POLITECNICO
MILANO 1863

Design and Implementation of Mobile Applications

Design Document

Guazzi Alessandro, Lei Leonardo, Mantegazza Niccolò

Prof. Baresi Luciano

Contents

1 Introduction **3**
- 1.1 Application Purpose . . . 3
- 1.2 Requirements . . . 3
- 1.3 Features Implemented . . . 4
  - 1.3.1 User Authentication . . . 4
  - 1.3.2 Trip Creation and Editing . . . 4
  - 1.3.3 Travel Method, Accommodation, and Activities . . . 4
  - 1.3.4 Expense Tracking . . . 4
  - 1.3.5 Trip Map Visualization . . . 5
  - 1.3.6 Public Trip Sharing . . . 5
  - 1.3.7 Community Feed and Favourite Trips . . . 5
  - 1.3.8 User Profile and Achievements . . . 5
  - 1.3.9 View Other Users' Profiles . . . 5
  - 1.3.10 Edit Profile Information . . . 5

2 Application Architecture **6**
- 2.1 Overview . . . 6
  - 2.1.1 Frontend Choice . . . 6
  - 2.1.2 Backend Choice . . . 7
- 2.2 Architectural Pattern . . . 7
- 2.3 Model Layer . . . 7
  - 2.3.1 Data Model (Frontend) . . . 7
  - 2.3.2 Data Management (Backend) . . . 8
- 2.4 View Layer . . . 9
- 2.5 Controller Layer . . . 9
- 2.6 External Services and Dependencies . . . 9
- 2.7 Project Folder Organization . . . 10
- 2.8 Sequence Diagrams . . . 11
  - 2.8.1 Login . . . 11
  - 2.8.2 User Creates a New Trip . . . 12
  - 2.8.3 User Adds a New Activity (Accommodation) to a Trip . . . 12
  - 2.8.4 User Edits Details of an Existing Trip . . . 13
  - 2.8.5 User Adds Created Trip to Calendar . . . 13

3 User Interface (UI) **14**
- 3.1 UI Design Choices . . . 14
- 3.2 Smartphone UI Details . . . 14
  - 3.2.1 Login and Registration . . . 15

3.2.2 Home Page . . . 15
3.2.3 Explorer Page . . . 16
3.2.4 Profile Page . . . 16
3.2.5 Medals Page . . . 17
3.2.6 Travel Stats Page . . . 18
3.2.7 Avatar Selection Page . . . 18
3.2.8 Account Settings Page . . . 19
3.2.9 Create Activity Page . . . 19
3.2.10 Edit Activity Page . . . 20
3.2.11 Map Page . . . 21
3.2.12 MyTrips Page . . . 21
3.2.13 Trip Page . . . 22
3.2.14 Upsert Trip Page . . . 23
3.3 UX Flowchart . . . 23
3.3.1 From Login/Authentication Page . . . 24
3.3.2 From Explorer Page . . . 24
3.3.3 From Profile Page . . . 25
3.3.4 From Trip Page . . . 25
3.4 Tablet UI . . . 25
3.5 Dark Mode . . . 26

4 Testing Campaign **27**
4.1 Testing Environment . . . 27
4.2 Unit Test . . . 27
4.3 Widget Tests . . . 28
4.4 Integration Tests . . . 33
4.5 Coverage Analysis . . . 34

1 Introduction

1.1 Application Purpose

The purpose of this application is to offer users a comprehensive and interactive travel planning experience. It is designed to help individuals easily create, organize, and manage their trips by allowing them to specify key travel details such as the mode of transportation, accommodations, and planned activities. In addition, the app includes a built-in cost management feature that enables users to keep track of the expenses associated with their travels.
Beyond personal trip planning, the application introduces a social component. Users with an account can share their trips with the community, browse the travel itineraries of other users, mark them as favorites, and get inspiration for their own future adventures. The app introduces gamification by awarding medals to users based on their achievements. Users can earn digital badges by completing trips and reaching specific milestones. These badges are displayed on the user's profile and are visible to the entire community, adding a motivational layer and a sense of accomplishment. This document explains the most important design choices we made and the motivations behind them.

1.2 Requirements

The following list contains the requirements that the application should satisfy.

ID	**Functional Requirement**
FR1	Users should be able to sign in/sign up in the app
FR2	Users should be able to logout from the app
FR3	Registered users should be able to edit their personal information
FR4	Registered users shall be able to create a new trip by specifying a title, travel dates and destination(s).
FR5	Registered users shall be able to add and edit Transportation method(s) within a trip.
FR6	Registered users shall be able to add and edit Accommodation(s) within a trip.
FR7	Registered users shall be able to add and edit Planned Activities within a trip.
FR8	Registered users shall be able to view a list of their own created trips.
FR9	Registered users shall be able to delete or modify existing trips.

FR10	Registered users shall be able to share their trips publicly with the community.
FR11	Registered users shall be able to browse trips shared by other users.
FR12	Registered users shall be able to mark shared trips as favourites.
FR13	Registered users shall be able to view a list of them favourited trips.
FR14	Registered users shall have a profile page displaying their medals.
FR15	The system shall assign medals to registered users based on predefined achievements (e.g., number of completed trips).
FR16	Registered users shall be able to view other users' profiles, including their medals.
FR17	Registered users shall be able to view a cost summary and detailed breakdown of expenses for each trip.
FR18	Registered users shall be able to view the map of a trip showing the destinations and relevant locations.

1.3 Features Implemented

The following list presents the core functionalities that have been implemented in the application, following a logical user interaction flow:

1.3.1 User Authentication

Registered users can sign up or log in using email and password credentials. A secure authentication system is implemented using Firebase Authentication. Users can also log out and access the app again later with persistent session support.

1.3.2 Trip Creation and Editing

Registered users can create a new trip by entering a title, destination(s), and travel dates. After creation, trips can be modified at any time to update transport, accommodation, activities, or notes.

1.3.3 Travel Method, Accommodation, and Activities

Within each trip, users can specify their transportation methods (e.g., train, plane), choose or describe accommodations, and add a list of planned activities, each with descriptions and metadata (e.g., date, time, location).

1.3.4 Expense Tracking

Each trip supports the management of associated costs. Registered users can input various expenses (transport, accommodation, activities, etc.) and view a summary of the total trip cost with breakdowns by category.

1.3.5 Trip Map Visualization

Users can view the map of each trip, which shows all destinations (cities) added during trip planning. This feature helps users visualize the geographic layout of their journey.

1.3.6 Public Trip Sharing

Trips can be marked as public or private. When shared, they become visible to the community of registered users. Each shared trip includes all core information and is displayed in the community feed.

1.3.7 Community Feed and Favourite Trips

Registered users can explore trips shared by others in a dedicated community section. Each trip can be opened for details, and users can mark them as favourites. A dedicated screen shows all favourited trips for quick reference and inspiration.

1.3.8 User Profile and Achievements

Each registered user has a profile page where their shared and completed trips are listed. A gamification layer is implemented: as users complete trips, they earn medals (badges) that are visible on their profile. These achievements add a social and motivational dimension to the platform.

1.3.9 View Other Users' Profiles

Users can tap on the author of a shared trip to view their public profile, including their badges, shared trips, and a summary of their activity within the app.

1.3.10 Edit Profile Information

Users can modify their personal information (e.g., name, profile picture, bio) directly from the settings or profile screen. Changes are reflected immediately across the app.

2 Application Architecture

2.1 Overview

At a high level, the project can be described as a client–server architecture where:

- **Frontend:** Implemented entirely in the Flutter framework, supporting both iOS and Android operating systems.
- **Backend:** Implemented using Firebase services, which provide both authentication and data storage:
  - Firebase Authentication manages secure sign-up, login, and session persistence.
  - Firebase Firestore is a document-based cloud database used for storing produced data such as users, trips, and activities.
  - Firebase Core acts as the "engine" that allows communication between Firebase and the mobile application.

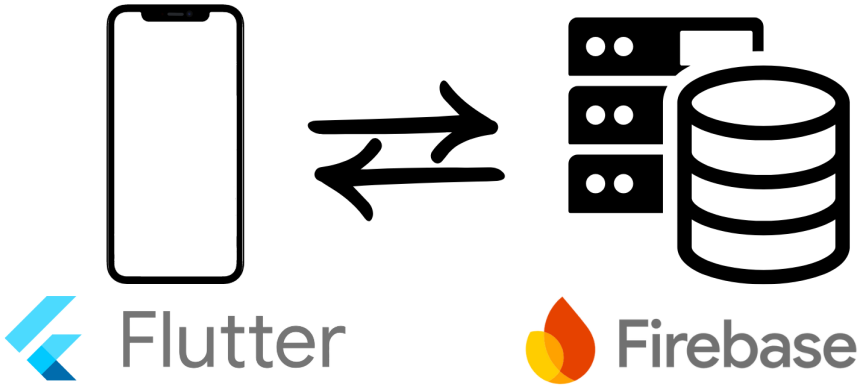


Figure 2.1:

2.1.1 Frontend Choice

For the frontend we decided to use the Flutter framework, because it allows us to create a single codebase that works both on Android and iOS.

