Nil Monfort Villa

Development of a taxi planning system

FINAL PROJECT

directed by Jordi Duch Gavaldà

Degree in Computer Engineering

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2024

Resum.

El projecte s'ha desenvolupat en resposta a la necessitat d'automatitzar les funcions dels agents de tràfic en les empreses de transport, particularment dins de la indústria del taxi. La tasca d'aquests agents requereix un alt consum de recursos i, a més, aquests professionals són difícils de reclutar. Això va motivar la creació d'un sistema de planificació per a taxis que, basant-se en determinats serveis i taxistes disponibles, genera rutes òptimes ajustades a configuracions específiques.

Encara que els agents poden conèixer les rutes i les congestions en zones específiques, aquesta eina proporciona un rendiment instantani amb una extensa gamma de paràmetres i configuracions que serien impossibles de gestionar manualment en un temps raonable. El projecte s'ha implementat utilitzant Python per al backend i Typescript per al frontend, amb el framework Vue.

Resumen.

Este proyecto se ha desarrollado en respuesta a la necesidad de automatizar las funciones de los agentes de tráfico en las empresas de transporte, especialmente dentro de la industria del taxi. La labor de estos agentes requiere un alto consumo de recursos y, además, estos profesionales son difíciles de reclutar. Esto motivó la creación de un sistema de planificación para taxis que, basándose en determinados servicios y taxistas disponibles, genera rutas óptimas ajustadas a configuraciones específicas.

Aunque los agentes pueden conocer las rutas y las congestiones en zonas específicas, esta herramienta proporciona un rendimiento instantáneo con una extensa gama de parámetros y configuraciones que serían imposibles de gestionar manualmente en un tiempo razonable. El proyecto se ha implementado utilizando Python para el backend y Typescript para el frontend, con el framework Vue.

Abstract.

This project was developed in response to the need to automate the duties of traffic agents in transportation companies, particularly within the taxi industry. The role of these agents is highly resource-intensive and these professionals are hard to recruit. This led to the creation of a taxi scheduling system that, based on specific services and available taxi drivers, generates optimal routes tailored to specific configurations.

Although agents might be aware of the routes and congestion in certain areas, this tool provides instant performance with a wide range of parameters and configurations that would be impossible to manage manually in a reasonable time. The project was implemented using Python for the backend and Typescript for the frontend, with the Vue framework.

Acknowledgements

I would like to express my deep gratitude for the support received during these years of career to my family, my partner and, especially, to the best friends and colleagues with whom I have developed this project. As the saying goes: "If you want to go fast, go alone; but if you want to go far, go with you." This project is a clear example of this.

Secondly, I would like to thank my director, Jordi Duch, for guiding me and solving my doubts in the best possible way.

Finally, I am deeply grateful to my former boss, Alejandro Bisbal, for giving me the opportunity to carry out this project and for all the great learning I have acquired thanks to him.

Index

Index of figures

1 Introduction

1.1 Context

The project had its beginnings in early March 2023. At that time, a friend from university, Carlos Martínez, joined Avantcab, a private transfer company in Mallorca, as a full stack developer. During this period, the director of the company, Alejandro, proposed the creation of a start-up with the aim of developing a private software system. This initiative would not only save the costs associated with the use of third-party software, which was particularly high, but would also solve the limitations of an old and inefficient system, which required up to two hours a day just to plan the next day's services.

Alejandro's vision also includes marketing this software to other companies that were in similar circumstances. Due to the magnitude of the project, Carlos, recognizing the need to have a team, requested my collaboration as well as that of Roger Massana, another student at the Rovira i Virgili University, with deep knowledge in full stack technologies.

Once the team was formed, the responsibilities were distributed: Carlos and Roger assumed the development of the backend using Spring Boot and the implementation in AWS[1], while the three of us collaborated in the development of the frontend. Personally, I conducted a study in machine learning to develop a flight delay predictor and route planning algorithm.

The delay predictor was essential, as it provides crucial data on flight arrival patterns, thus allowing for more accurate and efficient route planning. Despite the depth of this study, this will be mentioned briefly as it is part of the integral system of the planning algorithm, but the report will focus on the development of the scheduling algorithm.

This algorithm, together with the reservation manager, make up key elements of the application. Compared to the previous system, Logisplan, which was remarkably slow and cumbersome in terms of execution. Besides, it offered many unnecessary features and some were missing, according to Alejandro.

The new system to be developed promised not only time optimization but also a substantial reduction in operational and personnel costs. The ability to efficiently manage a single person's traffic rather than a full team, exemplifies the significant logistical and economic advantages of this new solution.

With this new personalized planning system and integrated with the reservation system, it is expected to achieve a significant improvement in management and cost

[1] Amazon Web Services

reduction, which is why we have started the development of the application that will be described below.

1.2 Description

The development of the algorithm for scheduling services has been carried out using the Python programming language, chosen for several key reasons. First, Python offers an extensive library of resources that facilitate the handling of data efficiently, highlighting libraries such as Pandas and NumPy. In addition, this language had previously been used in the development of the flight delay predictor, which provided a solid foundation and familiarity that simplified the continuity of work on the backend.

Python is renowned for its clear and readable syntax, which facilitates programming and reduces the time needed for code development and maintenance. This translates into greater efficiency during the testing and debugging phases of the algorithms. Its interpreted nature, along with scripting capacity, allows for rapid adjustments and iterations in development, essential in a dynamic start-up environment.

On the other hand, Flask, a Python framework, has been incorporated for the creation of APIs[2] that serve as a bridge between the backend and the frontend of the system. This strategic choice facilitates a more agile and secure integration within the general architecture of the application.

As for the development of the frontend, we have opted for TypeScript instead of JavaScript. The decision to use TypeScript is due to its ability to tip code explicitly, thus increasing the robustness of the software and reducing the likelihood of errors, especially in future application scalability scenarios.

In the field of frontend technologies, Vue.js has been selected, a JavaScript library that stands out for its simplicity and efficiency compared to alternatives such as React. In addition, it has made use of PrimeVue, a component library for Vue.js that offers a wide range of prefabricated components. This choice not only speeds up development, but also ensures engaging and professional visual consistency throughout the user interface.

[2] Aplication Programming Interface

1.3 Motivation

The motivations for doing this project have been diverse, and I will discuss them briefly.

First of all, having the opportunity to work in a job more focused on data science fascinated me, since I had worked for 8 months in web development. This experience gave me great learning, but it was clear to me that I would not want to dedicate myself to web development in the future. It was a monotonous task and I had the feeling that, within years, creating forms and other interfaces would be completely covered by AI, [3]like ChatGPT. On the other hand, having the opportunity to work in data science would allow me to find more attractive jobs in this field later on.

Secondly, it was an opportunity to put into practice my knowledge in Machine Learning that I had learned in different courses (Udemy and Stanford University) and the book "Hands-On Machine Learning" in flight prediction. I was fascinated by the idea of solving problems where only I, a search engine, and ChatGPT were responsible for transforming a blank IDE [4]into a final project. The ability to solve problems and one's tenacity were key in this process.

Finally, the idea of spending time with my friends as coworkers was a luxury that many people would dream of living.

1.4 Objectives

The main objective of this research work has been the development of an algorithm not only functional, but also efficient and effective, with a low computational cost. This algorithm is sought to be highly customizable and modular, allowing it to adapt to the specific needs of each client. In addition, the goal is to create a versatile system that can be marketed and adapted by other users outside the company.

In addition to technical development, this project has also had as secondary objectives to promote teamwork and enhance transversal skills or soft skills. Through collaboration with co-workers and joint problem solving, we have sought to improve skills such as communication, time management and the ability to work under pressure.

Another objective has been to deepen and expand my knowledge in data science and algorithms, areas that I find especially attractive and in which I want to continue progressing. This project has allowed me to apply the concepts learned in courses and books to real situations, thus consolidating my knowledge and improving my technical skills.

[3] Artificial intelligence

[4] Integrated Development Envoirment

2 Tools and technologies used

2.1 Frontend development tools

2.1.1 Vue

Vue is a progressive JavaScript framework used to build single-page user interfaces and applications. It is known for its ease of integration into projects with other existing libraries or projects, offering a lightweight and flexible structure. Vue facilitates the creation of complex applications using reusable and reactive components that efficiently manage the state and logic of the application.

2.1.2 PrimeVue

PrimeVue is a suite of UI components [5]for Vue that provides a wide variety of rich and dynamic elements, designed to integrate seamlessly into Vue applications. It offers components such as data tables, dialogs, menus and other interface elements, all with customizable styles and optimized for high performance and accessibility.

2.1.3 Typescript

TypeScript is a programming language that is a superset of JavaScript, which adds optional static types. This allows developers to write cleaner and easier to manage code, making it easier to detect errors before execution. TypeScript is especially useful in large-scale projects, like this one, where it helps keep code organized and less error-prone.

2.1.4 HTML

HTML [6] is the standard language used to create and design web pages. It serves as the skeleton of all web pages, structuring the content and multimedia elements of the user interface.

2.1.5 CSS

CSS [7] is the language used to define the visual appearance and design of web pages. It allows developers to separate design from content, facilitating aesthetic presentation without altering the underlying functionality.

---

[5] User Interface

[6] HyperText Markup Language

[7] Cascading Style Sheets

2.2 Backend development tools

2.2.1 Python

Python is a high-level, interpreted, general-purpose programming language. It is appreciated for its syntactic simplicity and its ability to make code easier to read. In addition, Python is widely used in web development, data science, automation, and many other fields, thanks to its versatility and extensive library of third-party tools and modules.

2.2.2 Flask

Flask is a microframework for web applications written in Python, which stands out for being lightweight and modular, allowing developers to add only the functionality they really need for their project. This makes it ideal for applications of all sizes, from small services to complex web applications.

2.2.3 Numpy and Pandas

Numpy is a library for Python that provides support for large arrays and multidimensional arrays, along with a collection of high-throughput mathematical functions for operating on these arrays. Pandas, on the other hand, is a library that offers data structures and operations to manipulate numerical tables and time series. It is indispensable in data science for data analysis and manipulation.

2.2.4 Open Route Service

Open Route Service is a route calculation API based on data from OpenStreetMap. It allows developers to integrate route planning, geocoding, and other geospatial services functionalities within their applications.

2.2.5 Spring Boot

Spring Boot is a framework for creating Java-based web applications and services. It simplifies the process of setting up and deploying Spring apps by providing a quick and easy way to start and run Spring-based applications with minimal settings.

2.2.6 Docker

Docker is a container platform that allows developers to package applications and their dependencies into standardized containers. This facilitates deployment and portability between different systems and infrastructures, ensuring that the application will work the same in all environments.

2.3 Development environments

2.3.1 PyCharm

PyCharm is an integrated development environment designed specifically for Python. It offers advanced tools for coding, debugging, testing and data visualization, facilitating the development of Python applications. PyCharm includes support for web frameworks such as Django, Flask and Pyramid, as well as for database management and other Python development technologies.

2.3.2 IntelliJ IDEA

IntelliJ IDEA is an advanced IDE developed by JetBrains, designed for efficient coding in various languages, including Java, Scala, and Kotlin, among others. It provides sophisticated tools for code management, debugging and quality analysis, and offers support for integration with multiple version management systems and web application frameworks.

2.3.3 Visual Studio Code (VSCode)

Visual Studio Code is a lightweight yet powerful source code editor that supports a wide variety of programming languages including Python, JavaScript, Java, C#, among others. VSCode is known for its ability to adapt to various development needs thanks to its extensive library of extensions, allowing everything from Git repository management to Docker container implementation.

2.3.4 Postman

Postman is a collaborative platform for API development that is used to verify, build, and document APIs. It makes it easy to create HTTP requests[8], configure test environments, and run request collections for automated testing. Postman provides an intuitive user interface to configure requests, view responses and debug, as well as organize and share resources in development teams.

[8] Hypertext Transfer Protocol

3 Requirements

In the context of this project, requirements are statements that specifically describe the functionalities, features or constraints that the taxi planning algorithm must meet. These requirements facilitate communication between developers, clients and project managers, clearly defining expectations and success criteria.

3.1 Functional requirements

3.1.1 RF1. Real-time data reception:

The algorithm can receive and process data in real time from an endpoint that provides updated information on pending taxi services and the available schedules of taxi drivers. This includes the ability to handle volatile and constantly changing data to enable planning decisions based on the most current state of resources.

3.1.2 RF2. Customizable configuration of scheduling parameters:

The system allows users to customize scheduling parameters from the frontend, allowing flexible selection of the specific day of services (current, next, etc.), as well as other critical parameters such as estimated journey duration, service priorities, and driver preferences.

3.1.3 RF3. Generation of optimal routes:

Based on the data received and the specified configuration, the algorithm calculates and assigns the most efficient routes for each taxi driver, optimizing time and geographical coverage. This process includes planning algorithms that take into account factors such as expected traffic, areas of highest demand and the proximity of taxi drivers to customers.

3.1.4 RF4. Dynamic update in response to changes:

When input data changes, such as last-minute service cancellations or additions, the algorithm recalculates routes to ensure continuous efficiency. This rapid response capacity is very important to maintain an adaptive service in the face of changing urban dynamics.

3.1.5 RF5. User interface for route display:

The system provides a graphical interface where users can view assigned routes and details corresponding to each mapping, including estimates of times, distances, and current status of services. This interface should be intuitive and facilitate real-time monitoring of operations.

3.1.6 RF6. Resource optimization and availability calculation:

The algorithm must incorporate functions to calculate the optimal number of taxis needed at any time, based on predictive analysis of future demand. This includes the ability to forecast peaks in demand and proactively adjust resource distribution to minimize wait times and maximize customer satisfaction.

3.1.7 RF7. Calculation of the optimal number of taxis required:

The algorithm must be able to determine the minimum number of taxis required to cover a certain number of scheduled services, thus maximizing operational efficiency. This calculation will consider variables such as the total number of services, the geographical distribution of services, and the temporary availability of taxi drivers, allowing the adjustment of the taxi fleet to ensure optimal business performance.

3.1.8 RF8. Smart planning and grouping of services:

The algorithm must be able to create routes that combine multiple services effectively, such as picking up a passenger at the airport, dropping them off at a hotel, and then heading to another town to pick up a second passenger and take them to the airport. This functionality will seek to form routes that maximize the use of available resources (taxis) while minimizing travel time and distance traveled, contributing to cost optimization and improvement of operational efficiency.

3.2 Non-functional requirements

3.2.1 RNF1. Performance and scalability:

The algorithm must process the data efficiently, regardless of the increase in the number of taxis or services, ensuring fast response times and the ability to handle high workloads without degrading performance.

3.2.2 RNF2. Data security:

All data transmissions must be secure, using encryption and standard security protocols to protect sensitive information. This includes the implementation of security measures for the protection of personal data and the integrity of transactions.

3.2.3 RNF3. Adaptability to various devices:

The user interface must be responsive, working properly on mobile devices, tablets and desktops to ensure a consistent and accessible user experience from any platform.

3.2.4 RNF4. Usability:

The interface should be intuitive and easy to use for traffic operators, with minimal learning curve and support for efficient navigation and fast task solving.

3.2.5 RNF5. Reliability:

The system must be robust and able to recover quickly from failures or failures, while maintaining critical availability for continuous operations without significant interruptions.

3.2.6 RNF6. Compatibility and Integration:

The system must be compatible with different existing platforms and technologies, allowing easy integration with other systems used by the company, such as database management systems, external APIs, and other software tools.

3.2.7 RNF7. Maintainability:

Code and system structure should be designed in a way that allows for easy updates and maintenance, including scaling functionality and fixing bugs efficiently.

3.3 Use Case Diagram

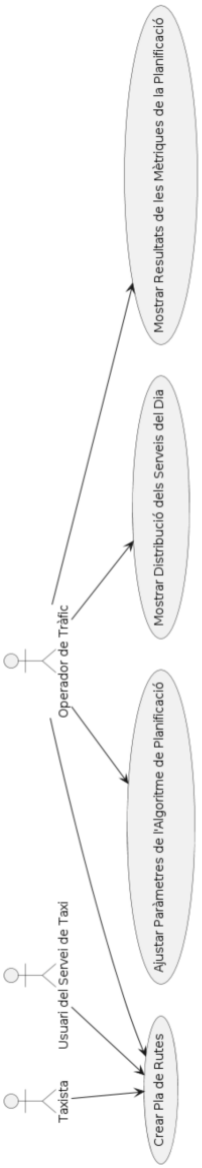

Figure 1 - Use Case Diagram.

3.4 Textual Use Cases

3.4.1 Cd'ú 01. Create Route Plan

Summary of functionality: Generate a daily route plan for taxi drivers based on the demands of the day's services.

Input Parameters: List of services of the day, includes locations and times requested.

