CLup project by Neroni, Pozzi, Vetere

POLITECNICO
MILANO 1863

Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Cristiano Neroni, Davide Pozzi, Maurizio Vetere
Version:	1.0
Date:	Late Fall 2020
Download page:	https://github.com/pollo-fritto/PozziNeroniVetere.git
Copyright:	Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

This document has been prepared to help you approaching Latex as a formatting tool for your Travlendar+ deliverables. This document suggests you a possible style and format for your deliverables and contains information about basic formatting commands in Latex. A good guide to Latex is available here https://tobi.oetiker.ch/lshort/lshort.pdf, but you can find many other good references on the web.

Writing in Latex means writing textual files having a `.tex` extension and exploiting the Latex markup commands for formatting purposes. Your files then need to be compiled using the Latex compiler. Similarly to programming languages, you can find many editors that help you writing and compiling your latex code. Here https://beebom.com/best-latex-editors/ you have a short oviewview of some of them. Feel free to choose the one you like.

Include a subsection for each of the following items[1]:

- Purpose: here we include the goals of the project
- Scope: here we include an analysis of the world and of the shared phenomena
- Definitions, Acronyms, Abbreviations
- Revision history
- Reference Documents
- Document Structure

Below you see how to define the header for a subsection.

1.1 Purpose

This document has the purpose to guide the developer in the realization of the software called Clup, an application that aims to manage queues digitally.
Due to the Coronavirus emergency grocery shopping needs to follow strict rules: supermarkets need to restrict access to their stores which typically results in long lines forming outside.The goal of this project is to develop an easy-to-use application that allows store managers to regulate the influx of people and that saves people from having to crowd outside of stores.
The application releases a number that gives the position in the queue and gives information about the time when that number is called, in this way the user is able to arrive to the supermarket and enter immediately.
Clup allows also the user to book a slot to enter the supermarket indicating the expected time to shop, or alternatively the application itself can infer it.
Finally the application can suggest different slots to visit the store, based on the influx of people, and slots in alternative stores, based on the day/hour preferences of the user.

[1] By the way, what follows is the structure of an itemized list in Latex.

1.1.1 Goals

G1	Anybody is guaranteed possibility to make shopping at any supermarket in reasonable time (def. reasonable)
G2	Users can get to know the least crowded time slots
G3	Fair users can make a reservation to enter in a supermarket
G4	Stores can easily monitor fluxes
G5	Only authorized users can access
G6	Crowds are dramatically reduced outside supermarket stores
G7	CLup should not decrease customer affluence beyond a reasonable level w.r.t. to normal (→ define reasonable)
G8	Same shopping capabilities guaranteed to offline users
G9	Find the best (less crowded, soonest available) alternative among local supermarket stores (of same franchise only?)
G10	Supermarkets do not overcrowd

1.2 Scope

1.2.1 World Phenomena

WP1	User leaves home to go to the supermarket
WP2	Users crowd outside the store
WP3	User arrives at the supermarket
WP4	User enters the supermarket
WP5	User does the grocery shopping
WP6	User exits the supermarket
WP7	Supermarkets restrict accesses in stores
WP8	User buys products of a non booked category

1.2.2 Shared Phenomena

SP1	User lines up using the application
SP2	User makes a reservation
SP3	User keeps track of how line evolves
SP4	User validates the entrance with a QR code
SP5	User receives suggestion for less crowded time slots
SP6	User receives suggestion for less crowded stores
SP7	CLup assigns a time slot
SP8	CLup signals max number of customers inside the store has been reached
SP9	CLup signals customer for improper behavior
SP10	Offline customer interacts with physical totem
SP11	User confirms booking
SP12	User confirms ticket reservation

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

Check in procedure:
Rush hours:
QR ticket:
Reserve entrance:

Malicious user:
Quick ticket

1.3.2 Acronyms

EWT: expected waiting time
ASAP: as soon as possible

1.3.3 Abbreviations

1.4 Revision History

- **v1.0**: First version of the document

1.5 Reference Documents

1.6 Document Structure

COPY PASTED FROM RASD TO ANALYSE

- Chapter 1 gives an introduction about the purpose of the document and the development of the application, with its corresponding specifications such as the definitions, acronyms, abbreviation, revision history of the document and the references. Besides, are specified the main goals, world and shared phenomena of the software.
- Chapter 2 contains the overall description of the project. In the product perspective are included the statecharts of the major function of the application and the model description through a Class diagram. In user characteristic are explained the types of actors that can use the application. Moreover, the product function clarified the functionalities of the application. Finally, are included the domain assumption that can be deducted from the assignment.
- Chapter 3 presents the interface requirement including: user, hardware, software and communication interfaces. This section contains the core of the document, the specification of functional and non-functional requirements. Functional requirements are submitted with a list of use cases with their corresponding sequence diagrams and some scenarios useful to identify specific cases in which the application can be utilised. Non-functional requirements included: performance, design and the software systems 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 documents

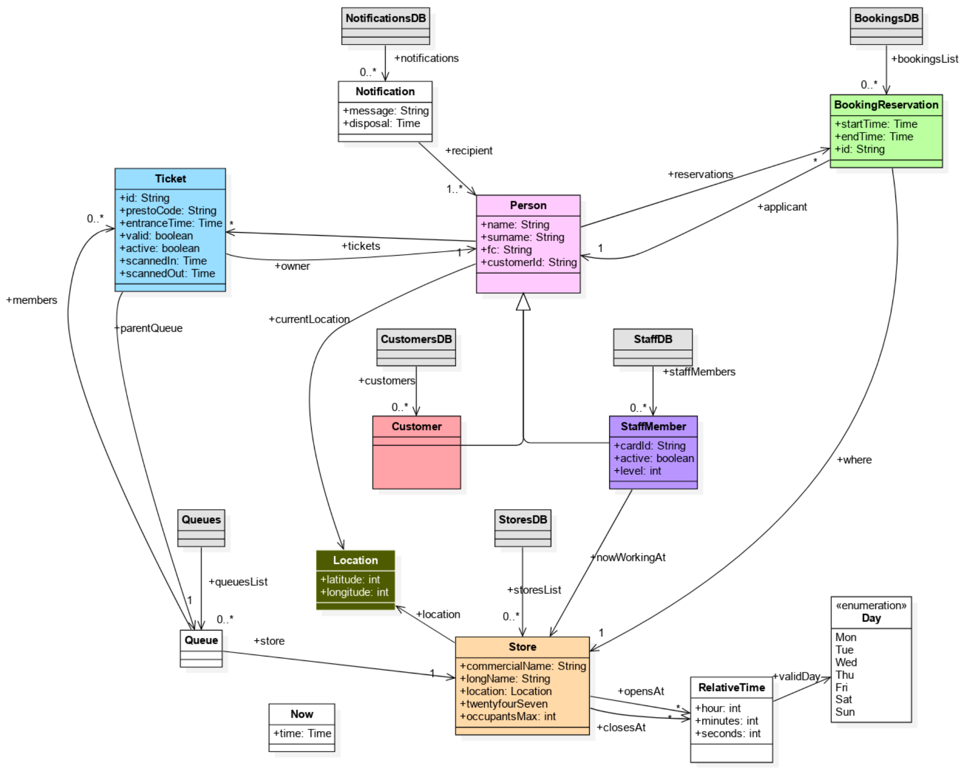

Figure 1: UML: class diagram.

2 Overall Description

Here you can see how to include an image in your document.

Here is the command to refer to another element (section, figure, table, ...) in the document: *As discussed in Section 1.6 and as shown in Figure ??, ...*. Here is how to introduce a bibliographic citation [1]. Bibliographic references should be included in a `.bib` file.

Table generation is a bit complicated in Latex. You will soon become proficient, but to start you can rely on tools or external services. See for instance this https://www.tablesgenerator.com.

