POLITECNICO
MILANO 1863

Design and Implementation of Mobile Applications

A.Y. 2023/2024

Design Document

Prof. Baresi Luciano

Falasca Luigia

Mercurio Antonio, 232984

1. Introduction

1.1 Purpose

Affinder is a rental app that aims at helping people to find or offer an accommodation or to find a flatmate. The idea came from our personal experience: as non-resident students in Milan, we found it difficult to find both a place to live in and compatible people to share the accommodation with.

The app allows users to set some filters to easily search for flat the most suitable solution.

1.2 Definitions, Acronyms, Abbreviations

In the following sections, are listed and explained definitions, acronyms and abbreviations that will be frequently used in the following chapters.

1.2.1 Definitions

- **Registered User**: an account registered in the application with email and password; a registered user can have both a personal profile and as many house profiles as the accommodations they are willing to offer.
- **Profile**: any personal profile or house profile.
- **Personal Profile**: it is the set of personal information about the person that is looking for an accommodation; a registered user can have zero or one personal profile.
- **House Profile**: it is the list of all the accommodations offered by the registered user; a registered user can have zero or more house profiles.
- **Filters**: referred to a single profile, some properties that another profile should satisfy in order to be interesting for the referred registered user.
- **Like**: an action performed by any profile inside the application to communicate their interest in another profile.
- **Dislike**: an action performed by any profile inside the application to communicate they are not interested in another profile.
- **Filtered House Profiles**: referred to a personal profile, all the house profiles that match the filters of the referred profile.
- **Filtered Personal Profiles**: referred to a house profile, all the personal profiles that match the filters of the referred profile.
- **Already Seen Profiles**: referred to a profile, all the other-side profiles to which an action was performed (like/dislike).
- **Possible Profiles**: referred to a profile, all the other-side profiles that match the chosen filters and that are not in the already seen profiles set.
- **Match**: it is the link between two profiles that have liked each other.

1.2.2 Acronyms

- MVC: Model-View-Controller;
- API: Application Programming Interface;
- SDK: Software Development Kits;

- HTTP: HyperText Transfer Protocol;
- FR: Functional Requirements.

1.2.3 Abbreviations

- APP: Application.

1.3 Requirements

- **FR1** users should be able to register in the app;
- **FR2** registered users should be able to sign in/log out from the app;
- **FR3** registered users should be able to create/delete a personal profile;
- **FR4** registered users should be able to create/delete house profiles;
- **FR5** registered users should be able to see their house profiles (if they have any);
- **FR6** registered users should be able to see their personal profile (if they have one);
- **FR7** registered users should be able to modify information about their personal profile;
- **FR8** registered users should be able to modify information about their house profile;
- **FR9** registered users through their personal profile should be able to see all the possible house profiles;
- **FR10** registered users through their house profile should be able to see all the possible personal profiles;
- **FR11** registered users through their house/personal profiles should be able to put a like/dislike;
- **FR12** registered users through their house/personal profiles should be able to see detailed information about the profiles they are looking into;
- **FR13** registered users through their house/personal profiles should be able to read their notifications;
- **FR14** registered users through their house/personal profiles should be able to set some filters;
- **FR15** registered users through their house/personal profiles should be able to see a list of profiles with which they have matched;
- **FR16** registered users through their house/personal profiles should be able to send messages only to profiles with which they have matched;
- **FR17** registered users through their house/personal profiles should be able to receive messages only from profiles with which they have matched;
- **FR18** registered users through their house/personal profiles should be able to see the location of house profiles.

1.4 Implemented Features

In the following section there is the list of the main features that have been implemented.

1.4.1 Login and Registration

An user can create a new account with email and password. If they already have an account, they can log-in with their credentials. If the authentication phase is successful, they

will land on the homepage (where they will be asked to choose if they are looking for a house or they're offering one); otherwise, they will be redirected to the log-in/register screen.

1.4.2 Create a Personal Profile

If users look for a house and they already have a personal profile, they will be redirected to their personal profile homepage. Otherwise, they will fill in a form with all the personal information, which are: name, surname, gender, employment, a short description of themselves, birthday and four pictures of themselves (the first is compulsory, the others are optional).

1.4.3 Create a House Profile

If users offer a house, they will be redirected to the list of all their house profiles. At this point they can decide whether to enter an already existing house profile (by clicking on the corresponding tile) or to create a new one (by clicking on the button on the right side of the app-bar). If they want to create a new house profile, they will fill in a form, with information such as: the type of accommodation they are offering (e.g. apartment, single room and so on), its address (through a widget based on a Google Place prediction mechanism), the starting rent day, the ending rent day, the price (in euros per month), the floor in which the accommodation is, the maximum number of people that can stay in the accommodation, the number of bathrooms and four pictures of the accommodation (the first is compulsory, the others are optional).

1.4.4 Modify/Delete Personal/House Profile

By entering an already existing profile, in the profile screen, the user can visualises the information about that profile. At this point, by clicking on a button, they can modify the profile, edit some information, or delete it.

1.4.5 Set/Update Filters

By entering an already existing profile, in the search screen, users can open a form where they set some filters. In the case of a house profile the filters regard the gender, the employment and the age of the people. Meanwhile, in the case of a personal profile, the filters regard the city and the price of the accommodation. Users can change filters any time they want.

1.4.6 Swipe Between Personal/House Profile

By entering an already existing profile, in the search screen, users visualise all the possible profiles, one at a time (house profiles will see personal profiles, personal profiles will see house profiles). At this point, users can perform three actions: put like or put dislike and see all the information about that profile. Whenever a profile performs a "put like" action, there is a match if the liked profile already reciprocated: in this case a dialog will pop up on the user's screen. In any case, whenever users perform a like/dislike action, a new profile (from all the possible ones) will be shown.

1.4.7 See Notifications

By entering an already existing profile, in the search screen, the users can visualise their notifications, by clicking on the corresponding button on the bar. This page will show them all the matches that specific profile has made (if any), ordered by the timestamp, from the most recent (top) to the least recent (bottom).

1.4.8 Chat

By entering an already existing profile, in the chat screen, users visualise all the profiles with which they have matched but they haven't started a chat with yet (on the top, scrolling from the left to the right). Just below them, users can find the started chats, ordered by the timestamp of the last message, from the most recent (top) to the least recent (bottom).

In both cases, by clicking on the tile, users will be redirected to the chat page where they can read previous messages or send new ones.

1.4.9 See the Position of a House Profile

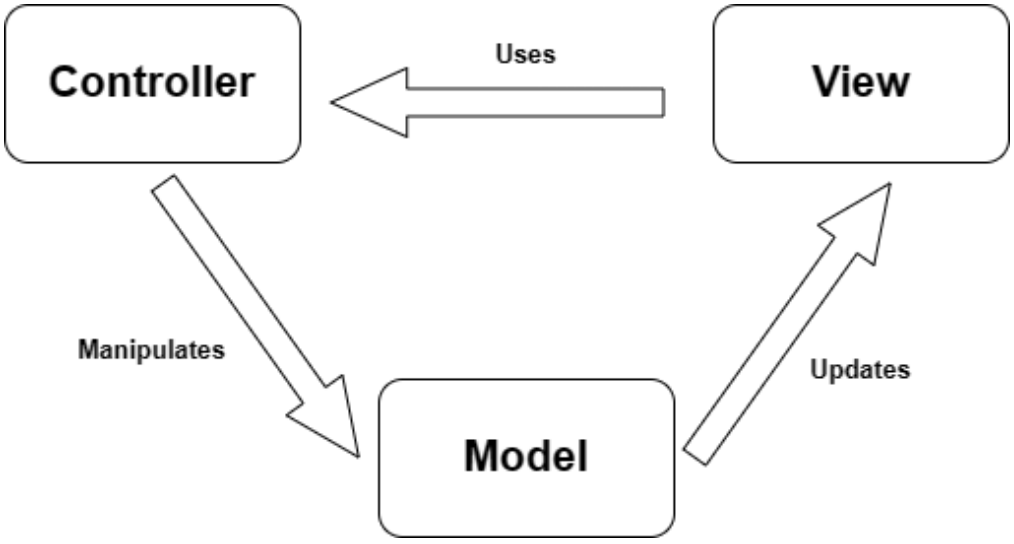
Every time users visualise information about a house profile, they can open a map, by clicking on a button, to see where the house is located.

2. Architectural Design

2.1 Overview

The system is based on a typical client-server architecture.

The internal structure of the client application follows the MVC pattern. It is an architectural pattern in software development that separates the business logic by dividing the related program responsibilities into three interconnected components.



For the server application, Firebase services and additional tools offered by Google Places are used. Firebase is an app development platform, backed by Google.

2.2 Model

The model handles data application. In our app the following data model is used:

- **Utente**: contains the unique identifier (UID) of the user.
- **HouseProfile** and **PersonalProfile**: contain all the information related to the House Profile and the Personal Profile respectively, with their UID.
- **Preference**: contains the profile's UID that made the action, the profile's UID that has received the action and the type of the action (like/dislike); this is used to keep track of the already seen profiles and it is used to check the potential match.
- **MatchPeople**: contains a UID couple that have matched with each other.
- **FilterHouse** and **FilterPerson**: contain all the properties of the house or person set by a profile.
- **Message**: contains the sender UID, the receiver UID, the message's timestamp and the message's text.
- **Chat**: contains a UID couple that started a chat, with the relative information about the messages history and the last message.

2.3 View

The view is responsible for presenting information to users. In the following section the core screens, replicated in both House Profile and Personal Profile, are described.

- **Wrapper**: is responsible for showing the authentication page (if the user is not authenticated) or the home screen (otherwise), this class also stores the authenticated user.
- **Login**: is responsible for the validation of the login credentials and the submission of those.
- **Register**: is responsible for the validation and submission of the user's data.
- **Home**: is responsible for the navigation between the House Profiles and the Personal Profile.
- **SearchLayout**: get the filtered profiles that are not already seen from the Database and handle all the events in the Search view.
- **Swipe**: is used to manage the single profile to fetch pictures and handle the action inside each profile card.
- **Notification**: shows all profile's matches, ordered by descendant timestamp.
- **ChatLayout**: shows new matches and initialised chats, allowing to start or continue them.
- **ChatPage**: is responsible for managing the chats with the messages.
- **InfoProfile**: fetches the profile from the given UID and puts it in its state to be able to display it.
- **ProfileLayout**: handles the changes of the user data.
- **MapScreen**: shows the location of the house profile using Google Places.
- **HouseList**: shows the list of all house profiles related to the user.
- **Filters**: shows a form in which users are asked to select parameters to filter their search.

2.4 Controller

The controller accepts the user's input, via the view, and it translates it into commands for the model and view. The controller modifies the state of the model and view based on the user's commands. This interacts also with the Server Application, calling Firebase functions. This allows Client Application to retrieve from or store data on the database. In the app, different classes for each macro-category of functions were implemented. The main ones, with the related methods, are:

- **AuthService**: handles the registration and login processes.

  – *registerWithEmailAndPassword*: registers a user into the system using an email and a password;

  – *getCurrentUser*: returns the current user;

  – *signInWithEmailAndPassword*: logs a user into the system using the email and the password;

  – *signOut*: logs a user out from the system.

- **DatabaseService**: handles all the information about users, profiles and filters.

  - *updatePersonalProfile*: creates/updates a Personal Profile;

  - *getPersonalProfile*: returns the Personal Profile;

  - *getAllPersonalProfile*: used in the Search Screen of the house profile, returns all the personal profiles in the Database;

  - *createHouseProfile*/*updateHouseProfile*: creates/updates a House Profile;

  - *getMyHouse/getHouses*: gets a/all House Profile(s) related to a specific user;

  - *getAllHouses*: used in the Search Screen of the personal profile, returns all the house profiles in the Database;

  - *getAlreadySeenProfile*: returns all the profiles that have been already seen by a specific profile;

  - *updateFilterHouse/getFilterHouse*: creates or updates/fetches filters related to a personal profile;

  - *updateFilterPerson/getFilterPerson*: creates or updates/fetches filters related to a house profile;

- **MatchService**: handles all the profiles' actions (like and dislike), checking for matches, and handles the notification as well.

  - *putPreferences/getPreferences*: creates/returns a Preference;

  - *createNewMatch*: creates a new Match in the Database;

  - *getMatchedProfile*: used in the Chat Screen to show all the matched profiles that haven't started a chat yet;

  - *getMatchWithTmp*: used in the Notification Screen to show all the matched profiles with the corresponding timestamp;

  - *checkMatch*: it checks if there is a match between two profiles; it is called in the Search Screen when a profile puts a like to another profile; if there is a match, it creates a new Match;

- *createNotification/resetNotification/getNotification*: creates, resets or returns a Notification, which is an Integer in the Database containing the number of unread notifications for a specific profile.

- **ChatService**: handles the chats.

  - *sendMessage*: used to the profile to send a new message; it also updates the last message and the notifications related to the chat;

  - *getMessage*: fetches the message to show it in the single ChatPage Screen;

  - *getStartedChat*: fetches the list of profiles with a started chat.

- **ImageService**: handles the images-related functions.

  - *uploadFile*: uploads the image in the database.

2.5 External Service

2.5.1 Firebase Authentication

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more, even though, for the purposes of this app, authentication methods are limited to emails and passwords.

2.5.2 Firebase Cloud Firestore

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps data in sync across client apps through real time listeners. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.

2.5.3 Firebase Cloud Storage

Cloud Storage for Firebase is built on fast and secure Google Cloud infrastructure for app developers who need to store and serve user-generated content, such as photos or videos.

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. For the purposes of this app, its use is limited to storing profile pictures and house images.

