POLITECNICO
MILANO 1863

Reservation

Design and Implementation of Mobile Applications course
Professor Luciano Baresi

WRITTEN BY RICCARDO FACCHINI
MAT: 899190

2018-2019

Deliverable specific information

Deliverable:	Design Document
Title:	Design Document
Authors:	Riccardo Facchini
Version:	1.0
Last revision:	February 17, 2019
Download page:	https://github.com/Riccardo95Facchini/Reservation.git
Copyright:	Copyright © 2019, Riccardo Facchini – All rights reserved

Contents

List of Figures

1 Introduction

1.1 Purpose

This document aims to detail the design of the software and of the architecture regarding the application Reservation. To do so it will be taken a more detailed approach for the description of each component and the overall architecture of the system.

1.2 Scope

Reservation is an appointment management application, designed to help the users with their daily routine by keeping track for them of their next appointments in registered shops and avoiding the hustle of queueing or trying to reach the shop managers on the phone.
The same application can be used by **customers** and **shop owners** to manage the reservations, for the former it will help with the actual reservation process and act as an agenda by keeping track of the next appointments taken, while for the latter it will display the next customers that reserved a spot while giving only minimum information to preserve the customer's privacy.

1.3 Definitions, Acronyms, Abbreviation

- DB: DataBase
- DBMS: DataBase Management System
- GPS: Global Position System
- UID: Unique Identifier

1.4 Document Structure

This document is structured has:

1. **Introduction**, it provides an overview of the entire document.
2. **Architectural Design**, it describes different views of components and their interaction.
3. **Algorithm Design**, it describes the main algorithm and query method.
4. **User Interface Design**, it provides an overview about the aspect of the user interfaces of the system.
5. **Implementation, Integration and Test Plan**, it describes the orders in which the components are implemented and the order of the integration.
6. **Appendix**, it contains software used.

2 Architectural Design

2.1 Overview

The system to be designed needs to help customers make reservations in their favourite shops in the least amount of time possible and also keep track of all the future appointments they have taken.
It also must allow shop owners to register their establishment to the platform and keep track of the next customers that have registered an appointment.
Since this interaction between user and system can be summarize as:

1. User request a service to the system.
2. System responds to the user with the requested service.

Based on this, a client-server architectural approach has been chosen.

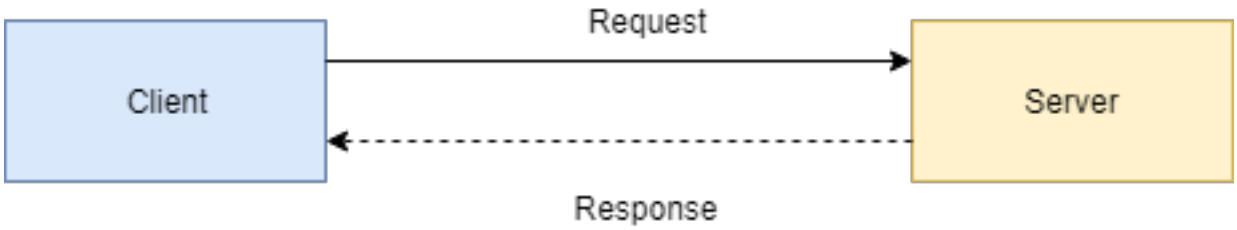


Figure 1: Client Server architecture

Furthermore, the system can be divided into three different subsystems: the presentation layer, the application layer and the data layer, where:

- The *Presentation Layer* provides the GUI of the system.
- The *Logic Layer* contains the logic of the application,that receives the requests from the user.
- The *Data Layer* stores and maintains the data needed from the system to works properly, i.e. user's & shop's information and reservations details.

2.2 Component View

2.2.1 Overview

In Figure 2 is possible to see the high level components of the system and the interfaces used to connect one to another, where

- *Firebase* provides the entry point for external resources;
- *Firestore* provides the database system;
- *Firebase Auth* provides the authentication system;
- The *Mobile Application* is the mobile application used by a user with his/her smartphone.

