Digital Travellers

An application for recurrent travellers

Miquel de Domingo i Giralt

TBD

Contents

List of Figures

List of Tables

1. Introduction

1.1 Introduction

Recurrent Travel is a master's thesis for the Architecture and Design of Software of the La Salle URL university.
The project is being developed in the context of the increasing demand for air transportation and the digitalization of the travel industry. This project aims to offer an innovative solution for those users who frequently travel for work or personal reasons, through the creation of an application that allows them to be notified of opportunities on their regular routes.

This masters's thesis is framed in the ambit of project planning and management, software design and clean software architectures, databases, testing, user interface design and user experience.

1.2 Context

Throughout the past decades, the European air transportation sector has undergone an unprecedented growth, primarily propelled by the democratization of pricing and the digitization of airlines. This expansion has resulted in a noticeable increase in the number of users who utilize both domestic and international flights.

In 2019, the volume of work flights increased to 1.28 billion dollars[1] and, even with the COVID-19 pandemic, the volume kept going up[2]. Business trips are essential for both interpersonal connection development and building trust between different companies[3]. Business trips are often conducted to visit the client, with over 44% of the business trips in Europe being made for client meetings and 32% for visiting the company offices in other cities[4]. Moreover, 30% of business travelers in Europe travel once a month, 62% once a year, and 5% between 21 and 40 times a year[4].

Group business trips are also a commonality, with over 50% of such being made by two or more workers[4]. When choosing a flight, 26% of the travelers in Europe will consider direct connections as the most important aspect of a flight, 19% will consider the price of the flight, 23% schedule coordination, and airport location 20%[4].

Taking into account the data before the COVID-19 pandemic, it is starting to be more common that employees are responsible of the booking of their own travels, with up to a 59% of them travelling in the United States, being also responsible to book the hotel[5]. Around the 69% of the travelers states that they book all the reservations, regardless of the type. Additionally, 79% of the business traveler have booked a trip through their mobile devices, indicating the increased use of mobile technology for this purpose[6].

In the post-pandemic work (2023), commercial aviation continues to be efficient, resilient and a key component to the development of the modern world. According to the annual report by IATA[7], it is expected that by the end of 2023, all regions will have surpassed pre-pandemic flight demand. Additionally, the charts how an average positive variation of 5.4% points in real GDP in the coming years.

- For 2023, it is projected that the number of passengers will exceed pre-COVID-19 levels, reaching up to 105%.
- By 2030, it is estimated that the number of global passengers will grow to 5.6 billion.

Consequently, medium-term growth forecasts for commercial aviation indicate a promising and expansive future for the sector.

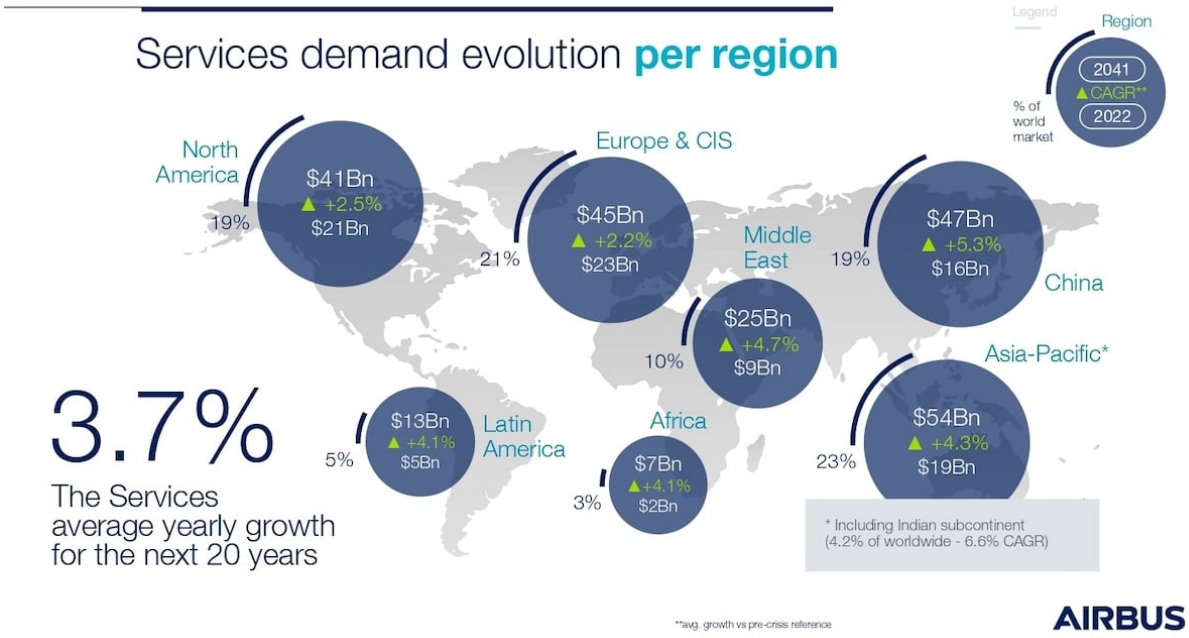

Figure 1.1: Evolution of flight demand of Airbus flights in the pre-pandemic world vs the post-pandemic.

Within this context of growth, it has appeared another type of user: recurrent travelers. This travelers, whether due to work, family or personal reasons, have incorporated the use of planes into their routine periodically throughout the year.

Recurrent trips are characterized by sharing the same point of origin and destination, similar departure and arrival times on all routes, and a stable repetition frequency throughout the year. Although airline digitalization and the ease of purchasing plane tickets online have greatly simplified the process, recurrent travelers still face difficulties when it comes to finding opportunities and optimizing their itineraries. The lack of specific solutions for this group of users has generated a growing demand for an application that notifies them of opportunities on their recurrent trips, facilitating the planning and management of their habitual travels.

Given the importance and frequency of business travel and the growing trend of employees taking care of their own reservations, it is crucial to address the callenges faced by these recurrent travelers.

1.3 Solution proposal

Throughout the project, we will focus on critical topis in software engineering. The hexagonal architecture will be addressed due to its ability to facilitat the development maintenance, and modification of software by separating responsibilities into logical layers, thus promoting efficient code maintenance for the product.

Domain-Drive Design (DDD) is an architecture pattern that emphasizes in solving specific business problems, improving communication and the efficiency of the software model. Applying DDD will help us create software that is more understandable and effective by focusing on the different domain concepts of Recurrent Travel.

Additionaly, the SOLID principles will be used ofr designing a modular, understandable, and extensible software. These principles are essential for establishing quality and maintainability

standards in software component design. It also ensures the creation of a robust, reliable, and easily maintainable solution.

To ensure reliability and proper functioning of our application, testing and validation techniques will be implemented.. Continuous Integration (CI) and Continuous Development (CD) will also be taken into account as part of the development cylce. Automation and version deployment will also take a spot in the cycle.

Furthere, the integration with external systems will also be studied. This can improve the efficiency of our software solution by leveraging the functionality of other existing services in the market.

Finally, the importance of good practices in software development will be strongly emphasized, focusing on resource management, risk mitigation, effective communication, and develivering an mvp that meets customer expectations.
Ultimately, this master's thesis project aims to offer an effective and efficient solution to enhance the experience of recurrent travelers, enabling them to optimize their trips and facilitate the planning and management of their regular travels.

1.4 Topics and concepts

Before moving into the details of the project, it is fundamental to be familiarised with some key concepts regarding software architecture and design, which will be addressed throughout this projet. In this regard, we will distinguish between general concepts, independent of the technology and environment inwhich they will applied, and more specific concepts for the backend and frontend development.

1.4.1 General concepts

Before starting, it has to be taken into account the concepts about software architecture and design in the different applications to develop. As explained in the goals of the project, the software to be developed should be scalable, mantainable and robust.

In first place, it is essencial to consider the meaning of software desing. Software design refers to the creation of a logical and organised structure taking into account from the architecture, to modularity, reusability and readability of the code.

Therefore, as a team we opted to apply hexagonal architecture as the foundation of our applications. Hexagonal architecture is a software pattern whose goal is to separate between business logic of its technical concerns. In order words, it separates the domain of an application from the logic it may have. It also promotes a greater independence between the components and simplifies the adaptation to underlying technological changes. Additionally, it promotes modularity and code reusability. The clear separation of responsibilities of each layer and component, it facilitates the maintenance and evolution of applications over time. Another important aspect about hexagonal architecture is its focus on interfaces and contracts. The definition of such interfaces and contracts ensures a more efficient communication and facilitattes the integration with other services or external systems.
Alongside hexagonal architecture, the team has opted to apply domain driven design in our projects. DDD is a methodology that it is focused in modeling the core business, focusing in specific concepts and rules of the domain. In this project, DDD allows to precisely define entities, value objects and aggregates from the current domain. Furthermore, the domain driven design allows the team to establish a common language between the members of the team, providing a more effective communication and a shared knowledge of functionalities and requirements.

It is also important to note the well-known SOLID principles, which have been used to develop a better software, at a lower level than the aforementioned concepts. Although not strictly an architectural concept, SOLID principles are equally imporant in terms of development, as they promote code modularity, extensibility and maintainability.

Regarding the deployment of the applications that will conform the system, the team has opted for cloud solutions. The usage of a cloud environment offers a set of significant advantagws in terms of scalability, availability and flexibility, which are properties that are wanted for the project.
The cloud, also known as *cloud computing*, refers to the management of the infrastructure and computing services over the internet, instead of hosting the applications in local servers.
Furthermore, deploying the applications in the cloud frees the team from maintaining the underlying infrastructure. The cloud service provided will take care of the server management, networking, and other required resources, allowing the team to focus on the development.

