Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project

myTaxiService

Requirements Analysis and Specifications Document

Ver. 4

Alessio Martorana - 860584

1 Introduction

1.1 Purpose

This document represent the Requirement Analysis and Specification Document (RASD). The main goal of this document is to completely describe the system in terms of functional and non-functional requirements, analyze the need of the customer to modelling the system, show the constraints and the limit of the software and simulate the typical use cases that will occur after the development. This document 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 customer and the developer.

1.2 Actual System

I suppose that there is no actual system and, hence, the entire system requested is to be created without using or modifying a previous system.

1.3 Scope

The aim of the project is to create a system which will optimize the taxi service of a big city. The main goals of the application are to simplify the access of passenger to the service and to guarantee a fair management of taxi queues.

The system should be able to register new users with their personal information: name, surname, email and password. The system should be also

able to keep track of every taxi driver in the city, dividing the city in “taxi zones” (of approximately 2 km²) and computing the position of every taxi in each zone thanks to the GPS position acquired from a web application provided to each taxi driver. Each zone is associated to a queue of taxi codes (one code is associated to one and only one taxi) in order to handle users requests. The system should also allow each taxi driver to confirm or decline a request from a user.

A user should be able to request a taxi and the system should notify the user if a taxi has confirmed the request and, in this case, the system should send to the user taxi code and waiting time.

The system should also manage taxi queues so that when a taxi request is received from a zone, the system forwards it at the first taxi in the queue associated to that zone; at this point, if the taxi confirms the request, a confirmation to the user is sent, otherwise – if the taxi decline – the system should forward the request to the taxi in the second position of the queue and move the taxi which declined in the last position of the queue.

The system should also provide some programmatic interfaces to allow the development of additional services on top of the basic system.

1.4 Goals

The system will have to provide the following features:

The user should be able to:

- [G1] Sign up into the system;
- [G2] Log-in into the system;
- [G3] Request a taxi;

- [G4] Be notified by the system about if the request has been accepted or declined and – in case of request confirmation – be notified about taxi code and waiting time;
- [G5] Confirm or cancel his taxi request after the system has confirmed his request and sent taxi code and estimated waiting time;

The Taxi driver should be able to:

- [G6] Log-in to the application through their web application;
- [G7] Inform the system about his availability;
- [G8] Receive user requests from the system;
- [G9] Confirm or decline users requests;

Moreover, the system should:

- [G10] Guarantee a fair management of taxi queues;
- [G11] Offer public Application Programming Interfaces to enable the development of additional services;

1.5 Definitions, acronyms, and abbreviations

In order to make the document more clear, in this paragraph some terms used in all the document are specified better.

- **Guest:** A person who hasn't either signed up or logged in to the application yet and, therefore, could only register to the application or log-in to the application;
- **User:** A person who has successfully logged in to the application and, hence, can use all the features of the application offered to users;
- **Taxi driver:** Driver of one of the taxis handled by the system;

- **Taxi management company (if not ambiguous, indicated as "company"):** Company which handles the taxi service of the city;
- **Taxi request (if not ambiguous, indicated as "request" or "user request"):** Indicates a taxi request performed by the user through the application;
- **Status (referred to taxi driver):** Indicates if the taxi driver is "Available" or "Unavailable", as specified in the next point;
- **Availability: "available"/"unavailable" (referred to taxi driver):** Indicates if a taxi driver is "at work" and so could be considered a valid taxi to which the system can forward a user request;
A taxi driver can change his status to "Unavailable" when he can't do his work for some reason (work breaks, damages, issues…)
- **Taxi response (if not ambiguous, indicated as "response or "answer"):** Is the answer of a taxi driver to a request, it can be "confirmed" or "declined" as specified in the next point;
- **Request confirmed (answer of a taxi driver to a user request forwarded by the system):** Indicates that the taxi driver is available and free from other tasks and can satisfy the user request;
- **Request declined (answer of a taxi driver to a user request forwarded by the system):** Indicates that the taxi driver cannot take the user request in charge;
- **City zone (if not ambiguous, indicated as "zone"):** Slices in which the system divides the city in order to manage taxis as said in the scope paragraph. Every zone measures about 2 km^2;
- **Zone queue (if not ambiguous, indicated as "queue"):** The queue in which taxis are placed. The zone queue is automatically created by the system in order to manage taxis, as said in the scope paragraph. The system creates one queue for any city zone;

1.6 Identifying Actors

- **Guest:** could only register to the application through a registration form or log-in to the application submitting his email address and password;
- **User**: A “Guest” turns into a registered user after a successful log-in to the application; since then he can request a taxi from his position using the application;
- **Taxi driver**: Drivers who drive taxis handled by the system. A ”Guest” turns into a taxi driver after a successful log-in from the mobile app;
- **External programmer**: A programmer who needs to extend the system and uses the Application Programming Interface offered by myTaxiDriver;

2 Overall description

2.1 Identifying Stakeholders

My stakeholder is the professor who assigned me the project, asking me to focus on the entire development process of a complex enterprise application, which includes requirement analysis, design, testing and project reporting phases. The professor also said that the implementation phase doesn't have to be performed and so I didn't analyze it. I assume that the main objective of the project is to demonstrate the ability to follow all the phases of the whole software development process.

I can imagine that some hypothetical stakeholders for the system could be the members of a company which handles the taxi service of the city, people living in the city and taxi drivers who work in the city.

I can also suppose that discussions with stakeholders have been done and surveys have been submitted to the stakeholders have been submitted.

2.2 Domain properties and assumptions

Domain properties and assumption that I suppose hold in the domain:

- A user gives correct information about his position when doing a request;
- If a request is confirmed by a taxi driver, a user remains settled in his position until the arriving of the taxi;
- At the arriving of the taxi which will serve a user request, the user must take the ride;
- Each taxi driver status, at any moment in his working time, must be set to "available" or "unavailable";

- When a taxi driver confirms, if the user confirms as well, a request he must go serve the request;
- At the receiving of a request, a taxi driver must confirm or decline the request at most in 30 second;
- When a taxi driver confirms a request and is notified the user confirmed as well, he begin to move immediately to serve the request, there is no time lapse between the confirmation and the departure of the taxi driver;
- From the moment a taxi driver receives a request, he reaches the user location at most in 15 minutes;
- If all taxis in a zone are busy in other requests, the user is informed about the impossibility to serve his request;
- All taxi drivers and corresponding taxi codes are already registered into a database - or, more in general, in some data collection - from which the application can read, and the credentials are sent to every taxi driver by e-mail and post mail.

I also suppose that the list of taxi drivers and taxi codes is always correct and up to date, for example because some employees of the taxi management company take care about filling the database with taxi drivers and correspondent taxi codes and constantly update the database as new taxi drivers begin their work career;

- Each taxi driver is associated with one and only one taxi;
- Each taxi is associated with one and only one taxi code, which is unique;
- Position data about taxis sent by the GPS are always correct and available at any time to the system;
- Each taxi driver must have a mobile device with android 6.0 operative system and GPS tracking. For example this mobile devices could be supplied by the taxi management company;

- Each taxi driver can run the mobile application (for example through a mobile support he owns) of the system, in order to send his position to the system and receive, confirm or decline user requests;
- Each taxi driver, when is in the “available” status, can immediately read the request, independently from the time in which the requests arrive;

2.3 Proposed System

The system will be composed by a server which will take care of dividing the city in city zones, management of the taxis through zone queues, receiving and forwarding of user requests and, more in general, of all the business logic; it will also guarantee the communication with users and taxi drivers, through either the web application or mobile app interfaces.
The other part of the system will be the clients, which can be of two types:

- User client
- Taxi driver client

User clients are clients used by user (either the web application or the mobile app) and propose to satisfy all the user goals described before.
Similarly, taxi clients are clients used by taxi drivers - in the form of a mobile app - and propose to satisfy all the taxi drivers goals described before. Two different mobile app are supplied to user and taxi drivers.

The web application will run into a web browser and – so – the interface will be included into the dynamically created web pages, the mobile app – instead - will provide graphical interface.

2.5 Software interfaces

- The operating system of the mobile devices has to be android 6.0, last android version at the drawing up of this document.
  www.android.com

2.6 Communication interfaces

- All messages among clients and server will be exchanged with HTTP/HTTPS protocol;

2.7 Product functions

Here a list of the major product functions of the system.

Functions offered to users:

- [F1] Sign-up;
- [F2] Log-in;
- [F3] Request of a taxi ride;

Functions offered to Taxi drivers:

- [F4] Log-in;
- [F5] User request confirmation or decline;

Functions offered to external programmers:

- [F6] Application Programming Interfaces to the system;

General functions of the system:

- [F7] Zone queue management;
- [F8] User requests handling;

2.8 User characteristics

Users and taxi drivers must be able to use all major functions of their mobile devices and of the android operative system, in order to correctly run and use myTaxiService mobile app.

2.9 Constraints

- The web application must be compatible with all most used browser: Google chrome, Opera, Mozilla Firefox and Internet Explorer;
- The mobile app must be runnable on android 6.0 Operative System;

3 Specific requirements

3.1 Functional requirements

User

- [R1] System shall store user information: name, surname, e-mail, password [G1, G2];
- [R2] System shall allow the user to sign-up into the system [G1];
- [R3] System shall not allow a user to register himself more than one time with the same username [G1];
- [R4] System shall check that username supplied in the registration must not be empty [G1];
- [R5] System shall check that password supplied in the registration must not be empty [G1];
- [R6] System shall allow the user to log-in into the system [G2];
- [R7] System shall allow the user to request a taxi ride [G3];
- [R8] If an user makes a request and a taxi driver accepts the request, the system shall notify the user that his request has been accepted [G4];
- [R9] In case a user request is accepted, system shall send the user the taxi code of the taxi serving the request and the estimated arriving time [G4];
- [R10] In case a user request is accepted, system shall prompt the user if he wants to confirm or decline the taxi ride [G5];

- [R11] The system shall decline a user's request if all the taxi drivers in the taxi drivers city zone queue have declined the request or the city zone queue is empty[G4];

Taxi Driver

- [R12] System shall allow the taxi driver to log-in into the system [G6];
- [R13] System shall allow the taxi driver to inform the system about his availability at any time [G7];
- [R14] The system shall allow taxi driver to be only in "available" or "unavailable" status at any time [G7];
- [R15] If a taxi driver is in the "available" status must be put in the queue of only one city zone [G10];
- [R16] If a taxi driver is in the "unavailable" status, he must not be put in any city zone queue [G10];
- [R17] If a taxi driver sets his status to "available" the system shall put him at the bottom of the queue which corresponds with the city zone where he is located [G10];
- [R18] If a taxi driver is in the "available" status, system shall allow taxi driver to receive user requests and to confirm or decline user's requests [G8];

External programmer

- [R19] System shall offer well documented Application Programming Interfaces in order to allow development of additional services [G11];

System in general

- [R20] If a taxi driver declines a user's request, system shall put him to the bottom of the queue and shall put the second taxi driver of the queue in the first position [G10];

3.2 Nonfunctional requirements

3.2.1 User interfaces

The user interfaces of either the web application and the mobile app are shown in this paragraph. All possible interactions actors can do with the system are described.