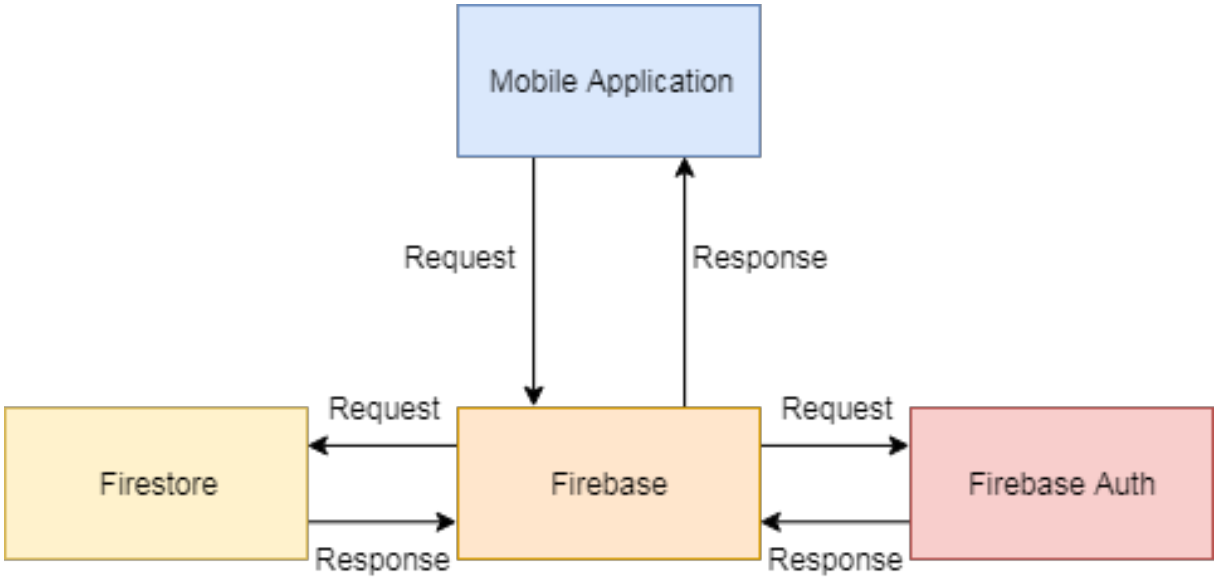


Figure 2: High level Component Diagram

2.2.2 Database View

Unlike traditional Entity-Relationship DB systems, Firestore is organized in Collections-Documents hierarchy that can be further nested, meaning a Document can contain one or more Collections containing other Documents and so on.
For this project only three Collections where used, each composed by a series of documents as shown in Figure 3

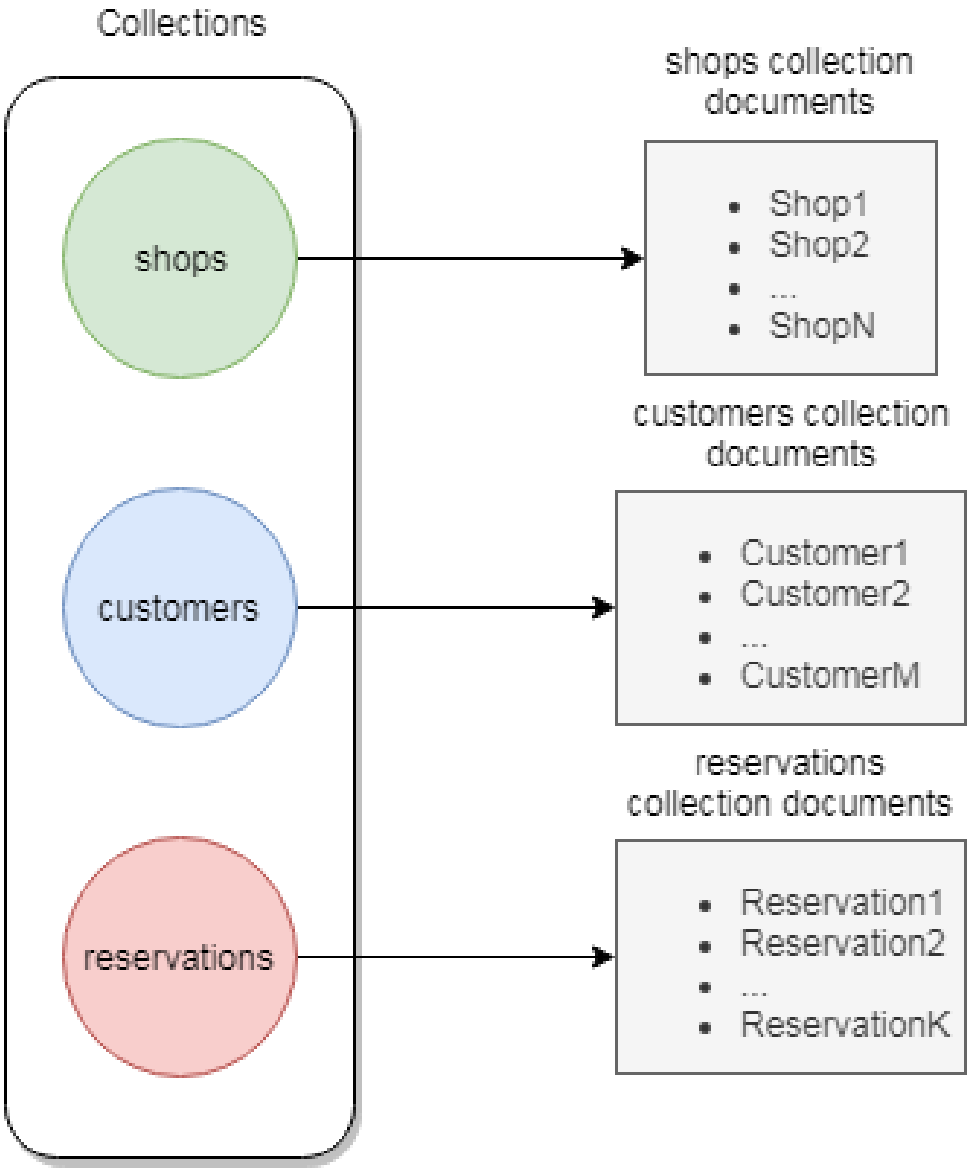


Figure 3: *High level* structure of the DB

While the three documents are shown as follows as a Java class would be. Shop Figure 4 and Customer Figure 5 are actually implemented in this way, while reservations Figure 6 differ depending on the type of user that is using the application, but only these information are needed to be recorded in the database.

