POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design Document Green Journey

Design and Implementation of Mobile Applications

Authors: **Kevin Ziroldi - 10764177**

Matteo Volpari - 10773593

Professors: Luciano Baresi
Academic Year: 2024-2025
Version: 1.0
Release date: 7-7-2025

Contents

1 | Introduction

1.1. Purpose

This Design Document outlines the functional blueprint of the **GreenJourney** mobile application. Its primary objective is to present a comprehensive description of the key components, detailing their roles, interactions, and the interfaces through which they communicate. By offering a clear view of the system's structure and functionality, this document serves as a foundational reference for developers, designers, and stakeholders involved in the application's development.

1.2. Scope

GreenJourney is a mobile application designed to raise awareness about the importance of sustainability in our daily lives. It specifically focuses on promoting eco-friendly choices in travel and commuting, encouraging users to consider more sustainable alternatives when planning their journeys.
Users can **search for a journey** by specifying a departure location, destination, and travel dates. GreenJourney will then provide a range of travel options across various modes of transportation, including bike, car, bus, train, and plane. The user can then choose and save the option that prefer and once the date of the trip has passed, the trip will automatically move from 'scheduled' to 'completed' and the user can confirm or delete it. Once a travel has been confirmed a user can compensate it by planting some trees to offset the CO_2 he has emitted.
In addition to this main feature, the application offers other side functionalities:

- **Machine learning:** if a user does not know where to go, he can get inspired and ask the AI for help. Within the app, there is a proprietary machine learning model that proposes up to five possible destinations that the user might like, based on features of places the user has visited, such as continent, living cost, economy level, outdoors and many others.

- **Reviews:** in order to make it easier to choose an eco-friendly journey even once you have arrived at your destination, GreenJourney is equipped with a city sustainability-oriented review system.
- **Ranking:** to put some competition between users of the app, there is a system of scores and badges based on how sustainable a user has been. Based on these scores, there are rankings to stimulate users to do better than others and earn the top of the leaderboard.
- **Dashboard:** finally, there is this section that allows the user to see several aggregated data on his past trips.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

Definition	**Meaning**
Segment	Basic unit of a journey, containing information such as duration, distance and means of transport to get from point A to point B
Travel	Set of segments saved via GreenJourney
Options	Set of segment sets proposed as a result of a user's search for a travel

Table 1.1: Definitions table

1.3.2. Acronyms

Acronym	Meaning
DD	Design Document
iOS	iPhone Operating System
iPadOS	iPad Operating System
MVVM	Model View ViewModel
MVC	Model View Controller
ORM	Object Relational Mapping
IATA	International Air Transport Association
ML	Machine Learning
GLM	Generalized Linear Model
DBMS	Database Management System
API	Application Programming Interface
CRUD	CREATE, READ, UPDATE and DELETE
UML	Unified Modeling Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
REST	Representational state transfer
CSV	Comma-Separated Values
UI	User Interface
UX	User Experience

Table 1.2: Acronyms table

1.3.3. Abbreviations

Abbreviation	Meaning
etc.	etcetera
app	application

Table 1.3: Abbreviations table

1.4. Reference Documents

- Software Engineering 2 A.Y. 2024/2025 Slides
- Design and Implementation of Mobile Applications A.Y. 2024/2025 Slides
- Bending Spoons lecture for DIMA course A.Y. 2024/2025 Slides
- UML Official specification

1.5. Document Structure

The DD document is divided into 7 main sections:

- **Introduction:** this section deals with the purpose of the document, the project's scope and a set of definitions, acronyms and abbreviations which are essential to understand the terminology used in the document. Moreover, there is a list of reference documents used in the drafting of the DD.
- **Overall description:** this section contains an overview of the system's architecture, starting with a description that includes high-level components and their interfaces.
- **User Interface design:** this section contains several screenshots of the application that are useful for understanding the design of its user interface. The graphical information is also enriched with diagrams that allow to understand the complete flow of use of the application.
- **Requirements:** in this section, the functional requirements of the application are presented. There is also a mapping between components and requirements that highlights the functional role of each component.
- **Implemenation, Integration and Testing Plan:** this section explains how we implemented, integrated and tested all the components of the application.
- **References:** this section contains a list of reference materials used in the preparation of this document.

2 | Architectural Design

2.1. Overview

This section provides an overview of GreenJourney's architecture, highlighting the key elements composing it. However, it doesn't provide details about the internal structure of the application, which will be explained in Section 2.2, nor about the deployment of the system, detailed in Section 2.3.

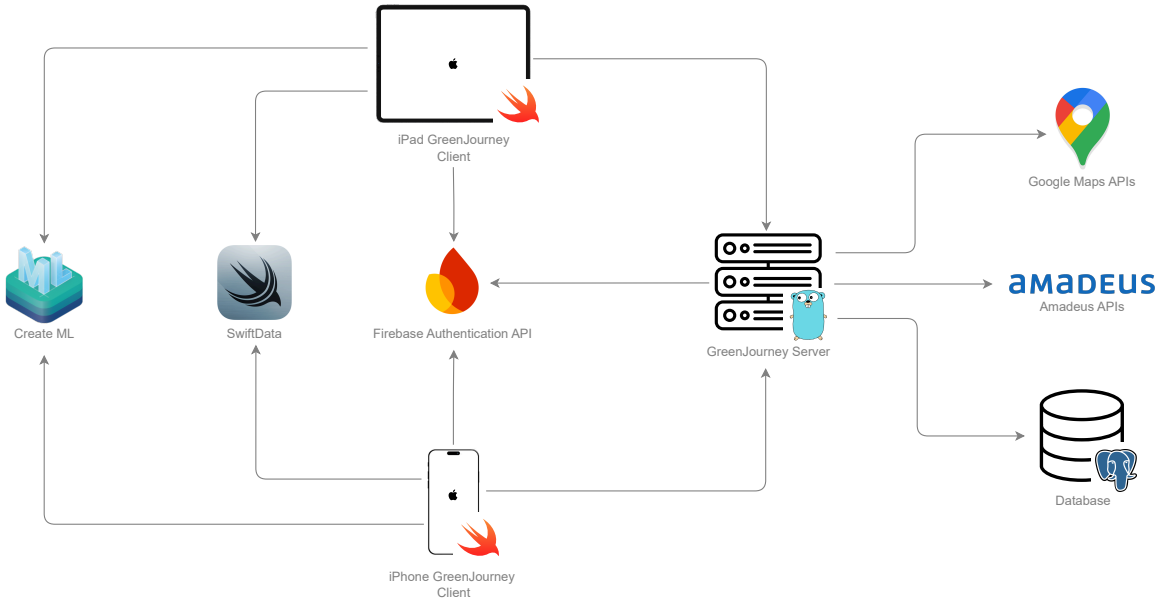

Figure 2.1: GreenJourney Overview diagram

GreenJourney implements a client-server architecture, in which the clients are an iOS app and an iPadOS app, while the application server is a RESTful server containing the business logic and interacting with the Database and with external APIs.
A GreenJourney client is a thick client, since it implements the presentation layer, but also contains part of the application layer and data layer. The iOS app and iPadOS app are developed following the Model-View-ViewModel (MVVM) architectural pattern, they

share the same data layer and application layer (Model and ViewModel), but they implement a different presentation layer (View) in order to best suite different screen sizes and form factors.

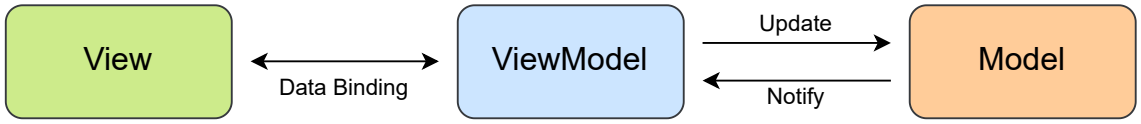

Figure 2.2: MVVM Design Pattern

The Mobile App uses two Apple frameworks:

- SwiftData: used to implement client-side permanent storage, which is useful to guarantee a base level of functionality in the absence of connection, to reduce the number of interactions with the Server and to have static data permanently loaded.
- Create ML: used to build a Machine Learning model. GreenJourney uses a GLM Classifier model to predict cities the user may want to visit based on the cities he has already visited.

GreenJourney server exposes RESTful APIs and implements the following functionalities:

- Authentication: the server authenticates clients using JSON Web Tokens (Firebase ID Tokens in particular).
- Business Logic: the server implements the logic of the application needed to process client requests, to handle user data and incoming requests.
- Data Management: the server communicates with the Relational Database through an ORM, performing CRUD operations needed to process client requests.
- External services: the server interacts with external services, performing requests to the RESTful APIs they expose.

The overview diagram also shows the external APIs used by GreenJourney server and clients.
Firebase Authentication API is used to authenticate users based on a Firebase ID Token. This API is used by the client during the login or signup to the Mobile Application. It is also used by the server to authenticate client requests based on the token they include in the header of requests requiring authentication.

All other APIs are used by GreenJourney server to retrieve travel options with different transport means and to standardize data from different APIs, as further specified in Section 5.1:

- Google Maps Distance Matrix API
- Google Maps Directions API
- Google Geocoding API
- Amadeus Flight Offers Search API
- Amadeus Airport City Search API

2.2. Component view

In this section we describe the internal structure of GreenJourney Mobile App through a Component diagram, which exposes the main components of the application and how they interact among them. Every component can expose an interface ("lollipop" symbol) or use an interface exposed by other components ("socket" symbol).

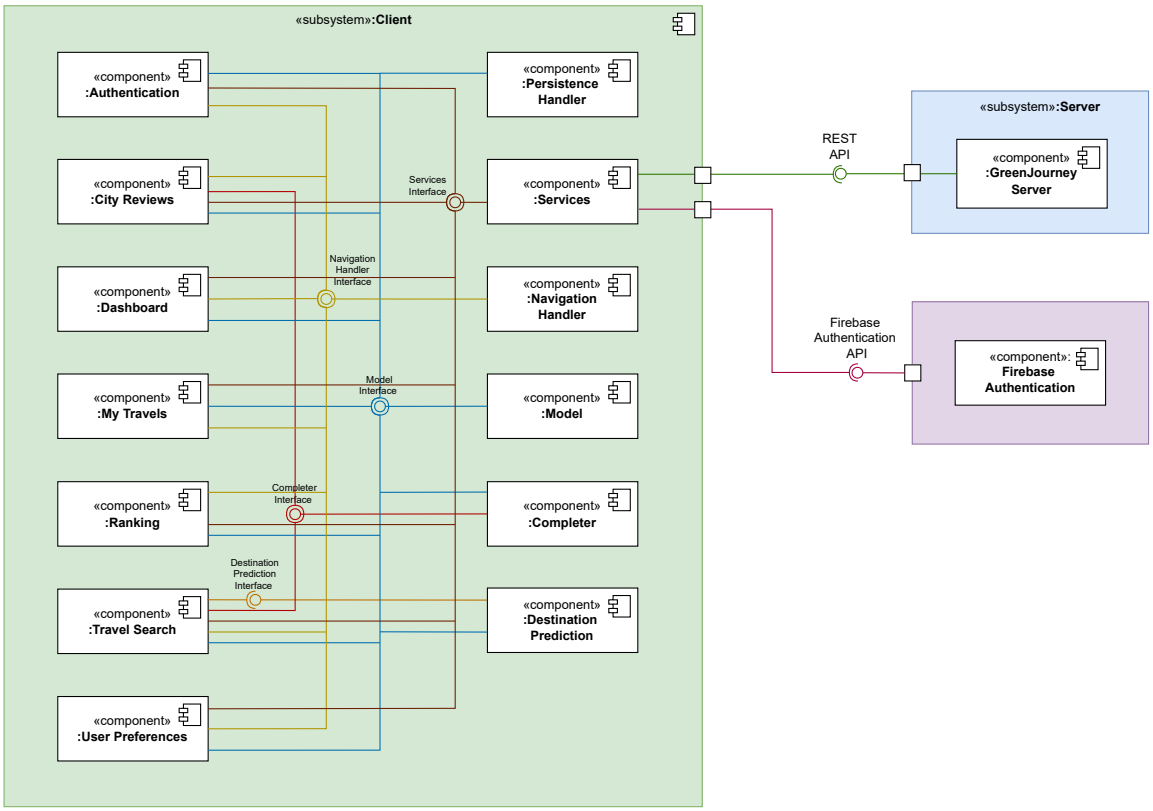

Figure 2.3: Component View diagram

GreenJourney App adopts a modular architecture, in which components represent the features of the application, which we will refer to as Feature components, and some handlers or utilities, i.e. Backend components. The Component diagram also shows two external components: GreenJourney Server and Firebase Authentication.

