Politecnico di Milano

Design and Implementation of Mobile Applications

2021/2022

DD

Design Document

SubscribeME – a subscription manager app

August 30, 2022

Giarduz Andrea
Grosso Veronica

Prof. Luciano Baresi

Contents

SubscribeME Design Document

1 Introduction

1.1 Purpose

The purpose of this Design Document is to provide a guide and a walk-through of the application *SubscribeME*, in order to explain the design choices we made and to show the logic behind its architecture.

1.2 Scope

SubscribeME is a mobile application that keeps track of all the memberships that the user activated. The users should be aware of how much money they are spending and in which subscriptions, and be reminded of each renewal date.

In this way, the users are always in control of their finances, without forgetting where their money is going, when the payment is due and whether the specific subscription is worth keeping.

1.3 Document Structure

The document is structured in eight sections:

1. **Introduction**: is an overview of the purpose of the Design Document and of the problem analyzed.
2. **Application Features**: presents the application features and the basic functioning of the application.
3. **User Characteristics**: explains the user characteristics and use cases adopted when the app was realized. It gives an overview of the target population that was considered during the design phase.
4. **Design Overview**: the system architecture is presented. Diagrams and graphs are used to help the reader visualize and understand the underlying structure and to see how the different components are linked to each other and work together. The implementation of the server-client architecture is explained.
5. **User Interface Design**: User Interface choices and designs, along with screenshots of the final implementation of the app.
6. **Implementation, Integration and Test Plan**: testing of the components and the results and statistics of these tests are presented analytically.
7. **Future Development**: the next steps of this project occupy the last part of the document, Section , giving a few inputs on possible future works that can be implemented.
8. **References**: includes the tools and the references used to define the document.

2 Application Features

There are five main features in the *SubscribeME* app:

- the **Login/Signup** function: the users need to signup and login to be able to use the application. There is a persistent layer for each user and all the information is saved in the Data Base.
- the **Home page**: a sum-up is provided, with the amount of money that the user is spending at that moment on subscriptions. There is also the reminder to pay for the memberships with the closest deadline.
- the **Subscriptions List**: the user should be able to add, modify and delete their subscriptions on the app. In order to add a membership, the user must provide the following details:
  - the name of the membership,
  - the date when they subscribed to the plan,
  - how often does the subscription activates,
  - the amount of money due,
  - which card is being currently used to pay for it,
  - whether the payment is automatic or not,
  - if they are sharing the subscription with any friends.

  In this last case, the user knows they owe money to a friend, even if the subscription may not be in their name. In particular, if the user is not the 'owner' of the subscription, they cannot edit it, but only withdrawing from the offer.
- the **Statistics page**: offers them the opportunity of analyzing first hand their expenses, seeing how many subscriptions per category (Music, Movies & TV, Shopping, Tech or Other) they activated and how much money they spend on each of them.
- the **Profile section**: it contains the Friends list and the feature to add new friends. Also there are some settings.

3 User Characteristics

The target population is anyone who has multiple subscriptions active at the same time, who wants some help in keeping track of the costs and of the deadlines. Most importantly, it is useful for people who tend to activate memberships and then forget about them, so that they do not waste any money in unused perks.

3.1 Use case Diagram

Figure 1 shows the use case diagram from the point of view of the typical user. Every action can be performed only upon registration and, the following times, upon login. The available features involve handling the subscriptions, visualize the statistics and future deadlines and to manage friends.

Going into further detail, a subscription must be added in order to be able to modify it, delete it and to see the related statistics and future deadlines.
Concerning the management of friends, they can be added providing the friend's registration e-mail. The second user can then accept the request or deny it; once accepted, the friend can also be removed.

