Table of Contents

1. Introduction

1.1 Description of the problem

Our object is to project a system to optimize the taxi service of Big_City, which government is our stakeholder, simplifying the access of passengers to it and, at the same time, guaranteeing a fair management of taxi queues. It will be composed of a web application and a mobile application allowing users to make ride requests and reservations and informing them about the code of the incoming taxi and the waiting time. In order to maintain the fairness of queues, the system is constantly updated about the availability of taxi drivers and call confirmations from a mobile application embedded in the cars. Big_City is divided in zones (2 km² each) and there is a one-to-one correspondence between them and the queues, which are lists of the identifiers of available taxis. Furthermore, the system automatically computes the distribution of taxis in each zone based on the GPS information it receives from each one. If a taxi receives a ride request, it means that it is the first queueing in that zone and it has the possibility to accept or reject it. If the driver accepts, the system will send a confirmation to the passenger, else the system will move their taxi to the last position of the queue and forward the request to the second taxi queueing. Moreover, Taxi reservations must occur at least two hours before the ride, specifying origin and destination, and allocated to a taxi by the system 10 minutes before the meeting time with the passenger. Lastly, programmatic interfaces (APIs) should be provided in order to enable the development of additional services.

1.2 Goals

When the system is online, it has to guarantee these functionalities:

- management of the queues of available taxis

- create and manage taxi ride requests

- create and manage taxi ride reservations

- scalability to additional services

1.3 Domain Properties / Assumptions

This is a list of some domain properties and assumptions that we think has to be true:

- the taxi service is entirely controlled by the government of Big_City, that owns the taxis and assigns an username and password to taxi drivers to let them access the service

- In Big_City, the government ensures that there is a number of taxis large enough to avoid delays and disruptions

- every taxi is identified by an unique code and must have an embedded Android Auto operating system, with myTaxiService and Google Maps (maps already downloaded and available offline) as the only usable and preinstalled applications

- the city is divided in zones that have no common coordinates among themselves and the range of each one is a fixed precondition (2 km^2)

- when taxi drivers inform the system that their taxi is available, they cannot move from the zone where they have declared availability

- taxis have an on-board GPS that always returns their accurate location to the system when their availability is turned on

- when a passenger requests a taxi with the mobile application, the GPS embedded in the smartphone always returns their accurate location to the system

- a passenger requests or reserves a Taxi only if they can take it in the exact meeting location they informed about with their GPS (no fake requests) or independently decided (no fake reservations)

- when a taxi driver confirms a ride, it will always reach the passenger in 10 minutes at most (maximum waiting time)

- when a passenger reserves a taxi, they will always give the right information about meeting date and time, departure and arrival location

- origin and destination of ride requests and reservations must be within Big_City

- when a user requests or reserves a taxi, the maximum number of passengers of the ride must be 4 included them

- taxi drivers inform the system that they completed a ride assigned to them only if they really have

1.4 Proposed system

The system is composed of these parts, managed by a back-end centralized server application:

- **Mobile application (for Users):** designed for Android-based and iOS-based smartphones, should be developed with the relative native programming language (Java and Swift/Objective C), or with a framework (like Ionic or Phonegap) that allows to develop a single hybrid application with web technologies (HTML5, CSS3, JavaScript) compatible with both Google and Apple operating systems

- **Web application (for Users):** composed by a front-end that should be very similar and provide the same functionalities of the mobile application to users

- **Mobile application (for Taxis):** designed for Android Auto, a particular operating system running on embedded devices for cars. All taxis in Big_City must have it installed by assumption

2. Actors

The actors involved in this project are four:

- **Guest:** a person that opens either the web application or the mobile application for the first time, facing its Login page. They are not already registered in the system, so they have to sign up, completing a registration form, before accessing the service

- **User:** a person that is already registered in the service with email address and password; they can log in into the system, request or reserve a taxi ride, check the list of their booked taxis, cancel a reservation

- **Taxi driver:** a person that is employed and automatically registered into the system by the government to drive a taxi in Big_City. They can inform the system about the availability of their taxi and accept, or not, a ride assigned by the system

3. Requirements

3.1 Functional requirements

GUEST:
- Sign up in the web application or mobile application by filling up a form with username and password

USER:
- Log in into the web application or mobile application with username and password
- Create a ride request
- Create a ride reservation
- Check the list of booked taxis
- Cancel a ride reservation

TAXI DRIVER:
- Confirm a ride(reservation or request)
- Reject a ride(reservation or request)
- update their availability

SYSTEM:
- propose a ride to a taxi
- confirm request or reservation to the user with related information
- insert, remove or move a taxi from a queue
- update availability of taxis

3.2 Non Functional requirements

(1) USER INTERFACE

The user interface of the web application and the mobile application for the users, and of the Android Auto application for the taxis must be user-friendly. To better explain it, we realized mockups for Android smartphones and Android Auto.
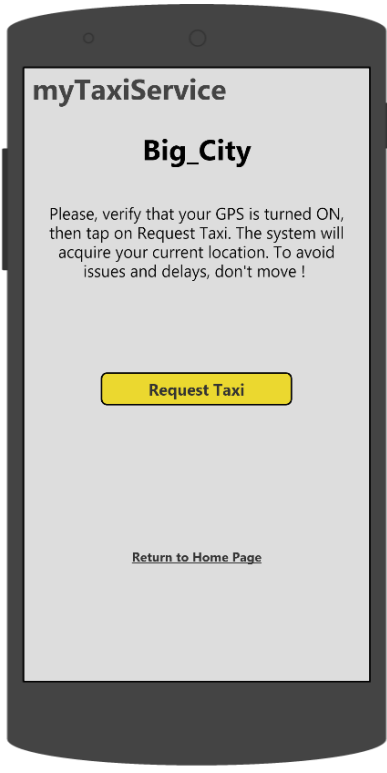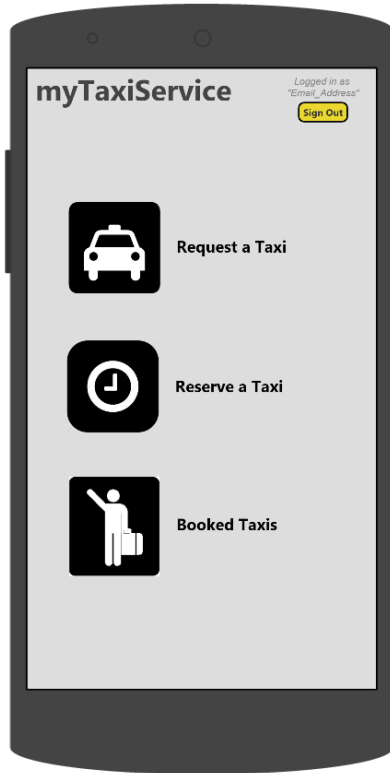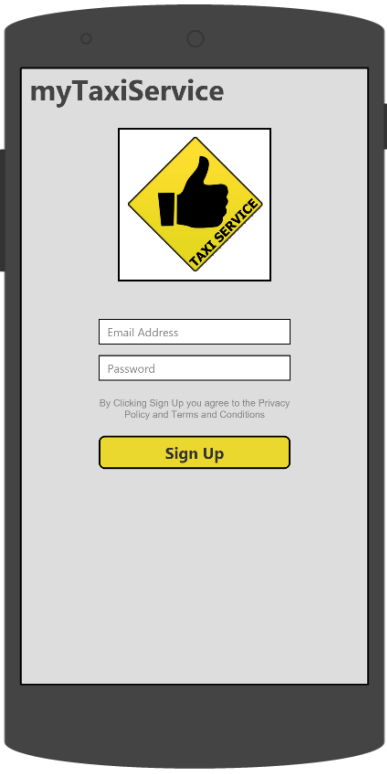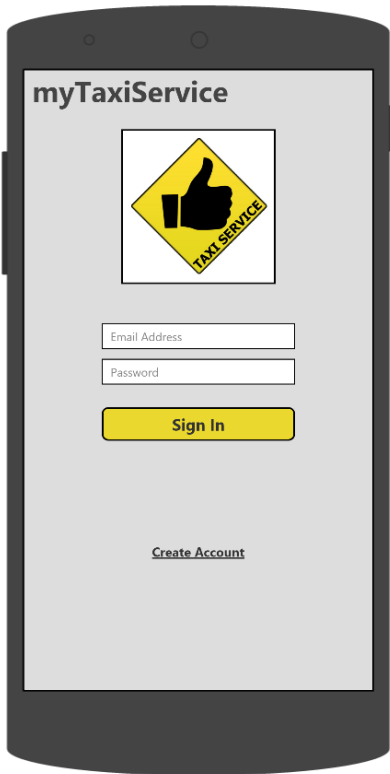
After downloading, installing and running the application from the Google Play Store, the user is facing the <u>Login Page</u> and is requested to insert their <u>Email Address</u> and <u>Password</u>, if already registered. Else, they has to tap on <u>Create Account</u>, action that leads to the <u>Registration Page</u> where desired <u>Email Address</u> and <u>Password</u> have to be inserted. If correct couple <u>Email Address</u>-<u>Password</u> inserted (i.e. it is in the database) the <u>Sign In</u> button, when tapped, leads to the <u>Home Page</u>; else, an error message appears and the user is invited to check the written information. If an <u>Email Address</u> is not already in the database, the <u>Sign up</u> button, when tapped, leads to the <u>Home Page</u>; else, an error message appears and the user is invited to check the written information.
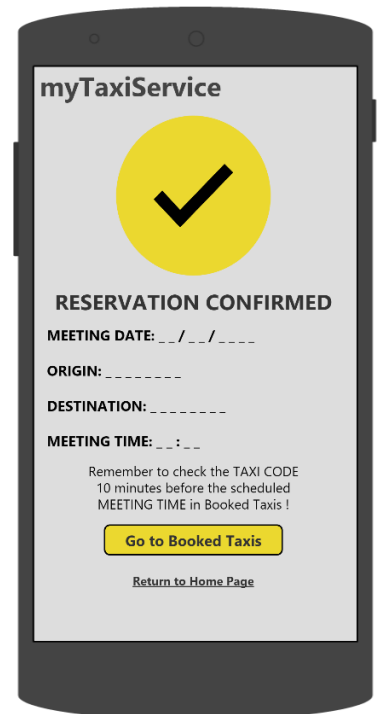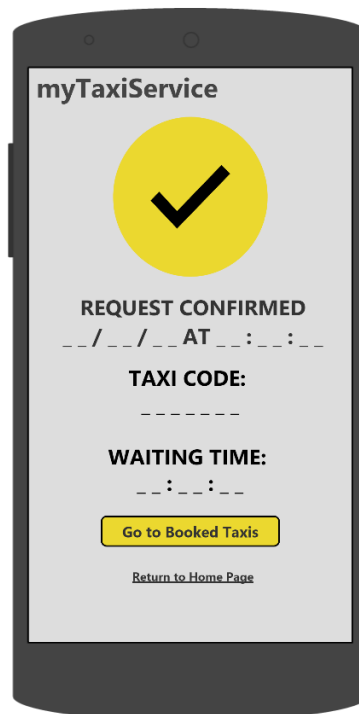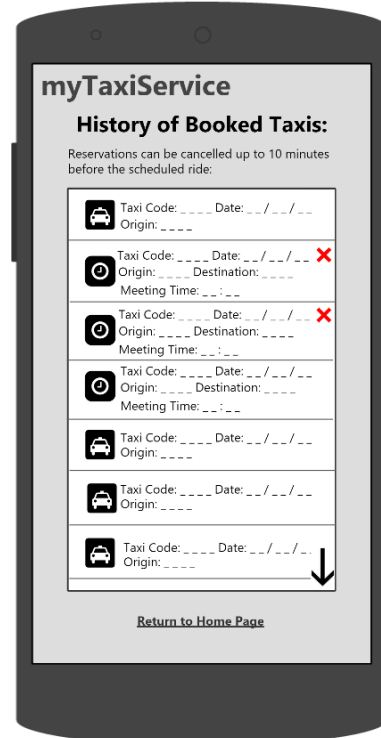
When the user reaches the <u>Home Page</u>, they can <u>Request a Taxi</u>, <u>Reserve a Taxi</u>, check the list of the <u>Booked Taxis</u> and <u>Sign Out</u> from the service and return to the <u>Login Page</u>.