2.2.1. Features components

Features components are all the components in the leftmost part of the diagram, they represent the main features of the App and are the components that user can directly access by interacting with the View. Each of these components refers to a ViewModel, a main View and a number of smaller Views that can be accessed by the main one. We provide a description of what each Feature component does.

Authentication

Authentication component allows a user to Login to the application both through email and password and with Google Sign in. Moreover it allows the user to Signup and verify

his email. Finally, once the user is logged, it allows him to modify his password.

City Reviews

GreenJourney implements in-app reviews to evaluate a city's environmental friendliness: users who visited a city can evaluate local public transport, presence of waste bins and waste management and presence and quality of green spaces in the city. City Reviews component allows the user to search the reviews for a certain city, leave a review for a city he has visited or find the best rated cities.

Dashboard

Dashboard component shows the user aggregate data about his travels thanks to graphs and other visual representations.

My Travels

My Travels component shows the user the list of his planned travels and completed travels. For each travel, it allows the user to:

- check the CO_2 emissions of the travel, check the number of trees to be planted to compensate emissions and plant trees
- see general data about the travel, such as distance, duration and price
- confirm the travel or delete it

Ranking

Ranking component handles the gamification in GreenJourney app. It shows the user his score in the app and his badges, which he can achieve by choosing ecologically friendly travel options and compensating his carbon emissions. Moreover, it shows two rankings containing GreenJourney's users who achieved the highest scores on long distance and short distance travels respectively.

Travel Search

Travel Search component allows the user to search the available travel options for a certain destination. For each means of transportation, it shows the user how to reach the search destination, including:

- general data such as distance, duration and price

- detailed data about the carbon footprint of the travel, such as emitted CO_2, number of trees to be planted to compensate emissions and equivalent price to travel with neutral carbon footprint
- details about all travel segments forming the travel option

User Preferences

User Preferences component allows the user to manage his personal information, add, edit or remove information and logout from the account.

2.2.2. Backend components

Backend components are all the other components in the Client subsystem. We call them Backend components since they are used from the Feature components; some of them have a View and ViewModel, while others just contain logic. We provide a detailed description below.

Completer

Completer component provides a text field with auto-completion used to search cities in the app. It also provides predictions on most likely departure cities and possible destination cities that the user may want to visit.

Destination Prediction

Destination Prediction component leverages the Machine Learning model trained using Create ML to predict cities that the user may want to visit in the future. It contains mainly logic, but also a simple View representing a button that the user can use to generate predictions.

Navigation Handler

Navigation Handler component is responsible for the navigation among Views in the application. It decides which View should be shown at startup, i.e. checks the login status, and implements a TabView for iPhones and a NavigationSplitView for iPads, which contain the main Views of the app.

Persistence Handler

Persistence Handler component is responsible for setting up client-side persistence: it loads the SQLite Database that SwiftData uses to persist data.

Model

Model component contains all classes representing the Model of the MVVM Architecture.

Services

Services component represents the Network Layer of the Mobile App, it contains the logic to perform API calls and to decode the responses both to GreenJourney Server and Firebase Authentication external service.

2.3. Class Diagram

In this section we report a high level class diagram of the application, which has been divided into two parts to ensure a better readability. The first diagram contains all the features of the application and the networking part, while the second diagram contains the Model and some other classes.

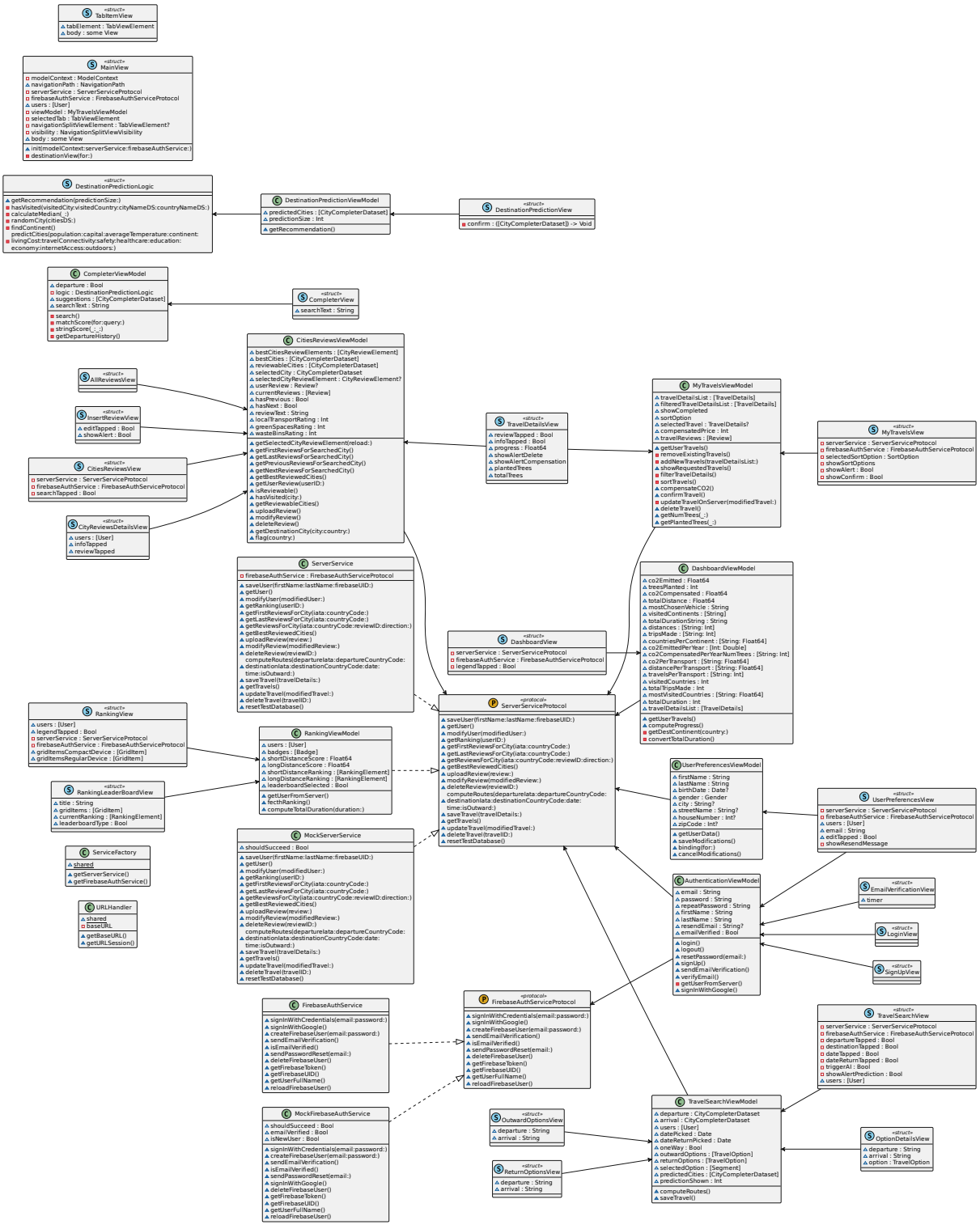

Figure 2.4: UML Class Diagram Features

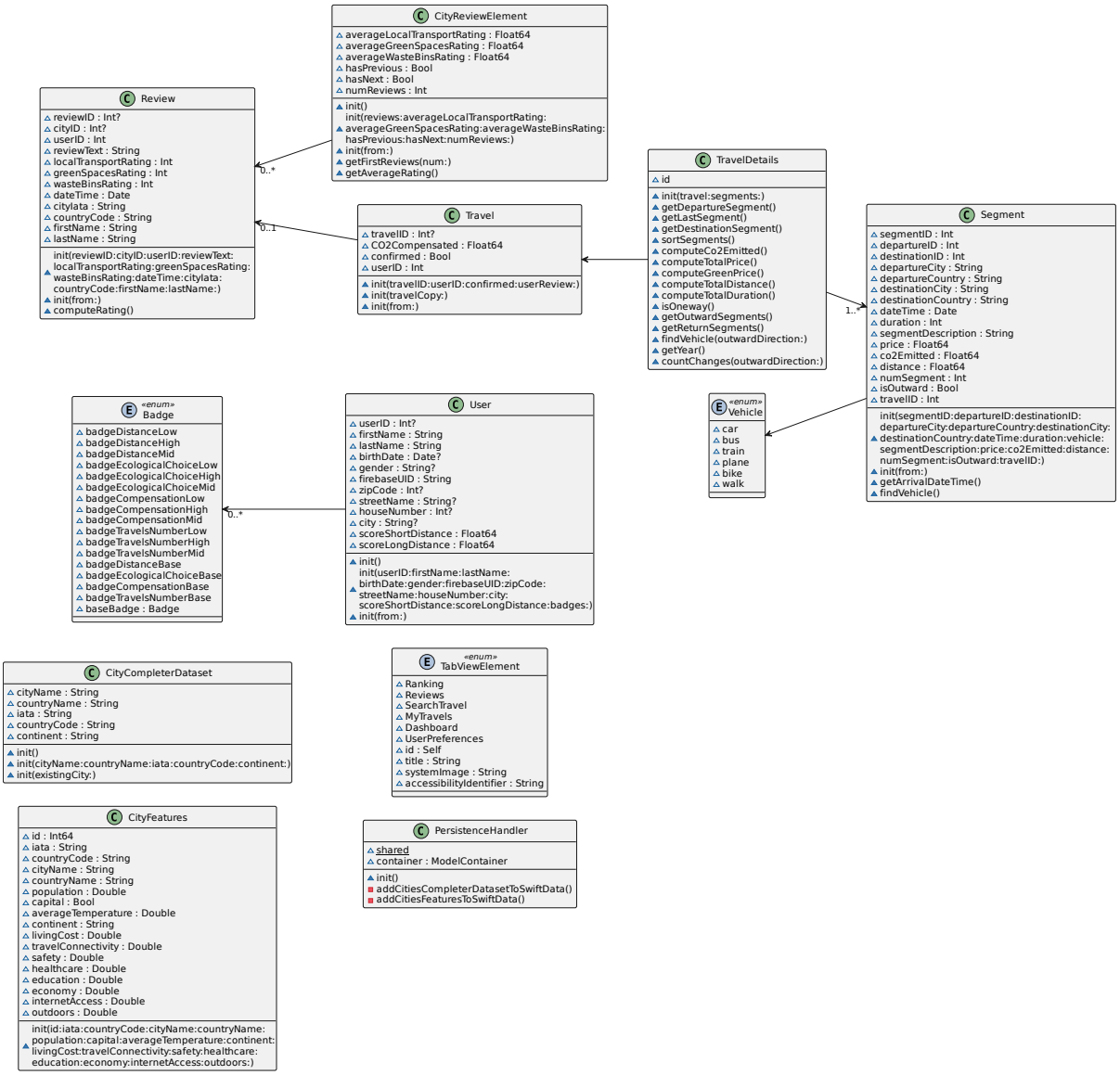

Figure 2.5: UML Class Diagram Model

2.4. Deployment view

GreenJourney uses a 3-tier architecture, the Deployment Diagram in figure 2.6 shows the tiers composing the system, their cardinality and the communication protocols they use to exchange information. The color of the devices in the Deployment Diagram refers to the layers in the three layers architecture and is common to the color of subsystems in the Component Diagram.

