Taxi service

Requirements Analysis and Specification Document

Authors:

Liu Yuqi
24HOURS

Xu Xiuli
24HOURS

November 5th 2015

Summary

1. Introduction

1.1 Description of the given problem

We will project and implement Taxi Service APP, which is a kind of vehicle scheduling system ,which helps passengers to take or reserve a taxi more convenient.

The system that will be developed has to simplify the access of passengers to the service and guarantee a fair management of taxi queues. Passengers can request a taxi either through a web app or mobile app.The taxi drivers use this app to inform the system about their availability and to confim that they are going to take care of a certain call .The passenger will receive the information about the taxi and when the taxi will arrive once a driver confirms the request.

1.2 Goals

If we think about possible users(passengers,drivers), we think that Taxi Service APP has to provide this main features:

For the passengers:

- Register as a User;
- Log into APP using the mobile phone number ;
- Request a taxi;
- Reserve a taxi by specifying the origin and the destination of the ride;
- Checking the information about the requested taxi ;
- Cancel a request or cancel a reservetion.

For the driver:

- Inform the system about the taxi's status(avaliability);
- Confirm or refuse an request or a reservation;

Also ,for the system ,the main feature:

- Answer the request (by informing the passenger the code of incoming-taxi);
- Automatically distributes the taxi in the various city zones (based on the GPS);
- Manage the taxi queues of different zones in the system ;
- Reject a request or a reservation;

1.3 Constraints

We will analyze the backgroud about this Taxi service system:

- Passengers will pay by cash ,so the system will not include the function of the payment;
- A passenger can only request a taxi until the passenger cancel or finish the ride;
- A passenger and the driver will contact by the phone after a request is confirmed ;
- A person who wants to use Taxi server App must have a mobile phone number, and must have a condition to access the Internet;
- Only on the situation of Reserve a taxi, a passenger can helps the other person use this taxi service .

1.4 Glossary

First of all we have to define some words that will be used in our documents.

- **Zones/City zones**: A taxi is belong to a zone of this city ,which is approximately 2 km2;

- **GPS**: Global positioning system.The system can receive the GPS information from each taxi in order to record where the taxi is.
- **Taxi queues**: A queue that record the order of the avaliable taxis. Each zone has a such taxi queue.when a request from a certain zone, the system forwards it to the first taxi queue in that zone.If the taxi confirms , then the system will send a confirmation to the passenger. If not, then the system will forward therequest to the second in the queue, and move the first taxi in the last
- **Passenger :** for passenger we mean a person who would like to use this APP to request or reserve a taxi service .A passenger should use his mobile phone number as the username to login the system.
- **Driver:** a driver is a person who is the driver of a taxi and who is able to deal with an request from the passenger through the Taxi service APP
- **Request a taxi**: A passenger who would like to take a taxi can use the requst a taxi button to order a taxi immiediately.
- **Reserve a taxi:** A passenger who would like to reserve a taxi must submit the reservation ahead of 2 hours before the ride. The system confirms the reservation to the user and allocates a taxi to the request 10 minutes before meeting time with user.

1.5 Proposed system

We propose a web platform and a mobile platform, but we think the main platform is the mobile platform ,because it is more convenient for a passenger. The passengers can use web terminal and mobile terminal ,which is according to the passengers preference. The driver , generally speaking, will use the mobile terminal because it is convenient for a driver who always working in a taxi . The administrator will be given a right to manage information and so on. Besides the specific user interfaces for passengers and taxi drivers ,the system offers also programmatic interfaces to enable the development of additional services.

1.6 Identifying Stakeholders

Our main stakeholder is the professor, who gave us the delivery of the project. and expects us to be able to finish the project within this semester.The professor asks us to focus on the whole development process of a complex enterprise application,which involves the following phases:
requirement analysis,design,implementation,testing and project reporting .So we will try to concentrate equally on each phase,not only on the implementation of the final product but also on the documentationl.We will focus first on the main functionality of the system, which are directly requested in the specification given to us by the professor ,then we will try to extend the functionality of the system in order to make the application as close as possible to an application that is ready to be launched in the market.

Starting from this considertation we can imagine that an hypothetical stakeholder for our system could be a company that would have asked us to realisze a platform to manage employees appointment in a more efficient way.

1.7 Other considerations about the system

We want to add these other considerations because they are about our thoughts about how taxi service APP should be once terminated.

So we think Taxi service APP has to be:

- Easy to use: we will try to “make it simple”, because the goal of the app is simplify the access of passengers to the taxi service .so we want the interface very simple.
- Have a nice look and feel: we will try to make a well designed application in the sense that we hope it will be nice to see and will fit to any user.

2. Actors Identifying

The actors of our system are basically three:

- **Passenger**: a passenger ,as we already mentioned in the glossary sect**ion** , is someone who sends a request for taking a taxi or makes an reservation.
- **Driver**: a driver ,as we alredady mentioned in the glossary section ,is the driver of a taxi ,who is able to deal with a taxi request .
- **External Developer**: a developer after login can access the API of the system

3. Requirements

General system requirements:

The system must simplify the access of passengers to the service. The system must guarantte a fair management of taxi queues.The system must update the taxi queues of different city zones, according to the taxi positons reported by GPS when a taxi's status is avaliable.The system must manage the taxi queues when a taxi driver confirms or reject an taxi request.The system must dispath the first taxi of the taxi queues in this zone to the passenger.After the driver confirm a request , the system answer the request by sending the taxi information(taxi code ,waiting time, driver's mobilephone number)to the passenger.

3.1 Functional Requirements

We list the functional requirements concerning each defined actor:

- **Passenger**: he can:
  - Register as a user;
  - Log into the APP using the mobile phone number;
  - Request a taxi;
  - Reserve a taxi by specifying the origin and destination(The reservation has to occur at least 2 hours before the ride);
  - Checking the information about the requested taxi;
  - Cancel a request or cancel a reservation;
- **Driver**: he can:
  - Register as a driver;
  - Log in;
  - Inform the system about the status of taxi(whether it is avaliable);
  - Confirm or refuse a request or a reservation;
  - Checking the reservation information of all the passengers he confirmed;
- **System**: can:
  - Automatically distribute the taxi into the taxi queues of different city zones, according to the taxi positons reported by GPS when a taxi's status is avaliable.;
  - Manage the taxi queues.If the driver reject a request of taxi service , the system will forward the request to the second of this taxi queue,and move the first taxi to the last position of this queue;
  - Answer a request.If the texi driver confirms a request ,then the system will send a confirmation to the passenger,also

- The system confirms the reservation to the passenger and allocates a taxi to the passenger 10 minutes before the meeting time.