In the <u>Request a Taxi</u> page, the user is requested to turn on the GPS on their smartphone, in order to let the system automatically acquire their current position and consider it as the meeting location. This is the only difference with the web version of the application, since in that case the user is required to specify their location in a form (since laptops do not have GPS module). If the user changes their mind and do not want to request a taxi anymore, they can tap on <u>Return to Home Page</u>. When the <u>Request Taxi</u> button is tapped, the <u>Request Confirmed</u> page appears after the processing of the request (<u>Processing Page</u> that is visualized for few seconds), with all the needed information. In the <u>Request Confirmed</u> page, the user can go to the <u>Booked Taxis</u> page with the relative button, or return to the <u>Home Page.</u>

In the <u>Reserve a Taxi</u> page, the user has to fill the required fields with all the requested details and then tap on <u>Reserve Taxi</u> (or on <u>Return to Home Page</u> if they are not convinced). Then, the <u>Reservation Confirmed</u> page with detailed information appears. Also in this case, the user can then go to the <u>Booked Taxis</u> page, or <u>Return to Home Page</u>. A push notification will arrive on the website or the smartphone 10 minutes before the meeting time, with the code of the incoming taxi (that will be displayed also in the "Booked Taxis" section). The <u>Booked Taxis</u> page contains a scrollable history of all the requested and reserved taxis with all the related information. The entries of the reserved taxis have the possibility to be cancelled by tapping on the red "X" that appears only up to 10 minutes

before the reserved ride (an additional yes-no "are you sure?" question is asked to the user for confirming the deletion). Also for this page, there is the possibility to Return to Home Page.

myTaxiService

Big_City

Please specify Meeting Date, Origin and Destination (within Big_City), Meeting Time of the ride. In order to avoid delays, try to be as detailed as possible:

MEETING DATE:
(DD/MM/YY)

ORIGIN:

DESTINATION:

MEETING TIME:
(HH : MM)

Reserve Taxi

Return to Home Page

---

myTaxiService

History of Booked Taxis:

Reservations can be cancelled up to 10 minutes before the scheduled ride:

Taxi Code: _ _ _ _ Date: _ _ / _ _ / _ _
Origin: _ _ _ _

Taxi Code: _ _ _ _ Date: _ _ / _ _ / _ _ ✖
Origin: _ _ _ _ Destination: _ _ _ _
Meeting Time: _ _ : _ _

Taxi Code: _ _ _ _ Date: _ _ / _ _ / _ _ ✖
Origin: _ _ _ _ Destination: _ _ _ _
Meeting Time: _ _ : _ _

Taxi Code: _ _ _ _ Date: _ _ / _ _ / _ _
Origin: _ _ _ _ Destination: _ _ _ _
Meeting Time: _ _ : _ _

Taxi Code: _ _ _ _ Date: _ _ / _ _ / _ _
Origin: _ _ _ _

Taxi Code: _ _ _ _ Date: _ _ / _ _ / _ _
Origin: _ _ _ _

Taxi Code: _ _ _ _ Date: _ _ / _ _ /_.
Origin: _ _ _ _ ↓

Return to Home Page

---

myTaxiService

Looking for an available Taxi...

Please Wait...

---

myTaxiService

✔

REQUEST CONFIRMED
_ _ / _ _ / _ _ **AT** _ _ : _ _ : _ _

TAXI CODE:
_ _ _ _ _ _ _

WAITING TIME:
_ _ : _ _ : _ _

Go to Booked Taxis

Return to Home Page

---

myTaxiService

✔

RESERVATION CONFIRMED

MEETING DATE: _ _ / _ _ / _ _ _ _

ORIGIN: _ _ _ _ _ _ _

DESTINATION: _ _ _ _ _ _ _

MEETING TIME: _ _ : _ _

Remember to check the TAXI CODE 10 minutes before the scheduled MEETING TIME in Booked Taxis !

Go to Booked Taxis

Return to Home Page

The Android Auto user interface of myTaxiService is designed to meet all the functionalities needed by taxi drivers. It is the default application that starts when the engine of the car is turned on and the driver cannot close it. When a driver begins their daily shift, they have to insert the username and password the government has provided them on a Login Page, in order to access the service and include their taxi into the system. Then, the driver is redirected to the Home Page (shown below), which provides them with the possibility to switch on and off their availability. If the availability is off the central screen remains clear, else the system can allocate a ride to the driver, which is displayed with the possibility of confirmation and rejection along with the location of the passenger (and also the destination in case the ride is reserved, else the field is empty). If the driver rejects the screen turns clear again, else Google Maps is automatically opened with the directions to the meeting location. During a ride, from the moment of confirmation, the availability switch is forced in the off state by the system and it cannot be used (else, an error message appears) until the driver taps on the "End Ride" button (this is to prevent that the system allocates a ride to a taxi that is still completing another one). When the passenger is aboard, the driver searches for the directions to the desired destination (declared at boarding time in case of requested ride). When the taxis reaches destination, the driver can close Google Maps and then the central screen of the user interface turns clear again.

(2) DOCUMENTATION

These are the documents about myTaxiService that will be released:

• **RASD (Requirement Analysis and Specification Document)**: contains the description of the scenarios, the use cases that describe them, and the models describing requirements and specification

• **DD (Design Document)**: contains a functional description of the system, and any other view useful to provide

(3) SOFTWARE SYSTEM ATTRIBUTES

- **Reliability:** System shall have an availability of 99.99% (four nines), which implies a 52.56 minutes downtime per year. The reliability of the system is strictly related to the reliability of the server it runs on.
- **Availability:** The system is running 24/7 so that a user can always request or reserves a taxi.
- **Capacity:** the system should be able to manage 1000 requests/second and manage at least 4000 connected users (passengers and taxi drivers) at a given time

- **Security:** All the communications between server and clients must be protected by strong encryption using the SSL protocol. Users' passwords must be hashed, salted and then stored in the database

- **Portability:** The back-end server software will be written in Java so that it can be run on every platform that supports the Java Virtual Machine. The web application shall run on every modern browser. The mobile application for users must be supported by the last two major versions of Android and iOS. The mobile application for taxis must be supported by Android Auto

- **Robustness:** If the taxi driver loses the GPS connection for a while, the ride is not canceled, available taxis remain in the queues and updates are simply suspended

4. Specifications

- **Management of taxi queues:** the system keeps a FIFO (First In First Out) queue of available taxis for each zone. When a taxi becomes available, the system will insert it in the queue of the relative zone using the location provided by the in-built GPS of the car. When a taxi becomes non-available, the system will remove it from the relative queue. When a taxi driver rejects a ride, the system will move it to the last position of the relative queue

- **Registration of a Guest:** the system allows a guest to register once with the same email address. If they try to use it again in the registration form, the system will not allow signing up. Once a user has correctly signed up, they will be automatically logged in into the system for the first time

- **User Login:** the system allows a user to sign in using email address and password specified in the registration form. If a user inserts a wrong couple username-password (i.e. they are not in the database), the system will not allow signing in

- **Taxi ride request:** if the user uses the web application it will be required to insert their location, whereas with the mobile application the position is automatically acquired with the smartphone's in-built GPS. The system will propose the ride to the first taxi queueing in the zone where the request comes from. If no taxi is available or everyone in that zone rejects the ride, the system will propose it to the first taxi queueing in the nearest zone. Once the ride is

allocated to taxi that has accepted it, the system will send the taxi code and the waiting time to the user in a confirmation page. A User cannot make requests if they have already assigned a taxi code for a previous, still ongoing, ride (request or reservation)

- **Taxi ride reservation:** the user is required to insert meeting date, meeting time, origin and destination of the ride. If the inserted meeting time is less than two hours from the moment of the reservation, the system will not allow the user to confirm it. The system will store the reservation of the user in a database and show a confirmation page to them. 10 minutes before the scheduled meeting time, as far as the assignment of a taxi is concerned, the reservation is managed like a normal taxi ride request and the system will inform the user about the code of the incoming taxi with a push notification. The system allows the user to cancel the reservation up to 10 minutes before the meeting time

- **List of Booked Taxis:** the system keeps a history of all the concluded and pending requests and reservations of a user and allows them to visualize it in a specific section of the application (web and mobile).

