LocalBuddy

Software Design Document

Michele Papale 899694

Politecnico di Milano

Contents

1. Introduction

1.1 Purpose

The aim of this document is to describe the software architecture of the LocalBuddy project, in order to use it during the development phase.
The application is developed for the course of "Design and Implementation of Mobile Applications" at Politecnico di Milano. The structure of the document will follow a top-down approach: from a general overview of the main components to a more specific view of smaller components.

1.2 Scope

This document describes the implementation details and all the design concepts of LocalBuddy. The software will consist of a chat-based application aimed to provide support to the traveler in simplifying his or her trips. Travelers will be able to contact local people (later called local buddies) in order to have advices, helps and everything concerns their trips.

1.3 Reference Documents

https://facebook.github.io/react-native/
https://firebase.google.com/
https://devcenter.heroku.com/
https://developers.google.com/maps/documentation/javascript/places-autocomplete
https://connectycube.com/

1.4 Document Structure

1. Introduction
This section contains a brief introduction to the main concepts of this paper such scope, definitions and references to main sources that helped us.

2. Overall Description
This section contains an introduction to the problem and to the project. Here we clarify what are the main functionalities LocalBuddy will have and we define some important concepts.

3. Functional Requirements

In this section we show the main important use case diagrams and scenarios.

4. Non Functional Requirements
This section explains what are the main non-functional requirements LocalBuddy will have.

5. Architectural Design
This chapter explains the architecture chosen for the project. There are also component and sequence diagrams to improve readability of the solutions we have chosen for LocalBuddy.

6. Specific Requirements
This chapter contains UI screens of the application.

7. External Services and Third Party Libraries
This chapter lists the external services and third party libraries LocalBuddy will use.

8. Implementation, integration and test plan
Here, you can find a little look to the implementation. There is a planning of how we decided to integrate the subcomponents.

2. Overall Description

2.1 Product Perspective

2.1.1 General Description

LocalBuddy is an application that wants to make comfortable as much as possible a travel in a new place where you have never been. It connects a traveler with a local person (in the following, called buddy). The traveler will choose a city and the application will list all the local buddies available. The idea is to allow people to find a “friend”, a person that helps the traveler during his local stay in that city, advising best places to go and giving the best possible advice.

2.1.1 Current System

The system-to-be will be implemented from the ground up as there is no existing system to start from. Nevertheless the application will greatly rely on existing systems for map and chat functionalities.

2.2 Product functions

Below we have listed the most relevant functions that the application must offer.

- [F1] - Allow visitors to login and sign up through the most known social network (Facebook) or to create a new account through email and password.
- [F2] - Allow users to find buddies in a specified city.
- [F3] - Allow users to find closest buddies.
- [F4] - Allow users to view buddies’ profile.
- [F5] - Allow users to chat with them.
- [F6] - Allow users to modify their profiles
- [F7] - Allow users to become buddies, specifying cities and other user preferences (who can find them and the sex of the traveler).
- [F8] - Allow users to delete their account, log out and modify password.
- [F9] - Allow users to create and delete meetings.
- [F10] - Allow users to select a monthly/list view of the appointments.
- [F11] - Allow the users to see past messages and meetings even if their mobile phones are not connected to a network.

To remove all doubts we have discussed in greater detail how the system will provide the most critical functionalities among the ones listed.

2.2.1 User characteristics

- **Visitor**
  A visitor is a non registered user. A visitor can only access the opening page of the application from where he can:
  - Sign up
  - Log in

- **Registered User**
  A registered user (later, simply called user) is a user who has signed up/logged in to the service through the provided social networks or by creating a new account through email and password. He/she can access all functionalities offered by LocalBuddy.

2.2.1 Looking for a Buddy

The user has to choose a city in which he wants to find a buddy. Then, it will be able to see a list of buddies and for each of them he can see his profile and, if he wants, he can start a chat. Of course, a user will be able to see a buddy for that city only if the age and sex constraints chosen by the buddy are satisfied by the user. The user can also search for closest buddies.

2.2.2 Meetings

Meetings can be of three types:

- Fixed Meetings

It is a meeting that has been created from a user and that the other one has accepted.

- New Meetings

Meeting that one user has created and the current user has to decide if to accept or reject it.

- Pending Meetings

It is a meeting that has been created from the current user and that has not accepted/rejected from the other user yet.

2.2.3 User Settings

Become Buddy: at the beginning each user is not a buddy. Through this setting, he will be able to become a buddy. He has to provide some informations:

- Who can find him: he can specify the range of the tourist's age and his sex,
- His cities: the cities where he wants to be a buddy.

Stop to be a Buddy: users through this setting can decide to be no more visible for the cities in which they are buddies.

Change his account password: users can change their account password

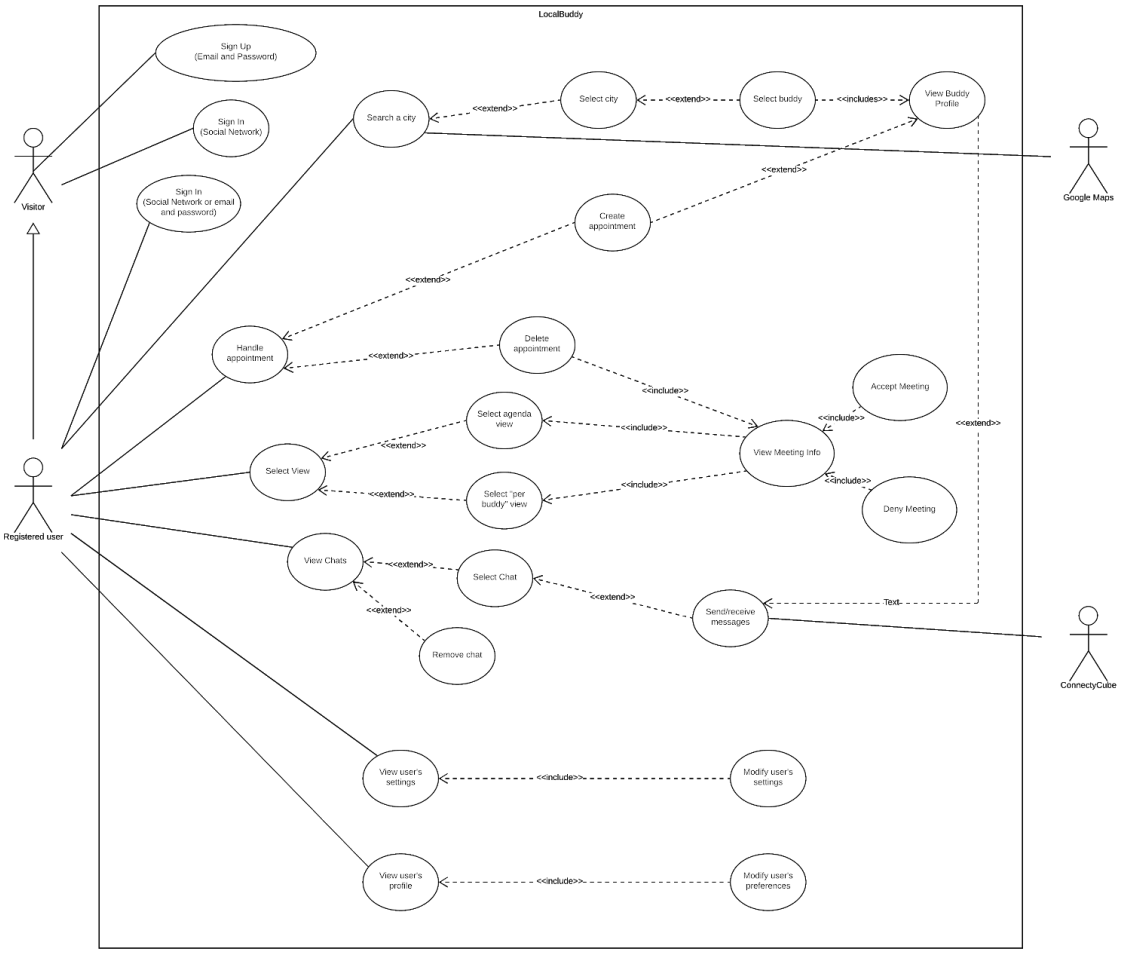
Delete his account: users can delete definitively the account they use to login in LocalBuddy. This action is not reversible.

Log out from his account: users can logout from their accounts. After this action, they will be directed to the login screen.

3. Functional Requirements

3.1 Use Case Diagrams

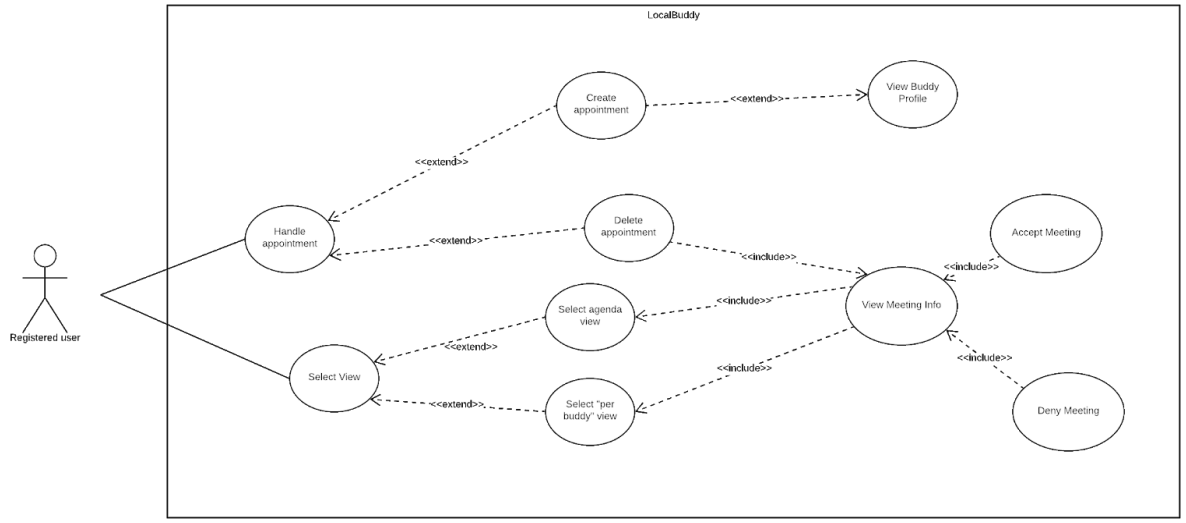
3.1.1 Entire project



To better clarify this use-case diagram, we will discuss in greater detail two important cases:

- Meetings
- Chats

3.1.2 Meetings



A registered user will be able to select two types of views:

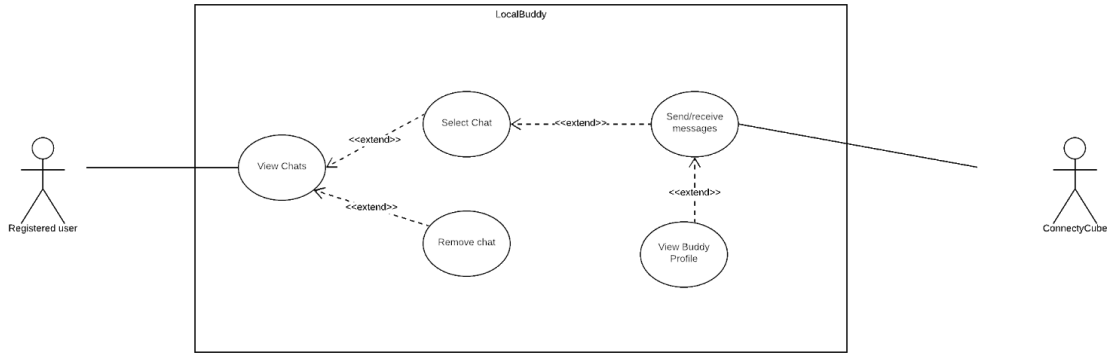
- Agenda view
- “per buddy” view

To fully understand the differences and to see these screens, the reader can go to [5.1.1 - User Interfaces - Smartphone].

User will be able to:

- create new appointments
- delete existing ones
- view Meeting info
- accept meeting
- deny/decline meeting

3.1.3 Chats



A registered user will be able to view his chats in a screen listing all the chats he has. From this screen, he can remove or select a chat. Clicking on a chat will redirect the user in a new screen where there are all the messages with a specific user. In this screen, the user can send/receive new messages

and by clicking on the name of the opponent he can see his profile. To see these screens, the reader can go to the reader can go to [5.1.1 - User Interfaces - Smartphone].

3.2 Scenarios

Scenario 1

Sign up, sign in, looking for a buddy and chat with him

Miriam is visiting Milan for fun. Not knowing how to move around the city she looks for an app on the Internet and finds LocalBuddy, the best app for her purpose and downloads it on her android device.

After signing up and logging in, she reaches the application's home page. Then, she goes in the search for a city page and she types "Milan". She finds Marco, a buddy of Milan, and, after she has seen his profile, she decides to chat with Marco in order to have the best advice about Milan.