The main advantages of Flutter in our project were simplicity and development speed. Its built-in widgets and packages allowed us to implement the required features without

unnecessary complexity, while the hot-reload functionality reduced development time by enabling immediate testing and adjustments.

2.1.2 Backend Choice

For the backend we chose Firebase, since it already provides many of the services we needed without having to build a custom server. We used:

- Firebase Authentication to manage login and registration securely.
- Cloud Firestore as the main database to store users, trips, and activities.

One of the main reasons for this choice is that Firebase integrates easily with Flutter, so we didn't need to spend time writing complex backend logic. In this way we could focus more on the frontend and the user experience, while still having a scalable and reliable backend managed by Google's infrastructure.

2.2 Architectural Pattern

The application follows a simplified architectural pattern inspired by the principles of clean architecture, with a clear separation between data, interface, and control logic.

At its core, the structure can be described as a Model–View–Controller (MVC) approach:

- **Model:** Represents and manages the application's data.
- **View:** Handles the user interface.
- **Controller:** Manages the interaction between the model and the view.

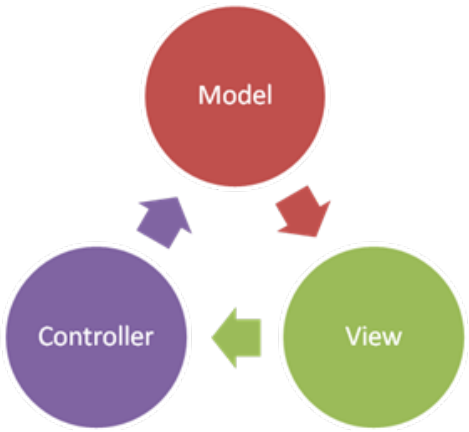


Figure 2.2:

2.3 Model Layer

2.3.1 Data Model (Frontend)

The data layer of the application is organized around a set of core models, designed to support trip planning, user management, and travel-related features. All models are

structured for compatibility with Cloud Firestore, allowing for efficient querying and real-time updates. The main models include:

- **Trip Model:** Represents a complete journey with metadata such as title, creator, travel dates, visited countries/cities, activities, expenses, and visibility.
- **Attraction Models:** define individual events or experiences within a trip, such as visits, tours, or entertainment. These models store basic information like type, optional description, and cost.
- **Accommodation, Flight & Transport Models:** represent different forms of logistics associated with a trip, including lodging and transportation by air, car, or train. They include common attributes like timing, location, and cost.
- **User Model:** stores personal information, preferences, and social data for each registered user, including profile details, created and saved trips, and visited countries.
- **Airport Model:** Contains metadata about airports, used in relation to flights.

2.3.2 Data Management (Backend)

The application uses Cloud Firestore as its backend for persistent, real-time data storage. Data is structured into collections, with each collection containing documents representing instances of the app's core models. The three primary collections are:

- **Users Collection:** The users collection stores the data of all registered users. Each document is identified by a unique userId corresponding to the authenticated Firebase UID. The document includes the user's personal information (e.g., name, surname, email, username), profile picture, birth date, and biography. It also keeps references to user-created trips, and visited countries, typically as lists of identifiers.
- **Trips Collection:** The trips collection contains all the trips created by users. Each trip document stores information such as the title, creatorInfo (user metadata), start and end dates, visited nations and cities, and trip visibility status (e.g., public or private). It also includes structured data such as expenses per category, a list of related activity IDs, and metadata like save counters, confirmation state, and a cover image reference.
- **Activities Collection:** The activities collection stores all activities associated with trips. Each activity is linked to a trip through a tripId, and contains information such as activity type, description, costs, and possibly embedded models (e.g., flight, transport, accommodation, attraction). By decoupling activities from the trip document itself, the app maintains a modular and scalable structure that facilitates individual activity management.

These collections are accessed through asynchronous Firestore calls using the cloud-firestore package, and the data is immediately mapped into our model classes for easy use.

2.4 View Layer

The view layer consists of Flutter widgets that render the interface and manage visual state. Each screen is represented by a dedicated widget tree. Layouts are designed to be responsive, adapting automatically for smartphones in portrait mode and tablets in landscape mode (although portrait is still supported).

2.5 Controller Layer

The controller responsibilities are divided between dedicated files and widget-level logic. Controllers that handle interactions with external services are implemented in separate files, ensuring modularity and clearer separation of concerns. However, some controller code that manages logic tightly coupled with the UI is embedded within stateful widgets, separated from the view code through dedicated methods. These controllers handle user inputs, trigger asynchronous operations such as Firestore queries, and update the local state to ensure that changes are reflected in the UI in real time.

2.6 External Services and Dependencies

The app integrates several external APIs and packages:

- **Google Places API**: Location-based autocomplete and place details.
- **Unsplash API:** High-quality travel-related images for trip previews.
- **Hexarate Currency Exchange API:** Real-time exchange rates for trip budgeting.

For geolocation and map rendering, the app uses:

- **Google Maps(google_maps_flutter):** Interactive maps and trip destinations.
- **countries_world_map:** Highlights visited countries on a world map.

Several additional packages are included to enhance usability and interface functionality:

- **country_picker:** Country selection for trips or profile setup.
- **intl:** Date formatting and internationalization.
- **http:** Simplified HTTP API calls.
- **flutter_typeahead:** Autocomplete UI component (Google Places data).
- **pie_chart:** Displays trip expenses summary visually.
- **add_2_calendar:** Adds trips to personal calendars.
- **mockito:** Testing framework for mocks, behavior simulation, and verification.

Here is a complete list of all dependencies from the pubspec.yaml file of our flutter project.

```yaml
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.8
  firebase_core: ^3.12.1
  firebase_auth: ^5.5.1
  cloud_firestore: ^5.6.5
  get: ^4.7.2
  country_picker: ^2.0.27
  intl: ^0.20.2
  http: ^1.3.0
  flutter_typeahead: ^5.2.0
  google_maps_flutter: ^2.10.1
  pie_chart: ^5.4.0
  dropdown_button2: ^2.3.9
  add_2_calendar: ^3.0.1
  mockito: ^5.4.4
  duration_picker: ^1.0.2
```

```yaml
dev_dependencies:
  flutter_test:
    sdk: flutter
  integration_test:
    sdk: flutter
  flutter_launcher_icons: ^0.14.3
  percent_indicator: ^4.2.4
  countries_world_map: ^1.2.3
  flutter_lints: ^4.0.0
  build_runner: ^2.4.13
```

Figure 2.3: Project dependencies

2.7 Project Folder Organization

The project follows a structured folder hierarchy:

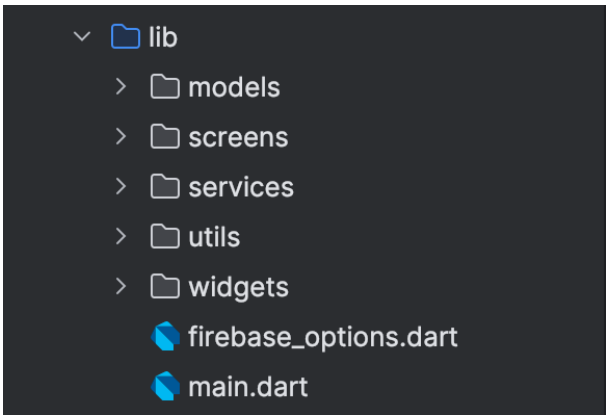


Figure 2.4: Project structure

- `models/` — Core data models (e.g., UserModel, TripModel).
- `screens/` — Main UI pages (each representing a screen/view).
- `services/` — Business logic and external interactions (auth, Firestore).
- `utils/` — Helper functions, constants, utilities.
- `widgets/` — Reusable UI components (buttons, cards, custom elements).

2.8 Sequence Diagrams

Sequence diagrams offer a visual representation of how different components or objects interact and cooperate within our system. These diagrams provide a clear understanding of the system's behaviour, communication patterns, and the responsibilities of each element.

For these reasons, the sequence diagrams of the most relevant actions of our application are shown.