- **Taxi Driver Login:** the system allows a taxi driver to sign in inserting username and password already provided. If a taxi driver inserts a wrong couple username-password (i.e. they are not in the database), the system will not allow signing in.

- **Confirmation/Rejection of a ride:** the system allows a taxi driver to accept or reject a proposed ride. When the taxi driver accepts a request, the route to the passenger is displayed on the on-board screen. If the driver refuses, their taxi will be moved to the last position in the queue of the zone where it is. The taxi driver has only 10 seconds to answer when a ride is proposed on the screen, after which the it is automatically rejected

- **Update taxi driver availability:** the system allows the taxi driver to set their availability on or off. When the taxi driver accepts a ride, their status will be automatically set to non-available and this cannot be changed until the driver informs the system that he has reached destination (with an "End Ride" button). When the taxi driver is around a certain zone and ready to accept rides, they will switch their status into available and they can turn it off whenever they want, except during a ride.

- **Scalability to additional services:** the system provides API (Application Programming Interfaces) for developers. For instance, in future it could be implemented an additional service that allows to keep track of fake requests (neglected here by assumption) to ban malicious users or a taxi sharing service

5. Scenarios

AN USEFUL ADVICE:

This evening, John is going to meet Chiara at her place for dinner but he does not know how since his parents are gone away for the weekend with the family car. He is walking on the way to her home when he reminds that his friend Marco used a new application, "myTaxiService", the day before and recommended it to him. Therefore, he decided to try it downloading and installing it on his IPhone 6S from the App Store. Once opened, he taps on "Create Account" since it is his first time using it. Then, he inserts the requested personal information and taps on "Sign Up". Now John is inside the system and can make a taxi request. The system forwards it to Luca that has just become available after a short break. Luca, however, does not notice the ride on his display in time. In fact, 10 seconds has passed and the request to his taxi has been automatically rejected. Therefore, the ride is proposed to Leonardo that accepts it and goes taking John at the meeting location.

MIKE'S HAPPY HOUR:

Mike went to a happy hour with his friends and now he wants to go back home. He takes his LG Nexus 5X, opens the application "myTaxiService", that he has already used many times, and signs in into the system. He selects "Request a Taxi" and the system acquires his location through the in-built GPS. Then, when he confirms, the system informs him that it is searching for an available Taxi. Chester, that has just begun his shift and switched his availability on, sees on the display of its Android Auto device that there is a ride request ready for him. Chester accepts the ride and goes taking the passenger. Mike sees a confirmation page with the taxi code and a waiting time of 8 minutes. After a couple of minutes, Mike forgets the taxi code and goes to the "Booked Taxis" section of the application to see it. Chester takes Mike in and asks him the desired destination. Mike arrives in front of his house, pays Chester and gets off the car. Right after, Chester decides to remain in the same zone and switches on again his availability, waiting for the system to assign him another ride.
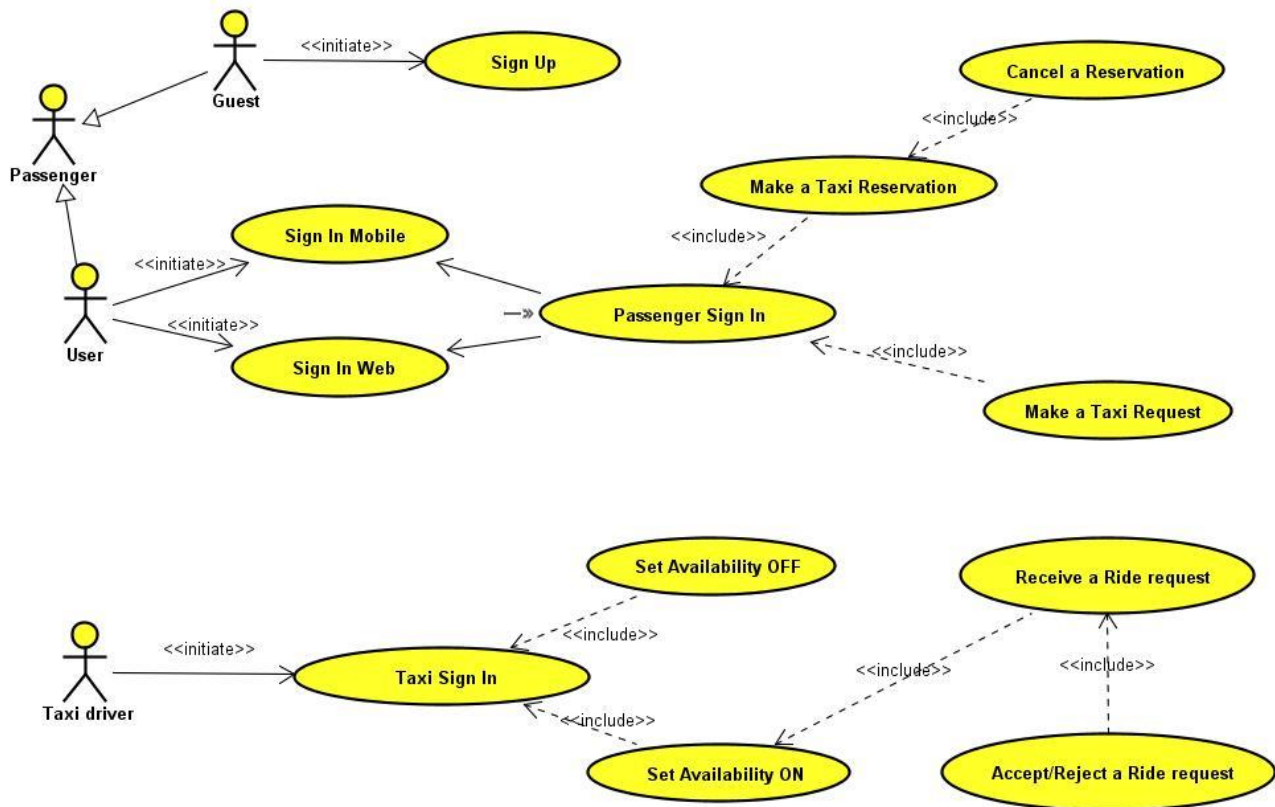
AURORA'S HOLIDAY:

Aurora is on holiday in Big_City and she has planned to go back home the DD/MM at HH:MM by train. The day before that date, she decides to make a taxi reservation since she will need a lift to the station. Therefore, she takes her laptop, opens the browser and goes to "www.mytaxyservice.com" website, and reserves a taxi for the following day. She immediately visualizes the confirmation page with all the specified details. Unfortunately, half an hour before the meeting time, Aurora is informed that the train will arrive with a delay of two hours. Consequently, she decides to cancel the taxi reservation from the "Booked Taxis" section of the mobile application installed in her Samsung Galaxy S6, since she has already switched off the laptop. An hour after, she decides to make a taxi request and go to the station in advance, waiting for the train on the platform. Since it is the rush hour, a taxi with a waiting time of 10 minutes has been assigned to her. In the end, she managed to arrive at the station and after a while, the train arrived.

THE LUNCH BREAK:

Valentino is a taxi driver of Big_City that has just finished a ride and moved to the zone B, setting his status as available and waiting. After 15 minutes in which no ride request from the system have arrived, he decides to switch his status into non-available and have a lunch break. Afterwards, he turns on again the availability and keeps waiting. After 5 minutes, the system proposes him a ride, but he immediately notices that the passenger is not in his current zone, but in the one beside it. Valentino confirms the ride and after taking the passenger to destination, he informs the system about the end of the ride and goes back to the zone B. Then, he stops in a parking lot and informs the system that he is available again.

6. UML models

6.1 Use Case diagram

6.2 Use Case description

Name	**A new passenger signs up into the service from the mobile application**
Actors	Guest
Entry Conditions	The Guest wants to register to "myTaxiService"
Flow of events	- The Guest opens the mobile application for the first time ever - The System shows to the Guest the Login page - The Guest taps on "Create an account" - The System shows the Registration page - The Guest inserts their email address, password - The Guest taps on Sign up

	• The System checks the correctness of the inserted information, confirms and redirects the Guest, that now is a User, to the Home page
Exit conditions	The information are stored in the database. Now, the Guest can login into the system, becoming a User
Exceptions	If the Guest inserts an invalid email address, or one already present in the database, the System shows an error message

Name	**A passenger logs in into the service from the web application**
Actors	User
Entry Conditions	The User wants to log in into the system
Flow of events	• The User opens a browser from their laptop and go to www.mytaxyservice.com • The system shows the Login page • The User inserts email address and password • The User clicks on the Sign in button • The system checks the correctness of the inserted information, confirms and redirects the User to the Home page
Exit conditions	The User is logged in into the system
Exceptions	If the User inserts an invalid couple email address-password, or one not present in the database, the system shows an error message

Name	**A passenger makes a taxi ride request from the mobile application**
Actors	User
Entry Conditions	The User is logged in into the service from the mobile application and wants to request a Taxi
Flow of events	• The User taps on "Request a Taxi" in the Home page of the mobile application

	• The System shows the Taxi Ride Request page
	• The User sends the request tapping on the relative button
	• The System acquires the GPS position of the User
	• The System finds a taxi that accepts the Request
	• The System stores the Request
	• The System shows a confirmation page with the code of the incoming taxi and the waiting time to the User
Exit conditions	A Taxi is properly assigned to the User
Exceptions	The User has already an active request (or reservation with an assigned Taxi code) that is not yet completed, the system shows an error message

Name	**A passenger makes a taxi ride reservation from the web application**
Actors	User
Entry Conditions	The User is logged in into the service from the web application and wants to reserve a Taxi
Flow of events	• The User clicks the button "Reserve a taxi" in the Home page of "www.mytaxiservice.com" website
	• The website shows the Taxi Ride Reservation page
	• The User compile the form with:
	o Meeting Date
	o Origin
	o Destination
	o Meeting Time
	• The User confirms clicking the button "Reserve a Taxi"
	• The System stores the Reservation
	• The System shows a Confirmation page
Exit conditions	The User immediately receives the confirmation from the system

