Computer Science and Engineering
Software Engineering 2 Project - Prof. Elisabetta Di Nitto

CLup – Customers Line-up

Design Document

Marco Di Gennaro (10596841)
Luca Danelutti (10604455)

January 10, 2021

Contents

1. Introduction

1.1 Purpose

This document represents the Design Document (DD). The purpose of the DD is to provide overall guidance to the architecture of the software product discussed in the RASD.

It contains the description of the architecture chosen to design the CLup System. Furthermore, it describes how the UI has to look like, and the map from the requirements defined in the RASD and the architectural components defined in this document. It also includes a set of design characteristics required for the implementations by introducing constraints and quality attributes, and it gives a presentation of the implementation, integration, and testing plan.

DD is addressed to the software development team who will have to implement the described system and it has the purpose to guide them through the development process.

1.2 Scope

The System aims to provide a solution to reduce overcrowding both inside and outside grocery stores.

The application would work as a digital counterpart to the common situation where people who are in line for a service retrieve a number that gives their position in the queue. The system should provide both the possibility to line up remotely (through an application on a mobile phone) and at the grocery store for those customers who do not have access to the required technology (**Line up functionality**). Each customer that lined up should receive a number. Users should wait until his/her number is being called (or close to being called) to approach the store. This should reduce overcrowdings outside supermarkets. Users can also scan a QR code when entering the grocery store, enabling the store manager to monitor entrances.

In addition to lining up directly, an advanced function is offered. Customers can also book a visit to the supermarket, similarly to booking a slot for visiting a museum. The system should be able to schedule customer visits correctly given that each visit will last differently from the others. CLup can ask the customer details about his/her visit or it can compute an estimated duration from previous visits of the same user (**Book a visit functionality**).

Ultimately, the system will have to be easy-to-use given that everyone needs to do grocery shopping and the more users will use the system remotely the more CLup will be effective.

More detailed information can be found on the RASD.

1.3 Definitions, Acronyms, Abbreviation

1.3.1 Definitions

QR code	A QR code is a type of matrix barcode, that is a machine-readable optical label that contains information about the item to which it is attached
Visit	In this document by "visit" we mean the visit to a supermarket in a specific time slot
Line up	In this document by "line up" we mean the line up to enter in a supermarket

1.3.2 Acronyms

RASD	Requirement Analysis and Specification Document
DD	Design Document
UI	User Interface
QR	Quick Response
API	Application Programming Interface
MVC	Model-View-Controller

1.3.3 Abbreviations

Gn	Goal number n
Rn	Requirement number n

1.4 Revision History

Date	**Description**
06-01-21	First Version
10-01-21	Final Version

1.5 Reference Documents

- Specification Document : "R&DD Assignment AY 2020-2021"
- Lecture slides

1.6 Document Structure

This document is composed of six chapters :

- **Chapter 1: Introduction.** This chapter describes the scope and the purpose of this DD. This chapter also includes the structure of the document and a set of definitions, acronyms and abbreviation used
- **Chapter 2: Architectural Design.** This chapter contains the architectural design choice, it includes an overview of the designed architecture, all the components, the interfaces, and the technologies used for the design of the application. It also includes the main functions of the interfaces and the process in which they are utilized (Runtime view and component interfaces). Finally, there is an explanation of the design pattern chosen and recommended to develop the system
- **Chapter 3: User Interface Design.** This chapter shows how the UI has to look like. It completes and deepens to the User Interfaces (3.1.1) subsection in the RASD
- **Chapter 4: Requirements Traceability.** This chapter explains how the requirements defined in the RASD map to the design elements defined in this document
- **Chapter 5: Implementation, Integration and Test Plan.** Here is the order in which is planned to implement the subcomponents of the system and the order in which is planned to integrate such subcomponents and test the integration
- **Chapter 6: Effort Spent.** This chapter includes information about the number of hours each group member has worked for this document
- **Chapter 7: References.**

2. Architectural Design

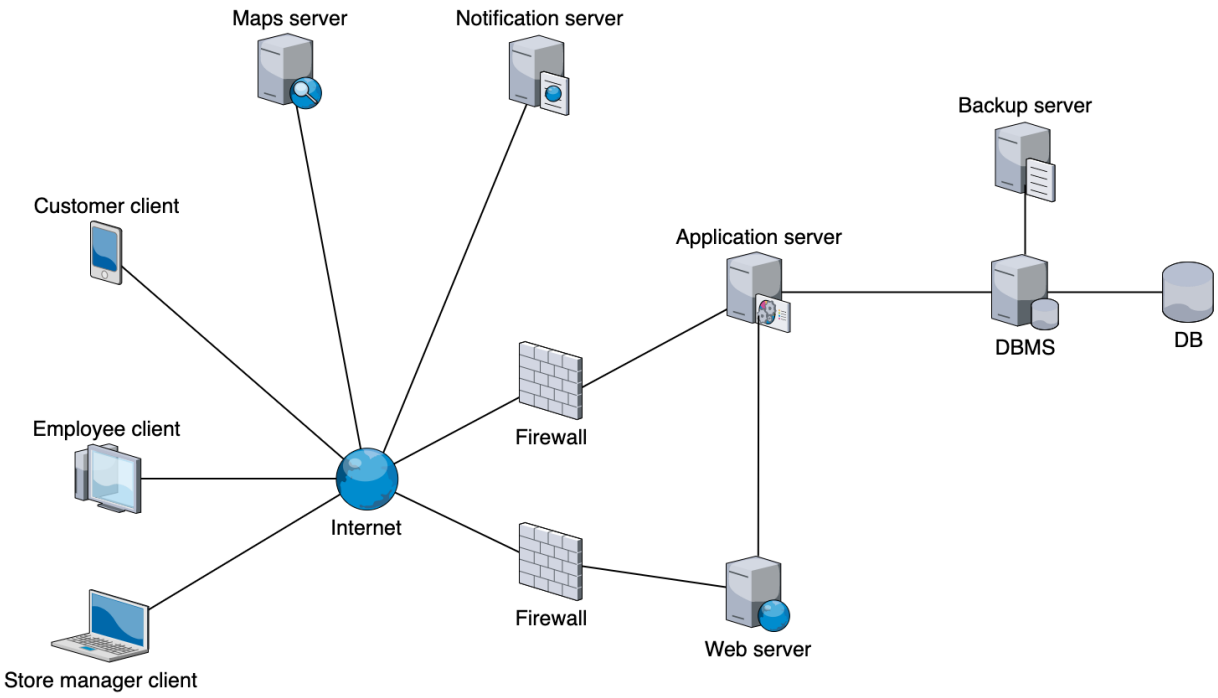
2.1 Overview

The application to be developed is a distributed one with three logic software layers: Presentation, Application, and Data. The first one manages the graphical interaction with the system, the second one handles the business logic of the application and the third one manages the storage of data involved. Thus we have a so-called three-tier architecture. This architecture is a good choice to provide characteristics like scalability and maintainability mentioned in the RASD document. Each tier can be deployed in a different hardware machine (or group of machines to better achieve reliability, availability, and scalability).

The mobile application communicates with the backend infrastructure directly with the application server, thus the Presentation layer is entirely contained in the app. The web application instead communicates with the Web Server, so the Presentation layer is distributed between the client and the server.

Communication with third parties happens at the application level with synchronous messages over the Internet. To guarantee an appropriate traffic control needed to fulfill the security requirements firewalls are installed between the web server and the Internet and between the application server and the Internet. In the Data layer, a backup server keeps copies of the database to eventually recover from faults without losses. More about the server-side deployment and replication information in the deployment section.

The following schema represents a high-level logic representation of the system to be developed and deployed.



2.2 Component view

To make it easier to read component diagrams, it was decided to present 4 diagrams. The first represents a high-level view that reports client-side and server-side components. The other 3 diagrams each represent one of the 3 main subsystems presented in the first diagram.

2.2.1 High level component diagram

The following diagram shows the main components of the system and the interfaces through which they interact. There are 2 sides:

- The **client-side** is composed of 2 components: Mobile Application for the customers and Web Application for employees and the store manager
- The **server-side** is the significant part to analyze more thoroughly, here are 3 main subsystems:
  - The *Customer Mobile Services* provides the interfaces to line up, book a visit and manage historical information and account information
  - The *Employee Web Services* supports the operations that the employee can perform on the web application. This component provides the interfaces to line up (for an external client), to confirm an entrance or an exit from the store
  - The *Store Manager Web Services* supports the operations that the store manager can perform on the web application. This component provides the interfaces to manage store and employees information and to visualizes data about entrances

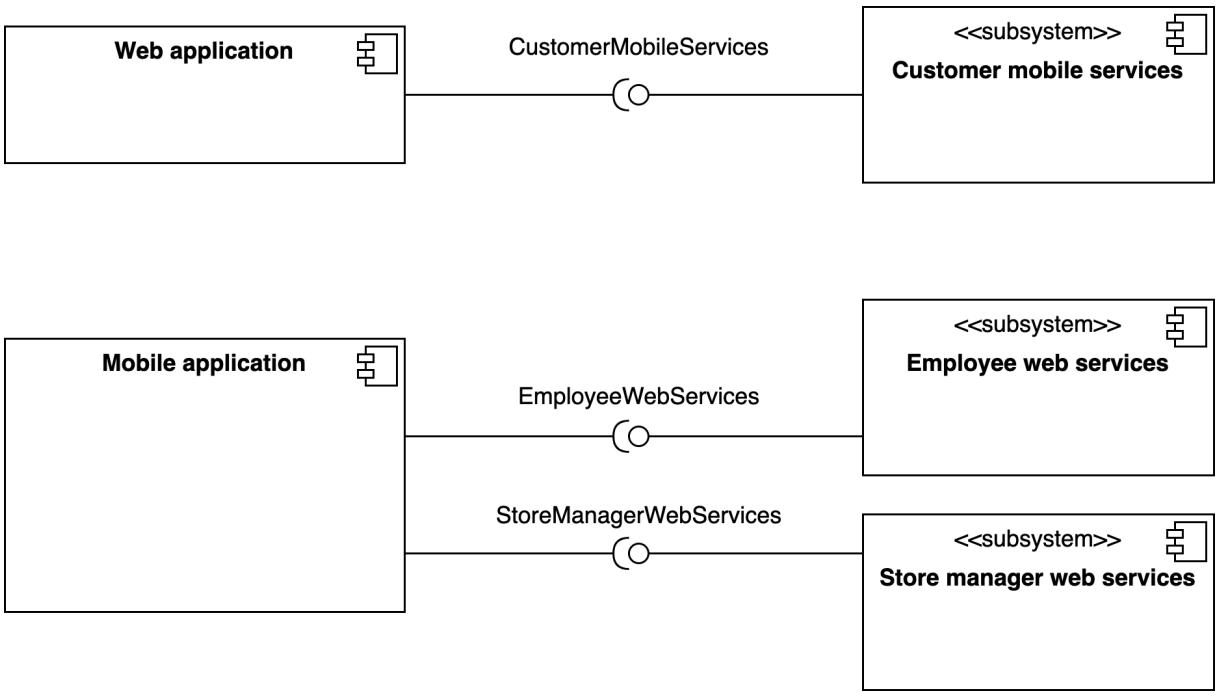


Figure 2.1: High level component diagram

2.2.2 Customer mobile services diagram

The following diagram shows how the subsystem **Customer Mobile Services** is composed and how its modules communicate with external modules. This subsystem contains 3 modules: a *Visit Reservation Module*, a *Line up Reservation Module* and an *Account Manager*. This modules provides to Mobile Application the following interfaces: *VisitHistory*, *LineUpHistory*, *ManageVisit*, *ManageLineUp*, *ProfileManagement*, *AccessManager*. The components of the Customer Mobile Services subsystem need to communicate with the DBMS, Queue Manager and Visit Scheduler and indirectly with Maps API.

