POLITECNICO
MILANO 1863

UrbanHunt

Design Document

Authors:
Francesco Fulco Gonzales
Lorenzo Mainetti

Date: February 3, 2022

Version: 1.0

Contents

1. Introduction

1.1. Purpose

This document is the Design Document (DD) of the *UrbanHunt* mobile application. The main purpose of this document is to give a functional and non-functional description of the system, through a detailed analysis of its architecture, its components and their interaction. Mockups and Use Cases helps to provide a full description of the user interface and experience of the system.

This document is written to be read and used by:

- Developers and programmers that implemented the requirements and will develop new features
- Testers and project managers that have to evaluate the efficiency of the system
- Customers and users most interested in validation system goals and high-level functionalities.

1.2. Scope

Here we give a quick summary of the application's purpose.
UrbanHunt offers to their users the possibility to discover new interesting and little-known places in any city or location around the world within a gamification framework. Users may unlock new places or add new ones: indeed the places are collected via crowdsourcing, for which users get a reward.

1.2.1. Goals

To sum up, the main functionalities are:

G.1 **Authenticate**: users should be able to have a permanent account in the app to save their progress and access the app's functionalities.

G.2 **Unlock places**: the system must let users discover new places, "unlocking" them, showing them relevant information about it and getting a reward.

G.3 **Contribute**: users should be able to contribute to the app by adding a new place, providing a picture, the description and its location. Users will get a reward upon adding a new place.

G.4 **Compete**: a leaderboard shows the ranking of the top scoring users in the world and in the user's country. It is designed to add a gamification aspect to the app and entice users to its use.

1.3. Definitions, Acronyms, Abbreviations

1.3.1. Definitions

- **Place:** The central entity of the app, defined by a position in the map, an image and a description, as well as some relevant category tags.
- **Contribute:** the addition of a new place to the app.

- **Categories:** set of tags that can be assigned to a place.

1.3.2. Acronyms

- **API:** Application Programming Interface
- **COTS:** Commercial off-the-shelf
- **DD:** Design Document
- **SDK:** Software Development Kit
- **UI:** User Interface through users require available functionalities.
- **UX:** User Experience, way in which the user interacts with the service.
- **VCS:** Version Control System

1.3.3. Abbreviations

- **A.i:** i-th Assumption
- **G.i**: i-th Goal
- **R.i:** i-th Requirement

1.4. Revision history

- Version 1.0, released on 03/02/2022

1.5. Document Structure

The sections of the Design Document (DD) are organized in the following way:

- **Chapter 1, Introduction**: presents the scope, purpose, and structure of the document and provides the definitions, acronyms and abbreviations used in it.
- **Chapter 2, Functionalities, Requirements and Constraints**: this chapter goes over the functionalities of the app and its high level components. Moreover it lays out the assumptions and requirements of the application, related to its goals. Finally it defines its constraints and non functional requirements.
- **Chapter 3, Architecture**: presents an in-depth description of the system's architecture. It defines the main technological components and the relationship between them. It is a useful chapter to understand the frontend and backend technologies that support the system.
- **Chapter 4, User Interface Design**: this section contains the sketches and screenshots for most views, along with a brief description of the functionality they provide.
- **Chapter 5, External Services and Libraries**: presents all external components that are used by the system, both by the backend (APIs) and the frontend mobile application (packages).
- **Chapter 6, Implementation, Integration and Test Plan**: it shows the implementation and integration plan of all the components and subcomponents. It also contains the testing strategy and final results.
- **Chapter 7, Improvements and additional features**: this chapter reports some additional ideas that might be implemented in the future.
- **Chapter 8, References**: in this last chapter are mentioned the references to documents or sites about technologies, patterns and architectures used in

this document.

2. Functionalities, Requirements and Constraints

2.1. Product Perspective

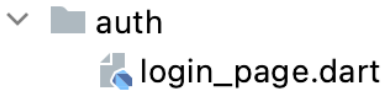
UrbanHunt aims to be the go-to app for anyone who wants to spend a weekend discovering their city or for any tourist looking for something different than the usual, well-known places of interest.
UrbanHunt focuses on providing a user-friendly interface, which is thought to entice users and make the experience as frictionless as possible, which is also reflected in its various signup methods, that any user preference:

- signup and login with email and password
- login with Facebook
- login with Google

Furthermore, *UrbanHunt* is already thinking about further improvements and functionalities which can be found in Section 7.

2.2. Product Functions

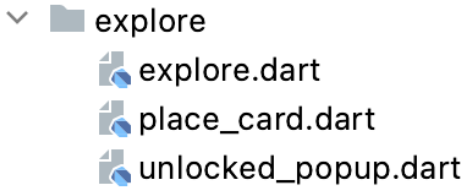
2.2.1. Authentication



The Login and Registration page allows the user to choose one of three methods to log into the application. If the user is already registered and he wants to log in with his email, a form is displayed in which he has to provide email and password. The user will automatically be logged in for his following sessions, without needing to authenticate again.
If the user does not remember his password can click on “Forgot your password” and he will immediately send to the reset password page in which he has to insert the email address in order to receive an email to reset the password.
If there is a new user who wants to register for the first time, he can tap on “Register” and he will be asked to fill a small form with just email and password. Alternatively, the user can choose to log using his Google or Facebook account.
The registration process is meant to be as lean and as fast as possible to give users a smoother experience. In fact some additional information such as the user’s country and username are automatically filled in and can be changed later from his profile settings. The country is inferred from his IP address using an apposite API, and the username is randomly generated. The picture is also

downloaded from Google or Facebook, or can be added later with email and password registration.

2.2.2. Explore



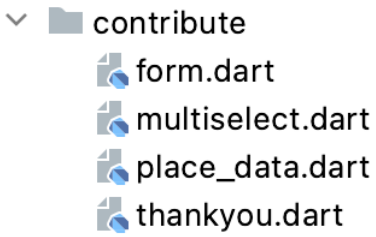
The explore page provides the core functionality of the app: letting the user visualize a map with all the places. The map is provided thanks to the Maps API. At a glance the user can see all the places in his surroundings and all over the world, represented by small locks, which appear open if the place has already been visited by the logged in user, closed if the user has not been in that place. A place can be unlocked only when the user is nearby that place.
Moreover the explore page shows the user location if the GPS is turned on, otherwise it uses the last recorded user location. The explore page also features a button to recenter the user in its position, which will trigger a popup in case the location is off, and a button to align the map to the north, that only shows up when the map is rotated.

2.2.3. Place

A place can be either locked or unlocked. On the basis of these two states different information is displayed. If the place is locked a very short intriguing description of the place is provided, and the image of the place appears blurred, to preserve a mystery around it, on the other hand if the place is unlocked there will be a full description of the place with some interesting notions about it, and a picture. Any place (locked and unlocked) displays the following elements:

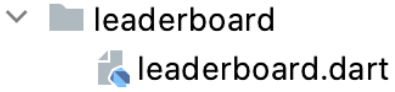
- Its distance from the user's location, or the last recorded location in case the user has GPS off;
- A button to open the location in google maps, useful to let the user plan his trip;
- Like/dislike buttons and the total like/dislike count. A user can only vote for places they have unlocked;
- The name and description (different depending on the place's state);
- The address
- Some tags that categorize the place, e.g. culture, nature, art etc.
- A picture of the place.

2.2.4. Contribute



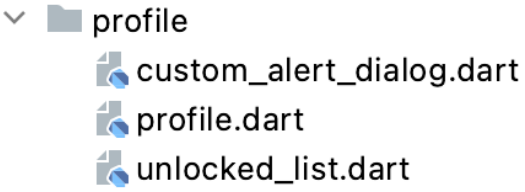
The contribute page lets a user add a new place, adding the above mentioned information. The form is clearly validated and displays relevant errors if the fields are incomplete or incorrectly formatted. To ease the user in filling the address the user can choose it by tapping on a map, or by writing the address in the search bar.

2.2.5. Leaderboard



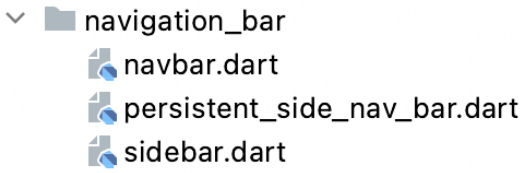
This page shows a ranking of the users, divided into two tabs, one global leaderboard that shows the best players in the world and a national leaderboard that depends on the user's selected country, which can be changed in the profile settings.

2.2.6. Profile



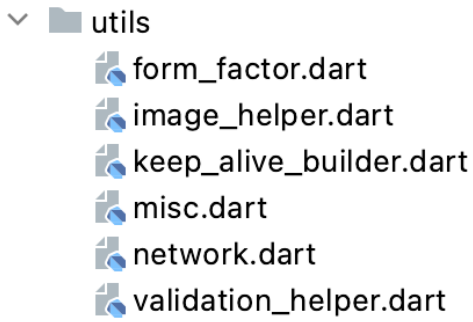
Here the user can edit his information, such as his username, email, password (if he logged with email and password) and country. He can also see the list of his unlocked places and for each display the place information. The user can also change his profile picture and log out of the account.

2.2.7. Navigation Bar



The navigation across the main pages is handled through a navigation bar: the user can navigate to Explore, Contribute, Leaderboard and Profile.
The navigation bar is provided in two customized designs based on the device orientation: a bottom navigation bar on portrait mode and a side navigation bar on landscape mode.

2.2.8. Utils



This directory contains all auxiliary and reusable functions, classes and widget that are used across the app. We decided to have it in a separate folder to increase decoupling and ease testing.

2.3. Assumptions and dependencies

Below we present the domain assumptions that must stand for the system to work as expected.

- **A.1** Users have either an email address, a Google account or a Facebook account with which they can register
- **A.2** Users can be identified by their email, which is unique
- **A.3** The external services used to provide users with our functionalities are reliable, supported by developers and updated regularly
- **A.4** The internet connection works properly without failures
- **A.5** Users accept to give us some basic personal information such as name and location
- **A.6** Users are willing to give us storage and location permissions on their phone
- **A.7** The users will contribute valuable content to the app and not spam
- **A.8** Most users live in an urban area or plan to use the app in a city
- **A.9** Users have a basic knowledge of the English language

2.4. Functional Requirements

In this section are outlined the various functional requirements of the application divided by goals.

