Software Engineering II 2020/2021

January 2021

Design Document

CLup: Customers Line-up Software

Version: 2

Authors:

R. Cedeno, D. Cumming, A. Pugliese

Contents

1. INTRODUCTION

1.1. Purpose

The need of avoiding physical interactions in order to prevent the spreading of the COVID-19 disease has put a hard constraint on the businesses that are based on in-person shopping since there are safety restrictions on how many people can be inside of a certain building while there must be a distance of at least one meter between them. To address these constraints, the software design of a specialized application for this purpose will be elaborated in this document.

The software to be developed is aimed to allow users to line-up to virtual queues to enter a store, guaranteeing safe conditions in terms of social distancing inside the building. Additionally, the software will allow users to book a queue number on a specific time slot. Moreover, the application must be able to give suggestions to the users based on supermarkets' availability and their positions, therefore, the application is proposed to be usable on mobile devices.

The software system to be designed must precisely fulfill the following goals:

G1. The in-person grocery shopping must be carried out guaranteeing the safety of customers and employees.

G2. The customers must be able to line-up in a store if they want to enter to shop as soon as possible.

G3. The customers must be able to book a number to make a reservation to enter a specific store at a certain date and time.

G4. If customers want to shop as soon as possible, they must receive suggestions on the most convenient options in terms of distance and waiting time.

G5. If customers want to book a number on a specific store, they must receive suggestions on the most convenient options in terms of availability.

Throughout the design of an application and during the development of it, it is fundamental to have the correct organization to achieve a successful project. To achieve a complete software regarding its purpose, design decisions need to be established from the beginning of the project. As a consequence of this, this document is one of the most important pieces of information that will guide the development team throughout the design process. This document is developed after creating the Requirements Analysis and Specification Document (RASD) which is available at all times to complement this process in an active way.

As defined by the IEEE the design documentation has a great importance which is described in the following paragraph:

> *"Is an integral part of the software development process, and a vital source of information. Technical documentation is a means to make knowledge about a software system explicit that would otherwise only remain implicit knowledge in the heads of the development team that easily gets lost over time. Further, documentation presents information at a higher level of abstraction than the system implementation, which makes it an important means for communication among stakeholders."*

This means that this type of technical documentation is to be used as a source of knowledge that is needed to have a high level of clarity because otherwise, it would mean that some fundamental points remain ambiguous, thus generating problems that might negatively affect completeness, performance or deadlines completion.

1.2. Scope

To describe the requirement engineering which serves as input for the design process, a brief explanation of the problem conceptual modeling is presented in this section. As the application to be developed will be situated in the context of the stores' supply and customers' demand, it is useful to model an abstraction of the set of occurrences on this physical environment in which the software is going to act, defining this as the **world phenomena.** On the world phenomena, there are two relevant classes: the software goals (described in the previous section) and the **domain assumptions**, defined as the set of phenomena from the physical world that are reasonably taken for granted in order to establish the connection with reality. On the other hand, the software and hardware system, denominated as the **machine**, will be influencing a certain part of the world with the purpose of bringing the requirements into completion.

The software engineering developed in this project will elaborate a set of requirements that must be fulfilled through phenomena that will happen both in the world and in the machine at the same time, which are denominated as **shared phenomena**. These phenomena can be controlled either by the world or the machine. In Figure 1, an abstract representation of the different types of phenomena is illustrated. The definition of the software requirements are designed in such a way that, jointly with the domain assumptions, will satisfy the software goals specified in the previous section. This means that

the domain assumptions are the conjectures obtained from the world phenomena that mutually complement the requirements fulfilled by the machine within the shared region.

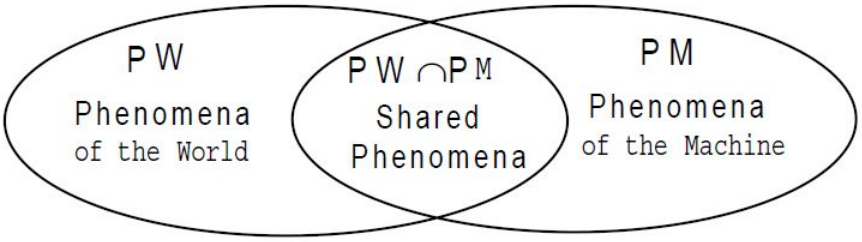

Figure 1: Boolean representation of the World and Machine Phenomena Model.

(taken from M. Jackson, 1996)

1.3. Definitions, Acronyms, Abbreviations

1. RASD: Requirements and Specifications Document
2. ASAP: As Soon As Possible
3. DBMS: Database Management System
4. OSM: Open Street Maps is a collaborative project to create a free editable map of the world.
5. QR Code: Quick response code, consisting of an image of a 2D matrix that can be read and interpreted by an application as a source of information
6. TIFF: Tag Image File Format, is a computer file format for storing raster graphics images

1.4. Revision history

Version 2 considers the following modifications with respect to the previous one:

1. Section 2.2.4 on Queue Manager is added in order to give specifications about the inner functionality of the Queue Manager software feature.
2. Section 2.2.5 on Data Model is added in order to specify the expected data structures that the software must use to bring the functionalities into completion.
3. Section 4.2 on Requirement Traceability is modified by including new requirement M8.

1.5. Overview

This document elaborates on the software design proposed for this project by using the previous RASD document as a core reference for establishing algorithmic and conceptual characteristics to be

developed. The information is presented in different layers in order to be read by different actors. In Section 2, there is an understanding of the architecture to be used in order to communicate the high-level features that the software is aimed to have as well as a set of guidelines and specifications for software development in such a way that it can be built in a coherent manner as long as the production goes in line with the architecture. In Section 3, there are guidelines of how the software has to look like from the user perspective, by giving specific details about the features that the interface should have in order to fulfill the expected functionalities. In Section 4, there is a connection between the defined components of Section 2 and the Requirements specified in the RASD in such a way that the accomplishment of these requirements by means of the components and their interactions can be understood. In Section 5, there is a planification, technical specification and guidelines of the implementation, integration and testing processes with their respective justifications from a technical point of view by considering time and resources optimization.

2. ARCHITECTURAL DESIGN

2.1. Overview

The architecture describes the core components of a software system and how they interact with each other. This concept is useful to have a general understanding of how the software will work, in such a way that the high-level characteristics can be clearly communicated among the different stakeholders. By taking into account this architecture, it will allow the development team to take the most relevant decisions in the earliest design stages. As long as the design goes in line with the architecture, it will move forward in an organized and coherent manner.

For this particular application, a layered architecture is needed since there is a hierarchy that has to be distinguished: the access logic needs to run on a different hardware than the DBMS, where supermarkets, queues, tickets, and users data will be stored. For this reason, a structured organization of the components is considered by separating them based on whether they run in the client's mobile device or the host server, thus, the so-called client-server architecture is employed. Components of both subsystems are in charge of different logical tasks of the software, but only server components (back-end) are able to access the DBMS in such a way that they are used as APIs by the client components (front-end) in order to get useful data for carrying out the functionalities. It is important to mention that the user interface will be implemented at the client layer and executed by the user's mobile browser capabilities, but most of the logic will be executed on the server side, which will be interacting with the client side through HTTP requests and responses.

Analyzing in a deeper level the proposed client-server architecture, it can be defined as a two-tier architecture employing a Distributed Logic, which means that there is application logic on both layers client and server. Thus, the following components can be defined in this architecture:

1. GUI - Graphical User Interface provided by the web application and developed for the user access to the software system.
2. Application 1 - It is the part of the software that runs on the user's mobile device when the web application is being accessed. It is characterized by providing the map for checking and selecting the different supermarkets (as illustrated in Section 3.2) and it also contributes with the tools needed by the user to request tickets.
3. Network - the communication channel which connects the requesting application provided in the user's interface and the application in charge of the queue management.
4. Application 2 - the part of the system in charge of managing the supermarkets' queues and suggestions. It provides the software implementation to manage most of the software logic.
5. Data - where the supermarkets, queues, tickets, and users data will be stored to be accessed by the back-end components.

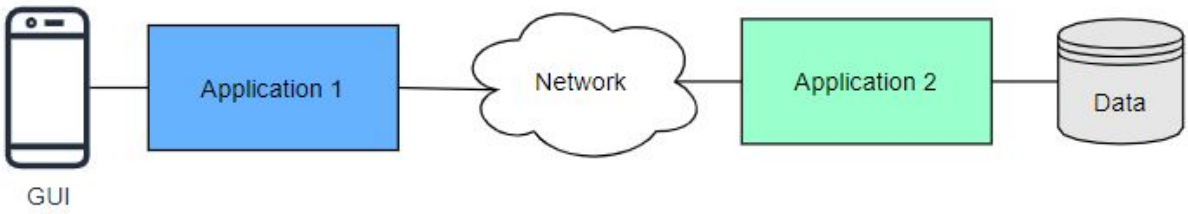

Figure 2: Architecture highest level

2.2. Component view

The components can be splitted into the two architecture subsystems that constitute the Client/Server architecture. When components of a subsystem communicate with components of the other subsystem, HTTP connection is employed. As mentioned before, the client components run on the user's mobile device browser while the server runs on the system host. In this section each of the components will be described according to their purpose as part of the system.

2.2.1 Client components

- *Authenticate*: Provides user credentials and session information to the different components across the system in order to personalize the content for each single user.

- *SupermarketData*: Represents the content data regarding supermarkets' availability, positions and suggestions that will be provided to the client under different formats: map (Section 3.2), ASAP availability list (Section 3.3) and booking time slots (Section 3.5)
- *QRCode*: Allows the client to visualize and manage the QR code tickets after making a Booking or ASAP Line-up request.
- VueGeoLocationAPI: Provides the user geographical location. This location will allow the system to provide supermarkets map displays and suggestions.

2.2.2 Server components

- *AuthenticationService*: Verifies the credentials of a user by allowing the application to provide an user session for accessing software content..
- *QRCodeGenerator*: Generates the QRCode upon user's requests of a booking or line-up reservation.
- *QueueManager*: Manages the queues of all users in all supermarkets. Further details about inner functionality on Section 2.2.4.
- *GetSupermarkets*: Provides the supermarkets' information regarding availabilities and suggestions to the user interface
- *DBMS*: Database Management Service that stores supermarkets, queues, tickets, and users data. Further details about entity relationships in the database on Section 2.2.5.

