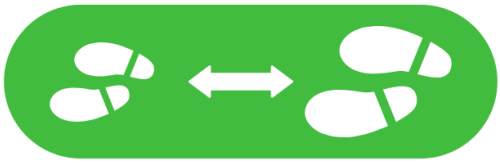
POLITECNICO
MILANO 1863

Computer Science and Engineering

Software Engineering 2 Project

Design Document

Version 2.0 - 10/01/2021



Customers Line-up

STAY SAFE

Authors

Enrico Gherardi
Ludovico Righi

Professor

Elisabetta Di Nitto

Tutor

Annalisa Binato

Contents

List of Figures

Chapter 1

Introduction

This chapter explains the purposes and scopes of this document and gives some keys of interpretation of it (abbreviations, definitions..).

1.1 Purpose

The purpose of this document is to give a more detailed description of the architectural choices made to perform into the software system all the functionalities described into the RASD document. Therefore, if the RASD describes all the features that the system must indulge according to the stakeholders needs, the DD describes how it has been thought to convert all the high-level system requirements into the design structure. It provides guidance and material which is intended to assist technical staff to perform system testing and implementation and, moreover, to facilitate future maintenance and features extension. List of topics covered by the document:

- High-level architecture.
- Main Components, interfaces and deployment.
- Runtime behavior.
- Design Patterns.
- UI details.
- Mapping between requirements and architecture.
- Implementation, integration and testing plan.

1.2 Scope

Here a short review of Customers Line-up scope referred to what just stated in the RASD document.
Customer Line-Up is a crowdsourced application whose purpose is to create a service to support people, specially in lockdown time, in doing grocery shopping. Anyone can Line-up from his home in a specific supermarket simply opening Customer Line-up application. For each Line-up request, every Customer has only to select in which supermarket he intends to go and with which mean of transport. After that, the System shows to the Customer his Waiting Time and generates for him a QR code that has to be scanned at the entrance and the exit of the store. Furthermore, the System retrieves the Customer's position and computes the duration of the trip to the supermarket considering the current traffic situation and the mean of transport selected. In this way, the System can send to the Customer a notification when he has to start. To let everyone to grocery shopping in a safe way, Customer Line-Up provides a fallback option for people who do not have access to the required technology. In fact, No-Tech Customers can be reported to the System by the Store Manager and can enter the store with a Ticket.
The software provide also a service to book a Visit in a specific Supermarket. The Customer uses the mobile phone interface to make a Booking Request indicating the involved supermarket, the date and time of the Visit and its Expected Duration. If the Customer is a Long-Term one, the system can mine this last parameter from previous data. Additionally, the Customer can indicate the exact list of items he intends to buy or their categories to enable the system to make a finer optimization allowing more people in the store. The system shows only dates and times when the supermarket is open and the number of its Bookings is lower than its capacity. It is important to underline that the system allows the Customer to make a single Booking Request at a time to permit a more fair distribution of the timeslots available. Also, for usage semplicity reasons, it permits only to cancel and not to modify the Request.
Finally, The system provides to the Store Managers a dedicated portal where they can check in real time the number of people inside their supermarket and the list of all the Bookings. Through this portal, they can report the entrances and the exits of No-Tech Customers.

1.3 Definitions, Acronyms, Abbreviations

This section gives some definitions in 1.3.1, acronyms in 1.3.2 and abbreviation in 1.3.3 which will be use in the document, in order to explain some concept and help the general understanding.

1.3.1 Definitions

- Customer: the user of the application who can line-up from his home or can book a visit.
- No-Tech Customer: the Customer of the supermarket that, for external reasons, cannot have an access to CLup in any way and has to enter the store with a ticket.
- Long-Term Customer: the customer who entered the supermarket using CLup at least 10 times.
- Store Manager: the user of the application that can monitor the affluence of his store, have to report to the system all the arriving No-Tech Customers and have to give them the tickets to enter.
- User: a person who uses the CLup software: he can be a Customer or a Store Manager.
- Waiting Time: the time the Customer have to wait before entering the supermarket.
- Real-Time Queue: the queue of all the Customers who had lined-up, who had booked and those without the required technology for a specific supermarket.
- Inferred Duration: the weighted average over the days and hours of the duration of a Long-term Customer's previous visits.
- Line-up Request: the request of a Customer for lining-up from his home in the selected supermarket.
- Booking Request: the request of a Customer to book a Visit in a specific supermarket.

- Visit: the result of a Booking Request.
- Ticket: a number generated by the System that permits to No-Tech Customers to enter the store.

1.3.2 Acronyms

- $RASD$: Requirement Analysis and Specification Document.
- DD: Design Document.
- API: Application Programming Interface.
- GPS: Global Positioning System.
- $DBMS$: DataBase Management System.
- $HTTPS$: HyperText Transfer Protocol Secure.
- TCP/IP: Transmission Control Protocol / Internet Protocol.
- RMI: Remote Method Invocation.

1.3.3 Abbreviations

- $CLup$: Costumers Line-up.
- G_N: Goal number N.
- R_N: Requirement number N.

1.4 Revision History

- Version 1.0 : 10/01/2020
- Version 2.0 : 07/02/2020
  In this version the following changes have been perfomed:
  - Updated Customer and Supermarket's attributes in the Class diagram.
  - Fixed a typo ("Smartphone OS" in WebServer) in Deployment view.

- Updated description of Line-Up Request and Cancel Booking Sequence diagrams.
- Check coerence with respect to the revised Ticket entrance system.
- Updated Effort Spent.

1.5 Reference Documents

- Specification document: "DD Assignment A.Y. 2020/2021".
- Lecture slides of professor M.Rossi and E.di Nitto of Politecnico di Milano.
- Customers Line-up RASD document.

1.6 Document Structure

The DD document is structured by seven chapters as describes below:

- Chapter 1: It presents an introduction. It is formed by a general description of the purpose and scope of the document and a list of definitions, acronyms and abbreviations used in the document.
- Chapter 2: It describes the architecture of the system with different representations, beginning with an overview, following with the component, sequence and interface diagrams to describe all the components and their interaction. In the end describes also the architectural and deployment choices.
- Chapter 3: it presents the mockups of the user interfaces that are close to the final layout.
- Chapter 4: It describes a mapping between the requirements listed in the RASD document and the components that manage them.
- Chapter 5: It describes the main guideline to the implementation and testing adopted for Customers Line-Up, in particular it specifies the flow of implementation and the flow of integration of each subsystem.

- Chapter 6: contains the effort, in terms of hours, which each member of the group spent working on the project.
- Chapter 7: includes the reference documents.

Chapter 2

Architectural Design

This chapter contains the description of the main architectural design choices. The section 2.1 describes the architecture in an high level, the section 2.2 describes the component of the system, the section 2.3 shows the deployment diagram to describe how the system will be physical deployed, the section 2.4 shows with sequence diagrams the flow of each components, the section 2.5 describe the main interfaces of the components, the section 2.6 describe the selected architectural style and patterns and the last section 2.7 describes other decisions about algorithms and data structures.

2.1 Overview

The software to be is designed as a Client-Server system with a three-tier architecture: Presentation Level, Business Logic or Application Layer and Data Access Layer. In particular:

- **Presentation Level**: handles the interaction with Customers and Store Managers. It contains the interfaces able to communicate with them and it is responsible for rendering of the information. Its scope is to make understandable the functions of the application to the users. In particular, this levels provides an interface both to the Customer (a mobile application) and Store Manager (a web application) in order to allow them to exploit, in a simple way, all the services of the application. The purpose is to make them as efficient and intuitive as possible.

- **Business logic or Application layer**: takes care of the functions to be provided for the users. It also coordinates the work of the application, making

logical decisions and moving data between the other two layers. In particular, it contains the Web Server and the Application Server.

- **Data Access Layer**: cares for the management of the information, with the corresponding access to the databases. It picks up useful informations in the database and passes them along the other layers.

This architecture has been chosen in order to make the software scalable and flexible to future integrations and modifications, without giving up a fair robustness and a good user experience. The communication between the various components is managed through the exchange of requests and responds, so, some of the main design choices have been taken in order to optimize the expenditure of the available resources. The client software is of limited complexity, acting in many cases only as an interface to the servers and computing only the essential functionalities. The Application Server is used to manage the main application logic; moreover, it contains the management of some of the functions that users do not need to access, or the ones that require an high expenditure of resources. One of the main purposes of the Application Server is to send requests to the database (in the Data Access Layer); in particular, by communicating with its DBMS it can handle, when necessary, all the operations of reading, writing and uploading of data.

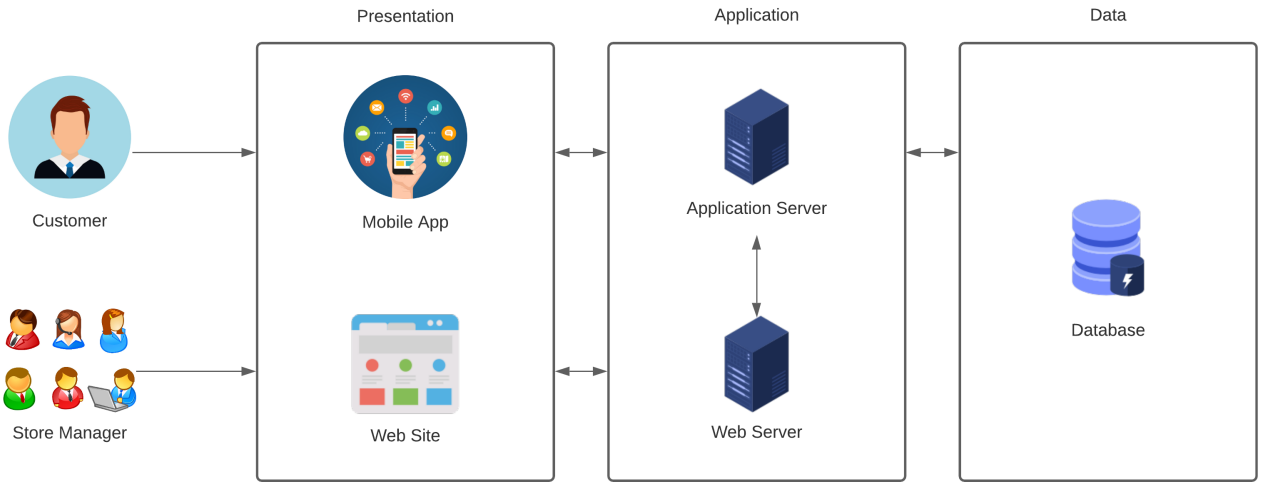


Figure 2.1: High-Level View of the Application Architecture

2.1.1 Class Diagram

The Class Diagram presented in the RASD document showed in a generic way the problem; instead, the following diagram has the objective to present the solution to that problem. For this reason, it is more implementation-oriented and detailed with respect to the one inserted in the RASD, and so it is more useful for developers. In particular, the following changes have been performed:

- Introduction of the main methods of StoreManager and Customer classes.
- Introduction of the User class, which is a generalization of the two classes StoreManager and Customer.
- Introduction of the data type of the attributes.
- Removal of the NoTechCustomer class, since it was useful only for modelling the problem and not the solution.

