Software Engineering II 2020/2021

December 2020

Requirements Analysis and Specification Document

CLup: Customers Line-up Software

Version: 1

Authors:

R. Cedeno, D. Cumming, A. Pugliese

Contents

1. INTRODUCTION

1.1 Purpose

The need of avoiding physical interactions in order to prevent the spreading of the COVID-19 disease has put a hard constraint on the businesses that are based on in-person shopping, since there are safety restrictions on how many people can be inside of a certain building while there must be a distance of at least one meter between them. In order to address these constraints, the requirements and specifications of a software project that manages the entering of the customers to the stores will be elaborated in this document.

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

As a first step for developing the requirement engineering, the purpose of the project is compiled by the following software goals:

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

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

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

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

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

1.2 Scope

To describe the requirement engineering approach to design a software that fulfil the goals, a brief explanation of the problem modelling is presented in this section. As the application to be developed will be situated in the context of the stores' supply and customers' demand, it is useful to model an abstraction of the set of occurrences on this physical environment in which the software is going to act, defining this as the **world phenomena.** On the world phenomena, there are two relevant classes: the software goals (described in the previous section) and the **domain assumptions**, defined as the set of phenomena from the physical world that are reasonably taken for granted in order to establish

the connection with reality (further details about the assumptions in section 2.4). On the other hand, the software and hardware system, denominated as the **machine**, will be influencing a certain part of the world with the purpose of bringing the requirements into completion.

The software engineering developed in this project will elaborate a set of requirements that must be fulfilled through phenomena that will happen both in the world and in the machine at the same time, which are denominated as **shared phenomena** (further details in sections 2.2 and 3). These phenomena can be controlled either by the world or the machine. In Figure 1, an abstract representation of the different types of phenomena is illustrated. The definition of the software requirements are designed in such a way that, jointly with the domain assumptions, will satisfy the software goals specified in the previous section. This means that the domain assumptions are the conjectures obtained from the world phenomena that mutually complements the requirements fulfilled by the machine within the shared region.

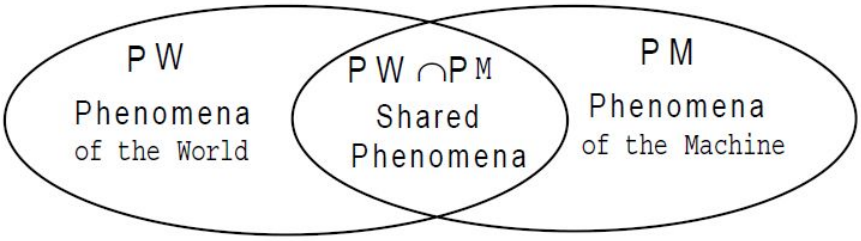

Figure 1: Boolean representation of the World and Machine Phenomena Model.

(taken from M. Jackson, 1996)

1.3 Definitions, Acronyms, Abbreviations

1. As Soon As Possible (ASAP)
2. Preferential Customers: people with disabilities, elders or pregnant women.
3. Recovery point objective (RPO): is the age of files that must be recovered from backup storage for normal operations to resume if a computer, system, or network goes down as a result of a hardware, program, or communications failure.
4. Recovery time objective (RTO): represents the amount of time that an application can be down and not result in significant damage to a business.
5. Mean time to repair (MTTR): encompasses all the operations necessary to detect and repair a system failure. For a software module, it can be defined as the time required to restart the application after a failure detection.
6. Mean time to failure (MTTF): average amount of time a non-repairable asset operates before it fails.

7. Secure by design: a product that has been designed secure from the foundation.
8. The EU General Data Protection Regulation (GDPR): defines the way that organizations and businesses should collect, control and process personal data. Its goal is to ensure that there is a balanced relationship between the rights of the individual and those who process their personal data, ensuring that the needs of the digital economy in Europe are enhanced through the seamless transfer of personal information in a protective environment.

1.4 Overview

This document provides the requirement engineering and specification of the software to be developed presenting the information in different layers, from the big picture of the software to the technical details. In Section 2, there is a general description of the software domain, as well as an elaboration about the expected functionalities, assumptions and dependencies. In Section 3, there are technical details and directions about the software requirements, aimed to be used by the developers as input for the design process. Finally, in Section 4 there is a logical modelling that supports the completeness of the requirements and assumptions to accomplish the project purposes.

2. OVERALL DESCRIPTION

2.1 Product perspective

In this section there is a description of the main properties and components of the software domain, which provides a general understanding of the operations carried out by the application system.

2.1.1 Class diagram

The main application domain entities and its main attributes and operations are illustrated in Figure 2. The User class is constituted with its registration data and a boolean flag to specify if it is a supermarket user or else it is a customer. This is a simple class diagram that describes the structure of the application where the main roles are the users and the supermarkets. The essential functionality is the booking that can be made by a user in a specific supermarket.

The Booking class represents lining-up or booking instances that the user may have. The Supermarket class is identified by the store's information and its state is represented by a queue which contains all the lining-up or booking information to manage the entering turns. Each supermarket has a set of Sections with a certain identifier and state regarding the amount of people that are shopping there. The TimeTable class was defined to manage the opening and closing hours for the supermarkets in each date in a simple way, e.g., assigning the same time table to all the supermarkets of the same brand or setting a different schedule in a certain date.

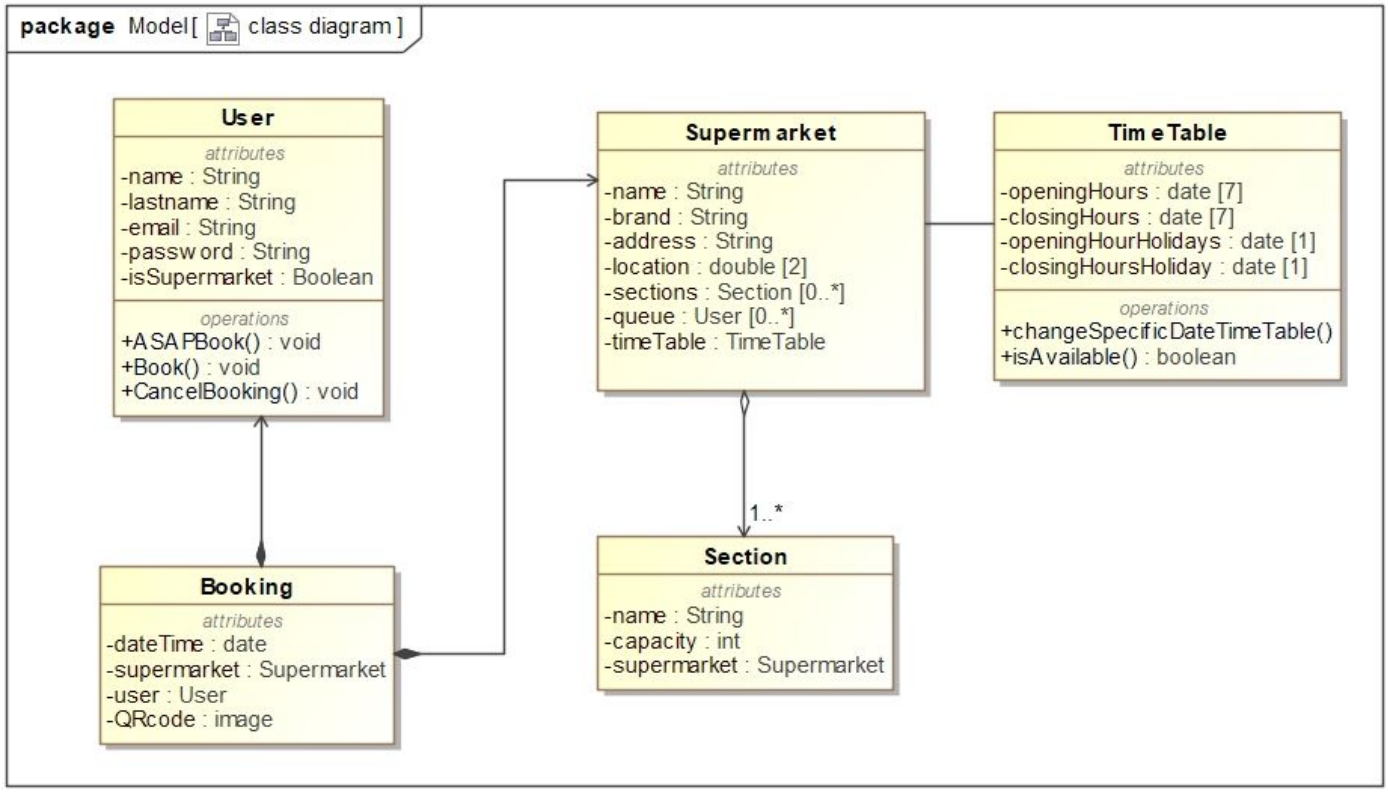

Figure 2: Class diagram of the software product

2.1.2 Statecharts

Statecharts or state machines are used to specify the behavior of each part of the designed system using final state transitions. In *Figure 3*, we can find the login statechart which shows how the application views are changing depending on the different authentication transitions that can arise.

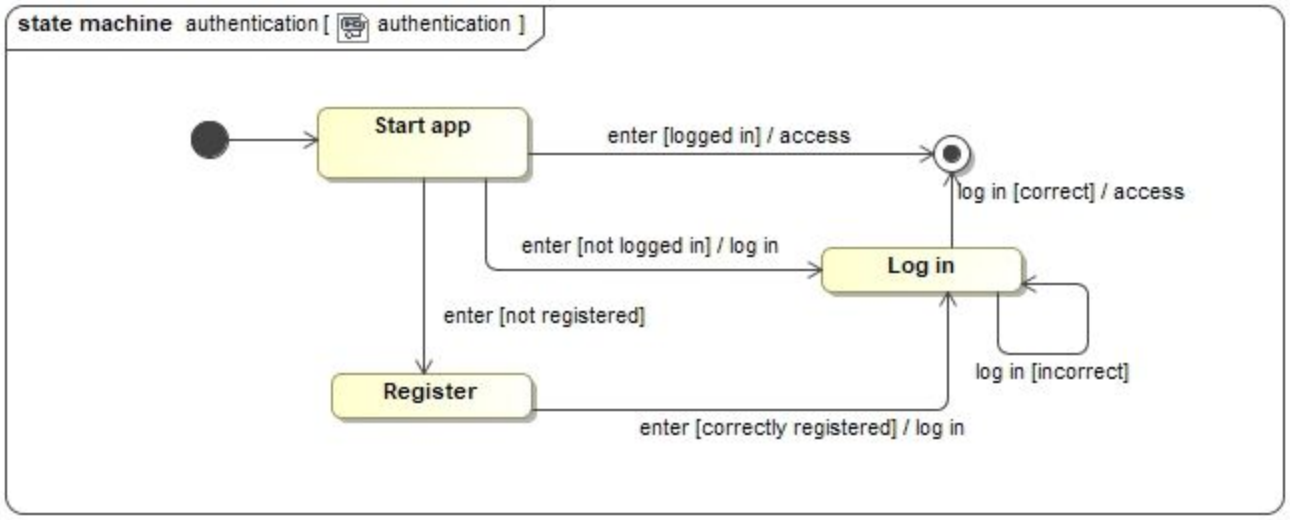

Figure 3: User Login statechart

The main statechart in *Figure 4* is the one that describes the behavior of the application when a ticket is requested. It shows the different options the user can choose when requesting a ticket, e.g., ASAP in any nearby supermarket, queue in a specific supermarket or reserve a time slot. We can also see the transitions from the map view to itself when different filters are applied considerent distinct parameters.