- **G.1** **Authenticate:** users should be able to have a permanent account in the app to save their progress and access the app's functionalities.
  - **R.1** The system must provide a form to let users register.
  - **R.2** The system must provide a form to let users to log in.
  - **R.3** The system must notify a registering user if the inserted email is already assigned to a registered account
  - **R.4** The system must notify a signing in user if the inserted password is wrong
  - **R.5** The system must notify the user to insert a valid email if the pattern inserted isn't correct

- **R.6** The system must allow users to reset their password using their email
- **R.7** The system must securely store tokens used for authentication
- **R.8** The system must allow users to log out
- **R.9** The system must allow users to delete their account
- **R.10** Users must allow users to modify their profile information, such as name, username, password, city and profile image

G.2 **Unlock places:** the system must let users discover new places, "unlocking" them, showing them relevant information about it and getting a reward.

- **R.11** The system must display all available places in the map
- **R.12** The map must display the user's location
- **R.13** A user can unlock a place only if he is very close to it.
- **R.14** An unlocked place has a different description than a locked place, and it doesn't show its picture.
- **R.15** An unlocked place contains all relevant information, such as category, address, description and picture
- **R.16** A user can like or dislike a place only if it is unlocked
- **R.17** The system must display all relevant errors, when the user is prevented from performing an action (e.g. too far to unlock, cannot like locked place)
- **R.18** Users must get a reward upon unlocking the place
- **R.19** The system must provide an easy way to navigate to the place
- **R.20** The system must allow an easy way to share the location with friends to encourage collaboration
- **R.21** Users must be able to see their unlocked places

G.3 **Contribute:** users should be able to contribute to the app by adding a new place, providing a picture, the description and its location. Users will get a reward upon adding a new place.

- **R.22** The system must allow users to add any place in the world, with its name, locked and unlocked description, category, picture and location
- **R.23** The system will stop users from submitting incomplete places and notify them of the error
- **R.24** The users must be rewarded for adding a new place

G.4 **Compete:** a leaderboard shows the ranking of the top scoring users in the world and in the user's country. It is designed to add a gamification aspect to the app and entice users to its use.

- **R.25** The system must show the best users on a leaderboard, both globally and in each user's country
- **R.26** Users must be able to see their score

2.5. Performance

UrbanHunt can support a virtually unlimited number of users, since it relies on a serverless scalable backend, which costs proportionally to its usage. The NoSQL

database Firestore uses is indeed the fastest and most scalable architecture for a large number of transactions as it allows horizontal scaling.

In this section we provide a set of graphics obtained through the Flutter-DevTools that show the actual performance of the application client:

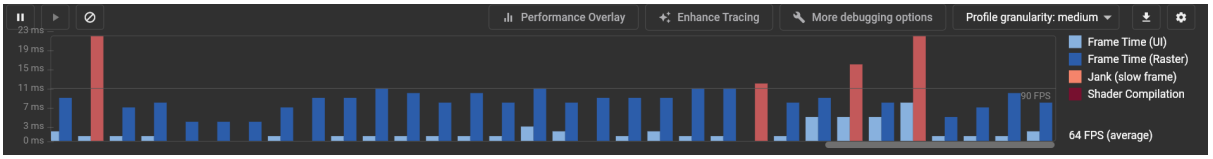


Figure 1: **Flutter frames chart**

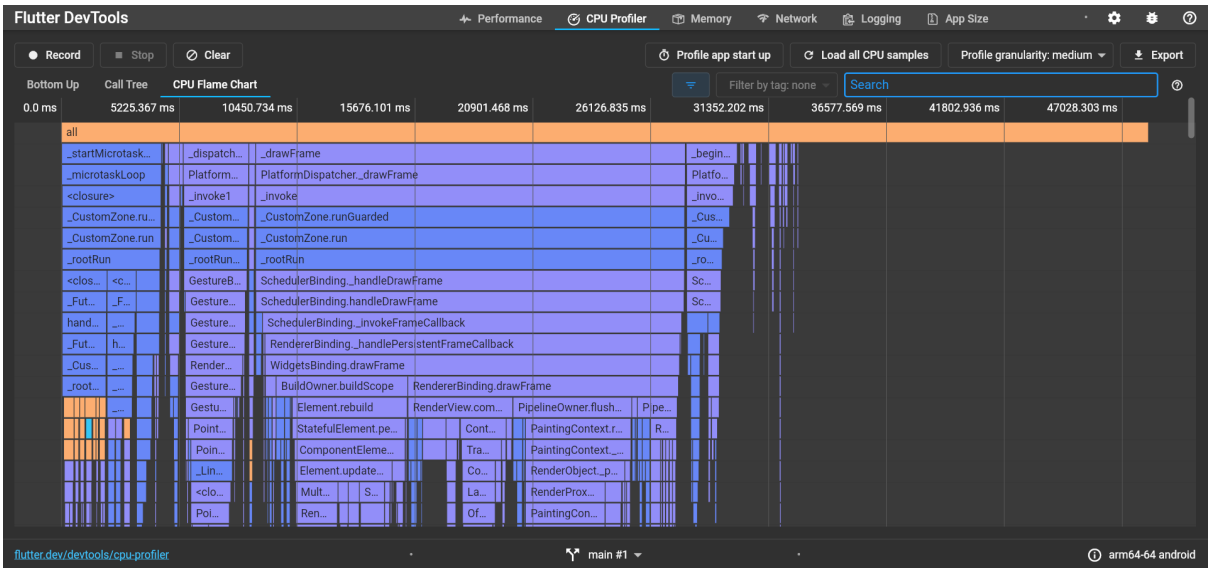


Figure 2: **Flame chart,** analysis of the CPU samples for the selected frame event

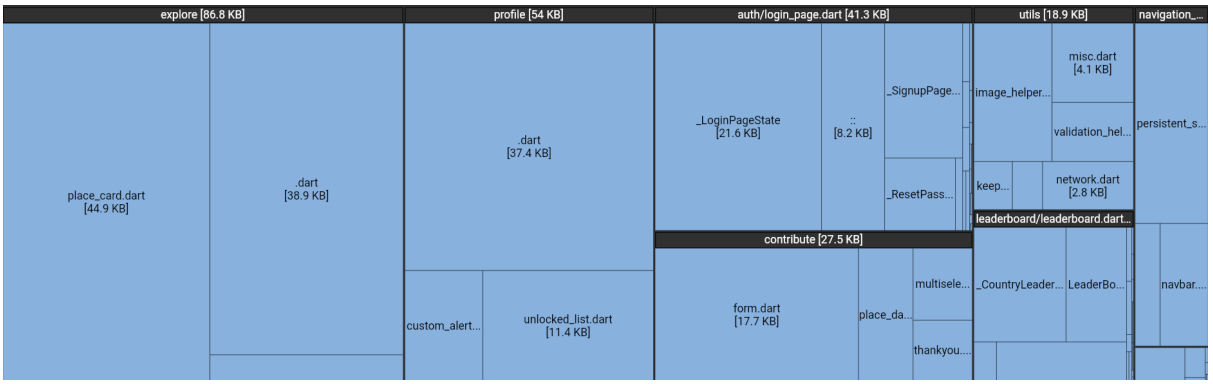


Figure 3: Analysis of the hierarchical structure of the **size data**

The profiling shown here was obtained on a physical device (OnePlus Nord).

2.6. Design Constraints

2.6.1. Standard Compliance

The app requires only the absolute minimum permissions needed to support core functionalities. On the registration side only email and a username are required, so no personal data. Clearly if the user wants to use Google or Facebook to log in, he must allow it and the requested information will be the strict necessary.
For the correct functioning of the app device GPS location is asked to visualize the user's position on the map.

2.6.2. Hardware Limitations

There are very few hardware limitations:

- Android or iOS device
- Internet connection
- GPS system (necessary to unlock places, which can still be displayed)

2.7. Software System Attributes

2.7.1. Reliability

The application should be available 24/7. Admins have the responsibility of the maintenance and should release updated versions of the app periodically.

2.7.2. Availability

The system has to provide a proper way to avoid the possibility of denied service, for example to have a system of redundant servers. This task is shifted on Firebase, which can be considered as a very reliable host. The system is expected to be available 99.99% of the time.

2.7.3. Security

All the sensitive information in the database has to be stored securely in order to avoid any chance, for an external and not authorized user, to read the data. The sensitive information mentioned above are users' credentials (email and password). On the servers, the security is guaranteed by Firebase policies: only admins can eventually access some data, while passwords aren't accessible by anyone.

2.7.4. Maintainability

In order to guarantee a good maintainability of the app, the code will be simplified as much as possible, and properly commented with natural language (English), especially in the sections in which the more logic resides.
Design patterns that are used and all the documentation will be provided.
Furthermore, the app will always be tested before a new release and changelogs will be generated for each new version of the system.

2.7.5. Compatibility

The target SDK is API level 31 (Android 12.0), however it supports Android version from 5.0 Lollipop or later versions (API level 20) due to dependency constraints.

3. Architecture

UrbanHunt employs the typical client-server architecture. The client-side was developed using Flutter, and the server-side through Firebase.

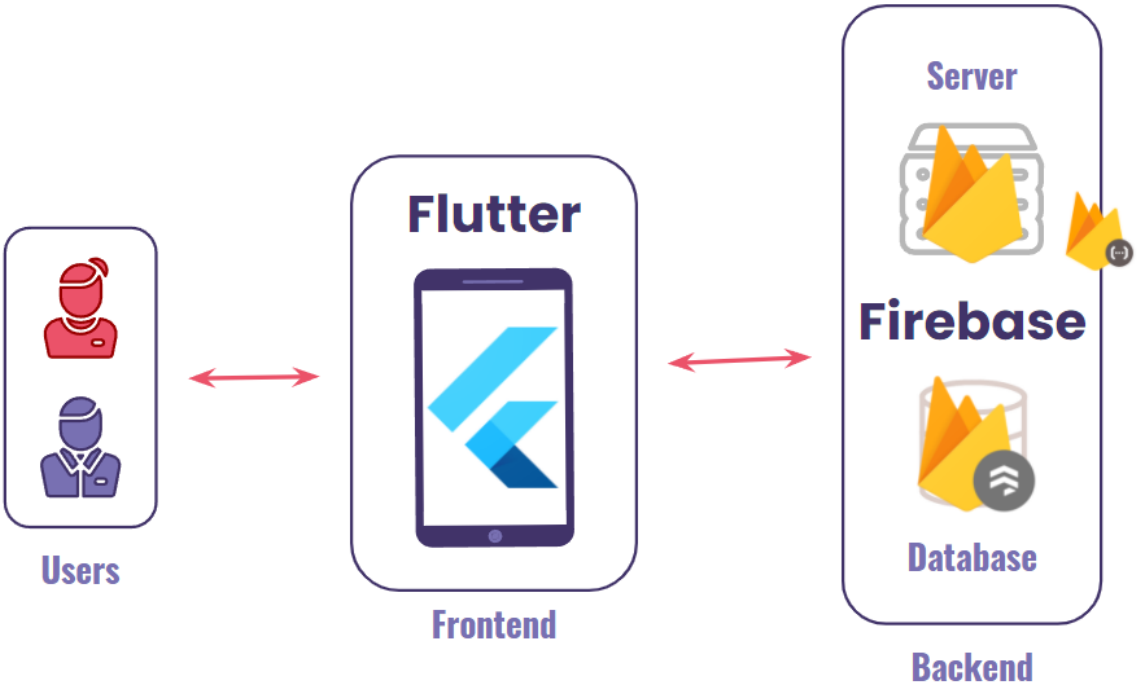


Figure 4: The architecture of the application

3.1. Client-side application: Flutter

UrbanHunt was developed using Flutter 2.8.1 using Dart 2.15.1.