Exceptions	If the specified Meeting Time is within two hours from the moment of the reservation, the reservation cannot be made (an error message appears upon clicking the "Reserve a Taxi" button) and an error message is displayed. If Origin is equal to Destination or some information are missing, an error message is displayed

|	
Name	**A passenger cancels a taxi ride reservation**
Actors	User
Entry Conditions	The User is logged in into the service and wants to delete a ride reservation
Flow of events	- The User taps on "Booked Taxis" from the Home page of their mobile application, which allows him to watch and monitor the list of all their booked rides (requests and reservations) scrolling down the page - The System shows him the relative page - The User picks one reserved ride and taps on the relative red "X" icon. A confirmation message appears with a "yes-no" choice. The User taps "yes" - The System deletes the Reservation - The System shows a message of successful cancellation
Exit conditions	The reservation is properly deleted
Exceptions	If the reserved ride takes place less than 10 minutes before the User opens the "Booked Taxis" section, it cannot be cancelled since the red "X" will not be present

Name	**A Taxi driver sets their availability on**
Actors	Taxi driver
Entry Conditions	The Taxi driver is logged in into the service and currently in non-available state

Flow of events	• In the Home page of the Android Auto application the Taxi driver can tap on a switch that allows him to make themselves available • The system adds the Taxi to the queue of its current zone
Exit conditions	The Taxi is added to the queue and it is ready to be picked up from the system when a new request or reservation comes up
Exceptions	The Taxi driver cannot switch on their availability during a ride. If they try to do it, the system shows an error message

Name	**A Taxi driver sets their availability off**
Actors	Taxi driver
Entry Conditions	The Taxi driver is logged in into the service and currently in available state
Flow of events	• In the Home page of the Android Auto application the Taxi driver can tap on a switch that allows him to make themselves non-available • The system remove the Taxi from the queue of its current zone
Exit conditions	The Taxi is removed from the queue
Exceptions	There are no exceptions

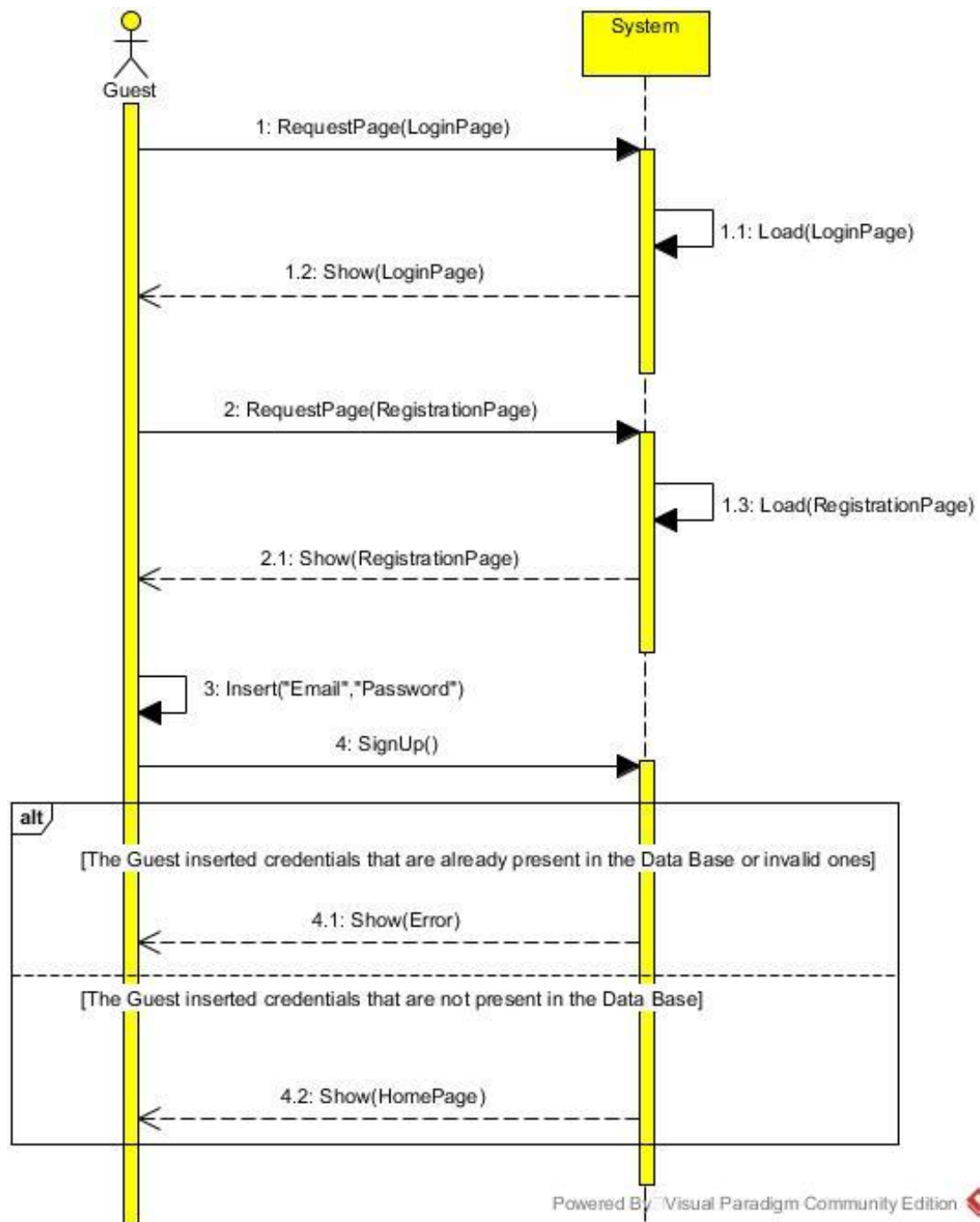| Name	**A Taxi receives a ride from the system**
Actors	Taxi driver
Entry Conditions	The Taxi driver is available and ready to manage a new ride provided by the system. Their Taxi is the first in the availability queue of its zone so an available ride is coming soon
Flow of events	• The Taxi driver is notified by the system on their Android Auto application, which shows all the information about the ride to be performed • The Taxi driver can accept or reject the ride tapping on the

|	relative buttons
Exit conditions	• The Taxi driver accepts the ride • Otherwise, if the Taxi driver rejects, the system will move their Taxi in the last position of the relative availability queue
Exceptions	If the taxi driver does not accept or reject the ride within 10 seconds, the request is automatically rejected