Scenario 2

Adding of a meeting and give a feedback

Miriam, after has chatted with Marco, decides to create a meeting with him in order to remember herself of the appointment they have established. When Marco receives the notification, he accepts the meeting. At the time of the meeting, both Miriam and Marco will receive a notification and, from that moment on, they will be able to give a feedback to the other.

Scenario 3

Updating user profile and user's settings

Marco hasn't updated his profile for a long time and he has moved from Milan to Naples. For this reason, he decides to change his photo profile and he changes the cities in which he is a buddy: he deletes Milan and he adds Naples.

3.3 Use Case Descriptions

Use case description scenario 1

Actors	Visitor that becomes user, user that is a buddy, Google Maps
Input conditions	The user is not registered yet.
Events flow	**1.** The user downloads the app. **2.** The user signs up filling all the required fields. **3.** The system checks some requirements (if the email is already stored in the database) **4.** The system saves the user's data or notify the user of the already existing e-mail in the database. **5.** The user inserts his credentials and logs in LocalBuddy. **6.** The user searches for "Mi". **7.** The system interacts with Google Maps that gives back cities that starts with "Mi". **8.** The user clicks on "Milan". **9.** The system shows buddies in Milan. **7.** The user starts chatting with Marco, a buddy in LocalBuddy.
Output conditions	The visitor became user and, at the end, has received some advices from a buddy.
Exceptions	**1.** The e-mail inserted by the user is already in the database. **2.** Visitor inserts wrong inputs during the enrolment or during the log in. For all the errors, the visitor/user is notified by the system.

Use case description scenario 2

Actors	User and a buddy
Input conditions	the user and the buddy are registered users, they have a chat and they have connection to Internet.
Events flow	**1.** A user adds a new appointment. **2.** He fills in all the required fields. **3.** At meeting time, LocalBuddy notifies both users of the meeting.. **4.** The user and the buddy, after the meeting will provide a feedback
Output conditions	Users was sure to not forget the meeting and after the meetings they provided a feedback to help the community becaming stronger.
Exceptions	**1.** The user hasn’t filled all the required fields.

Use case description scenario 3

Actors	User, Google Maps
Input conditions	The user is a registered user and he has a connection to Internet.
Events flow	**1.** A user wants to update his photo profile. **2.** He goes on the gallery and finds the right picture he wants to use in LocalBuddy. **3.** The system checks if there is Internet connection. **4.** The system updates the photo. **5.** User goes in “Settings”. **6.** The user deletes Milan from the cities where he is a buddy. **7.** The user searches for “Mi”. **8.** The system interacts with Google Maps that gives back cities that starts with “Na”. **9.** The user add Naples in the cities where he is a buddy.
Output conditions	The user has updated his photo profile and the cities where he is a buddy.
Exceptions	**1.** The user has searched for cities that does not exists.

4. Non Functional Requirements

4.1 Performance Requirements

LocalBuddy must run smoothly to ensure a comfortable navigation to users:

- When opening the application, the application should be loaded and responsive within 10 seconds.
- Response time during navigation between the application's pages and button pressing must be less than 0.2 seconds.
- When an appointment is added it must appear on the schedule within 1.0 seconds.

4.2 Reliability

The system must be fully operational 24/7. The application must be stable and never crash.

4.3 Security

The system will store the user's credentials, preferences and saved appointments. Security of such personal data is a primary concern. All the requests to the backend must be made secure in order to avoid that third parties change the state of the DB.

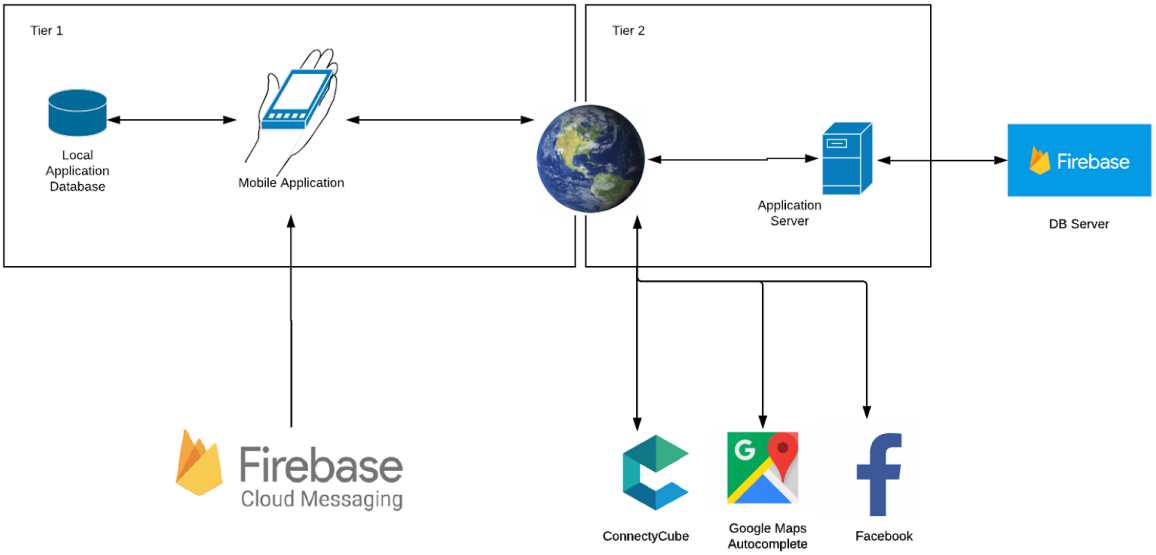
4.4 Mantainability

The system should be structured so to minimize maintenance efforts. Accurate usage predictions and high scalability are essential.

5. Architectural Design

5.1 Overview

LocalBuddy is structured on a two tier architecture. This division is both physical and logical: the first tier interacts with the client and with external services, it is the presentation layer; the second holds the application logic and stores the application data.



The system requires a “fat client”. This means that all application logic regarding offline functionalities must be stored within the client. This is the most robust solution since the application must work at all times and a stable internet connection cannot be assured.

On the application server, that will run on Heroku, will run Node with a process listening for API requests. To ensure security the requests are authenticated through the Firebase Admin SDK. Each request will have in the body a token that will be verified on server side. If this token is a valid one, the request will be performed successfully.

All functionalities that require communication with external services, such as searching for a city/buddy or sending messages in the chat, will be available only when an internet connection is established from the user’s device.
LocalBuddy’s database will store a copy of all local data because it is necessary for restoring user accounts, when logging in from a new device.

LocalBuddy will use as external services:

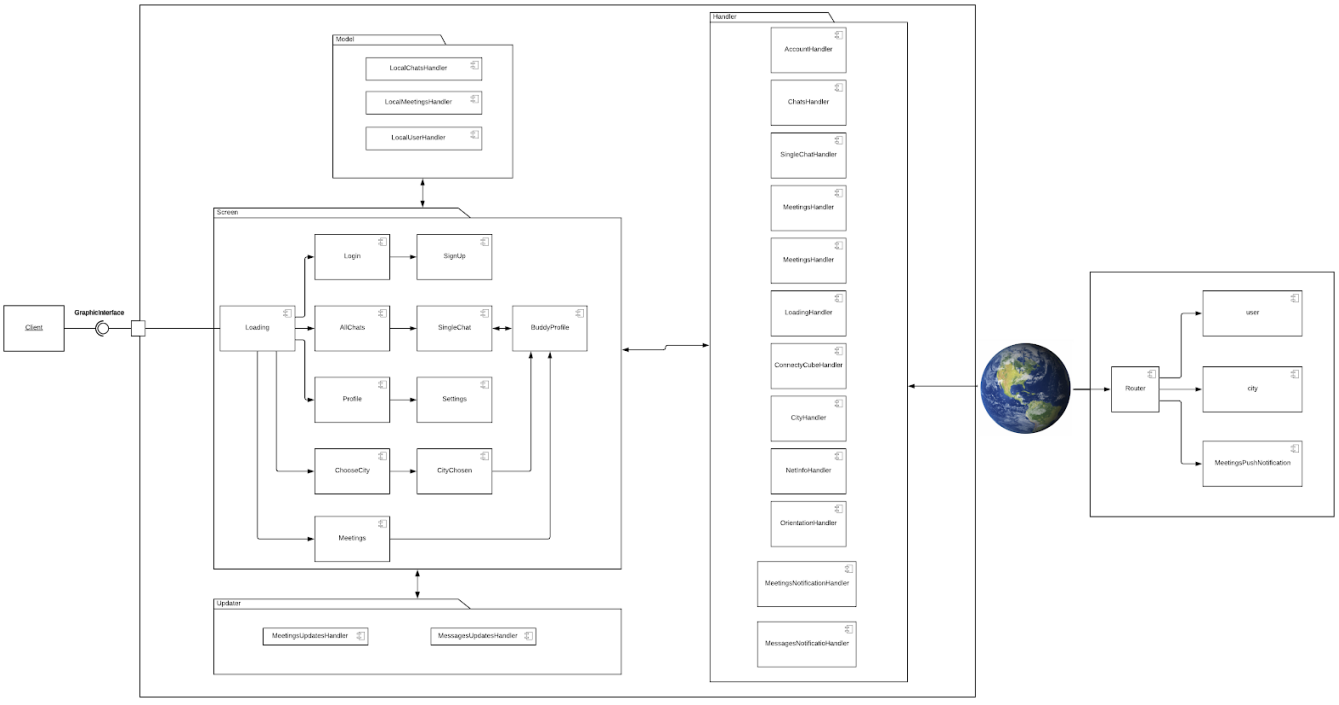
- Facebook, as a way to authenticate the user in order to provide a social login,
- Google Maps, to search all the possible cities in the world,
- ConnectyCube, a cloud communication platform to handle the backend of the chat.
- Firebase Cloud Messaging, in order to receive notifications on device

This solution ensures:

- system scalability
    - 2-tier architecture: in this way, in a future, it can be created a web application using the application server already implemented
- reliability:
    - offline functionalities
- security:
    - Application server listening only authenticated requests

5.2 Component view

Below, it is the component view of all the LocalBuddy project.



It is important to notice that the majority of business logic is not only on cloud but also on the client. This is due to the fact that the client stores everything in the smartphone's internal memory, as already stated in section [2.1 - Overview].

The basic application will be able to satisfy all user requests and to manage data on the mobile device through the LocalStateHandler. In this way, also if the user’s device is not connected to Internet, he will be able to use LocalBuddy.

Furthermore, NetInfoHandler controls whether the terminal is connected to internet and, in case of success, it will call the local handlers in order to update the copy of the model stored in local.

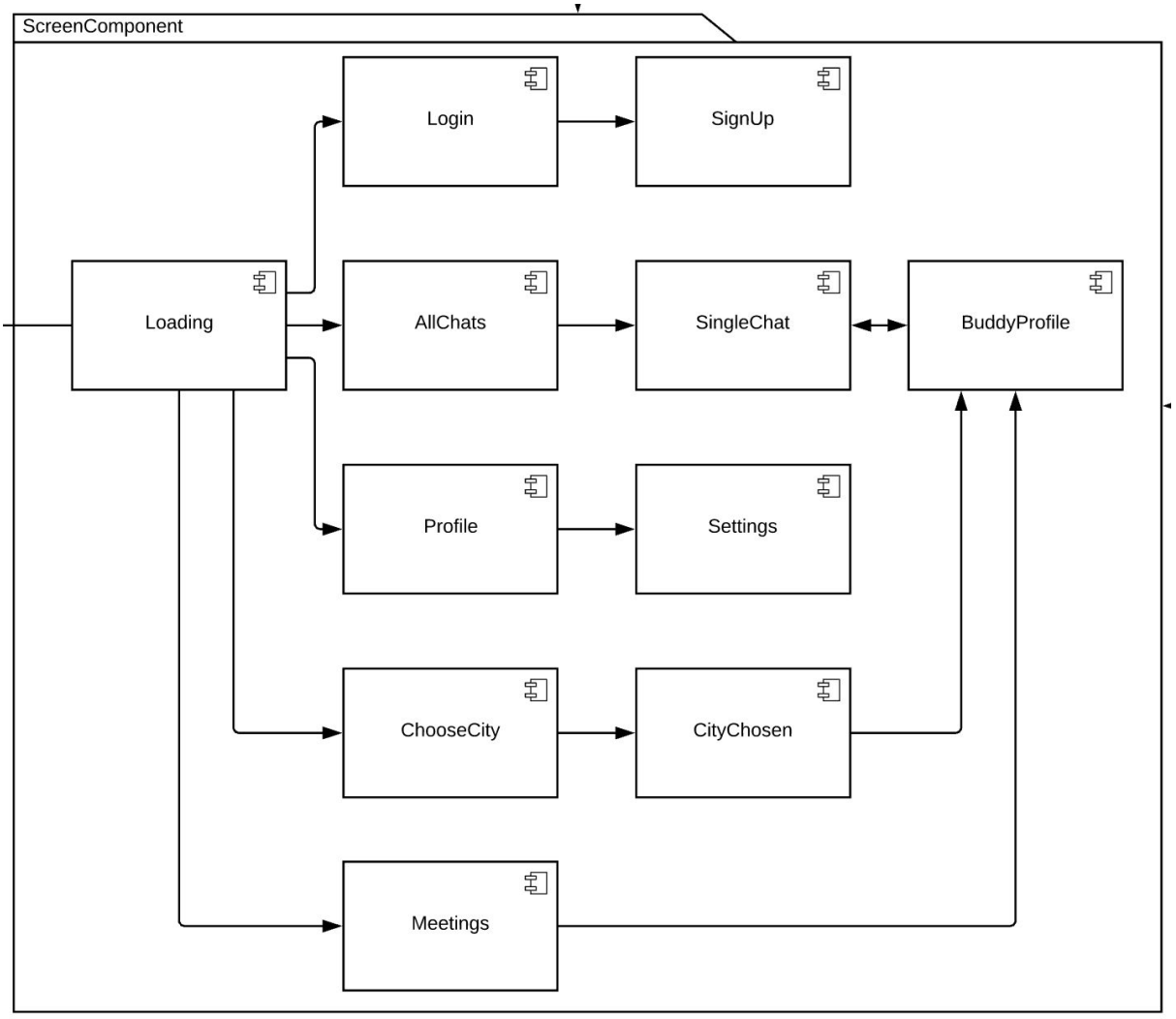
To better clarify each component and to give a better overview, below are shown the components organized by functional area.
Macro-areas on client device are:

