POLITECNICO
MILANO 1863

Requirements Analysis and Specification Document

CLup – Customer Line Up

Rei Barjami – 10588572

Simone Bianchera – 10611808

Niccolò Borrelli – 10615638

Contents

1 – Introduction

1.1 – Purpose

This document represents the Requirements Analysis and Specification Document (RASD).

The document goals are to analyse the customer real needs and to show the solution the software offers, its goals and the relationship between these and the functional and non-functional requirements, attributes and constraints. In addition to that, it will provide possible use cases and system behaviour after the release.

This document is addressed to: procurement that can use it as a contractual base, project manager to control the developing and to have a base for change (or risk) management, to management that can estimate a first baseline (of time and cost) and to developer

1.2 – Scope

1.2.1 – Brief problem description

The coronavirus emergency has put a strain on society on many levels. Many countries imposed the lockdowns, which allow people to go out from their home only for essential needs as for working or for grocery shopping. They usually enforced strict rules to the stores and to the people that can go out in specific situation, such as limiting the number of accesses to a building or keeping the distance of at least one meter between people. Often these rules guarantee a store with few people inside, but a long-crowded queue outside.

Grocery store is the main challenge. Everyone needs to do grocery shopping: food is an important part of everyone life, so it's important to minimize the possible risks that a queue outside could take and, also, it's needed an easy way to manage it, avoiding useless long waiting time, and a simple way to manage the influx of people. In addition to that, because of situation uncertainty, stores don't want to hire more people to manage this problem.

1.2.2 – Brief system description

"CLup – Customer Line Up" is a system that manages the queue automatically without the people in a physical line. It should, obviously, cover all the functionalities that a good physical queue has, such as not losing the position and knowing when it's customer's turn to enter the store. In addition to that, the system will provide the estimated waiting time: a software part will be dedicated to the notifies, so when the customer must go out to reach the store in time, depending on his/her position, and when it's his/her time to enter. Moreover, the customer could decide to book for a specific time or putting in a normal queue.

By a manager view, this app could permit an automated management of the queue, regulating the flux in the building by setting the maximum number of people that could be inside the store at the same time. It will also manage the delays without any human intervention. It will permit to monitor the entrances and the exits, in order to never exceed the set maximum number. In addition, it will permit every store to integrate the customers who don't have access to the required technology (a smartphone with GPS) through printing hand-out tickets when it will be needed.

Moreover, in case of booking or line-up impossibility, the system will suggest other stores of the same chain/congregate near the user (if they exist) or other booking time (if they exist and only in case of booking attempt)

1.2.3 – Current systems

The main app in the market is UFirst that guarantees only the possibility to retrieve a number and to know how many people precede the customer. The time in which the customer can "line-up" is statically imposed, indeed, usually, it's an hour before the closure. In addition to that there isn't any type of suggestion in case of impossibility to be inserted in the queue. Fortunately, the CLup has more features that can satisfy most of customer and store manager need, as described before.

1.3 – Goal

The goals written below are the mandatory parts that this application must have to achieve the purpose it has.

- [G1]: The system must allow customers to line up
- [G2]: The system must allow store managers to regulate the maximum influx of people in the building
- [G3]: The system must provide customers with an estimate of the waiting time
- [G4]: The system must manage customer entrances
- [G5]: The system must allow customer to book a visit in the store
- [G6]: The system must alert the customers when they have to leave to arrive in time at the store

[G1], [G5] and [G6] regard services offered to the customer: the first one permits to be put in the online queue (if accepted) without going in a physical line outside the store, the second one allows to book an entrance for a specific time, finally the last one is oriented to a customer-view, it guarantees to avoid customer forgetfulness.

[G3] provides an estimation of waiting time, instead of telling the customer in which position in queue he/she is. That's motivated by the possibility to organize the queue in different queue and makes the structure flexible as possible for the future improvements

Finally, [G6] helps [G2], indeed monitoring the entrances (using a code generated by the system and a smartphone to scan it) the regulation of the flux could be more effective. In particular, [G2] permits a better management in proximity of the building, indeed with a monitor the store can communicate whose the turn is. The choice of monitor is motivated by the presence of the hand-out customer.

1.4 – Domain Assumption

- [D1]: The technology to monitor entrances and leavings never fail
- [D2]: The store manager never sets non integer or negative values
- [D3]: The code generated will never be lost by hand-out customers
- [D4]: The store manager sets the departments of the store
- [D5]: The store manager sets the maximum number of people for every departments in the store
- [D6]: The customer will always reach the store by car
- [D7]: The devices that acquire users' position provide location with an error of 20 meters at most
- [D8]: The system knows the position of every store
- [D9]: The customer never turns off GPS
- [D10]: The internet connection works properly without failure

1.5 – Analysis and considerations about client requests

This section is dedicated to every consideration done in order to produce the most effective system balancing costs and requests for the customer.

In particular, the best way to plan finer the visit in the store is to assume that all the optional information the customer will put in the reservation, such as what type of items he/she will purchase and how much time she/he will stay inside the store, are respected. These two assumptions are rather strong if compared with the common day-life. It's usual to forget to select one item category and then to remember it when we are in the store. It's also commonplace to not estimating correctly the time we need, also if the total time depends also on other factors, such as the queue before the tills.

If they must be assumed true, the customer must build a supervision system that has to check every customer if they're sticking with their choice in order to avoid penalty (in case of health measure control) that the system can't control (because of assumptions). In addition, the estimation of the waiting time for the other customer will be necessarily worse because the system will assume true that the visit duration is always correct; in case of delay on this, every waiting time will increase due to that.

In this document, both will be considered false. In case of duration estimation, the system (through a user profile) will calculate the difference between his/her older duration visits guesses and the effective durations in order to provide a better approximation for the following ones.

In case of departments assumption, all customers are considered in every department at the same time, so no health measure way can be violated. Anyway, in this version the possibility to specify the item categories will be implemented. Whereas the client would build the supervision system, the functionality can be added in the further versions of the system.

1.6 – Definition, acronyms and abbreviations

1.6.1 – Definition

- Active customers: customers inside the building at the same time
- Delta delay: the difference between actual time spent inside the store by the customer and his/her estimation of the duration
- Hand-out customers: customers who don't have access to the required technology or are not able to use it
- To line up: type of reservation in which the customer is put in a virtual queue

1.6.2 – Acronyms

- RASD: Requirements Analysis and Specification Document

1.6.3 – Abbreviations

- CLup; CLup – Customer Line up
- [Gn]: n-goal.
- [Dn]: n-domain assumption.
- [Rn]: n-functional requirement.

1.7 – Revision History

Date	Version	
15/12	1.0	
20/12	2.0	• Change model: added new different actors • Reorganize goals • Change Requirements • Update UML • Reorganization Alloy code

1.8 – References

- IEEE Recommended Practice For Software Requirements Specifications - IEEE Std 830-1998
- 2.Alloy.pdf from Polimi Beep
- 4.f Structure of RASD from Polimi Beep
- https://www.uml-diagrams.org/
- https://creately.com/it/home/
- https://www.figma.com/

1.9 – Overview

The RASD is composed by five parts:

1. The first part focus on a briefly introduction to the current situation, to the solutions offered by the product, emphasizing its goals. This section analyses the current system available in the market, highlighting the main differences between them. Moreover, it specifies which conventions the document will adopt, to understand better the following part.
2. The second part offers a perspective of the product, underlining the main functions, attributes, constraints and user characteristic. It will also provide an initial explanation of requirements.
3. The third part focuses on all requirements and on use cases. It will also provide scenarios to justify all the functions.
4. The fourth part provides an explanation with Alloy, also reporting the results.
5. The fifth part is about the effort spent by the team to build this document.

2 – Overall description

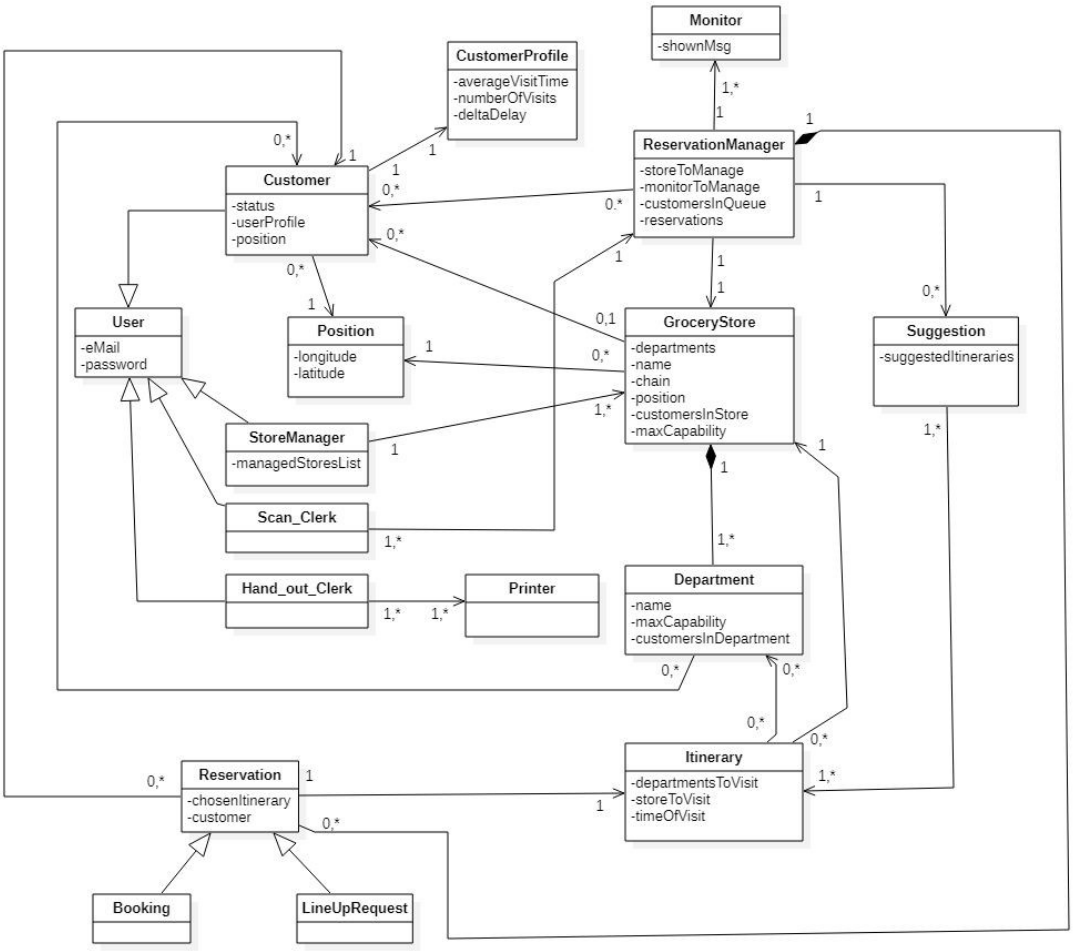
This section of the RASD should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 and makes them easier to understand.

2.1 – Product perspective

This product will be developed from scratch and, as said in the next section, it will use an external tool (Google Maps) to collect user position on which the alert and calculation system is based. It will ensure precision to our calculation and a better user experience because, usually, the user is accustomed to this tool.

The product, except for this, is almost totally self-contained.

To analyse better the structure, a UML class diagram is presented below, which is a high-level description of the component of the system.



Customer, *StoreManager*, *ScanClerk* and *Hand_out_clerk* are all subclasses of *User* because they have the attributes eMail and password and share the same log in and registration method.

ScanClerk is a special account which permits to scan the unique codes to allow customers to enter the store.

HandOutClerk is a special account which permits to do multiple line ups for different "Hand Out Customer" (it doesn't have a limit on the amount of possible line ups).

CustomerProfile is a class containing useful informations about every customer like averageVisitTime, deltaDelay represents the difference between the time a customer declares to stay in the store and the time he actually stays in the store, this informations are updated by Reservation when a customer exits the store.

GroceryStore and *Department* classes contain a list of Customer objects, these customers are the ones actually occupying the department and the store.

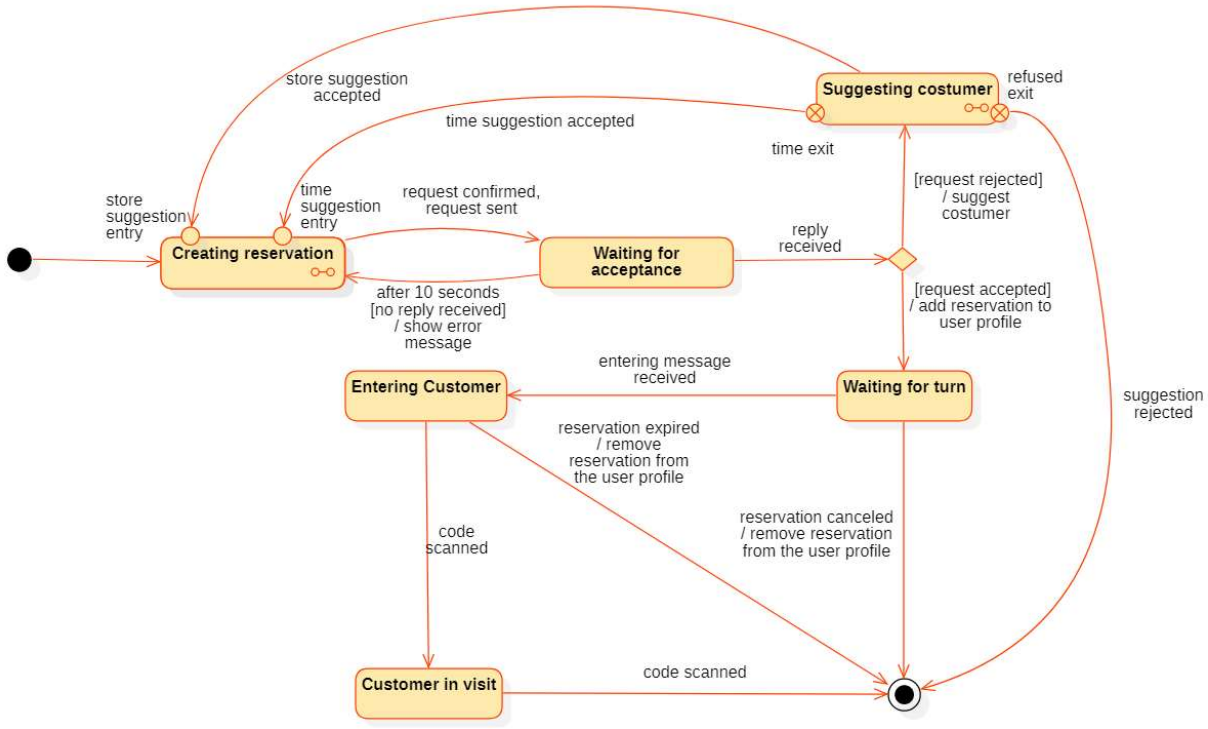
The purpose of the class *ReservationManager* is to manage the line up queue and to manage all the possible reservations (both *Booking* and *LineUp_Request*) so it contains a list of *Reservation* type objects (formalized via composition), it is also responsible to make new Suggestions.

Itinerary is a class representing the itinerary on a store that a customer declares to do on a reservation, so it contains the time in which the visit will be done, the chosen Store and a List of Department that could be empty if the customer doesn't want to specify this optional information

Suggestion is a class whose purpose is to suggest new itineraries on refused reservations, to do so it contains a list elements of class Itinerary.

2.1.1 – Behavioural state machine

According to the possible situations and to the different point of view (e.g. for the store manager or the customer), the system will change its state many times and to explain the development of the changing the state-charts will be introduced below.



