Requirements Analysis and Specification Document

November 5, 2015

POLITECNICO
MILANO 1863

Daniele Grattarola (Mat. 853101)

Ilyas Inajjar (Mat. 790009)

Andrea Lui (Mat. 850680)

Contents

1 Introduction

1.1 Purpose

The presented document is the Requirements Analysis and Specification document (RASD) for the MyTaxiService platform project.

The main purpose of this document is to analyze the problems that the new system is to solve and the customer's necessities, to identify use cases and actors, to describe the functional and non-functional requirements of the platform along with existing constraints, and to propose an in-depth specification of the system before moving on to the more concrete design phase.

This document is intended for stakeholders, software engineers, and programmers and is to be used as reference throughout the whole development of the system. Each phase of the project must be brought on with this document in mind, and any implementation of the system must be clearly designed to reflect the specifications presented here.

The secondary audience for this document includes system maintainers and developers who wish to integrate the platform's services within their own software.

1.2 Scope

The aim of this project is to create the MyTaxiService platform, a web based information system to manage the taxi service of Pallet Town.

The purpose of the platform is:

1. To provide users with a web application and a mobile application, with which to easily make use of the town's taxi service.

2. To provide taxi drivers with a mobile application, with which to manage customers' requests (submitted through the above mentioned systems)

In short, the platform is to be an integrated infrastructure to manage the initial interaction between customers and taxi drivers, to aid the former in requesting a taxi ride, and to enable the latter in managing such requests.

The platform will therefore be used exclusively to support the town's taxi service, and is aimed at making the taxi request process much efficient by reducing overall costs, communication overhead and potential requests overloads.

The platform will implement all necessary functions on both the users' side and the drivers', while keeping the two aspects separate and unaware of each other. It will also be given a particular focus to the storage of user data, in order to maximize privacy and security, and to the extensibility of the system, in order to make the platform flexible to potential changes in the requirements.

1.3 Definitions, acronyms, and abbreviations

Throughout this document, the following definitions will be applied without further explanations:

- **Platform**: the set of software applications and hardware infrastructure that make up the MyTaxiService service; these include the back-end server software, the web application and the mobile application used by the customers, the mobile application used by the taxi drivers, and all the necessary hardware needed to run the mentioned software and any needed support software.
- **System**: any subset of software components of the platform.
- **Back-end**: the software run on the back-end server of the platform which is used to handle the communication between the user applications. The term also addresses all the necessary software components that are needed to store data, perform calculations and manage the hardware (e.g. an operating system).
- **Customer-side application**: software run on a personal device which is used to send taxi requests to the system and to handle the system's replies. It is designed to be used by customers (see below) and can either be a mobile application (run on a smartphone or tablet) or a web application (run on any personal device through an Internet browser).
- **Taxi-side application**: software run on a personal device which is used to manage taxi requests forwarded by the system and to reply to the system with information on how to handle the requests. It is designed to be used by taxi drivers (see below) and is a mobile application (run on a smartphone or tablet).
- **User-side application**: the set of customer-side and taxi-side applications

- **Taxi driver**: the owner of a taxi license in Pallet Town, who uses the taxi-side application to interact with the platform.
- **Customer**: a person which intends to exploit the taxi service of the town, and who uses the user-side applications to interact with the platform.
- **User**: any customer or taxi driver who uses a user-side application.

The following acronyms will also be used in place of the extended form:

- **RASD**: Requirement Analysis and Specification Document
- **CGI**: Common Gateway Interface
- **DBMS**: Data Base Management System
- **RTO**: Recovery Time Objective
- **RPO**: Recovery Point Objective
- **SOC**: System On a Chip

The following convention will be used to refer to different items in the document:

- **sec. / secs.**: section / sections
- **req.**: requirement.

A typical use of the aforementioned abbreviation would be in the form "element Xx, sec. x.x.x" (e.g. *req. 1, sec. 1.3* - if this section contained a numbered requirement with index 1).

One last observation is to be done regarding the use of the singular *they*, which will be used to refer to single persons throughout the whole document.

1.4 Overview

The presented RASD is divided in sections and structured as follows:

- **Section 1 - Introduction**: contains support information to better understand the presented document and provides a brief introduction of the project, with its scope and general goals.

- **Section 2 - Overall description**: contains a high level descriptions of the platform to be created, with its functions, conceptual structure, relations with the outside world and design constraints.
- **Section 3 - Requirements**: contains an in-depth description of the platform's functional and non functional requirements, its use cases and actors, and a software design guide for developers.
- **Section 4 - Alloy modeling**: contains a description of the platform in the Alloy modeling language.
- **Section 5 - Additional comments**: contains a summary of the hours spent in producing the document.

2 Overall Description

2.1 Product perspective

The platform aims at replacing the traditional means of communication between taxi drivers and customers, and it is therefore designed as a stand-alone entity that is able to operate with a minimum set of dependencies towards external systems.

2.1.1 Software interfaces

- **Google Maps API**: used in both the back-end and the taxi-side applications, this free API is necessary to elaborate routes and display maps. The system interacts with the API through standard GET requests.
  - Mnemonic: map rendering and route elaboration
  - Documentation: https://developers.google.com/maps/
- **GPS**: this well known system is required by the mobile applications to get information about the users' position. The interaction happens through the standard GPS protocol provided by the OS of the users' devices.
  - Mnemonic: user position
  - Documentation: refer to OS-specific documentation
- **Red Hat Enterprise Linux**: operating system run on the back-end server.