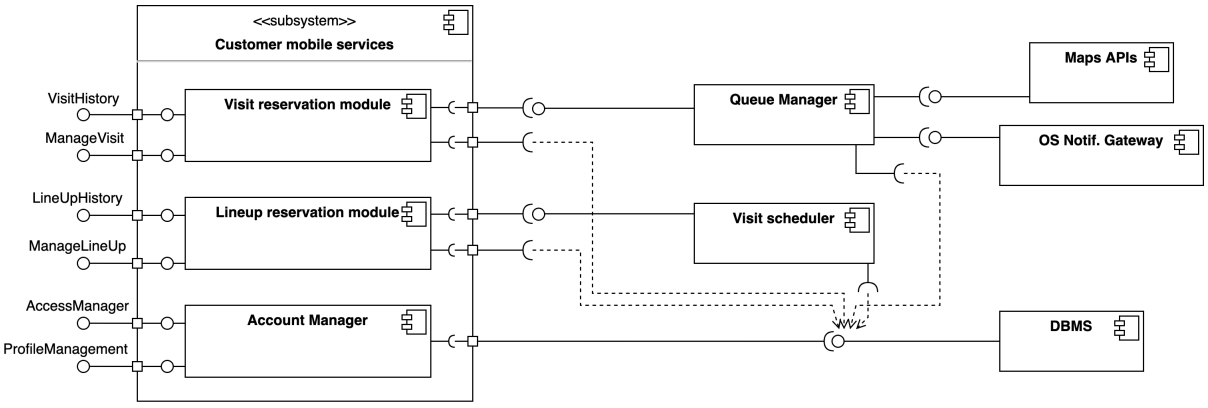


Figure 2.2: Customer Mobile Services component diagram

2.2.3 Employee web services diagram

The following diagram shows how the subsystem **Employee Web Services** is composed and how its modules communicate with external modules. This subsystem contains 3 modules: a *Line up Reservation Module*, a *Entrances and Exits Module* and an *Account Manager*. This modules provides to Web Application the following interfaces: *LineUp*, *ManageEntrances*, *ManageExits*, *AccessManager*. The components of the Employee Web Services subsystem need to communicate with the DBMS and the Queue Manager and indirectly with OS Notification Gateway.

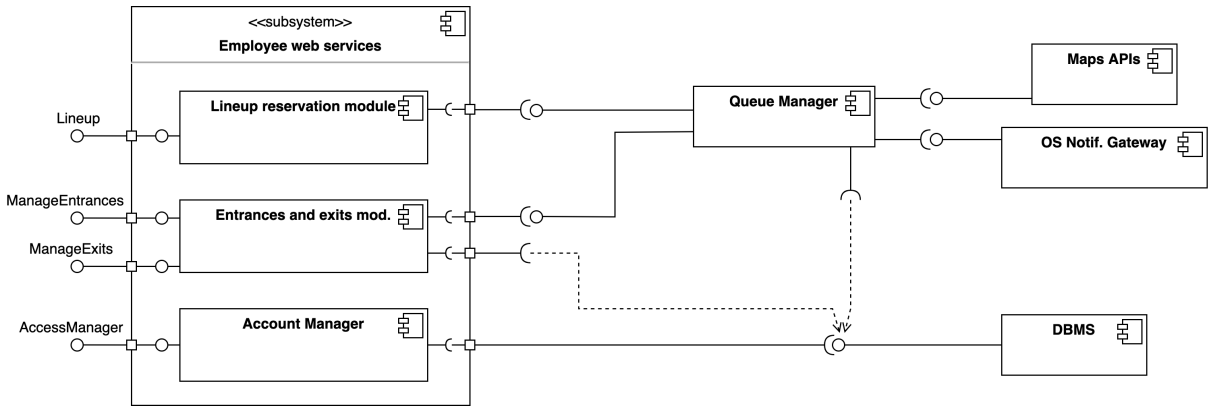


Figure 2.3: Employee Web Services component diagram

2.2.4 Store Manager web services diagram

The following diagram shows how the subsystem **Store Manager Web Services** is composed and how its modules communicate with external modules. This subsystem contains 3 modules: a *Statistics and Data Gateway*, a *Store Info and Employees Manager* and an *Account Manager*. This modules provides to Web Application the following interfaces: *Retrieve Statistics*, *StoreManagement*, *EmployeesManagement*, *ProfileManagement*, *AccessManager*. The components of the Store Manager Web Services subsystem need to communicate with the DBMS.

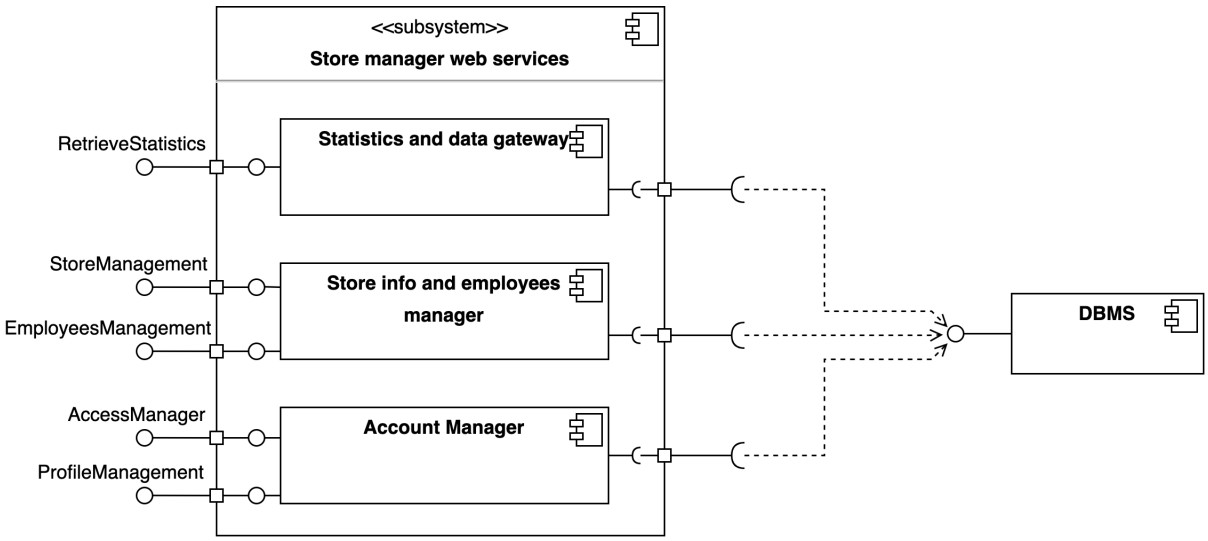


Figure 2.4: Store Manager Web Services component diagram

2.3 Deployment view

The system architecture is divided into 5 tiers and it's implemented using the Java Enterprise Edition framework.

- The first tier is the client tier: it contains the mobile application running on customers' devices and the web browser used by the store employees and the store manager to access the web application of CLup. The former communicates with the JEE server via a REST interface. The latter accesses the webserver and retrieves web pages.
- The second tier is the web tier: it is composed of the webserver implemented with the Apache HTTP platform. It mainly serves the static content and it is connected with the Tomcat script server to load the dynamic data.
- The third tier contains the script engine server. Tomcat has been chosen as the platform to generate dynamic content, via Servlet or JSP, requested by the Apache HTTP server.
- The fourth tier is the application logic tier: it is composed of the WildFly application server platform which handles the Enterprise Java Beans to which the Tomcat server connects.
- On the data tier (tier 5) there is the Database server. The connection with tier 4 is implemented with a JDBC connector.

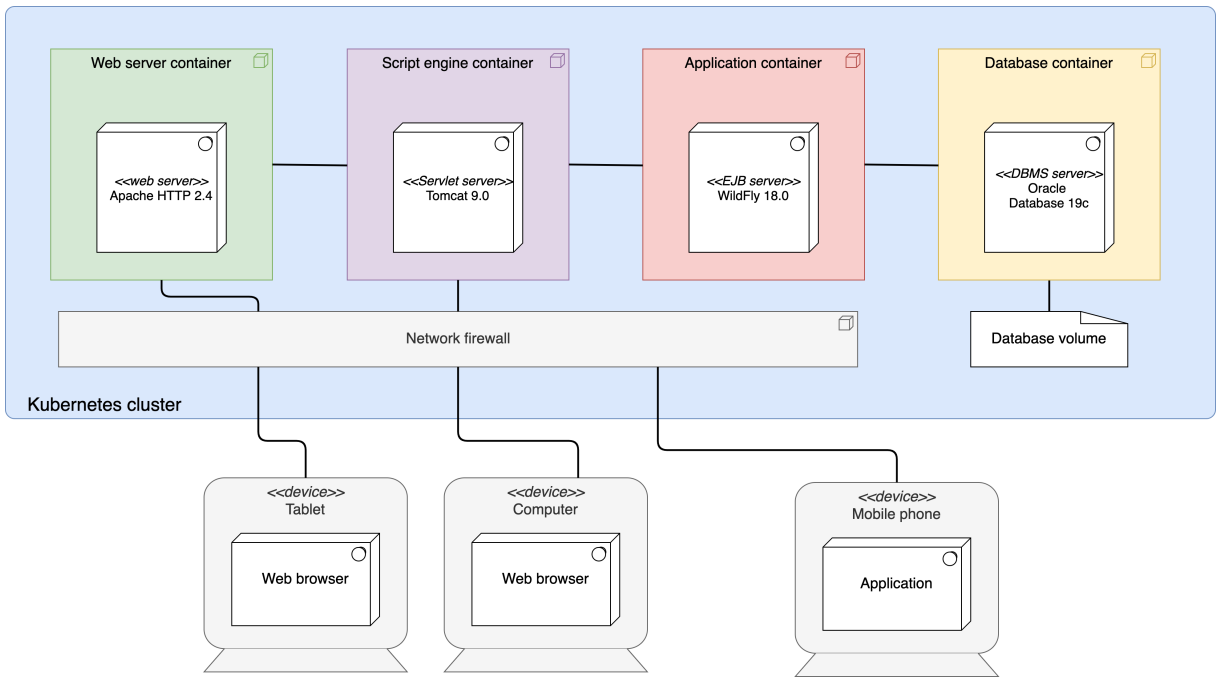


Figure 2.5: Deployment diagram

Recommended deployment:

All the tiers, except the first one, must be deployed as containers in a Kubernetes cluster. Each of the four tiers may be containerized and then deployed in a cluster. This process would produce many advantages such as:

- Easy replication management: it as easy as to change a number in a configuration file to scale up or down one of the tiers (except for the database, which is a little bit more involved).
- Portability: the application is extremely portable between different physical or cloud deployments, for example, allowing to switch between geographically different datacenters in a couple of minutes.
- Easy update management: update release is only a matter of shutting down some containers and bringing up new ones from updated images. Moreover, different versions of the system can coexist at the same time on the same cluster.

The Kubernetes cluster may be created and managed by the CLup deployment team. A better option may be to directly deploy the containers to Amazon EKS, Google Kubernetes Engine, or similar cloud services. In the latter case, tier 5 may use a service like Amazon EFS or Google Firestore to store the database, allowing to make copies of the database and protect against hard failures with an automatic backup solution offered by those cloud providers.

Recommended implementation:

- **Client tier:** the customer mobile application may be implemented using a cross-platform development framework like Flutter. Flutter allows writing the application code once and then to easily compile the source code for both Android and iOS systems. This would be a big advantage in terms of reducing development cost and time and of obtaining code maintainability.

- **Web tier:** the web pages of the web application may be implemented with HTML 5.0, CSS, and JavaScript.
- **Script engine tier:** the dynamic content may be generated using Java Servlets
- **Application logic tier:** the EJB application server may use stateless Java Beans connected using JPA with the Database server. This would allow having the client state completely stored on the DBMS, thus allowing to easily scale up or down the business instances.
- **Data tier:** the database may be implemented with MySQL Server Enterprise Edition.

2.4 Runtime view

In the following sequence diagrams, we are going to explain and represent the interactions that happen between the main components of CLup. This is still a high-level description of the actual interactions that will be developed. Thus function names, results, errors, and other details will be modified or added during the development process.