Departure parameters: Detailed route plan for each taxi driver.

Actors: Traffic operator.

Precondition: Services of the day must be defined and available in the system.

Postcondition: A route plan is available to be reviewed or adjusted by the traffic operator.

Main normal process:

1. The system collects service demands for the specific day.
2. The algorithm analyzes locations and times to optimize routes.
3. The system assigns services to taxi drivers based on proximity and availability.
4. The system generates a visual report of the routes for review.

Process alternatives and exceptions:

1st. If there are not enough taxi drivers available, the system notifies the operator for possible manual adjustments.

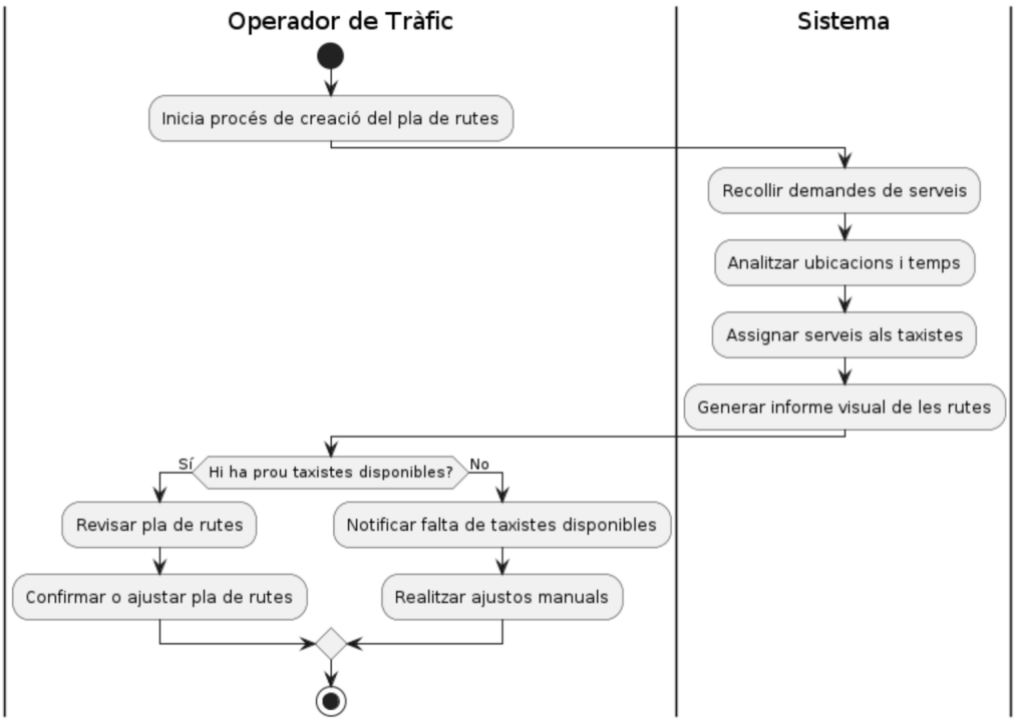

Figure 2 - Activity Diagram Cd'ú 01

3.4.2 *Cd'ú 02. Adjusting Scheduling Algorithm Parameters*

Summary of functionality: This use case allows the traffic operator to modify the parameters of the scheduling algorithm to optimize service performance according to the specific needs of the moment. The operator can minimize the number of taxi drivers needed, maximize service coverage, adjust heuristics on the specified day, and prioritize taxi performance over compliance with all services, or vice versa.

Input parameters:

- Selection of planning mode (optimization for fewer taxi drivers, compliance with all services, etc.).
- Other specific parameters such as maximum distance per route, maximum duration of service, etc.
- Number of taxis available.
- Rank of taxis to evaluate above and below.
- Algorithm configuration parameters, such as peak hours, maximum waiting time, etc.

Output parameters:

- Confirmation of the application of the new parameters.
- Preview how changes affect scheduling.
- Numerical and graphical representation of the corresponding metrics.

Actors:

- Traffic operator.

Precondition:

- The operator must be authenticated and authorized to make changes to the parameters of the algorithm.

Postcondition:

- The algorithm uses the adjusted parameters for generating future routes.

Main normal process:

1. The operator accesses the configuration section of the first dashboard from the frontend.
2. The system displays the current parameters of the algorithm.
3. The operator adjusts the number of taxis available using the first slider.
4. The operator adjusts the rank of taxis above and below using the second slider.
5. The operator adjusts the heuristics for the specified day on the middle panel.
6. The operator selects the desired scheduling mode in the right-hand panel, prioritizing between taxi performance and compliance with all services.
7. The operator saves the changes.
8. The system validates the parameters and recalculates a preview of how these changes might affect scheduling.
9. The system applies the parameters and updates any future schedules with this new configuration.

Process alternatives and exceptions:

1st. If the entered parameters are invalid (e.g. parameters that do not meet system limits), the system displays an error message and asks the operator to review them.

2nd. If it is not possible to apply the changes on the spot for any reason (e.g., connection problems), the system notifies the operator that the changes could not be saved.

Figure 3 - Sequence diagram Cd'ú 02

3.4.3 Use Case 03: Show Service Distribution of the Day

Functionality summary: This use case allows you to display the distribution of the day's services over the hours using a histogram on the homepage. This provides an overview of the number of services scheduled for the day and busiest times.

Input parameters:

- List of services of the day, including the hours of each service.

Output parameters:

- Visual histogram showing the frequency of services throughout the hours of the day.

Actors:

- Traffic Operator
- Supervisor
- System Administrator

Precondition:

- The data of the day's services must be loaded and available in the system.

Postcondition:

- A histogram is visible on the homepage, providing detailed information about the distribution of services throughout the day.

Main normal process:

1. The system collects data from the day's services, including scheduled hours.
2. The algorithm analyzes the hours of each service and generates a histogram.
3. The system displays the histogram on the main page, highlighting the periods with the highest concentration of services.
4. Users can interact with the graph to get additional information about scheduled services.

Process alternatives and exceptions:

1st. If there is no service data available for the day, the system displays a message indicating that there are no scheduled services.

2nd. If histogram generation fails, the system notifies the user and offers options to retry the data load or contact technical support.

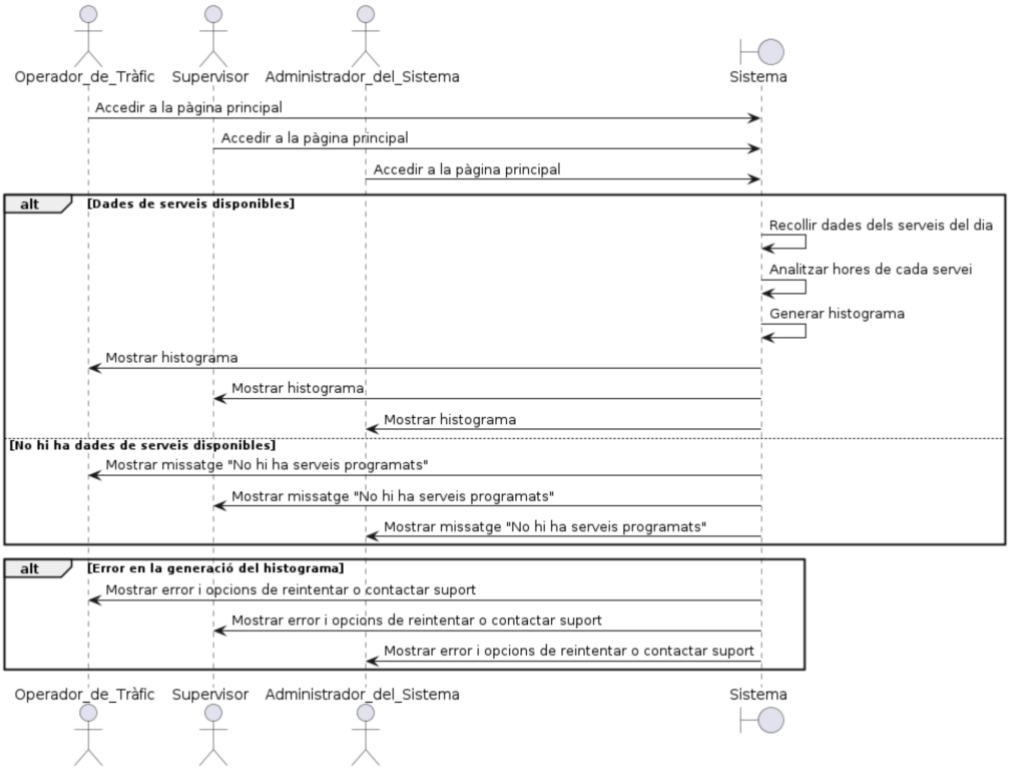

Figure 4 - Sequence diagram Cd'ú 03

3.4.4 Use Case 04: Show Scheduling Metrics Results

Summary of functionality: This Use Case allows you to display, under the service distribution diagram, the results of the algorithm scheduling metrics in an information panel. Metrics include the percentage of taxis used, the percentage of triangular services, the percentage of unassigned services, and the ratio of hours driven versus hours of service by drivers.

Input parameters:

- Results of algorithm-generated scheduling metrics.

Output parameters:

- Information panel showing:
  - Percentage of taxis used.
  - Percentage of triangular services.
  - Percentage of unassigned services.
  - Ratio of hours driven versus hours of service of drivers.

Actors:

- Traffic Operator
- Supervisor
- System Administrator

Precondition:

- Scheduling metrics data should be generated and available in the system.

Postcondition:

- An information panel is visible on the homepage, providing detailed information about scheduling metrics.

Main normal process:

1. The algorithm completes the schedule and generates the corresponding metrics.
2. The system collects data from scheduling metrics.
3. The system displays the following metrics in a panel under the service distribution diagram:
   - Percentage of taxis used.
   - Percentage of triangular services.
   - Percentage of unassigned services.
   - Ratio of hours driven versus hours of service of drivers.
4. Users can view these metrics to assess scheduling efficiency and make informed decisions.

Process alternatives and exceptions:

1st. If no metrics data is available, the system displays a message that no metrics are available to display.

2nd. If an error occurs in the collection or display of metrics, the system notifies the user and offers options to retry or contact technical support.

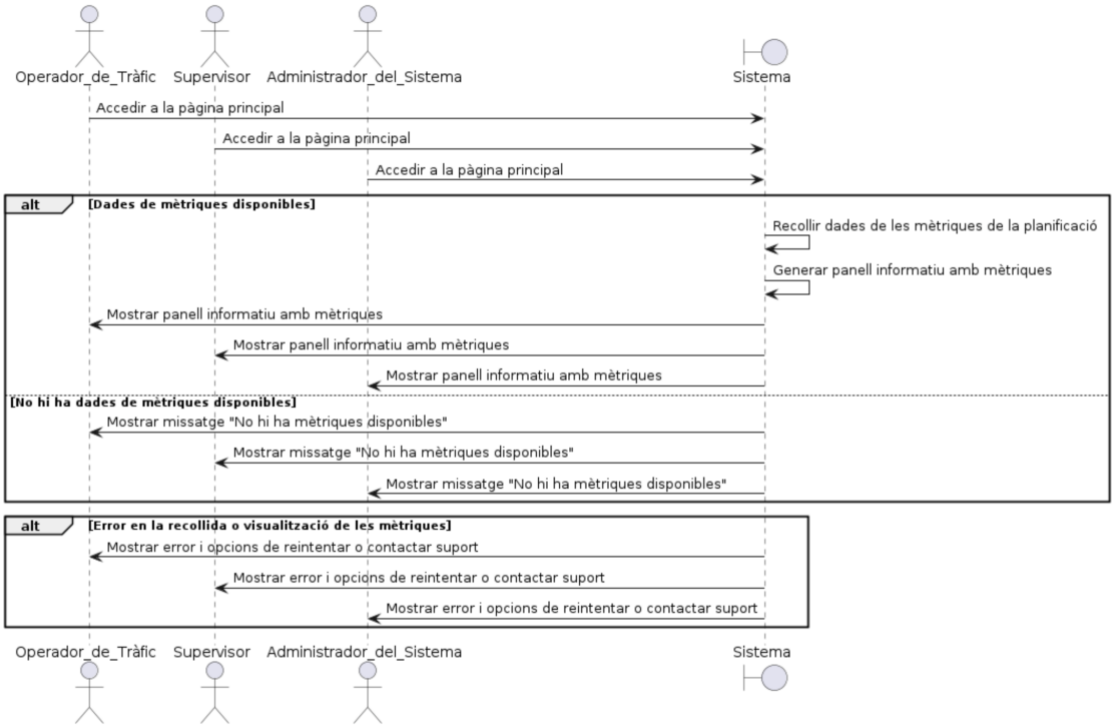

Figure 5 - Sequence diagram Cd'ú 04

4 Design

4.1 Application architecture

The architecture of the application is designed to manage and optimize taxi services through various phases of data processing. This architecture allows a smooth integration between the different components of the system, guaranteeing an efficient distribution and a relatively fast response to user requests.

The following diagram illustrates the interaction between the main components of the app. The system begins with the reception of service, conductor and configuration data, which are preprocessed along with a matrix of maps. This pre-processed data is sent to the trip planner to generate metrical, planned services. Finally, these results are sent to the front end for viewing and interaction by users.

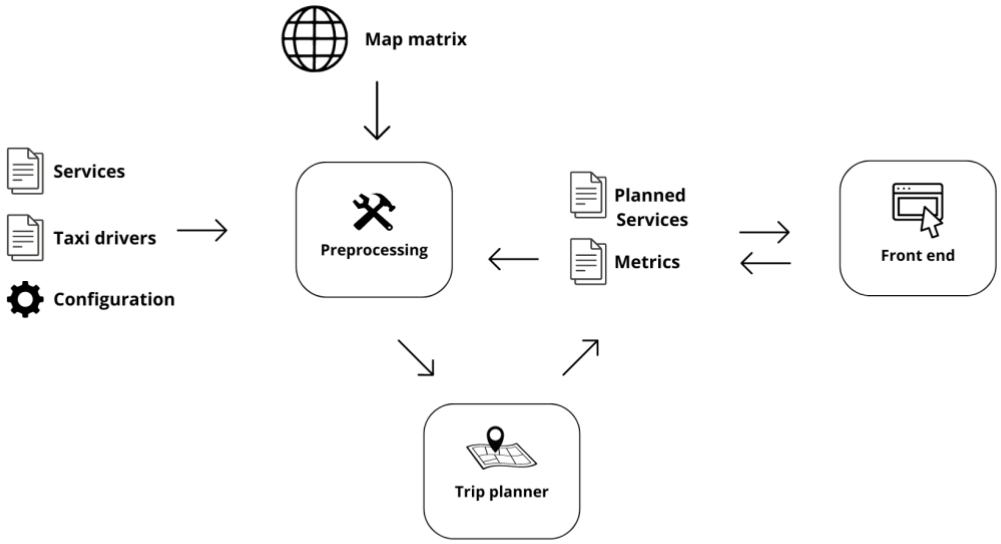

Figure 6 - Schema of the application

4.2 Backend architecture

The backend architecture is designed to efficiently manage application logic. This organized structure allows the separation of different functionalities, including the main algorithm, API management, cache, and data configuration. This division facilitates the maintenance and scalability of the system, ensuring that all requests are processed optimally.

```
VPR-algo/
├── .pytest_cache/
├── date/
├── plots/
├── SRC/
│   ├── something/
│   ├── API/
│   ├── caches/
│   ├── data_setup/
│   ├── drafts/
│   ├── tests/
│   ├── useful/
│   ├── .env
│   ├── __init__.py
│   ├── app.py
│   ├── app_prediction.py
│   ├── app_scheduler.py
│   ├── main_testing.py
│   └── run_tests.py
├── Venv/
├── .gitignore
├── get.json
└── requirements.txt
```

Explanation of directories:

date/:

Directory used to store datasets (services and drivers mainly) necessary for the application.

plots/:

Directory for storing test performance graphics.

src/:

The main source code directory containing submodules and the main code of the application.

something/:

Contains algorithms or the main logic of the application.

API/:

It contains API-related code, including endpoint definitions and request management.

Caches/:

Directory for cache management, from the daily distance matrix.

data_setup/:

It contains scripts for data configuration such as data preprocessing, singelton configuration manager and more.

drafts/:

Directory for draft scripts or experimental code under development.

Tests/:

It contains unit tests, integration tests, and other test cases.

useful/:

Utility functions and auxiliary scripts that provide common functionalities used through the application.

.env:

Environment variable definitions file.

init.py:

Initialization file for the src module, converting it into a package.

app_scheduler.py:

Script to manage scheduled tasks, the main.

4.2.1 Detailed flow

4.2.1.1 Receipt of the Petition

A request is sent to the server via the endpoint **/process**. This endpoint, defined within the Flask configuration, acts as the gateway for client interactions, managing HTTP requests that reach the system.