2.1 Product Perspective

CLup is an application aiming to decrease the hazard of contracting COVID-19 (or other contagious diseases) when going shopping to a supermarket. The components are two: the main one targets customers of supermarkets while the second one is available for store managers. Regarding the first component, customers will be required to register to the service the first time they use it by inserting their full name, email address, ID card, phone number and a password. The customer will also be requested to specify his physical address, or to enable the GPS, in order to allow CLup to find stores nearby; this last information can be changed anytime the user needs. If the customer is not willing to register or share his address, the service will not be available. Once the setup is done the customer is now able to access the homepage of the application, here he can tap on the button “Virtually queue” that allows him to see a list of stores inside a specified range from his location: for every store it is also specified the distance in kilometers from the user position, the number of people inside the store and its maximum occupancy, if the store is full it is instead displayed the number of people in line and the EWT. It is also possible to visualize stores on a map and, by tapping on one of them, to see the same information displayed in the list. Now, if the user chooses to reserve a spot in the line,

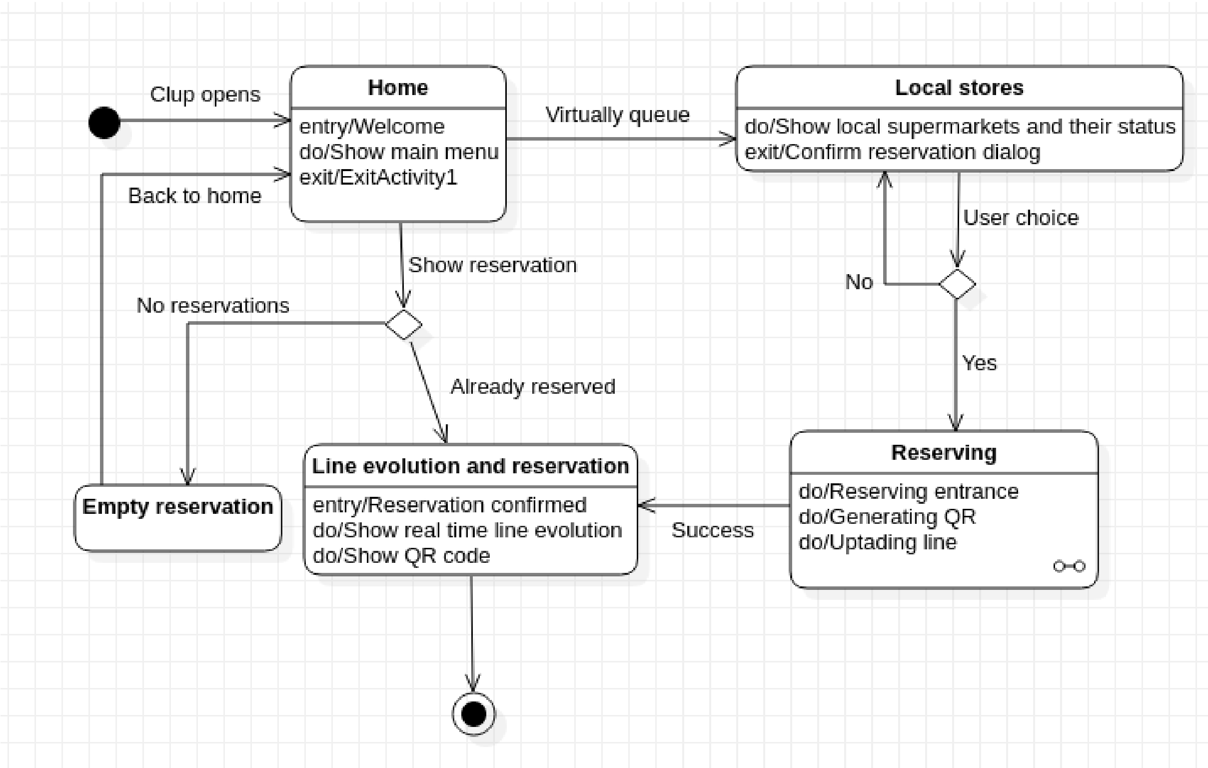

Figure 2: Statechart diagram: Virtually queue.

the application will open a confirm dialog specifying EWT and the expiration of the ticket. If the user refuses nothing happens, if he accepts instead CLup will process the request, show his ticket and the real time evolution of the line; the ticket is also visible from a home button. The process is shown in Figure 2.
The distance range in which CLup will look for supermarkets is specified by the user through the filter button in the homepage, this button will in fact open the filter screen in which, among other parameters, a sliding bar controls the distance and a dropdown list allows the user to filter the chains of supermarkets.

Another important feature is the possibility to book an entrance later in the day or in another day. The user can specify from the filters whether he prefers to choose the day or the store first and he can set the time range in which he wants to book. There is a dedicated button in the app's main screen that redirects the user to either the list/map of supermarkets or the calendar, and once the user chooses he will be respectively shown the calendar or the list/map, this time with colours to indicate the average crowdedness of stores/days given the set time range. When the user chooses the day and supermarket combination, a timetable spanning the chosen time range is shown, divided in 15 minutes time slots each one having again a colour to indicate the crowdedness. The user will be able to check his reservation on the home page and near the entrance time he will be provided an actual ticket. The process is shown in Figure 3.

The access at the supermarket is restricted by turnstiles with QR code readers, a staff member is expected to verify that nobody waits his turn in front of the entrance, jumps the turnstile or does anything irresponsible.
Customers which, for any reason, don't use the app will still be able to queue in CLup supermarkets by obtaining a printed ticket from a physical totem located near such stores; the

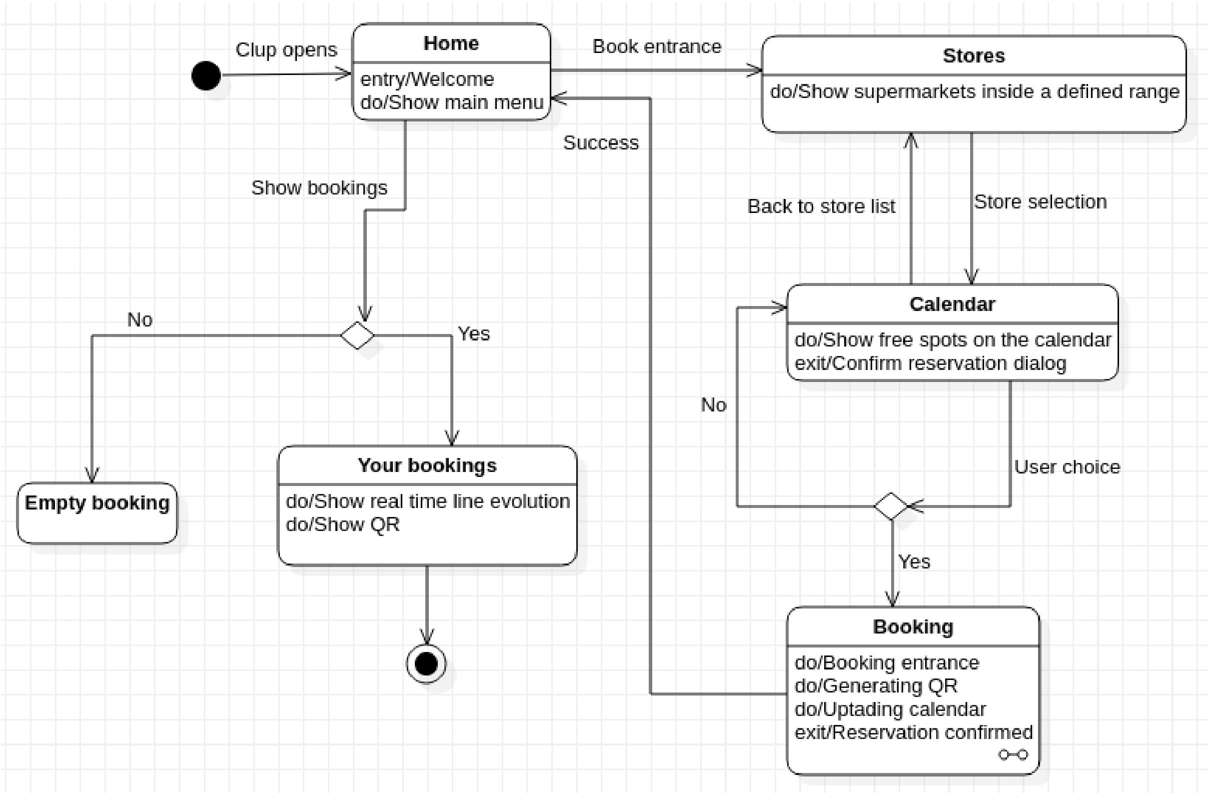

Figure 3: Statechart diagram: Book Entrance.

functioning of the application will be similar to the "Quick ticket" app function with the difference that the user can only obtain a ticket for the totem's store.
The tickets consist of a QR code and an easy to remember alphanumeric code alternative to enter the store. There will also be monitors that show the numbers allowed to enter and, eventually, delays.
The second component of CLup targets store managers: when the store decides to join the CLup network, the credentials to access the web app will be given. In this way the application will have a dedicated section to display flux data of the store selected (Che dati nello specifico? Interazione con i dati???Visualizzazione dati e possibilità di sospendere il servizio)

2.1.1 Scenarios

Scenario 1 Single user of the CLup platform, Bob, decides it's time to go shopping. Bob lives in Milan and this means he's currently in reach of **5 different supermarkets** belonging to the CLup network.
Bob then opens the app, checks the status of the current queue and notices the nearest supermarket has free room, 13 entrances left out of 55 total. It's fine for Bob, he starts walking towards it.