2.8.1 Login

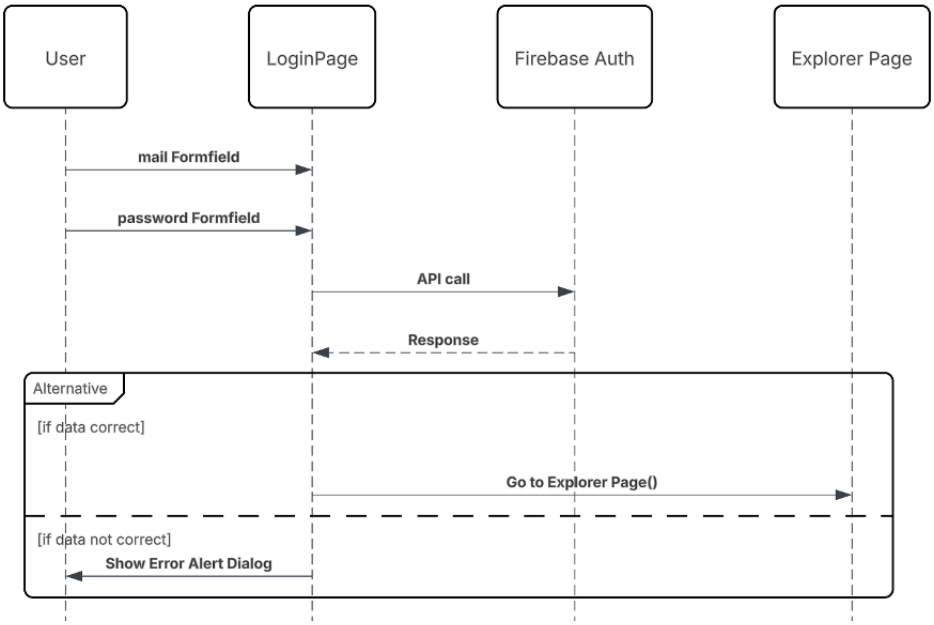


Figure 2.5: Sequence Diagram: Login

2.8.2 User Creates a New Trip

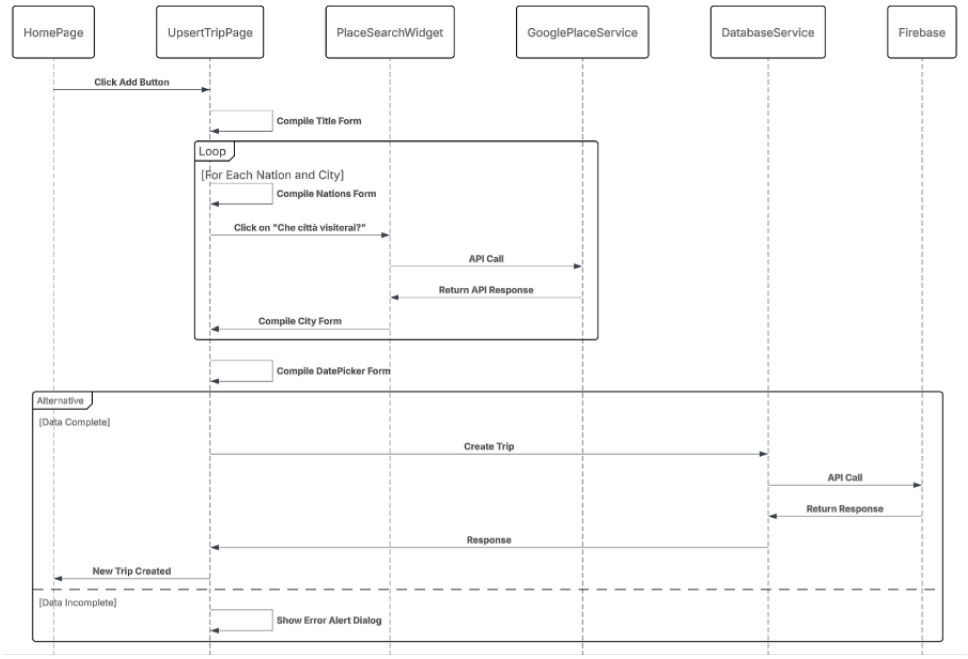


Figure 2.6: Sequence Diagram: User Creates a New Trip

2.8.3 User Adds a New Activity (Accommodation) to a Trip

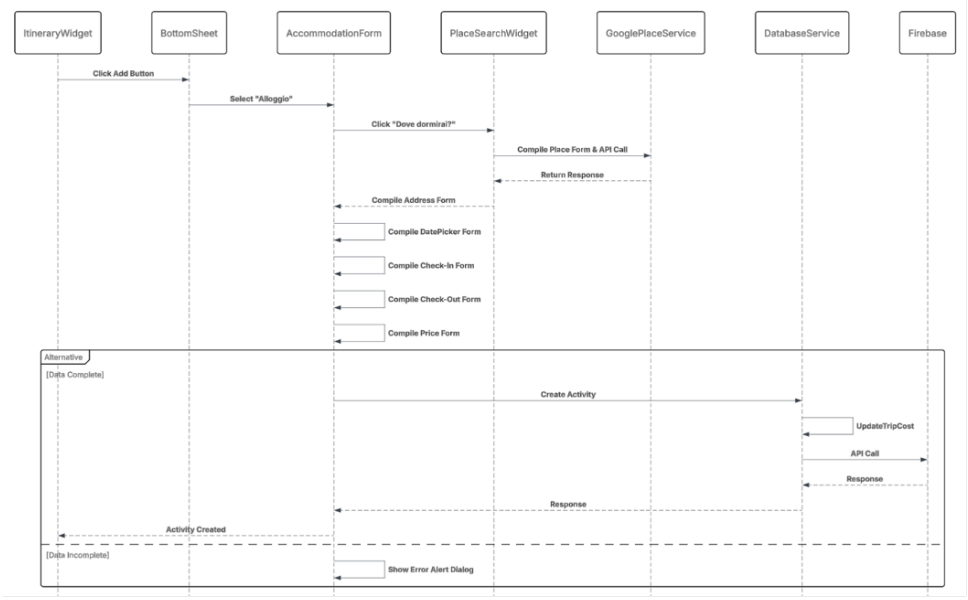


Figure 2.7: Sequence Diagram: Add New Activity (Accommodation)