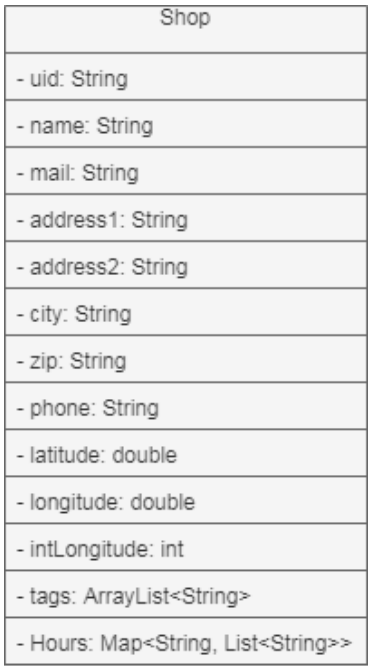


Figure 4: *Shop* document

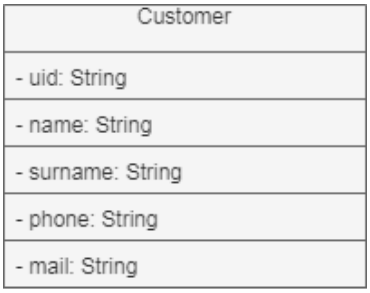


Figure 5: *Customer* document

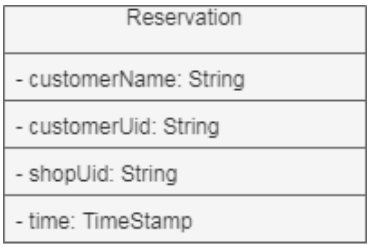


Figure 6: *Reservation* document

2.2.3 Firebase Auth View

The Firebase Authentication system handles automatically the authentication of the connected user by storing:

- Identifier (for example email address)
- Provider (Email/Gmail account/Phone etc...)
- Creation date
- Last sign in
- User UID (which is then used in the DB as an identifier)

2.2.4 Mobile Application

The *Mobile Application* is used by the user via its own smart device. The *Mobile Application* communicates directly with the Firebase system using the provided API.

2.3 Implementation choices

The technology chosen for the implementation on the system are all based on Java since it is the most common way of developing Android applications and the availability of documentation and other sources of learning materials are abundant and some experience was already obtained during past projects. It also offers the possibility of adding new functionalities in future, making the system more scalable given how the platform is constantly evolving
To summarize the technologies used:

Mobile application : entirely Android (Java) based with Firebase API added in order to use the external tools. Other modules such as RecycleViews and Cards have been added.

DBMS : Firestore was selected over the old real time database system since it has been developed as a successor to substitute it. I provides easy to use DB mechanisms, at the loss of complexity such as complex queries and it's not a relational DB system which is usually more familiar.

2.4 Runtime View

Here are represented the two most important runtime views by using diagrams that highlight the main actions the user has to follow in order to complete the given task.
In Figure 7 it's highlighted in red the interaction that the system has with the authentication system, then the two branches between choosing to register as a customer (green) or shop (blue) are available to be picked.
In Figure 8 are again highlighted in red the interactions of the application with the Firebase systems, while in blue is noted the optional choice to select a different distance than the default one.
It should be noted how in both cases none of the handled errors or problems are shown to keep the diagrams simple.

