APPartment Design Document

Design and Implementation of Mobile Applications
A.Y. 2021/2022

Authors
Riccardo Nannini
Ercolani Antonio

January 24, 2022

Prof. Luciano Baresi

POLITECNICO
MILANO 1863

Contents

1 Purpose

The purpose of this document is defining the main functionalities and design principles of the *APPapartment* mobile application, starting from the high-level client-server architecture to the detailed user interface. Section 2 gives a general overview of the product and its functionalities. Section 3 is about users characteristics and use cases, where the reader can really understand the purpose and the target of this application. Section 4 is dedicated to the system architecture, with in-dept analysis of how the system has been designed and built and how component cooperate with each other. Capther 5 and 6 are dedicated to the mobile application user interface and to the software testing phase while the last section is about possible developments that future releases of the application might incorporate.

2 Product Functions

The focus of the mobile application is to help people sharing a house or an apartment to deal with the regular problems that dividing a residence often creates. This is done by means of many different functionalities, that are sometimes connected with each other, accounting for the most common tasks that the typical *APPartment* user faces every day. Below the reader can find a list of the developed functionalities:

- **Payment manager** - this functionality allows users to register their payments for common goods/services and check their debts and credits towards the other roommates.
- **Stock management** - it is the place where users can check apartment missing items that needs to be bought. A user can add an item or remove one telling how much he spent for it, in order to let the payment manager update the balance.
- **Announcements** - with which users can write an important message or notes that will be visible to the other roommates.
- **Timetables and Calendar** - allows users to create custom timetables in order to organize and assign periodically repetitive tasks to all the roommates (e.g. define a weekly timetable for taking out the trash or cleaning turns etc.). Users can also create a single event.
  Timetables and special events are added to the Calendar where users have an overview of all the programmed tasks.

- **Notifications** - this particular functionality is basically the home page of the APPartment mobile application. In this screen are showed the last 50 events, chronologically ordered, generated by the above mentioned services. Useful information about the state of the apartment and links to the functionalities are also present.

3 User Characteristics

The user segment targeted by the application is composed by students, young adults and anyone who shares a house with someone else.
The application fits well with users that are not familiar with each others (e.g. students that never met before sharing an apartment) in order to manage almost anything that regards the house maintenance in a schematic and user-defined way, avoiding conflicts or tension around the division of the housework that might arise considering people with different background and habits.
Families that want to improve their house management, divide the housework more fairly or even involve their kids in the house maintenance would benefit from the use of this application.

3.1 Scenario and Use Cases

1. **Housework shifts:**
   Federico and Andrea are two university students that happened to share the same apartment. In order to fairly share the housework in an organized way, they decide to use APPartment's timetable function to create shifts for the common maintenance tasks such as throwing away the garbage, cleaning the various rooms and so on. Once one or more timetable regarding the housework are being created the application takes care of generation the various events in the application's calendar and to alert users when their turn is up.

2. **Shopping list:**
   Riccardo and Elena are a young couple cohabiting for their first time. Whenever they run out of something they use the APPartment's function 'Missing Items' to keep track of what is missing. The next time they will be going to do shopping they will check the list in order to make sure they bought anything they needed. Once everything has been bought again the list can be easily updated and a payment item can be optionally created in the application's payment manager.

3. **Fair payments**
   Three recent graduates are sharing an apartment in Milan while doing their first work experiences. They decided to share evenly every cost related to the house. Thanks to APPartment's payment manager the three guys are able to add any expense they incur to the manager and the application will take care of creating and maintaining a ledger. In

addition to expenses, roommates can also pay debts and make adjustment to said ledger.

4 Design overview

4.1 Overview

The APPartment system follows the usual client-server architecture. In fact, the mobile application installed in the user device communicates with the server that is responsible for the data storage and the majority of the business logic. In particular, APPartment exploits **Google Firebase** services in order to handle user authentication, back-end functions as well as data storage. The following picture presents an high level diagram of the APPartment architecture.

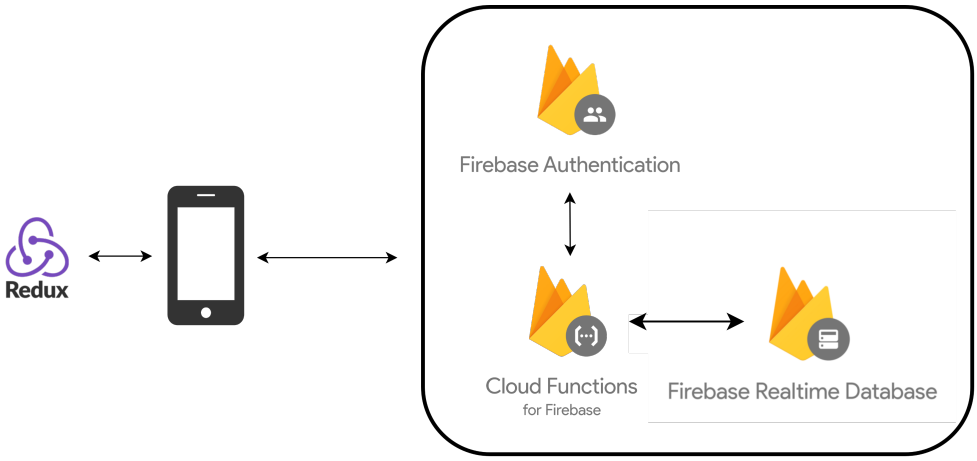


Figure 1: high level architecture

As the picture shows, the **Redux** library has also been used in order to handle the local state of the application and maintain a single local source of truth. Speaking of Firebase, the project utilizes three category of services offered by the Google's product:

- **Firebase authentication**: Firebase own authentication provider
- **Cloud functions**: Function As a Service paradigm used for the application back-end
- **Realtime databse**: NOSQL database used for permanent data storage

4.2 Client application

The APPartment mobile application has been developed using the **React Native** framework. This React-based component framework allows developers to program an application using Javascript plus React Native own components and build it into an Android or iOS app using the exact same code base.
Following an **MVC** pattern, we have decided to use the mobile application primarly as a *View*. As it will be more clear in the following sections, most of the application logic and the permanent data storage resides in the back-end while the front-end (i.e. the mobile application) is used to retrieve, display and allow user manipulation of data.
Speaking of the mobile internals, the application has been developed, apart from login/register and join/create apartment screens, around a bottom *tab navigator* with 4 tabs: **Home tab**, **Services tab**, **Calendar tab** and **Setting tab**.

Figure 2: APPartment bottom tab navigation

Each tab is, in fact, a set of several screens (apart from the Calendar one) and a separate, independently managed screen stack.
The following subsections will explain the details of each tab along with their screens and functionalities. For the user interface of the below described screens, please refer to Chapter 5.2

4.2.1 Home

The Home screen purpose is giving an *overview of the state* of each apartment service and displaying the notification/message produced by said services. It is divided in 2 parts. The top half is composed of four buttons with each of them giving a recap of the status of their corresponding service while also being a shortcut to the service own screen when pressed.
The bottom half of the screen is devoted to a chronologically ordered list of notifications. These notifications are nothing but messages created by the application services (and visible to each apartment's member). For instance, each time a timetable/event is created by an apartment member, a notification for the whole apartment is created. Same goes for payments, addition

Figure 3: Home button example

or removal of missing items, announcements and so on. It is a place where the flow of activities generated by apartment members interaction with the various services is shown.

4.2.2 Services

The Services tab main screen shows a list of the available services offered by the application. At the moment the list comprehends:

- Payments manager
- Timetables
- Stock management
- Announcement

The **Payment manager** main screen allows users to see their current balance as well as the list of every apartment member with information about how much money they are owed/in debt. Two distinct buttons redirect the user to two different screens, one used to generate a new transactions to be added to the payment manager, one to announce a debt pay off between members of the apartment.

