Politecnico di Milano
A.A. 2015-2016
Software Engineering 2: "MyTaxi"
Requirements Analysis and Specifications Document

Manzi Giuseppe (mat. 854470) &
Nicolini Alessandro (mat. 858858)

CONTENTS

1. Introduction

1.1 Purpose

Software Requirements Specification Document (RASD) is an unambiguous and complete specification document, helping *Clients* to describe their wishes and developers to understand what the *Client* wants. This document also shows constraints and the limit of the software and simulates the typical use cases that will occur after the development. RASD is intended to all developer and programmer who have to implement the requirements, to system analyst who want to integrate other system with this one, and could be used as a contractual basis between the *Client* and the developer.

1.2 Actual system

Till last year there were different taxi companies providing the service of managing taxi requests for the whole city. Any taxi driver could become member of one of these companies. When a *costumer* phoned the company, the system of the company forward the call to the right taxi driver. At the beginning of this year the government of the city decided to unify all the companies and provide a unique service for all the taxis.

1.3 Scope

The aim of the project is to create a new system to improve the quality of the service. The passenger can send his request either through a web application or a mobile app, in order to simplify the access to the service thanks to a user-friendly interface and simple functions. The use of mobile app will also be an advantage for taxi drivers, who can install it on their own private device or rent one provided by the government of the city, sign in and receive notification of requests they have to manage.
To guarantee a fair management of both requests and taxi queues, the city will be divided in taxi zones (approximately 2 km^2 each). The oldest request coming from a *costumer* must be the first managed in his zone. A request must be forwarded to the taxi that is free and has been waiting in that zone for more time.

1.4 Actors

Guest user: He can decide either to sign up or log in (whether already registered) or to request a taxi without signing in.
Registered costumer: He can reserve/request a taxi or see the list of all the rides he took or he has reserved. Reserved ride can be modified if it's more than two hours before the meeting time or deleted at any time.
Employee: He can register taxi drivers, after checking their identity.
Taxi driver: He is notified when a request/reservation is assigned to him. He can accept or decline the *assignment*. On his main screen he can see the map of the city, automatically see the way to the location of the *costumer* he has to pick up and look for the best way to a specific address.

1.5 Goals

[G1] Allow taxi drivers to have a personal reserved area that helps them managing rides.
[G2] Allow guest users to request a taxi to reach him to a specific address as soon as possible.
[G3] Allow *costumer* to have a personal reserved area. In addition to functionalities provided to guest users, a *registered costumer* can reserve a taxi for a specific time (at least two hours in advance), check previous requests and reservations and modify or delete them.
[G4] Guarantee a *fair queue management* (see 1.6.1).
[G5] Manage the distribution of the taxis in each zone and notify taxi drivers the need to move from a zone to another.

1.6 Definition, Acronyms and Abbreviations

1.6.1 Definition

- **Assignment**: the assignment by the system of a ride to a specific taxi.
- **Client**: the person/the company who commissioned the project.
- **Customer**: a person who makes use of the service provided by the *Client*.
  - **Registered costumer**: a *costumer* who signed up with the system and is logged in for the current session.
- **Employee of the "Mobility office"**: a person who works for the "Mobility office".
- **Fair queue management**: a queue management that ensures that he oldest request coming from a *costumer* must be the first managed in his zone and the request must be forwarded to the taxi that is free and has been waiting in that zone for more time.
- **Guest request**: a request made by a *guest user*.
- **Meeting address**: the address where the taxi must be at the *meeting time*, according to the reservation.
- **Meeting time**: the time at which the taxi must be on the *meeting place*, according to the reservation.
- **Personal reserved area**: a set of pages reachable after authentication where users have access to personal data, information, notifications and specific functions.
- **Request**: a request to take a taxi as soon as possible.
- **Reservation time**: the time at which the reservation is made.
- **Reservation**: a reservation of a taxi for a time that is more than two hours in the future.
- **User**: a person who uses the system.
  - **Guest user**: a *customer* who uses the system without logging in.
  - **Registered user**: either a *registered costumer* or an *employee*.

1.6.2 Acronyms

RASD: *Requirements Analysis Specifications Document*.
P.R.A.: *Personal Reserved Area*.

1.6.3 Abbreviations

Employee for *Employee of the "Mobility office"*.
[Gn] for *the n-th goal*.
[Rn.m] for *the m-th requirement whose aim is to lead to the n-th goal.*

1.6.4 Formatting conventions

- Words in italic are words whose meaning is explained in section 1.6.1 (Definitions).

1.7 Reference Documents

- *Specification Document: MyTaxiService Project AA 2015-2016.pdf.*
- *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.*
- *IEEE Std 1016^{tm}-2009 Standard for Information Technology - System Design - Software Design Descriptions.*

1.8 Document overview

This document is essentially structured in four part:

- **Section 1: Introduction**, it gives a description of the document and some basic information about software.
- **Section 2: Overall Description**, gives general information about the software product with more focus about constraints and assumptions.
- **Section 3: Specific Requirements**, this part list requirements, typical scenarios and use cases. To give an easy way to understand all functionality of this software, this section is filled with UML diagrams.
- **Section 4: Appendix**, this part contains some information about the attached .als file and some described screenshot of software used to generate it.

2. Overall Description

2.1 Product perspective

We well release a web site, an Android application and an iOs app. All these applications have not to be integrated with other existing system. They will not have any internal interface for administration but it will be only user based.
The application will provide new interface or API for integration with future project such as shared taxi.

2.2 User characteristics

A *costumer* who uses the applications is a person who wants an easy way to take or reserve a taxi. He could be interested either in taking use of the service as fast as possible or in having customized functionalities, such as the possibility to view the list of all his own reservation and to delete them. So we will provide both the possibility of reserve/request the taxi with or without logging in the system. Another kind of user is the taxi driver who need to be notified when a *costumer* is waiting for his taxi and to be lead to the *costumer*'s position by a GPS navigator system. This app is also a new way for taxi drivers to accept or decline the request of user in a few taps. *Employees* just need an easy way to register taxi drivers. All kind of users must be able to use a web browser or a mobile application.

2.3 Constraints

- *2.3.1 Regulatory policies*
  MyTaxi doesn't have to meet any regulatory policies.

- *2.3.2 Hardware limitations*
  MyTaxi doesn't have to meet any hardware limitations.

- *2.3.3 Interfaces to other applications*
  MyTaxi doesn't have to meet any interfaces to other applications.

- *2.3.4 Parallel operation*
  MyTaxi must support parallel operations from different users (both taxi drivers and passengers) when working with database.

- *2.3.5 Documents related*
  - Requirements and Analysis Specification Document (RASD).
  - Design Document (DD).
  - User's Manual.
  - Testing report.

2.4 Assumptions and Dependencies

2.4.1 Assumption

- Pre-existing system are owned by private societies, so it isn't possible to modify them.
- There is no need of a hierarchy of user to guarantee the safety of the system.
- A *customer* can reserve as many taxis as he wants.
- The user must have installed the app.
- Through the app the user that has been registered can manage his reservation
- Users and taxi have the same app but for login two different sections.
- Notification of the place of the call, for the taxi, and the code of the taxi that accept the ride, for the user, will be shown.
- We assume it is better to allow deletions of reservation even less than two hour before, because it is better to inform the taxi driver than not be at the appointment.

2.5 Possible future implementations

- Possibility to reserve by phoning to a call center.
- Taxi sharing: Possibility for the *costumer* to decide to share a taxi with others if possible, thus sharing the cost of the ride. In this case the user is required to specify the destination of all rides which he/she wants to share with others. If others are willing to start a shared ride from the same zone going in the same direction, then the system arranges the route for the taxi driver, defines the fee for all persons sharing the taxi and informs the passengers and the taxi driver.

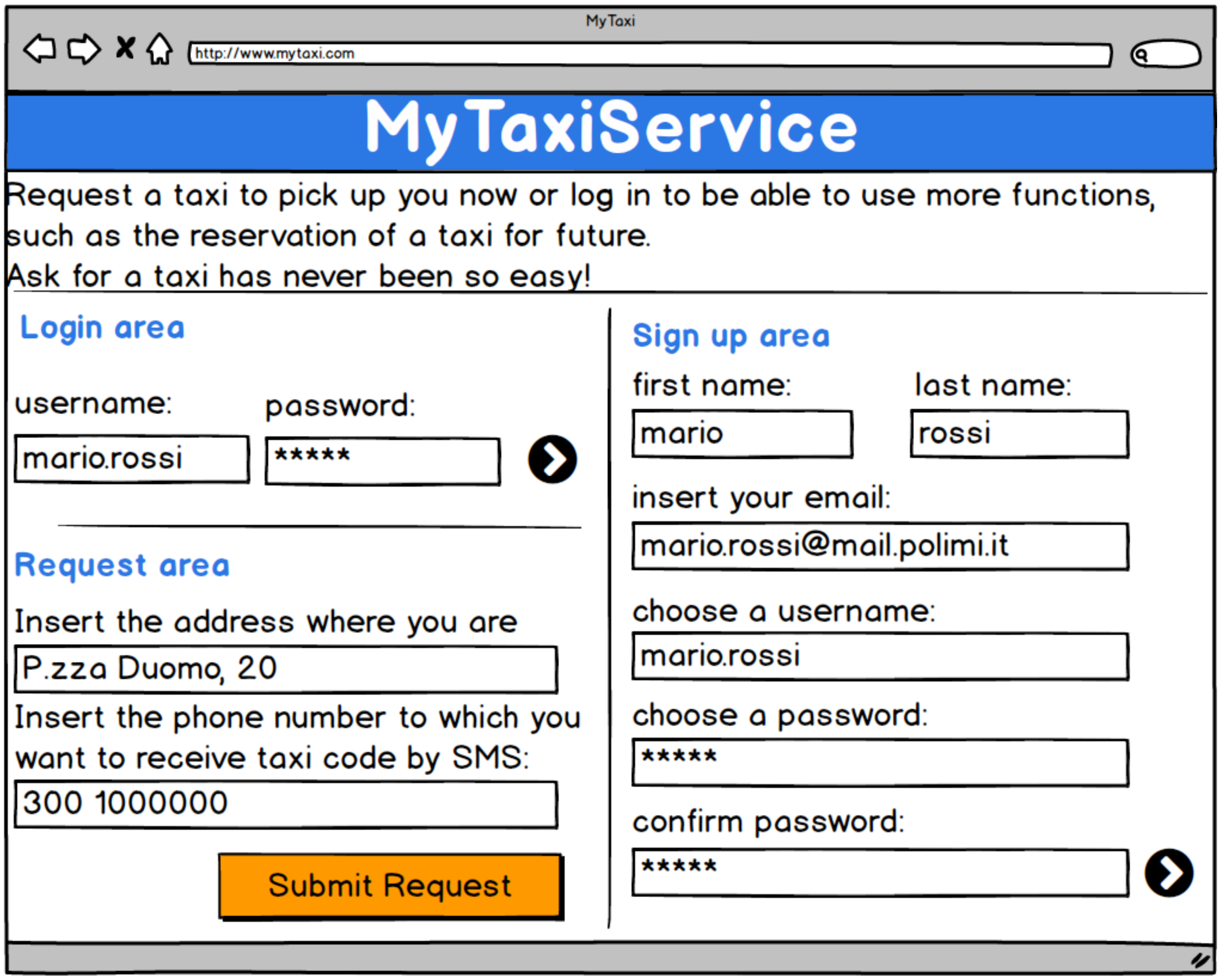
- In case of too many abuses, stakeholders could decide to remove the possibility for *costumers* to send request without signing in.