As soon as he approaches the supermarket (Bob's on foot), he checks the app and start the **check-in procedure**. It's not rush hours and 8 entrance are still left, so everything goes ok and Bob gets a **QR ticket**. He approaches the entrance, has his code **scanned by an automatic turnstile** and gets inside the supermarket.
In 36' time, Bob completes his shopping. He proceeds towards the exit, where another turnstile **scans his QR code once again to confirm exit.** He's now free to get home.

Scenario 2 Clara, mother of three children, now needs to go shopping. She's just downloaded CLup and has not figured out how to use it yet.

Clara decides to have a try right now, on the fly, and opens the app to check for local available supermarkets.
Unfortunately it's now **rush hours**, hence 2 of the 3 local supermarket show no currently available entrances and an **e.w.t. of 35 minutes**. Young mother decides to click on "Reserve entrance" and notices she has **15 minutes left to enter the store**. This is done in order to minimize false reservations impact on the service's availability.

Clara has to travel a 4 Km distance in her home town which seems reasonable, but since it's rush hours, **actually requires 25 minutes of time** to be travelled by car: her QR code has expired.
Fortunately, she checks CLup and can now see new free accesses in the other 2 CLup powered supermarkets, the nearest of which is only a kilometer away. She then reserves an access, reaches the supermarket in 10' time and is now free to do her shopping.

Scenario 3 James is a young unemployed man, living in the west, outer side of Rome. His **not particularly wealthy** condition does not overcome his strong medical conceptions, so that he's **particularly committed in avoiding queues** and other possible ways of contracting Covid-19 in general.

His fridge is starting to starve, so James - who still relies on a well aged Nokia 3310 for its calls and messaging - decides to go shopping. Despite being «less tech-ready» than average, James has nonetheless heard about a new app (a new way) of shopping and decides to give CLup powered supermarkets a try. Those with the lit CLup mark outside.
The nearest of the two eligible supermarkets in James' reach is 900m away and he's on foot. **Owning no smartphone**, James considers a reasonably not crowded time to go, 3 p.m., and walks towards the store.

Unfortunately James' guessing is wrong and the supermarket is **full**: a **big screen notifies no entrance is allowed for now**, and everybody has to stay clear of the entrance. He knows no alternative store as he owns no smartphone, but notices the big screen at the entrance has advices for him: next to the entrance, there's a **self service area** - enclosed by barriers and accessed by automated turnstiles - where James can have a ticket printed. **Only one person at a time** is allowed in, so that James and anybody else has nothing to worry about.
Right after printing its QR, James can notice the big screen now shows information about it, giving him (better, its ticket number, which reads AX625RQ) advice to **come back in 20 minutes** for entrance. He then goes for a walk.

25 minutes later James approaches the supermarket and a **green line** on the big screen says the owner of ticket AX625RQ is allowed to enter the store for another 10 minutes.
James happily heads towards the entrance door, has its paper ticket scanned at the turnstiles and enjoys its queue-less shopping.

Scenario 4 Sara is another young, unemployed woman who lives in an outer borough of Naples. She does own a smartphone, even though it's a bit **old and sometimes sluggish** in the use. She uses it primarily for texting even though CLup is installed and seems to work.

It's 10 am and Sara needs to go shopping, so opens up CLup and reserves an entrance to the nearest store. She reaches the entrance, looks for her QR code and notices **her smartphone is suddenly misbehaving**, randomly rebooting and not letting her accomplish the task. She could have memorized her *presto code* but she actually did not, and asking for a manual check-in is not an option since human interactions have to be avoided - the staff would not let her in.

Sara feels annoyed, and decides she has no time to spend waiting for her smartphone to get back to normal, so she will try and access **like an offline customer**. The store is almost empty but some other offline customers are to get their tickets.
She looks at the big screen over the entrance, someone is currently occupying the self area but that particular store has room for 5 consecutive offline customers, so she enters the fenced area, stopping at «one turnstile distance» from the guy currently occupying the self area. In 2 minutes approximately, Sara is able to reach the self machine, have a new ticket printed and get back out.

The big screen announces both the offline tickets are allowed in (there's few persons inside), hence Sara heads towards the turnstiles and gets inside the supermarket, on her way to buying her next smartphone.

Scenario 5 Michael's family lives outside Messina, in a nice cottage by the sea. Panorama is beautiful, going shopping though requires some effort.
Either Michael or his wife, Laura, have to take the car and travel 25 kilometers of state road to the city. This typically requires up to 1 hour in rush hours, and 35 minutes on average.

This is the type of situation in which the possibility of **booking** an entrance comes in handy. It's 11.30 a.m.: Laura opens CLup and books an entrance at 5 p.m., providing an estimated shopping time of 1 hour.
CLup's **alternative stores functionality** also plays a fundamental role for Laura and her family, as they often head towards Ganzirri - another city on Sicily's east coast, opposite direction than Messina - to do their shopping. There are other shopping districts down there, typically less crowded and easier to reach. Today's best alternative happens to be a supermarket in Messina city though.

At 4.10 pm, Laura gets in the car and heads towards the booked store. She arrives at 5.05 p.m, has her QR booking code scanned and gets in.
However, children are usually hungry and Laura's three kids make no exception to this. She hurries getting the job done quickly, but she inevitably ends up **exceeding the 1 hour slot she had booked.**
Right now the store is full, and this apparently concerning problem leads CLup system to **alert with reasonable notice one user**, whose entrance would have been right after Laura's exit, that he will have to wait an additional 15 minutes before entering the store.
In the end, Laura manages to get outside the supermarket 65 minutes after she got in.

Had she required more than that, at 71st minute CLup would have warned the aforementioned user to add another 10 minutes delay to his entrance, and so forth. So that everybody stays safe and **no overcrowding** takes place.
Straightforwardly enough, Laura's **delay inevitably becomes root of possible discomfort.** Nonetheless, CLup engine makes note of Laura's behaviour and adds her last shopping time to her personal data: this is going to be taken into account the next time she books a visit, and over time the system will become able to **forecast her actual shopping time**, thus reducing consequent discomforts.
It is worth noting that this really unfortunate situation generates a problem since Laura's delay occurs specifically when the store is full, condition without which the problem would not have been so concerning.
Also, comparing CLup management of the situation with standard management indicates a fairly good improvement: without CLup, the next customer could not have booked its visit (much less, being warned about delays), but instead he would simply have reached the store at 7 p.m and crowded to wait an indeterminate amount of time outside the store.

Scenario 6 Valerio is a tech oriented grandfather, whose grandson is committed about technology and pushes him towards the use of electronic devices.
Everything tends to go well, except sometimes Valerio *mis-taps* something on its smartphone. Today Valerio is trying to get used to the new shopping app his grandson has provided him with, and accidentally makes a booking for late afternoon, at 6 o'clock, at a superstore near his house.
Valerio seems not to notice his mistake, and simply closes the app. Hence the booking remains valid.

We again find ourselves in the very unfortunate situation in which, at 6 o'clock the superstore is full and Valerio's booking means one less entrance for someone who needed it. This actually represents a problem for the very next 15 minutes after 6 p.m., since at 6.16 the booking is automatically cancelled and the next user in current queue is notified he can proceed.

Also, system makes note of Valerio's mistake and reports it to the superstore's Staff. They will be presented a comprehensive report, reading which they will be able to decide whether to contact Valerio for explanations or simply ignore the incident, taking into account factors only humans can evaluate (like Valerio's age).

