POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA,
INFORMAZIONE E BIOINGEGNERIA

Design Document

Version 1.0 – xx/01/2020

CLup

software system

Ercolani Antonio - 10621728

Vittorio Fabris - 10562731

Riccardo Nannini – 10626268

A.Y. 2020/2021

Reference Professor - Elisabetta Di Nitto

Contents

1 Introduction

1.1 Purpose

The purpose of this document is defining the main design principles of the CLup software system, taking as input the concepts defined in the RASD. This document treats many topics regarding the software design. It starts from the high-level architecture choices and continues with the description of the main components, also describing how they interact with each other. The last section is about the system implementation, integration and the testing phases, useful to the developer to put together the various design aspects during the system development.
The reader can find a more detailed list of the treated topics in section 1.3.6.

1.2 Scope

CLup is an application that aims to provide the users with the possibility to queue to enter in a store, preserving as much as possible their safety and health. The lining up procedure can be done in two different ways: nearby the store with a physical ticket or from the application, where a virtual ticket is generated. In this way the crowd in the neighborhood of the stores is reduced, and so as a consequence the risks to get in touch with other people is decreased too. People lining up from home will start approach the store only when the system provides them a notification. Customers will then enter the store only when their turn has come, by the recognition of the QRcode on their ticket.
This application is built in order to be as easy as possible so that it can be used by customers of all the ages. Customers can simply line-up to the store they prefer, but they can also take advance of other services that are implemented inside the book a visit feature. Store managers have a different interface to deal with the system, as they have different things to check and for sure different goals, such as monitor entrances/exits and then allow more people in the store if it is possible.
However, more specific and deep descriptions of the available features can be found in the RASD document.

1.3 Definitions, Acronyms, Abbreviations

In this section we explain the meaning of some technical terms used in the document.

1.3.1 Definitions

QR CODE	A *Quick Response code* is a kind of bar-code, readable by machines to retrieve information
prova	prova
prova2	prova

1.3.2 Acronyms

RASD	A *Quick Response code* is a kind of bar-code, readable by machines to retrieve information
DD	Design Document
GPS	Global Positioning System

1.3.3 Abbreviations

1.3.4 Revision History

1.3.5 Reference Documents

1.3.6 Document Structure

Here a list of the topics treated in each chapter of this Design Document.

Chapter 1 is an introductory chapter, where are presented the purpose and the scope of this document. It also includes tables about acronyms and technical definitions.

Chapter 2 is the core of the Design Document. Here we can find the main architectural decisions, starting from the high-level design patterns. Then, there's a description of every single components and the interaction with each other. Lots of diagrams are included among the different subsections to better explain the presented concepts.

Chapter 3 presents a deep description of the User Interface by means of a large number of detailed mockups and UX diagrams.

Chapter 4 contains the strongest link to the RASD. In fact, it shows a mapping between the requirements presented in the RASD and the architectural components presented in chapter 2.

Chapters 5 aims to give to the developers the guidelines of the implementation, integration and test phases, from both an high-level and a low-level point of view.

Chapters 6 and 7 contains respectively tables about the effort spent by each group member in writing this document and the document references.

2 Architectural Design

2.1 Overview

Three logic layers define the architecture of the application. This division has been made so that eventual needed updates that are related to only one of the parts does not affect the other two, leaving them independent from this point of view. The application has to be made in the Client-Server behavior, where the clients can join from different devices and work on the system through some hardware and software components that will be better explained into details in the next sections of the document. The three discussed layers are:

- Presentation Level (P) : it's the higher level of the application and deals directly with the users. It shows to the users the actions they can do in a specific moment and so it has to display them the info in a well-ordered and understandable way. All the features and functions need to be made properly available when needed

- Business Logic or Application Layer (A) : it's the intermediate level the application and so it handles the data between the other two layers, checking and elaborating them so that the logic and the functionalities of the application are preserved and handled in the right way.

- Data Access Layer (D) : It's the level that assures to keep data independent to the logic of the application. It handles the info with respect to the databases of the system, managing them when requested by the layer on its top.

Ideally, the sequence of the process begins with the request of the user to do a specific action when he wants to apply for a displayed available functionality, and to do this he will set the activation of a specific method that will trigger the application logic to manage it. When the user concludes his procedure to being inserted in the queue of a store, sends a request to the server that handles the received data: if the lining-up has been made from home then the virtual ticket (its data) will be sent to the user, otherwise If the request of the ticket has been made from the totem at the store, the data sent back from the server will contain a valid or not valid ticket (the logic takes care of how many physical tickets have already been released and stops sending valid tickets if the threshold has been reached yet).

What's more, for what concerns the users, every time they select and submit their preferences (mean of transport, favorite products, ...) they will send data to the server, which makes them stored in the databases and allows the user to proceed through the next steps of the lining up procedure. Stored data will be useful when the user requests to see his statistics: they will be elaborated and sent back to the user, that would see them displayed on the screen of his device.

The system has to deal with store managers too, and so the server will periodically update the available data of the manager, that would monitor the situation of the store in real time: this will allow the manager to send the request to allow more people enter the store, handled by the application logic and whose results are displayed back to the manager.

//TODO HIGH LEVEL DIAGRAM with EXPLANATION

2.2 Component view

Component Diagram
In the current section will be explained the interactions of the various components that the system contains. The component view diagram allows to better understand the flow in the **Application Server**, which is organized in multiple subsystems so that the different competence areas are properly divided with respect to the functionality they're built for.
The various components can communicate with each other through different interfaces. There are three components in the client side are the **customer** and the **store manager apps** and the **totem** where people take their physical ticket, which is connected to the application server too. The **Router** takes care to redirect to the proper subsystem or component the requests and responses that are generated during the lifetime of the system.
In the Application Server the **StoreHandler** and the **CustomerHandler** components are made so that they can handle the basic functions of the communication between the system and the store on one hand and between the system and the customer on the other one. General apps notifications and features that are not included in the ones that populates the queuing and book a visit subsystems rely here. In this way, if the system will handle and implement in the future some other small possible functions, these two components could be the ones that could handle the communications to guarantee to the store managers and the customers the best experience possible in the application.

What's more, the application server is connected to:

- **QRcodeManager**: it cares about the generation of the tickets.
- **StoreSlidingDoors**: it allows to open the doors of the shop if the scanned ticket is valid.
- **GoogleMapsService**: it is necessary to retrieve data about the position of the store and of the customer.

For the sake of completeness it has been shown also the **DBMS service**, but the main focus is made towards the Application Server, which is the core of the application's logic.

In the following paragraphs the various subsystems will be briefly explained.

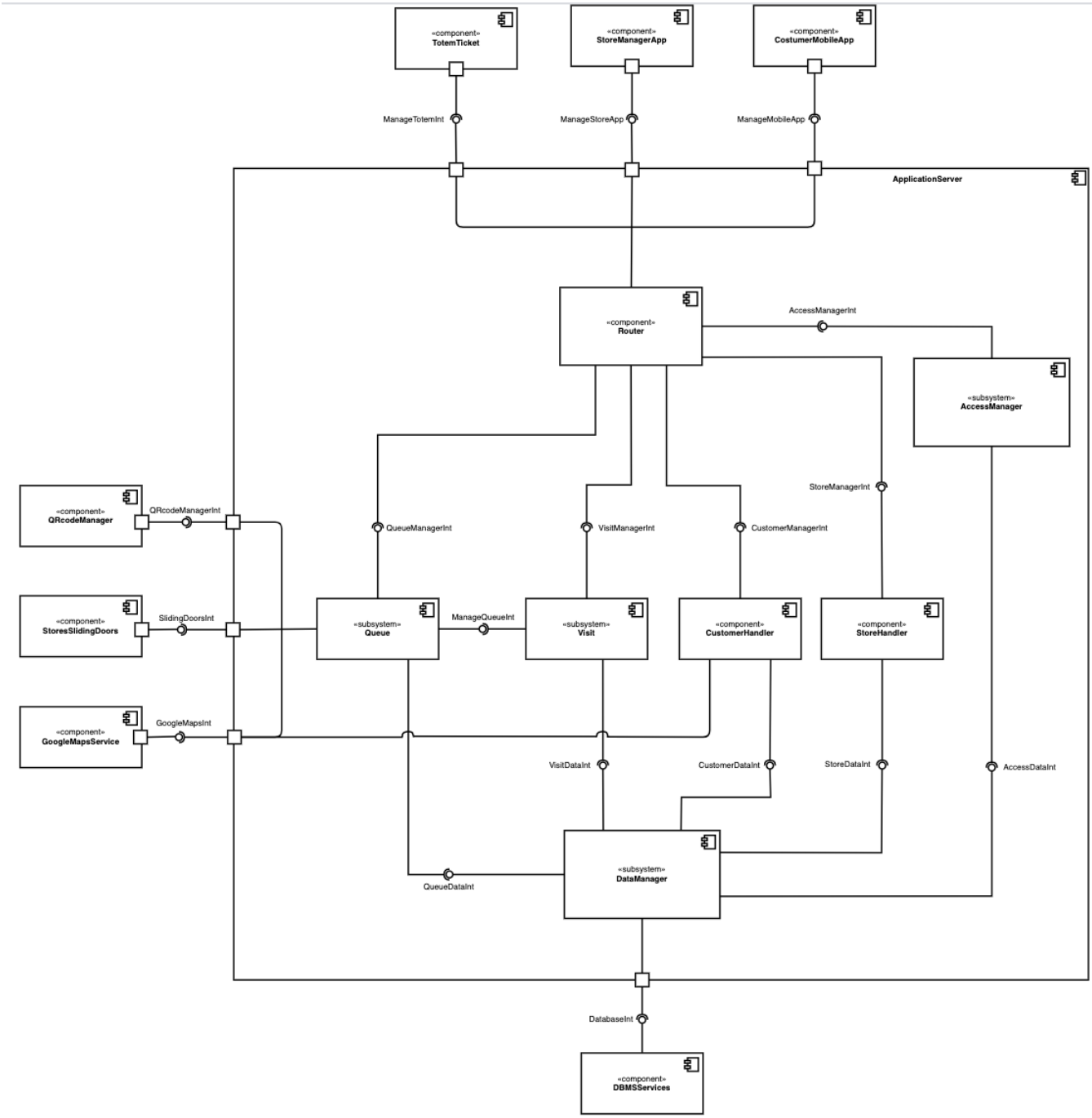


Figure 1: General Component Diagram

Access Manager subsystem
The access manager cares about the sign up and the login of the users, both the customers and the store managers. The access is handled differently for the two entities, that have to provide different kind of information (the signing up and the authentication is different if the user is a store, because the system needs to accurately verify through some given documents the validity and the existence of the shop, while the sign up of a customer is more immediate because an existent valid email and a password would be sufficient), and the access to the data manager can be made with different purposes: with the sign up data are stored in the database, so that users are registered to the system, while with the login the validity of the inserted data (username and password) are checked with the ones that are already stored in the DB.

