Politecnico di Milano

SOFTWARE ENGINEERING II PROJECT

RASD

REQUIREMENTS ANALYSIS AND SPECIFICATION DOCUMENT

Professor:
Matteo Giovanni Rossi

Authors:
Stefano Vanerio
Agnese Straccia
Cristiano Serafini

Date:
23 December 2020

Version:
Version 3.0

CONTENTS

1 Introduction

1.1 Purpose

This document, “Requirements Analysis and Specification Document”, provides a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context. The introduction includes the objectives of the project. Its purpose is to give a specific description in terms of functional and non-functional requirements, goals and domain assumptions, world and shared phenomena, scenarios and use cases and all the features and specifications that characterize the behaviour of the system. It explains both the application domain and the system to be developed in order to guide the developers in the realization of the software, but also it provides a high-level representation for Customers and Users.

1.2 Context

The coronavirus emergency has put a strain on society on many levels, due to many countries imposing lockdowns that allow people to exit their homes only for essential needs and enforcing strict rules even when people are justified in going out (such as limiting the number of accesses to buildings and keeping a distance of at least one meter between people). In particular, grocery shopping, one of the most essential needs, can become a challenge in the presence of such strict rules. Indeed, supermarkets need to restrict access to their stores to avoid having crowds inside, which typically results in long lines forming outside, which are themselves a source of hazards. So, in this context we take advantage of the technology which helps us to avoid situations of high spread of the infection. The system described below is an easy-to-use application called CLup, which intends to provide a way to improve this situation by helping people to exit their home only for real needs and to avoid gatherings of people for instance in front of grocery stores.

1.3 Scope

The aim of the system is to ensure the distance between people and to develop an easy-to-use application that, on the one side, allows store managers to regulate the influx of people in the building and, on the other side, saves people from having to line up and stand outside of stores for hours on end. The software has to guarantee some useful functionalities on the application, the main ones are:

- “Lining up” feature: this functionality allows users to “line up” from their home and to retrieve an online ticket without going in front of the store and raising the possibility of creating crowds. In this way, people leave their home only when it is their turn and the system warns them about the expected time needed to reach the store.

- "Book a visit" feature: the application must give the opportunity to make a store visit reservation by inserting date, time and an expected duration of the visit and also by selecting a list of items they would like to buy. Moreover, the system stores reservation data in order to infer the duration time of the visit for long-term customers. This would allow the application to plan visits in a better way, for example allowing more people in the store, if it knows that they are going to buy different things, hence they will occupy different spaces in the store when they visit.

- Suggest alternatives and options feature: the application must suggest to the customers alternatives slots in a day/time range for visiting the store, to balance out the number of people in the store, or different stores of the same chain if the preferred one is not available.

This system is useful also for store managers who can manage the limited number of people inside the building and monitor accesses to the stores.
The application, in order to be effective, needs to be very easy to use, in this way it can be used by a large range of people including people who do not have access to the required technology, indeed for this purpose there is also the possibility to hand out "tickets" on the spot, thus acting as proxies for the customers.

After this first description of CLup, we can move on to the description of the goals we want to meet by combining domain assumptions and requirements (respectively sections 2.4 and 3.2.1).

1.3.1 Goals

- G1: Users should be able to virtually line up through the app to enter the store.

- G2: Store managers should be able to regulate the influx of people inside the building.

- G3: Users should be able to "book a visit" to the supermarket in order to go shopping.

- G4: Users should be able to know different alternatives and suggestions.

    - G4A: Users should be able to know alternative time slots for visiting the store, during the booking, if the preferred one is not available.

    - G4B: Users should be able to know different stores of the same chain if the preferred one is not available.

    - G4C: Users should be periodically informed about different free day/time slots for visiting the store.

1.4 Glossary

1.4.1 Definitions

- QR code: is a type of Matrix Barcode, it consists in a group of black squares on a white background. It is possible to read it with a device like a camera. It contains some data codified in patterns that are present in horizontal and vertical lines.

- OTP: is a password that is valid only for one transaction or session in a computer system or digital device. It consents to be protected from some types of attacks and grants a major level of security.

1.4.2 Acronyms

- API: Application Programming Interface
- GPS: Global Positioning System
- OTP: One Time Password
- QR Code: Quick Response Code
- RASD: Requirements Analysis and Specification Document

1.4.3 Abbreviations

- B-Visit: Book a Visit
- CS: Customer Scenario
- DA: Domain Assumption
- G: Goal
- L-Up: Line Up
- MS: Manager Scenario
- R: Requirement
- SP: Shared Phenomen
- UCD: Use Case Description
- WP: World Phenomenon

1.5 Document Structure

Here it is brief discussion about the document structure and how it is divided into sections.
The document is structured as follows:

- **Introduction**: this section is a general introduction including the description of the problem, the purpose and the scope in which the goals of the system to be satisfied are pointed out. There are also the Acronyms and Abbreviations sections to make the document easier to read.

- **Overall Description**: in this section there is a description of the behaviour of the system through the State Diagrams representation in some critical situations. Class

Diagram is used to highlight the model structure and to show relationships between the main entities involved. There are also the main functions provided by the system. At the end Domain Assumptions and World and Shared Phenomena are identified.

- **Specific Requirements**: this is the body of the document. First, the product under consideration is described in more details through the external software and hardware interfaces. The main part is represented by the Functional Requirements and it is shown how these, together with the domain assumptions, lead to the accomplishment of the goals. The section ends with the Uses Cases and Scenarios identification and The Traceability Matrix which matches requirements and uses cases.

- **Formal Analysis Using Alloy**: here the main functions are formally described in Alloy Language used to verify the system model defined by the specifications and also some Metamodel graphs are shown.

- **Effort Spent**: it shows the hours spent for each section by the group members.

- **References**: references used for the document.

2 Overall Description

2.1 Product Perspective

As a result of the previous introduction and the scope definition from the first section, we can now continue our analysis focusing on the interactions of the system.

In the first subsection we are going to see how the system communicates through the external interfaces with all the services necessary for the correct functioning of the application.

After that we are going to analyze how the various components inside the system interact to each other using two different types of diagram. The first one, for a more structural evaluation, is a UML diagram containing all the most important parts of the system. In the end, we use state diagrams to underline the dynamic characteristics of the most critical interactions.

2.1.1 System Interfaces

In this section we are going to describe the external interfaces with which the system had to interact in order to provide all the services. In this part, the interfaces used by the system are presented and, later in the document (Section 3.1), we will present the “users” interfaces. That analysis, in a more complete way, will continue in the Design Document.

In order to achieve the goals, the system needs to interact with some external services. They are shown in the picture below (Fig. 1):

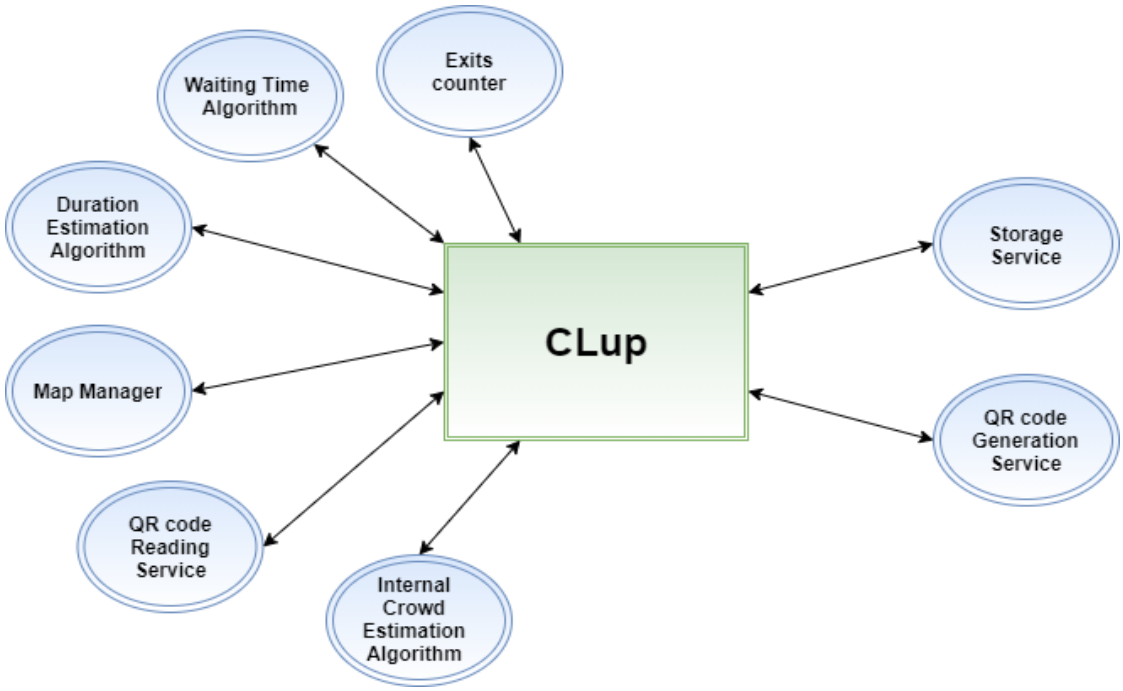

Figure 1: System Interfaces

In this description we will divide in two different types the external interfaces underlined in the picture above:

- First type (the system uses them to perform algorithmic operations):
    - **Duration Estimation Algorithm:** is used to estimate the duration of the visit of a long-term customer based on the duration of his/her previous visits. It needs to compute a large amount of data and estimates different duration for different day/time ranges.
    - **Waiting Time Algorithm:** is used to estimate the waiting time that a user must wait before his turn. It uses estimations based on previous similar situations, same days of the week and same period.
    - **Map Manager:** is used to estimate the distance between the customers and the supermarkets. It computes the optimal path knowing the current state of all the streets of the city (work in progress streets, traffic jams, ...) and estimates the time needed to reach the supermarket from the current position of the device of the customer. It needs to be updated frequently and knows the real-time changes of the street situation.
    - **QR code Reading Service:** is used to read the QR code generated during the "LineUp" or "Book a Visit" request formulation. It is necessary to check if the current customer is one of the stored ones in the system that has made a request. It computes the visual reading of the QR code and sends to the system the information stored in it.
    - **Internal Crowd Estimation Algorithm:** is used to estimate, thanks to the list of the products chosen from the customer during the formulation of a "Book a visit" request, if it is possible to modify the maximum number of people inside the store without risks for the customers. If a person has to buy only products in a not-crowded section he notifies it to the system. For its correct functioning, the algorithm needs the updated internal map of the supermarket and all the information about the shopping list of the customer. It must implement geometric algorithms (like Binary Space Partitioning for example) for the correct estimate of the distribution in order to respect the "minimum distance" between people.
    - **Exits Counter:** is used to count how many customers left after they enter the supermarket in order to know how many customers are inside the store.
- Second type (the system must rely on them):
    - **Storage Service:** is used to manage all the data useful for the system in order to achieve the goals presented in the previous section.

- **QR code Generation Service:** is used to generate a unique QR code during the "LineUp" or "Book a Visit" request formulation. The QR code must store in its structure a quite simple group of data (customer identifier, day, time, supermarket identifier).

2.1.2 Model Structure

The analysis of the system can now continue with the specification of the internal structure. The following diagram (Fig. 2) represents a high-level UML class diagram; it considers the most important objects and their relations with the final objective to reach the functionalities defined by goals described in the first section.

The main classes in the UML class diagram are:

- **Account**: represents the general user of the application. All the users need to provide a username and a password to identify themselves in the application. A user can be a Customer interested in access to the store or a Manager interested in controlling the influx of people.
- **Customer**: identifies a person who wants to buy something in the supermarket. He must provide his information to the system with a Login functionality. Then he chooses what service he wants to use between "Book a Visit" and "Line Up". He must provide the correct position during the making of the request. Optionally the customer can also provide an email during the registration for the retrieval of lost password.
- **Manager:** identifies the administrator of the supermarket (or the administrator of a group of them). He must provide his information with a Login functionality and then he can change some parameters related to the maximum capacity of the supermarket to avoid crowds inside the store manually.
- **Supermarket:** identifies the supermarket that the customer has chosen for his requests. It contains all the information about the opening hour, the closing hour, the reference to its position, the product availability, the chain (if there is one) and the maximum capacity. It also contains the references to the two different types of requests "Book a Visit" and "Line Up" and the maximum capacity customers for both the services to avoid crowds inside the store.
- **Request:** represents the general request made by the user, each type of request requires some information related to the user identification, the supermarket chosen and a QR code for the access (generated from an external component, *QR code Generation Service*).
- **Book a visit Request:** specific request for the booking of a visit to the supermarket. It contains all the information about the specific time and date when the visit is booked. For the "duration" field it can be filled by the customer or generated by an external data analysis algorithm that estimates the time needed using the customer's habits (*Duration*

Estimation Algorithm). It can also contain a reference to the products that a person would buy or to generic categories of products, needed to help another external component (*Internal Crowd Estimation Algorithm*) to estimate the possible maximum capacity of the store in a specific moment avoiding the crowding of specific sectors in the supermarket.

- **Category:** represents the different categories in which every product of the supermarket is classified. In this way the *Internal Crowd Estimation Algorithm* can easily estimate where a person can move inside the building knowing his shopping list.

- **Product:** represents a specific product that a person would buy. It is used also as a reminder if the customer uses the digital "shopping list". Clearly, it is used from the *Internal Crowd Estimation Algorithm.*

- **Notification:** corresponds to a periodic notification sent to all the customers interested in this specific service provided by the application. It provides all the useful information to the customer about the day/time range when it is possible to "Book a visit" when the notification is generated.

- **Line Up Request:** specific request for the digital lining up functionality. It contains the identification number of the ticket (a sequential number), the estimated waiting time (thanks to the *Waiting Time Algorithm*) and the estimated time to reach the supermarket when a notification is shown in the application (for example: "Your turn is imminent!"). It is possible to create a request also with a physical Ticket Machine placed outside the supermarket (to allow access to the supermarket to all the customers, also the ones without a digital device).

- **Ticket Machine:** corresponds to a physical machine placed outside the supermarket where it is possible to make simplified tickets only for the "Line Up" service. It needs only interaction with the Line Up functionality and has a very user-friendly interface.

- **Position:** stores the coordinates obtained by the GPS of the device in case of user interaction or the fixed coordinates corresponding to the position of a specific supermarket.

- **Street:** represents all the streets of a defined city, it is used for filtering all the possible supermarkets in the city using the application.

- **City:** represents the entire area of a defined city. This class is important to be considered as a filter in the functionalities of the system during the selection of a supermarket.

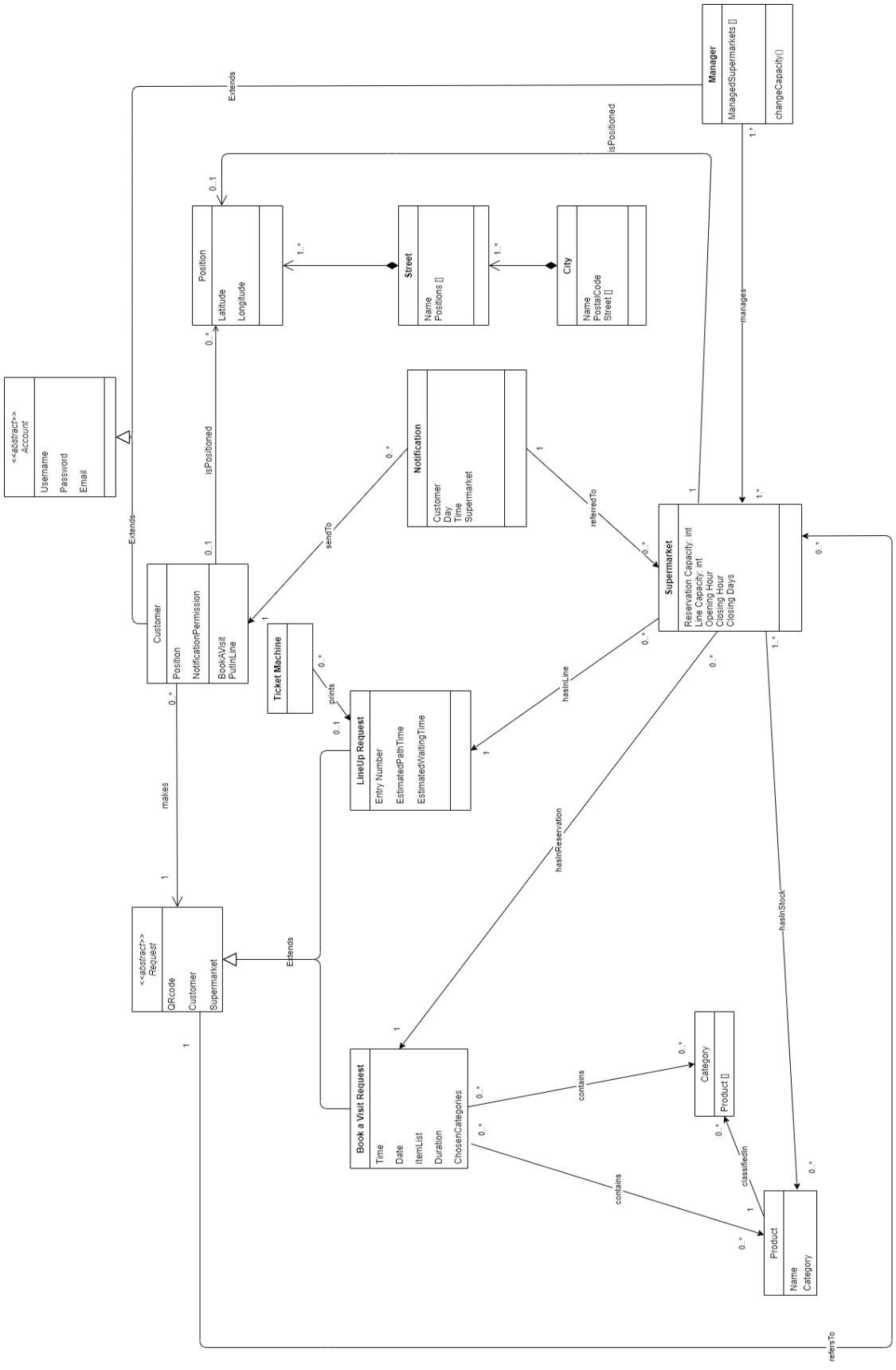

Figure 2: High-level UML class diagram

2.1.3 State Diagrams

In the next section we are going to analyse the main functionalities of the system in a dynamic situation, it is important to highlight the behaviour of a single object in response to a series of events in the system. We will use state diagrams to describe the most critical aspects of the objects previously described in the UML class diagram (Fig. 2).

Line up

This diagram starts with a “LineUp” request. It is important to remember that each request will be refused by the system if the requesting person is already in the related queue.

The diagram (Fig. 3) starts when a notification is received in the unprocessed state. First the system checks if the user is already in the queue. If the answer is positive, the notification is rejected and the diagram ends, otherwise the system needs to store the user information and make some operations. The system puts the customer in the virtual line and then it needs to use the functionalities provided by the external services to calculate a reasonably precise estimation of the waiting time and the time needed to reach the store.

The “LineUp diagram” ends with the generation of the QR code and the communication to the customer that the request is successful.