  - Mnemonics: back-end OS
  - Documentation: https://access.redhat.com/documentation/en/red-hat-enterprise-linux/

- **Apache WebServer**: software bundle run on the back-end, which is used to allow interaction between the back-end system and the user-side application. It is used to handle standard HTTP requests from the customer-side web application and to provide CGI services to all the user-side applications.
  - Mnemonics: web-server
  - Documentation: https://httpd.apache.org/docs/2.4/

- **SQLite**: DBMS software run on the back-end, which is used to store user data in a non-resource-intensive fashion.
  - Mnemonics: DBMS
  - Documentation: https://www.sqlite.org/docs.html

2.1.2 User interfaces

The platform must be accessible to users through three different interfaces, which must be transparent to the back-end and must provide a unified user experience.

[I1] **Customer web application**: standard dynamic website generated with information associated to the user. It is accessible through any standard web browser.

[I2] **Customer-side mobile application**: mobile application developed with proprietary graphical libraries in order to maintain the same business identity throughout the different mobile OSs that the application targets.

[I3] **Taxi-side mobile application**: mobile application developed with proprietary graphical libraries in order to maintain the same business identity throughout the different mobile OSs that the application targets.

2.1.3 Device support

The back-end system is designed to be run exclusively on usual x64 server hardware and may therefore require fine-tuning of its environment, but needs to be flexible for hardware scaling (such as replication) in order to ensure its longevity.

The user-side mobile applications, on the other hand, are designed to be run on standard SOC architectures like the vast majority that is found on smartphones and tablets, and take into account the differences in the mobile operating systems that can be run on the SOC.

Finally, the customer-side web application does not have specific device requirements, since it relies on a different level of abstraction (web browser) which is not to be considered as a hardware interface.

2.2 Product functions

The fully developed platform should implement the following high level functions.

- **Customer-side application**:

  [F1] Allow users to request a taxi.

  [F2] Allow users to manage their profile.

  [F3] Allow users to issue taxi reports.

- **Taxi-side application**:

  [F4] Allow users to manage incoming requests.

  [F5] Allow users to manage their profile.

  [F6] Allow users to issue customer reports.

  [F7] Allow users to report a technical problem.

- **Generic platform functions:**

  [F8] Manage user authentication processes.

  [F9] Manage taxi queues in the different zones of the town.

2.3 Actors and related functionalities

The platform may have different functional and non-functional requirements based on the user that is interacting with it. Throughout this document, the set of these different requirements has been referred to with the terminology specified in sec. 1.3 (e.g. taxi-side application, customer-side application, etc.).

This section, instead, is aimed at giving a high level description of the platform based on the different actors that participate in its use:

[A1] **Guest**: any person who wishes to start a session with the platform through one of the user-side applications. A1 actors may:

- Register a profile into the platform
- Log into the platform
- Retrieve their password

After completing a login, A1 actors may become either of the two following types of actor.

[A2] **Customer**: a person as defined in sec. 1.3. Based on what actions A2 completes, they can be further specialized in:

[A2.1] **Logged-in Customer**: an A1 actor who logged in to the platform with a customer profile. A2.1 actors may:

* Issue a taxi request
* View and edit their profile

[A2.2] **Passenger**: an A2.1 actor who issued a taxi request. A2.2 actors may:

* Cancel or edit the issued taxi request
* Issue a taxi report

[A3] **Taxi driver**: a person as defined in sec. 1.3. A3 actors may:

- Handle (accept or refuse) a taxi request issued by an A2 actor.
- Edit their availability status
- Issue a customer report
- Report a technical problem

2.3.1 User characteristics

Users do not need specific competences to use the app other than a basic knowledge of web browsing (customers only) and of their mobile operating system (both customers and taxi drivers).

2.4 Constraints

The development of the platform will be limited by the following constraints to which the system must be subject to. These are not listed as part of the specific requirements of the platform because they stem from circumstances beyond the designers' control, and are not directly part of the customer's requests.

[C1] **Privacy policy**: processing of user data, be it personal or related to the use of the platform, is to be carried out according to the national laws and the ways and means of said processing must be clearly defined and available to the users at any time.

[C2] **Taxi licenses**: the use of the taxi-side application and the related access to the platform, is intended exclusively for taxi drivers officially recognized by the town's appropriate taxi organizations.

[C3] **Data integrity and data security**: users' personal data must be stored according to specific standards that guarantee their integrity and security. Further analysis of this constraint will be carried out in sec. 3.9.

[C4] **Legal waiver of responsibility**: due to the nature of the platform's domain, the use of the platform can be associated to car accidents and other related possible injuries; users are therefore required to accept a legal disclaimer, in order to use the platform.

2.5 Domain assumptions

Some domain assumption have to be introduced in order to further define the scope of the project, and to clearly state under which assumptions the development should proceed in those areas that are not explicitly covered by the initial customer's specification.

The assumptions are the following:

[DA1] It is assumed that the user-side devices are capable of running the intended user-side applications, according to the descriptions in sec. 2.1.3. Hardware or software compatibilities will not be discussed further.

[DA2] It is assumed that a public database exists, which can be exploited to perform checks on the official status of taxi drivers in order to ensure the satisfaction of constraint C2, sec. 2.4.

[DA3] It is assumed that the number of users in the first year of operation will not exceed the platform's capacity, and modifications to the platform will not be needed. Later modifications are not discussed in the present document.

[DA4] It is assumed that any preexisting taxi reservation service shall not be integrated with the platform.

[DA5] It is assumed that the zones in which the town is divided are static and do not change frequently.

[DA6] It is assumed that the town's taxi drivers are uniformly distributed between the zones.

