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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

Contents

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

List of Figures

List of Tables

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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.

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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	Users can see alternative stores
G6	Only authorized users can access
G7	Crowds are dramatically reduced outside supermarket stores
G8	CLup should not decrease customer affluence beyond a reasonable level w.r.t. to normal ($\rightarrow$ define reasonable)
G9	Prevent department overcrowding
G10	
G11	No inhibition of purchase/access to any product category
G12	Same shopping capabilities guaranteed to offline users
G13	Find the best (less crowded, soonest available) alternative among local supermarket stores (of same franchise only?)

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

1.3.2 Acronyms

1.3.3 Abbreviations

1.4 Revision History

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

1.5 Reference Documents

1.6 Document Structure

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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

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.

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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.

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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

2.3 User Characteristics

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	(check) 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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

3 Specific Requirements

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

3.1 External Interface Requirements

3.2 Functional Requirements

3.3 Performance Requirements

3.4 Design Constraints

3.5 Software System Attributes

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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
}*/
```

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

```
    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 CustomerDB{
    customers: set Customer
}

one sig StaffDB{
    staffMembers: set StaffMember
}

one sig Bookings{
    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 eachStoreOneQueueMax{
    all disj q, q1 : Queue | q.store≠q1.store
}

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

```
}

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.twentyfour=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 CustomerDB.customers
}

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

```
}

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 Bookings.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': Bookings, 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]{
```

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

```
    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': Bookings | book[b, b', p, s, st, e]
        ↪ implies bookingsNotExceedingMaxOccupants[st, s, e]
}

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

```
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.twentyfour=True ∧ s1.twentyfour=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]
*/
```

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

5 Effort Spent

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved

References

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

Copyright © 2020, Neroni | Pozzi | Vetere – All rights reserved