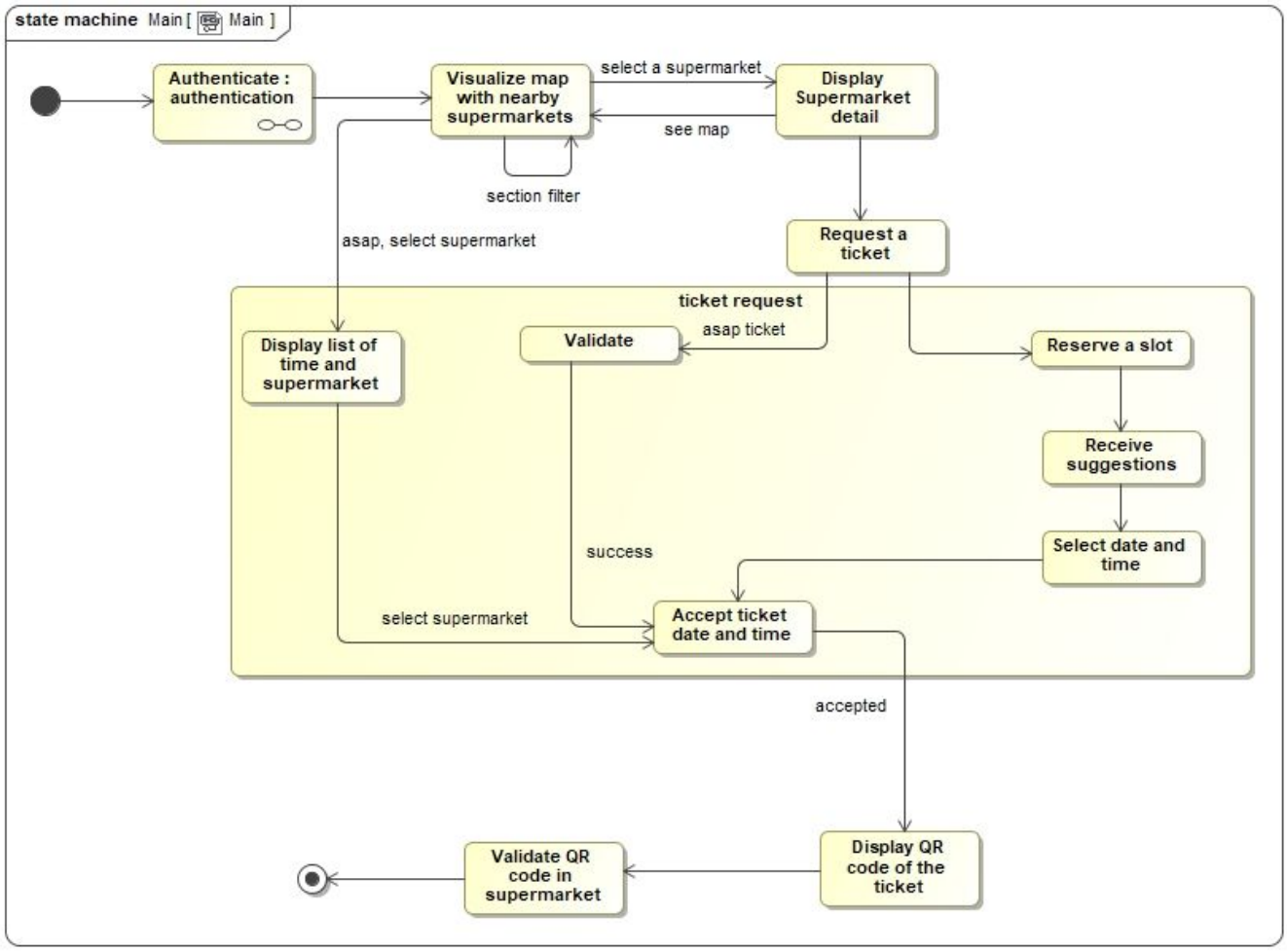

Figure 4: App ticket request statechart

Finally, the physical ticket request statechart in *Figure 5* describes the behaviour of the application when running in a device placed in the supermarket managed by its staff. It can be stated that in place tickets are just for the current queue, i.e., reservations cannot be requested.

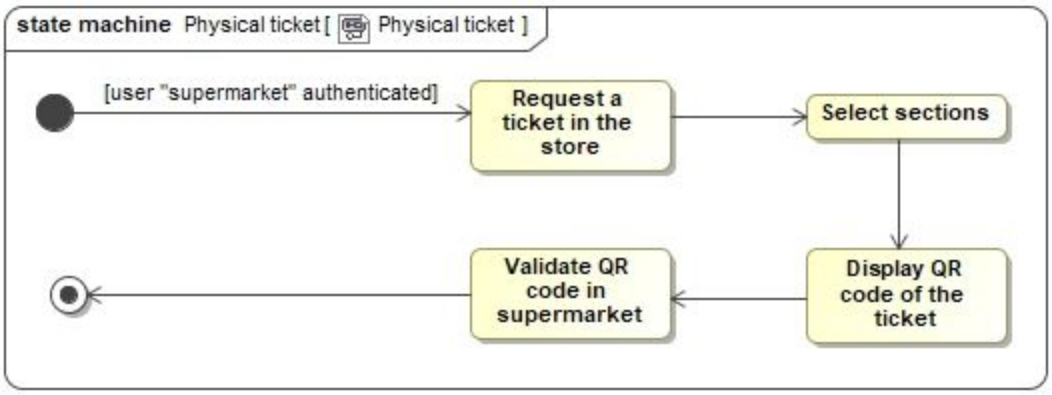

Figure 5: Physical ticket request statechart

2.2 Product functions

In order to represent the effect of the software system in the world, the shared phenomena (defined in section 1.2) are portrayed by the software requirements. This subset of phenomena is constituted by the operations carried out by the software system on the real world by means of functionalities that fulfill requirements needed to bring the goal into completion. The following lists specify each one of these phenomena as a functionality performed by the software to satisfy the requirements. The technical details of the requirements are explained in Section 3.

2.2.1 Functionalities controlled by the world

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

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

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

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

2.2.2 Functionalities controlled by the machine

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

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

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

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

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

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

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

2.3 User characteristics

The users that play a role in this application are the following:

2.3.1 Customer

A person who accesses the application through his/her smartphone and authenticates to use the application services. These services will consist in viewing the waiting time in nearby supermarkets, requesting a ticket to go to any of these supermarkets ASAP, booking an appointment to shop at a certain time slot and receiving suggestions according to time, location and/ or preferences.

2.3.2 Supermarket hub

This device placed in the supermarket will provide physical tickets to people that are not able to access the application. It must be stated that only ASAP tickets may be requested, i.e., no bookings can be made. Furthermore, the hub will be controlled by supermarket staff.

2.4. Assumptions, dependencies and constraints

As explained in Section 1.2, the accomplishment of requirements is not enough to guarantee the satisfiability of the goals, since they rely also on the assumptions made on the world domain. Thus, the disjunction between the requirements and the domain assumption entails the software goals. It is important to note that if the assumptions do not hold, then there is no completeness for achieving the software purpose, so it is important to assume pertinent, clear and realistic situations that will be used to logically prove the goals in the formal modelling in Section 4. In particular, the following domain assumptions are defined for the requirements engineering:

D1. There is a virus that is spread if people are close to each other. If the virus is spread, then there is an unsafe condition. The in-person shopping will be safe as long as customers do not enter a full section.

D2. In order to guarantee safety, each store has a maximum capacity of customers that can be inside the store or a subsection of it.

D3. Customers have the intention of entering the store to buy groceries ASAP or on another time slot.

D4. Stores have subsections in which different product categories are placed.

D5. Customers that want to buy groceries ASAP, will line-up in order to wait for their turn to enter the store.

D6. Customers that want to buy groceries in a certain time slot, will book a number for entering a store at that time.

D7. Customers know the store sections where the products that they want to buy are located. They request to line-up or book on the sections in which they want to buy.

D8. The Stores will always have a slack capacity aimed for people with disability. It never will happen that more people with disabilities arrive at a store at the same time.

D9. Customers will have an unique account with their respective email address.

D10. Each store has a doorman that will be able to let preferential customers get inside. The stores will always count with a certain extra capacity for preferential customers.

D11. When customers scan a QR code and this is approved by the system, they enter the store immediately.

3. SPECIFIC REQUIREMENTS

3.1. External Interface Requirements

3.1.1 User Interfaces

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

3.1.1.1 Log-in interface

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

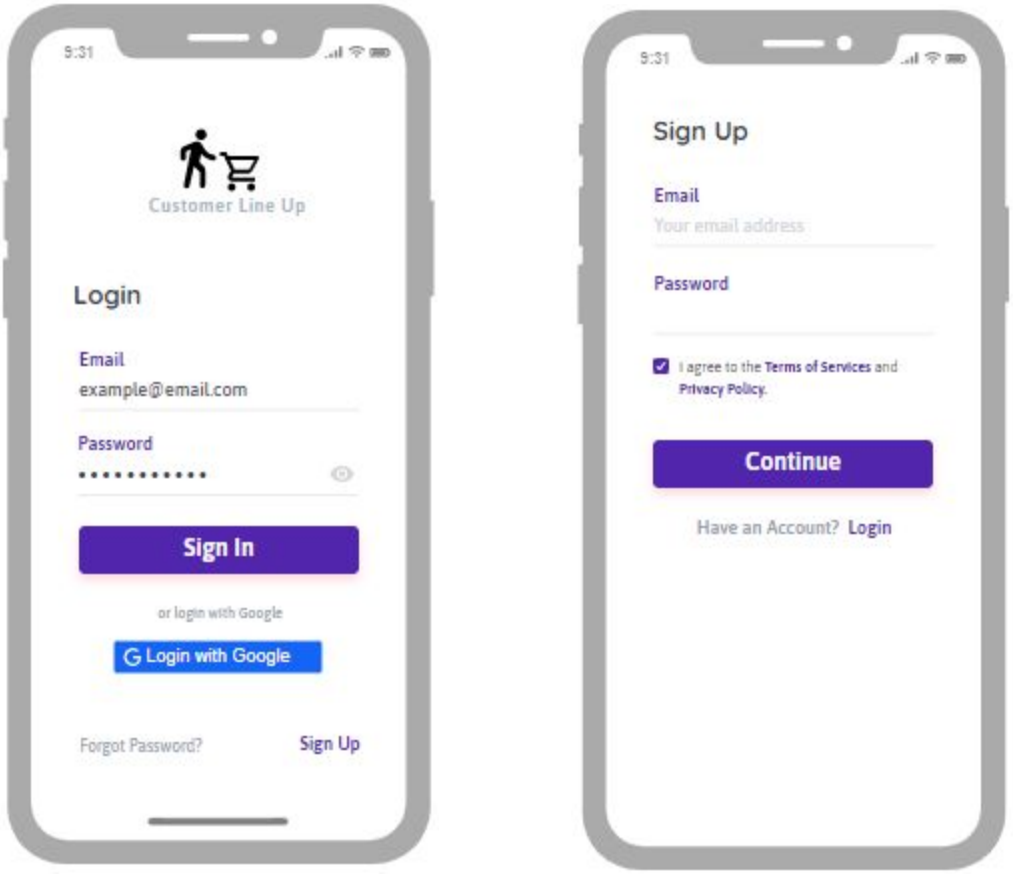

Figure 6: Login and sign up user interfaces

When the users enter the application, in the first instance they will find a typical and well-known log-in page with sign-in and forgot-my-password options. These mockups are related to the state diagram in Figure 2.

3.1.1.2 Map interface

When the users successfully log in, they enter the core of the interface which is based on a map service, where different supermarket locations are explorable by means of map snippets such as drag or zoom to personalize the display. Jointly with the map interface, a section preferences tool will be available in order to make the user able to discern between supermarkets estimated waiting times by means of semaphore symbology (black color means closed store) as shown in Figure 7. The supermarkets' waiting time color will be updated when the users select a certain preference regarding the section in which they want to buy in. Furthermore, a special button for lining-up to a supermarket ASAP is available.