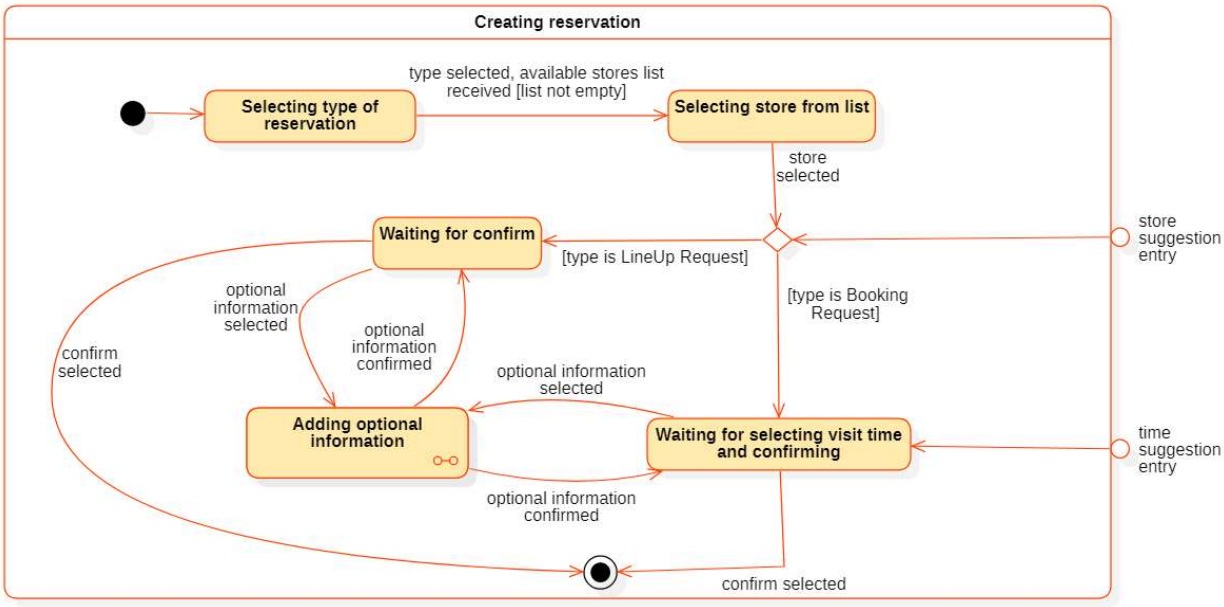
This diagram analyses the states to complete a transaction in the customer side.

The composite states with the hidden decomposition are explained in the diagrams below, they're used to make the diagram clearer and more understandable.

The final state is reached in only three situations: the customer entered the store and exited, the reservation is cancelled by the customer or expired due to his/her delay or, finally, the suggestion sent by the system is rejected by the customer, so she/he won't create another request (or will only change the type).

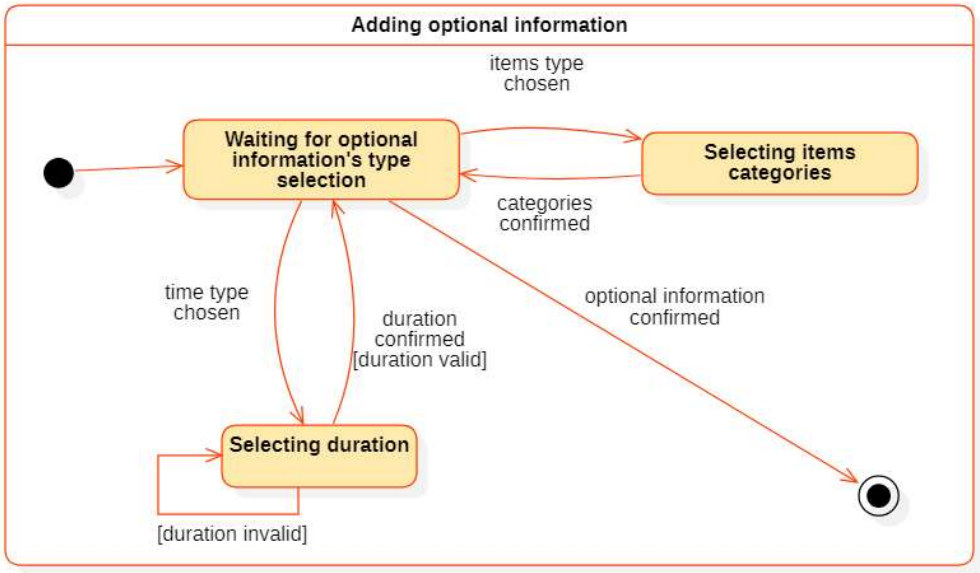
The same event (“code scanned”) occurs from “Entering Customer” and from “Customer in visit” because: the code scanned is the same and the scan system is totally equal both for entrances and for leavings.

To not mess the diagram, the power off transition isn’t inserted. In case of this event, if the system will preserve the reservation created and put in the queue by the system, so the customer could use the associated code (if the power off is ended and the reservation isn’t expired).



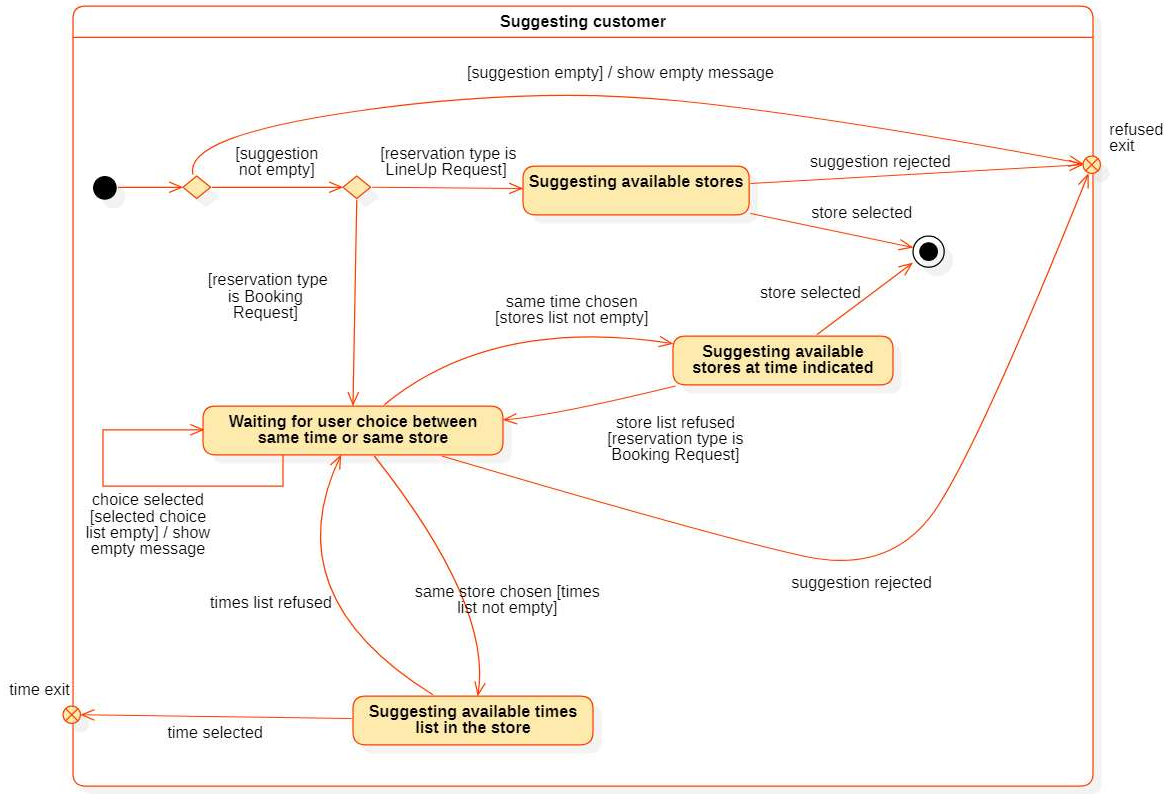
This diagram models the steps to create a reservation: it must contain, obviously, the store in which the customer wants to go and, in case of booking, the time she/he wants to enter. There is also the possibility to add optional information in it, such as which items categories the customer would purchase and a duration estimated (in both type of reservation).

The “confirm” by the customer authorizes the system to send the request to the server system.



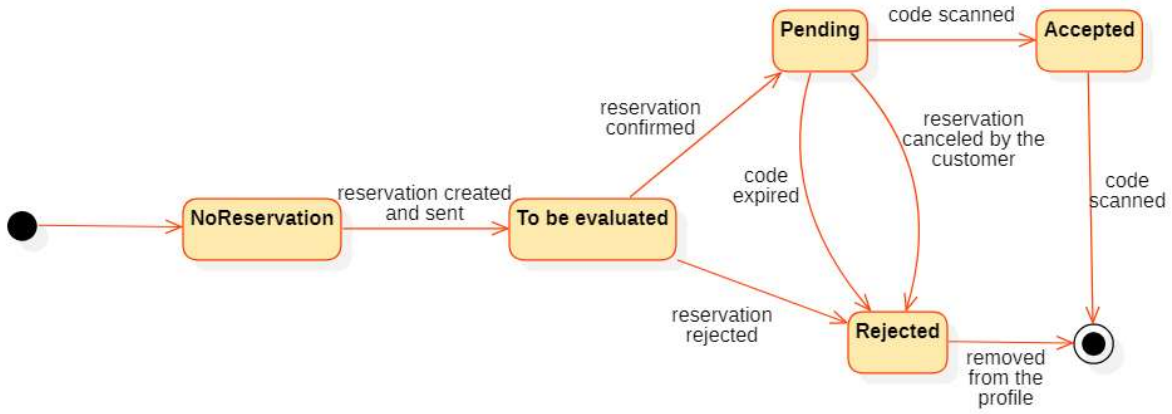
In this state the customer can specify optional information, such as the items categories he will buy and the duration in which he/she will be inside the store.

The client system will control if the inserted duration doesn't violate simple constraints, such as: it isn't bigger than 2 hours and it isn't shorter than 5 minutes (minimum time to buy a single item, pay it and exit the store).



After the rejection, a suggestion by the system is always provided. It could be empty, if there's no availability, or it could have only one list empty (in case of booking) or, also, no part empty.

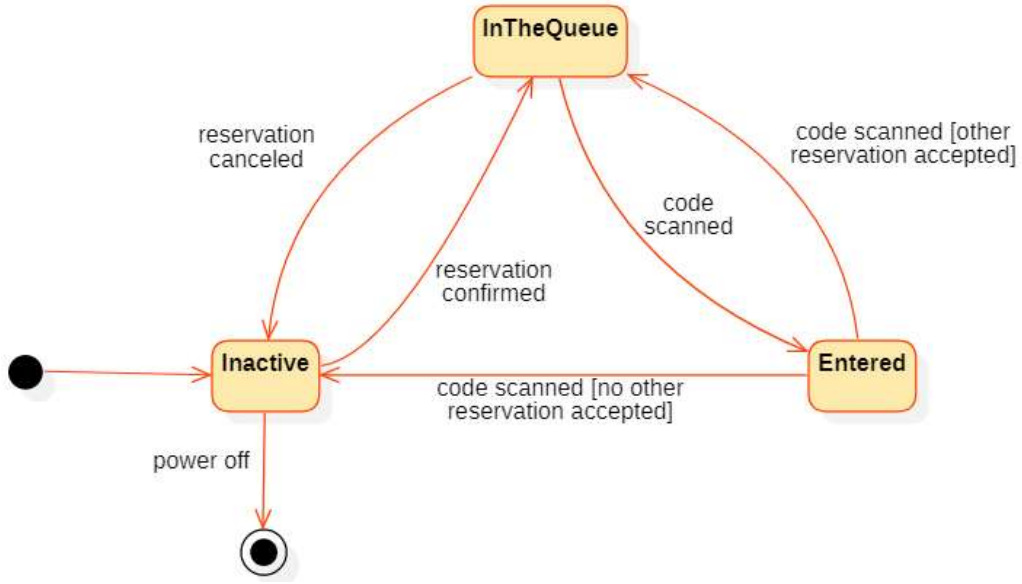
In the first scenario, the suggestion is considered as "refused" by the customer. Obviously, the initial type of request influences the suggestion shape provided by the application, indeed, in case of LineUp Request, the only list showed is that contains the available store without any specification about the time in which they're available. However, in case of Booking Request, the suggestion will contain two lists: one with the available times at the same store and one with the available stores at the time indicated in the reservation. In the booking the state "Waiting for user choice between same time or same store" guarantees the customer's possibility to choose which list he/she wants to read. If the selected one is empty, a message is shown.



The evolution of the reservation states is described in this diagram. A reservation could be:

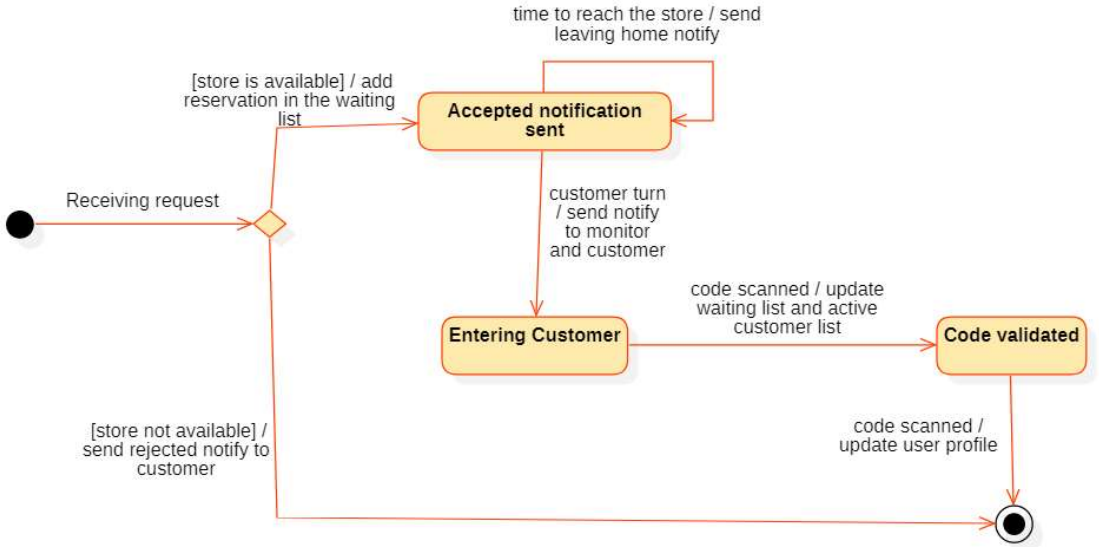
- ToBeEvaluated: the system hadn't checked it yet.
- Pending: the system accepted the request and put it in the queue
- Rejected: the system rejected the request or the customer cancelled the reservation.
- Accepted: the customer is entered

With the second scan of the code the reservation data are registered in the user profile.



A customer could be:

- Inactive: he/she has pending/accepted reservation.
- InTheQueue: he/she has no accepted reservation and at least one in state "Pending"
- Entered: he/she has one reservation accepted (obviously, it's impossible to have more than one because no customer will have ubiquity power)



A server state machines for every new reservation is provide. The main focus is probably the effect generated by the customer turn. A notify is sent to the customer device and also to a monitor that can communicate to every customer who has to enter. It's useful for the hand-out customer. After the second code scanned the user profile is updated and the delta delay calculated.

2.1.2 – User interfaces

The user interfaces must contain short message with a neutral font. It must be intuitive due to the vast audience nature. The notifications must be simple and, also, with short message, such as “It’s time to go” for the incoming time of departure, or “It’s your turn”. In addition to that, the used colours can’t be green and red to explain different concepts, such as “red” for “not your turn” and “green” for “it’s your turn” because of the possible presence of colour-blindness. Hereunder, some examples are showed.

