POLITECNICO
MILANO 1863

Software Engineering 2 Project
a.y. 2020-21

Customers Line-up
Design Document

Version 0.0

Banfi Stefano Alessandro
Bresciani Matteo

Referent professor: Di Nitto Elisabetta

December 12, 2020

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, Acronyms, Abbreviations . . . 4
  - 1.3.1 Definitions . . . 4
  - 1.3.2 Acronyms . . . 4
  - 1.3.3 Abbreviations . . . 5
- 1.4 Revision history . . . 5
- 1.5 Reference Documents . . . 5
- 1.6 Document Structure . . . 5

2 Architectural Design **6**
- 2.1 Overview: High-level components and their interaction . . . 6
- 2.2 Component view . . . 7
  - 2.2.1 Application layer . . . 8
  - 2.2.2 Client's layers: CLup and CLup Operator . . . 9
  - 2.2.3 Data Layer . . . 9
- 2.3 Deployment view . . . 11
- 2.4 Runtime view . . . 12
  - 2.4.1 Login and registration . . . 13
  - 2.4.2 Booking, delay and cancellation . . . 14
  - 2.4.3 Enter and Exit from the market . . . 16
  - 2.4.4 Receptionist . . . 17
  - 2.4.5 Notifications . . . 18
- 2.5 Component interfaces . . . 19
- 2.6 Selected architectural styles and patterns . . . 21
- 2.7 Other design decisions . . . 21

3 User Interface Design **23**
- 3.1 Mobile Interface: CLup . . . 24
- 3.2 Desktop Interface: CLup Operator . . . 30

4 Requirements Tracebility **32**

5 Implementation, Integration and Test Plan **35**
5.1 Implementation 35
5.2 Integration 36
5.3 Testing 36

6 Effort Spent **38**

7 References **39**
7.1 Software used 39
7.2 Bibliography 39

Chapter 1

Introduction

1.1 Purpose

The Design Document aims to give usefull information to help in software development by providing the details for how the software should be built. In particolar it should be detailed enough so that developers could code the project without having to make any significant decisions.
This is done thanks to detailed description with graphical documentation of the software design for the project including different diagram types and other supporting requirement informations.

1.2 Scope

The main scope of the system is to provide users the possibility to make a booking in order to give access to the market. This could be done with two options: the first allows users to be inserted in the virtual queue; instead, the second give the possibility to schedule the booking in a precise moment in an particular day.
So, the system have to reply users' requests in real time without waiting more than few seconds due to its reliability.

To achieve this, the system is organized with a **3 tiers architecture** which divides the systems in independent modules: presentation, application and a data tier. The detailed architecture will be described as well in the next chapter.

1.3 Definitions, Acronyms, Abbreviations

See also definitions already menitoned in the RASD.

1.3.1 Definitions

- **CLup** mobile applicaiton used by users;
- **CLup Operator**: desktop applicaiton used by Receptionist;
- **Application Server**: part of the application layer. It refers to the server needed to run the
- **Booking**: it's the generic appointment. It could be either a Visit or a Reservation;
- **Avarage shopping time**: it corresponds to mean considering all appointments of all the users of the Market;
- **Bottom navigation bar**: graphical object that allows the user to display different destinations at the bottom of a screen;
- **Horizontal fragmentation**: it consist in divide a table into a set of smaller table;
- **Cross-platform development**: software development that consists to build an application using a universal language;
- **Servelet**: program that runs within a Web server;
- **SMS Gateway**: service that allows a computer to send or receive text messages;

1.3.2 Acronyms

- **DB**: Database;
- **ACID**: Atomicity, Consistency, Integrity and Durability;
- **DBMS**: Database Management System;
- **RDBMS**: Relational Database Management System;
- **SQL**: Structured Query Language;
- **HTTPS**: Hypertext Transfer Protocol Secure;
- **MVC**: Model View Controller;
- **SMS**: Short Message Service;
- **CI**: Continous Integration;
- **CD**: Continous Delivery;

- **QA**: Quality Assurance;

1.3.3 Abbreviations

- **Rn**: n-th requirement;

1.4 Revision history

1.5 Reference Documents

This document is strictly based on:

- The specification of the **RASD and DD assignment** of the Software Engineering II corse, held by professor Matteo Rossi and Elisabetta Di Nitto at the Politecnico di Milano, A.Y 2020/2021;
- **Slides** of Software Engineering 2 course on BEEP;

1.6 Document Structure

1 **Introduction**: it gives an overview of the document, by providing a brief introduction of the problem and information about the terminology used;

2 **Architectural Desing**: this section aims to describe mainly the system's structure. In particular it's focused on its components the interaction between them and all necessary details for a possible implementation;

4 **User Interface Design**: it provides the user interfaces of our system in order to understand the flow of the GUI part of our system, in relation of the actions done. In particular is focused on the UI of CLup and CLup Operator;

4 **Requirements Traceability**: in this section we link each requirement already mentioned in the RASD to the main component of our architecture;

5 **Implementation, Integration and Test Plan**: it's focused on the development process, focusing on how it will be integrated and tested;

6 **Effort Spent**: it shows the time spent to realize this document, divided for each section;

7 **References**: it contains the references to any documents and to the Softwares used in this document.

Chapter 2

Architectural Design

2.1 Overview: High-level components and their interaction

The system is organized following the three tiers architecture. This aims to decouple logical layers in order to gurantee an horizontal scalability and an high fault tolerance. Graphically it's shown in the figure 2.1.

Presentation layer. It's the front-end layer which consists of the user interface. We have two types of user interface, depending on his functionality:

- **CLup**: It's the mobile application used by users who have a smartphone. They can manage their booking by themselves;
- **CLup Operator**: It's the desktop application used by receptionists that act as an intermediary to manage booking of users that have only a mobilephone.

Application layer. It deals with the model of the system, by containing the business logic of the application. In our system it consists in a remote server to which mobile and desktop applications have to connect due to manage any booking.

Data layer. It's composed by a data storage system. It includes:

- User sensitive data asked during the registration process;
- Information about user's grocery shopping;

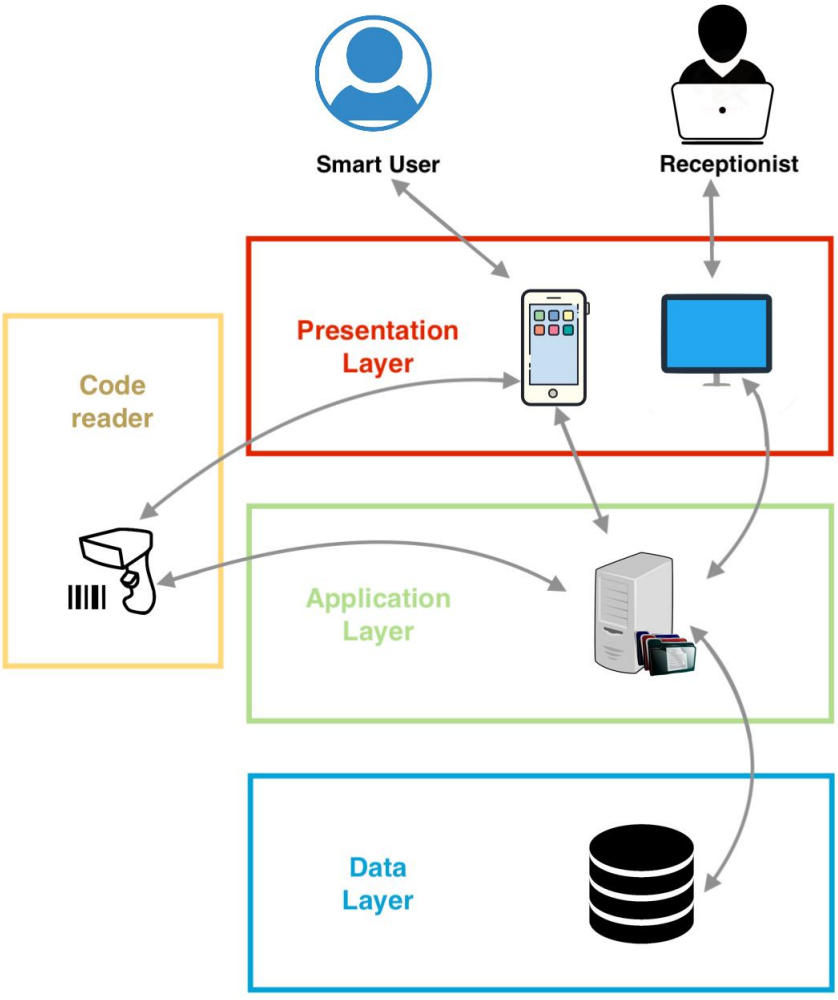


Figure 2.1: Three tiers architecture of the system

2.2 Component view

In this section we describe the architecure with respect to its components and how they are wired together to form our system. Indeed, communication between Server and Clients is ensured by appropriate components which are called **Network Manager**.

By an high-level point of view (figure 2.2) the main subsysystems correspond almost to the 3 layers in the 3 tiers architecture, with the addition of the **Market's subsystem**.

The last one includes the **QRCode reader**, which has different functionality:

- It checks if QRCode at the entrance/exit is valid. If it's valid it will submit it. In addition a manual insertion of the QRCode is provided, due to allows Mobile Users to insert his code sent by SMS;
- It controls the **Shop door**. When a QRCode is submitted, the reader allows users to enter by opening it;

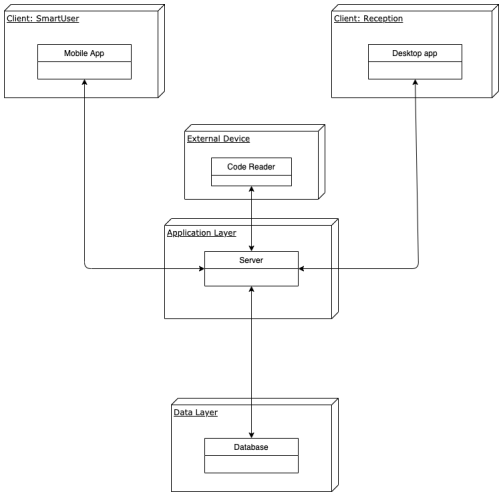


Figure 2.2: High-Level component diagram

It follows a detailed description of the other significant components (figure 2.3)

2.2.1 Application layer

It's the core of our system. It's composed by:

- **Controller**: In our model this is th core ofour business logic. It deals with the stochastic time estimations to control the flux of users inside and outside the market. Moreover it uses interfaces provided from the other components linked to it;
- **Auth User and Auth Reception**: these components grants respectively the access of users and receptionist by verifying their credential. In particular they are also accountable for users' registration;
- **Visit Schedule and Reservation Queue**: they manage respectevely Visit and Reservation requests of Users and Receptionists. In addition they provides QRCodes;

- **Notification Manager**: it's the component necessary to notify users about their turn in the queue. Furthermore, it repsonsible for sending SMS to user with mobilephone through an *SMS Gateway*;

2.2.2 Client's layers: CLup and CLup Operator

Each one of them is composed by components in a similar way, even if they have different functionalities. Both they've got, as we can see in the figure 2.3:

- **GUI component**: it provides the graphical interface in which User and Receptionist can interact with the system;
- **Network Manager**: it's used to interact with the application server by sending or receiving informations;

CLup is composed also by a **Request Handler** which exposes methods to User to handle any request or action needed. Then, it interacts with the Network Manager to send request to the Server.

