Requirements Analysis and Specification Document

Matteo Greselin , Seyeed Mohsen Kashfi Haghighi

Contents

1.Introduction

1.1 Purpose

THE PURPOSE OF this document is, providing requirements and specifications of Taxi service solution called 'myTaxiService'. This solution aims at optimizing taxi service through mobile and web application by providing fair management of taxi queues, simplification of passengers access to this service. The app recieves Request/Reservation from passengers and selects drivers according to their positions in the queue of corresponding zone and let drivers accept or refuse the service. The system also provides Application Programming Interface (API) to enable further developments.

1.2 Scope - Goals

OUR MAIN GOALS are:

- [G1]Simplify the access of passengers to the system
- [G2]Guarantee a fair management of taxi queues
- [G3]Minimize the waiting time of passenger
- [G4]Maximize the usage of drivers(Minimize the idle time of available drivers)
- [G5]User-friendly App with efficient access to DB and efficient loads of servers

1.3 Proposed System

WE PROPOSE a solution for mobile and web platform. There are different characteristics which will be applied to each of these two platforms. Users are divided into four different categories:

- Guest: will be able to access the public pages (home page, about us, contact us, help, register).
- Passenger: they must login into system then they will be allowed to Request/Reserve a taxi.
- Drivers: They must login into system then they will receive Request/Reservation after that they can accept or refuse it.

- Admin: he/she must login into system, then can monitor all activities in the system.
- all users except guests can manage their profile.

1.4 Glossary

1.4.1 Definitions

User	: Every person that uses our system.
Passenger	: is a registered user that can request or reserve a taxi.
TaxiDriver	: is a registred user that can accept or refuse a ride.
Ride	: Is an event starts by a request/reserve form passenger and continues with accept from drive.
Request	: An action from a passenger for requesting a taxi to go to a specific.
Reservation	: An action from a passenger for reserving a taxi to go somewhere.
System	: In this document we use system to indicate our application.
Availibility	: Is a situation in which indsicates either a taxi driver is available for next service or no.

1.4.2 Acronyms

RASD	: Requirements Analysis and Specification Document.
DB	: DataBase. DBMS: DataBase management system.
API	: Application Programming Interface.
OS	: Operating System.

1.4.3Abbreviations

[Rn]	: n-functional requirement.
[Dn]	: n-domain assumption.
[Gn]	: n-goal.
[Sn]	:n-specification

1.2 Assumption

With respect to our specification document, there are some assumption to do in order to improve comprehensibility:

- [D1] We assume that every users have a internet connection;
- [D2] We assume that the city is divided into 4 zones;
- [D3] We assume that taxi drivers can set "unavailibility";
- [D4] We assume that canceling reservation is impossible after driver allocation;
- [D5] We assume that canceling Reservation/Request is admitted
- [D6] We assume that Taxi Drivers has prefixed workshift, in order to always guarantee the presence of some Taxi
- [D7] The taxi driver must be able to reach the meeting point within 15 minutes at most.
- [D8] Each taxi driver must be the owner and use one taxi.

1.3 Stakeholders

Our stakeholders are :

- 1) Prof. Di Nitto the professor of the course software Eng 2 as product owner.
- 2) target audiences including Passengers and Drivers.
- 3) Any developer wants to work on our App for further developement.

1.4 References

Specification Document: Assignments 1 and 2 (RASD and DD)

IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. (Revision of IEEE Std 830-1993)

2. Actors Identifying

1. Guest: a person who hasn't registered and can only exploit basic functionalities such as looking for help.
2. Passenger: a registered user who will be allowed to request/Reserve a taxi.
3. Taxi Drivers: a registered user who can accept/refuse a ride.
4. Administrator: a registered user in which can monitor all the events happen in the system.

3. Requirements

3.1 Functional Requirements

3.1.1 Sign up

- Registration of new user:
    - [R1] passenger must not already registered
    - [R2] Passenger information must be included such as: fistname, lastname, tell, address, email, username and password is required.
    - [R3] Driver must not already registered.
    - [R4] Driver information must be included like: fistname, lastname, tell, address, email, username and password is required.
    - [R5] Driving lisence number, taxi plate number is required.
    - [R6] The system must send an e-mail to the address specified by the user for confirmation.
    - [R7] The system must ask for the password two times.
    - [R8] Both passwords must be identical.

3.1.2 Log In

- [R1] Users must enter valid username and password.
- [R2] Password reset must be included in order to help the user renew his/her password.

3.1.1 Request

- [R1] The system must specify the location of the passenger and check that she is in the zone or not.
- [R2] Passenger must be already logged in.
- [R3] Request can be done only by passengers.
- [R4] Passenger must specify start location and destination location.
- [R5] Passenger cannot create a request if she has already a pending request.
- [R6] After request acceptation, the position, plate number and arrival time of the taxi driver will send as a notification to passenger.

3.1.4 Reservation

- [R1] Reserve can be done only by passengers
- [R2] Passenger must be already logged in
- [R3] Passenger must specify the time of reservation as well as Start location and destination.
- [R4] Passenger must not be allowed to cancel the reservation after allocation
- [R5] The system must send an acknowledgement to the passenger in order to reservation confirmation.
- [R6] The system lets the passenger to enter valid information.

3.1.5 Accept/Refuse

- [R1] Accept/refuse can be done only by taxi drivers.
- [R1] Taxi Driver must inform the system about accepting or refusing in 5 minutes.
- [R3] Taxi Driver must not be allowed to cancel the acception after allocation.
- [R4] Taxi Driver must not be allowed to log out after allocation.
- [R5] The system must not send a new request/reservation to the taxi driver which hasn't finished her ride yet.

3.1.6 Queue Management

- [R1] Taxi Driver must be allowed to set unavailability option whenever she wants.
- [R2] Taxi Driver must be in the end of the queue when she set the availability option on.
- [R3] Taxi Driver must not be allowed in the queue until setting the availability option on.
- [R4] The system must add taxi driver into the longest passenger queue

3.1.7 Administrator Functionalities

- [R1] Administrator can send notification to each user.
- [R2] Administrator can see monthly reports about service usage.
- [R3] Adrministrator must block or delete accounts of users that not respect our service policies.
- [R4] Administrator must not be allowed to create request or reservation with his special account.

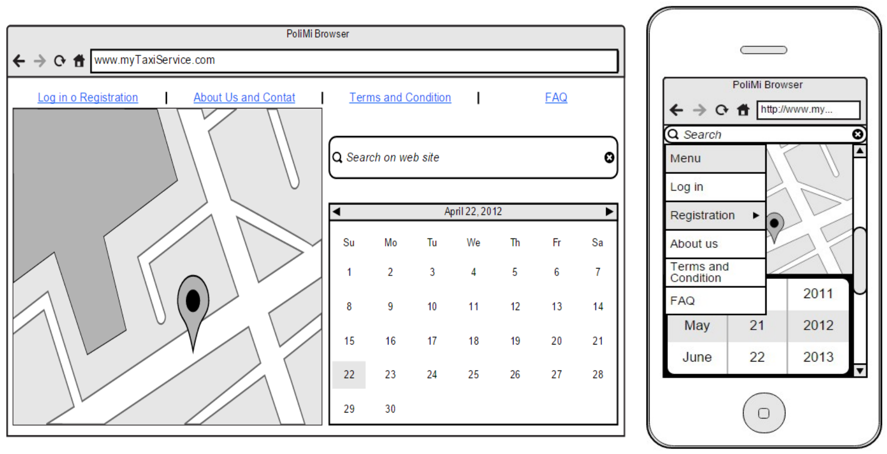
3.1.8 Profile Management

- [R1] User must be already logged into the system in order to edit her profile.
- [R2] Taxi Driver must not be allowed to change her licence number
- [R3] Taxi Driver must be allowed to change her car number only with special request.
- [R4] User will be allowed to edit her personal information such as Home address or telephone num.

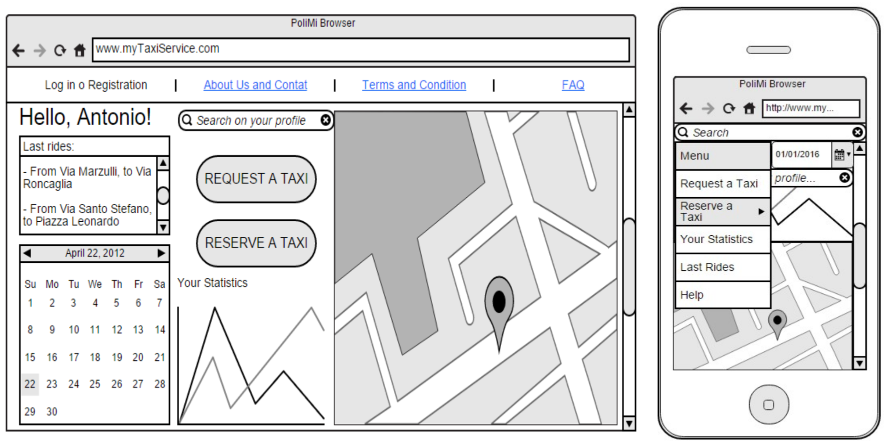
3.2 Non Functional Requirments

3.2.1 User Interface

THE INTERFACE of our application is expected to be used via web or mobile application. There are two main pages, one is a public page and another one is personal page consist of the details of the user. In the public page, on the top we could find 4 links that allow to navigate into the web site and below them we have a google maps, a calendar and a search bar.



• After log in, in the personal page, every user could find any information about riding, for example there will be two buttons for request and reserve a taxi, a calendar on the left, above that there will be a description of last rides. Obviously on the right every user could find a google map in order to search for the destination. Finally on the bottom of the page there will be a statistic graph that indicate something like number of rides in different months.



3.2.2 User Characteristics

MyTaxiService® is a service reserved to people that are more than 18 years old. People who are less than 18 years old must be accompanied by their parents. myTaxyService® is prepared also for people with disability but they should inform the system to take them a taxi with necessary equipment.

3.2.3 Constraints

In this part we want to consider some particular constrains, not all trivial, that binding the use of our application by user with respect to rules of conduct, obligations and prohibitions.

- In every country, taxi service must follow regulatory policies. Our app must provide to follow these policies and we must be ready to updates them in order to eventual expansion of our used domain.

- Our application must be able to interfaces with a map service (like google maps), with the GPS, with internet and with every OS (iOS, Android, Windows, Linux). In order to this we will not need to use too much restrictive process, for example HD graphic could produce problem in small smartphone.

- A critical issue for the app will be days with strikes. We expect that in those periods the system will receives anomalous number of request and we will need to stop it when will reach a previously defined maximum waiting time.
- In order to avoid duplicate requests/reservation, we will consider a timeout to complete it. After it every data insert until that moment will be delete. If during a request or a reservation should miss internet connection, when the device will reconnect to system the user will move on his home page.
- In order to preserve the privacy of our user we must provide to introduce encryption methods in the main databases.

3.2.4 Security

IN THIS PART we want to ensure that there is a high protected area in order to avoid any fraud.

1. The communications between server and clients must be completely secure and protected by strong encryption such as SSL protocol.
2. Users password must not stored in plain text in the database, they must be hashed and encrypted.
3. If a user try more than 4th times for log in, block the IP address for 10 minutes.

3.2.5 Availability

1. Availabilty is one of the most important factor that a system must notice of it, so our system must provide a high availability situation even upto 98%.

3.2.6 Documentation

DURING this project we should do several phases:

1. Project Plan : to define our system, goals and how we can obtain them
2. 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
3. DD: Design Document, to define how we should implement our system.
4. Testing Document: it's a report of our system testing

4.Specifications

IN THIS section we want to specify some details about how we will reach our main goals.

- [S1] Only a registered user can request or reserve a taxi;
- [S2] Administrator can remove or block a passenger account, if he recives report about misbehavior;
- [S3] When a ride is done, GPS signals detects that the taxi driver which has been assigned a ride has really reached the destination indicated in the request or in the reservation;
- [S4] If a passeger remove her registration, every pending request or pending reservation will be delete;
- [S5] Every user should see only her personal page with his statistics, she can not see statistics of other user according to privacy policies
- [S6] Each taxi drivers has to be accepted by company at first, After that the company will register the taxi driver to the service

5.Scenarios Identifying

Now we are going to propose some real example about how our application should be used in real world.