As images show, the general layout of the web application provides the name of the system in the top left of the window and the information about who is using the application in the top right of the window. Moreover in the left part of the window the name and/or the functionality of the actual page is shown.

In the mobile app the general layout provides again the name of the system in in the top left of the screen and the information about who is using the application in the top right of the screen; in the screen area just below this last area, the name and/or the functionality of the actual page is shown.

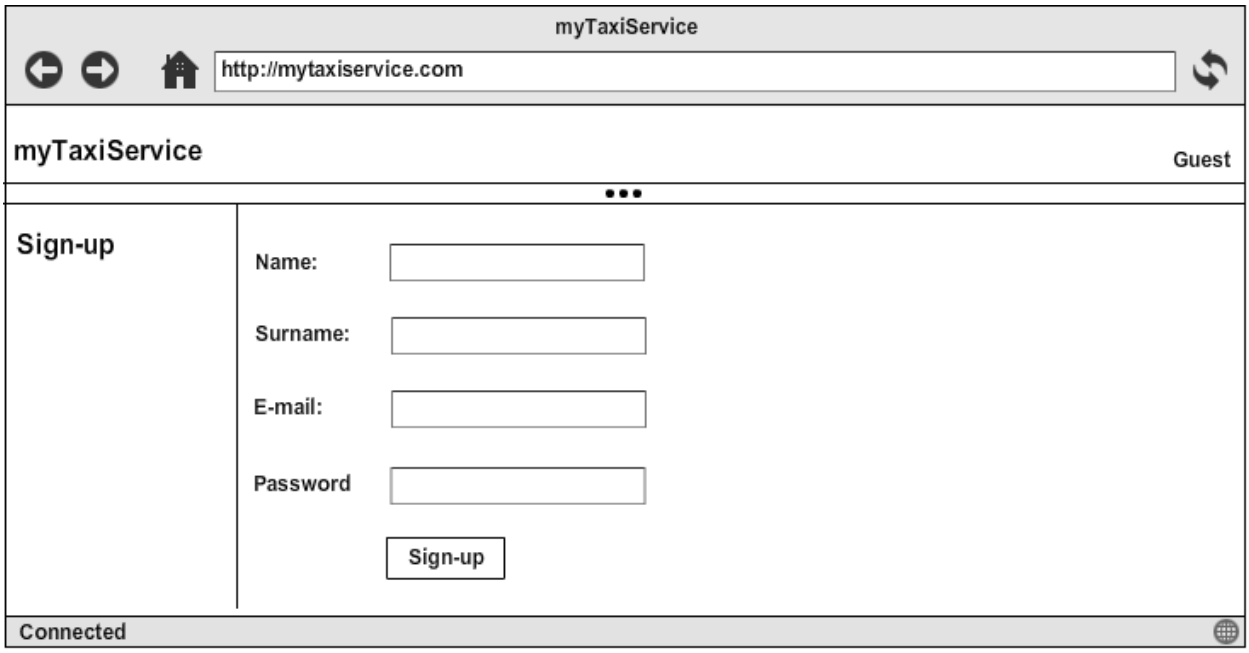
- **User**
  - **Guest web application home page:** is the page of the web application (which only users could use) in which a guest can choose if sign-up or log-in to take user privileges. The user can choose from the two links in the center.

- **Guest web application sign-up page:** is the page of the web application (which only users could use) in which a guest signs up into the system to and get the possibility to log in.

  The user signs up submitting his personal data:

  - *Name*
  - *Surname*
  - *E-mail*
  - *Password*

  and then click on the “Sign-up” button.



- **Guest web application log-in page:** is the page of the web application (which only users could use) in which a guest logs in into the system to take user privileges.

  The user logs in submitting:

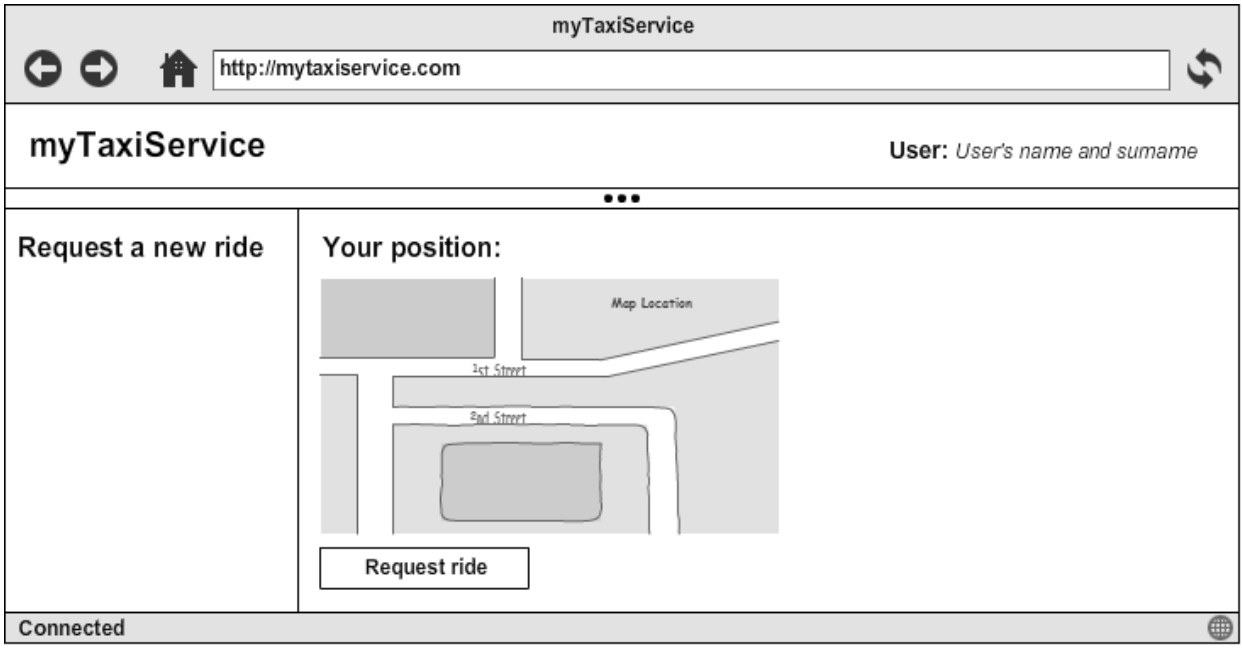
  - *E-mail*
  - *Password*

  and then click on the “Log-in” button.

- **User web application request a ride page:** is the page of the web application (which only users could use) in which a user can request a taxi ride. The user must specify
  - *The street where he's located*
  - *The nearest house number from the his position*

  and then click on the "Request ride" button.

- **User web application request a ride page (Map version):** Is the same of the previous interface, but in this case the user can see his position in a map.



- **Guest user mobile app home page:** is the page of the mobile app provided to users, in which a guest can choose if sign-up or log-in to take user privileges. The user can choose from the two links on the top.

- **Guest user mobile app sign-up page:** is the page of the mobile app provided to users in which a guest signs up into the system to be registered into the system and have the possibility to log in.

  The user signs up submitting his personal data:

  - *Name*
  - *Surname*
  - *E-mail*

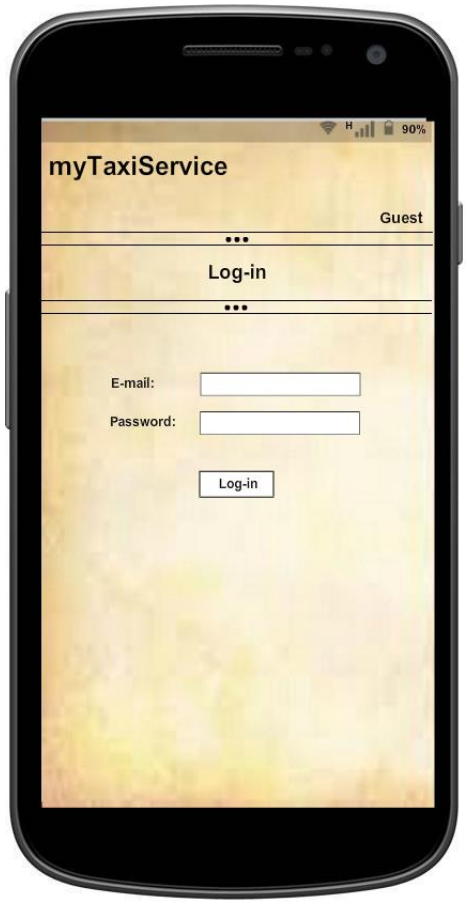
  and then click on the “Sign-up” button.

- **Guest user mobile app log-in page:** is the page of the mobile app provided to users in which guest logs in into the system to take user privileges.

  The user logs in submitting:

  - *E-mail*
  - *Password*

  and then click on the “Log-in” button.