Instead, CLup Operator has the **Booking Manager**, which is a similar component to the previous one but with more functionalities. In particular it handles any booking of mobile users and their sensitive information. So it's allowed by application server to query all users' sensitive data with higher privileges than CLup.

2.2.3 Data Layer

In our architecture the data layer is composed by a relational DB needed to store informations about users and their dynamics in the market.

In particular it should be connected to the server placed in the application layer. To reach the goal we plan to adopt a **RDBMS** in order to deal with a relational database. Indeed, it ensures consistency of data due to the *ACID* properties.

In addition, it can process a large amount of data, which is suitable for our application.

Besides, due to a better efficiency, RDBMS provides *horizontal fragmentation*. It allows to fragment entities of our model with respect to each different market. This is why tuples regarding different markets will never be queryied together. The only which won't be fragmented is the Market entity.

Moreover, it includes a software program which is designed to capture request over a network of the application server. From this each user and receptionist in fact could retrieve informations needed through SQL Query by connecting to it.

An other important aspect is the security of the data due to mitigate any risks of violation. In order to do that we limit its access exclusively only to the

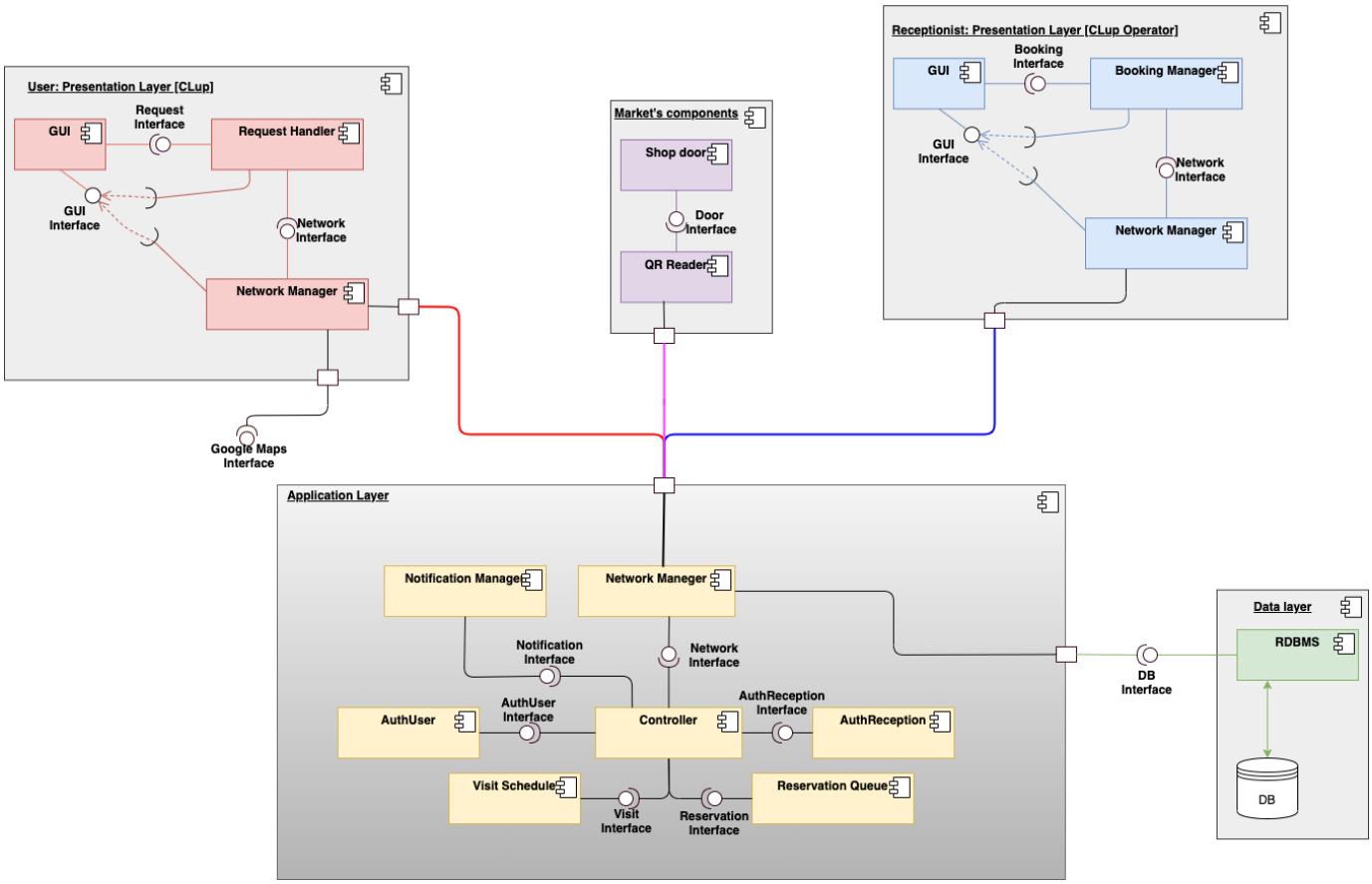


Figure 2.3: Component Diagram

application server. Communication between them will be also encrypted and accounts' passwords will be hashed.

2.3 Deployment view

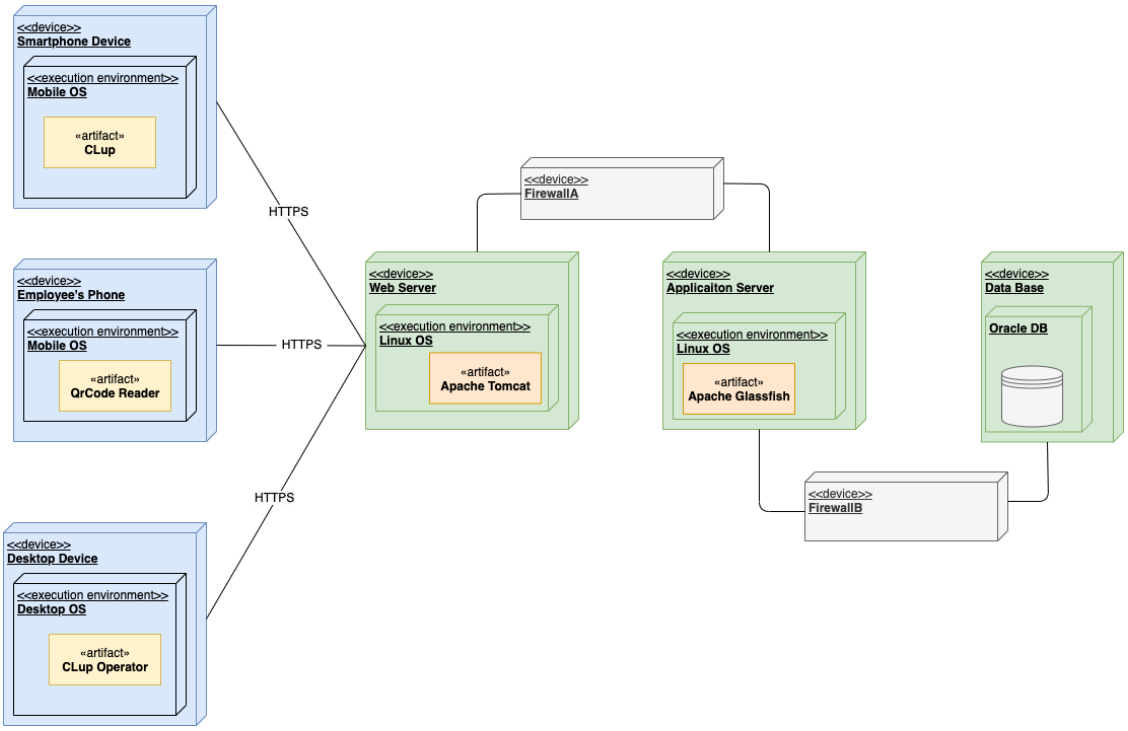


Figure 2.4: Deployment Diagram

In this section we'll explain the structure of our run-time system and the interaction between the hardware and the software parts. This is shown in the deployment diagram in the figure 2.4.

For the client's side, CLup and CLup Operator will be hosted respectevely on a mobile and desktop device. Softwares will be designed both following a cross-platform development. We choose cross-platform to ensure an easier and quicker implementation; moreover it includes a better uniformity and maintenability. The main OSs choosen are respectevely iOS and Android for CLup and macOS and Windows for CLup Operator.
In addition is provided a third device used by a generic employee of the market which, interfaced with a QRCode scanner, sends to the server any request for entering or leaving the market.

Instead, the server is organized as well following a **3 Tier Server Architecture**. Each tier is composed by:

1. **Web Server**: it's the node to which every clients connect. It handles each request from the clients connecting by HTTPS and is built using **Apache Tomcat**.
   This, supported by Apache Software Foundation, a nonprofit corporation, is an open source software which is lightweight and suitable for our web server;
2. **Application Server**: this node hosts the core of our business logic. We set **Apache Glassfish** which is leading of the Java EE standard to run servlets. Compared to Tomcat, is a full-blown Java EE application server, because it includes more features. Indeed it provides backup and recovery services due to a better availability a reliability;
3. **Data Base**: this machine contains each data of our system. In fact it's composed by a **RDBMS** in order to store and retrieve data. Besides, we choose **Oracle DB** from Oracle Corporation as management system on it. We choose it because of its high performances, which are necessary to process data quickly.
   In particular it can handles large volume of data since it have to manage a many users' requests simultaneously;

Each tier is divided from the others with **2 Firewalls** which aim to filter any incoming and outgoing network traffic. In particular:

- **FirewallA**: it filters traffic coming from the WS to AS. It allows requests from receptionist, logged user and employees' devices. This is due to mantaining the application server's integrity;
- **FirewallB**: this protects the DB by filtering any packets, except for the application server. In this way we gurantee an high security against any data violation;

2.4 Runtime view

In this section it will be illustrated the main runtime procedures of our system. We describe these by using Sequence Diagram in order to formalize how our components work together and in which order are used.

2.4.1 Login and registration

Both processes start from the user's device where a request is elaborated by the *Request Handler*. After the request passes to the Server's side where either login or registration is processed by *Auth User*. Then, it established, by accessing to the *RDBMS*, if the request can be validated or not. A return message was sent to the users for a success or failure operation.
It's not mentioned the receptionist' login because is structurally the same.