2.5.4 Google Places

The Places API is a service that accepts HTTP requests for location data through a variety of methods. It returns formatted location data and imagery about establishments, geographic locations, or prominent points of interest. For the purposes of this app, it is used as a method to guarantee the correctness of inserted addresses and to extract from them all the information necessary for the app to work (e.g. city, latitude and longitude), which are also used to build markers on maps through which houses locations are displayed and shown to the users.

2.6 Dependencies

In the following, the list of the most notable dependencies used in the project:

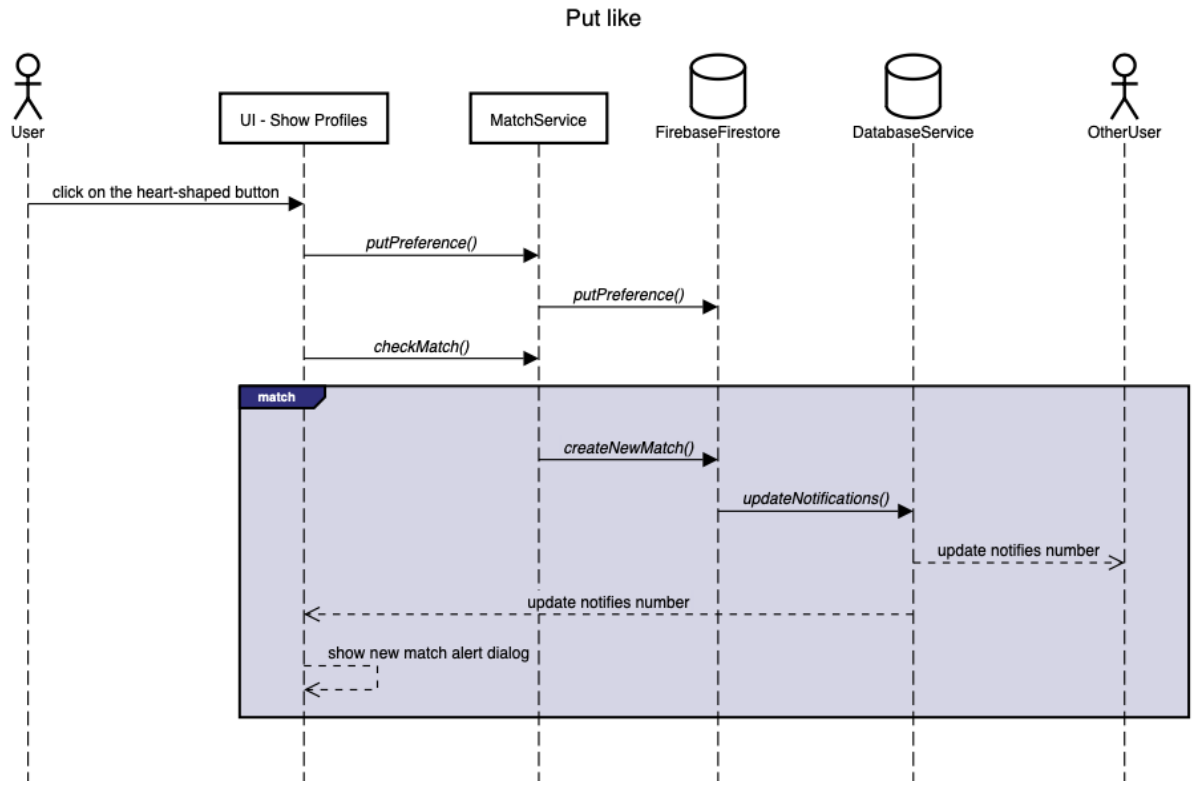
- **firebase_core**: used to initialise Firebase services;
- **firebase_auth**: used for authentication with email and password;
- **cloud_firestore**: necessary to use the Cloud Firestore API;
- **firebase_storage**: necessary to use the Firebase Cloud Storage API;
- **image_picker**: used for picking images from the image library, and taking new pictures with the camera;
- **image_picker_ios**: used for iOS for picking images from the image library, and taking new pictures with the camera;
- **path**: provides common operations for manipulating paths, used for the upload of the image;
- **path_provider**: a Flutter plugin for finding commonly used locations on the filesystem;
- **google_fonts**: package to use fonts from fonts.google.com;
- **google_maps_flutter**: provides a Google Maps widget;
- **google_places_flutter**: provides methods for helping the users insert correct addresses and help them search for them through prediction mechanisms;
- **geocoding_resolver**: provides methods for converting between geographic coordinates and addresses;
- **coverage**: provides coverage data collection, manipulation, and formatting for Dart.

2.7 Runtime view

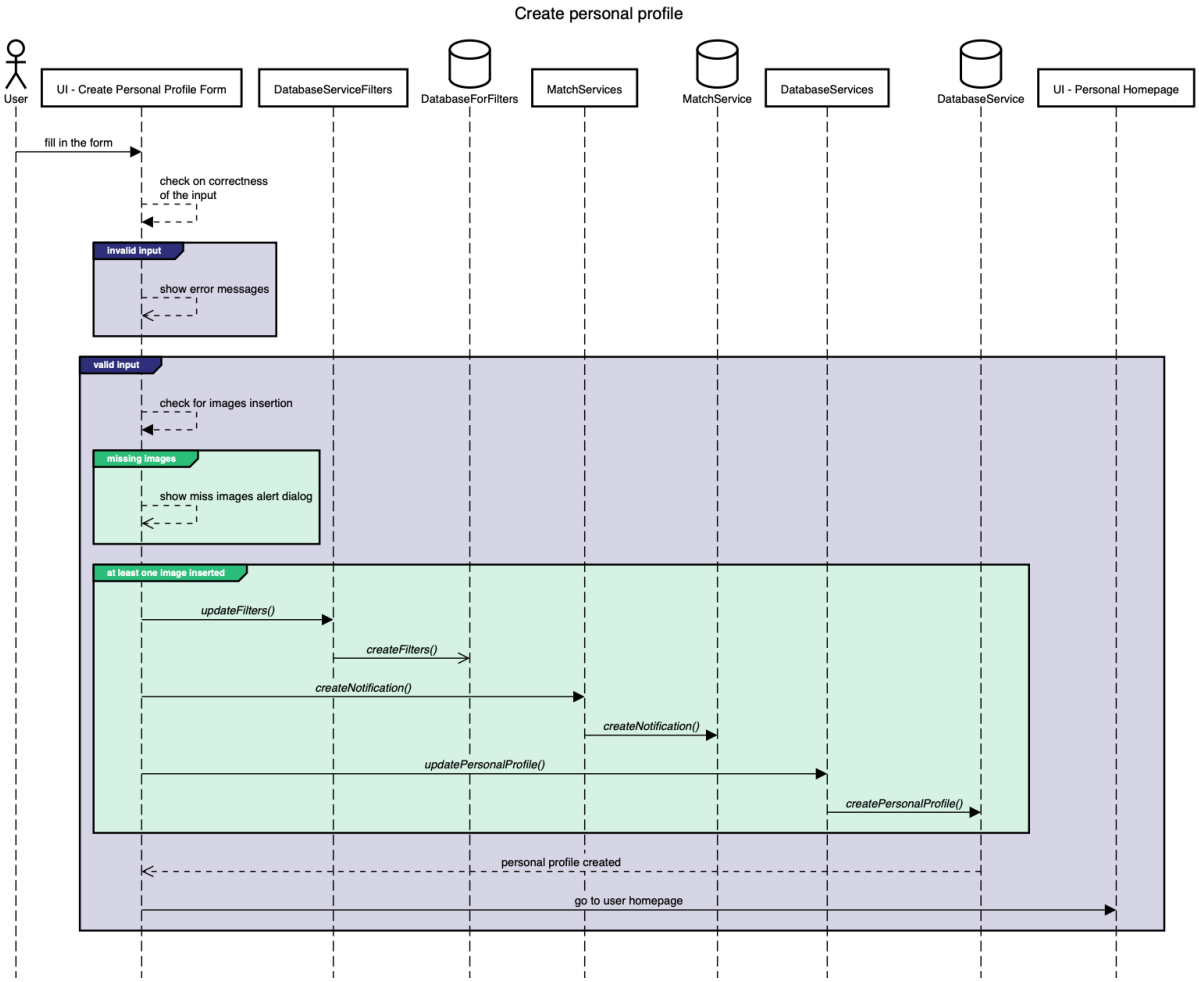
The following diagrams show three examples of process interactions triggered by an action performed by the user.

The first one shows how the “like” mechanism is handled and managed starting from the UI up to the database; the second one shows what happens when an user wants to create a Personal Profile; the third one shows how the registration process is handled.

