

TAXISERVICE

REQUIREMENTS ANALYSIS AND SPECIFICATION DOCUMENT

Authors: Jacopo Silvestri 773082 Simone Penati 850448

Reference Professor: Mirandola Raffaella



**POLITECNICO
DI MILANO**

0. Index

1. INTRODUCTION	4
1.1 Description of the Problem	4
1.2 Goals.....	4
1.3 Domain Proprieties	5
1.4 Glossary	5
1.5 Assumptons	6
1.6 Proposed System.....	6
1.7 Identify Stakeholders.....	6
2. ACTORS IDENTIFICATION.....	6
3. REQUIREMENTS	7
3.1 Functional Requirements.....	7
3.2 Non-Functional Requirements.....	9
3.2.1 Log In and Sign Up Interface	9
3.2.1 User (Customer) Interface	10
3.2.1 User (Taxi Driver) Interface	10
3.2.1User (Customer) Request Taxi Interface Interface	11
3.2.5 User (Customer) Reservation Taxi Interface	11
3.3 Constraints	12
3.3.1 Regulatory Policies	12
3.3.2 Hardware Limitations	12
3.3.3 Interfaces to Other Applications	12
3.3.4 Performance Requirements.....	12
3.3.5 Design Constraints.....	12
3.3.6.1 Availability.....	12
3.3.6.2 Maintainability.....	12
3.3.6.3 Portability.....	12
3.4. Security.....	12
3.4.1 External Interface Side.....	12
3.4.2 Server Side.....	12
3.4.3 Application Side.....	12
3.5 Future Possible Implementation.....	12
3.6 Documentation.....	13
4. SCENARIOS.....	14
5. UML.....	15
5.1 Use Case Diagram.....	15
5.2 Use Case Description.....	16
5.3 Flowgraph Diagrams.....	24
5.4 UML Models.....	26
5.4.1 Class Diagram.....	26
5.4.2 Sequence Diagram - Sign Up.....	27
5.4.3 Sequence Diagram - Login.....	28
5.4.4 Sequence Diagram - Taxi Request.....	29
5.4.5 Sequence Diagram - Taxi Reservation.....	30
5.5 Finite State Diagram.....	31

6.0 Alloy Model.....	32
6.0.1 Signatures.....	32
6.0.2 Abstract Entities.....	32
6.0.3 Implementations and Local Signatures.....	33
6.0.4 Facts.....	35
6.0.5 Assertions.....	38
6.0.6 Predicates.....	39
6.1 Clarification.....	39
7. Used tools.....	40

1. Introduction

1.1 Description of the given problem

We will project and design TaxiService, which is an online service that will guarantee a fair management of taxi queues, while simplifying the access of passengers to the service. Other than this, it will enable an efficient system for Taxi reservation for all the app's clients.

The system should be able to register new users with their information, such as name, surname, phone number and e-mail.

Once a customer is registered to the application, he should be able to use it to call a taxi at any given location inside the city area. The system should answer the request with information about the license plate of the incoming taxi and the expected waiting time.

Taxi drivers use the mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call. The system offers a fair taxi management by dividing the city in areas and assigning taxis to their corresponding area using their GPS location, organizing them in a dynamic queue. In particular, the system should forward a ride request to the first taxi queuing in the calling customer's zone, whose driver can then decide whether to accept it or decline it. A customer can also reserve a taxi by specifying the origin and the destination of the ride. The reservation has to occur at least two hours before the desired time. In this case, the system confirms the reservation to the user and allocates a taxi to the request ten minutes before the meeting time with the user.

The system should also offer programmatic interfaces, in order to allow the development of additional features.

1.2 Goals

Here are the main feature that TaxiService will grants its users.

Users (Customers) should be able to:

- Sign up to the system as Customers;
- Log into the system;
- Call a taxi at a certain location, within the city limits.
- Be notified if his/her request have been accepted;
- Reserve a taxi ride for later in the day;

Users (Taxi Drivers) should be able to:

- Sign up to the system as Taxi Drivers;
- Log into the system;
- Accept and Refuse calls.
- Change their status (Available, Unavailable, Busy);

1.3 Domain Properties

We suppose the following statements to always hold true in the considered domain:

- When a request for a taxi is made, the origin position of the ride is always specified.
- The customer will always be present to actually benefit from the requested service.
- When a taxi receives a request, the acceptance or denial of the request from the driver will happen in a short time (within thirty seconds).
- The origin and destination of a ride will always be inside the city limits.

1.4 Glossary

For clarity's sake, we will now define some words that will be often used in our documentation of the project, so that their meaning will be unequivocal.

- **USER (Customer):** for user (customer) we mean a person already registered in the system as a person who wants to be a TaxiService's customer. We will indicate this also as "Customer".

The profile includes all these informations:

- Name;
- Surname;
- Email;
- Username;
- Password.
- Telephone Number;

And optionally:

- Picture;
- Address.

- **USER (Taxi Driver):** for user (taxi driver) we mean a person already registered in the system as a taxi worker which will be able to accept/refuse taxi calls and change their availability status. We will indicate this also as "Taxi Driver".

The profile includes all these informations:

- Name;
- Surname;
- Email;
- Username;
- Password;
- Taxi License Code;
- License Plate.

- **GUEST:** with Guest we indicate a user who has not yet signed up, either as a Customer or a Taxi Driver. The only action performable as Guest is the Sign Up (to register into the system).

Signing up and thus registering into the system is an action only performable by a Guest; it cannot be performed by either a Customer or a Taxi Driver.

REQUEST: a Customer requests a taxi for immediate use.

RESERVATION: a Customer reserves a Taxi to benefit from the service later in the day.

1.5 Assumptions

Here we clarify some points from both the specification document and this document, which could otherwise be ambiguous, thus making some assumptions.

- Reservations are accepted by the system only if made for the same day in which are requested.
- When making a reservation, a Customer must always specify the desired time.
- Reservations can be cancelled up to fifteen minutes before the scheduled time.
- Once a reservation has been made, it is no longer possible to modify the origin, destination or time.
- The Taxi Driver chosen by the system cannot decline a reservation call.
- There are no connection problems nor server down events. The GPS connection is always working.
- The system will always know all the Taxi Drivers' positions by their GPS coordinates.
- A single User (Customer) cannot make more than one request at the same time.
- A single User (Taxi Driver) cannot accept one request if has already carrying on another one.

1.6 Proposed System

We will project an online application using the JEE platform. A central server will manage all the request made by both the web software and the mobile application, while also run the business logic and generate the dynamic calls used for the taxi reservations. Additionally the server will also update the status of all the Taxi Drivers and manage taxi queues in every city area independently from one another. Customers' and Taxi Drivers' information will be registered in an dedicated online database, which will hold all the account data and an history of their past interactions.

1.7 Identifying Stakeholders

The main stakeholder is prof. Raffaella Mirandola, who gave us a general description of the problem and a time schedule that we must respect. We are to work on the development of an online metropolitan-service application, which will involve writing the requirements analysis, create its design and run the testing and the project reporting. Our objective is to focus on each of these phases to produce a quality project which should be able to be market-quality by the end of the semester. A possible stakeholder for our application could be a public traffic agency or a private taxi s.r.l. that wants to optimize their taxi services to fight back the emerging success of car sharing.

2. Actors Identification

There are three actors in our system:

- Guest: a guest is someone who hasn't signed up yet, and is only able to sign up as a User (Customer) or as a User (Taxi Driver).
- User (Customer): a User (Customer), or simply Customer, is a user who has signed up and is logged in as such. The Customer may request a taxi, reserve one for a certain time of the day or cancel his previous reservations.
- User (Taxi Driver): a User (Taxi Driver), or simply Taxi Driver, is a user who has signed up and is logged in as such. The Taxi Driver has the capability to accept/refuse calls and change its online status.

3. Requirements

After the analysis of our Goals and Domain Functions, these are the requirements that the TaxiService applications needs to fulfill.

- Registration functionality
 - The system has to provide the capability for a Guest to register his data into the dedicated Database as a User (Customer) or a User (Taxi Driver).
- Login functionalities.
 - A User (either Customer or TaxiDriver) has to perform the Login action. The Login gives access to different services based on the type of User.
- Taxi Request.
 - The system has to provide a function that allows Users (Customer) to request a taxi through the TaxiService application.
 - The system will then have to inform the passenger about the license plate of the incoming taxi and waiting time.
 - The system has to provide a function that allows a user to visualize his pending request and the ability to cancel it before the taxi arrives.
- Taxi Status Tracking.
 - The system has to provide Taxi Drivers a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call.
 - The System should also provide the traceability of every Taxi Vehicle via GPS.
- Area-based Taxi queue Management.
 - The system should optimize waiting time through the correct management of taxi queues by dividing the city in taxi zones (approximately 2 km squared each) and by associating to each zone a queue of taxis.
- Taxi Reservation;
 - The system has to provide the capability to reserve a Taxi, to be in a certain place at a certain hour. After a reservation has been made, the system has to forward that request ten minutes before the selected time.