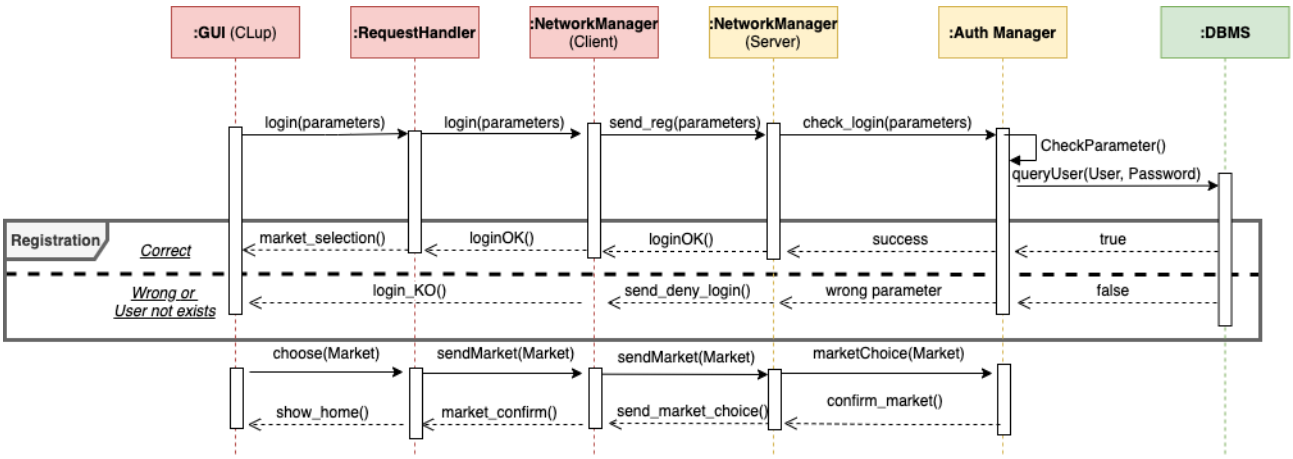


Figure 2.5: The sequence diagram of the CLup's app login.

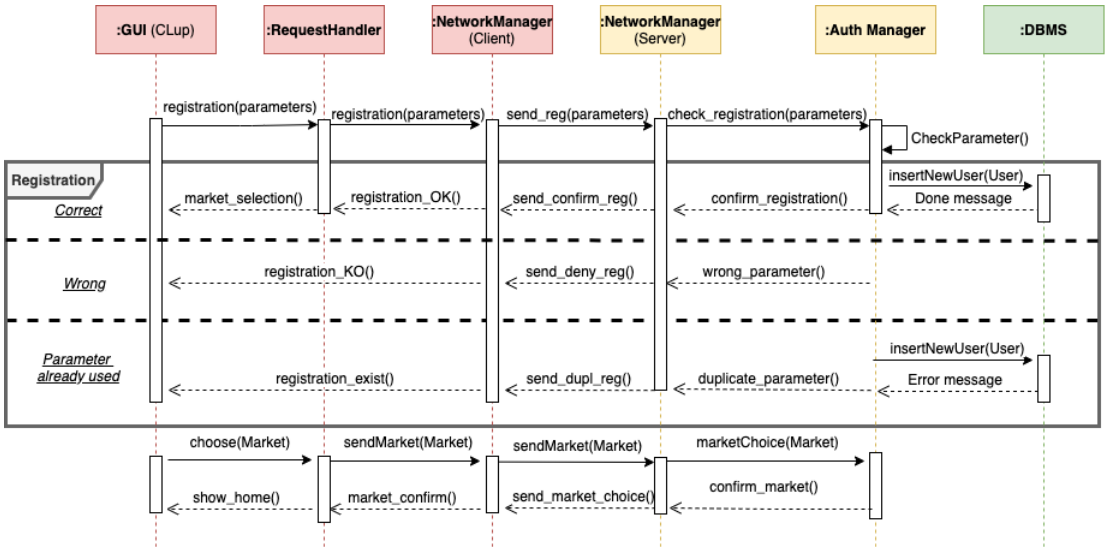


Figure 2.6: The sequence diagram of the CLup's app registration.

2.4.2 Booking, delay and cancellation

Any appointment is managed by *Request Handler* which interacts with the *GUI* component to compile an application. Then it's sent to the Server thanks to the *Network Manager* components and processed by the *Reservation Queue* or by the *Visit Schedule*. Any relevant data in our system will be updated, delated, or inserted in our Data base through the RDBMS.

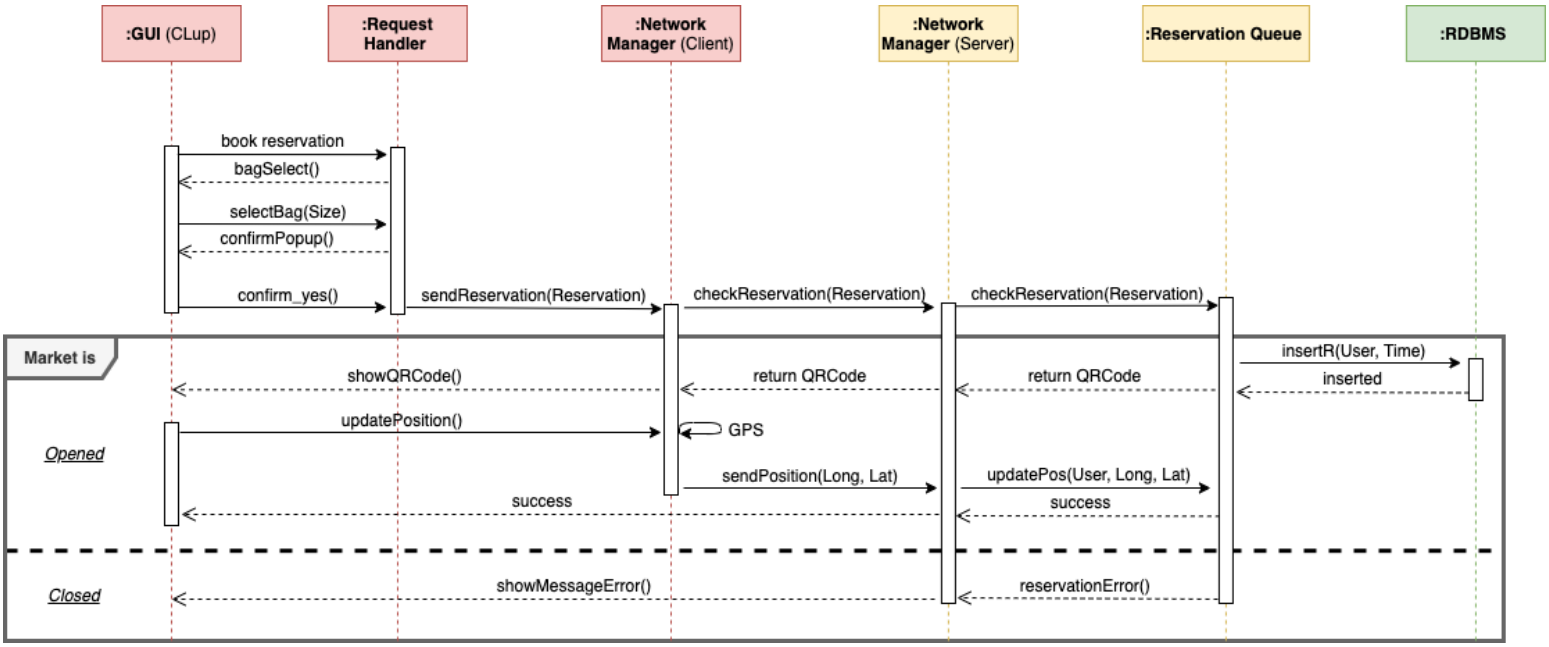


Figure 2.7: Sequence Diagram of the procedure to be inserted in queue for a Reservation.

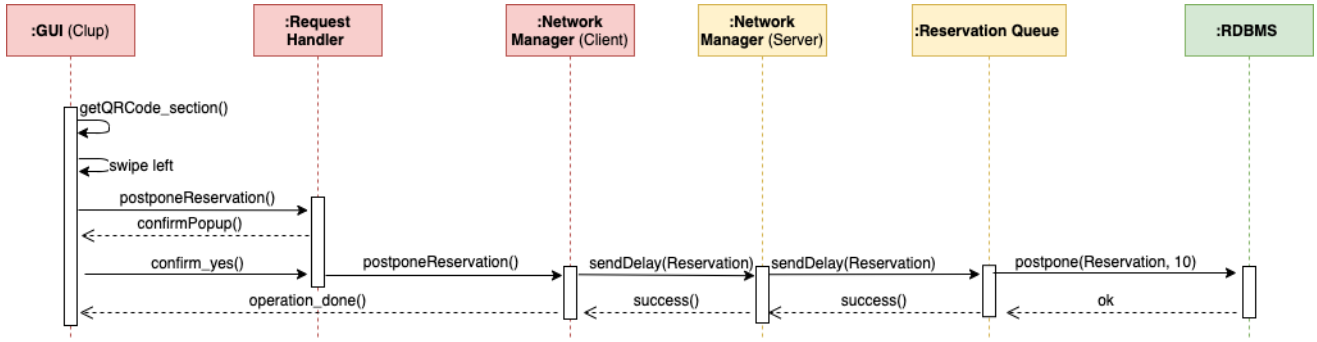


Figure 2.8: The diagram shows how a generic user postopones his turn in the queue.

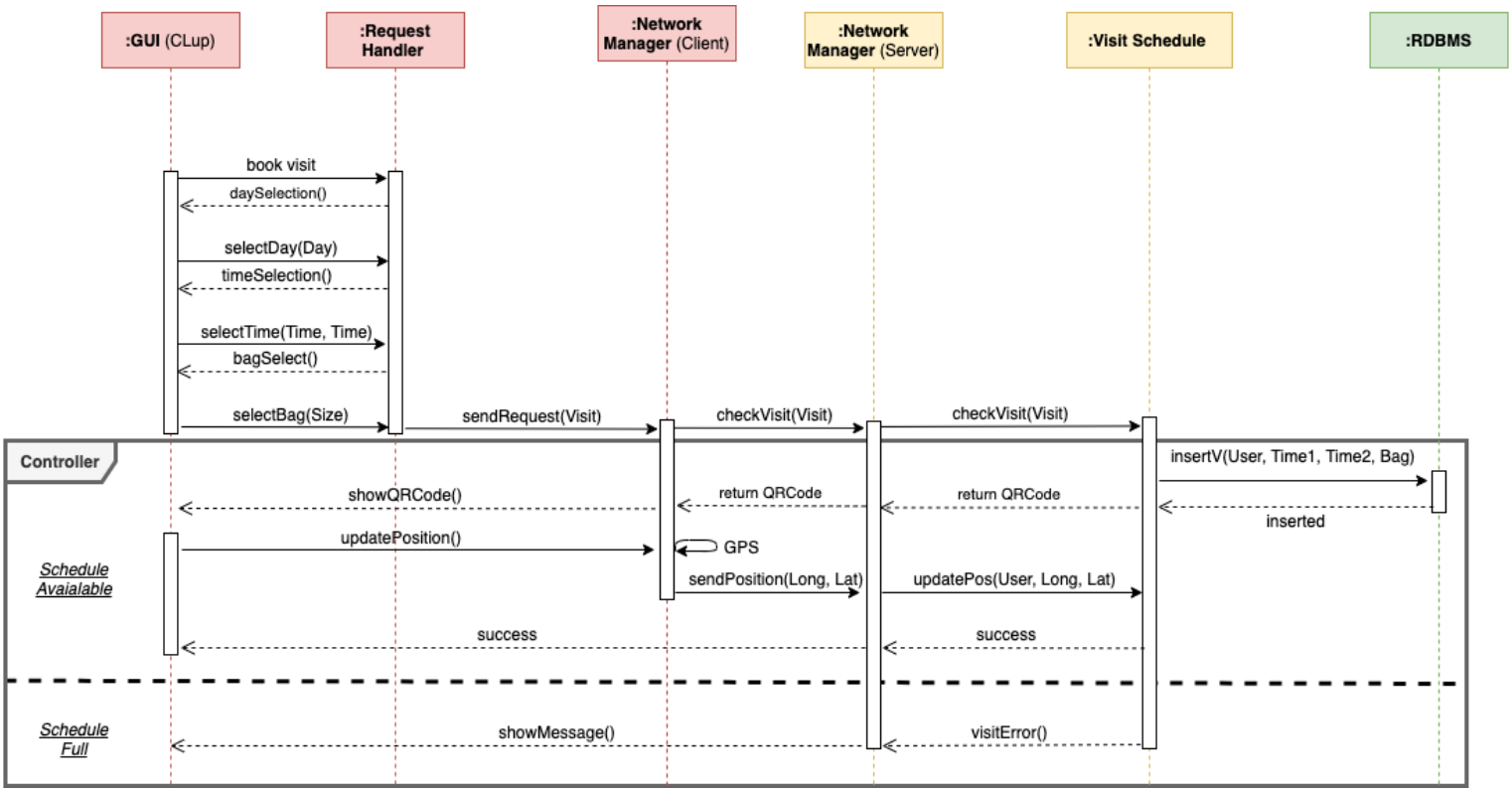


Figure 2.9: Sequence Diagram of the booking procedure of a visit made by an user.

