## 1 One

## 2 Two

## 3 Specific Requirements

This section of the document is dedicated at giving an in-depth description of the platform's requirements, and is to be kept as reference during all future phases of development.

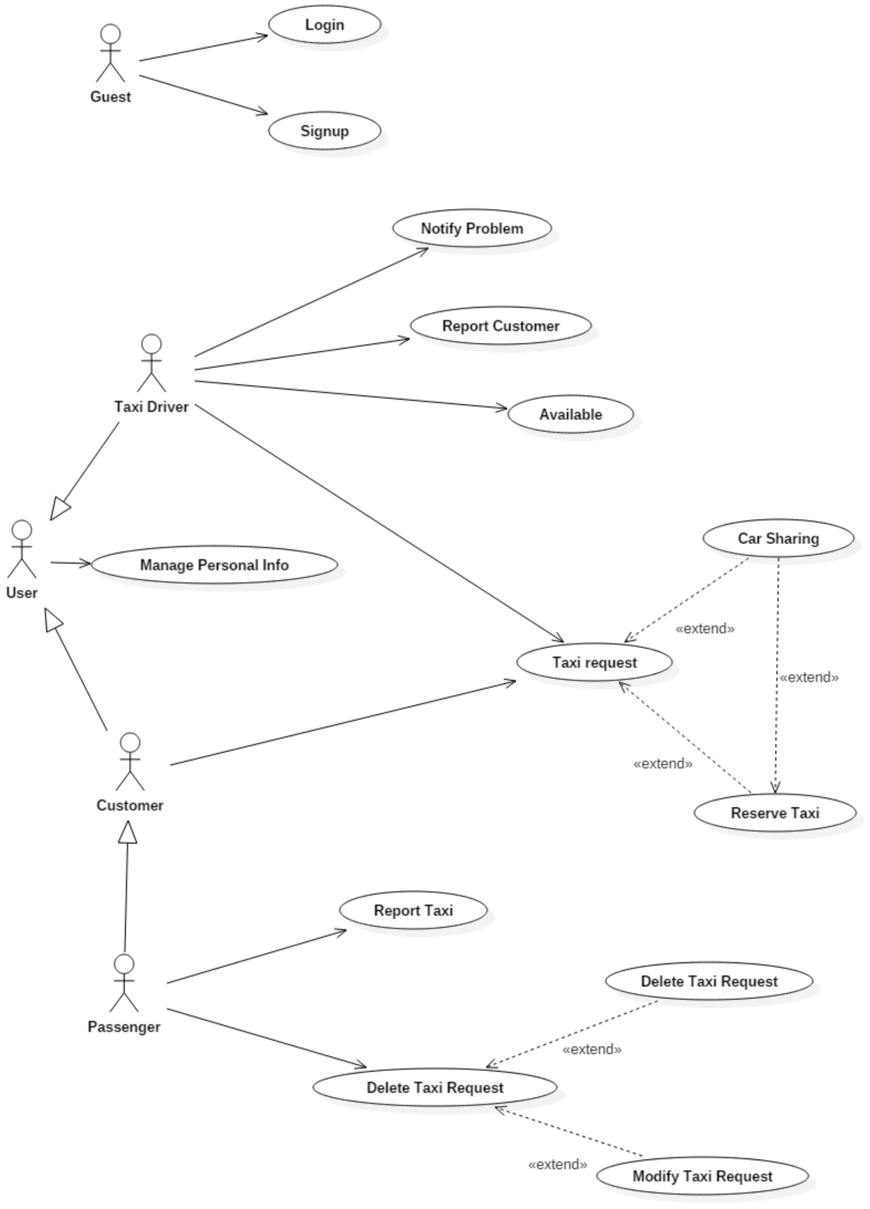
### 3.1 External interfaces

Being MyTaxiService a fully service-oriented platform, its only external interfaces must be those reserved for the final users; there is no need to design specific maintenance access to the back-end system as this is already fully standardized and does not need specific functionalities other than the usual system administration tools.

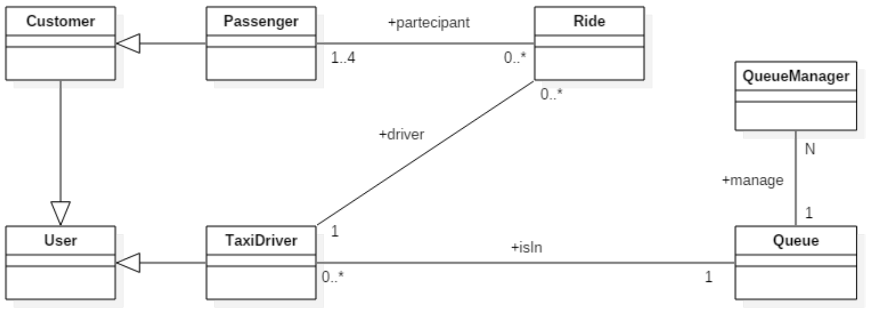
As briefly described in section **??**, the main principle that must guide the design of the external interfaces of the platform is that of business identity continuity. This section contains a set of design mock-ups that are to be kept as reference during the development of the user interfaces.

## 3.2 Functions

## 3.3 Use Cases



## 3.4 Class Diagram



## 3.5 Scenarios

To help the reader understand the above stated requirements, a brief description of how a use case might look like in the real world is given below.

In the examples, Adam, Michelle and Joanne are customers who intend to request a taxi and Hector, Monica, Jim and Samuel are taxi drivers of the town.

### 3.5.1 Sign up

Adam has just downloaded the customer-side app and wants to sign up into the platform. He requests the customer registration page, fills the form and submits the request to the system. If Adam's e-mail and username are unique, the system gives Adam a confirmation of the success of the operation and redirects Adam to the login page; otherwise, an error message is displayed on Adam's phone.

### 3.5.2 Login

Adam, now registered, inserts the username and password in the login form and clicks the login button; the system checks the information and, if the username-password combination is correct, redirects Adam to his own user profile page; otherwise, an error message is displayed on Adam's phone.

### 3.5.3 Available

Hector, already logged into the platform, starts his working by day opening his taxi-side application and communicating his availability to the system. The system updates the taxi queue in Hector's zone and sends Hector a notification with his position in the queue.

### 3.5.4 Taxi request

Adam, now logged into the system, wants to book a taxi to go home. He opens the taxi request page on the app, and requests a taxi. The system forwards

Adam's request to the queue associated with Adam's position, and Hector, which is the first taxi driver in the queue, is notified with the request.

Unfortunately, Hector has now decided to take a break and does not want to take charge of this ride; he refuses Adam's request by tapping a button on the app, and the system forwards the request to Monica, the taxi driver immediately after Hector in the queue.

As she accepts Adam's request, Adam receives a notification on the app with the estimated waiting time.

#### 3.5.5 Book a Taxi

While on Monica's taxi, Adam wants to book a taxi for that evening at 6 PM, in order to go to the cinema. He opens the *Taxi request* page of the app, and fills and submits the request form.

The system checks the information (sending eventual error notifications back to Adam) and forwards Adam's request to Jim, by using a specific selection algorithm over taxi drivers in the queue associated to Adam's zone.

Jim decides to accept Adam's booking, and will keep his schedule free for the time that Adam requested.

#### 3.5.6 Car sharing

Michelle and Joanne live in the same neighborhood, and they both decide to go see a fair on the other side of the Town. Since they are both short on money, after opening the *Taxi request* page of the app they both check the *car sharing* option; they then submit their requests.