3.1 Functional Requirements

Now that we have defined the main feature of TaxiService, we can find some functional requirement concerning each defined actor:

A Guest can

- Sign up;

A Customer can:

- Login;
- Modify his profile information;
- Request a Taxi.
- Check the license plate of the requested Taxi;
- Check the waiting time for the incoming Taxi;
- Reserve a Taxi;
- Cancel a Taxi Reservation;
- Cancel a Taxi Request;

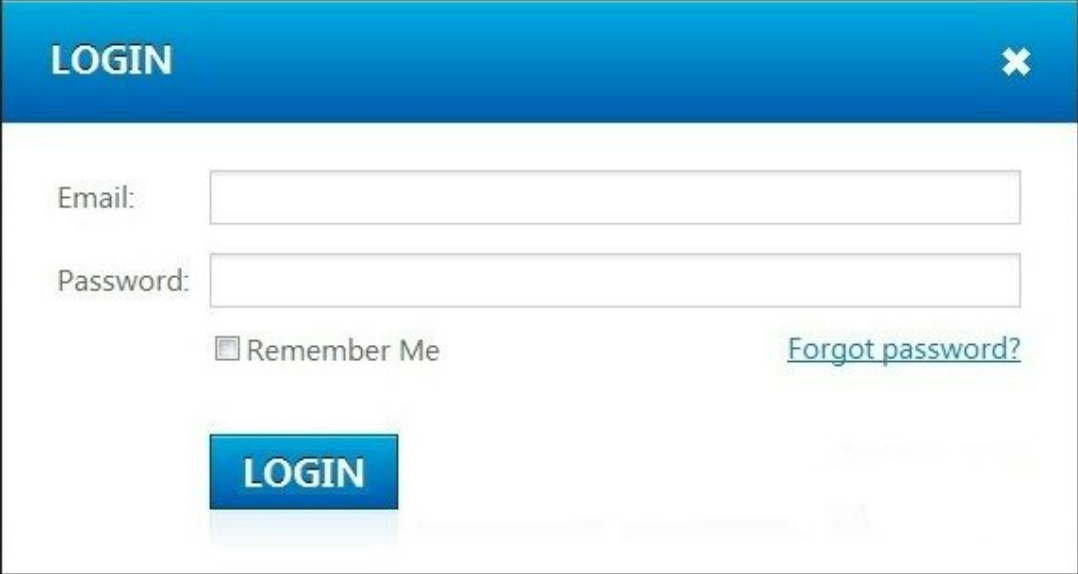
A Taxi Driver can:

- Login;
- Modify his profile information;
- Update its status;
- Accept Requests;
- Refuse Requests;


3.2 Non-Functional Requirements

3.2.1 Login and Sign Up Interfaces

This is the first page of both the web and the mobile application. Before a Guest who already has an account can do anything else, he have to login. If does not possess an account, the only option is to Sign Up (by inserting his name, surname, username, phone number, email and password), and then continue with the Login procedure.



A login interface mockup. It features a blue header bar with the word "LOGIN" in white and a close button (X) in the top right corner. Below the header, there are two input fields: "Email:" and "Password:". Below the "Password:" field, there is a checkbox labeled "Remember Me" and a link labeled "Forgot password?". At the bottom, there is a blue button with the word "LOGIN" in white.



A "Create a New Account" interface mockup. The title "Create a New Account" is centered at the top. Below it, a subtitle reads "Let's set up your account. Already have one? [Sign in here.](#)". The form consists of two columns of input fields. The left column has fields for "username", "email", and "password". The right column has fields for "name", "surname", and "phone number". At the bottom, there are two buttons: "As Customer" and "As Taxi Driver".

3.2.2 User (Customer) Interface

From the User (Customer) Interface a User (Customer) can view its profile information, change them, request a Taxi, reserve a Taxi or perform a Logout Operation.



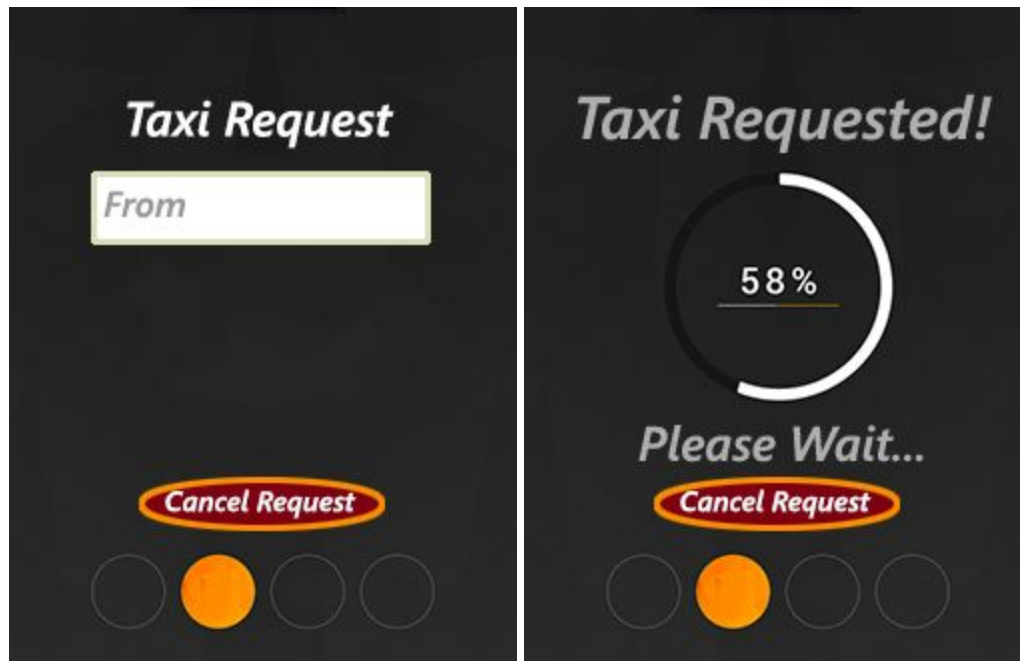
3.2.3 User (Taxi Driver) Interface

From the User Customer Interface a User (Customer) can view its profile information, change them, view its status, update its status, accept or decline a requests, accept reservations or perform a Logout operation. The Taxi #code is equal to its licence plate.



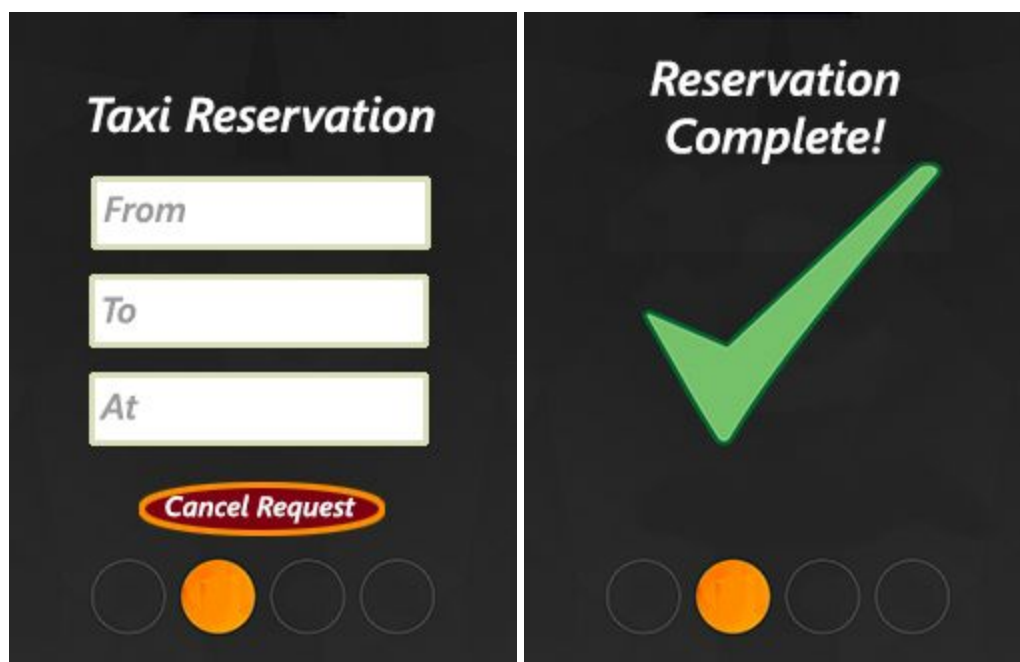
3.2.4 User (Customer) Request Taxi Interface

In this simple interface, reachable by pressing “Request a Taxi” on the User (Customer) Interface the User (Customer) has to select in which location he/she desires to take the Taxi. After the request process is completed, a waiting screen will be displayed. The User (Customer) will have the possibility to cancel its request before some User (Taxi Driver) picks it up.