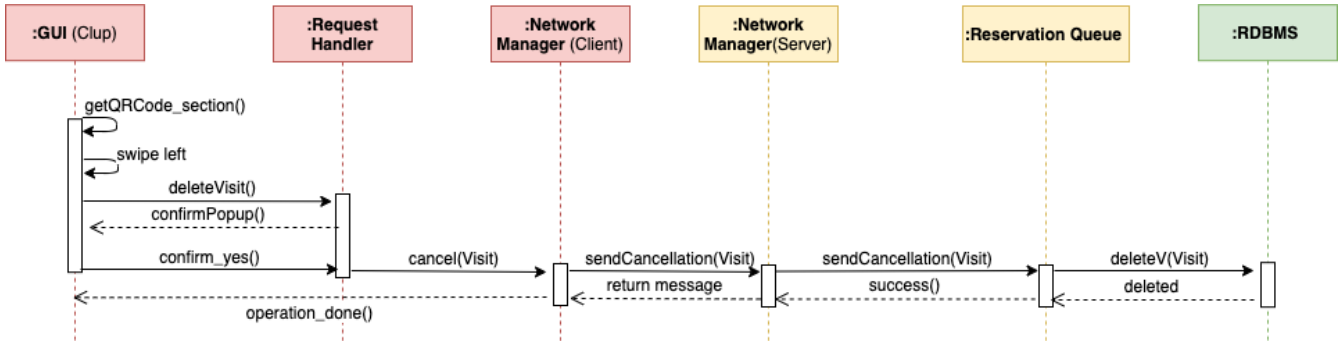


Figure 2.10: The diagram shows how a generic user deletes his turn from the schedule.

2.4.3 Enter and Exit from the market

This sequence diagram aims to formalize the interaction of the user in the system when he arrives at the entrance of the market and starts his grocery shopping. An user submit his QRCode through a *QR Reader* which is sent to the Server; then, it will be checked by the *Controller*. Both times of entry and exit will be stored thorugh the RDBMS if the QRCode is valid.

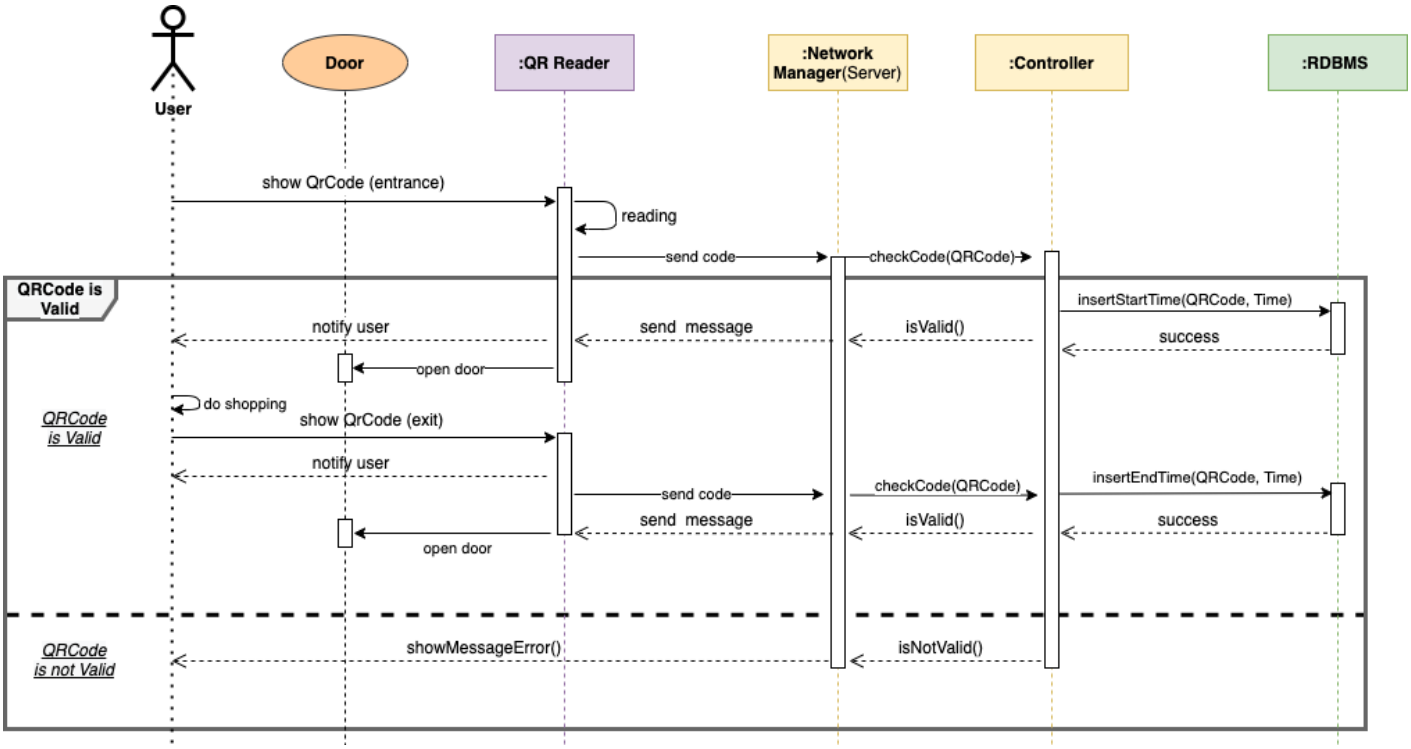


Figure 2.11: The SD explain how an user is allowed to enter and exit from the market.

2.4.4 Receptionist

In this sequence diagram, it's modelled the interaction of the components when a user without a smartphone wants to book a seat in the queue by calling the receptionist of his market. Firstly, while he's speaking on the mobile phone, the receptionist deals with his registration (in the case of a new user), or his recognition (in the case is already registered). Secondly the procedure to book a reservation will be almost the same in the case described in the SD in figure 2.7. The main difference it's the component *Booking Manager*, which differently has higher privileges to retrive and store data in the DB.

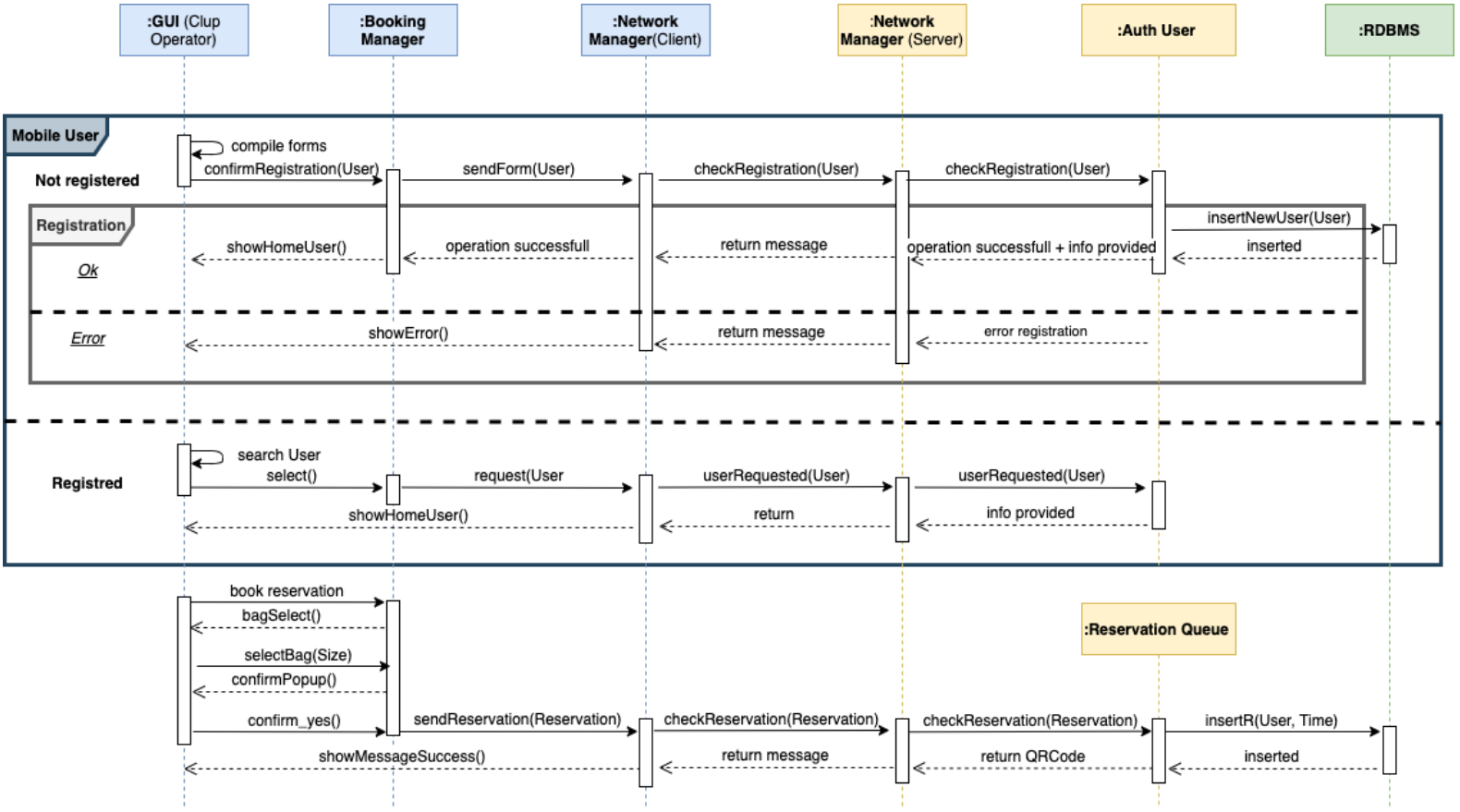


Figure 2.12: The SD exaplain how a reservation of mobile user is booked by the receptionist.

