **Results presentation**: How are the results of the optimization presented?
- **Stationary time**: Can we see the amount of time a vehicle is stationary?
- **Empty time**: Can we see the amount of time a vehicle is driving with no cargo?
- **Compare optimization**: Can we compare optimizations?
- **Save optimization**: Can we save optimizations?
- **Fuel problem**: Can the fuel problem be solved using this tool?

	Routific	**OptimoRoute**	**Track-POD**	**eLogii**	**ElasticRoute**
Web based	✓	✓	✓	✓	✓
Cloud based	✓	✓	✓	✓	✓
Mobile integration	✓	✓	✓	✓	✗
Multi user	✗	✓	✓	✓	✓
Optimization	Time Distance	Time Distance Working time	Time Distance	Time Distance	Time
Live tracking	✓	✓	✓	✓	✗
ETA	✓	✓	✓	✓	✓
Multi depot	✓	✗	✓	✓	✓
Optimization time	N/A	N/A	N/A	N/A	✓
Results presentation	Routes on map Drivers work plan	Routes on map Drivers work plan	Routes on map Drivers work plan	Routes on map Drivers work plan	Routes on map Drivers work plan
Stationary time	✗	✗	✗	✗	✗
Empty time	✗	✗	✗	✗	✗
Compare optimization	✓	✗	✗	✗	✗
Save optimization	✓	✓	✗	✓	✓
Fuel problem	✗	✗	✗	✗	✗

Table 2.2: Tools - Features and technical comparisons

If we take a look at Table 2.2 we can see that all this tools are web and cloud-based. Only ElasticRoute does not provide any integration with a mobile application.

Regarding the optimation, the majority of this tools allow to do optimizations by time and distance. However, ElasticRoute only allows to optimize routes by time. Additionaly, OptimoRoute allows to optimize routes by working time. This can be extremely valuable.

OptimoRoute is the only tool that does not allow us to have a multi depot. Track-POD does not allow to save optimizations. These can be considered major flaws.

All tools present the optimization results in the same way: routes displayed on the map and through a driver's work plan.

Finally, none of these tools provide important metrics such as stationary time and empty time. In addition, they are unable to solve the fuel problem.

As far as our solution is concerned, we can take some important notes. We can see the importance of the results presentation. The routes displayed on the map as well as the work plan of each worker should be a must in our application. These are undoubtedly points to be taken into account when gathering the requirements.

Tasks

- **Size of the problems**: Size of the problems the tool can handle, regarding the number of tasks.
- **Type**: Does the tool have different types of tasks?
- **Import tasks**: Can we import tasks from external sources?
- **Balance by drivers**: Can we balance the tasks by driver?

	Routific	**OptimoRoute**	**Track-POD**	**eLogii**	**ElasticRoute**
Size of the problems	2000 tasks per optimization	More than 750 tasks per optimization	N/A	Unlimited	1000 tasks per optimization
Type	✓	✓	✓	✓	✓
Import tasks	✓	✓	✓	✓	✓
Balance by driver	✓	✓	✓	N/A	N/A

Table 2.3: Tools - Tasks - Features and technical comparisons

Table 2.3 shows us that eLogii is way more powerful than the others, when it comes to the number of tasks it can handle. Routific, with 2000 tasks per optimization, looks pretty decent when compared to the rest of the tools. Track-POD does not provide any information about this subject.

Regarding the type of tasks, all the tools provide this feature. In addition, they all can import tasks from external files.

Last but not least, when it comes to balance tasks by driver, eLogii and ElasticRoute do not provide any information about this.

Drivers

- **Skills**: Can we assign skills to drivers?
- **Multi driver**: Can we have more than one driver?
- **Import drivers**: Can we import drivers from external sources?
- **Breaks**: Can we have non-mandatory breaks?
- **Mandatory breaks**: Can we have mandatory breaks?
- **Shifts**: Can we have shifts between drivers?
- **Associated costs**: Can the driver have an associated cost (e.g., 0.89€ per Km) ?

	Routific	**OptimoRoute**	**Track-POD**	**eLogii**	**ElasticRoute**
Skills	✓	✓	✗	✓	N/A
Multi driver	✓	✓	✓	✓	✓
Import drivers	✓	✓	✓	✓	✓
Breaks	Only for meals	Only one break	Only one break	✗	No limit
Mandatory Breaks	✗	✗	✓	✗	✗
Shifts	✗	✗	✗	✗	✗
Associated costs	✗	✓	✓	✗	✗

Table 2.4: Tools - Drivers - Features and technical comparisons

Table 2.4 contains all the information about the Drivers on each tool. All the tools allow to import drivers and also have multiple drivers. These drivers can have skills associated on Routific, OptimoRoute and eLogii.

When it comes to optional breaks, there are some differences. OptimoRoute and Track-POD only let the driver have one break. Routific only allows the driver to have a meal break. ElasticRoute does not have a limit of breaks and eLogii does not provide any type of break. Regarding the mandatory breaks, those that are mandatory by law, only Track-POD has them.

OptimoRoute and Track-POD allows to have associated costs to the drivers.

Finally, none of these tools lets the drivers take turns.

Vehicles

- **Size of the problems**: Size of the problems the tool can handle, regarding the number of vehicles.
- **Capacity**: What capacity dimensions can we assign to the vehicle (e.g., Weight and Volume)?
- **Type**: Can we have types of vehicles (e.g., Refrigerated and Truck)?
- **Import vehicles**: Can we import vehicles from external sources?

	Routific	**OptimoRoute**	**Track-POD**	**eLogii**	**ElasticRoute**
Size of the problems	Unlimited	N/A	N/A	>1000 vehicles	Unlimited
Capacity	Only 1 dimension	Up to 4 dimensions	Yes (not specified)	Yes (not specified)	Weight Volume Seats
Type	✗	✓	✓	✓	✓
Import vehicles	✗	✓	✓	✓	✓

Table 2.5: Tools - Vehicles - Features and technical comparisons

Table 2.5 depicts all the information about the vehicles for each tool.

Regarding the size of the problems, we were not able to gather much. With Routific and ElasticRoute we can have an unlimited number of vehicles. With eLogii, according to the official website, we can have more than 1000 vehicles.

When it comes to the capacities, there are a few differences. Routific only allows to have one configurable dimension. OptimoRoutes lets us have up to 4 dimensions. ElasticRoute has 3 non-configurable dimensions: weight, volume and seats. eLogii and Track-POD do not have any information about the number of dimensions.

Finally, with the exception of Routific, all tools can import vehicles and have types assigned to them.

2.3.3 Interpretation

The present chapter allows us to draw some conclusions regarding our project.

All these tools previously analyzed focus on generic route optimization problems. As we have seen, none of these is capable of solving a more specific problem. For example, the fuel problem.

Each business is different and as such, we need to have this in consideration when building an FSM system. One of these systems can work very well for one company while for another it can have the opposite effect. In other words, many of these tools, despite solving similar problems, may not solve ours.

As mentioned earlier, these tools can solve many problems. However, in most cases it does not solve them in the best way possible. A customized solution that adapts to the business, always has added value. A company that uses a tailor-made solution can easily recover its investment in the tool.

Finally, by doing this analysis we were able to gather a good amount of information on how our tool should look like. Both at the functional and structural level. We were able to see some features that can add value to our tool. Additionally, we can also get the idea of some characteristics that are not desirable, thus we should avoid them.

2.4 Implementation framework

As we will mention further in section 3.3, we are restricted in relation to the tools that we can use in the frontend development. As such, within the two candidates, we must find out which is the best suitable for the job. This section serves to make a comparison between these two technologies, **Angular** and **React**.

The best way to make a comparison of this type is through a table, just like it was previously done when comparing the tools on the market.

It is worth mentioning that these tools, despite having similar objectives, have many differences. A superficial analysis was carried out putting in perspective the main aspects of each one of these tools. The following characteristics were analyzed:

- **Free**: Is the tool free to use?
- **Open source**: Is the tool open source?

- **Company**: What company supports the tool.
- **Release**: Year of release.
- **GitHub Stars**: Number of stars in GitHub
- **Type**: What kind of tool is it (e.g., framework or library)?
- **Data binding**: How is the data binded by the tool?
- **Language**: What kind of language is used?
- **DOM**: What type of Document Object Model (DOM) is used?
- **Learning curve**: How is the learning curve of this tool?
- **Complexity**: How is the complexity of this tool?
- **Flexibility**: How flexible is this tool?

As mentioned before, in this analysis only the main aspects of each tool were taken into account. No exhaustive research was carried out, as it was not worth it. In Table 2.6 we can observe the resulting comparison vector.

	Angular	**React**
Free	Yes	Yes
Open source	Yes	Yes
Company	Google	Facebook
Year	2016	2013
GitHub Stars	69.7k (01-10-2021)	162k (01-10-2021)
Type	Framework	Library
Data binding	Two-way data binding	One-way data binding
Language	TypeScript	JavaScript XML (JSX)
DOM	Incremental DOM	Virtual DOM
Learning curve	High	Low
Complexity	High	Low
Flexibility	Moderate	High

Table 2.6: Comparison - Angular vs React (Adapted from ImaginaryCloud [5])

Regarding **open source** and the fact that they are **free**, both tools are on the same level. Both have big companies supporting them, **Angular** has **Google** and **React** has **Facebook**. However, if we take a look at when they were released, we can see that Angular is 3 years more recent. On the other hand, if we look at the number of stars on GitHub there is almost a 100k difference favoring React.

Regarding the type of tool, there is also a difference between the two here. Angular is a **framework**, while React is a **library**. In a library, we are responsible for the course of our application actions [8]. This does not happen in a framework. This is an important aspect to have in mind, since there is a small degree of freedom at stake here.

Angular has **two-way data binding**, which means that if we change an input in the UI it will also change the state of a component. This does not happen in React since it has **one-way data binding**.

When it comes to the programming language used there there are also differences. Angular uses **TypeScript** while React uses **JavaScript XML (JSX)**. TypeScript is a superset of JavaScript and can be considered more simple version because it is a typed language. JSX allows us to write HTML code in React.

With regard to **DOM**, Angular uses an **Incremental DOM** while React uses a **Virtual DOM**. Both have their advantages and disadvantages. However, **Virtual DOM** has a slightly better performance.

The **learning curve** it is more steep in Angular. This can be easily explained by the use of the TypeScript language. Additionally, Angular has some concepts that are only used in this framework. Angular is also more complex and in a certain way, more powerful than React.

At this point, it is important to mention that we will use **Django** for the **backend** development. Django will provide the main functionalities of our tool. With that in mind, if we use Angular for the frontend, we will probably end with an heavy application.

In conclusion, we can probably consider that React is the obvious choice. It is the lightest, simplest and flexible of the two tools.

Chapter 3

Requirements

The proper identification of the software requirements is one of the most important processes in a software engineering project. Additionally, the requirements have a significant influence in the architecture of the software to be developed. So, they must be defined a priori, before any decision about the architecture is made. This chapter serves to document those requirements.

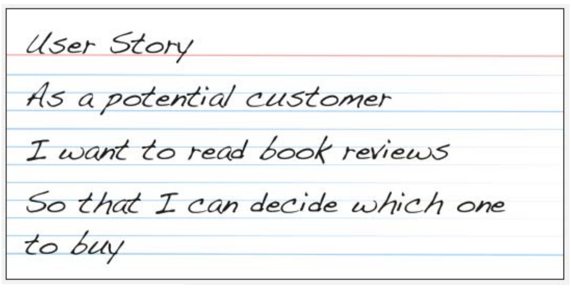
In Section 3.1 we describe the functional requirements, documented via user stories, adopting a high-level view. In Section 3.2 we can observe the Quality Attributes (also known as Non-functional Requirements) captured through scenarios. In Section 3.3 the restrictions to which our project is subject are presented. Section 3.4 states how the quality attributes were prioritized through a voting.

3.1 Functional Requirements

Functional requirements represent the functionalities of our system. They can be defined as the way the system behaves under different conditions [9].

These Functional Requirements (high-level) were gathered through meetings with the company. These meetings were also extremely important to validate them. In addition, the information collected in the state of the art was also taken into account. User stories were used in order to document the functional requirements.

A **user story** is an informal way of describing a feature of a software system through text [10]. This method has the following components: **user**, **task** and **goal**. The **user**, as the name says, is the one interacting with the system. A **task**, is what the user will do in order to achieve a specific **goal**. Lets analyze an example. Imagine a person that it is looking for a book to buy. How can we write a user story for this scenario? Figure 3.1 illustrates an example.



User Story
As a potential customer
I want to read book reviews
So that I can decide which one to buy

Figure 3.1: User story example (from emergn [2])

Table 3.1 represents our functional requirements documented using user stories. We have 18 user stories in total. Since the writing of the intermediate report, two new user stories have been added, with IDs 17 and 18, respectively.

ID	Name	User story
1	Login	As an **unauthenticated user**, I want to securely authenticate myself in the system, in order to be able to access the main dashboard where it is possible to optimize my tasks.
2	Define vehicles	As a **user**, I want to be able to define a set of characteristics for the vehicles (eg weight, type of cargo, compartments, etc ...), in order to be taken into account in the optimization process.
3	Define drivers	As a **user**, I want to be able to define a set of characteristics for drivers (eg schedules, availability days, skills, etc ...), in order to be taken into account in the optimization process.
4	Define tasks	As a **user**, I want to be able to define the work to be planned (set of tasks), in order to get it optimized taking into account the restrictions imposed.
5	Select resources	As a **user**, I want to be able to select the resources I want to use in my work plan, in order to achieve my business goals.
6	Interpret Gantt	As a **user**, I want to be able to analyze the result of the work plan, in a Gantt diagram format (with information such as waiting time, loading time, working time, etc.), in order to be able to assess the quality of the work. solution.
7	Interpret KPI	As a **user**, I want to be able to analyze the key performance indicators related to the optimized work plan, in order to be able to assess the quality of the solution.
8	Interpret routes	As a **user**, I want to be able to analyze the route of the work plan on the map, in order to be able to assess the quality of the solution found.
9	Save plan	As a **user**, I want to be able to save the optimized work plan so that I can consult it later.
10	Erase plan	As a **user**, I want to be able to delete the optimized work plan, as it does not meet my business goals.
11	Export plan	As a **user**, I want to be able to export my final work plan in an appropriate format (eg .csv), in order to be able to use it in another context.
12	Import resources and tasks	As a **user**, I want to be able to import (for example, through a .csv) my resources (vehicles and drivers) and tasks simultaneously, in order to minimize the time for setting up the plan.
13	Compare plans	As a **user**, I want to be able to compare two or more work plans, taking into account their key performance indicators, in order to be able to understand which is the best for my business.
14	Re-optimize plan through Gantt	As a **user**, I want to be able to drag resources on the work plan in the form of a Gantt diagram, in order to get a new work plan that takes into account the changes made.
15	Consult saved work plans	As a **user**, I want to be able to obtain the list of previously saved optimized work plans, in order to be able to use them again.
16	Integration with the Sentilant system	As a **developer**, I want to be able to integrate the web interface with the current Sentilant product, so that it can function as an operational management tool.
17	Internationalization	As a **user**, I want to be able to change the application language.
18	Edit routes	As a **user**, I want to be able to edit the optimized routes (e.g., change drivers and add new tasks).