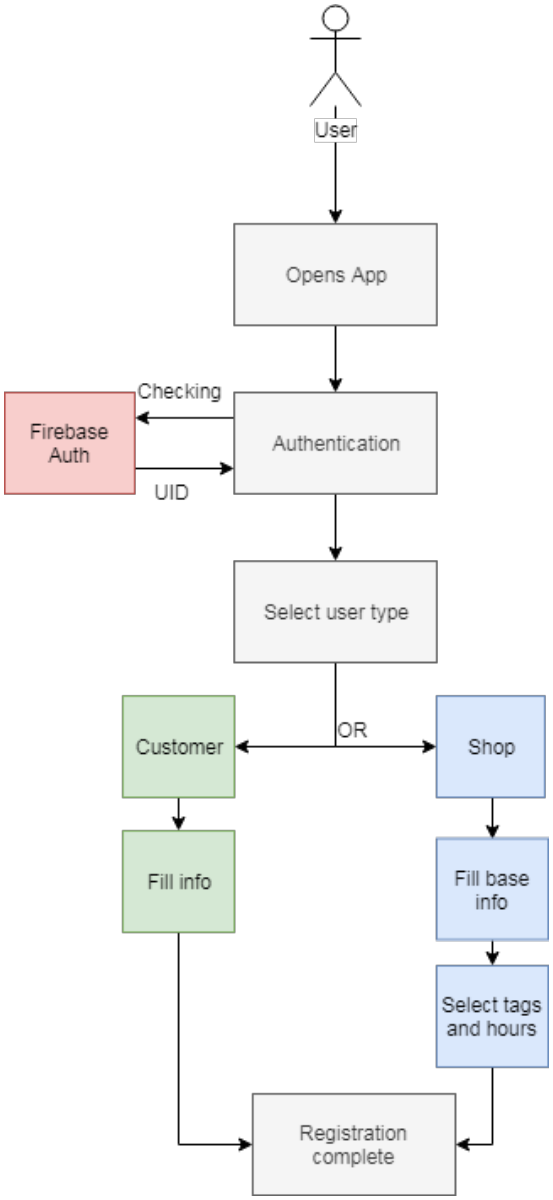


Figure 7: *User* registration diagram

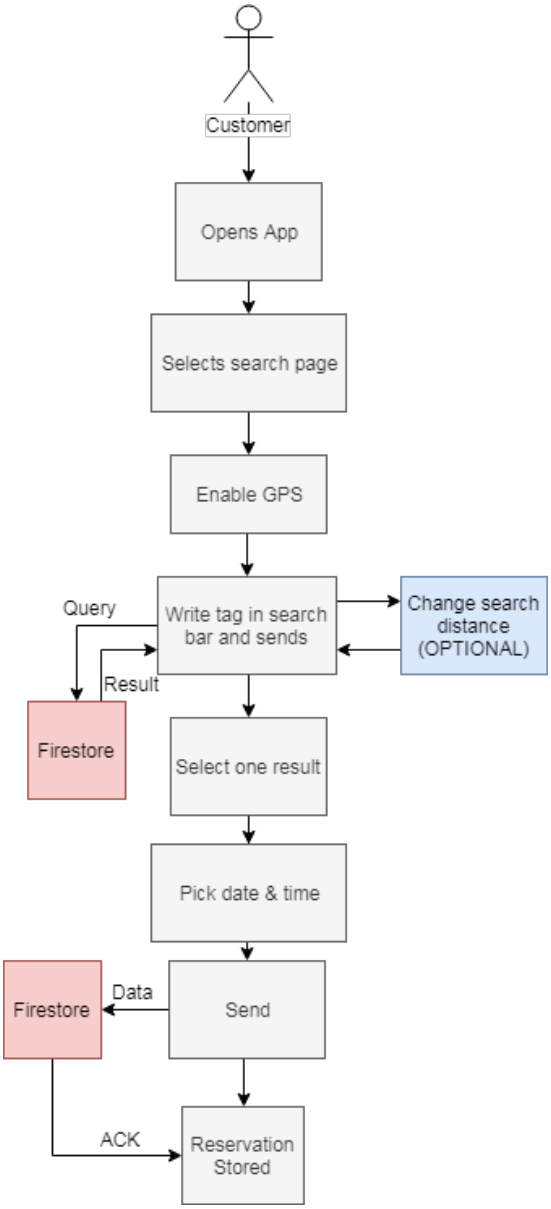


Figure 8: *Customer* reservation process

3 Algorithm Design

3.1 Introduction

In this section, it will be described the main algorithms of the system, which is the search of a shop in the area.

3.1.1 Distance Calculation

Since the GPS system is heavy on battery consumption, it's not always reliable (especially when just turned on) and the user may want to search in a different location than where he/she is right now, when the customer wants to start a search a dialog shown in Figure 16 asking for the address is shown, then the only needed values of **latitude** and the **longitude** are extracted.
Given the maximum **distance** the user wants the search to be done, the deltas for both the latitude and the longitude can be computed as follows:

$$deltaLat = (distance/earthRadius) * (180 * Math.PI)$$
$$deltaLng = Math.toDegrees((dist/(earthRadius * Math.cos(Math.toRadians(currentLat)))))$$

3.1.2 Queries

Due to a limitation of Firestore, queries are made by searching for shops with the given tag, the latitude inside the min and max values (found respectively by subtracting and adding to the current latitude the value of deltaLat) and then a third parameter called *intLongitude* is used, corresponding to the longitude truncated.
This has to be done since Firestore doesn't allow for queries with different search parameters except for equals, so instead of doing two queries (one for latitude and one for longitude) and then merging the common values, one query is done limiting the search to only one longitude and then the values retrieved are checked locally if they are inside the range. In the edge cases where the search spans over more than one longitude integer value, then more than one query will be made with the different values set, each query will return a different set than the others obviously so it must only be checked if the shops are in the correct range and no control between queries result has to be done.
Here is the example of the first (and usually only) query:

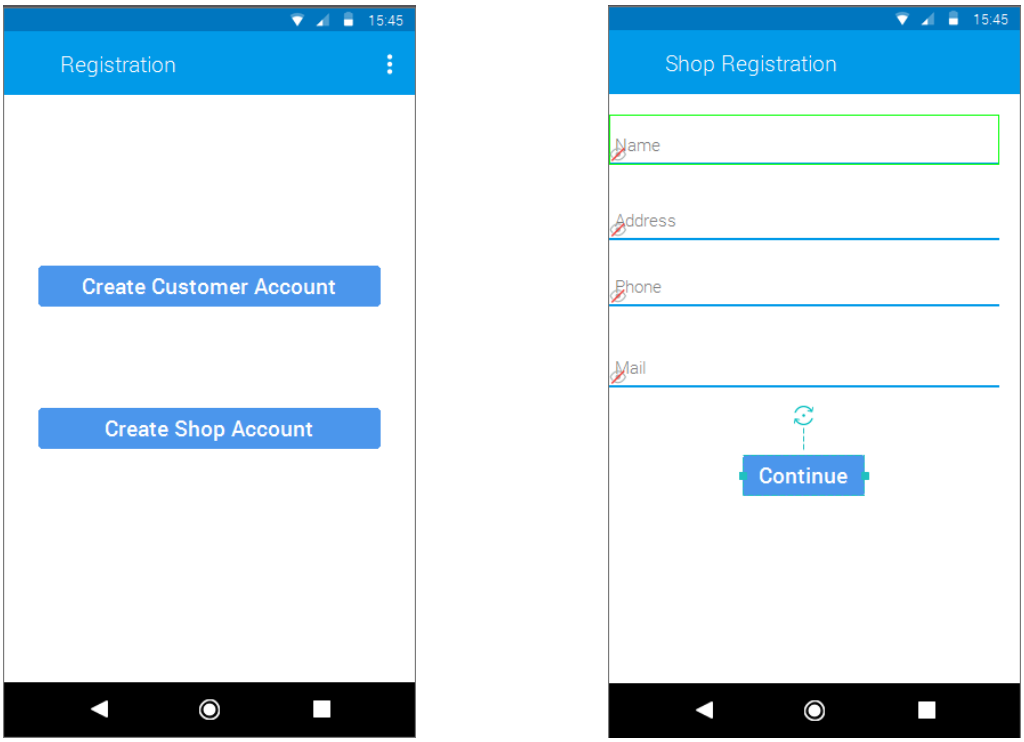
```
shopsCollection.whereArrayContains("tags", searchedText)
    .whereGreaterThanOrEqualTo("latitude", minLat)
    .whereLessThanOrEqualTo("latitude", maxLat)
    .whereEqualTo("intLongitude", minIntLng).get();
```