2.4.5 Notifications

This describes how the the application server is working when it has to notify an user about his upcoming turn. At the beginning, the server through the *Controller* checks the schedule by verifiying if can allows a new user in the market. When it could do it, ask for the next user in head of the queue. The first notification is sent in advance to allert the user because it will be almost his turn. While the second will be sent when the turn arrived and the user is allowed to enter.

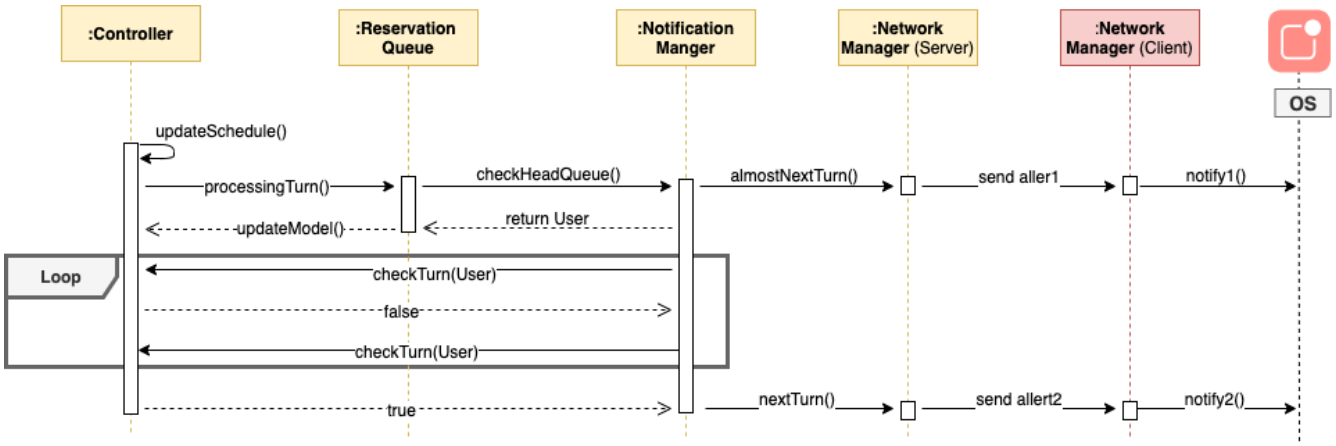


Figure 2.13: The sequence diagram exaplain how the system notify for its turn in the queue.

2.5 Component interfaces

In this section we describe the interfaces needed for the communication between each component. Moreover a list of the main functionalities provided by them is provided.

- **Request Handler Interface**: it exposes to the GUI component the following methods:
  - *void selectBag(Size)*: used to confirm the bag size during the booking request (Reservation or Visit);
  - *void confirmYes()* and *void confirmNo()*: used to confirm or not a Reservation;
  - *Boolean bookVisit()* and *Boolean bookReservation()*: used to open a booking request;
  - *Boolean delayReservation()*: used to open a Reservaiton delay;
  - *Boolean cancelReservation()* and *Boolean cancelVisit()*: used to open a Reservation delay;
- **GUI Interface**: it interacts with the Network Manager and Request Handler components in order to control the **View** part of our system.
- **Network Interface** (CLup): it interacts with the Request Handler component due to send request to the Application Server;
- **Network Interface** (CLup Operator): it interacts with the Booking Manager component in order to send request to the Application Server;
- **Scanner Interface**: it interacts with the QR Reader component due to open or close the market's door, depending on the validity of the QRCode;
- **Google Maps API Interface**: it interacts with the Network Manager (CLup) component due to locate the own position and the time necessary to reach the market;
- **Network Interface** (Server): it interacts with the Controller component in order to:
  - Send responses to the clients;
  - Update, insert, delete and query data of the DB;
- **Reservation Interface**: it exposes to the Controller component mainly the following methods:
  - *Boolean checkReservation(Reservation)*: it checks and confirms or declines a Reservation request from an User/Receptionist;
  - *Boolean updatePos(User, Long, Lat)*: it updates the current position of an User, in order to estimate his time arriving;

  - *Boolean sendCancellation(Reservation)*: it allows to cancel a Reservation;
  - *Boolean sendDelay(Reservation)*: it allows to postopone the turn of an User in queue;
- **Visit Interface**: it exposes to the Controller component mainly the following methods:
  - *Boolean checkVisit(Reservation)*: it checks and confirms or declines a Reservation request from an User/Receptionist;
  - *Boolean updatePos(User, Long, Lat)*: it updates the current position of an User, in order to estimate his time arriving;
  - *Boolean sendCancellation(Reservation)*: it allows to cancel a Reservation;
  - *Boolean sendDelay(Visit)*: it allows to postopone the turn of an User in queue;
- **Notification Interface**: it interacts with the Controller component in order to schedule on time notifications that have to be sent to the users;
- **Auth User** and **Auth Rceptionist Interfaces**: they interact with the Controller component in order to check login and registration request from Users and Receptionists;
- **DB Interface**: it interacts with the Network Manager component in order to insert, delete and update data in the DB. It exposes to the Network Manager mainly the following methods:
  - *newUser(User, Password, Informations)*: inserts data regarding a new user;
  - *insertReservation(User, Reservation)* and *insertVisit(User, Visit)*: inserts data regarding a new Reservation or Visit;
  - *deleteV(Visit)* and *deleteR(Reservation)*
  - *postpone(Reservation, 10)*: postpone a seat in the queue for a Reservation by 10 minutes;
  - *insertStartTime(QRCode, Time)* and *insertEndTime(QRCode, Time)*: insert the startTime and endTime for a grocery shopping in the market, associated to its QRCode;
  - *checkCredential(User, Password)*: queries the DB verifying if the user and his credential are correct;

2.6 Selected architectural styles and patterns

These are the following pattern proposed for a potentially implementation.

3 Tiers Architecture

As already mentioned in the section 2.1 is formed by the Presentation, Application and Data layers. By separating each layer scalability of each component will be improved. Moreover flexibility is added because of the reduction of development cycle times. Developer in fact can upgrade or replace independently one tier without affecting the others.

Model-View-Controller

MVC is an architectural pattern which can be used to organize the input, processing, and output of an application. It's composed by:

- **Model**: it's where the data objects of our business logic is stored;
- **Controller**: it's a layer that responds to the users and to changes in the model;
- **View**: it represent the user interface;

A MVC pattern allows a faster development due to the fact that each layer can be built in parallel to the others. This, also supports an asynchronous technique, which could make our system faster. This is will be very significant for notifications that have to be sent to users in precise moments. **Observer**

This is a software design pattern usefull for objects which have to notify automatically any state changes to their dependents. We need it due to propagates any changes in our model to its components in a very efficient mannor.

For instance it could be very helpfull for managing a queue variation when it happens (a user cancels his reservation).

2.7 Other design decisions

Users Flow in the market

To reach the goal of controlling people into the market and to avoid the queue formation, the Controller component needs an **algorithm** that allows called users to directly enter without waiting outside.

This algorithm will be based on:

- A maximum threshold of people that the market can contain. We call this threshold **maxThreshold**;

- A tolerance **margin**, to manage unexpected events and to guarantee that MaxThreshold will not exceeded in case of slowdowns;
- A **data gathering system** that will estimate, basing on users habits, when the system have to notice the next user in queue;

The system will be based on informations entered by the users and by statistics of the collected data. First of all, the shopping size will be requested when the user takes the reservation; in this way it will be possible to predict the maximum shopping time. This parameter will be based on a **user reliability index** that will be estimated by computing how the time of parmanence in the market deviates from his avarage time with the same shopping size. (deviazione standard)
For example, the fact that a user selects a "Small" size and goes shopping for 1 hour is less reliable than one that goes shopping for 30 minutes in the shop. So accordingly to the reliability index, the next user in queue will be noticed with a different timing.

The **average time** according to shopping size will be estimated on all user reservations and due to his reliability index.

In the case of a new registered user, with no or few reservations, the duration will be estimated exclusively on his average time due to the lack of accuracy of data collected. In case of usual customer with a low reliability index, the shopping time will be considered according to the client habits without consider clients' **average shopping time**.

The system for each user into the supermarket will monitor the duration of the shopping and when will be near to average, accordingly to the shopping size and reliability index, will call the next user. In order to decide the moment in which a user is called, we consider also the travel time which is how long he takes to reach the shop, based on his position. If the user will not share own position, the system will use a *standard travel time* (10/15 minutes). In fact, usually the customers go shopping near their home.

The system will also consider the visit that will be scheduled during the day in order to manage entrance. When the system will notice a people overload will have to slow down calls of new customers, in order to never exceeded the entrance maximum number in the market.

$$PeopleInMarket + Margin + PeopleVisit <= MaxThreshold$$

Chapter 3

User Interface Design

In this chapter we will illustrate most of action allowed in CLup and CLup Operator using UI flowchart digrams. UI flowchart diagrams are used to model the interactions that users have with the software, by understanding how the system is expected to work.
Diagrams are built using the mockups already put in the RASD, but in a more detailed way.