Table 3.1: Functional Requirements - User stories

3.2 Quality Attributes

When we are defining the functional requirements (high-level or not), we are answering one question: "**WHAT** should the system do?" [11]. However, we need to answer one more question: "**HOW** should the system work?" [11]. Quality Attributes (QA) answer this question.

Quality attributes, also known as **Non-Functional Requirements (NFRs)** can be considered characteristics that the system must have in addition to its functionalities. Non-Functional Requirements usually are in conflict with each other. For that reason they can be very hard to define. [12].

In order to make this process easier, quality attributes scenarios are used. Figure 3.2 illustrates a generic quality attributes scenario. Each scenario has 6 distinct components: **source**, **stimulus**, **artifacts**, **environment**, **response** and **response measure**.

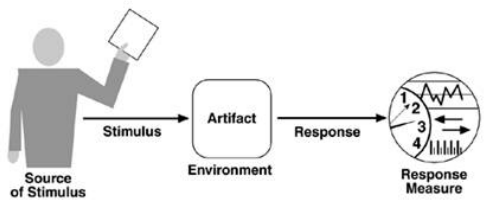


Figure 3.2: Quality Attribute Scenario (from University of New Brunswick [3])

Authentication - Scenario 1

Description: The application prevents access by unauthenticated users. The system checks the user's credentials and prevents them from having access if the credentials are incorrect.

Quality Attribute	Authentication
Source	Unauthenticated user
Stimulus	The user attempts to access the system
Artifacts	System
Environment	Runtime
Response	The system checks the validity of the user's credentials. The user has access to the system if the credentials are valid. Otherwise, access is prevented.
Response measure	There are no unwanted accesses to the system by unauthenticated users

Table 3.2: Scenario 1 - Security

Availability - Scenario 2

Description: In the event that a user needs to obtain an optimized work plan for their tasks, the system must be available and capable of responding to the customer's needs.

Quality Attribute	Availability
Source	User
Stimulus	The user asks the system to optimize a work plan
Artifacts	System
Environment	Runtime
Response	The system presents an optimized work plan to the user
Response measure	The system has a 99.95% availability

Table 3.3: Scenario 2 - Availability

Testability - Scenario 3

Description: Whenever a system functionality is introduced or modified, we need to make sure that no new errors and/or flaws have been introduced.

Quality Attribute	Testability
Source	Developer
Stimulus	The developer wants to introduce changes to the system in production
Artifacts	System
Environment	During development
Response	The change is introduced in the production system
Response measure	No errors were introduced

Table 3.4: Scenario 3 - Testability

Resilience - Scenario 4

Description: The system must be able to detect invalid inputs introduced by the user, whether larger than the supported size or even in incorrect formats, right from the start. These inputs must be detected right at the interface (frontend) in order to avoid being sent to the logical part (backend) thus avoiding possible errors and resulting in a better management of resources.

Quality Attribute	Resilience
Source	User
Stimulus	The user enters a set of invalid tasks and/or resources
Artifacts	System
Environment	Runtime
Response	The system warns the user with a specific error message
Response measure	The system maintains its normal operation

Table 3.5: Scenario 4 - Resilience

Usability - Scenario 5

Description: Each time the user performs an action within the system, he receives feedback on it (eg, error messages, alerts, success messages), so that he can be aware of what is happening.

Quality Attribute	Usability
Source	User
Stimulus	The user asks the system to perform an action
Artifacts	System
Environment	Runtime
Response	The system, if possible, performs the user's action
Response measure	The system provides the user feedback on the action taken

Table 3.6: Scenario 5 - Usability

Usability - Scenario 6

Description: In the need for the user to optimize a work plan, he must be able to have an overview of the same plan on a single page, in order to facilitate its interpretation.

Quality Attribute	Usability
Source	User
Stimulus	The user optimizes a work plan
Artifacts	System
Environment	Runtime
Response	The system presents the optimized work plan solution to the user
Response measure	All information on the optimized work plan is presented on a single page

Table 3.7: Scenario 6 - Usability

Usability - Scenario 7

Description: Possibility for the user to modify the optimization result in order to obtain information in real time about the impact of this change. Example: dragging resources on the Gantt diagram, resulting from the optimization of the work plan, and consequently obtaining a new optimization in real time (user story 14)

Quality Attribute	*Usability*
Source	User
Stimulus	The user interacts with the gantt diagram, resulting from the optimization, in order to introduce changes in the work plan
Artifacts	System
Environment	Runtime
Response	The system, taking into account the changes made, calculates the new work plan
Response measure	The system presents the new work plan to the user

Table 3.8: Scenario 7 - Usability

Compatibility - Scenario 8

Description: The application works correctly and consistently across different types of web browsers.

Quality Attribute	Compatibility
Source	User
Stimulus	The user tries to access the system in a web browser other than the one he usually uses
Artifacts	System
Environment	Runtime
Response	The system allows the user to be able to use the system
Response measure	The user is able to use the system without noticing any difference in its normal operation

Table 3.9: Scenario 8 - Compatibility

Extensibility - Scenario 9

Description: The system must be able to connect to different information points (e.g., databases), without the need to change the source code.

Quality Attribute	Extensibility
Source	System
Stimulus	Need to add a new source of information for the system to retrieve data (vehicles, drivers or tasks).
Artifacts	External source
Environment	Runtime
Response	Developers develop a new independent component to establish the connection
Response measure	There is no need to change the remaining source code

Table 3.10: Scenario 9 - Extensibility

Extensibility - Scenario 10

Description: The system must be able to be embedded in DrivianTasks (Sentilant's current solution) in order to be presented to the user.

Quality Attribute	Extensibility
Source	DrivianTasks
Stimulus	Need to embed the system in DrivianTasks
Artifacts	System
Environment	Runtime
Response	The system is embedded in DrivianTasks
Response measure	Authentication, authorization and connection to tenants is functional

Table 3.11: Scenario 10 - Extensibility

3.3 Constraints

In software architecture design, we may have some restrictions that affect directly and/or indirectly the architecture of the tool to be built. As such, they have to be taken into account before designing the architecture.

These constraints can be one of two types: **Business** or **Technical**. In this project only technical constraints were found.

Technical constraints are impositions that are made during the system design process. As such, these have to be fulfilled. Usually these constraints are provided by the stakeholders of the project [13].

Within Sentilant, certain development tools are used that are already part of the implementation processes. As such, we have to use these tools during this internship. We are therefore facing a technical constraint.

The development of the project is divided into two phases: **backend** and **frontend**. On that account, we were able to identify two technical constraints.

Frontend

Regarding the frontend development of our tool, we were given two options:

- **React**: Open-source JavaScript library developed by Facebook, specialized in web development
- **Angular**: Open-source TypeScript framework developed by Google, specialized in web development

Backend

When it comes to backend development, we must use Django. **Django** is an open-source framework, written in **Python**, also focused on web development.

3.4 Prioritization

Sometimes NFRs are in conflict with each other, resulting in a trade-off. A **trade-off** is when a quality attribute is being affected in a positive way while the other is affected in a negative way [14]. In a nutshel, one is favored and the other is harmed. In order to manage this, is a good practice in Software Engineering to prioritize NFRs.

In order to decide the priority of each one of the quality attributes scenarios previously defined we decided to prioritize them using a voting system.

Each participant was entitled to 10 votes, which he could distribute across the scenarios of each one of the QA. The number of votes was decided based on the number of QA scenarios. We have 10 scenarios, so we have 10 votes. No restrictions were imposed, just that the number of votes did not exceed the previously defined number. Additionally, each vote is anonymous, so there is no type of influence whatsoever.

The voting was carried out using an excel sheet created for this purpose. This sheet was sent to each one of the stakeholders and later returned to me.

Besides me, two more people participated in this vote: **Bruno Cabral (Advisor)** and **Rafael Henriques (Developer)**.

The results obtained can be observed in Table 3.12.

	Votes
Authentication	
Scenario 1	3
Availability	
Scenario 2	8
Testability	
Scenario 3	1
Resilience	
Scenario 4	6
Usability	
Scenario 5	5
Scenario 6	3
Scenario 7	2
Compatibility	
Scenario 8	0
Extensibility	
Scenario 9	0
Scenario 10	2
Total	30

Table 3.12: Vote results

In Table 3.13 we can observe the Quality Attributes Scenarios ordered by priority.

Priority	**Quality Attribute - Scenario**	**Votes**
1	Availability - Scenario 2	8
2	Resilience - Scenario 4	6
3	Usability - Scenario 5	5
4	Authentication - Scenario 1	3
4	Usability - Scenario 6	3
5	Extensability - Scenario 9	2
5	Real time - Scenario 10	2
6	Testability - Scenario 3	1
7	Compatibility - Scenario 7	0
7	Extensibility - Scenario 8	0

Table 3.13: Quality Attributes Scenarios prioritized

We can draw some conclusions by analyzing this table. We can see the importance of some quality attributes scenarios such as Scenario 2 (Availability), Scenario 4 (Resilience) and Scenario 5 (Usability). In addition, we can also observe that there were 3 draws, as such these scenarios have the same priority.

Chapter 4

Tool Overview

From the requirements documented in Chapter 3, we can begin to visualize how our tool will look like. This chapter covers some design aspects as well as some technical concepts of the system to be built.

Section 4.1 discusses some design aspects, through mockups, that are important to analyze before starting the implementation. Section 4.2 provides the reader with an early high-level architecture of our system. Section 4.3 has information about the implementation status of each requirement. It also describes how the system is organized when it comes to code, and what libraries and frameworks were used. Finally, we have a little description on how the system works.

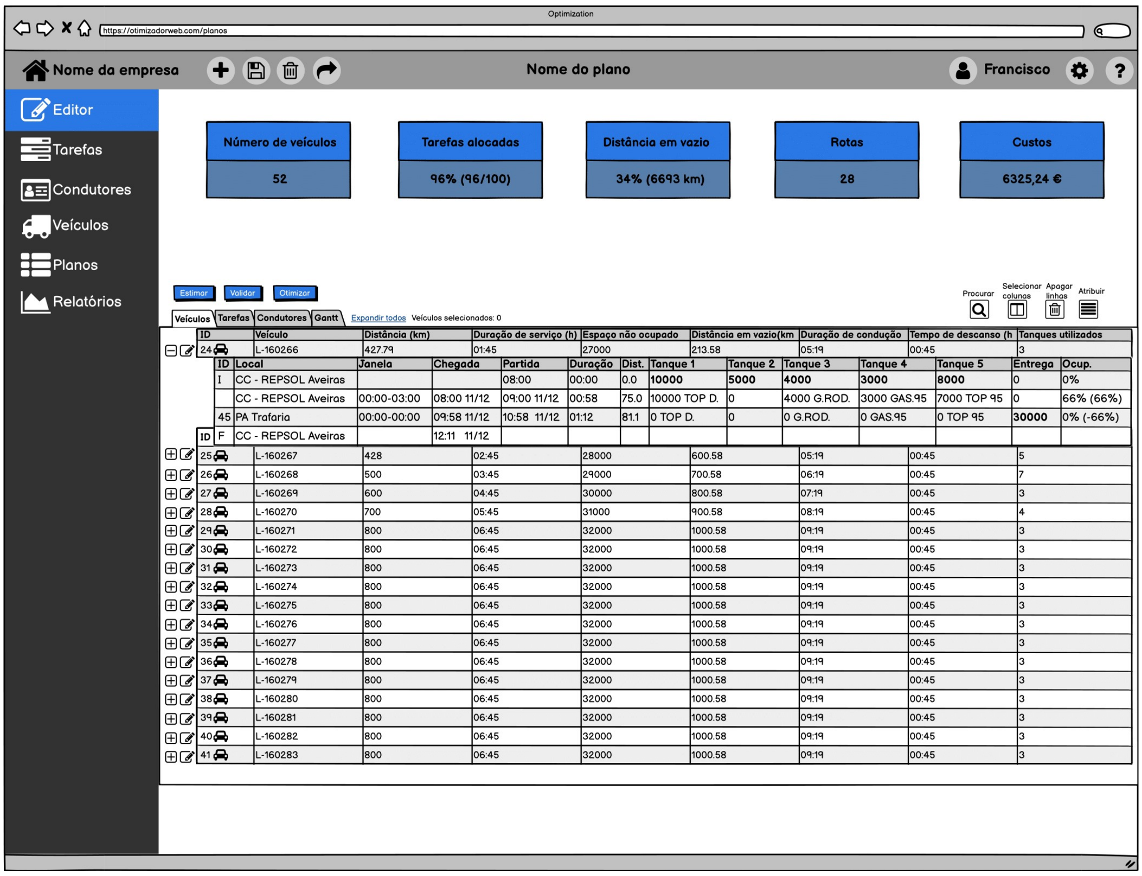
4.1 Design

Since we are building a tool that may be used by customers, there are some precautions to take. In this section we pay special attention to Usability. There are many tools on the market with similar goals. As such, we want to build one that stands out by providing a good user experience. When discussing with my advisor, we came to the conclusion that our tool should follow two basic principles:

- Whenever a user optimizes a work plan, all information related to the optimized work plan should be displayed on a single page

- The main functionalities of the tool should always visible and quickly accessible to the user

With this in mind, 2 **mockups** were built using **Balsamiq**. Figure 4.1 illustrates a possible optimization of a work plan. As mentioned earlier, all relevant information is presented to the user on a single page.