The **Timetables** screen make users visualize what timetables/events have been created, by whom and their duration. Users can delete a timetable/event or create new ones, with two distinct dedicated screens for timetables and events. While events, as the word suggest, are single occurrence that last for a day only, timetables let users specify a recurring occurrence and a list of members to be periodically assigned to said occurrence. The feature has been thought with the intent of creating turns for recurring tasks that are needed in the everyday life of house mates (e.g. emptying trash cans, cleaning etc). Timetables are defined, in addition to the list of apartment members that will take part in it, with a start and end date along with a period, the

amount of days after which a the timetable task is reassigned to a member. APPartment sends timetable information to the back-end that is in charge of generating an event each *period* days starting from the start data until the end date while assigning a member to each occurrence. The back-end will then store in the database the generated events, making them visible in the calendar screen of the application.

The **Stock management** service main screen allows users to visualize missing items that have been added by some apartment members. Another screen, accessible from the main one, allows users to add missing items while a second one allows users to remove them once they have been restocked. When removing a missing item, the application asks the user wether it want to add a payment to the payment manager related to the restocked item or not. If the user selects the positive option, it is redirected to the payment manager's new payment screen where the new disbursement can be added.

The **Announcements** screen let users visualized apartment's announcement create by members, create new ones as well as remove them. Announcement are in chronological order, from the latest to the oldest, and can also be visualized in the Home notification section.

4.2.3 Calendar

The Calendar tab is composed by the a calendar component. This agenda-style calendar displays every apartment event available in the database in its corresponding day. It is possible to navigate the calendar in order to watch events of future/past days and weeks.

4.2.4 Settings

The Setting tab contains a list of management services that are used to visualize/change information about the user account and the apartment. In particular, the user can see its username and email, modify its account's password and sign out.
Information about the apartment are also shown: users can find here the access code, a 6 characters code that is required for the new users to join the apartment, the list of apartment members and have also the possibility to 'lock' the apartment, denying the entry of new users into the apartment even if they know the access code.

4.2.5 Component diagram

The following pictures illustrate the mobile application component diagram. Note that it comprehends the 'Home module' only i.e. the part of the application that is involved once the user has already signed in/up and created/joined an apartment. A component diagram regarding the login/registration and creation/joining of an apartment was not produced due to the 'linear' and simple structure of that part of the application consisting in just a sequence of different screens without complex subsystems and modules.

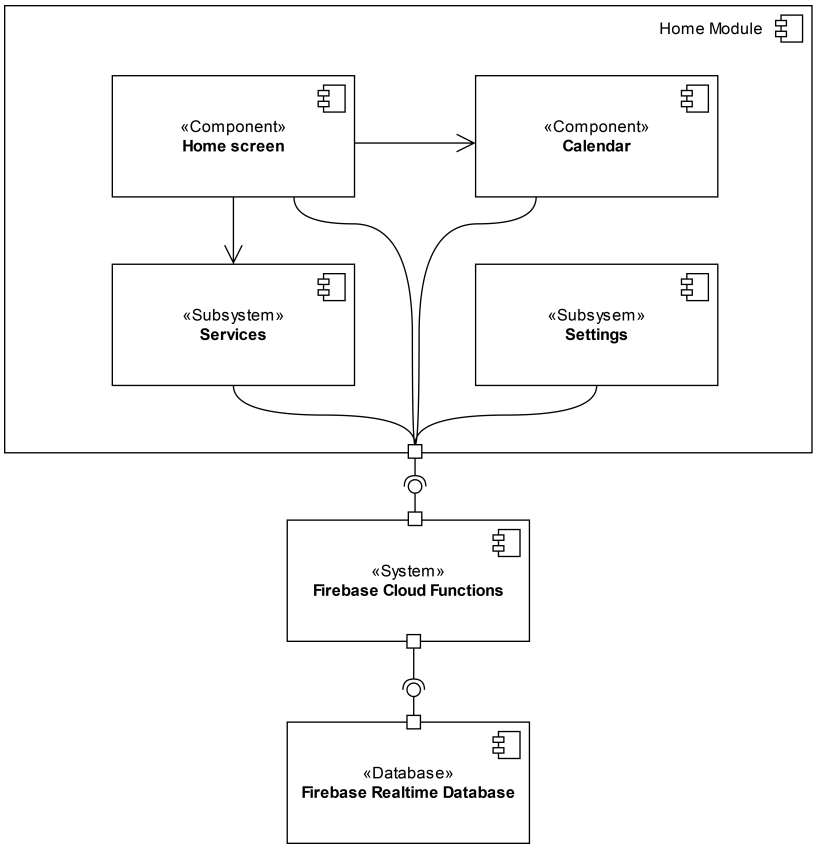


Figure 4: General component diagram

The Service and Setting subsystems are illustrated in great details in the following images. Keep in mind that each of these components interacts with the application back-end although the subsystem diagram doesn't show a detail of it. This has been done to avoid making the diagram excessively intricate and difficult to understand.

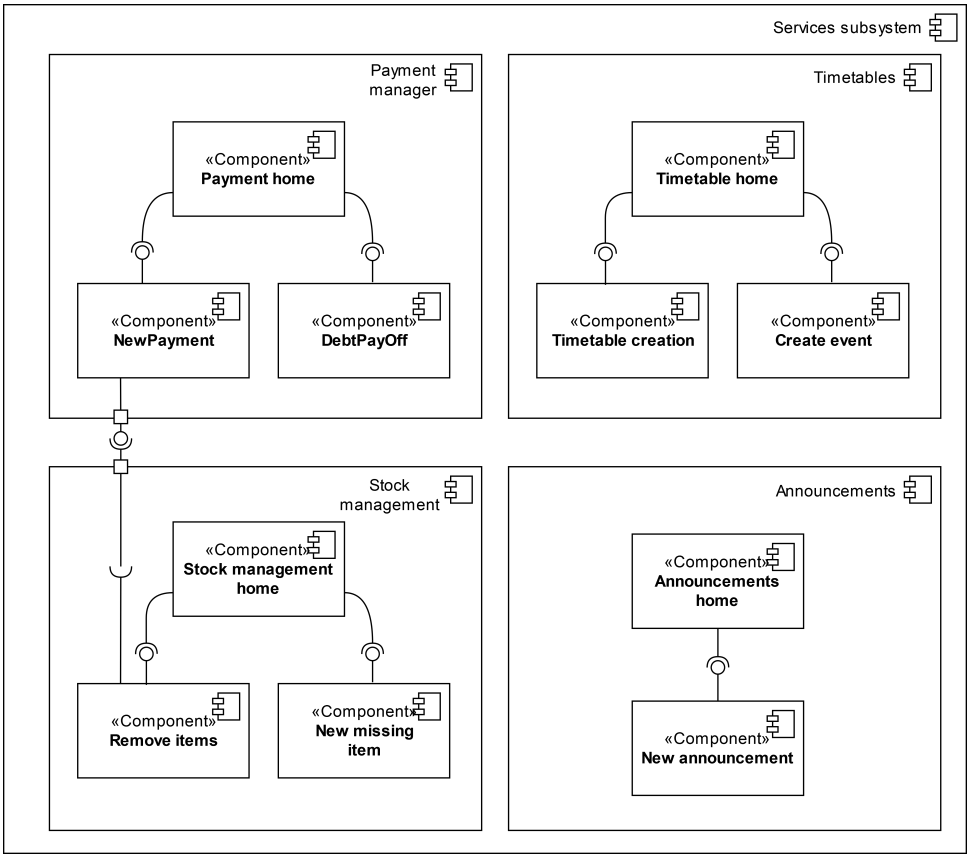


Figure 5: Services subsystem

4.3 Firebase authentication

APPartment requires users to register in order to work. This is a necessity since each apartment created in the application is nothing but a logical group of users and some common data. Therefore the context requires user **authentication** and **authorization** in order to properly function.
APPartment uses Firebase powered authentication service which is able to interact with the entire Firebase ecosystem, easing the authorization process (e.g. a Firebase Cloud function is able to authorize data access based on the user id).
The application provides a simple email and password login and requires no other data (except for a username) in order to register a new user. In addition, once a user performs the login into the application, the system will remember the logged user and will not require future logins unless the user manually signs out or the password changes.

