Milan Locker

Design and Implementation of Mobile Applications

Design Document

A.Y. 2022/2023

Riccardo Gianni - 10868744
Andrea Alberto Bertinotti - 10616719

POLITECNICO
MILANO 1863

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to provide an exhausting explanation of the design decisions we have made during the development of our application. In particular, in the document, we will provide an overview of all the aspects concerning the development of a cross-platform application, including its architecture and all its pages' interface design. In addition to this, we will focus on the implementation process, the integration of the external components, and how the whole system has been tested.

1.2 Scope

Milan Locker is a cross-platform application developed to simplify the life of travelers, workers, and all the people who are tired of going around Milan bringing bags and packages. The idea behind the application is to create a system of lockers all around the city of Milan where clients can safely store their bags and possessions for a limited time at low prices. The purpose of the application, in this sense is to offer to the client an intuitive and fast way to book one or more slots of the lockers, and to manage and keep track of all their reservations from a single place. In this way, all the people who need to store their personal belongings can download the application on their favourite device (that could be either a smartphone, a tablet or a laptop), create an account and easily start creating and managing locker reservations.

1.3 Features description

The following sections describe all the functionalities we implemented in the application.

1.3.1 Lockers navigation

It is the first page that opens up after the application has been successfully loaded, and it is the only one that can be seen and used without being logged-in to the system. It consists of a map of Milan, displaying a marker for each of the lockers available around the city and that can be booked by the user. If needed, the user can press on the markers to display some details about the selected locker, such as price or address.

1.3.2 Login and logout

The login page allows users to enter their email and password combination in order to be authenticated by the system and "unlock" all the pages of the application, that could not be accessed without a valid account. At the same time, from inside the application pages, it is possible to log out and restore the native state of the system, with all the functionalities apart from the lockers navigation one being "locked".

1.3.3 Sign up

The sign-up page allows to provide a new combination of email and password in the authentication database and allows users who still don’t have an account to create one and start using all the functionalities offered by the application.

1.3.4 Profile completion and update

After the user has created a new account, he can use all functionalities of the application, but his profile is not completed yet. For this reason, the user is asked to insert his contact information, such as name, phone number and address (i.e., to complete his profile) when he opens up the profile page for the first time. In this way, the account will be associated with a person who has all the contact information needed to use the system. After the profile has been completed, the user can edit all its fields whenever he wants through the edit profile page. In this way, if some contact information changes, they can easily be updated both in the application and in the database.

1.3.5 Booking creation and deletion

This feature allows the user to create a new reservation and book a locker cell for a limited period of time. In this way, the user can specify all the parameters needed for the booking creation, such as drop-off date and time, duration, and dimension of the needed cell, and save the reservation in the corresponding database. Once the booking has been saved, the user can visualize all the details on the corresponding page and, if needed, delete the booking and remove it from the database, making the locker available again in the previously booked time slot.

1.3.6 View active and past bookings

These two pages allow the users to see all their locker reservations. In particular, on the first page, they can see all the upcoming or ongoing bookings that haven't expired yet with all the corresponding details, while on the reservation history page, there are all the bookings the user has made, including the expired ones. We decided to make this distinction to separate the active bookings from the expired ones and allow the user to the reservations that are still active in a clearer way.

1.3.7 Reporting problems

This feature allows the user to let the developers know if there are problems with the application. In particular, it allows to send a description of the problem to a database that can be seen by the developers. In this way, every time a new problem is reported, the owners of the application can understand what's wrong with the application and fix it.

2 Architectural Design

2.1 Overview

Milan Lockers is developed with Flutter, which is an open-source framework by Google for building natively compiled multi-platform applications from a single codebase. Flutter is powered by Dart, a programming language designed for client development optimized for fast apps on any platform. The application relies on some external services, such as *Firebase Authentication*, *Google Maps Platform*, and *Cloud Firestore* for the database part.

Figure 2.1: Architecture overview

2.2 External services

2.2.1 Firebase Authentication

When the application launches, the locker map is accessible without an account. However, to perform any other tasks, it is necessary to have an account and log in. *Firebase Authentication* is used by the application, which offers an easy-to-use back-end service to manage user authentication. The chosen login mode is the classic email and password. Users must create an account linked with their email address and set a password to secure access. When a user registers for the first time, a new user account is created and connected to a unique identifier that identifies the user across the app.

2.2.2 Google Maps Platform

A crucial aspect of the user experience in using the app is the ease with which the location of the nearest locker can be identified. One user-friendly way in which this can be achieved is through an interactive map showing the geographic location of lockers scattered throughout the city. The service to which this is entrusted is *Google Maps Platform*, which provides a reliable tool for displaying customized maps based on Google Maps data. Through this service, it is possible to clearly locate lockers with custom markers so that they are immediately visible to the user.

2.3 Data management

2.3.1 Cloud Firestore

Cloud Firestore performs all functions related to data storage and retrieval. Cloud Firestore is Firebase's newest cloud database for mobile app development. Firestore leverages the NoSQL document-based data model, which stores data in documents. The documents are grouped in collections, which act as containers used to organize data and create queries. Unlike a SQL database, there are no tables or rows. Each document, in turn, contains a set of fields associated with a value. Our data is organized in three different main collections, whose structure is described below.

Lockers is the collection that contains documents related to the lockers that you can select when making a reservation. Each document contains information regarding the name of the locker, its address (in text format), its geographic location, and the hourly fare of its cells.
Each document is associated with a subcollection of documents each corresponding to a specific locker cell. Each cell is associated with a name (containing information regarding the size of that cell), and its hourly fare. In turn, each cell is associated with a subcollection of documents representing the hourly slots for which the cell has already been booked. Each reservation slot document contains information about the relative reservation, the user who made the reservation, the locker, and the cell booked.

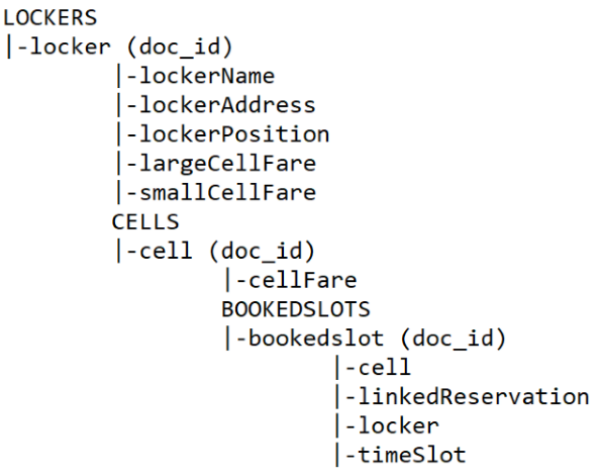

Figure 2.2: Lockers collection structure

Users is the collection of documents that contain information regarding users registered to the service. Each document is identified by the unique user identifier that distinguishes each user, and contains the email and, if he has decided to provide them, his first name, last name, address, and phone number.
Each user corresponds to a subcollection of reservations, which contains documents concerning the reservations made by the user. Each reservation document contains information regarding the locker and its address, cell, duration in hours, the start and end date of the reservation, and its price.