Número de veículos	Tarefas alocadas	Distância em vazio	Rotas	Custos
52	96% (96/100)	34% (6693 km)	28	6325,24 €

Veículos selecionados: 0

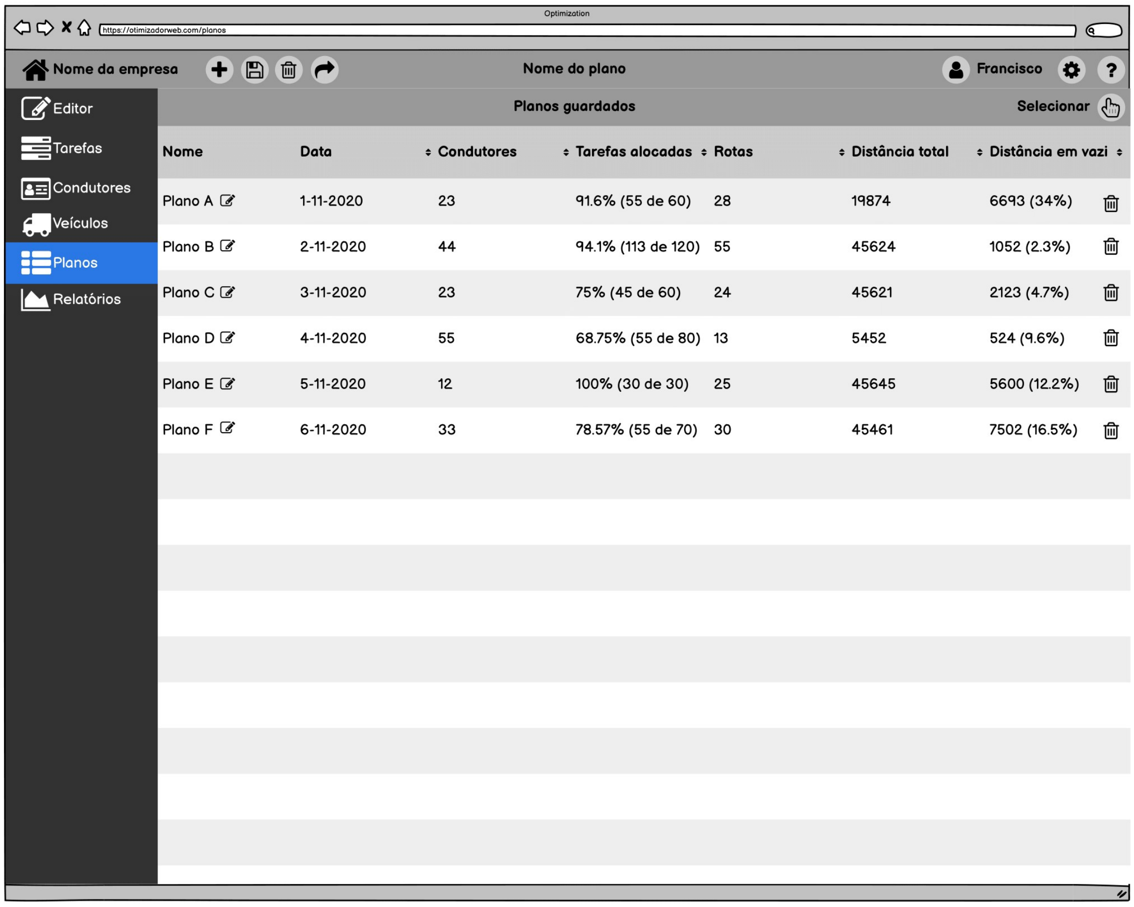
ID	Veículo	Distância (km)	Duração de serviço (h)	Espaço não ocupado	Distância em vazio(km	Duração de condução	Tempo de descanso (h	Tanques utilizados
24	L-160266	427.79	01:45	27000	213.58	05:19	00:45	3

ID	Local	Janela	Chegada	Partida	Duração	Dist.	Tanque 1	Tanque 2	Tanque 3	Tanque 4	Tanque 5	Entrega	Ocup.
I	CC - REPSOL Aveiras			08:00	00:00	0.0	**10000**	**5000**	**4000**	**3000**	**8000**	0	0%
	CC - REPSOL Aveiras	00:00-03:00	08:00 11/12	09:00 11/12	00:58	75.0	10000 TOP D.	0	4000 G.ROD.	3000 GAS.95	7000 TOP 95	0	66% (66%)
45	PA Trafaria	00:00-00:00	09:58 11/12	10:58 11/12	01:12	81.1	0 TOP D.	0	0 G.ROD.	0 GAS.95	0 TOP 95	**30000**	0% (-66%)
ID F	CC - REPSOL Aveiras		12:11 11/12										

ID	Veículo	Distância (km)	Duração de serviço (h)	Espaço não ocupado	Distância em vazio(km	Duração de condução	Tempo de descanso (h	Tanques utilizados
25	L-160267	428	02:45	28000	600.58	05:19	00:45	5
26	L-160268	500	03:45	29000	700.58	06:19	00:45	7
27	L-160269	600	04:45	30000	800.58	07:19	00:45	3
28	L-160270	700	05:45	31000	900.58	08:19	00:45	4
29	L-160271	800	06:45	32000	1000.58	09:19	00:45	3
30	L-160272	800	06:45	32000	1000.58	09:19	00:45	3
31	L-160273	800	06:45	32000	1000.58	09:19	00:45	3
32	L-160274	800	06:45	32000	1000.58	09:19	00:45	3
33	L-160275	800	06:45	32000	1000.58	09:19	00:45	3
34	L-160276	800	06:45	32000	1000.58	09:19	00:45	3
35	L-160277	800	06:45	32000	1000.58	09:19	00:45	3
36	L-160278	800	06:45	32000	1000.58	09:19	00:45	3
37	L-160279	800	06:45	32000	1000.58	09:19	00:45	3
38	L-160280	800	06:45	32000	1000.58	09:19	00:45	3
39	L-160281	800	06:45	32000	1000.58	09:19	00:45	3
40	L-160282	800	06:45	32000	1000.58	09:19	00:45	3
41	L-160283	800	06:45	32000	1000.58	09:19	00:45	3

Figure 4.1: Dashboard - Mockup 1

In Figure 4.2, we can see a potential look of the window where the optimized plans will be saved.



Nome	Data	Condutores	Tarefas alocadas	Rotas	Distância total	Distância em vazi
Plano A	1-11-2020	23	91.6% (55 de 60)	28	19874	6693 (34%)
Plano B	2-11-2020	44	94.1% (113 de 120)	55	45624	1052 (2.3%)
Plano C	3-11-2020	23	75% (45 de 60)	24	45621	2123 (4.7%)
Plano D	4-11-2020	55	68.75% (55 de 80)	13	5452	524 (9.6%)
Plano E	5-11-2020	12	100% (30 de 30)	25	45645	5600 (12.2%)
Plano F	6-11-2020	33	78.57% (55 de 70)	30	45461	7502 (16.5%)

Figure 4.2: Dashboard - Mockup 2

It is easy to see in both mockups that there is a navigation bar on the left side. This navigation bar facilitates the user to access the main features of the application quickly. In addition, these mockups show how our application will likely look in terms of design.

4.2 Architecture

In this section we can find a high-level system architecture, mapped from the requirements. It is important to mention that this architecture only takes functional aspects into account.

In order to design our architecture we used the C4 Model [15]. Although this model has 4 distinct views to document an architecture, only the first 3 were used: Context, Container and Component.

4.2.1 System Context Diagram

This type of Diagram (System Context) is a good starting point to design a software architecture. We can observe the system as if we were an outside spectator. In this diagram we can see our system, the systems with which it interacts and its users.

Figure 4.3 illustrates the System Context Diagram for our system.

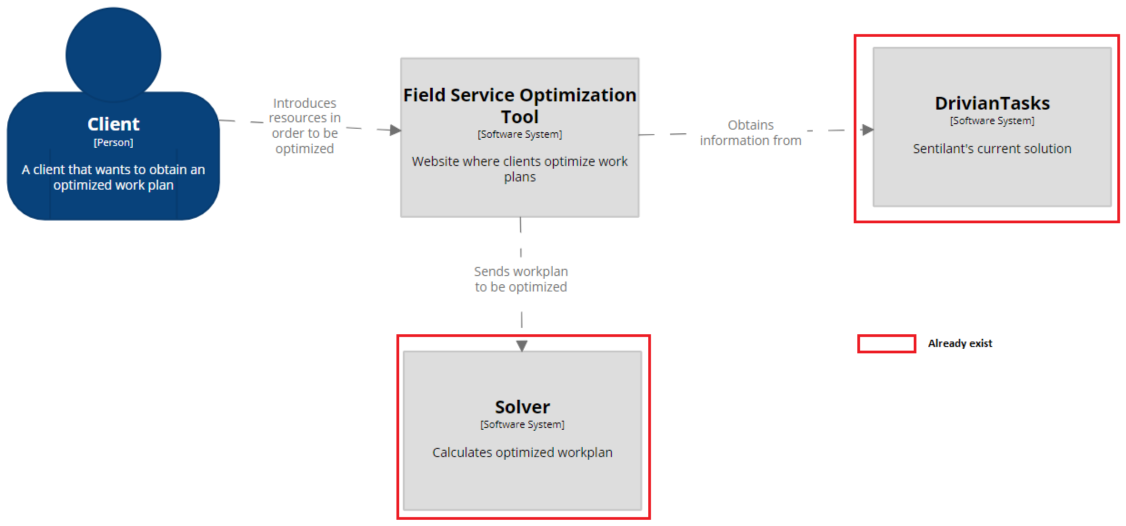

Client
[Person]
A client that wants to obtain an optimized work plan
Introduces resources in order to be optimized
Field Service Optimization Tool
[Software System]
Website where clients optimize work plans
Obtains information from
DrivianTasks
[Software System]
Sentilant's current solution
Sends workplan to be optimized
Solver
[Software System]
Calculates optimized workplan
Already exist

Figure 4.3: System Context Diagram

At the center, we have our system, **Field Service Optimization Tool**. This system represents the web application with which the user (**Client**) will interact in order to obtain their optimized work plans. This plan is sent to another system (**Solver**) where it will be optimized. In addition, we have another system from which we can retrieve and/or send information, **DrivianTasks**. DrivianTasks is Sentilant's current solution.

4.2.2 Container Diagram

If we zoom-in on our system (**Field Service Optimization Tool**) we can obtain a Container Diagram. In a nutshell, a container is like a separate application or even a data store. Each system is composed by several containers. Additionally, in this Diagram we can also see how this containers communicate with each other.

Figure 4.4 illustrates the Container Diagram for our system.

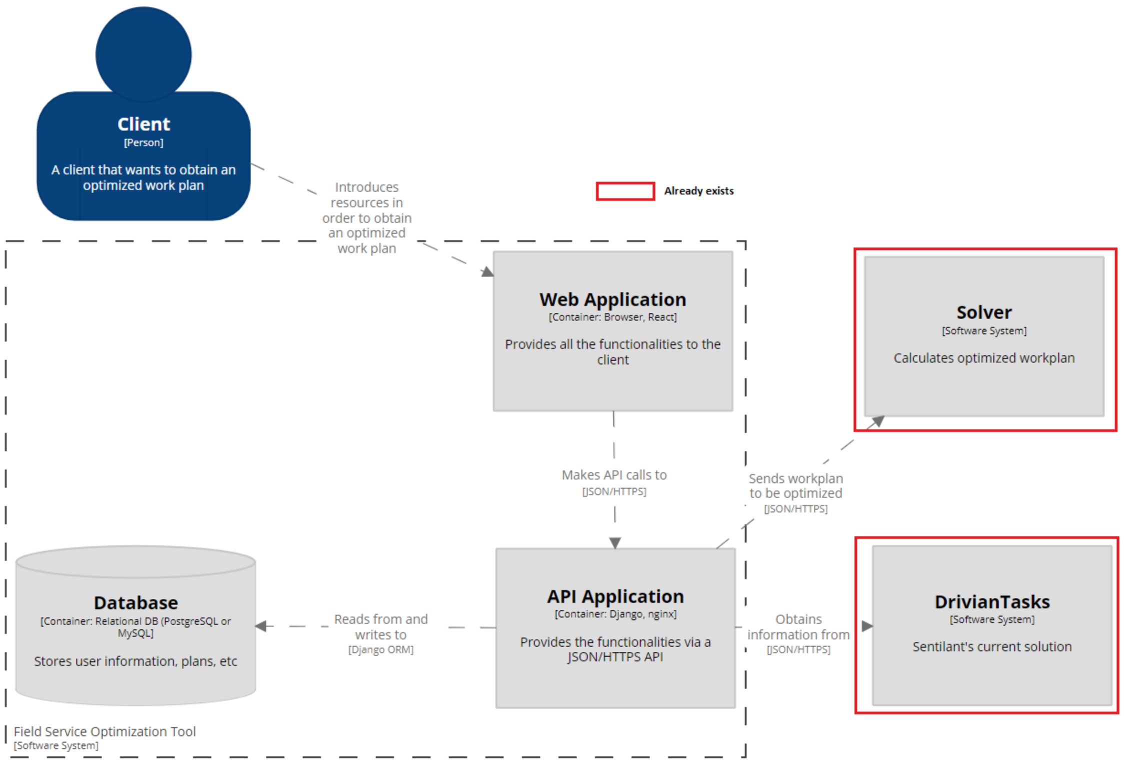

Client
[Person]
A client that wants to obtain an optimized work plan
Introduces resources in order to obtain an optimized work plan
Already exists
Web Application
[Container: Browser, React]
Provides all the functionalities to the client
Solver
[Software System]
Calculates optimized workplan
Makes API calls to
[JSON/HTTPS]
Sends workplan to be optimized
[JSON/HTTPS]
Database
[Container: Relational DB (PostgreSQL or MySQL)]
Stores user information, plans, etc
Reads from and writes to
[Django ORM]
API Application
[Container: Django, nginx]
Provides the functionalities via a JSON/HTTPS API
Obtains information from
[JSON/HTTPS]
DrivianTasks
[Software System]
Sentilant's current solution
Field Service Optimization Tool
[Software System]

Figure 4.4: Container Diagram - Field Service Optimization Tool

The **Web Application** container represents the interface (website) the user will interact with. To obtain full access to this interface the user needs to authenticate first. This Web Application is where the work plan is introduced in order to be optimized. It also serves to observe the results obtained. This container is the most important one when it comes to Usability, one of our most important NFRs. This container is in constant communication with the API Application via a JSON/HTTPS API.

The **API Application** is responsible for communicating with the rest of the systems (Solver and DrivianTasks) and with the last container, the Database. The **Database** is where all our data will be stored, such as: user's credentials, previously optimized work plans and so on. These two containers are mainly responsible for the two NFRs with the highest priority in our application: Availability and Resilience.

Regarding the main responsibilities and the technologies used within these containers we have the following:

- **Web Application**: Responsible for providing all the functionalities to the client (e.g., login and optimize work plan). It will make API calls using JSON/HTTPS to the API Application in order to obtain the functionalities. Will be developed using React.

- **API Application**: Responsible for the main functionalities of the application. Main point of communication. Will be sending the plans to the solver, via JSON/HTTPS, in order to be optimized. In addition, it will also be in constant communication with DrivianTasks to obtain information. Finally, interacts with the Database using Django ORM, in order to read or persists data. This container will be built using Django and will run on a nginx web server.

- **Database**: Responsible for persisting all the information of our application (e.g., credentials, work plans and resources). It will be a Relational DB, probably using PostgreSQL or MySQL.

4.2.3 Component Diagram

If we follow the same approach, and zoom-in on each one of our containers, we can decompose them in components. A component can be defined as the major structural building blocks within a container. In addition, we also take into account their interactions.

In this subsection we divided the two main containers of our system: **Web Application** and **API Application**. The database was not decomposed because it was not worth it.

Web Application

As mentioned in the previous section, the Web Application container is responsible for interacting with the user. By doing so it provides all the functionalities of our application.