4.2.1.2 Data Processing

The validated data is then prepared and transformed (from JSON to DataFrame) using different functions. These functions adapt the data for analysis, ensuring that they are in the right format to be processed efficiently. This step is critical to providing accurate and reliable calculations in subsequent stages. Although it seems a simple task, this has been one of the parts that has required more debugging, since not having strong typing with Python have appeared many errors when executing the program.

4.2.1.3 Service Distribution Optimization

With the data already prepared, several optimization algorithms are executed that seek to improve the distribution and allocation of services. These include generating distance matrices to plan routes and custom algorithms for assigning services to drivers based on location, availability and other critical factors.

The details are in the specification.

4.2.1.4 Response Training

Once processing and optimization is complete, the results are compiled and packaged within a JSON response structure. This response includes details such as schedules of assigned services, optimized routes, and any other relevant results. Finally, the formatted response is sent back to the customer.

4.3 Frontend Architecture

The frontend architecture is designed to provide a simple but rich user interface, using Vue.js. This modular structure allows efficient management of components, navigation, state management and communication with the backend, ensuring a smooth user experience. The key components of the frontend architecture are listed below:

FRONTENDCONECTADRIVE

```
├── public/
├── SRC/
│   ├── assets/
│   │   ├── demo/
│   │   ├── images/
│   │   ├── layout/
│   │   └── styles.scss
│   ├── boot/
│   ├── components/
│   ├── composable/
│   ├── config/
│   ├── consts/
│   ├── helpers/
│   ├── layout/
│   ├── mocks/
│   ├── router/
│   ├── services/
│   │   ├── bookings/
│   │   ├── business/
│   │   ├── drivers/
│   │   ├── flights/
│   │   └── locations/
│   ├── stores/
│   ├── types/
│   ├── useful/
│   ├── views/
│   ├── App.vue
│   ├── import-meta-env.d.ts
│   ├── main.ts
│   ├── registerServiceWorker.js
```

```
│ └── style.css
```

This scheme is relatively extensive since it is the entire frontend, the taxi part is a module.

The key components of this architecture are described below:

1. **Explanation**
2. **Audience/**:
    - Directory for public files that do not need to be compiled and are accessible directly from the browser.
3. **src/**:
    - The main source code directory containing submodules and the main code of the application.
    - **assets/**:
        - It contains static resources such as images, styles and other support files.
    - **boot/**:
        - App initialization scripts.
    - **components/**:
        - It contains reusable Vue.js components.
    - **components/**:
        - It contains composables (hooks) to reuse the logic of Vue Composition API.
    - **config/**:
        - Contains configuration files for the application.
    - **consts/**:
        - It contains constants used through the application.
    - **helpers/**:
        - Auxiliary functions and utilities.
    - **layout/**:
        - It contains layout components and page structures.
    - **mocks/**:
        - It contains simulated data or mocks for testing.
    - **router/**:
        - Vue Router route configuration for in-app navigation.
    - **services/**:
        - It contains services for communication with external APIs and data management.

            …
    - **stores/**:
        - It contains warehouses for managing the state of the application with Pinia.
    - **types/**:
        - TypeScript type definitions used in the application.
    - **useful/**:
        - Utility functions and auxiliary scripts.
    - **views/**:
        - It contains app views, which are page-level components.

- **App.vue**:
  - Main component of the Vue app.
- **main.ts**:
  - Main entry point for app initialization.
- **style.css**:
  - CSS file for global styles.
  -

4.3.1 SPA structure[9]:

The frontend is built as a single-page application using Vue.js, which allows dynamic loading of content without the need to reload the page. This improves the user experience and reduces the perceived loading time.

4.3.2 Management of states with Pinia:

Pinia is used for state management across the frontend. Statuses represent current app information, such as user data or algorithm settings. Pinia allows you to create a centralized and reactive state that can be accessed and modified from any component, thus facilitating communication between components and ensuring that changes in the state are reflected throughout the application.

Using Pinia is especially suitable compared to using props, especially considering the architecture of the application. Passing information through props in an application with multiple nested components can be complex and difficult to maintain, since each level must transmit the data.

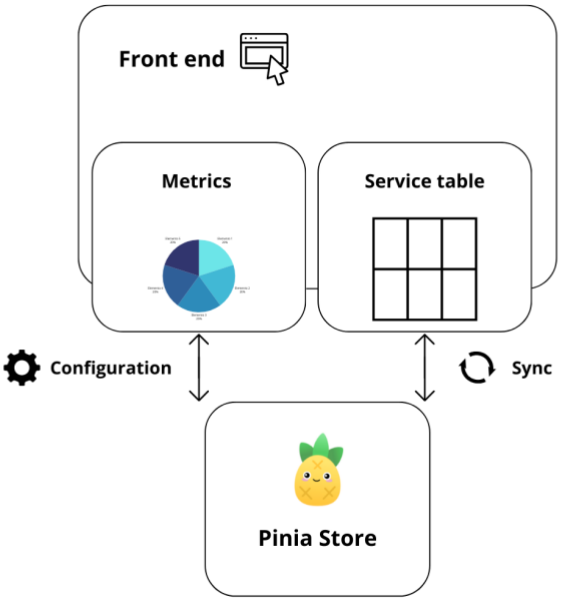

Figure 7 - Frontend States Scheme

[9] Single Page Application

4.3.3 Communication with the Backend:

For interaction with the backend, the Axios library is used to make HTTP requests. Axios is a JavaScript library that facilitates HTTP requests from the browser and Node.js, allowing you to manage asynchronous responses and errors easily. This integration is encapsulated in dedicated services (/services), which abstract the logic of requests. Subsequently, the call from the General Store is used to obtain the data.

4.3.4 Components and Data Visualization:

The frontend makes extensive use of reusable components designed with PrimeVue, which offer a rich and customizable interface. These components include data tables, diagrams, buttons, and other user interface elements that can be customized and reused throughout the app. This has been a great advantage since creating components like in DataTable would have required a lot of dedication.

4.3.5 Navigation and Routing:

Vue Router is used to manage navigation within the SPA app. This includes defining routes, managing redirects, and protecting routes that require authentication. Unlike other frameworks such as Vue's Nuxt, or Angular, these already have a directory structure that does automatic routing.

4.4 Wireframes

Wireframe of how changing pages and displaying a taxi driver shows the services assigned to him.

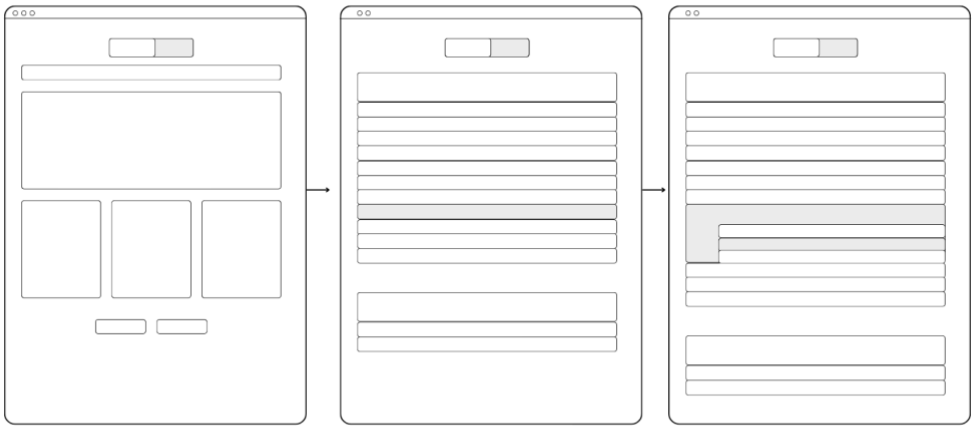

Figure 8 - Wireframe menu of calculated services.

Wireframe of how by clicking on the distribution of services this unfolds and shows a dynamic tabla with the distribution of taxis throughout the day.

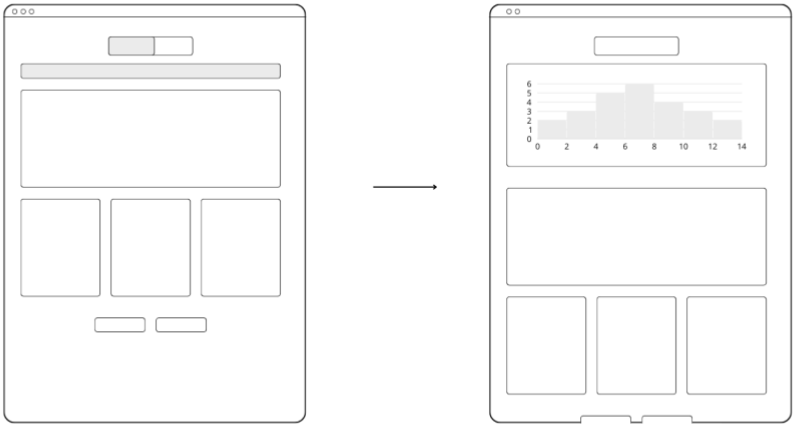

Figure 9 - Drop-down distribution of current services

5 Implementation

5.1 Service scheduler algorithmic implementation

This section explains in detail the logic behind taxi distribution, including the type of algorithm chosen, the reasons for this choice, and an assessment of the alternatives considered.

5.1.1 Choice of Algorithm

The algorithm chosen for taxi distribution is based on a combination of heuristic optimization and rules-based mapping, with elements of greedy algorithm. This approach has been selected for several reasons:

- **Computational Efficiency:** Heuristic optimization allows reasonably good solutions to be found in a reasonable amount of time, even for complex problems with many variables and constraints. This is crucial for real-time applications where decisions need to be made quickly.
- **Flexibility:** This type of algorithm is flexible and allows you to easily incorporate new rules and restrictions, such as peak hours, maximum waiting times and maximum service durations. This adaptability is essential to manage the complexities of taxi service.
- **Simplicity and Maintainability:** Although more advanced algorithms such as neural networks or linear programming models could offer more optimal solutions, they are also more complex to implement and maintain. A heuristic approach makes code simpler and easier to understand and maintain.

5.1.2 Greedy function in the algorithm

Although the main algorithm is not strictly greedy, it incorporates elements of this approach into the initial service allocation process. In this first step, services are assigned to available taxi drivers who comply with time and availability restrictions, similar to a greedy algorithm. This initial phase looks for a quick and viable solution, but it can get stuck in local optima.

Example of greedy assignment:

- For each service, look for the first available taxi driver who can fulfill it.
- Assign the service to the selected taxi driver without considering the impact on future services.

This approach is quick and easy, but can result in an inefficient distribution of services, where some taxi drivers have many hours of work while others have very few.

5.1.3 Heuristic Optimization and Reassignment

To overcome the limitations of the initial greedy approach, the algorithm incorporates a heuristic optimization phase. After the initial assignment, underutilized taxi drivers are

identified and their services are regrouped with unassigned services. This stochastic or random process allows us to escape from suboptimal solutions and find a better combination of services.

Example of heuristic optimization:

- Identify underused taxi drivers with less than 7 hours of service (configurable variable).
- Regroup the services of these taxi drivers with the services not assigned.
- Try to reallocate these services in such a way as to optimize the use of taxi drivers.

This process is repeated until a more balanced solution is reached.

5.1.4 Differences between Greedy Algorithm and Heuristic Optimization

- **Greedy algorithm**

The greedy algorithm makes local decisions based on immediate profit without considering future impact. It is easy to understand and implement, and its calculation time is fast because it only considers the most optimal choice at each step. However, this approach can get stuck in local optima and often fails to achieve overall optimal solutions.

- **Heuristic Optimization and Rule-Based Allocation**

Heuristic optimization considers the overall impact of each decision and applies heuristics to optimize the overall solution. It is more complex but offers better quality solutions. This approach may be slower due to multiple iterations and reassignments, but it is more adaptable and able to consider multiple factors and constraints to achieve a more efficient solution.

5.1.5 Alternatives considered

5.1.5.1 Linear Programming (LP)

Linear programming involves defining variables and constraints to model a problem, allowing an objective function to be optimized. It requires an accurate description of all constraints and variables. It is capable of handling very large problems with high computational intensity. Solvers like CPLEX or Gurobi are especially efficient at solving large problems in reasonable time. However, for very large problems with many variables and constraints, computational intensity may be high, which might not be ideal for a production environment with real-time constraints.

If linear programming were to be used for this problem, it would be necessary to define variables for each possible assignment of services to taxi drivers and create restrictions to ensure that each service is fulfilled within time, and that taxi drivers do not exceed their working hours. The target function could be to minimize total passengerless driving time.

5.1.5.2 Genetic Algorithms

Genetic algorithms are highly adaptable to deal with complex and nonlinear problems. Their ability to escape local maximums allows them to find more efficient global solutions compared to other algorithms. However, these algorithms require a lot of fine-tuning of parameters, such as mutation and crossover rates, to achieve optimal results. In addition, it

can take many generations to converge to an acceptable solution, which can significantly increase the calculation time.

A genetic algorithm for taxi distribution could start with an initial population of random solutions (service assignments to taxi drivers). Each iteration, or generation, would apply selection, cross, and mutation operators to create new solutions. Solutions are rated based on a fitness function, such as minimizing total passengerless driving time, and the best ones are selected to train the next generation population. The implementation of this alternative has been carried out but a functional solution has not been obtained, since the nature of the algorithm made debugging very complicated. In addition, the current solution is efficient, since without the bottleneck of the distance matrix generation the result is practically instantaneous.

5.1.5.3 Neural Networks and Machine Learning

Neural networks and machine learning can generalize complex patterns from large amounts of historical data. These use layers of interconnected nodes to process information in a similar way to the human brain. Layers adjust their weights during training to identify relationships and patterns in the data. This capacity for continuous learning allows a progressive improvement in decision-making. However, these models need a lot of data to be trained effectively, and decisions made can be difficult to interpret and justify, especially in critical systems where the reasons behind each decision need to be explained.

Using a neural network to predict optimal service allocation could involve training a model with historical service and assignment data. The model could learn to predict which taxi drivers to assign to each service to minimize passengerless driving time and maximize customer satisfaction.

5.2 Algorithm for Determining the Optimal Number of Taxis

To find the optimal number of taxis during a season, an algorithm was developed that defines an objective function in order to minimize or maximize it according to user preferences. These preferences can be set from the main screen and may vary by season. For example, users may prioritise the use of fewer taxis to maximise collaboration with other taxi drivers or may prefer to have more taxis to cover all services with less collaboration.

5.2.1 Description of the Algorithm

The algorithm uses an approach that combines simulation with optimization to find the optimal number of taxis. The process followed by this algorithm is described below:

5.2.1.1 Definition of the Objective Function

The objective function is responsible for evaluating each combination of taxis and services according to a set of weights that the user defines. These weights indicate the relative importance of each metric to be maximized or minimized, such as the number of completed services, waiting time, economic performance, among others. The target function considers these weights and calculates a score for each possible combination of taxis and services, with the aim of finding the combination that obtains the best overall score.

```
def evaluate_priority(metrics, weights):    score = 0

    for weight, metric in zip(weights, metrics):
        if weight > 0:  # We want to maximize
                score += (weight - metric) * abs(weight)
        else:  # We want to minimize
                score += (metric + abs(weight)) * abs(weight)
    return score
```

5.2.1.2 Simulation for Each Number of Taxis

Different scenarios are simulated with a varied number of taxis, within a range specified by the user. For each number of taxis within this range, the algorithm performs the following operations:

- **Data generation:** From the real data of the services (such as collection times, locations, durations), a simulation of taxi operations is generated.
- **Initial assignment:** A first assignment of services to available taxis is made. This initial allocation is adjusted to meet the time and availability restrictions of taxi drivers.
- **Optimization:** After the initial allocation, assignments are reevaluated to improve overall efficiency. This includes identifying underutilized taxis and reallocating their services to balance workload.

5.2.1.3 Massive and Parallel Execution

To determine the best number of taxis, the simulation was run massively and in parallel for each possible value of taxis within a given range. Using the **joblib** parallelization library, multiple simulations were launched simultaneously, each evaluating a different number of taxis. This approach allows to speed up the evaluation process and find the optimal number of taxis in a reasonable time.

For each simulation, performance metrics are calculated based on the previously defined target role. These metrics include:

- Percentage of completed services.
- Average waiting time.
- Use of taxis.
- Economic performance.

