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.

//MOCKUPS <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

3.2 Functional requirements

The following set of functional requirements must be kept as official reference for all further phases of development of the platform.

These requirements are meant as a more in-depth description of the main system functionalities listed in section **??**, and must be considered alongside the domain assumptions in section **??**.

3.2.1 Back-end requirements

1. The system must act as the central point of communication between taxi drivers and customers (functions F1 and F4).

    (a) The system must forward the customers' requests to taxi drivers.

    (b) The system must forward the taxi drivers' replies (in response to requests of point 1.(a)) to customers.

    (c) The system must manage the exceptions that may happen in the flow of events (see section 3.6 for more specific description of these exceptions)

2. The system must manage the customers taxi reservation requests and forward them to taxi drivers when necessary. (functions F1 and F4).

3. The system must manage the taxi queues associated to the different zones of the city (function F9).

   (a) The application must keep track of the availability statuses of all taxi drivers (see also requirement 3 of section 3.2.3) and manage the queues accordingly.

4. The system must manage the operational databases.

   (a) The system must store user operation critical data (username, password, e-mail address, name, date of birth).

   (b) The system must keep track of changes in user information when prompted (functions F2 and F5).

   (c) The system must store user records submitted through the *Report* functions (functions F3 and F6).

5. The system must perform the necessary validation and consistency checks on any data it handles in order to ensure the functional and non functional requirements listed in the following sections.

3.2.2 Generic user application requirements

1. The application must act as external interface between users and the back-end.

   (a) The application must enable users to register into the platform.

   (b) The application must enable users to log into the platform.

   (c) The application must provide users with an interface to manage their personal information (functions F2 and F5).

2. The application must display confirmations and error messages forwarded by the back-end (see requirement 1 of section 3.2.1, and section 3.6).

3. The application must provide all the implementation-specific requirements listed in the following sections.

3.2.3 Taxi-side application specific requirements

1. The application must notify the taxi driver when a request from a customer is forwarded by the back-end (function F4).

2. The application must show the taxi driver's position in their local queue.

3. The application must enable the taxi driver to toggle their availability status between *available* and *unavailable*.

4. The application must enable the taxi driver to submit *Customer* reports (function F6).

(a) The application must require information about the ride if the information is not directly deductible from the known data.

5. The application must enable the taxi driver to submit *Technical problem* reports (function F7).

   (a) The application must require information about the technical problem before submitting the report to the system.

   (b) The application must require the taxi driver to state whether they want a replacement to complete the ride (if the technical problem happens when a passenger is on board) before submitting the report to the system.

3.2.4 Customer-side application specific requirements

1. The application must enable the customer to request the services of a taxi. (function F1).

   (a) The application must enable the customer to request a taxi in the very short term (service is exploited as soon as the taxi is available).

   (b) The application must enable the customer to reserve a taxi in advance.

      i. The application must require the customer's destination before submitting this kind of request to the system.

   (c) The application must enable the customer to select the *Taxi sharing* option (it allows users with similar routes at the same hour to share the taxi and split the fee)

      i. The application must require the customer's starting point and destination before submitting this kind of request to the system.

   (d) The application must enable the customer to select both options of points 1.(b) and 1.(c) of this section at the same time.

2. The application must enable the customer to submit a *Taxi* reports (function F3).

   (a) The application must require information about the ride if the information is not directly deductible from the known data.

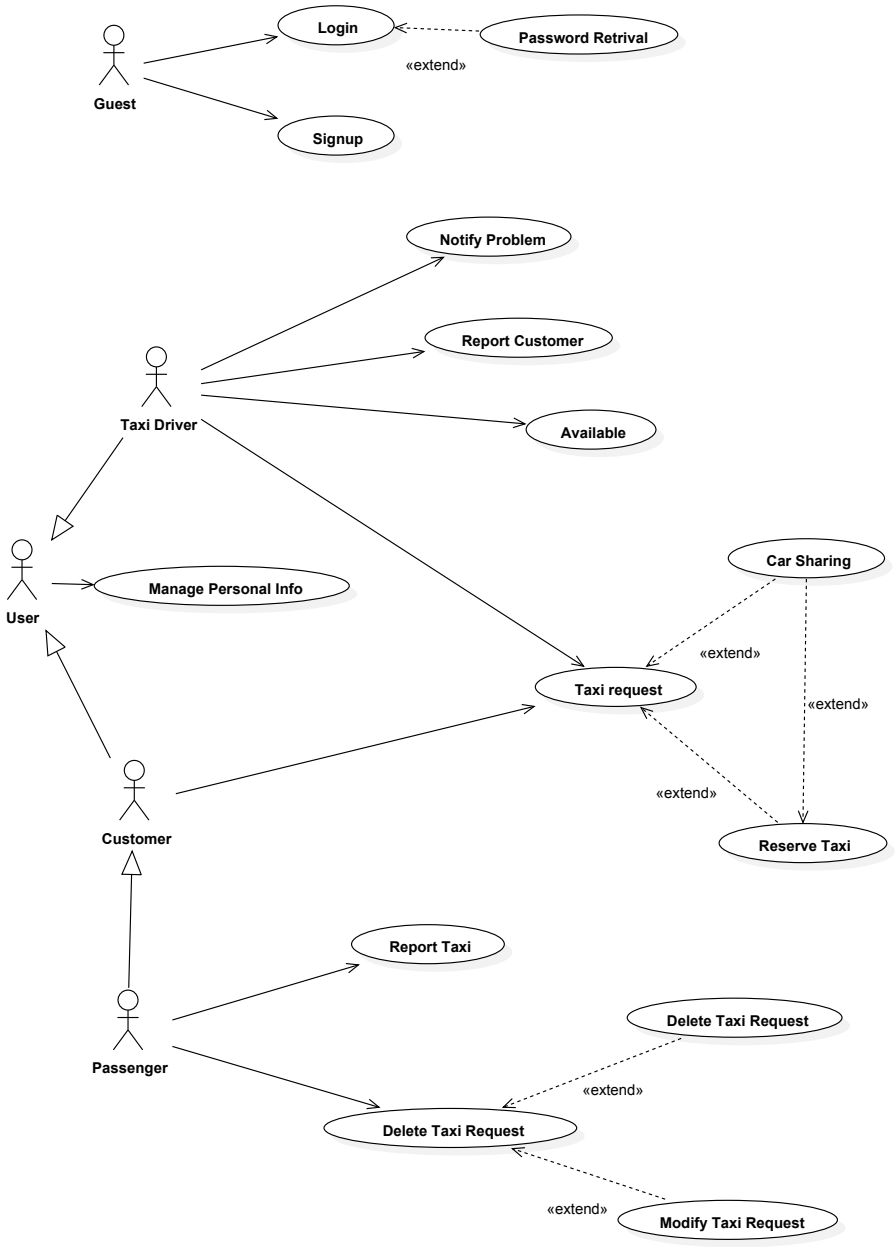
3.2.5 Functional constraints

1. Taxi reservations must can only be issued at least 2 hours before the requested time of the ride (function F1, associated to req. 3.2.4 1.(b)).