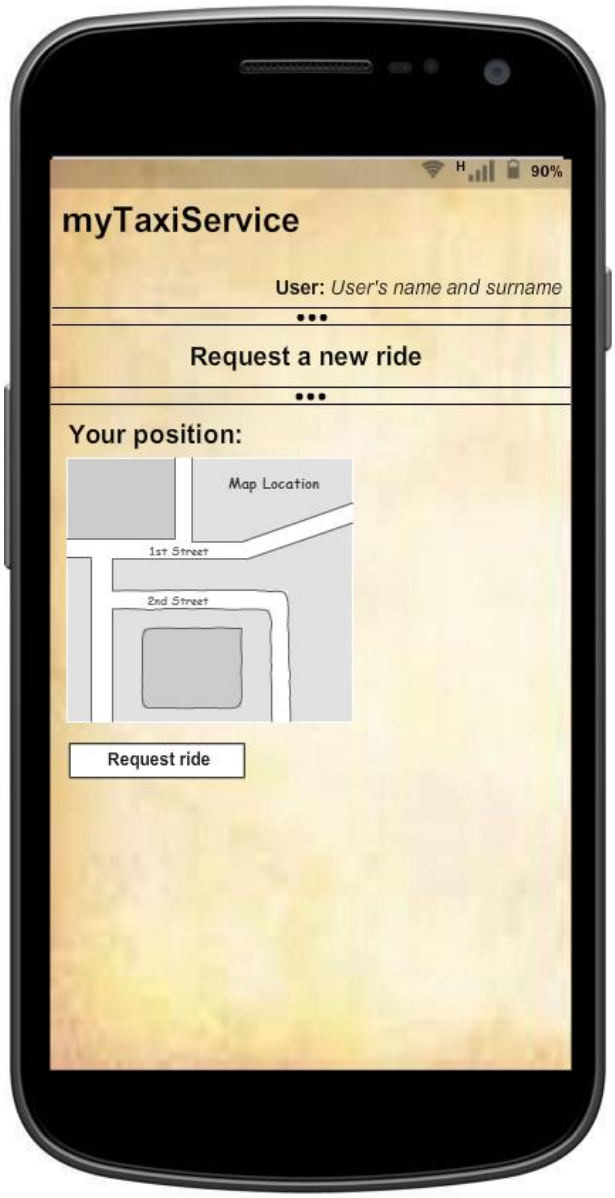


- **User mobile app request a ride page:** is the page of the mobile app provided to users in which a user can request a taxi ride. The user must specify

  - *The street where he's located*
  - *The nearest house number from the his position*

  and then click on the "Request ride" button.

- **User mobile app request a ride page**: Is the same of the previous interface, but in this case the user can see his position in a map.

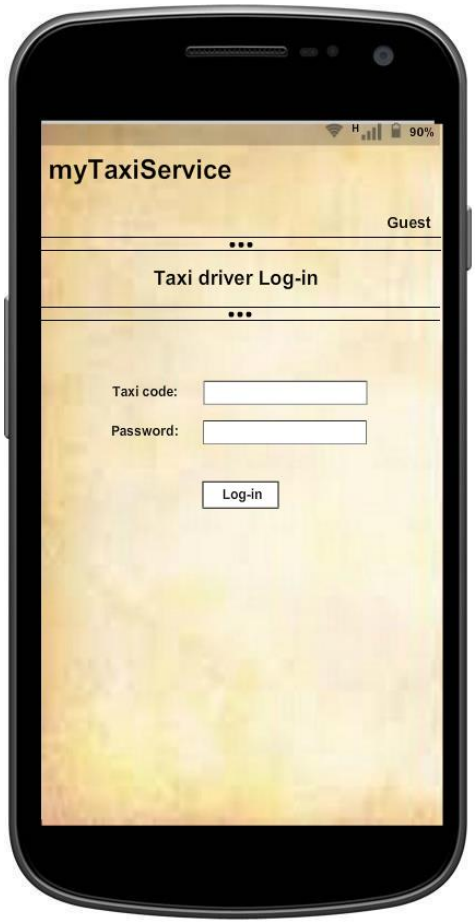


- **Taxi driver**
  - **Guest taxi driver mobile app home page:** is the page of the mobile app provided to taxi drivers, in which a guest can only log-in to take taxi drivers privileges.

    The taxi driver logs in submitting:

    - *Taxi Code*
    - *Password*

    and then click on the “Log-in” button.

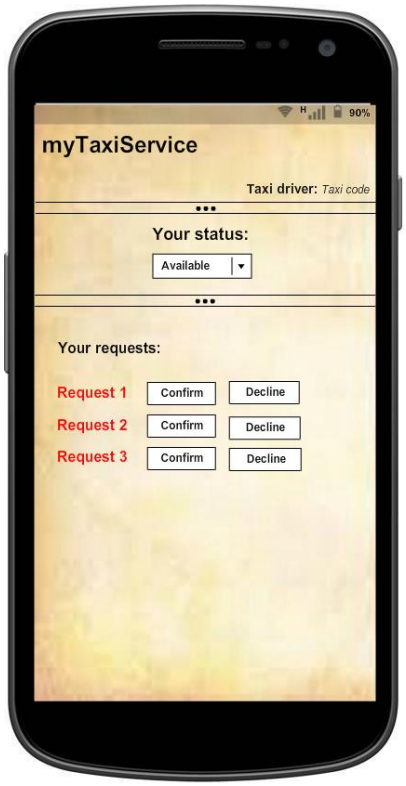


- **Taxi driver mobile app work page:** is the page of the mobile app provided to taxi drivers in which a taxi driver can set his status and manage requests made by users and forwarded by the server.

  The taxi driver can choose his status among “available” or “unavailable” from a list.

  For request management, the taxi driver sees the request as a list and can choose to confirm or decline the request.

The image only shows a sketch of the interface; strings “request 1”, “request 2” etc. are thought to be substituted with city street and nearest home number to w here the user who sent the request is located.



3.2.2 Performance requirements

- 90% of the amount of citizens must be able to sign-up into the system;
- System shall be able to handle at least 10000 request every second;
- Up to 4000 connected users and 2000 taxi driver at time must be able to be connected;

3.2.3 Software system attributes

3.2.3.1 Availability

- The system shall offer the service 24 hours per day, 7 days per week;
- Updates of the system must not stop supply of the service, at any time;
- In case of crash, the system must restart and become available again in 15 minutes;

3.2.3.2 Reliability

- The system shall have a maximum downtime of 1 hour per year;
- The maximum amount error on taxi arriving time calculation must be of 5 minutes;
- If a GPS connection lost event occur, the system must not cancel the ride;

3.2.3.3 Portability

- The mobile app must be developed considering that, in the future, it will be made compatible with other famous mobile operating systems such: Windows Phone and iOS;

3.2.3.4 Maintainability

- Source code must be written using as more standard design patterns as possible;
- Source code and entire system design must be well documented;

3.2.3.5 Security

- User passwords must be stored as encrypted;
- Communications among server and clients must be encrypted, in order to avoid security problems like - for example – credentials sniffing.

4 Scenarios identifying

Some possible scenarios for the system are described in this chapter.

- Al really doesn't like walking; he likes to move in taxi. So he's looking for a way to request a taxi at any time and from anywhere he is. He does a google search from his personal computer and finds out that myTaxiService could be the right application for his needs, so he decide to open the web application. He reads the brief description of myTaxiService from the guest home page and decide to give a try. He clicks in the "sign-up" link and compiles a form inserting his personal information: name, surname, e-mail and password. The system accepts his registration and Al visualizes the log-in page in the web application.

- Al decides to request a taxi using myTaxiService; he's at home so decides to make the request from myTaxiService web application. He goes into myTaxiService web application and clicks "Log-in". The web application prompts Al to fill a form with his credentials: e-mail and password. He inserts the data into the form and the application accepts the login and shows Al the request a ride page in the web application.
  Al selects the street where he lives and his house number from the lists in the page and then clicks on the "request a ride" button. The system receives Al's request and forwards the request to the first taxi driver of the queue of the zone in which Al's house is located. Al views a "request forwarded, please wait" message.

- Julian is a taxi driver registered to myTaxiService system, he's just starting his work, he runs his mobile app and sets his status to "available". The system receives the information and adds Julian to the queue of the zone where Julian is located; the system knows about

Julian location zone thanks to GPS information sent by the taxi driver mobile app.

- Julian, in his working time, stops for a coffee break, so changes his status to “unavailable”. The system removes Julian to the queue of the zone where Julian is located; the system knows about Julian location zone thanks to GPS information sent by the taxi driver mobile app. At the end of the break Julian comes back to work and sets his status again to “available”.

- Julian is parked at the side of a street in the same zone of Al’s house, he receives a request from myTaxiService in his mobile app. Julian his free from other requests, so confirms the request, reads the street and house number of the request and goes serve the request. Since Al’s request was confirmed, the system shows to Al Julian taxi code and estimated waiting time and it asks Al if he wants to confirm or cancel his taxi request; estimated time is good enough for Al, so he confirms his request.

  While Julian is serving the request he receives another request but, since he’s already serving a request, he decline the request clicking on the “Decline” button.

- Alice is already signed up into myTaxiService, she just finished her shopping and wants to go home, so runs the myTaxiService mobile app and logs-in; the system shows her the request a ride page in the mobile app and she selects the street in which she is and the nearest house number to her position from the lists visualized and then clicks on the “request a ride” button. The system receives Alice’s request and forwards the request to the first taxi driver of the queue of the zone in

which Alice is located. Alice views a “request forwarded, please wait” message.

5 UML models

All UML modelling part is described here. Object like “buttons” and “page” are hypothesized only to clearly express the situation.

5.1 Use cases

Here most important and general use cases derived from the previous scenarios are described.

Name	**Web application sign up**
Actors	Guest
Entry Conditions	Guest hasn’t the credentials to log-in to the application
Flow of events	• Guest opens the “Sign-up” link either in the home page; • The system shows to the guest the sign up page with the sign up form; • Guest fills the sign up form and press the button “Sign-up” • The system check the data, confirms sign up and shows the log in page to the guest.
Exit conditions	Submitted data is acquired from the system and stored in a database and the guest can now log-in.
Exceptions	In case that: • Guest inserts wrong type of data or doesn’t respect any specific limitation on input format. • Whether the guest type a wrong password or he doesn't fill the mandatory fields an error will be shown and the system will continue showing the sign-up page.

Name	**Web application login for user**
Actors	Guest, User
Entry Conditions	Guest has the credentials, that is the sign up phase has already been done.
Flow of events	• Guest opens the “Log-in” link in the web application home page; • System shows the user login form to the guest; • Guest fills all the login form with his email and password and clicks the “Log-in” button; • System checks login form data, accepts the login and redirect the user to the request ride page;
Exit conditions	User is logged into the application and the request ride page is shown.
Exceptions	Inserted credentials are wrong and an error is shown.