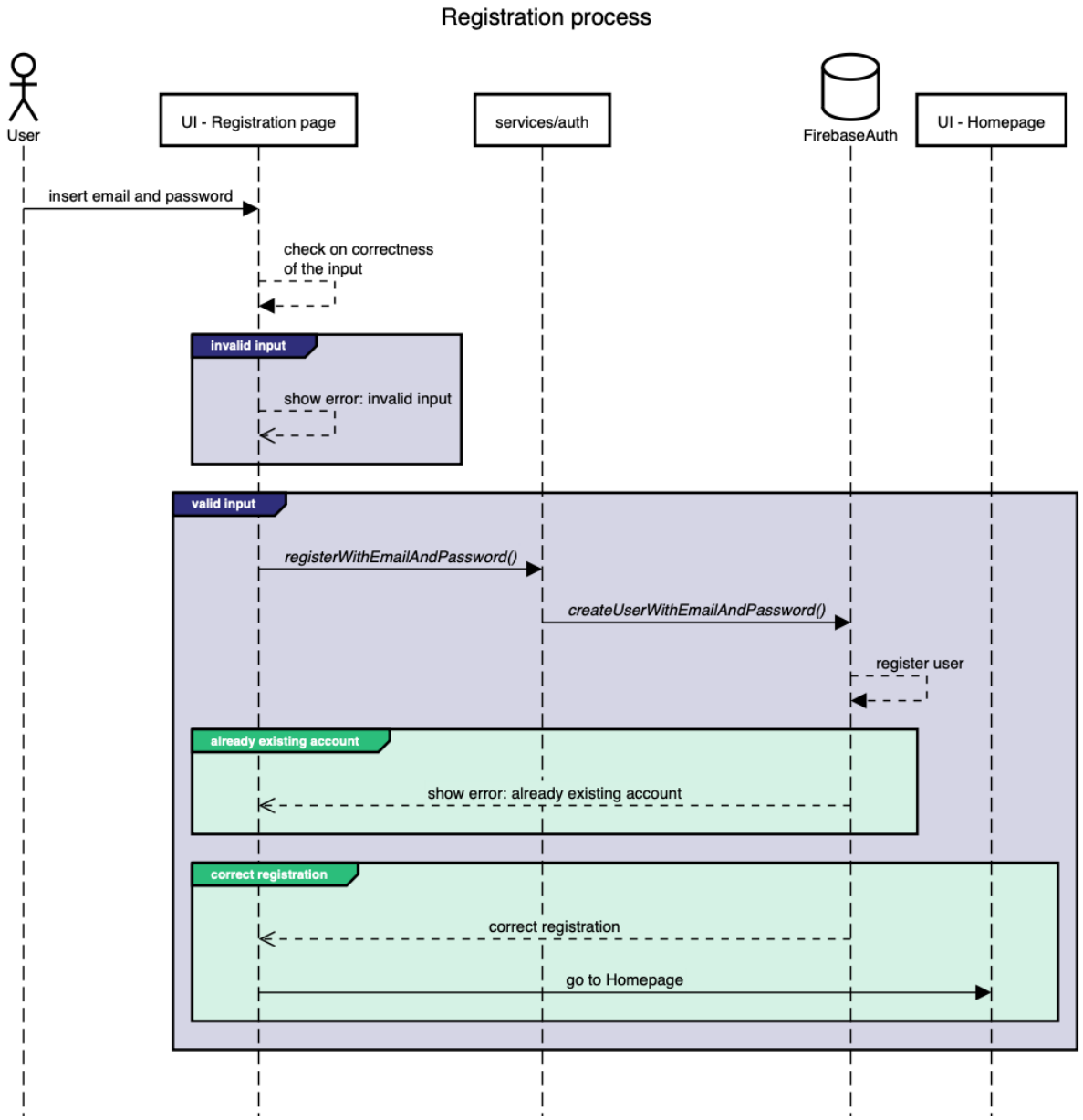
2.7.1 Put like



2.7.2 Create personal profile



2.7.3 Registration process



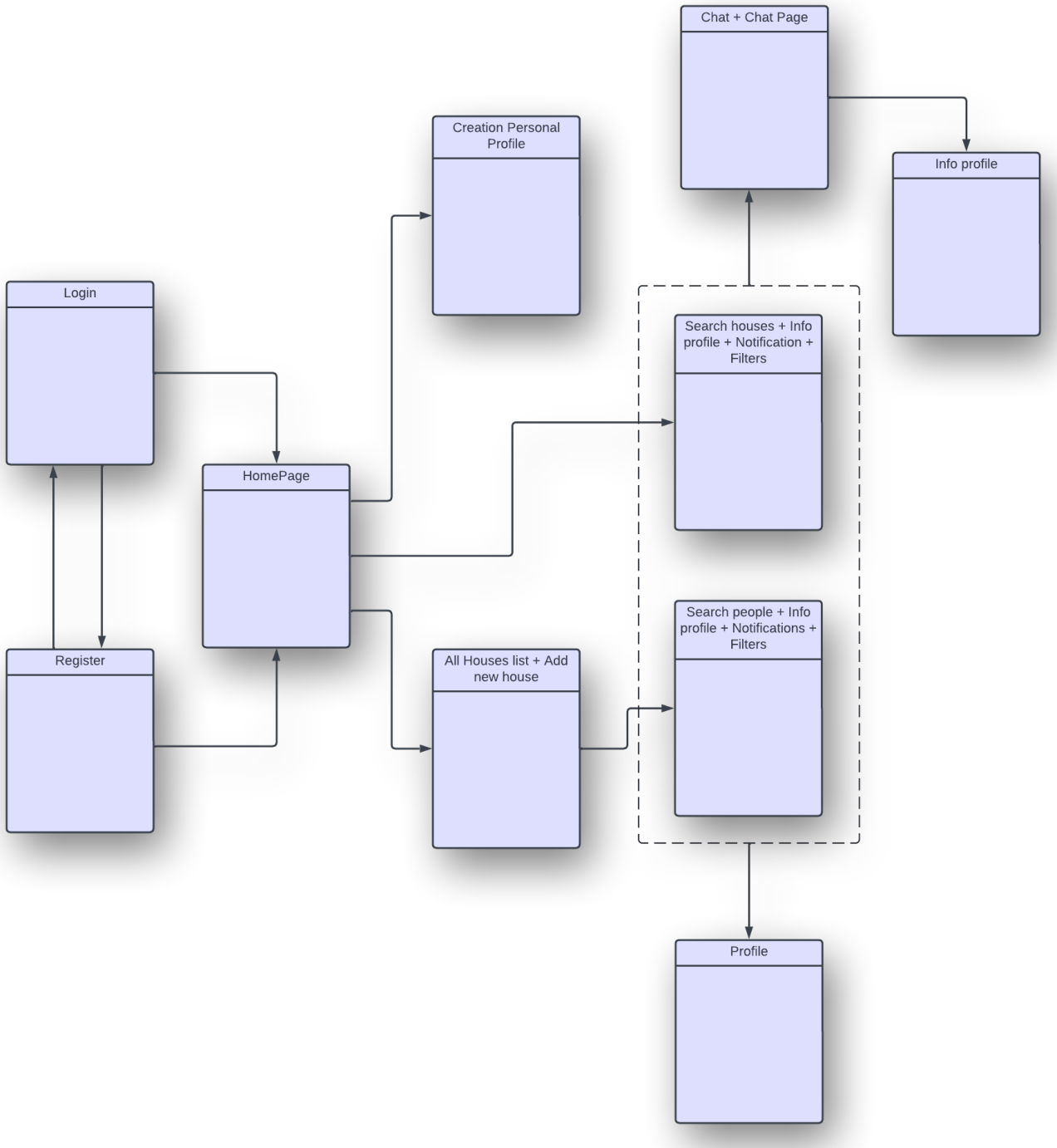
3. User Interface (UI)

In the following two paragraphs, users' experience and screenshots of the different pages are shown.

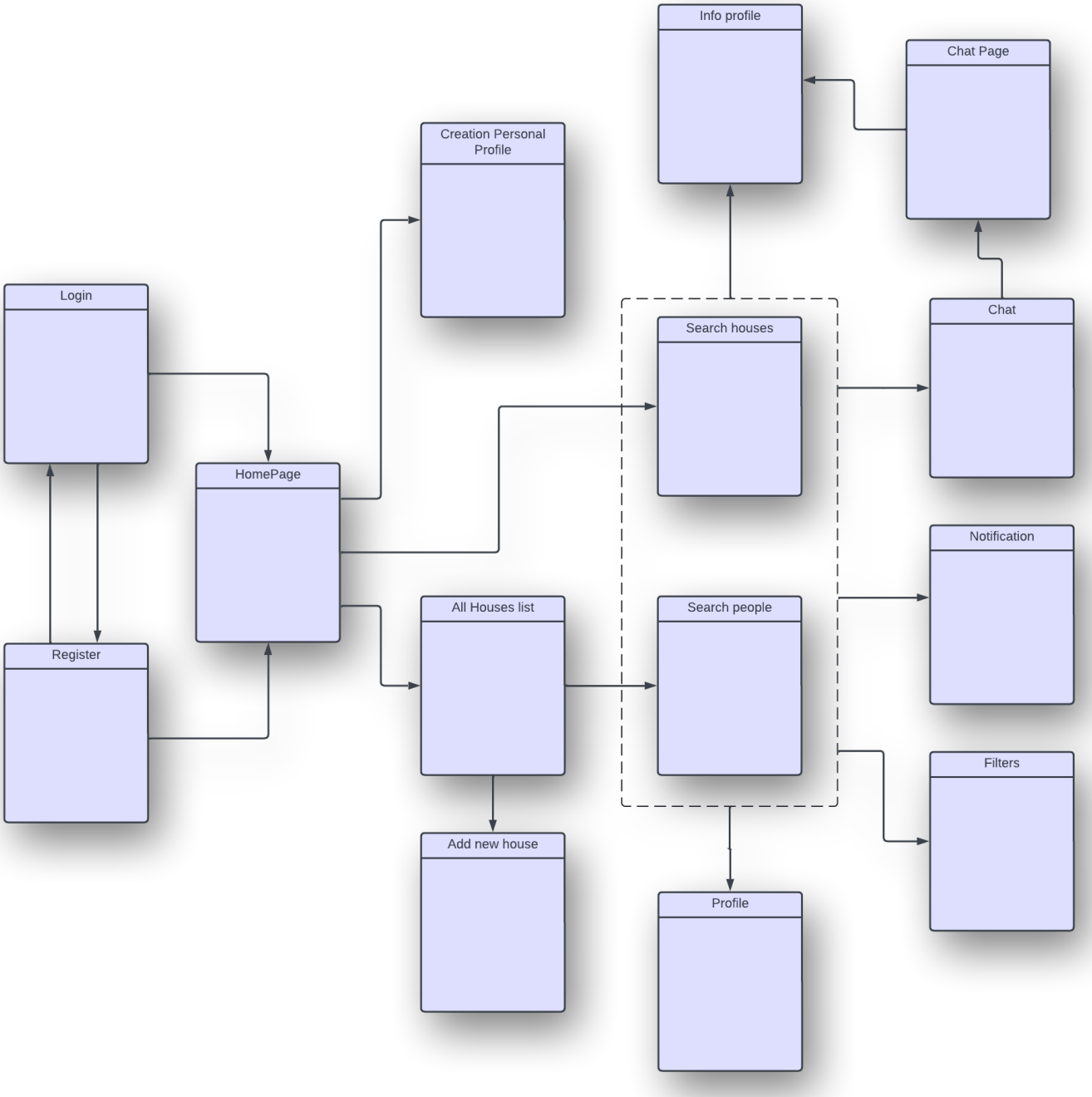
3.1 User Experience (UX)

In the following section, paths that users can follow among different pages are shown both in the case of Tablet User Experience and in the case of Mobile Phone Experience.

