Requirements Analysis and Specification Document

Losio Davide Francesco, Luchetti Mauro, Mosca Paolo

November 6, 2015

Contents

1 Introduction

1.1 Purpose

This document states the requirements of a taxi booking service for the government of a large city. The requirements stated below serve as a basis for the acceptance procedure of this system by the interested stakeholders: the government officer Mr.Smith Brown and the head of the taxi-drivers' committee Mr.Chris Red. The document is also intended as a starting point for the design phase and in general to give a sketch of the possibilities that would be offered by the realization of our product.

1.2 Scope

The MyTaxiService application automates the taxi accommodation described by our stakeholders. Its purpose is to provide:

a) a simplified access for the passengers to the service;

b) the guarantee of a fair management of taxi queues.

in particular through the on-line and mobile application booking facilities offered. Once this system is installed, the taxi queues will be managed automatically by our software; this ensures a fair distribution of the clients to all the taxi-drivers subscribed to the service. In the same time a passenger in search of a taxi can do his reservation anytime, from everywhere and also instantly. Furthermore it will be possible to reserve a taxi by specifying the origin and the destination of the ride and also a taxi sharing option will be implemented.

1.3 Glossary

- **MTS**: Acronym which stay for **M**y**T**axi**S**ervice, the licensed name of our project.
- **GUI**: Acronym which stay for **G**raphical **U**ser **I**nterface, which is the application graphical interface.
- **Visitor**: Not registered user of MTS application.
- **User**: Registered user of MTS application.
- **TB**: Acronym which stay for **T**era-**B**yte, a unit of measure of the memory capability.
- **Taxi-Driver**: taxi-driver user of the application.
- **Passenger**: taxi-passenger user of the application.
- **DBMS**: **D**ata**B**ase **M**anagement **S**ystem

1.4 Overview

Section 2 of this document gives a general overview of the system. It aims to show the background of the requirements stated in the section below. Section 3 gives more specific requirements for the offered functions. These functions are categorized according to the class of users they support: passengers and taxi-drivers, respectively.

2 Overall Description

2.1 Product perspective

The product is thought to be independent and totally self-contained. There isn't a previous existent system to upgrade, so we are going to build up a completely new system starting from a blank piece of paper.

APIs and code Libraries from this project will be provided. It will be possible use them in future projects and they will be available also for third parties. This is done in order to encourage the enhance of the application functionalities also with some partner companies.

2.2 Product function

The main features provided by MyTaxiService are:

- **Immediate request:** When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone. If the taxi confirms, then the system will send a confirmation to the passenger. Otherwise the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.
- **Book a taxi:** When a book request arrives, the system confirms the reservation to the user involved. Ten minutes before the meeting time MyTaxiService provides to allocate a taxi in the same way of an immediate request.
- **Taxi-sharing:** When a taxi-sharing request (TSR) arrives, the system computes all possible trajectories in according to all other TSR from different users and draws the best route for the taxi driver, defines the fee for all clients sharing the taxi and informs them and the taxi driver.

- **Taxi queue management:** The system guarantees a fair management of taxi queues. In particular, the city is divided in taxi zones. Each zone is associated to a queue of taxis. The system automatically computes the distribution of taxis in the various zones based on the GPS information it receives from each taxi. When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.
- **Support to function expandability:** Besides the specific user interfaces for passengers and taxi drivers, the system offers also programmatic interfaces to enable the development of additional services on top of the basic one.

2.3 User Characteristics

It is assumed that both the two main type of user do have the basic knowledge of operating the Internet or mobile application to have access to it. In order to allow real-time connectivity Taxi-Drivers have to provide themselves with some mobile device, weather it's a smart-phone, a tablet or a laptop.

2.4 Constraints

2.4.1 Environment

- It is predictable that a perfect connectivity, specially on the mobile devices, won't be guarantee all time. This could be a problem, in particular for taxi drivers. In this case the best thing is supposed to be a change in the availability state of the driver, which became automatically unavailable after a specific time.
- A maximum response time in the communication between taxi-drivers and costumers should be set-up, to avoid useless waste of time in several different use cases.

2.4.2 Memory

- The maximum external storage capacity for the catalogs of the system will be 3 TB.

2.5 Assumption and Dependencies

- It is assumed that in this system there may be unfair customers but not unfair taxi drivers. This behavior goes against the interests of the taxi drivers themselves. For "unfair" is meant that a passenger can book a taxi but does not attend at the established time, or does not attend at all.
- It is assumed that the user has correctly set up his device to receive notifications.
- It is assumed that a Passenger-User who has enabled the sharing option, implicitly accepts to pay the whole fee in case the system doesn't found anyone to share the ride.
- It is assumed that taxis are uniquely identified by an alphanumeric code, that is their taxi number, given at the system during drivers registration. This code helps customers to identify the driver assigned to them.
- It is assumed that both passengers and taxi drivers use their respective registration form.
- It is assumed that, having the starting point and the destination of a ride, the system can compute the fee for that specific ride.
- It is assumed that within ten minutes a taxi is always available.

3 | Specific Requirements

3.1 External Interface

3.1.1 User Interface

A wide range of different screen formats and size has to be supported. In order to make the application easy to use, the user has to be able to complete a booking in about three sequentially screens, and the taxi driver has to be capable of accepting or rejecting a request with an immediate one-tap decision. The System has also to provide one option that allows the Taxi-Driver to change his availability state in real-time. Mainly the home page of the application has to allow the passenger to select the service he want to use. Otherwise, on the taxi-side, the same page will be focused on keeping track of the actual position in the taxi-queue and in a quick feedback of the driver availability, keeping in mind that a taxi customer can change the taxi situation in anytime and not necessarily thorough the usage of the application.

To summarize, for the Passenger:

- The first screen gives a login form.
- The user has to choose between two types of services (taxi now, book a taxi).
- The user interface allows to choose the addresses by manually entering it or by using GPS services.

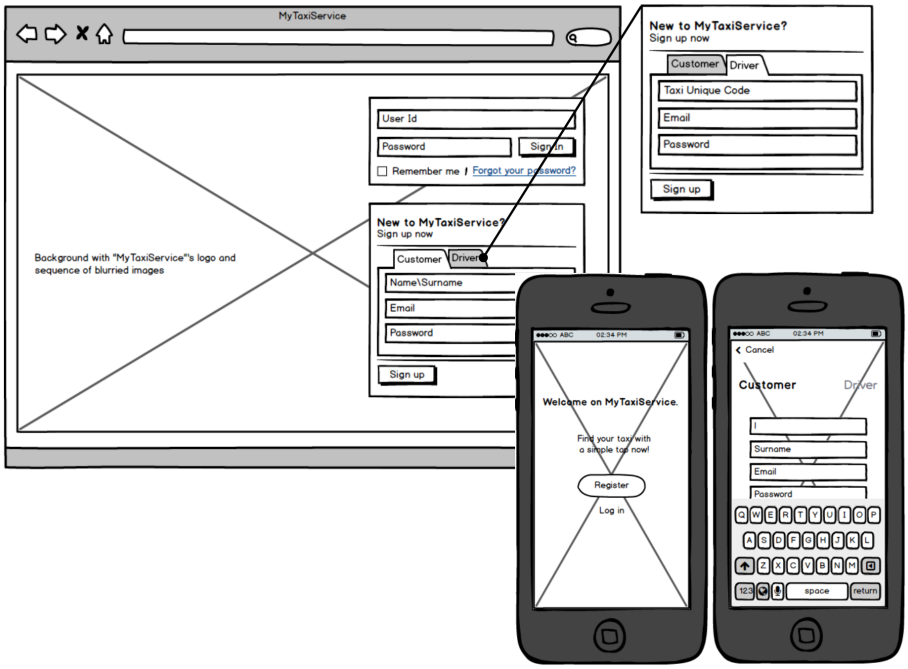
for the Taxi-Driver user:

- The first screen gives a login form.
- The user has to select his availability (available, unavailable). The unavailable state has to be the default mode.
- When a request arrives, a notification appears. This both allows "accept" or "reject" option.
- When the driver accepts a request, the system shows a map with the best route to the client.

