RASD
Requirement Analysis and Specification Document

POLITECNICO
MILANO 1863

ANDREA MANGLAVITI
DAVIDE MARINARO
LUCA MARINELLO

Version 1.0
Date 23/12/2020
Prof. Matteo Giovanni Rossi

Contents

1. Introduction

1.1 Purpose

This document is the Requirements Analysis and Specification Document (RASD) of CLup (Customer Line-up) application whose purpose is to guide the developer in the realization of the software.

It focuses on the description of the main goals considering the context in which the software will be used, the domain and its modelling, the requirements and specification that best characterize the software. Moreover, it will go over the analysis of various scenarios with the uses cases that describe them. It also includes the draft if the interfaces and a brief description of functional, non-functional requirements and the attributes that distinguish the quality of the system.

Finally, to understand better the development of the document, it contains the revision history that describes the phases of the production of the document, with the references used and the description of its structure.

1.2 Scope

The main goal of the software is on one hand allowing store manager to regulate accesses of people to their store and on the other hand helping users not to wait forming long lines outside the store if it is full. Basically, it is an application that helps both store managers managing users' access and users saving their time.
Customer Line-Up (CLup) offers these main functionalities:

- Basic Service: first of all, allows users to choose a store near their actual position and, once they chose, the app allows them to check whether they can access a store or they have to line-up. In the former case they receive a ticket to access the store without waiting as soon as they arrive. In the latter case, they receive a ticket to line-up for entering the store; also, the application warns users about the expected waiting time to enter the selected store and, considering how much time they spend to reach the selected store from the real time position provided by GPS sensors, it notifies them when it is recommended to go to the store not to lose the place in the line-up. The ticket can be generated using a QR code that has to be scanned before entering. Secondly, the application should help managers to regulate the influx of people into their stores providing, for example, statistics about the average time spent by users in the store and other useful information mined with machine learning techniques using data form users collected from the application. Lastly, while managing accesses, the app has to consider that there are people that do not have access to the required technology, and so it has to provide fallback options.

- Book a Visit functionality: It is the first advanced functionality of the application. Since not all of the users will visit all of the sections of the supermarket, but some will pick up few products thus there will not be a uniform distribution in the market. Therefore, the application would allow the users to book a slot of time to visit the supermarket, asking the users either to provide an expected duration of the visit or to provide a list of products to purchase. Both of this information will allow the application to manage in effectively the line of customers. Moreover, the application can infer an expected time spent in the market for long term users through data mining.
- Suggest Alternatives functionality: It is the second advanced functionality of the application. Since it is possible that the selected supermarket has a long time waiting due to all people lined up by the application, it could calculate waiting time expected in other near supermarkets and suggest the user these if the time is lower than the selected one. Moreover, the software can learn the habits of the user and send notification when one of his/her usual time slots is free.

1.2.1 World Phenomena

WP1	User takes a smartphone with himself
WP2	Customer needs to buy something
WP3	Customer goes to the supermarket
WP4	Customer waits to enter
WP5	Customer buys products

1.2.2 Shared Phenomena

SP1	Customer enters the supermarket
SP2	Customer exits the supermarket
SP3	Customer scans the QR
SP4	User receives the QR code
SP5	User receives notification about waiting time
SP6	User compiles the form

SP7	User removes himself form the line-up
SP8	User books a visit
SP9	User cancels a visit
SP10	User receives alternatives
SP11	The QR code expires
SP13	Non-User receives a physical QR code without app

1.2.3 Goals

G1	Allow user to create a ticket
G2	Allow user to enter the supermarket
G3	Allow non-user to enter the supermarket
G4	Allow user to book a visit
G5	Allow user to check ticket
G6	Allow user to delete a ticket
G7	Allow user to check a visit
G8	Allow user to delete a visit
G9	Suggest alternatives for tickets to users
G10	Suggest alternatives for visits to users
G11	Send notification to user about free available slots
G12	Build statistics using collected data
G13	Allow customer to exit the supermarket

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Customer	A person that has to buy something at the supermarket
User	A person with a smartphone that downloaded the application and uses it
Non-User	A person without a smartphone or that do not have the application nor uses it
Valid Entrance	A ticket or visit that allows the customer (if visit, only user) to enter the supermarket
[General] Entrance	Term used to refer to both visits and tickets, pointing out their common attributes instead of their differences

1.3.2 Acronyms

RASD	Requirement Analysis and Specification Document
GPS	Global Positioning System
QR Code	Quick Response Code

1.3.3 Abbreviations

CLup	Customer Line-Up (name of the application)
WPn	World Phenomena number n
SPn	Shared Phenomena number n
Gn	Goal number n
Dn	Domani assumption number n
Rn	Requirement number n

1.4 Revision History

10/11/2020	RASD of CLup is created
15/12/2020	First Version of RASD (Draft)

1.5 Reference Documents

- Specification Document: “CLup - Customer Line-up Optional Project Assignement.pdf”
- Slides of the lectures

1.6 Documentation Structure

Chapter 1 introduces the focus of this document explaining the main functionalities of the software and including a brief analysis of phenomena and goals. It is also present a section that explains the most used terms and acronyms. Finally, it shows the revision history and the structure of the document.

Chapter 2 contains an overall description of the software modelled through a UML class diagram and some UML state diagrams that shows the flow of events of the main functionalities. Moreover, it includes a brief explanation of the main functionalities along with the description of the characteristics of the actors interacting with the application. Finally, it contains the domain assumptions that can be inferred from the assignment document and the described models.

Chapter 3 contains all the requirements the software needs and so it is the core of the document. First, it briefly explains external interfaces requirements, including user, hardware software and communication ones. Then it goes over functional requirements, also defining a list of use cases associated with their own sequence diagrams; this section also includes some scenarios useful to clarify how the application can be used. Finally, it illustrates the following non-functional requirements: performance, design and software system attributes

Chapter 4 includes the alloy code and the corresponding metamodels generated from it, with a brief introduction about the main purpose of the alloy model.

Chapter 5 shows the effort spent for each member of the group.

Chapter 6 includes the reference to documents and tools used to produce this document.

2. Overall Description

2.1 Product Perspective

2.1.1 UML Description

The UML in the next page describes at high-level the model of the software to be developed. It mainly highlights how the application can integrate all the functionalities requested in the assignment: the basic service and the advanced functionalities. Obviously, the UML diagram does not include all the needed classes, but only the most relevant to understand how it works. CLup offers several functionalities other than the basic service of taking a ticket to line-up: for this reason, we have included classes such as Position, Product or Category, and we have created a class GeneralEntrance that generalizes the two types of entrance permitted: the normal one using a ticket (basic service) and the advanced one booking a visit (book a visit advanced functionality).

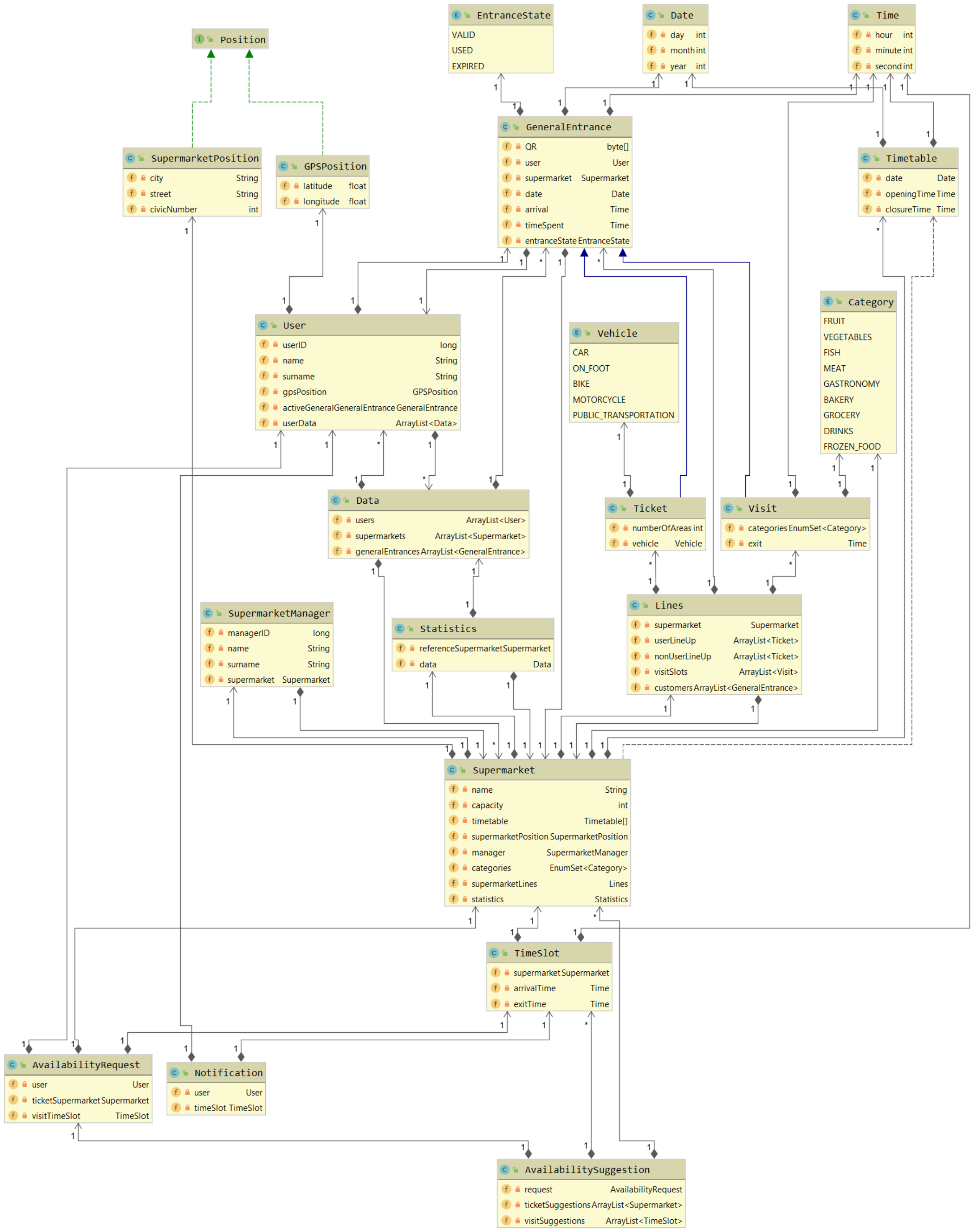
The user can simply download the app and start using it benefiting of its services. Supermarket managers, on the other hand, have to install the app with a server that can receive data from and send data to all users through internet connection. Now we are going to identify the main characteristics of the app referring to the classes in UML.

The user can create a ticket to line-up in a specific supermarket giving some information to better manage lining-up: for this reason, the ticket class is provided with some useful attributes such as categories (a set of Category representing what kind of products user wants to purchase), expectedArrival and expectedExit, to better manage queues of the supermarket. When the ticket is created, the application generates a valid and unique QR code that can be used to enter the supermarket. Finally, the Ticket class that will contain all this information, extends the class GeneralEntrance, since taking a Ticket is only one of the way user can enter the supermarket.

The user can book a visit to go to the supermarket, but only in previously selected wards. This is the second and last way a user can enter the supermarket and, consequently, also the class Visit extends the class GeneralEntrance. This class inherits all the attributes of GeneralEntrance and for this reason it is very similar to Ticket class, but it will be used in a different way since it has different characteristics.

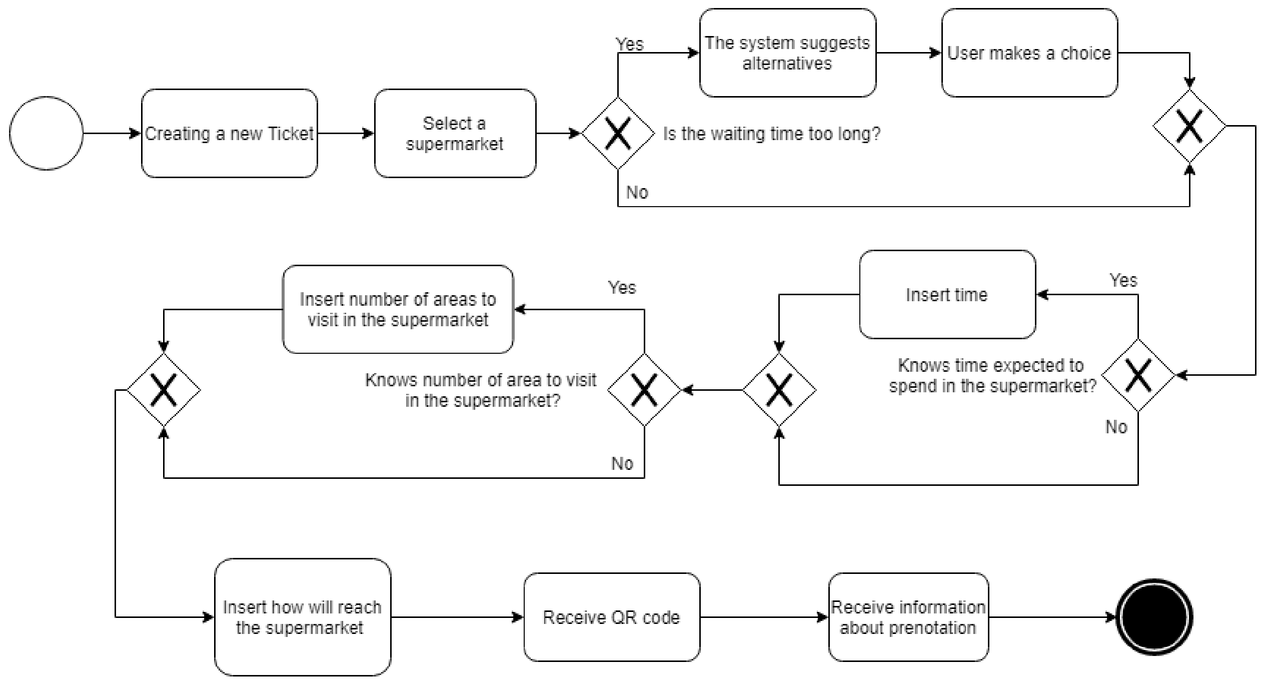
The user can receive some notifications from the app. The goal of these notifications is to suggest the user more advantageous alternatives to go to the supermarket. For instance, if the supermarket is too crowded, the application will analyse data stored in the local database and data coming from servers in supermarkets (Data class) to find a more advantageous solution: the user can choose to accept or refuse that suggestion.

Finally, the app can provide supermarket managers with statistics on accesses on their supermarkets, analysing data stored in their databases (Data and Statistic classes).



2.1.2 State Charts

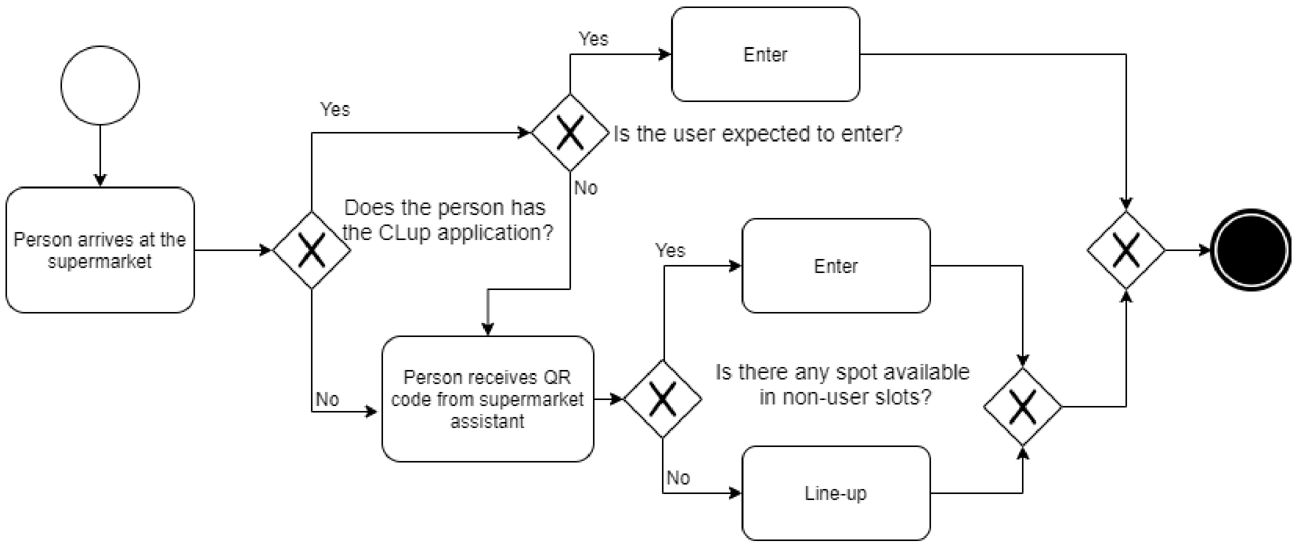
Now we are going to analyse critical aspects of the application, modelling behaviours and showing the evolution over time of their states through appropriate state diagrams, which are reported below.



Creating a new Ticket
Select a supermarket
Yes
The system suggests alternatives
User makes a choice
Is the waiting time too long?
No
Insert number of areas to visit in the supermarket
Yes
Insert time
Yes
Knows number of area to visit in the supermarket?
Knows time expected to spend in the supermarket?
No
No
Insert how will reach the supermarket
Receive QR code
Receive information about prenotation

Figure 1 – State diagram 1: Creating a ticket and receiving QR Code

In the first state diagram (figure 1), we model the creation of a new ticket by a user showing the main actions that he has to perform until receiving the QR Code.



Yes
Enter
Yes
Is the user expected to enter?
No
Person arrives at the supermarket
Does the person has the CLup application?
Yes
Enter
Is there any spot available in non-user slots?
No
Person receives QR code from supermarket assistant
No
Line-up

Figure 2 – State diagram 2: Manage people arriving

In the second state diagram it is modelled how CLup manage people without arriving at one specific supermarket. In particular, we thought that each supermarket has a percentage (e.g., 10%) of slots available for non-user customers.

