FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Framework for collecting and processing georeferencing data

Diogo Sampaio Duarte Cardoso

WORKING VERSION

Integrated Masters in Electrical and Computers Engineering

Supervisor: Dr. Prof. Luís Paulo Reis

Second Supervisor: Artur Dias

March 3, 2020

Abstract

Unprecedented amounts of data are constantly being generated by all kind of sources, from the connected digital ecosystem to the internet of things. The ability to take advantage of this amount of data plays an important role in achieving successful business strategies. In order to do this, ingesting large amounts of data into a central repository and processing it is the first step, before any analytics, predictive modeling or reporting can be done.

A use case for real-time data streams and processing is geolocation data. This facilitates the tracking of a set of devices, from smartphones to GPS modules, which can be used to create an overview of all the connected assets, in order to provide an easier and improved management system. This can be combined with analytics and prediction models to provide extra data which can backup and create new opportunities and improvements to the current workflows and processes.

In this dissertation, a framework for real-time data ingestion and consumption is designed and implemented. The data ingestion infrastructure is reponsible for ingesting geolocation data which is generated by GPS modules installed in a fleet of vehicles and smartphones, cleaning and transforming such data to match the desired output data schema, and storing it in a database for real-time and on-demand use cases.

In order to present the geolocation data in real-time, a map-centered web application was developed. This web application uses the real-time location of a fleet of vehicles to display markers on a map, together with business data, which allows a operations team to extract more insights and improve the scheduling and routing processes.

To provide more control over costs and aid internal auditing processes that are connected with vehicle metrics such as distances and durations, an automated report was created using a business inteligence software. This report is generated daily and uses business data to calculate estimated trips durations and distances, which are then compared with the actual trips durations and distances that are stored through the ingestion framework.

In summary, the geolocation data ingestion framework is the foundation for multiple applications and use-cases. Some of these uses cases are described and implemented in this dissertation, though there are many others. The goal is to provide a solid and scalable solution for an infrastructure that is future-proof.

Resumo

Hoje em dia, verifica-se uma quantidades de dados sem precedentes a ser constantemente gerada por todos os tipos de fontes, desde o ecossistema digital à internet das coisas. A capacidade de tirar proveito dessa quantidade de dados desempenha um papel importante na obtenção de estratégias de negócios bem-sucedidas. Para isso, a ingestão de grandes quantidades de dados num repositório central e o seu processamento são a primeira etapa, antes de qualquer análise, modelagem preditiva ou relatório possa ser produzido.

Um caso de uso para fluxos de dados em tempo real e processamento são dados de geolocalização. Isto permite, por exemplo, o rastreamento de um conjunto de dispositivos, desde smartphones a módulos de GPS, que podem ser usados, posteriormente, para criar uma visão geral de todos os ativos conectados, a fim de fornecer um sistema de gestão melhorado e mais avançado. Este tipo de sistemas podem ser combinados com modelos de análise e previsão para fornecer mais dados que podem ajudar a justificar ou criar novas oportunidades, e aprimorar os processos existentes.

Nesta dissertação, é projetada e implementada uma *framework* para ingestão e consumo de dados em tempo real. A infraestrutura de ingestão de dados é responsável pela ingestão de dados de geolocalização gerados por módulos GPS instalados numa frota de veículos e smartphones, pela transformação desses dados para fazer corresponder a estrutura de dados ao pretendido, e pelo armazenamento em base de dados, para uso em tempo-real ou *on-demand*.

Para apresentar os dados de geolocalização em tempo real, foi desenvolvida uma aplicação web. A aplicação usa a localização em tempo real de uma frota de veículos para pintar marcadores num mapa, juntamente com dados de negócio, o que permite que uma equipa de operações extraia mais informações e melhore os processos de agendamento e cálculo de rotas.

Foi também criado um relatório automatizado usando um software de *business inteligence* que fornece métricas relacionadas com os veículos, como as distancias percorridas e durações das viagens. Isto resulta num maior controlo sobre os custos do negócio e assiste em processos de auditoria interna. Este relatório é gerado diariamente e usa dados do negócio para calcular as durações e distâncias estimadas das viagens, que são comparadas com as durações e distâncias reais das viagens.

Em resumo, a *framework* de ingestão de dados de geolocalização é a base para várias aplicações e pode ser usada para diversos fins. Alguns desses casos de uso são descritos e implementados nesta dissertação, embora existam muitos outros. O objetivo é fornecer uma solução sólida e escalável para uma infraestrutura preparada para o futuro.

Acknowledgments

To my parents, my brother and my grandparents for the support throughout the journey of becoming an engineer, and life in general.

To Wegho, the company where I worked the last 6 months, for providing the means to do my dissertation and an healthy and enjoyable environment to work in. In particular, I'd like to thank my co-supervisor Artur Dias, for his input on the numerous discussions we had, and support in general.

To my supervisor, Dr. Professor Luís Paulo Reis, for taking this dissertation and providing support, feedback and ideas which allowed me to improve a lot.

To all my friends, for accepting nothing less than excellence from me.

To each and every one of you - Thank you.

Diogo Cardoso

"Luck is where opportunity meets preparation."

Seneca

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
BI	Business Inteligence
GPS	Global Positioning System
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
UI	User Interface
UML	Unified Modeling Language
UX	User Experience

Chapter 1

Introduction

1.1 Context

The 21st century is being marked by the disruption of a large number of markets. The possibility to change how things have always been done, go outside of the traditional ways and offer services that the costumers have always dreamed of, is now becoming possible due to the available technologies. New product development, more targeted marketing and better decision-making are happening thanks to Big Data and cross-referencing information across multiple data sources.

Unprecedented amounts of data are constantly being generated by all kind of sources, from the connected digital ecosystem to the internet of things. Since the 1980s, the capacity to store information per-capita has nearly doubled every 40 months. In 2012, there were 2.5 exabytes of data being generated every day. By 2025, this number is expected to be around 163 zettabytes.[7] To make sense of this amount of data is not an easy task.

The ability to take advantage of this amount of data plays an important role in achieving successful business strategies. In order to do this, ingesting large amounts of data into a central repository and processing it is the first step, before any analytics, predictive modeling or reporting can be done. A large part of data-driven applications rely on real-time data processing, mostly due to the use of many connected resources such as internet of things, mobile devices, social networks and sensors, which adds new challenges. Real-time processing requires continuous and sequential transactions of limitless streams of input data, which is why it is critical that the Big Data streams are processed with very short latency.

A use case for real-time data streams and processing is geolocation data. This facilitates the tracking of a set of devices, from smartphones to GPS modules, which can be used to create an overview of all the connected assets, in order to provide an easier and improved management system. This can be combined with analytics and prediction models to provide extra data which can backup and create new opportunities and improvements to the current workflows and processes.

1.2 Motivation

Wegho provides services to homes and offices, each having a team of 2 or 3 suppliers, that travel using Wegho's vehicle fleet. Most of the vehicles are equiped with GPS modules, provided by CarTrack - a company specialized in vehicle location systems. The GPS modules transmit location data and CAN events to a CarTrack server, which is accessible through a web application and a SOAP API.

Until now, neither solution was actively being used by Wegho, due to several factors:

- The web application was developed several years ago, runs on Adobe Flash which is being deprecated in 2020 [8], is extremelly slow and has a poor user experience. Its current use is limited to alerts, in particular, for emergency scenarios such as theft.
- The lack of business data means that business insights and metrics are not easily obtained. The web application doesn't offer integration with external data sources so a real-time overview of both the vehicles and services, and relevant metrics related to efficiency and scheduling are not possible.

As Wegho's focus shifts towards improving operational processes around its fleet and service scheduling, the lack of a suitable tool was a big bottleneck.

Besides technical and logistic improvements, a fleet management system allows more control over costs, opens the possibility of adopting a benefit program for suppliers which relies on efficiency metrics, and integration with consumer products such as the mobile application.

1.3 Wegho

Wegho is a platform that provides services and connects services providers to costumers. Wegho currently offers services in different areas such as cleaning, painting, plumbing, electrical and fitness, though they are constantly adding new services. Wegho currently targets both B2C and B2B channels and its services are available in Porto and Lisboa, Portugal.

1.4 Goals

In order to gain advantage over the competition through the analysis of data that originates from their operations, Wegho is looking for a infrastructure to ingest and process data, which will feed analytical models and generate analytics and reports. This data is mostly geolocation and generated by GPS modules and smartphones.

The goal is to develop such infrastructure which is able to:

- Ingest data from multiple sources, such as GPS modules and smartphones, in real-time The GPS modules equipped in every vehicle and the smartphones used by the service providers generate geolocation data which should be ingested in real-time.

- Process data in real-time All ingested data must be transformed and processed in real-time, for different reasons. The ingested data schema may require cleaning in case of duplicates or invalid data, transformation in case the output data schema does not match the input data schema, or even to merge the geolocation data with Wegho's operational data.
- Store data in a database In order to be able to do further data analysis, all data must be saved in a database or data-warehouse.
- Design and implement an analytical model for vehicle/driver gas consumption Given the geolocation data from the vehicles and the services, an analytical model should be able to predict gas consumption per vehicle/driver.

This infrastructure will provide the foundation for multiple applications and tools - one of which to replace CarTrack's web application. This application will join the ingested geolocation data with Wegho's reference data. The main features of this web applications are:

- Real-time map with fleet, employees and services location
- Trips history
- Schedule services

Besides the web applications, there are other use cases such as:

- Real-time map with the location of the service provider, in the costumer's app In order to enhance the costumer's experience and improve transparency, the location of the service provider can be displayed in real-time in the costumer's app, providing a more precise time of arrival and enable notifications in case of unexpected delays.
- Update service location The service location can lack precision due to the use of reverse geocoding [9] in the booking process. In order to update the location for future services, the service provider's mobile application can be used to access the phone's GPS and get its exact location.

Chapter 2

State of the art

In order to make the best decisions on the design and implementation of the system proposed in this thesis, it is required to acquire knowledge of its related concepts, technologies, and previous work and research. In this chapter, the goal is to describe the highest level of general development, and the evolution of such level of development.

2.1 Big Data

2.1.1 Concept

Big Data refers to the large, diverse collection of data that grows at ever-increasing rates. While the act of ingesting and storing large amounts of information is not new, the concept of Big Data gained momentum in the early 2000s when industry analyst Doug Laney defined Big Data as the "three V's" [10]:

- Volume: refers to the amount of generated and stored data. Organizations collect huge amounts of data from a variety of sources, which range from terabytes to petabytes of data. Data size determines the potential value of the data, and whether it can be considered Big Data or not.

- Variety: refers to the different types and nature of the data. Data comes in all types of formats from a wide range of sources - from structured, numerical data in traditional databases to unstructured text documents, emails, photos, videos, etc. This variety of unstructured data adds new challenges to storage, mining and data analysis processes.

- Velocity: refers to the speed at which new data is generated and ingested. Data streams are coming in at unprecedented speeds and must be dealt with in a timely manner. How fast the data is generated and processed determines the real potential in the data.

With the growing adoption of Big Data in recent years, two concepts were added to what is now refered as the "five V's" of Big Data [11].

- Veracity: refers to the trustworthiness of the data. With huge amounts of unstructured data being generated, the quality and accuracy of such data is less controllable, though usually volume makes up for the lack of quality or accuracy.
- Value: refers to the real value provided to the business. Extracting value from Big Data is not an easy task, but it is important to measure the impact of the results and insights gathered from all the data.

2.1.2 Applications

The primary goal of Big Data applications is to help businesses in decision making by relying on data analysis which can be in the form of logs, activity reports, statistics or analytical models. More and more organizations are leveraging the benefits provided by Big Data applications such as enhancing customer experience, cost reductions, better targeted marketing or making existing processes more efficient. Examples of industry domains that use Big Data include healthcare, government and media. Below is a description of how Big Data is used in these industries and the challenges they face in the adoption of Big Data applications.

- Healthcare

  The healthcare sector has access to huge amounts of data but has been lagged in using Big Data, because of limited ability of standardize and consolidate data. Other challenges include the use of data from different readily available sensors. Some of the ways in which Big Data has contributed to healthcare include:

  - Reduction on costs of treatment since there is less chances of having to perform unnecessary diagnosis
  - Epidemic outbreaks prediction and setting up preventive measures
  - Detection of preventable diseases in early stages which prevents them from getting worse which in turn makes their treatment easier and more effective.
  - Identification of drug's side-effects

- Government

  In governments, the adoption and use of Big Data creates opportunities to reduce costs and increase productivity and innovation. Since governments act on a variety of different domains, Big Data has a wide range of applications, namely energy exploration, financial market analysis, cyber security, transportation management and homeland security. The biggest challenges is the integration and interoperability of Big Data across different government departments and affiliated organizations, since the same data sets are often used between them.

- Media and Entertainment

In the media and entertainment industry, the generation of large amounts of data is inevitable. Internet connect devices and social media platforms are widely used and continue to grow. Their goal is to serve content and messages that are in line with the consumer's mindset which is achieved by simultaneously analyzing costumer data along with behavioral data. Some of the benefits extracted from Big Data include:

- Predicting the interests of audiences
- Getting insights from customer reviews
- Measure content performance

2.2 Cloud Service Models

2.2.1 Introduction

Thanks to the cloud, companies are now able to not only distribute developments remotely, but also deploy them online. It also allowed users to access a wide variety of applications on-demand. In cloud computing, these are known as "as-a-service" and are commonly identified with an acronym ending in "aaS". Before these services were made available, the only option was to use an "on-premises" solution, which takes a lot of time and resources.

Considering that all systems involved in this project require some sort of infrastructures, it is important to understand the differences and advantages of the various deployment models.

There are three main "as-a-service" models:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

A summary of the key differences between these models and the legacy on-premise model is shown below.

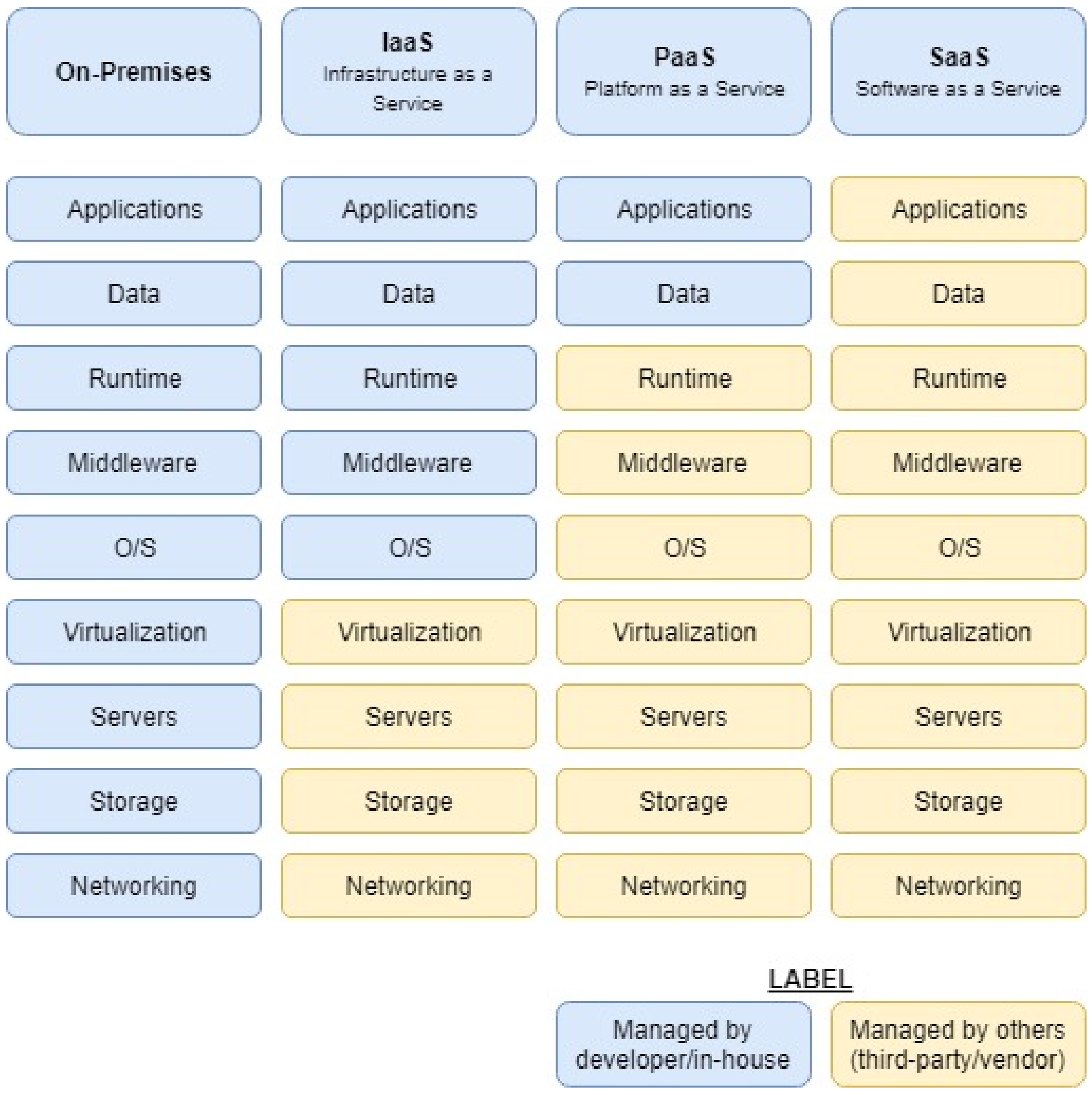

On-Premises
IaaS Infrastructure as a Service
PaaS Platform as a Service
SaaS Software as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
LABEL
Managed by developer/in-house
Managed by others (third-party/vendor)

Figure 2.1: Cloud Service Models

2.2.2 SaaS: Software as a Service

Also known as cloud application services, this model represents the most used option for businesses in the cloud market. SaaS relies on the internet to deliver applications, which are managed by a third-party. Most of these applications run directly on a web browser and usually don't require any downloads or installation on the client side.