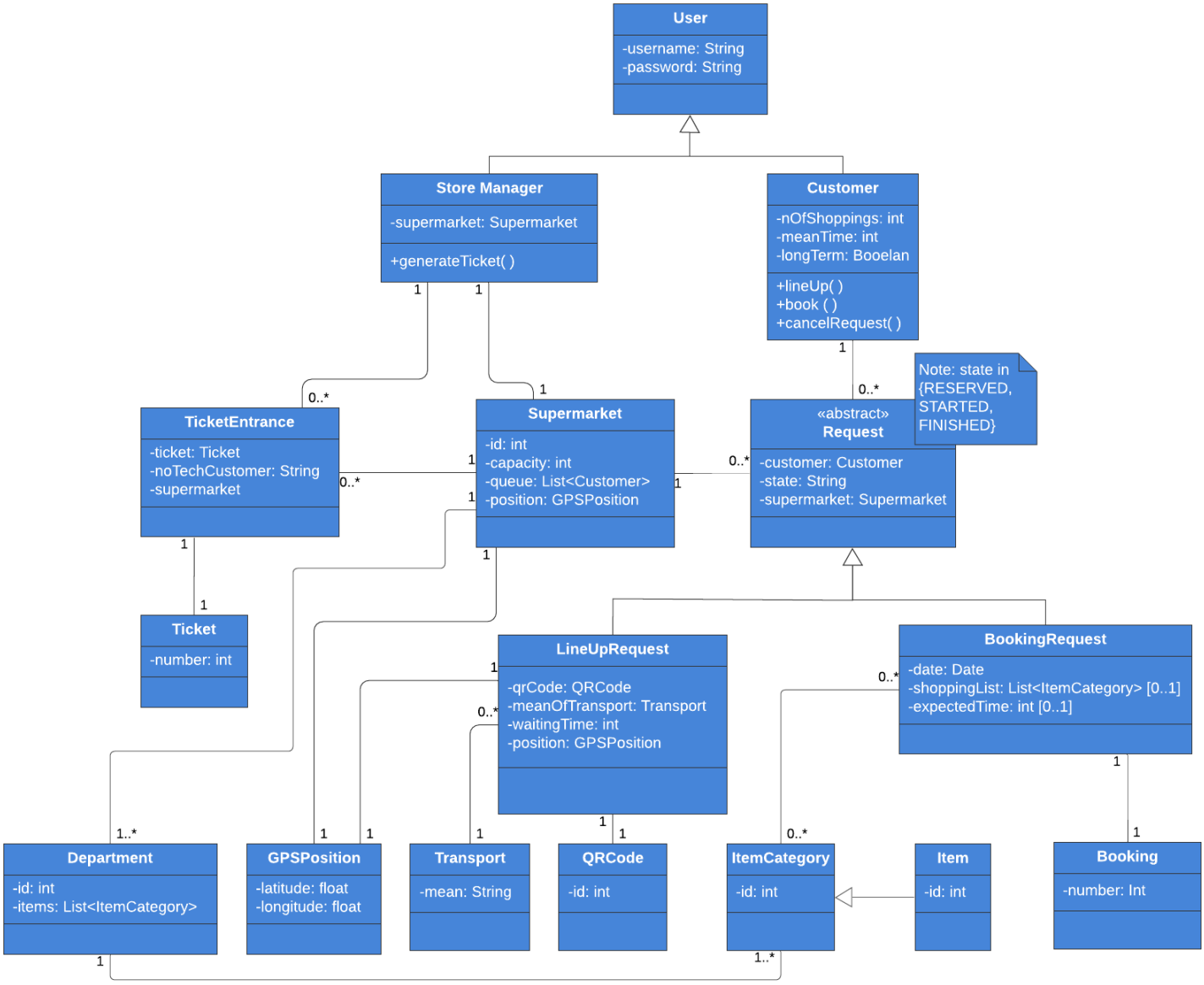


Figure 2.2: Class Diagram

2.2 Component View

The following Component Diagram gives a specific view of the system focusing on the representation of the internal structure of the Application Server, showing how its components interact. The Application Server contains the Business Logic of our software. The other elements in the diagram have been depicted in a simpler way

just to show how the communication is structured among these components and the Application Server. The dashed arrows are used to connect the Application Server components to external ones.

- **GPSService**: this component permits to the MobileApplication to connect to the GoogleMapsServices. This choice is made for future maintainability in the case of future mapping service's changing. It retrieves the informations needed for the Google Maps APIs and communicates the time needed to reach the supermarket to the MobileApplication.
- **DatabaseAccess**: This component permits to access the database and gives more maintainability in the case of future DBMS's changing.
- **StoreManagerService**: this component comprises two subparts:
  - **TicketService**: this component permits to report a No-Tech Customer to the RealTimeQueueManager. In particular, this it can report the entrance of a No-Tech Customer and also his exit.
  - **AffluenceService**: it is meant for the Store Managers which access the WebApplication and wants to monitor the affluence of their Supermarket. It communicates with the RealTimeQueueManager to retrieve the number of people inside the store and the ones who have lined-up. It is also connected with the DatabaseAccess component to retrieve the list of Bookings.
  - **AuthenticationService**: it is meant for the Store Managers which access the WebApplication and needs to log in to be able to use all the functionalities to which it is subscribed. It is also used to register a new Store Manager.
- **CustomerServices**: this component comprises two subparts:
  - **LineUpService**: this component handles the receipt of the Line-Up Request that Customers do through the Mobile Application. It reports the Customer to the RealTimeQueueManager, generates the corrispondent QR code and returns to the Mobile Application all the informations regarding the succeed Line-Up.

- **BookingService**: it handles the receipt of the Booking Request that Customers do through the Mobile Application. This component is connected to the DatabaseAccess component to retrieve the time slots where the supermarket is open and it is not full and to store the informations about the succeed Booking. Thanks to the informations about the items (or their categories) that some the Customers indicates in their Requests, this component can plan the visits in a finer way. Finally, for Long-Term Customers it can infer the Expected Duration of their visits.
- **AccountManager**: it is meant for the Customers which access the MobileApplication and needs to log in to be able to use all the functionalities to which it is subscribed. It is also used to register a new Customer.

- **RealTimeQueueManager**: This is the core component for the Line-Up and Ticket system. It receives all the Ticket and Line-Up Requests and it is also connected to the DatabaseAccess component to retrieve the list of the next Bookings with their corresponding items. Thanks to all this information, the RealTimeQueueManager can manage the people inside the Supermarket, the people that are queuing from their home and their corresponding Waiting Times.

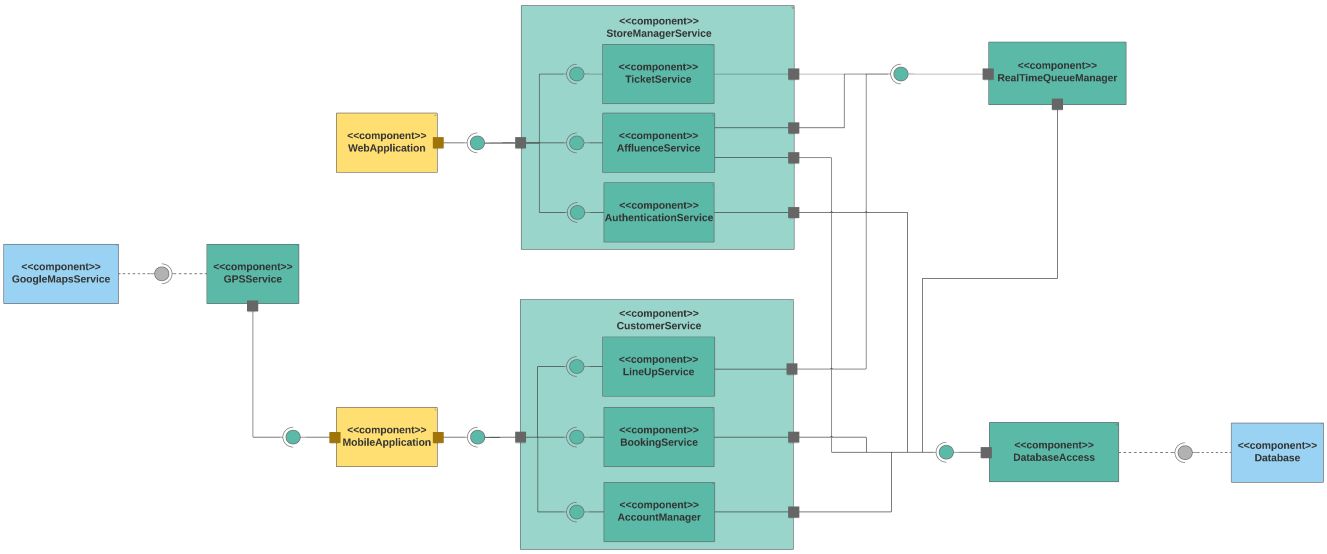


Figure 2.3: Component Diagram

2.3 Deployment View

The following Deployment Diagram describes the execution architecture of the software system; it includes nodes such as hardware or software execution environments and the middleware connecting them. In other words, this kind of diagram describes the physical deployment of information generated by the software program on hardware components: the information that the software generates is called an artifact. The purpose of this section is to visualize how the system will be physically deployed on the hardware. As can be clearly seen from the figure, the system is divided into three tiers:

- **Presentation Tier**: it contains the presentation logic for Customers and Store Managers. In particular:
  - **Smartphone**: Customers exploit the **CLupMobileApp** for accessing the booking and line-up services.
  - **PC/Laptop**: Store Managers have access to their services through a simple **WebApp**.

  All the operations both on MobileApp and on WebApp side are possible through the communication via server, that retrieves all kind of data needed.

- **Application Tier**: the main logic of the application will be deployed here. In particular, it comprises:
  - **Web Server**: this component communicates and serves the WebApp through the HTTPS protocol; it is mainly used for static requests, and so, when dynamic contents are required, it contacts the Application Server using JavaRMI for further processing.
  - **Application Server**: this component contains the actual application logic: it handles all the requests and provides the appropriate answers for all the offered services. This server can communicate directly with the MobileApp through HTTPS protocol; moreover, it handles also some requests of the WebApp that the Web Server cannot satisfy and that, for this reason, are forwarded to the Application Server: in this case, the two servers communicate exploiting JavaRMI. If needed, this component could

be replicated in order to avoid a single point of failure and to guarantee better performances avoiding a bottleneck effect.

- **Data Tier**: will store all the persistent data for the Customers and StoreManagers, such as their usernames and passwords, as well as the Bookings, Line-ups and all the other kind of informations useful for the application. In particular, this tier contains the **DatabaseServer**, which communicates with the ApplicationServer through the standard protocol TCP/IP and provide access to the physical database.

In the system, also other two kind of devices are exploited:

- **Firewall**: this device monitors incoming and outgoing network traffic and permits or blocks data packets based on a set of security rules. Its purpose is to establish a barrier between the internal network and incoming traffic from external sources in order to block malicious traffic like viruses and hackers.
- **Load Balancer**: it distributes the workload across multiple servers to increase capacity; this is needed beacuse CLup is an application which, hopefully, will be used by a very huge amount of users. Moreover, this device can improve the reliability trying to avoid the overload of any single server: this is fundamental because grocery shoppings are one of the most essential needs of the people.

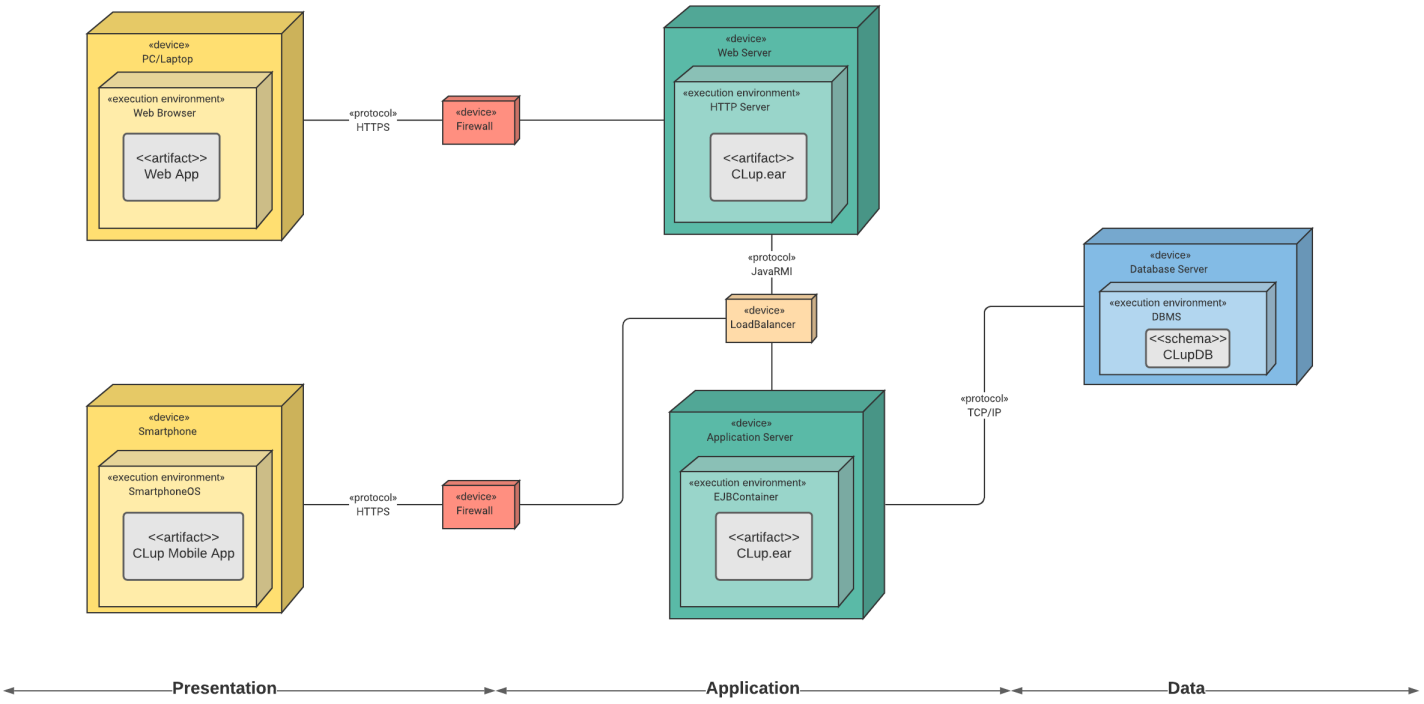


Figure 2.4: Deployment Diagram

2.4 Runtime View

In this section, the most relevant sequence diagrams are shown to describe the way components interact to accomplish specific tasks of the application.