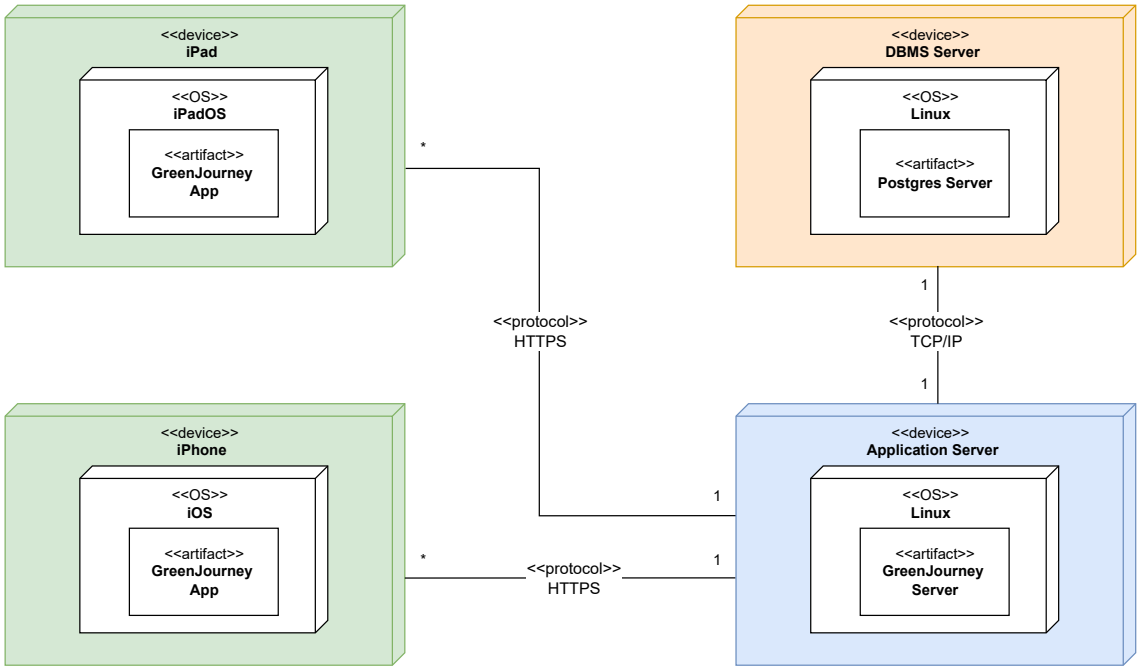

Figure 2.6: Deployment View diagram

The tiers composing the architecture are:

1. Client Tier: the client tier can be either an iPhone running GreenJourney iOS application or an iPad running GreenJourney iPadOS application.
2. Application Server Tier: this tier includes GreenJourney Server that implements business logic, handles the interaction with the Database and with external APIs.
3. DBMS Server Tier: the last tier contains the DBMS Server, which stores persistent data for GreenJourney.

2.5. Runtime view

In this section we present some Sequence diagrams showcasing the behaviour of the Mobile App at runtime. We report here some examples that allow to better understand the role of each component in the application and how components interact among them.

2.5.1. Login

The first Sequence diagram shows how the Login process is handled. It is important because it shows how the application interacts with both Firebase Authentication API

and GreenJourney server.

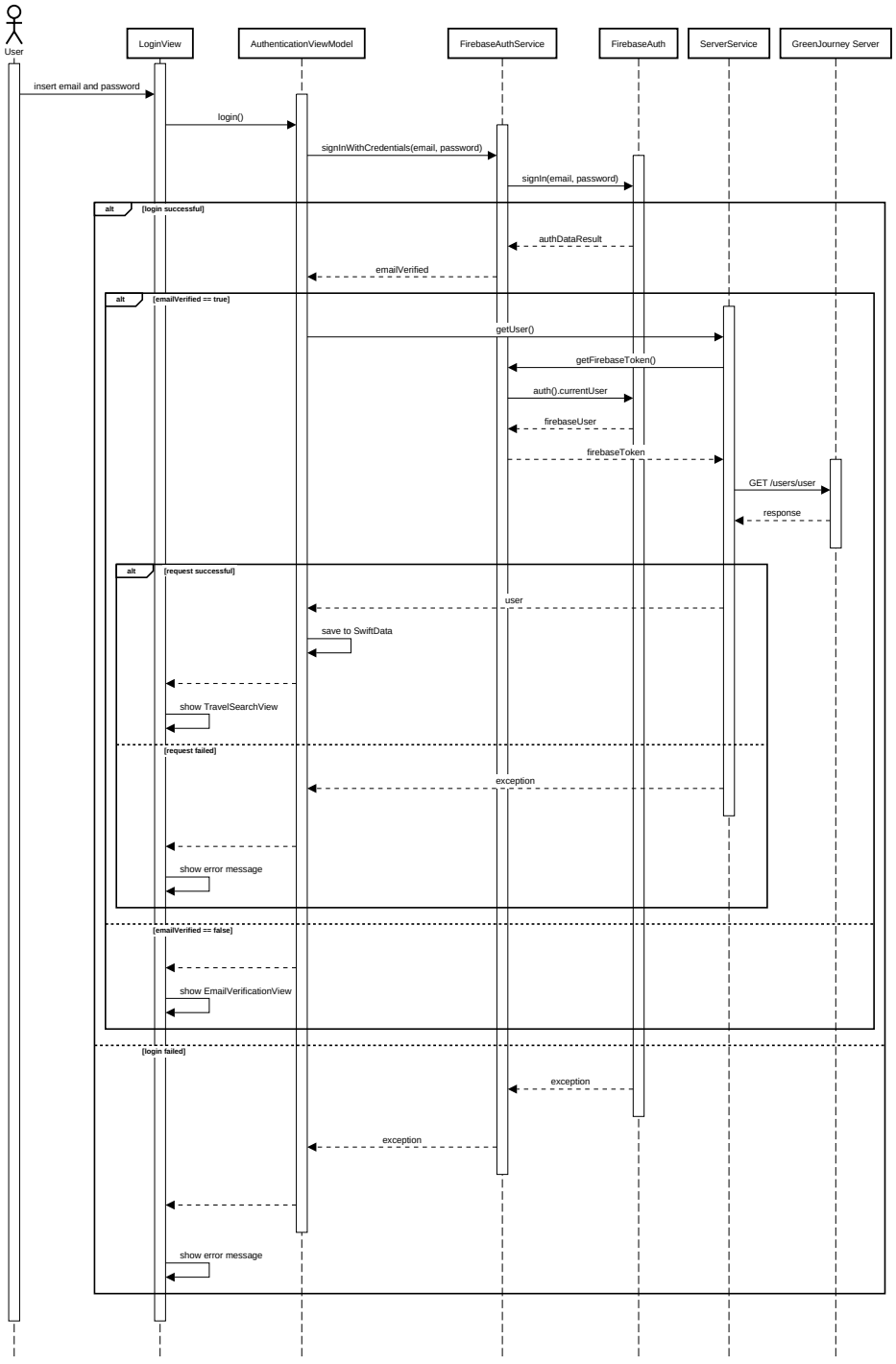

Figure 2.7: Sequence diagram Login

2.5.2. Travel search

The second Sequence diagram, shows how a travel search works, a central feature of GreenJourney.

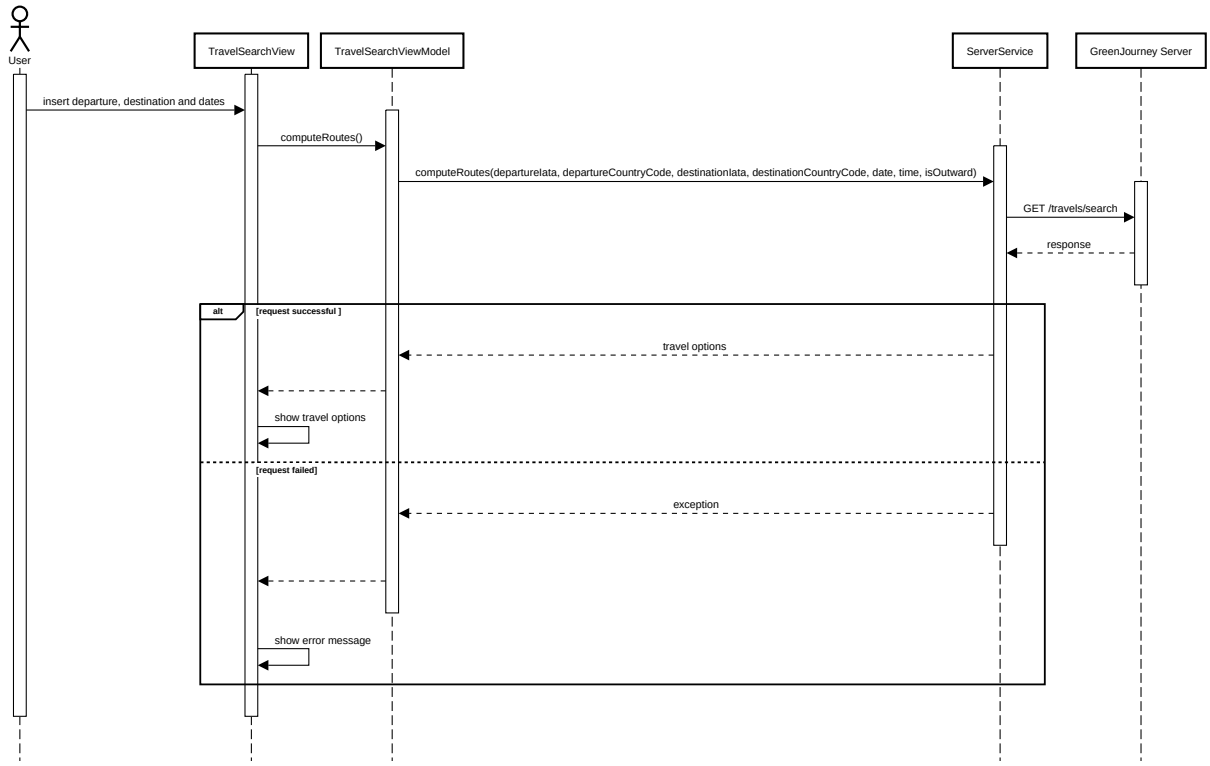

Figure 2.8: Sequence diagram Travel search

2.5.3. CO_2 Compensation

Finally, we present a last fundamental feature of the application, which is the compensation of carbon emissions. This runtime view also shows an HTTP PUT request and how the applications deals with the synchronization of the client-side storage in SwiftData.

