



POLITECNICO

MILANO 1863

Politecnico di Milano

AA 2020 – 2021

Computer Science and Engineering

Software Engineering 2 - Project

DD

Design Document

Claudio Alfredo Emanuele – 963293

Antonio Guadagno – 966763

| | |
|--|----|
| 1. <i>Introduction</i> | 3 |
| 1.1. <i>Purpose</i> | 3 |
| 1.2. <i>Scope</i> | 3 |
| 1.3. <i>Definition, acronyms, abbreviation</i> | 4 |
| 1.3.1. <i>Definitions</i> | 4 |
| 1.3.2. <i>Acronyms</i> | 4 |
| 1.3.3. <i>Abbreviations</i> | 4 |
| 1.4. <i>Revision history</i> | 4 |
| 1.5. <i>Reference document</i> | 4 |
| 1.6. <i>Document structure</i> | 4 |
| 2. <i>Architectural design</i> | 5 |
| 2.1. <i>Overview</i> | 5 |
| 2.1.1. <i>High level components</i> | 7 |
| 2.2. <i>Component view</i> | 9 |
| 2.3. <i>Deployment view</i> | 12 |
| 2.4. <i>Runtime view</i> | 14 |
| 2.5. <i>Component interfaces</i> | 24 |
| 2.6. <i>Selected architectural styles and patterns</i> | 25 |
| 2.7. <i>Other Design Decision</i> | 26 |
| 3. <i>User interface design</i> | 27 |
| 3.1. <i>Mockups</i> | 27 |
| 4. <i>Requirements traceability</i> | 36 |
| 5. <i>Implementation, integrations and test plan</i> | 39 |
| 5.1. <i>Overview</i> | 39 |
| 5.2. <i>Implementation plan</i> | 39 |
| 5.3. <i>Integration strategy</i> | 41 |
| 6. <i>Effort spent</i> | 44 |
| 7. <i>References</i> | 44 |

1. Introduction

1.1. Purpose

The present document is the Design Document (DD) of Clup, a software that will help store managers and customers to respect all the restrictions imposed by governments, due to the pandemic, during the grocery shopping activities.

The purpose of this document is to provide more detailed information on the architecture of the software described in the RASD.

Indeed, if the RASD is useful to provide an abstract view of the system with its functionalities, the DD is essential for the software development because it provides more detailed information on the architecture of the system itself. In fact, the Design Document goes deeper into details of the design, providing an overall guidance to the architecture of the system. For this reason, the DD document is primarily addressed to the software development team.

In particular, the following topics are touched by the document:

- High-level components and their interaction
- Component view
- Deployment view
- Runtime view
- Component interfaces
- Selected architectural styles and patterns
- User interface design
- Requirements traceability:
- Implementation
- Validation
- Testing

1.2. Scope

Clup is an application designed to assist both store managers and customers during the Coronavirus emergency. Grocery shopping is an essential need, but overcrowding must be avoided. For this reason, the software is designed to allow store managers to regulate the afflux of people inside grocery stores and to save people from lining up outside the stores, letting them to line up from home.

The application can be used by registered customers to get a ticket or to book a visit to a grocery store.

Registered customers will be able to monitor the situation of the queue in real time and will also receive a notification when it's time for them to head to the store.

Also people who don't have access to the required technology can interact with the system, of course not using the mobile application but using the ticket distributors that are in front of the grocery stores.

This kind of customers, identified as "unregistered customers", won't be able to book a visit and they will not be able to line up from home too because they need to reach the store to get a ticket on the spot.

Further information are given in the RASD.

1.3. Definition, acronyms, abbreviation

1.3.1. Definitions

- **Registered Customer:** a Customer who has already installed the application “Clup” on his/her smartphone and is already registered to the service.
- **Unregistered Customer:** a Customer who does not have access to the required technology or who does not want to sign up to the service.
- **Store manager:** manager of a grocery store who signs up to the service.

1.3.2. Acronyms

RASD: Requirement Analysis and Specification Document.

DD: Design Document

GPS: Global Positioning System

DBMS: Data Base Management System

API: Application Programming Interface

TSL: Transport Layer Security

MVC: Model-View-Controller

1.3.3. Abbreviations

Rn: functional requirement number n

1.4. Revision history

05/01/2020 – First version of DD

1.5. Reference document

Clup specification document

Slides on the course website

Oracle web site

1.6. Document structure

- *Chapter 1:* this chapter introduces the Design Document, specifying the utility of the project and the structure of the DD.
- *Chapter 2:* in this chapter the components of the system are presented, relationships between them are specified and the system architecture is described.
- *Chapter 3:* mockups to show how the user mobile app and the dedicated web app should look like are presented
- *Chapter 4:* the relationship between requirements and system components presented in chapter 2 is described
- *Chapter 5:* shows the effort spent for each member of the group.
- *Chapter 6:* includes the reference documents.

2. Architectural design

2.1. Overview

The software architecture is a three layers structure that follows the PAD pattern (Presentation, Application, Data access).

Presentation layer: the presentation layer provides an interface that allows the user to interact with the system. The purpose of this layer is to render all the information that are useful to the user and also to provide an easy-to-use user interface that allows customers and store managers to benefit of all the system functionalities. Considering that the app is addressed to users of all ages and all social classes, the GUI should be very simple and intuitive.

Application Layer: the application layer is responsible for the management of all the functionalities that the system provides to the users: it handles the business logic of the application. The purpose of this layer is to coordinate the work of all the components of the system, to make logical decision and also to manage the exchange of information between the other layers (the interaction of the user with the system should result in an action performed by the system according to the user input).

Data access layer: the data access layer takes care of the management of the information, storing it in a database and also managing the access to the database itself. The purpose of this layer is not only to store information provided by the users but also to provide this information to users when is asked (if the users have the needed permissions).

The architecture is a client-server architecture with client and server allocated in different physical machines. Client and Server communicate each other via internet thanks to the interfaces provided by the different components (hardware and software) that compose the system.

The architecture follows the MVC pattern (Model-View-Controller) and so the interaction between the user and the system consists in the user asking for a functionality and the server providing the required functionality. Depending on the required functionality, the server behaves differently.

A Registered Customer wants to take a ticket.

The server shows to the customer all the grocery stores that are located in his/her same municipality (showing also the distance between the store and the customer current position, the number of people that are lining up and the estimated waiting time). Then the server asks to the customer how he/she intends to reach the store: this information is useful to calculate the time needed by the customer to reach the store and so to calculate when it's time to send a notification to alert him/her that it's time to head to the store. In order to emit the ticket, the server has to analyse information that are stored in the database such as the number of people that are currently lining up for the selected store, the distance between the store and the current customer position and the real time traffic conditions. The server has also to generate an unique QR code associated with the ticket and to update information about the queue situation.

A Registered Customer wants to book a visit.

The procedure is similar to the once described above. The server receives the request, then it asks to the customer for the address where he/she will be located just before the moment of the visit and for the date and the time of the visit. Then the server uses maps to find all the stores that are located in the same municipality of the selected address and consults the database to find all the stores that have free slots for a booked visit for the selected date and time. The user selects a store and then the server has to store in the database the information of the booked visit. The server has also to generate a ticket associated to the visit.

An Unregistered Customer wants to get a ticket.

The only way that an Unregistered Customer has to interact with the system is to use the ticket distributor that are located in front of the stores. The server receives the request from the ticket distributor and this time it has only to update the information about the queue of the store and to generate a ticket with a unique QR code.

Every time that a QR Code is scanned, the server has also to update general information about the store (such as the number of customers per day and the average time of the visit), to update the conditions of the queue and to update the status of the ticket.

A Store Manager retrieves/updates information about his/her grocery store

Store managers can connect to the web app to retrieve or to update information about the grocery store. In the first case, the server receives the request and gets from the database all the information about the specific grocery store (such as the number of people that enters the store day by day and the average time of the visit, general or for a specific registered customer. In the second case, the server receives the update information by the store manager (such as new opening hours, new number of people that are allowed to simultaneously enter the store) and the stores it in the data base.

Last but not least, the application is a distributed application, so the three layers previously described will be installed on different physical machines. Further information will be provided later in the present document.

2.1.1 High level components

The application to be developed is a distributed application structured according to the classic three-tiers architecture.

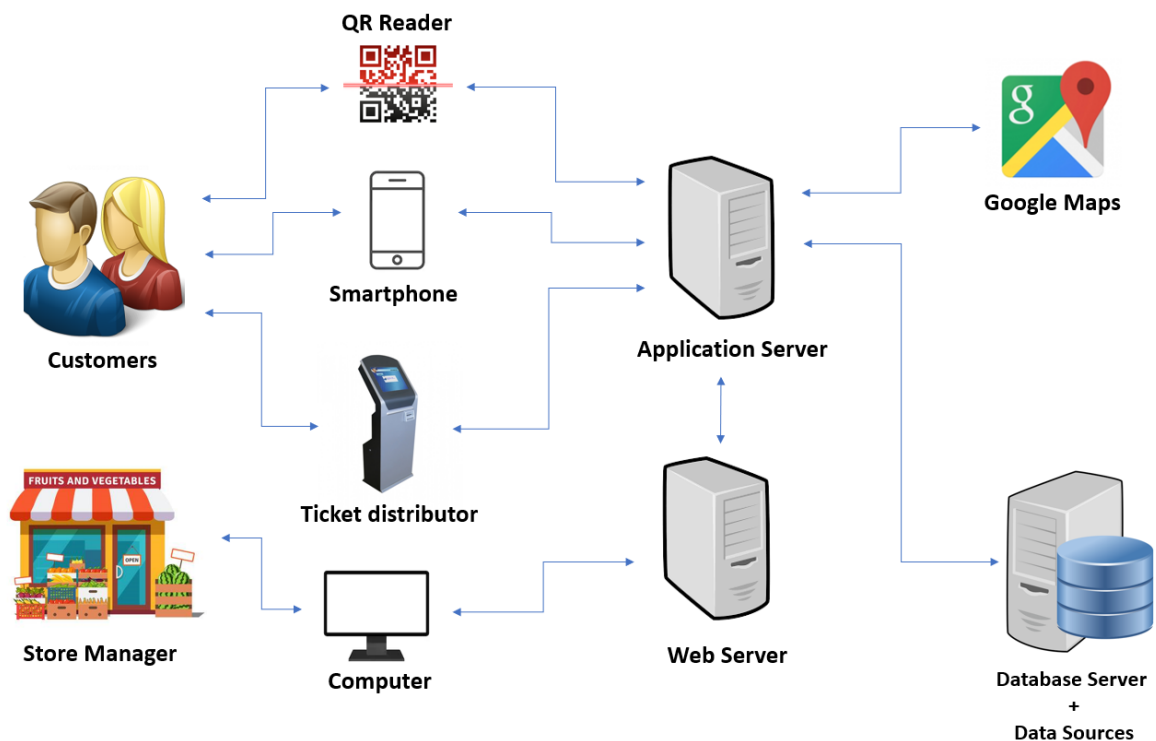
The three-tiers architecture is a well-established software application architecture that organizes applications into three logical and physical computational tiers.

The three tiers of the architecture are the user interface, the business logic (the application tier where data is processed) and the data tier (where data associated with the application is stored and managed).

This software architecture is thought to guarantee to the system very important characteristics such as

- **Scalability:** by separating out the different layers, it is possible scale one independently from the others, depending on the real needs at any given time.
- **Reliability and Availability:** by hosting different parts of the application on different servers and using cached results.
- **Security:** the middle layer guarantees that the user cannot have a direct access to the database.

It's important to say that we refer to a three-tier architecture because we considered the Application Server together with the Web Server. Otherwise, if you want to consider the Web server as strictly distinct from the Application server, the system can be considered a four-tier architecture.



To access the system functionalities, Registered Customers are provided with a mobile application that they can install on their smartphone, while Store Managers have to use the dedicated WebApp.

Registered Customers, once logged into the mobile application, can communicate directly with the Application Server (so they can ask for a ticket or for a booked visit), while Store Managers interact with the system by using the dedicated WebApp managed by the Web Server.

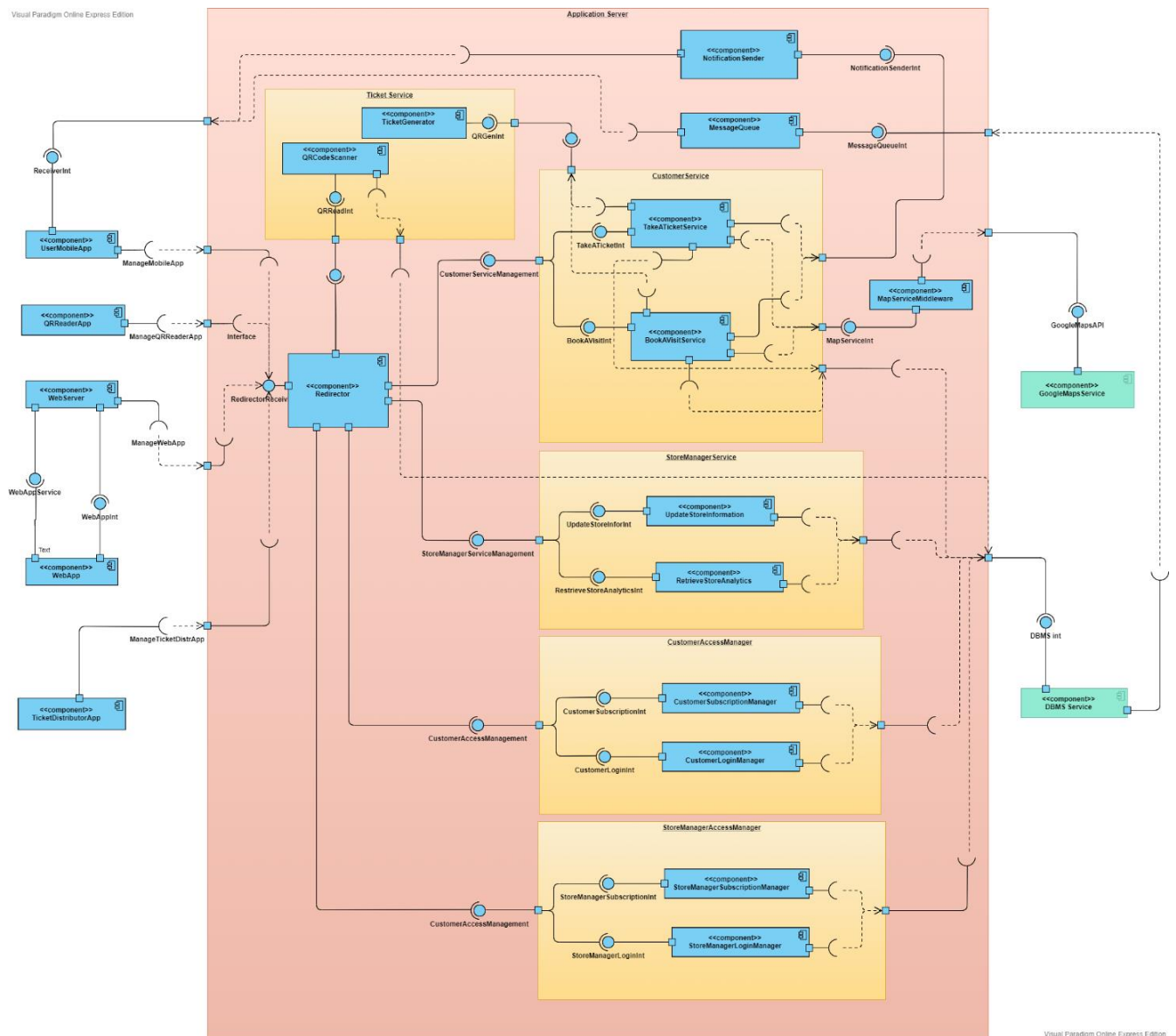
Unregistered Customers interacts with the system through ticket distributors that are in front of the grocery stores. Using ticket distributors, Unregistered Customers can get a ticket but not book a visit.