2.4.1 Customer Login

This sequence diagram shows the login procedure executed by a Customer.
Firstly, after having opened the CLup MobileApplication on his smartphone, an already registered Customer fills in the form with his username and password. Then a first simple check is actuated by the MobileApplication itself, which controls whether one or both these two fields are missing, and in that case it shows an error message. Conversely, if this is not the case, the AccountManager is contacted and this component, in turn, propagates the request to the DatabaseAccess that provides the access to the Database, where all the users credentials are stored permanently. Therefore, if in the Database there is couple of values username and password that matches the

one inserted by the Customer, then the login is successfully completed, otherwise an error message is shown.
Notice that, even if not represented in these sequence diagrams, an analogous procedure, based on the WebApplication and AuthenticationService components, can obviously be done also by the Store Manager.

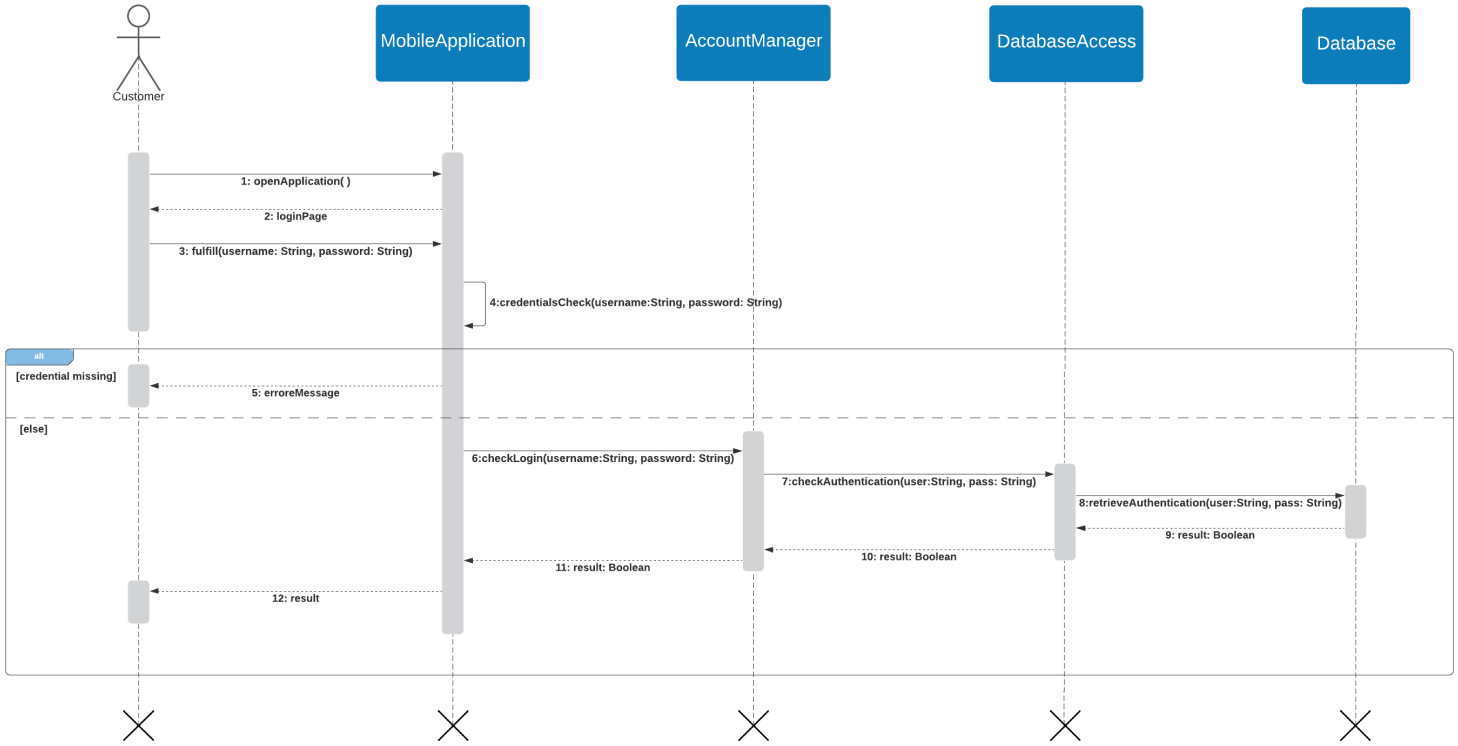


Figure 2.5: Customer Login Sequence Diagram

2.4.2 Store Manager Registration

In this sequence diagram it is shown the process of registration attended by the Store Manager. The Store Manager, using his own computer, after having switched from the login page to the registration one, does the registration filling in the form with his personal username, email, password and the certificate file. Then, the WebApplication controls that all the credentials and the certificate are indicated and that the passwords do not mismatch. Then, the registration request is sent from the WebApplication to the AuthenticationService component. After, this last one calls the DatabaseAccess component to store the information in the Database. So, the result

of this operation is propagated back to the Store Manager, following in reverse order the same path as before, and if the result is positive the registration is successful, otherwise it should be repeated.

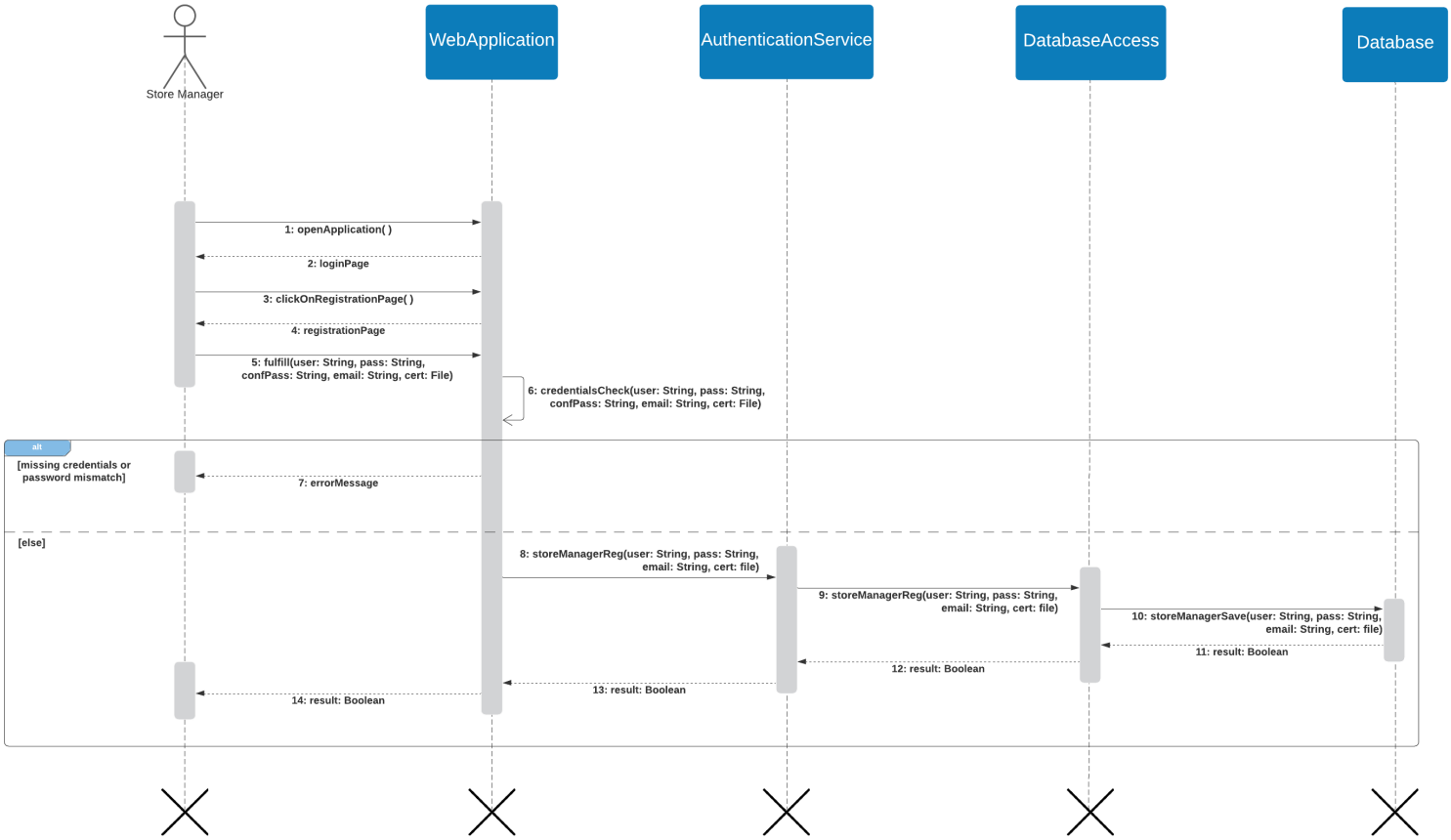


Figure 2.6: Store Manager Registration Sequence Diagram

2.4.3 Line-Up Request

This sequence diagram is used to show the Line-Up process.
Firstly, after having clicked on the Line-Up tab, the Customer selects the supermarket and the means of transport he intends to use to reach the store. The MobileApplication checks if all the data have been inserted and are correct, then it sends the LineUpRequest to the LineUpService component in the Application Server. The LineUpService, in turn, calls the RealTimeQueueManager to add the Customer to the Real-Time Queue and to compute his Waiting Time; then, it generates the corrispondent QR Code and returns the LineUpRequest to the MobileApplication, so

that the Customer is shown all the useful informations.
Notice that the the RealTimeQueueManager does not have to access every time the Database, because it is updated periodically.