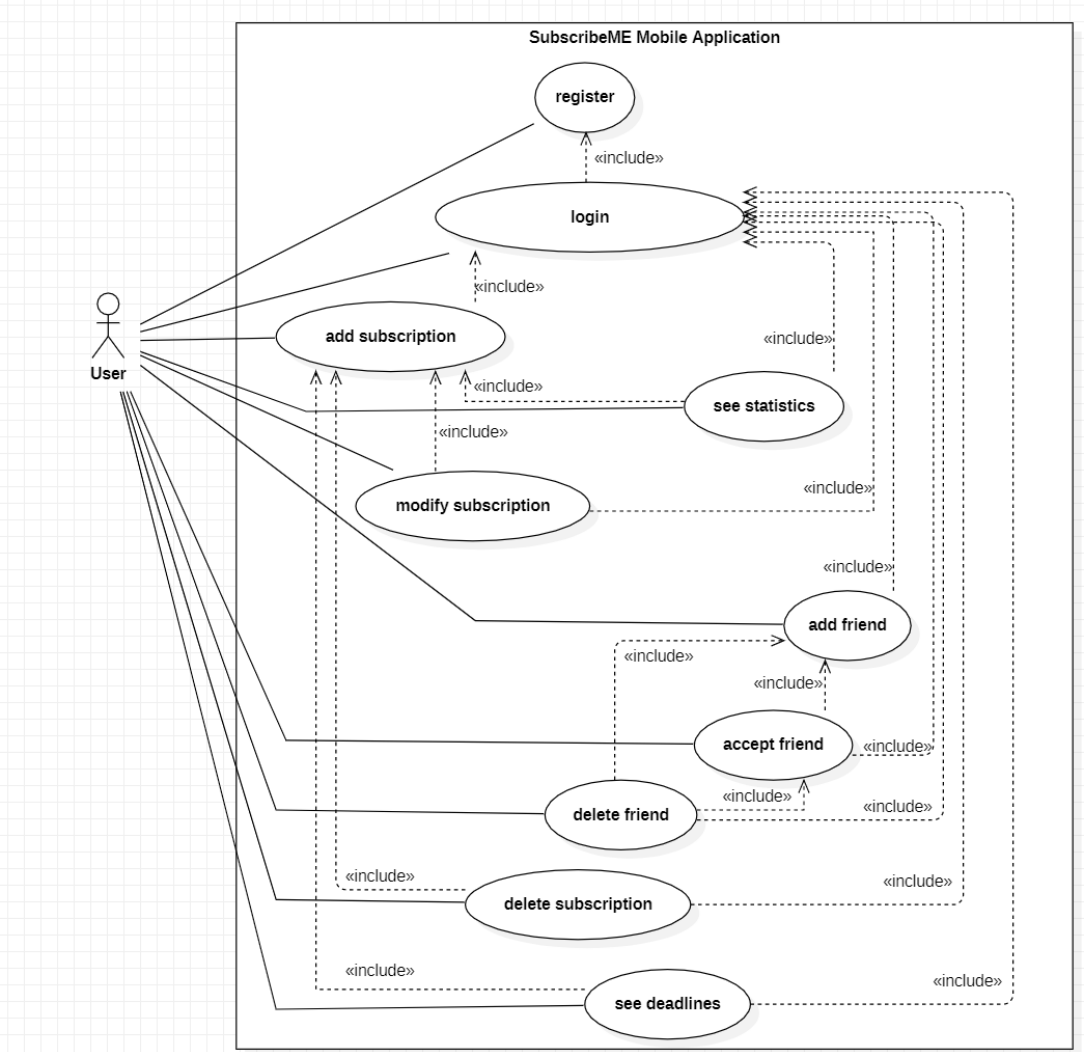

Figure 1: Use Case diagram

3.2 Scenarios

3.2.1 Reminder of the payment for a subscription

Arya activated a monthly Apple plan, as an additional warranty for her devices. She wants to be sure to be covered, because these devices are fundamental for her job. She is also very busy and she can get distracted, so she uploaded the information about this subscription on *SubscribeME* and then totally forgot. Luckily, at the time of the renewal of the offer, she sees on the app that she needs to pay for the plan.

3.2.2 Keeping track of the subscriptions

John loves subscribing for every free-trial he puts his eyes on. He cannot miss any opportunities. He often registers with all of his personal and card information. He receives a notification from the bank, his account is empty: he check the billing transcript and he realizes that he had forgotten to unsubscribe from the trials and he has been paying for months for offers he did not even use. He now wants to keep track of every subscription he activates, so he downloads *SubscribeME* to help him monitor his memberships and finances.

3.2.3 Monitoring the Statistics

Mary is a university student and is on a budget, since she does not have a job yet. With the recent price increases and inflation, she cannot afford to waste any money. That is why she decides to use *SubscribeME*: she adds all of her subscriptions, she goes to the 'Statistics' page and she realizes that she is definitely spending much more than she was aware of. The majority of her subscriptions are for 'Movies & TV': she does not need all of them.