2.2.3 External Components

- *OSMService*: external service that provides the OSM basemap.

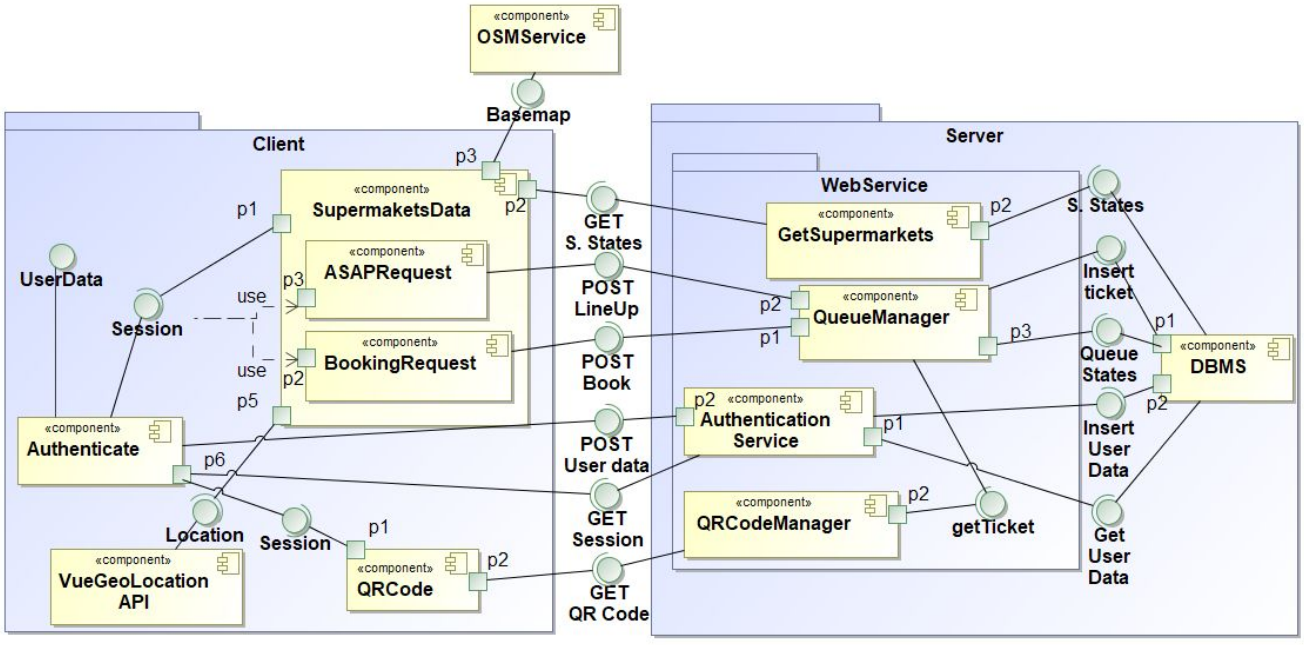

Figure 3: Component diagram

2.2.4 Queue Manager

It is expected that many users request to line-up or book to many different sections combinations. The authorization to enter a supermarket will depend on sections' selection and sections' availability in such a way that the system will give authorization to enter if and only if all selected sections have room for allowing customers to buy in. For this reason, the queuing is not First Come First Served, but an heuristic that lets users enter as soon as their requested sections have room. As a second heuristic (in case the first one presents a tie), the order of customers' request arrival is considered.

In order to represent the states of the queues in the software, the system should dynamically keep the track of the people entering and leaving the supermarkets and their sections. The events that update sections' state is when customers request or cancel a ticket and enter or exit the store, but only when someone gets out the store, the system can trigger the authorization for other users to enter, so each time this happens, the clients' virtual queue should be checked and estimation time should be updated.

In order to highlight the inner functionality of the software's queue manager, Figure 4 depicts an illustrative instance in which there are 3 supermarket's sections and 3 waiting customers. The one that firstly requested to line-up is interested in buying meat and fruits, the second one wants to buy bread and fruits and the third one only fruits. The meat and bread sections are full and the fruit section is not full. In this situation, the customer that arrived in the last place is allowed to enter immediately since its section is free. Once a shopping customer leaves the bread section, the second waiting customer can enter, meanwhile, the first one still has to wait for its turn.

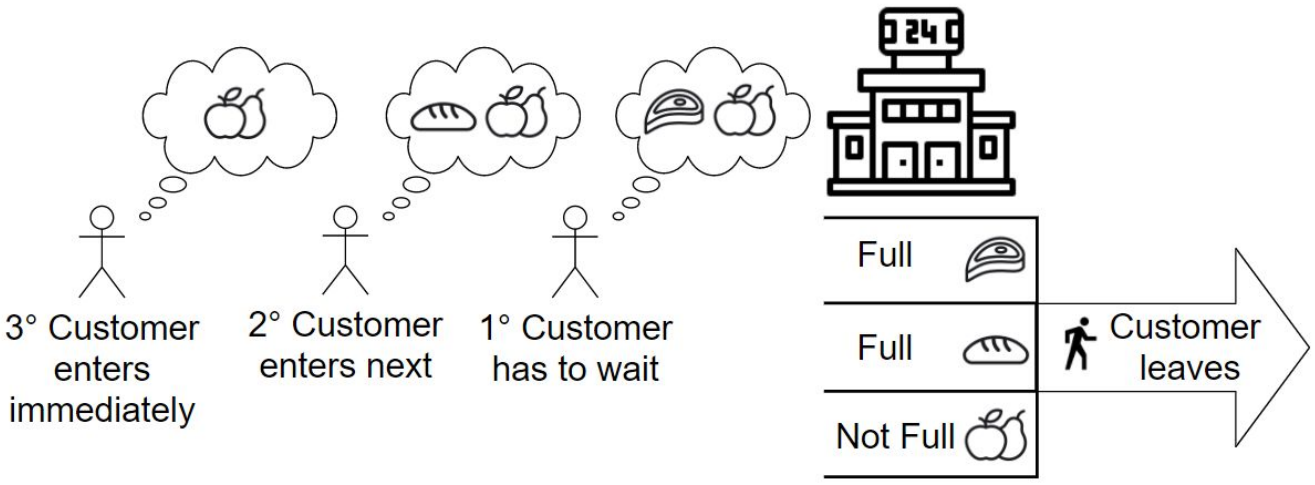

Figure 4: Queue Management Sketch

In order to employ the previously explained heuristics, the data that the Queue Manager will use as input will be two tables specifying the customers that are shopping in each supermarkets' sections and the ones that are waiting for their turn. Further details about the data structure of the Queue Manager input will be given in the next section.

2.2.5 Data Model

The main source of data that the different software components will use is the DBMS, which will be the place where different useful entities will be somehow connected between each other. The proposed DBMS design considers different entities whose relationships are modelled in Figure 5 and described in the following list::

- User: represents the different registered users. It includes the username (has to be unique), password (hashed for security reasons), complete name and email (has to be unique).
- Request: represents each user request, by specifying whether it is a line-up or booking request (type_id), the datetime, an unique request ID and the supermarket that the request is directed to. As specified in the RASD, each user can only have one request, and each request has one and only one user.
- Supermarket: represents the supermarket by specifying its unique ID, name, logo (for displaying in the map), geographic coordinates, address and maximum number of customers that are allowed to shop there at the same time[1].
- Section: specifies the different sections that the supermarkets can have, by establishing its unique ID, the section name and the maximum number of customers that are allowed to shop there at the same time.
- Waiting: each entity represents a user that is **waiting to shop** at a certain supermarket's section after doing a certain request. It specifies the section ID, username, request ID and supermarket ID. It is important to note that if a user is waiting for shopping at a certain number of supermarket's sections, this table will have as many entities associated with that user as sections it has requested, hence, there would exist a redundancy in the request ID, username and supermarket ID.
- Shopping: similarly to the previous type of entity, each one represents a user that is **shopping** at a certain supermarket's section after doing a certain request. It specifies the section ID, username, request ID, supermarket ID and datetime in which the user entered the store. This table has the same redundancies as the previous one, since, as specified in the RASD, once a user enters a supermarket, it is considered as it is shopping at all requested sections at the same time, and once it leaves, all of them are set free.

[1] in case there is a legal constraint, but in principle the software functionalities are concerned only on sections' capacity

As mentioned in the previous section, the Waiting and Shopping tables are updated each time a user requests to line-up or book and enters or leaves a supermarket. These data structures are the inputs that the Queue Manager will use to compute the users' turns or time estimations each time they're updated.