Below are stated Mockups representing the main interfaces which the two type of users should deal with. To them follows a list of typical usage scenarios. These usage scenarios will be used as acceptance criteria: 80% of the users must be able to go through them within 2 minutes.

3.1.1.1 Login and registration

The following mockup shows the presentation page of both web and mobile application. The login and registration features are here available. A visitor can easily choose between the customer and the driver registration. The "sign in" button is used by both the users that have to insert their own email. The system loads the correct page relying to it.



3.1.1.2 "Taxi now" option

The following mockup shows the main feature of the system, the "taxi now" option. Both for the web and for the mobile app the user has to fill the fields shown. For addresses there are three possibilities:

- Writing them in the corresponding fields
- Clicking on the GPS button which finds their position
- Clicking on the map. Using this method the user must also choose whether the selected point is of departure or arrival.

Then the user can choose a number of passengers and, if wants, switch on the sharing option.

3.1.1.3 "Booking" option

The "Booking" option screen is similar to the previous. The user must choose in addition the date and time of the reservation.

3.1.1.4 Booking cancellation

This mockup shows the user reservation list. Through this screen, clicking the bin icon, an user can delete a reservation.

3.1.1.5 Taxi found notification

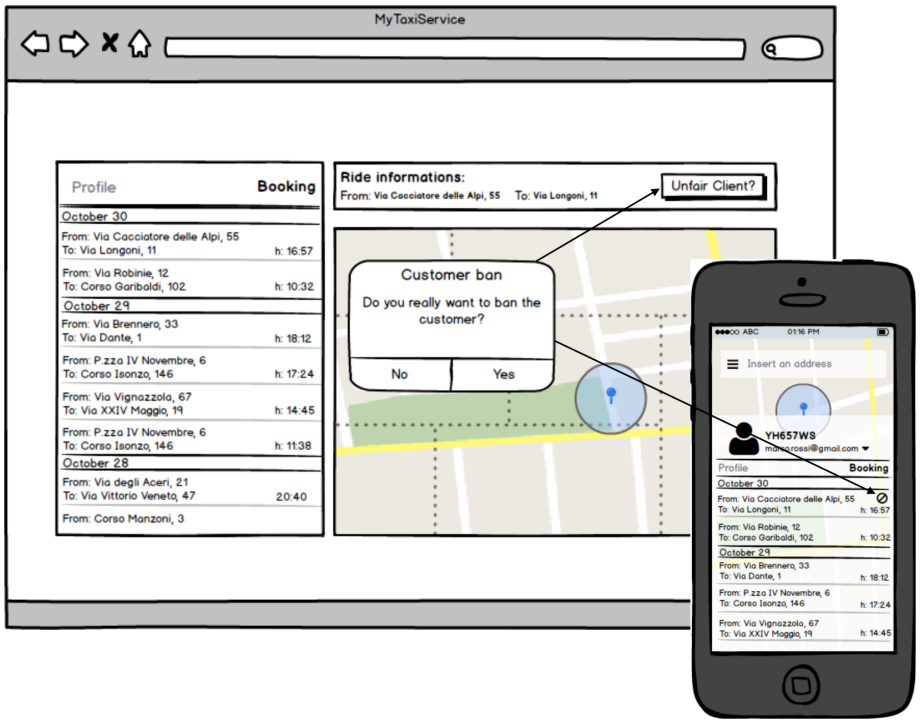
This mockup shows the view of an answer notification by the system.

3.1.1.6 Driver main page

This mockup shows the first page loaded when a driver sign in. In this page, clicking on the availability button, a driver can choose his status. The mockup also shows an incoming call request.

3.1.1.7 Driver booking page and customer ban

This mockup shows the driver booking page. This screen is useful for keeping track of bookings previously accepted and also for using the ban function. This possibility is available for drivers only for the reservation they have in charge in that moment.



3.1.2 Hardware interface

A several variety of devices have to be supported: smartphones, tablets, laptops and desktops. Due to this, the same application needs to be visualized in different window size. The user interface should definitely support an easy usage of touch-screen devices, that are the ones it is thought will be more used to access on MTS application. To simplify it should be screen-oriented. Devices should be enabled with internet connection and optionally also with GPS module.

3.1.3 Software Interface

The system will rely on Google map APIs. In particular will be used:

- Mapping system:

  Name: Google Map API;

  Version: 3.0.4;

  Source: https://developers.google.com/maps/.

This allows to take advantage of probably the most complete mapping system of the planet, and not less important, to reduce useless developing costs that probably would not benefit of the same quality.

- DBMS system:

  Name: MySQL;

  Version: 5.6.21;

  Source:https://www.mysql.it/.

This DBMS has been chosen among the others because it is one of the most lighter and easy to use. It is also a free-ware. In particular the DBMS should be implemented with the aim of keep data credentials of both the two main type of users, either sorted or memorized. It has to perform the necessarily verifying procedure during the login phases.
The inputs are:

- users' credential for the first registration;
- users' identification data for the login procedure.

and the outputs are:

- the result of login procedure, here intended as the success in identifying the user or in failure.

All the other personal data should remain confidential in order to avoid privacy law issues. It should always be user's decision the disclosure of personal information.
The database system interface will be more carefully projected in the next developing Phase.

3.1.4 Communication Interface

To communicate will be used the ordinary Internet connection, whereas 3g or WI-fi module will provide this functionality on the mobile devices.

3.2 Functional Requirements

3.2.1 Passenger-user functions

The Passenger-user, after his log-in identification, can instantly access to his reserved feature through a graphical main menu. Subsequent actions are described below:

- **Taxi now/Book a taxi** The following requirements have to be satisfied by the system in order to perform the correct execution of the booking:

  R1 The system shall check at eventual user's ban before allowing the user to do reservations.

  R2 The system shall contact the first queued taxi driver in the specific city-zone requested by the user.

  R3 The system shall create a private session for each reservation, situation with concurrent requests for the same driver must be avoided.

  R4 The system shall grant the correct priority in queuing taxis. A driver who does not accept a reservation request must be moved in the bottom of the queue.

  R6 The system shall avoid wrong inputs such as same start and destination address.

  R7 The system shall notify to the requesting user that a taxi driver has been found.

  R8 The system shall notify to the requesting user the waiting time of the first taxi, the fee and the taxi code number.

  R9 The system shall guarantee that a reservation has to occur at least two hours before the ride.

  R10 The system shall guarantee that each request corresponds an answer message which also brings a taxi confirmation.

  R11 The system shall give a confirmation about the booking taken in charge.

  inputs

  starting and ending position;

  date;

  time;

  sharing option.

  outputs

  booking acknowledged or denied

- **Taxi sharing** The following requirements have to be satisfied by the system in order to perform the correct execution of the function:

  R1 The system shall check if another user has already reserved a taxi for the same destination with the share option activated. If this happens, the best route is calculated and sent to the driver.

  inputs

  tap option enabling;

  outputs

  change in option visual check;

- **Reservation deleting** The following requirement has to be satisfied by the system in order to perform the correct execution of the deleting:

  R1 The system shall grant the possibility to delete a reservation for a user. This must be true only two hours before, whereupon this possibility is no longer available.

  inputs

  chosen reservation

  outputs

  deleting acknowledged or denied

3.2.2 Taxi driver-user functions

The Taxi driver-user, after his log-in identification, can instantly access to his reserved feature through a graphical main menu. Subsequent actions are described below:

- **accept/reject a request** The following requirements have to be satisfied by the system in order to perform the correct execution of the answer:

  R1 The system shall grant the access to the driver GUI only for those accounts that have the correct credentials.