Figure 4.5 illustrates the Component Diagram for our Web Application.

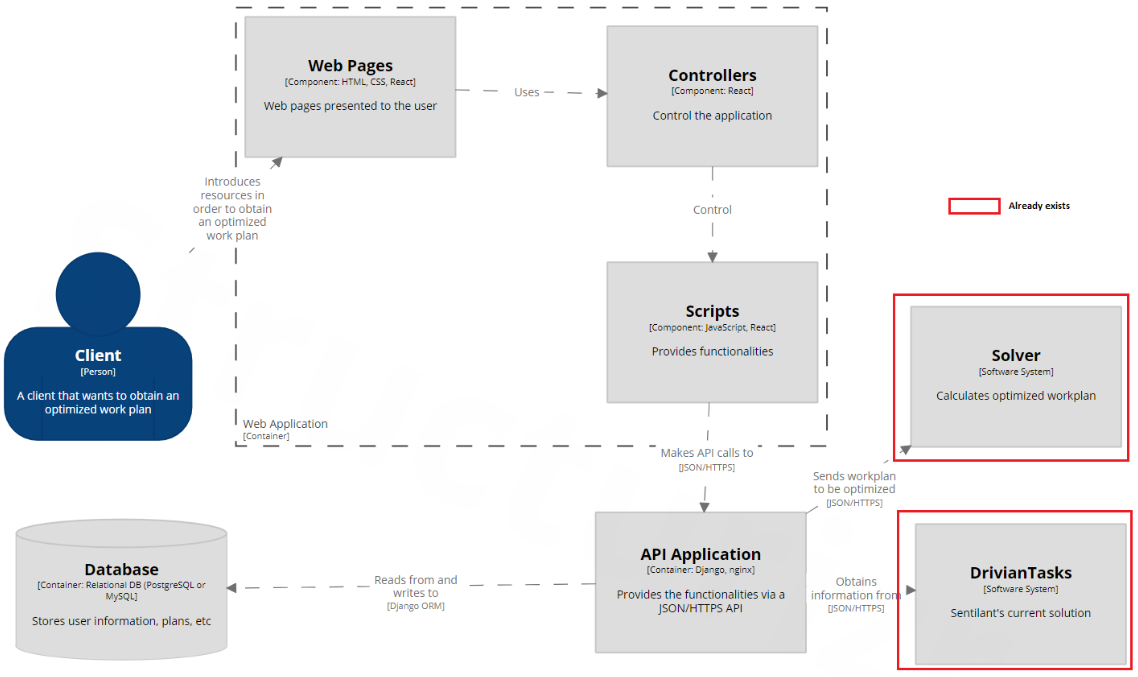

Web Pages
[Component: HTML, CSS, React]
Web pages presented to the user
Uses
Controllers
[Component: React]
Control the application
Control
Scripts
[Component: JavaScript, React]
Provides functionalities
Introduces resources in order to obtain an optimized work plan
Client
[Person]
A client that wants to obtain an optimized work plan
Web Application
[Container]
Already exists
Solver
[Software System]
Calculates optimized workplan
Makes API calls to
[JSON/HTTPS]
Sends workplan to be optimized
[JSON/HTTPS]
Database
[Container: Relational DB (PostgreSQL or MySQL)]
Stores user information, plans, etc
Reads from and writes to
[Django ORM]
API Application
[Container: Django, nginx]
Provides the functionalities via a JSON/HTTPS API
Obtains information from
[JSON/HTTPS]
DrivianTasks
[Software System]
Sentilant's current solution

Figure 4.5: Component Diagram - Web Application

We have the following components:

- **Web Pages**: They represent all web pages presented to the user. HTML, CSS and React will be used to build them.

- **Controllers**: Responsible for controlling the flow of our web application. React will be used for this aspect.

- **Scripts**: All the scripts that provide functionalities in some way. JavaScript and React will be used to develop them.

API Application

The API Application container is responsible for providing the main functionalities of our application. In order to do this, it also interacts with other external systems.

Figure 4.5 illustrates the Component Diagram for our Web Application.

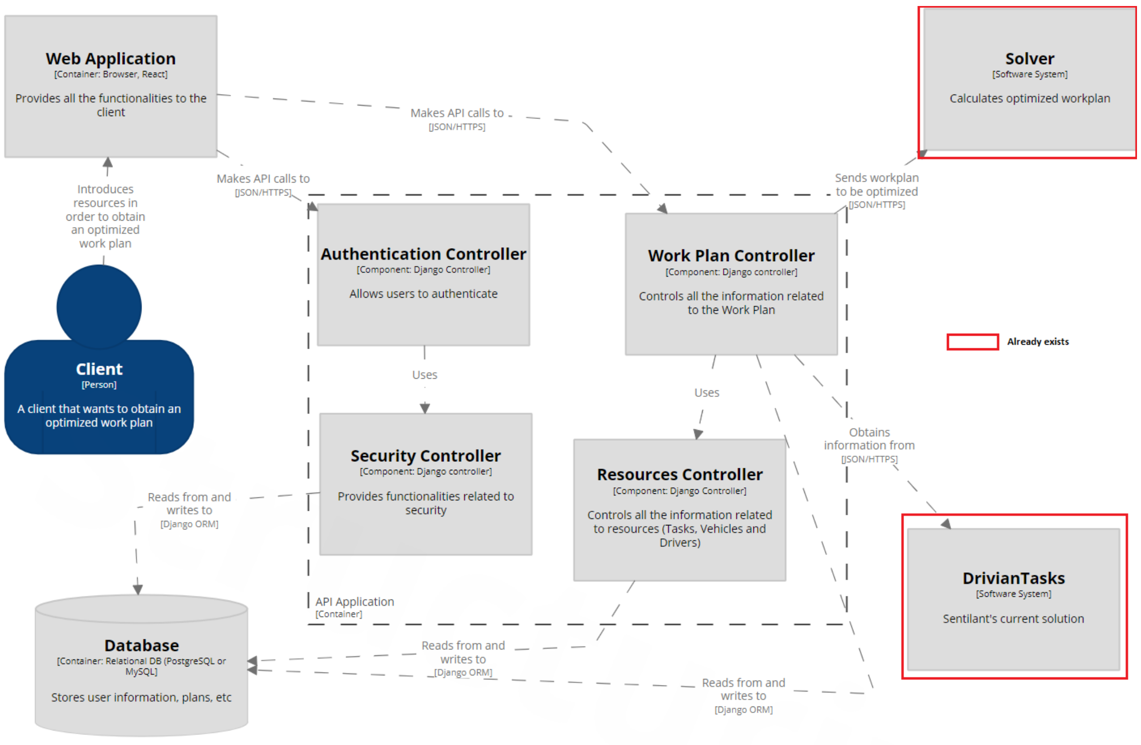

Web Application
[Container: Browser, React]
Provides all the functionalities to the client
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Solver
[Software System]
Calculates optimized workplan
Sends workplan to be optimized
[JSON/HTTPS]
Introduces resources in order to obtain an optimized work plan
Client
[Person]
A client that wants to obtain an optimized work plan
Authentication Controller
[Component: Django Controller]
Allows users to authenticate
Uses
Security Controller
[Component: Django controller]
Provides functionalities related to security
Work Plan Controller
[Component: Django controller]
Controls all the information related to the Work Plan
Already exists
Uses
Resources Controller
[Component: Django Controller]
Controls all the information related to resources (Tasks, Vehicles and Drivers)
Obtains information from
[JSON/HTTPS]
DrivianTasks
[Software System]
Sentilant's current solution
Reads from and writes to
[Django ORM]
API Application
[Container]
Database
[Container: Relational DB (PostgreSQL or MySQL]
Stores user information, plans, etc
Reads from and writes to
[Django ORM]
Reads from and writes to
[Django ORM]

Figure 4.6: Component Diagram - API Application

We have the following components, all built using Django:

- **Authentication Controller**: Allows users to authenticate themselves in order to access the application.
- **Work Plan Controller**: Responsible for controlling all the information related to the work plans to be optimized. Needs to communicate with the external solver.
- **Security Controller**: Makes sure that the authentication process is done in a secure way.
- **Resources Controller**: Responsible for controlling all the information regarding the resource of our application, such as: Tasks, Vehicles and Drivers.

4.3 Software development

In this section, we document all the developments carried out throughout the implementation phase. In addition, we also describe how the system is structured and how it works.

4.3.1 Implemented requirements

In **Chapter 3**, 18 high-level functional requirements were documented through user stories. However, not all have been implemented. This can be easily explained. Some requirements, although useful, do not add significant value to the tool. In addition, since the intermediate report, 2 new functional requirements emerged. Lastly, some of these took a lot more of work to implement than might have been expected. As a consequence, some were given more priority than others.

Below, Table 4.1 contains the list with all the functional requirements and the respective implementation status.

ID	User story name	Status
1	Login	Not implemented
2	Define vehicles	Finished
3	Define drivers	Finished
4	Define tasks	Finished
5	Select resources	Partially
6	Interpret Gantt	Finished
7	Interpret KPI	Finished
8	Interpret routes	Finished
9	Save plan	Finished
10	Erase plan	Finished
11	Export plan	Not implemented
12	Import resources and tasks	Finished
13	Compare plans	Finished
14	Re-optimize plan through Gantt	Not implemented
15	Consult saved work plans	Finished
16	Integration with the Sentilant system	Not implemented
17	Internationalization	Finished
18	Edit routes	Finished

Table 4.1: Functional requirements - Development status

4.3.2 System description

As mentioned earlier, our tool is mainly made up of two distinct parts: the backend, built using Django, and the frontend, built in React. In addition, whenever a new plan needs to be optimized, we call an external solver API, which is currently inside a docker container. The solver, receives a list of resources (vehicles and drivers) and tasks, and returns the optimized workplan. Since this solver is outside the context of this internship, we will not get into details on how it works.

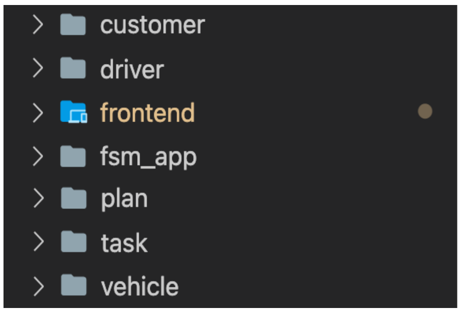
In our tool, we can have multiple clients using the same web application and at the same time, their data is totally isolated. In other words, a client can only see the data that concerns him. This is the definition of multi tenancy, an important principle for this project.

In order to achieve this multi tenant system, we needed to use a django application called **django-tenants** (`https://github.com/django-tenants/django-tenants`). This application allows us to have multiple tenants with isolated data. This is achieved via different **PostgreSQL schemas**, one for each client.

Regarding the frontend, we use React to build our web application interface. Furthermore, all the decorative elements are from the **material-ui framework** (`https://material-ui.com/`)

Lastly, together with React we used **Webpack** (`https://webpack.js.org/`) as a module bundler and **Babel** (`https://babeljs.io/`) as a Javascript transpiler. In addition, **leaflet** (`https://leafletjs.com/`) was used to build the map dialog that is presented to the user. **React Google Charts** (`https://react-google-charts.com/`) was used to build the Gantt chart with all the tasks that a driver has.

When it comes to the organization of the code, Figure 4.7 shows how it is structured. This, adopting a high-level view, without great detail.



```
> customer
> driver
> frontend
> fsm_app
> plan
> task
> vehicle
```

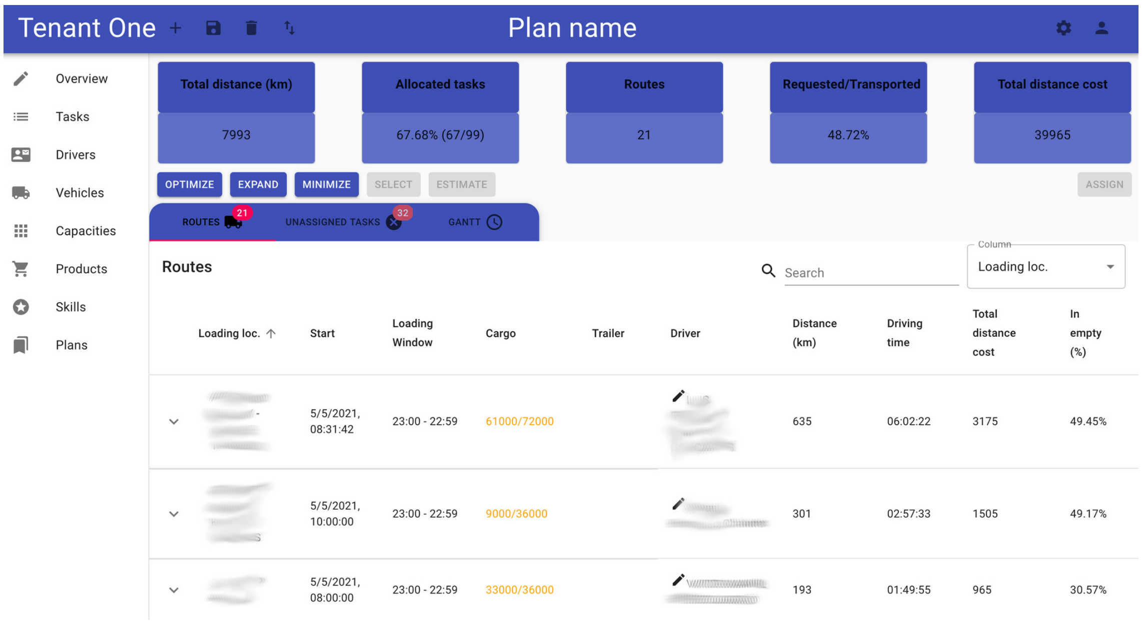
Figure 4.7: Code structure

Each one of these folders, by definition, is a **django application**. The ***fsm_ app*** represents the main application, where we have the structural logic of our application. With the exception of the frontend folder, all the others represent one of the main entities of the system. The ***frontend*** folder, despite being a django application, contains all the code related to the frontend. That is, all React components with which the user interacts.

4.3.3 System features

As mentioned in the previous subsection, each tenant has an isolated schema with all its data. With that said, lets analyze some of the main functionalities.

Figure 4.8 depicts the Overview page, which is the main page of the system. This page already has an optimized work plan. For privacy reasons, some data fields were blurred, since we are dealing with a real test case.



Total distance (km)	Allocated tasks	Routes	Requested/Transported	Total distance cost
7993	67.68% (67/99)	21	48.72%	39965

Routes

Column: Loading loc.