- Screen, that lists the screens of the app
- Model Handler, components that handle the local state
- Backend Handler, components that handle the requests to the backend and to external services,
- Updater, components that updates the state of the client because of updates from backend

Macro-areas on server are:

- Router
- Handler

5.2.1 Screen Components

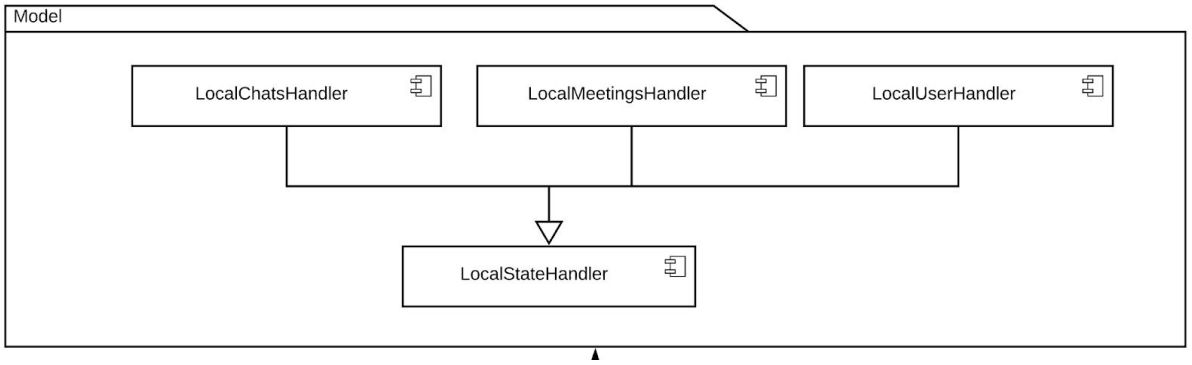


Below, a brief description of each component:

- *Loading:* It will be the entry component of the app. It will redirect to the right screen depending on the auth status (if it is already logged in, the user will be redirected to AllChats. If the user is not already logged in, he will be redirected to Login component)
- *Login:* Manages sign in procedures (both email and password and Facebook login)
- *SignUp:* Manages sign up procedures.
- *AllChats:* component used in order to list all the chats the user has
- *SingleChat:* used in order to show the selected chat
- *BuddyProfile:* Component used to show the profile of a buddy
- *Profile:* Used to show the profile of the user currently logged in
- *Settings:* Component that shows settings the user can modify
- *ChooseCity:* Component used to search other users, starting by searching a city

- *CityChosen:* Lists all the buddies for the city specified by the user
- *Meetings:* Lists all the meetings the user has

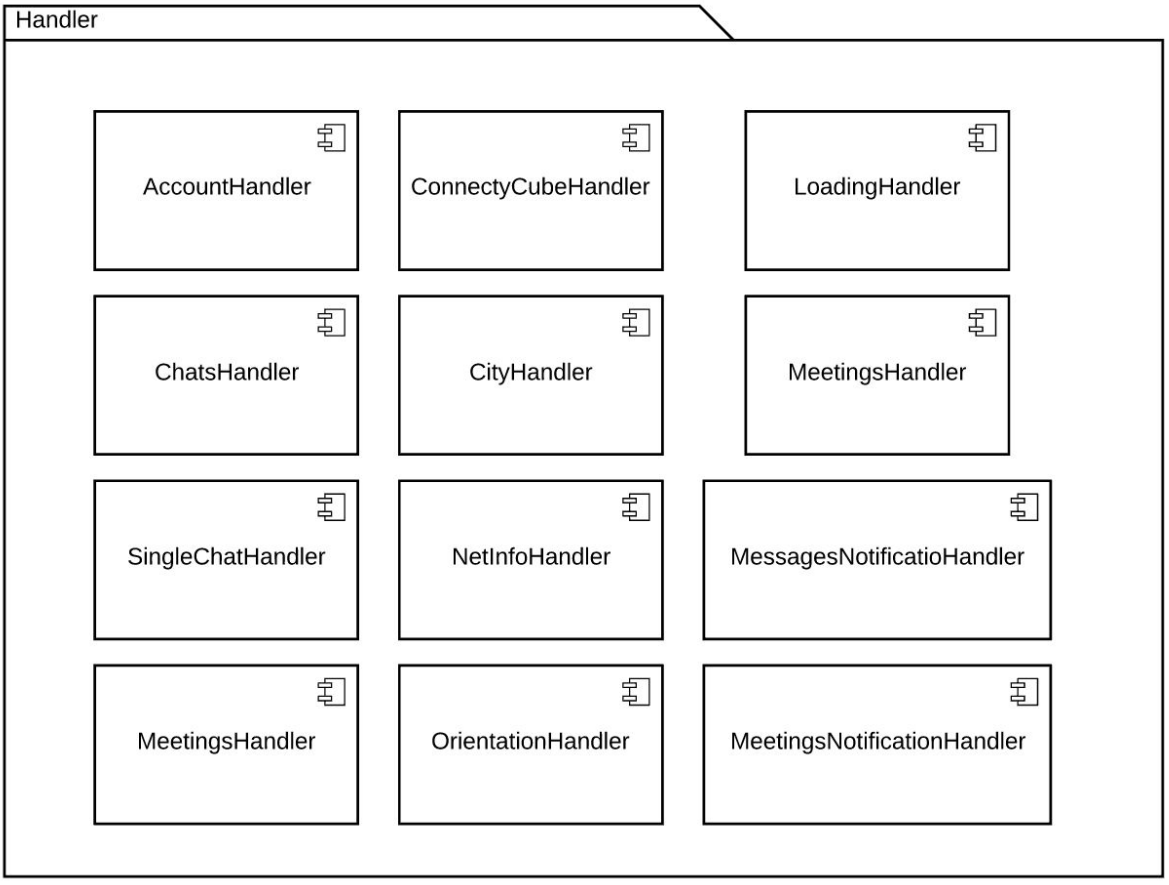
5.2.2 Model Handler Components



Below, a brief description of each component:

- *LocalChatsHandler:* Manages the local state regarding the chats
- *LocalMeetingsHandler:* Manages the local state regarding the meetings
- *LocalUserHandler:* Manages the local state regarding the info of the user currently logged in and his preferences/settings

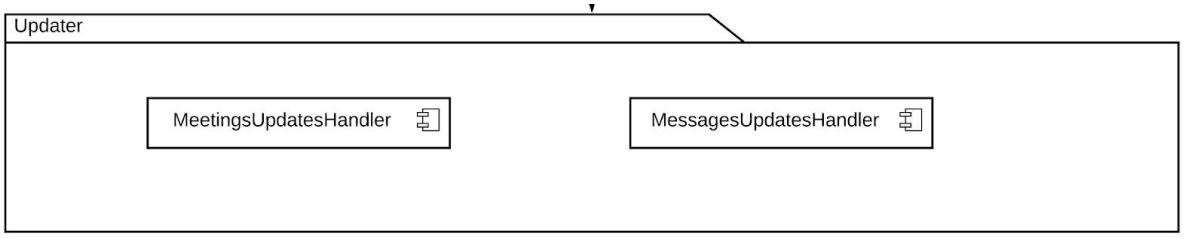
5.2.3 Backend Handler Components



Below, a brief description of each component:

- *LoadingHandler:* Used by Loading component to handle the initialization phase of the app
- *ConnectyCubeHandler:* Used to interact with ConnectyCube to handle the chat
- *AccountHandler:* It interacts with the backend in order to manage the authentication and the user account
- *ChatsHandler:* It interacts with ConnectyCube in order to manage the chats
- *SingleChatHandler:* It interacts with ConnectyCube in order to manage the single dialog
- *MeetingsHandler:* It interacts with the backend in order to handle meetings (creation, deletion)
- *CityHandler:* Used to retrieve info about cities
- *MeetingsNotificationHandler:* Used to handle the notifications regarding the meetings (new meetings, accepted meetings, denied meetings)
- *MessagesNotificationHandler:* Used to handle the notifications regarding the chats (new messages)
- *NetInfoHandler:* It handles the infos about the connection of the device
- *OrientationHandler:* It handles the infos about the orientation of the device

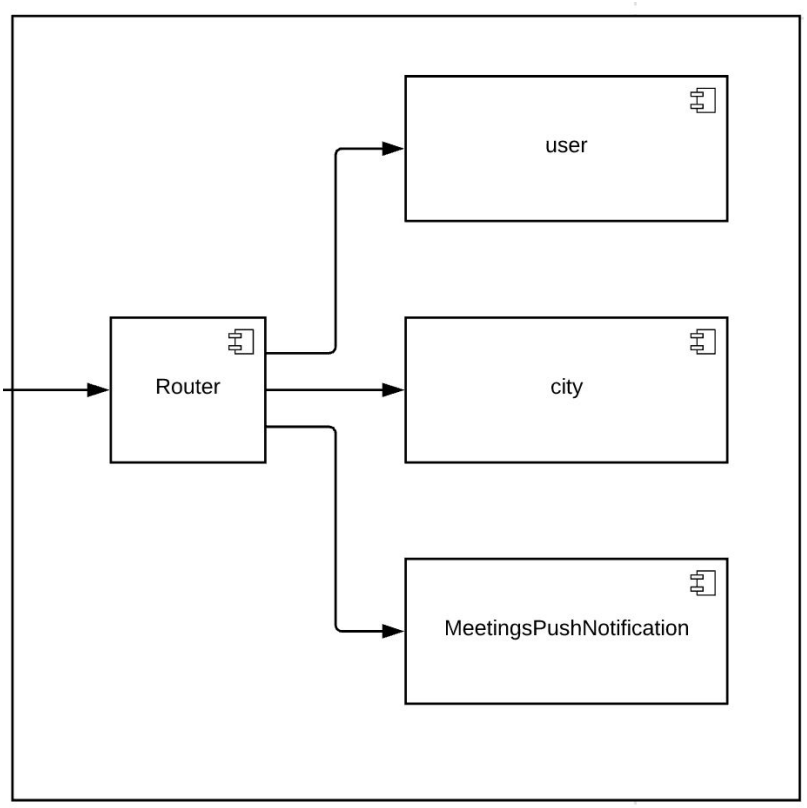
5.2.4 Updater Components



Below, a brief description of each component:

- *MeetingsUpdatesHandler:* Updates the client state when the are updates regarding the meetings
- *MessagesUpdatesHandler:* Updates the client state when the are updates regarding the chat

5.2.5 Server Components



Below, a brief description of each component:

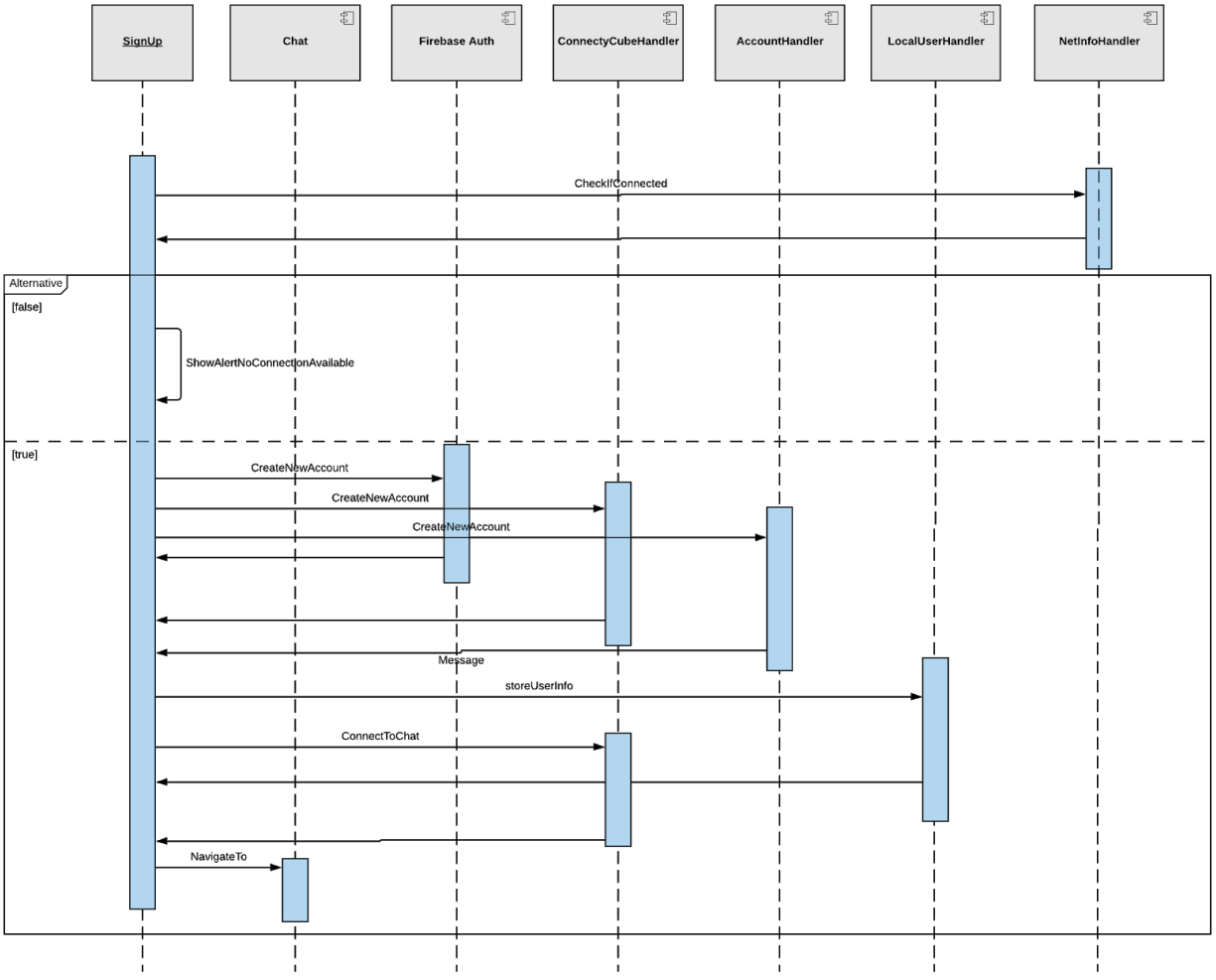
- *Router:* Starting point of a request that arrives on the backend. It dispatches the request to the right handler
- *User:* It is the component that interacts with the database in order to handle all the requests about a user
- *City:* It is the component that interacts with the database in order to handle all the requests about a city
- *MeetingsPushNotification:* It is the component that dispatches notifications regarding meetings.

5.3 Runtime view

The following diagrams illustrate how the main components of the system interact when using the most common features of the application.

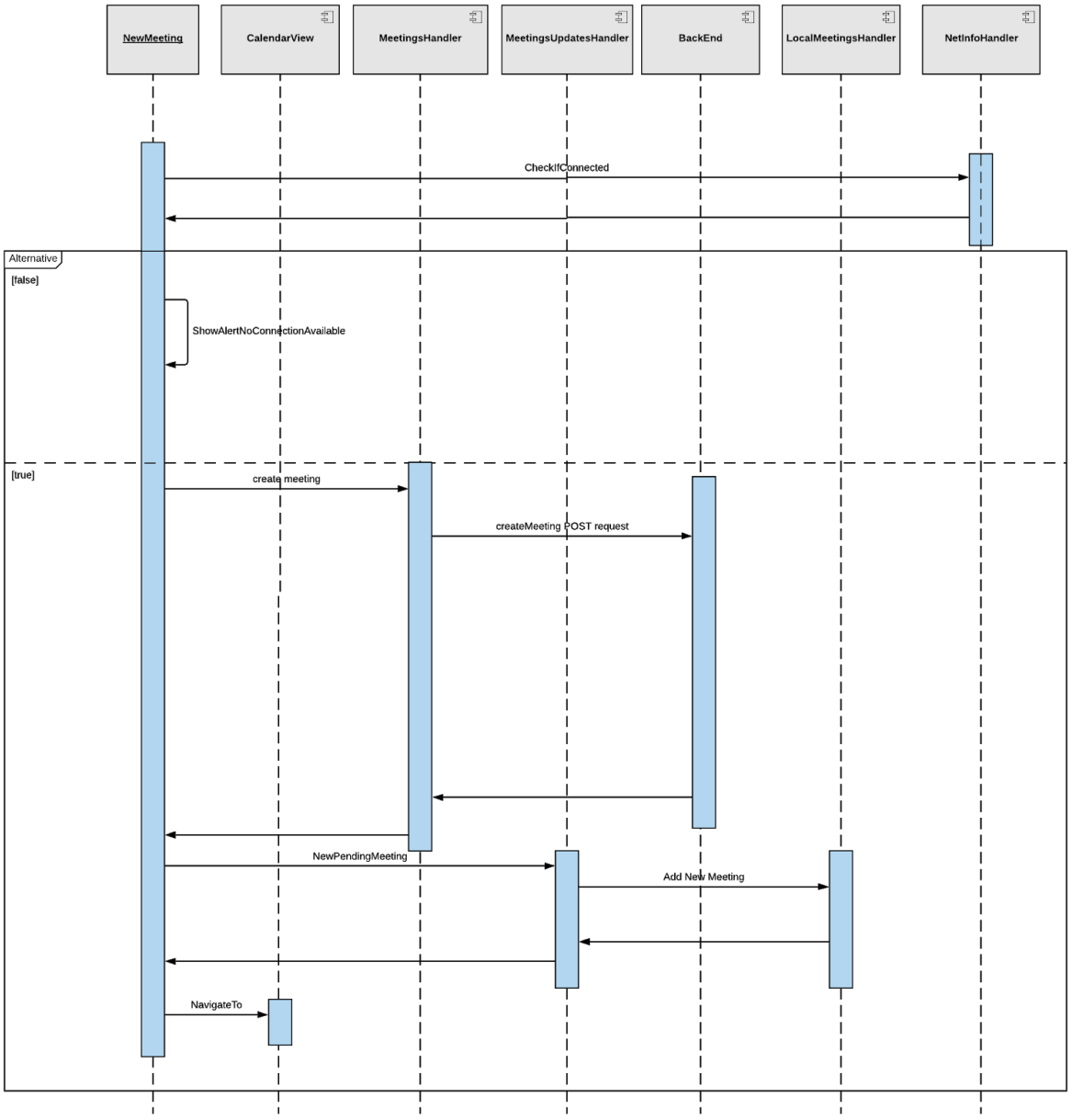
5.3.1 Sign Up

SignUp procedure will start after the user has filled all the required fields (in case the user has missed one field, it will be thrown an error). First thing to check is if the user is connected or not.
If it is not, it will be thrown an alert.
If it is, it will be send a request to Firebase Auth, ConnectyCube and on the application server to create a new account with the user credentials. Then, the user info will be stored in local to increase the performance. The ConnectyCubeHandler will connect the user to the chat and, at the end, the user is redirected to the Chat screen.
In the event of an error, the system will show an error message.



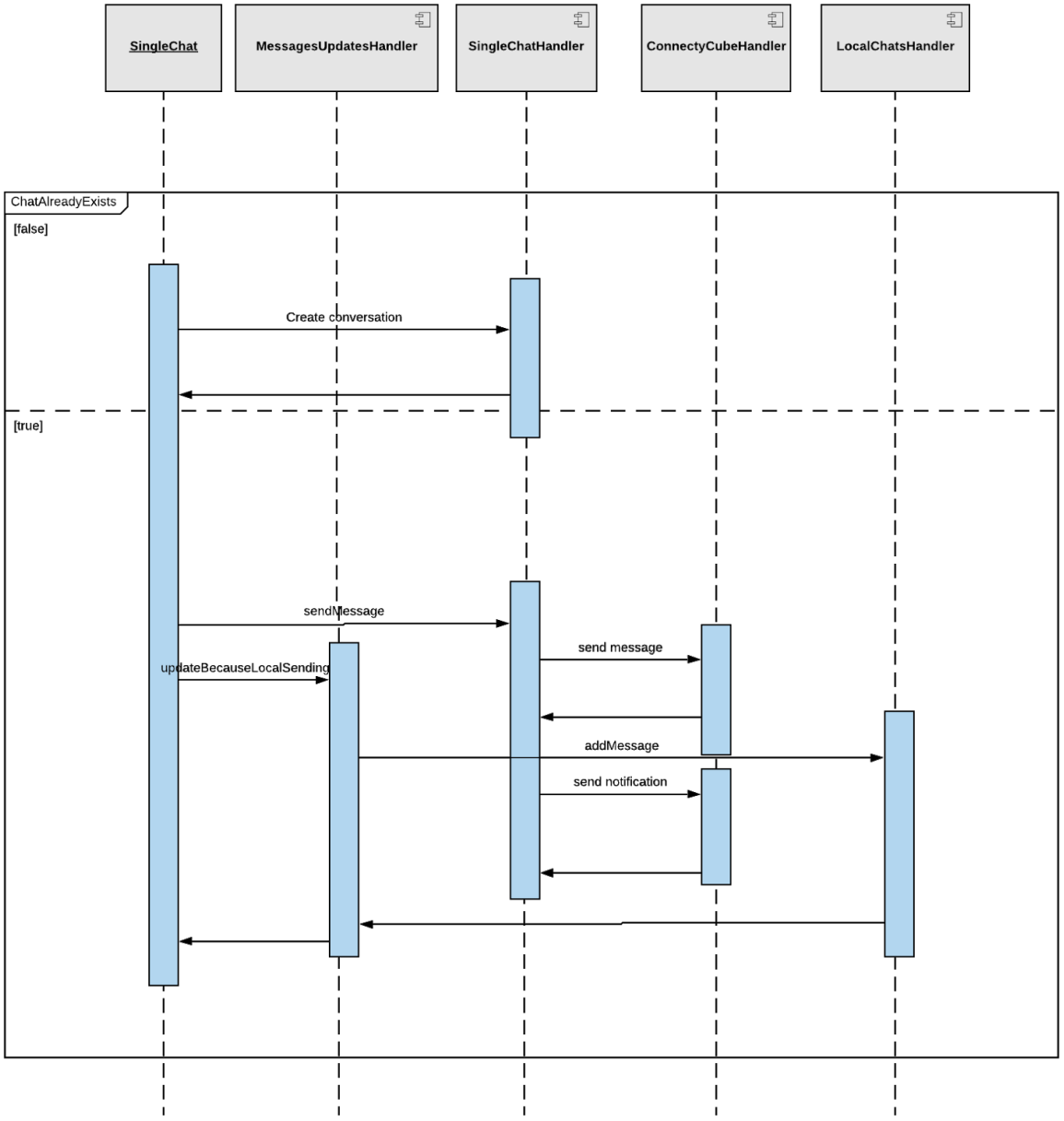
5.3.2 Creating an Appointment

An appointment is created if the user is in the NewMeeting screen and he clicks on the “Save” button. He has to choose another user, the date and the time of the meeting. If one of the fields is empty, the system will throw an error.



5.3.3 Send a message

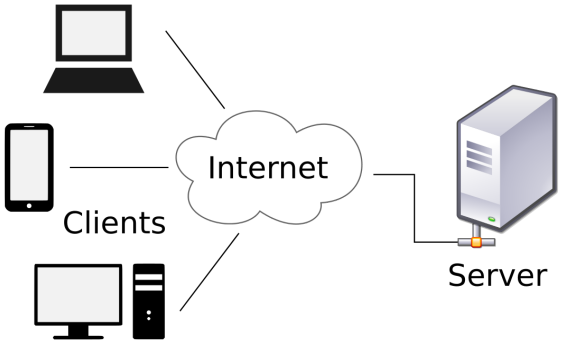
A message is sent when the user clicks on the proper button in SingleChat screen.



5.4 Selected Architectural Styles And Patterns

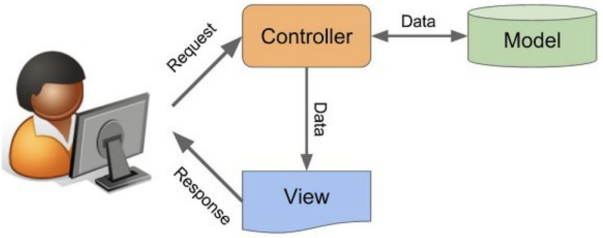
5.4.1 Client-Server

As for all the distributed applications, also LocalBuddy will be implemented using the Client-Server pattern. This means that there will be a client, i.e. the final user, who requests a service and a server, i.e. a provider of the service requested.



5.4.2 Model-View-Controller

It maps on the architecture shown in section [2.1 - Overview]. The Model in this application is composed by all the model handler components described in [4.2.2 - Model Handler Components] , while the Controller contains all the components shown in section [4.2.3 - Backend Handler Components]. The View includes all the components concerning the GUI that are described in [4.2.1 - Screen Components].



The image above explains how it will work. The client inserts or requests some information, managed by the Handler. The handler, then, will interact with the model and will pass to the View the informations to show.