Application Server and Web Server, which together compose the Application tier, communicate with each other. In particular, the Web Server forwards the requests to the Application Server. To fasten and lighten the communication, caches are used in front of the Presentation tier.

Finally, the Application Server communicates synchronously with the Database Server (data access layer) to retrieve information or asynchronously to store information when needed. Google Maps is used as an external service in order to locate grocery stores and also to calculate the time needed by customers to reach the grocery's.

2.2. Component view

In the following diagram, we focus on the examination of the previously mentioned components, with main focus on the Application Server which contains all the components that manage the business logic of the application. The Application server communicates with all the other nodes, it gathers information from the external services, manages user accounts and saved data from the DB server and also takes requests and send back responses to the User.



As we can see, the Application Server consists of the following components:

- 1) **MapServiceMiddleware**: this component provides the interface `MapServiceInt` that is used by `TakeATicketService` and `BookAVisitService` to locate the current Registered Customer position and to find all the grocery stores near him/her (or near the address inserted during the booking of a visit). This component is useful because, if in a future version of the system the component `GoogleMapsService` will be replaced by a different similar component, we can change only the `MapServiceMiddleware` component without changing other nodes of the Application Server.
- 2) **NotificationSender**: this component provides the interface `NotificationSenderInt` and it's used by `TakeATicketService` and `BookAVisitService` to send notifications to the RegisteredCustomer via the `ReceiverInt` interface offered by `UserMobileApp`. Notifications are sent to the user to alert him/her that it's time to head to the store or to remind him/her of a booked visit.
- 3) **TakeATicketService**: this component receives ticket requests from `UserMobileApp` and from `TicketDistributorApp`. If the request is sent by `UserMobileApp`, `TakeATicketService` uses the `MapsAPI` interface to find all the grocery stores near the RegisteredCustomer current position and uses the `NotificationSender` interface to send a notification to the RegisteredCustomer 10 minutes before that the estimated waiting time is equal to the time needed by Registered Customer to reach the store selected for the visit. Otherwise, if the request is sent by `TicketDistributorApp`, this component only emits a ticket and provides the information about the length of the queue and the estimated waiting time. This component interacts with the DBMS Service to store information about tickets.
- 4) **BookAVisitService**: this component receives booking requests from the user's mobile app. It uses the `GoogleMapsAPI` interface to find all the grocery stores near the address inserted by the RegisteredCustomer during the booking procedure and selects only that stores that have a bookable ticket available for the date and the time selected by the Registered Customer (with a tolerance of 30 minutes). Moreover, it uses the interface `NotificationSender` to send a notification to the RegisteredCustomer 2 hours before the time of the visit or at the 6 PM of the day before if the time of the visit is earlier than the 10 AM. This component interacts with the DBMS Service to store information about visits.
- 5) **Redirector**: this component provides the `RedirectorReceiver` interface. It receives requests from `UserMobileApp`, `WebServer` and `TicketDistributorApp` and then redirect these messages to the right component of the Application Server. It acts as the Façade in a Façade Pattern.
- 6) **CustomerLoginManager**: it manages the authentication of Registered Customers. To fulfil its goals, this component needs to communicate with the DBMS to check that credentials are correct.
- 7) **CustomerSubscriptionManager**: this component contains all the procedures to allow customers to register to Clup services.
- 8) **StoreManagerLoginManager**: it manages the authentication of store managers. To fulfil its goals, this component needs to communicate with the DBMS to check that credentials are correct.

- 9) **StoreManagerSubscriptionManager**: this component manages all the procedures to allow store managers to register to Clup services. This component differs from CustomerSubscriptionManager because the registration form sent to the store managers contains additional fields with respect to the one sent to customers, in particular, it contains a field for documents that certifies that the person who is registering to the service really runs an activity that is a grocery store.
- 10) **UpdateStoreInformation**: this component allows store managers to update store information such as opening hours, the number of people that are allowed to temporarily access the store and to physically lining up in front of it and the category of goods offered by the store.
- 11) **TicketGenerator**: this component generates Tickets with the relative QR Codes.
- 12) **QRCodeScanner**: this component is used by QRReaderApp, it reads the QRCode and also saves in the DataBase information related to the time of the visit.
- 13) **MessageQueue**: this component is used to keep the customer updated in real time. When the situation of a queue changes in the database, the updated information are sent to MessageQueue by the DBMS Service and MessageQueue sends the information to the UserMobileApp using ReceiverInt.

The external components of the system are:

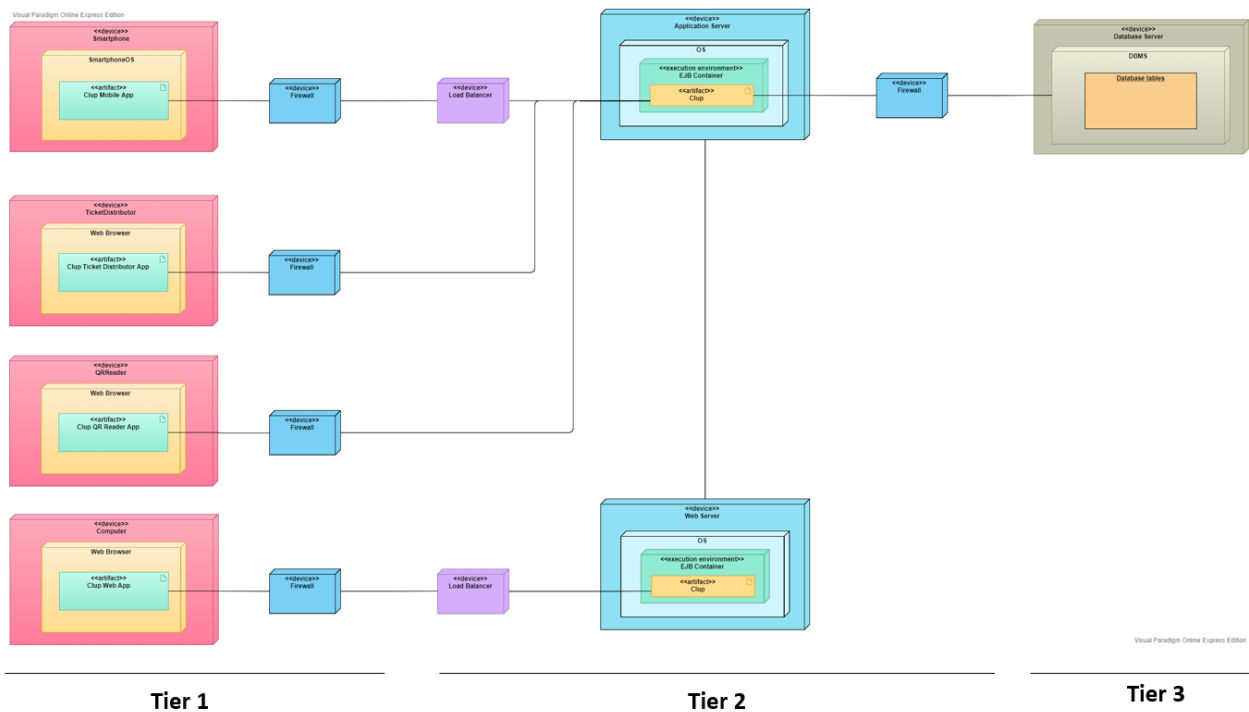
- 1) **GoogleMapsService**: it provides the GoogleMapsAPI interface used by MapServiceMiddleware to locate the Registered Customer Position, and the grocery stores near his/her position (or near the position specified in case of booked visits).
- 2) **DBMSServices**: this component allows the other components of the Application server to interact with the database. The Interface provided by this component contains methods to enable efficient creation, manipulation and querying of the database.

[An alternative solution could have been to introduce a Data Manager Component between the Application Server and the DBMS Service so that, if we decide for some reason to change the DBMS Service, we don't have to apport changes to several components of the Application Server but just to the Data Manager Component.]

2.3 Deployment view

The System architecture is divided in 3 tiers and it's based on the JEE Framework. In the following, the Clup deployment diagram is presented: the image below shows the execution architecture of the system and represents the distribution of software artifacts to deployment targets (nodes). Artifacts, in general, represent pieces of information that are used or are produced by a software development process and are deployed on nodes that can represent either hardware devices or software execution environments.

In the diagram, external systems are not represented to focus only on the components that host the core functionalities of the application or on the components for which the deployment is effectively executed.



The 3 tiers respectively contain:

Tier 1: in this tier the presentation logic is deployed.

Registered customers must be provided with a mobile application on their smartphones. The mobile application is the easiest way for a customer to have access to the services offered by Clup. The mobile application must be available for both Android and iOS in order to make it available on most of the devices. Registered customers use the mobile application to communicate with the application server in order to take tickets and to book visits. Store managers must be provided with a dedicated Web App on their computers. Also in this case, the Web App must be compatible with at least Google Chrome, Safari, Opera, Microsoft Edge and Firefox, most used web browsers. Store managers use the dedicated Web App to retrieve information concerning their grocery store (information such as the average time of the visit or the number of customers that enter the store day by day) but also to eventually update grocery store information (such as the opening hours or the number of people that are allowed to contemporarily access the building). Ticket distributors need their own application to let Unregistered Customers to interact with the system in order to get tickets on the spot. QR Readers need their own application to let Customers to scan the QR Code on their tickets.

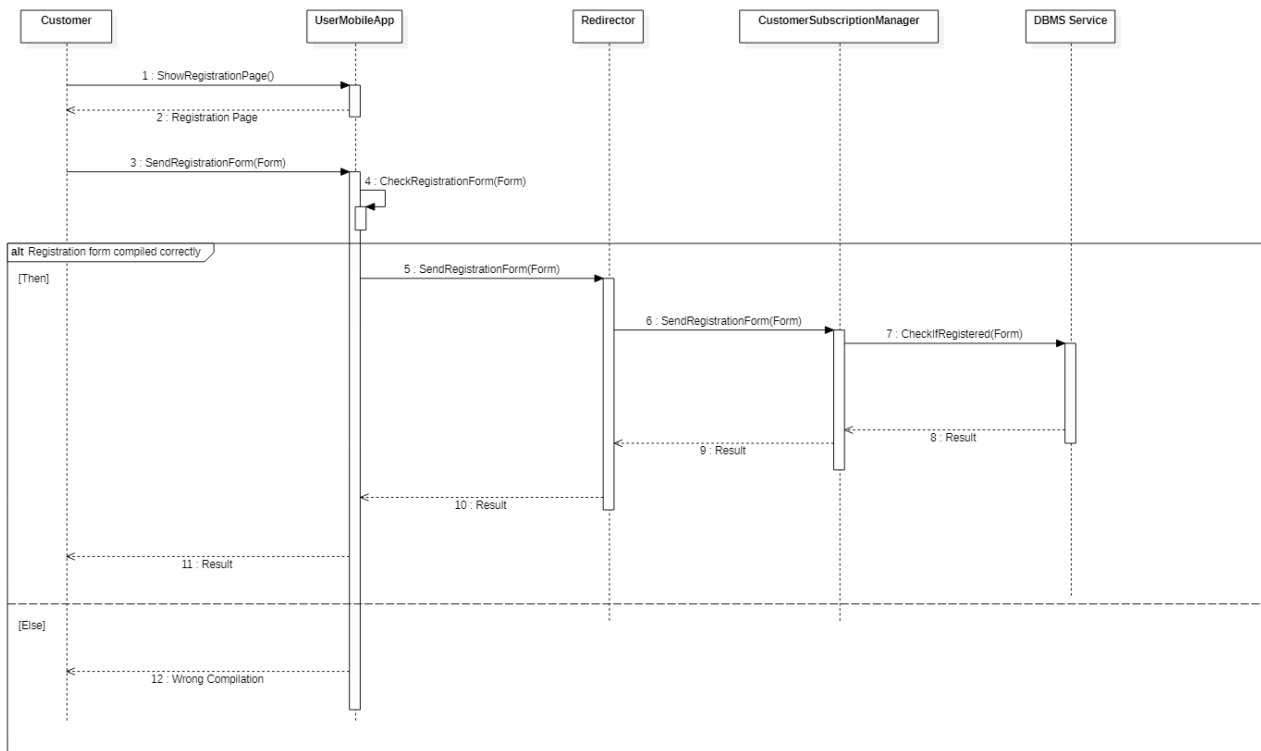
Tier 2: in this tier the application logic is deployed.

Tier 2 includes both the Application and the Web server. The Application server implements the business logic, it handles the user's requests and also provides answers to that requests for all the offered services. The application server is directly addressed by Customers through the Mobile App, the Ticket Distributor App and the QR Reader App. It also handles some requests that are forwarder by the Web Server and sent by Store Managers (for example if they want to retrieve stats regarding their grocery's). To provide all these services, the Application Server must communicate with the Database. The Web Server delivers static web content to store managers and also let the store managers to interact with the business logic of the system.

Tier 3: this tier the data access management is deployed. A DBMS is needed and a good choice can be Oracle Database because it is widely used, it is secure, it offers good results in terms of performance, scalability and affidability, it also supports large databases and reduces the CPU time to process data. The communication with the DBMS is performed via JPA.