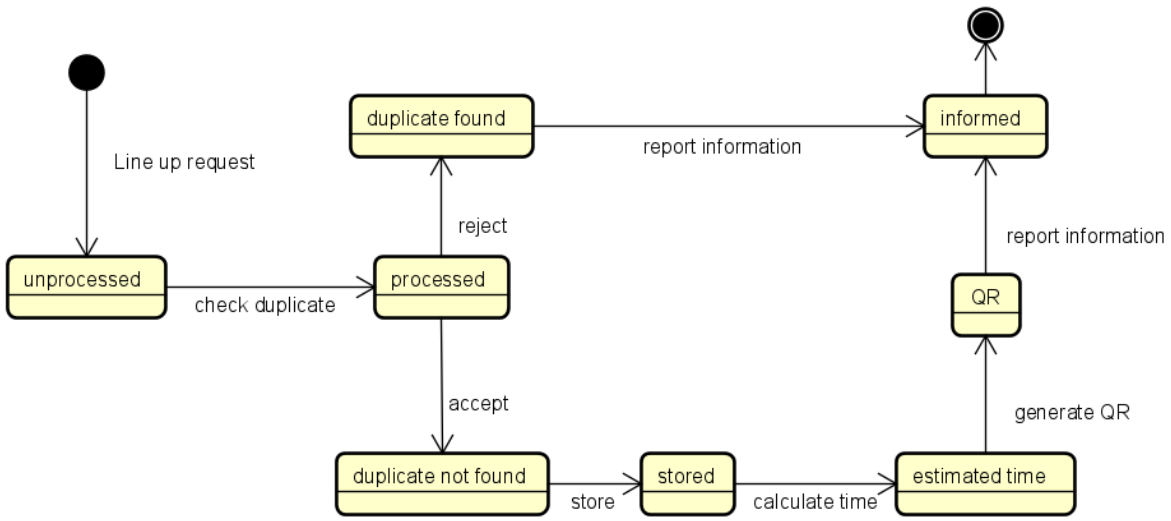

Figure 3: State Diagram “LineUp” request

Duration

In this diagram (Fig. 4) we want to highlight how the duration of the visit into the supermarket is checked. The diagram starts when a booking request is received. At first the system must check if the customer has inserted the approximate expected duration of the visit or not. If the answer is positive the request is stored.

If he/she is a long customer and the duration field is not inserted, the duration diagram continues with the estimation by the external service of a reasonable duration for the visit. It ends with the storing of all the information related to the booking request.

After that, the system will check if the store is fully booked and, if the answer is positive, further checks of the shopping list will be carried out (Fig. 5) otherwise the reservation will be stored, and the ticket is generated.

In other cases, the diagram terminates reporting the problem encountered.

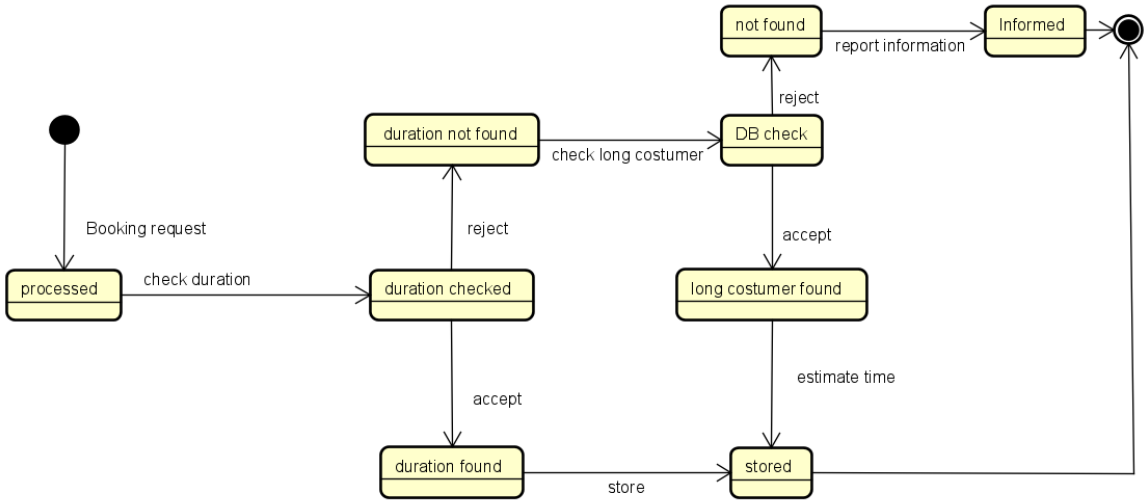

Figure 4: State Diagram Duration Checking

Shopping List

Shopping list is a quite complex functionality, for this reason we think it needs a state diagram to better understand how it works (Fig. 5). It is important to point out that the diagram only begins if there are no available reservation slots for “book a visit” functionality and this means also that the duration check was successful (Fig. 4).

At first the system checks if the customer has inserted the grocery list to understand if the internal departments he will visit will be crowded or not. If the customer has not inserted the shopping list the system informs the user and then the diagram terminates.

Otherwise, if the departments are crowded the system will not be able to allow reservation, it will notify at first other available time slots in the same supermarket and then the supermarkets of the same chain that may still have free slots. Then the diagram terminates. Otherwise, in the case of departments not crowded, the system allows the reservation and the diagram terminates.

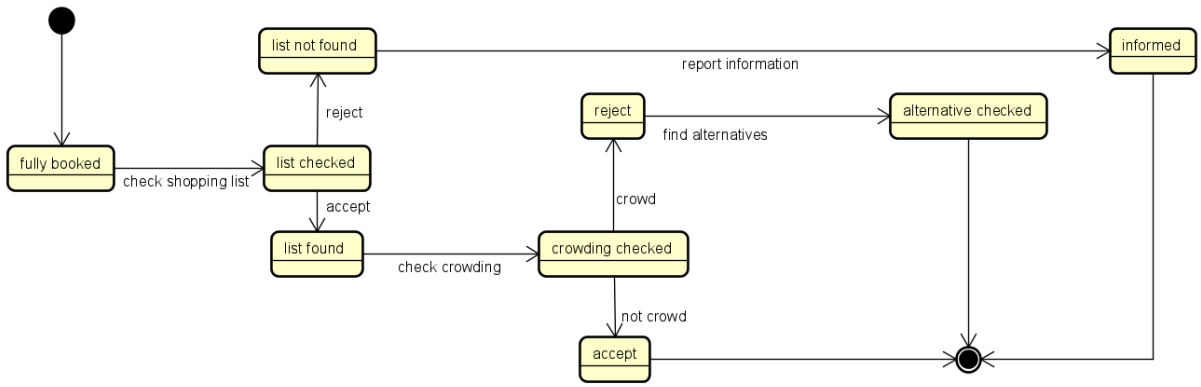

Figure 5: State Diagram Shopping List Checking

Periodic Notifications

This Notification diagram is important to better understand how notifications are managed (Fig. 6). This diagram will be repeated periodically by the system. It starts by scanning the list of users in the database, then for each customer the system checks whether it can send periodic notifications. If the answer is positive it checks if the customer has not received them recently and if he/she is also a long customer (we can know his preferred slot time and data) in that case, it will send a notification to the customer and the diagram terminates, in all other cases it ends without sending anything.

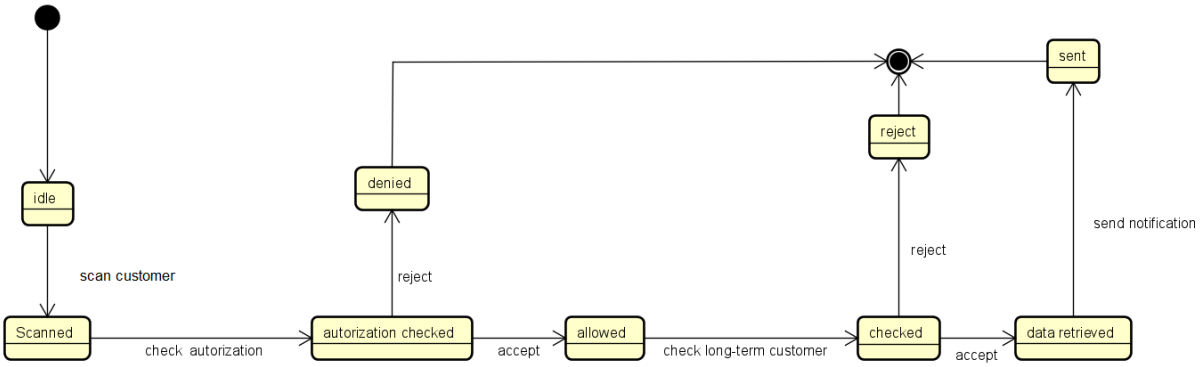

Figure 6: State Diagram Notification Authorization

2.2 Product Functions

In this subsection we are going to briefly discuss the main functions that the application should provide to the users. CLup is an application that mainly helps to limit the contagion between people by allowing them to do shopping in the supermarket in a safer way, but it also helps store managers to avoid creating crowds inside the store. In particular, the customer can make two types of request: Line Up request and Book A Visit request, depending on if he wants to either get an online ticket to enter the store or make an online booking with also the selection of items. This application cannot check that people, who are inside the building, keep the right distance from each other, but it takes it for granted. Another function described is the Notification Function, in this case it is important to remark that this functionality is active only for users who have allowed notifications on their devices. The three functions described above are available only for customers and all of them require a login registration with username and password and an optional email to possibly restore a forgotten password. Whereas the Manager Function allows store managers to better manage people inside the building and monitor entrances. In this case, after a registration when the manager also provides the supermarket identifier (or identifiers because he can also have more than one), if he wants to change capacity parameters, he must login with his credentials and with an OTP password that guarantees his authentication (major details about that feature are provided in the Design Document).

2.2.1 Line Up Function

Customers are allowed to virtually line up through the application by means of a simple click. In this way, they can avoid physically waiting in front of the supermarket and eventually create crowds, which is not an ideal situation in this context. This function is granted only if the user has enabled the GPS on the device used. By choosing this functionality, the software must go through the following stages:

- Supermarket choice: The customer must decide the supermarket in which he wants to go shopping, from a recommended list of the nearest ones based on his current position via GPS. If the preferred one is not in the list, he can also insert it or filter the list by street or city.

- Duplicate check & storing: Before storing the request, the duplicate check is performed to prevent a request from a person who already has a ticket that has not yet been used. If no duplicate is found, the system stores the request.

- Time calculation: In order to guide the customer to reach the store, it uses the *Map Manager* to estimate the time needed from the current position of the customer device to reach the store. The customer is supposed to follow the shortest path, otherwise, if he were too early or too late, it could still create crowds at the entrance which does not ensure compliance with the safety distance.

- Ticket release: At the end of the process, the system generates the QR Code, by using the *QR Code Generation Service*, that the customer must scan at the entrance to the supermarket in order to go in. Moreover, the customer is provided with a unique number which represents his turn and with an estimation of the waiting time by using *Waiting Time Algorithm* so he can get an idea on when to go out.

All the shops are supposed to have a Ticket Machine at the entrance, this allows people also to "hand out" a physical ticket without using the application on their devices. These tickets are considered as Line Up tickets by the system.

2.2.2 Book a Visit Function

The application also offers the possibility to book a visit to go to the store, in order to plan doing shopping in a more technological way also by selecting items that the customer wants to buy. The system ensures that anyone who makes a reservation request for a certain date and at a certain time does not have to wait for entering because the store certainly has not reached the maximum number of people. This means that, in the meantime from the booking time to the actual visiting time, there cannot be a certain number of line up requests that have determined the filling of the store. This because the system divides the maximum store capacity into two sub-capacities, one relating only to the line up request and the other to the booking requests, in this way there are no conflicts. After having clicked on this feature, the customer can fill the request by the following steps:

- Supermarket choice: the customer must decide in which supermarket to make the booking. This selection is done exactly as in Line Up Function and during the filling of the request the customer can always change his selection.

- Slot selection: the customer must select date and time of the booking. This field is mandatory.

- Items selection: at this point customer can select which product to choose from a list, or he can also choose only categories. We have assumed that all the products on the list are available, in such a way there is no possibility that a person who goes to the supermarket does not find any of the pre-selected products. This field is not mandatory, but if inserted it allows the system to better manage internal departments of the store, in fact if two persons purchase different products the system can decide to allow more people even if the shop is already full, by applying *Internal Crowd Estimation Algorithm.*

- Duration estimation: the customer can also insert an estimation time of the visit duration. This field is mandatory for non-long-term customers, on the other side is optional for long-term ones, indeed in this case the duration can be inferred by the system itself depending on his previous visits by applying Duration Estimation Algorithm.

2.2.2.1 Suggested Alternatives Function

When the “Book A Visit” request is sent, if the selected date and time slot is not available, the system suggests some alternatives to facilitate the customer choice. In particular, if the customer has inserted the list of items he wants to purchase, the system informs him that the store is fully booked at that time and it advises him to change slot or also to change supermarket if all the available suggested slots are not what the customer wants (Fig. 19). Otherwise, if the customer has not inserted his shopping list, the system gives him the possibility to change slot (or also supermarket as in the previous case) or to insert the shopping list, in such a way the system maybe can allow the customer to go even if the store is full if the request passes the crowding check based on his items selection as shown in the Mock-ups (Fig. 17).

2.2.2.2 Notification Function

This function is an additional feature that allows customers to periodically be informed regarding free slots, in terms of date and time, of a particular store. It is optional because the customer can decide whether to turn it on or not, but if active, it requires the customer's device to have active notifications. Once activated, it can be deactivated again. The system generates and sends notifications depending on data stored in the system: for instance, if a customer usually goes to do shopping on a certain day, the system knows it and it notifies the best time to go to the store in which it is not too crowded.

2.2.3 Manager Function

A store manager is a particular user that cannot make requests; indeed, his application interface is different and includes:

- the supermarket he runs: he can have even more than one and he can also have the option of adding others or deleting one existing.

- number of allowed persons: he can decide how many people are allowed at the same time in the store, he can also change this number to manage entrances. He communicates with the service at the entrance that scans QR codes, in this way when he changes the number, the machine is updated and if needed it stops to scan tickets until the correct number of people in the store is reached.

2.3 User Characteristics

In this section we want to specify in a more detailed way users characteristics. We have identified two types of User, based on different requests that can make: typical Customers that is a general client of the application and store Managers, they interact with CLup through different interfaces. It is important to remark that also Ticket Machine can be considered as a special User, indeed, by releasing physical tickets at the store entrances, it interacts with the system sending Line Up requests, even if it has no username or password.

- **Customer:**
    - Must have a device to login to CLup.
    - Must login with username, password and an optional email and always inserts correct data while registering.
    - Must enable the GPS in his device.
    - Can make a Line Up request, by choosing the supermarket.
    - Can make a Book A Visit request, by choosing the supermarket, date and time, the list of items and the duration of the visit.
    - Can be informed about other options of different supermarkets.
    - Can retrieve QR code and a ticket number.
    - Can turn on periodic notifications to be informed about available free slots to go to the store.
    - Is supposed to always follow the shortest path from the current position to the store.

- **Manager:**
    - Must login through a web interface with username, password and email.
    - Must insert an OTP sent by the system and a unique identifier of the store in order to be recognized.
    - Can manage one or more supermarkets.
    - Must decide the number of allowed people in the stores.
    - Can change the number of allowed people in the store.

2.4 Domain Assumptions

Domain properties are descriptive assertions assumed to hold in the world to better explain properties that cannot be controlled by the system itself. They are useful to describe the world in which CLup works and they help to accomplish the goals described in Section 1.3.1 together with the functional requirements, described later in Section 3.2.1.

DA1: Customers always insert correct data while registering to CLup.

DA2: If a user makes a reservation he effectively goes to the supermarket in time.

DA3: Users must follow the shortest path to the shop.

DA4: The devices used have date and time working properly.

DA5: Internet connection works always without errors.

DA6: QR code is always readable.

DA7: Maps of Italy are well known, complete and up to date.

DA8: Product disposition inside the shops is always known.

DA9: Users must buy what they previously have selected.

DA10: Opening and closing hours of the shops are always known.

DA11: Two persons in the shop always keep distance of at least one meter.

DA12: All shops have a unique identifier.

DA13: The number of people permitted by law in the shops is known.

DA14: Estimated time from one place to another is always correct.

DA15: Estimated time of a person in a shop is always what he has previously inserted.

DA16: Estimated time of a long-term customer is always correct.

DA17: All users have devices' notifications up and running.

DA18: All shops have a ticket release machine at the entrance.

DA19: All shops have a ticket QR code reading machine (or something similar) at the entrance, in order to read the tickets.

DA20: All items selectable in the application are always available in the shop.

DA21: The system knows all the positions of supermarkets.

DA22: The GPS module of the devices on which CLup runs always works correctly and has an accuracy of 2 meters.

DA23: All shops have a sensor (or something similar) that takes into account customers' exits.

2.5 The World and the Machine

Now we are going to point out the world and shared phenomena of CLup. The world is the environment, the partition of the reality affected by machines including stakeholders and any external system. The machine is the system we must develop. Therefore, shared phenomena are phenomena controlled by the world and observed by the machine and vice-versa.

N°	WORLD PHENOMENA
WP1	No queue outside the supermarket.
WP2	Situation of necessity for which it is necessary to regulate the number of people.
WP3	The customer goes to the shop.

Table 1: World Phenomena

N°	SHARED PHENOMENA
SP1	The customer accesses the application.
SP2	The manager accesses the application.
SP3	The customer makes a booking request.
SP4	The customer makes a line up request.
SP5	The customer looks for available supermarkets.
SP6	The manager regulates internal capacity.
SP7	The customer scans the ticket upon entering.

Table 2: Shared Phenomena

3 Specific Requirements

This section is important to better explain all the requirements needed by our system in order to work properly and achieve all the goals previously described.

3.1 External Interface Requirements

3.1.1 User Interfaces

In the following section is possible to look at the mock-ups of the most important parts of the application. A more detailed analysis of them is provided in the Design Document and also a chart showing how the different screenshots are linked together will be present in the document. This brief preview is shown in order to fully understand how the different functionalities required by the goals (Section 1.3.1) are reached and how the users interact with the system in the Use Cases of the following section (Section 3.3.3).

In the first part are shown the screenshots for the “Line Up” functionality, then are presented the screenshots of the “Book a Visit” functionality and in the end are presented the Ticket Machine interface and the “Manager” functionality used by the Supermarket’s Manager.