3. Specific requirement

3.1 External Interface Requirements

3.1.1 User Interfaces

1. ***Site - Home page***



Request a taxi to pick up you now or log in to be able to use more functions, such as the reservation of a taxi for future.
Ask for a taxi has never been so easy!

Login area

username: mario.rossi
password: *****

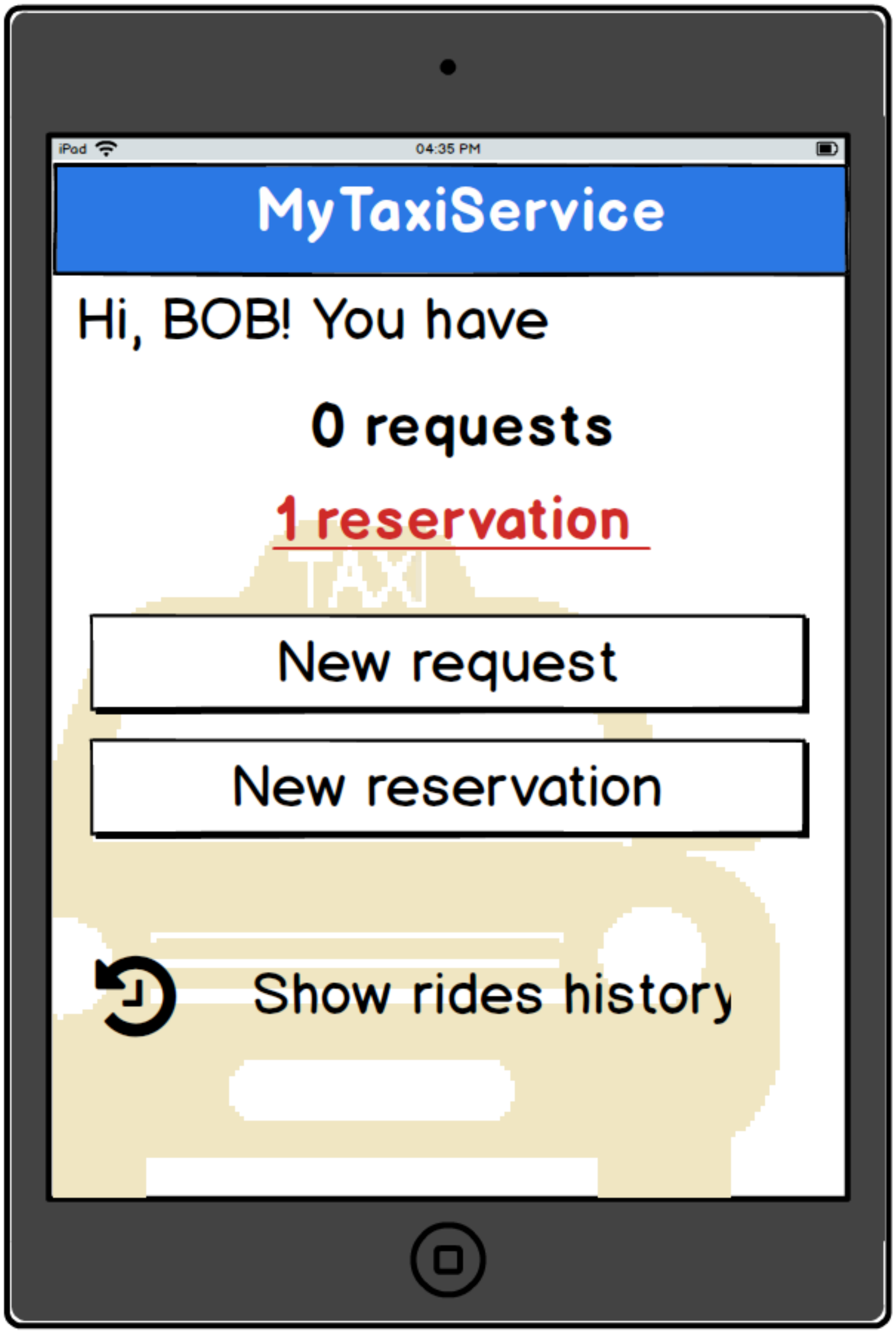
Request area

Insert the address where you are
P.zza Duomo, 20
Insert the phone number to which you want to receive taxi code by SMS:
300 1000000

Sign up area

first name: mario
last name: rossi
insert your email: mario.rossi@mail.polimi.it
choose a username: mario.rossi
choose a password: *****
confirm password: *****

