National College of Ireland

BSc (Honours) in Computing (BSHC4)
Software Development
2022/2023
Gabriel Salas Segura
19104162
x19104162@student.ncirl.ie

Travel Smart Plus

Software Requirements Specification

Contents

Executive Summary

This document outlines the software requirements for the Travel Smart Plus application, an Android application that uses machine-learning algorithms to predict travel itineraries for business users, automating the job of a corporate travel agent and removing the need to select items separately.

Most companies rely on Travel Management Companies (TMC) to book their travels which often requires multiple steps and fees to complete a reservation. This application aims to automate and simplify this process, reducing booking times and costs.

The purpose of this document is to provide a clear and concise description of the functionality and features of the system and serve as a guide for the development process. It includes relevant information about the idea and technologies, along with a detailed explanation of each one of the requirements, their implementation, interface and testing.

1 Introduction

1.1 Background

This application was born as an idea to simplify the booking process for corporate travel. Having worked in the corporate travel industry for a few years, I understand how much companies rely on Travel Management Companies (TMC) to manage their travel arrangements.

Although some TMCs, such as CWT or Concur, have platforms to allow users to book their trips, these applications are not that different from other applications in the market, and users have to research and select each element on their own, which can be time-consuming.

To save time, most users will send a request with their travel preferences directly to the TMC. A travel agent will research the best options for that specific user, send it back for approval and book all flights and hotels using a global distribution system (GDS), resulting in back-and-forth emails, long turnarounds and multiple fees and commissions.

1.2 Aims

TravelSmart Plus aims to streamline and automate the travel booking process using the power of machine learning algorithms, filling a gap in the market, as most travel applications using machine learning only provide basic predictions based on a budget or market trends.

The main novelty features that make this application unique to other applications in the market are:

- **Automatic booking search:** The application uses machine learning algorithms to process real-life flight and hotel data and predict an itinerary with the best option for the user, based on previous bookings and preferences. This removes the need to select each item manually from a long list, simplifying the booking process. You get what you need without having to look for it.

- **Book trips with one single click:** After the application displays the predicted itinerary to the user, they can click the "Book" button, and the application will process the booking using Amadeus, removing the need to enter your details for each booking and reducing booking time, fees and commission. All passenger data required to process the booking will be encrypted and securely stored.

1.3 Technologies

This section highlights some of the technologies chosen for the development process; after carefully researching and comparing the options available.

Back End

- **Ktor:** Ktor is a powerful and flexible asynchronous that allows you to build server-side applications using Kotlin (Ktor, 2023). The simple integration with Kotlin makes it the ideal choice for the back-end API; one single language for the whole development process.
- **Amadeus API:** Amadeus is one of the leading travel technology companies and offers an array of services that allows you to search for live flight and hotel data from different countries. This data is processed by the application to build the itinerary predictions. (Amadeus, n.d.)
- **Google Places API:** The places API provided the images and addresses. Their data set size, accuracy and the free credits are some of the features that made this the best choice for this project. (Google, 2023)
- **PostgreSQL:** A powerful, open-source object-relation database (PostgreSQL, n.d.). Its scalability, large support community and AWS RDS support make it ideal for this project.
- **AWS:** Amazon Web Services offers a wide range of could computing services, including a free tier for most of the services. The application API is deployed using an EC2 instance connected to an RDS instance that hosts the database.

Front End

- **Android Studio:** Although considered more of a tool, it is the official IDE for Android development and includes all the tools and technologies required to develop an Android application, making it the best IDE for the development process. (Google, n.d.)
- **Kotlin:** Kotlin is a modern and concise programming language. It has features like null safety and coroutines that make it a popular choice for Android development. It also integrates seamlessly with Java, giving you a lot of flexibility. (Kotlin , n.d.)
- **OKHttp:** OkHttp is an efficient HTTP client for Android applications and it is used to make the HTTP call to the API along Retrofit.
- **Retrofit:** This type-safe REST client simplifies the HTTP calls.

1.4 Structure

This report includes six sections that aim to provide a complete overview of the project:

- **Introduction:** Covers the project's idea, background, objectives, unique features, and technologies used during the development process.
- **System:** Outlines the main functional and non-functional software requirements to guide the development process.
- **Conclusions:** Summarise the key findings and outcomes, including the strengths and limitations of the project.
- **Further Development or Research:** Explores some of the ideas to further develop the project in the future.
- **References:** Lists all sources that contributed to the project's research and development; cited throughout the report.
- **Appendices:** Includes all supporting materials and documents, such as the project proposal and journals.

2 System

2.1 Functional Requirements

This section defines the features and capabilities of the travel application. The requirements have been prioritized based on their importance to the overall functionality of the application.

ID	Requirement	Priority
FR-1	User sign up	Critical
FR-2	User sign in	Critical
FR-3	Book trip	Critical
FR-4	User management	Critical
FR-5	Trip management	Important
FR-6	Authorise	Important
FR-7	User sign out	Important
FR-8	Passenger Information Management	Important
FR-9	Staff travel management	Desirable
FR-10	Profile management	Desirable
FR-11	Setup Account	Desirable

2.1.1 Use Case Diagram

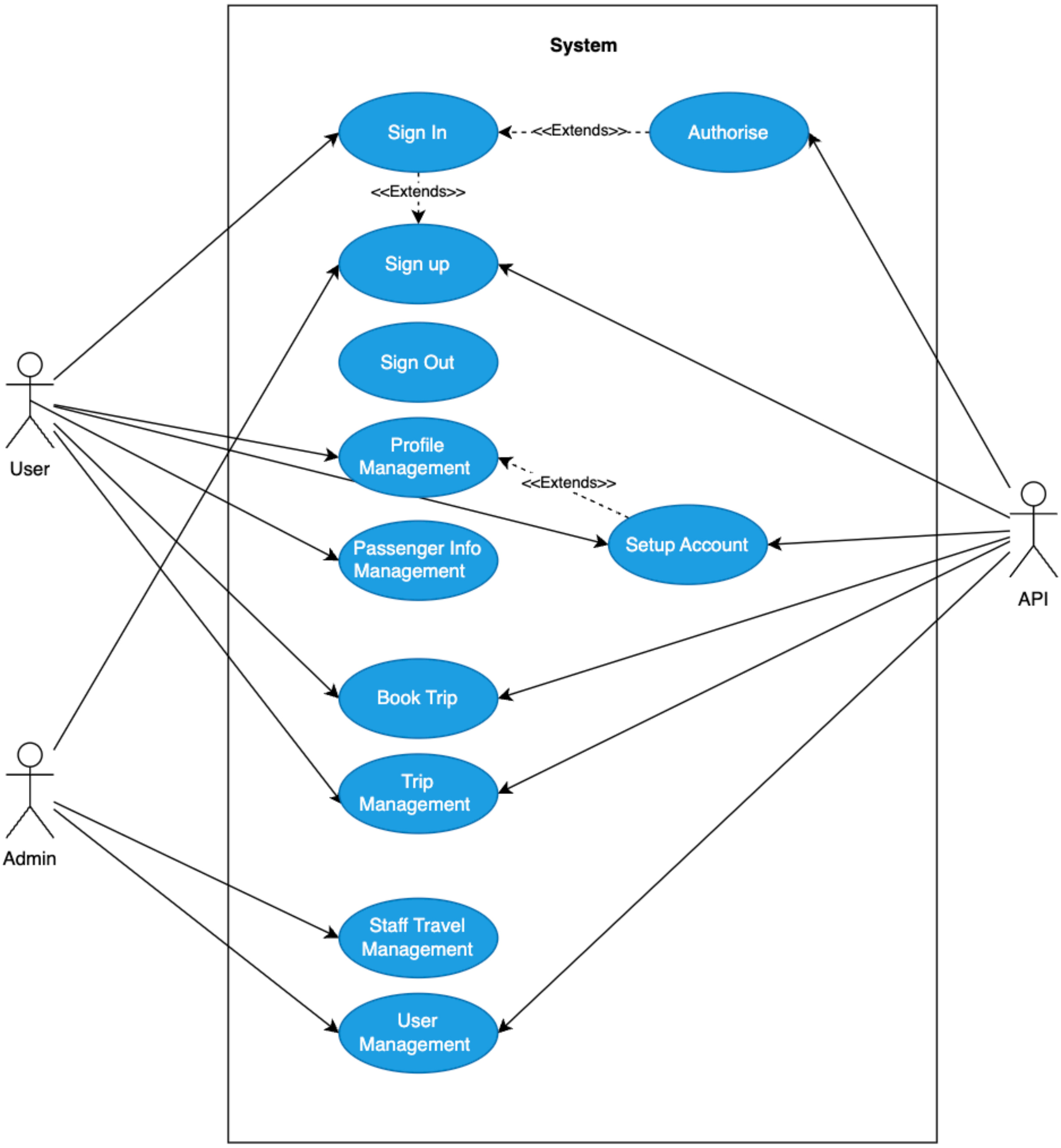


Figure 1 - Use case diagram

2.1.2 Requirement 1: User Sign Up

2.1.2.1 Description & Priority

The sign-up requirement allows an administrator to register their company and gain access to the application. This requirement involves creating a form to submit the administrator and company information. The system should validate the user input and call the API to store the company and first administrator in the database.

This requirement is critical as the administrator needs to create an account to use the application and add other users.

2.1.2.2 Use Case – FR-1

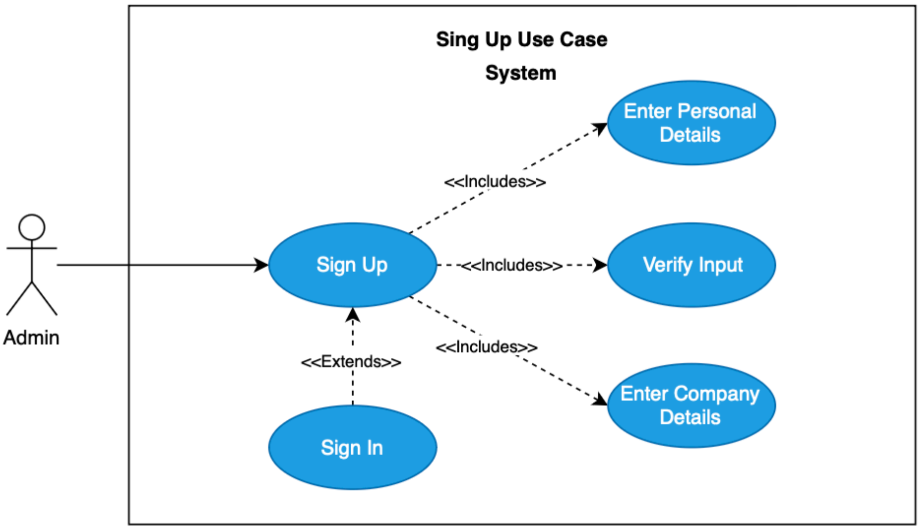
Scope

The scope of this use case is to allow an administrator to register a new account.

Description

This use case describes the process of creating a new account. The Admin must enter their details and company information to complete the sign-up.

Use Case Diagram



Flow Description

Preconditions

- The Admin has downloaded the app and is connected to the internet.
- The Admin has not already signed up.

Activation

This use case starts when the Admin taps the “Sign Up” button on the landing screen.

Main flow

1. The system directs the Admin to the sign up screen.
2. The Admin inputs their company and personal details, including name, email, password, company name and DUNS number (international unique company identifier).
3. The Admin taps the “Sign Up” button.
4. The system validates the input. **(A1)(A2)**
5. The system makes a call to the API to store the user and company details in the database. **(E1)**
6. The system completes the account creation and automatically signs the user in.

Alternate flow

A1: One or more input fields are blank

1. The system detects that some of the fields are blank and displays an error message to the Admin.
2. The system highlights the empty fields and prompts the Admin to enter missing information.
3. The Admin enters the missing information.
4. This use case continues at ***step 3***.

A2: The password doesn’t match

1. The system detects that the password and confirmed password don’t match and displays an error to the Admin asking to re-enter the details.
2. The Admin enters the password again.
3. The system validates that both password fields match.
4. This use case continues at ***step 5***

Exceptional flow

E1: The company is already registered

1. The system detects that the company entered by the Admin already has an account.
2. The system displays a message to the Admin asking them to contact their company to be added to the existing account.
3. The use case **ends**.

E2: The user is already registered

1. The system detects that the email entered by the Admin is already registered and displays a message to the Admin.
2. The system redirects the Admin to the sign in page.
3. The use case **ends**

Termination

Admin is successfully registered and is signed in.

Postcondition

The Admin has created an account and the system goes into a waiting state

2.1.3 Requirement 2: User Sign In

2.1.3.1 Description & Priority

The user sign-in allows registered users to authenticate and access the app functions and resources. The user must enter their email and password, and the system must authenticate the user. If the credentials are valid, the API will authorise the user and generate a JSON Web Token (JWT) and a session and return them to the user.

The session will be stored locally, allowing the system to check if a user is signed in and keep the session alive until they sign out.

This requirement is critical as users should be able to sign in to use the app, and it guarantees the integrity of the app resources.

2.1.3.2 Use Case – FR-2

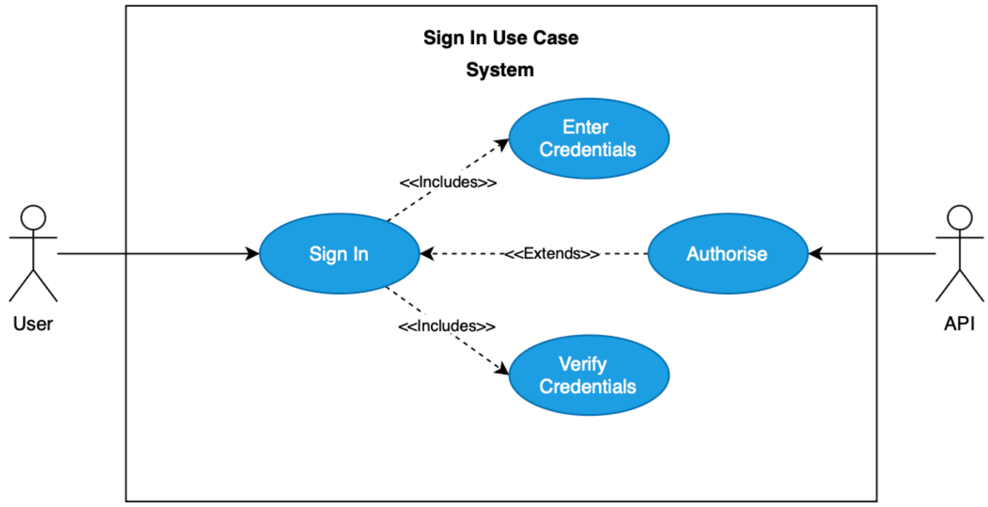
Scope

The scope of this use case is to authenticate the User.

Description

This use case describes the process a User needs to follow to sign into their account and gain access to the app resources.

Use Case Diagram



Flow Description

Preconditions

- The User has an active account.
- The User has a valid email and password.
- The User is not currently signed in.

Activation

The use case starts when the User taps the “Sign In” button on the landing screen.

Main flow

1. The system directs the User to the sign in screen.
2. The User inputs their email and password.
3. The User taps the “Sign In” button.
4. The system will validate the input. **(A1)**
5. The system sends a request to the API to validate the credentials.
6. The API authenticates the User by checking if they are registered and their password is valid. **(E1)**
7. The API authorises the User. (See Use Case FR-6)
8. The API creates a session for the User and stores it in the database.
9. The API sends the JWT token and the session information back to the system.
10. The system saves the session and JWT locally and directs the user to the main screen.

Alternate flow

A1: One or more input fields are blank

1. The system detects that some of the fields are blank and displays an error message to the User.
2. The User enters the missing information and taps the “Sign in” button.
3. This use case continues at ***step 4***.

Exceptional flow

E1: The User enters an invalid email or password

1. The API cannot find the email address or the password is not valid
2. The API sends an unauthorised HTPP error to the system.
3. The system displays “Invalid email or password” error to the User.
4. The use case ***ends***.

Termination

The user is signed in and directed to the main screen.

Postcondition

The User is authenticated and has access to the app resources. The system goes into a waiting state.

2.1.4 Requirement 3: Book Trip

2.1.4.1 Description & Priority

This requirement involves using machine learning to predict a complete itinerary for the user, avoiding the need to hand-pick each one of the elements and saving the user time and effort.

The user enters the destination and dates, and the API use Amadeus API to look for travel options and build an itinerary based on user preferences, previous trips, and company policies. The user should see the option to accept or edit the itinerary, and the API will process the booking.

This requirement is critical for the system since this is the core feature of the application and essential to its success.

2.1.4.2 Use Case – FR-3

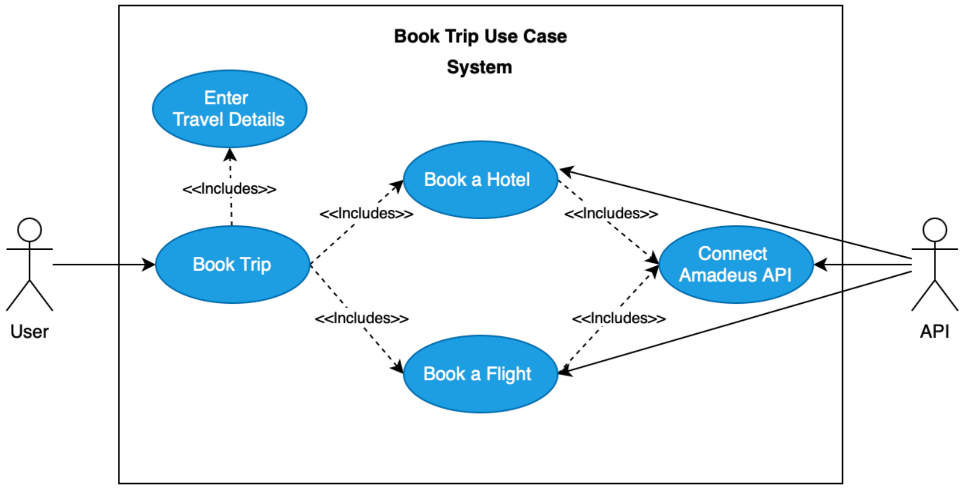
Scope

The scope of this use case is to book a trip for the User.

Description

This use case describes the process of booking a complete travel itinerary for the user. The booking will include flights and hotels, which are determined by machine learning algorithms based on user preferences and other relevant factors.

Use Case Diagram



Flow Description

Preconditions

- The User is signed in and authorised
- The User has entered the necessary travel information, such as destination and travel dates.

Activation

This use case starts when the User opens the app and navigates to the main screen, which includes the option to book a trip.

Main flow

1. The User enters the travel information, such as destination, travel dates, type and the number of adults.
2. The User taps the “Search” button.
3. The system validates the input. **(A1)**
4. The system sends a request to the API with all the travel details.
5. The API looks for flights and hotels and uses machine learning algorithms, such as K-NN, to create an itinerary according to the User preferences and company policy, and sends it back to the system. **(E1)(E2)**
6. The system displays the complete itinerary to the User.
7. The User accepts the itinerary. **(A2)**
8. The system sends the confirmation to the API.
9. The API sends the request to Amadeus API to book all the options and sends the confirmation back to the system. **(E1)**
10. The system displays the confirmation screen.

Alternate flow

A1: One or more fields are black

1. The system detects that some of the fields are blank and displays an error message to the User.
2. The system highlights the invalid fields and prompts the User to enter missing details.
3. The User enters the required information.
4. The use case continues in ***step 2***.

A2: The User wants to edit one or more of the suggested options

1. The User taps the "Edit" button next to the option they want to edit.
2. The system sends a request to the API to get a list of alternatives.
3. The API responds with alternate options, whether flights or hotels.
4. The system displays a list of the options to the User on a different screen.
5. The User selects the new option
6. The system displays the updated itinerary
7. The use case continues in ***step 7***.

Exceptional flow

E1: The API fails to connect to Amadeus' API

1. The API is unable to connect to Amadeus's API
2. The API responds to the system with an internal server error
3. The system displays the error to the User and asks them to try again later.
4. The use case **ends**.

E2: No options found

1. The API is unable to find any options matching user travel information, preferences and company policy.
2. The API responds to the system with an error, such as not found.
3. The system displays a message to the User and asks them to start again with other travel details.
4. The use case **ends**.

Termination

The User has been redirected to the confirmation screen where they can see their itinerary.

Postcondition

The User has successfully booked their trip. The trip is saved in the database and can be accessed by the User.

2.1.5 Requirement 4: User management

2.1.5.1 Description & Priority

The User management requirement allows an administrator to create, edit and delete users.

The administrator can add a new user to the account by entering their name, email and role, and the system will then create a temporary password that the user can use to log in. When the user is created, the administrator can edit their role or delete the user completely. The administrator will also have the option to view the list of all users.

This requirement is critical as only administrators will have the option to add new users to the account. This requirement also helps administrators manage user privileges and ensure data integrity.