  R2 The system shall control that all the drivers waiting in the queue are eventually available.

  R3 The system shall grant that only one driver at a time have access to a reservation request.

  R4 The system shall notify at the first queued driver the request.

  R5 The system shall update the taxi driver user's ride schedule for each confirmation sent.

  inputs

  the tap of the chosen option

  outputs

  acknowledge message of the input

- **Change of the availability state** The Taxi-Driver should be allowed to change his availability state. The following requirements have to be satisfied by the system in order to perform the correct feature execution:

  R1 The system shall provide a "not available" option that allows taxi drivers to exit from the waiting queue in case they pick up a person without using the service.

  R2 The system shall automatically switch to "not available" a driver who takes charge of a reservation. This driver is removed from the queue.

  R3 The system shall automatically switch to "not available" a driver who is not enqueued and switch to "available" an enqueued one.

  inputs

  tap of availability switch

  outputs

  change in the availability visual check

- **ban for unfair clients** To guarantee that the on-line reservation system will not be used with malicious intention, the system has also to satisfied the requirements stated below:

  R1 The system shall provide an "unfair client" option that allows taxi drivers to communicate an inappropriate client behavior (e.g. false reservation).

  R2 The system shall recognize a banned client and prevents any reservation for a limited chosen time (e.g. 1 week).

  R3 The system shall notify to a banned client his condition when he receives a ban.

  R4 The system shall display "ban information", such as time left, for banned clients that logged-in.

  R5 The system shall notify to a banned client that he is no longer banned when ban time is over.

- **Queue Managing** To guarantee the correct drivers queue managing the following requirements have to be satisfied:

  R1 The system shall guarantee that a taxi driver user can be enqueued in only one queue.

  R2 The system shall guarantee that a zone can have only one queue.

3.3 Scenarios

3.3.1 Marta registers to MyTaxiService

Marta used to call a taxi every week-end to reach his boyfriend in Seveso. Ceases to call every time the number of service, she finds on the web the MyTaxiService's home page. With only few steps, following the instructions of the system, she can enter her name, surname, email and password. After receiving the confirmation email, she can correctly log in in to the system.

3.3.2 Marco books a taxi for his cousin

Marco is an already registered user. When his cousin decides to visit him, he asks if Marco can pick him up at the airport. Unfortunately, for the arrival date, Marco is busy. So he decides to book a taxi using the "booking" function of MyTaxiService's site. First he has to choose where send the taxi so he inserts the airport's address, then uses the GPS button to fill the destination field. Clicking on the calendar icon below, he can easily select the date. He also selects the time 30 minutes after the arrival hour. "1 passenger" is the default selection so he can confirm pressing the "search" button. Then the system notifies that the booking has been scheduled.

3.3.3 Sara books instantly a taxi

Sara has heard about MyTaxiService by a friend. Because of some mechanical problems with her car, she decides to try to control how much it would cost to reach by taxi her office. She logs in with her credentials to the main page and does a search through the screen "Taxi now". She uses the GPS icon to locate herself on the map and inserts her office's address as destination. The system, in a few seconds, finds an available taxi driver and notifies Sara the waiting time, the cost for the desired route and the taxi code. She decides to confirm the reservation by clicking the "Yes" button on the notification.

3.3.4 Antonio rejects a taxi call for the opposite site of the city

Antonio is a MyTaxiService's registered taxi driver by some time. At the beginning of his working day he accesses his account through the application and sets his status on "available" by clicking on the button. The system puts him in queue in the specific area connected to the GPS signal received and, after a few minutes, it forwards a request of a user. Since the destination stated in the request is located in the opposite site of the city, Antonio decides to reject it. The system forwards the request to the next available taxi driver and puts Antonio at the bottom of the queue.

3.3.5 The driver Carlo decides to register at MyTaxiService

During the lunch break Carlo and Andrea, two very good friends taxi drivers, often meet for lunching together. Andrea talks Carlo about MyTaxiService and immediately he decides to try to use the features it offers. Carlo connects to the site, enters his unique taxi code, his email and a password. By clicking on the “sign up” button he can now accesses to his main page. Carlo sets immediately his status on "available".

3.3.6 Alice shares a taxi

Alice has to go across town and decides to use the services offered by MyTaxiService. She accesses through her credentials to her account and selects the starting point and the destination entering manually the addresses. The system notifies that a taxi driver would be available within a few minutes but the rate seems too high by Alice. She decides not to confirm the reservation by clicking “no” on the notification and to add the "sharing" option. The system automatically detects three people who would like to share a taxi in her own route. The price reported this time is about half than before. Alice decides to confirm the reservation.

3.3.7 Matteo books a taxi but does not take it

Matteo has left the car in the forced removal and decides to call a taxi to get home after work. He makes a normal request using MyTaxiService. When Giovanni, a taxi driver, accepts the request, the system notifies both of them the booking confirmation. While Matteo is waiting for the taxi he meets a friend, who immediately asks if he wants a ride. When Giovanni arrives at the destination does not find any customers. He decides to wait a few minutes but eventually is forced to report to the system the inconvenience. So Giovanni logs on to his page and, selecting the reservation, clicks the " unfair customer" button, causing the Matteo’s ban. Matteo will not be able to use the functionalities of MTS till the end of the ban period.

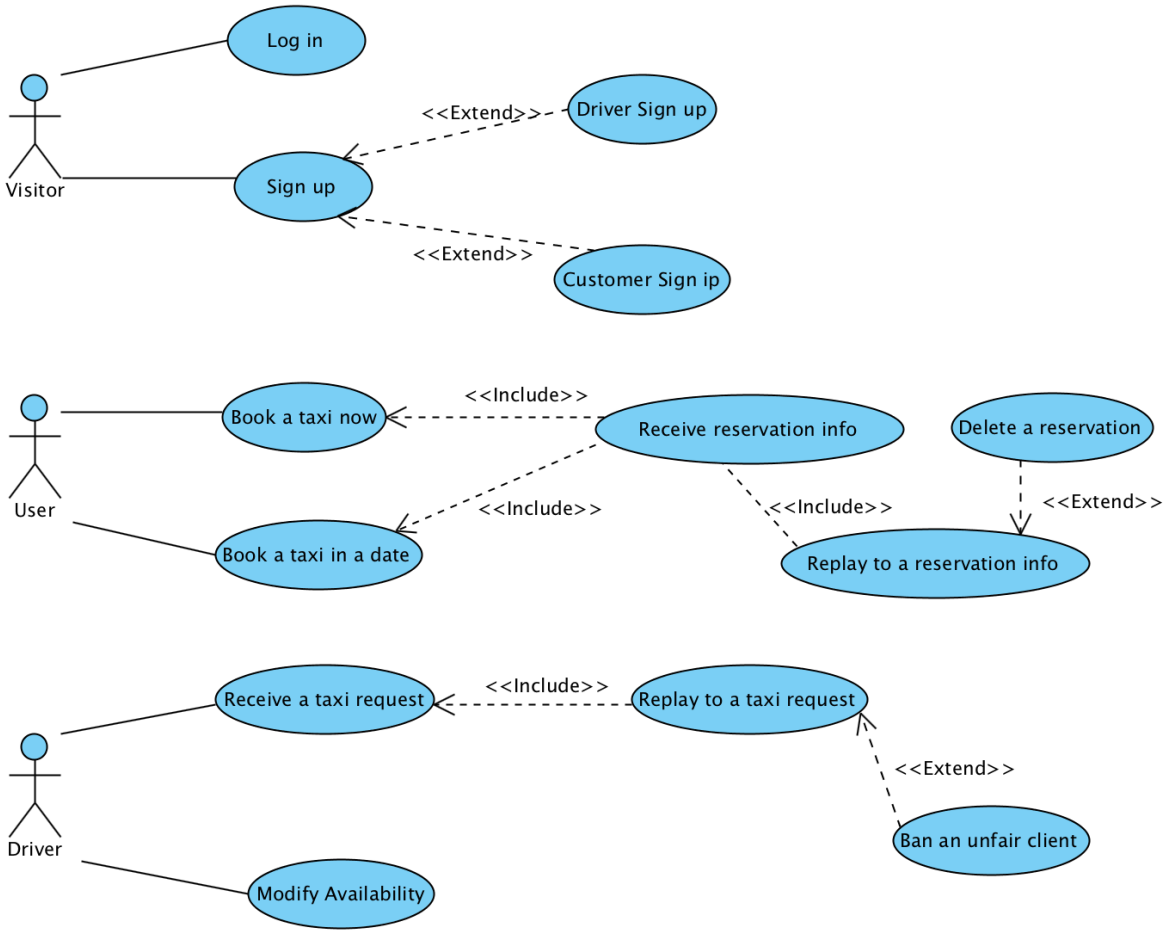
3.4 UML Models

3.4.1 Use Case Diagram

Use cases derived from the scenarios identified before:

- Log in
- Sign up
- Book a taxi now
- Receive reservation info
- Respond to a reservation info
- Book a taxi in a specific date
- Delete a taxi reservation
- Receive a taxi request
- Respond to a taxi request
- Modify availability
- Ban an unfair client

Starting from these functionalities, the following diagram has been build:

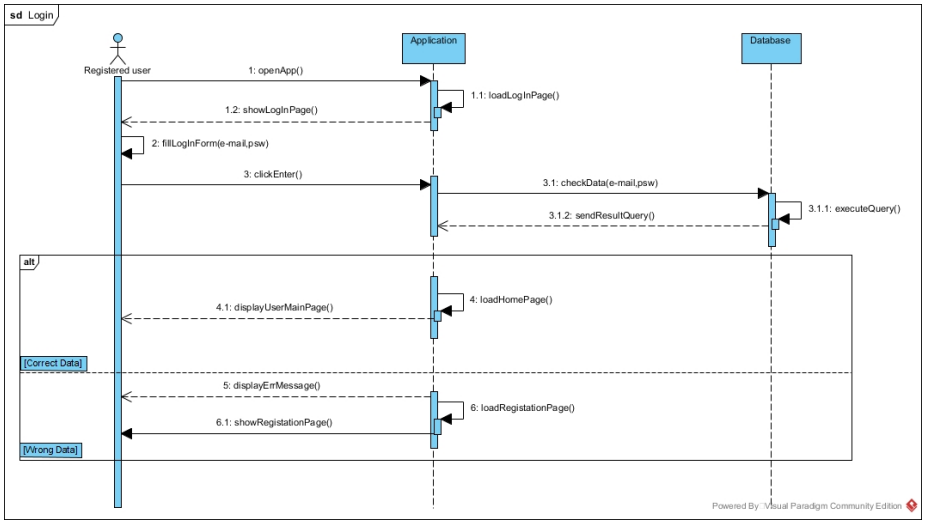


3.4.2 Use Case Description

Use cases mentioned above are described in this session in a more specific way. It has been described the main features providing summary tables and, in the main cases, sequence diagrams.

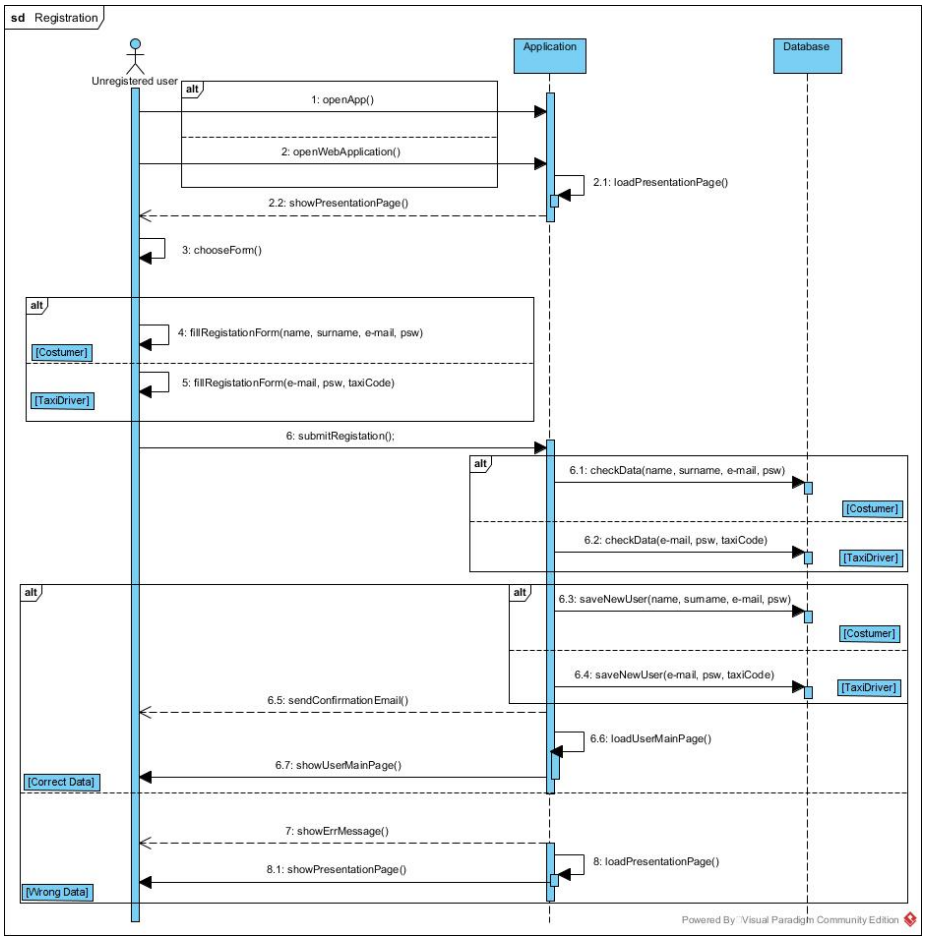
3.4.2.1 Log in

Name	Log in
Actors	User
Entry conditions	User has correctly created an account
Flow of events	• The user visits the web page of MyTaxiService • The user enters his username (email) and password • The user clicks on the “Sign in” button
Exit conditions	The system shows the user’s main page
Exceptions	The user inserts a wrong username or password. In this case an error message must be shown to the user