2. *App - Registered costumer area - Home page*

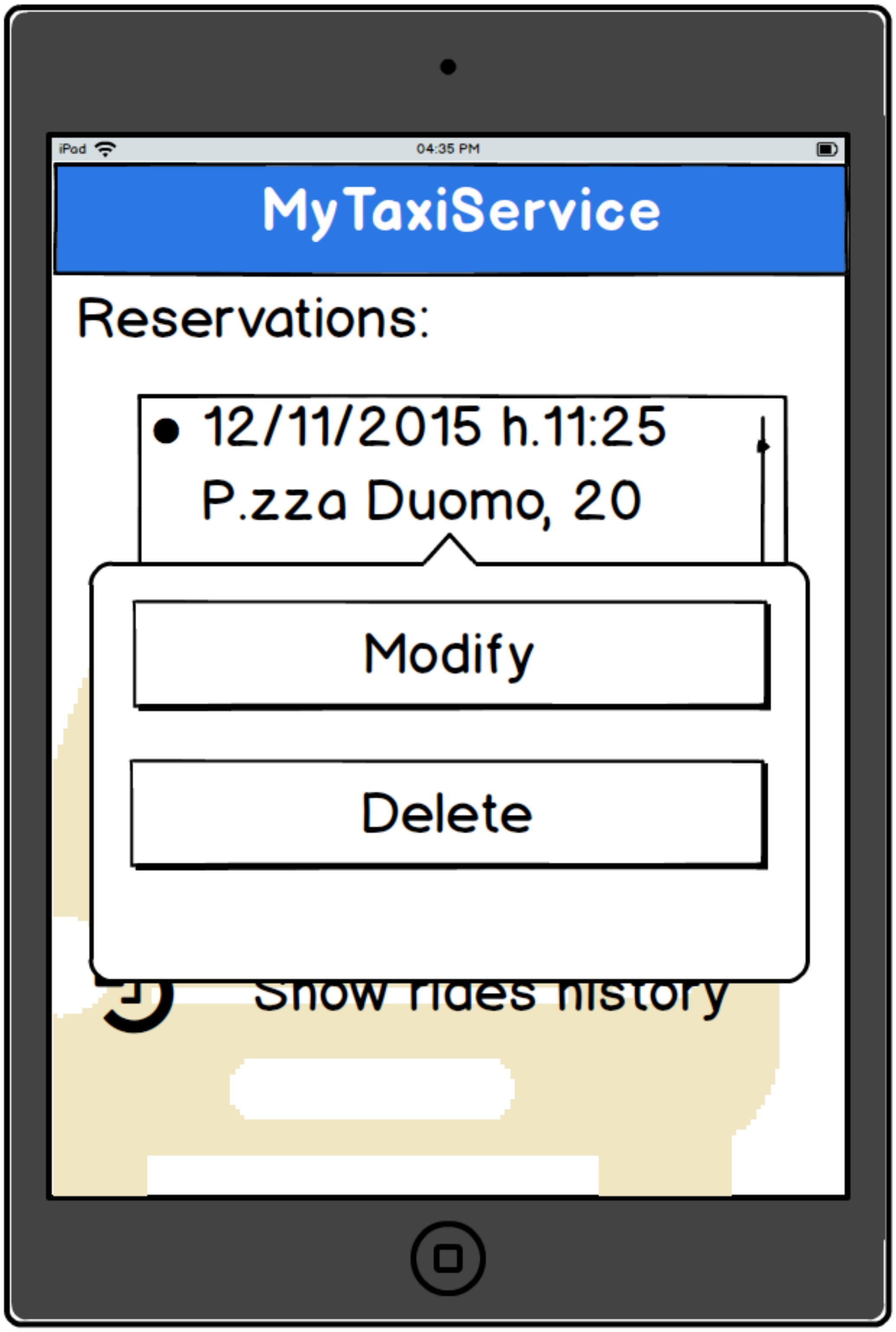


Hi, BOB! You have

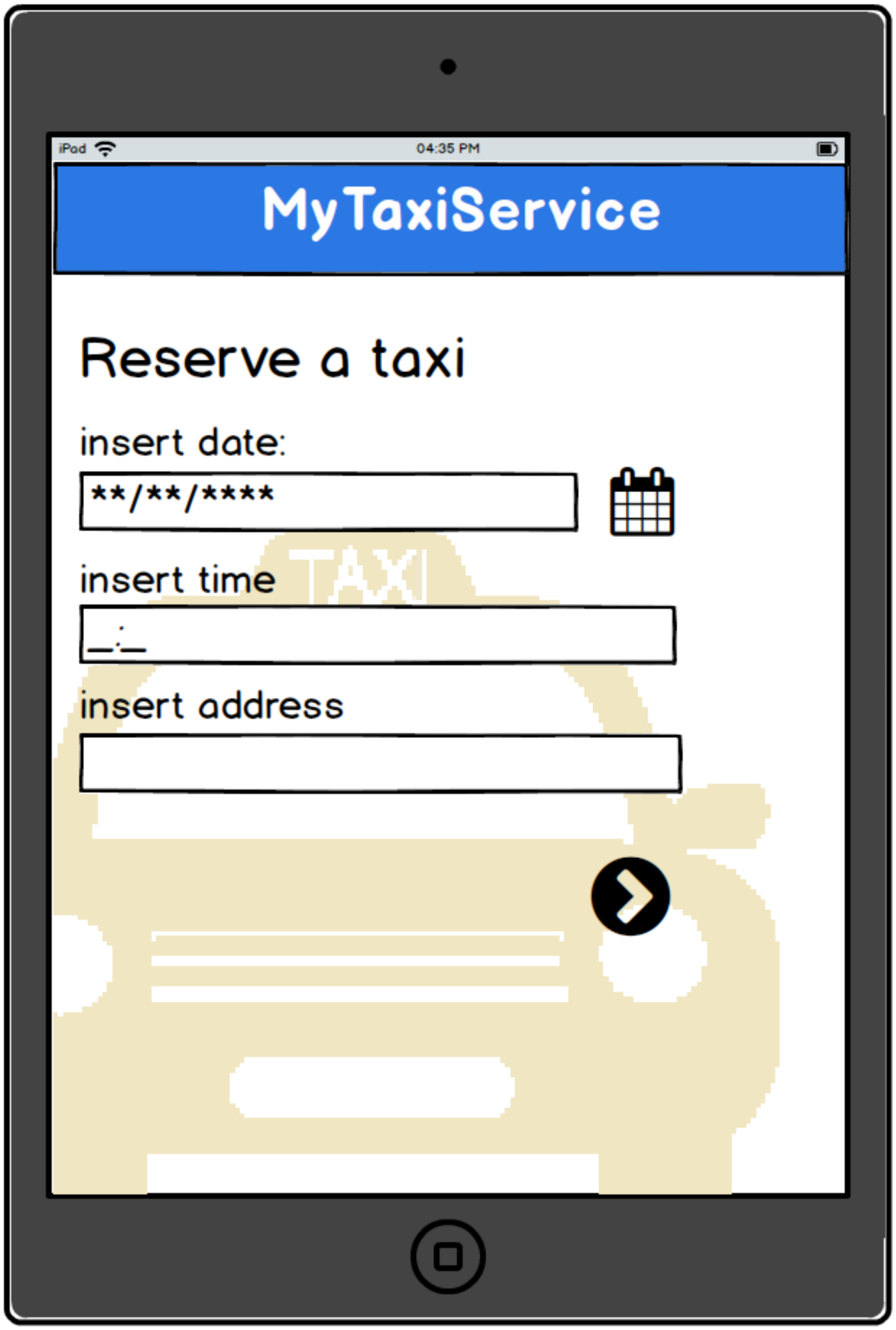
0 requests

1 reservation

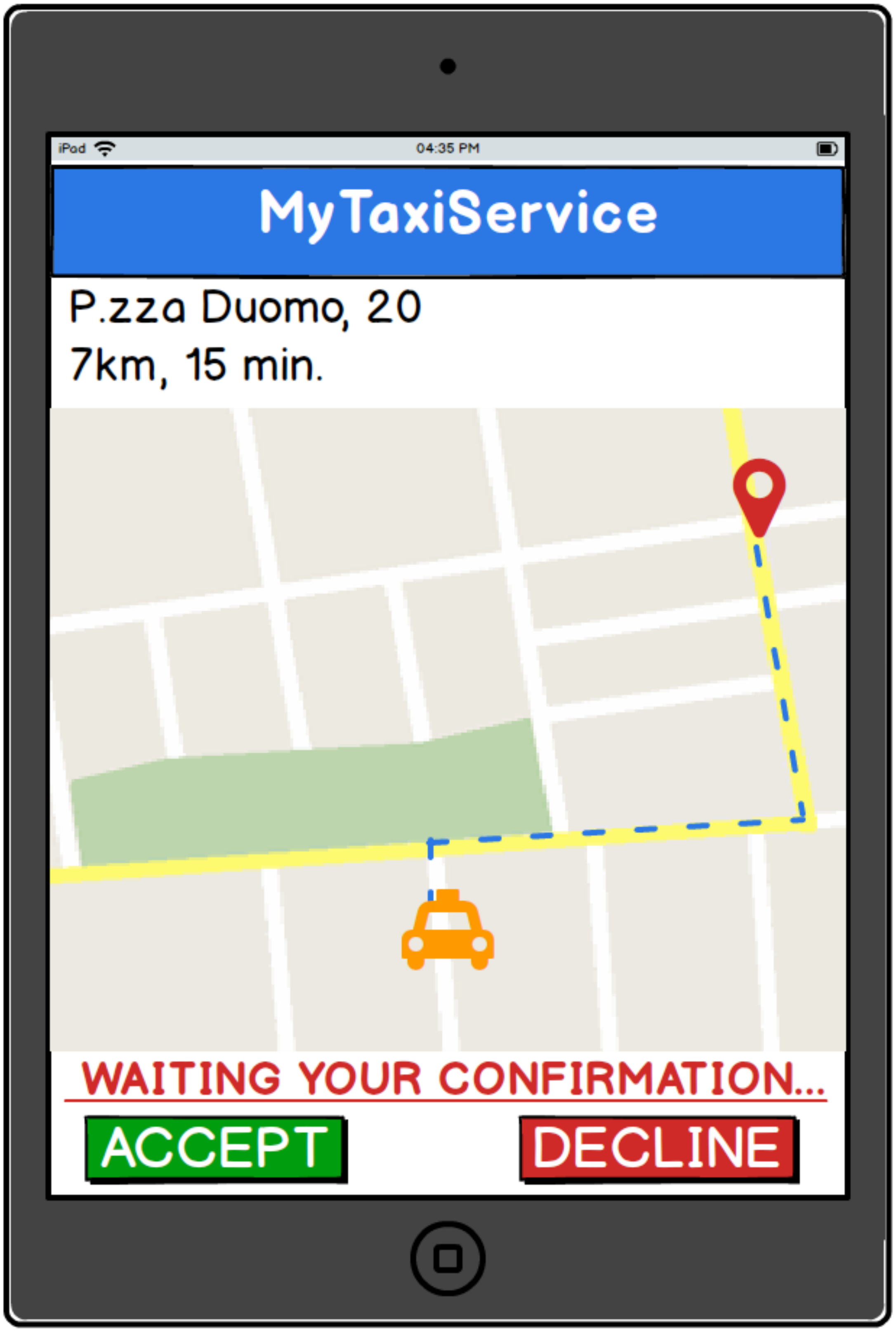
3. App - Registered costumer area - Reservations list page



4. *App - Registered costumer area - New reservation page*



5. App - Taxi driver area - Notification page



3.1.2 API interfaces
MyTaxi must use Google Maps API (http://developers.google.com/maps) to show city maps and check for addresses correctness.

3.1.3 Hardware interfaces
This project does not support any hardware interfaces.

3.1.4 Software interfaces
- Database Management System (DBMS):
  - Name: MySQL.
  - Version: 5.6.21
  - Source: http://www.mysql.it/
- Java Virtual Machine (JVM).
  - Name: JEE
  - Version: 7
  - Source: http://www.oracle.com/technetwork/java/javaee/tech/index.html
- Application server:
  - Name: Glassfish.
  - Version: 4.1.
  - Source: https://glassfish.java.net/
- Operating System (OS).
  - Web application must be able to run on any SO which supports JVM specified before.
  - Mobile app must be able to run on:
    - Android v4.0 or later
    - iOS v.5 or later

3.2 Functional requirements

3.2.1 - [G1] Allow taxi drivers to have a personal reserved area that helps them managing rides.
- [R1.1] *Employee* using terminals of mobility office can register taxi drivers who come to the office with their documents (ID, driving licence, taxi licence). *Employee* has not a personal profile, all terminals of the mobility office are allowed to register taxi drivers.
- [R1.2] The system gives taxi driver a username (taxi ID) and a random password.
- [R1.3] Taxi drivers can login using the app on a mobile device having GPS function.
- [R1.4] All the information and notifications relating to a specific taxi driver and all the functions he can use are available only by that specific taxi driver in his P.R.A.
- [R1.5] In his P.R.A. taxi driver can set his actual state as "available" or "unavailable".
- [R1.6] When the system assigns a ride to a specific taxi driver he is notified in his P.R.A. He can either accept or decline to manage the ride.
- [R1.7] A map of the city is shown in the taxi driver's P.R.A. where he can see his position and the shortest way to the destination address.
- [R1.8] Taxi driver must confirm *costumer*'s presence when he reaches the meeting address.

- [R1.9] *Taxi driver* username is taxi code.

3.2.2 - [G2] Allow guest users to request a taxi to reach him to a specific address as soon as possible.

- [R2.1] Guest users can insert an address and a mobile phone number and ask for a taxi coming as soon as possible to there.
- [R2.2] If enough taxis are available, the system assigns one of them to each incoming guest request.
- [R2.3] The code of the taxi assigned to a guest request is sent by SMS to the number the *costumer* provided during the request.

3.2.3 - [G3] Allow costumer to have a personal reserved area. In addition to functionalities provided to guest users, a registered costumer can reserve a taxi for a specific time (at least two hours in advance), check previous requests and reservations and modify or delete future ones.

- [R3.1] Guest users can sign up the system providing his first and last names and an email. He must choose a username not already assigned to another user and a password. All these four pieces of information are mandatory to complete the registration.
- [R3.2] Guest users who have access credential can login. A user can login only once per session.
- [R3.3] In their P.R.A. registered users can insert an address and ask for a taxi coming as soon as possible to there.
- [R3.4] The code of the taxi assigned to a *registered costumer* request is notified in the P.R.A. of the *registered costumer* who made the request.
- [R3.5] In their P.R.A. registered users can insert a date, a time and an address to make a reservation for a taxi. It must be avoided to make a reservation for a time that is less than 2 hours after the reservation time.
- [R3.6] The code of the taxi assigned to a *registered costumer* request is notified in his P.R.A.
- [R3.7] If enough taxis are available, the system assigns one of them to each incoming *registered costumer* request and to each incoming reservation.
- [R3.8] In his P.R.A. a registered user can see a list of all his old requests and reservations and a list of his future ones.
- [R3.9]In any moment, a registered user can decide to modify the meeting address of a future reservation or delete it. Deleting process is not reversible, all reservation data will be lost.
- [D1] Email address used for registration must be formally correct.

3.2.4 - [G4] Guarantee a fair queue management.

- [R4.1] The city area is divided in some zones. The system perfectly knows the boundaries of each zone.
- [R4.2] For each zone there is a queue of requests and reservations.

