POLITECNICO DI MILANO

Computer Science and Engineering

Design Document

Customers Line-up

Software Engineering 2 Project

Academic year 2020 - 2021

22 December 2020

Version 1.0

Authors:
Samuele NEGRINI
Giorgio PIAZZA

Professor:
Matteo Giovanni ROSSI

Revision history

Date	Revision	Notes
22/12/2020	v.1.0	First release.

Contents

Chapter 1

Introduction

1.1 Purpose

This document contains the complete design description of the *Customers Line-up* system. This includes the architectural features of the system and the details of what operations each module will perform. It also shows how the requirements and use cases detailed in the *CLup, Requirement Analysis and Specification Document* will be implemented in the system.

The primary audiences of this document are the software developers and testers.

1.2 Scope

Customer Line-up (CLup) is an **easy-to-use** application which aims to settle for various queuing problems faced by supermarkets and their customers.
On the one side, it allows store managers to regulate the influx of people in the building and, on the other side, it saves people from having to line up and stand outside of stores for hours on end.

Customers can enter a queue in *real-time* by **taking a ticket** via different channels such as Self Service Ticketing Kiosk and a Mobile App called Customer App. During this process, the user is given an estimation of the waiting time and the **leave-at-time** (i.e. the time they need to depart from their current position to reach the store). This ticket comprehends a queue number, which identify user's position in the queue, and a QR code, which is used for the ticket validation.
The validation process will be performed by a store employee using a dedicated application (Store App), which will allow them to scan QR codes and submit data to the CLup Server.

The Web App, accessible through any modern web browser, will provide store managers a dashboard from which they can monitor customers flow and check the journey map of all the clients inside the store at a given time.

The Customer App also support an *advanced functionality* where a customer can **book a visit** to the store by indicating the approximate expected duration of the trip and the main categories of items they intend to buy. For long-term customers, it suggests a time inferred by the system based on an analysis of the previous visits.

This application works 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 *legacy system* will be completely superseded by the application. Indeed, its effectiveness is strictly bound to the number of users who use it.

1.3 Glossary

1.3.1 Definitions

Term	Definition
Customers	Identifies the store customers.
Employees	Used in this document to mean both entrance-staff and cashiers.
Store Pass	General term that comprehends both tickets and bookings.
Ticket	Pass generated from the system which is comprehensive of the Queue number and QR code.
Booking, Reservation	Pass generated from the system as a result of the reservation process.
Queue number	Identify user's position in the queue.
QR code	Type of matrix barcode, used by the system for the ticket validation.
System	Totality of the hardware/software applications that contribute to provide the service concerned. Also referred as CLup, Application, Platform.
Legacy System	Used in this document to mean the physical ticketing system where you retrieve a ticket from a stand.

1.3.2 Acronyms

Acronyms	Term
CLup	Customers Line-up
RASD	Requirements Analysis and Specification Document
QR	Quick Response
GPS	Global Positioning System
UI	User Interface
API	Application Programming Interface
OS	Operating System
REST	REpresentational State Transfer
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
DB	DataBase
DBMS	DataBase Management System

1.3.3 Abbreviations

Abbreviations	Term
e.g.	Exempli gratia
i.e.	Id est
R	Requirement

1.4 Reference documents

- Project assignment specification document.
- CLup, Requirements Analysis and Specification Document.
- Course slides on beep.

1.5 Document structure

This document is presented as it follows:

1. **Introduction**: the purpose of this document along with an overall description of the main functionalities of the system.
2. **Architectural Design**: high level overview of how responsibilities of the system are partitioned and assigned to subsystems. It identifies each high level subsystem and the roles assigned to them. Moreover, it describes how these subsystems collaborate with each other in order to achieve the desired functionality.
3. **User Interface Design**: it describes the functionality of the system from the user's perspective. It shows how the user will be able to interact with the system to complete all the expected features and the feedback information that will be displayed to the user.
4. **Requirements Traceability**: it provides a cross-reference that traces components to the requirements contained in RASD document. A tabular format is used to show which system components satisfy each of the functional requirements from the RASD.
5. **Implementation, Integration and Test Plan**: an explanation of how the implementation, integration and test plan will be carried out and the technologies to be used.
6. **Effort Spent**: keeps track of the time spent to complete this document. The first table defines the hours spent as a team for taking the most important decisions, the seconds contain the individual hours.

Chapter 2

Architectural Design

2.1 Overview

The system will be developed from scratch and it will completely replace the legacy system.

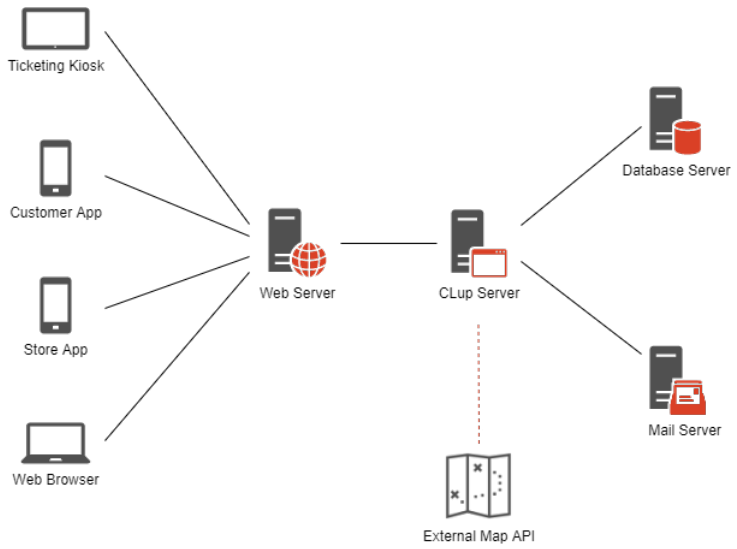

Figure 2.1: CLup system diagram.

It is composed by a client part and a server part:

- **Server side:**
  - *Application Server (CLup Server):* server where all the logic is located. It communicates with other servers and is the central point of the system.
  - *Web Server:* server used for the communication with the clients.
  - *Database Server:* server where all data are stored.
  - *Mail Server:* server used to send confirmation emails about the bookings.
  - *External map API:* API used to retrieve data about the distance of the user from the store. This information will be used to inform the user of when leave the current place.

- **Client side:**
  - *Customer App:* application installed on customers smartphone. It allows to retrieve a ticket or book a visit.
  - *Store App:* application installed on employees smartphone. It allows to validate store passes.
  - *Ticketing Kiosk:* tablet installed at the entrance of the store to which a printer is attached. It has installed a modified version of the Customer App which is able to print the ticket on place. The application is bounded to the store it is installed at. It works exactly as a normal Customer App, it allows you to take tickets only for bounded store but not to make reservations.
    This component will be omitted in the next sections because, for the functions it provides, it behaves just as a Customer App.
  - *Web Browser:* used by store employees and managers to access the web dashboard.

2.2 Component view

2.2.1 High Level

The following component diagram highlights all the components of the system and describes both his internal and external interactions. A general overview of the whole system is shown in Figure 2.2. Further details will be provided in the subsequent diagrams.

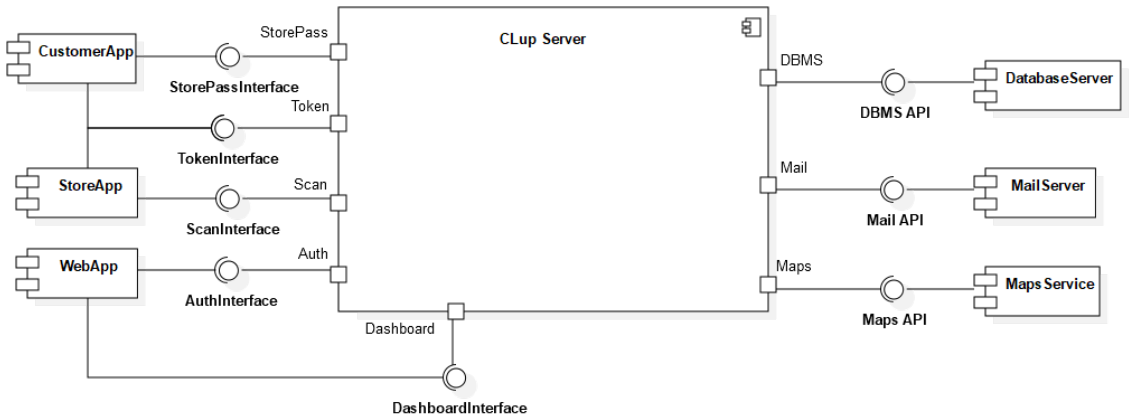

Figure 2.2: *High level* component diagram.

The components in Figure 2.2 are:

- **CLup Server**: contains the business logic of the entire system. This component offers to both customers and supermarkets the features of *CLup*. On the one side, the interfaces provided by this components allow the client side (mobile apps and web app) to communicate with the server. On the other side, it interfaces with external systems such as the database server, the email server and the maps service. Further detailed in Figure 2.3.

- **Customer App**: represents the mobile application installed on the customers' devices. Upon authentication via the **Token Interface**, it allows customers to access the store pass functions of *CLup* by using the **StorePass Interface**. Periodically, it asks the server for information on store passes and distance from stores. In case of updates, for example a store pass is deleted or it's time to leave, it shows a notification to the user.
- **Store App**: represents the mobile application installed on the devices of the store. Upon authentication with credentials via the **Token Interface**, it allows the store employees to access the scan functions of *CLup* by using the **Scan Interface**.
- **Web App**: represents the web application reachable by any modern web browser by the store managers. Upon authentication with credentials via the **Auth Interface**:
  - allows the store managers to access the dashboard functions of *CLup* by using the **Dashboard Interface**;
  - allows CLup admins to register and delete stores from the system by using the **Dashboard Interface**.

  Furthermore, it offers to the customers the landing pages of both the *confirmation* and *deletion links* required by the booking process via the **Dashboard Interface**.
- **Database Server**: provides the interface to the *CLup Server* to deal with the data management process.
- **Mail Server**: provides the interface to the *CLup Server* to deal with the email sending and receiving services needed. The email system is necessary for both the process of registration of stores and the validation of the customers' booking.
- **Maps Service**: provides the interface to the *CLup Server* to deal with the maps data. It is required to retrieve the best possible path between two positions and to calculate the *leave-at-time*.
  The service is Google Distance Matrix API and provides travel distance and time for a matrix of origins and destinations, based on the recommended route between start and end points.

2.2.2 CLup Server

The following component diagrams describe the internal structure of the application server, which contains the business logic of the system.