- Antonio need a taxi to go home after work. He usually use public transportaion but today there is a strike in the city. He search throught the internet and found the taxi service of the city. So he sees the description of the service and notes that he has to register in the website. He decide to register as a passenger and after inserting the requirement information he received a code that he can use it for taking or reserving a taxi. Therefore he requests a taxi and the system will inform a taxi driver who is in the first position of that zone queue. After that a taxi driver will decide to accept a request or not, if taxi driver accept the system notify Antonio the waiting time and taxi code. Otherwise system will inform the next taxi driver in the queue.

- Antonio is a registered user yet. He is at work now and suddenly he remembers that he has to go to his mother house for dinner. So he wants to reserve a taxi for 8 o'clock but the reservation should be done 2 hours before the ride. At 6:30 he open his app and select the reservation process. The system will ask him to insert the currently and destination location, and also the time of departure. If Antonio wants to cancel his reservation he has to do it until 15min before riding time. When the system save the reservation, confirm it to Antonio. Finally at 7:50 the system will assign the reservation to a taxi driver.

- Francesco is a taxi driver of Milan register in our App. He is at work and waiting for passenger when he sees a notification on his device that is the request from a passenger for a ride, then he will decide to accept the request or not. So he decides to refuse it, In this case the passenger is too far. Francesco will go to the end of the queue, the system will try to ask to Alex that is the next taxi driver in the queue and the request is accepted.

- Francesco's ride is finished. For coming back to the queue he needs to inform the system that he is available so the system will add him to the biggest queue. Suddenly an urgent problem is occured and he has to leave the queue. So he set the Unavailibility option to inform the system that he wants to leave. Finally the system will remove him from the queue then if he wants to come back to the queue he should again set the availibility option.

- Our system needs to be updated and fixed some brief problems. For this reason the Administrator needs to inform their users that in the next week the system will not work for 3 hours. So the information will send on the first page of every users.

- We decide to extend our system by adding a feature like taxi sharing. Marco, our programmer, will be able to modify the application and develop it. But does our privileges are enough to restrict him not to access to the whole data of the system and he can only work on specific field.

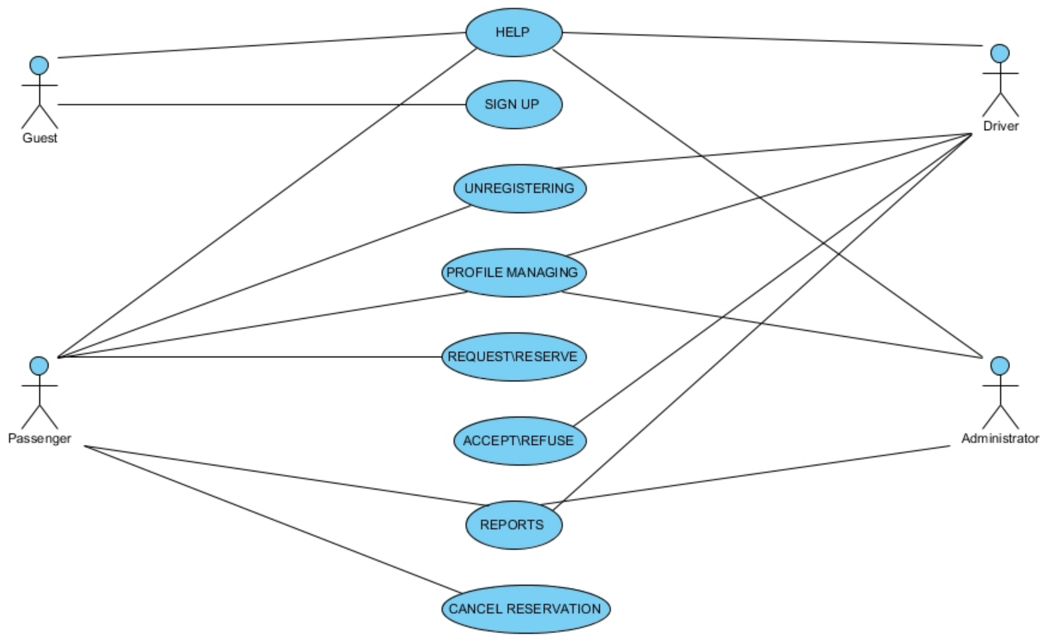
6. UML Models

In these parts we want represent UML models from the scenarios that we mentioned in the previous paragraph. We propose a Use Case and a Sequence Diagram for each of the next models:

- Main Scenario
- Sign Up
- Login
- Unregister
- Profile Managing
- Request/Reserve or Cancel Reservation
- Cancel Reservation
- Accept/Refuse
- Availibility/Unavailibility
- Administrator Functionalities
- Help, Reports and Statistics

6.1 Use Case And Sequence Diagram

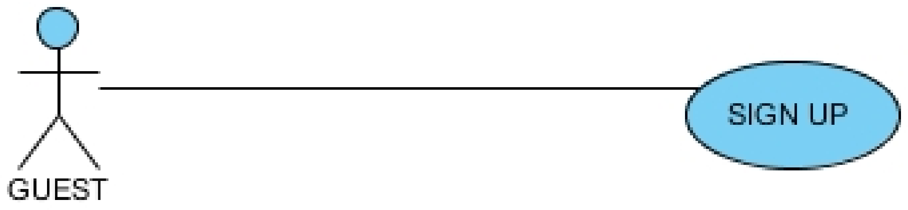
6.1.1 Main Scenario

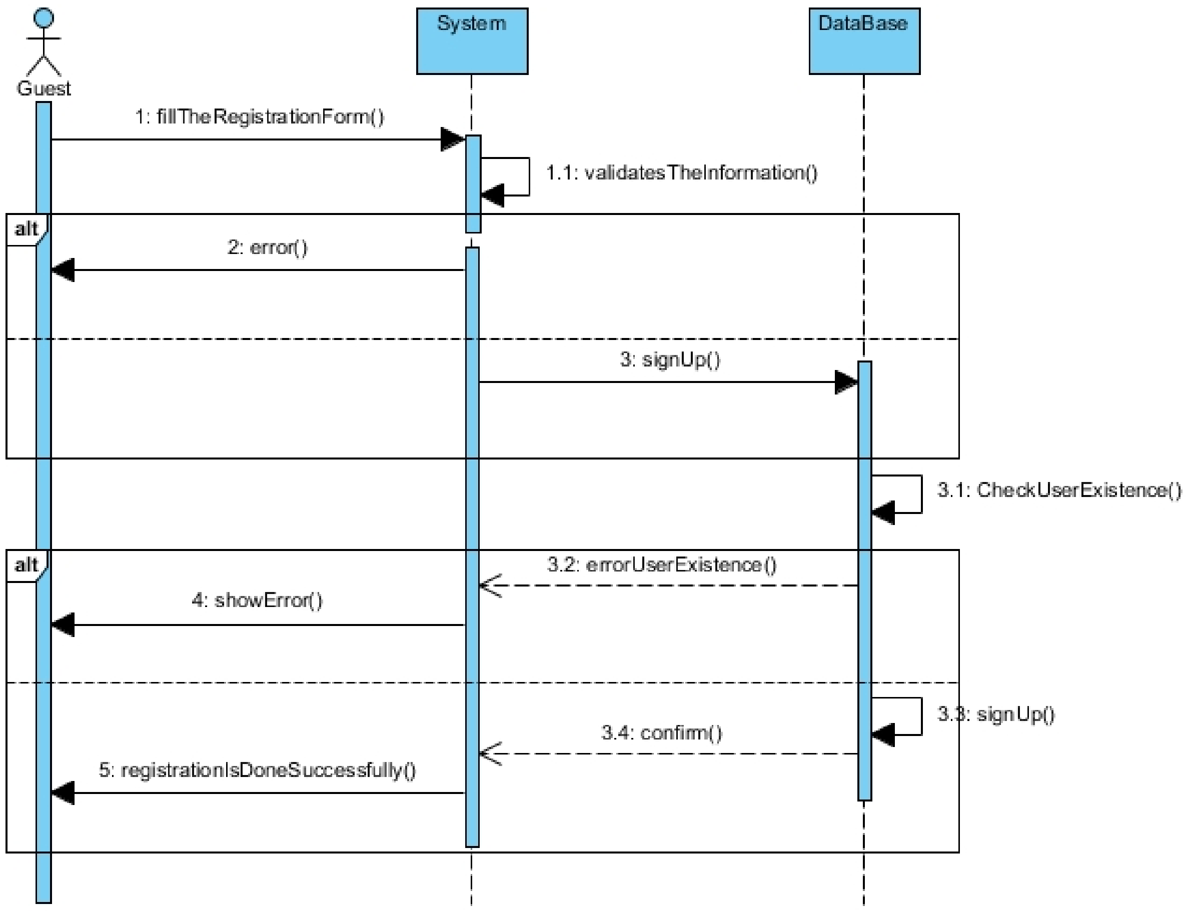


6.1.2 Sign Up

- **Reached Goals**: [G1]
- **Actors** : Guest
- **Starting Situation** :
  - A Guest see the home page in order to use our service.
- **Flows Event:**
  - Guest wants to register in our system.
  - System open a registration form.
  - Guest has to fill at least the mandatories field.
  - confirm the registration.
  - the application will save on DB.
- **Final Situation** :
  - registration has successfully done and myTaxiService® has a new user.
- **Exception** :

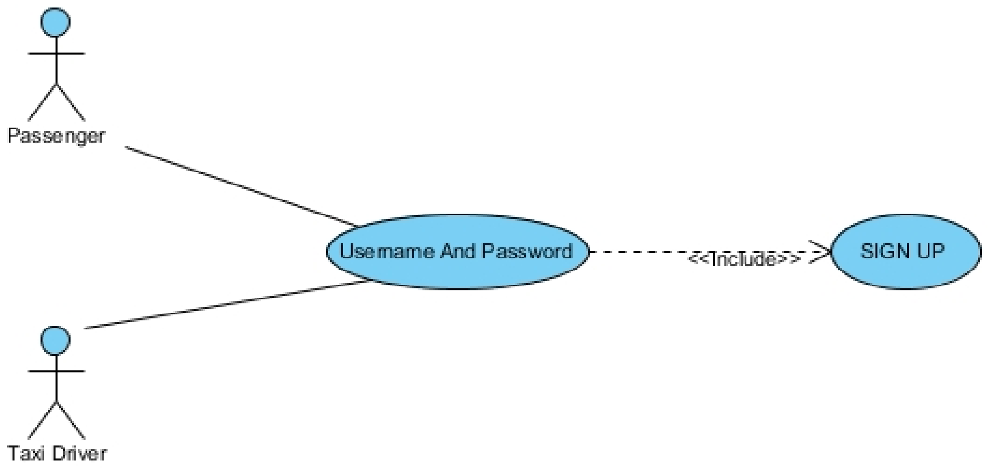
1. The username has already used by another user.
2. Some fields did not fill correctly.
3. The password did not meet the required complexity.
4. The connection between system and DB is failed.