- [R4.3] Requests must be ordered chronologically so that the oldest request coming from a *costumer* must be the first managed in his zone.
- [R4.4] 10 minutes before meeting time, reservation is inserted between the backmost reservation in the queue and the oldest request.
- [R4.5] For each zone there is a queue of taxis where the elements are ordered according to the time they have been waiting in that zone. The request must be forwarded to the taxi that is free and has been waiting in that zone for more time.
- [R4.6] The position of each taxi is automatically obtained by GPS whether his state is set as available.
- [R4.7] Whenever a taxi driver set his state as unavailable he his deleted from the queue his taxi belongs.
- [R4.8] Whenever a taxi driver set his state as available he is added at the back of the queue to which his position belongs.
- [R4.9] Whenever a taxi driver moves from a zone to another his taxi is deleted from the starting zone's queue and added at the back of the destination zone's one.
- [R4.10] Whenever a taxi driver accepts a ride, his state is automatically set as unavailable.
- [R4.11] Whenever a taxi driver declines a ride, he is automatically moved to the back of the queue.
- [R.4.12] If taxi driver communicates that the *costumer* is not present, he is moved to the front of the queue.

3.2.5 - [G5] Manage the distribution of the taxis in each zone and notify taxi drivers the need to move from a zone to another.

- [R5.1] For each zone the system computes the number of taxis in that zone and the number of requests and reservations managed during the last hour.
- [R5.2] Unless the number of available taxis in the city is not enough to guarantee more than 2 taxis per zone, if there are less than 2 taxis in a zone queue, a random taxi driver in one of the adjacent zones is notified.
- [R5.3] Unless the number of available taxis in the city is not enough to ensures that the number of taxis in a certain zone is be greater than the number of requests and reservations during the last hour, some random taxi drivers in the adjacent zones are asked to move from their zone to the one where the lack has been detected.

3.3 Scenarios

3.3.1 Scenario 1

Raffaella is an Italian singer, dancer, television presenter, and actress and is late for her show in Piazza Duomo. All her fans are waiting for her.
Raffaella is in her favourite bar in Milan "Bar Virgilio" and she has just finished having dinner; while she is having a coffee, she opens from her iPhone the app MyTaxi and requests a taxi, inserting the address where she wants to be picked up (Via Vincenzo Monti, 22). The first taxi in the queue is notified and the driver accepts to take care of the ride. In few seconds the code of the taxi is showed on her smartphone. The taxi came after a minute and takes Raffaella to her screaming fans.

3.3.2 Scenario 2

Sergio, Luca, Vito and Antonio are friends that want to celebrate Vito's graduation in Computer science and engineering.
Before going to Alcatraz Milano, they decide to use MyTaxi app to reserve a taxi for coming back home at 6 a.m., since Milan underground closes at 00.30 and Sergio is already registered for this service. Sergio logs in and taps on the reservation button, then insert Alcatraz address (via Valtellina, 3) and the time they want to be picked up. It's 5.50 a.m. and Gigi, the first taxi driver in the queue order, is at the end of his work shift and receives the notification of a new ride but he is very tired and so he decides to decline and go home for sleeping.
The request is forwarded to the second of the queue, Michele, who accepts and takes the entire friends home.

3.3.3 Scenario 3

Carlo was chosen by Marco and Giulia as witness for their wedding and he reserved a taxi 7 days ago to reach the church. Two days before the meeting, the wedding planner informs Carlo that Giulia founded out that Marco has been cheating her for months. The wedding is cancelled, so Carlo wants to delete his reservation.
He logs in and suddenly sees the link to all his reservations. After tapping on it, he chooses the reservation he wants to delete and complete the operation.

3.3.4 Scenario 4

At home, Gigi, a taxi driver that work from the 1982, receives a letter from Comune di Milano about the new app MyTaxi that all the taxi of the city had to be installed.
The next day, Gigi, goes to the office of the taxi of Milan. Here Francesca, an *employee* of the office, registers Gigi and gives him his Username and password. Gigi starts his work shift installing the MyTaxi app on his device and logging in.

3.3.5 Scenario 5

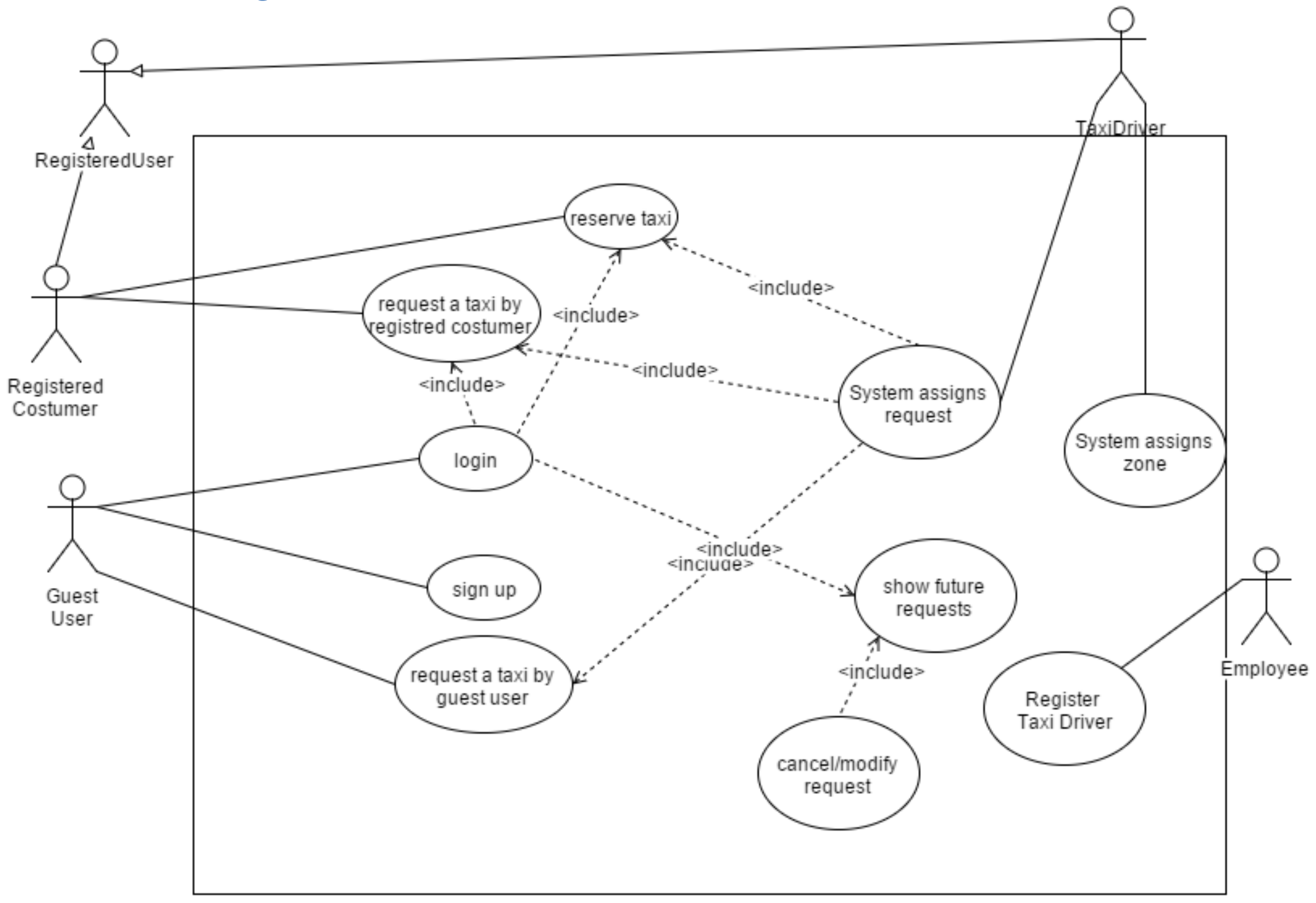
Caterina after the first use of the new MyTaxi want to sign up and enjoy all the features of the application. She begin the process of registration and chose "Cate" as username and then fill all the other mandatory fields. When he clicks the "confirm" button the registration process fails and the application alerts her that the Username "cate" is already used as username by another user.
Caterina changes her Username in "caterina.rossi" than clicks the "confirm" button.
Registration process correctly ends and the new profile is created.
MyTaxi sand an email to confirm registration. Caterina is successfully registered.

3.3.6 Scenario 6

It's the 15th of August in Milan and Luca goes to work with his taxi. The city is desert because all the people are away for "Ferragosto". Using his phone he opens the app MyTaxi, he does the login and wait for a notification for a ride. After 10 minute of silence, the app send a notification to Luca that invite him to go near the zone of Duomo that is the only area of Milan where there are people looking for a taxi. Luca decides to moves there to start working, thinking to all his friends who are at the sea.