2.1.5.2 Use Case – FR-4

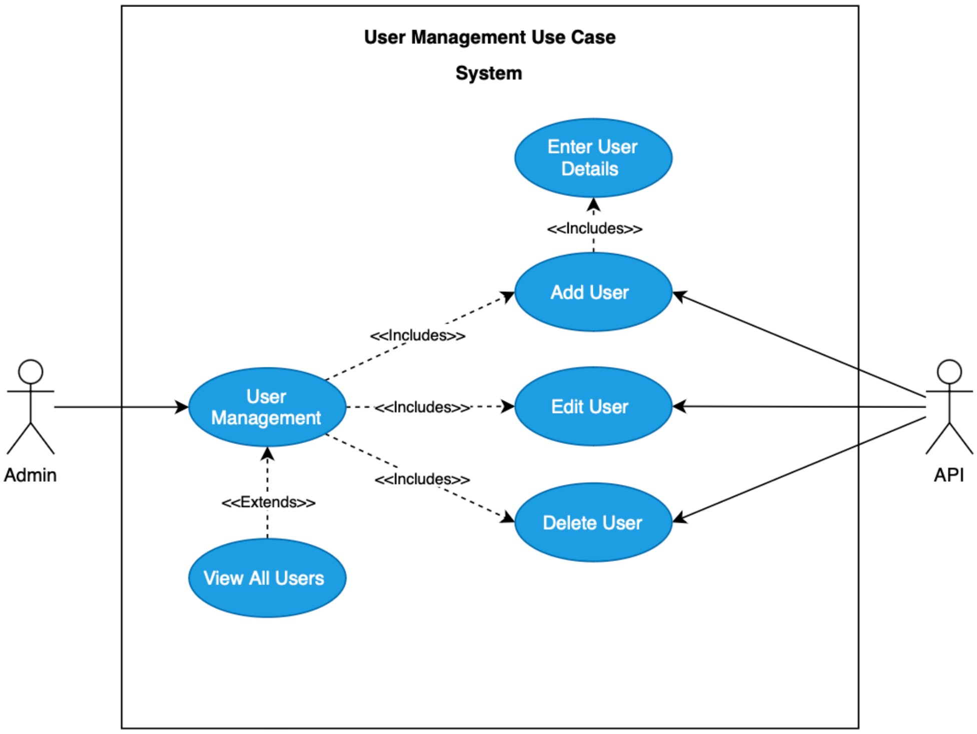
Scope

The scope of this use case is to allow the administrator to manage the users in their company.

Description

This use case describes the steps the administrator needs to follow to add a new user, edit their role, delete the user, and view all their active users.

Use Case Diagram



Flow Description

Preconditions

- The Admin needs to be signed in.
- The Admin has the appropriate role to manage users.

Activation

The use case starts when the Admin taps the “Users” tab on the main screen.

Main flow

1. The system directs the Admin to the user's screen and displays a list of all active users.
2. The Admin taps the “New User” button.
3. The system displays a form to add a new user.
4. The Admin enters the user’s information, including name, email and role, and taps the “Submit” button.
5. The system validates the input. **(A1)**
6. The system sends the data to the API.
7. The API generates a temporary password for the new user, stores all the information in the database and responds to the system with a successful message. **(E1)**
8. The API responds to the system with a created message.

9. The system directs the Admin back to the users' screen and displays a message.
10. The Admin can open the user’s profile to edit and delete the user.

Alternate flow

A1: The Admin enters invalid user information

1. The system detects that some of the fields are blank or contain invalid information.
2. The system highlights the fields and prompts the Admin to correct details.
3. The Admin enters the required information.
4. This use case continues at ***step 5***.

Exceptional flow

E1: The user already exists

1. The API detects that the email entered by the Admin is already registered and sends the message to the system.
2. The system displays an error message to the Admin.
3. The use case **ends**

Termination

The Admin has been redirected to the users' screen.

Postcondition

The system has added, edited, or deleted the user as requested by the Admin. The system goes into a waiting state.

2.1.6 Requirement 5: Trip Management

2.1.6.1 Description & Priority

The trip management requirement allows users to view the itinerary for their upcoming trips and cancel the trip. Users should be able to view their trips easily and quickly, with the option to delete their booking if their plans change.

All trips will be securely stored in the database and can only be accessed by the user and their account administrators.

This requirement is important to create a better experience for the user and learn from user preferences.

1.1.1.1 Use Case – FR-5

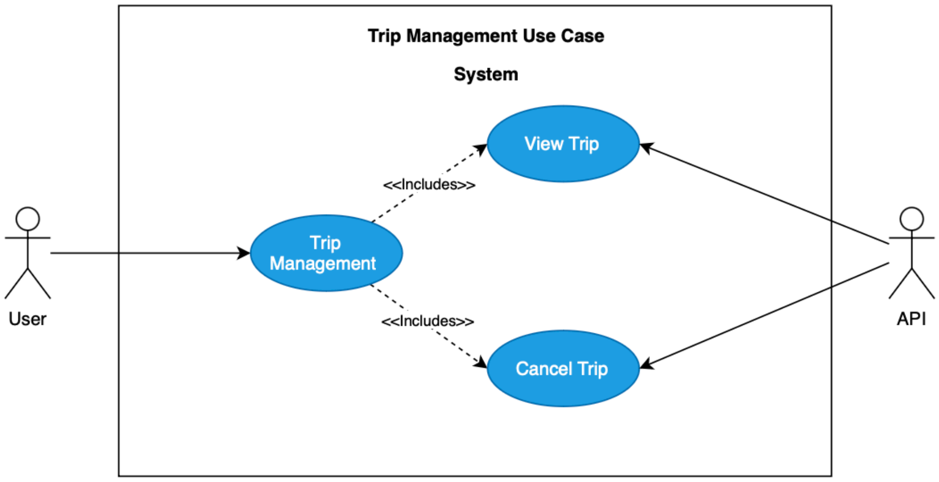
Scope

The scope of the use case is to allow a user to view and cancel their upcoming trips.

Description

This use case outlines the steps the users need to follow to view and cancel their upcoming trips.

Use Case Diagram



Flow Description

Preconditions

- The User is signed in.
- The User has created at least one trip.

Activation

The use case starts when the User taps the “Trips” tab on the main screen.

Main flow

1. The system directs the User to the trips screen and sends a request to the API to get all upcoming trips for that specific user.
2. The API will respond with a list of upcoming trips. **(E1)**
3. The system will display all the trips with some basic information, such as travel date and destination.
4. The User taps the trip they want to view or delete.
5. The system displays the complete itinerary for that trip on a new screen.
6. The User can view all the details and tap the “Cancel” button if they want to delete the trip.

Alternate flow

No alternate flows for this use case

Exceptional flow

E1: The User has no upcoming trips

1. The API cannot find any upcoming trips for the User and responds with not found error.
2. The system will display a message to the User to inform them that there are no upcoming trips.
3. The use case **ends**.

Termination

The User has viewed their trip or successfully deleted it.

Postcondition

The system goes into a waiting state.

2.1.7 Requirement 6: Authorise

2.1.7.1 Description & Priority

The authorisation allows an authenticated user to be authorised using a JWT token that will allow them to access the app resources.

For security reasons, the JWT tokens have a short expiration time. When a user makes a request with an expired token, the API will reply with an unauthorised HTTP error. The user can then make another request to get a new token using their session information.

The API will check if the user has an active session and will issue a new token for the user. If the user has no active session or the session information is invalid, the user will be signed out and asked to sign in again.

This requirement is important to ensure that only authorised users can access the API and guarantee the integrity of the app resources.

2.1.7.2 Use Case – FR-6

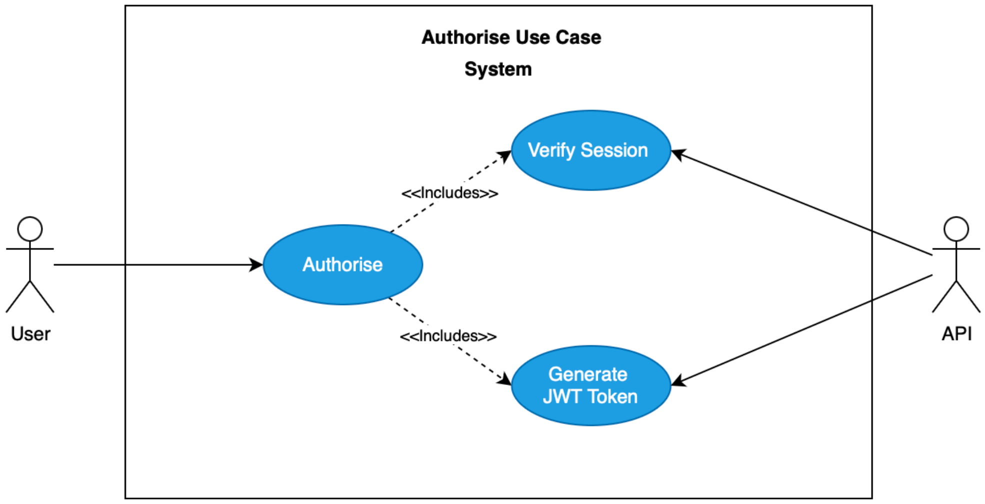
Scope

The scope of this use case is to authorise a User that is already signed in.

Description

This use case outlines the steps that a user needs to follow to get a JWT token that allows them to access the API and other app resources.

Use Case Diagram



Flow Description

Preconditions

- The user is signed in and has an active session.

Activation

This use case starts when the User makes a request with an expired token.

Main flow

1. The API sends an unauthorised error to the system.
2. The system sends a request to reauthorise the User that includes the session information, such as the user id and session key.
3. The API checks if the User has an active session that matches their user id and session key. **(E1)**
4. The API generates a new JWT token and sends it back to the system.
5. The system stores the JWT token locally.

Alternate flow

No alternate flows for this use case

Exceptional flow

E1: The User has no active sessions or the session key is invalid

1. The API cannot find an active session with the user id or the session key doesn't match the active session keys for that user.
2. The API revokes all active user sessions, if any, and sends an unauthorised error to the system.
3. The system signs the user out and redirects them to the sign in page.

Termination

The system has stored the new JWT token in local memory.

Postcondition

The API has generated a new JTW token for the User and the system can continue with any pending requests.

2.1.8 Requirement 7: Sign Out

2.1.8.1 Description & Priority

The sign out requirement allows users to log out of the application. When a user signs out, the API revokes the session that matches the session key.

The sign-out requirement is important as it ensures the privacy and security of user data and helps to provide a better overall user experience.

2.1.8.2 Use Case – FR-7

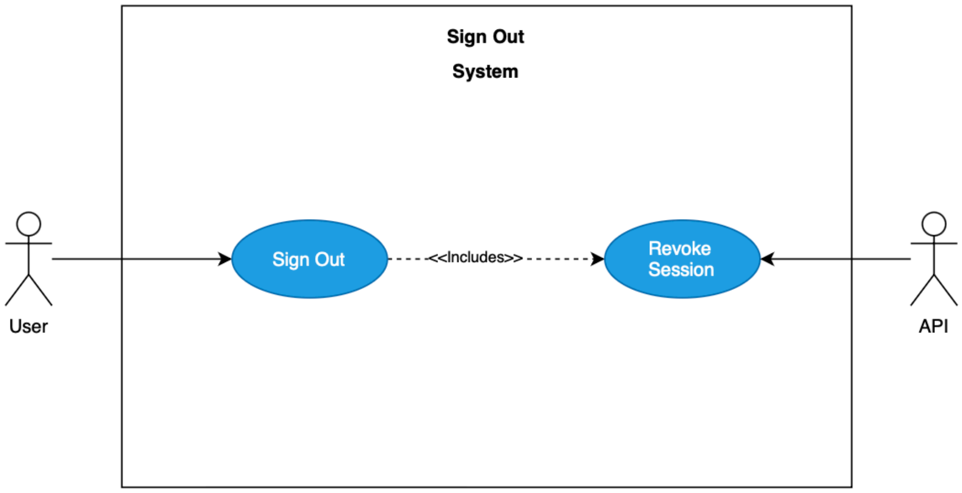
Scope

The scope of this use case is to sign out the user and revoke their session.

Description

This use case describes the steps the User follows to sign out and how the API revokes the session.

Use Case Diagram



Flow Description

Preconditions

- The User is signed in.

Activation

This use case starts when the User taps the “Sign Out” button.

Main flow

1. The system sends a request to the API to sign out the user.
2. The API will revoke the user session that matches both the user id and session key, and sends a success message to the system.
3. The system removes the session from local storage and directs the user to the landing screen.

Alternate flow

No alternate flows

Exceptional flow

No exceptional flows

Termination

The user is redirected to the landing screen.

Postcondition

The user session is revoked and the system goes into a waiting state.

2.1.9 Requirement 8: Passenger Information Management

2.1.9.1 Description & Priority

The passenger information management requirements allow the user to add their travel information, such as date of birth, passport number and expiry date. This data will be used to process the booking through Amadeus.

2.1.9.2 Use Case – FR-09

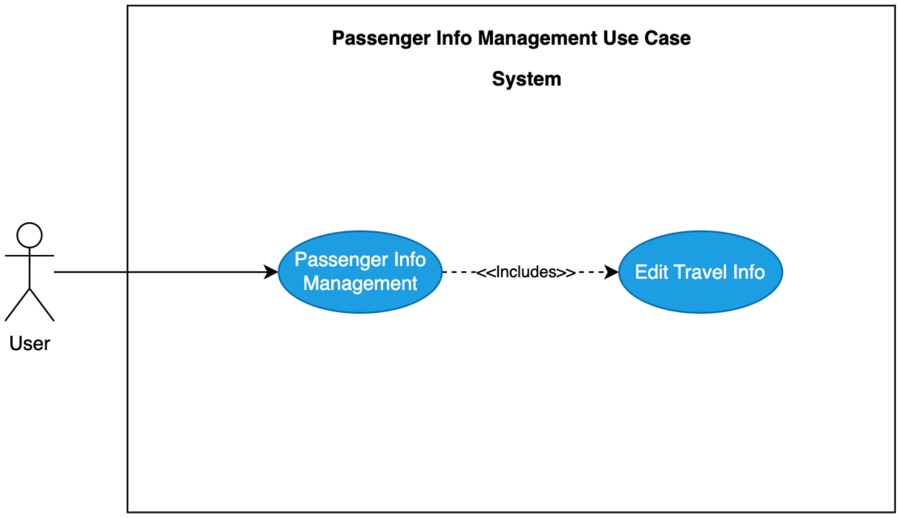
Scope

The scope of this use case is to allow the user to manage their personal travel information.

Description

This use case describes the process a user should follow to add and edit their travel data.

Use Case Diagram



Flow Description

Preconditions

- The User is signed in.

Activation

This use case starts when the User taps the Edit button on the profile screen.

Main flow

1. The system directs the User to the edit screen.
2. The User enters all the required travel information: date of birth, passport number and expiry date and nationality.
3. The system validates the input. **(A1)**

Alternate flow

A1: The User leaves blank fields

1. The system detects that some of the fields are blank.
2. The system highlights the fields and prompts the User to correct details.
3. The User enters the required information.
4. This use case continues at ***step 4.***

Exceptional flow

E1: User cancels edit

1. The User clicks the "Cancel" button
2. The use case ends and the user is redirected to the profile screen.

Termination

The Admin has clicked "Saved."

Postcondition

The system redirects the User to the profile screen and data is stored in the database.

2.1.10 Requirement 9: Staff Travel Management

2.1.10.1 Description & Priority

The staff travel management requirement allows the administrator to manage their company travel.

An administrator can view a list of all the trips that the staff has booked and cancel any trip if needed.

This requirement is desirable to allow companies to manage their travel effectively and improve application usability.

2.1.10.2 Use Case – FR-10

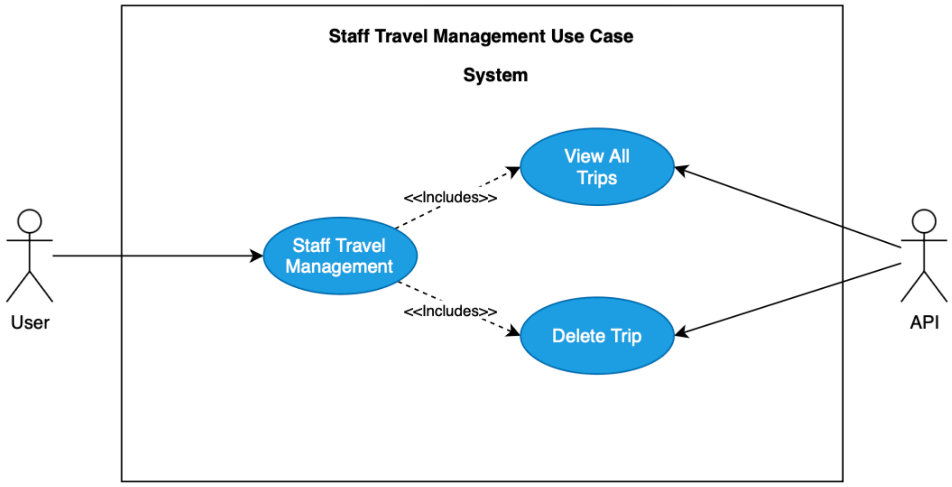
Scope

The scope of this use case is to allow the administrator to view and cancel their user's trips.

Description

This use case describes the process an Administrator should follow to view and delete a user's trip.

Use Case Diagram



Flow Description

Preconditions

- The Admin is signed in.
- There is at least one trip confirmed.

Activation

This use case starts when the Admin taps the "All Trips" tab on the main screen.

Main flow

1. The system directs the Admin to the "All Trips" screen and requests all the trips for the company.
2. The API gets all the trips for that specific company and responds to the system with a list of trips. **(E1)**
3. The system will display all the trips to the Admin.
4. The Admin opens the trip and the system shows the full itinerary.
5. The Admin can tap the "Cancel" button to delete the trip.

Alternate flow

No alternate flows for this use case.

Exceptional flow

E1: No trips found

1. The API cannot find any trips for the company and returns a not found error to the system.
2. The system displays a message to the User to inform them that there are no trips.
3. The use case **ends**.

Termination

The Admin has viewed or deleted the trip and received appropriate feedback.

Postcondition

The system goes into a waiting state.

2.1.11 Requirement 10: Profile Management

2.1.11.1 Description & Priority

The profile management requirements allow users to manage their personal information and preferences within the app. This includes the ability to view and edit personal information such as name, email, email and travel preferences.

The requirement is desirable as it allows users to manage essential details used to book their trips and can improve user engagement and satisfaction.

2.1.11.2 Use Case – FR-1

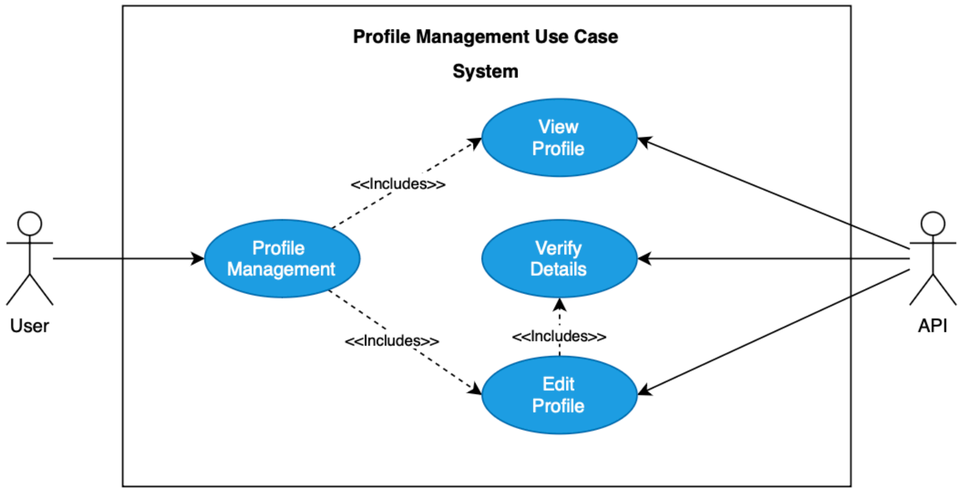
Scope

The scope of this use case is to allow a user to view and edit their profile.

Description

This use case outlines the steps a user must follow to view their profile and edit their details, such as name, password, and travel preferences.

Use Case Diagram



Flow Description

Preconditions

- The User is signed in

Activation

This use case starts when the user taps the ”Profile” tab on the main landing screen.

Main flow

1. The system directs the User to the profile screen and requests user details to the API.
2. The API responds with the User’s details.
3. The system displays the information to the User.
4. The User taps the “Edit” button.
5. The system displays a form to edit the name, email or password.
6. The User enters the new information.
7. The system validates the inputs. **(A1)**
8. The system sends a request to update the user profile.
9. The API updates details in the database and responds to the system with a successful message. **(A2)**

Alternate flow

A1: The User enters invalid user information

1. The system detects that some of the fields are blank or contain invalid information.
2. The system highlights the fields and prompts the User to correct details.
3. The User enters the required information.