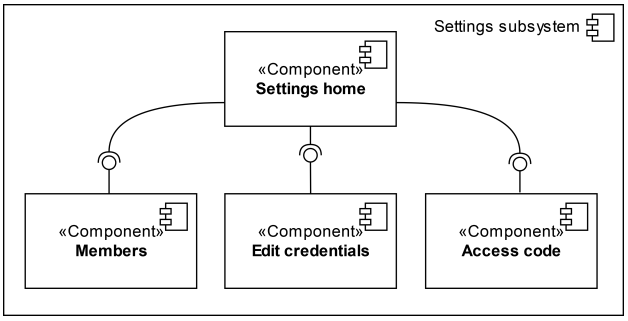


Figure 6: Setting subsystem

4.4 Cloud Functions

The application back-end is provided through the Firebase own **Function as a Service** paradigm using Firebase Cloud functions.
The advantages of this approach are that there is no need to setup and maintain a web server (either on premise or hosted) because everything is handled by the Cloud provider. The developers work can only focus on the back-end logic while Firebase takes care of making it online and accessible through the internet.
The Cloud Functions service is composed of **functions** (in the computer science sense of the word) that are executed in the Google's servers and are invoked by HTTP(S) requests. Single functions can be dynamically added and removed without impacting the ones already present and running. The service also allows the creation of trigger-like functions that run only once data gets modified in a particular section of the data store.
We exploit the service for many purposes, from managing access to the database in order to perform the typical CRUD operations, to background manipulation of said data and for executing tasks with a high computational demand that could affect the user experience and the user devices.
In fact, the call and the execution of the cloud functions are always done **asynchronously**; screens are rendered immediately and the data returned by the functions is displayed only once it is ready.

4.4.1 Back-end API

The back-end is logically divided in five modules with each one exposing different API to the client application. One module, **Apartment**, is designed to offer general functionalities and utilities regarding apartments while the remaining four (**Announcements**, **Payments**, **Stock Management** and **Timetables**) offer back-end functions to their corresponding service. The

following diagram depicts the APIs in detail.

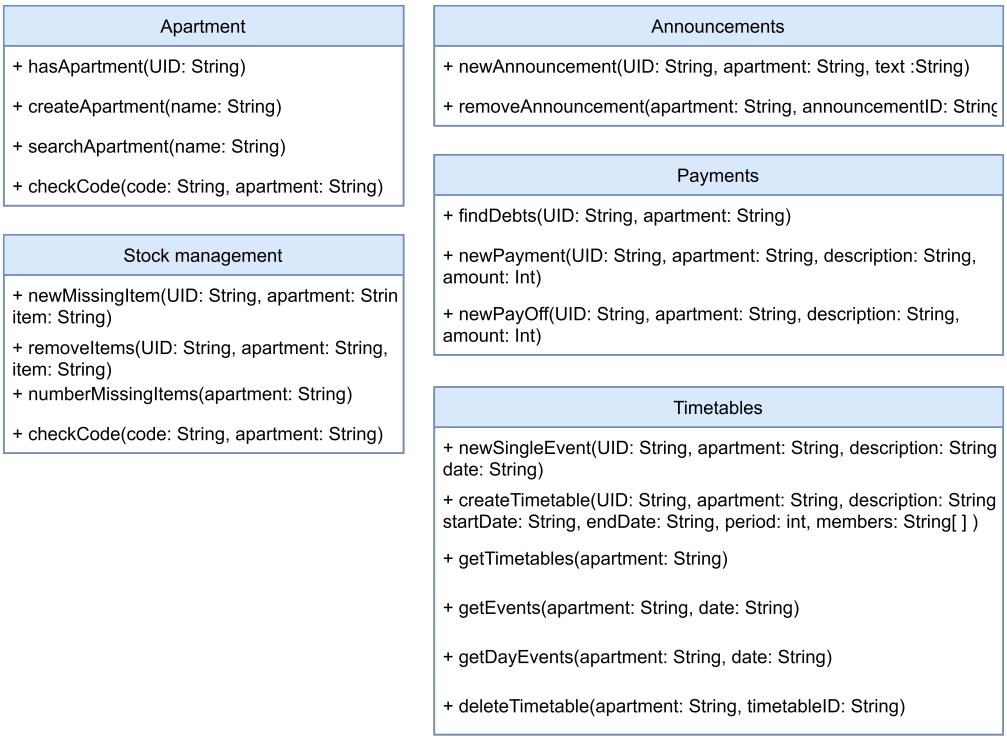


Figure 7: Back-end API

In addition, a **trigger** function exists for each service in order to generate notifications as soon as a new operation on the database are detected.

4.5 Realtime Database

The application employs a Realtime Database powered by the Google Firebase infrastructure. Realtime Database is a **cloud-hosted NOSQL database** where data is stored in the form of a JSON object. Unlike a relational database, there are no table or records; when you add data to the JSON tree it becomes a node of the existing JSON structure.

4.5.1 Database structure

The figure below shows the structure of the APPartment database. Note that it is a best practice, when working with this type of database, to *flatten* the data structure as much as possible, at the cost of denormalizing the

database. This is due to the implementation of this kind of database that, when retrieving a node, retrieves all its subtree, possibly generating a huge traffic of data where most of it might not be needed in that particular query.