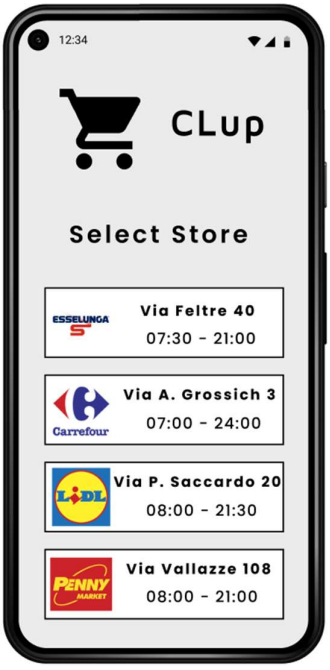
Mockup 1 – Icon

Mockup 2 – Login / Sign Up

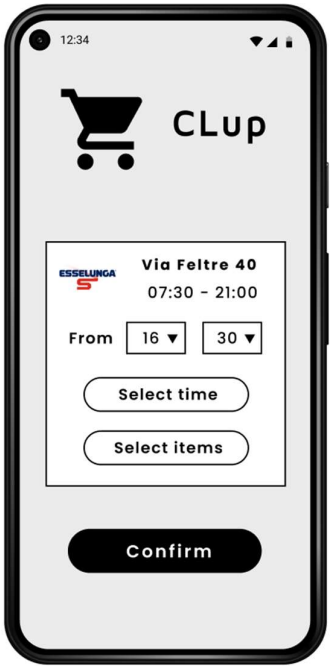
Mockup 3 – Sign Up Form

Mockup 4 – Login Form

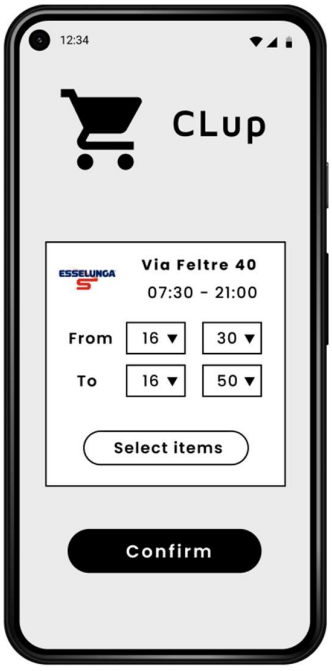
Mockup 5 – Customer Home



Mockup 6 – List of Store

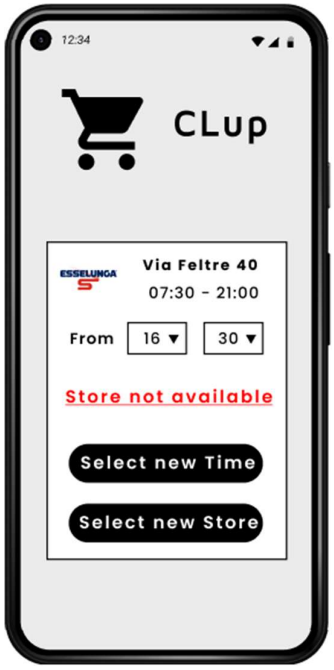


Mockup 7 – Reservation after selecting store



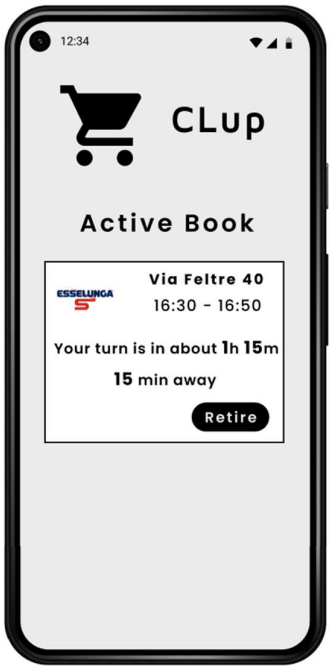
Mockup 8 – Reservation after selecting time of visit

Mockup 9 – List of Departments



Mockup 10 – Store not available

Mockup 11 – Customer Profile



Mockup 12 – Active Booking Info

Mockup 13 – Store Manager Home

Mockup 14 – Departments Live Info

2.1.3 – Hardware interfaces

The system has no hardware interfaces.

2.1.4 – Software interfaces

The system doesn't provide any API to external applications.

2.2 – Product functions

In the direction of defining unambiguous boundaries for the system in development, a specific list of the functions that the product will/won't offer is provided and precisely explained. The first four is related to the user need, the second part regards the store manager uses.

2.2.1 – User Registration / Authentication

It guarantees that a user can register himself/herself to the application to gain all the service the system provides.

2.2.2 – Line up / Book a visit

2.2.2.1 – Line up

It permits the customer to access the first available free slot in the queue, according also with the specified information given by him/her. The possibility to line up is calculated dynamically by the system. The first available free slot is not always at the end of the queue. It is also possible that a customer makes multiple lines up.

2.2.2.2 – Book a visit

The customer could make a booking for a specific time in a specific store. This reservation type has a higher priority than the line-up. If a booking is accepted, only a delay in the customer leavings could increase the waiting time, otherwise the time in the reservation will be always respected, no customer can skip it if it causes an increase of the booking waiting time.

2.2.2.3 – Common features

The system permits the possibility to put the customer in the online queue. A unique code will be generated and assigned to every reservation and their waiting time will be continuously updated. A notification will be also sent in two main cases:

a. The waiting time is updated to 0 and it is customer turn
b. The waiting time is equal (or greater with a range of tolerance) to the estimating spent time in reaching the store.

The estimating time in which the customer will reach the store is extrapolated using the position the store, the customer's one and Google Maps to calculate it. Google Maps will always be used in "car mode". A possible improvement in the further version could be to ask customer in which way they want to reach the store.

2.2.3 – Cancel the reservation

In case of possible mistakes, the users are able to cancel the reservations done and they can make another one. The possibility to delete that is forbidden, obviously, if the customer is already entered.

2.2.4 – Queue management

The system manages all aspects around the queue: it maintains the customers position in the "line", it keeps tracking of the user inside the building to avoid the overgoing of the health measures constraint. In addition to that, it controls and manages the user delays (with a tolerance parameter) or reservations cancels and organizes the queue in a way that can support them, avoiding, for instance, the possibility to give 5 minutes to a user that needs 10 minutes to reach the store.

The end of the queue is managed dynamically, indeed from the closure store time the system calculates when to stop accepting reservations.

Moreover, every new reservation the queue is re-organized to avoid wasting of available time. This last part has a strong rule: the queue is modified in a way that the waiting time of every customer can't never increase. If a change in the line causes an increase in a waiting time of a customer, it's rejected.

2.2.5 – Registration of the store

It's possible to register every store by own manager through authentication, also in case of store chain. The app supports the insertion of the timetable, the type of items, the mean average of the customers visit time, the maximum number of people inside the store, the maximum number of people per department.

2.2.6 – Collecting user data

The system collects for every user the duration of his/her visit, calculating an average for that costumer (in case she/he won't put an estimation in the following reservation).
With it the system becomes able to calculate an average of the "delta delay" in order to provide a better approximation for the waiting time of the other customers.

2.2.7 – Store Manager Authentication

Every store manager is able to login in the application and accesses to the other management functions.

2.2.8 – Creation of special accounts

The special accounts are divided in two types: Scan_Clerk and Handout_Clerk.

The first one permits the monitoring of the entrances: this account is the only one with a section dedicated in the code scan through the smartphone camera. The scanning of the code of the specific reservation guarantees an update in the information contained in the system, such as removing the specific reservation from the queue and putting it in the store active customer.

The second one allows people, who don't have access to the required technology or can't operate with it, to use the system service. Indeed, it permits to create a potential unlimited number of reservation (obviously if the store is still available for another requests). Due to the customer technology conditions, the reservations are printed by a printer linked to this account. The ticket will contain the code that has to be scanned and an estimation of the waiting time. The usual first notify, which advise the customer to leave to reach in time the store, are useless because they're considered as customer with zero distance from the store. The second notify, which communicates the customer that it's his/her turn, is shown by a monitor directly connected with the system.

The hand-out customers have no privilege compared to the online ones. The only case in which they can skip the line is when a customer is absent (or beyond the tolerated delay) and no customer will reach the store in time.

2.2.9 – Display store's info

The store manager is able to see dynamically the collected data about the store, so what is the average of the visit, how many customers are inside the building, how long the queue is.

3 – Specific Requirements

3.1 – Functional Requirements

3.1.1 – Customer Scenarios

Scenario 1

Maurizio is a desperate housewife that wants to cook a lasagna for his wife's birthday. Opening the fridge, he realizes there is no bechamel. Outside it's very cold and Maurizio doesn't want to wait in the queue and to risk a flu. He has also to clean the kitchen before cooking and he can't be that if she must wait outside the grocery store.
Fortunately, he downloaded CLup and is already registered, so: he opens the app and lines up for the nearest grocery store. The app estimates the arrival time from his position to the store is about 7 minutes and it shows him the total waiting time (that's about 20 minutes), so he can start cleaning the kitchen. He hasn't always to control the time because the app notifies that his turn is coming in 10 minutes, so Maurizio successes in cleaning the kitchen and goes to the grocery to buy bechamel.

Scenario 2

Franco is an off-campus student who is really busy due to lessons, laboratories and sports. His timetable is a very strict schedule, he has a determined free interval time for every day, in which he must doing shopping. He can use CLup to solve this problem: instead of wasting time in a queue and risking to not doing the shopping because of that, he can book a visit to his favourite store. Moreover, he can specify the expected time of a visit and the items he's going to buy. By booking Franco is ensured he will go grocery shopping at the time he wants and be safe at home as soon as possible where he can eat his favourite dish: "pasta con tonno".

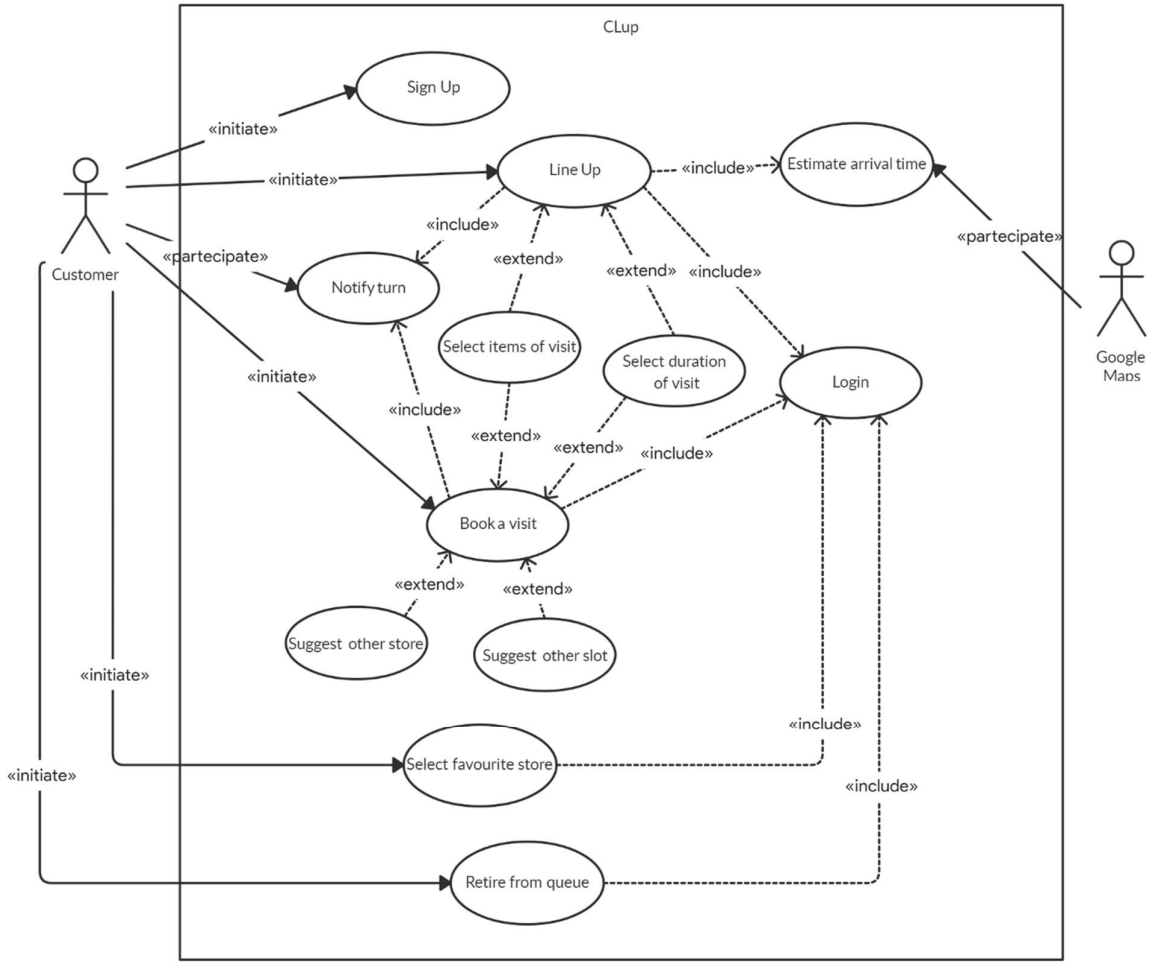
Scenario 3

Sara is a delivery woman who works all day, she prefers to do grocery shopping very late, but not always her favourite grocery store is open. When she books her visit and her favourite grocery shop is closed or full, the system suggests her other stores open at that time so she can keep doing grocery shopping whenever she wants.

Scenario 4

Michele is a scatterbrain guy. He forgot that this evening he will have to prepare a dinner for his childhood friends. Using CLup, he's trying to book a visit in the only grocery store that has sushi packs. Unfortunately, the interval time he planned to go is already full because he chose it late. In this case, CLup suggests to Michele alternative slots in the same store, so he can fix to his forgetfulness and pick the sushi packs for the dinner.

Use Case Diagram



Use Cases

ID	**1**
Name	Sign Up
Actor	Customer
Input	E-mail and password
Entry Conditions	The system does not contain any customer registered with the e-mail provided
Events Flow	1. The customer chooses the "Sign up" option. 2. The customer fills the mandatory fields. 3. The customer chooses the confirmation option. 4. The system saves the data.
Exit Conditions	The customer is registered and the system has his data stored.
Output	Confirm registration message
Exceptions	1. The customer is already signed up. In this case the system warns the user and suggests him/her to login. 2. The customer didn't fill all of the mandatory fields with valid data. 3. The e-mail is already registered. All the exceptions are handled by notifying the customer and taking him/her back to the sign up activity.

ID	**2**
Name	Login
Actor	Customer
Input	E-mail and password
Entry Conditions	The system contains a customer registered with the e-mail provided
Events Flow	1. The customer clicks on the "Login" button. 2. The customer enters username and password in the respective fields. 3. The customer clicks on the "Confirm" button.
Exit Conditions	The customer is logged in and the system take him to the home page
Output	Confirm login message
Exceptions	1. The customer enters invalid E-mail. 2. The customer enters invalid Password. All the exceptions are handled by notifying the customer and taking him/her back to the login activity.

ID	3
Name	Select favourite store
Actor	Customer
Input	Store selected
Entry Conditions	The customer has already logged in (Use Case #2).
Events Flow	1. The customer clicks on the “Profile” button. 2. The customer clicks on the “Change” button. 3. The customer chooses between the possible stores.
Exit Conditions	The customer chooses his/her favourite store and the preference is saved to his/her profile
Output	Confirm change message
Exceptions	-

ID	4
Name	Line Up
Actor	Customer
Input	Store selected
Entry Conditions	The customer has already logged in (Use Case #2).
Events Flow	1. The customer clicks on the “Line Up” button. 2. The customer selects the store he/she wants to line up. 3. Possible start of Use Case #8. 4. Possible start of Use Case #9. 5. The customer clicks on “Confirm” button.
Exit Conditions	The customer is regularly lined up to the selected store
Output	Confirm line-up message
Exceptions	1. The store he/she has selected is closed. 2. The waiting time exceeds the timetable of the store.

ID	5
Name	Book a visit
Actor	Customer
Input	Store selected, time of entry at the store
Entry Conditions	The customer has already logged in (Use Case #2).
Events Flow	1. The customer clicks on the "Book Visit" button. 2. The customer selects the store he/she wants to book for. 3. The customer selects the time of his/her visit. 4. Possible start of Use Case #8. 5. Possible start of Use Case #9. 6. The customer clicks on "Confirm" button. 7. Possible start of Use Case #6. 8. Possible start of Use Case #7.
Exit Conditions	The customer has booked a visit to the selected store starting from the selected time
Output	Confirm booking message
Exceptions	1. The same visit already booked All the exceptions are handled by notifying the customer and taking him/her back to the book a visit activity.