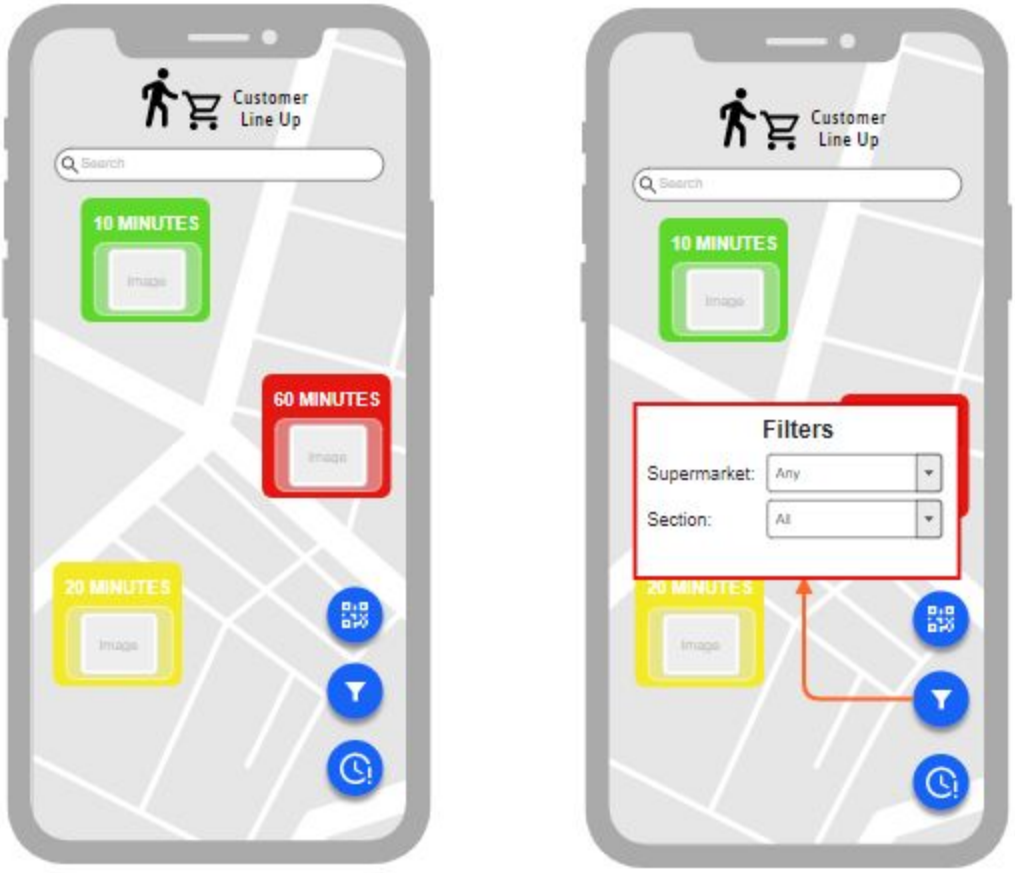

Figure 7: Map and filter user interfaces

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

3.1.1.3 ASAP supermarket list interface

When the users click the ASAP lining-up button on the map interface, it redirects to a list of the most convenient options in terms of estimated waiting time and location as shown in Figure 8. Clicking on any option will redirect to the line-up options of the clicked supermarket where users will be able to confirm the lining-up.

Figure 8: List of supermarkets suggestions for ASAP line-up user interface

3.1.1.4 Line-up options interface

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

Figure 9: Supermarket pop-up user interface

3.1.1.5 Booking interface

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

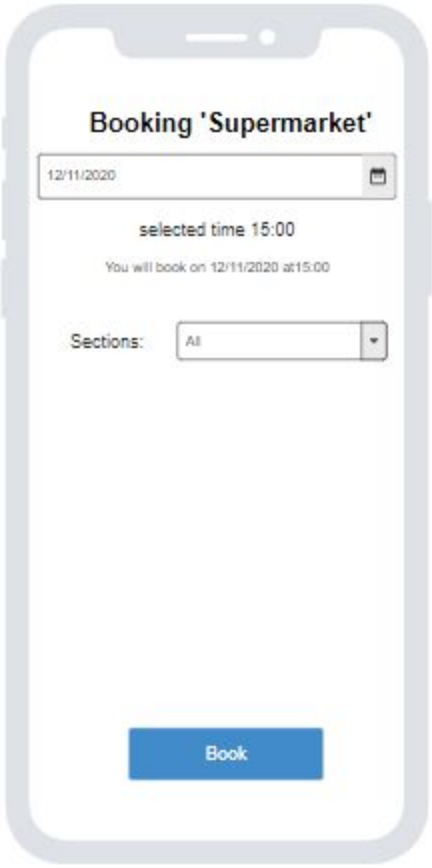

Figure 10: Booking in a specific supermarket user interface

3.1.1.6 QR code interface

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

Figure 11: QR code user interfaces

3.1.2 Hardware Interfaces

The system will communicate with the users by a mobile web application, so the hardware interface for customers is a standard smartphone. On the other hand, the supermarkets will have ticket machines (able to print QR codes for the clients) and QR scanners, which are connected with the application server of the system.

3.1.3 Software Interfaces

Since this is a mobile application, it has to be able to run on the latest versions of the most used mobile browsers: Chrome 86, Safari 14, Edge 86, Firefox 82 and Opera 72.

On the other hand, the ticket machine must have a special software that makes the user able to line-up in the queue as specified in UC7.

3.1.4 Communication Interfaces

This system uses the internet to communicate between the users' mobile devices or supermarkets' machines (ticket generator and QR scanner) and the application server.

The QR scanner must be communicated with the supermarket's door so it is opened when the QR code is accepted or kept closed when rejected.

3.2. Functional Requirements

In Figure 12, the diagram illustrates the use cases that characterize the software functionalities and how they are related among each other. There are 2 types of actors that operate over a total of 7 use cases of which 3 of them depend on the Login use case. The use cases from the supermarket are independent from the ones applicable to the remote user. It is important to highlight that the Registration depends on the Google login external service.

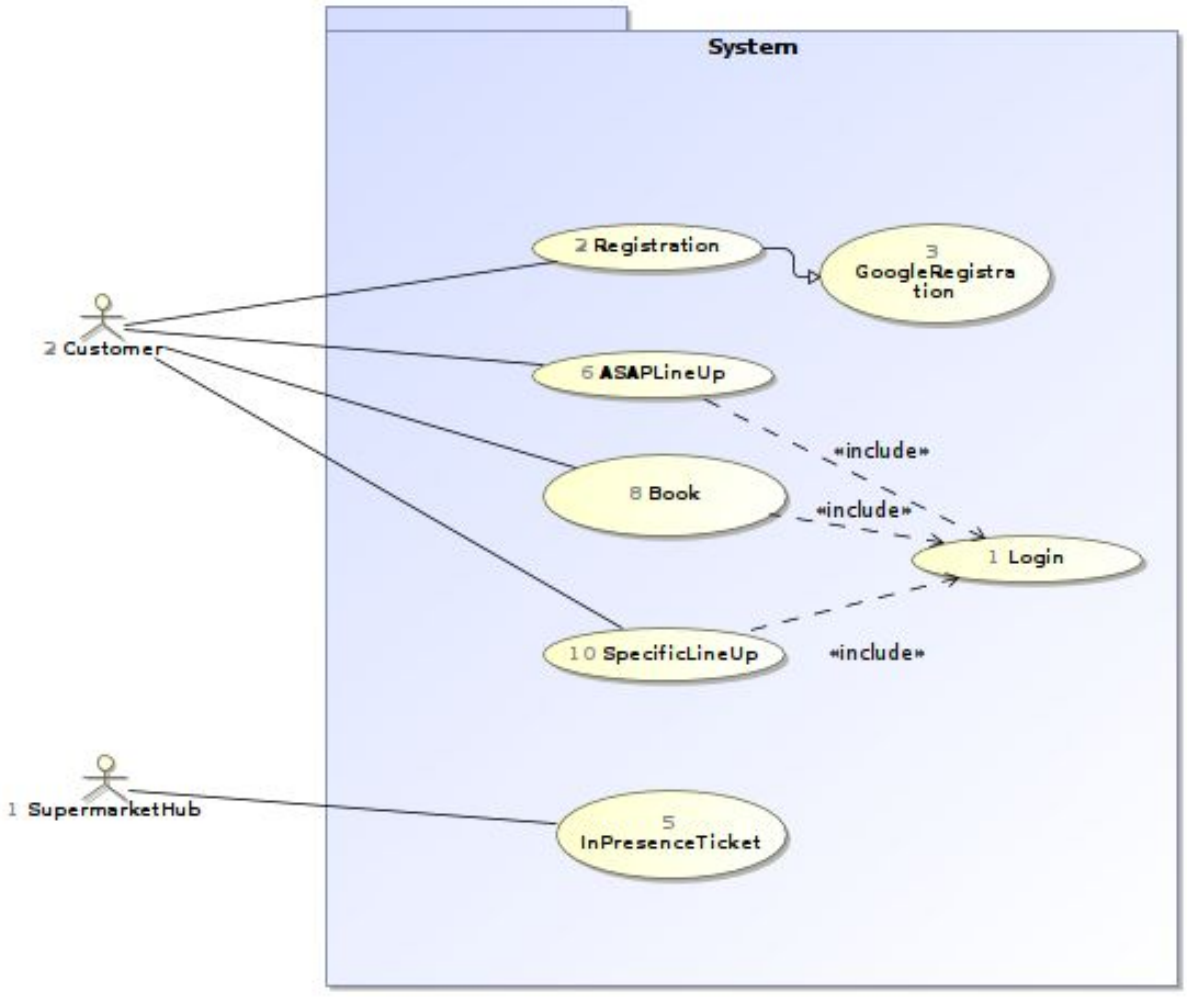

Figure 12: Use case diagram

Further details about UCs are explained in the following sections.

3.2.1 Use Case 1: Login

Name	Login
Actor	User
Entry Condition	The User has entered the application, which displays the login main page
Event Flow	1. The user enters the username and password 2. The application queries the user's data in the User DB 3. The User DB return the outcome of the query 4. In case the user input exists, the application accepts the login.
Exit Condition	The user is logged in
Exception	The user input is not in the User DB, then the login is rejected.
Special Req	User must be registered in the application

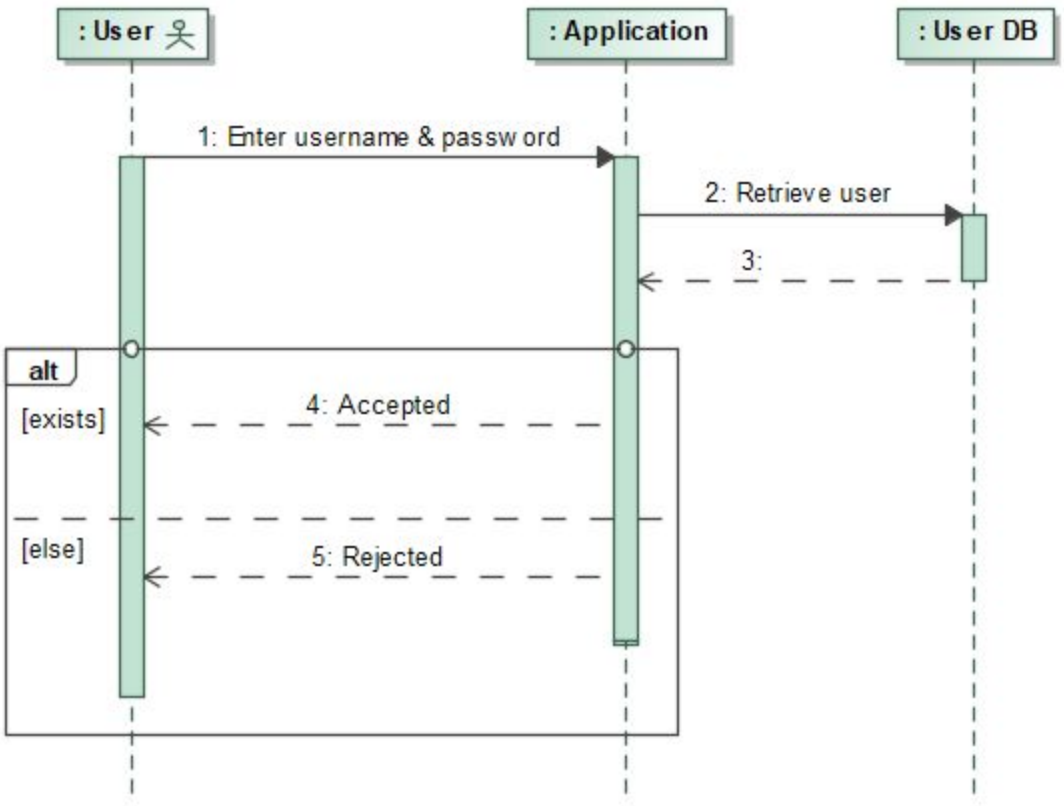

Figure 13: UC1 Login Sequence Diagram

3.2.2 Use Case 2: Registration

Name	Registration
Actor	User
Entry Condition	The User has entered the application, which displays the login main page
Event Flow	1. The user clicks on the 'Register' link 2. The Application returns a page with the form to be filled by the user 3. The user fills the required information and submits. 4. The application verifies that the form is filled completely 5. The application queries the User DB in order to check if the username and email are already registered. 6. The User DB returns the query outcome. 7. The application sends a verification link to the user's email 8. The user clicks on the verification link in the received email 9. The user's registration data is inserted in the User DB 10. The system confirms to the user that the registration was done.
Exit Condition	The user signs into the application with the registered account
Exception	● The user doesn't receive verification email ● The username or email is already registered
Special Req	User must have an email account

