RASD – Requirement Analysis and Specification Document

POLITECNICO
MILANO 1863

Immordino Alessandro – 969549
Pala Riccardo – 969598
Polvanesi Giacomo – 971083

Politecnico di Milano

A.A. 2020/21

Table of Contents

1 Introduction

1.1 Purpose

1.1.1 General purpose

This document represents the *Requirements Analysis and Specification Document* (RASD) referring to *CLup – Customers Line-up* system. It contains a detailed description of the proposed solution, listing all the Goals and the concerning Domain Assumptions and Requirements that satisfy them. It includes a general description of the *software-to-be*, involving the description of the main functions, interfaces, functional requirements, all the formal analysis brought through Alloy models and, finally, a section devoted to the effort spent writing the document.

This document main purpose is to provide developers with a guideline for implementation of the requirements relating to the *software-to-be*, guaranteeing a formal means of communication between them and the stakeholders.

1.1.2 Goals

The following list includes the main goals we want to achieve by implementing our software.

[**G1**]: Allow Client to enter a grocery store only if he has got a reservation.

[**G2**]: Allow Client to enter a grocery store only if it has not reached its maximum capacity.

[**G3**]: Maximize the number of Clients doing grocery shopping within each store.

[**G4**]: Prevent number of Clients in the *queue* from exceeding the maximum allowed number.

[**G5**]: Offer every Client the opportunity of booking a *visit* for doing grocery shopping.

[**G6**]: Suggest other options if User reservation request cannot be satisfied.

1.2 Scope

CLup is a booking service for grocery shopping. This *software-to-be* is focused on easing the access to the supermarkets during coronavirus emergency which imposed strict rules that must be observed by society. Deployment and usage of this application allow store managers to monitor entries and prevent people forming crowds outside the supermarkets. The service wants to ensure that distances between people are maintained and restrictions about the limited number of accesses in a building are respected.

CLup offers its users two types of reservation:

- **Real-Time Reservation** is the basic function of the *software-to-be*. The customer can search for his favourite supermarket and then he can retrieve a ticket to virtually line up from his smartphone. The system provides the *expected time of entrance* to access the supermarket and notifies users when their turn is coming. People who do not have access to the technology are not left out: fallback options to get a ticket avoiding the use of the application are provided. Indeed, one can obtain physical tickets from designated devices placed outside the stores.
- **Planned Reservation** is an advanced service which allows users to book a *visit* for doing grocery shopping at a later time. They can choose store, arrival time and duration of the *visit* from those available. This type of reservation gives the client a big advantage: the priority over those who made a *real-time reservation*. Indeed, users who plan a *visit* can jump the queue without waiting and can enter the store at the time fixed by the reservation. It could happen that customer request is not satisfied. For instance, a supermarket could be full at the selected time. In this case, the system provides him with suitable alternatives. *CLup* takes care of proposing the user another *schedule* for a *visit* in the same store; otherwise, it suggests some supermarkets, close to the selected one, available at the required time.

1.2.1 World phenomena

W1	Going to a supermarket
W2	Change of the restriction rules adopted by the government
W3	Increase of need of doing grocery shopping in certain periods

1.2.2 Shared phenomena

SP1	Access to the supermarket	Observed by the World	Controlled by the Machine
SP2	Cancelling a reservation	Observed by the Machine	Controlled by the World
SP3	Making a reservation	Observed by the Machine	Controlled by the World
SP4	Generating a reservation	Observed by the World	Controlled by the Machine
SP5	Generating a ticket	Observed by the World	Controlled by the Machine
SP6	Duration of a *visit*	Observed by the Machine	Controlled by the World
SP7	Turnout of People	Observed by the World	Controlled by the Machine
SP8	Lining up near the supermarket	Observed by the World	Controlled by the Machine

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- *Real-time reservation:* reservation allowing Client to join immediately the *virtual line-up* in order to enter the supermarket as soon as possible.
- *Planned reservation:* reservation allowing User to schedule a *visit* for doing grocery shopping afterwards.
- *Queue (Line):* physical line-up of people waiting for their turn outside a supermarket.
- *Virtual line-up:* ordered set of all *real-time reservations,* concerning a certain supermarket, including those neither scanned nor expired.
- *Expected time of entrance:* accurate estimate of the *visit* starting instant for a *real-time reservation*.
- *Maximum tolerated delay:* time interval, from the *visit* starting instant, after which a reservation, not scanned yet, expires
- *Schedule:* time slot of a *planned reservation* from the starting, to the ending instant.
- *Ticket number:* number of a *real-time reservation* describing its relative position in the *virtual line-up.*
- *Visit:* activity of doing grocery shopping.

1.3.2 Acronyms

- RASD = Requirement Analysis and Specification Document.
- API = Application Programming interface.
- QR = Quick Response.

1.3.3 Abbreviations

- Wn = nth world phenomena.
- [Gn] = nth goal.
- SPn = nth shared phenomena.
- [Rn] = nth requirement.
- [Dn] = nth domain assumption.
- ETE = expected time of entrance.
- MTD = maximum tolerated delay.

1.4 Reference documents

- Specification Documents: "R&DD Assignment AY 2020-2021"
- Market-leading Queue Management Software for enterprise brands: https://www.qudini.com/solutions/queue-management-system/
- MoSCoW method: https://en.wikipedia.org/wiki/MoSCoW_method
- IEEE Std 830-1993 - IEEE Guide to Software Requirements Specifications.

1.5 Document structures

Chapter 1: It is the introduction of the RASD. Here are highlighted the purposes by listing the main goals the application must reach. Moreover, there is a section of this chapter dedicated to the scope of the application, describing the main functionalities the *software-to-be* will provide, and its aim. In the introduction there is also a general view about the interactions between the external world and the application domain, stressing out which are the main fields that are affected.

Chapter 2: It offers an overall description of the *software-to-be*. First there is a description of the actors involved in their usage lifecycle and their role. Then detailed information about their interactions are provided by means of a class diagram of the software, including its most important entities. State chart diagrams are represented in order to deepen the lifecycle of some components. Some boundaries and facts, that are assumed to hold in the world, are listed in the domain assumption section: combined with the requirements of the next chapter, they will satisfy the goals of the *software-to-be*.

Chapter 3: It is the body of the document. Here, all requirements of the *software-to-be* are listed. They are related to multiple features: user, software and hardware interfaces. It then describes some scenarios concerning different possible real situations the software must handle, followed by the description of functional requirements, use cases and sequence diagrams. Finally, the non-functional requirements are defined through software attributes, performance requirements and design constraints.

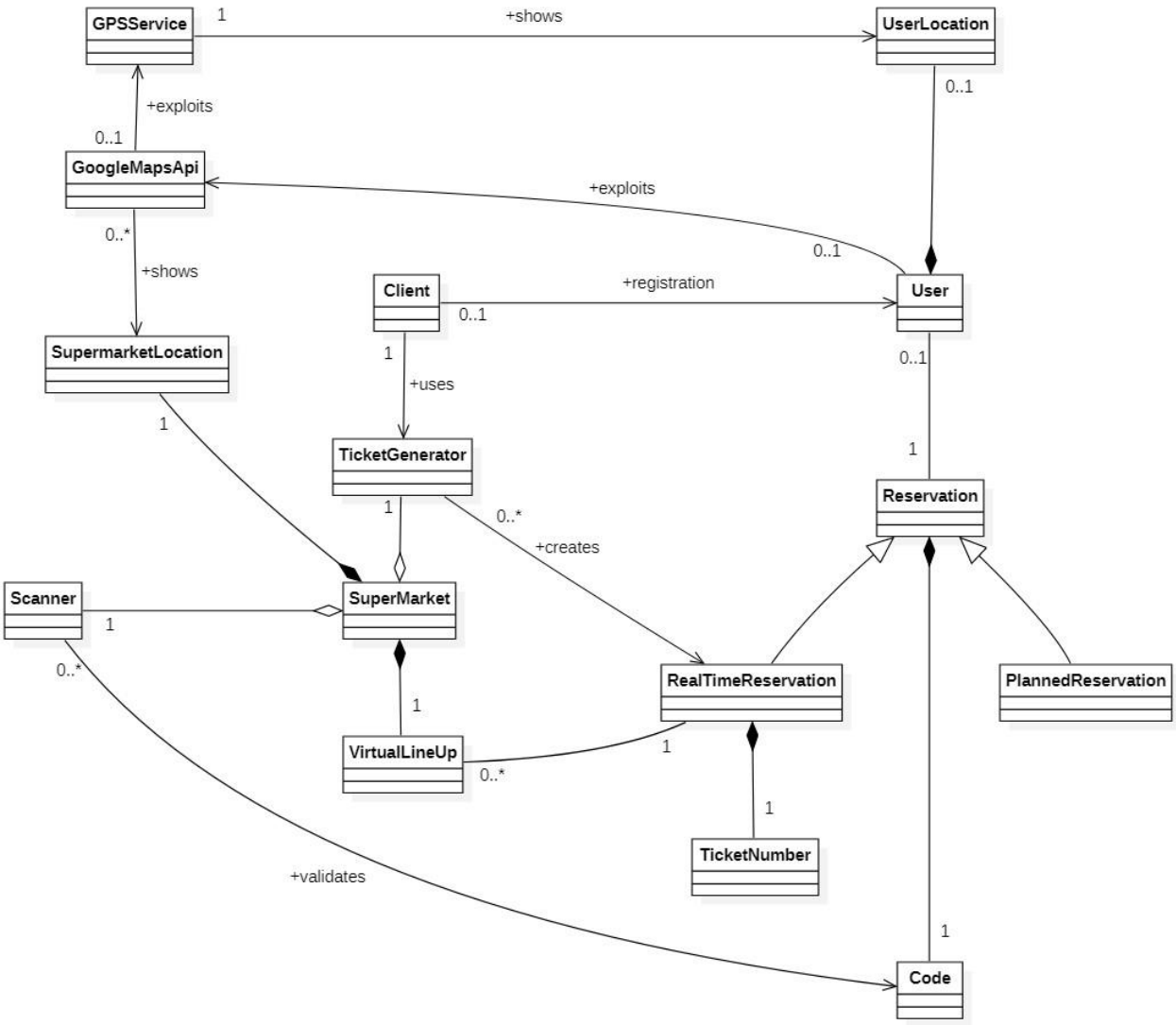
Chapter 4: It describes the Alloy Model which highlights detailed features and boundaries between the entities of the *software-to-be*.