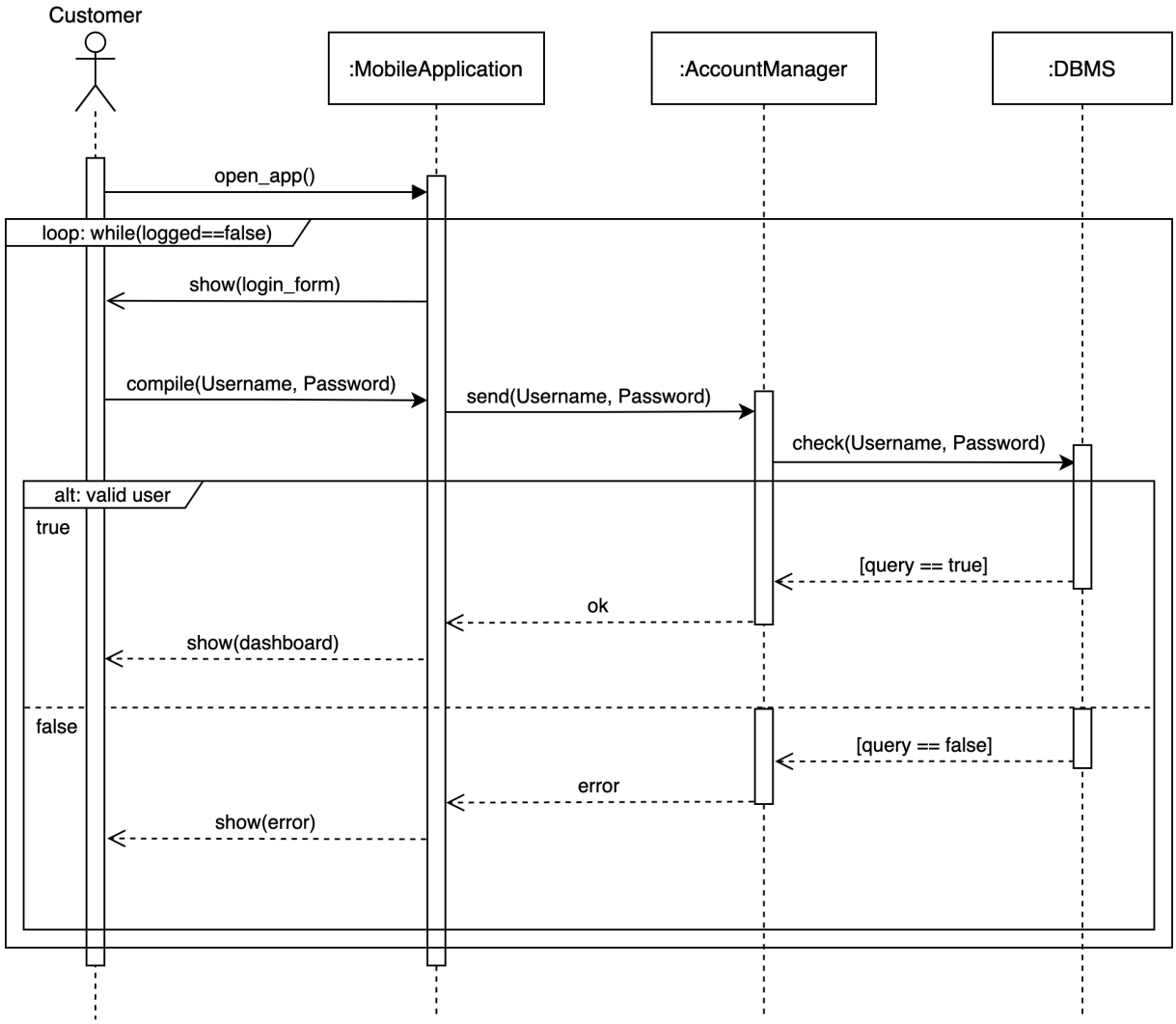


Figure 2.6: Customer Login

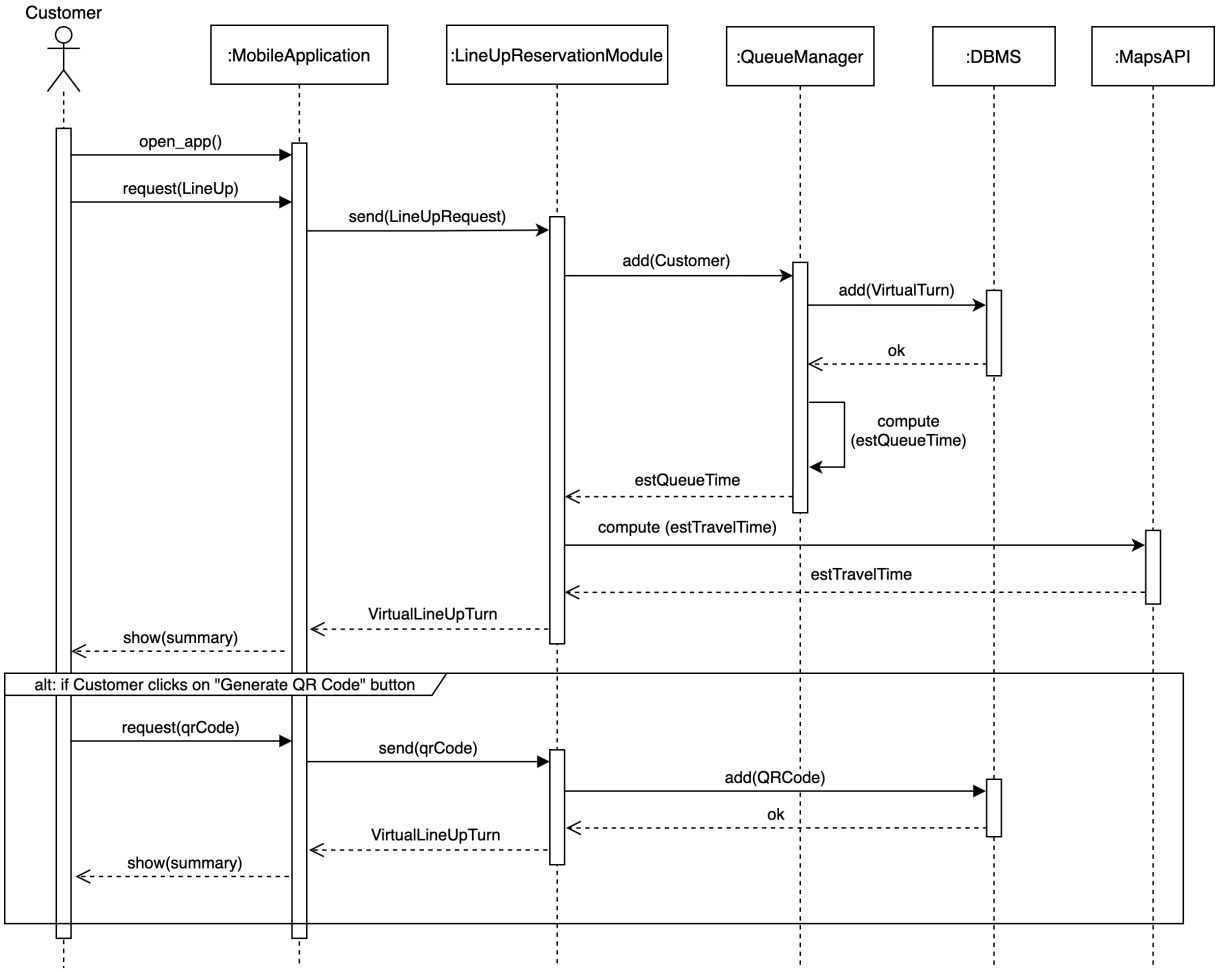


Figure 2.7: Lineup via application

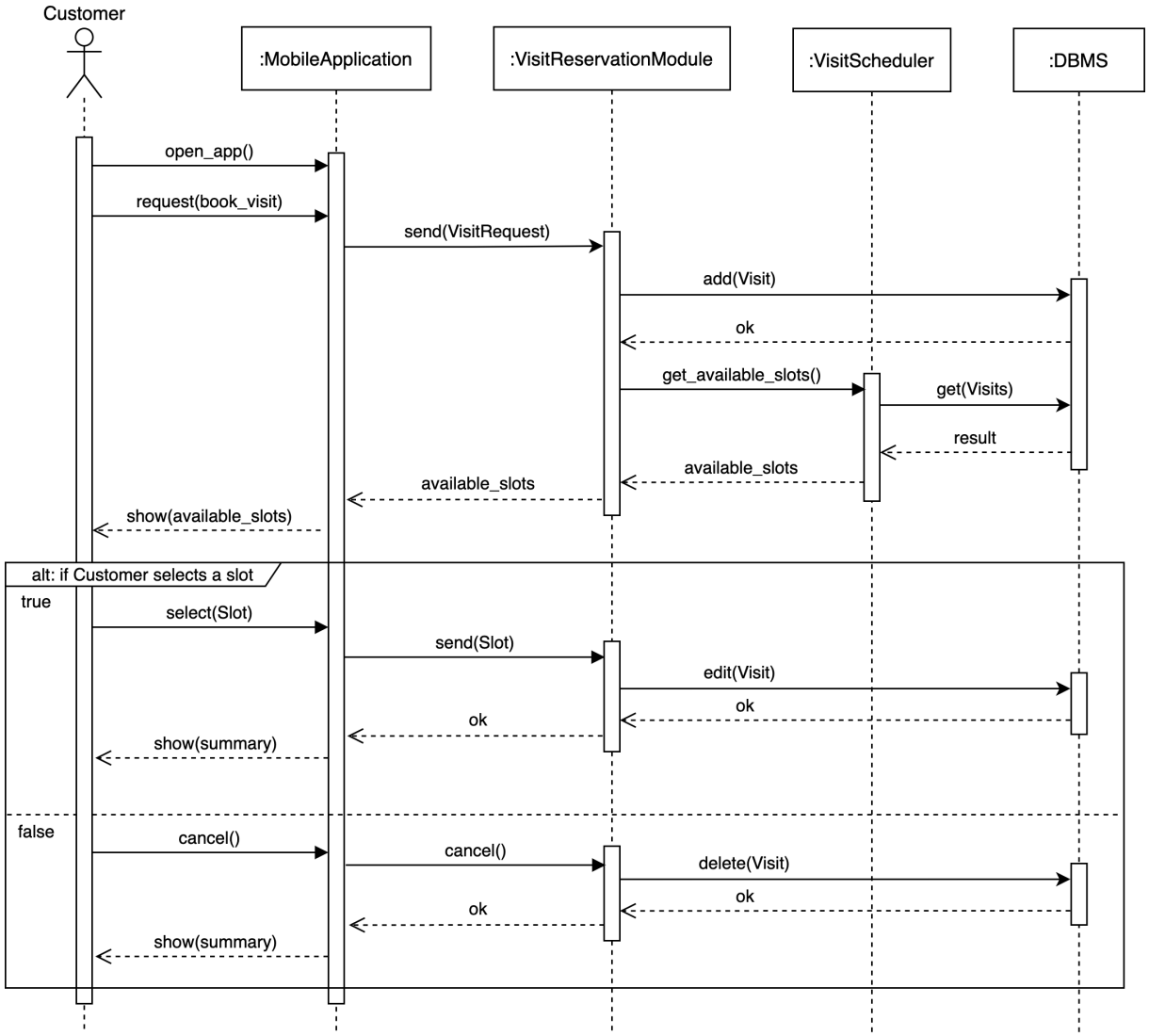


Figure 2.8: Book a visit

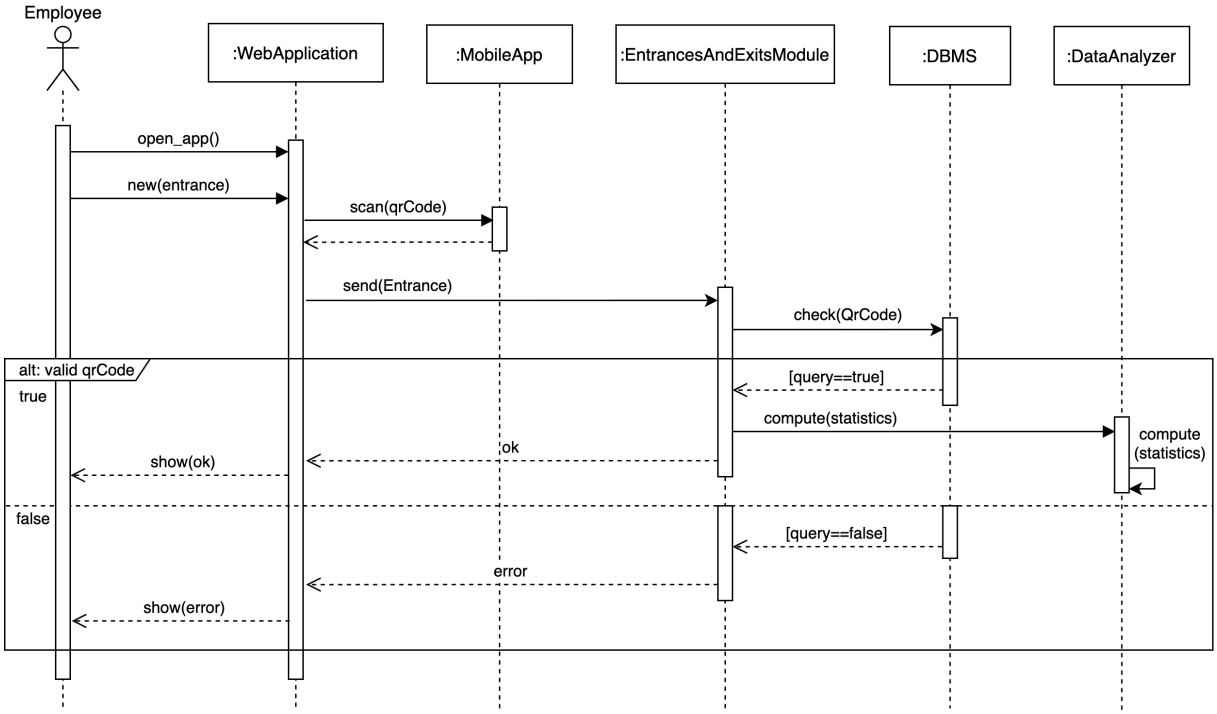


Figure 2.9: Entrance with a QR Code

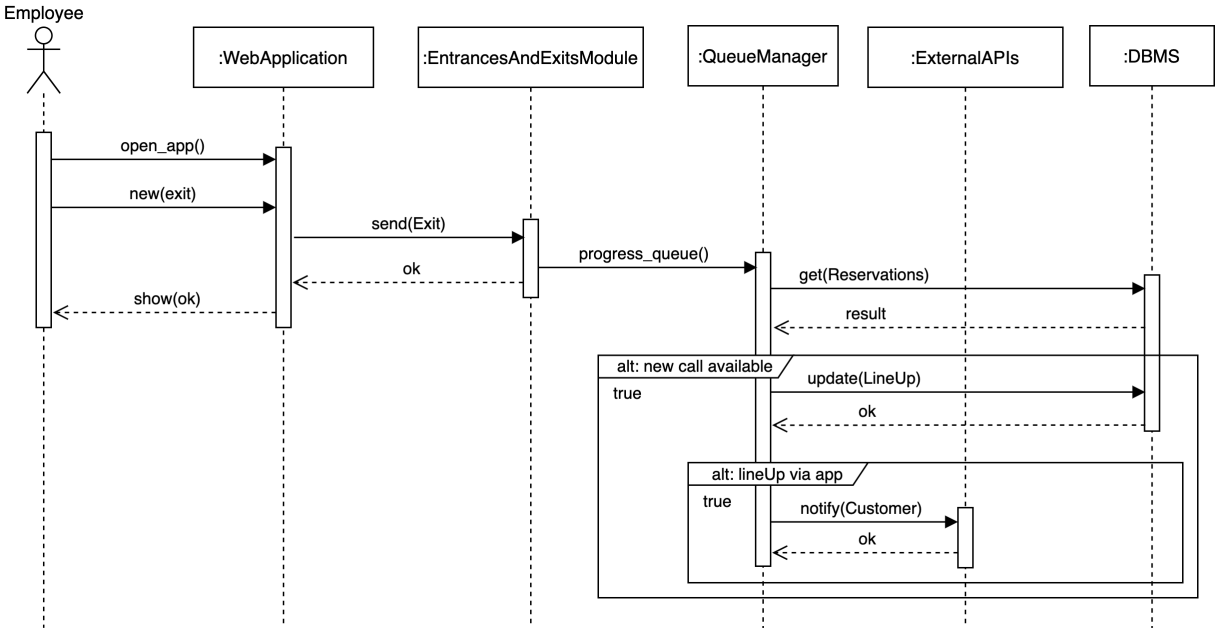


Figure 2.10: Exit report

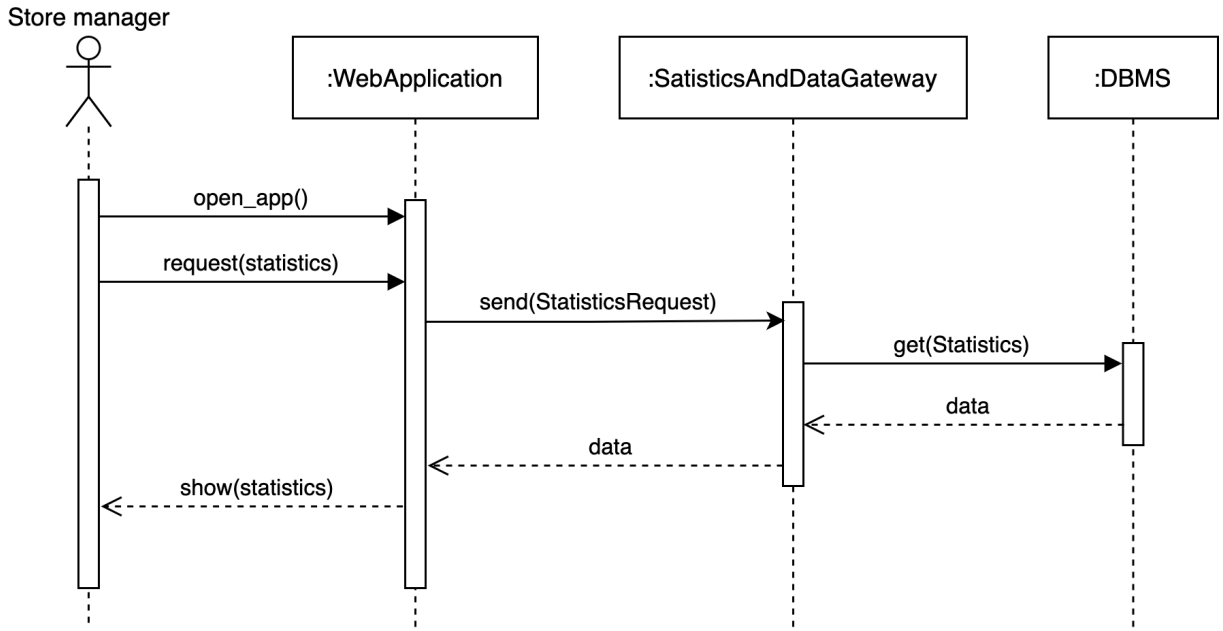


Figure 2.11: Statistics and charts visualization

2.5 Component interfaces

In this section the interfaces of the components are analyzed, defining the methods of accessing them. To ensure greater readability, below are three diagrams divided into three diagrams for each subsystem of the server-side of our system.