The metrics of each simulation are evaluated to find the combination that maximizes or minimizes the target function according to the preferences set by the user.

```
def find_best_number_of_taxis(n_services, df_final, distance_matrix,
weights, n_taxis_range, CONFIG):
    def score_for_n_taxis(n_taxis, config):
        result = run_simulation(n_taxis, n_services, df_final,
distance_matrix, config)
        return evaluate_priority(result, weights), n_taxis

    n_taxis_scores = Parallel(n_jobs=-
1)(delayed(score_for_n_taxis)(n_taxis, CONFIG) for n_taxis in
range(*n_taxis_range))

    best_score, best_n_taxis = min(n_taxis_scores, key=lambda x: x[0])

    return best_n_taxis
```

5.2.2 Advantages and Limitations of the Algorithm

Advantages

- **Flexibility:** It allows you to adjust the objective function according to the specific needs of each season, being able to prioritize different aspects such as collaboration or service coverage.
- **Efficiency:** The use of parallel simulations allows finding the optimal number of taxis in a reasonable time.
- **Adaptability:** The algorithm can incorporate new rules and constraints without the need for significant changes in its structure.
- **Global Optimization:** The combination of initial allocation and subsequent optimization allows to overcome the limits of local optima, finding more balanced solutions.

Limitations

- **Computational Complexity:** Despite the use of parallelization, simulation can require considerable time for a large number of services and taxis.
- **Dependence on Configuration:** The results obtained strongly depend on the weights and parameters configured by the user, requiring a careful configuration to obtain the best results.
- **Variability in Results:** Due to the stochastic nature of the reassignment process, results may vary slightly between different executions.

5.2.3 Algorithm conclusion

This algorithm, through a combination of parallel simulation, is able to find balanced solutions that maximize the efficiency and utilization of available resources.

Unfortunately, the practical usefulness of this algorithm cannot be seen in the frontend, as it was one of the functionalities planned for future versions of the system. At the time of development, this was not a priority task. However, logic is developed and functional in the backend, which would allow its integration in the future when deemed necessary.

On the other hand there was another drawback which was in communicating customer priorities. This part can be very useful but it is a second derivative and really this was a task that should have been developed later since it was not urgent.

5.3 Backend implementation

In this section, the implementation of the backend will be explained with a debugging approach to the program. The decision to adopt such an explanation is due to the existence of a report of the development of the system over the months. This memoir includes all the problems, big and small, and all the design decisions that were made. This would make it very difficult to write a brief and concise memoir. Therefore, instead of explaining the development chronologically, it is better to explain how the program works from start to finish, detailing the key aspects and adding the necessary comments, but without going into all the minutiae of development.

5.3.1 First steps

5.3.1.1 Start of Operation

The operation begins with the entry point of the algorithm, which receives a JSON with the data to be planned, the configuration of the schedule and the number of taxis. This data is transmitted to the system to update local settings and allow modifications. In the SingletonConfigManager class, there is already default data, but it checks for modifications and, if so, changes are made.

```
@app.route('/process', methods=['POST'])
def process():
    data = request.json
    new_date = data.get('date')  # Default date if not provided
    config_updates = data.get('config')
    optimal_config_updates = data.get('optimalConfig')

    result = process_algo(new_date, config_updates,
optimal_config_updates)

    return jsonify(result)
```

5.3.1.2 Data Download

Next, the data of the services of the day in question and the next are downloaded. This decision was made by the client, as the services last up to 28 hours a day. This implies that the structural schedule lasts more than one day, and you have to plan in 28 hours. This decision was a complication of implementation since it was decided in the last moments of development. The solution found was to create a DataFrame combining the data from 4:00 AM on the current day to the first four hours of the next day. Thus, the data can be analyzed correctly and ensure that there is no pending service from the previous day.

```
def adjust_service_times(df1, df2):
    # Convert 4 AM to minutes since midnight
    four_am_in_minutes = 4 * 60

    # Remove services from df1 that are scheduled before 4 AM (240
minutes)
    df1 = df1[df1['Pickup_Time'] >= four_am_in_minutes]

    # Remove services from df2 that are scheduled after 4 AM and add 24
hours to the pickup time
    df2_after_four_am = df2[df2['Pickup_Time'] < four_am_in_minutes]
    df2_after_four_am.loc[:, 'Pickup_Time'] += 24 * 60  # Add 24 hours in
minutes

    # Concatenate the modified df2 to the end of df1
    adjusted_df = pd.concat([df1, df2_after_four_am], ignore_index=True)

    return adjusted_df
```

This is just one example of how enriching it has been to use pandas for data processing and the efficiency of their methods.

5.3.1.3 Extraction of Information from Services

The function to extract information from services makes a request to the endpoint that is in the AWS cloud. Since the project has been decoupled, there is a try-catch where, if it fails, a locally stored service sheet is used to demonstrate how the algorithm works. This allows you to choose by day and take the corresponding services.

5.3.2 Initial data processing

```
def transform_real_data_to_dataframe(df_real: pd.DataFrame) ->
pd.DataFrame:
. . .
df_real['serviceTimeInMins'] = df_real['serviceTime'].apply(
    lambda x: int(x.split(':')[0]) * 60 + int(x.split(':')[1]))
df_transformed = pd.DataFrame({
    'Pickup_Location': df_real['origin'].apply(
        lambda x: tuple(reversed(x['coordinates'])) if 'coordinates' in x
and x['coordinates'] is not None else None),

 . . .
    return df_transformed
in x and x[
                'coordinates'] is not None else None),
df_transformed = df_transformed.dropna(subset=['Pickup_Location',
'Dropoff_Location'], how='any')
...

return df_transformed
```

5.3.2.1 Data creation and pre-processing

Once we have the DataFrame combined with the necessary locations, we must obtain the distance matrix to know the location of each point. After much research, the free library ORS has been used, which provides distance information free of charge. However, it is a bit slow and generates bottlenecks, compared to the algorithm's runtime. To have ORS[10] we had to install it with a Docker Compose file, configuring a service for the openrouteservice (ORS) application. We will also try to specify container construction, port mapping, volume assembly for persistent data, and environment variables for memory configuration and data files.

One option initially considered was to use the Google Maps API. However, this option was later discarded due to the high cost of requests involved in its use.

5.3.2.2 Optimization of the Distance Matrix

One proposed solution was to determine locations for 24 hours and store the array locally in a temporary JSON, which is destroyed at the end of the day. This saves time in processing. An alternative was also created in case of ORS failure, returning a distance matrix with random results from 10 to 81, representing transport times on the island of Mallorca. This solution has been useful at various stages of development.

5.3.2.3 Fill in the DataFrame

Once we have the distance matrix, we must fill in the DataFrame section with the approximate service time in minutes. This attribute is initially empty and is completed after calculating distances. In addition to this attribute, DataFrame includes several other attributes relevant to the correct operation of the backend.

The real data is transformed into the required format by standardizing the names of the locations, filtering the relevant rows and determining the type of service (entry or exit from the airport). Service times are converted to minutes from midnight and a new DataFrame is created with the following attributes:

Pickup_Location: Location of collection (coordinates).

Dropoff_Location: Drop location (coordinates).

Pickup_Location_Name: Name of the collection location.

Dropoff_Location_Name: Name of drop location.

Pickup_Time: Pick-up time in minutes from midnight.

Aprox_Service_Time (mins): Approximate service time in minutes (initially 0).

Service_Type: Type of service (ENTRANCE or DEPARTURE).

Driver_ID: Driver identifier (initially empty).

---

[10] Open Route Service

Flight_Info: Flight information.

Origin: Origin of the service.

Reference_Code: Service reference code.

Service_Date: Date of service.

Vehicle_ID: Vehicle identifier (initially empty).

Vehicle_Type: Type of vehicle (initially empty).

Vehicle_Capacity: Vehicle capacity (initially empty).

The Aprox_Service_Time (mins) column is then updated with the distance matrix values for each location combination. This update gives each service an approximate service time based on calculated distances.

5.3.2.4 Distribution of services

Before determining the optimal taxi schedule, it is necessary to calculate the distribution of services. This is done with a function similar to MapReduce's algorithm. All the data of the service hours are added, rounded, and a dictionary is created for each hour, acting as a counter and returning the object of the distribution.

```
def calculate_hourly_distribution(assigned):
    all_pickup_times = [service[0]['start_time'] for service in assigned]
    all_pickup_hours = [pickup_time // 60 for pickup_time in
all_pickup_times]
    hourly_distribution = [all_pickup_hours.count(hour) for hour in
range(24)]
    print(hourly_distribution)

    return hourly_distribution
```

5.3.3 Main Scheduling Algorithm

5.3.3.1 Final Processing and Scheduling Algorithm

Once the preprocessing is complete: the final DataFrame with the data, the distance matrix and the conductor information, we can proceed to the most complex part of the system: the logic behind the scheduling algorithm.

5.3.3.2 Transformation from Simple Services to Triangular Services

The first step of the algorithm is to transform simple services into triangular services. A triangular service consists of packing a service from the airport to the hotel with a service from the hotel to the airport, thus maximizing the efficiency of the journey and saving fuel and resources. To carry out this transformation, services are classified into two lists: one for airport to hotel services (A2H) and one for hotel to airport services (H2A).

The function that is responsible for finding the optimal combinations of triangular services follows a specific process to ensure that the selected combinations maximize service efficiency. The detailed process is described below:

1. **Classification of Services during Peak Hours**: Services are classified according to their priority during peak hours, ensuring that the services with the highest demand are served first. This is done by determining whether a service is within the peak hours defined in the configuration.
2. **Distance Dictionary Key Conversion**: Distance dictionary keys are converted from list tuples to tuple tuples, making it easy to access distance values between locations.
3. **Iteration and Combination of Services**:
    - Each service from the airport to the hotel is traveled and the end time of the service is calculated.
    - For each airport service, it is verified that the pick-up and drop-off locations are not zero.
    - The key to the distance between the drop-off location of the A2H service and the collection location of the H2A service is determined.
    - The transition time between the two services and the waiting time to the H2A service are calculated.
4. **Validation of the Combination**:
    - The wait time is compared with the maximum wait time defined in the configuration, differentiating between peak and off-peak hours.
    - The distance from the current location to the service is compared with the distance from the airport to the service.
    - It is verified that the waiting time is within the acceptable range and that the combination results in a total duration of service within the limits set by the configuration.
    - If all conditions are met, the combination of A2H and H2A services is returned along with the waiting time.

5.3.3.3 Validation and Creation of Triangular Services

Once an optimal combination of services has been identified, an atomic service object is created. This purpose guarantees that the taxi driver completes the service without interruptions. The services thus grouped together are added to a list that will later be assigned to the schedules of taxi drivers. This decision was made, since after much purification, there were services that overlapped, this occurred because the triangular services had not been structured as atomic.

```
if a2h_service and h2a_service:
    compound_service = {
        'nested_services': [a2h_service, h2a_service],
        'Pickup_Time': a2h_service['Pickup_Time'],
        'Aprox_Service_Time (mins)': (h2a_service['Pickup_Time']) +
h2a_service['Aprox_Service_Time (mins)'] - (
        a2h_service['Pickup_Time']),
        'waiting_time': waiting_time
    }
    taxi_schedules.append([compound_service])
    airport_to_hotel.remove(a2h_service)
    hotel_to_airport.remove(h2a_service)
```

5.3.3.4 Allocation of Services to Taxi Drivers

After creating the list of triangular services, the assignment of these services to taxi drivers is moved. This process involves creating a list of taxi drivers with main attributes such as service time, driving time, availability and list of assigned services.

5.3.3.5 Driver Data Collection

The process begins by collecting data from drivers to create a structured list of taxis. Each ticket represents a taxi, initially with no assigned driving or service hours. Each one is associated with an empty list intended to accumulate the services that will be assigned later.

```
driver_info = load_driver_info()
taxi_services = [ {
    "hours_drive": 0, "hours_service": 0,
    'id': driver['id'],
    'name': driver['name'],
    'surname': driver['surname'],
    'isWorkingDay': driver['isWorkingDay'],
    'timeRange': {'start': driver['start_time'], 'end':
driver['end_time']},
    'services': []
} for driver in driver_info]
```

5.3.3.6 Verification and Allocation of Services

Each service is reviewed to determine its duration and it is verified if it can be assigned to a taxi driver without overlaps with other services.

First, you get the start and end times of the service. Then check if the taxi is working that day. If the taxi does not have previous services, adjust its schedule based on the current service. It then checks whether the service is within the taxi time window and, finally, checks for conflicts with other existing taxi services. If there are no conflicts, return True; otherwise, return False.

Below is a visual example of how the driver's schedule would be represented where the color black is a service assigned to him.

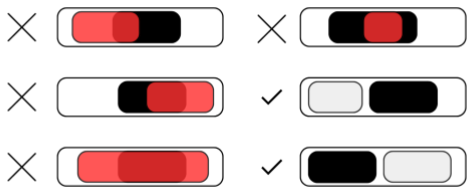

Figure 10 - Service assinging scheme

```
def is_taxi_available_for_service(taxi: Dict[str, Union[int,
List[Dict[str, Union[str, int]]]]],
                                  service: Dict[str, Union[str, int]]) ->
bool:
    start_time, end_time = get_service_times(service)

    if not taxi['isWorkingDay']:
        return False

    if not taxi["services"]:  # If the taxi has no services yet
        taxi_original_start = time_to_minutes(taxi["timeRange"]["start"])
        taxi_original_end = time_to_minutes(taxi["timeRange"]["end"])
        taxi_working_hours = taxi_original_end - taxi_original_start

        # Adjust taxi start and end time based on the first service's
start time
        if start_time < taxi_original_start and (taxi_original_start -
start_time <= 60):
            time_difference = taxi_original_start - start_time
            taxi["start_time"] = start_time
            taxi["end_time"] = taxi_original_end - time_difference
        elif start_time > taxi_original_start and (start_time -
taxi_original_start <= 60):
            taxi["start_time"] = start_time
            taxi["end_time"] = start_time + taxi_working_hours

        else:
            taxi["start_time"] = taxi_original_start
            taxi["end_time"] = taxi_original_end

    # Check if service time is outside the taxi's time window
    if start_time < taxi["start_time"] - 60 or end_time >
taxi["end_time"]:
        return False

    # Check for conflicts with existing services
    for existing_service in taxi['services']:
        existing_start, existing_end =
get_service_times(existing_service)
        if (existing_start <= start_time < existing_end) or \
           (existing_start < end_time <= existing_end) or \
           (start_time <= existing_start and existing_end <= end_time):
            return False

    return True
```

This previous function has been one of the most elaborate and key of the whole project, because if there were services that overlapped, it was useless that the algorithm was very fast.

The testing that has been done is explained later.

5.3.3.7 Improved Allocation

The next step is the effective checking and assignment of services to available taxis, ensuring that they do not exceed the maximum hours of service allowed. After the assignment, an ordering of the services is carried out within each list of each taxi to ensure a logical and efficient flow. Taxis that exceed the threshold of hours of service are moved to a separate list. These would be considered taxis that are well used, depending on the parameters set in the configuration and also make the list of pending taxis shorter.

This image shows a metaphor to understand the operation and performance of the algorithm in a simple way. You could compare the services as stones and the hours of service of the taxi driver as buckets. The more pebbles we have, the more volume of stone we can try to fit in the bucket. If proportionally there are more buckets than stones, the yield will be low, since the combinations will be limited

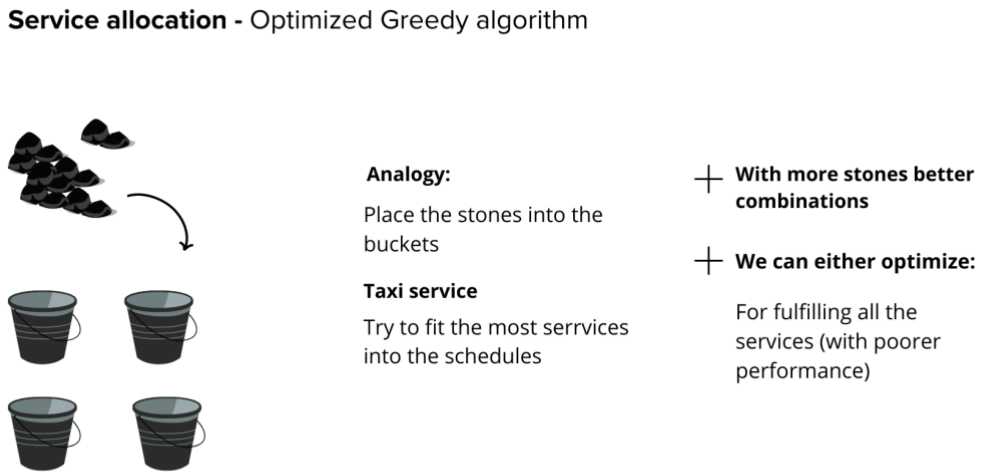

Figure 11 - Algorithm work analogy

5.3.4 Optimization of Assignments

5.3.4.1 Iterative Reassignment

The system performs iterations to improve the distribution of services in taxis that have been underused. During this process, the aim is to optimise the allocation of services to prevent taxis from being idle for long periods or making inefficient journeys.