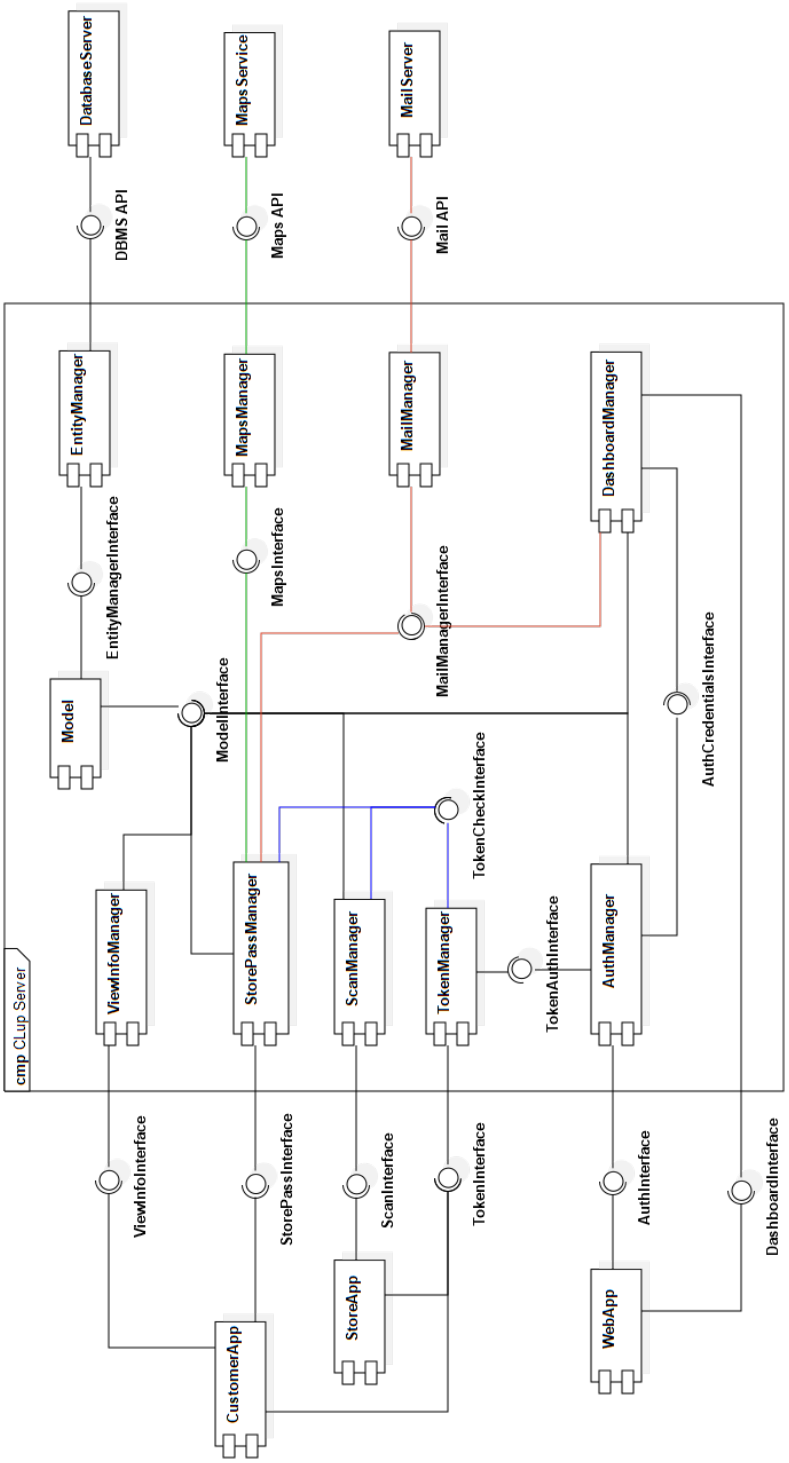

Figure 2.3: *CLup Server* component diagram.

The components in Figure 2.3 are:

- **StorePassManager**: handles the main feature offered to the customers to line up at stores. In particular, it manages all the logic of store passes life-cycle, from their creation to expiration and deletion. Further detailed in Figure 2.4.
- **ScanManager**: handles the main feature offered to the store employees. In particular, it manages all the logic of store passes scan and validation. Further detailed in Figure 2.6.
- **TokenManager**: deals with token generation, decoding and verification for the mobile applications. It interfaces with the AuthManager through the **TokenAuth Interface** to integrate the authentication when using credentials. It is based on the JWT an internet standard for creating JSON token with a payload. Further details are provided in section 5.2.
- **AuthManager**: deals with the authentication of CLup admins, store managers and employees. Allows the generation of store credentials. Further detailed in Figure 2.5.
- **DashboardManager**: handles the main features offered through the WebApp. In particular, it offers a dashboard that allows:
  - **Store managers**:
    - to monitor the status of their store;
    - to change the maximum number of people inside the store;
    - to manage the bookings of the customers.
  - **Store employees**: to view the next store passes that will be called.
  - **CLup admins**: to manage the creation and deletion of stores. It interfaces with the MailManager to send the credentials to the store PEC address.
  - **Customers**: to confirm and delete their bookings via link.
- **Model**: high-level component which represent the data on the server and acts as a mask to the database server. Since it is the entry point to the data, almost every component needs to interface with this one.
- **EntityManager**: deals with all the management of the data needed by the system. It perform object-relational mapping to the **Model** component and uses the methods provided by the **DBMS API** to execute queries on the database.
- **MailManager**: interfaces with the Mail API to send emails.
- **MapsManager**: deals with the maps data. It retrieves the best possible path between two positions and to calculate the *leave-at-time*.

2.2.3 Store Pass Manager

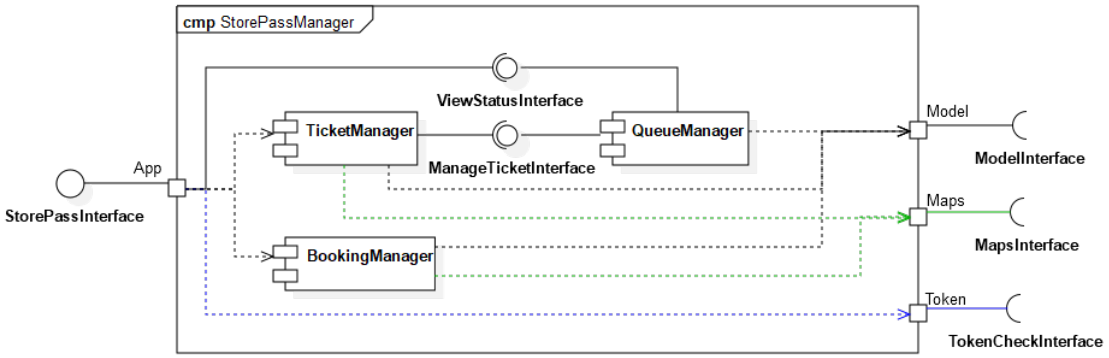

Figure 2.4: *Store Pass Manager* component diagram.

This component depends from the **TokenAuthInterface** because it is necessary to check the validity of the auth token. The components in Figure 2.4 are:

- **TicketManager**: handles the tickets creation for a specific store. Provides methods to the Customer App to update the current status of the queue and through the **MapsInterface** can access the maps data to provide the leave-at-time.
- **BookingManager**: handles the booking creation for a specific store. It calculates the expected booking time for a customer, based on an analysis of the previous visits. It provides methods to the Customer App to update the current status of the queue and through the **MapsInterface** can access the maps data to provide the leave-at-time.
- **QueueManager**: deals with the queue management for each stores subscribed and calculates the estimated waiting time in queue. It offers an interface to the *Ticket Manager* which allows it to add and remove tickets from the queues. Furthermore, the **ViewStatusInterface** provides the current status of the queue.

2.2.4 Auth Manager

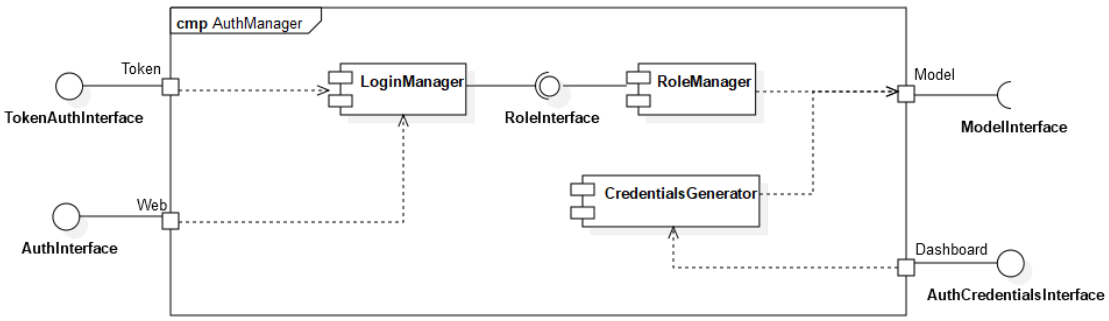

Figure 2.5: *Auth Manager* component diagram.

The components in Figure 2.5 are:

- **LoginManager**: handles the login and credentials check of CLup admins, store managers and store employees in the system. It interfaces with the RoleManager through the **RoleInterface** to give the right permissions to the logged user.
- **RoleManager**: handles the role based access control (RBAC): an authorization scheme that grants access rights based on the role of the use. In particular, this components grants user authentication and authorization:
  - Authentication: request and verify the identity of CLup admin, store managers and employees attempting to login using a usercode and password;
  - Authorization: verify the permissions of the logged user to perform any requested action (e.g. adding or removing a store, creating a new item category, etc.) before performing it.
- **CredentialsGenerator**: handles the credentials generation through the **AuthCredentialsInterface** when a CLup Admin adds a new store to the system. In the event that the CLup admin request new credentials for stores, this component will generate new ones.

2.2.5 Scan Manager

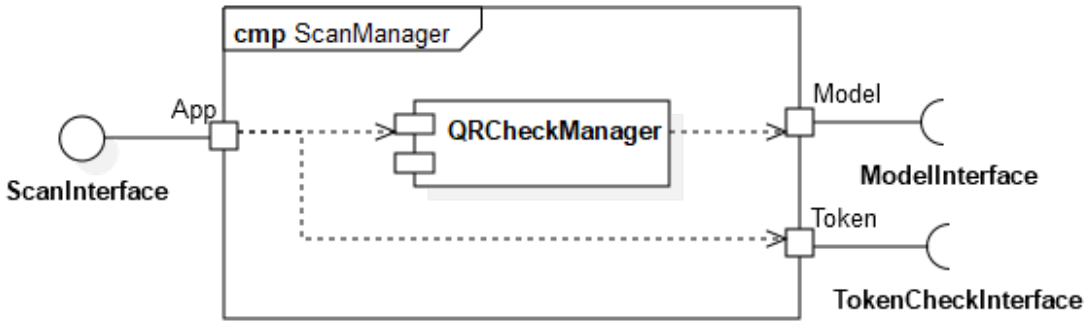

Figure 2.6: *Scan Manager* component diagram.

This component depends from the **TokenAuthInterface** because it is necessary to check the validity of the auth token. The components in Figure 2.6 are:

- **QRCheckManager**: it handles the scanning and validation process of QR codes. The scanned code validity will be checked versus the one stored in the Model via the ModelInterface.

2.3 Deployment view

The system presents a **three-tier architecture** as shown in Figure 2.7. This is a well-established software application architecture that organizes applications into three logical and physical computing tiers. The chief benefit of three-tier architecture is that because each tier runs on its own infrastructure, each tier can be developed simultaneously by separate developers and can be updated or scaled as needed without impacting the other tiers.