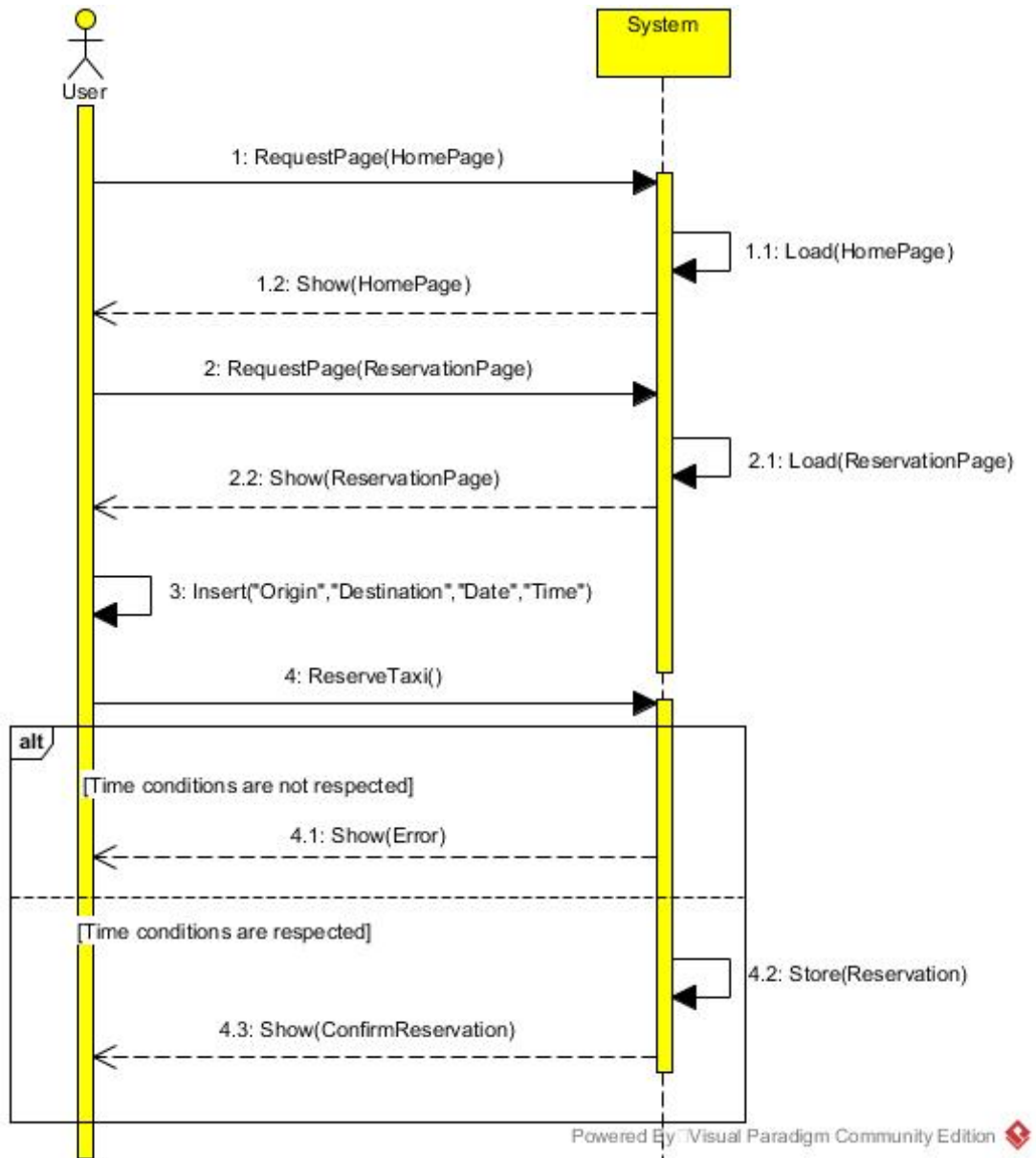## 6.3 Sequence diagrams

USER SIGN IN
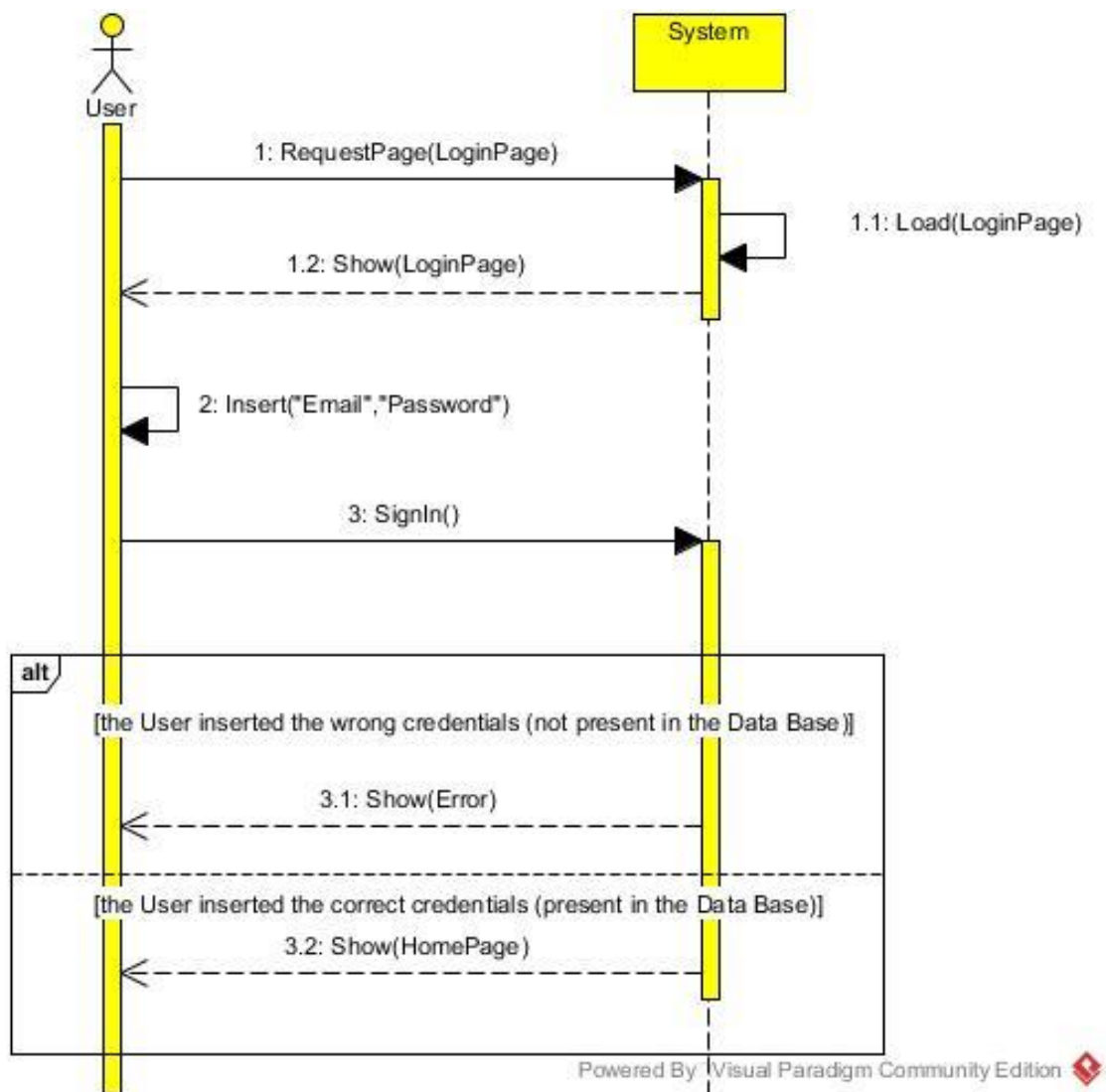
GUEST REGISTRATION

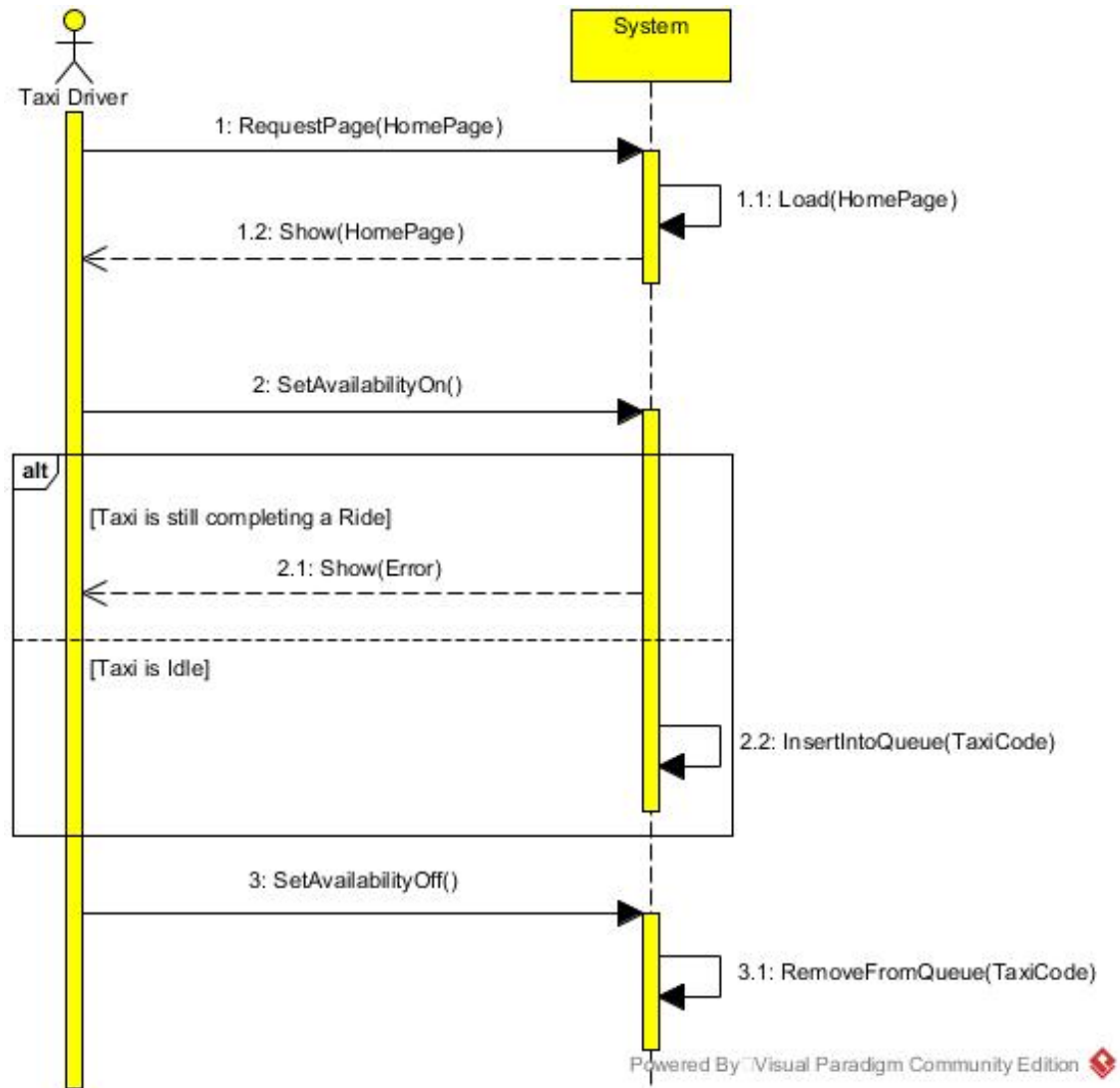# USER MAKES A RIDE REQUEST AND A TAXI RECEIVES A RIDE

In this Sequence diagram, there is a request made by a user and the system that