We have chosen this type of architectural pattern because, in addition to being widely used, it allows:

- simultaneous independent development of different sections. This means that there is a complete division between the business logic and what is shown.
- easiness of modification of code and bug fixing
- last, but not least, multiple views for a model: this allow us to easily change the GUI without affecting the Model.

5.4.3 XMPP

Extensible Messaging and Presence Protocol (XMPP) is a communication protocol for message-oriented middleware based on XML (Extensible Markup Language). It is the de-facto standard in chat applications. It enables the near-real-time exchange of structured yet extensible data between any two or more network entities.

5.4.4 Publish Subscribe

Publish–subscribe is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without knowledge of which subscribers, if any, there may be.
Similarly, subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of which publishers, if any, there are.

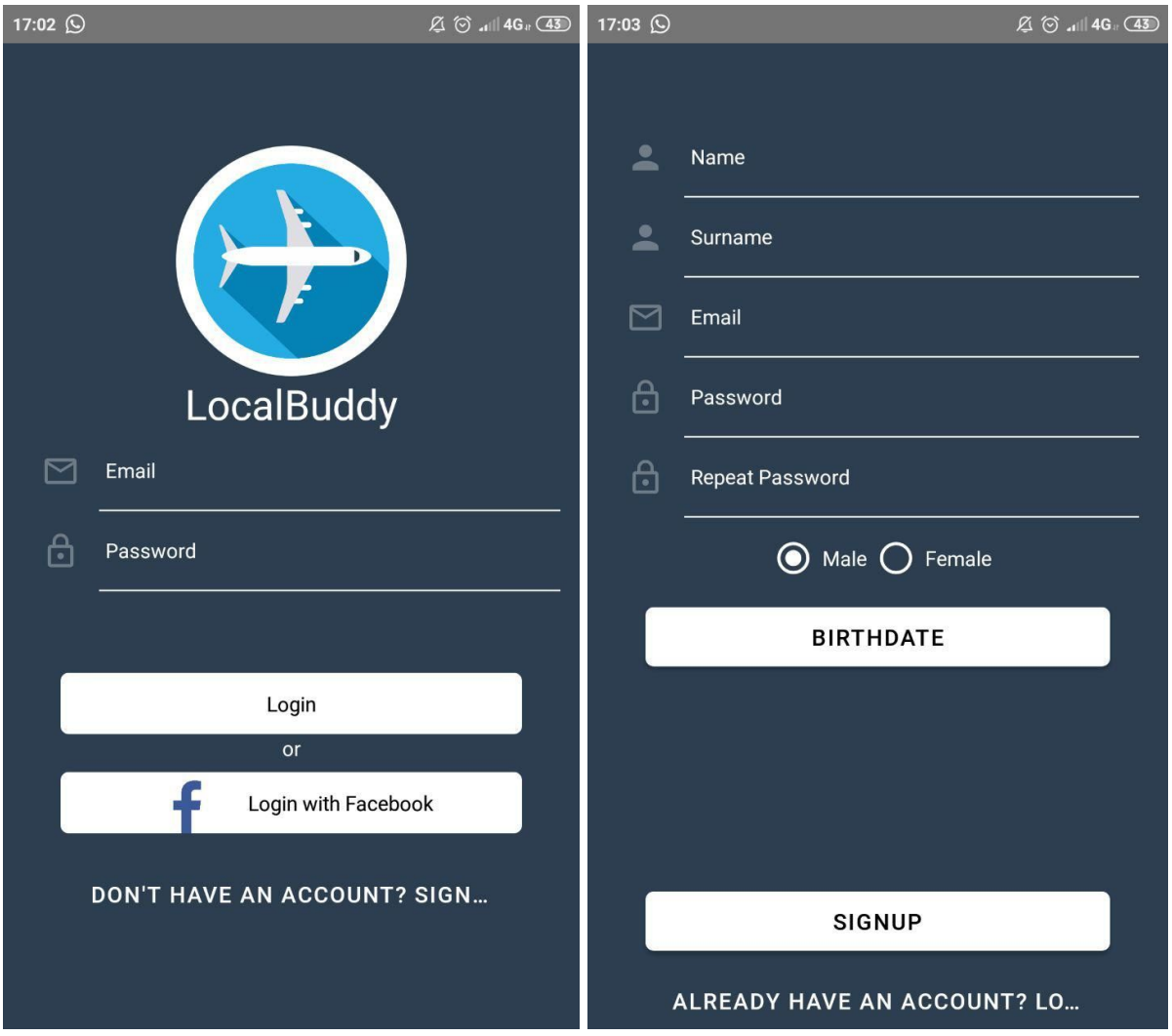
We will use this pattern several times in order to update the state of the components that are already rendered and mounted in react-native when there is un update, e.g.: a new message or a new meeting.

6. Specific Requirements

6.1 User Interfaces

Below are shown user interface both for smartphone and for tablet of the LocalBuddy.

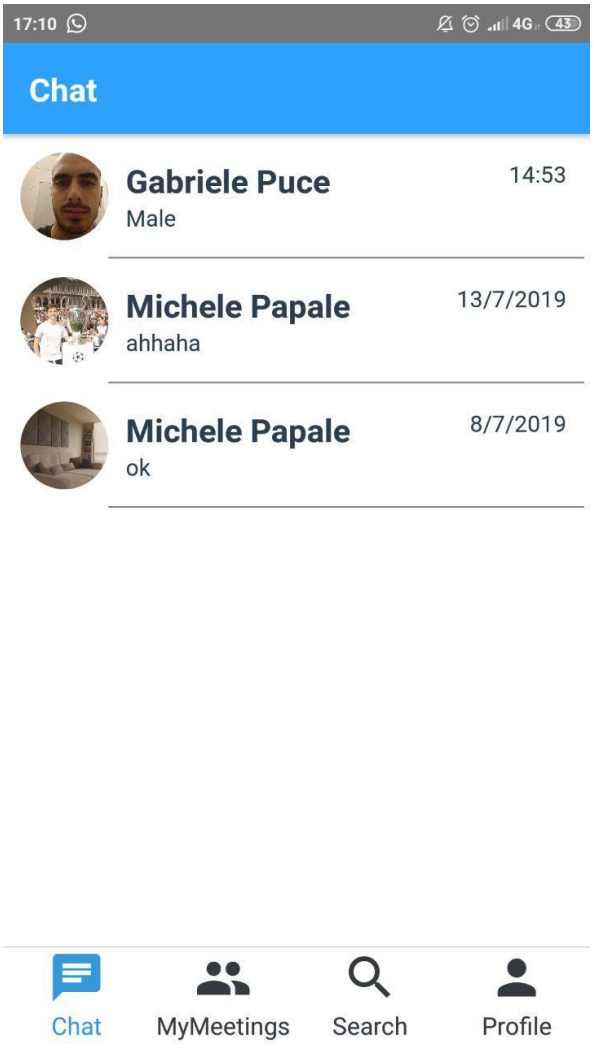
6.1.1 Smartphone



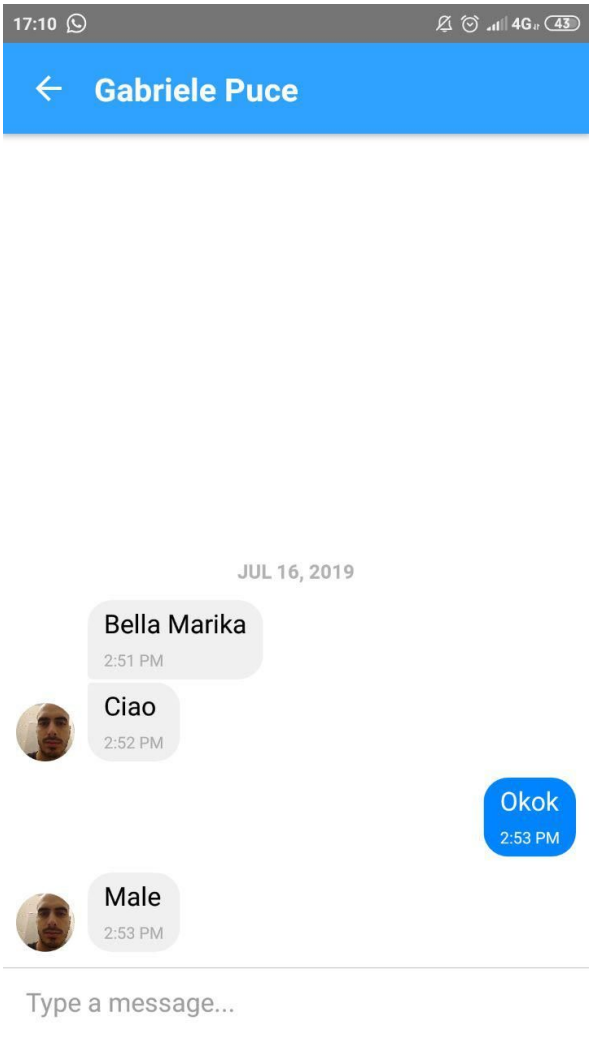
A B

A. The login page allows users and visitors to sign in through the provided social networks. Users can sign providing the credentials (email and password), previously registered when signing up. Visitors can also access the signup page.
   Accessible by:
   a. Visitor
   b. Registered User

B. This is the LocalBuddy SignUp page. From this page the user can signUp. If the user does not fill all the fields in a proper way, the system will prompt an error message.



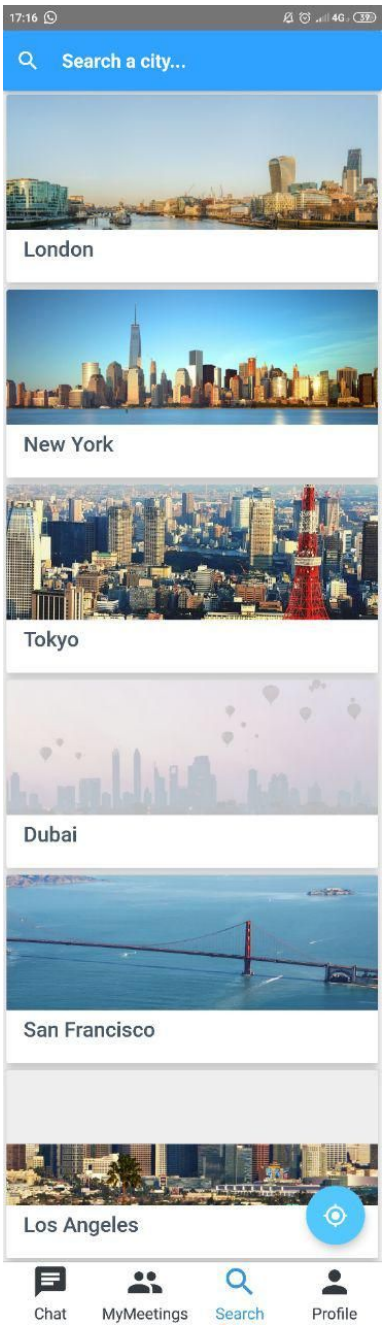
C



D

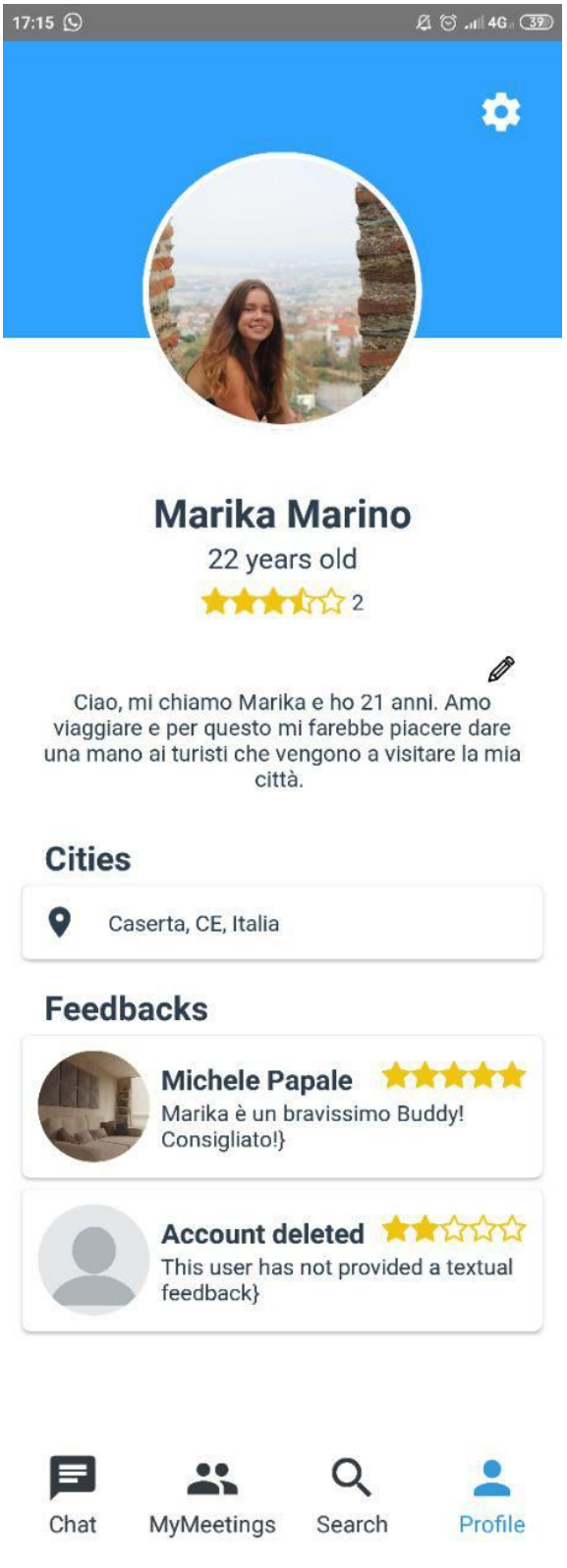
C. This HomePage of LocalBuddy. It shows all the chats the user has. Clicking on a chat, the app will redirect the user in D.