2. Requests associated to taxi reservations must be forwarded to taxi drivers exactly 10 minutes before the requested time (function F9, associated to req. 3.2.4 1.(b)).

3. *Taxi* and *Customer* reports must be accepted only on condition that the ride happened in a 24 hours time frame from the submission request functions (functions F3 and F6).

3.3 Use Cases



3.4 Logical database requirements

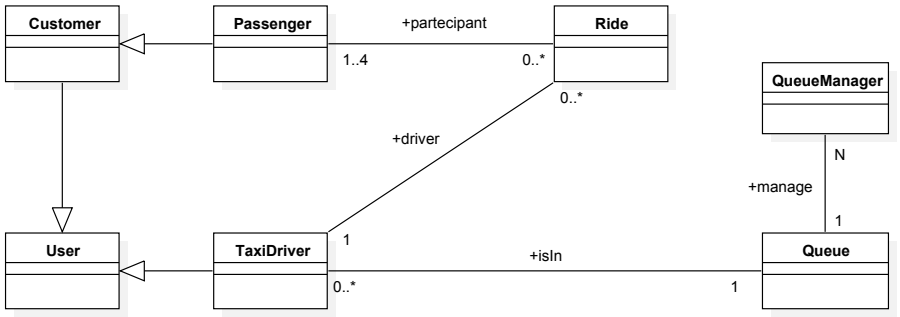
The majority of data processed by the DMBS consists of Java *String* and *Date* objects. Images and complex objects might be stored in the database, too, and must therefore be considered during the design of the database.

The majority of stored data is not frequently accessed as most of the operational variables are strictly mutable (users' positions, queues, non-booked rides) and therefore not in need to be stored.

Stored data might consist of stable personal information of the users and, while not immutable, it can be considered as rarely accessed.

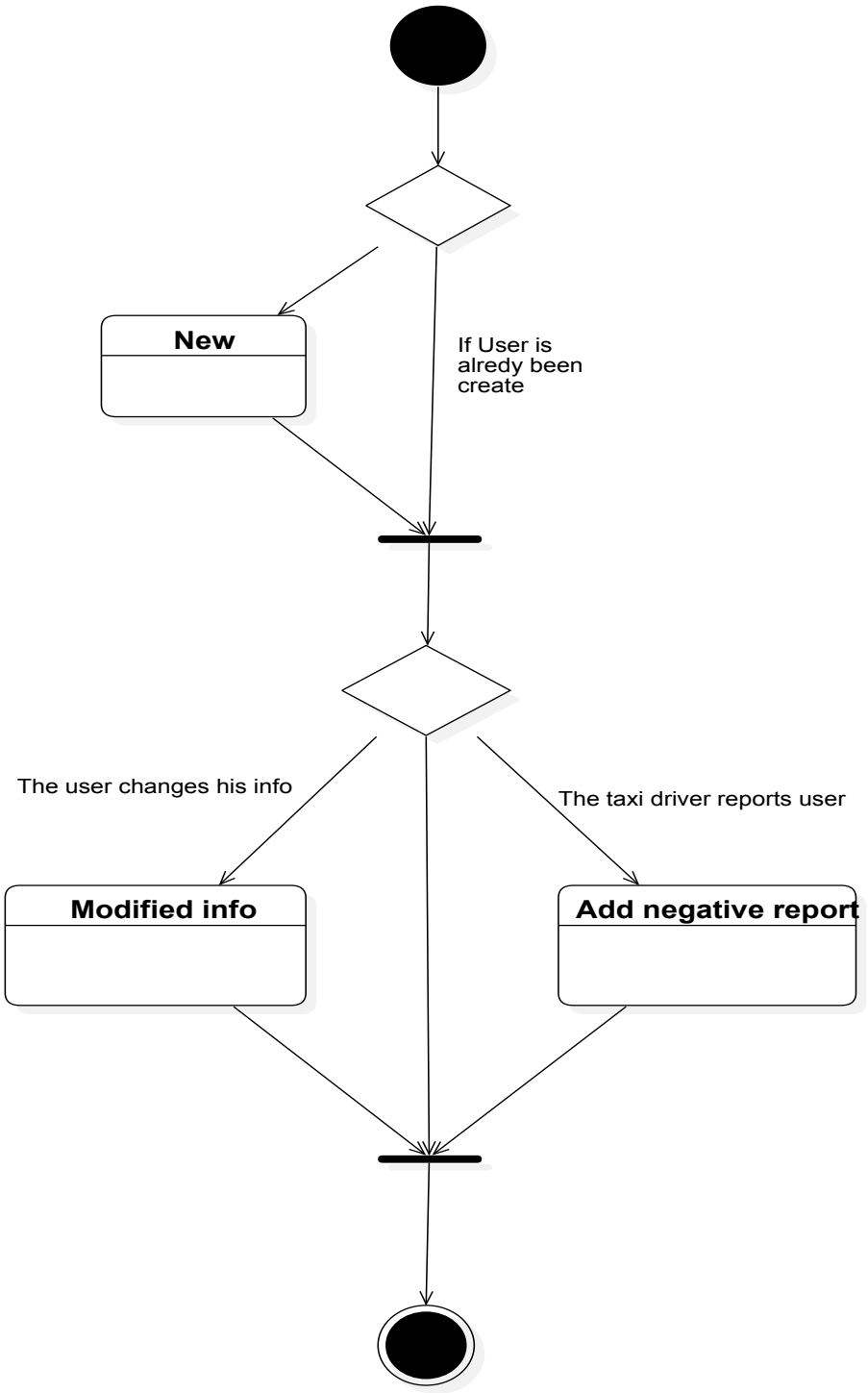
A deeper insight on the data classes and operational variables is found in the next sections of this document, so it is advised to keep this section only as a general reference and to approach the design of the database by looking at the more specific descriptions given there.