3.1.1 Tablet UX

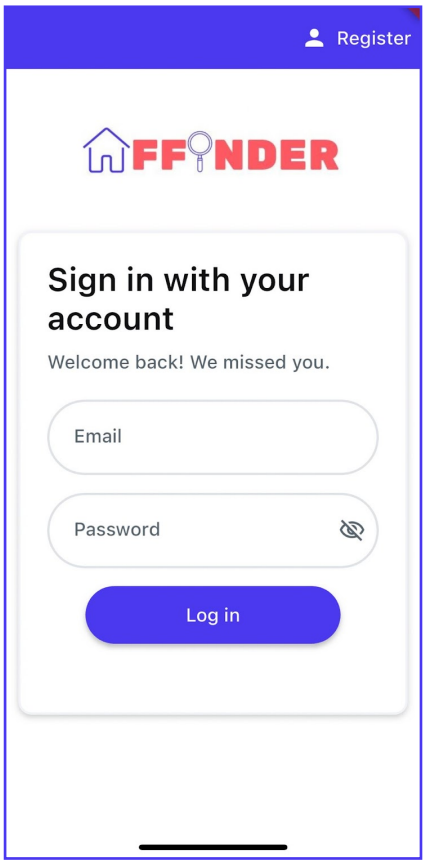


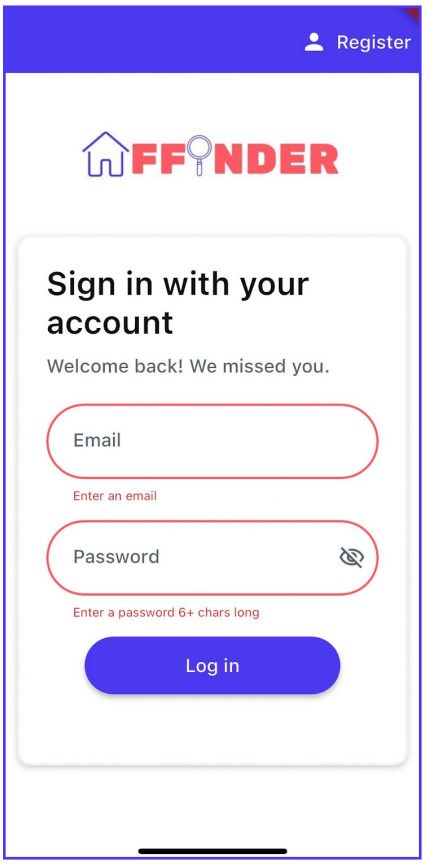
3.1.2 Mobile Phone UX

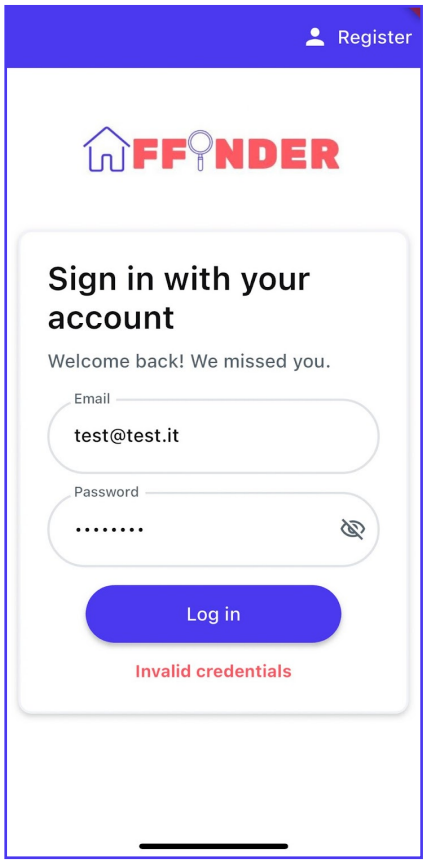


3.2 Application Pages

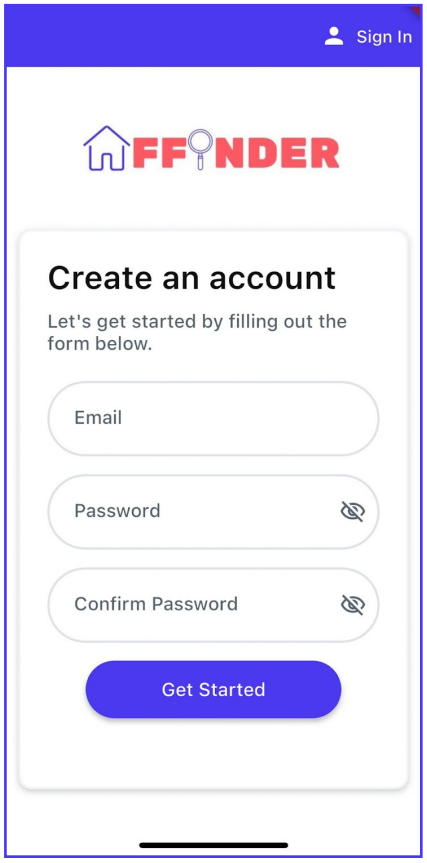
3.2.1 Login

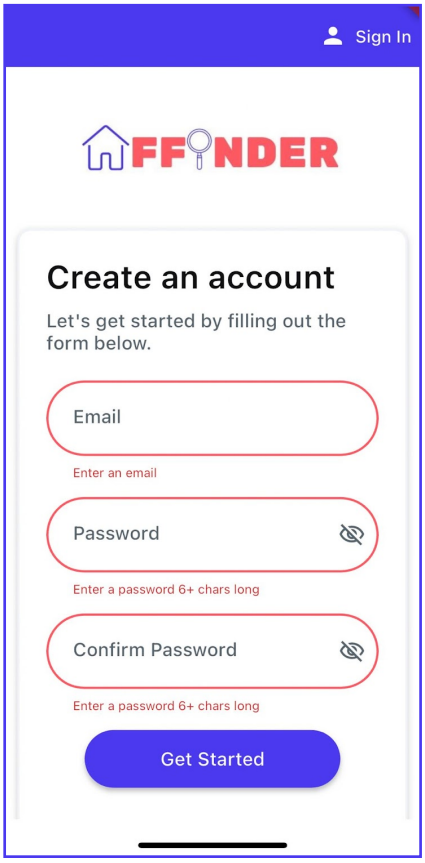


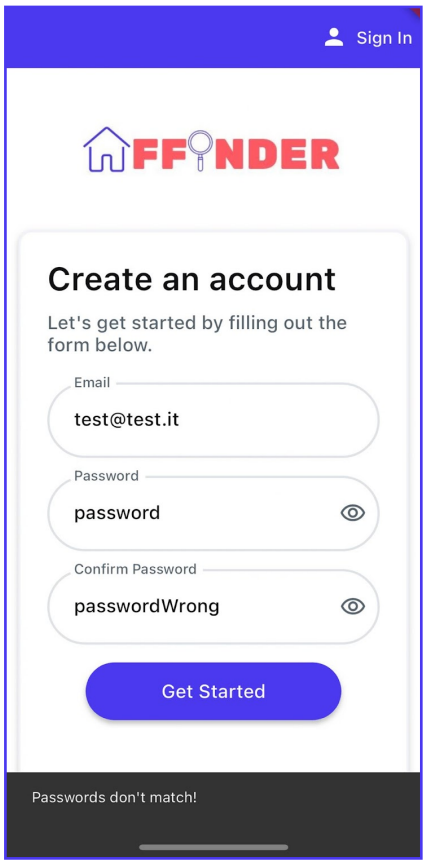


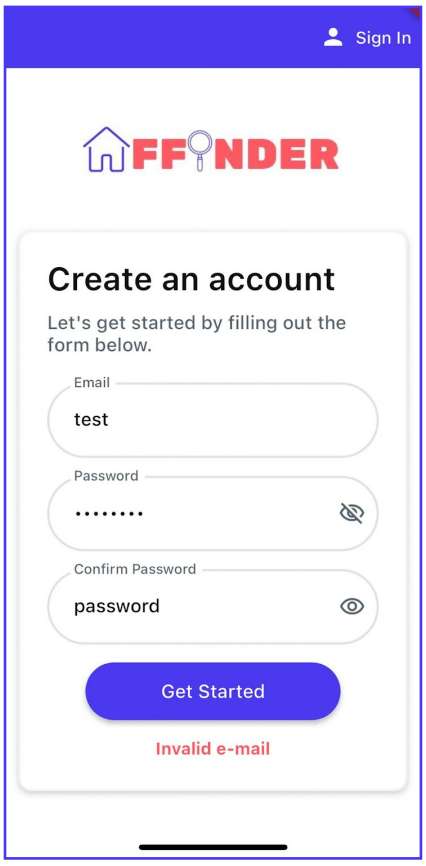


3.2.2 Register

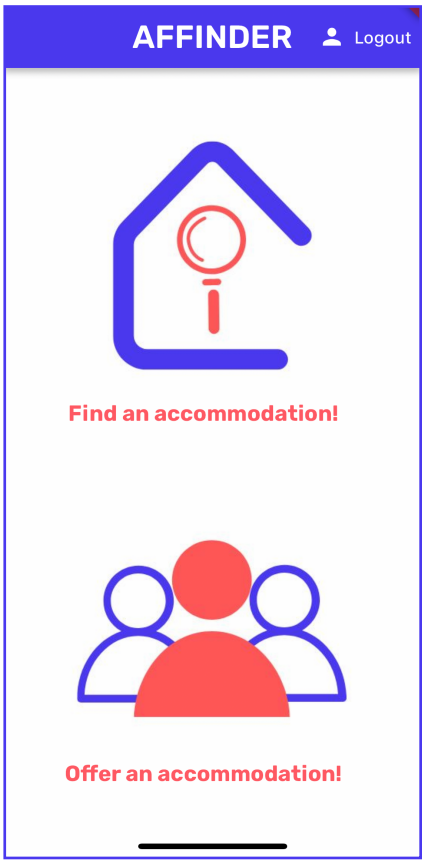


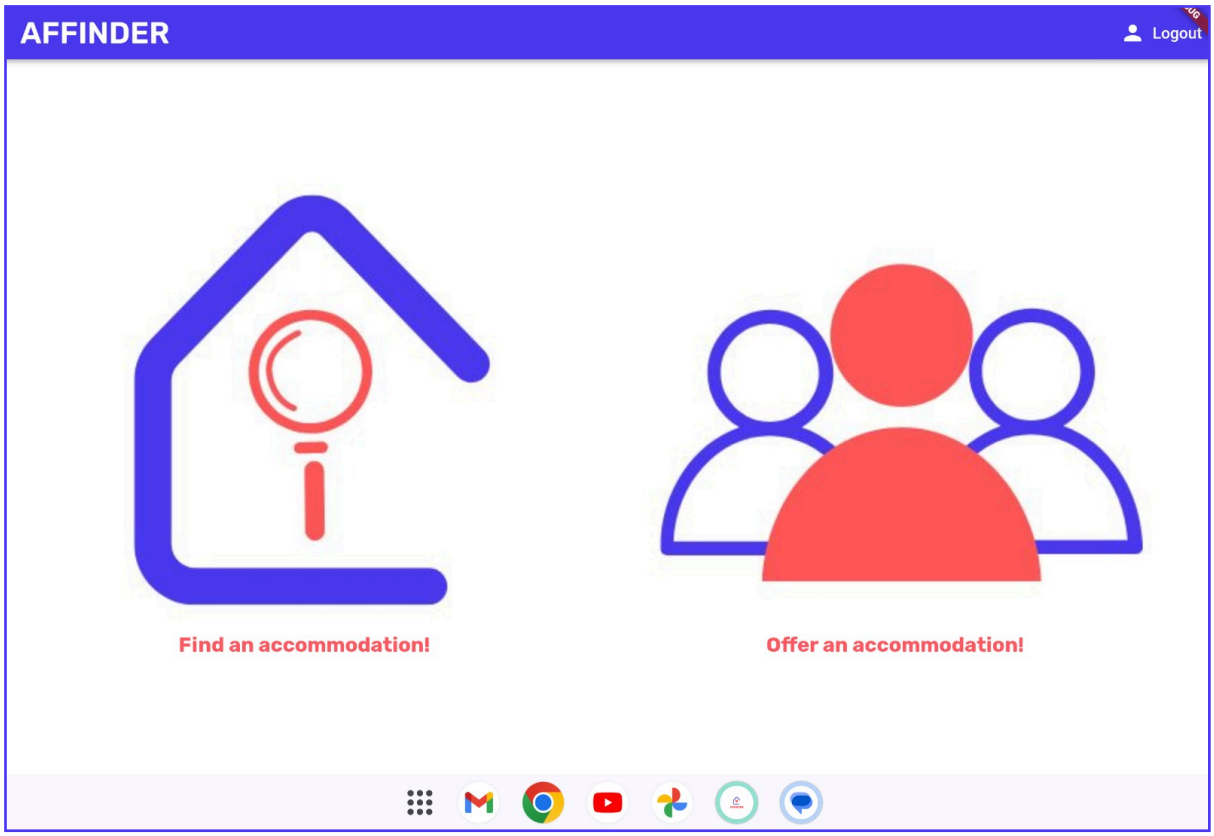




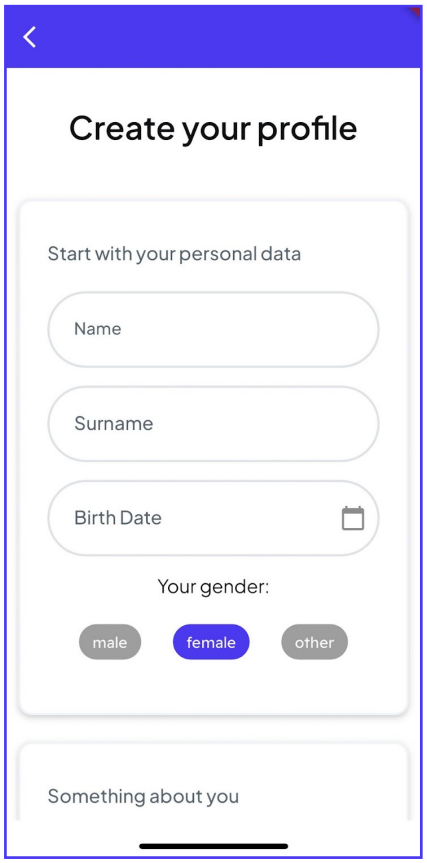


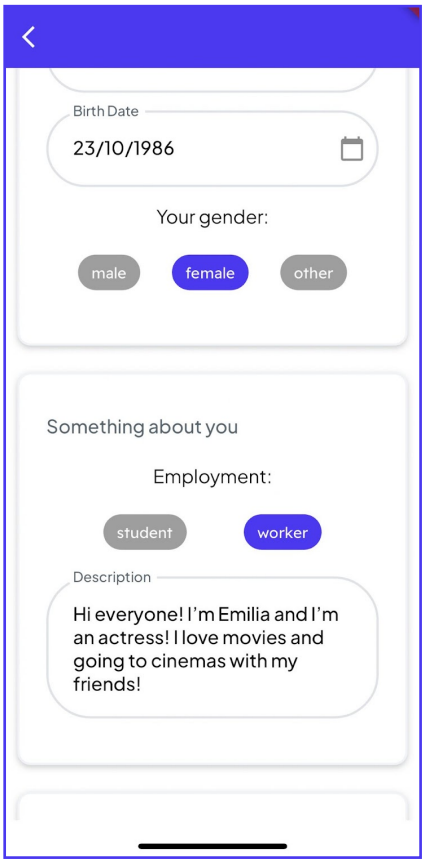
3.2.3 Homepage

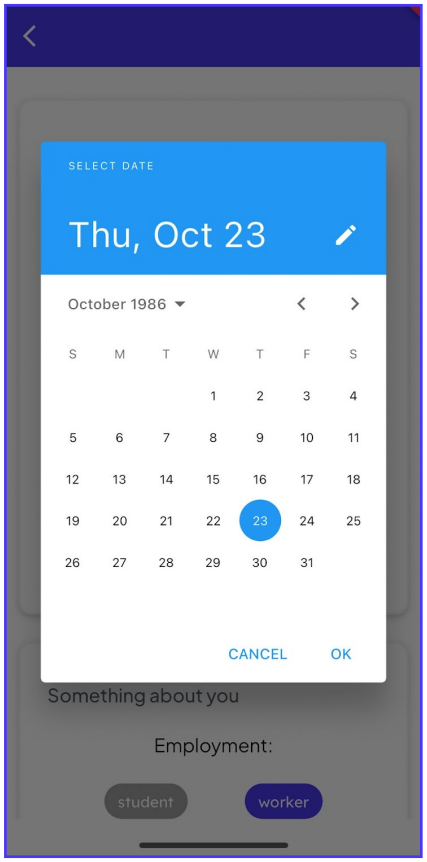


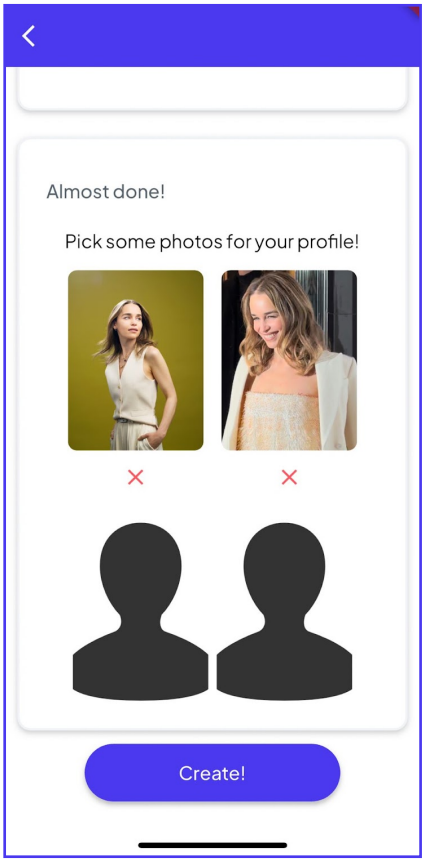


3.2.4 Create Personal Profile

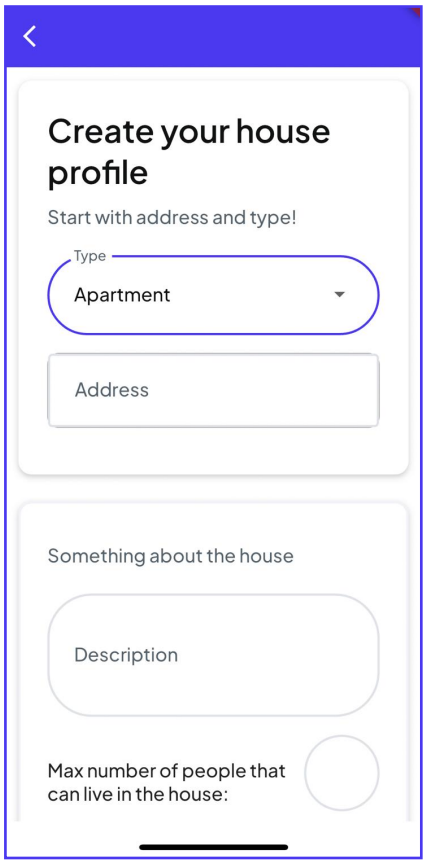


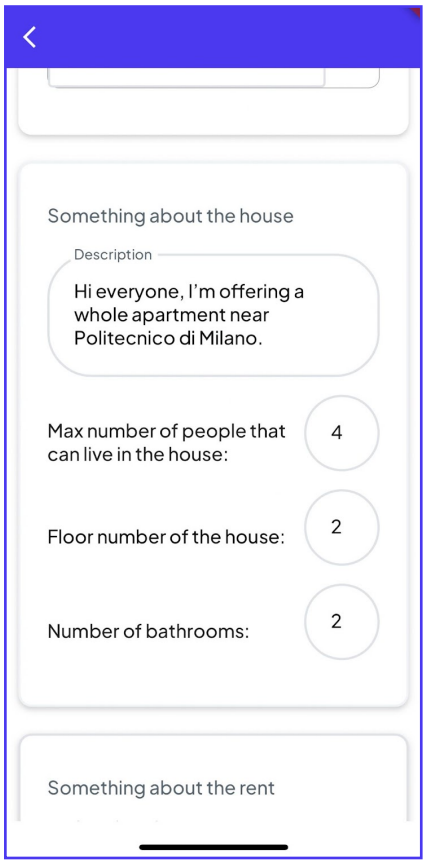