2.4 Runtime view

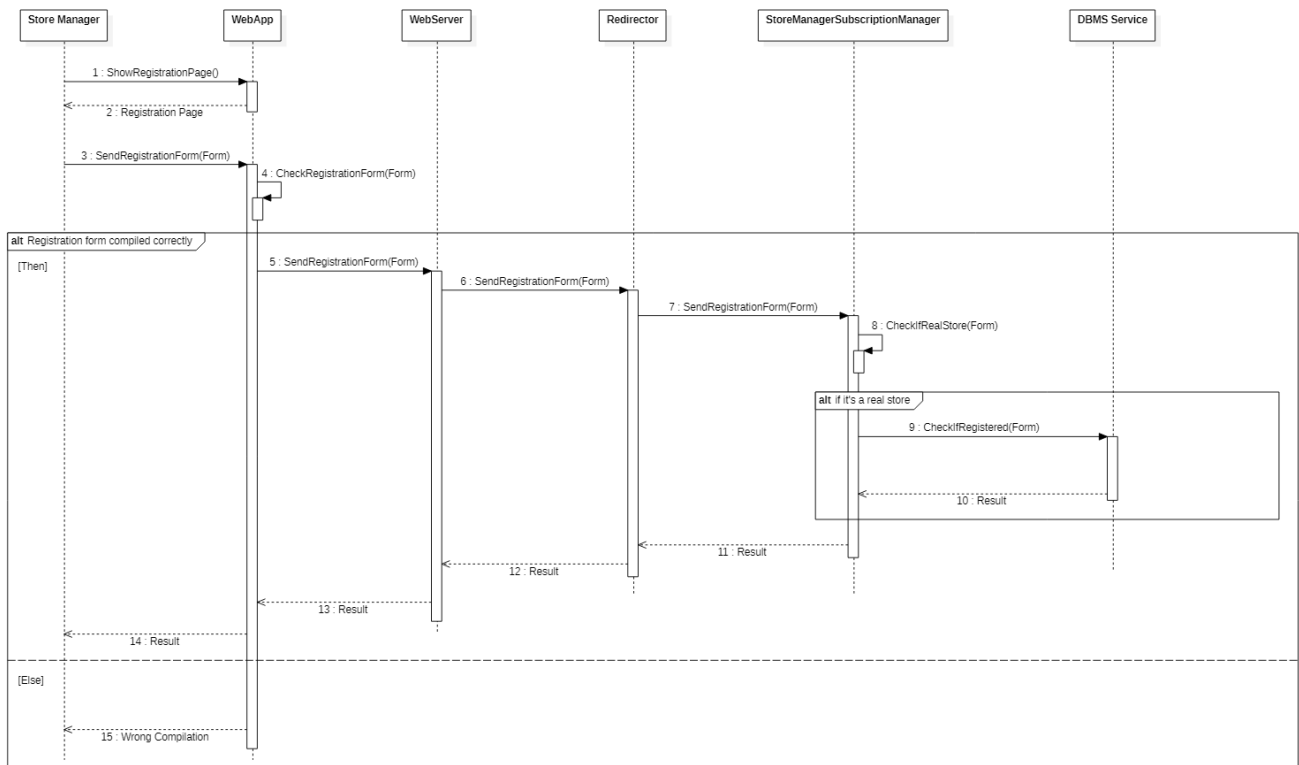
2.4.1 Customer Registration



The Customer asks for the registration page to the UserMobileApp, then compiles the registration form and sends it. Before to send the registration form to the Redirector, the UserMobileApp controls if the form is correctly filled. We have decided for a client that controls if the form is compiled correctly because we don't want to overload the application server with requests that will never be accepted just because some fields are not properly compiled. Anyway, the task to control if a form is properly filled is very easy and can be executed by the user client without any problem (in this phase we only control if all the fields have been filled with a valid value, of course we don't control, for instance, if the chosen username is available because controls like this will require a communication with the Database).

If the registration form is properly compiled, the UserMobileApp forwards the request to the Redirector, the Redirector forwards the request to the CustomerSubscriptionManger and the CustomerSubscriptionManger asks to the DBMS Service to check if the user is already registered. If the user is not registered, the "Result" contains a confirmation message and then the customer is redirected to the log in page, otherwise, if the customer is already registered to the service (his/her phone number is already associated to an existing account), "Result" contains an error message that informs the Customer that his/her phone number is already associated to an account. If the registration form is not compiled properly, the UserMobileApp shows an error message to the Customer to alert him/her that the some fields of the form contains invalid information or are empty.

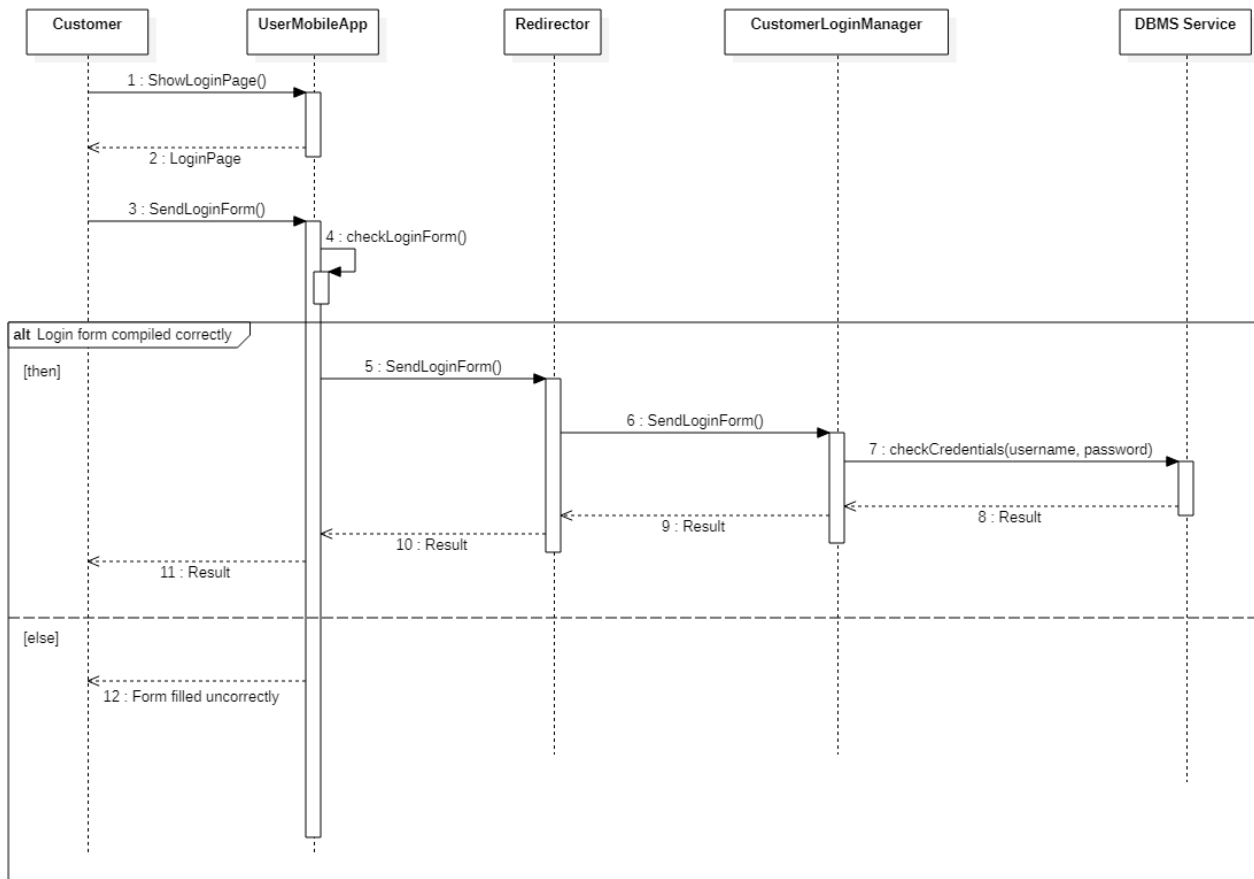
2.4.2 Store Manager Registration



Store managers registration is very similar to Customers registration, the only important difference is that, in the case of Store managers registration, we need to verify that the Store manager is registering a real grocery store in order to avoid jokes or cheating attempts. To do that, StoreManagerSubscriptionManger provides the functionality needed to check if a grocery's is a real store by analysing the documents that the Store manager must provide when fills the registration form. (Note that the documents required are permissions issued by a municipality, so it's not easy to falsify them).

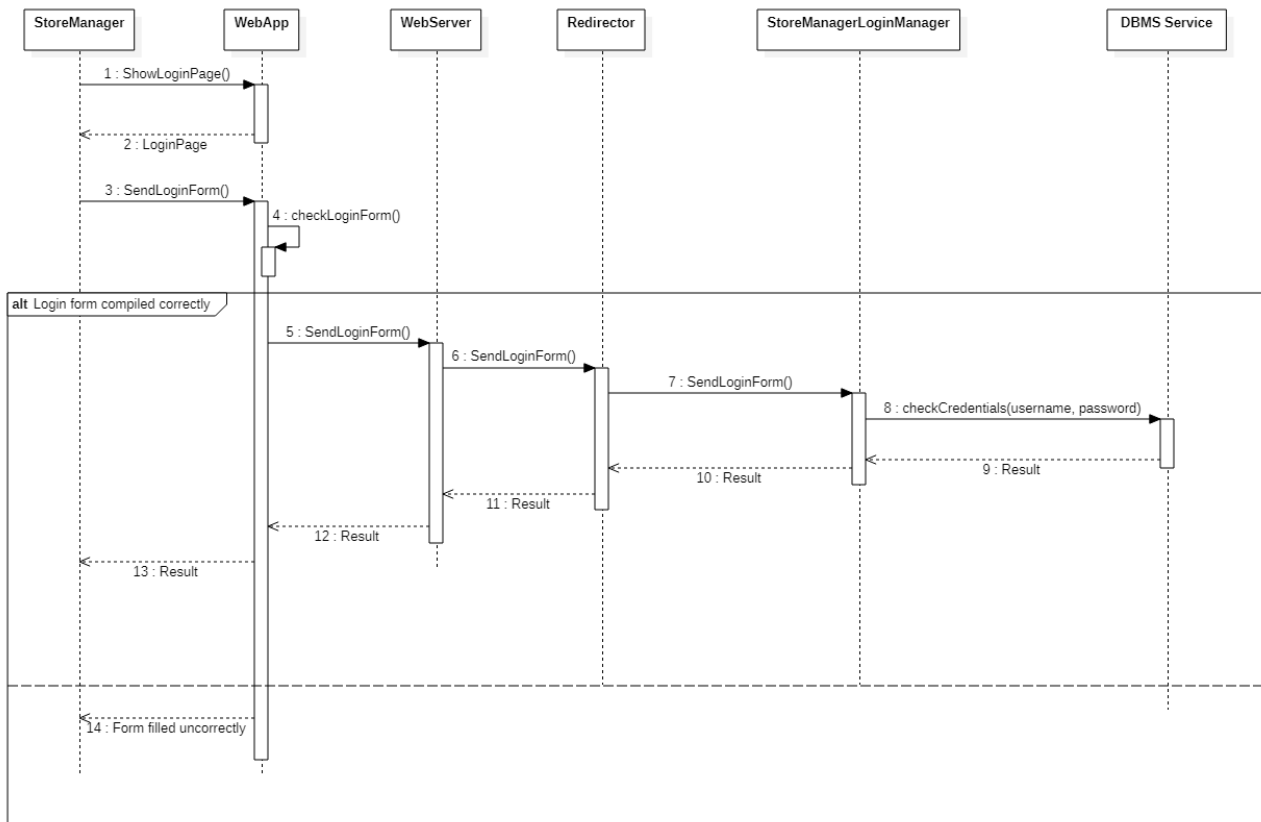
[An alternative solution could have been to introduce an AdministratorApp to let the administrators of the service to check grocery store's documents manually, but at the end we preferred an analysis performed by a machine considering that the number of grocery's that will subscribe to the system can be very high and so it could be not easy to analyse each request by hand: it could take a lot of time or it could require a lot of people examining the documents. We have to consider that grocery shopping is an essential activity and we cannot let customers and store managers to wait maybe for days before that an administrator evaluate the request by hand.]

2.4.3 Registered Customer log in



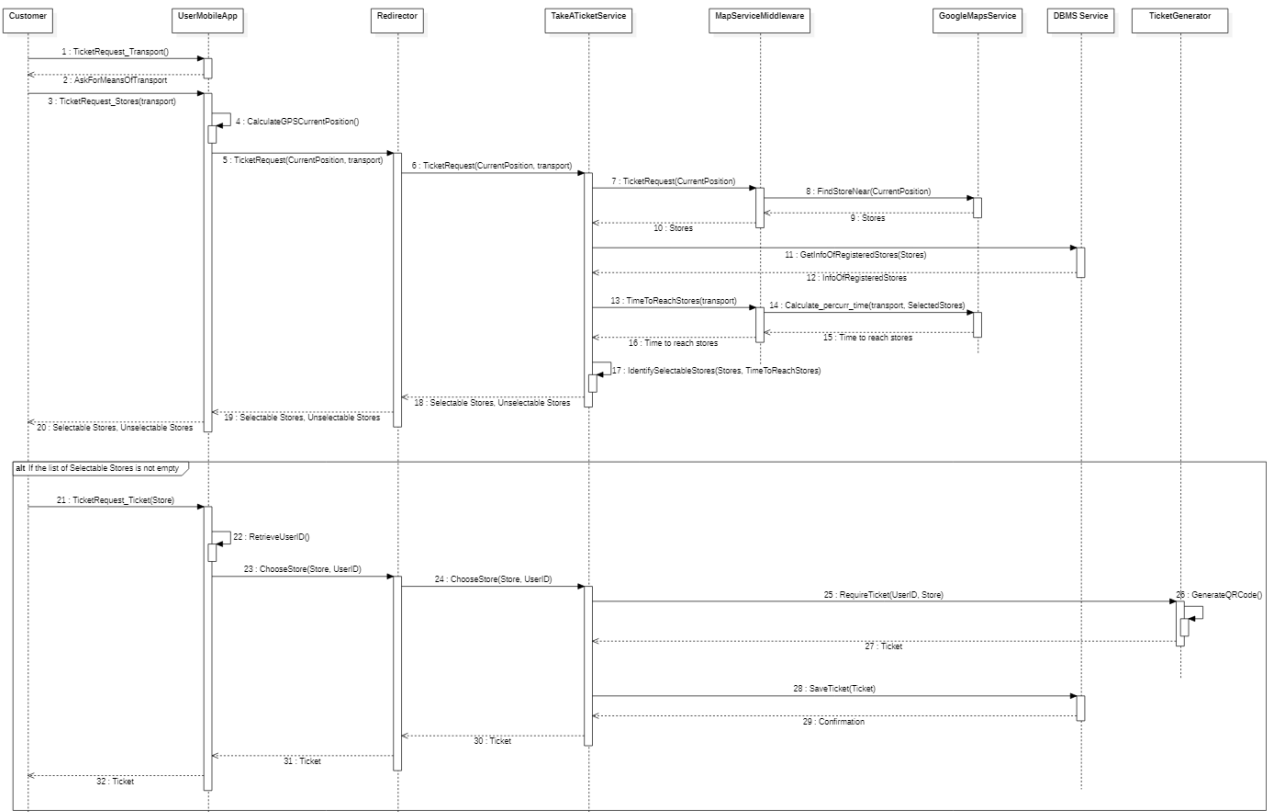
The Customer asks for the log in page to the UserMobileApp, then compiles the log in form and sends it. Before to send the log in form to the Redirector, the UserMobileApp controls if the form is correctly filled. If the login form is properly compiled (username field and password field are not empty), the UserMobileApp forwards the request to the Redirector, the Redirector forwards the request to the CustomerLoginManger and the CustomerLoginManger asks to the DBMS Service to check if the credentials are correct. If credentials are wrong, "Result" contains an error message and the Customer is redirected to the log in page, otherwise, if credentials are correct, "Result" contains a confirmation message and the Customer is redirected to the page where he/she can choose to get a ticket or to book a visit. If the registration form is not compiled properly, the UserMobileApp shows an error message to the Customer to alert him/her that the some fields of the form contains invalid information or are empty.