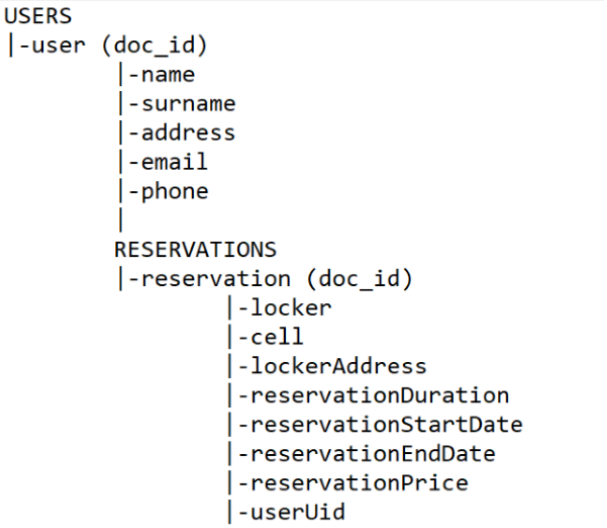

Figure 2.3: Users collection structure

Reports is the collection in charge of collecting all reports submitted by users. Each report corresponds to a document that contains the unique identifier of the user who made the report, the date and time at which it was sent, and the message body of the report.

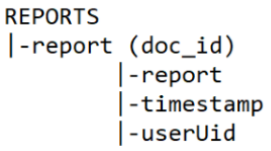

Figure 2.4: Reports collection strucure

2.4 Reference documents and websites

- Flutter documentation
- Firebase documentation
- Cloud Firestore documentation
- Google Maps Platform documentation

3 User Interface Design

3.1 UX diagrams

In the diagrams, the actions to go back are often omitted to simplify the diagrams. We chose to make a different UI for tablets (and big-screened devices in general) and smartphones in order to make the most of all the space the device screen can offer, but in general, the UX is the same for the two.

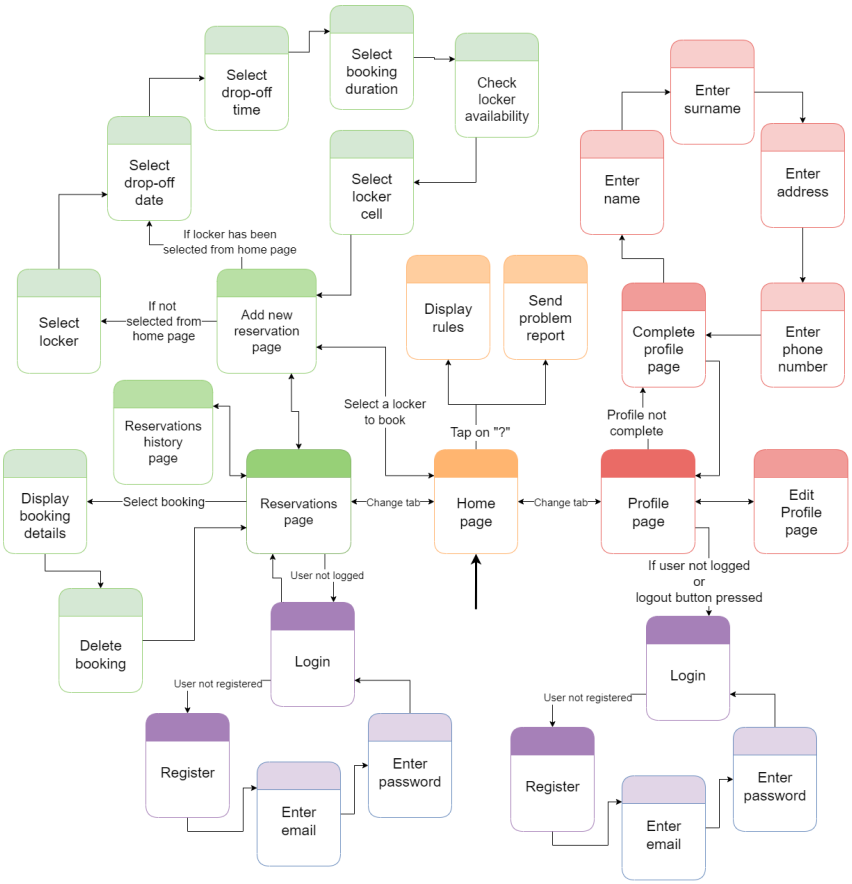

Figure 3.1: UX app diagram

3.2 User interface

The aim of this section is to show the design of the main screens of our application, describing the purpose of each visual component, and the functionalities implemented in the pages. As we said before, for some pages of the application we decided to create two different versions in order to exploit as much as possible the available space on the device screen. In both the layouts we decided to implement a navigation through a bottom bar in order to make it easier to move between the various pages and to make our app as intuitive as possible.

3.2.1 Splash screen

It is the first screen that is visualized when the application is started. It simply displays the logo of the application on a white background and its purpose is to welcome the user while the background processes restore the application state and load data from the database. For this page, we decided not to change the disposition of the elements since it is not a relevant page and there is only one component inside of it.

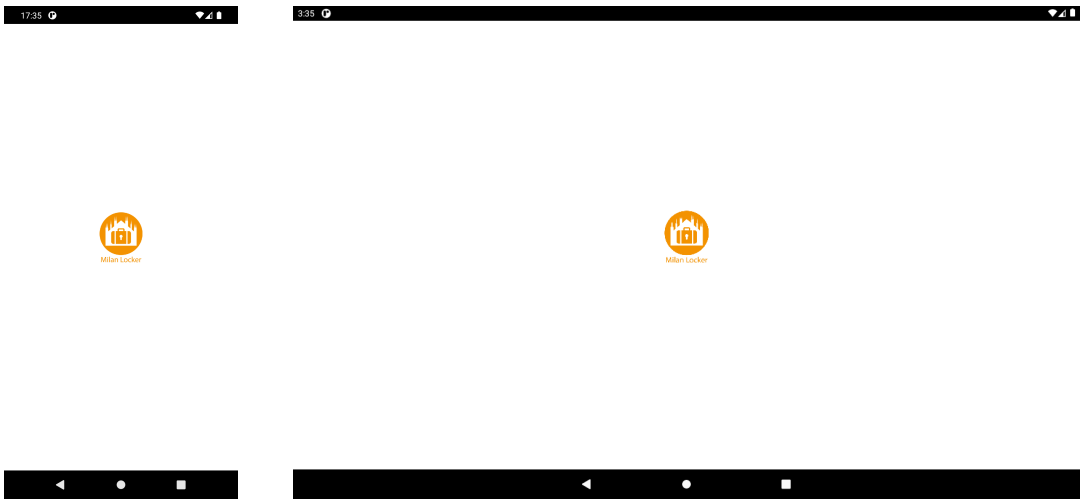

Figure 3.2: Splash screen UIs

3.2.2 Home page

As said before, the home page is the only one that can be used without having an active account. It contains the map of the city of Milan and allows the user to see and select each locker present all around the city (displayed through custom markers) open its details and create a new reservation for it by opening the relative page through a button. In addition to this, from the dedicated button on the app bar, it is possible to open the information page. On this page, we decided to differentiate the UI between the smartphone and tablet by changing the detail box of each locker. In particular, for the smartphone version, we created a small box placed in the bottom part of the screen containing only the details of the locker and the button to create a reservation on it. On the other hand, for the tablet version, we decided to create a bigger dialog box placed in the middle of the page and containing, in addition to all the information present in the mobile version, an image of the locker. We chose this layout since the available space on the smartphone screens is much less than the tablet ones.