4. This use case continues at ***step 7***.

A2: New email is already registered

1. The User enters an email address that is already registered
2. The API responds with a conflict.
3. The system displays a message to the User asking them to enter a different email.
4. The User enters a new email.
5. The use case continues in **step 7**.

Exceptional flow

No exceptional flows for this use case.

Termination

The system displayed existing or new details to the User.

Postcondition

The system goes into a waiting state.

2.1.12 Requirement 11: Setup Account

2.1.12.1 Description & Priority

The setup account use case allows new users to edit their preferences and update the temporary password created by their administrator. The priority is a desirable requirement.

2.1.12.2 Use Case – FR-1

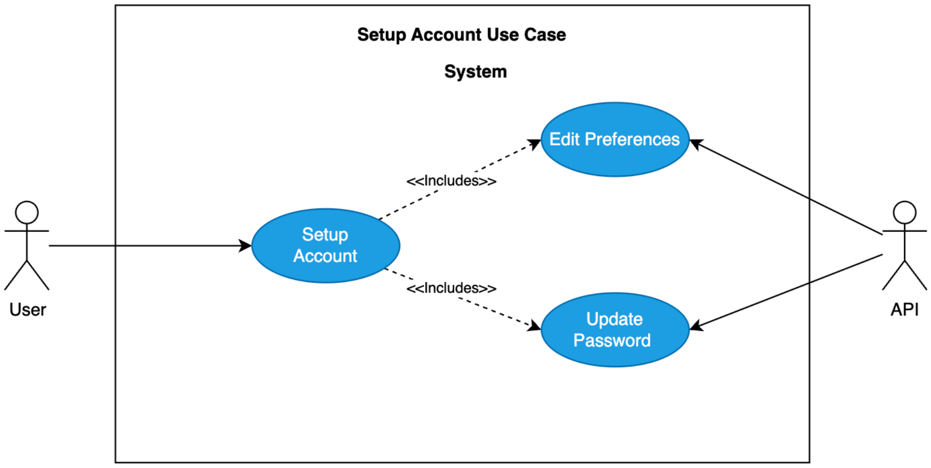
Scope

The scope of this use case is to allow new user to set up their accounts.

Description

This use case outlines the steps a user must follow to update their password and select the preferences during setup.

Use Case Diagram



Flow Description

Preconditions

- The User signs in for the first time.
- The User is active.

Activation

This use case starts when the User signs in for the first time.

Main flow

1. The system directs the User to the setup screen.
2. The User enters a new password and confirms the password.
3. The system validates the password. **(A1)**
4. The User taps “Continue.”
5. The system displays the screen to select preferred airlines and hotels.
6. The User selects the preferred airlines and hotels.
7. The system validates the inputs. **(A2)**
8. The API updates details in the database and responds to the system with a successful message.

Alternate flow

A1: The password doesn’t match

1. The system detects that the confirmation password does not match the password.
2. The system highlights the fields and prompts the User to correct details.
3. The User corrects the password.

4. This use case continues at ***step 3***.

A2: Some of the fields are empty

1. The System detects that some of the fields are empty.
2. The System prompts the User to select all preferred airlines and hotels to continue.
3. The User enters all required information.
4. The use case continues in **step 7**.

Exceptional flow

No exceptional flows for this use case.

Termination

The User clicks "Done" and User is redirected to the main screen.

Postcondition

The system goes into a waiting state.

2.2 Data Requirements

- User data: This includes basic personal information such as name, email address, and password. It may also include additional information such as date of birth and phone number.
- Trip data: This includes information related to the user's travel plans such as departure and arrival dates and destination.
- Data protection: All data must be stored securely and the application should provide users with the option to manage their data to comply with GDPR.

2.3 User Requirements

- An administrator should be able to create an account for their company.
- An administrator should be able to add, edit and delete users.
- The user should be signed in.
- The user should be able to search for a trip and edit the itinerary that is created for them.
- The user should be able to view the itinerary for upcoming trips.
- The user should be able to cancel a trip.
- The user should be able to view and edit their profile.

2.4 Environmental Requirements

- The app requires Android OS or later.
- The app requires a stable internet connection to function.
- The app may require a certain amount of storage space to be downloaded and installed on the user's device.

2.5 Usability Requirements

- The app must have an intuitive and user-friendly interface that allows users to easily navigate and find the features they need.
- The app should include accessibility features, such as support for text-to-speech and screen readers.

2.6 Design & Architecture

The front end follows an MVVM (Model-View-ViewModel) architecture, a popular pattern for Android development and allows for a separation of concerns and improved readability. The code is divided into three main components:

- **Model:** The Model handles data sources, services, and API calls and provides some of the methods used to manipulate the data.
- **View:** the Views are responsible for the UI component and user interactions. Following Google's design patterns (Google , 2023), the app has one main Activity that handles most use cases using Fragments and a Navigation component. There are four other Activities to handle functions like authentication and setup and separate their functionality from the main requirements.
- **ViewModel:** the ViewModel acts as an intermediary between the Model and the View, processing all the data and related business logic (Google, 2023). The data is stored LiveData variables, allowing the Fragments to observe changes and share data among them.

The back end follows a modular architecture, and it has been divided into four main sections:

- **Model:** similar to the front end, the Model handles the data business logic. It encapsulates the data using data classes and defines the table's structure.
- **DAO:** the DAO (Data Access Object) uses Exposed to interact with the database (JetBrains, 2022). It contains all methods required to perform the basic CRUD (Create, Read, Update, Delete) operations and extracts these operations from the rest of the application logic.
- **Services:** the services integrate the different APIs and DAO, acting as an intermediary between the application logic and external services.
- **Routes:** define the API endpoint and handle the HTTP requests. It serves as the entry point to the application's back end.

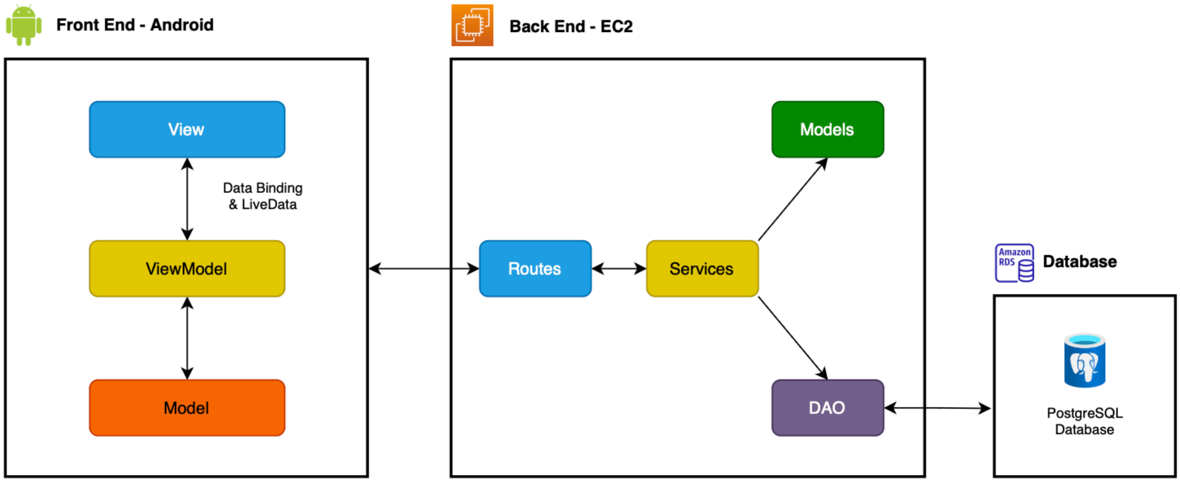
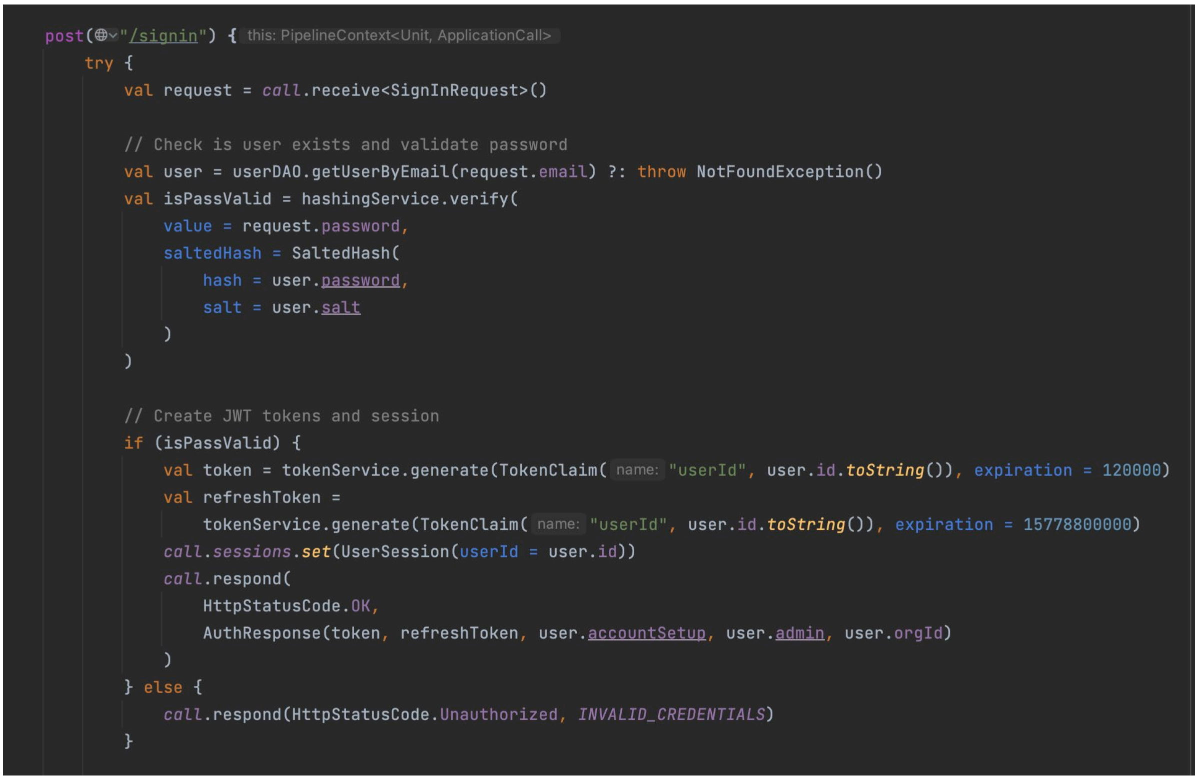


Figure 2 - Architecture Diagram

2.7 Implementation

2.7.1 Authentication

The authentication is handled using a JWT token that is passed in the Authorization header. When the user signs in, two tokens are created and sent back to the user; a short-lived token is used for each request and a long-lived token is used as a refresh token (Figure 3).

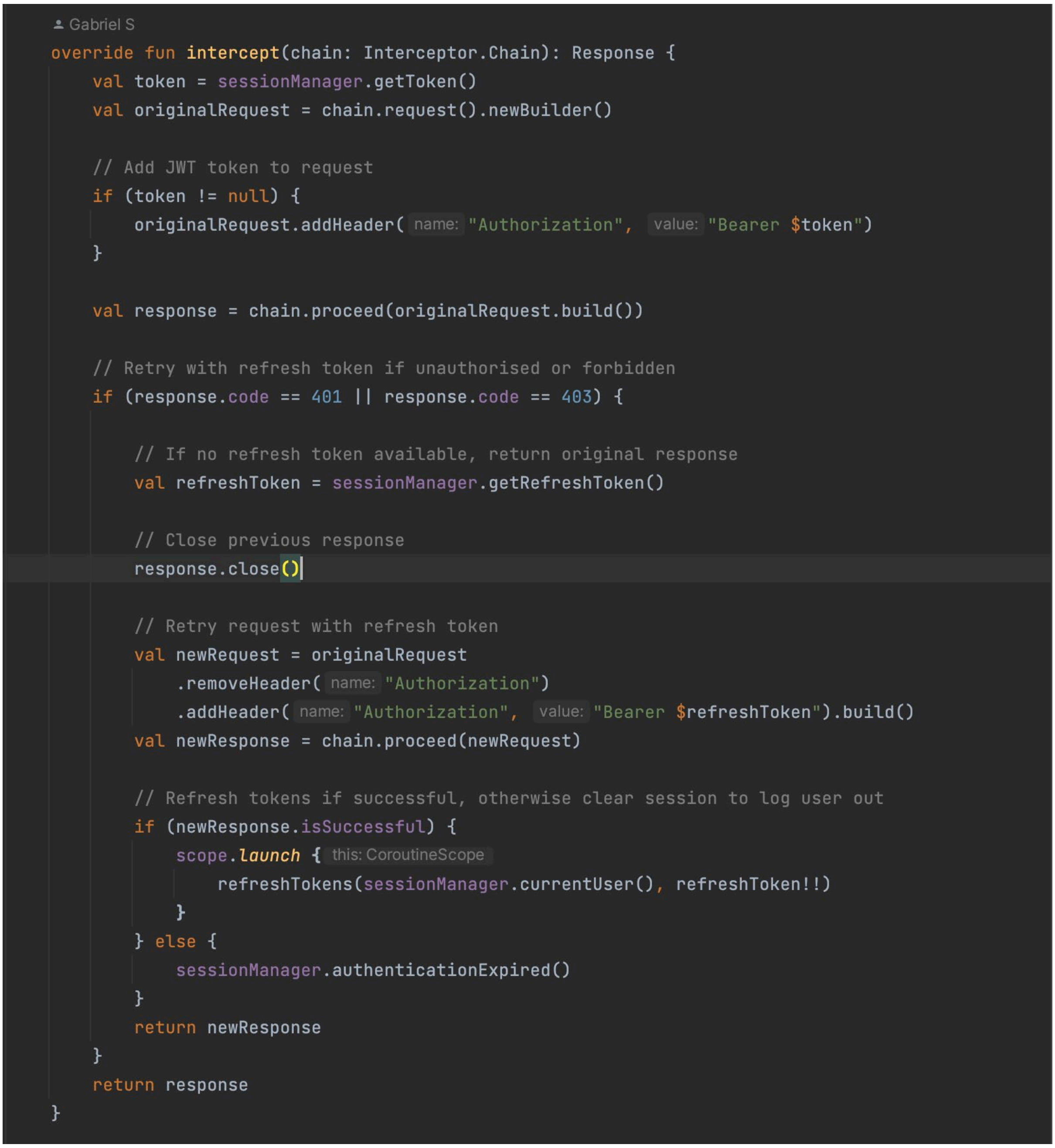


```
post(⊕⌄"/signin") { this: PipelineContext<Unit, ApplicationCall>
    try {
        val request = call.receive<SignInRequest>()

        // Check is user exists and validate password
        val user = userDAO.getUserByEmail(request.email) ?: throw NotFoundException()
        val isPassValid = hashingService.verify(
            value = request.password,
            saltedHash = SaltedHash(
                hash = user.password,
                salt = user.salt
            )
        )

        // Create JWT tokens and session
        if (isPassValid) {
            val token = tokenService.generate(TokenClaim( name: "userId", user.id.toString()), expiration = 120000)
            val refreshToken =
                tokenService.generate(TokenClaim( name: "userId", user.id.toString()), expiration = 15778800000)
            call.sessions.set(UserSession(userId = user.id))
            call.respond(
                HttpStatusCode.OK,
                AuthResponse(token, refreshToken, user.accountSetup, user.admin, user.orgId)
            )
        } else {
            call.respond(HttpStatusCode.Unauthorized, INVALID_CREDENTIALS)
        }
```

Figure 3 – API Sign in route

Both tokens are saved locally using the Android shared preferences. When the user sends a request to the API, an authentication interceptor adds the token to the request, and the API will validate this token using the Ktor authentication plugin.

If the token is expired, the interceptor will retry the request with the refresh token, which should return a fresh token pair if successful. If the refresh token is invalid, the user is automatically signed out and a message will be displayed asking them to sign in one more time (Figure 4).



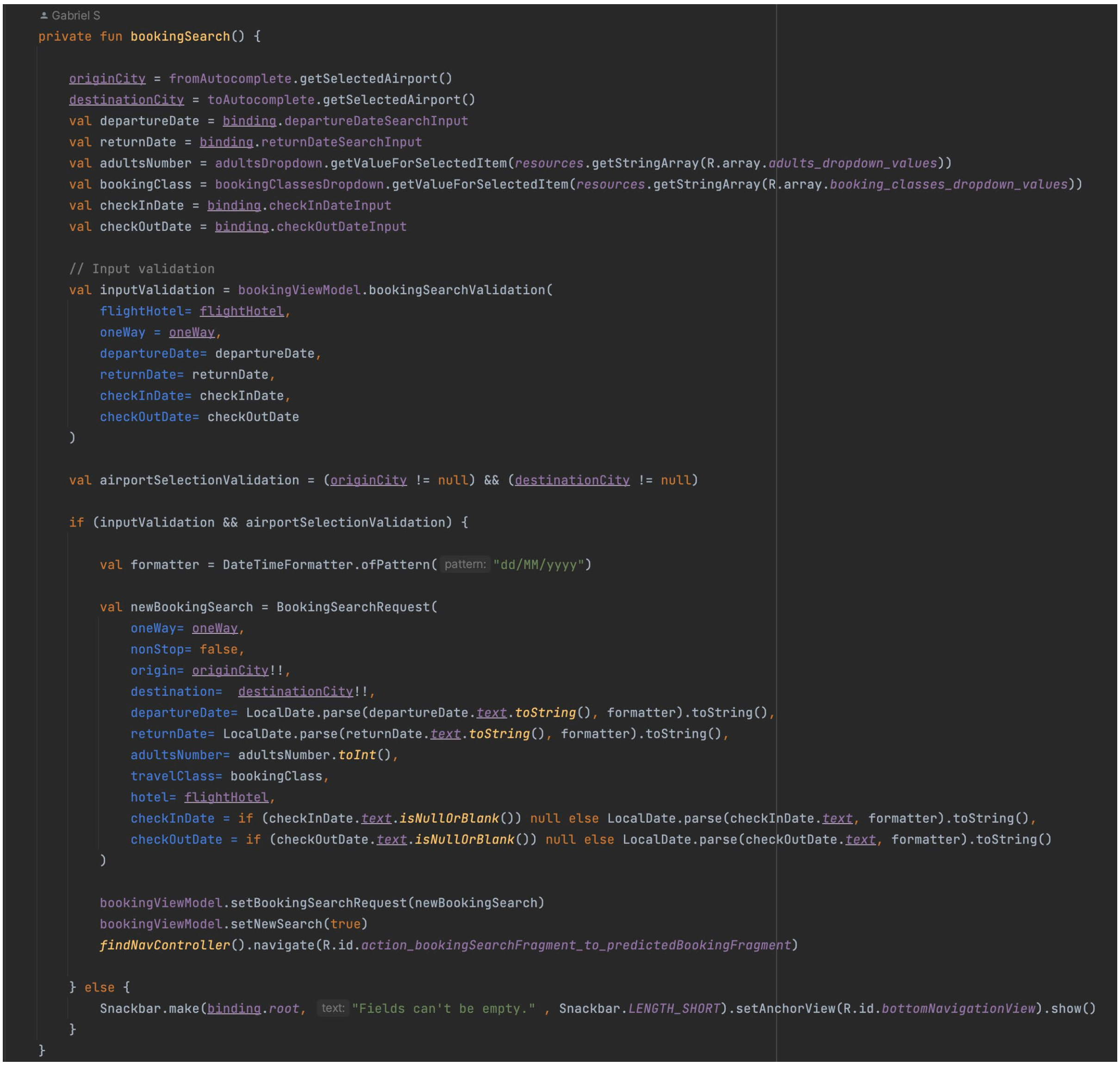
```
Gabriel S
override fun intercept(chain: Interceptor.Chain): Response {
    val token = sessionManager.getToken()
    val originalRequest = chain.request().newBuilder()

    // Add JWT token to request
    if (token != null) {
        originalRequest.addHeader( name: "Authorization",  value: "Bearer $token")
    }

    val response = chain.proceed(originalRequest.build())

    // Retry with refresh token if unauthorised or forbidden
    if (response.code == 401 || response.code == 403) {

        // If no refresh token available, return original response
        val refreshToken = sessionManager.getRefreshToken()

        // Close previous response
        response.close()

        // Retry request with refresh token
        val newRequest = originalRequest
            .removeHeader( name: "Authorization")
            .addHeader( name: "Authorization",  value: "Bearer $refreshToken").build()
        val newResponse = chain.proceed(newRequest)

        // Refresh tokens if successful, otherwise clear session to log user out
        if (newResponse.isSuccessful) {
            scope.launch {  this: CoroutineScope
                refreshTokens(sessionManager.currentUser(), refreshToken!!)
            }
        } else {
            sessionManager.authenticationExpired()
        }
        return newResponse
    }
    return response
}
```

Figure 4 - Authentication Interceptor Implementation

This authentication process guarantees the integrity of the data on the back end by only allowing requests from verified users.

2.7.2 Booking

The booking process consists of two stages: search and confirmation. During the search stage, the user enters all the information required to search for a trip, and the system validates the input and sends a request to the API (Figure 5).