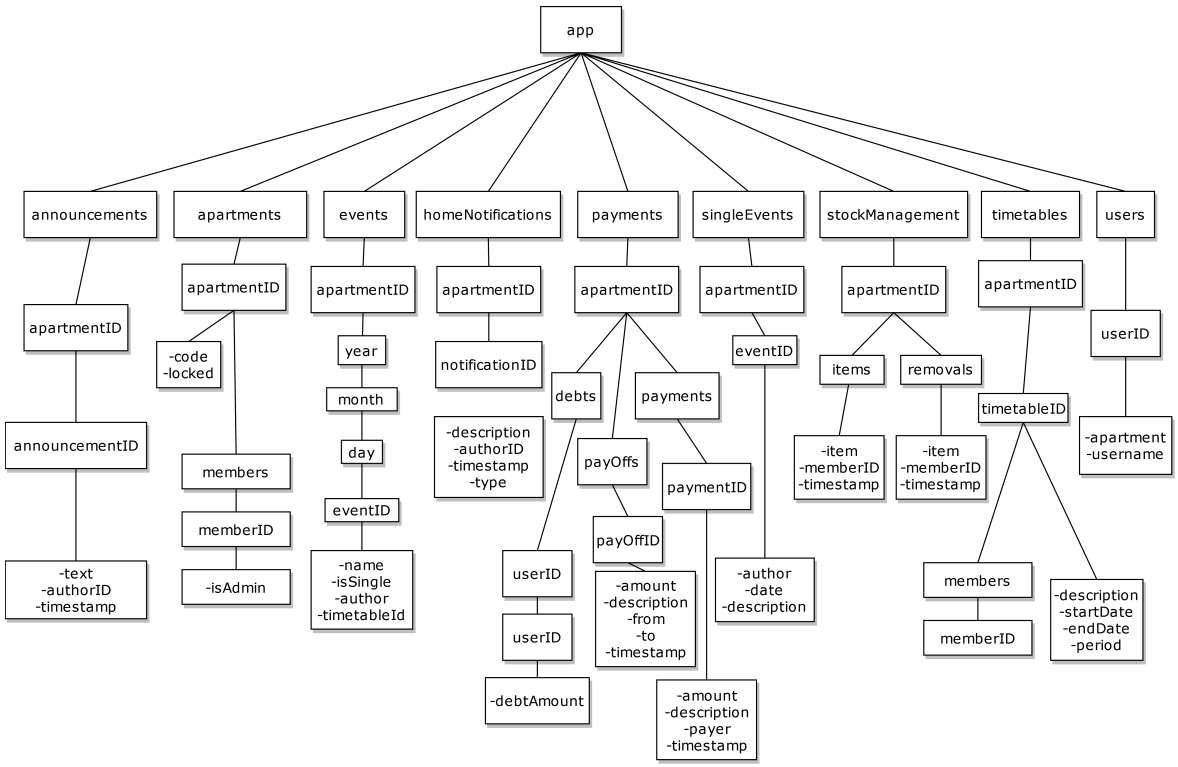


Figure 8: Database tree structure

The following paragraphs briefly cover the design of the database.

Announcement
This subbranch stores every announcements created within APPartment. Each apartment has a subtree where every new announcements creates a new node. The attributes of every announcements are:

- **text**: the description of the announcement
- **authorID**: the author's ID
- **timestamp**: creation timestamp

Apartments
The apartments subtree has a node for each apartment created in the application. The values of each apartment are:

- **code**: secret code used to access the apartment
- **locked**: boolean used to enable/disable the entrance of new members
- **members**: subtree containing every apartment member's ID and a boolean that describes whether they are admin of the apartment.

Events
The creation of a timetable/single Event in the database is going to enable a trigger that computes every event related to said timetable/single Event (e.g. the creation of the timetable "Kitchen cleaning" with a period of 2 days is going to generate an event every 2 days that is going to be stores in this subtree). The events are stores in their respective apartment subbranch and classified by their day of occurrence. The attributes are:

- **name**: the description of the event
- **isSingle**: boolean that specifies if the event belongs to a timetable or is a single event
- **author**: the creator's ID
- **timetableId**: reference to the timetable that generated the event.

Home notifications
This subtree stores the in-app notifications generated by the interactions with the various services:

- **description**: description of the notification
- **authorID**: the author's ID
- **timestamp**: creation timestamp
- **type**: service that created the notification

Payments
The payments subtree contains 3 subbranches. The **payments** subbranch stores every payment that a member of an apartment registered in the app. Its attributes are:

- **amount**: the amount of the payment
- **description**: payment's description

- **payer**: ID of the member that performed the payment
- **timestamp**: creation timestamp

The **payoffs** subtree stores every payoff performed by two apartment's member. It saves:

- **amount**: the amount of the payoff
- **description**: payoff's description
- **from**: ID of the member that performed the payoff
- **to**: ID of the member that received the payment
- **timestamp**: creation timestamp

The payment and payOff subtree work as a ledger of every transaction that ever happened in the apartment. The current balance of each apartment's member can therefore be computed by analyzing each of these subtrees. Due to efficiency reason, another subtree, **debts**, is updated via a trigger every time a transaction is added either in the payment or payOff subbranch, updating the current balance of every apartment member.

Single Event
This subbranch stores events that are not part of a timetable but rather a single occurrence.

- **author**: the author's ID
- **date**: event date
- **description**: event description

Stock management
This subtree stores every missing item added by an apartment member. The application keeps track of this item in the items subbranch and stores them in the removals subbranch once they have been removed. The data saved in these branches are:

- **item**: item name
- **memberID**: ID of the member that added this item to the list
- **timestamp**: creation timestamp

Timetables
Each timetable created within an apartment is stored here. In particular it saves:

- **description**: description of the schedule
- **startDate**: the starting date of the timetable
- **endDate**: the ending date of the timetable
- **period**: the frequency of the timetable
- **members**: list of apartment members that are part of the schedule

Users
Each registered user is stored here. The application saves:

- **username**: user's username
- **apartment**: user's apartmentl

4.6 Redux

APPartment uses Redux in order to manage its local state. This particular library has been chosen due to its capabilities of providing a **centralized single source of truth** throughout the whole application. In particular, a Redux store is initialized when the user open the APPartment application (given it already signed in and joined/created an apartment). This initialization is done trough a back-end call in order to retrieve information on the user and its apartment. The store will therefore save information such as user ID, apartment name, apartment members and so on. This information will be available throughout the application and can be updated in case of changes.

5 User interface design

This chapter presents the user interface of the mobile application by UX diagrams and screenshots.

5.1 UX diagrams

As said, the chapter starts with a series of UX diagrams that show the general flow of the screens giving a high-level idea of the user experience. We wrote "general" as some links were omitted, like the *return* from a screen to the previous, for a greater presentation clarity.

In the next diagrams the light blue blocks represent a specific screen.

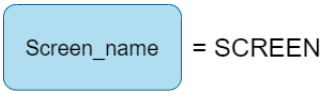


Figure 9: UX diagram - Legend

The first diagram shows the flow from the *login* to the *Join/Create screen* in which a user who has not already joined an apartment can create or join a new one.

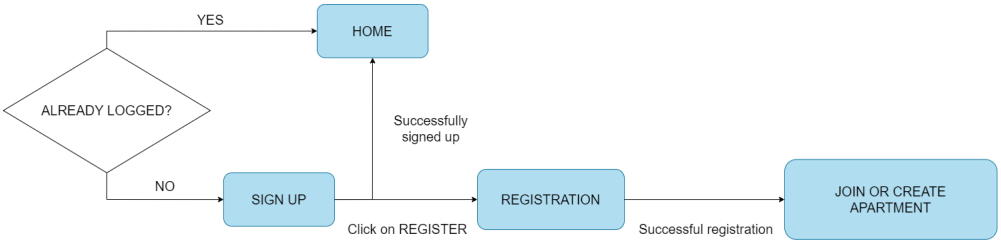


Figure 10: UX diagram - Login phase

In the next diagram the user is guided to the join or creation of an apartment, after that the user is routed to the home page of the apartment.
We can notice that when a user tries to join an existing apartment he's required to insert an access code that only who is already in the apartment knows.

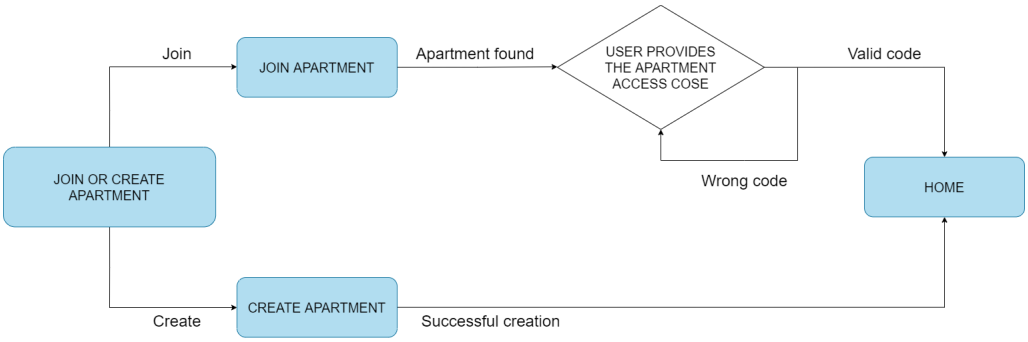


Figure 11: UX diagram - Join/create apartment phase

In the last diagram we can see the *Home* screens "hierarchy". As said before it is arranged as a tab navigation from which the user can reach the calendar, the services and the settings. The services screen is organized in a basic list from which the user can choose the one of them.