Once the architecture and deployment aspects have been explained, it is crucial to incorporate an element that strengthens the robustness of the system: testing. In this project, two types of tests are distinguised: unit tests and functional or acceptance tests. It is worth mentioning that testing is not limited to this two types, many more types of testing exist, but such will not be covered in this specific project.
On the one hand, unit testing aims to validate the correct functioning of individual units or functional pieces of code. While the main objective is to ensure the expected functioning of such elements, their true value emerges in the future when modifications need to be made to old or legacy code. When making changes to existing code, developers may not take into account all the potential side effects that may arise from such changes. Thus, unit testing provides validation and security to the developer, ensuring that their changes do not break any existing functionality.
On the other hand, it is also essential to include functional tests alongside unit test. These tests aim to validate the overall behaviour of a system by attempting to recreate as close as possible the production environment. The focus relies on component interaction and complete workflows or user flows. Functional tests are particularly useful for detecting potential issues in the integration of different modules or components of the system. By simulating the real usage, it becomes easier to identify problems in communication between components, errors in business logic, or differences in user experience.
The combination of both type of tests, unit and functional, provides a comprehensive testing approach that covers both the internal validation of each component and the overall verification of the system as a whole.

1.4.2 Backend concepts

The backend for the Recurrent travel appliction is the core of data processing in order to simplify the user experience in terms of finding travels. This application layer it is crucial for its functionality and system performance. However, it is not direclty exposed to the client, meaning that it will require of a frontend.
When developing the frontend, it is really important to keep scalability and performance as the two main factors of the applications.

1.4.2.1 Gradle

Gradle is an advanced and flexible build automation system. Unlike other systems, it uses a Groovy-based language for its configuration file, which is more readable and easier to understand.

- Dependency Management: It allows for efficient and flexible dependency management, which is vital in a project of this scale.
- Automation: It is used to automate the build and deployment process, making updates easier and ensuring that all parts of the system are synchronized.
- Versatility: It supports both Java-based projects and other languages like Kotlin, making it extremely flexible for any language change or future additions. This ensures that this project has the freedom to adapt and evolve over time.
- Performance: It is designed to perform well in large-scale projects, thanks to features like caching and incremental execution. This means that only the parts of the project that have changed since the last build will be compiled, saving time and resources. This is especially valuable for this project, which can grow in complexity over time.
- Integration with IDEs and CI/CD: It integrates well with popular integrated development environments (IDEs) like IntelliJ IDEA and Eclipse. This means that developers can use and manage Gradle directly from their IDE. It also integrates with continuous integration/delivery (CI/CD) tools like Jenkins and Travis CI, facilitating automatic deployment and updates.

Therefore, the choice of Gradle for Recurrent travel not only facilitates dependency management and project automation but also provides a solid and flexible foundation for the project's evolution and growth in the future.

1.4.2.2 Java

Java is a widely used programming language known for its portability, efficiency, and support for object-oriented programming.

- Portability: Java is platform-independent, allowing the backend code to run on any operating system that has the Java Runtime Environment (JRE) installed, facilitating deployment and scalability.
- Maturity and Community Support: It has been a staple in the software industry for over two decades, which means it has a wealth of libraries, frameworks, and community support.
- Reliability and Stability: Java has a long track record of reliability and stability, making it a safe choice for critical and large-scale applications. Its strong exception handling system and automatic garbage collector help minimize errors and prevent memory leaks, improving overall application robustness.
- Security: Java was designed with security in mind. Its security features include a strong type system, bytecode verification before execution, and absence of pointers, helping prevent common errors that can lead to security vulnerabilities.
- Support for Enterprise Development: It provides a rich ecosystem of frameworks and libraries for enterprise development, such as Spring, Hibernate, and Apache Camel, among others. These tools simplify the implementation of a variety of backend functionalities, from database access and security to integration with other systems and services.

Therefore, choosing Java as the programming language for the backend of Recurrent travel provides a solid and robust foundation for building a secure, scalable, and high-performance application.

1.4.2.3 Spring boot

Spring Boot is a framework that simplifies the configuration and deployment of Spring-based applications.

- Rapid Development: It provides default configuration that accelerates development by eliminating the need for extensive manual configuration.
- Integration: It integrates with many tools and libraries, such as Hibernate for data persistence and Spring Security for security, making it easy to implement these features.
- Integration with DevOps: It integrates well with DevOps practices and tools. For example, it integrates with Jenkins for continuous integration and with Docker for containerization, facilitating development, testing, and deployment of applications.

In summary, choosing Spring Boot for the development of this project enables quick startup, scalability, and easy maintenance. Its wide range of features and integrations facilitate the creation of a high-quality application that meets the demands and expectations of frequent travelers.

1.4.2.4 Unit test

The implementation of testing is a crucial component to ensure software quality and user satisfaction in this project. In the context of unit testing, technologies like JUnit and Mockito are used.

JUnit is the most widely used unit testing framework for Java that helps developers design and execute tests to verify the behavior of small units of code, such as individual methods or classes. In this project, the usage of JUnit provides the following benefits:

- Quality Assurance of Code: By using JUnit for unit testing, it can be ensured that each unit of code functions correctly before integration. This helps detect and correct errors early in the development cycle.
- Facilitates Updates and Maintenance: Unit tests with JUnit facilitate updates and maintenance of the application. Whenever a change is made to the code, unit tests can be executed to ensure that the changes haven't introduced new errors.

Mockito is a popular Java mocking framework used to mimic objects and behaviors of the test class. For the project, Mockito can be used to simulate application logic that is not directly related to the code unit being tested, such as database dependencies. The benefits of Mockito in this context include:

- Isolation of Code Units for Testing: It allows creating simulated objects (mocks) of external dependencies, meaning that unit tests can focus on the code under test without worrying about the behavior of dependencies. This makes the tests more reliable and easier to write.
- Simulation of Diverse Test Scenarios: It is possible to simulate a variety of test scenarios, such as the behavior of the application when an external service fails or returns unexpected data. This helps ensure that Recurrent travel can properly handle these situations in production.

In summary, the combination of JUnit and Mockito enables comprehensive and effective unit testing in "Recurrent Travel," resulting in more reliable and maintainable software.

1.4.3 Frontend concepts

In terms of frontend development, it is important to consider the same concepts mentioned in the first section. Although more details will be covered in future sections, unlike server repositories, the client repository has been developed as a mono-repository. There is a difference between a mono-repository and a multi-repository, such as:

- *Multirepo*. The multirepo approach means to have multiple applications in different repositories. The main benefits of such approach are the fact that teams can separately work in the repository while at the same time the repository is kept smaller and cleaner.
- *Monorepo*. The monorepo approach is the opposite of the multirepo: all the applications are developed from the same repository. Such approach allows maintaining build and deployment patterns altogether. However, application versioning may be harder to manage.

When aiming to develop a client-scalable application while maintaining a single shared domain layer, the mono-repository architecture has been preferred.

1.4.3.1 Nx

One of the main challenges that a team can face when working with a mono-repository is its maintenance. Maintenance includes tasks such as library updates and managing CI/CD pipelines, if they exist.
Fortunately, there are tools that simplify this maintenance, known as build tools. In the JavaScript world, a tool called Nx has gained significant popularity. Translated from its documentation: *Nx is a next-generation build system with first-class support for mono-repositories and powerful integrations.*[8]

The use of such a tool significantly simplifies the maintenance of JavaScript and/or TypeScript mono-repositories. Some of its other strong points are:

1. The same developers of Nx maintain plugins and similar tools that are easily integrated with an existing repository.
2. Instant generation of internal libraries.
3. The ability to run tests and deployments only on the affected parts, that is, on the modified code, allows limiting the necessary work in Continuous Integration (CI) environments. This involves running tests for the modified code and its dependencies.
4. All created libraries and applications include all the necessary dependencies, scripts, and tools for serving, testing, building, and deploying in a streamlined manner.
5. It provides a rich ecosystem of plugins and utilities within the same base library.

Furthermore, it provides extensive support for the most commonly used libraries and frameworks in frontend development, such as NextJs, the chosen library for developing the frontend application.

Figure 1.2: Nx logo

1.4.3.2 TypeScript and JavaScript

Currently, JavaScript is the most widely used language worldwide for many reasons. However, due to its lack of types, some applications become harder to debug and more error-prone. That's one of the reasons why Microsoft developed TypeScript, which is a strict syntactical superset of JavaScript. Code written in TypeScript is transcompiled to JavaScript during the build time.
By using TypeScript, we provide a highly productive environment when developing the different applications. It not only reduces the number of hard-to-detect errors caused by type issues but also provides all the benefits of ECMAScript.

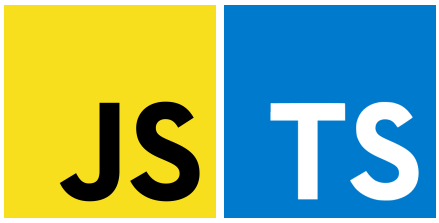

Figure 1.3: JavaScript and TypeScript logos

1.4.3.3 NextJs

As mentioned before, Next.js has been chosen as the front-end framework to build the applications with. This framework is built on top of Node.js, which enables React-based web application functionalities such as server-side rendering (SSR) and static websites. On the Next.js homepage, they provide the following description: *Used by some of the world's largest companies, Next.js enables you to create full-stack Web applications by extending the latest React features, and integrating powerful Rust-based JavaScript tooling for the fastest builds*[9].

Development with Next.js involves structuring the code in a specific way so that the compiler can generate packages in the most efficient manner, avoiding unnecessary code being sent to the client and thus reducing page load time. This feature is known as code-splitting.
React is a framework designed to simplify the construction of web components. As it gained popularity, developers started building full-fledged applications based on React. However, this meant serving a large amount of JS code to the client, which slowed down webpage loading. Next.js automatically solves this problem without the need for any configuration.