3.1 Mobile Interface: CLup

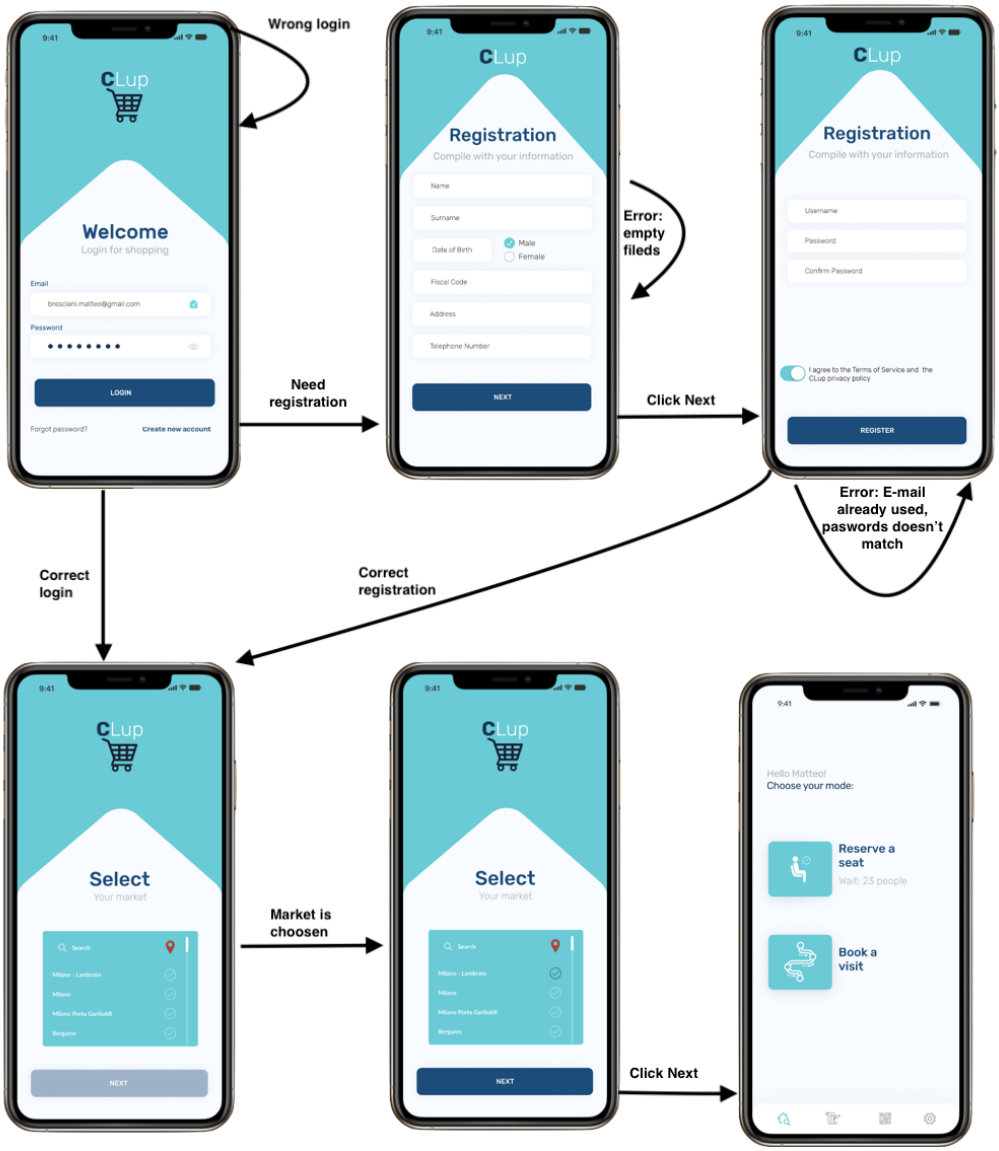


Figure 3.1: Login and registration interface: starting from the first screenshot the user must authenticate or register himself to use CLup. Hypothetically if a User signs up for the first access, he will be already logged in. After procedure the home screen will be displayed.

Figure 3.2: From the home screen it's possibile to move in the other three app section by selecting them in the Bottom Navigation Bar

Figure 3.3: Procedure needed to book a Visit.

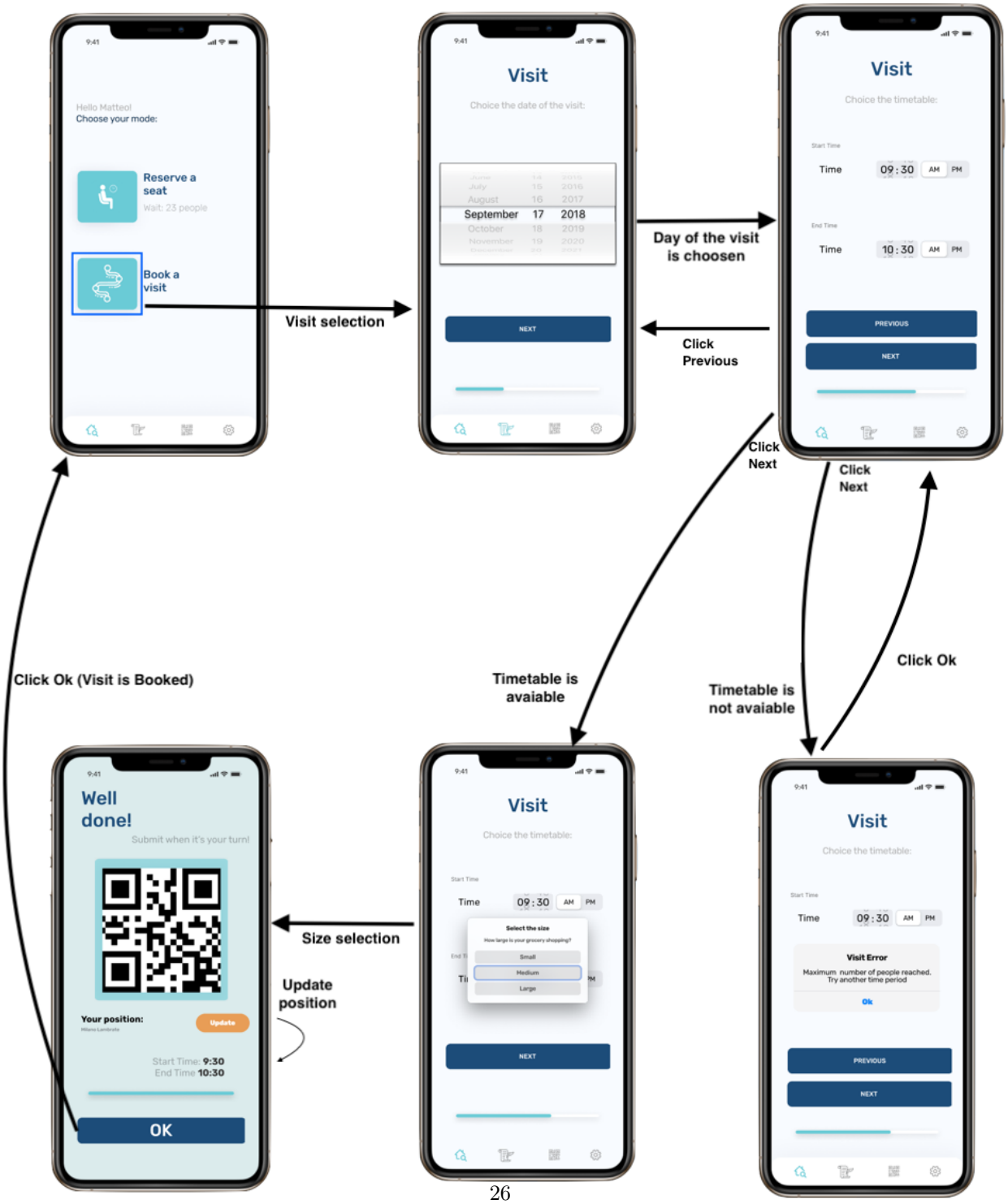


Figure 3.4: Procedure needed to make a Reservation.

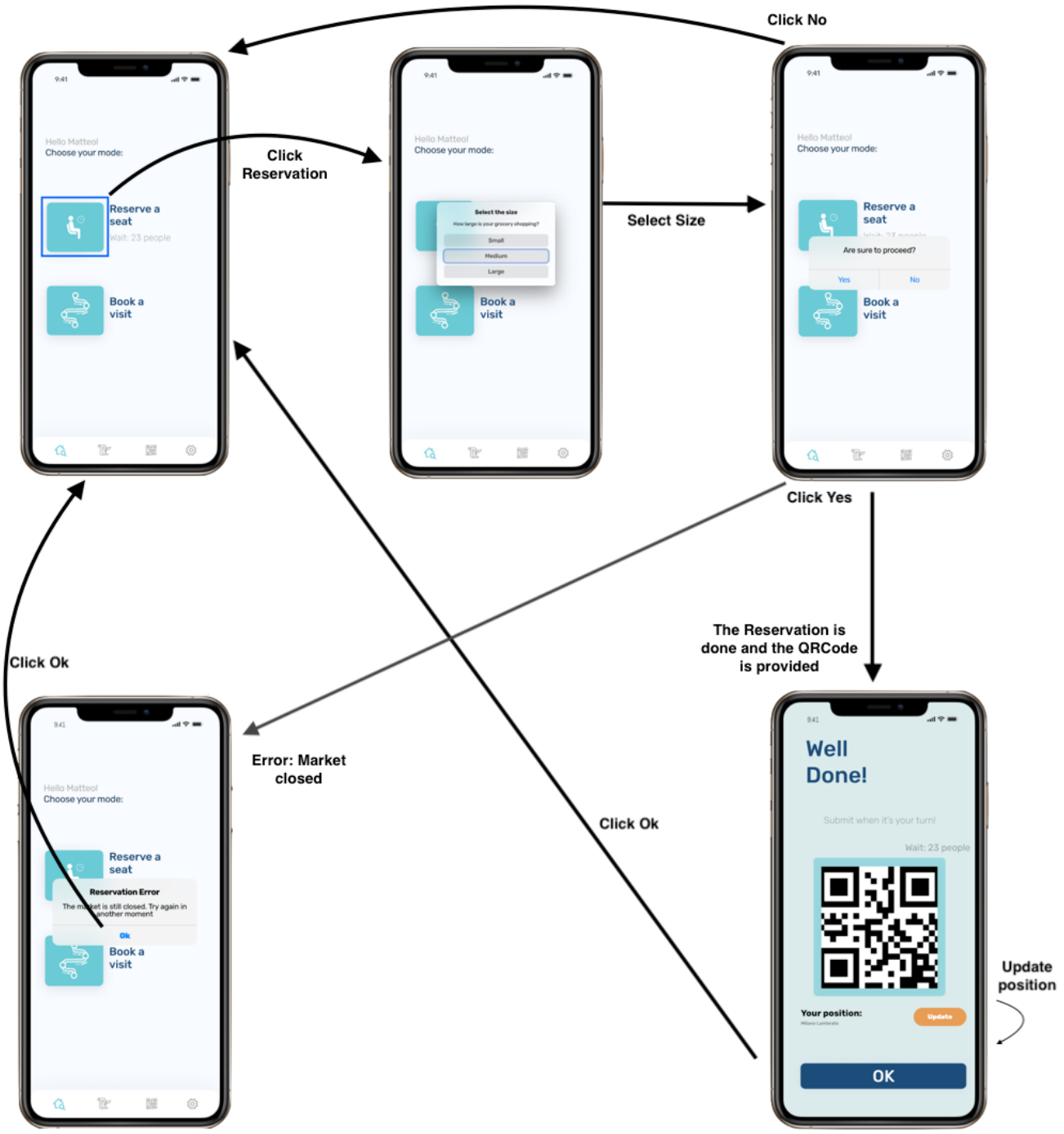
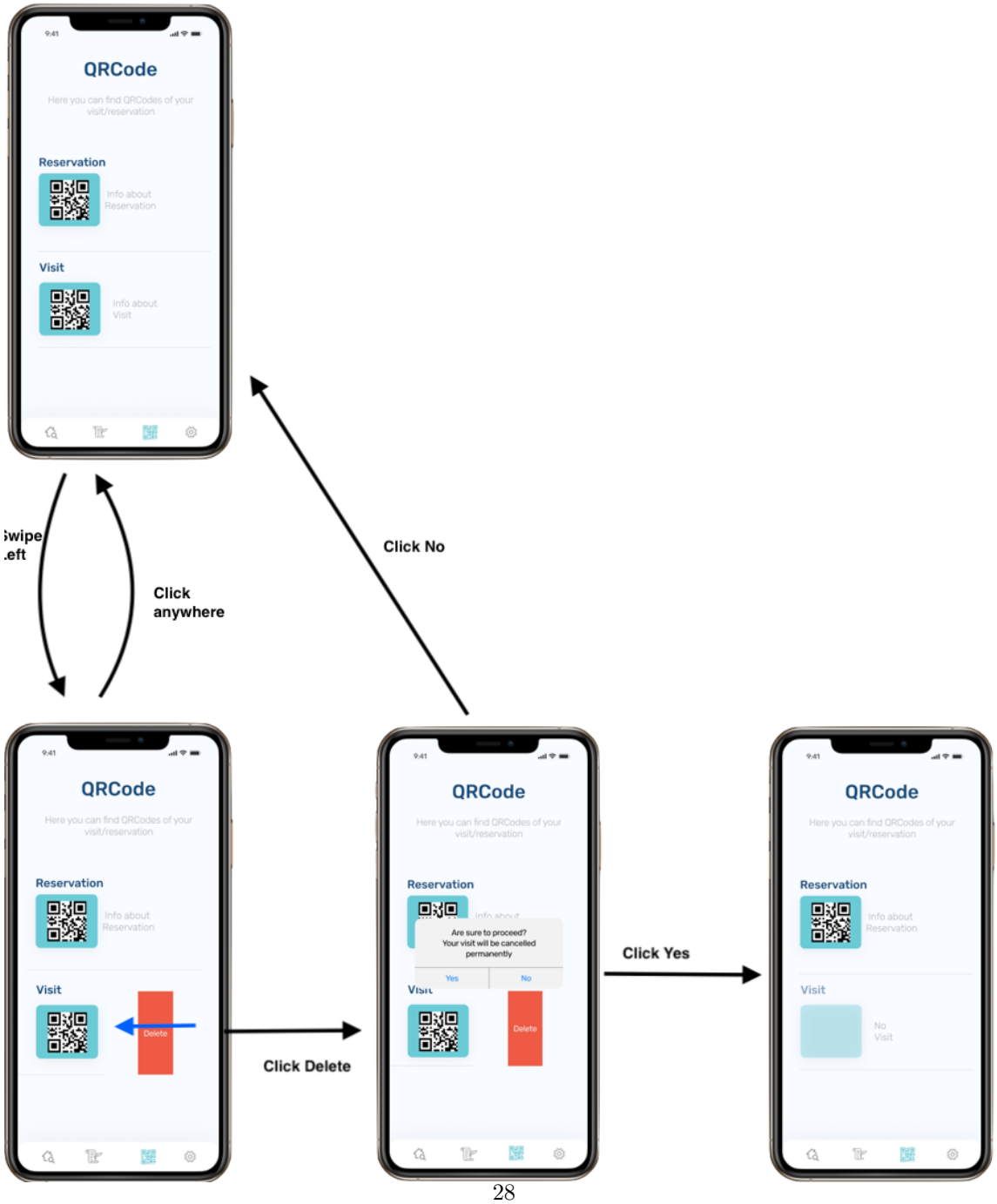