The system performs a check on Michelle and Joanne's requests

#### 3.5.7 Manage *Reserve Taxi* Request

Later that day, Adam browses the platform's website from his laptop's browser, and opens the *Manage taxi request* page to change the booking time from 6PM to 7PM.

The system checks whether Adam's request is acceptable (there must be at least two hours between the current time and the requested time), and eventually forwards the changes to Jim.

Jim accepts the modification and a confirmation is sent back to Adam.

#### 3.5.8 Report Taxi

Jim picks Adam up at 7PM. During the ride Jim lights up a cigarette and is unreasonably rude towards Adam.

Adam opens the *Report taxi* page on the app, to file a complaint about Jim's behavior. The system updates Jim's profile information with the new report and confirms the success of the operation to Adam.

### 3.5.9 Report Customer

After the ride, Adam is annoyed by the behavior of Jim and refuses to pay for the ride.

Jim opens the *Report user* page, fills the complaint form and submits it to the system. The system updates Adam's profile information with the new report and confirms the success of the operation to Jim.

### 3.5.10 Manage Personal Information

Joanne has opened a new main email account.

She opens her profile page from the app, clicks on the *edit* button and changes her email address to match the new one; she then submits the new information.

The system performs a check on the information, updates Joanne's profile and notifies the success of the operation to Joanne.

### 3.5.11 Report Problem

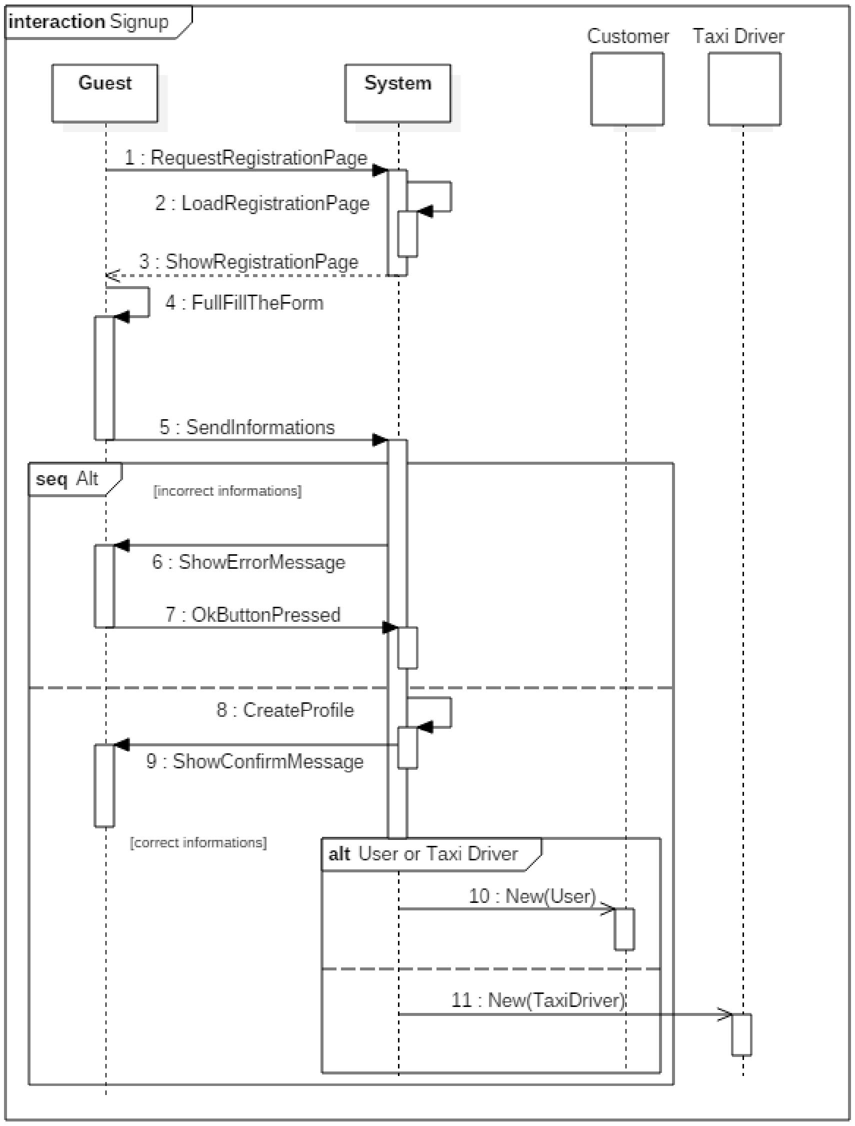
During a ride, Hector has a problem with his taxi's engine and can't bring Joanne to her destination.

Through the *Report problem* page of the app, he notifies the problem to the system by filling the form and submitting. The system acknowledges the report and asks Hector if he'll be needing a new taxi; Hector confirms, and the system forwards his request to Samuel, who is the first taxi driver in Hector and Joanne's current zone.