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

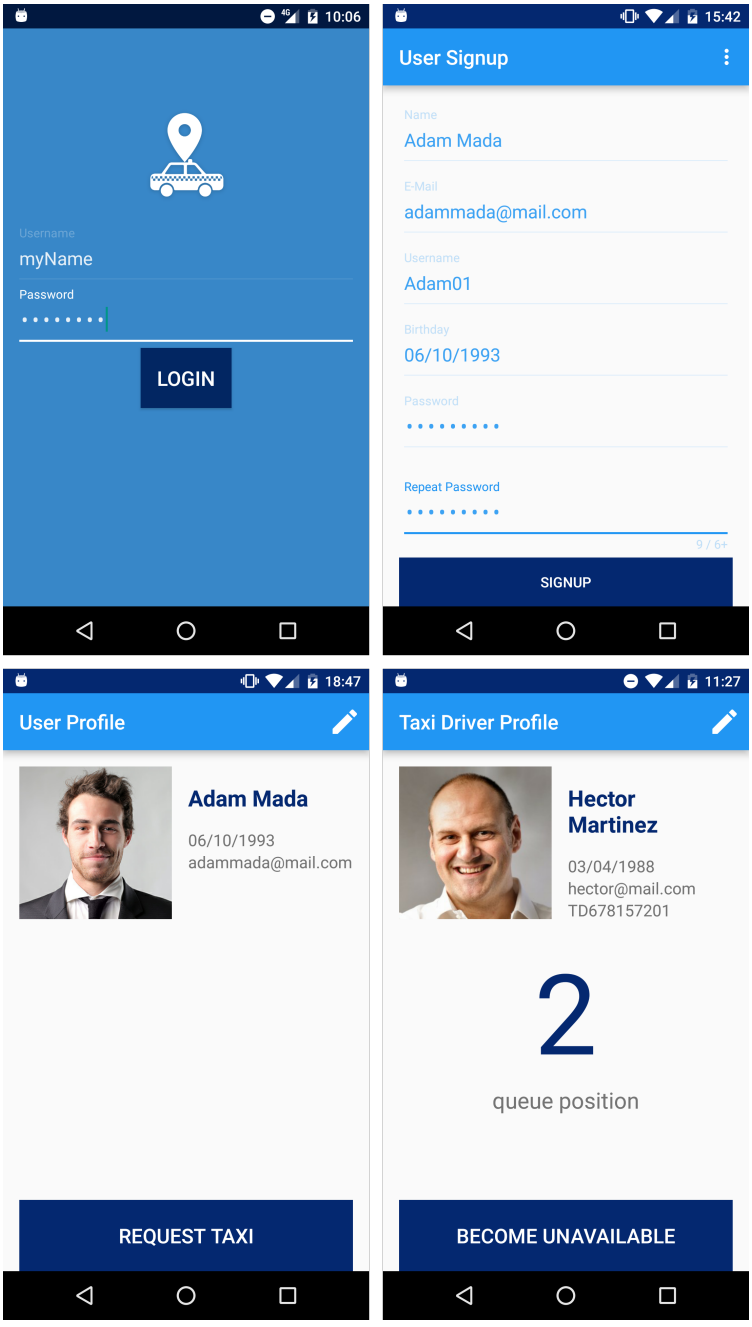
Being MyTaxiService a fully service-oriented platform, its only external interfaces must be those reserved for the final users; there is no need to design specific maintenance access to the back-end system as this is already fully standardized and does not need specific functionalities other than the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

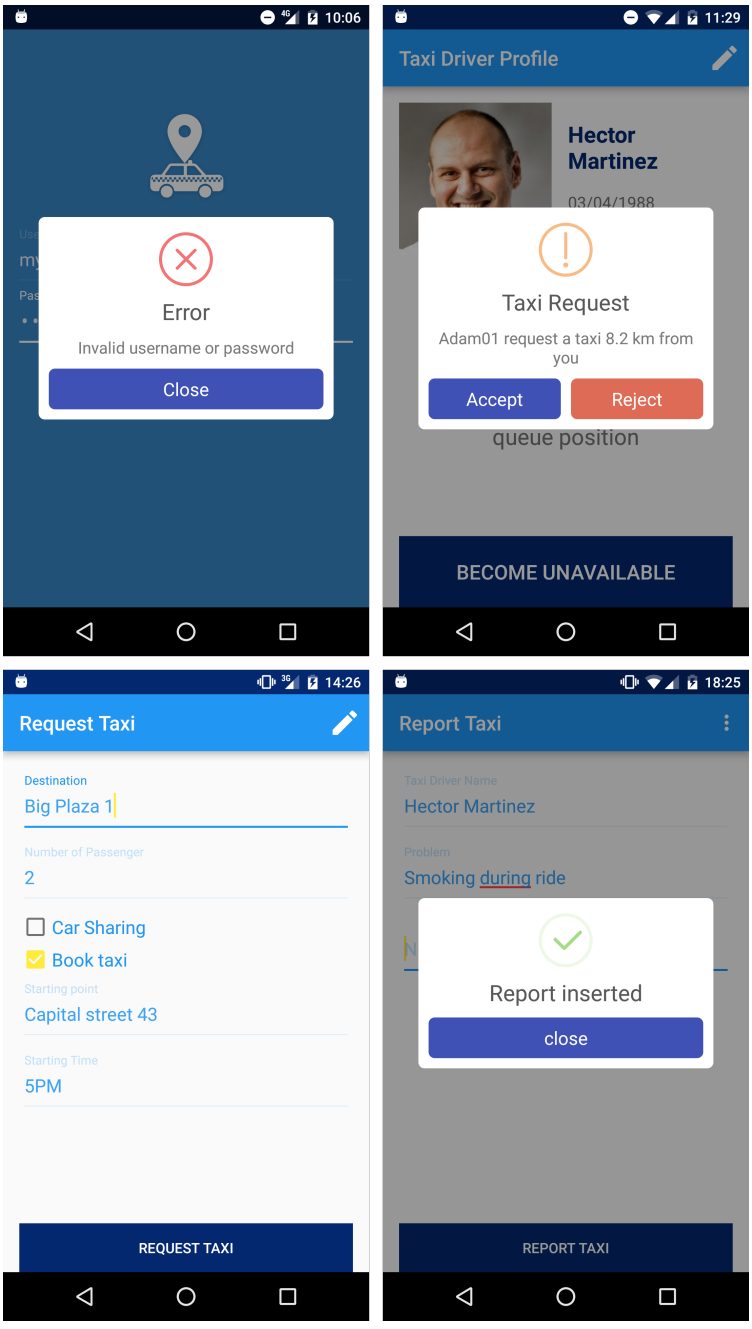
As briefly described in sec. 2.1.2, the main principle that must guide the design of the external interfaces of the platform is that of business identity continuity; therefore, this section also contains a set of design mock-ups that are to be kept as reference during the development of the user interfaces.