3.4.1 Class diagram

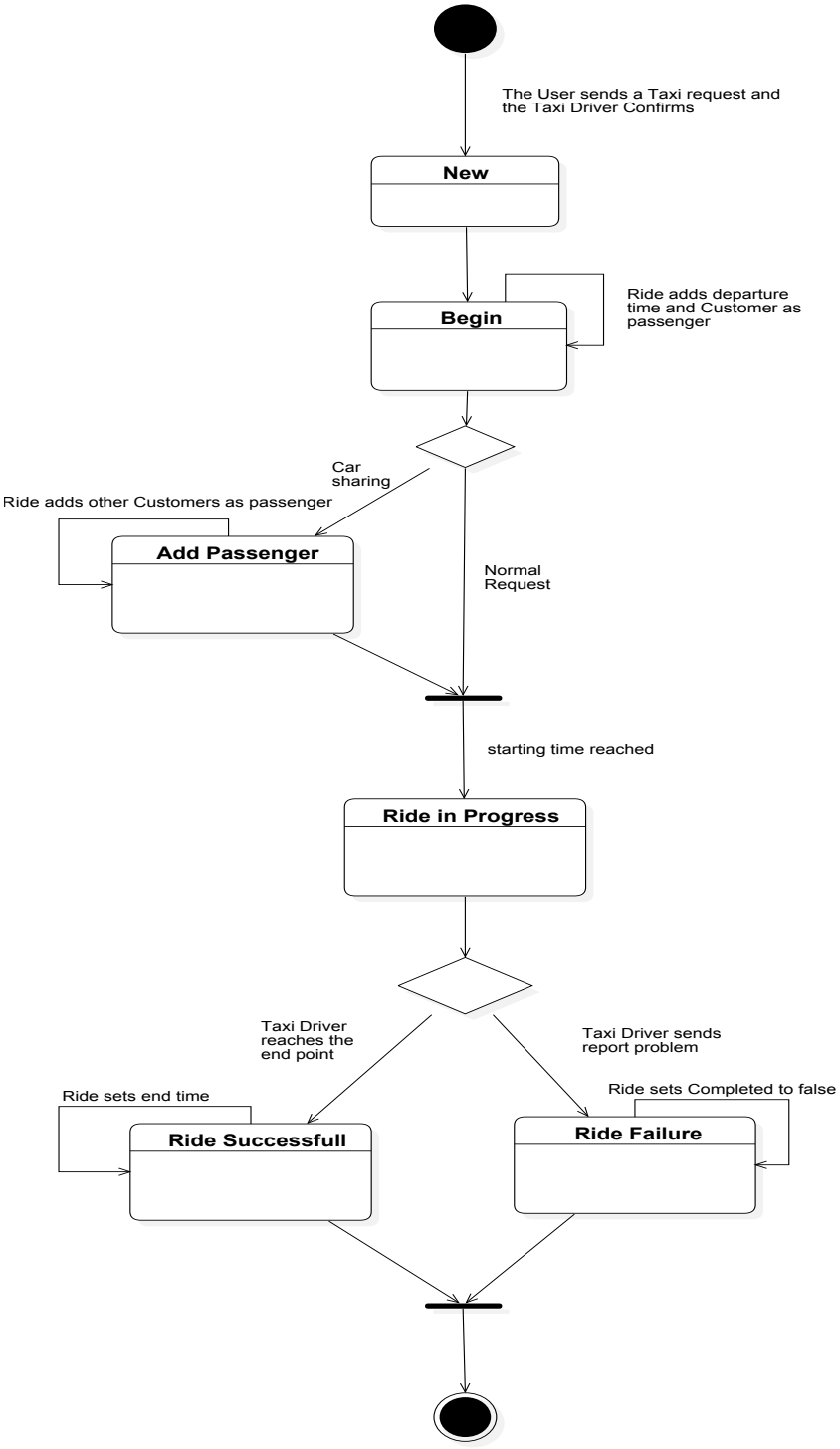


3.4.2 Entity Behaviors (State chart diagrams)

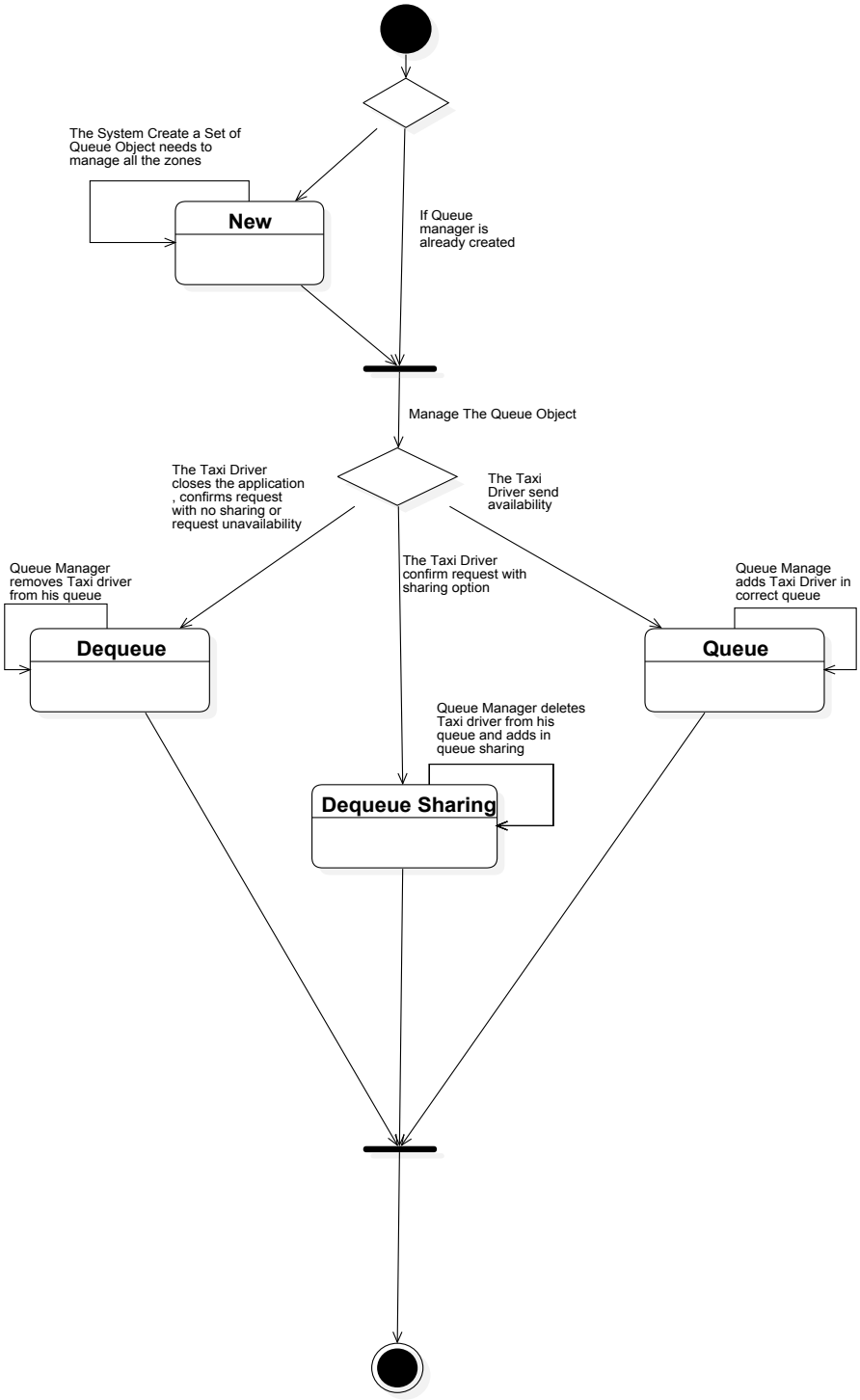
[SC1] User


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

[SC2] Ride


The User sends a Taxi request and the Taxi Driver Confirms
New
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
Ride sets end time
Ride sets Completed to false
Ride Successfull
Ride Failure

[SC3] 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
Queue Manager removes Taxi driver from his queue
The Taxi Driver confirm request with sharing option
Queue Manage adds Taxi Driver in correct queue
Dequeue
Queue
Queue Manager deletes Taxi driver from his queue and adds in queue sharing
Dequeue Sharing

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.