3.4 UML models

3.4.1 Use case diagram



3.4.2 Descriptions and sequence diagrams

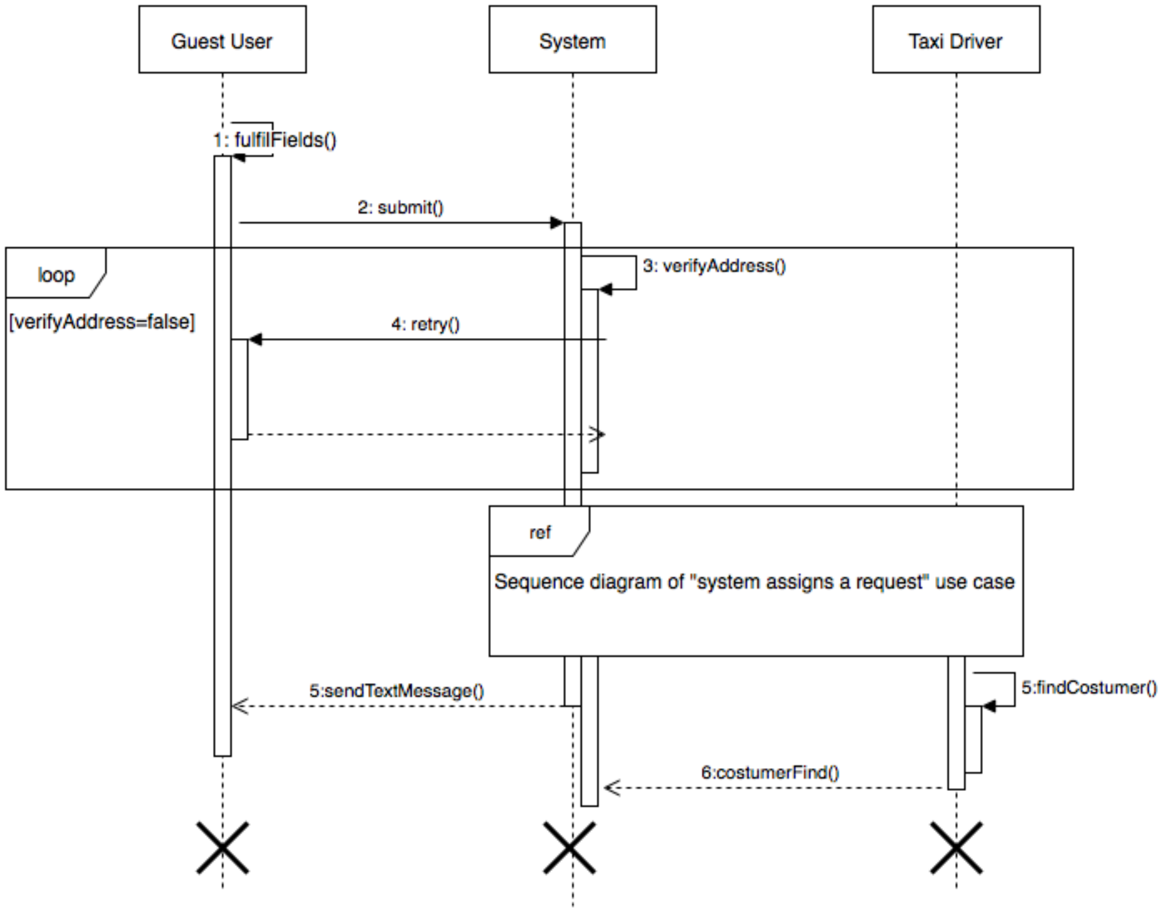
3.4.2.1 Guest user requests a taxi

- *Description*

Name	**Guest user requests a taxi**
Actors	Guest user, taxi driver
Goal	[G2], [G4]
Related requirements	[R2.1], [R2.2], [R2.3], [R4.11]
Basic flow	1. Guest user fulfils the fields “address” and “phone number”; 2. Guest user submits the request; 3. The system invokes the use case “system assigns a request”; 4. The system sends a text message with taxi code to the phone number provided during step 1; 5. The taxi driver get to the given address and confirms the presence of the *costumer*;
Exit conditions	– The taxi has been assigned to the request and the taxi driver accepted; – The taxi state is unavailable.
Exceptions	1.a. Address does not exists: 1.a.1 The app/site shows an error message; 1.a.2 Back to step 1. 5.a. *Costumer* isn’t there: 5.a.1 Taxi driver notify the system that *costumer* is not there; 5.a.2 The system move the taxi driver to the front of the queue.

- ***Sequence diagram***

Guest user requests a taxi



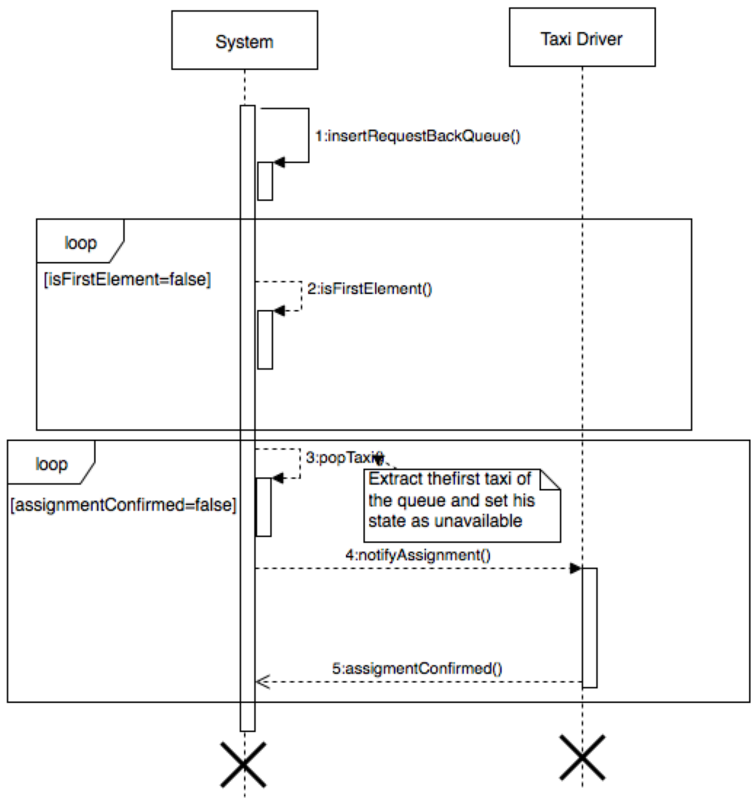
3.4.2.2 Request assignment

- *Description*

Name	**System assigns a request**
Actors	Taxi driver
Goal	[G4]
Related requirements	[R4.2], [R4.3], [R4.9], [R4.10]
Basic flow	1. The system inserts the request as last element of the queue of the zone the address belongs to; 2. When all the previous requests has been managed, the system assigns the first taxi in the queue of the zone to the request and set the taxi state as unavailable; 3. The app notifies taxi driver and he accepts to manage the ride;
Exit conditions	– The taxi has been assigned to the request and the taxi driver accepted; – The taxi state is unavailable.
Exceptions	2.a No taxi is available: 2.a.1 A text message of unmanageable request is sent by the system to the phone number provided during step 1; 3.a Taxi driver declines the request: 3.a.1 The system moves taxi driver to the back of the queue; 3.a.2 Back to step 2.

- ## *Sequence diagram*

System assigns a request


System
Taxi Driver
1:insertRequestBackQueue()
loop
[isFirstElement=false]
2:isFirstElement()
loop
[assignmentConfirmed=false]
3:popTaxi()
Extract thefirst taxi of the queue and set his state as unavailable
4:notifyAssignment()
5:assigmentConfirmed()

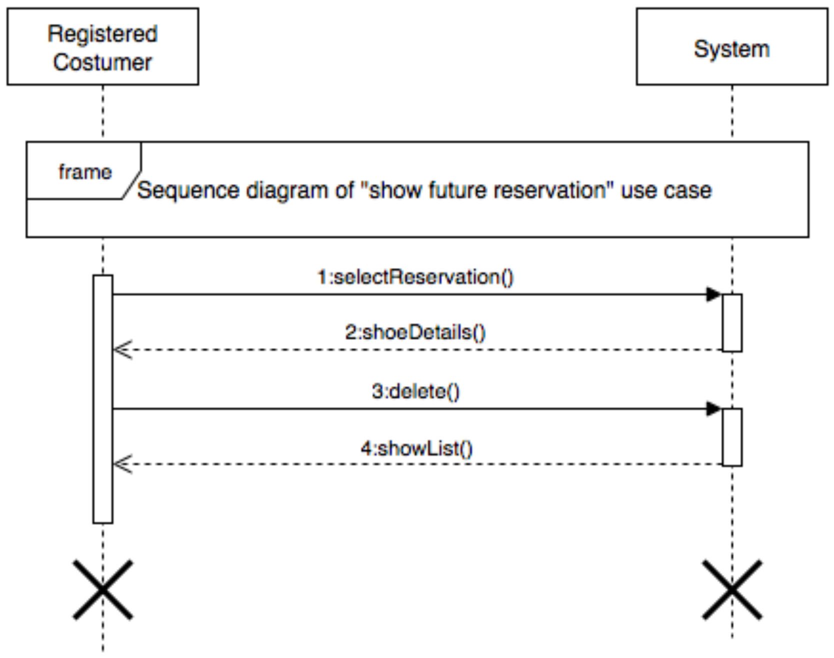
3.4.2.3 Registered costumer wants to cancel a reservation

- *Description*

Name	***Registered costumer*** **wants to cancel a reservation**
Actors	*Registered costumer*
Goal	[G3]
Related requirements	[R3.8], [R3.9]
Basic flow	1. Guest user invokes "Show future requests" use case. 2. The system asks to the user if he wants either modify or delete the selected reservation; 3. The user tap/click on delete. 4. The system deletes all the data about that reservation and shows P.R.A. home page.
Exit conditions	– All the data about the reservation are deleted.
Exceptions	