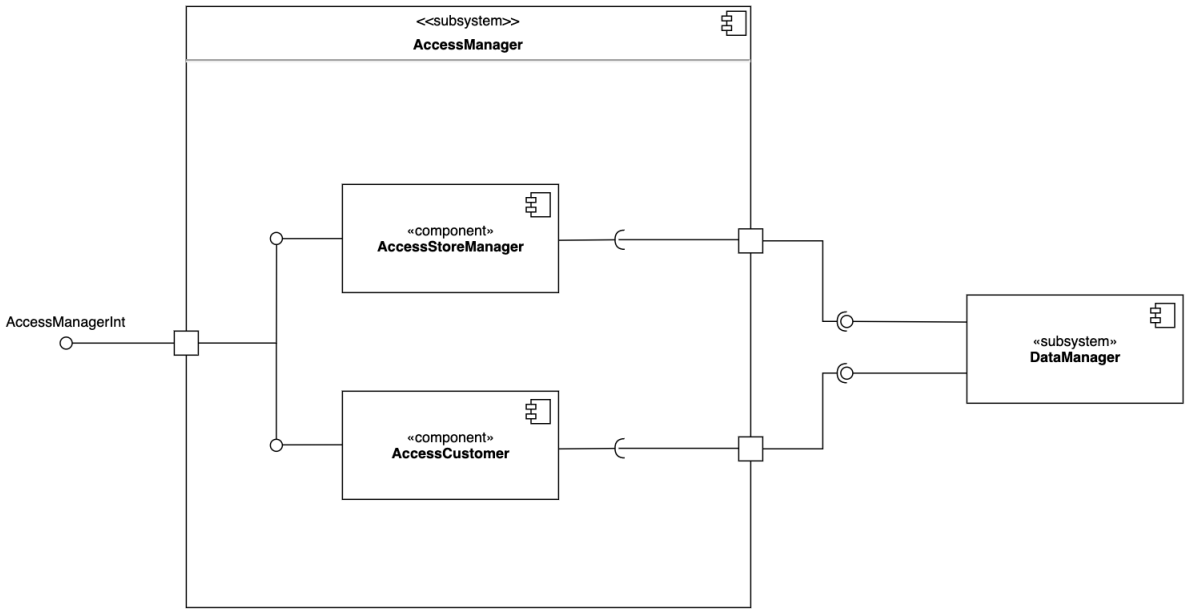


Figure 2: AccessManager subsystem component diagram

Queue subsystem
The Queue subsystem is the part of the system devoted to handling the people queueing to the store. Its responsabilities concern updating and managing the queue of people to a certain store, provide notification to users when they need to start approaching the store and handle the creation/recognition of Qr code tickets interfacing with an external module. The Queue subsystem is composed of three main components:

- **QueueHandler** is devoted to handling the queue logic beside of the application. It is responsible for putting users in the queue as well as removing them when their turn has come. It interacts with the Data Manager to store data about the queue and with the Sliding door interface to lock/unlock the door accordingly to Qr code scannings.
- **NotificationService** takes care of notifing CLup users when they need to reach the store. It interfaces with GoogleMapsAPI in order to estimate the time a user needs to reach the store given his position and transport mean.
- **Ticket Manager** is responsible for interfacing with the external software component *QrcodeManager* in order to create, assign and recognize Qr codes.

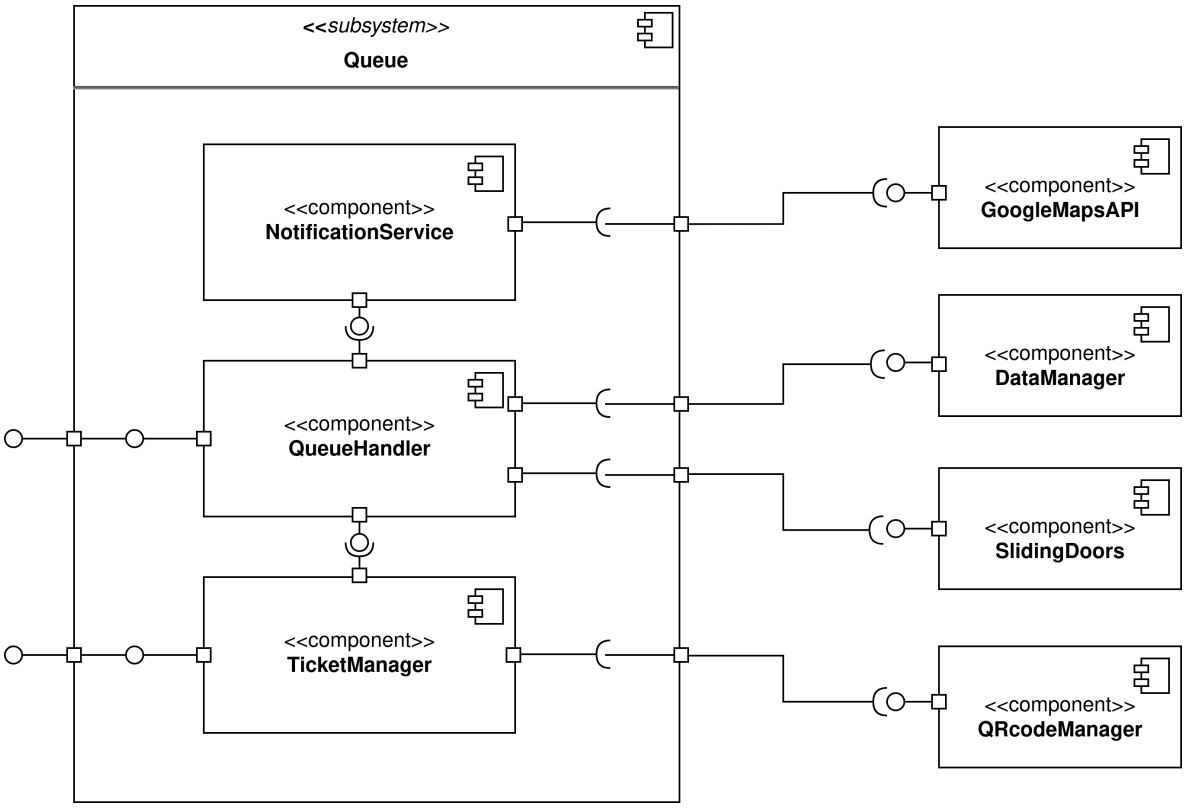


Figure 3: Queue subsystem component diagram

Visit subsystem

The visit subsystem aims to handle everything that is related to booking visits. It is composed by 4 components as the reader can see in the picture below.

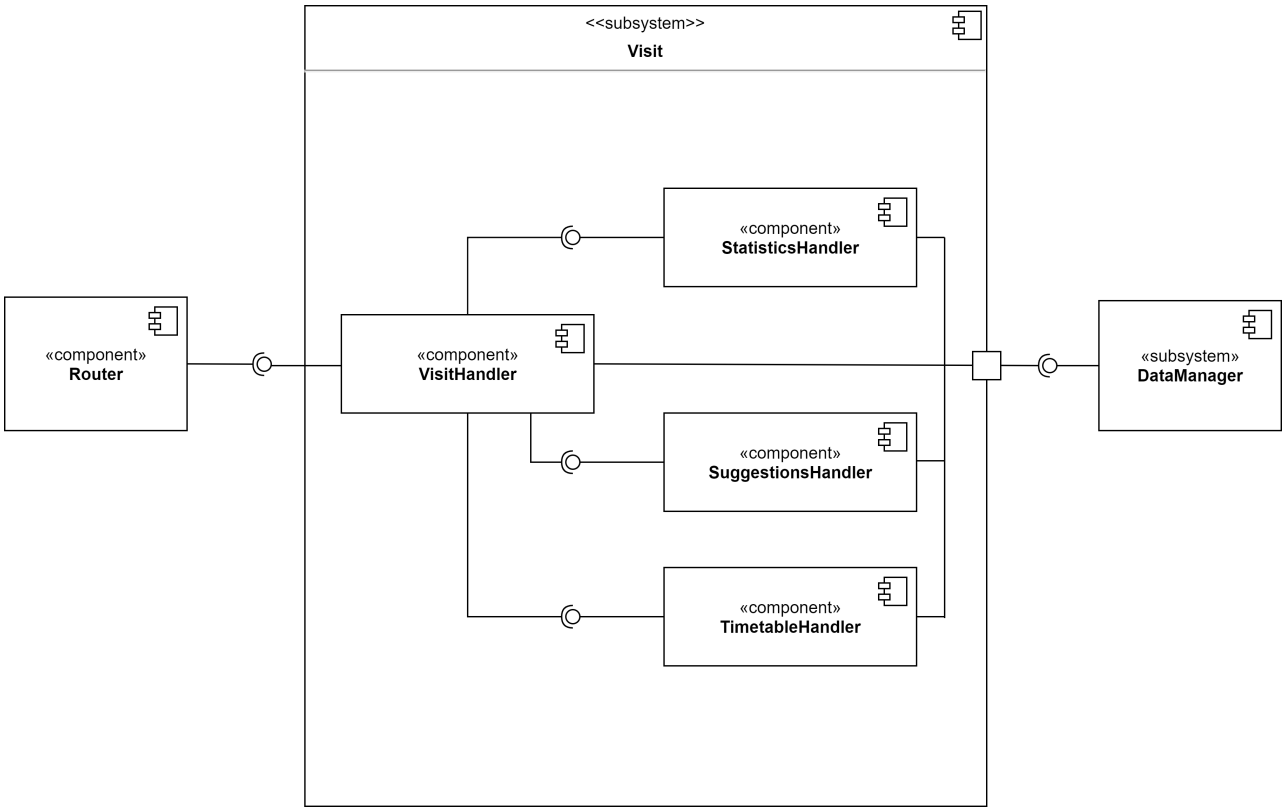


Figure 4: Visit subsystem

- **VisitHandler** fulfill the main booking visits tasks and works as an "orchestrator" of this subsystem. Firstly it handles the requests that comes from outside this subsystem. To performs its tasks VisitHandler combines its "general" capabilities with the more specialized functionalities offered by the other components;
- **StatisticHandler** collects user statistics and provide them to VisitHandler. There's a strong connection with the DataManager component which is a link to the database, where the statistic are stored;
- **SuggestionsHandler** computes the suggestions showed to the user during the booking visit phase;
- **TimetableHandler** take care of the visits timetable. For example it is able to return the available visits time-slot.