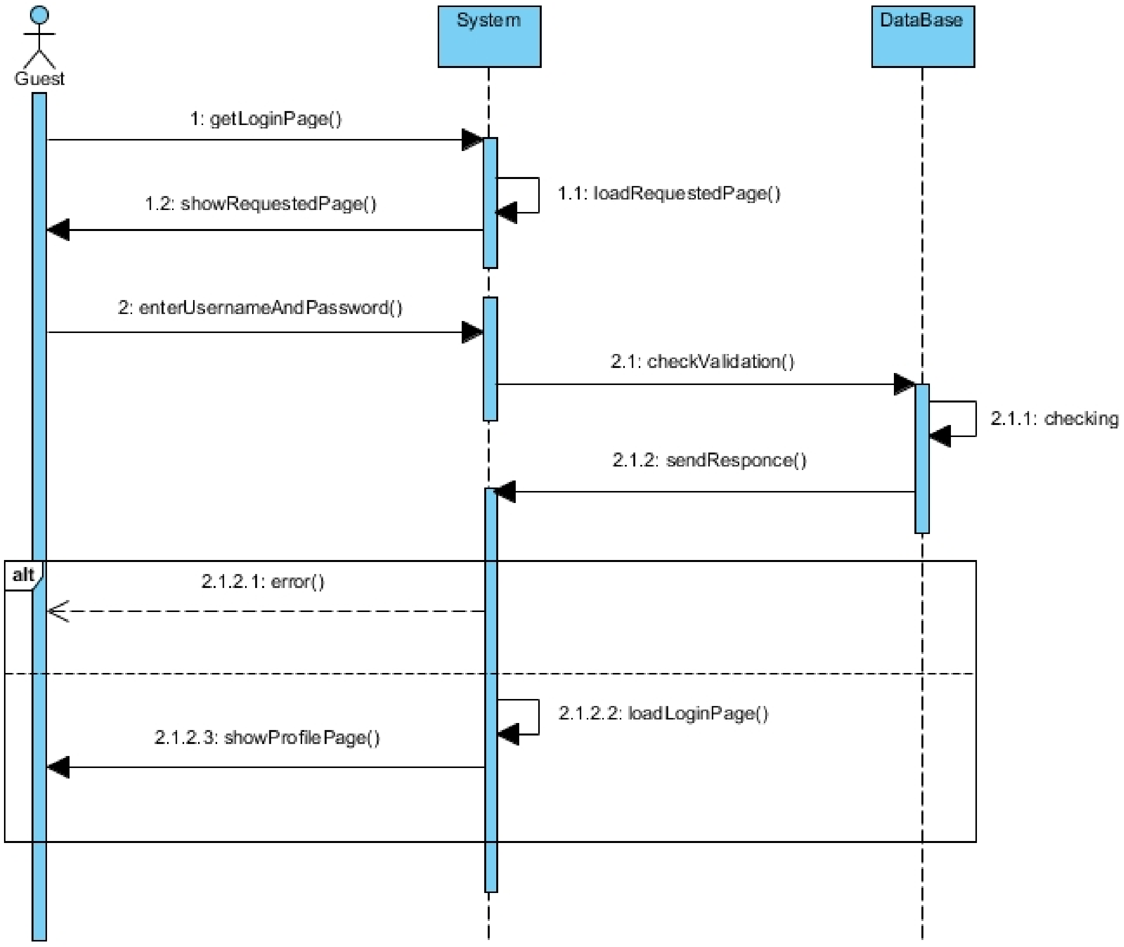



Guest
System
DataBase
1: fillTheRegistrationForm()
1.1: validatesTheInformation()
alt
2: error()
3: signUp()
3.1: CheckUserExistence()
alt
3.2: errorUserExistence()
4: showError()
3.3: signUp()
3.4: confirm()
5: registrationIsDoneSuccessfully()

6.1.3 Login

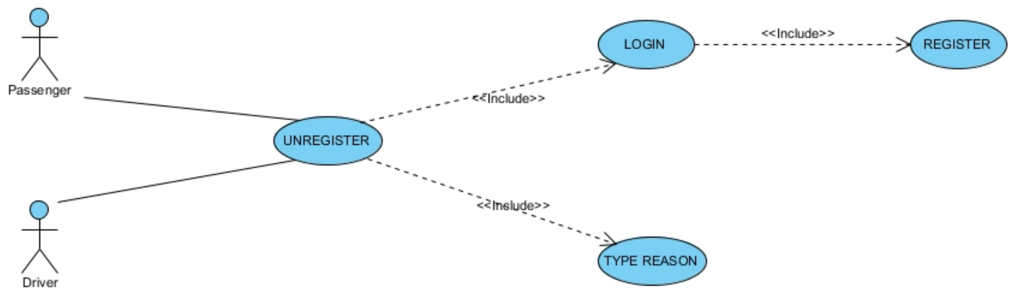
- **Reached Goals**: [G1]
- **Actors** : Passenger, Adminstrator, Taxi Driver
- **Starting Situation** :
    - Users are already registered in the system
- **Flows Event:**
    - The system shows the Login form
    - Guest enters Username and Password
    - Guest confirm the form.
    - The system opens the profile page of user.
- **Final Situation** :
    - A guest is promoted to a User and having a profile page.
- **Exception** :
    1. An error message will be shown if username and password are wrong.
    2. The connection between system and DB is failed.

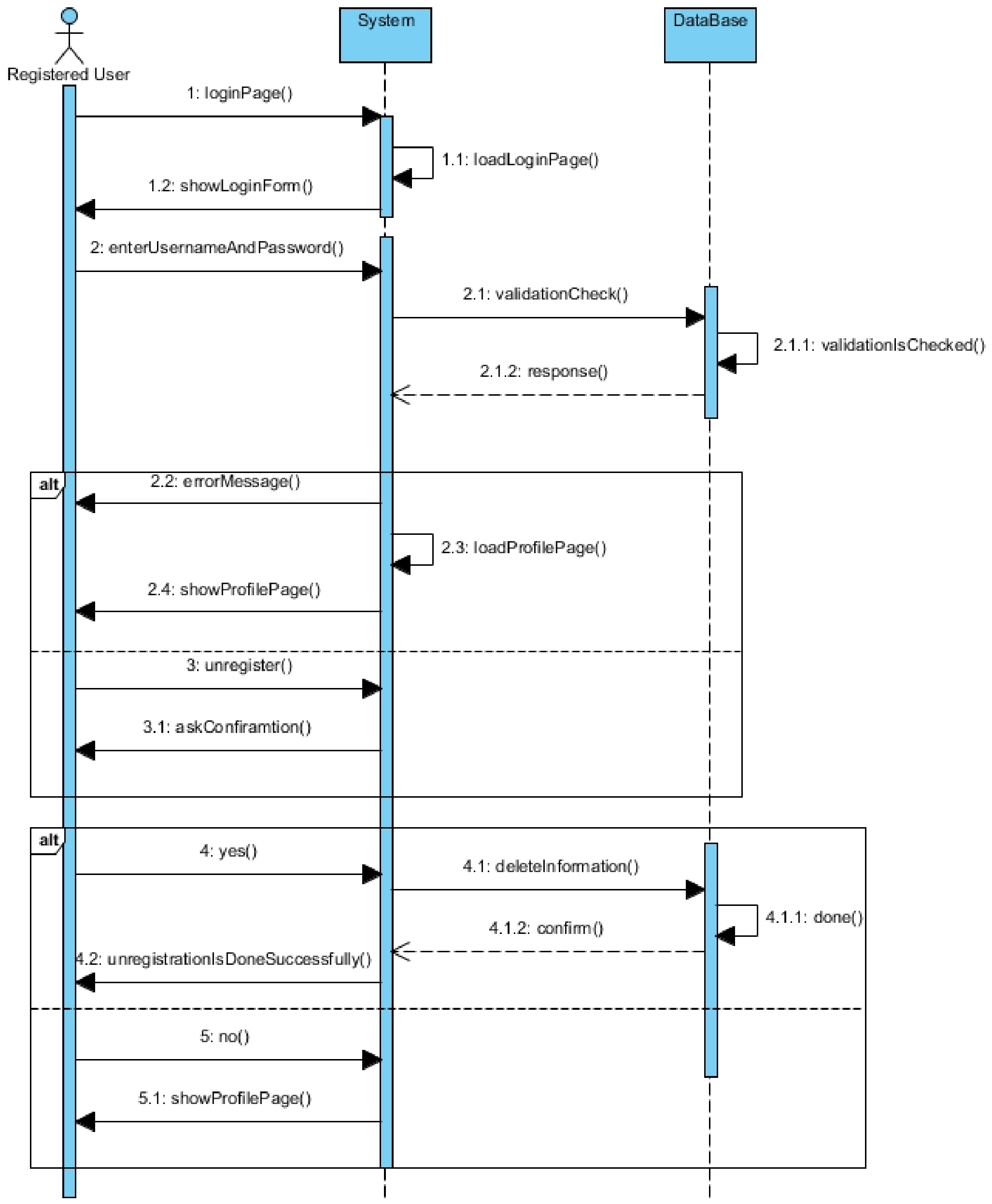



Guest
System
DataBase
1: getLoginPage()
1.1: loadRequestedPage()
1.2: showRequestedPage()
2: enterUsernameAndPassword()
2.1: checkValidation()
2.1.1: checking
2.1.2: sendResponce()
alt
2.1.2.1: error()
2.1.2.2: loadLoginPage()
2.1.2.3: showProfilePage()

6.1.4 Unregister

- **Reached Goals**: [G1]
- **Actors** : Passenger, Taxi Driver
- **Starting Situation** :
  - The user is logged in and can see profile page
- **Flows Event:**
  - A user wants to unregister
  - The system asks for confirmation.
  - The user confirm.
  - The system deletes the information of that user form DB.
- **Final Situation** :
  - The user is demoted to a Guest and can not access to profile page.
- **Exception** :
  1. The user regret to unregister.
  2. The connection between system and DB is failed.