- Delivery

  SaaS usually eliminates the need to download, install and manage applications on every computer, which would require extra IT staff. Vendors manage all of the potential technical issues, such as data, middleware, servers, and storage, allowing business to streamline their maintenance and support.

- Advantages

Due to its delivery model, SaaS allows business to reduce the time and money spent on IT staff to do the tedious tasks of download, install and upgrade applications. This allows the company to focus on more pressing matters and issues. Most SaaS providers operate a subscription model with a fixed, inclusive monthly fee, which makes it easier to budget accordingly.

- Characteristics
  - Centralized management
  - Hosted on a remote server by a third-party provider
  - Available over the internet
  - Users are not responsible for hardware or software updates
- Use cases
  - Startups and small companies and don't want to allocate time and money for server issues or software
  - Short-time projects
  - Low usage applications
  - Applications that require both web and mobile access
- Examples
  - Google Suite [12], Dropbox [13], Salesforce [14], GoToMeeting [15]

2.2.3 PaaS: Platform as a Service

PaaS or cloud platform services, provide a wide range of cloud components and are mainly consumed by applications. By using PaaS, developers rely on frameworks supplied by the provider to build and create customized applications. The servers, storage and networking can be managed in-house or by a third-party while developers can maintain control of the applications.

- Delivery

  While similar to the previous model, PaaS provides a platform for software development instead of the software itself. This platform is delivered over the internet and allows developers to focus on building the software while not having to worry about operating systems, software updates, storage or infrastructure.
- Advantages
  - Makes the development and deployment of applications a simple and cost-effective process.
  - Scalable

  - Highly available
  - Greatly reduces the amount of coding, saving a lot of time and money

- Characteristics

  - Resources can be easily scaled up or down as your business changes
  - Provides a set of services which assist with the development, testing and deployment of applications
  - Easy to run without extensive system administration knowledge

- Examples

  - AWS Elastic Beanstalk [16], Heroku [17], Google App Engine [18], OpenShift [19]

2.2.4 IaaS: infrastructure as a Service

IaaS or cloud infrastructure services, consist in highly scalable pay-as-you-go services such as storage, networking, and storage, via virtualization. It's a self-service model in the sense that allows businesses to access and monitor things like servers, networking, storage, and other services, and it allows them to purchase resources on-demand instead of having to invest in expensive on-premise resources.

- Delivery

  The infrastructure provided by these services usually include servers, network, operating systems, and storage, through virtualization technology. These cloud servers are usually accessible through a dashboard or an API, which provide full-access to the infrastructure. IaaS offers the same capabilities as a traditional data center without having to manage or maintain all of it, which is costly and labor-intensive.

- Advantages

  - Flexible and highly scalable
  - On-demand resources
  - Complete control over the infrastructure
  - Accessible by multiple users
  - Cost-effective

- Characteristics

  - Resources are available as a service
  - Services are highly scalable
  - Offers complete control over the infrastructure
  - Flexible and dynamic, as they can be replaced at any time

- Examples
  - Digital Ocean [20], Linode [21], AWS EC2 [22], Google Compute Engine [23]

2.2.5 On-premise

Before cloud computing solutions were made available, companies had to rely on on-premise solutions which can be costly in terms of hardware and maintenance. This model is usually not as scalable as cloud services, since it requires hardware updates, and in a situation of downgrade, it results in resource wasting. Although this type of hosting is not used in most situations nowadays, there are still certain companies that benefit from it. Industries and services like healthcare and financial services companies are usually required to have their infrastructure hosted locally or on an authorized local third-party provider as they are required to collect, store, and process private data. Public and government institutions also usually require on-premise infrastructures as a matter of national security.

2.2.6 Conclusion

The increasing popularity of IaaS, PaaS, and SaaS is reducing the need for on-premise hosting. Its crucial for a business to understand the differences between each model and choose accordingly. Each model offers different features and functionalities and different levels of control over the entire system which on-premise hosting cannot provide.

- IaaS provides the most flexibility, specially when building custom applications, as well as providing a general data center for storage.
- PaaS is built on top of an IaaS platform in order to reduce the need for system administration. It allows developers to focus on the development of the application instead of infrastructure management.
- SaaS offers ready-to-use, out-of-the-box solutions that meet a particular business need. Usually built on top of IaaS or PaaS platforms.

2.3 Data Warehouse

2.3.1 Concept

A data warehouse (DW), also known as enterprise data warehouse (EDW), is a core component of business intelligence, which enables data consolidation, analysis and reporting. It integrates data from multiple sources, usually on a regular frequency, in a central repository. In a data-centric age, data and analytics provide a lot of value to businesses in order to stay competitive. The use of reports, dashboards, and analytics tools to extract insights from their data, allows them to monitor business performance and support decision making. These resources are powered by data warehouses, which store data efficiently and deliver query results at fast speeds, to multiple users concurrently.

2.3.2 Architecture

Traditional data warehouses are typically structured in three tiers:

- Bottom tier: a database server, usually a RDBMS, that stores extracted data from different sources
- Middle tier: an OLAP or OLTP server that implements the operations, transforming the data
- Top tier: query and reporting tools for analysis and business intelligence

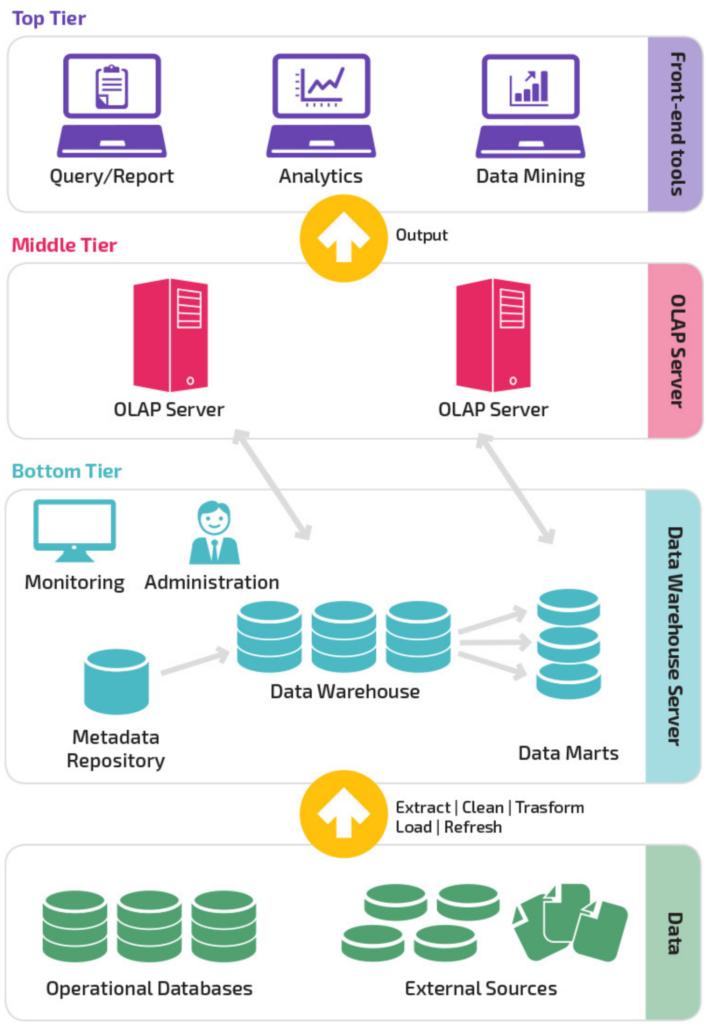


Figure 2.2: Data Warehouse architecture [1]

2.3.3 Design

When designing a DW, there are three approaches worth considering:

- Top-down: also known as the Inmon approach [2], its where the data warehouse is built first and is seen as the core component of the analytical environment. Data is extracted, summarized and distributed from the centralized warehouse to one or more data marts.

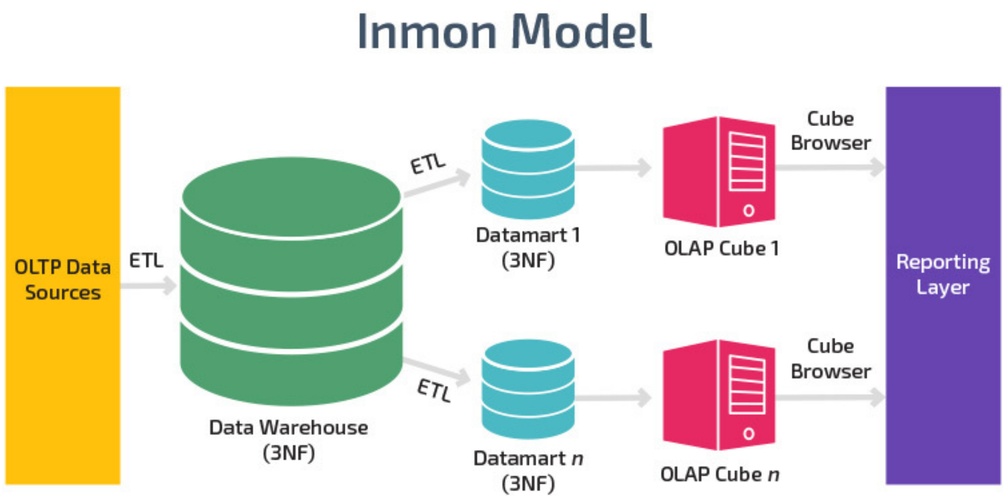


Figure 2.3: Inmon's (top-down) design approach [2]

- Bottom-up: also know as the Kimball approach [2], its where data marts are created first. Each data mart focuses on a specific business process, and are later integrated using a data warehouse bus architecture.

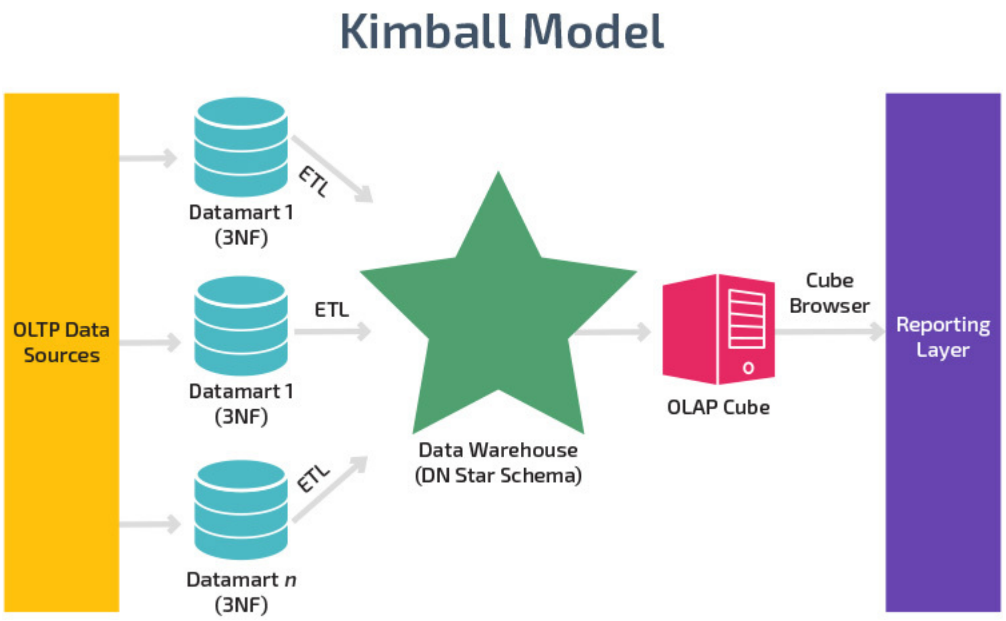


Figure 2.4: Kimball's (bottom-up) design approach [2]

- Hybrid: this approach includes aspects from both the top-down and bottom-up methods. It seeks to combine the speed of the bottom-up approach with the integration achieved in a top-down design.

2.4 Data Ingestion

Data ingestion is a process by which data is moved from one or more sources to a destination, whether for immediate use or stored for further analysis. Sources may be in-house applications, databases, SaaS, or anything capable of generating data. The destination is typically a data warehouse, data mart, database, or a document store. Incoming data may be in different formats and usually requires cleaning and transformation in order to be analyzed together with data from other sources. Data can be ingested in batches or streamed in real-time. When data is ingested in batches, data items are imported regularly in scheduled intervals, which is useful to generate logs or reports. When data is ingested in real time, each item is imported as it is emitted by the source, which is useful when the information is time-sensitive such as in monitoring critical systems. An efficient data ingestion process starts by prioritizing data sources, validating individual files and routing data items to the correct destination.

2.4.1 Data Ingestion Paradigms

For a long time, data ingestion paradigms would rely on an extract, transform and load (ETL) procedure, in which data is taken from the source, manipulated to fit the schema of the destination system or the needs of the business, then added to that system.

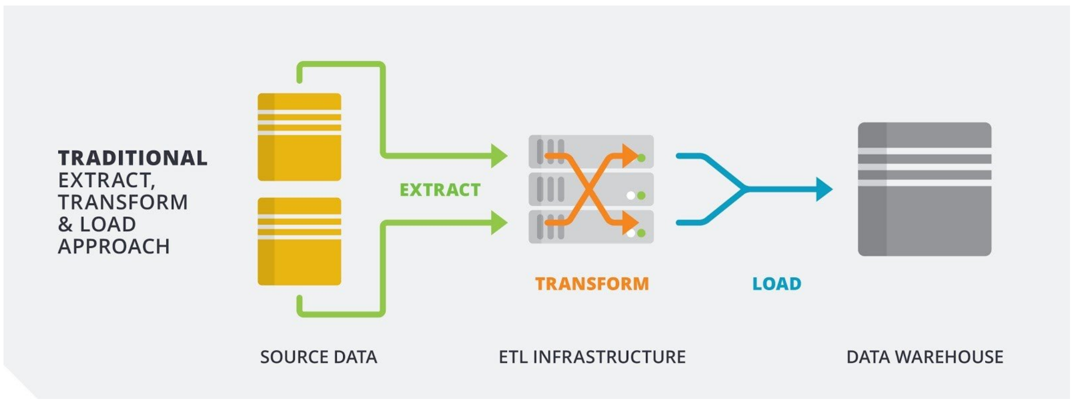


Figure 2.5: ETL paradigm

Today, cloud data warehouses like Amazon Redshift [24], Google BigQuery [25] and Microsoft Azure SQL Data Warehouse [26] can scale compute and storage resources in a matter of seconds or minutes in a cost-effective way. This allows data engineers to skip the transformations step and load all the raw data into the storage system. This raw data can then be transformed by using SQL at query time. This new approach has changed ETL into ELT, which is an efficient way to replicate data in a cloud infrastructure.

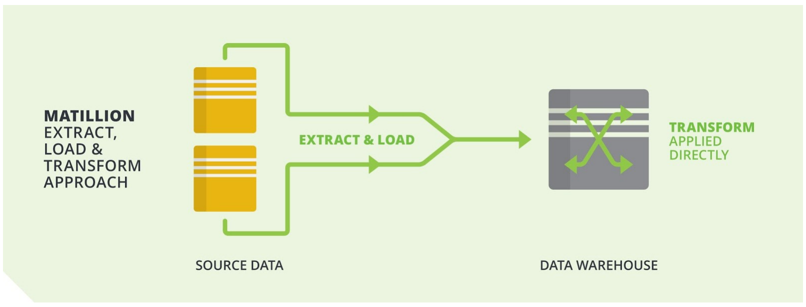


Figure 2.6: ELT paradigm

ETL is not only used to replicate data to a cloud platform, but it removes the need to write complex transformations as part of the data pipeline. Most importantly, ETL gives different teams the freedom to develop ad-hoc transformations according to their particular needs.