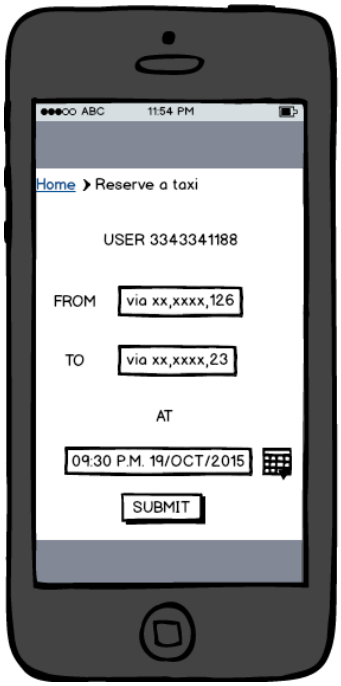
- **Administrator**: he can:
  - Log in;
  - Maintenance the taxi information(for example, add a taxi information and so on);

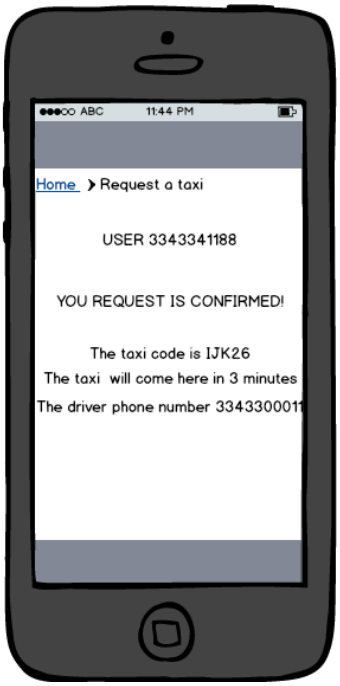
3.2 Non Functional Requirements

- Mo**b**ile applications s**h**all **b**e developed for Android mo**b**ile platform
- Server application s**h**all **b**e developed usin**g java** tec**h**nolo**g**y
- Mo**b**ile App interface s**h**ould **b**e simple and user friendly

3.2.1 The User interface

The user (passenger) interface:





3.2.2 Documentation

We will draft these documents to well-organize our work in the way to do in a fewer time the best work as possible:

- **RASD**: Requirement Analysis and Specification Document, to well-understand the given problem and to analyze in a detailed way which are our goals and how to reach them defining requirements and specification.
- **DD**: Design Document, to define the real structure of our web application and its tiers.
- **JavaDoc comments in the source code**: to make anyone that wants to develop the platform or do maintenance on it understand the code.
- **Testing Document**: a report of our testing point and result.

3.2.3 Architectural considerations

We will use J2EE platform with a database in which all system's information will be stored.

What is this information will be clearer watching the Class Diagram given below, because it could be considered as a basis for our database, but the precise ER Diagram will be drawn in the DD.

4. Scenarios Identifying

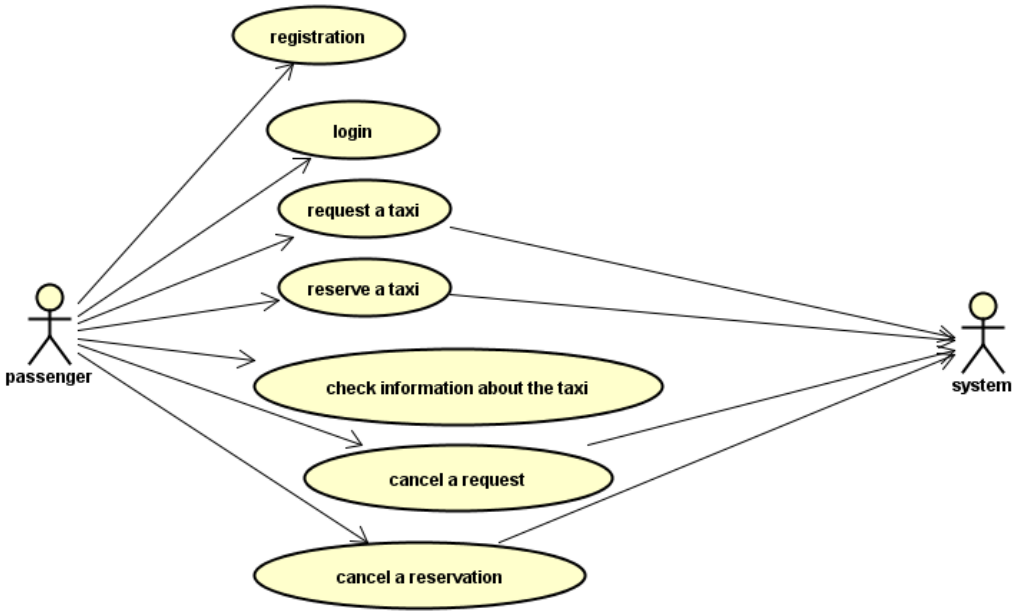
Here are some possible scenarios of Taxi service APP:

- Mike is off work, and he wants to take a taxi to home,because it is 10p.m. So he open the mobileAPP and press the “request a taxi now” button. (of course his mobile phone is able to connect the Internet) . And 30 seconds later, the screen of the mobile shows “congratulations, your order is confirmed by a taxi driver , the taxi code is xxxx, his phone number is xxxx, you have to wait about 3minutes. Also , the driver will contact you by a call. 1minute later, the driver calls mike and ask the exact address.2minutes later , the taxi comes ,mike take on the taxi and tell the driver where he wants to go. And mike arrives his home successful.
- Lucy’s grandpa wants to go to hospital to do a physical examination today after noon . His appoint time is 3p.m. So Lucy open the web APP of the taxi service .Login the system using his grandpa’s mobile phone number. She fill in the information of the origin and the destination and make a pointment at 2p.m.. 1minutes later ,she got the confirmation message and the information of the taxi. About 1:50p.m, here comes the taxi .Her grandpa get on the taxi and get to the hospital successful.
- Tom wants to go to the school by taxi because he get up late. So he open the mobile taxi service app ,and request a taxi . But after he confirm his request ,he remember today is saturday**!** But he just order a taxi, so he press the” cancel your request “button. And he call the driver by the mobilephone number showed on the screen. He explain the reason and say sorry to the driver.

- Leon is a taxi driver. now it is the 2.p.m .He wants to go home to have a rest. but now the screen of the mobile shows there is a taxi request ,so he press the refuse button. and drive his car to his home.

5. UML Models

5.1Use Case Passenger



5.1.1 Use Case “registration”

Use Case ID	PASSENGER1
Name	Registration
Goal	Register to the application to be able to order/reserve a taxi
Participating actors	Server and passengers
Precondition	Not registered before
Main scenario	1.A passenger who has not registered before register to the system 2.Application accept the registration and send it to server 3.Server confirm the information of the visitor 4.The passenger become a user of the application and the system
Exceptions	
Extensions	
Dependent UC	

5.1.2 Use Case “log in”

Use Case ID	PASSENGER2
Name	Log in
Goal	Log into the application to be able to order/reserve a taxi
Participating actors	System and passengers
Precondition	
Main scenario	1.A passenger clicks the ‘log in’ button to log into the application
Exceptions	
Extensions	
Dependent UC	

5.1.3 Use Case “request a taxi”

Use Case ID	PASSENGER3
Name	Request a taxi
Goal	To order a taxi
Participating actors	System and passengers
Precondition	
Main scenario	1.A passenger clicks the button for ordering a taxi 2.Application sends the GPS coordinates to the system
Exceptions	
Extensions	
Dependent UC	

5.1.4 Use Case “reserve a taxi”

Use Case ID	PASSENGER4
Name	Reserve a taxi
Goal	To reserve a taxi
Participating actors	System and passengers
Precondition	The passenger should reserve a taxi 2 hours earlier
Main scenario	1.A passenger clicks the button for reserving a taxi 2.Application sends the GPS coordinates to the system 3.The system will attribute a taxi in free status from the taxi queue according to the city zone
Exceptions	
Extensions	
Dependent UC	