Flutter is a UI toolkit developed by Google, that lets developers create beautiful and fast UI's with relative ease. It's meant to be cross platform, which means that it enables developers to make production level Android and iOS apps with a single codebase. Flutter 2 also supports web, desktop, which are not the goal of our app and have not been implemented.

Differently from other cross-platform frameworks, Flutter avoids performance problems caused by the need for a JavaScript bridge by using a compiled programming language, namely Dart. Dart is compiled “ahead of time” (AOT) into native code for multiple platforms. This allows Flutter to communicate with the platform without going through a JavaScript bridge that does a context switch. It also compiles to native code which in turn improves app startup times. In Flutter, it is all about Widgets, which are the elements that affect and control the view and interface to an app.

3.2. Server-side application: Firebase

The server-side of the application is entirely hosted by Google Firebase, which is a Backend-as-a-Service (BaaS) that relies on the more complex and flexible Google

Cloud Platform (GCP) cloud provider. The backend of the app is completely serverless. Serverless is a cloud-native development model that allows developers to build and run applications without having to manage servers. There are still servers in serverless, but they are abstracted away from app development. This gives a great advantage in terms of ease and speed of development, and perfectly suits our use-case, as we are more interested in these factors than squeezing all possible performance and customizing our backend code, which does not provide much value to the end user, especially during the earlier days of the app.

The server-side functionalities used by the app can be divided into the database, storage, and cloud functions, which are all implemented through services provided by Firebase and can be accessed and modified through the built-in console or the cli tools.

3.2.1. Firestore Database

Cloud Firestore is a NoSQL document database that simplifies storing, syncing, and querying data for mobile and web apps at global scale. Its client libraries provide live synchronization and offline support, while its security features and integrations with the Firebase and Google Cloud platforms accelerate building truly serverless apps.

In our data model we created two collections: places and users. The users collection contains all user data, plus the subcollection of unlockedPlaces, that contains the place unlocked by that user with specific information such as unlockDate and like/dislike interactions. The id of the unlocked places is the same as the original place in the places collection (reference), since duplication in this case was not suitable since unlockedPlaces data is not queried often.

The places collection contains all place data that are displayed on the map, and is populated through the contribute page.

places

place_id

address
city : "Milano"
country : "IT"
street : "6, Viale Brianza"
categories
0 "Culture"
1 "Food"
creatorId : "creator_user_id"
dislikes : 0
likes : 3
imgpath : "imagepath"
location : [45.48302795084957° N, 9.205589778721334° E]
name : "Mysterious Place"
lockedDescr : "Something to discover"
unlockedDescr : "Here is the story behind this place"

users

user_id

username : "Lore"
country : "IT"
score : 13
imageURL : " "

unlockedPlaces

unlockedPlace_id

disliked : true
liked : false
unlockDate : 27 january 2022 19:15:45 UTC+1

Figure 5: Firestore database structure

3.2.2. Storage

Cloud Storage for Firebase is a powerful, simple, high-scalability, and cost-effective object storage service. For the end-user it results in being secure and fast. Its functionality is letting us upload and download the data from anywhere. Firebase Storage SDK allows us to store the files and share with anyone and integrates with Firebase authentication.

We used Storage to store profile images and place images. Profile images are automatically retrieved from the user's Google or Facebook account, or manually uploaded, and Place images are uploaded from the contribute form.

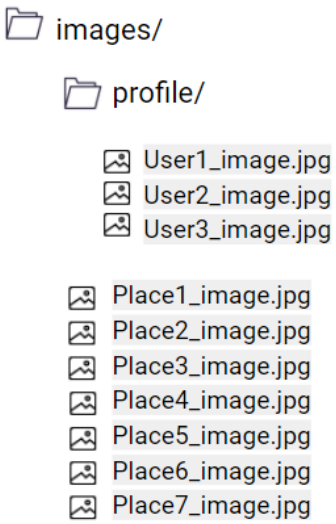


Figure 6: Storage structure

3.2.3. Authentication

Firebase Authentication handles every aspect of authentication including Sign-in, Sign-up and authentication of email. Besides the typical email and password sign-in method, it also integrates third party auth providers such as Facebook, Google and others.

UrbanHunt lets users log in with email and password, Facebook and Google, and takes care of merging all accounts into one, by checking if they have the same email, in order to give users the maximum choice and flexibility.

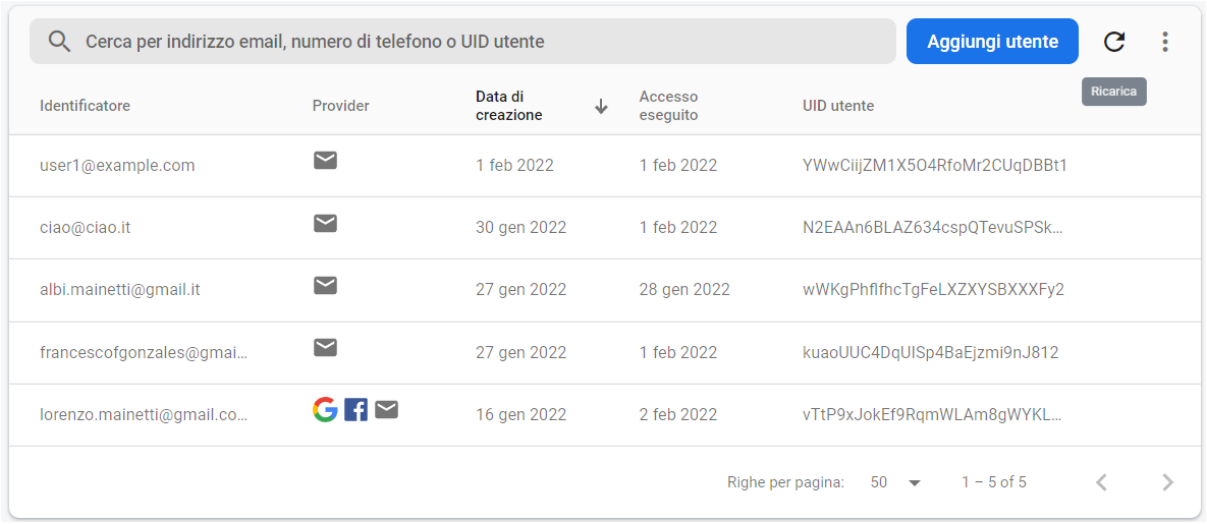


Figure 7: Authentication services

3.2.4. Functions

Firebase Cloud Functions offer a scalable solution for running back-end code in Google's Cloud Infrastructure and executing in response to app events. It lets developers write in JavaScript or Typescript.
This serverless approach has some great advantages:

- No need to run and maintain our own server
- Isolated code base for back-end code
- Billing only for the actual execution time of the code
- Highly scalable cloud infrastructure

UrbanHunt uses functions four functions:

- addPlacePoints: to give points to the users whenever they contribute a new place, which is triggered by an addition to the place collection
- deleteProfile: to remove the user document from users collection, triggered by delete event by Authentication
- firestoreDeleteProfileCleanUp: to remove his profile picture from storage, triggered by the previous delete from storage
- unlockPlacePoints: to give points to the users when they unlock a new place, which is triggered by an addition to his unlockPlaces collection.

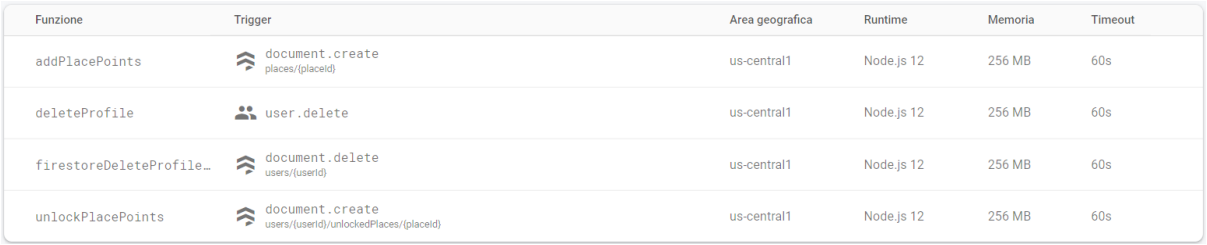


Figure 8: Deployed cloud functions

4. User Interface Design

UrbanHunt is implemented in Flutter, which comes with the great benefit of being cross-platform. Flutter provides the material design package that provides developers with pre-built UI components.
Below we provide the mockups and screenshots for the mobile app application that will be used by our users.

4.1. Logo

The logo of the application is minimalistic and visual. The blue simplified map recalls the app main color, while the red locked icon suggests the app main goal, that is to discover new uncommon places. The white background makes the logo stand out on the screen of the devices.