Name	**Mobile app login for taxi driver**
Actors	Guest, taxi driver
Entry Conditions	The guest has the credentials received by mail and email and runs his mobile app.
Flow of events	• System shows the taxi driver login form, which requires the submission of taxi code • Guest fills all the login form with his taxi code and password and clicks the “Log-in” button; • System checks login form data, accepts the login and redirect the taxi driver to the request taxi drivers work page;
Exit conditions	Taxi driver logs into the application and taxi driver work page is shown
Exceptions	Inserted credentials are wrong and an error will be shown.

Name	**Make a taxi request**
Actors	User
Entry Conditions	User has logged in
Flow of events	• System shows the user the request ride page with a form composed by: - Street where the user is located - Nearest house number to the user - A “request ride” button • User fills the form with requested data and clicks on the request ride button; • System shows a “request forwarded, please wait” message;
Exit conditions	• A taxi driver has taken the user request in charge and the system informs the user with taxi code and estimated waiting time and asks the user if he confirms or cancel his taxi ride request; • No taxi is able to take in charge the user request and the system informs the user about this;
Exceptions	No exceptions

Name	**Taxi driver receives a request**
Actors	Taxi driver
Entry Conditions	The taxi driver status is “available”.
Flow of events	• Taxi driver receives the notification of the user request in his mobile app work page;
Exit conditions	• Taxi driver wants to confirm or decline request and the “Taxi driver accepts or rejects of a request” begins
Exceptions	No exceptions

Name	**Taxi driver changes his status**
Actors	Taxi driver
Entry Conditions	The taxi driver is already logged in the mobile app, views the work page and decides to communicate his status to the system
Flow of events	• From the work page, taxi driver select his status from the list among “available” and “unavailable”; • The system stores the information and based on the status, adds or removes taxi driver from the zone queue (system knows which the zone queue is thanks to GPS information sent by taxi driver mobile app);
Exit conditions	• If taxi driver changes his status to “available”, taxi driver is added to the zone queue. • If taxi driver changes his status to “unavailable”, taxi driver is added to the zone queue.
Exceptions	No exceptions

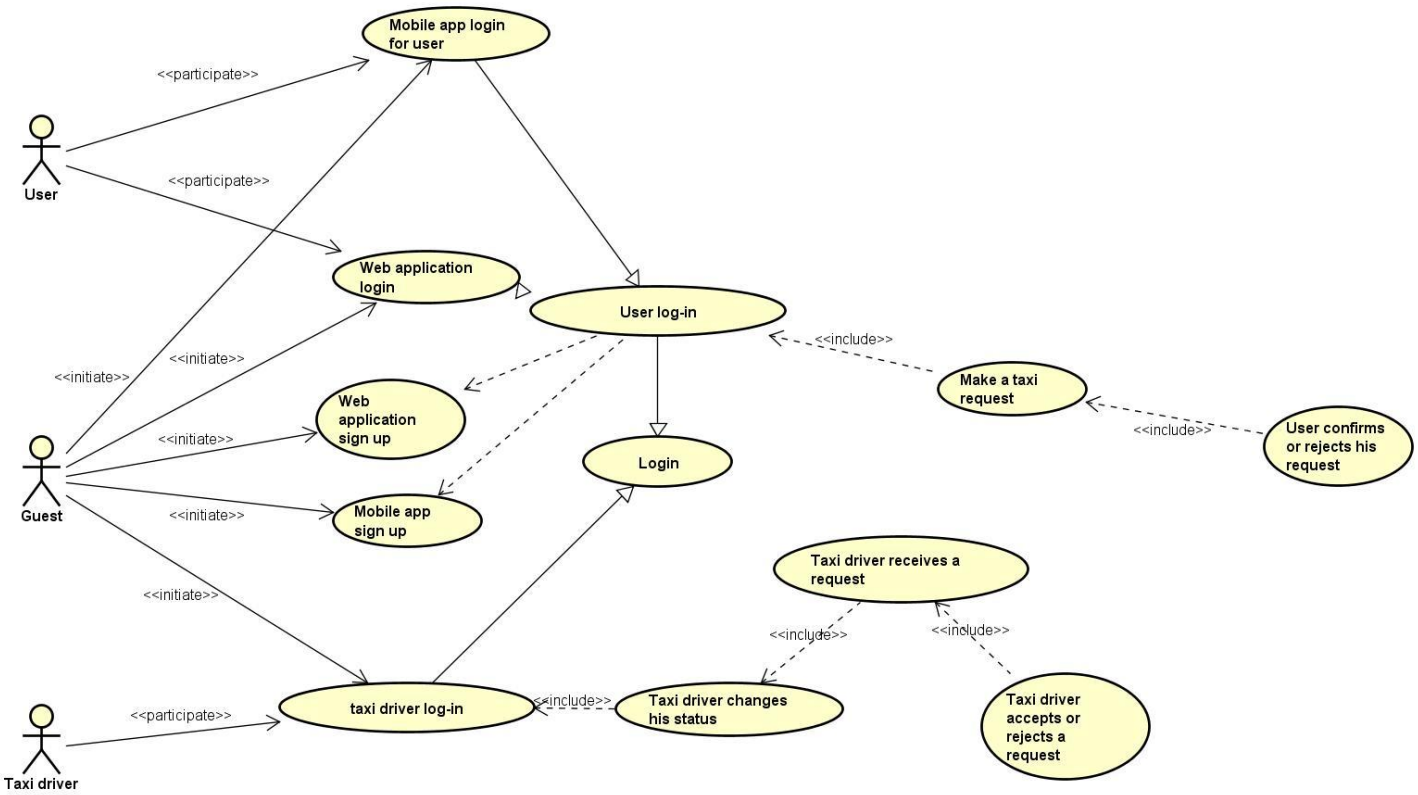
Name	**User confirms or rejects his request**
Actors	User, taxi driver
Entry Conditions	User has made a taxi ride request and the request was confirmed by a taxi driver, so the user was prompted with taxi code and estimating waiting time.
Flow of events	• From the request confirmed page, where the user can see taxi code and estimated waiting time, the user selects “confirm” or “cancel” in order to confirm or cancel the request; • The system forwards the information to the taxi driver;
Exit conditions	• The request is deleted from taxi driver’s mobile app work page . • The system shows user the request ride page, either in web application or mobile app.
Exceptions	No exceptions

Name	**Taxi driver accepts or rejects of a request**
Actors	Taxi driver
Entry Conditions	Taxi driver has received a request so he views the notification of the user request in his mobile app work page;
Flow of events	• Taxi driver accepts the user request through the “Confirm” button or declines the user request through the “Decline” button;
Exit conditions	• If taxi driver accepts the request the user is informed with taxi code and estimated waiting time; • if taxi driver declines the request, the system forwards the user request to the next taxi in the zone queue;
Exceptions	If taxi driver doesn’t confirm or decline the request within 30 second the request is automatically declined.

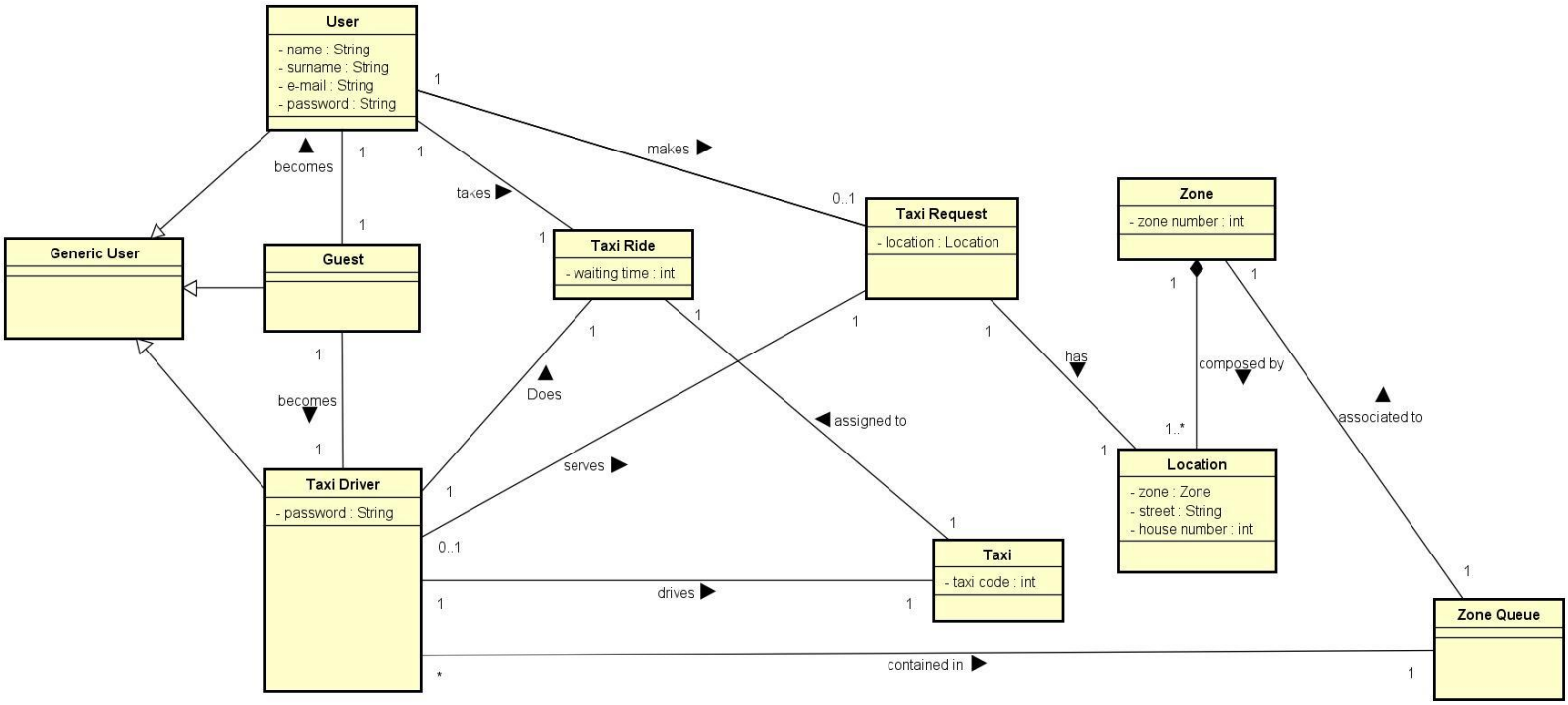
Name	**Mobile app login for user**
Actors	Guest, User
Entry Conditions	Guest has the credentials, that is the sign up phase has already been done.
Flow of events	• Guest opens the “Log-in” link in the web application home page; • System shows the user login form to the guest; • Guest fills all the login form with his email and password and clicks the “Log-in” button; • System checks login form data, accepts the login and redirect the user to the request ride page;
Exit conditions	User is logged into the mobile app and the request ride page is shown.
Exceptions	Inserted credentials are wrong and an error is shown.