2.8.4 User Edits Details of an Existing Trip

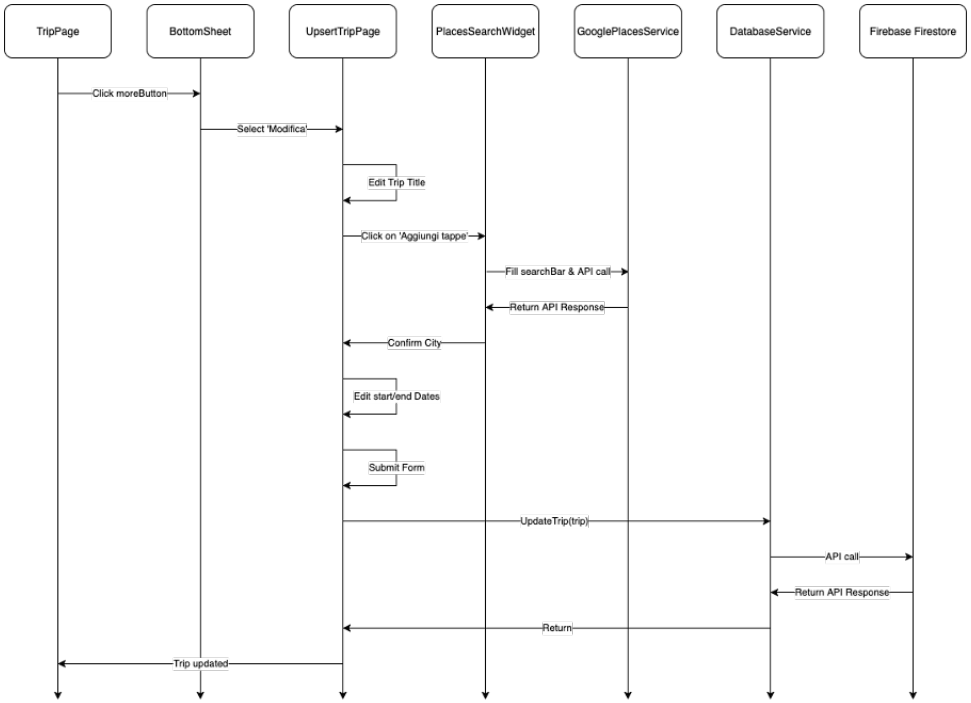


Figure 2.8: Sequence Diagram: Edit Trip Details

2.8.5 User Adds Created Trip to Calendar

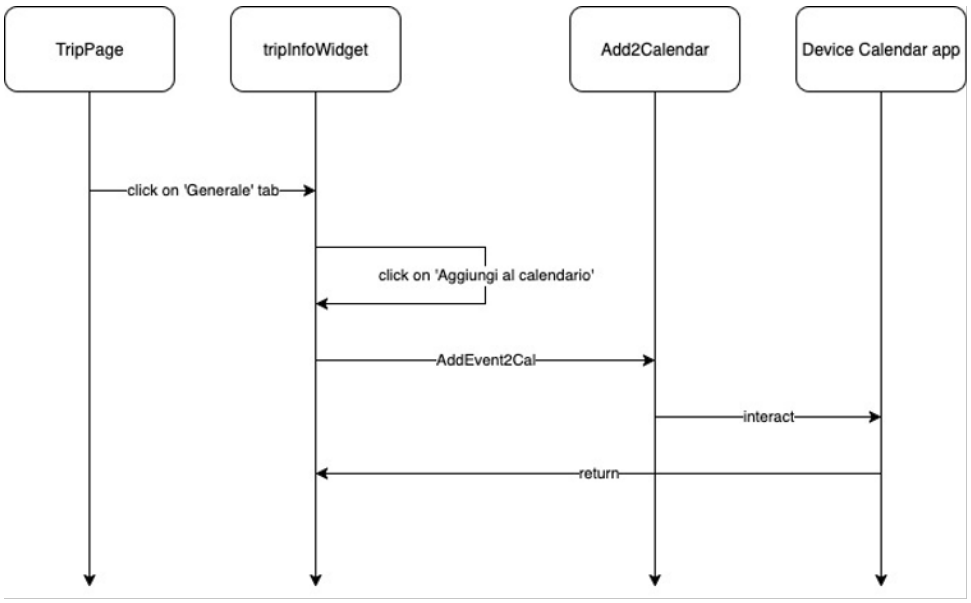


Figure 2.9: Sequence Diagram: Add Trip to Calendar

3 User Interface (UI)

This section outlines the design and layout of the application's main screens. The goal is to provide users with an intuitive experience as they plan trips, explore content, and interact with their profiles. The layout automatically adapts to different devices, ensuring that both smartphones and tablets make full use of the available screen space.

In the following subsections, we describe the key interface elements and the reasoning behind our design choices, focusing on layout, navigation, and visual consistency.

3.1 UI Design Choices

The application's user interaction model is built around simplicity and clarity, ensuring smooth navigation without overwhelming the user. Main navigation is handled through a bottom navigation bar, while most pages are opened by pushing them onto the stack using Flutter's Navigator.push.

For interactions that require structured input—such as creating or editing a trip, entering budget details, or updating profile information—we rely on Flutter's Form widget in combination with TextFormField. This choice allows us to enforce input validation and offer immediate visual feedback.

When users need to choose from a set of options without navigating to a new page, the interface employs bottom sheets. This approach keeps the experience lightweight and avoids unnecessary context switches.

For secondary feedback or non-critical actions, we use snackBars. These appear briefly at the bottom of the screen and then disappear automatically, ensuring that important messages are conveyed without interrupting the user's flow.

3.2 Smartphone UI Details

This section provides a detailed, screen-by-screen analysis of the application, presenting an overview of each main screen and its features.

3.2.1 Login and Registration

The first screen displayed when the application is opened for the first time or after a logout is the login page. It provides users with a minimal and clean interface to enter their email and password credentials. A text button at the bottom of the form allows users to easily switch between the login and registration modes. This toggle mechanism ensures that both actions are accessible from the same screen without additional navigation steps.
The registration form requires users to input basic personal information along with their credentials. Input validation is performed to prevent incomplete or incorrect data submission. Upon successful login or registration, users are redirected to the home page of the application, where they can begin creating or exploring trips.

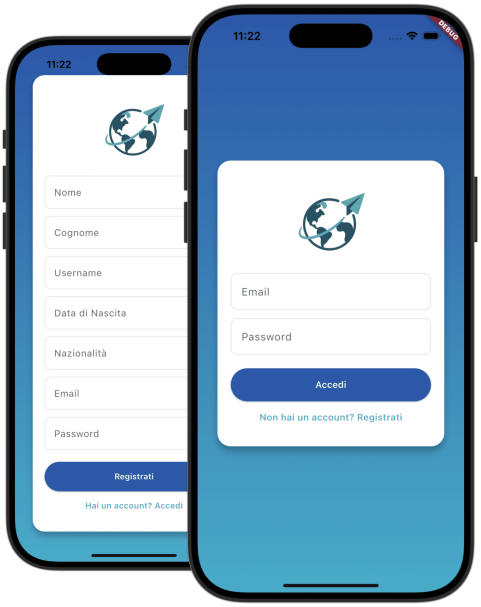


Figure 3.1: Login and Authentication Pages

3.2.2 Home Page

The Home Page represents the structural foundation of the application and serves as the container for navigating between all major sections. It simply consists of:

- App Bar (top): A minimal bar displaying the application name and a circular avatar that opens the Profile page.
- Navigation Bar (bottom): Contains icon-based navigation items, each representing a distinct section of the app. Tapping an item switches the central content area, without affecting the App Bar. Additionally, it includes an add button for creating a new trip.

These two elements provide quick access to the primary areas of the app: the Profile page via the App Bar, and the Explorer and MyTrips pages via the bottom navigation bar. This layout is built using Flutter's Scaffold widget, offering a simple and organized way to manage main screens while keeping navigation smooth.

3.2.3 Explorer Page

The Explorer Page provides access to the social and community-driven side of the application. In this screen, registered users can browse through trips publicly shared by other users. Each trip is presented as a card containing essential information such as the title given by the creator, the destinations, and a cover image. By tapping on a trip, users can view its detailed content, including transportation, accommodation, activities, and budget information.
The Explorer Page includes a dedicated search bar at the top of the screen, allowing users to quickly find trips by entering their title.
Additionally, the page supports sorting and filtering options to organize trips according to different criteria – such as most recent or most liked – allowing users to easily discover relevant or popular content.
Finally, users can mark trips as favourites with a simple interaction; favourited trips are then saved in a dedicated section for later consultation.

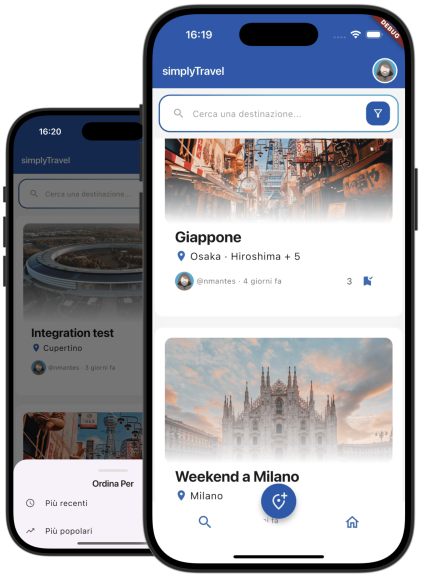


Figure 3.2: Explorer Page

3.2.4 Profile Page

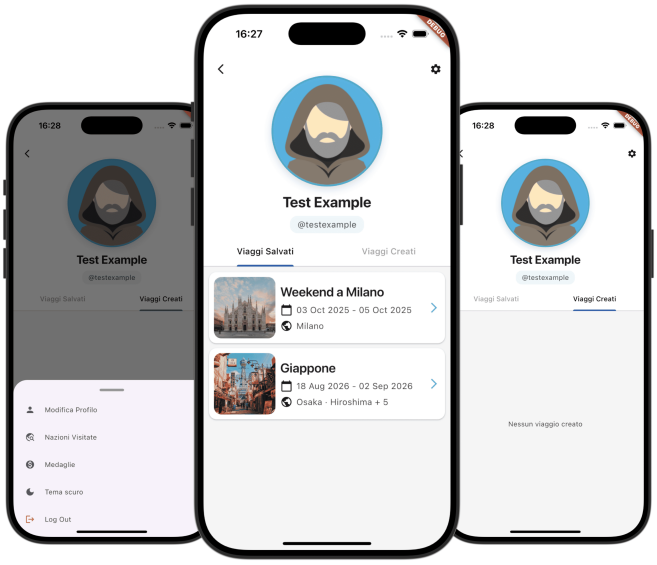


Figure 3.3: Profile Page

The Profile Page provides users with an overview of their identity and activity within the application. At the top of the screen, primary profile information is displayed, including the user's profile picture and display name. A settings button is available to access the

profile editing interface, where users can modify their personal data and update their profile image.

Below the personal information section, the profile page is divided into two main tabs:

- Created Trips: this section lists all the trips the user has created, regardless of their visibility status (public or private). Each trip can be tapped to open its detailed view, where it can be edited or deleted.
- Saved Trips: this section contains all the trips the user has marked as favourites while exploring the community. These trips are read-only and provide quick access to inspirational journeys made by others.

3.2.5 Medals Page

The Medals Page introduces a gamified element to the application, designed to reward users for their travel achievements and to visually track their global exploration progress. The page displays a collection of achievement medals, divided by geographical region: there are 5 medals for each continent, plus an additional set of 5 global medals representing overall accomplishments.
Each medal corresponds to a specific goal, for example visiting at least one country in a continent or reaching a percentage of coverage of countries within a continent.
Medals are visually represented as locked or unlocked depending on the user's progress. Once earned, they become visible in the user's collection and are also shown on their public profile. This gamification system adds an engaging incentive to travel planning and encourages users to explore new parts of the world.