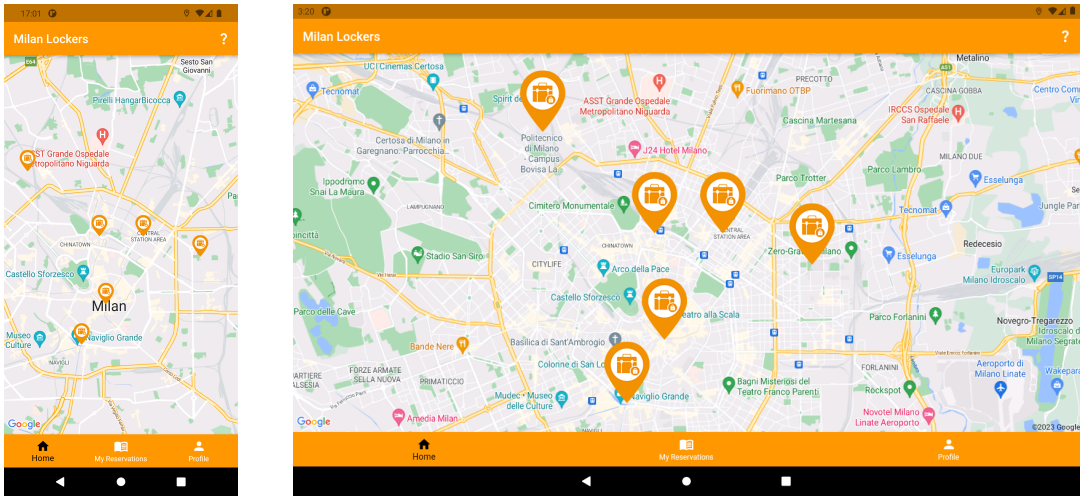

Figure 3.3: Home page UIs

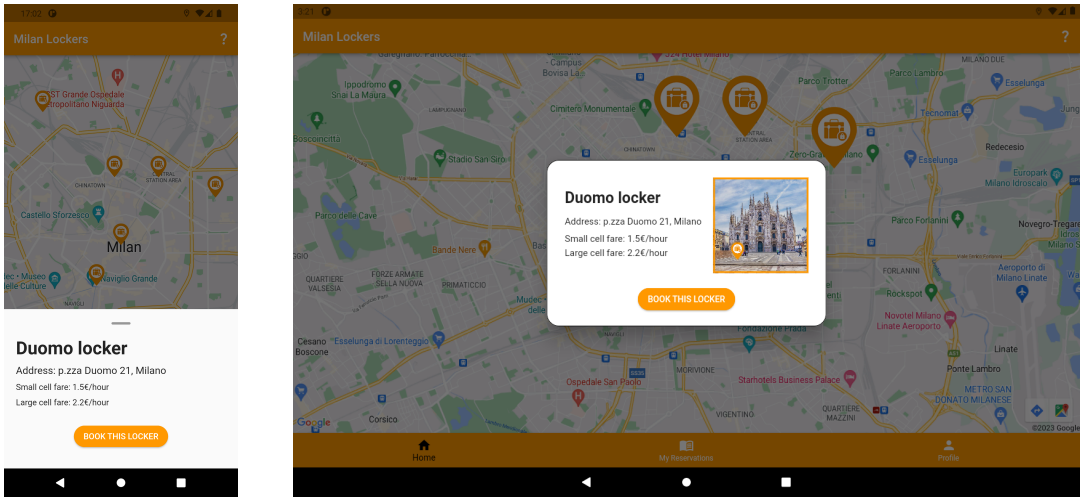

Figure 3.4: Home page UIs

3.2.3 Login and register screens

The login and register screens, which are quite similar to each other, allow the user to insert his combination of email and password (if he already has one), and to create a new account by inserting a new pair of email and password respectively. For this reason, the visual elements present on the pages are very few: two text fields for email and password respectively, the app logo on the upper part of the page, a button to complete the process, and a link to switch between login and register page.

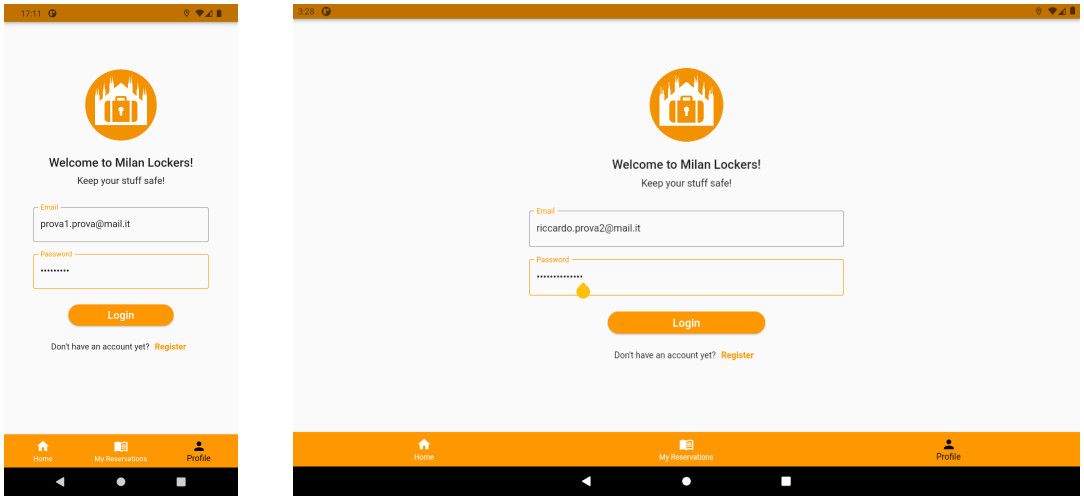

Figure 3.5: Login page UIs

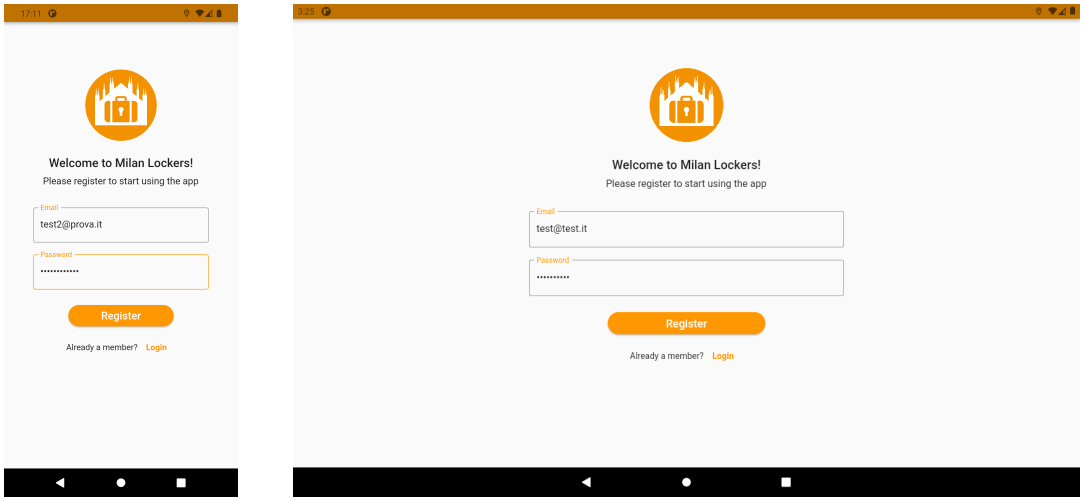

Figure 3.6: Register page UIs

3.2.4 Information page

This page, which is displayed only if the user is logged in to the application system (if not, the login page is displayed), allows one to read the rules of the locker booking system and to report any problem with either the lockers or the application. It is accessible from the home page through the "?" button in the app bar and it has two different layouts for smartphones and tablets. On the smartphone page the elements are displayed in a column view: on the topmost part of the screen are visualized the system rules and terms and conditions, while on the bottom part there is a button that opens a dialog box that allows to send a description of what's wrong with the lockers or the application. The dialog box automatically closes on report submission and a snackbar is displayed to inform the user of the successful report submission.