4 User Interface Design

These are some representations of how the mobile application should look like.

4.1 Mockups

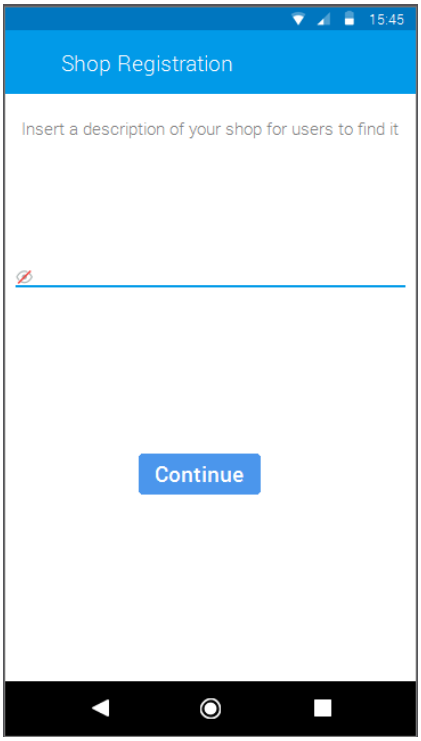
In this section some concept Mockups that were used to then develop the first iteration of the application are shown, it should be noted that the finished product must follow the concept of these designs and not use the actual graphical elements here showcased, but a professional artist should be instead hired to design the interface given these guidelines.



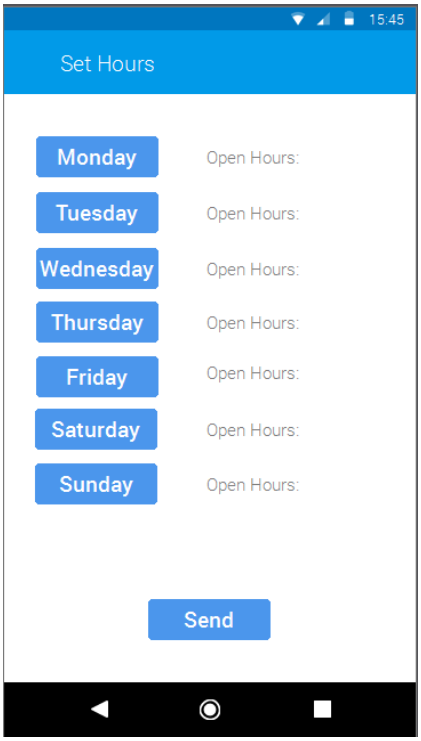
(a) User type choice

(b) Shop registration

Figure 9: Login on the left (a) and first step of Shop registration on the right (b)

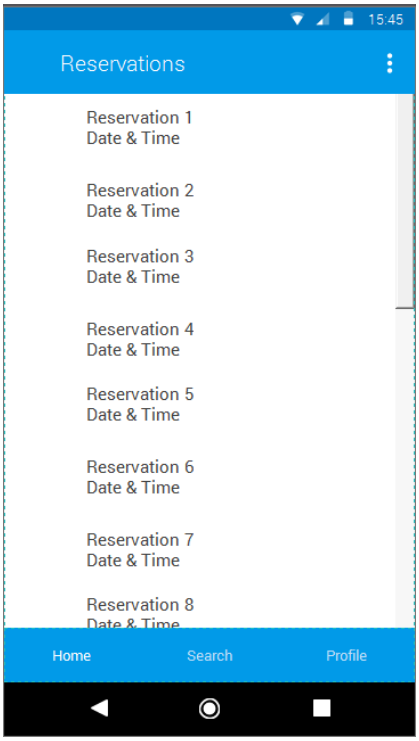


(a) Shop description request

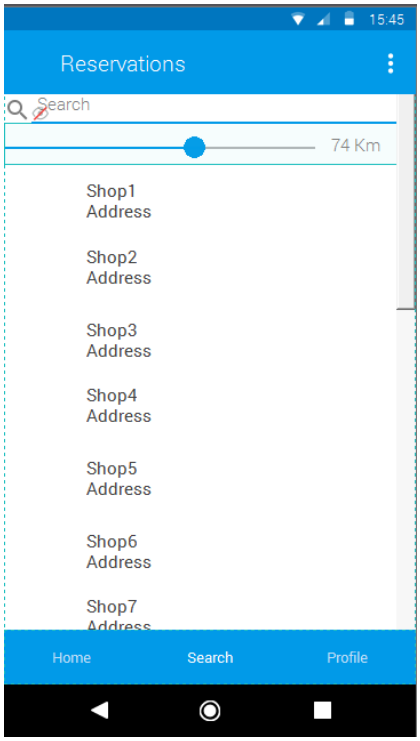


(b) Shop hours request

Figure 10: Second page for shop registration (a) and third one(b)