However, CLup's reservation procedure includes an explicit confirmation dialogue, which is aimed at reducing this type of inconvenience.

Scenario 7 Let's now talk about Alex. He's a young man, living with some sort of anxiety: he's always worried of being late, loosing his seat, and so on. Today's day is going to be full of engagements and despite being roughly 7 o'clock a.m, Alex is now making a booking for a shopping during the evening.
Alex opens up CLup, clicks on the booking section and he's immediately presented with a list of 10 supermarkets available in Norcia, where he lives. Alex *chooses not to choose*: he books a slot at 7 p.m. at each one of the 10 supermarkets.

Now, this is a perfectly common situation, potentially induced by other factors - many of which being far less innocent than Alex's.
However, CLup's booking **system prevents booking more than two entrances on the same day**, hence preventing Alex's behaviour. He is in fact given negative response after the second booking, with a kind informative message explaining the situation.

The reasons behind this are straightforward. CLup aims at **improving, safening and optimizing the shopping experience** in general for the end user. This of course translates into the possibility of **shopping planning**, but has to **prevent prevarication** of some users over others as well.
Alex's behaviour, without this kind of policies, would inevitably reduce the number of persons allowed to go shopping at the same time, failing aforementioned purposes and possibly worsening the user experience w.r.t. the current situation.

Scenario 8 Middle-aged woman Debora also happens to be incline to «compulsive booking», but in a slightly different manner than Alex.
She is used to book services she will not have time to take advantage of in the end, and this also applies to CLup shopping booking. In the last 7 days, she booked a staggering total of 10 different slots, actually never going to the supermarket afterwards.

Again, this kind of situation can be avoided. CLup integrates a customizable policy about **fake booking prevention** that each store can configure according to their likes.
Between a minimum of 3 and a maximum of 10 fake bookings will result in CLup **shutting off user's ability to make bookings**, leaving only the "line up" functionality available for use. The booking feature may then be restored after manual intervention or a preset amount of time.

Note how this is not going to prevent anyone from shopping - since lining up is always allowed, just like normal - thus guaranteeing goals `G7`, `G9` and `G10`; at the same time though, it helps the booking feature works at its best by reducing disruptions, possibly increasing the overall stores' throughput.

2.2 Product Functions

CLup has NNNN main functionalities, each one of them is essential to reach the goals of the application.

2.2.1 Virtually queue in stores

This is the core of CLup, this function allows the user to virtually queue with a ticket to enter, as soon as possible, the supermarket he chooses. To do this the user needs to set his starting location, his preferred store chains and the distance range in which the application should look for stores, then CLup will present the stores located inside that range in form of a sorted list or a map with supermarkets highlighted. The list specifies for each store the distance from the starting location, the EWT to enter based on the size of the store and the number of people who already have a ticket for it, the number of people who are already inside and the maximum occupancy. The alternatives sorting system will be discussed later. The information displayed in the list is also visible from the map if the user selects a supermarket. When the user selects a store the application will show him the current EWT to enter and remind him that he has 15 minutes after his turn has come before the ticket definitely expires. Finally it will ask him if he is sure to reserve the entrance. This confirmation dialog is mainly useful to make sure that the user is aware of the timings and he can organize his commitments to go shopping in time. On the other hand, having a confirmation dialog is useful to prevent unintentional reservations by distracted or non-techy users. After the reservation is done, the user can follow the evolution of the line from the application which will give him real time information: the numbers of the tickets that are supposed to enter, the number of people ahead of the user and the estimated wait time. At this point the user should have enough information to arrive at the supermarket neither too early nor too late: in the first case he must wait in the car or away from the store, in the second one his ticket could expire and he would have to get a new one and wait again for his turn. When the time comes, CLup notifies the user that he should enter within 15 minutes. At the entrance the customer scans his QR code, or inserts the code of his ticket, and the turnstile lets him in. Obviously, if the store is not full and it is the turn of many customers to enter CLup allows them to access in any order, if a customer tries to scan the QR at the wrong time, ticket expired or that ticket has not been called yet, the turnstile won't let him pass. When the shopping is finished the customer scans again his QR code, or inserts the code, notifying the CLup system that there is one more entrance available in that store. People who do not have the app can still virtually queue by getting a ticket printed from the physical totems near CLup powered stores where there is also a big screen showing the entering ticket numbers. On the printed ticket it is specified the EWT but of course they will miss the possibility to be updated on the queue status because it is forbidden to wait in front of the store, also the totems give tickets only for the store where they are located. To get more accurate estimations, through the entering and exiting turnstiles, the system stores the shopping time of each user in order to build statistics. To avoid abuses of this function, it is not possible to reserve more than one entrance at the same time, the user will be able to reserve another entrance on the same day only after he left the first store. For the same reasons it will be later described that the store managers can prevent users from making new reservations if they let too many tickets expire..

2.2.2 Book entrance

This feature allows the user to book a ticket to enter in a specific time slot of a specific day, it is available in the mobile app and targets users who want to schedule their shopping time instead of just going as soon as possible. Just as for the previous function, the user needs a little setup for the starting location and eventually other filters, then he can visualize the matching solutions starting either from a list or a map, or starting from a calendar., Either of the starting views then takes the user to the other one (general calendar -> map/list for that day, general map/list -> calendar for that store) and after selecting both the store and the day a list of time slots for that day will be shown. Once the user decides the combination of store, day and time he can go on with the booking procedure and book his slot by giving confirmation. The application will process his request and, when the time comes, the application will generate the QR code (and notify the user) one hour in advance, from there the user will have 1:15 hours to enter the store before the ticket expires. If the user arrives early, even if the store is empty, the QR will only be valid for the chosen time. From this point on, the functioning is just as in the previous feature.
Also in this case abuses are discouraged: it is not possible to book more than an entrance per day and a ban mechanism is implemented, Clup counts the number of times a user reserves an entrance and then does not go to the store, if this value exceeds a threshold selected by the supermarket the user can not book an entrance to that supermarket until the number of booking absence is reset two weeks after .

2.2.3 Suggestions among different stores and times

This function was actually anticipated in the previous ones and aims at balancing the flux of people between stores and between different hours but, while the previous ones had a dedicated interface button, this one works in background: when the user wants to book or reserve an entrance, if he chooses to visualize stores in list mode, the list is accurately sorted with the purpose of putting in first place the least crowded stores and those with fewer probability of being chosen by other customers based on the statistics. If the user chooses the map visualization each store will have associated a color that indicates its crowdedness. In any case CLup will notify the user if there is a better solution outside of the set filters.

2.2.4 Store management

This function is only for the store managers, it allows them to:

- **view affluence statistics**: the manager can see the status of the supermarket crowdedness, the line to enter, the calendar with all the booked entrances and the throughput. However store managers
- **stop new entrances**: temporarily prevent coming users with valid tickets from entering the store
- **change current store capacity**: set the maximum capacity of the store at a certain time so that the automatic queuing system can plan the queue accordingly
- **inspect misbehaving user automatically generated reports**: The system automatically generates reports from users who miss their reservations or take too long shopping in the supermarket, these are then merged into statistics that allow the managers to use in an informed manner the next feature in this list
- **block or unblock users from accessing CLup for a certain store**: given the reports it is possible to prevent some users from reserving tickets, or also it is possible to let a user

reserve again after an eventual clarification with him. It is also possible to set thresholds for which the system automatically blocks or unblocks users

2.3 User Characteristics

The application targets two different entities: users and store managers.

- **Users** are the main concern of CLup, they represent every customer of supermarkets and actually differentiate between registered and offline users, they need to shop and they need to do it safely. They want to shop at stores without waiting outside and by spending as little time as possible using the application or totem, for this reason the application is very simple and intuitive.
- **Store managers** are supermarket accountants willing to find safer shopping solutions for their customers and their staff.

2.4 Assumptions, Dependencies and Constraints

2.4.1 Domain Assumptions

Follows a list of assumptions made about the domain CLup focuses on.