2.4.4 Store Manager log in



The Store manager log in is essentially identical to the Registered Customer log in, of course in this case components related to the Store Manager are considered instead of component related to Registered Customer.

2.4.5 Registered Customer takes a ticket



The Registered Customer uses the UserMobileApp to send a request for a ticket: the first step is to search all the stores that are subscribed to the service and that are in the same municipality of the Customer current position. The UserMobileApp asks to the Customer the means of transport with which he/she wants to reach the store, then uses the GPS of the Customer device to calculate the Customer current position and finally forwards the request, the chosen means of transport and the Customer current position to the Redirector.

The Redirector forwards the request, the means of transport and the Customer current position to TakeATicketService, TakeATicketService forwards the request and the Customer current position to MapServiceMiddleWare and MapServiceMiddleWare uses the interface provided by Google Maps to retrieve all the stores near the Customer current position (also the once that are not subscribed to the Clup service). Once that MapServiceMiddleWare obtains all the stores that are in the same municipality of the Customer, it sends them back to TakeATicketService.

Using the obtained list of stores, TakeATicketService asks to the DBMS Service to retrieve from the Database information about stores that are registered to the system: if a store in the list obtained by the MapServiceMiddleWare is registered to the service, TakeATicketService needs to retrieve the estimated waiting time and the length of the queue of that store. Stores that are not registered to the system are just discarded and not considered anymore.

Once that TakeATicketService receives the information of the Registered Stores that are in the same municipality of the Customer current position, it sends to MapServiceMiddleWare the means of transport chosen by the Customer and MapServiceMiddleWare asks to Google maps to calculate the time needed by the Customer to reach each of the Registered Stores. The information about the time needed by the Customer to reach each registered store (considering the selected means on transport) is sent to TakeATicketService and, using this information, TakeATicketService can divide the stores in 2 categories: Selectable Stores (stores where the Customer can get a ticket) and Unselectable Stores (stores where the Customer cannot get a ticket because they

are closed or because they will close before that the Customer will be allowed to enter).

TakeATicketService sends the two list of selectable and unselectable stores back to the user via the Redirector and then the UserMobileApp.

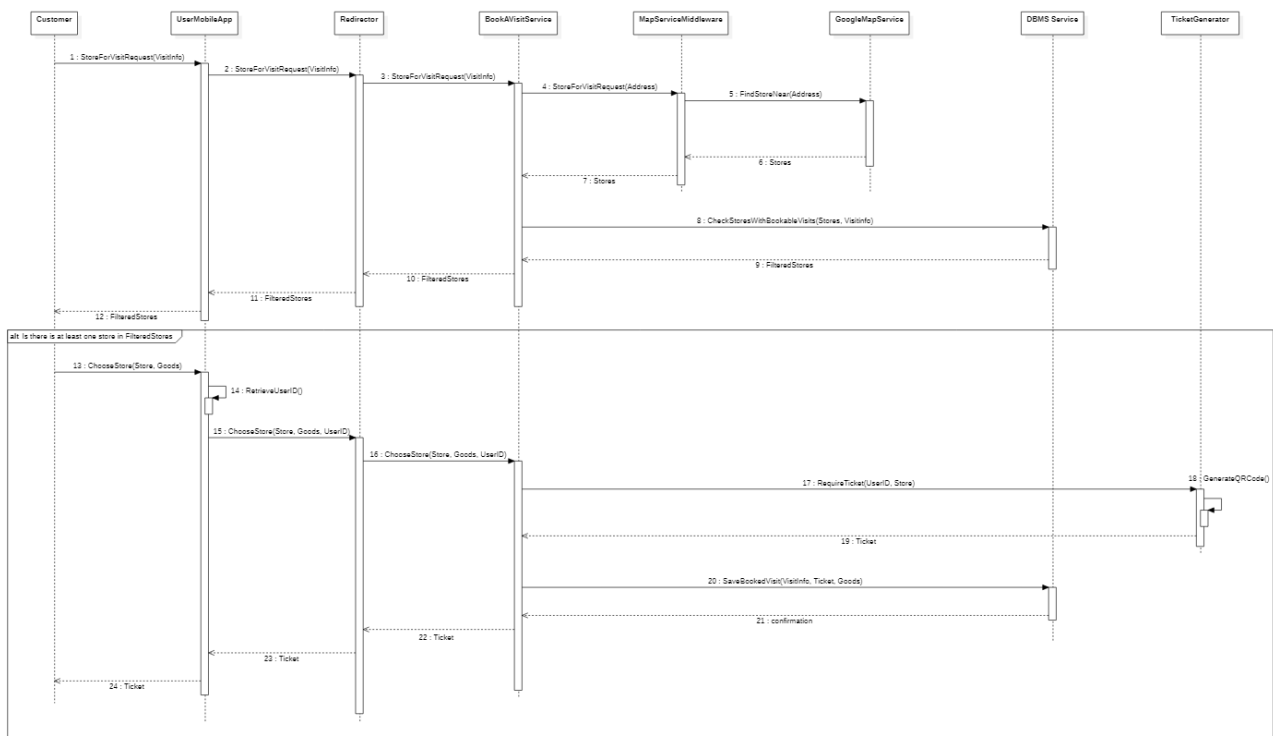
Unselectable stores are sent to the user because, if there isn't any selectable store, the Customer needs to understand if he/she cannot get a ticket because there are not stores in his/her municipality that are subscribed to the system or because there are stores in his/her municipality that are subscribed to the system but they are all closed or will close before that he/she will be allowed to enter.

If there is at least one selectable stores, the Customer chooses the store where he/she wants to go, the UserMobileApp retrieves the UserID and then sends the request for a ticket (with also information about the selected store and the UserID) to the Redirector who forwards this information to TakeATicketService.

Now that TakeATicketService has the selected store and the UserID, it asks to TicketGenerator to generate a new Ticket (with an associated QRCode) uniquely associated to a user, to a grocery store and to a QR Code.

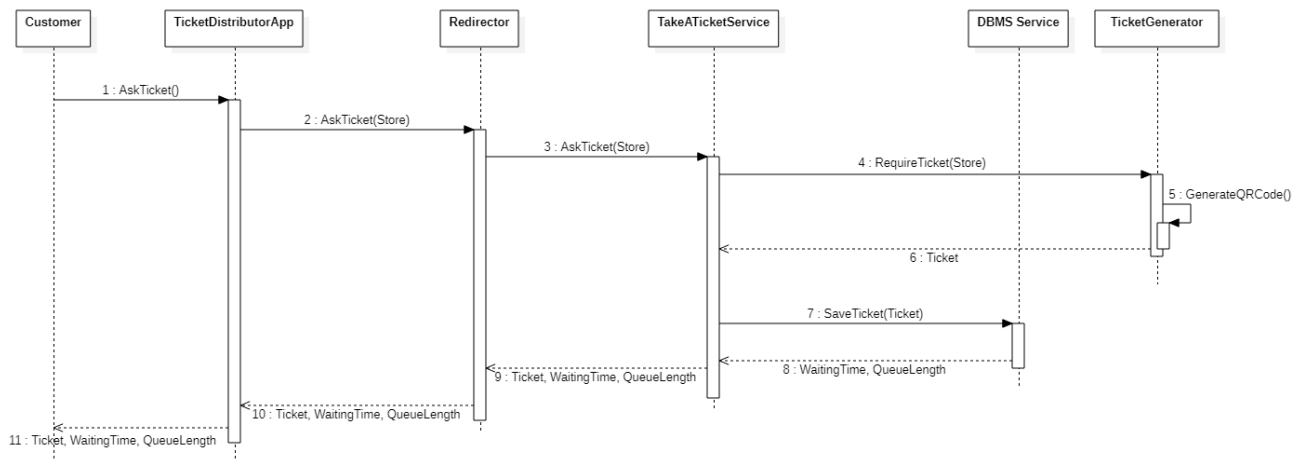
TakeATicketService asks to the DBMS Service to save information about the ticket in the Database and then sends back the ticket to the Registered Customer via Redirector and then the UserMobileApp.

2.4.6 Registered Customer books a visit



The procedure to book a visit to a grocery store is very similar to the procedure followed by a Registered Customer to take a ticket. The main differences are that the UserMobileApp doesn't have to calculate the Customer current position because the Registered Customer has to insert the address he/she will be just before the moment of the visit and that when the Registered Customer selects a store he/she cal also selects the kind of goods that he/she intends to buy.

2.4.7 Unregistered Customer gets a ticket using the ticket distributor



The Customer asks for a ticket to the TicketDistributorApp, TicketDistributorApp forwards the request and the store to the Redirector and the Redirector forwards the request to TakeATicketService (TicketDistributorApp knows the store because a ticket distributor is associated to exactly one store).

TakeATicketService asks to TicketGenerator to generate a new unique Ticket with an associated QRCode (this time the ticket cannot be associated to a specific Customer because Unregistered Customers are by definition not registered to the System), TicketGenerator generates the Ticket and sends it back to TakeATicketService.

TakeATicketService asks to the DBMS Service to save information about the ticket in the database.

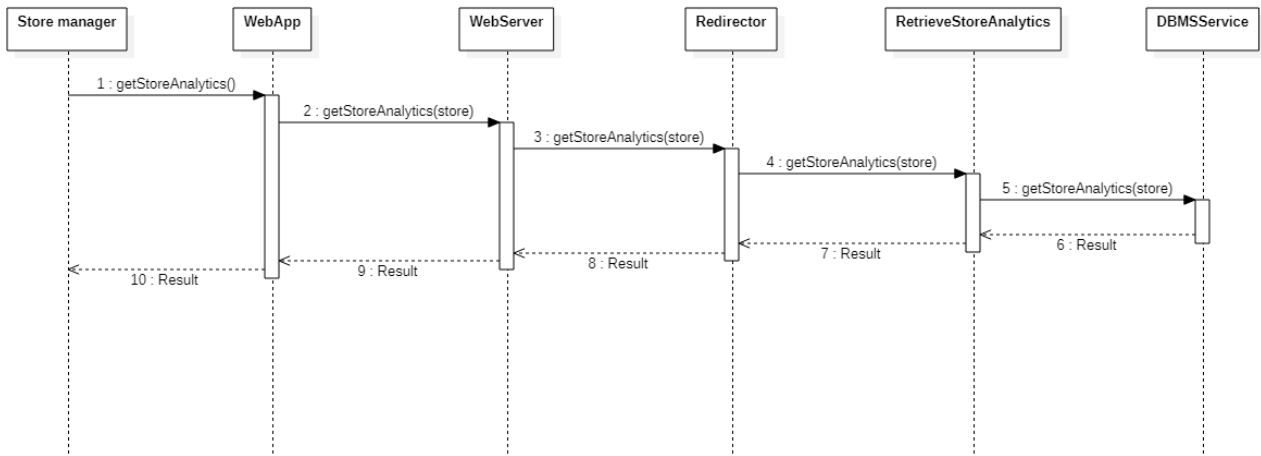
DBMS Service saves the ticket in the database and also retrieves from it some useful information such as the estimated waiting time and the length of the queue, then sends this information back to TakeATicketService.

TakeATicketService sends back to the User the Ticket and information about the estimated waiting time and the length of the queue.

This time, the Customer is not asked to choose a grocery store because a ticket distributor is associated to exactly one store and so there is no choice. Of course the Customer is not asked the means of transport because if he/she is using the ticket distributor, he/she already has reached the store.

The TicketDistributorApp also provides with the ticket information on the estimated waiting time and the length of the queue: this information will appear on the screen of the Ticket Distributor so that the Customer can read it.

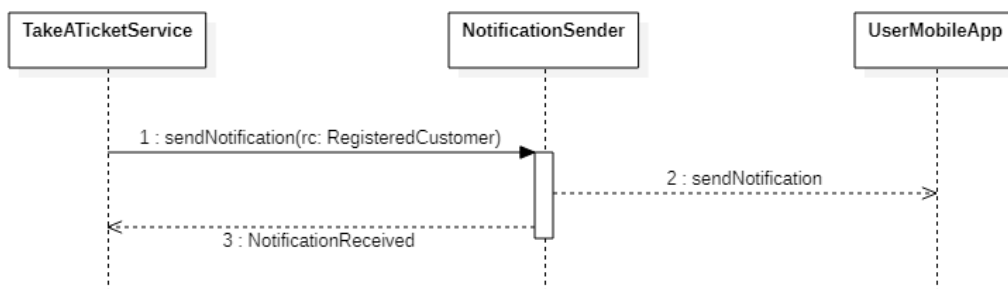
2.4.8 Store manager retrieves store analytics



The Store Manager asks for the analytics to the WebApp. The Request arrives via WebServer and via Redirector to RetrieveStoreAnalytics which queries the DB via DBMS Service to retrieve the information related to the store. The result is forwarded back to the WebApp.

The WebApp is able to understand which is the selected stores because the Store manager asks for the information starting from the page dedicated to that store.

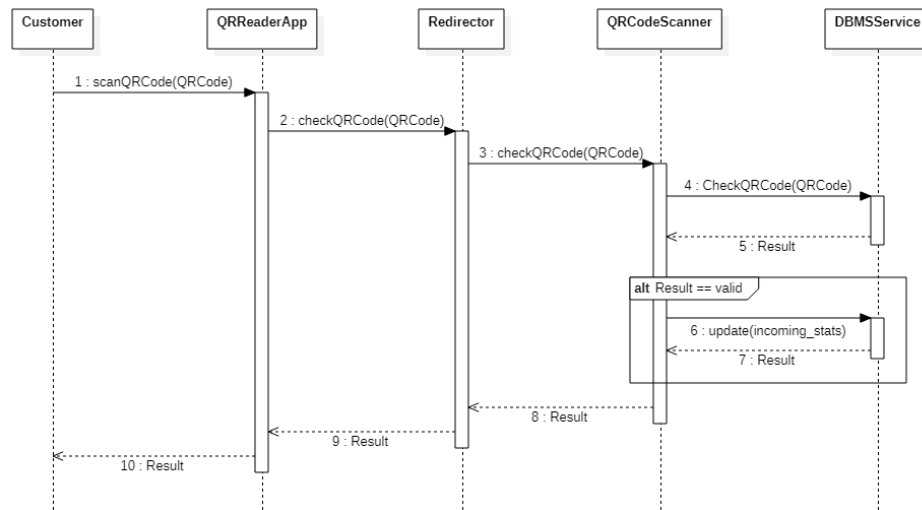
2.4.9 TakeATicketService sends a notification to Registrered Customer to allert him/her that it's time to head to the store



10 minutes before that the estimated waiting time is equal to the time needed by Registered Customer to reach the store, the component TakeATicketService calls the method sendNotification, provided the interface offered by NotificationSender, specifying the Registered Customer who will receive the notification.