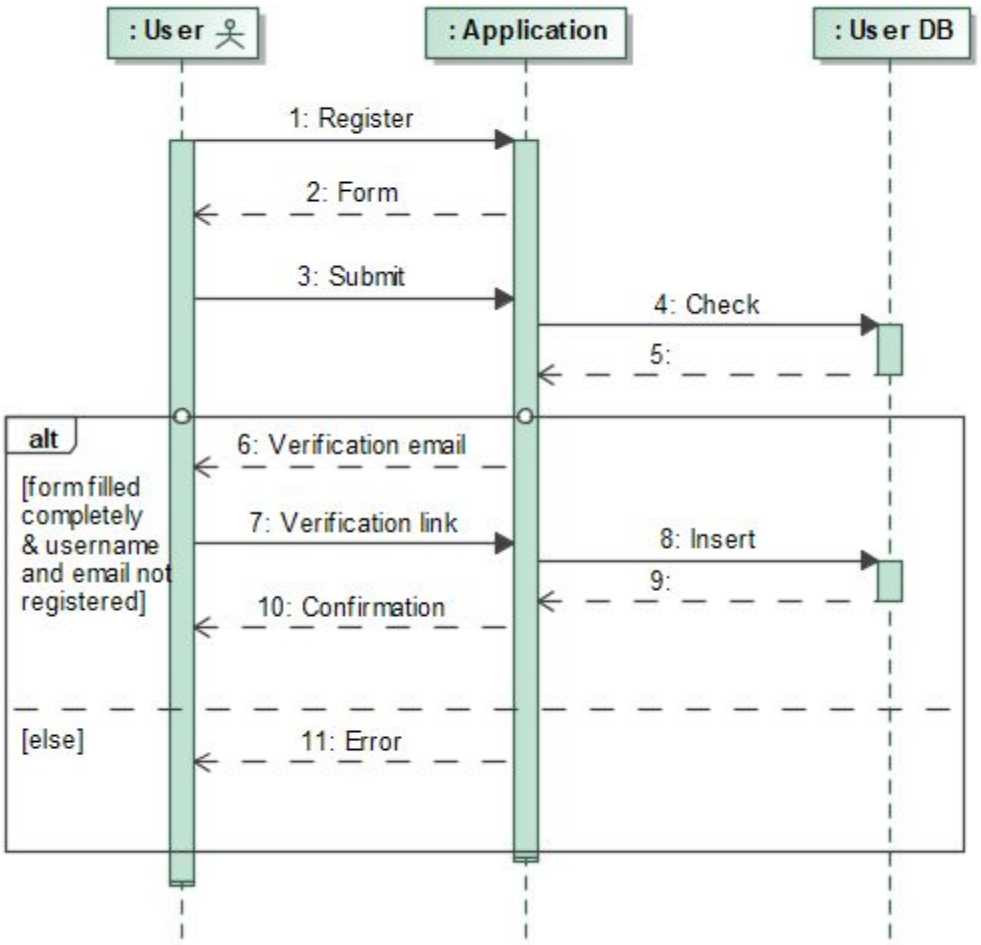

Figure 14: UC2 Registration Sequence Diagram

3.2.3 Use Case 3: Google Registration

Name	Google Registration
Actor	User
Entry Condition	The User has entered the application, which displays the login main page
Event Flow	1. The user selects the “Register using Google” 2. The application queries to the API the embedded Google sign-in. 3. The Google API returns the embedded sign-in widget. 4. The Google sign-in widget is rendered in the page. 5. The user clicks on his/her Google account. 6. The authentication procedure is carried out by the Google API. 7. The confirmation is displayed by the application.
Exit Condition	The application signs in the user with its google account.
Exception	The user is not providing correct credentials for Google or is not

	registered.
Special Req	The user must be registered in Google

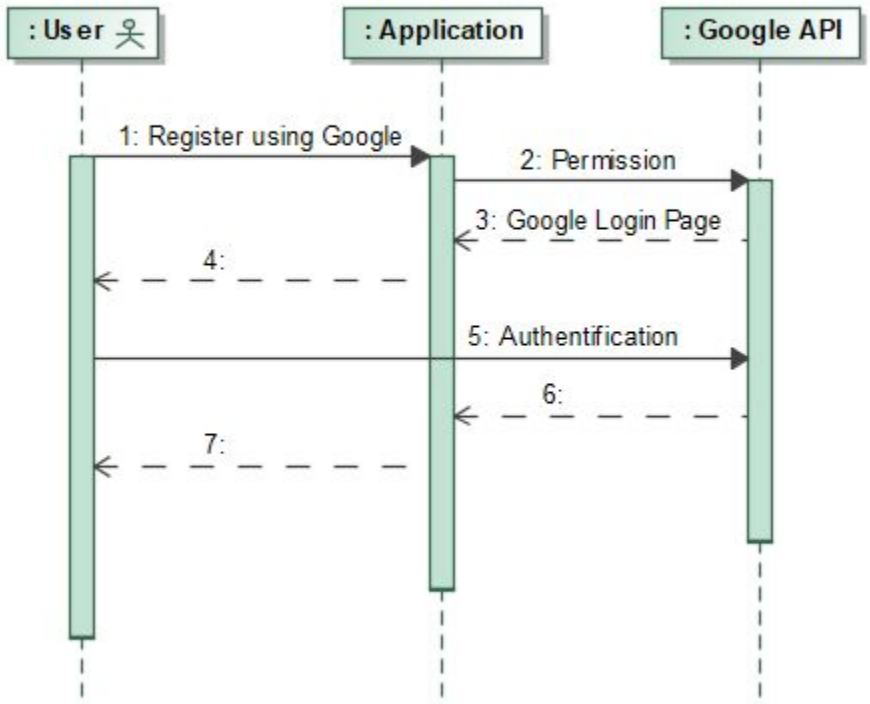

Figure 15: UC3 Google Registration Sequence Diagram

3.2.4 Use Case 4: Specific Line-Up

Name	Specific Line-Up
Actor	User
Entry Condition	The User has entered the application, which displays the login main page
Event Flow	1. The user logs into the application 2. The application requests the queue states of the supermarkets around the user's position. 3. The application shows the map page, which display different supermarkets options around and their availability 4. The user selects the desired sections to visit or select them all (default option). 5. The application displays supermarkets' availability on the map depending on the section selection. 6. The user selects in the map the supermarket to buy in.

	7. The application displays the pop-up of the selected supermarkets with the line-up and book options to pick a number. 8. The user selects the option to line-up 9. The application does a request to update the state of the queue manager by inserting the number of the user. 10. The queue manager returns the confirmation of the new queue number. 11. The application calculates the waiting time for the user. 12. The application generates an entrance QR Code for the user and displays it with its respective dynamic waiting time, which is updated once every certain interval time. 13. The application notifies the user when 5min are remaining in the time waiting counter
Exit Condition	The exit conditions are illustrated on sequence diagram in Figure 17 1. The system detects that the QR code has been scanned and approved, so the user enters the store (Assumption D11). 2. The user drops the queue, so the number is discarded 3. The time is out, so the system doesn't detect a QR scan after 20 minutes it is the turn of the user.
Exception	The supermarket is not in opening hours when the user will be given permission to enter.
Special Req	The supermarket availability needs to be updated every 1 minute