Loading loc.	Start	Loading Window	Cargo	Trailer	Driver	Distance (km)	Driving time	Total distance cost	In empty (%)
[redacted]	5/5/2021, 08:31:42	23:00 - 22:59	61000/72000		[redacted]	635	06:02:22	3175	49.45%
[redacted]	5/5/2021, 10:00:00	23:00 - 22:59	9000/36000		[redacted]	301	02:57:33	1505	49.17%
[redacted]	5/5/2021, 08:00:00	23:00 - 22:59	33000/36000		[redacted]	193	01:49:55	965	30.57%

Figure 4.8: Overview page

The navigation sidebar allows the user to move freely between pages. The header component has the name plan as well as some action buttons that can be used to save, erase or create a new plan. In addition, it also has the settings button where we can change the language. Whichever page we are on, these components are always accessible.

In the "Overview" page we have all we need to know about a plan. On the top we have a few cards with some KPIs such as total distance and percentage of allocated tasks. In the middle of the screen, we have a tab panel that is composed by 3 main components: "Routes", "Unassigned tasks" and "Gantt".

The "Routes" tab contains a customized table with all the optimized routes in the plan. Each route has a driver assigned. In addition, it also has some metrics such as driving time and total distance. If we click on the arrow icon on the left, the route is expanded. When the route is expanded we can see all the tasks that route has. Here we can take some actions such as edit task, insert new task and delete task. We can also see the products and respective quantities that are being loaded/unloaded. There is also a special button that opens a popup dialog with a map, Figure 4.9. This map has all the locations of the tasks that compose the route being analyzed. **Blue** represents the loading location. **Green** represents the unloading locations.

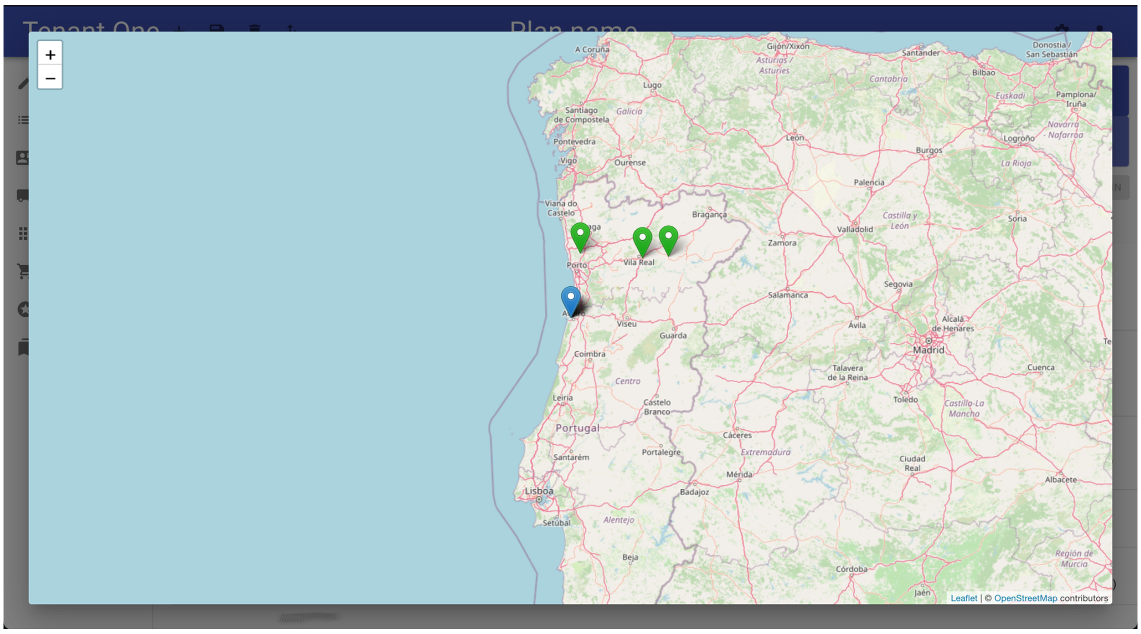


Figure 4.9: Map dialog

In the "Unassigned Tasks" tab has all the tasks that were not assigned. Here, we can assign a task manually to a driver.

Lastly, the "Gantt" tab contains all the drivers in a route, with the respective tasks distributed over time.

Regarding the optimization process, we have some buttons below the cards. The main button, "Optimize" sends all the resources (vehicles and drivers) and tasks to the external solver API in order to receive the optimized work plan. Whenever there is any change in the routes already optimized that requires re-optimization, the "Estimate" button becomes active.

All the remaining pages available on the navigation sidebar, such as "Tasks", "Drivers", "Vehicles", "Capacities", "Products" and "Skills" allow the user to import these entities into the system. In order to do this, the user has to select a .csv file with a specific format. However, all data that is in the system will be used when it is time to optimize the plan. A future implementation would be a feature where the user could be able to choose which resources and tasks he wants to select.

The "Plans" page has a list with all the previously saved plans, as well as some general metrics about each one.

Chapter 5

Validation

In the development of a software engineering project, two very important questions are raised. As defined by Barry Boehm, we have to ask ourselves the following: "Did we build the product right?" and "Did we build the right product?" [16]. The answer to these questions is directly related to the terms verification and validation, respectively.

Throughout this document, we have been answering the first question. Activities previously done, such as requirements gathering **(Chapter 3)** and architecture design **(Chapter 4)** are part of this process.

At this point, we need to make sure that the system is able to fulfill its purpose. Which means, perform the tasks for which it was designed. In other words, we need to validate our tool. We have to validate if the objectives previously defined, were met. With that in mind, a dynamic approach through testing was adopted.

This chapter is composed by three different sections. In section 5.1 we test the simplest components of the system, through unit testing. Section 5.2 contains integration testing in which we test how these components interact with each other. Finally, in section 5.3 the system is tested as a whole. To do this, we check if the requirements previously defined are fulfilled. Since the usability is quite important in the context of our solution, in Section 5.4 we measure the usability levels with a usability checklist.

5.1 Unit testing

Unit testing consists in testing a small part of the system in isolation, which we call unit. In other words, we test the simplest components that make our system, separately.

We can divide our django backend project into 4 separate components: driver, vehicle, task and plan. Each of these concerns one of the main entities of our system. These components can be divided in 3 main parts:

- **models.py** - concerns the data model
- **views.py** - concerns the requests and responses
- **urls.py** - concerns the urls of the views

Each component, also known as django app, has a folder named tests which contains 3 files. Each file has the test cases for each part. We can observe an example below, Figure

5.1

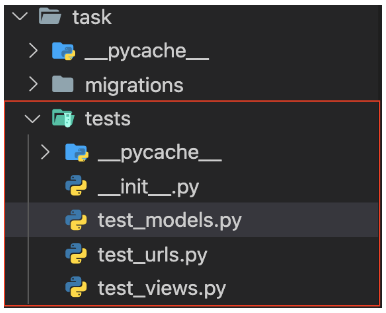


Figure 5.1: Test folder structure for Task application

In this section, since we are only performing unit testing, we will only focus on testing models.py and urls.py. The tests that concern views.py will be described in the next section. The reason for this is that views.py involve interactions between different units. As such, this is outside the scope of unit testing.

One of the advantages of unit testing is that if we need to make any change, we are able to guarantee that it did not affect what was already done. In order to do that we just need to rerun the tests. This is known as **regression testing**.

A **pass/fail criteria** was defined, in order to know if a test is valid or not. With regard to the pass/fail criteria, we consider that a test has passed as long as it has an outcome equal to the expected outcome. In addition, if a test throws an error, the test is not accepted.

In order to run these tests we used **Django's test-execution framework**. However, since we built a multitenant system we also needed to use django-tenants test package [17].

This type of tests were written along with the code. As such, it made no sense to use any coverage criteria. Finally, all these tests are done through the use of assertions.

In the tables bellow we can observe the unit tests results for the entity Driver. Table 5.1 concerns the models. Table 5.2 concerns the urls.

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Result
Driver_U_M1	Create driver	test_driver_is_created	Driver created	Driver created	Pass
Driver_U_M2	Create skill and add to driver	test_driver_has_skill	Driver has a skill associated	Driver has a skill associated	Pass
Driver_U_M3	Delete driver	test_delete_driver	Driver is deleted	Driver is deleted	Pass
Driver_U_M4	Remove skill from driver	test_delete_skill_from_driver	Skill is removed from driver	Skill is removed from driver	Pass
Driver_U_M5	Edit driver	test_edit_driver	Driver information is edited	Driver information is edited	Pass
Driver_U_M6	Add workblocks to driver	test_add_workblocks_to_driver	Workblock is added to driver	Workblock is added to driver	Pass
Driver_U_M7	List drivers	test_list_drivers	All drivers are shown	All drivers are shown	Pass

Table 5.1: Driver - Models - Unit testing

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Result
Driver_U_U1	Create driver url	test_driver_is_created	CreateDriverView is called through the url	CreateDriverView is called through the url	Pass
Driver_U_U2	Create skill url	test_create_skill_url	CreateSkillView is called through the url	CreateSkillView is called through the url	Pass
Driver_U_U3	List drivers url	test_list_drivers_url	ListDriversView is called through the url	ListDriversView is called through the url	Pass
Driver_U_U4	Get driver by id url (when given a driver id)	test_get_driver_by_id_url	GetDriverByIDView is called through the url	GetDriverByIDView is called through the url	Pass
Driver_U_U5	Edit driver	test_edit_driver	Driver information is edited	Driver information is edited	Pass
Driver_U_U6	Add workblocks to driver	test_add_workblocks_to_driver	Workblock is added to driver	Workblock is added to driver	Pass
Driver_U_U7	List drivers	test_list_drivers	All drivers are shown	All drivers are shown	Pass

Table 5.2: Driver - Urls - Unit testing

A total of 39 test cases were used for unit testing. 22 of these are related to models, and the remaining 17 to urls. If we take a look at Appendix A we can observe all the test cases that were made for unit testing the Models. We have 4 distinct tables, one for each entity (Driver, Task, Vehicle and Plan). Additionally, Appendix B, contains all the test cases for unit testing the Urls.

5.2 Integration testing

Integration testing consists in testing the units once these are combined. In these type of tests, interactions between different units are under analysis.

As mentioned earlier, in this section we will focus on testing the views.py. In a nutshell, a django view receives a web request and returns a web response. Basically, corresponds to the interaction between different units tested in the previous section, such as models and urls.

We are testing the views mentioned in the previous section. A total of 17 test cases were used. The same amount of test cases used for unit testing the urls.

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Result
Driver_I_V1	Sends a POST request to create a driver	test_create_driver_POST	Web response that confirms the driver is created	Web response that confirms the driver is created	Pass
Driver_I_V2	Sends a POST request to create a skill	test_create_skill_POST	Web response that confirms the skill is created	Web response that confirms the skill is created	Pass
Driver_I_V3	Sends a GET request to list all drivers	test_list_drivers_GET	Web response that lists all drivers	Web response that lists all drivers	Pass
Driver_I_V4	Sends a GET request to obtain driver with a specific id	test_get_driver_by_ID_GET	Web response that lists the driver with that ID	Web response that lists the driver with that ID	Pass

Table 5.3: Driver - Views - Integration testing

The Appendix C contains the rest of the test cases for integration testing. We also have 4 tables, one for each different entity, with the respective test cases.

5.3 System testing

In System testing, the system is tested as a whole. It is one of the last stages of the validation process, and one of the most important. In this type of tests we check whether the requirements have been met or not. In order to do this, based on the requirements in Chapter 4, we created a list of functional tests. As such, we are under a functional testing approach.

In addition, it is worth mentioning that the input used throughout this testing methodology corresponds to a real-world example. This is extremely important at this point, since we are trying to validate our tool.

Before starting, we need to have a **coverage criteria**. Coverage criteria can be defined as the amount of program that is examined in a test suite [18]. Since it is impossible to test all the inputs a program can take, we need to define this criteria prior to testing. Normally, the criteria defined is the percentage of code executed by a test suite. In this case, instead of measuring the percentage of code executed, we will cover all the main functionalities.

In Table 5.4 we have an example of a test case. The rest of the test cases can be found attached to this document, Appendix D. In this appendix we a test case for each one of the main functionalities.

Test ID	**F_TC_1**
User Story	2. Define vehicles 12. Import resources and tasks
Description	The user has a list of vehicles, with the appropriate characteristics, that are imported into the system
Pre conditions	None
Steps to reproduce	1. List of vehicles and characteristics are in a .csv file (in a specific format) 2. Navigate to "Vehicles" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button
Expected Result	A message appears saying the vehicles were successfully inserted into the system
Actual Result	A message appears saying the vehicles were successfully inserted into the system
Result	Pass

Table 5.4: Test Case 1 - Functional Testing

A total of 17 test cases were used. All tests passed successfully, which allows us to take an important step towards the validation of our tool.

Furthermore, just like mentioned earlier, the input used corresponds to an input from a real client from Sentilant, in the fuel problem. The fact that this input gives the results we were expecting also helps in the validation process.

5.4 Usability checklist

Since the scope of this internship is a frontend tool, usability is an important factor to be taken into account. In addition to fulfill the goals for which it was designed, the system has to be user-friendly. In a nutshell, it has to be easy for the end user to understand how the tool works.

At this point, the best thing to do would be a usability test. However, since we are talking about a relatively small project, this process was not carried out. As an alternative, a usability checklist was applied. This list allows us to check if the most important usability principles are being fulfilled.

In order to do that, we used a usability checklist made by Sapo (`https://ux.sapo.pt/checklists/usabilidade/`). This UX checklist was applied and later translated to English for consistency reasons with the present document.

Although some do not apply, the checklist is divided into the following topics:

- Navigation and feedback
- Layout
- Readability
- Forms and messages

- Help
- Social networks
- Responsive
- Performance

This checklist proved to be quite useful. It allowed us to correct some points that were not being fulfilled. For example, we were not providing any information on user's location when navigating through the website. Thanks to this process we were able to detect this problem and fix it. We can observe this principle being applied bellow, Table 5.5.

ID	Name	Description
2	Feedback on user location is provided	We must always inform the user's location, preferably through a clear visual feedback in the menu, indicating the section in which the user is.

Table 5.5: Navigation and feedback - Feedback on user location is provided.

If at least **80%** of these metrics are met, we can say that our tool is within acceptable usability levels.

There are a total of 28 principles. However, it makes no sense to apply 6 of them to our tool. That leaves us with a total of 22. Our tool complies with 18 of these rules, which corresponds to approximately 82%. With that said, we can assume that our usability levels are within the standards.

In addition to this checklist, during the implementation process we always tried to have in mind the two principles defined in Section 4.1.

If we take a look at the state of our solution, we can easily see these two principles are being fulfilled. All the information about an optimized work plan is displayed to the user in one page. This is very important as it makes its interpretation easier for the end user. In addition, all the main functionalities are always visible and and can be easily accessed.

In conclusion, our solution passed all the tests carried out, along with the fact that it has the minimum usability levels. This allows us to validate our tool with some confidence.

Analogously to what happened with the tests, the rest of this checklist is attached to this document, Appendix E. This appendix is organized by the usability topics previously listed. If a principle is colored **green** means it is verified. Otherwise, it is colored **red**.