Figure 3.5: The diagrams shows how it's possibile, from the QRCode section, to cancel a Visit. The procedure will be the same also for a Reservaiton cancellation. In particular the following screenshoots illustrate the procedure in a scenario in which a user has booked both Reservation and Visit.



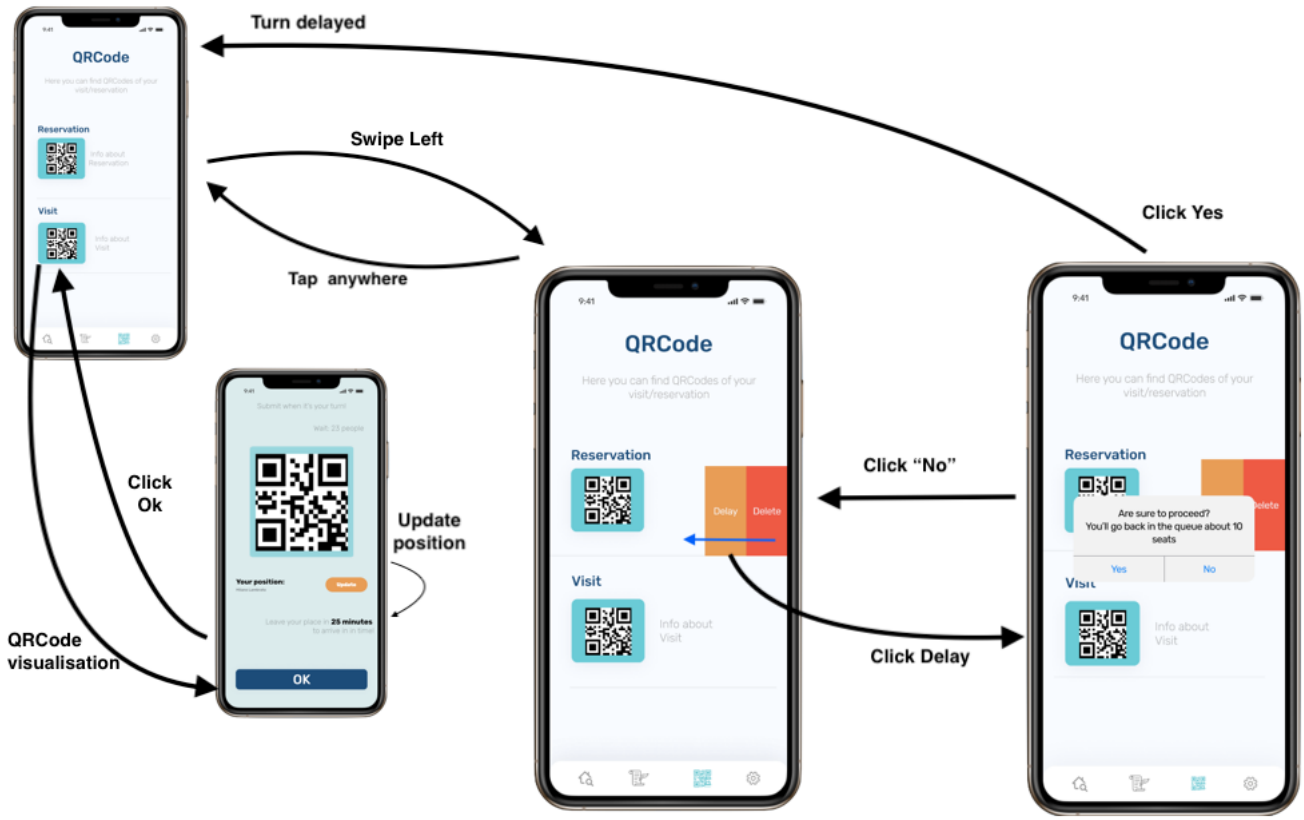


Figure 3.6: The diagrams shows how it's possibile, from the QRCode section, to postpone the own turn in queue for a Reservation. In particular the following screenshoots illustrate the procedure in a scenario in which a user has booked both Reservation and Visit.

3.2 Desktop Interface: CLup Operator

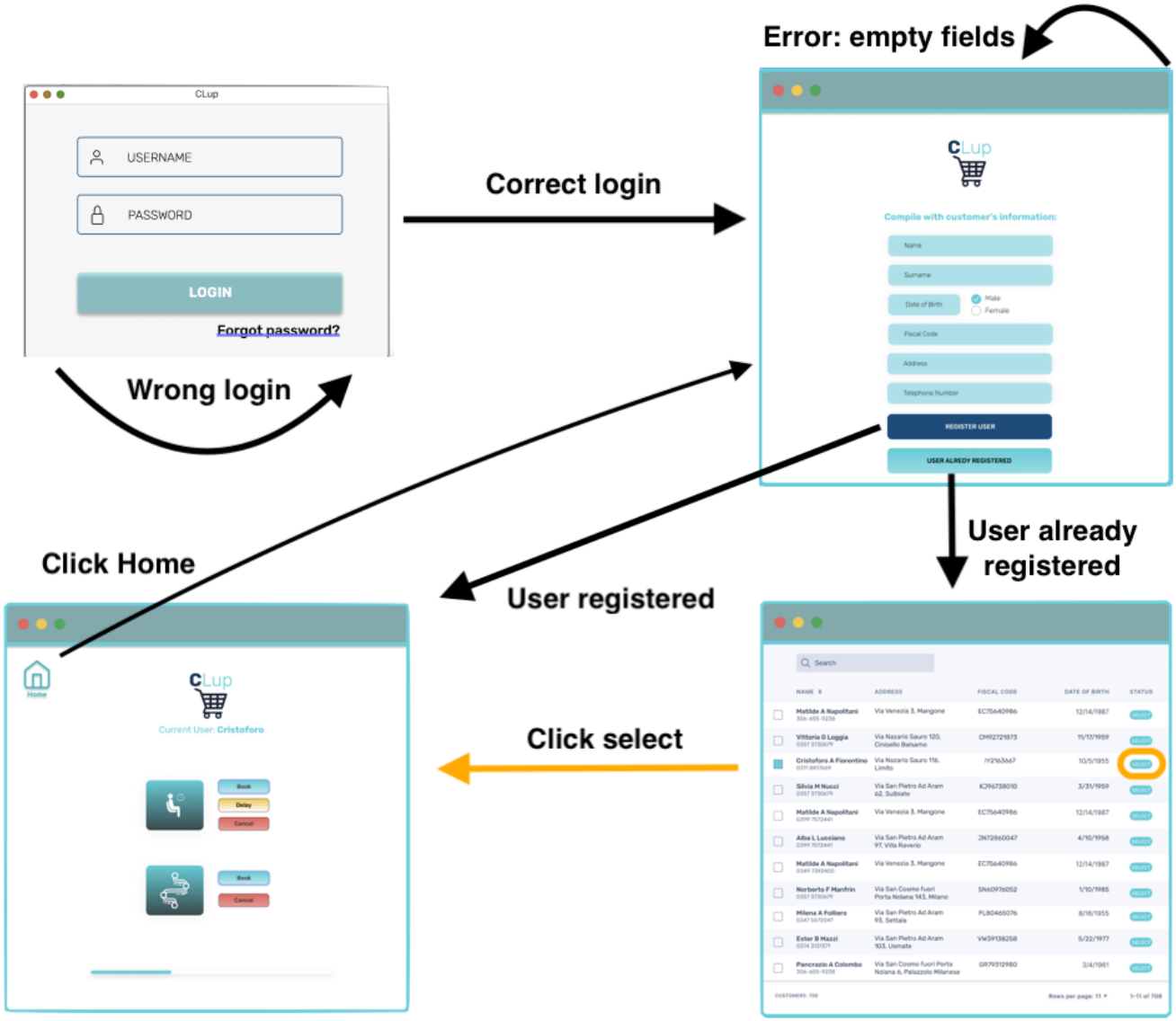


Figure 3.7: A receptionist must authenticate himself before taking into account the user's request. After this the receptionist is able to select an existing or register a new user in order to satisfy his request.