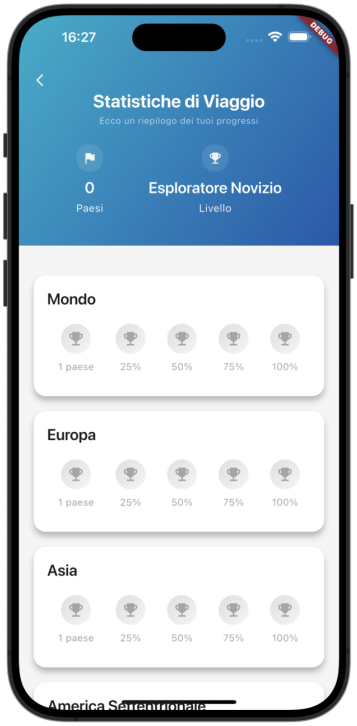


Figure 3.4: Medals Page

3.2.6 Travel Stats Page

The Travel Stats Page provides users with a visual and quantitative summary of their travel history across the globe. At the core of the page is a world map, initially displayed in grayscale. As the user completes trips that include countries in their itineraries, the corresponding nations on the map are highlighted dynamically, allowing users to visually track the extent of their global exploration.
Below the map, the page displays a set of progress bars, one for each continent. Each progress bar reflects the number of countries visited in that specific continent, based on the destinations included in the user's completed trips.

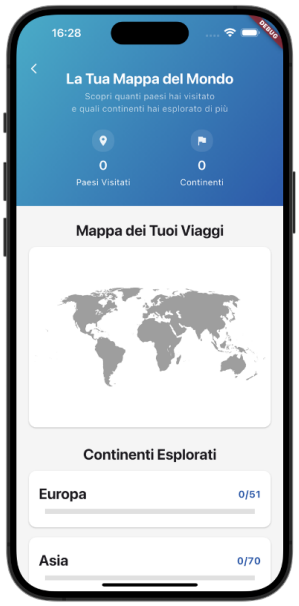


Figure 3.5: Travel Stats Page

3.2.7 Avatar Selection Page

The Avatar Selection Page allows users to personalize their profile by choosing a profile image from a set of predefined avatars. Upon accessing this page, users are presented with a grid layout showcasing a collection of avatar icons that represent a variety of styles and identities. By tapping on an avatar, the user immediately updates their profile picture, which is then reflected across the application. This approach provides a lightweight alternative to uploading a personal photo.

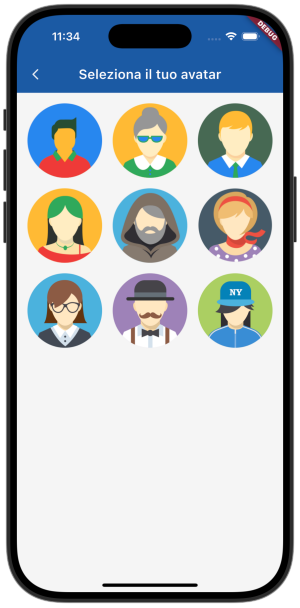


Figure 3.6: Avatar Selection Page

3.2.8 Account Settings Page

The Account Settings Page allows registered users to manage and update their basic personal information associated with their account. This page includes editable fields for first name, last name, username, birthdate, and other optional profile details such as a short bio.
The form is pre-filled with the current user data retrieved from the backend (e.g., Firestore), and supports real-time editing with input validation to ensure data integrity. Changes can be confirmed via a dedicated Save button, which updates the information in the database and immediately reflects the changes across the app, including in the user's profile and public views.

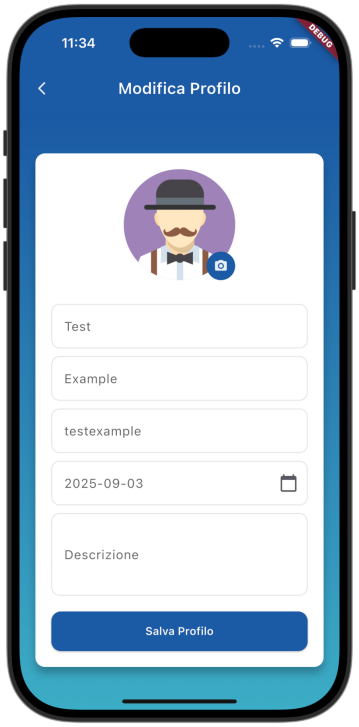


Figure 3.7: Account Settings Page

3.2.9 Create Activity Page

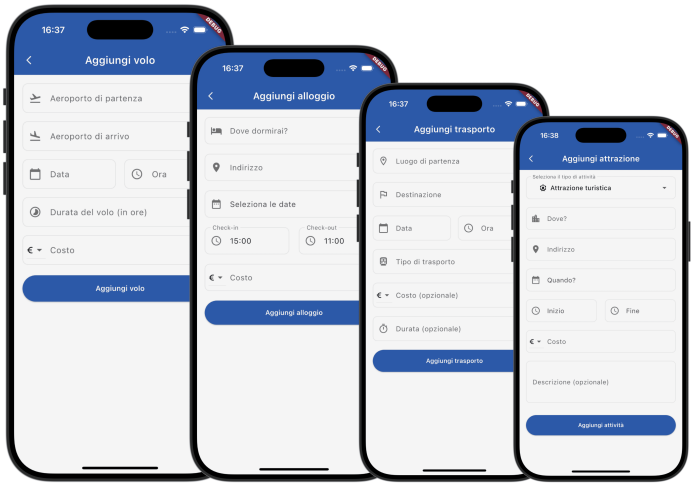


Figure 3.8: Create Activity Pages

The Create Activity Page enables users to add new elements to a trip, making their travel plan more detailed and personalized. When creating an activity, the user first selects its type—flight, accommodation, transport, or attraction. Based on this choice, the

page adapts to display the most relevant fields, ensuring that the information entered is appropriate for that activity. The form remains simple and intuitive, allowing users to fill in essential details such as the activity name, an optional location, timing, and additional notes. Once saved, the new activity is added to the chosen trip and immediately visible in its detailed view.

3.2.10 Edit Activity Page

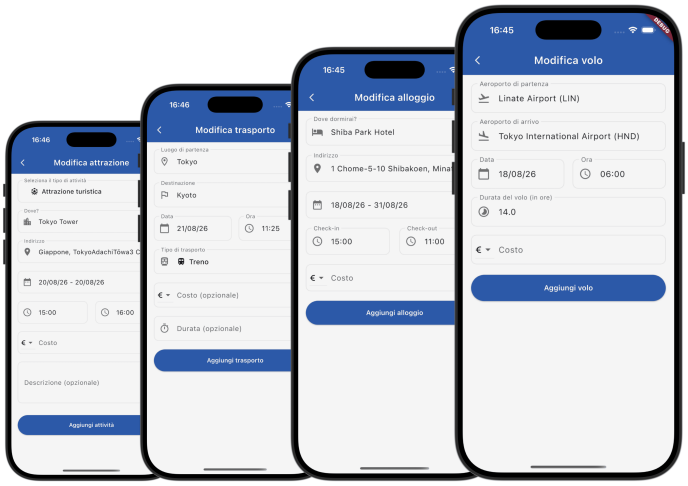


Figure 3.9: Edit activity Pages

The Edit Activity Page allows users to modify the details of an existing activity within a trip. The page presents a form pre-filled with the current data of the selected activity, allowing for easy adjustment. Depending on the type of activity, the fields shown may vary to reflect the specific attributes associated with that category. Users can update general information such as the title, location, date and time, and any additional notes or content related to the activity. Once the modifications are saved, the updated activity is stored in the database and immediately reflected in the trip's detailed view. From an architectural perspective, the same page used for creating an activity is reused in this procedure.

3.2.11 Map Page

The Map Page provides a geographical visualization of a trip by displaying all relevant locations associated with that journey on an interactive map. Each destination (city) is represented as a marker, giving users a clear and intuitive overview of the spatial distribution of their trip.
The map is built using **Google Maps**, allowing for smooth interaction such as zooming, panning, and tapping on individual markers to view basic details about the corresponding place.
The Map Page is accessible from within each trip's detail view and serves as a useful complement to the textual list of activities and destinations.