## 3.6 Flow of events

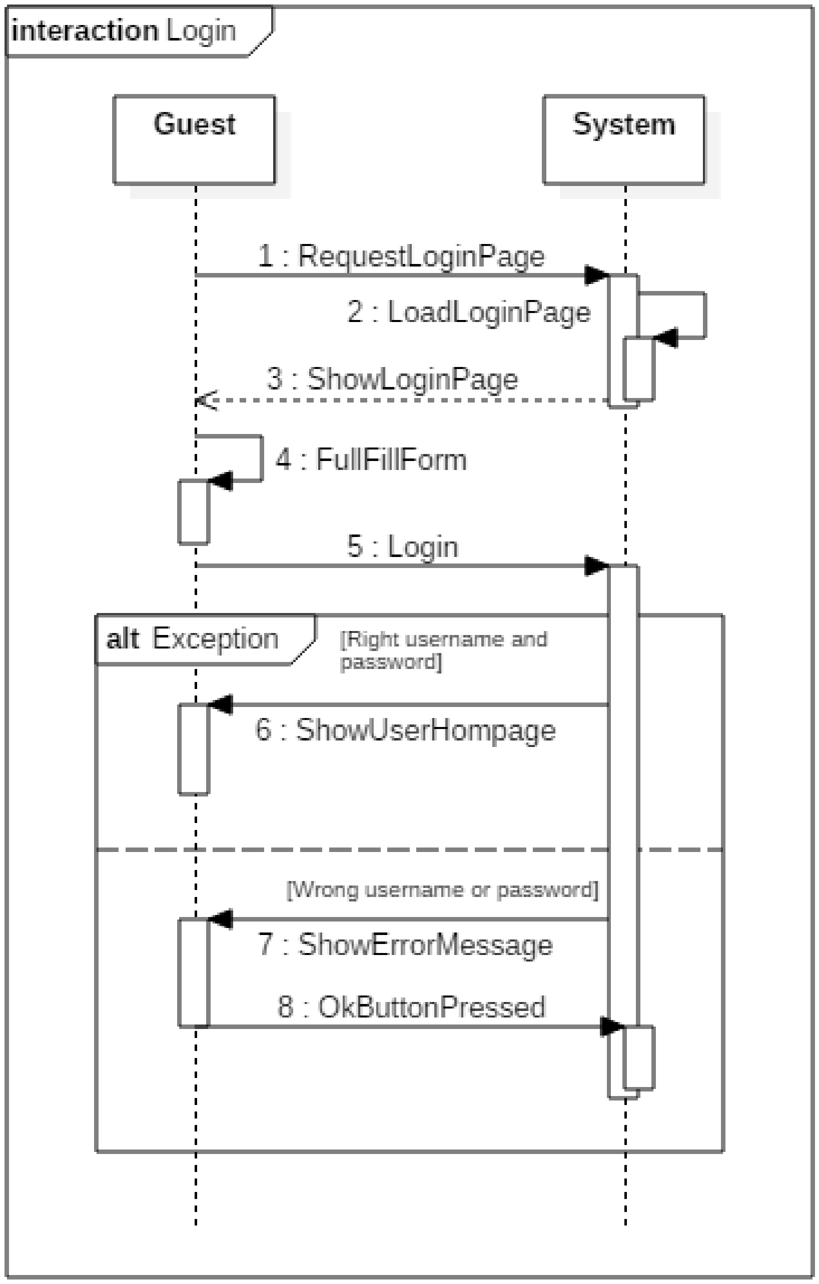
### 3.6.1 Sign-up

| | |
|---|---|
| Actors | Guest |
| Preconditions | The guest is not registered into the system |
| Execution Flow | 1. The guest requests the registration page<br>2. The system asks for the sign-up information<br>3. The guest fills the form and submits the request<br>4. The system checks the uniqueness of the username and e-mail<br>5. The system creates the customer (or taxi driver) profile<br>6. The system sends the confirmation to the guest |
| Postconditions | The guest is now a registered user |
| Exceptions | The e-mail or username are not unique or, in the case of a taxi driver sign-up, the license is not valid |


interaction Signup
Guest
System
Customer
Taxi Driver
1 : RequestRegistrationPage
2 : LoadRegistrationPage
3 : ShowRegistrationPage
4 : FullFillTheForm
5 : SendInformations
seq Alt
[incorrect informations]
6 : ShowErrorMessage
7 : OkButtonPressed
8 : CreateProfile
9 : ShowConfirmMessage
[correct informations]
alt User or Taxi Driver
10 : New(User)
11 : New(TaxiDriver)

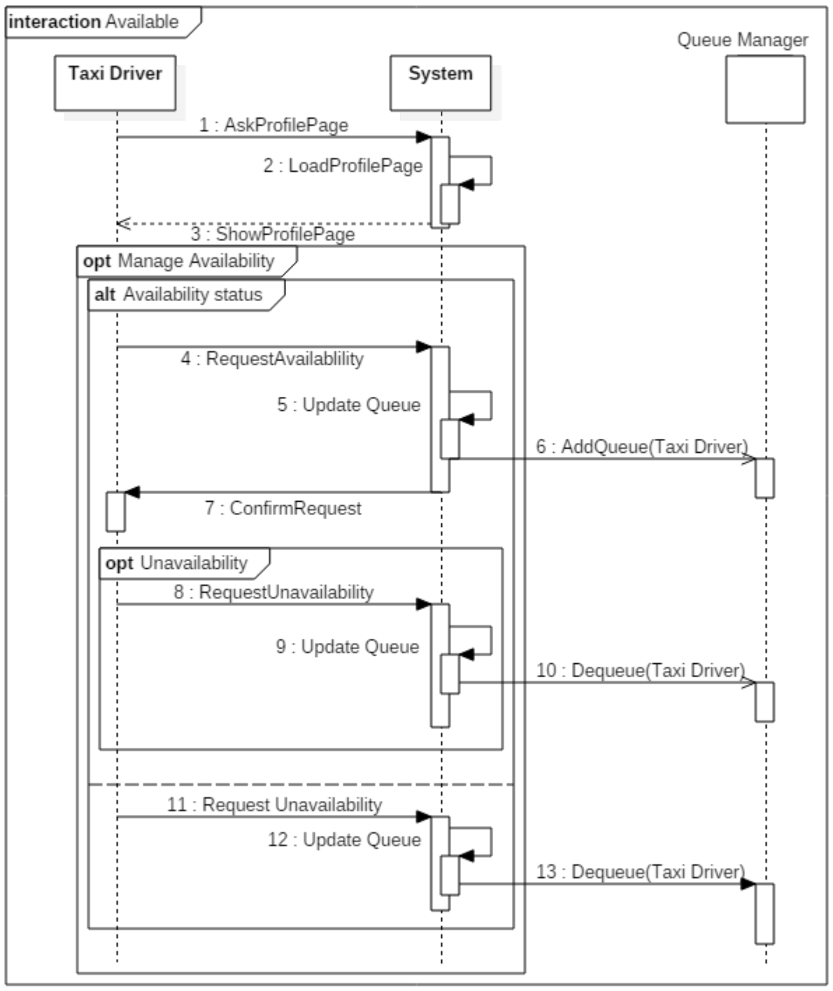
### 3.6.2 Login

| | |
|---|---|
| Actors | Guest |
| Preconditions | The guest has already a profile into the system |
| Execution Flow | 1. The guest requests the login page<br>2. The system requires the login information (Username, password)<br>3. The guest fills the form and submits the request<br>4. The system checks the username and password<br>5. The system sends a login confirmation<br>6. The guest is logged into the system<br>7. The guest is redirected to the user profile page |
| Postconditions | The guest is now a logged-in user |
| Exceptions | The username-password combination is incorrect, so the guest cannot log in |


interaction Login
Guest
System
1 : RequestLoginPage
2 : LoadLoginPage
3 : ShowLoginPage
4 : FullFillForm
5 : Login
alt Exception
[Right username and password]
6 : ShowUserHompage
[Wrong username or password]
7 : ShowErrorMessage
8 : OkButtonPressed