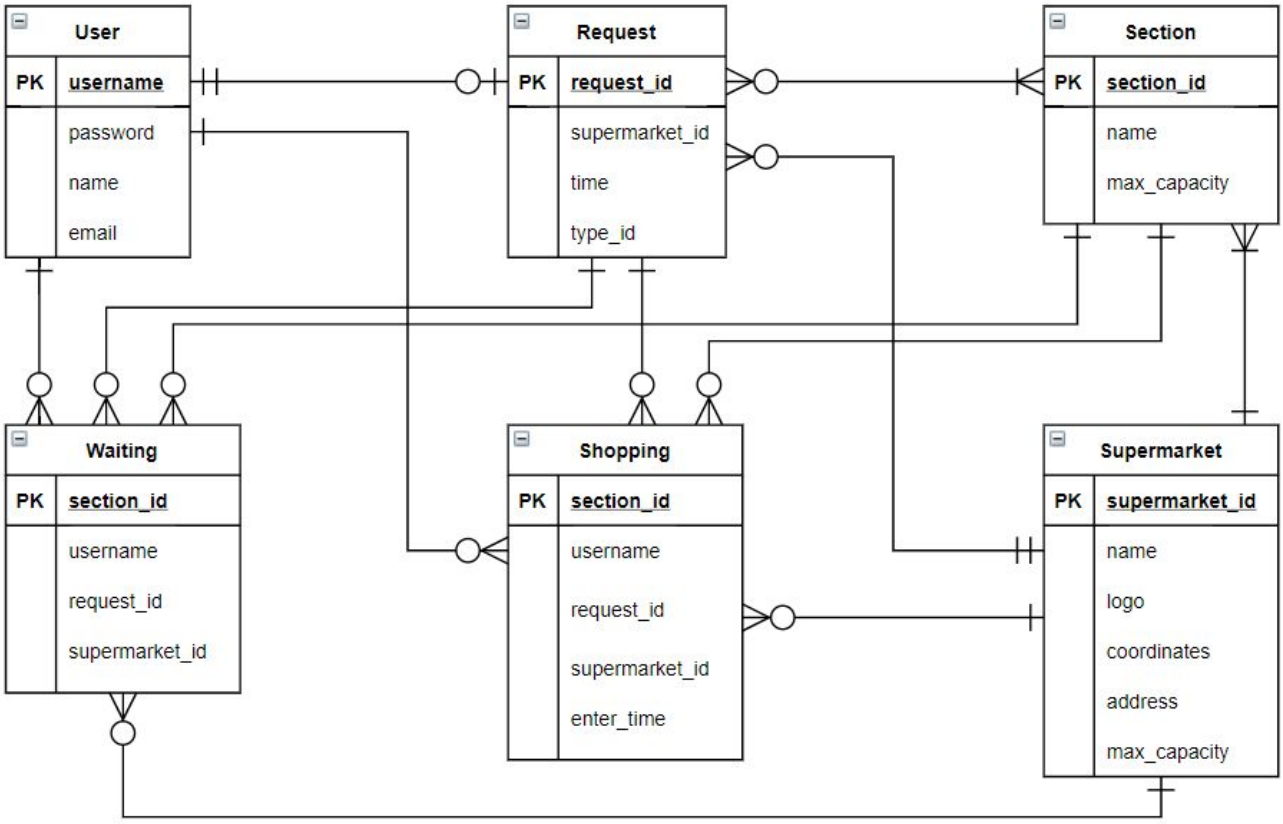

Figure 5: Entity-Relationship Model of the database

2.3. Component interfaces

The system components have to communicate with each other in order to interchange data and accomplish the expected software functionalities. Different components interfaces are defined in this section. To completely describe component interfaces, the data type, departure and arrival components are specified in the following table:

Interface	Description	Data type	Departure	Arrival
User Data	Username and password from log-in form.	JSON	User	Authenticate

Session	Logged-in user's Session data	JSON	Authenticate	SupermarketsData QRCode
Basemap	Background OSM map	TIFF	OSMService	SupermarketsData
Location	It contains the geographical GPS coordinates of the user	Tuple of floats	VueGeoLocation API	SupermarketsData
GET Supermarket States	HTTP GET method. It transfers a list of suggestions in form of functional data structure that contains supermarkets' names, distances and availabilities based on user preferences	JSON	GetSupermarkets	SupermarketsData
POST Line-up	HTTP POST method. It contains the line-up request made by the user	JSON	ASAPRequest	QueueManager
POST Book	HTTP POST method. It contains the book request made by the user	JSON	BookingRequest	QueueManager
POST User Data	HTTP POST method. It contains the user credential to be checked in order to log-in or register.	JSON	Authenticate	Authentication Service
GET Session	HTTP GET method. It transfers the Session information generated at the server to be used by the client.	JSON	Authentication Service	Authenticate
GET QR Code	HTTP GET method. It transfers the QR code data generated at the server to be used as a physical ticket by the user	PNG	QRCodeGenerator	QRCode
getTicket	Line-up or Booking information generated at the QueueManager to be used by QRCodeGenerator	JSON	QueueManager	QRCodeGenerator

Supermarket States	SQL query. Supermarkets' static information to be used to compute suggestions or visualizations by getSupermarkets	JSON	DBMS	GetSupermarkets
Insert Ticket	SQL Insert. It updates the tickets table by inserting a successfully generated ticket.	JSON	QueueManager	DBMS
Queue States	SQL query. Supermarkets' Sections availabilities to be used to compute suggestions or visualizations	JSON	DBMS	QueueManager
Insert User Data	SQL Insert. It inserts into the user data table, the information of a new user after successful registration.	JSON	Authentication Service	DBMS
Get User Data	SQL query. It retrieves a user for the authentication process.	JSON	DBMS	Authentication Service

2.4. Deployment view

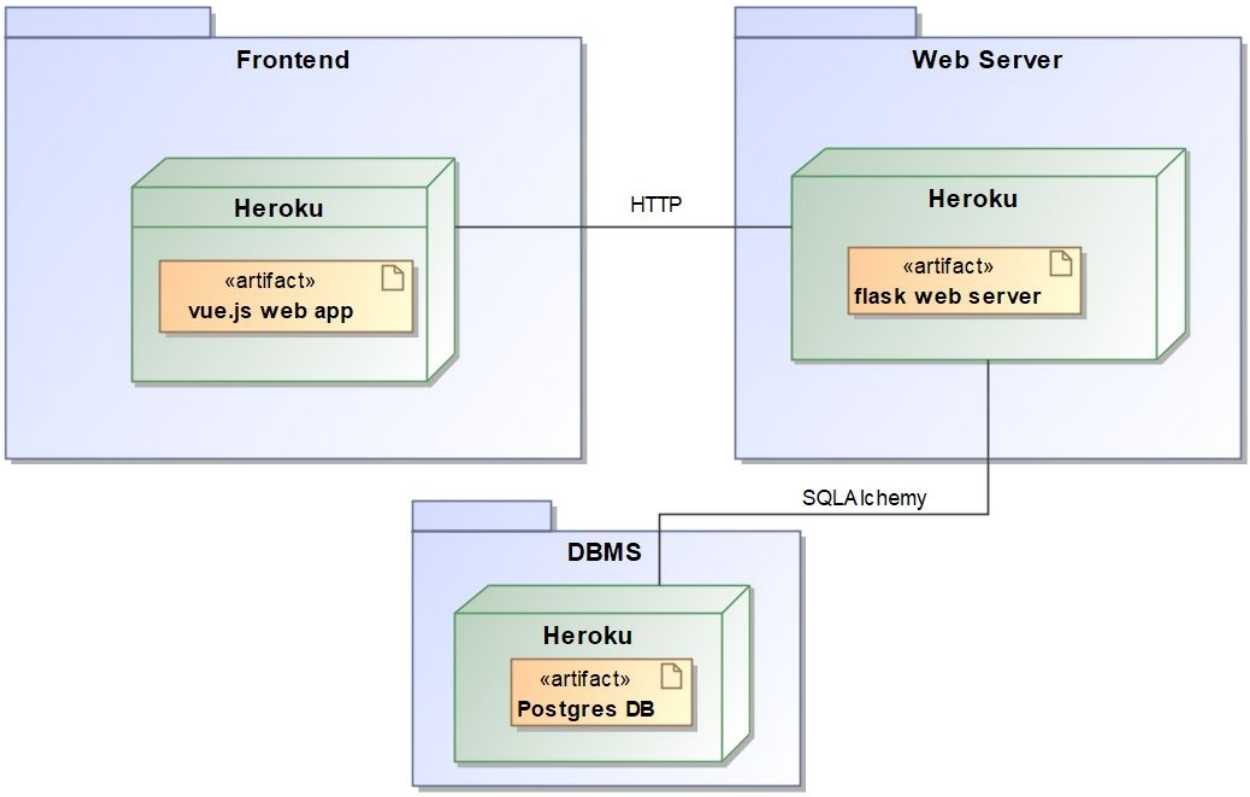

Figure 6: Deployment diagram

In the Deployment Diagram shown in Figure 6 we can see the principal components of the software deployment:

1. Vue.js web application: is an open-source model–view–viewmodel frontend JavaScript framework for building user interfaces that will be implemented using Nuxt, an open source web application framework based on Vue.js. In this web application will be implemented all the visual components like the login and register pages. The user location will be retrieved using the VueGeoLocation API which will be used to center the map and show the supermarkets located nearby. The map will be displayed using Leaflet and the map layer will be OSM.This will be later deployed to an individual Heroku app to be accessed from any device.
2. Flask Web Server: Flask is a micro web framework written in Python that will be used to implement the logic of the application i.e., login authentication, user registration, booking requests and any other required functionality. This Web Server will communicate with the web application via HTTP request, e.g., the user requests to login and sends his data using a POST request and the web server will authenticate him using Json Web Token (JWT) and give as a response to the web application a session.

3. Postgres Database: it is a relational database management system (DBMS) emphasizing extensibility and SQL compliance. It will contain all the tables where the data will be inserted and queried. The communication with the web server will be by migrating the database using SQLAlchemy, meaning also that the tables' columns will be defined using models in Python.

All these principal components will be deployed in individual applications in Heroku to be accessible from any device.

2.5. Runtime view

The interaction between components is carried out in such a way that front-end components communicate with back-end components contained in the Web Service, which are used as API while accessing the DBMS. The runtime behavior of a representative case that depicts both the Lining-up and Booking internal process is summarized in the following steps:

1. There is an authentication process using Authentication Service as API and accessing the users data table for checking user input validity.
2. Once session is obtained, OSM Service is called to be included in the client map and supermarkets' states (availability) and information are collected by using the Queue Manager and GetSupermarkets function as APIs for accessing the different supermarkets' useful dynamic and static data stored in the DBMS.
3. User's location is sent to the server logic in order to compute suggestions based on both location and availability. This information is sent back to the SupermarketsData component, which displays this information in the client as a part of the user interface.
4. Then, we suppose that the user confirms the lining-up or booking on a certain supermarket, which triggers an insertion of the Line-up or Book, managed by the Queue Manager server API, in the DBMS.
5. Once the DBMS is updated with the new Line-up or Book, the QR code is generated in the server logic and sent to the QR code user interface, which can be accessed through the user session.