Chapter 6

Internship Planning

This chapter serves to document the internship planning. Section 6.1 shows the processes adopted in terms of software development, in the context of the company. In Section 6.2 we can find the work done in the first semester and in the second semester, respectively. Section 6.3 contains a small risk analysis, in order to follow the best software engineering practices.

6.1 Development Methodology

In software development, when planning, it is very important to choose an appropriate development methodology. With this in mind, the methodology used by Sentilant is Agile.

Agile is an iterative process in which a team divides a project in several parts (sprints) in order to continuously improve the product together with the stakeholders [19].

Figure 6.1 illustrates how this kind of methodology works. The **Product Backlog** is our starting point and represents the requirements, through user stories, to be implemented in the project. From the Product Backlog, some user stories are selected to be part of the **Sprint Backlog**. Before a Sprint begins, the Sprint Backlog is defined in a meeting usually referred as **Sprint Planning**. A **Sprint** is a period of time (2-4 weeks) in which a team completes the tasks previously assigned. These tasks correspond to the Sprint Backlog. During a Sprint it is common to have daily meetings to monitor progress within the team. Whenever a Sprint is finished, the product is incremented. This process is repeated until the product is finished.

This methodology is based on an iterative process, as such it has some advantages. The main reason is that since it is very flexible, it is much easier to respond to possible changes that may arise throughout the project.

In an Agile methodology we have to assign roles to the members involved in this process. We are a small group, so some adaptations had to be made.

In this methodology I have two roles: **Team Lead** and **Developer**. This project is being carried out in the context of my curricular internship, so I will be the only doing all the development. For the same reason, I can also be considered the Team Lead, since the responsibility for the success of the project is mine.

As a **Developer** I was in charge of product design, implementation and testing. As a **Team Lead** I have to make sure that everything is going as expected. Otherwise, I have to make the necessary changes.

Additionally, the rest of the company will act as **Stakeholders**. Their main job is to ensure that the project meets its business objectives. They are also responsible for the management of the Product Backlog.

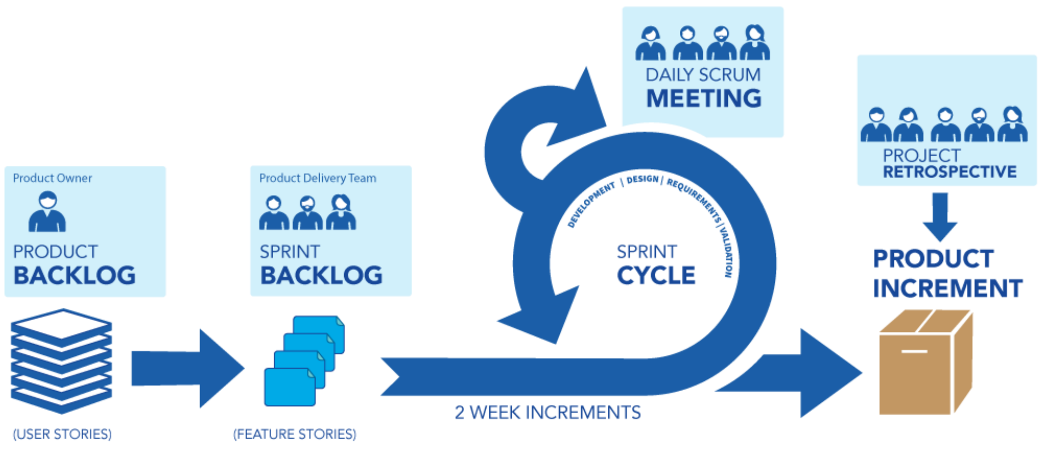


Figure 6.1: Agile methodology (from Bits in Glass [4])

6.2 Planning

In this section we can find out how the internship was planned over time. We can observe the approximately 10 months of internship divided by two semesters.

6.2.1 First Semester

As planned, the internship started in the middle of September, with a face-to-face meeting with the entire company. This meeting took place in a meeting room at Instituto Pedro Nunes (IPN), where Sentilant is based. It is important to mention that, taking into account the context of the COVID-19 pandemic, all Direção Geral de Saúde (DGS) recommendations were followed throughout the meeting.

To mitigate the risk of the pandemic, **Slack** and **Zoom** were used as the main means of communication. Slack mainly for quick conversations and scheduling meetings. Zoom was used to hold video conference meetings.

At the meeting, in addition to introducing me to the company members, I was given some more information about the internship. My first task was to read my colleague's Rafael Henriques, internship intermediate report, which is related to my internship. In doing so, I would be learning about his work and at the same time learning more about the topic. This served as a kind of introduction, since I would work with him, within the same subjects.

For better interpretation, this report was read twice while taking some notes. In addition, I was asked to write the objectives of the internship, as a guarantee that I had understood them. At the same time, I was also trying to get in touch with the tools that I would use in development, in this case Angular, React and Django.

In this period, in order to receive feedback, I started to have remote weekly meetings with the company via Zoom. These meetings lasted at most 1 hour.

Figure 6.2 illustrates the planning for the first semester. We have two colors to represent the time window of each task. In **red** we have the **expected** time. In **green** we have the **effective (actual)** time that each one took.

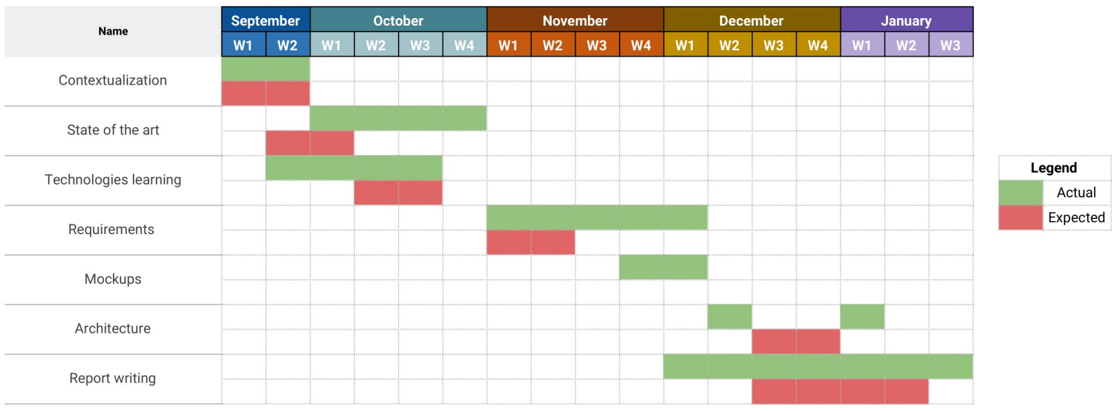


Figure 6.2: First Semester Planning - Gantt Chart

At the beginning of October, I was in a position to start gathering material for my state of the art. For that, I started by researching all the tools that could make sense to analyze. The state of the art took roughly a month to gather information. It can be easily explained taking into account the large number of tools found. Nevertheless, I continued to try to learn some of the tools that were going to be used in development.

In early November, we started to identify high-level requirements. In this part, we also paid attention to the restrictions imposed. During the requirements survey, some mockups were also designed.

We arrived in December, close to the holidays, and it was time to start designing a high level architecture of our tool to be presented in the present report.

The rest of the time was reserved for writing the intermediate report.

6.2.2 Second Semester

The majority of the second semester was dedicated to the development process. As such, we have a total of 7 Sprints.

Figure 6.3 illustrates the planning for the second semester. As we can see, there were some differences between the **actual** vs **expected** timeline.

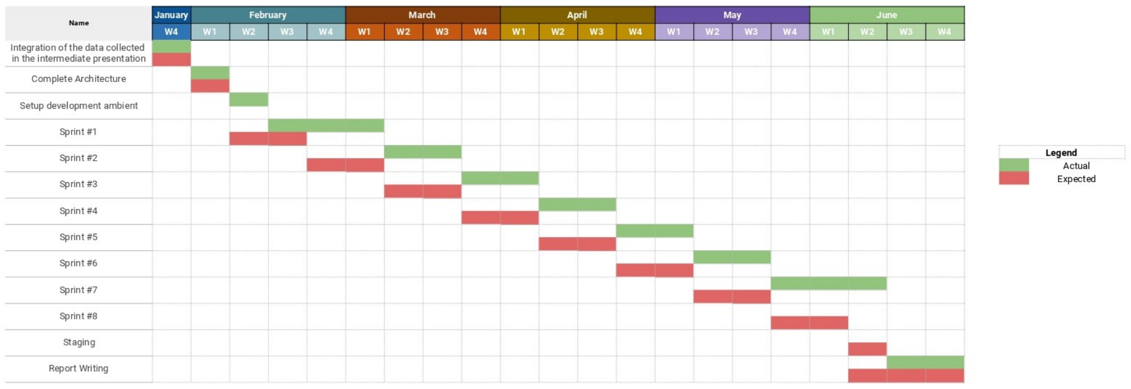


Figure 6.3: Second Semester Planning - Gantt Chart

We started the semester by discussing the information collected in the intermediate presentation. For that reason, a meeting was held with the advisor in the company and the advisor in the department.

The next step involved completing the architecture. This process took a short time as we came to the conclusion that there was not much to add. However, in this process, some new tools emerged that had not been discussed in the first semester. As such, it was necessary to configure these tools and prepare the development environment. This step was not taken into account in the first planning, and as such there was an advance of approximately one week here.

That said, the development process started. Most sprints lasted 2 weeks, with 2 exceptions, that lasted 3 weeks (Sprint#1 and Sprint#7). As a result, we had one less sprint. However, this is perfectly fine since we are using a flexible methodology, Agile. Throughout each sprint, in addition to the implementation, some tests were also performed.

Although a staging phase was planned, it ended up not happening.

Lastly, the final two weeks of June were set aside for writing the final report.

The following table, Table 6.1, describes briefly what was done in each one of the sprints.

Sprint	**Description**	**Duration**
1	Design the data model Creation of API endpoints to insert entities (vehicles, drivers and tasks) into the system	3 weeks
2	Frontend design Import .csv files into the system (via interface)	2 weeks
3	Integration with the Solver API	2 weeks
4	Data presentation in the frontend	2 weeks
5	Save plan feature Map presentation	2 weeks
6	Internationalization Gantt chart	2 weeks
7	Route manipulation	3 weeks

Table 6.1: Planning for each sprint

6.3 Risk Analysis

Throughout the development process of a Software Engineering project, one of the responsibilities of the Project Manager (PM) is to identify risks. A risk is a potential problem that may affect, directly or indirectly, the outcome of a software project [20]. As such, it can significantly influence its success. Subsequently, it is important to identify them early on, and also find a way to mitigate them. In order to identify these risks, at the beginning of each sprint, a short discussion was held with the advisor. In Table 6.2 we can observe all the risks identified in this project and the respective mitigation strategies.

Description	**Consequences**	**Mitigation**	**Probability**	**Impact**	**Sprint**
Focusing the development only in the backend and leave the frontend part once this is completed. Bearing in mind that two frameworks (Django and React) are being used without previous experience.	This could lead to a poor and/or incomplete frontend component.	Start from the beginning working on both parts in order to obtain experience	Medium	Medium	2
Small details are always coming up that cause the data model to be constantly changing.	We need to be constantly erasing and creating the DB. These changes can affect significantly the schedule of the project.	Assess from the beginning if the data model is capable of modeling diverse real problems in order to avoid continuous changes	High	Medium	3

Table 6.2: Identified risks

Chapter 7

Conclusion

After about 10 months, this curricular internship came to an end. Throughout all this process, there were no major problems or delays. In addition, it was successful since all the previously defined objectives were achieved. As such, we can conclude that the project went as expected.

In the first semester we were able to define the main functional aspects of our solution. Through the realization of the state of the art we managed to find that there is no ideal tool to solve our problem. In addition, we gathered a good amount of characteristics and features that should be considered for our solution. High-level functional requirements have also been defined. From these, a high-level architecture was designed.

In the second semester we started the development process. The development methodology adopted was Agile. There was a total of 7 sprints. Each sprint lasted about 2 weeks, with exception for two sprints.

Also during this semester, it was necessary to carry out a validation. This was important as it allows us to make sure we built the right product. In order to achieve this, three types of testing methodologies were followed: unit testing, integration testing and functional testing.

Lastly, the in the final step of the validation process, the usability of the system was evaluated. This had to be done since the usability levels are a very important aspect of this project.

Regarding the future work issue. Due to lack of time, some of the functional requirements were not implemented (e.g., authentication). The next logical step would be to implement them. Later, it would be interesting to integrate this solution with Sentilant's current solution, DrivianTasks. Finally, in this type of tool there is always some new feature to develop. Whether by need or by customer request.

In conclusion, we delivered a tool that meets the requirements for which it was built. As mentioned earlier, all the main objectives were fulfilled. As such, this tool can be used by the end user in order to achieve their business goals and help reduce costs.

References

[1] Rafael Filipe Carreira Henriques. Work plan. In *Tailored Field-Service Optimization*, 2020.

[2] Emergn. How to write a great user story for product planning and development, 2017. https://www.emergn.com/blog/write-great-user-stories/, (Accessed on 2020-01-06).

[3] University of New Brunswick. Understanding quality attributes, 2011. https://www.cs.unb.ca/~wdu/cs6075w10/sa2.htm, (Accessed on 2020-01-11).

[4] Aaron Emmert, Bits in Glass. Agile methodologies enhance appian delivery – part 1, 2019. https://bitsinglass.com/agile-methodologies-enhance-appian-delivery-part-1/, (Accessed on 2020-01-08).

[5] João Reis. Angular vs react: A comparison of both frameworks, 2020. https://www.imaginarycloud.com/blog/angular-vs-react/, (Accessed on 2020-01-08).

[6] Margaret Rouse. What is field service management?, 2016. https://searchcustomerexperience.techtarget.com/definition/field-service-management-FSM, (Accessed on 2020-01-10).

[7] Verizon Connect. What is route optimization?, 2020. https://www.verizonconnect.com/ca/glossary/what-is-route-optimization/, (Accessed on 2020-01-11).

[8] Brandon Wozniewicz. The difference between a framework and a library, 2019. https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/, (Accessed on 2020-01-10).

[9] QRA. Functional vs non-functional requirements: The definitive guide, 2020. https://qracorp.com/functional-vs-non-functional-requirements/, (Accessed on 2020-01-08).

[10] Wikipedia. User story, 2017. https://en.wikipedia.org/wiki/User_story, (Accessed on 2020-01-06).

[11] Nikolay Ashanin. Quality attributes in software architecture, 2018. https://medium.com/@nvashanin/quality-attributes-in-software-architecture-3844ea482732, (Accessed on 2020-01-11).

[12] A. A. A. Saeed and S. Lee. Non-functional requirements trade-off in self-adaptive systems. In *2018 4th International Workshop on Requirements Engineering for Self-Adaptive, Collaborative, and Cyber Physical Systems (RESACS)*, pages 9–15, 2018.