(a) Customer Home



(b) Customer Search

Figure 11: The customer's home page (a) and the search screen (b)

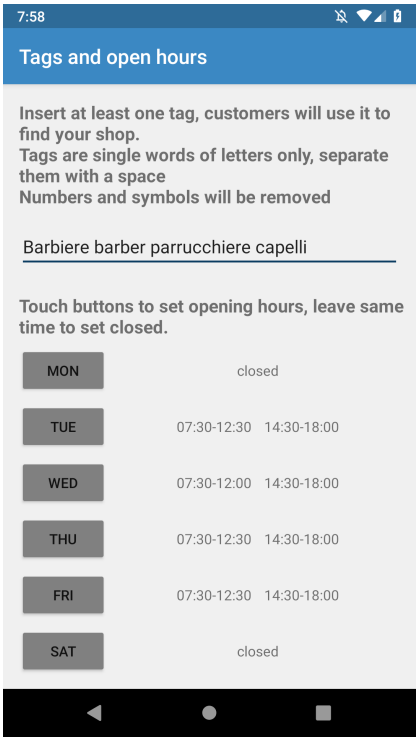
4.2 Application Screenshots

Here are instead shown the screenshots of the implementation of the application from the emulator. Some pages are not shown since they would look the same as others (for example editing a shop's information opens the same screens of the registration with the fields already inserted).

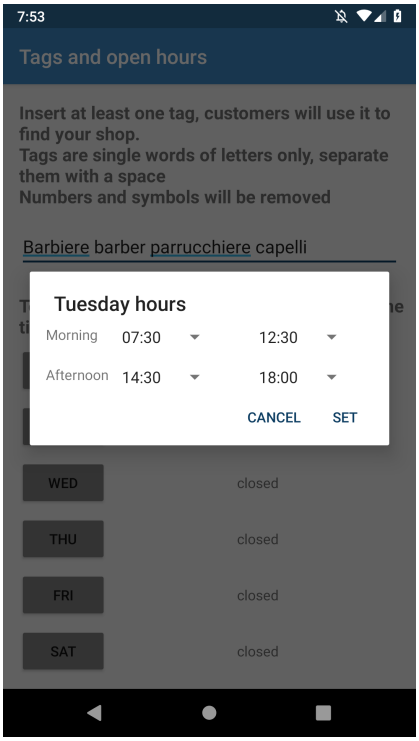
(a) Registration Home

(b) First shop registration page

Figure 12: Registration home on the left(a) first of the two registration pages for the shop on the right(b)

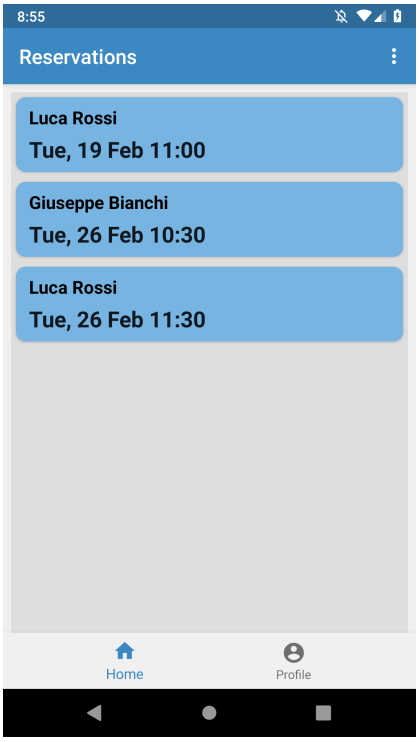


(a) Insertion of tags and selection of opening hours

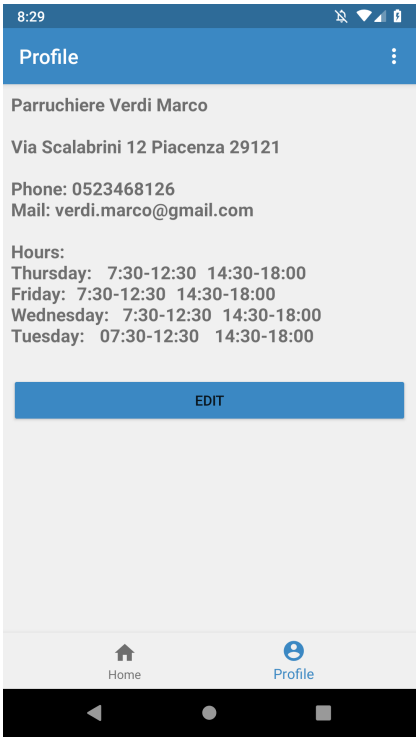


(b) Opening hours detail

Figure 13: Page for inserting tags and hours (a) with detail of the alert shown to pick times(b). Last buttons are below the view.