2 Overall description

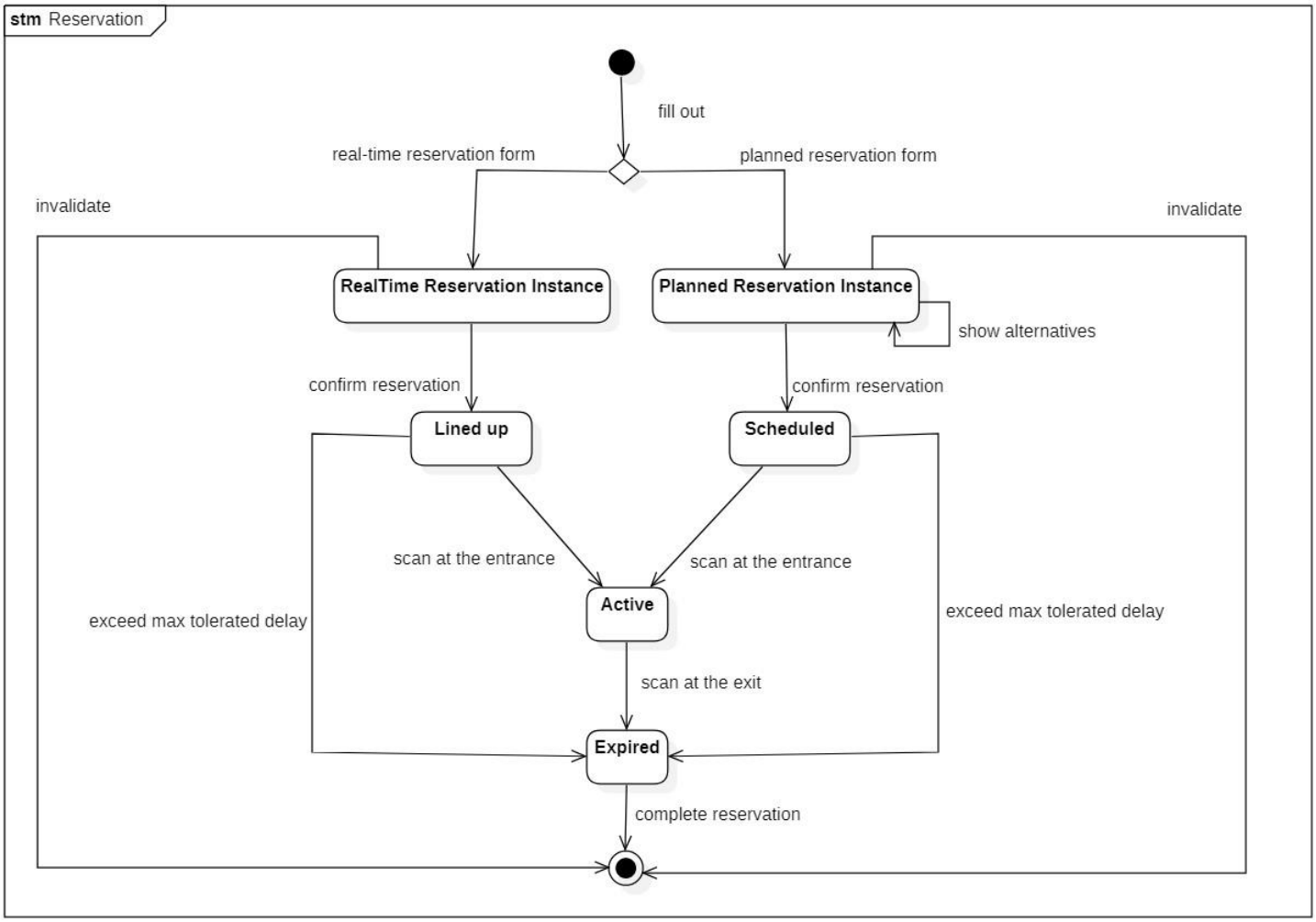
2.1 Product perspectives

The idea behind *CLup* is that people can use the mobile app to search for their favourite supermarket among those belonging to the system by exploiting Google Maps API and then they can decide to make a *planned reservation* or a *real-time reservation*. Each supermarket has also a device, called TicketGenerator, that allows people that are not able to use mobile app services to make a *real-time reservation*, producing the corresponding *ticket number*. The flow of the accesses in a supermarket is handled by another device, called Scanner, that validates the reservations and let the sliding doors open.

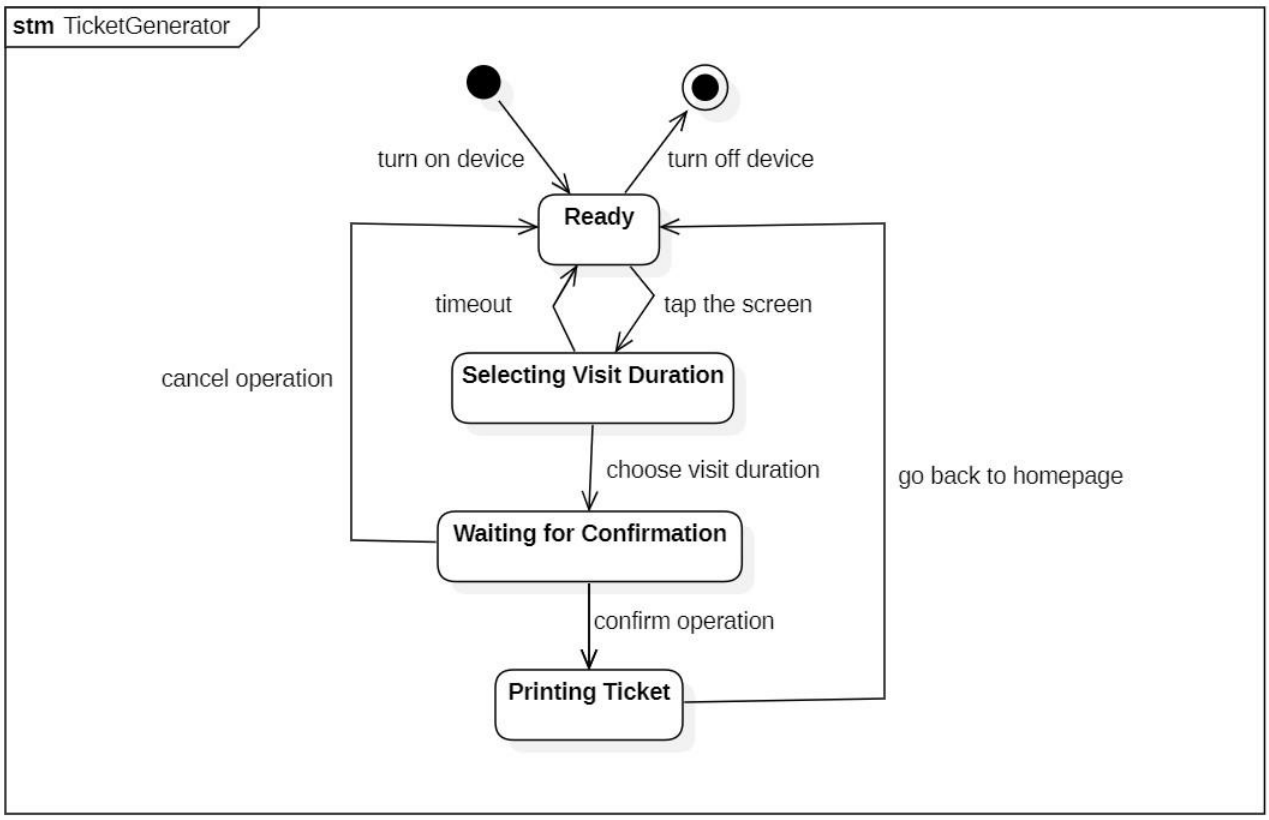
The following class diagram represents interactions among involved entities, which were just described above.



The following state chart diagram depicts the lifecycle of a reservation, from its generation up to its expiration, that occurs once it is scanned at the exit. Depending on its type, the reservation goes through different states: a lined-up state if it is a *real-time reservation* or in a scheduled state if it is a *planned reservation*. This is because users that have decided to plan their *visit* does not have to wait in *line* for their turn. They simply can enter if the starting time of the *visit* has come.



This state chart diagram describes all the states through which the TicketGenerator goes during its lifecycle.



2.2 Product functions

In this section, the main functions of the *software-to-be* are listed in a descendent order of relevance:

2.2.1 Reservation management

This function is the most important for the *software-to-be*. It consists of handling the reservations made by Clients, deciding whether to insert them in the corresponding *virtual line-up* or in the database timetable of the selected supermarket, depending on they are *real-time* or *planned reservations*. In the latter case, the waiting time is computed and the ETE is showed to the Client, including the *ticket number.* The computation of the ETE is based on how many Clients have a *real-time reservation* with a lower *ticket number* and how many *planned reservations* are scheduled for that day. The aim is to fill the first available spot considering how many people are expected to be inside the facility and the *visit* duration.

At this point, either Client refuses it, if the ETE is not what he expected, or he confirms the booking. The reservation management permits the flow of the *queue* of each supermarket and grants a place inside the supermarket for who has got a *planned reservation*. It also limits to one the number of active reservations for each User, avoiding a possible overlap between two booking done by the same person.

2.2.2 Accesses management

The aim of this function is to regulate the accesses to supermarkets. The idea is that Scanner reads the QR code related to the reservations, which then are validated or not. This function calculates how many people are inside the supermarket by summing the number of reservations that were scanned at the entrance but have not been scanned at the exit yet, to the number of *planned reservations* that are in the timetable. This number is then subtracted to the maximum capacity of the facility: if the result is equal to zero it means that another validation will cause an excess of the maximum capacity. If the result is a positive number, the display outside shows the *ticket number* of the last *real-time reservation* in the *virtual line-up* allowed to enter. So, every Client in the *virtual line-up,* having a *ticket number* lower than the showed one, are authorized to enter. Otherwise, if Clients that are allowed to enter do not pass the reservation beneath Scanner within the fixed *maximum tolerated delay,* the reservation will expire and they will have to make another reservation to enter. Users can check in real-time the status of their reservation and the status of the *virtual line-up* via app.

2.2.3 Alternatives proposal management

This function is addressed to Users who try to make a *planned reservation* and consists of showing different supermarkets in the case the selected one is not available, since the maximum capacity is reached. It finds two suitable choices that are proposed to User, having some features in common with the not more available one: the same *schedule* in the nearest supermarket or the same supermarket but in a different *schedule*. Client opts for an alternative or refuses returning then to the home page. In the first case, the system shows the form again to generate the reservation. It may happen that the selected choice is no more available if another User did a reservation during the time interleaved between selecting a choice and submitting the form again. In these situations, the system shows options until User either refuses proposal or finally gets a reservation.