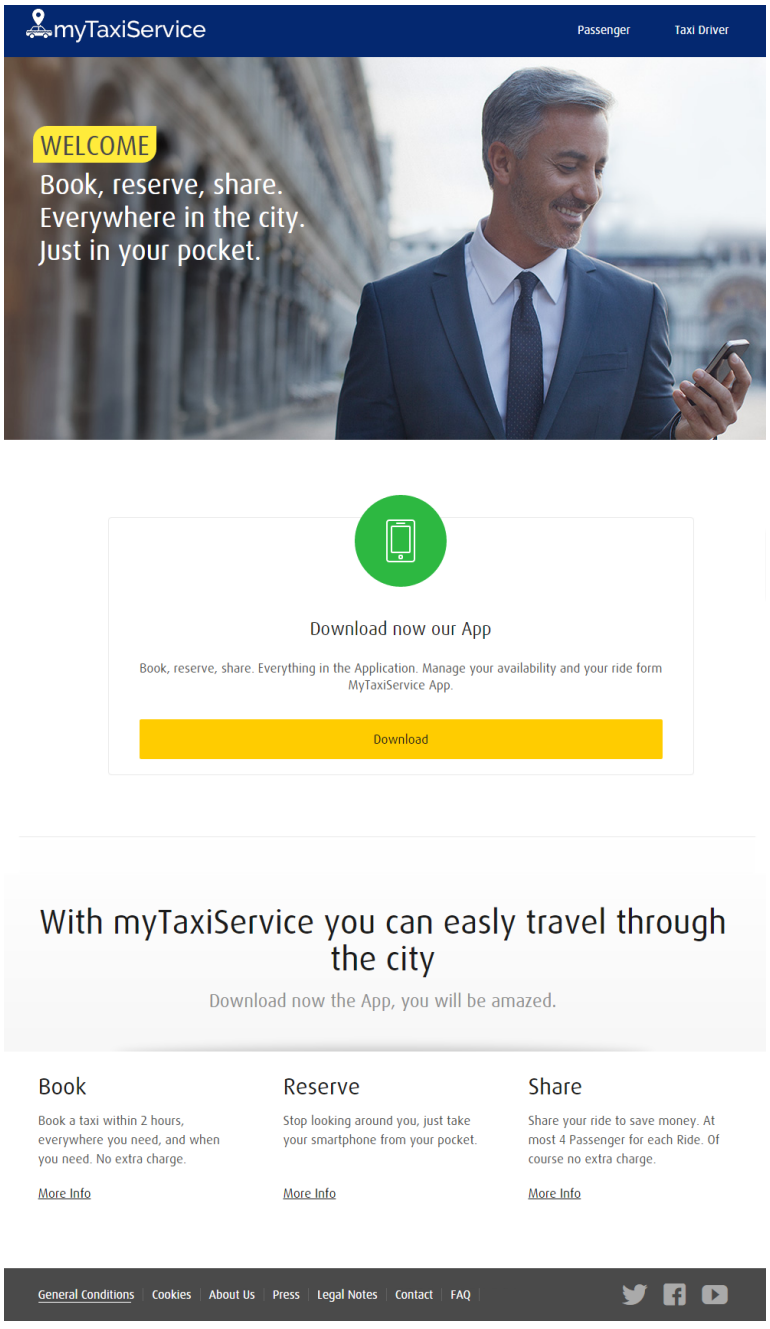
3.1.1 Interface requirements

1. Users must be able to access any function offered by the platform (and intended for them) in no more than 5 actions (clicks, taps, scrolling, form filling...).
2. The customer-side web application (interface I1, sec. 2.1.2) must implement a responsive design in order to fit most screen sizes.
3. The user-side interfaces must implement the same design language throughout the supported platforms, in order to maintain a consistent business identity.
4. All customer-side interfaces must enable access to the same functionalities.

3.1.2 Interface mock-ups




10:06
Error
Invalid username or password
Close
11:29
Taxi Driver Profile
Hector Martinez
03/04/1988
Taxi Request
Adam01 request a taxi 8.2 km from you
Accept
Reject
queue position
BECOME UNAVAILABLE
14:26
Request Taxi
Destination
Big Plaza 1
Number of Passenger
2
Car Sharing
Book taxi
Starting point
Capital street 43
Starting Time
5PM
REQUEST TAXI
18:25
Report Taxi
Taxi Driver Name
Hector Martinez
Problem
Smoking during ride
Report inserted
close
REPORT TAXI


myTaxiService
Passenger
Taxi Driver
WELCOME
Book, reserve, share.
Everywhere in the city.
Just in your pocket.
Download now our App
Book, reserve, share. Everything in the Application. Manage your availability and your ride form MyTaxiService App.
Download
With myTaxiService you can easly travel through the city
Download now the App, you will be amazed.
Book
Book a taxi within 2 hours, everywhere you need, and when you need. No extra charge.
More Info
Reserve
Stop looking around you, just take your smartphone from your pocket.
More Info
Share
Share your ride to save money. At most 4 Passenger for each Ride. Of course no extra charge.
More Info
General Conditions
Cookies
About Us
Press
Legal Notes
Contact
FAQ

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 sec. 2.2, and must be considered alongside the domain assumptions in sec. 2.5.