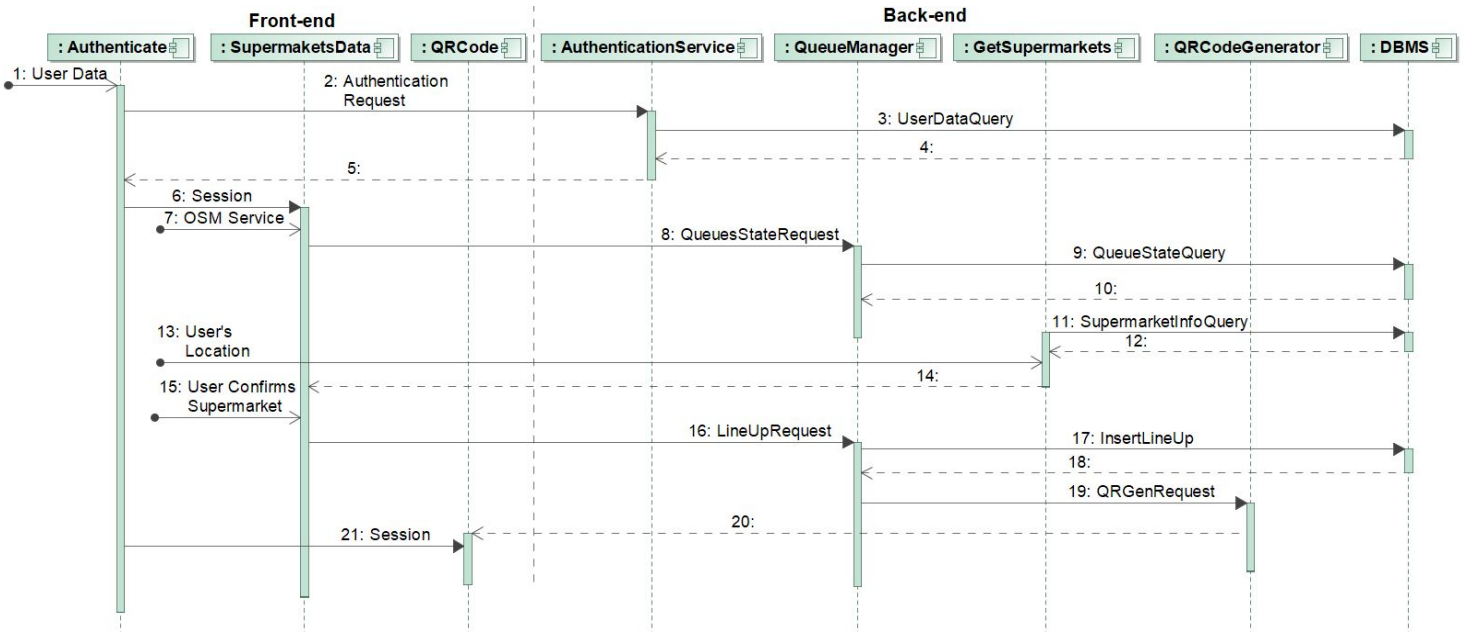

Figure 7: Component sequence diagram

2.6. Selected architectural styles and patterns

2.6.1. MVVM — Model View ViewModel

Also known as the Model-View-Binder, it is a programming technique that binds data sources from the provider and consumer together and synchronizes them. It gets rid of lots of boilerplate code that we traditionally need to keep View and Model in sync. This allows us to work on a higher level of abstraction. We can describe two important concepts of this model:

1. ViewModel is an object that exposes bind-able properties and methods which are consumed by the View.
2. MVVM has an additional Binder entity that is responsible for keeping View in sync with the ViewModel. Every time a property on ViewModel changes, View is automatically updated to reflect the changes on UI.

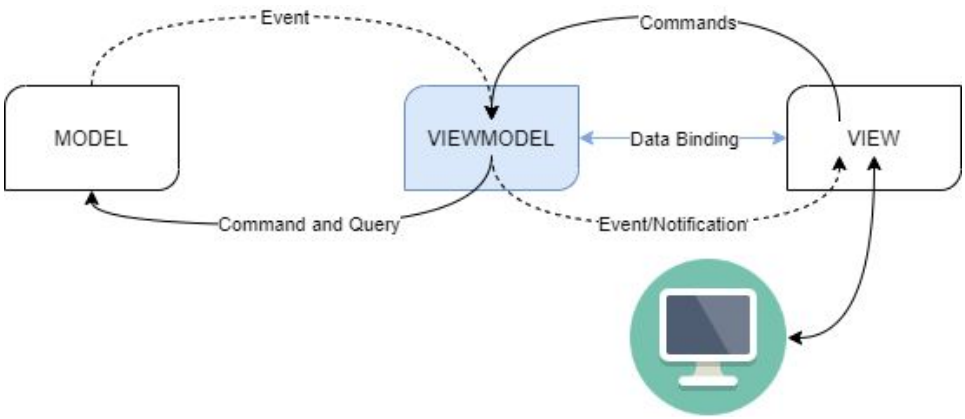

Figure 8: MVVM - Model View ViewModel

The chosen pattern is MVVM because we will use Vue.js to build the interactive web interface and it is focused on the ViewModel layer of the MVVM pattern. It connects the View and the Model via two way data bindings. Actual DOM manipulations and output formatting are abstracted away into Directives and Filters.

2.6.2. Client-Server Pattern

Client server pattern is a network architecture that consists of a server and multiple clients, provider of a service and requesters. In this project the client side is represented as a website designed for mobiles (mobile first web application) that can be accessed from any device, this component contains all the views, e.g., login, map, supermarkets lists, that will interact and receive information from the user. The application server will contain the application logic and a database to store data if necessary.The client, also called frontend, will send Http requests to the application server through an Application Processing Interface (API) that has all the endpoints of the application server and the formats in which the communication must be done.

A key advantage of this pattern is that the core functionalities are separated from the visual part of the whole project. This allows a higher level of abstraction and the client and server can be easily changed only considering the communication endpoints. A disadvantage can be the usage of a single server and having many requests at the same time or a system failure, in this case we may consider to have multiple servers and a load balancer always depending on the number of people expected to use the application.

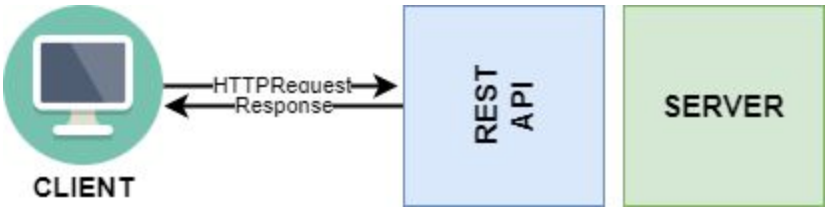

Figure 9: Client-Server model

3. USER INTERFACE DESIGN

The user-system interaction will be driven by a map-based user interface design through which the users will find a graphical interface suitable for their experience with the software. The philosophy behind the user interface design is minimizing the number of different application pages in such a way that each one of the requirements can be addressed in a user-friendly way.

3.1. Log-in interface

The first view of the application will be a splash screen with the logo and the option to log in or sign up only in case that the user is not already logged in to the application. After that, depending on the option chosen, one of these views is shown.

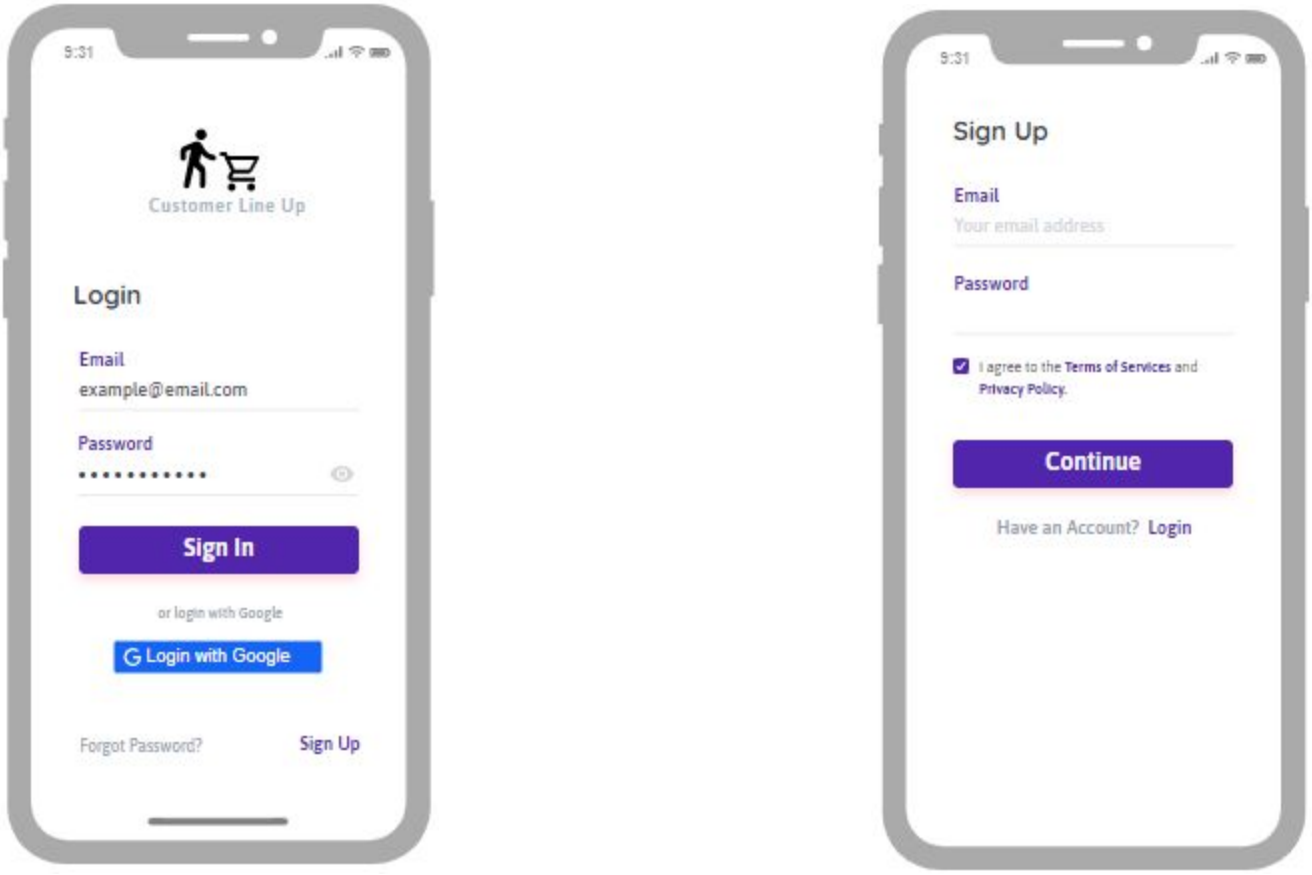

Figure 10: Login and sign up user interfaces

When the users enter the application, in the first instance they will find a typical and well-known log-in page with sign-in and forgot-my-password options.