5.3.4.2 Monitoring and Stability

Each iteration involves checking existing assignments and testing new combinations to see if they improve the efficiency of taxi distribution. Changes in allocations are monitored, and if no significant changes are detected for 50 consecutive iterations, this indicates that the system has reached a point of stability. At this point, the iterative process can be stopped to avoid unnecessary investment of time and resources into more iterations that are likely not to produce substantial improvements.

```
def improve_taxi_allocation(taxi_services, unassigned_services, CONFIG,
max_iterations=500, stability_threshold=50):
    iteration = 0
    stable_count = 0  # Track the number of iterations without changes

    prev_unassigned_count = len(unassigned_services)  # Count of
unassigned services in the previous iteration

    while iteration < max_iterations:
        underperforming_taxis = [taxi for taxi in taxi_services if
taxi['hours_drive'] < CONFIG["UNDERUTILIZED_THRESHOLD"]]
        if not underperforming_taxis:  # If no underperforming taxis,
break the loop
            break

        # Call reorder_taxis_optimized to try and optimize the taxi
utilization
        taxi_services, unassigned_services =
reorder_taxis_optimized(taxi_services, unassigned_services, CONFIG)

        # Check if the state has changed
        if len(unassigned_services) == prev_unassigned_count:
            stable_count += 1
        else:
            stable_count = 0  # Reset the stable count if there's a
change
        if stable_count >= stability_threshold:  # If the state has been
stable for the threshold, break
            break
        prev_unassigned_count = len(unassigned_services)  # Update the
previous count

        # Increase the iteration count
        iteration += 1

    return taxi_services, unassigned_services
```

5.3.4.3 Escape from Local Maximums

This iterative method also allows the system to escape local maxima, i.e. suboptimal configurations where other improvements might be possible. Thanks to the ability to reassign services between underused taxis and combine them with those not yet assigned, the opportunity to reorganize the allocation is provided repeatedly. This makes it possible to discover new, more efficient configurations that may have initially been hidden.

```
def reorder_taxis_optimized(taxi_services, unassigned_services, CONFIG):


    # Step 1: Identify underutilized taxis (less than 7 hours of service)
    underutilized_taxis = [taxi for taxi in taxi_services if
                           taxi['hours_drive'] <
CONFIG["UNDERUTILIZED_THRESHOLD"]]  # less than 7 hours

    # Step 2: Remove the services from these taxis and add them to an
underutilized_services list
    underutilized_services = []
    for underutilized_taxi in underutilized_taxis:
        underutilized_services.extend(underutilized_taxi['services'])
        underutilized_taxi['services'] = []
        underutilized_taxi['hours_drive'] = 0
        underutilized_taxi['hours_service'] = 0

    # Step 3: Merge the underutilized_services with the
unassigned_services
    combined_services = [item for sublist in unassigned_services for item
in sublist] + underutilized_services

    # Step 4: Attempt to assign these services to the emptied taxis
    services_to_remove = []  # We'll track services that get assigned
here
    for service in combined_services:
        start_time, end_time = get_service_times(service)
        hours = end_time - start_time
        assigned = False
        for taxi in taxi_services:
            can_assign = is_taxi_available_for_service(taxi, service)
            if can_assign:
                potential_hours_service = calculate_taxi_window(taxi,
service)
                if potential_hours_service <=
CONFIG["MAX_SERVICE_HOURS"]:  # Check total service hours
                    taxi["hours_drive"] += hours
                    taxi["services"].append(service)
                    assigned = True
                    services_to_remove.append(service)  # Mark service
for removal
                    taxi["services"].sort(
                        key=lambda x: x['Pickup_Time'] if
'nested_services' not in x else x['nested_services'][0][
                            'Pickup_Time'])
                    break
```

```
    # Now, remove the assigned services from combined_services
    for service in services_to_remove:
        combined_services.remove(service)

    # Recreate the unassigned_services list of lists structure
    remaining_unassigned = [service for service in combined_services if
                            not any(service in taxi['services'] for taxi
in taxi_services)]
    remaining_unassigned = [remaining_unassigned[i:i + 1] for i in
range(0, len(remaining_unassigned), 1)]

    return taxi_services, remaining_unassigned
```

5.3.4.4 Identification and Redistribution

This function begins by identifying taxis that have had less than 7 hours of service (constant that can be modified from the oven), considering them to be used below. Services are then withdrawn from these taxis and prepared for a new assignment. The retired services are combined with others that are not yet assigned, creating a new pool of services available to be redistributed.

The combined services are then attempted to be reassigned to available taxis, taking care to adjust them within the operational limits and schedules of taxis. This process continues until all services are assigned or the 500-iteration limit is met (if it takes a long time it can be reduced). This approach ensures that the use of available resources is maximized and the operational efficiency of the system is maintained.

5.3.5 Last steps

5.3.5.1 Calculation of total hours

A detailed calculation of the total hours of service carried out by each taxi is made. This assessment provides a clear picture of each driver's individual contribution to overall service performance. By closely monitoring time and productivity, management is facilitated and the performance of each taxi within the system is optimized.

5.3.5.2 Generation of Results

Upon completion of these calculations, the system generates an exhaustive set of results. This includes a detailed description of the status of each taxi, documenting the completed services for each taxi. In addition, the schedules of the original services are reviewed and listed, along with those that still need to be allocated or redistributed. This organization ensures that no resource remains unused efficiently, or at least leaves room to be modified by a traffic agent.

5.3.5.3 Performance Metrics

It is also important to create accurate metrics that quantify system performance. These metrics include the number of triangular services processed and the total services managed during the evaluated period. By calculating percentages such as the number of completed

services by type and the proportion of taxis that have actually been in operation, a solid basis for informed decisions is obtained. The metrics also assess the proportion of unassigned services and average hours worked compared to service hours, providing a key indicator of operational efficiency and labor distribution fairness.

5.4 Frontend implementation

5.4.1 Initial Data Upload

1. **Show load indicator**: Until the data is available, a load indicator is displayed to inform the user that the system is working on data recovery. This initial load is done using an **onMounted** function, which is a component lifecycle hook in frameworks such as Vue.js.
2. **Verify the local cache**: Before making the API call, it is verified that the data is already stored locally (cache). This is done to optimize response time and avoid unnecessary calls to the backend. If data is found in the local cache (stored, for example, in localStorage), it is loaded from there and immediately displayed to the user.

```
onMounted(async () => {
  const today = new Date().toISOString().split('T')[0] // Get today's
date
  const cachedData = localStorage.getItem('apiCache_${today}') // Try
to get the cached data

  loading.value = true
  if (cachedData) {
    If cache exists, use it
    console.log('Using cached data...')
    const data = JSON.parse(cachedData)

    store.setData(data)

    console.log('Cache used successfully!')
  } else {

    // If no cache, fetch data from API
    try {
      console.log('Fetching data from API...')
      await store.updateBackend()

      console.log('API data fetched successfully!')
      Cache the API response locally
      localStorage.setItem('apiCache_${today}',
JSON.stringify(store.backendData))
      console.log('Data cached for faster future loads.')
    } catch (error) {
      console.error('There was an error fetching the data!', error)
```

```
    }
  }
  loading.value = false
})
```

3. **Backend API call**: If no data is found in the local cache, a backend API call is made to get the necessary data. This call is made using a library such as axios to handle HTTP requests.
4. **Response processing**: Once the API response is received, the received data is processed to ensure that it is in the correct format to be used on the frontend.

```
export async function processServices(config?: Config, optimalConfig?: OptimalConfig) {
  const response = await axios.post<year>('http://127.0.0.1:5000/process', {
    date: '2023-9-6',
    config: config,
    optimalConfig: optimalConfig
  })

  console.log(response.data)

  return {
    results: response.data.results,
    n_services: response.data.schedule.n_services,
    n_triangulars: response.data.schedule.n_triangulars,
    metrics: response.data.metrics,
    transformedTaxiSchedules: tranformTaxiSchedules(response.data.schedule.taxi_services),
    taxiSchedulesToDistribute: response.data.schedule.taxi_schedules_to_distribute.reduce(
      (acc: year, val: year) => acc.concat(val),
      []
    ),
    service_distribution: response.data.service_distribution
  }
}

const tranformTaxiSchedules = (taxiSchedules: year[]) => {
  console.log(taxiSchedules)
```

```
  return taxiSchedules.map((schedule, index) => {
    const hours_drive = schedule.hours_drive // Adjust as needed
    const hours_service = schedule.hours_service // Adjust as needed

    return {
      ID: index + 1,
      total_services: schedule.services.length, // Adjust this as needed
      hours_drive,
      hours_service,
      services: schedule.services // Adjust this as needed
    }
  })
}
```

5. **Updating the status and concealment of the load indicator**: Finally, the processed data is loaded into the component state and the load indicator is hidden. At this point, the data is ready to be visualized by the user.

5.4.2 Component Structure

The main component is structured in a series of sections that are shown or hidden depending on the state of the application and user interactions. This structure is implemented using **v-if** conditions to manage the visibility of each part of the content.

1. **Load indicator**: Displayed when data is not yet available.

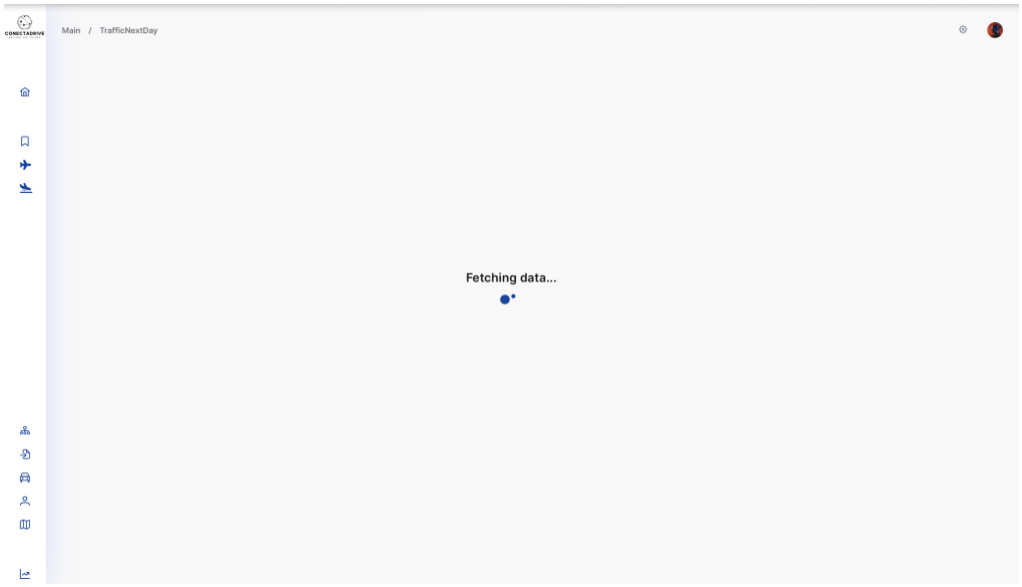

Figure 12 - Main page loading the login data

2. **Main content**: Once the data is loaded, the main content is displayed organized in different views.

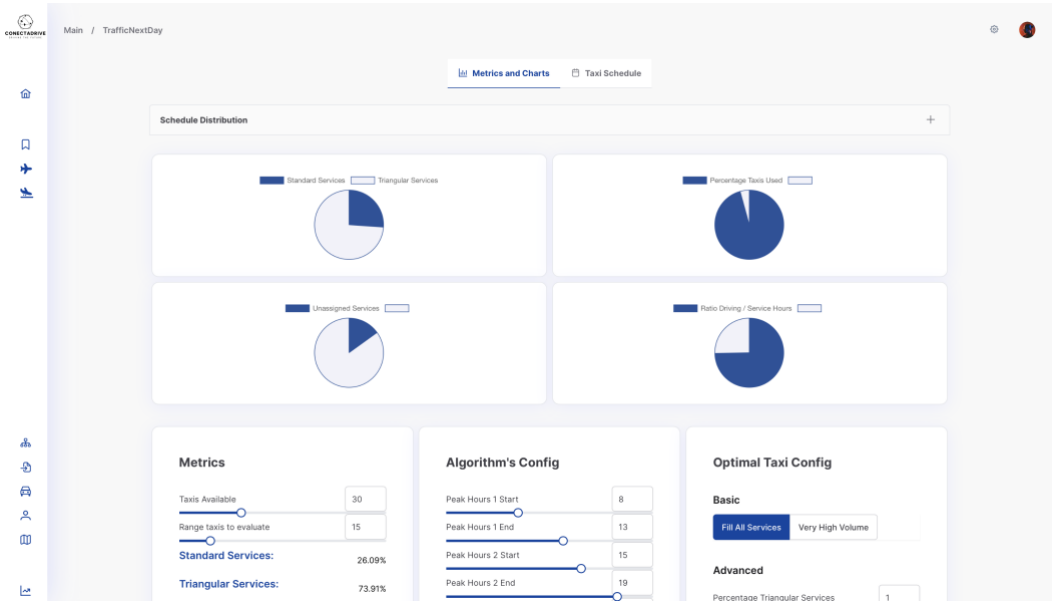

Figure 13 - Main page

5.4.3 Tabs and Navigation Menu

The tabbed menu allows users to switch between different views: metrics and service scheduling. This functionality is managed with a tabbed object that defines the different views available and the actions to be performed when a tab is selected.

1. **Tab Definition**: A data structure is created that defines the available tabs and associated icons, as well as the function that will run when a tab is selected.
2. **Manage Active View**: The status of the component keeps information about which tab is currently active. When a user selects a different tab, this information is updated, and the corresponding content is displayed using **v-if conditions**.
3. **User interaction**: When the user selects a tab, the active view is changed and the content associated with that tab is displayed. This provides an intuitive way to navigate between the different functionalities of the application without the need to reload the page.

5.4.4 Metrics View

The metrics view is displayed when the user selects the corresponding tab or the default page loads. This view includes charts and other visuals to show system metrics. The graphs are generated with the data obtained from the backend and are presented in different formats such as bars and cakes to provide a detailed view of the performance of the taxi service.

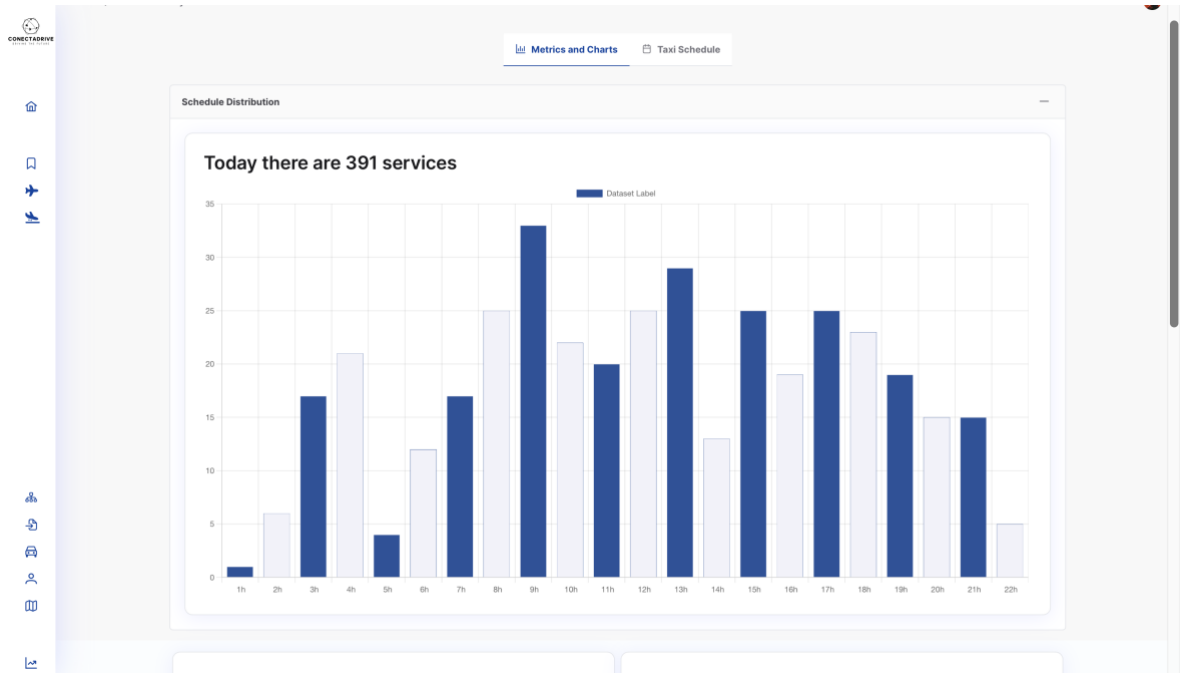

Figure 14 - Distribution of services per hour

5.4.5 Service Scheduling View

The Service Scheduling view shows a full detail of the service assignments for each taxi. This view uses a data table to show assigned services and their characteristics. Each row in the table represents a taxi, and you can expand the rows to see additional details about the services assigned to each taxi.