2.4.2 Challenges

One of the main aspects of delivering value though the use of BI tools, analytics or any insights powered by data pipelines, is to ensure data is being ingested efficiently. A faulty ingestion pipe can emit incomplete and inaccurate data, which will result in errors. It's important to know the limitations and properly design the ingestion platform before the actual implementation. Some of the main challenges are described below:

- Multiple source ingestion

  In order to draw better insights from combined data, it's important to define what data is needed and how it can be transformed and ingested into different systems. This also includes the definition of the best data store and format to store the data in.

- Scaling

  Heavy-data applications can have a very fast growth in terms of velocity and volume of data. To mitigate this, it's important that companies realize the need to scale up before choosing the solution they are planning to adopt, for example, between on-promises or a managed service.

- Moving data

  Data must be transported and processed between the sources and the destination. There are primarily two approaches to this: batch jobs and streams. The choice between these approaches depends mainly on the necessity of real-time analysis. A more in-depth explanation is presented on the next section.

2.4.3 Evolution of distributed streaming platforms

Due to the rapidly expanding data scale, distributed streaming platforms have been under research for over a decade. Most of the early research focused on single-node systems such as

Aurora [27], which are currently outdated due to their inefficiency. The introduction of MapReduce [28] has revolutionized modern distributed streaming platforms given its scalability and fault-tolerance features. MapReduce has become the core of multiple data stream processing frameworks such as S4 from Yahoo [29], Apache Storm [30], Apache Samza [31], Apache Spark [32] and TimeStream from Microsoft [33]. Although most of these solutions are widely used, they are built for batch-processing applications. Due to the requirements of low-latency high-update frequency data streams, stream-processing platforms, such as Apache Kafka Streams [34], Apache Spark and Apache Druid [35], have been developed to better fit this need.

2.4.4 Data processing

2.4.4.1 Batch processing

The analysis of large sets of data, which are collected over past time periods, is done with batch processing. This is the most common approach to data ingestion. In this approach, the ingestion layer collects and groups source data periodically and sends it to the destination system. The groups of data can be defined in multiple ways: any logical ordering, conditional, scheduled or by size. Batch processing is used when data is not time-sensitive and there's no need for real-time analysis, which usually results in an easier and more affordable solution than streaming ingestion.

2.4.4.1.1 MapReduce

MapReduce is a programming paradigm that enables processing and generating Big Data with a parallel, distributed algorithm on a cluster. The term "MapReduce" refers to two different and separate tasks. The first is the map job, which processes tuples by running a user-defined mapping function, usually filtering and sorting, and generates a new, intermediate tuple for the reduce job. This reduce job performs a summary operation, usually merging intermediate values associated with the same key.

The "MapReduce Framework" orchestrates the processing of data by automatically running tasks in parallel, managing all communications and data transfers between multiple sources and destinations, between multiple distributed servers, and also providing redundancy and fault tolerance features.

Though multiple libraries have been written in many programming languages, with different levels of optimization, the most popular implementation of this model is the open-source software Apache Hadoop [36]. Most cloud providers offer PaaS solutions based on Hadoop: AWS EMR [37], Google Cloud Dataproc [38] and Microsoft Azure HDInsight [39].

2.4.4.2 Stream processing

A stream-based application needs to be able to ingest, process and analyze data in real-time, and, in many cases, combine multiple streams, both live (unbounded streams) and archived (bounded streams), in order to provider better insights. To this end, Big Data processing runtimes (eg:

Apache Spark, Apache Kafka Streams and Apache Druid) designed for both batch and stream processing are being widely adopted and replacing more traditional batch-oriented processing models (eg: MapReduce, Hadoop) that are unable to fulfil the requirements of low-latency high-update frequency data streams. Some PaaS solutions include Amazon MSK [40], Confluent Cloud on GCP [41] and Microsoft Azure HDInsight.

2.4.4.3 Micro-batching processing

Some streaming platforms (such as Apache Spark Streaming) also use batch processing. The difference is that ingested groups of data are smaller or prepared at shorter intervals, though not processed individually like in pure streaming platforms. This type of processing is often called micro-batching.

2.4.5 Message brokers

To make the process of data exchange simple and reliable, communication between sources and data processing platforms usually rely on message brokers. These message brokers can act as a buffer for messages, to ensure a logical order between incoming messages, and route the messages to multiple applications or users. It effectively creates a centralized store for data which becomes the source of truth.

For a distributed architecture, there are four typical message patterns that are widely used [42]:

- PAIR: communication is established strictly within one-to-one peers
- Client/Server: server distributes messages according to client's requests
- Push/Pull: messages are distributed to multiple processors, arranged in pipeline
- Pub/Sub: connects a set of publishers to a set of subscribers

In a context of data ingestion, not all four patterns are suitable for stream processing, which needs to handle high velocity and large volume of data. The PAIR message pattern clearly limits the distribution of large volume of messages since it distributes messages on an one-to-one basis, which makes it not suitable for a processing node due to unpredictable nature of data content and format. The above mentioned problem also affects the Client/Server pattern. The remaining two patterns - Push/Pull and Pub/Sub - are the common choice for stream processing systems. The Push/Pull pattern is a one-way stream processing pattern, where the upstream nodes invoke the downstream nodes when they are finished processing tasks. This means that streams can go through several processing stages, depending on the number of consumers, and no messages are sent upstream. Pub/Sub is the most popular messaging pattern. Systems that implement this pattern have a broker that manages the distribution of messages, by matching the topics subscribed by receivers to the topics published by producers. Any application can be a subscriber or a publisher, and the broker will always transmit the topics from the publisher's end to the subscribers' end.

There are various message brokers available, being the most popular Apache Kafka and RabbitMQ [43]. There are also PaaS solutions namely Amazon Kinesis [44], Google Cloud PubSub [45] and Microsoft Azure Event Hub [46].

2.5 Data visualization

2.5.1 Concept

As seen in the section Big Data 2.1, the amount of data being created is growing every year, and shows no sign of slowing down. Even if the data pipeline is properly designed and implemented, this data is only useful if valuable insights can be extracted from it and acted upon. Data visualization is the representation of data in a graphical format. Its purpose it to make it easier to understand data values and communicate important concepts and ideas. Although this is not the only way to analyze data, data visualization offers a fast and effective way to:

- Interpret information quickly: using a graphical representation allows businesses to interpret large amounts of data in clear, cohesive ways and draw conclusions from that data. Given the way the human brain works, the analysis of data in graphical form is usually easier and faster than looking at spreadsheets or reports with hundreds of lines.
- Spot trends: spotting trends in large amounts of data is usually hard, specially when there are multiple sources. Big Data visualization techniques can make it easy to spot these trends and give businesses an edge over the competition.
- Identify correlations: a data visualization tool allows businesses to not only get answers to their questions, but also to discover what unexpected insights the data can reveal. Identifying relationships and unexpected patterns in data can provide huge value to businesses.
- Present the data to others: When new insights are uncovered, usually the next step is to present those insights to others. The use of a graphical representation provides an effective and easy way to understand and gets the message across quickly.

2.5.2 Implementation

2.5.2.1 Requirements

Before adopting any data visualization tools, it's necessary to have a solid grasp on the data, and define the goals, needs and audience of such tools. In order to prepare a business for data visualization technologies, some requirements should be met.

- Understand the data, including its size and cardinality (the uniqueness of data values)
- Determine what metrics and information should be visualized
- Know the target audience and how it processes visual information
- Use visuals that get the message across quickly, in an effective and easy way

2.5.2.2 Techniques

One of the biggest challenges with data visualization is choosing the most effective way to visualize the data to surface any insights it may contain. In some cases, simple BI tools such as pie charts or histograms may show the whole picture, but with large and diverse data sets, more advanced and complex visualization techniques may be more appropriate. Some examples of data visualization techniques include:

- Linear: Lists of items.
- 2D/Planar/geospatial: Cartograms, dot distribution maps, symbol maps, contour maps.
- 3D/Volumetric: 3D computer models/simulations.
- Temporal: Timelines, time series charts, scatter plots.
- Multidimensional: Pie charts, histograms, tag clouds, bar charts, tree maps, heat maps.
- Tree/hierarchical: Dendrograms, radial tree charts, hyperbolic tree charts.

2.5.3 Challenges

Big Data visualization can provide a lot of value to businesses, but before being able to take advantage of it, some issues need to be addressed. These include:

- Data quality: The insights provided by Big Data visualization are only as accurate as the data that is being visualized. If the data is inaccurate or out of date, then any insights that result from it are not reliable.
- Hardware resources: Big Data visualization is essentially a computing task, which can be more or less demanding depending on the type of analysis required. This may demand the use of powerful expensive hardware, fast storage systems, or even move to a cloud service.

Other challenges include perceptual and interactive scalability. The first usually happens when visualizing large data sets where every data point is represented, leading to over-plotting and consequently overwhelming users' perceptual and cognitive capacities. The solution is to reduce the data through sampling or filtering. The scalability issues arise when querying large data stores which leads to high latency, disrupting the user experience.

2.5.4 Fleet tracking

2.5.4.1 Concept

Fleet tracking is a management system that uses vehicle tracking devices, usually equipped with a GPS or GLONNASS module, and software to monitor and manage a fleet of vehicles. The typical architecture of these systems includes:

- GPS tracking: A device is installed in the vehicle and captures GPS information, apart from other vehicle information such as CAN data, at regular intervals, and sends it to a remote server. The CAN data can include fuel capacity, engine temperature, altitude, door status, tire pressure, ignition status, battery status, odometer, engine RPM, throttle position, among others. Some of these systems will also include connection ports to remotely control the vehicle, through its ECU.

- Remote server: The server is responsible to receive, process and store data. Usually, it also serves the data through an API.

- User Interface: The user interface provides a way for users to access fleet information in real-time, past data and generate reports.

These systems can be used for several purposes, such as: fleet tracking, routing, dispatching and security. More advanced use-cases include monitoring schedule adherence and automation jobs based on vehicle location or routes. In early 2009, the American Public Transportation Association estimated that around half of all transit buses in the United States were already using a GPS-based tracking system to trigger automated stop announcements. [47] By gathering location data in real-time as a transit vehicle follows its route, a computer is able to assert its location relative to the next stop and trigger the announcement at the right time. The schedules are also updated in real-time by comparing the current location and time with the programmed schedule. Some transit agencies also provide online maps with real-time location of their vehicles to provide information regarding waiting times, so the costumers can base their schedule on real data instead of the schedulled time of arrival.

Other scenarios where this technology is used include:

- Vehicle recovery: The tracking devices, usually equipped with GPS modules, provide real-time infomation which allow police to track and recover the vehicle. Some devices include remote control, which can shut down the vehicle when necessary.

- Field service management: Companies that have a field service workforce for services such as maintenance, cleaning and repairs, are able to plan field worker's schedules efficiently or site arrival information to costumers.

- Fuel Monitoring: Monitor the fuel through the tracking device in combination with a fuel sensor.

- Asset tracking: Allows companies to track valuable assets for insurance purposes on to closely monitor movement and operating status.

2.5.5 Tools

There are a lot of Big Data solutions in the market, including big names such as Microsoft (Microsoft Power BI [48]), SAP (SAP Lumira [49]), SAS (SAS Visual Analytics [50]) and IBM

(Watson Analytics [51]). Other specialist software vendors include Tableau Software [52], Qlik [53] and TIBCO [54]. Libraries like D3 [55] and Fusion Chart XT [56] enable the development of web-based custom data visualization solutions.

Chapter 3

Development

3.1 Real Time Location

3.1.1 Data Ingestion

The location data is generated by CarTrack's GPS modules installed in Wegho's vehicles. To access this data, the CarTrack's API provides an endpoint *get_vehicle_detailed_event* which returns, among other data, the GPS coordinates of the vehicle. The request and response data schemas are detailed below.

Request

Field	Type	Description
registration	string	The vehicle's registration
start_ts	date	Start date
end_ts	date	End date
username	string	CarTrack credentials

Table 3.1: Request schema for *get_vehicle_detailed_events endpoint*

Response

Field	Type	Description
id	number	
registration	string	Vehicle's registration
event_ts	date	Event timestamp
event_description	string	Event description
longitude	decimal	GPS longitude
latitude	decimal	GPS latitude
odometer	number	Vehicle's odometer value in km
bearing	number	Vehicle's bearing in degrees
ignition	boolean	Engine ignition status
speed	number	Speed in kmh
mapzone		
temp1	decimal	Sensor 1 temperature in C^o
temp2	decimal	Sensor 2 temperature in C^o
temp3	decimal	Sensor 3 temperature in C^o
temp4	decimal	Sensor 4 temperature in C^o
door1_status	boolean	Driver door status
door2_status	boolean	Passenger door status
received_ts	date	Event received timestamp

Table 3.2: Response schema for *get_vehicle_detailed_events endpoint*

Its important to note that this endpoint doesn't always return data; it requires a CAN Bus event to be triggered in order to generate data regarding that event. This results in no data being returned when the vehicle is off.

If the car is moving, an event is generated every 10 seconds - the maximum update frequency of the location data. Otherwise, the response will be either empty or contain information about an event generated by the CAN Bus. Although these events are not documented in the API, it was possible to extract a list of events from several requests.

Name	Type/Values	Description
ign	boolean	Ignition status
dHDG	Unknown	Unknown
dODO	number	Odometer
reverse	boolean	Reverse status
motion	start/end	Vehicle starts/stops moving

Table 3.3: CAN Bus events provided by *get_vehicle_detailed_events*

Now that the maximum frequency of location data updates is known, it is necessary to develop a solution to fetch it periodically, in an automated way. To achieve this, a script was developed to make requests to the API. This script relies on a time-based job scheduler, also known as cron job, to automate its execution. cron is a Linux utility which schedules a command or script to run automatically at a specified time and date. A cron job is the scheduled task itself.

The script was deployed in Google Cloud Platform using Cloud Functions. A Cloud Function is a "serverless execution environment for building and connecting cloud services" [3], in which the user writes a single-purpose event-driven function that is executed in a managed environment.

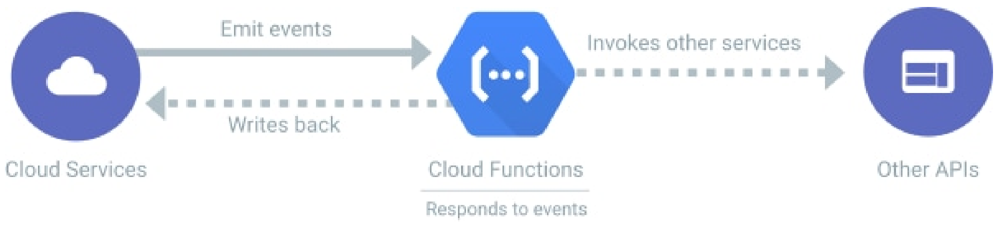


Figure 3.1: Google Cloud Functions overview [3]

Unlike the traditional way of running a function (or script) which involves setting up and managing an infrastructure, Cloud Functions provide an easy way to run the code in the cloud in a fully-managed, highly available and fault tolerant environment. Other advantages include the billing structure - the user is only billed when the function runs - and easy integration with other cloud services, such as Google Firebase. For this particular use-case, given the maximum update frequency of 10 seconds of the location data (ie: the function doesn't need to run constantly) and the connection to Google's Firebase results in a great fit for using Google Cloud Functions.

A Cloud Function is event-driven, so it requires an event to trigger its execution. There are multiple supported event types: Cloud Pub/Sub, Cloud Storage, HTTP, Stackdriver Logging and Firebase. As previously established, the function needs to be executed periodically which requires a time-based trigger. Although it's not possible to use a job scheduller as an event source directly, Google Cloud Platform provides a work-around [57] for this use-case. The solution is to use Cloud Pub/Sub - a messaging queue service that allows sending and receiving messages between applications - which acts as a bridge between the Cloud Sheduler job and the Cloud Function.