D1	Accesses to the store can be monitored
D2	Exits from the store can be monitored
D3	One customer per authorization given is allowed in by the Staff
D4	Users are reasonably able to manage their time while following the queue evolution
D5	Users can estimate the time required to arrive to the store
D6	Users who arrives too early at the supermarket don’t wait in front of the entrance
D7	Customers keep the safe distance
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D9	Users insert the right starting location or their GPS works
D10	Store managers give the right information about supermarkets
D11	Staff guarantees access control systems operativeness

3 Specific Requirements

Organize this section according to the rules defined in the project description.

3.1 External Interface Requirements

3.1.1 User Interfaces

Here will go wireframes

3.1.2 Hardware Interfaces

Both users and store managers can use the application through a mobile phone or a personal computer. Users unable to do so will use totems provided by stores.

3.1.3 Software Interfaces

map API
send ticket/booking requests to CLup
send turnstile entrances/exits to CLup
query automatically generated reports and obtain their statistics

3.1.4 Communication Interfaces

The only type of communication required by CLup is a stable internet connection.

3.2 Functional Requirements

3.2.1 List of requirements

R1	Every user can generate a quick ticket for any store
R2	Whenever user makes initiates a booking procedure, CLup must be able to compute a suggested least crowded time slot based on historical data
R3	CLup must elaborate and upload data about current global customer affluence to the store during use
R4	CLup must admit only valid QR codes for entrance
R5	CLup must allow users to know current queue status
R6	CLup must update user on tickets' validity change
R7	CLup must inform offline users about new tickets (un)availability
R8	CLup must allow users to indicate which product category they are going to purchase while booking
R9	CLup must suggest alternative stores when the combination of selected store|time gives no results
R10	CLup must reserve a non null number of paper tickets at any time for offline customers use
R11	CLup must gather all stores' data about entrance fluxes
R12	CLup is able to cross affluence data of any supermarket
R13	CLup keeps track of people who book an entrance and don't come
R14	CLup allows store managers to stop quick tickets availability
R15	CLup is able to generate QR codes
R16	CLup is able to authenticate users
R17	CLup is able to store users' data
R18	CLup is able to process users' data
R19	CLup makes quick ticket invalid after 15 minutes delay
R20	CLup can use stores' data to sort every store by crowdedness
R21	Users can see available day/time slots of a supermarket through CLup
R22	CLup shows to store managers flux data about their supermarket (forse questo è quello che intendeva R3)
R23	CLup must be able to process reservations

Table 1: Requirements list

3.2.2 Mapping

Goals	Requirements	Domain Assumptions
G1	R1, R6, R7, R10, R15,R23	D1,D5,D8
G2	R2, R11, R12, R18, R21	D1, D2, D8, D10
G3	R9, R13, R15, R21, R23	D5, D8, D10
G4	R3, R11, R22	D1, D2, D8, D11
G5	R4, R6, R13, R15, R19	D1, D2, D3, D8, D10, D11
G6	R1, R2, R5, R6	D4, D5, D6, D7, D8, D10
G7	R1, R5, R6, R7, R9, R10, R20, R21, R23	D2, D4, D5, D8, D9, D10
G8	R1, R7, R10	D6, D7, D8, D10
G9	R3, R11, R12, R20	D1, D2, D8, D9, D10, D11
G10	R2, R3, R4, R11, R12, R20	D1, D2, D3, D7, D8, D10, D11

Table 2: Goal mapping summary

G1	**Anybody is guaranteed possibility to make shopping at any supermarket in reasonable time (def. reasonable)**
R1	Every user can generate a quick ticket for any store
R6	CLup must update user on tickets' validity change
R7	CLup must inform offline users about new tickets (un)availability
R10	CLup must reserve a non null number of paper tickets at any time for offline customers use
R15	CLup is able to generate QR codes
R23	CLup must be able to process reservations
D1	Accesses to the store can be monitored
D5	Users can estimate the time required to arrive to the store
D8	Malicious users are not enough in number or coordination to prevent Clup to work

Table 3: G1 Mapping

G2	**Users can get to know the least crowded time slots**
R2	Whenever user makes initiates a booking procedure, CLup must be able to compute a suggested least crowded time slot based on historical data
R11	CLup must gather all stores' data about entrance fluxes
R12	CLup is able to cross affluence data of any supermarket
R18	CLup is able to process users' data
R21	Users can see available day/time slots of a supermarket through CLup
D1	Accesses to the store can be monitored
D2	Exits from the store can be monitored
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D10	Store managers give the right information about supermarkets

Table 4: G2 Mapping

G3	**Fair users can make a reservation to enter in a supermarket**
R9	CLup must suggest alternative stores when the combination of selected store|time gives no results
R13	CLup keeps track of people who book an entrance and don't come
R15	CLup is able to generate QR codes
R21	Users can see available day/time slots of a supermarket through CLup
R23	CLup must be able to process reservations
D5	Users can estimate the time required to arrive to the store
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D10	Store managers give the right information about supermarkets

Table 5: G3 Mapping

G4	**Stores can easily monitor fluxes**
R3	CLup must elaborate and upload data about current global customer affluence to the store during use
R11	CLup must gather all stores' data about entrance fluxes
R22	CLup shows to store managers flux data about their supermarket
D1	Accesses to the store can be monitored
D2	Exits from the store can be monitored
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D11	Staff guarantees access control systems operativeness

Table 6: G4 Mapping

G5	**Only authorized users can access**
R4	CLup must admit only valid QR codes for entrance
R6	CLup must update user on tickets' validity change
R13	CLup keeps track of people who book an entrance and don't come
R15	CLup is able to generate QR codes
R19	CLup makes quick ticket invalid after 15 minutes delay
D1	Accesses to the store can be monitored
D2	Exits from the store can be monitored
D3	One customer per authorization given is allowed in by the Staff
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D10	Store managers give the right information about supermarkets
D11	Staff guarantees access control systems operativeness

Table 7: G5 Mapping

G6	**Crowds are dramatically reduced outside supermarket stores**
R1	Every user can generate a quick ticket for any store
R2	Whenever user makes initiates a booking procedure, CLup must be able to compute a suggested least crowded
R5	CLup must allow users to know current queue status
R6	CLup must update user on tickets' validity change
D4	Users are reasonably able to manage their time while following the queue evolution
D5	Users can estimate the time required to arrive to the store
D6	Users who arrives too early at the supermarket don't wait in front of the entrance
D7	Customers keep the safe distance
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D10	Store managers give the right information about supermarkets

Table 8: G6 Mapping

G7	CLup should not decrease customer affluence beyond a reasonable level w.r.t. to normal (→ define reasonable)
R1	Every user can generate a quick ticket for any store
R5	CLup must allow users to know current queue status
R6	CLup must update user on tickets' validity change
R7	CLup must inform offline users about new tickets (un)availability
R9	CLup must suggest alternative stores when the combination of selected store|time gives no results
R10	CLup must reserve a non null number of paper tickets at any time for offline customers use
R20	CLup can use stores' data to sort every store by crowdedness
R21	Users can see available day/time slots of a supermarket through CLup
R23	CLup must be able to process reservations
D2	Exits from the store can be monitored
D4	Users are reasonably able to manage their time while following the queue evolution
D5	Users can estimate the time required to arrive to the store
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D9	Users insert the right starting location or their GPS works
D10	Store managers give the right information about supermarkets

Table 9: G7 Mapping

G8	Same shopping capabilities guaranteed to offline users
R1	Every user can generate a quick ticket for any store
R7	CLup must inform offline users about new tickets (un)availability
R10	CLup must reserve a non null number of paper tickets at any time for offline customers use
D6	Users who arrives too early at the supermarket don't wait in front of the entrance
D7	Customers keep the safe distance
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D7	Customers keep the safe distance

Table 10: G8 Mapping

G9	Find the best (less crowded, soonest available) alternative among local supermarket stores (of same franchise only?)
R3	CLup must elaborate and upload data about current global customer affluence to the store during use
R11	CLup must gather all stores' data about entrance fluxes
R12	CLup is able to cross affluence data of any supermarket
R20	CLup can use stores' data to sort every store by crowdedness
D1	Accesses to the store can be monitored
D2	Exits from the store can be monitored
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D9	Users insert the right starting location or their GPS works
D10	Store managers give the right information about supermarkets
D11	Staff guarantees access control systems operativeness