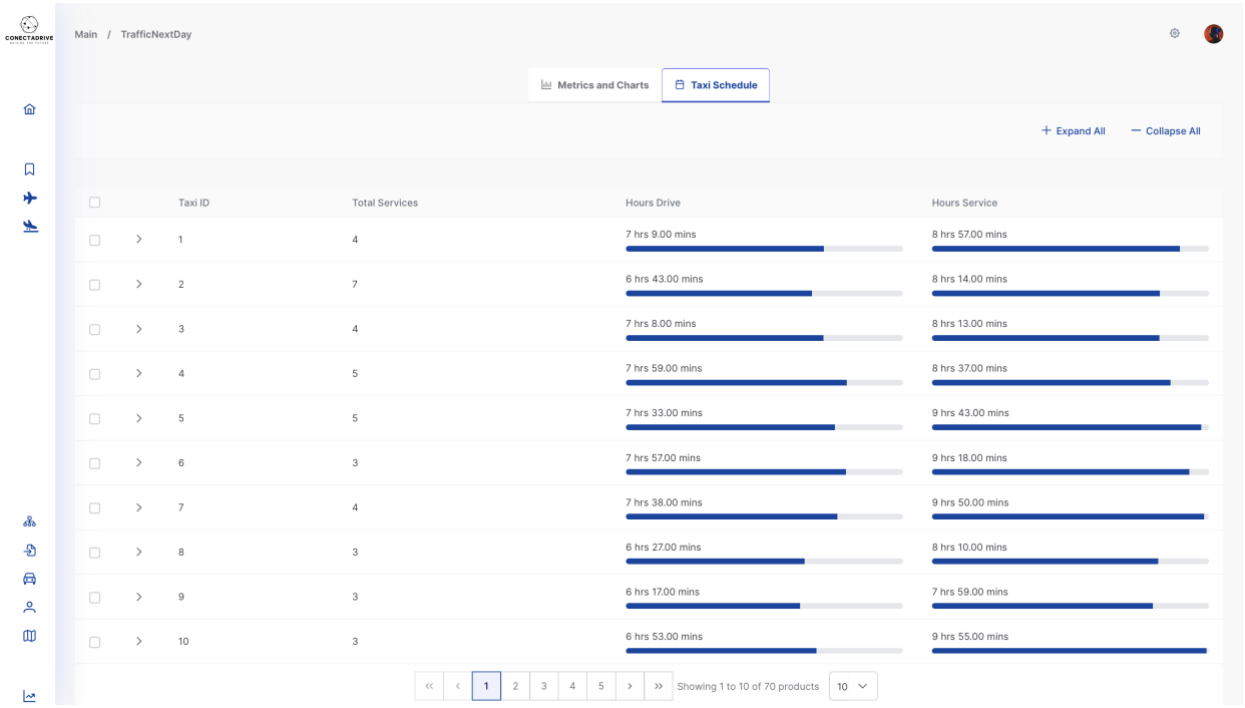

Figure 15 - Service table

The table includes columns for different service attributes, such as pick-up and drop-off location, pick-up time, approximate service time, and service type. You can also see triangular services, which group round trips between the airport and hotels.

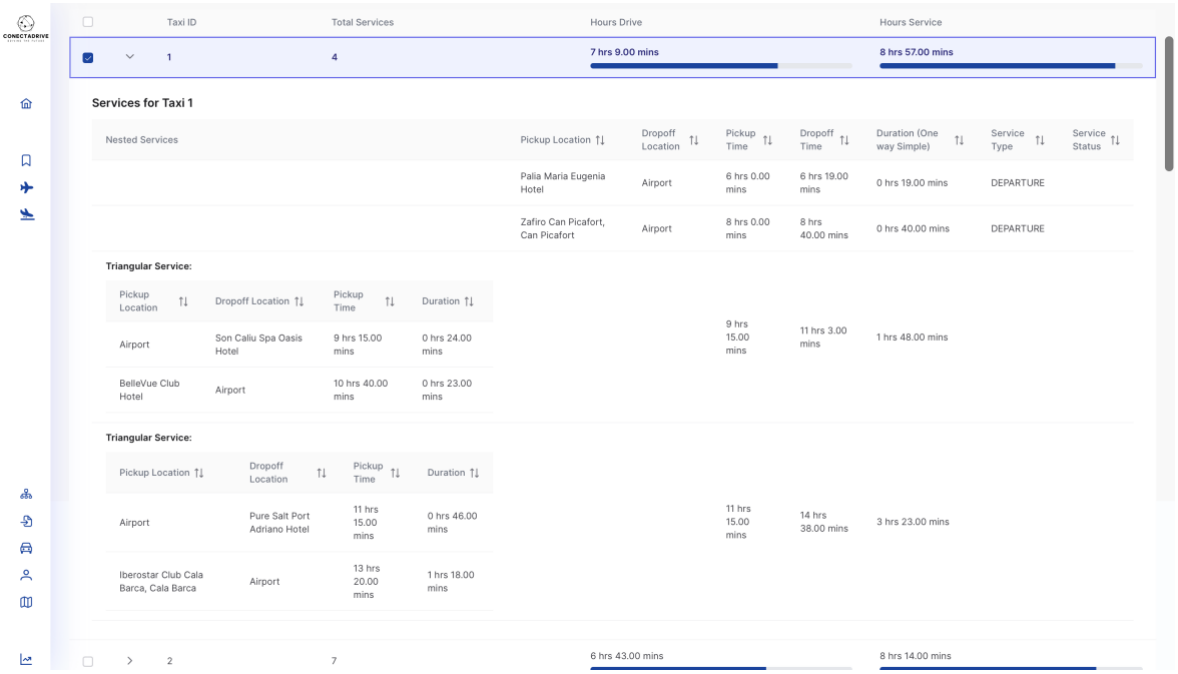

Figure 16 - Service table deployed

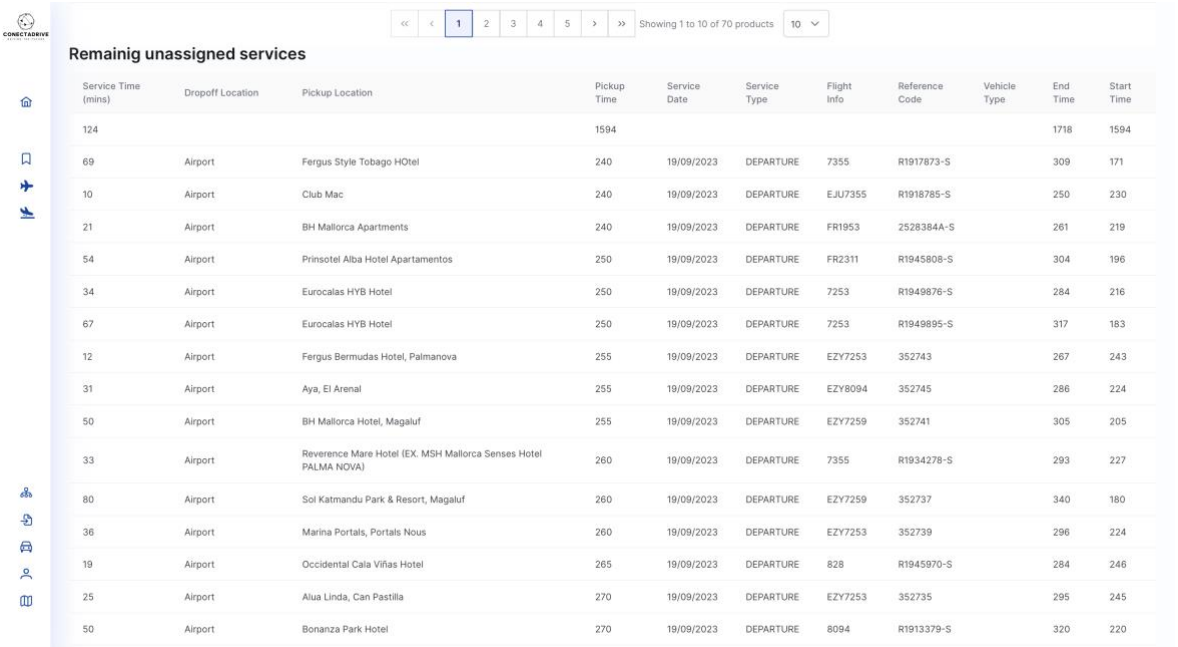

Figure 17 - Unassigned service table

5.4.6 Settings

The frontend allows you to configure various parameters for the scheduling algorithm. This setup is done through an interactive interface that includes input controls and sliders.

Users can adjust peak times, maximum wait times, and other important parameters to optimize system operation.

In addition, detailed metrics about the efficiency of the system are shown, such as the number of services assigned, the total driving and service time, and the proportion of taxis used. These metrics are calculated from the data obtained from the backend and are displayed in real time to provide valuable information for decision making and continuous optimization of taxi services.

A clarification on the difference between the 2 sliders in the middle and the left, is that the one in the middle manages the internal configuration of the parameters of the algorithm and the other tries to optimize the number of taxis according to the objectives of the day / season.

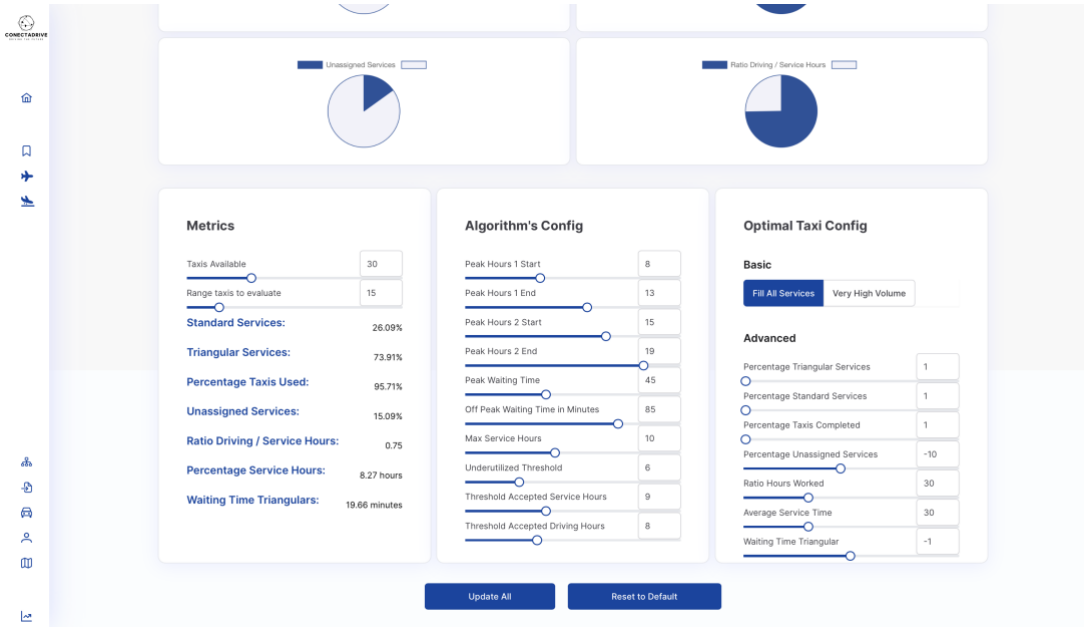

Figure 18 - Configuration Dashboard

5.4.7 Interaction and Update

When the user makes changes to the configuration, these changes are processed and a new backend call is made to recalculate the service mappings. Updated results are automatically displayed on the frontend.

The data table also allows the selection of multiple taxis and the management of unassigned services. Unfortunately, it has not been possible to implement a CRUD[11] yet for the Datatable, in order to eliminate services and have them recalculated with the algorithm with the new combinations.

[11] CREATE READ UPDATE DELETE

5.5 Study and creation of a flight delay predictor

5.5.1 Introduction

This part of the project has constituted a third of the working time dedicated to the company. The flight delay predictor is a key element in determining which flights are not usually punctual, an extremely common problem at airports. This problem also seriously affects taxi companies, since passengers who arrive at the airport and need to be transported to a hotel depend entirely on the arrival time of their flight. The uncertainty about the exact arrival time of the planes generates great planning difficulties.

During high season, when more than 600 services can be performed in one day, about half of these are "tickets", that is, transfers from the airport to a hotel. Having an accurate prediction of flight delays would optimize taxi management, making it possible to make additional trips instead of leaving taxis waiting for customers who will still be late to arrive.

5.5.2 Flight Delay Predictor Study

This part will briefly explain the study that was carried out to develop a predictor of flight delays. This predictor is essential to improve the planning and management of transport services, providing accurate information about possible flight delays.

A member of the team used a public flight API from Spain to obtain all the necessary information about air traffic. One of the initial limitations of the project was to work with only three months of data. This time limitation made it difficult to model data with a lot of variability, since significant information that was not available in that period could be lost.

However, in our case, this limitation was not a significant drawback, since the model had a dominant bias. This means that while the data was relevant to model accuracy, the lack of a longer period was not an issue initially. This matter will be explained in more detail in the following steps.

5.5.3 Import of Libraries and Data Upload

The first part of the document consists of importing essential libraries for data analysis and predictive modeling, such as **pandas**, **numpy**, **matplotlib**, **seaborn**, and machine learning tools such as **scikit-learn**. In addition, the historical flight dataset is loaded.

5.5.4 Data preparation

5.5.4.1 Management of Null Values

Null values are treated using techniques such as the elimination of rows or columns with null values, or the imputation of values by statistical methods.

```
Columns date (total 26 columns):
 # Column Non-Null Count Dtype
---  ------                      --------------  -----
 0 ID 24205 Non-Null Object
 1 compañía 24205 non-null object
 2 numvuelo 24205 non-null object
...
 24 poicinta 22098 non-null float64
 25 hora_llegada 24191 Non-Null Object
```

5.5.4.2 Encoding of Categorical Variables

Categorical variables are encoded in numerical format using techniques such as Label Encoding and one-hot coding, allowing Machine Learning models to process this data effectively. Label Encoding assigns a unique numerical value to each category, while the one-hot encoding transforms categories into binary vectors, where each category is represented by a value '1' in the corresponding position and '0' in the others. Although one-hot encoding is simple, it is not recommended to use it when there are many columns, as is our case, as it can significantly increase the dimensionality of the dataset.

It is also important to encode categorical variables because many graphs can only be made with numerical inputs.

```
from sklearn.preprocessing import LabelEncoder
dfe = df.copy()
le = LabelEncoder()

for col in dfe.columns[dfe.dtypes == 'object']:
    dfe[col] = dfe[col].fillna('missing')
    dfe[col] = le.fit_transform(dfe[col])

dfe.info()
```

5.5.4.3 Data Normalization

Data is normalized to ensure that all variables have a comparable range. This practice is mandatory depending on the algorithm you want to use (Trees, Neural Networks ...) but generally it is good practice to carry it out.

5.5.5 Exploratory Data Analysis (EDA)

5.5.5.1 Column Treatment

A correlation analysis was performed between the columns to identify those with high correlation. Columns with high correlation were removed, as the presence of highly correlated variables can introduce multicolinearity and negatively affect model performance. In addition, columns considered irrelevant for predicting flight delays were deleted, thus optimizing the quality of the data used for modeling.