3.2.5 User (Customer) Reservation Taxi Interface

In this simple interface, reachable by pressing “Reserve a Taxi” on the User (Customer) Interface the User (Customer) has to select in which location he/she desires to take the Taxi and its destination, as well as the time of the event.



3.3 Constraints

3.3.1 Regulatory Policies

TaxiService doesn't have to meet any regulatory policies.

3.3.2 Hardware Limitations

TaxiService doesn't have to meet any hardware limitations.

3.3.3 Interfaces to Other Applications

TaxiService doesn't have to meet any interfaces to other applications.

3.3.4 Performance Requirements

Performance have to ensure the usability of the application, while guaranteeing waiting times close to zero when a user is provided with a good internet connection.

3.3.5 Design Constraints

This project will be implemented in Java JEE.

3.3.6.1 Availability

This service has to be available 24/7, and therefore need a dedicated server to manage taxi queues and forward all the request with the minimum down time possible.

3.3.6.2 Maintainability

There will be a large Documentation File in order to ensure that the application code will be easily maintainable and open to future changes and updates.

3.3.6.3 Portability

TaxiService should work on any operative systems which supports JVM and DBMS.

3.4. Security

3.4.1 External Interface Side

TaxiService application implements a login authentication to protect the information of users. The Password has to be at least 8 Characters long, with at least one number and one capital letter. No special characters are allowed. The system will require the User to actively change his password every year, and the new password has to meet the same requirement of length, number presence and capital letter presence.

3.4.2 Server Side

The division between the data files and the application should guarantee a good security measure for the TaxiService's Server. There are three main units: the application server, the database and the web server. Every unit should be protected by a firewall and all the User (both Customer and Taxi Drives) are forbidden to access directly to the DBMS.

3.4.3 Application Side

A good security measure would be the implementation of the SSH keys for the identification of the trusted computers without involving any password. Another fundamental security method is the implementation of the HTTPS protocol connection in order to guarantee communication confidentiality and integrity and also mutual authentication.

3.5 Possible Future Implementation

- Enhanced Security Policy. It could be a Two-factor authentication with a code sent by email or sms to the user, or a Smart-card authentication. It could also be biometric: the System could recognize fingerprints, retina or voice analysis.
- E-mail notification. E-mail notification should be present to confirm the reservation of a taxi or as a reminder for the User (Customer) of the time and location chosen.

- Enhance the managing of Taxi queues by extending the research for a Taxi in some of the nearby area which are not very busy with requests.
- A function which tells the user how many taxi are available in any area of the city, which constantly updates every minute.
- City Traffic Information.
- Possibility to rate the service offered by a certain User (Taxi Driver) and write a small comment on his profile page, visible by other Users.

3.6 Documentation

A series of documents will be published in order to organize the progress of our project and keep our professor constantly updated. They are:

- The R.A.S.D. (Requirement Analysis and Specification Document) which illustrates our objectives and domain functions and describes the whole system and the respective models, requirements and specifications.
- The Design Document, which will present all the models which we will use for our project (UML, Alloy, etc.).
- The Testing Report, which will present the results of the testing activity made by people of other groups.

4. Scenarios

Here we outline some possible scenarios for TaxiService.

- John's car is in for some fixing at his mechanic's garage, but he has to attend to an important corporate party and doesn't want to miss it. No one of his colleagues can give him a ride, so he is desperate: the walk to the party location is too long. Luckily, he heard about the city's new improvement to the taxi service: he starts up his pc and looks it up on his web browser.

He clicks on the TaxiService link and loads the website's home page.

He clicks on the "Sign Up" button, because he needs to register and create an account in order to use the service.

He quickly fills in the registration form, with his name, surname, phone number, username and chosen password.

The system accepts his registration and John can access to the user's home page.

- Jane's train gave up two stops before hers, and she is forced to walk to her office... or she would be, hadn't she signed up as a customer for TaxiService! She produces her smartphone and opens up the application.

She logs in with her username and her password, and then proceeds to tap on the "Request a taxi" button. The request is accepted, and she receives the license plate of the car that is soon coming to pick her up.

- Mark is a taxi driver, and his work day has just started. He takes his seat on his taxi car, logs in with his username and password, and waits for a call.

There is an incoming request for a ride from Fisherman Boulevard, number 45. He is free so he taps on the "Accept Request" button, turns on the engine and drives to the address.

- Anne is going to attend a huge party for her younger brother's degree. She intends to enjoy the night, and have a few drinks with her friends and family. She is a responsible young woman, however, so she already intends not to drive herself back home.

The party should end at about 1 A.M., so she opens TaxiService from her web browser, where she is already logged in, and clicks on the "Reserve a taxi" button.

She inputs the party location as origin point and her home address as destination, and selects 1 A.M. as the scheduled time.

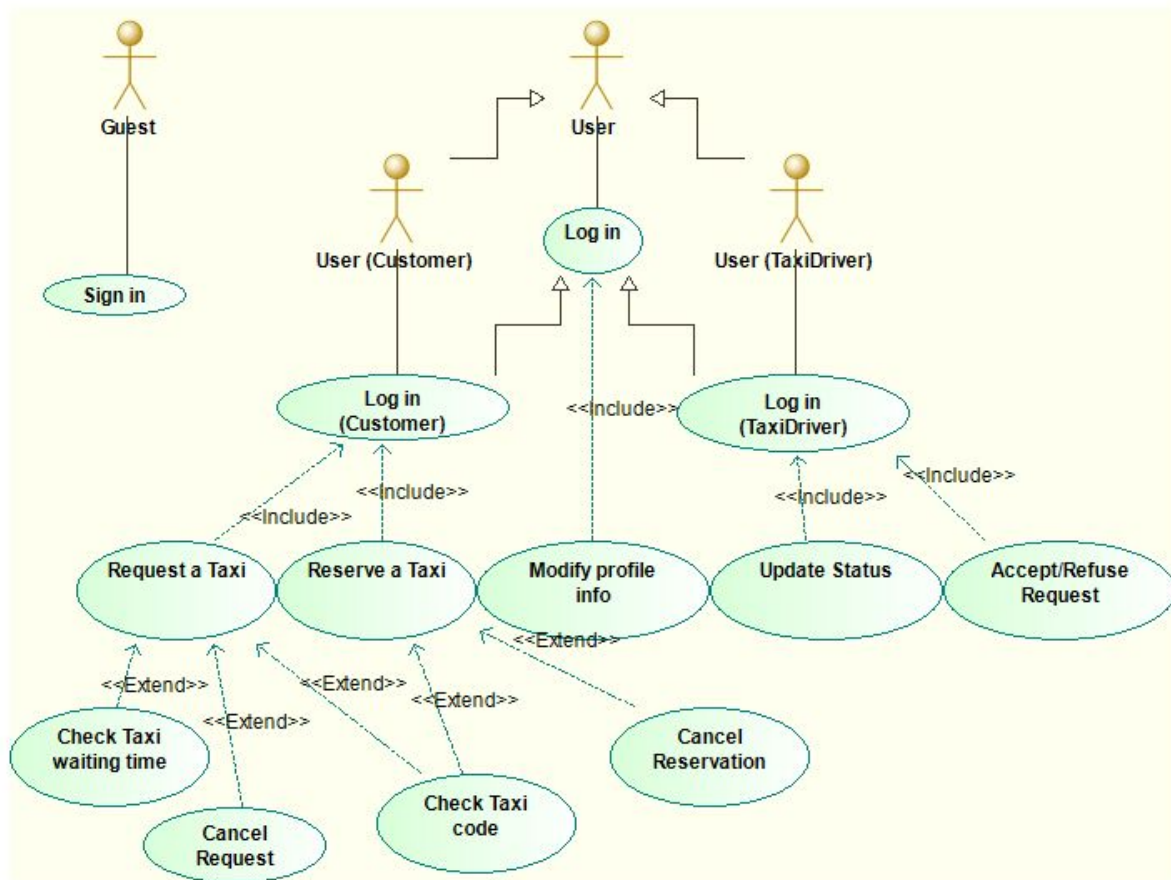
The system confirms the successful prenotation.

5. UML

We can easily infer some use cases from the aforementioned scenarios:

- Sign Up
- Login
- Taxi Request
- Taxi Reservation
- Request Acceptance
- Request Denial
- Modify personal information
- Visualize profile.