- *Sequence diagram*

Cancel a reservation



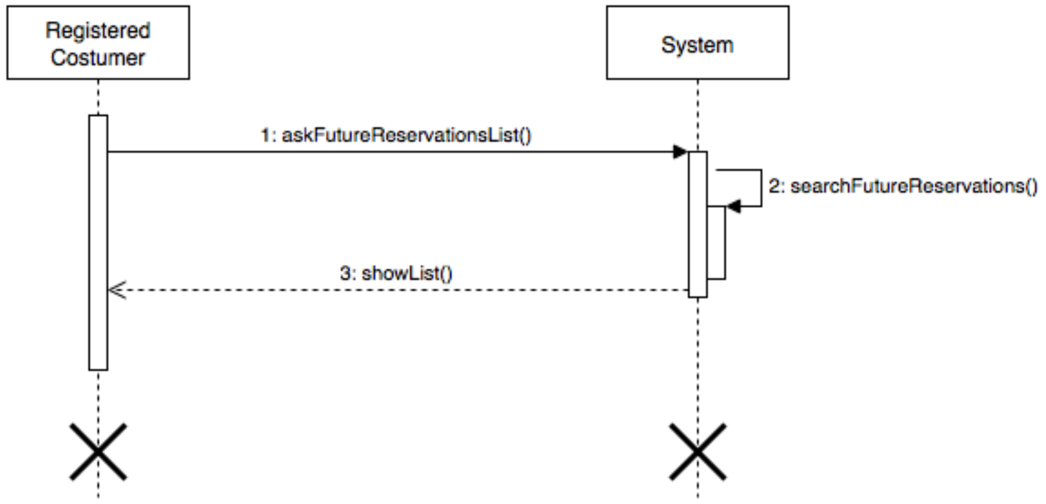
3.4.2.4 Show future requests

- ***Description***

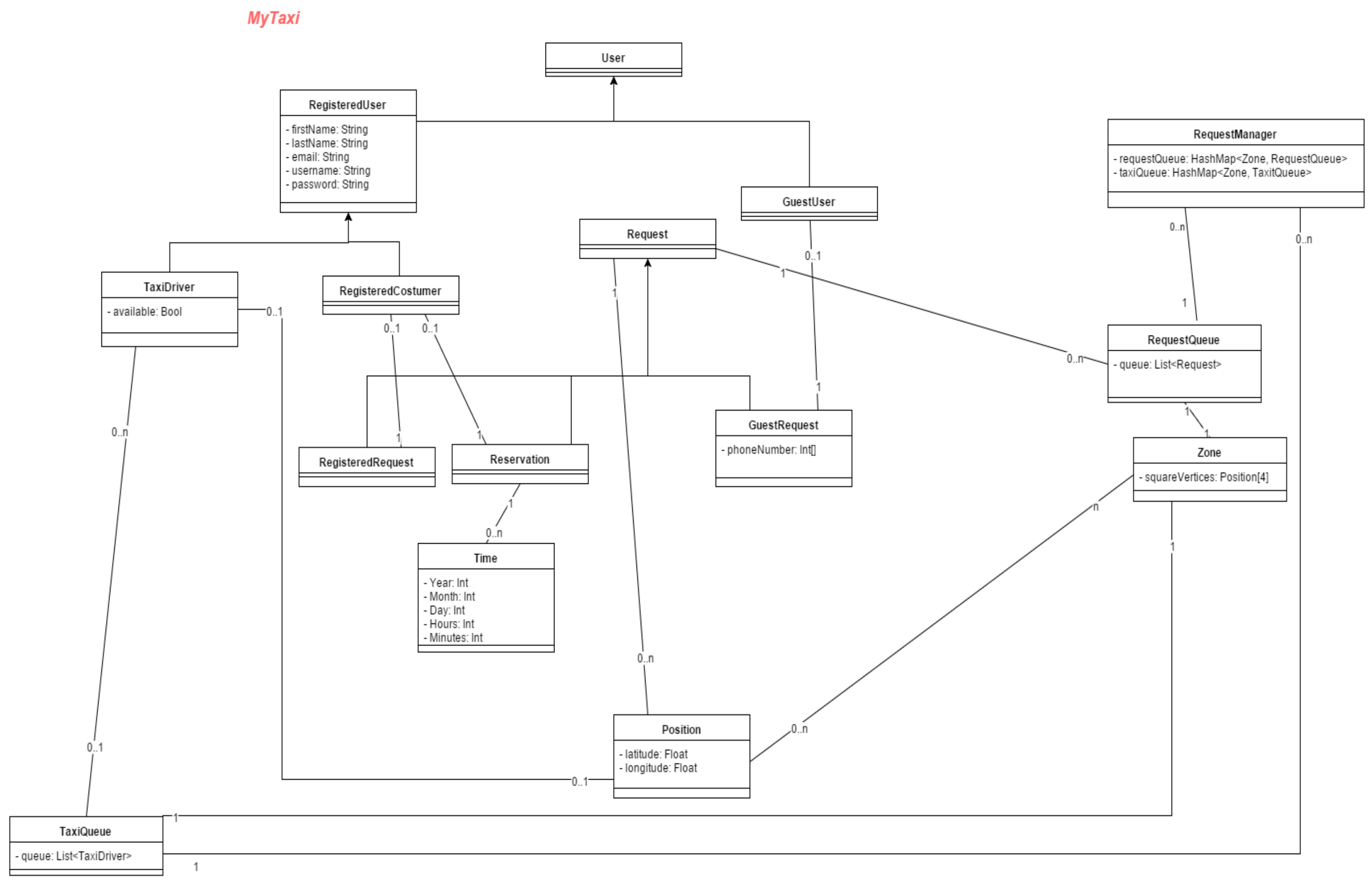
Name	Show future reservations
Actors	*Registered costumer*
Goal	[G3]
Related requirements	[R3.8]
Basic flow	1. The *registered costumer* tap/click on the link to his future reservations; 2. System looks for all future costumer reservations in database 3. The app/site shows the list of all the *registered costumer*'s future reservation and he selects the one he wants to delete;
Exit conditions	– The *Registered costumer* sees the list of all future requests.
Exceptions	3.a The user has no future reservation: 3.a.1 The link is inactive.

- ***Sequence diagram***

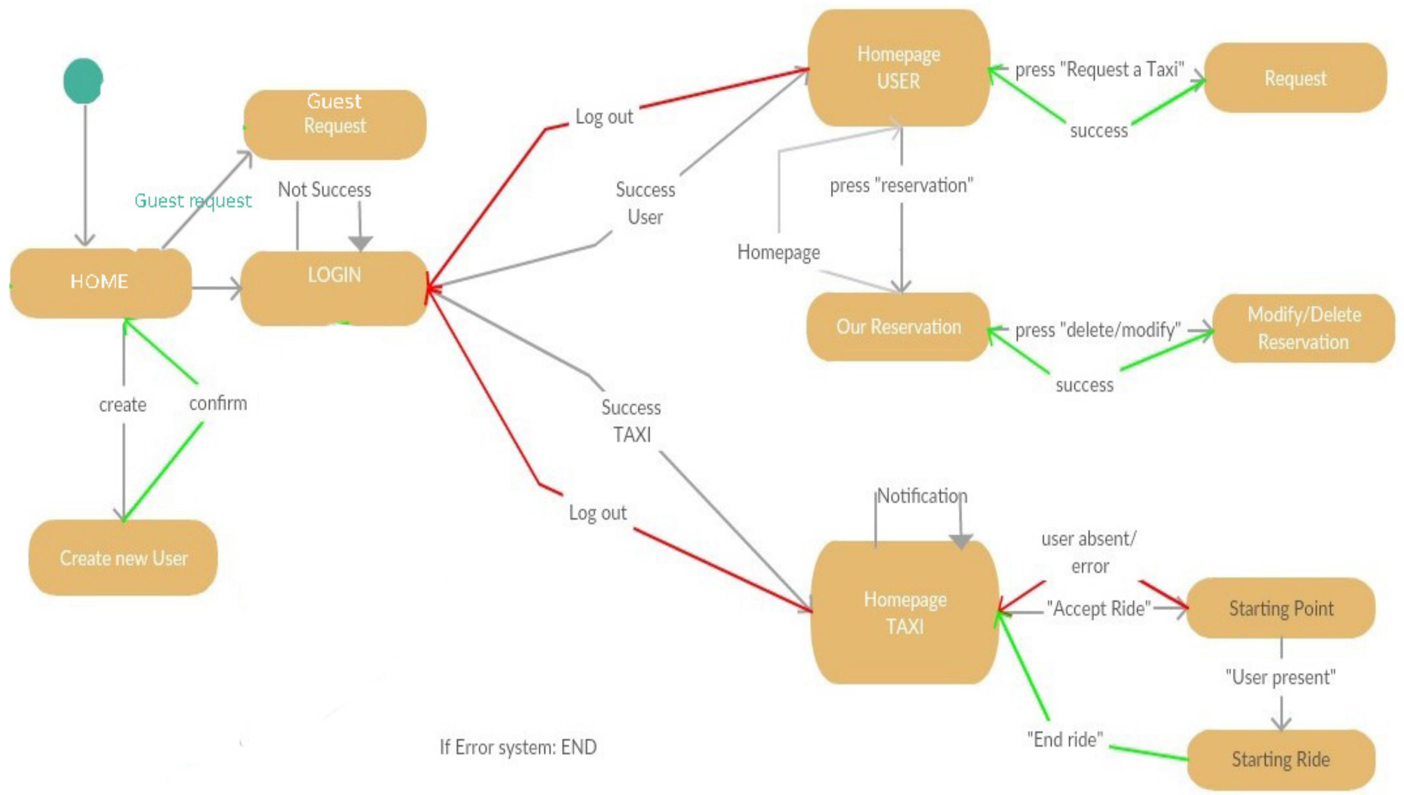
Show future reservations



3.4.4 Class Diagram



3.5.5 State Diagram



3.6 Non-functional requirements

3.6.1 Performance Requirements

We assume the response time of the system is close to zero, so the performance are essentially bounded by users device (Ram, ecc) and by internet connection.

3.6.2 Design Constraints

The application will be developed with Java EE so it will inherit all language's constraints.

3.6.3 Software System Attributes

3.6.3.1 Availability

The application will be accessible online 24/7. We decide to host all system into cloud platform like Amazon Elastic Compute Cloud. This solution gives more scalability to performance required by system and it could reduce the cost for dedicated server, maintaining an high level of performance especially in case of full load with a lot of connected users.

3.6.3.2 Maintainability
The application does not provide any specific API.
All the documentation is available and complete for future implementations and/or problem.

3.6.3.3 Portability
The application could be used on any SO which supports JVM and DBMS.

3.6.3.4 Security
The system must protect all sensible data stored in the database. Personal data will be managed respecting law n. 675 of 12/31/1996 about personal data.

3.7 Constraint
3.7.1 We assume that all taxis are equipped with an iOs or Android Mobile device that have installed the App MyTaxi. This device must have a GPS connectivity, so every time a driver is updating his availability, the system has direct access not only to all the information regarding his vehicle (taxiID) but also to his current position and so to his current zone.

3.7.2 MyTaxi mobile app must be available for all the main operating systems available right now on the market: IOS, android, windows phone (future).

3.7.3 MyTaxi mobile app must not pass the 9MB size threshold.

3.7.4 MyTaxi website app must fit any laptop, tablet or desktop screen and must perform independently from the operating system.

4. Appendix

4.2 Alloy

4.3.1 Alloy usefulness
UML is considered as a standard diagram in the first phase of modelling software system. However UML doesn't give a complete view of the system we're trying to model. One of the most important lacks is the absence of an easy way to show constraints on semantic models. Alloy is a declarative specification language, so it is especially useful to model static properties of components in Software Design specifics. Alloy integrates UML models pointing out the relations between entities and the constraints on these relations.

4.3.2 Alloy models
We created 3 .als files: the first one shows the whole system with all the static constraints, the second and the third ones shows queues and the relation between the two versions of a queue before and after an enqueue or a dequeue are applied.