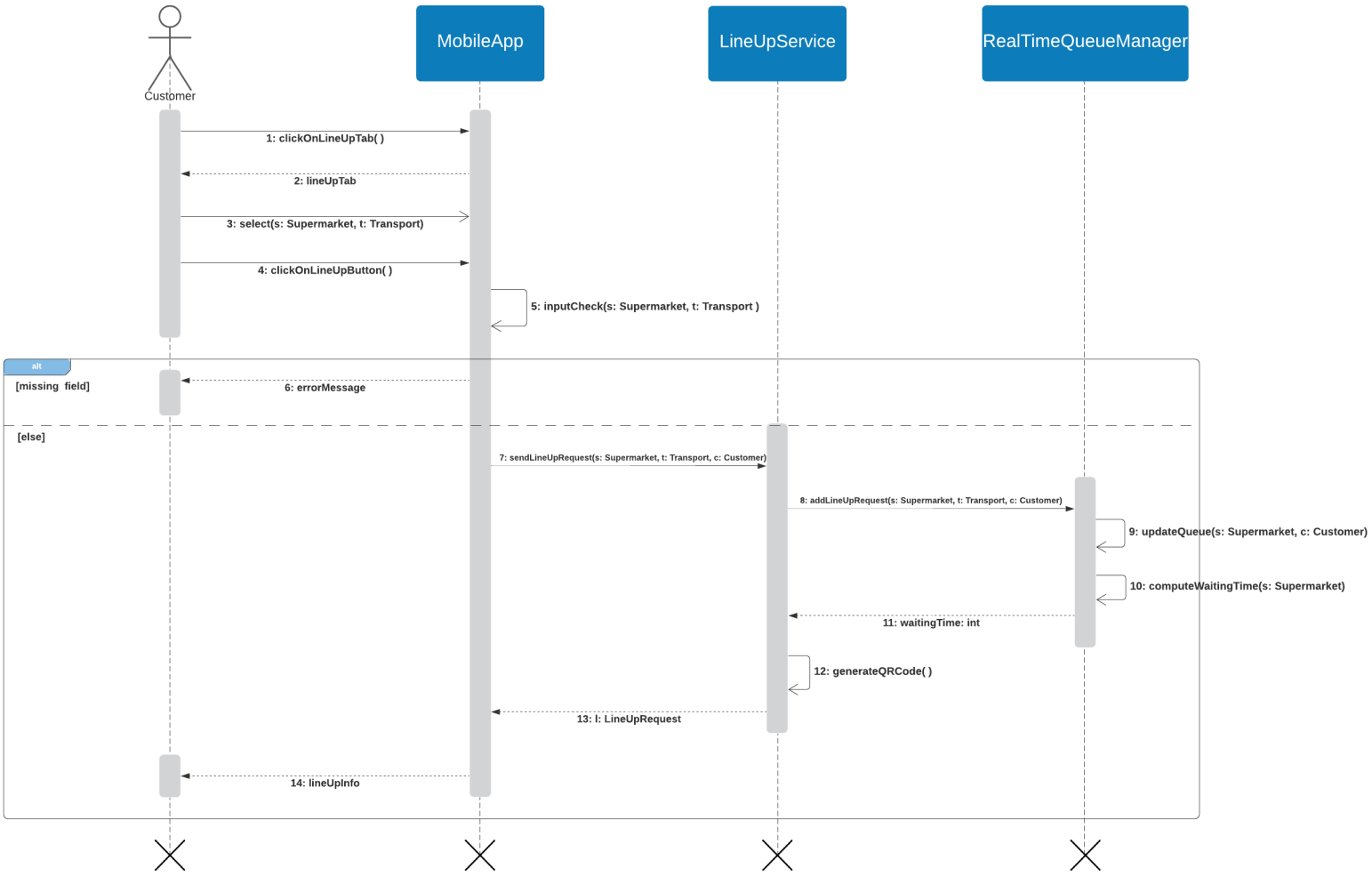


Figure 2.7: Line-Up Sequence Diagram

2.4.4 Booking Request

This sequence diagram is used to show the Booking Process.
Firstly, after having clicked on the Booking tab, an already logged-in Customer selects the supermarket. The MobileApplication uses the BookingService, DatabaseAccess and Database components to retrieve the time slots where the supermarket is open and not full. Then, if the Customer is not a Long-Term one, he must necessary

indicate the Expected Duration of his visit, while instead, if he is a Long-Term one, this information is not mandatory. Moreover, both kinds of Customers can, optionally, indicate the list of items, or their categories, that they intend to buy. Once clicked the Book button, the MobileApplication checks if all the data have been inserted and are correct, then it sends the BookingRequest to the BookingService component in the Application Server. This last one infers the Expected Duration for those Long-Term Customers that have not indicated explicitly the ExpectedDuration, and, thanks to the DatabaseAccess and Database components, store the Booking in the Database. Finally, it update the AADs for the visit-optimizations (see section 2.8.2) and returns the Booking to the MobileApplication, so that the Customer is shown all the useful informations.

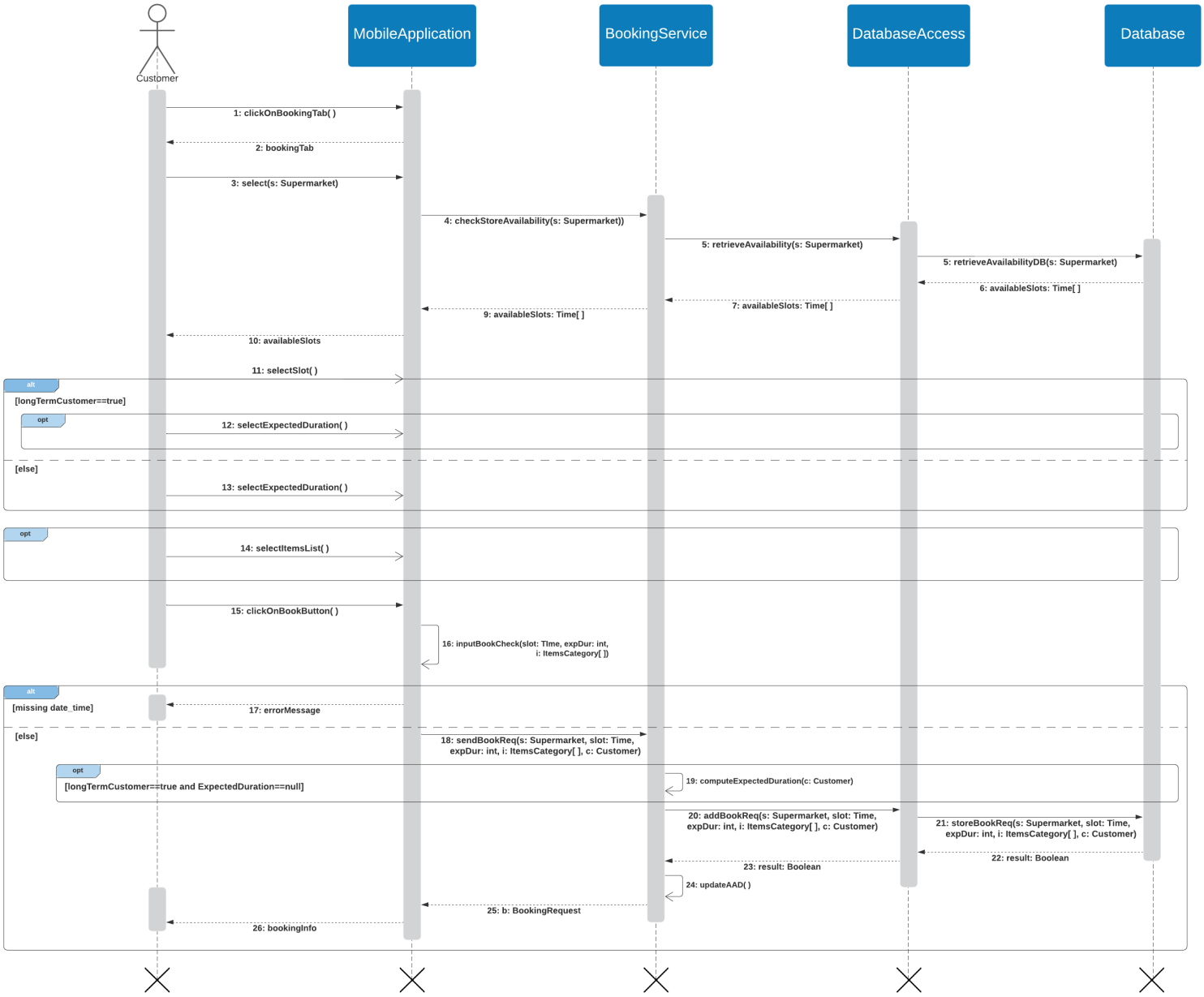


Figure 2.8: Booking Sequence Diagram

2.4.5 Cancel Line-Up

This sequence diagram shows the produre of cancelling a Line-Up performed by the an already lined-up Customer: if instead a Customer has not an active LineUpRequest, he cannot try to do this operation, since the cancel button is displayed only in this

case, as presented and explained in the User Interface section.
Firstly, after having clicked on the LineUpTab and then on the the mentioned button, the request is propagated from the MobileApplication to the LineUpService.
Subsequently, the LineUpService forwards the request to the RealTimeQueueManager that deletes the LineUpRequest from the queue. Finally, a confirmation message is propagated back to the Customer.

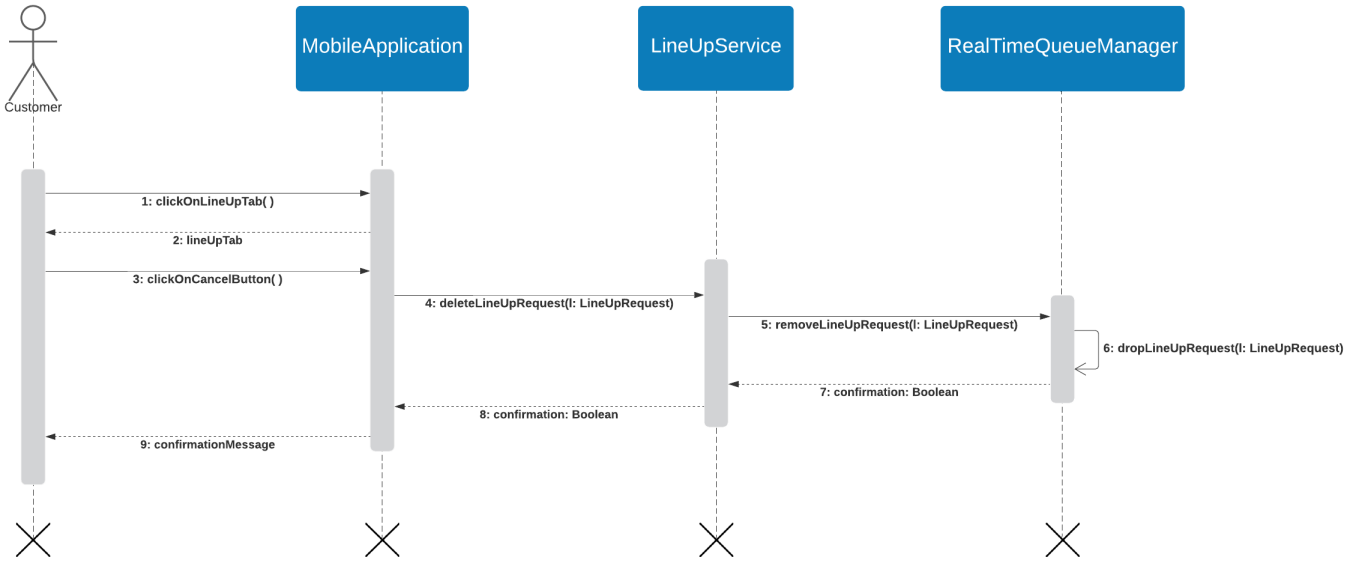


Figure 2.9: Cancel Line-Up Sequence Diagram

2.4.6 Cancel Booking

This sequence diagram shows the produre of cancelling a Booking performed by the an already booked Customer: if instead a Customer has not an active BookingRequest, he cannot try to do this operation, since the cancel button is displayed only in this case, as presented and explained in the User Interface section.
Firstly, after having clicked on the BookingTab and then on the the mentioned button, the request is propagated from the MobileApplication to the BookingService.
Subsequently, the BookingService forwards the request to the DatabaseAccess that provides the access to the Database from which the BookingRequest is deleted through an SQL query. Finally, a confirmation message is propagated back to the Customer. Notice that if the Booking was already in the RealTimeQueueManager, it will be removed at the next periodical update.

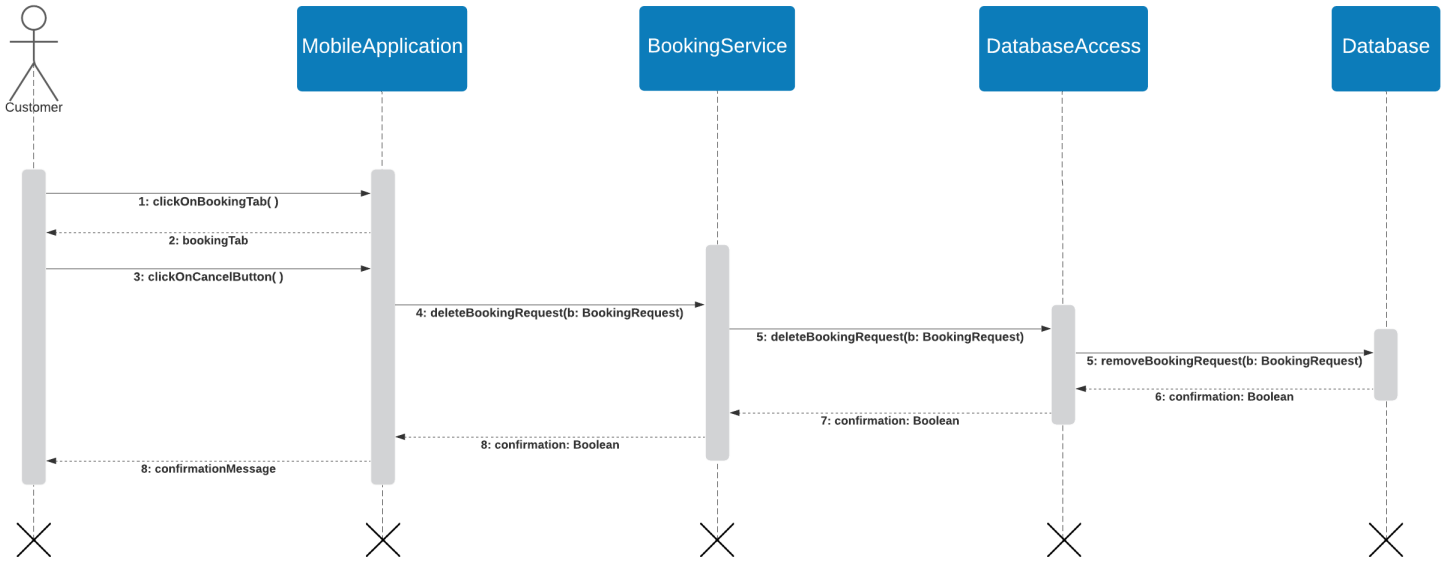


Figure 2.10: Cancel Booking Sequence Diagram

2.4.7 Ticket Generation

This sequence diagram shows the procedure, performed by Store Managers, for generating a Ticket in case a Non-Tech Customer arrives at the supermarket.
Firstly, after having clicked on the NewTicket button, the WebApplication check whether the supermarket is full calling the TicketService, which ,in turn, retrieves this information from the RealTimeQueueManager. In fact, if the supermarket is full, the No-Tech Customer will have to wait that at least one Customer exits the store before entering, in order to always respect the maximum capiency. Therefore, if there is enough space the TicketService generates a new Ticket and the RealTimeQueueManager gets updated counting this new entrance. Otherwise, if the store is full, an error message is shown and the Store Manager will have to repeat the operation.