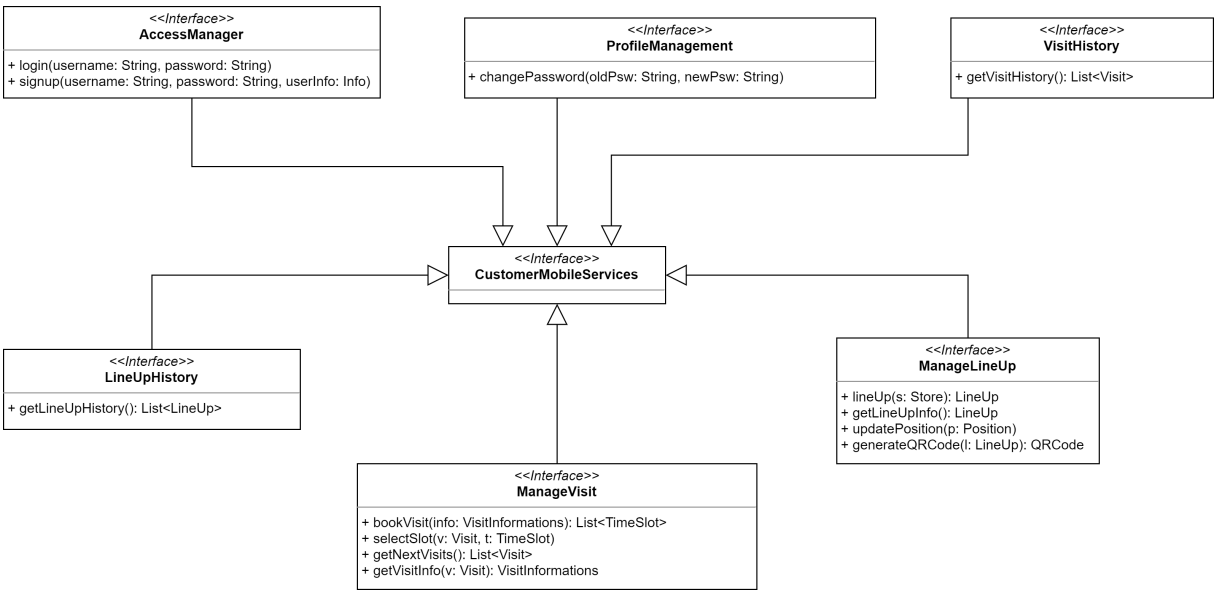


Figure 2.12: Customer Mobile Services interfaces

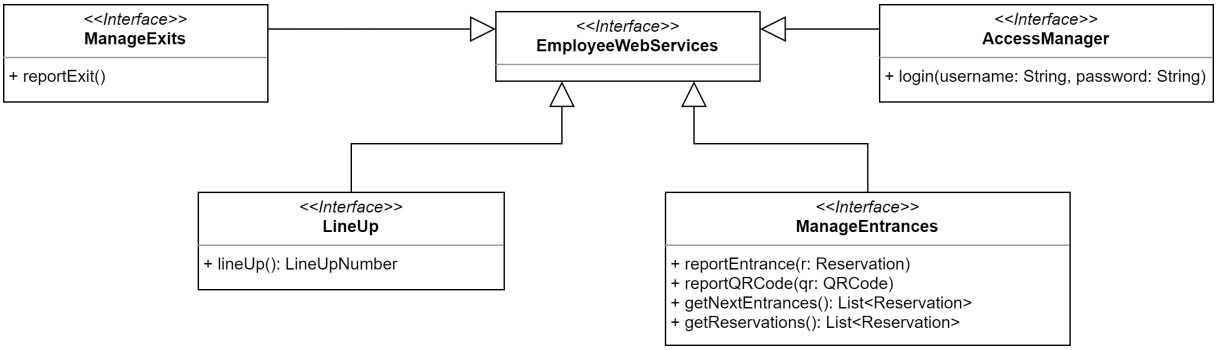


Figure 2.13: Employee Web Services interfaces

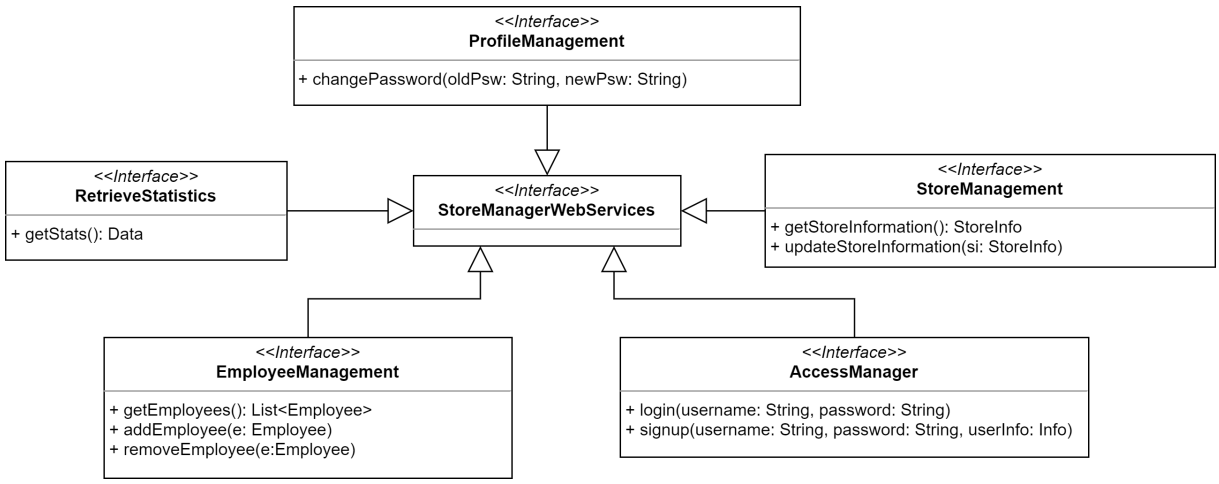


Figure 2.14: Store Manager Web Services interfaces

2.6 Selected architectural styles and patterns

2.6.1 Architectural styles

In the overview section (2.1) there is a description of how a **three-tier client-server** architecture has been used to design the system. This choice favors, first of all, a greater decoupling of the system, increasing reusability, scalability and flexibility.

Regarding the transport protocol adopted, our choice is **HTTPS**. It's the secure version of HTTP, which is the primary protocol used to send data between the web browser and a website (like the web app used by Employees and the Store Manager). HTTPS is encrypted to increase the security of data transfer. This is particularly important when users transmit sensitive data, such as by logging into their account.

For the communication between the application and the system, our choice is a **REST** architecture. The goal was to reduce the coupling between client and server components. It fits very well for the scope because the goal is to update the server-side regularly, without touching the client software. This choice introduces the following constraints:

- Layered Architecture
- Client-Server
- Stateless system
- Cacheable

All these constraints are respected by the choices previously made.

2.6.2 Patterns

Selected design patterns:

- **Facade pattern** simplifies a complicated system by providing a single interface to a set of interfaces within a subsystem. It is used for some components designed in the component diagram in section 2.2.

Recommended architectural patterns for implementation:

- **Model-View-Controller pattern** is an architectural pattern that divides a given software application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted by the user. MVC is recommended because it is the best design pattern to structure our three-tier client-server architecture presented in the previous sections and it is one of the most common and effective ways to avoid a dangerous level of coupling between the various parts of the whole system.

Recommended design patterns for implementation:

- **Observer pattern** allows you to notify one or more objects about changes in the state of other objects within the system. It is recommended because it is essential to the previously recommended MVC model and is also useful in other CLup system applications.
- **Visitor pattern** is a way of separating an algorithm from an object structure on which it operates. A practical result of this separation is the ability to add new operations to existing object structures without modifying the structures. It decoupling the operations from the object structure. This pattern is useful for the MVC application too.
- **Factory pattern** is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. It is particularly useful if applied in combination with the MVC pattern.

2.7 Other design decisions

Maps APIs: to calculate the estimated time to reach the store, a public maps service is necessary. Google Maps APIs are a good choice to do that. In particular, the Routes service will be used to estimate the travel time from the customer to the grocery shop.

Notifications APIs: to deliver notifications to customers about their position in the queue Firebase Cloud Messaging may be used. Firebase is easy to integrate with the Flutter framework and provides an easy set-up to send push notifications to both iOS and Android clients.

3. User Interface Design

In addition to the views provided in the RASD document, in the next few pages, we are going to present additional mockups which will give a complete overview of how the application and the web application will look like.