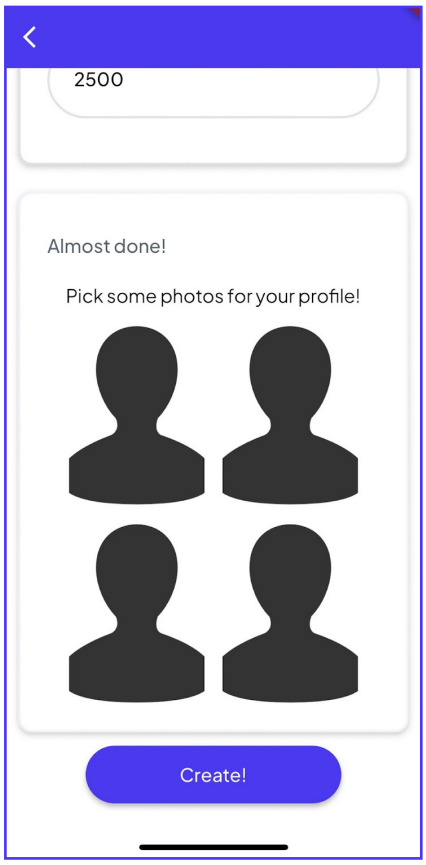




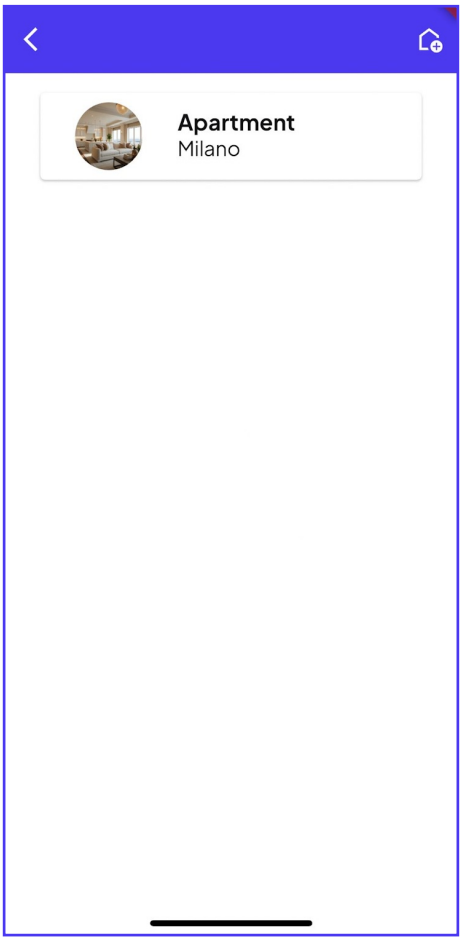
3.2.5 Create House Profile

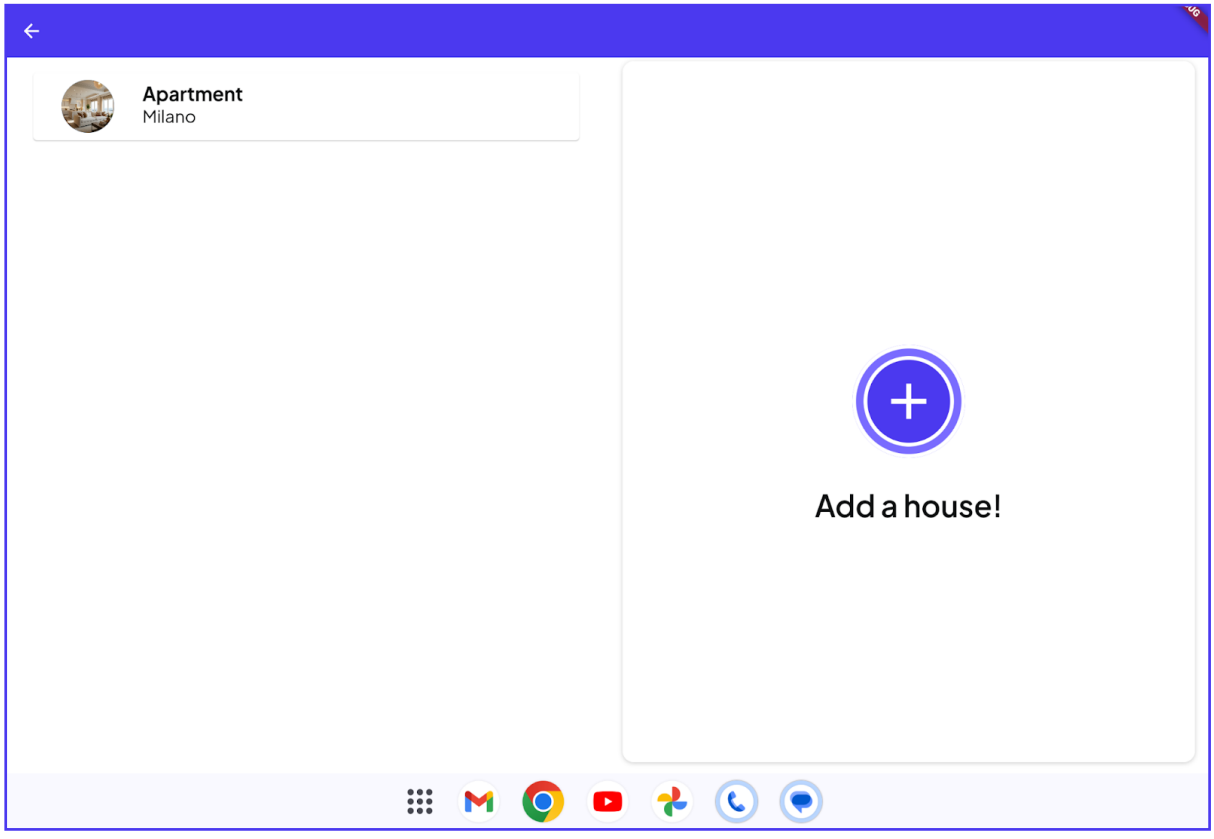




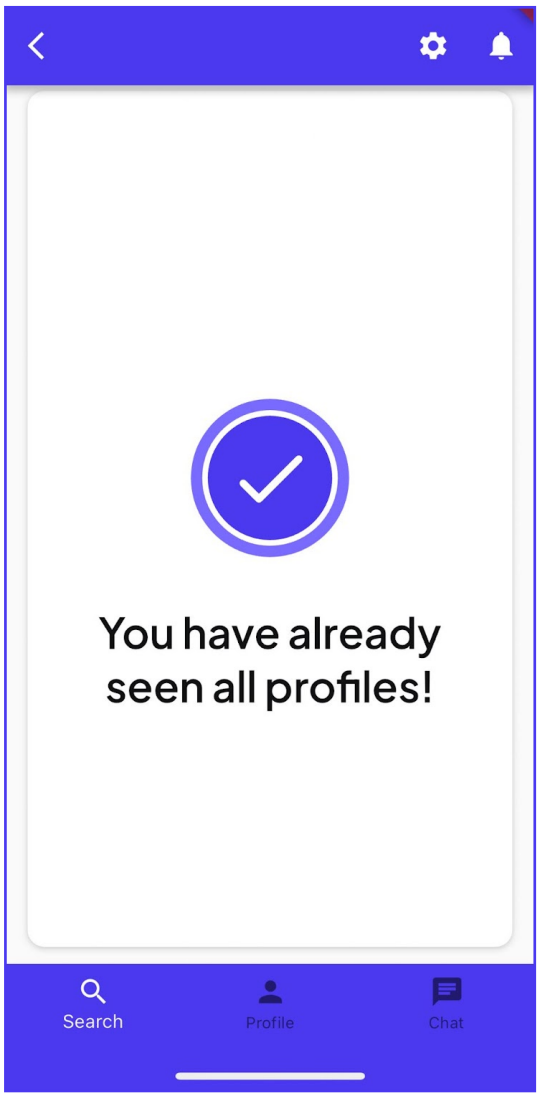


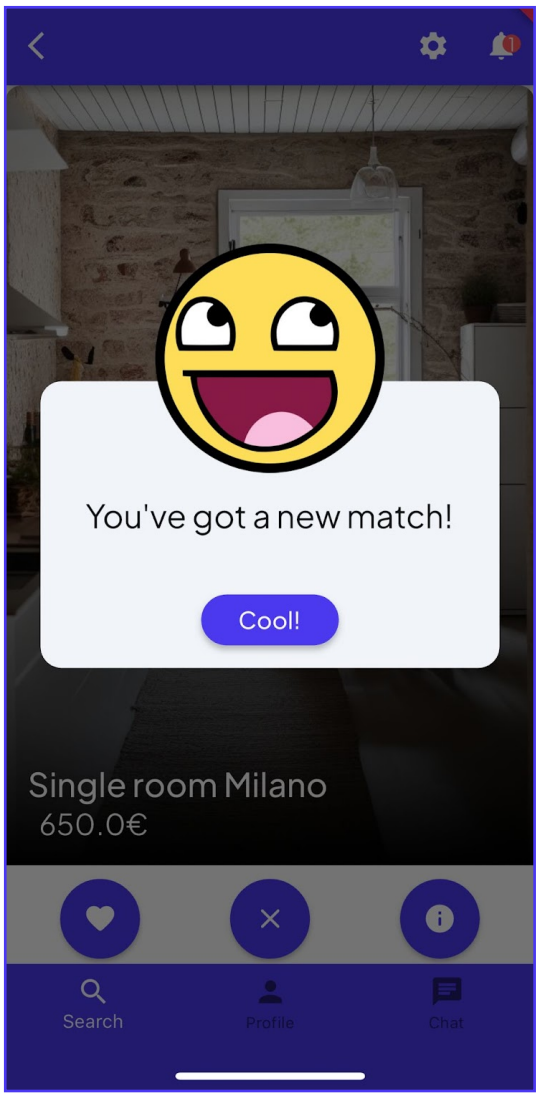
3.2.6 List House Profiles

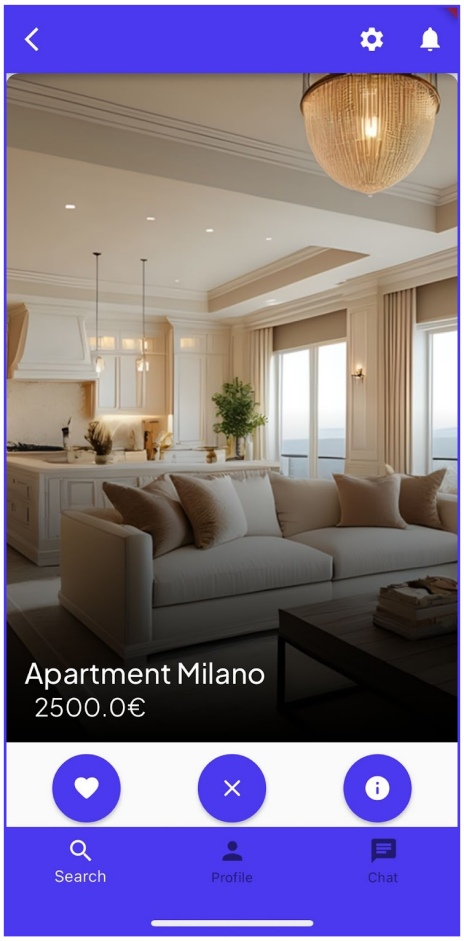


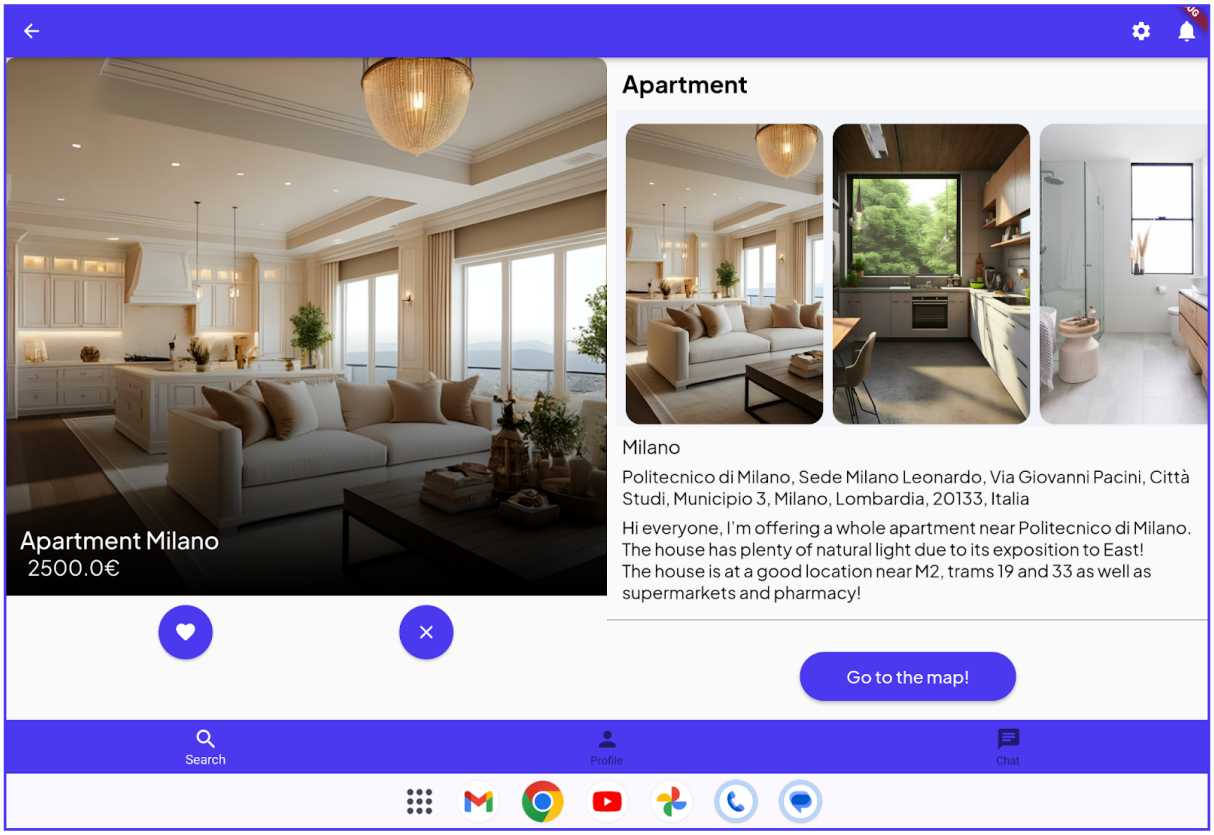


3.2.7 Search

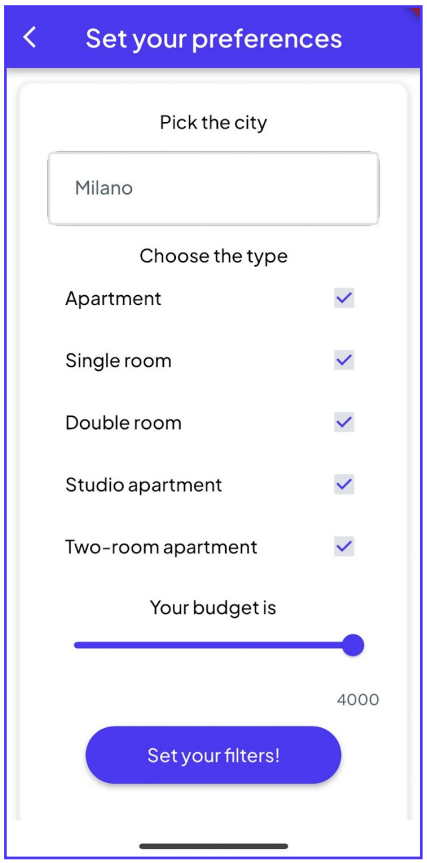

You have already seen all profiles!
Search
Profile
Chat

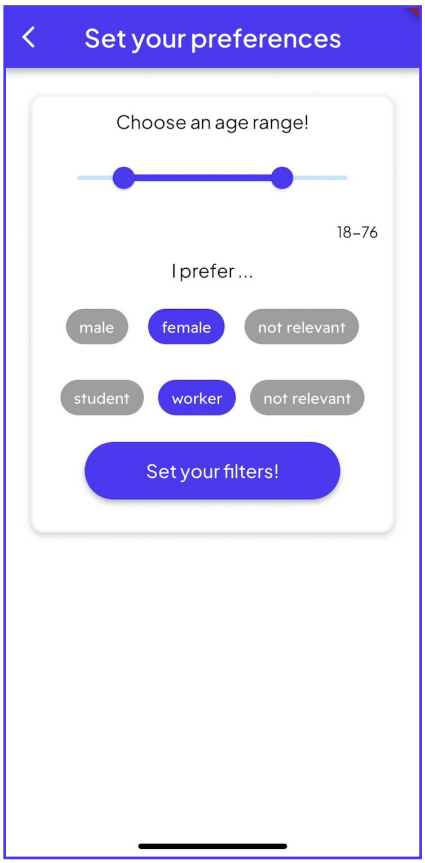

You've got a new match!
Cool!
Single room Milano
650.0€
Search
Profile
Chat

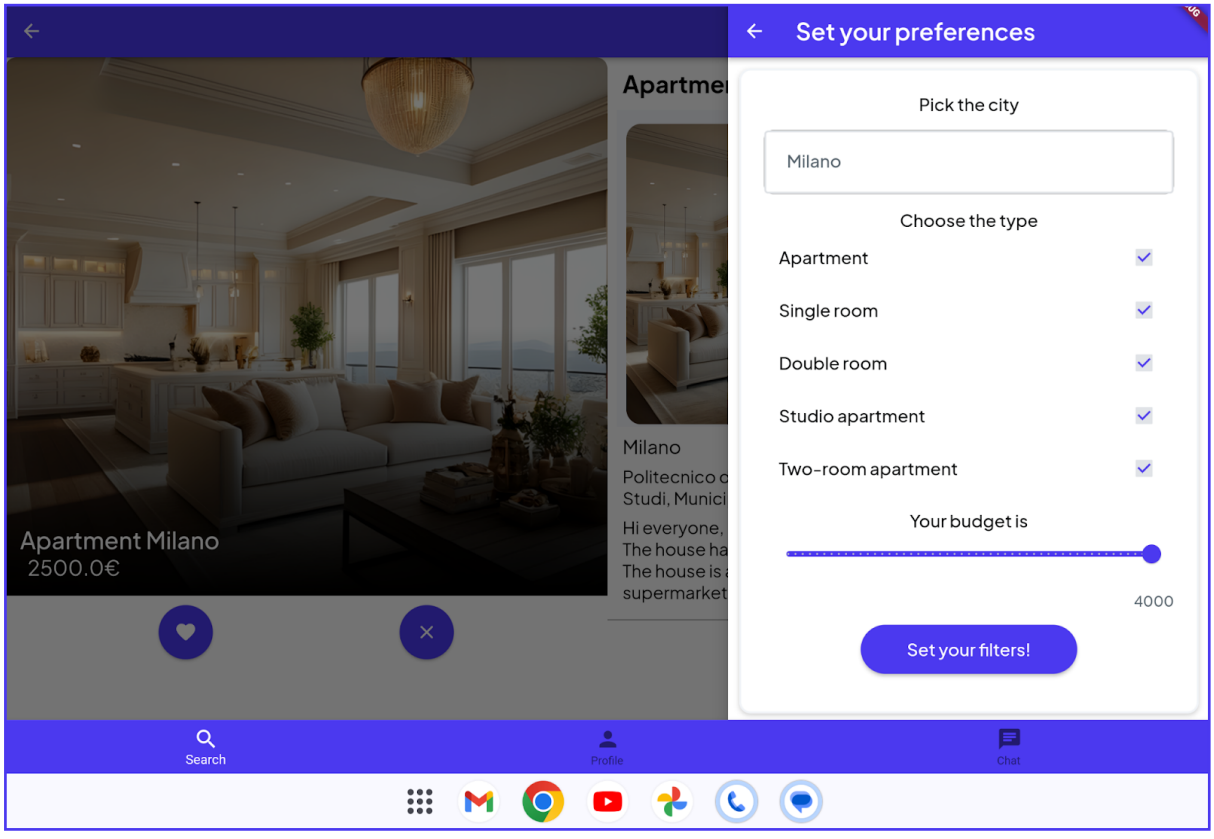

Apartment Milano
2500.0€
Search
Profile
Chat


Apartment Milano
2500.0€
Apartment
Milano