Figure 2.11: Ticket Generation Sequence Diagram

2.4.8 Send Notification

This sequence diagram shows the mechanism through which the Customer is notified. In particular, the MobileApplication exploits the GPSService to retrieve the position from the GoogleMapsService. A second similar request is done to retrieve the tripTime: at this point the MobileApplication check the WaitingTime and the tripTime values and when these two are equal, sends a notification to the Customer.

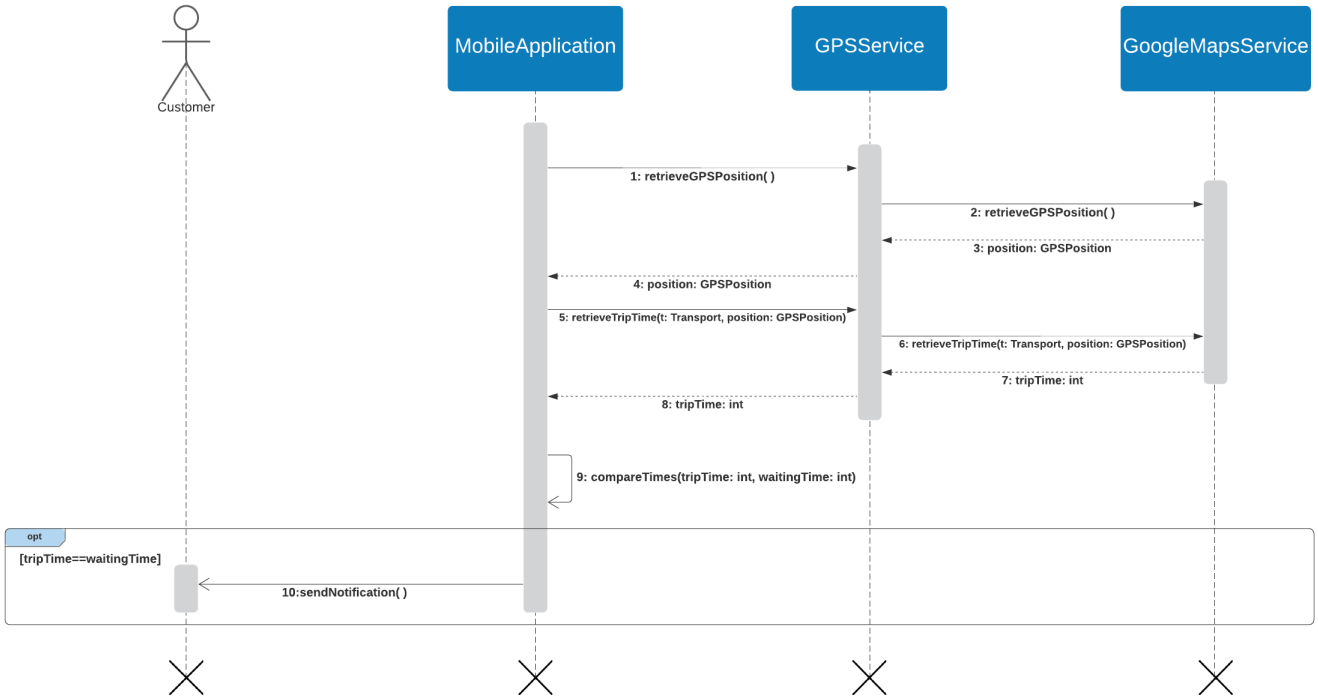


Figure 2.12: Send Notification Sequence Diagram

2.4.9 Check Affluence

In this diagram is shown how a Store Manager can check the affluence if his supermarket. After he has logged in in the and has clicked on the Affluence tab, the WebApplication calls the AffluenceService. This last component calls the RealTimeQueueManager and can return all the needed informations.

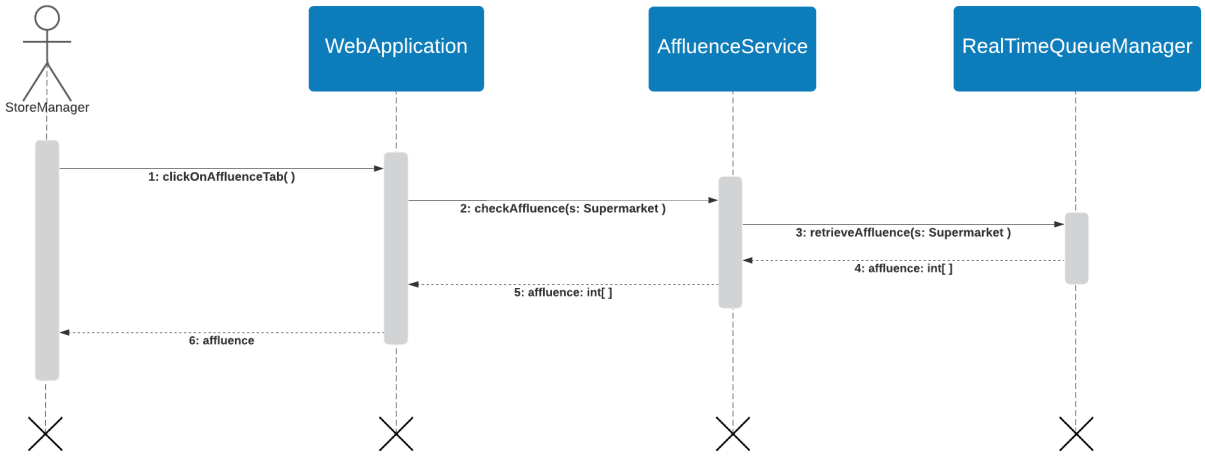


Figure 2.13: Check Affluence Sequence Diagram

2.4.10 Check Bookings

In this diagram is shown how a Store Manager can check the bookings if his supermarket. After he has logged in in the and has clicked on the Booking tab, the WebApplication calls the AffluenceService. This last component calls the DatabaseAccess component that can retrieve all the needed informations from the Database.

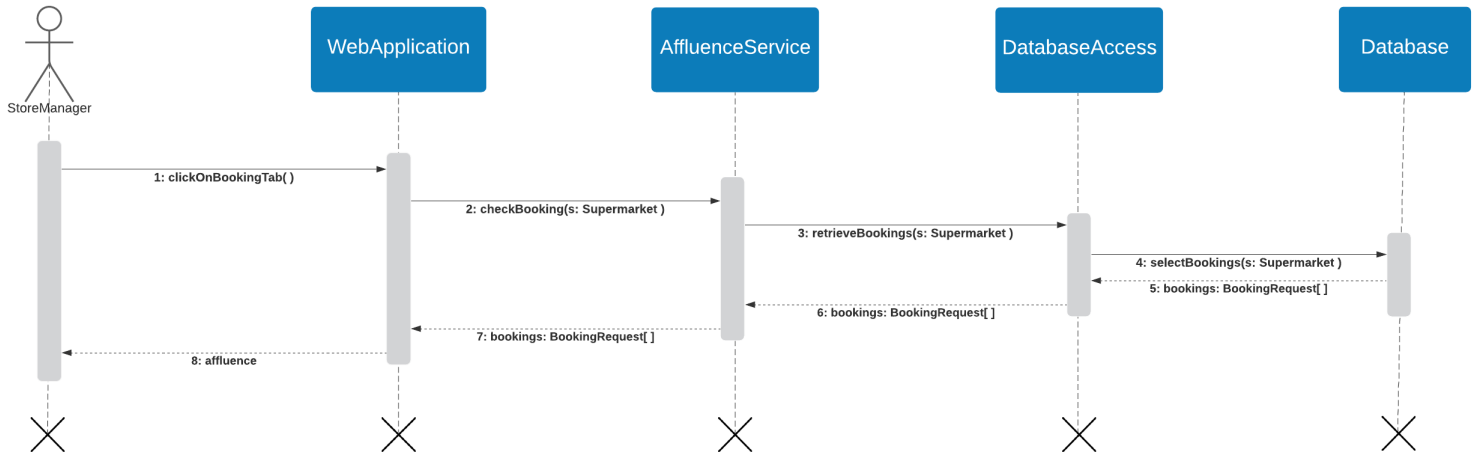


Figure 2.14: Check Bookings Sequence Diagram

2.5 Component Interfaces

In the following diagram are described the main methods which can be invoked on the interfaces and their interactions, referring to the most important processes reported in the Runtime View section.
One aspect is fundamental to be pointed out: in general, methods written in the Component Interfaces diagram are not to be intended exactly as the methods that the developers will write, but they are a logical representation of what component interfaces have to offer. They will be adapted facing the various aspects that will come out during the implementation of the code.

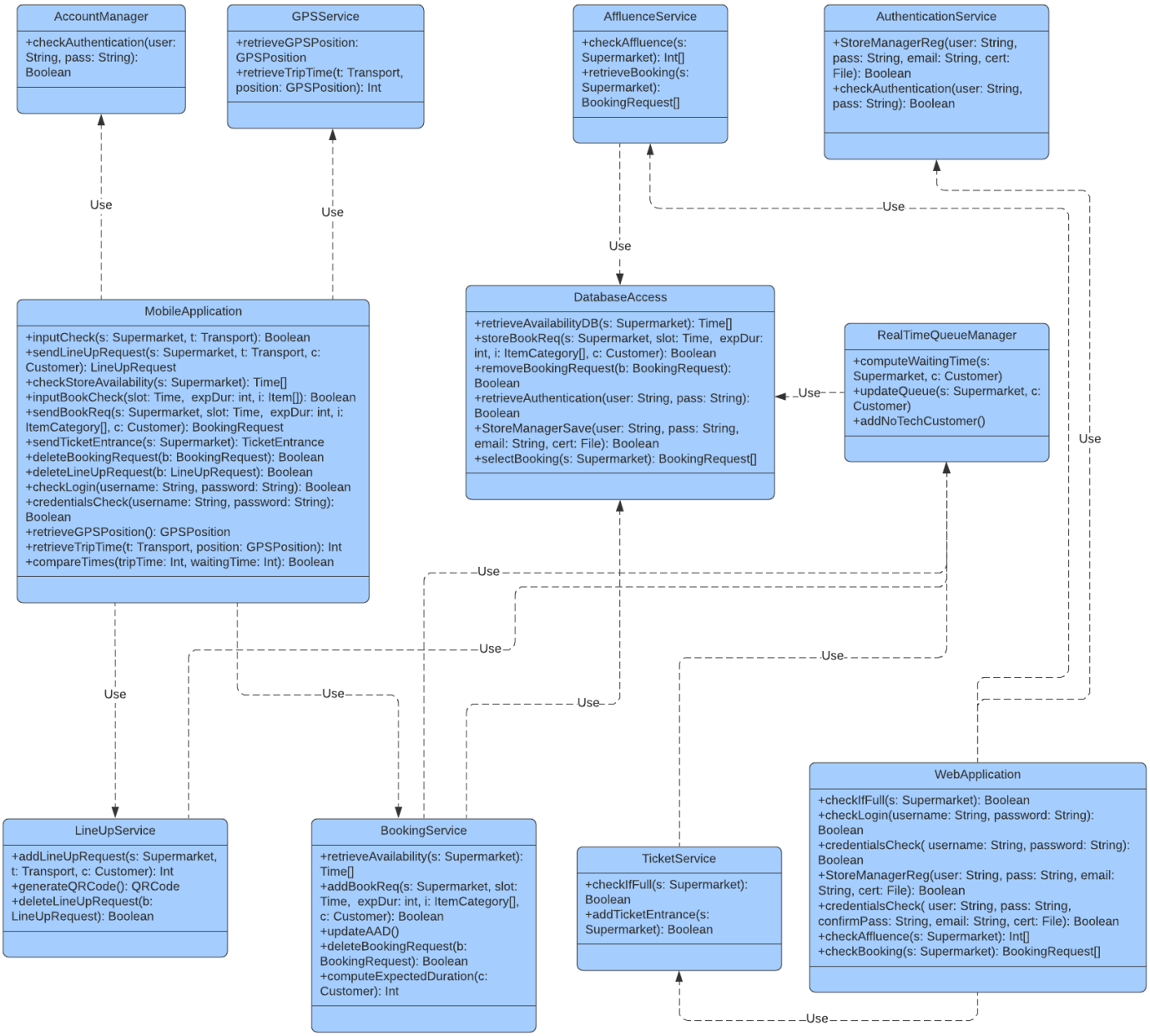


Figure 2.15: Interface Diagram

2.6 Selected Architectural Styles and Patterns

This section presents the decisions about architectural styles and patterns exploited in CLup. In particular the system adopts a Client-Server architecture (see 2.6.1) and

a MVC pattern (see 2.6.2).