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 sec. 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 (also see req. 3, sec. 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 enable registered users to retrieve a forgotten password (by sending them a temporary password on the email address associated to the account)
3. The application must display confirmations and error messages forwarded by the back-end (see sec. 3.6 and req. 1, sec. 3.2.1).
4. 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 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 (the 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.

   (e) The application must enable the customer to reserve a ride for more than 1 person (see req. 4, sec. 3.2.5)

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.

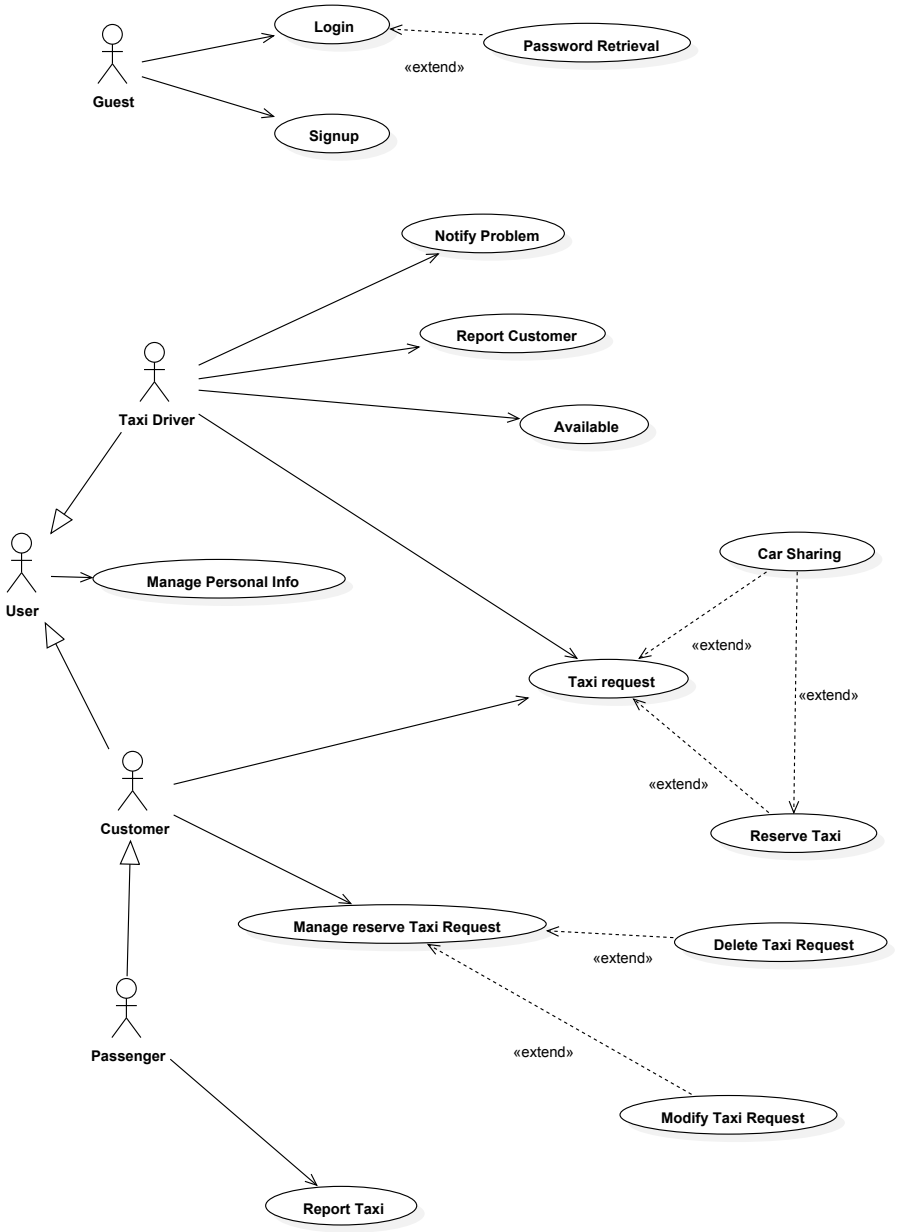
3.2.5 Functional constraints

1. Taxi reservations must can only be issued at least 2 hours before the requested time of the ride (function F1, associated to req. 1.(b), sec. 3.2.4).
2. Requests associated to taxi reservations must be forwarded to taxi drivers exactly 10 minutes before the requested time (function F9, associated to req. 1.(b), sec. 3.2.4).
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).
4. A *Ride* object must allow up to 6 possible passenger between application users (possibly in *Car sharing* mode) and additional passenger added by a user (see requirement 1.(e), sec. 3.2.4)

3.3 Use cases

This section contains a graphical UML description of the possible use cases of the platform, associated to the respective actors.

A more in-depth description of the use cases will be given in secs. 3.5 and 3.6.


Guest
Login
Password Retrieval
«extend»
Signup
Notify Problem
Report Customer
Taxi Driver
Available
User
Manage Personal Info
Car Sharing
«extend»
Taxi request
«extend»
«extend»
Customer
Reserve Taxi
Manage reserve Taxi Request
Delete Taxi Request
«extend»
Passenger
«extend»
Modify Taxi Request
Report Taxi

3.4 Entities and objects

3.4.1 Logical database requirements

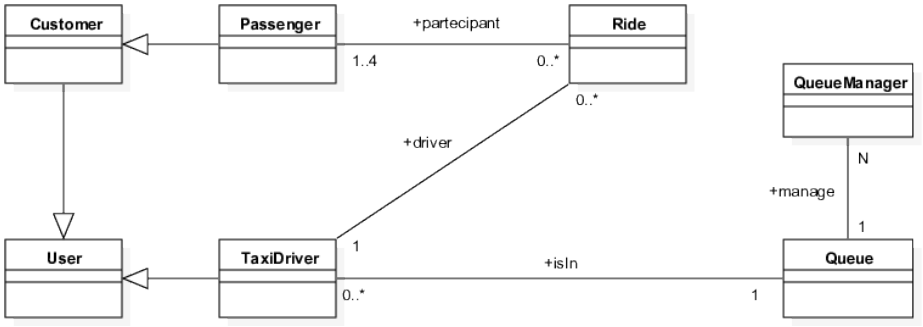
The majority of data processed by the DBMS consists of Java *String* and *Date* objects. Images and complex objects might be stored in the database, too, and must therefore be considered during the design of the database.