- **Tier 1: presentation tier**. It is the front end layer in the three-tier system and consists of the user interface. It communicates with others layers via API calls on the HTTPS protocol.
  The Customer App is installed on compatible Android or iOS phones. Distribution of the appropriate executable will be handled by the corresponding app stores. For the Local Ticketing Kiosk, a custom non-public executable will be installed on an Android tablet.
  The Store App is installed on Android or iOS phones and will not be publicly available.
  Finally, the Web App will be accessible through web browser by connecting to a dedicated website.
- **Tier 2: business logic tier**. It contains the functional business logic which drives the application's core capabilities. Multiple client components can access the this tier simultaneously, so this layer must manage its own transactions.
  The CLup Server App is installed on a dedicated server with a running instance of Apache TomEE 8.
- **Tier 3: data management tier**. It comprises of the database/data storage system and data access layer. Data is stored in a persistent way into a MySQL database and accessed by the business logic tier via API calls.

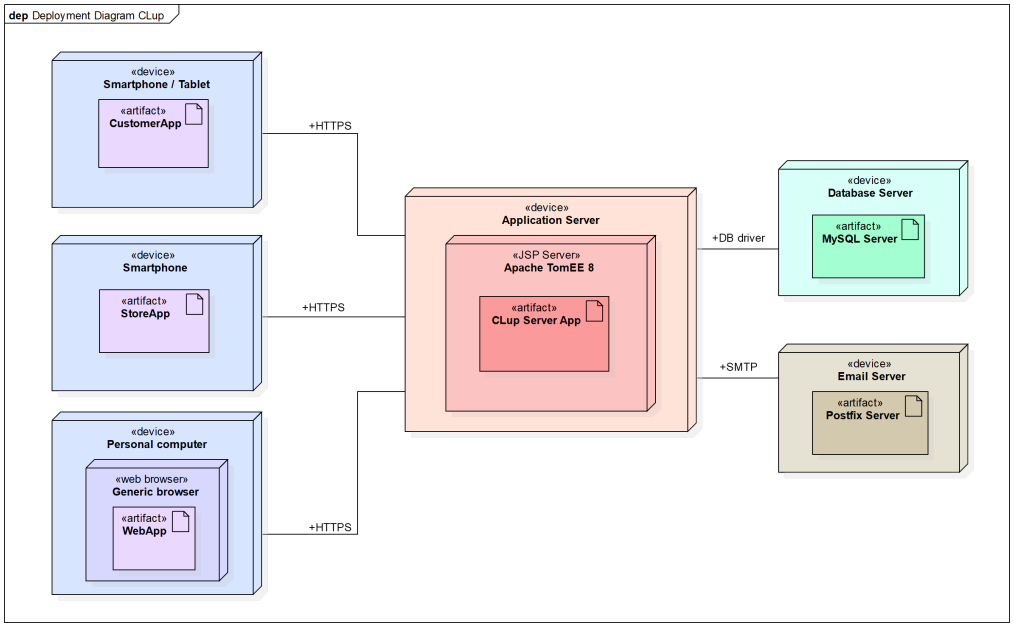

Figure 2.7: *Deployment diagram* of the CLup system.

2.4 Runtime view

This section describes the most important components interactions of the system.
For the sake of simplicity the sequence diagrams are based on the first level of components.

2.4.1 Customer App token

At the startup, the Customer App will try to request a token to the server via the token manager. Having a token is mandatory in order to take any action with the app.
This process is really important because the token is the only way to identify a customer since no registration is expected. The app will reiterate the process till a valid token is gotten.
This is an automatic process and no action is performed by the user during the operation.

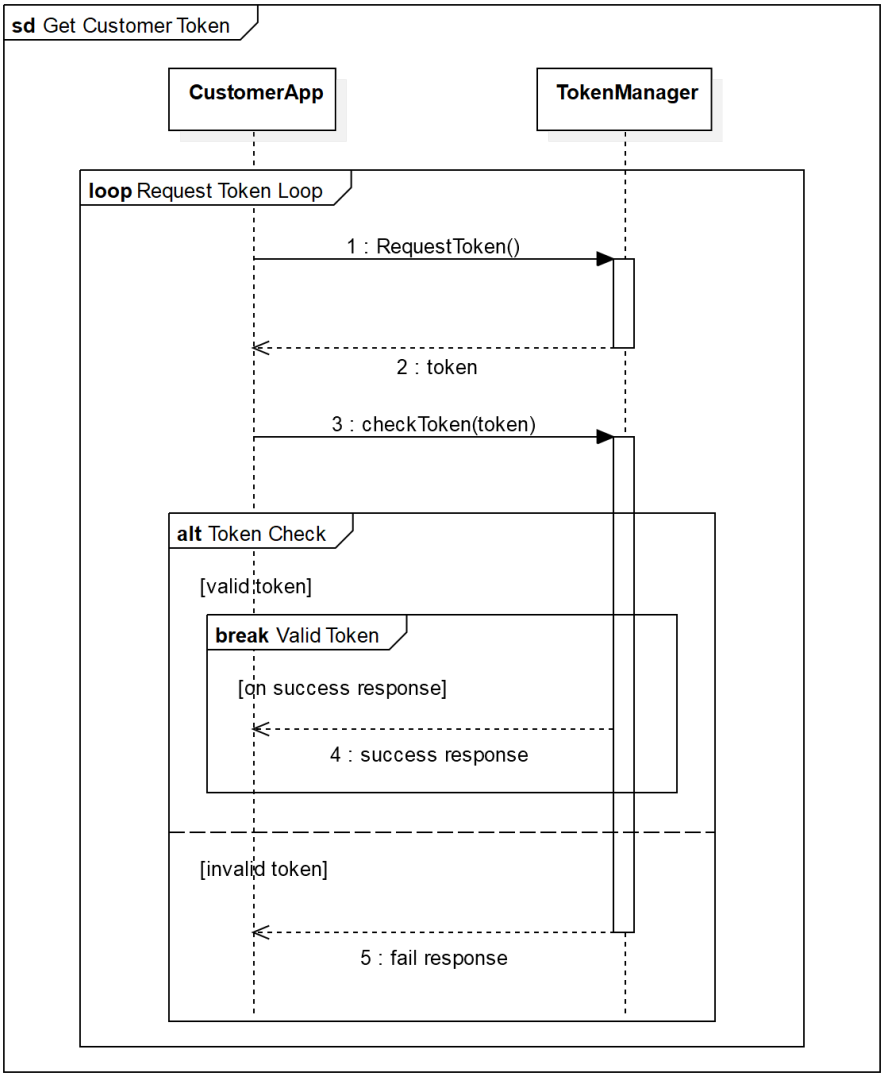

Figure 2.8: Customer app token request.

2.4.2 Ticket request

The request for a ticket starts from the Customer App and contains both token and selected store. The token validation is the first operation performed by the server and consequently, if all the checks go well, a ticket is retrieved.

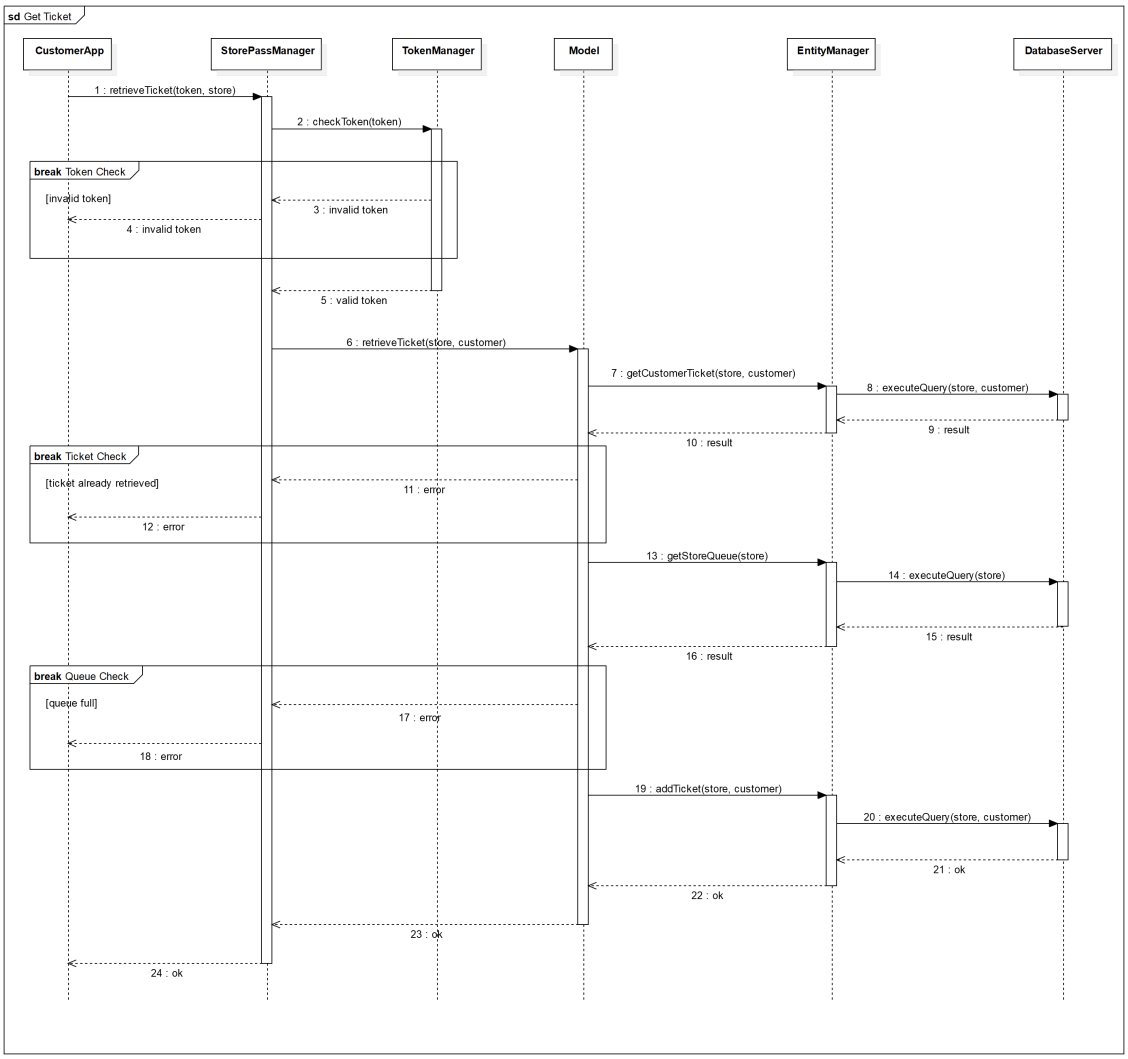

Figure 2.9: Ticket request.

2.4.3 Booking request

A user can book a visit for a store if he hasn't booked a visit to the same store and in the same day yet. This check is done together with the token validation and others checks.
Since a booking must be confirmed via a link, an email is sent at the end of the whole process through the mail server.