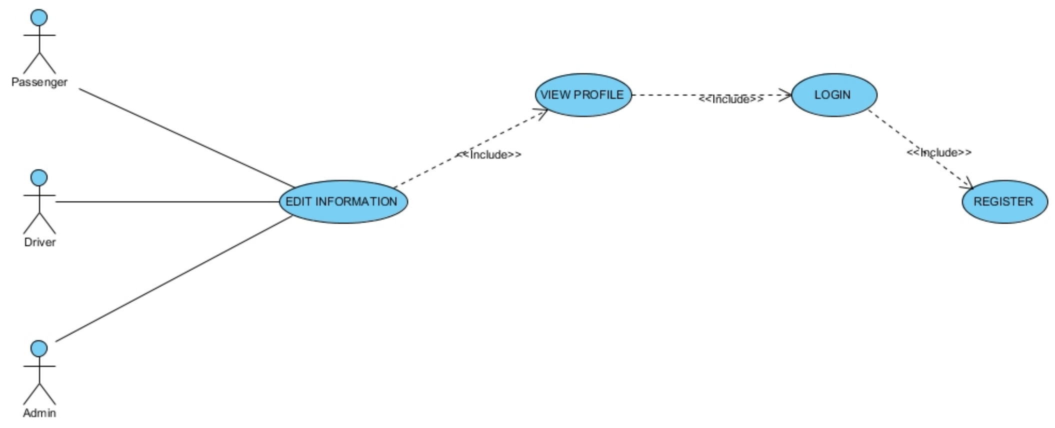


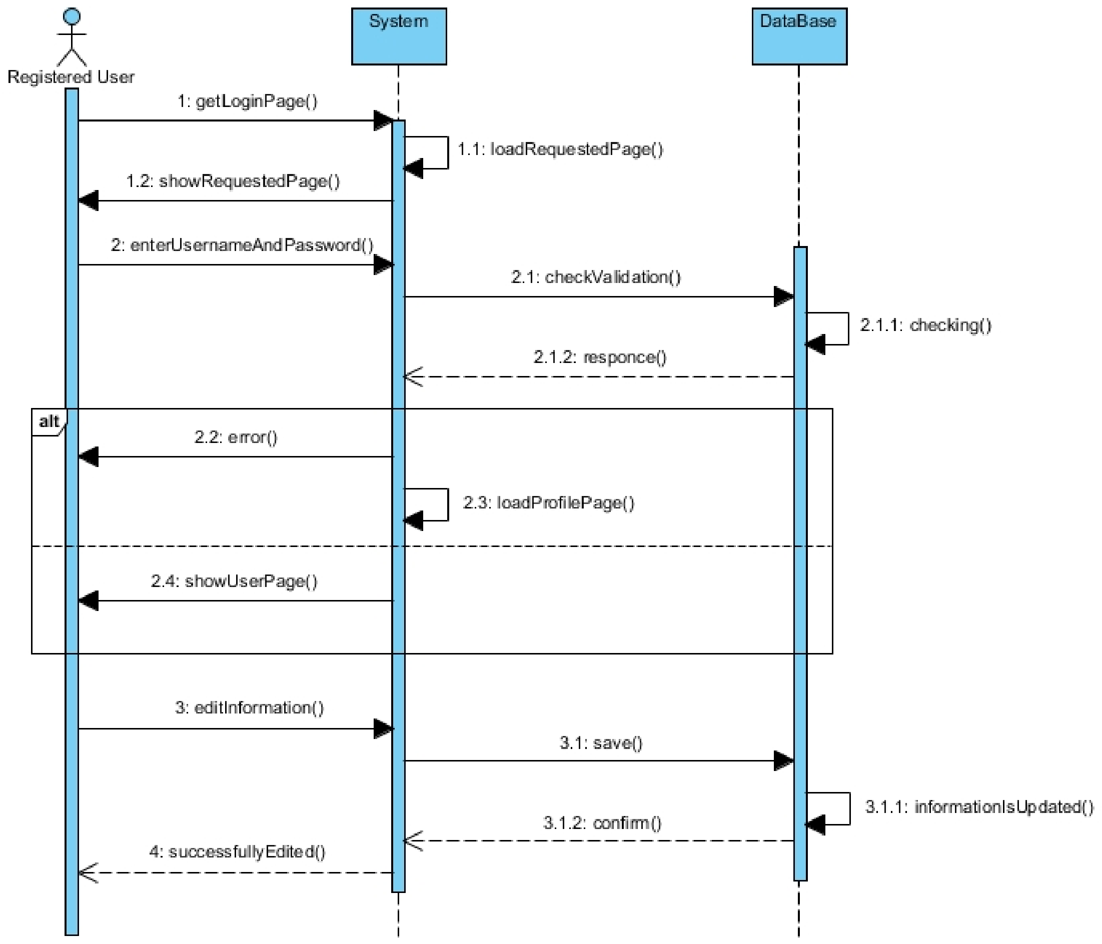


Registered User
System
DataBase
1: loginPage()
1.1: loadLoginPage()
1.2: showLoginForm()
2: enterUsernameAndPassword()
2.1: validationCheck()
2.1.1: validationIsChecked()
2.1.2: response()
alt
2.2: errorMessage()
2.3: loadProfilePage()
2.4: showProfilePage()
3: unregister()
3.1: askConfiramtion()
alt
4: yes()
4.1: deleteInformation()
4.1.1: done()
4.1.2: confirm()
4.2: unregistrationIsDoneSuccessfully()
5: no()
5.1: showProfilePage()

6.1.5 Profile Managing

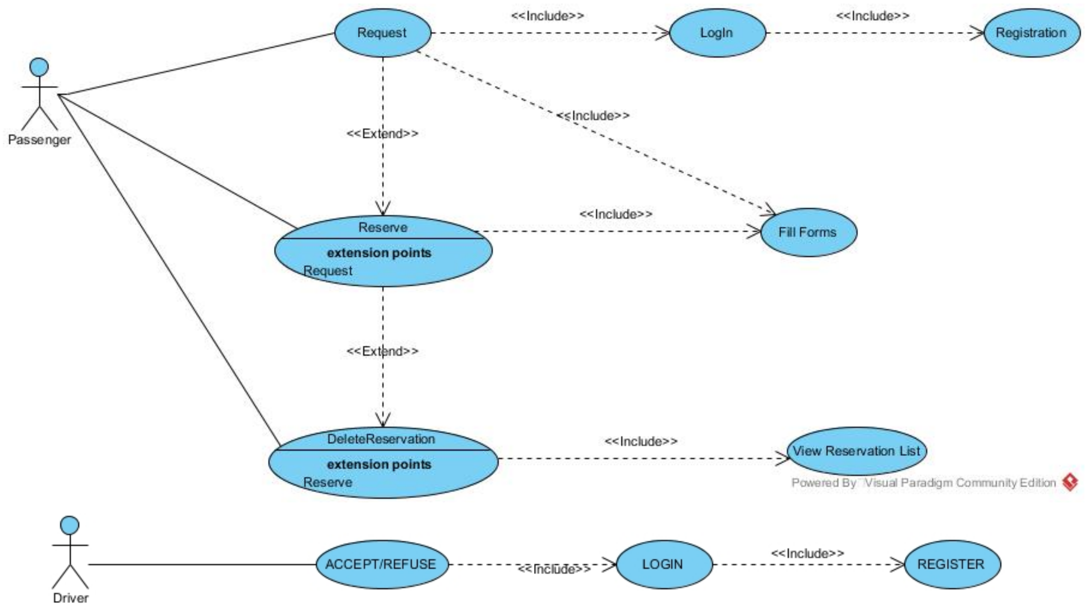
- **Reached Goals**: [G2]
- **Actors** : Passenger, Taxi Driver, Administrator
- **Starting Situation** :
  - A user can access to his/her profile page
- **Flows Event:**
  - The users want to Edit their profile.
  - The system open the information of users that has been saved on DB.
  - Users modify their information (add,Change,Delete)
  - Users confirm the action that they did
  - System will update information on the DB.
- **Final Situation** :
  - The information of User is updated
- **Exception** :
  1. The user regrets to change his/her information
  2. The connection between system and DB is failed.

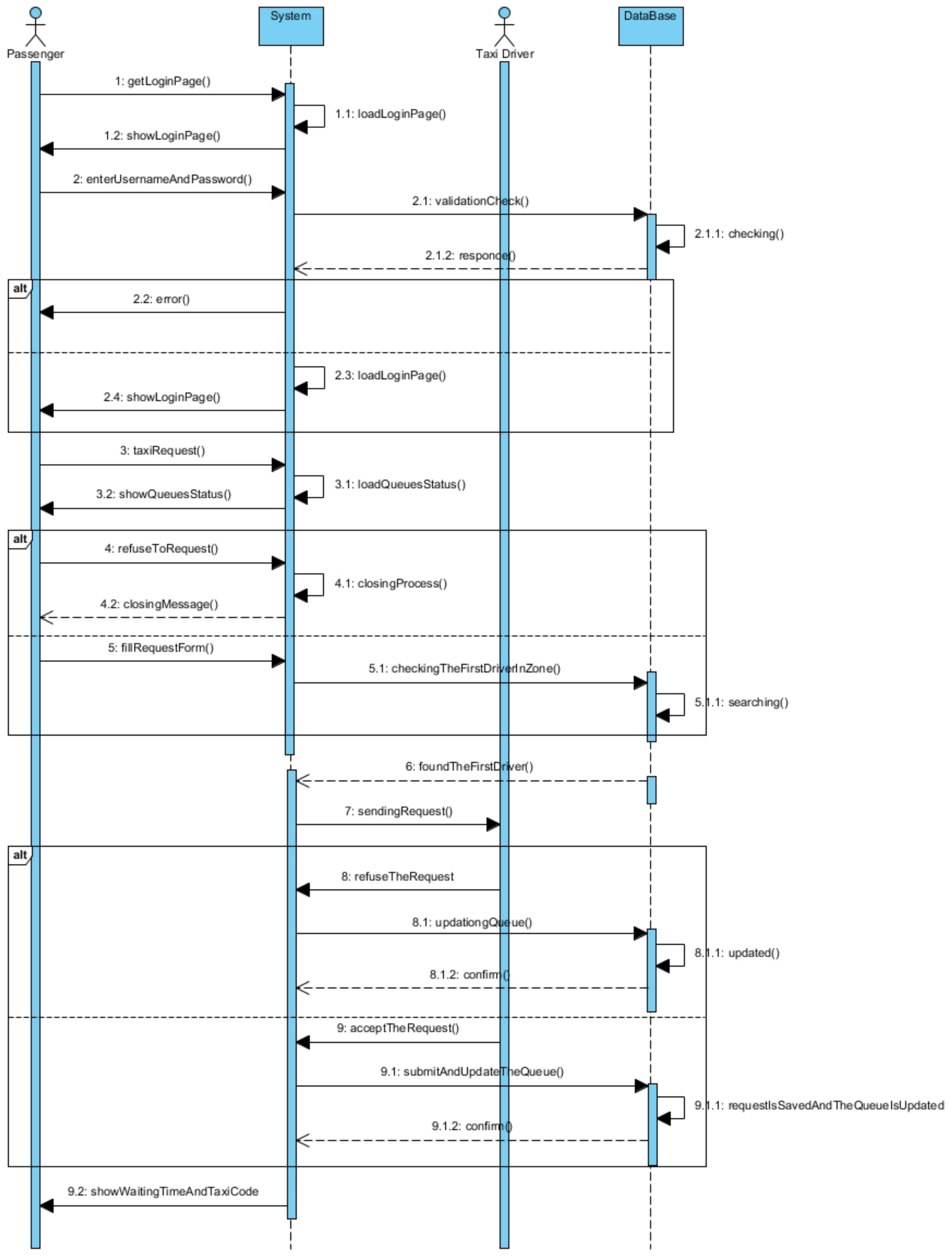


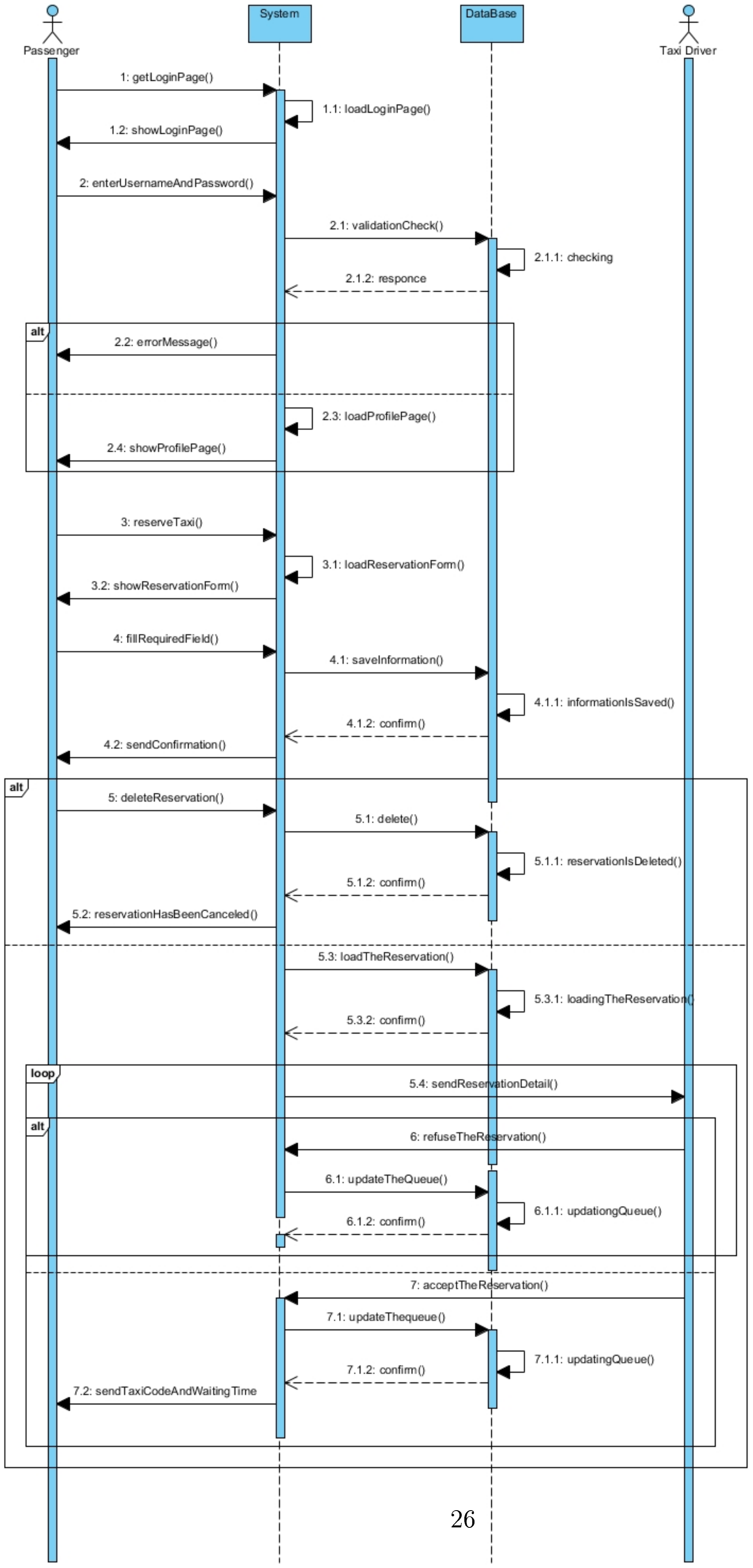

Registered User
System
DataBase
1: getLoginPage()
1.1: loadRequestedPage()
1.2: showRequestedPage()
2: enterUsernameAndPassword()
2.1: checkValidation()
2.1.1: checking()
2.1.2: responce()
alt
2.2: error()
2.3: loadProfilePage()
2.4: showUserPage()
3: editInformation()
3.1: save()
3.1.1: informationIsUpdated()
3.1.2: confirm()
4: successfullyEdited()

6.1.6 Request/Reserve or Cancel Reservation and Accept/Refuse

- **Reached Goals**: [G1]
- **Actors** : Passenger, Taxi Driver
- **Starting Situation** :
  - Passenger and Taxi Driver is already logged in and they can use their profile page.
- **Flows Event:**
  - The passenger Request\Reserve a taxi in his personal panel
  - The passenger fill the form of Request\Reservation
  - The system adds this Request\Reserve to the DB
  - The system will inform the first taxi driver in queue
  - The taxi driver confirm the Ride
  - The system will save the data of ride in the DB.
- **Final Situation** :
  - The ride is done successfully
- **Exception** :
  1. The passenger Cancel the Reuqest\Reservation.
  2. Tha taxi driver Refuse the Request\Reservation.
  3. The connection between sytem and DB is failed.