sends a ride to a taxi, **which can be it or another one.**
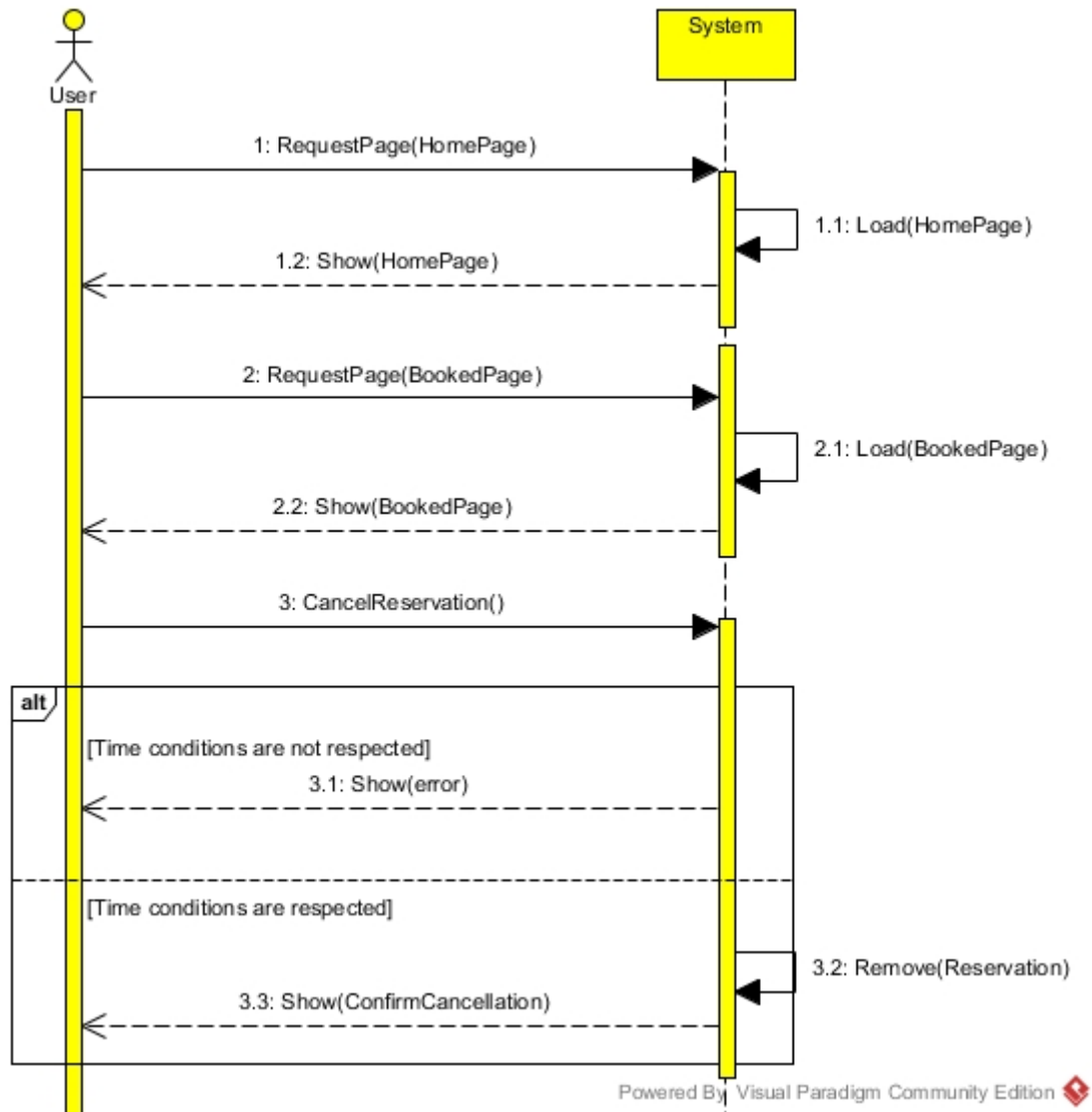
USER MAKES A TAXI RIDE RESERVATION
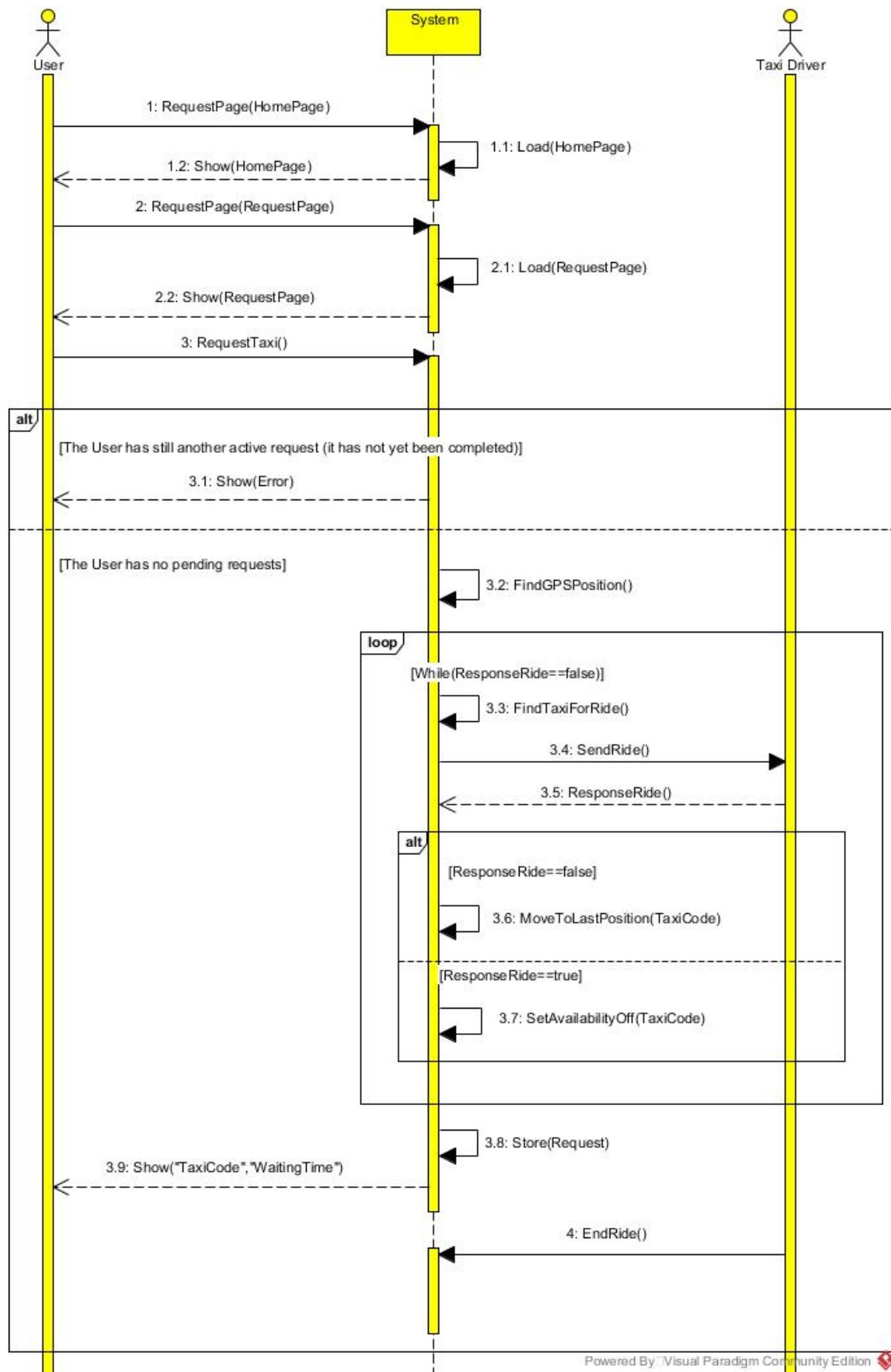
Powered By Visual Paradigm Community Edition

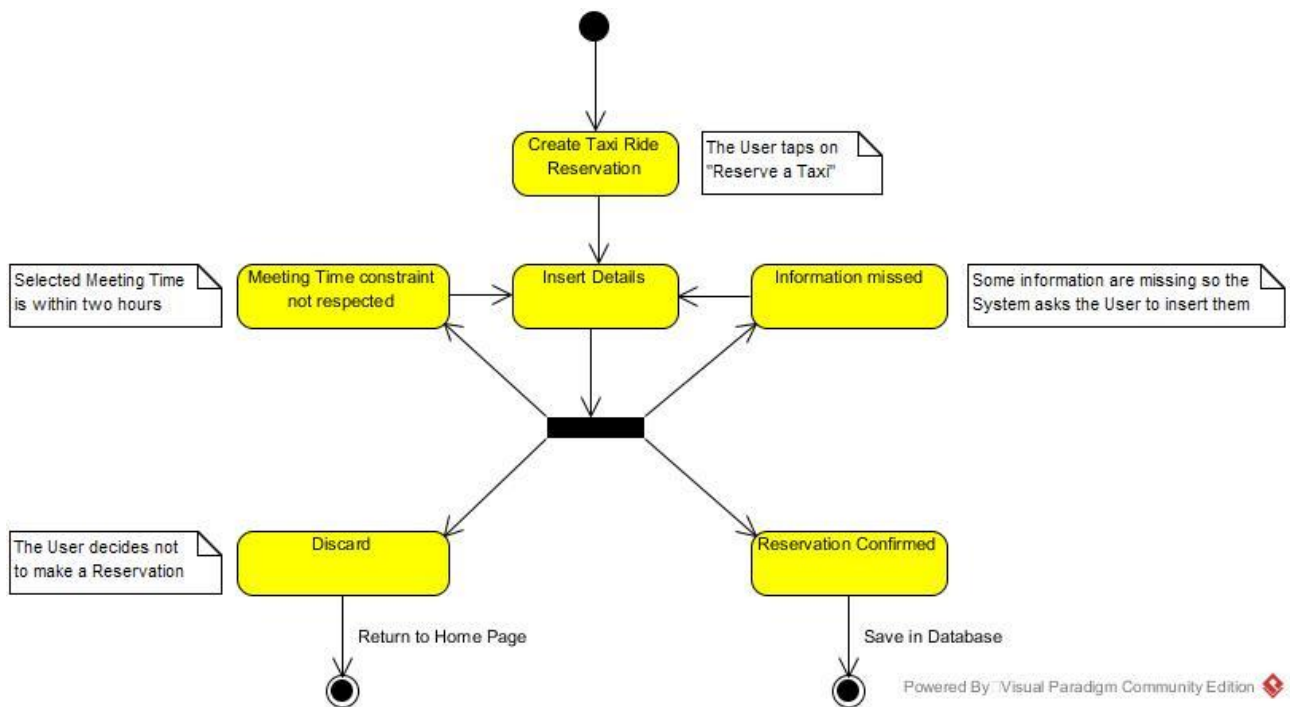# TAXI DRIVER UPDATES THEIR AVAILABILITY (ON - OFF)