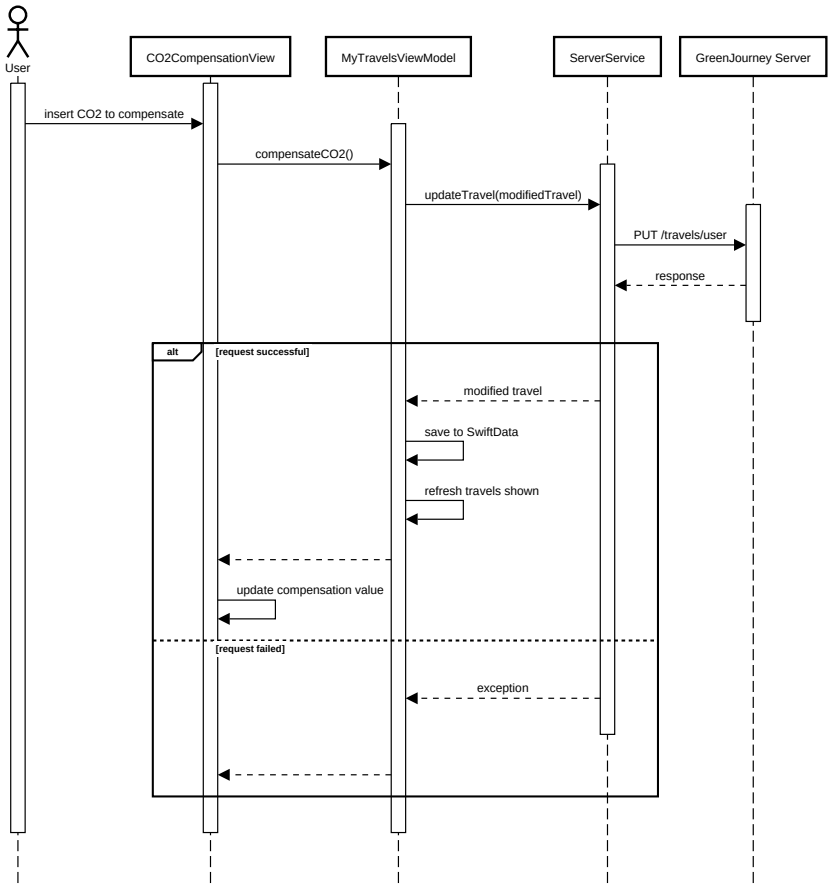

Figure 2.9: Sequence diagram CO_2 Compensation

2.6. Architectural Styles and Patterns

2.6.1. Client-Server Architecture

The Client-Server Architecture splits the system into two main parts: the Client and the Server. In GreenJourney, the Client is a Swift mobile application, while the Server is a Golang RESTful server. A GreenJourney Client is a "thick" Client, since it contains the presentation, but also a small part of logic and of data, managed through SwiftData.

2.6.2. RESTful Architecture

GreenJourney Server is a RESTful server, a server exposing a RESTful API. REST is a software architectural style used in networked applications, which provides uniform interface semantics, rather than application-specific implementations and syntax. The Server is stateless and exposes a RESTful API that the Client can use to communicate with the Server through HTTPS requests (GET, POST, PUT and DELETE). The response

will contain a JSON representation of data, that the Client will use to display new content to the user.

2.6.3. Model-View-ViewModel (MVVM)

The Swift mobile application uses MVVM, a software design pattern that allows to divide the application in interconnected component and achieve separation of concerns. The 3 elements composing MVVM are: Model, View and ViewModel. The Model handles data and business logic. The View contains UI elements and is responsible for displaying data to the user. Finally, the ViewModel acts as intermediary between Model and View, providing data binding.

2.6.4. Model-View-Controller (MVC)

The Golang Server uses MVC, another software design pattern used to break the complexity of the application and identify precise roles for each component. Inside MVC, components are divided into: Model, View and Controller. As for MVVM, the View is responsible for the visual representation of data. The Controller act as intermediary, processing input, converting it to commands for the Model. Finally, the Model handles data and business logic, but also updates the View.

2.6.5. Relational Database

The system uses a relational Database to efficiently manage structured data. This approach ensures data integrity and supports complex queries, making it ideal for handling relationships between the various entities present in GreenJourney system.

2.6.6. Token-based authentication

For secure user authentication, the system employs Token-Based Authentication, based on JSON Web Tokens. This method ensures stateless and scalable authentication, allowing users to access resources securely without requiring the storage of session data on the server. Tokens are generated by Firebase and are used to validate subsequent requests.

2.6.7. Object-Relational Mapping (ORM)

GreenJourney Server uses an ORM system called GORM to interact with the PostgreSQL Relational Database. ORM is a programming technique that allows data to be seamlessly mapped between Relational DBMS and an object-oriented programming language.

2.6.8. Dependency Injection

Dependency injection is a Design Pattern in which an object or function receives other objects or functions that it requires, as opposed to creating them internally. It allows to decouple components and increase reusability and testability. We used this Design Pattern extensively for ViewModel and it simplified a lot the testing of the Swift application.

3 User Interfaces Design

3.1. Introduction

The purpose of this section is to illustrate the design of the user interfaces and the general flow of use of the application.
The flow is presented only for the iPhone, after which some iPad and iPhone screens are shown in both light and dark mode.

3.2. Login and navigation bar

The flow of access to the application is shown here: starting from the **Login** screen, one can directly access the navigation bar that presents the five main screens. If, on the other hand, a user has not yet registered, he can do so by going to the **Signup** page, where, after entering his data, he can register and confirm his email address. At this point, he is able to access the navigation bar.
On all five screens of the navigation bar, there is a button that links to the **Profile** page where the user can enter some personal data or log out.

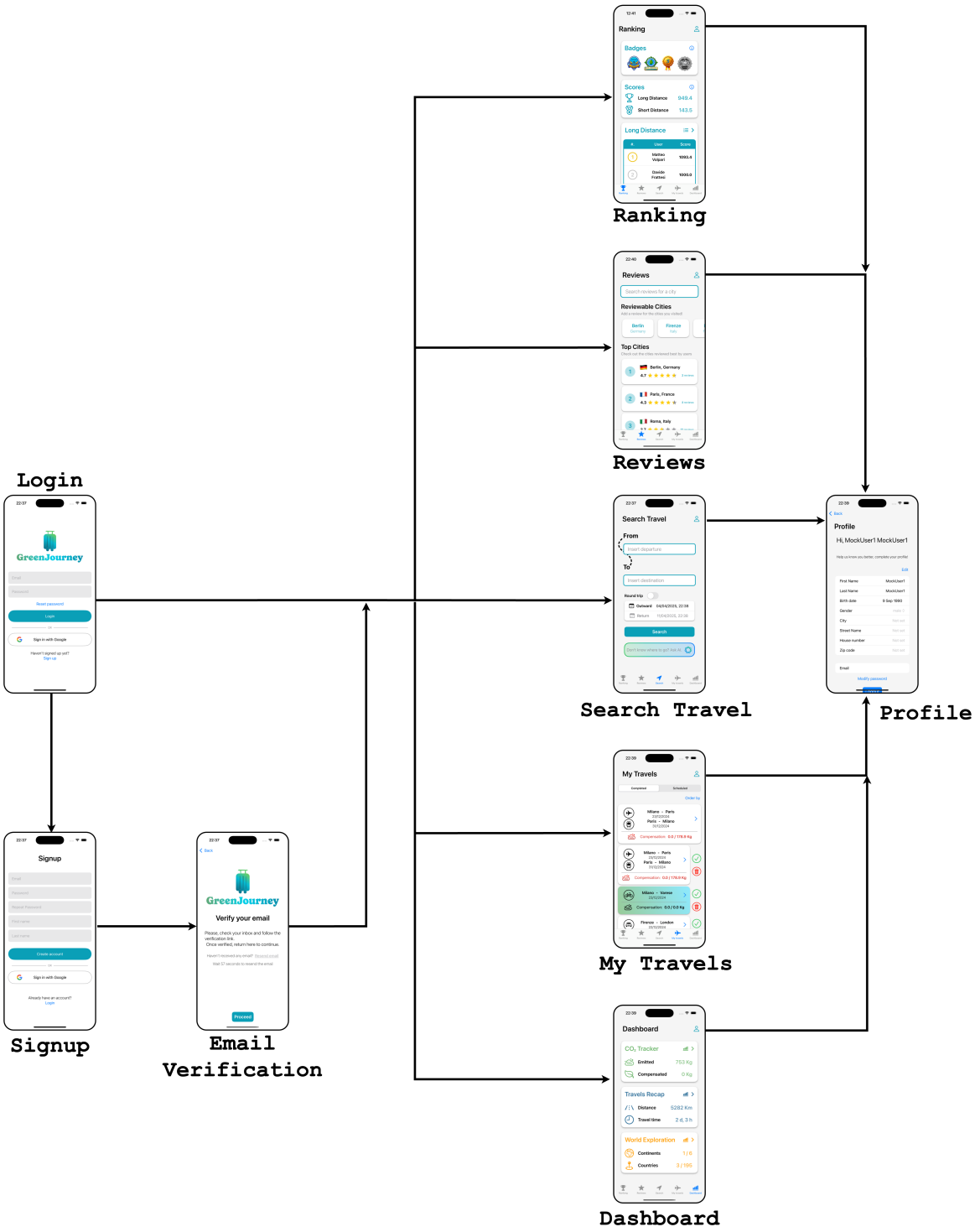

Figure 3.1: Login and navigation bar

3.3. Travel Search

This screen is the beating heart of the application, here users can search for all the travel options that interest them most or be inspired by the generative power of GreenJourney's AI predictive model.
For each travel option that is proposed, the user can see the details and save the trip.