[13] Michael Keeling. Dealing with constraints in software architecture design, 2014. `https://www.neverletdown.net/2014/10/dealing-with-constraints-in-software-architecture.html`, (Accessed on 2020-01-05).

[14] Darius Sas and Paris Avgeriou. Quality attribute trade-offs in the embedded systems industry: an exploratory case study. *Software Quality Journal*, 28(2):505–534, Jun 2020.

[15] Simon Brown. The c4 model for visualising software architecture, 2011. `https://c4model.com/`, (Accessed on 2020-01-09).

[16] Hoang Pham. Verification and validation. In *Software Reliability*, 2000.

[17] Django Tenants Documentation. Tests, 2020. `https://django-tenants.readthedocs.io/en/latest/test.html`, (Accessed on 2021-06-21).

[18] Jeff Offutt Paul Ammann. Coverage criteria. In *Introduction to Software Testing*, 2013.

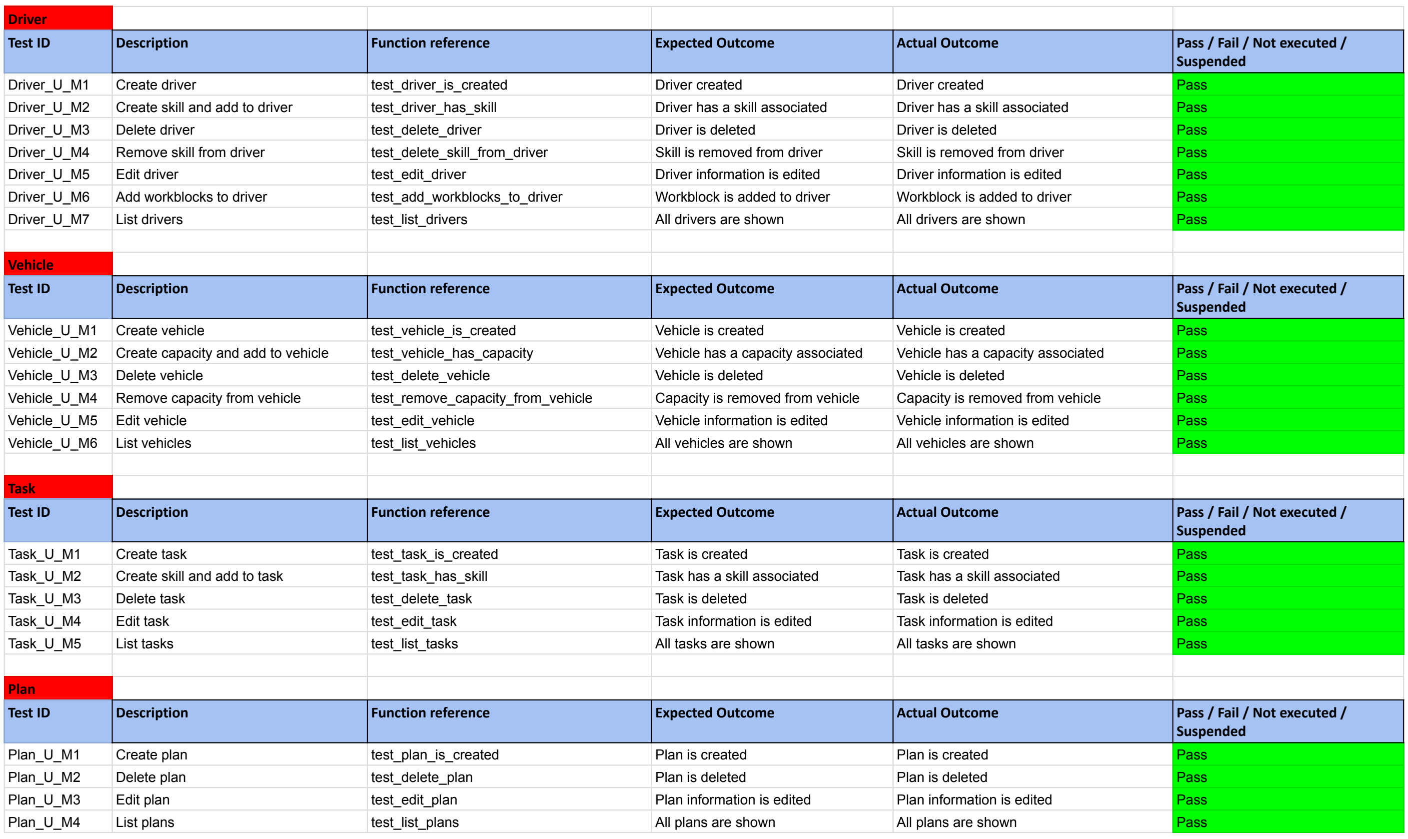
[19] Wrike. What is agile methodology in project management?, 2018. `https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/`, (Accessed on 2020-01-09).

[20] Cast Software. Risk management in software development and software engineering projects, 2021. `https://www.castsoftware.com/research-labs/risk-management-in-software-development-and-software-engineering-projects`, (Accessed on 2021-06-15).

Appendices

This page is intentionally left blank.

Appendix A



Driver

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Driver_U_M1	Create driver	test_driver_is_created	Driver created	Driver created	Pass
Driver_U_M2	Create skill and add to driver	test_driver_has_skill	Driver has a skill associated	Driver has a skill associated	Pass
Driver_U_M3	Delete driver	test_delete_driver	Driver is deleted	Driver is deleted	Pass
Driver_U_M4	Remove skill from driver	test_delete_skill_from_driver	Skill is removed from driver	Skill is removed from driver	Pass
Driver_U_M5	Edit driver	test_edit_driver	Driver information is edited	Driver information is edited	Pass
Driver_U_M6	Add workblocks to driver	test_add_workblocks_to_driver	Workblock is added to driver	Workblock is added to driver	Pass
Driver_U_M7	List drivers	test_list_drivers	All drivers are shown	All drivers are shown	Pass

Vehicle

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Vehicle_U_M1	Create vehicle	test_vehicle_is_created	Vehicle is created	Vehicle is created	Pass
Vehicle_U_M2	Create capacity and add to vehicle	test_vehicle_has_capacity	Vehicle has a capacity associated	Vehicle has a capacity associated	Pass
Vehicle_U_M3	Delete vehicle	test_delete_vehicle	Vehicle is deleted	Vehicle is deleted	Pass
Vehicle_U_M4	Remove capacity from vehicle	test_remove_capacity_from_vehicle	Capacity is removed from vehicle	Capacity is removed from vehicle	Pass
Vehicle_U_M5	Edit vehicle	test_edit_vehicle	Vehicle information is edited	Vehicle information is edited	Pass
Vehicle_U_M6	List vehicles	test_list_vehicles	All vehicles are shown	All vehicles are shown	Pass

Task

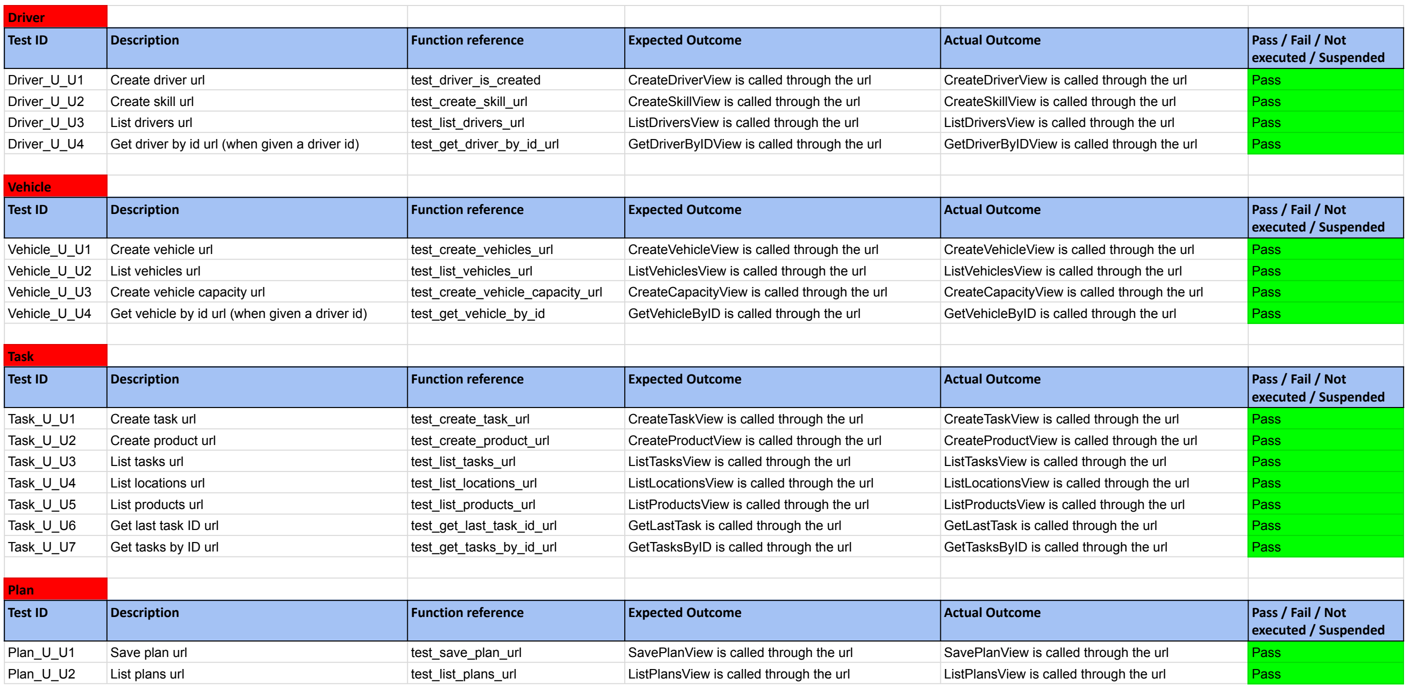
Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Task_U_M1	Create task	test_task_is_created	Task is created	Task is created	Pass
Task_U_M2	Create skill and add to task	test_task_has_skill	Task has a skill associated	Task has a skill associated	Pass
Task_U_M3	Delete task	test_delete_task	Task is deleted	Task is deleted	Pass
Task_U_M4	Edit task	test_edit_task	Task information is edited	Task information is edited	Pass
Task_U_M5	List tasks	test_list_tasks	All tasks are shown	All tasks are shown	Pass

Plan

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Plan_U_M1	Create plan	test_plan_is_created	Plan is created	Plan is created	Pass
Plan_U_M2	Delete plan	test_delete_plan	Plan is deleted	Plan is deleted	Pass
Plan_U_M3	Edit plan	test_edit_plan	Plan information is edited	Plan information is edited	Pass
Plan_U_M4	List plans	test_list_plans	All plans are shown	All plans are shown	Pass

This page is intentionally left blank.

Appendix B



Driver

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Driver_U_U1	Create driver url	test_driver_is_created	CreateDriverView is called through the url	CreateDriverView is called through the url	Pass
Driver_U_U2	Create skill url	test_create_skill_url	CreateSkillView is called through the url	CreateSkillView is called through the url	Pass
Driver_U_U3	List drivers url	test_list_drivers_url	ListDriversView is called through the url	ListDriversView is called through the url	Pass
Driver_U_U4	Get driver by id url (when given a driver id)	test_get_driver_by_id_url	GetDriverByIDView is called through the url	GetDriverByIDView is called through the url	Pass

Vehicle

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Vehicle_U_U1	Create vehicle url	test_create_vehicles_url	CreateVehicleView is called through the url	CreateVehicleView is called through the url	Pass
Vehicle_U_U2	List vehicles url	test_list_vehicles_url	ListVehiclesView is called through the url	ListVehiclesView is called through the url	Pass
Vehicle_U_U3	Create vehicle capacity url	test_create_vehicle_capacity_url	CreateCapacityView is called through the url	CreateCapacityView is called through the url	Pass
Vehicle_U_U4	Get vehicle by id url (when given a driver id)	test_get_vehicle_by_id	GetVehicleByID is called through the url	GetVehicleByID is called through the url	Pass

Task

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Task_U_U1	Create task url	test_create_task_url	CreateTaskView is called through the url	CreateTaskView is called through the url	Pass
Task_U_U2	Create product url	test_create_product_url	CreateProductView is called through the url	CreateProductView is called through the url	Pass
Task_U_U3	List tasks url	test_list_tasks_url	ListTasksView is called through the url	ListTasksView is called through the url	Pass
Task_U_U4	List locations url	test_list_locations_url	ListLocationsView is called through the url	ListLocationsView is called through the url	Pass
Task_U_U5	List products url	test_list_products_url	ListProductsView is called through the url	ListProductsView is called through the url	Pass
Task_U_U6	Get last task ID url	test_get_last_task_id_url	GetLastTask is called through the url	GetLastTask is called through the url	Pass
Task_U_U7	Get tasks by ID url	test_get_tasks_by_id_url	GetTasksByID is called through the url	GetTasksByID is called through the url	Pass

Plan

Test ID	Description	Function reference	Expected Outcome	Actual Outcome	Pass / Fail / Not executed / Suspended
Plan_U_U1	Save plan url	test_save_plan_url	SavePlanView is called through the url	SavePlanView is called through the url	Pass
Plan_U_U2	List plans url	test_list_plans_url	ListPlansView is called through the url	ListPlansView is called through the url	Pass

This page is intentionally left blank.