Ten minutes before 6PM the system will forward Adam's request to the first taxi in the queue at that time, and similarly to the previous scenario a taxi will be assigned to Adam.

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; after checking the option, the apps automatically adds a field in the reservation form in which they must specify their intended destination; they then submit their requests.

The system performs a check on Michelle and Joanne's requests and, since their routes match, it automatically assign them to the same taxi.

Based on whether they decided to book the taxi or simply request it, the taxi will be chosen similarly to scenarios 3.5.5 or 3.5.4 respectively.

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 possibly modifies to time on which to forward the request to the queue.

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 record 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 record 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

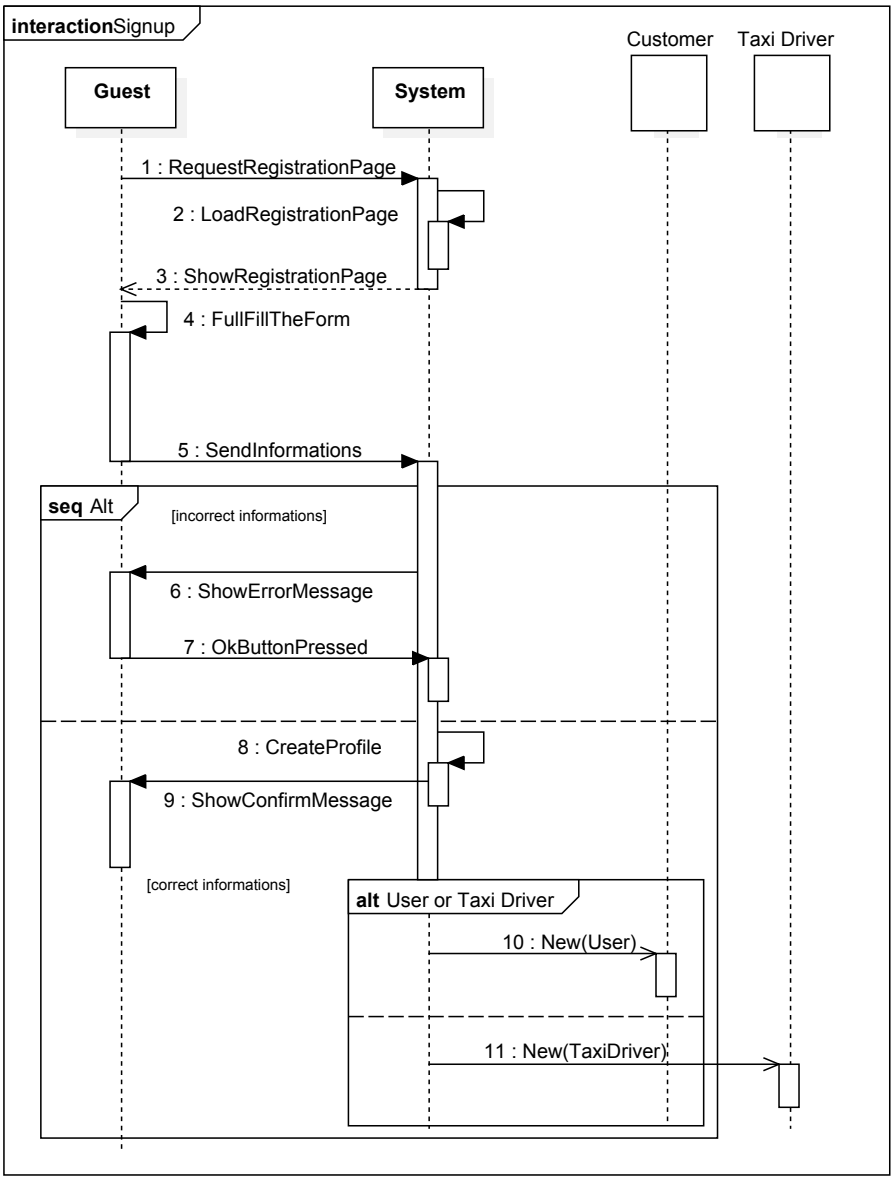
During a ride, Hector has a problem with his taxi's engine and can't bring Joanne to her destination.