2.3 User characteristics

2.3.1 Actors

- *User:* person registered to the system who is enabled to use the services via app.
- *Client:* customer of any store belonging to the system.
- *TicketGenerator:* device, located at the entrance of any store, allowing customer to make a *real-time reservation* without using the app.
- *Scanner:* device, located at the entrance of any store, dealing with code reservation scanning.

2.4 Assumptions, dependencies and constraints

The following properties are assumed to hold. These are assumptions concerning the analysed world that the system cannot check.

[**D1**]: Automatic sliding doors regulates entries into each store.

[**D2**]: At the entrance of each store, a device deals with reservation code scanning.

[**D3**]: Due to health restrictions imposed by the pandemic situation, each store has a maximum capacity of people authorized to be within at the same time.

[**D4**]: Client having a *real-time reservation* does not line up until the *expected time of entrance* is sufficiently close.

[**D5**]: Client having a *planned reservation* does not have to line up.

[**D6**]: Clients waiting in *queue* follow health rules in force. The restrictions impose the observance of a physical distance between two people in open spaces.

[**D7**]: At the entrance of each supermarket a display shows the number of last *real-time reservation* allowed to enter in a certain time, it also displays the maximum capacity and the number of Clients within the store in that moment.

[**D8**]: A device located outside the store helps Clients, who do not have the app, to make a *real-time reservation*.

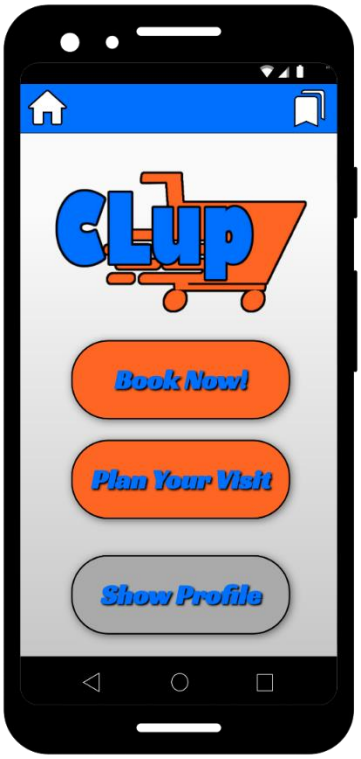
[**D9**]: At the checkout, the reservation code is scanned again.

[**D10**]: A unique position is related to each store.

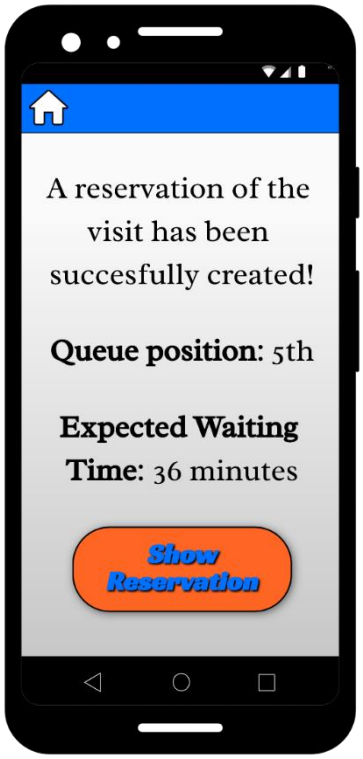
3 Specific requirements

3.1 External interface requirements

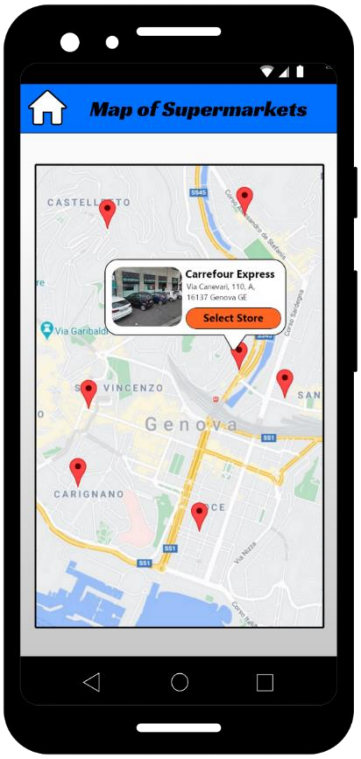
3.1.1 User interfaces



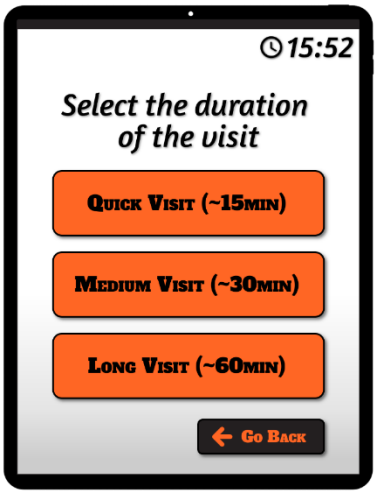
Mobile App – Homepage



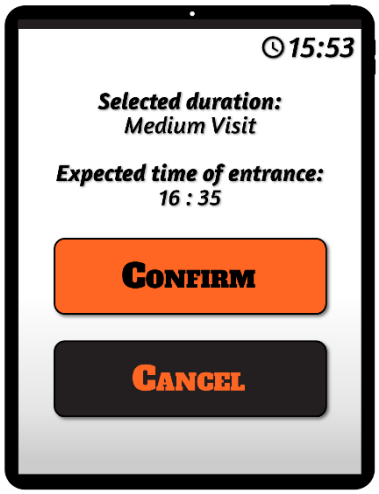
Mobile App – Reservation Confirmation



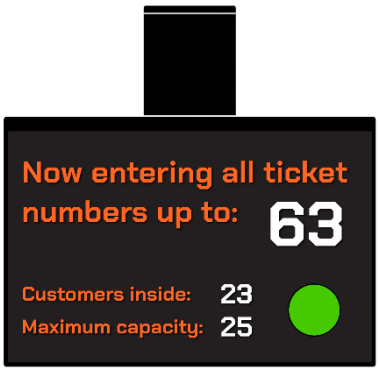
Mobile App – Map of Supermarkets



TicketGenerator – Select Duration



TicketGenerator – Reservation Confirmation



Display

3.1.2 Hardware interfaces

The supermarkets need some hardware interfaces in order to create a communication between the Scanner and the sliding doors, between the TicketGenerator and the system and finally between the Scanner and the display.

3.1.3 Software interfaces

The *software-to-be* requires some software interfaces to create a communication between its subcomponents. Here the most important ones are listed:

- Google Maps API: it is needed by User to search for all supermarkets registered to *CLup* services.
- Extract data interface: this interface provides functions such as:
  - Fetching data about the duration of the reservations in the timetable of each supermarket, to send these to the subcomponent which deals with ETE computation.
  - Retrieving all the *planned reservations* of a given supermarket in order to ensure that their number does not overcome the maximum capacity in every moment.
- Communication with Client: some interfaces that ensure an exchange of data between software and Clients, for example showing their position in the *virtual line-up*, are necessary.

3.2 Functional requirements

For the purpose of highlighting the different relevance of each requirement, we used a prioritization technique known as *MoSCoW* method.

In particular:

- MUST represents a critical requirement which is essential in order to consider the *software-to-be* a success.
- SHOULD identifies a high-priority constraint which is very important but not strictly necessary for application success.
- COULD describes a desirable but not necessary requirement that could improve the user experience.

Following constraints make the *software-to-be* achieve the purposes proposed at the beginning of this document (section 1.1.2). Assuming domain properties hold, these requirements deal with the satisfaction of the goals previous mentioned.

[**R1**]: System must assign a unique code to every reservation.

[**R2**]: System should produce a QR code for each reservation based on its unique code.

[**R3**]: System must assign a *ticket number* to every *real-time reservation*.

[**R4**]: System must unlock the sliding door if and only if the reservation code scanning gives a positive outcome.

[**R5**]: System must lock sliding doors right after Client entrance.

[**R6**]: A reservation code scanning must give a positive outcome only if such reservation concerns the supermarket Client is attempting to enter.

[**R7**]: A code scanning of a *real-time reservation* must give a positive outcome only if the store has not reached its maximum capacity.

[**R8**]: A code scanning of a *real-time reservation* must give a positive outcome only if the corresponding reservation number is among those of the *virtual line-up* authorized to enter, since their entrance does not make the store exceed the maximum capacity.

[**R9**]: A code scanning of a *planned reservation* must give a positive outcome only if the actual time is included in the interval between the starting time of its *schedule* and the *maximum tolerated delay*.

[**R10**]: System must keep track of the number of Clients within the store.

[**R11**]: System should provide Client, having a *real-time reservation*, with *expected time of entrance*.

[**R12**]: System could notify User, who made a reservation, that his turn is coming.

[**R13**]: System should allow the *line* to flow anyway, even if one of Clients next in the *queue* did not arrive within the *maximum tolerated delay*.

[**R14**]: System must validate a *real-time reservation* request only if it occurs during the hours of operation of the chosen supermarket.

[**R15**]: System must validate a *planned reservation* request if and only if at least one slot is available in the chosen store for the selected *schedule*.

[**R16**]: System must forbid User from making a reservation if he has got another active one.

[**R17**]: System must propose making a *planned reservation* starting from some hours later the current time.

[**R18**]: System should detect an alternative *schedule* for doing grocery shopping in the chosen store if the selected one is not available.

[**R19**]: System could find the closest stores to the chosen one, in which the selected *schedule* is available.

[**R20**]: System must deallocate the slot occupied by a reservation when its *schedule* expires, or Client terminates his *visit,* or he does not scan his reservation within a certain *maximum tolerated delay*.

3.3 Requirements traceability

G1	Allow Client to enter a grocery store only if he has got a reservation.
D1	Automatic sliding doors regulates entry into each store.
D2	At the entrance of each store a device deals with reservation code scanning.
R1	System must assign a unique code to every reservation.
R2	System should produce a QR code for each reservation based on its unique code.
R3	System must assign a *ticket number* to every *real-time reservation*.
R4	System must unlock the sliding door if and only if the reservation code scanning gives a positive outcome.
R5	System must lock sliding doors right after Client entrance.
R6	A reservation code scanning must give a positive outcome only if such reservation concerns the supermarket Client is attempting to enter.