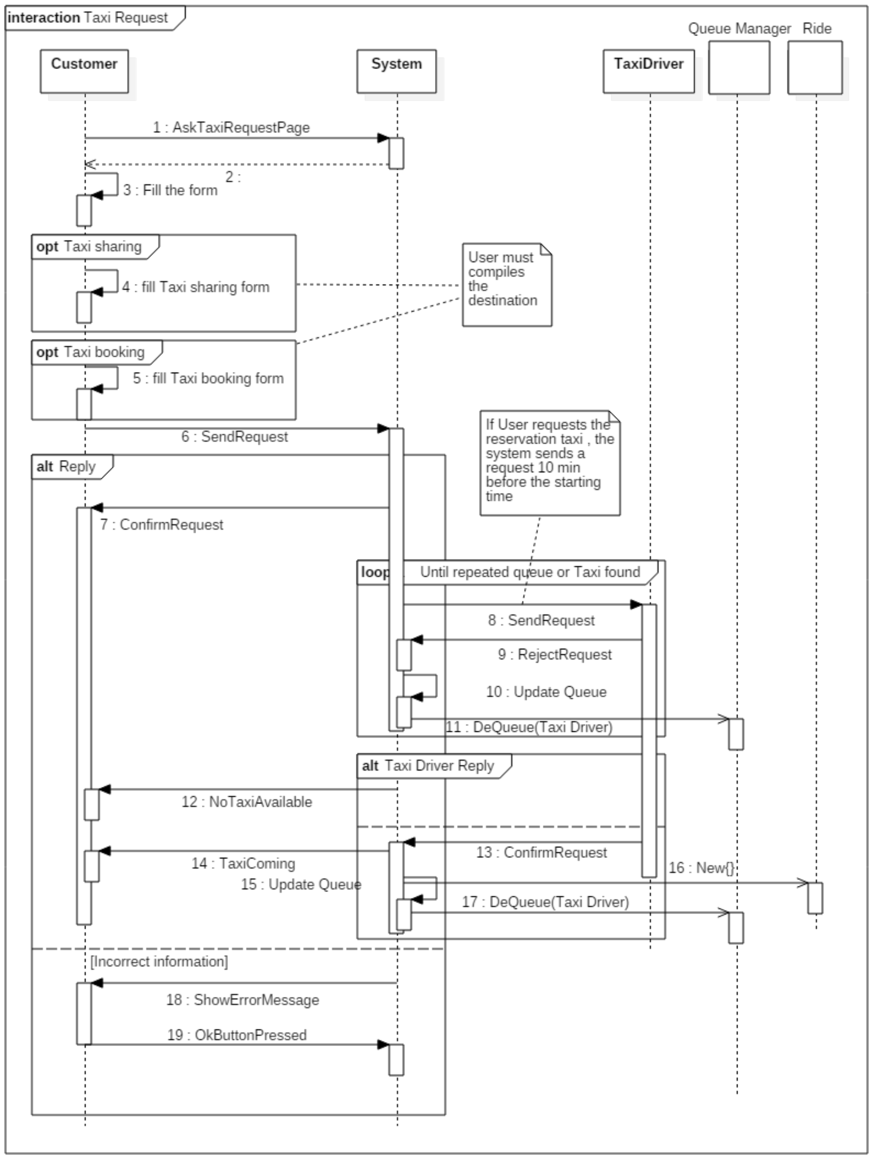
#### 3.6.3 Available

| | |
|---|---|
| Actors | Taxi driver |
| Preconditions | |
| Execution Flow | 1. The taxi driver requests their profile page<br>2. The system displays the user's personal information<br>3. The taxi driver can choose to change their availability, becoming available or unavailable<br>4. The taxi driver sends the request to the system<br>5. The system updates the queue<br>6. The system returns a confirmation to the taxi driver |
| Postconditions | The taxi driver is now available |
| Exceptions | • The taxi driver is located in a invalid zone<br>• The taxi driver is carrying a passenger<br>• The taxi driver is not available |


interaction Available
Taxi Driver
System
Queue Manager
1 : AskProfilePage
2 : LoadProfilePage
3 : ShowProfilePage
opt Manage Availability
alt Availability status
4 : RequestAvailablility
5 : Update Queue
6 : AddQueue(Taxi Driver)
7 : ConfirmRequest
opt Unavailability
8 : RequestUnavailability
9 : Update Queue
10 : Dequeue(Taxi Driver)
11 : Request Unavailability
12 : Update Queue
13 : Dequeue(Taxi Driver)

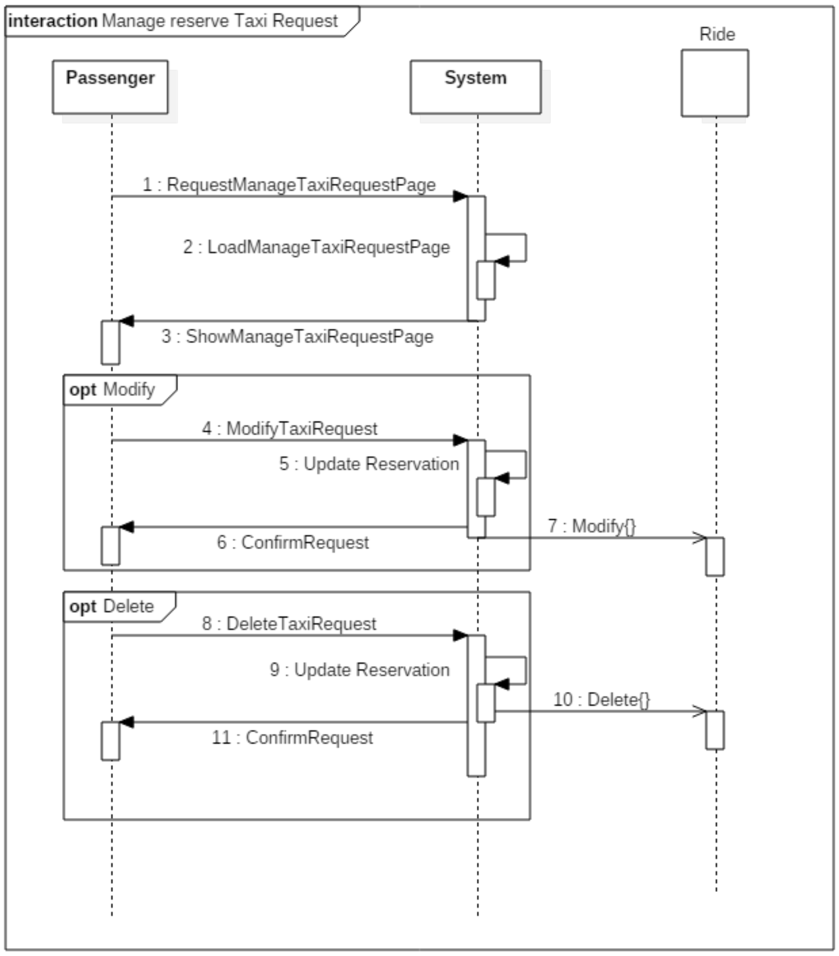
### 3.6.4 Taxi Request

| | |
|---|---|
| Actors | Customer, Taxi driver |
| Preconditions | The user should not be banned |
| Execution Flow | 1. The customer requests the *Taxi request* page<br>2. The system asks for the type of request that the user wants to issue<br>3. The customer fills the request form and sends the information to the system<br>4. The system forwards the request to the first taxi driver of the local queue<br>5. If the taxi driver answers positively to the request, he takes charge of the ride, otherwise he denies the request<br>6. If the taxi driver accepts the request, the system notifies to the customer the incoming taxi and changes the availability of the taxi driver; otherwise, the system updates the queue and forwards the request to the new first taxi driver of the queue.<br>7. If there are no taxis available, the system notifies so to the user. |
| Postconditions | If the request is accepted by a taxi driver, the customer is now a passenger |
| Exceptions | • The customer provides incorrect information in the request form<br>• The customer is not in a valid position (*e.g. outside the town*) |


interaction Taxi Request
Customer
System
TaxiDriver
Queue Manager
Ride
1 : AskTaxiRequestPage
2 :
3 : Fill the form
opt Taxi sharing
4 : fill Taxi sharing form
User must compiles the destination
opt Taxi booking
5 : fill Taxi booking form
6 : SendRequest
alt Reply
7 : ConfirmRequest
If User requests the reservation taxi , the system sends a request 10 min before the starting time
loop Until repeated queue or Taxi found
8 : SendRequest
9 : RejectRequest
10 : Update Queue
11 : DeQueue(Taxi Driver)
alt Taxi Driver Reply
12 : NoTaxiAvailable
13 : ConfirmRequest
14 : TaxiComing
15 : Update Queue
16 : New()
17 : DeQueue(Taxi Driver)
[Incorrect information]
18 : ShowErrorMessage
19 : OkButtonPressed