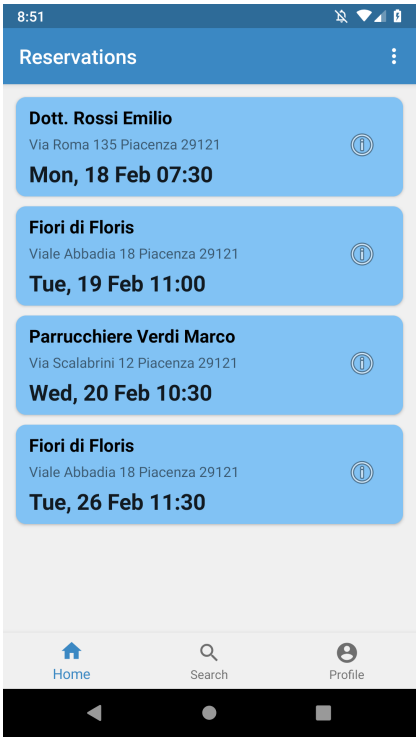


(a) Shop's home page

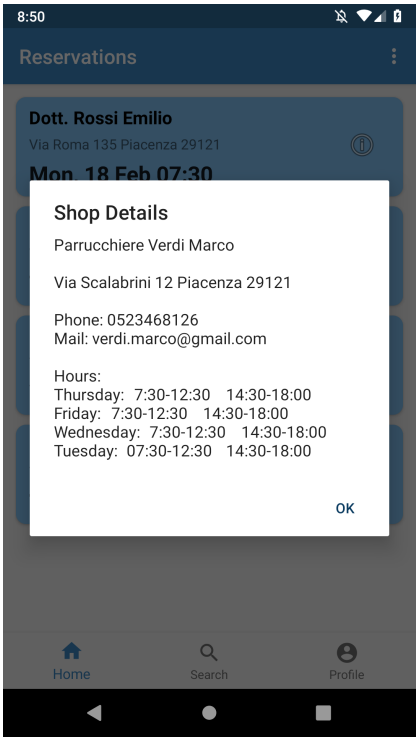


(b) Shop's profile

Figure 14: Shop home page with next reservations (a) and shop profile recap with button to edit info(b).

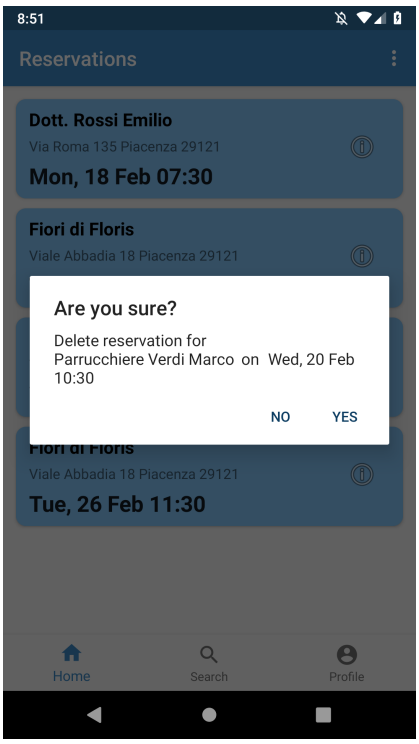


(a) Customer's home page

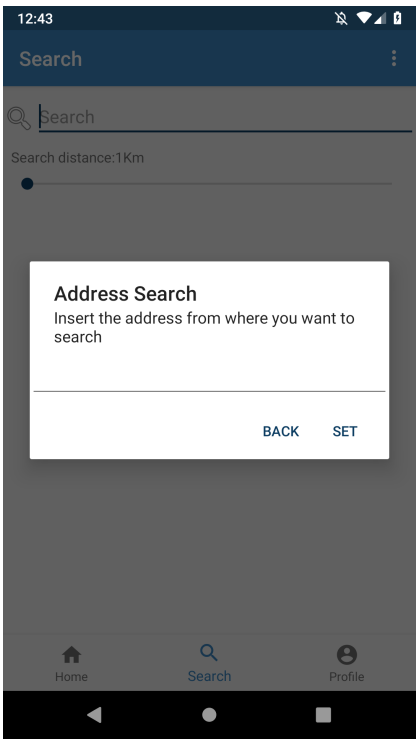


(b) Details

Figure 15: Customer home page with next reservations (a) details shown when the (i) icon is pressed(b).