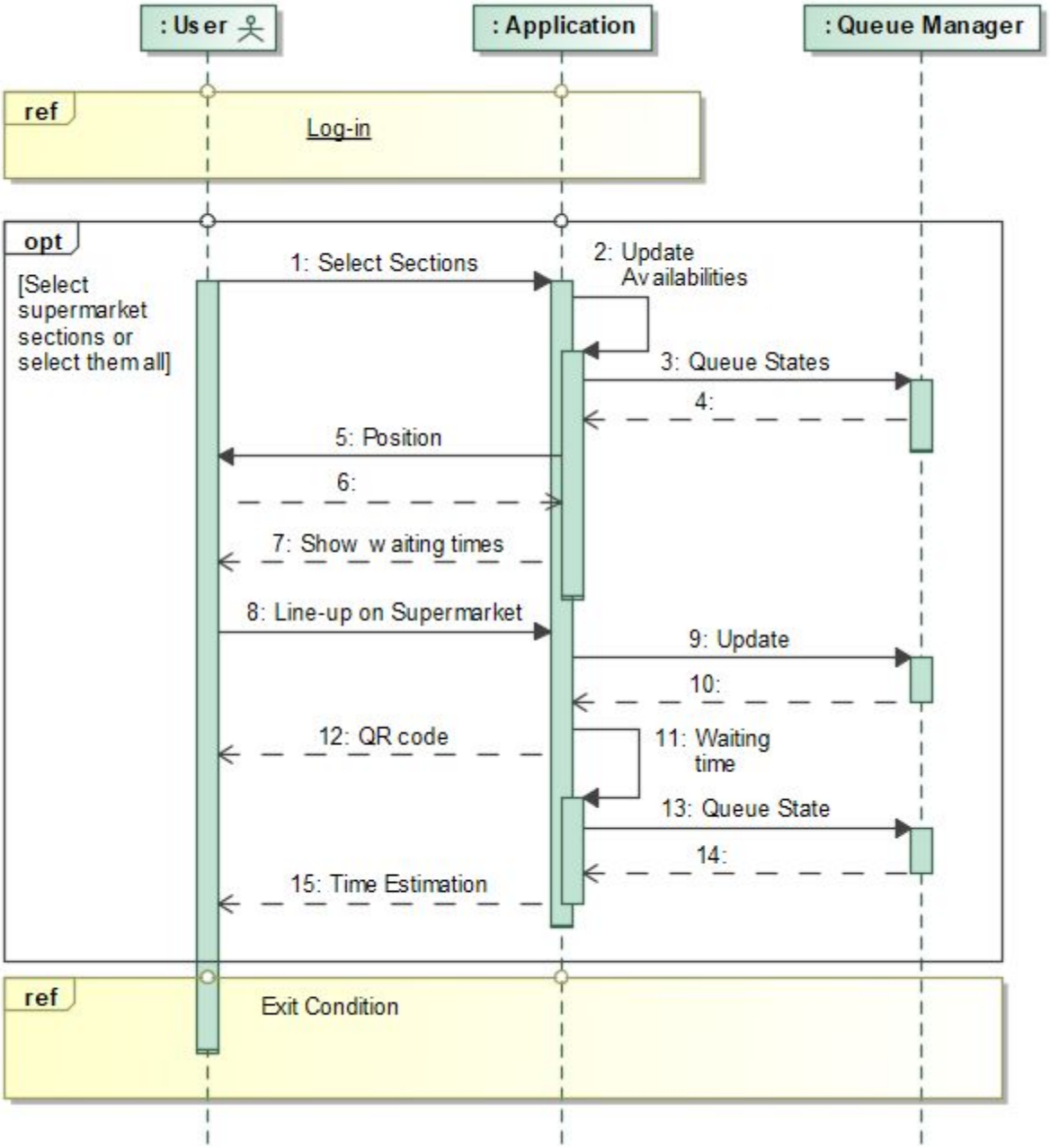

Figure 16: UC4 Specific Line-up Sequence Diagram

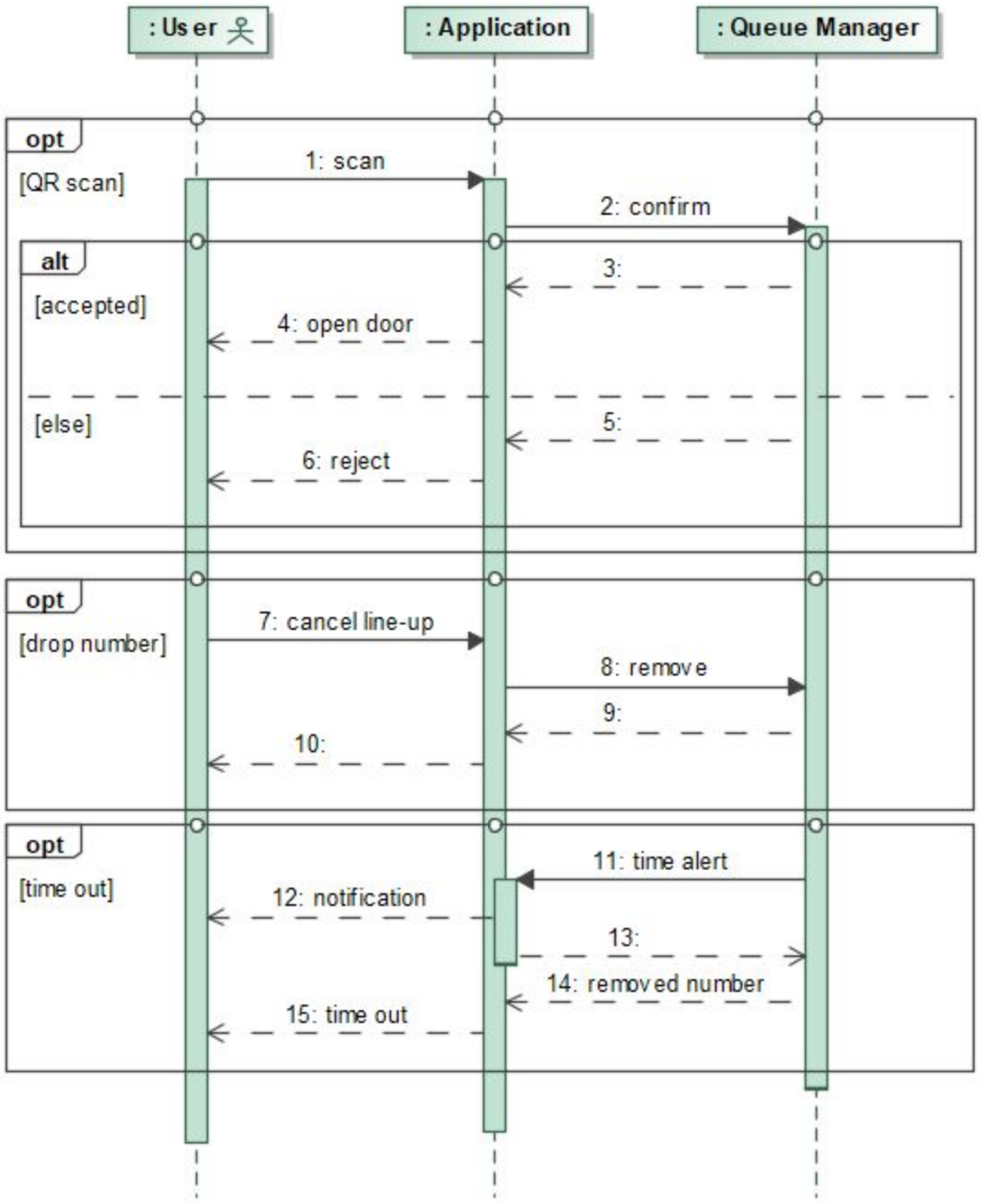

Figure 17: UC4, UC5 and UC6 Line-up Exit Condition Sequence Diagram

3.2.6 Use Case 5: ASAP Line-Up

Name	ASAP Line-Up
Actor	User
Entry Condition	The User has entered the application, which displays the login main page
Event Flow	1. The user logs into the application 2. The application requests the queue states of the supermarkets around the user's position. 3. The application shows the map page, which display different supermarkets options around and their availability 4. The user selects the desired sections to visit or select them all (default option). 5. The application displays supermarkets' availability on the map depending on the section selection. 6. The user selects in the ASAP Line-up button 7. The application computes and displays a hierarchical list of the supermarkets that have less waiting times (given section selection) and that are closer to the user. 8. The user selects a supermarket from the list. 9. The application does a request to update the state of the queue manager by inserting the number of the user. 10. The queue manager returns the confirmation of the new queue number. 11. The application calculates the waiting time for the user. 12. The application generates an entrance QR Code for the user and displays it with its respective dynamic waiting time, which is updated once every certain interval time. 13. The application notifies the user when 5min are remaining in the time waiting counter
Exit Condition	The exit conditions are illustrated on sequence diagram in Figure 17 1. The system detects that the QR code has been scanned and approved, so the user enters the store (Assumption D11). 2. The user drops the queue, so the number is discarded 3. The time is out, so the system doesn't detect a QR scan after 20 minutes it is the turn of the user.
Exception	There are no supermarkets open
Special Req	The application must have permission to access the user's location.

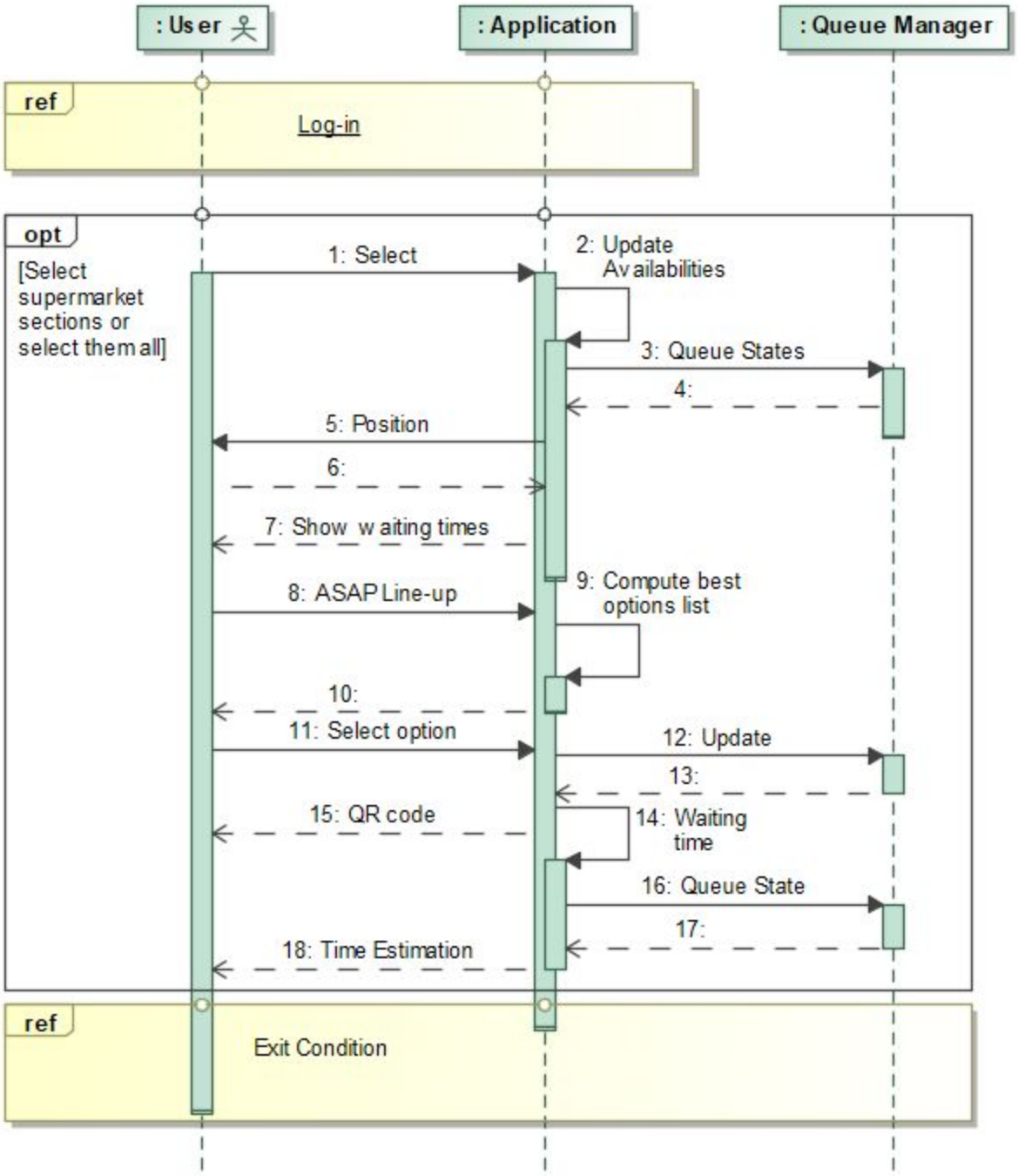

Figure 18: UC5 ASAP Line-up Sequence Diagram

3.2.5 Use Case 6: Book

Name	Book
Actor	User
Entry Condition	The User has entered the application, which displays the login main page
Event Flow	1. The user logs into the application 2. The application requests the queue states of the supermarkets around the user's position. 3. The application shows the map page, which display different supermarkets options around and their availability 4. The user selects the desired sections to visit or select them all (default option). 5. The application displays supermarkets' availability on the map depending on the section selection. 6. The user selects in the map the supermarket to buy in. 7. The application displays the pop-up of the selected supermarkets with the line-up and book options to pick a number. 8. The user selects the option to book 9. The application computes time slots suggestions for the selected supermarket. 10. The user selects a certain day in the day selection widget. 11. The application recomputes time slots suggestions for the selected day. 12. The user confirms a time slot. 13. The application does a request to update the state of the queue manager by inserting the number of the user. 14. The queue manager returns the confirmation of the new queue number. 15. The application calculates the waiting time for the user. 16. The application generates an entrance QR Code for the user and displays it with its respective dynamic waiting time, which is updated once every certain interval time. 17. The application notifies the user when 5min are remaining in the time waiting counter
Exit Condition	The exit conditions are illustrated on sequence diagram in Figure 17 1. The system detects that the QR code has been scanned and approved, so the user enters the store (Assumption D11). 2. The user drops the queue, so the number is discarded 3. The time is out, so the system doesn't detect a QR scan after 20 minutes it is the turn of the user.

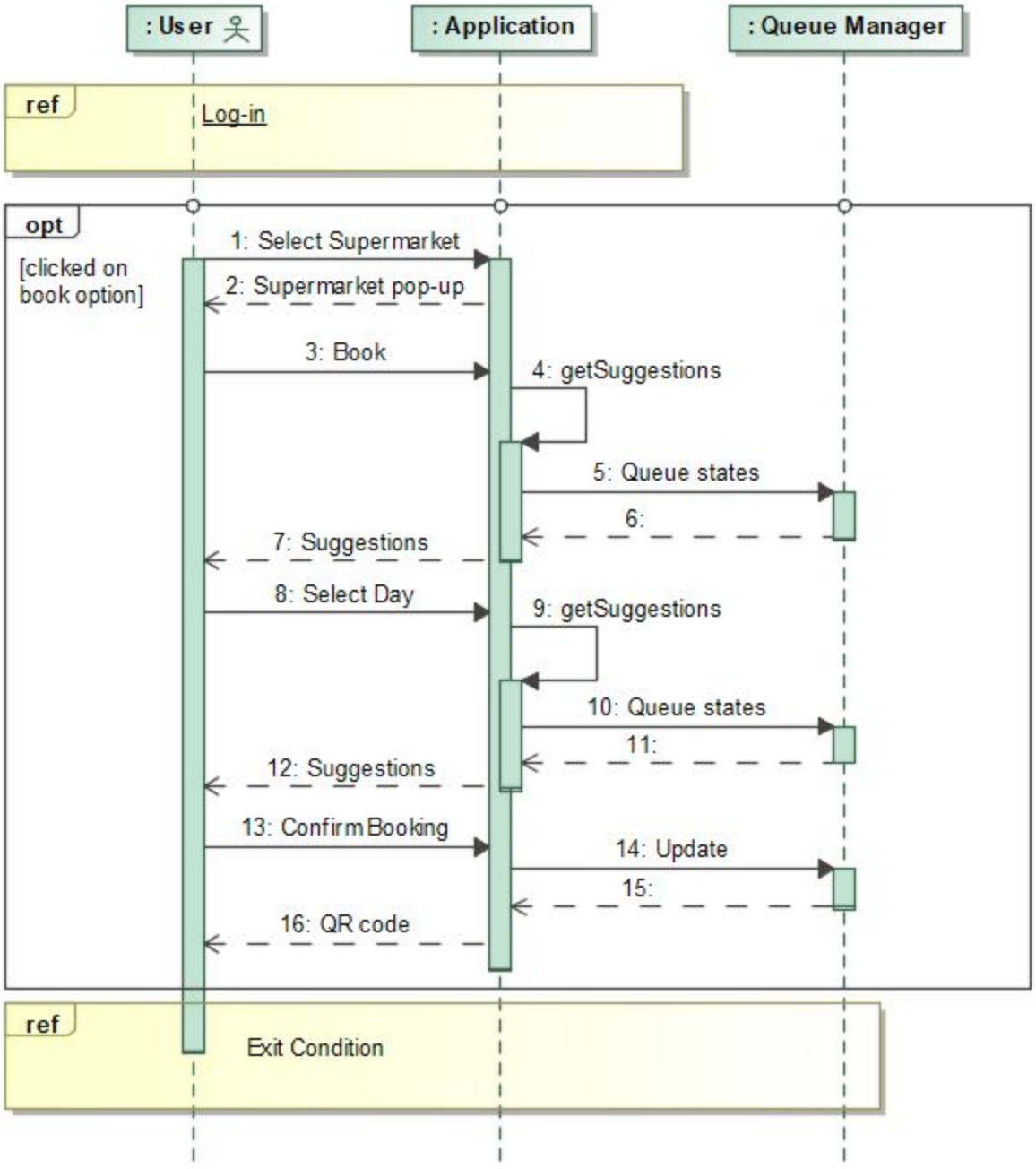

Figure 19: UC6 Book Sequence Diagram

3.2.9 Use Case 7: In-presence ticket

Name	In presence ticket
Actor	Supermarket Hub
Entry Condition	The supermarket hub is turned on and the user is outside the supermarket
Event Flow	1. The user clicks a button to get a number to enter the store 2. The application displays section options. 3. The user selects the desired sections to visit or select them all (default option). 4. The hub application does a request to update the state of the queue manager. 5. The queue manager returns the confirmation of the new queue number. 6. The application calculates and displays a estimated waiting time for the user. 7. The hub generates and prints a QR code ticket for the user.
Exit Condition	The user gets the printed ticket
Exception	1. The supermarket hub is out of service, paper or ink. 2. A preferential customer arrives. In this case the supermarket's doorman let them enter and the extra capacity for this purpose is spent out.
Special Req	1. The system detects that the QR code has been scanned and approved, so the user enters the store (Assumption D11). 2. The time is out, so the system doesn't detect a QR scan after 20 minutes it is the turn of the user.