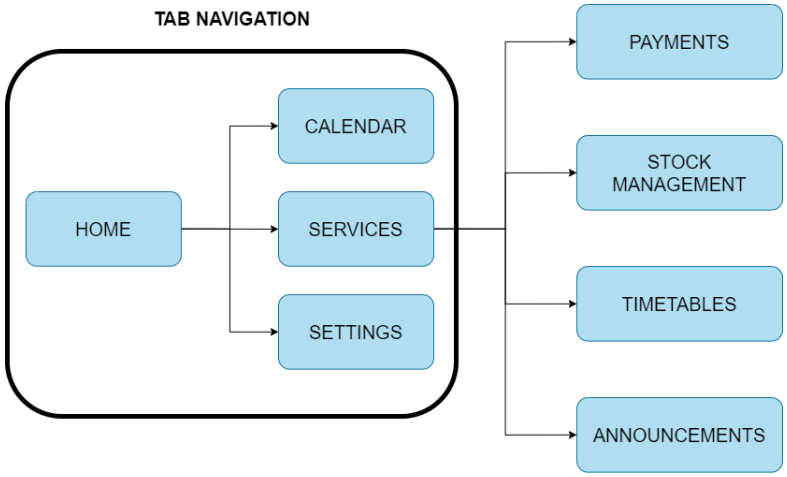


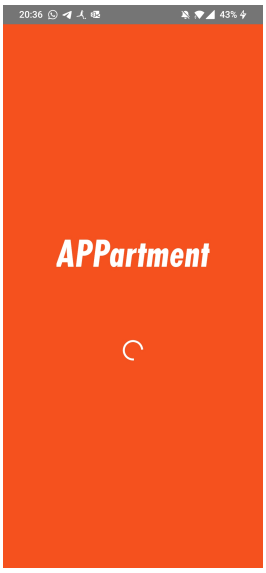
Figure 12: UX diagram - Home

The internal flow of the settings and of each functionality is not reported as it is not really meaningful for the purpose of the chapter.

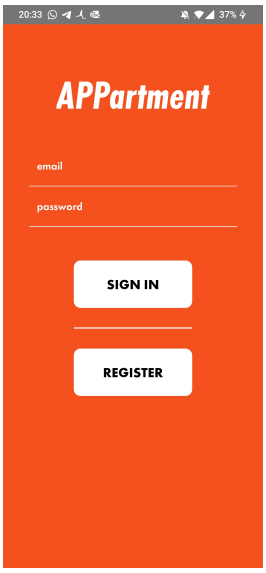
5.2 User Interface

The description of the user interface design continues showing some screenshots in order to give a deeper description of the mobile application look and feel.
The chapter is divided in two group of screenshots, each one related to a different device category. In fact, APPartment has a **multi-device support**, not only in terms of OS but also in terms of device layouts. Each screen automatically adapts itself to the layout of the device on which the application is running.