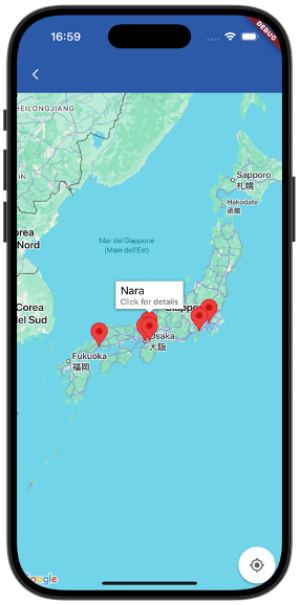


Figure 3.10: Map Page

3.2.12 MyTrips Page

The MyTrips Page is one of the two main screens displayed after a successful login and acts as the central hub for the user's travel planning activities. It presents a scrollable list of all trips created by the user, each shown with a concise summary including the title, destination, and date range.
To improve organization, the trips are divided into two tabs: **Upcoming Trips** and **Past Trips**. This layout places greater emphasis on upcoming journeys, helping users focus on their future travel plans while still allowing access to their travel history.
Each existing trip in the list can be tapped to access its detailed view, where users can further edit, review, or delete it.

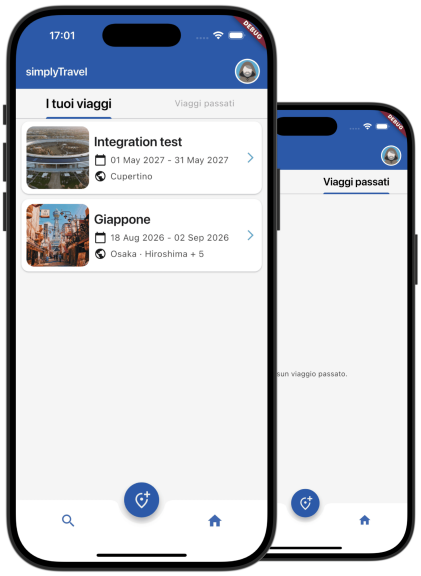


Figure 3.11: MyTrips Page

3.2.13 Trip Page

Figure 3.12: Trip Page

The Trip Page provides a complete and structured view of a single trip, bringing together all its components in one place. Navigation within this page is managed through a tab bar, which allows users to switch between the Itinerary, General Info, and Cost Summary sections. At the top, the trip's basic details are displayed, including the title and the dates. The Itinerary tab presents a chronological list of planned activities and events. Additionally each activity is expandable for more details or editing options. The General Info tab offers access to additional features, such as adding the trip to the user's calendar and opening the interactive map view, which visualizes all trip locations with markers for better geographic context. Finally, the Cost Summary tab displays a visual breakdown of expenses across categories like transportation, accommodation, and activities, helping users stay aware of their budget and spending.

3.2.14 Upsert Trip Page

The Upsert Trip Page serves as a unified interface for both creating a new trip and editing the details of an existing one. The page presents a structured form that guides the user through the essential components of a trip, including the title, visited countries and travel dates.
When accessed in creation mode, the form is empty and ready to be filled with new information. In edit mode, it is pre-populated with the current details of the selected trip, allowing users to adjust easily.
The interface includes validation to ensure the correctness of the input (e.g., non-empty title, valid date ranges).

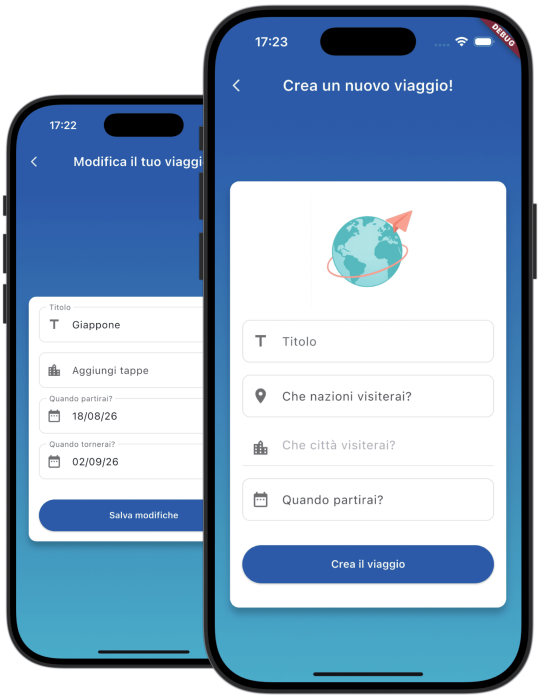


Figure 3.13: Upsert Trip Page

3.3 UX Flowchart

The UX Flowchart provides a visual overview of the application's user experience, illustrating how users navigate between screens and interact with various features.

The following presents the main flowchart of our application:

3.3.1 From Login/Authentication Page

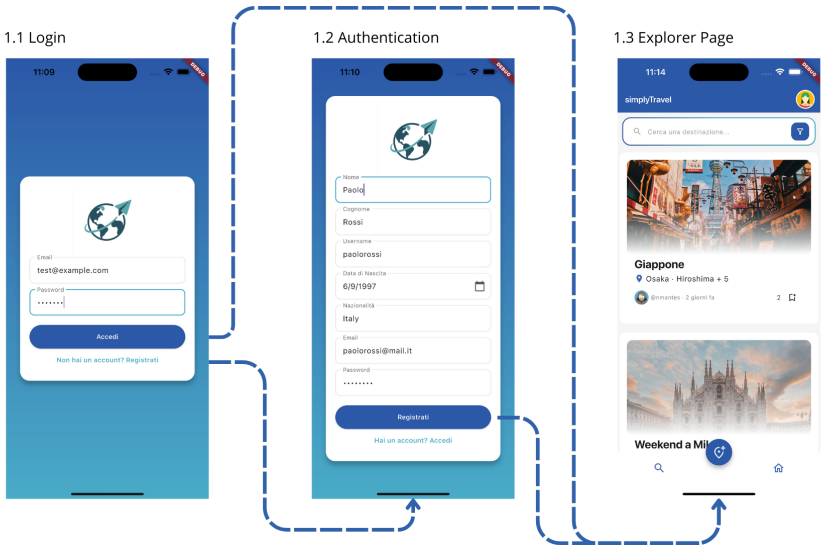


Figure 3.14: UX Flowchart: From Login/Authentication Page

3.3.2 From Explorer Page

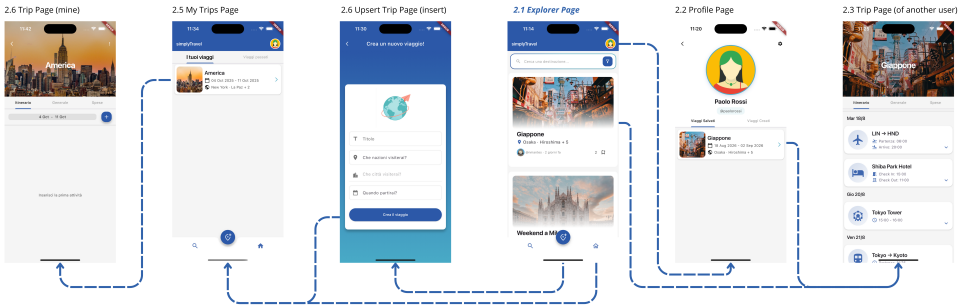


Figure 3.15: UX Flowchart: From Explorer Page

3.3.3 From Profile Page

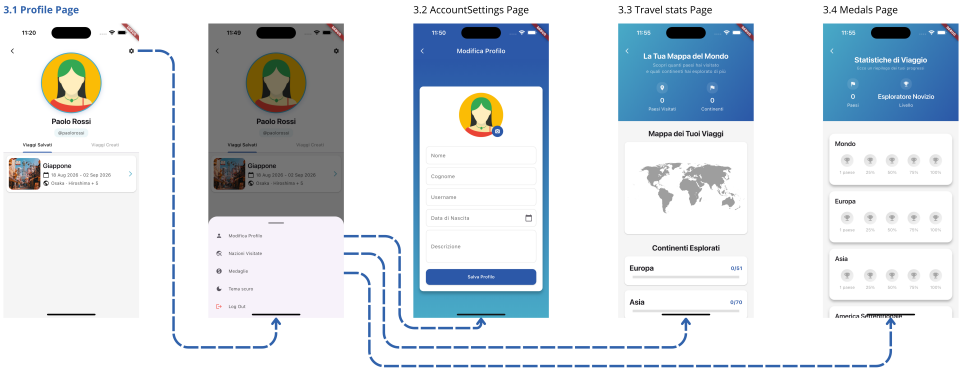


Figure 3.16: UX Flowchart: From Profile Page

3.3.4 From Trip Page

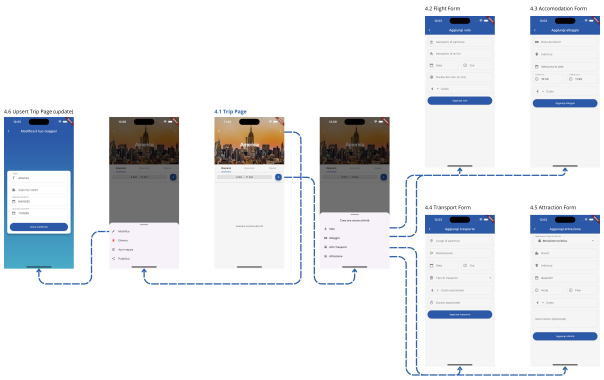


Figure 3.17: UX Flowchart: From Trip Page

A more clear view of these diagrams can be found here.