5.1.5 Use Case “Check information about a taxi”

Use Case ID	PASSENGER5
Name	Check information about a taxi and its driver
Goal	Passenger can check information about the taxi driver’s name, position, code of taxi and phone number.
Participating actors	Passengers and taxi drivers
Precondition	The passenger has requested a taxi
Main scenario	1.The application can show information of the taxi driver and the taxi to a passenger once the passenger has requested or reserved a taxi.
Exceptions	
Extensions	
Dependent UC	

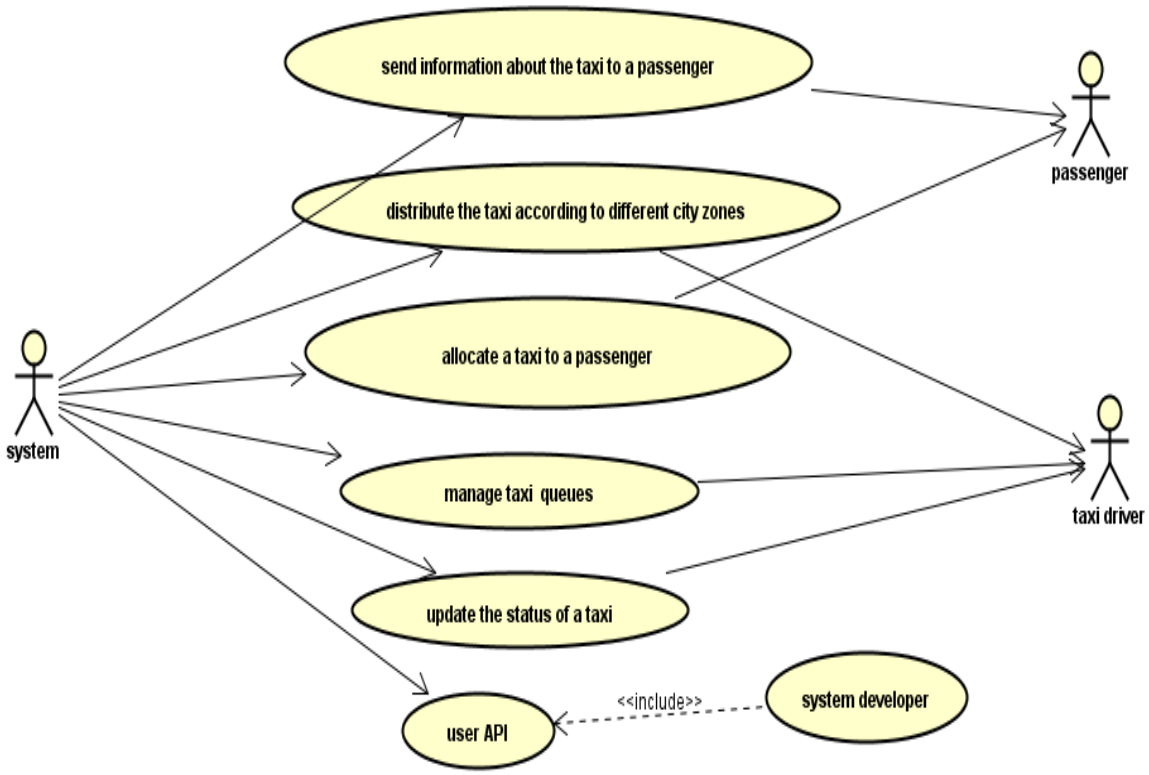
5.1.6 Use Case “cancel a request”

Use Case ID	PASSENGER6
Name	Cancel a request
Goal	Passenger can cancel the request for a taxi for any reason
Participating actors	Passengers and taxi drivers
Precondition	The passenger has requested a taxi
Main scenario	1.The passenger cancels an order before a taxi driver confirms the request. 2.The passenger cancels an request for a taxi after the taxi driver has confirmed the request, at this circumstance, the passenger has to call the taxi driver to cancel it.
Exceptions	
Extensions	
Dependent UC	

5.1.7 Use Case “cancel a reservation”

Use Case ID	PASSENGER7
Name	Cancel a reservation
Goal	Passenger can cancel a reservation for a taxi for any reason
Participating actors	Passengers and taxi drivers
Precondition	The passenger has reserved a taxi
Main scenario	1The passenger cancels a reservation before a taxi driver confirms it. 2The passenger cancels a reservation for a taxi after the taxi driver has confirmed the request, at this circumstance, the passenger has to call the taxi driver to cancel it.
Exceptions	
Extensions	
Dependent UC	

5.2 Use Case System



5.2.1 Use Case “send information about the taxi to a passenger”

Use Case ID	System1
Name	Send information about the taxi to a passenger
Goal	To inform the passenger about the taxi’s code and its driver’s name, phone number, address and credit and so on.
Participating actors	System and passengers
Precondition	The passenger has requested a taxi
Main scenario	1.The passenger has requested for a taxi, then the system will immediately allocate a taxi to the passenger according to the passenger’s location and at the same time show the driver’s information to the passenger
Exceptions	
Extensions	
Dependent UC	

5.2.2 Use Case “distribute a taxi according to different city zones”

Use Case ID	System2
Name	Distribute a taxi according to different city zones
Goal	To minimize the total waiting time for a passenger
Participating actors	System and taxi drivers
Precondition	The passenger has requested a taxi
Main scenario	1.The passenger has requested for a taxi 2.The system receives the request of a passenger for a taxi, it will ,first of all, check the location of the passenger using the GPS sent by the passenger’s mobile device. 3.The system then will automatically allocate a taxi for the passenger according to different city zones and change the status of the taxi and put it into the last position of the waiting queue.
Exceptions	
Extensions	
Dependent UC	

5.2.3 Use Case “allocate a taxi to a passenger”

Use Case ID	System3
Name	Allocate a taxi to a passenger
Goal	To allocate a taxi to a passenger
Participating actors	System and taxi drivers
Precondition	The passenger has requested or reserved a taxi
Main scenario	1.The system will allocate a taxi to a passenger by its location i.e. ,the zones the passenger is in.
Exceptions	
Extensions	
Dependent UC	

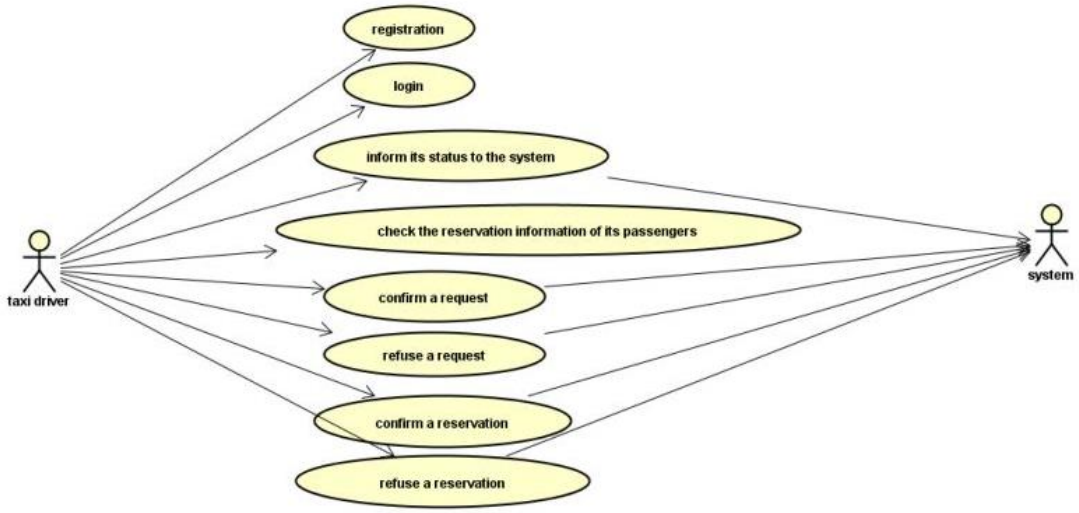
5.2.4 Use Case “manage taxi queues”