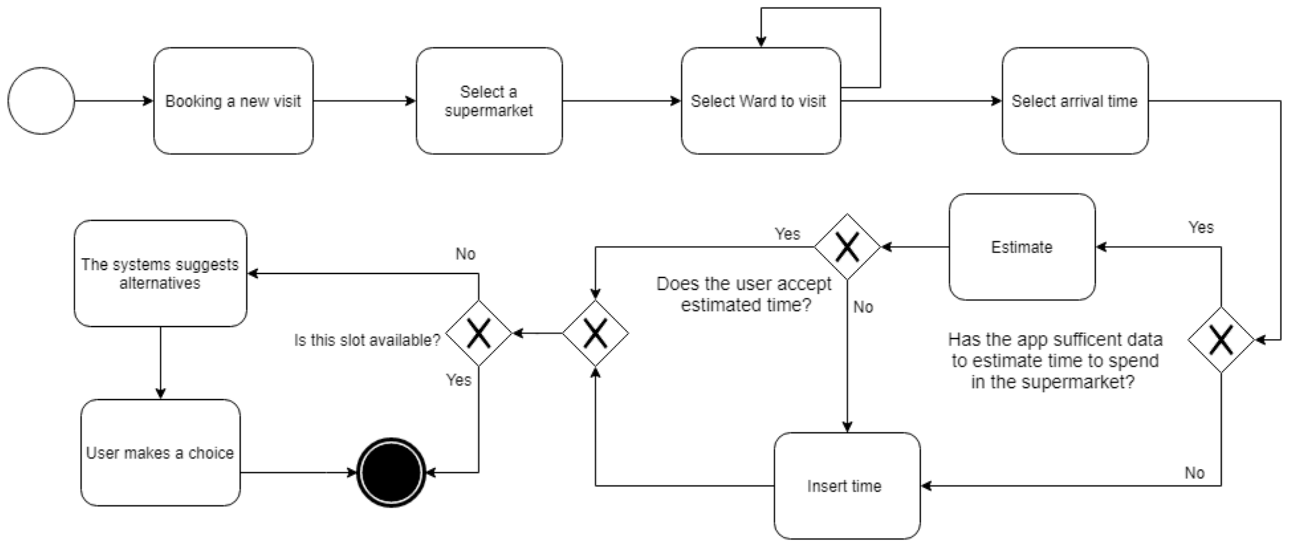
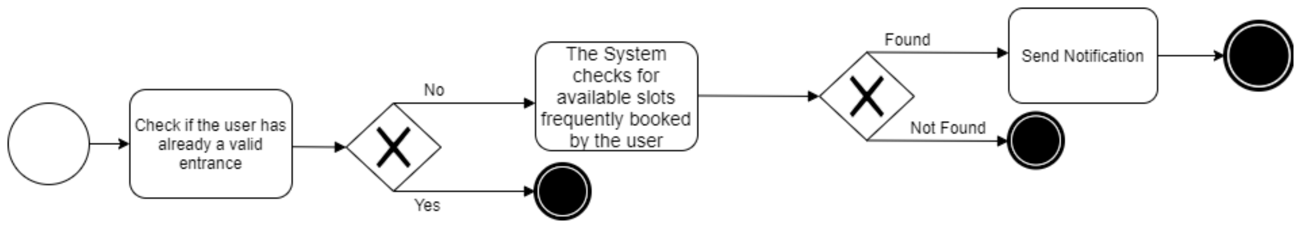


Figure 3 – State Diagram 3: Book a visit

In the third state diagram it is modelled the "Book a visit" function which allow user to book a visit in the supermarket. This function is available only for few days after the current one (e.g., 2 days).



Check if the user has already a valid entrance
No
Yes
The System checks for available slots frequently booked by the user
Found
Not Found
Send Notification

Figure 4 – State Diagram 4: Send Notification

In the fourth state diagram it is modelled the "Send Notification" functionality which is able to infer data to find days and time slots the user is usual to go to the supermarket and notify him in case one of those slots is available and fits the current day.

2.2 Product Functions

In this paragraph, we describe the main functionalities of the applications

- Line-Up management: This functional constitutes the core of the application, specifically allows the users of the application to pick up a virtual ticket, composed by a QR code and by a number, that will put them in a virtual queue for a supermarket of their choice. When the turn of the user will be near to being called the application will notify him to go to supermarket. The application to allow the user to arrive in time will calculate an estimated time of arriving through the GPS position of the user's smartphone. Moreover, the application, guarantee a smooth scrolling of the queue, will reserve to a small percentage of the non-user costumers of the supermarket Immediate access to the supermarket. If the number of the costumer, exceeds the maximum capacity of the supermarket, any eventual non-user will be provided with a ticket from the supermarket.
- Monitor the access: This function, through data collection and successive mining of the data collect, allows the application to understand the habits of the users inside the supermarket and then calculate an expected time spent by the users for each visit of the store. The application, to collect the data needed, will monitor the access of each user, through the QR code of the ticket and then monitoring the exit from the same store. In addition, the monitoring of accesses is fundamental to guarantee the access to the supermarket only to those that have a ticket that "demonstrate that the user can enter".
- Book a visit: This function is even more efficient in dealing with the virtual line, making the application even more time saving for the users. It allows them to book a slot of time in which the user wants to go to the supermarket. In doing so the user after choosing the supermarket, could indicate the wards that he intends to visit. Moreover, if the application has enough data will calculate the expected time that the user will spend inside the store, on the contrary it will ask to the user to insert the time that he thinks will spend doing shopping. After the booking phase is completed, the application as for the main functionality will notify when the user should go to the supermarket.
- Suggest alternatives: This functionality is used to balance out the number inside the stores of the chain. If the supermarket user wants to go is crowded, then the system will suggest other appropriate alternatives (both in time and/or place).
- Send notification: This last functionality activates when the user has no active entrances, but in the past he/she booked some visits. With data of those previous visits, the application infers whether the current day is one of the usual days the user goes shopping: if true, the application checks the availability of one of the usual slots of the user and then, if it finds something, it notifies the user. Finally, the user can decide whether to accept or refuse the proposal.

2.3 User Characteristics

The actors of the application are the following:

- User: someone who downloads the application on his device and wants to avoid wasting time waiting to enter to supermarket. User also want to be safe in this period of pandemic due to Coronavirus and so use the app reducing drastically crowds.
- Non-user: customer of the chain of supermarkets, that does not utilize the application, either because he does not want to/cannot install the application on the smartphone or because he does not know how to install/use it.
- Supermarket Manager: director of the supermarket that examines statistics produced by the system that builds them retrieving data form databases.

2.4 Domain Assumptions

D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D3	General entrances of users are saved in their own smartphones
D4	Each supermarket provides non-users a non-digital QR code
D5	QR codes are readable and scannable
D6	The internet connection works properly
D7	Each user has a smartphone
D8	Each smartphone has a GPS sensor
D9	GPS is active while the application is running
D10	GPS provides the exact position with a maximum error of 100m
D11	The application is chain-dependent from only one supermarket chain
D12	Timetables of supermarkets are known
D13	The session of generation of a general entrance lasts at most 10 minutes
D14	Data exchanged between server and clients are not corrupted
D15	Line-ups of supermarkets are correctly registered in the database
D16	Each supermarket has limited capacity
D17	Each supermarket accepts a limited number of non-users
D18	User cannot be in more than one supermarket simultaneously
D19	Each general entrance contains one and only one supermarket
D20	When QR code is scanned to exit, its status changes from VALID to USED

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

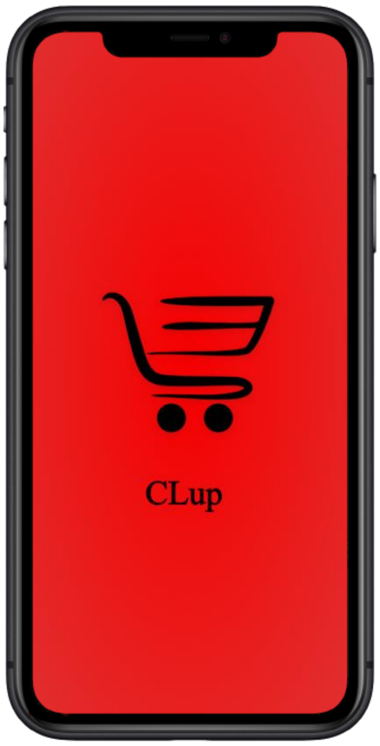
The application customer-side must have a simple and immediate GUI because customers of a supermarket are of all ages and the older one could be not very expert in technology. The device must be able to provide position and communicate using internet connection; therefore, a smartphone is the suitable device.

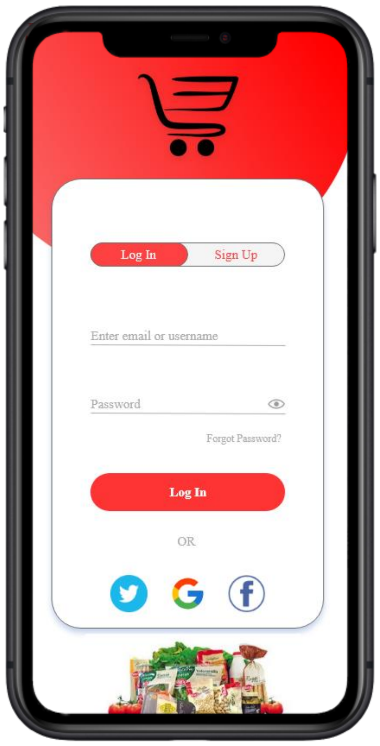
The application manager-side did not need a simple GUI, but however it should be immediate to provide information and statistic rapidly. It requires services delivered by a central server and, therefore, a computer with a browser to run a WebApp is a suitable device.

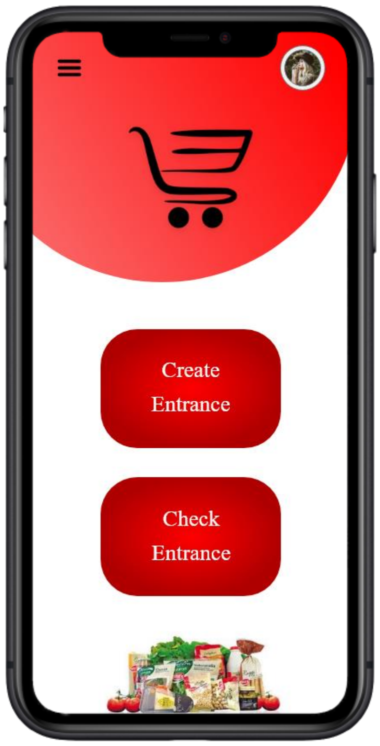
The following mock-ups give the basic idea of what:

- The icon of the application looks like (customer-side)
- The home page of the mobile application looks like (customer-side)
- The home page of the WebApp looks like (manager-side)

More detailed mock-ups will be presented in the Design Document (DD) part for User Interfaces.







3.1.2 Hardware Interfaces

On the user side, the functionalities addressed to him/her require a GPS sensor to calculate the position relative to supermarkets: as a consequence, the hardware interface he/she should own and use is a smartphone (with a working GPS sensor) to download, install and run the application.

On the supermarket manager side, the functionalities can be exploited only through a WebApp. Therefore, his/her office should be provided with a computer.

Finally, servers are required to allow the distributed system to work properly.

3.1.3 Software Interfaces

On the bare functionality side, the software uses the following external interfaces:

- Geographic Map: the software uses a public API to provide to the user his position in real time to calculate the time he would spend in reaching the supermarket and a map of the area to identify the position of supermarkets.
- Connection Protocol: the software uses specific rules to communicate with CLup servers. This protocol includes rules to:
    1. Send the request to line-up in a supermarket
    2. Receive ticket QR code after validation of request
    3. Send the request to book a visit in a supermarket
    4. Receive visit QR code after validation of request
    5. Notify possible free slots that can attract the user
    6. Send the request to access services of data mining (supermarket manager only)
    7. Receive client-side the services of data mining (supermarket manager only)
- Data Mining: the software uses an internal centralized system of data mining that can classify data and organize it providing to the supermarket manager the statistics he wants to analyze.

3.1.4 Communication Interfaces

The devices are connected to CLup servers through internet.

3.2 Functional Requirements

3.2.1 List of Requirements

Note: some requirements contain the following notation:

- *p = number of people inside the supermarket*
- *c = capacity of supermarket*
- *cu = number of slots reserved to everyone (cu < c, since we are reserving some slots exclusively as a fallback option for people that cannot access the required technology)*
- *u = number of people in the user line-up*

R1	User is registered
R2	User has to select supermarket from a given list
R3	Software should allow user to insert the time he would spend in the supermarket
R4	Software should allow user to insert the categories of products he would purchase in the supermarket
R5	User has to insert the means of transport he would use to reach the supermarket from a given list
R6	Customer has to scan QR code to enter
R7	QR code has to be valid
R8	If p < cu, the first user in the user queue can enter
R9	If a slot reserved to all becomes free, the first user in the user queue can enter
R10	Supermarket has to give QR to non-user
R11	If p < c AND u = 0, the first non-user in non-user queue can enter
R12	If a slot reserved for non-users becomes free, the first non-user in non-user queue can enter

R13	User has to select arrival time
R14	If possible, Software should estimate the time user would spend in the supermarket
R15	User has to select the time he would spend in the supermarket
R16	At least one valid ticket has to have been created
R17	User has to select the ticket he wants to delete
R18	Software must remove the ticket from the queue where it was placed
R19	Software must modify the queue where the ticket was placed
R20	Software must generate a valid ticket
R21	Software must update the waiting times of all other ticket of the queue of the supermarket
R22	One VALID visit has to be booked
R23	User has to select the visit he wants to delete
R24	Software must remove the visit from the scheduled ones of the supermarket
R25	Software must receive information from supermarket
R26	User cannot reserve in the past
R27	Connection must stay up for the duration of the operation
R28	Data of users are stored in databases
R29	Software tracks incoming and outcoming fluxes of customers of supermarket

R30	Software should check if more advantageous options than the current one are available
R31	Software should suggest alternatives found
R32	Software should allow user to choose an alternative
R33	If user chooses an alternative, software must modify the ticket with updated information
R34	If user chooses an alternative, software must modify the visit with updated information
R35	Software should mine information of the user about his usual slots
R36	If the current day is one of the days user usually goes to supermarket AND there is at least one slot available, software should notify user with slots available of that day
R37	If during the current day there is available at least one of the usual time slots of the user, software should notify user with that/those slot/s
R38	If user accepts one of the suggestions, software should create a valid ticket with all the information
R39	Data received are not corrupted
R40	Software stores data about fluxes in databases
R41	Supermarket managers have access to data of all databases of the supermarket and of users
R42	Software has to analyze data
R43	Software has to build statistics
R44	Data of VALID and USED tickets are stored
R45	Data of VALID and USED visits are stored
R46	Software should not allow user to generate a general entrance while there is another one still valid

R47	Customer has to scan QR code to exit

3.2.2 Mapping

GOALS	DOMAIN ASSUMPTIONS	REQUIREMENTS
G1	D1, D2, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16, D18, D19	R1, R2, R3, R4, R5, R20, R26, R27, R28, R44, R46
G2	D1, D2, D3, D5, D7, D14, D15, D16, D18, D19	R1, R6, R7, R8, R9, R28, R29, R44, R45
G3	D1, D4, D5, D15, D16, D17, D19	R6, R7, R10, R11, R12, R28, R29
G4	D1, D2, D6, D7, D8, D9, D10, D11, D12, D13, D14, D16, D18, D19	R1, R2, R3, R4, R13, R14, R15, R20, R26, R27, R28, R45, R46
G5	D1, D2, D3, D6, D7, D14, D15	R1, R7, R16
G6	D1, D2, D3, D6, D7, D14, D15, D19	R1, R16, R17, R18, R19, R21, R27, R28, R44
G7	D1, D2, D3, D6, D7, D14	R1, R7, R22
G8	D1, D2, D3, D6, D7, D14, D19	R1, R22, R23, R24, R27, R28, R45
G9	D6, D7, D8, D9, D10, D11, D12, D14, D15, D16, D19	R1, R25, R28, R30, R31, R32, R33, R39, R44
G10	D6, D7, D8, D9, D10, D11, D12, D14, D16, D19	R1, R25, R28, R30, R31, R32, R34, R39, R45
G11	D6, D7, D8, D9, D10, D11, D12, D14, D16, D18	R1, R25, R27, R28, R35, R36, R37, R38, R44, R45
G12	D6, D11, D14	R28, R29, R40, R41, R42, R43, R44, R45
G13	D1, D2, D3, D4, D5, D7, D14, D19, D20	R1, R10, R44, R45, R47

G1	ALLOW USER TO CREATE A TICKET
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D6	The internet connection works properly
D7	Each user has a smartphone
D8	Each smartphone has a GPS sensor
D9	GPS is active while the application is running
D10	GPS provides the exact position with a maximum error of 100m
D11	The application is chain-dependent from only one supermarket chain
D12	Timetables of supermarkets are known
D13	The session of generation of a general entrance lasts at most 10 minutes
D14	Data exchanged between server and clients are not corrupted
D15	Line-ups of supermarkets are correctly registered in the database
D16	Each supermarket has limited capacity
D18	User can visit only one supermarket at a time
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R2	User has to select supermarket from a given list
R3	Software should allow user to insert the time he would spend in the supermarket
R4	Software should allow user to insert the categories of products he would purchase in the supermarket
R5	User has to insert the means of transport he would use to reach the supermarket from a given list
R20	Software must generate a valid ticket
R26	User cannot reserve in the past
R27	Connection must stay up for the duration of the operation

R28	Data of users are stored in databases
R44	Data of VALID and USED tickets are stored
R46	Software should not allow user to generate a general entrance while there is another one still valid

G2	ALLOW USER TO ENTER THE SUPERMARKET
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D3	General entrances of users are saved in their own smartphones
D5	QR codes are readable and scannable
D7	Each user has a smartphone
D14	Data exchanged between server and clients are not corrupted
D15	Line-ups of supermarkets are correctly registered in the database
D16	Each supermarket has limited capacity
D18	User cannot be in more than one supermarket simultaneously
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R6	Customer has to scan QR code to enter
R7	QR code has to be valid
R8	If p < cu, the first user in the user queue can enter
R9	If a slot reserved to all becomes free, the first user in the user queue can enter
R28	Data of users are stored in databases
R29	Software tracks incoming and outcoming fluxes of customers of supermarket
R44	Data of VALID and USED tickets are stored
R45	Data of VALID and USED visits are stored

G3	ALLOW NON-USER TO ENTER THE SUPERMARKET
D1	Each general entrance owns one unique QR code
D4	Each supermarket provides non-users a non-digital QR code
D5	QR codes are readable and scannable
D15	Line-ups of supermarkets are correctly registered in the database
D16	Each supermarket has limited capacity
D17	Each supermarket accepts a limited number of non-users
D19	Each general entrance contains one and only one supermarket
R6	Customer has to scan QR code to enter
R7	QR code has to be valid
R10	Supermarket has to give QR to non-user
R11	If p < c AND u = 0, the first non-user in non-user queue can enter
R12	If a slot reserved for non-users becomes free, the first non-user in non-user queue can enter
R28	Data of users are stored in databases
R29	Software tracks incoming and outcoming fluxes of customers of supermarket

G4	ALLOW USER TO BOOK A VISIT
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D6	The internet connection works properly
D7	Each user has a smartphone
D8	Each smartphone has a GPS sensor
D9	GPS is active while the application is running
D10	GPS provides the exact position with a maximum error of 100m
D11	The application is chain-dependent from only one supermarket chain

D12	Timetables of supermarkets are known
D13	The session of generation of a general entrance lasts at most 10 minutes
D14	Data exchanged between server and clients are not corrupted
D16	Each supermarket has limited capacity
D18	User cannot be in more than one supermarket simultaneously
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R2	User has to select supermarket from a given list
R3	Software should allow user to insert the time he would spend in the supermarket
R4	Software should allow user to insert the categories of products he would purchase in the supermarket
R13	User has to select arrival time
R14	If possible, Software should estimate the time user would spend in the supermarket
R15	User has to select the time he would spend in the supermarket
R20	Software must generate a valid ticket
R26	User cannot reserve in the past
R27	Connection must stay up for the duration of the operation
R28	Data of users are stored in databases
R45	Data of VALID and USED visits are stored
R46	Software should not allow user to generate a general entrance while there is another one still valid

G5	ALLOW USER TO CHECK A TICKET
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED

D3	General entrances of users are saved in their own smartphones
D6	The internet connection works properly
D7	Each user has a smartphone
D14	Data exchanged between server and clients are not corrupted
D15	Line-ups of supermarkets are correctly registered in the database
R1	User is registered
R7	QR code has to be valid
R16	At least one valid ticket has to have been created

G6	ALLOW USER TO DELETE A TICKET
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D3	General entrances of users are saved in their own smartphones
D6	The internet connection works properly
D7	Each user has a smartphone
D14	Data exchanged between server and clients are not corrupted
D15	Line-ups of supermarkets are correctly registered in the database
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R16	At least one valid ticket has to have been created
R17	User has to select the ticket he wants to delete
R18	Software must remove the ticket from the queue where it was placed
R19	Software must modify the queue where the ticket was placed
R21	Software must update the waiting times of all other ticket of the queue of the supermarket
R27	Connection must stay up for the duration of the operation
R28	Data of users are stored in databases