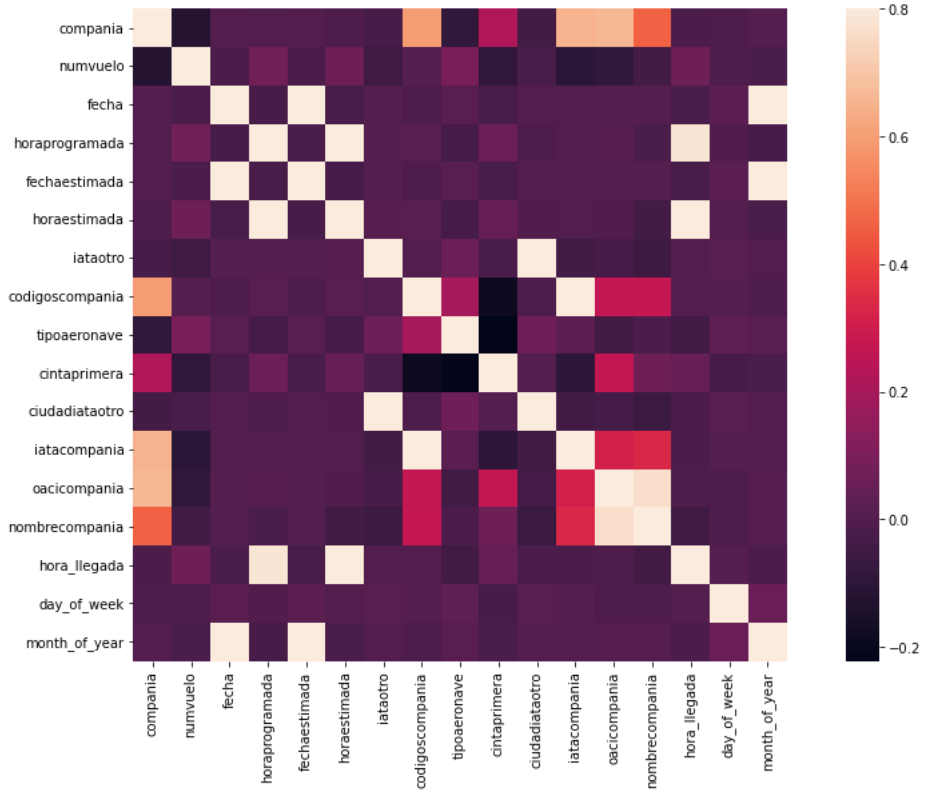

Figure 19 - Correlation of columns

5.5.5.2 Data Visualization

Multiple visualizations are used to explore the data and identify patterns or anomalies. This includes scatter plots, histograms, box and whisker plots, and heat maps to show correlations between variables. These visualizations help detect important trends and relationships between attributes.

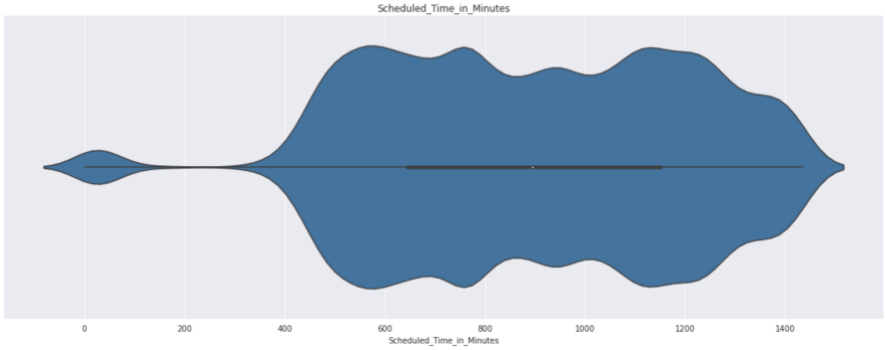

Figure 20 - Violin distribution

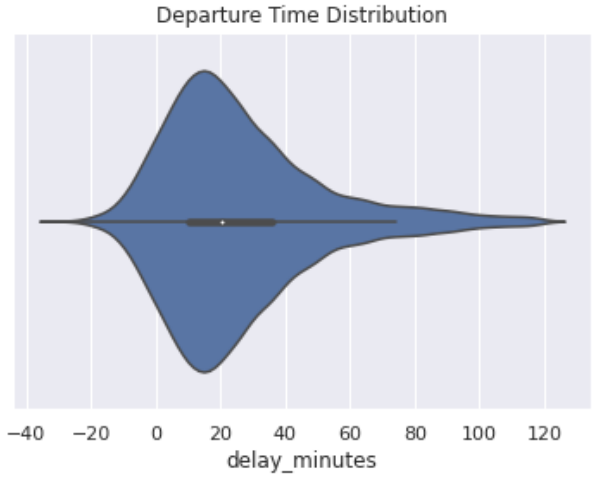

Figure 21 - Distribution delays

5.5.5.3 Feature Enigneering

Different new columns are created to try to capture nuances of the data for a better modeling. For example, assign a type of delay based on minutes late.

```
    dfe['DELAY_CATEGORY'] = None
    dfe.loc[dfe['delay_minutes'] <= 0, 'DELAY_CATEGORY'] =
'OnTime_Early'
    dfe.loc[(dfe['delay_minutes'] > 0) & (dfe['delay_minutes'] <= 15),
'DELAY_CATEGORY'] = 'Small_Delay'
    dfe.loc[(dfe['delay_minutes'] > 15) & (dfe['delay_minutes'] <= 45),
'DELAY_CATEGORY'] = 'Medium_Delay'
    dfe.loc[dfe['delay_minutes'] > 45, 'DELAY_CATEGORY'] = 'Large_Delay'
```

And below is the diagram of delays:

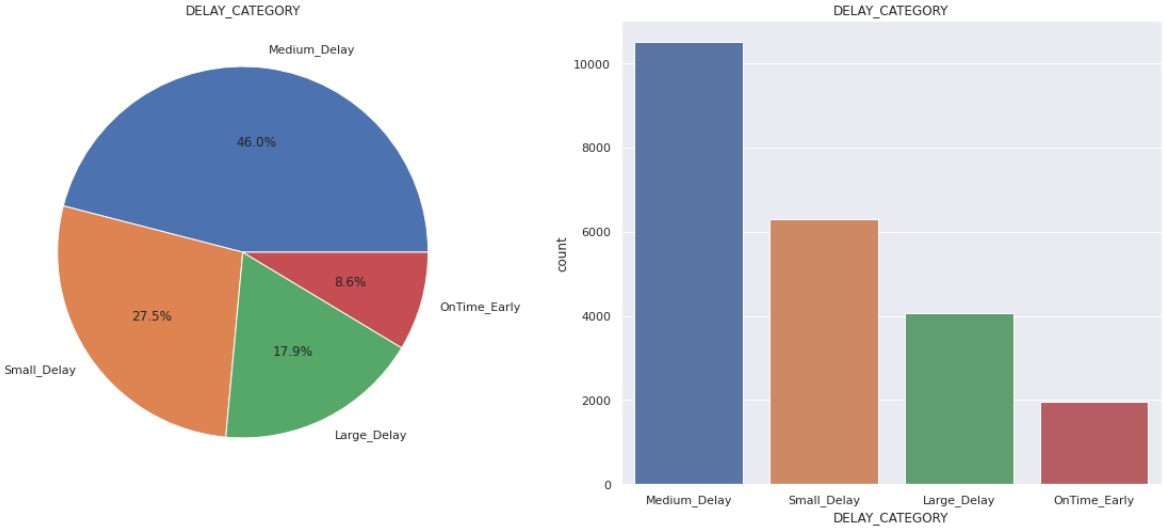

Figure 22 - Types of delay

How the flight company influences was also studied.

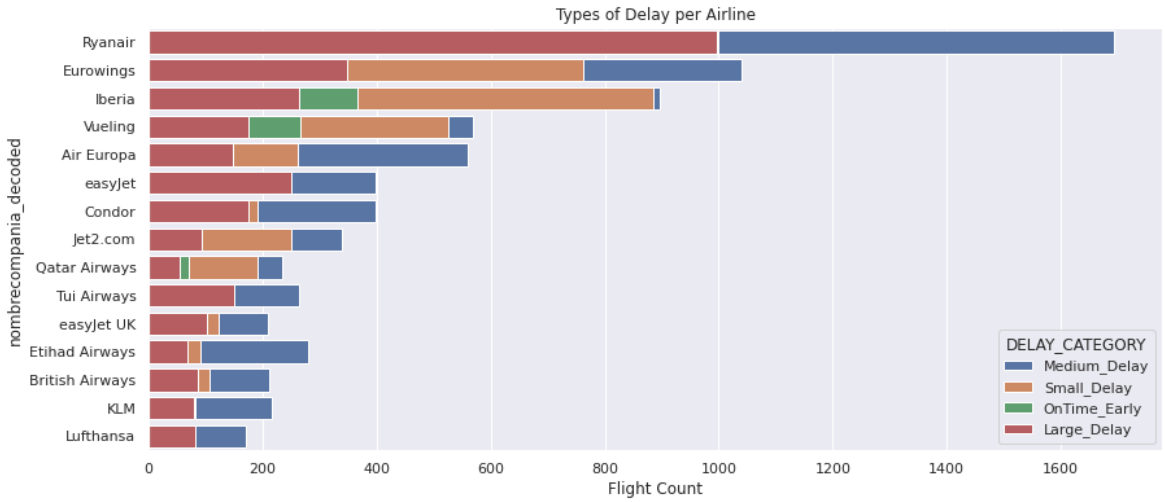

Figure 23 - Delay for company

The first impression they had of the EDA results was that the delay was considerable. Initially the estimate would have been that the vast majority of flights arrived on time, at an interval of 10 minutes, but the data showed that flights generally arrived 20 minutes late. After consulting my former boss about the problem, he told me that it could be a daily distribution, that flights were frequently delayed.

5.5.6 Classification of Delays

First, a classification model is built to determine whether a flight will be delayed or not. Several models were trained to determine which were more effective:

- **Logistic Regression:** Used to model the probability of binary classes, in this case, whether a flight will be delayed or not.
- **Decision Tree:** A model that iteratively divides the dataset into subsets based on the feature that provides the most information.
- **Gradient Boosting:** An assembly technique that combines multiple weak models (usually decision trees) to create a strong model, progressively improving accuracy.
- **Random Forest:** Another assembly method that uses multiple decision trees trained with random subsets of data to improve robustness and accuracy.

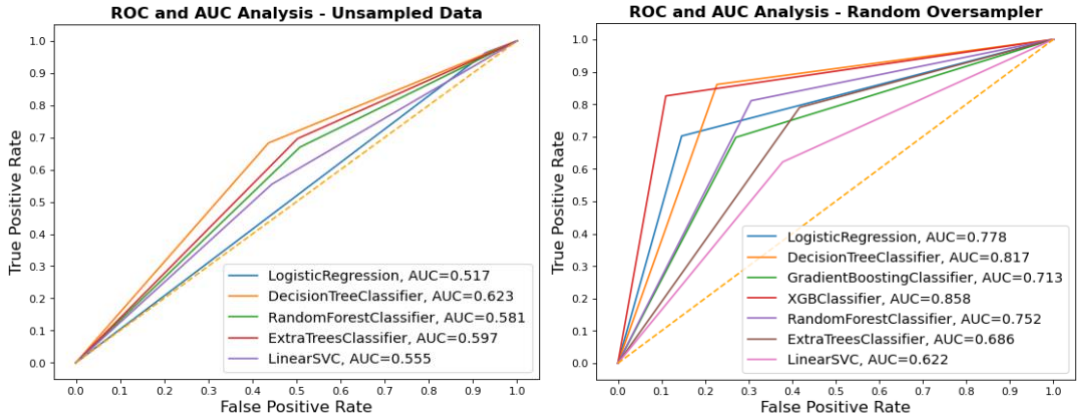

Figure 24 - Performance Classification

The data is divided into training and test sets, using a ratio of 70% for training and 30% for testing. This split allows models to be trained with most of the available data, while a meaningful test set is used to evaluate model performance on unseen data. This helps prevent overtuning and ensure that models generalize well to new data.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.drop(["delay", "delay_minutes"], axis = 1)
y = df[["delay"]] #variable that we want to predict
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.1, random_state = 3)
X_train.head()

scaler = StandardScaler()
X_train= scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

To ensure the robustness of the models, cross-validation techniques are also applied. This technique involves dividing the training set into multiple subsets and training the model several times, each time with a different subset as a validation set and the rest as a training set.

Rating models are compared using various evaluation metrics to determine their performance:

	Precision	Recall	F1-Score	Support
0	0.68	0.69	0.69	827
1	0.82	0.81	0.82	1438
accuracy			0.77	2265
Macro AVG	0.75	0.75	0.75	2265
weighted avg	0.77	0.77	0.77	2265

Balanced Accuracy (Test Set): 0.752

F1-Score (Test Set): 0.816

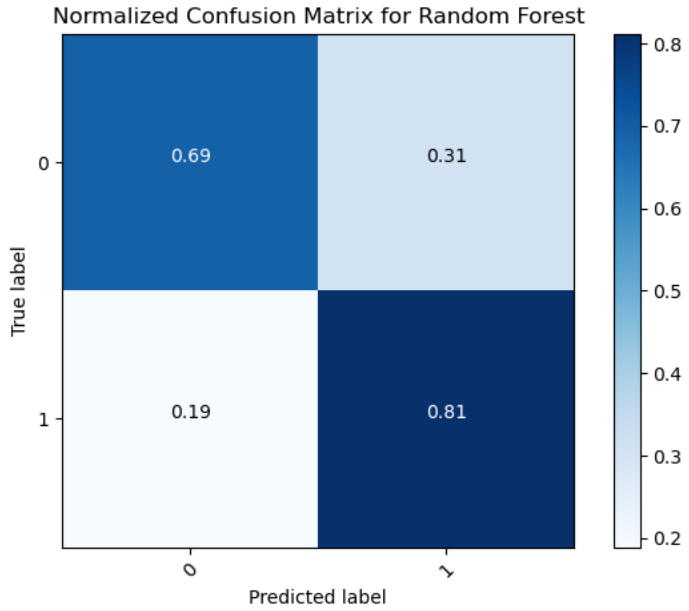

Figure 25 - Example of confusion matrix

- **Accuracy:** The ratio of correct predictions to total predictions.
- **Accuracy:** The ratio of true positives to total positive predictions.
- **Sensitivity (Recall):** The proportion of true positives over the total positive real cases.
- **Area under the ROC curve[12] (AUC[13]):** Measures the ability of the model to distinguish between classes. (Shown 2 pages ago)

[12] Reciever Operator Characteristc

[13] Area Under the Curve

5.5.7 Delay Regressor

For regression, only flights that have suffered a delay are considered. The original DataFrame is filtered, selecting delay_minutes as the dependent variable and removing it from the independent variables. The data is divided into training sets (90%) and test sets (10%) to ensure a good evaluation of the model, since this time we have even less data.

```
    # Only Delayed Flights are Considered
    # Filtering Data for Regression Training and TestingLabels_Train
    df = df[df["delay"] == 1]
    df.reset_index(inplace=True, drop=True)
    Labels = np.asarray(df["delay_minutes"])
    Features = df.drop(columns = ['delay_minutes', 'delay'])
    Features_Train, Features_Test, Labels_Train, Labels_Test =
train_test_split(Features, Labels, test_size=0.20, train_size=0.8,
random_state=42)
    of Features
    of Labels
```

Several models are developed:

- **Linear Regression:** A simple model that assumes a linear relationship between the independent variables and the dependent variable (duration of the delay).
- **Decision Tree for Regression:** A model that divides the dataset into subsets based on the feature that minimizes prediction error.
- **Random Forest for Regression:** An assembly method that uses multiple decision trees to reduce prediction variance.
- **Gradient Boosting for Regression:** An assembly technique that builds a strong model by combining multiple weak models, progressively improving the prediction error.

The metrics used to evaluate these regression models include:

- **Mean Squared Error:** The mean of the squares of errors, which heavily penalizes large errors.
- **Mean Absolute Error:** The average of the absolute values of errors, which provides a measure of the average error.
- **R^2 (R2 Score):** A measure of the proportion of the variation of the dependent variable that is explained by the independent variables.

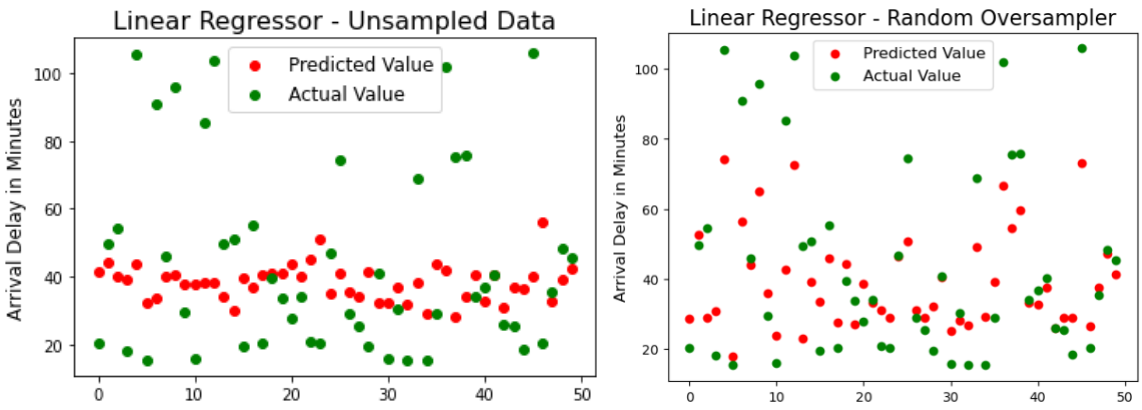

Figure 26 - Regression results comparison

Both models optimize the result after being evaluated to improve their performance. For each model, a search is carried out for the best hyperparameters using techniques such as Grid Search or random search. This optimization allows you to find the settings that maximize your model's performance metrics.