Use Case ID	System4
Name	Manage taxi queues
Goal	To optimize the performance of the whole system, to distribute taxis much more efficiently and make sure equality of the system.
Participating actors	System and taxi drivers
Precondition	
Main scenario	1.When a request arrives from a certain zone,the system forward it to the first taxi queuing in that zone. 2.If the taxi confirms a request from a passenger, then the system will send a confirmation to the passenger. 3.If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi to the last position in the queue.
Exceptions	
Extensions	
Dependent UC	

5.2.5 Use Case “update the status of a taxi”

Use Case ID	System5
Name	Update the status of a taxi
Goal	To change the states of a taxi
Participating actors	System
Precondition	
Main scenario	1.If a taxi has changed to ‘busy’ status from ‘free’ status, then the system will update the status of this taxi.
Exceptions	
Extensions	
Dependent UC	

5.3 Use Case driver



The user case description for the driver:

5.3.1 Use case “registration”

Use Case ID	TAXI DRIVER1
Name	Registration
Goal	Register to the application to be able to get order
Participating actors	Server and drivers
Precondition	Not registered before
Main scenario	5.A driver who has not registered before register to the system 6.Application accept the registration and send it to server 7.Server confirm the information of the driver 8.Driver become a driver of the system
Exceptions	
Extensions	
Dependent UC	

5.3.2Use case “login”

Use Case ID	TAXI DRIVER2
Name	Log in
Goal	Log into the application to be able to use the system
Participating actors	Server and drivers
Precondition	The drivers has already registered the system
Main scenario	1.A driver clicks the ‘log in’ button to log into the application
Exceptions	
Extensions	
Dependent UC	

5.3.3Use case “inform status”

Use Case ID	TAXI DRIVER3
Name	Inform its status to the system
Goal	To inform if it is available
Participating actors	Server and driver
Precondition	The driver has already login the system
Main scenario	2.A driver clicks the button available 3.The server get the status of the tax. 4.The server get the gps from the taxi and send the taxi to the city zone queue.
Exceptions	
Extensions	
Dependent UC	

5.3.4Use case”check the reservation information”

Use Case ID	TAXI DRIVER4
Name	Check the reservation information
Goal	To check the information of the reservation
Participating actors	Server and driver
Precondition	The driver has already login the system
Main scenario	1.A passenger clicks the button check all the reservation 2.The server will send the information about all the reservation
Exceptions	
Extensions	
Dependent UC	

5.3.5Use case”confirm a request”

Use Case ID	TAXI DRIVER5
Name	Confirm a request
Goal	A driver confirm the request
Participating actors	Server and drivers
Precondition	There is a taxi service request and the driver statue is available
Main scenario	1The system show the order information 2.A driver clicks the ‘confirm request’ button to receipt the order. 3.The system receive the confirm information.
Exceptions	
Extensions	
Dependent UC	

5.3.6Use case”refuse a request”

Use Case ID	TAXI DRIVER6
Name	Refuse a request
Goal	To refuse a request of taxi ride
Participating actors	Server and driver
Precondition	The driver has already login the system There is taxi service request.
Main scenario	1The system show the order information 2.A driver clicks the ‘refuse request’ button to refuse the order. 3.The server will put the taxi to the last position of the queue and send a request to the second driver
Exceptions	
Extensions	
Dependent UC	

5.3.7Use case”confirm a reservation”

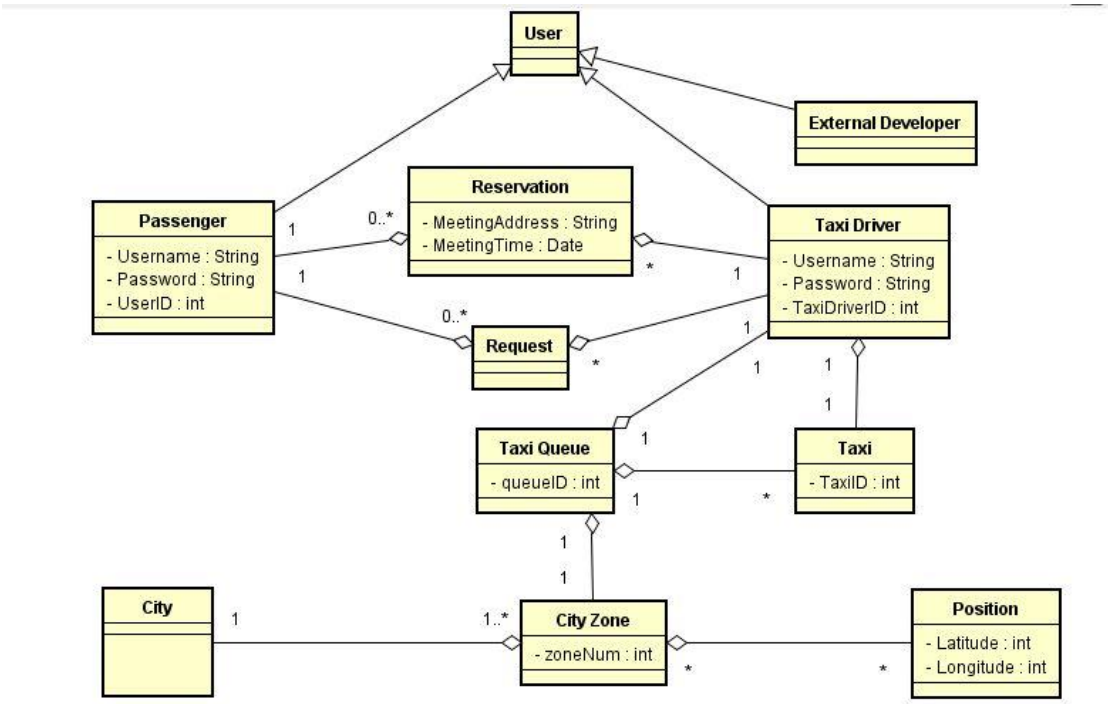
Use Case ID	TAXI DRIVER7
Name	Confirm a reservation
Goal	To confirm a reservation of a certain ride
Participating actors	Server and driver
Precondition	The driver’s status is available
Main scenario	1The system show the reservation information 2.A driver clicks the ‘confirm reservation’ button to receipt the order. 3.The system receive the confirm information.
Exceptions	
Extensions	
Dependent UC	

5.3.8Use case”Refuse a reservation”

Use Case ID	TAXI DRIVER8
Name	Refuse a reservation
Goal	Refuse a reservation
Participating actors	Server and driver
Precondition	The driver’s status is available
Main scenario	1The system show the reservation information 2.A driver clicks the ‘refuse reservation’ button to receipt the order. 3.The system receive the refuse information.
Exceptions	
Extensions	
Dependent UC	