3.2 Map interface

When the users successfully log in, they enter the core of the interface which is based on a map service, where different supermarket locations are explorable by means of map snippets such as drag or zoom to personalize the display. Jointly with the map interface, a section preferences tool will be available in order to make the user able to discern between supermarkets estimated waiting times by means of semaphore symbology (black color means closed store) as shown in Figure 11. The supermarkets' waiting time color will be updated when the users select a certain preference regarding the section in which they want to buy. Furthermore, a special button for lining-up to a supermarket ASAP is available.

Figure 11: Map and filter user interfaces

The map interface will also contain a filter button which will display a pop-up to allow the users to select the desired supermarket(s) and the sections they want to visit.

3.3 ASAP supermarket list interface

When the users click the ASAP lining-up button on the map interface, it redirects to a list of the most convenient options in terms of estimated waiting time and location as shown in Figure 12. Clicking on any option will redirect to the line-up options of the clicked supermarket where users will be able to confirm the lining-up.

Figure 12: List of supermarkets suggestions for ASAP line-up user interface

3.4 Line-up options interface

Each supermarket will have a line-up options interface containing the main information about the supermarket: location, distance, address, and estimated waiting time with the respective color symbology. The objective of this interface is to make the user able to select between lining-up ASAP to shop in a supermarket or booking to shop at another time. If the users select lining-up ASAP, they will receive a QR code from the application. If the users select the booking button, it redirects to the booking interface. Additionally, this interface will have a section preferences tool through which the user will be able to update the estimated waiting time according to the section selection.

Figure 13: Supermarket pop-up user interface

3.5 Booking interface

When the user clicks the booking button on the lining-up options of a certain supermarket, it redirects to an interface in which the users will be able to select a day and time to book. When they select a day, the system will suggest a convenient time to book based on availability and the section selection that was done by the requesting user, which may also be updated in this interface. Once the day and time were selected, the booking can be confirmed and the QR code will be provided by the application.

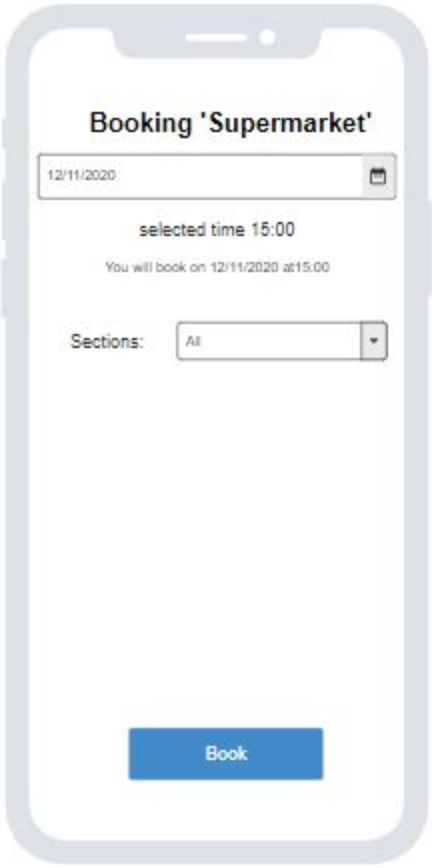

Figure 14: Booking in a specific supermarket user interface

3.6 QR code interface

The QR code will be displayed when booking ASAP or for a specific date and time in the application, and the user can scan it in order to be able to enter the store. In fact, the users can access their line-up or book ticket at any moment because in the main view there is a QR code button where all the available QR can be found.

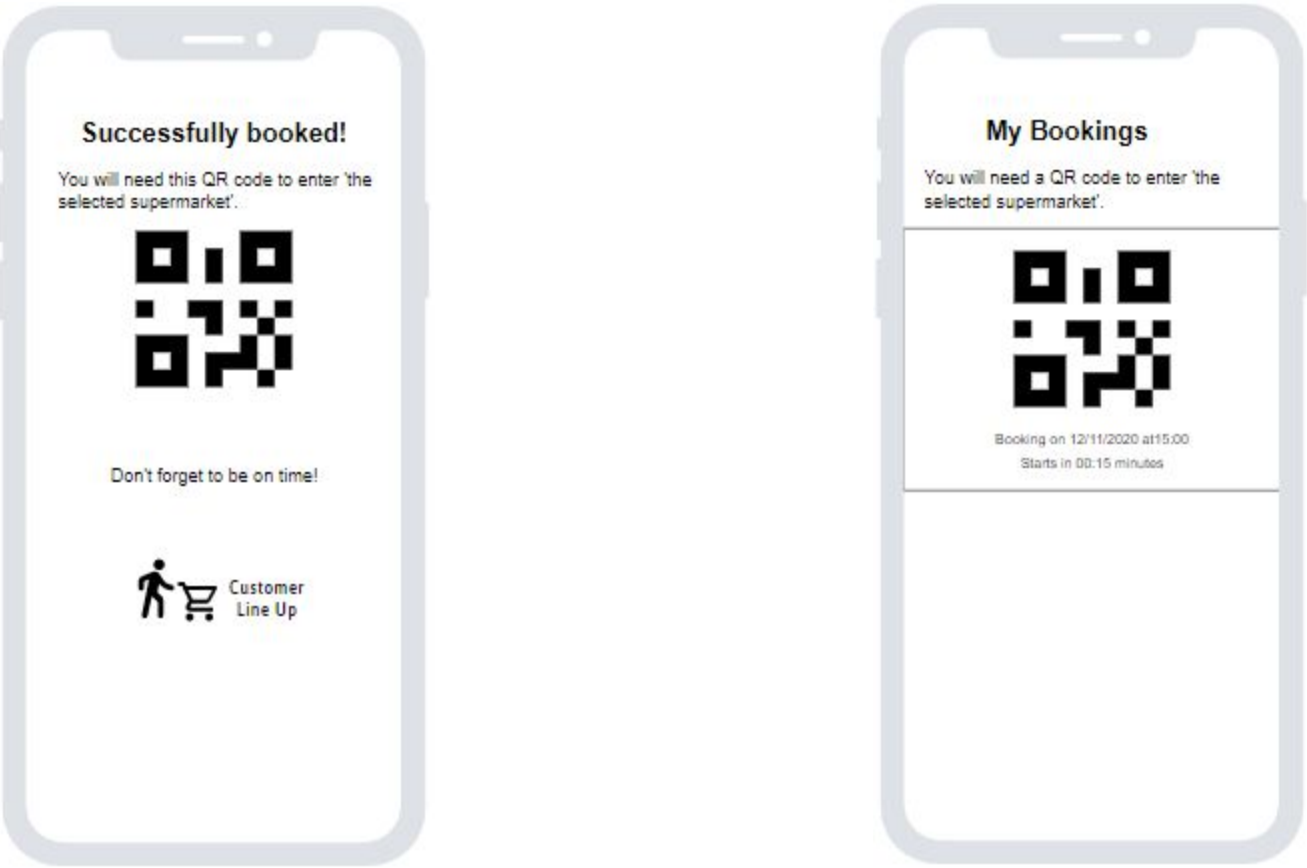

Figure 15: QR code user interfaces

4. REQUIREMENTS TRACEABILITY

Given the defined software components, it is important to understand how the components play a role in fulfilling the requirements that were defined in the RASD document. In this section, a mapping of the requirements into components will be elaborated in order to comprehend the importance of the components that partially give their contribution on specific requirements that, as analyzed in the RASD, will jointly bring the goals into completion.

The requirements are summarized into specific functionalities that are controlled by the world or the machine, and represent the effect of the software system in the shared zone (defined in Section 1.2). This subset of phenomena is constituted by the operations carried out by the software system on the real world by means of functionalities that concretize the purposes. The following lists specify each one of these requirements, and which ones of the defined components play a role in satisfying them.

4.1 Requirements controlled by the world

Requirement W1: The application will allow users to request a number for entering a store. The request may be for entering ASAP (Line-up) or on another time slot (Book). The line-up request can be executed from the user's mobile device as well as from a ticket machine outside the store. The book request can only be executed from the user's mobile device.

Components that fulfil W1: The main actor that fulfils this requirement is the SupermaketsData component, which provides the interface tools for the clients to make them able to line-up or book. The GetSupermarkets server component provides the information regarding availability and distance and the QueueManager server component concretizes the user request by accessing the DBMS.

Requirement W2: The users will specify the categories of groceries that they want to shop. In case a user does not know clearly this information, all categories will be assumed.

Components that fulfil W2: The SupermarketsData client interface allows the user to select the preferred sections to buy, which will trigger an event that asks the GetSupermarkets server component to provide the information regarding availability, based on the section selection.

Requirement W3: In case the system notices the users that it is their turn to enter the store, they can scan the provided QR code. In case they do so, they immediately enter the store, otherwise they lose the number after a certain time tolerance.

Components that fulfil W3: The client component that will alert the user in case its turn is soon, is the SupermarketsData, which will receive the supermarkets' state from the GetSupermarkets component in order to trigger an event in case the user has to be advised by the system to enter the store. The ticket to enter the store is provided by the QRCode client component that receives it from the QRCodeGenerator server component.

Requirement W4: The system will allow users to register and log in, so a certain state will be associated with the mobile application users. The application can only be used by logged in users.

Components that fulfil W4: The client component that allows the users to log-in or register is the Authentication, which uses AuthenticationService server component as API for accessing the DBMS and validate or reject the request.

4.2 Requirements controlled by the machine

Requirement M1: The system will control the amount of people inside the store by means of a queuing manager system, which will provide permission to enter if and only if all the sections selected by the user have capacity. Only when it is users' turn, the system will accept the QR code as a ticket and will make the user able to enter the store. This management does not consider the entering of preferential customers, since the stores will always count with an extra capacity for these cases.

Components that fulfil M1: The QueueManager client component will handle the turns of the users based on the section preferences´and supermarkets' availability. All the queues' content will be stored in the DBMS, which will provide information to the GetSupermarkets server component in order to notify the users when it is their turn through the SupermarketsData client component.