3.4 Tablet UI

For the tablet version of the application, we designed a custom layout for the main screens, optimized for larger displays rather than simply scaling up the mobile interface. While the mobile app follows a sequential navigation flow, the tablet interface adopts a master–detail pattern (whenever possible). In this layout, the left panel (master) lists items such as trips, while the right panel (detail) shows the selected content. This structure lets users view and interact with information more efficiently, without constantly navigating back and forth.

The detail views reuse the same screens as the mobile application to ensure design consistency and component reuse. Minor adjustments were applied so these views can fit alongside the master panel, improving usability on larger screens. Responsive layouts were implemented using a custom Responsive widget wherever needed, which uses a LayoutBuilder to return the appropriate layout based on the screen size. To keep the

project structure simple, we did not create separate files for tablet pages; instead, we maintained a logical separation within each page file, using functions or methods to handle the different layouts for mobile and tablet.

Below are example of the Explorer page and the Profile Page:

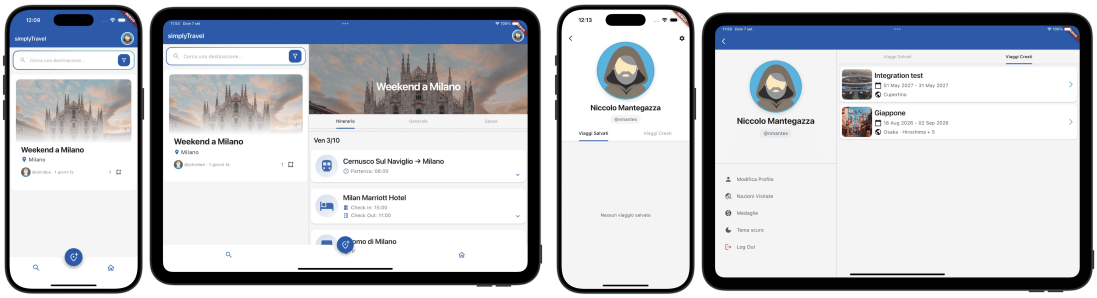


Figure 3.18: Examples of Tablet UI: Explorer and Profile Pages

For other simpler parts of the app, like activity insertion forms or other input screens, we kept the same components from the mobile version. They were only adjusted to scale better on the tablet layout. In the same way, some pages that did not need major changes, such as the TravelStats page, were kept almost the same with only small adjustments.

This design approach takes advantage of the available screen space on tablets to create a more effective and user-friendly experience, while also reducing the number of navigation steps compared to the mobile version.

3.5 Dark Mode

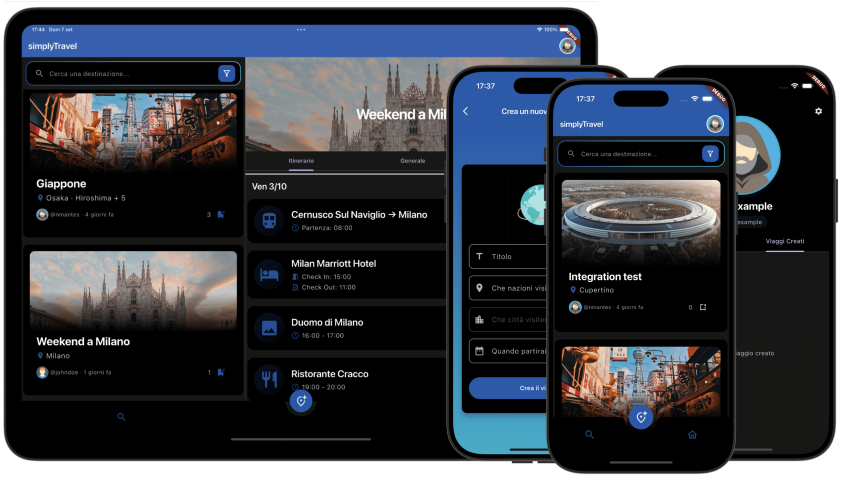


Figure 3.19: Dark mode scree

We decided to develop a Dark Mode since it is a very popular choice among users and improves comfort in low-light environments. However, the main theme of the application remains the Light Mode, which serves as the default design.

4 Testing Campaign

4.1 Testing Environment

The tests developed for the application are performed in two different environments according to the kind of test to perform:

- **Mock Environment:** In this environment all the external services are replaced with a mock version of them; this is done to separate the behaviour of the services with respect to the implemented function that we tested and to overcome the limitation of the Flutter test environment, where no external services are working. Thanks to the architecture of the application, this environment is implemented simply by changing the implementation of our service classes with static data.
- **Deploy Environment:** In this environment the tests are supposed to run on a real device, this is costly and slows down the testing process, but is needed to test the integration of our app with the reliable external services we choose and to test the integration of different parts of the app.

4.2 Unit Test

Flutter unit tests are supposed to run in a mock environment due to Flutter constraints. Since our app depends mostly on external services, the number of unit tests is limited. They mainly focus on testing the models, as most of the application logic consists of straightforward calls to these external services with different parameters.

Here is the list of unit tests (grouped by the model they test):

Model	**Test Description**
TripModel	Verify `fromFirestore` correctly deserializes Firestore data into a model instance
TripModel	Verify `toFirestore` correctly serializes a model instance into a Firestore-compatible map
AttractionModel	Verify `fromFirestore` correctly deserializes Firestore data into a model instance
AttractionModel	Verify `toFirestore` correctly serializes a model instance into a Firestore-compatible map
AirportModel	Verify `fromJson` correctly creates an Airport object

AirportModel	Verify `fromJson` handles missing fields
UserModel	Verify `fromFirestore` correctly deserializes Firestore data into a model instance
UserModel	Verify `toFirestore` correctly serializes a model instance into a Firestore-compatible map

Only `AttractionModel` is reported here as an example for activity models, since all activity model tests follow the same structure.

Additionally, some pages contain a limited number of unit tests that test minor computational functions:

Page	**Test Description**
Medals Page	Should extract country codes from valid trips
Medals Page	Should handle trips with null nations
Medals Page	Should handle trips with empty nations
Medals Page	Should handle nations with null or empty codes
Medals Page	Should return empty list for empty trips
Medals Page	Should count countries correctly by continent
Medals Page	Should handle empty country list
Medals Page	Should handle countries from single continent
Medals Page	Should handle invalid country codes
Medals Page	Should remove duplicates from list
Medals Page	Should calculate correct level based on percentage
TripExpense Page	Should calculate expenses percentages correctly

4.3 Widget Tests

The aim of this type of testing is to ensure that a proper widget is loaded and populated successfully, by feeding it with some data coming from mock APIs and checking through proper assertions if it is correctly displayed. Flutter Widget tests run in the mock environment. Both tablet and smartphone widgets are tested on the same kind of test, with small modification to match the relative design. The list of the most important tests is available below (divided by screen):

Page Tested	**Test Description**
Login/Registration	Should start in login mode by default
Login/Registration	Should switch to registration mode when toggle is tapped

Page Tested	**Test Description**
Login/Registration	Should reset form when switching from login to registration
Login/Registration	Should not show validation errors when all fields are filled correctly
Login/Registration	Should call signInWithEmailAndPassword on successful login
Login/Registration	Should call createUserWithEmailAndPassword on successful registration
Login/Registration	Should handle FirebaseAuthException on login failure
Login/Registration	Should handle FirebaseAuthException on registration failure
Home Page	MyHomePage shows AppBar and NavigationBar
Home Page	NavigationBar switches between pages correctly
MyTrips Page	Should show loading indicator while waiting for trips
MyTrips Page	Should show empty message when no trips
MyTrips Page	Should show upcoming trips in the correct tab
MyTrips Page	Should show past trips
MyTrips Page	Should show both upcoming and past trips
MyTrips Page	Should show error message on failure
MyTrips Page	Should renders tablet layout and allows selecting a trip
Explorer Page	Should displays loading indicator while fetching user
Explorer Page	Should show empty message when no trips
Explorer Page	Should shows trips when available
Explorer Page	Should sort trips by date
Explorer Page	Should sort trips by popularity
Explorer Page	Should updates save counter when save button is clicked
Profile Page	User is correctly loaded when authenticated
Profile Page	Should handles when user not authenticated
Profile Page	signOut calls AuthService.signOut
Profile Page	initState initializes futures correctly