ID	6
Name	Suggest other store
Actor	Customer
Input	Store selected
Entry Conditions	1. The customer is booking a visit to a store (Use Case #5). 2. At the time of the visit the store is either closed or full.
Events Flow	1. The customer selects "Change store" 2. The customer selects the store he/she wants to book for, different from the previous. 3. The customer clicks on "Confirm" button.
Exit Conditions	The customer has booked a visit to the selected store starting from the selected time
Output	Change store message
Exceptions	-

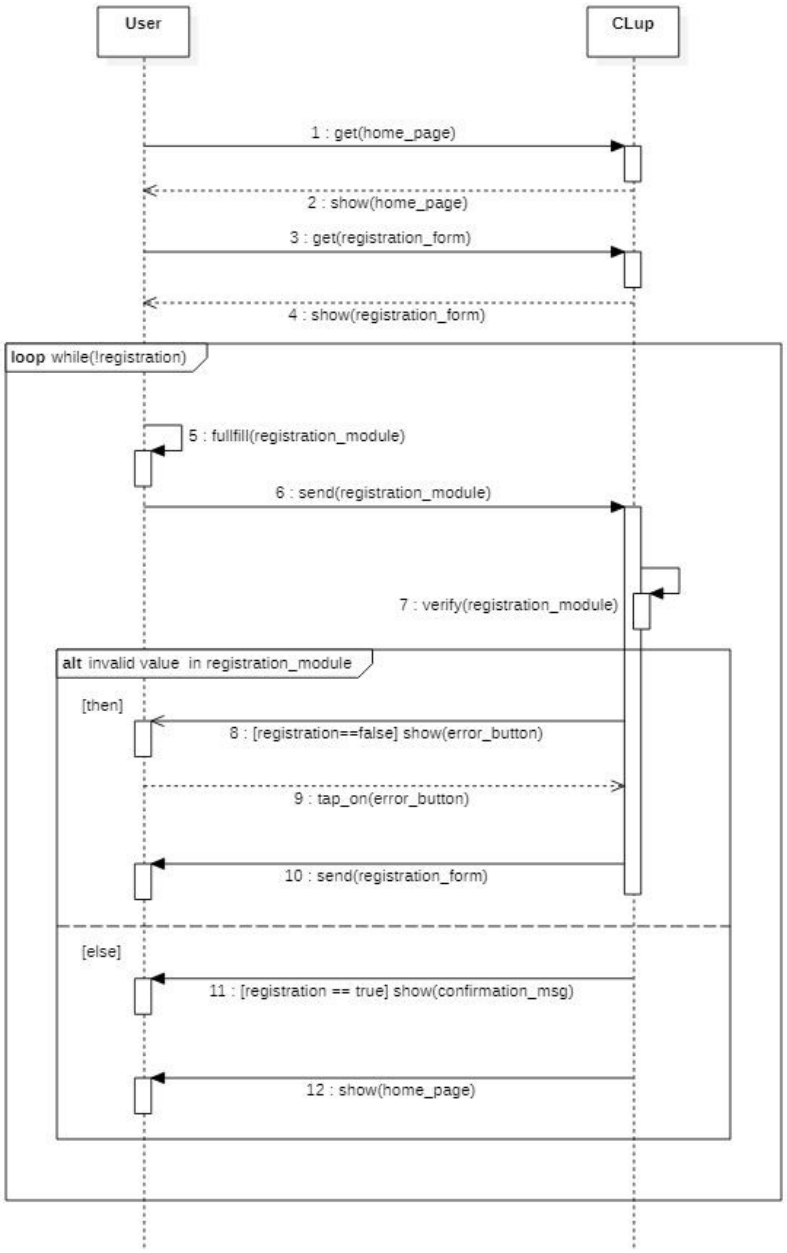
ID	7
Name	Suggest other slot
Actor	Customer
Input	Time of entry at the store
Entry Conditions	1. The customer is booking a visit to a store (Use Case #5). 2. At the time of the visit the store is either closed or full.
Events Flow	1. The customer selects “Change slot” 2. The customer selects the time he/she wants to book for, different from the previous. 3. The customer clicks on “Confirm” button.
Exit Conditions	The customer has booked a visit to the selected store starting from the selected time
Output	Change slot message
Exceptions	-

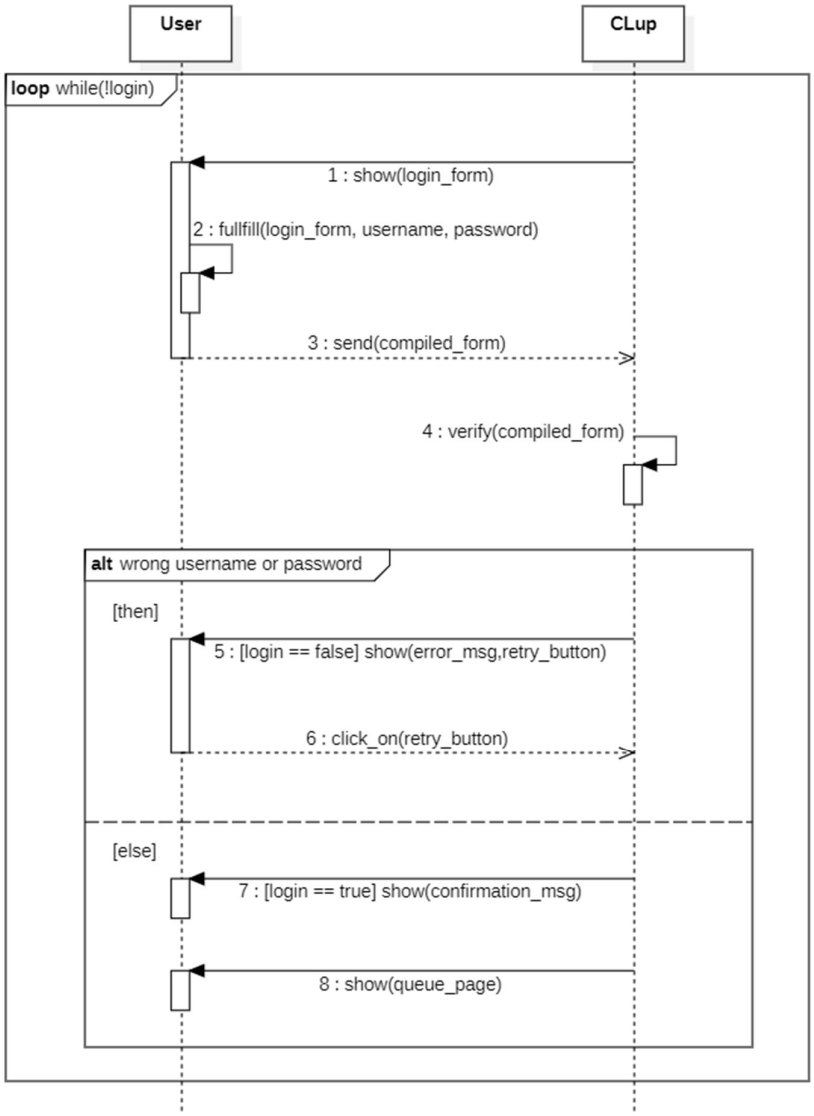
ID	8
Name	Select duration of visit
Actor	Customer
Input	Time of exit from the store
Entry Conditions	The customer is booking a visit to a store (Use Case #4 or #5).
Events Flow	1. The customer clicks on the “Select time” button. 2. The customer selects the duration of his/her visit. 3. The customer clicks on the “Confirm” button
Exit Conditions	The customer has selected the duration of his/her visit and it’s saved in the reservation
Output	-
Exceptions	1. The time of the visit exceeds the store timetable All the exceptions are handled by notifying the customer and taking him/her back to the select time of visit activity.

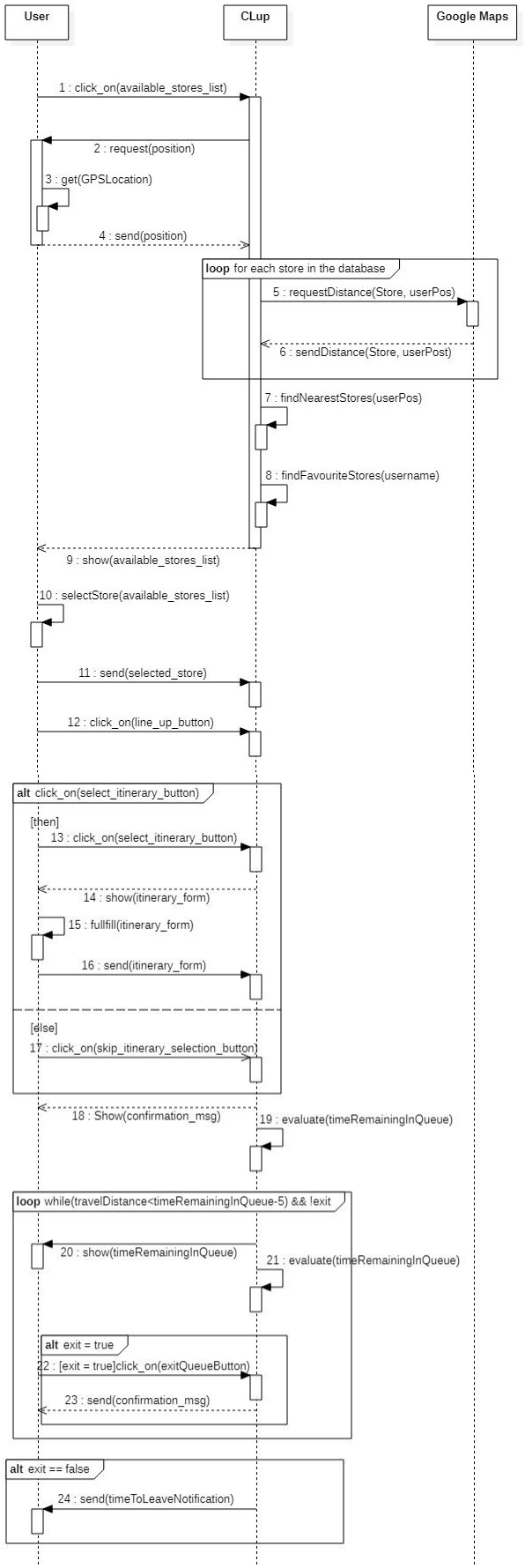
ID	9
Name	Select items of visit
Actor	Customer
Input	The departments the customer intends to visit during the visit of the store
Entry Conditions	The customer is booking a visit to a store (Use Case #4 or #5).
Events Flow	1. The customer clicks on the “Select departments” button. 2. The customer selects the departments of his/her visit. 3. The customer clicks on the “Confirm” button
Exit Conditions	The customer has selected the items of his/her visit and they’re saved in the reservation
Output	-
Exceptions	-

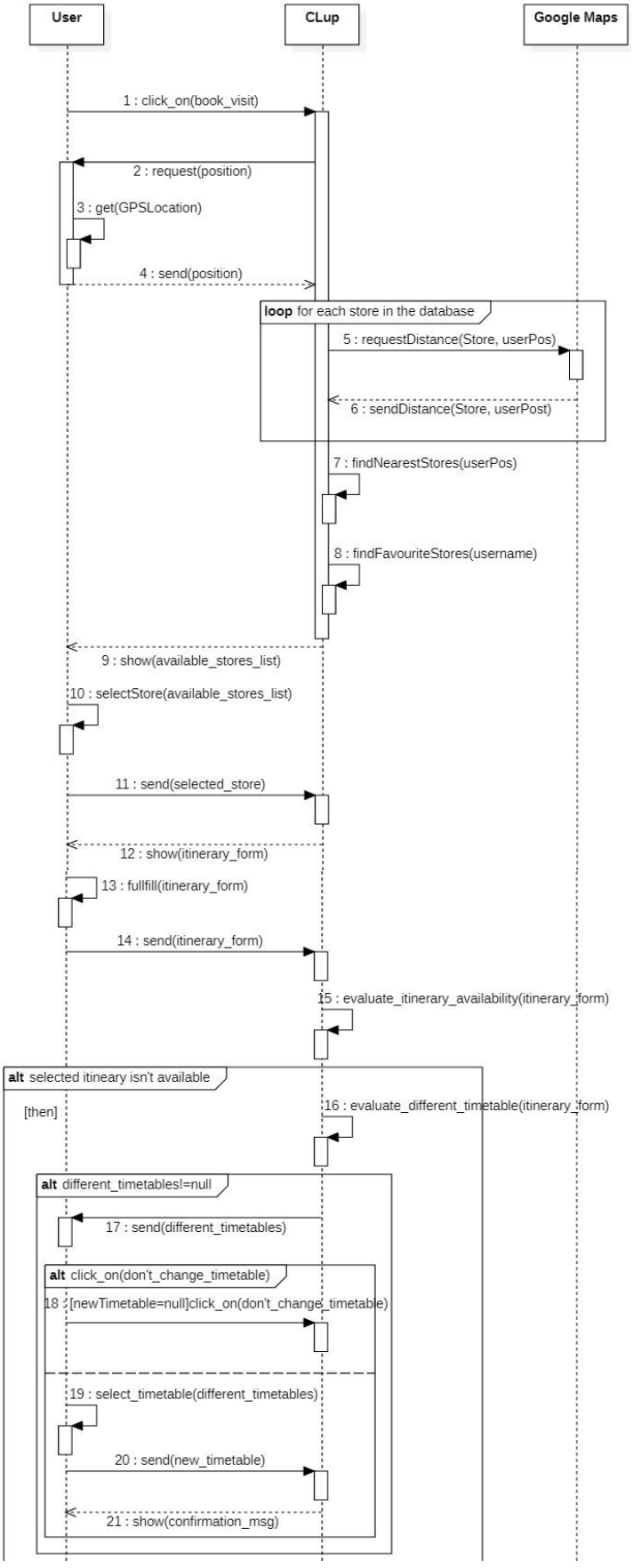
ID	10
Name	Retire from a queue
Actor	Customer
Input	-
Entry Conditions	The customer has lined up or booked a visit to a store (Use Case #4 or #5).
Events Flow	1. The customer clicks on the “Profile” button. 2. The customer clicks on the “Active Book” button. 3. The customer clicks on the “Retire” button
Exit Conditions	The customer has exited the queue of the store he has previous selected
Output	Confirm retire message
Exceptions	1. The customer has already entered the store All the exceptions are handled by notifying the user and taking him/her back to the active book screen.

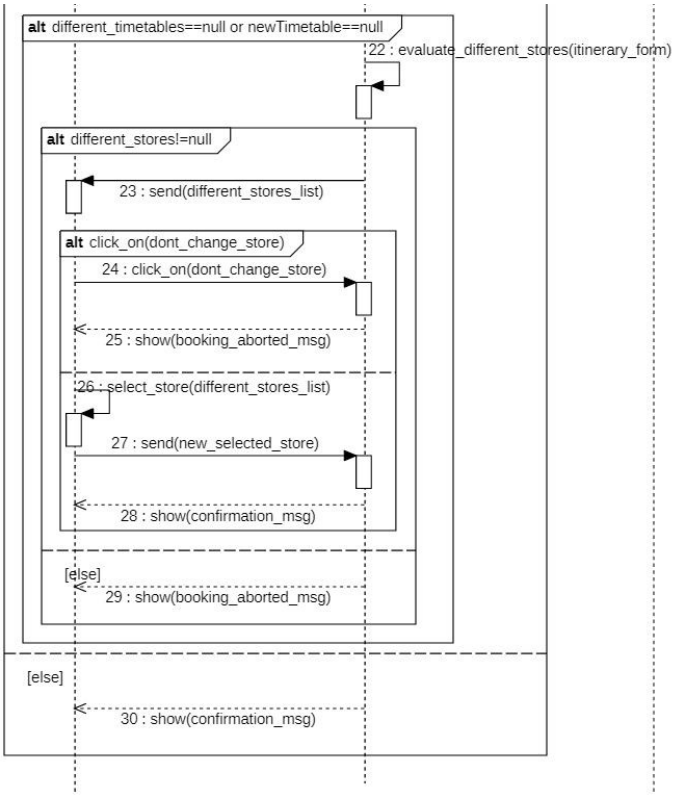
Sequence Diagram




User
CLup
loop while(!login)
1 : show(login_form)
2 : fullfill(login_form, username, password)
3 : send(compiled_form)
4 : verify(compiled_form)
alt wrong username or password
[then]
5 : [login == false] show(error_msg,retry_button)
6 : click_on(retry_button)
[else]
7 : [login == true] show(confirmation_msg)
8 : show(queue_page)