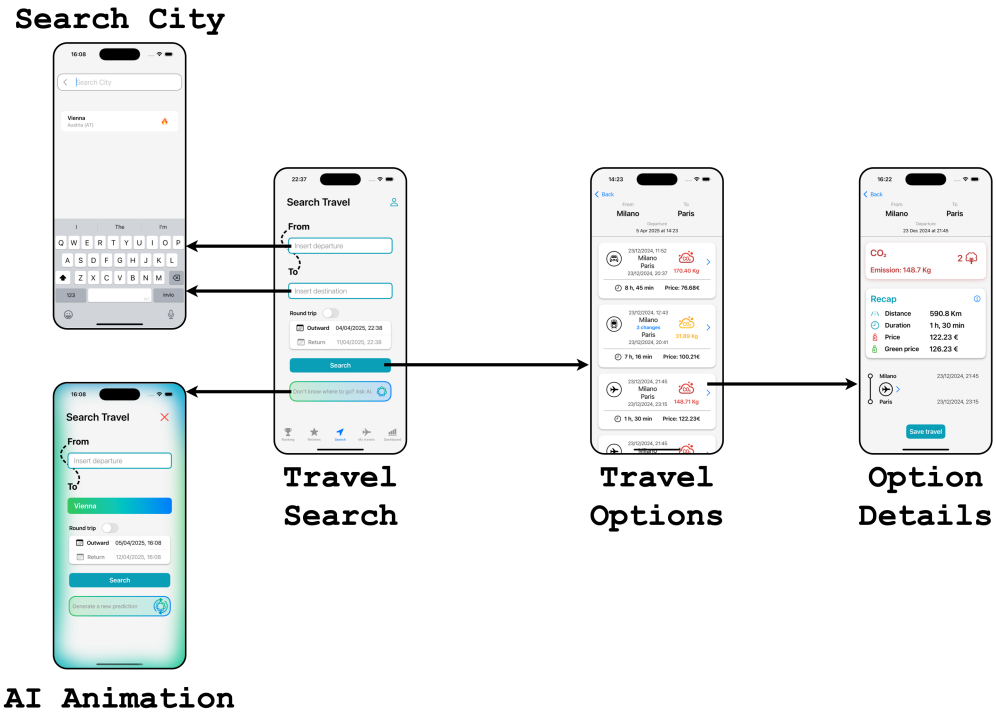

Figure 3.2: Travel search

3.4. My Travels

On the **My Travels** screen, the user can see all the trips he has completed and those he has saved. Each trip contains all the details of its segments.
A trip that is completed and has been confirmed by the user contains a 'Compensation' section where the user can compensate for the CO_2 emitted for that trip.
Also on confirmed trips there is a 'Leave a review' button that allows the user to leave a review on the destination visited via a modal window.

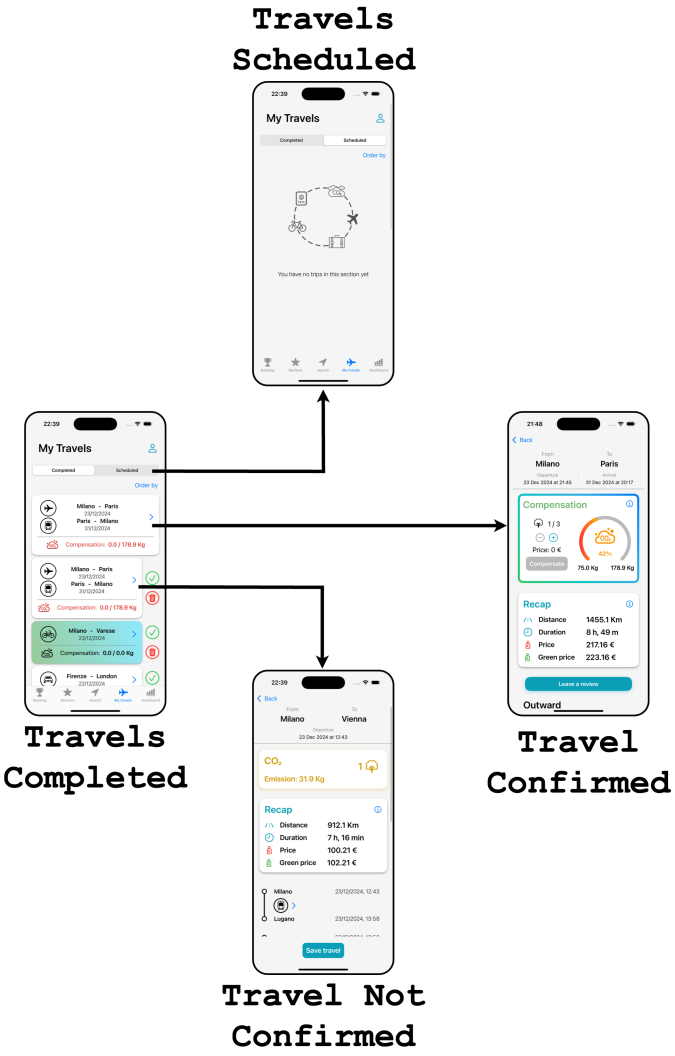

Figure 3.3: My traveks

3.5. Reviews

On the **Reviews** page there are 3 main elements:

- A search field that allows the user to search for reviews for any city.
- A list of cities he has already visited and for which he has not yet left a review.
- A list list of the best 5 cities by average number of reviews left by users.

For each city there is a detail page where there are the average scores, the last five reviews that have been left by users and if the logged in user has visited this city a button that allows him to leave a review or edit his existing one.
Clicking on the "See all reviews" button will open a page that shows all the reviews that have not been left by users for that city, organized by pages, on each page there are ten reviews.

Figure 3.4: Reviews

3.6. Ranking

On the **Ranking** screen there are two blocks concerning personal user information, which are badges and scores, and below these two other blocks relating to the two rankings in the app: long distance and short distance.
The two rankings are differentiated by trips longer or shorter than 800 Km and can be expanded to see the top 10 positions. For users in the rankings, the logged-in user can view some aggregated data and their badges.

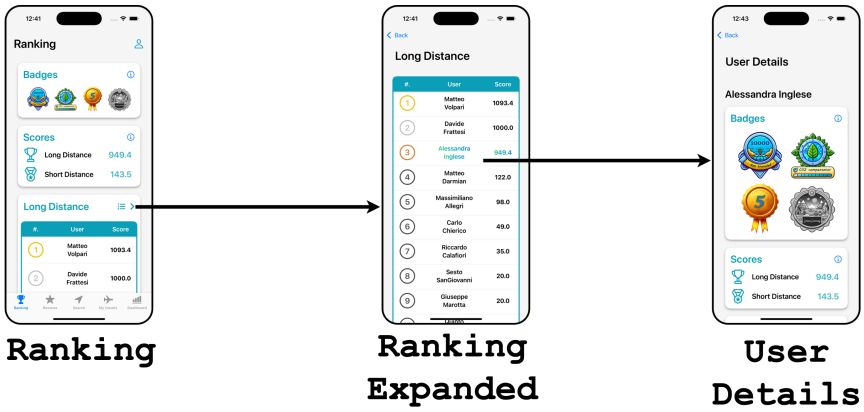

Figure 3.5: Ranking

3.7. Dashboard

On the last main screen, the **Dashboard**, there are 3 expandable blocks containing 3 categories of aggregated data:

- CO_2 Tracker
- Travel Recap
- World Exploration

Each of these screens contains numerous graphs and data on the user's past trips.

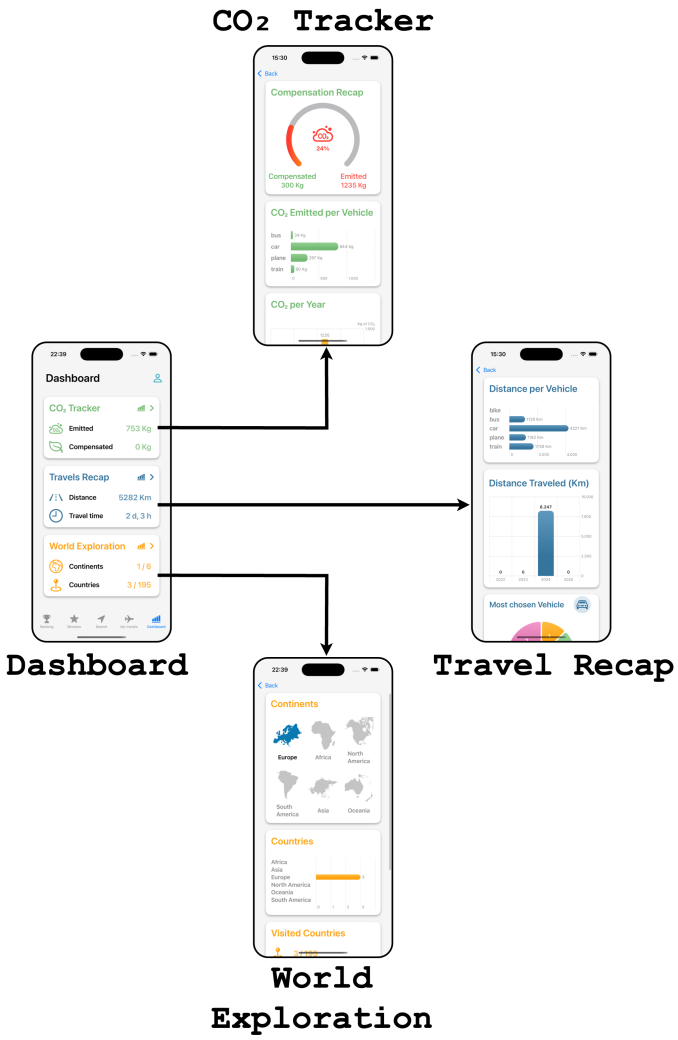

Figure 3.6: Dashboard

3.8. UI

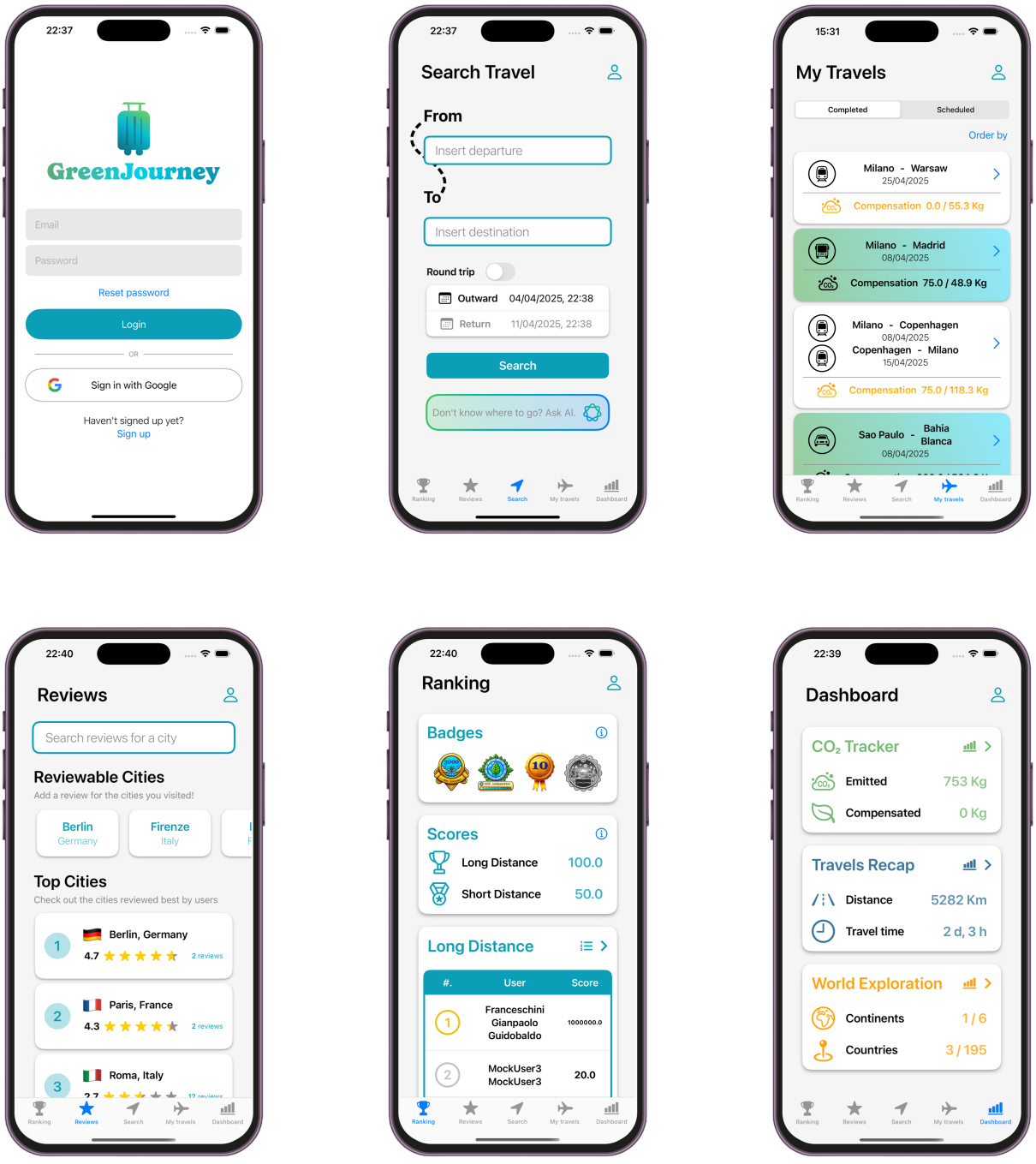

Figure 3.7: Main iPhone UI mockups.