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 needed 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.

3.4.2 Class diagram

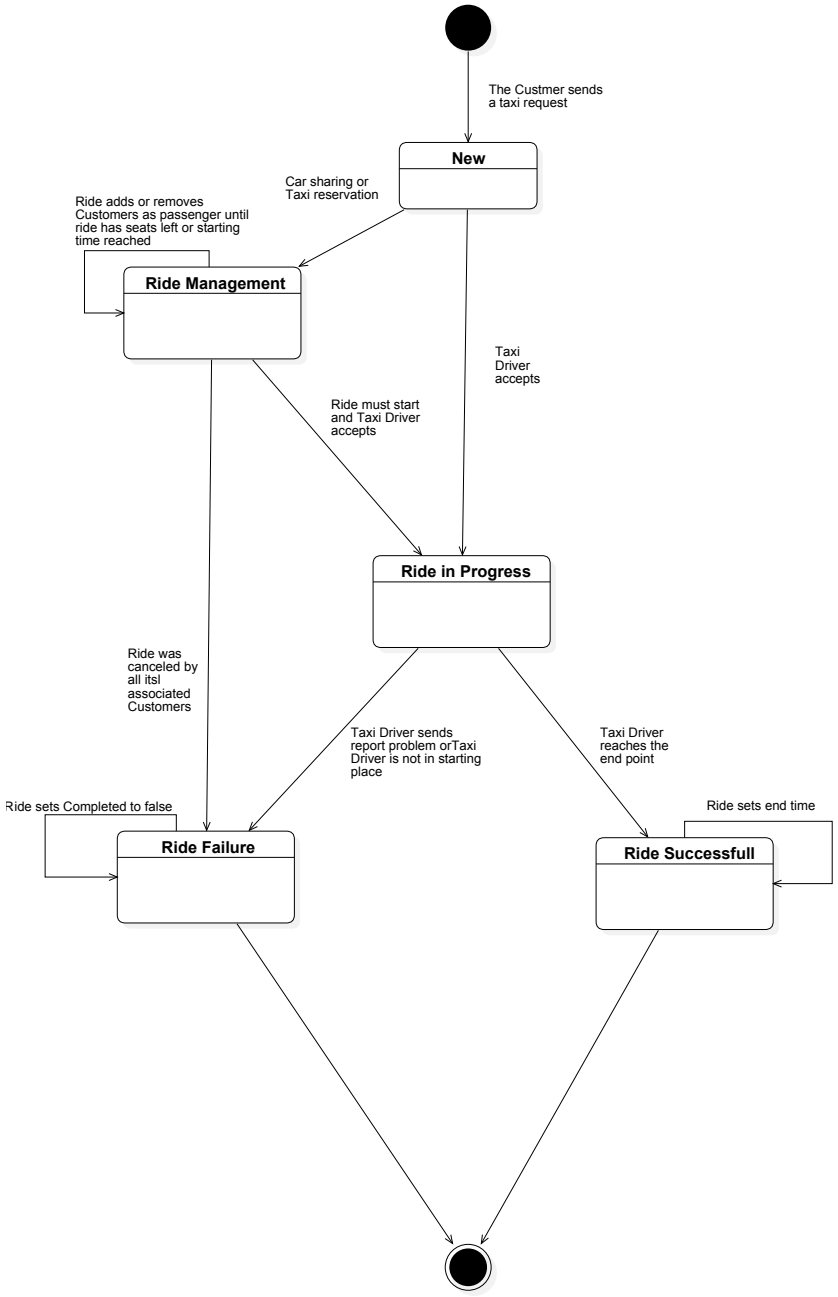


3.4.3 Entity behaviors

[EB1] Taxi driver

Taxi drivers, once logged in, can choose to toggle their availability between the two main statuses *Available* (if the driver is waiting for incoming requests and is in a local queue) and *Unavailable* (if the driver is unable or unwilling to accept new requests; this applies also to drivers who are carrying passengers, who are automatically considered *unavailable* once the accept a request).

[EB2] Ride


The Custmer sends a taxi request
New
Car sharing or Taxi reservation
Ride adds or removes Customers as passenger until ride has seats left or starting time reached
Ride Management
Taxi Driver accepts
Ride must start and Taxi Driver accepts
Ride in Progress
Ride was canceled by all itsl associated Customers
Taxi Driver sends report problem orTaxi Driver is not in starting place
Taxi Driver reaches the end point
Ride sets Completed to false
Ride sets end time
Ride Failure
Ride Successfull

[EB3] Queue manager

The queue manager is the software component in charge of smartly managing the local queues and reservations scheduling in the platform. It has access to all the instances of *ride, taxi driver* and *customer* objects and can modify these objects' statuses in reaction to different events.