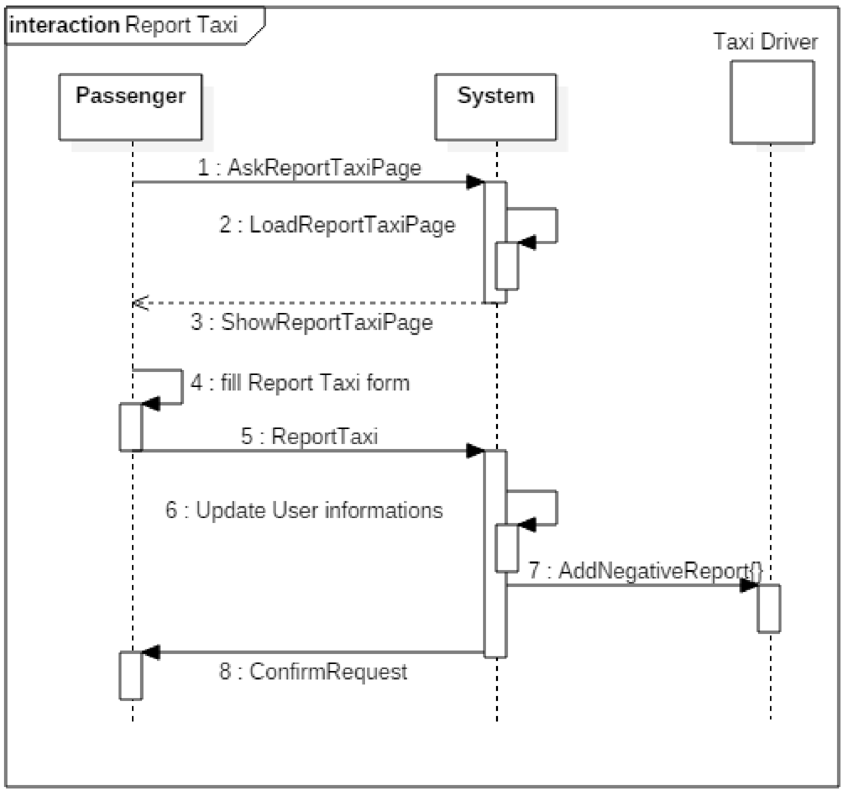
### 3.6.5 Manage *Reserve Taxi* Request

| | |
|---|---|
| Actors | Passenger |
| Preconditions | |
| Execution Flow | 1. The passenger requests the *Taxi request management* page<br>2. The page is generated by the system on the user's app<br>3. The passenger can modify the request by filling the *Modify request* form<br>4. The system modifies the request and returns a confirmation to the passenger<br>5. The passenger can delete the request, submitting the operation to the system<br>6. The system updates the queue and returns a confirmation to the passenger |
| Postconditions | • If the passenger chooses to modify the request, the request is updated<br>• If the passenger chooses to delete the request, the request is canceled |
| Exceptions | • The passenger provides incorrect information in the *Modify request* form<br>• The passenger cancel the request too late |


interaction Manage reserve Taxi Request
Passenger
System
Ride
1 : RequestManageTaxiRequestPage
2 : LoadManageTaxiRequestPage
3 : ShowManageTaxiRequestPage
opt Modify
4 : ModifyTaxiRequest
5 : Update Reservation
6 : ConfirmRequest
7 : Modify{}
opt Delete
8 : DeleteTaxiRequest
9 : Update Reservation
10 : Delete{}
11 : ConfirmRequest

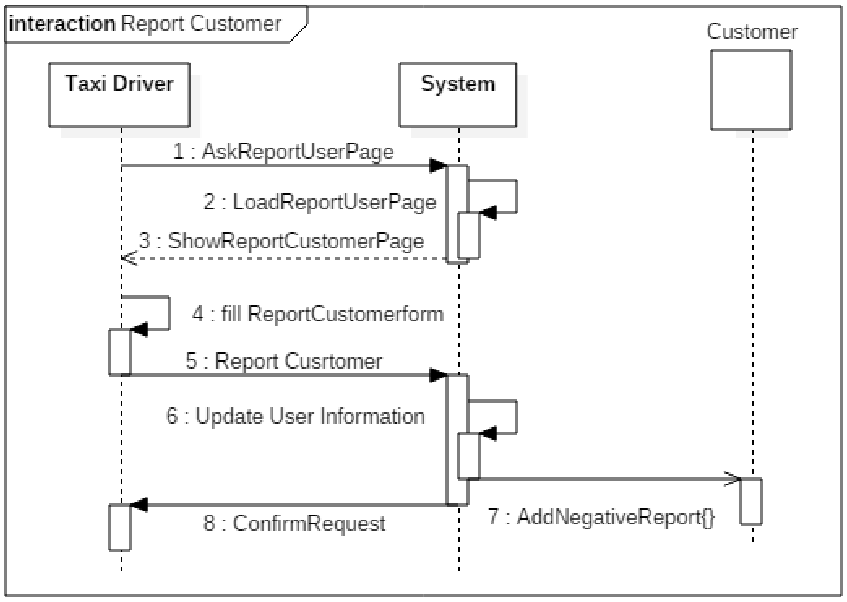
### 3.6.6 Report Taxi

| | |
|---|---|
| Actors | Passenger |
| Preconditions | |
| Execution Flow | 1. The passenger requests the *Report taxi* page<br>2. The page is shown on the customer's application<br>3. The passenger fills the form and submits the report<br>4. The system checks the submitted data<br>5. The system updates the taxi driver information<br>6. The system notifies to the passenger the success of the operation |
| Postconditions | The taxi driver is reported by the passenger |
| Exceptions | The passenger provides incorrect information in the report form |


interaction Report Taxi
Taxi Driver
Passenger
System
1 : AskReportTaxiPage
2 : LoadReportTaxiPage
3 : ShowReportTaxiPage
4 : fill Report Taxi form
5 : ReportTaxi
6 : Update User informations
7 : AddNegativeReport()
8 : ConfirmRequest