Data Manager subsystem

The purpose of the Data Manager subsystem is creating a common interface, shared with ther whole system, to access data stored in the database.
Instead of using directly the DBMS API, each component that needs to manipulate data that are stored in the DB is forced to use one of the two components that make the subsystem.
The reason behind this decision is to **decouple** our system from the DBMS: a change in the DBMS API or switching DBMS vendor will affect only our Data Manager subsystem instead of the whole application.
In particular:

- **DataRetriever** is devoted to data fetching

- **DataStorer** is responsible for storing data in the DB

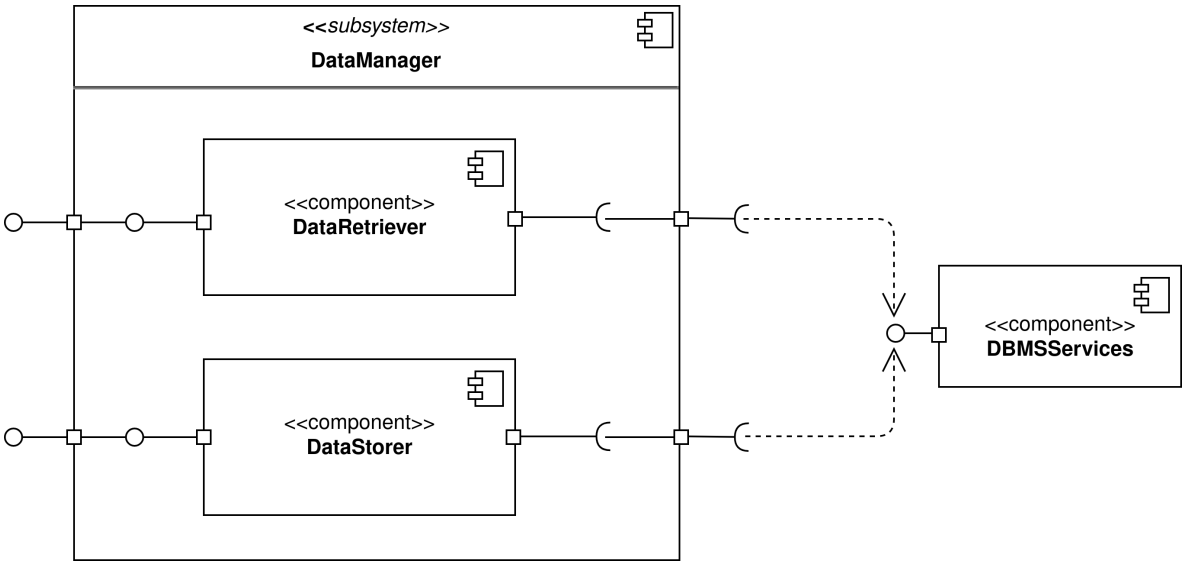


Figure 5: Data Manager subsystem component diagram

Moreover, the following diagram shows a general UML representation of the system's Model.
Each component that interacts with the Model is supposed to interface with the DataManager component and invoke the methods for retrieving/storing data.

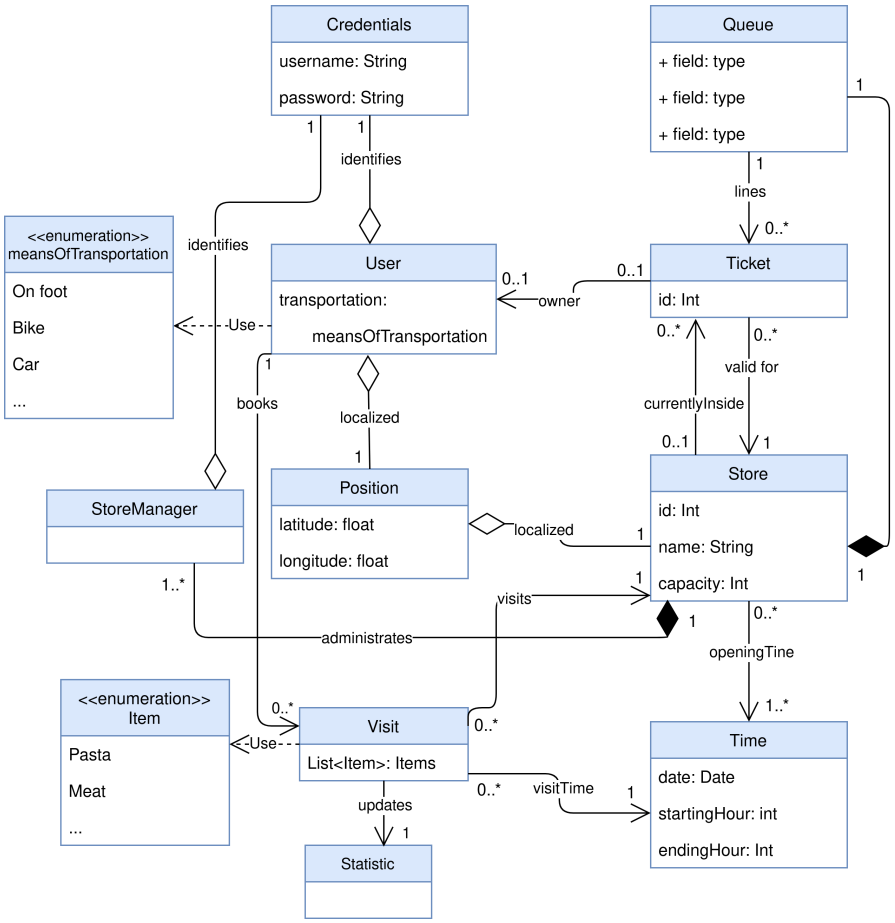


Figure 6: Model UML diagram

2.3 Deployment view

The figure below shows the distribution of the CLup components among the hardware ones. Some details are omitted, like the external interfaces, to maintain the focus on the main components of each architecture tier.

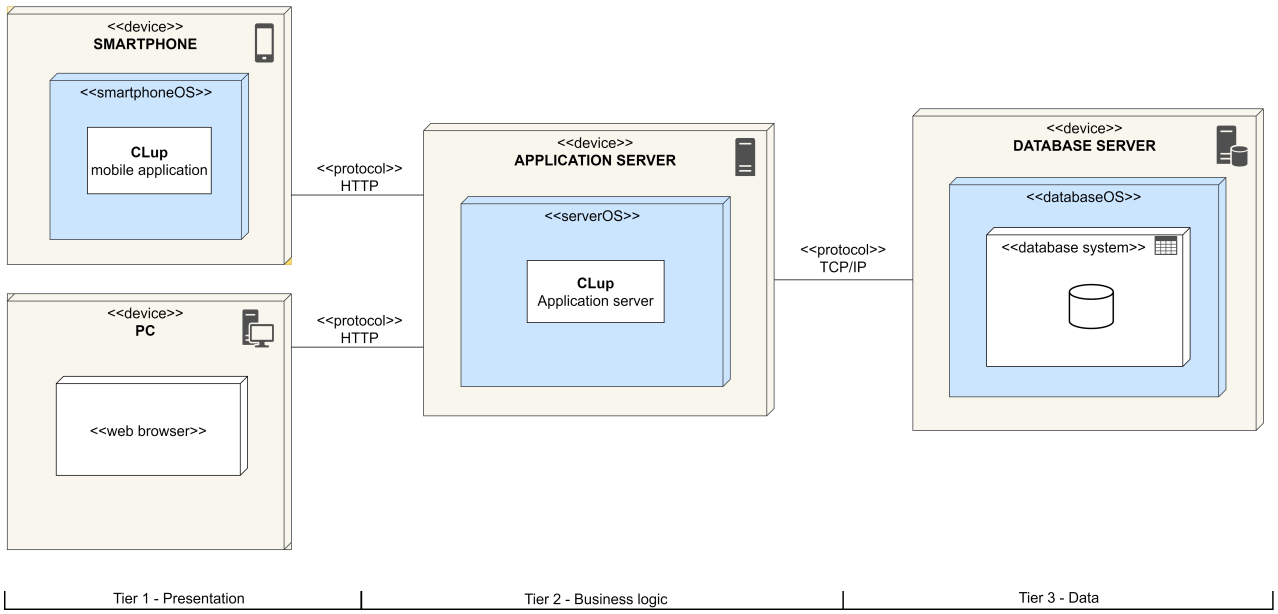


Figure 7: CLup deployment view

The presentation tier contains the devices by which costumers and store managers can interact with the system. Costumer can use their own smartphones where they have to install the CLup mobile application. There are no restrictions about the operating system of the device, the app must be developed both for Android and IOS, for being compatible with the almost totality of the smartphone.
On the other side, store managers can interact with CLup with any personal computer by means of the CLup web application, therefore they don't have to install anything but a web browser. The reader should notice the communication protocol between the personal devices and the Application Server: HTTP.

The intermediate tier contains the core of the CLup software system. Firstly, as the reader can see in the component diagram, the Application Server performs all the tasks related to the business logic. Secondly, this tier has the task of manage all the requests that come from the users devices. In this case the technology involving the server device and its operating system are left to the developers.

Finally, in the third tier, Database Server, is located the data management. Whenever the Application Server has to store or retrieve data it has to exploit the components of this tier. As in the second tier, the choice of the device of this server and its operating system is left to the developers. The only recommendation is about the nature of the database. A relational one must be preferred for a better general organization of the CLup data.

2.4 Runtime View

The sequence diagrams presented below show the behavior of the system, in terms of components interaction. The diagrams are focused on the back-end computations, the interaction between the mobile application and the application server is not considered.

Line-up

The first sequence diagram takes into account the remote line-up of a costumer. After the costumer chooses the store and communicates the intention of line-up, the system starts the line-up procedure. QueueHandler, according to the data stored in the database, computes the feasibility of the line-up. In the case it is possible, TicketHandler and QRcodeManager generate the QRcodeTicket that is sent to the costumer.

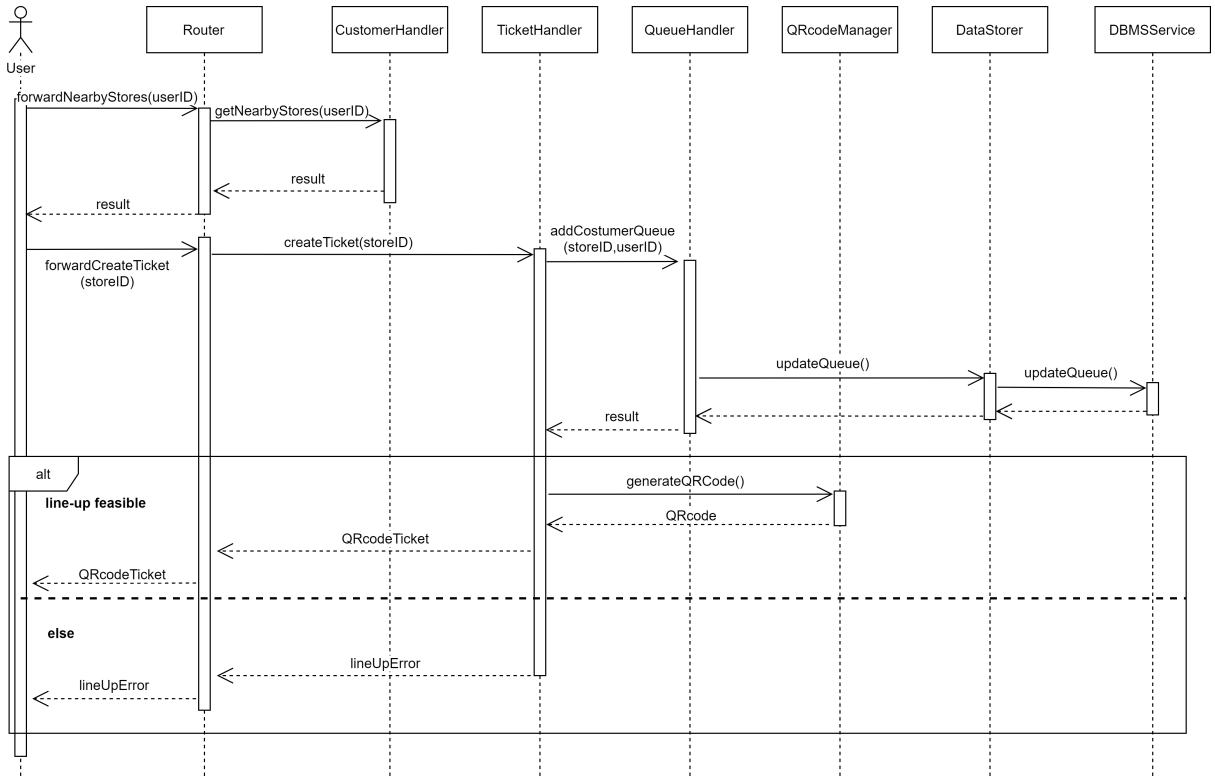


Figure 8: Line-up sequence diagram

Physical Line-up

Everything starts with the costumer that requires a ticket at the totem outside the store. From this point the flow of events is the same as the previous diagram. The unique difference is that the ticket is not sent to the costumer mobile phone but is returned by the totem, if the costumer is allowed to line-up.