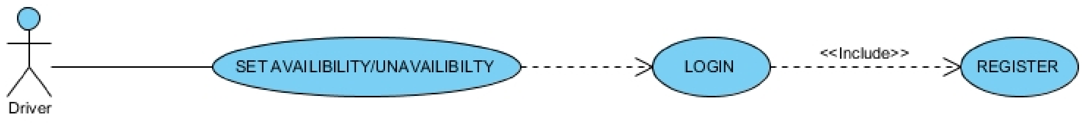


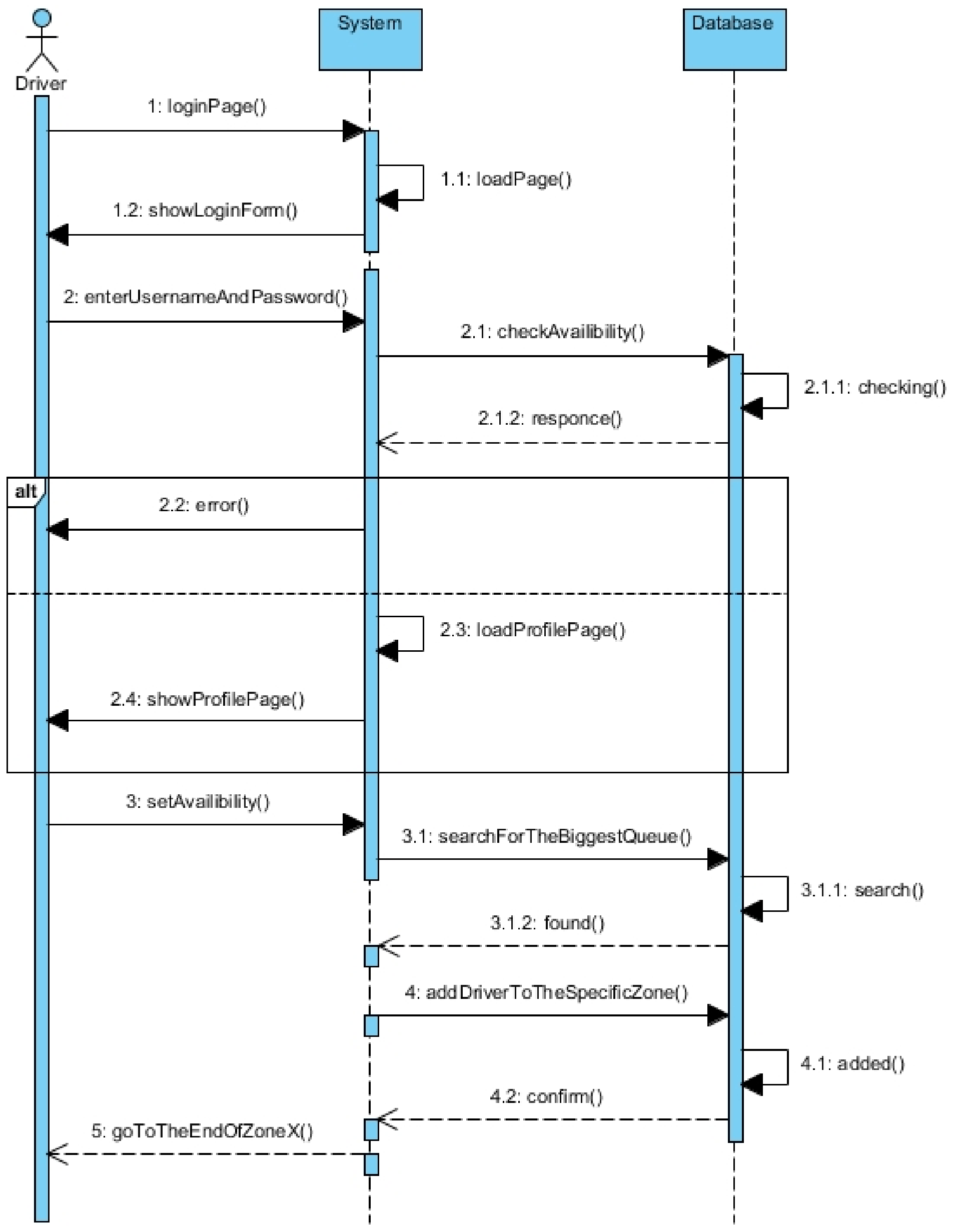

Passenger
System
Taxi Driver
DataBase
1: getLoginPage()
1.1: loadLoginPage()
1.2: showLoginPage()
2: enterUsernameAndPassword()
2.1: validationCheck()
2.1.1: checking()
2.1.2: responce()
alt
2.2: error()
2.3: loadLoginPage()
2.4: showLoginPage()
3: taxiRequest()
3.1: loadQueuesStatus()
3.2: showQueuesStatus()
alt
4: refuseToRequest()
4.1: closingProcess()
4.2: closingMessage()
5: fillRequestForm()
5.1: checkingTheFirstDriverInZone()
5.1.1: searching()
6: foundTheFirstDriver()
7: sendingRequest()
alt
8: refuseTheRequest
8.1: updationgQueue()
8.1.1: updated()
8.1.2: confirm()
9: acceptTheRequest()
9.1: submitAndUpdateTheQueue()
9.1.1: requestIsSavedAndTheQueueIsUpdated
9.1.2: confirm()
9.2: showWaitingTimeAndTaxiCode

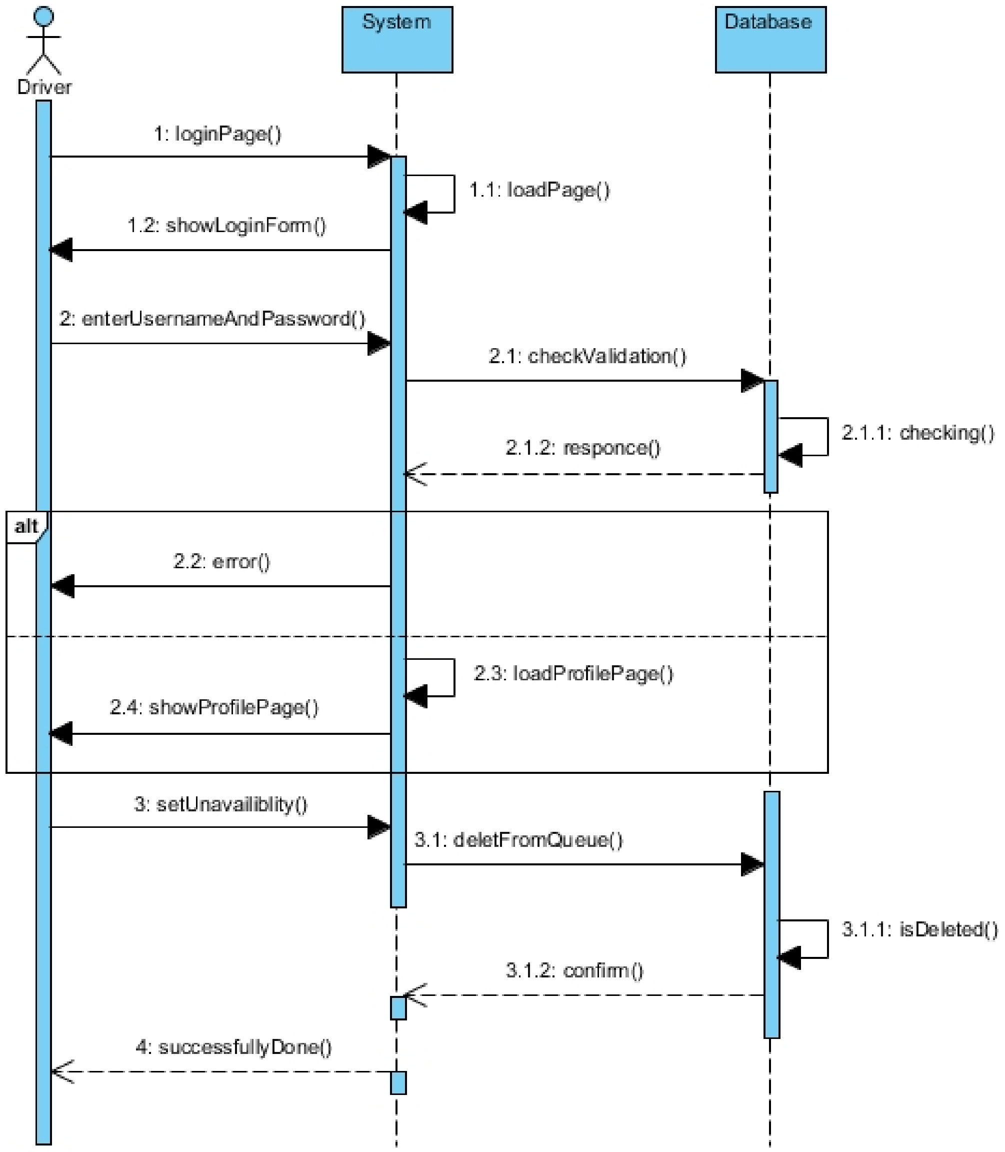

Passenger
System
DataBase
Taxi Driver
1: getLoginPage()
1.1: loadLoginPage()
1.2: showLoginPage()
2: enterUsernameAndPassword()
2.1: validationCheck()
2.1.1: checking
2.1.2: responce
alt
2.2: errorMessage()
2.3: loadProfilePage()
2.4: showProfilePage()
3: reserveTaxi()
3.1: loadReservationForm()
3.2: showReservationForm()
4: fillRequiredField()
4.1: saveInformation()
4.1.1: informationIsSaved()
4.1.2: confirm()
4.2: sendConfirmation()
alt
5: deleteReservation()
5.1: delete()
5.1.1: reservationIsDeleted()
5.1.2: confirm()
5.2: reservationHasBeenCanceled()
5.3: loadTheReservation()
5.3.1: loadingTheReservation()
5.3.2: confirm()
loop
5.4: sendReservationDetail()
alt
6: refuseTheReservation()
6.1: updateTheQueue()
6.1.1: updationgQueue()
6.1.2: confirm()
7: acceptTheReservation()
7.1: updateThequeue()
7.1.1: updatingQueue()
7.1.2: confirm()
7.2: sendTaxiCodeAndWaitingTime