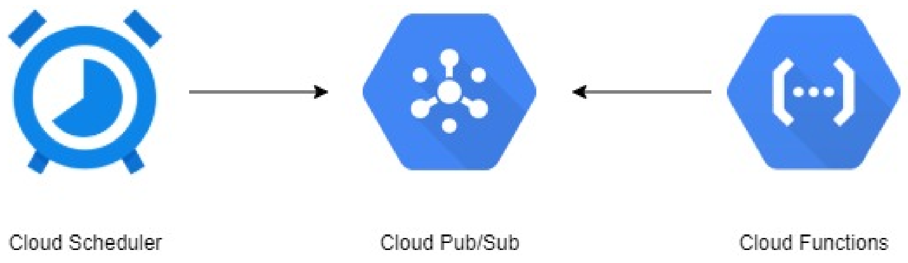


Figure 3.2: Using Pub/Sub as a bridge between Cloud Scheduler and Cloud Functions

In order to setup this workflow, it's required to:

- Create a topic in Pub/Sub

- Create a job in Cloud Scheduler that runs every 10 seconds and sends a message to a Pub/Sub topic when the job is executed
- Create a script for the Cloud Function, and set its execution trigger to the Pub/Sub topic created

The script was developed in JavaScript, and uses the strong-soap library [58] to make the requests to the CarTrack's SOAP API. In order to manage the API credentials, Google Key Management System [59] was used. This system allows the credentials to be encrypted in the script and only decrypted on run-time. It also requires execution previleges which are granted by the Function owner, which limits the exposure if someone tries to decrypt the credentials outside a particular environment. In order to store the location data, Google's Firestore SDK [60] provides an easy way to access the database from the Function environment.

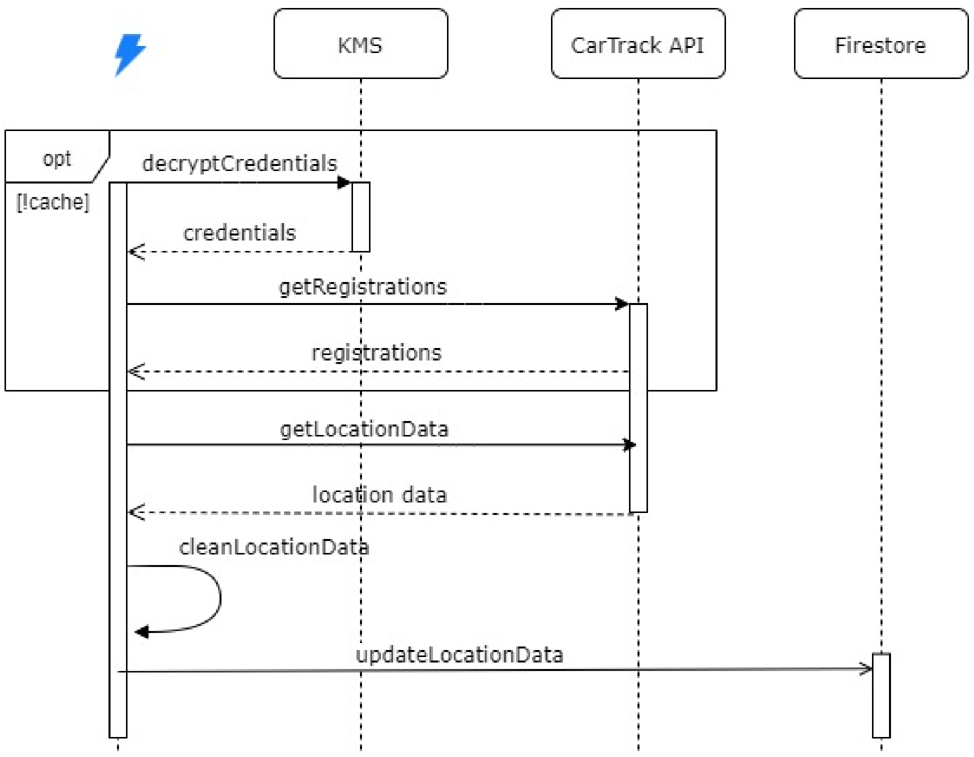


Figure 3.3: Location function sequence diagram

The script starts by decrypting the API credentials using KMS. Then, it makes a request to *get_vehicle_details* to fetch all vehicle registrations. This request is necessary in order to make the function future proof, in case a new vehicle is added to Wegho's fleet or an exiting vehicle is removed, the only action required is to restart the function. Both the API credentials and all vehicle registrations are served from cache [61] in subsequent calls, so it only runs on the first execution. This saves execution time which results in faster data updates.

The next step is to make a request to *get_vehicle_detailed_events* in order to fetch the location data. This request is executed for each registration. To mitigate a negative impact on performance, all the requests are prepared and executed in parallel. Ideally, this endpoint would allow multiple registrations to be queried on a single request.

Finally, the response data is filtered to remove unnecessary data fields and match the database schema, and then sent to Firebase. This process is described in detail in the following chapters.

3.1.2 Data Transformation

The data schema returned by the API endpoint consists of a large number of properties and values that are not relevant for this system. In order to save storage space and, most importantly, match the storage data schema, only the relevant data is kept and passed onto the next step.

Field	Type	Description
registration	string	Vehicle's registration
event_ts	date	Event timestamp
longitude	decimal	GPS longitude
latitude	decimal	GPS latitude

Table 3.4: Filtered response schema for *get_vehicle_detailed_events endpoint*

At this stage, invalid or duplicate data is also removed.

3.1.3 Data Storage

The data storage solution must be capable of handling batch updates, multiple times per minute (limited to the number of vehicles times 10s of maximum frequency, as described in the previous sub-chapter) and provide a subscription-like feature to feed the data into the data view layer in real-time. Most database engines, SQL and NoSQL, such as PostgreSQL and MongoDB, offer both features, making any of them a valid choice - PostgreSQL uses the *notify* command [62] while MongoDB uses *change streams* [63]. For this particular case, a NoSQL solution was chosen due to its scalability features, better performance in write and read queries and the fact that there's no need for relational queries.

In regards to NoSQL, there are on-premises and PaaS solutions, namely MongoDB and Google Firebase, and given the amount of data being relatively low, the cost of running and managing an infrastructure for a MongoDB database outweighs its benefits, hence the decision of using Firebase.

A collection named "geo-location" was created to store location data. The schema of the documents matches the filtered schema 3.4 and each vehicle has its own document. Each document is updated whenever a location update is received (ie: the location of the vehicle changes).

3.2 Trips

3.2.1 Data Ingestion

As in the previous chapter 3.1.1, trip data is also provided by CarTrack. By taking advantage of the ignition property in the location data, CarTrack can generate trips by looking for changes in this property. When the ignition property changes from "off" to "on", a new trip is started; otherwise, a trip is ended. While the ignition key has the same value, the trip distance and duration is updated. The trip data is then stored in CarTrack's database and made accessible through their API, specifically, the *get_all_trips* endpoint. This endpoint has the following schema:

Field	Type	Description
registration	string	The vehicle's registration
start_ts	date	Start date
end_ts	date	End date
username	string	CarTrack credentials

Table 3.5: Request schema for *get_all_trips endpoint*

Field	Type	Description
vehicle_id	number	Vehicle's id
start_timestamp	date	Trip start timestamp
end_timestamp	date	Trip end timestamp
start_location	string	Trip start location
end_location	string	Trip end location
trip_distance	number	Trip total distance in meters
trip_time	time	Trip total duration
registration	string	Vehicle's registration
driver_id	number	Driver's id
driver	string	Driver's name
client_vehicle_description	string	Vehicle's description

Table 3.6: Response schema for *get_all_trips endpoint*

The solution used for extracting the trips data is similar to the solution described in section 3.1.1 - a Cloud Function triggered by a Pub/Sub message sent by a Scheduler event. The difference being the execution frequency of the Scheduler job which, in this case, is every 5 minutes. Unlike the location data, the non-periodic nature of trips makes it unnecessary to use a faster frequency, so this value was defined according to the project requirements, in particular, the usage of trips data by the end-users on the data visualization tool.

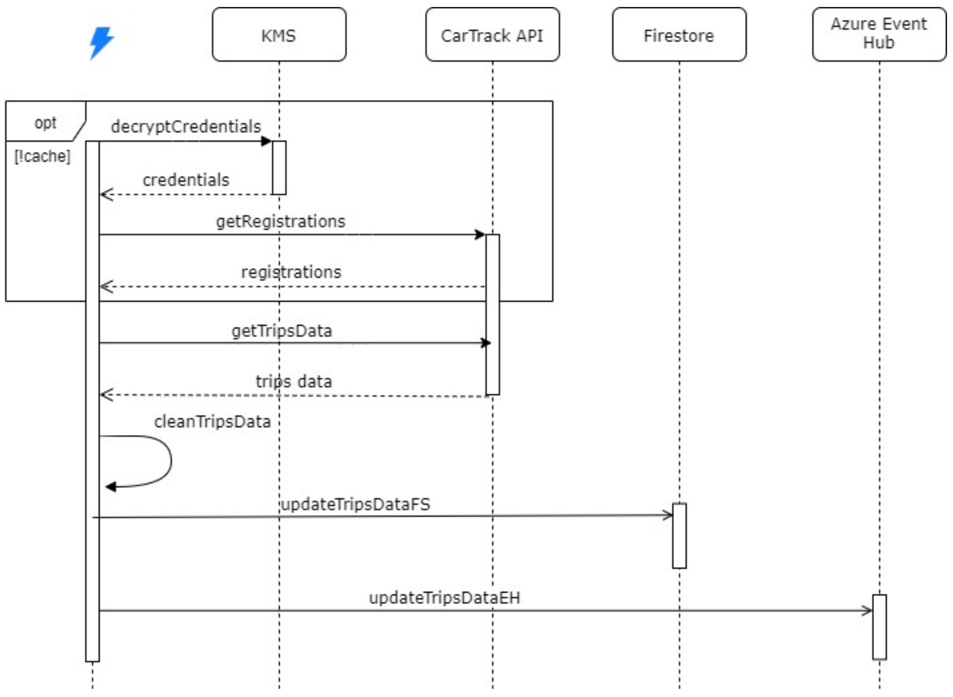


Figure 3.4: Trips function sequence diagram

The first part of the script uses the same principle as the location data script described in the previous section 3.1.1, where the API credencials are decrypted and a request is made to CarTrack's API in order to fetch all vehicle registrations.

Afterwards, a request is made to *get_all_trips* in order to fetch the trips data. This request is executed for each vehicle registration. The response data is then filtered to remove unnecessary data and match the database schema, and then sent to Firebase and Azure Event Hubs. This process is described in detail in the following chapters.

3.2.2 Data Transformation

Similarly to the location data transformation described in section 3.1.2, only a part of the data returned by the trips endpoint is relevant for this use-case. The filtered data retains the relevant fields, which are illustrated below.

Field	Type	Description
start_timestamp	date	Trip start timestamp
end_timestamp	date	Trip end timestamp
trip_distance	number	Trip total distance in meters
trip_time	time	Trip total duration
registration	string	Vehicle's registration

Table 3.7: Filtered response schema for *get_all_trips endpoint*

After filtering the data, there are more steps required in order to properly process trips. Given the update frequency of 5 minutes, the trips returned by the API are not actual trips, but parts of trips - which were named "sub-trips". Since the API does not provide a unique identifier for each trip, the solution to handle these "sub-trips" is to query the database before actually inserting or updating the trip data. This way, it's possible to assert whether the "sub-trip" is actually a valid "sub-trip", an invalid "sub-trip" or a new trip. These "sub-trips" were identified by executing multiple requests to the API, and are described below:

- "Ghost" sub-trips

  These sub-trips have the same value for *start_timestamp* and *end_timestamp*. Since trips with 0 duration are not possible, these were called "Ghost" sub-trips and are discarded.

- Sub-trips fully contained in a trip

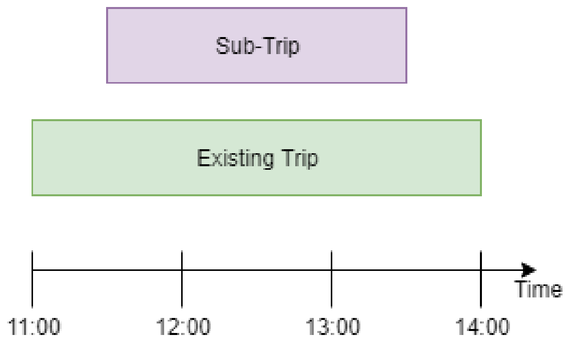


Figure 3.5: Sub-trip contained in a trip

  These sub-trips have a *start_timestamp* greater than an existing trip *start_timestamp*, and an *end_timestamp* smaller than an existing trip *end_timestamp*. This means that the sub-trip is contained in an exisitng trip, and can be discarded as it doesn't add new information.

- Sub-trips with different start timestamps

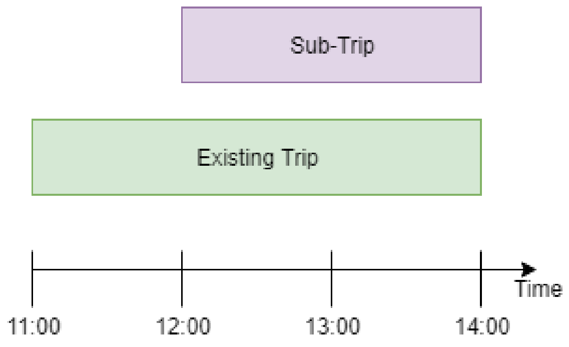


Figure 3.6: Sub-trip with different start timestamp

  These sub-trips have the same *end_timestamp* as an existing trip, but a different *start_timestamp*. These sub-trips are discarded since they are part of an existing trip.

- Sub-trips with different end timestamps

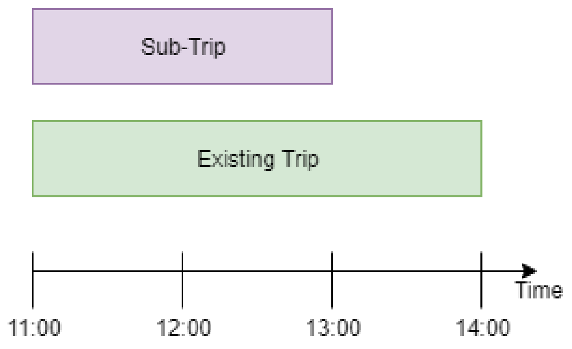


Figure 3.7: Sub-trip with different end timestamp

These sub-trips have the same *start_timestamp* as an existing trip, but a smaller *end_timestamp*. Since a trip already exists with the same *start_timestamp* value, it is correct to assume that the new *end_timestamp* is valid and means that the trip is still ongoing.

- New trip

After checking all the previous scenarios, it is safe to assume that a new trip has started. In this case, a new document is created.

3.2.3 Data Storage

The trips data will be used for different purposes: a web application, and report generation using business inteligence tools. These applications have different requirements, so different storage solutions were used.

The web application requires trips data to be updated regularly, so the user can check recent trips for every vehicle. To this end, the trips data is stored in Firebase, similarly to the location data. A collection named "geo-trips" was created, where the schema of the documents match the filtered schema 3.7. Each document represents a trip.

The BI tools require business data to be taken into account, so they can generate better and useful insights. Currently, Wegho's operational data is stored on a SQL Server database, hosted on Azure. Since the database is not accessible externally, the trips data has to be stored in Azure as well. Although Azure offers different database engines, the need for relational queries and the fact that Wegho's operational database uses SQL Server, made decision to use a new SQL Server database straightforward.

Two different solutions for sending the trips data to the database were considered: a Cloud Function running on Azure, and Azure Event Hubs. The cloud functions on Azure work similarly to Google Cloud Functions, while Event Hubs can be compared to Google Pub/Sub - a fully-managed, real-time data ingestion service. The advantage of using Azure Event Hubs is its integration with Azure Stream Analytics - a real-time analytics service - which allows trips data to be joined with business data in different SQL Server databases, in an easy way. Any of the solutions would be a valid choice, but Event Hubs ended up being the one used for this.

3.2.4 Report

In order to generate a daily report, a business analytics software is the most suitable tool for the job. To this end, Microsoft Power BI was used to generate the report. Power BI provides integration with multiple data sources, including SQL Server, and has a variety of features including custom dashboards, advanced reporting options, powerful data governance tools, and ad hoc analysis.

The goal of the daily report is to improve efficiency in the schedulling of bookings and routing of vehicles. It can also be used to account for non-work related trips, which are allowed but not unlimited. One way to accomplish this is by comparing estimated trips duration and distance with the actual trips duration and distance. The estimated trips data is generated by a Cloud Function hosted in Azure. This function starts by fetching all bookings for a given day from Wegho's operational database, groups them by vehicle registration and sorts them by start date, and makes an API request to Mapbox Directions API [64] to get data regarding distance and duration. This data is saved in the same database so it can be used for analytics and optimization processes. As described in the previous section 3.2.3, the actual trips data is stored in a SQL Server database, also hosted on Azure, which makes the process of joining data simpler.