2.6.1 Client-Server Architecture

The software to be is designed as a Client-Server system with a three-tier architecture: Presentation Level, Business Logic or Application Layer and Data Access Layer. This design architecture has already been described in section 2.1, and in it is briefly reported here only for completeness purposes.

2.6.2 Model-View-Controller

The Model-View-Controller pattern has been selected in order to guarantee the maintainability and the reusability of the code. This software pattern, which divides the software in three parts, as can be easily deduced by his name, is particularly suitable for the development of the CLup application because it avoid a dangerous level of coupling between the different parts of the system. In particular the aim is to split the internal representation of the informations from the ways A brief description of the three parts is the following:

- **Model**: it is composed by the database and it receives user input through the Controller.
- **View**: it is composed by the MobileApplication UI and the WebApplication UI and so it represents the Presentation level.
- **Controller**: it is composed by all the server components and it represents the logic of the system.

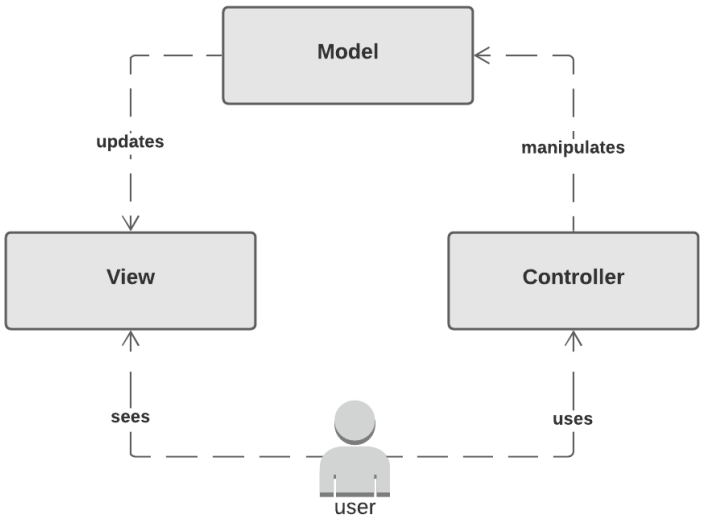


Figure 2.16: Model-View-Controller pattern

Thanks to this it is possible to create components independently of each other and simultaneous development is simplified.
Moreover the MVC schema fits particularly well with other two design patterns:

- Observer pattern: lets one or more objects be notified of state changes in other objects within the system. It is fundamental in the interaction between View and Model.
- Factory pattern: exposes a method for creating objects, for instance both the BookingRequests and the LineUpRequests, allowing subclasses to control the actual creation process.

2.7 Other Design Decisions

This section describes decision about algorithms and data structures used in Customer Line-up. In particular to compute the Waiting Time (see 2.7.1) and the Booking optimization (see 2.7.2).

2.7.1 Next Waiting Time

For this algorithm, we need two data structures.

- A List that contains all the Time To Live (TTL) of all the people inside the store.
- A List that contains all the customer that are queuing from their home.

A Key parameter is AIT (Average Inside Time): the average time that people who have lined-up or are entered with the ticket stay in the supermarket.
The AIT is calculated:

- For No-Tech Customers as the weighted average over days and times of the grocery shopping times of a specific supermarket.
- For Customers who have Lined-Up as as the weighted average over days and times of their grocery shopping times.
- For Customers who have Booked it was previouly indicated as Expected Duration or, only for Long Term Customers, can be infered by the System from their past visits.

The TTL is calculated:

- For No-Tech Customers and Customers who have lined-up as the difference from their AIT and the time they already spent inside the store.
- For Customers who have Booked as the difference from the Expected Duration and the time they already spent inside the store.

If the supermarket is not full, there are not people who are queuing and the waiting time is zero. When the supermarket is full, this algorithm is used to infer the waiting time.

```
public static int nextWaitingTime() {

        if(supermarket.size() < CAPACITY) {
                return 0;
        }

        Collections.sort(supermarket);

        for(int i = queue.size() - 1; i >= 0; —i) {

                int minTTL = supermarket.get(0);
```

```
                WT += minTTL;

                for(int j = 0; j < supermarket.size(); ++j) {
                        supermarket.set(j, supermarket.get(j) - minTTL);
                }

                //The Customer leaves the store
                supermarket.remove(0);

                //The next queuing Customer is removed from the Queue
                //and entres the Store
                supermarket.add(queue.get(i).getAIT());
                queue.remove(i);

                Collections.sort(supermarket);
        }
        return WT;
}
```

2.7.2 Booking Optimization

Customer Line-up permits, on average, to use only the 95% of the total capacity of every supermarket provided by the Prime Minister's Decree (DPCM). This choice has been taken to be more careful and to prevent emergencies. This value of 95% can can increase by 5% depending on the value of the AADs (average affluence of a department).
To compute this value we need two data structures:

- A list for every Booked customer that contains the probability to go to a specific department. For example, if a customer indicates items that belongs to 4 different departments, the probability to reach them is 0.25 and the others is 0.
- A list that contains the preview data structure for all the customers who have booked in a specific time slot.

For every department is computed its AAD. The ADD is the sum of all the probabilities of every booking customer to go to the specific department divided by the capacity of that one.
If an AAD is higher than 1 for at least one department, then the maximum number of people inside the store remains the 95% of the supermarket capacity. Else it is the 95% + (5% * (number of booked people / capacity)).

To better dispay the AAD formula, the second data structure is represented as a matrix where the columns (j) are the departments and the rows (i) are the customers.

$$AAD_j = \sum_i M_{ij} / DepCapacity_j$$

Chapter 3

User Interface Design

CLup contains two user interfaces: the first one is the mobile application, which is used by Customers to make Line-Up Requests and Bookings. The other one is the website interface, used by Store Managers to check the supermarket affluence and to request tickets for No-Tech Customers. The following mockups, some of which were already presented in the RASD document, shows the idea of how the system should work.