D. This is the SingleChat page, the page where users chat chat.

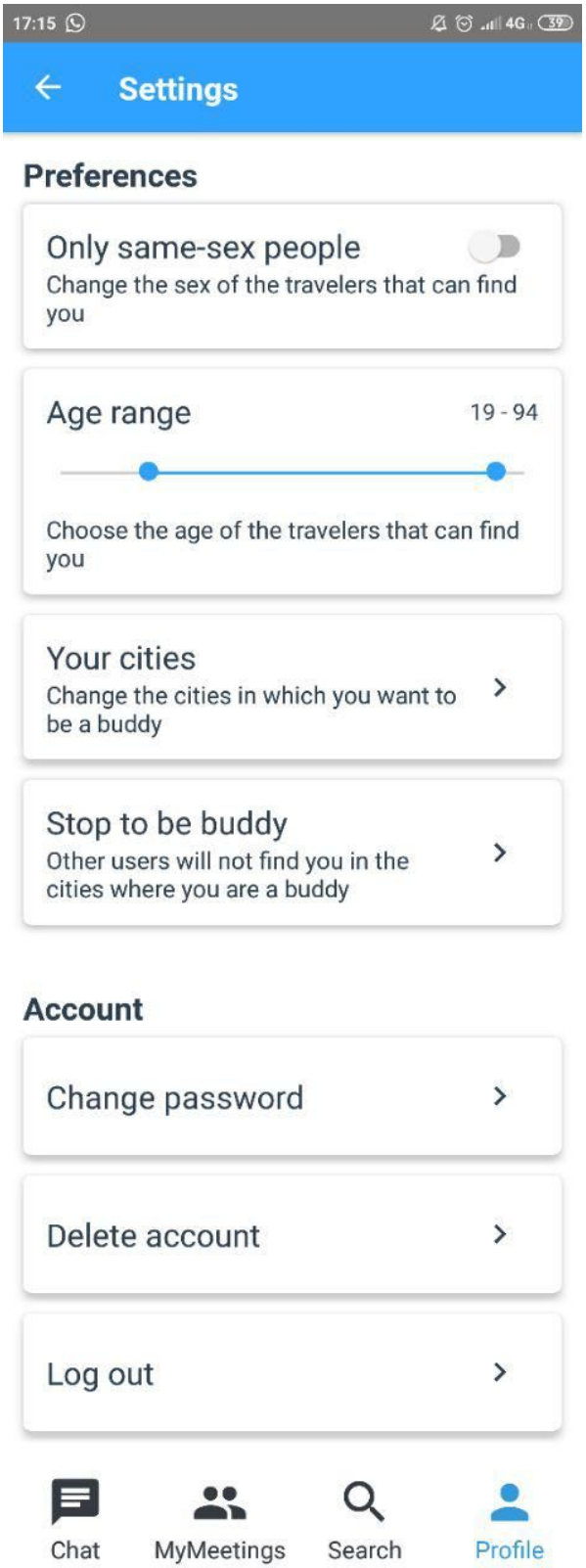


F. This screen shows how will be printed the cities that matches the string the user has inserted in the chat.

E. This is the ChooseCity Page. From this screen, the user can select a predefined city, or it can search a city by searching through the searchbar. Users can find buddy also by clicking on the gps button, it redirects to a page containing all the buddies that are present in the same city of the user.



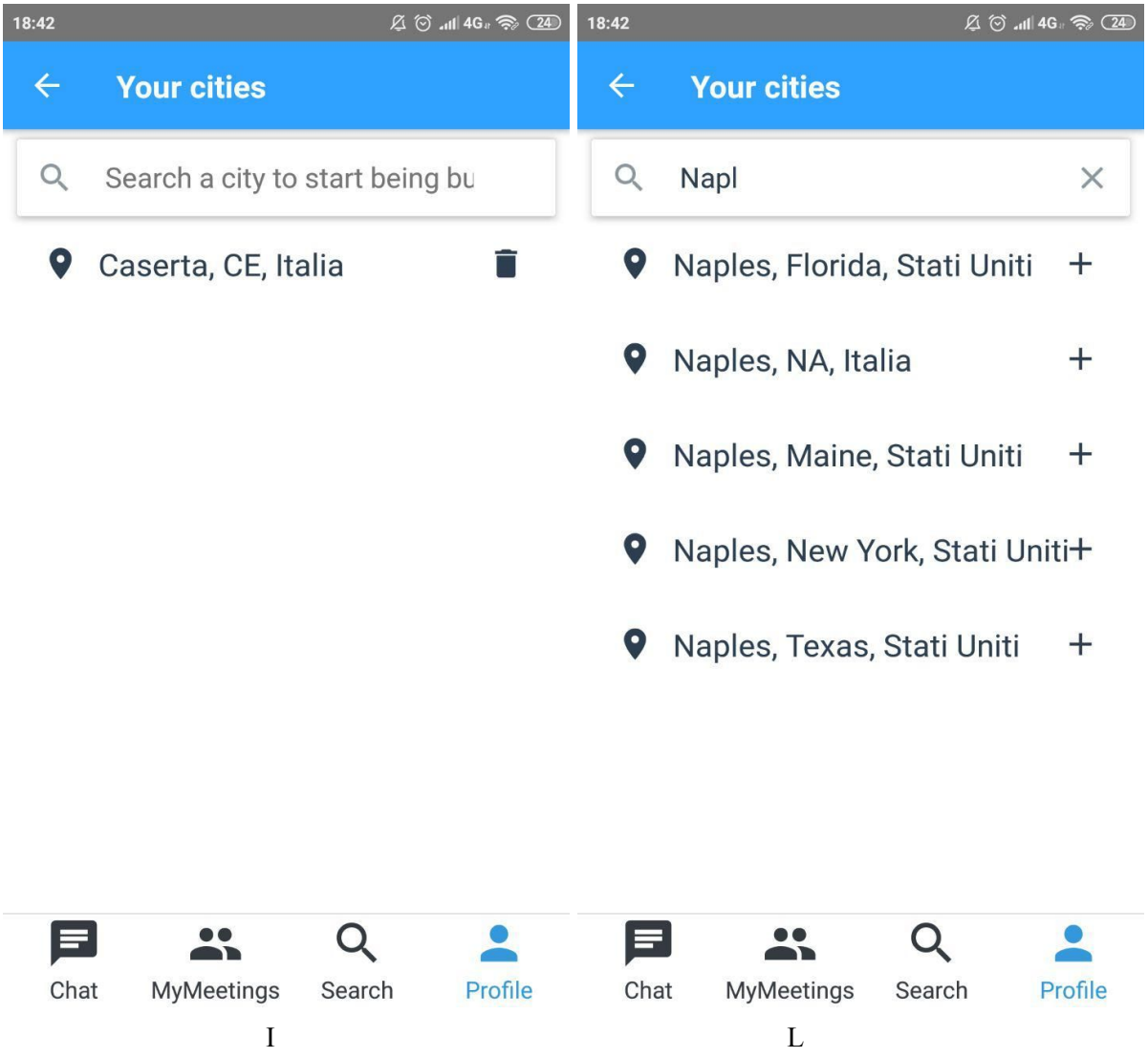
G. This is the user profile.By clicking on the user image, the user can update



H. This is the user's Setting Page. From here the user can edit the application

his photo profile
using a new photo or a photo taken
by the gallery.

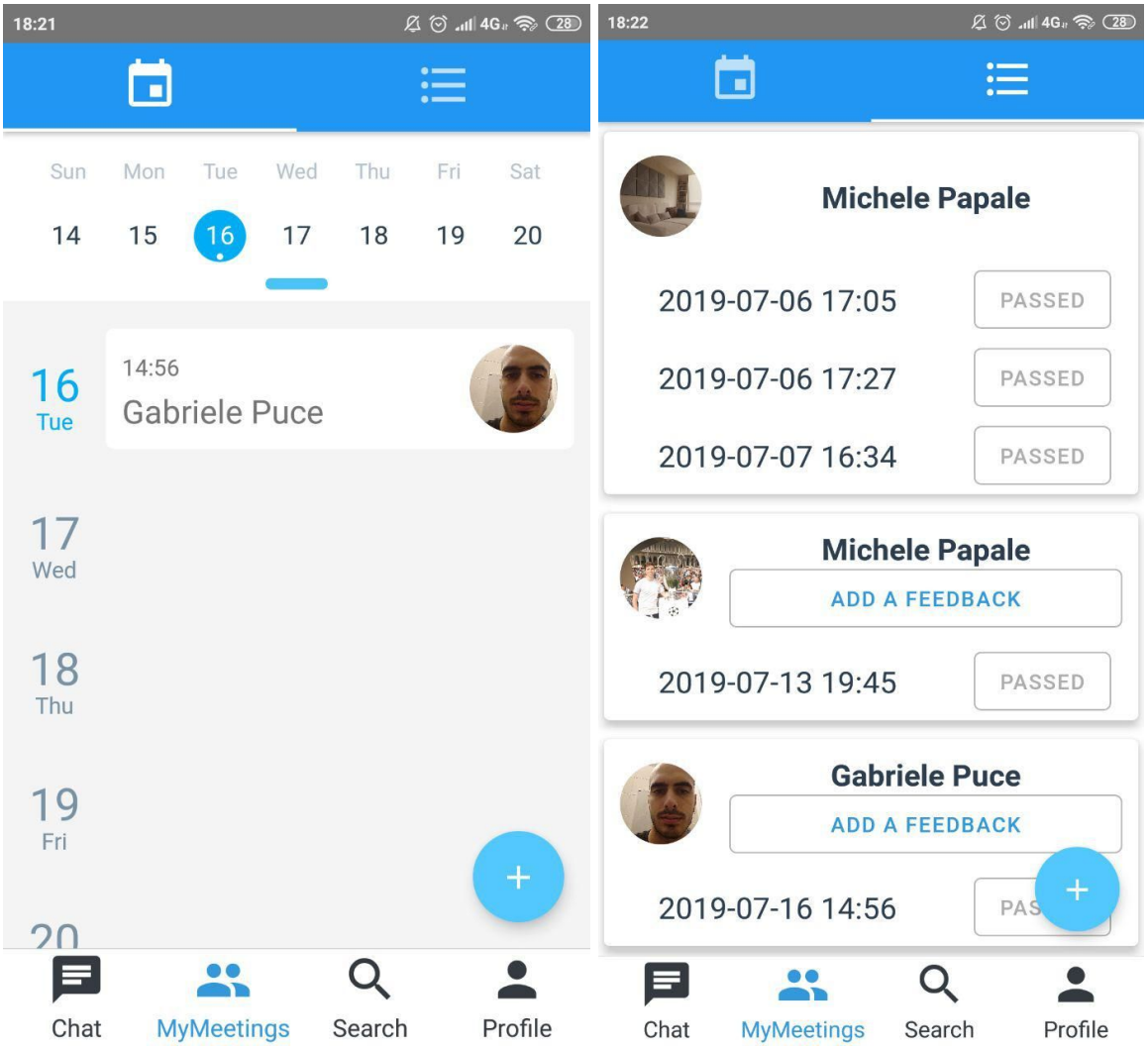
Settings and his personal Preferences.
Tapping on Your Cities will be redirected to I



I L

I. This is the CitiesOfTheBuddy screen. Here, the user can see all the cities where he is buddy. Clicking on the bin, he will remove himself on the relative city. Searching a new city, the screen will be updated by the system showing the cities that match the name inserted.

L. This is the CitiesOfTTheBuddy screen where the user has inserted in the searchBar “Napl”.



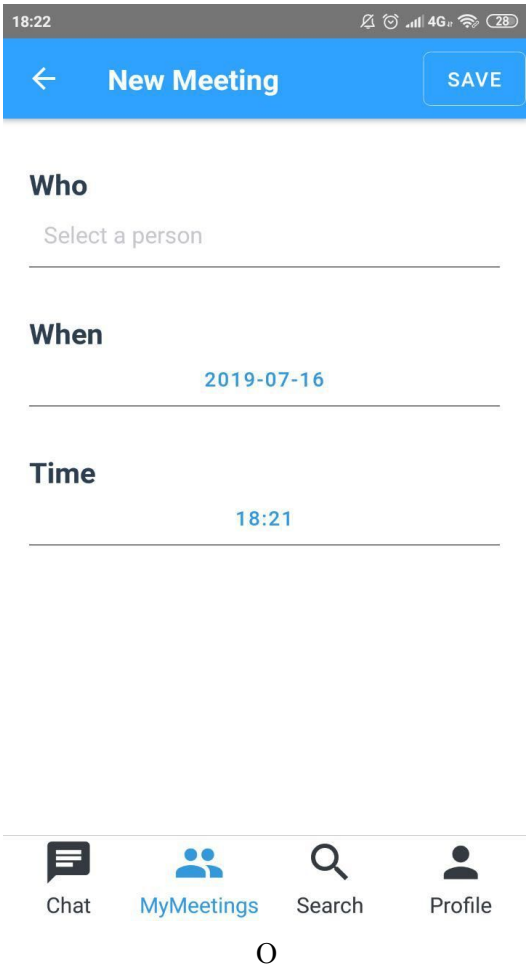
M N

M. This screen shows the user's daily schedule. From here the user can select the appointments to view detail (eg. description). Appointments are ordered chronologically (A timeline is displayed on the left for navigation).