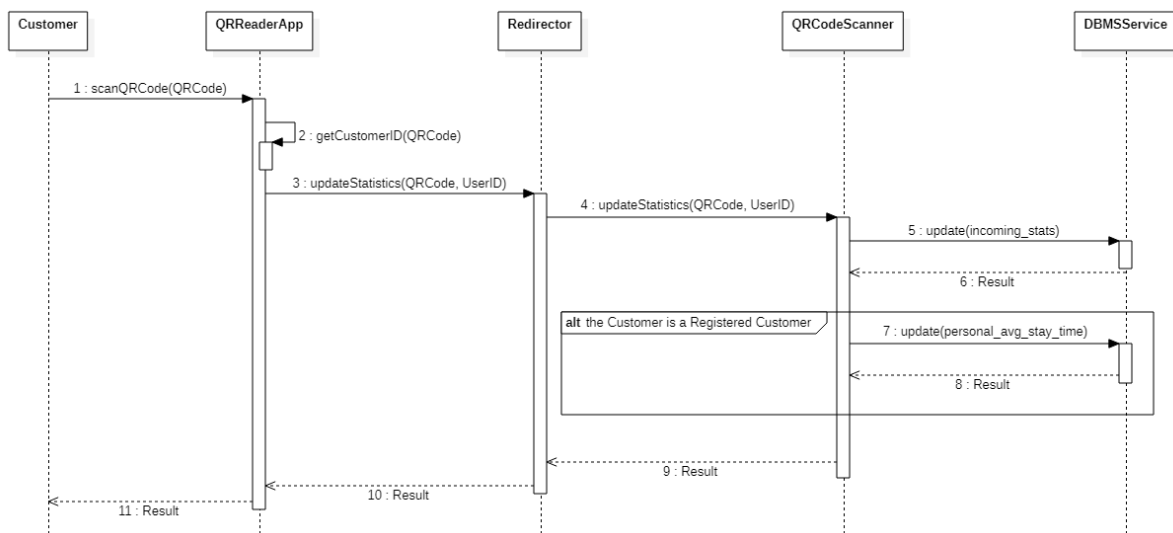
The NotificationSender sends the notification (an asynchronous message) to the UserMobileApp and sends a message back to the TakeATicketService to confirm that the notification has been sent. The notification is an asynchronous message because, if the smartphone of the RegisteredCustomer is turned off, we don't expect to receive the confirmation that the message has been received. The notification that reminds the Customer of a booked visit is very similar to this one.

2.4.10 A Customer enters in a grocery store



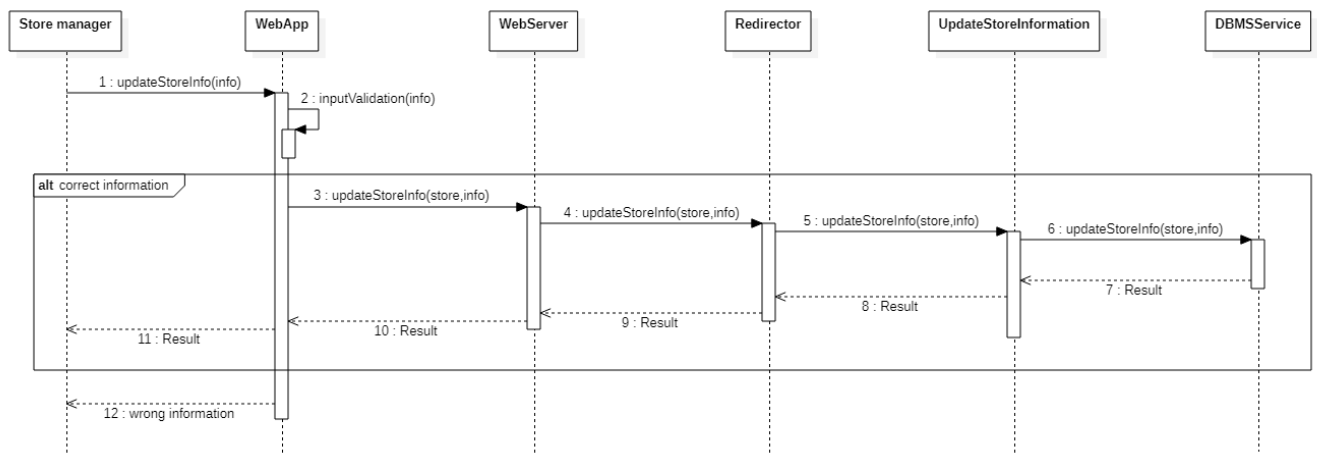
The Customer scans the QRCode of the ticket using the QR Reader of the store and then the QRReaderApp sends the QRCode to the QRCodeScanner via Redirector. The QRCodeScanner then queries the Database using the DBMS Service to check if the QR Code is a valid QRCode (so a QRCode that is associated to that store and to an active ticket). Then the DBMS Service uses the ticket information to update the stats (for instance the number of people that entered the store in that day).

2.4.11 A Customer leaves the grocery store



The Customer scans the QRCode of the ticket using the QR Reader of the store, then the QRReaderApp check if the customer is a Registered Customer (retrieving the CustomerID and checking if is not null). If the Customer is a Registered Customer, QRCodeScanner not only asks to the DBMS Service to update general statistics but also the specific stats of the Registered Customer.

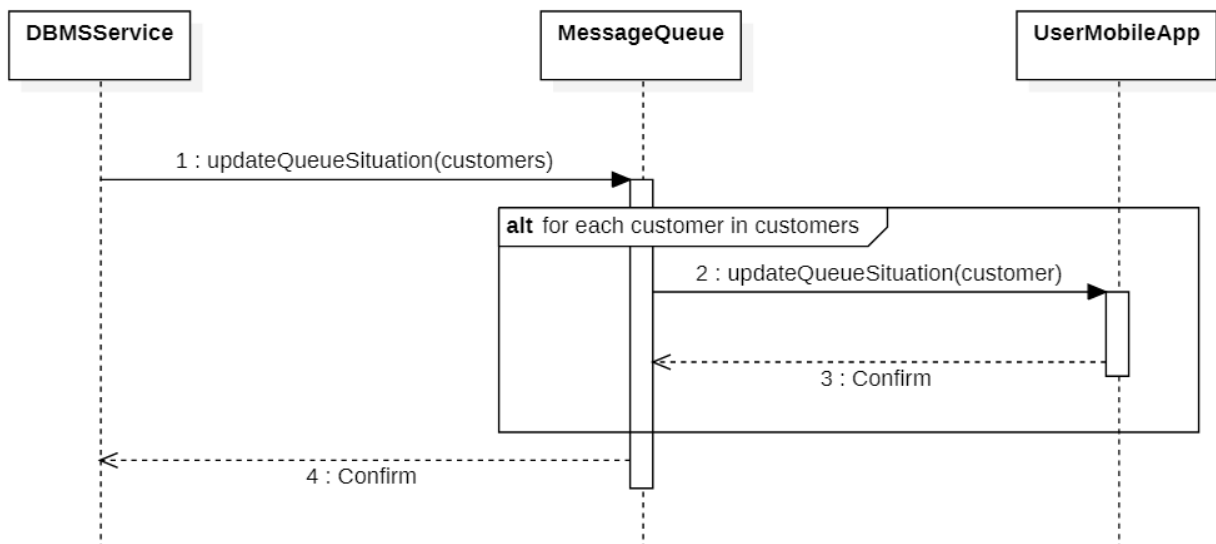
2.4.12 Store Manager updates store information



The store manager asks to the WebApp to update the store information, the request is forwarded to the Web Server, then to the Redirector, to the UpdateStoreInformation and finally to the DBMS Service that changes the store information in the database.

In WebApp, inputValidation controls that, also with new information, the number of people that are allowed to enter the store is greater than 0, that the list of goods sold in the store is not null and that the closing hour is strictly after the opening hour of the store. If these 3 conditions are not respected, the system doesn't allow the update of the store information and WebApp sends an error message to the store manager.

2.4.13 Customers can see queue situation in real time

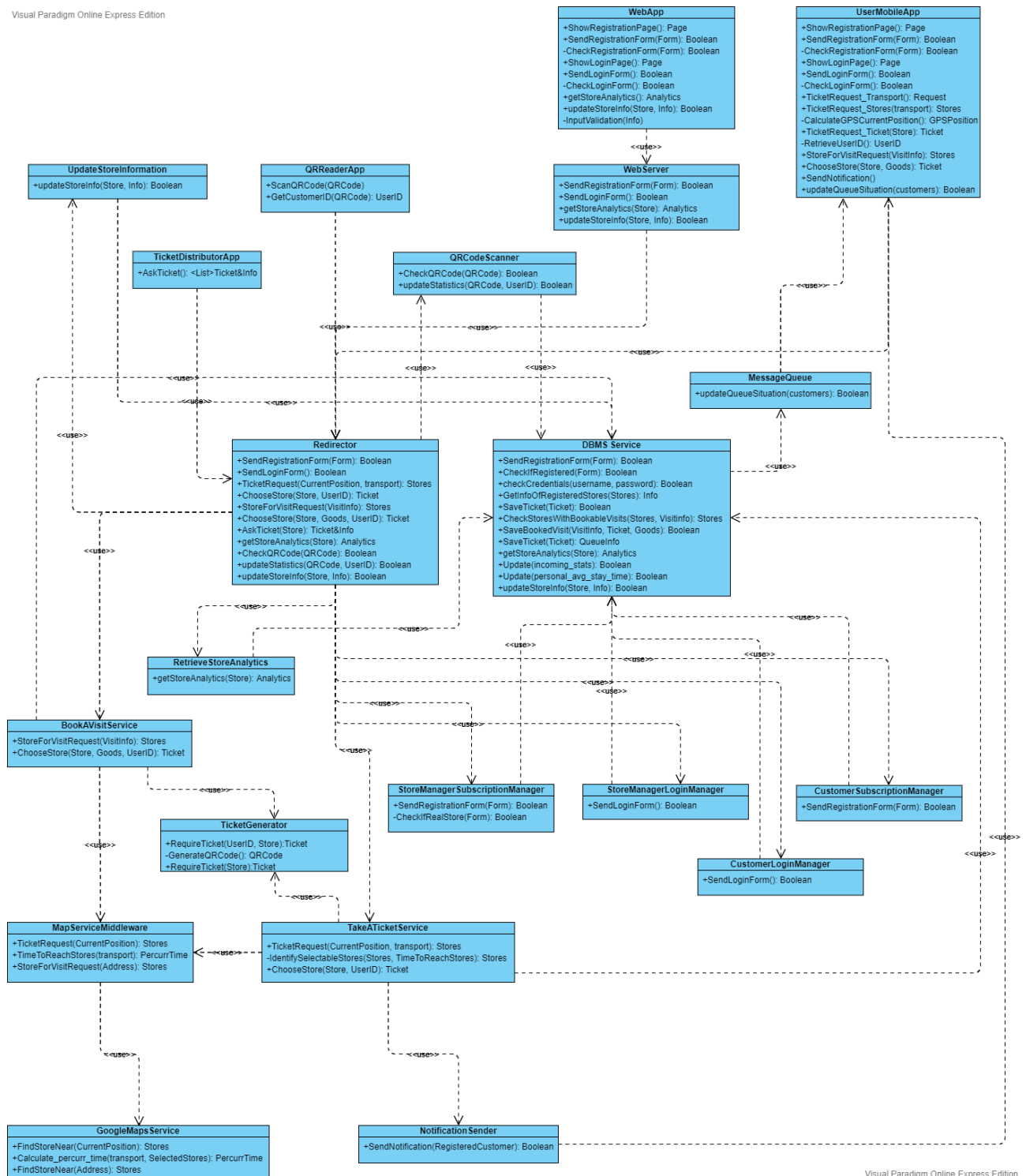


When the situation of a queue changes in the database (for example because the QR code of a ticket associated to that queue is scanned or because a ticket expires), the DBMS Service notifies via MessageQueue all the Customers that are in that queue. In this way, Customers can see the situation of the queue in real time using the mobile app.

2.5 Component interfaces

For legibility reasons, *CustomerService*, *StoreManagerService*, *CustomerAccessManager*, *StoreManagerAccessManager* and *QR Service* have not been introduced

Visual Paradigm Online Express Edition



2.6 Selected architectural styles and patterns

Clup will be a 3-tiers application and the 3 tiers will be: Client (thin), Server (considering the Application Server and the Web Server as one) and Database.

These 3 tiers will be connected via internet and will exchange data according to the following protocol:

Concerning the client-server architecture, Store managers will use the dedicated web app which communicates with the web server which communicates with the application server.

Registered Customers will instead use the mobile application that communicated directly with the application server. Finally, the application server will act as a client querying the database server.

All the communications exploit the HTTP protocol, using TLS when dealing with sensitive data in order to guarantee the security and the reliability of the connection and also to authenticate the identity of the communicating parties.

Regarding the format in which the data will be transmitted, JSON will be used because it is suitable for the data interchange between client and server and, despite its simplicity, it is enough to satisfy our purposes.

Design Patterns

MVC (Model–view–controller)

Model-View-Controller is a software design pattern that it's used to develop user interfaces. MVC divides the program logic into 3 interconnected elements. This is done to separate the internal representation of the information (model) from the way in which information is presented to the user.

Model: it is the application's dynamic data structure, independent of the user interface. It directly manages data, logic and rules of the application.

View: representation of the information in any form (chart, diagram, tables etc). Multiple views of the same information are possible.

Controller: accepts inputs from the user and converts them into commands for the model or for the view.

Façade

The Façade pattern provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use for the user. In our system, the Redirector component covers the role of the façade in the communication between the client and the application server.

Structuring a system into subsystems helps reduce complexity. A common design goal is to minimize the communication and dependencies between subsystems. One way to achieve this goal is to introduce a facade object that provides a single, simplified interface to the more general facilities of a subsystem.

Observer

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It's used by the MVC pattern.

State

Assigns a state to an object and allows the object to alter its behavior when its internal state changes.

Let's consider the class Ticket (that represents a grocery store's ticket). A Ticket object can be in one of several different states: Issued, EntranceScanned, ExitScanned, Expired. When a Ticket object receives requests from other objects, it responds differently depending on its current state. The State pattern describes how Ticket can exhibit different behavior in each state.

2.7 Other design decisions

Thin Client

We opted for Thin Clients designed to communicate with the Server and to use functionalities offered by it. Just little of the logic is implemented client-side, for instance the functions that check if the fields of a form are filled with proper values and that mandatory fields are not left empty before to send a form. However, this is not enough to consider our clients Fat Clients. The choice of using Thin Clients it's also due to the fact that the Clup Mobile App is intended to run on all kind of smartphones, also the once with underperforming hardware.

Relational Database

We opted for a relational database because this is the best choice to specify relationships between entities (such as Customers and Tickets, Store Managers and Stores) and to deal with many transactions. Relational databases are also a good choice to perform data analysis in an easy way on a large number of data. Moreover, using a relational database, it's possible to guarantee the ACID properties for the transactions.