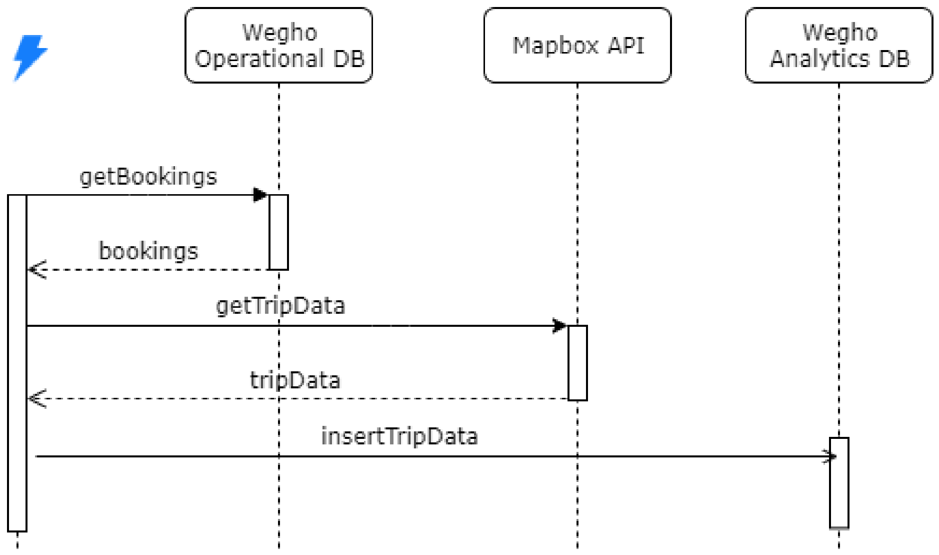
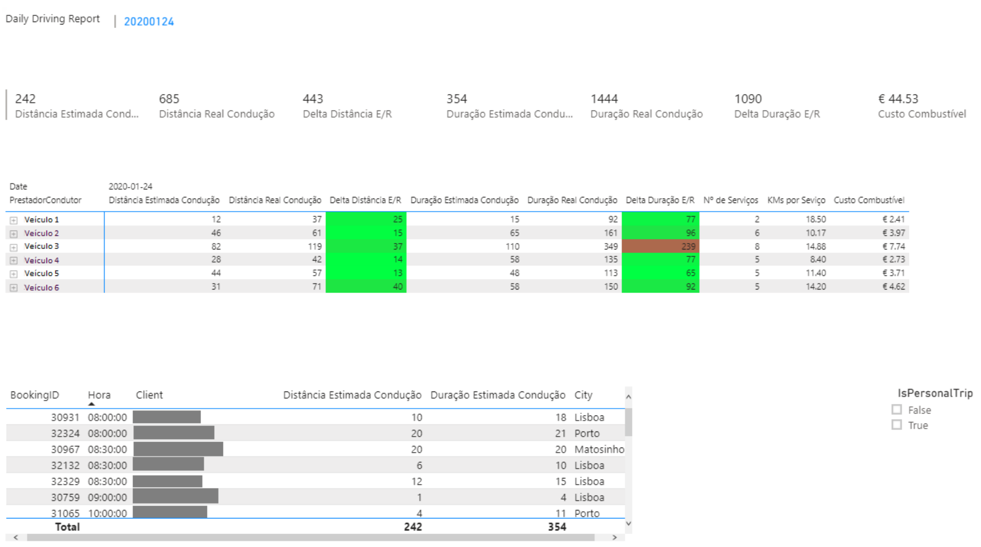


Figure 3.8: Trips Cloud Function

The report created in Power BI is powered by a SQL query that retrives all data required for populating the report fields. An example of such report can be seen below.


Daily Driving Report | 20200124

242	685	443	354	1444	1090	€ 44.53
Distância Estimada Cond...	Distância Real Condução	Delta Distância E/R	Duração Estimada Condu...	Duração Real Condução	Delta Duração E/R	Custo Combustível

Date	2020-01-24								
PrestadorCondutor	Distância Estimada Condução	Distância Real Condução	Delta Distância E/R	Duração Estimada Condução	Duração Real Condução	Delta Duração E/R	Nº de Serviços	KMs por Seviço	Custo Combustível
Veículo 1	12	37	25	15	92	77	2	18.50	€ 2.41
Veículo 2	46	61	15	65	161	96	6	10.17	€ 3.97
Veículo 3	82	119	37	110	349	239	8	14.88	€ 7.74
Veículo 4	28	42	14	58	135	77	5	8.40	€ 2.73
Veículo 5	44	57	13	48	113	65	5	11.40	€ 3.71
Veículo 6	31	71	40	58	150	92	5	14.20	€ 4.62

BookingID	Hora	Client	Distância Estimada Condução	Duração Estimada Condução	City
30931	08:00:00	[redacted]	10	18	Lisboa
32324	08:00:00	[redacted]	20	21	Porto
30967	08:30:00	[redacted]	20	20	Matosinho
32132	08:30:00	[redacted]	6	10	Lisboa
32329	08:30:00	[redacted]	12	15	Lisboa
30759	09:00:00	[redacted]	1	4	Lisboa
31065	10:00:00	[redacted]	4	11	Porto
Total			242	354	

Figure 3.9: Trips Report (Power BI)

The report is generated every day regarding trips from the previous day. The first section is a summary of the most important data: the sum of estimated trips data and the sum of actual trips data, deltas and gas cost. The table below shows the individual trips data by vehicle driver. The duration and distance deltas are highlighted from green (small delta) to red (large delta) since these are the values that are most relevant. The bottom table shows all bookings assigned to each vehicle.

3.3 Web Application

3.3.1 Technologies

3.3.1.1 Languages

The core languages in front-end development are HTML, CSS and JavaScript. Although a simple web site can be built with just HTML, current web sites and web applications use all three languages. It's also common to include jQuery, a JavaScript library that provides an abstraction and easy-to-use API for DOM manipulation, event handling, animation, and AJAX requests. It was originally used for browser compatibility, since it provides polyfills for modern Web APIs that are not available in older browsers. Nowadays, browser compatibility has improved a lot, and jQuery is not being used as much. On the other hand, JavaScript libraries and frameworks such as React [65], Angular [66] and Vue [67] have gained popularity due to their performance and features.

- HTML

HTML stands for Hyper Text Markup Language, and it's the standard markup language for the Web. HTML elements are represented by <> tags and are the building blocks of HTML pages. These elements can have attributes which come in keyvalue pairs, such as id="some_id", and provide additional information about the element. The HTML specification, written by WHATWG, defines all HTML elements, besides other Web related technologies. Some of these elements include: <body> which is the element that contains the visible page content, <h1> which is an heading, and <p> which is a paragraph.

- CSS

  Cascading Style Sheets, or CSS, is a style sheet language used for adding style (e.g. fonts, colors, spacing) to Web documents. It's designed to enable the separation of presentation and content. CSS is a rule-based language, where each rule consist of one or more selectors, and a declaration block. A selector targets the HTML element that is going to be styled. Inside the declaration block, one or more declarations, which take the form of property and value pairs, specify a property of the selected element(s) and its value.

- JavaScript

  JavaScript, also referred to as JS, was initially created to "make web pages alive". It is a high-level, just-in-time compiled, multi-paradigm programming language which implements the ECMAScript standard. The programs written in this language are called scripts. They can be written directly in an HTML document, using the <script> tag, or imported. In a browser environment, it is used mainly for enhancing the user interaction with the Web site. This is possible due to Web APIs such as the DOM API, which allows JS to manipulate HTML and CSS, the Geolocation API, which accesses geolocation information, the Canvas and WebGL APIs, which enable animated 2D and 3D graphics, Audio and VIdeo APIs, which allow multimedia resources to be displayed, or even record a web camera feed, among others.

With the increasing growth of the Internet, specially dynamic Web sites and Web applications, new technologies were created and are being developed to improve the developer experience. Some of these include:

- TypeScript

  TypeScript is a typed superset of JavaScript, originally developed by Microsoft, which compiles to plain JavaScript. The key feature TypeScript brings to JavaScript is a type system. JavaScript is a dynamically typed language, meaning a single variable can contain a string, a number, a function, or any other data type. What TypeScript does it define what a given variable can contain. A summary of TypeScript features is described below:

  - Static typing: Types can be added to variables, functions, properties, etc. This helps the compiler and warns about potential errors in code, before even running the script or application. Types also help when using libraries and frameworks, as they let developers know exactly what type of data APIs expect.

- Type Inference: Implicit typing performed by TypeScript itself, so that developers don't need to provide types where the compiler can find them on its own.
- Autocomplete tooling support: Autocomplete tools like Intellisense and most IDE's builtin autocomplete tools provide hints and tips as code is written. Due to the static types, they are able to offer even more predictive assistance than usual.
- Syntax: Although the recent version of ECMAScript (2019) includes many of the features that have been present in TypeScript from earlier stages, some are still missing, such as, abstract classes and access modifiers.

These features result in a better experience for developers and are the reason why its popularity has been growing in later years.

- Sass

  Sass is a CSS preprocessor, which adds special features such as variables, inheritance, nested rules and mixins into regular CSS. The aim is to make the coding process simple and more efficient, that can be easily maintained, thus reducing the amount of CSS required otherwise.

3.3.1.2 Libraries and Frameworks

There are many libraries and frameworks for front-end development. A framework can be defined as a large number of components which support the development of web applications. These components aim to help developers by reducing the overhead associated with common activities performed in web development, such as, routing, API interaction and session management. They are usually used for developing dynamic web applications. The most used web frameworks for new projects are, currently, React, Angular and VueJS.

A library can be defined as a lightweight version of a framework, as it doesn't offer the same range of capabilities. For this reason, they are more flexible and hand over more control to the developer. Usually, it's necessary to include other libraries as the complexity of the web application increases, such as, state management and routing libraries. The most used web framework for new projects is, currently, ReactJS.

Nowadays, React is the most popular among all frameworks and libraries, which results in a bigger and better ecosystem around it, with multiple solutions.

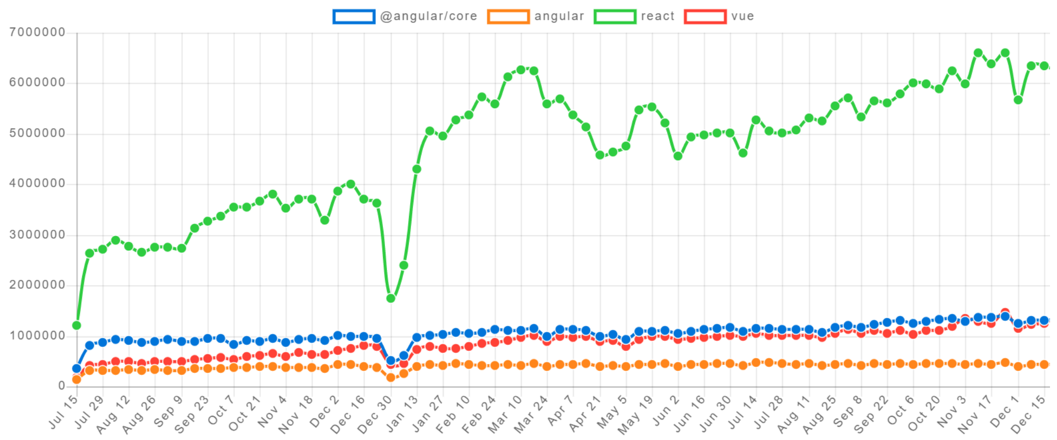


Figure 3.10: npm download stats for React, Angular, AngularJS and Vue[4]

There are multiple reasons for this popularity and why it's a great solution:

- Virtual DOM

  The Document Object Model is an interface for HTML documents, which represents the page so that programs or scripts can change its structure, style and content. The document is treated as a tree structure where each branch ends in a node, and each node contains objects.

  Updating the DOM is often considered slow, and most of the JavaScript frameworks update the whole DOM, which makes it slower. In most cases, this is unnecessary since only part of the DOM should be updated.

  To solve this problem, React relies on the concept of virtual DOM, where a virtual representation of the DOM is kept in memory and synced with the "real" DOM. This process is called reconciliation and it's current implementation is named React Fiber. The reconciliation mechanism includes React elements, life-cycle methods and the render method, and the diffing algorithm applied to a component's children. The React Fiber engine is what allows manipulating the virtual DOM to be much faster than the "real" DOM. When an update is made, the whole virtual DOM is updated, and then React compares the differences between the updated virtual DOM with the pre-updated "real" DOM. This is the diffing process and its result is the actual update that will affect the "real" DOM.

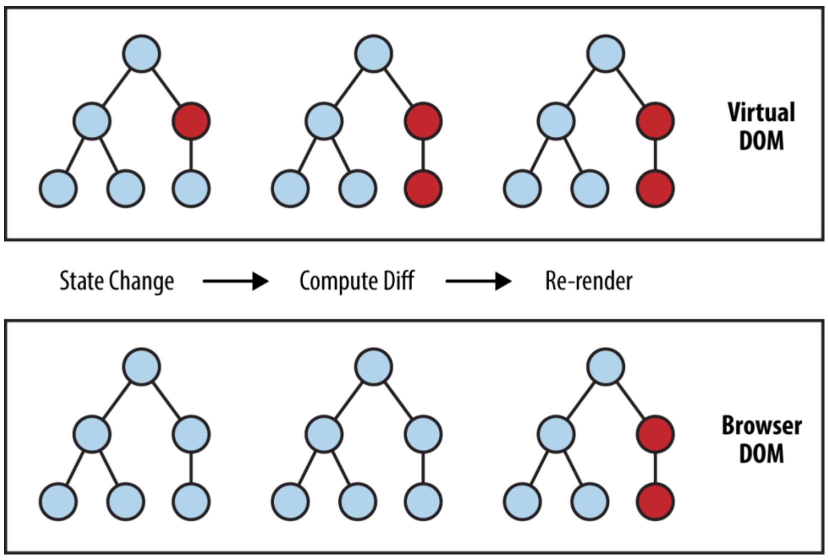


Figure 3.11: React Virtual DOM [5]

- Components

  A React application is made up of multiple components, where each component has its own logic and controls. Conceptually, they are similar to JavaScript functions since they accept inputs, called "props", and return React elements. There are two types of components: functional and class components. The first type is literally a JavaScript function whereas the later provides more features, such as life-cycle methods. They provide a way to split the UI into independent, reusable pieces which help to maintain the code when working on larger scale projects.

- One-way data binding

  As seen in the previous point, React applications are made up multiple components. These components are functional in nature: they accept inputs and return React Elements. One-way data binding means that data flows only one way.

  The view is a result of the application state. State can only change when actions are triggered. When actions are triggered, the state is updated. A component owns its state. Any data affected by this state can only affect the component's children, and never its parent, or its siblings, or any other component.

  There are some key advantages to this approach:

  - More control over data, resulting in a less error prone application
  - Better debugging, as the data and its origin are well know

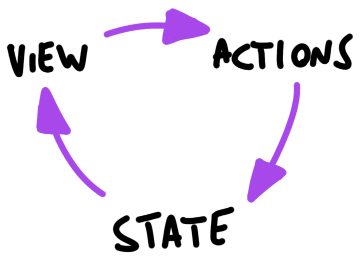


Figure 3.12: React one way data binding [6]

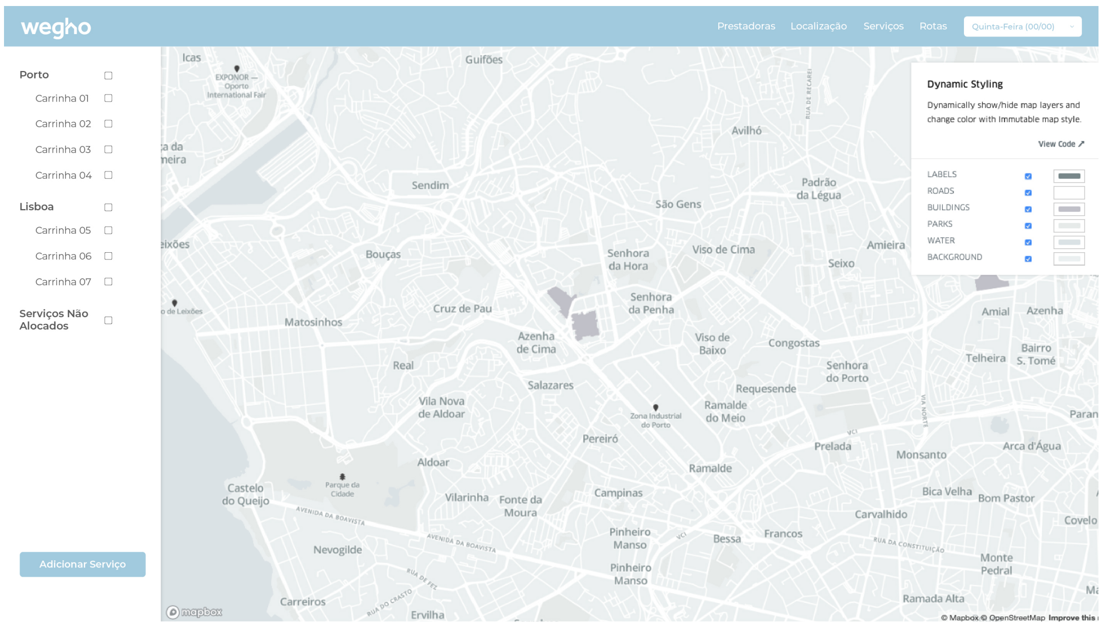
A commonly used library in conjunction with React is Redux [68]. Redux is a "predictable state container for JavaScript apps" [69]. It helps writing web applications that "behave consistently, run in different environments (client, server, and native), and are easy to test".