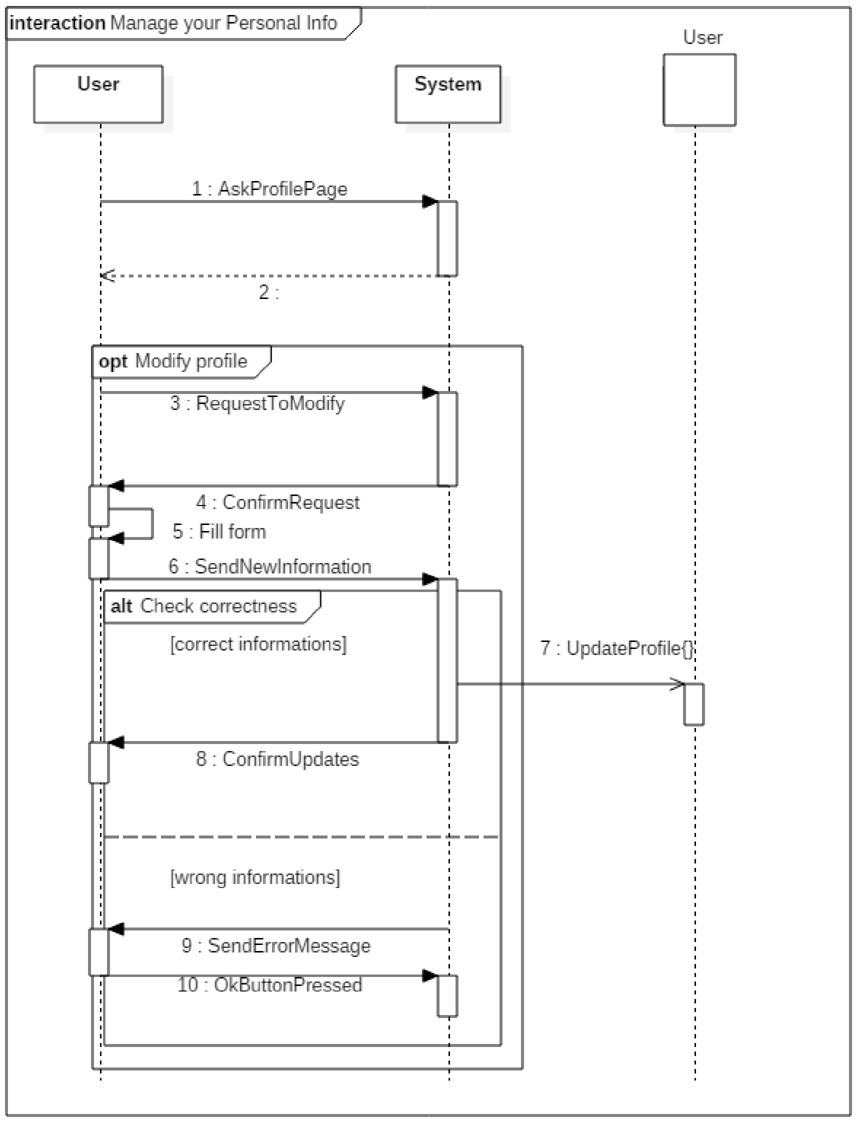
### 3.6.7 Report Customer

| | |
|---|---|
| Actors | Taxi driver |
| Preconditions | The interaction between customer and taxi driver must have happened at least 24 hours before |
| Execution Flow | 1. The taxi driver requests the *Report customer* page<br>2. The page is shown on the taxi driver's application<br>3. The taxi driver fills the form and submits the report<br>4. The system checks the submitted data<br>5. The system updates the user information<br>6. The system notifies to the taxi driver the success of the operation |
| Postconditions | The customer is reported by the taxi driver |
| Exceptions | The taxi driver provides incorrect information in the report form |



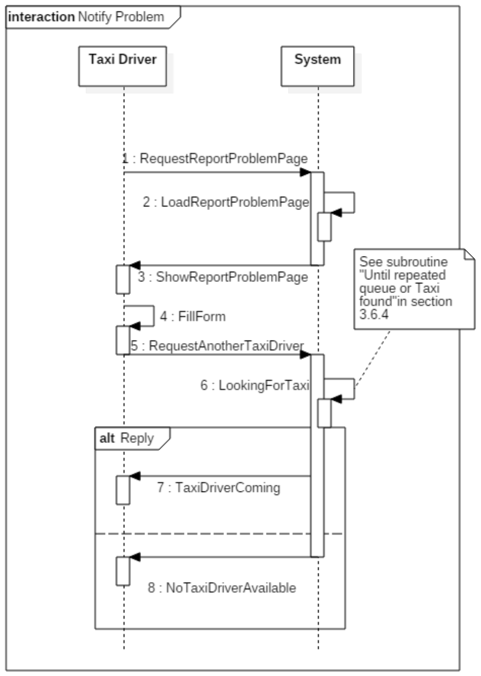
### 3.6.8 Manage Personal Information

| | |
|---|---|
| Actors | Customer or Taxi driver |
| Preconditions | |
| Execution Flow | 1. The user requests the their profile page<br>2. The user's personal information is shown on the user's application<br>3. The user can request to edit their profile<br>4. The system returns the editable information of the profile<br>5. The user can edit their information and send the changes to the system<br>6. The system performs a check on the new information<br>7. If the information is correct, a confirmation is sent back to the user; otherwise, an error messagge is shown. |
| Postconditions | If the user modifies their profile with correct information, the profile is changed |
| Exceptions | The user provides incorrect information |


interaction Manage your Personal Info
User
System
User
1 : AskProfilePage
2 :
opt Modify profile
3 : RequestToModify
4 : ConfirmRequest
5 : Fill form
6 : SendNewInformation
alt Check correctness
[correct informations]
7 : UpdateProfile()
8 : ConfirmUpdates
[wrong informations]
9 : SendErrorMessage
10 : OkButtonPressed