Ticket Distributors and QR Readers dedicated apps

Considering that Customers can get tickets from Ticket Distributors and that they have to scan QR Code using the QR Code Readers that are in front of the stores, we need for sure a simple dedicated app for Ticket Distributors (in order to let Customers to get tickets using the functionalities provided by TakeATicketService component) and also for QR Readers (an internal application that scans the QR Code and retrieves simple information such as the time of the visit to send to the DBMS Service in order to let it update the information of stores and customers).

3 User interface design

3.1 Mockups

In the following, different mockups will show how most important pages of the mobile application used by customer and of the dedicated web application used by the store manager should look.

3.1.1 Customer Home Page

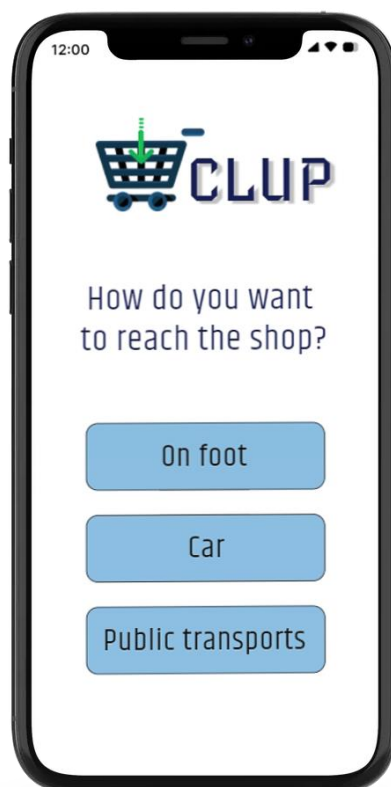


If the customer is already registered to the system, he/she can insert his/her username and password in order to log in. Otherwise, the customer can click on the “Sign Up” button in order to subscribe to the system.

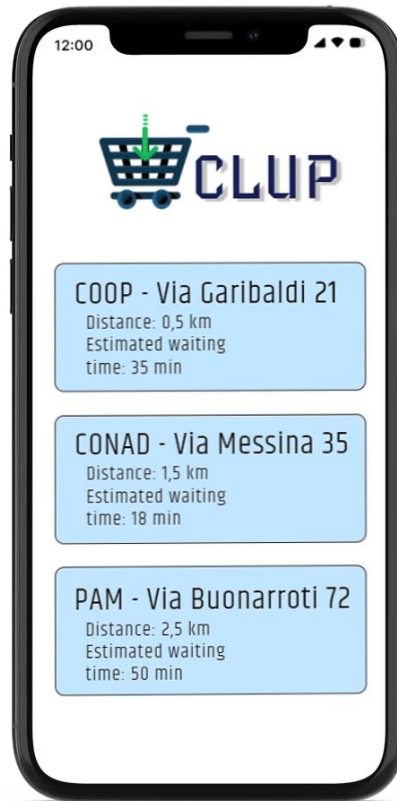
3.1.2 Registered Customer chooses if he/she wants to get a ticket o to book a visit



3.1.3 Registered Customer gets a ticket – Registered Customer chooses the mode of transport



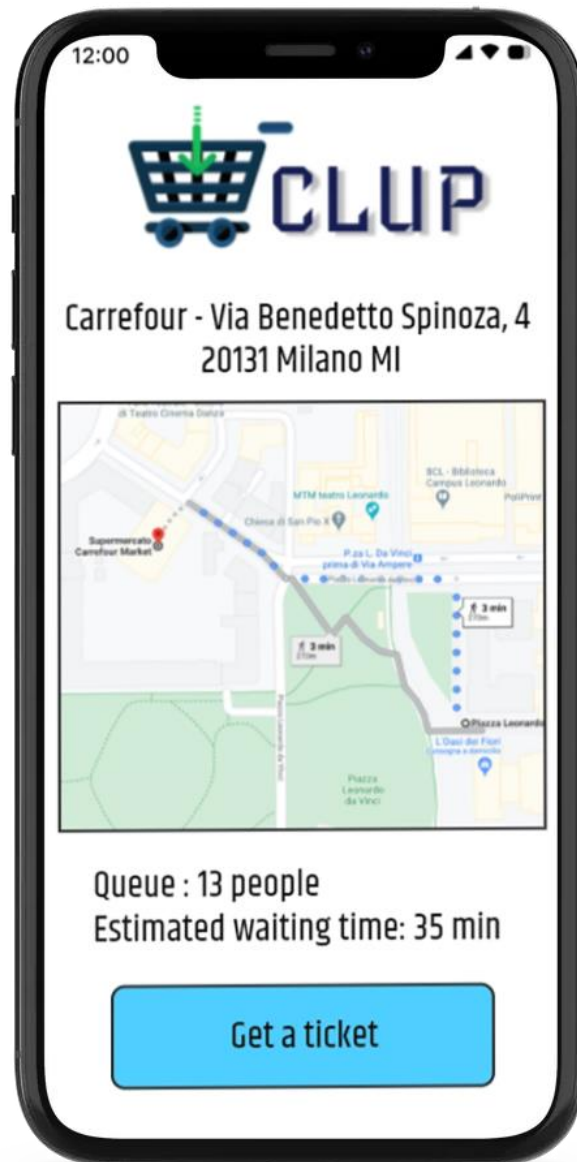
3.1.4 Registered Customer gets a ticket – Registered Customer chooses a store



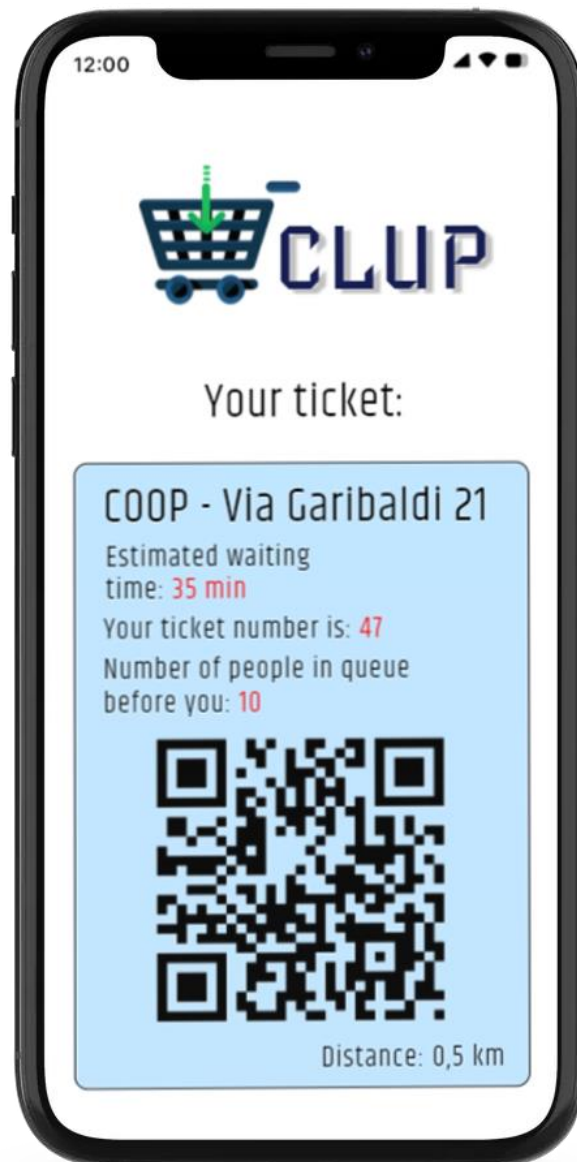
The System shows all the stores in the municipality, but only open grocery stores in which the customer will be allowed to enter before their closure are selectable. Using information about the mode of transport, the Customer current position and the estimated waiting time of each store, the System will not let the customer to select also that stores that will close before that the Registered Customer will be allowed to enter. Unselectable stores are always showed at the bottom of the list, after the once that are selectable.

Also stores that cannot be selected (because they are closed or they will close before that the customer will be allowed to enter) are showed in order to let the Customer understand if there are stores registered to the system in the municipality where he/she currently is.

3.1.5 Registered Customer gets a ticket – Registered Customer sees the store on the map and gets a ticket



3.1.6 Registered Customer gets a ticket – Registered Customer's Ticket




“Estimated waiting time” and “Number of people in queue before you” are in-real-time information.

3.1.7 Registered Customer books a visit – Registered Customer selects data and time of the visit and inserts the address where he/she will be located just before the moment of the visit

The image shows a mobile application interface for 'CLUP'. At the top, the status bar shows the time 12:00. The app's logo, featuring a shopping cart with a green arrow pointing down and the word 'CLUP' in blue, is centered. Below the logo, the text 'Address where you will be just before the moment of the visit' is displayed above a text input field. Underneath the address field is the label 'Date' followed by a calendar widget for December 2020. The calendar shows the 16th as the selected date. Below the calendar is the label 'Time :' followed by a time selection input field. At the bottom of the screen is a large blue button labeled 'Find stores'.

12:00



Address where you will be just before the moment of the visit

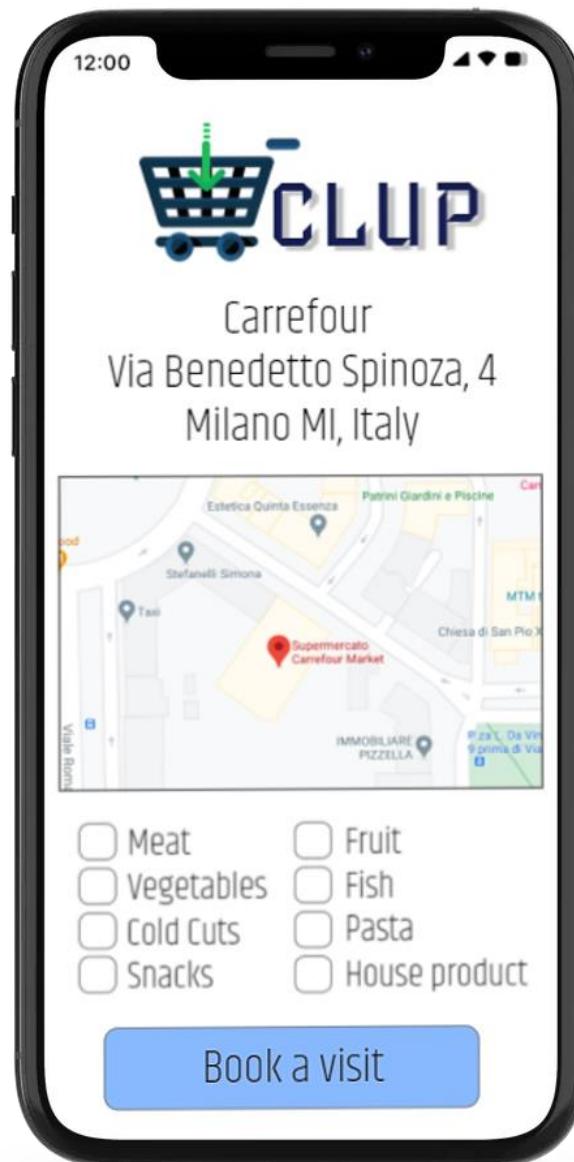
Date

dicembre 2020

| lu | ma | me | gi | ve | sa | do |
|----|----|----|----|----|----|----|
| 30 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Time :

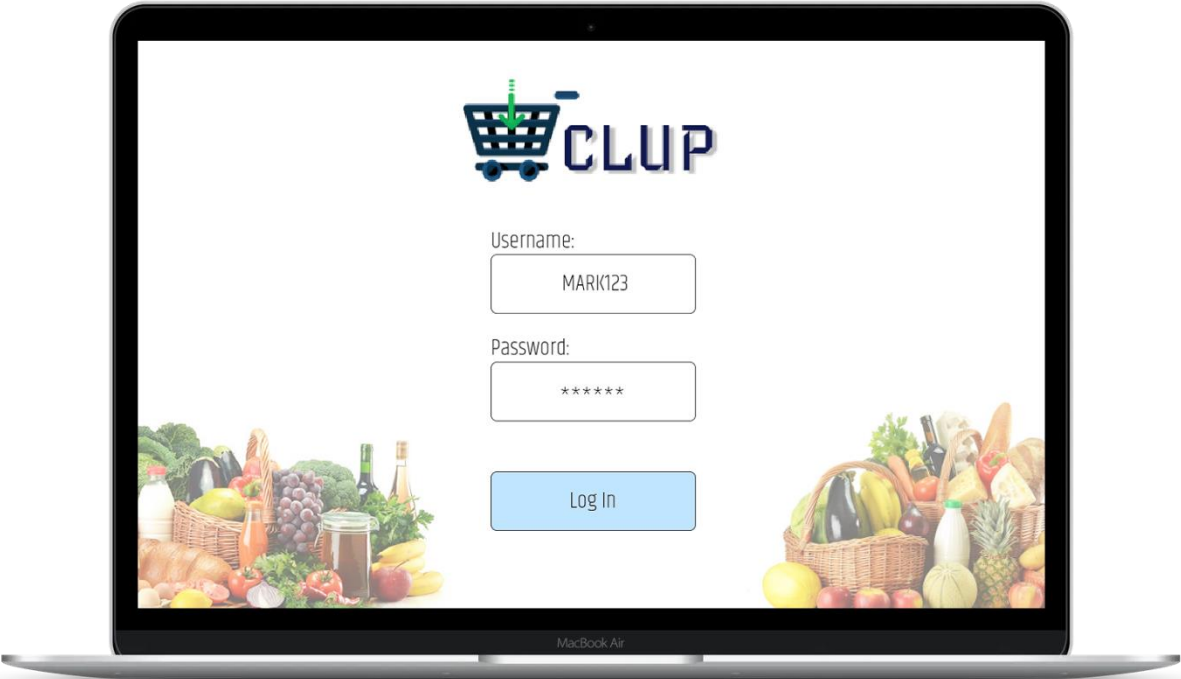
3.1.8 - Registered Customer books a visit – Registered Customer selects a store and (optionally) the kind of goods that he/she intends to buy