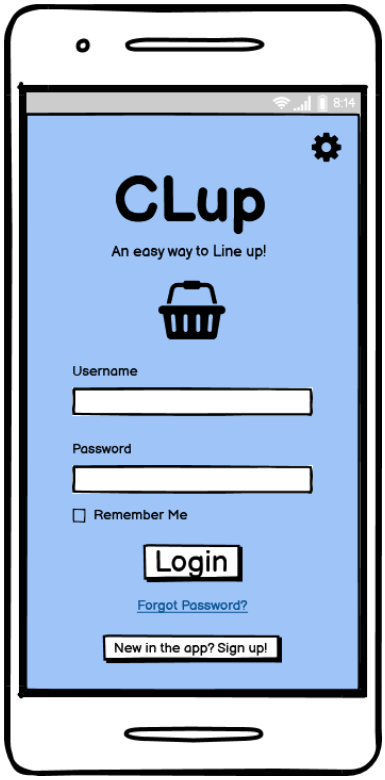

Figure 7: Login Customer

Figure 8: Registration Customer

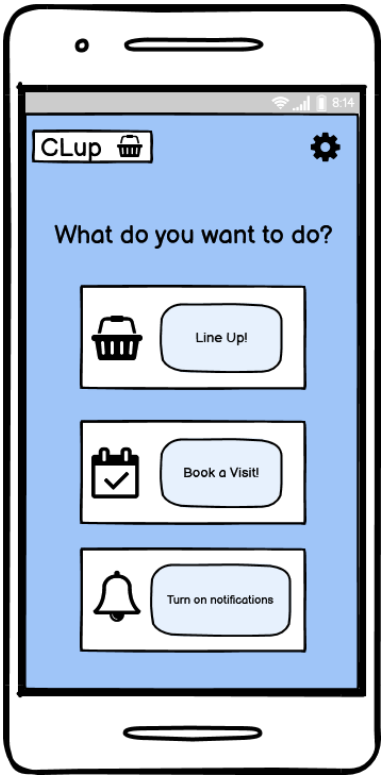

Figure 9: Home Page

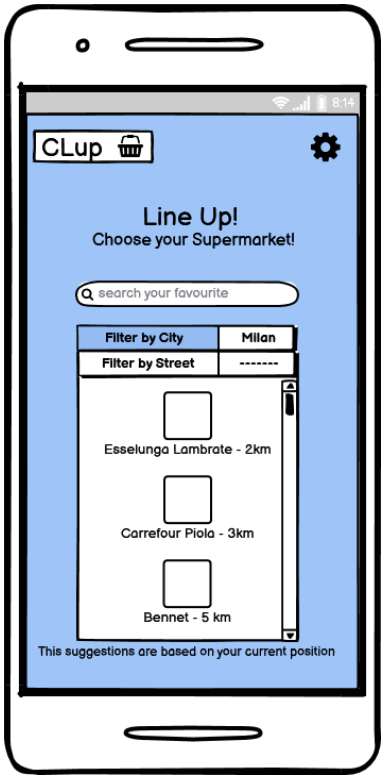

Figure 10: Choice of Supermarket (L-Up)

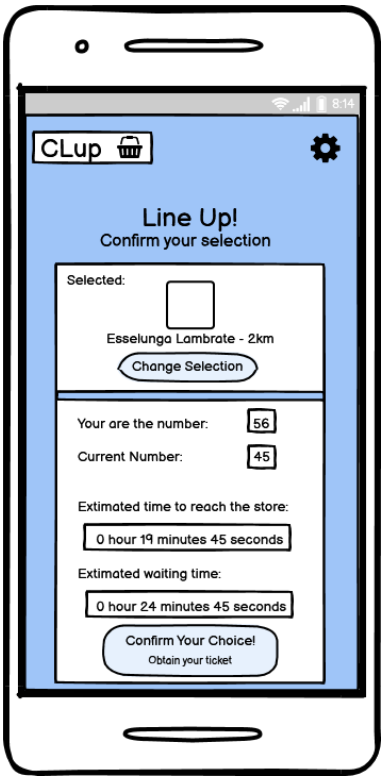

Figure 11: Ticket Preview (L-Up)

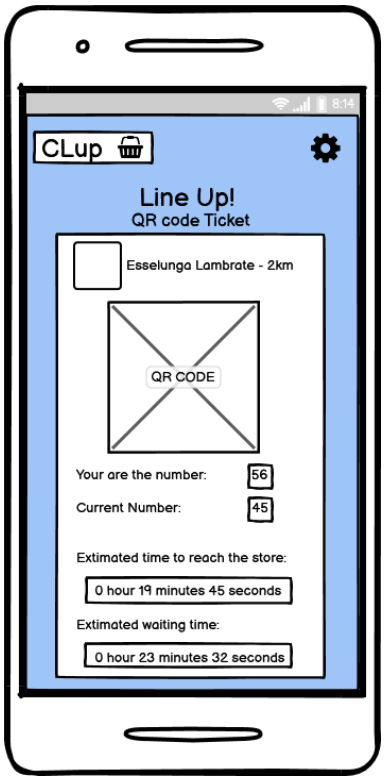

Figure 12: Ticket (L-Up)

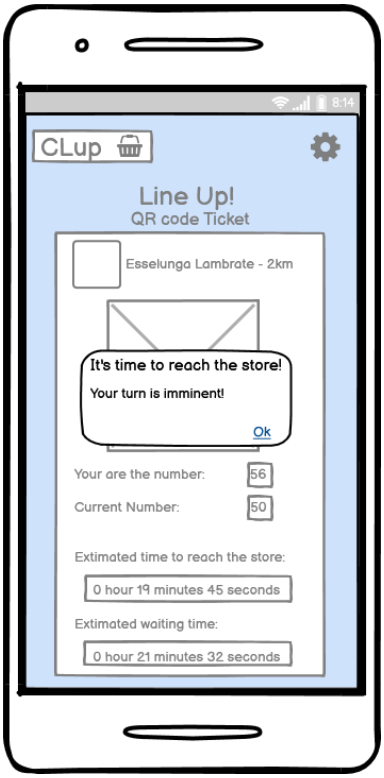

Figure 13: Ticket Notification (L-Up)

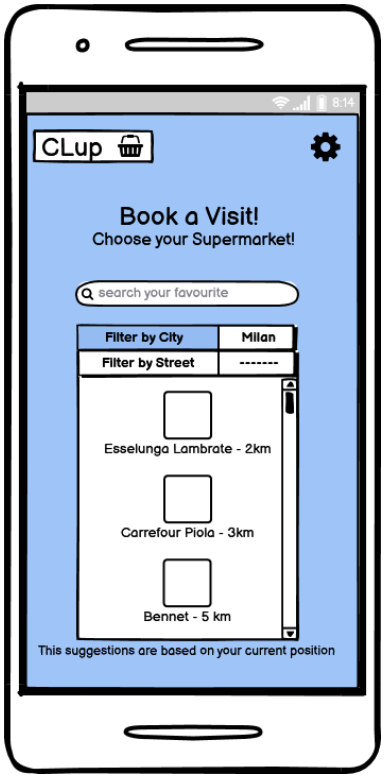

Figure 14: Choice of Supermarket (B-Visit)

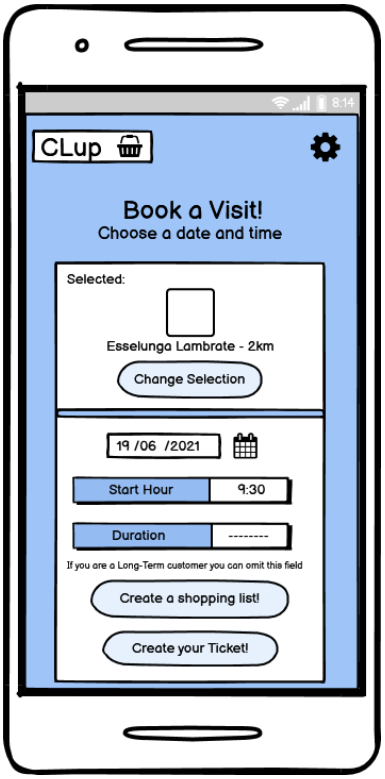

Figure 15: Time Slot Choice (B-Visit)

Figure 16: Fully Booked Notification (B-Visit)

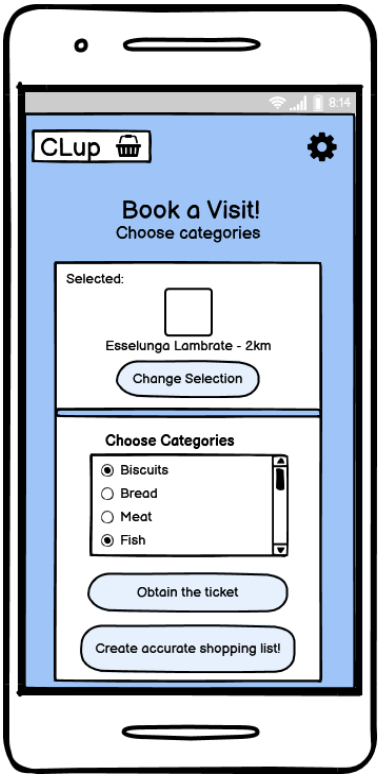

Figure 17: Choice of Product Categories

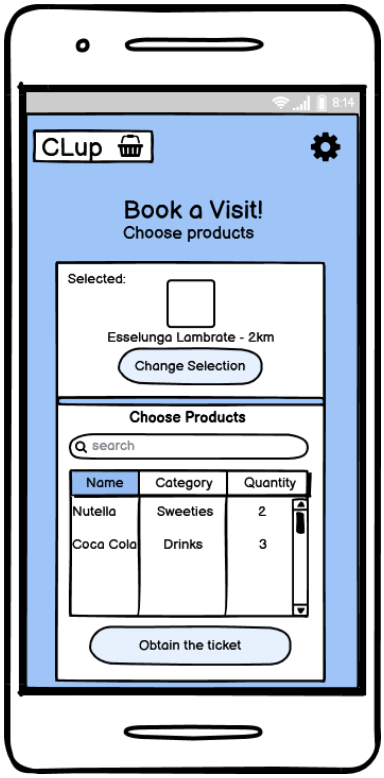

Figure 18: Shopping List

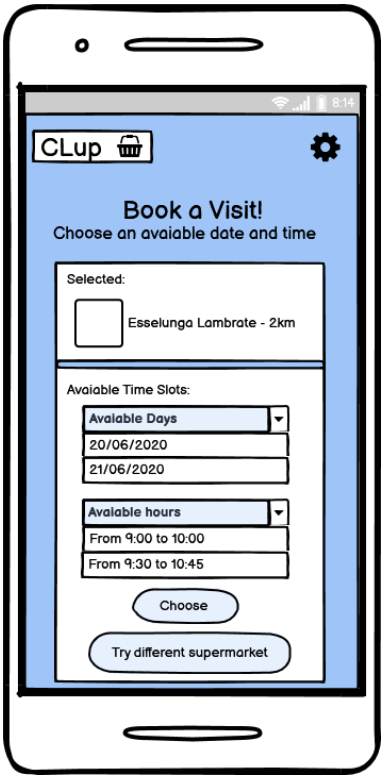

Figure 19: Suggested Time Slots

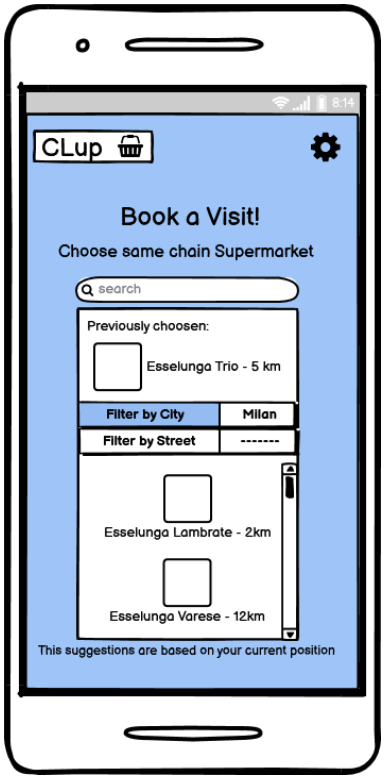

Figure 20: Alternative Stores

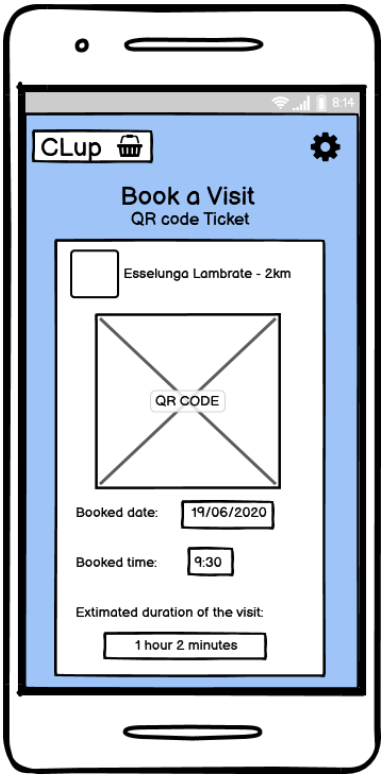

Figure 21: Ticket (B-Visit)

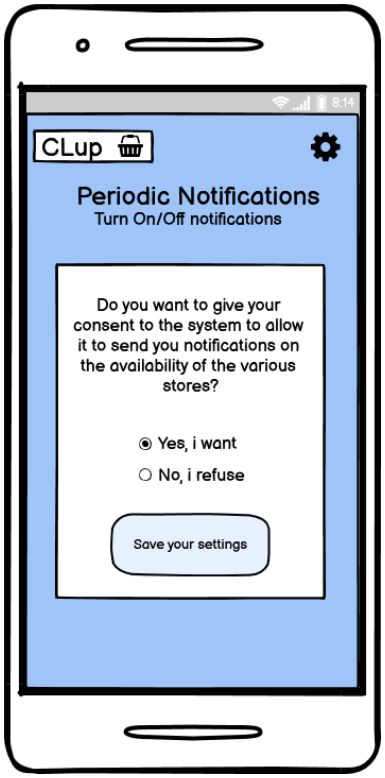

Figure 22: Notification Management

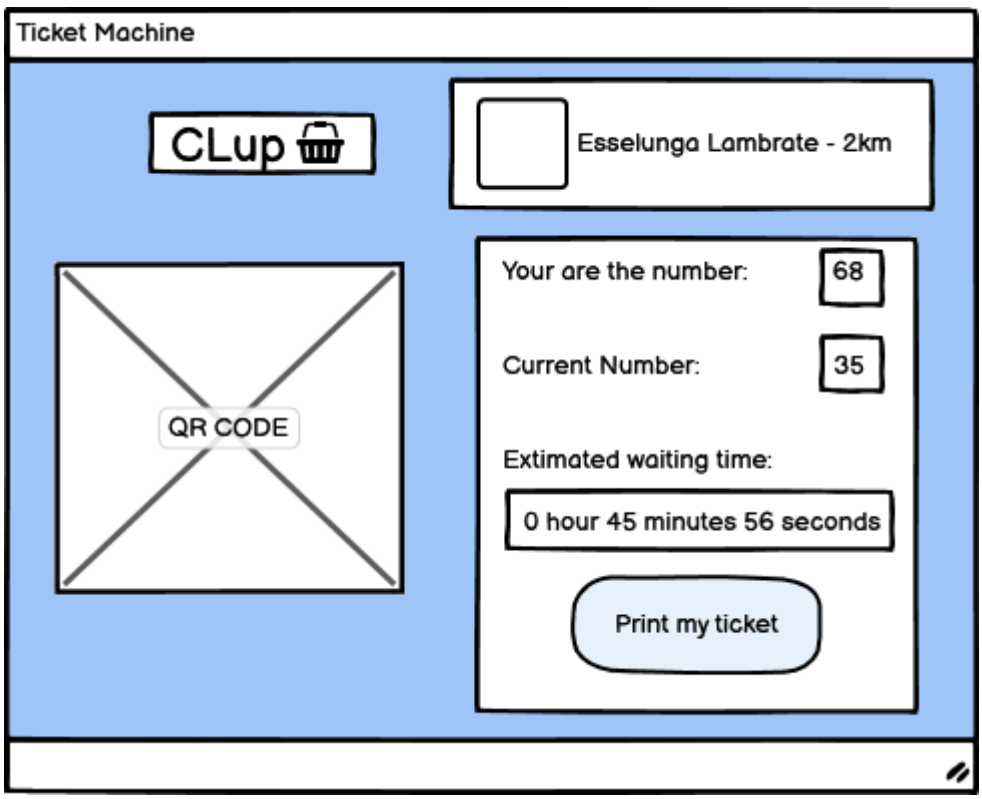

Figure 23: Ticket Machine (L-Up)