Requirement M2: The system will receive the geographical position and preferences from the customers, and it will return the state of the stores regarding the amount of people currently buying and the distance with respect to the customer. Based on this information, the system will always find a suggestion hierarchy regarding the convenience of going to the different stores. The system suggestions will contain stores with sections in which the customers want to buy and these sections must be not at full capacity.

Components that fulfil M2: The client component in charge of providing user position is VueGeoLocationAPI. The GetSupermarkets server component takes the supermarkets' state and the user location to provide suggestions by means of the SupermarketsData client component.

Requirement M3: In case customers want to shop ASAP, they will receive suggestions of alternative stores before confirming the lining-up. The suggestions are made on the basis of store availability and position.

Components that fulfil M3: The SupermarketsData client component will provide to the user a list of convenient supermarkets regarding the user section preferences and location. The GetSupermarkets server component provides to the client the supermarkets' availability information by accessing the DBMS. The client component in charge of providing user position to the client is VueGeoLocationAPI.

Requirement M4: In case customers want to book a number on a certain store and time slot, they will receive time slot suggestions based on availability before confirming the booking.

Components that fulfil M4: The SupermarketsData client component will provide to the user the option to book on each supermarket. After selecting the option to book, time slot suggestions will be provided by the client component. The GetSupermarkets server component provides to the client the supermarkets' availability information that allows the client to provide the time slot suggestion by accessing the DBMS.

Requirement M5: When the users confirm a certain lining-up or booking, the system will send them the corresponding QR code to be used as a ticket when it is their turn.

Components that fulfil M5: The ticket to enter the store is provided by the QRCode client component that receives it from the QRCodeGenerator server component.

Requirement M6: The system will provide one ticket at a time to each user. Another ticket can be requested if and only if the current ticket is once dropped or used.

Components that fulfil M6: The QueueManager server component is in charge of providing one ticket at a time to each user. This functionality is totally controlled by the server.

Requirement M7: Customers will be notified by the system in case their turn to enter the store is soon. Additionally, they must be warned about their distance to the store in case they are too far from the store.

Components that fulfill M7: The client component that will alert the user in case its turn is soon, is the SupermarketsData, which will receive the supermarkets' state from the GetSupermarkets server component in order to trigger an event in case the user has to be advised by the system to enter the store. Additionally, the GetSupermarkets server will receive user position from the UserLocation client component in order to notify the users in case they're too far.

Requirement M8: When customers request a ticket, they should continuously receive from the system an estimated time to have their turn to enter the store. Time estimations should be computed based on previous shopping times that each section of each supermarket has shown in the past.

Components that fulfil M8: The DBMS is expected to keep track of the timestamps of customers entering and leaving the supermarkets as well as the selected sections. Based on this, the GetSupermarkets server component is able to compute the average time that the customers take in shopping on each section of each supermarket. Based on the average times and the user section selection, the time estimation can be computed to send it to the SupermarketsData client component that provides it to the user.

5. IMPLEMENTATION, INTEGRATION, AND TEST PLAN

5.1 Overview

The strategy for software implementation provides a road map that describes the steps to follow during the implementation of software and the integration of different components. Testing will be present in all the phases, from unit testing of individual components, to components integration and system validation to guarantee the fulfillment of the requirements.

5.2 Implementation Plan

The two main parts of the application will be considered to define the implementation plan, the client and the server, which will encompass the small components defined in the Component diagram in Figure 3. The approach chosen is the Bottom Up approach in which the lower level components or the ones with fundamental functionalities will be implemented first to facilitate the implementation of

higher level components. The same principle will apply for testing because each component will be tested individually by unit test and then used to facilitate the testing of higher level components.

The advantages of using this approach is that the bug localization is easier because the testing activities start as soon as the first component is ready, which makes the testing phase almost simultaneous to the implementation phase. Meanwhile, the only disadvantage will be that an early prototype will not be available because the highest level components are implemented at the end.

The technologies that will be used to implement the application are already defined:

1. Vue.js for the client
2. Flask for the server
3. Postgres for the database, in this case it is important to highlight that the connection between the database and the server will be done using SQLAlchemy, an open-source SQL toolkit and object-relational mapper for the Python programming language. This will allow us to define the database models, create the tables and query directly from Flask via migrations.

It is relevant to mention that some server components are directly related to the client components, between each couple of client-server components the one with the highest priority will be the server component. But immediately after the server component is implemented and tested, the client component can start to be implemented if it does not require the functionalities of other server components.

Having said that the order in which components should be implemented in the client and server is the following and will be explained in detail after the enumeration:

1. Server
- 1.1. DBMS
- 1.2. AuthenticationService
- 1.3. GetSupermarkets
- 1.4. QueueManager
- 1.5. QRCodeGenerator

2. Client
- 2.1. Authenticate
- 2.2. VueGeoLocationAPI
- 2.3. SupermarketsData
- 2.4. ASAPRequest
- 2.5. BookingRequest
- 2.6. QRCode

The first component that should be implemented is the DBMS because all other components implement functionalities, use or insert information in the database. Implementing the DBMS means creating an empty database in Postgres and a model in Flask defining all the classes that will contain the constructor and the structure of each table.

After the database, the next key component to implement will be the Authentication Service component that consists in the register and login. It is necessary because any person willing to use the application must be registered, meaning that all the other functionalities will work only if there is a session. This will be implemented using JWT and will leave the authentication endpoints to be consumed by the Authenticate component in the client side.

After that, the next component will be the GetSupermarkets server component which will return a list of the nearby supermarkets given a location. Considering this, the VueGeoLocationAPI client component can be implemented next.

Having the nearby supermarkets information and a user session it is possible to proceed with the main functionality of the application, the booking requests which the job of the QueueManager server component. Right after that the ASAPRequest and BookingRequest client components can be implemented.

When a booking request is successful, a QR code is given to the user to enter the supermarket. Then, the QRCodeGenerator server component is implemented and later the QRCode client component where users will be able to see the available QR codes.

5.3 Integration Plan

Following the bottom up approach defined earlier, each step of the integration will be specified. Before doing so we will introduce the concept of stubs and drivers that are dummy programs that act as a substitute for components that have not been implemented and simulate the data communication for integration and testing purposes. Stub is called by the component to be integrated and drivers call the component to be tested.

Following the order detailed in the implementation plan:

1. Initially we have only created the database and specified its structure. After that the implemented Authentication Service has to be integrated with the database, but in order to work this will need at least a username and password to login or register, being this a user request. This dummy data will be provided by the Driver meanwhile the authenticate client component is being implemented.

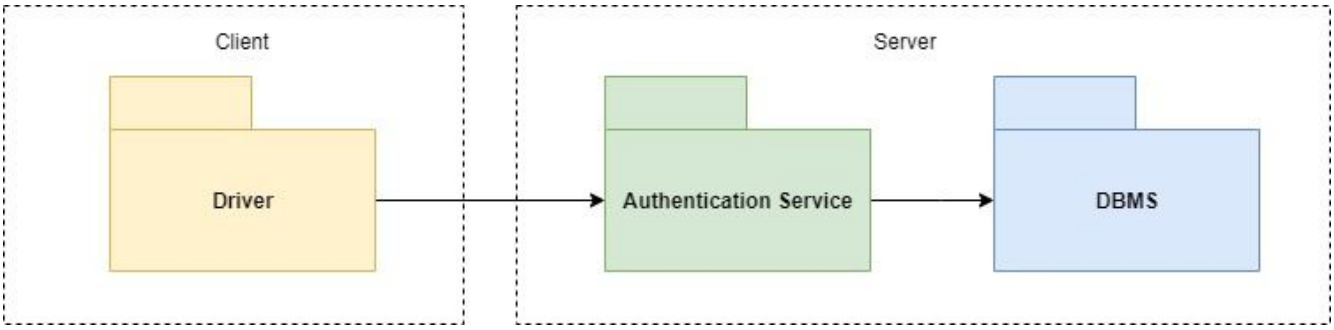

Figure 16: Authentication Service integration

2. When the Authenticate client component is ready, the Driver can be replaced with it and the corresponding integration test is performed.

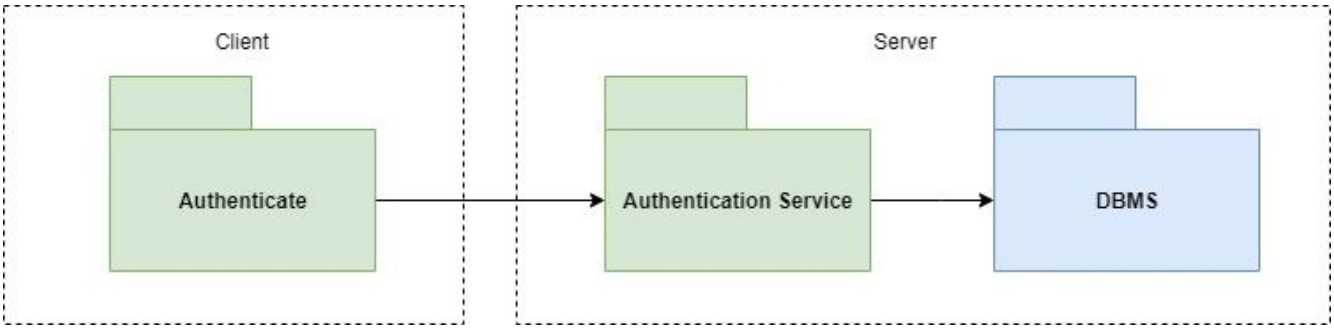

Figure 17: Authenticate integration

3. To retrieve the list of nearby supermarkets from a given location, the GetSupermarkets function is implemented on the server side, but it needs the SupermarketsData client component to request it given the user session and location and a Driver will be there to simulate it.