3.2.4 Share a subscription

Amber and Luc are twins and they want to activate the Spotify family plan, to split the money of the subscription. However, Amber does not trust Luc to be honest with her about the price and to keep his end of the deal, so they both use *SubscribeME* in order to keep track of the deadlines and the money that they are spending. Both see the information about the plan and can monitor it.

4 Design Overview

4.1 Architectural Design Overview

The *SubscribeMe* application is based on the client-server paradigm. The user connects via the mobile app (the client) to the services provided in the server, such as the application logic (backend) and the database. Overall, the implementation structure can be described as a Model-View-Controller pattern. On one hand, the client contains the View and some input verification (part of the Controller); the server, on the other hand, has additional checks (the rest of the Controller) and the whole application Model.

The platform employed to implement the server architecture is Google Firebase, an out-of-the-box solution to manage the backend, as well as all the authentication functions and the data storage (Firestore).
In order to keep an updated and coherent state of the whole application, the Redux Toolkit library has been used. The Toolkit version is the latest release of the Redux code and it is meant to simplify and shorten the Redux configuration and code.

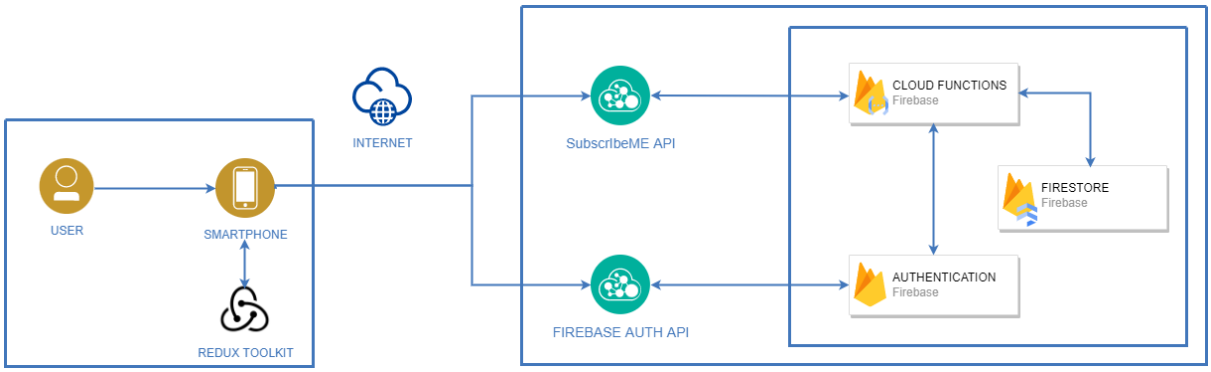

Figure 2: High Level Architecture Diagram

In Figure 2, it is possible to better understand the structure of the entire application. The application on the user's device interacts directly with Redux Toolkit and, through internet, with the application own APIs and the Firebase Authentication API. The former has access to the Cloud Functions stored in Firebase, while the latter uses the Authentication Functions from the Firebase own authentication provider. The Cloud Functions need access to the data stored in the Firestore database.

4.2 Client Application

The technology used to implement the application is React Native. This allowed a faster, cleaner and, most of all, single code base that can at the same time render an iOS and an Android app. The framework uses JavaScript as underlying programming language.

The client side contains mainly the View of the MVC pattern. Its structure is presented in Figure 3 and in is divided in two main branches, in order to keep every domain separated from the others and allow the user to better navigate from one section to the other, both logically and physically in the frontend implementation.

In particular, the 'main application', as stated in Figure 3, is implemented as a bottom tabular navigation, for a fast switch from page to page. The other links are stack navigation links, one for each domain, so that the history of one stack does not influence or negatively impacts the other stacks.