Name	**Mobile app sign up**
Actors	Guest
Entry Conditions	Guest hasn't the credentials to log-in to the application
Flow of events	• Guest opens the "Sign-up" link either in the home page; • The system shows to the guest the sign up page with the sign up form; • Guest fills the sign up form and press the button "Sign-up" • The system check the data, confirms sign up and shows the log in page to the guest.
Exit conditions	Submitted data is acquired from the system and stored in a database and the guest can now log-in.
Exceptions	In case that: • Guest inserts wrong type of data or doesn't respect any specific limitation on input format. • Whether the guest type a wrong password or he doesn't fill the mandatory fields an error will be shown and the system will continue showing the sign-up page.

5.2 Use case diagram

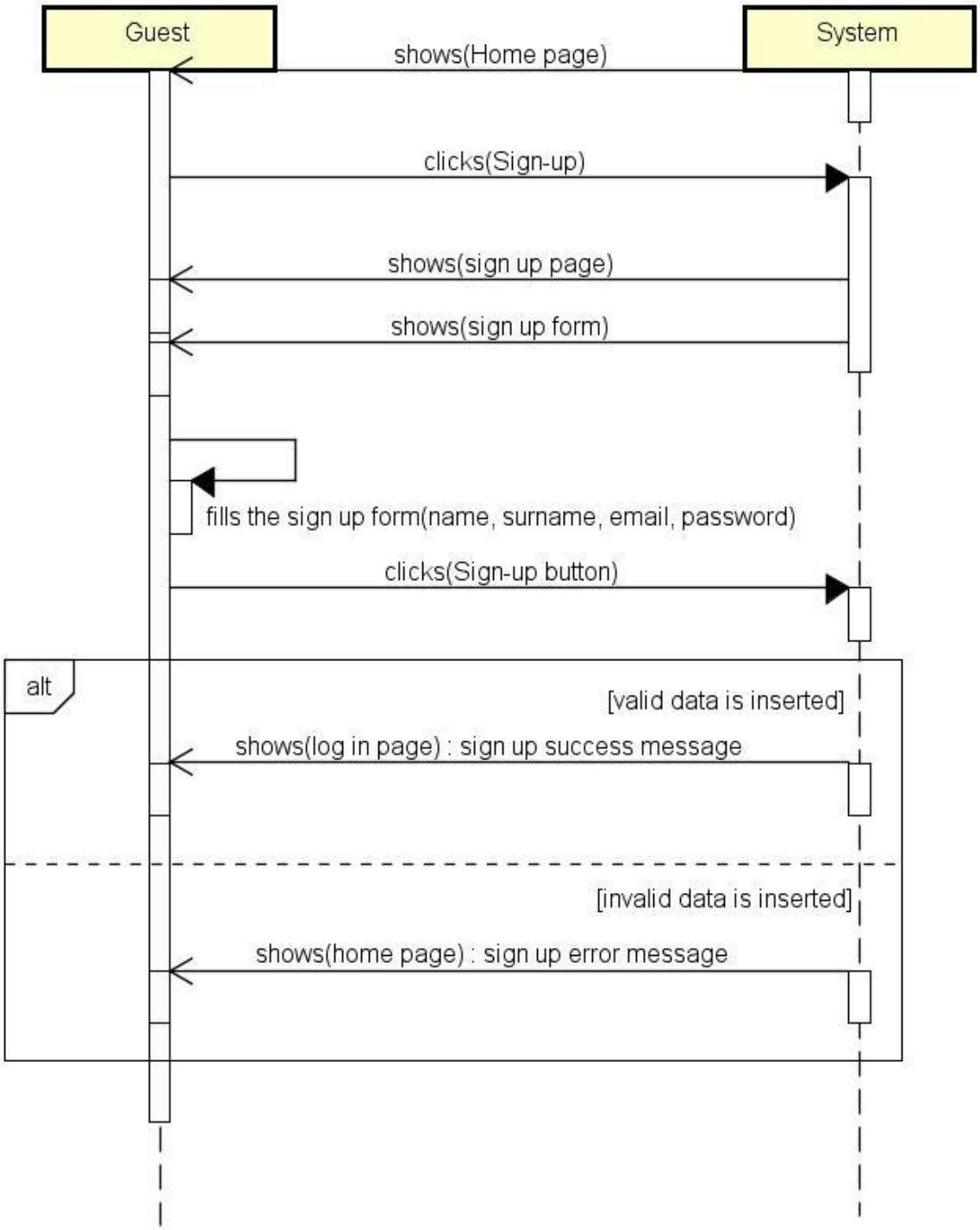


5.3 Class diagram

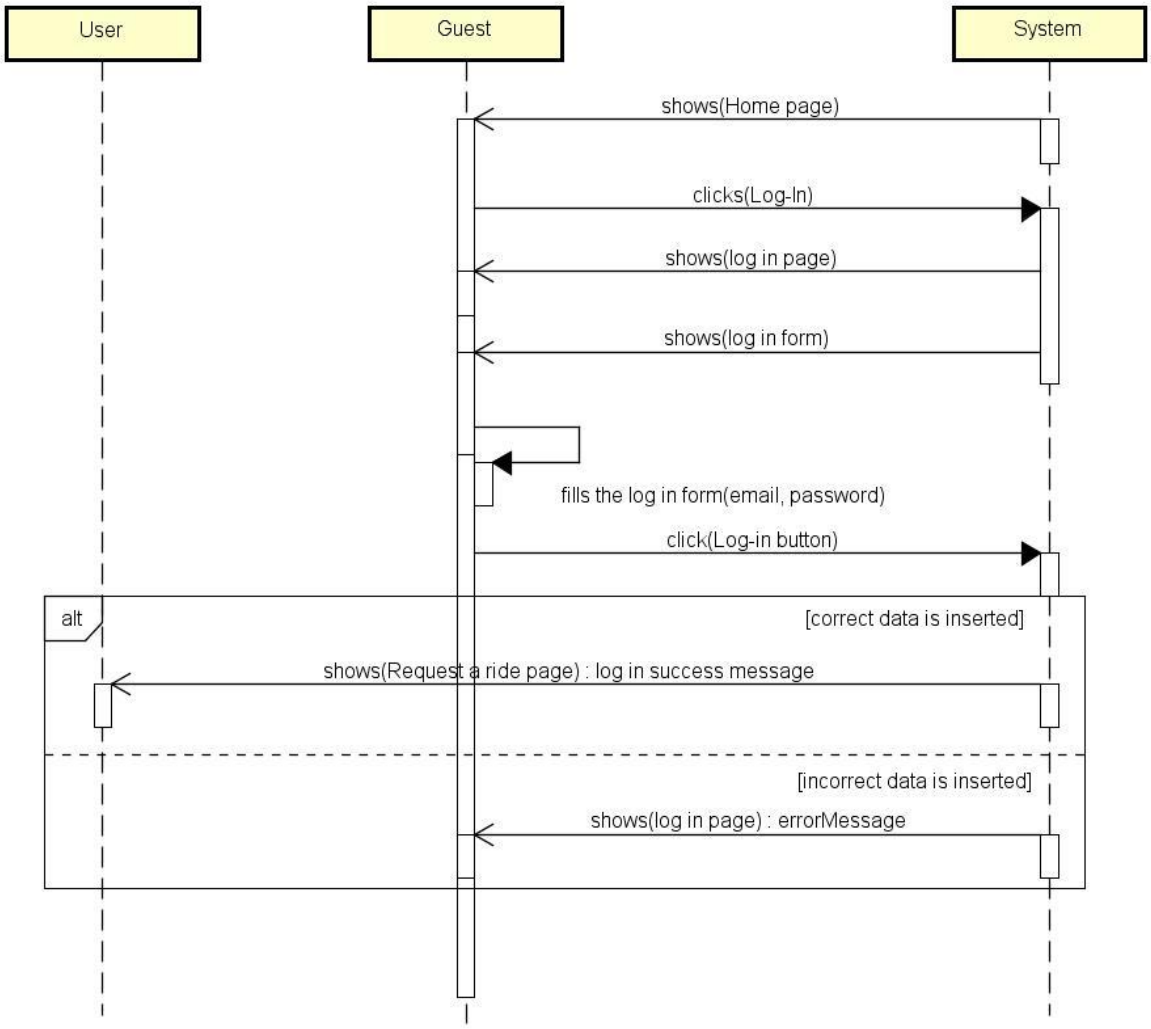


5.4 Sequence diagrams

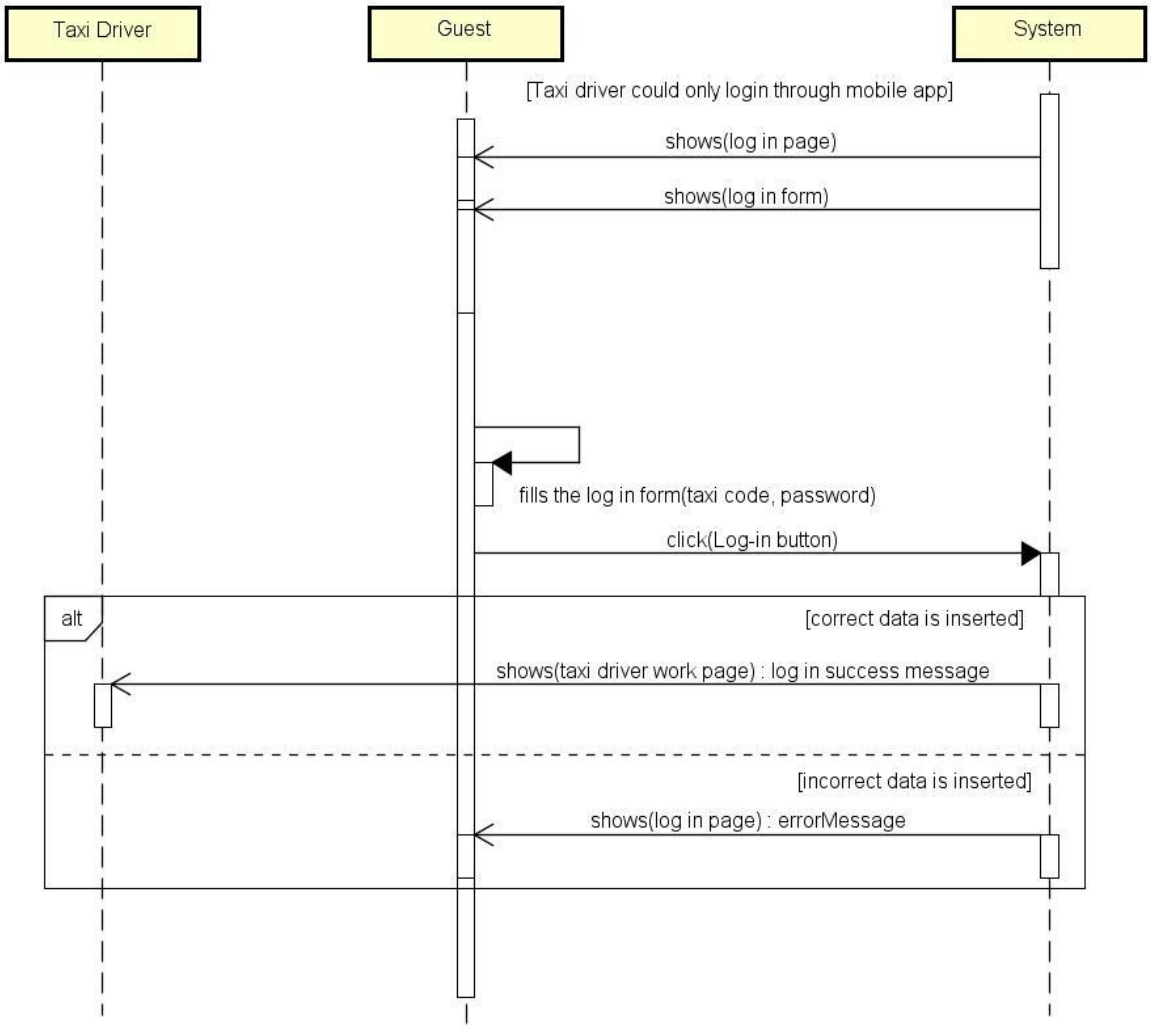
5.4.1 Sign up



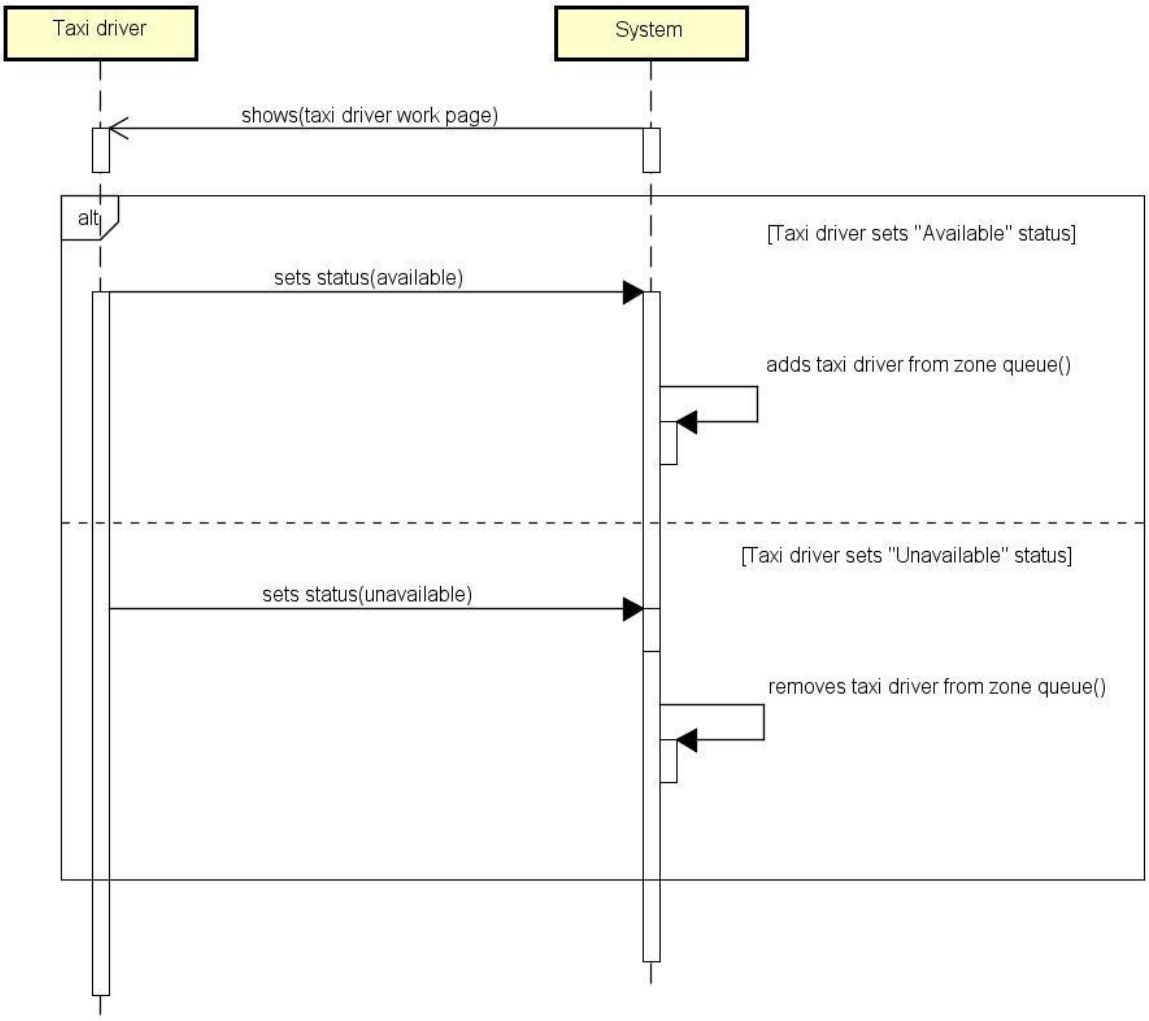
5.4.2 User log in



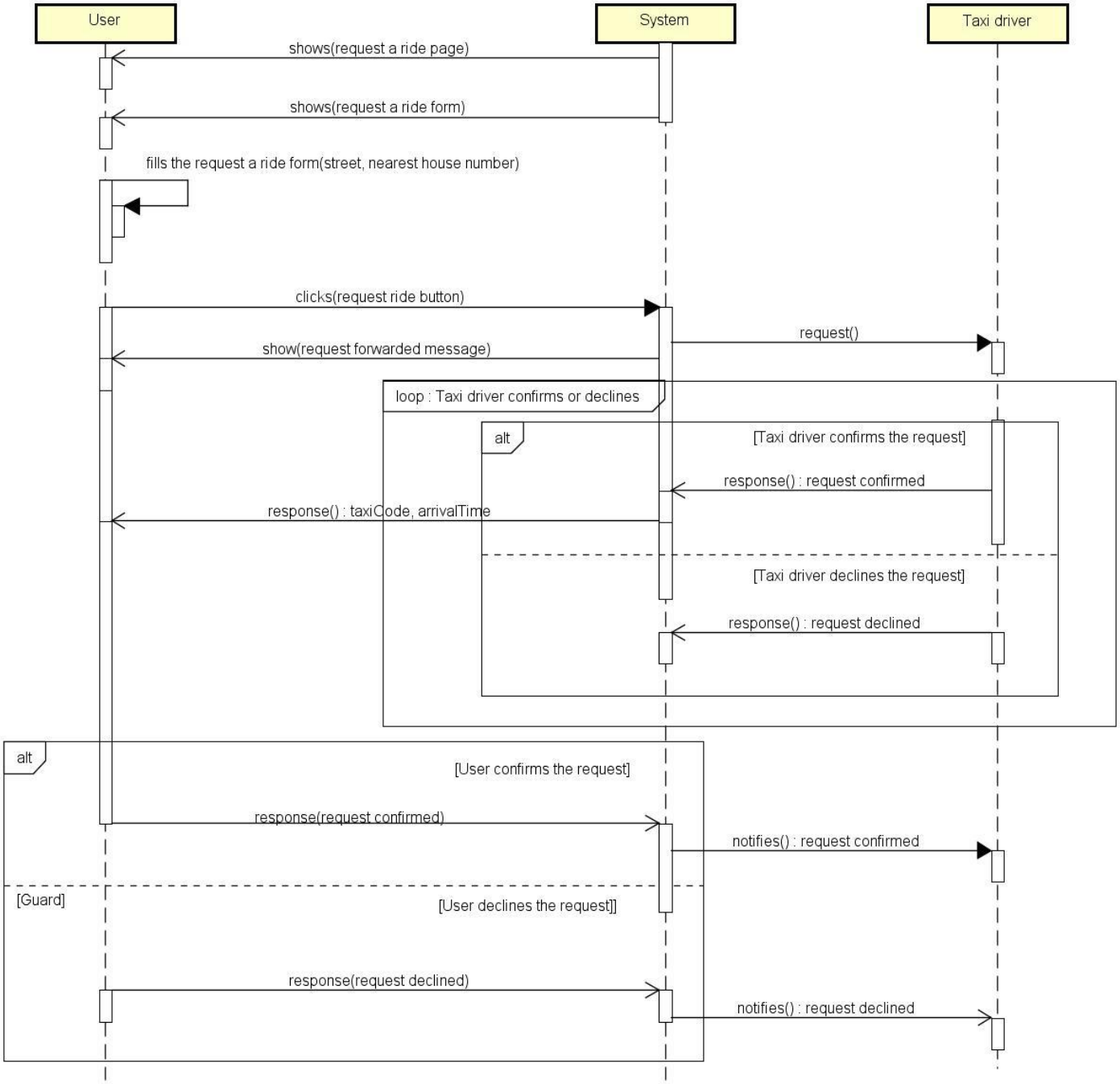
5.4.3 Taxi driver log in



5.4.4 Taxi driver status set

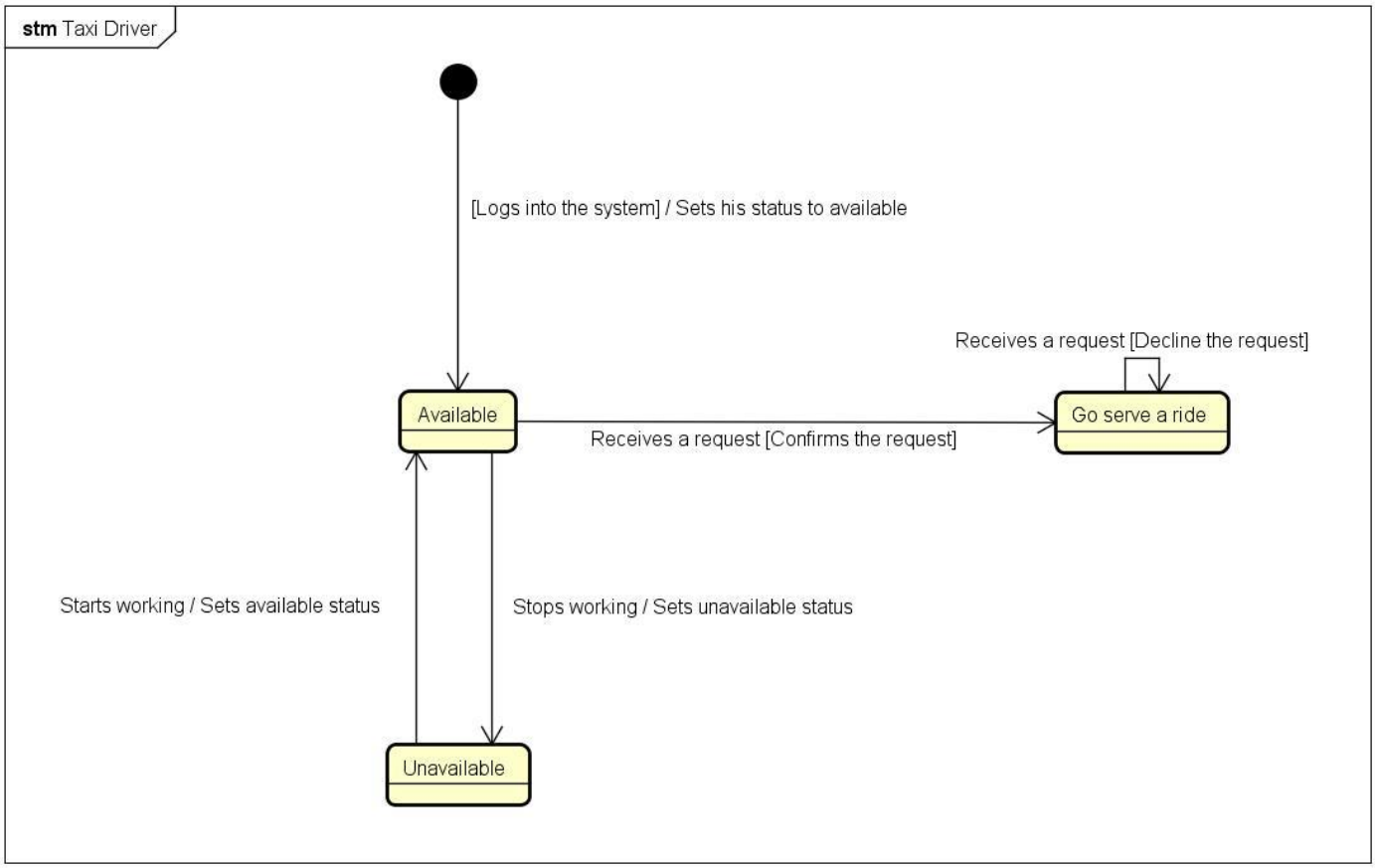


5.4.5 Request


User
System
Taxi driver
shows(request a ride page)
shows(request a ride form)
fills the request a ride form(street, nearest house number)
clicks(request ride button)
request()
show(request forwarded message)
loop : Taxi driver confirms or declines
alt
[Taxi driver confirms the request]
response() : request confirmed
response() : taxiCode, arrivalTime
[Taxi driver declines the request]
response() : request declined
alt
[User confirms the request]
response(request confirmed)
notifies() : request confirmed
[Guard]
[User declines the request]]
response(request declined)
notifies() : request declined

5.5 State chart diagrams

5.5.1 Taxi driver



6 Alloy modelling

I used Alloy to verify the modelling of the system. In this paragraph Alloy code and alloy worlds generated will be reported.

6.1 Alloy code

```
//=========== Signatures

abstract sig Bool{}

sig True extends Bool {}
sig False extends Bool {}

sig PositiveInt { integer: Int }

sig GenericString {}

abstract sig GenericUser{}


sig TaxiDriver extends GenericUser{

        taxi: one Taxi,
        location: one Location

}
```

```
sig User extends GenericUser{

        email: one GenericString,