User
CLup
Google Maps
1 : click_on(available_stores_list)
2 : request(position)
3 : get(GPSLocation)
4 : send(position)
loop for each store in the database
5 : requestDistance(Store, userPos)
6 : sendDistance(Store, userPost)
7 : findNearestStores(userPos)
8 : findFavouriteStores(username)
9 : show(available_stores_list)
10 : selectStore(available_stores_list)
11 : send(selected_store)
12 : click_on(line_up_button)
alt click_on(select_itinerary_button)
[then]
13 : click_on(select_itinerary_button)
14 : show(itinerary_form)
15 : fullfill(itinerary_form)
16 : send(itinerary_form)
[else]
17 : click_on(skip_itinerary_selection_button)
18 : Show(confirmation_msg)
19 : evaluate(timeRemainingInQueue)
loop while(travelDistance<timeRemainingInQueue-5) && !exit
20 : show(timeRemainingInQueue)
21 : evaluate(timeRemainingInQueue)
alt exit = true
22 : [exit = true]click_on(exitQueueButton)
23 : send(confirmation_msg)
alt exit == false
24 : send(timeToLeaveNotification)


User
CLup
Google Maps
1 : click_on(book_visit)
2 : request(position)
3 : get(GPSLocation)
4 : send(position)
loop for each store in the database
5 : requestDistance(Store, userPos)
6 : sendDistance(Store, userPost)
7 : findNearestStores(userPos)
8 : findFavouriteStores(username)
9 : show(available_stores_list)
10 : selectStore(available_stores_list)
11 : send(selected_store)
12 : show(itinerary_form)
13 : fullfill(itinerary_form)
14 : send(itinerary_form)
15 : evaluate_itinerary_availability(itinerary_form)
alt selected itineary isn't available
[then]
16 : evaluate_different_timetable(itinerary_form)
alt different_timetables!=null
17 : send(different_timetables)
alt click_on(don't_change_timetable)
18 : [newTimetable=null]click_on(don't_change_timetable)
19 : select_timetable(different_timetables)
20 : send(new_timetable)
21 : show(confirmation_msg)

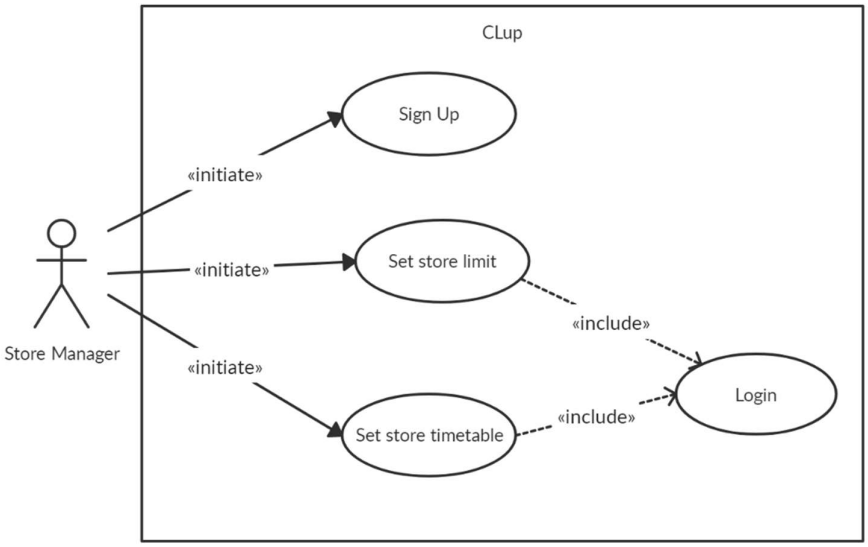

alt different_timetables==null or newTimetable==null
22 : evaluate_different_stores(itinerary_form)
alt different_stores!=null
23 : send(different_stores_list)
alt click_on(dont_change_store)
24 : click_on(dont_change_store)
25 : show(booking_aborted_msg)
26 : select_store(different_stores_list)
27 : send(new_selected_store)
28 : show(confirmation_msg)
[else]
29 : show(booking_aborted_msg)
[else]
30 : show(confirmation_msg)

3.1.2 – Store Manager Scenarios

Scenario 1

While the emergency is going on, due to this, Italian Prime Minister issues DPCM that handles lockdowns and red zones. Leopolda, a Carrefour store manager, reading the last issued DPCM that puts more strict rules (in particular, for the crowds in grocery stores). She must find a solution to guarantee the sanitary measures, the correctness of the queue and that in the store can access a determined number of people.
Therefore, she decides to use CLup because he can set the maximum number and changing it every time he wants and the system will register this value and it will manage queue according to that.

Use Case Diagrams



Use Cases

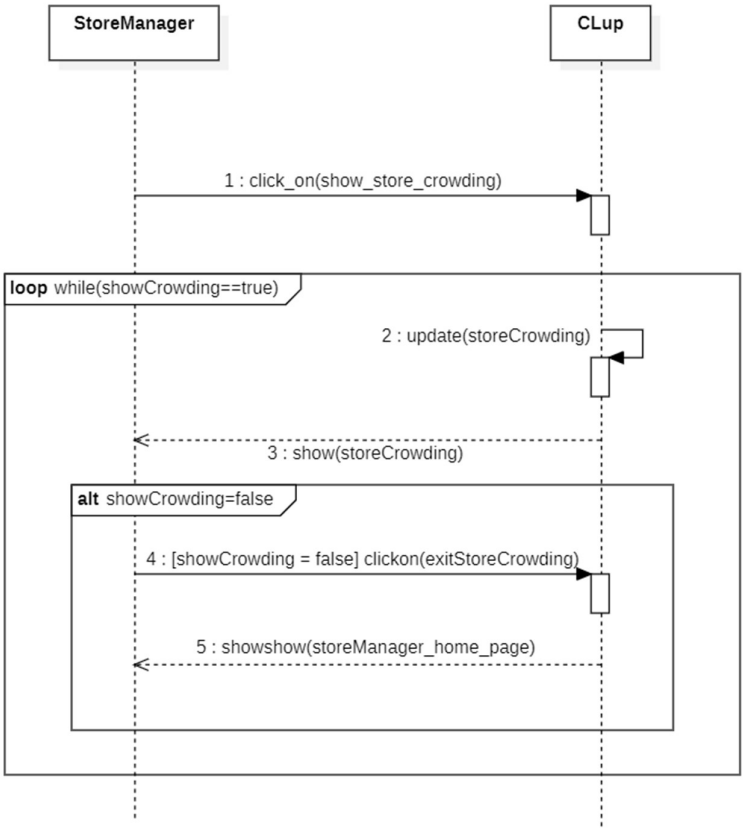
ID	**11**
Name	Sign Up
Actor	Store Manager
Input	E-mail, password and store managed
Entry Conditions	The system does not contain any store manager registered with the e-mail and the store provided
Events Flow	1. The store manager chooses the "Sign up" option. 2. The store manager fills the mandatory fields. 3. The store manager chooses the confirmation option. 4. The system saves the data.
Exit Conditions	The store manager is registered and the system has his data stored.
Output	Confirm registration message
Exceptions	1. The store manager is already signed up. In this case the system warns the user and suggests him/her to login. 2. The store manager didn't fill all of the mandatory fields with valid data. 3. The e-mail is already registered. All the exceptions are handled by notifying the store manager and taking him/her back to the sign up activity.

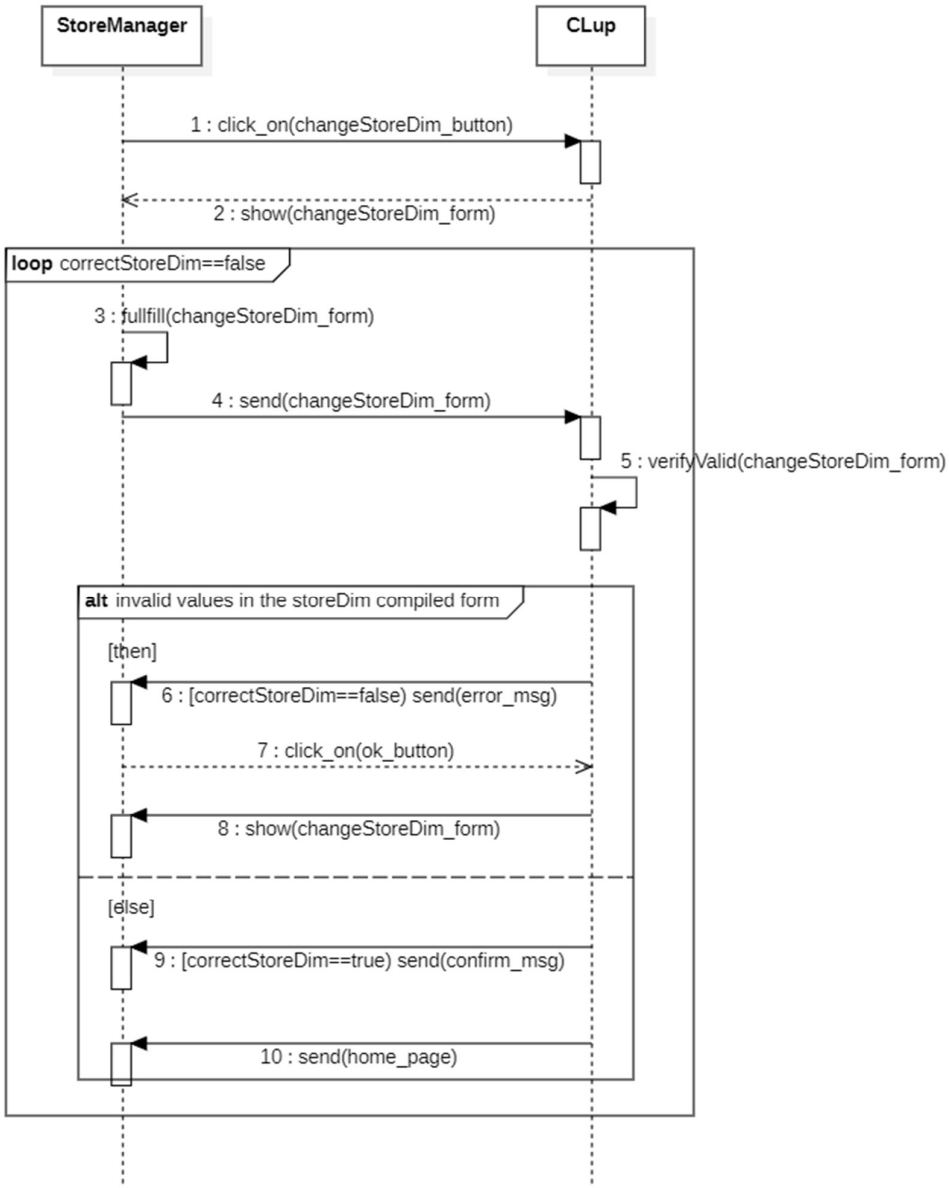
ID	**12**
Name	Login
Actor	Store Manager
Input	E-mail and password
Entry Conditions	The system contains a store manager registered with the e-mail provided.
Events Flow	1. The store manager clicks on the "Login" button. 2. The store manager enters e-mail and password in the respective fields. 3. The store manager clicks on the "Confirm" button.
Exit Conditions	The store manager is logged in and the system take him to the home page.
Output	Confirm login message
Exceptions	1. The store manager enters invalid e-mail. 2. The store manager enters invalid password. All the exceptions are handled by notifying the store manager and taking him/her back to the login activity.

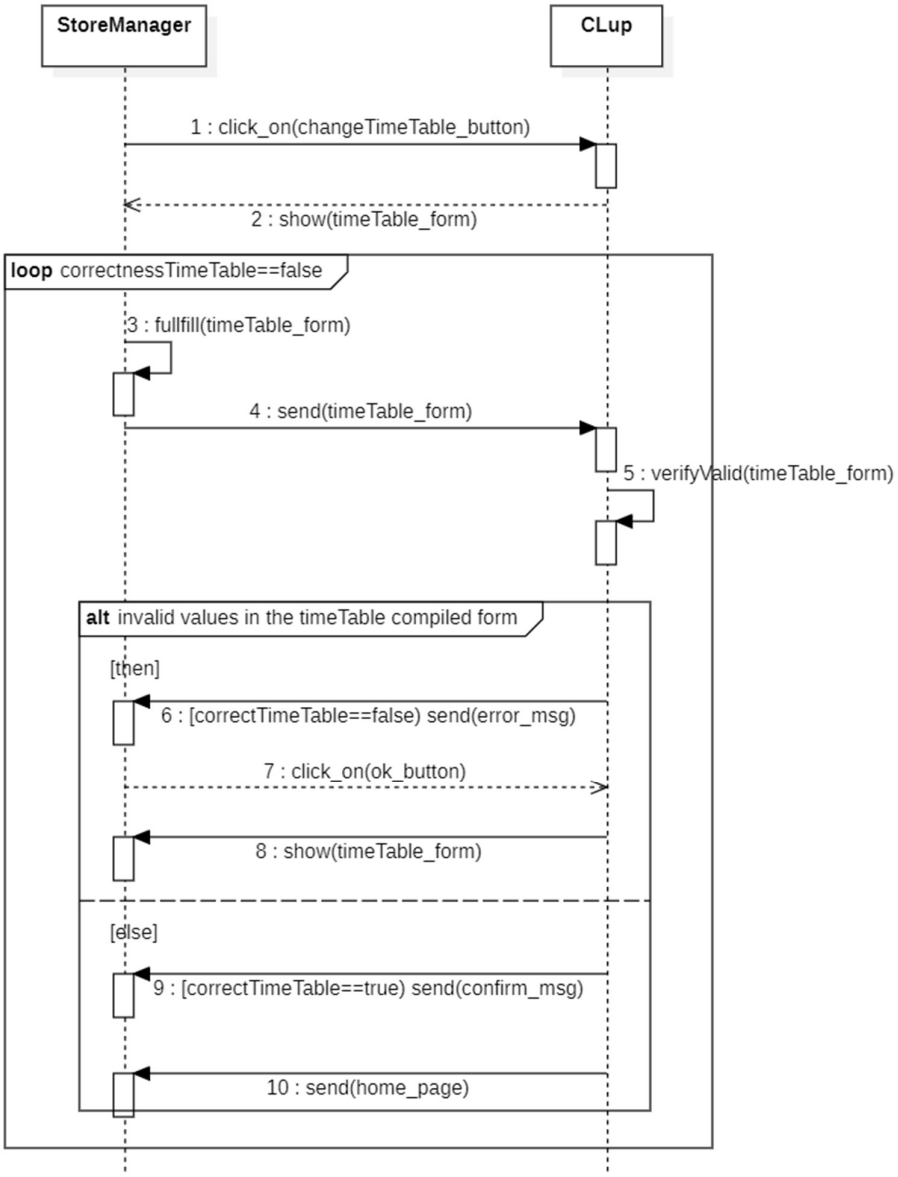
ID	13
Name	Set store limit
Actor	Store Manager
Input	Limit of departments of his/her store
Entry Conditions	The store manager has already logged in.
Events Flow	1. The store manager clicks on the “People Limit” button. 2. The store manager enters the limit. 3. The store manager clicks on the “Confirm” button.
Exit Conditions	The new store limit is set and saved in store manager profile
Output	Confirm change message
Exceptions	1. The new store limit is either zero or negative All the exceptions are handled by notifying the store manager and taking him/her back to the set store limit activity.

ID	14
Name	Set store timetable
Actor	Store Manager
Input	Timetable of the store
Entry Conditions	The store manager has already logged in.
Events Flow	1. The store manager clicks on the “Timetable” button. 2. The store manager enters the timetable. 3. The store manager clicks on the “Confirm” button.
Exit Conditions	The new store timetable is set and saved in store manager profile
Output	Change timetable message
Exceptions	1. Opening time is after closing time

Sequence Diagram




StoreManager
CLup
1 : click_on(changeStoreDim_button)
2 : show(changeStoreDim_form)
loop correctStoreDim==false
3 : fullfill(changeStoreDim_form)
4 : send(changeStoreDim_form)
5 : verifyValid(changeStoreDim_form)
alt invalid values in the storeDim compiled form
[then]
6 : [correctStoreDim==false) send(error_msg)
7 : click_on(ok_button)
8 : show(changeStoreDim_form)
[else]
9 : [correctStoreDim==true) send(confirm_msg)
10 : send(home_page)


StoreManager
CLup
1 : click_on(changeTimeTable_button)
2 : show(timeTable_form)
loop correctnessTimeTable==false
3 : fullfill(timeTable_form)
4 : send(timeTable_form)
5 : verifyValid(timeTable_form)
alt invalid values in the timeTable compiled form
[then]
6 : [correctTimeTable==false) send(error_msg)
7 : click_on(ok_button)
8 : show(timeTable_form)
[else]
9 : [correctTimeTable==true) send(confirm_msg)
10 : send(home_page)

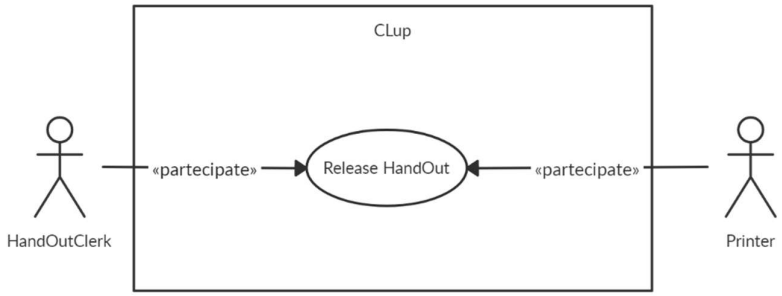
3.1.3 HandOutClerk Scenarios

Scenario 1

Sandrino is a retired veteran, he doesn't like technology, so he doesn't have a smartphone. Therefore, he must go to the store and retrieve hand out ticket printed by the store that shows an estimated waiting time. He's still worried about his waiting time because a lot of people with the app could line up faster.