Figure 1.4: Next logo

Development with Next.js involves structuring the code in a specific way so that the compiler can generate packages in the most efficient manner, avoiding unnecessary code being sent to the client and thus reducing page load time. This feature is known as code-splitting.
React is a framework designed to simplify the construction of web components. As it gained popularity, developers started building full-fledged applications based on React. However, this meant serving a large amount of JS code to the client, which slowed down webpage loading. Next.js automatically solves this problem without the need for any configuration.

1.4.4 Unit and acceptance testing

The importance of testing has already been mentioned, and the client is no exception. Testing the client brings the same benefits as testing the server, as it is the part that will be used by the end user. Nx provides plugins to enable testing with different libraries.

Figure 1.5: Jest logo

At the unit testing level, both for the domain code and the component code, the Jest library has been used. Jest is a unit testing and mocking library developed and maintained by Facebook. Currently, it is one of the most famous, if not the most famous, libraries for testing. It provides a powerful CLI with the ability to run tests on modified code. It has also gained popularity for its easy integration with all types of repositories, as its configuration is straightforward.
However, Jest does not provide all the tools to, for example, test component code in an isolated environment. In conjunction with Jest, a library has been used that extends Jest's capabilities. This library is *testing-library*, and its specific plugins for React have been used.

Figure 1.6: Cypress logo

At the acceptance testing level, the repository is set up to run acceptance tests using the cypress.io framework. Cypress is more than just a testing tool. It provides a graphical user interface to see what is being tested, where it fails, and other features. It is an extremely powerful tool for UI testing, which has always been a challenging topic. As stated on its website, *Cypress enables you to write faster, easier, and more reliable tests*[10].

1.5 Stakeholders

In the context of the *Recurrent Travel* project, the goal is to simplify the experience of recurrent travelers through an application that notifies them regarding opportunities in their commonly travelled routes. In order to ensure the highest success in the project, it is essential to identify the parties interested in the project, as well as understanding their need.
In the next sections, the interested parties will be identified for the projet. It has been decided to organise the stakeholders in a power/interest model[11] in order to simplify the visualization of the information and simplify the analysis. Such model can be widely found in the definition of engineering requirements.

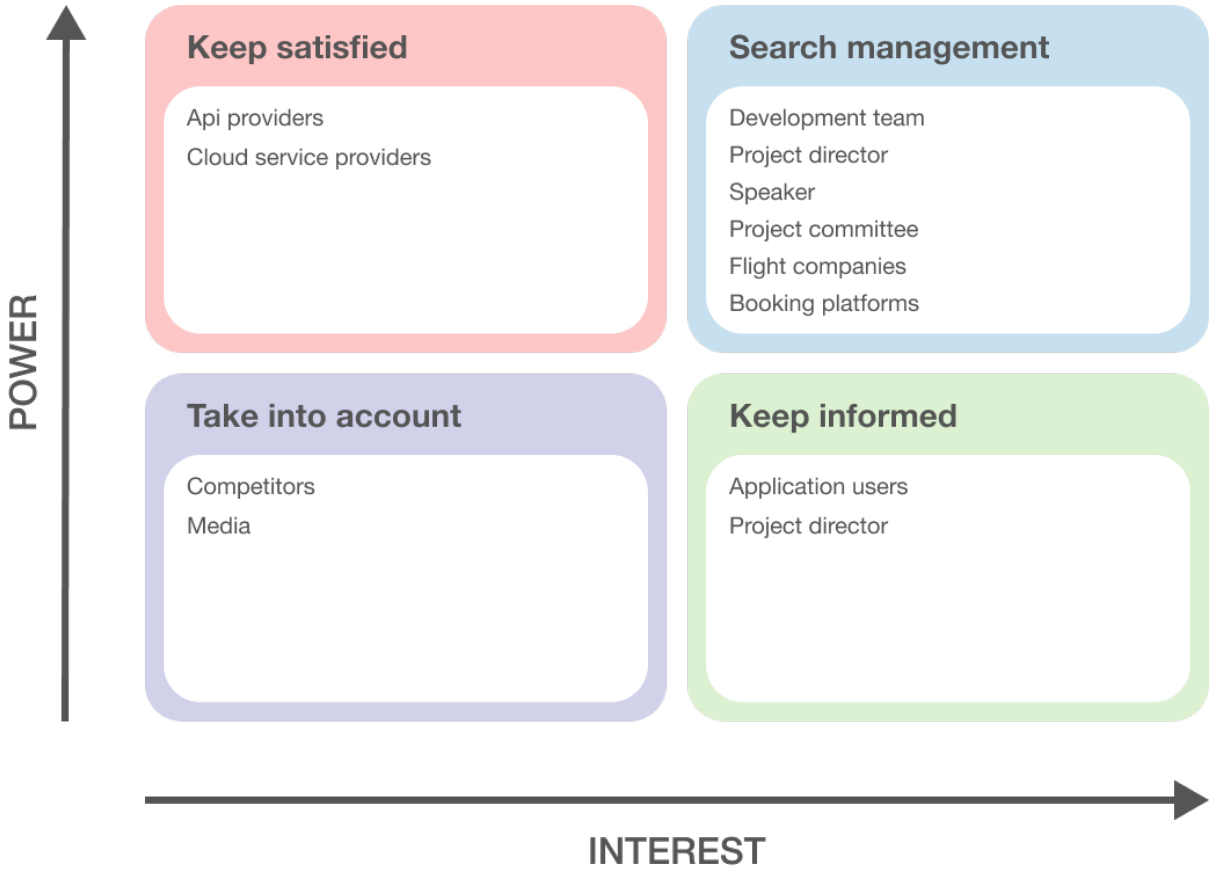

Figure 1.7: Representation of the stakeholders with the power-interest model

- **Lower power and lower interest**
  - **Competitors**: Other companies that provide equal or similar solutions in the market.
  - **Media**: Media that is specialized in the same technology and tourism.
- **Lower power and high interest**
  - **Final users**: Recurrent travelers that will habitually use the application as well as taking advantage of the notifications received.
  - **Families or recurrent travel companies**: Peopl that could be indirectly beneficied from the application as it simplifies the fact of keeping a connection with their relatives, or those who can optimize their business model.
- **High power and lower interest**
  - **API providers and external services**: Companies that will provide with the API servers and services that will be used in the application development, yet they are not directly involved with the project.
- **High power and high interest**
  - **Developers**: The team of 4 developers that will work in the design, implementation and deployment of the application.
  - **Responsable de proyecto**: The person responsible for supervising the project and providing guidance to the development team.
  - **Speaker**: The project's Master's thesis evaluator, who assesses the performance of the development team and the final outcome of the project.

- **Project committee**: La Salle URL faculty members who supervise and evaluate the Master's thesis project.
- **Airlines**: Air transportation companis that could benefit from the market release of the application, due to the utilization of air travel services by the app users.
- **Booking platforms**: Companies that could integrate the application into their services or that the application itself could integrate into their search systems to enhance the experience of their recurring customers.

1.6 Alternatives analysis

In the current landscape, there are other applications and travel-tech companies that aim to address the needs of frequent travelers and optimize the habitual joruneys. The following analysis overviews market alternatives, highlighting their key features and potential differences with the Recurrent travel application:

- **TravelPerk**[12]: TravelPerk is a business travel management platform that offers a solution to plan, book and manage corporate travels. The platform allows companies to centralise the management of employee trips, establishin travel policies, optimize expenses and enhance the traveler's experience. While TravelPerk focuses on the corporate realm and is not specifically designed for recurrent travelers, its cost optimization and travel management offerings could be considered comparable to the ones offered by Recurrent travel.

- **Hopper**[13]: Hopper is an applicatin that uses machine learning algorithms to automatically predict and analise the tendencies of flights and hotels, with the goal of helping users find the best offers. Although Hopper is not focused in recurrent travelers, its focus to price prediction and travel opportunity search could attract users of the Recurrent travel. However, unlike Recurrent travel, Hopper does not allow the possibility to configure customised trips taking into account user preferences of time schedule, frequency and route.

- **Skyscanner**[14]: Skyscanner is a flight, hotel and car renting search engine, which compares prices between the different service providers. Even though Skyscanner is a popular tool to find travel offers, it is not specifically designed to attend the necessities of recurrent travelers, as it does not provide the configuration of time schedule, frequency and rote.

To sum up, although there are several market alternatives in the travel-tech sector, none of them specifically and comprehensively address the needs of recurrent travelers in terms of presonalization, travel oportunity search and cost optimization. Therefore, the proposal of Recurrent travel positions itself as an innovative solution in the actual market, with the potential to significantly enhance the experience of frequent travelers.

1.7 Scope

1.7.1 Goals

The master's thesis project Recurrent travel aims to develop a web application that assists recurrent travelers in optimizing the planning and management of their regular trips, offering a personalized and efficient solution for finding the best travel deals based on their preferences. To achieve this objective, the following sub-objectives have been identified:

- Specific needs of recurrent travelers, particularly those who travel frequently for work, family, or personal reasons, are identified and analyzed.
- The current market of travel applications and services is investigated to determine opportunities and gaps in relation to the needs of frequent travelers.
- An easy-to-use web application is designed and developed, accessible from various devices, allowing users to configure personalized trips by specifying the route, preferred hours, ideal price, and travel frequency.
- Efficient search algorithms and techniques are implemented to find relevant travel opportunities for users, utilizing Crawling/Scrapping techniques and public APIs.
- A notification system is developed to timely and effectively inform users about travel opportunities that match their preferences.
- The quality and maintainability of the developed software are ensured by applying SOLID principles, Domain-Driven Design, hexagonal architecture, and testing and validation techniques.
- The application is integrated with external systems and cloud environments using continuous integration and deployment techniques to increase software efficiency and ensure the rapid and constant delivery of improvements.
- The application is integrated with different flight search platforms like Skyscanner.
- The security and privacy of user data, including preferences, travel history, and personal information, are guaranteed.
- The application is developed within an agile framework, focusing on iterative and continuous value delivery, with defined delivery times to ensure a timely response to business needs and adaptability to changes in requirements.