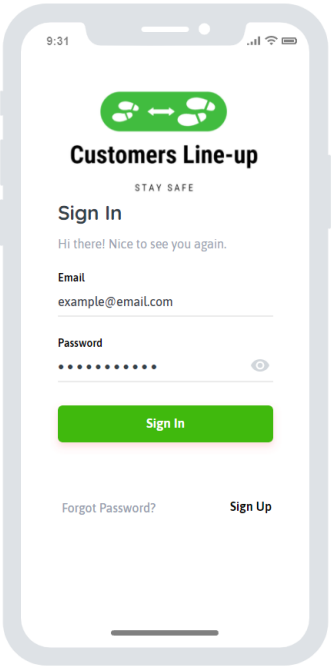


Figure 3.1: Login

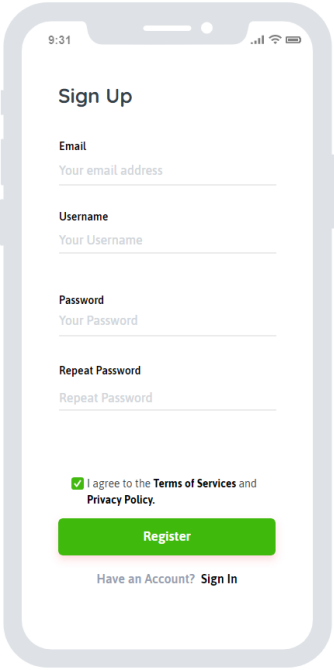


Figure 3.2: Registration

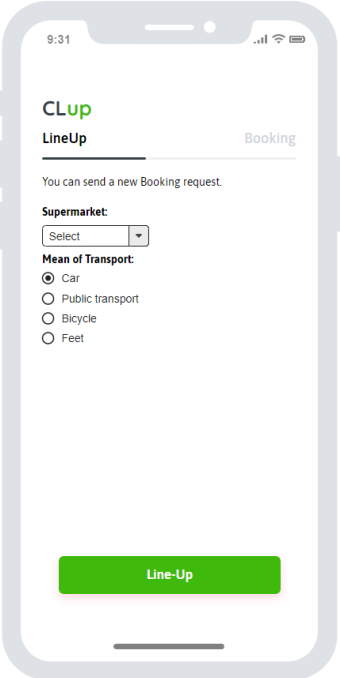


Figure 3.3: New Line-Up

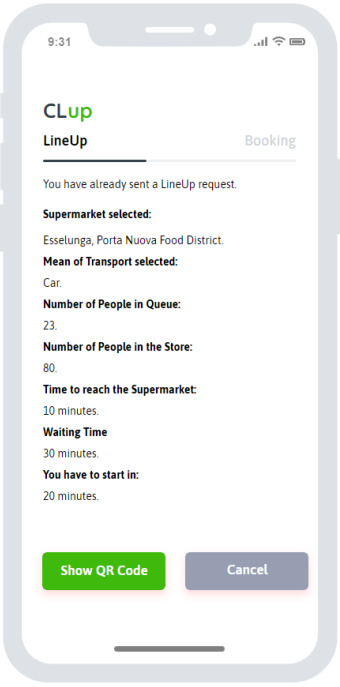


Figure 3.4: Line-Up sent

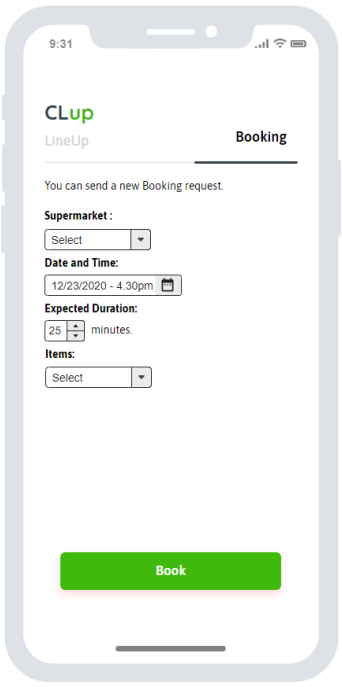


Figure 3.5: New Booking

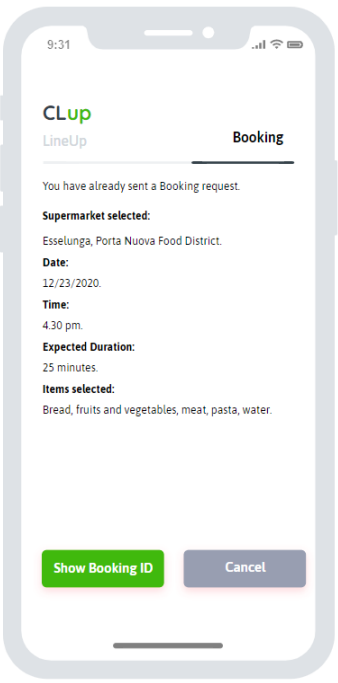


Figure 3.6: Booking sent

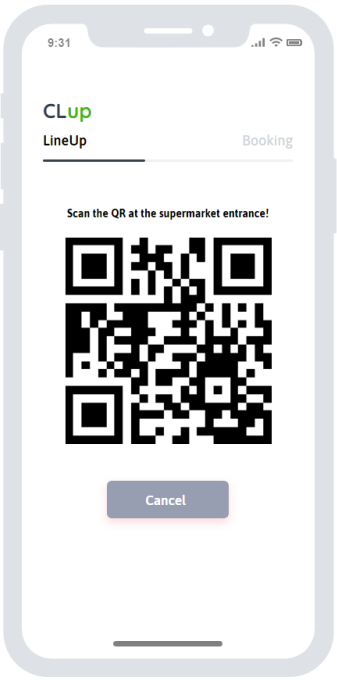


Figure 3.7: Scan QR Code

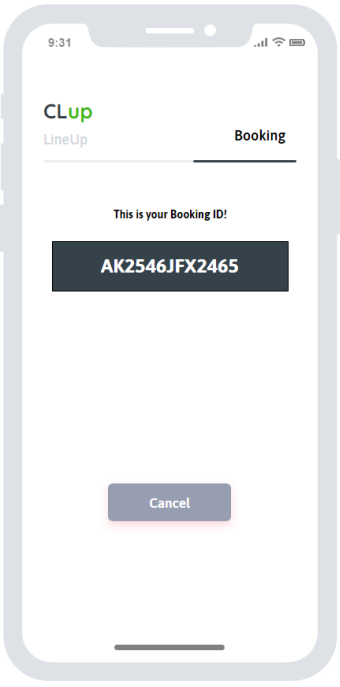


Figure 3.8: Show Booking ID

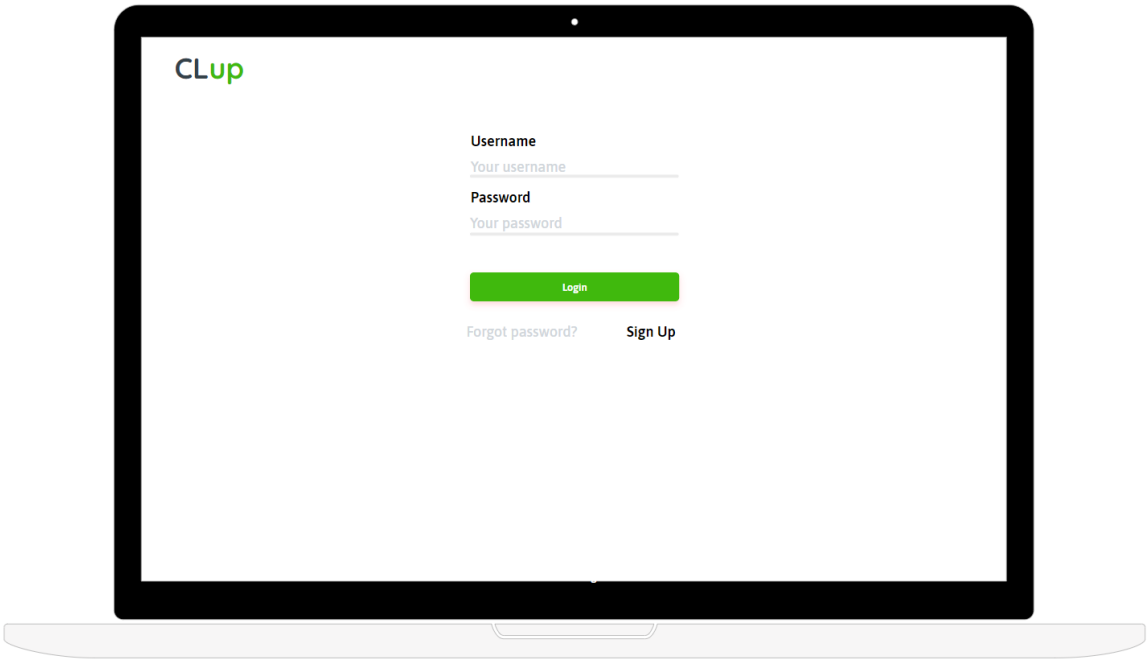


Figure 3.9: WebApp Login

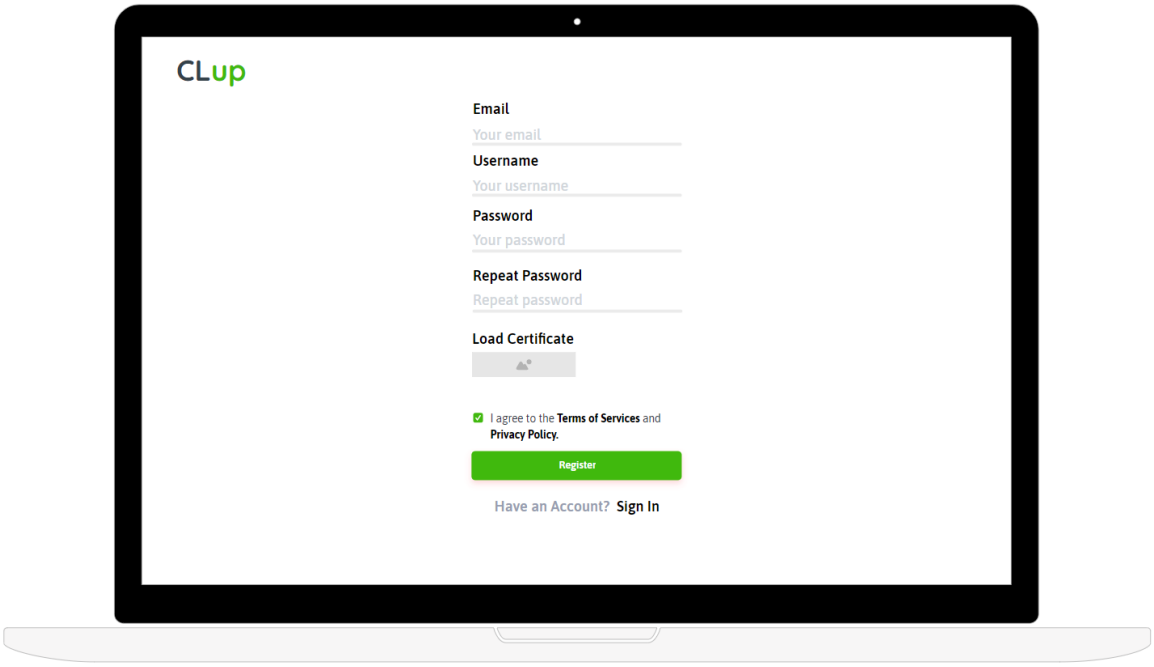


Figure 3.10: WebApp Registration

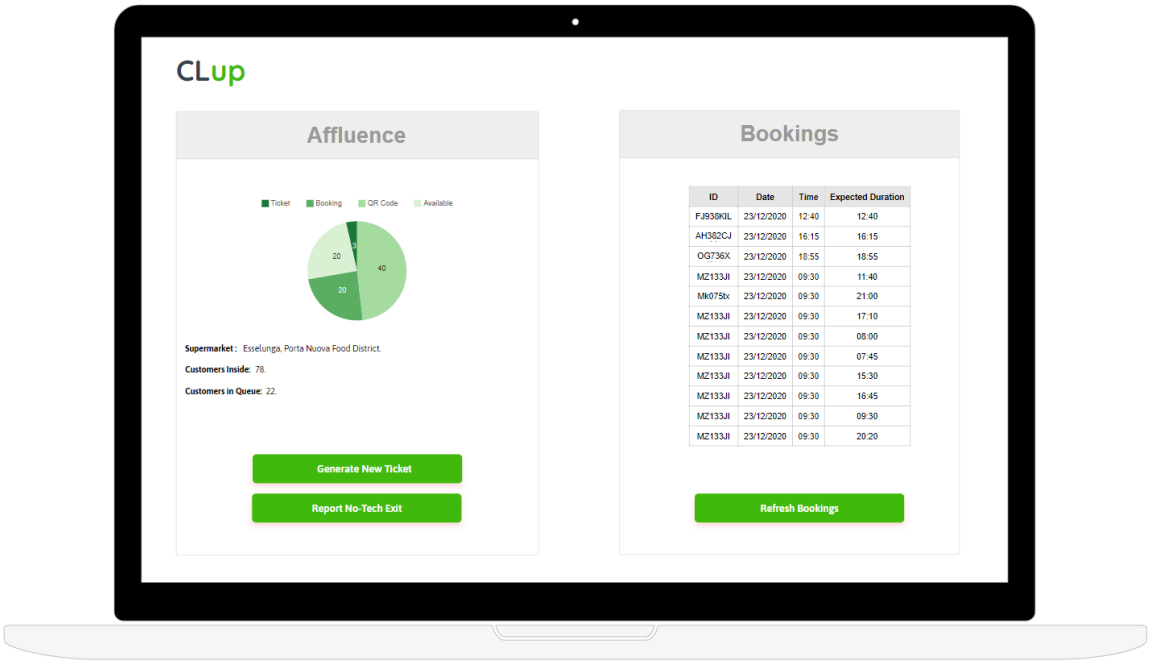


Figure 3.11: WebApp Homepage

Chapter 4

Requirements Traceability

This chapter presents the mapping between the requirements, defined in the RASD document in order to achieve the prefixed goals, and the components introduced in the Component Diagram. This task is important because it permits to check whether all requirements have been considered and to control that every component is used at least one time. Moreover, in this way it is also possible to ensure that a proper modularity and separation of concerns of the components have been reached during the Design phase.

- R_1 : The System retrieves the GPS position of the Customer.
  - MobileApp
  - GPSService
  - GoogleMapsService
- R_2 : The System sends the QR code for the entrance.
  - MobileApp
  - CustomerService
  - LineUpService
- R_3 : The Customer indicates with which mean of transport he intends to go to the supermarket.
  - MobileApp
  - CustomerService

  - LineUpService

- R_4 : The Customer indicates the supermarket in which he wants to go.
  - MobileApp
  - CustomerService
  - LineUpService
  - BookingService

- R_5 : The System shows the Customer his Waiting Time.
  - MobileApp
  - CustomerService
  - LineUpService
  - RealTimeqQueueManager

- R_6 : The Customer select date and time of his Booking.
  - MobileApp
  - CustomerService
  - BookingService

- R_7 : The Customer can input the expected duration of his visit.
  - MobileApp
  - CustomerService
  - BookingService

- R_8 : The System computes the Inferred Durations of the visits for Long-Term Customers who didn't input it.
  - CustomerService
  - BookingService
  - DatabaseAccess
  - Database

- R_9 : The Customer can input the exact list of items to buy or their categories.
  - MobileApp
  - CustomerService
  - BookingService
- R_{10} : The System computes the time needed for the Customer to reach the supermarket based on his currently GPS position and on his mean of transport.
  - MobileApp
  - GPSService
  - GoogleMapsService
- R_{11} : The System notifies the Customer when he have to start.
  - MobileApp
- R_{12} : Customers scan their QR code when they both enter or exit from the supermarket.
  - MobileApp
  - CustomerService
  - LineUpService
  - RealTimeQueueManager
- R_{13} : The System computes the Waiting Time considering the Real-Time Queue.
  - CustomerService
  - LineUpService
  - RealTimeQueueManager
  - DatabaseAccess
  - Database
- R_{14} : The System increases entrances considering the affluence of the departments.

  - CustomerService
  - BookingService
  - DatabaseAccess
  - Database

- R_{15} : Store Manager sends to the System a Ticket Request for No-Tech Customers.
  - WebApplication
  - StoreManagerService
  - TicketService

- R_{16} : The System generates the Ticket and sends it to the Store Manager.
  - WebApplication
  - StoreManagerService
  - TicketService
  - RealTimeQueueManager

- R_{17} : The System updates the Real-Time Queue.
  - StoreManagerService
  - TicketService
  - CustomerService
  - LineUpService
  - RealTimeQueueManager
  - DatabaseAccess
  - Database

Chapter 5

Implementation, Integration and Test Plan

This chapter presents the approaches that will be used during the following three crucial steps of the software lifecycle: in particular, in section 5.1 and 5.2 the Implementation Plan and the Integration Stategy are described, while in section 5.3 the Testing Plan is presented.