The core algorithms of the queue manager have to:

- Handle one local queue for each zone of the town.
- Respond to taxi drivers' replies to customer requests.
  - If a taxi driver accepts a ride request, they must be de-queued and set as *Unavailable*.
  - If a taxi driver refuses a ride request, they must be de-queued and set as *Unavailable*[1].
  - If a taxi driver notifies a technical problem (function F7 of sec. 2.2), they must be de-queued and set as *Unavailable.*
- Keep track of all the customer ride reservations and manage eventual changes.
  - Dynamically assign taxi drivers to a customer when the requested time arrives.
  - Respond to changes in a customer's reservation (consistency checks are carried out by other components of the platform) and possibly remove the request from the schedule.

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.

[1] This behavior is different from the initial customer specification; it has been modified to prevent a possible unwanted situation in which all the taxi drivers in a queue refuse the request and:

1. The taxi drivers are continuously notified with the customer's request.
2. The user never gets a failure notification, because the system is kept looping through a *de-facto* unavailable queue.

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 (tracked by the GPS on Adam's phone), 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 creates a *Ride* object which will be kept in a *Pending* status.

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.

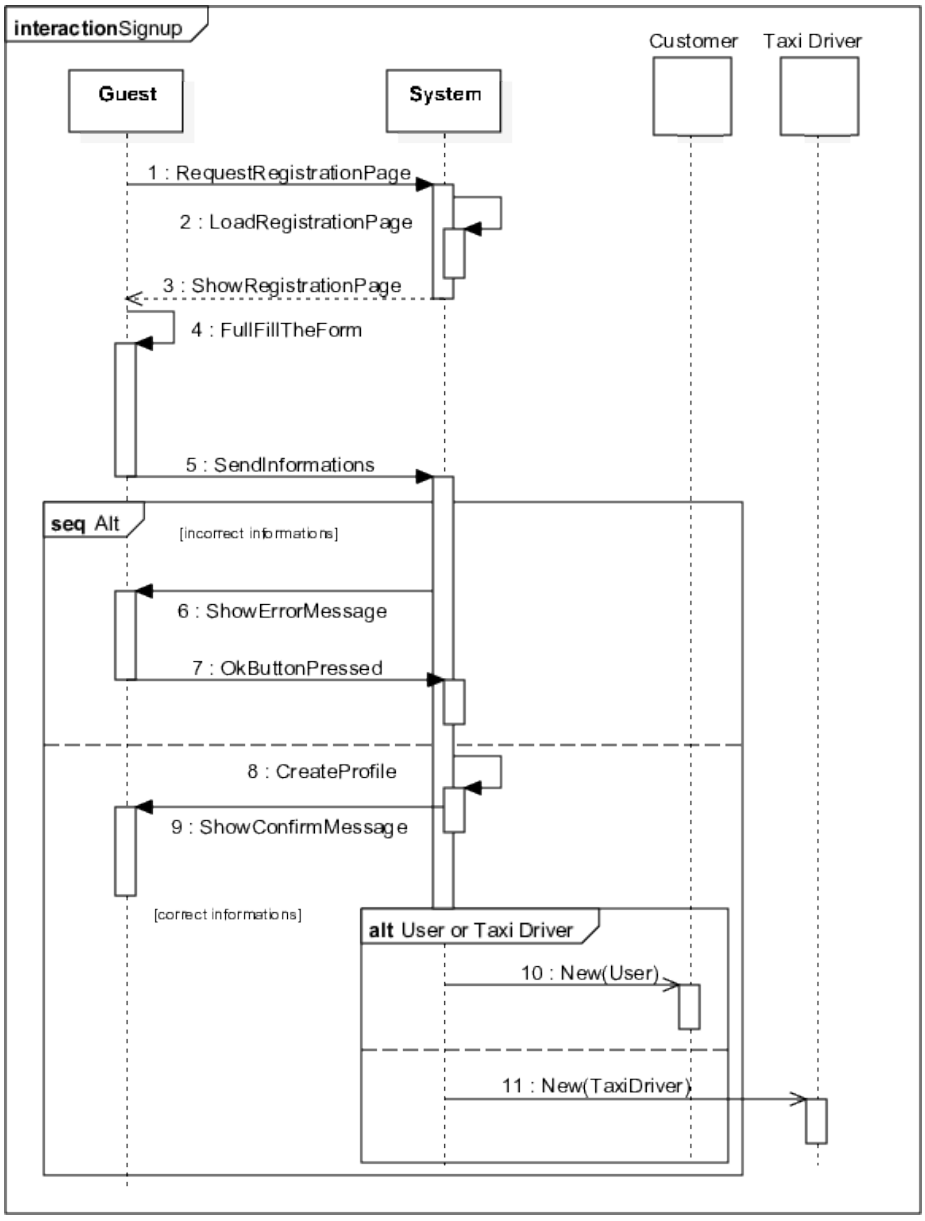
Through the *Report problem* page of the app, he notifies the problem to the system by filling the form and submitting. The system acknowledges the report and asks Hector if he'll be needing a new taxi; Hector confirms, and the system forwards his request to Samuel, who is the first taxi driver in Hector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

3.6 Flow of events

This section contains a description of how different use cases should happen, divided in sequential steps. The participating actor are those described in sec. 2.3.