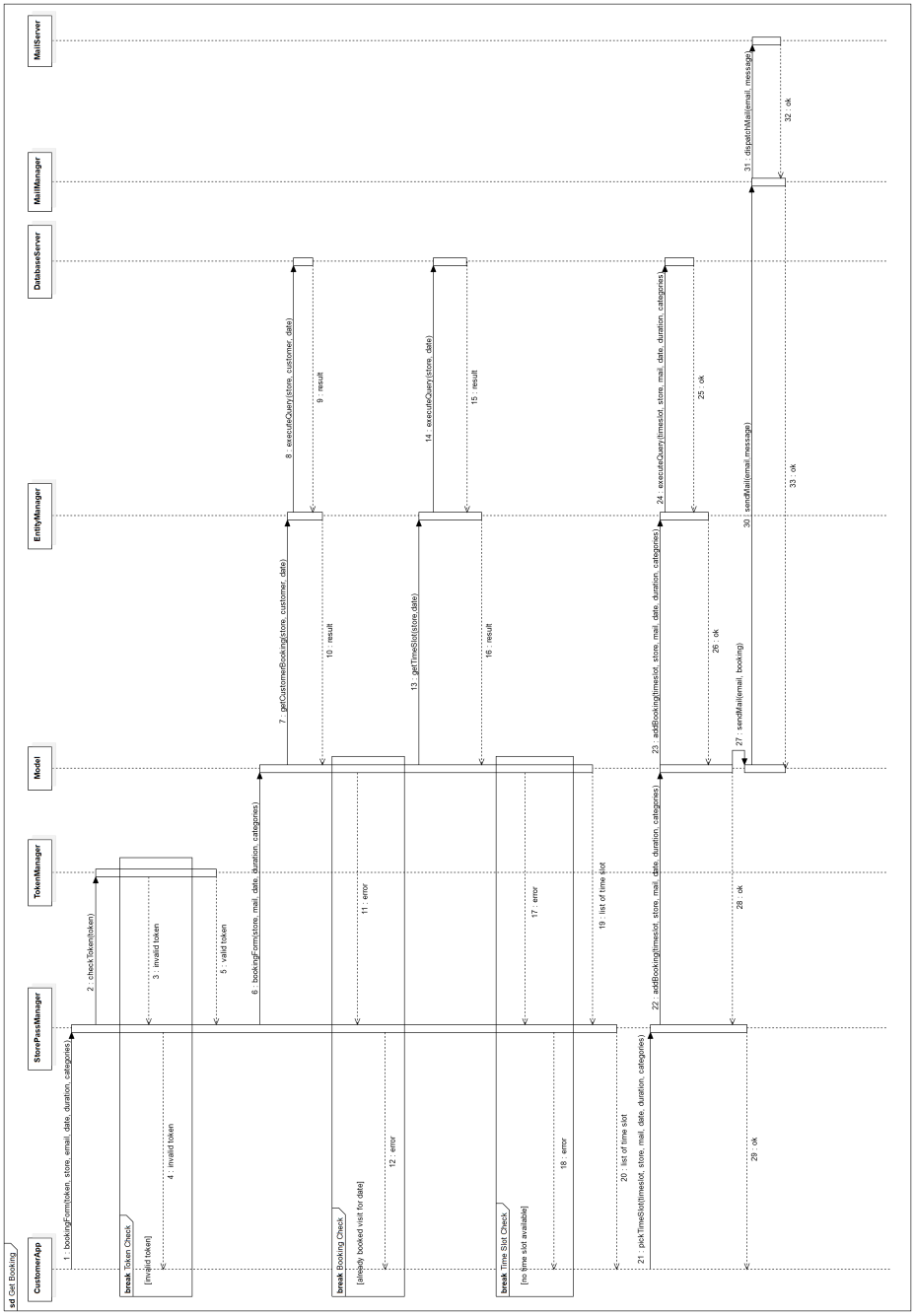

Figure 2.10: Booking request.

2.4.4 Leave At Time

The Customer App reminds the user to leave the place where he is in order to ensure that he arrives on time.
The server fetches all the valid store passes of the day and computes the distance of the customer from the respective store. If the customer should leave the place where he is, a notification is sent to him. For the sake of simplicity the token validation process is omitted.

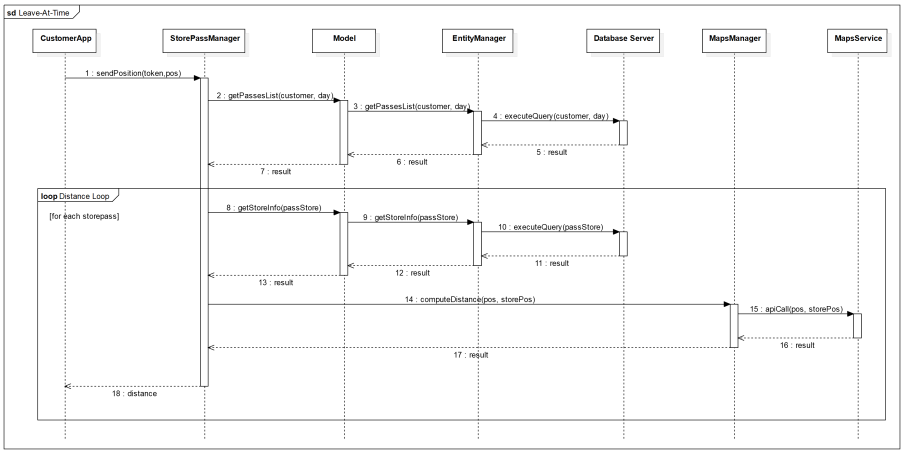

Figure 2.11: Leave at time function.

2.4.5 Web App login

The following sequence diagram represents the login process to the Web App. CLup admins, store managers and store employees will authenticate to the system by submitting their credentials via a form.

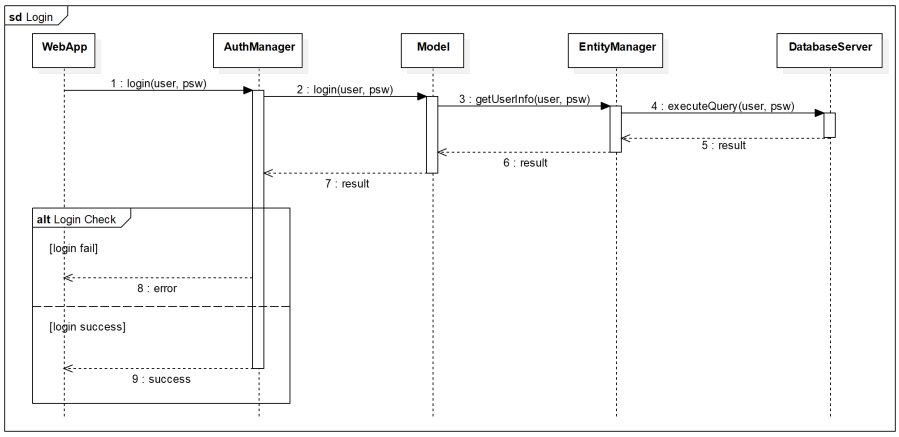

Figure 2.12: Web App login.

2.4.6 Check store status

After logging in, the store managers can see the store status, for instance the visits booked. The following sequence diagram represents this process.

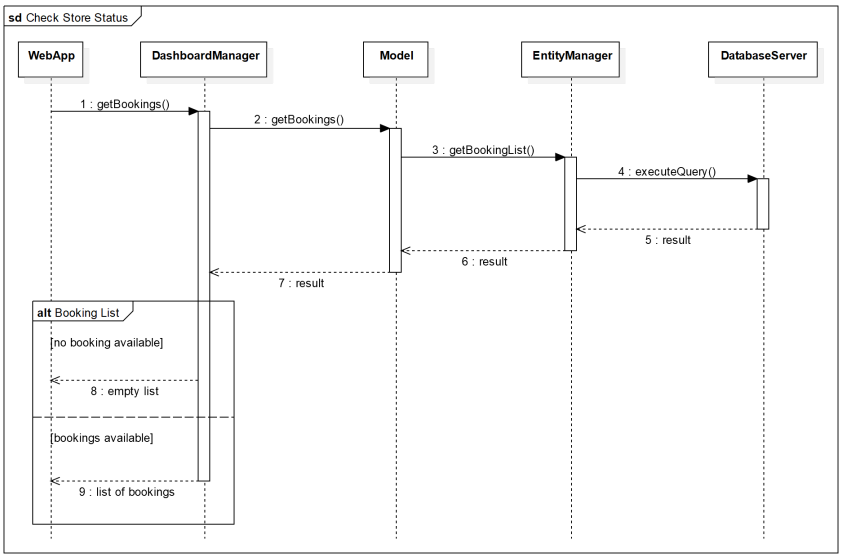

Figure 2.13: Check store status.

2.4.7 Store App login

As the Customer App, the Store App also has a token to operate. This time there is a login process before the generation of the token.

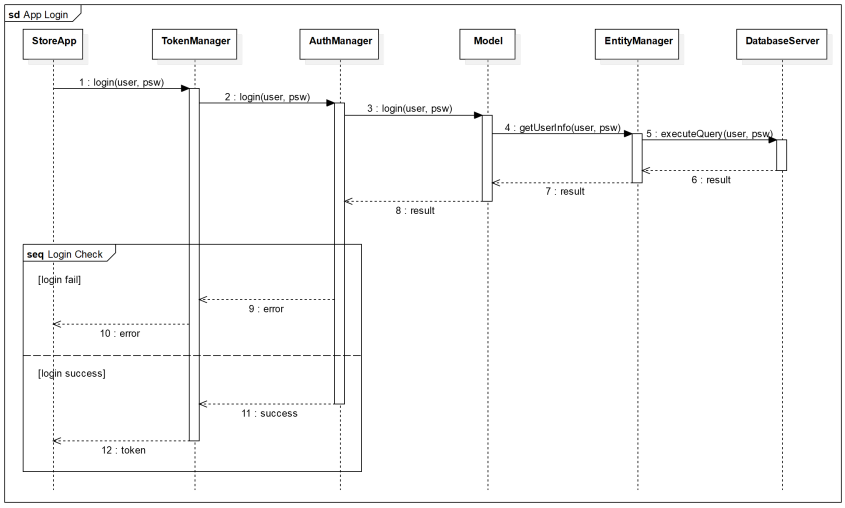

Figure 2.14: Store app login.

2.4.8 QR Code scan

The scan of the QR is the most important part of the Store App. After the token validation, the QR code is validated as the sequence diagram shows. If the pass is valid, his status is updated.