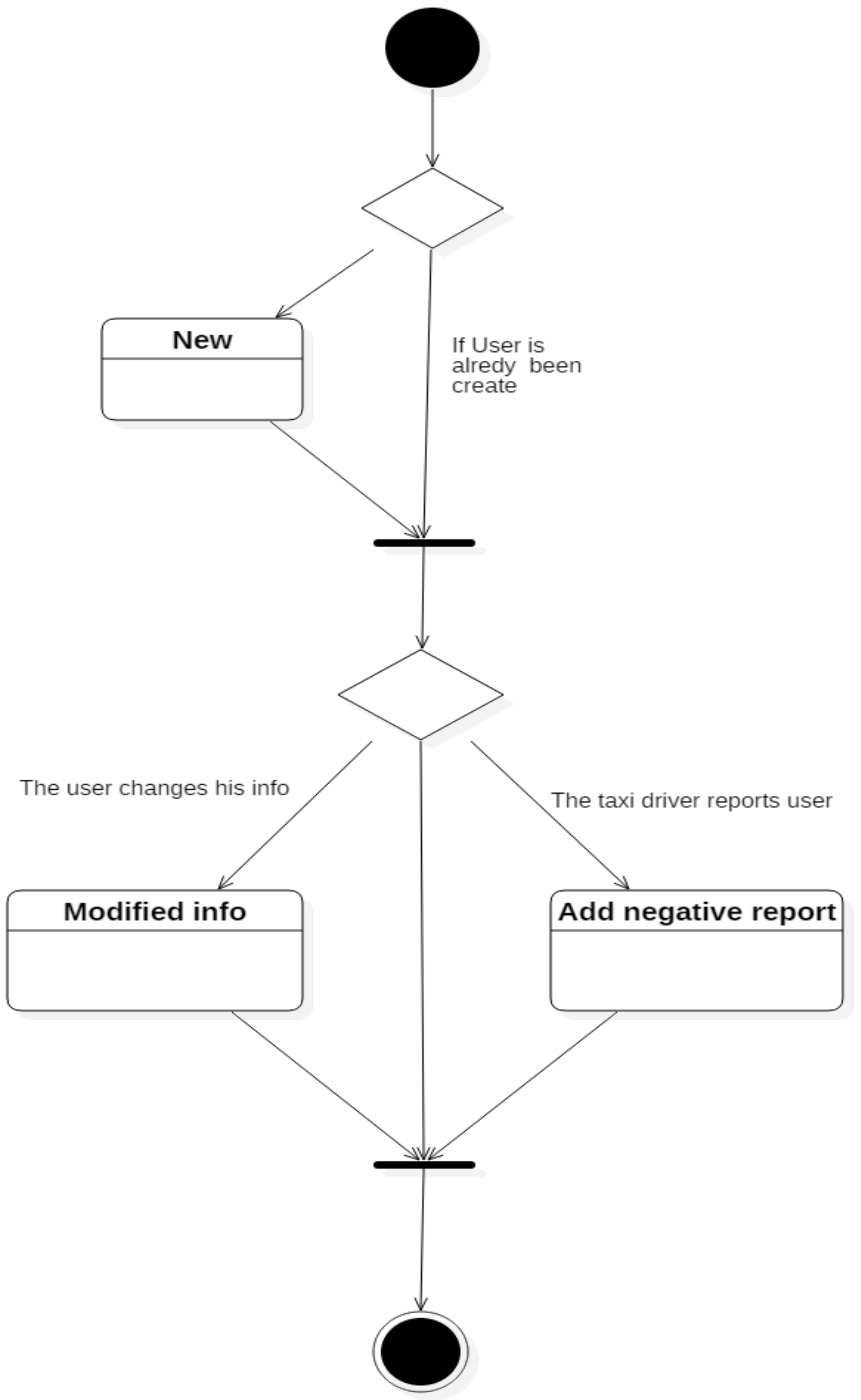
### 3.6.9 Report Problem

| | |
|---|---|
| Actors | Taxi driver |
| Preconditions | |
| Execution Flow | 1. The taxi driver requests the *Report problem* page<br>2. The page is shown on the taxi driver's application<br>3. The taxi driver fills the form and submits the information<br>4. If the taxi driver has a passenger on board, they can request another taxi to drive the passenger to their destination<br>5. If the taxi driver requests another taxi the system looks for an available taxi driver<br>6. If an available taxi driver is found, a notification is sent to both drivers.<br>7. If the new taxi driver accepts the ride, a confirmation is sent to the driver who is submitting the report; otherwise, the systems looks for other taxi drivers until someone accepts the ride, and returns a failure notification otherwise. |
| Postconditions | The problem is reported to the system |
| Exceptions | • The taxi driver is located in a invalid zone<br>• The taxi driver fills the form with incorrect information |

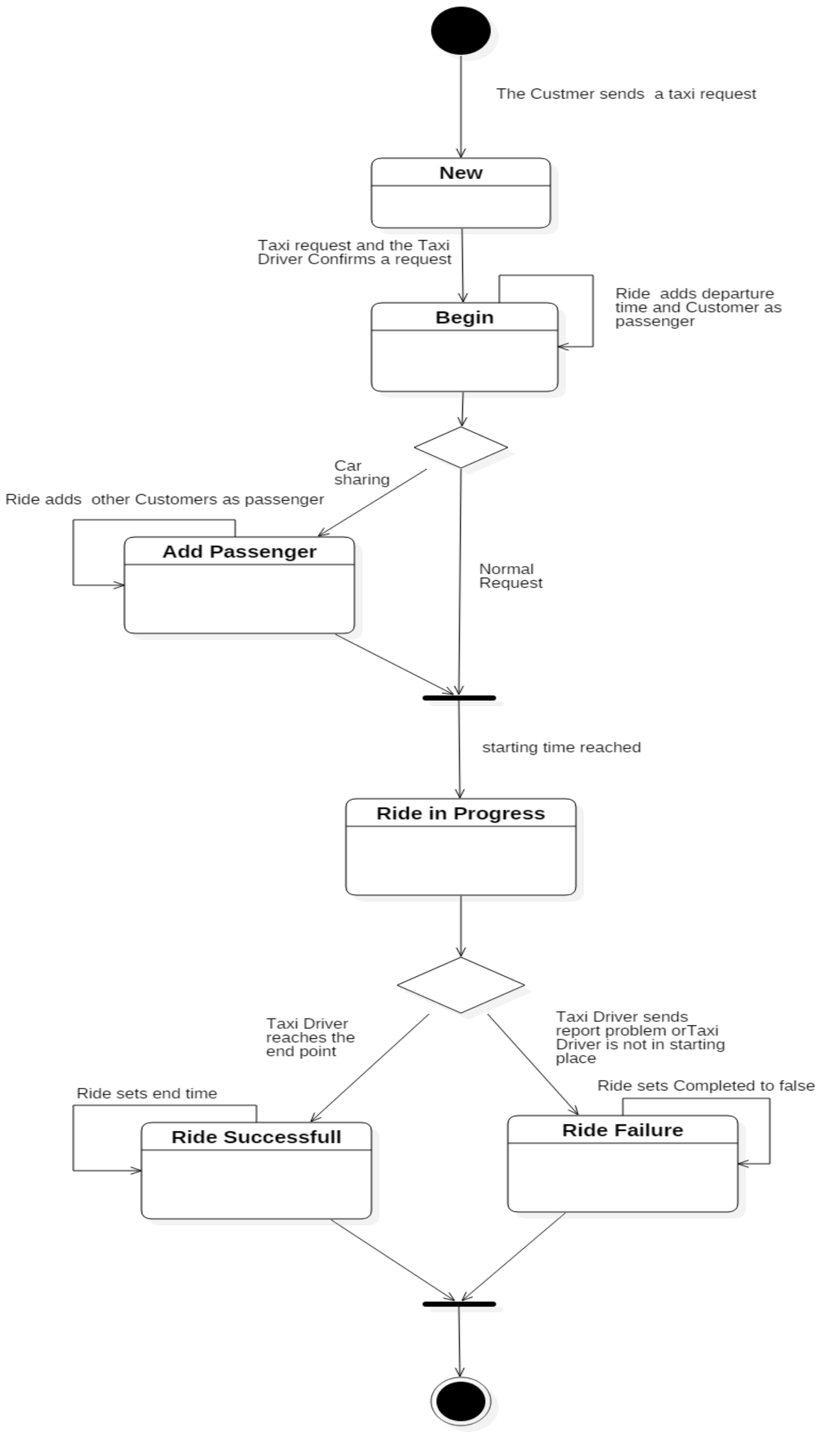

interaction Notify Problem
Taxi Driver
System
1 : RequestReportProblemPage
2 : LoadReportProblemPage
3 : ShowReportProblemPage
4 : FillForm
5 : RequestAnotherTaxiDriver
6 : LookingForTaxi
See subroutine "Until repeated queue or Taxi found"in section 3.6.4
alt Reply
7 : TaxiDriverComing
8 : NoTaxiDriverAvailable