Figure 9: The application logo

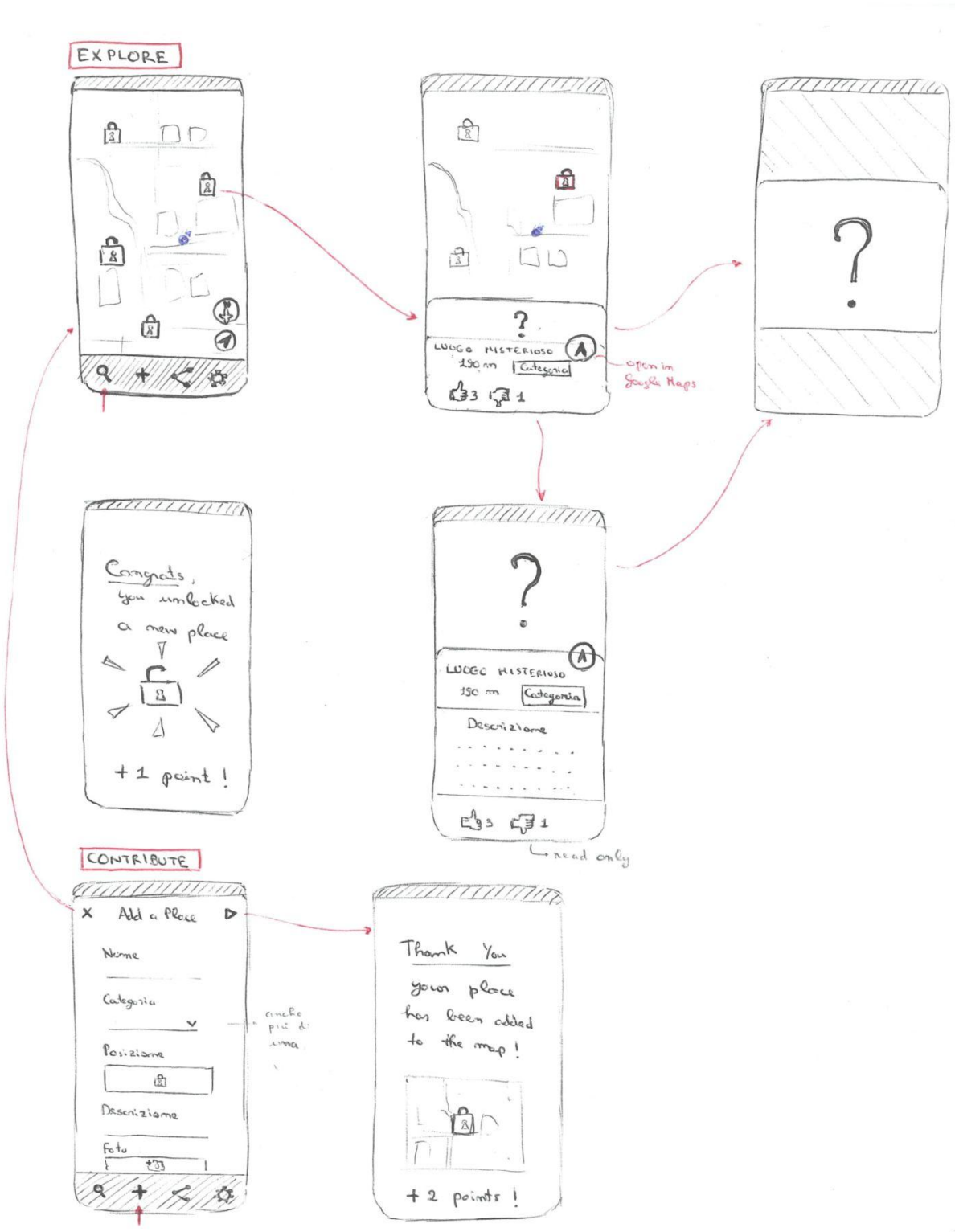
4.2. Theme

The theme of the application is overall minimalistic as the logo, not redundant and with the aim to provide an enjoyable experience to the user, also on the graphical appearance.
It is a modern design inspired by many top apps on the market.
The dominant colors are a matte white for the background of the pages, in order to be less bothersome, and an indigo blue for some major details, buttons and text fields.
Other secondary colors are green and red that intuitively are the color of respectively unlocked markers and locked markers on the Explore page, and black for text.
The main font chosen for the app is Roboto, a neo-grotesque font designed directly in Google's laboratories.

4.3. Initial sketches

For the UI design we started from some initial handmade-sketches for every screen that helped us to reach the final look implemented within our application and to see the different correlations between screens. The starting idea was very similar to the

current one and it evolved through different phases such as implementation and testing.



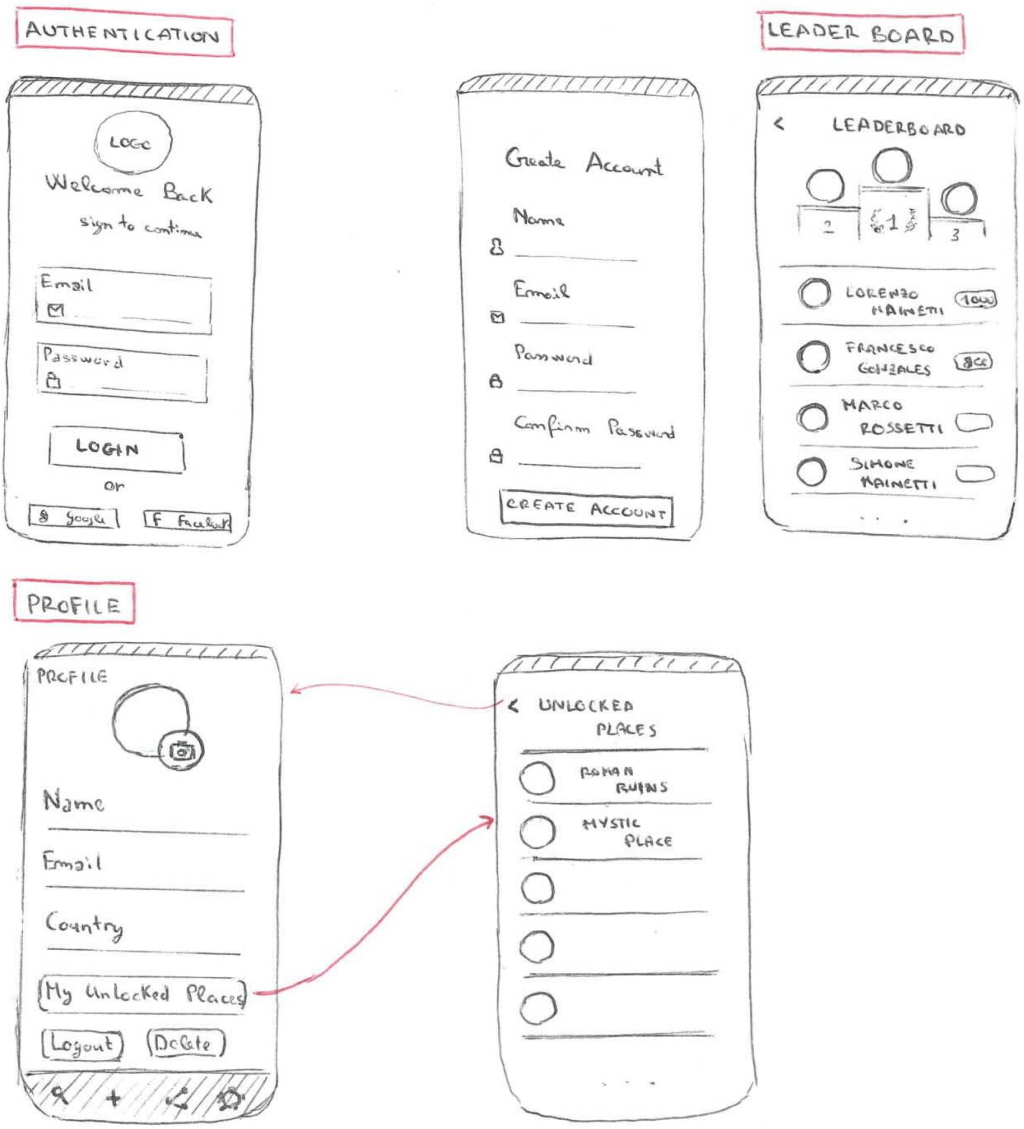


Figure 10: Initial sketches

4.4. Sign in, Sign up, Forgot password

Sign in is the very first screen presented upon opening the app. It shows the logo, the login options, and the possibility to navigate to the forgot password screen, where the user can have a reset link sent to the email, or the sign up page, in which a user may register using email and password. Other third party authentication methods are almost immediate.

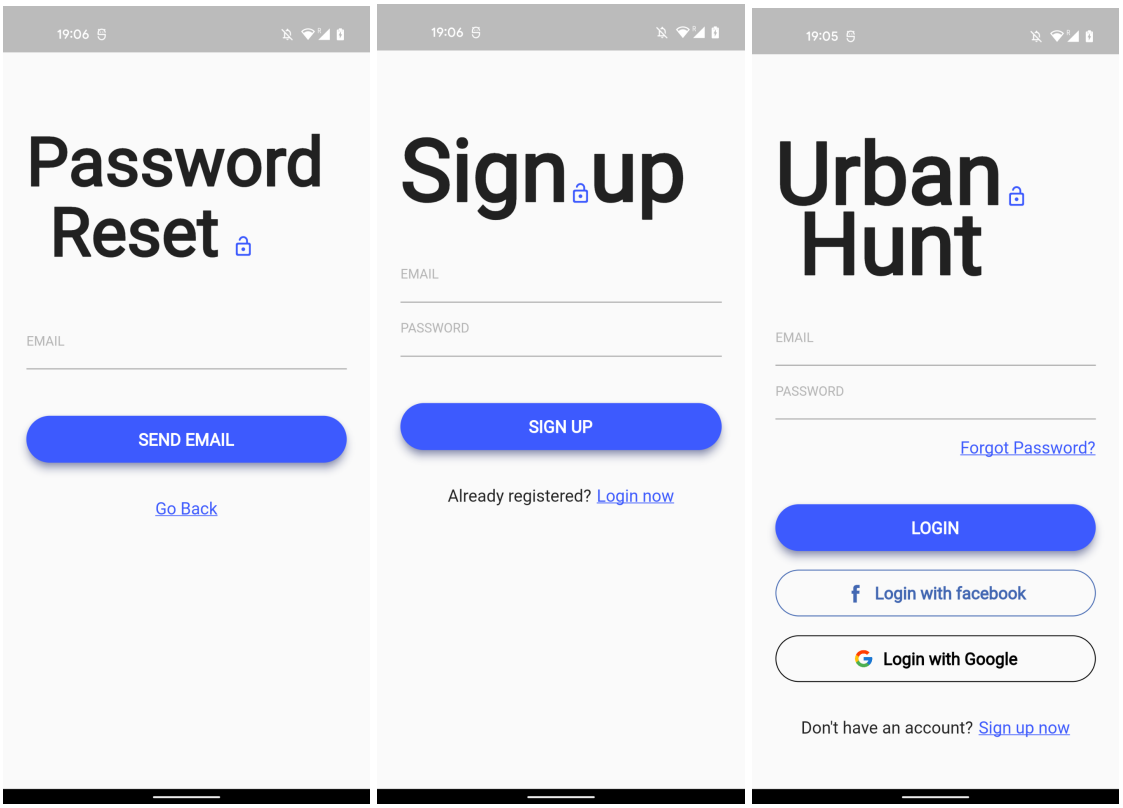


Figure 11: Login and Registration screens

4.5. Explore

The explore page contains all places with a custom marker, which is shaped as a red closed lock marker if it's locked, green unlocked otherwise. We also applied clustering to visually group very close markers, again using a custom design with a number inside indicating the number of grouped places.