6.1.7 Availibility/Unavailibility

- **Reached Goals**: [G2]
- **Actors** : Taxi Driver
- **Starting Situation** :
  - Taxi driver is already logged in
- **Flows Event:**
  - The taxi driver Set the availibility
  - The system finds him/her by identifier
  - move the taxi driver into the last position of the queue
- **Final Situation** :
  - Taxi driver is in the queue
- **Exception** :
  1. Taxi driver set unavailibility
  2. The connection between System and DB is failed

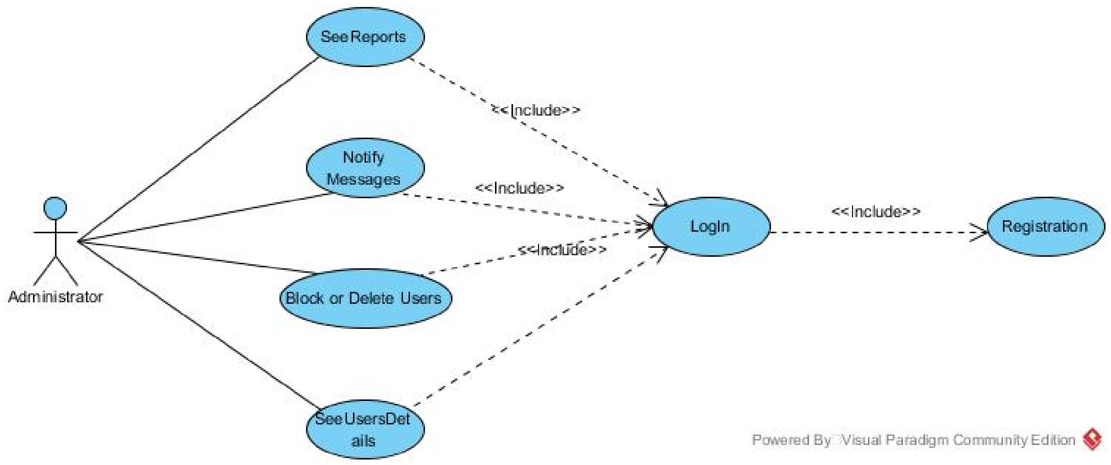


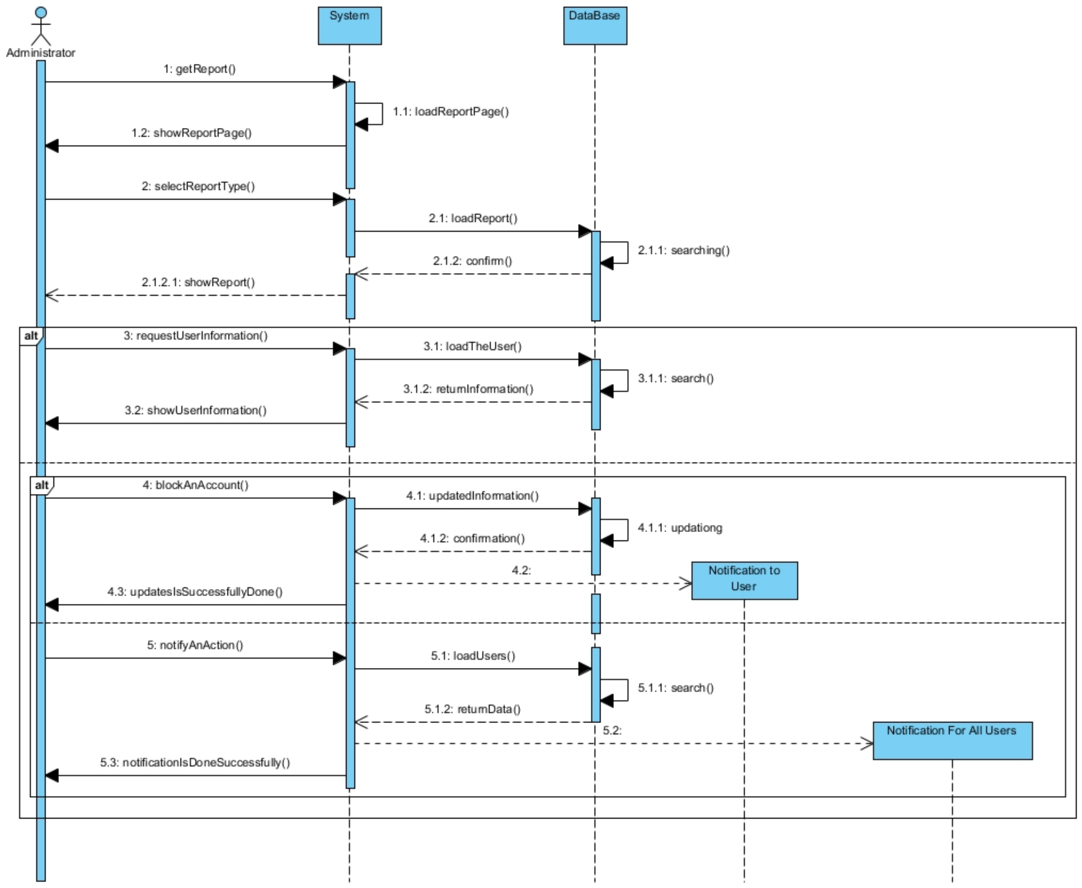

Driver
System
Database
1: loginPage()
1.1: loadPage()
1.2: showLoginForm()
2: enterUsernameAndPassword()
2.1: checkAvailibility()
2.1.1: checking()
2.1.2: responce()
alt
2.2: error()
2.3: loadProfilePage()
2.4: showProfilePage()
3: setAvailibility()
3.1: searchForTheBiggestQueue()
3.1.1: search()
3.1.2: found()
4: addDriverToTheSpecificZone()
4.1: added()
4.2: confirm()
5: goToTheEndOfZoneX()


Driver
System
Database
1: loginPage()
1.1: loadPage()
1.2: showLoginForm()
2: enterUsernameAndPassword()
2.1: checkValidation()
2.1.1: checking()
2.1.2: responce()
alt
2.2: error()
2.3: loadProfilePage()
2.4: showProfilePage()
3: setUnavailiblity()
3.1: deletFromQueue()
3.1.1: isDeleted()
3.1.2: confirm()
4: successfullyDone()

6.1.9 Administator Functionalities

- **Reached Goals**: [G1]
- **Actors** : Administrator
- **Starting Situation** :
  - administrator is logged in
- **Flows Event:**
  - administrator send a request to see monthly report
  - system load the report from DB
  - administrator block the user
  - system will remove the blocked user
- **Final Situation** :
  - administrator can see reports and do some functionality on users
- **Exception** :
  1. The connection between system and DB is failed

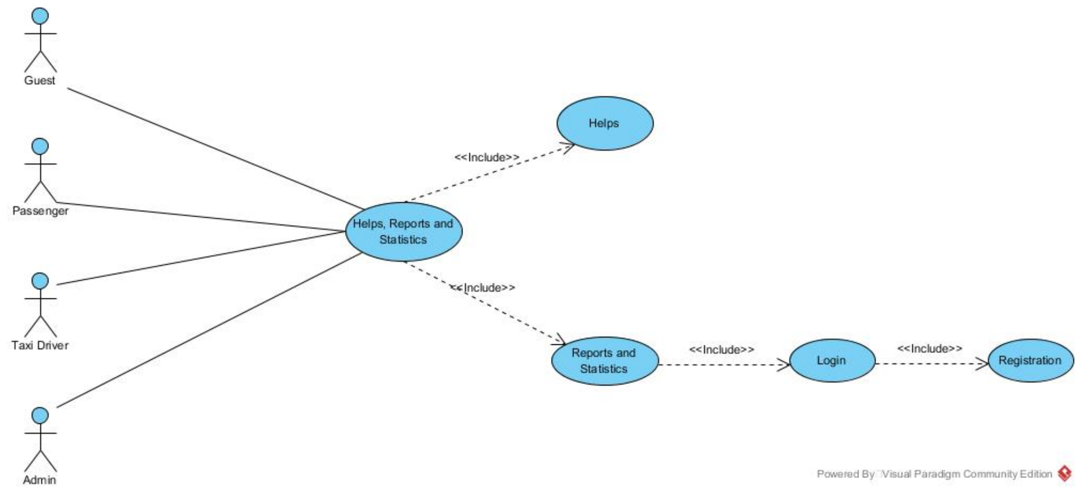




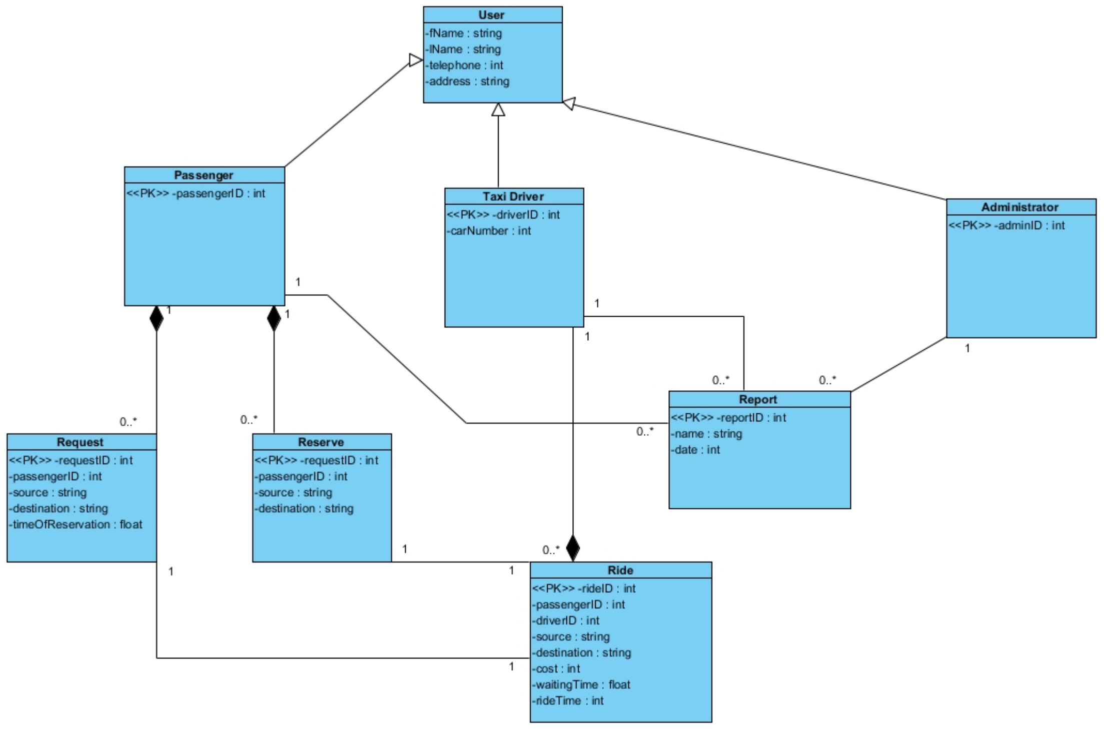
Administrator
System
DataBase
1: getReport()
1.1: loadReportPage()
1.2: showReportPage()
2: selectReportType()
2.1: loadReport()
2.1.1: searching()
2.1.2: confirm()
2.1.2.1: showReport()
alt
3: requestUserInformation()
3.1: loadTheUser()
3.1.1: search()
3.1.2: returnInformation()
3.2: showUserInformation()
alt
4: blockAnAccount()
4.1: updatedInformation()
4.1.1: updationg
4.1.2: confirmation()
4.2:
Notification to User
4.3: updatesIsSuccessfullyDone()
5: notifyAnAction()
5.1: loadUsers()
5.1.1: search()
5.1.2: returnData()
5.2:
Notification For All Users
5.3: notificationIsDoneSuccessfully()

6.1.10 Help, Reports and Statistics

- **Reached Goals**: [G5]
- **Actors** : Guest, Passenger, Taxi Driver, Administrator
- **Starting Situation** :
  - Someone needs help or registered user wants see reports
- **Flows Event:**
  - System load the specific pages and datas from DB
- **Final Situation** :
  - Guests and registered users will show what they want
- **Exception** :
  1. The connection between system and DB is failed