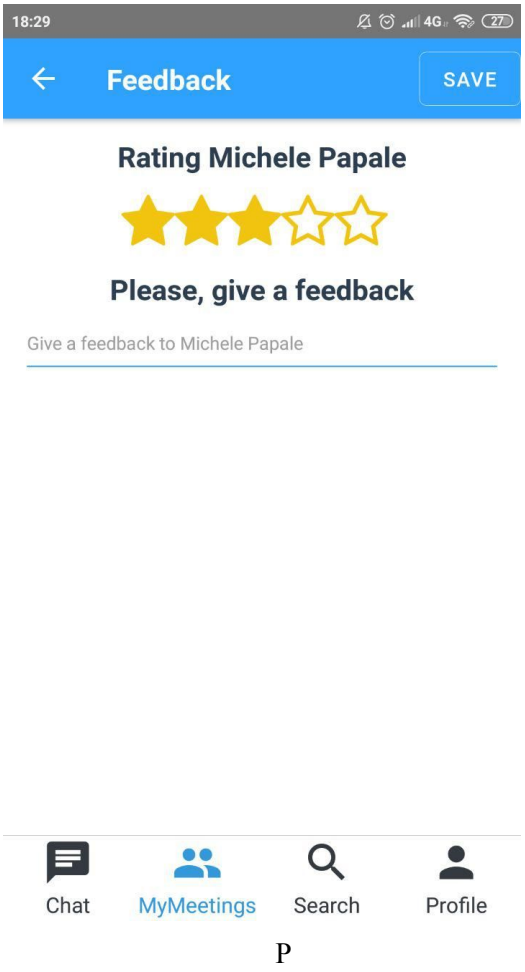
Accessible by:

a. Registered User

N. This screen shows the user's appointments in a "per buddy" view. From here it can also click on the "Add a feedback" button in order to give a feedback to the other user.



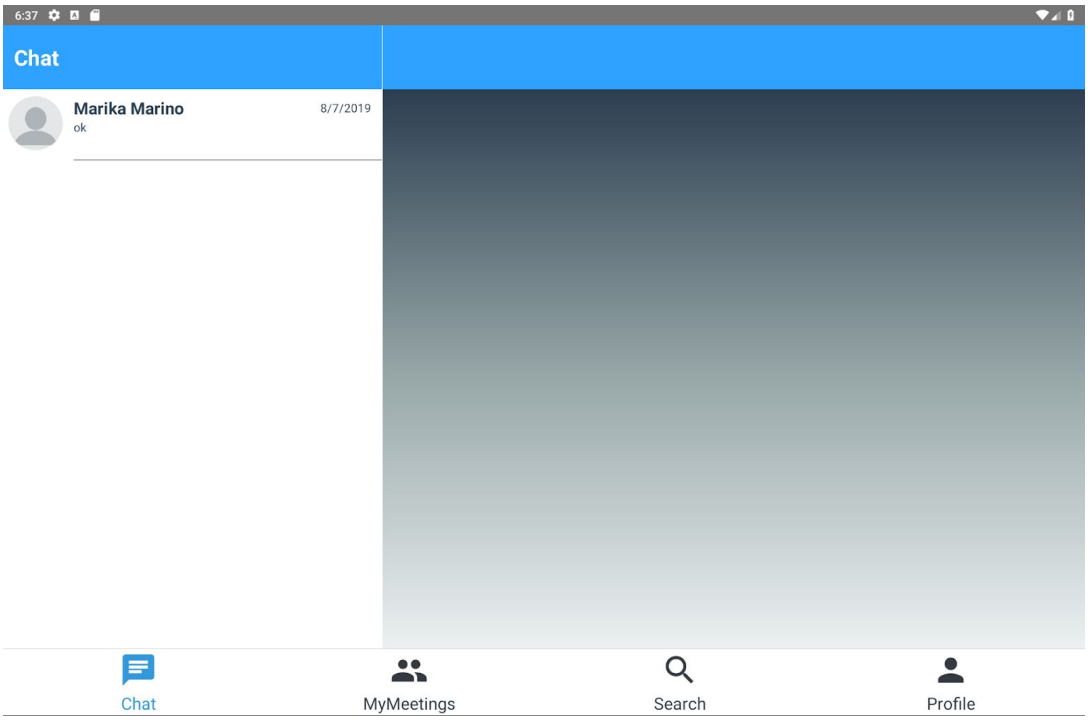
O



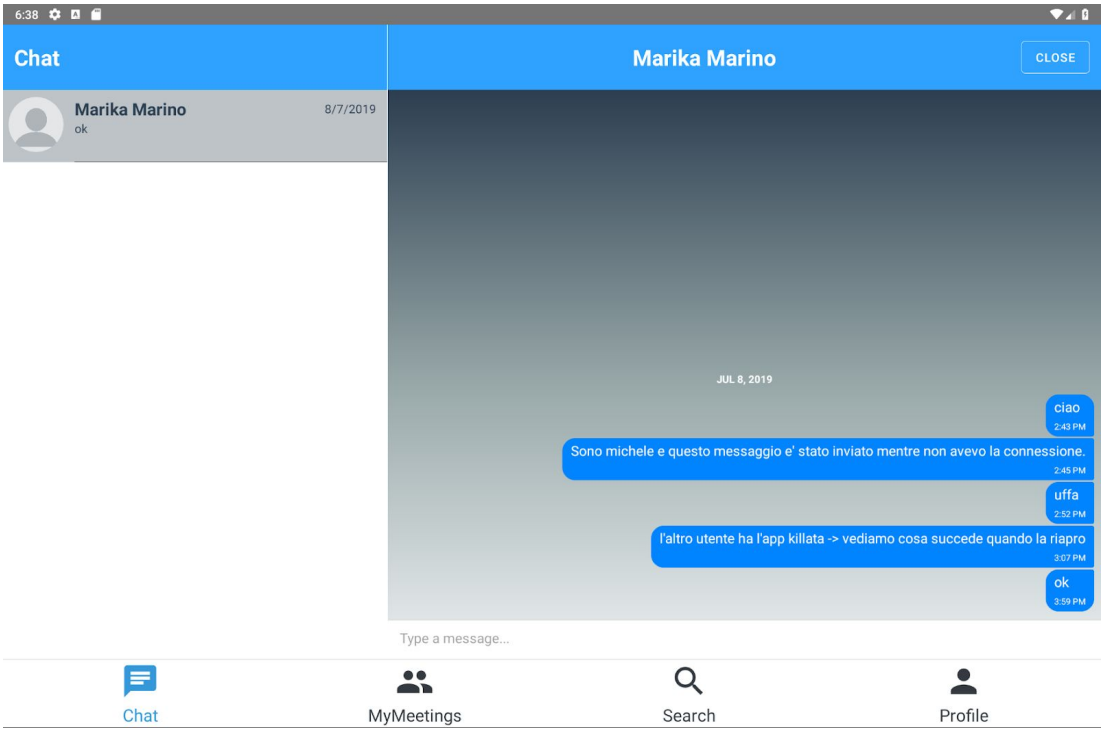
P

O. This screen shows a NewMeeting page. The user has to provide the name of the other user, the date and the time.
P. This screen shows how a user can add a feedback to another user. He has to provide a rating and, optionally, a textual feedback.

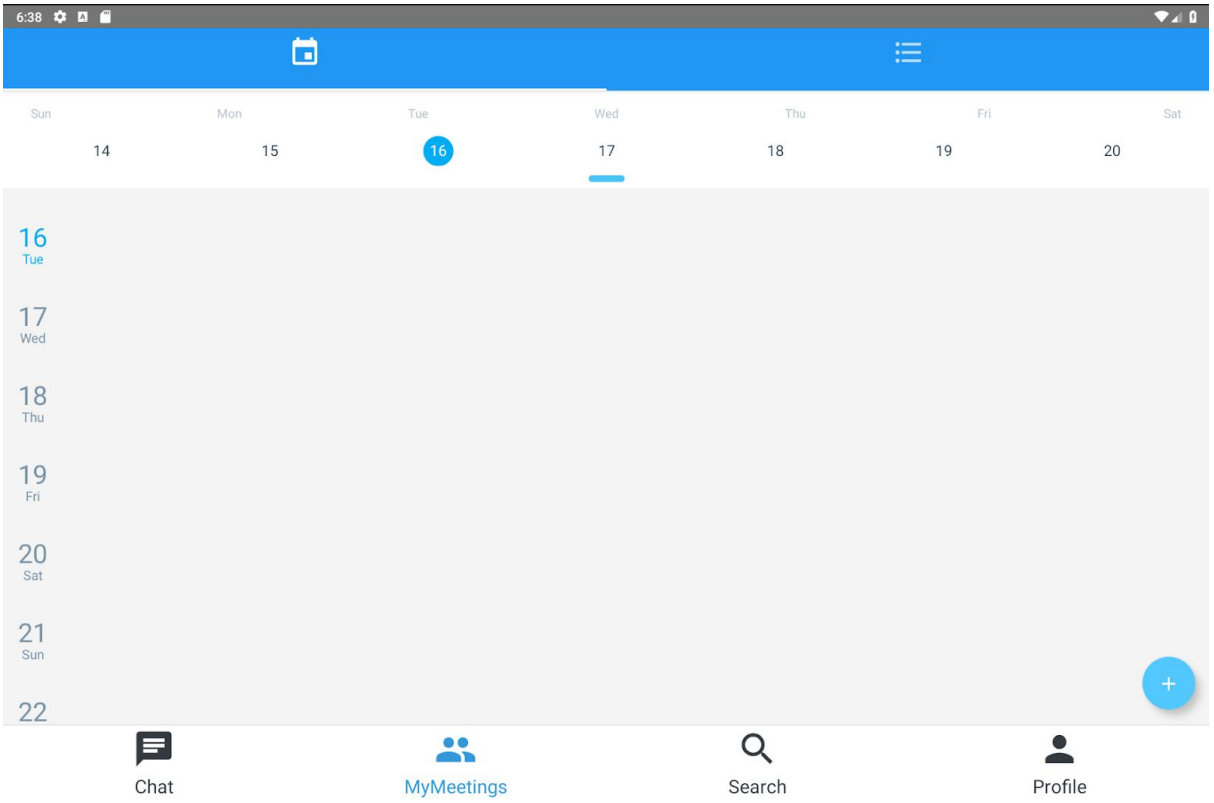
6.1.2 Tablet



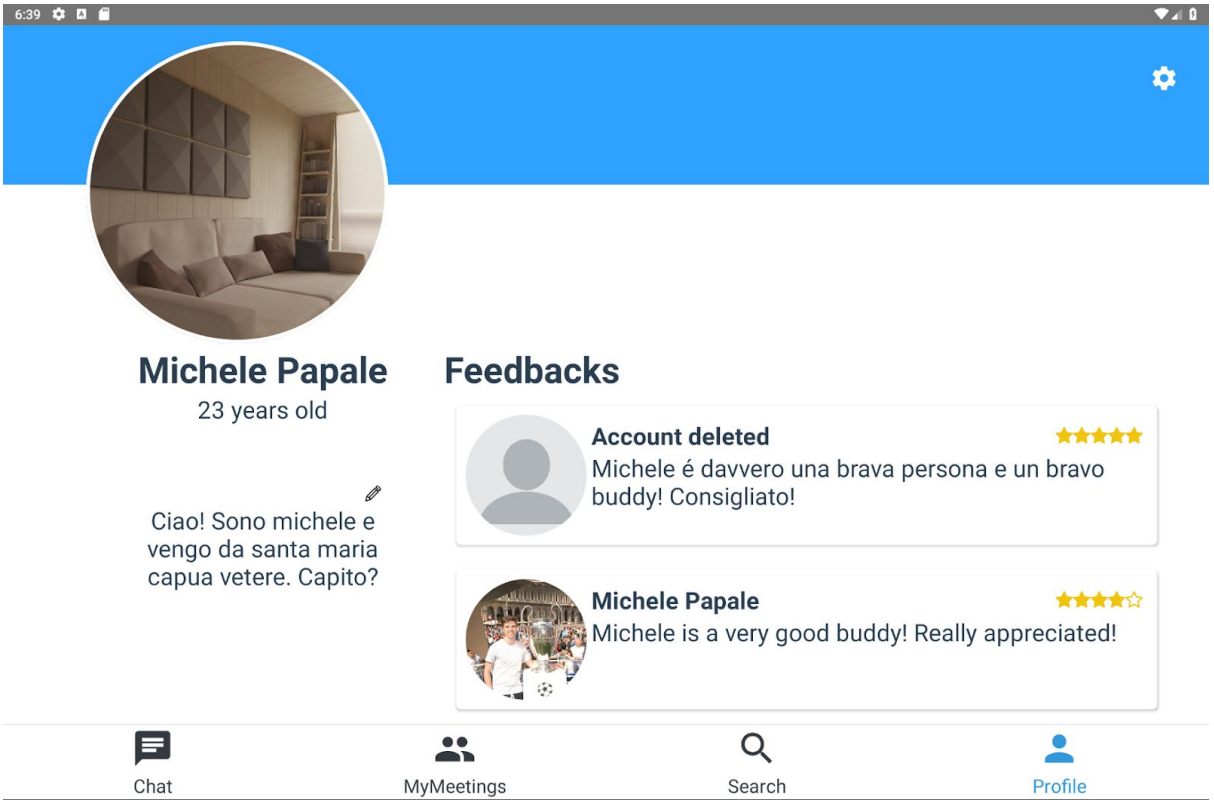
A. This is the homepage of LocalBuddy fom a tablet. It contains the list of the chats on the left hand side. On the right hand side, instead, there are the messages of a signle chat, if it is selected.