Politecnico di Milano, Sede Milano Leonardo, Via Giovanni Pacini, Città Studi, Municipio 3, Milano, Lombardia, 20133, Italia
Hi everyone, I'm offering a whole apartment near Politecnico di Milano.
The house has plenty of natural light due to its exposition to East!
The house is at a good location near M2, trams 19 and 33 as well as supermarkets and pharmacy!
Go to the map!
Search
Profile
Chat

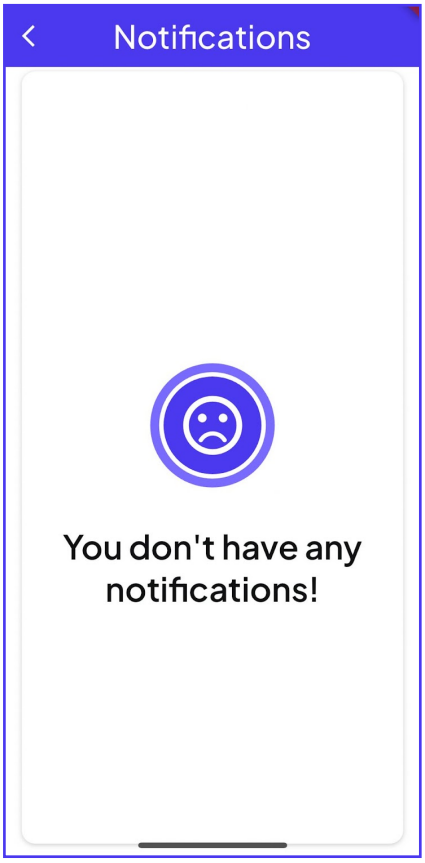
3.2.8 Filters

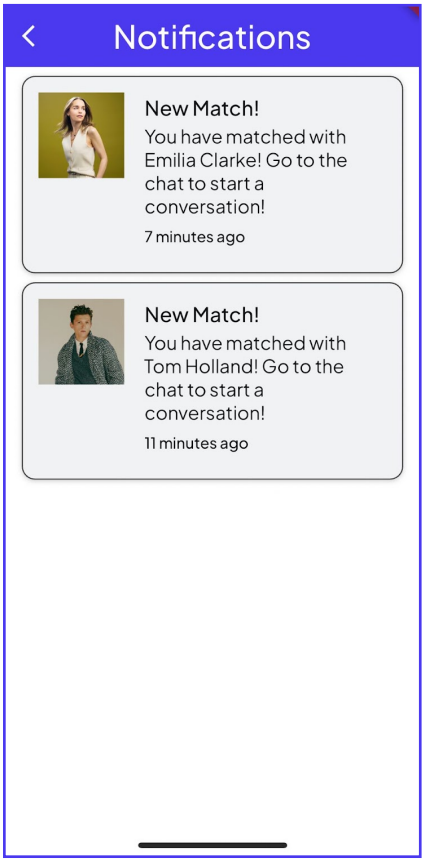

Set your preferences
Pick the city
Milano
Choose the type
Apartment
Single room
Double room
Studio apartment
Two-room apartment
Your budget is
4000
Set your filters!

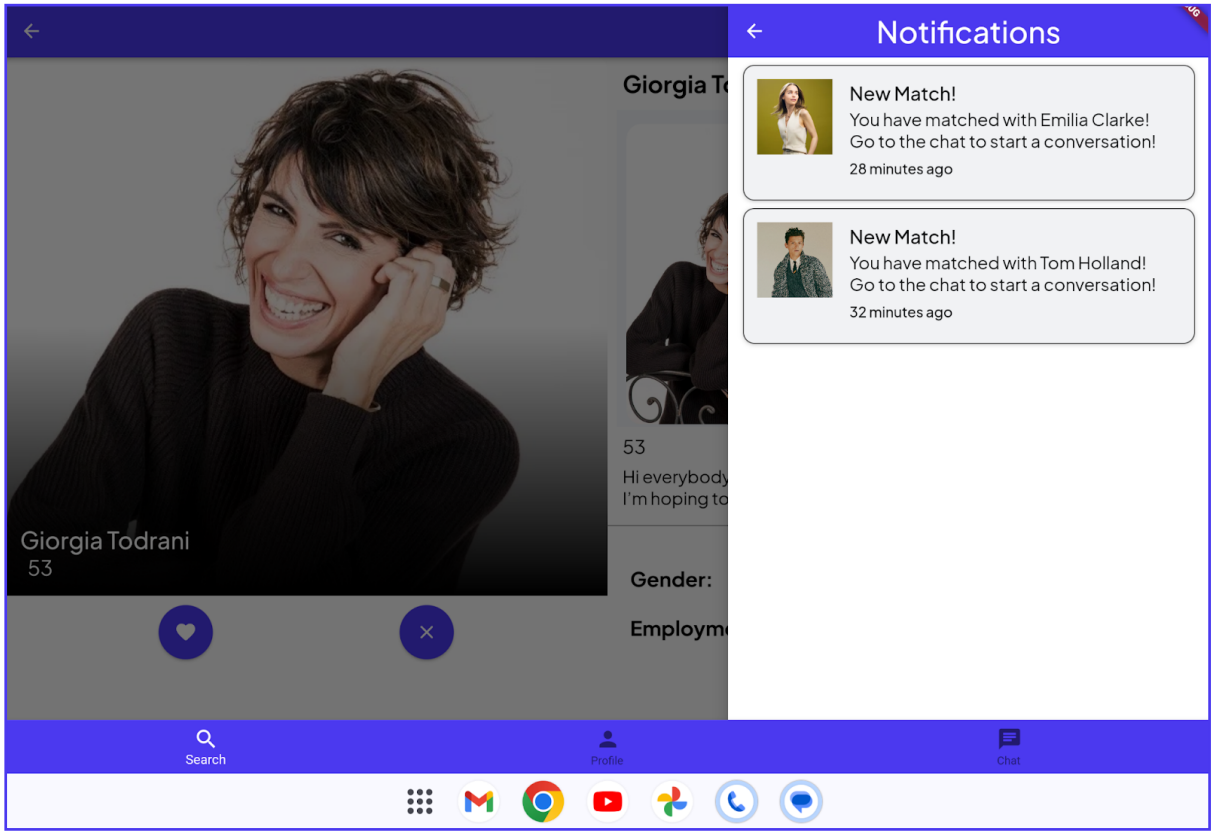

Set your preferences
Choose an age range!
18-76
I prefer ...
male
female
not relevant
student
worker
not relevant
Set your filters!