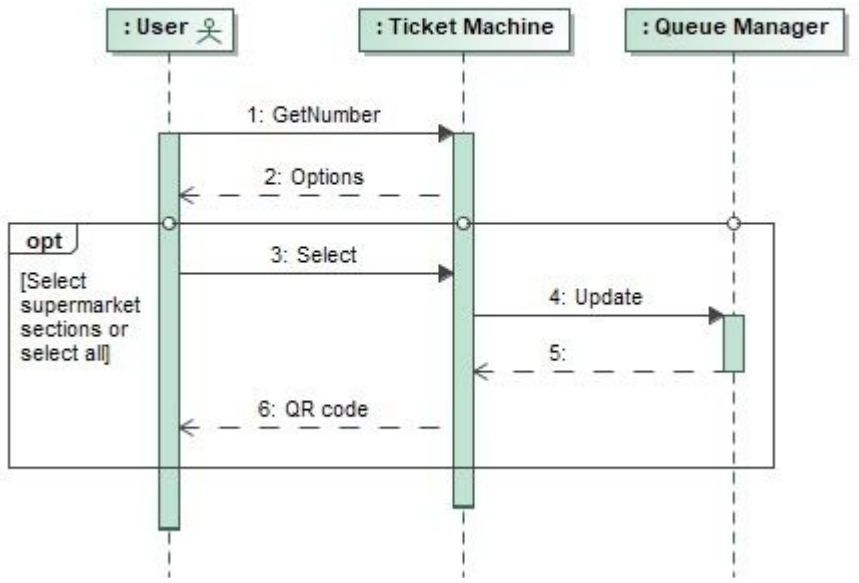

Figure 20: UC7 In-presence ticket Sequence Diagram

3.3. Performance Requirements

Based on the above user cases, the interactive user application must fulfil the following functional requirements:

1. Information of the supermarket queue and availability must be always available in the system.
2. The map must have filtering options that are configured in order to satisfy the different needs of the user explained in the Use Cases. These filters must apply the user selection in real time on the map. The filters can be personalized by the user. They can select a number of characteristics that the supermarket needs to have available, as well as time of accessibility
3. The application must show in real-time the time remaining for the entrance to become valid, as well as the supermarket location, user location and QR Code.
4. The application must generate in real-time suggestions of date, time and supermarkets according to the user's location and time needs.

3.4. Design Constraints

3.4.1 Standards compliance

1. The main constraint regarding standards compliance for a web app is the privacy, more specifically the personal data collection management.
    1.1. The EU General Data Protection Regulation (GDPR) specifies that customers' personal data collected to provide the core services of the app does not need their consent. Then, a privacy policy may be specified if desired.
    1.2. If any data analytic is performed using cookies, then the application cookie policy must be specified, and the customer must agree with it to use the application.
    1.3. According to each supermarket privacy policy a deletion period of the customer data may be specified.
    1.4. Customers can ask to delete their account and the personal data associated with it.
2. Location privacy. This application can be considered as a location-based service (LBS) as it uses a customer's location, and other constraints, to provide a suggestion of the best groceries to shop.
    2.1. A location data policy may be specified, and the user must agree with it.
    2.2. If any statistic is made out of the collected data for purposes or analytics outside of the application domain, then the data must be anonymized, and the exact data of the users will not be revealed. Any location privacy technique can be used for this purpose.

3.4.2 Hardware limitations

1. The customers require a smartphone with GPS to use the application to request a ticket.
2. The supermarkets must have a device managed by the supermarket staff through which the customers can request a ticket.
3. The supermarket must have a QR code reader to let the customers validate their tickets before beginning and after finishing their shops. This reader may be connected to the system which controls the entrance door.

3.4.3 Any other constraint

1. The must contain in the main page an interactive map, where the user location is shown at all times. The map must show the available supermarkets (i.e. currently open and operative).
2. Each of the supermarkets must be clickable, and a pop-up with options related to the selection must be shown.
3. The user must be able to make a reservation in an available supermarket in the desired date and time (as long as it doesn't overpass the 7 day limit timeframe) by obtaining a virtual ticket consisting of a QR code to be used in the selected supermarket.
4. Users can request a ticket if it is the first time that they use the app or if the last ticket they requested has been used or is expired. That means that the users can have only one ticket at a time.
5. The user must be able to cancel at any time their reservation in case it is desired.
6. The user must be able to create a new account in the system as long as they have an available email and was not used before in the same application.
7. The supermarket hub must be able to request an "unlimited" amount of tickets. This means that it should be possible to request in every time in which the supermarket is operative.

3.5. Software System Attributes

3.5.1 Reliability

It refers to the continuity of a correct service without failure or repair and can be expressed as a function of the elapsed time and the mean time to failure (MTTF) of the infrastructure.

However, nowadays more applications are using cloud-based services, infrastructures as a service (IaaS) that allow us to rent servers for compute and storage according to each one's necessity. Then, the reliability of this application will depend on the reliability of the IaaS provider as well as how good the choice of the infrastructure is. A good choice can be interpreted as choosing the right cloud instance with enough resources to perform adequately.

Backing up data is a key point for reliability and must be also considered when acquiring an IaaS. Data backup and recovery are important to be able to provide a 24/7 service or at least be close to it.

The recovery time objective (RTO) and the recovery point objective (RPO) may be computed to decide how frequent and which information should be recovered. This is crucial to avoid serious service disruption.

After a reliability test in the code evaluation phase, the application should be able to respond within a reasonable waiting time, e.g., 8 seconds maximum.

3.5.2 Availability

It is the readiness of correct service and can be expressed as a function of the MTTF and the mean time to repair (MTTR). Despite the system is not critical, the availability should be high because a ticket may be requested at any moment.

The availability may be at least 99% which is equivalent to 3.65 days downtime per year. Also a schedule may be defined as many of the supermarkets are closed at late-night and the waiting time is expected to be null.

3.5.3 Maintainability

It refers to the reparations needed to restore the correct service. To fulfil that definition, a recovery plan must be specified considering the definitions of RTO and RPO.

The hardware maintainability, e.g., server downtime, is related to the IaaS provider while code maintenance and software fixes are the responsibility of the support personnel. Well documented code is expected as a guide for future programmers.

3.5.4 Usability

The application must be user friendly, allowing users to request a ticket promptly.

Google authentication must be an option to let the user sign and log in the application in a faster way.

The application will use semaphore colors (green, yellow and red) to help the user identify the shops according to their waiting time. Also, suggestions will be implemented to help the user find a suitable choice in a shorter time.

3.5.5 Security

The application must follow a security culture, it means applying it from the design to the implementation and during the lifetime of the application.

Security standards must be considered when designing the application to be secure since the foundation and be later guiding principles for the developers. A framework must be defined to guarantee security.

A data management policy must be defined and communicated to the users.

3.5.6 Portability

The capacity of the application to run in different environments. A mobile-first web application will be implemented so it can support a wider range of devices.

4. FORMAL ANALYSIS USING ALLOY:

In this section, a formal modelling that deductively proves that the software goals are being fulfilled given the requirements and domain assumptions is presented. The modelling captures the most relevant aspects of the world and machine phenomena in order to show how the proposed requirements, jointly with reasonable domain assumptions, are enough to achieve the project purposes. Two models were considered to demonstrate the completeness of the requirements, the first one that considers customers entering to stores or waiting their turn in a simplified way, and the second one which shows requests for lining-up from users and suggestions responses from the system in a more complex way. For each of the models, representative instances are illustrated as well as meaningful assertions are developed in order to show how the goals are being fulfilled.

4.1 Safe Shopping Model

The objective of this model is to develop the deduction of goal G1 from the fulfilment of requirements M1 and M6 (details in Section 2.2), jointly with domain assumptions D1, D2 and D3 (details in Section 2.4). In Figure 21, a representative model instance is shown. In this instance there are 3 customers: Customer 0 lined-up for a single section which is full, so the system does not give authorization to enter yet. Customer 1 lined-up for 3 sections that have availability, so the customer is able to enter the store. Customer 2 selected all sections in which one of them is full, so he/she has to wait.

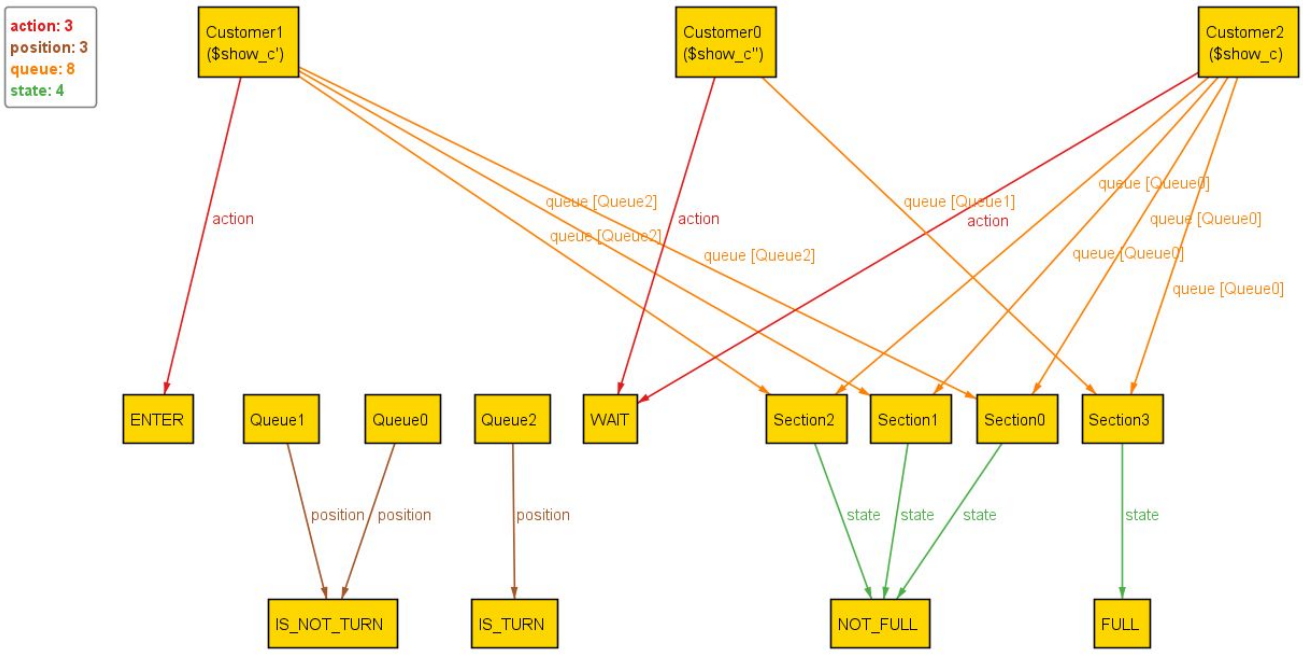

Figure 21: Safe Shopping Model representative instance.