Fortunately to him, the guy before Sandrino there is Giuseppino, a moody guy, that suddenly decides to not go to the store. Thanks to the queue management system of CLup, after 5 minutes it recognizes that Giuseppino won't arrive, it controls the hand-out ticket queue (noticing that's not empty) and the online one (no one can reach the store), as a result, it allows Sandrino to enter the store and to buy the bresaola he loves

Use Case Diagram



Use Case

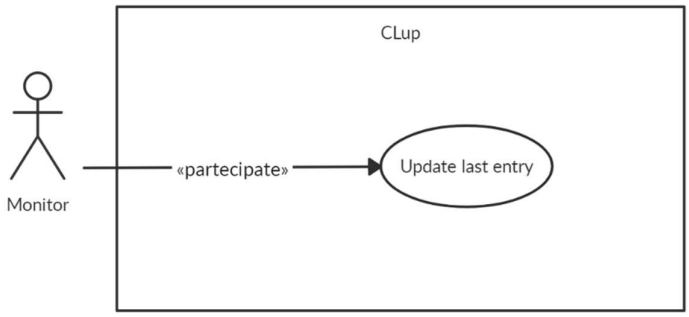
ID	**15**
Name	Release HandOut
Actor	HandOutClerk
Input	Hand-out request
Entry Conditions	The store is open.
Events Flow	1. The system receives an hand-out request. 2. The HandOutClerk release a line-up reservation.
Exit Conditions	The hand-out request is regularly lined up to the store
Output	Confirm reservation message
Exceptions	The waiting time exceeds the timetable of the store.

3.1.4 Monitor Scenarios

Scenario 1

Ginevra is a girl with the head in the clouds. She decides to line up with CLup. After the first notification she leaves home to reach the store. When she arrives at the store, she totally forgets about the second notification that will tell her that's her turn, so he puts her phone in silence mode. Fortunately for her and, in general, for all hand-out customers CLup is connected with a monitor that shows whose the turn is. So, Ginevra can enter the store and make her shopping

Use Case Diagram



Use Case

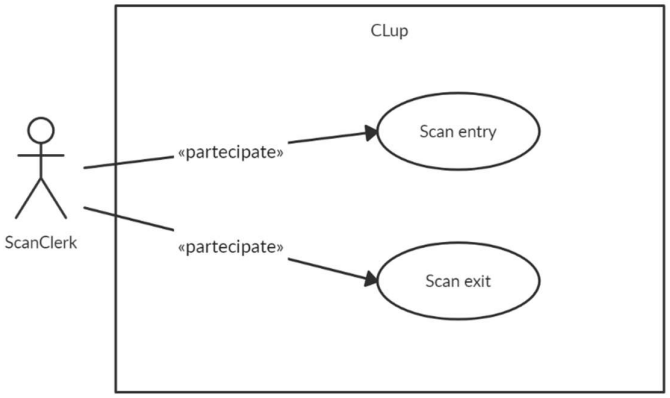
ID	**16**
Name	Update last entry
Actor	Monitor
Input	Update notification from the system
Entry Conditions	The store is open and a customer can enter inside it.
Events Flow	1. The system sends a notification to Monitor. 2. Monitor updates his status.
Exit Conditions	The monitor shows the customer allowed to enter.
Output	Number of reservation of the customer allowed to enter.
Exceptions	-

3.1.5 ScanClerk Scenarios

Scenario 1

Paola is a store manager worried about the possible sanctions if the store won't follow the DPCM rules. She would like to find a way to avoid hiring more staff, having more cost and losing the customers. In particular she doesn't want to have too many people in monitoring the entrances and the leavings.
Fortunately for her, CLup has a special account that permits, through smartphone, the scan of the reservation code, so she can monitor how many customers are inside and can use two people of her staff to do it. Moreover, she won't have any additional cost because all the staff has a smartphone on which Clup can be downloaded. In addition, there's no difference between scanning the entrances and the leavings, so she can manage the staff turns as she wants.

Use Case Diagram



Use Case

ID	**17**
Name	Scan entry
Actor	ScanClerk
Input	Reservation code of a customer
Entry Conditions	The store is opens and a customer get close to the ScanClerk.
Events Flow	1. The ScanClerk scan his/her reservation code.
Exit Conditions	The customer becomes an active customer of the store.
Output	Confirm entry message.
Exceptions	The customer is not allowed to enter the store

ID	**18**
Name	Scan exit
Actor	ScanClerk
Input	Reservation code of a customer
Entry Conditions	The store is open and an active customer of the store get close to the ScanClerk.
Events Flow	1. The ScanClerk scan his/her reservation code.
Exit Conditions	The customer is no longer an active customer of the store.
Output	Confirm exit message.
Exceptions	-

3.2 Requirements

G1: The system must allow customers to line up:

- **R1**: The application allows customers to make a reservation only if it is possible to enter the store
- **R2**: The system generates a unique code for each reservation
- **R3**: The system frees customers from the queue if they're late more than 5 minutes
- **R4**: The system must show the predicted waiting time of the customer
- **R5**: The customer who lines up is inserted in the queue of the store
- **R6:** If a customer has already lined up in a store and he/she hasn't left the queue yet, he/she can't line up again in it
- **R7**: The system must allow customers to specify the duration of their visit
- **R8**: The system must allow customers to specify the departments of the items they want to buy
- **R9**: The system provides the customer a list of the stores opened at that time if the preferred one is unavailable
- **R10**: The system must suggest different stores of the same or even different chain if the preferred one isn't available
- **R11**: The system must allow customers to cancel their reservation before they enter the store
- **R12**: The system confirms the customer the acceptance of the asked reservation
- **R13**: In case of confirmed reservation the system must insert the customer in the queue of the selected store
- **R14**: The system must reject line up requests if the stores chosen are not available
- **R15**: The system must permit only one line up a time for the same store and the same customer
- **D10**: The internet connection works properly without failure

G2: The system must allow store managers to regulate the maximum influx of people in the building

- **R16**: The application limit access to the store if the number of people is equal to the maximum possible
- **R17**: The system must allow to monitor the entrances
- **R18**: The system must allow to monitor the leavings
- **R19**: The system must allow store managers to set the maximum number that can enter the store
- **R20**: The system allows store manager to define the zones of the store
- **R21**: The system allows store manager to define the maximum number of people for every zone in the store
- **R22**: The system must provide the store manager the number of customers for every zone at the moment
- **D1**: The technology to monitor entrances and leavings never fail
- **D2**: The store manager never sets non integer or negative values
- **D3**: The code generated will never be lost by hand-out customers
- **D4**: The store manager sets the departments of the store
- **D5**: The store manager sets the maximum number of people for every departments in the store
- **D10**: The internet connection works properly without failure

G3: The system must provide customers with an estimate of the waiting time

- **R23**: The system must keep track of the average time of permanence of customers in the store
- **R24**: The system sends a notification if the new waiting time is less than 20 minutes in case of missed customers
- **D10**: The internet connection works properly without failure

G4: The system must manage customers entrances

- **R25**: After the scan of the unique code the system removes the customer from the queue
- **R26:** After the scan of the unique code the system inserts the customer into the store list of active customers
- **R27**: The system must send a notification to the customer when it's her/his time to enter.
- **R28**: After an active customer leaves the store (through the re-scanning of the same code) , the same customer is removed from the active customer of the store
- **R29**: When there is a free space in the active customers, the system must decide which is the next customer to enter
- **R30**: If there is a free space available in the store and no online customer can reach the store in time, the first hand out customer in the queue can enter the store
- **D1**: The technology to monitor entrances and leavings never fail
- **D10**: The internet connection works properly without failure

G5: The system must allow customer to book a visit in the store

- **R2**: The system generates a unique code for each reservation
- **R7**: The system must allow customers to specify the duration of their visit
- **R8**: The system must allow customers to specify the departments of the items they want to buy
- **R9**: The system provides the customer a list of the stores opened at that time if the preferred one is unavailable
- **R10**: The system must suggest different stores of the same or even different chain if the preferred one isn't available
- **R11**: The system must allow customers to cancel their reservation before they enter the store
- **R31**: The system permits the customer to select which store he wants to book a visit
- **R32**: The system permits the customer to select when he/she will go to the store
- **R33**: The system confirms the customer the acceptance of the asked reservation
- **R34**: In case of confirmed reservation the system must insert the customer in the queue of the selected store
- **R35**: The system includes suggestions of alternative slots to customers if the selected time isn't available
- **R36**: The system must allow customers to select a preferred store
- **R37**: The system must reject bookings in which the time chosen is one not available in the store
- **R38**: The system must reject bookings if their times for the same customer intersect each other
- **D10**: The internet connection works properly without failure

G6: The system must alert the customers when they have to leave to arrive in time at the store

- **R39**: The system must calculate the distance between the customer and the store
- **R40**: The system must acquire the customer position
- **R41**: The system must send a notification if the waiting time of the queue is less than 5 minutes plus the arrival time to the store from their position
- **D6**: The customer will always reach the store with a car
- **D7**: The devices that acquire users' position provide location with an error of 20 meters at most
- **D8**: The system knows the position of every registered store
- **D9**: The customer never turns off GPS
- **D10**: The internet connection works properly without failure

Traceability Matrix

Requirement	Use Case
R1	Book a Visit (#5) Line Up (#4)
R2	Book a Visit (#5) Line Up (#4)
R3	Book a Visit (#5) Line Up (#4)
R4	Line Up (#4)
R5	Line Up (#4)
R6	Line Up (#4)
R7	Select Duration of Visit (#8)
R8	Select Items of Visit (#9)
R9	Book a Visit (#5) Line Up (#4)
R10	Suggest other Store (#6)
R11	Retire from the Queue (#10)
R12	Book a Visit (#5) Line Up (#4)
R13	Line Up (#4)
R14	Line Up (#4)
R15	Line Up (#4)
R16	Book a Visit (#5) Line Up (#4)
R17	Set Store Limit (#13)
R18	Set Store Limit (#13)
R19	Set Store Limit (#13)
R20	Set Store Limit (#13)
R21	Set Store Limit (#13)
R22	Set Store Limit (#13)
R23	Notify Turn
R24	Notify Turn
R25	Book a Visit (#5) Line Up (#4)
R26	Book a Visit (#5)

	Line Up (#4)
R27	Notify Turn
R28	Book a Visit (#5) Line Up (#4)
R29	Line Up (#4)
R30	Line Up (#4)
R31	Book a Visit (#5)
R32	Book a Visit (#5)
R33	Book a Visit (#5)
R34	Book a Visit (#5)
R35	Suggest other Slot (#7)
R36	Select Favourite Store (#3)
R37	Book a Visit (#5)
R38	Book a Visit (#5)
R39	Estimate Arrival Time
R40	Estimate Arrival Time
R41	Estimate Arrival Time Notify Turn

3.3 – Performance Requirements

The system has to be able to serve a great number of users (100.000) simultaneously. It has to guarantee quick (<1 second), reactive and correct responses.

3.4 – Design Constraints

3.4.1. Regulatory policy

The system will have to ask for users' data for the registration and authentication.

Moreover, it will have to ask for users' permission in order to retrieve and use their positions without (at least in a first implementation) storing them. It will respect the GDPR policy.

There will be also collected data from the visit the customer do, such as average time of that. They will be used to improve the queue management.

3.4.2. Hardware limitations

• Mobile App

- iOS or Android smartphone (iOS 9.0 or Android 6.0)

- 2G/3G/4G/5G connection

- GPS

The store needs a wi-fi printer that has to be connected with the system and a smartphone in which the application is installed and the camera has no damage.

3.4.3 Interfaces to other applications

In the first release no public interfaces will be opened and third-party services won't be able to interoperate with CLup.

3.5 – Software System Attributes

3.5.1 – Reliability

The system must run continuously without interruptions, so it must have a frequency of failures of 0.01%. To make the system more fault tolerant redundancy might be applied so the central database and the running processes might be duplicated; there will also be done reliability testing for this purpose.

3.5.2 – Availability

The System must be available 99.8% of the time, this can be obtained by also implementing the redundancy and fault tolerance mentioned before in the Reliability section.

3.5.3 – Security

Users password and GPS location, which are sensitive information, must be protected by encryption when they are sent through the network; also the central database containing the data reside must be protect (through techniques of Database Security) to avoid external and internal attack and also to handle malfunctions of the hardware.

3.5.4 – Maintainability

The development of the application must be done in a modular way to make easier to fix and modify it in the future and also to make the system capable to facilitate addition of new features and options to meet new possible requirements in the future; for that there might be used Design patterns with standard terminology. To improve the maintainability the code will be provided with Documentation.

3.5.5 – Compatibility

This is a mobile application, so it needs to be compatible with Android and IOS systems to work in the 99% of the smart phones.

3.6 – Minimum Viable Product

In this section we will analyse which are the requirements with more priority, those requirements that will constitute the core of the application. To achieve this many features are not present: the possibility to book a visit, the possibility to select duration and items of the visit, the estimate of the waiting time, the advice to leave to reach the store in time and the suggestion of other store and slot as well. The features that will constitute the MVP are: the line-up request management, the store manager influx management and the customer entrance and leaving management.

Feature	Requirements
Line-Up Request Management	R1, R2, R3, R5, R6, R9, R11
Store Manager Influx Management	R12, R13, R14, R15, R17, R18, R19
Entrance and Leaving Management	R22, R23, R25, R26, R27

4 – Formal Analysis Using Alloy

In this section an analysis of some critical aspects of the system is provided exploiting Alloy. The focus is on some static constraints:

- It cannot happen that two different customers have the same e-mail
- Every store has at least one department and is bigger than every of them
- Every reservation is unique and addressed to a single store
- Every reservation is *accepted* only if the store is available, *removed* otherwise

Then, some other necessary structural constraints are expressed and commented in the Alloy formal notation. These constraints are verified analysing the worlds generated running the model and checking some relevant assertion and predicates.