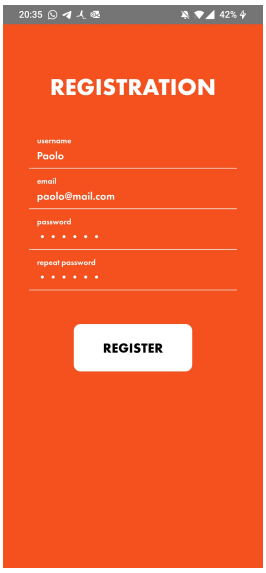
5.2.1 Smartphones



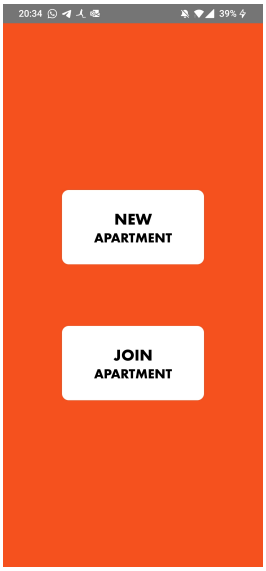
(a) App opening



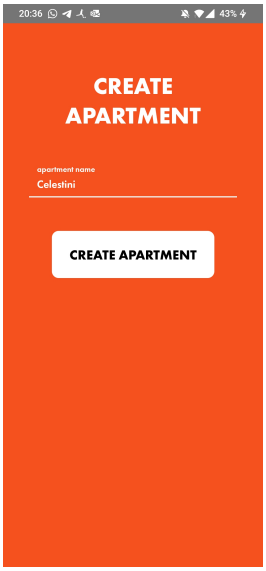
(b) Login



(c) Registration



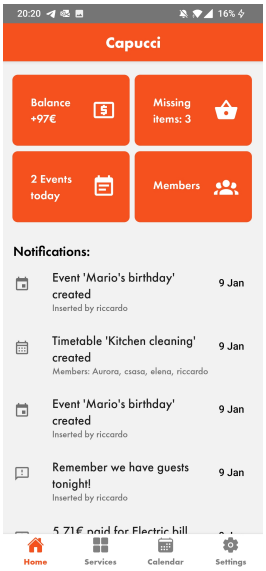
(d) Join / Create apartment



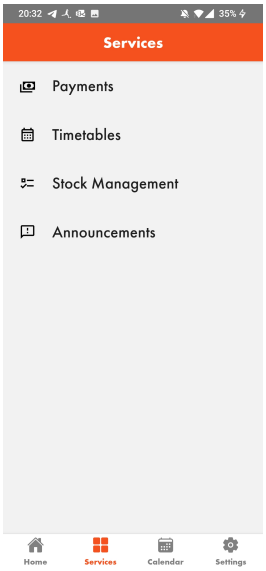
(e) Create apartment

(f) Apartment search

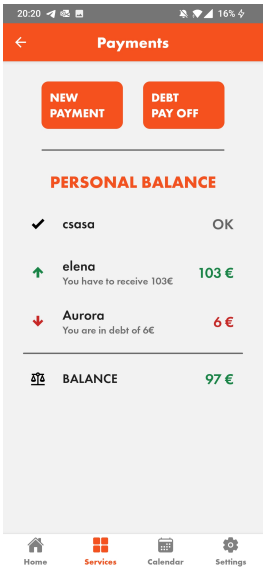
(g) Apartment join



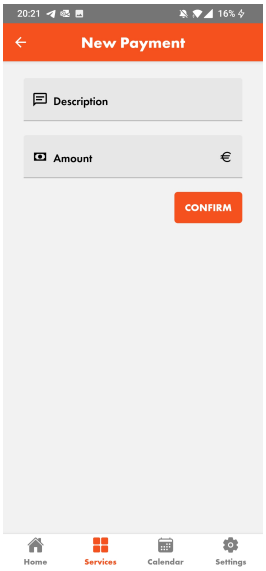
(h) Home



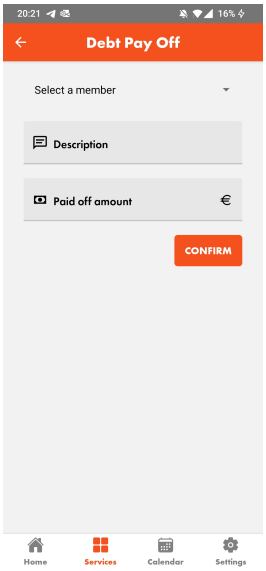
(i) Services



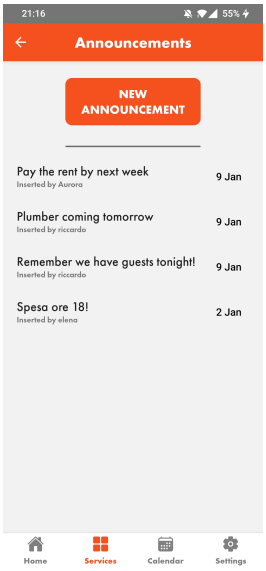
(j) Payments home



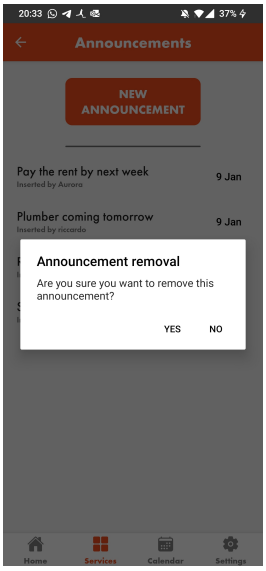
(k) New payments



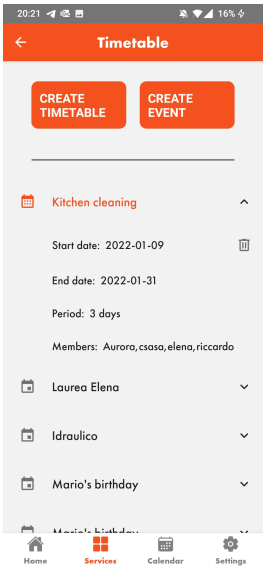
(l) Debt pay off



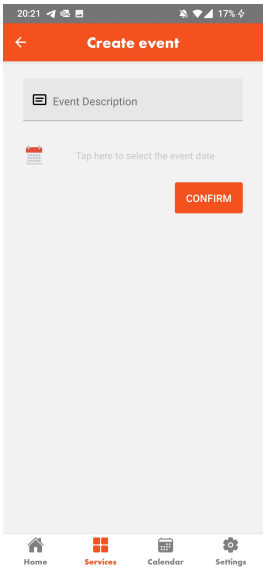
(m) Announcements



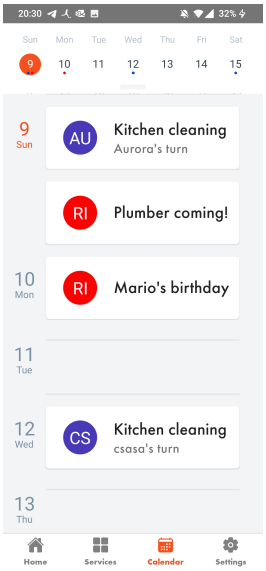
(n) Announcement removal pop-up



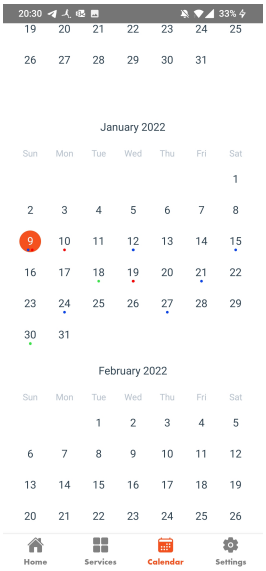
(o) Timetables home



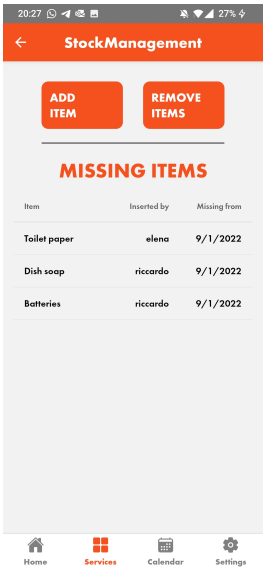
(p) Create event



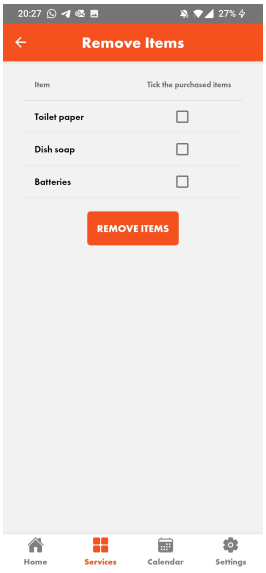
(q) Daily calendar



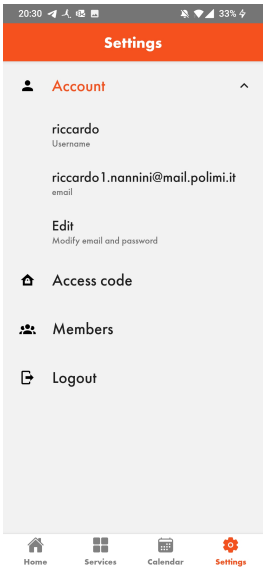
(r) Monthly calendar



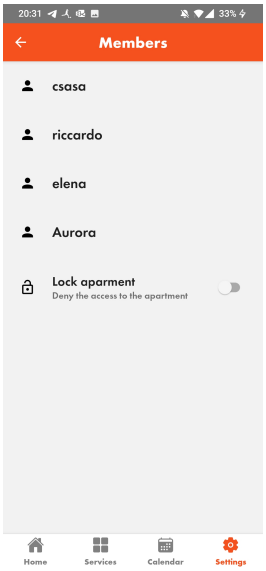
(s) Missing items home



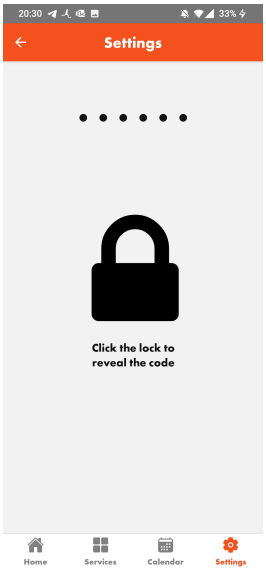
(t) Remove missing items



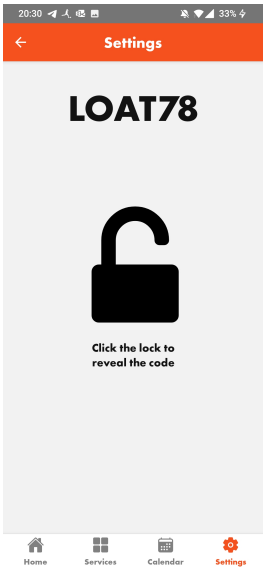
(u) Settings



(v) Members



(w) Access code locked



(x) Access code unlocked

5.2.2 Tablets

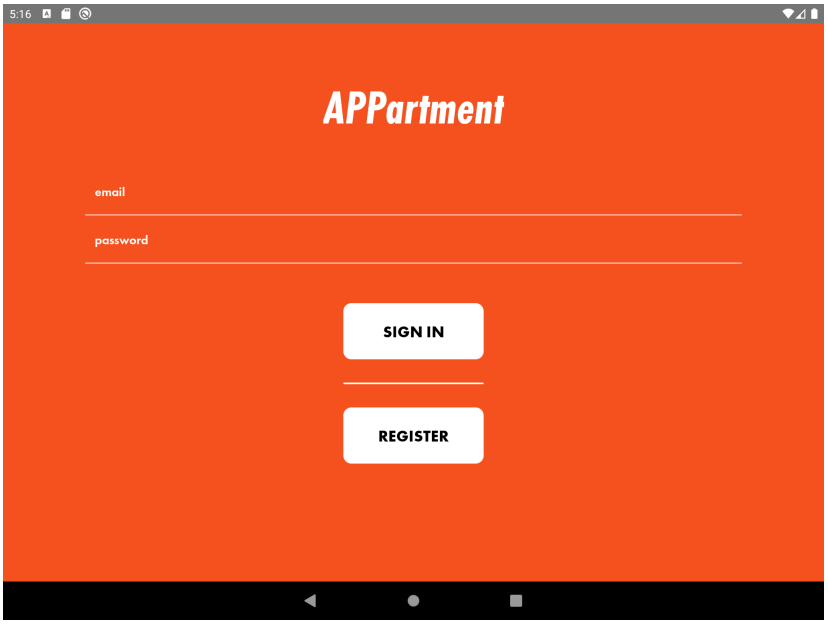


Figure 14: Login screen

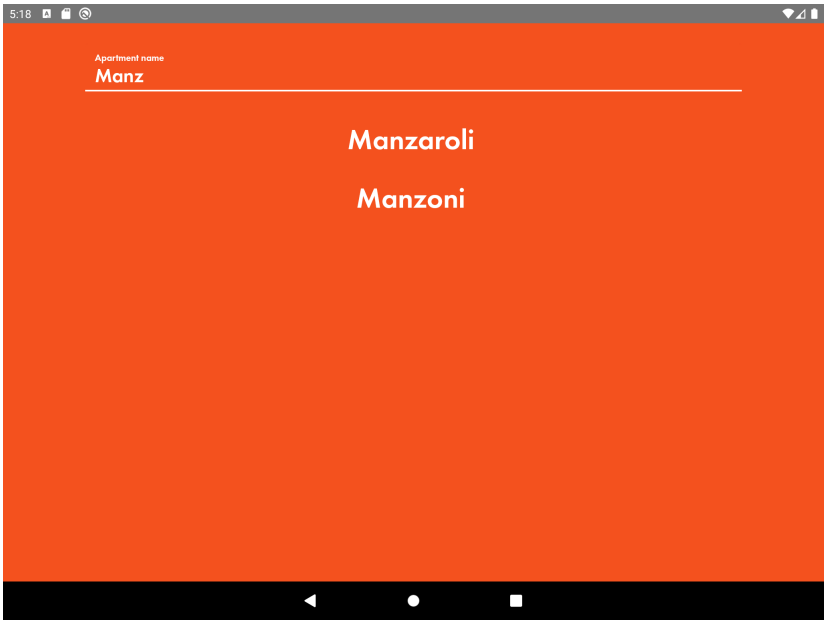


Figure 15: Apartment search

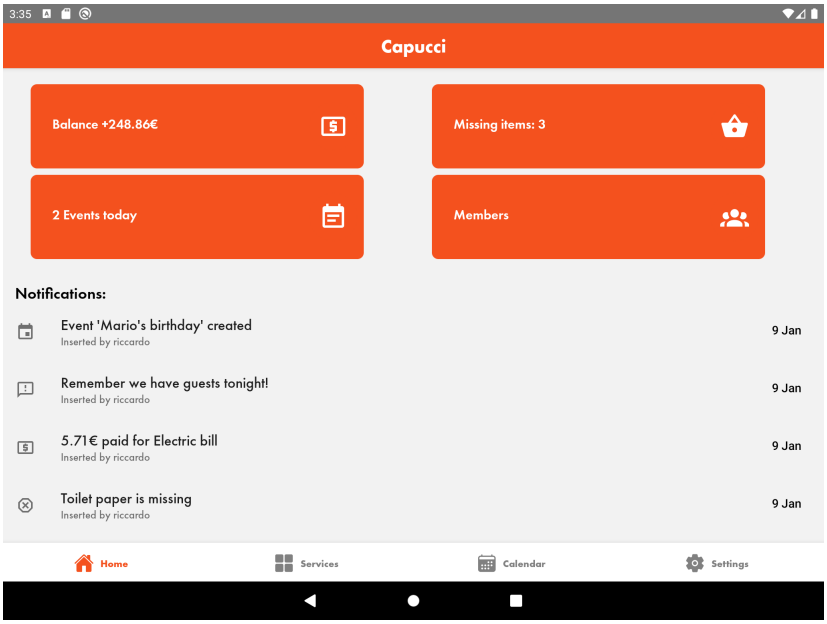


Figure 16: Home screen

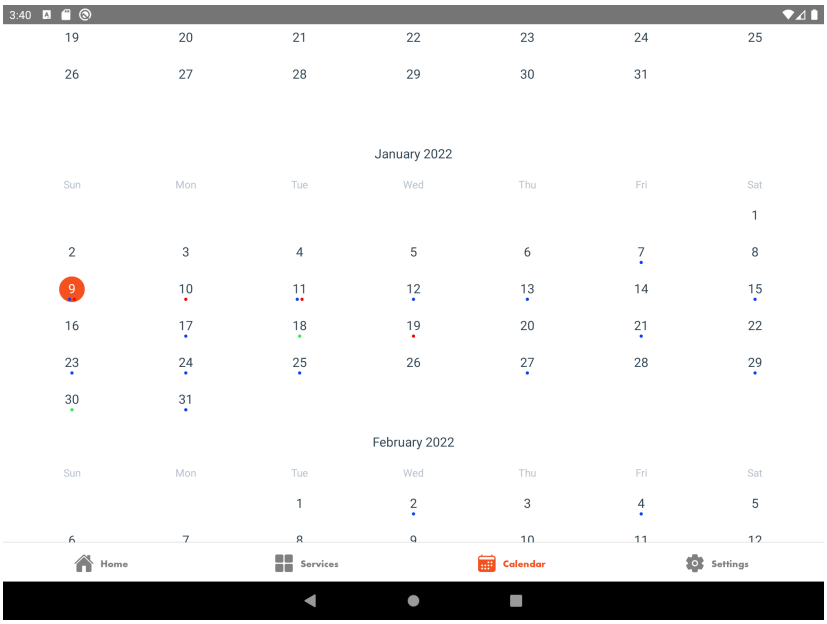


Figure 17: Monthly calendar

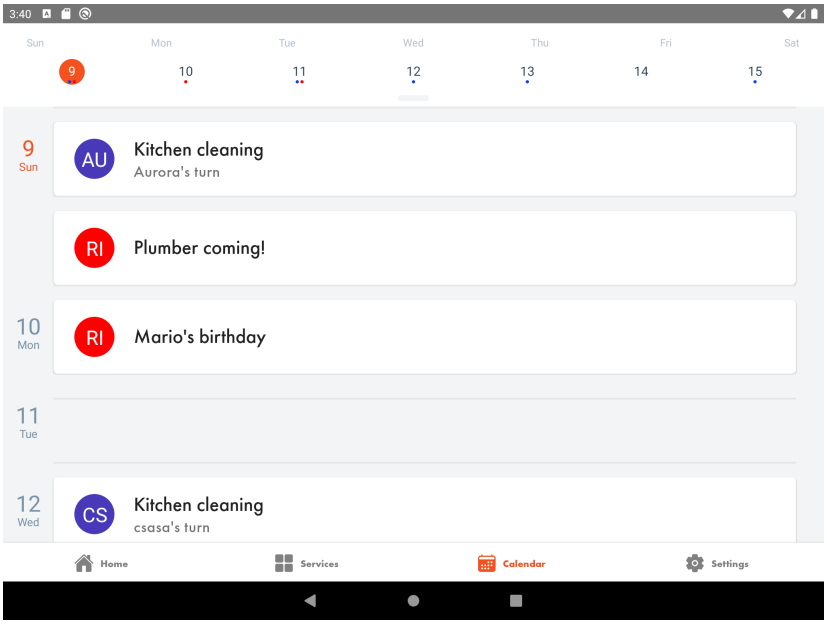


Figure 18: Daily calendar

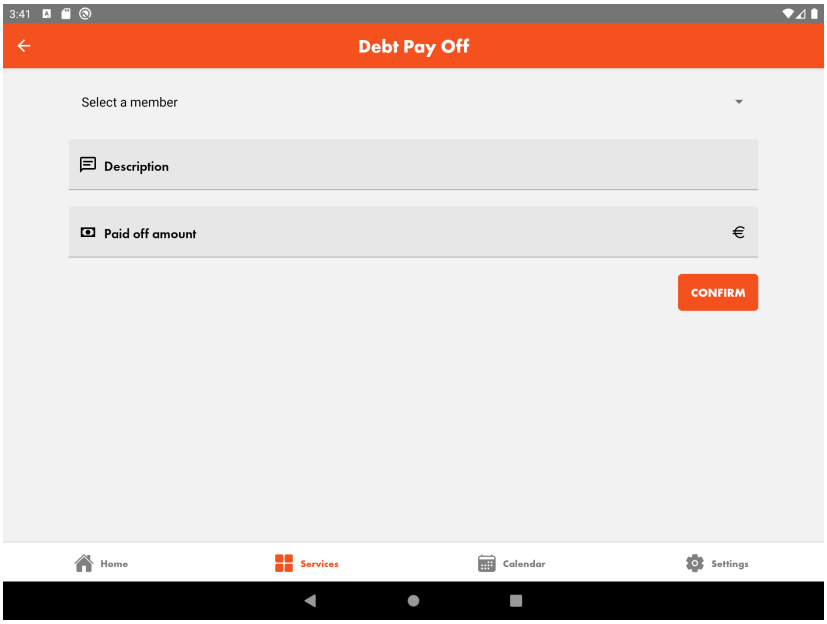


Figure 19: Debt pay off

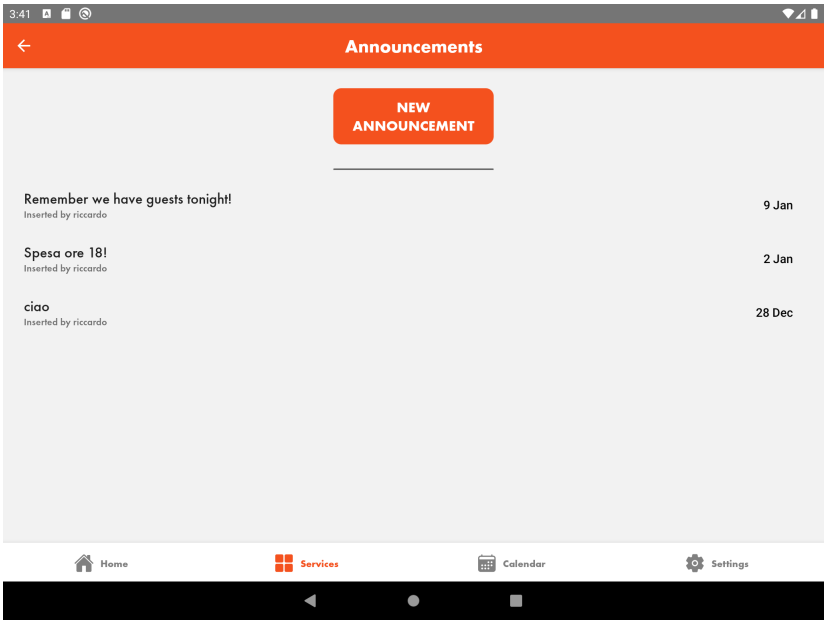


Figure 20: Announcements home

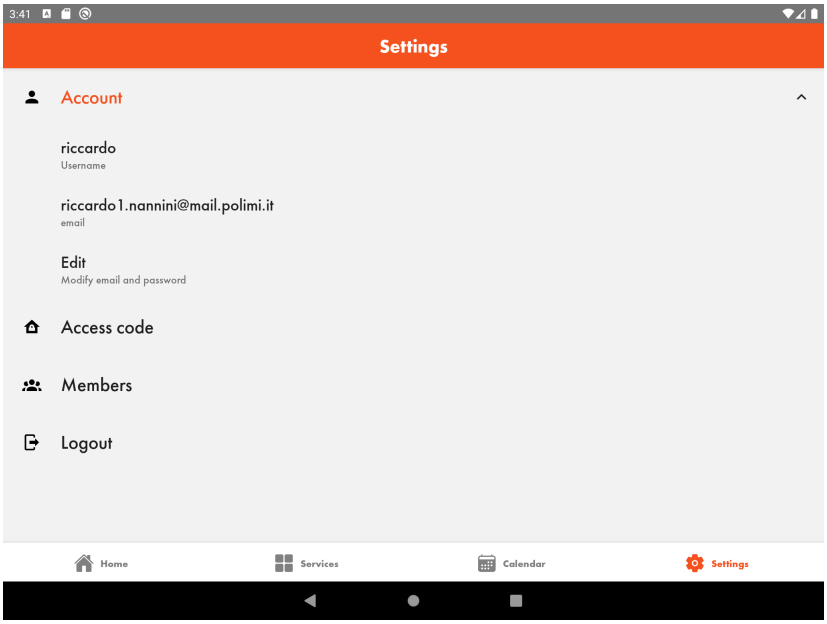