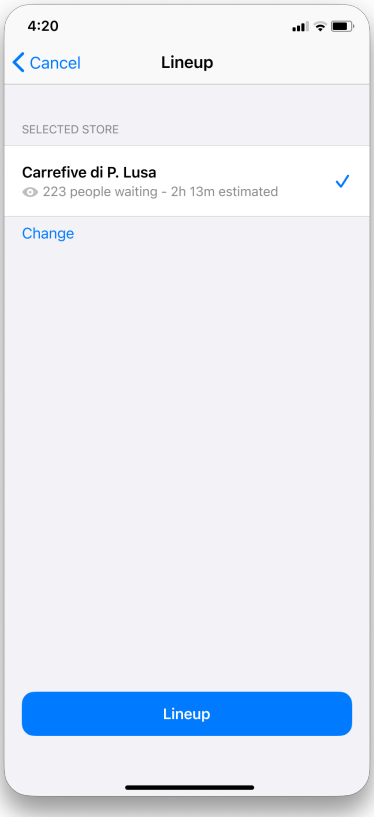


Figure 3.1: Lineup page

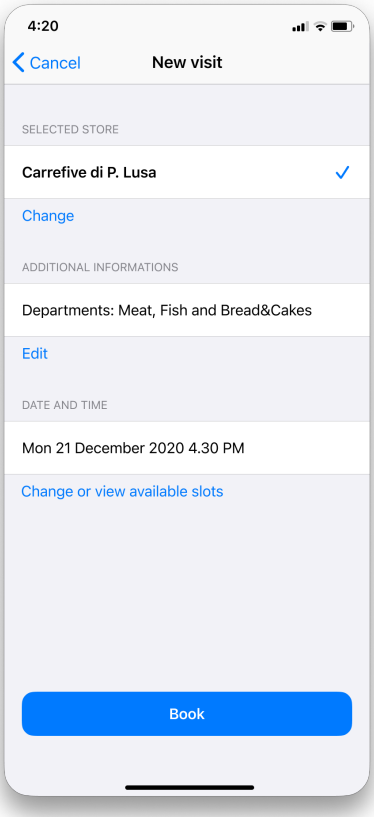


Figure 3.2: Book a visit

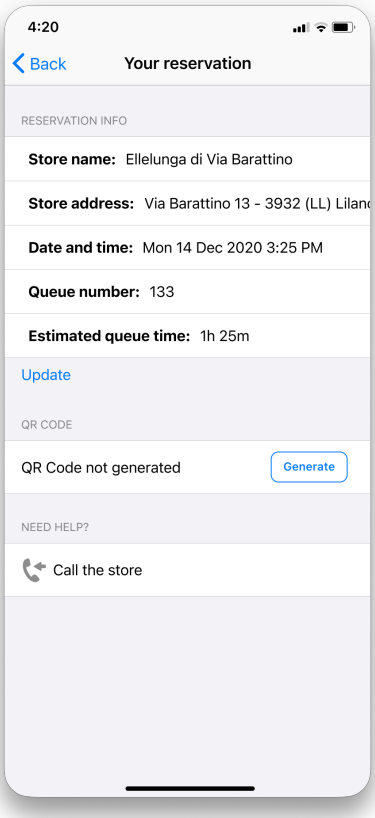


Figure 3.3: Reservation details

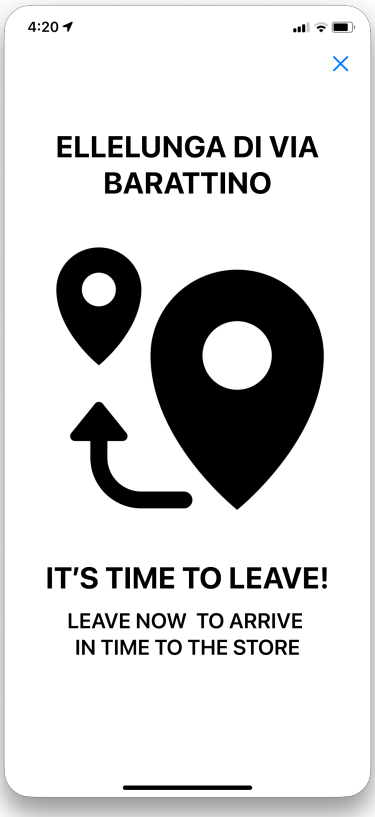


Figure 3.4: Notification

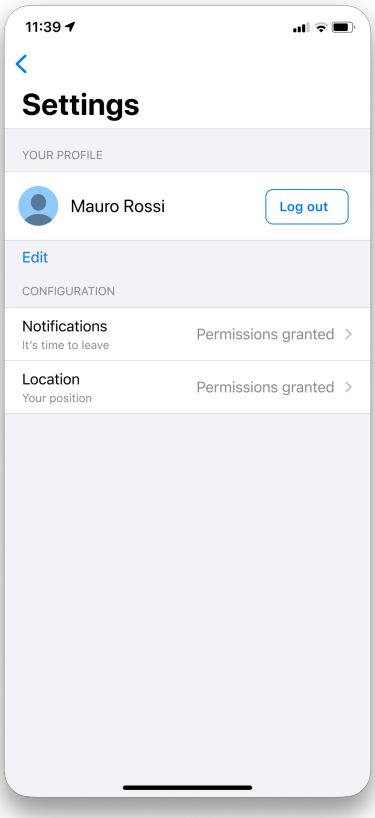


Figure 3.5: Settings

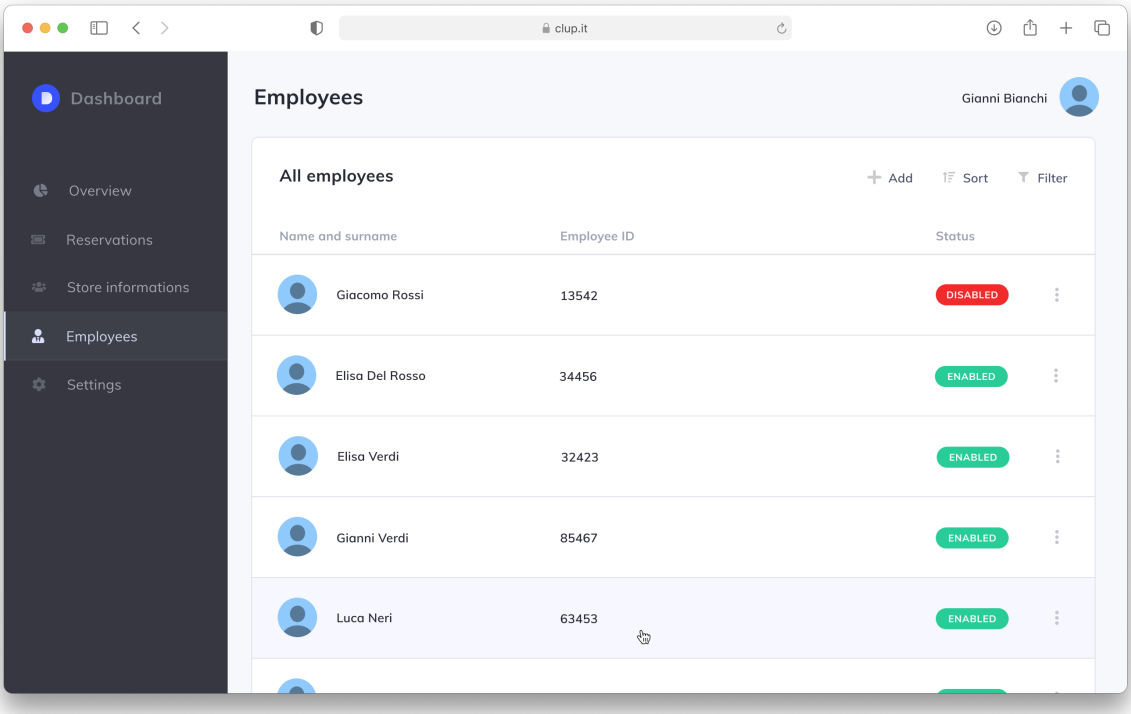


Figure 3.6: Employees page

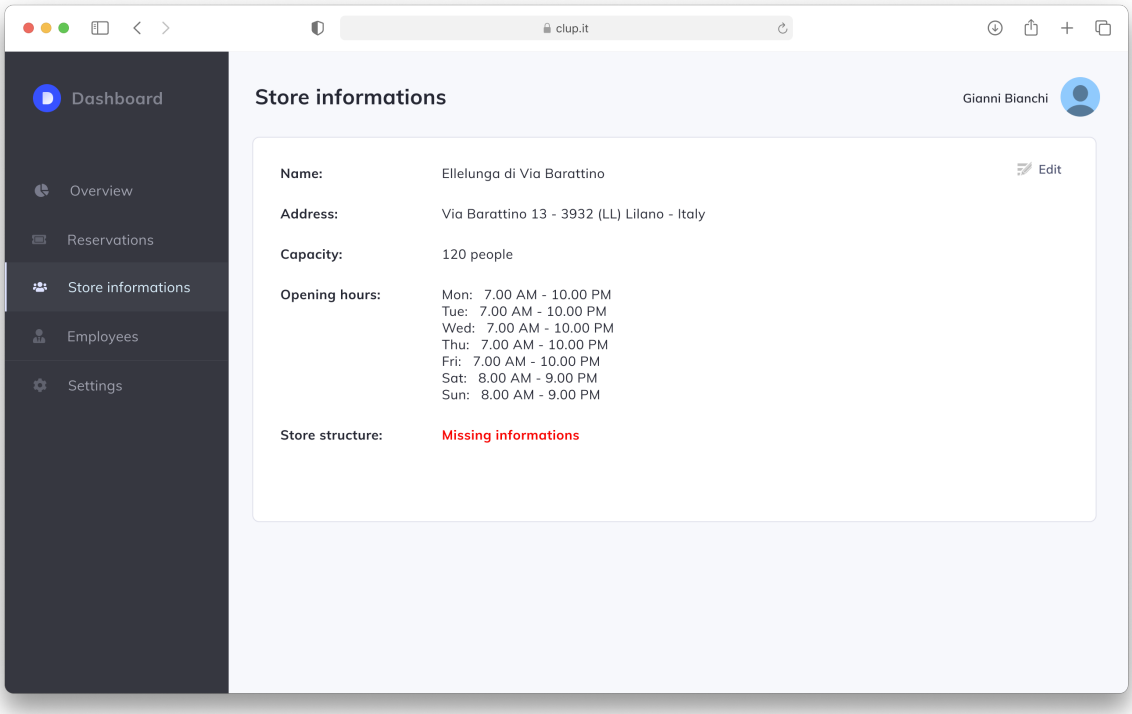


Figure 3.7: Store informations

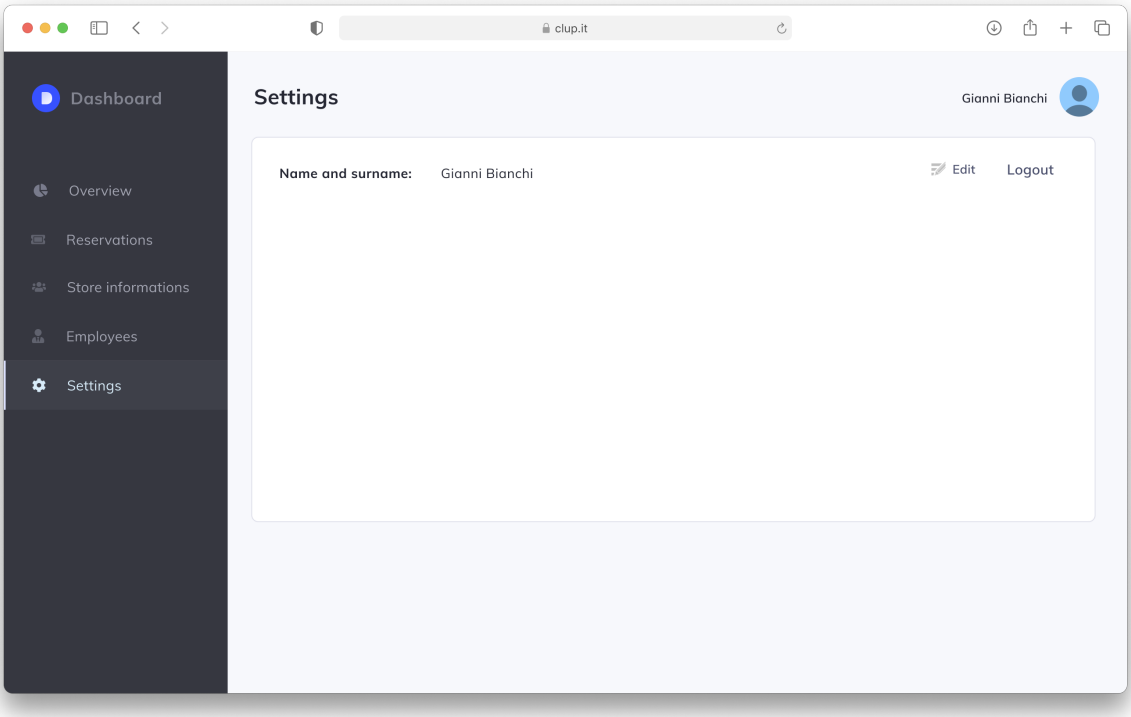


Figure 3.8: Settings

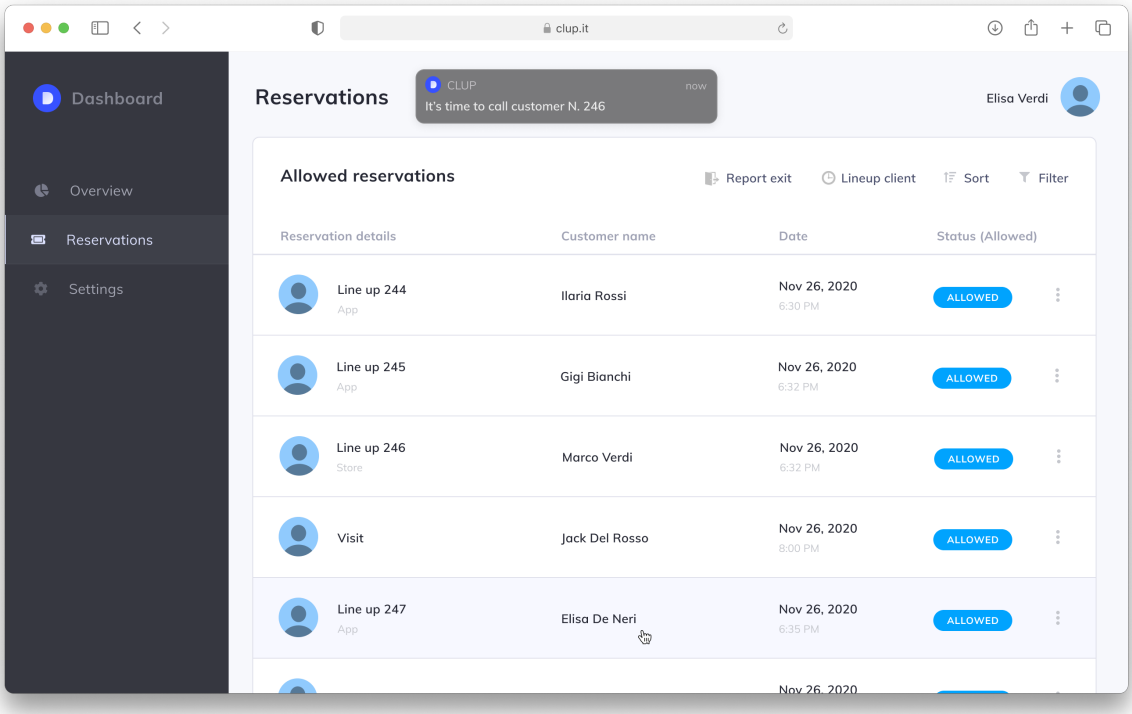


Figure 3.9: Reservation page (Employee view)

4. Requirements Traceability

This chapter explains how the requirements defined in the RASD map to the design elements (components) defined in chapter 2 of this document. To aid readability, all goals and their requirements are listed again, and then their components are mapped. In this chapter, special attention is given to the server-side modules. We ignore, therefore, the client-side components (not very significant in this project phase). Below are the requirements traceability tables.

G1	**Everyone can use and interact with the CLup system accordingly with itsfeatures and processes**
R1	Visitors are allowed to register as customers through the CLup application
R2	The store manager is allowed to register as store manager through the CLup web app
R3	Customers are allowed to log in inside the application
R4	The store manager and the employee are allowed to login in inside the web application
R6	The store manager can create and edit the employees' accounts associated to his/her store
	Components mapped to the above requirements: • Customer Mobile Services – AccountManager • Employee Web Services – AccountManager • Store Manager Web Services – AccountManager – StoreInfoAndEmployeesManager

G2	**All customers who reserve a place in the queue will be called**
R3	Customers are allowed to log in inside the application

R4	The store manager and the employee are allowed to login in inside the web application
R6	The store manager can create and edit the employees' accounts associated to his/her store
R8	The employee can report the entrance related to a line up number or a visit and the exit of each person from the store
R9	The line up numbers and the visits are invalidated after a period of time if not used to enter the store
R11	The CLup system notifies the employee when it's time to call a visitor who previously asked to line up at the entry point
R17	The application monitors the customer's position and computes the estimated travel time
R18	The application notifies the customer when the estimated queue waiting time is near the estimated travel time
R19	The system generates a unique (in an appropriate time interval) number to identify the position in the queue