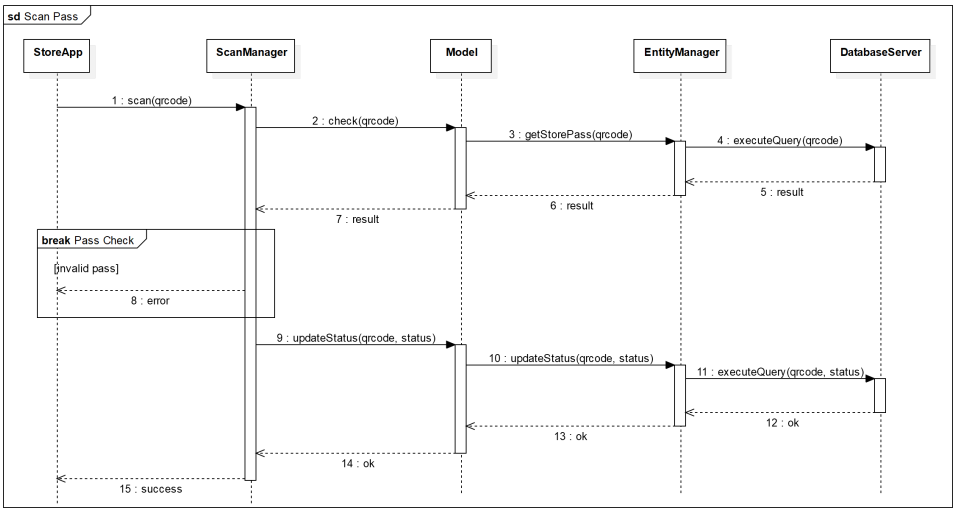

Figure 2.15: Store app scan process.

2.4.9 Store registration

The store registration is performed by a CLup admin via the Web App after his login. Prompted information are checked from the system and then, if the store doesn't already exist, the store is saved into the database. After that, the credentials for the store managers and store employees are generated and sent to the store PEC address.

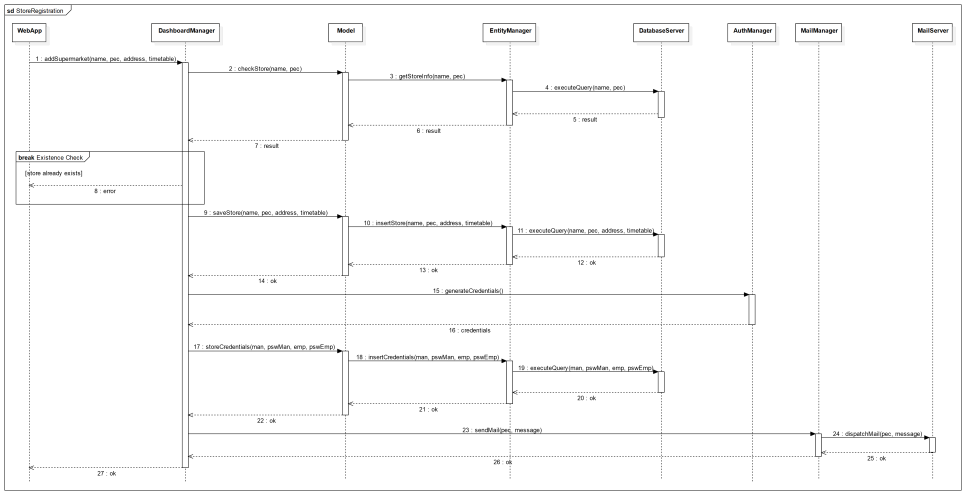

Figure 2.16: Store registration.

2.5 Component interfaces

This section lists all the methods that each component interface provides to the other components.

- **ViewInfoInterface**
  - getStores(token, filters)
  - getStoreInfo(token, store)
- **StorePassInterface**
  - retrieveTicket(token, store)
  - bookingForm(token, store, email, date, duration, categories)
  - pickTimeSlot(timeslot, store, mail, date, duration, categories)
  - getPassesList(token)
  - getPassInfo(token, pass)
  - deletePass(token, pass)
- **ScanInterface**
  - scan(qrcode)
- **TokenInterface**
  - requestToken()
  - checkToken(token)
  - login(user, psw)
- **AuthInterface**
  - login(user, psw)
- **DashboardInterface**
  - getStoreStatus()
  - getTickets()
  - getBookings()
  - deletePass(pass)
  - updateStoreInfo(info)
  - addSupermarket(name, pec, address, timetable)
  - regenerateCredentials(store)
  - deleteBooking(delete_code)
  - confirmBooking(confirm_code)

- **TokenAuthInterface**
  - login(user, psw)
- **AuthCredentialsInterface**
  - generateCredentials()
- **TokenCheckInterface**
  - checkToken(token)
- **MailManagerInterface**
  - sendMail(email, message)
- **MapsInterface**
  - computeDistance(pos, storePos)
- **ModelInterface**
  - retrieveTicket(store, customer)
  - bookingForm(store, mail, date, duration, categories)
  - addBooking(timeslot, store, mail, date, duration, categories)
  - login(user, psw)
  - getBookings()
  - scan(qrcode)
  - updateStatus(qrcode, status)
  - checkStore(name, pec)
  - saveStore(name, pec, address, timetable)
  - storeCredentials(man, pswMan, emp, pswEmp)
  - updateCredentials(store, man, pswMan, emp, pswEmp)
  - getStores(filters)
  - getStoreInfo(store)
  - getPassesList(customer)
  - getPassesList(customer, day)
  - getPassInfo(pass)
  - deletePass(pass)
  - getStoreStatus()
  - getTickets()
  - deletePass(pass)
  - updateStoreInfo(info)
  - confirmBooking(booking)

- **EntityManagerInterface**
  - getCustomerTicket(store, customer)
  - getStoreQueue(store)
  - addTicket(store, customer)
  - getCustomerBooking(store, customer, date)
  - getTimeSlot(store, date)
  - addBooking(timeslot, store, mail, date, duration, categories)
  - getAllStorePasses(store)
  - getUserInfo(user, psw)
  - getBookingList(store)
  - getStorePass(qrcode)
  - updateStatus(qrcode, status)
  - getStoreInfo(name, pec)
  - insertStore(name, pec, address, timetable)
  - insertCredentials(man, pswMan, emp, pswEmp)
  - updateCredentials(store, man, pswMan, emp, pswEmp)
  - getStores(filters)
  - getStoreInfo(store)
  - getPassesList(customer)
  - getPassesList(customer, day)
  - getPassInfo(pass)
  - deletePass(pass)
  - getStoreStatus(store)
  - getTickets(store)
  - deletePass(pass)
  - updateStoreInfo(store, info)
  - confirmBooking(booking)

2.6 Selected architectural styles and patterns

- **Three-tiers architecture**: as stated in section 2.3, the CLup system adopt this well-established software application architecture.
- **Client-Server architecture**: in this architecture, clients and servers are separate logical objects that communicate over a network to perform tasks together. A client makes a request for a service and receives a reply to that request; a server receives and processes a request, and sends back the required response.

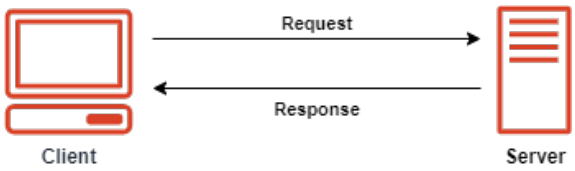

Figure 2.17: Client-Server architecture.

- **REST API**: Representational State Transfer (REST) is a set of commands commonly used with web transactions. It allows the usage of HTTP to obtain data and generate operations on those data in all possible formats, such as XML and JSON.
- **Model-View-Controller**: software design pattern that divides the program logic into three interconnected elements.
  - Model: central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
  - View: any visual representation of such as a chart, diagram or table.
  - Controller: responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

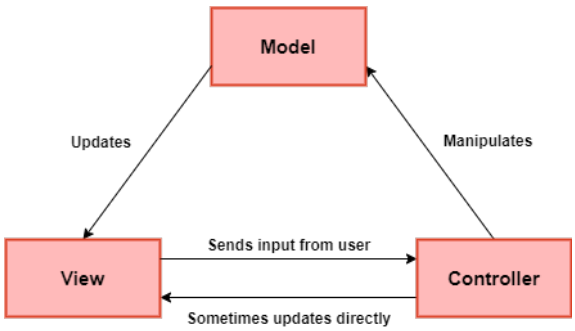

Figure 2.18: Model-View-Controller.

2.7 Other design decisions

2.7.1 Database Structure

As already mentioned all the data will be stored in a database. The chosen DBMS is MySQL since it is an **open-source relational database management system** (**RDBMS**).
The following diagram explains how the database is structured.

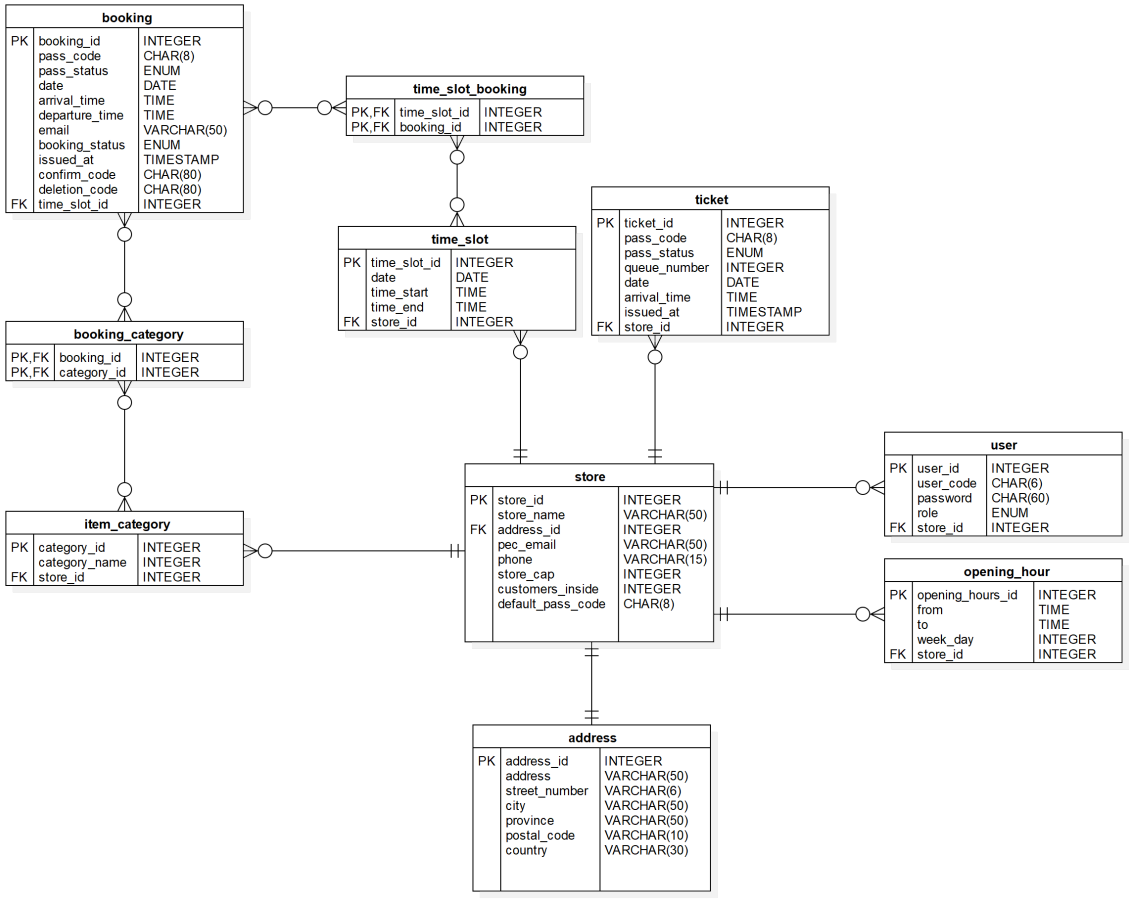

Figure 2.19: Database structure.