R44	Data of VALID and USED tickets are stored

G7	ALLOW USER TO CHECK A VISIT
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D3	General entrances of users are saved in their own smartphones
D6	The internet connection works properly
D7	Each user has a smartphone
D14	Data exchanged between server and clients are not corrupted
R1	User is registered
R7	QR code has to be valid
R22	One VALID visit has to be booked

G8	ALLOW USER TO DELETE A VISIT
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D3	General entrances of users are saved in their own smartphones
D6	The internet connection works properly
D7	Each user has a smartphone
D14	Data exchanged between server and clients are not corrupted
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R22	One VALID visit has to be booked
R23	User has to select the visit he wants to delete
R24	Software must remove the visit from the scheduled ones of the supermarket

R27	Connection must stay up for the duration of the operation
R28	Data of users are stored in databases
R45	Data of VALID and USED visits are stored

G9	SUGGEST ALTERNATIVES FOR TICKETS TO USER
D6	The internet connection works properly
D7	Each user has a smartphone
D8	Each smartphone has a GPS sensor
D9	GPS is active while the application is running
D10	GPS provides the exact position with a maximum error of 100m
D11	The application is chain-dependent from only one supermarket chain
D12	Timetables of supermarkets are known
D14	Data exchanged between server and clients are not corrupted
D15	Line-ups of supermarkets are correctly registered in the database
D16	Each supermarket has limited capacity
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R25	Software must receive information from supermarket
R28	Data of users are stored in databases
R30	Software should check if more advantageous options than the current one is available
R31	Software should suggest alternatives found
R32	Software should allow user to choose an alternative
R33	If user chooses an alternative, software must modify the ticket with updated information
R39	Data received are not corrupted
R44	Data of VALID and USED tickets are stored

G10	SUGGEST ALTERNATIVES FOR VISITS TO USER
D6	The internet connection works properly
D7	Each user has a smartphone
D8	Each smartphone has a GPS sensor
D9	GPS is active while the application is running
D10	GPS provides the exact position with a maximum error of 100m
D11	The application is chain-dependent from only one supermarket chain
D12	Timetables of supermarkets are known
D14	Data exchanged between server and clients are not corrupted
D16	Each supermarket has limited capacity
D19	Each general entrance contains one and only one supermarket
R1	User is registered
R25	Software must receive information from supermarket
R28	Data of users are stored in databases
R30	Software should check if more advantageous options, than the current one, are available
R31	Software should suggest alternatives found
R32	Software should allow user to choose an alternative
R34	If user chooses an alternative, software must modify the visit with updated information
R39	Data received are not corrupted
R45	Data of VALID and USED visits are stored

G11	SEND NOTIFICATION TO USER ABOUT FREE AVAILABLE SLOTS
D6	The internet connection works properly
D7	Each user has a smartphone
D8	Each smartphone has a GPS sensor
D9	GPS is active while the application is running

D10	GPS provides the exact position with a maximum error of 100m
D11	The application is chain-dependent from only one supermarket chain
D12	Timetables of supermarkets are known
D14	Data exchanged between server and clients are not corrupted
D16	Each supermarket has limited capacity
D18	User cannot be in more than one supermarket simultaneously
R1	User is registered
R25	Software must receive information from supermarket
R27	Connection must stay up for the duration of the operation
R28	Data of users are stored in databases
R35	Software should mine information of the user about his usual slots
R36	If the current day is one of the days user usually goes to supermarket AND there is at least one slot available, software should notify user with slots available of that day
R37	If during the current day there is available at least one of the usual time slots of the user, software should notify user with that/those slot/s
R38	If user accepts one of the suggestions, software should create a valid ticket with all the information
R44	Data of VALID and USED tickets are stored
R45	Data of VALID and USED visits are stored

G12	BUILD STATISTICS USING COLLECTED DATA
D6	The internet connection works properly
D11	The application is chain-dependent from only one supermarket chain
D14	Data exchanged between server and clients are not corrupted
R28	Data of users are stored in databases
R29	Software tracks incoming and outcoming fluxes of customers of supermarket
R40	Software stores data about fluxes in databases

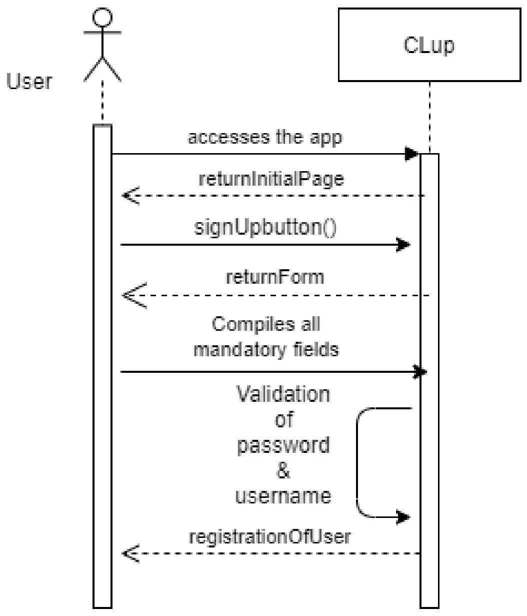
R41	Supermarket managers have access to data of all databases of the supermarket and of users
R42	Software has to analyze data
R43	Software has to build statistics
R44	Data of VALID and USED tickets are stored
R45	Data of VALID and USED visits are stored

G13	ALLOW CUSTOMER TO EXIT THE SUPERMARKET
D1	Each general entrance owns one unique QR code
D2	If QR code is not scanned within 15 minutes the estimated entrance, its status will change from VALID to EXPIRED
D3	General entrances of users are saved in their own smartphones
D4	Each supermarket provides non-users a non-digital QR code
D5	QR codes are readable and scannable
D7	Each user has a smartphone
D14	Data exchanged between server and clients are not corrupted
D19	Each general entrance contains one and only one supermarket
D20	QR code status is USED after it is scanned to exit from the supermarket
R1	User is registered
R10	Supermarket has to give QR to non-user
R44	Data of VALID and USED tickets are stored
R45	Data of VALID and USED visits are stored
R47	Customer has to scan QR code to exit

3.2.3 Use Cases & Sequence diagrams

1. Registration of User

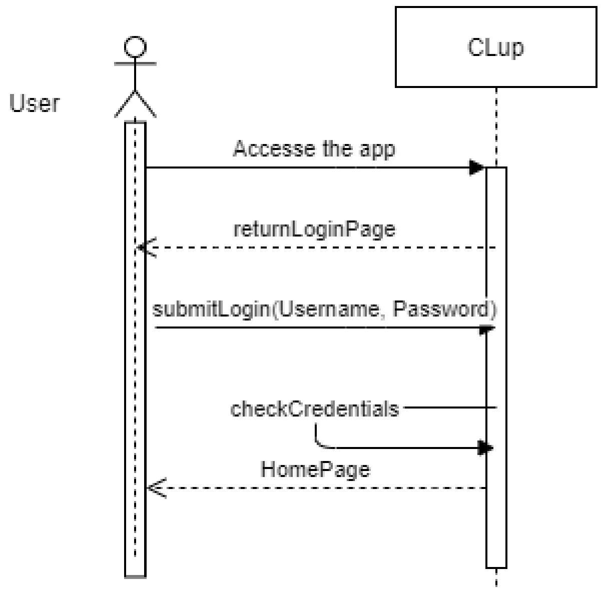
Name	REGISTRATION OF USER
Actors	User
Entry Conditions	User has the internet connection available on its device The internet connection is available
Event Flow	1) User accesses the App through its device 2) User visualizes the initial page of the App 3) User clicks on "Sign up" button 4) User compiles all mandatory field 5) User clicks on "Allow CLup to access the GPS position" button 6) The system confirms the registration of the User
Exit Conditions	User is successfully registered to the application
Exceptions	1) User is already present in the system 2) User did not fill up all mandatory field with valid data If one of the two situations happens, the application will throw an error message and will return to the registration form page



2. Login of the User

Name	LOGIN OF THE USER
Actors	User

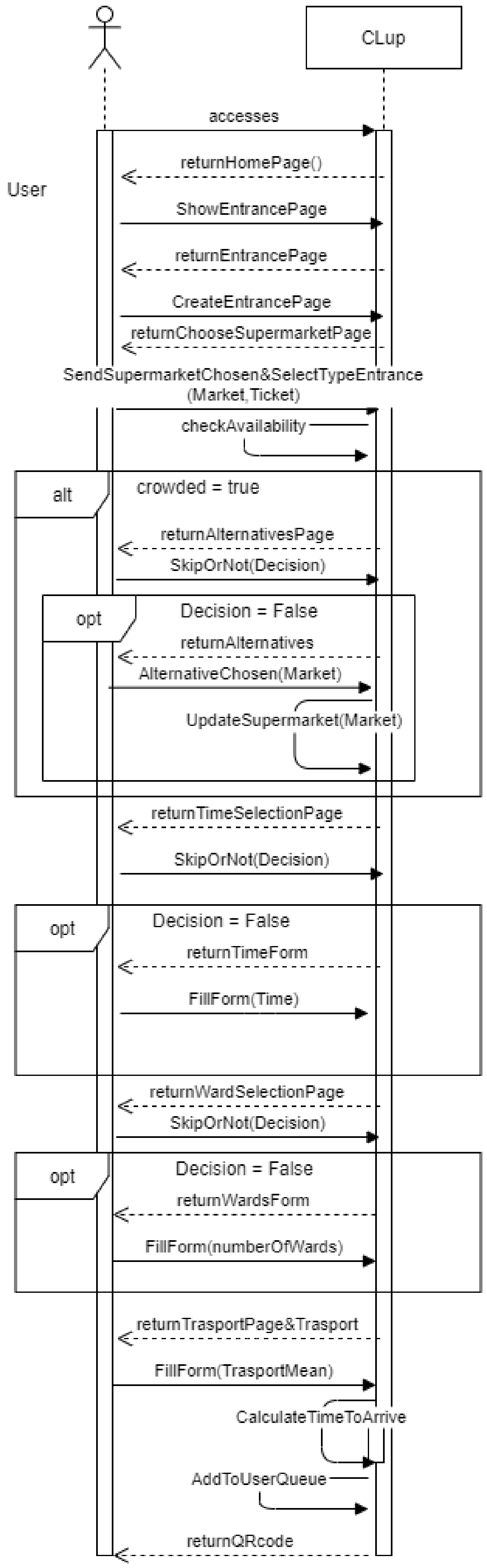
Entry Conditions	User is already registered to the application service Internet connection is available
Event Flow	1) User accesses the App through its device 2) User compiles the fields "Username" and "Password" 3) User clicks on "Login" button 4) The system opens the application home page 5) If there are some suggestion to view, User receives a notification
Exit Conditions	User has access to the services of CLup
Exceptions	1) User enters invalid Username 2) User enters invalid Password If one of the above conditions is detected, the application notifies the User taking he/she to the login screen



3. Create a Ticket

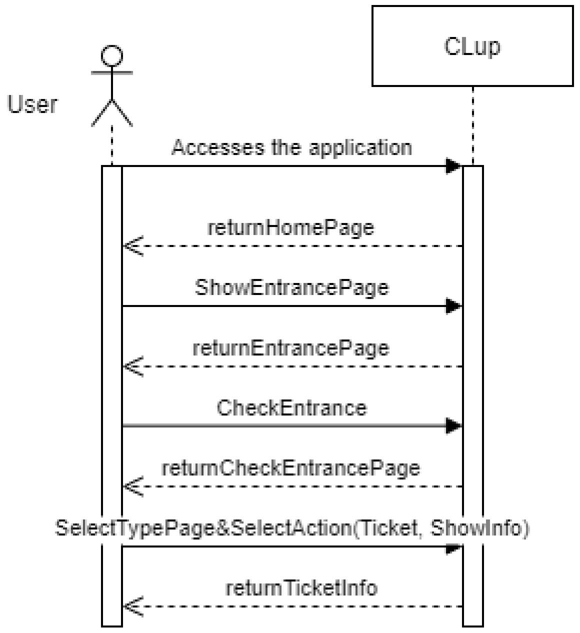
Name	CREATE A TICKET
Actors	User
Entry Conditions	User has already installed the application on its device; user wants to go to the supermarket; The internet connection is available; user already logged;

Event Flow	1) User opens the application on its device 2) User clicks on "Create Entrance" button 3) User clicks on "Create a Ticket" button and in the same page 4) User selects the supermarket where he/she wants to go 5) The System checks the availability of that supermarket 5.1) If the waiting-time of the selected supermarket is very high, then the software will show other alternatives 5.2) User can choose to select the alternative he/she prefers or continue with the selected supermarket 6) User can insert the time he/she thinks he/she will spend in the supermarket or skip this request 7) User can insert how many areas of the supermarket he/she is expected to visit or skip this request 8) User inserts how he/she will reach the supermarket 9) The System calculates how much time User needs to arrive to the supermarket 10) User receives the QR code 11) User receives info about the ticket
Exit Conditions	User receives correctly QR code and info about ticket AND a valid Ticket is created
Exceptions	1) User fills incorrectly one or more fields 2) User leaves one or more mandatory field incomplete If one of the above exceptions is detected, the application will take back User to the Home page. 3) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later" 4) Time expected to enter is after closing of the supermarket If the above condition is detected, the application displays the message "The supermarket could close before your entry!" 5) The application is forcibly closed before the operation is completer If the above condition is detected, the operation is aborted


CLup
User
accesses
returnHomePage()
ShowEntrancePage
returnEntrancePage
CreateEntrancePage
returnChooseSupermarketPage
SendSupermarketChosen&SelectTypeEntrance
(Market,Ticket)
checkAvailability
alt
crowded = true
returnAlternativesPage
SkipOrNot(Decision)
opt
Decision = False
returnAlternatives
AlternativeChosen(Market)
UpdateSupermarket(Market)
returnTimeSelectionPage
SkipOrNot(Decision)
opt
Decision = False
returnTimeForm
FillForm(Time)
returnWardSelectionPage
SkipOrNot(Decision)
opt
Decision = False
returnWardsForm
FillForm(numberOfWards)
returnTrasportPage&Trasport
FillForm(TrasportMean)
CalculateTimeToArrive
AddToUserQueue
returnQRcode

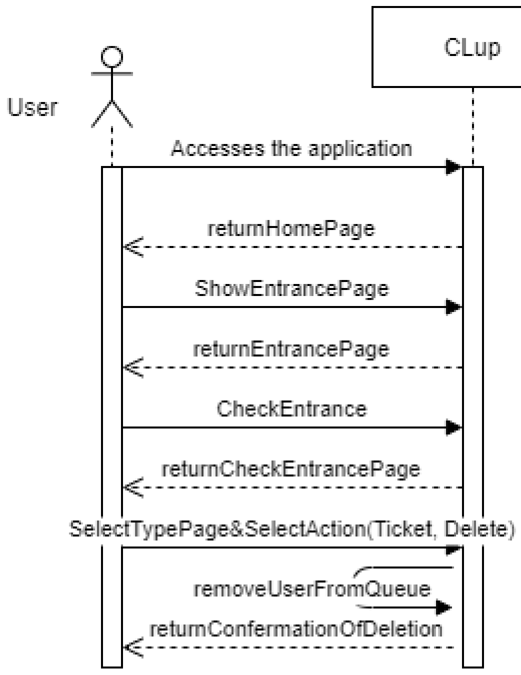
4. Check a ticket

Name	CHECK A TICKET
Actors	User
Entry Conditions	User has already installed the application on its device; The internet connection is available; user already logged;
Event Flow	1) User opens the application on its device 2) User clicks on "Check Entrances" button 3) User clicks on "Tickets" button and in the same page 4) User clicks on "Ticket Info" option
Exit Conditions	The ticket with all its information
Exceptions	1) User does not have any ticket If the exception is detected, the application will display a blank white screen with this message "No tickets" 2) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later"



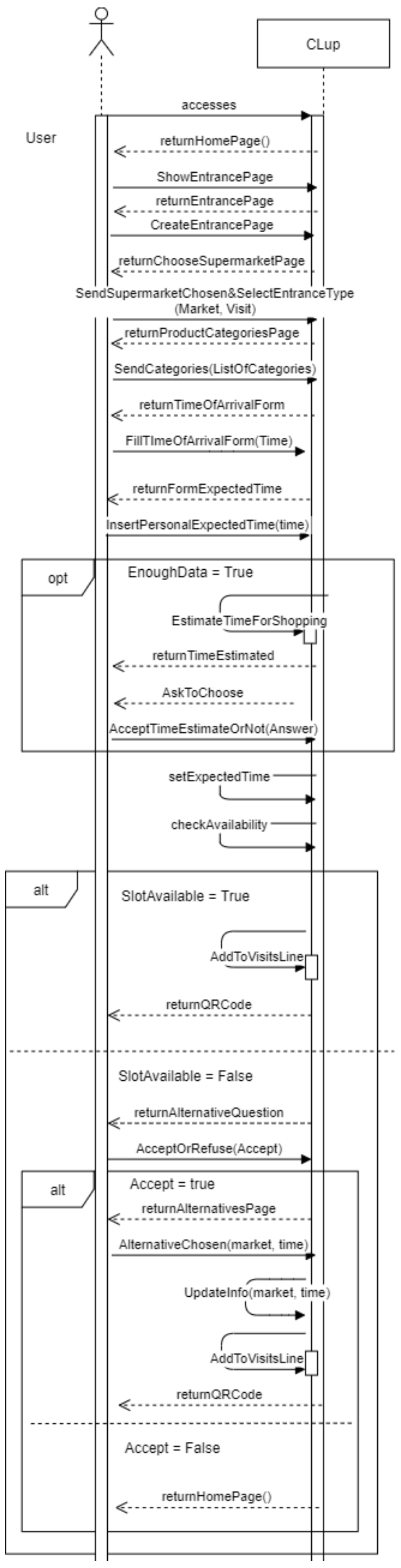
5. Delete a ticket

Name	DELETE A TICKET
Actors	User
Entry Conditions	User has already installed the application on its device; User has a valid ticket registered on the smartphone; The internet connection is available; user already logged;
Event Flow	1) User opens the application on its device 2) User clicks on "Check Entrances" button 3) User clicks on "Tickets" button and in the same page 4) User selects the "Delete" option
Exit Conditions	Ticket is deleted
Exceptions	1) User does not have any ticket If the exception is detected, the application will display a blank white screen with this message "No tickets" 2) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later"



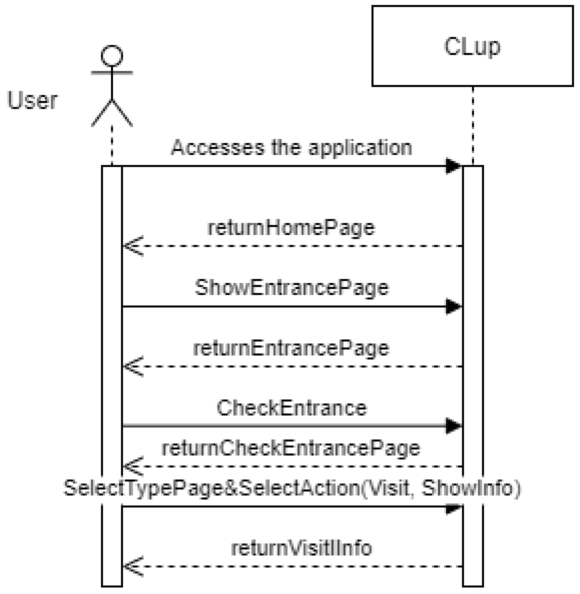
6. User books a visit

Name	USER BOOKS A VISIT
Actors	User
Entry Conditions	User has already installed the application on its device; user wants to schedule a visit to the supermarket; the internet connection is available; user already logged;
Event Flow	1) User opens the application on its device 2) User clicks on "Create Entrance" button 3) User clicks on "Book a Visit" button and in the same page 4) User selects the supermarket he/she wants to go to 5) User can select categories of products to buy 6) User inserts arrival time 7) If the system has sufficient data, it estimates expected time to spend in the supermarket 8) User can accept estimated time or insert expected time or skip this request 9) The System checks the availability of that supermarket 9.1) If the selected slot is not available, then the software will show other alternatives including both different time slots for the same supermarket AND same time slot but in different supermarkets 9.2) User chooses the alternatives he prefers or quits 10) User receives QR code
Exit Conditions	User receives correctly QR code and info about booking
Exceptions	1) User fills incorrectly one or more fields 2) User leaves one or more mandatory field incomplete If one of the above exceptions is detected, the application will take back User to the Home page. 3) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later" 4) Time expected to enter is after closing of the supermarket