        password: one GenericString
}


sig Taxi{

        taxiCode: PositiveInt

}

sig Location{

        street: one GenericString,
        houseNumber: one PositiveInt

}

sig Zone{

        locations: some Location,
        queue: one Queue

}
```

```
sig Queue{

        taxiDrivers: set TaxiDriver

}


sig Ride{

        source: one Location,
        destination: one Location,
        startTimestamp: one PositiveInt,
        endTimestamp: one PositiveInt,
        tookBy : one User,
        doneBy: one TaxiDriver,
}

sig Request{

        ride: lone Ride,
        requestedBy: one User,
        servedBy: lone TaxiDriver,
        confirmedByUser: one Bool,
        confirmedByTaxiDriver: one Bool

}
```

```
//=========== Facts

//PositiveInt means only positive integers

fact  {

all positiveInt: PositiveInt | positiveInt.integer >= 0

}

//Each user is associated with a unique email

fact uniqueEmailForEachUser{

        all disj u1,u2: User | u1.email != u2.email

}


// Each taxi driver is associated with a unique taxi

fact uniqueTaxiForEachTaxiDriver{

        all disj t1,t2: TaxiDriver | t1.taxi != t2.taxi

}
```

```
// Each taxi is associated with a unique taxi code

fact uniqueTaxiCodeForEachTaxi{

        all disj t1,t2: Taxi | t1.taxiCode != t2.taxiCode

}

//No different queues with same taxi drivers

fact NoDifferentQueuesWithSameTaxiDriver {

        all disj q1,q2: Queue | no t1: TaxiDriver | (t1 in q1.taxiDrivers) and (t1
in q2.taxiDrivers)

}

//Different zones doesn't have same queue

fact noSameQueueForDifferentZones {

        all disj z1,z2: Zone | !(z1.queue = z2.queue)

}
```

```
//Same location can't be in different zones

fact noSameLocationInDifferentZones {

        all disj z1,z2: Zone | no l1: Location | (l1 in z1.locations) and (l1 in z2.locations)

}

//Location of each taxi drivers in a zone queue is inside locations group of the zone

fact taxiDriverLocationInsideZoneLocations {

        all z1: Zone, t1:TaxiDriver | t1 in z1.queue.taxiDrivers implies t1.location in z1.locations

}

//User takes only one ride at time

fact userTakesOnlyOneRideAtTime {

        all u1: User | no disj r1,r2: Ride | (r1.tookBy = u1) and (r2.tookBy = u1) and ( (r2.startTimestamp.integer >= r1.startTimestamp.integer) and (r2.startTimestamp.integer <= r1.endTimestamp.integer) )

}
```

```
//Taxi driver does only one ride at time