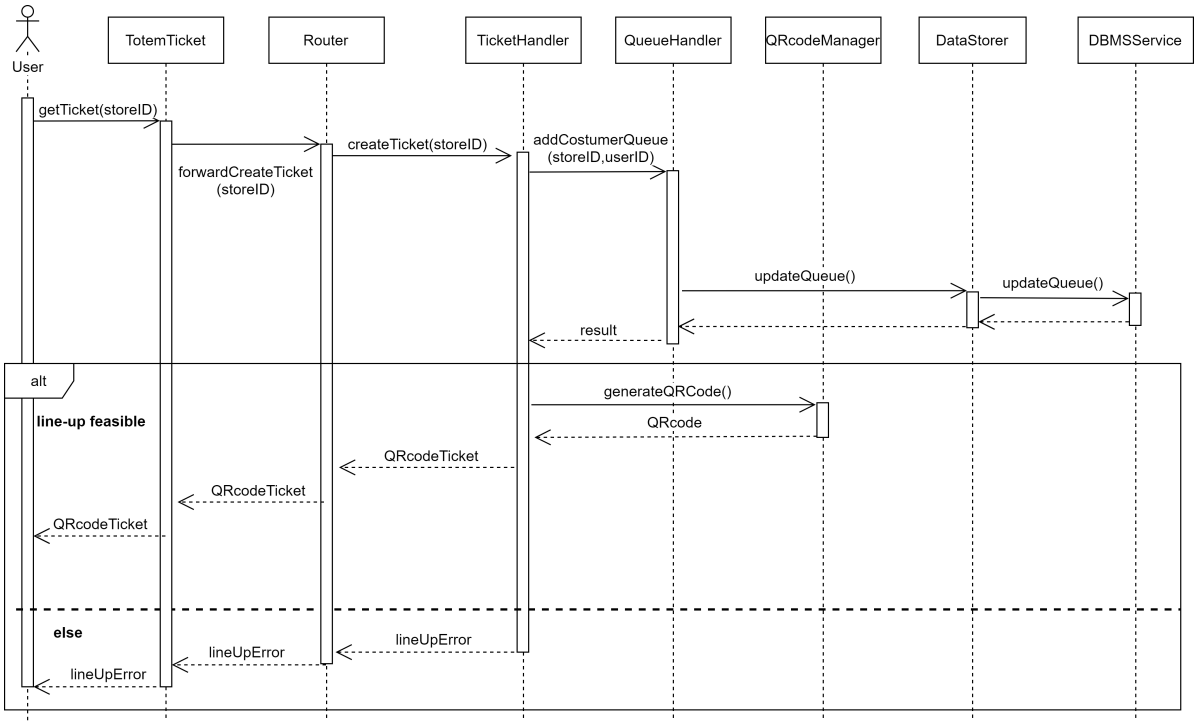


Figure 9: Physical Line-up sequence diagram

User enters into the store

When the user scans his QRcode (either via his mobile app or physical ticket), the request is handled by TicketManager that retrieves the virtual ticket associated with the code.
Afterwards, QueueHandler checks whether the scanned ticket is actually the next user in the line. If so, the automatic doors are unlocked and the queue is update, otherwise an error message is reported to the user.

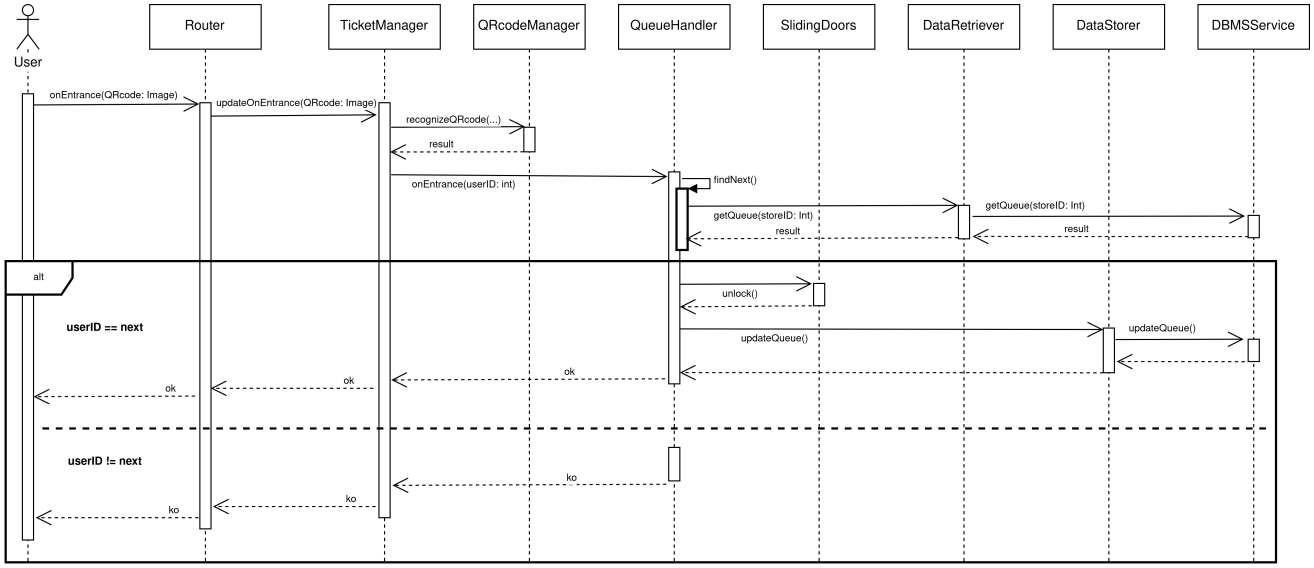


Figure 10: User enters into the store sequence diagram

User books a visit

The user starts by indicating the item that he is going to buy among the available ones and the preferred time slot.
The system checks if the visit is feasible (visit overlapping, store opening time etc.) and creates a visit object in the database along with some statistics. An error is reported to the user if some checks went wrong.

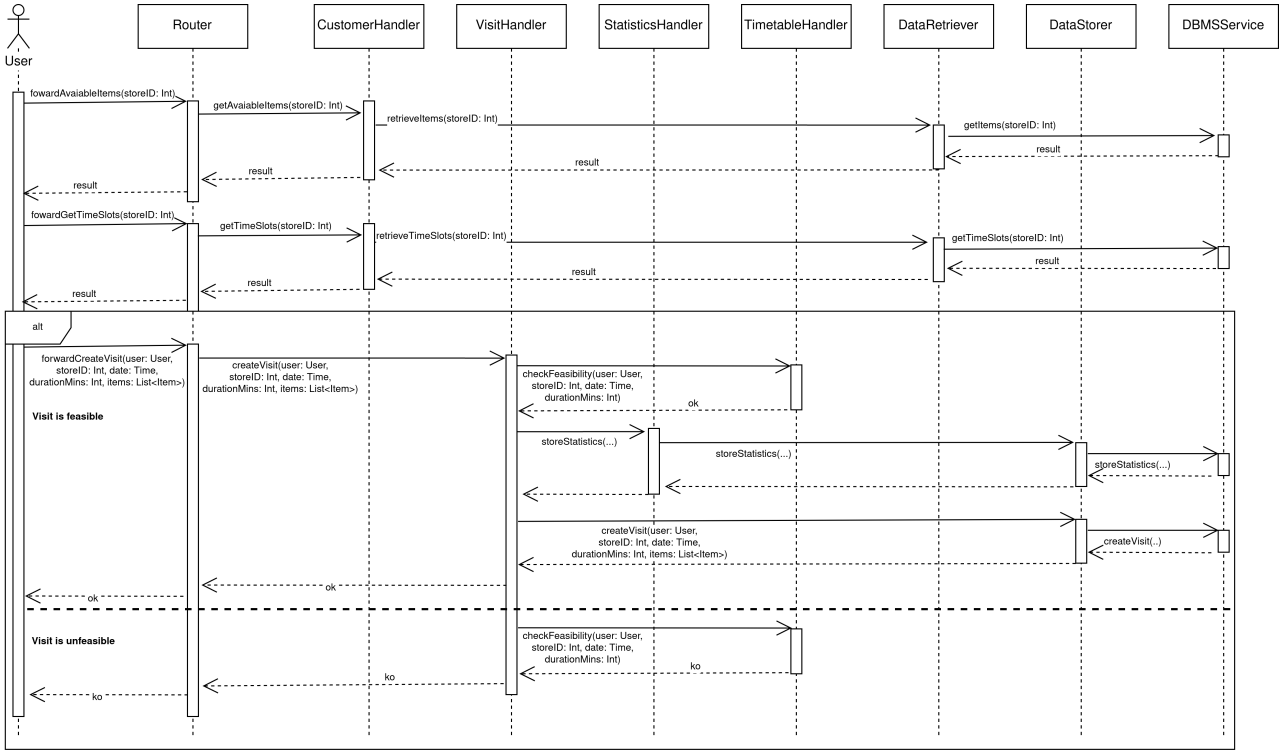


Figure 11: User enters into the store sequence diagram

Increase store capacity

The increase of the capacity starts when the store manager clicks on the correspondent button (this interaction is omitted as said in the section opening). The critical component involved in this scenario is StoreHandler that according to the store data, retrieved from the database, decides if the capacity can be increased or not.

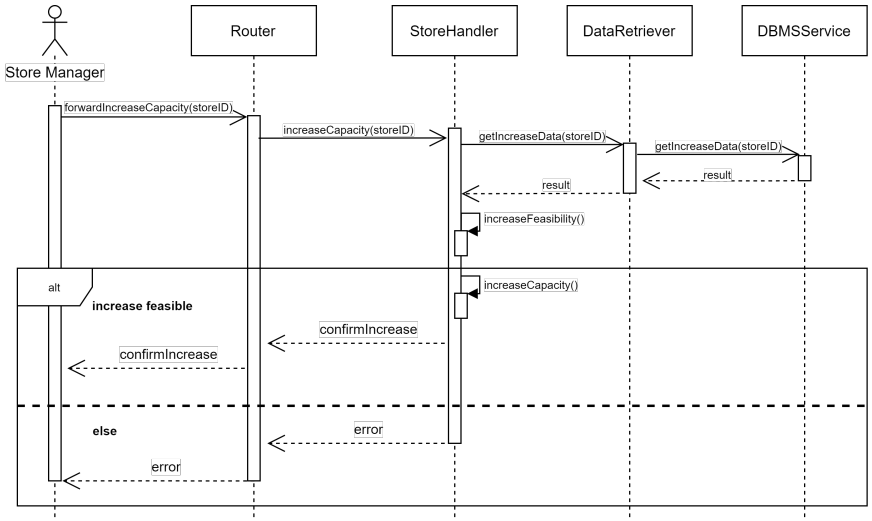


Figure 12: Increase store capacity sequence diagram

2.5 Component interfaces

The following diagram describes the main component interfaces of the system.
They are not necessarily supposed to be developed as they are but their goal is to give an overview of what type of methods the interfaces are going to need.
Some minor interfaces were omitted while the *RouterInterface* and *DataManagerInterface* are not described as they are directly correlated to the other interfaces and can be easily derived.