R20	The system pushes forward the queue based on the reported exits and the scheduled visits
	Components mapped to the above requirements: • Customer Mobile Services – AccountManager – LineUpReservationModule • Employee Web Services – AccountManager – EntrancesAndExitsModule • Store Manager Web Services – AccountManager – StoreInfoAndEmployeesManager • QueueManager

G3	**Allow customers to enter the store once their number has been called or ifthey have booked a visit for that time slot**
R3	Customers are allowed to log in inside the application

R4	The store manager and the employee are allowed to login in inside the web application
R6	The store manager can create and edit the employees' accounts associated to his/her store
R8	The employee can report the entrance related to a line up number or a visit and the exit of each person from the store
R10	The employee can see the line up numbers/visits that are allowed to enter the store
R14	The CLup system can acquire the scanned QR code
	Components mapped to the above requirements: • Customer Mobile Services – AccountManager • Employee Web Services – AccountManager – EntrancesAndExitsModule • Store Manager Web Services – AccountManager – StoreInfoAndEmployeesManager

G4	**Customers who go to the supermarket without a number/booking are allowedto line up at the store**
R4	The store manager and the employee are allowed to login in inside the web application
R6	The store manager can create and edit the employees' accounts associated to his/her store
R7	The employee can line up a visitor who asked for that and can give him/her the number associated with the position in the queue

Components mapped to the above requirements:

- Employee Web Services
  - AccountManager
  - LineUpReservationModule
- Store Manager Web Services
  - AccountManager
  - StoreInfoAndEmployeesManager

G5	**Inside the grocery store it must be feasible to follow Covid19 regulations**
R4	The store manager and the employee are allowed to login in inside the web application
R5	The store manager can insert and edit store information
R6	The store manager can create and edit the employees' accounts associated to his/her store
R7	The employee can line up a visitor who asked for that and can give him/her the number associated with the position in the queue
R8	The employee can report the entrance related to a line up number or a visit and the exit of each person from the store
R10	The employee can see the line up numbers/visits that are allowed to enter the store
R15	The CLup application has a section to line up
R20	The system pushes forward the queue based on the reported exits and the scheduled visits
R21	The CLup application has a section to book a visit
R23	The system schedules visits based on related details