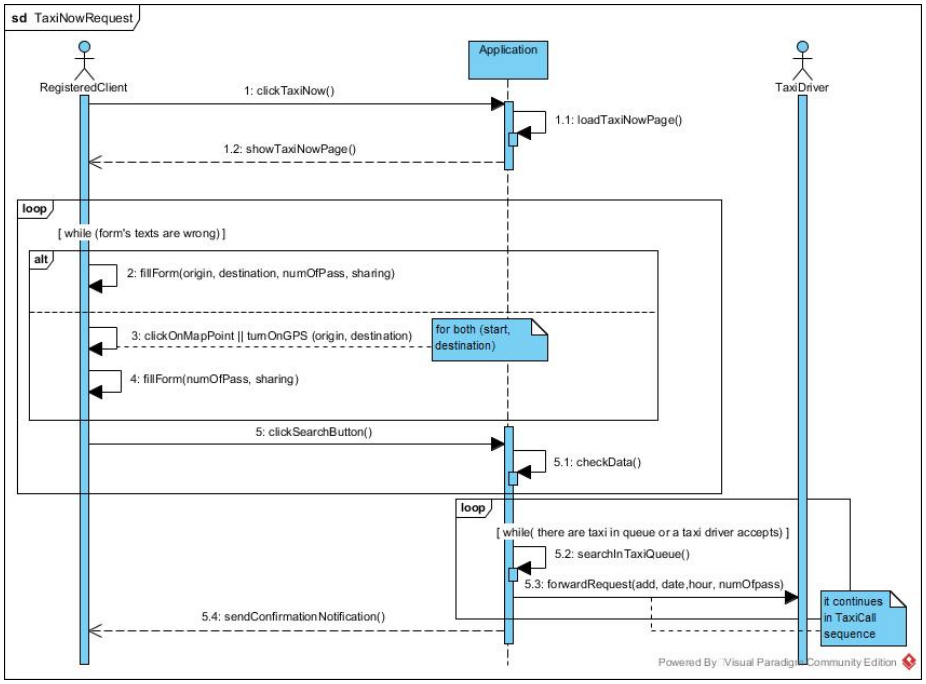
3.4.2.2 Sign up

Name	Sign up
Actors	Visitor
Entry conditions	The visitor is not already registered
Flow of events	• The visitor goes at the MyTaxiService's web page or opens the app • The visitor chooses among the two option "customer" or "driver" and clicks on the corresponding one • The customer visitor has to insert: 1. Name 2. Surname 3. Email 4. Password • The driver visitor has to insert: 1. Taxi unique code 2. Email 3. Password • The visitor clicks on the "Sign up" button • The system send him a confirmation email and shows his personal page
Exit conditions	The system registers the visitor as a new user.
Exceptions	The visitor inserts an already existing email or taxi unique code. The visitor doesn't fill all the fields. In both cases an error message must be shown to the user.


sd Registration
Unregistered user
Application
Database
alt
1: openApp()
2: openWebApplication()
2.1: loadPresentationPage()
2.2: showPresentationPage()
3: chooseForm()
alt
4: fillRegistationForm(name, surname, e-mail, psw)
[Costumer]
5: fillRegistationForm(e-mail, psw, taxiCode)
[TaxiDriver]
6: submitRegistation();
alt
6.1: checkData(name, surname, e-mail, psw)
[Costumer]
6.2: checkData(e-mail, psw, taxiCode)
[TaxiDriver]
alt
alt
6.3: saveNewUser(name, surname, e-mail, psw)
[Costumer]
6.4: saveNewUser(e-mail, psw, taxiCode)
[TaxiDriver]
6.5: sendConfirmationEmail()
6.6: loadUserMainPage()
6.7: showUserMainPage()
[Correct Data]
7: showErrMessage()
8: loadPresentationPage()
8.1: showPresentationPage()
[Wrong Data]
Powered By Visual Paradigm Community Edition

3.4.2.3 Book a taxi now

Name	Book a taxi now
Actors	User
Entry conditions	User has correctly logged in
Flow of events	• The user clicks on the “Taxi now” label (default option, usually this action is not needed) • The user inserts the starting point using one of the following methods: 1. By entering it manually in the corresponding field 2. By using the GPS button that locates him 3. By clicking on the map and selecting “start” • The user inserts the destination using one of the following methods: 1. By entering it manually in the corresponding field 2. By using the GPS button that locates him 3. By clicking on the map and selecting “destination” • The user selects the number of passengers • The user clicks on the “Sharing” option (if wants) • The user clicks on the “Search” button
Exit conditions	The system sends the request to the corresponding queue of drivers. The user waits for an answer and start the “Receive reservation info” use case.
Exceptions	The user selects an invalid address. The system shows an error message.



3.4.2.4 Receive reservation info

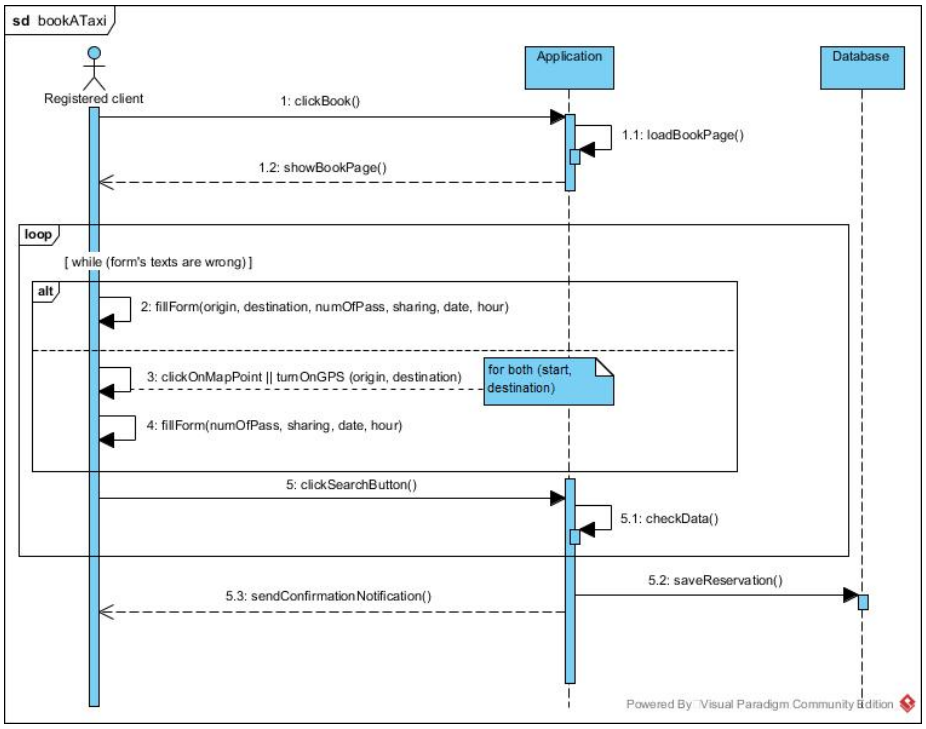
Name	Receive reservation info
Actors	User
Entry conditions	User has finished one of the “book” use case by clicking on the “Search” button. Driver has finished the “respond to a taxi request” use case by clicking on the “Yes” button.
Flow of events	• The system estimates time and cost for the reservation • The system sends a notification to the user • The user views the notification
Exit conditions	The user responds to the notification starting the “Respond to a reservation info request” use case.
Exceptions	No driver responds to the request in the session time. The system shows a “No taxi available” message.

3.4.2.5 Respond to a reservation info

Name	Respond to a reservation info
Actors	User
Entry conditions	User has visualized the reservation infos
Flow of events	• The user clicks “yes” or “no” to make his choice about the reservation • The system sends a confirmation to the driver who has accepted the request
Exit conditions	The user finishes his reservation process. The system adds the reservation infos in the user “booking” page. The homepage is shown.
Exceptions	If errors with the respond occurs in the session time, the system shows a “error” message. A “deletion” notification is sent to the driver.