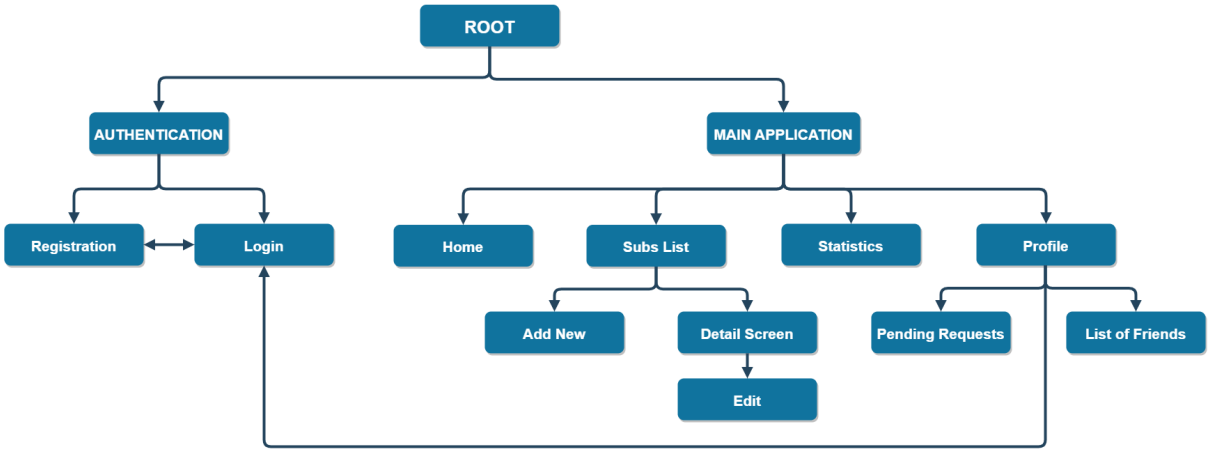

Figure 3: Application Navigation Tree: the structure of all the pages the user can navigate to.

4.2.1 Application Pages

Home Screen

The Home Screen's function is to summarize the useful data that the user can later analyze more in detail in other sections. Specifically, there is a greeting message and two boxes that display:

- the financial information: how much money the user is spending a year/a month and how many subscriptions they subscribed to;
- the payment reminders: which are the next two subscriptions that need to be renewed and payed and in how many days.

Subscription List Screen

The Subscription List Screen shows two separate lists of plans:

- 'My Subscriptions': are all of the offers that the users activated on their own behalf and, optionally, for other friends. They are the owners of the subscription and the only ones who can edit it.
- 'Shared with me': vice versa, these are the plans that a friend activated and shared with the user. In this case, the only possible action is to delete the plan and to rescind the offer.

Clicking on the subscription item opens a Detail Page, where the user can review all the information about the membership. There is the possibility to remove it and possibly edit it (if the user is the owner).

At the bottom of the Subscription List Page there is a button 'Add New'. This is for inserting in the database a new membership, and the form requires many fields to fill in all the data that can be useful to the user.

Statistics Screen

The Statistics Screen renders some graphs, in order to visualize the aggregated data about the memberships. The plans can be flagged with only one of the following categories:

– Movies & TV,

- Shopping,
- Music,
- Tech,
- Other.

There are two main graphs:

1. 'Your Category Shares': it is a pie chart to show the percentage of plans active in each category.
2. 'Money spent a month for each category': the sum of the money spent in each category thanks to an approximation computed on a monthly basis.

Profile Page

The Profile Page provides some settings, both in friends and profile management:

- 'Manage Friends':
  - 'Add a friend': provides the basic instructions to add a friend (by inserting the friend's e-mail)
  - 'Pending Requests': allows the navigation to a new page, that shows all the requests not yet accepted nor rejected, both in the Inbox and Sent.
  - 'List of Friends': shows in a new screen the list of friends and allows the user to delete them.
- 'Profile Settings': editable account information
  - 'Name': the account name, used for friends to easily recognize each other (non-unique information);
  - 'E-mail': unique identifier, can be changed upon insertion of the profile password;
  - 'Password': profile password necessary for the login, can be changed upon correct insertion of the old one.

A 'Logout' button brings the user back to the authentication page.

4.2.2 Component Diagram

The *SubscribeME* application component diagram is pictured in Figure 4. It illustrates the 'Root' components, alias the main application screens and how the Root communicates with Redux Toolkit and with the Cloud Functions. To avoid confusion, internal links from the single components to the Root interfaces are omitted.

Other diagrams are not shown, due to their trivial structure and linear flow (e.g. login and registration pages).