The following code lines are the Alloy development of the model, in which an assertion that proves the accomplishment of goal G1 is shown as valid in the Alloy application.

```
//Customers have a number on a certain queue
// and perform an action (entering or waiting)
sig Customer{
queue: Queue -> Section,
action: Action
}
// Assumption D3: Customers that want to buy groceries,
// will line-up in order to wait for their turn to enter the store.
abstract sig Action{}
one sig ENTER extends Action{}
one sig WAIT extends Action{}
// Assumption D2: Store's sections can be full or not full
sig Section {
state: State
}
abstract sig State{}
one sig FULL extends State{}
one sig NOT_FULL extends State{}
// A Certain queue can provide a turn or not yet
sig Queue{
position: one Position
}
abstract sig Position{}
one sig IS_TURN extends Position{}
one sig IS_NOT_TURN extends Position{}
//Requirement M1: The system will control the amount of people
// inside the store by means of a queuing system,
// which will provide permission to enter if and only if
// all the sections selected by the user have capacity.
fact M1_GiveTurn{
(all c : Customer | (c.queue.Section).position = IS_TURN iff (Queue.(c.queue)).state = NOT_FULL
)
}
// Requirement M1: The queuing system will not allow customers
// to enter if it is not their turn.
fact M1_TurnControl{
(no c : Customer |
 c.action=ENTER and (c.queue.Section).position=IS_NOT_TURN
)
}
// Requirement M6: The system will be able to provide one ticket
// at a time to each user.
fact M6_UniqueTicket{
//Customer has only 1 queue
(all c : Customer |
(let q = (c.queue).Section | #q=1)
)
and
//Queue has only 1 Customer
(all q : Queue |
(let c = queue.Section.q | #c=1)
```

```
52. )
53. }
54. //G1. The in-person grocery shopping must be carried out
55. // guaranteeing the safety of customers and employees.
56. //D1. The in-person shopping will be safe as long as
57. // customers do not enter a full section.
58. assert G1D1_SafeShopping{
59. no c: Customer | (Queue.(c.queue)).state=FULL and c.action=ENTER
60. }
61. // the model is presented with a representative instance
62. pred show{
63. #Section=4
64. #Customer=3
65. some c : Customer | #Queue.(c.queue)=#Section
66. some c : Customer | #Queue.(c.queue)<#Section and c.action=ENTER
67. some c : Customer | #Queue.(c.queue)<#Section and c.action=WAIT
68. }
69. run show for 10
70. // goal is checked
71. check G1D1_SafeShopping
```

```
Executing "Check G1D1_SafeShopping"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  734 vars. 57 primary vars. 1502 clauses. 77ms.
  No counterexample found. Assertion may be valid. 15ms.
```

4.2 Requests-Responses Model

The objective of this model is to develop the deduction of goals G2, G3, G4 and G5 from the fulfilment of requirements M2, M3 and M6 (details in Section 2.2), jointly with domain assumptions D2, D3, D4, D5, D6 and D7 (details in Section 2.4). In order to elaborate a model that explains more complex phenomena, it was necessary to create an abstraction that involves more signatures and more sophisticated functions to describe the modelling environment. In Figure 22, the different signatures with the relation between them are displayed. There is a class *Customer* that wants to shop on a set of *Section* at a certain *Time* (soon or later), is located at a certain *Position* (close or far) with respect to a *Store* and requests a certain *Queue* with a certain *Type* of request (line-up or book). The *Queue* class is related to a set of *Section* belonging to a *Store,* and has a certain *State* (availability: full or not full). As a response, the system gives to the customer a *Suggestion* on a certain *Store*.

In Figure 23, a representative model instance to illustrate UC4 (Specific Line-up) is shown. In this instance there is a customer that wants to shop soon in 2 sections, so the system suggests the store that has these sections available (not full) and it is closer, however, the customer line-up on another store that is full and far, so he/she must wait to enter.

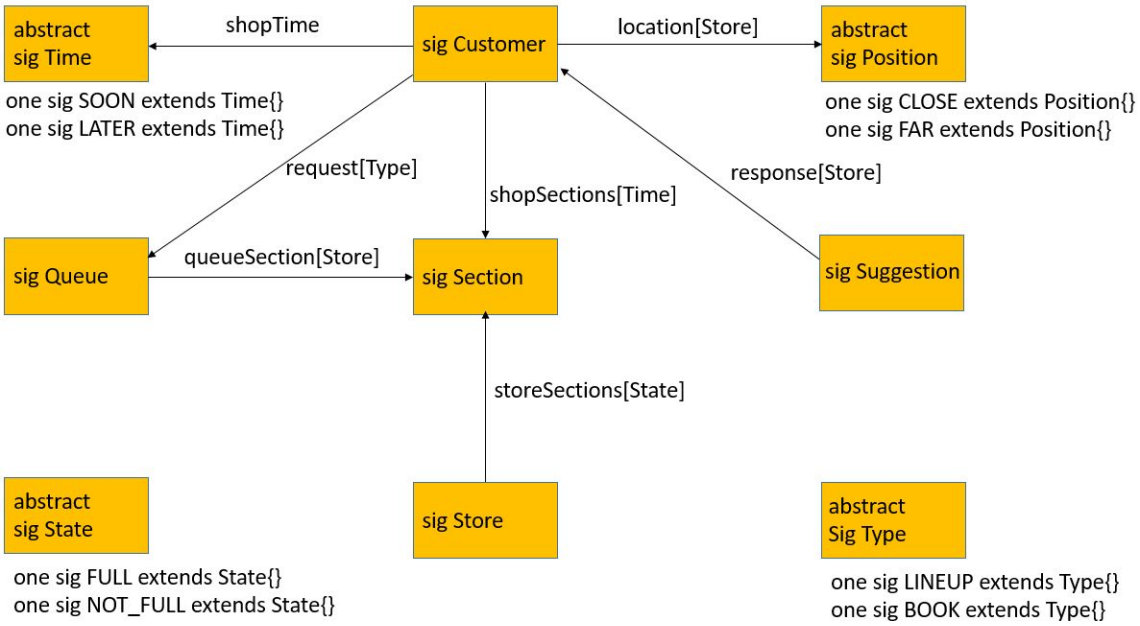

Figure 22: Requests-Responses Model Signatures

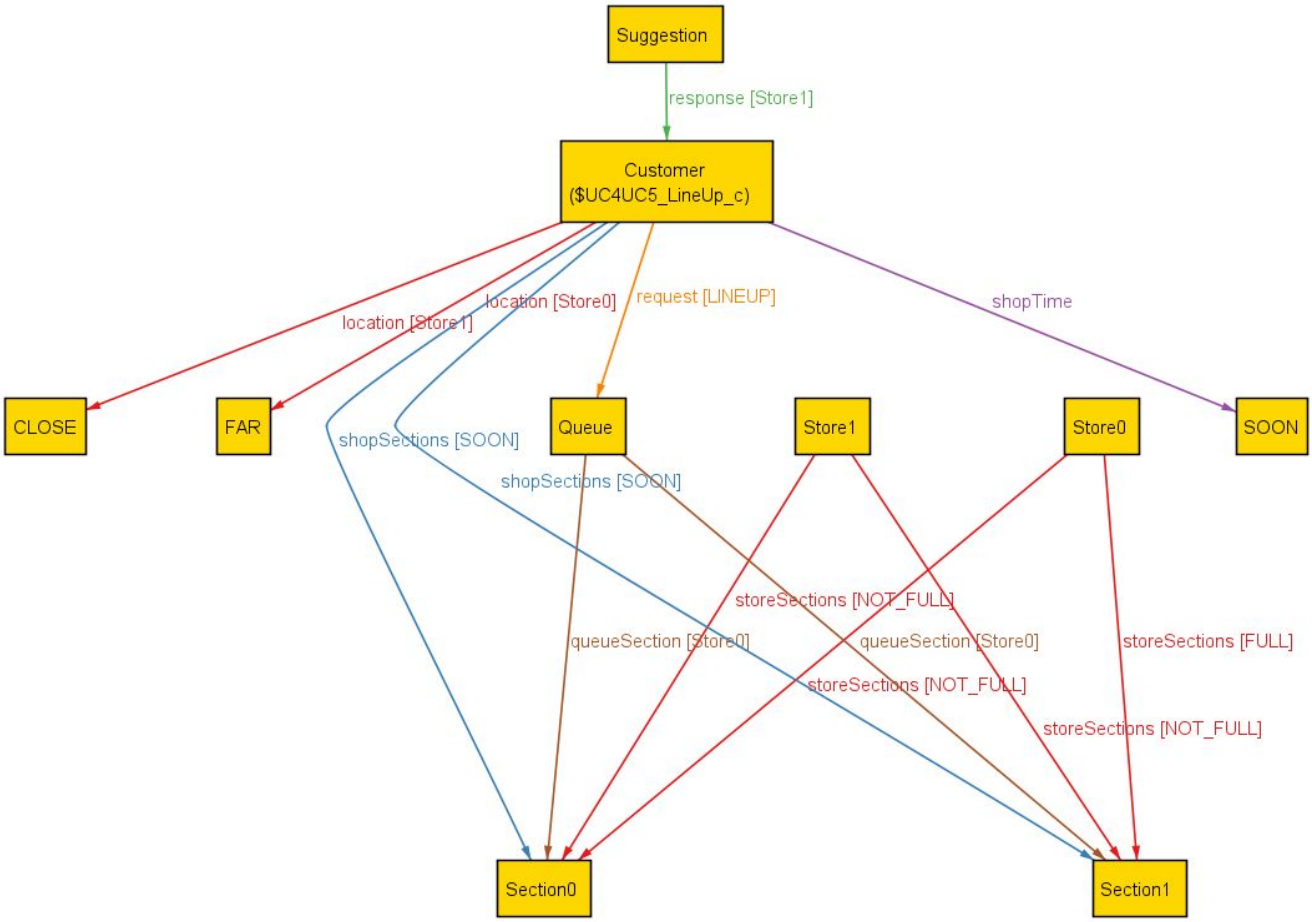

Figure 23: Requests-Responses representative instance of UC4.

In Figure 24, a representative model instance to illustrate UC5 (ASAP Line-up) is shown. In this instance there is a customer that wants to shop soon in 2 sections, so the system suggests the store that has these sections available (not full) and it is closer, so the customer line-up on that suggested store in order to shop ASAP.

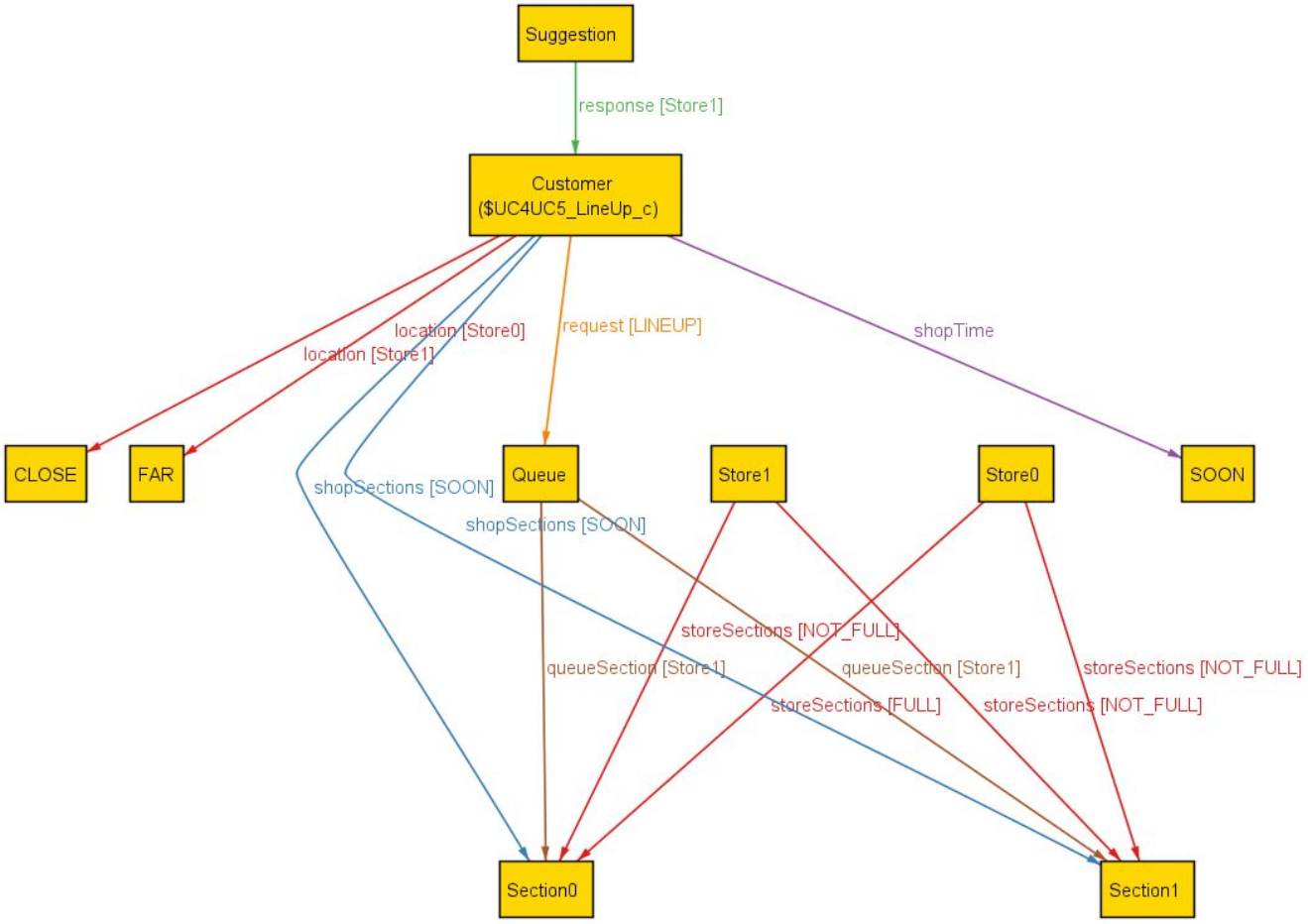

Figure 24: Requests-Responses representative instance of UC5.