USER VIEWS LIST OF BOOKED TAXIS AND CANCELS A TAXI RESERVATION

6.4 State Chart diagrams
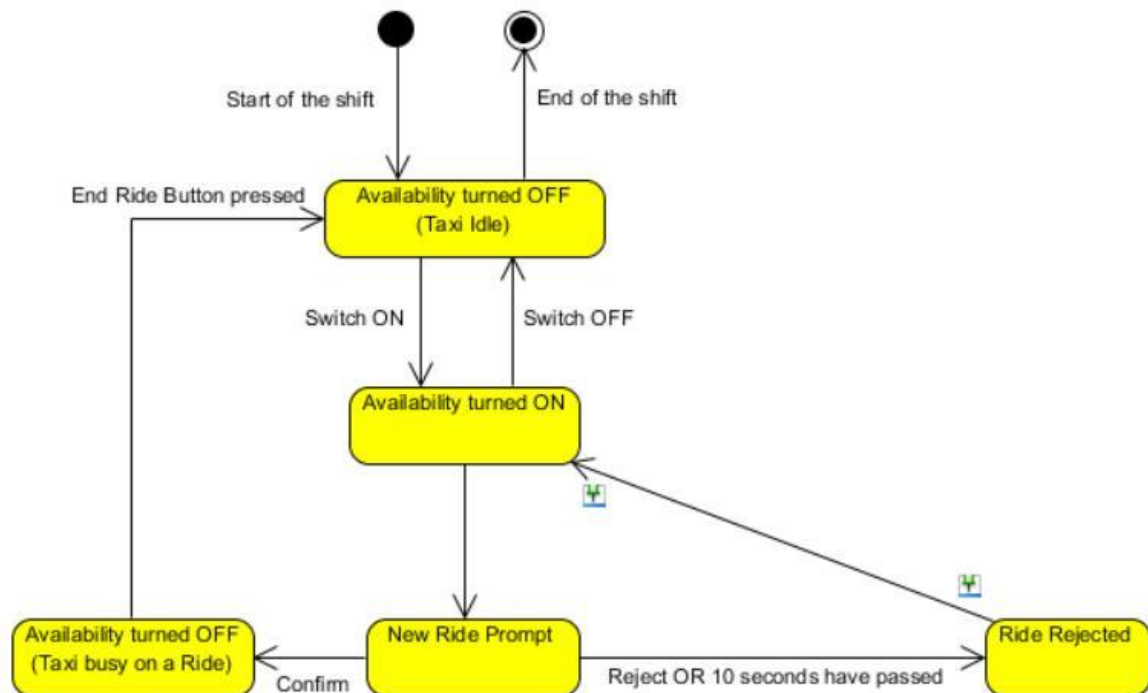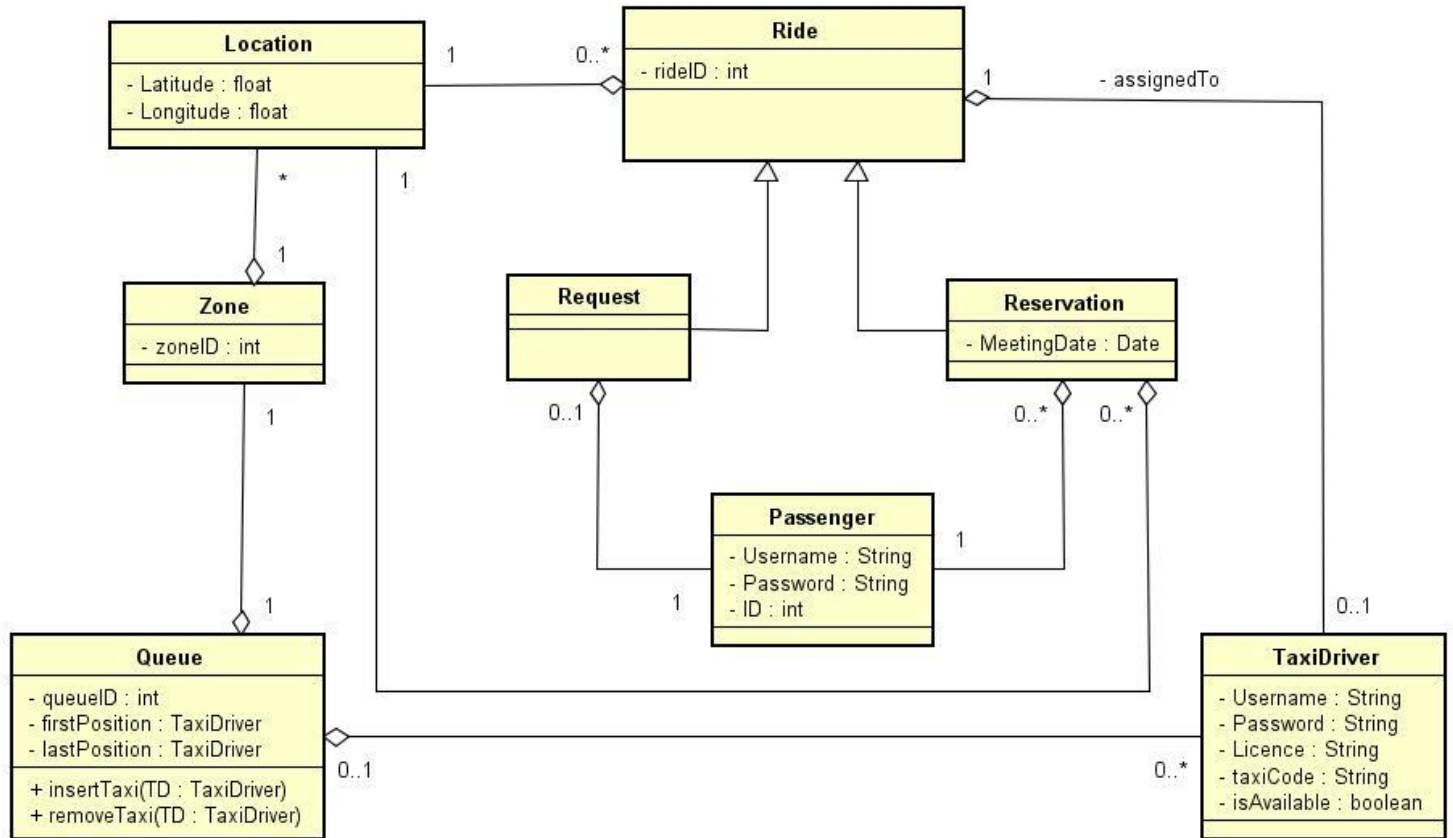
CREATE TAXI RIDE RESERVATION

UPDATE TAXI AVAILABILITY

6.5 Class diagram

Location
- Latitude : float
- Longitude : float

Ride
- rideID : int

- assignedTo

Zone
- zoneID : int

Request

Reservation
- MeetingDate : Date

Passenger
- Username : String
- Password : String
- ID : int

Queue
- queueID : int
- firstPosition : TaxiDriver
- lastPosition : TaxiDriver

+ insertTaxi(TD : TaxiDriver)
+ removeTaxi(TD : TaxiDriver)

TaxiDriver
- Username : String
- Password : String
- Licence : String
- taxiCode : String
- isAvailable : boolean

7. Alloy

7.1 Modeling

SIGNATURES

```
module mytaxiservice
open util/boolean
sig string{}
sig float{}
sig TaxiDriver{
    username : one string,
    password : one string,
    taxicode : one string,
    licence : one string,
    isAvailable : one Bool
}
sig Passenger{
    username : one string,
    password : one string
}
abstract sig Ride{
    idRide:one int
    origin : one Location,
    has : one Passenger,
    assigned_to : lone TaxiDriver
}
sig Request extends Ride{}
sig Reservation extends Ride{
    date : one Int,
    destination : one Location
}
sig Location
{
    latitude : one float,
    longitude : one float
}
sig Zone{
    idzone : one Int,
    queue : one Queue,
    vertex: some Location
}{ #vertex=4
}
sig Queue{
    idqueue : one Int,
    available_drivers : set TaxiDriver,
    firstPosition : lone TaxiDriver,
    lastPosition : lone TaxiDriver }
```

FACTS

```
fact Location{
// Two locations cannot have the same latitude and longitude
    no disj l1,l2:Location | l1.latitude=l2.latitude and l1.longitude=l2.longitude
}
fact Zone{
// Two or more zones cannot have the same idzone
    no disj z1,z2:Zone | z1.idzone=z2.idzone
// Two or more zones cannot be associated to the same queue
    no disj z1,z2:Zone | z1.queue = z2.queue
// Two or more zones cannot have the same vertex
    all disj z1,z2:Zone | z1.vertex!=z2.vertex
}
fact Queue{
// Two or more queues cannot have the same idzone
    no disj q1,q2:Queue | q1.idqueue=q2.idqueue
// Taxi drivers that are in first and last position of a queue must be among the available drivers of that queue (so, in the queue)
    all q:Queue,t:TaxiDriver | (q.firstPosition=t or q.lastPosition=t) implies t in q.available_drivers
// If a queue is not empty, the first position and the last position Taxi driver must exist in it
    all q:Queue | #q.available_drivers>0 implies (q.firstPosition!=none and q.lastPosition!=none)
// If a queue contains more than one taxi driver, the Taxi driver in first position is different from the one in last position
    no q:Queue | #q.available_drivers>1 implies (q.firstPosition=q.lastPosition)
// One queue is associated to one zone
    all q:Queue | one z:Zone | q in z.queue
}
fact TaxiDriver{
// Two or more Taxi drivers cannot have the same username
    no disj t1,t2:TaxiDriver | t1.username=t2.username
// Two or more Taxi drivers  cannot have the same licence
    no disj t1,t2:TaxiDriver | t1.licence=t2.licence
// Two or more Taxi drivers cannot have the same taxicode
    all disj t1,t2:TaxiDriver | t1.taxicode!=t2.taxicode
// One taxi driver can be at most in one queue at the same time
    no t:TaxiDriver | some disj q1,q2: Queue | t in q1.available_drivers and t in q2.available_drivers
// If a taxi driver is assigned to a request, he must not belong to any queue
    all t:TaxiDriver,r:Ride,q:Queue | r.assigned_to=t implies t not in q.available_drivers
// A taxi driver can serve at most one request at the same time
    no disj r1,r2:Ride | r1.assigned_to=r2.assigned_to and r1.assigned_to!=none
// If a Taxi driver has an assigned ride, they are not available
    all t:TaxiDriver,r:Ride | t in r.assigned_to implies t.isAvailable=False
// If a Taxi driver is in a queue, they are available
    all t:TaxiDriver,q:Queue | t in q.available_drivers implies t.isAvailable=True
}
fact Passenger{
// Two or more Passengers cannot be have the same username
    no disj p1,p2:Passenger | p1.username=p2.username
// A Passenger cannot have two rides with the same date
    all disj r1,r2:Ride | r1.has=r2.has implies r1.date!=r2.date
// A Passenger can do one request at once
    no disj r1,r2:Request | r1.has = r2.has
}

fact Ride{
// Two or more Ride cannot have the same idRide
    all disj r1,r2:Ride | r1.idRide!=r2.idRide
// ride that has an origin different from destination
    all disj r1,r2:Ride | r1=r2 implies r1.origin!=r2.destination
// A Passenger can have only one ride assigned at once
    all disj r1,r2:Ride | (r1.has=r2.has and r1.assigned_to!=none) implies r2.assigned_to=none
// If a Passenger has one request they cannot have a reservation already assigned to a taxi
    all r1:Request,r2:Reservation | r1.has=r2.has implies r2.assigned_to=none
}
```