5.1 Use Case Diagram



5.2 Use Case Description

Name	Sign Up (Customer)
Actors	Guests
Entry Conditions	The Guest isn't registered to the application.
Flow of events	<ul style="list-style-type: none">• The guest enters the browser software/mobile application;• The guest writes in the form:<ul style="list-style-type: none">-Name-Surname-Email-Username-Password-Phone number• The guest clicks on the "As a Customer" button;• The system shows him his personal page.
Exit Conditions	Registration successfully done.
Exceptions	An exception can be caused if the username the guest inserts already exists or if some field that are not optional aren't filled.

Name	Sign Up (Taxi Driver)
Actors	Guests
Entry Conditions	The Guest isn't registered to the application.
Flow of events	<ul style="list-style-type: none"> • The guest enters the browser software/mobile application; • The guest writes in the form: <ul style="list-style-type: none"> -Name -Surname -Email -Username -Password -Phone number • The guest clicks on the "As a Taxi Driver" button; • The guest fills the new form with his taxi license code and his license plate number. • The system shows him his personal page.
Exit Conditions	Registration successfully done.
Exceptions	An exception can be caused if the username the guest inserts already exists, the license code/plate number is invalid or if some field that are not optional aren't filled.

Name	Login (Customer)
Actors	User (Customer)
Entry Conditions	The User has yet to login to the application.
Flow of events	<ul style="list-style-type: none"> • The user enters the website/mobile application. • The user fills in the text fields in the home page with username and password. • The user checks the “As a Customer” button. • The user clicks on the “Login” button.
Exit Conditions	The system shows the user his profile page.
Exceptions	Wrong Username or Password. Message is sent to the User.

Name	Login (Taxi Driver)
Actors	User (Taxi Driver)
Entry Conditions	The User (Taxi Driver) has yet to login to the application.
Flow of Events	<ul style="list-style-type: none"> • The user enters the website/mobile application. • The user fills in the text fields in the home page with username and password. • The user checks the “As a Taxi Driver” button. • The user clicks on the “Login” button.
Exit Conditions	The system shows the user his profile page.
Exceptions	Wrong Username or Password. Message is sent to the User.

Name	Modify User Profile (Customer)
Actors	User (Customer)
Entry Conditions	The User (Customer) is already logged in to the application.
Flow of events	<ul style="list-style-type: none"> • The User (Customer) press the “Change Profile Info”. • The guest then can modify in the form: <ul style="list-style-type: none"> -Name -Surname -Email -Password -Phone number -Address -Profile Image • The guest clicks on the “Confirm Changes” button; • The system shows him his renewed personal page.
Exit Conditions	Profile infos changes successfully done.
Exceptions	An exception can be caused if the username the guest inserts already exists or if some field that are not optional aren't filled.

Name	Modify User Profile (Taxi Driver)
Actors	User (Taxi Driver)
Entry Conditions	The User (Taxi Driver) is already logged in to the application.
Flow of events	<ul style="list-style-type: none"> • The User (Taxi Driver) press the “Change Profile Info”. • The guest then can modify in the form: <ul style="list-style-type: none"> -Name -Surname -Email -Password -Phone number • The guest clicks on the “Confirm Changes” button; • The system shows him his renewed personal page.
Exit Conditions	Profile infos changes successfully done.
Exceptions	An exception can be caused if the username the guest inserts already exists or if some field that are not optional aren't filled.

Name	Taxi Request
Actors	User (Customer)
Entry Conditions	The User (Customer) is already logged in to the application.
Flow of Events	<ul style="list-style-type: none"> • The User press the “Request a Taxi” Button from the profile change. • The User selects from which streets the taxi will pick up the user. • The user press the yellow “Ok” Button to confirm the address.
Exit Conditions	The request is completed and the system shows the user his taxi’s license plate and an estimate of the elapsed time to his arrival.
Exceptions	Wrong address. Message is sent to the User.

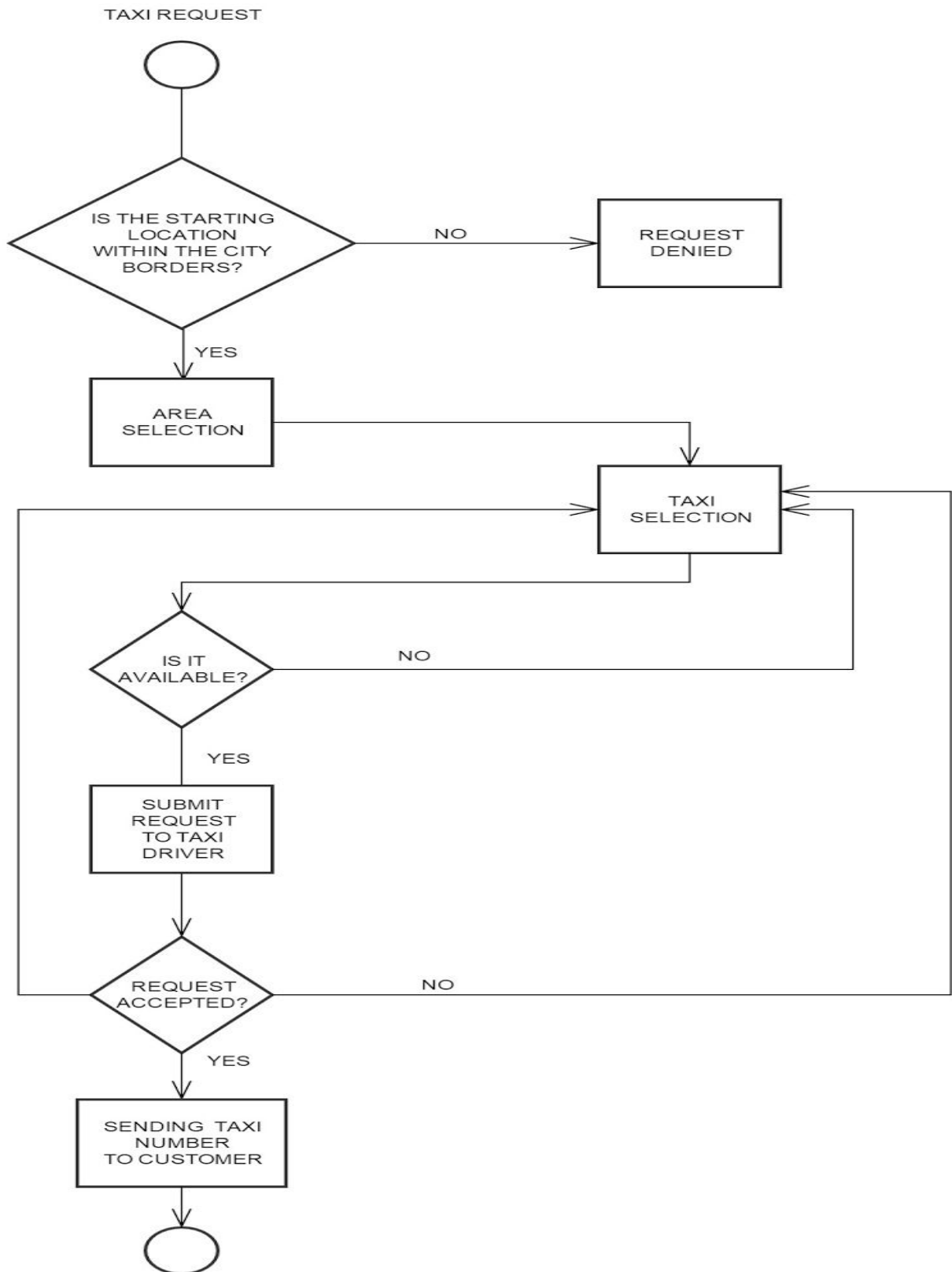
Name	Taxi Reservation
Actors	User (Customer)
Entry Conditions	The User (Customer) is already logged in to the application.
Flow of Events	<ul style="list-style-type: none"> • The User press the “Reserve a Taxi” Button from the profile change. • The User selects from which streets the taxi will pick up the user, the time in which it has to be there and its future destination. • The user press the yellow “Ok” Button to confirm the address.
Exit Conditions	The reservation is completed and the system shows the user so.
Exceptions	<ul style="list-style-type: none"> • Wrong address. Message is sent to the User. • Invalid Time. Message is sent to the User. • Wrong Destination. Message is sent to the User.