3.3.2 User Interface

The user interface is the means that allows users and computers to interact. An UI can include screens, keyboards or a mouse, but in a web application context it provides a way for users to interact with the web application, usually through a web browser. The UI is often talked about in conjunction with user experience (UX), which may include the interface's aesthetics, response time and the content that is presented to the user.

For a map-centered web application, the user interface must focus on the map while providing a way for users to interact with it.

The first iteration of the user interface design was created by Wegho's former UI/UX designer, and is composed of a wide map, which clearly defines the focus of the web application, an header where different features are toggled on and off, and a sidebar for filtering vehicles and districts.



Dynamically show/hide map layers and change color with Immutable map style.

Figure 3.13: UI prototype

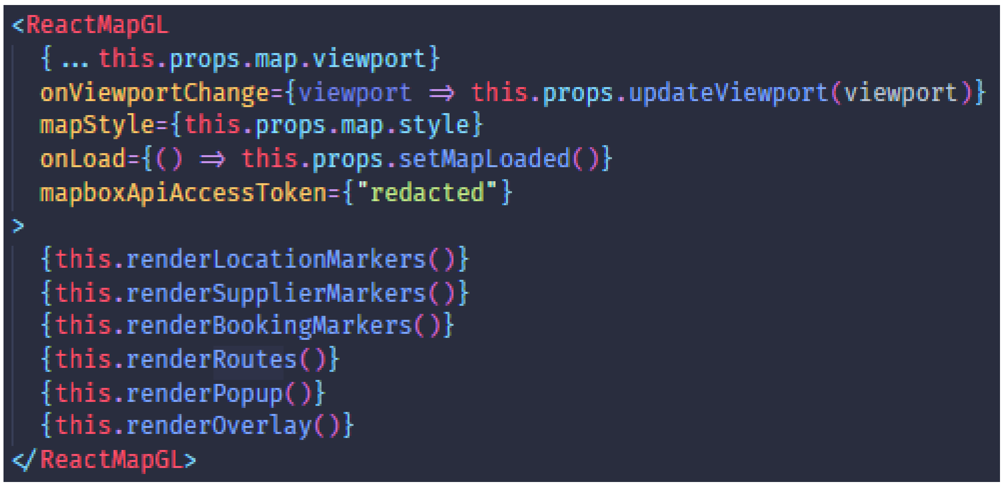
3.3.3 Implementation

In order to kick-start the web application development, a React boilerplate - Create React App [70] - was used. This boilerplate is officially supported by Facebook (the React maintainers) and its one of the most used to create single-page React applications. This boilerplate includes the initial setup for a npm project, React core dependencies and the react scripts package. The latter is the core of this boilerplate - it includes a linter (ESLint [71]), a transpiler (Babel [72]), a testing library (Jest [73]), a bundler (Webpack [74]) and other tools.

3.3.3.1 Map

The map is provided by Mapbox [75] and integrated with the React web application through React-Map-Gl [76], a component library developed by Uber. The pricing structure of Mapbox and its easy integration with React supported the decision to use this stack, instead of Google Maps or other map provider.

The ReactMapGL component provided by the React-Map-Gl library was used to display the map. This component takes several props: style, width, height, latitude, longitude, zoom and a onViewportChange function. The style prop is for setting up custom styles, which can be designed with the Mapbox Studio [77]. The width and height props are related to the map container, while the remaining props are used for the first render of the map, and are updated when the user navigates the map by the onViewportChange function.



```
<ReactMapGL
  {... this.props.map.viewport}
  onViewportChange={viewport => this.props.updateViewport(viewport)}
  mapStyle={this.props.map.style}
  onLoad={() => this.props.setMapLoaded()}
  mapboxApiAccessToken={"redacted"}
>
  {this.renderLocationMarkers()}
  {this.renderSupplierMarkers()}
  {this.renderBookingMarkers()}
  {this.renderRoutes()}
  {this.renderPopup()}
  {this.renderOverlay()}
</ReactMapGL>
```

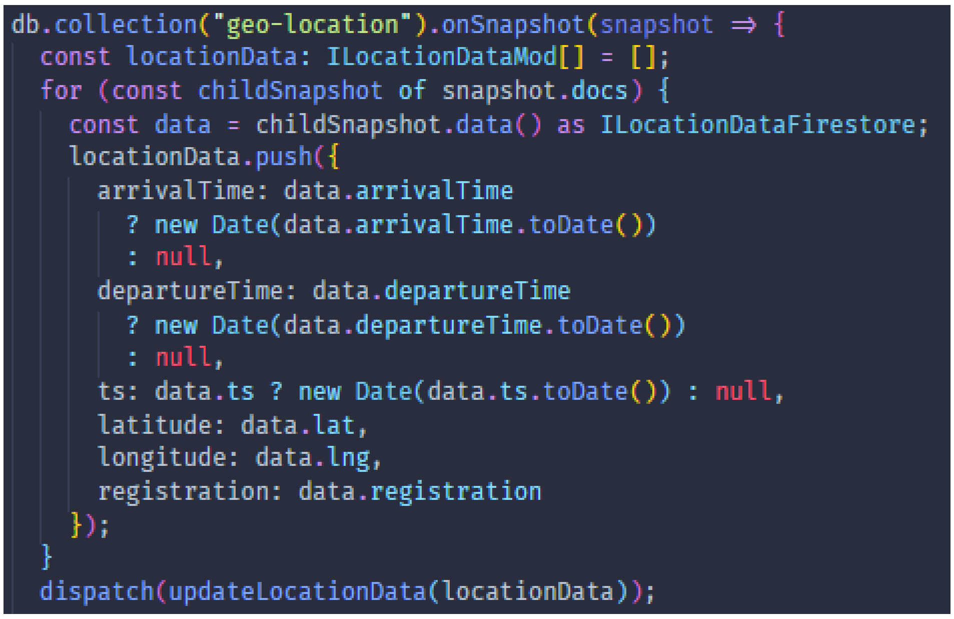
Figure 3.14: Map Component

The Mapbox API access token is required to access the Mapbox API. Since this is a client-side application running in a web-browser, the client will always have access to the token. To avoid unauthorized requests to the Mapbox API, the token is configured to only allow requests from the domain where the web application is hosted.

3.3.3.2 Location

As defined in chapter 3.1.3, location data is stored in Google Firestore. In order to interact with the database from the web application, the Firebase JavaScript SDK [60] was used. This library can be used in multiple environments, namely the web browser, and provides an easy way to query Firestore databases.

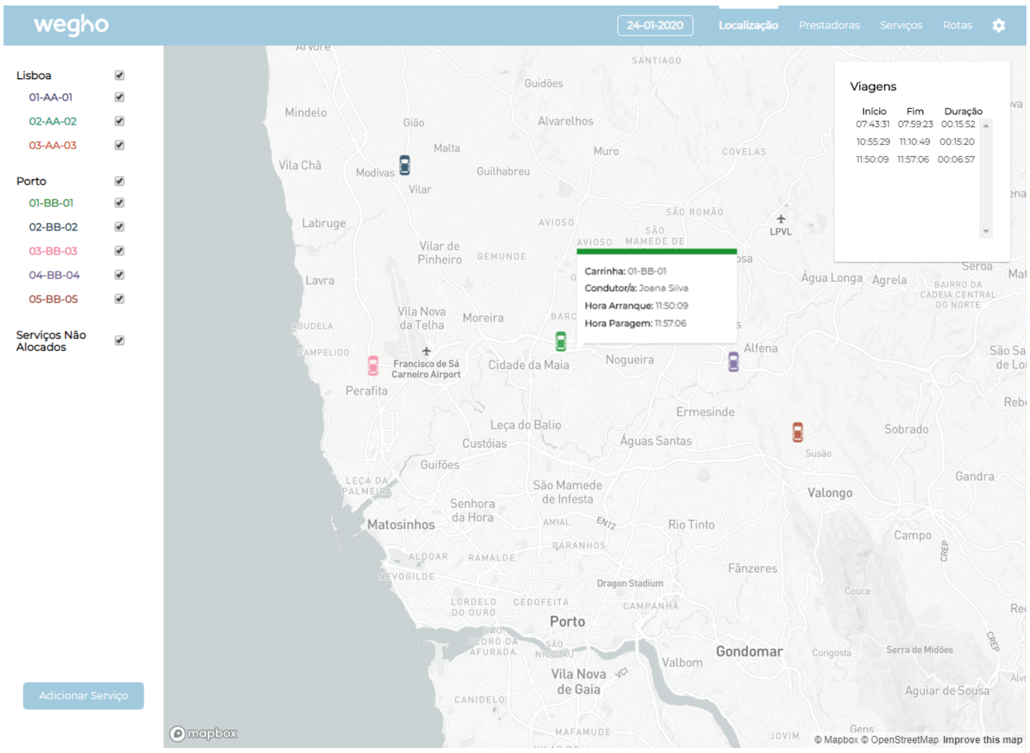
The *onSnapshot* method from the SDK was used to access the location data in real-time. This method "listens" to updates on documents from a given collection and triggers a callback function whenever there are changes. The callback function updates the location data in the application store by dispatching the *updateLocationData* action, which in turn updates the location markers in the map.



```
db.collection("geo-location").onSnapshot(snapshot => {
  const locationData: ILocationDataMod[] = [];
  for (const childSnapshot of snapshot.docs) {
    const data = childSnapshot.data() as ILocationDataFirestore;
    locationData.push({
      arrivalTime: data.arrivalTime
        ? new Date(data.arrivalTime.toDate())
        : null,
      departureTime: data.departureTime
        ? new Date(data.departureTime.toDate())
        : null,
      ts: data.ts ? new Date(data.ts.toDate()) : null,
      latitude: data.lat,
      longitude: data.lng,
      registration: data.registration
    });
  }
  dispatch(updateLocationData(locationData));
```

Figure 3.15: subscribeToLocationData function

The subscribeToLocationData function is called on page load since the vehicle location is shown by default. Simultaneously, data regarding bookings and suppliers is fetched from Wegho's operational database. This includes driver information, which is merged with vehicle location data in order to identify vehicle drivers and connect vehicles with booking.



Viagens

Início	Fim	Duração
07:43:31	07:59:23	00:15:52
10:55:29	11:10:49	00:15:20
11:50:09	11:57:06	00:06:57

Carrinha: 01-BB-01
Condutor/a: Joana Silva
Hora Arranque: 11:50:09
Hora Paragem: 11:57:06

Figure 3.16: Location feature

3.3.3.3 Suppliers

Suppliers data is stored in Wegho's operational database, which is accessible through an API, and contains information about all suppliers that provide services for Wegho. This data is used to draw markers in the map and provide general information about suppliers.

This endpoint accepts GET requests and its schema is described below.

Field	Type	Description
Date	Date	The date being queried

Table 3.8: Request schema for GetSuppliers

Field	Type	Description
SupplierID	number	Supplier unique identifier
SupplierName	string	Supplier name
HomeDistrictName	string	Supplier home district
HomeAddress	string	Supplier address
Latitude	number	Supplier address latitude
Longitude	number	Supplier address longitude
Registration	string	Supplier vehicle registration

Table 3.9: Response schema for GetSuppliers

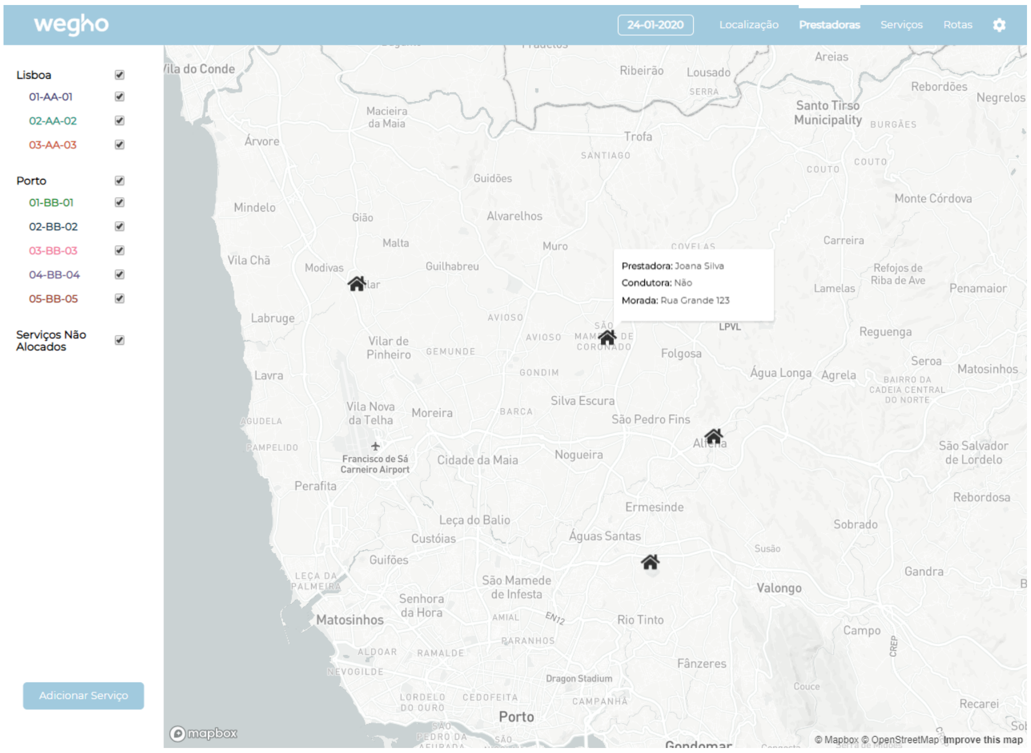


Figure 3.17: Suppliers feature

3.3.3.4 Bookings

Bookings data is also stored in Wegho's operational database, and is accessible through the same API. A booking is a service booked by a client. This data is used to draw markers in the map and provide general information about the service.

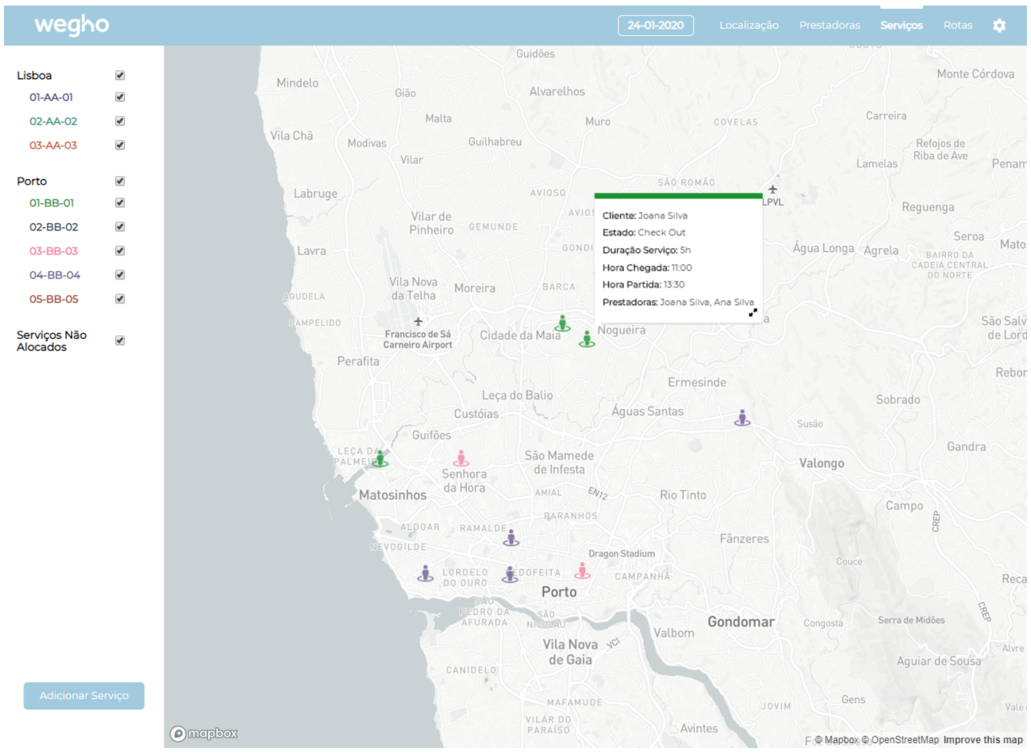
This endpoint accepts GET requests and its schema is described below.