Table 11: G9 Mapping

G10	Supermarkets do not overcrowd
R2	Whenever user makes initiates a booking procedure, CLup must be able to compute a suggested least crowded time slot based on historical data
R3	CLup must elaborate and upload data about current global customer affluence to the store during use
R4	CLup must admit only valid QR codes for entrance
R11	CLup must gather all stores' data about entrance fluxes
R12	CLup is able to cross affluence data of any supermarket
R20	CLup can use stores' data to sort every store by crowdedness
D1	Accesses to the store can be monitored
D2	Exits from the store can be monitored
D3	One customer per authorization given is allowed in by the Staff
D7	Customers keep the safe distance
D8	Malicious users are not enough in number or coordination to prevent Clup to work
D10	Store managers give the right information about supermarkets
D11	Staff guarantees access control systems operativeness

Table 12: G10 Mapping

3.2.3 Use cases

1. **Registration of new account**

Name	Registration of new account
Actors	User
Entry Condition	User installed and opened the app and doesn't have an account or wants to register another one
Event Flow	(a) User opens the app. (b) Login screen loads. (c) User taps "Sign up". (d) TODO REGISTRATION WIREFRAME.
Exit Conditions	User now has an account with which he can log in
Exception	(a) There is no internet when the user presses "create account" "No internet" popup, the user can either wait for internet to come back or discard the incomplete account creation by going back to main screen

2. **User login**

Name	User login
Actors	User
Entry Condition	User already has an account
Event Flow	(a) User opens the app. (b) Login screen loads. (c) User already has an account. (d) User inputs his credentials and presses “login” (e) Main screen loads
Exit Conditions	User logged in and is now in main screen, from where he can virtually access all of the app’s functionalities
Exception	(a) Credentials are wrong wrong credentials popup, user stays in login screen (b) There is no internet connection no internet warning pop up, user still logs in if he used previously inserted and saved credentials so he can still edit settings and filters or look at his history

3. **Quick ticket request**

Name	ASAP ticket request
Actors	User
Entry Condition	User successfully logged in and is in main screen
Event Flow	(a) User taps “Show stores” (b) User is taken to “Quick results” screen (c) IF LIST MODE (from settings) stores with queue time, size and distance are shown in a list according to setted filters, if show more is pressed more stores are loaded and the list is made scrollable, if location button is pressed a map showing the store’s location is shown (TODO browse stores on map) IF MAP MODE (from settings) stores with queue time, size and distance are shown on a map, if list button is pressed User is taken to the previously descripted “list mode” case (d) User taps on a store (e) Confirmation pop up is shown (f) If user confirms “Ticket screen” is shown, otherwise he is taken back to 3
Exit Conditions	User has a ticket with updating due time to enter the store
Exception	(a) The store blocked ticket requests after 5 user is told that the store is no longer available and remains in list/map to make an eventual different choice (b) There is no internet connection after 1 user stays in main screen with a dismissable “no internet” pop up (c) User already has an ASAP ticket after 1 user stays in main screen with a dismissable “you can only have one ASAP ticket”

4. **Quick ticket request at physical Totem**

Name	ASAP ticket request at physical Totem
Actors	User
Entry Condition	User starts interacting with CLup tablet (?) outside the store
Event Flow	(a) POSSIBLE IDENTIFICATION (fiscal code or ID card) (b) User sees current queue for this store and decides whether to confirm or cancel (c) A card ticket with remainder and QR is printed (d) User leaves the station with his printed ticket
Exit Conditions	User has a ticket that he can scan to enter the store when he comes back at the written date and time
Exception	(a) The store blocked ticket requests User can't get a ticket and is asked to come back another time

5. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the "filters" button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

6. **Plan visit**

Name	ASAP ticket request
Actors	User
Entry Condition	User successfully logged in and is in main screen
Event Flow	(a) User taps "Show stores" (b) User is taken to "Quick results" (c) User inputs the expected amount of time he will be shopping (d) Calendar or store map/list is shown depending on what was loaded given the filter settings (e) User chooses date or store (from map/list) (f) Stores map/list or calendar depending on what was shown already (g) User chooses the store or the day (h) CLup shows time slots for that store on that day (i) User taps on a time slot (j) Confirmation pop up is shown (k) If user confirms "Ticket screen" is shown, otherwise he is taken back to 3
Exit Conditions	User has a ticket with updating due time to enter the store
Exception	(a) The store blocked ticket requests after 5 user is told that the store is no longer available and remains in list/map to make an eventual different choice (b) There is no internet connection after 1 user stays in main screen with a dismissable "no internet" pop up (c) User already has an ASAP ticket after 1 user stays in main screen with a dismissable "you can only have one ASAP ticket" (d) Day/store/time slot is full the choice is refused with a popup and user has to choose an alternative

7. **Enter store**

Name	Enter store
Actors	User
Entry Condition	User has a valid QR ticket and is at the store entrance
Event Flow	(a) User scans his QR ticket or inputs his alphanumeric code at the turnstile (b) The turnstile informs the user he can enter and unlocks (c) The user enters the store
Exit Conditions	The user is in the store and can start shopping
Exception	(a) The ticket expired its 15 minutes validity time The turnstile remains locked and informs the user that his ticket is not valid, the ticket expiration causes a report to be generated for the manager independently from the fact that the user tried to enter anyway (b) The code is wrong The turnstile remains locked and informs the user that the code is not valid

8. **Exit store**

Name	Exit store
Actors	User
Entry Condition	User has a valid QR ticket and is exiting the store
Event Flow	(a) User scans his QR ticket or inputs his alphanumeric code or inputs his CLup email or nickname at the turnstile (b) The turnstile informs the user he can exit and unlocks (c) The user exits the store
Exit Conditions	The user is in the store and can start shopping
Exception	(a) The code is wrong or no code can be provided or credentials are wrong The turnstile informs the user that he has to try again and after the third try it is unlocked anyway, the user exit won't be tracked and will cause a report for the misinterpreted long shopping to be generated for him anyway

9. **Store staff stops new entrances**

Name	Stops new entrances
Actors	Store manager
Entry Condition	Store manager is in management web page
Event Flow	(a) Staff clicks on stop new entrances (b) Staff is asked to insert admin password (c) Staff is asked to confirm and he does
Exit Conditions	No new tickets can be issued and currently released tickets are put on hold, meaning that valid tickets won't be accepted by the turnstiles (queue time replaced with a suspened entrances message) The page now offers the possibility to allow entrances again
Exception	None

10. **Store manager view affluence statistics**

Name	View affluence statistics
Actors	Store manager
Entry Condition	The manager is in management web page
Event Flow	(a) Staff clicks on "View affluence statistics" (b) A page showing various statistics with customizable filters and options about people affluences is shown
Exit Conditions	Manager knows their customers' behaviours (e.g. affluence and permanence times) and can act accordingly
Exception	(a) None

11. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the “filters” button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

12. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the “filters” button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

13. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the "filters" button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

14. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the “filters” button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

15. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the “filters” button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

16. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the “filters” button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

17. **Edit filters**

Name	Edit filters
Actors	User
Entry Condition	User pressed filter button from main screen
Event Flow	(a) User is in main screen and taps the "filters" button (b) User is in filters screen (c) User changes the filter parameters he wants to change among: i. distance range ii. store type iii. default booking time (ignored by ticket request) iv. default calendar or stores first when booking (ignored by ticket request) v. default map or list view when booking (ignored by ticket request) vi. ??? (d) User presses back to main screen button (e) Popup to confirm and save or discard the new filters is shown
Exit Conditions	New filters are set and they will affect the next ticket request or visit plan
Exception	(a) User closes the app without saving the filters Filter modifications are lost

3.3 Performance Requirements

3.4 Design Constraints

3.4.1 Standards compliance

3.4.2 Hardware limitations

3.4.3 Any other constraint

3.5 Software System Attributes

3.5.1 Reliability

3.5.2 Availability

3.5.3 Security

3.5.4 Maintainability

3.5.5 Portability

4 Formal Analysis Using Alloy

Organize this section according to the rules defined in the project description.