fact taxiDriverDoesOnlyOneRideAtTime {

        all t1: TaxiDriver | no disj r1,r2: Ride | (r1.doneBy = t1) and
(r2.doneBy = t1) and ( (r2.startTimestamp.integer >= r1.startTimestamp.integer)
and (r2.startTimestamp.integer <= r1.endTimestamp.integer) )

}

//User who requests the ride is the one who takes the correspondant ride

fact userWhoRequestsTheRideTakesTheRide {

        all rq1: Request | rq1.requestedBy = rq1.ride.tookBy

}

//Taxi driver who serves the request is the one who does the correspondant ride


fact taxiDriverWhoServesTheRequestDoesCorrespondantRide {

        all rq1: Request | rq1.servedBy = rq1.ride.doneBy

}
```

```
// ========== Assertions


//Same taxi driver must not be in different queues

assert      noSameTaxiDriverInDifferentQueues {

            all disj q1,q2: Queue |  no t1: TaxiDriver | (t1 in q1.taxiDrivers) and
(t1 in q2.taxiDrivers)

}

check noSameTaxiDriverInDifferentQueues

//There must not be different taxis with same taxi code

assert noDifferentTaxiDriversWithSameTaxiCode {

            no disj t1,t2: TaxiDriver | t1.taxi.taxiCode = t2.taxi.taxiCode

}

check noDifferentTaxiDriversWithSameTaxiCode
```

```
//There must not be a ride took from a user different from the one who made the correspondant request


assert NoRidesTookFromAUserDifferentFromCorrespondantRequestUser {

        all rq1: Request, r1: Ride, u1: User |  ( ( (r1 in rq1.ride) and (u1 in r1.tookBy) )  implies (u1 in rq1.requestedBy) )

}

check NoRidesTookFromAUserDifferentFromCorrespondantRequestUser

//=========== Predicates


pred show {

}

run show for 5

//Shows a world with one user, one taxi driver and one Request made; the request is confirmed both by the user and taxi driver

pred OneRequestMadeAndConfirmed{

        #User = 1
        #Request = 1
        #TaxiDriver = 1
```

```
        some rq1: Request | (rq1.confirmedByUser = True) and
    (rq1.confirmedByTaxiDriver = True)

    }
    run OneRequestMadeAndConfirmed


    //Shows a world with one user, one taxi driver and one Request made; the request
    is confirmed by the taxi driver but not by the user

    pred OneRequestMadeAndNotConfirmedByUser{

        #User = 1
        #Request = 1
        #TaxiDriver = 1

        some rq1: Request | (rq1.confirmedByTaxiDriver = True) and
    (rq1.confirmedByUser = False)

    }
run OneRequestMadeAndNotConfirmedByUser
```