Chapter 3

User Interface Design

3.1 Mockups

The Customer App is the interface which permit customers to enjoy CLup services. Whether installed on a smartphone or on a ticketing kiosk, it is the only way for a customer to use CLup. User interface mockups of most important pages of the app are shown below.

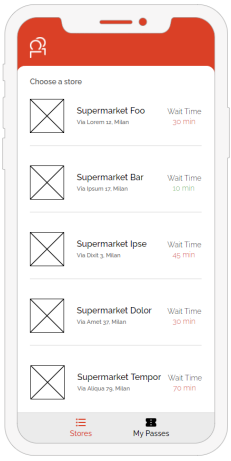

Figure 3.1: App home.

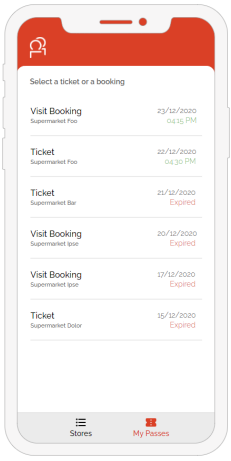

Figure 3.2: Tickets list.

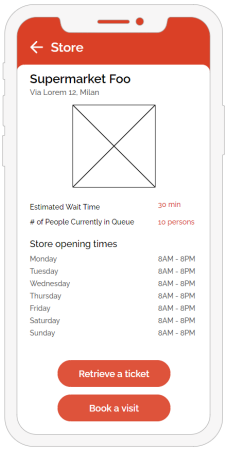

Figure 3.3: Store page.

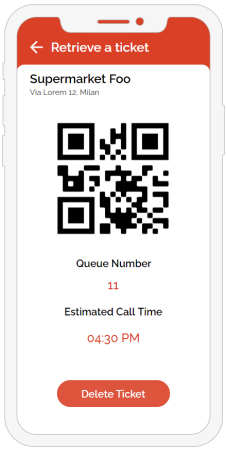

Figure 3.4: Ticket Retrieved.

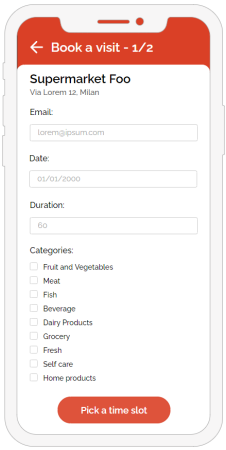

Figure 3.5: Book a visit (1/2).

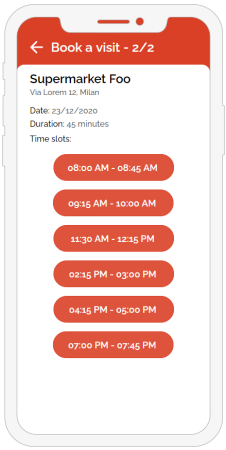

Figure 3.6: Book a visit (2/2).

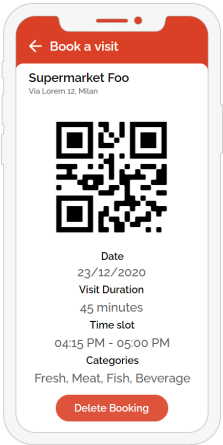

Figure 3.7: Visit booked.

The Store App is the interface used by employees to scan store pass QRs. User interface mockups of most important pages of the app are shown below.

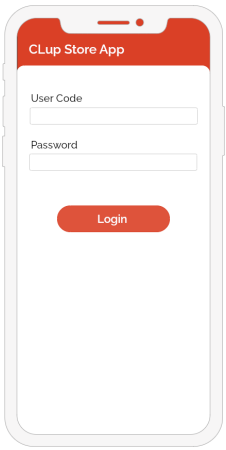

Figure 3.8: Login page.

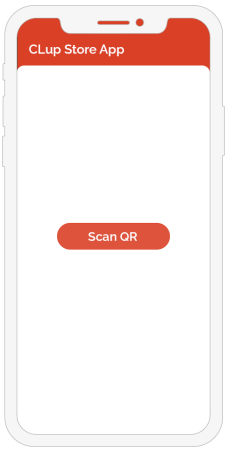

Figure 3.9: Home page.

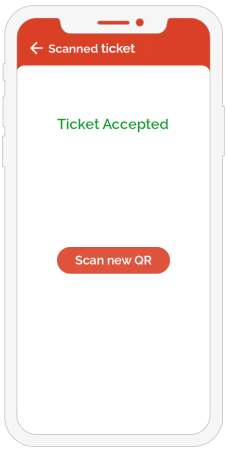

Figure 3.10: Ticket Scanned (accepted).

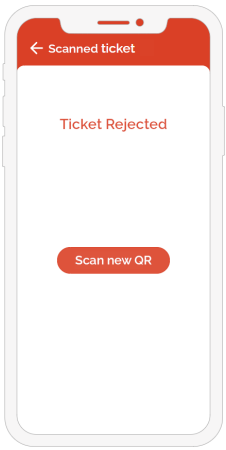

Figure 3.11: Ticket Scanned (rejected).

Store managers and store employees use a web based interface for enjoying CLup features. The CLup admins can login at the same interface as well. User interface mockups of most important pages of the web dashboard are shown below.

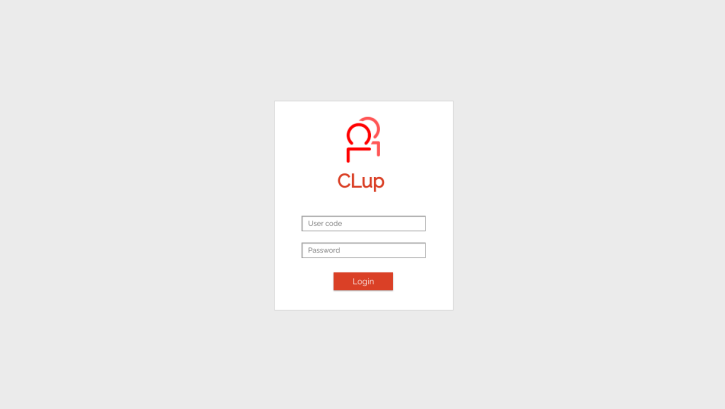

Figure 3.12: Dashboard login.

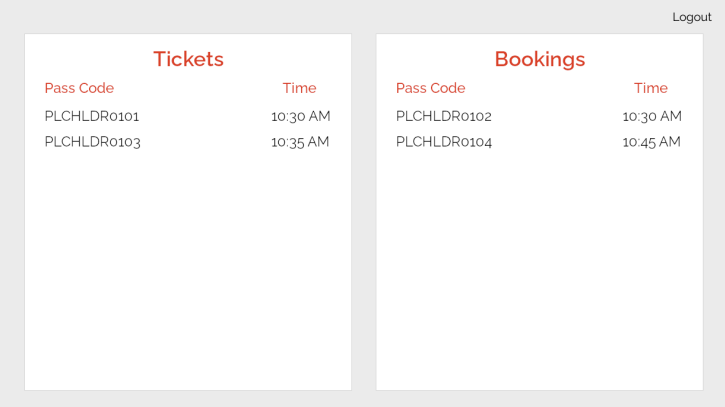

Figure 3.13: Employee homepage.

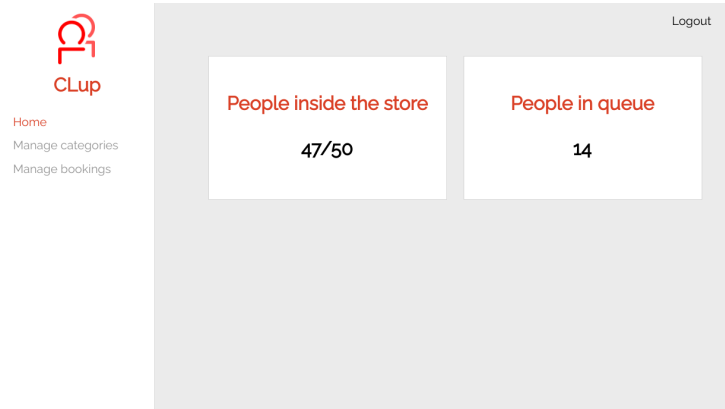

Figure 3.14: Manager homepage.

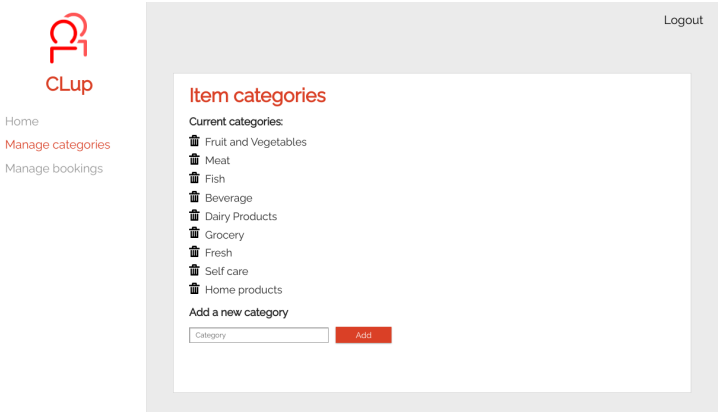

Figure 3.15: Item categories page.

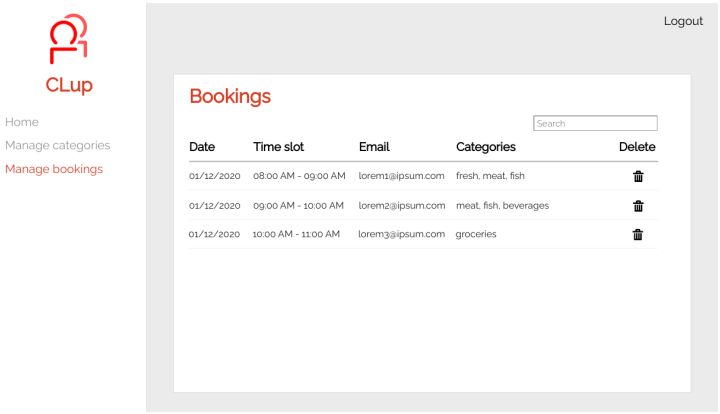

Figure 3.16: Bookings list page.

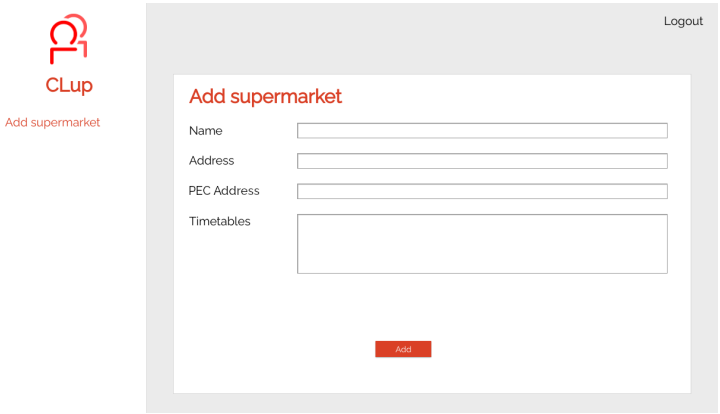

Figure 3.17: Admin new supermarket page.