Figure 21: Settings

6 Testing

6.1 Application logic testing

The testing campaign includes white-box unit tests of several application components, both in front-end and back-end. The tests have been realized using the **Jest** framework and automated through **npm** (Node package manager).
Many of the Firebase Cloud Functions, the Function As a Service framework used for APPartment, composing the backend were tested with particular attention to the input parameter received through HTTP(S) requests. Test were aimed at finding out wether backend functions accepting user inputs were behaving correctly in case of missing or erroneous inputs as well as maliciously crafted ones (e.g. input that could not be generated given the application front-end but could be crafted through tools that allow you to create custom HTTP(S) requests such as Postman, Burp and similar).

```
Test Suites: 4 passed, 4 total
Tests:       86 passed, 86 total
Snapshots:   0 total
Time:        2.892 s
Ran all test suites.
```

Figure 22: Back-end test results

Regarding front-end testing, unit tests have been conducted on many application's screens with special focus on the ones allowing user manipulation of data. In particular, several tests were aimed at verifying that user inputs were correctly handled and that errors were correctly recognized and managed when the user entered a wrong input (e.g. testing that no passwords shorter than 6 characters were accepted in user registration screen).

```
Test Suites: 5 passed, 5 total
Tests:       64 passed, 64 total
Snapshots:   0 total
Time:        1.937s, estimated 15s
Ran all test suites.
```

Figure 23: Front-end test results

6.2 Simulator testing

The application has also been tested in an Android simulator. The testing was aimed at finding out how the application would have responded to screens of different sizes.