Field	Type	Description
Date	Date	The date being queried

Table 3.10: Request schema for GetBookings

Field	Type	Description
BookingID	number	Booking unique identifier
StatusID	number	Booking status identifier
StatusDescription	string	Booking status description
Latitude	number	Booking address latitude
Longitude	number	Booking address longitude
DistrictName	string	Booking address district
Address	string	Booking address
Duration	number	Booking duration in hours
StartDate	date	Booking start date
EndDate	date	Booking end date
Notes	string	Booking notes
FirstName	string	Client first name
LastName	string	Client last name
PhoneNumber	string	Client phone number
LSuppliers	Suppliers	Suppliers assigned to the booking

Table 3.11: Response schema for GetBookings



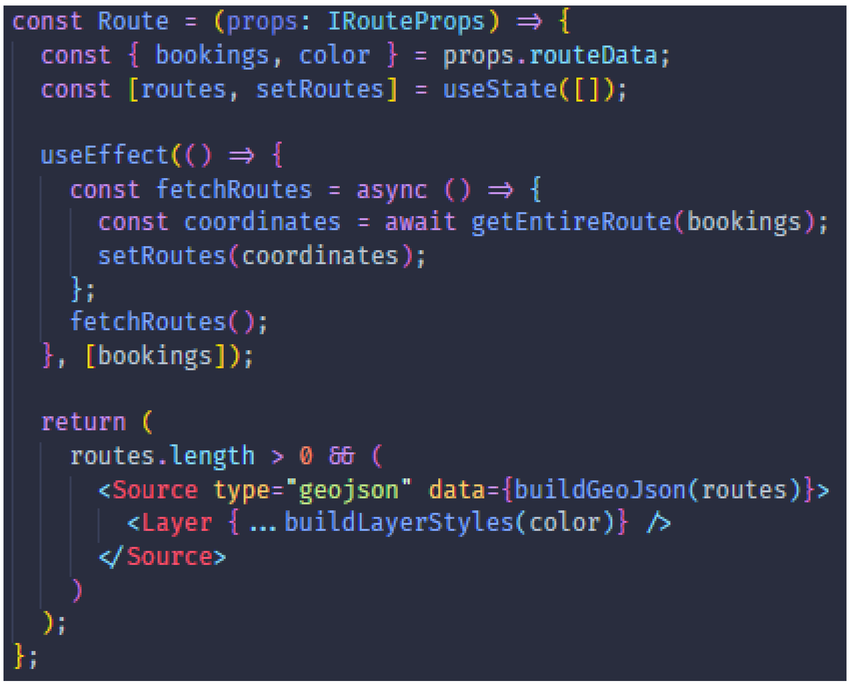
Cliente: Joana Silva
Estado: Check Out
Duração Serviço: 5h
Hora Chegada: 11:00
Hora Partida: 13:30
Prestadoras: Joana Silva, Ana Silva

Figure 3.18: Bookings feature

3.3.3.5 Routes

Routes are created for each vehicle, for a given day. Usually, routes start and end at a suppliers home address, since suppliers currently take vehicles home. All bookings associated with each vehicle are sorted by start time and added to a list of coordinates. Using Mapbox's Direction API [64], a route is created between all points and then drawn on the map.

The Route component relies on the Source and Layer components of React-Map-Gl to draw the routes effectively.



```
const Route = (props: IRouteProps) ⇒ {
  const { bookings, color } = props.routeData;
  const [routes, setRoutes] = useState([]);

  useEffect(() ⇒ {
    const fetchRoutes = async () ⇒ {
      const coordinates = await getEntireRoute(bookings);
      setRoutes(coordinates);
    };
    fetchRoutes();
  }, [bookings]);

  return (
    routes.length > 0 && (
      <Source type="geojson" data={buildGeoJson(routes)}>
        <Layer {...buildLayerStyles(color)} />
      </Source>
    )
  );
};
```

Figure 3.19: Route component

The Source component creates a map source from a geoJSON object. The geoJSON format is the stardard format for encoding geographic data structures [78]. This object is created from the route coordinates received by the Directions API call, which is executed in the useEffect hook. The Layer component adds a layer to the map and is responsible for styling. In this case, the layer is of type line, and will connect every point in the route.

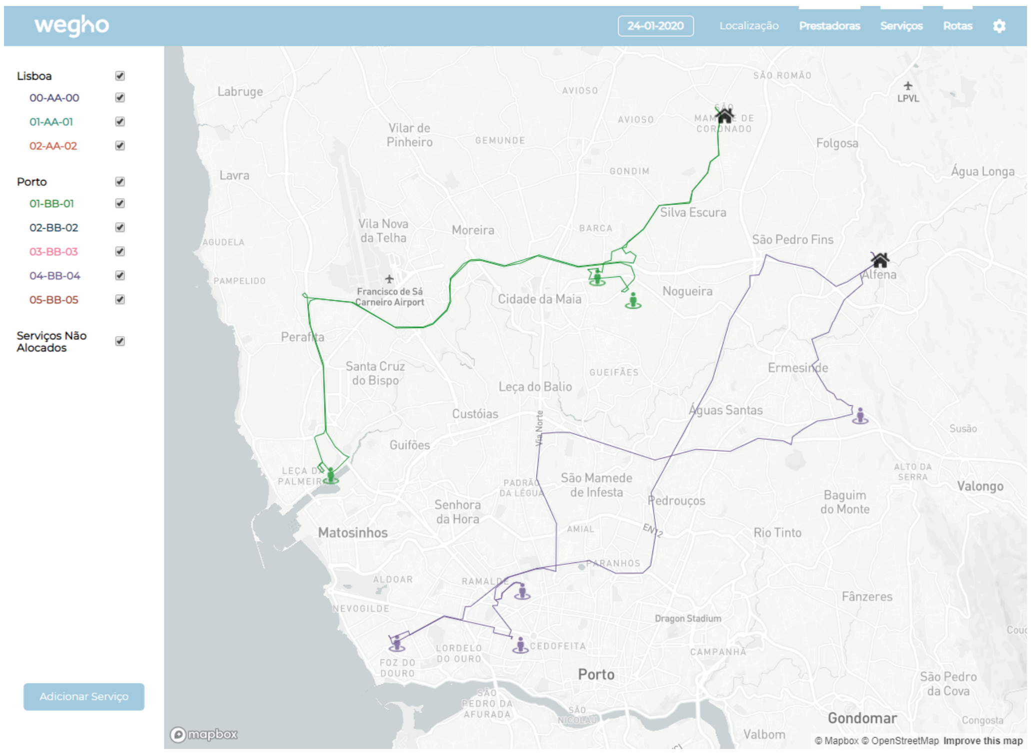


Figure 3.20: Routes feature

3.3.3.6 Timeline

The timeline feature allows users to navigate between past, current and future dates.

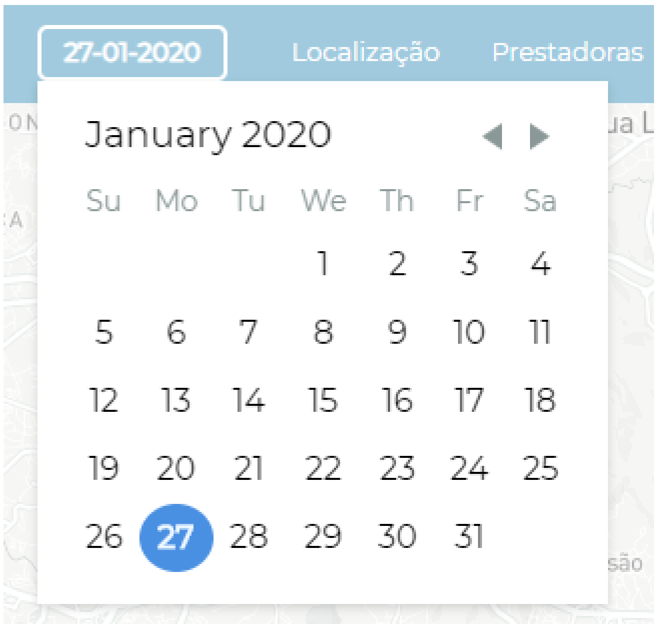


Figure 3.21: Timeline feature

When selecting a past date, bookings and suppliers data are updated to that particular date. Although location data is not be available for past dates (its not saved in the database), all trips are available for analysis.

In case of selecting a future date, only the bookings and suppliers features are available, since current location of the vehicles is not relevant, and its not possible to calculate routes while the schedule is not closed. The existing bookings for that day will show as unassigned, represented by a grey color, which helps the support team to build schedules.

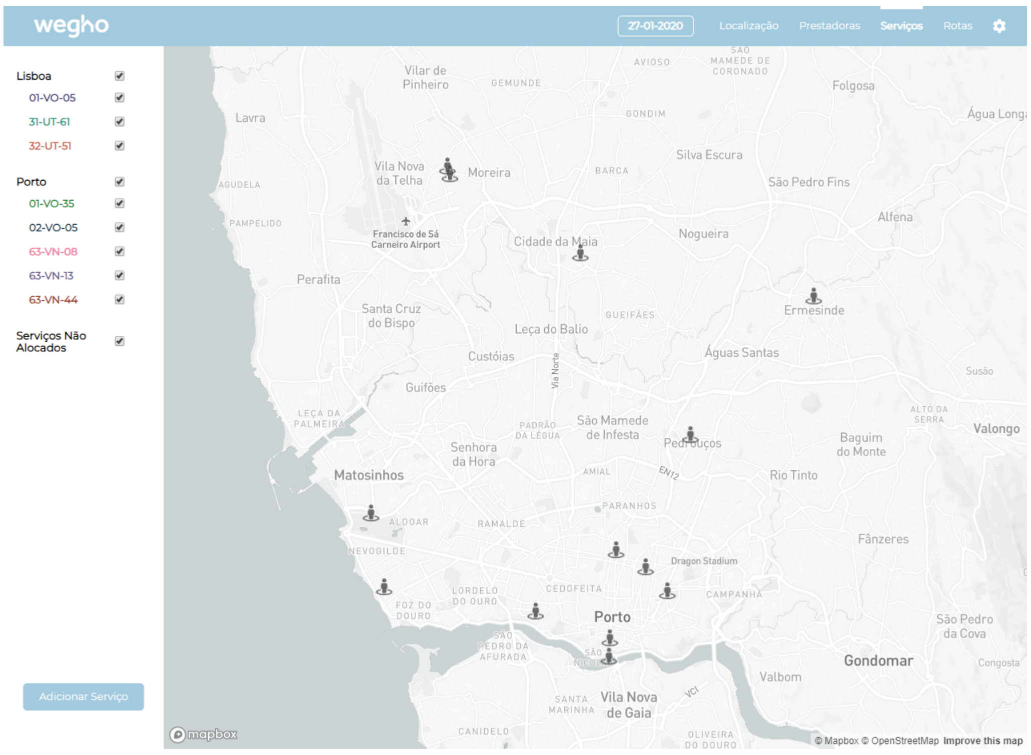


Figure 3.22: Unassigned bookings

For bookings that had not been created at that time, a "Add Booking" feature was added. This consists of a form where the user enters the costumer data and assigns a vehicle to that service.

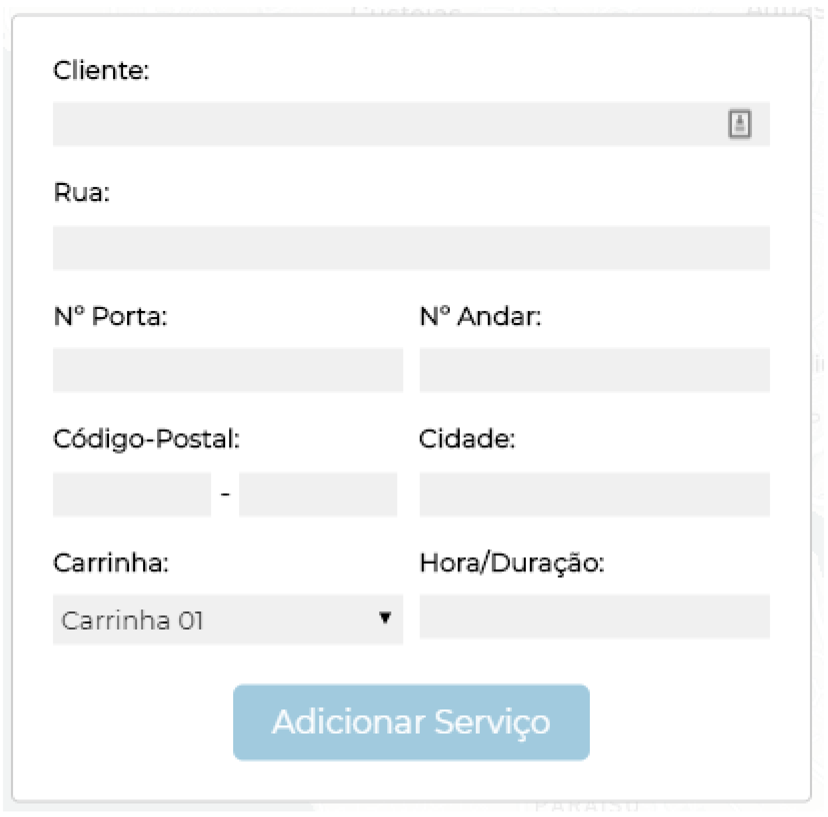


Figure 3.23: Add Booking feature

Chapter 4

Tests and Results

4.1 Data Ingestion Framework

The goal of the data ingestion framework is to ingest, process and store geolocation data. The developed solution was deployed on September, 2019 and was improved upon in the following weeks. It's now considered stable, as it doesn't produce any errors and the expected data matches the source data.

In order to test data consistency between CarTrack's data and the stored data, a test was developed to compare both data sources at a given time. This test queries, in parallel, the CarTrack's API endpoint *get_vehicle_detailed_event* which is used in the ingestion function 3.1.1 and the Firestore collection "geo-location", and retuns whether or not the data for each vehicle is the same.

This test was executed everytime the ingestion function was changed, for intervals between 1 hour and 1 day, to guarantee the accuracy of the data being visualized in the web application. This test allowed *sub-trips* to be found and corrected, and it currently results in 100% accuracy.

A different test was developed in order to assert the performance of the new web application against CarTrack's current web application. This test was expected to show a slight advantage for CarTrack's web application, since its likely hosted on the same infrastructure as their ingestion system and can take advantage of different communication systems and protocols that are not available for external applications. Suprisingly, the latency between the location data updates in the API and the corresponding web application view, was actually smaller on the new web application, overall. Due to the fact that CarTrack's web application is built on Adobe Flash, it was not possible to implement an automated test. Instead, a visual test with both web applications side-by-side allowed to draw some conclusions about the delay on the data updates. Altough a precise number could not be obtained (order of miliseconds), it was clear that the location markers were being updated faster in the new web application. This surprising result can be justified, most likely, by the use of old and deprecated technologies by CarTrack, such as Adobe Flash.

4.2 Web Application

4.2.1 UI/UX

The first user interface prototype was a good foundation for the web application, but the real test relies on the actual end-users to test it and provide feedback about their user experience. The user interface was improved after several iterations, until the results met the users expectations.

The first prototype of the web application had all features and the base styles that were worked upon, but lacked an adequate user experience.