4.1 Alloy code

```
enum Day{Monday, Tuesday, Wednesday, Thrusday, Friday, Saturday, Sunday}

abstract sig Bool{}
one sig True extends Bool{}
one sig False extends Bool{}
sig Char{} -- using this to be able to write constraints on strings length
sig Float{
    integer: Int,
    decimal: Int
}{
    decimal>0
}

sig Date{
    year: Int,
    month: Int,
    day: Int,
}{
    year>0
    --year<=3000 //we can't say this with <=10 bits for Int
    month>0
    month<13
    day>0
    day<32
}

sig Time{
    date: Date,
    hour: Int,
    minutes: Int,
    seconds: Int,
}{
    hour<24
    hour≥0
    minutes<60
    minutes≥0
    seconds<60
    seconds≥0
}

sig RelativeTime{
    validDay: one Day,
    hour: one Int,
    minutes: one Int,
    seconds: one Int,
}{
    hour<24
    hour≥0
    minutes<60
    minutes≥0
    seconds<60
    seconds≥0
}

sig Location{
    latitude: one Float,
    longitude: one Float
}/*{ --it'd be nice, but solver doesn't let us
    latitude.integer<85
    latitude.integer>-85
    longitude.integer<180
    longitude.integer>-180
}*/
```



```
sig Store{
    commercialName: seq Char,
    longName: seq Char,
    location: one Location,
    opensAt: set RelativeTime,
    closesAt: set RelativeTime,
    twentyfourSeven: one Bool,
    occupantsMax: one Int
}{
    twentyfourSeven=False implies (#opensAt>0 ∧ #closesAt>0 ∧ #opensAt=#closesAt)
    twentyfourSeven=True implies (#opensAt=0 ∧ #closesAt=0)
    occupantsMax>0
}

abstract sig Person {
    name: seq Char,
    surname: seq Char,
    fc: seq Char,
    customerId: seq Char,
    tickets: set Ticket,
    reservations: set BookingReservation,
    currentLocation: lone Location //used for timed notifications
}{
    #name>2
    #surname>2
    //#fc>=11
    //#customerId=12
}

one sig Now{
    time: one Time
}

sig Customer extends Person{}

sig StaffMember extends Person{
    cardID: seq Char,
    nowWorkingAt: one Store,
    active: one Bool,
    level: one Int
}{
    #cardID>3
    level>0
}

sig BookingReservation {
    applicant: one Person,
    startTime: one Time,
    where: one Store,
    endTime: one Time,
    id: seq Char
}{
    //aTimeBeforeB[startTime, endTime]
}

sig Ticket{
    owner: one Person,
    parentQueue: Queue, --identifies parent Queue
    id: seq Char,
    prestoCode: seq Char,
    entranceTime: one Time,
    valid: one Bool,
    active: one Bool,
    scannedIn: lone Time,
    scannedOut: lone Time
}{
    #this.@id>0
}

sig Notification{
    recipient : one Person,
    message: seq Char,
```



```
    disposal : one Time //when are we sending this notification
}

sig Queue{
    members: seq Ticket,
    store: one Store, -- each queue refers to a specific store, 1 store <--> 1 queue
    id: seq Char, --each queue has an ID
    estimatedNextEntrance: lone Time
}{
    #members>0
    #this.@id>0 -- TODO
}

one sig NotificationsDB{
    notifications: set Notification
}

one sig Queues{
    queuesList: set Queue
}

one sig CustomersDB{
    customers: set Customer
}

one sig StaffDB{
    staffMembers: set StaffMember
}

one sig BookingsDB{
    bookingsList: set BookingReservation
}

one sig StoresDB{
    storesList: set Store
}

--facts----------------------------------------
fact userAndFiscalCodesUnique{
    all disj pers,pers1 : Person | pers.fc ≠ pers1.fc ∧ pers.customerId≠pers1.customerId
}

fact reservationConsistency{
    all r: BookingReservation | r.startTime.date=r.endTime.date ∧ aTimeBeforeB[r.
        ↪ startTime, r.endTime]
}

fact noDuplicatedCustomers{
    all disj cust,cust1: Person | cust.customerId ≠cust1.customerId
}

fact dayConsistency{ --we should also handle leap years...
    all date : Date | (date.month=11 ∨date.month=4 ∨ date.month=6 ∨ date.month=9) implies
        ↪  date.day<31 ∧
    (date.month=2) implies date.day<30
}

fact noDBMismatch{
    all p : Person | (isCustomer[p] implies !isStaff[p]) ∧ (isStaff[p] implies !
        ↪ isCustomer[p])
}

fact allPeoplesBelongToDB{
    all p: Person | isCustomer[p] ∨ isStaff[p]
}

fact allStoresBelongToDB{
    all s: Store | s in StoresDB.storesList
}

fact allNotificationsBelongToDB{
    all n: Notification| n in NotificationsDB.notifications
```



```
}

/*fact allQueuesBelongToDB{
    all q: Queue | q in Queues.queuesList
}*/

fact eachStoreOneQueueMax{
    all disj q, q1 : Queue | q.store≠q1.store
}

fact noDuplicatedTickets{
    all q: Queue | !q.members.hasDups
}

fact userHasNoMultipleTicketsSameDay{
    all disj t,t1: Ticket| (t.owner=t1.owner ∧ t.valid=True ∧ t1.valid=True) implies
    (aDateBeforeB[t.entranceTime.date, t1.entranceTime.date]∨aDateBeforeB[t1.entranceTime
        ↪ .date, t.entranceTime.date])
}

fact eachTicketHasParentQueue{
    all t: Ticket | one q: Queue | q=t.parentQueue
}

fact eachReservationHasApplicant{
    all r: BookingReservation | one p: Person | p=r.applicant
}

fact twoWayCorrespondanceTicketQueue{
    all t: Ticket | all q: Queue | t.parentQueue=q iff t in q.members.elems
}

fact twoWayCorrespondanceReservationOwner{
    all r:BookingReservation | all p: Person | r.applicant=p iff r in p.reservations
}

fact twoWayCorrespondanceTicketOwner{
    all t:Ticket, p:Person | (t.owner=p implies t in p.tickets) ∧ (t in p.tickets implies
        ↪  t.owner=p)
}

fact onlyOneBookingPerDayPerUser{
    all disj b, b1: BookingReservation | !(b.startTime.date=b1.startTime.date ∧ b.
        ↪ applicant=b1.applicant)
}

fact eachOpeningDayHasAlsoClosing{
    all s:Store| all o:RelativeTime | o in s.opensAt implies
    (one c:RelativeTime | c.validDay=o.validDay ∧ c in s.closesAt)
}

fact closingTimeAfterOpening{
    all s:Store | s.twentyfourSeven=False implies (all o,c: RelativeTime|
    (o in s.opensAt ∧ c in s.closesAt ∧ o.validDay=c.validDay) implies
        ↪ aRelativeTimeBeforeB[o, c])
}

--functions-------------------------------------
fun retrieveTicketsStore[t:Ticket]: one Store {
    t.parentQueue.store
}

fun getCurrOccupants[q: Queue]: one Int {
    #{t: Ticket | t.active=True ∧ t in q.members.elems}
}

fun getBookedOccupants[s: Store, start:Time, end:Time] : one Int{
    #{x: BookingReservation | x.where = s ∧ (sameTime[start, x.startTime]∨ aTimeBeforeB[
        ↪ start, x.startTime])
    ∧ (sameTime[x.startTime, end] ∨ aTimeBeforeB[x.endTime, end])}
    //number of reservations whose start time >= start and end time <= end
}
```



```
fun computeDisposalTime[ticketTime: Time, userLocation: Location]: one Time{
    {x: Time}
}

--predicates -----------------------------------
pred isCustomer[p:Person]{
    p in CustomersDB.customers
}

pred isStaff[p:Person]{
    p in StaffDB.staffMembers
}

pred aDateBeforeB[a:Date , b: Date]{
    a.year<b.year∨(a.year=b.year ∧ a.month< b.month)∨(a.year=b.year ∧ a.month= b.month∧a.
        ↪ day<b.day)
}

pred aRelativeTimeBeforeB[a,b:RelativeTime]{
    a.validDay=b.validDay ∧ ((a.hour<b.hour) ∨ (a.validDay=b.validDay ∧ a.hour=b.hour ∧ a
        ↪ .minutes<b.minutes) ∨