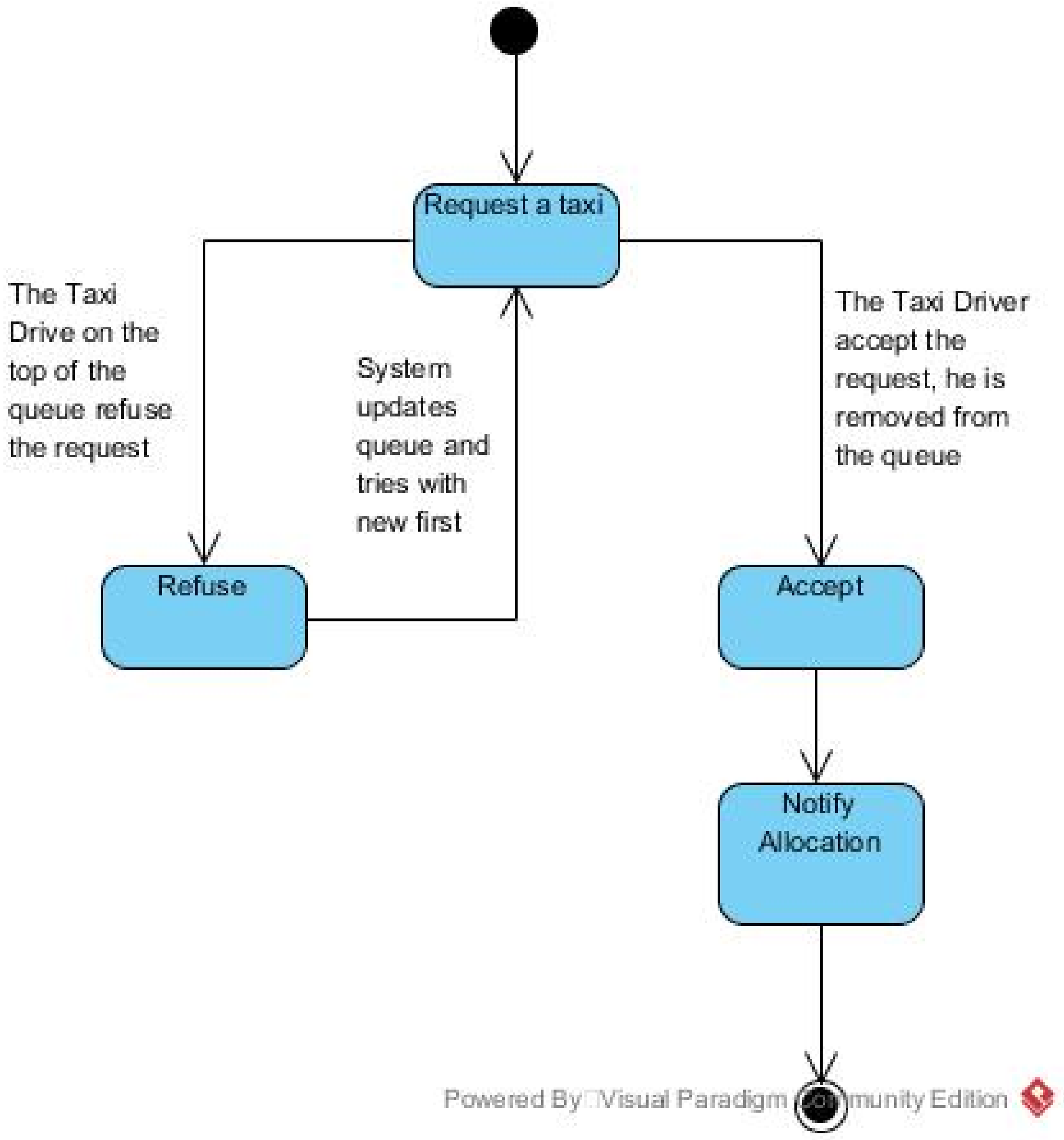
6.2 Class Diagram



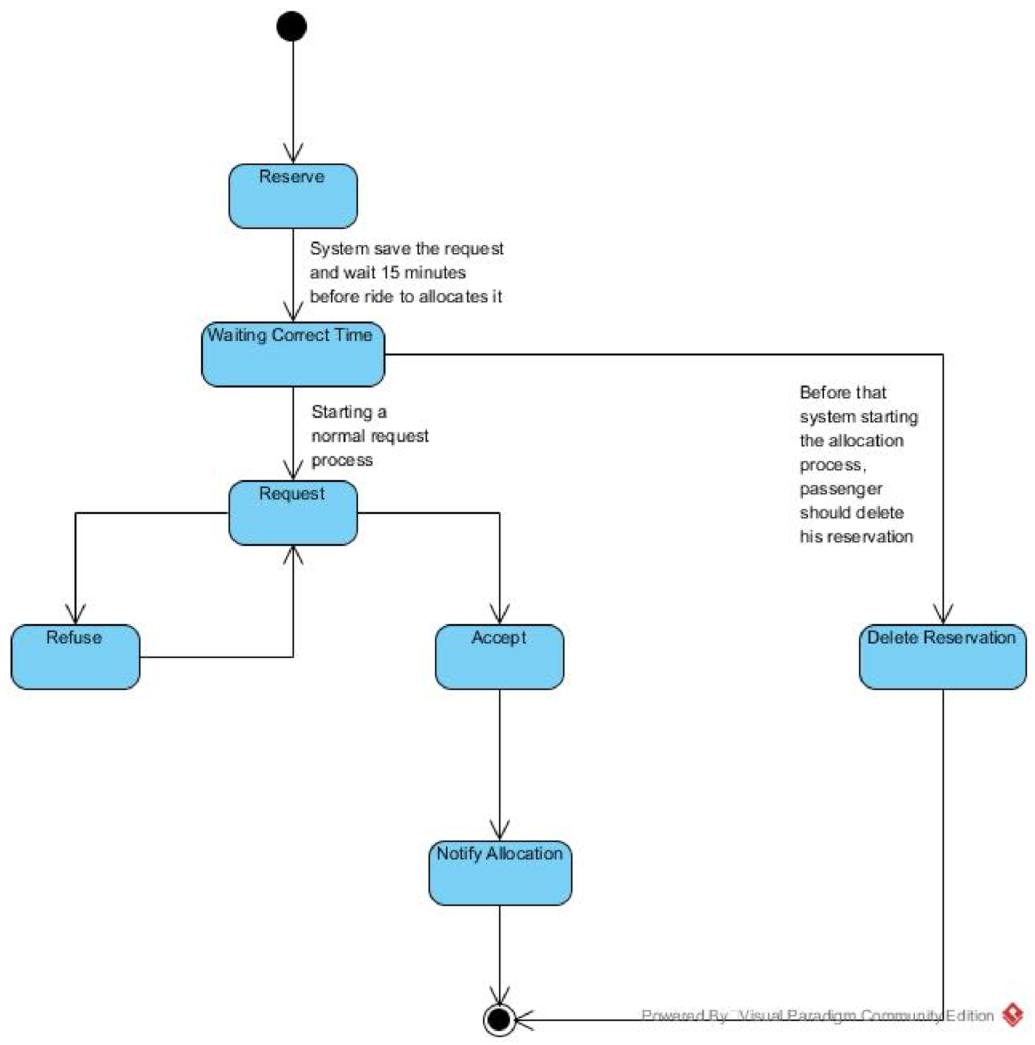
6.3 State Chart Diagram

In myTaxiDriver® there are two parts that is better specify with this diagram on order to point out the life cycle of these two entities.

6.4.1 Request a Taxi


Request a taxi
The Taxi Drive on the top of the queue refuse the request
System updates queue and tries with new first
The Taxi Driver accept the request, he is removed from the queue
Refuse
Accept
Notify Allocation
Powered By Visual Paradigm Community Edition

6.4.2 Reserve a taxi



7.Alloy Modelling

In this chapter we want demostrate the concistency of our idea for myTaxiService®. For doing this we have used Alloy, that generate a simulation of our real word. In the first section we propose the code that rappresent out model. In the second one we report three rappresentation about parts of our service.

1. Messaging between actors without any request or reservation.
2. Flow events during a request
3. Flow events during a reservation.

We don't report a general image becouse it should not add nothing to the clarity of the project. However, running shows that our model is consistent.

7.1 Alloy Code

```
module myTaxiService

//°°°°°°SIGNATURES°°°°°°//

sig Guests{
        loginPassenger : some Passenger,
        loginTaxiDriver : some TaxiDriver,
        loginAdmin : one Admin,
        seeHelp : lone Helps
        }
        {
        #Guests = 1
}

sig Passenger{
        request : lone Request,
        reserve : set Reserve,
        logout : lone LogOut,
        seeReport : lone SeeReportOrStatistics,
        seeHelp : lone Helps,
}


sig SystemWithDatabase{
        request : set ForwardsReq,
        reserve: set AllocateRes,
        confirmRes: set ConfirmRes,
        confirmReq: set ConfirmReq
        }
        {
        #SystemWithDatabase=1
}

sig TaxiDriver{
        availability : lone TDAvailable,
        notavailability : lone TDNotAvailable,
        seeHelp : lone Helps,
        seeReport : lone SeeReportOrStatistics,
        logout : lone LogOut
}

sig Admin{
        seeReport : lone SeeReportOrStatistics,
        sendMessage : lone AdminNotify,
        logout : lone LogOut
        }
        {
        #Admin = 1
}

sig LogOut {
        senderA: lone Admin,
        senderP: lone Passenger,
        senderT: lone TaxiDriver,
        reveiver: one Guests
        }
        {
        #LogOut = 1
}

sig Request {
        sender: one Passenger,
        receiver: one SystemWithDatabase
}
```

```
sig ForwardsReq{
        sender: one SystemWithDatabase,
        receiver: one TDAvailable
        }
        {
        #ForwardsReq = #Request
}

sig Reserve {
        sender: one Passenger,
        receiver: one SystemWithDatabase
}

sig AllocateRes{
        sender: one SystemWithDatabase,
        receiver: one TDAvailable
        }
        {
        #AllocateRes = #Reserve
}

sig TDAvailable{
        accept : lone Accept,
        refuse : lone Refuse
}


sig TDNotAvailable{
        availability : one TDAvailable
}

sig Accept {
                sender: one TDAvailable,
                receiver: one SystemWithDatabase
}

sig Refuse {
        sender: one TDAvailable,
        receiver: one SystemWithDatabase
}

sig ConfirmReq {
                sender: one SystemWithDatabase,
                receiver: one Passenger
}

sig ConfirmRes {
        sender: one SystemWithDatabase,
        receiver: one Passenger
}

sig SeeReportOrStatistics{
        senderA: lone Admin,
        senderP: lone Passenger,
        senderT: lone TaxiDriver,
        receiver: one SystemWithDatabase
        }
        {
        #SeeReportOrStatistics = 1
}
```