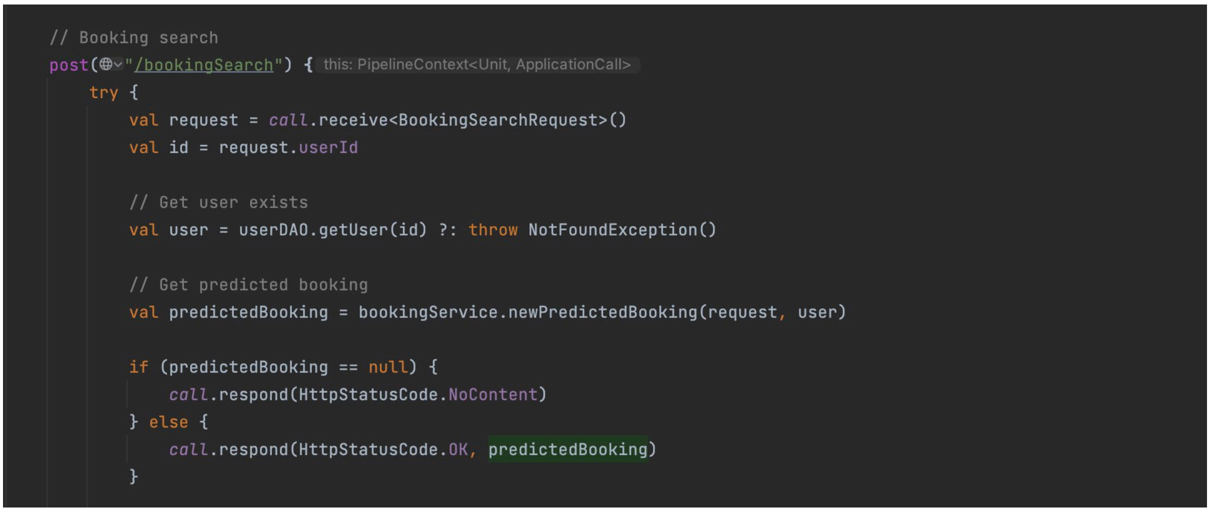


```
Gabriel S
private fun bookingSearch() {

    originCity = fromAutocomplete.getSelectedAirport()
    destinationCity = toAutocomplete.getSelectedAirport()
    val departureDate = binding.departureDateSearchInput
    val returnDate = binding.returnDateSearchInput
    val adultsNumber = adultsDropdown.getValueForSelectedItem(resources.getStringArray(R.array.adults_dropdown_values))
    val bookingClass = bookingClassesDropdown.getValueForSelectedItem(resources.getStringArray(R.array.booking_classes_dropdown_values))
    val checkInDate = binding.checkInDateInput
    val checkOutDate = binding.checkOutDateInput

    // Input validation
    val inputValidation = bookingViewModel.bookingSearchValidation(
        flightHotel= flightHotel,
        oneWay = oneWay,
        departureDate= departureDate,
        returnDate= returnDate,
        checkInDate= checkInDate,
        checkOutDate= checkOutDate
    )

    val airportSelectionValidation = (originCity != null) && (destinationCity != null)

    if (inputValidation && airportSelectionValidation) {

        val formatter = DateTimeFormatter.ofPattern( pattern: "dd/MM/yyyy")

        val newBookingSearch = BookingSearchRequest(
            oneWay= oneWay,
            nonStop= false,
            origin= originCity!!,
            destination=  destinationCity!!,
            departureDate= LocalDate.parse(departureDate.text.toString(), formatter).toString(),
            returnDate= LocalDate.parse(returnDate.text.toString(), formatter).toString(),
            adultsNumber= adultsNumber.toInt(),
            travelClass= bookingClass,
            hotel= flightHotel,
            checkInDate = if (checkInDate.text.isNullOrBlank()) null else LocalDate.parse(checkInDate.text, formatter).toString(),
            checkOutDate = if (checkOutDate.text.isNullOrBlank()) null else LocalDate.parse(checkOutDate.text, formatter).toString()
        )

        bookingViewModel.setBookingSearchRequest(newBookingSearch)
        bookingViewModel.setNewSearch(true)
        findNavController().navigate(R.id.action_bookingSearchFragment_to_predictedBookingFragment)

    } else {
        Snackbar.make(binding.root,  text: "Fields can't be empty." , Snackbar.LENGTH_SHORT).setAnchorView(R.id.bottomNavigationView).show()
    }
}
```

Figure 5 - Booking search in Fragment

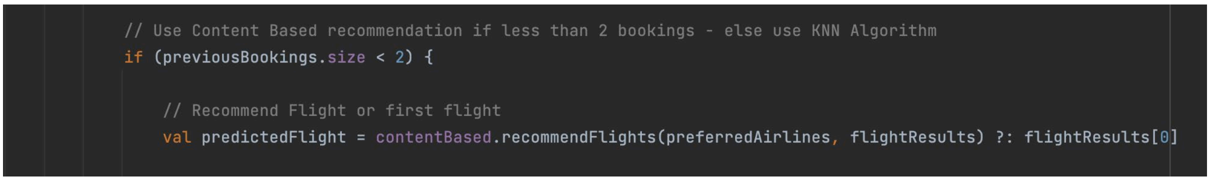
In the back end, the booking search route confirms the user making the request exists, then calls the booking service to make the prediction and returns the relevant response (Figure 6).

The booking service is responsible for calling the external APIs to get flight and hotel data. This data is processed using different algorithms to make predictions; a content-based algorithm when there are no previous bookings and a K-NN algorithm when there is a previous history (Figure 7 and Figure 8).



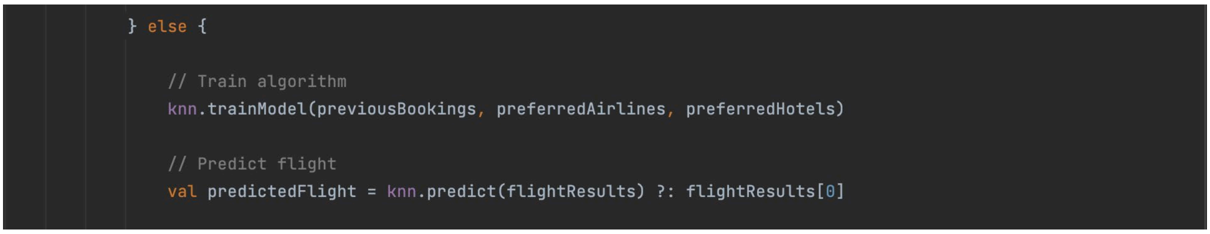
```
// Booking search
post("/bookingSearch") { this: PipelineContext<Unit, ApplicationCall>
    try {
        val request = call.receive<BookingSearchRequest>()
        val id = request.userId

        // Get user exists
        val user = userDAO.getUser(id) ?: throw NotFoundException()

        // Get predicted booking
        val predictedBooking = bookingService.newPredictedBooking(request, user)

        if (predictedBooking == null) {
            call.respond(HttpStatusCode.NoContent)
        } else {
            call.respond(HttpStatusCode.OK, predictedBooking)
        }
```

Figure 6 - Booking Search Route



```
        // Use Content Based recommendation if less than 2 bookings - else use KNN Algorithm
        if (previousBookings.size < 2) {

            // Recommend Flight or first flight
            val predictedFlight = contentBased.recommendFlights(preferredAirlines, flightResults) ?: flightResults[0]
```

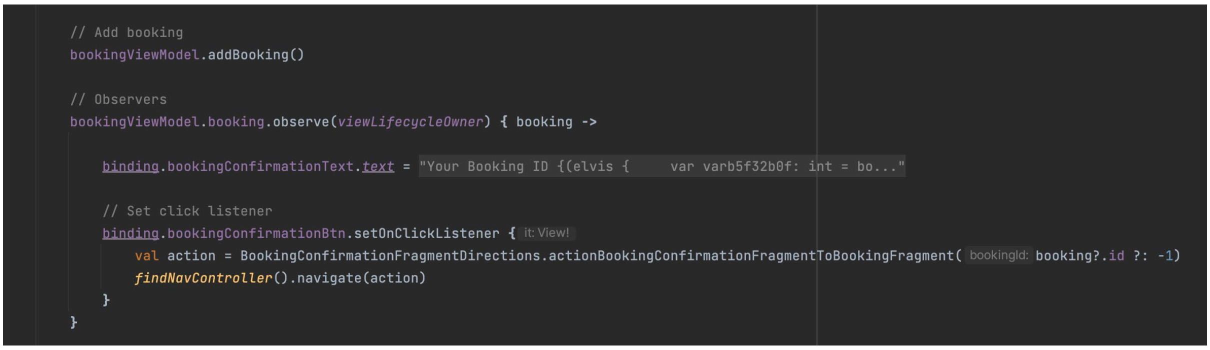
Figure 7 - Booking Service Content-Based Recommendation



```
        } else {

            // Train algorithm
            knn.trainModel(previousBookings, preferredAirlines, preferredHotels)

            // Predict flight
            val predictedFlight = knn.predict(flightResults) ?: flightResults[0]
```

Figure 8 - Booking Service KNN Algorithm

During the confirmation stage, the user can confirm the predicted itinerary with a single click. This sends a second request to the API to confirm the booking and return the confirmation with the booking ID and reference. The call is made after navigating to the Booking Confirmation Fragment to create a smooth UI transition (Figure 9).



```
// Add booking
bookingViewModel.addBooking()

// Observers
bookingViewModel.booking.observe(viewLifecycleOwner) { booking ->

    binding.bookingConfirmationText.text = "Your Booking ID {(elvis {      var varb5f32b0f: int = bo..."

    // Set click listener
    binding.bookingConfirmationBtn.setOnClickListener { it: View!
        val action = BookingConfirmationFragmentDirections.actionBookingConfirmationFragmentToBookingFragment( bookingId: booking?.id ?: -1)
        findNavController().navigate(action)
    }
}
```

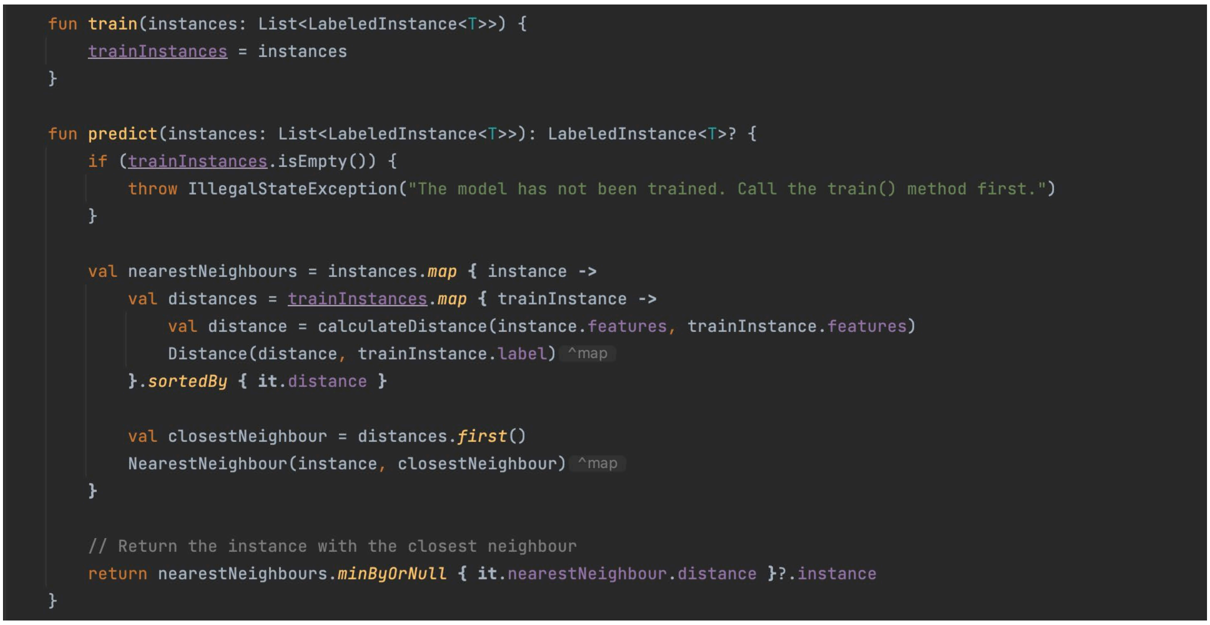
Figure 9 - Booking Confirmation Fragment Implementation

Alternatively, the user can edit any of the recommendations before confirming their booking. New selections are saved and used to train the algorithms.

2.7.3 KKN Algorithm

After carefully evaluating different algorithms, the *k*-nearest neighbours algorithm (KNN) was implemented due to its capacity to handle non-linear complex data, easy implementation and previous experience.

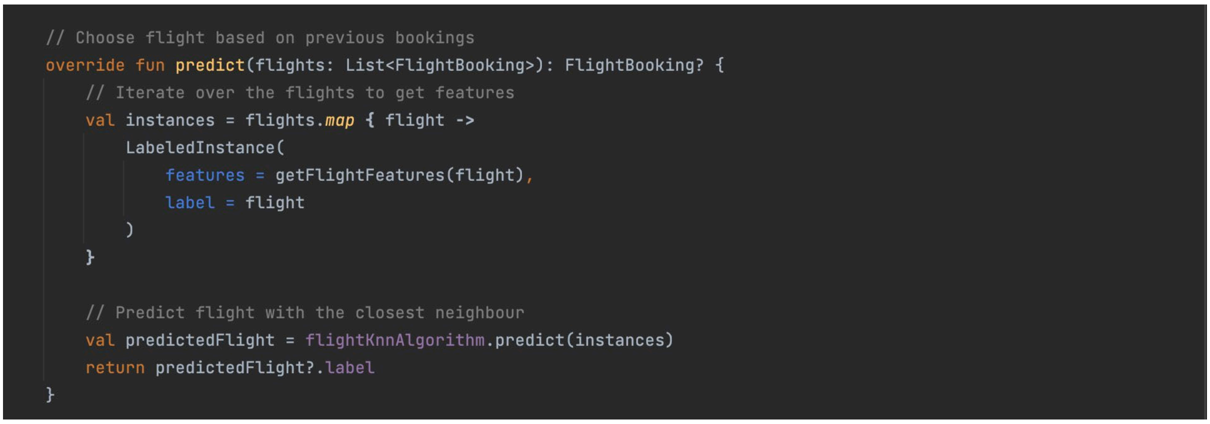
However, the KNN algorithm has been modified to use 1-nearest neighbour (1-NN) to choose the instance with the single nearest neighbour, as all instances are the same class, and the main goal is to find the flight or hotel closest to an existing booking (Figure 10).



```kotlin
fun train(instances: List<LabeledInstance<T>>) {
    trainInstances = instances
}

fun predict(instances: List<LabeledInstance<T>>): LabeledInstance<T>? {
    if (trainInstances.isEmpty()) {
        throw IllegalStateException("The model has not been trained. Call the train() method first.")
    }

    val nearestNeighbours = instances.map { instance ->
        val distances = trainInstances.map { trainInstance ->
            val distance = calculateDistance(instance.features, trainInstance.features)
            Distance(distance, trainInstance.label) ^map
        }.sortedBy { it.distance }

        val closestNeighbour = distances.first()
        NearestNeighbour(instance, closestNeighbour) ^map
    }

    // Return the instance with the closest neighbour
    return nearestNeighbours.minByOrNull { it.nearestNeighbour.distance }?.instance
}
```

Figure 10 - KNN Algorithm Implementation

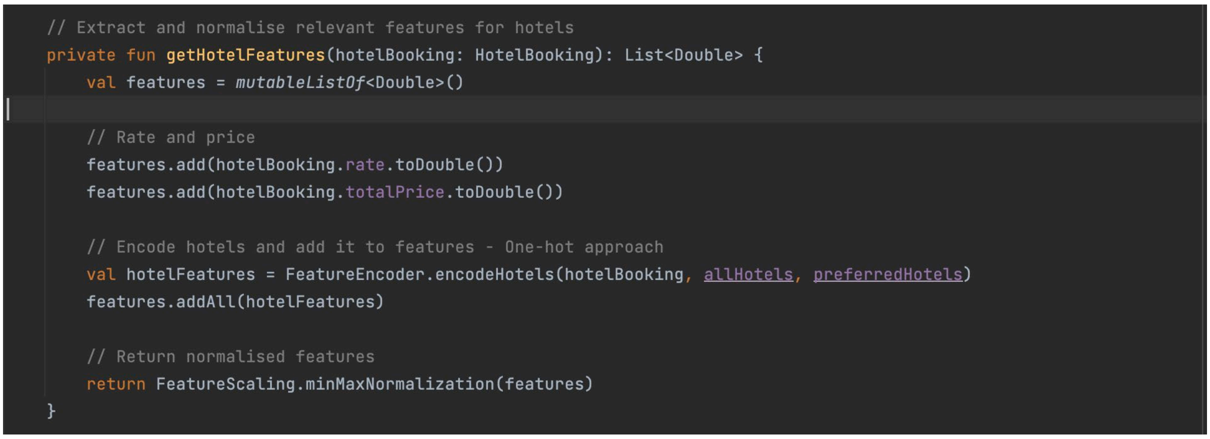
Once the algorithm has been trained with previous booking data, the prediction function extracts the features of each flight or hotel to create a list of labelled instances. These instances are passed to the KNN algorithm to predict the one closest to an existing flight or hotel (Figure 11).



```kotlin
// Choose flight based on previous bookings
override fun predict(flights: List<FlightBooking>): FlightBooking? {
    // Iterate over the flights to get features
    val instances = flights.map { flight ->
        LabeledInstance(
            features = getFlightFeatures(flight),
            label = flight
        )
    }

    // Predict flight with the closest neighbour
    val predictedFlight = flightKnnAlgorithm.predict(instances)
    return predictedFlight?.label
}
```

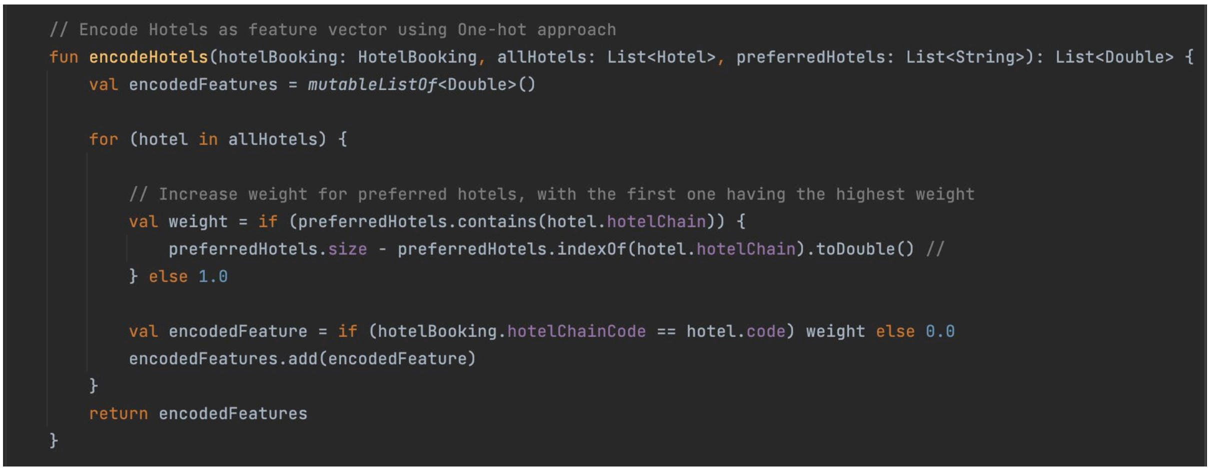
Figure 11 - Flight Prediction Example

When extracting features, each feature is encoded and normalised using numerical values. The weight of preferred airlines and hotels is increased to influence the prediction (Figure 12).



```
// Extract and normalise relevant features for hotels
private fun getHotelFeatures(hotelBooking: HotelBooking): List<Double> {
    val features = mutableListOf<Double>()

    // Rate and price
    features.add(hotelBooking.rate.toDouble())
    features.add(hotelBooking.totalPrice.toDouble())

    // Encode hotels and add it to features - One-hot approach
    val hotelFeatures = FeatureEncoder.encodeHotels(hotelBooking, allHotels, preferredHotels)
    features.addAll(hotelFeatures)

    // Return normalised features
    return FeatureScaling.minMaxNormalization(features)
}
```