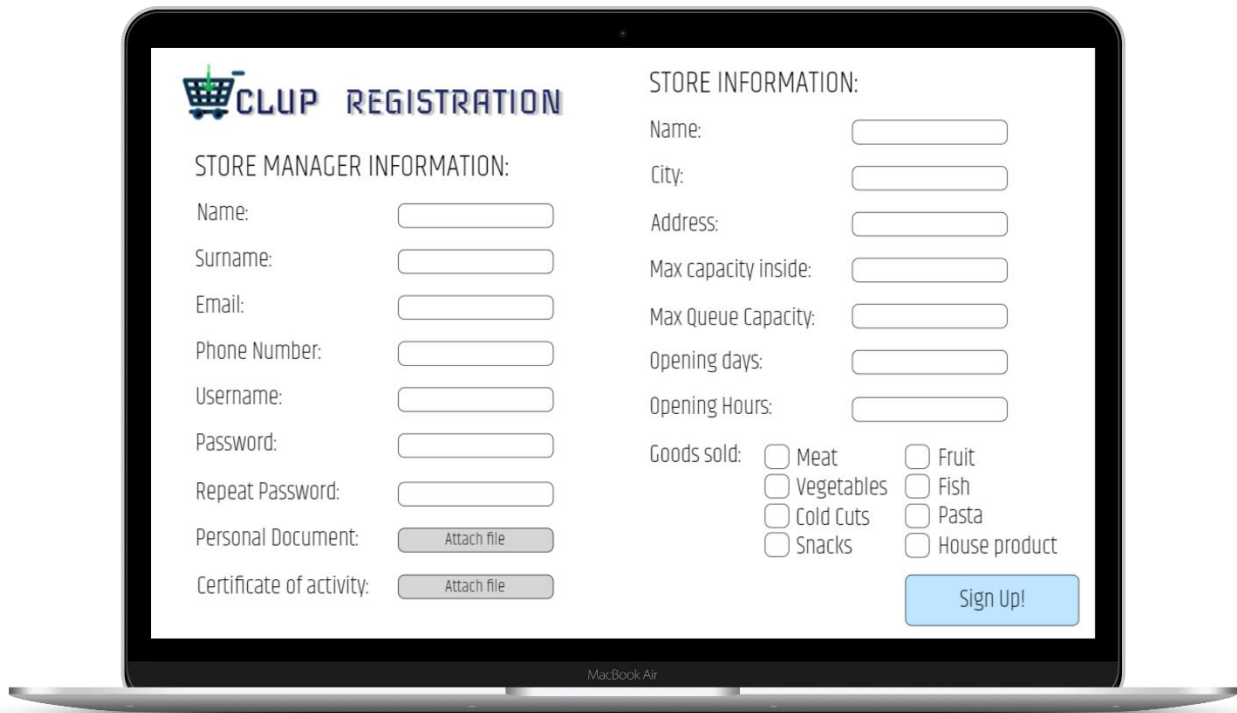
3.1.9 – Store Manager home page



3.1.10 – Store Manager log in



3.1.11 – Store manager registration form



The screenshot shows a laptop displaying the 'CLUP REGISTRATION' form. The form is divided into two main sections: 'STORE MANAGER INFORMATION' and 'STORE INFORMATION'. The 'STORE MANAGER INFORMATION' section includes fields for Name, Surname, Email, Phone Number, Username, Password, Repeat Password, Personal Document (with an 'Attach file' button), and Certificate of activity (with an 'Attach file' button). The 'STORE INFORMATION' section includes fields for Name, City, Address, Max capacity inside, Max Queue Capacity, Opening days, and Opening Hours. Below these fields is a 'Goods sold' section with checkboxes for Meat, Vegetables, Cold Cuts, Snacks, Fruit, Fish, Pasta, and House product. A 'Sign Up!' button is located at the bottom right of the form.

CLUP REGISTRATION

STORE MANAGER INFORMATION:

Name:

Surname:

Email:

Phone Number:

Username:

Password:

Repeat Password:

Personal Document:

Certificate of activity:

STORE INFORMATION:

Name:

City:

Address:

Max capacity inside:

Max Queue Capacity:

Opening days:

Opening Hours:

Goods sold:

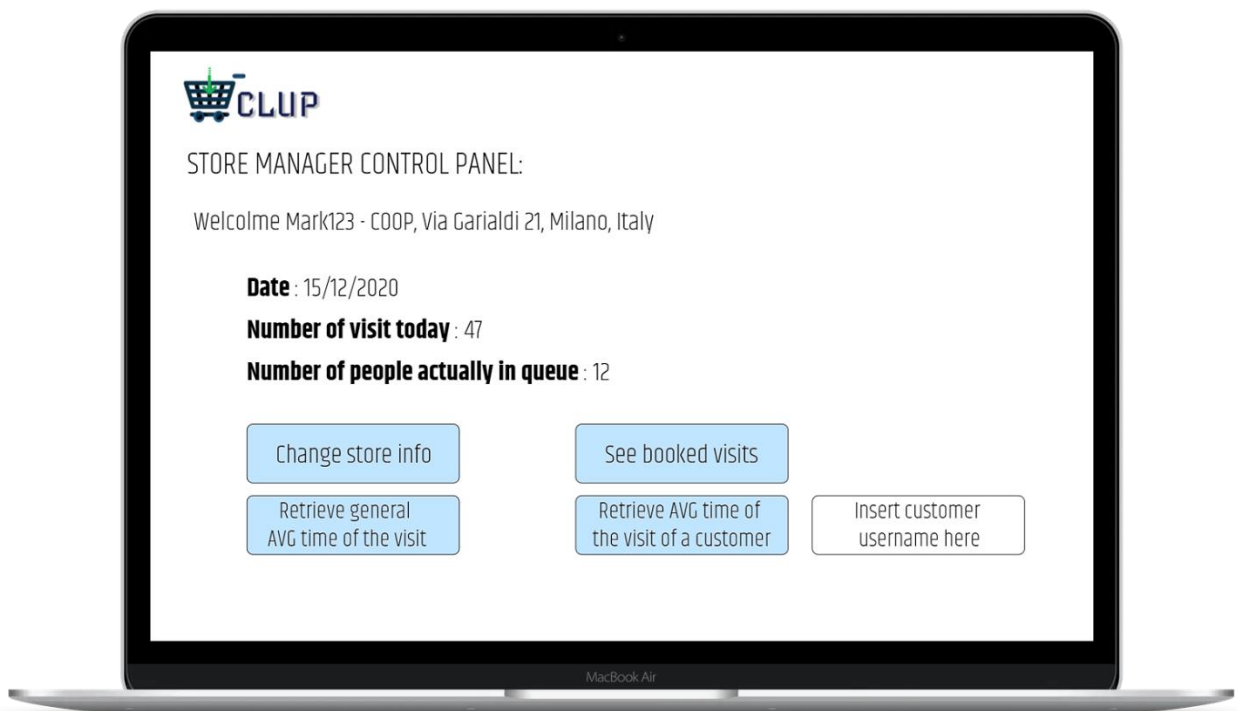
☐ Meat ☐ Fruit

☐ Vegetables ☐ Fish

☐ Cold Cuts ☐ Pasta

☐ Snacks ☐ House product

3.1.12 – Store Manager control panel



The screenshot shows a laptop displaying the 'CLUP STORE MANAGER CONTROL PANEL'. The panel displays the store name 'Welcolme Mark123 - COOP, Via Garialdi 21, Milano, Italy'. Below this, it shows the date '15/12/2020', the number of visits today '47', and the number of people actually in queue '12'. There are five buttons: 'Change store info', 'See booked visits', 'Retrieve general AVG time of the visit', 'Retrieve AVG time of the visit of a customer', and 'Insert customer username here'.

CLUP

STORE MANAGER CONTROL PANEL:

Welcolme Mark123 - COOP, Via Garialdi 21, Milano, Italy

Date : 15/12/2020

Number of visit today : 47

Number of people actually in queue : 12

4 Requirements traceability

[Each requirement is associated to at least one component and each component is associated to at least one requirement]

R1 - When a Customer tries to use the application Clup, the System controls if the device used has a GPS sensor integrated. If not, the Customer is notified that he/she cannot use the application because he/she doesn't have the required technology

Components – UserMobileApp

R2 - The System shall ask to the Customer the permission to access the GPS position. If the Customer doesn't want to give the authorization, the System informs him/her that he/she can't use the service

Components – UserMobileApp

R3 - A Registered Customer must be able to log in to the System using his/her credential

Components – UserMobileApp, Redirector, CustomerLoginManager, DBMS Service

R4 - A Registered Customer can use the system only if he/she is logged in

Components - UserMobileApp, Redirector, CustomerLoginManager, DBMS Service

R5 - A Registered Customer cannot take more than one ticket at a time

Components - UserMobileApp, Redirector, TakeATicketService, DBMS Service

R6 - A Registered Customer can book more than one visit at a time but not in the same date and the same time

Components - UserMobileApp, Redirector, BookAVisitService, DBMS Service

R7 - A Registered Customer can cancel the request of a ticket losing his/her turn and also the request of a booked visit

Components - UserMobileApp, Redirector, TakeATicketService, BookAVisitService, DBMS Service

R8 - A Registered Customer must be able to see the real time situation of the queue using his/her smartphone

Components - UserMobileApp, MessageQueue, DBMS Service

R9 - Everyone (both Registered and Unregistered Customers) must have the possibility to get a ticket, or using the dedicated app on the smartphone or using the ticket distributor that all the stores subscribed to the System must have

Components - UserMobileApp, Redirector, TakeATicketService, TicketDistributorApp, DBMS Service

R10 - When a Registered Customer wants to get a ticket online, the System asks him/her if he/she intends to reach the store by car, using public transports or on foot in order to better calculate the estimated time needed by the Registered Customer to reach the store

Components - UserMobileApp, Redirector, TakeATicketService, MapServiceMiddleware, GoogleMaps Service

R11 - When a Registered Customer wants to get a ticket online, the System asks him/her to choose from a list of stores that are currently open and located in the same municipality of the current Registered Customer position. Stores that will close before that the customer will be allowed to enter are not selectable.

Components - UserMobileApp, Redirector TakeATicketService, MapServiceMiddleware, GoogleMaps Service

R12 - When a Customer gets a Ticket (both online and in presence), the System generates an unique QR code associated with the ticket

Components - UserMobileApp, TicketDistributorApp, Redirector TakeATicketService, BookAVisitService, TicketGenerator, DBMS Service

R13 - When a Registered Customer books a visit, he/she can select the category of goods that he/she intends to buy

Components - UserMobileApp, Redirector, BookAVisitService, DBMS Service

R14 - When a Registered Customer books a visit, the System asks him/her to insert data and time of the visit and also the address of the place where the Registered Customer will be just before the moment of the visit in order to identify and then show on screen the list of stores having a bookable visit on the selected date and time and near the selected address

Components - UserMobileApp, Redirector, BookAVisitService, DBMSService, MapServiceMiddleware, GoogleMaps Service

R15 - When a Registered Customer gets a ticket using his/her smartphone, then the System sends him/her a notification 10 minutes before that the estimated waiting time is equal to the time needed by Registered Customer to reach the store in order to notify him/her that is time to head to the store

Components - TakeATicketService, NotificationSender, UserMobileApp

R16 - When a Registered Customer books a visit, 2 hours before the time of the visit the System sends a notification to Registered Customer to remember him/her the booked visit. If the visit is scheduled for the 10:00 AM or earlier, the System sends the notification at 6:00 PM of the day before.

Components - BookAVisitService, NotificationSender, UserMobileApp

R17 - If a Registered Customer booked a visit online and then his/her smartphone is turned off when the System sends the notification to remind the booked visit, then the Registered Customer will be able to see the notification when he/she turns on the device

Components - BookAVisitService, NotificationSender, UserMobileApp

R18 - If a Customer is not in front of the store when it's his/her turn, the System waits for 2 minutes then Customer loses his/her turn and the turn passes to the next Customer in line

Components - DBMS Service, MessageQueue, UserMobileApp

R19 - In order to sign up, a Store manager has to prove that he/she they really runs a grocery shop sending documents related to the management of the store itself and licenses (required documents are later specified in the Use cases section)

Components - WebApp, WebServer, Redirector, StoreManagerSubscriptionManager, DBMS Service

R20 - A registered Store manager must be able to log in to the System using his/her credentials

Components - WebApp, WebServer, Redirector, StoreLoginManager, DBMS Service

R21 - A Store manager can use the system only if he/she is logged in

Components - WebApp, WebServer, Redirector, StoreLoginManager, DBMS Service

R22 - Using the WebApp, a Store manager can select the number of customers allowed to be contemporarily in queue in front of the store

Components - WebApp, WebServer, Redirector, UpdateStoreInformation, StoreManagerSubscriptionManager, DBMS Service

R23 - Using the WebApp, Store managers can select the number of customers allowed to be contemporarily inside the store

Components - WebApp, WebServer, Redirector, UpdateStoreInformation, StoreManagerSubscriptionManager, DBMS Service

R24 - Using the WebApp, Store managers can see information about their grocery store such as the average time of the visit (both general and for a specific Registered Customer), the number of people that entered the store day by day and the booked visits

Components - WebApp, WebServer, Redirector, RetrieveStoreAnalytics, DBMS Service

R25 - The System is able to use GPS, information about traffic condition in real time and information about movement methods chosen by Registered Customer to calculate the estimated time needed by a Registered Customer to reach the store

Components - MapServiceMiddleWare, GoogleMapsService

R26 - The System must provide to a Registered Customer the estimated waiting time for the line up in real time

Components - DBMS Service, MessageQueue, UserMobileApp

R27 - When a Registered Customer wants to get a ticket online, the System shows also stores that cannot be selected (because they are closed or they will close before that the customer will be allowed to enter) in order to let the Customer understander if there are stores registered to the system in the municipality where he/she currently is. Of course, unselectable stores cannot be selected and they are showed at the bottom of the list, after the once that are selectable (if exists).

Components - UserMobileApp, Redirector, TakeATicketService, MapServiceMiddleware, GoogleMapsService, DBMS Service

R28 - Customers should be able to scan the QR Code on their tickets both when they enter and when they leave the store. The system should be able to update queues and stores information with data taken from the scan of the QR Codes.

Components - QRCodeScanner, DBMS Service

5 Implementation, integrations and test plan

5.1 Overview

It is almost impossible to develop a bug-free software, especially if the software is big and complex as Clup is. For this reason, in order to find the highest possible number of bugs, the phase of implementation, integration and testing is very important and should be planned and performed carefully.

In the following, we will present and explain our implementation, integration and test plan.

Of course, it's known that testing cannot guarantee the total absence of bugs in the software, anyway it is useful to find a certain number of them and to make the system to work better.

5.2 Implementation plan

The entire System (both server and client side) will be developed, tested and integrated following the bottom up approach.

The bottom up approach is in general the easiest one: it allows you to build the software incrementally and so to test the application step by step, also if not all the functionalities have been already implemented. The bottom up approach also facilitates bug tracking and allows you to develop a certain number of components, to test and integrate them and then to proceed implementing, integrating and testing other components.

Server side and Client side will be implemented in parallel in order to test them together.

We will start implementing CustomerAccessManager, StoreManagerAccessManager and StoreManagerService, which are the most independent components. They will use the external component DBMS Service and will need drivers to simulate the call of their methods.

In parallel to these 3 components it's possible to develop, client-side, UserMobileApp, WebApp and WebServer, which will be useful for the testing of components that we will implement in the following.

Then, we will develop NotificationSender, TicketGenerator and MapServiceMiddleware. In a bottom up approach is fundamental to develop first that components that are used by the others and these 3 components are used both by TakeATicketService and BookAVisitService. If the UserMobileApp has already been developed, it's possible to test NotificationSender without the usage of a stub (stubs are usually not used in the bottom up approach).

Once that these 3 components have been implemented and tested, it is possible to develop and test 2 of the most important components of the Application Server: TakeATicketService and BookAVisitService. To test these 2 components, we will require drivers in order to simulate the calls of their methods, this because client and server are not already connected (the Redirector has not been implemented until now).

In parallel to these 3 components, it's possible to develop, client-side, TicketDistributorApp and QRReaderAPP.

Then, we can proceed developing and testing MessageQueue (that uses the interface offered by UserMobileApp, that has been already implemented) and QRCodeScanner.