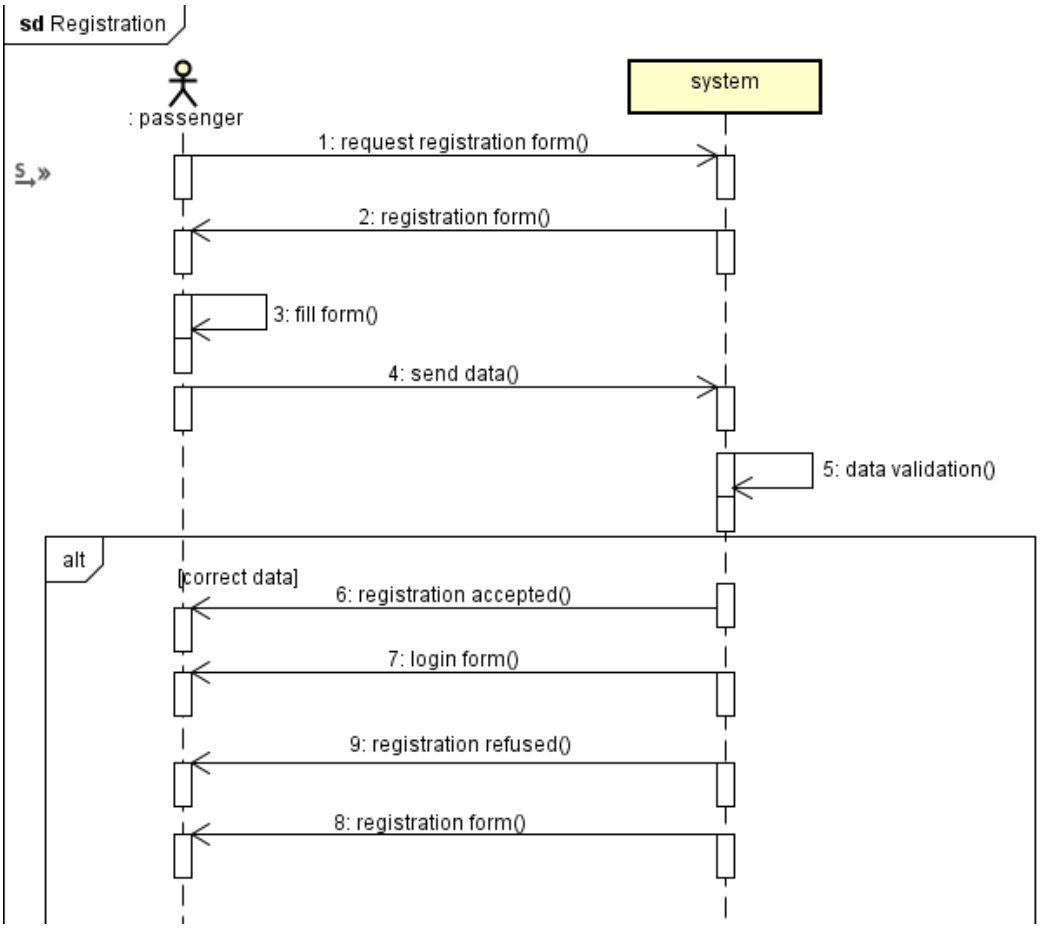
5.4class diagram

The class diagram of this system as follow:

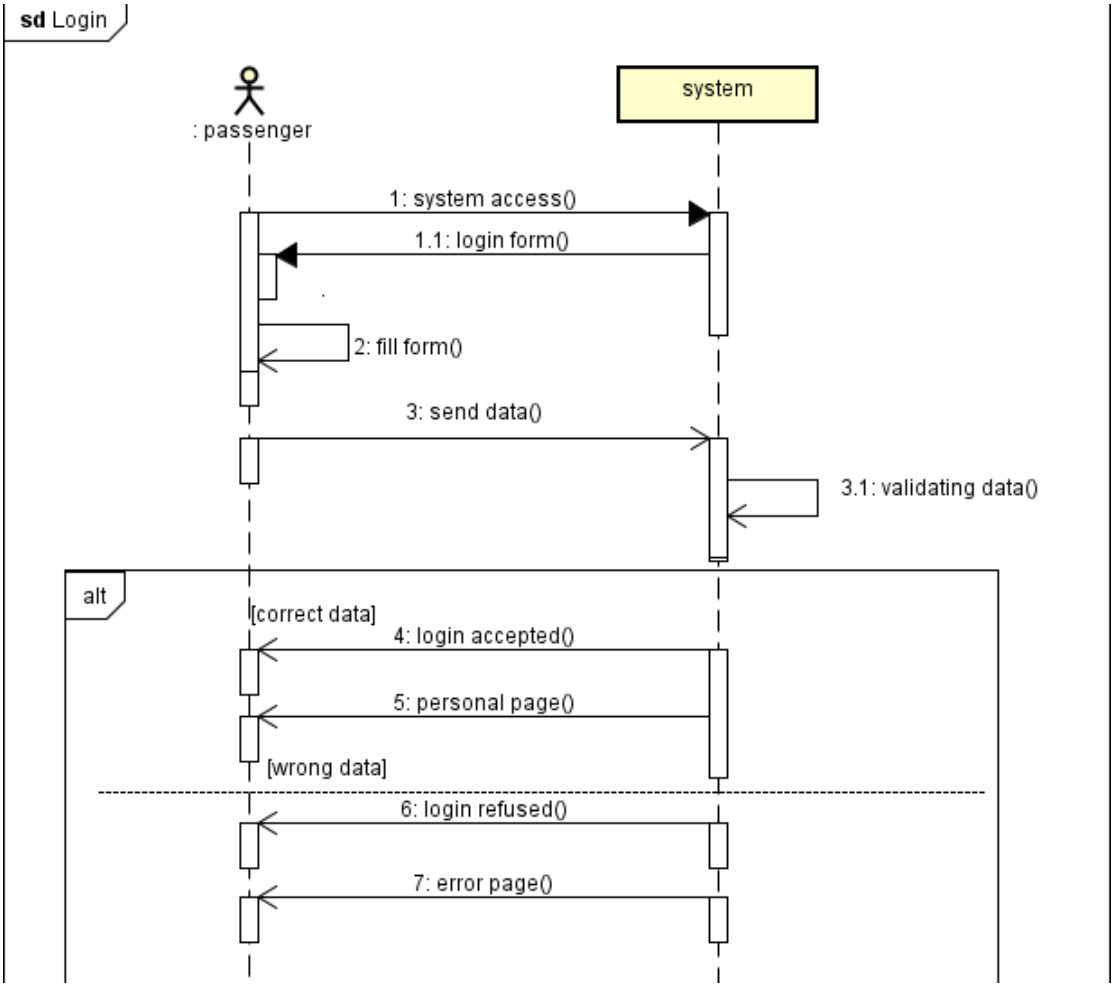


5.5 Sequence diagram

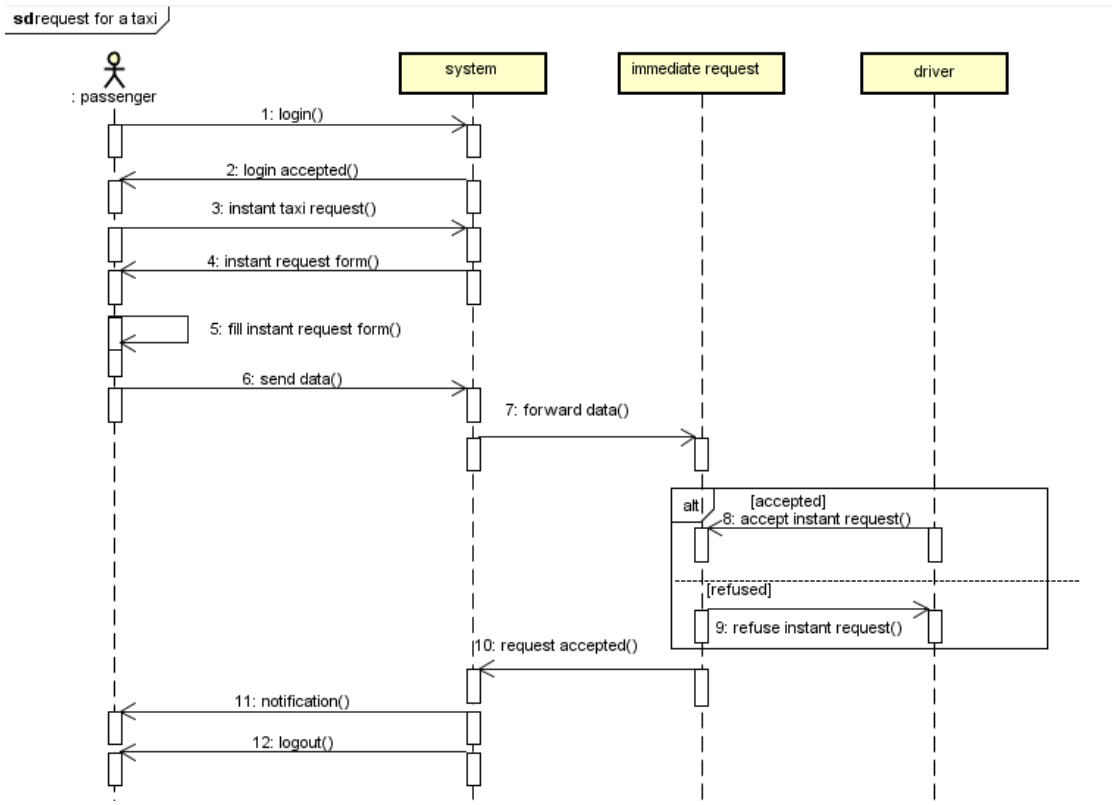
5.5.1 Register



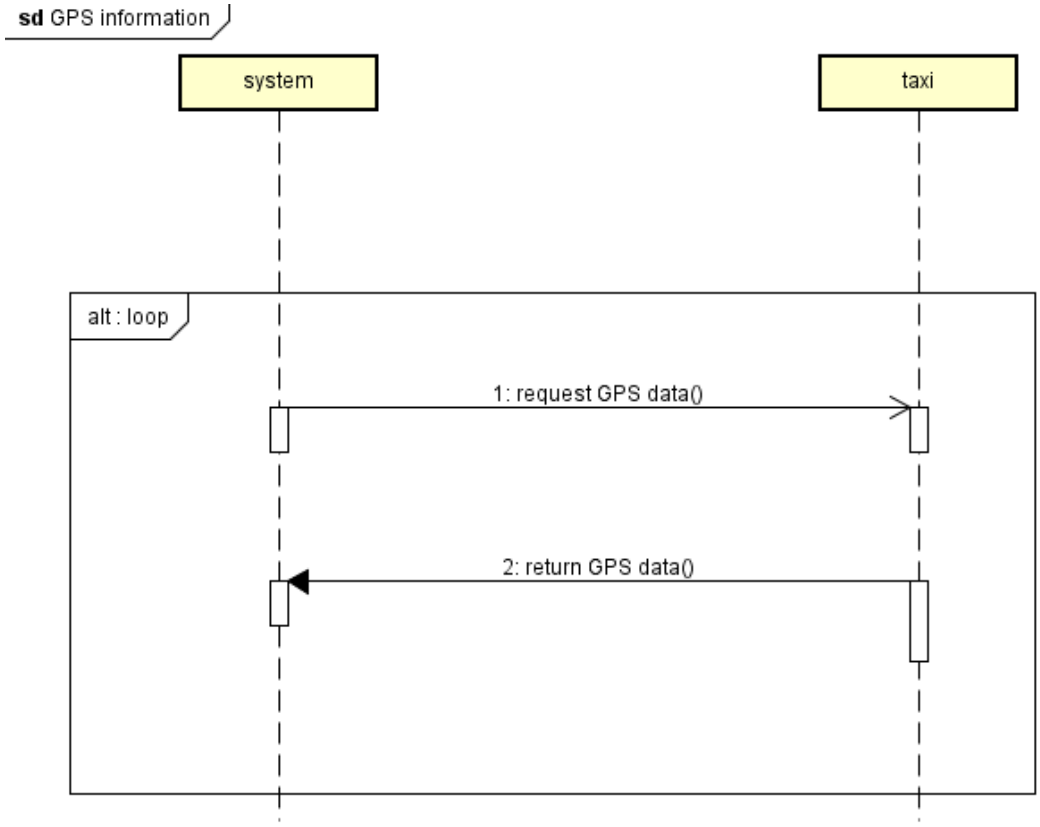
5.5.2 login


sd Login
: passenger
system
1: system access()
1.1: login form()
2: fill form()
3: send data()
3.1: validating data()
alt
[correct data]
4: login accepted()
5: personal page()
[wrong data]
6: login refused()
7: error page()