ASSERTIONS AND PREDICATES

We run the simulation with only the strictly needed fields in order to generate a readable representation of two possible worlds:

```
assert TaxiAssignedToTwoOrMoreQueue{
    no t:TaxiDriver | some  disj q1,q2: Queue | t in q1.available_drivers and t in  q2.available_drivers
}

check TaxiAssignedToTwoOrMoreQueue

assert PassengerWithTwoRideAssigned{
    no disj r1,r2:Ride | r1.has=r2.has and r1.assigned_to!=none and r2.assigned_to!=none
}

check PassengerWithTwoRideAssigned

pred MoreRide{
    #Ride>5
    #Passenger=3
    some disj r,r1:Ride | r.assigned_to!=none and r1.assigned_to!=none
}

run MoreRide for 6 but 8 Ride

pred GenericWorld {
    #Queue>2
    #Ride>2
    #Passenger>2
    some q:Queue | q.available_drivers!=none
    some disj r,r1:Ride | r.assigned_to!=none and r1.assigned_to!=none
}

run GenericWorld for 10
```

7.2 Alloy analyzer

```
Executing "Check TaxiAssignedToTwoOrMoreQueue"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  1266 vars. 96 primary vars. 1927 clauses. 583ms.
  No counterexample found. Assertion may be valid. 87ms.

Executing "Check PassengerWithTwoRideAssigned"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  1272 vars. 93 primary vars. 1978 clauses. 115ms.
  No counterexample found. Assertion may be valid. 38ms.

Executing "Run MoreRide for 6 but 8 Ride"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  6652 vars. 326 primary vars. 10998 clauses. 297ms.
  Instance found. Predicate is consistent. 183ms.

Executing "Run GenericWorld for 10"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  15963 vars. 740 primary vars. 27325 clauses. 530ms.
  Instance found. Predicate is consistent. 397ms.

4 commands were executed. The results are:
  #1: No counterexample found. TaxiAssignedToTwoOrMoreQueue may be valid.
  #2: No counterexample found. PassengerWithTwoRideAssigned may be valid.
  #3: Instance found. MoreRide is consistent.
  #4: Instance found. GenericWorld is consistent.
```
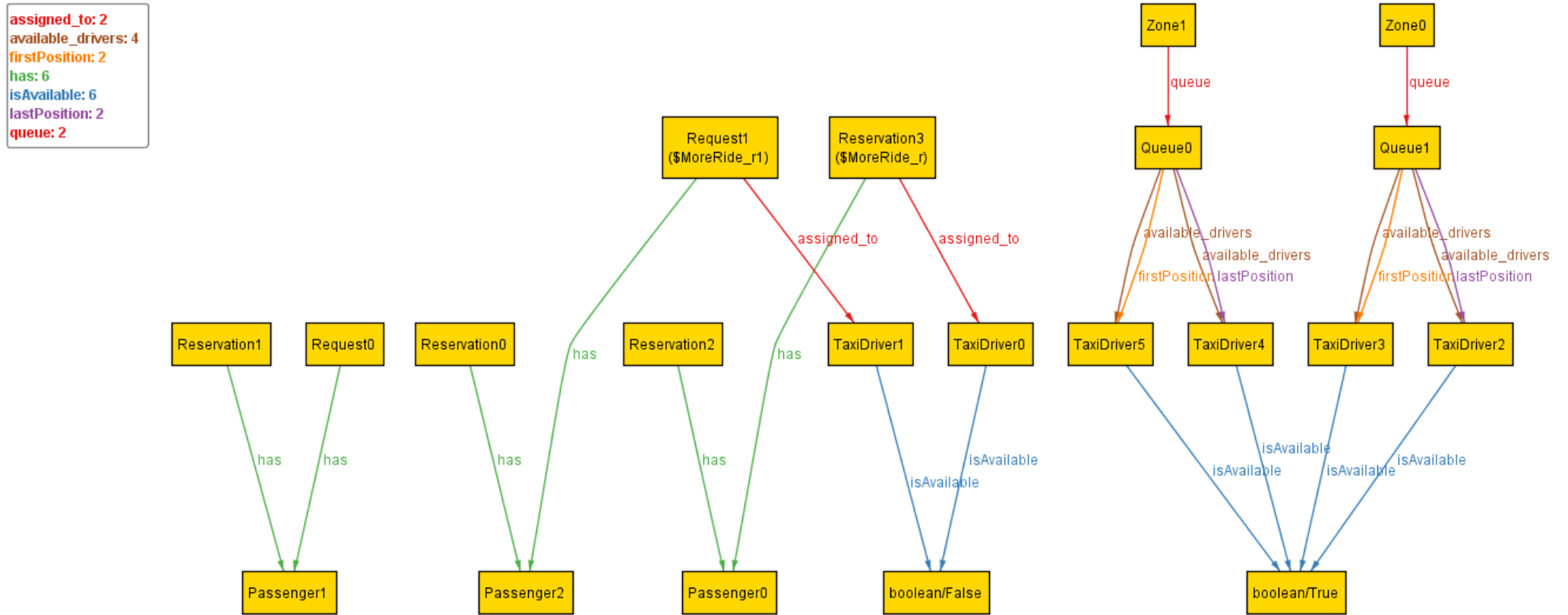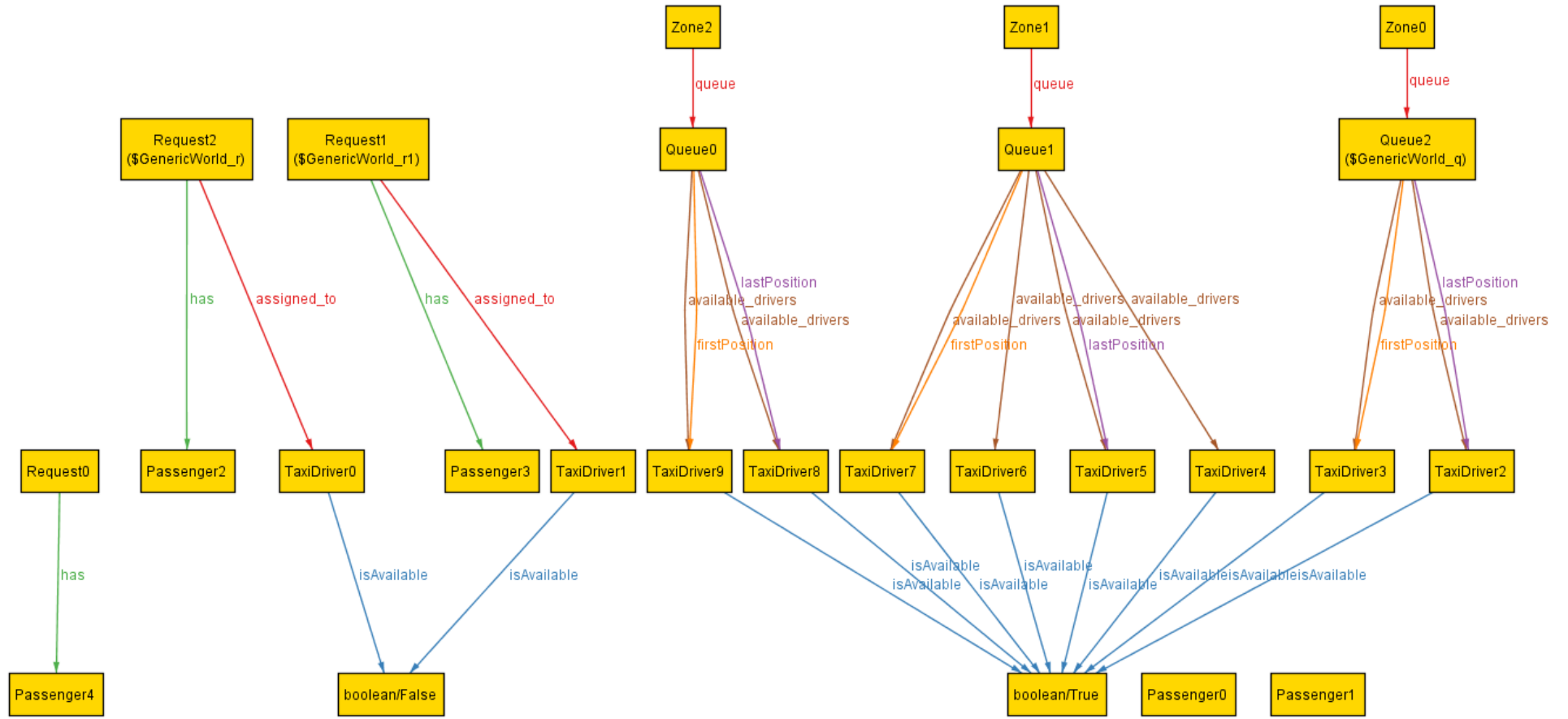
7.3 Worlds generated

8. Used tools

- **Microsoft Word 2013:** to redact and format the document

- **Proto.io:** to create the user interface mockups

- **Alloy Analyzer 4.2:** to explore the model of our system in terms of consistency

- **Astah Professional:** to create Use Cases Diagram and Class Diagram

- **Visual Paradigm 12.2 Community Edition:** to create State Chart Diagrams and Sequence Diagrams

9. Hours of work

Since we are neighbors, we have worked **together almost all the time** at each other's home and **equally shared all the tasks and efforts**. We have worked on this document **for a total of 48 hours.**