1.7.2 Functional requirements

The functional requirements describe the specific functionalities that the Recurrent travel application must provide to meet its objectives and satisfy user needs. The main functional requirements of the project are as follows:

1. User authentication: The application must allow users to register and authenticate to access their personal accounts and manage their preferences and recurrent trips.
2. Recurrent travel configuration: Users should be able to configure recurrent trips by specifying the origin and destination, preferred dates, and travel frequency.
3. Flight opportunities search: The application must periodically and automatically search for travel opportunities that match user preferences, using Crawling/Scrapping techniques and/or public APIs.
4. Filtering and result operations: The application must be able to filter and sort search results for travel opportunities, considering the corresponding user and the corresponding alert configuration.
5. Flight opportunity notifications: The application must send notifications to users when travel opportunities that match their preferences are found, either by email or through messaging applications.

6. Travel alerts management: Users should be able to manage received alerts by marking them as read, archiving them, or deleting them, as well as modify their notification preferences.

7. Flight booking: Although it is not necessary for the application to allow direct flight booking, it should facilitate the process by providing links to airline websites or travel providers where users can complete the reservation.

1.7.3 Non-functional requirements

The non-functional requirements are those that describe the quality characteristics of the system, such as usability, performance, security, and scalability, among others. The main non-functional requirements of the Recurrent travel project are as follows:

1. Usability: The application must be easy to use and have an intuitive and appealing interface, with a clear and consistent navigation structure and presentation of information that facilitates user understanding and decision-making.

2. Performance: The application must be capable of performing travel opportunity searches quickly and efficiently, offering results in a reasonable time, even during high demand periods or with a large number of concurrent users.

3. Security: The application must protect users' personal information and preferences by applying appropriate security measures, such as data encryption, authentication and authorization, and prevention of common attacks like SQL injection or cross-site scripting.

4. Availability: The application must be available at all times, ensuring high uptime and the ability to recover quickly from possible failures or service interruptions.

5. Scalability: The application must be able to scale to support the growth in the number of users and resource demands, both in terms of infrastructure and functionalities. This involves the ability to add new servers, load balancing, and improving application performance as necessary.

6. Interoperability: The application must be able to interact and integrate with external systems, such as airline APIs, using common standards and protocols.

7. maintainability: The application's source code must be easy to understand, modify, and maintain, following established design principles and patterns, such as hexagonal architecture, Domain-Driven Design, and SOLID principles. Additionally, version control practices and adequate documentation should be implemented to facilitate collaboration and long-term project maintenance.

8. Privacy and compliance: The application must comply with applicable laws and regulations regarding the protection of personal data and user privacy within the European framework.

1.7.4 Risks and barriers

During the development process of the Recurrent travel project, several challenges and barriers may arise that, if not taken into account, could result in project failure or hinder its success. The following is an analysis of the main challenges and barriers that may be faced:

1. Changes in project requirements: Changes in project requirements may occur throughout the application development, whether they are functional or non-functional, requiring adjustments in the design and implementation. These modifications could delay project progress and increase its complexity and associated costs. To address this risk, it is essential to properly plan and prioritize requirements, maintain constant communication with stakeholders, and be flexible in adapting to changes.

2. External applications integrations: The Recurrent travel application will heavily depend on integration with external APIs and services from airlines and travel providers. Access to these systems may be limited, unstable, or change without prior notice, which would affect the application's functionality and create issues in obtaining up-to-date and accurate information. To mitigate this risk, it is crucial to establish recovery mechanisms in case of failures, perform frequent integration testing, and consider the possibility of using multiple data sources to ensure information availability.

3. Scalability and performance problems: If the application experiences a rapid increase in the number of users and resource demands, scalability and performance issues may arise, negatively impacting the user experience. To address this risk, it is critical to design and implement a scalable architecture, using appropriate techniques and technologies such as cloud computing and load balancing, that allow adaptation to the changing project needs.

4. Privacy and compliance: The Recurrent travel application must comply with applicable laws and regulations regarding the protection of personal data and user privacy, which may pose challenges in implementing security measures and respecting users' rights. To tackle this risk, it is essential to stay informed about applicable regulations and incorporate privacy practices from the design stage, ensuring that the application meets legal requirements and safeguards user information.

5. Time and resource limitations: The master's project has a limited time for completion and may face constraints in terms of available resources, such as the development team's time and allocated budget. These limitations could affect the ability to successfully complete the project within the established timeframe. To address this risk, it is crucial to conduct proper planning, set realistic goals, and prioritize tasks based on their importance and urgency.

6. Market compeitition: The travel and transportation applications market is highly competitive, with numerous players offering similar solutions. This could make it challenging to differentiate "Recurrent Travel" and position it in the market. To tackle this risk, it is essential to identify and communicate the application's competitive advantages, such as its focus on frequent travelers. Additionally, staying informed about market trends and advancements, as well as being willing to adapt and continuously improve the application based on user needs, is crucial.

1.8 Working methodology

In further sections, the working methodology is explained in more depth. Nonetheless, from the early beginning, the team decided to divide the work in responsibilities. This division has allowed the team members, each with different areas of expertise, be able to provide the project with their knowledge in such areas.
There have been bi-weekly meetings, with the goal to share to the team and the team tutor what has been done during the last two weeks, as well as planning what should be done the following. Furthermore, each meeting was an opportunity to discuss with the tutor some of the

project features and functionalities.

In terms of development, th team has organized through the GitHub Projects tool. With this tool, it has been able to plan the tasks in a style that combines elements of Scrum and Kanban. Being integrated with GitHub, it allows for a seamless interactions between repositories and the project. For example, it provides automation for moving tasks from one column to another. Since the team has worked mostly asynchronously, the usage of the board has been crucial to give a view of the project status, as well as identifying what elements were taking too much time, that were blockers of other tasks.

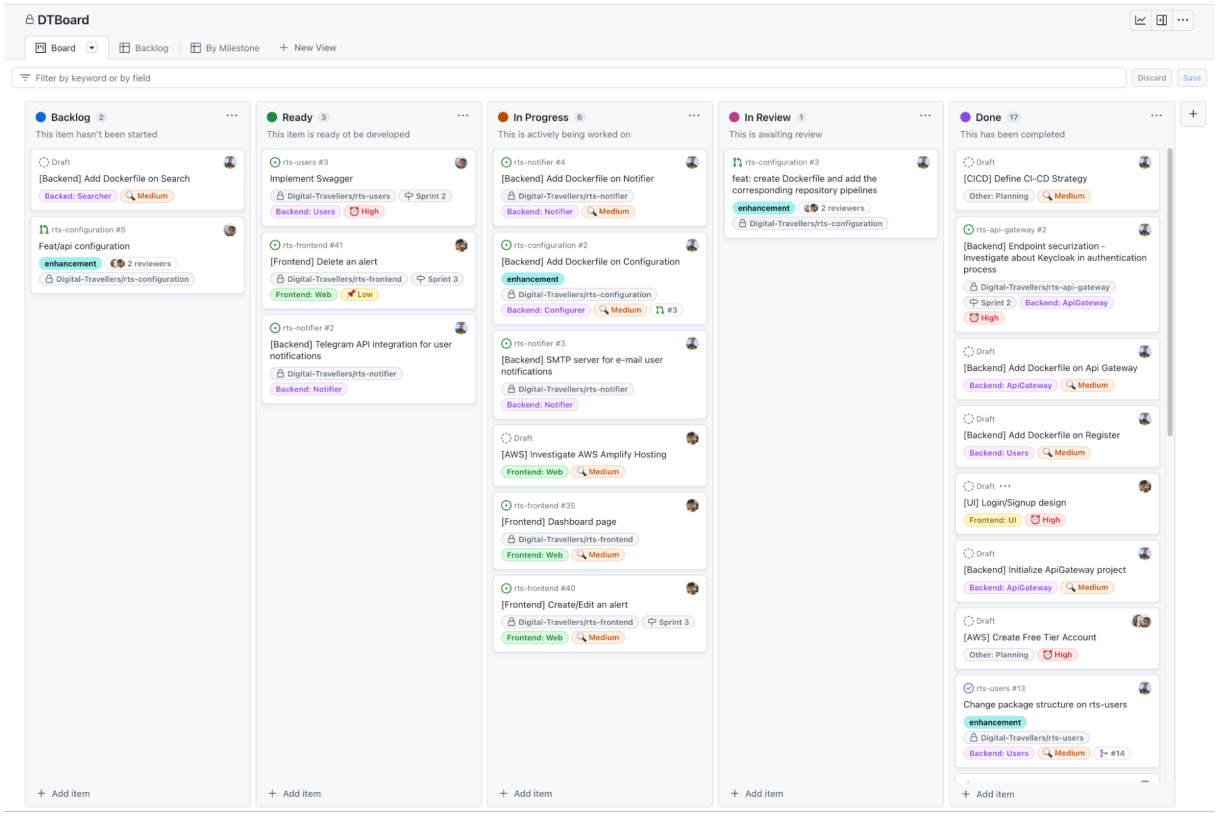

Figure 1.8: Screen capture of the GitHub Projects tool

2. Planning

2.1 Task division

Recurrent Travel is supported by a team of four professionals responsible for the application development. One has expertise in frontend, one in full stack development, and the remaining two have expertise in backend development and devops. Despite each member having specific expertise in their respective fields, the team is cross-functional, and the project development can be carried out by them without the need for external dependencies.
The project consists of the following elements:

- A user interface application to allow the user to interact with the application.
- Four services that include the backend server logic, database management and data scrapping.
- A backend communication system, such as a queue, to allow the different services to communicate between them.