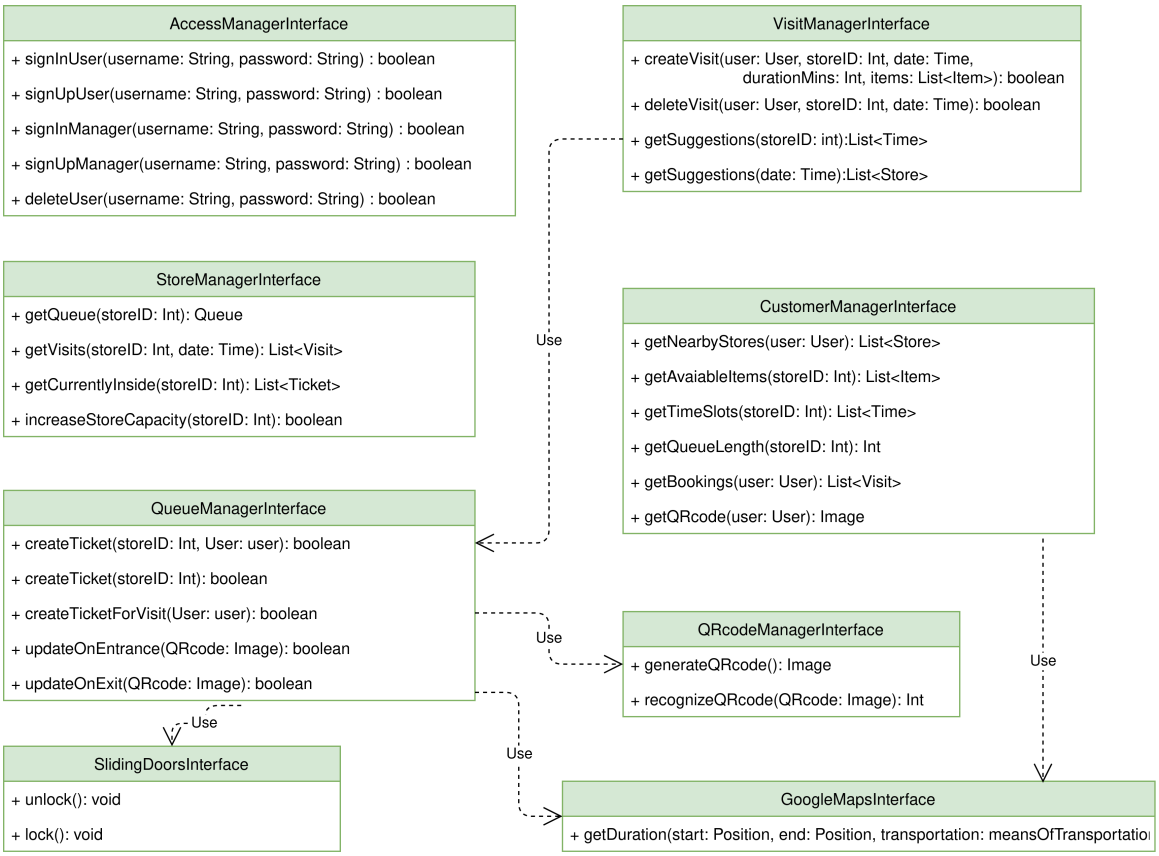


Figure 13: Component interfaces

2.6 Selected architectural styles and patterns

2.7 Algorithms and data structures

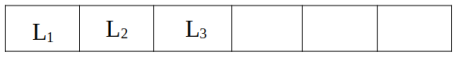
2.7.1 Queue

One of the challenges faced when designing the system is how to properly integrate, in the same virtual queue, in person customers, CLup users that lined up with the mobile app and CLup users that booked a visit.
In order to manage the line as smoothly as possible, booked visits are being treated similarly to a normal line up.
Visit Handler needs a routine that, once the time of the booked visit is reached, inserts our booked user in the queue.
This insertion however is not like other insertions because the user is virtually inserted as the first of our queue.

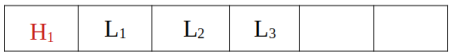
In order to achieve this behavior and ease the integration between different type of users, the data structure of the queue should be a *priority-driven FIFO queue*.
The ordering policy of the queue is the following: entities are ordered firstly based on their priority (i.e. no entities can be preceded by an entity of lower priority) and secondly in a *first come first served* manner (i.e. FIFO ordering between entities of the same priority).

This design is exploited as follows: given two priority values **H** and **L** (with priority(H) > priority(L)), requests to line up (either physically or through the CLup app) are served by inserting the user in the queue with the L priority (this creates a plain FIFO queue if only this types of requests are present).
Instead, booked visits are inserted in the queue with the H priority, placing them before other L entities.

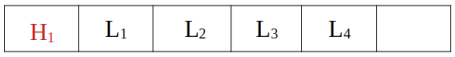
The following example shows the queue logic explained above.
The queue is intended to be read from left to right (i.e. the leftmost item is the first of the queue).



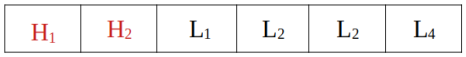
(a) Initial state of the queue



(b) H1 enters the queue



(a) L4 enters the queue



(b) H2 enters the queue

With this policy, once a visit time has come, the related user is inserted in the queue with the H priority making him the first (or among the first if many booked users are in queue) to enter as soon as a customer exits the store.
This queue can be implemented in several ways, two of them are:

- Two different FIFO queues, one for the H priority and one for the L. The next() method of the queue could be as follows:

```
/*Java example: consider H and L declared as follows
import java.util.LinkedList<E>
LinkedList<Ticket> H = new LinkedList();
LinkedList<Ticket> L = new LinkedList(); */

public Ticket next() {
        if (H.peekFirst() != null) return H.pollFirst();
        return L.poolFirst();
}
```

- A unique list/array where every item has a flag that specifies his priority. Insert operation will put items in the structure as described by the logic explained above while the next() function will simply retrieve the first element.

2.7.2 Timetable

When the system is handling the generation of a visit it has to check whether or not the user has other visits that would overlap with the new one.
This can be done by retrieving all user future booked visits and using the following function checkOverlapping().
Note that the function's inputs are Time objects previously defined in the UML class diagram.

```
//Java example
import java.util.Date;

//true if t1 and t2 are overlapping, false otherwise
public boolean checkOverlapping(Time t1, Time t2) {
        if (t1.getDate().equals(t2.getDate())) {
                return (t1.getStartingHour() <= t2.getEndingHour() &&
                        t2.getStartinHour() <= t1.getEndingHour());
        }
        return false;
}
```

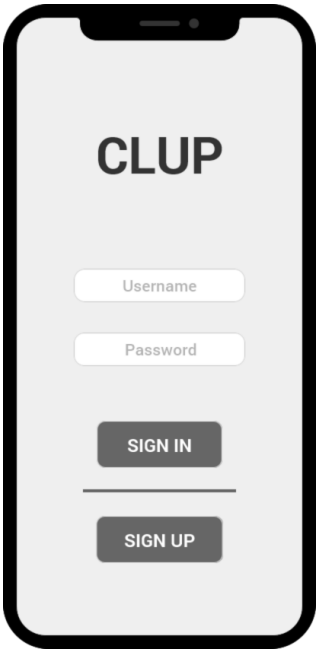
3 User Interface Design

This chapter aims to give a general idea of the user interface, both of the costumer mobile application and the store manager webapp. This is done by means of mockups and UX diagrams.

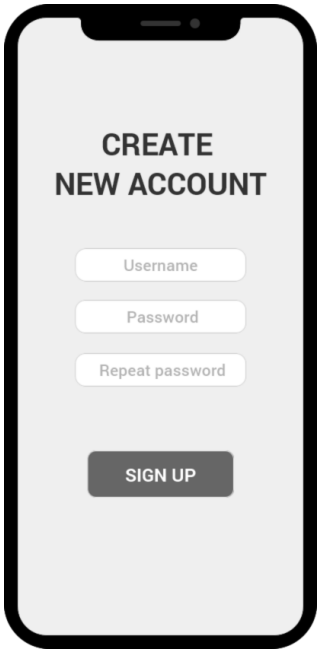
3.1 Mockups

In the design process of the user interface the main guideline was the Requirement R2 (*The user and manager applications are clear, intuitive and simple to use* - RASD). Therefore, the screen contains only the necessary components and the interaction with them is limited to a few of intuitive form and buttons.
The presented mockups cover almost the totality of the screens you will find on the completed user interface. They show only the components needed to address the CLup goals, buttons such as "return to the previous screens" are not usually taken into account.

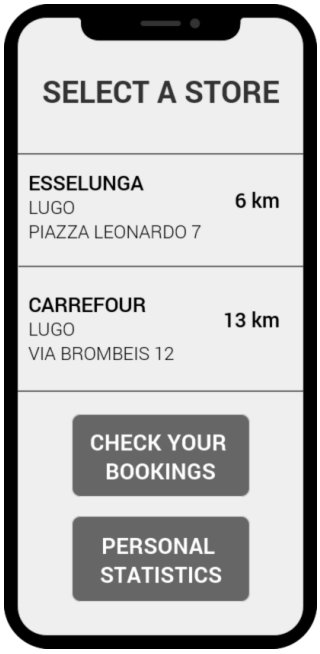
3.1.1 Mobile Application Mockups



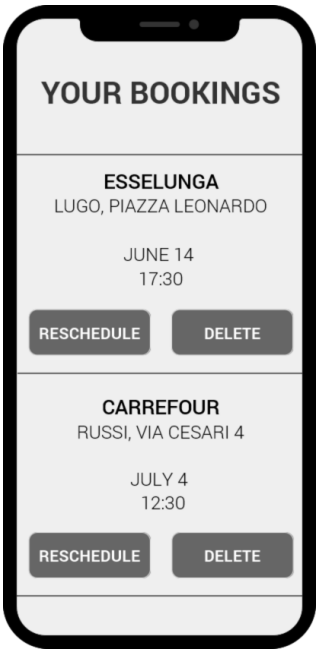
(a) Sign-in screen



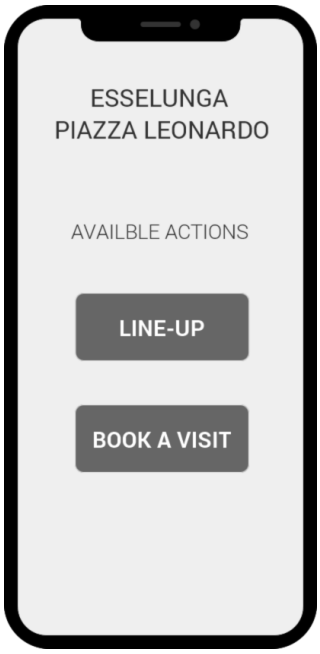
(b) Sign-up screen



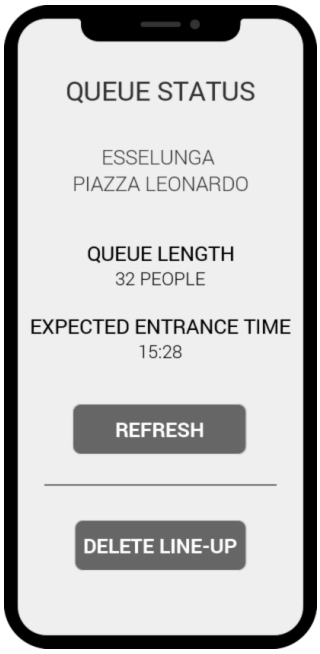
(c) Store selection/booking check screen



