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[**?**] and, even with the COVID-19 pandemic, the volume kept going up[**?**]. Business trips are essential for both interpersonal connection development and building trust between different companies[**?**]. 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[**?**]. Moreover, 30% of business travelers in Europe travel once a month, 62% once a year, and 5% between 21 and 40 times a year[**?**].

Group business trips are also a commonality, with over 50% of such being made by two or more workers[**?**]. 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%[**?**].

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[**?**]. 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[**?**].

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[**?**], 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.

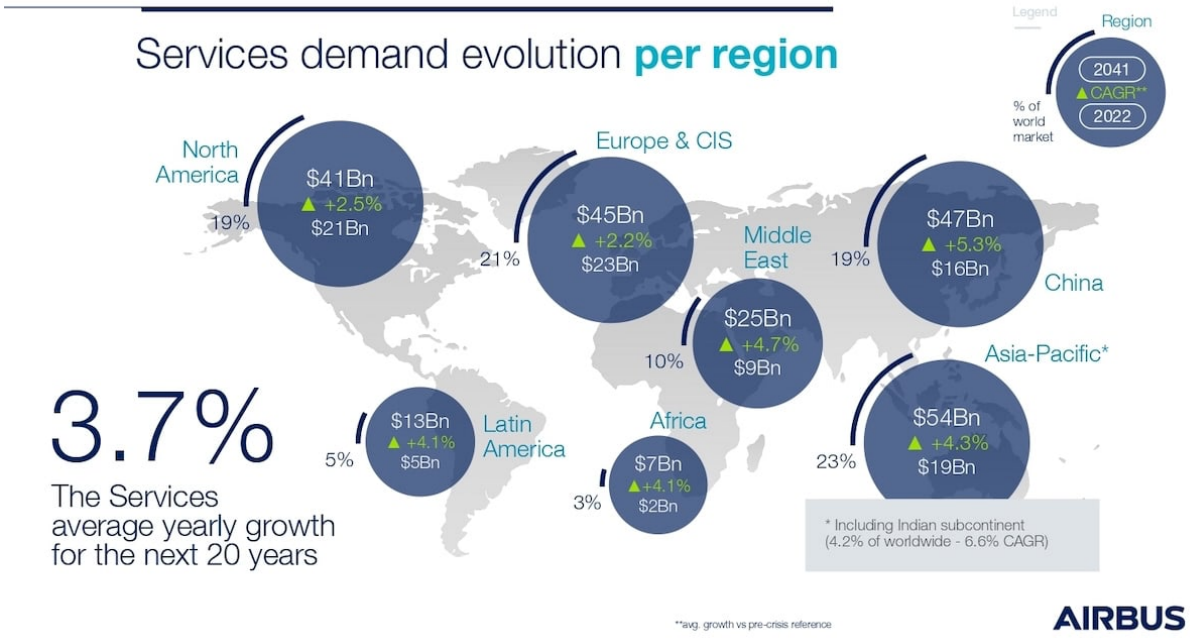

Figure 1.1: Evolution of flight demand of Airbus flights in the pre-pandemic world vs the post-pandemic.

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 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.

1.3.1 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.

1.3.2 Backend concepts

1.3.3 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.

1.3.3.1 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.*

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

1.3.3.2 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.

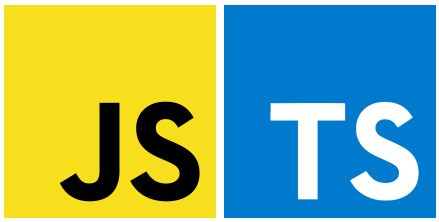

Figure 1.3: JavaScript and TypeScript logos

1.3.3.3 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*.

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.

1.3.4 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: Jest 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*.

1.4 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[**?**] 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.

- **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**[label = -]
  - **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**[label = -]
  - **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.

1.5 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**[**?**]: 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**[**?**]: 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**[**?**]: 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.

1.6 Scope

To be determinated

1.7 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.