CLup
User
accesses
returnHomePage()
ShowEntrancePage
returnEntrancePage
CreateEntrancePage
returnChooseSupermarketPage
SendSupermarketChosen&SelectEntranceType (Market, Visit)
returnProductCategoriesPage
SendCategories(ListOfCategories)
returnTimeOfArrivalForm
FillTImeOfArrivalForm(Time)
returnFormExpectedTime
InsertPersonalExpectedTime(time)
opt
EnoughData = True
EstimateTimeForShopping
returnTimeEstimated
AskToChoose
AcceptTimeEstimateOrNot(Answer)
setExpectedTime
checkAvailability
alt
SlotAvailable = True
AddToVisitsLine
returnQRCode
SlotAvailable = False
returnAlternativeQuestion
AcceptOrRefuse(Accept)
alt
Accept = true
returnAlternativesPage
AlternativeChosen(market, time)
UpdateInfo(market, time)
AddToVisitsLine
returnQRCode
Accept = False
returnHomePage()

7. Check a visit

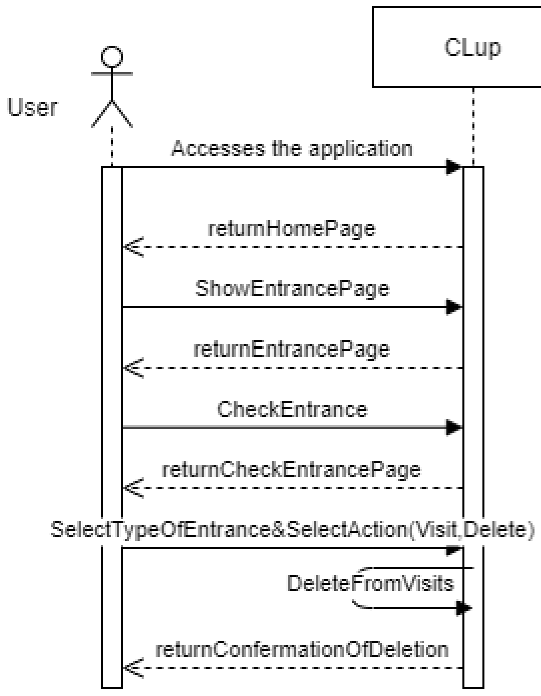
Name	CHECK A VISIT
Actors	User
Entry Conditions	User has already installed the application on its device; The internet connection is available; user already logged;
Event Flow	1) User opens the application on its device 2) User clicks on "Check Entrances" button 3) User clicks on "Visits" button and in the same page 4) User clicks on "Visit Info" button
Exit Conditions	The visit with all its information
Exceptions	1) User does not have any visit If the exception is detected, the application will display a blank white screen with this message "No Visits" 2) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later"



8. Delete a visit

Name	DELETE A VISIT
Actors	User

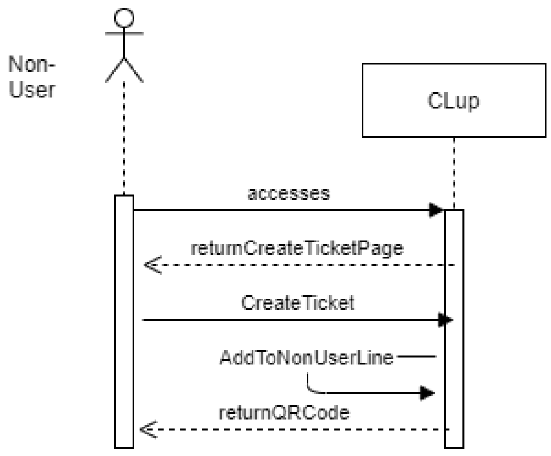
Entry Conditions	User has already installed the application on its device; user has a valid visit registered on the smartphone; The internet connection is available; user already logged;
Event Flow	1) User opens the application on its device 2) User clicks on "Check Entrances" button 3) User clicks on "Visits" button and in the same page 4) User clicks on "Delete Visit" option
Exit Conditions	Visit is deleted
Exceptions	1) User does not have any visit If the exception is detected, the application will display the message "No visit booked to be deleted" 2) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later"



9. Create a Ticket for non-user

Name	CREATE A TICKET FOR NON-USER
Actors	Non-User
Entry Conditions	Non-User goes to the supermarket

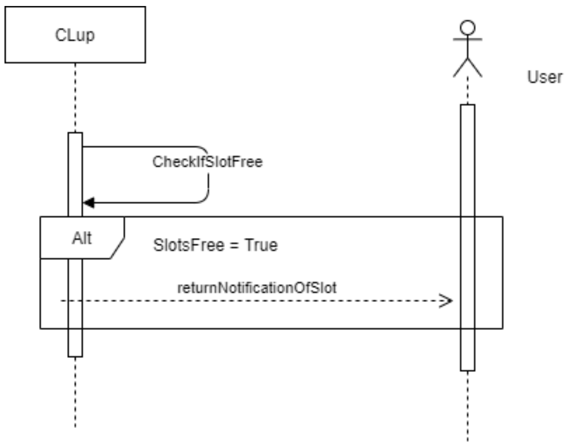
Event Flow	1) Non-user goes up to the ticket machine 2) Non-user clicks on "Create a Ticket" button 3) The system will insert non-user in the non-user queue 4) Non-user receives the QR code
Exit Conditions	Non-User receives correctly QR code and info about ticket
Exceptions	1) The internet connection is not available If the above condition is detected, the application displays the message "No Internet Connection Available! Try again Later" 2) Time expected to enter is after closing of the supermarket If the above condition is detected, the application displays the message "The supermarket could close before your entry!"



10. Notify available slots

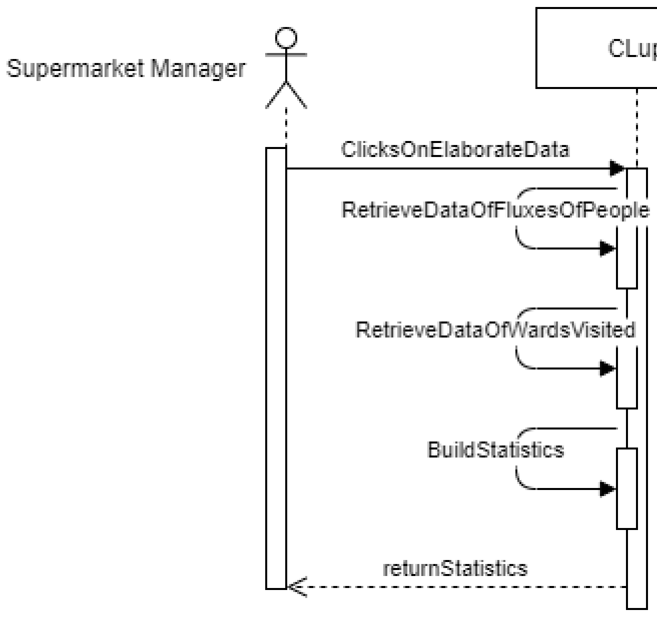
Name	**NOTIFY AVAILABLE SLOTS**
Actors	User

Entry Conditions	The system has collected enough data to build statistics; User has already booked a visit once; User has no valid general entrance created; The internet connection is available
Event Flow	1) One or more slots for "book a visit" frequently booked by the user is free 2) The application notifies it to the user
Exit Conditions	The system notifies the user of the available slots
Exceptions	If the system has not enough data available to generate statistics, it will not generate any notifications



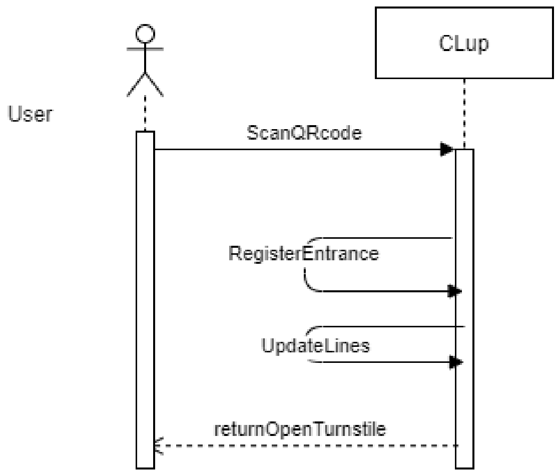
11. Build Statistics

Name	BUILD STATISTICS
Actors	Supermarket Manager
Entry Conditions	The System has collected enough data to build statistics
Event Flow	1) Supermarket Manager clicks on "Elaborate Data" 2) The system retrieves data of incoming and outcoming fluxes of people 3) The system retrieves data of wards visited in the supermarket 4) The system builds statistics
Exit Conditions	Statistics can be analysed
Exceptions	If the system has not enough data available to generate statistics, it will generate an error



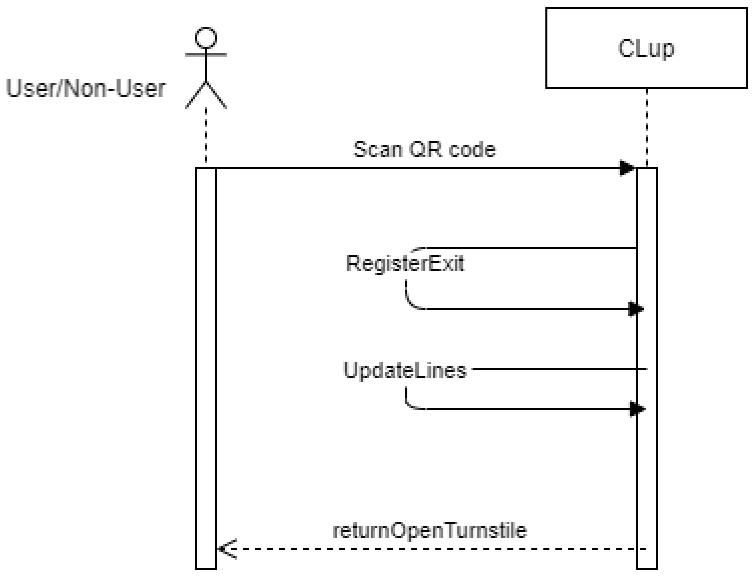
12. Customer enters in the supermarket

Name	**CUSTOMER ENTERS THE SUPERMARKET**
Actors	User/Non-User (Customer)
Entry Conditions	The system has calculated the next customer that can enter; The next customer has a valid ticket/visit
Event Flow	1) Customer scans the QR code 2) The system registers the entrance 3) The system updates the lines of the supermarket
Exit Conditions	The turnstile opens; the customer can enter; lines are updated
Exceptions	If the QR code is not valid, the turnstile will not open and is generated an error message.

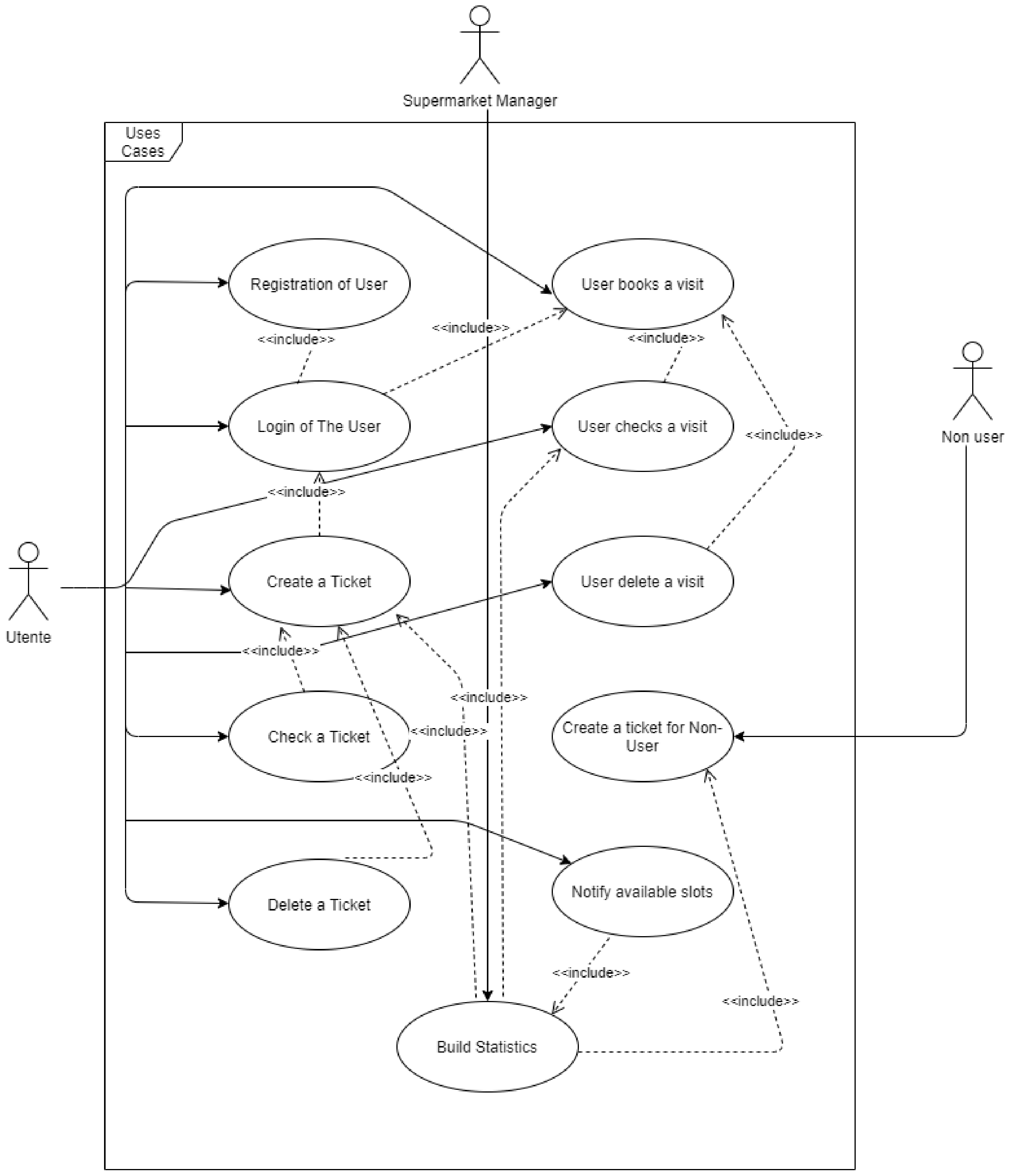


13. Customer exits from the supermarket

Name	CUSTOMER EXITS FROM THE SUPERMARKET
Actors	User/Non-User (Customer)
Entry Conditions	The customer has finished to shop;
Event Flow	1) Customer scans the QR code 2) The system registers the exit 3) The system updates the lines of the supermarket
Exit Conditions	The turnstile opens; the customer can exit; lines are updated
Exceptions	none



3.2.4 Use Cases Diagram


Supermarket Manager
Uses Cases
Registration of User
User books a visit
<<include>>
<<include>>
<<include>>
Login of The User
User checks a visit
<<include>>
Non user
<<include>>
Create a Ticket
User delete a visit
Utente
<<include>>
<<include>>
Check a Ticket
<<include>>
Create a ticket for Non-User
<<include>>
Notify available slots
Delete a Ticket
<<include>>
<<include>>
Build Statistics

3.2.5 Scenarios

Sigismondo's digital transformation

Sigismondo is an old man that is not able to use any technological devices. Indeed, he does not have a smartphone and consequently he does not use CLup. Therefore, he goes to the supermarket and takes a ticket with the help on an employee and sometimes he has to wait outside while other people arriving after him enter before he does. So, one day, he asks to an employee of the supermarket how that can happen and learns about the CLup app. After this extraordinary discovery, he decides to ask his grandson to help him with this unknown technology. After some day of practice with his new smartphone and with the application, he successfully creates a ticket and enjoys going to the supermarket without being overcome by younger people.

Fatto in casa da Marisa

Marisa is a girl who always wants to try cooking new recipes and after watching a cooking tv program ("Fatto in casa da Benedetta") decides to recreate that recipe but she does not have all the needed ingredients. Therefore, she takes her smartphone, she opens CLup application and finally she creates a ticket choosing her favourite supermarket and filling other fields in the form. After ten minutes, the application notifies her that she should leave to arrive at the supermarket in time. When she arrives at the supermarket, she opens the application and checks the ticket she will use to enter; then she scans the QR code associated at the ticket machine just outside the supermarket and she can enter to buy the needed ingredients.

Alfonso and the diapers

Alfonso is a scatter-brained young dad and now his wife is criticizing him because he forgot to buy diapers for their young son. So, he immediately tries to remedy and starts creating a ticket. After he selects the nearest supermarket, the system checks its availability and detects that its waiting-time is quite long. Consequently, the system suggests some alternative supermarkets with a shorter waiting-time and Alfonso chooses the most convenient alternative. Within thirty minutes, Alfonso is back home with the diapers for his baby.

Thuany the businesswoman

Thuany is a very busy and organized woman and, therefore, she always optimizes her time. Every time she notices or remembers something missing that she has to buy at the supermarket, she writes it on the notes. Moreover, she is very methodical in what she does; indeed, she usually goes to the supermarket in the evening after work. The system learns about this regularity and consequently it sends Thuany notifications about free available slots in her usual supermarket at her usual time. When she receives this notification, she checks her list on the notes and if needed she creates the ticket the system suggested. In this way, she saves a lot of time.

Chef Marcello and the Christmas dinner

Marcello is confident amateur chef that enjoys cooking for his family. The Christmas dinner is a wonderful day to astonish his family with his particular dishes, but he needs fresh ingredients (e.g. fish). Nevertheless, he knows that during this period there is always a long waiting-time to enter in the supermarket; therefore, he decides to book a visit some days before to be sure not to ruin the dinner.

Giovanna the Stakhanovite

Giovanna has booked a visit in the supermarket in the evening because she finishes late to work. That day, she realizes at some point that she has a lot of work to do and she will not be able to go to the supermarket. Therefore, she deletes the visit she created to book a new one for the day after.

Giuseppe and Murphy's law OR Giuseppe's law

Giuseppe is a very unlucky man, especially when there are deadlines to respect. Indeed, he creates the ticket and selects the car as means of transport to reach the supermarket. Unfortunately, as always, he realizes just before to leave that his car is broken and so decides to go to the supermarket with his old bike. Obviously when he arrives, he tries to scan the QR to enter but it is too late, and the ticket is no more valid. So, he decides to line-up as a non-customer. At least this time he is lucky and the non-user line-up is very short.

Andrea and his loyal customers

Andrea is the supermarket manager and he wants to analyze the affluences in his supermarket during this month. Therefore, he clicks on "Elaborate Data" to launch the system and let it retrieve data and mine the required information. The system builds the statistics and let Andrea learn about how many customers visited his supermarket, the most visited areas, the peak hours and his most loyal customers.