3.2 User Interface Flow Diagram

The following user interface flow diagram represents the user flow through the Customer App.

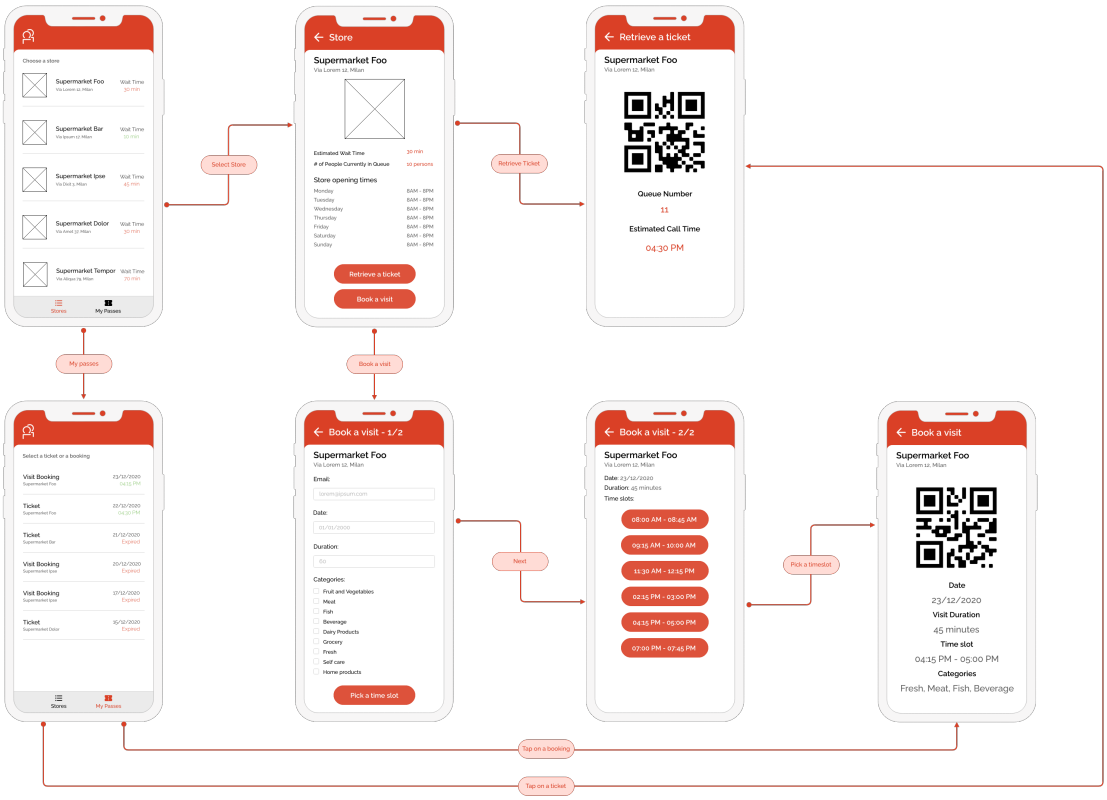

Figure 3.18: User interface flow diagram.

Chapter 4

Requirements Traceability

4.1 Customer

R.1 The system shall allow customers to line-up remotely in a store queue.
CustomerApp, **TokenManager**, **TicketManager**, **QueueManager**

R.2 The system shall generate a new ticket when a customer enters a queue.
TicketManager

R.3 The system shall allow customers which do not have a smartphone to get a ticket in place.
CustomerApp, **TokenManager**, **TicketManager**

R.4 The system shall allow customers to view the number of people lined up in a queue.
CustomerApp, **TokenManager**, **QueueManager**

R.5 The system shall give customers an estimated waiting time.
CustomerApp, **TokenManager**, **QueueManager**

R.6 The system shall fetch the GPS position while the user has retrieved a store pass.
MapsManager

R.7 The system shall allow customers to leave a queue.
CustomerApp, **TokenManager**, **TicketManager**, **QueueManager**

R.8 The system shall allow customers to filter stores by name.
CustomerApp, **TokenManager**, **ViewInfoManager**

R.9 The system shall notify customers when it's time to leave for the store.
CustomerApp, **TokenManager**, **MapsManager**

R.10 The system shall allow customers to book-a-visit to the store and send them the confirmation link and receipt via email.
CustomerApp, **TokenManager**, **BoookingManager**, **MailManager**

R.11 The system shall allow book-a-visit customers to specify the main categories of item they intend to buy.
CustomerApp, **TokenManager**

R.12 The system shall allow customers to delete a booked visit.
CustomerApp, **TokenManager**, **BoookingManager**

R.13 The system shall notify customers when a ticket or booked visit is deleted.
CustomerApp, **TokenManager**

R.14 The system shall accept bookings based onto the already booked category items.
BoookingManager

4.2 Store manager

R.15 The system shall allow a registered store manager to login by using their credentials.
WebApp, **AuthManager**

R.16 The system shall allow store managers to view the current status of people inside the store.
WebApp, **AuthManager**, **DashboardManager**

R.17 The system shall allow store managers to view the current status of people in the queue.
WebApp, **AuthManager**, **DashboardManager**

R.18 The system shall allow store managers to view the booked visits to the store.
WebApp, **AuthManager**, **DashboardManager**

R.19 The system shall allow store managers to set a maximum cap of people inside the store.
WebApp, **AuthManager**, **DashboardManager**

R.20 The system shall allow store managers to delete tickets and booked visits.
WebApp, **AuthManager**, **DashboardManager**

4.3 Store employee

R.21 The system shall allow a registered store employee to login by using their credentials.
WebApp, **StoreApp**, **AuthManager**

R.22 The system shall allow store employee to view the current status of people inside the store.
WebApp, **AuthManager**, **DashboardManager**

R.23 The system shall allow store employee to view the current status of people in the queue.
WebApp, **AuthManager**, **DashboardManager**

R.24 The system shall allow store employee to scan QR codes.
StoreApp, **AuthManager**

R.25 The system shall allow store employee to validate store passes.
StoreApp, **AuthManager**, **ScanManager**

4.4 CLup admin

R.26 The system shall allow CLup admins to register new supermarkets.
WebApp, **AuthManager**, **Dashboard Manager**

R.27 The system shall generate new manager and staff credential for each supermarket registered.
WebApp, **AuthManager**

Chapter 5

Implementation, Integration and Test Plan

The implementation, integration and testing of the system will follow a bottom-up approach, without omitting the dependencies between components within the same subsystem. This approach is chosen both for the server side and the client side, that will be implemented and tested in parallel. An incremental integration facilitates bug tracking and generates working software quickly during the software life cycle.
External services do not need to be unit tested since it is assumed that they are reliable.

5.1 Plan Definition

What is described below is the implementation plan of the server-side, as it is the most complex part which presents the most critical parts of the system.
Please note that the client-side will be implemented and tested in parallel with the server development. The various client applications will therefore be integrated in the final diagram to show the system as a whole.

In the first step (Figure 5.1) the Model and the EntityManager are implemented and unit tested using a driver for the components that are still under implementation. These components are responsible for all the interactions with the database and are needed by the majority of all the other components.

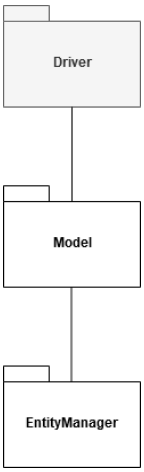

Figure 5.1: Bottom-up approach first step.

In the second step (Figure 5.2) the components responsible for the authentication are implemented. This is to give priority to the access and permissions mechanism of the system, required by other components.

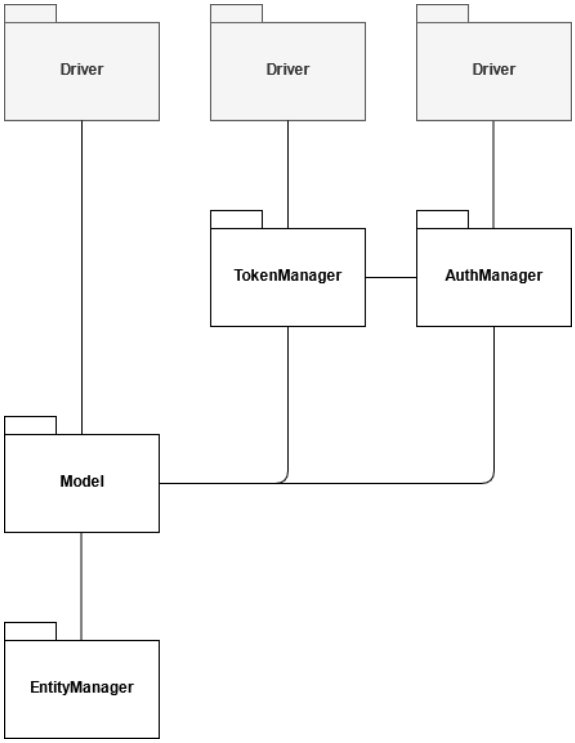

Figure 5.2: Bottom-up approach second step.

The third step (Figure 5.3) shows the implementation of the DashboardManager. A stub is introduced to simulate the MailManager component.

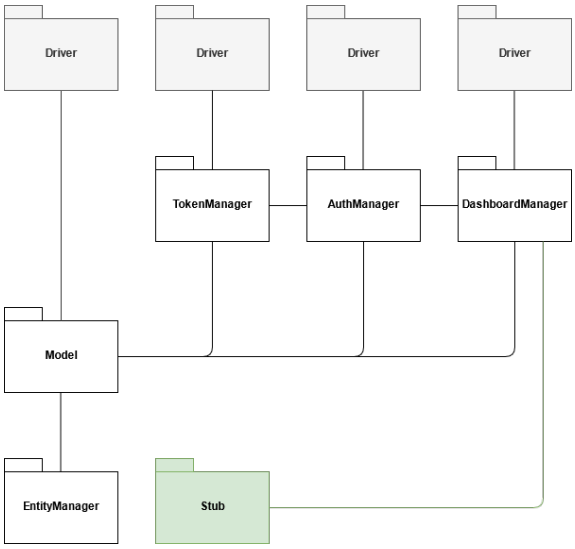

Figure 5.3: Bottom-up approach third step.

In the fourth step (Figure 5.4) the WebApp is integrated and supersedes the drivers for AuthManager and DashboardManager. Also the StorePassManager component is implemented and tested with its own driver.