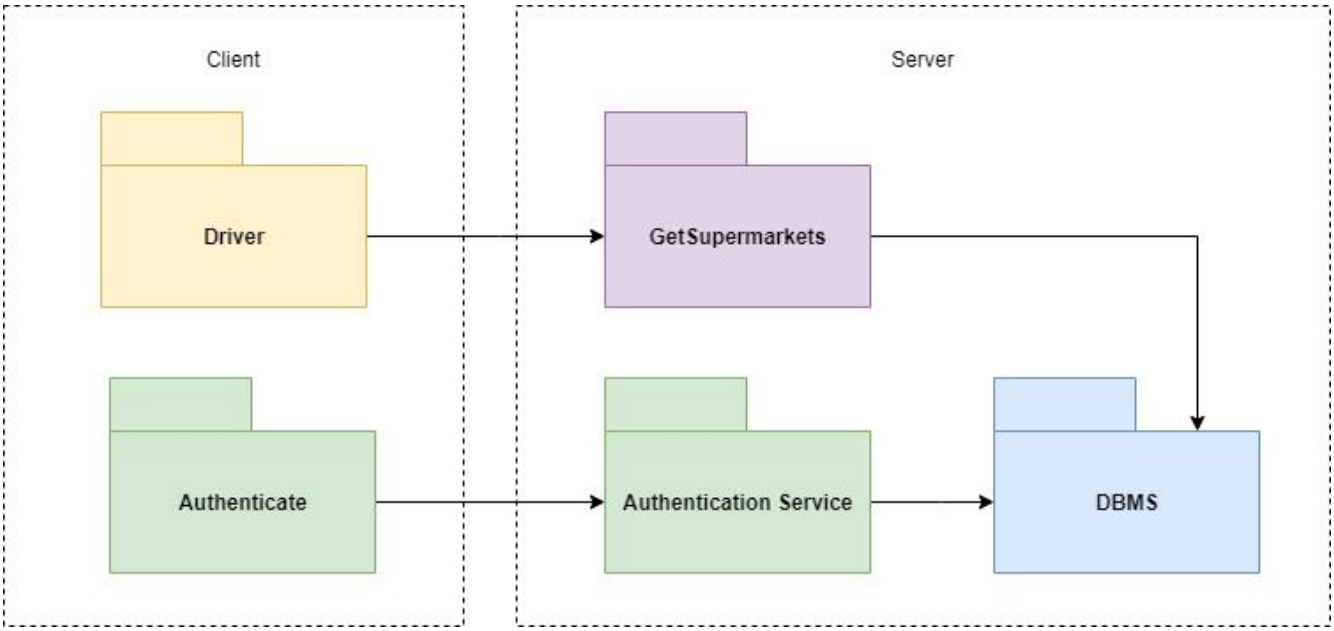

Figure 18: GetSupermarkets integration

4. The SupermarketsData is implemented to request the GetSupermarkets service, it uses the client session from authenticate and the VueGeoLocation API that is directly included and just tested for integration.

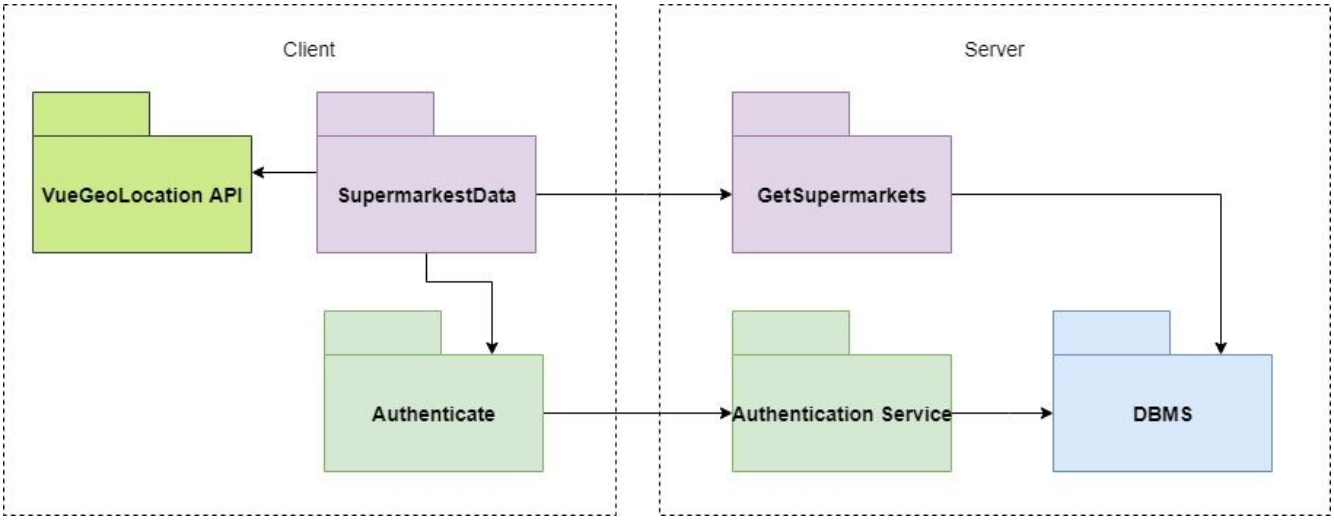

Figure 19: SupermarketsData integration

5. Then, having the nearby supermarkets list and the user information it is possible to implement the QueueManager server component. It will need a Driver to request a booking on the client side.

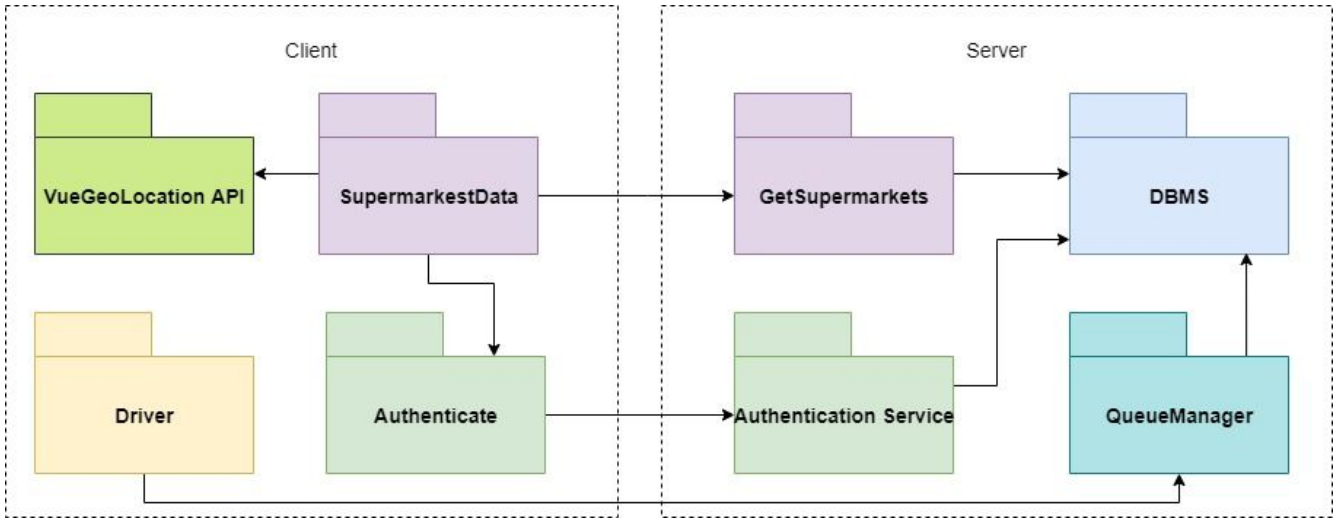

Figure 20: QueueManager integration

6. Once the QueueManager is integrated, we can include the ASAP and Booking Requests that will make use of all the available data: the supermarkets list according to the user location and the session after logging in. All this information is already handled by SupermarketsData, so the two types of booking requests will be subcomponents of it. Now the user is able to request a position in the queue and receive a response.

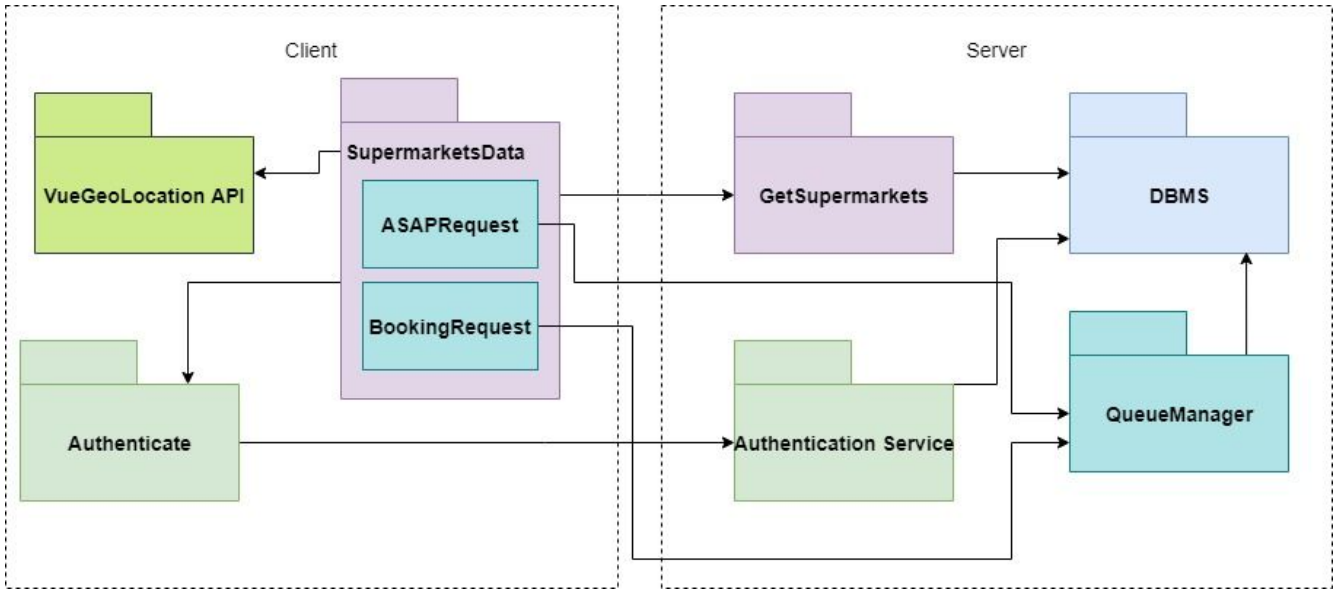

Figure 21: Booking requests integration

7. After the integration of all these components it is possible to deliver the most important product of the application to the user, the QR code. This will be called by the Queue manager to encode a key and return a QR code according to the position in a specific supermarket queue.