Figure 12 - Hotel Feature Extraction

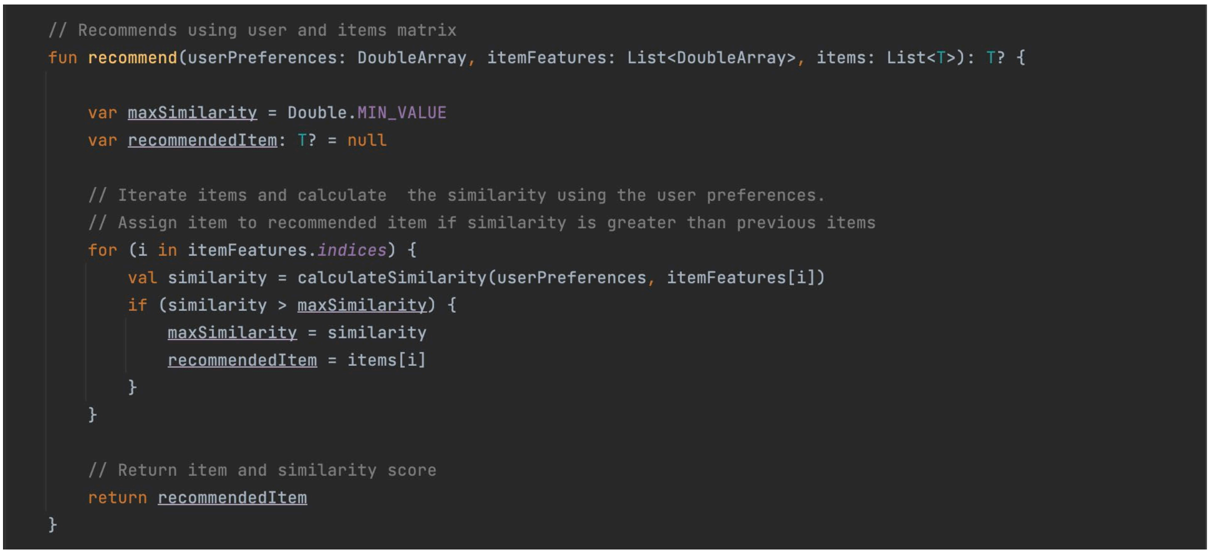


```
// Encode Hotels as feature vector using One-hot approach
fun encodeHotels(hotelBooking: HotelBooking, allHotels: List<Hotel>, preferredHotels: List<String>): List<Double> {
    val encodedFeatures = mutableListOf<Double>()

    for (hotel in allHotels) {

        // Increase weight for preferred hotels, with the first one having the highest weight
        val weight = if (preferredHotels.contains(hotel.hotelChain)) {
            preferredHotels.size - preferredHotels.indexOf(hotel.hotelChain).toDouble() //
        } else 1.0

        val encodedFeature = if (hotelBooking.hotelChainCode == hotel.code) weight else 0.0
        encodedFeatures.add(encodedFeature)
    }
    return encodedFeatures
}
```

Figure 13 - Hotel Encoding

2.7.4 Content-Base Algorithm

Predicting an itinerary using the KNN algorithm mentioned above requires data to train the algorithm, which raises a challenge for new users with previous bookings.



```
// Recommends using user and items matrix
fun recommend(userPreferences: DoubleArray, itemFeatures: List<DoubleArray>, items: List<T>): T? {

    var maxSimilarity = Double.MIN_VALUE
    var recommendedItem: T? = null

    // Iterate items and calculate  the similarity using the user preferences.
    // Assign item to recommended item if similarity is greater than previous items
    for (i in itemFeatures.indices) {
        val similarity = calculateSimilarity(userPreferences, itemFeatures[i])
        if (similarity > maxSimilarity) {
            maxSimilarity = similarity
            recommendedItem = items[i]
        }
    }

    // Return item and similarity score
    return recommendedItem
}
```

Figure 14 - Content-Based Algorithm Implementation

To overcome this problem, a content-based algorithm is used to predict itineraries when users have fewer than two previous bookings. Content-based algorithms use item features to recommend other items similar to the user preferences, which makes it ideal for this use case (Figure 14).

The user preferences and each one of the items used for prediction are encoded into a numerical value using a one-hot approach to create a matrix. The user preferences are then compared with each item to calculate their similarity using the dot product.

	Air France	Aer Lingus	Ryanair	Iberia	United
Preferences	0	2	1	0	0
Flight #1	0	0	0	1	0
Flight #2	0	1	0	1	0

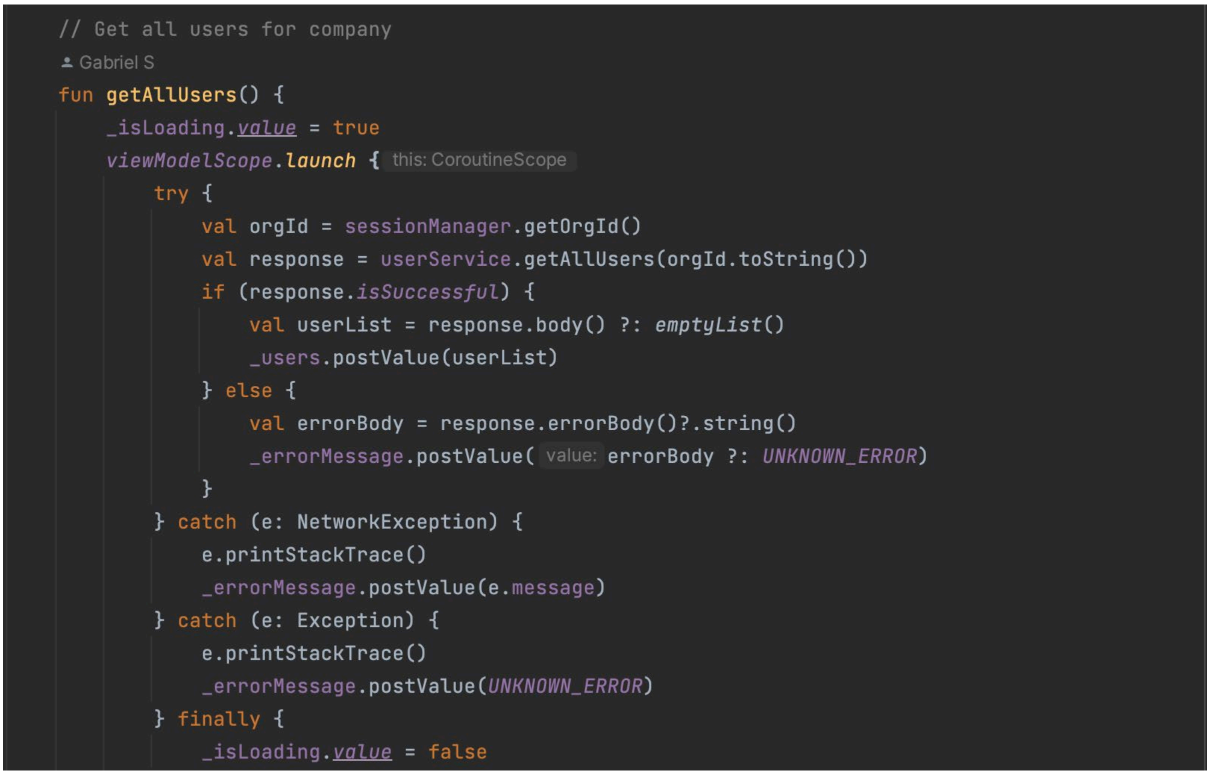
$$sim(pre, F1) = (0 * 0) + (2 * 0) + (1 * 0) + (0 *) = 0$$
$$sim(pre, F2) = (0 * 0) + (2 * 1) + (1 * 0) + (0 *) = \mathbf{2}$$

The table above shows how flight results are compared to preferences to determine their similarity. In this case, flight #2 is one of the user's preferred airlines and the predicted flight.

2.7.5 User Management

The user management implementation consists of a series of functions to allow an administrator to view, add, edit and delete other users in their company.

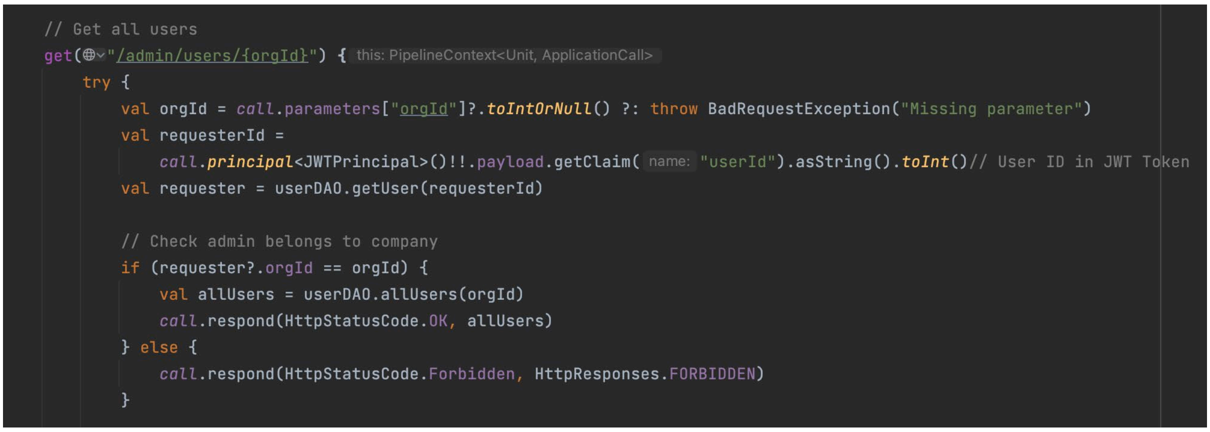
When the administrator visits the Users fragment, a call is made through the user view model to get all users. This call is made in the onResume() call-back to guarantee new data is available when the fragment is resumed or becomes visible for the first time (Figure 14).



```
// Get all users for company
Gabriel S
fun getAllUsers() {
    _isLoading.value = true
    viewModelScope.launch { this: CoroutineScope
        try {
            val orgId = sessionManager.getOrgId()
            val response = userService.getAllUsers(orgId.toString())
            if (response.isSuccessful) {
                val userList = response.body() ?: emptyList()
                _users.postValue(userList)
            } else {
                val errorBody = response.errorBody()?.string()
                _errorMessage.postValue( value: errorBody ?: UNKNOWN_ERROR)
            }
        } catch (e: NetworkException) {
            e.printStackTrace()
            _errorMessage.postValue(e.message)
        } catch (e: Exception) {
            e.printStackTrace()
            _errorMessage.postValue(UNKNOWN_ERROR)
        } finally {
            _isLoading.value = false
```

Figure 15 - getAllUser() function in User View Model

On the server side, the route validates the organisation ID parameter, extracts the requestor's user ID from the JWT token payload and finally checks if the administrator belongs to the same company before returning all users for that specific company to the client (Figure 15). This route is authenticated using Ktor's authentication plugin.

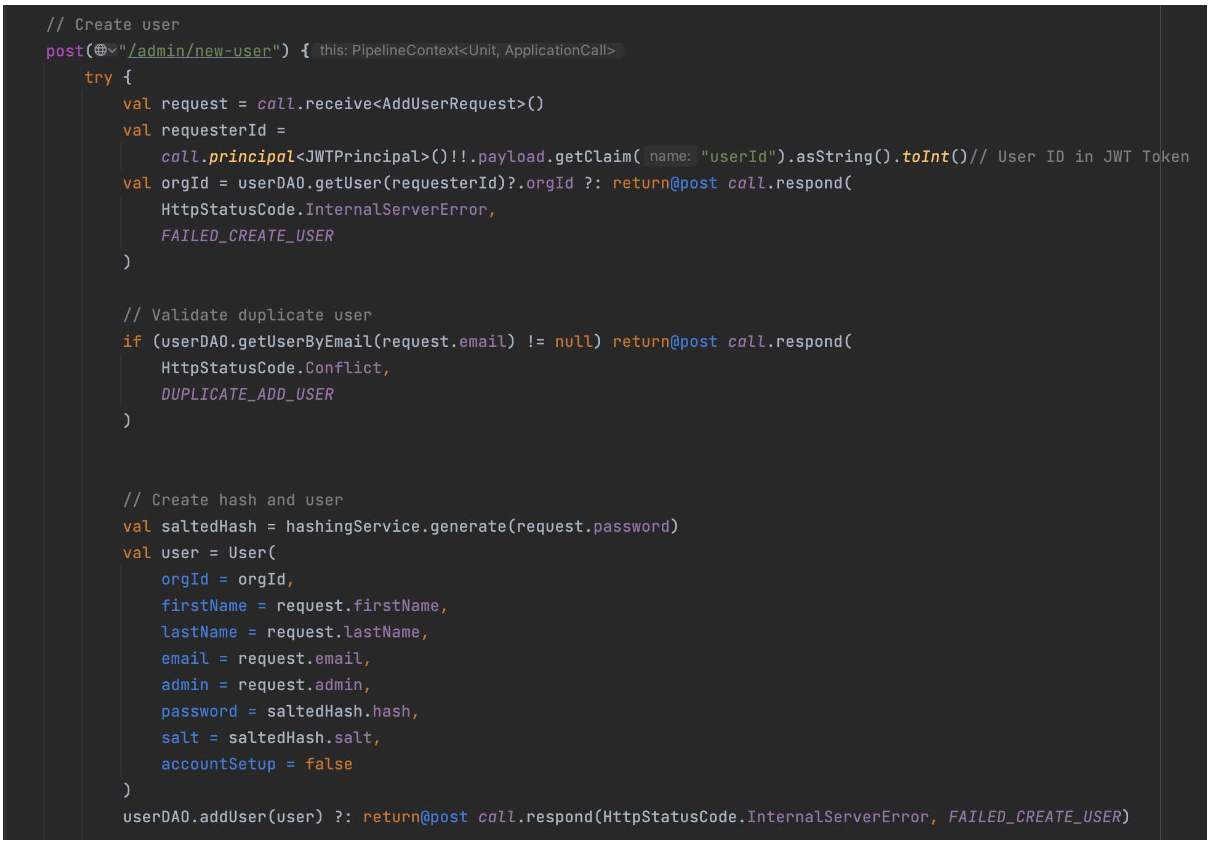


```
// Get all users
get("/admin/users/{orgId}") { this: PipelineContext<Unit, ApplicationCall>
    try {
        val orgId = call.parameters["orgId"]?.toIntOrNull() ?: throw BadRequestException("Missing parameter")
        val requesterId =
            call.principal<JWTPrincipal>()!!.payload.getClaim( name: "userId").asString().toInt()// User ID in JWT Token
        val requester = userDAO.getUser(requesterId)

        // Check admin belongs to company
        if (requester?.orgId == orgId) {
            val allUsers = userDAO.allUsers(orgId)
            call.respond(HttpStatusCode.OK, allUsers)
        } else {
            call.respond(HttpStatusCode.Forbidden, HttpResponses.FORBIDDEN)
        }
```

Figure 16 - Get All Users Route in API

The users are displayed using a RecyclerView with a custom adapter allowing to reuse the items and filter results using the Filterable class. The adapter also adds an on-click listener to each item using a custom interface to make the item clickable and access the correct user profile.

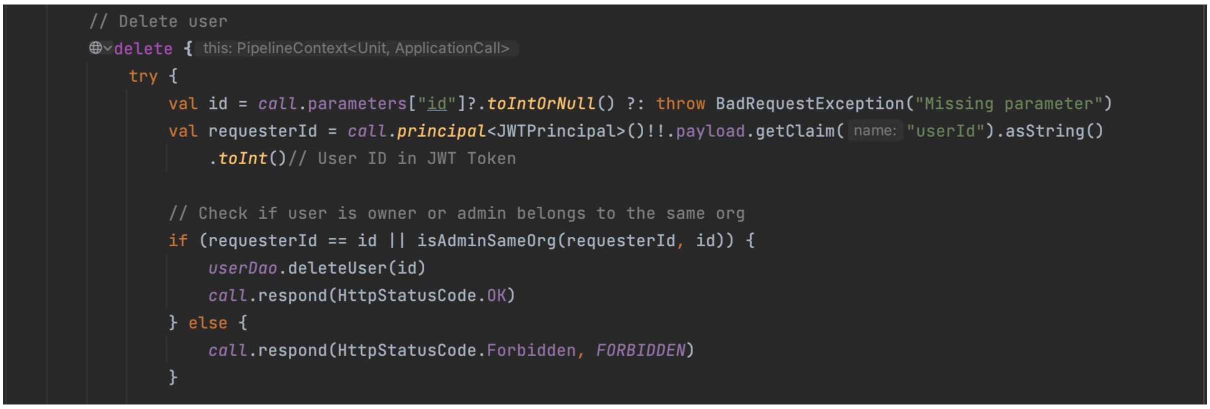
The administrator can also add new users. When they click the button, they are redirected to the new user fragment, which contains the form to add a new user. All the details are validated before submitting the form, and the API will once again authenticate the request and validate the user ID in the JWT token before adding the data to the database (Figure 16).



```
// Create user
post("/admin/new-user") { this: PipelineContext<Unit, ApplicationCall>
    try {
        val request = call.receive<AddUserRequest>()
        val requesterId =
            call.principal<JWTPrincipal>()!!.payload.getClaim( name: "userId").asString().toInt()// User ID in JWT Token
        val orgId = userDAO.getUser(requesterId)?.orgId ?: return@post call.respond(
            HttpStatusCode.InternalServerError,
            FAILED_CREATE_USER
        )

        // Validate duplicate user
        if (userDAO.getUserByEmail(request.email) != null) return@post call.respond(
            HttpStatusCode.Conflict,
            DUPLICATE_ADD_USER
        )


        // Create hash and user
        val saltedHash = hashingService.generate(request.password)
        val user = User(
            orgId = orgId,
            firstName = request.firstName,
            lastName = request.lastName,
            email = request.email,
            admin = request.admin,
            password = saltedHash.hash,
            salt = saltedHash.salt,
            accountSetup = false
        )
        userDAO.addUser(user) ?: return@post call.respond(HttpStatusCode.InternalServerError, FAILED_CREATE_USER)
```

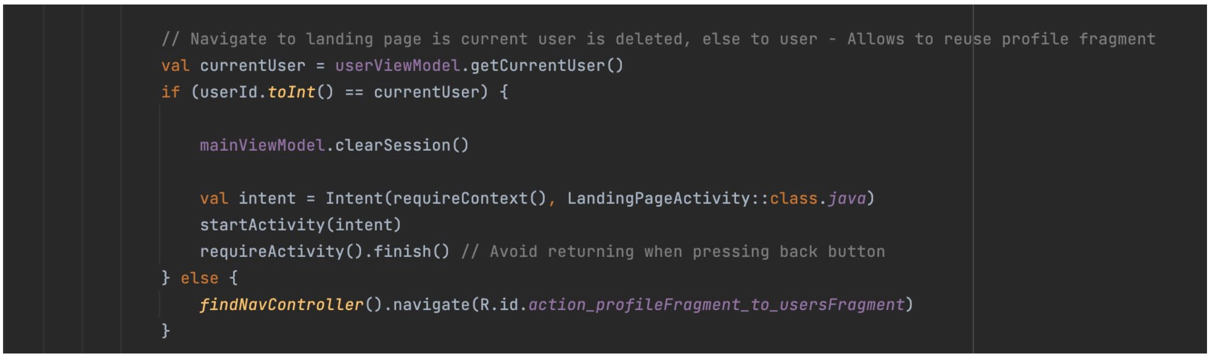
Figure 17 - New User Route in API

To delete a user, the system displays a dialogue to the user before sending a request to the API. The API validates the user ID in the call parameters and the JWT token deletes the user and sends a confirmation to the client (Figure 17). This option is also available to a user deleting their account; in such case, the system redirects the user to the landing page after deleting the account (Figure 18).



```
// Delete user
delete { this: PipelineContext<Unit, ApplicationCall>
    try {
        val id = call.parameters["id"]?.toIntOrNull() ?: throw BadRequestException("Missing parameter")
        val requesterId = call.principal<JWTPrincipal>()!!.payload.getClaim( name: "userId").asString()
            .toInt()// User ID in JWT Token

        // Check if user is owner or admin belongs to the same org
        if (requesterId == id || isAdminSameOrg(requesterId, id)) {
            userDao.deleteUser(id)
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(HttpStatusCode.Forbidden, FORBIDDEN)
        }
```

Figure 18 - Delete User Route in API



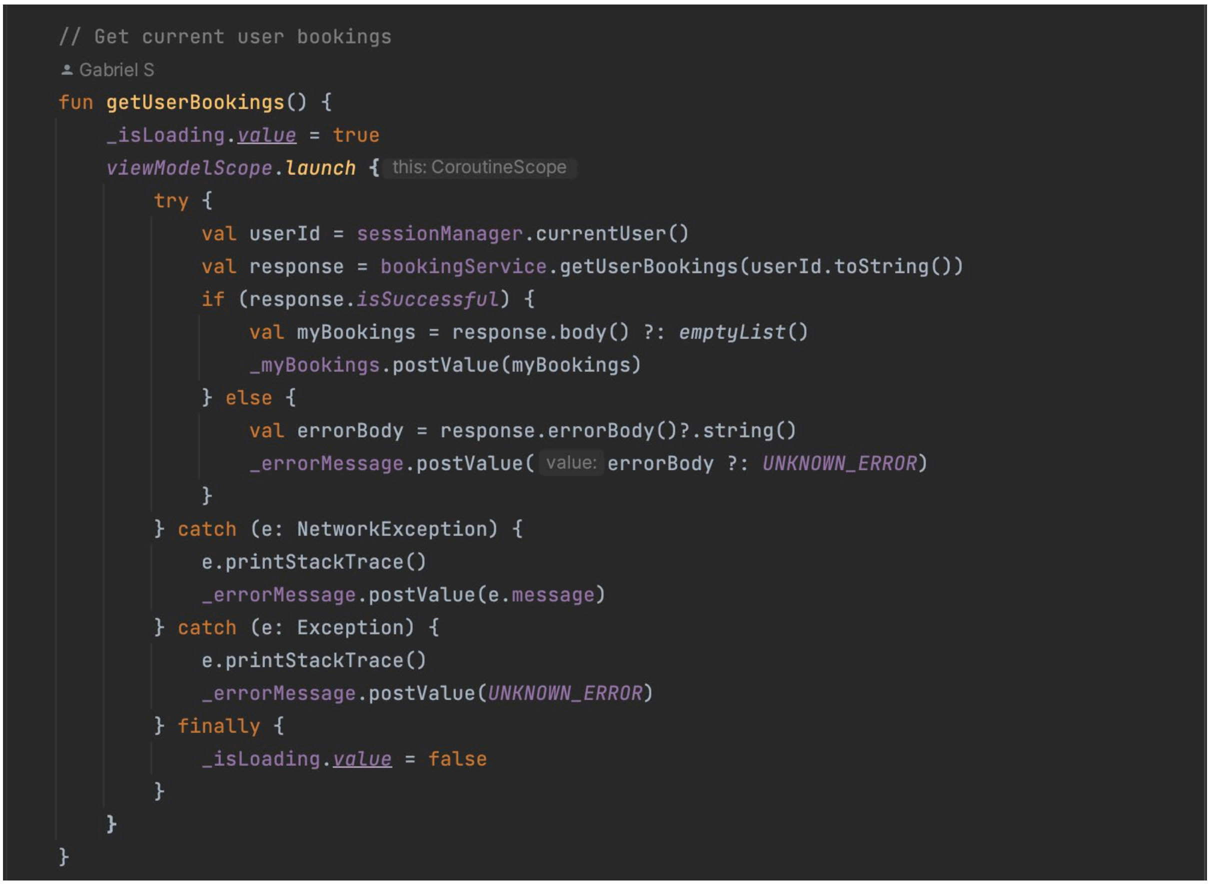
```kotlin
// Navigate to landing page is current user is deleted, else to user - Allows to reuse profile fragment
val currentUser = userViewModel.getCurrentUser()
if (userId.toInt() == currentUser) {

    mainViewModel.clearSession()

    val intent = Intent(requireContext(), LandingPageActivity::class.java)
    startActivity(intent)
    requireActivity().finish() // Avoid returning when pressing back button
} else {
    findNavController().navigate(R.id.action_profileFragment_to_usersFragment)
}
```

Figure 19 - Delete User Implementation in Fragment

2.7.6 Booking Management

The booking management implementation includes several functions to view and cancel bookings.