G2	Allow Client to enter a grocery store only if it has not reached its maximum capacity.
D1	Automatic sliding doors regulates entry into each store.
D2	At the entrance of each store a device deals with reservation code scanning.
D3	Due to health restrictions imposed by the pandemic situation, each store has a maximum capacity of people authorized to be within at the same time.
D9	At the checkout the reservation code is scanned again.
R4	System must unlock the sliding door if and only if the reservation code scanning gives a positive outcome.
R5	System must lock sliding doors right after Client entrance.
R6	A reservation code scanning must give a positive outcome only if such reservation concerns the supermarket Client is attempting to enter.
R7	A code scanning of a *real-time reservation* must give a positive outcome only if the store has not reached its maximum capacity yet.
R8	A code scanning of a *real-time reservation* must give a positive outcome only if the corresponding reservation number is among those of the *virtual line-up* authorized to enter, since their entrance does not make the store exceed the maximum capacity.
R9	A code scanning of a *planned reservation* must give a positive outcome only if the actual time is included in the interval between the starting time of its *schedule* and the *maximum tolerated delay*.
R10	System must keep track of the number of people within the store.
R20	System must deallocate the slot occupied by a reservation when its *schedule* expires, or Client terminates his *visit,* or he does not scan his reservation within a certain *maximum tolerated delay*.

G3	Maximize the number of Clients doing grocery shopping within each store.
D3	Due to health restrictions imposed by the pandemic situation, each store has a maximum capacity of people authorized to be within at the same time.
D7	At the entrance of each supermarket a display shows the number of last *real-time reservation* allowed to enter in a certain time, it also displays the maximum capacity and the number of Clients within the store in that moment.
D9	At the checkout the reservation code is scanned again.
R3	System must assign a *ticket number* to every *real-time reservation*.
R10	System must keep track of the number of Clients within the store.
R11	System should provide Client, having a *real-time reservation*, with *expected time of entrance*.
R12	System could notify User, who made a reservation, that his turn is coming.
R13	System should allow the *line* to flow anyway, even if one of Clients next in the *queue* did not arrive within the *maximum tolerated delay*.
R20	System must deallocate the slot occupied by a reservation when its *schedule* expires, or Client terminates his *visit,* or he does not scan his reservation within a certain *maximum tolerated delay*.

G4	Prevent number of Clients in the *queue* from exceeding the maximum allowed number.
D4	Client having a *real-time reservation* does not line up until the *expected entrance time* is sufficiently close.
D5	Client having a *planned reservation* does not have to line up.
D6	Clients waiting in queue follow health rules in force. The restrictions impose the observance of a physical distance between two people in open spaces.
D7	At the entrance of each supermarket a display shows the number of last *real-time reservation* allowed to enter in a certain time, it also displays the maximum capacity and the number of Clients within the store in that moment.
D9	At the checkout, the reservation code is scanned again.
R3	System must assign a number to every *real-time reservation* in order to represent its position in the *virtual line-up.*
R10.	System must keep track of the number of Clients within the store
R11	System should provide Client, having a *real-time reservation*, with *expected entrance time.*
R12	The system could notify User, who made a reservation, that his turn is coming.
R13	The system should allow the *line* to flow anyway, even if one of Clients next in the *queue* did not arrive within the *maximum tolerated delay*.
R20	System must deallocate the slot occupied by a reservation when its *schedule* expires, or Client terminates his *visit,* or he does not scan his reservation within a certain *maximum tolerated delay*.

G5	Offer every Client the opportunity of booking a *visit* for doing grocery shopping.
D8	A device located outside the store helps Clients, who do not have the app, to make a *real-time reservation*.
D10	A unique position is related to each store.
R14	System must validate a *real-time reservation* request only if it occurs during the hours of operation of the chosen supermarket.
R15	System must validate a *planned reservation* request if and only if at least one slot is available in the chosen store for the selected *schedule*.
R16	System must forbid User from making a reservation if he has got another active one.
R17	System must propose making a *planned reservation* starting from some hours later the current time.

G6	Suggest other options if User reservation request cannot be satisfied.
D10	A unique position is related to each store.
R18	System should detect an alternative *schedule* for doing grocery shopping in the chosen store if the selected one is not available.
R19	System could find the closest stores to the chosen one, in which the selected *schedule* is available.

3.4 Scenarios, use cases and sequence diagrams

3.4.1 Scenarios

3.4.1.1 Scenario 1: Real-time reservation via TicketGenerator

Anna is an elderly lady who does not own a smartphone. She needs to do grocery shopping, so she usually *visit*s her trusted supermarket that is in front of her building.

One day she arrives near the supermarket and she notices the presence of a device near the entrance. Then she approaches it trying to understand its functionalities. A message is shown on the screen inviting the customers to use the device to get a ticket in order to enter the supermarket. This device asks her to touch the screen to proceed. Anna does it and then she is asked to choose one of the three options, according to her needs: "Quick Visit (~15min)", "Medium Visit (~30min)", "Long Visit (~60min)". After having selected the second one, the device provides her with an expected time of entrance of about half an hour from the current time. At this point, device asks her either to confirm or to cancel the operation. Anna presses on the "Confirm" button. The device generates the ticket concerning her booking and it invites her to return to the facility once her turn is coming. Anna looks the ticket where number “38” is marked, so she walks away.

After 20 minutes Anna returns near the supermarket. There are some minutes left before the expected time of entrance of her visit. She reads on the display that the last number allowed to enter is “36”, therefore she joins the queue.

3.4.1.2 Scenario 2: Real-time reservation via app

Alice is in the city centre for some business and would like to buy something for dinner. She accesses the *CLup* application that she downloaded on her smartphone and that she has already been using for some time.

After logging in, Alice is in the homepage, from where she can reach the Real-time reservation form directly by pressing the relative button.

In that page System suggest Alice to book a visit in the supermarket that she included in her preferences at the time of registration. However, this time Alice is not in the proximity of her trusted supermarket, so she looks for a different one.

Since she does not know exactly the grocery stores in the area, she clicks on the map icon that appears in the top right corner. A pop-up asks her to turn on her smartphone location functionality, so that the system can recognize her position and show her the nearest supermarkets. Alice clicks on the closest one to her position. System shows her the exact address and the distance in meters from her current location. Alice confirms and returns to the previous screen. This time, however, the name and address of the supermarket are the

ones she has just selected on the map. At this point Alice presses the "Quick Visit (~15min)" button to answer the request "Select the duration of the visit", which is just below the field showing the address of the selected supermarket. Pressing on "Submit request" Alice sends the reservation query to System, which promptly answers to her by attaching an expected time of entrance of 40 minutes from that moment. It asks her if she wants to confirm or to return to the previous screen.

Alice is not willing to wait 40 minutes, so she decides to make another attempt: she then goes back to the previous page, selects another supermarket near her location from the map, clicks again on "Quick Visit (~15min)" and sends the request. System responds promptly, indicating an expected time of entrance that implies a 15-minute waiting time.

This time Alice is satisfied and decides to confirm. The system then sends her the ticket for her reservation and invites her to reach the store only a few minutes before the expected time of entrance.

The selected supermarket, as indicated on the map, is 500 meters away from her location. Therefore, Alice closes the application and decides to calmly heads towards the grocery shop.

3.4.1.3 Scenario 3: Planned reservation

At 16:00 Mario's mother asks him to go to the supermarket for buying some things for the dinner. Mario should finish his homework for school before going, so he takes his smartphone and opens *CLup*. After logging in, he selects "Plan your visit" button, afterwards he clicks map icon to search for the supermarket where his mom usually goes to. After selecting the facility and confirming, three options appear on the touchscreen: "Quick Visit (~15min)", "Medium Visit (~30min)" and "Long Visit (~60min)". Mario selects the first option, so another view appears on his smartphone. It requires him to insert the start time of the visit. Since he will not finish his homework until 18:00, he inserts 18:30 in designated field.

At this point a message appears saying the store is available for the selected time. Mario confirms the booking, so the app comes back to the home page. Mario opens "My reservation" section.

A new page shows all details of the visit. From this page Mario downloads the QR code and then he focuses on his homework. At 18:10 an alert comes from *CLup* on his smartphone. It warns him that the time of his visit is coming and only a 10-minute delay is tolerated.

Mario suddenly heads towards the supermarket, reaching it at 18:.31. He immediately goes in front of the sliding door where a scanner is located. Mario puts on it the QR code he has previously downloaded. The scanner recognizes positively his reservation, so the sliding door automatically opens. Mario enters the supermarket and sliding door automatically closes behind him. Now he can finally buy the ingredients his mother required.

3.4.1.4 Scenario 4: Planned reservation with alternatives

Sara is a young student who has been working part-time as a cashier in a store every afternoon from 16:00 onwards.

At 8:00 Sara is going to attend a class which starts after 15 minutes. She remembers having invited her best friend for dinner, but she has not got enough food. She is going to buy some fresh pasta, some nut sauce, two steaks and some red wine. She expects to spend at most 30 minutes to take everything, paying and going out.

However, Sara remembers she is finishing class at 15:00. Fortunately, her supermarket is right underneath her house. However, she has not got much time and knows she will have to hurry up in order to not be late for work.

Sara, without further delay, opens *CLup* application installed on her smartphone and logs in with her credentials. She presses the "Plan your visit" button and, as usual, she is redirected to the form already filled out with information about her favourite supermarket. Sara selects "Medium Visit (~30min)" and opt for 15:15 as the start time of the visit.

After having carried out the appropriate checks, System informs Sara, by means of a message printed on the screen, that the supermarket has already reached the maximum capacity in the specified time slot, due to the other reservations made. At this point System offers her two options: either making a reservation in the same supermarket at 16:00 (as well as the first available time) or making a reservation for the indicated time in another supermarket.

Since Sara cannot change her schedule, she decides to take the second way touching the corresponding button. After a brief search by System, Sara is redirected to a page with three views, each containing information of one of the three supermarkets found (they are the closest to the one she indicated earlier and available for the desired time slot). She realizes that each box also includes the distance between the indicated preference and the proposed structure. So, she decides to opt for the closest one, which is only 100 meters far from the previous one.

She is now asked to confirm the choice. After a quick summary about reservation data, Sara confirms, then a positive message appears. Finally, the application returns to the homepage.

3.4.1.5 Scenario 5: Ticket scanning for entrance

Gianluca is at home waiting for *CLup* notifying him that his turn to do grocery shopping is coming.

When the message arrives on his smartphone, he heads towards the grocery store.

Once he gets there, he finds no one waiting for the entrance. Gianluca checks the number showed on the screen near the sliding doors and he finds out he can enter. Therefore, he opens *CLup* and clicks on "My reservation" section. On this section he can displays the QR code concerning his reservation.

Gianluca tries to scan his QR code by the designated device. In one moment, Scanner reads the reservation code and outputs a green light. Sliding doors automatically open and Gianluca can enter the store to finally do the grocery shopping.

3.4.1.6 Scenario 6: Registration

Giorgia is coming back home by bus after work. She is listening to music and scrolling the posts on her Facebook wall.