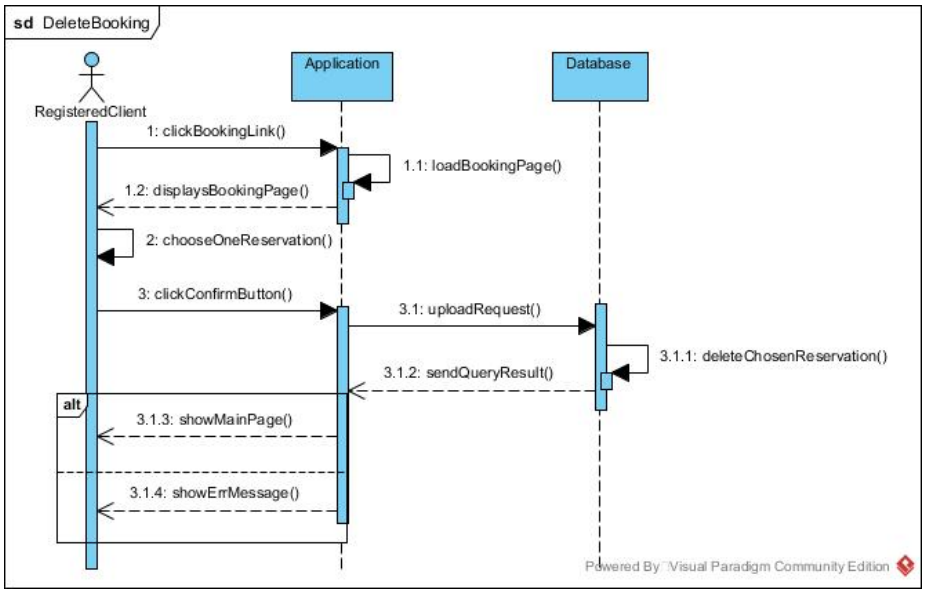
3.4.2.6 Book a taxi in a specific date

Name	Book a taxi in a specific date
Actors	User
Entry conditions	User has correctly logged in
Flow of events	• The user clicks on the “Book” label • The user inserts the starting point using one of the following methods: 1. By entering it manually in the corresponding field 2. By using the GPS button that locates him 3. By clicking on the map and selecting “start” • The user inserts the destination using one of the following methods: 1. By entering it manually in the corresponding field 2. By using the GPS button that locates him 3. By clicking on the map and selecting “destination” • The user selects the number of passengers • The user selects an hour and a date by entering it or by clicking on the calendar icon • The user clicks on the “Sharing” option (if wants) • The user clicks on the “Search” button
Exit conditions	The system sends the request to the corresponding queue of drivers. The user waits for an answer and start the “Receive reservation info” use case.
Exceptions	The user selects an invalid address or date. The system shows an error message.



3.4.2.7 Delete a taxi reservation

Name	Delete a taxi reservation
Actors	User
Entry conditions	User has correctly logged in. User has correctly done a reservation.
Flow of events	• The user clicks on the “booking” link. • The user chooses which reservation wants to delete by clicking on it. • The user clicks on the “Confirm” button.
Exit conditions	The system deletes the chosen reservation by updating the DB. The user waits for an answer. The user’s main page is shown.
Exceptions	The user sends the delete request less than twenty minutes of the meeting time. The system shows an error message.

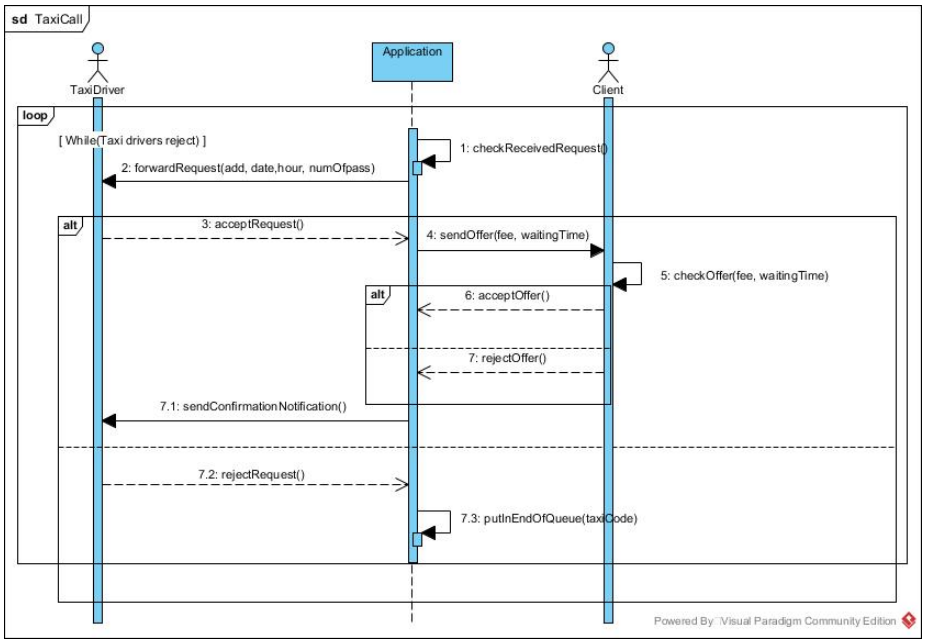


3.4.2.8 Receive a taxi request

Name	Receive a taxi request
Actors	Driver
Entry conditions	Driver has correctly set his status on “available”
Flow of events	• The system has an incoming request from a certain zone • The system send a notification to the first driver in queue • The driver views the notification
Exit conditions	The driver responds to the notification starting the “Respond to a taxi request” use case.
Exceptions	There is no exception

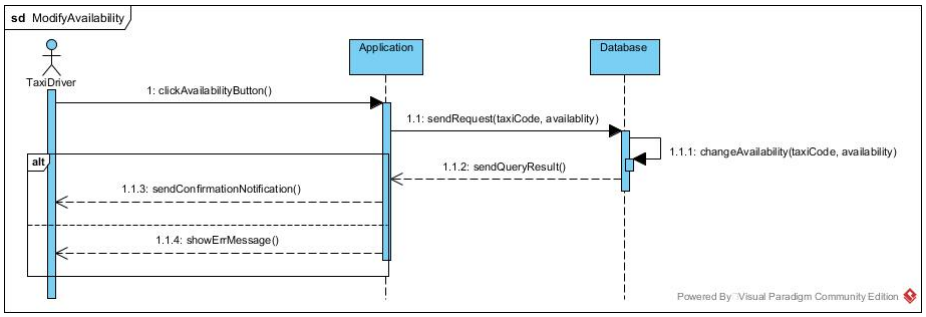
3.4.2.9 Respond to a taxi request

Name	Respond to a taxi request
Actors	Driver
Entry conditions	Driver has visualized the reservation request
Flow of events	• The driver clicks “yes” or “no” for accepting or rejecting the call • The system handles the decision by: 1. Exiting the driver from the waiting queue in case of “yes” decision 2. Moving the driver at the bottom of the queue in case of “no” decision
Exit conditions	The driver finishes his reply process. The system adds the reservation infos in the driver “booking” page. The map to reach the customer is shown.
Exceptions	If error occurs with the reply’s forwarding, the system sends the request to the following queued driver



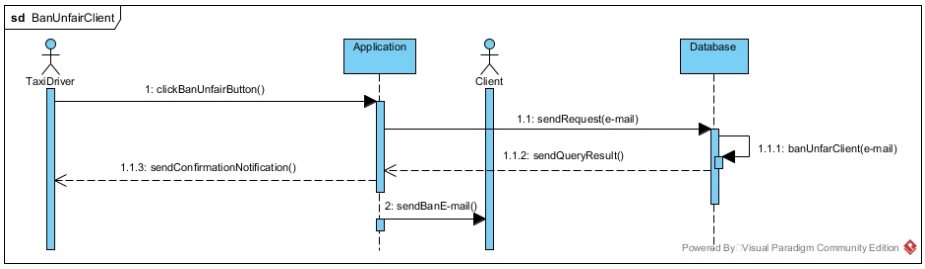
3.4.2.10 Modify availability

Name	Modify availability
Actors	Driver
Entry conditions	Driver has correctly logged in
Flow of events	• The driver clicks on the availability he wants to select among: 1. Available 2. Not available (default one) • The system adds/deletes him in/from the zone queue
Exit conditions	The system shows the change in availability
Exceptions	There is no exception