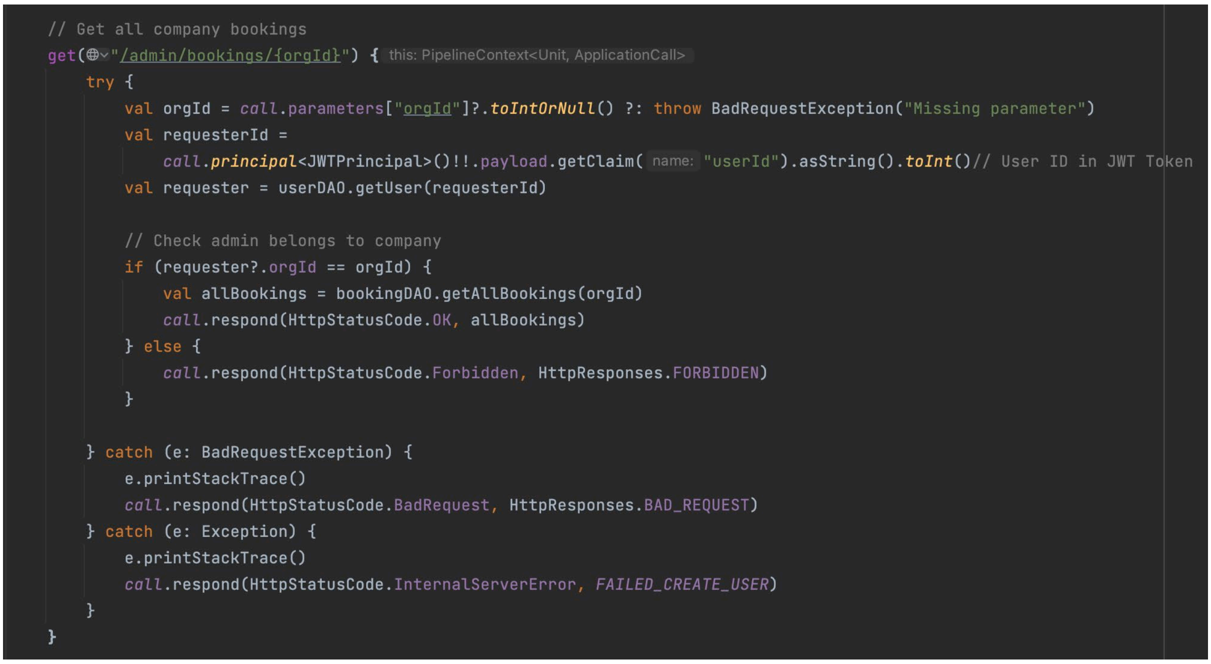
When a user visits the Bookings fragment, a call is made to request all user's bookings. Bookings are displayed to the user using a RecyclerView with a custom adapter, reusing the booking card item and adding an image to the booking (Figure 19).



```kotlin
// Get current user bookings
Gabriel S
fun getUserBookings() {
    _isLoading.value = true
    viewModelScope.launch { this: CoroutineScope
        try {
            val userId = sessionManager.currentUser()
            val response = bookingService.getUserBookings(userId.toString())
            if (response.isSuccessful) {
                val myBookings = response.body() ?: emptyList()
                _myBookings.postValue(myBookings)
            } else {
                val errorBody = response.errorBody()?.string()
                _errorMessage.postValue( value: errorBody ?: UNKNOWN_ERROR)
            }
        } catch (e: NetworkException) {
            e.printStackTrace()
            _errorMessage.postValue(e.message)
        } catch (e: Exception) {
            e.printStackTrace()
            _errorMessage.postValue(UNKNOWN_ERROR)
        } finally {
            _isLoading.value = false
        }
    }
}
```

Figure 20 - Get All User Bookings Implementation in View Model

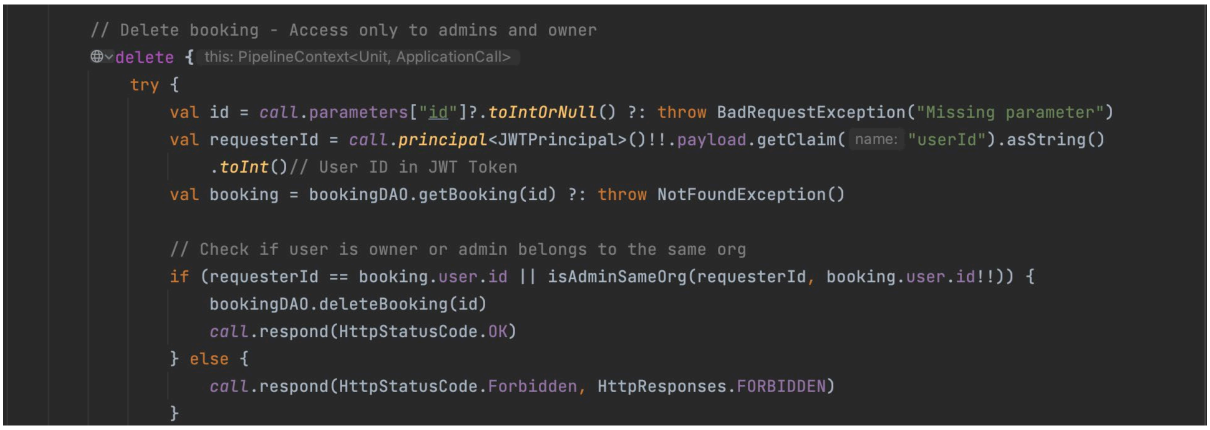
Administrators can visit the all bookings fragment, which calls the API to obtain all the bookings for a company. The API verifies the organisation ID parameter and user ID in the JWT before returning the response to the client (Figure 20).



```
// Get all company bookings
get("/admin/bookings/{orgId}") { this: PipelineContext<Unit, ApplicationCall>
    try {
        val orgId = call.parameters["orgId"]?.toIntOrNull() ?: throw BadRequestException("Missing parameter")
        val requesterId =
            call.principal<JWTPrincipal>()!!.payload.getClaim( name: "userId").asString().toInt()// User ID in JWT Token
        val requester = userDAO.getUser(requesterId)

        // Check admin belongs to company
        if (requester?.orgId == orgId) {
            val allBookings = bookingDAO.getAllBookings(orgId)
            call.respond(HttpStatusCode.OK, allBookings)
        } else {
            call.respond(HttpStatusCode.Forbidden, HttpResponses.FORBIDDEN)
        }

    } catch (e: BadRequestException) {
        e.printStackTrace()
        call.respond(HttpStatusCode.BadRequest, HttpResponses.BAD_REQUEST)
    } catch (e: Exception) {
        e.printStackTrace()
        call.respond(HttpStatusCode.InternalServerError, FAILED_CREATE_USER)
    }
}
```

Figure 21 - Get Company Bookings Route in API

Both staff users and administrators can cancel a booking. When the user clicks the cancel button, a dialogue is displayed, and the request is sent to the API upon confirmation. The route in the API validates the parameter and requestor before deleting the booking (Figure 21).



```
// Delete booking - Access only to admins and owner
delete { this: PipelineContext<Unit, ApplicationCall>
    try {
        val id = call.parameters["id"]?.toIntOrNull() ?: throw BadRequestException("Missing parameter")
        val requesterId = call.principal<JWTPrincipal>()!!.payload.getClaim( name: "userId").asString()
            .toInt()// User ID in JWT Token
        val booking = bookingDAO.getBooking(id) ?: throw NotFoundException()

        // Check if user is owner or admin belongs to the same org
        if (requesterId == booking.user.id || isAdminSameOrg(requesterId, booking.user.id!!)) {
            bookingDAO.deleteBooking(id)
            call.respond(HttpStatusCode.OK)
        } else {
            call.respond(HttpStatusCode.Forbidden, HttpResponses.FORBIDDEN)
        }
```

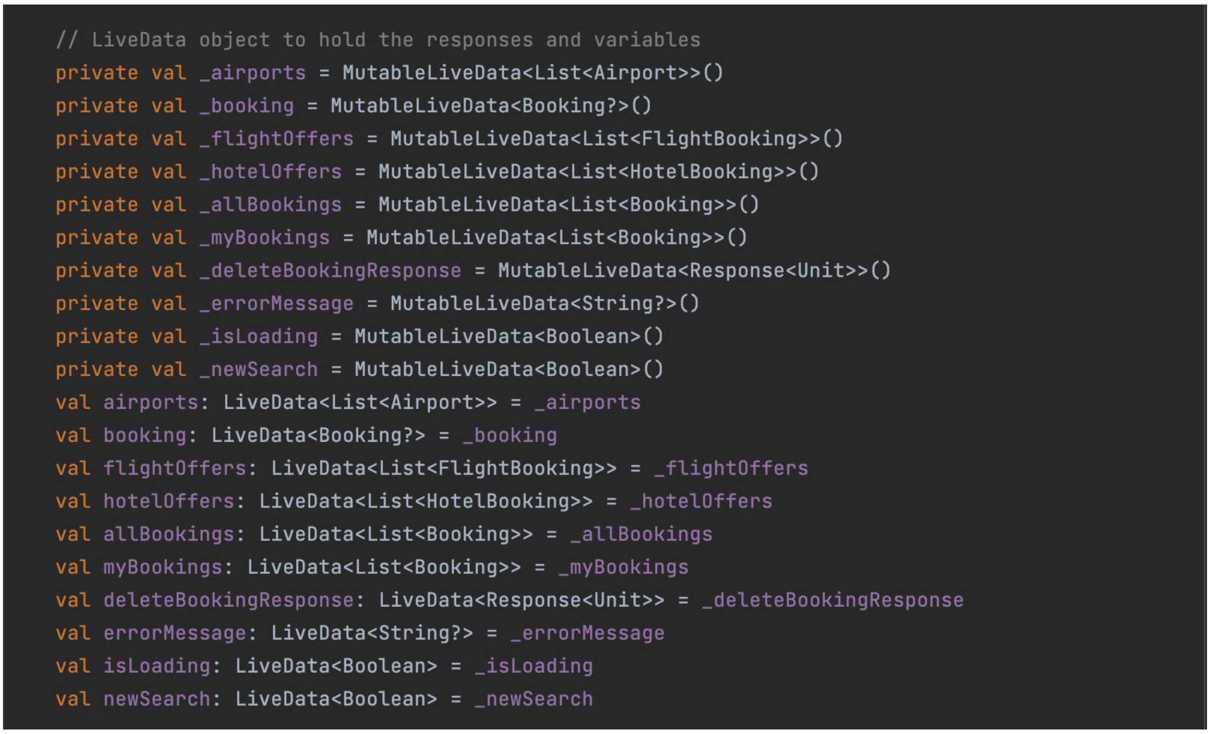
Figure 22 - Delete Booking Route in API

2.7.7 Live Data

LiveData is an observable data class which notifies its observer every time data changes (Google Android for Developers, 2023).

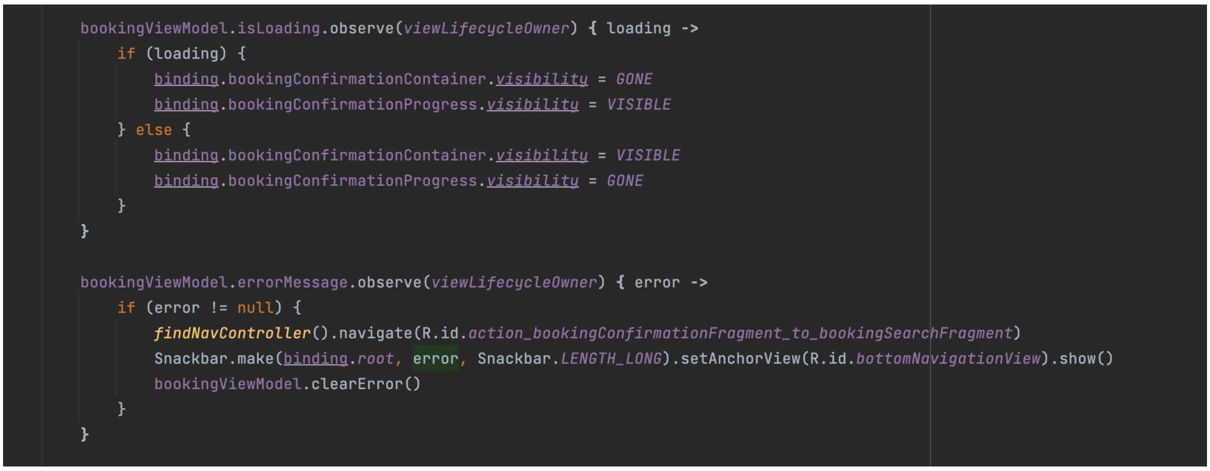
This is used throughout the app to notify activities and fragments about data changes and update the UI accordingly. In the examples below, the fragments observe the LiveData in the view models to display a loading animation when calls are processing and show errors to the user (Figure 15).

Each data object has a private variable used in the View Model, which is then assigned to a read-only public variable that can be observed by the activities and fragments and guarantees data integrity (Figure 14).



```
// LiveData object to hold the responses and variables
private val _airports = MutableLiveData<List<Airport>>()
private val _booking = MutableLiveData<Booking?>()
private val _flightOffers = MutableLiveData<List<FlightBooking>>()
private val _hotelOffers = MutableLiveData<List<HotelBooking>>()
private val _allBookings = MutableLiveData<List<Booking>>()
private val _myBookings = MutableLiveData<List<Booking>>()
private val _deleteBookingResponse = MutableLiveData<Response<Unit>>()
private val _errorMessage = MutableLiveData<String?>()
private val _isLoading = MutableLiveData<Boolean>()
private val _newSearch = MutableLiveData<Boolean>()
val airports: LiveData<List<Airport>> = _airports
val booking: LiveData<Booking?> = _booking
val flightOffers: LiveData<List<FlightBooking>> = _flightOffers
val hotelOffers: LiveData<List<HotelBooking>> = _hotelOffers
val allBookings: LiveData<List<Booking>> = _allBookings
val myBookings: LiveData<List<Booking>> = _myBookings
val deleteBookingResponse: LiveData<Response<Unit>> = _deleteBookingResponse
val errorMessage: LiveData<String?> = _errorMessage
val isLoading: LiveData<Boolean> = _isLoading
val newSearch: LiveData<Boolean> = _newSearch
```

Figure 23 - LiveData private variables and read-only variables



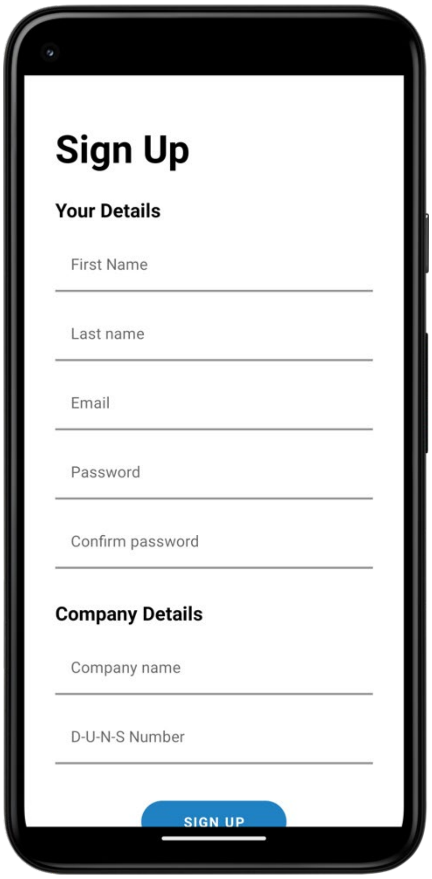
```
bookingViewModel.isLoading.observe(viewLifecycleOwner) { loading ->
    if (loading) {
        binding.bookingConfirmationContainer.visibility = GONE
        binding.bookingConfirmationProgress.visibility = VISIBLE
    } else {
        binding.bookingConfirmationContainer.visibility = VISIBLE
        binding.bookingConfirmationProgress.visibility = GONE
    }
}

bookingViewModel.errorMessage.observe(viewLifecycleOwner) { error ->
    if (error != null) {
        findNavController().navigate(R.id.action_bookingConfirmationFragment_to_bookingSearchFragment)
        Snackbar.make(binding.root, error, Snackbar.LENGTH_LONG).setAnchorView(R.id.bottomNavigationView).show()
        bookingViewModel.clearError()
    }
}
```

Figure 24 - Observer example

2.7.8 Deployment

The API was deployed using an EC2 instance using a fat JAR file containing all project files and dependencies. A service was created to run the application even when the terminal is closed (Mukadam, 2021).

2.8 Graphical User Interface (GUI)



Sign Up

Your Details

First Name

Last name

Email

Password

Confirm password

Company Details

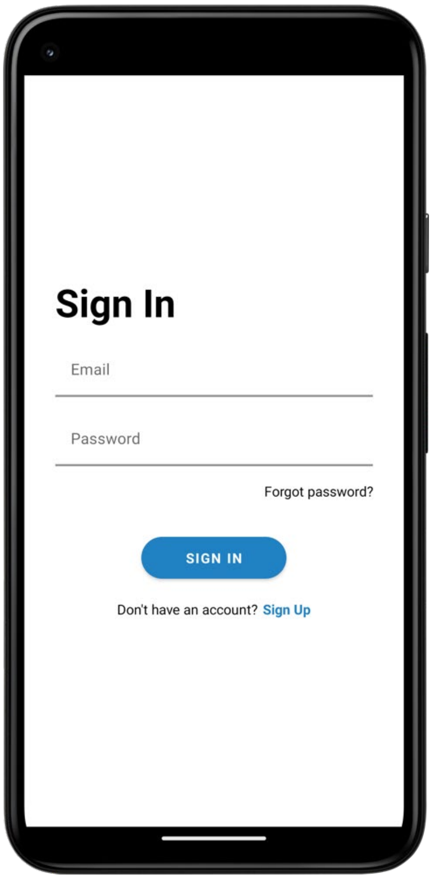
Company name

D-U-N-S Number

Sign up Screen

Displays sign-up form allowing new users to create an account.

All inputs are validated before submitting the form and clear errors are displayed to the user.