Taking into account this proposal and the abilities of each team member of the team, the division of the responsibilities is distributed considering that the majority of the work focuses on backend development. Therefore, most part of the resources are assigned to the backend, with specialised members leading and executing tasks related to such area. The member with strong knowledge in interface design and frontend development will be assigned to the development of the user interface, user experience and frontend business logic. The planning and overall structure of the project are taken care by the member with knowledge in software architecture and design. Code quality, testing and similar aspects are elements that each member will have to handle by themselves. Lastly, the entire development is overseen and coordinated by a project leader.

With the previous parameters in mind, the initial planning consisted in the creation of the tickets in order to be able to accomplish the expected tasks, which were defined every two weeks. It is also worth mentioning that tasks could vary in time, as it is really hard to detect possible problems or issues in the development, when planning the tasks.

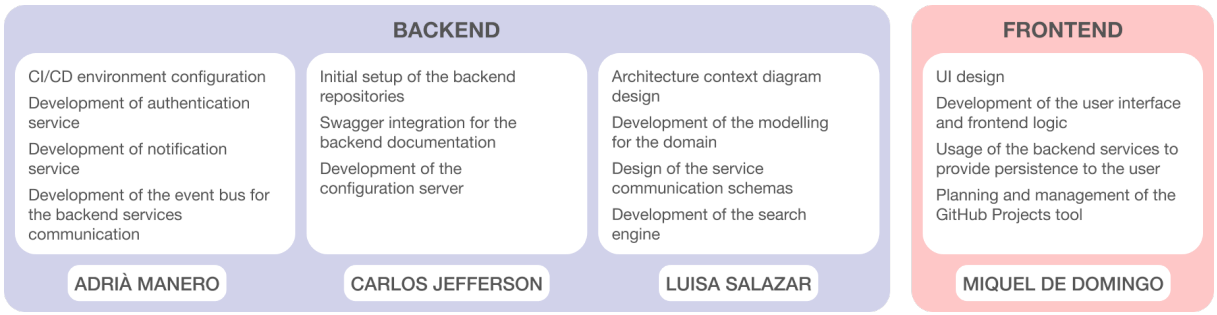

Figure 2.1: Task distribution as a schema

It is important to point out that during the development of the process the each member of the team is not exclusively focused in the tasks described above, rather it contributes and collaborates to the execution of the assigned tickets to other members, when help is needed, helping diminish the handicaps of the project.
At the same time, there are tasks that are executed by all members of the team and are part of the global work required to develop the project:

- Definition of the rules in terms of code (using tools that validate a certain style of code, also known as linters) and to contribute to each repository.
- Planning of the expected work to be finished for each sprint.
- Code review of each developed task, in order to provide a second opinion in the code *to be merged*.
- Study, evaluation and proposal of technologies both for development and deployment of the systems to be developed.

Being more specific, the following are the tasks for each team member.

Miquel de Domingo - Frontend

- **Website design**. Involves all the proposed user interface designs that the application will have, including responsive designs for small screen devices, since the application will be accessed through the browser.
- **Configuration of CI and CD**. This task involves design and execution of strategies that provide frontend repository with continuous integration and deployment strategies.
- **Interactivity control**. As the frontend developer, along with the design, it should be capable to structure a coherent flow of activities and actions that the user will be taken through while interacting with the application, leveraging and experience.
- **Usage of REST APIs**. The frontend application should be capable to communicate with the backend services in order to provide the expected user experience in terms of persitence.
- **Planning and management of GitHub Projects**. As a team that uses agile methodologies to make ends meet, this task involves the member to structure the team's rules and finding tools that align with the chosen methodology.

Adrià Manero - Backend and DevOps

- **Configuration of CI and CD**. This task involves design and execution of strategies that provide backend repositories with continuous integration and deployment strategies.
- **Development of the user auth**. The service should be responsible for the direct communication with the frontend user, with the usage of an API gateawa to redirect requests to other backend services.
- **Development of the notification service**. The service should be responsible for triggering the expected notifications received by the search service, for each created alert.
- **Development of the event bus**. This component will allow backend services to subscribe to message queues and trigger events for each request based on the queue.

Carlos Jefferson - Backend

- **Backend repository setup**: To maximize the shared expertise in Java and Gradle among the backend professionals, each service was placed in repositories with consistent configuration templates, providing a standardized starting point.
- **Swagger integration**. The Swagger library should be incorporated in order to simplify the asynchronous development between the frontend and the backend. Thus, the swagger docs should be the only source of truth.

- **Configuration service creation**. This service manages user registration of recurrent trips and generates alerts for the search service. It also handles persistence of user-defined recurrent trips.

Luisa Salazar - Backend

- **Context diagram**: This task involves the definition of the diagrams that will provide a general view of the architecture of the systems, including the frontend application to the backend services.
- **Domain layer modelling**. Involves defining the shema that will be used in the database, as well as modelling such entities in the domain layer, including their logic, for each service that requires it.
- **Communication schema definition**. It is important to have defined the JSON schemas that will be used to communicate between the backend services. This JSON schemas will benefit the development of the request/responses sent to the message broker.
- **Development of the search engine**. The search engine should be able to perform the search and obtain results, if any, for the travel alerts created by each user. At the same time, it is responsible to send the results obtained to the message broker in order to be able to send notifications to the user.

2.2 Project management system

As a project management approach, the development team followed an Agile methodology combined with an adapted Kanban approach. This allowes the creating of a comprehensive project management approach philosophy tailored to the team and a concrete plan for delivering a high-quality project.
This decision was made taking into account the necessities of the project, the team and the approach to develop the project. Each of the member of the team already has experience working with agile methodologies and are familiarised with the principles of it. The usage of this methodology aims to:

1. Encouraging collaboration between the mebmers, as the methodology is collaborative in its nature, promoting teamwork and shared decision-making.

2. Ensuring a fast delivery of results, as it promotes constant and time effective delivery cycles. This allows for early feedback and continuous improvement.

3. Valuing people over processes, acknowledging the diversity of team members and adapting in each iteration, in order to accommodate individual circumstances and needs.

2.2.1 Agile but not scrum

It is important to note that the agile methodology has been adapted to the development of the project, as it is not tied to Scrum, because of the team work dynamics, time constraints and asynchronous disponibility. Scrum methodologies implies a more synchronous ceremonies which the team was not able to satisfy. Nonetheless, the team has successfully had biweekly meetings, which included work presentation, revision in order to obtain feedback for what has been developed, and planning for next meetings.

Consequently, the team has been based in a Kanban methodology, which suits perfectly to the needs of the project, as it provides a truly simple approach to find an equilibrium between work that needs to be done and the disponibility of each team member. The Kanban methodology

it is based in a continuous improvement methodology, in which tasks are extracted from a list of pending actions from the constant workflow.

To sum up, for the current team, the combination of both methodologies has allowed us to develop fast, asynchronously and in parallel.

2.3 Project management tool

In line with the project management methodology adopted, the team also relies on a tool to aid in the planning and to find a more efficient management of the work to be carried out. The usage of such task-tracking tool, it simplifies the creation, management and listing of task to do, in progress and completed.
By using this tool, the team can streamline the planning process and ensure effective and asynchronous time management. The team and each member is able to prioritise the tasks to be completed in each sprint, track their progress and monitor the status by looking at the board. This approach helps each member to have a better organization of the work, improving efficientcy, and enhancing collaboration. The task-tracking tool simplifies task management and provides a visual representation of the *state of the work*.

There are many different taks-tracking tools and systems, both for agile and non-agile project management. However, the team has prioritised keeping the project simple and finding the maximium automation with the minimum configuration, as key aspects for the tool to choose. This has lead the team to opt for GitHub Projects, an integrated tool within the GitHub application, over what could be considered its major competitor, Jira.

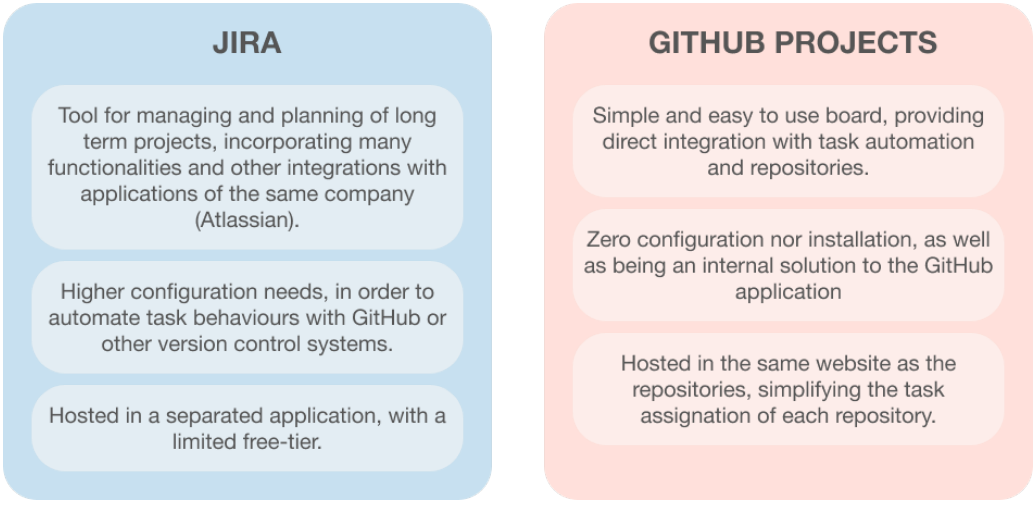

Figure 2.2: Comparison between Jira and GitHub projects

The features of Jira simplified in the above chart, present it as a task-managing tool for more complex projects. However, considering the specific development characteristics of this project, it is clear that the time constraints play an important role in preferring one tool over the other. This is a short-term project, and requires a mangement tool that primarly allows the organization of issues and provides an easy visualisation. Nonetheless, GitHub projects can be scalable to bigger projects, which would allow future teams to keep using it as their task-managing tool. Therefore, the features provided by GitHub Projects's boards, as an adaptable and flexible tool that enables tracking and planning of work within the code repository, are ideal for the execution of this project. It also facilitates the organization of tasks and provides a straightforward way to visualise the status of their resolution.