On the tablet version, the screen is divided into two sections: one for the rules and the other for the report submission. For what concerns the "report a problem" section, differently from the smartphone layout, the text box to be filled is immediately displayed on the page, and no dialog box is used.

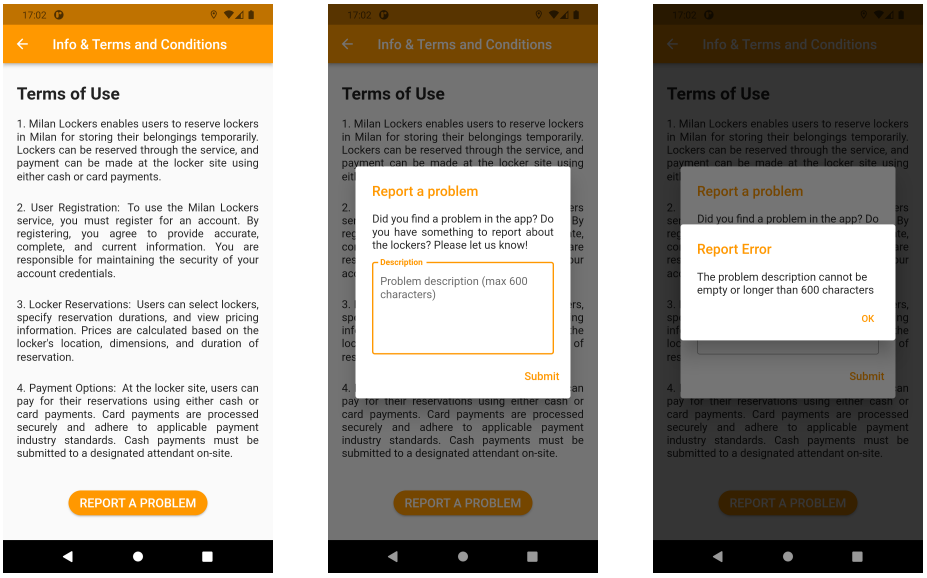

Figure 3.7: Info screen mobile UI

Figure 3.8: Info screen tablet UI

3.2.5 Reservations pages

The reservation pages are a group of screens that contain all the information and the functions needed to book the lockers and visualize their details and will be described in the following sections.

Reservations list page

This page, which can be considered the main one since it is the first one that displays when the user moves to the reservations tab of the navigation bar, displays all the active reservations of the logged user with all their details (including the drop-off and pick-up time and the price). In addition to this, there is a button that allows the user to delete the selected reservation and make the locker available again in the previously booked time slots. The page has two different designs for smartphones and tablets to exploit as much as possible all the available screen space. For what concerns the smartphones, the reservations are listed through expansible tiles that, while compressed show only the name of the locker, the drop-off date, and the locker address. The deletion button and the complete details of each reservation (which include the drop-off and pick-up dates and times, the duration, the booking price, the number and dimension of the booked locker cell, and a significant image) are shown only when the tile is expanded by tapping on it.

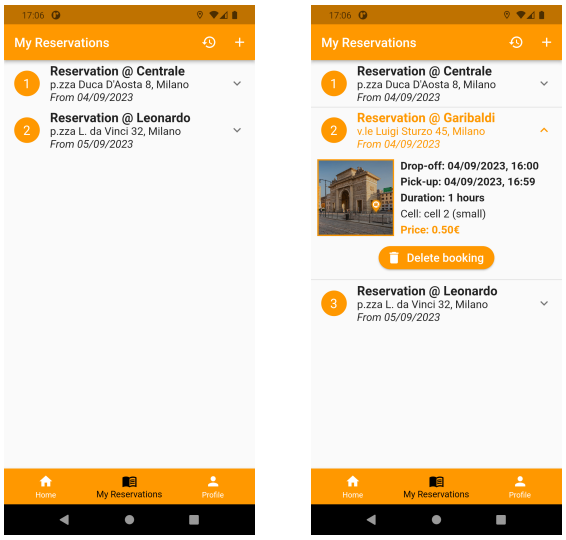

Figure 3.9: Reservation list page mobile UI

On the other hand, for the tablet page, the screen is divided into two sections: on the left, there is the list of all the active bookings (which are not expansible) with the name of the locker and the drop-off date only displayed on them, while on the right side are displayed the deletion button and all the details of the selected reservation, including a bigger image with respect to the smartphone version.