In Figure 25, a representative model instance to illustrate UC6 (Book) is shown. In this instance there is a customer that wants to shop later in 2 sections, so the system suggests the store that has these sections available (not full), so the customer book on that store even if it is more far.

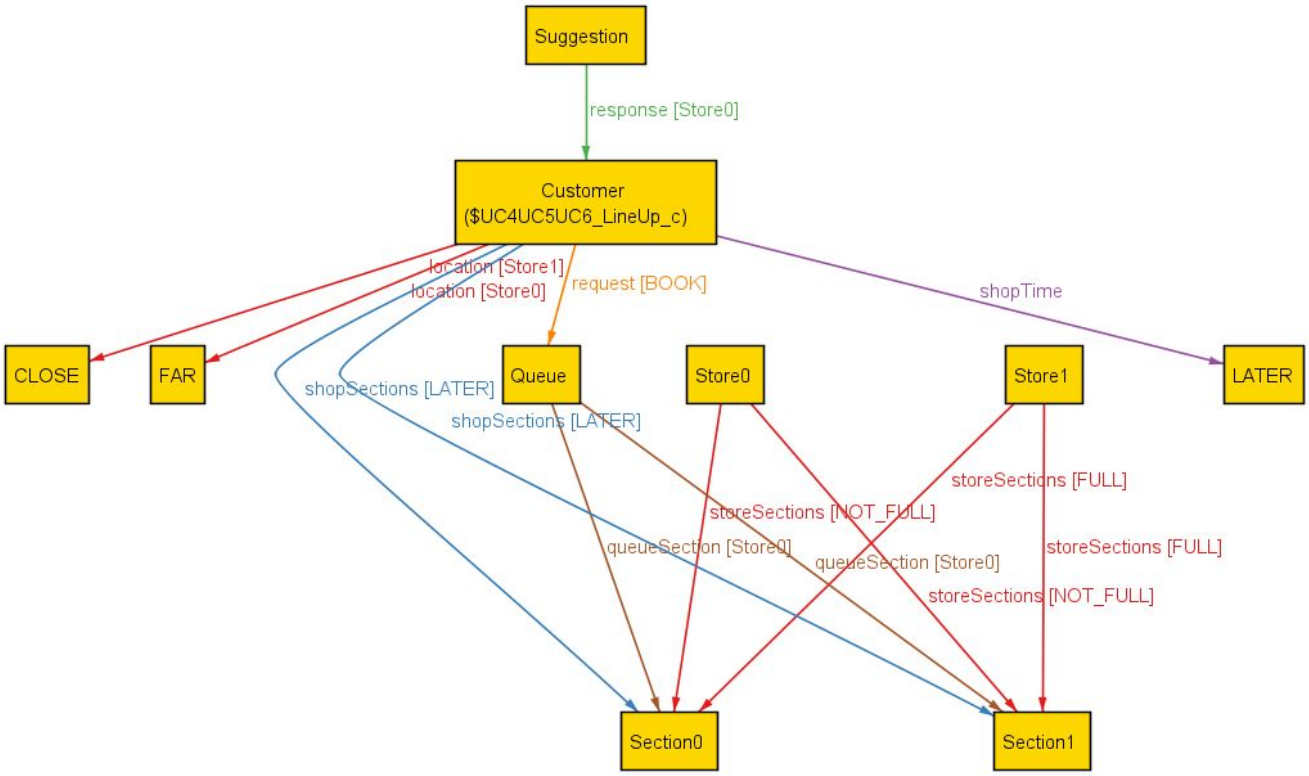

Figure 25: Requests-Responses representative instance of UC6.

The following code lines are the Alloy development of the model, in which two assertions are validated in the application in order to prove the accomplishment of goals G2, G3, G4 and G5.

```
// Assumption D3: Customers have intention of going to shop at
// a certain time. They also have a certain position wrt Stores
// Customers can also have a number on a certain queue
// and perform an action (entering or waiting)
sig Customer{
location: Store -> Position,
shopTime: Time,
request: Type -> Queue,
shopSections: Time -> Section
}
// Stores have Sections, which have a certain State
// Assumption D2: Store's Sections can be full or not full
sig Store{
storeSections: State -> Section
}
// Position of Customers wrt Stores
abstract sig Position{}
one sig CLOSE extends Position{}
one sig FAR extends Position{}
// Time in which Customers want to shop
abstract sig Time{}
```

```
one sig SOON extends Time{}
one sig LATER extends Time{}
// Sections' State
abstract sig State{}
one sig FULL extends State{}
one sig NOT_FULL extends State{}
// Customers' Requests Type
abstract sig Type{}
one sig LINEUP extends Type{}
one sig BOOK extends Type{}
// Stores' Sections
sig Section{}
// Suggestion of a Store to a Customer
sig Suggestion{
response: Store -> Customer
}
// Queue to a set of Store's Sections
sig Queue{
queueSection: Store -> Section
}
// FACT Stores have only one position
fact FACT_UniqueStorePosition{
(all p: Position, c : Customer |
(let s = c.(location.p) | #s=1)
)
}
// FACT Customers are close or far from a certain Store
fact FACT_CloseOrFar{
(all  c : Customer, s : Store |
(let d = s.(c.location) | #d=1)
)
}
// Assumption D2: Each section can be full or not full
fact D2_SectionState{
(all s : Section, st : Store |
(#st.(storeSections.s)=1)
)
}
// Assumption D3: Customers want to buy soon or later, not both
fact D3_SoonOrLater{
(all  c : Customer |
(let t = (c.shopSections).Section |#t=1)
)
}
// Assumption D3: customers will shop at the time they want
fact D3_CustomersWill{
(all c : Customer |
((c.shopSections).Section=c.shopTime)
)
}
// Assumption D4: Stores have at least 1 section
// and sections are associated to at least 1 store
fact D4_SectionStore{
(all  s : Store |
```

```
76. (let secs = State.(s.storeSections) | #secs>0)
77. )
78. and
79. (all s : Section |
80. (let stor = storeSections.s.State | #stor>0)
81. )
82. }
83. //Assumptions D5 and D6: if the customers want to buy soon, they line-up
84. // if the customers want to buy later, they book
85. fact D5D6_RequestDecission{
86. (all  c : Customer |
87. ( c.shopTime=LATER implies (c.request).Queue=BOOK)
88. and
89. (c.shopTime=SOON implies (c.request).Queue=LINEUP)
90. )
91. }
92. //Assumption D7: the sections that the customers want to buy in
93. // are the ones that they request for
94. fact D7_CustomerWill{
95. (all  c : Customer |
96. (Time.(c.shopSections) = Store.(Type.(c.request).queueSection))
97. )
98. }
99. // Requirement M6: Queues are associated to only 1 customer
100.    fact M6_QueueCustomer{
101.    (all  q : Queue |
102.    (let c = request.q.Type | #c=1)
103.    )
104.    }
105.    // Requirement M6: The system will be able to provide one ticket
106.    // at a time to each user.
107.    fact M6_UniqueTicket{
108.    // Customer has only 1 queue
109.    (all c : Customer |
110.    (let q = (c.request).Queue| #q>0 implies #q=1)
111.    )
112.    }
113.    // Requirement M6: Customer can only perform 1 type of request at a time
114.    fact M6_UniqueQueue{
115.    (all  c : Customer |
116.    (let q = Type.(c.request) |#q>0 implies #q=1)
117.    )
118.    }
119.    // Requirement M6: Request are associated to a single store
120.    fact M6_UniqueStore{
121.    (all  q : Queue |
122.    (let s=(q.queueSection).Section | #s=1)
123.    )
124.    }
125.    // Requirement M2: if Customers make a request, they must receive a suggestion
126.    fact M2_RequestSuggestion{
127.    (all  c : Customer |
128.    (#c.request>0 implies #response.c>0)
129.    )
```

```
}
// Requirement M2: Suggestions must contain stores with sections in which
//the customer wants to buy in and these sections must be not full
fact M2_SectionSuggestion{
(all  s : Suggestion |
(s.response.Customer in storeSections.(Time.(Store.(s.response).shopSections)).NOT_FULL)
and
(#s.response.Customer.storeSections.(Time.(Store.(s.response).shopSections))=1)
)
}
// Requirement M3: If the customer wants to shop as soon as possible, the store suggestion
// must be close to him/her
fact M3_RequestSuggestion{
(all c: Customer |
(c.shopTime=SOON implies Suggestion.(response.c).(c.location)=CLOSE)
)
}
//Requirement M2: for each customer there is one best suggestion
fact M2_Suggestion{
(all s : Suggestion |
(let c = s.response|#c=1)
)
and
(all c : Customer |
(let s = response.c|#s>0 implies #s=1)
)
}
// the model is presented with representative cases
// The following predicate is a simple instance to illustrate Lining-up/Booking UCs 4, 5 & 6
pred UC4UC5UC6_LineUp{
// Only 2 sections, 1 customer, 1 request
#Section=2
#Customer=1
#(Customer.request).Queue=1
#Store.(storeSections.Section)=2 //there are FULL and NOT_FULL sections
(all s : Store | //stores contain all sections
(let s=s.storeSections | #s=#Section)
)
(some c : Customer |// the customer wants to buy in more than 1 section
(let s=Time.(c.shopSections) | #s>1)
)
}
//G2. The customers must be able to line-up in a store if they want to enter to shop as soon as
possible.
//G3. The customers must be able to book a number in order to make a reservation to enter a specific
//store at a certain date and time.
assert G2G3_Requests{
(all c: Customer | #(c.request).Queue>0)
and
(all c: Customer | c.shopTime=SOON implies (c.request).Queue=LINEUP)
and
(all c: Customer | c.shopTime=LATER implies (c.request).Queue=BOOK)
}
//G4. If customers want to shop as soon as possible, they must receive suggestions on the most
convenient
```

```
183.    // options in terms of distance and waiting time.
184.    // G5. If customers want to book a number on a specific store, they must receive suggestions on the
    most
185.    // convenient options in terms of availability.
186.    assert G4G5_Suggestions{
187.    (all c: Customer |
188.    (c.shopTime=SOON implies Suggestion.(response.c).(c.location)=CLOSE and
    Suggestion.(response.c).storeSections.(Time.(c.shopSections))=NOT_FULL)
189.    )
190.    and
191.    (all c: Customer |
192.    (c.shopTime=LATER implies
    Suggestion.(response.c).storeSections.(Time.(c.shopSections))=NOT_FULL)
193.    )
194.    }
195.    // run demo instances
196.    run UC4UC5UC6_LineUp for 10
197.    // check goals
198.    check G2G3_Requests
199.    check G4G5_Suggestions
```

```
Executing "Check G2G3_Requests"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  2661 vars. 147 primary vars. 5924 clauses. 6ms.
  No counterexample found. Assertion may be valid. 2ms.

Executing "Check G4G5_Suggestions"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  2764 vars. 147 primary vars. 6114 clauses. 8ms.
  No counterexample found. Assertion may be valid. 31ms.
```

5. EFFORT SPENT:

Jesus Rodrigo Cedeño Jimenez	24 hours
Diego Andres Cumming Cortes	24 hours
Angelly de Jesus Pugliese Viloria	24 hours

6. REFERENCES

1. M. Jackson. The World and the Machine. (1996).
2. Fakhroutdinov, K. (2020). Unified Modeling Language (UML) description, UML diagram examples, tutorials and reference for all types of UML diagrams - use case diagrams, class, package, component, composite structure diagrams, deployments, activities, interactions, profiles, etc. Retrieved 13 November 2020, from https://www.uml-diagrams.org/
3. Computer Infrastructures: dependability slides. (2020).
4. Legal Guidelines For The Use Of Location Data On The Web — Smashing Magazine. (2016). Retrieved 13 November 2020, from https://www.smashingmagazine.com/2016/03/location-data-web-development-and-the-law/
5. General Data Protection Regulation (GDPR) – Official Legal Text. (2020). Retrieved 24 December 2020, from https://gdpr-info.eu/