2.4 GitHub Projects

2.4.1 Team organization

In order to properly develop the project, the GitHub organization members have been divided in two teams. The *backend* team consisting of the three[1] members with backend related tasks. The *frontend* team consisting of the member[2] with frontend related tasks.

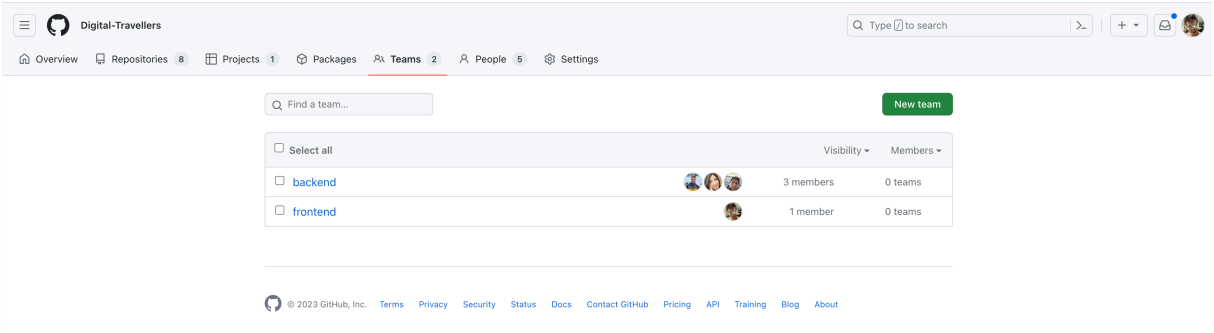

Figure 2.3: Comparison between Jira and GitHub projects

The goal of having projects within the project is to reflect the structure of the team with mentions and permissions in a cascade format. The members of each team can send notifications to an entire team or ask for reviews for a team. There, each member of the team will receive the notiication. GitHub organizations allow the creating of teams, which can have sub-teams within. In the current team structure, such nested level is not required.

2.4.2 Board view

The board of a project is what allows the developers to visualise the state of the project. It can accept from issues defined in the board (not linked to any repository), issues linked to a repository and even pull requests, specific for a repository aswell. It also allows filtering and grouping by many different options. This filtering and grouping can also be stored as views in a separate tab.

[1] Adrià Manero, Luisa Salazar and Carlos Jefferson
[2] Miquel de Domingo

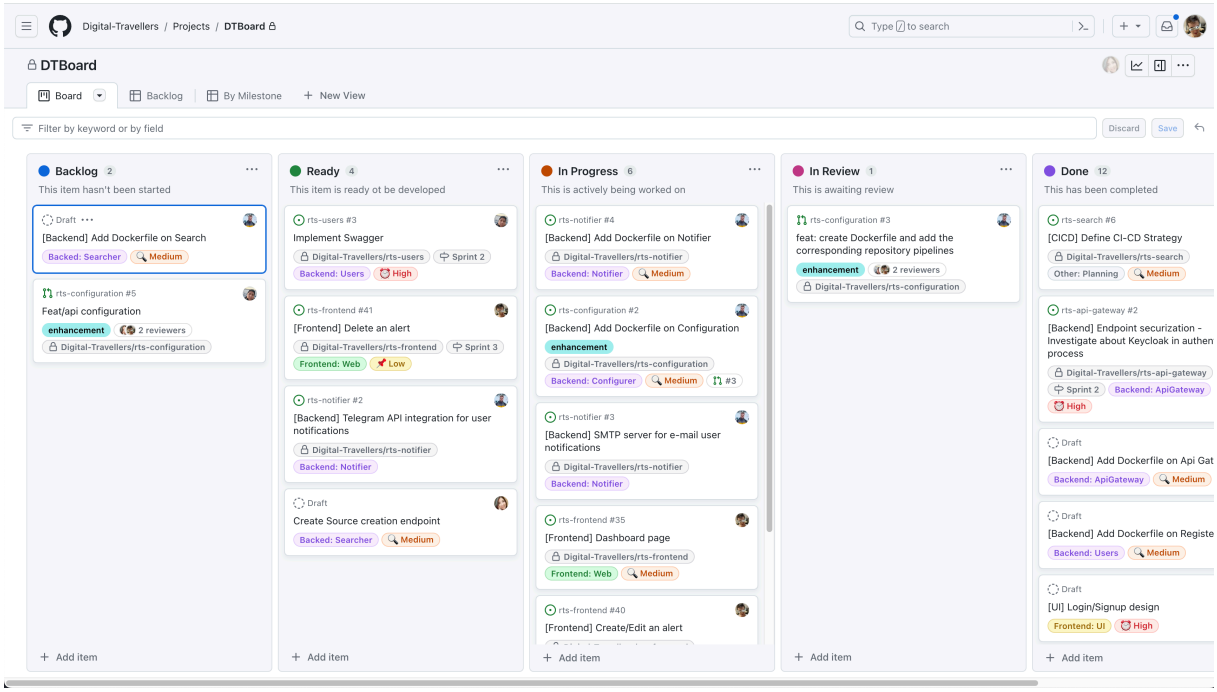

Figure 2.4: Visualisation of the board

Since GitHub does not have the concept of sprints, the team has used milestones as an alternative to defining the sprints. Even though milestones are repository specific, they can be grouped for different repositories as long as they have the same name.

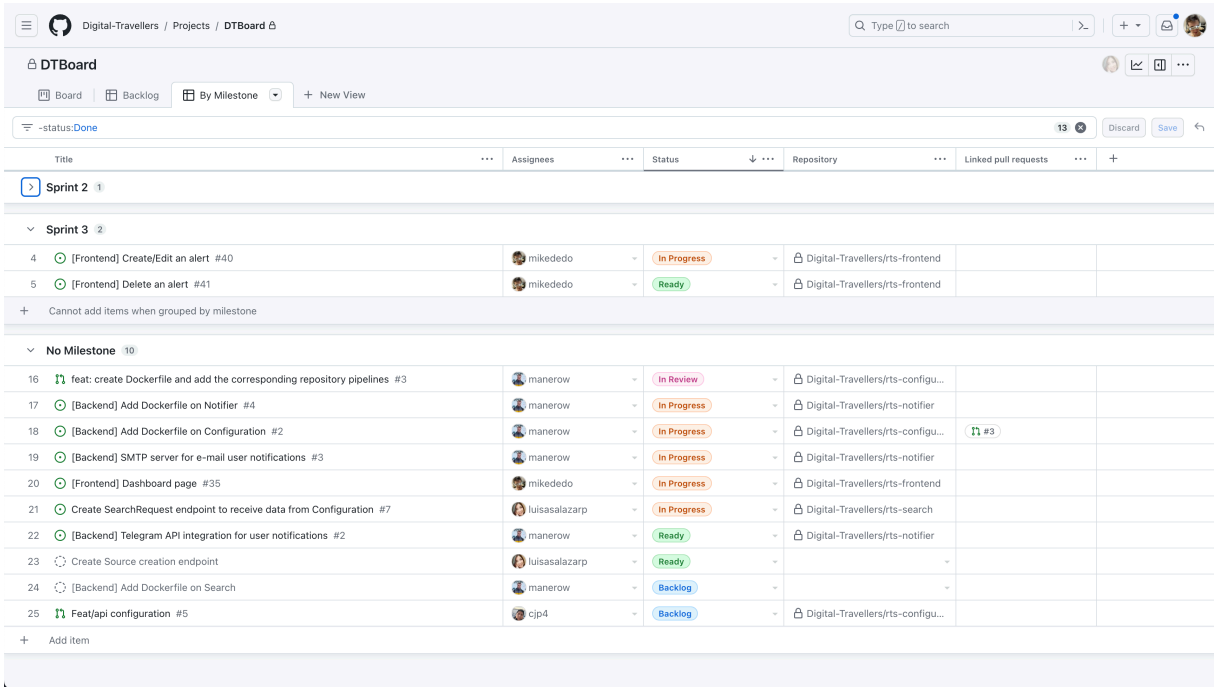

Figure 2.5: Visualisation of the board, grouping by milestones

Another benefit of directly using the board from GitHub is the possibility of constantly being up-to-date with the project status, as the project will also be updated with the activity (creation of pull requests and issues) of each repository.
It also allows to have personalized field for each ticket, allowing the team to add extra infor-

mation just with the visualisation of the ticket. In this case, the following meta-information has been defined for each ticket:

- Status. The status allows the developers to know if the ticket is in the backlog, ready to be developed, in progress, in review or waiting to be reviewed, and done, in case the ticket has been merged successfully.
- Project. A small tag that gives information to which project the ticket is related.
- Priority. It allows to know which tickets should be prioritised over others.

The addition of such fields is fairly simpler than its competitors.

3. Frontend development

The goal of this chapter is to explain how the frontend application has been planned, organised and developed. In order to be the most time efficient, planning is a key aspect before starting to develop. Furthermore, before starting to develop the frontend, a minimal UI (user interface) should be created, since it will fasten the development.
It is also important to note that the planning should also keep free time for the writing of the thesis, as it is also part of the project.

3.1 Planning

When planning the work for the frontend project, our main goal was to avoid being blocked by backend progress. It is common for frontend and native developers to fall behind schedule because they are unable to progress until the backend is complete. One solution to reduce this lag between frontends and backends is to use GraphQL. However, in our case, we opted to apply GraphQL ideas in a RESTful API, resulting in the following steps:

- Defining the expected request body of a specific endpoint, in order to let the frontend know *what the service expects to receive*.

- Defining the response, if any, that an endpoint would return for a given call.