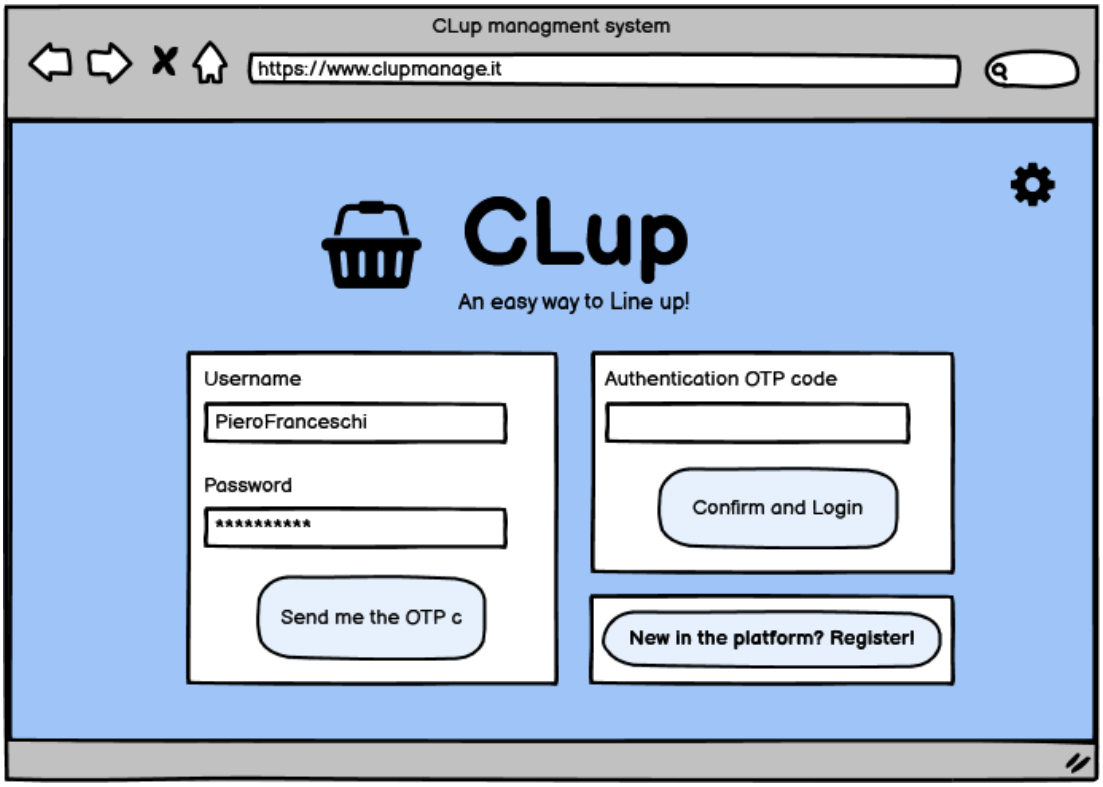

Figure 24: Login Page Manager

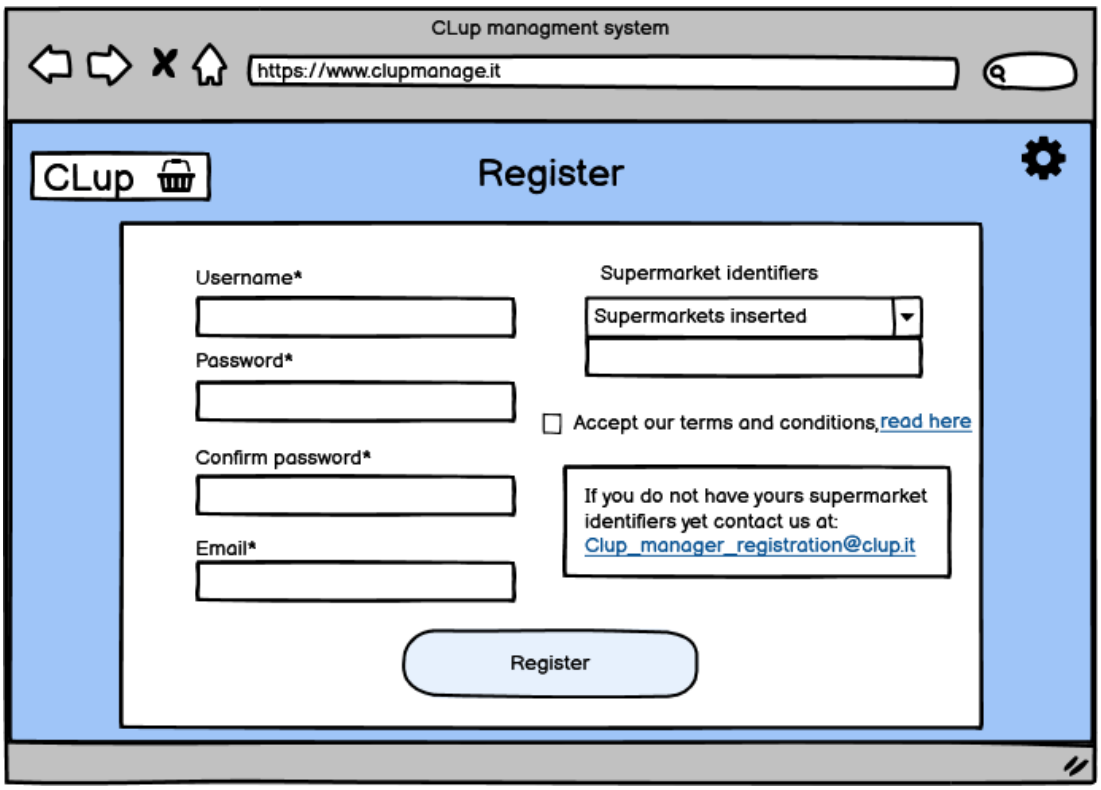

Figure 25: Registration Page Manager

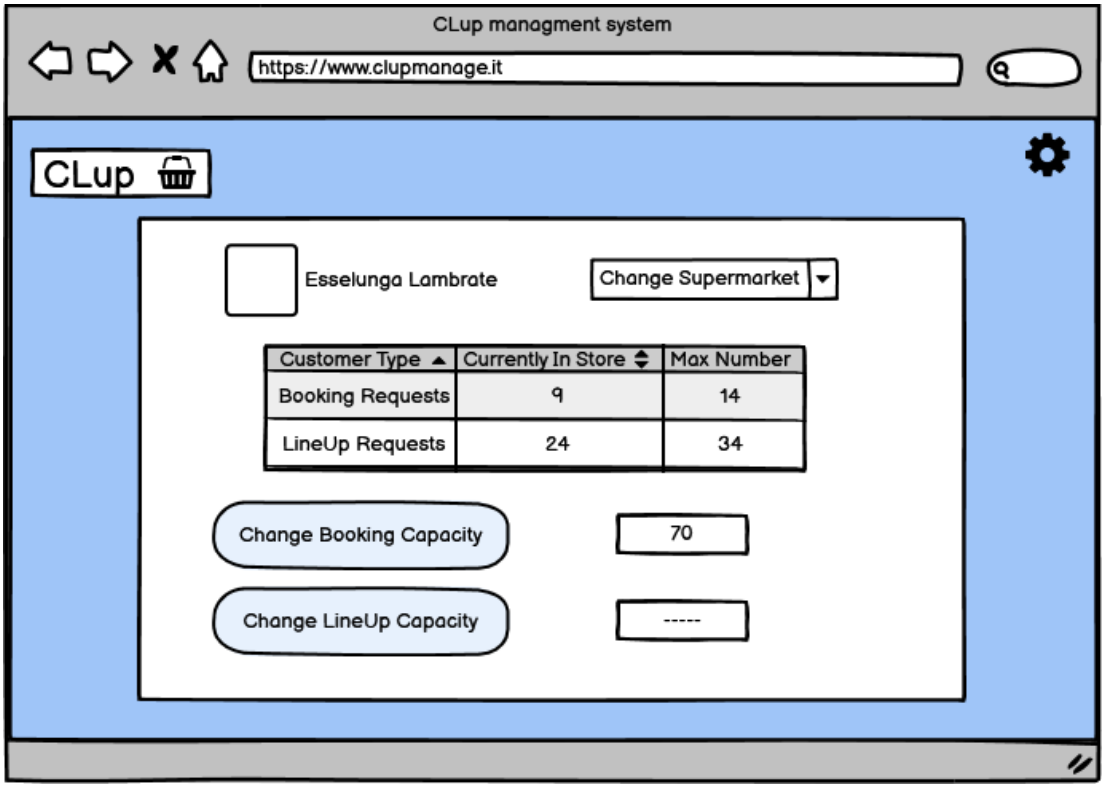

Figure 26: Home Page Manager

3.1.2 Hardware Interfaces

The interfaces needed by the system for its correct functioning are the ones related to the customer device, in particular the access to the notification system of the device in order to send the periodic notifications of the “Notification Function” (Section 2.2.2.2) and the access to the GPS system for the geo-localization of the device during the creation of a “Book a Visit” request or a “Line Up” request (respectively Sections 2.2.2 - 2.2.1) in order to obtain the correct estimated time needed to reach the store.

3.1.3 Software Interfaces

Software interfaces are related to the external services needed by the application for his functioning. A small analysis of what are and why are important is reported in Section 2.1.1 and fully described in the Design Document. The system needs to interact with different services and for this reason has different interfaces. The application does not provide API for other external applications.

3.1.4 Communication Interfaces

A fundamental aspect to take into account while the development of the system is the fact that the customer application and the central system needs to communicate in real time for the management of the different requests and the update of the user interface. So, we can expect from this specification a system capable of receiving multiple requests and managing all of them communicating in different ways for the different services to be guaranteed to the customer.

3.2 Functional Requirements

In this subsection we give a complete list of the functional requirements of our system and then we specify the mapping among **Goals, Requirements** and **Domain Assumption.**

3.2.1 Requirements

R1: The system must allow customers to register.

R2: The system must allow managers to register.

R3: The system must allow customers to log in.

R4: The system must allow managers to log in.

R5: The system must guarantee that each username is unique.

R6: The system must save customers registration data.

R7: The system must prevent users from using special characters in their username.

R8: The system must retrieve the GPS position while the user makes a reservation.

R9: The system must allow users to insert in which store they want to go.

R10: The system must allow users to insert a shopping list of items available in the store.

R11: The system must allow users to choose categories of products.

R12: The system must provide users with the QR code.

R13: The system must provide users with the ticket number.

R14: The system must allow users to insert visit date on the booking.

R15: The system must allow users to insert visit time on the booking.

R16: The system must be able to send notifications to the user.

R17: The system must allow customers to filter by city.

R18: The system must allow customers to filter by street.

R19: The system must estimate the waiting time before customers' turn.

R20: The system must infer the usual duration of the visit to the store of a long-term customer.

R21: The system must estimate the necessary time to arrive at the store.

R22: The system must notify users in advance about their turns.

R23: The system must allow users to indicate the approximate expected duration of the visit.

R24: The system must allow managers to know how many "line up customers" are inside.

R25: The system must allow managers to know how many "book a visit customers" are inside the shop.

R26: The system must allow managers to add a shop in the managed ones.

R27: The system must allow managers to remove a shop in the managed ones.

R28: The system must allow managers to change the maximum booking capacity of the managed shops.

R29: The system must allow managers to change the maximum line up capacity of the managed shops.

R30: The system must allow customers to be informed about the availability of different time slots.

R31: The system must allow customers to be informed about the availability of different supermarkets of the same chain.

R32: The system must allow customers to know if the store is in one of its closing days or not.

R33: The system must save managers registration data.

3.2.2 Goals Mapping

GOALS	REQUIREMENTS	DOMAIN ASSUMPTIONS
G1	R1, R3, R5, R6, R8, R9, R12, R13, R17, R18, R19, R21, R22	DA1, DA2, DA3, DA4, DA5, DA6, DA7, DA10, DA11, DA13, DA14, DA18, DA19, DA21, DA22, DA23
G2	R2, R4, R5, R7, R24, R25, R26, R27, R28, R29, R33	DA5, DA10, DA11, DA13, DA15, DA16, DA19, DA20, DA21, DA22, DA23
G3	R1, R3, R5, R6, R7, R8, R9, R10, R11, R12, R14, R15, R17, R18, R19, R20, R23, R32	DA1, DA2, DA4, DA5, DA6, DA7, DA8, DA9, DA10, DA11, DA13, DA15, DA16, DA19, DA20, DA21, DA22
G4A	R1, R3, R5, R6, R7, R8, R9, R14, R15, R30, R32	DA4, DA5, DA10, DA11, DA13
G4B	R1, R3, R5, R6, R7, R8, R9, R17, R18, R31, R32	DA1, DA4, DA5, DA7, DA10, DA13, DA21, DA22
G2C	R1, R3, R5, R6, R7, R9, R16, R17, R18, R32	DA1, DA4, DA5, DA10, DA13, DA17, DA21, DA22

Table 3: Goals mapping

G1: Users should be able to virtually line up through the app to enter the store.

R1 The system must allow customers to register.

R3 The system must allow customers to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save customers registration data.

R7 The system must prevent users from using special characters in their username.

R8 The system must retrieve the GPS position while the user makes a reservation.

R9 The system must allow users to insert in which store they want to go.

R12 The system must provide users with the QR code.

R13 The system must provide users with the ticket number.

R17 The system must allow customers to filter by city.

R18 The system must allow customers to filter by street.

R19 The system must estimate the waiting time before customers' turn.

R21 The system must estimate the necessary time to arrive at the store.

R22 The system must inform users in advance about their turns.

DA1 Customers always insert correct data while registering to CLup.

DA2 If a user makes a request he effectively goes to the supermarket in time.

DA3 Users must follow the shortest path to the shop.

DA4 The devices used have date and time working properly.

DA5 Internet connection works always without errors.

DA6 QR is always readable.

DA7 Maps of Italy are well known, complete and up to date.

DA10 Opening and closing hours of the shops are always known.

DA11 Two persons in the shop always keep distance of at least one meter.

DA13 The number of people permitted by law in the shops is known.

DA14 Estimated time from one place to another is always correct.

DA18 All shops have a ticket release machine at the entrance.

DA19 All shops have a ticket QR code reading machine (or something similar) at the entrance, in order to read the tickets.

DA21 The system knows all the positions of supermarkets.

DA22 The GPS module of the devices on which CLup runs always works correctly and has an accuracy of 2 meters.

DA23 All shops have a sensor (or something similar) that takes into account customers' exits.

G2: Store managers should be able to regulate the influx of people inside the building.

R2 The system must allow managers to register.

R4 The system must allow managers to log in.

R5 The system must guarantee that each username is unique.

R7 The system must prevent users from using special characters in their username.

R24 The system must allow managers to know how many "line up customers" are inside.

R25 The system must allow managers to know how many "book a visit customers" are inside the shop.

R26 The system must allow managers to add a shop in the managed ones.

R27 The system must allow managers to remove a shop in the managed ones.

R28 The system must allow managers to change the maximum booking capacity of the managed shops.

R29 The system must allow managers to change the maximum line up capacity of the managed shops.

R33 The system must save managers registration data.

DA5 Internet connection works always without errors.

DA10 Opening and closing hours of the shop are always known.

DA11 Two persons in the shop always keep distance of at least one meter.

DA12 All shops have a unique identifier.

DA13 The number of people permitted by law in the shop is known.

DA19 All shops have a ticket QR code reading machine (or something similar) at the entrance, in order to read the tickets.

DA23 All shops have a sensor (or something similar) that takes into account customers' exits.

G3: Users should be able to "book a visit" to the supermarket in order to go shopping.

R1 The system must allow customers to register.

R3 The system must allow customers to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save customers registration data.

R7 The system must prevent users from using special characters in their username.

R8 The system must retrieve the GPS position while the user makes a reservation.

R9 The system must allow users to insert in which store they want to go.

R10 The system must allow users to insert a shopping list of items available in the store.

R11 The system must allow users to choose categories of products.

R12 The system must provide users the QR code.

R14 The system must allow users to insert visit date on the booking.

R15 The system must allow users to insert visit time on the booking.

R17 The system must allow customers to filter by city.

R18 The system must allow customers to filter by street.

R20 The system must infer the usual duration of the visit on the store of a long-term customer.

R23 The system must allow users to indicate the approximate expected duration of the visit.

R32 The system must allow customers to know if the store is in one of its closing days or not.

DA1 Customers always insert correct data while registering to CLup.

DA2 If a user makes a reservation he effectively goes to the supermarket in time.

DA4 The devices used have date and time working properly.

DA5 Internet connection works always without errors.

DA6 QR is always readable.

DA7 Maps of Italy are well known, complete and up to date.

DA8 Product disposition inside the shop is always known by the system.

DA9 Users must buy what they previously have selected.

DA10 Opening and closing hours of the shops are always known.

DA11 Two persons in the shop always keep distance of at least one meter.

DA13 The number of people permitted by law in the shops is known.

DA15 Estimated time of a person in a shop is always what he has previously inserted.

DA16 Estimated time of a long-term customer is always correct.

DA19 All shops have a ticket QR code reading machine (or something similar) at the entrance, in order to read the tickets.

DA20 All items selectable in the application are always available in the shop.

DA21 The system knows all the positions of supermarkets.

DA22 The GPS module of the devices on which CLup runs always works correctly and has an accuracy of 2 meters.

DA23 All shops have a sensor (or something similar) that takes into account customers' exits.

G4: Users should be able to know different alternatives and suggestions.

G4A: Users should be able to know alternative time slots for visiting the store, during the booking, if the preferred one is not available.

R1 The system must allow customers to register.

R3 The system must allow customers to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save customers registration data.

R7 The system must prevent users from using special characters in their username.

R8 The system must retrieve the GPS position while the user makes a reservation.

R9 The system must allow users to insert in which store they want to go.

R14 The system must allow users to insert visit date on the booking.

R15 The system must allow users to insert visit time on the booking.

R30 The system must allow customers to be informed about the availability of different time slots.

R32 The system must allow customers to know if the store is in one of its closing days or not.

DA4 The devices used have date and time working properly.

DA5 Internet connection works always without errors.

DA10 Opening and closing hours of the shop are always known.

DA11 Two persons in the shop always keep distance of at least one meter.

DA13 The number of people permitted by law in the shop is known.

G4B: Users should be able to know different stores of the same chain if the preferred one is not available.

R1 The system must allow customers to register.

R3 The system must allow customers to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save customers registration data.

R7 The system must prevent users from using special characters in their username.

R8 The system must retrieve the GPS position while the user makes a reservation.

R9 The system must allow users to insert in which store they want to go.

R17 The system must allow customers to filter by city.

R18 The system must allow customers to filter by street.

R31 The system must allow customers to be informed about the availability of different supermarkets of the same chain.

R32 The system must allow customers to know if the store is in one of its closing days or not.

DA1 Customers always insert correct data while registering to CLup.

DA4 The devices used have date and time working properly.

DA5 Internet connection works always without errors.

DA7 Maps of Italy are well known, complete and up to date.

DA10 Opening and closing hours of the shops are always known.

DA13 The number of people permitted by law in the shops is known.

DA21 The system knows all the positions of supermarkets.

DA22 The GPS module of the devices on which CLup runs always works correctly and has an accuracy of 2 meters.

G4C: Users should be periodically informed about different free day/time slots for visiting the store.

R1 The system must allow customers to register.

R3 The system must allow customers to log in.

R5 The system must guarantee that each username is unique.

R6 The system must save customers registration data.

R7 The system must prevent users from using special characters in their username.

R9 The system must allow users to insert in which store they want to go.

R16 The system must be able to send notifications to the user.

R17 The system must allow customers to filter by city.

R18 The system must allow customers to filter by street.

R32 The system must allow customers to know if the store is in one of its closing days or not.

DA1 Customers always insert correct data while registering to CLup.

DA4 The devices used have date and time working properly.

DA5 Internet connection works always without errors.

DA10 Opening and closing hours of the shop are always known.

DA13 The number of people permitted by law in the shop is known.

DA17 All users have devices' notifications up and running.

DA21 The system knows all the positions of supermarkets.

DA22 The GPS module of the devices on which CLup runs always works correctly and has an accuracy of 2 meters.

3.3 Use Cases Identification

In this section we illustrate Scenarios, Use Case Diagram and Use Case Description. We have divided them based on whether they refer to customers or managers, in this way the functionalities each one refers to, are more readable. Each Use Case Description has its own Sequence Diagram which explicates a dynamic view of messages exchanged. At the of the section, there is the Traceability Matrix which helps to better understand correspondence between Use Cases and Requirements through proper identifiers.

3.3.1 Scenarios

Here we are going to identify some scenarios for the Customer and for the Manager. Scenarios are a concrete, focused, informal description of a single feature of the system to be with the aim of better understanding how CLup works and how it can be useful in some events of real life.

Customer

CS1: Franco is an engineer and he has a very busy life due to his work. He does not have enough time to spend at the supermarket to do shopping. In this context CLup booking feature is very useful because it allows him to plan shopping by choosing the list of items on the application. Moreover, he does not need to go around the store and then maybe discover that the item he is looking for is not there because the app only shows items that are available in the store. In this way CLup offers him the comfort of going to the shop, avoiding waiting to enter, going directly to the relevant departments of the store and finally he can exit with the minimum time spent. All this and moreover in a safe way to avoid contagion!

CS2: An 84-year-old elderly lady named Anna lives alone in Milan. She has two adult children who no longer live in her same house. Despite her age, she is an independent lady and does not need any help. Therefore, everything she does, she does it herself and without help, including going to the grocery store when she needs it but in this dangerous situation, due to the virus, is a risk. CLup also allows elderly people, who do not have access to technology, to deal with its features, indeed Anna can go to the store and get her physical ticket from the Ticket Machine at the entrance. In this way, she will be able to do shopping in a safer way without taking too many risks.

CS3: Marco is an off-site master's degree student at the Politecnico di Milano and during the week he has very uncomfortable lesson times from Monday to Friday, he also enrolled in an extracurricular course and keeps him busy even on Saturdays. The only free day of the week is Sunday and on which he can go to the shop, since in this situation, it is the only allowed way to relax and free his mind. From this point of view, he uses CLup which knows from data stored that Marco does shopping always on Sundays and every week it suggests the best time to go to the store on Sunday of that week in order to find it not too crowded.

CS4: Valeria is a girl who works as an assistant cook in a retirement home and needs to go out to buy pasta in a grocery store because it is running out. Even though she is a young girl, she cannot afford to expose herself to the virus because of the people she is in contact with. A friend advises her to register on CLup and use LineUp function, in such a way Valeria can wait her turn without going out and without being in direct contact with too many people.

CS5: Luca is a young wedding planner and his work reflects very well how he is in his life, in fact he is a perfectionist, an organized guy. He loves to plan his life as well as weddings. This week he had planned to go shopping Wednesday afternoon at 5pm, but strangely he forgot to book. So, he tries to book by using CLup, but unfortunately the store is full at that time. Through to the advanced functionality of CLup, he is informed about other near supermarkets. Therefore, thanks CLup he discovers a new near supermarket and in addition to that, he does not have to change slot time to do shopping, which would have been a very unpleasant event for all his future plans!

Manager:

MS1: Guido runs a Carrefour near Lambrate and he uses CLup to better manage his store. Due to the worsening of the Covid19 situation, The Prime Minister issues a new DPCM according to which Lombardy becomes a red zone, this implies new strong restrictions in the region. In response to this, Guido decides to safeguard people who go to his shop by decreasing the number of allowed people in the building. He can do this through the CLup Web Interface.