3.3 Performance Requirements - Non-Functional Requirements

The system should be able to send the QR code to the user not after 10 sec the request has been received. In addition, in this short time the request should be processed, the user inserted in the correct queue and the QR code correctly associated to the that user.

As regard changes in the queue (delay or anticipation), the system should be able to alert users in less than 60 seconds, recalculating their waiting-time and resolving any conflict.

As regard the notification of when a user should leave to arrive at the supermarket in time, the software should alert at least 10 minutes before the calculated departure time: in this way it can cover everyday accidents.

Moreover, it must be able to guarantee the simultaneous connection of a minimum base of 100.000 individuals.

3.4 Design Constraints - Non-Functional Requirements

3.4.1 Standards compliance

The code should follow the requirements contained in this document. Furthermore, its comments should be clear.

3.4.2 Hardware limitations

The software application requires a mobile device able to capture the position and to send data to the system via internet connection.

Supermarket managers do not need necessarily a mobile device, but they can use a computer connected to internet to build and view statistics.

Finally, the system requires a central server (composed by a reliable array of partitioned servers, in which each partition is dedicated to a supermarket of the chain) and a remote server for each supermarket (clone of the corresponding partition in the central one).

3.5 Software System Attributes - Non-Functional Requirements

3.5.1 Reliability and availability

The reliability of the system depends on its services; since the system offers services that are not related to emergency situations, it should be up for a 99% of times. This means that the MTTR (the time required to restore its functionalities) should be about 3.65 days per year. This is very affordable, since the system must be up only while the supermarket is open, instead, when it is closes, the system can be up to provide the functionality to book a visit. However, during the year there are many periods in which supermarkets are closed and an availability of 99% is more than enough to provide the correct functionalities with a low probability of unpleasant situations.

In order to guarantee this time of downtime, the system must have an appropriate infrastructure. Each supermarket will have its own server with the system and the database necessary to provide its functionalities. Moreover, the headquarter of the chain of supermarkets must have also a replica of the system and a full backup of every database of each supermarket: with this solution, if the system of one of the supermarkets fails, the system of the headquarter will temporarily replace it. Each supermarket and the headquarter must guarantee appropriate personnel able to recover the failure in an adequate amount of time.T

In case of failure, users must be notified within 15 minutes with a specific message addressed to the devices either via application, e-mail or phone number (if known). Moreover, an analogous message should be sent whenever the failure is fixed.

3.5.2 Security

Supermarket data stored in databases should be hashed to guarantee integrity and encrypted using a symmetric key algorithm. In addition, user data should be stored in a database with all fields in clear except for the password: it should be hashed with salt and the output will be saved in the corresponding field. Finally, also the user database should be encrypted with a symmetric key algorithm. In case of password recovery, the system should send an email to the user who made the request; the user will follow the instruction to change the password that will be replaced in the corresponding field of the database.

3.5.3 Maintainability

The target programming language of the software is Java Enterprise Edition (JEE) integrated with JPA (Java Persistence API) to manage databases. Code must be written following the guidelines in this document, the SOLID principles and maintaining a high level of abstraction. Comments covering all aspects of the code are mandatory to better understand the code and to help fixing failures. Finally, developer should include testing routine that covers at least 75% of the entire code, analysing its most fragile aspects, excluding software interfaces.

3.5.4 Portability

Since the software to be is a distributed system, it should run on different platforms (PCs, Web Server, mobile devices, etc.) with different operating systems including Windows, MacOS and all Linux releases. Moreover, it must support operating systems of mobile devices, including both Android and iOS. Finally, the web service, mostly exploited by supermarket managers, should correctly run on every browser (Chrome, Safari, Firefox etc.).

3.5.5 Scalability

This new software will be very helpful to manage the supermarket queue and save the time of customers that its use would increase rapidly: the system must be scalable without the need to modify core parts of the software for the expected rising number of users; it should continue work at least until 10 times the base of initial expected users (100.000 individuals), guaranteeing the same reaction time of 10 seconds. Finally, it should be very easy and not invasive add to the system a new supermarket.

3.6 Additional Specifications

3.6.1 Mandatory Fields

At the moment of the registration, the user will have to compile the following mandatory fields:

- E-mail
- Name
- Surname
- City, Province, CAP, State
- Phone Number

3.6.2 Means of transport to reach the supermarket

During the creation of a ticket, the user can choose from a list of predefined means of transport to reach the supermarket, which are specified below:

- On foot
- Car
- Bike
- Motorcycle
- Public Transportation

3.6.3 Types of categories of products

During the creation of a general entrance, the user can choose from a list of categories of products available in the supermarket. We report some examples of categories:

- Fruit and vegetables
- Fish
- Meat
- Gastronomy
- Frozen food
- Bakery
- Drinks
- Grocery

4. Formal Analysis with Alloy

4.1 Alloy Code

This final section is dedicated to the Alloy model of CLup software. It includes the main structure of the software to be, with all the main actors, instances and constraints it has to follow. The purpose of this model is to demonstrate that the designed system is feasible. Here there are the main points we wanted to demonstrate:

- Each user can have only one valid entrance at a time
- The system stores all data needed to its correct functioning
- Each valid entrance can be only in one line
- Notifications can exist only if there are stored data about that user
- Each supermarket cannot contain more customers than its maximum capacity
- All the supermarkets have the characteristics to be considered as such
- The system stores only positions of registered user
- The system correctly suggests available supermarkets during the creation of a ticket or a visit

Finally, we also created a complex situation that reproduces a real scenario to prove the correct behaviour of the software. We report the alloy code and, after that, all the predicates and assertions proved of the model.

```
/*
** in this model we assume that timetables of supermarkets are updated every
** midnight so that there are not timetables of past days and the first
** element of seq timetable is the timetable of the current day
** Also, we assume that this application will be used by only one chain of
** Supermarkets
*/


/* ** SIGNATURES ** */

/*
** Signatures of Position are empty because no further details are needed
** We only need to state that a position in different form another one
** and Alloy does it already
*/
abstract sig Position{}
sig SupermarketPosition extends Position {}
sig GPSPosition extends Position {}
```

```
/*
** Signature of Time with details because we need to verify if
** a time is precedent to other times
** There is also a fact that states the correctness of a time
*/
sig Time {
    hour: one Int,
    minute: one Int,
    second: one Int
} {
    hour > 0
    minute > 0
    second > 0
}

/*
** Signature of Date: it is empty for the same reason stated in
** comment of signature Position
*/
sig Date {} {
    all d: Date | some tt: Timetable | tt.date = d
}

/*
** Data represents the database of the system and contains data of
** registered users, supermarkets of the chain, valid/used entrances
*/
one sig Data {
    users: set User,
    supermarkets: set Supermarket,
    entrances: set Entrance
}

/*
** Signatures Customer modelizes the different behaviour of a registered
** customer (User) and unregistered ones (modelized as a Singleton)
*/
abstract sig Customer {}
sig User extends Customer {
    name: lone String,
    surname: lone String,
    position: one GPSPosition,
    activeEntrance: lone Entrance
} {
    activeEntrance != none implies (activeEntrance.customer = this and activeEntran
ce.state = VALID)
}
one sig nonUser extends Customer {}
```

```
/*
** Supermarket, with some constraints obvious
*/
sig Supermarket {
    name: lone String,
    capacity: one Int,
    position: one SupermarketPosition,
    timetable: seq Timetable,
    supermarketCategories: set Category,
    manager: one Manager,
    ticketMachine: one TicketMachine,
    lines: one Lines,
    statistics: one Statistics
} {
    #timetable > 0
    #supermarketCategories > 0
    !(timetable.hasDups)
    capacity > #lines.people or capacity = #lines.people
    ticketMachine.supermarket = this
}

/*
** Signatures completing the description of the Supermarket one
*/
sig Statistics {
    supermarket: one Supermarket
} {
    supermarket.statistics = this
}
sig Manager {
    supermarket: one Supermarket
} {
    supermarket.manager = this
}
sig TicketMachine {
    supermarket: one Supermarket
} {
    supermarket.ticketMachine = this
}

/*
** Signatures completing the desctiption of Entrance one
*/
sig QR {}
abstract sig EntranceState {}
one sig VALID extends EntranceState {}
one sig USED extends EntranceState {}
one sig EXPIRED extends EntranceState {}
```

```
/*
** signature that modelize the category of products in the supermarket
** each category must belong to at least one supermarket, otherwise
** it has no sense to exist (customers cannot purchase products of a
** category that does not exist in any supermarket)
*/
sig Category {} {
    all c: Category | some s: Supermarket | c in s.supermarketCategories
}

/*
** signature that modelize the vehicle that user should take to reach the
** supermarket, to help the system calculate the waiting time.
** each vahicle should be associated to one or more tickets
*/
sig Vehicle {} {
    all v: Vehicle | some t: Ticket | t.vehicle = v
}

/*
** Signature of possible entrances:
** 1) The main signature contiains all common attributes, important to
**    identify the entrance
** 2) Ticket signature extends Entrance one, adding its own specific
**    characteristic
** 3) Visit signature also adds some constraints, such as customer has
**    to be a registered one, or the consistency of the timeslot
** As regard the constraints of the signature:
** 1) we state that if an entrance is done by a customer and it is
**    valid, then it is the active entrance of that user: in this way,
**    we have that if  an entrance does not appear as the active entrance
**    of a user, then its state must not be valid or its owner is a nonUser
** 2) we state also that if the owner of the entrance is a nonUser, then
**    there cannot be a user that has that entrance as an active entrance
*/
abstract sig Entrance {
    code: one QR,
    customer: one Customer,
    supermarket: one Supermarket,
    date: one Date,
    arrivalTime: lone Time,
    timeSpent: lone Time,
    state: one EntranceState,
} {
    ((customer in User and state = VALID) implies customer.activeEntrance = this)
    customer = nonUser implies (no u: User | u.activeEntrance = this)
}
sig Ticket extends Entrance {
    vehicle: one Vehicle
```

```
}
sig Visit extends Entrance {
    categories: set Category,
    exitTime: one Time
} {
    customer in Data.users
    arrivalTime != none
    timeIsPrecedent[arrivalTime, exitTime]
    #categories > 0
    all c: Category | c in categories implies c in supermarket.supermarketCategorie
s
}

/*
** Timetable of the supermarket, in which for each day it is stated opening
** and closure time, providing consistency with predicate timeIsPrecedent
*/
sig Timetable {
    date: one Date,
    openingTime: one Time,
    closureTime: one Time
} {
    timeIsPrecedent[openingTime, closureTime]
}

/*
** Core of the application, that allows the correct lining-up of customers
** There are two constraints:
** 1) registered customers must line-up only in their own line
** 2) unregistered customers must line-up in their own line (fallback option)
*/
sig Lines {
    supermarket: one Supermarket,
    people: set Entrance,
    userLineUp: set Ticket,
    nonUserLineUp: set Ticket,
    visits: set Visit
} {
    supermarket.lines = this
    all e: Entrance | e in people implies e.date = supermarket.timetable.first.date
    all e: Entrance | e in userLineUp implies (e.customer in User and e.date = supe
rmarket.timetable.first.date)
    all e: Entrance | e in nonUserLineUp implies (e.customer = nonUser and e.date =
 supermarket.timetable.first.date)
}

/*
** Time slot of visits
*/
```

```
sig TimeSlot{
    supermarket: one Supermarket,
    arrivalTime: one Time,
    exitTime: one Time
} {
    supermarket in Data.supermarkets
    timeIsPrecedent[arrivalTime, exitTime]
}

/*
** Signature that modelizes the request to line-up in a specific supermarket
** enabling the availability check of space and the computing of other
** solutions
*/
sig AvailabilityRequestTicket {
    user: one User,
    supermarket: one Supermarket
} {
    user.activeEntrance = none
}

/*
** Signature equivalent as the precedent, but for Visits
*/
sig AvailabilityRequestVisit {
    user: one User,
    timeSlot: one TimeSlot
} {
    user.activeEntrance = none
}

/*
** This is the response for the previous request (Ticket)
** If the requested supermarket is in the list, then the list must
** contain only that supermarket
*/
sig SuggestTicket {
    request: one AvailabilityRequestTicket,
    ticketSuggestions: set Supermarket
} {
    #ticketSuggestions > 0
    request.supermarket in ticketSuggestions implies #ticketSuggestions = 1
    request.supermarket not in ticketSuggestions implies (
        all s: Supermarket | s in ticketSuggestions iff (
            #request.supermarket.lines.userLineUp > #s.lines.userLineUp and
            s.capacity > #s.lines.userLineUp
        )
    )
}
```

```
/*
** This is the response for the previous request (Visit)
** As before, if the requested time slot is in the list, the the list
** must contain only that slot
** Otherwise the list will contain all those time slots that satisfy only
** one of the following requirements:
** 1) same supermarket, but different time slot
** 2) same time slot, but different supermarket
*/
sig SuggestVisit {
    request: one AvailabilityRequestVisit,
    visitSuggestions: set TimeSlot
} {
    #visitSuggestions > 0
    request.timeSlot in visitSuggestions implies #visitSuggestions = 1
    request.timeSlot not in visitSuggestions implies (
        all ts: TimeSlot | ts in visitSuggestions implies ((
            ts.supermarket = request.timeSlot.supermarket and (
            (timeIsPrecedent[ts.exitTime, request.timeSlot.arrivalTime] and
            timeIsPrecedent[request.timeSlot.supermarket.timetable.first.openingTim
e, ts.arrivalTime])
            or (timeIsPrecedent[request.timeSlot.exitTime, ts.arrivalTime] and
            timeIsPrecedent[ts.exitTime, request.timeSlot.supermarket.timetable.fir
st.closureTime]))
            ) or
            (ts.supermarket != request.timeSlot.supermarket and
            ts.arrivalTime = request.timeSlot.arrivalTime and
            ts.exitTime = request.timeSlot.exitTime)
        )
    )
}

/*
** Signature of notification, that can arrive only to registered users without
** an active entrance. The purpose of this signature is to show that if such a
** notification exists, the corresponding user has no active entrance and its
** time slot correspond to at least one of that user's used visit
*/
sig Notification {
    user: one User,
    notification: one TimeSlot
} {
    user in Data.users
    user.activeEntrance = none
    some v: Visit | (v in Data.entrances and v.customer = user) implies (
        notification.supermarket = v.supermarket and notification.arrivalTime = v.a
rrivalTime
        and notification.exitTime = v.exitTime
    )
}


/* ** FACTS ** */

/*
** data must contain only data of registered customers (users), data of
** supermarkets and data of non expired entrances of registered customers
** (users)
*/
fact dataConsistency {
    all u: User | u in Data.users
    all s: Supermarket | s in Data.supermarkets
    all e: Entrance | e in Data.entrances iff (e.state != EXPIRED and e.customer in
 User)
}

/*
** every entrance must have one unique QR to distinguish it
** from other entrances
*/
fact uniqueQR {
    all e1, e2: Entrance | e1.code = e2.code iff e1 = e2
    all qr: QR | one e: Entrance | e.code = qr
}

/*
** every supermarket is unique, i.e. there cannot be more
** supermarkets with same references
*/
fact uniqueSupermarkets {
    all s1, s2: Supermarket | s1.position = s2.position iff s1 = s2
    all s1, s2: Supermarket | s1.lines = s2.lines iff s1 = s2
    all s1, s2: Supermarket | s1.ticketMachine = s2.ticketMachine iff s1 = s2
    all s1, s2: Supermarket | s1.manager = s2.manager iff s1 = s2
    all s1, s2: Supermarket | s1.statistics = s2.statistics iff s1 = s2
    #Supermarket = #SupermarketPosition
}

/*
** users cannot be in the same position, otherwise they would be the same
** person (we exclude multiple smartphones for one person)
*/
fact noUsersSamePosition {
    all u1, u2: User | u1.position = u2.position iff u1 = u2
    #User > #GPSPosition or #User = #GPSPosition
}
```

```
/*
** customers are not allowed after closure
** (only visits scheduled for future days are allowed)
*/
fact noCustomersInsideAfterClosure {
    all s: Supermarket | all t: Time |
    timeIsPrecedent[s.timetable.first.closureTime, t] implies (
        (#s.lines.people = 0 and #s.lines.userLineUp = 0 and #s.lines.nonUserLineUp
 = 0) and
        (all v: Visit | v in s.lines.visits implies v.date != s.timetable.first.dat
e)
    )
}

/*
** customers are not allowed before opening
** (only visits scheduled for future days are allowed)
*/
fact noCustomersInsideBeforeOpening {
    all s: Supermarket | all t: Time |
    timeIsPrecedent[t, s.timetable.first.openingTime] implies (
        (#s.lines.people = 0 and #s.lines.userLineUp = 0 and #s.lines.nonUserLineUp
 = 0)
    )
}

/*
** no entrances after closure
*/
fact noEntrancesAfterClosure {
    all s: Supermarket | no tk: Ticket | timeIsPrecedent[s.timetable.first.closureT
ime, tk.arrivalTime]
        and s.timetable.first.date = tk.date
    all s: Supermarket | no v: Visit | timeIsPrecedent[s.timetable.first.closureTim
e, v.arrivalTime]
        and s.timetable.first.date = v.date
}

/*
** no entrances before opening
*/
fact noEntrancesBeforeOpening {
    all s: Supermarket | no tk: Ticket | timeIsPrecedent[tk.arrivalTime, s.timetabl
e.first.openingTime]
        and s.timetable.first.date = tk.date
    all s: Supermarket | no v: Visit | timeIsPrecedent[v.arrivalTime, s.timetable.f
irst.openingTime]
        and s.timetable.first.date = v.date
}
```

```
/*
** This is the most important fact, because it shows that entrances can
** be correctly lined-up  (purpose of the application)
** 1) every entrance that is lined-up in every lines it must be VALID
** 2) an entrance to be valid must be correctly lined-up in one and only
**    one sublines of the corresponding supermarket
** 3) every supermarket lines cannot contain entrances of other
**    supermarkets
*/
fact correctlyLined {
    all e: Entrance | all l: Lines | (e in l.people or e in l.userLineUp or e in l.
nonUserLineUp
        or e in l.visits) implies e.state = VALID
    all e: Entrance | e.state = VALID iff inLines[e, e.supermarket.lines]
    all l: Lines | all e: Entrance | l != e.supermarket.lines implies (e not in l.p
eople and
        e not in l.userLineUp and e not in l.nonUserLineUp and e not in l.visits)
}

/*
** User cannot send more than one request at a time while creating
** an entrance and, consequently (iff), he/she cannot receive
** more than one response
*/
fact noContemporaneousRequests {
    all rt: AvailabilityRequestTicket | all rv: AvailabilityRequestVisit | rt.user
!= rv.user
    all rt1, rt2: AvailabilityRequestTicket | rt1.user = rt2.user iff rt1 = rt2
    all rv1, rv2: AvailabilityRequestVisit | rv1.user = rv2.user iff rv1 = rv2
}

/*
** Consistency of answers from server: one for each request
*/
fact oneAnswerForEachRequest {
    all st1, st2: SuggestTicket | st1.request = st2.request iff st1 = st2
    all sv1, sv2: SuggestVisit | sv1.request = sv2.request iff sv1 = sv2
}

/*
** As before, but for notifications
*/
fact noContemporaneousNotification {
    all n1, n2: Notification | n1.user = n2.user iff n1 = n2
}

/*
** As said before, this fact states that a notification can
** happen onlyif the customer has already booked and
** exploited at least one visit
*/
fact noNotificationWithoutData {
    all u: User | (no v: Visit | v in Data.entrances and v.customer = u) implies (
        no n: Notification | n.user = u
    )
}

/*
** Timetables can exists only if they are linked to a supermarket
*/
fact NoLonelyTimetables {
    no tt: Timetable | (all s: Supermarket | #(s.timetable.indsOf[tt]) = 0)
}


/* ** PREDICATES ** */

/*
** predicate that allows to check if an entrance e correctly
** belongs to lines l, that is, e belongs to one and only one
** of the sublines of l
*/
pred inLines[e: Entrance, l: Lines] {
    (e in l.people and e not in l.userLineUp and e not in l.nonUserLineUp and e not
 in l.visits) or
    (e not in l.people and e in l.userLineUp and e not in l.nonUserLineUp and e not
 in l.visits
        and e in Ticket) or
    (e not in l.people and e not in l.userLineUp and e in l.nonUserLineUp and e not
 in l.visits
        and e in Ticket) or
    (e not in l.people and e not in l.userLineUp and e not in l.nonUserLineUp and e
 in l.visits
        and e in Visit)
}
run inLines

/*
** predicate that allows to check if a time t1 precedes
** a time t2
*/
pred timeIsPrecedent [t1: Time, t2: Time] {
    (t2.hour > t1.hour) or (t2.hour = t1.hour and t2.minute > t1.minute)
}
run timeIsPrecedent

/*
```

```
** This predicate aims to show the correct separation of lines
** in a supermarket
*/
pred LineDistinction {
    #Ticket = 2
    #Visit = 2
    #User = 3
    #Supermarket = 1
    one t: Ticket | t.customer in User
    #Supermarket.lines.people = 1
    #Supermarket.lines.userLineUp = 1
    #Supermarket.lines.nonUserLineUp = 1
    #Supermarket.lines.visits = 1
}
run LineDistinction for 7

/*
** This predicate aims to show the correct behaviour of
** a notification
*/
pred timeToShop (n: Notification, u: User, v: Visit){
    n.user = u
    #Notification = 1
    #User = 1
}
run timeToShop for 7

/*
** This predicate aims to show the correct answer to
** an availability request from a user during the creation
** of a ticket (indeed that user has no active entrance).
** We put 3 supermarkets, in which the request is made on
** the most crowded, instead the two other are way less crowded
** and they can appear in the recommendations
*/
pred suggestionToChangeSupermarket (u: User, s, s', s'': Supermarket, st: SuggestTi
cket,
    rt: AvailabilityRequestTicket, d: Date) {
    #Supermarket = 3
    #s.lines.people = 1
    s.capacity = 1
    #s.lines.userLineUp = 2
    #s'.lines.people = 0
    s'.capacity = 1
    #s'.lines.userLineUp = 0
    #s'.lines.nonUserLineUp = 0
    #s'.lines.visits = 0
    #s''.lines.people = 0
    s''.capacity = 1
    #s''.lines.userLineUp = 0
    #s''.lines.nonUserLineUp = 0
    #s''.lines.visits = 0
    s' != s''
    rt.user = u
    rt.supermarket = s
    st.request = rt
    s not in st.ticketSuggestions
    all super: Supermarket | super.timetable.first.date = d
}
run suggestionToChangeSupermarket for 7

/*
** This predicate aims to show the correct answer to
** an availability request from a user during the creation
** of a ticket (indeed that user has no active entrance).
** We put 3 supermarkets, in which the request is made on
** the most crowded, instead only one of the two others is way
** less crowded and it can appear in the recommendations
*/
pred suggestionToChangeSupermarket_OneSuggestion (u: User, s, s', s'': Supermarket,
 st: SuggestTicket,
    rt: AvailabilityRequestTicket, d: Date) {
    #Supermarket = 3
    #s.lines.people = 1
    s.capacity = 1
    #s.lines.userLineUp = 2
    #s'.lines.people = 0
    s'.capacity = 1
    #s'.lines.userLineUp = 0
    #s'.lines.nonUserLineUp = 0
    #s'.lines.visits = 0
    #s''.lines.people = 1
    s''.capacity = 1
    #s''.lines.userLineUp = 2
    s != s''
    rt.user = u
    rt.supermarket = s
    st.request = rt
    s not in st.ticketSuggestions
    all super: Supermarket | super.timetable.first.date = d
}
run suggestionToChangeSupermarket_OneSuggestion for 10 but 5 int, 10 seq

/*
** This predicate aims to show the correct answer to
** an availability request from a user during the
** creation of a visit (indeed that user has no active entrance).
** We put 2 supermarkets, in which the request is made on
** the most crowded, instead the other one is way less crowded
** and it can appear in the recommendations
*/
pred suggestionToChangeSupermarket_Visit (u: User, s, s': Supermarket, ts: TimeSlot
, sv: SuggestVisit,
    rv: AvailabilityRequestVisit, d: Date) {
    #Supermarket = 2
    #s.lines.people = 1
    s.capacity = 1
    #s.lines.userLineUp = 2
    #s'.lines.people = 0
    s'.capacity = 1
    #s'.lines.userLineUp = 0
    #s'.lines.nonUserLineUp = 0
    #s'.lines.visits = 0
    rv.user = u
    rv.timeSlot = ts
    ts.supermarket = s
    sv.request = rv
    ts not in sv.visitSuggestions
    all super: Supermarket | super.timetable.first.date = d
}
run suggestionToChangeSupermarket_Visit for 7

/*
** This predicate aims to show the correct answer to
** an availability request from a user during the creation
** of a visit (indeed that user has no active entrance).
** We put 1 supermarkets, so that the selected slot is not
** available and the suggested slot is another one of the
** same supermarket, but different timing
*/
pred suggestionToChangeSlot_Visit (u: User, s: Supermarket, ts: TimeSlot, sv: Sugge
stVisit,
    rv: AvailabilityRequestVisit, d: Date) {
    #User = 1
    #Ticket = 0
    #Visit = 0
    #s.timetable = 1
    #Supermarket = 1
    #s.lines.people = 0
    s.capacity = 1
    rv.user = u
    rv.timeSlot = ts
    ts.supermarket = s
    sv.request = rv
    ts not in sv.visitSuggestions
    all super: Supermarket | super.timetable.first.date = d
}
```