This approach established a clear contract between the backend and the frontend, with separated implementations that both parties agreed to respect.
To facilitate development, we used a service worker to mock every API call made in the local environment of a frontend developer. The service worker respected the specified contract, allowing us to develop the frontend without being blocked by the backend. Once the app was deployed, the service worker was disabled, and the API calls were made to the *actual* RESTful API. This approach allowed for fast and non-blocking development for both frontend and backend.

The second thing to take into account before starting to structure and develop the frontend are the designs. Since there was not enough time to create a system design and then enough designs that would cover all use cases of our app, we opted for simple designs, which are shown in later sections. As a first sprint, or sprint zero, I was in charge of prototyping the frontend application, which would simplify the process of developing the frontend app.

Next steps are more involved in the development and architecture of the application. As explained in previous sections, we got together every two weeks, as well as keeping an asynchronous communication. We considered each meeting to be a deadline, and in the meeting we would discuss the next steps to take or changes if any. However, since there are always unexpected tasks, it has not been easy to strictly follow the devised roadmap. The following diagram illustrates an aproximate planning of the sprints.
It is important to note that this initial planning is really subjective to changes, as in any app development, there may be incidents, bugs, or other issues that could temporarily block the development of the application. Nonetheless, the goal was to stick as much as possible to the expected plan. Also, Sprint 0 has not been added to the schema since it can be merged with the tasks in the Sprint 1.

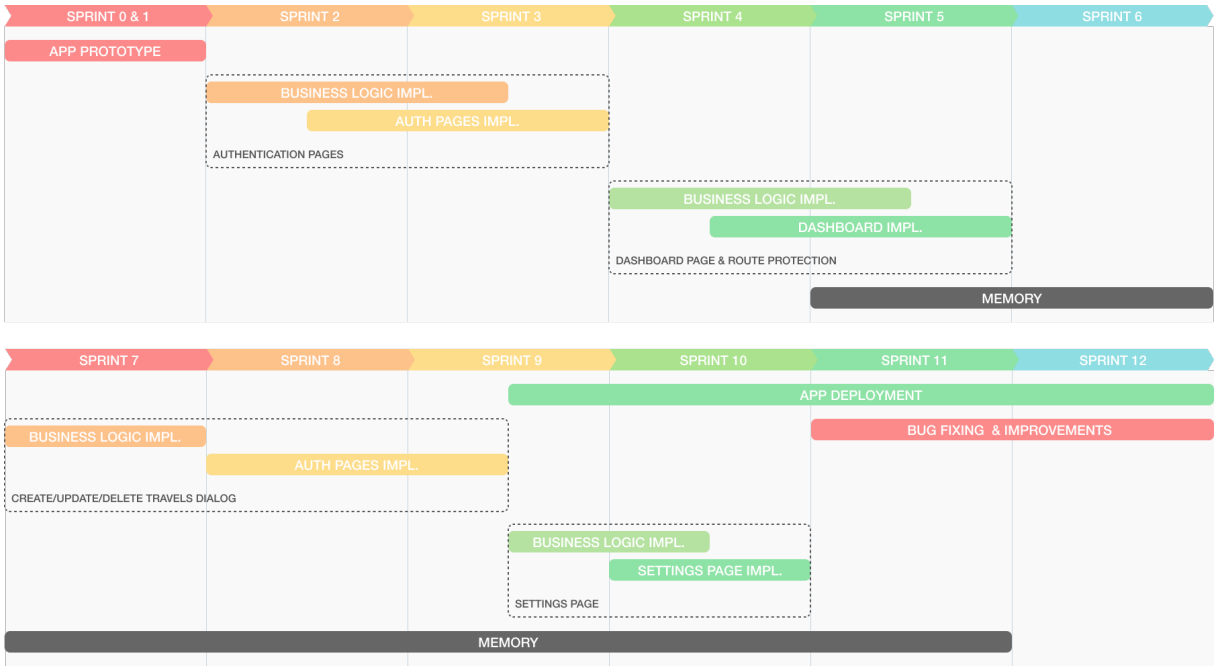

Figure 3.1: Initial sprint planning

Aside from the Sprint 0 and 1 section, other sections consist of two weeks.

3.2 Sprint development

3.2.1 Sprint 0 and 1

As explained previously, the goal of the sprint 0 was to start prototyping the application. Since there was no design system defined, it took a bit more than two weeks to finish the prototypes, even though the were some part that would change in a future.
The design not only helped us visualise the entities and important parts of our application, but also helped me architecture the monorepository. Once tool used to simplify the developer experience is Nx. Using such build system, tasks as having internal libraries and separated applications are easily handled. Therefore, the goal of the sprint 1 was to set up the repository and have it ready to roll. As I was defining the architecture of it, I was able to test possible use cases that I could come across while developing the frontend. This helped me prevent possible time-consuming issues, by handling them in an early stage.
One of the requirements for developing the application was to follow hexagonal architecture patterns and domain-driven design. However, frontend frameworks and technologies often deviate from traditional structures. For example, the React framework primarily uses composition over inheritance and follows a reactive paradigm. As a result, applying these concepts to the frontend posed a significant challenge. More details about the final design will be provided in subsequent chapters.

3.2.2 Sprint 2 and 3

Once the architecture was defined, the first goal of the sprint was to develop the frontend domain logic for user sign up and sign in. As the first code written for the core frontend, it was important to establish a solid structure that could easily accommodate changes. The initial architecture planning proved successful, with the following benefits:

- The hexagonal architecture layers (domain, application, and infrastructure) could be easily decoupled.

- The structure could be easily refactored and scaled as needed.
- The domain logic was separated from the frontend implementation, making it reusable.

Once the domain was defined and ready to be connected to a frontend application, the next step was to implement the authentication pages. This process was divided into two steps:

1. Since most components had yet to be created, an initial implementation was created in the respective package. This implementation was open to modifications and designed to be reusable within different applications, although it was already attached to the React framework.
2. With the components created and the business logic defined, the last step was to connect both through the view layer.

Since there was not a lot of logic involved, it was possible to stick to the original plan, and start developing the frontend views (log in and register pages).
Once the logic architectue had been defined, the goal of the sprint was to start developing the domain of the frontend that will contain the logic to sign up and sign in a user. Being the first code written in the core of the frontend, it was very sensitive to change in terms of structure. However, the initial architecture planning turned out to work seamlessly well, this being:

- Easily decoupling the diferent layers from the hexagonal architecture (domain, application and infrastructure).
- Simplicity to escalate the context as well as to refactor it.
- Containing only business logic code, being completely unaware of any frontend implementation, which allows such logic to be reused.

Once the domain have been defined and was ready to be connected to a fronent application, the next step was to start implementing the authentication pages of the design. This process was divided in two steps:

1. Most of the components had yet to be created, therefore an initial implementation, open to modifications, was created in the respective package. Note that, even though it is only used in one application, another idea of the components internal library is that it can be reused within different applications. However, it is already attached to a framework, which, in this case, is React.
2. Having the components created and the business logic defined, the last step was a simple as connecting both through the view layer.

Authentication pages require of a very basic layout, and most of the logic happens in the form. However, as the logic had already been implemented in the core package, such logic required only to be connected with the components. Therefore, it simplified a lot the development of the view.
Initially it may not seem as an advantage, however if ever appeared the necessity of developing another application that required the same logic, it would mean that it could be reused. Therefore, the developers could only focus in the implementation of the frontend, and then attaching the logic to it.

3.2.3 Sprint 4 and 5

For most of the tasks, to development of them involves a first implementation of the business logic, a second part which may be optional which involves the development of the required components, and a third one that is to involves applying the logic to the view.

Similar to the authentication pages, the first task of the dashboard page involved creating the initial travel mode. In the dashboard page, the user can see a list of travels and interact with them. The interaction or management would be later developed. The travel model is quite complex, as it is fetched from the backend in a specific model, meaning that it should be parsed and transformed to the expected object.
Another setback that appeared during the development of the logic for the dashboard was the travel model changes, which implied having to add, update or delete all fields that differed. As some parts of the dashboard already included the usage of the intial proposition of the travel model, the specific parts had to be updated.
This exposed one of the common problems between the core package and the frontend implementations that use that package. Any breaking change in the core, as it is the updation of a model, can generate side-effects to the frontend that depends on it. Knowing this issue, I came up with two solutions:

1. First would be to create a versioning of the packages, meaning that the frontend would depend on a concrete version of the core package. At start it could be a good solution, yet it can be really easy to start leaving frontends outdated. The update of the frontend is not inevitable, but it would allow the team to update the changes in the main branch, and migrate each frontend step by step.

2. Second would be to work in feature branches, meaning that for every breaking change in the core package, all the affected applications would be updated accordingly. This implies that the team would have to focus on the migration in order to accept the new changes.

In this case, second option was chosen as versioning the packages was not something planned from the starting point and, even though it is possible to be added, updating all frontends was more simple. Additionally, only one application had to be updated.

3.2.4 Sprint 6

In the sixth sprint, the team agreed to focus in writing the common parts of the memory for the thesis. The goal was to finish as soon as possible such common parts and later start writing non-common parts.
Even though it was not a complex part, the team required of more coordination than usual to start writing the common parts. A part or some parts were assigned to each member, and a shared Google Docs containted all the parts.
After a groupal revision, a couple of changes needed to be made, meaning that the memory common parts task would expand to more that the one sprint, as initially planned.

3.2.5 Sprint 7 and 8

These sprints were planned to be focused on the development of the travel management from the frontend. However, as a team we agreed to *push* the memory and finish the common parts as soon as possible. This resulted in stopping the development of the frontend temporarily.