Sign In

Email

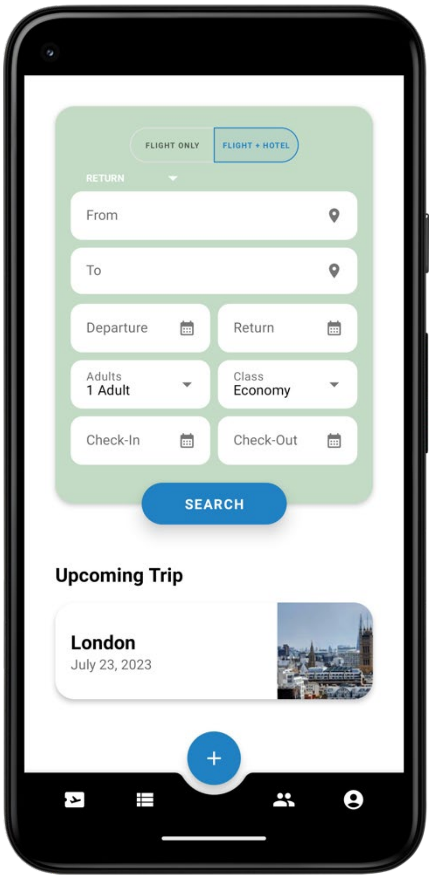
Password

Sign In Screen

Displays the options to sign in for registered users. It also shows an option to navigate to the Sign Up screen if the user does not have an account.

Main Screen

Displays a search box to allow users to enter their travel details. It also displays an upcoming booking, if available.



FLIGHT ONLY | FLIGHT + HOTEL

RETURN

From

To

Departure | Return

Adults: 1 Adult | Class: Economy

Check-In | Check-Out

SEARCH

Upcoming Trip

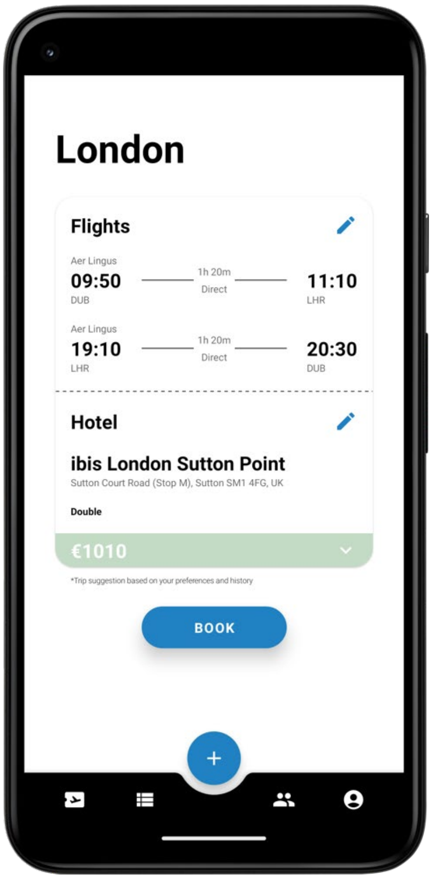
London
July 23, 2023

+

Predicted Booking Screen

Displays an entire itinerary based on the user's preferences and previous bookings, removing the need to select each item separately and allowing users to book their trip with one click.

The users also have the option to edit any of the suggestions. The new selections will help train the algorithm and improve future predictions.



London

Flights

Aer Lingus
09:50 DUB — 1h 20m Direct — 11:10 LHR

Aer Lingus
19:10 LHR — 1h 20m Direct — 20:30 DUB

Hotel

ibis London Sutton Point
Sutton Court Road (Stop M), Sutton SM1 4FG, UK

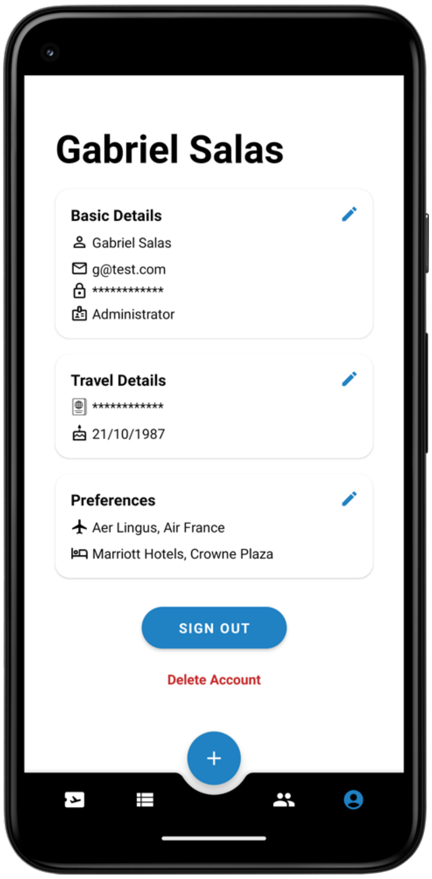
Double

€1010

*Trip suggestion based on your preferences and history

BOOK

+



Gabriel Salas

Basic Details

Gabriel Salas

g@test.com

****

Administrator

Travel Details

****

21/10/1987

Preferences

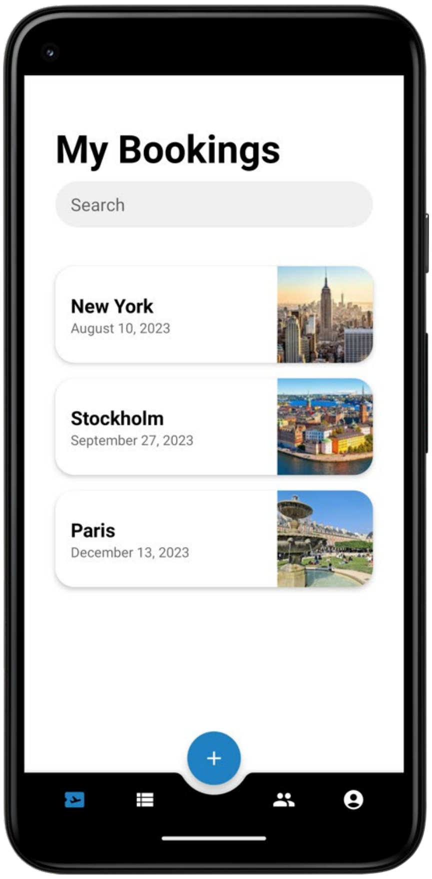
Aer Lingus, Air France

Marriott Hotels, Crowne Plaza

Profile Screen

Displays the user’s information, such as name, email, travel data and preferences.

Users have the option to edit any of these details or delete their accounts.



My Bookings

New York
August 10, 2023

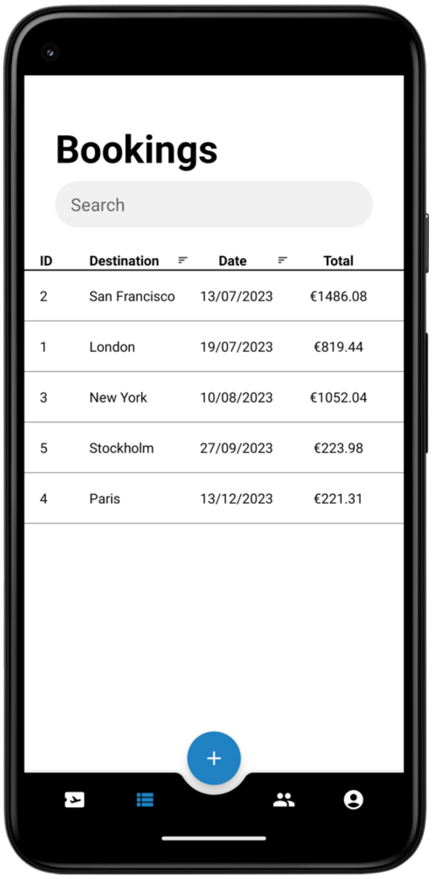
Stockholm
September 27, 2023

Paris
December 13, 2023

My Bookings Screen

Displays the upcoming bookings for the current user. The search bar can be used to search for specific destinations and booking IDs.

Each booking displays an image using Google Places API.



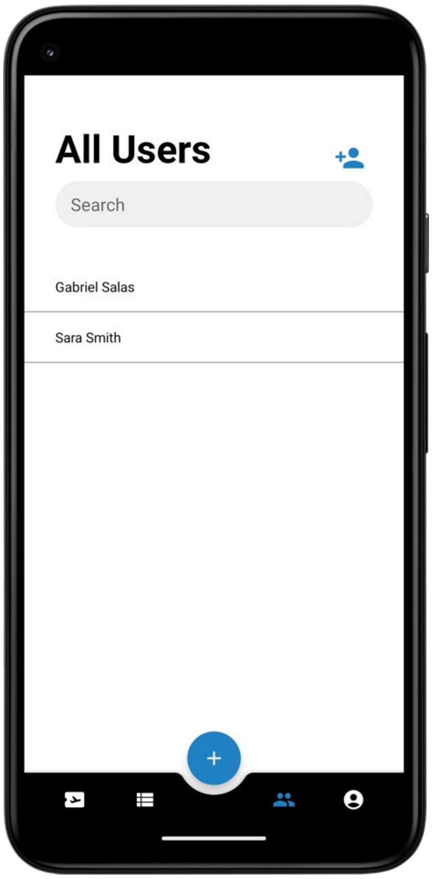
Bookings

Search

ID	Destination	Date	Total
2	San Francisco	13/07/2023	€1486.08
1	London	19/07/2023	€819.44
3	New York	10/08/2023	€1052.04
5	Stockholm	27/09/2023	€223.98
4	Paris	13/12/2023	€221.31

Bookings Screen

Administrators can view all the bookings for their company. The search bar allows administrators to search for destinations, IDs and countries.



All Users

Search

Gabriel Salas

Sara Smith

All Users Screen

Displays all the users to an administrator. It has the option to add new users to the account and search users by name and email address.

2.9 Testing

The project has been developed using a test-driven development approach. A set of unit, integration and functionals test were written to guide the implementation process.

2.9.1 Objectives

- Guide the development process. Following the test-driven development approach, each test is run before implementation, playing a fundamental part in the development process.
- Verify that all requirements are met and all functionalities are working as expected.
- Identify any bugs and security vulnerabilities.

2.9.2 Scope

- **Unit testing:** it focuses on the individual components of the application, such as the models used to store data, the serialization functions, view models logic and the machine learning algorithms.
- **Integration testing:** targets the interaction between different elements of the application. It includes the different services, external API calls and functions that interact with the database.
- **Functional testing:** it focuses on testing the main requirements of the application as a whole. This included the main requirements and use cases.
- **Manual beta testing:** focuses on testing the UI/UX, identifying any bugs and gathering feedback before releasing the project. This will be performed by a small group of testers (family and friends).

2.9.3 Testing Tools

- **Junit:** this is the main testing framework. It allows us to set up the testing environment and verify results using assertions.
- **MockK:** this Kotlin testing library allows you to mock the elements used in the Unit testing, isolating the test from the rest of the code.

2.9.4 Test Cases

The following test cases were used to guide manual testing using Android Emulator. Each test was performed by a small group of 5 beta testers.

ID	1	**Priority**	High
Description	The user attempts to sign up with a positive outcome		
Test Data	• First name: John • Last name: Smith • Email: john@test.com • Password: Pass1234! • Company name: MyCompany LTD		

	• DUNS: 12345678

Steps	Expected Result	Actual Result
1. Click the sign up button on the landing screen	The user is redirected to the main screen and a new user and company is added to the database.	The user was redirected to the database and records were created in the database.
2. Fill out the form using test data		
3. Click the Sign-Up button		

ID	2	Priority	Medium
Description	The user attempts to sign up with a duplicate company		
Test Data	• First name: Sara • Last name: Smith • Email: john@test.com • Password: <any password> • Company name: MyCompany LTD • DUNS: 12345678		

Steps	Expected Result	Actual Result
1. Click the sign up button on the landing screen	An error message is displayed “Duplicate organisation.” The user is asked to sign in.	The message was successfully displayed to the user.
2. Fill out the form using test data		
3. Click the Sign-Up button		

ID	3	Priority	High
Description	The user attempts to sign in using valid credentials		
Test Data	• Email: john@test.com • Password: Pass1234!		

Steps	Expected Result	Actual Result
1. Click the sign in button on the landing screen	The user is redirected to the main screen.	The user was successfully signed in and redirected to the main screen.
2. Enter credentials		
3. Click sign in button		

ID	4	Priority	High
Description	User search books a trip with a flight and hotel		
Test Data	• From: London (LHR) • To: New York (JFK) • Departure/Check-in: 23/11/23 • Return/Checkout: 28/11/23 • Adults: 1 • Class: Economy		

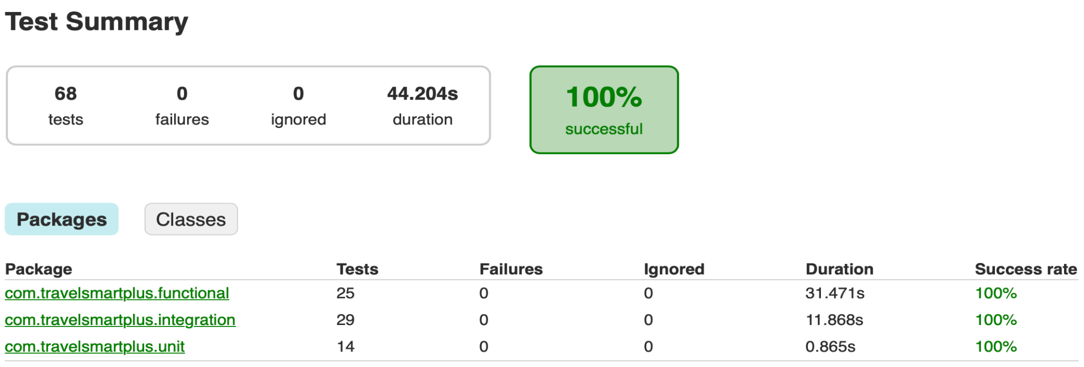
Steps	Expected Result	Actual Result
1. Enter information in the search box	The system displayed the predicted itinerary. When the user clicks Book, the	The itinerary was successfully displayed. The user was redirected to the

2. Click the Search button	user is redirected to the confirmation screen and the booking is added to the database.	confirmation screen and data was added to the database.
3. Click Book button		

ID	5	Priority	High
Description	Add a new user with a Staff role		
Test Data	• First name: Paul • Last name: • Email: p@test.com • Role: Staff • Password: <any password>		

Steps	Expected Result	Actual Result
1. Click User tab	A confirmation message is displayed to the user. The user shows in the user list and the database is updated.	The confirmation message was displayed. The users' list and database were updated to reflect the new user.
2. Click Add User button		
3. Fill out the form with user data		
4. Enter any temporary password		
5. Click Save		

2.9.5 Automated Testing Results

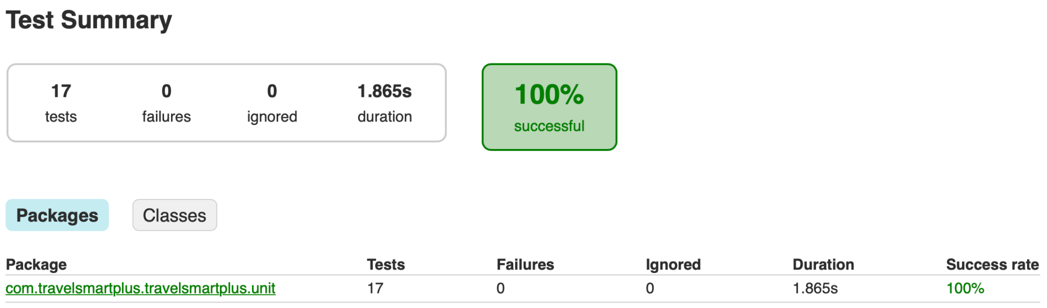


Test Summary

68	0	0	44.204s	100%
tests	failures	ignored	duration	successful

Package	Tests	Failures	Ignored	Duration	Success rate
com.travelsmartplus.functional	25	0	0	31.471s	100%
com.travelsmartplus.integration	29	0	0	11.868s	100%
com.travelsmartplus.unit	14	0	0	0.865s	100%

Figure 25 - Back end Testing Summary



Test Summary

17	0	0	1.865s	100%
tests	failures	ignored	duration	successful

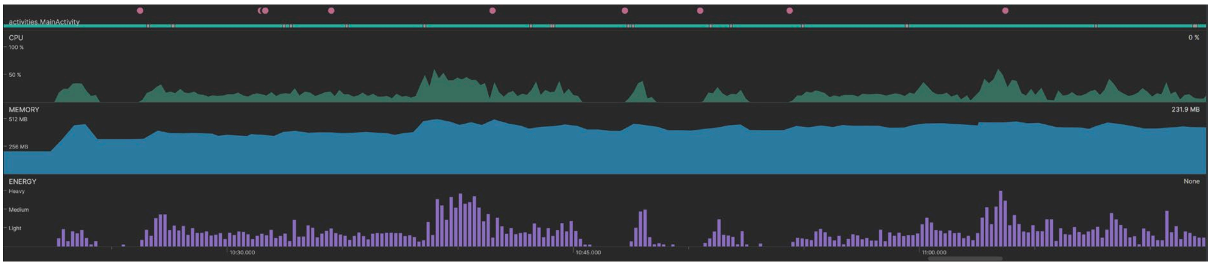
Package	Tests	Failures	Ignored	Duration	Success rate
com.travelsmartplus.travelsmartplus.unit	17	0	0	1.865s	100%

Figure 26 - Front end Testing Summary

2.10 Evaluation

2.10.1 Performance Evaluation

The Android Profiler was used to measure the performance of CPU, memory and battery usage while performing the main use cases, such as booking a trip, adding a user and viewing all bookings and users.



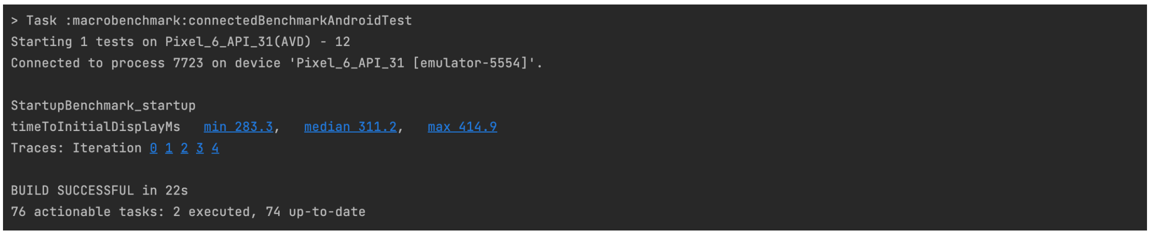
This shows an average CPU usage of 36% and Memory usage of 240MB when performing main tasks. The battery usage peaks at medium when performing tasks like booking search, and low when performing other tasks that use fewer resources.

2.10.2 Network Evaluation

The main network calls were evaluated using the Network Inspector in Android Studio. Each call was made three times to calculate the average time.

Call	1st Run	2nd Run	3rd Run	Average
Sign In	468ms	92ms	81ms	213.66ms
Booking Search	4s 791ms	8s 207ms	6s 964ms	6s 654ms
My Bookings	2s 85ms	331ms	333ms	916ms
All Bookings	336ms	272ms	252ms	286.66ms
All User	113ms	87ms	111ms	103.66ms
User Profile	75ms	108ms	71ms	84.66ms
Add User	141ms	116ms	149ms	135.33ms
Delete User	136ms	84ms	90ms	103.33ms

2.10.3 Startup Benchmark Results



```
> Task :macrobenchmark:connectedBenchmarkAndroidTest
Starting 1 tests on Pixel_6_API_31(AVD) - 12
Connected to process 7723 on device 'Pixel_6_API_31 [emulator-5554]'.

StartupBenchmark_startup
timeToInitialDisplayMs   min 283.3,   median 311.2,   max 414.9
Traces: Iteration 0 1 2 3 4

BUILD SUCCESSFUL in 22s
76 actionable tasks: 2 executed, 74 up-to-date
```

Figure 27 - Startup Macro benchmark

3 Conclusions

One of the main strengths of the application is the ability to predict complete itineraries, which can change how we book a trip by taking advantage of machine learning to remove the need to select each element of our trip manually. This feature is particularly relevant in a world that is becoming more reliant on technology and artificial intelligence, helping us get things done with little effort and interaction.

Concerning the weaknesses, the main limitation is the limited features. Due to time constraints, the application focuses on booking prediction and booking and user management, which can be seen as a disadvantage against other applications in the market.

Overall, I think the application offers a new way of booking a trip and it could be extended to reach the general public and not only business users.

4 Further Development and Research

With additional time and resources, some features could be improved and extended. Currently, it is not possible to process the reservation using the Amadeus API as it requires additional verification and a contract with an airline consolidator, all of which can be obtained with additional resources.

Another area of improvement is the authentication process, as there is no option to revoke the JWT tokens. With more time, a solution could be implemented to securely store tokens in the database, allowing to revoke tokens and increase the security.