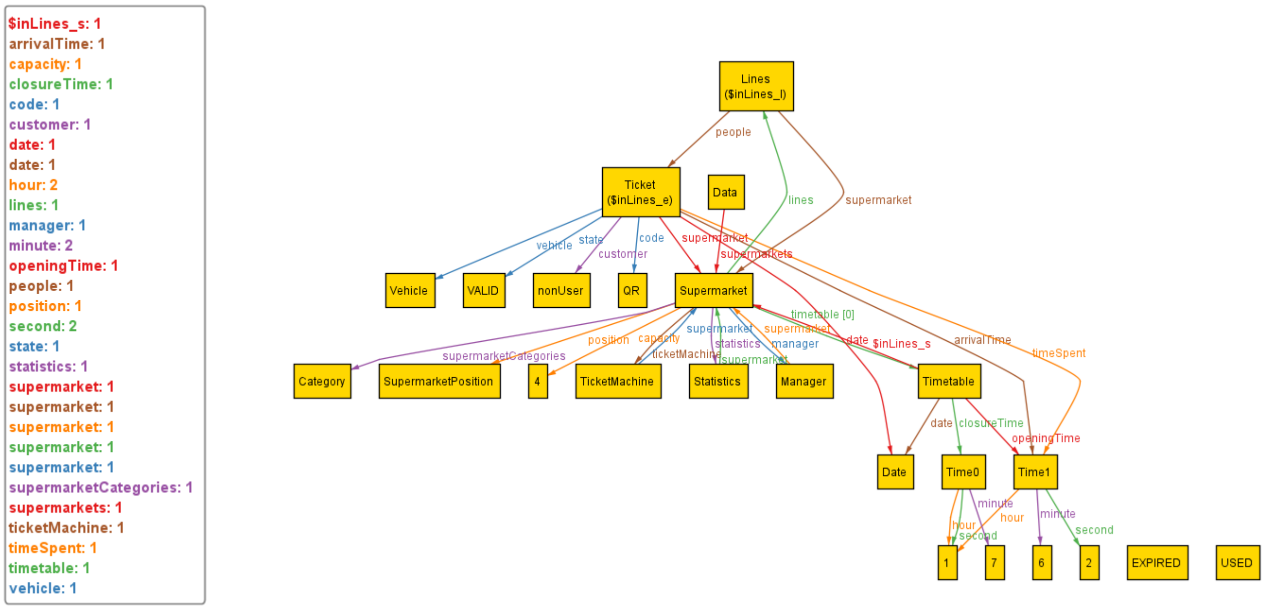
```
run suggestionToChangeSlot_Visit for 7

/*
** This pred wants to show the correct behaviour of
** the answer to a request for an available supermarket
** during the process of creation of a ticket
*/
pred requestAvailabilityTicket_Available(u: User, s: Supermarket, st: SuggestTicket
,
    rt: AvailabilityRequestTicket, d: Date) {
    #User = 1
    #Supermarket = 1
    #s.lines.people = 0
    #s.lines.userLineUp = 0
    s.capacity = 1
    rt.user = u
    rt.supermarket = s
    st.request = rt
    s in st.ticketSuggestions
    #Ticket = 0
    #s.timetable = 1
    all super: Supermarket | super.timetable.first.date = d
}
run requestAvailabilityTicket_Available for 7

/*
** This pred wants to show the correct behaviour of
** the answer to a request for an available supermarket
** during the process of book a visit
*/
pred requestAvailabilityVisit_Available(u: User, s: Supermarket, ts: TimeSlot, sv:
SuggestVisit,
    rv: AvailabilityRequestVisit, d: Date) {
    #User = 1
    #Supermarket = 1
    #s.lines.people = 0
    #s.lines.userLineUp = 0
    s.capacity = 1
    ts.supermarket = s
    rv.user = u
    rv.timeSlot = ts
    sv.request = rv
    ts in sv.visitSuggestions
    #Ticket = 0
    #s.timetable = 1
    all super: Supermarket | super.timetable.first.date = d
}
run requestAvailabilityVisit_Available for 7
```

```
/*
** predicate that verifies the correctness of
** a very complex situation
*/
pred ExtremeModel (d: Date) {
    #User > 3
    #Supermarket > 1
    #Entrance > 6
    #Visit > 1
    #SuggestTicket > 0
    #SuggestVisit > 0
    #Notification = 1
    all s: Supermarket | s.capacity = 2
    all s: Supermarket | #s.lines.people = s.capacity
    some s: Supermarket | #s.lines.userLineUp > 0
    some s: Supermarket | #s.lines.nonUserLineUp > 0
    some s: Supermarket | #s.lines.visits > 0
    some e: Entrance | e.state = EXPIRED
    some t: Ticket | t.customer in User
    some v: Visit | v.state = VALID
    all super: Supermarket | super.timetable.first.date = d
}
run ExtremeModel for 10 but 5 int, 10 seq


/* ** ASSERTIONS ** */

/*
** This assertion wants to prove that
** a notification cannot exist without
** some data
*/
assert allNotificationsHasData {
    all n: Notification | some v: Visit | v.customer = n.user
}
check allNotificationsHasData

/*
** This assertion wants to prove that
** a supermarket does not contains more customers
** that its capacity
*/
assert noMoreCustomersThanCapacity {
    all s: Supermarket | s.capacity > #s.lines.people or s.capacity = #s.lines.peop
le
}
check noMoreCustomersThanCapacity

/*
```

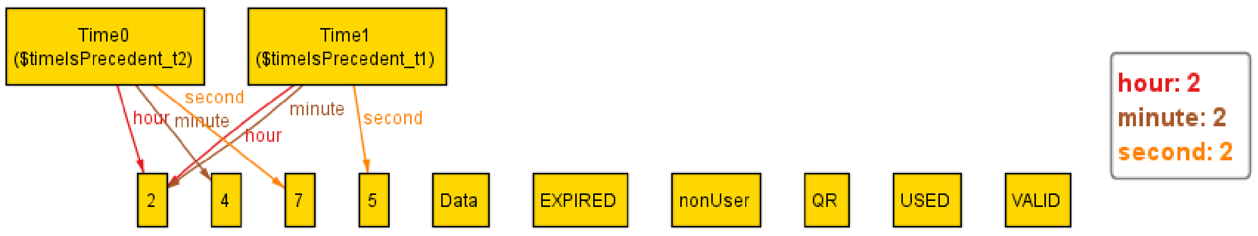
```
** This assertion wants to prove that every
** supermarket has all the characteristics
** to be considered a supermarket
*/
assert everySupermarketHasAllFunctionalities {
    all m: Manager | one s: Supermarket | s.manager = m
    all tm: TicketMachine | one s: Supermarket | s.ticketMachine = tm
    all st: Statistics | one s: Supermarket | s.statistics = st
    all l: Lines | one s: Supermarket | s.lines = l
    all sp: SupermarketPosition | one s: Supermarket | s.position = sp
}
check everySupermarketHasAllFunctionalities

/*
** This assertsion wants to prove that
** our model does not allow "ghost users",
** that are GpsPositions without an owner
** (we cannot register GpsPositions that
** does not belong to anyone)
*/
assert noGhostUsers {
    all gps: GPSPosition | one u: User | u.position = gps
}
check noGhostUsers
```