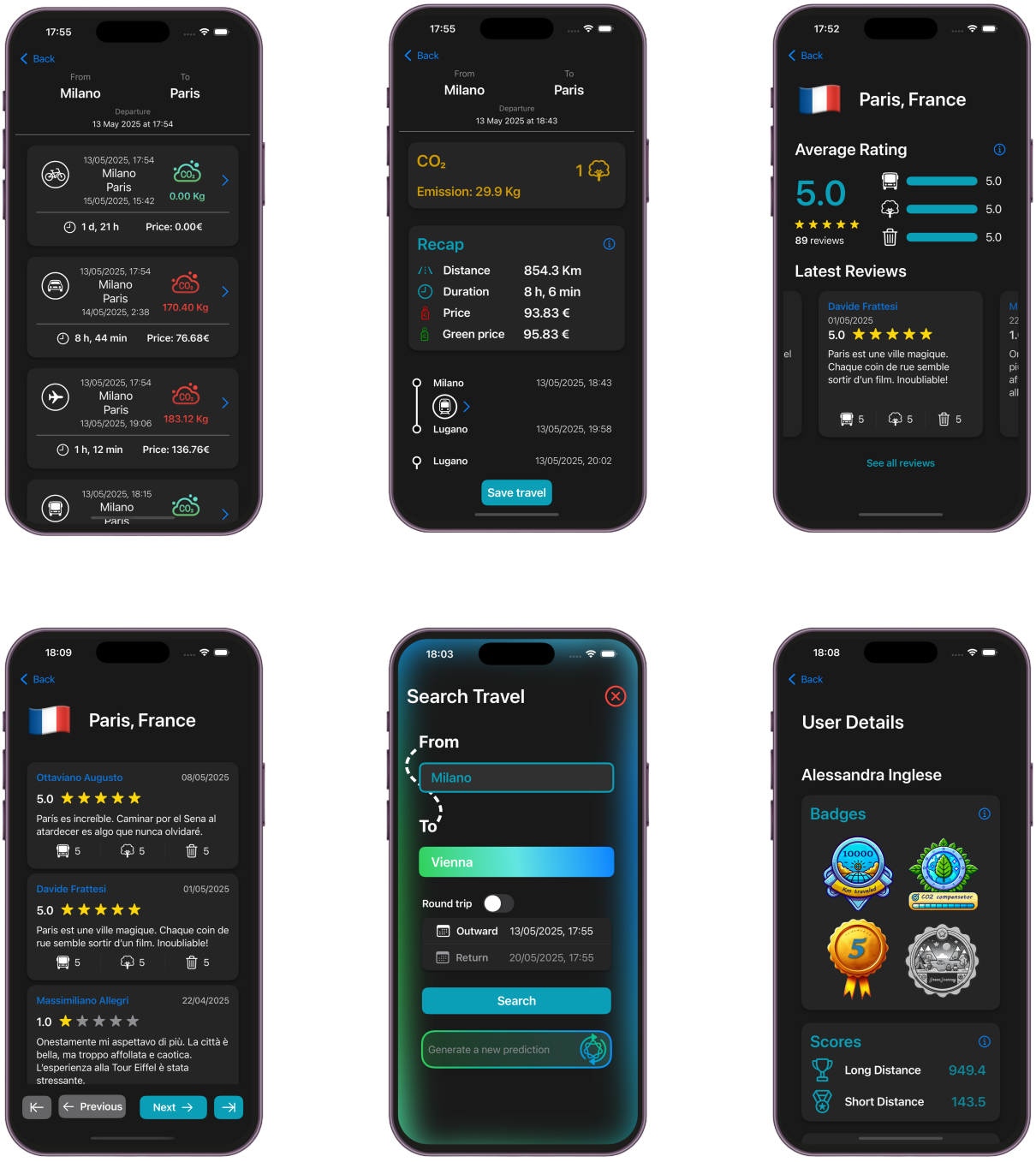

Figure 3.8: iPhone UI mockups in dark mode.

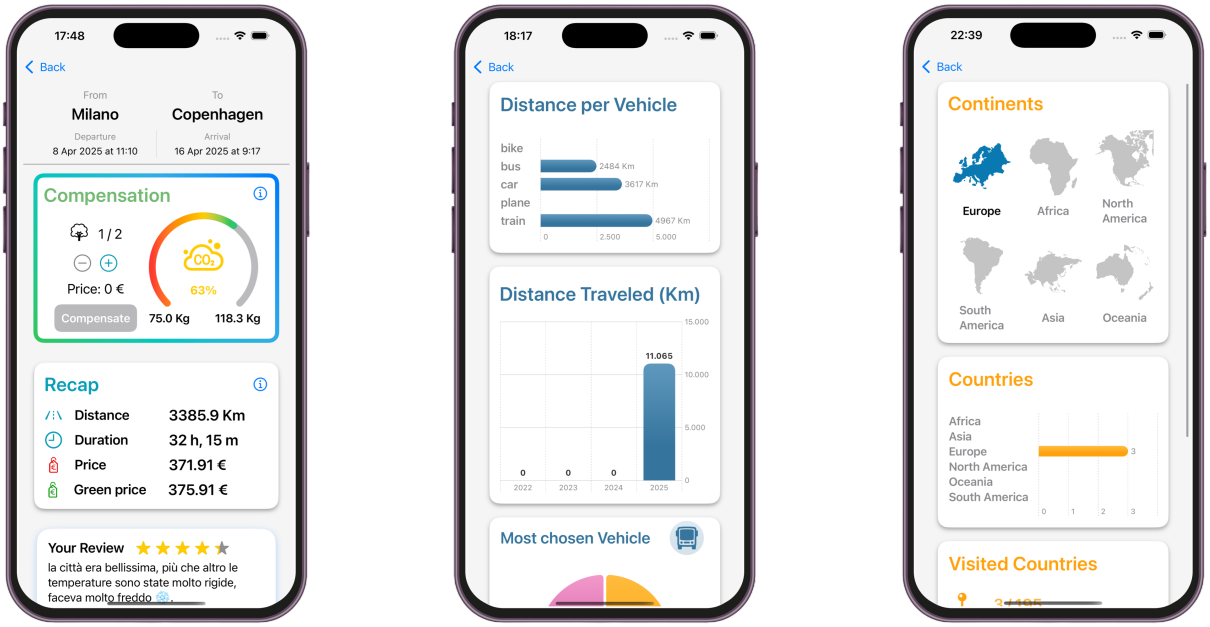

Figure 3.9: Other iPhone UI mockups.