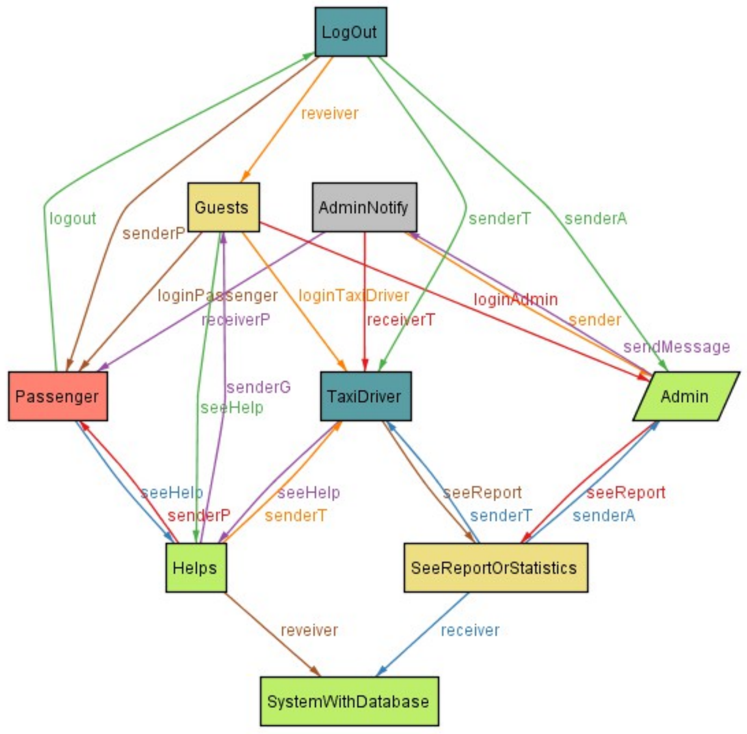
```
sig AdminNotify{
        sender: one Admin,
        receiverP : set Passenger,
        receiverT : set TaxiDriver
}

sig Helps{
        senderG: lone Guests,
        senderP: lone Passenger,
        senderT: lone TaxiDriver,
        reveiver: some SystemWithDatabase
        }
        {
        #Helps = 1
}



//°°°°°°ASSERTIONS°°°°°°//

//Check that there shouldn't be request or reservation without login like passenger
assert NoRequestOrReserveWithoutUser{
        all res: Reserve, req:Request, g:Guests | #g.loginPassenger = 0 => (#res + #req) =0
}

check NoRequestOrReserveWithoutUser

//Check that an AdminNotify should be sent to any passenger
assert AdminNotifyForAll{
        no disj p1, p2: Passenger | (one admNot: AdminNotify| admNot.receiverP = p1 && admNot.receiverP =
p2 )
}

check AdminNotifyForAll

//Check that each user cannot do anythings if he have done logout
assert noActionWithLogOut{
        all p:Passenger | #p.logout >0 => (#p.request = 0 && #p.reserve = 0 && #p.seeReport = 0)
        all tx: TaxiDriver | #tx.logout >0=>(#tx.availability =0 && #tx.notavailability=0 && #tx.seeReport =0)
        all a: Admin | #a.logout >0 => (#a.seeReport = 0 && #a.sendMessage = 0)
}

check noActionWithLogOut

//°°°°°°FACTS°°°°°°//

fact passengerAndGuestsRestrictions{
        all p: Passenger | #p.request<2
        all p: Passenger | (#p.request > 0 || #p.reserve > 0 || #p.seeReport > 0 ) => (#p.logout = 0)
        all p: Passenger | (#p.logout = 1) => (#p.request = 0 && #p.reserve = 0 && #p.seeReport = 0)
        no disj req1, req2: Request | req1.sender = req2.sender
        no disj an1,an2:AdminNotify | an1.receiverP =an2.receiverP
        all g: Guests | (#g.loginTaxiDriver = #TaxiDriver) &&  (#g.loginPassenger = #Passenger) //&&
                        (#TaxiDriver >= #Passenger)
}
```

```
fact taxiDriverRestriction{
        all tx : TaxiDriver | #tx.availability + #tx.notavailability = 1
        all av : TDAvailable | (#av.accept + #av.refuse) = 1
        all tx : TaxiDriver | ((#tx.availability + #tx.notavailability = 1) || #tx.seeReport>0) => #tx.logout = 0
        all tx : TaxiDriver | #tx.logout = 1 => ((#tx.availability + #tx.notavailability) = 0
                        && #tx.seeReport = 0)
        all av: TDAvailable, tx : TaxiDriver, notav : TDNotAvailable |
                        ((tx.availability = av || (notav.availability = av && tx.notavailability = notav)) =>
                        #(av.accept + av.refuse) =1)
        all av: TDAvailable, tx : TaxiDriver, notav : TDNotAvailable | #av = #tx && #notav <= #tx
        all av: TDAvailable, notav : TDNotAvailable {one tx : TaxiDriver |(notav.availability = av <=>
                        tx.notavailability = notav)}
        all av: TDAvailable {one tx : TaxiDriver |tx.availability = av}
        no disj ac1, ac2: Accept | ac1.sender = ac2.sender
        no disj re1, re2: Refuse | re1.sender = re2.sender
        no disj forReq1, forReq2: ForwardsReq | forReq1.receiver = forReq2.receiver
        no disj allRes1, allRes2: AllocateRes | allRes1.receiver =allRes2.receiver
        all allRes: AllocateRes { all forReq: ForwardsReq | allRes.receiver != forReq.receiver}
}

fact adminRestriction{
        all a: Admin | (#a.seeReport>0 || #a.sendMessage > 0) => #a.logout =0
        all a: Admin | #a.logout = 1 => (#a.seeReport=0 && #a.sendMessage = 0)
}

fact correctCorrispondingBetweenParts{

        //each Taxi Driver will have his availability/notavailability
        no disj tx1,tx2: TaxiDriver | tx1.availability=tx2.availability
        no disj tx: TaxiDriver, avnot:  TDNotAvailable  | tx.availability = avnot.availability
        no disj avnot1,avnot2 :TDNotAvailable | avnot1.availability= avnot2.availability
        no disj av1,av2: TDAvailable  | av1.accept=av2.accept && av1.refuse = av2.refuse
        all av : TDAvailable | (#av.accept + #av.refuse) = 1
        all tx:TaxiDriver, res : Reserve, req: Request | #tx >= (#req + #res)

        //each passenger will have his request and his reserve
        no disj p1,p2:Passenger | p1.request = p2.request
        no disj p1,p2:Passenger | p1.reserve = p2.reserve

        //correct matches in Help and ReportOrStatistics
        all p:Passenger, r:Request | p.request=r <=> r.sender = p
        all p:Passenger, r:Reserve | p.reserve=r <=> r.sender = p
        all p:Passenger, h:Helps | p.seeHelp = h <=> h.senderP = p
        all g:Guests, h:Helps | g.seeHelp = h <=> h.senderG = g
        all tx : TaxiDriver, h:Helps | tx.seeHelp = h <=> h.senderT = tx
        all p:Passenger, s:SeeReportOrStatistics | p.seeReport = s <=> s.senderP = p
        all ad : Admin, s:SeeReportOrStatistics | ad.seeReport = s <=> s.senderA = ad
        all tx : TaxiDriver, s:SeeReportOrStatistics| tx.seeReport = s <=> s.senderT = tx

        //AllocateRes and ForwardReq applicate only at real TDAvailable
        all admnot : AdminNotify| (#Passenger + #TaxiDriver) = (#admnot.receiverP + #admnot.receiverT)
        all tx:TaxiDriver, tdav: TDAvailable, tdnot: TDNotAvailable, al : AllocateRes, fo : ForwardsReq|
                ((tdnot.availability= tdav && tx.notavailability = tdnot) || tx.availability = tdav) =>
                ((al.receiver =tdav || fo.receiver = tdav) => ((#tdav.accept+#tdav.refuse) = 1))
        all a:Accept, tdav:TDAvailable| a.sender = tdav <=> tdav.accept = a
        all r: Refuse, tdav:TDAvailable| r.sender = tdav <=> tdav.refuse = r
}
```

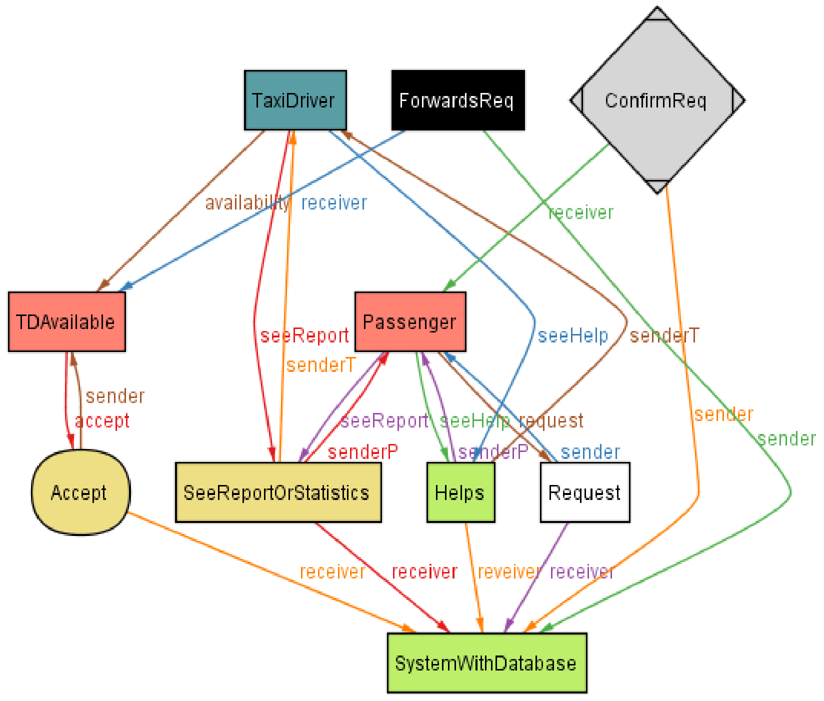
```
//°°°°°°PREDICATES°°°°°°//

//Here we want point out the relation between Users, Admin and System with  respect to messages interaction
//Here there isn't any confirm because it need also to one request, one accept and all the other
pred showMessagesAndNotify(){
	#AdminNotify = 1
	#ConfirmRes = 0
	#ConfirmReq = 0
	#Helps = 1
	#Passenger = 1
	#Request = 0
	#Reserve = 0
	#SeeReportOrStatistics = 1
	#TDAvailable = 0
	#TDNotAvailable = 0
}

run showMessagesAndNotify for 3

//Here we want point out on a basic situation with a passenger use our app for a request or for a reservation,
//without admin interactions.Taxi is available and accept the option.
pred showEasyReserve(){
	#Accept = 1
	#Admin = 0
	#AllocateRes = 1
	#ConfirmRes = 1
	#ConfirmReq = 0
	#ForwardsReq = 0
	#Passenger = 1
	#Refuse = 0
	#Request = 0
	#Reserve = 1
	#TaxiDriver =1
}

run showEasyReserve for 2

pred showEasyRequest(){
	#Accept = 1
	#Admin = 0
	#AllocateRes = 0
	#ConfirmRes = 0
	#ConfirmReq = 1
	#ForwardsReq = 1
	#Passenger = 1
	#Refuse = 0
	#Request = 1
	#Reserve = 0
	#TaxiDriver =1

run showEasyRequest for 2

pred show(){}

run show for 3
```

7.2 Generated World

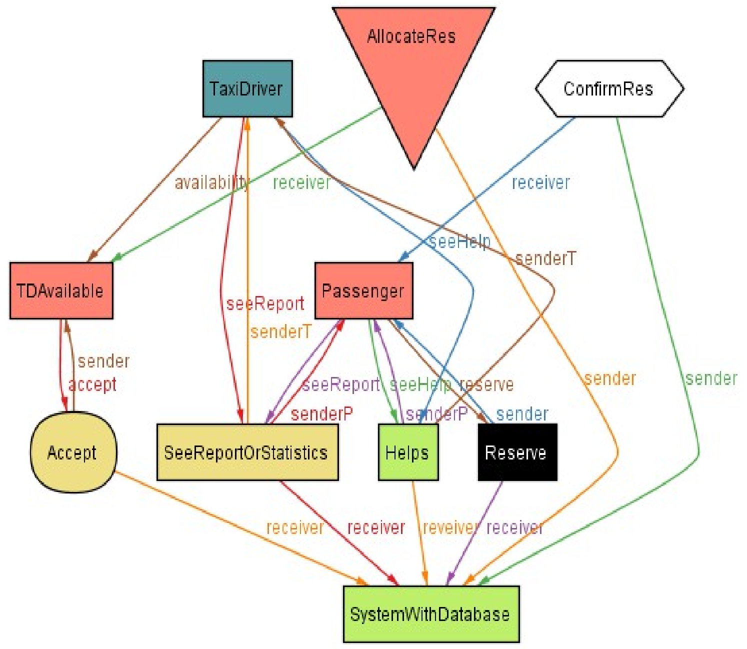
7.2.1 Messagging



7.2.2 Request



7.2.3 Reserve



8.Used Tools

FOR CREATE THIS documet we have used:

LYX to create a well-formatted document

moqups.com to create the UI sketches;

VisualParadigm10 Community Edition to create Sequence Diagrams and State Charts;

AlloyAnalyzer4.2 to prove the consistency of our model;