The booking prediction is another area that could be improved in the future. The machine learning algorithms can be enhanced to include additional features, such as company policies and partners. Additionally, a new feature can be implemented to offer live suggestions during a trip, such as car rentals, car-sharing services, shuttles, airport information, and any other valuable information.

5 References

- Amadeus, n.d. *Amadeus for Developers.* [Online]
  Available at: https://developers.amadeus.com
  [Accessed 2023].
- Google , 2023. *Migrate to the Navigation component.* [Online]
  Available at: https://developer.android.com/guide/navigation/migrate
  [Accessed 2023].
- Google Android for Developers, 2023. *LiveData overview.* [Online]
  Available at: https://developer.android.com/topic/libraries/architecture/livedata
  [Accessed 2023].
- Google, 2023. *Google Maps Platform.* [Online]
  Available at: https://developers.google.com/maps/documentation/places/web-service/overview
  [Accessed 2023].
- Google, 2023. *ViewModel overview.* [Online]
  Available at: https://developer.android.com/topic/libraries/architecture/viewmodel
  [Accessed 2023].
- Google, n.d. *Android for Developers.* [Online]
  Available at: https://developer.android.com
  [Accessed 2023].
- IBM, n.d. *K-Nearest Neighbors Algorithm.* [Online]
  Available at:
  https://www.ibm.com/topics/knn#:~:text=The%20k%20value%20in%20the,as%20its%20single%20nearest%20neighbor.
  [Accessed May 2023].
- JetBrains, 2022. *Exposed.* [Online]
  Available at: https://github.com/JetBrains/Exposed/wiki
  [Accessed 2023].
- Kotlin , n.d. *Kotlin Docs.* [Online]
  Available at: https://kotlinlang.org/docs/home.html
  [Accessed 2023].
- Ktor, 2023. *Ktor Server.* [Online]
  Available at: https://ktor.io/docs/welcome.html
  [Accessed 2023].
- Mukadam, H., 2021. *Deploying Ktor Web Service on AWS EC2 Instance.* [Online]
  Available at: https://dev.to/hsm59/deploying-ktor-web-service-on-aws-ec2-instance-209h
  [Accessed 2023].
- PostgreSQL, n.d. *PostgreSQL.* [Online]
  Available at: https://www.postgresql.org
- Rosidi, N., n.d. *Step-by-Step Guide to Building Content-Based Filtering.* [Online]
  Available at: https://www.stratascratch.com/blog/step-by-step-guide-to-building-content-based-filtering/
- Shakhnarovich, G., Darrell, T. & Indyk, P., 2005. *Nearest-Neighbor Methods in Learning and Vision : Theory and Practice.* s.l.:The MIT Press.

6 Appendices

6.1 Project Proposal

National College of Ireland

Project Proposal

Business Travel Application

20 December 2022

BSc (Honours) in Computing

Software Development

2022/2023

Gabriel Salas Segura | 19104162| x19104162@student.ncirl.ie

Contents

Objectives

I plan to develop a one-stop solution for business travel that uses machine learning to predict and show itineraries and useful suggestions based on the user's preferences and schedule.

The project seeks to achieve the following objectives:

1. Help users save time and effort by providing a user-friendly platform to book and manage flights and hotels following their company's travel policy. The application can use machine learning to learn from the user's travel patterns and preferences to suggest personalised travel itineraries. For example, once the user searches for a city, the app could automatically display a full itinerary that the user can accept with one single click.
2. These itineraries can include other useful information, such as airport lounges, shuttles, restaurants, events and more, and they can update in real-time based on the user's location and any changes in their schedule.
3. The app would learn from other users in the same company or industry to offer better suggestions.

Background

I used to work for a travel management company (TMC), and one of the main issues I found is that most requests are processed manually by travel advisors, which can be time-consuming. Even solutions like TravelPerk or Concur, intended to simplify the booking process, require users to follow many different steps to book a simple trip.

To add to this problem, most solutions only focus on booking flights, hotels and cars but don't provide any suggestions for the rest of the trip, resulting in users having to use other resources to find this information, which can be a problem for someone with a packed schedule.

My idea is to create a one-stop application that uses machine learning to predict and suggest trip itineraries and other useful information, allowing business users to book a trip with a single tap and see information about airports, shuttles, restaurants and more.

State of the Art

Currently, there are travel management companies and applications that allow users to book and manage their business travel, such as BCD Travel, TravelPerk and Concur or in-house travel desks. However, most of these solutions will have travel agents process the requests manually or will require users to choose each flight and hotel one by one, which can be time-consuming.

On top of that, most alternatives only focus on booking flights, hotels and cars, leaving users with the responsibility of searching for other parts of their trip on other platforms.

My project intents to simplify the booking process by automating the tasks of a business travel agent, offering personalised itineraries to business users, including all the other components of a trip that are often ignored by existing solutions.

This application will put the user's needs and satisfaction at the front while respecting companies' policies and limits, giving business travellers all they need for their trip in one place.

Technical Approach

This project will follow an agile and test-driven development approach, focused on delivering small improvements regularly and using automated tests to verify that the code is working as intended and catch errors and bugs in the early stages of development.

The development process would involve the following steps:

- **Define the goals and requirements:** The first step will be to identify the requirements using the project objectives and use cases. This will help to prioritise the features and functionality and break down the project into small pieces or "sprints." Once I have identified all the requirements, I will use Trello to manage the project and keep track of the progress.
- **Use Trello to manage the project:** Once I have identified all the requirements, I will use Trello to manage the project and keep track of the progress.
- **Write automated tests:** As the application is being developed, I will write automated tests for each new feature to ensure that they are working as intended. Tests will be run regularly to catch any errors or bugs.
- **Review and update the project plan:** I will review and update the project plan to ensure that all goals are met. This involves adapting to feedback and making any necessary changes.

Technical Details

I intend to develop an Android application using Kotlin. Kotlin is a modern language that runs on the Java virtual machine and has become quite popular for Android development.

The other technologies I will use are:

- **Server:** I plan to use Ktor to create the server for my application. Ktor is a framework for building asynchronous web applications in Kotlin, making it easy to integrate with the rest of the project.
- **Machine Learning Algorithms:** I plan to use the KNN (k-nearest neighbours) algorithm to offer recommendations to the customers, as it is simple and intuitive.
- **Design patterns:** I will try to implement different design patterns to solve common design problems and save time. Some design patterns that might be useful for this project are the observer, factory and decorator.
- **Libraries:** Some of the libraries I plan to use are:
    - Kovenant: provides a set of asynchronous libraries for Kotlin, which could be useful for making API calls.

    - OKHttp: a simple HTTP client for Kotlin, which could be useful for interacting with web services that provide travel information.

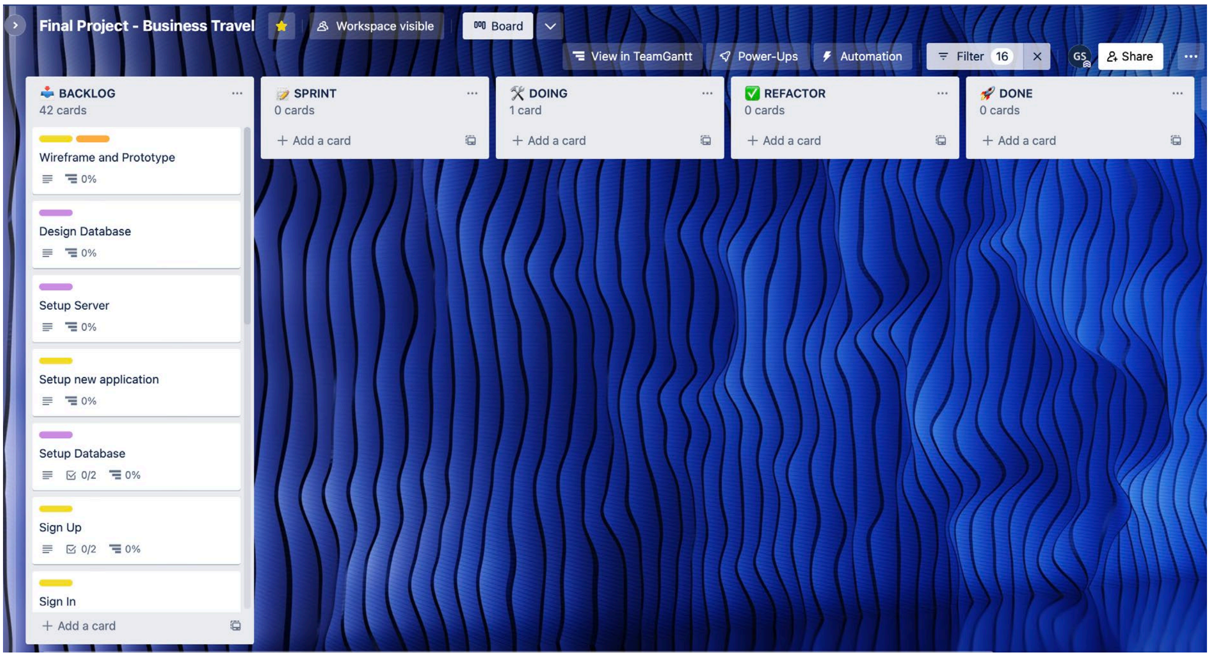
- **Database:** the database will be created using PostgreSQL to simplify the back-end deployment using Heroku.

Special Resources Required

As of now, I am not using any special resources for my application other than Figma to design the user interface.

Project Plan

Using Trello, I have defined the main requirements for the project and added them to the backlog. Some of the requirements will be added to the sprints every 4 weeks, and they will be reviewed and updated as needed:



Final Project - Business Travel

BACKLOG
42 cards

- Wireframe and Prototype — 0%
- Design Database — 0%
- Setup Server — 0%
- Setup new application — 0%
- Setup Database — 0/2 — 0%
- Sign Up — 0/2 — 0%
- Sign In

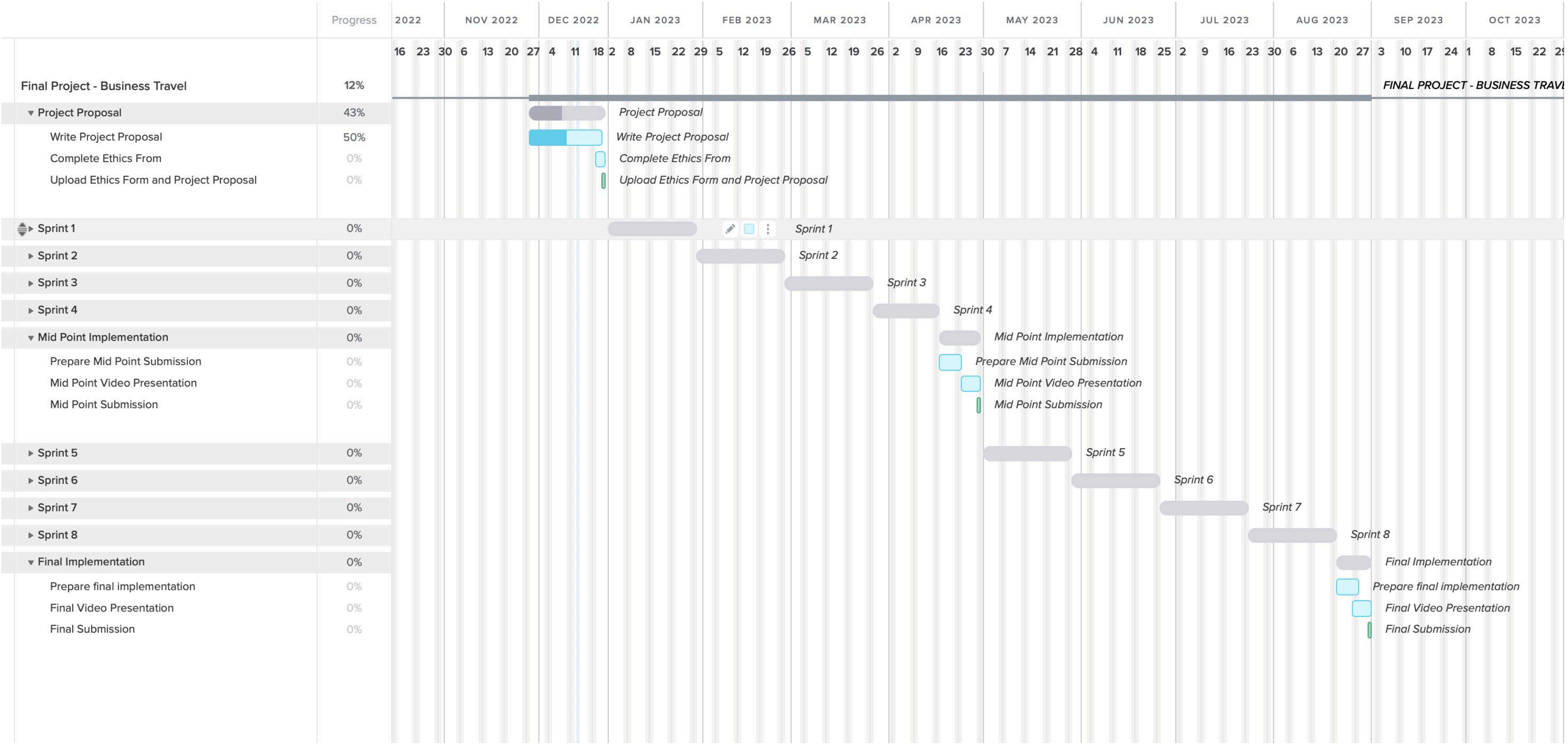
SPRINT
0 cards

DOING
1 card

REFACTOR
0 cards

DONE
0 cards

I have created a project roadmap using a Gantt chart to outline the overall timeline for the project and identify the key milestones and deliverables. Each sprint has 4 weeks, and the task will be added during the planning stage:



	Progress
Final Project - Business Travel	12%
▾ Project Proposal	43%
Write Project Proposal	50%
Complete Ethics From	0%
Upload Ethics Form and Project Proposal	0%
▸ Sprint 1	0%
▸ Sprint 2	0%
▸ Sprint 3	0%
▸ Sprint 4	0%
▾ Mid Point Implementation	0%
Prepare Mid Point Submission	0%
Mid Point Video Presentation	0%
Mid Point Submission	0%
▸ Sprint 5	0%
▸ Sprint 6	0%
▸ Sprint 7	0%
▸ Sprint 8	0%
▾ Final Implementation	0%
Prepare final implementation	0%
Final Video Presentation	0%
Final Submission	0%

Testing

Following a test-driven development approach, I will write different tests before implementing the functions:

- **Unit testing**: test individual units or components of the app to ensure that they are working as intended, such as the functionality of the machine learning model, classes or some of the main methods. This will use Synthetic Data to guarantee privacy and security.

- **Integration testing**: test how well the various units or components of the app work together. This can include testing the integration of the user interface with the other elements and external APIs.

- **Beta testing:** I will deploy the app to a small group of family and friends as a beta test.

6.2 Reflective Journals

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: November

What? This month I defined my idea and presented the project pitch. I also met with my supervisor to discuss the project idea and created an action plan.	
So What? I defined the scope of my project and obtain the necessary feedback to create the project proposal. Part of the feedback was that I should focus on defining specific and complex use cases.	
Now What? I will define the specific use cases of my application and focus on the ones that are more innovative and challenging, and complete my project proposal.	
Student Signature	

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: December

What?

This month was focused on researching the different technologies I was planning to use, such as reading the Ktor and Kotlin developer guide, and watching videos to get a better understanding, as this is my first project using Kotlin and Ktor.

So What?

This research stage has helped me understand how to start the implementation of the project, and also gave some ideas to improve the requirements.

Now What?

I will now focus on design the UI and database, as well set up the project.

Student Signature	

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: January

What?

- □ Designed the UI/UX and database.
- □ Set up the API that will handle the database and external API requests. I used Ktor as the back-end framework.
- □ Set up the front-end. I created the project in Android Studio and added all dependencies, and also defined the file structure for my project.

So What?

Consider what that meant for your project progress. What were your successes? What challenges still remain?

The design process has allowed me to get a better idea of how the application will work and also define the main features or use cases, that will help with the requirement specification and development. I still need to complete part of the UI design and get a better understanding of the server configuration, as this is my first time using Ktor.

Now What?

I will start the second sprint, which focuses on implementing the login and authentication functionalities. This will also require to implement the API requests that will handle these features.

I will also start the requirement specifications, as this will help with the development process.

Student Signature	Gabriel Salas

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: February

What?

During the last month, I completed the second sprint of the project, which included the following task:

□ I worked on implementing the authentication functionality in the back-end, which included creating the sign-up and sign-in requirements. Initially, I started by defining the requirements and researching the best practices for secure authentication, such as JWT tokens.

□ I worked on designing the user interface and flow for both the sign-up and sign-in screens, and started the implementation in Android Studio. This is still work on progress.

So What?

The completion of these tasks represent an important milestone for the overall progress of the project, as these are two of the critical requirements for the app's success.

Unfortunately, I was not able to complete the front end for these functionalities. I will carry this over the third sprint.

Now What?

I plan to start the third sprint. This sprint will focus on implementing the front-end for the authentication functionalities and researching and implementing the booking requirement, which represents the main app functionality.

Student Signature	Gabriel Salas

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: March

What?

During this month I worked on the third Spring which focused on the implementation of the following requirements:

□ I complemented the front-end implementation for the sign-up and sign-in activities. This was something I was not able to complete the previous month.

□ I started the back-end implementation for the booking requirement. This included choosing the right algorithm to make predictions, creating the tests and implementing the booking search route.

So What?

The booking activity is the main requirement and it is crucial for the success of the project. By implementing the back-end I helped guaranteed the project success, and addressed one the major concerns.

Now What?

The next sprint will focused on the front-end implementation of the booking requirement. I also need to address some of the bugs encountered during this sprint:

□ I need to implement an algorithm that allows me to make predictions when an user has no booking history. This may include asking user for flight and hotel preferences.

□ I might need to implement View Models to handle data between activities.

Student Signature	Gabriel Salas

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: April

What?

This was the fourth Sprint and I worked on the following tasks:

- I completed the back-end implementation for the booking requirement. To be able to predict bookings for new users, I created a new requirements which asks users for their preferences and helps them set up the account. This preferences are used to make simple predictions when there is not enough data.
- I worked on the mid-term presentation.

So What?

Due to time constraints, I was not able to complete the booking implementation, as I used most of my time to work on the mid-term documentation and presentation.

Although I have an idea of what I want to do for the “Set up” requirements, I couldn’t find the time materialise this idea.

Now What?

I will carry the booking back-end and front-end implementation to the next Sprint. I will need no define the Set up requirement to be able to implement as the booking requirements depends on this implementation.

Student Signature	Gabriel Salas

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: May

What?

This month was quite productive, I was able to work on the following tasks:

- I defined and implemented the Set Up activity. This allows news users to update their password and set their preferences. These preferences are then used to make simple predictions when there is not enough data.
- I completed the implementation of the Booking requirement, including both the front and back end.
- I implemented a new Content-Based filtering algorithm to provided recommendation based only in the user preferences. I uses a matrix to calculate the dot product and determine the similarity.
- I implemented a new requirements that allows users to make changes to the predicted booking, such as manually selecting a different flight or hotel.

So What?

These requirements are all crucial for the app success. All tests related to these requirements have passed, which means and I can now focus on other tasks.

Now What?

I will now focus on the other requirements, such as the user and booking management. I will also try to update the documentation to reflect the new requirements.

Student Signature	Gabriel Salas

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: June

What?

This month I focused on implementing the last requirements, such as the user profile, and user and booking management. This included implementing the routes in the API to access the necessary user and booking data from the database, and the front-end implementation.

During this month, I also to change the front-end approach and implement a Navigation component to handle the navigation between different fragment. This allowed to have one main activity in the app to handle the most of the fragments (use cases), in line with modern Android design patterns, which aims to have a single or few activities.

So What?

The progress this work allowed me to complete all the specified requirements. Thanks to this progress, I will now be able to focused on the final documentation and fixing any bugs.

Now What?

There a few bugs I need to address:

- There is an issue with the transition between the set up and main activity. The main activity load without calling the airport list, which throws an exception.
- The refresh token service is not working as expected. The interceptor retries the call with the refresh token but doesn't refresh the tokens at the end.

I also need to focus on the final documentation and presentation.

Student Signature	Gabriel Salas

Supervision & Reflection Template

Student Name	Gabriel Salas Segura
Student Number	19104162
Course	Computing Project (BSHCSDE4)
Supervisor	Vladimir Milosavljevic

Month: July

What?

This month I focused on refactoring the code and fixing any low priority bugs, such as minor issues with interface and implementing the Google Places API to also add addresses to the hotel results.

I also worked on the documentation and the final presentation. This include completing the technical report, creating the poster and the final presentation.

So What?

All the work done during this month was fundamental for the success of the project, as I was able to improve the quality of the code and complete the documentation, which is one of the main areas of the project.

Now What?

This month concludes the development process for my project. However, I do plan to implement some of the features mentioned in the further development section of the technical report, as this project could be used as part of my portfolio and help when I am looking for new job opportunities.

Student Signature	Gabriel Salas