Appendix C

Driver					
Test ID	**Description**	**Function reference**	**Expected Outcome**	**Actual Outcome**	**Pass / Fail / Not executed / Suspended**
Driver_I_V1	Sends a POST request to create a driver	test_create_driver_POST	Web response that confirms the driver is created	Web response that confirms the driver is created	Pass
Driver_I_V2	Sends a POST request to create a skill	test_create_skill_POST	Web response that confirms the skill is created	Web response that confirms the skill is created	Pass
Driver_I_V3	Sends a GET request to list all drivers	test_list_drivers_GET	Web response that lists all drivers	Web response that lists all drivers	Pass
Driver_I_V4	Sends a GET request to obtain driver with a specific id	test_get_driver_by_ID_GET	Web response that lists the driver with that ID	Web response that lists the driver with that ID	Pass

Vehicle					
Test ID	**Description**	**Function reference**	**Expected Outcome**	**Actual Outcome**	**Pass / Fail / Not executed / Suspended**
Vehicle_I_V1	Sends a POST request to create a vehicle	test_create_vehicle_POST	Web response that confirms the vehicle is created	Web response that confirms the vehicle is created	Pass
Vehicle_I_V2	Sends a GET request to obtain all vehicles	test_list_vehicles_GET	Web response that lists all vehicles	Web response that lists all vehicles	Pass
Vehicle_I_V3	Sends a POST request to create a vehicle capacity	test_create_capacity_POST	Web response that confirms the vehicle capacity is created	Web response that confirms the vehicle capacity is create	Pass
Vehicle_I_V4	Sends a GET request to obtain vehicle with a specific id	test_get_vehicle_by_ID_GET	Web response that lists the vehicle with that ID	Web response that lists the vehicle with that ID	Pass

Task					
Test ID	**Description**	**Function reference**	**Expected Outcome**	**Actual Outcome**	**Pass / Fail / Not executed / Suspended**
Task_I_V1	Sends a POST request to create a task	test_create_task_POST	Web response that confirms the task is created	Web response that confirms the task is created	Pass
Task_I_V2	Sends a POST request to create a product	test_create_product_POST	Web response that confirms the product is created	Web response that confirms the product is created	Pass
Task_I_V3	Sends a GET request to list all tasks	test_list_tasks_GET	Web response that list all tasks	Web response that list all tasks	Pass
Task_I_V4	Sends a GET request to list all locations	test_list_locations_GET	Web response that list all locations	Web response that list all locations	Pass
Task_I_V5	Sends a GET request to list all products	test_list_products_GET	Web response that list all products	Web response that list all products	Pass
Task_I_V6	Sends a GET request to obtain last inserted task ID	test_get_last_task_ID_GET	Web response with the ID of the last inserted task	Web response with the ID of the last inserted task	Pass
Task_I_V7	Sends a GET request to obtain tasks with a specific ID	test_get_task_by_ID_GET	Web response with the tasks requested by ID	Web response with the tasks requested by ID	Pass

Plan					
Test ID	**Description**	**Function reference**	**Expected Outcome**	**Actual Outcome**	**Pass / Fail / Not executed / Suspended**
Plan_I_V1	Sends a POST request to save a plan	test_save_plan_POST	Web response that confirms the plan is saved	Web response that confirms the plan is saved	Pass
Plan_I_V2	Sends a GET request to list all plans	test_list_plans_GET	Web response that list all the plans	Web response that list all the plans	Pass

This page is intentionally left blank.

Appendix D

Test ID	User Story	Description	Pre conditions	Steps to reproduce	Expected Result	Actual Result	Result
F_TC_1	2. Define vehicles 12. Import resources and tasks	The user has a list of vehicles, with the appropriate characteristics, that are imported into the system	None	1. List of vehicles and characteristics are in a .csv file (in a specific format) 2. Navigate to "Vehicles" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button	A message appears saying the vehicles were successfully inserted into the system	A message appears saying the vehicles were successfully inserted into the system	Pass
F_TC_3	3. Define drivers 12. Import resources and tasks	The user has a list of drivers, with the appropriate characteristics, that are imported into the system	None	1. List of drivers and characteristics are in a .csv file (in a specific format) 2. Navigate to "Drivers" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button	A message appears saying the drivers were successfully inserted into the system	A message appears saying the drivers were successfully inserted into the system	Pass
F_TC_4	4. Define tasks 12. Import resources and tasks	The user has a list of tasks, with the appropriate characteristics, that are imported into the system	None	1. List of tasks and characteristics are in a .csv file (in a specific format) 2. Navigate to "Tasks" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button	A message appears saying the tasks were successfully inserted into the system	A message appears saying the tasks were successfully inserted into the system	Pass
F_TC_5	Not defined	The user has a list of vehicle capacities, that are imported into the system	None	1. List of vehicle capacities are in a .csv file (in a specific format) 2. Navigate to "Capacities" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button	A message appears saying the vehicles capacities were successfully inserted into the system	A message appears saying the vehicles capacities were successfully inserted into the system	Pass
F_TC_6	Not defined	The user has a list of products, that are imported into the system	None	1. List of products are in a .csv file (in a specific format) 2. Navigate to "Products" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button	A message appears saying the products were successfully inserted into the system	A message appears saying the products were successfully inserted into the system	Pass
F_TC_7	Not defined	The user has a list of skills, that are imported into the system	None	1. List of skills are in a .csv file (in a specific format) 2. Navigate to "Skills" page 3. Select "Import file" and select the .csv file 4. Select "Submit" button	A message appears saying the skills were successfully inserted into the system	A message appears saying the skills were successfully inserted into the system	Pass
F_TC_8	5. Select resources	All the available resources and tasks loaded into the system are ready to be used in the optimization process	We need to have tasks, products, vehicles (with vehicles capacities) and drivers previously imported to the system	1. Navigate to "Overview" page 2. Select "Optimize" button	After a few seconds we receive the optimized plan (routes, unasigned tasks and metrics) from the external solver.	After a few seconds we receive the optimized plan (routes, unasigned tasks and metrics) from the external solver.	Pass
F_TC_9	6. Interpret gantt	Once the plan is optimized we can have a general view in a Gantt chart	We need to have a previously optimized plan	1. Navigate to "Overview" page 2. Select "Gantt" tab	We can observe the results of the optimized plan in a Gantt diagram format	We can observe the results of the optimized plan in a Gantt diagram format	Pass
F_TC_10	7. Interpret KPI	Once the plan is optimized we can have see some important KPIs that allows us to have an opinion in the quality of the optimized plan (e.g., total distance and percentage of assigned tasks)	We need to have a previously optimized plan	1. Navigate to "Overview" page 2. Observe the cards in the top middle of the screen	We can observe some important KPIs about the plan	We can observe some important KPIs about the plan	Pass
F_TC_11.1	8. Interpret routes	Once the plan is optimized we can see all the routes listed with information such as cargo location, starting time, driver, among others	We need to have a previously optimized plan	1. Navigate to "Overview" page 2. Observe the routes list	We can observe the list of routes of the optimized plan	We can observe the list of routes of the optimized plan	Pass
F_TC_11.2	8. Interpret routes	We can expand a specific route and observe the list of tasks and respective details	We need to have a previously optimized plan	1. Navigate to "Overview" page 2. Click the "Arrow" icon in a route to expand it	We can observe the list of tasks in each route individually	We can observe the list of tasks in each route individually	Pass
F_TC_11.3	8. Interpret routes	We can expand a specific route and observe the tasks on map	We need to have a previously optimized plan	1. Navigate to "Overview" page 2. Click the "Arrow" icon in a route to expand it 3. Click the "Position" icon to open a map dialog	We can observe the starting position and the tasks on the map	We can observe the starting position and the tasks on the map	Pass
F_TC_12	9. Save plan	We can save the optimized plan	We need to have a previously optimized plan	1. Select a name for the plan in the Header component of the application 2. Click the "Floppy" icon in the Header component to save the plan	A message appears informing the user that the plan was successfully saved	A message appears informing the user that the plan was successfully saved	Pass
F_TC_13	10. Erase plan	We can delete the optimized plan	We need to have a previously optimized plan	1. Click the "Garbage" icon in the Header component to delete the plan	A message appears informing the user that the plan was successfully erased	A message appears informing the user that the plan was successfully erased	Pass
F_TC_14	13. Compare plans	We can compare previously saved plans in general	We need to have more than one plan previously saved	1. Navigate to "Plans" page	We can observe the list of plans previously saved with a few KPIs that allows the user to compare them generally	We can observe the list of plans previously saved with a few KPIs that allows the user to compare them generally	Pass
F_TC_15	15. Consult saved plans	We can open previously saved plans	We need to have a previously optimized plan saved	1. Navigate to "Plans" page 2. Click on the plan name to open it	We are redirected to the "Overview" page where we can see the overview of the plan we tried to open	We are redirected to the "Overview" page where we can see the overview of the plan we tried to open	Pass
F_TC_16	17. Internationalization	We can change the application language (between portuguese and english)	None	1. Click the "Settings" icon in the Header component 2. Select the flag of the language	The page is refreshed and loads the selected language	The page is refreshed and loads the selected language	Pass

This page is intentionally left blank.

Appendix E

ID	Name	Description	Result
Navigation and Feedback			
1	Feedback is always provided on user actions	The user should always receive immediate feedback on their actions, so that they know that the system has received their command and is processing it. This feedback can be provided in several ways: when the user hovers over a link (a: hover); when the user clicks on a link (a:active); when the user navigates with the keyboard (a: focus); when the user is filling in form fields (input [type=text]:focus, textarea:focus); when the user clicks on a link that opens on the same page (#id: target); and/or when the action triggered by the user takes a long time to be processed (progress bar, "loading" icon or other information that gives the feeling that the action is taking place and that the user must wait for its completion).	🟩
2	Feedback on user location is provided.	We must always inform the user's location, preferably through a clear visual feedback in the menu, indicating the section in which the user is. This information can also be complemented with "breadcrumb" type navigation in case there are more than 2 hierarchical levels of navigation.	🟩
3	The titles of links and menus are clear and understandable	The texts in the menus must be clearly visible so that the user can understand from the outset what content they will see if they click on a particular link. Whenever possible, the use of abbreviations in the main menus should be avoided.	🟩
4	Clickable items look clickable and different from the rest of the content	Clickable items must be clearly distinguishable from other items. Links in the middle of the text must be immediately identifiable as such and must not be confused with the rest of the text. Once the appearance of the links is defined, that aspect must be kept identical on all other pages of the site.	🟩

5	Non-clickable items don't look like links or buttons	The use of underlined texts when they do not contain links should be avoided. Additionally, in the middle of the text, different colors should not be used in words, sentences or paragraphs. A consistency must be maintained in which the text always maintains the same color across all pages, as the use of text in a different color can also be associated with the existence of a link to another page. Even more serious is to use the same color chosen for links in text that is not a link (if you do not use underlined links, it is advisable to identify them with a different color/appearance from the rest of the text).	🟩
6	Link text makes sense when read out of context	Users should be able to look at links and automatically understand something about their destination even before they click. The use of terms such as "click here" can be quite counter-productive and when read out of context does not provide additional information.	🟩
7	There are no broken links	Links should be tested so that there are no broken links (to non-existent pages, or to incorrect pages)	🟩
8	There is enough padding on the paging links	In paging links, an extra spacing (padding) must be created around each link. This makes navigation easier as links placed only on one character become too narrow and difficult to click. By creating this extra spacing you increase the clickable area on the links and at the same time give better visual feedback to the user.	Not applicable
Layout			
9	Pages are consistent across the entire website.	Important information and clickable items (major and minor navigation blocks) should always be placed in the same places throughout the entire website.	🟩

10	There is a specific style sheet for printing	The website must be prepared so that the contents can be read online or on paper. According to some studies, the reason users choose to print an article or read it online is because of its size. The larger the size of the article, the more likely users are to choose to print it rather than read it online. Thus, the option to print the contents of a web page must always be provided. This option must be done through a specific style sheet (CSS) for printing and not through a link that opens the same article in a different version (optimized for printing).	Not applicable
11	No fixed heights are defined for the elements	It is important that the graphical appearance does not limit the contents and that it grows as the content increases or the text size increases. Thus, the graphic elements should not have fixed heights, as the contents that will be placed inside can grow more than the height defined at the beginning, thus breaking the layout.	🟥
12	The icons used are consistent with the actions they perform.	Icons must always represent the same actions and should not be reused for different actions throughout the website.	🟩
13	Decorative images are not used in the middle of the HTML	The use of decorative images in the middle of the HTML (eg rounded corners, spacing images, etc) should be avoided. All elements related to the presentation/decoration must be included via CSS. On the other hand, images that convey important information or that are part of the content must be included directly on the page via the IMG tag and not as a background image of a DIV.	Not applicable
Readability			
14	Line spacing has been increased to improve readability	A minimum line spacing of 1.4 points must be maintained in the text blocks of the contents (texts and articles). Greater line spacing helps make text easier to read and reduce the feeling of tired eyes after reading long text on the screen. This spacing can be easily added via a CSS line: line-height:1.4;	🟥

15	Critical information (which requires the user's attention) is highlighted enough on the page	Critical information can be of various types, but it is usually related to information about changed content on the page or validation of data entered by the user in forms. It could be warnings, information, or errors. The highlight effect should not be overused as it is more efficient when used a few times on the page. On a website where the user is constantly confronted with highlight text, the effect of drawing the user's attention is easily lost.	🟩
16	Only bold text blocks are used to highlight relevant information on the page.	Excessive use of bold text should be avoided. Bold text should only be used to highlight certain words or phrases. Excessive use makes the emphasis we want to give to the highlighted elements to lose its effect. If everything is highlighted, nothing stands out.	🟩
17	There is sufficient contrast between the text color and the background color	The color used in the texts must make a sufficient contrast with the background color to ensure good readability. A bad contrast between the two colors can make the texts unreadable for a good part of the population, even for people with "normal" vision.	🟩
Forms and Messages			
18	The mandatory items are distinguishable from the others	Users must be able to clearly distinguish the fields in which completion is mandatory from the other fields. These days, most websites use an asterisk in front of the field name to identify them as required; other websites use the word "required" instead of the asterisk. Both solutions are valid, but the use of an asterisk requires a caption at the top of the form to indicate that fields marked with * are mandatory.	🟩
19	Error messages are next to the elements that contain the error.	Error messages must be indicated next to the fields that contain the error and not just at the top or end of the form. This allows for better contextualization of errors and helps to understand where users need to correct them.	🟩

20	The main actions are clearly distinguishable from the secondary actions on the forms.	There must be a visual differentiation between the main actions and the secondary actions in order to avoid potential errors on the part of the user. This differentiation also helps to clearly see which action confirms and which action cancels the form.	🟩
21	The forms are working and send to the correct recipients	The forms must all be tested and the data reception accounts must be verified to confirm that the data was received successfully.	🟩
22	Search is working and error messages are adequate	The search form must return results for the searches carried out and when there are no results, a message must be displayed indicating that the search did not return any results.	🟩
Help			
23	There is contextual help for more complex interactions	Since most people do not usually read the instructions/help before starting to use a system, the most useful way to help users navigate and overcome situations that can be more complicated is to provide contextual help at the right time. and in the right place, where we know users will need it.	🟥
24	Error messages help resolve the issue	Error messages should be clear and should help the user to correct the error. A message such as: "An error occurred while filling out the form" does not help at all to know why the error occurred or how to correct it.	🟥
Social networks			
25	Added "Open Graph" tags	Open Graph tags must be added at the beginning of all content pages that allow you to indicate a series of parameters about the content of the page and that will be used in shares on social networks.	Not applicable
Responsive			

26	The site has been tested on devices of various sizes	When developing responsive websites, they should be tested on multiple devices with different screen sizes to confirm that the layouts are correctly applied.	Not applicable
Performance			
27	All static files have been minified and compressed	All static files (eg CSS, JS and HTML) must be minified and compressed. This saves a lot of KB every time pages are loaded.	🟩
28	The contents were optimized for mobile	It is important to ensure that the content that a responsive website loads is optimized to be consumed under these circumstances. This includes serving lighter images, replacing Flash elements (if any) with HTML5 equivalents (eg video players) and reducing decorative elements to as few items and files as possible.	Not applicable