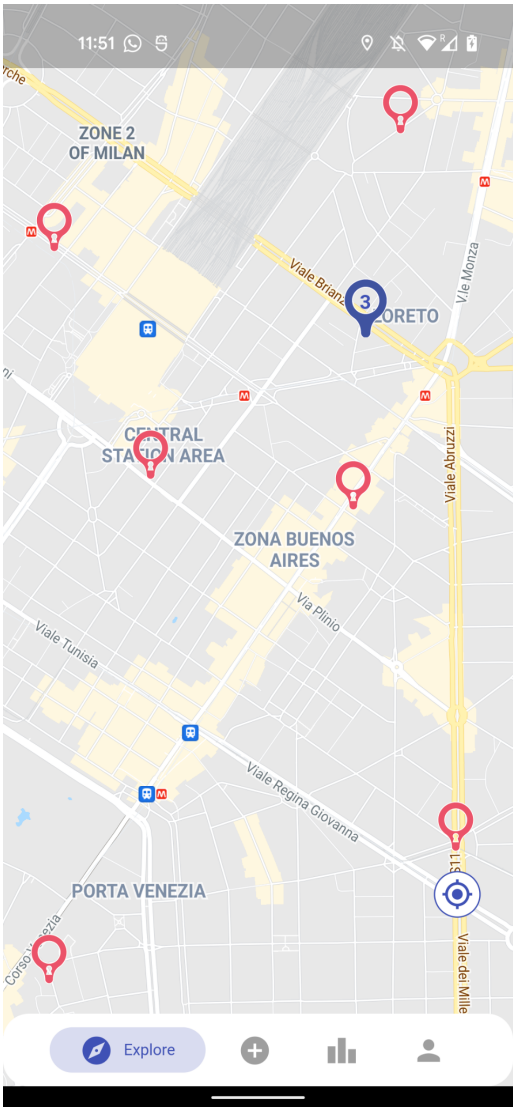


Figure 12: Explore page screen

4.6. Place

Places are shown differently depending on whether they are locked or unlocked. If the place is locked a very short intriguing description of the place is provided, and the image of the place appears blurred and if the place is unlocked there will be a full description of the place with some interesting notions about it, and a picture.

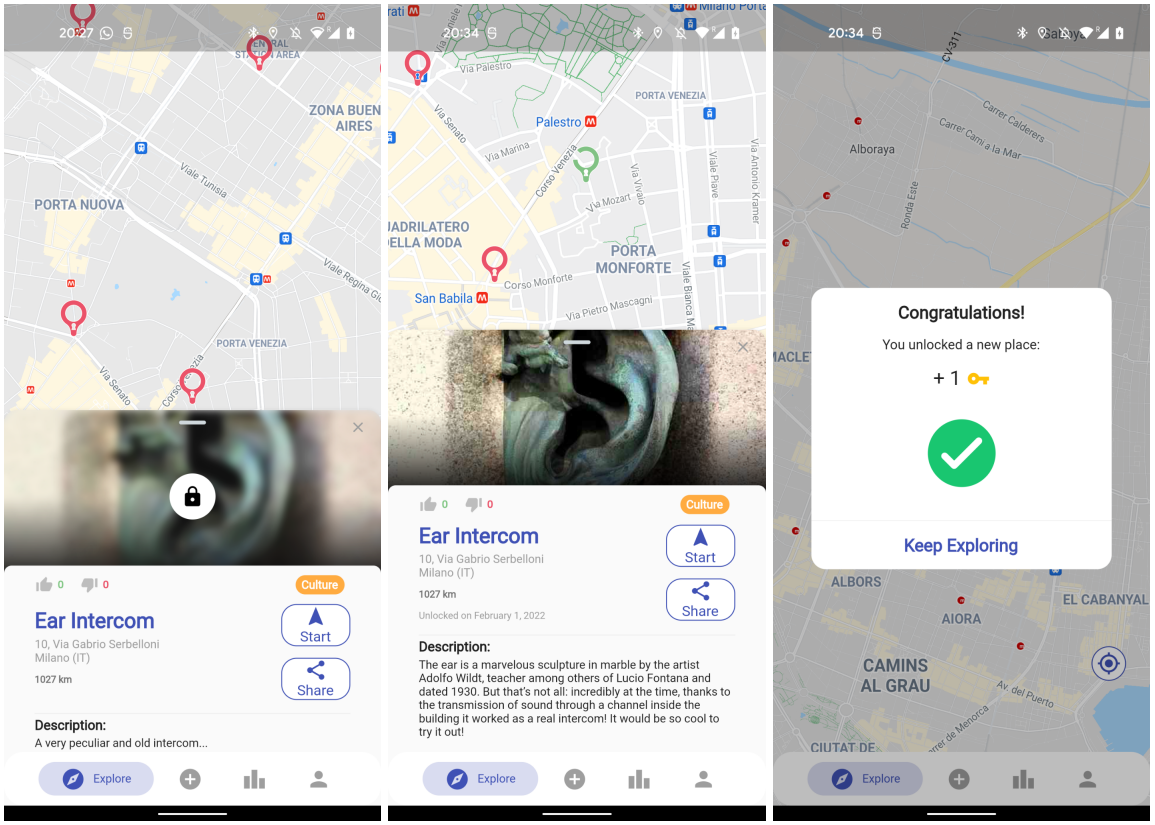


Figure 13: Place card screens

4.7. Contribute, Select address

This page lets a user add a new place. The form is clearly validated and displays relevant errors if the fields are incomplete or incorrectly formatted. To ease the user in filling the address the user can choose it by tapping on a map, or by writing the address in the search bar.

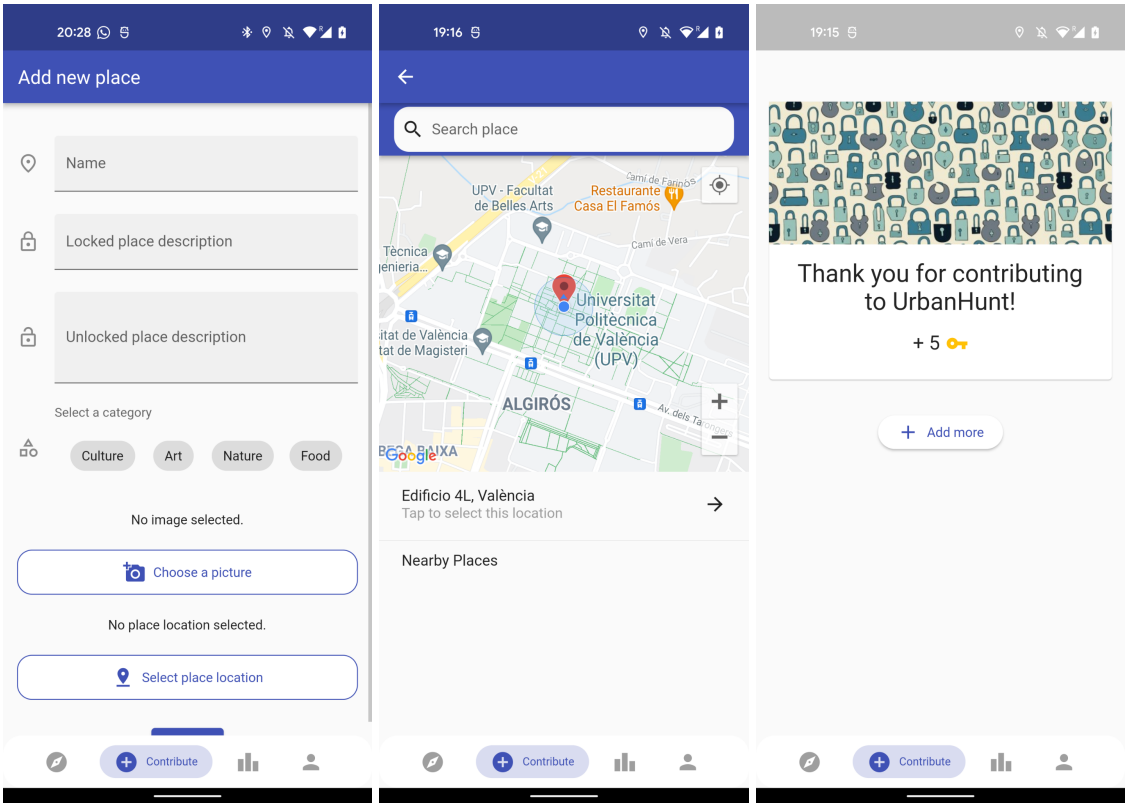


Figure 14: Contribute page screens

4.8. Leaderboard

The leaderboard displays the ranking of the users globally and nationally, that depends on the user's selected country, which can be changed in the profile settings.

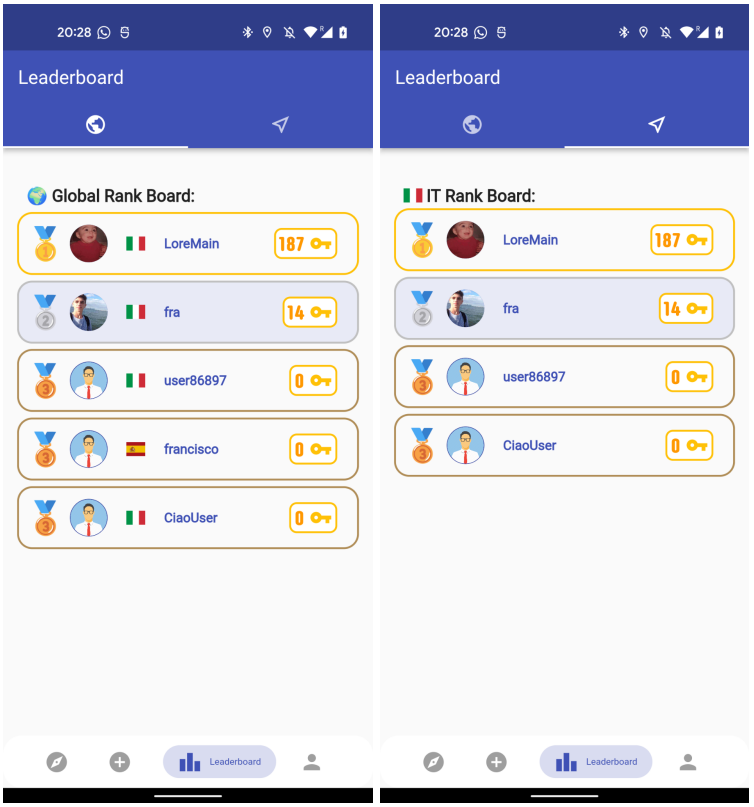


Figure 15: Leaderboard page screens

4.9. Profile, Unlocked places

The profile lets the user edit his information, such as his username, email, password (if he logged with email and password) and country. He can also see the list of his unlocked places and for each display the place information. The user can also change his profile picture and log out of his account.