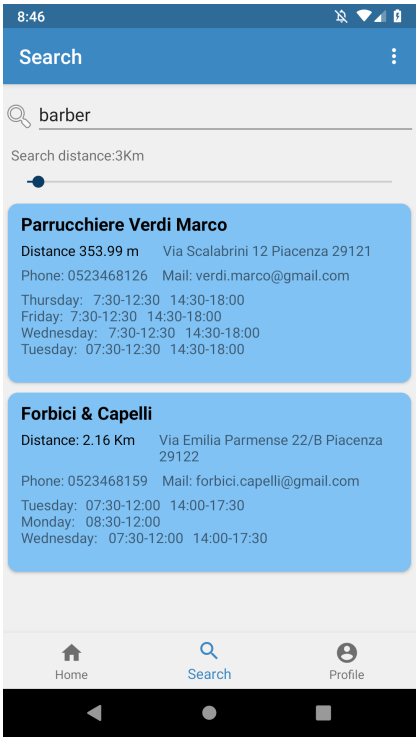


(a) Delete reservation

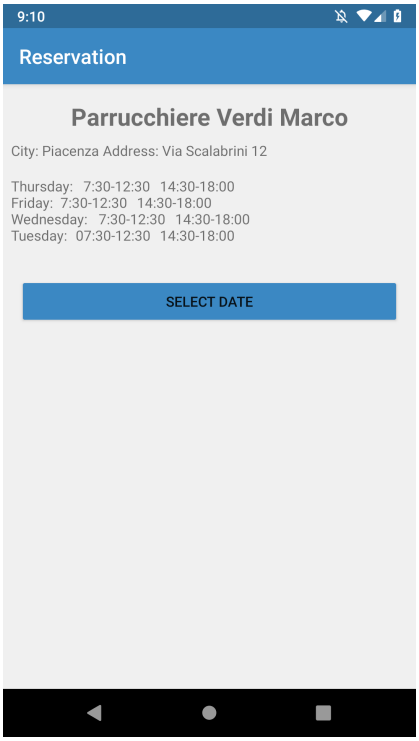


(b) Search address

Figure 16: Popup after long click on a reservation card (a) and search page when customer prompted for the search start address(b).

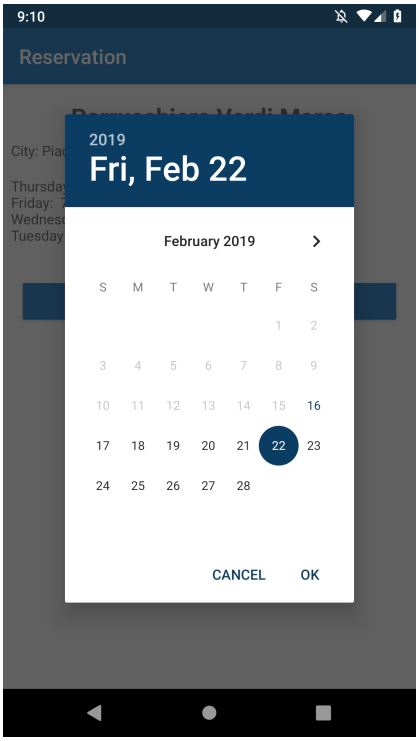


(a) Search shop

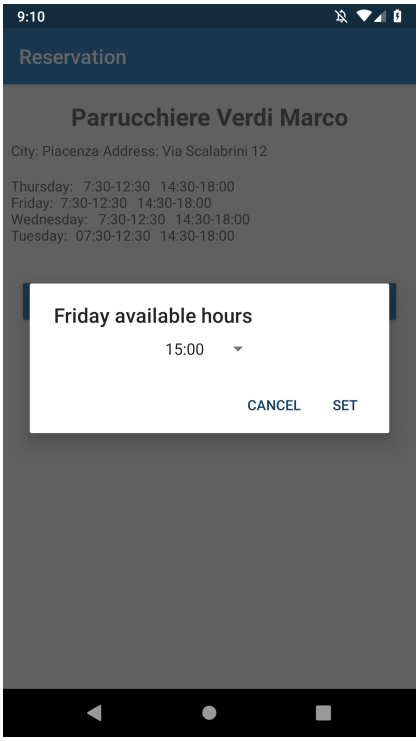


(b) Search result selection

Figure 17: Page when search is successful (a) and page after a shop is selected (b).



(a) Picking the day



(b) Hour selection popup

Figure 18: Day pick popup dialog (a) and selection of available hours (b).

5 Implementation and Integration

5.1 Introduction

In this section is present the documentation regarding the implementation order of the different components of the system and their integration with one another.

5.2 Entry Criteria

What follows is an illustration of the requisites that need to be fulfilled before the Integration phase can start.

Design Document:
This document and the initial mockups must be finished before starting with the development, it must be noted how it has to be updated after the development is finished in order to integrate it with what changed during development as the DD is not fixed and is subject to changes if needed.

Documentation:
Each method and class must be fully documented using JavaDoc in order to assure that the code is easily understandable, making it easier to extend and maintain even by different people that may end up working on the system in the future.
Names of classes, methods and variables must be chosen with the intent of communicating the reason of their existence and not be confusing or too similar to one another, also they should follow the standard Java naming conventions.
Also names should use a cascade system such as: Activity_TypeOfUser_Screen_Details.

5.3 Elements to be Integrated

1. *Database*: Both DB and DBMS are handled by the Firebase API, in particular Firestore, meaning that until a proper release is planned the free to use testing plan is more than enough.
2. *Client*: Composed by the *mobile application* it holds the logic of the system and the presentation system. The integration should focus on making sure that the communication doesn't imply high waiting time and that the graphical interface behaves as it should regarding the received data.

6 Appendix

6.1 Software & Hardware Used

1. Texmaker as an editor for LaTeX.
2. Draw.io for diagrams.
3. Justinmind Prototyper for Mockups.
4. Android Studio 3.3.1 for developing.
5. Wondershare Filmora9 for elevator pitch video.
6. Microsoft PowerPoint for presentation.
7. Nexus 5X with Android Oreo 8.1.
8. Git & GitKraken.