Apartment Milano
2500.0€
Milano
Hi everyone,
Set your preferences
Pick the city
Milano
Choose the type
Apartment
Single room
Double room
Studio apartment
Two-room apartment
Your budget is
4000
Set your filters!
Search
Profile
Chat

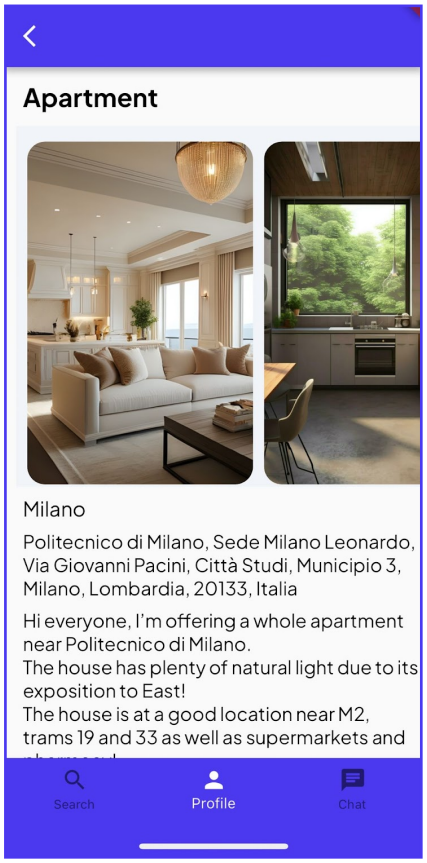
3.2.9 Notifications

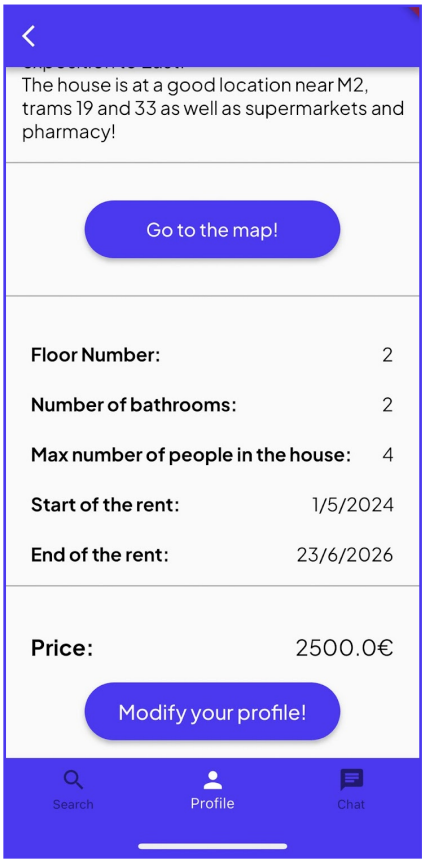




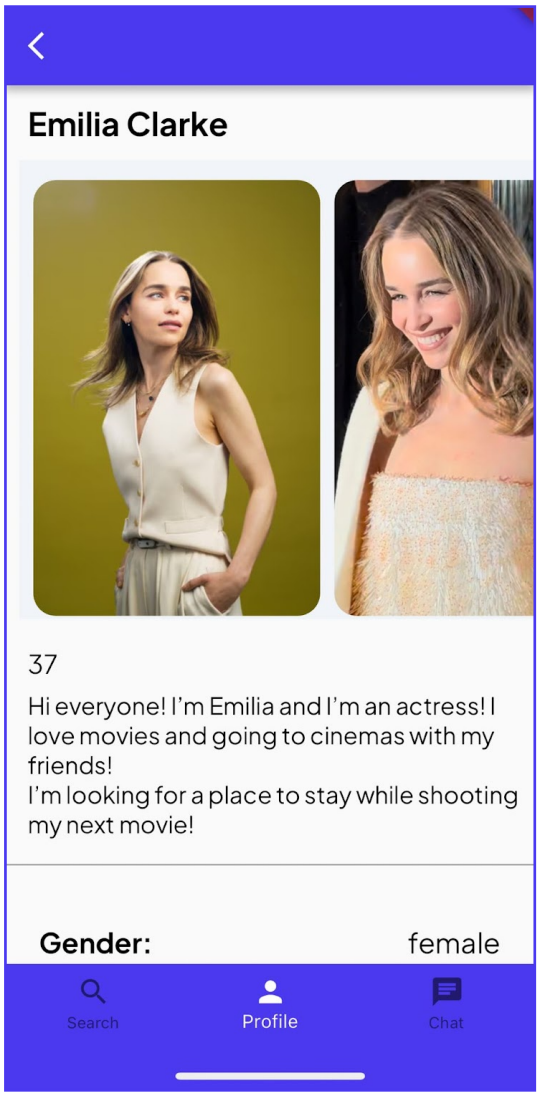


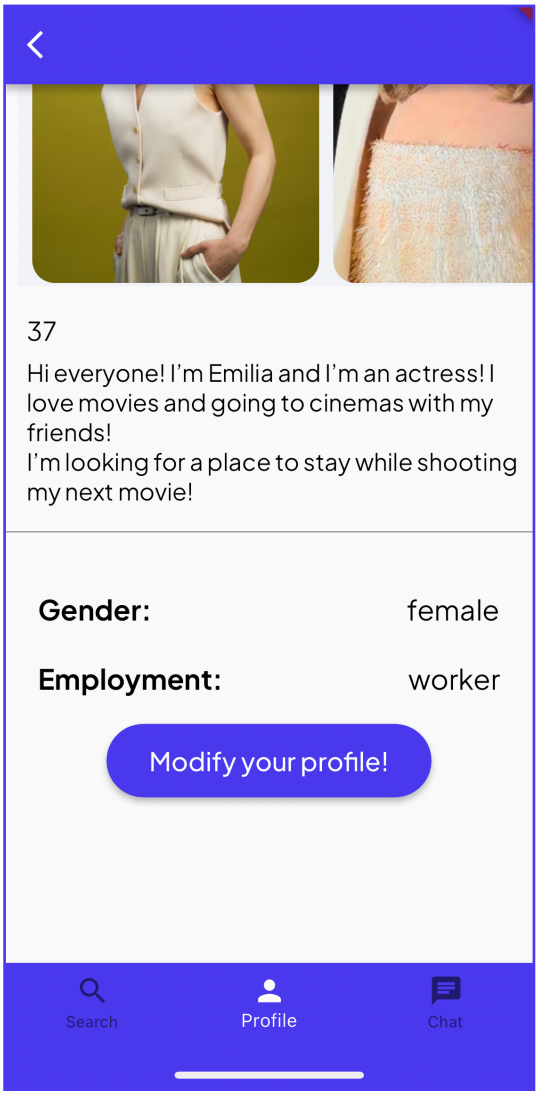
3.2.10 House Profile



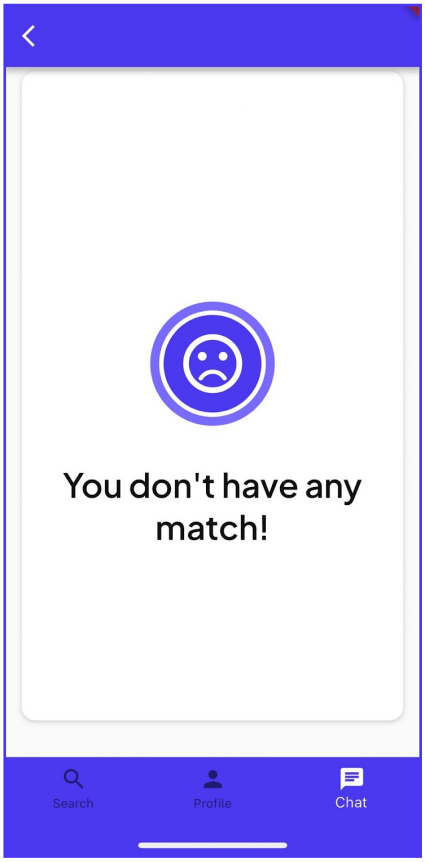


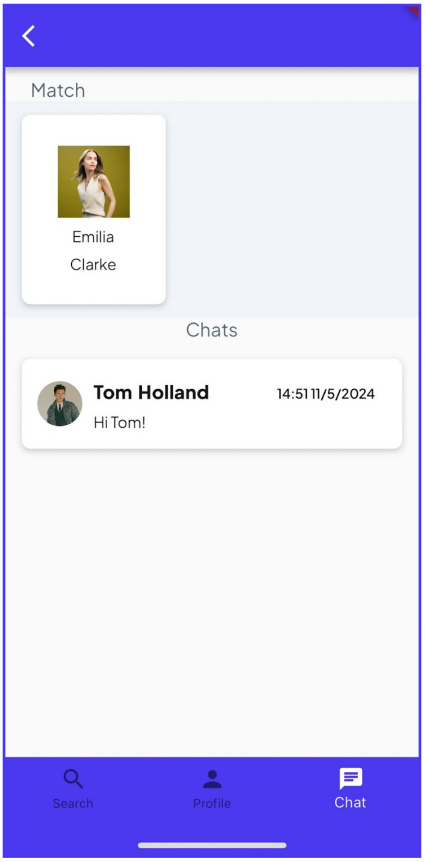
3.2.11 Personal Profile

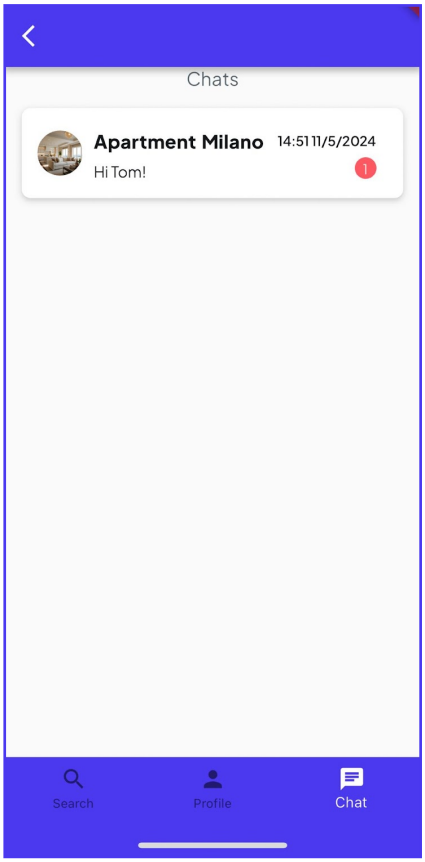


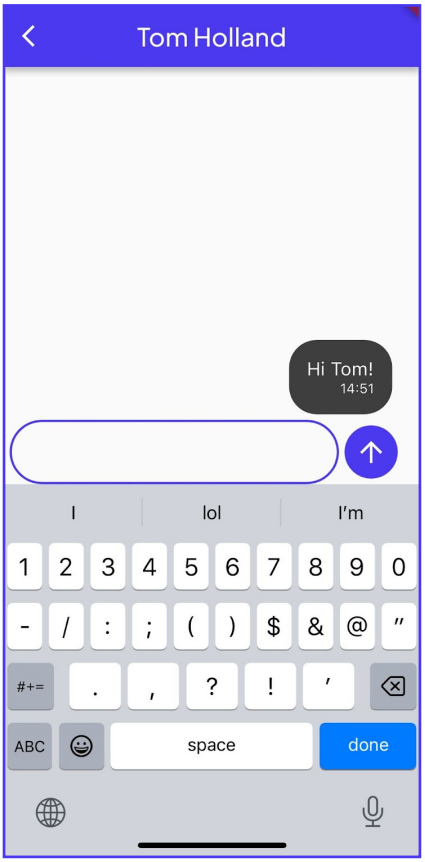


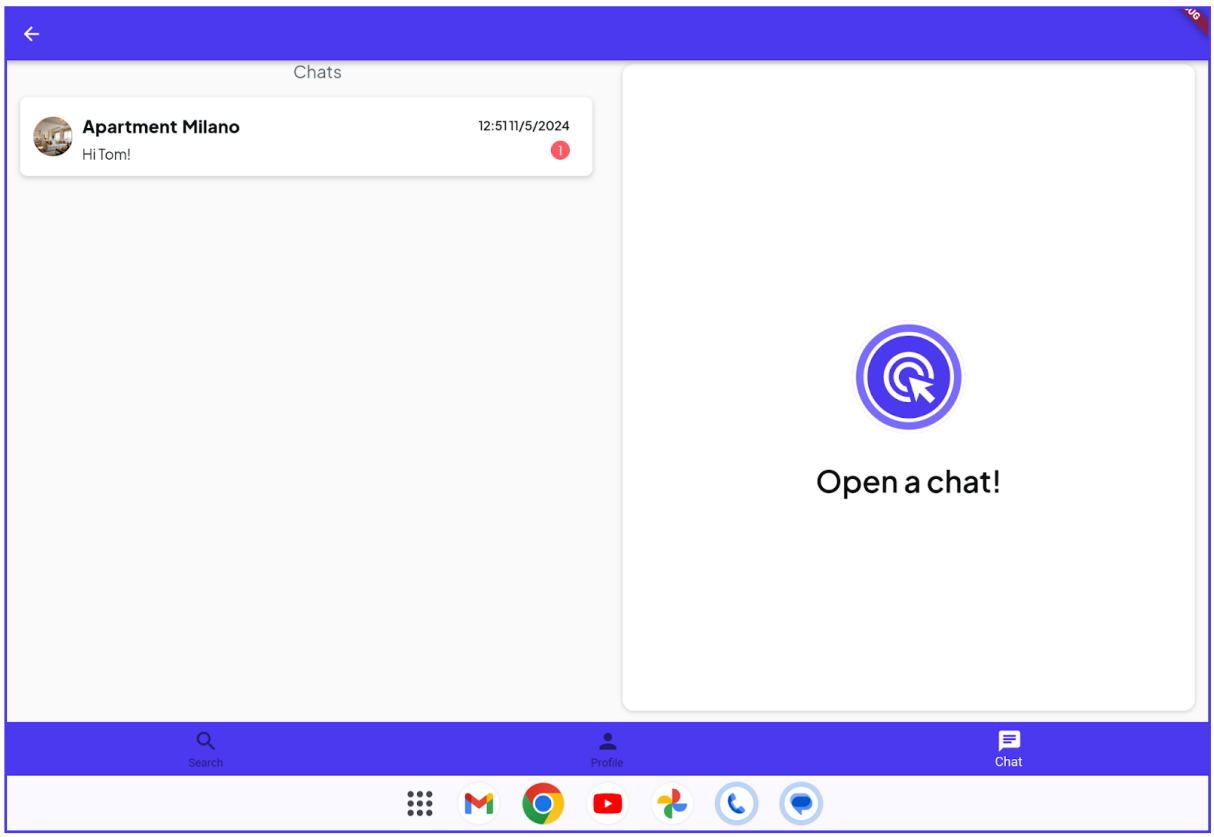
3.2.12 Chat

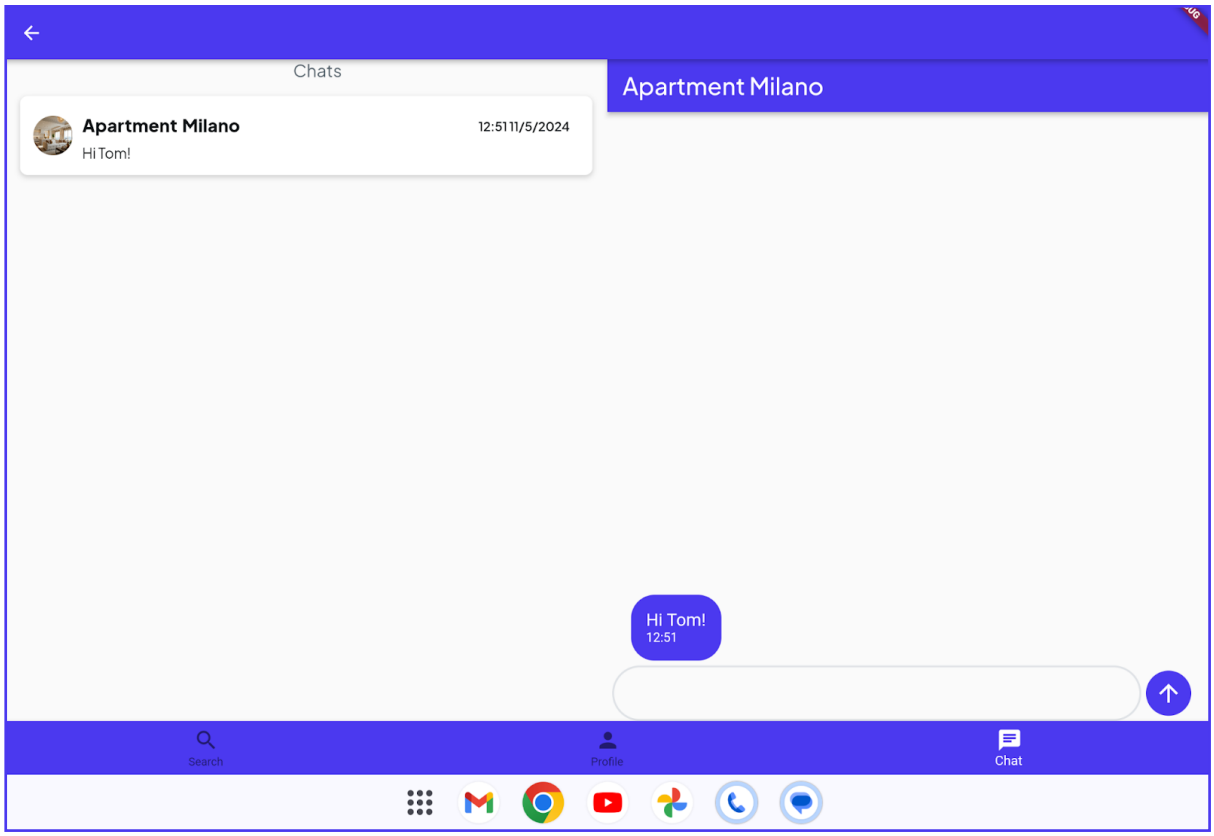

You don't have any match!
Search
Profile
Chat


Match
Emilia Clarke
Chats
Tom Holland
14:51 11/5/2024
Hi Tom!
Search
Profile
Chat


Chats
Apartment Milano
14:51 11/5/2024
Hi Tom!
1
Search
Profile
Chat


Tom Holland
Hi Tom!
14:51
I
lol
I'm
space
done


Chats
Apartment Milano
12:51 11/5/2024
Hi Tom!
1
Open a chat!
Search
Profile
Chat


Chats
Apartment Milano
Apartment Milano
12:51 11/5/2024
Hi Tom!
Hi Tom!
12:51
Search
Profile
Chat

4. Testing

4.1 Testing Environment

Due to the limitations of the flutter test environment, in which external services (e.g. Firebase, Firestore) can't be used, all the tests have been performed by mocking all the needed external functions.

Services related to authentication were simulated by using the Mockito package, which uses mock functions to handle tasks originally handled by Firebase Auth package (e.g. login and registration phases).

All the other external services (in particular those related to queries to the database) have been mocked by using static data given as parameters or hard-coded into the widgets to be tested.

4.2 Unit Test

Flutter unit tests are supposed to run in the mock environment due to flutter constraints. The app heavily depends on external services so most of the logic implemented is a trivial call to them with different parameters. Because of these limitations together with the significative dependence on the external services, no reasonable unit tests were performed.

4.3 Widget Test

Widget tests were performed by considering each screen one at a time in order to check their correct population and the correct functioning of each of widget composing them, both in tablets and mobile phones.

As mentioned before, widget tests rely on mocking all the external services which are not available in the testing environment, hence the tests were run under a mock package which is "alphaTestLib".

In the following table, a list of the tests that have been performed with a short description of their goal.

File	Description	Mobile Phone	Tablet
alphaTestLib/shared/empty.dart	Correct population	-	-
alphaTestLib/shared/loading.dart	Correct population	-	-
alphaTestLib/shared/constant.dart	Correct population	-	-
alphaTestLib/shared/dialog.dart	Correct population	-	-
alphaTestLib/shared/image.dart	Correct population	-	-
alphaTestLib/shared/swipe_between_images.dart	Correct population	-	-

File	Description	Mobile Phone	Tablet
alphaTestLib/screens/home/home.dart	Correct population and correct functioning of buttons	V	V
alphaTestLib/screens/house_profile/all_profile.dart	Correct population and correct functioning of like, dislike and info buttons	V	V
alphaTestLib/screens/house_profile/form_house_filter.dart	Correct population and correct functioning of buttons and choice chips used for setting filters	V	V
alphaTestLib/screens/house_profile/form_house_profile_adj.dart	Correct population and correct functioning of the widgets used, together with a check on their behaviour also on corner cases (e.g. start rental date is after end rental date, profile created without some of the required information and so on)	V	V
alphaTestLib/screens/house_profile/form_modify_house.dart	Correct population and correct functioning of the widgets used	V	V
alphaTestLib/screens/house_profile/homepage_house_profile.dart	Correct population for all of the three possible layouts (search, profile and chat, both in case of non-existing matches and in case of started chats)	V	V
alphaTestLib/screens/house_profile/notification.dart	Correct population, also in case of no notifications	V	V
alphaTestLib/screens/house_profile/show_all_my_house_profile.dart	Correct population and correct functioning of buttons	V	V
alphaTestLib/screens/house_profile/show_detailed_profile.dart	Correct population	V	V
alphaTestLib/screens/house_profile/show_detailed_profile.dart	Correct population and correct functioning of the widgets used, together with a check on their behaviour also on corner cases (e.g. empty input)	V	V
alphaTestLib/screens/login/register_screen.dart	Correct population and correct functioning of the widgets used (e.g. button to choose the visibility of passwords), together with a check on their behaviour also on corner cases (e.g. empty input, mismatch between password and confirm password fields)	V	V
alphaTestLib/screens/personal_profile/all_houses.dart	Correct population and correct functioning of like, dislike and info buttons	V	V

File	Description	Mobile Phone	Tablet
alphaTestLib/screens/personal_profile/form_filter_people_adj.dart	Correct population and correct functioning of checkboxes used for choosing filters	V	V
alphaTestLib/screens/personal_profile/form_personal_profile_adj.dart	Correct population and correct functioning of the widgets used, together with a check on their behaviour also on corner cases (e.g. empty form, missing photos and so on)	V	V
alphaTestLib/screens/personal_profile/modify_personal_profile.dart	Correct population and correct functioning of buttons (e.g. buttons used to remove photos)	V	V
alphaTestLib/screens/personal_profile/notificationPerson.dart	Correct population, also in case of no notifications	V	V