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

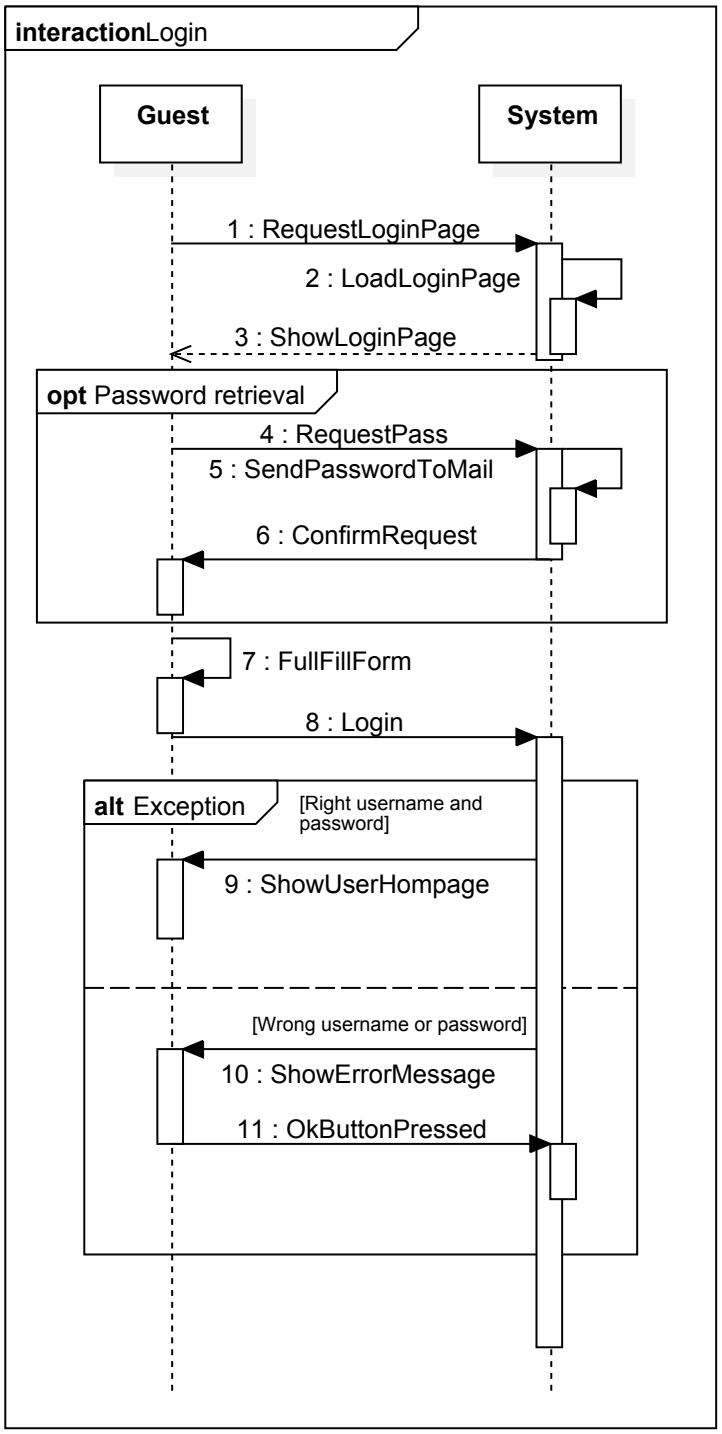
3.6.1 Sign-up

Actors	??Guest
Preconditions	The guest is not registered into the system.
Execution Flow	1. The guest requests the registration page. 2. The guest fills the registration form and submits the request. 3. The system checks the uniqueness of the username and e-mail. 4. The system creates the customer (or taxi driver) profile . 5. 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: an error message is shown.


interactionSignup
Customer
Taxi Driver
Guest
System
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)

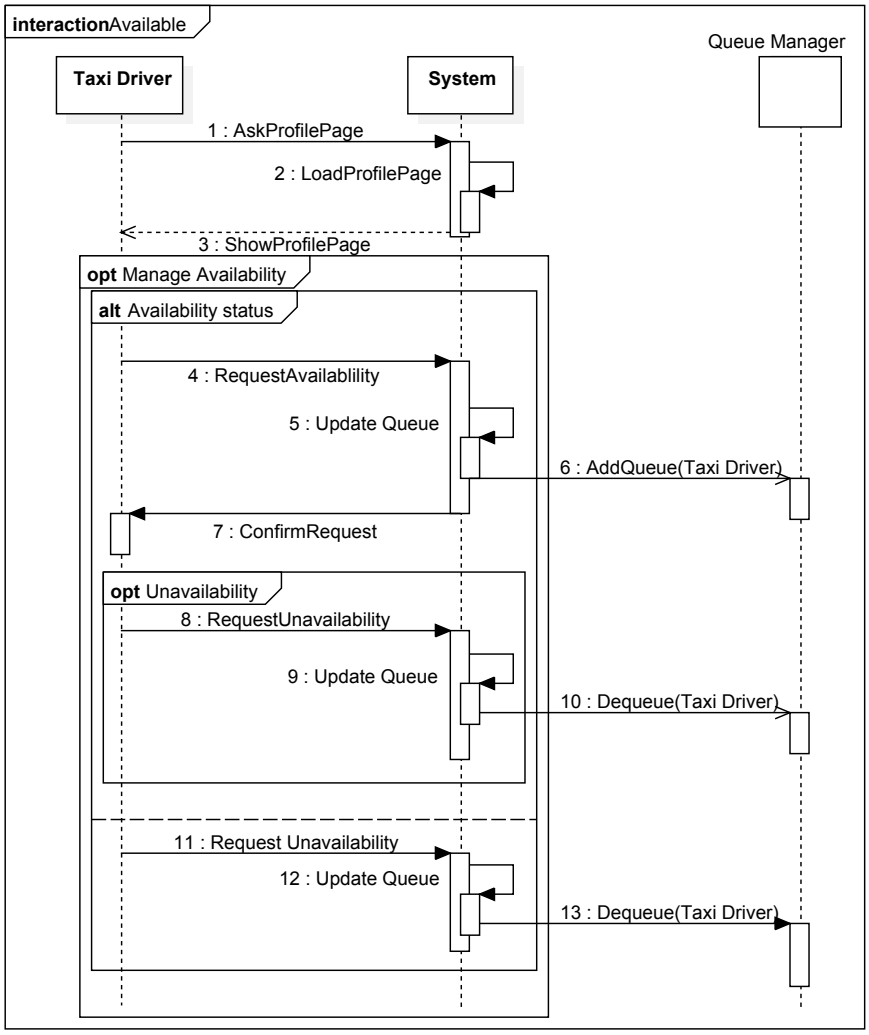
3.6.2 Login

Actors	Guest
Preconditions	The guest is already registered into the system.
Execution Flow	1. The guest requests the login page. 2. The guest fills the form and submits the request. 3. The system checks the username and password. 4. The system sends a login confirmation. 5. The guest is logged into the system. 6. The guest is redirected to the user main page.
Postconditions	The guest is now a logged-in user.
Exceptions	The username-password combination is incorrect, so the guest cannot log in: an error message is shown.


interactionLogin
Guest
System
1 : RequestLoginPage
2 : LoadLoginPage
3 : ShowLoginPage
opt Password retrieval
4 : RequestPass
5 : SendPasswordToMail
6 : ConfirmRequest
7 : FullFillForm
8 : Login
alt Exception
[Right username and password]
9 : ShowUserHompage
[Wrong username or password]
10 : ShowErrorMessage
11 : OkButtonPressed

3.6.3 Available

Actors	Taxi driver
Preconditions	The taxi driver is logged into the system.
Execution Flow	1. The taxi driver opens the app. 2. The taxi driver changes their availability by tapping a button: a request is sent to the system. 3. The system updates the queue. 4. The system returns a confirmation to the taxi driver.
Postconditions	The taxi driver has now changed their availability.
Exceptions	• The taxi driver is located in a invalid zone and they try to become *available*: an error message is shown.