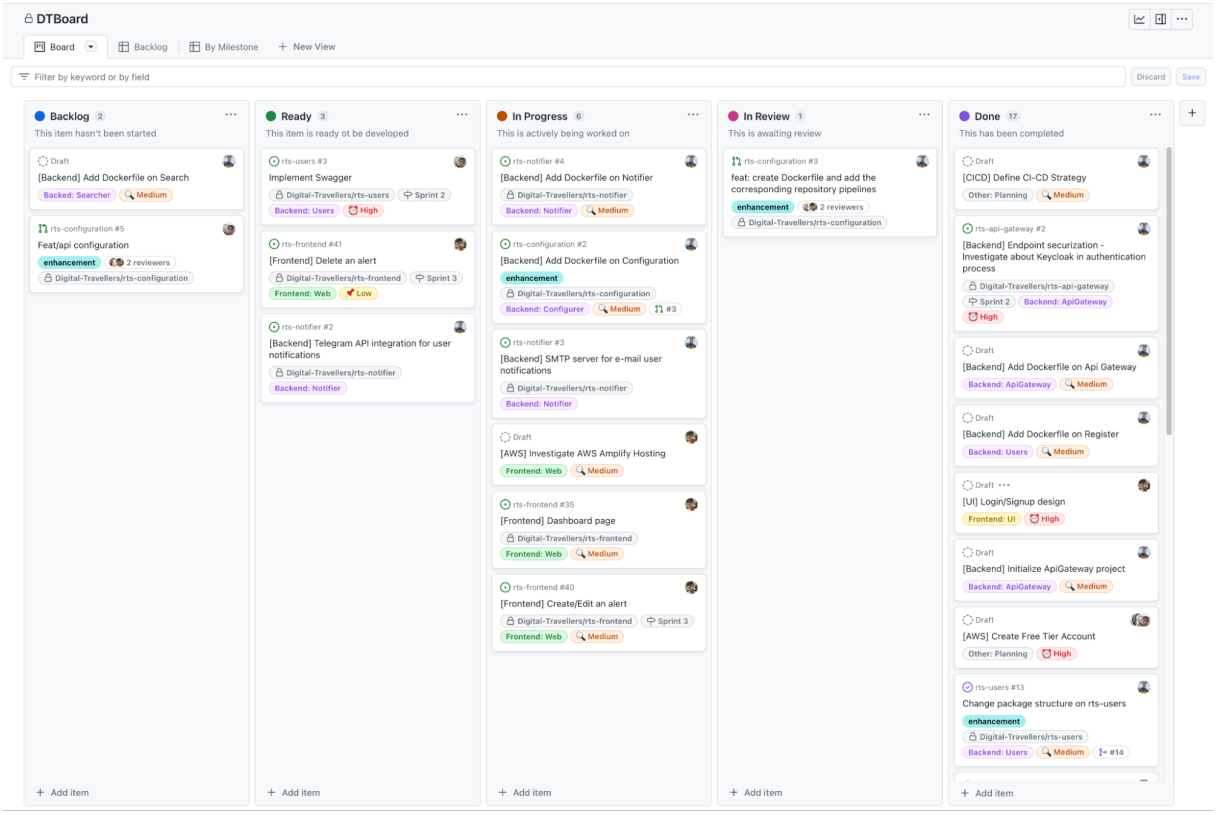

Figure 1.7: Screen capture of the GitHub Projects tool

2. Frame of work

3. Project structure

4. Theoretical background

5. Project structure

5.1 Team

Esto no sé como preferís explicarlo. A tener en cuenta:

- Sprints cada dos semanas.
- Proyecto en GitHub con las tareas.
- Anything else?

5.2 Backend

5.3 Frontend

Regarding the frontend, the application will be stored in a monorepository. The idea of keeping the code in a single repository is to simplify the workflow of the developer or developers. By having a single repository, changes in the domain layer can be made at the same time that the presentation layer is updated. Nonetheless, this small architecture decisions will be explained more in depth in further sections.

5.3.1 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.
TODO: Roadmap picture

5.3.1.1 Sprint 0 and 1

As explained previously, the goal of the sprint 0 was to start prototyping the application. Since there was no system design 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.

5.3.1.2 Sprint 2

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.

Once the architecture 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.

5.3.1.3 Sprint 3 and further

The past few sprints have been somewhat relaxed due to some meetings coinciding with local bank holidays and being postponed. Nevertheless, the focus was on ensuring that the product meets its minimum requirements, which include listing travels created by the user, enabling the creation, update, and deletion of travels, and ensuring proper authentication protection for the application.

The initial implementation of the travel listing followed the agreements made during Sprint 0, but little thought had been given to its design, resulting in a rather poor implementation. However, after one of the meetings, we agreed on a more complex concept of travel, which required updating the initial core implementation of the frontend. Fortunately, the initial design was open to extension, making the required changes easy to implement. This was also a way of testing whether the structure of the context library was well-designed or not.

TODO: Explain deployment process