## 3.7 Entities Behavior

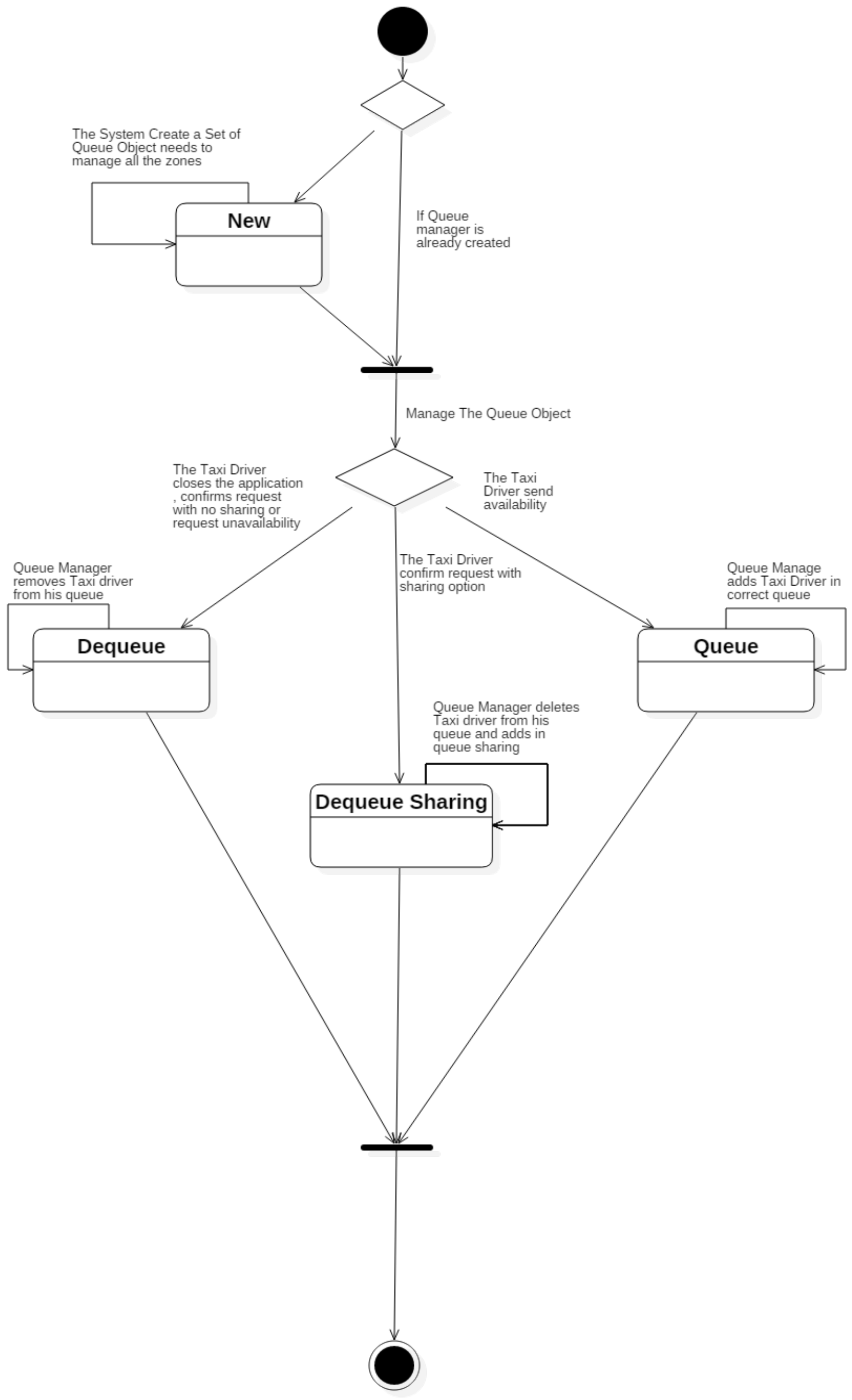
### 3.7.1 User


New
If User is alredy been create
The user changes his info
The taxi driver reports user
Modified info
Add negative report

### 3.7.2 Ride


The Custmer sends  a taxi request
New
Taxi request and the Taxi Driver Confirms a request
Begin
Ride  adds departure time and Customer as passenger
Car sharing
Ride adds  other Customers as passenger
Add Passenger
Normal Request
starting time reached
Ride in Progress
Taxi Driver reaches the end point
Taxi Driver sends report problem orTaxi Driver is not in starting place
Ride sets end time
Ride sets Completed to false
Ride Successfull
Ride Failure

### 3.7.3 Queue Manager


The System Create a Set of Queue Object needs to manage all the zones
New
If Queue manager is already created
Manage The Queue Object
The Taxi Driver closes the application , confirms request with no sharing or request unavailability
The Taxi Driver send availability
The Taxi Driver confirm request with sharing option
Queue Manager removes Taxi driver from his queue
Dequeue
Queue Manage adds Taxi Driver in correct queue
Queue
Queue Manager deletes Taxi driver from his queue and adds in queue sharing
Dequeue Sharing

## 3.8 Performance requirements

- The system should support at least all the taxi driver registered into the Town Database
- The system should elaborate user incoming information
- The system should provide the faster path for each ride
- The system should calculate the final price of the ride with a maximum error of 10%
- The system should provide to a passenger the arrival time (maximum error 20%) and change the path in case of traffic

## 3.9 Logical database requirements

- The system databases has consistency check, data integrity check

## 3.10 Design constraints

- GPS precision limitations: average 3m error
- Internet congestion

## 3.11 Standards compliance

## 3.12 Software system attributes

## 3.13 Availability and Reliability

Since MyTaxiService is a service-oriented platform, its reliability parameters directly relate to its availability parameters. The platform's ability to function under the stated conditions is indeed its ability to respond to users' requests at any given time, hence the strict relation between the two. It has been decided to treat the two aspect as one, and the related non functional requirements are listed in this section.

1. The platform's services must be available to the users 24/7.
2. The RTO parameter must be kept at minimal levels (less than 1 minute) at any given time.
   (a) Mission critical data must be locally mirrored on fast hardware (e.g. stored in RAID1 arrays with flash storage).
3. The RPO parameter must be kept at minimal levels (less than 10 seconds) at any given time.
   (a) Any data must be locally stored in a 10 second time frame from its creation.

   (b) Any locally stored data must be locally and remotely mirrored in a 1 minute time frame from its memorization.

4. Data integrity checks must be periodically performed between the main data storage unit and the secondary backups, in order to ensure the success of disaster recovery operations.

5. The implementation of the platform must prefer the absence of service to an incorrect or unsound one.

   (a) No data exchanges must happen during the disaster recovery operations.

   (b) Data stored in memory in the event of a system failure or security breach must be considered corrupt and no attempts must be made at recovering it.

## 3.14 Security

The following non functional requirements cover the security aspects of the platform in order, among other reasons, to satisfy the C3 constraint in section ??.

1. Access to the user data through the intended applications must be password protected.

   (a) A ban system must exists to prevent brute-forcing of the users' passwords.

2. Sensitive user data (like passwords) must be stored under at least one encryption layer, after having been *salted*. This applies to secondary storage, too.

   (a) Decryption of the above mentioned data must happen exclusively at runtime and the *cleartext* information must never be sent through any communication channels.

3. Operations on the platform must be performed exclusively by logged users (with the exception of the guest registration).

4. HTTP data exchanges between the back-end and the user-side applications must be encrypted with a recognized SSL certificate (HTTPS protocol).

5. Access to the back-end system must be protected both via hardware and software means.

   (a) A physical firewall must exists between the Internet and the back-end main router.

(b) Access to the system must be enabled via IP address whitelisting, rather than blacklisting.

(c) Root login must be disabled for remote sessions.

(d) Password login must be disabled and signed PKA must be enforced, for any type of session.

(e) Access logs must be kept, backed up, and regularly analyzed.

6. Mission critical data must be stored with a particular attention to data integrity.

## 3.15 Maintainability

The following non functional requirements are meant as a small guideline for programmers and designers in the development phase.

1. The codebase for all developed software must be highly modular to facilitate possible changes in the platform's functions and possible integration with other systems; this applies especially to the back-end modules.

2. The codebase for all developed software must be thoroughly documented with both in-code comments and official documentation, in order to facilitate a possible outsourcing of the maintenance phase.

## 3.16 Portability

The following non functional requirements consider technical details of the platform's implementation in order to analyze its portability requirements.

When seen as a whole, the platform consists mainly of its user-side applications, and the back-end accounts for about 25% of the codebase; nonetheless, since the user-side applications are strictly OS dependent, as specified in section **??**, portability is an issue which has to be tackled in back-end development, in order to keep costs to a minimum in the case of possible changes in the platform (e.g. an integration with a pre-existsing system). Therefore:

1. The back-end software must be developed in Java Enterprise Edition.

2. Integration with support modules must happen through JEE libraries.

3. Any system related calls, communication protocols and thread related calls in the back-end must be OS independent (the use of wrapper libraries is encouraged over a case-by-case analysis).