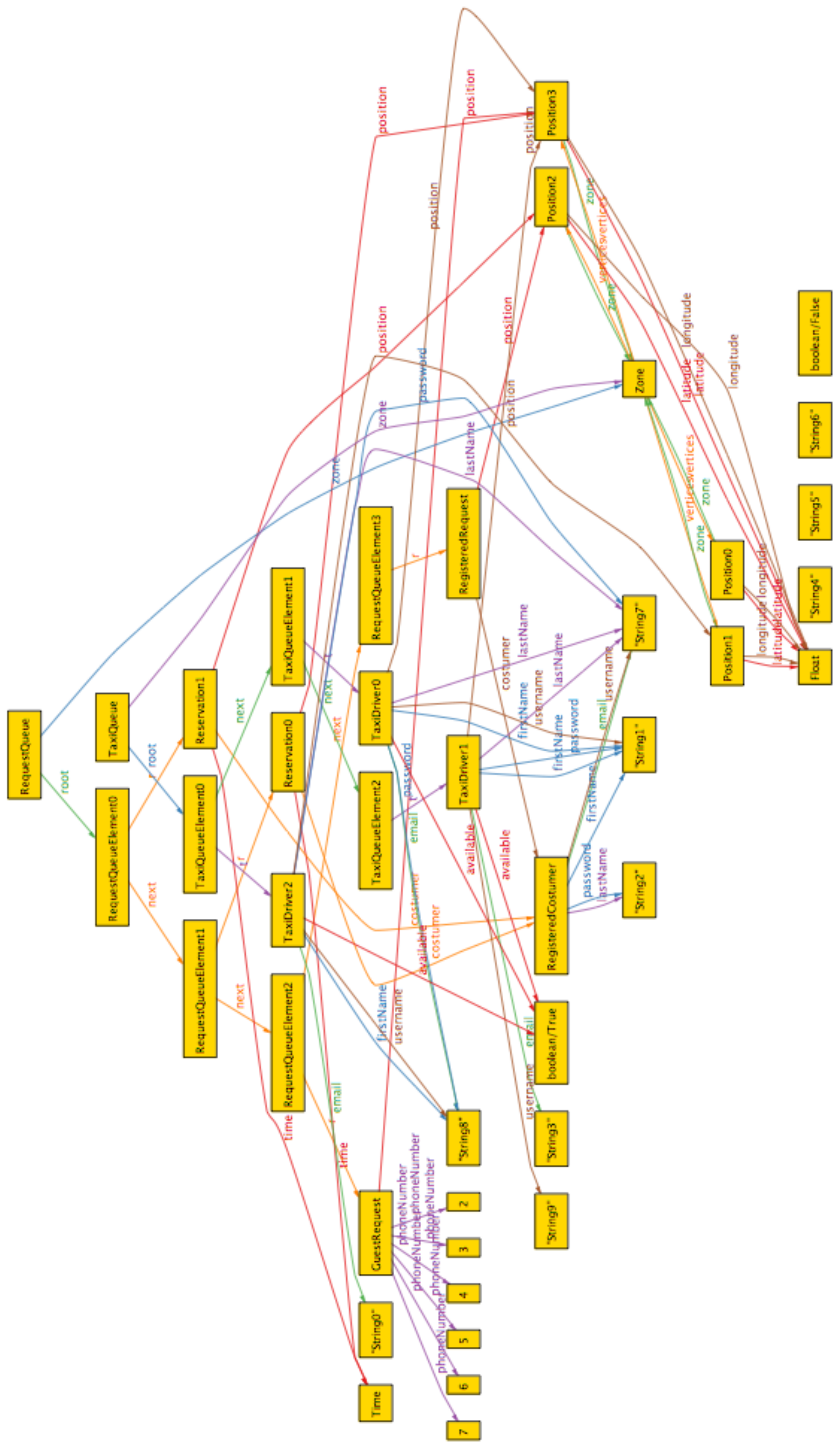
4.3.2.1 Whole system

- ***Code:***

```
 //UTILITIES
open util/boolean
sig Float {}

// SIGNATURES

//Type of users
abstract sig User{
}

abstract sig RegisteredUser{
firstName: String,
lastName: String,
email: String,
username: String,
password: String
}

sig RegisteredCostumer extends RegisteredUser{}

sig TaxiDriver extends RegisteredUser {
position: Position,
available: Bool
}

sig GuestUser extends User{}

//zone
sig Zone {
vertices: set Position
} {#vertices = 4}

//position
sig Position{
longitude: Float,
latitude: Float,
zone: Zone
}

//time
sig Time{/*
year:Int,
month: Int,
day: Int,
hours: Int,
mins: Int*/
}
//{year>0 and month>0 and month<=12 and day>0 and day<=31 and hours>=0 and hours<=23 and mins>0 and
mins<60}
//solver cannot find instances with this right restriction so we commented it

//type of requests
```

```
abstract sig Request{
position:Position
}

sig RegisteredRequest extends Request{
costumer:RegisteredCostumer
}

sig GuestRequest extends Request{
phoneNumber: some Int
} {#phoneNumber = 6 and (all x:phoneNumber | x > 0)} //#phoneNumber should be 10 but solver cannot find
instances

sig Reservation extends Request{
costumer:RegisteredCostumer,
time:Time
}

//queue signatures
sig TaxiQueueElement {
t: TaxiDriver,
next: lone TaxiQueueElement
}

sig TaxiQueue{
zone: Zone,
root: TaxiQueueElement
}

sig RequestQueueElement {
r: Request,
next: lone RequestQueueElement
}

sig RequestQueue{
zone: Zone,
root: RequestQueueElement
}

//FACTS
fact allElementsBelongToSomeQueue {
                    all e:TaxiQueueElement | one q:TaxiQueue | e in q.root.*next and
                    all e:RequestQueueElement | one q:RequestQueue | e in q.root.*next
          }

fact nextNotCyclic {
          no e:TaxiQueueElement | e in e.^next
          no e:RequestQueueElement | e in e.^next
}

fact eachTaxiInOnlyOneQueue{
          (no disj e1, e2:TaxiQueueElement | e1 in TaxiQueue.root.*next and e2 in TaxiQueue.root.*next and
e1.t=e2.t)
}

fact eachRequestInOnlyOneQueue{
        (no disj e1, e2:RequestQueueElement | e1 in RequestQueue.root.*next and e2 in
RequestQueue.root.*next and e1.r=e2.r)
}

fact eachRequestInRightZoneQueue{
        (all q:RequestQueue, r1:RequestQueueElement | r1 in q.root.*next implies r1.r.position.zone = q.zone)
}

fact ReservationsHasHigherPriorityThanRequests{
        (all q:RequestQueue, e:RequestQueueElement, r1:Request | (e in q.root.*next and r1 in e.r and not(r1
in Reservation)) implies (no r2:Reservation | r2 in e.next.r))
}

fact ifTaxiInQueueThenAvailable {no t1:TaxiDriver | t1 in TaxiQueueElement.t and t1.available = False}

fact ifTaxiNotInQueueThenUnavailable  {no t1:TaxiDriver | not(t1 in TaxiQueueElement.t) and t1.available =
True}

fact aQueuePerZoneAndAZonePerQueue{
        (all z: Zone | one q: TaxiQueue | q.zone = z) and (all q: TaxiQueue | one z: Zone | q.zone = z) and
        (all z: Zone | one q: RequestQueue | q.zone = z) and (all q: RequestQueue | one z: Zone | q.zone = z)
}

fact noMultipleRegistrationForSameEmail { no disj ru1, ru2:RegisteredUser | (ru1.email = ru2.email)}

fact univoqueUsername { no disj ru1, ru2:RegisteredUser | (ru1.username = ru2.username)}

//PREDICATES

pred show(){
#TaxiQueue>0
all q:TaxiQueue | #(q.root.*(this/TaxiQueueElement <: next))>1
#GuestRequest>0
#Reservation>1
}

run show for 4 but exactly 10 String
```