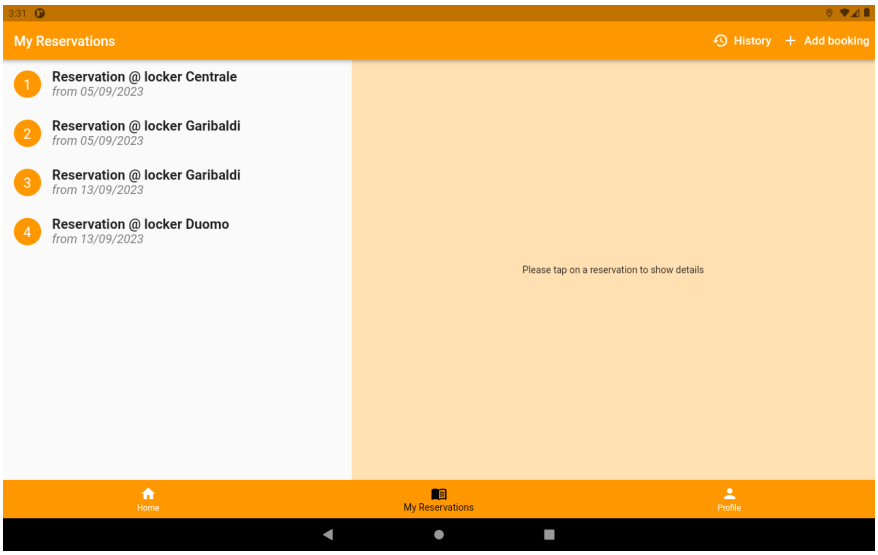

Figure 3.10: Reservation list page tablet UI

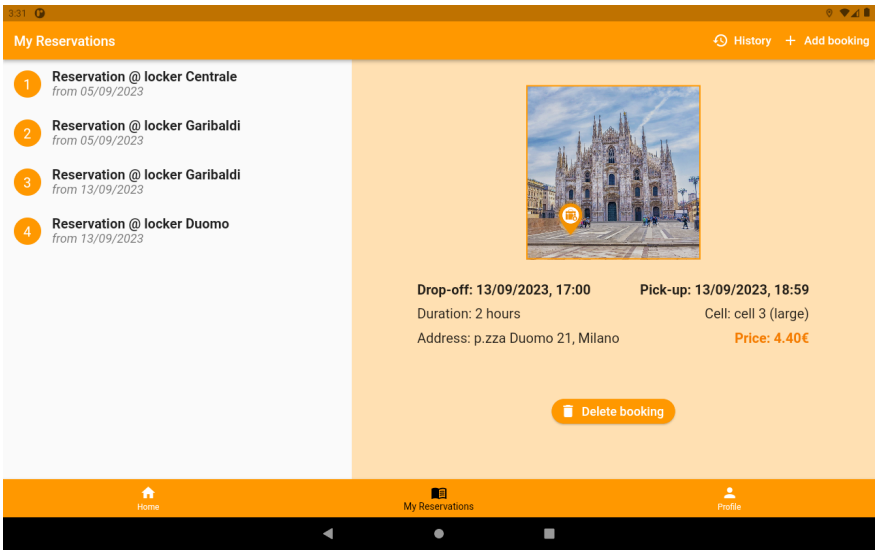

Figure 3.11: Reservation list page tablet UI

Reservations history page

On this page, which presents no difference between the smartphone and tablet layouts, are listed all the reservations the user has ever made (including the expired ones, that are not shown in the reservation list). This allows the user to keep track of all the bookings he has made up to that moment. The reservations are displayed through a list and each of them is placed on a tile containing the name of the booked locker and the drop-off and pick-up date and time. Since this is a simple consultation page where no particular action can be performed, we decided not to differentiate between mobile and tablet versions.

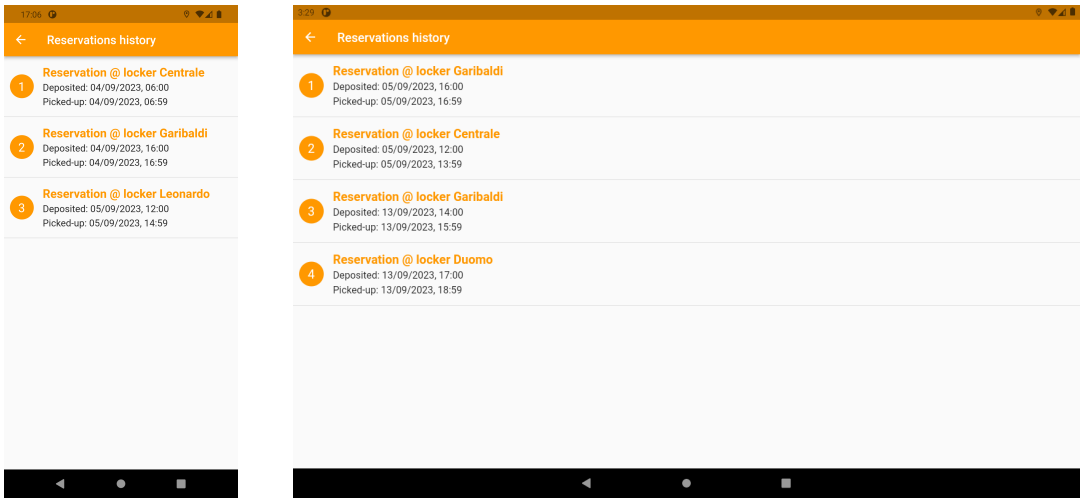

Figure 3.12: Reservations history page UIs

Add new reservation page

On this page, which can be accessed both from the reservations list page and from each locker marker in the home page map (and presents a little difference between the two paths of access), are displayed all the fields and the information that are needed to make a new reservation. In particular, the fields that are present on the page are the following:

- Drop-down menu for locker selection (if not already selected from the map).
- Calendar view for drop-off date selection.
- Drop-down menu for drop-off time selection.

- Drop-down menu for reservation duration.
- Button for availability check
- Drop-down menu for locker cell selection among the available ones.

In addition to this, the page presents a significant image for the selected locker (if no locker is selected, the app logo is displayed), the selected locker address, and the automatically calculated booking price. On the bottom part of the page, there are the confirmation button and a help button that displays a tooltip message containing some useful information for the user. Depending on the fact the page is opened from the locker map (by tapping on a locker marker) or the reservation list page, the drop-down menu for locker selection is enabled or not. In particular, if the user selects a marker on the map and wants to book that specific locker, the locker field of the page is disabled and the corresponding locker is automatically selected, while if the user opens up the page from the reservations list, all the lockers are available through the drop-down menu. Since the number of elements on the page is large, we decided to differentiate the layout in smartphones and tablets. For what concerns the smartphones, the image is placed on top of the screen and its size is quite small in order to fit everything on the screen. Then all the fields are displayed on the middle part of the screen, and in the bottom part are placed the address, the booking price, and the two buttons. On the other hand, in the tablet layout the screen is divided into two "zones": on the left one are placed all the fields and the two buttons, while on the right one, there are the locker address, the price of the reservation, and a bigger image with respect to the mobile phones layout.

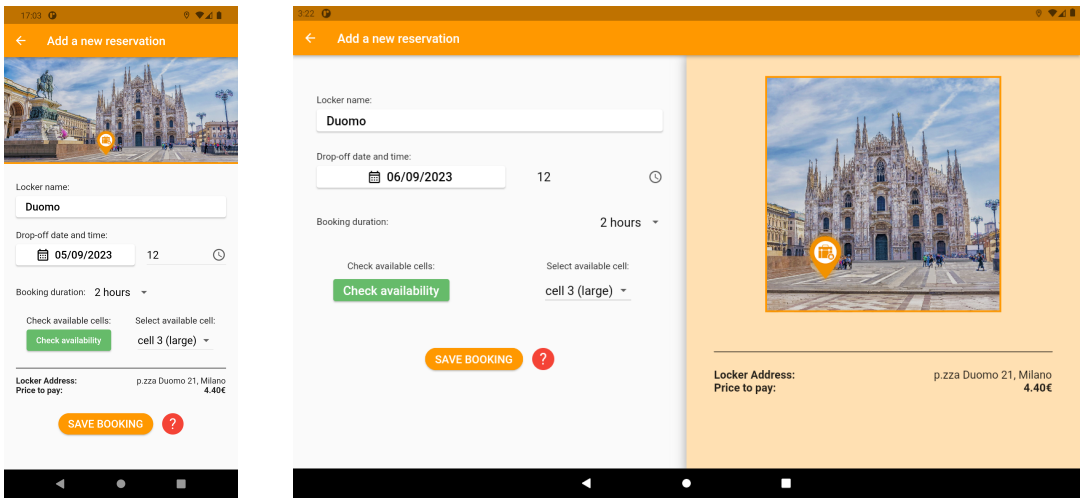

Figure 3.13: Add new reservation page UIs (from home page)

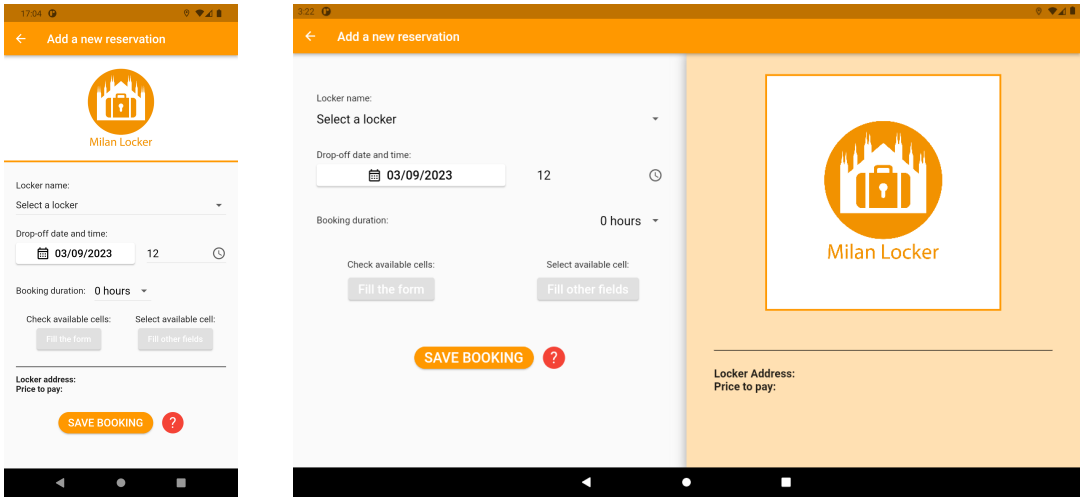

Figure 3.14: Add new reservation page UIs (from reservations page)