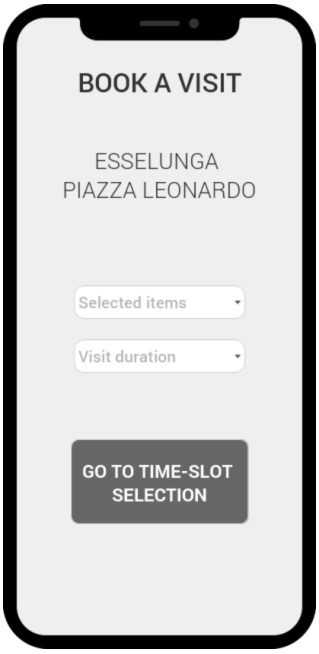
(d) Booking check screen



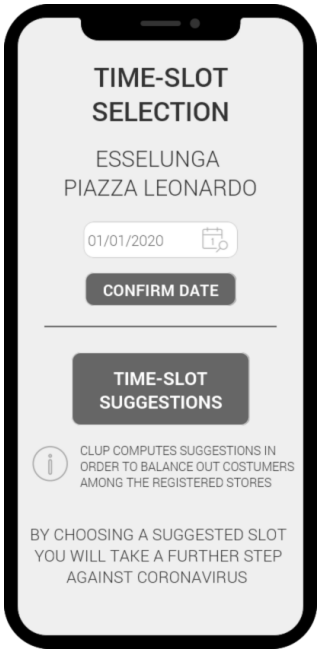
(e) Store available actions screen



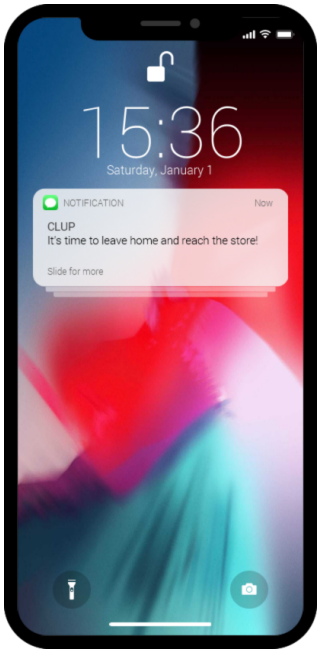
(f) Queue status screen



(g) Book a visit screen



(h) Time-slot selection screen



(i) CLup notification

3.1.2 WebApp Mockups

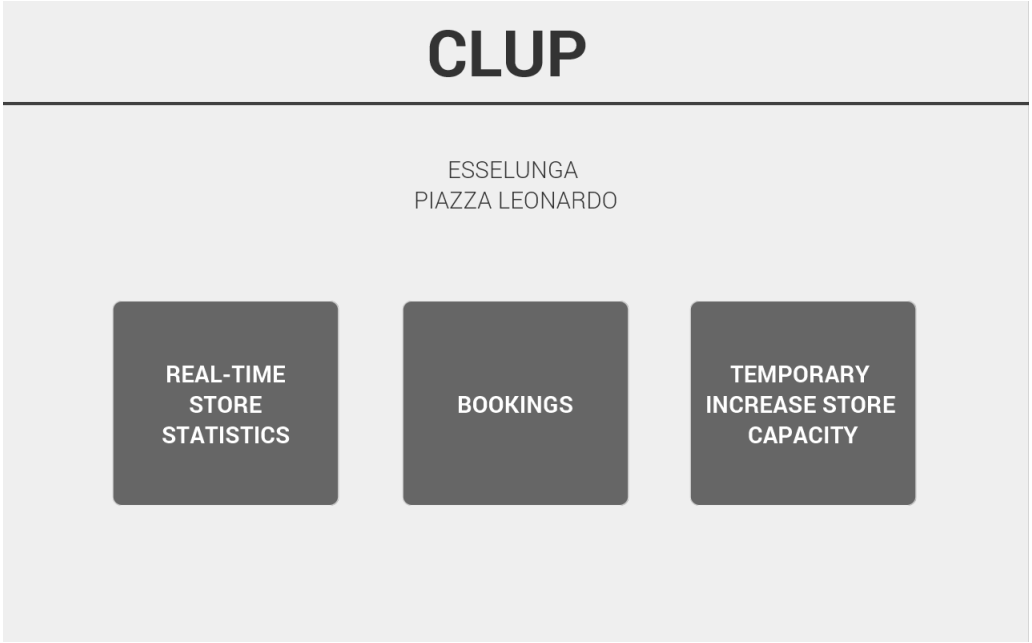


Figure 17: Home screen

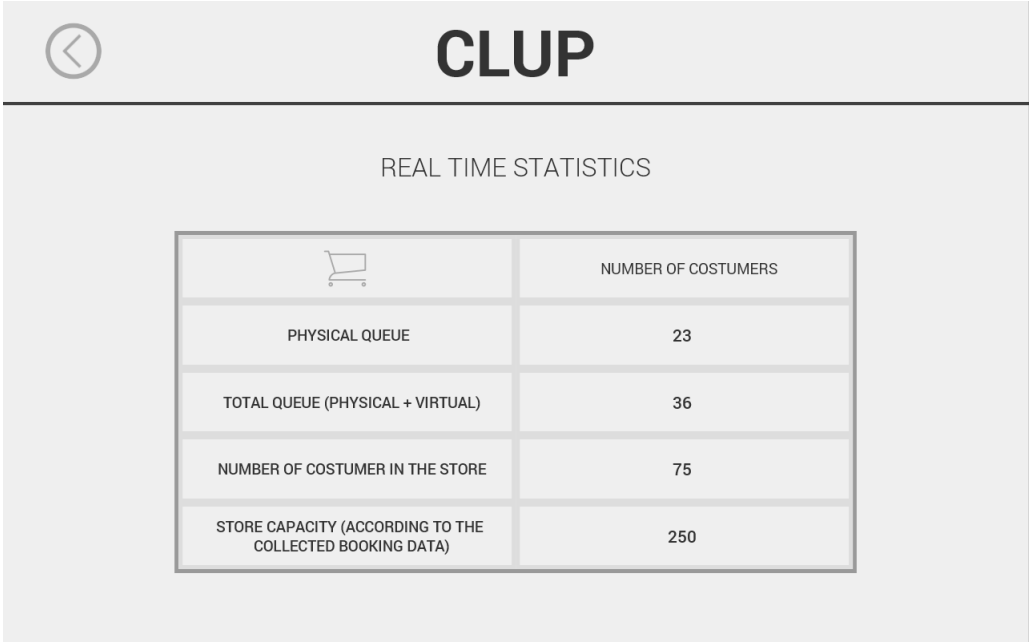


Figure 18: Store statistics screen

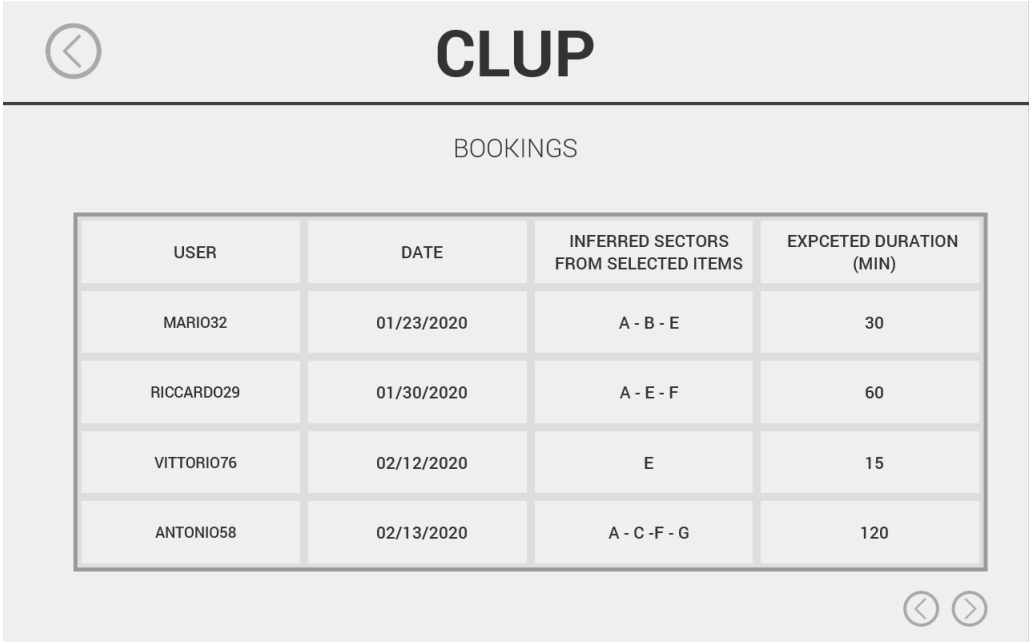


Figure 19: Store bookings screen

3.2 UX diagrams

The chapter ends with two UX diagrams. They show the general flow of the screens giving more details on the user experience. "General" because some link between screens where omitted, like the "return" from a screen to the previous.