5.1 Implementation Plan

The first thing to do is to decide the order in which component will be implemented: the approach that has been chosen for this task, and also for the Integration and Testing ones, is the Bottom-Up one. This is mainly for two reasons: firstly it is by far the simplest one with respect to other incremental approaches such as Top-Down or Thread-based. This is an important advantage because it permits to reduce the probability of errors and typically guarantees more efficient tracking in case of bugs. The second reason is that this approach permits to perform testing and integration in parallel with the implementation in a quite easy way: in fact, after having implemented a component, Unit testing is perfomed directly by programmers and as the system grows through the Integration testing it is checked that the different modules interacts correclty. At the end of this incremental process, once the whole system is complete, and accurate System testing will ensure that every requirement has been satisfied.
The only drawback of this approach is that it is not possibile to release an "early-version" of the product in order to receive feedbacks, for instance about the UI or the

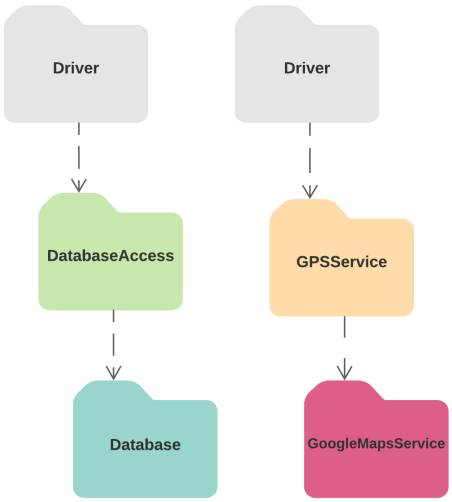
usability, directly from the actually users of the application. However, this issue is obviuosly compensated by a the higher execution speed that the Bottom-Up approach ensures.
Moreover, it is important to keep in mind that, when possible, parallelizing the work is crucial in order to minimize the Time to Market of the product. For this reason, no other constraints than those strictly needed and determined by the software design should be introduced, so as not to slow down the implementation, integration and testing phases.
In the Bottom-Up approach, to establish the order in which the implementation must be carried out, the dependencies between components are exploited: if a component rely on another one, necessarily the latter must be integrated before the former. Therefore, it is evident that this method suggests to start with the components which are completely independent from the other ones, or that, as in this case, only rely on external components such as the Database and the GoogleMapsService. For this reason, the first two components to be implemented and tested are the **DatabaseAccess** and the **GPSService**.
Then, the next component that should be implemented is the **RealTimeQueueManager**, because it only depends on the DatabaseAccess component.
In parallel with that, also the two components used for the login and registration purposes, namely the **AuthenticationService** and the **AccountManager**, could be implemented. In fact, as the RealTimeQueueManager, they only depends on the DatabaseAccess and so they pose no problems.
Going on, programmers could continue with the two main components of the entire system: the **CustomerService** and the **StoreManagerService**. But before, since these two are macro-components, their still unimplemented sub-components must be done. In particular, the **LineUpService**, the **BookingService**, the **TicketService** and the **AffluenceService** have to be carried out. These components are almost independent one from another and they only rely on the already present DatabaseAccess and RealTimeQueueManager. Finally, once also the two macro-components have been fully integrated and tested, the **MobileApplication** and the **WebApplication** could be realized. So, as expected, these two component, that are used in the interaction with Customers and Store Managers respectively and that constitute the Presentation level of the system, are implemented only at the end because they heavily rely on all the other components. In conclusion, to recap, the components of the

system will be implemented in the following order:

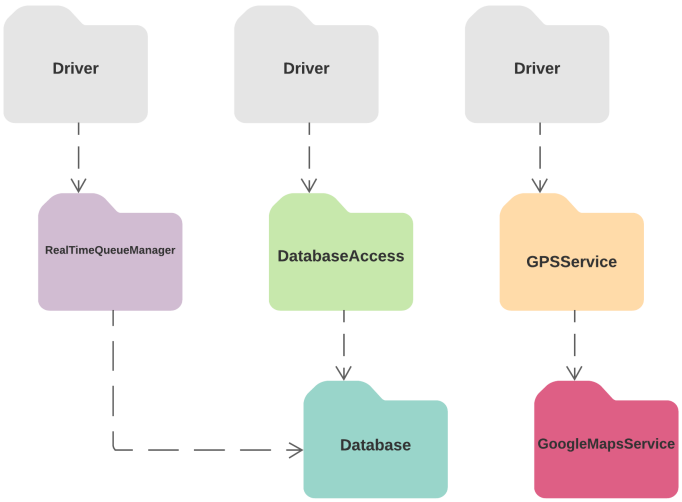
1. DatabaseAccess and GPSService
2. RealTimeQueueManager, AuthenticationService and AccountManager
3. LineUpService, BookingService, TicketService and AffluenceService
4. CustomerService and StoreManagerService
5. WebApplication and MobileApplication

5.2 Integration Strategy

In this section it is shown the order in which the components of the system may be integrated with each other in order to build the entire software. As already stated, also the integration stategy follows a Bottom-Up approach and also here, the dependency relations are exploited as a guide. For this reason, the first two components to be integrated and tested are the DatabaseAccess and the GPSService.
Notice that every time that it is needed to test a component which is used by other ones that have not been integrated yet, Drivers are exploited. These auxiliary "components", the task of which is only to simulate the still missing modules, play a fundamental role because without them it would not be possible to perform Unit tests as early.
Another aspect to be underlined is that the Database and the GoogleMapsService, even though they are external components and so they are directly exploited without having to implement them, are however integrated as soon as possible in order to test whether their interfaces can communicate properly respectively with DatabaseAccess and with GPSService.

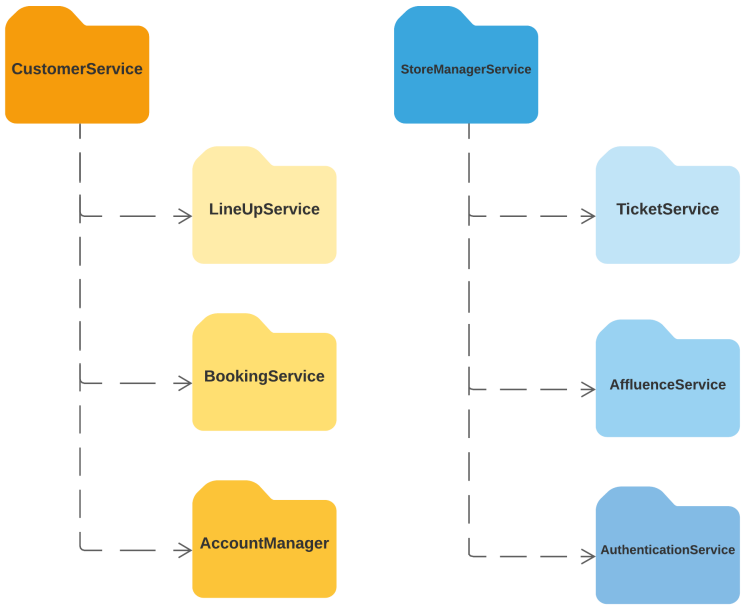


After that, the RealTimeQueueManager can be integrated, since this component only depends on DatabaseAccess. As before, a Driver is introduced in order to simulate its functionalities and perform Unit testing.

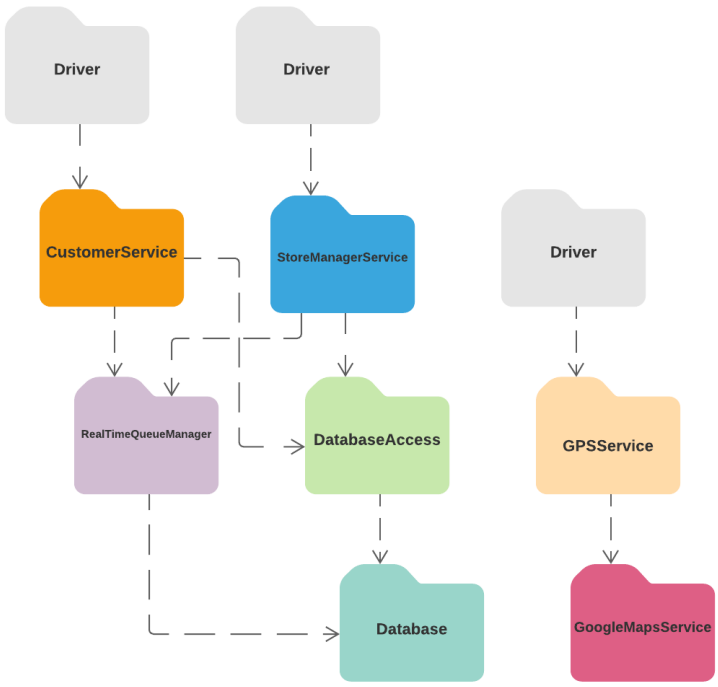


Then, it is possible to start integrating those components that represent the Business Logic of the system. In particular, all the subcomponents of the CustomerService and StoreManagerService can be integrated. For the sake of representation, first we show only how sub-components are put together into macro-components, and then how the macro-components interact with the rest of the system. However, before doing that obviously every single sub-component is Unit tested. So, as just stated,

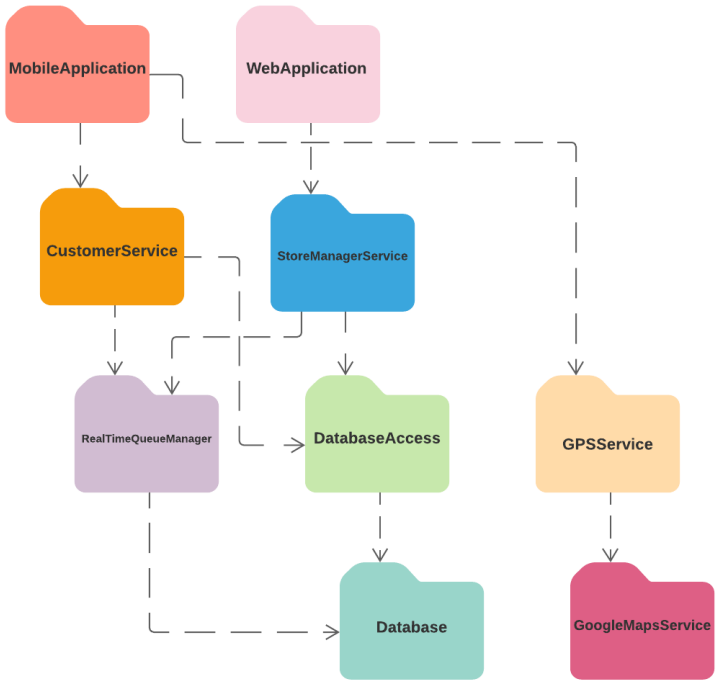
first the two macro-components are built.



Now that the two macro-components have been built through their sub-components, they can be integrated with the other ones.



Finally, the last two components to be integrated and tested are the WebApplication and the MobileApplication.



5.3 Testing Plan

In this section it is briefly described the idea to follow to perform the Testing of the whole artifacts that have been produced during the entire software lifecycle. Of course, a more precise and deep analysis of this fundamental phase will be accurately described in the Implementation and Testing Document (ITD), in which test cases and their outcomes will be provided as well. In that document, in particular different kind of testing will be performed:

- Unit Testing: every single component is tested before being integrated with the rest of the system.
- Integration Testing: every module, composed by different components is tested.
- System Testing: the whole system is tested as a "black-box" to check whether all the requirements are satisfied and so to complete the Verification process.

- Performance Testing: the system is tested with the foreseen workload in order to identifies possible bottlenecks with respect to response time, utilization and thoughput.
- Load Testing: the system is tested with the maximum load, which is reached incrementally by increasing it until a certain threshold. This testing exposes bugs such as memory leaks, mismanagement of memory, buffer overflows and identifies upper limits of components.

The general idea, as it has been already stated in this chapter, is to test the artifacts following a Bottom-Up approach, the same used also for the implementation and integration phases. The main advantage of this approach is that it is straightforward to be carried out, and typically for this reason it guarantees a satisfying result. In addition, this method is also very useful in order to optimize the task and parallelize the work between different teams. Moreover, it is vital to test not only the software modules produced by programmers but also every other artifacts: for instance, every document, starting from the RASD untill the ITD, must be carefully revised and keep updated every time that a change is performed.

Chapter 6

Effort Spent

Ludovico Righi

Task	Hours
Chapter 1	/
Chapter 2	15
Chapter 3	1
Chapter 4	2
Chapter 5	11

Enrico Gherardi

Task	Hours
Chapter 1	2
Chapter 2	18
Chapter 3	/
Chapter 4	2
Chapter 5	7

Chapter 7

References

This document has been written referencing to the folllowing documents:

- IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications.
- Specification document: "R&DD Assignment A.Y. 2020/2021".
- Lecture slides of professor M.Rossi and E.di Nitto of Politecnico di Milano.

The tools that have been exploited are the following:

- LucidChart: to build the diagrams.
- LaTeX: to write the pdf file.
- GitHub: for versions controller.