Suddenly she stops on an advertisement which promotes a new booking service adopted by several supermarket chains. Some stores will start using this system to manage the stream of people going to do grocery shopping, in order to respect the new restrictions imposed by the government concerning Coronavirus emergency.

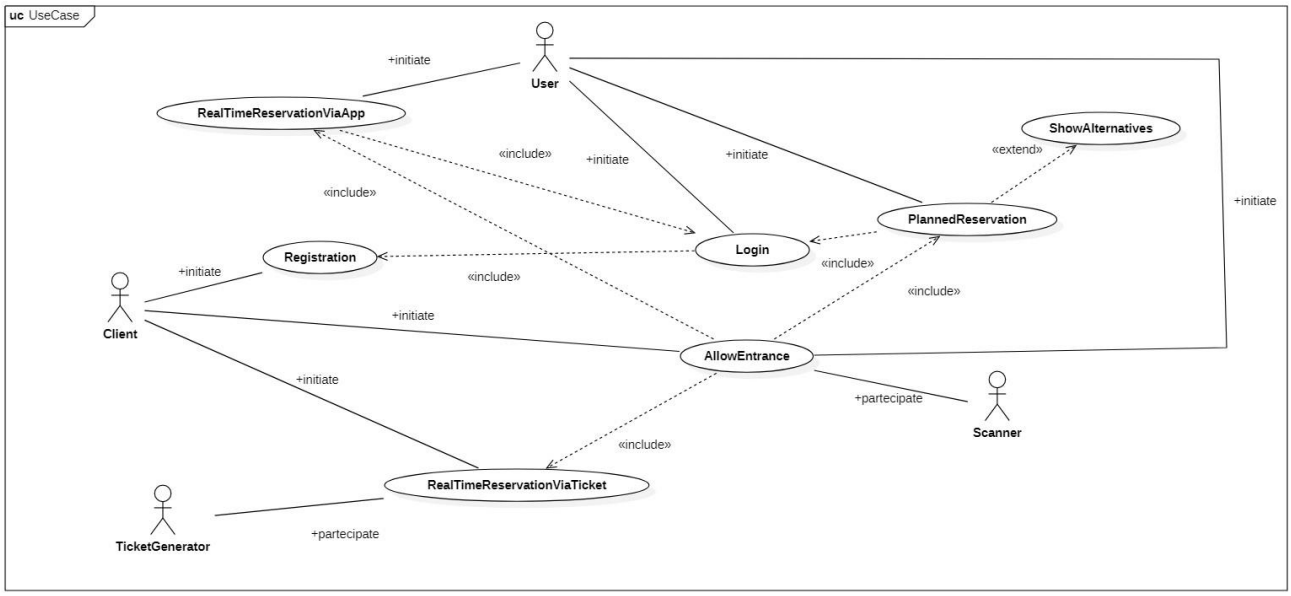
Giorgia is interested, so she opens the apps store of her smartphone to search and install this application.

Once she downloads *CLup*, she tries to open it. In the starting page she is required to log into the application or to sign up if one has never done before. Giorgia clicks on "Registration" sign. She is required to provide some general information, like first name, last name, ID number and birthday date. Then she moves on and system asks her to choose her favourite supermarket filling out the field with the address of its location or searching it on the map by clicking the map icon. Giorgia starts writing some digits on the field and the system promptly suggests her some stores matching up with the research. She selects hers. Finally, she ticks the option "Accept privacy policy" and confirms.

Now registration is completed. *CLup* welcomes Giorgia and it brings her to the starting page.

3.4.2 Use cases

Use case diagram:



3.4.2.1 Use case 1:

Name: Make a *real-time reservation* via TicketGenerator

Actors: Client, TicketGenerator, System

Entry Conditions: Client has got no account on *CLup*.

Event Flow:

1. Client touches the screen.
2. Client selects "Medium Visit (~30min)" option.
3. TicketGenerator sends a request to System based on selected option.
4. System estimates the waiting time and sends relative ETE to TicketGenerator.
5. TicketGenerator shows ETE to Client.
6. Client confirms his request.
7. TicketGenerator sends the request to System.
8. System confirms the reservation.

Exit Conditions: System places the reservation made by Client in the *virtual line-up*. TicketGenerator prints the ticket including the *ticket number*.

Exceptions: The waiting time added to the estimated *visit* duration exceeds the closing time of the store. In this case, Client cannot book a *visit* on this day.

3.4.2.2 Use case 2:

Name: Make a *real-time reservation* via app

Actors: User, System

Entry Conditions: User has already registered on *CLup*.

Event Flow:

1. User logs into the application.
2. User clicks on "Book now!" sign accessing to the relative functionality.
3. User opens the map of supermarkets.
4. User authorizes the use of GPS localizer on his smartphone.
5. User selects a store, among those identified by System as the closest to his position.
6. User chooses "Quick Visit (~15min)" option.
7. User submits his request.
8. System calculates the waiting time and shows ETE to User.
9. User cancels his request coming back to the previous screen.
10. User selects another store, near to his currently position, using the map.
11. User chooses "Quick Visit (~15min)" option.
12. User submits his request.
13. System calculates the waiting time and shows ETE to User.
14. User confirms his reservation.

Exit Conditions: System adds the reservation to the *virtual line-up*. QR code concerning the reservation is available.

Exceptions: The waiting time added to the estimated *visit* duration exceeds the closing time of the store. In this case, Client cannot book a *visit* on this day.

3.4.2.3 Use case 3:

Name: Make a *planned reservation*

Actors: User

Entry Conditions: User has already registered on *CLup*.

Event Flow:

1. User logs into the application.
2. User clicks on "Plan your visit" sign accessing to the relative functionality.
3. User opens the map of supermarkets.
4. User selects a store.
5. User chooses "Quick Visit (~15min)" option.

6. User selects the initial time of the *visit*.
7. User submits his request.
8. System checks the number of reservations at the selected time does not exceed the maximum capacity of the supermarket.
9. System returns a positive response.
10. User confirms his reservation.

Exit Conditions: The *schedule* referring to the reservation is added to the database timetable. QR code concerning the reservation is available. System sets an alert that notifies User 20 minutes of the *visit* starting time.

3.4.2.4 Use case 4:

Name: Make a *planned reservation* with alternatives

Actors: User

Entry Conditions: User has already registered on *CLup*.

Event Flow:

1. User logs into the application.
2. User clicks on "Plan your visit" sign accessing to the relative functionality.
3. User confirms auto-filled form including information about his favourite supermarket.
4. User chooses "Medium Visit (~30min)" option.
5. User selects the initial time of the *visit*.
6. User submits his request.
7. System checks the number of reservations at the selected time does not exceed the maximum capacity of the supermarket.
8. System returns a negative response.
9. System looks for the first suitable time slot in the specified facility.
10. System shows User two alternatives in order to overcome the unavailability.
11. User chooses to search for another facility available in the same time slot.
12. System seeks the three closest facilities available in the specified time slot, then shows them to User.
13. User selects one of the proposed alternatives.
14. User views the auto-filled form including information about the selected supermarket and the *visit* initial time.
15. User chooses "Medium Visit (~30min)" option.
16. User submits his request.
17. System checks the number of reservations at the selected time does not exceed the maximum capacity of the supermarket.
18. System returns a positive response.
19. User confirms his reservation.

Exit Conditions: The *schedule* referring to the reservation is added to the database timetable. QR code concerning the reservation is available. System sets an alert that notifies User 20 minutes of the *visit* starting time.

Exceptions: Less than three alternative available facilities are found within a certain maximum distance, so only the found ones are shown. If no suitable facilities were found, System print a message that informs User of the negative result.

3.4.2.5 Use case 5:

Name: Ticket scanning for entrance

Actors: Client, Scanner

Entry Conditions: Client has already made a *real-time reservation* and he has got his ticket.

Event Flow:

1. Client receives *CLup* notification advising him that his turn is coming.
2. Client moves to the grocery store.
3. Client checks if his *ticket number* is less or equal than the one displayed on the screen.
4. Client retrieves from *CLup* the QR code referred to his reservation.
5. Client places the QR code on Scanner.
6. Scanner outputs a green light.
7. System enables the sliding doors.
8. Client enters the store.

Exit Conditions: System removes the corresponding reservation from the *virtual line-up.*

Exceptions: Scanner outputs a red light: either it may not be his turn or his reservation expired, otherwise an error occurred on the device.

3.4.2.6 Use case 6:

Name: Registration

Actors: Client

Entry Conditions: Client has got no account on *CLup*.

Event Flow:

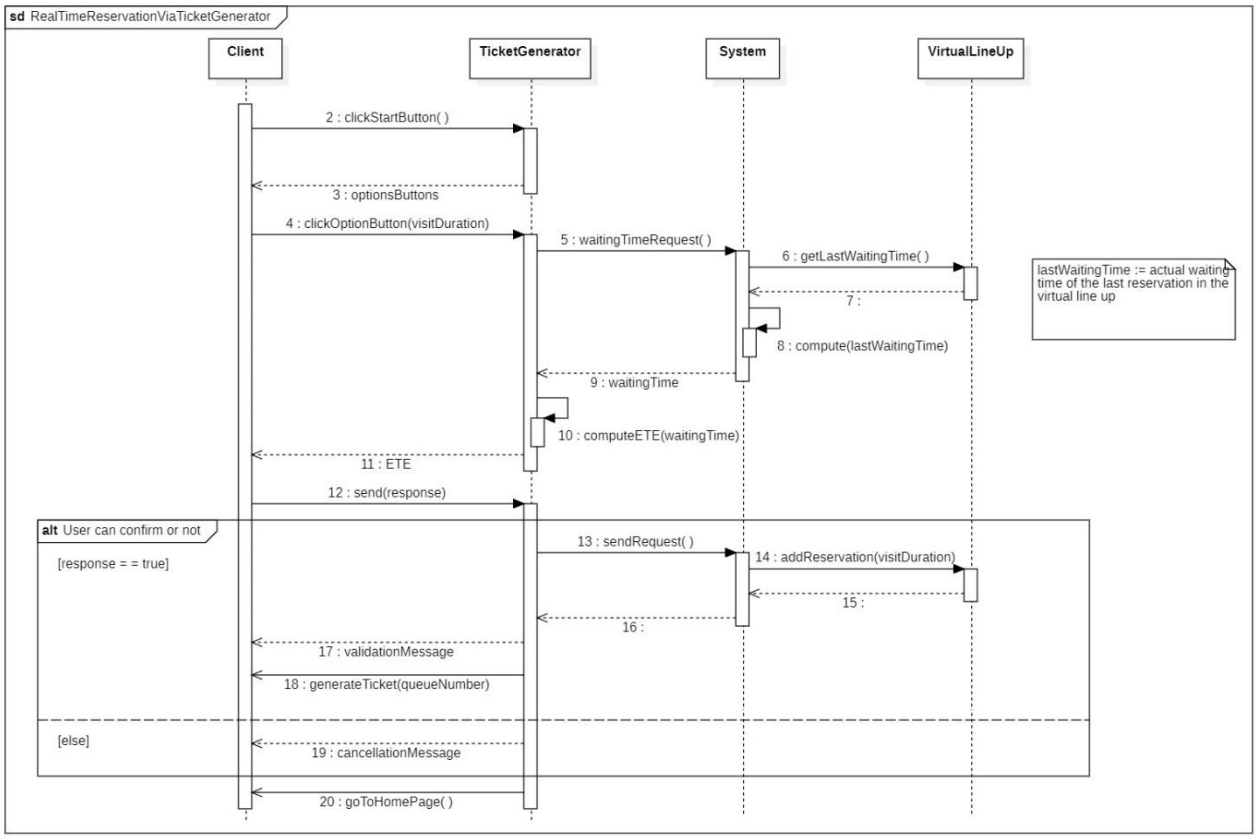
1. Client downloads and installs *CLup* on his smartphone.
2. Client opens *CLup*.
3. Client fills out a form providing his personal information.
4. Client chooses his favourite supermarket.
5. Client submits the registration request.
6. System notifies Client that registration is successful and welcomes him.