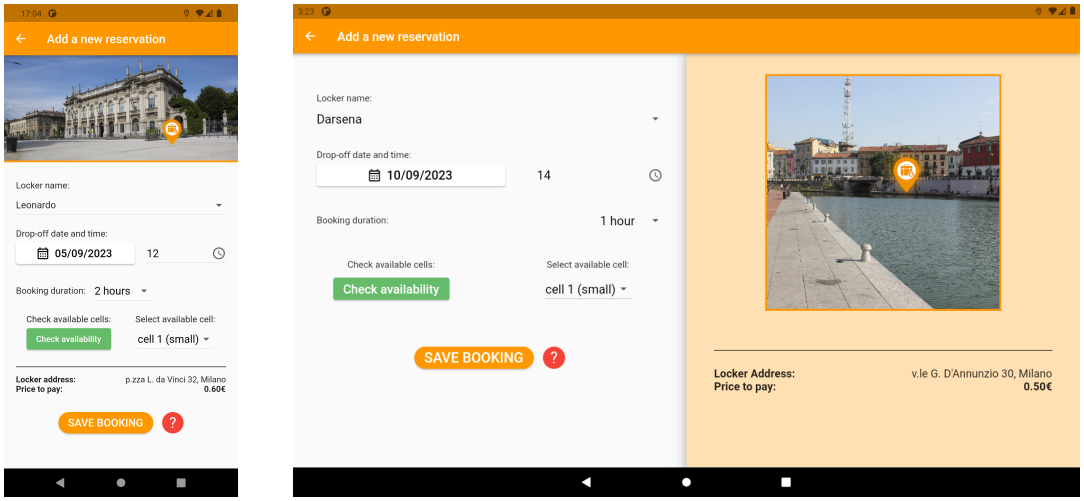

Figure 3.15: Add new reservation page UIs (from reservations page)

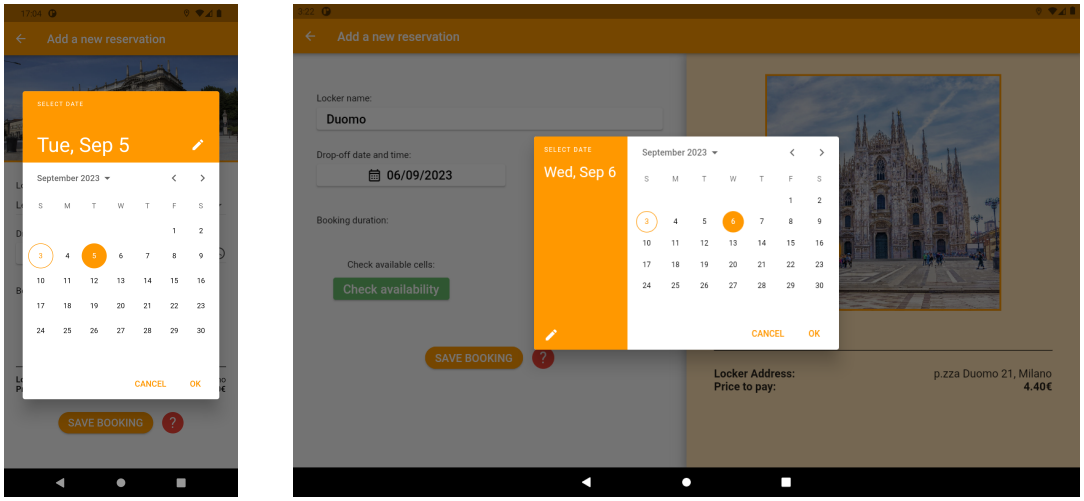

Figure 3.16: Add new reservation page UIs (from reservations page)

3.2.6 Profile-related pages

The profile pages described in the following sections are a group of screens that contain all the user information and the functions needed to edit it.

Profile page

On this page, which has no different layout for mobile phones and tablets, are displayed the contact information of the logged-in user. In particular, in the upper part of the screen is displayed the user's full name and an image with the user's name and surname initials. In the central part of the screen, the contact information is displayed through some boxes and includes the user's full name, the account email, the phone number, and the address. Finally, in the bottom part of the page, there is a button that allows the user to open the edit profile page and a floating action button that manages the logout from the application. As a remark, the whole page is not displayed if the user is not logged in, but the login page is displayed instead. In the same way, if the user account has just been created and the profile has not been completed yet, the profile page is substituted with the profile completion page.

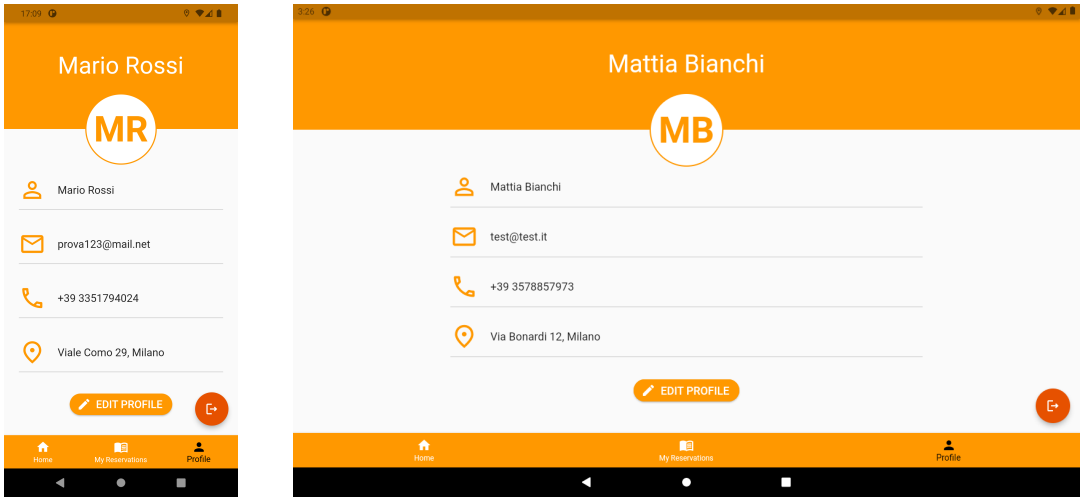

Figure 3.17: Profile page UIs

Edit profile page

This page allows the user to edit his contact information through some text fields. All the contact information but the email can be modified by the user and saved through the dedicated button placed on the bottom part of the page. All the fields must respect some criteria of length and format and, in case of errors, a dialog box is displayed to inform the user.