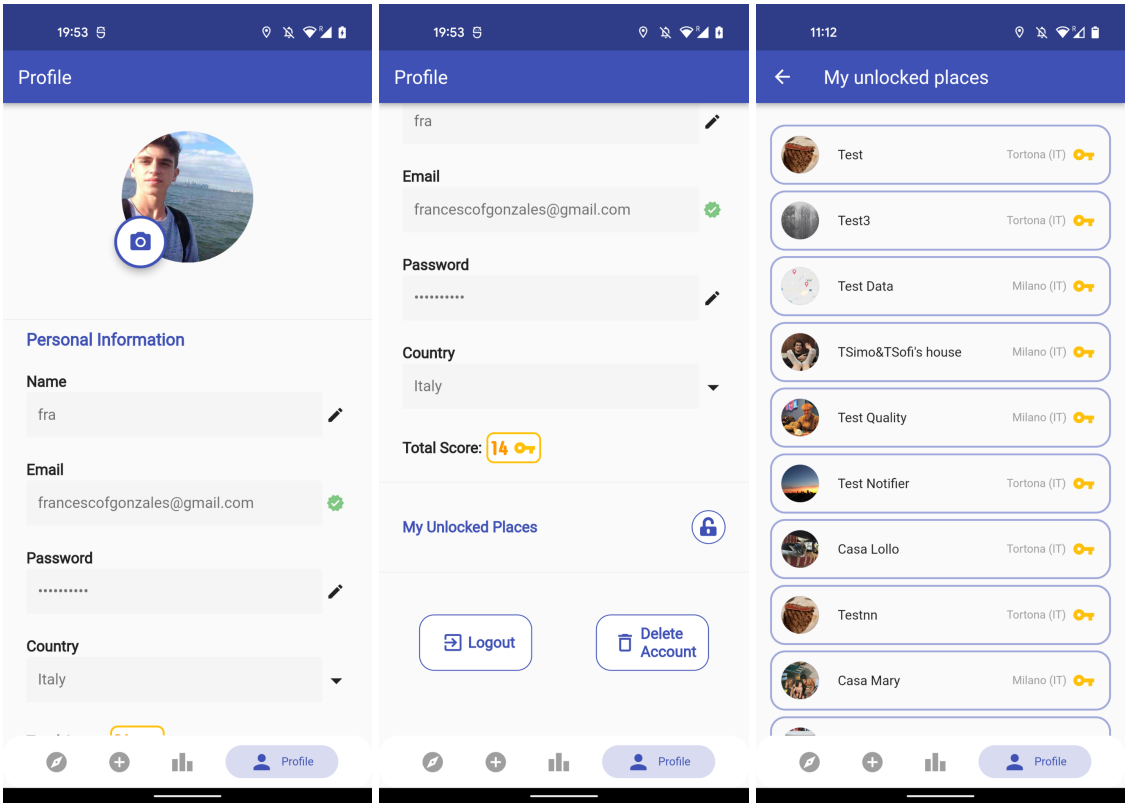


Figure 16: Profile page and unlocked list screens

4.10. Tablet layout

The application also supports bigger screens and it's completely usable on tablets. In order to better improve the user experience many screens were adjusted to fit the bigger screen and to take advantage of the space. The following pictures show some of the screens that were updated to satisfy these requirements.

All pages and most widgets were adjusted to be centered and adaptively resize according to the screen width, and some custom widgets are displayed in different ways. This is the case for the navigation bar, which is turned into a collapsible sidebar in the tablet view, also the place card takes advantage of the additional width and is shown on the left side instead of at the bottom, and snackbars have a different design. Portrait layout in tablets is also supported, and it displays elements using the right mixture of mobile and tablet widgets.