4.3 Firebase

The application requires authentication to be used, so that the application can properly store each individual's memberships and details. There is also some sensible data, such as the card's last four digits, so it is recommended to register with a strong password. Indeed, there is a 'Password Strength Bar' that helps the user realize whether the typed password is good enough for them.

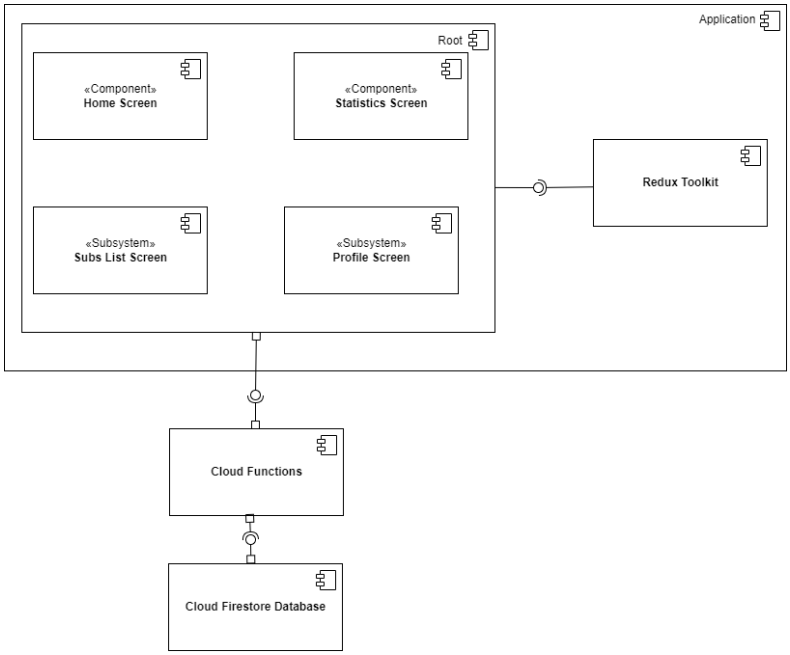

Figure 4: General component diagram for the mobile application.

The authentication per se is handled with Google Firebase, which has a simple interface that allows to automatically manage the registration (with the verification e-mail) and, if needed, the password retrieval e-mail (with the link for a safe reset of the password). The registration requires a chosen name, the e-mail and a password. Firebase already checks that the information provided is compatible with the wanted data. Nevertheless, we implemented some client-side input verifications as well, in order to allow only:

1. names with alpha-numeric values and less than ten characters. The name is not unique, but is used by friends to easily recognize each other when adding them to shared subscriptions. Since the same name can be used by different users, the e-mail is always shown as well, to avoid any ambiguity.

2. e-mails correctly formatted;

3. passwords with at least six characters.

In order to improve the user's experience, the logged user will keep access in the future, with no need of further authentication every time he opens the application. Only in case of manual logout, the user needs to access again through the login page.

4.4 Cloud Functions

The Cloud Functions use node.js as language to express the backend logic. They integrate flawlessly with the other services of the Firebase ecosystem. For instance, they offer a complete SDK to interact with the authentication and database products. They abstract the whole server infrastructure; the setup time and costs are perfectly suited for an application of this size, but they can also be easily scaled to handle a heavier workload.

Together with the authentication, they are the endpoints of the backend accessible by the application. They implement the logic of the application, and can be divided in two types of functions:

- onCall functions: triggered by events in the frontend.They encapsulate the authentication information of the user and possible input data. These ones are clearly the only Cloud functions accessible from the 'outside', and thus happen through an HTTPS connection.
- trigger functions: functions that respond to precise events of the Firestore database and/or of the authentication services (e.g. creation of a user, deletion of a subscription). They are used to keep the backend data consistent at all times.

In our implementation, we exploit the authentication information included with each call, to manage and control accesses to the data only if the user is actually authorized to do so. For example, only the owner of a subscription can modify its data. Maliciously crafted requests would fail these checks and consequently would be rejected.

Furthermore, each response is in a JSON format, including message about the success of the requests and the required data. Specific error messages are associated to specific points of failure, in order to keep the user informed of the nature of the error, while also guiding the developers in solving any bug.