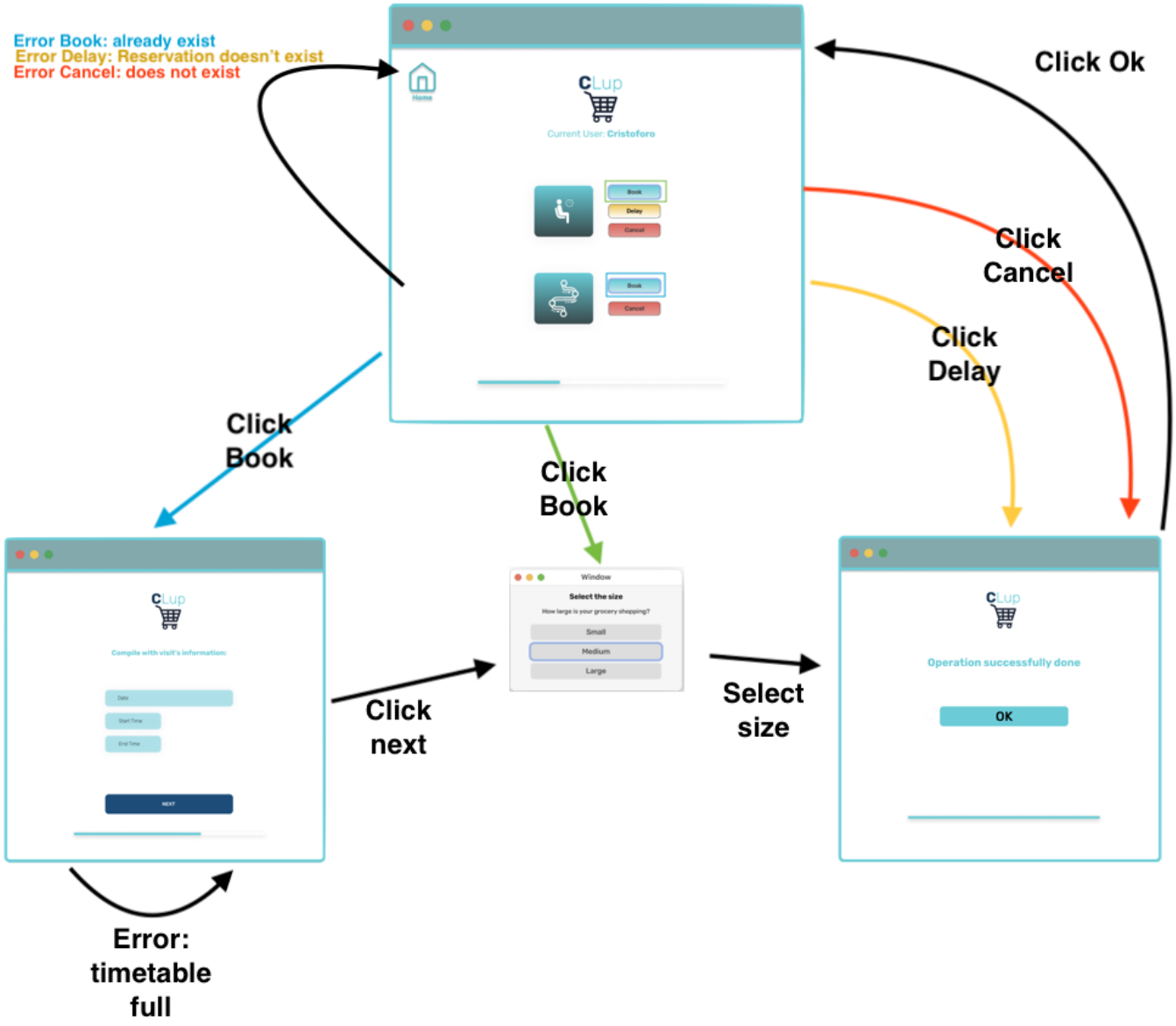


Figure 3.8: The following screenshots illustrate how a receptionist can manage user's request through some steps.

Chapter 4

Requirements Tracebility

The purpose of this section is to associate all goals and requirements defined in the RASD (section 3.2) at each elements of our system physically. Showing the requirements traceability is really necessary for a good comprehension due to implementation. In particular each requirement will be assign to the components needed to achieve it. Each component is accurately described in the section 2.2.

G1 User enters once arrived at the market. **[R1]-[R2]-[R3]-[R4]-[R5]-[R8]**

- **GUI (CLup)**;
- **QR Reader**;
- **Network Manager (Server)**;
- **Controller**;
- **RDBMS**;

G2 Put a limit to the number of Users in the market. **[R4]-[R6]-[R7]**

- **Notification Manager**;
- **Network Manager (Server)**;
- **Controller**;
- **RDBMS**;

G3 Smart User can make a Reservation. **[R8]-[R10]-[R11]-[R15]-[R18]**

- **Auth User**;
- **GUI (CLup)**;
- **Request Handler**;

- **Network Manager (CLup)**;
- **Network Manager (Server)**;
- **Reservation Queue**;
- **Controller**;
- **RDBMS**;

G4 Smart User can make a Visit. **[R10]-[R11]-[R14]-[R15]-[R16]-[R18]**

- **Auth User**;
- **GUI (CLup)**;
- **Request Handler**;
- **Network Manager (CLup)**;
- **Network Manager (Server)**;
- **Visit Schedule**;
- **Controller**;
- **RDBMS**;

G5 Mobile User can make a Reservation. **[R2]-[R8]-[R12]-[R13]-[R14]-[R17]-[R21]**

- **Auth User**;
- **Auth Receptionist**;
- **GUI (CLup Operator)**;
- **Booking Manager**;
- **Network Manager (CLup Operator)**;
- **Network Manager (Server)**;
- **Reservation Queue**;
- **Controller**;
- **RDBMS**;

G6 Mobile User can make a Visit. **[R2]-[R12]-[R13]-[R14]-[R15]-[R16]-[R17]-[R21]**

- **Auth User**;
- **Auth Receptionist**;
- **GUI (CLup Operator)**;
- **Booking Manager**;

- **Network Manager (CLup Operator)**;
- **Network Manager (Server)**;
- **Visit Schedule**;
- **Controller**;
- **RDBMS**;

G7 Smart User can cancel a Booking. [**R10**]-[**R11**]-[**R19/R20**]

- **Auth User**;
- **GUI (CLup)**;
- **Request Handler**;
- **Network Manager (CLup)**;
- **Network Manager (Server)**;
- **Reservation Queue / Visit Schedule**;
- **Controller**;
- **RDBMS**;

G8 Mobile User can cancel a Booking. [**R12**]-[**R13**]-[**R17**]-[**R19/R20**]-[**R21**]

- **Auth User**;
- **Auth Receptionist**;
- **GUI (CLup Operator)**;
- **Booking Manager**;
- **Network Manager (CLup Operator)**;
- **Network Manager (Server)**;
- **Reservation Queue / Visit Schedule**;
- **Controller**;
- **RDBMS**;

Chapter 5

Implementation, Integration and Test Plan

5.1 Implementation

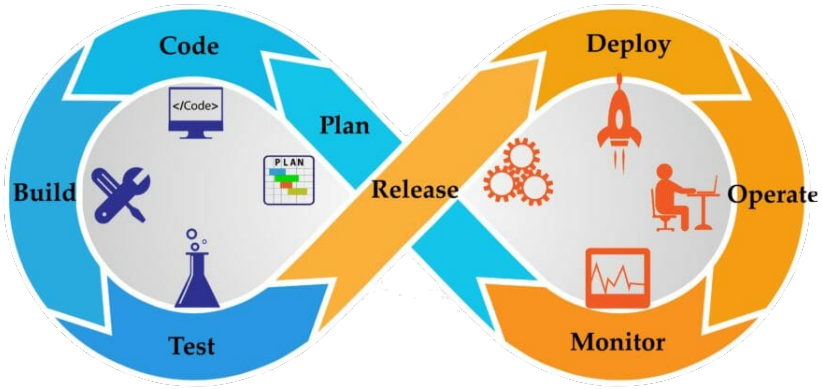


Figure 5.1: DevOps approach.

The implementation follow the **DevOps approach**. According to this model, the developer and operation teams work together, across the entire lifecycle, from the development/deployment to test operations. This approach entails a lot of benefits:

- **speed**: quickly it adapts to market needs and perform faster the implementation of the new features;
- **rapid delivery**: it consists in light and frequent updates in order to fix

any bugs and make the application more secure in case of vulnerability. The frequent update makes easy finding and fixing any bugs;

- **scalability**: the DevOps approach helps us to manage our development, testing, and production environments in a repeatable and more efficient manner;
- the use of **microservices**, which are indipendent from each other and from the system, helps to implement new features and allow to modify only the component involved.

5.2 Integration

We'll use the following ones, which are the main methods for a DevOps approach. These improve developers interactions and reduce time for next releases.

Continuous integration Continuous integration is a software development practice where developers regularly update any code modification into the main branch of the central repository as often as possible. In this way builds are created and test runned due to a better collaboration. The target of CI is to find potential bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery CD is an extension of continous integration where, instead, code changes are automatically built, tested, and prepared for releases. In this way, developers will always have a deployment-ready build artifact to run. It also includes unit test in order to automate testing which improve productivity. Moreover, bugs are solved with more readiness.

It will be used **Git**, an open source distributed version control system, as source control and **Jenkins** as continuous integration. Jenkins is a popular open source tool to perform continuous integration and build automation. It also monitors the execution of the steps and allows to stop the process, if one of the steps fails. Jenkins can also send out notification in case of a successfull or failure build.

5.3 Testing

Continuous Testing needs to be a key element in DevOps approach for a right implementation.
This is a procedure in which testing is performed earlier in the software lifecycle, by increasing quality, shortening long test cycles and reducing defects in software. This is necessary with DevOps because software moves continuously from development to testing to deployment.

Another important aspect of testing a product software is the **Quality Assurance**. QA testing is the process that ensures the best quality possibile of the

software developed. This is made by combining automated tests and manual testing. In our case, QA is required to be aligned with respect to a DevOps cycle. This is due to the fact that DevOps don't have to check software quality as an external entity. QA infact will be a part of the development process made by each team. So testers have to work very close to developers to collaborate on testing for the entire software development lifecycle.

Chapter 6

Effort Spent

In the following tables it's summerized the effort time spent from us. In particular it has to be mentioned that we work together

Chapter/Task	**Hours**
Git setup	
Introduction	
Architectural Design	
User Interface Design	
Requirements Traceability	
Implementation Integration Test	2
	Total
	35,5

Table 6.1: Matteo Bresciani's effort

Chapter/Task	**Hours**
Git setup	2
Introduction	2
Architectural Design	2
User Interface Design	2
Implementation Integration Test	2
Requirements Traceability	1
	Total
	35,5

Table 6.2: Stafano Banfi's effort

Chapter 7

References

7.1 Software used

- **LaTeX**: used to write and to build the document [https://www.draw.io/];
- **Draw**: used to create class, sequence and case diagram [https://www.draw.io/];
- **GitHub**: used to store and manage project repository [https://github.com/];
- **GitHub Desktop**: is the official GitHub application which allows us to contribute to the project repository in an easy way [https://desktop.github.com];
- **Figma**: used to design mockups provided in this document [https://www.draw.io/];

7.2 Bibliography

- Slides of *Software Engineering 2* course [https://beep.metid.polimi.it/];
- *R&DD Assignment A.Y. 2020-2021* [https://beep.metid.polimi.it/];
- *AA 2020/2021 Software Engineering 2 - Requirements Analysis and Specification Document- Bresciani Matteo, Stefano Banfi* [];
- *Oracle DB documentation* [https://docs.oracle.com/cd/E24693_01/nav/development.html];
- *Amazon Web Service documentation* [https://aws.amazon.com/devops/];