4.1 – Alloy Code

```
open util/integer

sig Time{
   time: one Int
}{
   time >= 0
}

sig Position{
   latitude: one Int,
   longitude: one Int,
}

sig DataTime{
   day: one Int,
   month: one Int,
   time: one Int
}{
   day > 0
   month > 0
   time > 0
}

sig Department{
   name: one Int,
   maximum_capability: one Int
}{
   maximum_capability > 0
}

sig Store{
   position: one Position,
   name: one Int,
   departments: set Department,
   actual_data: one DataTime,
   time_start: one Time,
   time_closure: one Time,
   average_time_of_visit: one Time,
   tolerance_of_delay: one Time,
}

//It doesn't contain any other attributes (differently by the UML) because only these underwritten are useful for testing all requirements (except showing data)
sig Customer_Profile{
   average_visit_time: one Time,
   delta_delay: one Int,
   preferred_store: one Store
}{
```

```
  delta_delay >= 0
}

abstract sig User{
  username: one Int,
  password: one Int
}

abstract sig Customer_status{}
sig InTheQueue extends Customer_status{}
sig Entered extends Customer_status{}
sig Inactive extends Customer_status{}

sig Customer extends User{
  position: one Position,
  profile: one Customer_Profile,
  cus_status: one Customer_status
}

sig Handout_Clerk extends User{
  associated_store: one Store
}

sig Scan_Clerk extends User{
  associated_store: one Store
}

sig Itinerary{
  store_chosen: one Store,
  departments_list: set Department,
  duration_estimated: one Int
}{
  duration_estimated >= 0
}

abstract sig Reservation_status{}
sig ToBeEvaluated extends Reservation_status{}
sig Pending extends Reservation_status{}
sig Accepted extends Reservation_status{}
sig Rejected extends Reservation_status{}
sig Removed extends Reservation_status{}

abstract sig Reservation{
  customer: one Customer,
  data_reservation: one DataTime,
  code: one Int,
  itinerary: one Itinerary,
  res_status: one Reservation_status,
  time_to_arrive: one Time //time for reach the store
}

sig LineUp_Request extends Reservation{}

sig Booking_Request extends Reservation{
  data_booking: one DataTime
}

sig Monitor{
  reservation_turn: one Int
}

sig Reservation_Manager{
  lineUp_list: set LineUp_Request,
  booking_list: set Booking_Request,
```

```
  res_inside: set Reservation,
  store: one Store,
  monitor: one Monitor,
  maximum_waiting_time: one Time
}

abstract sig Suggestion{
  available_stores_list: set Store,
  available_times_list: set Int,
}

sig Suggestion_LineUp extends Suggestion{
  reservation_rejected: one LineUp_Request
}

sig Suggestion_Booking extends Suggestion{
  reservation_rejected: one Booking_Request
}

fact timeClosureAfterStart{
  all s:Store | s.time_closure.time > s.time_start.time
}


//constraints on time

//Every store has a data
fact allStoreHasADataTime{
  all s: Store | one d: DataTime | s.actual_data = d
}

//Every store has the same data (supposing that the stores are in the same country with no time lag)
fact allStoreHasTheSameData{
  all s1,s2: Store | s1.actual_data = s2.actual_data
}

//Every line-up request could be done the same day in which the store is, it can't be done neither the previous day
fact allLineUpDoneSameDay{
  all lr: LineUp_Request | all s: Store | lr.data_reservation.month = s.actual_data.month and
                    lr.data_reservation.day =s.actual_data.day and lr.data_reservation.time <= s.actual_data.time
}

//every CustomerProfile has a Customer that addresses to it
fact noFloatingProfiles{
  all cp: Customer_Profile | one c:Customer | c.profile=cp
}

//Every booking request must be done, obviously, in the past, even days before.
fact allBookingDoneInThePast{
  all b: Booking_Request | all s:Store | (b.data_reservation.month < s.actual_data.month and
                                                b.data_reservation.day - s.actual_data.day >= 25)
                  or (b.data_reservation.month = s.actual_data.month and b.data_reservation.day < s.actual_data.day)
                  or (b.data_reservation.month = s.actual_data.month and b.data_reservation.day = s.actual_data.day and
                        b.data_reservation.time < s.actual_data.time)
}


//Every booking request is for the present or the future days
fact allBookingForTheFuture{
  all b: Booking_Request | no s:Store | b.data_booking.month < s.actual_data.month
                    or (b.data_booking.month = s.actual_data.month and b.data_booking.day < s.actual_data.day)
                    or (b.data_booking.month = s.actual_data.month and b.data_booking.day = s.actual_data.day and
                    b.data_booking.time < s.actual_data.time)
}
```

```
//constraints on Department

//Departments in the same store can't have the same name
fact noSameDepartmentName{
  all disj d1,d2: Department, s:Store | (d1 in s.departments and d2 in s.departments) implies d1.name!=d2.name
}

fact departmentIsOnlyInOneStore{
  all disj s1, s2: Store | no d: Department | d in s1.departments and d in s2.departments
}

//constraints on Store

// Every store has a position
fact everyStoreHasAPosition{
  all s: Store | one p: Position | s.position = p
}

//Every store has a different position or name
fact everyStoreHasADifferentPositionOrName{
  all disj s1,s2: Store | s1.position != s2.position or s1.name != s2.name
}

//Every store has at least one department
fact everyStoreHasAtLeastOneDepartment{
  all s:Store | #s.departments > 0
}

//constraint on User

//Two users can't have the same username
fact noSameUsername{
  all disj u1, u2: User | u1.username != u2.username
}

//constraint on Customer

//No Customer Profile is shared
fact noSameCustomerProfile{
  all disj c1,c2: Customer | no cp: Customer_Profile | c1.profile = cp and c2.profile = cp
}

//no constraint on Handout_Clerk
//no constraint on Scan_Clerk

//constraint on Itinerary
fact departmentsBelongToAStore{
  all i: Itinerary | i.departments_list in i.store_chosen.departments
}

//constraints on Reservation, LineUp_Request and Booking_Request

fact noObiquity1{
  no disj l1,l2: LineUp_Request | l1.res_status = Accepted and l2.res_status = Accepted and l1.customer = l2.customer
}

fact noObiquity2{
  no disj b1,b2: Booking_Request | b1.res_status = Accepted and b2.res_status = Accepted and b1.customer = b2.customer
}

//no constraint on Monitor

//constraints on Reservation_Manager
```

```
//If a reservation is in reservation manager lists, it must have the same store associated to
fact allReservationInManagerIfSameStore1{
   all rm: Reservation_Manager | all l: LineUp_Request | l in rm.lineUp_list implies l.itinerary.store_chosen = rm.store
}

//If a reservation is in reservation manager lists, it must have the same store associated to
fact allReservationInManagerIfSameStore2{
   all rm: Reservation_Manager | all b: Booking_Request | b in rm.booking_list implies b.itinerary.store_chosen = rm.store
}

fact allReservationInsideTheStoreChosen{
   all rm: Reservation_Manager | all r: Reservation | r in rm.res_inside implies r.itinerary.store_chosen = rm.store
}

fact everyManagerHasDifferentMonitors{
   no disj r1,r2: Reservation_Manager | r1.monitor = r2.monitor
}

fact noDoubleLineUp{
   all r: Reservation_Manager | no disj l1,l2: LineUp_Request | l1.customer = l2.customer and l1 in r.lineUp_list and
                                                                l2 in r.lineUp_list
}

//Two reservation manager can't have the same store associated
fact noSameResManager{
   all disj r1,r2: Reservation_Manager | r1.store != r2.store
}

//Every store is covered by a reservation manager
fact everyStoreHasAReservationManager{
   all s: Store | one r: Reservation_Manager | r.store = s
}

fact onlyInOneList1{
   all rm: Reservation_Manager, l: rm.lineUp_list | l not in rm.res_inside
}

fact onlyInOneList2{
   all rm: Reservation_Manager, b: rm.booking_list | b not in rm.res_inside
}

fact onlyInOneList3{
   no rm: Reservation_Manager, r: rm.res_inside | r in rm.booking_list or r in rm.lineUp_list
}

fact everyResCantBeInMultipleManager1{
   all lr: LineUp_Request | no disj r1,r2: Reservation_Manager | (lr in r1.res_inside or lr in r1.lineUp_list) and
                              (lr in r2.res_inside or lr in r2.lineUp_list)
}

fact everyResCantBeInMultipleManager2{
   all b: Booking_Request | no disj r1,r2: Reservation_Manager | (b in r1.res_inside or b in r1.booking_list) and
                              (b in r2.res_inside or b in r2.booking_list)
}


//constraints on Suggestion

//Every lineUp suggestion can't suggest the same store chosen before
fact storeSuggestedNotEqualToChosen1{
   all s:Suggestion_LineUp | not s.reservation_rejected.itinerary.store_chosen in s.available_stores_list
}
```

```
//Every booking suggestion can't suggest the same store chosen before
fact storeSuggestedNotEqualToChosen2{
   all s:Suggestion_Booking | not s.reservation_rejected.itinerary.store_chosen in s.available_stores_list
}

//Every booking suggestion can't suggest the same time chosen before
fact timeSuggestedNotEqualChosen{
   all s: Suggestion_Booking | not s.reservation_rejected.data_booking.time in s.available_times_list
}

//LineUp suggestions have no times
fact noneTimesInLineUpSuggestion{
   all s: Suggestion_LineUp | s.available_times_list = none
}

//No same suggesttion
fact noSameSuggestion{
   no disj s1,s2: Suggestion_LineUp | s1.reservation_rejected = s2.reservation_rejected
}

//No same suggestion
fact noSameSuggestion2{
   no disj s1,s2: Suggestion_Booking | s1.reservation_rejected = s2.reservation_rejected
}


//constraints on Reservation Status

//Define every status for LineUp Request
fact defineLineUpStatus{
   all lr: LineUp_Request | ((lr.res_status = ToBeEvaluated or lr.res_status = Rejected or lr.res_status = Removed) iff
                              (no r: Reservation_Manager |  lr in r.res_inside or lr in r.lineUp_list)) and
                        (lr.res_status = Pending iff (one r: Reservation_Manager | r.store = lr.itinerary.store_chosen and
                              not lr in r.res_inside and lr in r.lineUp_list)) and
                        (lr.res_status = Accepted iff (one r: Reservation_Manager | r.store = lr.itinerary.store_chosen and
                              lr in r.res_inside and not lr in r.lineUp_list))
}

//Define every status for Booking Request
fact defineBookingStatus{
   all b: Booking_Request | ((b.res_status = ToBeEvaluated or b.res_status = Rejected or b.res_status = Removed) iff
                              (no r: Reservation_Manager |  b in r.res_inside or b in r.booking_list)) and
                        (b.res_status = Pending iff (one r: Reservation_Manager | r.store = b.itinerary.store_chosen and
                              not b in r.res_inside and b in r.booking_list)) and
                        (b.res_status = Accepted iff (one r: Reservation_Manager | r.store = b.itinerary.store_chosen and
                              b in r.res_inside and not b in r.booking_list))
}

//constraint on customer status
--Define every customer status
fact defineCustomerStatus{
   all c: Customer | ((c.cus_status = InTheQueue iff (some r:Reservation | r.res_status = Pending and r.customer = c)) and
            (c.cus_status = Entered iff (some r: Reservation | r.res_status = Accepted and r.customer = c))) and
            (c.cus_status = Inactive iff (no r: Reservation | r.customer = c and (r.res_status = Pending or
                  r.res_status = Accepted)))
}

//other constraints

//Only rejected reservations have a suggestion
fact suggestionForDeterminedStatus1{
   all lr: LineUp_Request | one s: Suggestion_LineUp | lr.res_status = Rejected iff s.reservation_rejected = lr
}
```

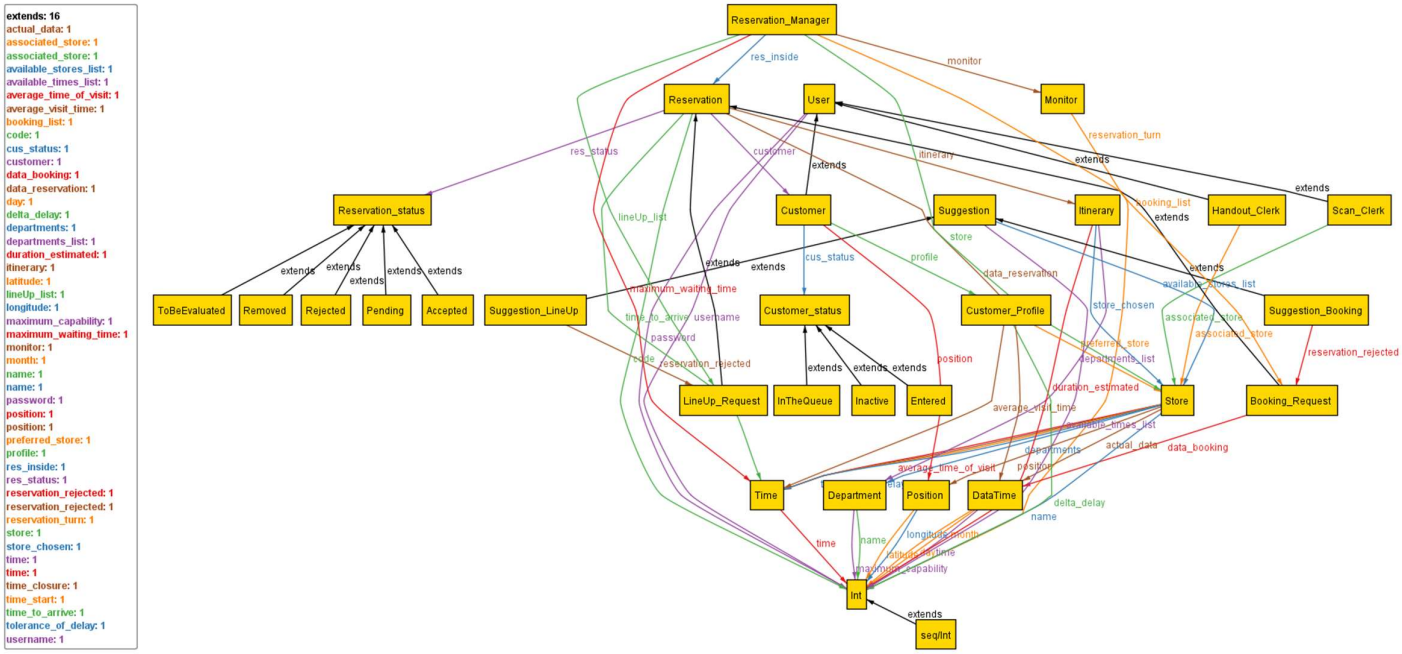
```
//Only rejected reservations have a suggestion
fact suggestionForDeterminedStatus2{
   all b: Booking_Request | one s: Suggestion_Booking | b.res_status = Rejected iff s.reservation_rejected = b
}

fact noFloatingDepartments{
   all d:Department | one s:Store | d in s.departments
}

fact noBeforeActualSuggestion{
   all s: Suggestion_Booking, t: Time | one st:Store | s.reservation_rejected.itinerary.store_chosen = st and
            t.time > st.actual_data.time and t.time in s.available_times_list
}

fact noSameBooking{
   all rm:Reservation_Manager | no disj b1,b2:Booking_Request | b1 in rm.booking_list and b2 in rm.booking_list and
                                    b1.code = b2.code
}

//The queue of line-up is not empty
pred notEmptyQueue[rm: Reservation_Manager, t_c: Int, t_estimated: Int]{
   #rm.lineUp_list> 0 or
   (some ti: DataTime | (ti in rm.booking_list.data_booking) and ti.time>= t_c and (ti.time < t_c +  t_estimated))
}

//Customer  insert time estimation (it's optional)
pred timeEstimationInserted[i: Itinerary]{
   some ti: DataTime.time | ti in i.duration_estimated
}

//Customer is a new users, so he/she his/hers average visit time and delta delay = 0
pred newUser[c: Customer]{
   c.profile.average_visit_time.time = 0 and c.profile.delta_delay = 0
}

//Customer isn't a new users, so he/she his/hers average visit time and delta delay >= 0
pred noNewUser[c: Customer]{
   c.profile.average_visit_time.time > 0 and c.profile.delta_delay >= 0
}

//The queue is empty and the customer is a new user, so every data comes from store data and customer position
pred storeAvailability1[rm: Reservation_Manager, r: Reservation]{
   newUser[r.customer]
   (r in rm.booking_list or r in rm.lineUp_list)
   not timeEstimationInserted[r.itinerary]
   not notEmptyQueue[rm, rm.store.actual_data.time.add[r.time_to_arrive.time], rm.store.average_time_of_visit.time ]
   (rm.store.time_start.time <=
rm.store.actual_data.time.add[rm.store.average_time_of_visit.time].add[r.time_to_arrive.time] and
            rm.store.actual_data.time.add[rm.store.average_time_of_visit.time].add[r.time_to_arrive.time] <
            rm.store.time_closure.time)
}

//Not a new user, empty queue, not inserted estimation
pred storeAvailability2[rm: Reservation_Manager, r: Reservation]{
   noNewUser[r.customer]
   (r in rm.booking_list or r in rm.lineUp_list)
   not timeEstimationInserted[r.itinerary]
   not notEmptyQueue[rm, rm.store.actual_data.time.add[r.time_to_arrive.time], rm.store.average_time_of_visit.time ]
   (rm.store.time_start.time <=
rm.store.actual_data.time.add[r.customer.profile.average_visit_time.time].add[r.time_to_arrive.time] and
            rm.store.actual_data.time.add[r.customer.profile.average_visit_time.time].add[r.time_to_arrive.time] <
            rm.store.time_closure.time)
}
```

```
//Time inserted, empty queue
pred storeAvailability3[rm: Reservation_Manager, r: Reservation]{
   (r in rm.booking_list or r in rm.lineUp_list)
   not notEmptyQueue[rm, rm.store.actual_data.time.add[r.time_to_arrive.time], r.itinerary.duration_estimated]
   timeEstimationInserted[r.itinerary]
   (rm.store.time_start.time <= rm.store.actual_data.time.add[r.itinerary.duration_estimated].add[ r.time_to_arrive.time]
and rm.store.actual_data.time.add[r.itinerary.duration_estimated].add[r.time_to_arrive.time] < rm.store.time_closure.time)
}

pred arriveAfterEndQueue[rm: Reservation_Manager, r: Reservation] {
   rm.maximum_waiting_time.time < r.time_to_arrive.time
}

//New Customer will arrive after the end of the queue
pred storeAvailability4[rm: Reservation_Manager, r: Reservation]{
   (r in rm.booking_list or r in rm.lineUp_list)
    notEmptyQueue[rm, rm.store.actual_data.time.add[r.time_to_arrive.time], rm.store.average_time_of_visit.time ]
   newUser[r.customer]
   not timeEstimationInserted[r.itinerary]
   arriveAfterEndQueue[rm, r]
   (rm.store.time_start.time <=
rm.store.actual_data.time.add[rm.store.average_time_of_visit.time].add[r.time_to_arrive.time]  and
   rm.store.actual_data.time.add[rm.store.average_time_of_visit.time].add[r.time_to_arrive.time]  <
   rm.store.time_closure.time )
}

//New Customer will arrive before the end of the queue
pred storeAvailability5[rm: Reservation_Manager,  r:Reservation]{
   (r in rm.booking_list or r in rm.lineUp_list)
   notEmptyQueue[rm, rm.store.actual_data.time.add[rm.maximum_waiting_time.time],
rm.store.average_time_of_visit.time ]
   newUser[r.customer]
   not timeEstimationInserted[r.itinerary]
   not arriveAfterEndQueue[rm, r]
   (rm.store.time_start.time <=
rm.store.actual_data.time.add[rm.store.average_time_of_visit.time].add[rm.maximum_waiting_time.time]  and
   rm.store.actual_data.time.add[rm.store.average_time_of_visit.time].add[rm.maximum_waiting_time.time]  <
   rm.store.time_closure.time)
}

//Customer will arrive after the end of the queue
pred storeAvailability6[rm: Reservation_Manager,  r:Reservation]{
   (r in rm.booking_list or r in rm.lineUp_list)
   notEmptyQueue[rm, rm.store.actual_data.time.add[r.time_to_arrive.time], r.customer.profile.average_visit_time.time]
   not newUser[r.customer]
   not timeEstimationInserted[r.itinerary]
   arriveAfterEndQueue[rm, r]
   (rm.store.time_start.time <=
rm.store.actual_data.time.add[r.customer.profile.average_visit_time.time].add[r.time_to_arrive.time]  and
   rm.store.actual_data.time.add[r.customer.profile.average_visit_time.time].add[r.time_to_arrive.time]  <
   rm.store.time_closure.time)
}

//Customer will arrive before the end of the queue
pred storeAvailability7[rm: Reservation_Manager,  r:Reservation]{
   (r in rm.booking_list or r in rm.lineUp_list)
   notEmptyQueue[rm,  rm.store.actual_data.time.add[rm.maximum_waiting_time.time],
r.customer.profile.average_visit_time.time]
   not newUser[r.customer]
   not timeEstimationInserted[r.itinerary]
   not arriveAfterEndQueue[rm, r]
   (rm.store.time_start.time <=
rm.store.actual_data.time.add[r.customer.profile.average_visit_time.time].add[rm.maximum_waiting_time.time]  and
```