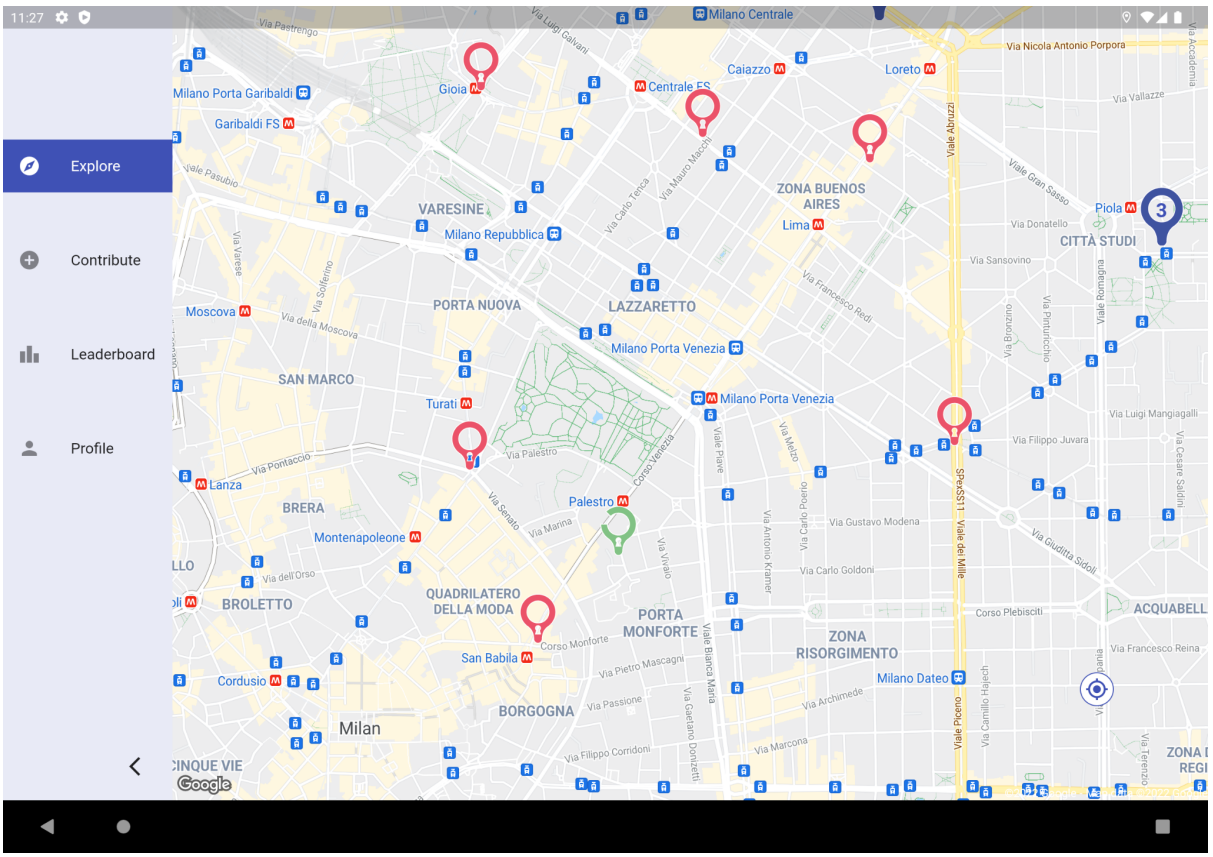


Figure 17: Explore page and extended sidebar screens

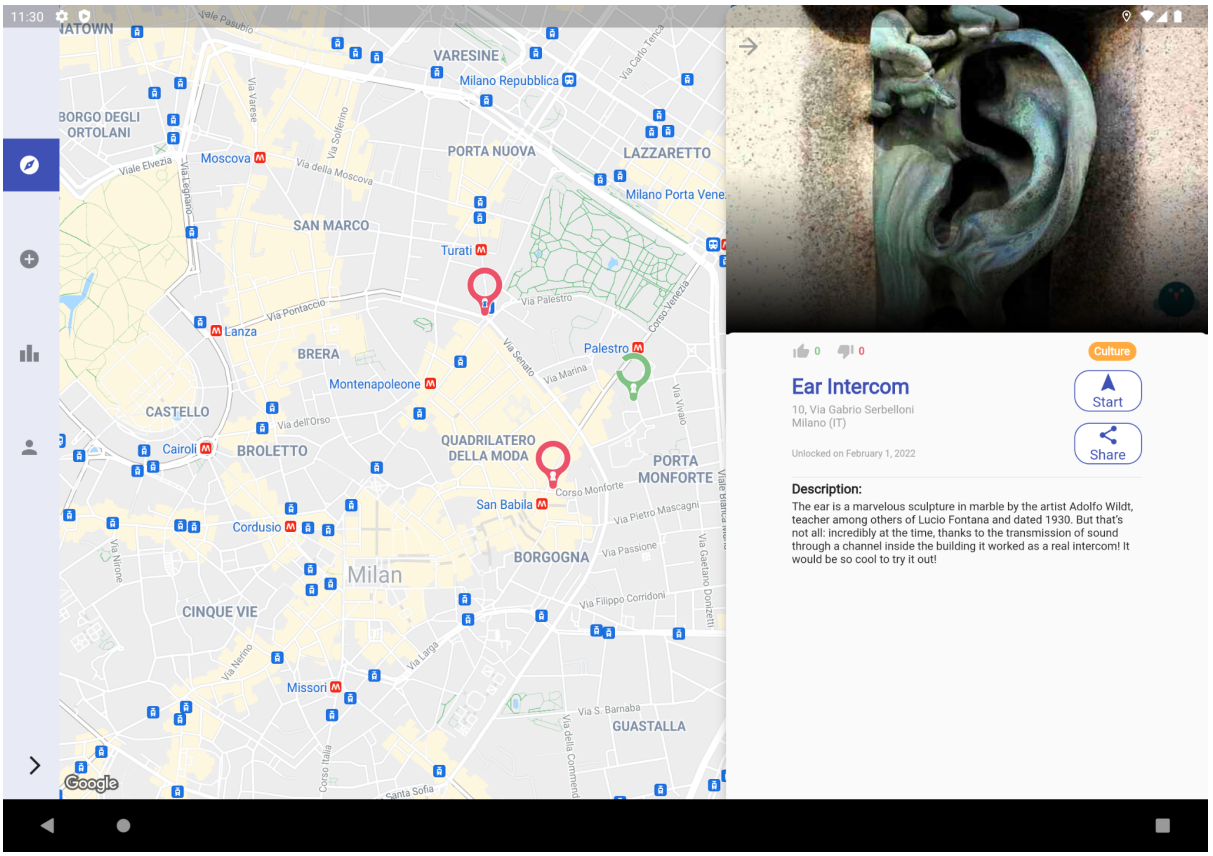


Figure 18: Place card and collapsed sidebar screens

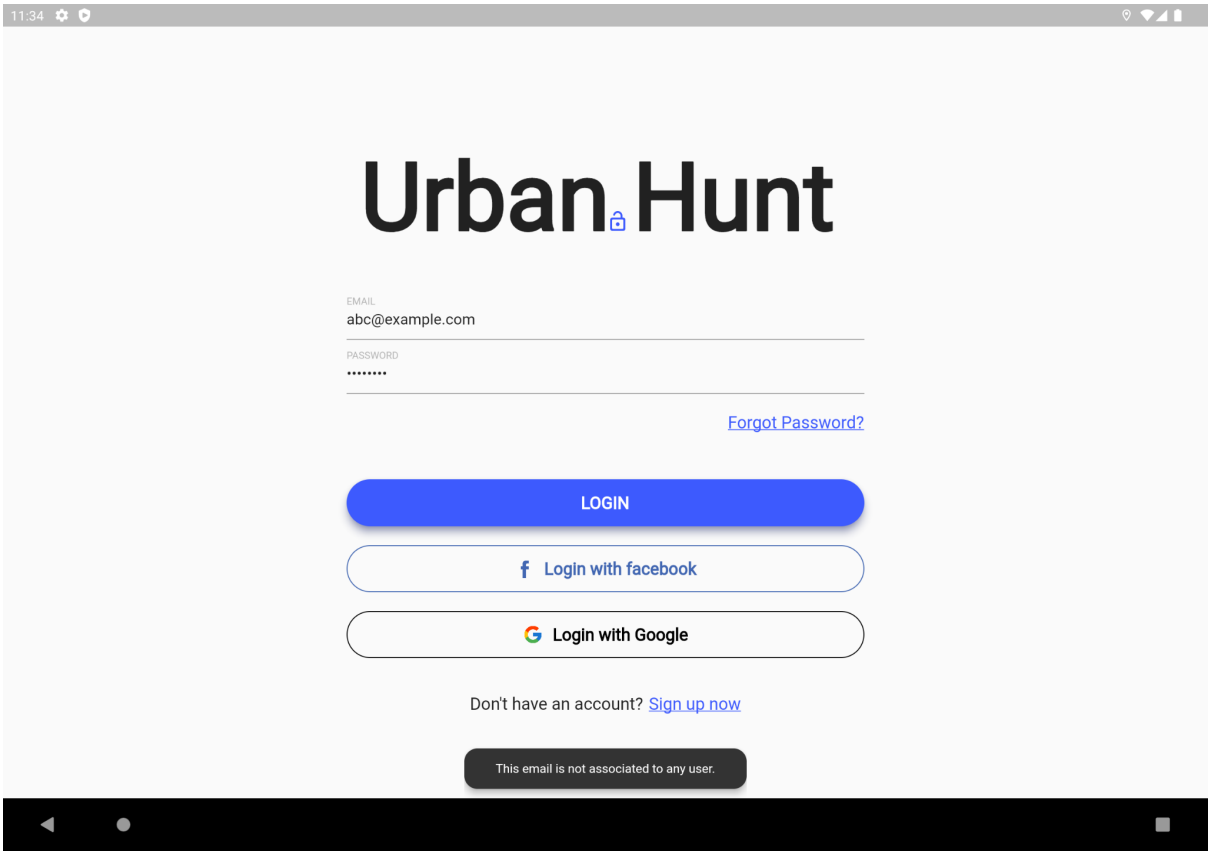


Figure 19: Login page screen

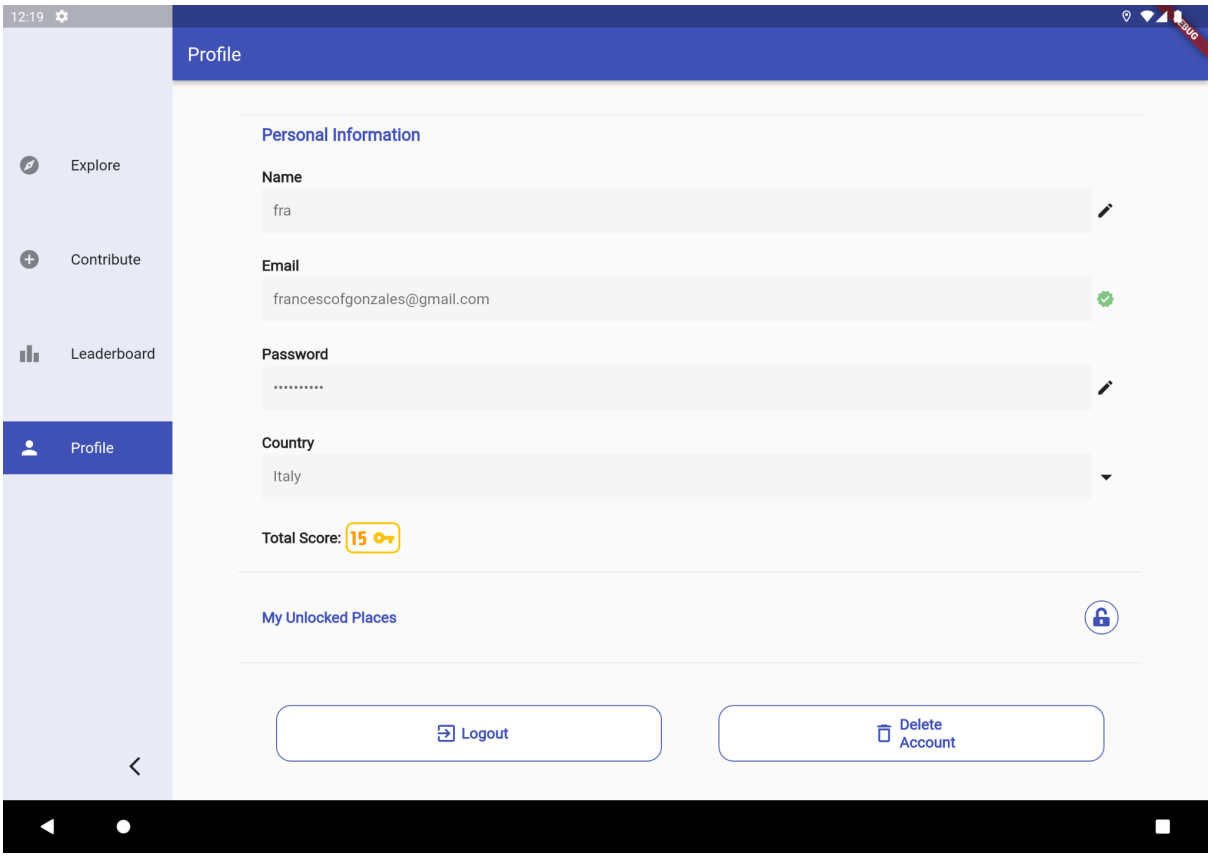


Figure 20: Profile page screen

5. External Services and Libraries

Besides Firebase and its APIs, which we thoroughly explained in Chapter 3.2, we leveraged other external services for a variety of different functions, both for backend functionalities and frontend components.

5.1. APIs

The following are the external APIs that we used to complete our backend by getting additional data, in some cases essential to provide the main functionalities of the app:

- **Google Maps**
  Google Maps is a paid API and related SDK, that allows us to implement a map into our application, by relying on Google's data and services infrastructure. It is a critical component of the app as the main page (Explore) relies on it.

- **Google Places**
  Google Places API and SDK let us integrate Google's Place details, search, and autocomplete into the app. We used it to let users find relevant places more easily in the contribute page.

- **Facebook Login**
  Lets users log in with Facebook, from which we retrieve their relevant data.

- **Google Login**
  Lets users log in with Google.

- **ip-api**
  This API provides simple IP Geolocation, by keeping and updating a database of IP blocks and their corresponding geographical location. We used this service to know users' countries without having to ask them at signup.

5.2. Packages

The application takes advantage of a list of packages offered by both the flutter team and third-party developers. In the following section we list some of the most relevant plugins that allowed us to save time and code while delivering a lot of functionalities and a strong user experience:

- **Firebase plugins**: used to access the various services, provided by firebase, used in the application. We included firebase_core, firebase_auth, cloud_firestore, firebase_storage.
- **Authentication**: flutter_facebook_auth and google_sign_in
- **Google maps services**: used to implement a google map widget in the Explore page. google_maps_flutter, google_maps_flutter_platform_interface, google_maps_cluster_manager.
- **Picker widgets**: image_picker, place_picker, country_picker.
- **Geolocator**: used to retrieve the location when searching for nearby cities.

- **Testing**: Useful mocking external libraries and internal entities. mockito, firebase_auth_mocks, firebase_storage_mocks, google_sign_in_mocks.

6. Implementation, Integration and Test Plan

6.1. Implementation and Integration

6.1.1. Implementation

UrbanHunt is a complex system composed of different layers, modules and sub-modules that interact and communicate among each other. Before starting with the implementation, the VCS tool must be correctly set up. Then, the database must be designed and implemented. After the database the serverless functions can be created, thus completing the backend. Once these components are created, it's time to start developing the mobile application: first, the UI is implemented using mock data to understand the flow of the user experience inside the application. Then the blocs that we employ to retrieve data and talk with the server should be created.

6.1.2. Integration

The integration of the system will be based on its architectural structure. The components that need to be integrated are:

- Firebase Authentication: It has the objective of performing the authentication process, including the management of authentication tokens and the encryption of users' information.
- Cloud Firestore: Its purpose is to act as a middleware between the clients and the database.
- Firebase Storage: This component is used to store large files.
- Cloud Functions: It is the module used to run server-side code.
- Mobile application: Its purpose is to act as a front-end application to let users access the service.
- External APIs: They are needed to provide additional data and features to the mobile application

There isn't a specific order of integration between these components since, aside from the mobile application, they are all COTS (provided using a SaaS paradigm) and ready to be used.

6.2. Testing

To implement the system, we decided to test the application mainly through debugging on both emulators and real devices since most of the system components are offered as SaaS. The only exception is the mobile application, for which we also performed unit testing on the main business logic components.