Components mapped to the above requirements:

- Customer Mobile Services
  - AccountManager
  - LineUpReservationModule
  - VisitReservationModule
- Employee Web Services
  - AccountManager
  - EntrancesAndExitsModule
  - LineUpReservationModule
- Store Manager Web Services
  - AccountManager
  - StoreInfoAndEmployeesManager
- QueueManager

G6	**Outside the grocery store there must not be long queues or overcrowding**
R4	The store manager and the employee are allowed to login in inside the web application
R6	The store manager can create and edit the employees' accounts associated to his/her store
R7	The employee can line up a visitor who asked for that and can give him/her the number associated with the position in the queue
R9	The line up numbers and the visits are invalidated after a period of time if not used to enter the store
R11	The CLup system notifies the employee when it's time to call a visitor who previously asked to line up at the entry point

	Components mapped to the above requirements: • Employee Web Services – AccountManager – EntrancesAndExitsModule – LineUpReservationModule • Store Manager Web Services – AccountManager – StoreInfoAndEmployeesManager • QueueManager

G7	**Customer is allowed to book a visit through the CLup system**
R3	Customers are allowed to log in inside the application
R21	The CLup application has a section to book a visit
R22	The CLup application shows available time slots for visits
R24	The customer can insert additional details like what he/she is going to buy, the estimated visit duration or what departments he/she will go to)
	Components mapped to the above requirements: • Customer Mobile Services – AccountManager – VisitReservationModule

G8	**Customer is allowed to line up through the CLup system**
R3	Customers are allowed to log in inside the application
R13	Customer can generate a QR code to enter the store
R15	The CLup application has a section to line up
R16	The CLup application shows the estimated queue waiting time, the queue size, the estimated travel time and the number related to the position in the queue
R19	The system generates a unique (in an appropriate time interval) number toidentify the position in the queue

	Components mapped to the above requirements: • Customer Mobile Services – AccountManager – LineUpReservationModule • QueueManager

G9	**The store manager is allowed to monitor entrances of customers that used theQR Code**
R4	The store manager and the employee are allowed to login in inside the web application
R5	The store manager can insert and edit store information
R12	The store manager can see charts and analysis related to entrances made with the QR code
R14	The CLup system can acquire the scanned QR code
	Components mapped to the above requirements: • Employee Web Services – AccountManager – EntrancesAndExitsModule • Store Manager Web Services – AccountManager – StoreInfoAndEmployeesManager • DataAnalyzer

G10	**Customer is allowed to approach the store in time with respect to his positionin the queue**
R3	Customers are allowed to log in inside the application
R16	The CLup application shows the estimated queue waiting time, the queue size, the estimated travel time and the number related to the position in the queue
R17	The application monitors the customer's position and computes the estimated travel time

R18	The application notifies the customer when the estimated queue waiting time is near the estimated travel time
	Components mapped to the above requirements: • Customer Mobile Services – AccountManager – LineUpReservationModule

5. Implementation, Integration and Test Plan

This chapter explains how to implement, integrate, and test the system designed in this document. We decided to explain this in 3 steps. The first one (**Implementation, Component Integration and Testing**) talks about how to implement the single component (the order), how to integrate and test them. The second step (**System Testing**) explains the plan to test the system implemented. The last step (**Additional Specification on Testing**) gives additional information to make easier the testing process.

5.1 Implementation, Component Integration and Testing

The system is divided in the following subsystems:

- Web application
- Mobile application
- Customer mobile services
- Employee web services
- Store manager web services
- Queue manager
- Visit scheduler
- External subsystems like: Maps APis, OS Notif. Gateway and DBMS

As said in the section 2.2, *Web application* and *Mobile application* compose the client-side of our system, *Customer mobile services*, *Employee mobile services* and *Store manager web services* compose the server-side business logic of our system, together with *Queue manager* and *Visit scheduler*.

The listed subsystems have to be implemented, tested, and integrated using a bottom-up approach. To do so, we provide a development order based on the dependencies to make possible a step-by-step implementation and components integration testing (respecting the technique of the bottom-up approach). To accelerate the development process, we opted for a mixed process, where there are macro-phases (implementation, unit testing, and integration testing) to be completed before others. As for the two sides: client-side and server-side, this can be developed in parallel, they will be integrated and tested together in the final integration part.

It should be noted that it is not considered necessary to implement the external components mentioned above as they are assumed to be external and reliable.

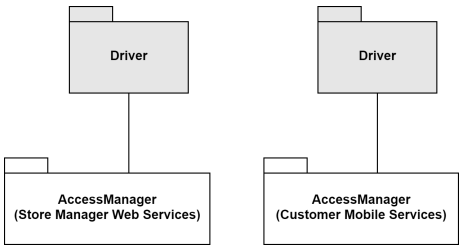
The following table presents the development stages with which we summarize the various steps of the development process and the execution order. For each step, a forecast of development difficulties has also been noted.

Order number	Development stage	Development difficult
1	Store Manager and Customers Sign up and Log in Developing	Low
2	Employee Registration by Store Manager Developing	Low
3	Employee Login Developing	Low
4	Visit Scheduling Business Logic Developing	Medium
5	Book a Visit Developing	Medium
6	Queue Business Logic Developing	Medium
7	Employee and Customer Line Up Reservation Developing	High
8	Entrances and Exits Developing	Low
9	Data Analyzer Developing	Medium
10	Statistics and Data Elaborator Developing	Low
11	Client-Side and Server-Side Integration	

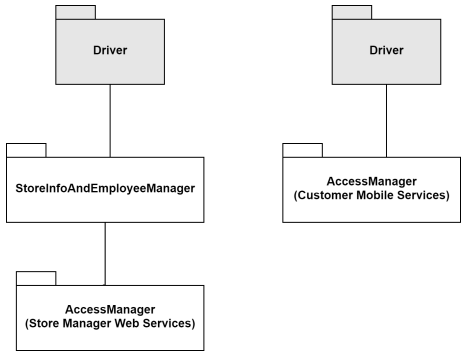
5.1.1 Development Stages Definition

In the following enumeration, we specify for each stage, what components are implemented in the stage, and how the components are integrated and tested (there is an integration diagram for each stage).

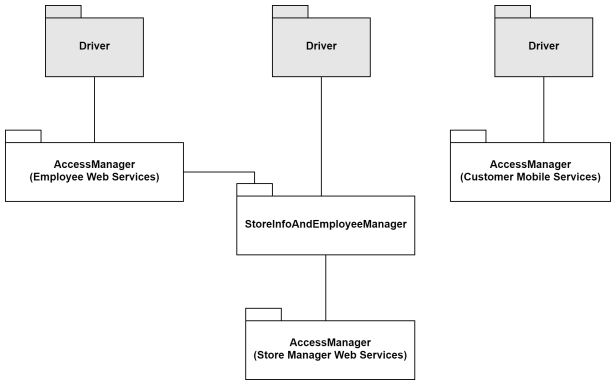
1. **Store Manager and Customers Sign up and Log in Developing**: In this step it's necessary to implement and produce unit test for the *AccountManager* component (Store Manager Web Services, Customer Mobile Services). This modules are the first two to be developed following a bottom-up approach, they communicates with the DBMS (external reliable component). We choose this as first step because some components need features developed in this stage (like store information about the actors)



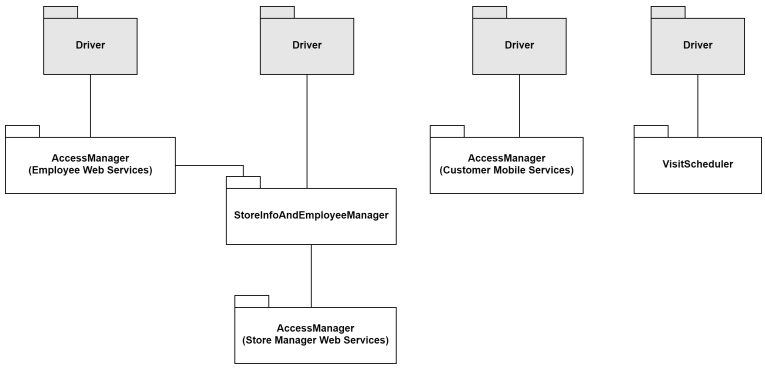
2. **Employee Registration by Store Manager Developing**: In this step it's necessary to implement and produce unit test for the component *StoreInfoAndEmployeeManager*. As you can see in the diagram, the component developed has to be integrated and tested with *AccessManager*. Following the bottom-up approach it uses a Driver to represents the high level components to be developed



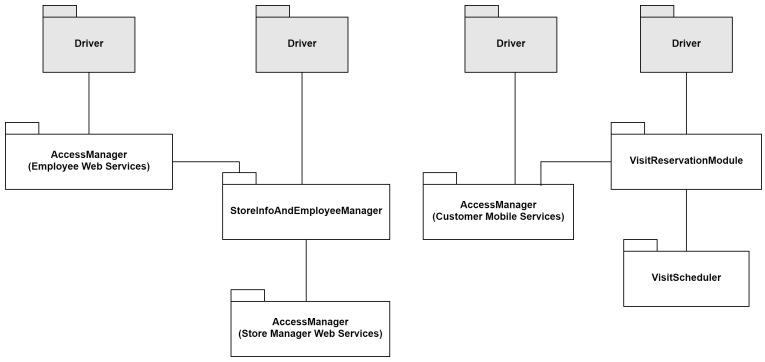
3. **Employee Login Developing**: In this step it's necessary to implement and produce unit test for the component *AccessManager* (Employee Web Services). As you can see in the diagram, the component developed has to be integrated and tested with *StoreInfoAndEmployeeManager*. Following the bottom-up approach it uses a Driver to represents the high level components to be developed



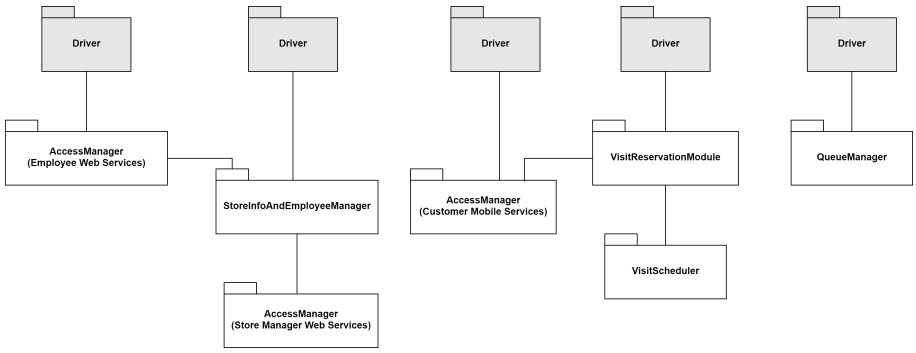
4. **Visit Scheduling Business Logic Developing**: In this step it's necessary to implement and produce unit test for the component *VisitScheduler*. As you can see in the diagram, the component in this stage hasn't to be integrated and tested with other components but, following the bottom-up approach it uses a Driver to represents the high level components to be developed



5. **Book a Visit Developing**: In this step it's necessary to implement and produce unit test for the component *VisitReservationModule*. As you can see in the diagram, the component in this stage has to be integrated and tested with *VisitScheduler* and *AccessManager*. Following the bottom-up approach it uses a Driver to represents the high level components to be developed