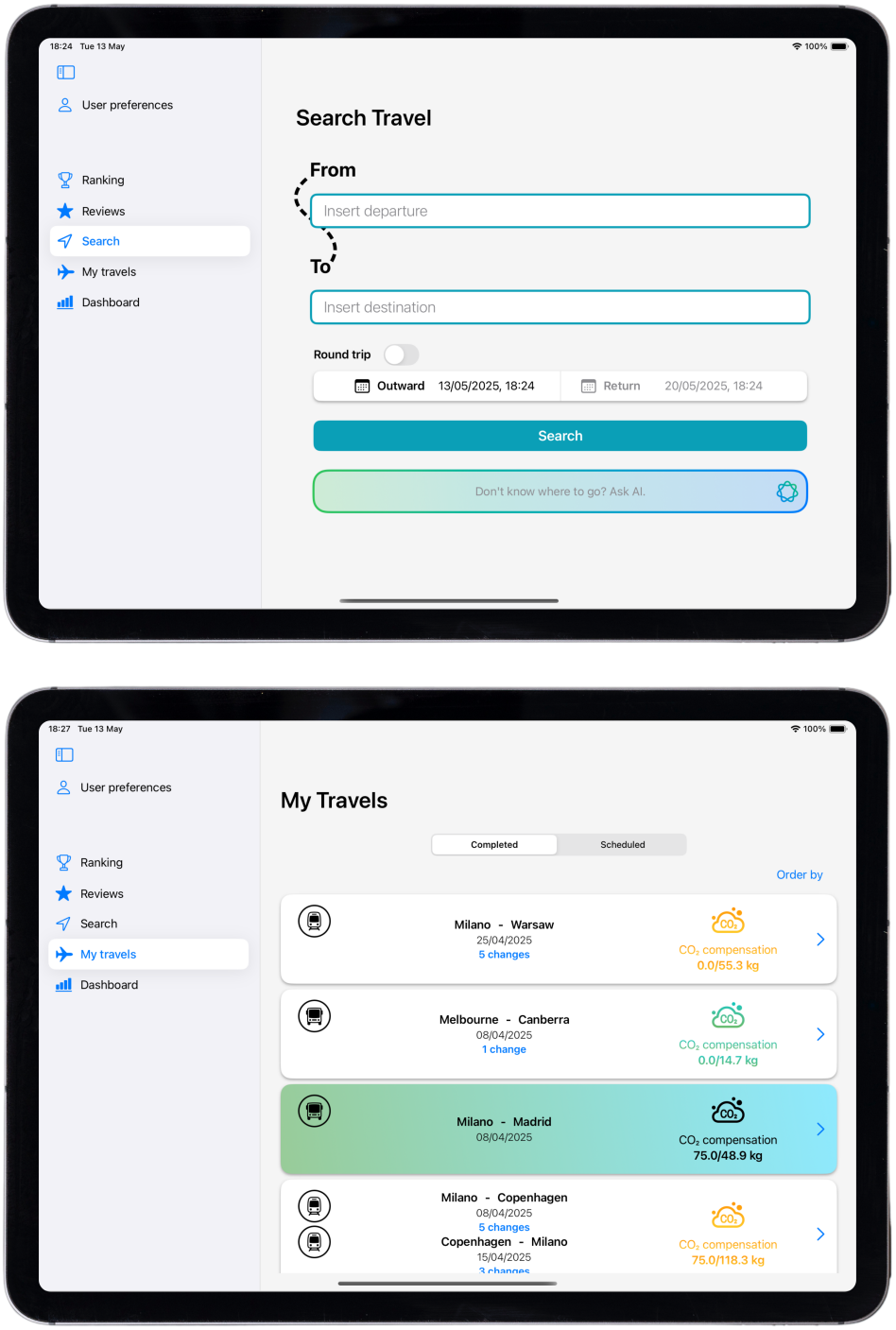

Figure 3.10: Horizontal iPad Ui mockups

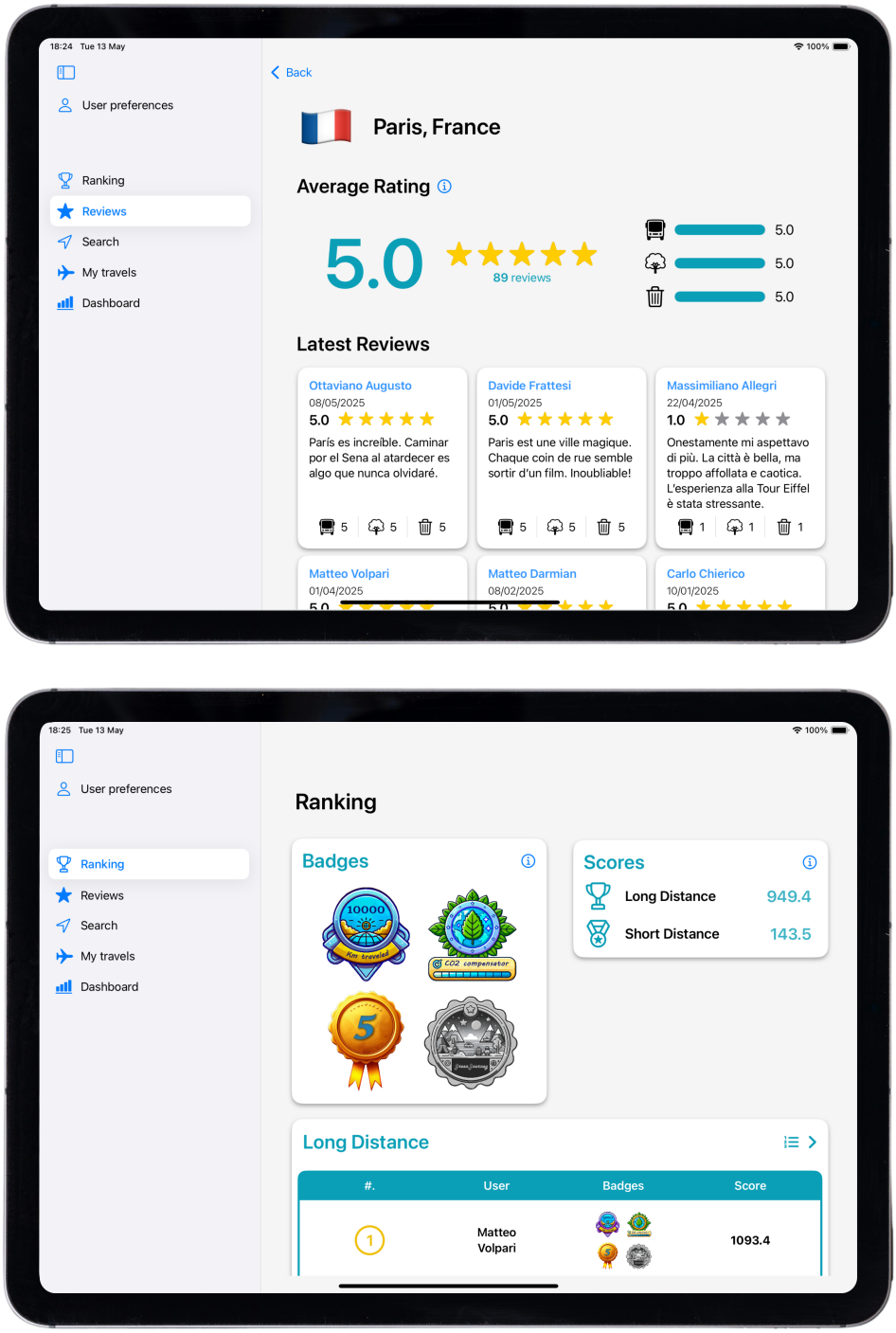

Figure 3.11: Other horizontal iPad UI mockups

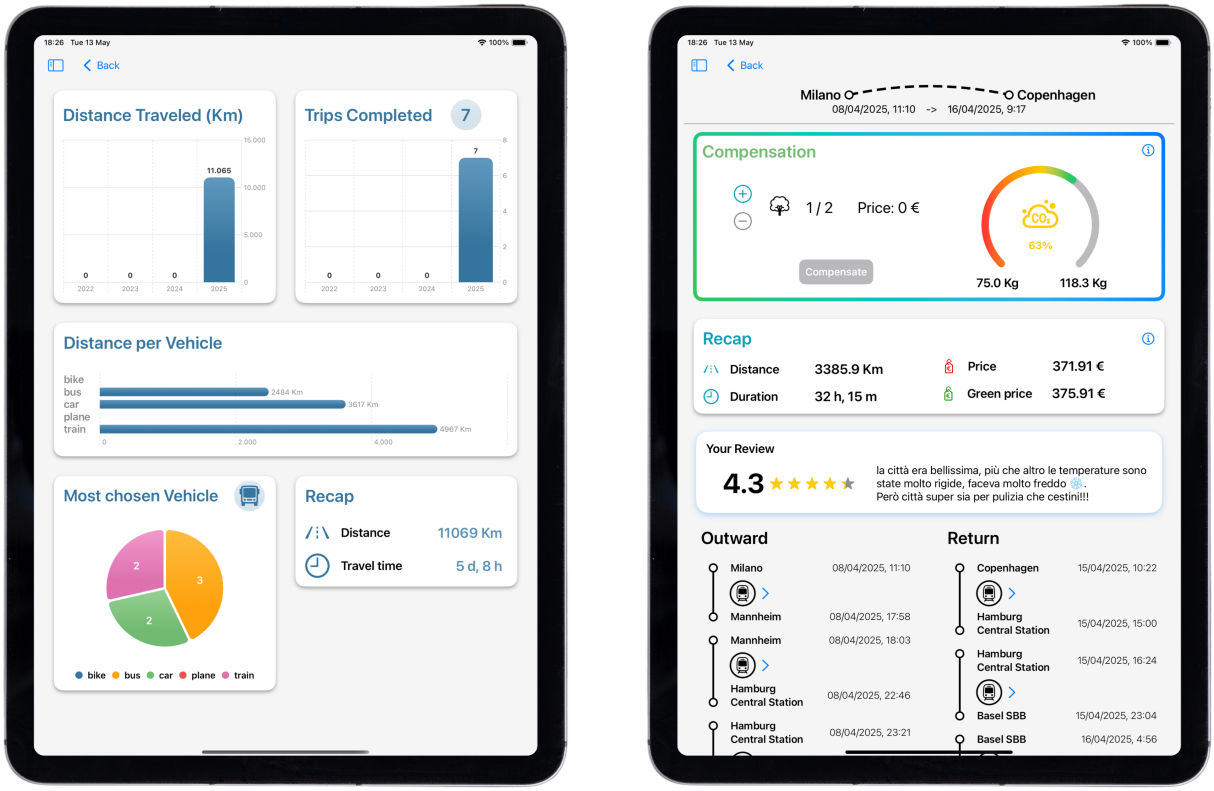

Figure 3.12: Some vertical iPad UI mockups

4 | Requirements

4.1. Functional Requirements

All functional requirements of the system are listed here:

- R1 The system shall allow unregistered users to sign up with email and password or with Google.
- R2 The system shall allow registered users to log in with email and password or with Google.
- R3 The system shall allow registered users to log out.
- R4 The system shall allow users to search for travel options for a certain destination.
- R5 The system shall allow the user to save the chosen travel option from those suggested by the system.
- R6 The system shall allow users to use a predictive AI model to choose the destination for their trips.
- R7 The system shall allow users to see their personal data and modify them if necessary.
- R8 The system shall allow users to see the scheduled and completed trips it has saved in the past.
- R9 The system shall allow users to confirm or delete a completed travel.
- R10 The system shall allow users to sort trips according to their chosen sorting criteria.
- R11 The system shall allow users to see all the details of each saved travel.
- R12 The system shall allow users to compensate the CO_2 emitted for a confirmed travel by planting trees.
- R13 The system shall allow users to leave a review for a city they have visited.
- R14 The system shall allow users to search and see reviews for a city.

R15 The system shall allow users to see their personal badges.

R16 The system shall allow users to see their personal scores.

R17 The system shall allow users to see the rankings.

R18 The system shall allow users to view scores, badges and some aggregated data for users ranked on the leaderboard.

R19 When a user confirm or compensate a travel, the system shall update the user's scores and badges.

R20 The system shall allow users to view all aggregated data on past trips in the dashboard section.

4.2. Requirements Traceability

This section presents a mapping matrix between requirements and application components, which is useful for understanding the roles of components within the system.

Components	Requirements
Authentication	R1 R2 R3
City Reviews	R13 R14
Dashboard	R20
My Travels	R8 R9 R10 R11 R12 R19
Ranking	R15 R16 R17 R18
Travel Search	R4 R5
User Preferences	R7
Destination Prediction	R6

Table 4.1: Requirements traceability table

5 | Implementation, Integration and Test Plan

5.1. Backend

The backend of our system consists of GreenJourney Server and the Relational Database it uses to persist data. GreenJourney Server is a RESTful server exposing RESTful APIs through HTTPS protocol. It is developed in Golang, a language providing high-performance networking and multiprocessing, following MVC design pattern to ensure separation of concerns. The application server relies on a PostgreSQL Relational Database, with which it interacts through GORM, an ORM library, that maps Database entities to the server's data structures composing the Model. GORM also manages a pool of connections to the Database and the use of prepared statements to increase performance and avoid SQL injection attacks. GreenJourney Server accepts incoming HTTPS requests from the exposed API endpoints and processes them using Goroutines, lightweight threads managed by the Go runtime. In order to serve Clients, the Green Journey Server interacts with several external services offered by Google and Amadeus, allowing to the compute routes and travel costs for different means of travel.

Firebase Authentication API

API used to authenticate users based on a Firebase ID Token. Some of the APIs exposed by GreenJourney Server require authentication: the Client inserts the user's Firebase ID Token into the header of the request and the Server uses the API offered by Firebase Authentication to authenticate the Client.

Google Maps Distance Matrix API

API used to retrieve travel options by bike and by car. This API provides aggregate information about the travel, such as total distance and duration, without turn-by-turn details. We use it for bike and car travel means since we don't need detailed data provided

by Google Maps Directions API, including all the roads that the user should travel.

Google Maps Directions API

API used to retrieve travel options by train and by bus. This API differs from Distance Matrix API, since it provides detailed information about all the steps that can compose a travel with a public means of transport. Thanks to this API, we can suggest travel options that involve more changes with trains or buses, including the name or code that identifies them.

Google Geocoding API

This API can be used to retrieve the human-readable address associated to some coordinates. It is used by GreenJourney Server in order to decode information provided by Google Maps Directions API about departure and arrival location of the intermediate steps.

Amadeus Flight Offers Search API

API used to retrieve travel options by airplane, it can retrieve accurate information about the travel, including the stopovers, the price and the flight code.