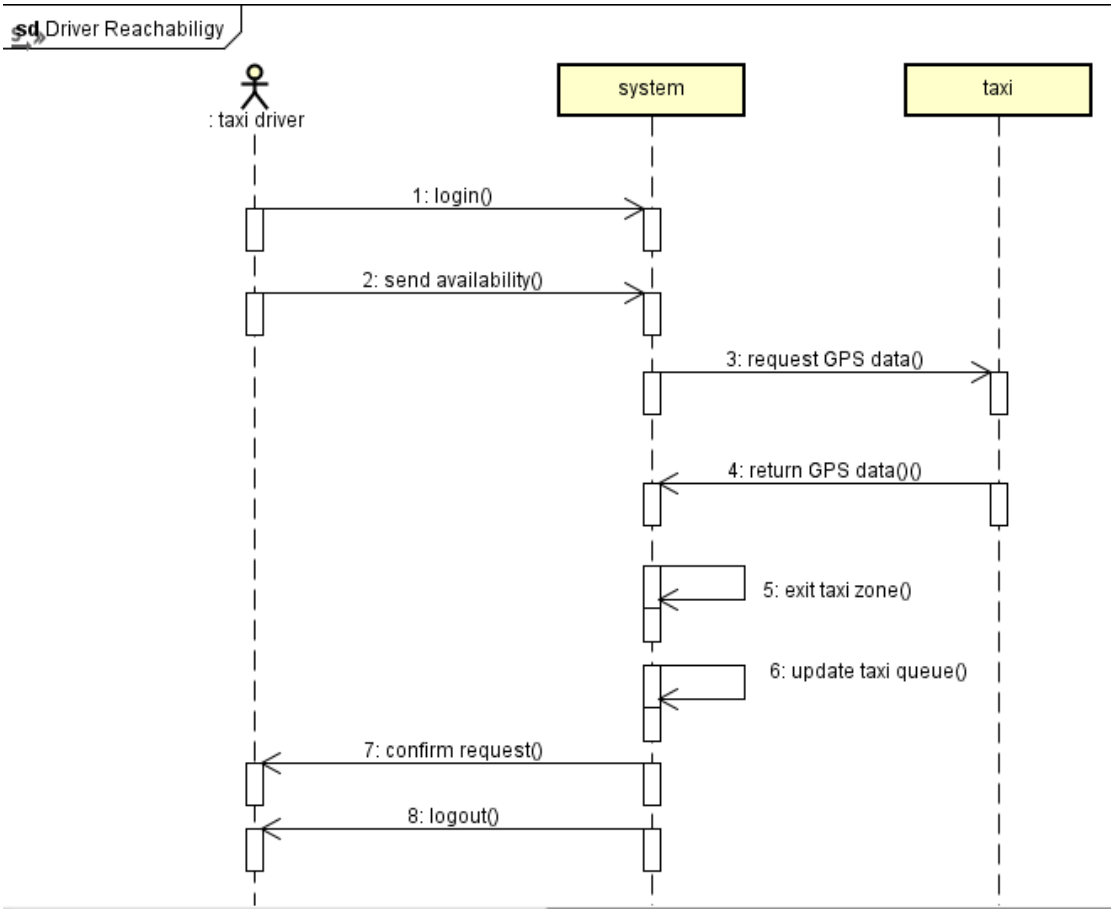
5.5.3 Request a taxi



5.5.4 Gps Information

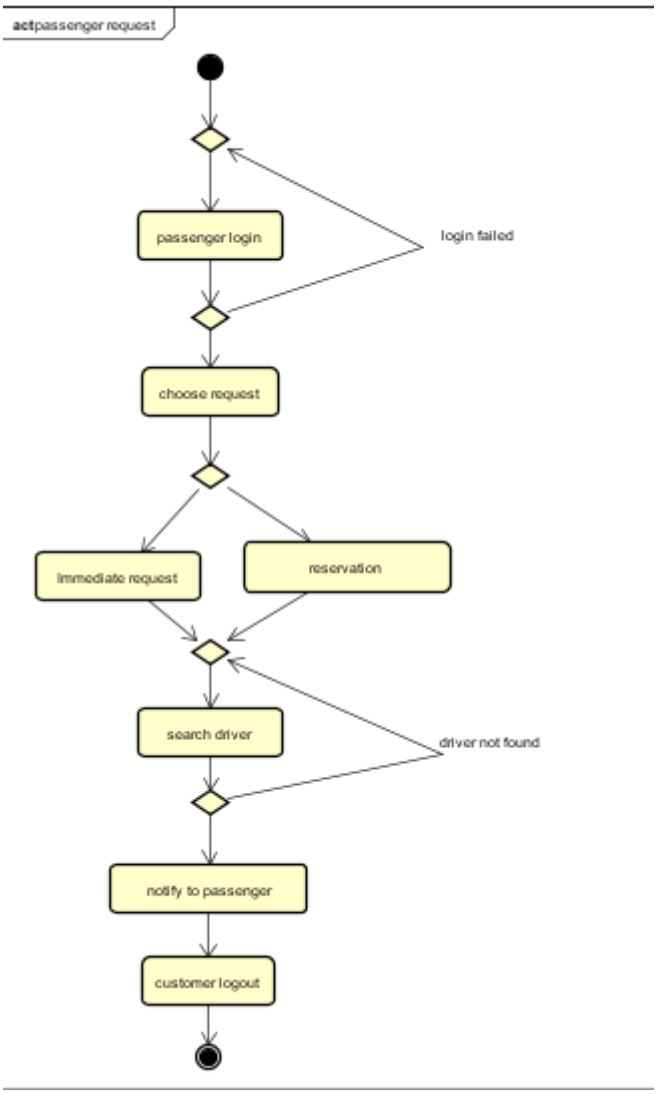


5.5.5 Driver availability

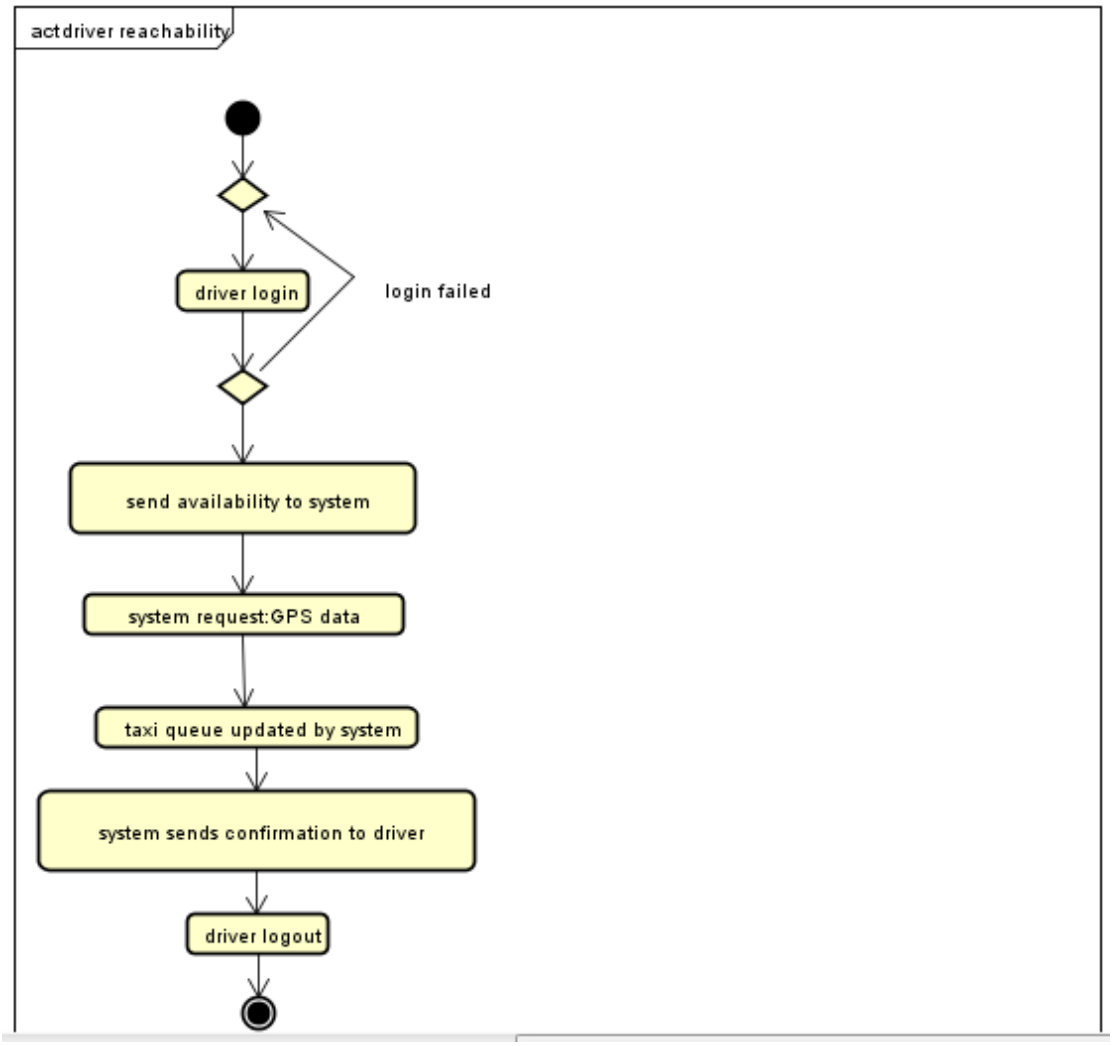


5.6 State chart diagrams

5.6.1 passenger request



5.6.2 driver availability



6. Alloy Modeling

In this paragraph we try to understand if our Class Diagram can be consistent using Alloy Analyzer. We report below the code used and some World generated by our predicates just to let understand that our model is consistent.

```
sig City{
	contains: some CityZone
}

sig CityZone{
	address: some Position,
	uses: one TaxiQueue,
	encloses: some Taxi
}

sig Position{
	locates: set Booking
}

sig Taxi{
	rides: set Booking
}

sig TaxiDriver extends User{
	taxi:one Taxi
}

sig TaxiQueue{
	include: set Taxi
}

sig Passenger extends User{
	booksShort: lone Request,
	booksLong: set Reservation,
}

sig ExternalDeveloper extends User{

}
abstract sig User{

}
abstract sig Booking{

}

sig Request extends Booking{
```

```
}

sig Reservation extends Booking{

}

//*********************************************************
//                          FACTS
//*********************************************************

//ALL ZONES BELONGS TO A CITY
fact CityZonesBelongToCity{
	all z: CityZone, c, c2: City | (z in c.contains and z in c2.contains) implies c=c2
	all z: CityZone | z in City.contains
}


//ALL ADDRESSES BELONG TO A ZONE
fact PositionBelongsToCityZones{
	all a: Position	| a in CityZone.address
}


//ADDRESSES ARE DISJOINT
fact PositionDisjoint{
	all a: Position | one z: CityZone | a in z.address
}


//A QUEUE CAN BE ONLY IN ONE ZONE
fact TaxiQueueOnlyInOneCityZone{
	all q: TaxiQueue | one z:CityZone| q in z.uses
}

//IF A TAXI IS NOT AVAILABLE IN NOT IN A QUEUE
fact TaxiNotAvailable{
	all s: Request, t: Taxi, q: TaxiQueue | (s in t.rides) implies t not in q.include
}

//IN A TAXI IS AVAILABLE IS IN A QUEUE
fact TaxiAvailable{
	all t: Taxi, q: TaxiQueue | (t in q.include) implies (no s: Request	| s in t.rides)
}

//ONLY ONE USER CAN BELONG TO A SHORT RESERVATION
fact ShortReservationForOnlyOneUser{
	all s: Request, u, u2: Passenger | (s in u.booksShort and s in u2.booksShort)
implies u=u2
	all s: Request | s in Passenger.booksShort
```

```
}
//*******************************************
fact oneDriverOneTaxi{
	all d:TaxiDriver|no disj t1,t2:Taxi|(t1 =d.taxi) and (t2 = d.taxi)
	all t:Taxi|no disj d1,d2:TaxiDriver| (t =d1.taxi) and (t = d2.taxi)
}

//ONLY ONE USER CAN BELONG TO A SHORT RESERVATION
fact LongReservationForOnlyOneUser{
	all l: Reservation, u, u2: Passenger | (l in u.booksLong and l in u2.booksLong)
implies u=u2
	all l: Reservation | l in Passenger.booksLong
}

//A SHORT RESERVATIONS IS BOUND TO ONLY ONE TAXI
fact ShortReservationForOnlyOneTaxi{
	all s: Request, t, t2: Taxi | (s in t.rides and s in t2.rides) implies t=t2
	all s: Request | s in Taxi.rides
}

//A TAXI CAN SERVE ONLY A SHORT RESERVATION
fact TaxiRidesOnlyOneShortReservation{
	all s, s2: Request, t: Taxi | (s!=s2 and s in t.rides) implies s2 not in t.rides
}