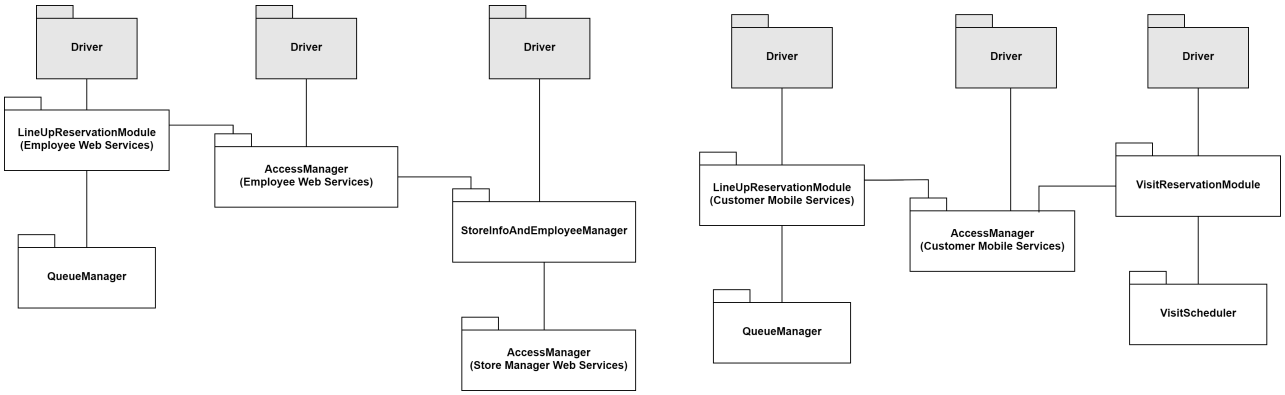


6. **Queue Business Logic Developing**: In this step it's necessary to implement and produce unit test for the component *QueueManager*. As you can see in the diagram, the component in this stage hasn't to be integrated and tested with other components. Following the bottom-up approach it uses a Driver to represents the high level components to be developed

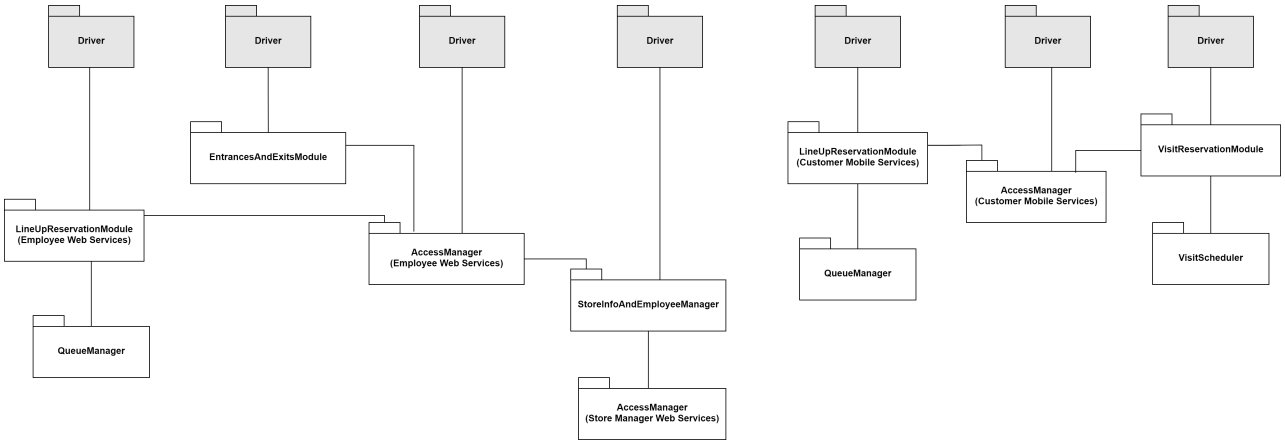


7. **Employee and Customer Line Up Reservation Developing**: In this step it's necessary to implement and produce unit test for the component *LineUpReservationModule* (Employee Web Services,

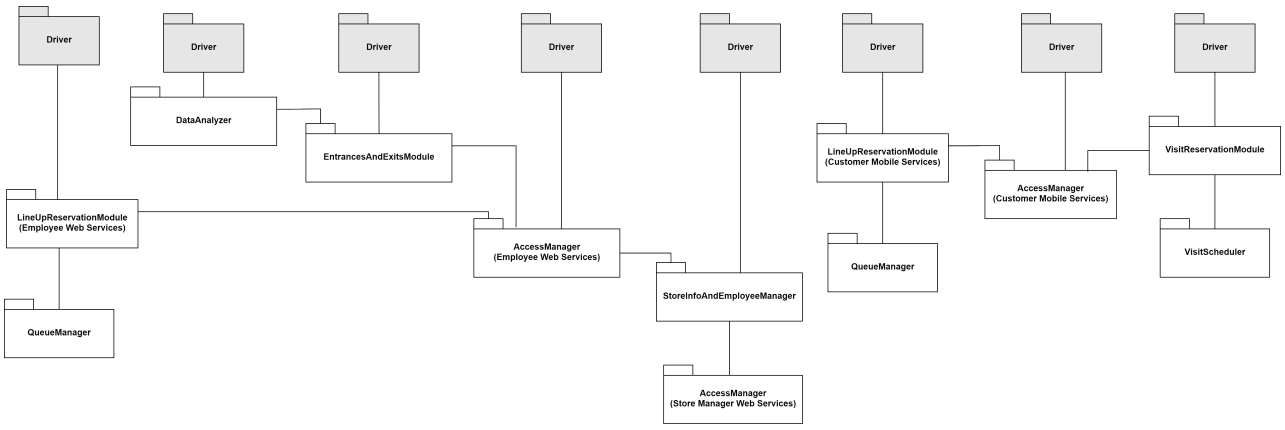
Customer Mobile Services). As you can see in the diagram, the component in this stage has to be integrated and tested with *QueueManager* and *AccessManager* (Employee Web Services, Customer Mobile Services). Following the bottom-up approach it uses a Driver to represents the high level components to be developed



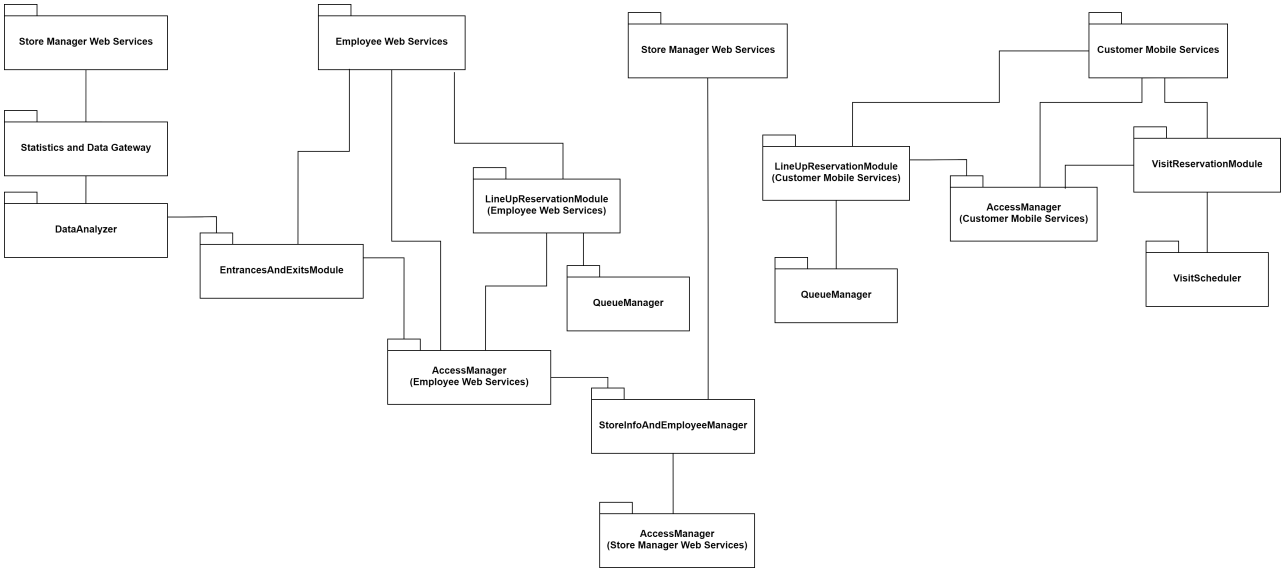
8. **Entrances and Exits Developing**: In this step it's necessary to implement and produce unit test for the component *EntrancesAndExitsModule*. As you can see in the diagram, the component in this stage has to be integrated and tested with *AccessManager* (Employee Web Services). Following the bottom-up approach it uses a Driver to represents the high level components to be developed



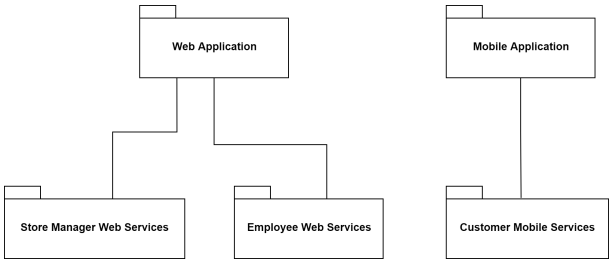
9. **Data Analyzer Developing**: In this step it's necessary to implement and produce unit test for the component *DataAnalyzer*. As you can see in the diagram, the component in this stage has to be integrated and tested with *EntrancesAndExitsModule*. Following the bottom-up approach it uses a Driver to represents the high level components to be developed



10. **Statistics and Data Elaborator Developing**: In this step it's necessary to implement and produce unit test for the component *StatisticsAndDataGateway*. As you can see in the diagram, the component in this stage has to be integrated and tested with *DataAnalyzer*. This is the last step, so, following the bottom-up approach, the top level components take the place of the Drivers



11. **Client-Side and Server-Side Integration**: In this step, as previously anticipated, the Client-Side components and the Server-Side components have to be integrated.



5.2 System Testing

Once the system is implemented, unit tested, integrated and integration tested, it must be tested as a whole, to verify that all features have been developed correctly and that they comply with the functional and non-functional requirements defined in the RASD. In order to test the latter thing, it's necessary to do system testing. At this stage, the software should be as close to the final product as possible. This testing phase should not be performed exclusively by developers, being a black-box technique, it can therefore also include stakeholders and shareholders. We can split the system testing into the following stages:

- *Functional testing*: To make sure that functionality of the product is working as per the requirements defined, within the capabilities of the system
- *Performance testing*: To identify bottlenecks affecting response time, utilization, and throughput. It needs an expected workload and an acceptable performance target before testing
- *Load testing*: To expose bugs such as memory leaks, mismanagement of memory, or buffer overflow. It identifies also the upper limits of the components. In order to test the latter thing the load until threshold must be increased and the system with the maximum load it can operate for a long period, must be load
- *Stress testing*: To make sure that the system recovers gracefully after failure. In order to test the latter thing, you have to try to break the system under test by overwhelming its resources or by taking resources away from it.

5.3 Additional Specification on Testing

In addition to the normal unit tests, integration tests and system tests mentioned in the previous sections, inspection code techniques will also be used by leveraging a team outside of the developer team.

Please comment on the code in such a way as to comply with the maintainability requirements mentioned in the RASD.

6. Effort Spent

6.1 Marco Di Gennaro

Time	**Task**
30m	Work on the LaTeX document structure
1h 30m	Writing of Chapter 1
2h 30m	Discussion on chapter 2 (Content definition)
1h	Discussion on chapter 4 (Content definition)
8h	Writing of sections 2.2 (definitions), 2.5 and 2.6
1h 30m	Writing of chapter 4
1h	Discussion on chapter 3 (Content definition)
2h	Discussion on chapter 5 (Content definition)
8h	Writing of chapter 5
1h	Revisioning of the document
27h	**Tot**

6.2 Luca Danelutti

Time	Task
2h 30m	Discussion on chapter 2 (Content definition)
1h	Discussion on chapter 4 (Content definition)
2h	Writing of section 2.1
2h	Writing of section 2.2 (component diagrams)
3h 30m	Writing of section 2.3 and 2.7
7h	Writing of section 2.4
1h	Discussion on chapter 3 (Content definition)
2h	Discussion on chapter 5 (Content definition)
4h	Writing of chapter 3
1h	Revisioning of the document
26h	**Tot**

7. References

- Diagrams made with: draw.io
- Mockups made with: Figma
- LaTeX code mades with: Visual Studio Code (LaTeX Workshop, LaTeX Utilities)