3.3.2 Use Case Diagram

The following picture (Fig. 27) represents a Use Case Diagram that shows how actors are involved in Use Cases and how Use Cases are linked to each other. In addition to that, the picture highlights all the functionalities provided by the system.

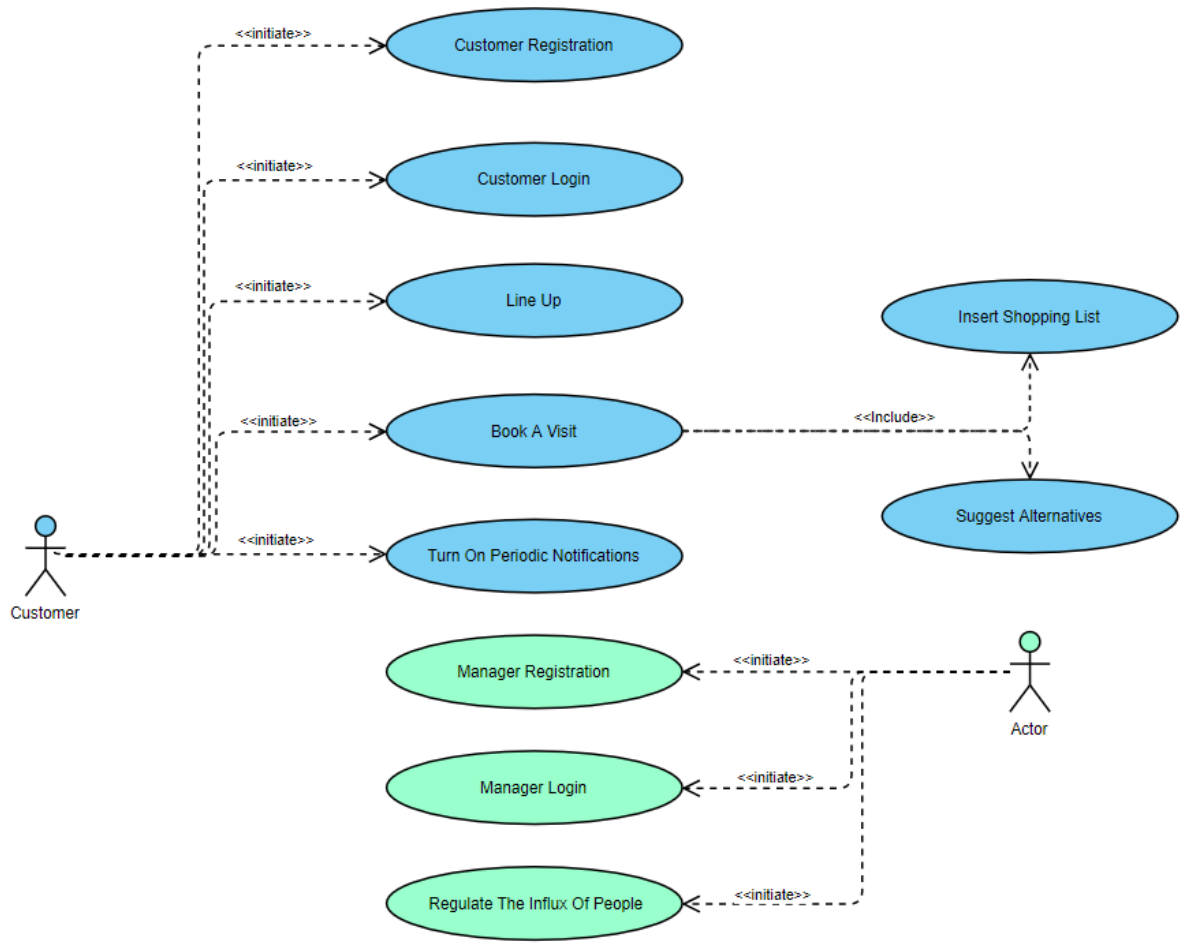

Figure 27: Use Case Diagram

3.3.3 Use Cases Description

Customer

UCD1 Customer Registration:

Name	Customer Registration
Actors	Customer
Entry Condition	The customer has started CLup on his device
Flow of events	(1) The customer clicks on sign up button (2) The customer chooses a username and a password (3) The customer confirms his password (4) The customer optionally inserts his email (5) The customer accepts the Terms and Conditions of CLup (6) The customer clicks on confirm button (7) The system checks the username to be unique (8) The system checks the email (9) The system stores the customer data
Exit Condition	The customer is registered in the system
Exception	• If the confirmed password is not the same as the first one, the system informs the customer that the two passwords are different. • If the username inserted by the user is already used by another user, or if the username contains any special character, the system displays an error message asking the user to insert a different one. • If the email is inserted and the system finds a duplicate it rejects the request.

Table 4: *Customer Registration* Use Case Description

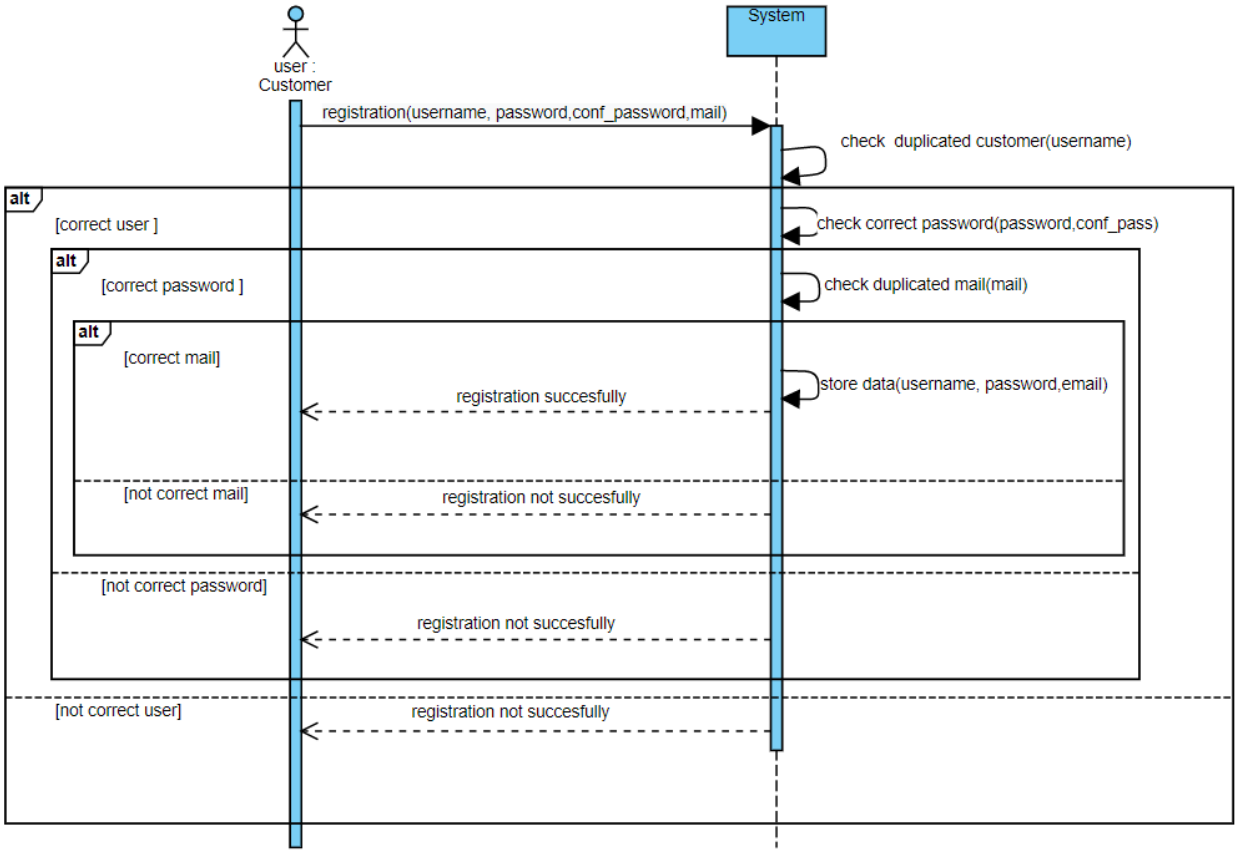

Figure 28: *Customer Registration* Sequence Diagram

UCD2 Customer Login:

Name	Customer Login
Actors	Customer
Entry Condition	The customer has started CLup on his device
Flow of events	(1) The customer inserts his username (2) The customer inserts his password (3) The customer optionally clicks on ‘Remember me’ (4) The customer clicks on ‘Login’ (5) The system checks the username to be existing (6) The system checks the password to be correct
Exit Condition	The customer is logged in
Exceptions	● If the system does not recognize the username inserted, the customer is not registered yet, or the username is incorrect. The system informs the customer about it. ● If the password is incorrect, the system informs the customer to reinsert it. If he wrongs again, the system suggests the customer to reset a new password by sending him an email.

Table 5: *Customer Login* Use Case Description

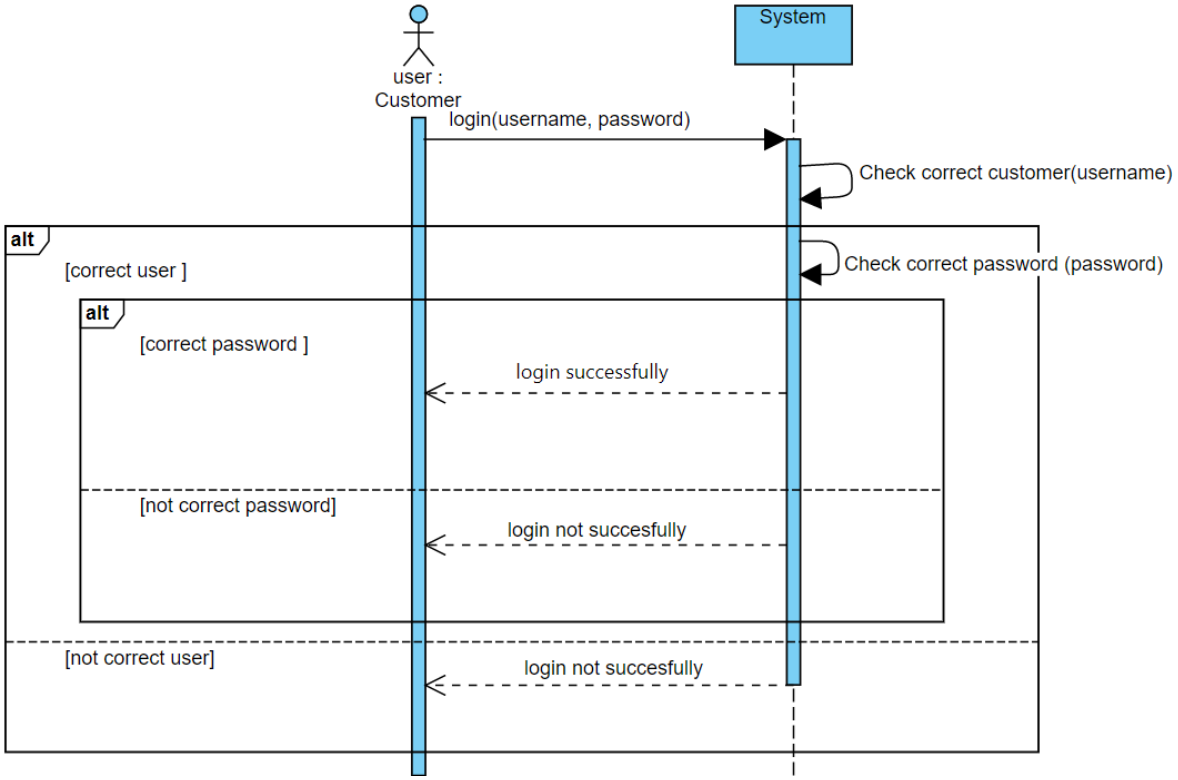

Figure 29: *Customer Login* Sequence Diagram

UCD3 Line Up:

Name	Line Up
Actors	Customer
Entry Condition	The customer is logged in
Flow of events	(1) The customer clicks on 'Line Up!' (2) The system retrieves GPS position of the customer device (3) The customer selects supermarket from a list (4) The system checks duplicate requests (5) The system stores the request (6) The system estimates time to reach the store (7) The system estimates waiting time (8) The system generates QR code (9) The system generates ticket number (10) The system returns the ticket
Exit Condition	QR code ticket for line up is shown to the customer
Exceptions	• If the system finds more than one duplicate request, it rejects the request.

Table 6: *Line Up Function* Use Case Description

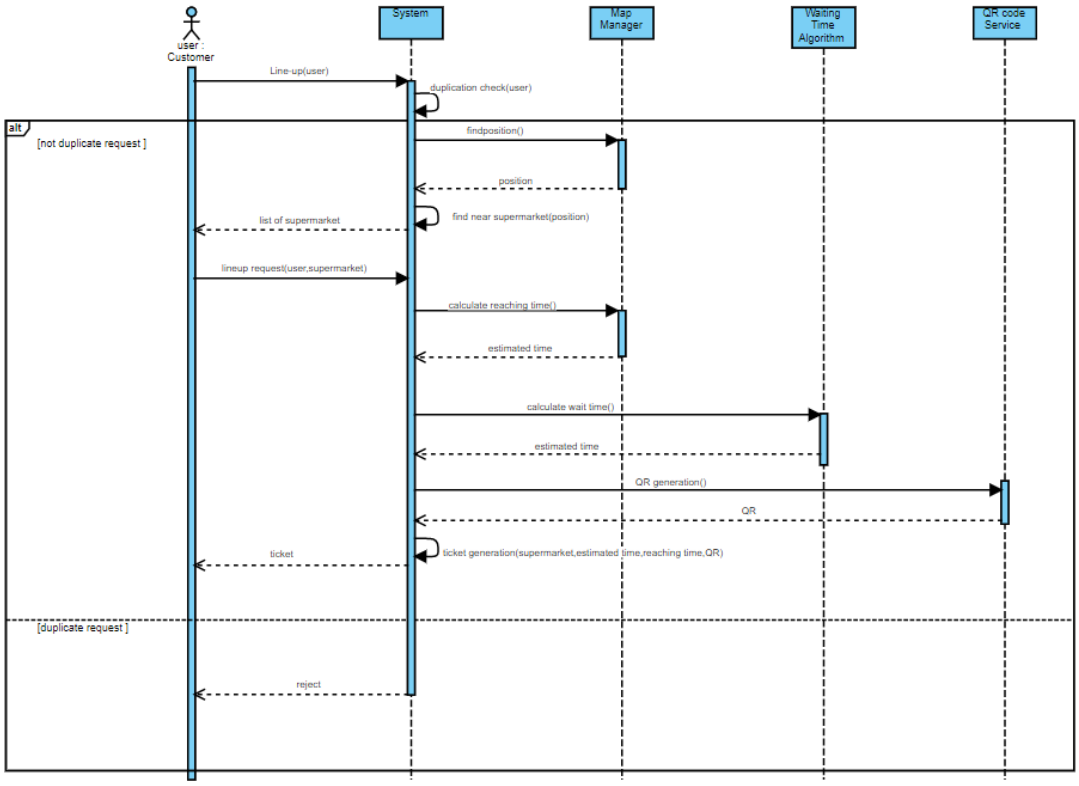

Figure 30: *Line Up Function* Sequence Diagram

UCD4 Book A Visit:

Name	Book a visit without shopping list
Actors	Customer
Entry Condition	The customer is logged in
Flow of events	(1) The customer clicks on ‘Book A Visit!’ (2) The customer selects supermarket from a list (3) The customer inserts booking date (4) The customer inserts booking time (5) The customer optionally inserts the expected duration of his visit (6) The customer clicks on ‘Create your ticket' (7) The system checks duplicate requests (8) The system checks the insertion of the duration (9) The system checks if the store at that time slot is full (10) The system stores the request
Exit Condition	The customer has booked a visit without inserting the shopping list and QR code ticket for booking is shown to the customer
Exceptions	● If the system finds more than one duplicate request, it rejects the request. ● If the system does not find the duration, it checks if the customer is a long customer: if he is then it infers itself the duration, otherwise it rejects the request. ● If the store is full, the system gives the possibility both to insert the shopping list to check if he can go anyway according to the crowding of the different compartments of the store, or to change supermarket/time slot.

Table 7: *Book A Visit Function* Use Case Description

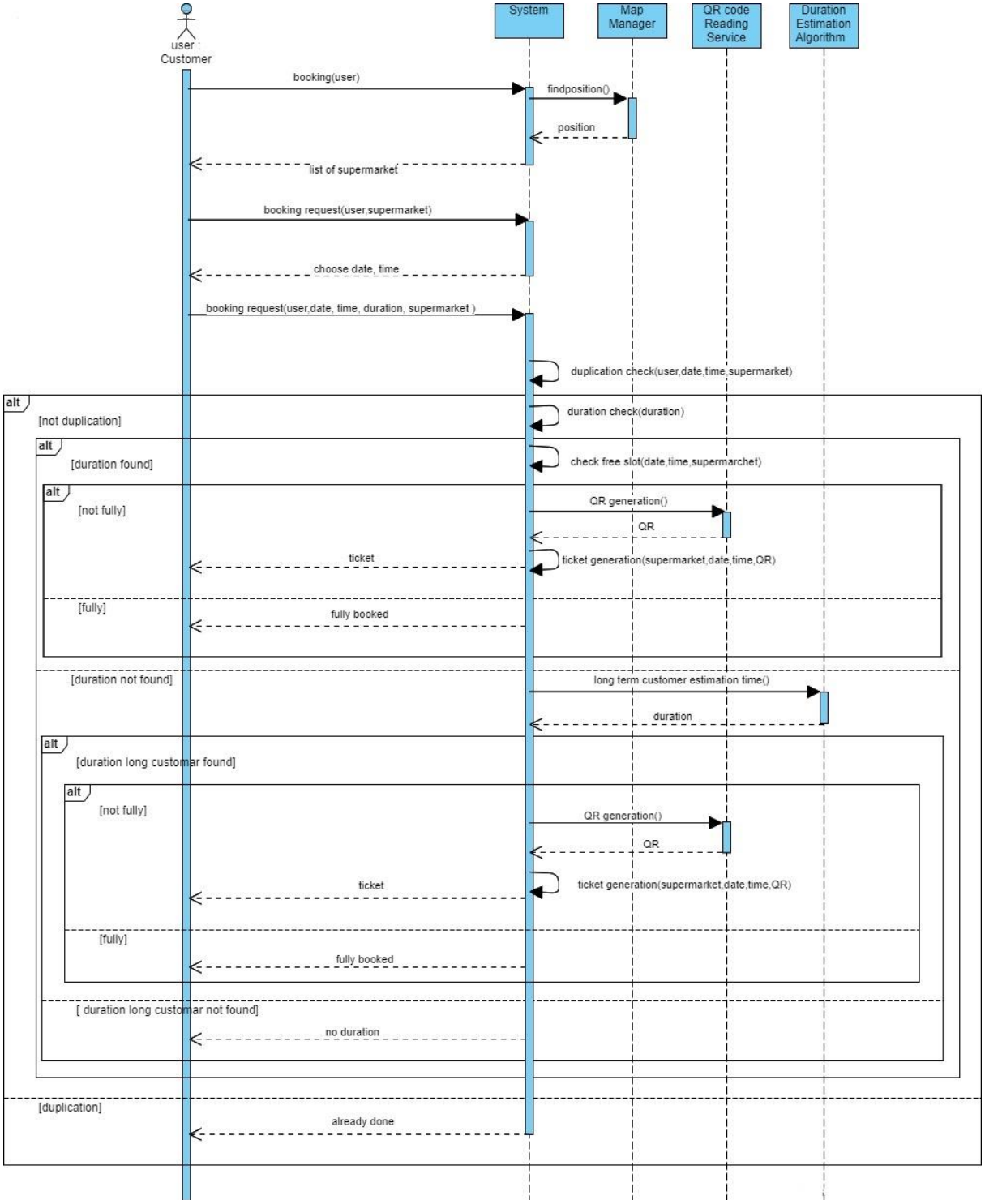

Figure 31: *Book A Visit Function* Sequence Diagram

UCD5 Insert Shopping List:

Name	Insert shopping List
Actors	Customer
Entry Conditions	(1) The customer is logged in (2) The customer has just inserted booking date and time
Flow of events	(1) The customer clicks on 'Create a shopping list!' (2) The customer chooses some categories (3) The customer clicks on 'Create accurate shopping list!' (4) The customer chooses some products (5) The system stores data inserted
Exit Condition	The customer has inserted shopping list in his booking
Exceptions	-

Table 8: *Insertion of the Shopping List* Use Case Description

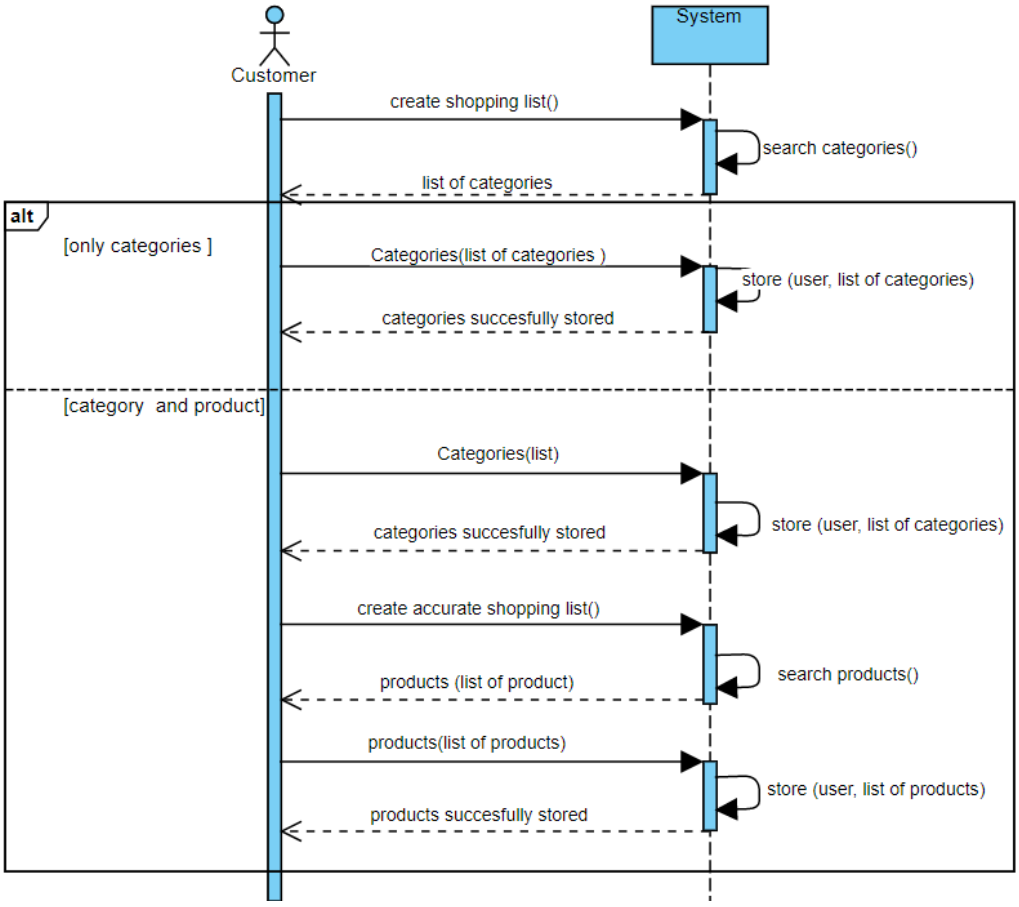

Figure 32: *Insertion of the Shopping List* Sequence Diagram

UCD6 Suggest Alternatives:

Name	Suggest alternatives
Actors	Customer
Entry Conditions	(1) The customer is logged in (2) The customer has sent Book a visit request without shopping list or with shopping list but the check crowding was not successful (3) The store is fully booked
Flow of events	(1) The customer receives Warning Pop Up (2) The customer clicks on 'Time Slots' (3) The system finds available slots (4) The system shows alternative time slots (5) The customer chooses an available time slot (6) The system checks alternative received (7) The system stores data
Exit Condition	The customer has chosen his preferred alternative
Exceptions	• If the check alternative is not successful, this means that the suggested time slots are not what the customer wants and the system suggests other supermarkets available, if the customer chooses another supermarket the system accepts the request otherwise it rejects.

Table 9: *Alternatives Suggestion* Use Case description

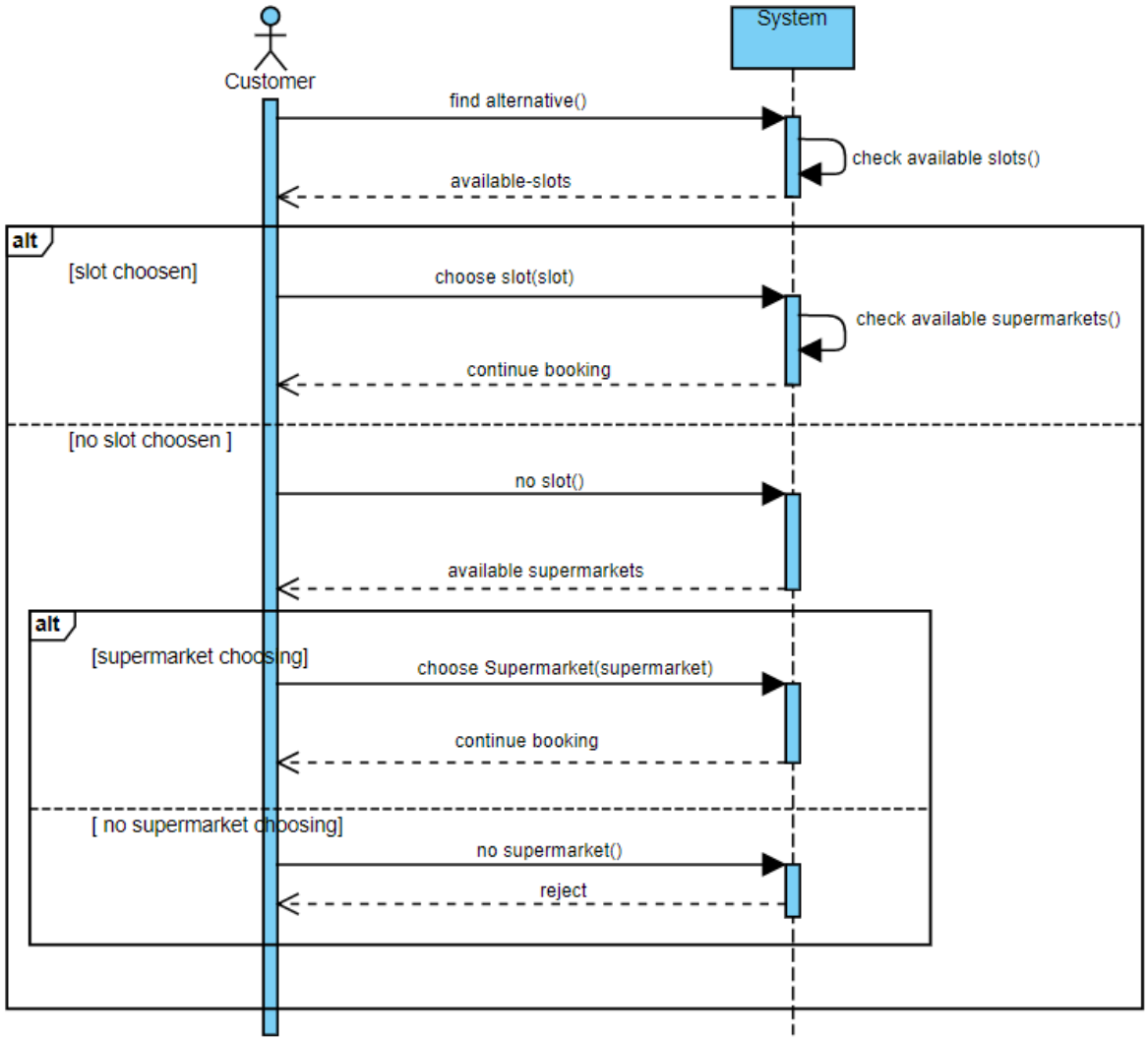

Figure 33: *Alternatives Suggestion* Sequence Diagram

UCD7 Turn On Periodic Notifications:

Name	Turn on periodic notifications
Actors	Customer
Entry Condition	The customer is logged in
Flow of events	(1) The customer clicks on 'Turn on notifications' (2) The system stores the permission
Exit Condition	The customer has turned on periodic notifications on his device
Exceptions	-

Table 10: *Turn On Periodic Notifications Function* Use Case Description

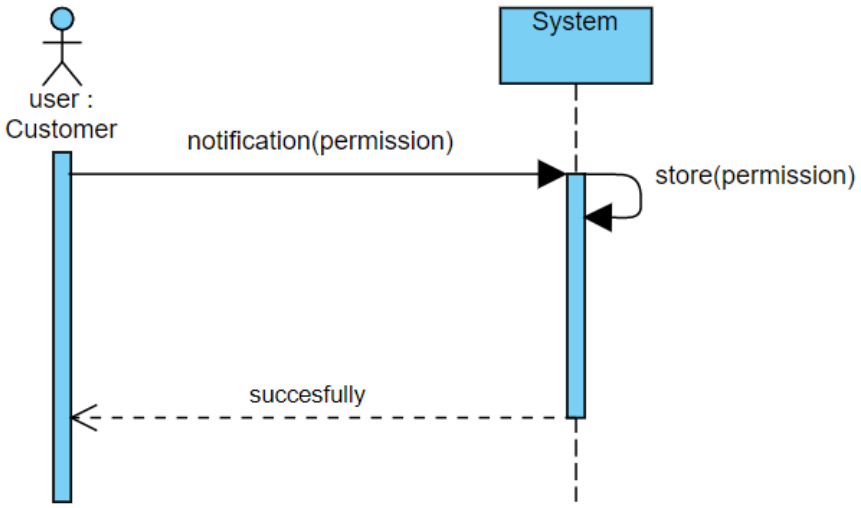

Figure 34: *Turn On Periodic Notifications Function* Sequence Diagram

Manager

UCD8 Manager Registration:

Name	Manager Registration
Actors	Manager
Entry Condition	Manager has started the CLup Web application
Flow of events	(1) The manager clicks on sign up button (2) The manager chooses a username and a password (3) The manager confirms his password (4) The manager inserts his email (5) The manager inserts his supermarket identifier (6) The manager accepts the Terms and Conditions of CLup (7) The manager clicks on 'Register' button (8) The system checks the username to be unique (9) The system checks the email (10) The system stores the manager data
Exit Condition	The manager is registered in the system
Exceptions	• If the confirmed password is not the same as the first one, the system informs the manager that the two passwords are different. • If the manager does not have the supermarket identifier or the one inserted is incorrect, he must click on the link suggested. • If the username inserted by the manager is already used by another user, or if the username contains any special character, the system informs the manager with a message error asking the manager to insert a different one. • If the system finds a duplicate it rejects the request.

Table 11: *Manager Registration* Use Case Description

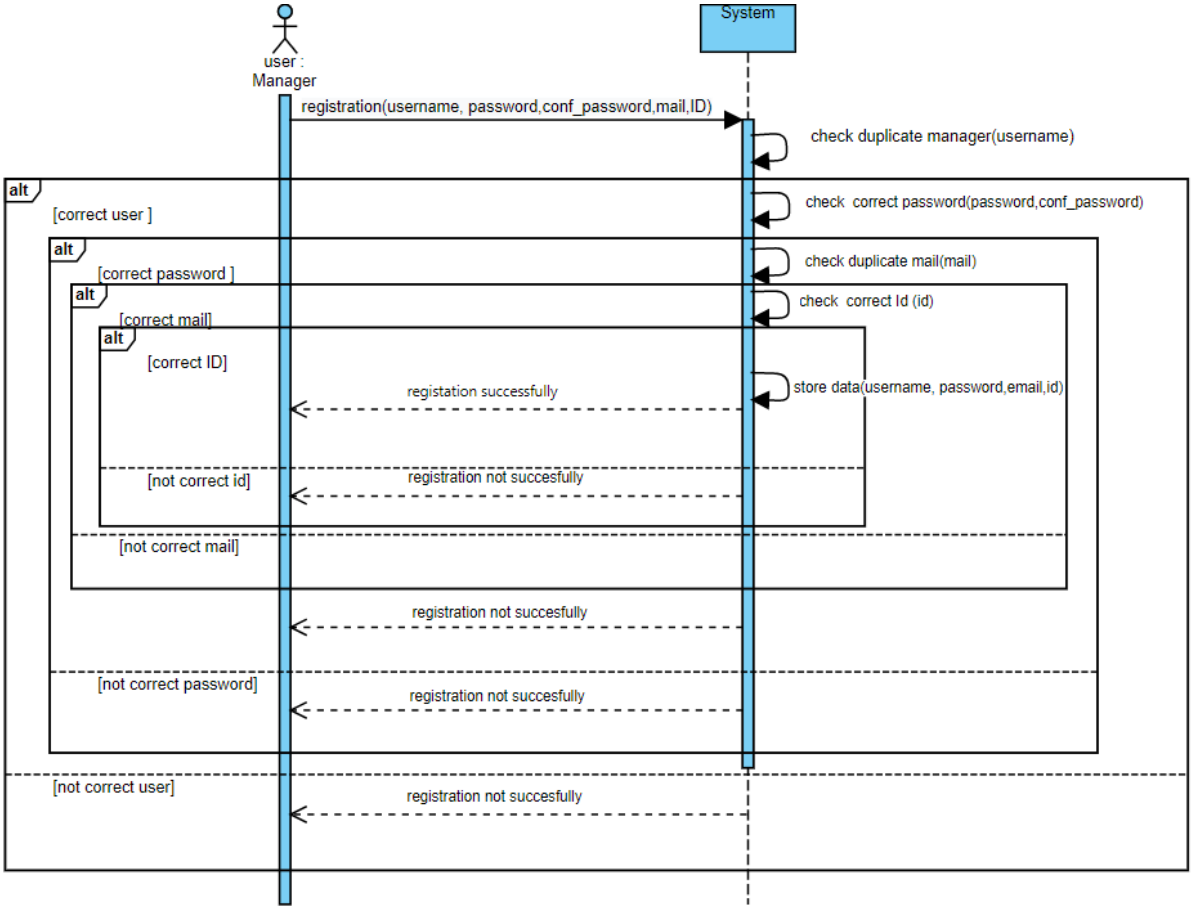

Figure 35: *Manager Registration* Sequence Diagram

UCD9 Manager Login:

Name	Manager Login
Actors	Manager
Entry Condition	The manager has started the CLup web application
Flow of events	(1) The manager inserts his username (2) The manager inserts his password (3) The manager optionally clicks on ‘Remember me’ (4) The system checks the username to be existing (5) The system checks the password to be correct (6) The manager clicks on ‘Send me the OTP code’ (7) The system generates the OTP code (8) The system sends the OTP code to the manager via email (9) The manager inserts the OTP code (10) The manager clicks on ‘Confirm and Login’
Exit Condition	The manager is logged in
Exceptions	● If the system does not recognize the username inserted, the manager is not registered yet, or the username is incorrect. The system informs the manager about it. ● If the password is incorrect, the system informs the manager to reinsert it. If he wrongs again, the system suggests the manager to reset a new password by sending him an email. ● If the manager does not receive the OTP or if he inserts a wrong OTP, the procedure is aborted.

Table 12: *Manager Login* Use Case Description

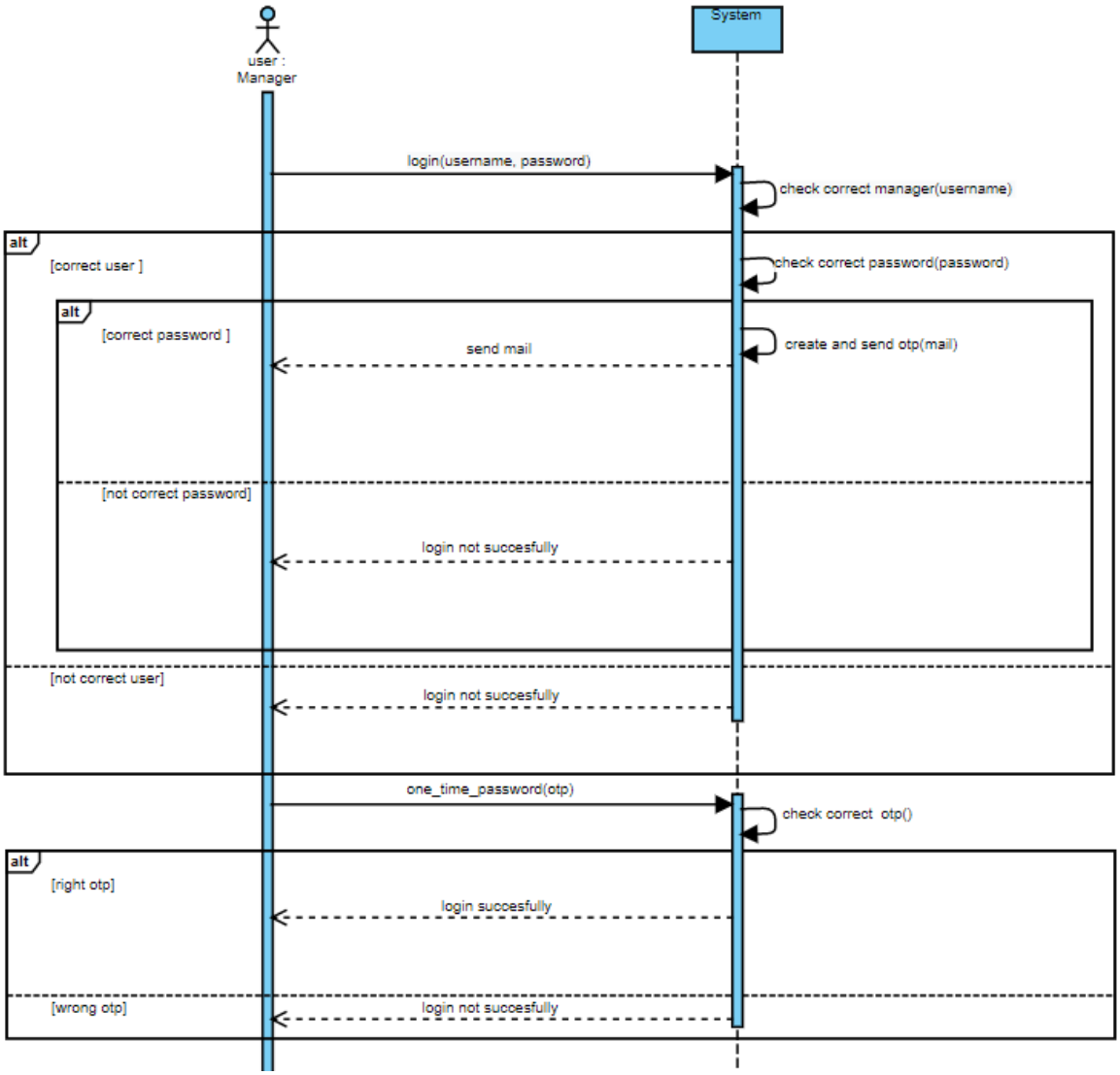

Figure 36: *Manager Login* Sequence Diagram

UCD10 Regulate The Influx Of People:

Name	Regulate influx of people
Actors	Manager
Entry Condition	The manager is logged in
Flow of events	(1) The manager increases/decreases the maximum number of booking/line up requests (2) The system stores the new capacity (3) The system sends info to the QR Scanner
Exit Condition	The capacity is changed
Exceptions	-

Table 13: *Regulate The Influx Of People Function* Use Case Description

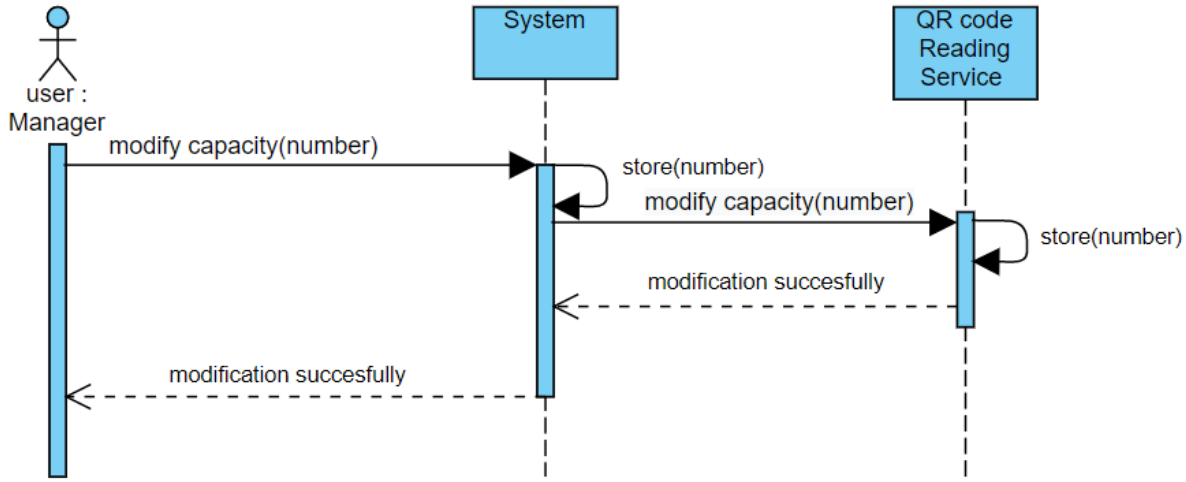

Figure 37: *Regulate The Influx Of People Function* Sequence Diagram

3.3.4 Traceability Matrix

	R1	R2	R3	R4	R5	R6	R7	R8
CS1	X		X		X	X	X	X
CS2								
CS3	X		X		X	X	X	
CS4	X		X		X	X	X	X
CS5	X		X		X	X	X	X
MS1		X		X	X		X	
UCD1	X				X	X	X	
UCD2			X		X	X	X	
UCD3	X		X		X	X	X	X
UCD4	X		X		X	X	X	X
UCD5								X
UCD6								X
UCD7	X		X		X	X	X	
UCD8		X			X		X	
UCD9				X	X		X	
UCD10		X		X	X		X	

Table 14: Traceability Matrix for requirements from R1 to R8

	R9	R10	R11	R12	R13	R14	R15	R16
CS1	X	X	X	X		X	X	
CS2				X	X			
CS3	X							X
CS4	X			X	X			
CS5	X					X	X	
MS1								
UCD1								
UCD2								
UCD3	X			X	X			
UCD4	X	X	X	X		X	X	
UCD5	X	X	X					
UCD6	X					X	X	
UCD7	X							X
UCD8								
UCD9								
UCD10								

Table 15: Traceability Matrix for requirements from R9 to R16

	R17	R18	R19	R20	R21	R22	R23	R24
CS1	X	X		X			X	
CS2			X			X		
CS3	X	X						
CS4	X	X	X		X	X		
CS5	X	X						
MS1								X
UCD1								
UCD2								
UCD3	X	X	X		X	X		
UCD4	X	X		X			X	
UCD5	X	X						
UCD6	X	X						
UCD7	X	X						
UCD8								
UCD9								
UCD10								X

Table 16: Traceability Matrix for requirements from R17 to R24

	R25	R26	R27	R28	R29	R30	R31	R32	R33
CS1								X	
CS2									
CS3								X	
CS4									
CS5						X	X	X	
MS1	X			X	X				X
UCD1									
UCD2									
UCD3									
UCD4								X	
UCD5								X	
UCD6						X	X	X	
UCD7								X	
UCD8									X
UCD9									X
UCD10	X	X	X	X	X				X

Table 17: Traceability Matrix for requirements from R25 to R33

3.4 Performance Requirements

CLup is an application that allows people to do shopping in a safer way in a problematic situation. It must be very efficient because a minimum error can cause a risky situation for the people who are using it and it can throw them into a situation of potential contagion. It must manage very carefully entrances and exits from a store because we want to avoid that people that have a ticket and go to the store at the hour indicated on the ticket and must wait at the entrance creating undesired crowds. In this case we have assumed that people who are going to the store follow the shortest path to reach the store, because the system cannot control an eventual delay. So, CLup guarantees that no crowds are created because it estimates a reasonably precise waiting time and it alerts people when their turns are approaching. Therefore, when a client makes a request to the system, it needs to store and respond as soon as possible. At the same time, it has also to handle people who are making a request in the same moment. We can reasonably say that CLup processes requests within 4 seconds. It also needs to be an effective app in fact it has to reach also people who cannot afford such a technology like older people by using Ticket Machine, and it has to be an easy-to-use app simply downloadable on all of devices but also for manager interfaces.

3.5 Design Constraints

3.5.1 Standard Compliance

The language through which CLup system communicates with the other components is a standard language. The interaction with the storage component is standard, it is not implemented, it is as any other standard interaction between a software and a component from which to retrieve data. Moreover, the QR ticket release has all the characteristics of a standard QR code and it needs to be readable by the *QR Reading Service*.

3.5.2 Hardware Limitations

People who can use CLup and have access to Book A Visit function or Periodic Notifications function (also Line Up function but having a device it is not mandatory because Ticket Machine at the entrance is available as previously said) must have a device on which the application needs to be installed. Moreover, it is important to remark that devices must have GPS active, to retrieve their positions so that CLup can suggest the nearest supermarkets, but also devices able to receive notifications useful for Periodic Notification function.

3.5.3 Any Other Constraint

Any other constraint that is relevant to highlight is the *data security policy* that CLup must respect, users' data must be stored because the system must know how many registered users it has but all of this data can't be visible to other users, therefore it must comply with the *privacy policy*.

3.6 Software System Attributes

3.6.1 Reliability

CLup should guarantee a high probability of absence of failures and for a certain time period. Components used are such that they can work properly without interruptions for a non-trivial time. We assume a low MTTR (Mean Time To Repair) for a further increase of the Reliability. When the system goes down CLup must inform users in advance about the issue that has just arisen and what functionalities are not available, in the same way it informs them when the system returns to work correctly.

3.6.2 Availability

CLup shall be continuously available to the user. It must guarantee rapid service recovery in order to minimize downtime (MTTR). It is relevant to say that CLup needs a better availability during the opening hours of the stores, and therefore with a higher probability during the day, whereas for supermarkets open 24 hours a day even at night but reasonably the requests received are much lower (in an extremely dangerous situation they may be even 0 because stores are not allowed to be open until a certain time). So, we can expect an availability of 95% during the night and an availability of 98% during the day.

3.6.3 Security

CLup must have a high level of security even if it does not require a big amount of data to the user. It must protect user's data and must not allow other users to have access to different users data. The login and registration procedures of users must always be successful and work without any unexpected events, if a user needs to reset his password CLup always sends encrypted data via mail. For a higher level of security CLup uses OTP to recognize and authenticate managers.

3.6.4 Maintainability

CLup has to be updated within a certain time with bugs and errors correction. In this way the implementation code needs to be well structured and entirely commented to promote the readability of the code. Moreover, it is written with the highest level of abstraction in such a way future updates or changes will not cause a large change in the implementation. The test coverage must include most of the code (i.e., not less than 85%).

3.6.5 Portability

CLup is an easy-to-use application simply downloadable on all stores and therefore available for all operating systems (iOS, Android, …) in such a way there are no restrictions on device types. Instead for what concerns manager it must be available for all operating systems (Windows, MacOS, …). Therefore, since it is an application portability of CLup should not be a problem.

4 Formal Analysis Using Alloy

The following section is dedicated to the analysis of the essential constraints and properties of the CLup software. The model represented is simplified and written explanations are added in the parts where it is not possible to reach an adequate clarification with only the written code.

The same is done in section where it is important the external result computed by external services that are obviously not analysed in this document in detail.

Inside this model we want to prove that:

- There are no duplicated entities of any type.
- Only customers can make requests to the system.
- The notifications are provided by the system only if the customer authorizes them.
- All the reservations made by the application are in available dates and times.
- All tickets are correctly assigned to the customer that makes the request.
- All valid requests are ACCEPTED by the system and the invalid ones are REJECTED.
- And many others less important ones.

4.1 Alloy Model

```
1  ////////////////////////////////////////////////////////////////////////////////////////
2  ////////////////////////////////////// SIGNATURES //////////////////////////////////////
3  ////////////////////////////////////////////////////////////////////////////////////////
4
5  abstract sig Account {
6          username: one Username,
7          password: one Password,
8          email: lone Email
9  }
10
11 sig Customer extends Account {
12         position: one Position,
13         requests: set Request,
14         notification_permission: one Boolean
15 }
16
17 sig Manager extends Account {
18         supermarkets: some Supermarket
19 } { #email=1}
20
```

```
21  sig Supermarket {
22          position: one Position,
23          supermarket_chain: lone SupermarketChain,
24
25          line_capacity: one Int,
26          reservation_capacity: one Int,
27
28          opening_hour: one Time,
29          closing_hour: one Time,
30          closing_days: set Date,
31
32          available_items: set Product,
33          lineup_requests: set LineUpRequest,
34          bookavisit_requests: set BookAVisitRequest
35  }
36  {line_capacity>=0 and reservation_capacity>=0}
37
38  sig SupermarketChain {
39          supermarkets: set Supermarket
40  }
41
42  sig TicketMachine {
43          supermarket : one Supermarket,
44          ticketAccount: one Customer //predefined customer of the machine
45  }
46
47  abstract sig Request {
48          customer: one Customer,
49          supermarket: one Supermarket,
50          QRcode: one QRCode,
51          status: one Status
52  }
53
54  sig LineUpRequest extends Request {
55          entry_number: one Int,
56          estimated_waiting_time: one Duration,
57          estimated_path_time: one Duration
58  }
59  {entry_number >= 0}
60
61  sig BookAVisitRequest extends Request {
62          category_list: set Category,
63          shopping_list: set Product,
64          date: one Date,
65          start_time: one Time,
66          duration: lone Duration
67  }
68
69  sig Notification {
```

```
70          receiver: one Customer,
71          supermarket: one Supermarket,
72          available_date: one Date,
73          available_time: one Time
74  }
75
76  sig Product {
77          name: one ProductName}
78
79  sig ProductName {}
80
81  sig Category {
82          products: set Product}
83
84  sig Username {}
85
86  sig Password {}
87
88  sig Email {}
89
90  sig QRCode {}
91
92  sig Position {
93          street: one Street} //position provided by external interface, linked to the
94                                  // nearest street
95
96  sig Street{
97          name: one StreetName,
98          positions: some Position,
99          city: one City}
100
101 sig StreetName {}
102
103 sig City {
104         name: one CityName, //for filtering the available supermarkets
105         streets: some Street} //can not be empty, all cities with at least one street
106
107 sig CityName {}
108
109 abstract sig StructuredTime {
110         hour: one Int,
111         minute: one Int,
112         second: one Int}
113 {       hour>=0                 //&& hour <= 23 and
114         minute >=0              //&& minute <=59 and
115         second >=0              //&& second <=59
116 }
117
118 sig Duration extends StructuredTime {}
```

```
119
120 sig Time extends StructuredTime {}
121
122 sig Date {}
123
124 abstract sig Status {}
125
126 one sig ACCEPTED extends Status {}
127 one sig REJECTED extends Status {}
128
129 abstract sig Boolean{}
130
131 one sig TRUE extends Boolean{}
132 one sig FALSE extends Boolean {}
133
134 ////////////////////////////////////////////////////////////////////////////////////////
135 ////////////////////////////////////// FACTS ///////////////////////////////////////////
136 ////////////////////////////////////////////////////////////////////////////////////////
137
138 //CUSTOMER
139
140 //there aren't usernames without accounts
141 fact NoUsernameWithoutAccount {
142         all u: Username |
143                 some a: Account |
144                         u in a.username
145  }
146
147 //there aren't password without accounts
148  fact NoPasswordWithoutAccount {
149         all p: Password |
150                 some a: Account |
151                         p in a. password
152  }
153
154 //there are no email without account
155  fact NoEmailWithoutAccount {
156         all e: Email|
157                 some a: Account |
158                         e in a.email
159  }
160
161 //all usernames are unique
162  fact UniqueUsernames {
163         no disj a1, a2: Account |
164                 a1.username = a2.username
165  }
166
167 //all email are unique
```

```
168 fact UniqueEmails {
169         no disj a1, a2: Account |
170                 a1.email = a2.email
171 }
172
173
174 //SUPERMARKET
175
176 //there are not two supermarket in the same spot
177 fact NoTwoStoresInSamePosition {
178         no disj s1, s2 : Supermarket |
179                 s1.position = s2.position
180 }
181
182 //the supermarket closes after the opening, at least one hour open
183 fact ClosingAfterOpening {
184         all s : Supermarket | s.closing_hour.hour > s.opening_hour.hour or
185                                         //24h open supermarket
186                                         (s.closing_hour = s.opening_hour)
187 }
188
189 //a supermarket belongs to only one chain
190 fact NoSupermarketsInTwoChains {
191         no disj sc1, sc2: SupermarketChain |
192                 #(sc1.supermarkets & sc2.supermarkets)>0
193 }
194
195 //all the supermarkets have one Manager
196 fact NoSupermarketsWithoutManager {
197         all s : Supermarket |
198                 some m : Manager | s in m.supermarkets
199 }
200
201 //if a supermarket is in a chain, the chain knows it
202 fact ChainSupermarketConsistency {
203         all s : Supermarket |
204                 some c : SupermarketChain |
205                         s in c.supermarkets iff s.supermarket_chain = c
206 }
207
208 //if the supermarket is all-day open, the opening and closing hour are at 0:0:0
209 fact ContinuedWorkingHourMidnight{
210         all s: Supermarket |
211                 (s.opening_hour=s.closing_hour) implies
212                (s.opening_hour.hour=0
213                and s.opening_hour.minute=0 and s.opening_hour.second=0 and
214                 s.closing_hour.second=0 and s.closing_hour.minute=0 and
215                s.closing_hour.second=0)
216 }
```

```
217
218
219 // STREETS AND CITIES
220
221 //there are no streets without cities
222 fact NoStreetWithoutCity {
223         all s: Street |
224                 one c: City |
225                         s in c. streets
226 }
227
228 //if a street is in a city, the city has the street
229 fact StreetCityConnected {
230         all s: Street , c: City |
231                 s in c. streets iff c in s. city
232  }
233
234 //there are not two streets with the same name in a city
235 fact NoStreetsOmonimousInCity {
236         all s1 , s2: Street |
237                 (( some c: City | s1 in c. streets and s2 in c. streets ) and
238                 not (s1=s2)) implies not(s1. name = s2. name)
239 }
240
241 //there are no street names without a street
242 fact NoStreetNameWithoutStreet {
243         all sn: StreetName |
244                 some s: Street |
245                         sn in s.name
246 }
247
248 //there are no city names without a city
249 fact NoCityNameWithoutCity {
250         all cn: CityName |
251                 some c: City |
252                         cn in c. name
253 }
254
255 //one position is only in one street
256 fact PositionOnlyInOneStreet {
257         no disj s1 , s2: Street |
258                 #(s1. positions & s2. positions) >0
259 }
260
261 //all the positions are in one street
262 fact NoPositionWithoutStreet {
263         all p: Position |
264                 some s: Street |
265                         p in s. positions
```

```
266 }
267
268 //if a position is in one street, the position knows the street
269 fact PositionStreetConnected {
270         all s: Street , p: Position |
271                 p in s. positions implies s in p. street
272 }
273
274
275 //PRODUCTS AND CATEGORIES
276
277 //all products are in one category
278 fact NoProductInTwoCategories {
279         no disj c1, c2 : Category |
280                 #(c1.products & c2.products)>0
281 }
282
283 //there are no products with duplicated name
284 fact NoProductNameDuplicated {
285         no disj p1, p2 : Product |
286                 p1.name=p2.name
287 }
288
289 //there are no product names without the products
290 fact NoProductNameWithoutProduct{
291          all pn: ProductName |
292                 some p: Product |
293                         pn in p.name
294 }
295
296 //all products are in one category
297 fact NoProductInNoCategories{
298         all p: Product |
299                 some c : Category |
300                         p in c.products
301 }
302
303
304 //REQUESTS
305
306 //all QR codes are different for different requests
307 fact NoDuplicateQR {
308         no disj r1, r2 :Request |
309                 r1.QRcode = r2.QRcode
310 }
311
312 //No QR without a request
313 fact NoQRwithoutTicket {
314         all q : QRCode |
```

```
315                     some r : Request |
316                             r.QRcode=q
317     }
318
319
320     //no requests without customer
321     fact NoRequestsWithoutCustomer {
322             all r : Request |
323                     some c : Customer |
324                             r in c.requests and r.customer=c
325     }
326
327     //no requests required by different customers
328     fact RequestOnlyOneCustomer {
329             no disj c1, c2 : Customer |
330                     #(c1.requests & c2.requests)> 0
331     }
332
333     //correct classification of request in "LineUpRequest" and "BookAVisitRequests"
334     fact RequestInCorrectSet {
335             all r1: Request |
336                      (r1 in r1.supermarket.lineup_requests and r1 in LineUpRequest) or
337                     (r1 in r1.supermarket.bookavisit_requests and r1 in
338                    BookAVisitRequest) or r1.status=REJECTED
339     }
340
341     //LINEUP REQUEST
342     //we assume that the customer makes the request in days when the supermarket is open
343     //it's possible to add a check using the retrieved data and time from the device using
344     //similar checks like the ones in the next section.
345     //The same is possible also for the checking of the hours when the supermarket is open in
346     //order to avoid the overstep of the closing hour with the estimated waiting time.
347     //We have made this assumption for a better readability in the alloy model.
348
349
350
351     //all entry numbers are different in the same supermarket
352     fact DifferentEntryNumber {
353             all disj r1, r2: LineUpRequest |
354                     (r1.supermarket = r2.supermarket) implies (r1.entry_number !=
355     r2.entry_number)
356     }
357
358     //all the lineup requests are related to only one supermarket
359     fact LineUpRequestOnlyInOneSupermarket {
360             all r : LineUpRequest |
361                     all s : Supermarket | s != r.supermarket implies r not in s.lineup_requests
362     }
363
```

```
364 //a lineup request is accepted iff is connected with the supermarket
365 fact RequestAcceptedLineUp {
366         all r: LineUpRequest | (r.status = ACCEPTED) iff
367                 (some s : Supermarket | r in s.lineup_requests)
368 }
369
370 //the estimated waiting time is always greater or equal then the time to reach the store
371 fact PathTimeUpperBoundWaiting {
372         all r: LineUpRequest |
373                 (r.estimated_waiting_time.hour >= r.estimated_path_time.hour) or
374                 (r.estimated_waiting_time.hour = r.estimated_path_time.hour and
375                  r.estimated_waiting_time.minute >= r.estimated_path_time.hour) or
376                 (r.estimated_waiting_time.hour = r.estimated_path_time.hour and
377                  r.estimated_waiting_time.minute = r.estimated_path_time.minute and
378                  r.estimated_waiting_time.second >=r.estimated_path_time.second) or
379                 (r.estimated_waiting_time = r.estimated_path_time)
380 }
381
382
383 //BOOK A VISIT REQUEST
384
385 //all the book a visit requests are related to only one supermarket
386 fact BookAVisitRequestOnlyInOneSupermarket {
387         all r : BookAVisitRequest |
388                 all s : Supermarket | s != r.supermarket implies r not in
389                                                 s.bookavisit_requests
390 }
391
392 //all the booked visits are in available days
393 fact RequestInAvailableDays {
394         all r : BookAVisitRequest | r.date not in r.supermarket.closing_days
395 }
396
397 //all the booked visits are in available hours
398 //due to the reduced expressive power of alloy is not easily checkable if
399 //                  start_time+duration <= supermarket.closing_hour
400 //because the sum of the hours, minutes and seconds could overstep the fixed interval
401 //for example start time: 22:40:00 + duration (00:30:00) = 22:70:00
402 //take into account this eventuality during the development of the code part
403
404 //checks that a request is in available hour
405 fact RequestInAvailableHours {
406         all r : BookAVisitRequest |
407                 (r.supermarket.closing_hour = r.supermarket.opening_hour)
408                 or ((r.start_time.hour < r.supermarket.closing_hour.hour)
409                         or (r.start_time.hour = r.supermarket.closing_hour.hour
410                         and r.start_time.minute < r.supermarket.closing_hour.minute)
411                 and
412                         (r.supermarket.opening_hour.hour< r.start_time.hour)
```

```
413                         or (r.supermarket.opening_hour.hour=r.start_time.hour
414                 and r.supermarket.opening_hour.minute<=r.start_time.minute))
415 }
416
417
418 //a book a visit request is accepted iff is connected with the supermarket
419 fact RequestAcceptedBookAVisit {
420         all r:BookAVisitRequest | (r.status = ACCEPTED) iff
421                 (some s : Supermarket | r in s.bookavisit_requests)
422 }
423
424 //a book a visit request is rejected and sended to the shoppinglist
425 //external check iff the capacity is full (NOT PRESENT IN THIS REDUCED
426 //REPRESENTATION)
427 fact RejectRequestOverCapacity {
428         all r: Request | (r.status = REJECTED) iff
429                 (#(r.supermarket.bookavisit_requests)=
430 r.supermarket.reservation_capacity)
431                 //and externalComponentShoppingList () = FALSE, for example
432 }
433
434
435 //TICKET MACHINE
436
437 //the ticket machine can not make booking requests
438 fact NoTicketMachineBooking {
439         no t : TicketMachine |
440                 #(t.ticketAccount.requests & BookAVisitRequest)>0
441 }
442
443 //there is only one ticket machine for each store
444 fact NoDoubleTicketMachine {
445         no disj t1, t2 : TicketMachine |
446                 #(t1.supermarket & t2.supermarket) >0
447 }
448
449 //there is only one predefined account for each machine
450 fact NoSamePredifinedAccount {
451         no disj t1, t2 : TicketMachine |
452                 #(t1.ticketAccount& t2.ticketAccount) >0
453 }
454
455 //ticketMachine on the same spot of the related supermarket
456 fact TicketMachinePosition {
457         no t1 : TicketMachine |
458                 t1.ticketAccount.position != t1.supermarket.position
459 }
460
461 //the ticket machine cannot obtain notifications from the system
```

```
462 fact TicketMachineNoNotifications {
463         no t1 : TicketMachine |
464                 t1.ticketAccount.notification_permission= TRUE
465 }
466
467 //NOTIFICATIONS
468
469 //notifications are not duplicated
470 fact NoDuplicateNotification {
471         no disj n1, n2 : Notification |
472                 (n1.receiver = n2.receiver) and (n1.supermarket = n2.supermarket) and
473                 (n1.available_date = n2.available_date)
474                 and (n1.available_time = n2.available_time)
475 }
476
477 //grants that a notification contains only hours in the working hours of the supermarket
478 fact NotificationTimeInWorkingHours {
479         all n: Notification |
480                 (n.available_date not in n.supermarket.closing_days) and
481                         ((n.supermarket.closing_hour = n.supermarket.opening_hour)
482                 or (
483                         (n.available_time.hour<n.supermarket.closing_hour.hour)
484
485                         or (n.available_time.hour=n.supermarket.closing_hour.hour
486                         and n.available_time.minute<n.supermarket.closing_hour.minute)
487
488                         or (n.available_time.hour=n.supermarket.closing_hour.hour
489                         and n.available_time.minute=n.supermarket.closing_hour.minute
490                         and n.available_time.second<n.supermarket.closing_hour.second)
491
492                         and (n.supermarket.opening_hour.hour<n.available_time.hour)
493                         or (n.supermarket.opening_hour.hour=n.available_time.hour
494                         and n.supermarket.opening_hour.minute<=n.available_time.minute)
495
496                         or (n.supermarket.opening_hour.hour=n.available_time.hour
497                         and n.supermarket.opening_hour.minute=n.available_time.minute
498                         and n.supermarket.opening_hour.second<=n.available_time.second)
499 ))
500 }
501
502 //notification only to customer that allows them
503 fact NotificationOnlyAllowed {
504         all n: Notification |
505                 (n.receiver.notification_permission)= TRUE
506 }
507
508
509
510 //TIME AND HOURS
```

```
511
512 //there is only one time with the same hour, minute and second
513 fact UniqueTime {
514         no disj st1, st2 : StructuredTime |
515                 st1.hour = st2.hour and st1.minute= st2.minute and st1.second=st2.second
516 }
517
518 ////////////////////////////////////////////////////////////////////////////////////////
519 ////////////////////////////////////COMMANDS////////////////////////////////////////////
520 ////////////////////////////////////////////////////////////////////////////////////////
521
522 pred FirstModel {
523         #LineUpRequest = 1
524         #BookAVisitRequest = 0
525         #Customer = 2 //you have to count also the ticket machine users
526         #Manager = 1
527         #Supermarket = 1
528         #Notification=0
529         #City = 1
530         #Street = 1
531         #Position = 2
532         #Product = 0
533         #Category = 0
534         #Date = 2
535         #TicketMachine = 1
536         #Password = 3
537         #SupermarketChain = 1
538         #QRcode= 1
539         #Time = 2
540 }
541
542 run FirstModel for 20                           //but 8 Int
543
544 pred SecondModel {
545         #LineUpRequest = 0
546         #BookAVisitRequest = 1
547         #Customer = 2 //you have to count also the ticket machine users
548         #Manager = 1
549         #Supermarket = 1
550         #Notification=0
551         #City = 1
552         #Street = 1
553         #Position = 3
554         #Product = 0
555         #Category = 0
556         #TicketMachine = 1
557         #SupermarketChain = 1
558         #Duration = 1
559 }
```

```
560
561 run SecondModel for 20                    //but 8 Int
562
563 pred ThirdModel {
564         #LineUpRequest = 1
565         #BookAVisitRequest = 1
566         #Customer = (3) //you have to count also the ticket machine users
567         #Manager = 1
568         #Supermarket = 2
569         #Notification=0
570         #City = 1
571         #Street = 2
572         #Position = 2
573         #Product = 2
574         #Category =2
575         #Date = 4
576         #TicketMachine = 2
577 }
578
579 run ThirdModel for 20                    //but 8 Int
580
581 pred FourthModel {
582         #LineUpRequest = 0
583         #BookAVisitRequest = 0
584         #Customer = (2) //you have to count also the ticket machine users
585         #Manager = 1
586         #Supermarket = 1
587         #Notification=1
588         #City = 1
589         #Street = 2
590         #Position = 2
591         #Product = 0
592         #Category =0
593         #Date = 4
594         #TicketMachine = 1
595         #Password = 3
596 }
597
598 run FourthModel for 20                    //but 8 Int
599
600
```