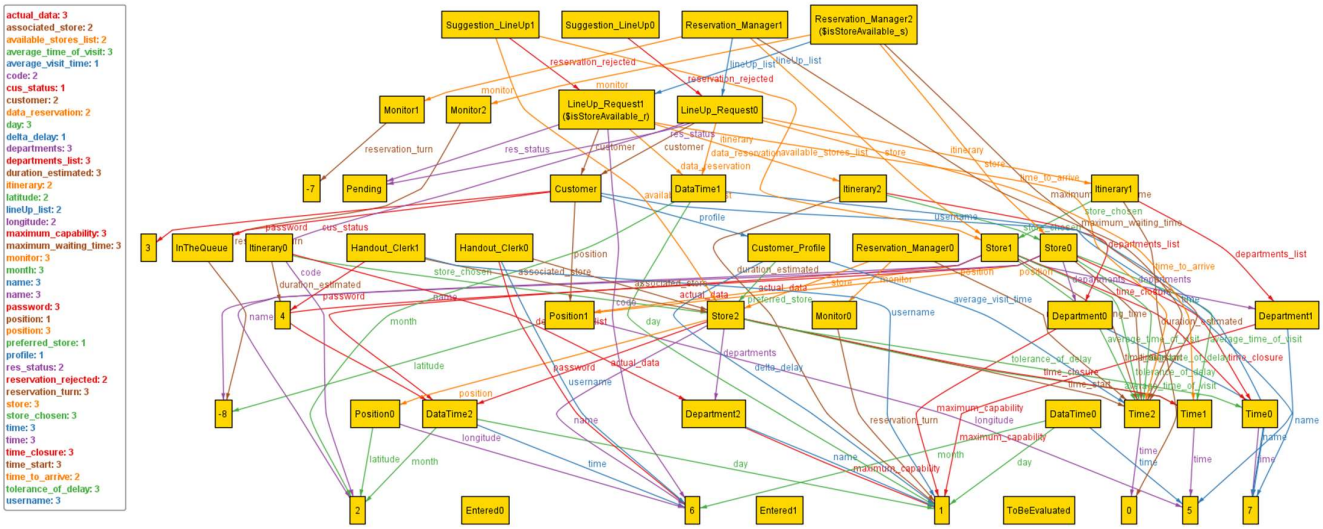
```
  rm.store.actual_data.time.add[r.customer.profile.average_visit_time.time].add[rm.maximum_waiting_time.time] <
  rm.store.time_closure.time)
}

//Customer will arrive after the end of the queue and estimate time (it doesn't matter if he/she is a new user)
pred storeAvailability8[rm: Reservation_Manager,  r:Reservation]{
  (r in rm.booking_list or r in rm.lineUp_list)
  notEmptyQueue[rm, rm.store.actual_data.time + r.time_to_arrive.time, r.itinerary.duration_estimated]
  timeEstimationInserted[r.itinerary]
  arriveAfterEndQueue[rm, r]
  (rm.store.time_start.time <=   rm.store.actual_data.time.add[r.itinerary.duration_estimated].add[r.time_to_arrive.time]
and
  rm.store.actual_data.time.add[r.itinerary.duration_estimated].add[r.time_to_arrive.time]  < rm.store.time_closure.time)
}

//Customer will arrive before the end of the queue and estimate time (it doesn't matter if he/she is a new user)
pred storeAvailability9[rm :Reservation_Manager,  r:Reservation]{
  (r in rm.booking_list or r in rm.lineUp_list)
  notEmptyQueue[rm, rm.store.actual_data.time.add[rm.maximum_waiting_time.time], r.itinerary.duration_estimated]
  timeEstimationInserted[r.itinerary]
  not arriveAfterEndQueue[rm, r]
  (rm.store.time_start.time <=
rm.store.actual_data.time.add[r.itinerary.duration_estimated].add[rm.maximum_waiting_time.time]  and
  rm.store.actual_data.time.add[r.itinerary.duration_estimated].add[rm.maximum_waiting_time.time]  <
rm.store.time_closure.time)
}

//Combination of all possible cases
pred isStoreAvailable[s :Reservation_Manager, r:Reservation]{
  storeAvailability1[s,r] or storeAvailability2[s,r] or storeAvailability3[s,r] or storeAvailability4[s,r] or storeAvailability5[s,r]
        or storeAvailability6[s,r] or storeAvailability7[s,r] or storeAvailability8[s,r] or storeAvailability9[s,r]
}

pred addRequestInTheQueue[l: LineUp_Request, rmAfter: Reservation_Manager, rmBefore: Reservation_Manager]{
  rmAfter.lineUp_list = rmBefore.lineUp_list + l
}

pred addLineUpInList[l: LineUp_Request, rm: Reservation_Manager]{
  l not in rm.lineUp_list implies (
  l.itinerary.store_chosen = rm.store and
  isStoreAvailable[rm, l] and
  addRequestInTheQueue[l, rm, rm])
}

pred removeRequestInTheQueue[l: LineUp_Request,  rmAfter: Reservation_Manager, rmBefore: Reservation_Manager]{
  rmAfter.lineUp_list  = rmBefore.lineUp_list  - l
}

pred removeLineUpInList[l: LineUp_Request, rm: Reservation_Manager]{
  l in rm.lineUp_list implies
  removeRequestInTheQueue[l, rm, rm]
}

assert addRemove{
  all rm: Reservation_Manager, lr: LineUp_Request |
        (addLineUpInList[lr,rm] implies removeLineUpInList[lr,rm]) implies lr not in rm.lineUp_list
}

pred noBookingInTheList[rm: Reservation_Manager]{
  #rm.booking_list = 0
}

pred addBookingRequestInBookingSet[b: Booking_Request, rmAfter,rmBefore: Reservation_Manager]{
  rmAfter.booking_list = rmBefore.booking_list + b
```

```
}

pred addBooking[b: Booking_Request, rm: Reservation_Manager]{
   b not in rm.booking_list implies (
   (b.itinerary.store_chosen = rm.store and isStoreAvailable[rm,b]) and
   addBookingRequestInBookingSet[b, rm, rm] )
}

pred removeBooking[b: Booking_Request,  rmAfter: Reservation_Manager, rmBefore: Reservation_Manager]{
   rmAfter.booking_list  = rmBefore.booking_list  - b
}

pred removeBook[b:Booking_Request, rm:Reservation_Manager]{
   b in rm.booking_list implies removeBooking[b,rm,rm]
}

assert addRemoveBooking{
   all rm: Reservation_Manager, b:Booking_Request | (addBooking[b,rm]  implies removeBook[b,rm]) implies
            b not in rm.booking_list
}
```

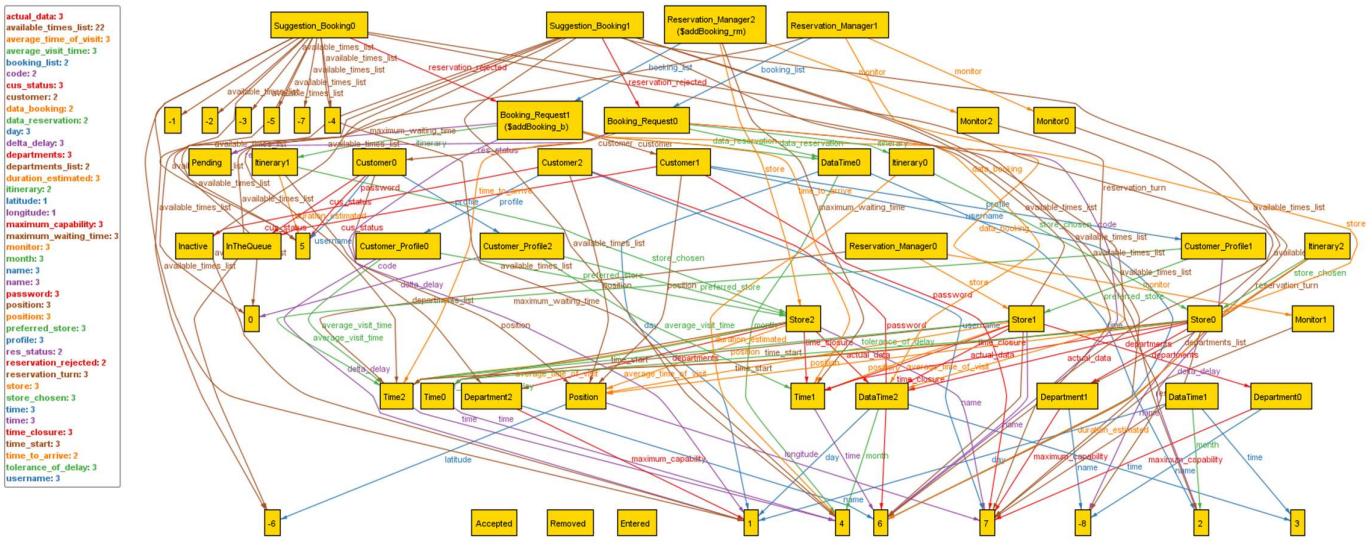
4.2 – Metamodel



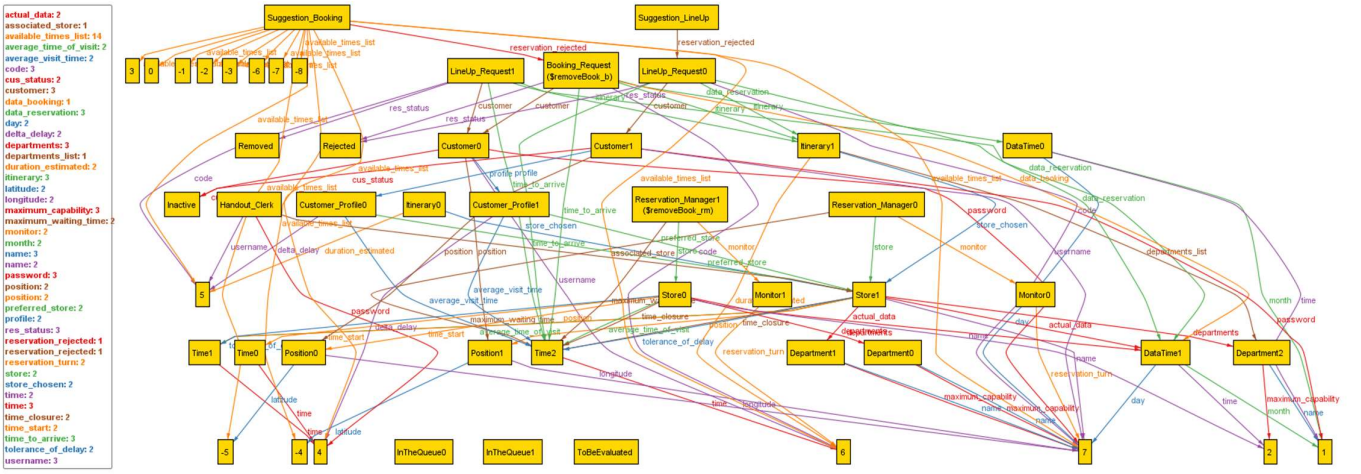
Metamodel



Metamodel – isStoreAvailable



Metamodel – addBooking



Metamodel – removeBooking

4.3 – Result of Assertion

```
Executing "Check addRemove"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  27286 vars. 1125 primary vars. 72870 clauses. 112ms.
  No counterexample found. Assertion may be valid. 7ms.
```

```
Executing "Check addRemoveBooking"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  27286 vars. 1125 primary vars. 72870 clauses. 159ms.
  No counterexample found. Assertion may be valid. 9ms.
```

4.4 – Result of Predicates

```
Executing "Run isStoreAvailable"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  27208 vars. 1125 primary vars. 72295 clauses. 98ms.
  Instance found. Predicate is consistent. 278ms.
```

```
Executing "Run addBooking"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  27239 vars. 1125 primary vars. 72365 clauses. 81ms.
  Instance found. Predicate is consistent. 242ms.
```

```
Executing "Run removeBook"
  Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
  22667 vars. 1125 primary vars. 56855 clauses. 67ms.
  Instance found. Predicate is consistent. 119ms.
```

5 – Effort Spent

Simone Bianchera

Chapter	Hours
1 – Introduction	3h
2 – Overall Description	12h
3 – Specific Requirements	6h
4 – Formal Analysis Using Alloy	30h

Niccolò Borrelli

Chapter	Hours
1 – Introduction	4h
2 – Overall Description	7h
3 – Specific Requirements	21h
4 – Formal Analysis Using Alloy	15h

Rei Barjami

Chapter	Hours
1 – Introduction	3h
2 – Overall Description	12h
3 – Specific Requirements	10h
4 – Formal Analysis Using Alloy	21h