4.2 Metamodel

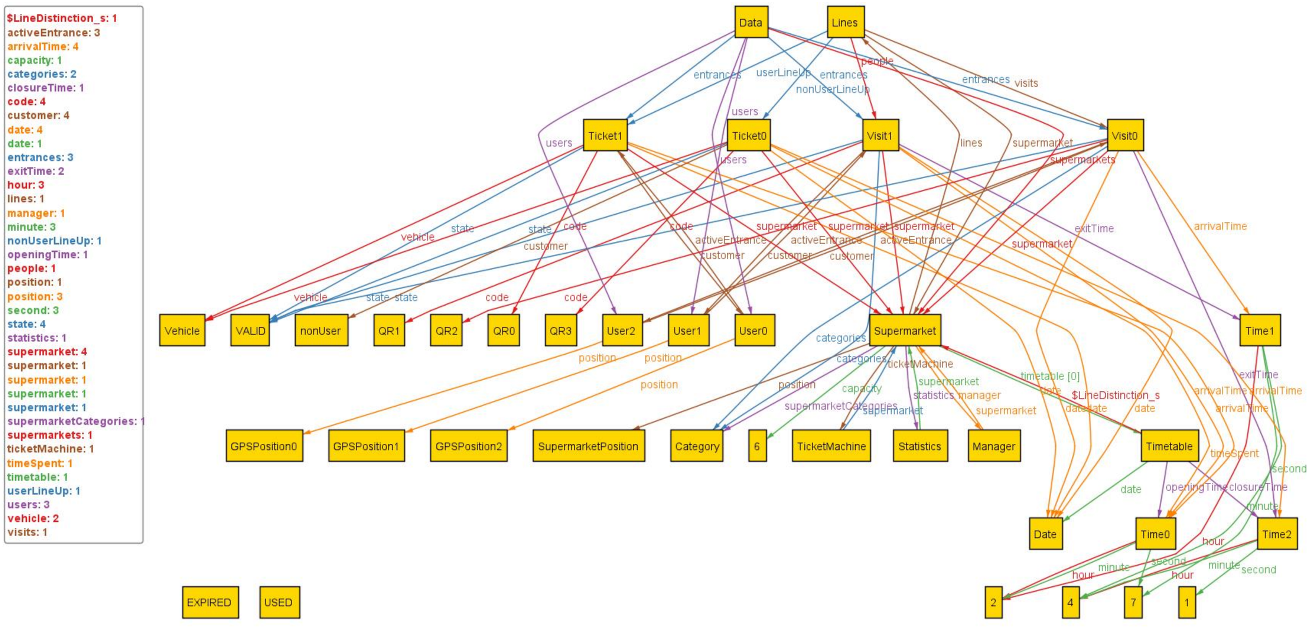
1. Predicate inLines



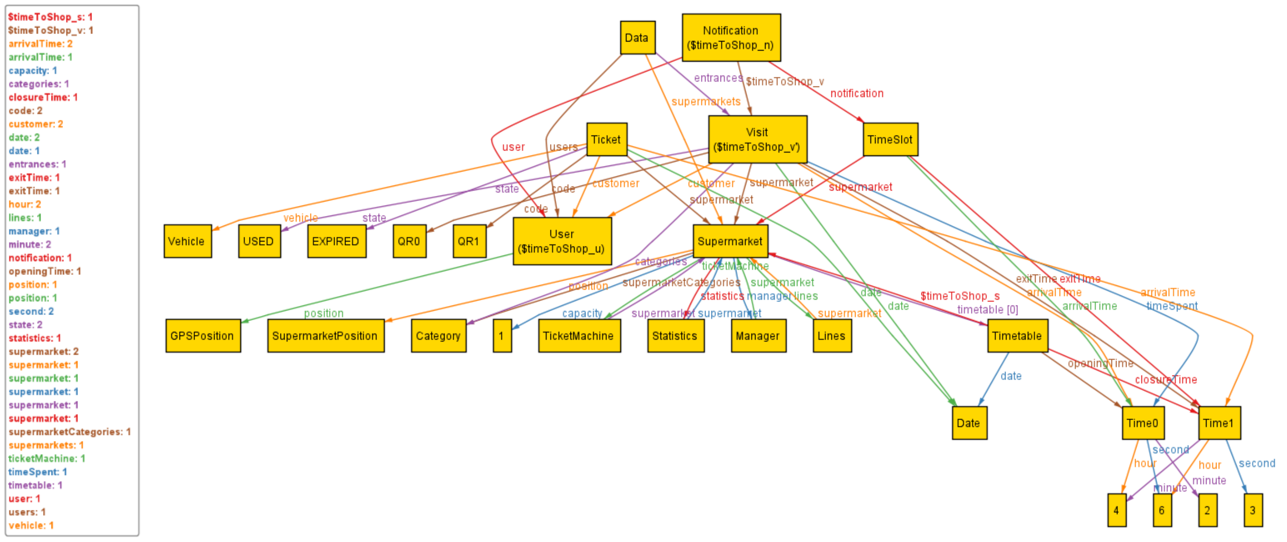
2. Predicate TimeIsPrecedent



3. Predicate LineDistinction

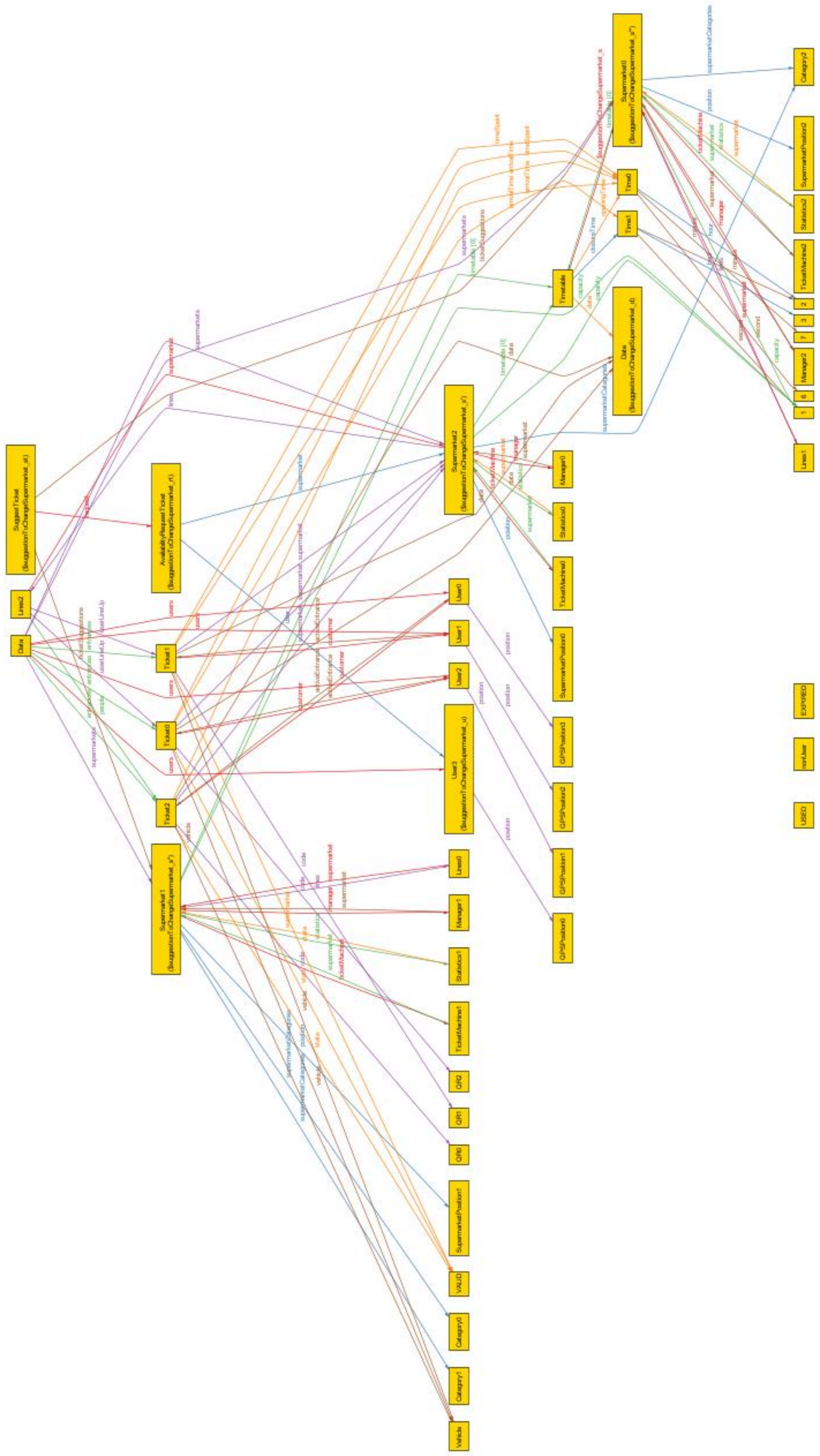


4. Predicate TimeToShop



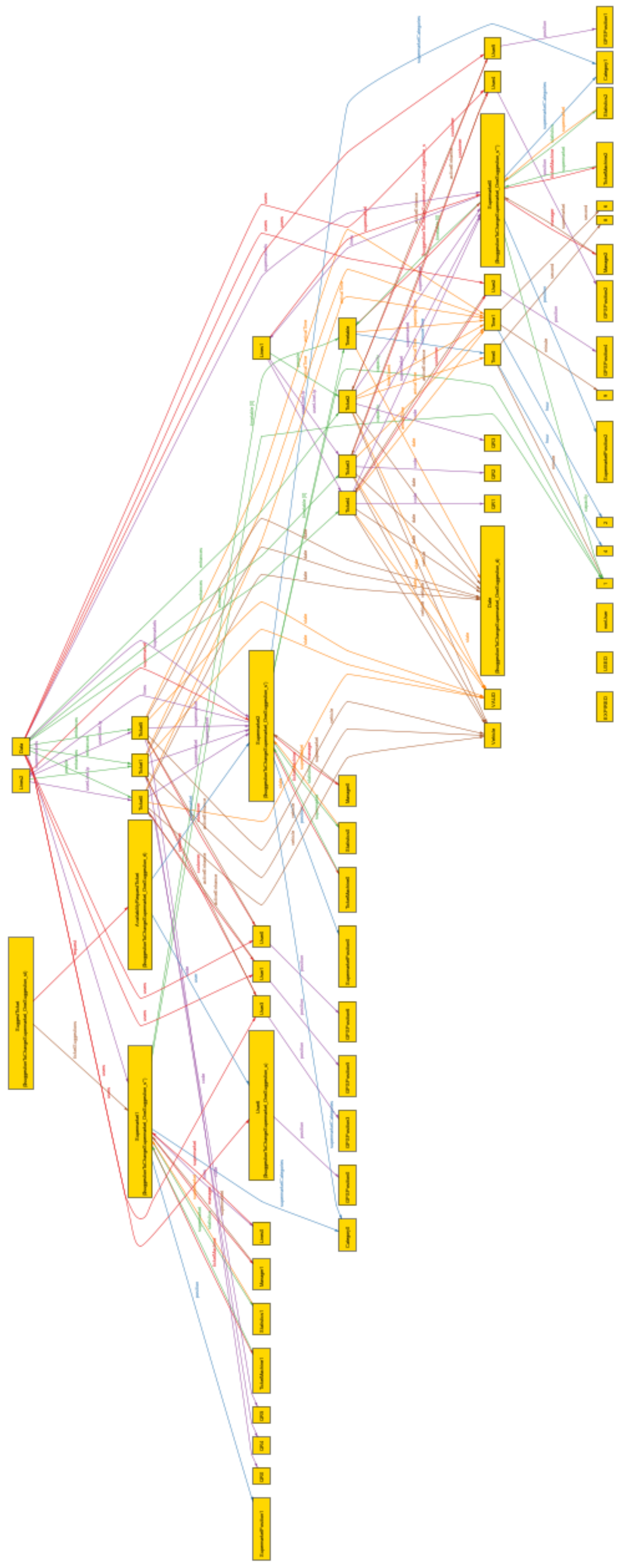
5. Predicate suggestionToChangeSupermarket

$suggestionToChangeSupermarket_s: 1
activeEntrance: 3
arrivalTime: 3
capacity: 3
closureTime: 1
code: 3
customer: 3
date: 3
date: 1
entrances: 3
hour: 2
lines: 3
manager: 3
minute: 2
openingTime: 1
people: 1
position: 3
position: 4
request: 1
second: 2
state: 3
statistics: 3
supermarket: 1
supermarket: 3
supermarket: 3
supermarket: 3
supermarket: 3
supermarket: 3
supermarketCategories: 4
supermarkets: 3
ticketMachine: 3
ticketSuggestions: 2
timeSpent: 2
timetable: 3
user: 1
userLineUp: 2
users: 4
vehicle: 3

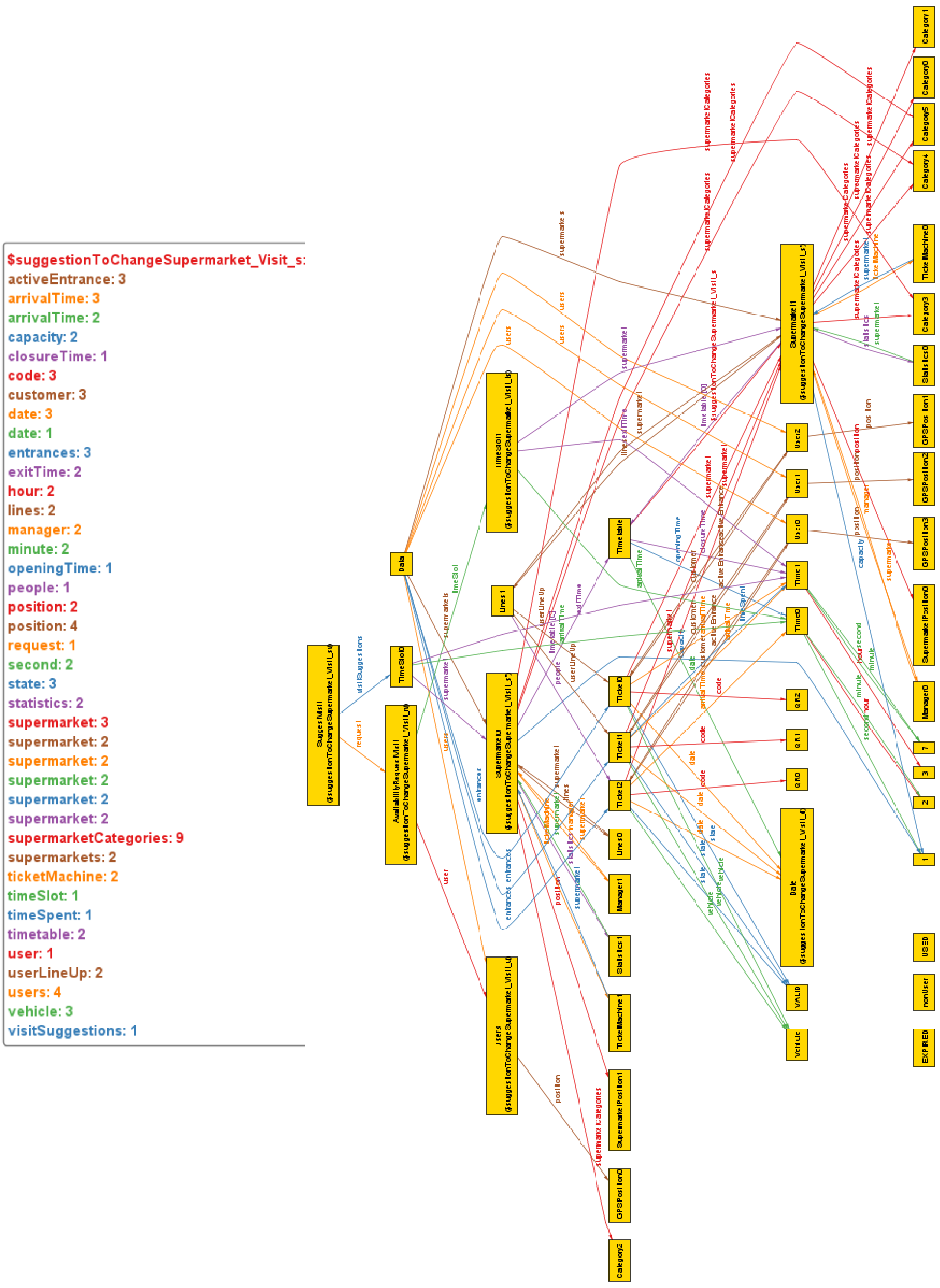


6. Predicate suggestionToChangeSupermarket_OneSuggestion

$suggestionToChangeSupermarket_OneSuggestion_s: 1
activeEntrance: 6
arrivalTime: 6
capacity: 3
closureTime: 1
code: 6
customer: 6
date: 6
date: 1
entrances: 6
hour: 2
lines: 3
manager: 3
minute: 2
openingTime: 1
people: 2
position: 3
position: 7
request: 1
second: 2
state: 6
statistics: 3
supermarket: 1
supermarket: 6
supermarket: 3
supermarket: 3
supermarket: 3
supermarket: 3
supermarketCategories: 4
supermarkets: 3
ticketMachine: 3
ticketSuggestions: 1
timeSpent: 1
timetable: 3
user: 1
userLineUp: 4
users: 7
vehicle: 6

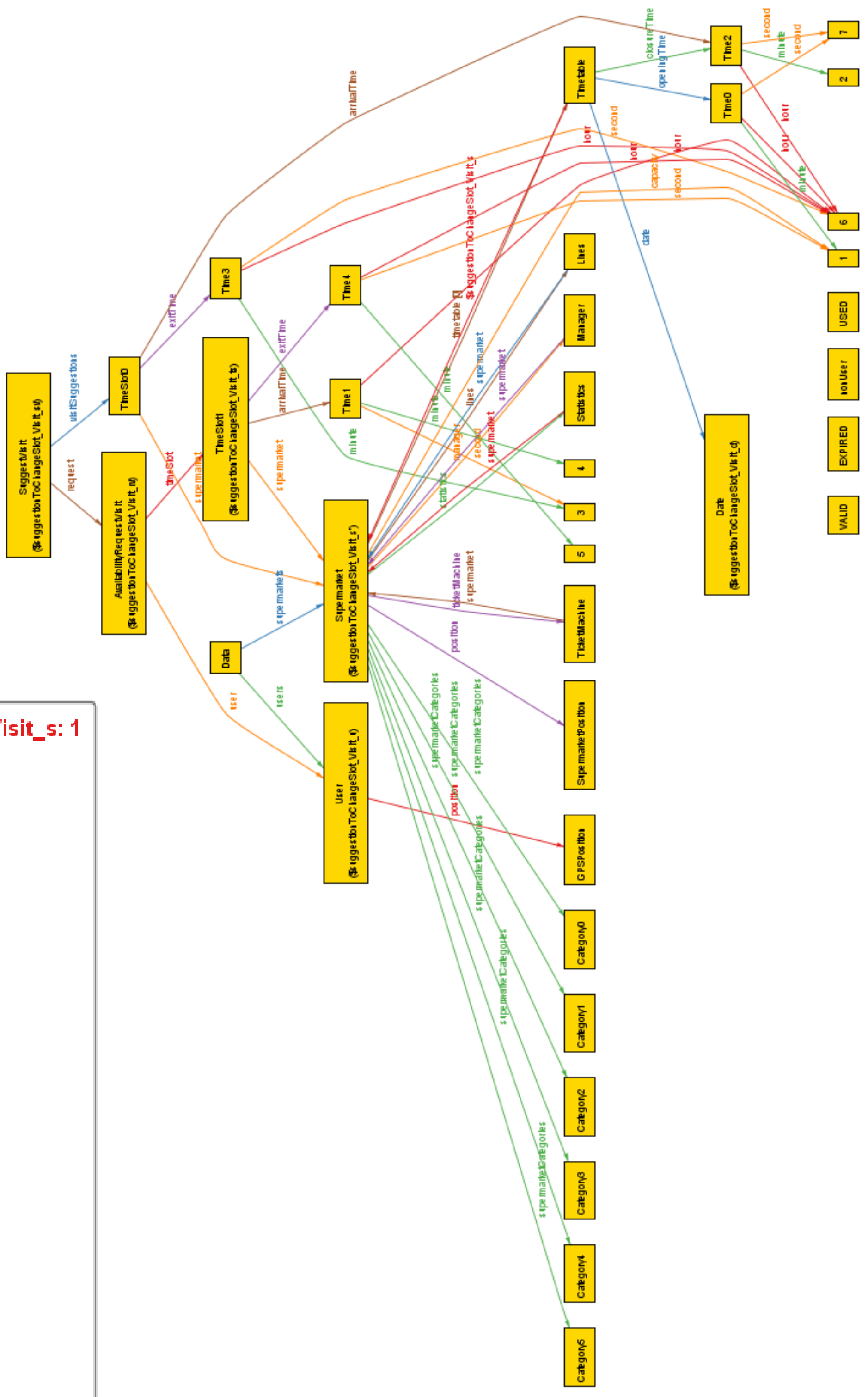


7. Predicate suggestionToChangeSupermarket_Visit



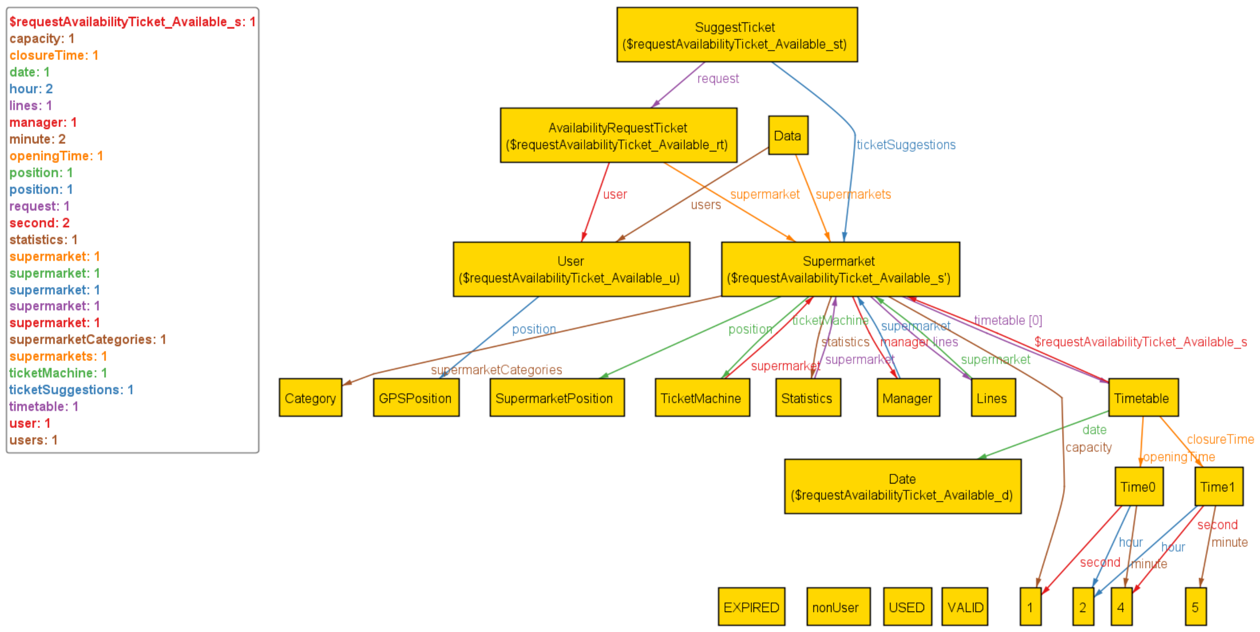
$suggestionToChangeSupermarket_Visit_s:
activeEntrance: 3
arrivalTime: 3
arrivalTime: 2
capacity: 2
closureTime: 1
code: 3
customer: 3
date: 3
date: 1
entrances: 3
exitTime: 2
hour: 2
lines: 2
manager: 2
minute: 2
openingTime: 1
people: 1
position: 2
position: 4
request: 1
second: 2
state: 3
statistics: 2
supermarket: 3
supermarket: 2
supermarket: 2
supermarket: 2
supermarket: 2
supermarket: 2
supermarketCategories: 9
supermarkets: 2
ticketMachine: 2
timeSlot: 1
timeSpent: 1
timetable: 2
user: 1
userLineUp: 2
users: 4
vehicle: 3
visitSuggestions: 1

8. Predicate suggestionToChangeSlot_Visit

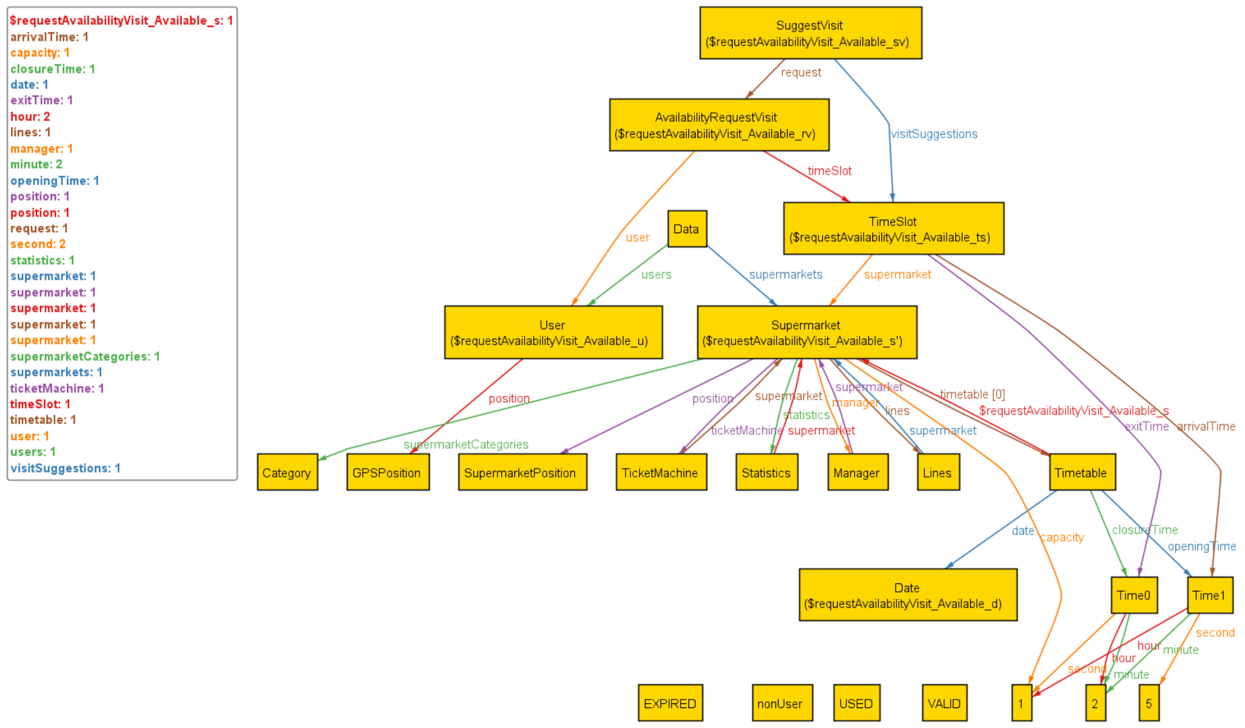

SuggestedVisit ($suggestionToChangeSlot_Visit_sv)
AvailabilityRequestVisit ($suggestionToChangeSlot_Visit_rv)
TimeSlot0
TimeSlot1 ($suggestionToChangeSlot_Visit_ts)
Time3
Time4
Time1
Supermarket ($suggestionToChangeSlot_Visit_s')
Data
User ($suggestionToChangeSlot_Visit_t)
Timetable
Time2
Time0
Date ($suggestionToChangeSlot_Visit_d)
Lines
Manager
Statistics
TicketMachine
SupermarketPosition
GPSPosition
Category0
Category1
Category2
Category3
Category4
Category5
7
2
6
1
4
3
5
USED
NonUser
EXPIRED
VALID