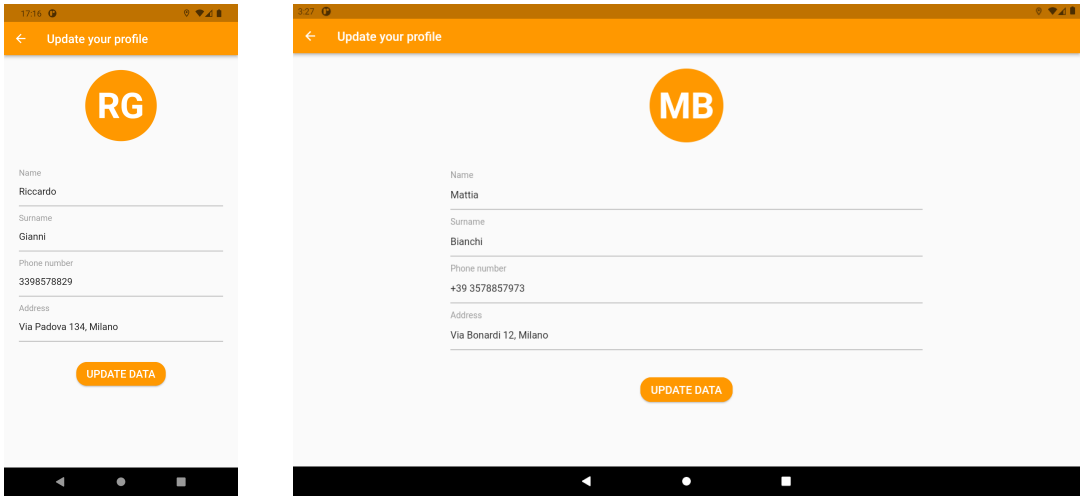

Figure 3.18: Edit profile page UIs

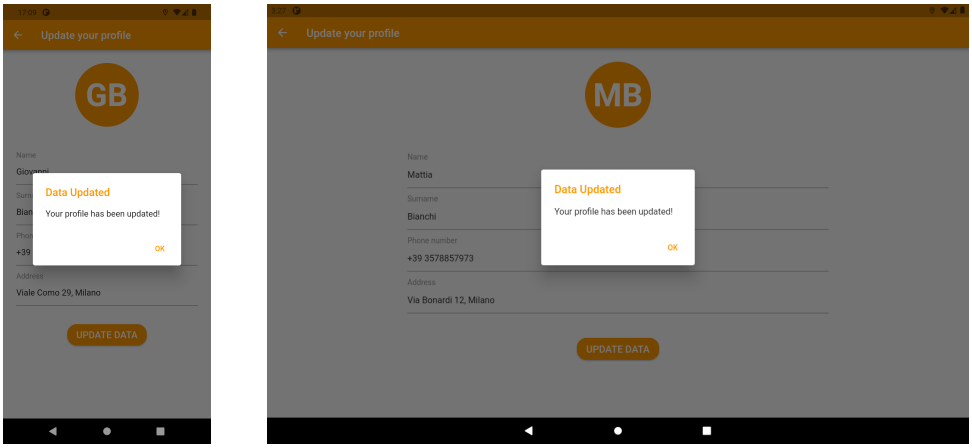

Figure 3.19: Edit profile page UIs

Profile completion page

This page, which is similar to the edit profile one with some small adjustments, allows the user to complete his profile after the creation of a new account. It is automatically displayed as soon as the user opens the profile page after the registration to the system, and contains the same fields of the edit profile page: the name and surname, the phone number, and the address, which are editable through the dedicated text fields. The profile completion process can be confirmed through the button placed in the bottom part of the page.

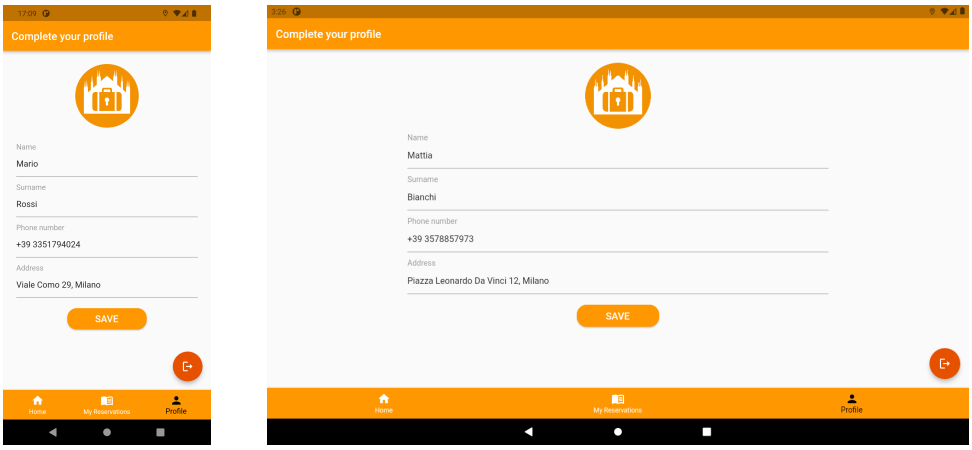

Figure 3.20: Complete profile page UIs

4 Implementation, Integration and test Plan

4.1 Implementation

4.1.1 Framework and programming language

Our goal for this project was to create a client application that can be distributed to different user devices, such as smartphones, tablets, or computers, with different operating systems, including iOS, Android, and Windows, in order to provide a service that can be as much available as possible. In order to achieve this goal, we decided to use Google's Flutter mobile app framework, whose architecture is shown in the figure below. All Flutter apps are written in Dart language and make use of many of the language's more advanced features. The main components included in the Flutter framework are: Dart platform, Flutter engine, foundation libraries, design-specific widgets, and Flutter development tools (DevTools) Every UI element in Flutter is defined as a "widget", which in turn can be formed by other basic widgets, and describes the logic, interaction, and design of the UI element itself.

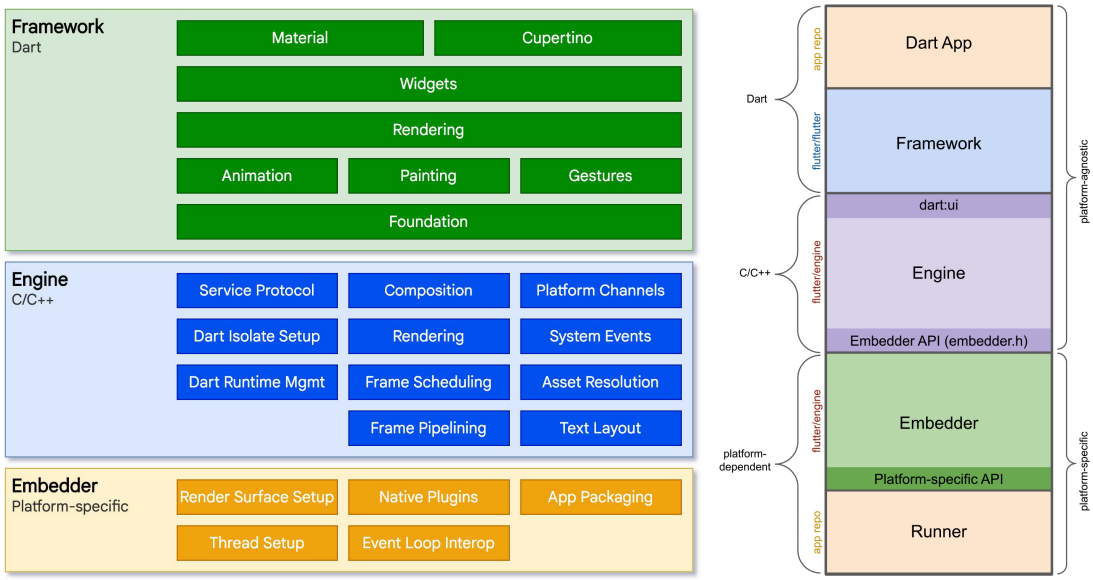

Figure 4.1: Flutter architecture

4.1.2 Support for tablets

As said in previous chapters, we wanted to introduce different user interface layouts for tablets in order to exploit all the available screen space. Thus, we adapted the organization of contents to wider screens using Flutter's MediaQuery, which permits specifying conditions relying on screen size.