Exit Condition: Client becomes User of *CLup*. He can log into the application.

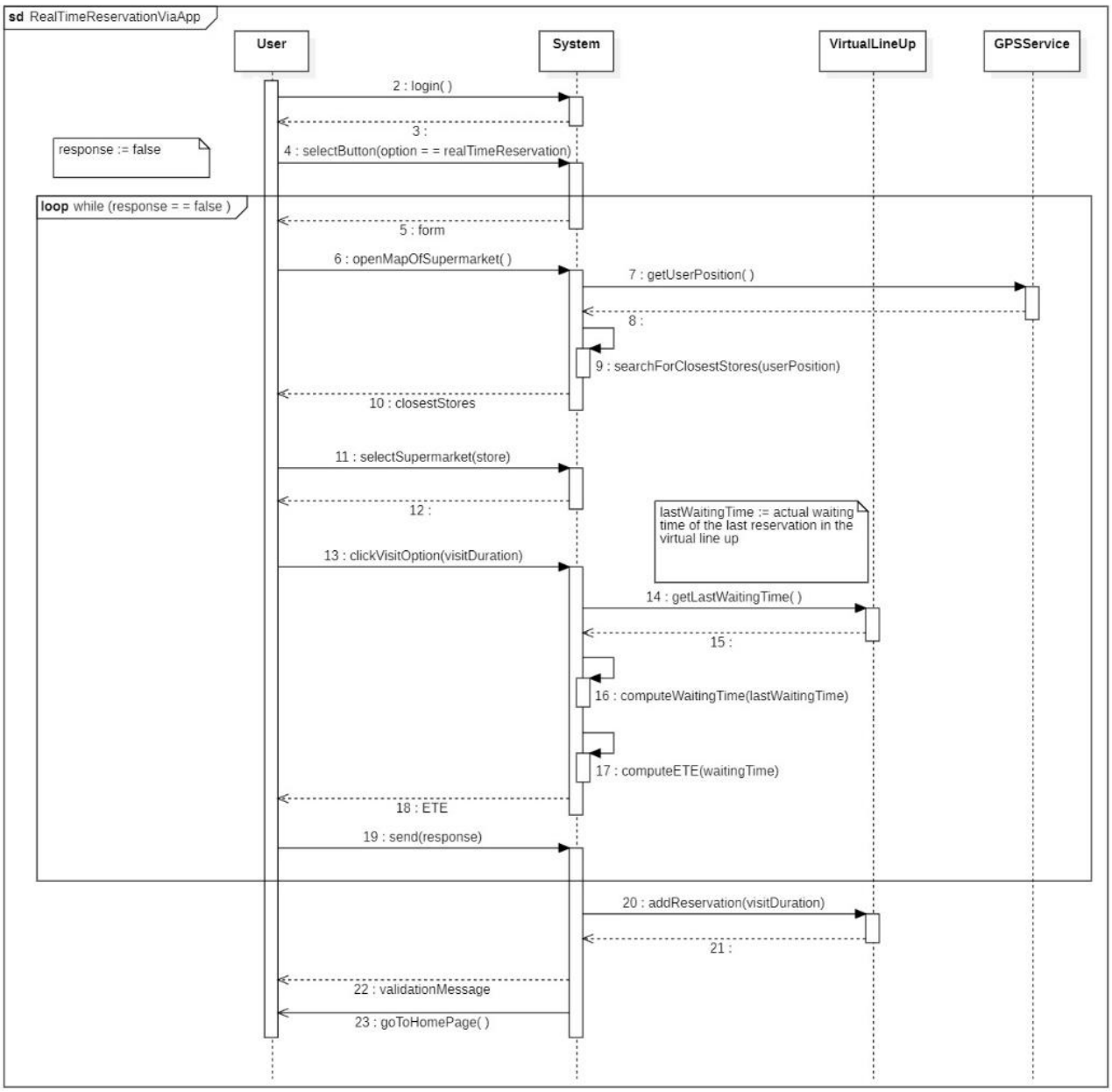
Exceptions: System does not validate the registration request because the provided email address belongs to another account. In this case System sends an error message to User and takes him back to the previous form.

3.4.3 Sequence diagrams

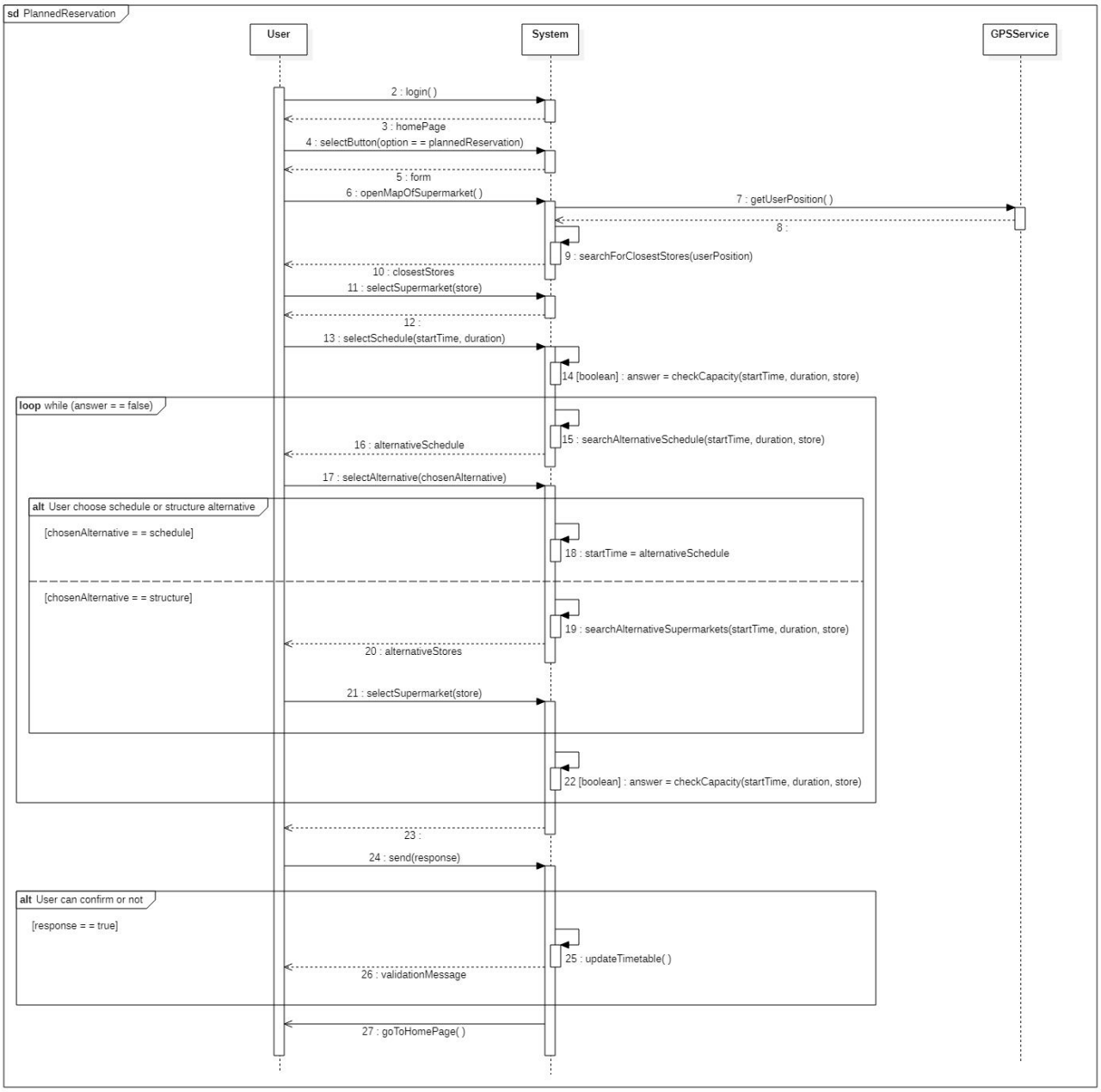
3.4.3.1 Sequence diagram 1: Make a real-time reservation via TicketGenerator