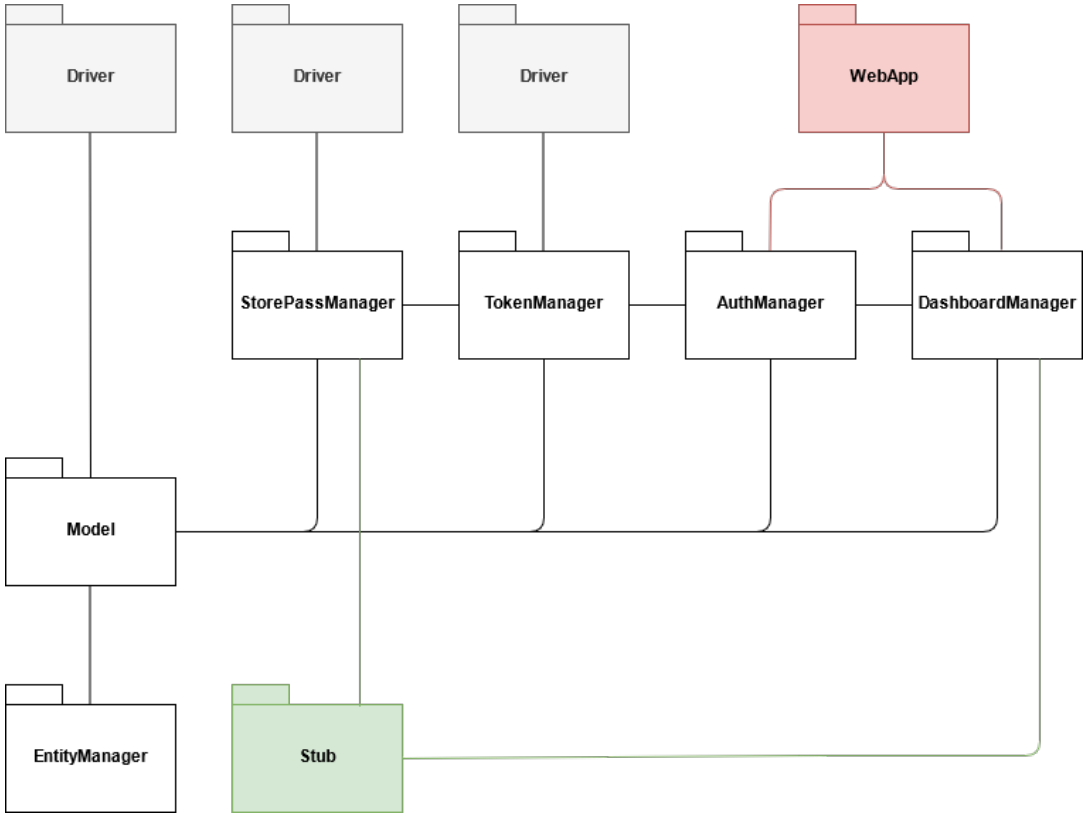

Figure 5.4: Bottom-up approach fourth step.

The fifth step (Figure 5.5) shows the implementation and testing of the remaining server components.

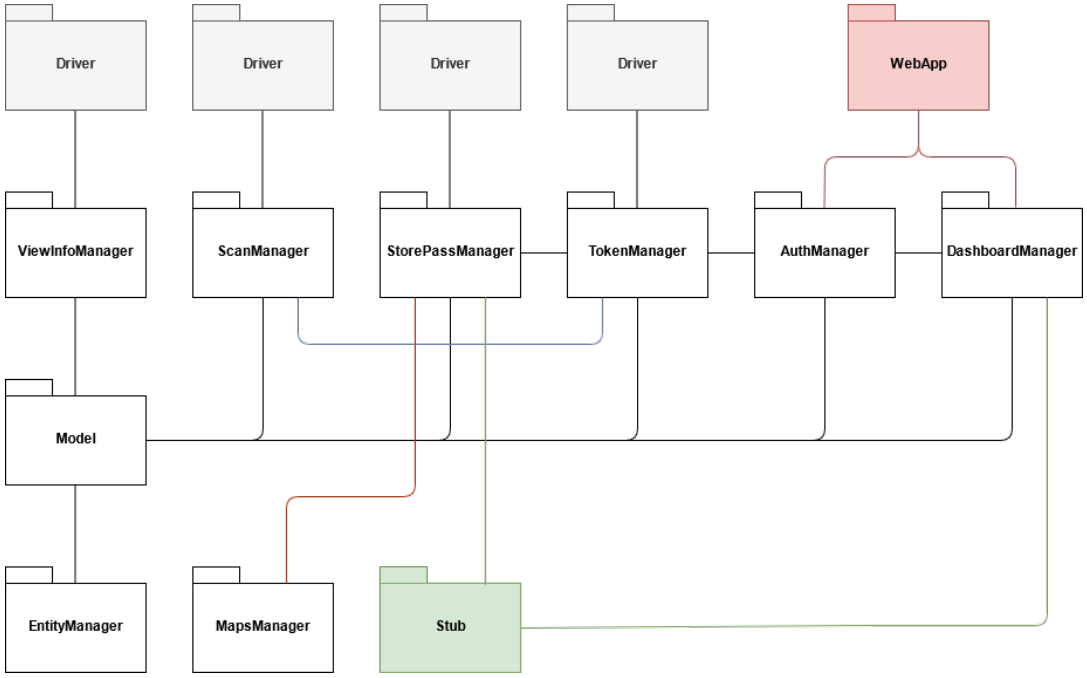

Figure 5.5: Bottom-up approach fifth step.

The final step (Figure 5.6) shows the implementation of the whole system.

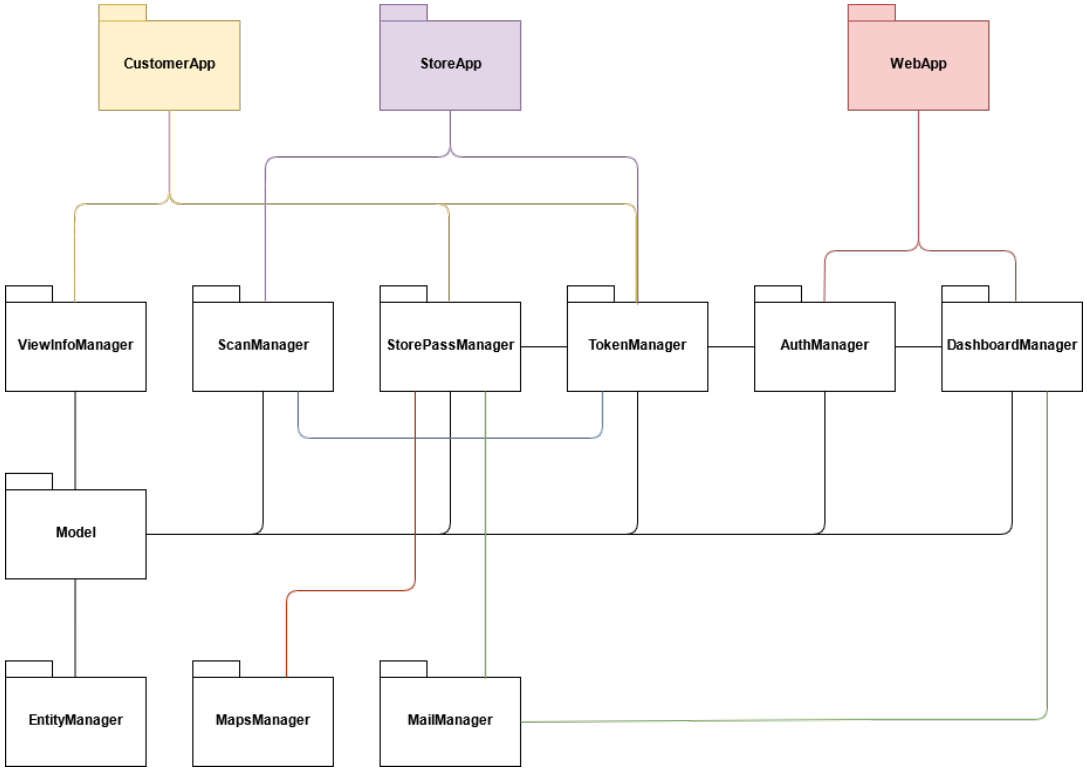

Figure 5.6: Bottom-up approach final step.

5.2 Technologies

This section will analyse the possible technologies available to develop each component of the system.

5.2.1 Mobile

The mobile applications must be accessible by as many users as possible, hence it must be developed for the major mobile OSes, Android and iOS. In order to achieve this, it is possible to adopt a cross-platform development framework which avoid the burden of having to use two different native codebases. A lot of alternatives are available, most of them consisting of a web view, such as Apache Cordova and Ionic. However, for scalability and performance reasons, these options are discarded, in favour of a more native approach with Flutter, an open-source UI software development kit created by Google. Since this framework compiles to native code, it provides better performance and scalability. At the same time, the availability of a big standard library of UI components avoids the need of third party packages which allows for faster development.

5.2.2 Back-end server

The back-end will be implemented in Java EE with the TomEE application/web server. Java EE consists of a powerful set of specifications and APIs which facilitates development of Enterprise Applications and allows developers to focus on the most important aspects of the application. It allows fast development and easy integration with documentation tools that help speed up the work.
The web server will conform to the REST architectural design, providing a REST API, which allows for great flexibility.

5.2.3 Front-end web app

Java templating with Thymeleaf will be used for the web app interface. Its *natural templating* capability ensures templates can be prototyped without a back-end. This makes development faster than other popular template engines such as JSP.

5.2.4 Database

MySQL server is one of the world's most popular databases. It is open source, reliable, cost-effective and easy to manage. For this reason, it is chosen for the data storage system.

5.2.5 Token System

For the token system the choice fell on JSON Web Token which is an internet standard.
When the store employee successfully logs in using their credentials or a customer opens the app for the first time, a JSON Web Token is returned and saved locally in the app, instead of the traditional approach of creating a session in the server and returning a cookie.
When the client wants to access a route the token is sent in the HTTPS request. This is a stateless authentication mechanism as the user state is never saved in server memory. The server's protected routes will check for a valid JWT in the request, and if it is present, the user will be allowed to the resources.

As JWTs are self-contained, all the necessary information is there, reducing the need to query the database multiple times.

5.2.6 Testing tools

Automated tests help ensure that the applications performs correctly before publishing them, while retaining features and bug fixes velocity. Several tools are needed:

- **JUnit**: a unit testing framework for the Java programming language. It allows to define the test for each of the parts developed in a Java application. It is very well documented and allows to be integrated with other testing tools.
- The **test** and **flutter_test** packages provided by the core Flutter framework: the *test package* provides the core framework for writing unit tests for the mobile applications and the *flutter_test package* provides additional utilities for testing widgets.
- **Mockito**: a popular mock framework which can be used in conjunction with JUnit and Flutter testing packages. Mockito allows to create and configure mock objects. Using Mockito greatly simplifies the development of tests for classes with external dependencies.

Chapter 6

Effort Spent

6.1 Teamwork

Task	Hours
Initial briefing	2
Deployment View	2
Requirements Traceability	2
Components interfaces	3
Selected architectural styles and patterns	1
Implementation, Integration and Test Plan	8
Document Revision	6

6.2 Samuele Negrini

Task	Hours
Introduction	3
Component View	11
Deployment View	0.5
Implementation, Integration and Test Plan	0.5

6.3 Giorgio Piazza

Task	Hours
Runtime View	11
Other design decisions	4
User Interface Design	3
Implementation, Integration and Test Plan	0.5