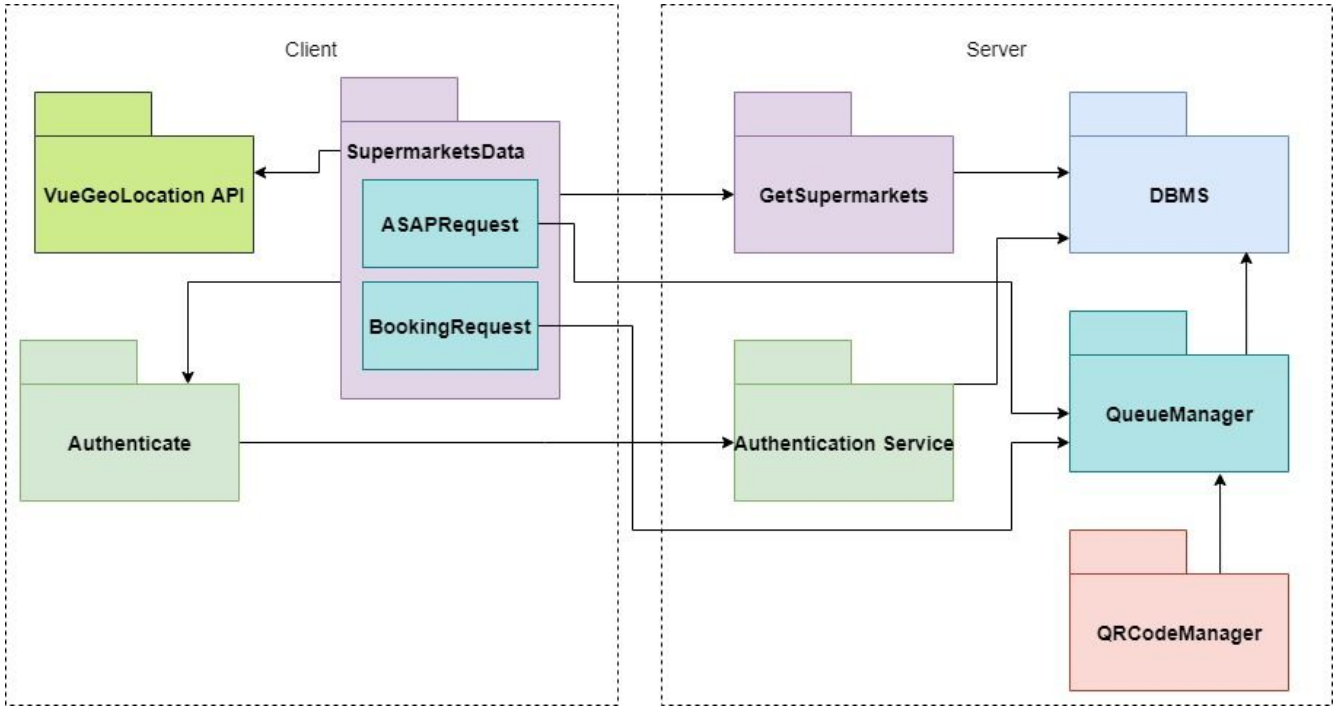

Figure 22: QRCodeManager integration

8. There will be a QRCode component on the client side to retrieve the user QRCode if available in case it is necessary.

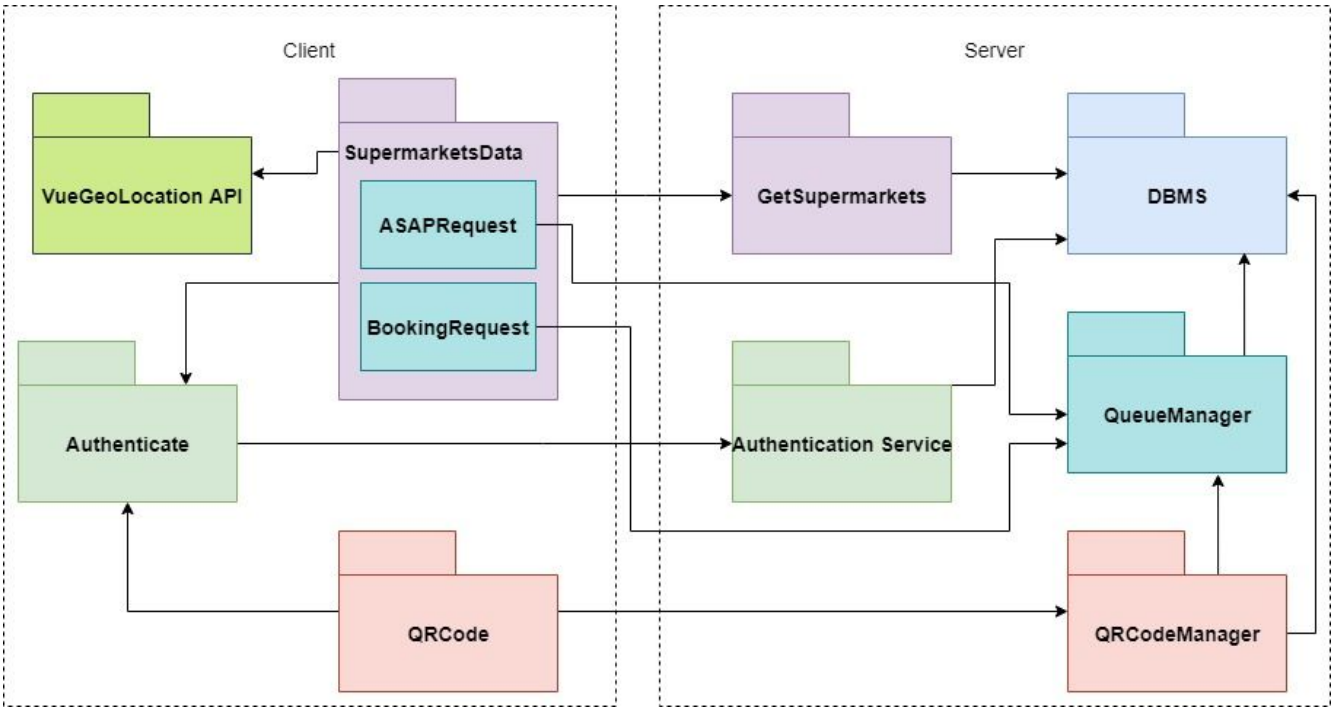

Figure 23: QRCode client component integration

5.4 Test Plan

5.4.1 Objectives and Scope

Testing is an essential part of Software development. One of the main reasons for this importance is that Hardware and Software is continuously evolving and changing, this causes that the components keep changing but not all of them in the same way and speed. Several types of testing phases and procedures are available to achieve this activity in an optimal way that can better help us obtain the best results.

For the development of this software, two types of testing will be performed. The first will be unit testing where single parts of the codes are tested; one of the characteristics of this type of testing is that it is fast to perform and not very demanding, this means that it will be performed in the developer's personal computers and permits to obtain immediate feedback. The second will be integrated testing, which gives more feedback on the integral part of the system, this usually requires the use of more time and resources, but it is important to understand the general behaviour of the software. Both of these kinds of testing will be performed, to be able to achieve full software functionality with the least amount of bugs and errors that could be present in the software release.

Besides the two kinds of testing (unit and integrated), the testing can be divided in two different stages: functional and performance testing.

5.4.2 Functional testing

Functional testing will be performed manually and run on a version very close to the application to be released, its main objective is to verify that the software is behaving as is expected by the user. This functional testing will be divided into the following components:

1. Internals
    a. Functions will be tested, the objective of this is to achieve what the user is expecting the application to do.
    b. Test code will be near to real code, therefore the real code.
    c. IDE that reports bugs clearly
2. Connections
    a. Target groups of modules and validate the interactions between them
    b. Mocking scaffolding will be performed during the integration testing that can be used as a dummy.
    c. Connections interaction testing
    d. Tools proposed to be used:
        i. Wiremock for API level (integration testing at HTTP level)
        ii. Pytest for component level in python
3. User functionality
    a. End to End testing (this involves the testing of the higher level part of the application) to bring the most confidence to test what the user is expecting
        i. To be performed done manually
        ii. Will require to spin up a testing mock
        iii. Tools:
            1. Robot framework
            2. Postman
            3. Selenium for front end testing

5.4.3 Performance testing

This will be the next level of testing, which has the main characteristic of being non functional. Will be performed after E2E testing, simulating the real world, and will take care of measuring: speed, times of interaction, amount of concurrent users and efficiency. Since it requires more testing it will not be performed manually due to the fact that it requires more hardware that simulates how real users interact. The main tool proposed to be used is Heroku, a platform available as Platform as a Service utilized mainly for building, delivering, monitoring and scaling apps.

The main objective of this testing phase is to test scenarios and if possible to be performed in non released versions to monitor how users are behaving on the version of the software and use the logs of the servers.

5.4.4 Test logistics

Software developers that have experience in testing will be performing the associated activities for testing following the test execution steps by doing a Bottom-up integration testing strategy where the lower level modules are tested first to avoid spending time to wait for all modules to be developed. The following are the general steps:

1. Preparation
2. Code execution
3. Assertions
4. Tear down

5.4.5 Testing Environment

The testing environment will be integrated by a strong cooperation involving the testing and the development team, broken up in the following schedule:

Operation	**Responsible**	**Time**
Test specifications	Testing designer	20 hours
Execute testing	Testing engineer	15 hours
Results and reporting	Testing engineer	10 hours
Total		45 hours

5.4.6 Deliverables

The testing deliverables will consist in the following documents:

1. Test log files
2. Testing bugs analysis report
3. Solutions to bugs
4. Test traceability matrix
5. Final testing and results report

6. EFFORT SPENT

Jesus Rodrigo Cedeño Jimenez	30 hours
Diego Andres Cumming Cortes	30 hours
Angelly de Jesus Pugliese Viloria	30 hours

7. REFERENCES

1. Getting Started - vue.js. (2021). Retrieved 4 January 2021, from https://012.vuejs.org/guide/
2. Contemporary Front-end Architectures. (2019). Retrieved 4 January 2021, from https://blog.webf.zone/contemporary-front-end-architectures-fb5b500b0231
3. TEST PLAN: What is, How to Create (with Example). Guru99. Retrieved: 9 January 2021. From: https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html
4. Integration Testing: What is, Types, Top Down & Bottom Up Example. (2020). Retrieved 9 January 2021, from https://www.guru99.com/integration-testing.html