Finally, we have the implementation of the Redirector that is the component responsible for most of the client-server connections.

It is important to underline that we will not test GoogleMapsService and DBMS Service (which are external components) but we will test the interaction between these external components and our system.

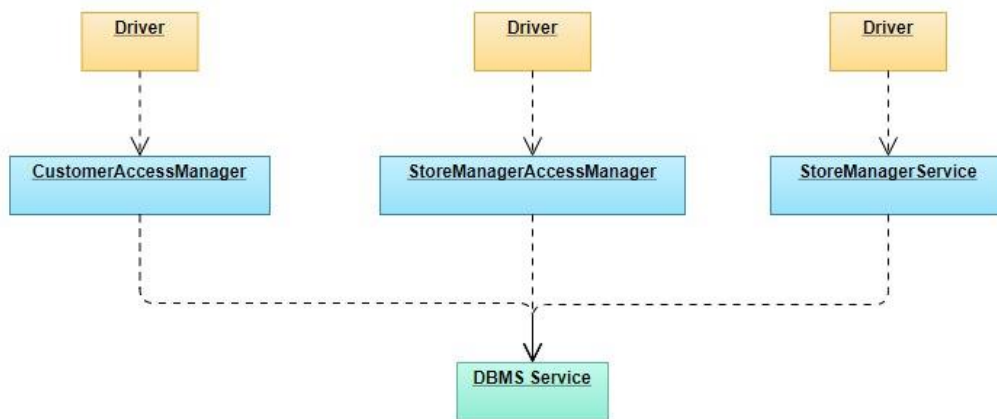
So, in the end, the order of the implementation of the components server side will be:

- 1) CustomerAccessManager, StoreManagerAccessManager, StoreManagerService
- 2) NotificationSender, TicketGenerator, MapServiceMiddleware
- 3) TakeATicketService, BookAVisitService
- 4) MessageQueue, QRCodeScanner
- 5) Redirector

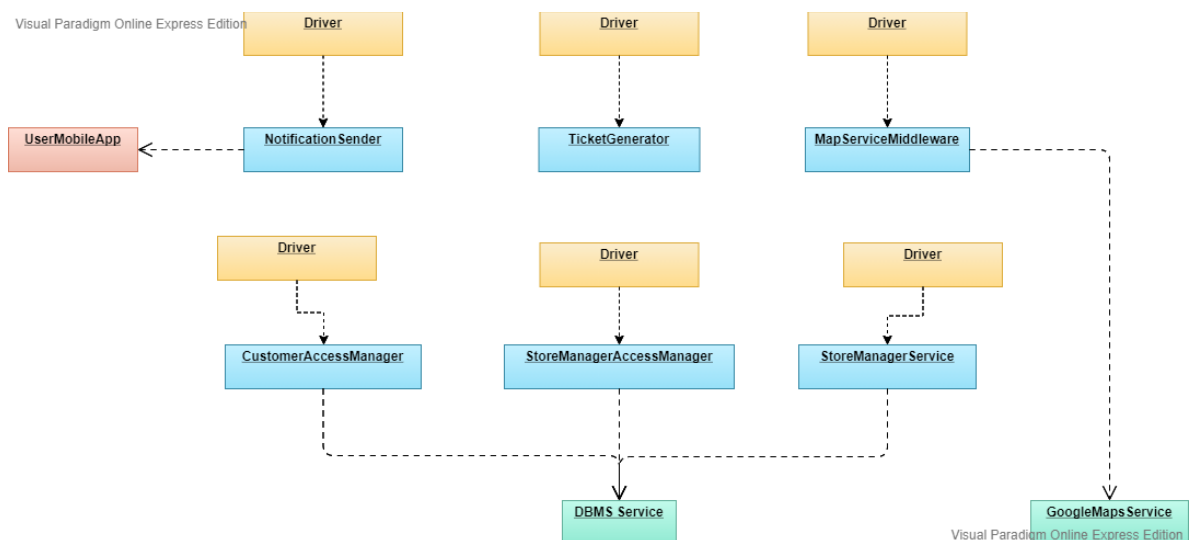
5.3 Integration strategy

To implement and test all the different functionalities offered by the system, a bottom-up approach will be used. In the following, we will describe how the process of implementation and integration testing takes place following a bottom-up approach.

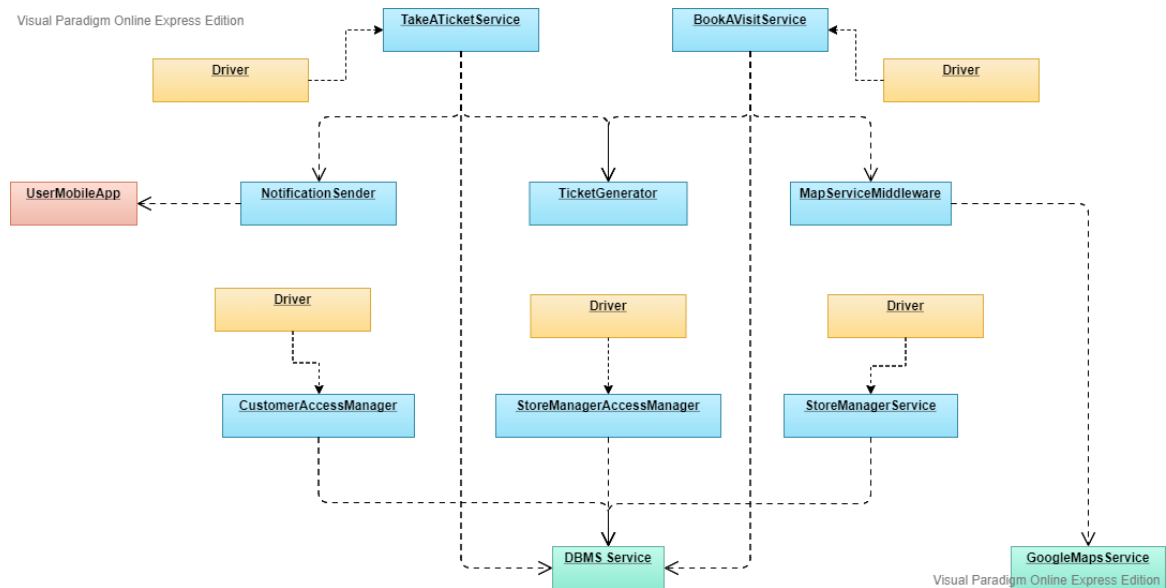
- 1) First CustomerAccessManager, StoreManagerAccessManager and StoreManagerService are implemented, unit tested and integrated with the external component “DBMS Service”. Drivers will be used to simulate components that are still under implementation and that uses the functionalities offered by the 3 implemented components. In parallel to these 3 components, it is possible to implement and unit-test UserMobileApp, WebApp and WebServer client-side.



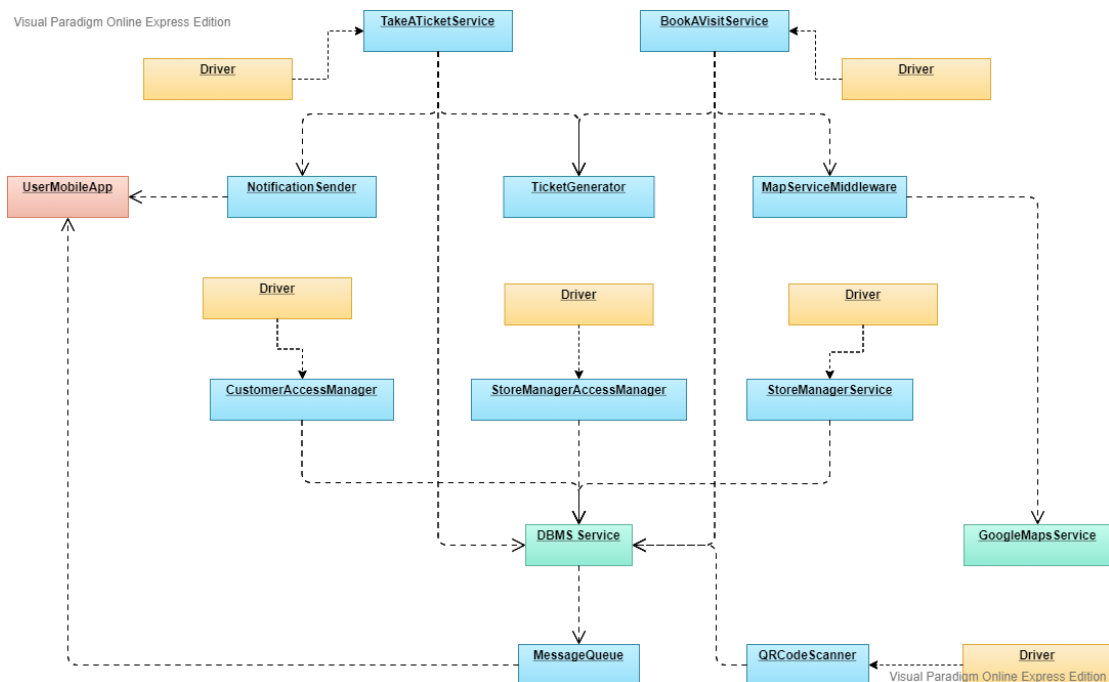
- 2) Then, NotificationSender, TicketGenerator and MapServiceMiddleware are implemented, unit tested and integrated to the system. These components will be used by TakeATicketService and BookAVisitService, the 2 key-components of the system.



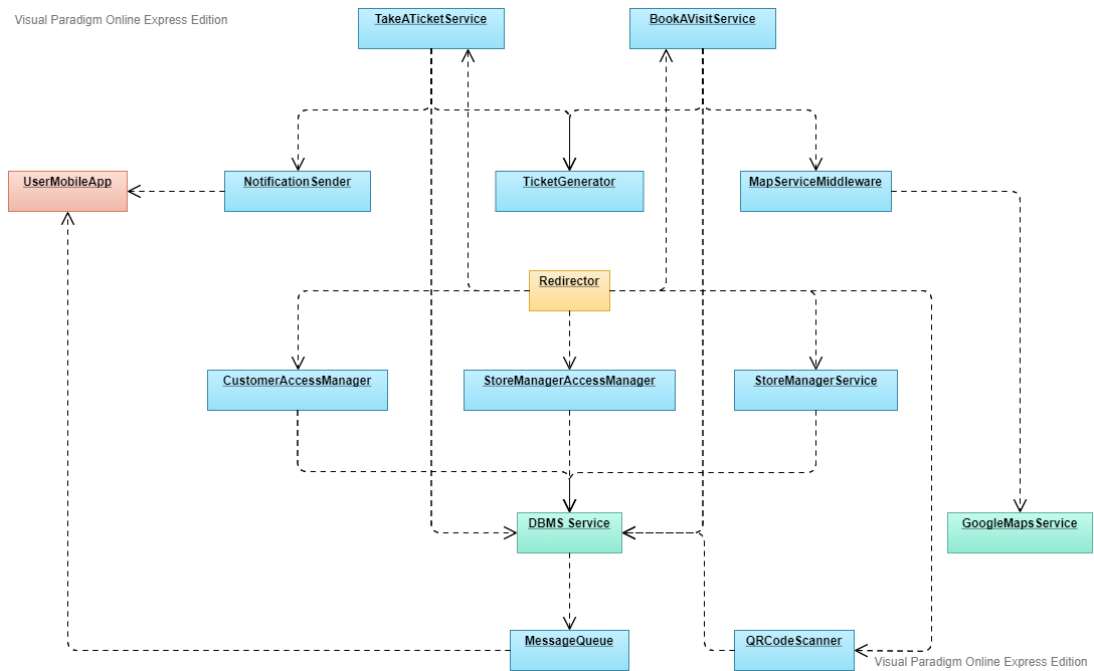
- 3) Then, the implementation and integration of the TakeATicketService and BookAVisitService takes place. These two components can be tested in parallel because there are no direct dependencies between them. In parallel to these 3 components, it is possible to implement and unit-test TicketDistributorApp and QRReaderAPP client-side.



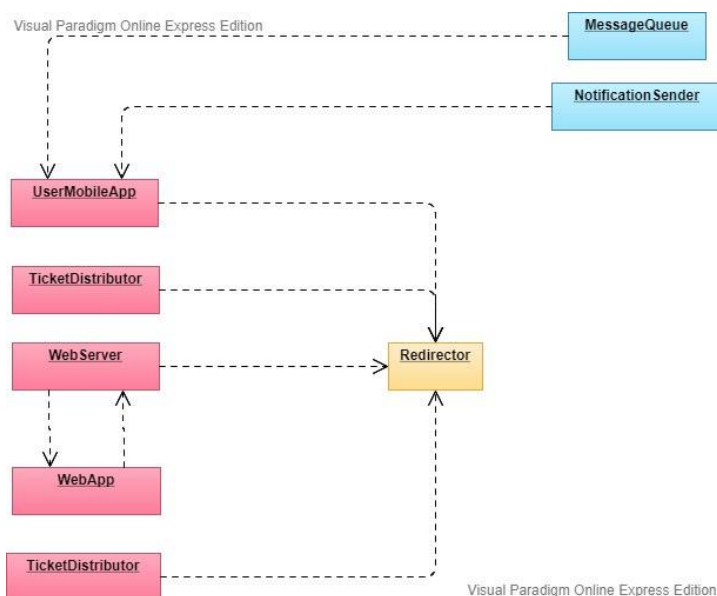
- 4) Then, we can proceed developing and testing MessageQueue (that uses the interface offered by UserMobileApp, that has been already implemented) and QRCodeScanner.



- 5) Then, the different drivers in the previous diagram are substituted by the Redirector which has to be implemented, unit-tested and integrated with the other components of the application server.



- 6) Finally, client side and sever side are integrated. This happens only when all the components of both sides have been implemented and tested (in particular, we cannot do this integration before without the Redirector that is the key-component for the communication between server and client).



Once that all the components have been integrated, we proceed with the system testing in order to verify if the complete application works properly. Our purpose is to check if all the requested functionalities are effectively offered by our system (functional testing) and eventually to identify bottlenecks that affect the performance of our system in terms of throughput and the response time. Finally it's important to perform a Load Testing and a Stress Testing to check how our system behaves in extreme situations.

6 Effort spent

Claudio Alfredo Emanuele

| | |
|--|----|
| Discussion about the design | 2h |
| Purpose & Scope | 1h |
| Architectural design – High Lev Design | 3h |
| Component View | 8h |
| Runtime View | 8h |
| Deployment View | 2h |
| Selected architectural styles and patterns | 2h |
| Mockups Review | 2h |
| Requirements traceability | 2h |
| Integration Strategy | 3h |

Antonio Guadagno

| | |
|---------------------------------|------|
| Discussion about the design | 2h |
| Architectural design - Overview | 2h |
| Component View | 8h |
| Runtime View | 7h |
| Component interfaces | 2h |
| Other design decisions | 1h |
| Mockups | 10h |
| Implementation Plan | 1.5h |

7 References

Course slides

Wikipedia

Oracle web site