4.4.1 *SubscribeME* APIs

The application APIs can be divided into four different categories:

1. manageSubscription (Figure 5): here there are all the functions which goal is to handle the subscriptions, such as retrieving all the subscriptions for a user, adding a plan, edit it or remove a subscription from a member who rescinded the offer.
2. manageUser (Figure 6): the functions to handle the profile requests, for example to set a name, to send a friends request or to reply to a request.
3. subscriptionMgmtTriggers (Figure 7): trigger functions to keep the database updated (on deletion or edit of a subscription).
4. userMgmtTriggers (Figure 8): trigger function to allow the insertion of a new user into the database in a consistent manner.

```
+ getUserSubscription()
+ setNewSubscription(subscriptionInfo: Object{})
+ editSubscription(subscriptionInfo: Object{})
+ removeMember(subscription: String, userToRemove: String)
+ deleteSubscription(subscription: String)
```

Figure 5: manageSubscription functions

4.5 Firestore database

Cloud Firestore is a NoSQL database hosted on cloud, in particular Firebase's newest database for mobile app development. Compared to the old Realtime Database, it provides faster queries and better scaling. Even if both databases have client-first SDKs (no servers to deploy and maintain) and realtime updates, Cloud Firestore manages the data as a collection of documents, which are very similar to JSON files. Since *SubscribeME* data have a pretty simple structure, this solution allows for a more agile and dynamic option.

```
+ setName(name: String)
+ addFriendRequest(email: String)
+ answerFriendRequest(friendUid: String, accepted: Boolean)
+ removeFriend(friendUid: String)
+ getFriendsData()
+ getCurrentUserInfo()
```

Figure 6: manageUser functions

```
+ onSubDelete()
+ onSubEdited()
```

Figure 7: subscriptionMgmtTriggers functions

```
+ onUserCreation()
```

Figure 8: userMgmtTriggers function

Firestore is set to be inaccessible from the 'outside', by protecting the database from unauthorized requests: only the database admin is allowed to edit any data. It is possible to see that in Figure 9: this is the only rule present in the Firestore database and it states that the only acceptable requests are from the admins. This way, this becomes a single point of failure when considering unwanted access and simplifies the rule definition. Every function of manipulation of the saved data or retrieval of information must be from the 'inside', through a Cloud Function.

```
1  rules_version = '2';
2  service cloud.firestore {
3    match /databases/{database}/documents {
4      match /{document=**} {
5        allow read, write: if false;
6      }
7    }
8  }
```

Figure 9: Firestore rule for accepting requests.

4.5.1 Database Structure

Given the Cloud Firestore structure as a collection of documents, we decided to store the data in two different collections:

- `subscriptions`: each documents, characterized by a unique identifier, contains:
  - `autoRenewal:` if the payment automatically generates at the end of the renewal period;

  - `card`: last four digits of the card used to pay;
  - `category`: category of the subscription (used to render the background color as well);
  - `currency`: payment currency;
  - `customName`: manually inserted name of the subscription;
  - `customType`: manually inserted type of the subscription;
  - `members`: list of friends with whom the user is sharing the subscription;
  - `name`: name of the subscription, chosen from a fixed list;
  - `owner`: user who created the subscription;
  - `price`: money due for the activation of the subscription;
  - `renewalDate`: date when the user activated the subscription;
  - `renewalPeriod`: length of repetition for the payment (e.g. week or month);
  - `type`: type of the subscription, chosen from a fixed list (e.g. student plan or family plan).

- `users`:
  - `friends`: list of friends added;
  - `pendingFriendsRecv`: list of pending friend requests received;
  - `pendingFriendsSent`: list of pending friend requests sent out;
  - `subscriptions`: list of subscriptions activated for a user.

This way, there can be repeated data (so the consistency in the database is harder to maintain), but it allows the management of faster and more complex queries.

4.6 Redux Toolkit

Redux Toolkit is the latest version of the Redux library. It has now become the standard way to write Redux logic and it overall simplifies the writing of Redux code, it speeds up development and provides good error catching, by managing the whole workflow.

More in detail, it keeps the application state updated, as a single point of truth for the whole application. Every page change updates in background the state, so that the user can get the correct data and enjoy an optimal experience at the same time. The strength of this library is represented by a centralized state: this is able to keep the data coherent throughout the different screens and pages.