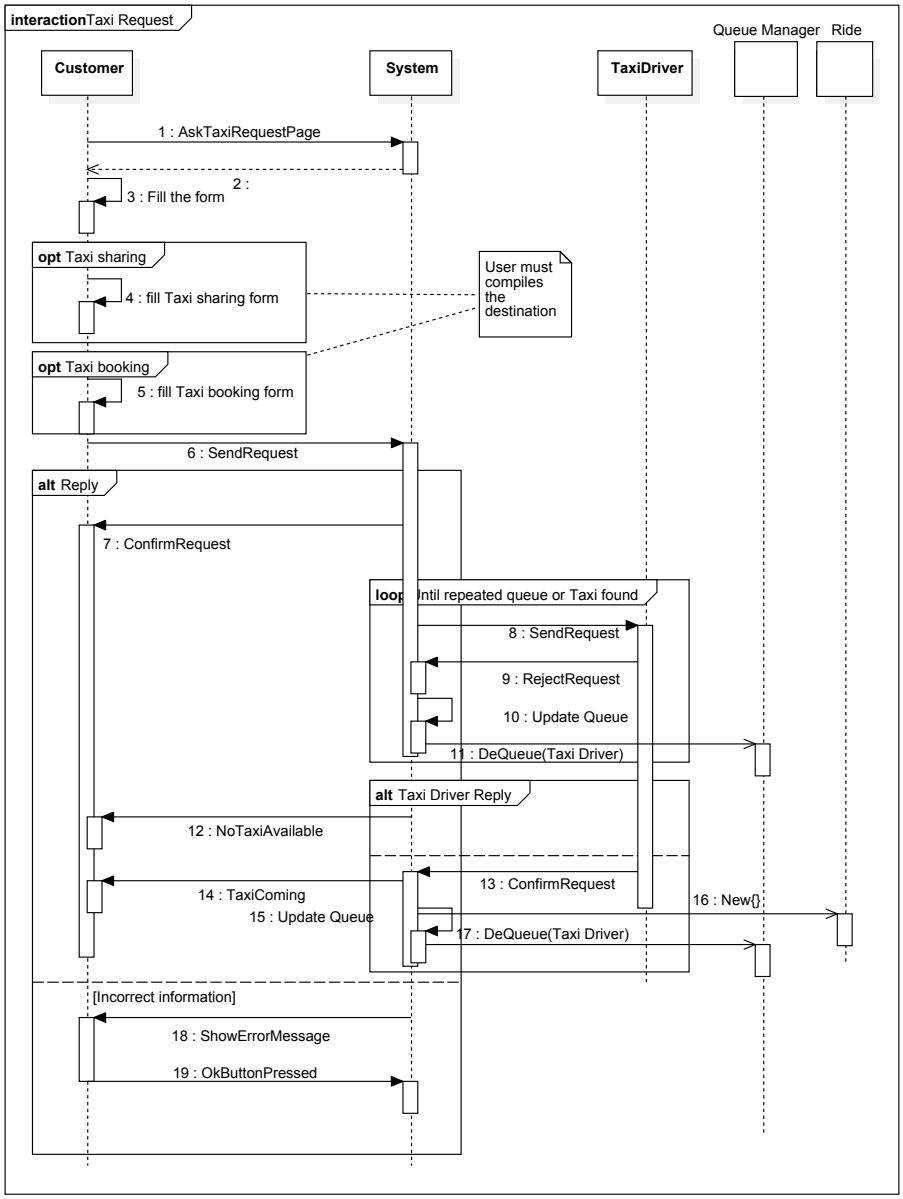
3.6.4 Taxi Request

Actors	Customer, Taxi driver
Preconditions	Both users must be logged in, the taxi driver must be available.

Execution Flow

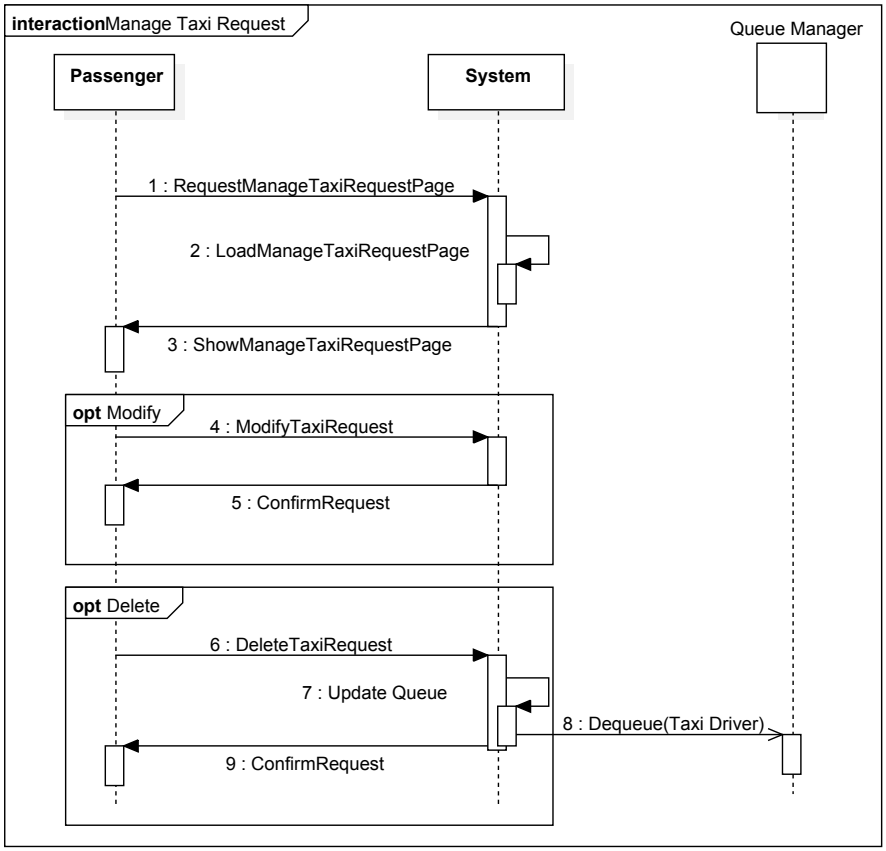
1. The customer requests the *Taxi request* page
2. The customer is shown a form to fill with optional information: they can choose to book a taxi and to enable the *Taxi sharing* option.
3. The customer fills the request form according to their preference and sends the information to the system.
4. Based on the type of request that the customer issued, the system generates a request with all the needed data and sends it to the first taxi driver in the right local queue, at the right time.
5. The taxi driver can either ignore the request or accept it.
6. If the taxi driver accepts the request, the system notifies to the customer the incoming taxi (with an approximate ETA) and changes the availability of the taxi driver; otherwise, the system puts the taxi driver at the end of the queue and forwards the request to the next first taxi driver of the queue.
7. If the issued request was a booking request or a request with *Taxi sharing* enabled, the system calculates the estimated fee for the passenger and adds it to the notification sent 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: an error notification is shown. • No taxis are available: the system notifies so to the user. • The customer is not in a valid position (*e.g. outside the town*): an error notification is shown.


interactionTaxi 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
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