4.2 Alloy Clarification

During the analysis using alloy we have noticed some limitations of the tool that can conditionate the model. The bigger problem is related to the “time” concept inside the model. It is possible to place bound limitations regarding the hour/minute/seconds only by modifying the predefined size of the integers. Adding the sentence “but 8 Int” inside the “run” command is possible to modify this setting, in this case is possible to add the constraints “hour <=23 and minute<=59 and second<=59” inside “StructuredTime” for a more precise model. In addition to that it is not possible to correctly compute the sum of a start time and a duration, like better explained in the code without incrementing a lot the complexity of the code. For this reason, we have omitted that part.

Another important aspect, related to the simulation part, is that the number of customers corresponds to the sum of human customers and ticket machines.

In the section below it is possible to read the explanation of the four analysed models, there is also a thumbnail linked to a Drive folder (Complete diagrams) with the images loaded. It is also possible to find the graphs inside the GitHub folder.

4.3 First Model

In this model (Fig. 38) the focus is on the LineUp service. There is one supermarket with its Ticket Machine that has an account dedicated to the “in person” customers. That account has created a LineUp Request and the other customer (clearly with his Username, Email and Password) is a normal customer using his device without any request active. There is also a Manager, with his own credentials, that manages the Supermarket. The real customer is in a specific position while the Ticket Machine is placed correctly in the same position of the Supermarket. All the usernames are different, and this happens also for the emails. All the Positions are in a Street that is in a City.
Here it is possible to find the graph with an optimal image quality: First Model.

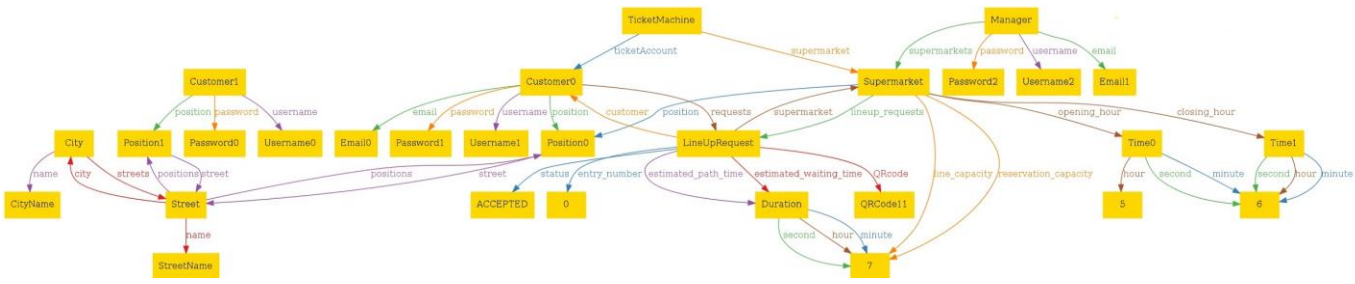

Figure 38: First Model

4.4 Second Model

In this model (Fig. 39) the focus is on the BookAVisit service. There is one supermarket with its related Ticket Machine and one user connected with his device. The customer has created a BookAVisit Request, that is accepted by the system and connected to its QRCode, Date, Starting Time of the visit and Duration. There is also a Manager, with his own credentials, that manages the Supermarket. In this case the supermarket is 24h open. The customer connected with his device is in a specific position that corresponds to the position of the supermarket and the ticket machine. This is a special case, obviously generated by the alloy tool. All the usernames are different, and this happens also for the emails. All the Positions are in a Street that is in a City.
Here it is possible to find the graph with an optimal image quality: Second Model.

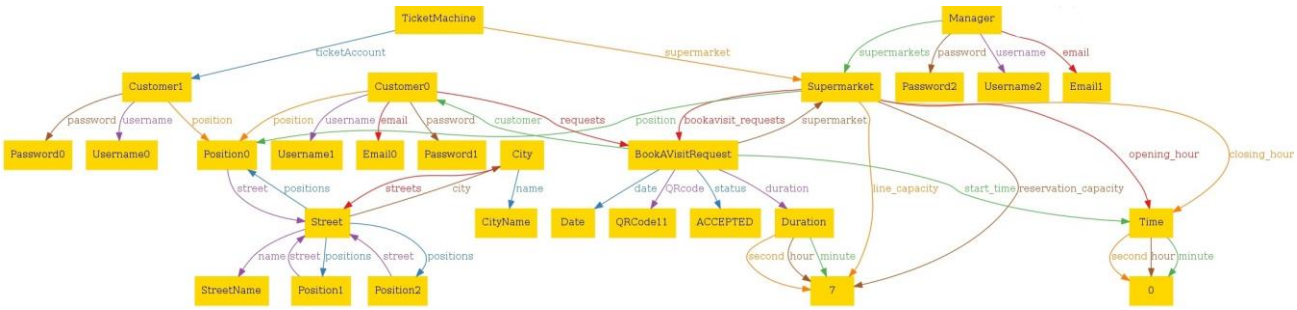

Figure 39: Second Model

4.5 Third Model

In this model (Fig. 40) both the BookAVisit service and the LineUp service are present. The aim of this representation is to show how multiple entities can appear in the model without any problems. In this case there are two different supermarkets, three customers (two ticket machines and one person with his/her device), and both the types of request. Is possible also to see the classification of a group of products in one category on the right side of the graph.
Here it is possible to find the graph with an optimal image quality: Third Model.

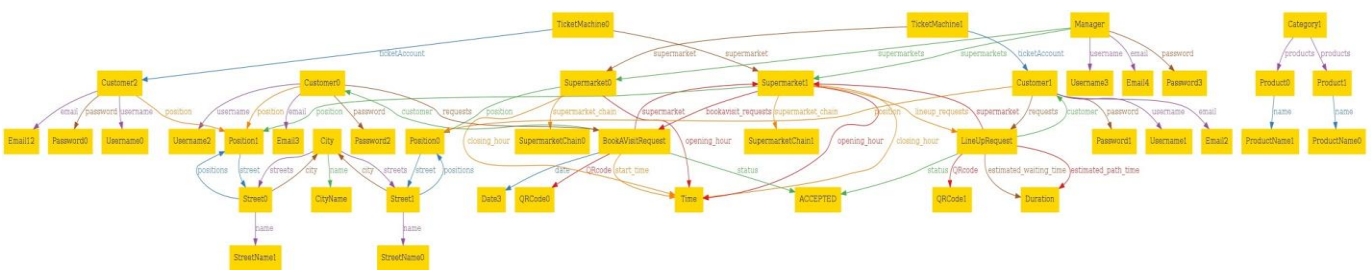

Figure 40: Third Model

4.6 Fourth Model

In this model (Fig. 41) the focus is on the Notification service. There is one supermarket with its related Ticket Machine and one user connected with his device. The customer is notified by the system with a Notification of an available slot and this can happen only because the “allow notifications” flag is set on true value. This notification contains the available date, the available time and obviously the supermarket name. There is also a Manager, with his own credentials, that manages the Supermarket. In this case the supermarket is 24h open. All the usernames are different, and this happens also for the emails. All the Positions are in a Street that is in a City. Here it is possible to find the graph with an optimal image quality: [Fourth Model.]

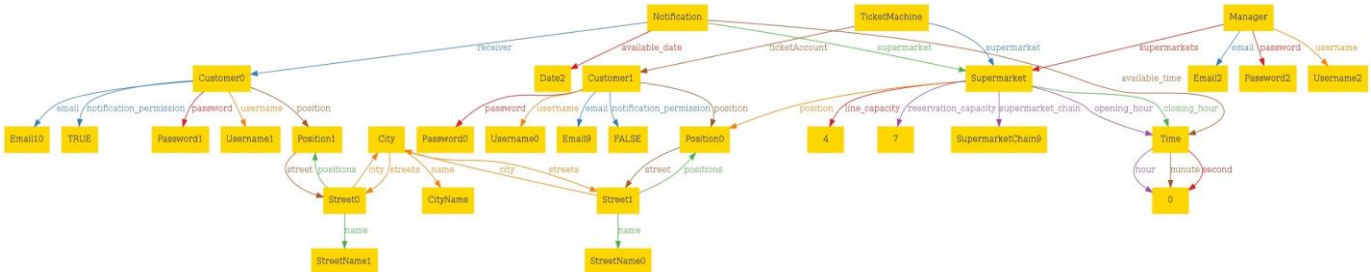

Figure 41: Fourth Model

5 Effort Spent

5.1 Teamwork

Task	Teamwork's Hours
Introduction	2
Product Perspective	4
Product Functions	3
Domain Assumptions	6
External Interface Requirements	1
Functional Requirements	4
Non-functional Requirements	1
Formal Analysis using Alloy	2

Table 18: Teamwork Effort

5.2 Individual Work

Task	Agnese's Hours
Introduction	3
Product Perspective	1,5
Product Functions	11
Domain Assumptions	1,5
External Interface Requirements	1,5
Functional Requirements	10
Non-functional Requirements	8
Formal Analysis using Alloy	3

Table 19: Agnese Straccia's Effort

Task	Cristiano's Hours
Introduction	0,5
Product Perspective	9,5
Product Functions	1,5
Domain Assumptions	6
External Interface Requirements	1
Functional Requirements	12,5
Non-functional Requirements	1,5
Formal Analysis using Alloy	3

Table 20: Cristiano Serafini's Effort

Task	Stefano's Hours
Introduction	0,5
Product Perspective	8
Product Functions	1
Domain Assumptions	1,5
External Interface Requirements	10
Functional Requirements	2
Non-functional Requirements	2
Formal Analysis using Alloy	16

Table 21: Stefano Vanerio's Effort

Appendices

A Revision History

The full revision history is available at this link: history.

- Version 1.0, online version on Google Docs.
- Version 2.0, final fixes, page enumeration and heading.

B Software and Tools used

- Git & GitHub as version control systems. The repository of the project is here.
- Google docs as a collaborative work platform.
- Draw.io for the pictures (interfaces and UML).
- Balsamiq for the mockups.
- Alloy as model analyzer.
- Astah Professional for the state diagrams.
- Visual Paradigm Online for the sequence diagrams and use cases.

6 References

[1] ISO/IEC/IEEE 29148. Systems and software engineering - Life cycle processes - Requirements engineering. Technical report, 2011.

[2] IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. Technical report, 1998. url: http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf.

[3] alloytool.org. Alloy Documentation. Software Design Group. url: http://alloytools.org/documentation.html.

[4] Michael Jackson. The world and the machine, 1995. url: http://mcs.open.ac.uk/mj665/icse17kn.pdf.

[5] Slides of the lessons, Software Engineering II, Professor Rossi.