$suggestionToChangeSlot_Visit_s: 1
arrivalTime: 2
capacity: 1
closureTime: 1
date: 1
exitTime: 2
hour: 7
lines: 1
manager: 1
minute: 7
openingTime: 1
position: 1
position: 1
request: 1
second: 7
statistics: 1
supermarket: 1
supermarket: 1
supermarket: 1
supermarket: 1
supermarket: 2
supermarketCategories: 1
supermarkets: 1
ticketMachine: 1
timeSlot: 1
timetable: 1
user: 1
users: 1
visitSuggestions: 1

9. Predicate requestAvailabilityTicket_Available

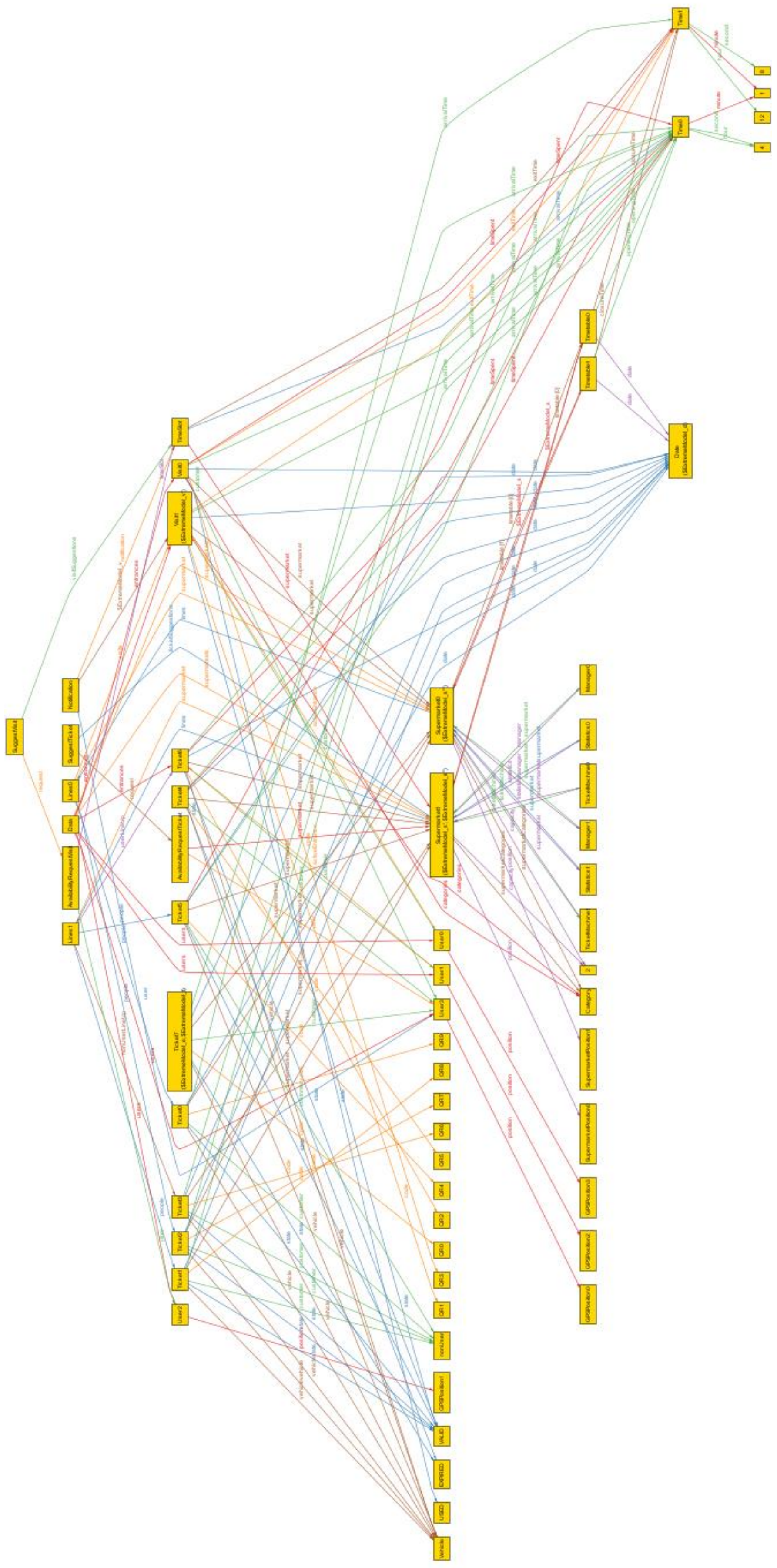


10. Predicate requestAvailabilityVisit_Available

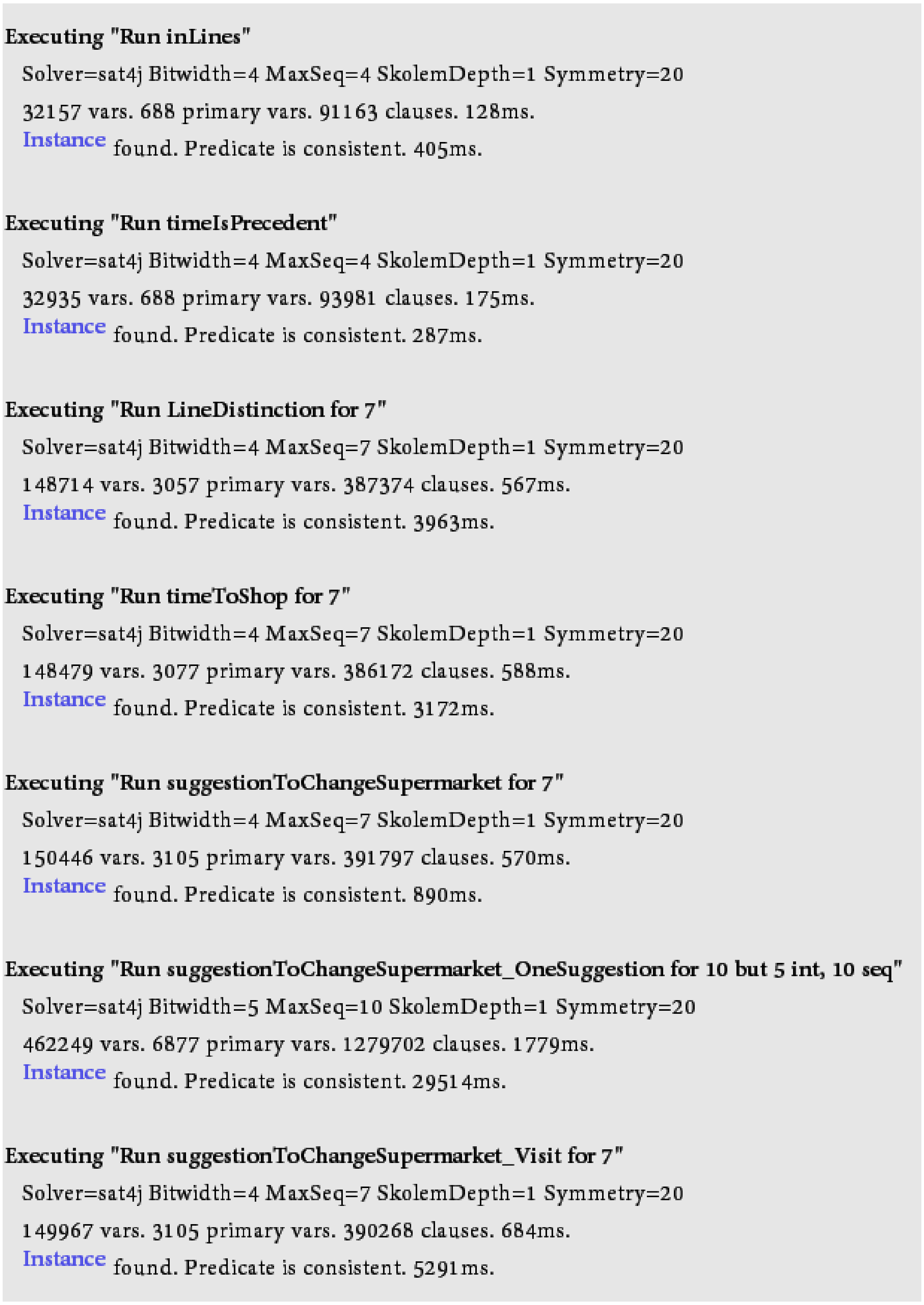


11. Predicate ExtremeModel

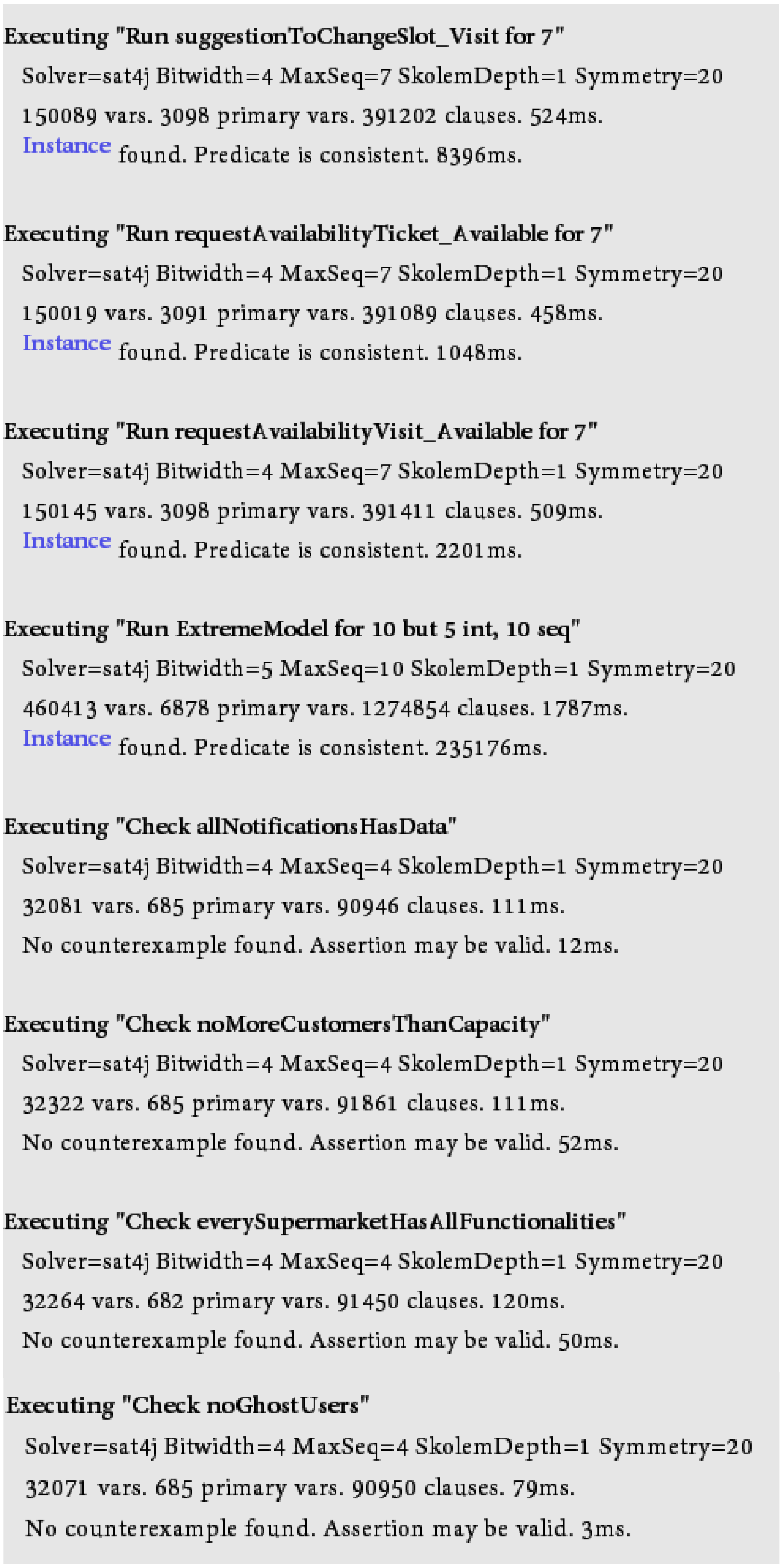
$ExtremeModel_s: 2
$ExtremeModel_v: 1
activeEntrance: 2
arrivalTime: 10
arrivalTime: 1
capacity: 2
categories: 2
closureTime: 2
code: 10
customer: 10
date: 10
date: 2
entrances: 3
exitTime: 1
exitTime: 2
hour: 2
lines: 2
manager: 2
minute: 2
nonUserLineUp: 1
notification: 1
openingTime: 2
people: 4
position: 2
position: 4
request: 1
request: 1
second: 2
state: 10
statistics: 2
supermarket: 1
supermarket: 10
supermarket: 2
supermarket: 2
supermarket: 2
supermarket: 2
supermarket: 1
supermarketCategories: 2
supermarkets: 2
ticketMachine: 2
ticketSuggestions: 1
timeSlot: 1
timeSpent: 4
timetable: 3
user: 1
user: 1
user: 1
userLineUp: 1
users: 4
vehicle: 8
visits: 1
visitSuggestions: 1



4.3 Result of Predicates and Assertions



```
Executing "Run inLines"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   32157 vars. 688 primary vars. 91163 clauses. 128ms.
   Instance found. Predicate is consistent. 405ms.

Executing "Run timeIsPrecedent"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   32935 vars. 688 primary vars. 93981 clauses. 175ms.
   Instance found. Predicate is consistent. 287ms.

Executing "Run LineDistinction for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   148714 vars. 3057 primary vars. 387374 clauses. 567ms.
   Instance found. Predicate is consistent. 3963ms.

Executing "Run timeToShop for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   148479 vars. 3077 primary vars. 386172 clauses. 588ms.
   Instance found. Predicate is consistent. 3172ms.

Executing "Run suggestionToChangeSupermarket for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   150446 vars. 3105 primary vars. 391797 clauses. 570ms.
   Instance found. Predicate is consistent. 890ms.

Executing "Run suggestionToChangeSupermarket_OneSuggestion for 10 but 5 int, 10 seq"
   Solver=sat4j Bitwidth=5 MaxSeq=10 SkolemDepth=1 Symmetry=20
   462249 vars. 6877 primary vars. 1279702 clauses. 1779ms.
   Instance found. Predicate is consistent. 29514ms.

Executing "Run suggestionToChangeSupermarket_Visit for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   149967 vars. 3105 primary vars. 390268 clauses. 684ms.
   Instance found. Predicate is consistent. 5291ms.
```



```
Executing "Run suggestionToChangeSlot_Visit for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   150089 vars. 3098 primary vars. 391202 clauses. 524ms.
   Instance found. Predicate is consistent. 8396ms.

Executing "Run requestAvailabilityTicket_Available for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   150019 vars. 3091 primary vars. 391089 clauses. 458ms.
   Instance found. Predicate is consistent. 1048ms.

Executing "Run requestAvailabilityVisit_Available for 7"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   150145 vars. 3098 primary vars. 391411 clauses. 509ms.
   Instance found. Predicate is consistent. 2201ms.

Executing "Run ExtremeModel for 10 but 5 int, 10 seq"
   Solver=sat4j Bitwidth=5 MaxSeq=10 SkolemDepth=1 Symmetry=20
   460413 vars. 6878 primary vars. 1274854 clauses. 1787ms.
   Instance found. Predicate is consistent. 235176ms.

Executing "Check allNotificationsHasData"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   32081 vars. 685 primary vars. 90946 clauses. 111ms.
   No counterexample found. Assertion may be valid. 12ms.

Executing "Check noMoreCustomersThanCapacity"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   32322 vars. 685 primary vars. 91861 clauses. 111ms.
   No counterexample found. Assertion may be valid. 52ms.

Executing "Check everySupermarketHasAllFunctionalities"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   32264 vars. 682 primary vars. 91450 clauses. 120ms.
   No counterexample found. Assertion may be valid. 50ms.

Executing "Check noGhostUsers"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   32071 vars. 685 primary vars. 90950 clauses. 79ms.
   No counterexample found. Assertion may be valid. 3ms.
```

5. Effort Spent

Andrea Manglaviti

Topic	Hours
Discussion on first part	2h
Purpose & Scope	1h
World & Shared Phenomena & Goals	1h
External Interfaces Requirements	1h
Performance Requirements & Design Constraints	0,5h
UML & State Charts	4h
Discussion on second part	3h
Writing Requirements	4h
Use Cases	2h
Discussion on third part	3h
Mapping on Mapping	3h
Alloy code	1h
Document Composition	3h
Domain Assumption	2,5h
Revision of Requirements, Domain Goals & Mapping	2h
Revision of UML & State charts	2h

Davide Marinaro

Topic	Hours
Discussion on first part	2h
Purpose & Scope	1h
World & Shared Phenomena & Goals	1h
External Interfaces Requirements	3h
Performance Requirements & Design Constraints	0,5h
UML & State Charts	4h
Discussion on second part	3h

Writing Requirements	4h
Use Cases	2h
Discussion on third part	3h
Discussion on Mapping	3h
Alloy code	15h
Document Composition	3h
Domain Assumption	2,5h
Revision of Requirements, Domain Goals & Mapping	2h
Software System Attributes	2h
Revision of UML & State charts	2h

Luca Marinello

Topic	Hours
Discussion on first part	2h
Purpose & Scope	1h
World & Shared Phenomena & Goals	1h
External Interfaces Requirements	3h
Performance Requirements & Design Constraints	0,5h
Discussion on second part	3h
Writing Requirements	4h
Use Cases	2h
Discussion on third part	3h
Discussion on Mapping	3h
Alloy code	8h
Document Composition	1h
Domain Assumption	2,5h
Sequence Diagram	4h
Revision of Requirements, Domain Goals & Mapping	2h
Revision of UML & State charts	2h

6. References

- All the diagrams have been made with draw.io: https://app.diagrams.net/
- Alloy code developed with VS Code (Alloy extension): Alloy Extension
- Alloy code executed with Alloy Analyzer: https://alloytools.org/
- Alloy guide: https://alloytools.org/download/alloy-language-reference.pdf