Once the common part had been written or more advanced, the team could focus back on the development. As explained in any previous task, first came the modelling of the alerts. Inside the core package, there was already a lot of business logic for the alerts, which required only

to be extended by adding the possibility to create, update and delete the travels.
Next, only a couple of components were created or required modification/extension. Inputs, buttons, text and so on had already been created as were required by the other tasks.
Finally, the connection between the logic and the frontend components. In further chapters it will be explained in more depth, yet this tasks also require a bit of state management. The library used to manage API calls and caching such calls is also used as a state management solution. This helps reduce the logic to be implemented in the core application, as it can be kept stateless.
All this tasks include the respective testing, both in the core and in the UI components.

Even though the development of this sprint seems really straightforward, it included more frontend logic than the other tasks. Basically, any action (creation, updation or deletion) of an alert should update the dashboard table. However, as a commonly known problem in React it is the amount of rendering that must be done, for any state changes. This topic will be explained in later chapters, yet the architectural decisions that impacted performance were made during this sprints.

3.2.6 Sprint 9 and further

As the memory took up part of the previous sprints, it was at this point that the initial planning started to tumble. As explained in the beginning, the roadmap was just an idea to align with the rest of the team on what to focus. Nonetheless, during the majority of the time, it was possible to stick to the roadmap.
The management of travel tasks required more time for full development, resulting in an underestimation of the total workload. As a result, these tasks were not resolved until the 12th sprint. Another factor contributing to the extended time required was the simultaneous development of the automated application deployment process. This deployment is a critical task as it allows access to the MVP outside of the local development environment. Additionally, as a team, we needed to ensure that all connections, from the frontend application to the backend application, functioned seamlessly.

With all this in mind, there was very little, to any time to develop the settings page. Also, since there was no service nor endpoint to update the user settings (also because of time constraints), such task was not developed.
Furthermore, end-to-end tests were also skipped because of time constraints.

4. Design

5. Cost analysis

After conducting a thorough planning and being conscious about all the tasks that are required to achieve the expected result, this chapter contains an analysis of the costs of this project, detailing the different investments that could be made.

It has been decided to divide investments dedicated to the project into two main categories: human related resources and non-human related resources.

Finally, provisions for contingencies and any unforeseen expenses that may arise are also taken into account.

5.1 Human costs

In any project, one of the key factors and usually the most expensive is the presonnel that will bring such project to live. In this case, the project is being developed by a group of students, as part of their degree's curriculum, meaning that there is no direct cost linked to the work done.

In order to analyse the human costs for this project, it is important to consider the different actors involved in it, and the time that will be spent in the project.

1. **Students**. Students will require the major part of the budget. As established by the university, the final project equals to 10 ECTS credits, which corresponds to a time expenditure between 250 and 300 hours, each student. In order to simplify the calculation, it has been supposed that each student will spend 275 hours.
2. **Tutor**. On the other side, during the development of the project, there were a set of scheduled meetings that involved the students and the tutor of the project. The project lasts around 22 weeks, and it has been estimated that, including the work outside the meetings plus the meetings, the tutor would spend around 3h each week.

Actor	Hours/actor	Num. of actors	Total hours
Student	275 hours	4	1100 hours
Tutor	66 hours	1	66 hours
			1166 hours

Table 5.1: Human resources costs

Given this information, the total cost of each actor can be calculated through:

$$T_c = C_{ha}(€/h) * N_h$$

In this equation, T_c stands for the total actor cost; C_{ha} stands for the cost for each hour of the actor working; N_h stands for the number of hours to work.

It should be considered that, outside the academic scope, the costs for a project inside a company are incremented due to social security taxes.

5.2 Non-human costs

The non-human costs of a project refer to all those that are not directly associated with the individuals participating in its development but are equally necessary for its successful com-

pletion. These costs encompass a wide range of material resources and services required to carry out project tasks.
In the case of the current software development project, non-human costs can be classified into three main categories: hardware costs, infrastructure costs, and indirect costs.

5.2.1 Hardware

The development of an application or a software design is required of a robust and reliable infrastructure. This infrastructure is essential for the implementation, testing and deployment of the application. The hardware costs can vary in price and include an extensive variety of items for the development and the testing of the product.
In order to calculate the hardware costs, it has been taken into account the indispensable for the project development. In terms of deployment infrastructure, free-tier subscriptions have been used, meaning there is no initial need for an investement in terms of infrastructure. For the hardware elements, their life spand and their depreciation has also been taken into account.

Hardware	Life span	Aprox. unit cost	Total cost
Personal computer	10 years	2.000€	8.000€
Mouse	5 years	20€	80€
Keyboard	5 years	50€	200€
Headset	5 years	40€	160€
			8440€

Table 5.2: Hardware costs

Given this information, the total cost of each actor can be calculated through:

$$T_c = C_u(€/h) * N_a$$

In this equation, T_c stands for the total cost; C_u stands for the cost for each unit; N_a stands for the number of actors to work.

5.2.2 Infrastructure costs

In term of infrastructure, the costs correpond to the cloud services offered by: Google Cloud for the backend applications, and Vercel for the frontend. This tools offer a free-tier or a free-credit tier, for educational purposes. The team has taken advantage of this tiers to deploy the application.

Google Cloud offers a $300 in credits to be used for 90 days. This allows the usage of an extensive catalog of services. In this case, the Google Kubernetes Engine (GKE) is the only service required. This service consists of a Kubernetes cluster self-managed.

Vercel, on the other hand, offers a free plan which is targeted to personal and educational projects. It includes unlimited bandwidth for static sites, 100GB for dynamic sites, as well as offering unlimited deployments and serverless functions.

Service	Duration	Pricing	Total cost
Google cloud	90 days	$300	0€
Vercel	Unlimited	$0	0€
			0€

Table 5.3: Infrastructure costs

5.2.3 Software expenses

The development of a product does not only imply associated costs to the infrastructure or hardware, but also to the tools and applications that allow the developers and teams to do their job. This can include operative systems, IDEs, project management software, databases, software libraries, frameworks, and so on.
It is important to note that such costs can vary a lot depending in the specific needs of the project, and the desicions of the development team. In the case of the project, given its academic focus, the development team has taken advantage of free tools in order to minimize the costs.

Software	Cost
Operative System	0€
GitHub Organization	0€
Discord	0€
IntellIJ Idea Community Edition	0€
Visual Studio Code	0€
NeoVim	0€
Docker hub	0€
	0€

Table 5.4: Software costs

5.2.4 Indirect expenses

In the execution of the project, it is not relevant to take into account costs directly linked to the production of the final project, yet also include the costs that are indirect to the development of the project: the indirect costs.

The indirect costs are crucial in the economic evaluation of a project, otherwise, their omission can be translated in an underestimation of the required resources.

5.2.4.1 Electricity

In the execution of any project, it is not only relevant to consider the costs directly linked to the production of the final product, but also those expenses that, although not directly related, are necessary for its successful completion. These are identified as indirect costs.

Indirect costs are essential in the economic evaluation of a project, as their omission can lead to an underestimation of the necessary budget. Identifying and quantifying them will allow for

more precise resource management and effective planning. Among these indirect costs are the consumption of electrical energy, which includes the use of computers and workplace lighting; internet connection, necessary for software development; rental of workspace; office supplies such as notebooks and pens; and potentially, additional training and education. All of these factors, although indirect, are crucial for the proper execution and planning of the project.

Indirect cost	Unit cost	Total cost
Electricity usage	X	0€
Internet connection	X	0€
Workspace rent	X	0€
Office material	X	0€
Additional training	0€	0€
Travel expenses	X	0€
		0€

Table 5.5: Indirect costs

Bibliography

[1] Unspecified author. Unspecified last update. **On:** *Businnes Travel Statista* [online]. May 15th 2023. **Available on**: https://www.statista.com/markets/420/topic/548/business-travel/#statistic2.

[2] CHEN, Guang, ENGER, Will, SAXON, Steve, and YU, Jackey. Unspecified last update. **On:** *What can other countries learn from China's travel recovery path?* [online]. May 15th 2023. **Available on**: https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/what-can-other-countries-learn-from-chinas-travel-recovery-path.

[3] TravelPerk. Unspecified last update. **On:** *Back to business: from offices to business travel, how does the US feel about returning to work?* [online]. May 15th 2023. **Available on**: https://www.travelperk.com/blog/back-to-business-travel-how-does-the-us-feel-about-returning-to-work.

[4] GHIJS, Stefaan. Unspecified last update. **On:** *The business flight travel suvey statistics* [online]. May 15th 2023. **Available on**: https://flyaeolus.com/blog/2017-business-travel-statistics/.

[5] KUNST, Alexander. September 3rd 2019. **On:** *Business travelers who book their own hotels U.S. 2017* [online]. May 15th 2023. **Available on**: https://www.statista.com/statistics/719856/business-travelers-who-book-their-own-hotels-us/.

[6] SHEIVACHMAN, Andrew. October 27th 2016. **On:** *Milennials are new the most frequent business travelers* [online]. May 15th 2023. **Available on**: https://skift.com/2016/10/27/millennials-are-now-the-most-frequent-business-travelers/.

[7] Unspecified author. Unspecified last update. **On:** *IATA Annual review 2022* [online]. May 15th 2023. **Available on**: https://www.iata.org/contentassets/c81222d96c9a4e0bb4ff6ced0126f0bb/annual-review-2022.pdf.

[8] Nx. **Available in**: https://www.nx.dev.

[9] NextJs. **Available in**: https://www.nextjs.org.

[10] Cypress. **Available in**: https://www.cypress.io.

[11] Unspecified author. Unspecified last update. **On:** *Stakeholder Management using the Power Interest Matrix - Solitaire Consulting* [online]. May 15th 2023. **Available on**: https://www.solitaireconsulting.com/2020/07/stakeholder-management-using-the-power-interest-matrix/.

[12] TravelPerk. **Available in**: https://www.travelperk.com.

[13] Hopper. **Available in**: https://www.hopper.com.

[14] Skyscanner. **Available in**: https://www.skyscanner.com.