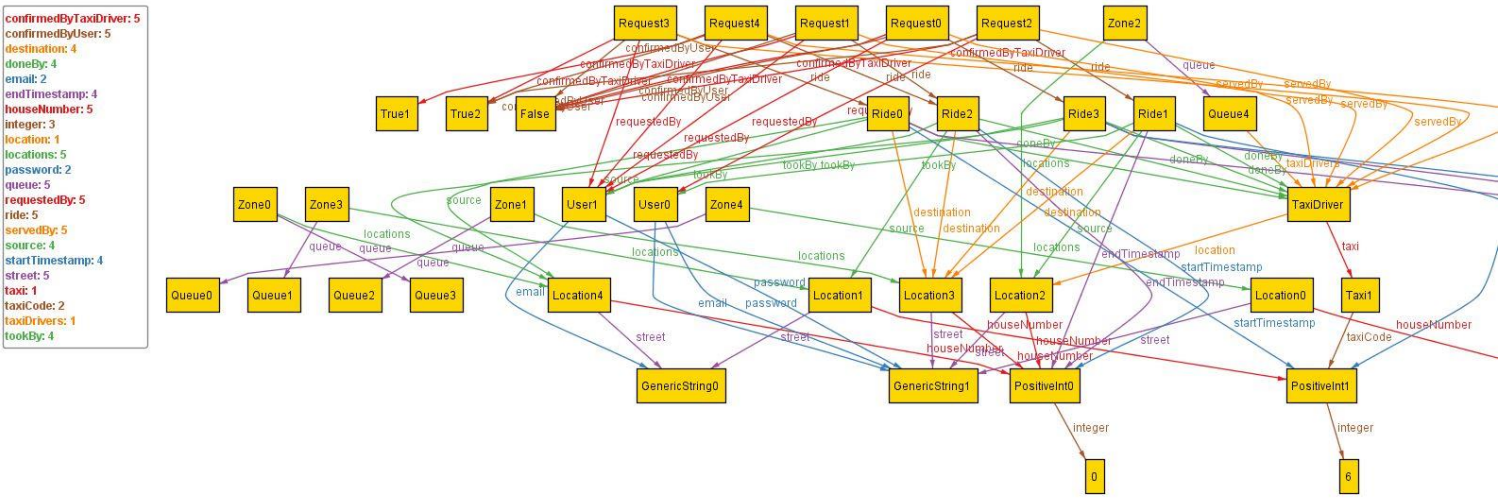
The result of the analysis is shown here:

6 commands were executed. The results are:
#1: No counterexample found. noSameTaxiDriverInDifferentQueues may be valid.
#2: No counterexample found. noDifferentTaxiDriversWithSameTaxiCode may be valid.
#3: No counterexample found. NoRidesTookFromAUserDifferentFromCorrespondantRequestUser may be valid.
#4: **Instance found.** show is consistent.
#5: **Instance found.** OneRequestMadeAndConfirmed is consistent.
#6: **Instance found.** OneRequestMadeAndNotConfirmedByUser is consistent.

6.2 Alloy worlds

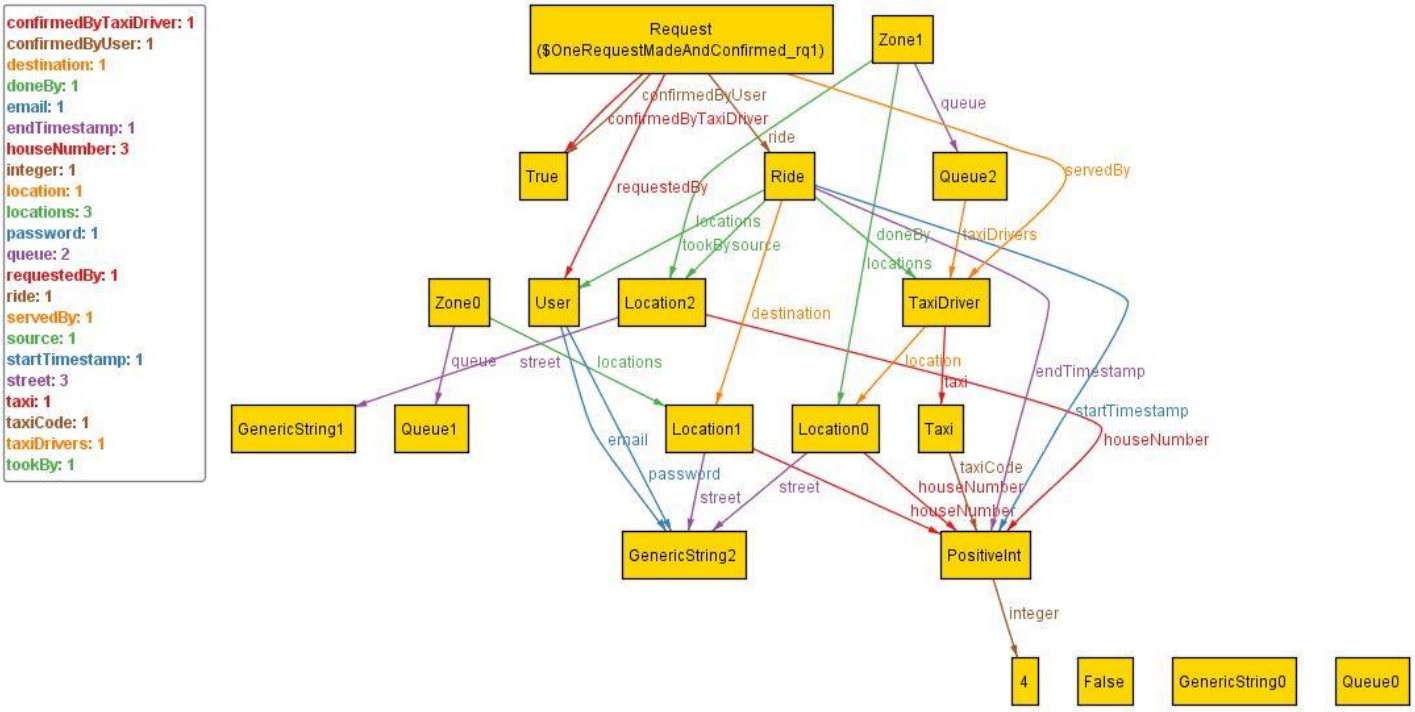
6.2.1 General World

The following world is a general world generated with the analyzer. It has been generated using the predicate show.



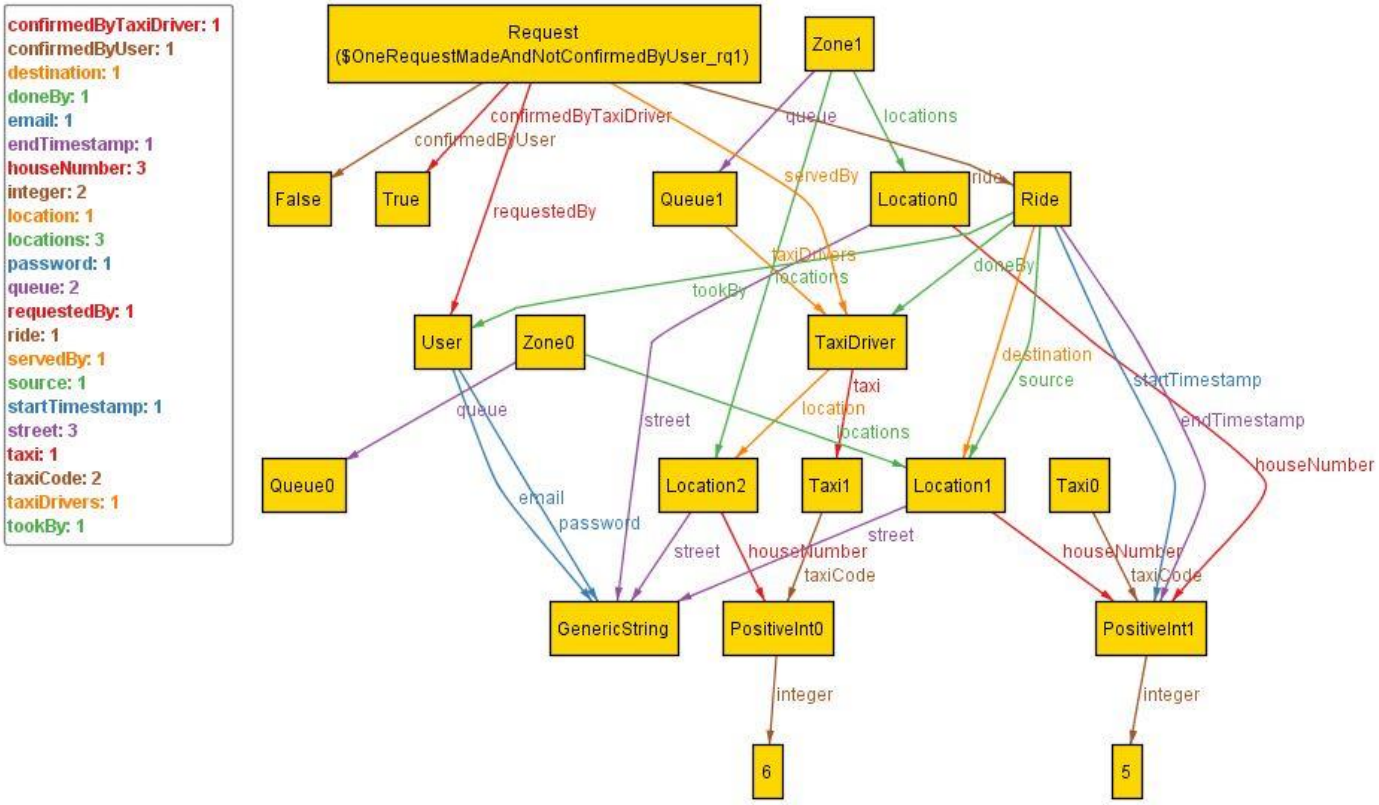
6.2.1 One request made and confirmed

The following world is a world with one user, one taxi driver and one Request made; the request is confirmed both by the user and taxi driver.



6.2.2 One request made and not confirmed by user

The following world is a world with one user, one taxi driver and one Request made; the request is confirmed by the taxi driver but not by the user.



7 Used tools

The tools I used for the creation of this RASD document are:

- **Microsoft Office Word 2010:** to redact the document
- **Astah:** For creating graphs
- **Alloy Analyzer 4.2:** For model consistence verification.
- **Mockflow (https://www.mockflow.com):** For user interface mockups.

8 Appendix

For the creation of this document I spent 90 hours.