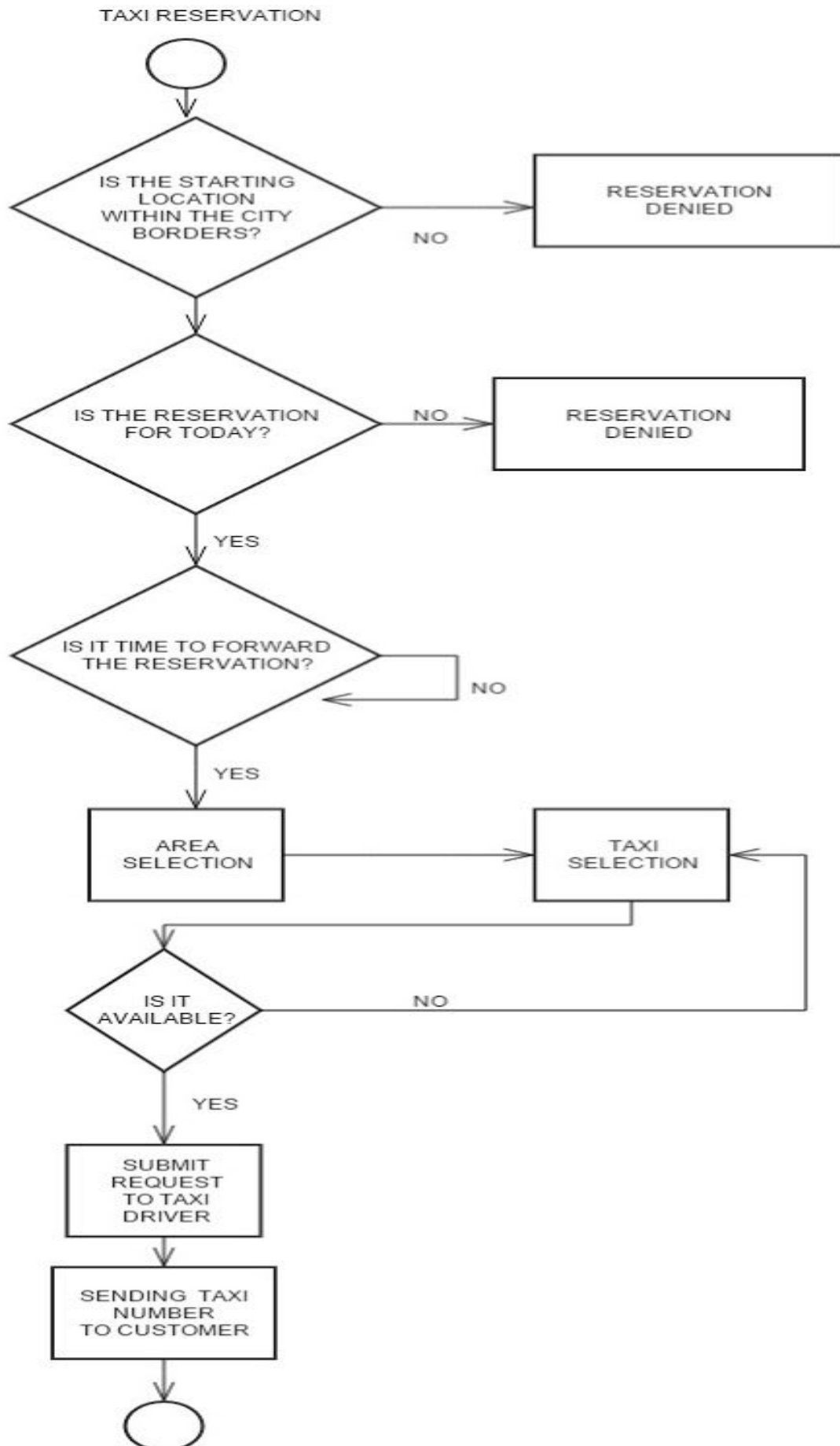
Name	Taxi Request Received (Accept case)
Actors	User (Taxi Driver)
Entry Conditions	The User (Taxi Driver) is already logged in to the application. His status is marked as “Available”.
Flow of Events	<ul style="list-style-type: none"> • The User (Taxi Driver) receives a Customer request, viewing the address he have to reach in order to pick up the customer. • The User (Taxi Driver) press the “Accept Request” button.
Exit Conditions	The request is successfully accepted.
Exceptions	None.

Name	Taxi Request Received (Refuse case)
Actors	User (Taxi Driver)
Entry Conditions	The User (Taxi Driver) is already logged in to the application. His status is marked as “Available”.
Flow of Events	<ul style="list-style-type: none"> • The User (Taxi Driver) receives a Customer request, viewing the address he have to reach in order to pick up the customer. • The User (Taxi Driver) press the “Refuse Request” button.
Exit Conditions	The request is successfully refused.
Exceptions	None.

Name	Taxi Reservation Received
Actors	User (Taxi Driver)
Entry Conditions	The User (Taxi Driver) is already logged in to the application. His status is marked as "Available".
Flow of Events	<ul style="list-style-type: none"> • The User (Taxi Driver) receives the Customer reservation ten minutes before the time the Customer set, viewing the address he has to reach in order to pick up the customer.
Exit Conditions	The reservation has been successfully notified.
Exceptions	None.