5 User Interface Design

5.1 User Interface Flow Diagrams

In this section, two flow diagrams are presented (Figure 10 and 11). They illustrate the action flow, from the User Interface point of view, to authenticate a user and to edit a subscription.

These two cases are taken into consideration because they involve the most complex interaction for the user. All the other cases (see the first deadlines, see the statistics, add a subscription) are trivial and linear, thus not explained with a graph.

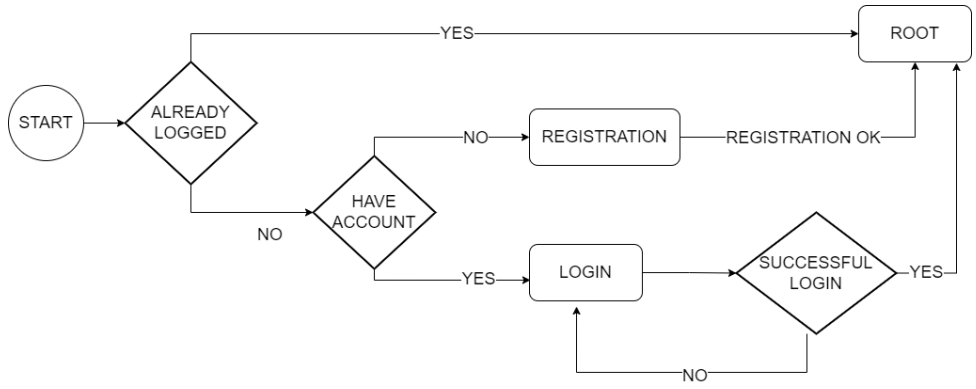

Figure 10: Login flow diagram.

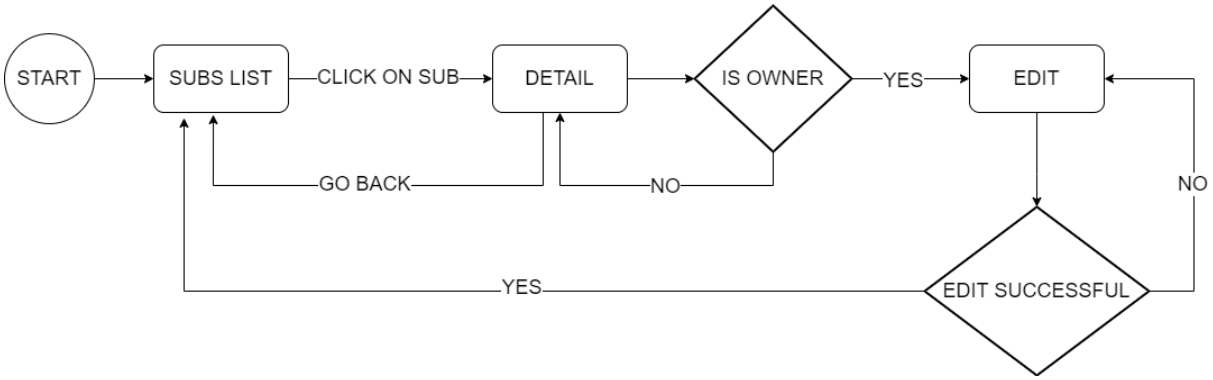

Figure 11: Login flow diagram.

5.2 Screenshots

The application design has been enhanced by customizing the interface specifically for smartphone and for tablet. The two devices typical characteristics have been considered during the design phase and this allowed a better implementation, acknowledging pros and cons of the portrait and landscape modes.

In particular, for what concerns the tablet implementation, the components sizes are enhanced, as well as the layout is changed in order to allow the maximum number of components to be shown at all times. This is why the main pages show a two-column layout, instead of a single column (portrait mode and smartphone design).

5.2.1 Smartphone Application

5.2.2 Tablet Application

6 Implementation, Integration and Test Plan

6.1 Implementation Order

6.2 Integration and Test Plan

7 Future Development

Some possible improvements and further implementations could be:

- notifications?
- automatic recognition of subscriptions
- payment system integrated
- chat between friends

8 References

The tools that we used are:

- LaTeX for the document editing;
- *starUML* for the use case diagram;
- *draw.io* for the navigation tree diagram;
- libraries.