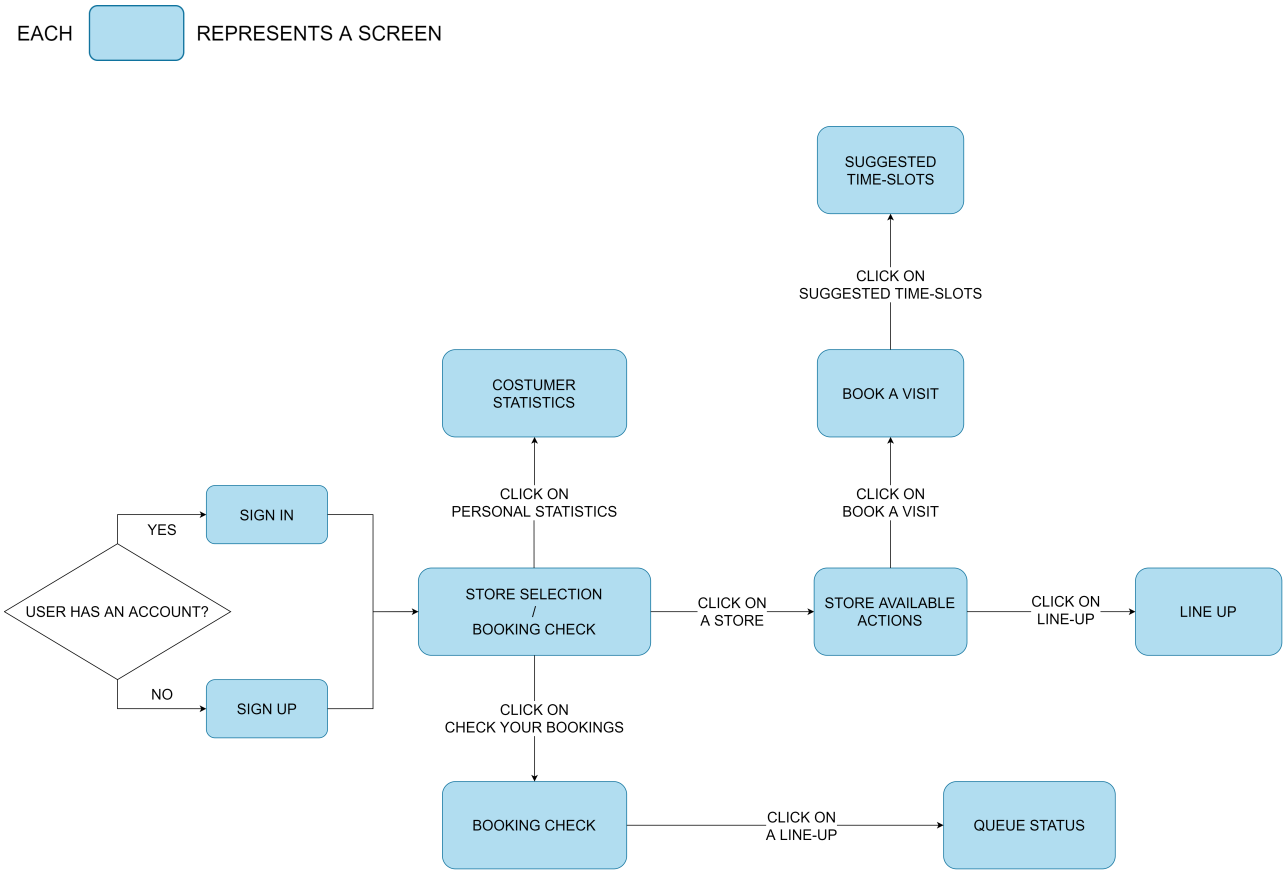


Figure 20: Mobile App UX diagram

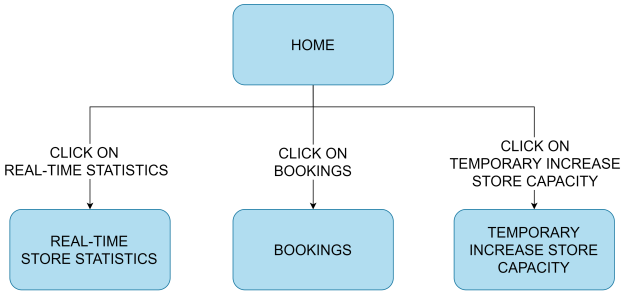


Figure 21: WebApp UX diagram

4 Requirements Traceability

In this section is presented a mapping between the CLup requirements (the reader can find them in the RASD - section 3.2.1) and the software components - section 2.2. The mapping is made looking for the components that perform the main operations in the satisfaction of a specific requirements. For example, when a costumer line-up in a store a lot of components are involved but the fundamental one, that manages the line-up, is the QueueHandler.

REQUIREMENT	**MAPPED COMPONENTS**
R1	AccessCostumer
R2	CostumerMobileApp
R3	VisitHandler CostumerHandler
R4	VisitHandler SuggestionsHandler TimetableHandler
R5	QueueHandler
R6	VisitHandler SuggestionsHandler
R7	TicketManager QRcodeManager
R8	NotificationsService QueueHandler
R9	NotificationService
R10	StatisticsHandler SuggestionsHandler
R11	StatisticsHandler
R12	QueueHandler TicketManager QRcodeManager

REQUIREMENT	MAPPED COMPONENTS
R13	QueueHandler TicketManager
R14	QueueHandler TicketManager
R15	CostumerMobileApp CostumerHandler
R16	VisitHandler StatisticsHandler TimetableHandler
R17	CostumerHandler
R18	QueueHandler
R19	CostumerHandler
R20	VisitHandler StatisticsHandler
R21	QueueHandler TicketManager
R22	StoreHandler
R23	StoreHandler
R24	VisitHandler TimetableHandler

5 Implementation, Integration and Test Plan

5.1 Overview

Testing is an important practice to make sure that the system behaves as we expect and is able to fulfill the requirements and reach the goals that it is supposed to do. For this reason, the system should be divided in more than a unique block when it is implemented: testing the whole system only when its design has been finished is definitely not a good practice and will lead with every probability to high costs of repair.
The various components of the system need to be checked independently and then gradually integrated one with the other, with other tests that are subsequent to their integration to make sure that the behavior is exactly what it is expected to be. Of course, it is impossible to find all the bugs in our system through testing, as we know that program testing can be used to show their presence, but not their absence.
This is the reason why this part of verification and validation is very important: release an application that is as much bug-free as possible. It is important that the verification and validation phases start as soon as the development of the system begins in order to find errors as quickly as possible.

5.2 Implementation Plan

As a consequence of what has been said above and also taking into account that the CLup system is a relatively small one, then it needs to be implemented, tested and integrated following a bottom-up approach. As in unit testing, drivers must be constructed for each leaf module and then, as the system is built up, they will be replaced by higher level components that could use the lower level subsystems' functionalities. Following this approach, it will be possible to parallelize implementation and the testing procedures.

The external components can be assumed to be already reliable, as they are implemented and used without any specific improvements. So, this assumption can be valid for the GoogleMapsService, the StoreSlidingDoors and QRcodeManager components. The integration with the subsystems that interface with them has to be accurately tested and validated.

The other components and subsystems that should be considered are the same ones that have been described and shown in the Component View section, that are listed below for the sake of reading them more confortably (the last three ones belong to the client side of the system):

- DBMS Services
- Data Manager subsystem
- Access Manager subsystem
- Store Handler
- Customer Handler
- Visit subsystem
- Queue subsystem
- Router
- CustomerMobileApp
- StoreManagerApp
- Ticket Totem

The first components that have to be implemented and tested are the DBMSServices, because they are the ones that allow to keep system's data updated to the last version and they allow to write on the Database, always recalling to make it persistent.

Then we can proceed with the Data Manager subsystem, that directly interfaces from the application server towards the DBMS Services and allows to ask and retrieve data through appropriate queries, or to send new data of new info collected during the users' registrations or their submitted preferences. Once the Data Manager has been completed, then the other subsystems can be built: Access Manager is independent from the other components, so it can be developed in parallel with other tasks. Because of its purpose, the only role that he has is to guarantee that accesses to the system are authenticated, and this function is perfectly isolated from the others.

In fact, when a customer wants to line up, he has only to face and interact with the Visit and Queue subsystems. Data that are necessary for the bookings and for queuing are retrieved from the Data Manager that takes them from the DB, and then inside the components are elaborated and proposed to the users. The two subsystems – Visit and Queue – do not interact with each other because as the more independent they are, the easier would be in the future to repair and modify them without having to handle other components as a cascade unwanted effect. The only exception is made when the Visit subsystem directly interfaces with the Queue subsystem to update the store's queue with the addition of a new customer.

Then comes the turn of the Customer Handler and Store Handler components. As the functionalities to store data of a visit have been already implemented and they should properly work, now all the other type of queries of the customers and the store managers can be managed and processed (e.g. booked visits, real time statistics, temporarily increasing the store capacity, ...), as their information can be already be stored in the DB.

The Router is the last component that is implemented and tested: its only role is to dispatch messages coming from different parts of the system and assures that they arrive to the right subsystem or component. It has only one main function, but it is very important for the correct behavior of the whole application. From the point of view of the client components, they can be implemented in parallel to the application server and once the AS has been completely tested, then the two parts can be merged to see if the whole system complies with the requirements and the overall goals.

Finally, we can list what is the strict sequence of constraint of component implementation that must be fulfilled (the client side components and the Access Manager are not included in the following list because their implementation, as previously said, can be done in parallel with the others):

- DBMS Service
- DataManager
- Queue subsystem
- Visit subsystem
- CustomerHandler and StoreHandler
- Router

5.3 Integration Strategy

As previously said in the chapter above, it's needed to be implemented the system with a **bottom-up approach**. In the following lines there will be a better explanation of how to deal with this procedure, with the different components that will be progressively integrated once they've been tested. Drivers are added on the top of the tree-components building because in this way different requests to apply some functions can be done.

The first components that have to be unit tested after the validation of the DBMSServices are the ones of the Data Manager subsystem. They are the first one that have to be implemented because storing and retrieving data from the DBMS is fundamental for every functionality of the system, that always deals with the most updated state of the system.

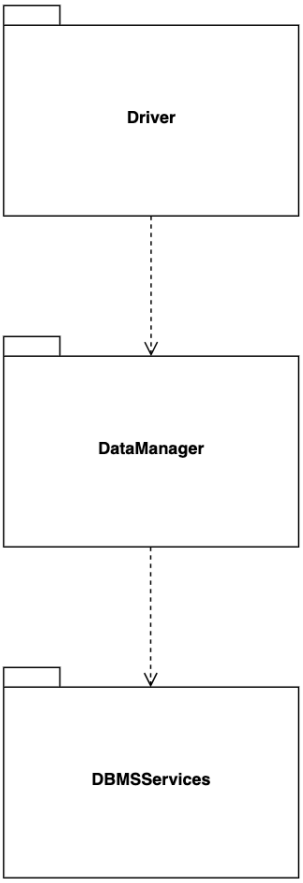


Figure 22

It is then the turn of the other components. The Access Manager subsystem can be tested linking it to the Data Manager and testing its functionalities in parallel with the other components that will be explained below. This can happen because it has no direct interaction with the other components.

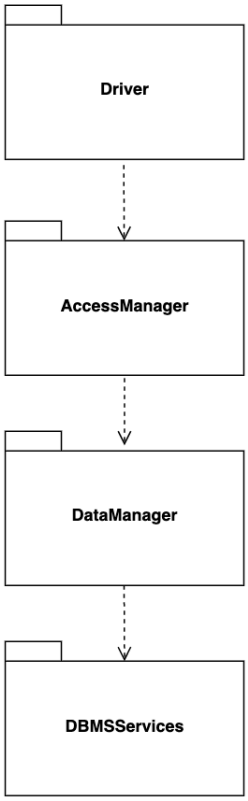


Figure 23

The external components to which the application server interfaces are supposed to be already tested and validated by their production companies, so the Queue subsystem and its components are the next candidates to be taken into consideration for the unit testing. The testers have to make sure that they can correctly interface to the external components too.

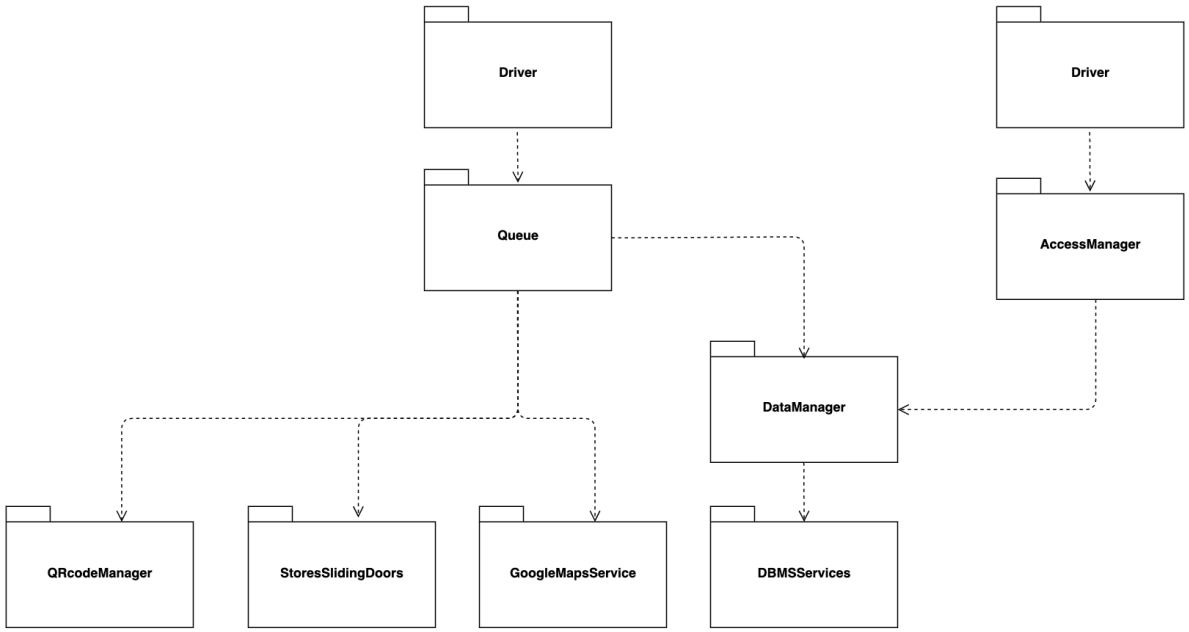


Figure 24

When the Queue subsystem has been determined successfully working, the Visit subsystem is the next one to be treated, as it is directly connected with the previously mentioned one. These tests need to be accurately checked as the lining up of a user and his booking procedure have to be as bug-free as possible to guarantee a proper navigation experience.