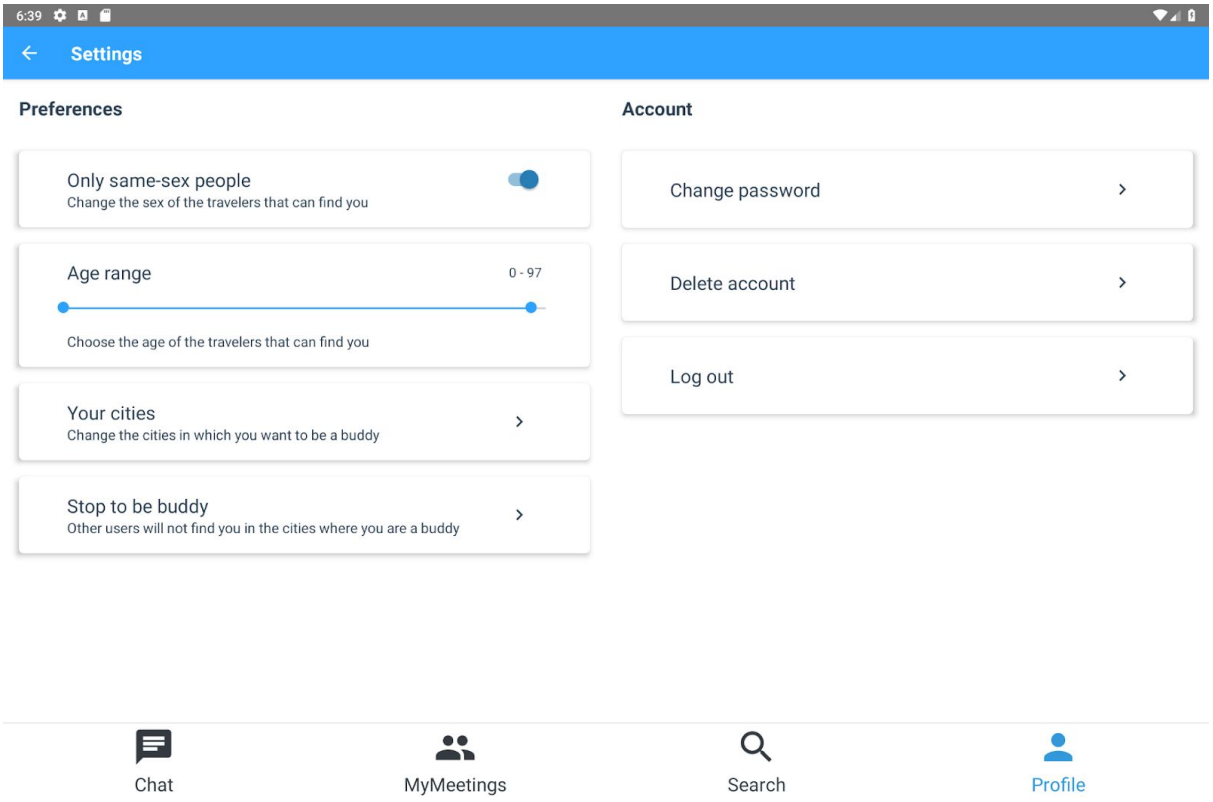
B. Screenshot that shows the homepage with a clicked chat.



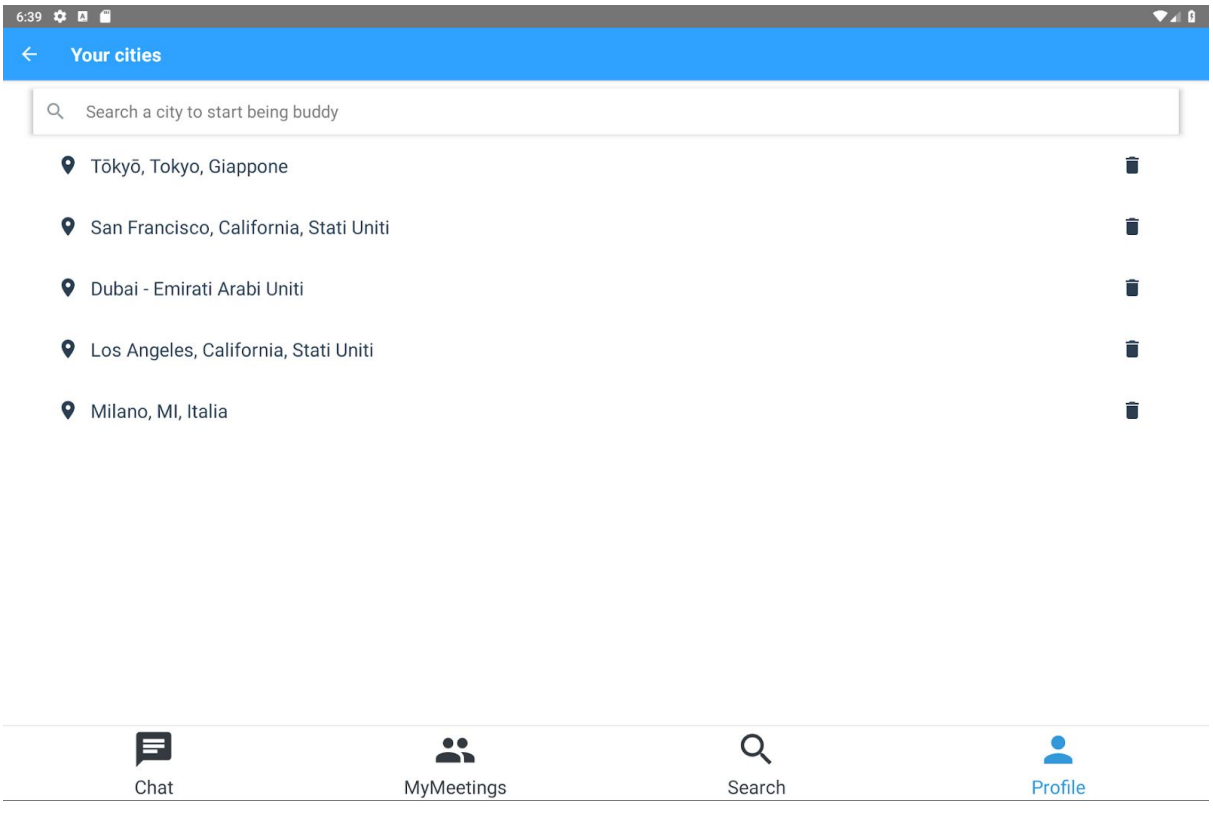
C. This screen shows the user's daily schedule. From here the user can select the appointments to view detail (eg. description). Appointments are ordered chronologically (A timeline is displayed on the left for navigation).



D. This is the user profile from a tablet view. By clicking on the user image, the user can update his photo profile using a new photo or a photo taken by the gallery.



E. This is the user's Setting Page. From here the user can edit the application Settings and his personal Preferences. Tapping on Your Cities will be redirected to F



F. This is the CitiesOfTheBuddy screen from a tablet view. Here, the user can see all the cities where he is buddy. Clicking on the bin, he will remove himself on the relative city. Searching a new city, the screen will be updated by the system showing the cities that match the name inserted.

6.2 Hardware Interfaces

To use all the functionalities of the application we will use a NoSQL cloud database in which all the data will be stored.

The clients, i.e. the users, are required to have a smartphone or a tablet available to use LocalBuddy.

7. External Services and Third Party Libraries

7.1 External Services

We expect to use the external services mentioned before for our application that are necessary for the proper behavior of the system. They are completely transparent to the user and fully integrated within the application. In this paragraph we give a brief description of each service and we explain the reason why we have decided to adopt them.

Facebook

We have decided to use Facebook to authenticate the user. The reason why we have chosen this social network as a way to authenticate users is pretty simple: it is the most used in the world.

Google Maps

We use Google Maps Autocomplete in order to complete the cities the user has starting typing. Although this service is not free, we decided to use this one because it is the best service that offers the possibility to find any city in the world.

ConnectyCube

When we started thinking what architecture was the best one in order to have the best user experience through the chat, we looked around and we became aware about the de-facto standard in this type of applications: XMPP. For this reason, we started searching for an external service that handles all the backend logic of a chat application, based on XMPP:

- 1-1 and group chats
- chat moderation
- video calls
- push notifications
- files storage
- users authorisation
- reauthentication

Firebase

Firebase is a mobile development platform. We chosen this service because it offers several products that we need. They are:

- Cloud Storage, in order to save in cloud the user photos

- Authentication, in order to authenticate the users on the platform
- Realtime Database, in order to save all the infos LocalBuddy will use to run properly
- Cloud Messaging, in order to send notifications about meetings
- Admin, in order to authenticate requests on the application server

7.2 Third Party Libraries

We will choose several third party libraries. Here, we list the most important:

- react-native-async-storage, in order to save in local the info of the user, meetings and chats
- react-native-netinfo, in order to know whether the user device is connected to the Internet or not
- react-native-calendars, in order to use the agenda view
- react-native-datepicker, in order to select the birthdate in sign up phase or to select the date and the time of a new meeting
- react-native-geocoding, a geocoding module for React Native to transform geographic coordinates (i.e. latitude and longitude) into a description of a location (i.e. street address, town name, etc.). This feature will be used when the user wants to find the buddies closer to him.
- react-native-gifted-chat, a component to display the chat between two users.
- react-native-image-picker, to upload an image (that the user will use as a photo of his profile).
- react-native-push-notification, to show notifications coming from firebase

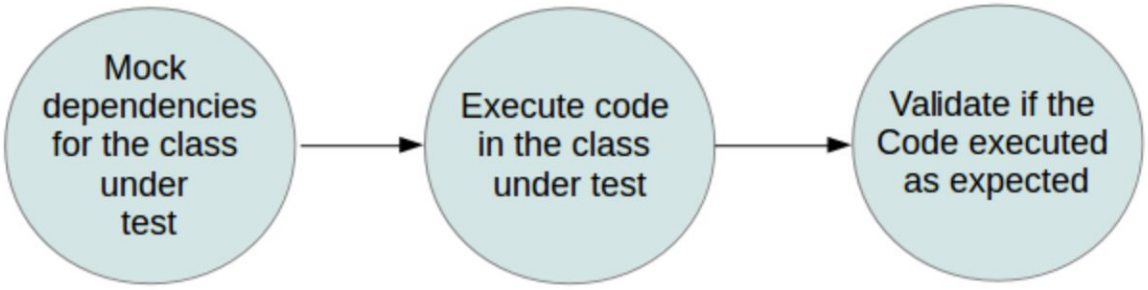
8. Implementation, Integration and Test Plan

For the implementation and test we have decided to follow an incremental bottom-up approach because it facilitates bug tracking and improves reusability. Furthermore, we will use an **extreme/agile** approach in order to build complete small portions of the app. We will develop all the functionalities in this order:

- Authentication
- Profile of the user
- Possibility to choose a city
- Listing all the buddies in a city
- Handling the filtering of the buddies
- Chat
- Meetings
- Notifications
- Feedback
- Handling the Local State in order to improve the performance
- Handling non-connected case
- GPS
- Handling Tablet View

Each time we will add a new feature to LocalBuddy we will test it. Tests will be performed through Jest.

Jest is a JavaScript Testing Framework able to support React (hence, also React Native). Jest simplifies testing for methods that have a lot of external dependencies. It allows to create unit tests with mock objects and functions, which are fake objects of given classes/functions. Then, after performing the tests, it allows to validate if they executed as expected.



We have done tests that show the correctness of the methods we created to handle the logic of the app.
As you can see below, we tested:

- the addition of a user
- the deletion of a user
- the addition of a feedback to a user
- the addition of a buddy to a city
- the deletion of a buddy from a city
- the creation of a new meeting