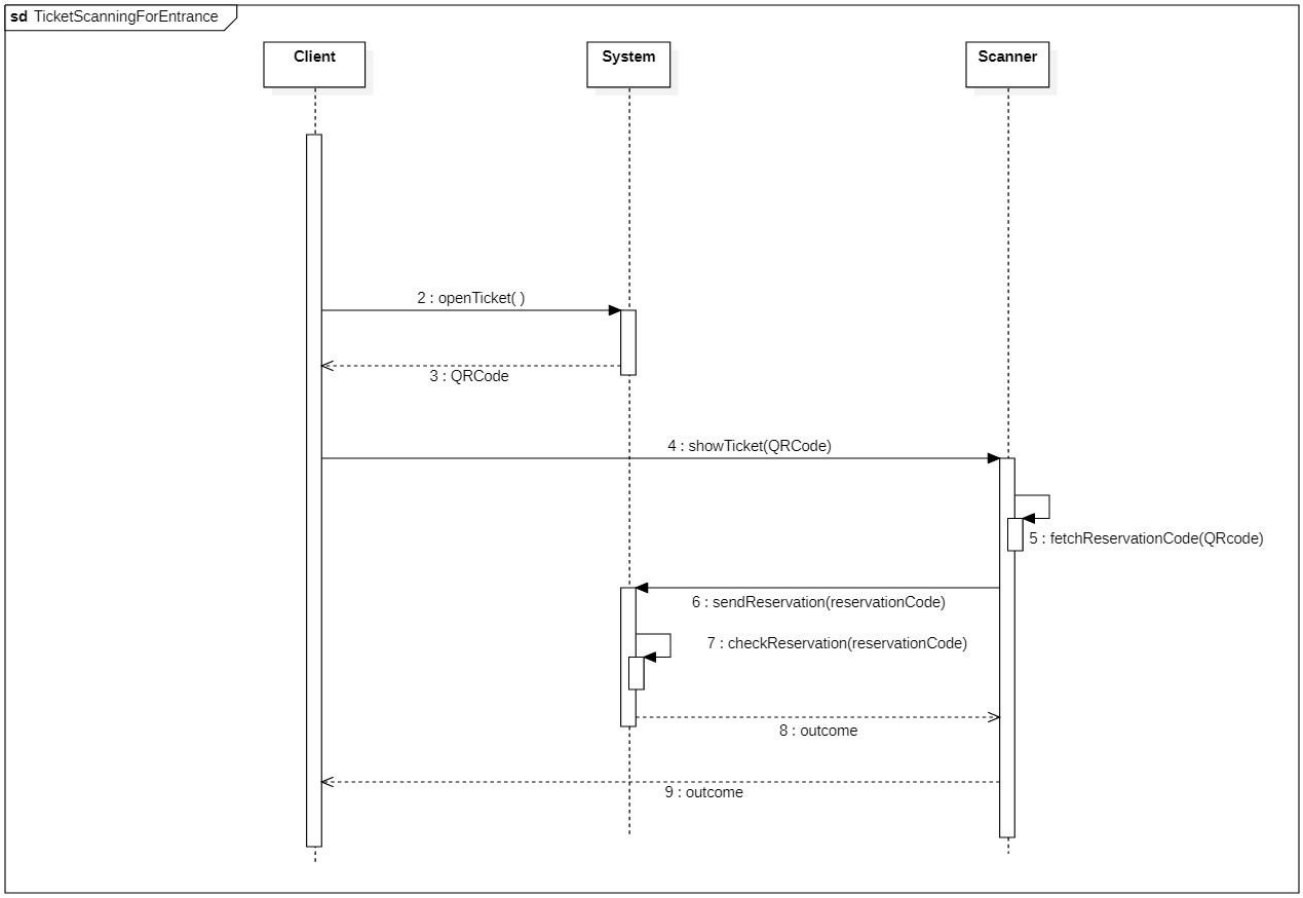
3.4.3.2 Sequence diagram 2: Make a real-time reservation via app



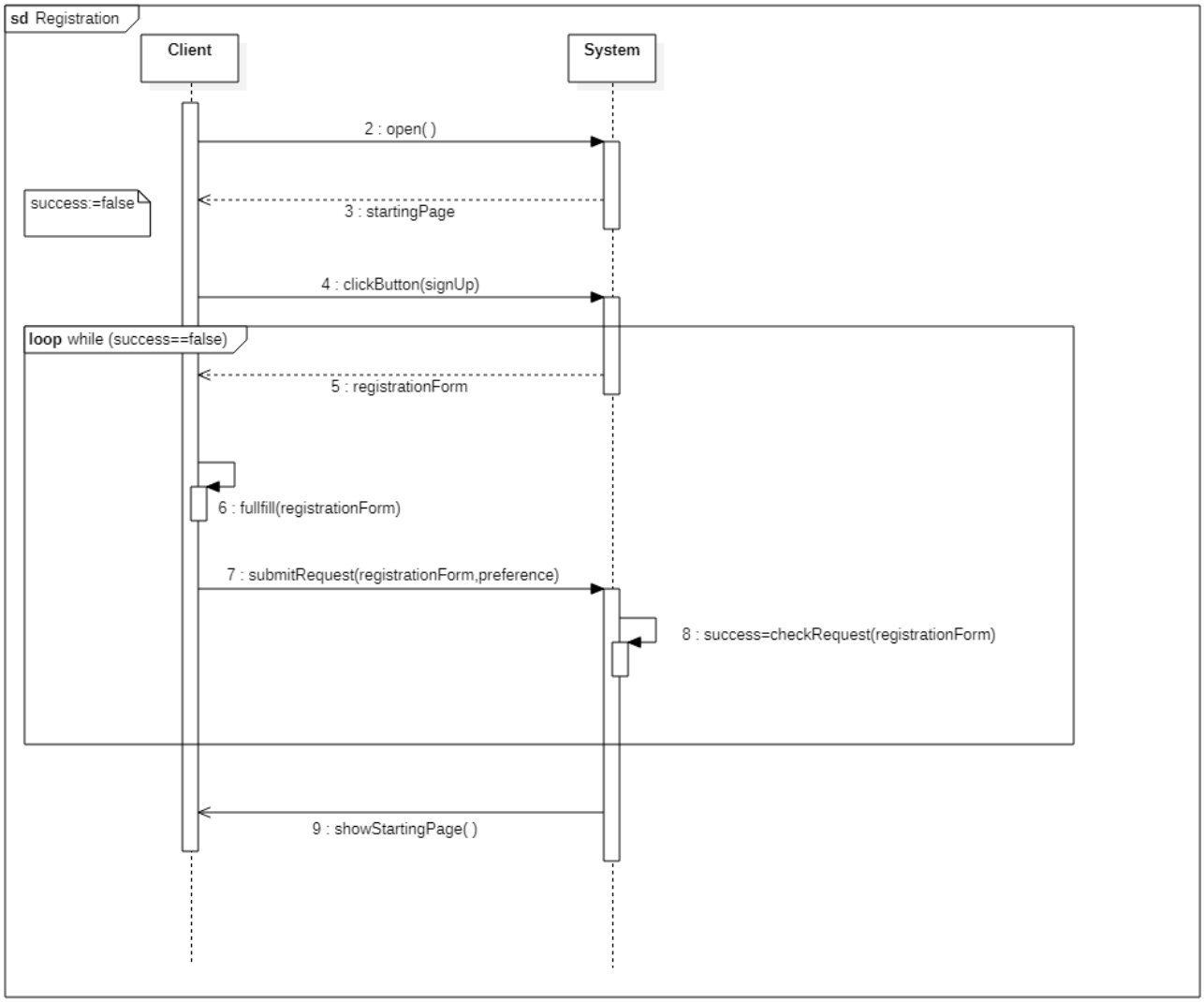
3.4.3.3 Sequence diagram 3: Make a planned reservation



3.4.3.4 Sequence diagram 4: Ticket scanning for entrance



3.4.3.5 Sequence diagram 5: Registration



3.5 Performance requirement

Assuming internet connection works properly, the system should be able to confirm a reservation request within 5 seconds. Moreover, the response of Scanner to a QR code reading should be less than 2 seconds. Finally, the interaction between Scanner and sliding doors should be as quick as possible.

3.6 Design constraint

3.6.1 Standard compliance

The software implementation must follow the requirements guidelines proposed in this document. Furthermore, comments about the code will have to be clear and focused.

3.6.2 Hardware limitations

Client limitations:

- iOS or Android Smartphone
- Internet Connection (2G/3G/4G/5G)

Supermarket limitations:

- Scanner, connected to the sliding doors, for access detection.
- Display showing the *ticket number* of the people allowed to enter.
- Touch screen device that plays the role of TicketGenerator.

Further detailed limitations related to the Supermarket will be explained in Design Document.

3.7 Software Attributes

3.7.1 Maintainability

During a pandemic period, like nowadays, restrictions imposed by governments could be varied in a very short time. These changes imply this document could be frequently revised. The system must be implemented so that all the occurring changes do not affect its general structure but only some features at code level.

3.7.2 Scalability

Due to the nationwide variable restrictions, every grocery store might have to change its access policy. The system must guarantee a high level of scalability with respect to the number of supermarkets and users that want to exploit the services.

3.7.3 Reliability

The system must ensure all the services during the supermarkets hours of work, in particular the more crowded ones. Very small deviations from this requirement will be obviously accepted. All the maintenance works could be done when supermarkets are less busy or during the non-operating hours, if possible.

3.7.4 Accuracy

The system must guarantee an accurate estimate of the *expected time of entrance*. In order to ensure this, an estimation error must be less than 7 minutes in order to handle the *queue* in an efficient way.

4 Formal analysis using Alloy

4.1 Introduction

The following section is dedicated to the modeling – and its corresponding formal analysis – of the software *CLup*.

The main goal is to show that the introduced constraints do not reciprocally conflict, in order to ensure that the system is firmly anchored on a solid and consistent basis. For this purpose, we make use of Alloy tools such as Facts and Predicates, that help us to carry out the analysis verifying that, respectively, some specific conditions are observed and the achieving of the most significant goals of the *software-to-be*.

In particular, we want to show that:

- A User is allowed to enter a given supermarket only if he is in possession of a reservation that refers to it.
- A User is allowed to enter a supermarket only if maximum capacity is not reached in that facility.
- Each supermarket must maximise the number of Users inside in each moment, in order to minimise the number of lined-up Users.

Besides, some remarks concerning the assumptions made – in order to simplify the model, preserving its significance – must be highlighted in advance:

- All reservation that ended their lifecycle (User exits the supermarket or such reservation expires) are not showed in the model as not interesting for the purpose of the analysis.
- All reservation are assumed to be associated with a User, that is we merge the reservations created via Ticket Generator with those made via app.
- Users enter the facilities following the order estabilished by the ticket numbers. This assumption has been introduced for the sake of simplicity, but does not affect the actual model consistency. In the real case all Users that own a ReadyToScan reservation can also enter randomly, since outside each store there will be shown the ticket number up to which reservation are ready to be scanned.