4.2 Integration

4.2.1 Libraries

In order to add new functionalities in our app and optimize the already present ones we used some external libraries provided for the Flutter framework, which will be described below:

- **flutter_native_splash**: used to customize Flutter's default white native splash screen with background color and splash image.
- **provider**: used to create a wrapper around InheritedWidget to make them easier to use and more reusable.
- **intl**: used to deal with date formatting and parsing.

- **google_maps_flutter**: used as default Flutter plugin for integrating Google Maps in iOS and Android applications.
- **cloud_firestore**: used to import all Cloud Firestore functionalities inside the application code.
- **firebase_core**: Flutter plugin for Firebase Core, enabling connecting to multiple Firebase apps.
- **location**: cross-platform plugin used to easy access to device's location in real-time.

5 Testing

5.1 Test plan

In order to test all the functionalities we implemented on our application and check if everything was correctly developed, we decided to follow two strategies: manual analysis with Flutter development tools and coded unit and widget tests. In particular, we decided to follow these two approaches since we knew the unit tests could not cover all the functionalities and aspects of the code, so extra manual tests would have needed to check if everything was correct. For what concerns the unit and widget testing, our aim was to reach a code coverage of at least 70%, in order to have most of the code automatically checked, and the rest of the code manually checked with the available Flutter tools.

5.2 Unit Tests

Flutter unit tests are supposed to run in the mock environment due to flutter constraints. Our app heavily relies on external services, since the lockers shown on the map exploit Google Maps API, while all the server part is handled through Firebase Firestore. Because of the employment of these external services, the possibility of running several unit tests is limited.

Thus, we targeted our tests to check the correct performance of the text fields in the forms and verify correct error handling. In particular, we have tested the login and register text fields, checking that the email is well formatted and the password is not too short. In addition, we also tested the function that is in charge of computing the time slots occupied by each reservation, making sure it returned the correct slots according to the provided reservation hour and duration.

5.3 Widget Tests

The goal of this type of test is to verify that the UI of a widget is correctly displayed and that interaction with the user happens as expected. Testing a widget involves multiple classes and requires a test environment that provides the appropriate widget lifecycle context. Since our application heavily relies on *Firebase Firestore*, we employed *fake_cloud_firestore* package to mock the back-end in order to provide fictitious documents to be displayed and used to test the widget.

Widget tests are deployed with a granularity based on the single app screen. For each screen, which is in turn constituted by multiple basic widgets, all the use cases are tested. Both mobile and tablet versions of each screen have been tested in the same way, with small adaptations in order to fit the relative design variations. The list of the most important tests in reported below:

Table 5.1: Screen Testing and Tests Performed

Screen Tested	**Test Performed**
New reservation (from list)	• Basic rendering check • Info tooltip works correctly • User tries to save booking without selecting a locker • User tries to save booking after selecting a locker but with no duration and cell provided • User tries to select available cells with no locker selected • User tries to check available cells with no locker selected • User tries to check available cells with no duration selected • User selects locker, time and duration and checks availability • User tries to make a reservation in the past (error is displayed) • User tries to make a reservation in the future (confirmation is displayed)

New reservation (form map)	• Basic rendering check • Info tooltip works correctly • User tries to save booking with no duration and cell provided • User tries to select available cells with no duration • User tries to check available cells with no locker selected • User tries to select cell with no duration selected • User selects date, time and duration and checks availability • User tries to make a reservation in the past (error is displayed) • User tries to make a reservation in the future (confirmation is displayed)
Reservation history	• Rendering with reservations in list • Rendering with empty list
Reservation list	• Rendering with reservations in list • Rendering with empty list • User taps on reservation to see details • User deletes reservation
User profile	• Basic rendering check

User profile edit	• Basic rendering check • Edit hints are correctly displayed • User provides too short name • User provides too short surname • User provides wrongly formatted phone number
Info screen	• Basic rendering check • User inserts too long report • User inserts report
Locker dialog box	• Basic rendering check

5.4 Coverage Analysis

After implementing all the tests, we performed a coverage analysis of the code, and the result is that on over 95 tests performed, the overall coverage on the lines of code is 76%, while, if we consider only the screens with which the user interacts, the level of coverage rises to 79.5%.

A manual analysis is done on the uncovered code to check that the lines not included in the tests are not relevant for this kind of test. The coverage data are generated by exploring flutter testing functionalities. Then, we generated a visual report in order to make it more immediate and readable. The report shows for each source file the line coverage percentage and the missing lines.

– All Files	2240/2948	76.0%
– lib	2240/2948	76.0%
– Services	8/71	11.3%
functions.dart	8/8	100.0%
auth_service.dart	0/12	0.0%
database_service.dart	0/51	0.0%
– Screen	2231/2807	79.5%
login_screen.dart	92/114	80.7%
register_screen.dart	82/100	82.0%
custom_dialog_box.dart	57/64	89.1%
– Reservations	1449/1883	77.0%
reservation_add_locker_tablet.dart	302/396	76.3%
reservation_add_locker_vertical.dart	249/340	73.2%
reservation_add_tablet.dart	327/425	76.9%
reservation_add_vertical.dart	260/353	73.7%
reservations_history.dart	54/55	98.2%
reservations_list_tablet.dart	82/112	73.2%
reservation_tile_horizontal.dart	77/77	100.0%
reservations_list_vertical.dart	48/75	64.0%
reservation_tile_vertical.dart	50/50	100.0%
info_screen_horizontal.dart	67/84	79.8%
info_screen_vertical.dart	84/104	80.8%
– User	400/458	87.3%
add_profile_screen.dart	142/162	87.7%
user_profile_screen.dart	114/126	90.5%
edit_profile_screen.dart	144/170	84.7%

Figure 5.1: Code Coverage

5.5 User Test

After the internal test phase was concluded, we tested the design of the app by releasing it to a small group of people who weren't involved in any stage of the development of the application. In the first phase, they were asked to perform a series of simple actions that were inspired by the test cases of each widget. Lastly, we let the users navigate freely in the application and collected their opinions in order to make graphic improvements.

6 Effort spent

Most of the app creation occurred from June to August 2023. Our team brainstormed in person at first to outline the project's particulars. Then, during development, we mainly collaborated remotely, keeping in daily contact and having regular video calls to update each other on advancements. Although we have not precisely measured the time devoted to the project, we believe that we averaged around 150 working hours each.
Although exact development time was not measured, it is estimated that roughly 150 hours of work per person were spent on the project.

Table 6.1: Working hour breakdown

Task	**Time spent**
Brainstorming and project outlining	5%
App development	75%
Testing	15%
Design document drafting	5%