- ***run show for 4 but exactly 10 String:***



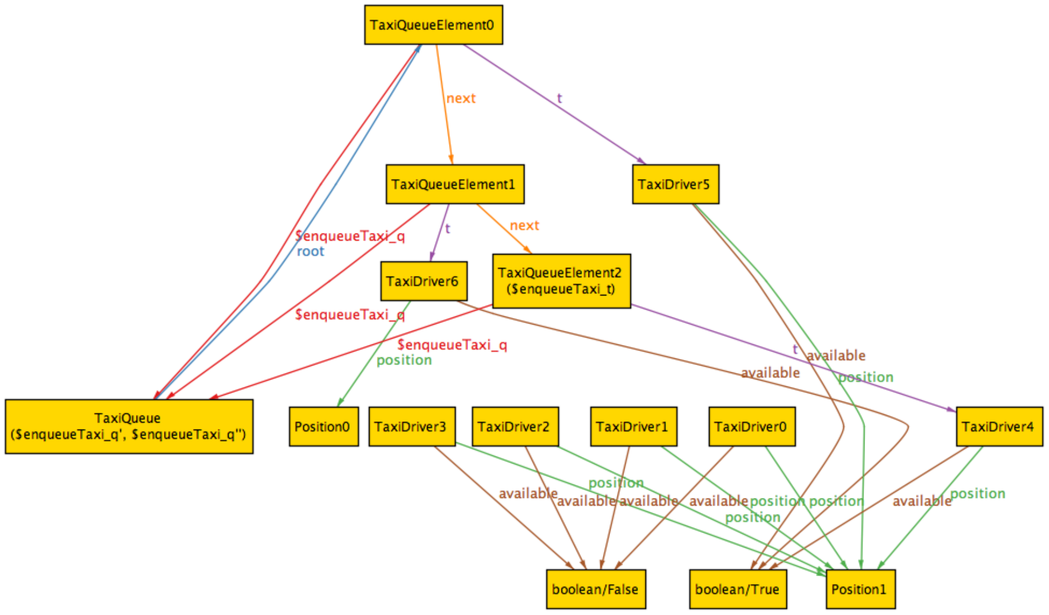
4.3.2.2 Taxi queue

- *Code:*

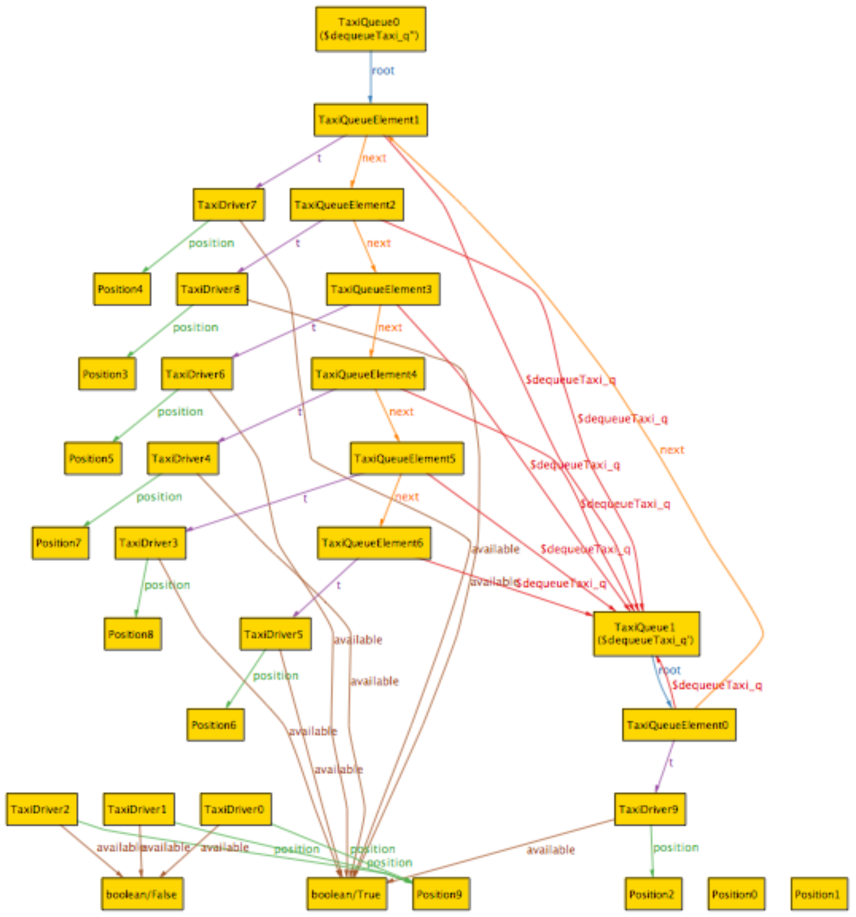
```
open util/boolean

abstract sig User{
}

abstract sig RegisteredUser{/*
firstName: String,
lastName: String,
email: String,
username: String,
password: String*/
}

sig Position{
}

sig TaxiDriver extends RegisteredUser {
position: Position,
available: Bool
}

sig TaxiQueueElement {
t: TaxiDriver,
next: lone TaxiQueueElement
}

sig TaxiQueue{
root: TaxiQueueElement
}

fact allElementsBelongToSomeQueue {
                    all e:TaxiQueueElement | some q:TaxiQueue | e in q.root.*next
          }

fact nextNotCyclic {
          no e:TaxiQueueElement | e in e.^next
}

fact eachTaxiInOnlyOneQueue{
          (no disj e1, e2:TaxiQueueElement | e1 in TaxiQueue.root.*next and e2 in
TaxiQueue.root.*next and e1.t=e2.t)
}
```

```
fact ifTaxiInQueueThenAvailable {no t1:TaxiDriver | t1 in TaxiQueueElement.t and t1.available =
False}

fact ifTaxiNotInQueueThenUnavailable  {no t1:TaxiDriver | not(t1 in TaxiQueueElement.t) and
t1.available = True}

//fact noMultipleRegistrationForSameEmail { no disj ru1, ru2:RegisteredUser | (ru1.email =
ru2.email)}

//fact univoqueUsername { no disj ru1, ru2:RegisteredUser | (ru1.username = ru2.username)}

pred enqueueTaxi[t: one TaxiQueueElement, q, q': one TaxiQueue] {
	q.root = q'.root and (one e:TaxiQueueElement | e in q'.root.*next and e.next = t) and
#t.next=0
}

pred dequeueTaxi[q, q': one TaxiQueue] {
	q'.root = q.root.next
}
```

- *run enqueueTaxi for 10 but exactly 1 TaxiQueue:*



- *run dequeueTaxi for 10 but exactly 2 TaxiQueue:*

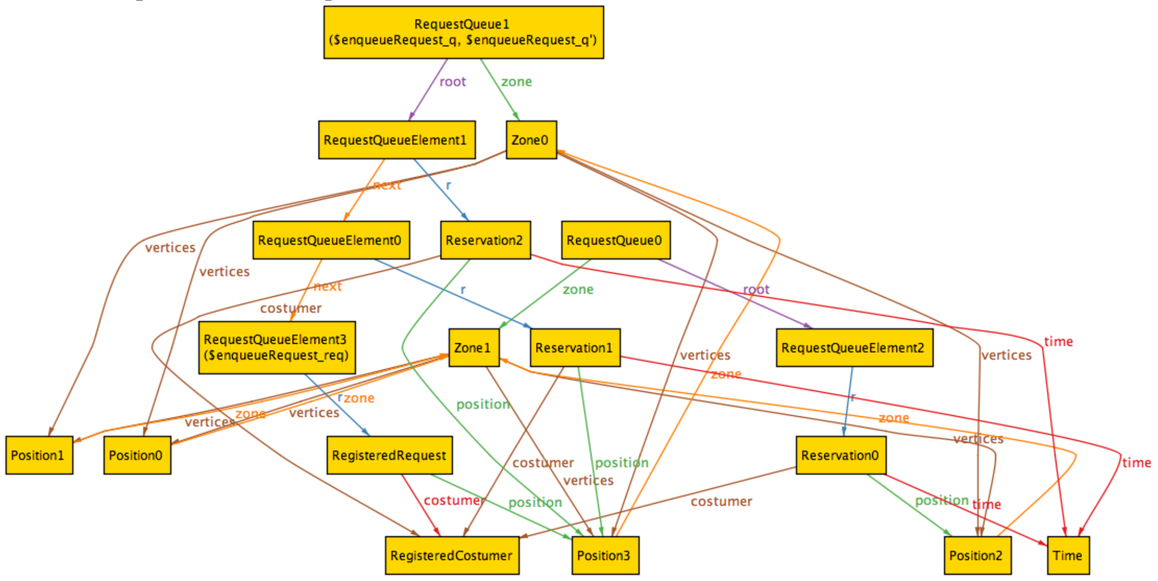


4.3.2.2 Request queue

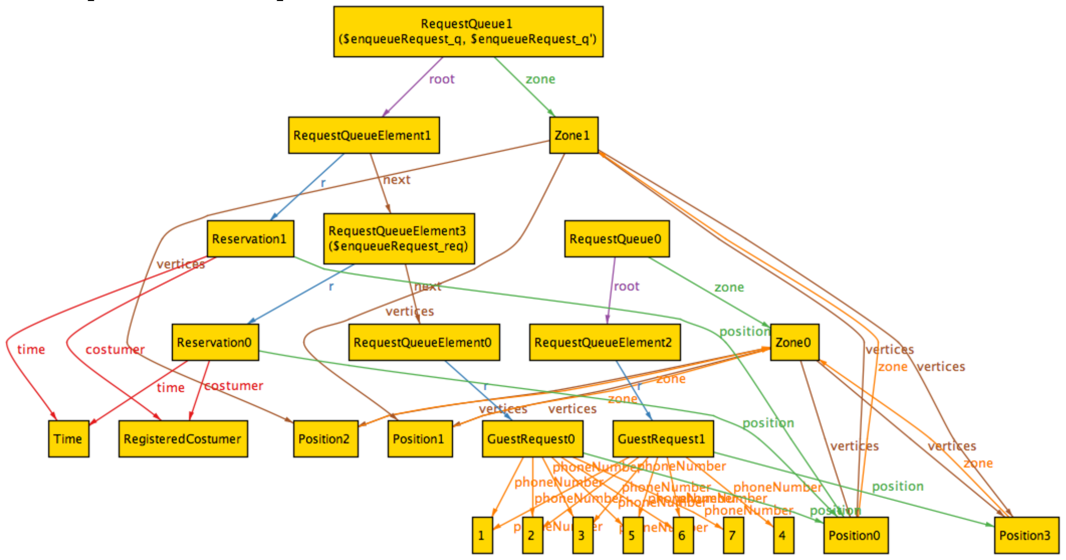
- *Code:*

```
sig RegisteredCostumer {}

sig Zone {
vertices: set Position
} {#vertices = 4}

//position
sig Position{
zone: Zone
}

//time
sig Time{/*
year:Int,
month: Int,
day: Int,
hours: Int,
mins: Int*/
}
//{year>0 and month>0 and month<=12 and day>0 and day<=31 and hours>=0 and hours<=23
and mins>0 and mins<60}
//solver cannot find instances with this right restriction so we commented it

//type of requests
abstract sig Request{
position:Position
}

sig RegisteredRequest extends Request{
costumer:RegisteredCostumer
}

sig GuestRequest extends Request{
phoneNumber: some Int
} {#phoneNumber = 6 and (all x:phoneNumber | x > 0)} //#phoneNumber should be 10 but
solver cannot find instances

sig Reservation extends Request{
costumer:RegisteredCostumer,
time:Time
}

sig RequestQueueElement {
r: Request,
next: lone RequestQueueElement
}

sig RequestQueue{
zone: Zone,
root: RequestQueueElement
}

//FACTS
fact allElementsBelongToSomeQueue {
		all e:RequestQueueElement | some q:RequestQueue | e in q.root.*next
	}

fact nextNotCyclic {
	no e:RequestQueueElement | e in e.^next
}

fact eachRequestInOnlyOneQueue{
	(no disj e1, e2:RequestQueueElement | e1 in RequestQueue.root.*next and e2 in
RequestQueue.root.*next and e1.r=e2.r)
}

fact eachRequestInRightZoneQueue{
	(all q:RequestQueue, r1:RequestQueueElement | r1 in q.root.*next implies
r1.r.position.zone = q.zone)
}

fact ReservationsHasHigherPriorityThanRequests{
	(all q:RequestQueue, e:RequestQueueElement, r1:Request | (e in q.root.*next and r1 in e.r
and not(r1 in Reservation)) implies (no r2:Reservation | r2 in e.next.r))
}

fact aQueuePerZoneAndAZonePerQueue{
	(all z: Zone | one q: RequestQueue | q.zone = z) and (all q: RequestQueue | one z: Zone |
q.zone = z)
}

pred enqueueRequest[req: one RequestQueueElement, q, q': one RequestQueue] {
	q.root = q'.root and (not(req.r in Reservation) implies ((one e:RequestQueueElement | e
in q'.root.*next and e.next = req) and #req.next=0) else (one e:RequestQueueElement | e in
q'.root.*next and e.next = req)  and not (req.next.r in Reservation))
}

pred dequeueTaxi[q, q': one RequestQueue]{
	q'.root = q.root.next
}
```

- *run enqueueRequest for 4 but exactly 2 RequestQueue:*
  - case 1: parameter req is not a Reservation:



  - case 2: parameter req is a Reservation:


RequestQueue1 ($enqueueRequest_q, $enqueueRequest_q')
root
zone
RequestQueueElement1
Zone1
next
r
Reservation1
RequestQueueElement3 ($enqueueRequest_req)
RequestQueue0
vertices
Reservation0
RequestQueueElement0
RequestQueueElement2
Zone0
time
costumer
position
Time
RegisteredCostumer
Position2
Position1
GuestRequest0
GuestRequest1
phoneNumber
1
2
3
5
6
7
4
Position0
Position3