Amadeus Airport City Search API

This API is used in combination with Amadeus Flight Offers Search API to retrieve the correspondence between the IATA code associated to the airport and the IATA code associated to the city. It is used by GreenJourney Server to find the city associated to possible stopovers.

Mock APIs

Finally, GreenJourney Server interacts with some mock servers, that provide information which we could not find for free from real APIs. Every mock server is a real web server, running on a different port and exposing a RESTful API, it receives HTTP requests and answers with predefined static values. We use the following mock servers:

- FuelCostAPI: returns the fuel cost for travels by car
- TollAPI: returns toll costs for travels by car
- TransitCostAPI: returns costs for travels with public means of transport

5.2. Frontend

The frontend of GreenJourney system is composed of two applications, the iOS app for iPhones and the iPadOS app for iPads. Both applications are developed using Swift and SwiftUI framework, they share the logic but have different UIs to better suite different screen sizes. Both applications communicate with Firebase Authentication API and GreenJourney Server through HTTP requests handled in an asynchronous way, using background threads. Concurrency in the app is handled with the new Swift Concurrency model based on async/await, which provides better readability with respect to closures. As previously stated in Chapter 2, the app is developed following MVVM Design Pattern, which best suites SwiftUI framework.
We partitioned the application into features, representing main portions of the frontend. Features are formed by:

- MVVM elements: a ViewModel, one or more Views, while the Model can be shared among features.
- Client-side application logic.
- Test classes for Unit Tests, IntegrationTests and UI Tests, as will be explained in the next section.

5.2.1. Model

The Model implements the data layer and part of the application layer of GreenJourney Client, but also manages client-side persistency with SwiftData and SQLite. The use of client-side persistent storage allows to reduce the interactions with the Server and provide a base level of functionality in the absence of internet connection.
We also use SwiftData to efficiently access static data about cities, since doing that directly from a bulky CSV file would be very inefficient and would result in a bad UX. In order to provide a smooth experience during the first load of the app, we use a custom configuration that loads an SQLite Database in which static data has been already loaded.

5.2.2. ViewModel

As previously anticipated, every feature of the frontend has a dedicated ViewModel. ViewModels expose methods that can be used by Views after an interaction of the user and prepare the data that the View shows to the user.
In order to communicate with remote services, ViewModels use Dependency Injection,

a design pattern that promotes loose coupling and improves testability. We used this pattern to inject both Server service and Firebase Authentication service into ViewModels: services handle networking in the mobile application and interacts with GreenJourney Server or Firebase Authentication API.

5.2.3. View

Views represent the real difference between the iOS application and the iPadOS one, since only few parts of the View are shared between the two applications.
We used Dependency Injection to inject the ViewModel into Views. Every feature of the application has one main View, which creates the ViewModel and injects it into the children Views; this approach allows to manage dependencies and separate concerns.

5.3. Testing

5.3.1. Testing environment

Extensive testing has been crucial during the development of the application, because it allowed us to catch bugs and issues early and strengthened system maintainability, letting us to safely and efficiently introduce new features or make changes to the existing ones. The test campaign for GreenJourney is performed with 3 types of tests: Unit Tests, Integration Tests and UI Tests.
The application itself has been developed with the objective of easing the testing phase, thanks to Dependency Injection design pattern. But, in order to perform extensive texting, we had to abstract remote services and implement some mock elements both in the frontend and the backend:

- **Testing Database**
  Since some tests use the real GreenJourney Server, we must ensure that tests don't alter data contained in the PostgreSQL Database. We have implemented a second Database, with the same structure as the real one, but used only for tests. Moreover, we introduced a new API endpoint that can be used by the client to reset the Testing Database. Every time we need to run tests, we simply need to change a boolean flag on the Server, which will use the Testing Database. Every test can use the exposed API endpoint to reset the Testing Database.

- **Mock Server Service**
  Some tests don't use the real Server, because they focus on testing the frontend and

expect some default data. In order to perform such tests and exclude the network, we abstracted the communication with GreenJourney Server thanks to Swift protocols. We developed a Mock Server Service, which returns default data contained in JSON files, with the same format of real Server responses.

- **Mock Firebase Auth Service**
  All tests conducted in GreenJourney focus on proprietary logic, so in our tests, we want to avoid authenticating using Firebase Authentication API. That's why we abstracted the communication with the authentication API and developed Mock Firebase Auth Service, a mock service which fakes authentication.

- **SwiftData configuration**
  In order not to alter the content of SwiftData when running tests, we use a different configuration than the one used for the application. While running tests, we use an in memory configuration, which doesn't load the SQLite Database used for the real application.

We will now describe all types of tests we performed, their objective and how we conducted them.

5.3.2. Unit Tests

Unit Testing is a testing technique that verifies the functionality of small, individual components of a program. With Unit Tests, we want to test the behavior of functions for specific inputs and check that the output is the expected one. These tests allowed us to catch bugs in the application logic.
In our application, we wrote Unit Tests for both Model and ViewModel classes using the new library Swift Testing. We tested all public function that the ViewModel exposes to the Views and the logic implemented by Model classes.
Since the objective of Unit Tests is checking the correctness of individual functions, we used the mock version of the Server Service, which returns mock data, without interacting with Green Journey Server.

5.3.3. UI Tests

UI Testing focuses on checking that elements composing the Views behave as expected, ensuring a smooth user experience.
Since Swift Testing doesn't support UI Tests, we adopted another library offered by Apple, XCUITest. We wrote UI Tests for all Views composing our UI; some tests are

shared between the iOS and iPadOS versions, while others are specific to one version.
For UI Tests, we used the mock version of the Server, since we focused on testing the UI and dealing with static data allowed us to examine the content of the UI more precisely.

5.3.4. Integration Tests

Unlike Unit Testing, Integration Testing is a testing technique that verifies that multiple parts of an application work correctly together.
Such as for Unit Tests, we adopted the new library Swift Testing. We developed Integration Tests following the modular structure of our application, based on features. We started testing independent features and then integrated other features with the previously tested ones.
During Integration Testing, we used the real version of the Server Service and of GreenJourney Server; this allowed us to test the network communication and the ability of our fronted of communicating with the backend.

5.3.5. Code coverage and tests number

We tested the application in depth by writing a total of 353 tests, divided as reported in the following table.

Test type	Number of tests
Unit Tests	226
Integration Tests	38
UI Tests	89

Table 5.1: Number of tests

To measure overall code coverage, we ran tests on both iPhone and iPad. The total number of tests can be seen in the Xcode report in the following figure. The report also shows that some UI tests were skipped on one device type because they are intended to run only on the other.

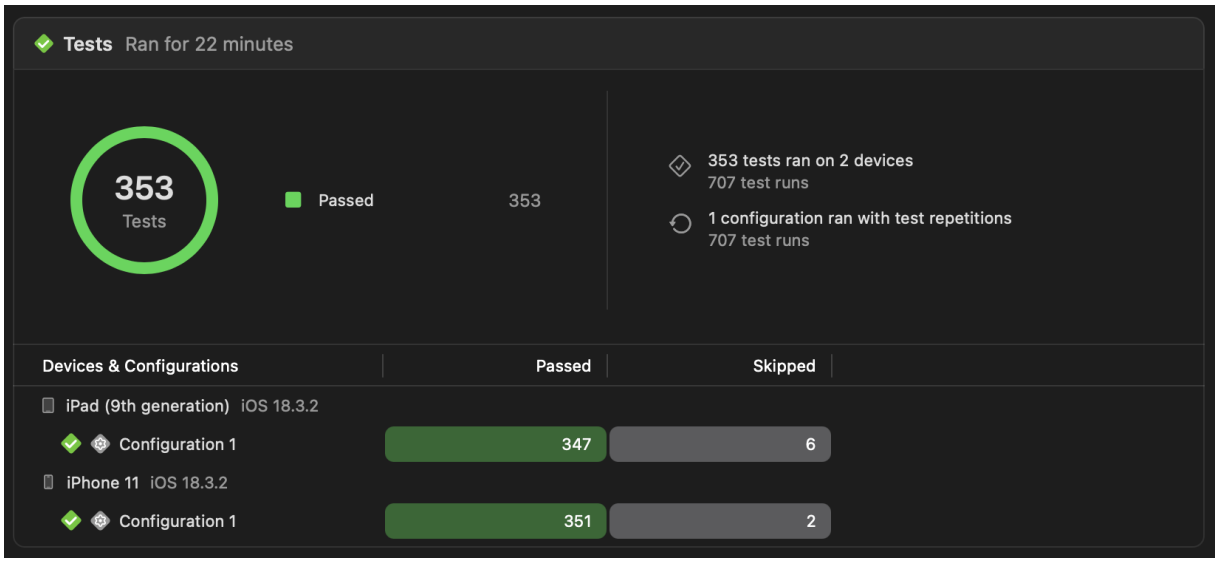

Figure 5.1: Total number of tests

Running these tests, we achieve a total coverage of 93.2%, as shown in Xcode code coverage report.

GreenJourney.app	93,2%	27.241	81,8%	84,9%
BadgeView.swift	100,0%	125	100,0%	100,0%
LegendBadgeView.swift	100,0%	226	100,0%	100,0%
InsertReviewButtonView.swift	100,0%	239	100,0%	100,0%
CircularProgressView.swift	100,0%	56	100,0%	100,0%
UserPreferencesButtonView.swift	100,0%	13	100,0%	0,0%
InfoCompensationView.swift	100,0%	148	100,0%	100,0%
HeaderView.swift	100,0%	490	48,8%	52,2%
DestinationPredictionViewModel.swift	100,0%	9	100,0%	100,0%
SignUpView.swift	100,0%	536	85,1%	86,2%
RankingResponse.swift	100,0%	4	100,0%	100,0%
CompleterView.swift	100,0%	279	100,0%	100,0%
OptionDetailsView.swift	100,0%	481	80,7%	91,7%
UserBadgesView.swift	100,0%	150	98,0%	94,7%
TopRoundedCornersView.swift	100,0%	23	100,0%	100,0%
StarView.swift	100,0%	112	99,1%	99,1%
CityReviewElement.swift	100,0%	44	100,0%	100,0%
ScoresView.swift	100,0%	364	100,0%	100,0%
DashboardView.swift	100,0%	513	100,0%	99,4%
Co2DetailsView.swift	100,0%	382	86,4%	89,8%
WorldExplorationView.swift	100,0%	407	86,7%	91,2%

Figure 5.2: Code coverage

6 | References

- https://firebase.google.com/docs/auth
- https://developers.google.com/maps/documentation/directions/overview
- https://developers.google.com/maps/documentation/distance-matrix/overview
- https://developers.google.com/maps/documentation/geocoding/overview
- https://developers.amadeus.com/self-service/category/flights/api-doc/flight-offers-search
- https://developers.amadeus.com/self-service/category/flights/api-doc/airport-and-city-search/api-reference
- https://developer.apple.com/documentation/swiftdata
- https://sqlite.org/docs.html
- https://developer.apple.com/documentation/createml
- https://pkg.go.dev/github.com/joho/Godotenv
- https://www.postgresql.org/docs/current/
- https://gorm.io/docs/index.html
- https://draw.io
- https://sequencediagram.org/instructions.html
- https://plantuml.com/class-diagram

List of Figures

List of Tables