4.2 Alloy Code

4.2.1 SIGNATURES

```
abstract sig ReservationStatus {}
one sig LinedUp extends ReservationStatus {}
one sig Scheduled extends ReservationStatus {}
one sig ReadyToScan extends ReservationStatus {}
one sig Scanned extends ReservationStatus {}

abstract sig SupermarketStatus {}
one sig EntranceAllowed extends SupermarketStatus {}
one sig MaxCapacityReached extends SupermarketStatus {}

abstract sig Reservation {
        -- Each reservation refers to one and only one supermarket
        refersTo: one Supermarket,
        status: one ReservationStatus
}
sig RealTimeReservation extends Reservation {
        ticketNumber: Int
} {
        ticketNumber > 0
}
sig PlannedReservation extends Reservation {}

sig Supermarket {
        status: one SupermarketStatus,
        maxCapacity: Int
} {
        maxCapacity > 0
}

sig User {
        -- [R14]: Each user can have at most one active reservation at a time
        booking: lone Reservation,
        doingGroceryShopping: lone Supermarket
}
```

4.2.2 FACTS

```
/*
[R4] & [R6] : One user can enter in a store (thus do grocery shopping) if and
only if he has a reservation that refers to that supermarket
*/
fact NoGroceryShoppingWithoutScan{
        all u: User
                |       let  r = u.booking
                -- For each user we need to assure that
                |       -- He has a scanned reservation and he is doing Grocery Shopping
                        -- in the supermarket the reservation refers to, or
                        (r.status in Scanned and u.doingGroceryShopping = r.refersTo)
                                or
                        -- He has a reservation not scanned, and he is not doing
                        -- Grocery Shopping anywhere
                        (r.status not in Scanned and u.doingGroceryShopping = none)
}

/*
[R1] : Each reservation has a unique code, thus there does not exist two users
with the same reservation
*/
fact NoSameReservationForTwoUser {
        no disjoint u1, u2 : User
                |       u1.booking = u2.booking
}

/*
[R1] : Each reservation is assigned to one and only one user
*/
fact OneReservationImpliesOneUser {
        no r : Reservation
                |       r not in User.booking
}

/*
[R3] : Each reservation is assigned a ticket number that is unique for the
supermarket they refer to
*/
fact NoSameNumberForTwoTicket {
        no disjoint r1, r2 : RealTimeReservation
                |       r1.refersTo = r2.refersTo
                                and
                        r1.ticketNumber = r2.ticketNumber
}
```

```
/*
[R5] & [R7] & [R10] : Users inside the supermaket must not exceed the maximum
capacity. This means that, in every moment, it can only be scanned a number of
reservations such that maximum capacity is not exceeded.
*/
fact UsersDoingGroceryShoppingDontExceedMaxCapacity {
	all s : Supermarket
		|	let	-- Reservations referring to supermarket s
				reservationsInS = refersTo.s,
				-- Reservations that cannot be scanned yet
				linedUpReservations = status.LinedUp,
				-- Reservations ready to be scanned
				readyReservations = status.ReadyToScan,
				-- Reservations already scanned
				scannedReservations  = status.Scanned
		|	#(reservationsInS & (readyReservations + scannedReservations)) <= s.maxCapacity
				and
			(
				#(reservationsInS & (readyReservations + scannedReservations))  <
s.maxCapacity
					implies
				#(reservationsInS & linedUpReservations) = 0
			)
}

/*
[R4] : A supermarket can allow an entrance if and only if the number of users
inside is lower than the maximum capacity
than
*/
fact SupermarketStatusCondition {
	all s : Supermarket
		|	let	-- Reservations referring to supermarket s
				reservationsInS = refersTo.s,
				-- Reservations already scanned
				scannedReservations  = status.Scanned
		|	s.maxCapacity = #(reservationsInS & scannedReservations)
				iff
			s.status in MaxCapacityReached
}
```

```
/*
[R3] : Each ticket number represents a position in the virtual line-up, thus also an
order of entering. Therefore the ticket number of a reservation ready to be
scanned must be greater then the one of a reservation already scanned,
and so on...
*/
fact TicketAreOrdered {
	all disj r1, r2 : RealTimeReservation
		|	-- Lined up ticket# > Ready to Scan ticket#
			(
				r1.refersTo = r2.refersTo and r1.status in LinedUp and r2.status in ReadyToScan
					implies
				r1.ticketNumber > r2.ticketNumber
			)
				and
			-- Ready to Scan ticket# > Scanned ticket#
			(
				r1.refersTo = r2.refersTo and r1.status in ReadyToScan and r2.status in Scanned
					implies
				r1.ticketNumber > r2.ticketNumber
			)
				and
			-- Lined up ticket# > Scanned ticket#
			(
				r1.refersTo = r2.refersTo and r1.status in LinedUp and r2.status in Scanned
					implies
				r1.ticketNumber > r2.ticketNumber
			)
}

/*
Constraints added in order to avoid incorrect ReservationStatus assignments
*/
fact RealTimeCannotBeScheduled {
	no r : RealTimeReservation
		|	r.status in Scheduled
}
fact PlannedCannotBeLinedUpOrReadyToScan {
	no r : PlannedReservation
		|	r.status in LinedUp
				or
			r.status in ReadyToScan
}
```

4.2.3 PREDICATES

```
/*
[G1] : A user is allowed to enter a supermarket only if he has a reservation that
refers to that supermarket
*/
pred AllowEntranceOnlyIfUserGotReservation {
        all u: User
                |       u.doingGroceryShopping != none
                                and
                        u.doingGroceryShopping in Supermarket
                                implies
                        u.booking in Reservation
}

/*
[G2] : A user is allowed to enter a supermarket only if maximum capacity is not
reached in the supermarket
*/
pred AllowEntranceOnlyIfMaxCapacityIsNotReached{
        all s: Supermarket
                |       let     -- Reservations referring to supermarket s
                                reservationsInS = refersTo.s,
                                -- Reservations ready to be scanned
                                readyReservations = status.ReadyToScan,
                                -- Users allowed to enter in supermarket s
                                usersAllowedToEnterInS = booking.(readyReservations & reservationsInS),
                                -- Users already inside supermarket s
                                usersInsideS = doingGroceryShopping.s
                |       #usersAllowedToEnterInS <= s.maxCapacity - #usersInsideS
}

/*
[G3] : Each supermarket must minimise the number of lined-up Users
*/
pred MinimiseLineUpUsers{
        all s: Supermarket
                |       let     -- Reservations referring to supermarket s
                                reservationsInS = refersTo.s,
                                -- Reservations that cannot be scanned yet
                                linedUpReservations = status.LinedUp,
                                -- Reservations ready to be scanned
                                readyReservations = status.ReadyToScan,
                                -- Users waiting to enter in supermarket s
                                usersForbiddenToEnterInS = booking.(linedUpReservations & reservationsInS),
                                -- Users allowed to enter in supermarket s
                                usersAllowedToEnterInS = booking.(readyReservations & reservationsInS),
                                -- Users already inside supermarket s
                                usersInsideS = doingGroceryShopping.s
                |       -- Presence of people forbidden to enter in supermarket s
                        usersForbiddenToEnterInS != none
                                implies
                        -- Implies that supermarket s has reached max capacity
                        #(usersInsideS + usersAllowedToEnterInS) = s.maxCapacity
}
```

4.2.4 RUNS

```
run AllowEntranceOnlyIfMaxCapacityIsNotReached for 12 but exactly 2 Supermarket, exactly 12 Reservation
run AllowEntranceOnlyIfUserGotReservation for 12 but exactly 2 Supermarket, exactly 12 Reservation
run MinimiseLineUpUsers for 12 but exactly 2 Supermarket, exactly 12 Reservation
```

4.3 Results

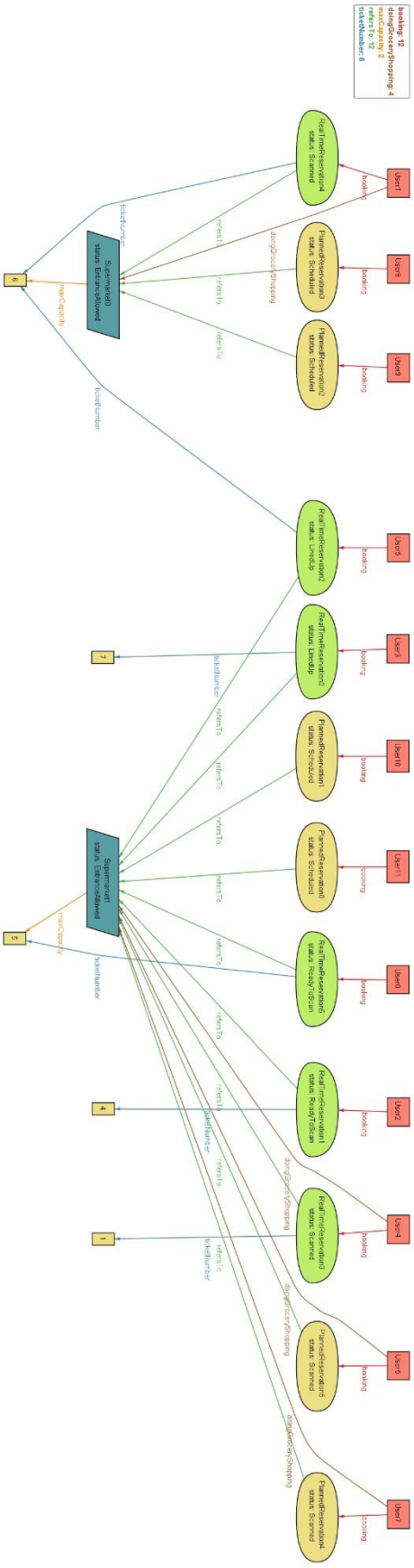
```
Executing "Run AllowEntranceOnlyIfMaxCapacityIsNotReached for 12 but exactly 2 Supermarket, exactly 12 Reservation"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  18807 vars. 648 primary vars. 42065 clauses. 53ms.
  Instance found. Predicate is consistent. 157ms.

Executing "Run AllowEntranceOnlyIfUserGotReservation for 12 but exactly 2 Supermarket, exactly 12 Reservation"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  17829 vars. 648 primary vars. 38259 clauses. 86ms.
  Instance found. Predicate is consistent. 122ms.

Executing "Run MinimiseLineUpUsers for 12 but exactly 2 Supermarket, exactly 12 Reservation"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  18755 vars. 648 primary vars. 41131 clauses. 72ms.
  Instance found. Predicate is consistent. 133ms.

3 commands were executed. The results are:
  #1: Instance found. AllowEntranceOnlyIfMaxCapacityIsNotReached is consistent.
  #2: Instance found. AllowEntranceOnlyIfUserGotReservation is consistent.
  #3: Instance found. MinimiseLineUpUsers is consistent.
```

4.4 Generated world



5 Effort spent

Immordino Alessandro

Goals and introduction	2.5
Domain assumptions	6.5
Functional requirements	12.5
Alloy	18
Scenarios	5
Use cases	2.5
Diagrams	9
Other	20.5
Total	76.5

Polvanesi Giacomo

Goals and introduction	7
Domain assumptions	2
Functional requirements	9.5
Alloy	16
Scenarios	6.5
Use cases	4.5
Diagrams	4
Other	21.5
Total	71

Pala Riccardo

Goals and introduction	5
Domain assumptions	4.5
Functional requirements	9.5
Alloy	15.5
Scenarios	7.5
Use cases	2
Diagrams	8.5
Other	20
Total	72.5