Some changes are more noticable than others, but overall result in a better user experience. These changes include:

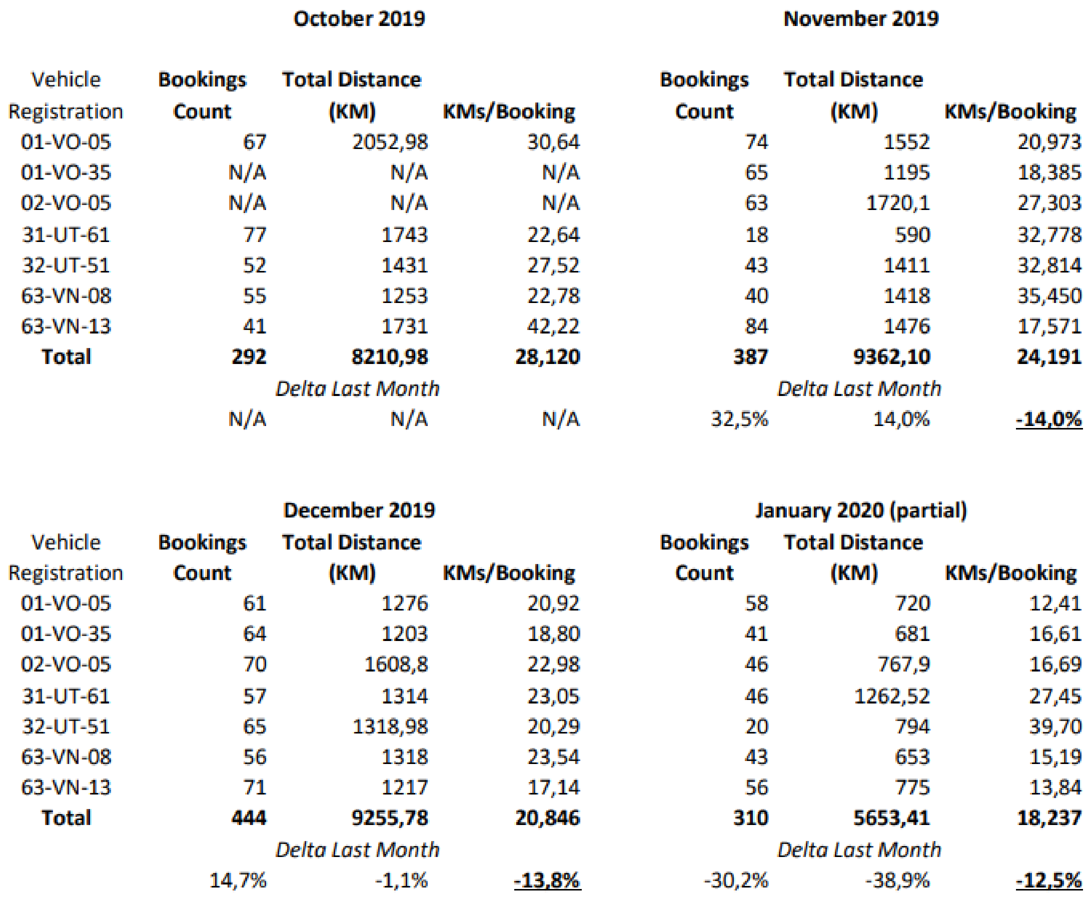
- Markers All markers were changed to icons that represent their meaning more clearly. They are also colored to match the vehicle, so its easier to identify what markers are connected.

- Vehicles list The vehicles list now shows the vehicles registration instead of a sequential numbering order. This helps identifying a vehicle when there's a need to check any type of information about it, or services and suppliers related to it. The vehicles registrations are also colored to match the markers.

4.3 Trips

The goal of the daily trips report is to improve efficiency in the schedulling and routing processes. It is also useful to account for non-work related trips, which are allowed but not unlimited.

The report has been generated daily since October 1st, 2019 and the results can be seen in the following table. These results are from reports generated between October 1st, 2019 and January 16th, 2020.



	October 2019			November 2019		
Vehicle Registration	Bookings Count	Total Distance (KM)	KMs/Booking	Bookings Count	Total Distance (KM)	KMs/Booking
01-VO-05	67	2052,98	30,64	74	1552	20,973
01-VO-35	N/A	N/A	N/A	65	1195	18,385
02-VO-05	N/A	N/A	N/A	63	1720,1	27,303
31-UT-61	77	1743	22,64	18	590	32,778
32-UT-51	52	1431	27,52	43	1411	32,814
63-VN-08	55	1253	22,78	40	1418	35,450
63-VN-13	41	1731	42,22	84	1476	17,571
Total	**292**	**8210,98**	**28,120**	**387**	**9362,10**	**24,191**
		Delta Last Month			*Delta Last Month*	
	N/A	N/A	N/A	32,5%	14,0%	**-14,0%**

	December 2019			January 2020 (partial)		
Vehicle Registration	Bookings Count	Total Distance (KM)	KMs/Booking	Bookings Count	Total Distance (KM)	KMs/Booking
01-VO-05	61	1276	20,92	58	720	12,41
01-VO-35	64	1203	18,80	41	681	16,61
02-VO-05	70	1608,8	22,98	46	767,9	16,69
31-UT-61	57	1314	23,05	46	1262,52	27,45
32-UT-51	65	1318,98	20,29	20	794	39,70
63-VN-08	56	1318	23,54	43	653	15,19
63-VN-13	71	1217	17,14	56	775	13,84
Total	**444**	**9255,78**	**20,846**	**310**	**5653,41**	**18,237**
		Delta Last Month			*Delta Last Month*	
	14,7%	-1,1%	**-13,8%**	-30,2%	-38,9%	**-12,5%**

Figure 4.1: Trips Results

To measure impact, the metric of choice was the average distance per booking, as it allowed to better understand how efficient the human-drawn routes are despite a variation of the number of bookings serviced. With an increase of over 30% from October 2019 to November 2019, mainly due to the addition of two extra teams, the operational team managed to only increase the total distance driven by 14%, lowering the average significantly.

The trend compares when comparing the most operationally similar months, November and December 2019 – an increase of bookings executed with an actual reduction in distance covered, thus resulting in a decrease of the main metric by 13.8%.

Delving into January 2020, with a partial sample of the month, nonetheless very relevant as the efficiency keeps growing with a likeness of another absolute growth of bookings provided by the same number of operational teams on the field.

Chapter 5

Conclusion

5.1 Main Contributions

The primary goal of the work presented in this dissertation was to develop a framework for ingesting and processing geolocation data in real-time. The work developed achieves this goal, by providing the foundation for new tools and solutions to be developed. This foundation was already used to develop a web application for a fleet management system that can be extended to even more features, and to feed a BI tool for automated report generation.

5.2 Future Work

Even though the work described in this dissertation is self-contained, in the sense that it may be readily used without further development, it does not constitute the ultimate solution for the problems that it addresses.

Some of the possible improvements include:

- Schedulling The schedulling feature allows new services to be added to the database and the map, but the current schedulling process at Wegho takes into account recurrent services, that are unassigned by default, and one-time services, so the current implementation can't replace the schedulling process. The solution would be to extend the bookings feature, so that unassigned services can be assigned to a vehicle and routes generated on-the-fly.

- Geofencing A geofence is a virtual geographic boundary around a physical location. This concept can be used to trigger a response, such as notifications or alerts, when a device enters or leaves a particular area. In this particular case, the devices are Wegho's vehicles. This can help in identifying scenarios where the drivers travel outside a certain zone that they are assigned to, or even provide early detection of car theft.

- Automated delay notifications The ability to automatically notify costumers in case of unexpected delays increases transparency and user satisfaction. By using the location data and bookings information, with a service such as Mapbox Directions, its possible to get an estimated time of arrival and display it on the costumer's mobile application.

References

[1] Panoply. Data warehouse architecture: Traditional vs. cloud. https://panoply.io/data-warehouse-guide/data-warehouse-architecture-traditional-vs-cloud/. [Online; accessed 27-June-2019].

[2] Panoply. Data mart vs. data warehouse. https://panoply.io/data-warehouse-guide/data-mart-vs-data-warehouse/. [Online; accessed 27-June-2019].

[3] Google. Google cloud functions. https://cloud.google.com/functions/docs/concepts/overview. [Online; accessed 17-January-2020].

[4] npm trends. @angular/core vs angular vs react vs vue. https://www.npmtrends.com/@angular/core-vs-angular-vs-react-vs-vue. [Online; accessed 24-January-2020].

[5] Bonnie Eisenman. Learning react native. https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html. [Online; accessed 24-January-2020].

[6] Flavio Copes. Unidirectional data flow in react. https://flaviocopes.com/react-unidirectional-data-flow/. [Online; accessed 24-January-2020].

[7] Martin Hilbert. The World 's Technological Capacity. *The World 's Technological Capacity*, 332(April):60–66, 2011.

[8] Adobe. Flash and the future of interactive content. https://theblog.adobe.com/adobe-flash-update/. [Online; accessed 17-January-2020].

[9] Google. Google maps reverse geocoding. https://developers.google.com/maps/documentation/geocoding/intro#ReverseGeocoding. [Online; accessed 24-January-2020].

[10] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. *Application Delivery Strategies*, page 4, 2001.

[11] Bernard Marr. Big data: The 5 vs everyone must know. https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know/, 2014. [Online; accessed 27-June-2019].

[12] Google. Google suite. https://gsuite.google.pt/. [Online; accessed 27-June-2019].

[13] Dropbox. Dropbox. https:/www.dropbox.com/. [Online; accessed 27-June-2019].

[14] Salesforce. Salesforce. https:/www.salesforce.com/. [Online; accessed 27-June-2019].

[15] GoToMeeting. Gotomeeting. https:/www.gotomeeting.com/. [Online; accessed 27-June-2019].

[16] Amazon. Aws elastic beanstalk. https://aws.amazon.com/elasticbeanstalk/. [Online; accessed 27-June-2019].

[17] Heroku. Heroku. https:/www.heroku.com/. [Online; accessed 27-June-2019].

[18] Google. Google cloud app engine. https://cloud.google.com/appengine/. [Online; accessed 27-June-2019].

[19] OpenShift. Openshift. https:/www.openshift.com/. [Online; accessed 27-June-2019].

[20] Digital Ocean. Digital ocean. https:/www.digitalocean.com/. [Online; accessed 27-June-2019].

[21] Linode. Linode. https:/www.linode.com/. [Online; accessed 27-June-2019].

[22] Amazon. Aws ec2. https://aws.amazon.com/ec2/. [Online; accessed 27-June-2019].

[23] Google. Google cloud compute engine. https://cloud.google.com/compute/. [Online; accessed 27-June-2019].

[24] Amazon. Aws redshift. https://aws.amazon.com/redshift/. [Online; accessed 27-June-2019].

[25] Google. Google cloud bigquery. https://cloud.google.com/bigquery/. [Online; accessed 27-June-2019].

[26] Microsoft. Microsoft azure sql data warehouse. hhttps://azure.microsoft.com/pt-pt/services/sql-data-warehouse/. [Online; accessed 27-June-2019].

[27] Stan Zdonik, Michael Stonebraker, Mitch Cherniack, and Magdalena Balazinska. The Aurora and Medusa projects. *Data Engineering*, 51:1–8, 2003.

[28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[29] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *2010 IEEE International Conference on Data Mining Workshops*, pages 170–177. IEEE, 2010.

[30] Apache Foundation. Apache storm. https://storm.apache.org/. [Online; accessed 27-June-2019].

[31] Apache Foundation. Apache samza. https://samza.apache.org/. [Online; accessed 27-June-2019].

[32] Apache Foundation. Apache spark. https://spark.apache.org/. [Online; accessed 27-June-2019].

[33] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu, and Zheng Zhang. TimeStream: Reliable Stream Computation in the Cloud. *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 1–14, 2013.

[34] Apache Foundation. Apache kafka. https://kafka.apache.org/. [Online; accessed 27-June-2019].

[35] Apache Foundation. Apache druid. https://druid.apache.org/. [Online; accessed 27-June-2019].

[36] Apache Foundation. Apache hadoop. https://hadoop.apache.org/. [Online; accessed 27-June-2019].

[37] Amazon. Aws emr. https://aws.amazon.com/emr/. [Online; accessed 27-June-2019].

[38] Google. Google cloud dataproc. https://cloud.google.com/dataproc/. [Online; accessed 27-June-2019].

[39] Microsoft. Microsoft azure hdinsight. https://azure.microsoft.com/pt-pt/services/hdinsight/. [Online; accessed 27-June-2019].

[40] Amazon. Aws msk. https://aws.amazon.com/msk/. [Online; accessed 27-June-2019].

[41] Confluent. Confluent cloud. https://www.confluent.io/confluent-cloud/. [Online; accessed 27-June-2019].

[42] Pieter Hintjens. *ZeroMQ: messaging for many applications.* " O'Reilly Media, Inc.", 2013.

[43] Pivotal. Rabbitmq. https://www.rabbitmq.com/. [Online; accessed 27-June-2019].

[44] Amazon. Aws kinesis. https://aws.amazon.com/kinesis/. [Online; accessed 27-June-2019].

[45] Google. Google cloud pub/sub. https://cloud.google.com/pubsub/. [Online; accessed 27-June-2019].

[46] Microsoft. Microsoft azure event hubs. https://azure.microsoft.com/pt-pt/services/event-hubs/. [Online; accessed 27-June-2019].

[47] American Public Transportation Association. Comments to access board docket number 2007. https://web.archive.org/web/20101120113642/https://www.apta.com/gap/fedreg/documents/apta_comments_access_board_bus_2009.pdf. [Online; accessed 17-January-2020].

[48] Microsoft. Microsoft power bi. https://powerbi.microsoft.com/en-us/. [Online; accessed 27-June-2019].

[49] SAP. Sap lumira. https://saplumira.com/. [Online; accessed 27-June-2019].

[50] SAS. Sas visual analytics. https://www.sas.com/pt_pt/software/visual-analytics.html. [Online; accessed 27-June-2019].

[51] IBM. Ibm watson analytics. https://www.ibm.com/watson-analytics. [Online; accessed 27-June-2019].

[52] Tableau. Tableau software. https://www.tableau.com/. [Online; accessed 27-June-2019].

[53] Qlik. Qlik. https://www.qlik.com/pt-br. [Online; accessed 27-June-2019].

[54] TIBCO. Tibco. https://www.tibco.com/data-visualization. [Online; accessed 27-June-2019].

[55] d3. d3js. https://d3js.org/. [Online; accessed 27-June-2019].

[56] Fusion Charts. Fusion charts xt. https://www.fusioncharts.com/. [Online; accessed 27-June-2019].

[57] Google. Google scheduler and pub/sub. https://cloud.google.com/scheduler/docs/tut-pub-sub. [Online; accessed 17-January-2020].

[58] Strongloop. strong-soap. https://github.com/strongloop/strong-soap. [Online; accessed 17-January-2020].

[59] Google. Key management service. https://cloud.google.com/kms/. [Online; accessed 17-January-2020].

[60] Google. Firebase sdk. https://firebase.google.com/docs/admin/setup?hl=en. [Online; accessed 17-January-2020].

[61] Google. Google cloud functions cache. https://cloud.google.com/functions/docs/bestpractices/tips. [Online; accessed 17-January-2020].

[62] PostgreSQL. Postgresql notify. https://www.postgresql.org/docs/current/sql-notify.html. [Online; accessed 17-January-2020].

[63] Google. Mongodb change streams. https://docs.mongodb.com/manual/changeStreams/. [Online; accessed 17-January-2020].

[64] Mapbox. Mapbox directions. https://docs.mapbox.com/help/how-mapbox-works/directions/. [Online; accessed 24-January-2020].

[65] Facebook. React. https://reactjs.org/. [Online; accessed 24-January-2020].

[66] Google. Angular. https://angularjs.org/. [Online; accessed 24-January-2020].

[67] Evan You. Vue. https://vuejs.org/. [Online; accessed 24-January-2020].

[68] Dan Abramov. Redux. https://redux.js.org/. [Online; accessed 24-January-2020].

[69] Dan Abramov. Getting started with redux. https://redux.js.org/introduction/getting-started/. [Online; accessed 24-January-2020].

[70] Facebook. Create react app. https://create-react-app.dev/docs/getting-started. [Online; accessed 24-January-2020].

[71] JS Foundation. Eslint. https://eslint.org/. [Online; accessed 24-January-2020].

[72] Babel. Babel. https://babeljs.io/. [Online; accessed 24-January-2020].

[73] Facebook. Jest. https://jestjs.io/. [Online; accessed 24-January-2020].

[74] webpack. webpack. https://webpack.js.org/. [Online; accessed 24-January-2020].

[75] Mapbox. Mapbox maps. https://www.mapbox.com/maps/. [Online; accessed 24-January-2020].

[76] Uber. React-map-gl. https://uber.github.io/react-map-gl/#/. [Online; accessed 24-January-2020].

[77] Mapbox. Mapbox studio. https://www.mapbox.com/mapbox-studio/. [Online; accessed 24-January-2020].

[78] Internet Engineering Task Force (IETF). The geojson format. https://tools.ietf.org/html/rfc7946. [Online; accessed 24-January-2020].