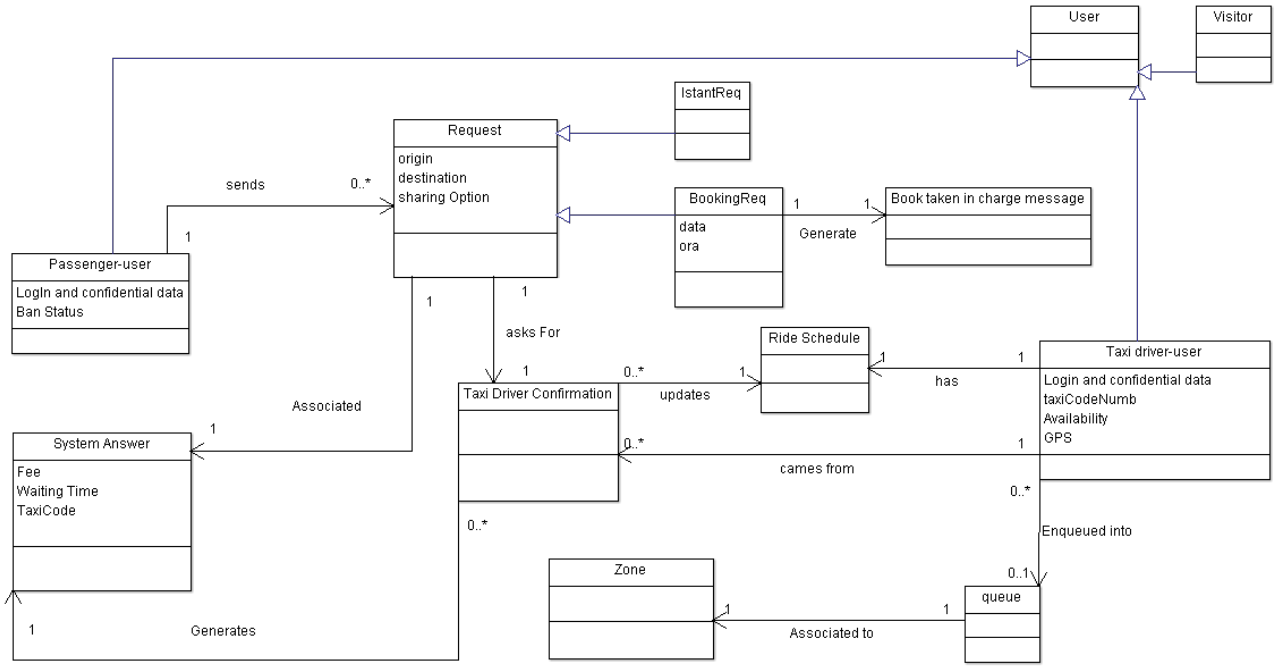
3.4.2.11 Ban an unfair client

Name	Ban an unfair client
Actors	Driver
Entry conditions	Driver has correctly accepted a request
Flow of events	• The driver clicks on the corresponding reservation in his page • The driver clicks on the “unfair client” button • The driver confirms his decision by clicking “yes” in the dialog message
Exit conditions	The system sends a “ban notification” to the unfair client and updates his status on the DB. The driver’s main page is shown.
Exceptions	If error occurs with the notification’s forwarding, the system simply doesn’t do nothing



3.4.3 Class Diagram Model

Below is stated an UML Class diagram intended as conceptual model for the application domain. It highlights the main relationship among the entities taking parts to our real world/system environment.



3.5 Non Functional requirements

3.5.1 Performance Requirements

The system must be interactive, in every action-response there are no immediate delays. In case of opening windows forms, of popping error or acknowledgment messages and saving the settings or sessions, there are delays much below 2 seconds. When connecting to the server, the delay is based editing on the distance of the 2 systems and the configuration between them, so there is high probability that there will be or not a successful connection in less than 20 seconds for sake of good communication. The system will initially support 1000 concurrent access points. Its maximum capacity is 2000 concurrent access points. The present database holds 100 000 different Passenger-users data and 1500 taxi-drivers data. The storage capacity needed for these data is about 2tb. On average 10 000 passengers and 100 drivers subscription are expected per year. The system must be able to serve 3000 users simultaneously. With this maximum load and, user interaction as listed in sections 3.2.1 and 3.2.2 must be answered within five seconds in 80% of the cases.

3.5.2 Software System Attributes

3.5.2.1 Reliability:

- The system shall be able to cope with the possible loss of connection between clients and central server by ensuring atomicity of all operations.
- Components of the project code will be tested alongside the implementation phase to ensure that they are functional.
- At least, integrated project code will be tested to ensure that greater than or equal to 90% of the code is covered at run-time.

3.5.2.2 Availability:

During normal office hours (9 am–5 pm) the system must be available 99% of the time. A backup of the system is made every day at 4 am.

3.5.2.3 Security:

The main security concern is for users accounts hence proper login mechanism should be used to avoid hacking. The id registration is way to spam check for increasing the security. Hence, security is provided from unwanted use of recognition software. Each user will only be able to access functionality that compete to his category by his identification. In particular it will be necessary a mechanism of advanced encryption to ensure a good level of security for the sensible data.

3.5.2.4 Maintainability:

A good level of software maintainability is required due to support future bug fixing and function expandability according to the releasing of MTS APIs. The software has to be wrote in modular, object oriented way. This could improve the integration also with third parties projects.

3.5.2.5 Portability:

The application must be compatible with many different devices in order to reach as many users as possible, so the application should not be platform dependent.

4 Appendix

4.1 Alloy model

The Alloy model stated below formally verifies the class diagram and his relations and is based on the previously stated assumptions, requirements and constraints.

```
abstract sig User {}

sig VisitorUser extends User {}

sig PassengerUser extends User {

        send: set Request,
        banned: lone Ban

}

one sig Ban {}

one sig PotentialSharableRide{}

sig TaxiDriverUser extends User {

        send: set TaxiConfirmation,
        enqueuedInto: lone Queue,
        has: one RideSchedule,
        availability: one Boolean,

}

sig RideSchedule{

        associatedTo: one TaxiDriverUser,
        updatedBy: set TaxiConfirmation

}

sig Destination{}

sig Origin{}

abstract sig Request{

        askFor: one TaxiConfirmation,
        sendedBy: one PassengerUser,
        startsFrom: one Origin,
        endsIn: one Destination,
        sharingAvailable: lone PotentialSharableRide
```

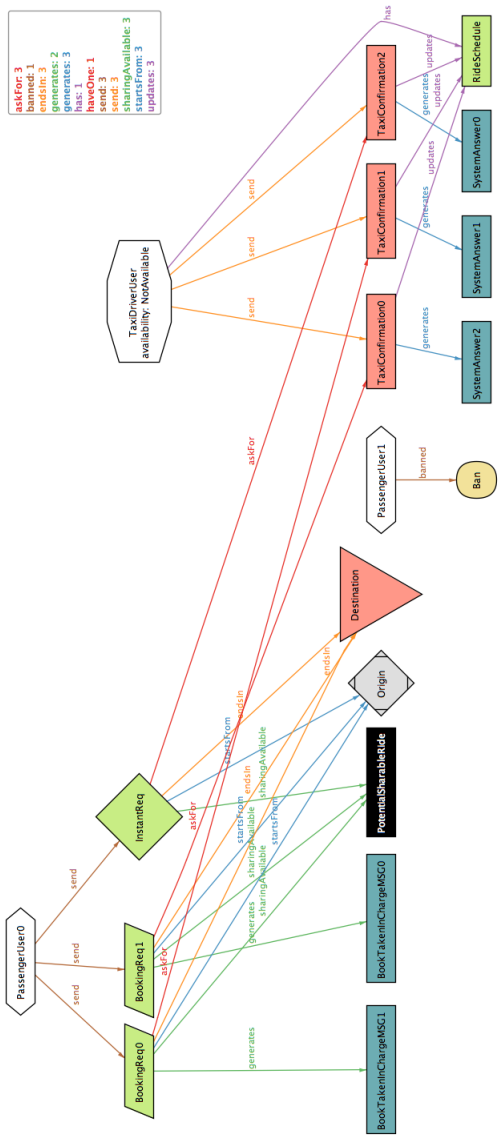
```
}


sig TaxiConfirmation{

                camesFrom: one TaxiDriverUser,
                neededBy: one Request,
                generates: one SystemAnswer,
                updates: one RideSchedule

}

sig InstantReq extends Request {        }

sig BookingReq extends Request {

        generates: one BookTakenInChargeMSG

 }

sig BookTakenInChargeMSG {

        generatedBy: one BookingReq

}

sig SystemAnswer{

        generatedBy: one Request,
        associated: one TaxiConfirmation

}

sig Queue{

        contains: set TaxiDriverUser,
        associatedTo: one Zone

}

sig Zone {

        haveOne: one Queue

}

abstract sig Boolean {}

one sig Available extends Boolean {}
one sig NotAvailable extends Boolean {}

//Fact eliceted from class diagram, assumptions and requirements

//Request relations:

        fact eachReqHasOnePUser{
                all r: Request , p: PassengerUser | r in p. send <=> p in r. sendedBy
        }

        fact eachRequestNeedATaxiConfirmation{
                all r: Request , t: TaxiConfirmation | r in t. neededBy  <=> t in r. askFor
        }

        // A booking taken in charge generates MSG

        fact BookingMSG {
```