5.5.8 Differences from oversampled data

This whole modeling process mentioned above has been carried out with different data. In the first case, it would be the data with the distribution of standard flights, with the heavy tail. This means that the data is worked as it is, without adjusting the imbalance in the lag categories. In this context, classification and regression models are constructed following the standard steps described above.

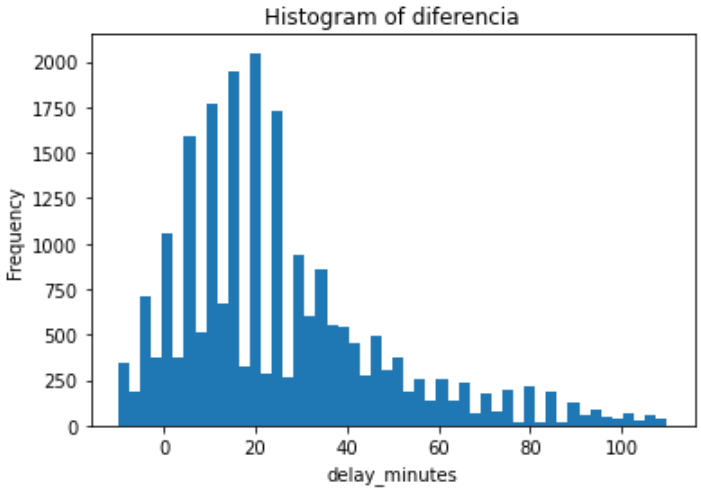

Figure 27 - Heavy tail

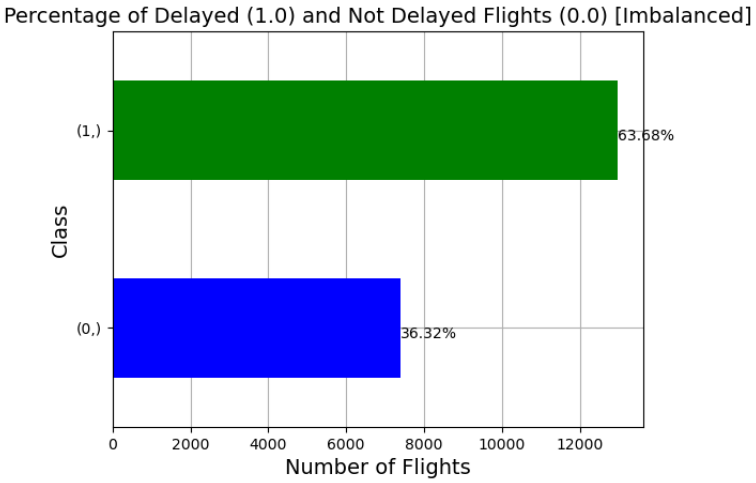

Figure 28 - Distribution before oversampling

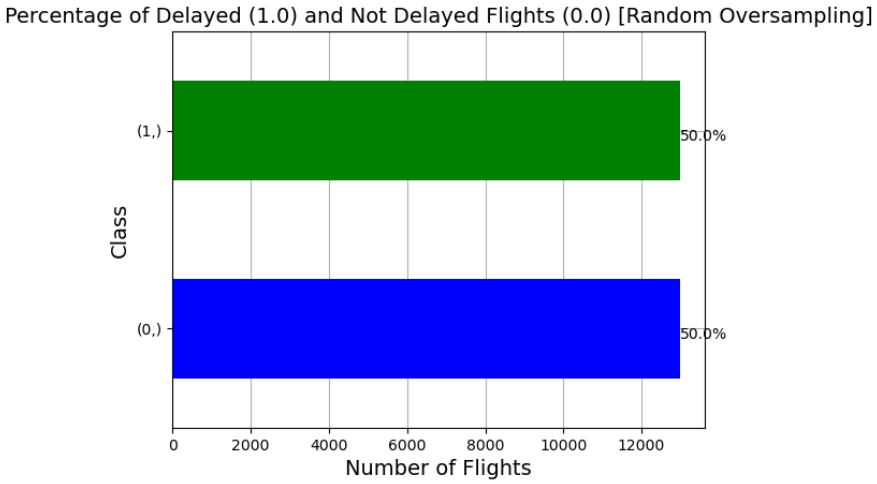

Figure 29 - Distribution after oversampling

Ranking metrics such as accuracy, accuracy, and sensitivity show poor performance compared to models trained with balanced data. For regression models, metrics such as mean quadratic error, mean absolute error, and R^2 serve as the basis for comparison with improved models of the second notebook.

In the file where there is oversampling, data balancing techniques such as RandomOverSampler and SMOTE are used[14] to treat debalancing in delay categories. These techniques create new instances for minority classes, ensuring that the model has an equitable representation of all categories.

[14] Synthetic Minority Oversampling Technique

Data splitting and cross-validation are performed similarly, but balanced data offers better representation of minority categories.

Ranking metrics are significantly better thanks to the use of balanced data. For example, the accuracy of the Random Forest Classifier increased to about 89% after data balancing. Regression models also show notable improvement. With the addition of the XGBRegressor model, the Random Forest Regressor's R2 Score increased to 0.85, indicating a better ability of the model to explain variance in the data.

5.5.9 Conclusions

The most difficult part of the study was done, and the next task would be to create all the infrastructure in the cloud. Some of this infrastructure had already been developed by another member of the team. With an endpoint from the algorithm code, scheduled same-day delays could be called every day. This operation could be calculated at zero hours at the beginning of each day in order to increase the hours of the incoming service and make taxi planning more accurate.

Another drawback that was later found was that the nature of the data was somehow distorted, as flights were delayed more than 10 minutes than they should average. It was discovered that when saving the data in the database, a parameter was being saved that was later than indicated. In order not to go into technical details of an airport, it could be said that instead of starting counting from when the plane landed, it began to count from when the finger had been coupled for passengers to leave. Despite this error, the distribution of the arrival of flights remains the same, but all shifted to the left, having fewer minutes of delay on average.

6 Evaluation

6.1 Testing

To ensure that the developed algorithm works correctly and avoids service overlaps, we have used the **pytest** library to carry out exhaustive tests. Pytest is a widely used tool for testing in Python, which facilitates the creation and execution of automated tests. This library allows us to define specific test cases and verify that our code meets the expected requirements.

The tests developed focused mainly on ensuring that the services did not overlap with each other. This was one of the most critical features of the application, as service overlap could cause serious problems in taxi planning. In addition, these tests were very useful during development, making it possible to verify that each change in the code did not adversely affect the logical operations related to the insertion of services in the taxi drivers' schedule.

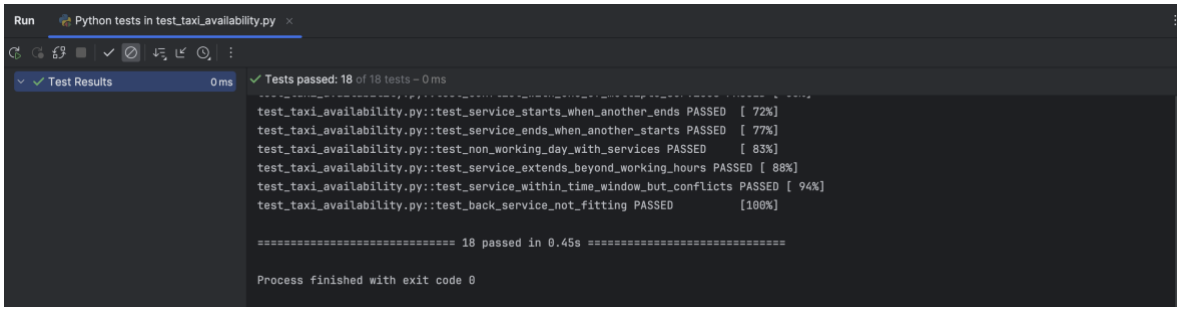

Figure 30 - Running the test script

6.1.1 Using Pytest

Pytest is a library that allows you to define test functions with asserts that check if the code outputs are as expected. In the project, **pytest** was used to define several test cases covering different scenarios of assigning services to taxi drivers.

6.1.2 Objectives of the Tests

The tests were designed to cover a variety of situations, including:

- Allocation of services that begin before, during and after taxi working hours.
- Verification that services do not overlap with existing services.
- Verification of the correct assignment of consecutive services.
- Management of services on non-working days.

6.1.3 Benefits of Automated Testing

Automated tests with **pytest** have several advantages:

- **Early detection of errors:** They allow to identify and correct errors in the initial stages of development.
- **Safety in future changes:** They ensure that modifications do not introduce new regressions or errors.
- **Documentation:** They serve as living documentation of the expected functionalities of the code.
- **Ease of use: pytest** is simple to use and can run thousands of tests in a few seconds, providing detailed reports of results.

6.2 Code Optimization and Profiler Use

In this project, it is essential that the taxi planning algorithm is as optimal as possible. This algorithm must be lightweight and fast so that it can be scalable and executed multiple times throughout the day. This is vital to ensure that changes are instantaneous and that the algorithm can effectively manage dynamic scheduling needs. To achieve this goal, we have used various code optimization techniques throughout development.

6.2.1 Using the Profiler

One of the optimization techniques we have used is the profiler. A profiler is a tool that allows you to measure code efficiency in terms of runtime and memory usage. It identifies the parts of code that consume the most resources and helps detect bottlenecks. This is crucial for optimizing the algorithm, as it allows you to focus on sections of the code that need improvement.

6.2.2 Optimization of Data Structures

In addition to the use of the profiler, we have also applied various techniques to optimize data structures:

6.2.2.1 Using Numpy for Numerical Calculations:

We have used Numpy to manage numerical calculations as it is much more efficient than using Python lists. This library optimizes mathematical operations and reduces execution time.

6.2.2.2 Memorization of Functions:

We have applied the memoization technique to avoid recalculating repeated results. This significantly reduces runtime for functions that are frequently called with the same arguments, saving time and resources.

6.2.2.3 Effective Management of Services:

To optimize planning, we have ordered services according to their priority during peak hours. This allows services to be allocated more efficiently and reduces waiting time, improving the overall use of resources.

6.2.3 Key Functions for Optimization

During the profiling process, it was identified that the code invested a lot of time in loops that calculated the service times for each taxi driver and service. To address this issue, two key features were implemented: **calculate_and_store_service_times** and **calculate_taxi_hours_service**.

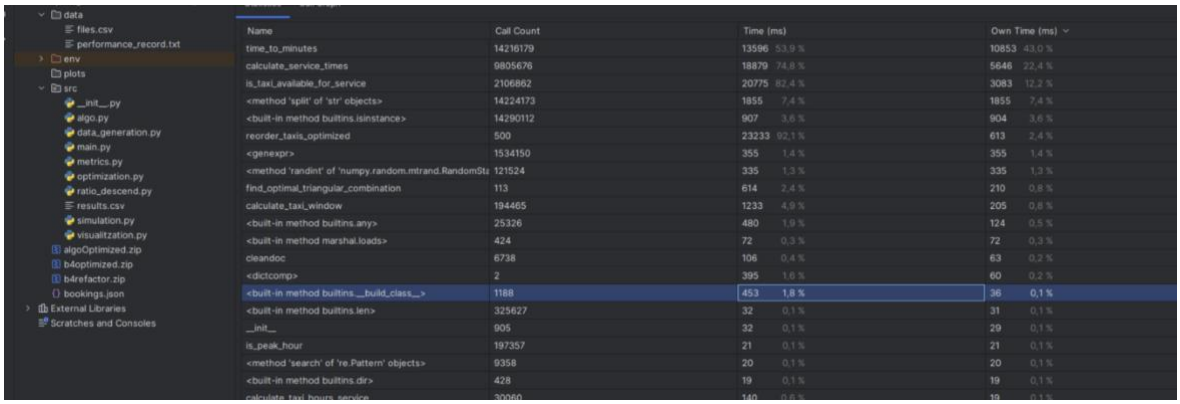

Figure 31 - Profiler trim

- **calculate_and_store_service_times**: This function calculates and stores service times for each service. Instead of recalculating these times repeatedly within loops, this function precomputes and saves them, significantly reducing processing time.
- **calculate_taxi_hours_service**: This function calculates the total hours of service for each taxi. It optimizes the process by adding service times efficiently, avoiding unnecessary recalculations and improving the overall speed of the algorithm.

7 Project Conclusions

If I had to summarize the most important attribute or learning drawn from this experience, it would be tenacity and character to face difficult problems. Skills can be acquired, and with enough time anyone can learn anything, but the character needed to face a difficulty is a more complex task, which I have had to test many times throughout this project.

From the beginning, facing the challenge of developing a flight delay predictor was a really complex task, considering that my knowledge of machine learning and machine learning was relatively limited, since I had only taken a couple of online courses. Still, I decided to be self-taught to catch up to the problem. The learning curve was difficult at first, but the best way to learn something is by doing it in practice. Above all, understanding the nature of the data (flight delay bias) was key to finding an effective solution.

When this part of the project was almost finished, my boss asked someone to get to work on the algorithm for scheduling the services. Since the rest of my colleagues already had other tasks assigned, I had to start from scratch in this new task. This second challenge was even more difficult at first, because designing an algorithm from scratch, with all the complications and minutiae that this entails, was very challenging. However, little by little I progressed, first grouping services that could be combined, then trying to make combinations and groupings of services, among others.

There were weeks of development that were extremely frustrating, in which nothing code was advanced, but time was spent debugging line by line, which was very wearing and frustrating until the problem was found. Even so, the satisfaction of this last project has been the best part of everything. Seeing how I was able to build a system that solves a problem and that was gradually taking shape, with positive feedback from my former boss, was very encouraging.

Looking ahead, I'd like to narrow down the essence of the program so I can create an API that other people who need to plan services can use. The truth is that seeing this project with no way out, taking into account the energy invested in it and seeing that it is currently not solving any problem, when it is really a fully functional application, generates a certain pity.

Finally, I can say that this has been one of the most enriching experiences professionally, both in terms of hard skills and soft skills. This experience and opportunity would not have been possible without the good friendships and support I have received along the way.

8 Resources used

8.1 Bibliography:

1. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Aurélien Géron.
2. *Python Data Science Handbook* by Jake VanderPlas.
3. *Machine Learning Yearning* by Andrew Ng.

8.2 Websites:

1. Scikit-Learn Documentation - Official documentation of the Scikit-Learn library, includes tutorials, user guides and API reference.
2. TensorFlow Documentation - Official TensorFlow documentation, including guides, tutorials and API references for machine learning and neural networks.
3. Keras Documentation - Keras official documentation, includes tutorials for building and training neural network models.
4. Pandas Documentation – Official Pandas documentation, includes user guides, tutorials and API references for data manipulation and analysis.
5. Matplotlib Documentation - Matplotlib official documentation, includes tutorials, examples and API references for data visualization.
6. NumPy Documentation - Official NumPy documentation, includes user guides and API references for numerical and mathematical operations.
7. Pytest Documentation - Official Pytest documentation, includes guides for creating and running unit tests.
8. Joblib Documentation - Official Joblib documentation, includes guides for parallelization and function cache.
9. Seaborn Documentation - Official Seaborn documentation, library for statistical data visualization based on Matplotlib.
10. Plotly Documentation - Official documentation of Plotly, library for the creation of interactive graphics.
11. Statsmodels Documentation - Official documentation of Statsmodels, library for statistical models in Python.
12. SciPy Documentation - Official documentation of SciPy, library for scientific computing and advanced numerical techniques.
13. Jupyter Documentation - Official Jupyter Notebook documentation, includes guides for creating and sharing documents containing live code, equations, visualizations and explanatory text.
14. OpenAI GPT-3 Documentation - Official OpenAI API documentation, including guides and tutorials for GPT-3 model integration.
15. Flask Documentation - Official documentation for Flask, a microframework for Python web applications.

8.3 Software:

1. Python 3.x - Programming language used to develop the project.
2. Jupyter Notebook - Interactive environment to develop and share code in Python.
3. PyCharm – Integrated development environment (IDE) used to write and debug Python code.
4. Git – Version control system used to manage project source code.
5. Docker – Platform used for creating, deploying and executing applications within containers.

8.4 Hardware:

1. MacBook Pro M3.
2. HP Laptop with Intel Core i7 processor, 16GB of RAM, 512GB SSD.

8.5 Other resources:

1. Google Colab - Cloud computing platform for creating and running Jupyter notebooks.
2. AWS (Amazon Web Services) - Cloud infrastructure used for model training and data storage.
3. arXiv - Machine learning papers and research articles available on arXiv.

8.6 Online courses:

1. Machine Learning - Coursera, taught by Andrew Ng, Stanford University.
2. Deep Learning Specialization - Coursera, taught by Andrew Ng and his team, Stanford University.
3. AI For Everyone - Coursera, taught by Andrew Ng, Stanford University.