Different problems were found and fixed both with greater and smaller screens than the one typically used in the development process.

6.3 On device testing

The testing campaign included tests conducted on real devices. These tests were rolled out in order to find out if the application behaved correctly in a real device.
The application was tested in a Samsung device and in a OnePlus one. No particular problems were found apart from minor visualization issues that, due to the great difference in screen sizes typical of the Android world, had not been noticed until then.

7 Future development

The last section presents some missing feature that could be developed in the future in order to complete and improve the whole system.

Multiple apartment per user - currently the system supports only 1 apartment per user that could be obviously a limitation if someone needs to manage even just one more.

Integration with existing payment systems - the payments feature allows our users to keep track of their debts and with this improvement they will be able to pay off them directly through an existing payment system, such as *Satispay*, integrated in the feature.

Multiple task timetables - this improvement allows users to create timetables in which they specify a set of tasks that are assigned in turn to every room mate.
For example, speaking about house cleaning, if we state that tasks are different parts of the house, the feature will assign the cleaning of a different part of the house to every user and will switch every N days.

Apartment chat - a simple real-time chat for a faster communications than the one provided by Announcements.

Push notifications - a push notification service to alert user for incoming events and other stuff related to the timetables.

OAuth - the open standard could be adopted to allow users login with other well known authentication providers.