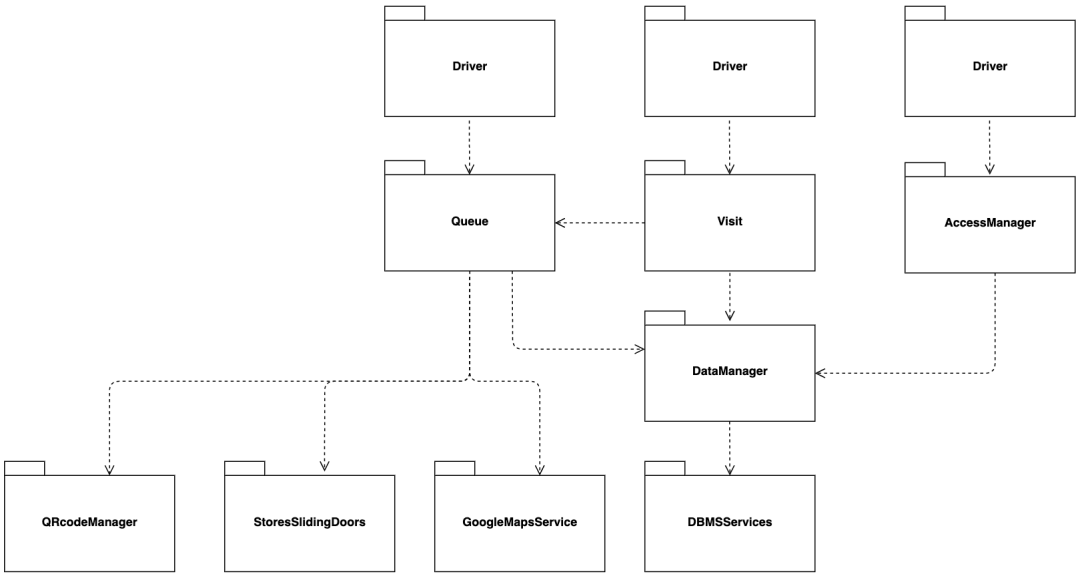


Figure 25

Once that these integrations are made, both the Customer Handler and Store Handler components can be unit tested and integrated. Notice that also here the integration with the GoogleMaps Service needs to be made, as the external component is assumed to be perfectly working as we were doing in the previous step. Drivers are properly added to make sure that the step by step integration can be made without any

problem, fulfilling all the possible functionalities that a component needs to be ready to deal with.

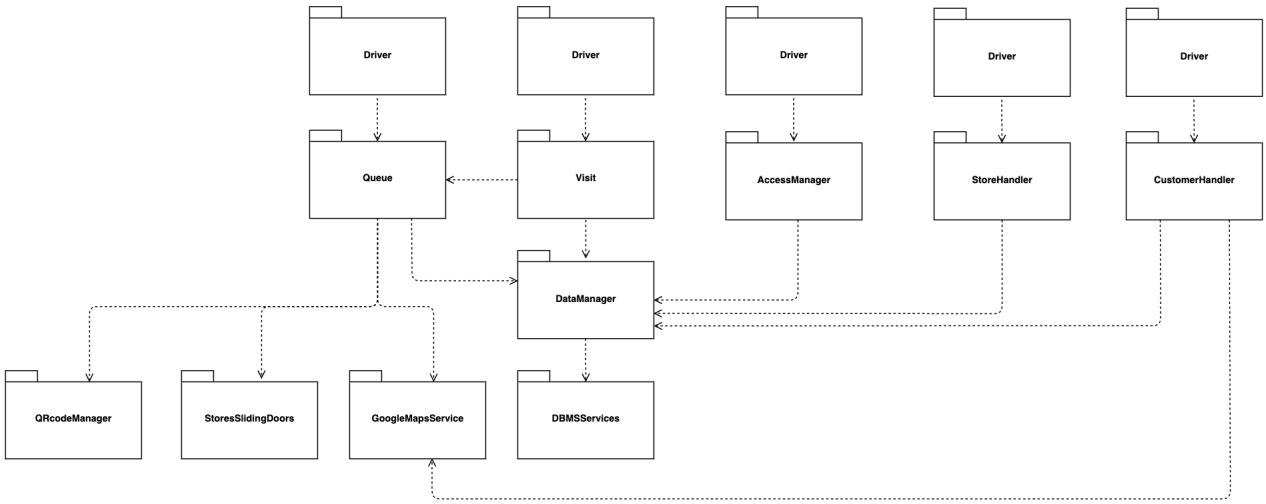


Figure 26

To complete the Application Manager the last component that has to be implemented, unit-tested and then added is the Router, that substitutes all the drivers that were drawn in the previous diagram. It allows to make possible the flow of the called methods from the client to the server and dispatch them to the right component.

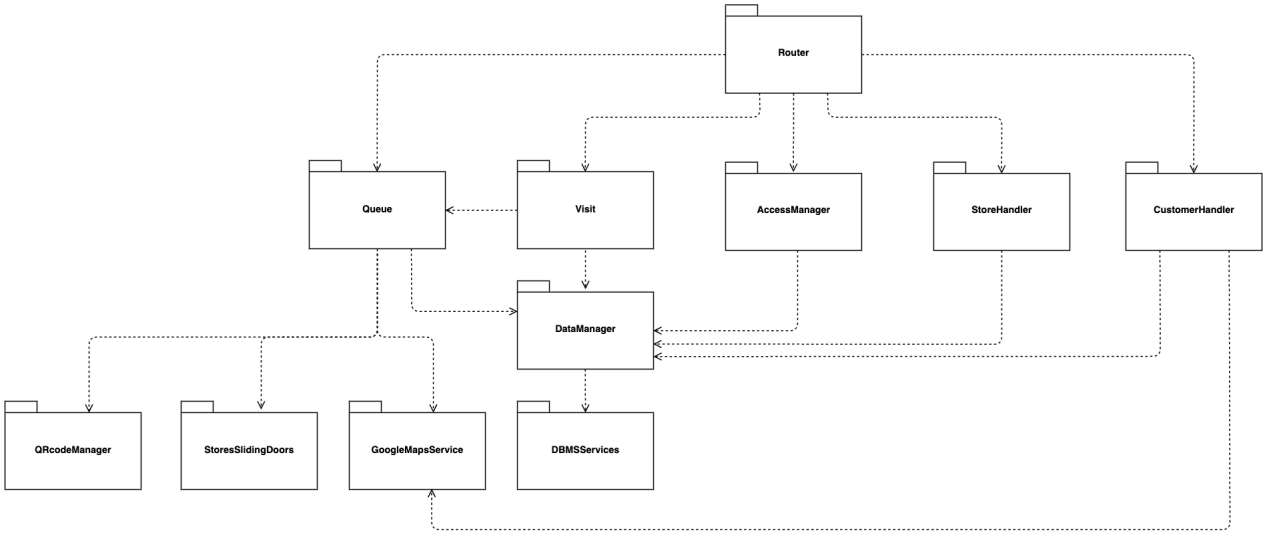


Figure 27

In conclusion, Client side components, which can be implemented in parallel to the application server ones, are unit tested and integrated to the whole application system. In this way, a final testing session can be performed to validate the overall system.

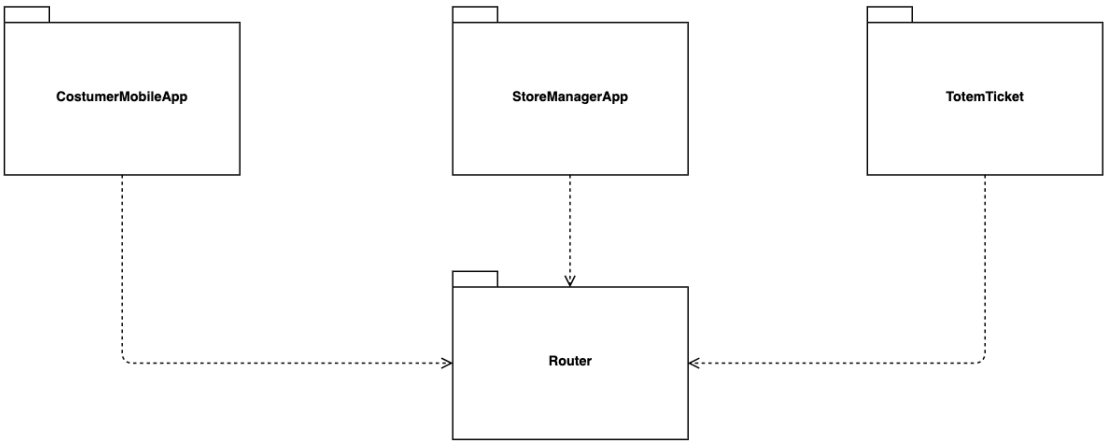


Figure 28

5.4 System Testing

As the integration of the various components of the system has been managed, the whole system testing concerns the check of functional and non-functional requirements, where the testing environment should be as close as possible to the production environment.
Functional testing, as the name suggests, verifies the satisfaction of the functional requirements of the system. It's a good indicator for the correct implementation of the system.
Moreover, the **Load testing** and **Performance testing** have a main role in this system: memory leaks, mismanagement of memory and the identification of the upper limits of the components can be made by increasing the load of the system for a long period; what's more, identifying bottlenecks affecting response time, utilization and throughput makes possible to find inefficient algorithms, optimize queries and if there are any hardware/network issues.
Stress testing allows to assure that the system recovers correctly after failure.

6 Effort spent

Antonio Ercolani:

Purpose and Document Structure	**1h**
User Interface Design	**10h**
Component view discussion	**3h**
Component view	**3h**
Deployment view	**4h**
Sequence diagrams	**6h**
Requirement traceability discussion	**1h**

Vittorio Fabris:

Scope and organization of chapter 1	**1,5h**
Architectural Design Overview	**3h**
Component view discussion	**3h**
Component View	**8h**
Validation and Testing	**12h**
Review Meetings	**5h**

Riccardo Nannini:

Component view discussion	**3h**
Component view	**3h**
Component interfaces	**3,5h**
UML model	**2h**
Runtime view	**4,5h**
Algorithms and data structures	**4,5h**

7 References