//A USER EXISTS ONLY IF HAS BOOKED SOMETHING
fact UserIfBookedSomething{
	all u: Passenger | (u in booksShort.Request) or    (u in booksLong.Reservation)
}

//A TAXI CAN BE ONLY IN ONE QUEUE
fact TaxisOnlyInOneTaxiQueue{
	all t: Taxi, q , q1: TaxiQueue | (t in q.include and t in q1.include) implies q=q1
}
  //A TAXI CAN BE IN ONLY ONE ZONE
fact TaxiOnlyOneCityZone{
	all t: Taxi, z,z2: CityZone | (t in z.encloses and t in z2.encloses) implies z=z2
	all t: Taxi | t in CityZone.encloses
}

//TO ONE BOOKING CORRESPONDS ONLY ONE ADDRESS
fact BookingOnlyOnePosition{
	all b: Booking, a, a2: Position | ( b in a.locates and b in a2.locates) implies a=a2
	all b: Booking | b in Position.locates
}

//A TAXI THAT IS IN A QUEUE IS IN THE ZONE RELATED TO THE QUEUE
fact CityZoneTaxiTaxiQueueAreRelate{
	all t: Taxi, q: TaxiQueue, z: CityZone | (t in q.include and q in z.uses) implies (t
in z.encloses)
```

```
}

//NO TAXI IS BOUND TO A LONG RESERVATION
fact LongReservationsHaveNoTaxi{
	all l: Reservation, t: Taxi | l not in t.rides
}


//*********************************************************
//                          ASSERTS
//*********************************************************

//THIS ASSERTION VERIFIES THAT AN ADDRESS BELONGS TO ONLY ONE
ZONE
assert PositionInOnlyOneCityZone{
	all a: Position, z, z2: CityZone | (a in z.address and a in z2.address) implies z=z2
}

//THIS ASSERTION VERIFIES THAT A BOOKING HAS ONLY ONE ADDRESS
assert BookingOnlyOnePosition{
	all a, a2: Position | no b: Booking | (b in a.locates and b in a2.locates and a!=a2)
}

//THIS ASSERTION VERIFIES THAT A CITY MUST EXIST IF THERE ARE
ZONES, TAXIS AND QUEUES
assert ExistsCity{
	all z: CityZone, t: Taxi, q: TaxiQueue | ((z in City.contains) implies some City)
and ((t in q.include) implies some TaxiQueue)
}

//THIS ASSERTION VERIFIES THAT A USER HAS BOOKED SOMETHING
assert    UsersHaveBookedSomething{
	all u: Passenger | (u in booksShort.Request) or    (u in booksLong.Reservation)
}

//THIS ASSERTION VERIFIES THAT A TAXI BELONGS TO ONLY ONE
QUEUE
assert TaxiInOnlyOneTaxiQueue{
	all q , q1: TaxiQueue | no t: Taxi | (t in q.include and t in q1.include and q!=q1)
}

//THIS ASSERTION VERIFIES THAT A TAXI IS LOCATED TO ONLY ONE
ZONE
assert TaxiInOnlyOneCityZone{
	all z, z2: CityZone | no t: Taxi | (t in z.encloses and t in z2.encloses and z!=z2)
}

//THIS ASSERTION VERIFIES THAT A SHORT RESERVATION IS DONE BY
ONLY ONE TAXI
assert ShortReservationForOnlyOneTaxi{
```

```
        all t, t2: Taxi | no s: Request | (s in t.rides and s in t2.rides and t!=t2)
}

//THIS ASSERTION VERIFIES THAT A LONG RESERVATION IS BOOKES BY ONLY ONE USER
assert LongReservationForOnlyOneUser{
        all u, u2: Passenger | no l: Reservation | (l in u.booksLong and l in u2.booksLong and u!=u2)
}

//THIS ASSERTION VERIFIES THAT A SHORT RESERVATION IS BOOKES BY ONLY ONE USER
assert ShortReservationForOnlyOneUser{
        all u, u2: Passenger | no s: Request | (s in u.booksShort and s in u2.booksShort and u!=u2)
}

pred show(){
#Request > 1
#Reservation > 1
#City=1
#CityZone = 3
#TaxiDriver=3
#Taxi=3
}


run show for 5
```