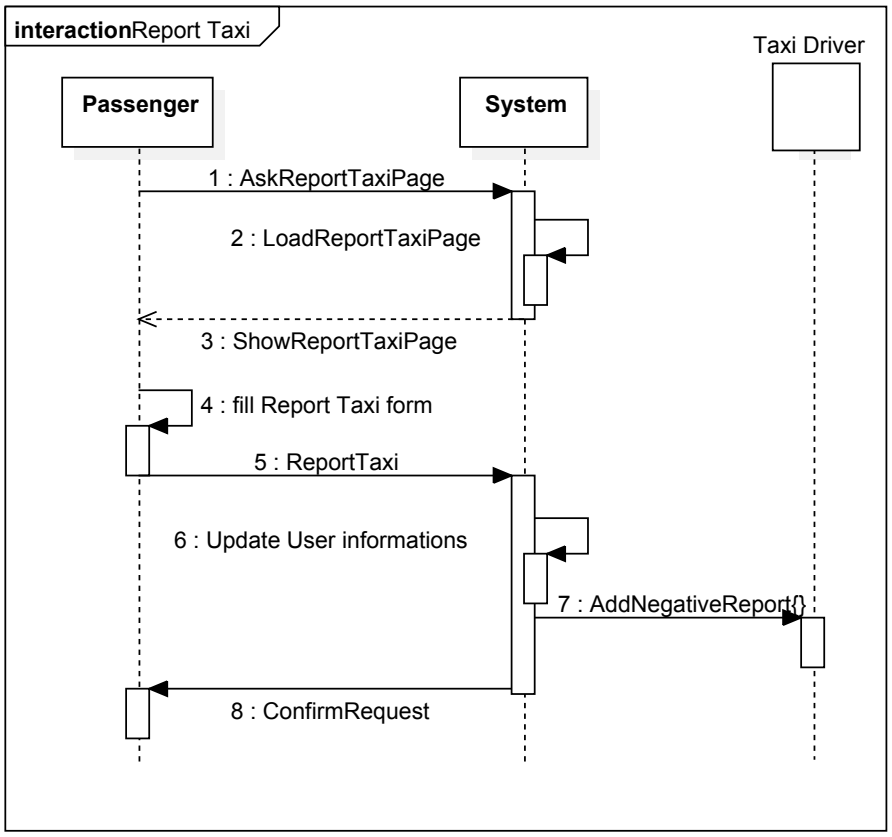
3.6.5 Manage *Reserve Taxi* Request

Actors	Customer
Preconditions	• The customer must have reserved a taxi. • The customer must be logged in.
Execution Flow	1. The customer requests the *Taxi request management* page. 2. The customer can modify the request by filling a form and submitting it, or delete the request by tapping (or clicking) a button. 3. The system modifies the request and returns a confirmation to the passenger. 4. The request is forwarded to the right local queue accordingly; if the user canceled their request, the request is not sent.
Postconditions	• If the customer chooses to modify the request, the request is updated. • If the customer chooses to delete the request, the request is canceled.
Exceptions	• The customer provides incorrect information in the *Modify request* form: an error message is shown. • The customer tried to cancel the request when the request has already been forwarded to the taxi driver: an error message is shown and the modification is not allowed.


interactionManage Taxi Request
Passenger
System
Queue Manager
1 : RequestManageTaxiRequestPage
2 : LoadManageTaxiRequestPage
3 : ShowManageTaxiRequestPage
opt Modify
4 : ModifyTaxiRequest
5 : ConfirmRequest
opt Delete
6 : DeleteTaxiRequest
7 : Update Queue
8 : Dequeue(Taxi Driver)
9 : ConfirmRequest

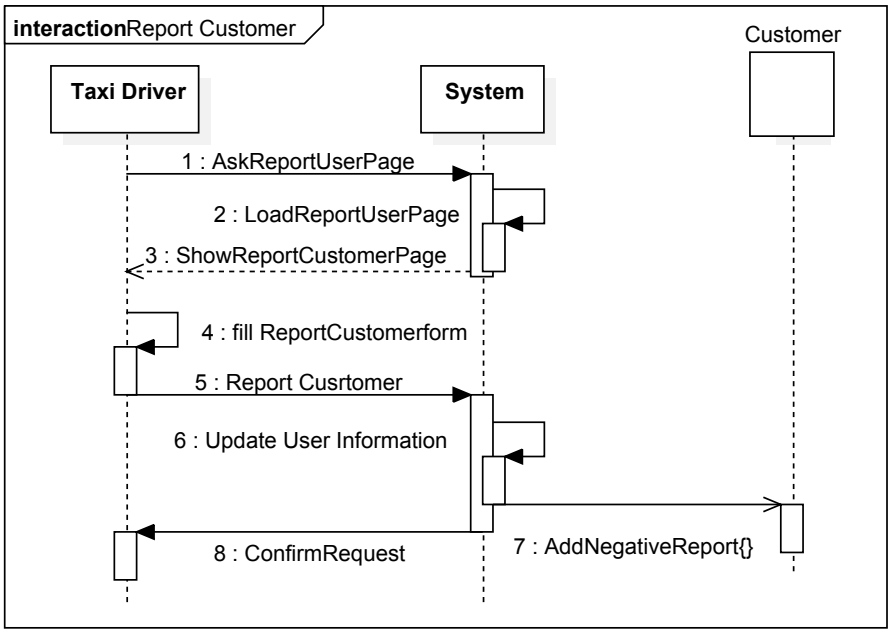
3.6.6 Report Taxi

Actors	Passenger
Preconditions	• The interaction between the passenger and the taxi driver must have happened at most 24 hours before. • The passenger must be logged in.
Execution Flow	1. The passenger requests the *Report taxi* page. 2. The passenger fills the form and submits the report. 3. The system checks the submitted data. 4. The system updates the taxi driver's record. 5. 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: an error message is shown.