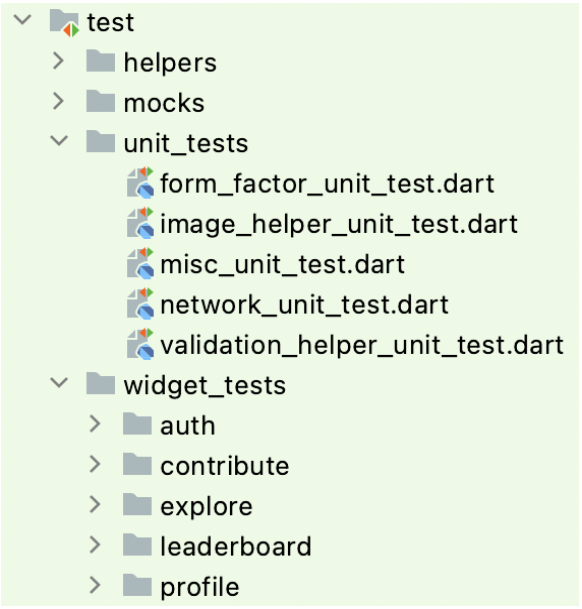


Figure 22: Tests of the app

6.2.1. Unit testing

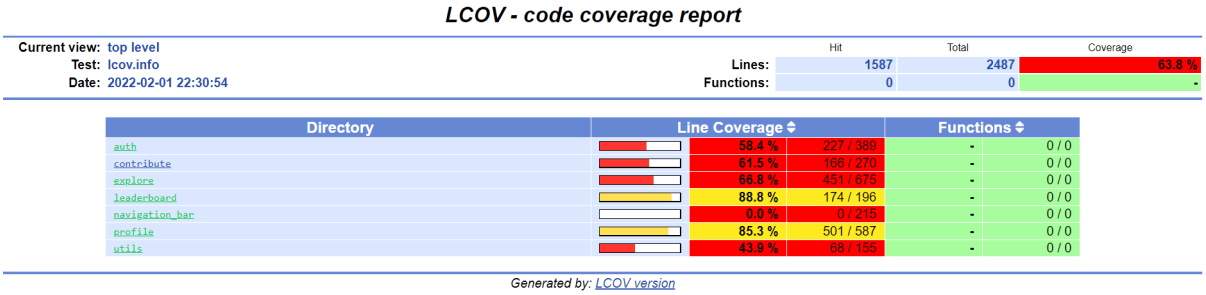
For what concerns unit testing, the objective is to ensure that the business logic components are behaving correctly. In some cases it was impossible to test some components since they were highly dependent on external modules, or required to mock permissions.

It should be noted that *UrbanHunt* does not contain complex highly independent business logic functions, that is the primary target of unit testing, rather its complexity comes rather from the integration of many external modules, and interactive widgets.

6.2.2. Widget testing

After unit testing, we carried out widget tests for those components containing important logic. In widget testing, the testing library wraps the widget that is under analysis in order to provide a BuildContext to it.

Then, the test environment performs a sequence of interactions (taps, scrolls, etc.) and scans the screen in order to verify that the widget's UI appears and works as expected.



LCOV - code coverage report

Current view: top level
Test: lcov.info
Date: 2022-02-01 22:30:54

	Hit	Total	Coverage
Lines:	1587	2487	63.8 %
Functions:	0	0	-

Directory	Line Coverage		Functions	
auth	58.4 %	227 / 389	-	0 / 0
contribute	61.5 %	166 / 270	-	0 / 0
explore	66.8 %	451 / 675	-	0 / 0
leaderboard	88.8 %	174 / 196	-	0 / 0
navigation_bar	0.0 %	0 / 215	-	0 / 0
profile	85.3 %	501 / 587	-	0 / 0
utils	43.9 %	68 / 155	-	0 / 0

Generated by: LCOV version

Figure 21: Code coverage report

6.2.3. Use cases for debugging

In this subsection are listed the main activities performed to debug the application. They are organized page by page, and they cover all the interactions that a user can have with the application.

Name	**Register**
Actor	User
Input Condition	User has *UrbanHunt* installed on his smartphone, and it is not registered
Event flow	1. The user opens the application 2. The user clicks on the sign up now button 3. The user inserts his email 4. The user inserts a password 5. The user confirms the password 6. The user presses the register button
Output Condition	The system logs the user into the service
Exception	1. Email pattern not respected ⇒ email error reported by the app 2. Password pattern not respected ⇒ password error reported by the app 3. The user already exists ⇒ user notified to use sign in

Name	**Login**
Actor	User
Input Condition	User has *UrbanHunt* installed on his smartphone, and it is registered but not authenticated
Event flow	1. The user opens the application 2. The user inserts his email 3. The user inserts a password 4. The user presses the login button
Output Condition	The system logs the user into the service
Exception	1. Email pattern not respected ⇒ email error reported by the app 2. Password pattern not respected ⇒ password error reported by the app 3. Wrong password inserted ⇒ password error reported by the app 4. The user not exists ⇒ user notified to use sign up

Name	**Forgot password**
Actor	User
Input Condition	User has *UrbanHunt* installed on his smartphone, and it is registered but he doesn't remember his password
Event flow	1. The user opens the application 2. The user clicks on "Forgot password" button 3. The user inserts his email 4. The user receives an email 5. The user modify his password
Output Condition	The system send an email to change password
Exception	1. Email pattern not respected ⇒ email error reported by the app 2. The user not exists ⇒ user notified to use sign up

Name	**Login with Facebook**
Actor	User
Input Condition	User has *UrbanHunt* installed on his smartphone and a facebook account
Event flow	1. The user opens the application 2. The user clicks on login with facebook button 3. The user allow the use of facebook to login
Output Condition	The system logs the user into the service
Exception	1. Facebook is disabled ⇒ error reported by the app 2. The user reject to allow the use of facebook => error reported by the app

Name	**Login with Google**
Actor	User
Input Condition	User has *UrbanHunt* installed on his smartphone and a Google account
Event flow	1. The user opens the application 2. The user clicks on login with Google button 3. The user allow the use of Google to login
Output Condition	The system logs the user into the service
Exception	1. Google is disabled ⇒ error reported by the app 2. The user reject to allow the use of Google => error reported by the app

Name	**Unlock place**
Actor	User
Input Condition	User is logged, his provided email is verified, he has the explore page open, he is less than 15 meters away from a place and the GPS is on
Event flow	1. The user clicks on a locked marker icon on the map that is less than 15 meters away 2. The user clicks on the lock icon on the place card image
Output Condition	The system shows the user an unlock popup with an unlock animation and the gained point
Exception	1. The user is more the 15 meters away from the selected place => error is reported by the app 2. The user is logged with a not verified email => error is reported by the app 3. The GPS is off => the user is asked to turn it on

Name	**Like place**
Actor	User
Input Condition	User is logged, his provided email is verified, he has the explore page or the unlocked list page open and he previously unlocked a place
Event flow	1. The user clicks on a unlocked marker icon on the map or on a place on the unlocked list 2. The user clicks on the like icon on the place card
Output Condition	The system updates the number of likes and dislikes
Exception	1. The user is logged with a not verified email => error is reported by the app 2. The user selected a locked place => error is reported by the app

Name	**Dislike place**
Actor	User
Input Condition	User is logged, his provided email is verified, he has the explore page open and he previously unlocked a place
Event flow	1. The user clicks on a unlocked marker icon on the map or on a place on the unlocked list 2. The user clicks on the dislike icon on the place card
Output Condition	The system updates the number of likes and dislikes
Exception	1. The user is logged with a not verified email => error is reported by the app 2. The user selected a locked place => error is reported by the app

Name	**Add new place**
Actor	User
Input Condition	User is logged, his provided email is verified and he has the contribute page open
Event flow	1. The user insert the name, unlocked description and unlocked description of the place 2. The user selects the categories for the new place 3. The user selects an image for the new place 4. The user selects the location of the new place 5. The user clicks on the submit button
Output Condition	The system adds the new place to the db and redirect the user to the ThankYou page showing the gained points
Exception	1. The name/descriptions are not valid => error is reported by the app 2. The user did not choose an image => error is reported by the app 3. The user did not choose a location => error is reported by the app

	4. The user is logged with a not verified email => error is reported by the app

Name	**Update country**
Actor	User
Input Condition	User is logged and has the profile page open
Event flow	1. The user taps the button "Country" 2. The user selects a new country from the list.
Output Condition	The system updates the user's country.
Exception	-

Name	**Update username**
Actor	User
Input Condition	User is logged and has the profile page open
Event flow	1. The user taps the pencil button next to the username 2. The user writes a username in the textfield 3. The user taps "Save".
Output Condition	The system updates the user's username.
Exception	-

Name	**Update profile picture**
Actor	User
Input Condition	User is logged and has the profile page open
Event flow	1. The user taps the camera button 2. The user selects an image from the local storage
Output Condition	The system updates the user's profile picture.
Exception	-

Name	**Change password**
Actor	User
Input Condition	User is logged with email and password and has the profile page open
Event flow	1. The user taps the pencil button next to the password field 2. The user writes a new password in the textfield 3. The user taps “Save”.
Output Condition	The system changes the user’s password
Exception	1. User did not authenticated recently a. Systems open an alert dialog telling to the users that a recent authentication is needed b. Users taps on "OK" c. The system redirects the user to the authentication page d. The user authenticates in to the service e. The user repeats the change password process

Name	**Logout**
Actor	User
Input Condition	User is logged and has the profile page open
Event flow	1. The user taps the button "Logout"
Output Condition	The system redirects the user to the sign in page
Exception	-

Name	**Delete account**
Actor	User
Input Condition	User is logged and has the profile page open
Event flow	1. The user taps the button "Delete account"
Output Condition	The system redirects the user to the sign in page
Exception	1. User did not authenticated recently a. Systems open an alert dialog telling to the users that a recent authentication is needed b. Users taps on "OK" c. The system redirects the user to the authentication page d. The user authenticates in to the service e. The user repeats the deletion process

7. Improvements and additional features

Following we report some additional ideas that might be implemented in the future in the *UrbanHunt* app:

- Edit places suggestions
- Add your friends
- Add quizzes to unlock places
- Machine Learning for crowdsourced data filtering
- Share unlocked place on social media
- Visualize other users on the map

8. References

This section includes the documents and the main sources referenced during the writing of this document and useful for the development of the application:

- https://flutter.dev/docs
- https://developer.android.com/
- https://pub.dev/
- https://firebase.google.com/docs
- https://firebase.flutter.dev/docs
- https://developers.google.com/maps/documentation
- https://developers.facebook.com/docs/facebook-login/
- https://ip-api.com/docs
- https://material.io/design
- https://dribbble.com/