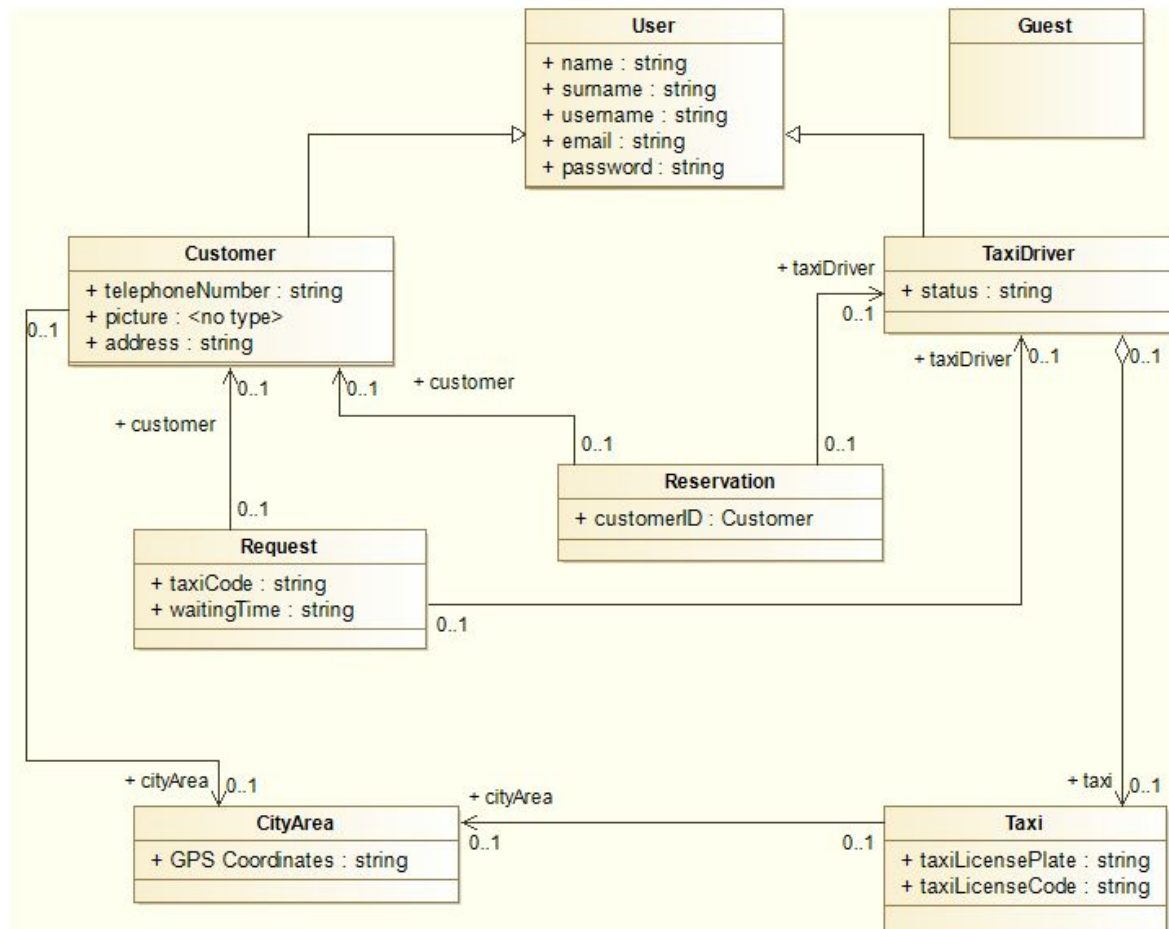
5.3 Flowgraph Diagrams



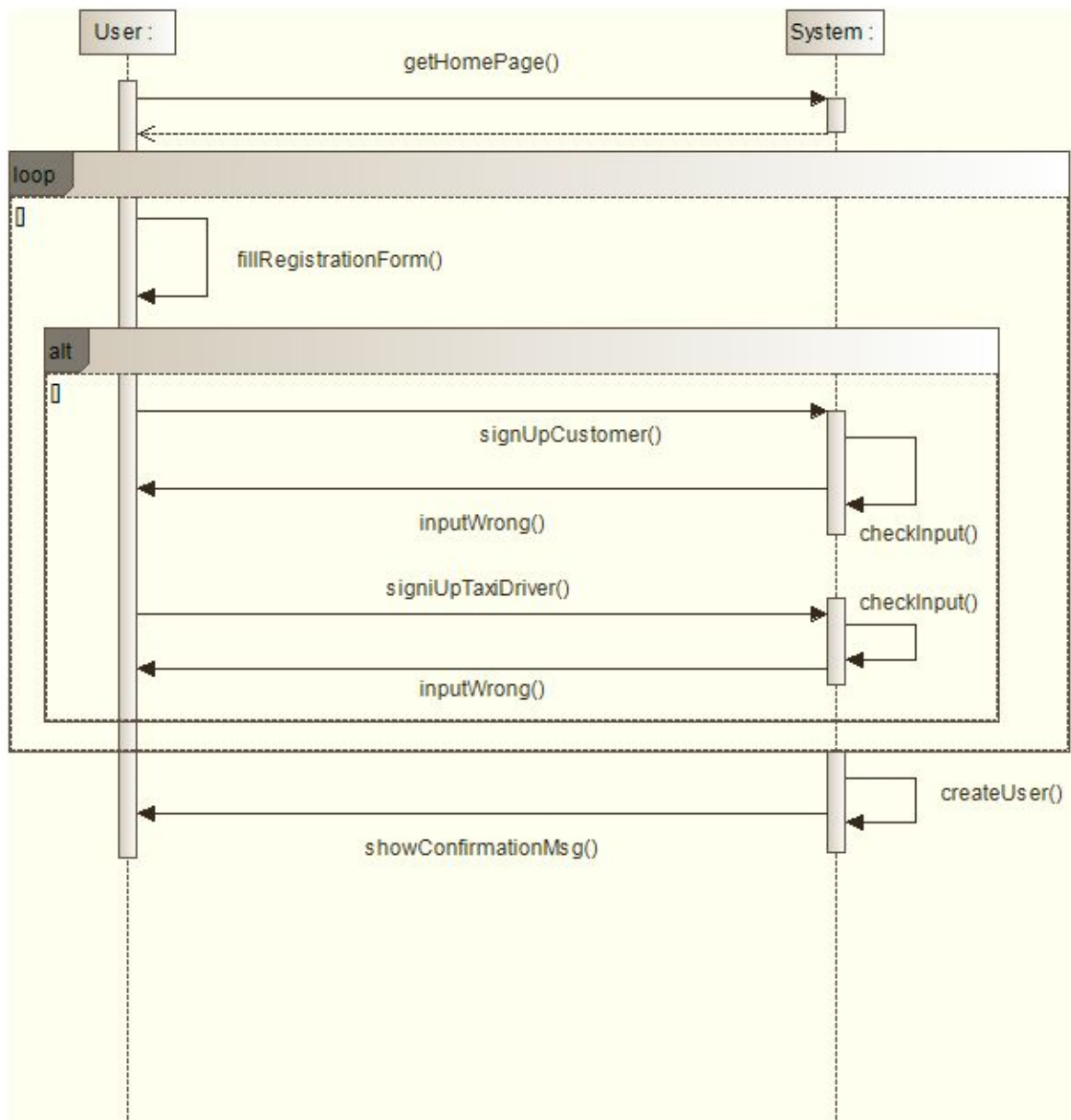


5.4 UML Models

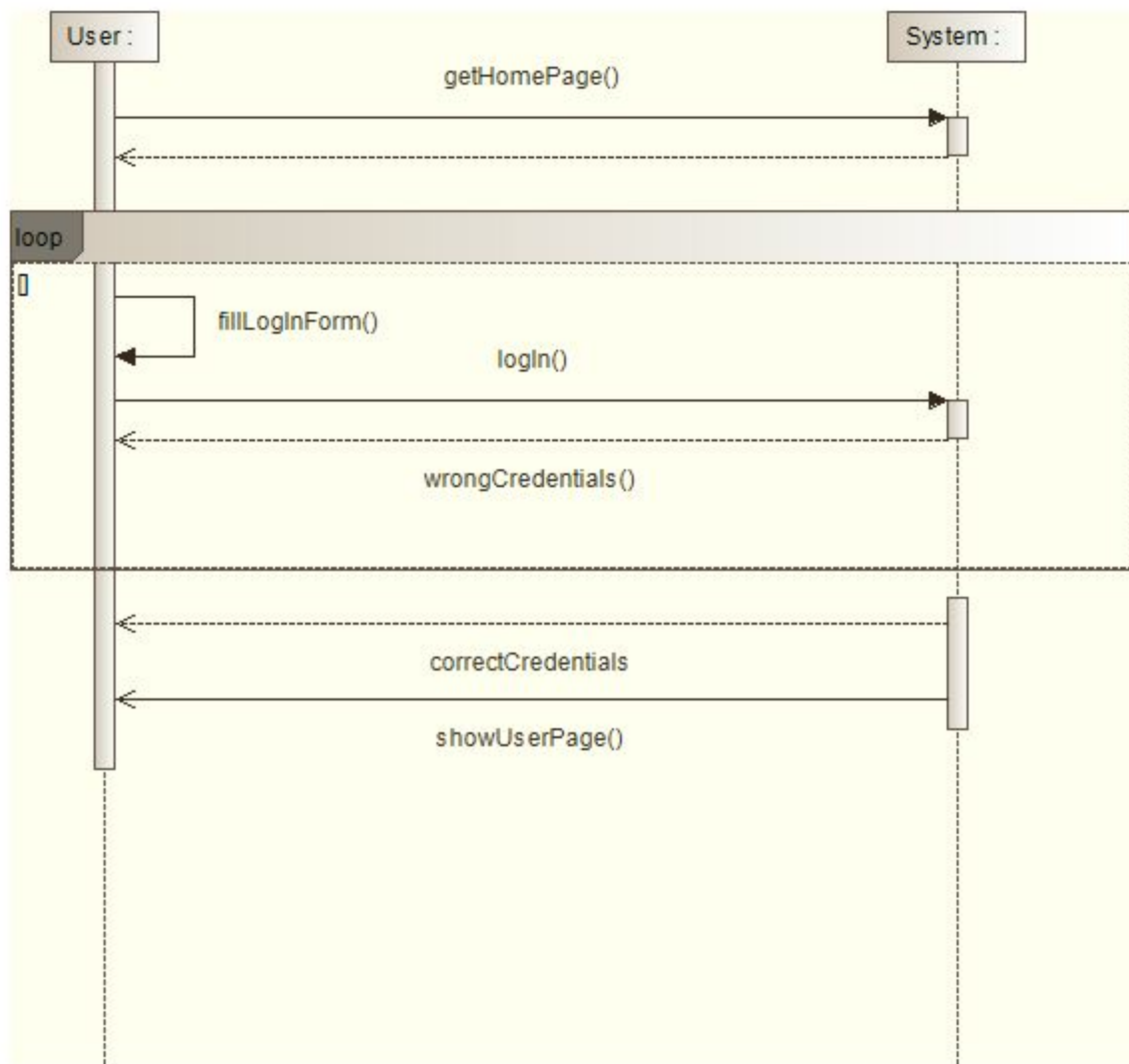
5.4.1 Class Diagram



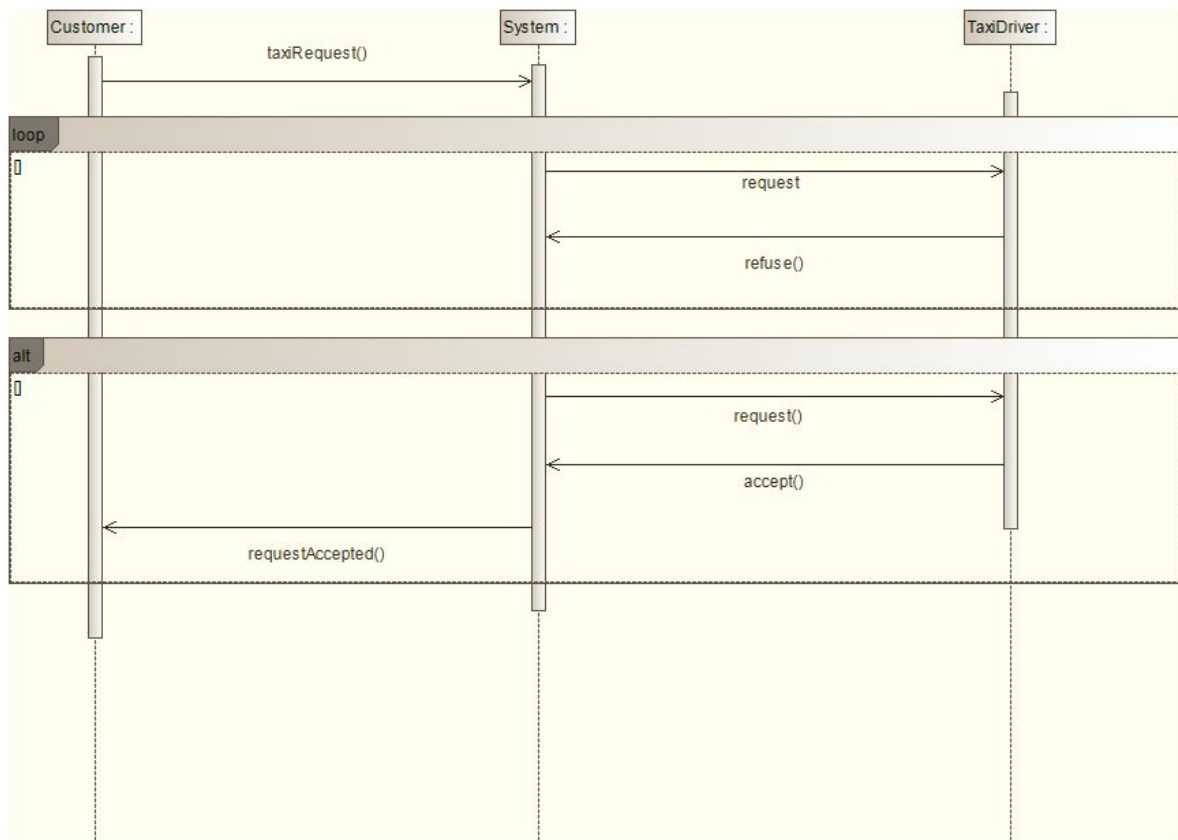
5.4.2 Sequence Diagram - Sign Up



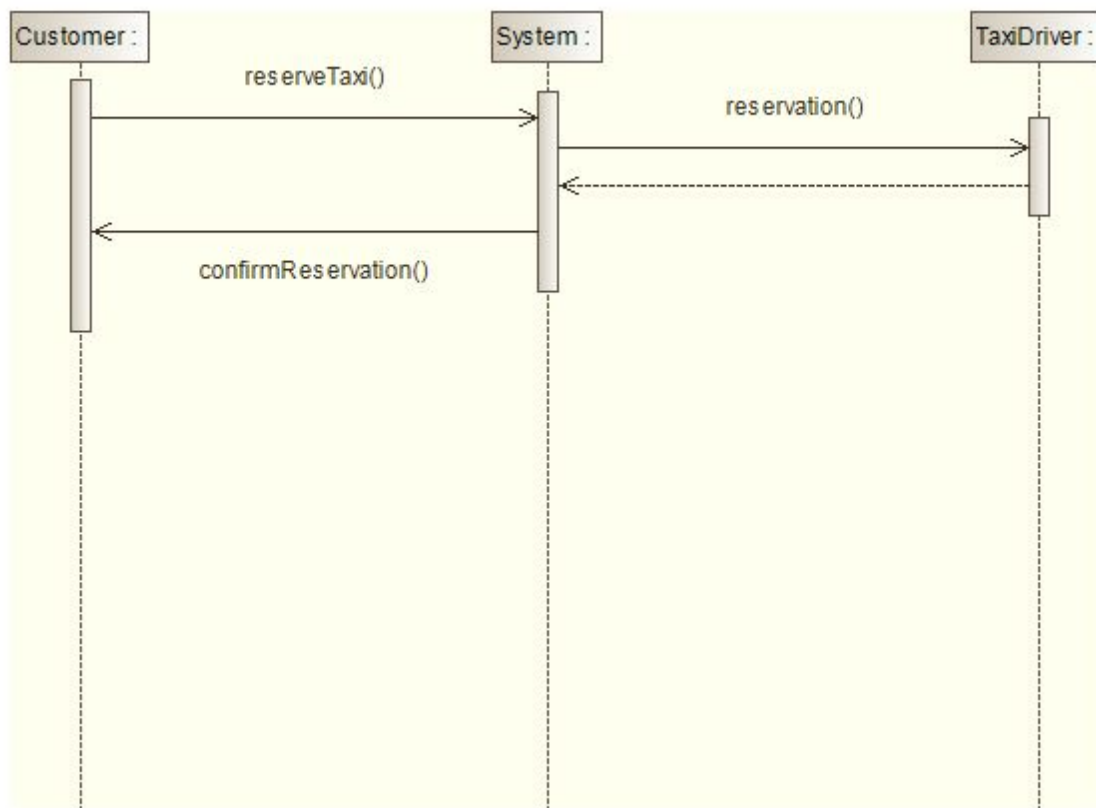
5.4.3 Sequence Diagram - Login



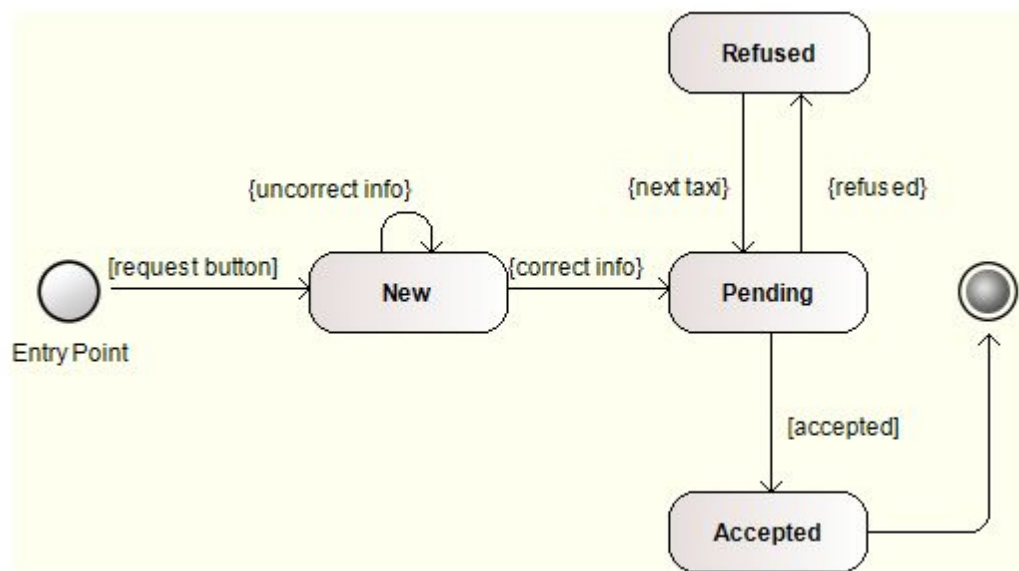
5.4.4 Sequence Diagram - Taxi Request



5.4.5 Sequence Diagram - Taxi Reservation



5.5 Finite State Diagram



6. Alloy Model

6.0.1 Signatures

```
sig Integer{}
```

```
sig Strings{}
```

6.0.2 Abstract Entities

```
//abstract class of TaxiCall
```

```
abstract sig TaxiCall{  
    code: one Integer,  
    owner: one UserCustomer,  
    addr: one Location,  
    time: one EventTime,  
    status: one Strings,  
    associatedTaxiDriver: one UserTaxiDriver  
}
```

```
//abstract class of request
```

```
abstract sig Request extends TaxiCall{}
```

```
//abstract class of reservation
```

```
abstract sig Reservation extends TaxiCall{}
```

```
//abstract class of user
```

```
abstract sig User{}
```

```
//abstract generic date-time class
```

```
abstract sig EventTime{  
    year: one Integer,  
    month: one Integer,  
    day: one Integer,  
    hour: one Integer,  
    minutes: one Integer  
}
```

```
//abstract class of notify
```

```
abstract sig Notify{  
    code: one Integer,  
    message: one Strings  
}
```


6.0.3 Implementations and Local Signatures

//class of Location

```
sig Location{  
    street: one Strings,  
    streetNum: one Integer,  
    cityArea: one CityArea  
}
```

//class of Icon

```
sig Icon {  
    path: one Strings  
}
```

//class of Notification

```
sig Notification extends Notify{  
    receiver: one User,  
    sender : one TaxiCall  
}
```

//class of Guest

```
sig Guest extends User{}
```

//class of UserCustomer

```
sig UserCustomer extends User{  
    name: one Strings,  
    surname: one Strings,  
    email: one Strings,  
    phonenumber: one Integer,  
    address: one Strings,  
    username: one Strings,  
    password: one Strings  
}
```