interactionReport 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

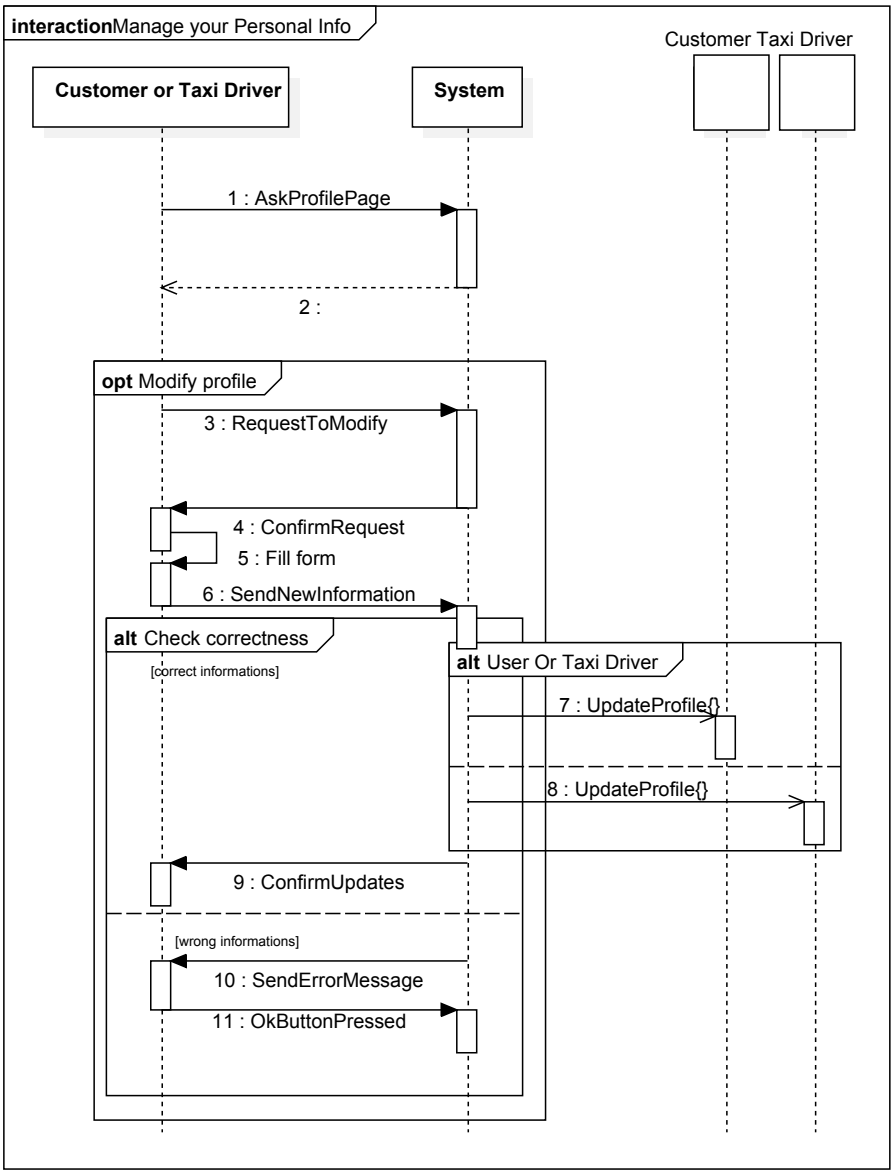
3.6.7 Report Customer

Actors	Taxi driver
Preconditions	• The interaction between the passenger and the taxi driver must have happened at most 24 hours before. • The taxi driver must be logged in.
Execution Flow	1. The taxi driver requests the *Report customer* page. 2. The taxi driver fills the form and submits the report. 3. The system checks the submitted data. 4. The system updates the custormer's record. 5. 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: an error message is shown.


interactionReport Customer
Customer
Taxi Driver
System
1 : AskReportUserPage
2 : LoadReportUserPage
3 : ShowReportCustomerPage
4 : fill ReportCustomerform
5 : Report Cusrtomer
6 : Update User Information
7 : AddNegativeReport{}
8 : ConfirmRequest

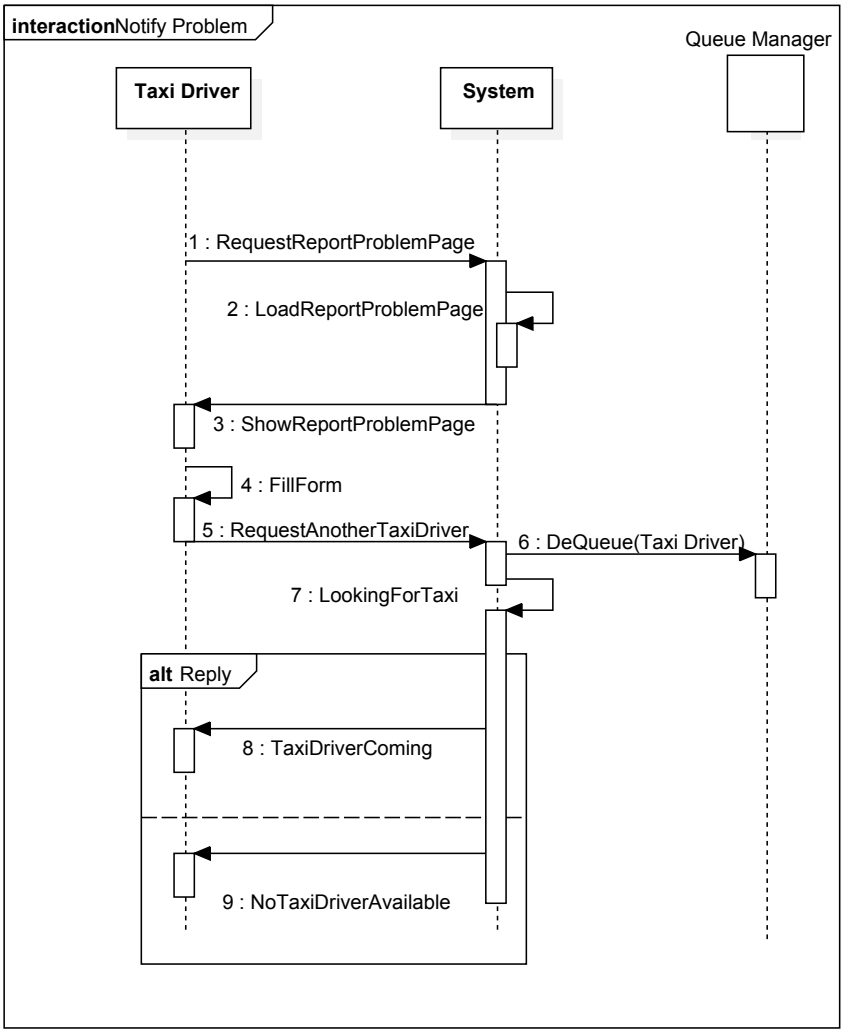
3.6.8 Manage Personal Information

Actors	Customer or Taxi driver
Preconditions	The user must be logged in.
Execution Flow	1. The user requests their profile page. 2. The user's personal information is shown on the user's application. 3. The user can begin editing their profile information by tapping (or clicking) on the *Edit* button. 4. The user edits their information and submits the changes to the system. 5. The system performs a check on the new information. 6. If the information is correct, a confirmation is sent back to the user.
Postconditions	The user profile information is changed.
Exceptions	The user provides incorrect information: an error message is shown.


interactionManage your Personal Info
Customer Taxi Driver
Customer or Taxi Driver
System
1 : AskProfilePage
2 :
opt Modify profile
3 : RequestToModify
4 : ConfirmRequest
5 : Fill form
6 : SendNewInformation
alt Check correctness
[correct informations]
alt User Or Taxi Driver
7 : UpdateProfile{}
8 : UpdateProfile{}
9 : ConfirmUpdates
[wrong informations]
10 : SendErrorMessage
11 : OkButtonPressed

3.6.9 Report Problem

Actors	Taxi driver
Preconditions	The taxi driver must be logged in
Execution Flow	1. The taxi driver requests the *Report problem* page. 2. The taxi driver fills the form and submits the information regarding a technical problem that they are experiencing. 3. If the taxi driver has a passenger on board, they can request another taxi to drive the passenger to their destination. 4. If the taxi driver requests another taxi the system looks for an available taxi driver, with the usual procedure. 5. If a taxi driver accepts the request, a confirmation is sent to the driver who is submitting the report.
Postconditions	The technical problem is reported to the system
Exceptions	• The taxi driver is located in a invalid zone: an error message is shown. • The taxi driver requests a second taxi, but no available taxi is found: an error message is shown to the user.



3.7 Performance requirements

Some indicative non functional requirements regarding performance have been identified as follows.

1. The platform must support a number users equals to 3 times the number of registered taxi drivers in the Town. Estimates must be calculated each year and modifications to the infrastructure must be made accordingly.
2. The system must process 99% of requests in less than 5 seconds.
3. The system must support parallel processing of the request to a degree proportional to at least 25% of the average number of requests.

4. Management of the database must be transparent to the user, which must have the impression of a continuous interaction with the system.
5. Estimates of the costs of the rides must be precise with a 10% error margin.
6. Estimates of the taxi drivers' ETA must be precise with a 10% error margin.

3.8 Availability and Reliability Requirements

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.

3.9 Security Requirements

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.

3.10 Maintainability Requirements

The following non functional requirements regarding the maintainability of the codebase 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.

3.11 Portability Requirements

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).