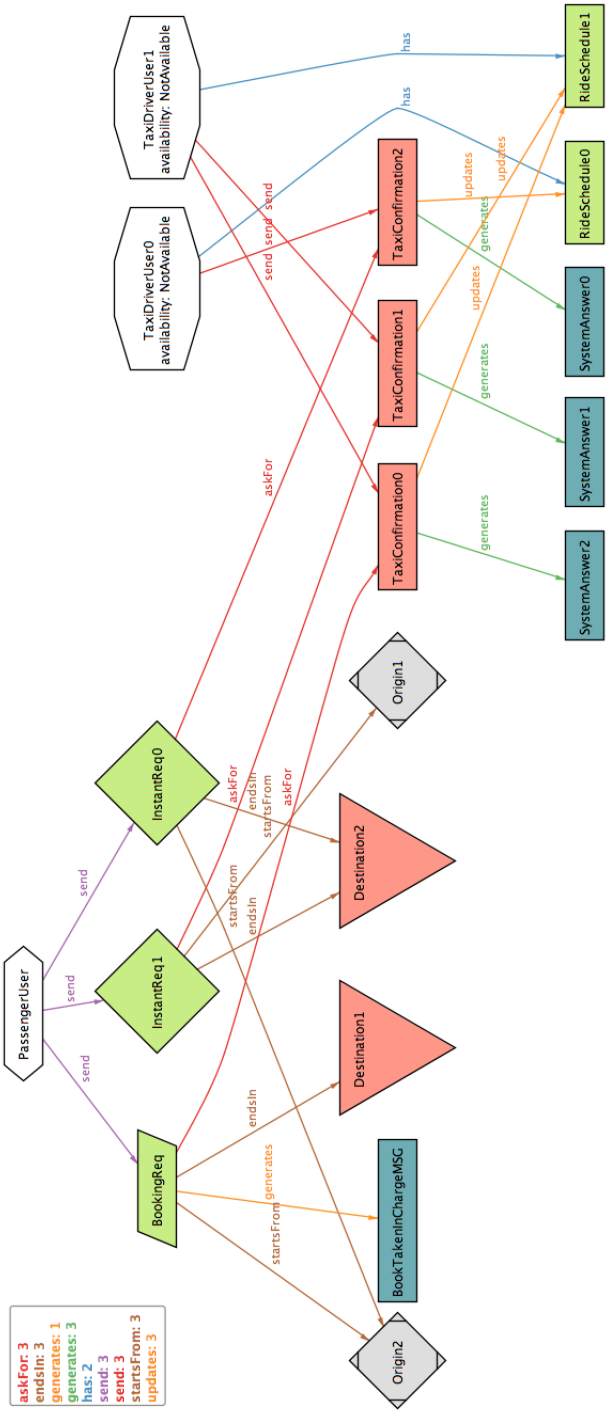
```
                all b: BookingReq , m: BookTakenInChargeMSG   | b in m. generatedBy
<=> m in b. generates
        }

//Taxi Confirmation relations:

        fact eachTaxiConfIsSendedByATaxiDriverUser{
                all t: TaxiDriverUser ,  c: TaxiConfirmation | t in c. camesFrom
<=> c in t. send
        }

        fact confirmationToOneSystemAnswer{
                all s: SystemAnswer ,  c: TaxiConfirmation | s in c. generates
<=> c in s. associated
        }

        fact confirmationUpdatesRideSchedule{
                all c:TaxiConfirmation, r:RideSchedule | r in c. updates  <=> c in r. updatedBy
        }

        fact confirmationUpdatesRightSchedule{
                all c:TaxiConfirmation | some r:RideSchedule, t:TaxiDriverUser |
(t in c. camesFrom <=> t in r. associatedTo) and
                (r in   c.updates <=> r in t. has ) and ( t  in r. associatedTo)
        }

//Queue Relations:

        fact eachTaxiCouldBeEnqueuedOnlyInOneQueue{
                all t: TaxiDriverUser ,  q: Queue | t in q. contains  <=> q in t. enqueuedInto
        }

        fact oneQueueForOneZone{
                all q: Queue ,  z: Zone | z in q. associatedTo  <=> q in z. haveOne
        }

        // A driver is removed from the queue if he send a confirmation

        fact driverRemovedFromTheQueue{
                all t:TaxiDriverUser, q: Queue, c: TaxiConfirmation | t in c. camesFrom <=>
t not in q. contains
        }

        // A Driver not in queue is not available

        fact TaxiNotInQueueISunaVailable {
                all t:TaxiDriverUser, q: Queue | t not in q. contains <=> (t.availability =
NotAvailable)  and t not in q. contains
        }

        // A Driver in queue is available

        fact TaxiNotInQueueISunaVailable {
                all t:TaxiDriverUser, q: Queue | t in q. contains <=> (t.availability =
Available)
        }

//Ride Schedule Relations:

        fact oneRideScheduleForOneDriver{
                all r: RideSchedule, t: TaxiDriverUser | r in t. has <=> t in r. associatedTo
        }

//Ban Relations:

        fact aBannedUserCannotSendRequest{
                all p:PassengerUser | (#p.banned = 1) implies (p.send = none)
        }
```

```
//Sharing Relations:

        //If two request have the same origin and destination, they are sharable

        fact SharingFinding {
                all r1,r2: Request | ((r1.startsFrom = r2.startsFrom) and
                (r1.endsIn = r2.endsIn) and not(r1.startsFrom = none)
                and not(r1.endsIn = none)and not(r1= r2)) implies ( (#r1.sharingAvailable >
0) and
                (#r2.sharingAvailable > 0) )
        }


pred show{

}

run show  for 4
```

4.2 Generated world

Below are stated a couple of worlds generated with Alloy. These examples are made with a 3 element scope and they aim to give a visual feedback about the world consistency.




TaxiDriverUser1
availability: NotAvailable
TaxiDriverUser0
availability: NotAvailable
RideSchedule1
RideSchedule0
TaxiConfirmation2
TaxiConfirmation1
TaxiConfirmation0
SystemAnswer0
SystemAnswer1
SystemAnswer2
Origin1
Origin2
Destination2
Destination1
InstantReq0
InstantReq1
BookingReq
BookTakenInChargeMSG
PassengerUser
has
send
askFor
updates
generates
endsIn
startsFrom
askFor: 3
endsIn: 3
generates: 1
generates: 3
has: 2
send: 3
send: 3
startsFrom: 3
updates: 3

4.3 Used Tools

We used the following tools to make the RASD document:

- LyX 2.1: to redact and format the document;
- Balsamiq mockups : to design the mockups;
- Argo: to draw the Class Diagram;
- Visual Paradigm 10 Community edition: to draw the Use Cases Diagram and the Sequence Diagrams;
- Alloy Analyzer 4.2: to model and to prove the consistency of the world we have shaped.

4.4 Time Spent

For redact, correct and review this Document we spent almost 20 hours per person.