Page Tested	**Test Description**
Profile Page	Should display loading indicator while fetching user data
Profile Page	Should display ui elements correctly
Profile Page	Should display default profile image when user has no profile pic
Profile Page	Should display custom profile image when available
Profile Page	Should display error message when user data fails to load
Profile Page	Should display mobile layout on small screens
Profile Page	Should display tablet layout on large screens
Profile Page	Should display saved trips correctly
Profile Page	Should display empty message when no saved trips
Profile Page	Should display empty message when no created trips
Profile Page	Should navigate to TripPage when trip card is tapped
Profile Page	Should open settings modal on mobile when settings icon is tapped
Profile Page	Settings menu displays all options
Profile Page	Should handle errors during trip loading
Trip Page	Correctly renders trip info when trip is created by the logged user
Trip Page	Correctly renders trip info when trip was created by other users
Trip Page	Should correctly render actions options
Trip Page	Tapping on 'Apri mappa' navigate to 'MapPage'
Trip Page	Privacy toggle updates trip privacy
Trip Page (ItineraryWidget)	Should render activities from database
Trip Page (ItineraryWidget)	Should open bottom sheet when add button is pressed
Trip Page (ItineraryWidget)	Should render add activity button and navigates to Create Activity Page on tap
Trip Page (ItineraryWidget)	FlightCard widget displays data and expands correctly

Page Tested	**Test Description**
Trip Page (ItineraryWidget)	AccommodationCard Widget displays data and expands correctly
Trip Page (ItineraryWidget)	TransportCard Widget displays data and expands correctly
Trip Page (ItineraryWidget)	AttractionCard Widget displays data and expands correctly
Trip Page (TripInfoWidget)	Renders correctly (mobile)
Trip Page (TripInfoWidget)	Renders correctly (tablet)
Trip Page (TripInfoWidget)	Should NOT show calendar button when isMyTrip is false
Trip Page (TripInfoWidget)	Should display correct status for past trips
Trip Page (TripInfoWidget)	Should display correct status for future trips
Trip Page (TripInfoWidget)	Should display correct status for ongoing trips
Trip Page (TripInfoWidget)	Tap on "Open map" opens MapPage
Trip Page (TripExpenseWidget)	Should show loading indicator while data loads
Trip Page (TripExpenseWidget)	Should render an error message if no trip is returned
Trip Page (TripExpenseWidget)	Displays expense data correctly
Trip Page (TripExpenseWidget)	Should handles null expenses
Trip Page (TripExpenseWidget)	Should show a currency dropdown
Trip Page (TripExpenseWidget)	Should render large expenses with correctly formatted sum
Trip Page (TripExpenseWidget)	Should render expenses ¿= 1M with compact format
Trip Page (TripExpenseWidget)	Should toggle back and forth correctly between EUR and USD
Map Page	Calculates center of cities correctly for initial camera position
Map Page	Map display marker for each city in the trip
Map Page	OpenDetails shows correct city info in dialog
Medals Page	Should show user not found when user is null
Medals Page	Should show loading indicator while data is loading

Page Tested	**Test Description**
Medals Page	Should display correct medal states for achievements
Medals Page	Should display correct level and countries visited
Upsert Trip Page	Tapping country field opens CountryPickerWidget
Upsert Trip Page	Tapping city field opens PlacesSearchWidget
Upsert Trip Page (Insert)	Submits form data
Upsert Trip Page (Insert)	Display selected countries as chips
Upsert Trip Page (Insert)	Selecting date field open DateRangePickerDialog
Upsert Trip Page (update)	Correctly initialize all data
Upsert Trip Page (update)	Submits update form data
Upsert Trip Page (update)	Display selected cities as chips
Upsert Trip Page (update)	Should not allow to add a city twice
Travel Stats Page	Loading State shows CircularProgressIndicator
Travel Stats Page	Should displays correct data and UI elements on success
Create/Edit Activity Page	Should render all form fields
Create/Edit Activity Page	Should show validation errors if required fields are empty
Create/Edit Activity Page	Should prefill fields when editing existing activity
Create/Edit Activity Page	Should update currency dropdown and enter cost
Create/Edit Activity Page (Flight)	Should not validate cost field with negative values
Create/Edit Activity Page (Flight)	Should create flight with correct data when form is submitted
Create/Edit Activity Page (Flight)	Tapping on start date field opens date picker and selects date
Create/Edit Activity Page (Flight)	Tapping on departure time field opens time picker and selects time
Create/Edit Activity Page (Accomodation)	Should update currency dropdown and enter cost
Create/Edit Activity Page (Accomodation)	Should create accommodation with correct data when form is submitted
Create/Edit Activity Page (Accomodation)	Should open Places Search Widget when location field is tapped
Create/Edit Activity Page (Accomodation)	Tapping on check-in and check-out time field opens time picker and selects time

Page Tested	Test Description
Create/Edit Activity Page (Accomodation)	Should handles currency conversion
Create/Edit Activity Page (Attraction)	Should update activity type dropdown
Create/Edit Activity Page (Attraction)	Should open PlacesSearchWidget when location field is tapped
Create/Edit Activity Page (Attraction)	Should update attraction with correct data when form is submitted
Create/Edit Activity Page (Attraction)	Should show snackbar when validation fails
Create/Edit Activity Page (Transport)	Should allow selecting the departure date
Create/Edit Activity Page (Transport)	Should allow selecting the duration
Create/Edit Activity Page (Transport)	Should display all transport types when tapping the type dropdown

4.4 Integration Tests

This phase extends the previous widget testing. These tests are performed in the deployment environment, allowing us to evaluate the system's stability when interacting with external APIs and ensuring a satisfactory level of reliability from this perspective. Given the nature of these tests, we focused on verifying the primary user flows within the application, ensuring that all core actions a user can perform, from the initial login/registration page onward, work correctly and smoothly.

The resulting tests are organized as follows:

Authentication

- Sign in with existing credentials.
- Log in with wrong credentials.
- Log in with correct credentials and go to Home Page
- Log out.

Explorer Page

- The Explorer page displays public trips, and user interactions behave as expected.
- Order public trips in Explorer Page according to date or popularity

Profile

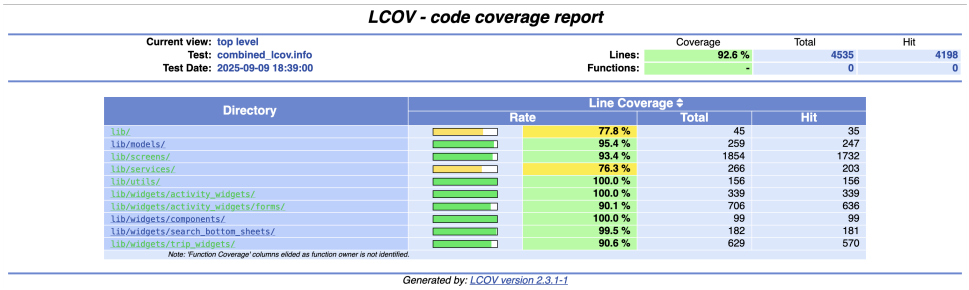
- Change profile data.
- Change color theme from dark to light and vice versa.
- User open Travel Stats page

Trip related

- Create a trip.
- Add new flight to a trip.
- Add new accommodation to a trip.
- Edit activity details.
- Edit trip details.
- Remove activity from a trip.
- Open map for a trip
- Check expenses and change currency.
- Change trip privacy setting.

4.5 Coverage Analysis

Coverage data were generated using Flutter testing tools, and a HTML report was produced for improved readability. The report provides, for each source file, the percentage of covered lines as well as the specific lines that were not executed during testing.



LCOV - code coverage report

Current view: top level
Test: combined_lcov.info
Test Date: 2025-09-09 18:39:00

	Coverage	Total	Hit
Lines:	92.6 %	4535	4198
Functions:	-	0	0

Directory	Line Coverage ‡ Rate		Total	Hit
lib/		77.8 %	45	35
lib/models/		95.4 %	259	247
lib/screens/		93.4 %	1854	1732
lib/services/		76.3 %	266	203
lib/utils/		100.0 %	156	156
lib/widgets/activity_widgets/		100.0 %	339	339
lib/widgets/activity_widgets/forms/		90.1 %	706	636
lib/widgets/components/		100.0 %	99	99
lib/widgets/search_bottom_sheets/		99.5 %	182	181
lib/widgets/trip_widgets/		90.6 %	629	570

Note: 'Function Coverage' columns elided as function owner is not identified.

Generated by: LCOV version 2.3.1-1

Figure 4.1: Test coverage

The testing process achieved 92.6% coverage across 4,535 lines of code over a set of 197 widget/unit tests and 24 integration tests, demonstrating a robust strategy that supports both code quality and application reliability.

Most application modules reached excellent coverage levels, with several components—including activity widgets and reusable UI elements—achieving 100% coverage.

Two modules, however, recorded coverage below 90%. This is because these modules correspond to controller code managing external services, which include many lines of exception handling. Such error scenarios are difficult to reproduce in a test environment, making it challenging to achieve higher coverage.