//class of UserTaxiDriver

```
sig UserTaxiDriver extends User{  
    name: one Strings,  
    surname: one Strings,  
    email: one Strings,  
    phonenumber: one Integer,  
    address: one Strings,  
    username: one Strings,  
    password: one Strings,  
    licenseNumber: one LicenseNumber,  
    licensePlate: one LicensePlate,  
    status: one Strings,  
    currentPosition: one Location  
}
```

```
//class of LicenseNumber
sig LicenseNumber {
    licenseNumber : one Integer
}
```

```
//class of LicensePlate
sig LicensePlate {
    licensePlate : one Strings
}
```

```
//class of CityArea
sig CityArea {
    cityAreaId : one Integer,
    gpsCoordinates : one Strings
}
```

6.0.4 Facts

//EventTime must not be empty

```
fact noEmptyDate {  
    all et : EventTime | (#et.year = 1) and (#et.month = 1)  
        and (#et.day = 1) and (#et.hour = 1) and (#et.minutes = 1)  
}
```

//Locations must not be empty

```
fact noEmptyLocations {  
    all l : Location | (#l.street = 1) and (#l.cityArea = 1) and (#l.streetNum = 1)  
}
```

//Notify must not be empty

```
fact noEmptyNotify {  
    all n : Notify | (#n.message = 1)  
}
```

//TaxiCall must not be empty

```
fact noEmptyTaxiCall {  
    all tc : TaxiCall | (#tc.code = 1) and (#tc.addr = 1)  
        and (#tc.owner = 1) and (#tc.status = 1) and (#tc.associatedTaxiDriver = 1)  
        and (#tc.time = 1)  
}
```

//CityArea must not be empty

```
fact noEmptyCityArea {  
    all ca : CityArea | (#ca.cityAreaId = 1) and (#ca.gpsCoordinates = 1)  
}
```

//LicensePlate must not be empty

```
fact noEmptyLicensePlate {  
    all lp : LicensePlate | (#lp.licensePlate = 1)  
}
```

//LicenseNumber must not be empty

```
fact noEmptyLicenseNumber {  
    all ln : LicenseNumber | (#ln.licenseNumber = 1)  
}
```

//Notification must not be empty

```
fact noEmptyNotification {  
    all nf : Notification | (#nf.sender = 1) and (#nf.receiver = 1)  
}
```

//No Notify clones

```
fact noClonedNotify {  
    no disj n,m: Notify | n.code = m.code  
}
```

```

//No LicensePlate clones
fact noClonedLicensePlate {
    no disj lp1, lp2 : LicensePlate | lp1.licensePlate = lp2.licensePlate
}

//No LicenseNumber clones
fact noClonedLicenseNumber {
    no disj ln1, ln2 : LicenseNumber | ln1.licenseNumber = ln2.licenseNumber
}

//No CityArea clones
fact noClonedCityArea {
    no disj ca1, ca2 : CityArea | ca1.cityAreaId = ca2.cityAreaId
    or ca1.gpsCoordinates = ca2.gpsCoordinates
}

//No Location clones
fact noClonedLocation {
    no disj loc1, loc2 : Location | loc1.street = loc2.street
    and loc1.cityArea = loc2.cityArea
    and loc1.streetNum = loc2.streetNum
}

//No TaxiCall clones
fact noClonedTaxiCall {
    no disj tc1,tc2: TaxiCall | tc1.code=tc2.code
}

//No UserCustomer clones
fact noClonedUserCustomers {
    no disj uc1,uc2: UserCustomer | (uc1.email = uc2.email)
    or (uc1.username = uc2.username)
}

//No UserTaxiDriver clones
fact noClonedUserTaxiDrivers {
    no disj ut1,ut2: UserTaxiDriver | (ut1.email = ut2.email)
    or (ut1.username = ut2.username)
    or (ut1.licensePlate = ut2.licensePlate)
    or (ut1.licenseNumber = ut2.licenseNumber)
}

//No Customers as TaxiDriver or viceversa
fact noHybridUsers {
    no disj ut1 : UserTaxiDriver, uc2 : UserCustomer | (ut1.email = uc2.email)
    or (ut1.username = uc2.username)
}

//There can be no EvenTime without TaxiCalls
fact noEventWithoutEvents {
    all et : EventTime | (one tc : TaxiCall | tc.time = et)
}

```

```

}

//There can be no LicensePlates without TaxiDrivers
fact noPlatesWithoutDrivers {
    all lp : LicensePlate | (one td : UserTaxiDriver | td.licensePlate = lp)
}

//There can be no LicenseNumber without TaxiDrivers
fact noLicenseNumbersWithoutDrivers {
    all ln : LicenseNumber | (one td : UserTaxiDriver | td.licenseNumber = ln)
}

//The CityArea for a call must be the same of the driver
fact sameCityArea {
    all td : UserTaxiDriver, tc : TaxiCall | td.currentPosition.cityArea = tc.addr.cityArea
}

//Ensures that on every call there is one notification sent to the taxi driver and one to the customer
fact NotificationManagement {
    all tc : TaxiCall | (some disj nf1,nf2 : Notification | nf1.sender = nf2.sender
        and nf1.receiver = tc.owner
        and nf2.receiver = tc.associatedTaxiDriver)
}

//Ensures that every call send notifications
fact NotifyMe {
    all tc : TaxiCall | (some n1 : Notification | n1.sender = tc
        and (n1.receiver = tc.owner or n1.receiver = tc.associatedTaxiDriver))
}

//These two facts ensure that the password is a separated
//string from the account information, for security reasons
fact BasicSecurity1 {
    all uc : UserCustomer | uc.password != uc.username
        and uc.password != uc.name and uc.password != uc.surname
        and uc.password != uc.email
}
fact BasicSecurity2 {
    all utd : UserTaxiDriver | utd.password != utd.username
        and utd.password != utd.name and utd.password != utd.surname
        and utd.password != utd.email
}

```

6.0.5 Assertions

//Checks that there are no two distinct taxi calls to the same taxi driver at the same time

```
assert noOverlappingTaxiCall {  
    no disj tc1, tc2 : TaxiCall | tc1.associatedTaxiDriver = tc2.associatedTaxiDriver  
        and tc1.time = tc2.time  
}  
check noOverlappingTaxiCall
```

//Checks that no single customer can make two different taxi calls from two different locations at the same time

```
assert noUbiquity {  
    no uc : UserCustomer | (some disj tc1, tc2 : TaxiCall | tc1.owner = tc2.owner  
        and tc1.owner = uc and tc1.addr != tc2.addr  
        and tc1.time = tc2.time)  
}  
check noUbiquity
```

//Checks that no single customer can make two calls at the same time

```
assert noMultipleCalls {  
    no disj tc1, tc2 : TaxiCall | tc1.owner = tc2.owner  
        and tc1.time = tc2.time  
}  
check noMultipleCalls
```

6.0.6 Predicates

//Shows a world with all actors in order to simulate all the basic interactions

```
pred showGeneric {  
    #UserCustomer > 2  
    #UserTaxiDriver > 2  
    #Request > 2  
}  
run showGeneric for 6
```

```
pred showHybridCall {  
    #UserCustomer = 1  
    #UserTaxiDriver = 1  
    #TaxiCall = 2  
}  
run showHybridCall
```

//Shows a world where there are at least two taxi calls not from the same customer

```
pred separateCalls {  
    some uc: UserCustomer, disj tc1, tc2 : TaxiCall | tc1.owner = uc  
    and tc2.owner != uc  
}  
run separateCalls for 10
```

6.1 Clarification

Here to clarify something about the alloy modelization: we used a single Strings to describe the gpsCoordinates of a CityArea, because we intend those coordinates as the upper-left corner of the $2km^2$ area.

6.2 Report and Generated Worlds

Here we present the worlds generated by running two of the three predicates (run separateCalls for 10 yields a huge image, and represents a more general situation), and the results Alloy Analyzer gave when checking the asserts and running the predicates.

6 commands were executed. The results are:

- #1: **Instance found.** showGeneric is consistent.
- #2: **Instance found.** showHybridCall is consistent.
- #3: **Instance found.** separateCalls is consistent.
- #4: No counterexample found. noOverlappingTaxiCall may be valid.
- #5: No counterexample found. noUbiquity may be valid.
- #6: No counterexample found. noMultipleCalls may be valid.

6.2.1 Generated World for Predicate “showGeneric”

6.2.2 Generated World for Predicate “showHybridCall”

Due to the huge dimensions of the .png files of the Generated Worlds, we put them as attached files in the Deliveries folder.

7. Used tools

The main tools we used to create this RASD document are:

- Google Drive Documents: to write and redact this document;
- Modelio: to create the Class, State and the Use Case Diagrams.
- Gliffy Editor: to create the Flowgraph Diagrams.
- Alloy Analyzer 4.2: to prove the consistency of our model.

In order to redact and write this document we spent approximately 30 hours per person