    (a.validDay=b.validDay ∧ a.hour=b.hour ∧ a.minutes=b.minutes∧a.seconds<b.seconds))
}

pred aTimeBeforeB[a: Time, b: Time]{
    aDateBeforeB[a.date, b.date]∨ (a.date=b.date ∧ a.hour<b.hour) ∨ (a.date=b.date ∧ a.
        ↪ hour=b.hour ∧ a.minutes<b.minutes) ∨
    (a.date=b.date ∧ a.hour=b.hour ∧ a.minutes=b.minutes∧a.seconds<b.seconds)

}

pred sameTime[a, b : Time]{
    !(aTimeBeforeB[a,b]∨aTimeBeforeB[b, a])
}

pred userHasBooked[p: Person]{
    some r: BookingReservation | r in BookingsDB.bookingsList ∧ r.applicant=p
}

pred hasTicket[p: Person]{
    some q: Queue | some t: Ticket | t.owner=p ∧ t in q.members.elems
}

pred maxOccupantsNotExceeded[s: Store]{
    all q: Queue| q.store = s implies plus[getCurrOccupants[q], 1]<s.occupantsMax
}

pred bookingsNotExceedingMaxOccupants[s: Store, start: Time, end: Time]{
    getBookedOccupants[s, start, end]+1≤s.occupantsMax
}

pred hasTicketForThisStore[p: Person, s: Store]{
    some t: Ticket | t in p.tickets ∧ retrieveTicketsStore[t]=s
}

pred activateTicket[t : Ticket]{
    t.active=True ∧ t.scannedIn=Now.time
}

pred expireTicket[t: Ticket]{
    t.active=False ∧ t.scannedOut=Now.time ∧ t.valid=False
}

pred allowUserIn[p:Person, thisStore: Store]{
     //ensures we're not going to exceed store's capacity with a new ticket
    maxOccupantsNotExceeded[thisStore] ∧ (some t: Ticket| hasTicketForThisStore[p,
        ↪ thisStore] ∧ t.valid=True
    ∧ activateTicket[t]) //activates ticket to track user's entrance/exit
}

//adding a reservation
```



```
pred book[b, b': BookingsDB, a: Person, start:Time, store: Store, end: Time]{
    bookingsNotExceedingMaxOccupants[store, start, end]//ensures we're not going to
        ↪ exceed store's capacity with new bookings
    aTimeBeforeB[Now.time, start] //we don't want reservations in the past
    b'.bookingsList.applicant = b.bookingsList.applicant +a
    b'.bookingsList.startTime= b.bookingsList.startTime + start
    b'.bookingsList.where= b.bookingsList.where + store
    b'.bookingsList.endTime= b.bookingsList.endTime + end
}


pred getQuickTicket[q, q': Queues, a: Person, t:Time, s: Store]{
    q'.queuesList.members.elems.owner = q.queuesList.members.elems.owner + a
    q'.queuesList.members.elems.entranceTime= q.queuesList.members.elems.entranceTime+t
    q'.queuesList.store= q.queuesList.store + s
    (all ticket: Ticket | (ticket.owner=a∧ticket.entranceTime=t)implies ticket.valid=True
        ↪ )//new tickets are valid
    (some v1, v2: NotificationsDB | { //generate notifications accordingly
        v2.notifications.recipient=v1.notifications.recipient + a
        v2.notifications.disposal=v1.notifications.disposal + computeDisposalTime[t, a.
            ↪ currentLocation]
    })
}

pred deleteQuickTicket[q, q': Queues, t: Ticket]{
    q'.queuesList.members.elems.owner = q.queuesList.members.elems.owner - t.owner
    q'.queuesList.members.elems.entranceTime= q.queuesList.members.elems.entranceTime - t
        ↪ .entranceTime
    (#q'.queuesList.members.t≥1) ∨ (q'.queuesList.store= q.queuesList.store -
        ↪ retrieveTicketsStore[t])
    (all ticket: Ticket | (ticket.owner=t.owner∧ticket.entranceTime=t.entranceTime)
        ↪ implies ticket.valid=False)//old tickets are invalid
}

pred temporaryStopStore[s: Store]{
    all q: Queue, t : Ticket | (q.store = s ∧ t in q.members.elems) implies t.valid=False
}

pred exitStore[t: Ticket]{
    expireTicket[t] ∧ (some q, q': Queues | deleteQuickTicket[q, q', t])
}


pred notificationDispatch{
    all n: Notification| aTimeBeforeB[n.disposal, Now.time] implies sendNotification[n]
}

pred sendNotification[n: Notification]{}

--assertions-----------------------------------
assert customersInCustomersDB{
    all c: Customer | isCustomer[c]
}

assert staffMembersInStaffDB{
    all s: StaffMember | isStaff[s]
}

assert noOrphanTicket{
    no t: Ticket | some p: Person | t.owner=p ∧ !hasTicket[p]
}

assert noTicketNoEntry{
    no p: Person, s: Store | !hasTicketForThisStore[p,s] ∧ allowUserIn[p, s]
}

assert getQuickTicketGrantsEnter{
    all p: Person, t:Time, s:Store, disj q,q': Queues | getQuickTicket[q, q',p, t, s]
        ↪ implies allowUserIn[p,s]
}

assert generateTicketDoesNotExceedMaxOccupants{
```



```
    all p: Person, t:Time, s:Store, disj q,q': Queues | getQuickTicket[q, q',p, t, s]
        ↪ implies maxOccupantsNotExceeded[s]
}

assert bookingDoesNotExceedMaxOccupants{
    all p: Person, s,e :Time, st:Store, disj b,b': BookingsDB | book[b, b', p, s, st, e]
        ↪ implies bookingsNotExceedingMaxOccupants[st, s, e]
}

assert neverAllowInMoreThanMax{
    no p:Person, s:Store | allowUserIn[p, s] ∧ !maxOccupantsNotExceeded[s]
}

assert delUndoesAdd{
    all disj q, q', q'': Queues, p: Person, t: Time, s: Store, ticket: Ticket |
    (ticket.owner=p ∧ ticket.entranceTime=t ∧ getQuickTicket[q, q', p, t, s] ∧
        ↪ deleteQuickTicket[q', q'', ticket])
    implies
    (q.queuesList.members = q''.queuesList.members)
}

--commands----------------------------------------
check delUndoesAdd for 7 Int
check neverAllowInMoreThanMax for 7 Int
check bookingDoesNotExceedMaxOccupants for 7 Int
check generateTicketDoesNotExceedMaxOccupants for 7 Int
check getQuickTicketGrantsEnter for 7 but 7 Int
check customersInCustomersDB for 7 Int
check staffMembersInStaffDB for 7 Int
check noOrphanTicket for 7 Int
check noTicketNoEntry for 7 Int
run {some t: Ticket | t.valid=True} for 7 Int
run hasTicket for 7 Int
run hasTicketForThisStore for 7 Int
run {some p:Person | hasTicket[p] ∧ isCustomer[p]} for 7 Int
run {some p:Person | hasTicket[p] ∧ isStaff[p]} for 7 Int
run {some p:Person, s:Store | allowUserIn[p,s] ∧ isStaff[p]} for 7 Int
run {some p:Person, s:Store | allowUserIn[p,s] ∧ isCustomer[p]} for 7 Int
run userHasBooked for 7 Int
run isCustomer for 7 Int
run isStaff for 7 Int
run aDateBeforeB for 7 Int
run book for 7 Int
run getQuickTicket for 7 Int
run {some s, s1: Store | s.twentyfourSeven=True ∧ s1.twentyfourSeven=False} for 7 Int
run aTimeBeforeB for 7 Int
run maxOccupantsNotExceeded for 7 Int
run temporaryStopStore for 7 Int
run notificationDispatch for 7 Int
run exitStore for 7 Int
run deleteQuickTicket for 7 Int

/* v2.0 Useful additions:
- bookings constraints and 2-way correspondances with applicant [DONE]
- no overcrowding: [DONE]
    - store capacity [DONE]
    - check when creating new ticket [DONE]
    - check when booking [DONE]
- exit from store deactivates ticket [DONE]
- delete ticket when deactivated [DONE]
- staff:
    - temporary deactivate store [DONE]
- notifications:
    - add location to the customer [DONE]
    - add queue estimated next entrance [DONE]
- goal-related assertions (no overcrowding, no multiple tickets etc) [DONE]
*/
```


5 Effort Spent

Provide here information about how much effort each group member spent in working at this document. We would appreciate details here.

References

[1] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software and Systems Modeling*, 10(3):313–336, 2011.