alphaTestLib/screens/personal_profile/show_details_personal_profile.dart	Correct population	V	V
alphaTestLib/show_info_personal_profile.dart	Correct population	V	V
alphaTestLib/screens/personal_profile/user_homepage.dart	Correct population for all of the three possible layouts (search, profile and chat, both in case of non-existing matches and in case of started chats)	V	V

Coverage Analysis

The widget tests that were performed and described in the previous section, have covered an overall significative percentage of the lines of code: 98.5%.

The lines of code that have not been covered by the testing phase were all checked by hand and none of them was crucial for the application to work properly or related to external services and therefore excluded from the testing phase.

Directory	Lines of code	Coverage
- lib	4237/4302	98,5%
- alphaTestLib	4237/4303	98,5%
- screens	4230/4295	98,5%
- home	66/66	100,0%
home.dart	66/66	100,0%
- login	337/343	98,3%
register_screen.dart	191/194	98,5%
login_screen.dart	146/149	98,0%
- chat	53/53	100,0%
chat.dart	53/53	100,0%

			- shared		121/121	100,0%
				constant.dart	18/18	100,0%
				dialog.dart	34/34	100,0%
				empty.dart	37/37	100,0%
				swipe_between_images.dart	17/17	100,0%
				loading.dart	7/7	100,0%
				image.dart	8/8	100,0%
			- personal_profile		1646/1674	98,3%
				user_homepage.dart	275/285	96,5%
				show_info_personal_profile.dart	70/70	100,0%
				all_houses.dart	171/174	98,3%
				form_filter_people_adj.dart	177/178	99,4%
				modify_personal_profile.dart	446/460	97,0%
				notificationPerson.dart	60/60	100,0%
				show_details_personal_profile.dart	70/70	100,0%
				form_personal_profile_adj.dart	377/377	100,0%
			- house_profile		2007/2038	98,5%
				show_all_my_house_profile.dart	62/64	96,9%
				all_profile.dart	169/173	97,7%
				form_house_filter.dart	151/161	93,8%
				form_house_profile_adj.dart	482/482	100,0%
				form_modify_house.dart	555/559	99,3%
				homepage_house_profile.dart	279/290	96,2%
				notification.dart	59/59	100,0%
				show_detailed_profile.dart	130/130	100,0%
				info_house_profile.dart	120/120	100,0%
			- models		7/7	100,0%
				filters.dart	2/2	100,0%
				houseProfile.dart	1/1	100,0%
				message.dart	1/1	100,0%
				personalProfile.dart	1/1	100,0%
				preference.dart	1/1	100,0%
				user.dart	1/1	100,0%

4.4 Integration Test

This kind of testing campaign aims at verifying the correctness of the interaction among widgets.

As for the widget testing phase, a mock environment has been considered during the integration testing phase.

In particular, integration testing focuses on how the application moves among screen whenever the user performs a given action (e.g. taps on a button).

The following table shows what paths have been tested and the user's action that should trigger them.

From	To	When users…
Homepage	User Homepage	Click on "Find an accommodation!"
Homepage	House Homepage	Click on "Offer an accommodation!"
Homepage	Authentication page	Click on "Logout"
Search Layout (for house profiles)	Detailed Personal Profile page	Click on the info button
House Homepage	Notification page	Click on the bell-shaped icon
House Homepage	Filters page	Click on the engine-shaped icon
House Homepage	House Profile page	Click on the button on the navigation bar
House Homepage	Chat Layout	Click on the button on the navigation bar
Chat Layout (both for House Profile and Personal Profile)	Chat page	Click on a match or on a already started chat
Login page	Homepage	Properly fill the login fields and click on the login button
Search Layout (for personal profiles)	Detailed House Profile page	Click on the info button
User Homepage	Notification page	Click on the bell-shaped icon
User Homepage	Filters page	Click on the engine-shaped icon
User Homepage	Modify Personal Profile form	Click on "Update"
User Homepage	Chat Layout	Click on the button on the navigation bar

Just like the widget testing phase, all the paths that were not covered by an integration test were either trivial to test or somehow related to external services, hence they were excluded from the testing phase but still checked by hand during simulations.

4.5 Usability Test

In addition to the already seen test campaigns, the app was also given to a restricted number of people that were asked to freely use it. The goals of this kind of testing campaign were actually two:

1. Receiving feedback regarding the User Interface in order to take them into account when finalising the app, with the purpose of making it the most user-friendly as possible.
2. Receiving reports about possible bugs and/or errors in the app, in order to fix them.
3. Receiving suggestions regarding new potential features to implement.

The Usability Testing campaign was found to be extremely useful because it provided with a different point of view (from an unbiased perception of the product) which disclosed errors that weren't been noticed and that could have affected the correct functioning of the whole app, especially when dealing with corner cases.

It also helped to obtain a final result that is both functional and aesthetically pleasing.

5. Future Developments

During the development of all the phases of tests, but in particular during the usability one, it emerged that the proposed application could be integrated with additional functionalities that may make it more complete and/or versatile, but were not implemented yet due to budget and/or time limitations.

Examples of additional functionalities that may be developed in the future are the following:

1. Availability in other languages different than English;
2. Support for push notifications on real devices;
3. Signing in using Identity Providers (e.g. Google, Facebook,…);
4. Adding functionalities to manage credentials (e.g. update password/email, delete accounts);
5. Adding a layer in the registration process through a confirmation email.