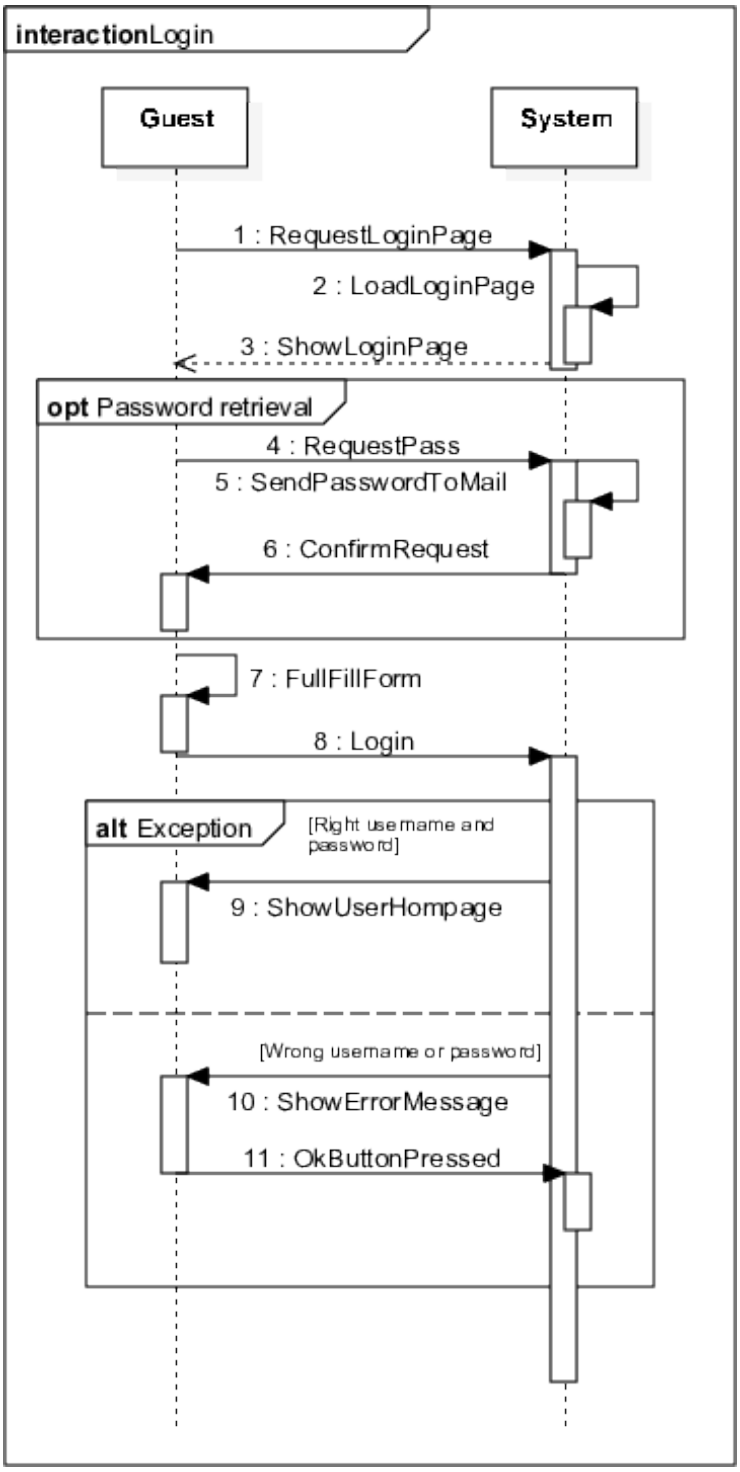
3.6.1 Sign-up

Actors	A1 - 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
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)

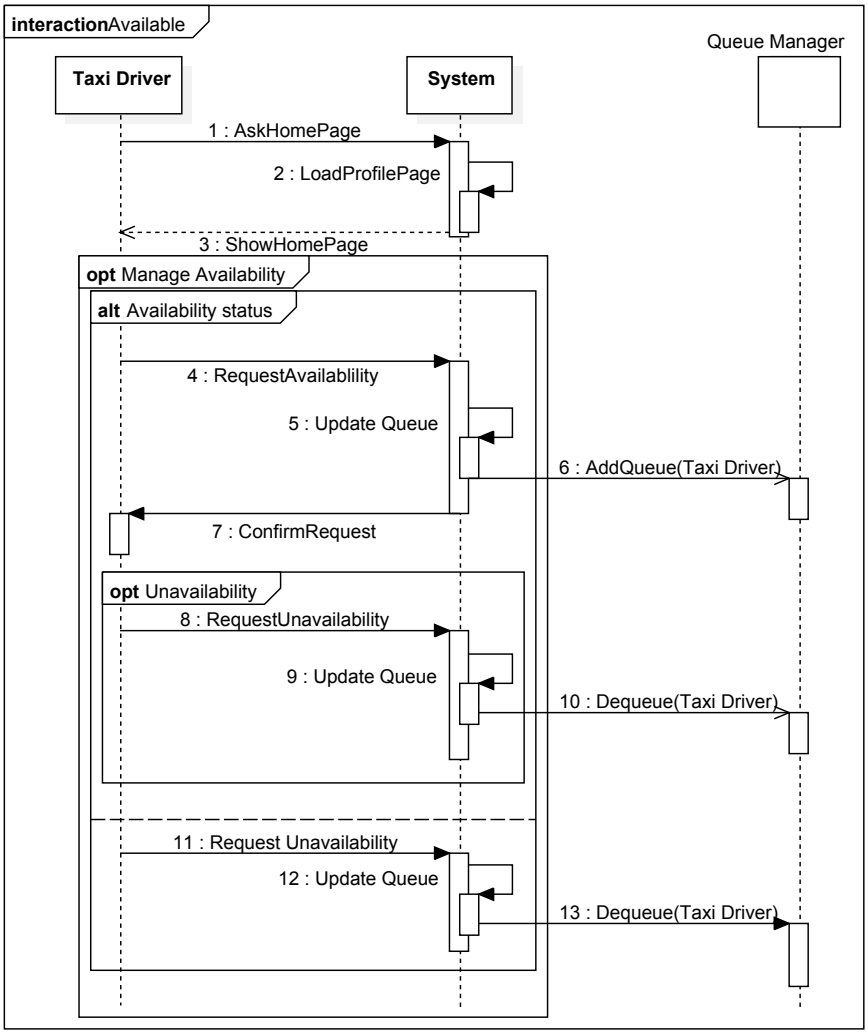
3.6.2 Login

Actors	A1 - 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	A3 - 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.



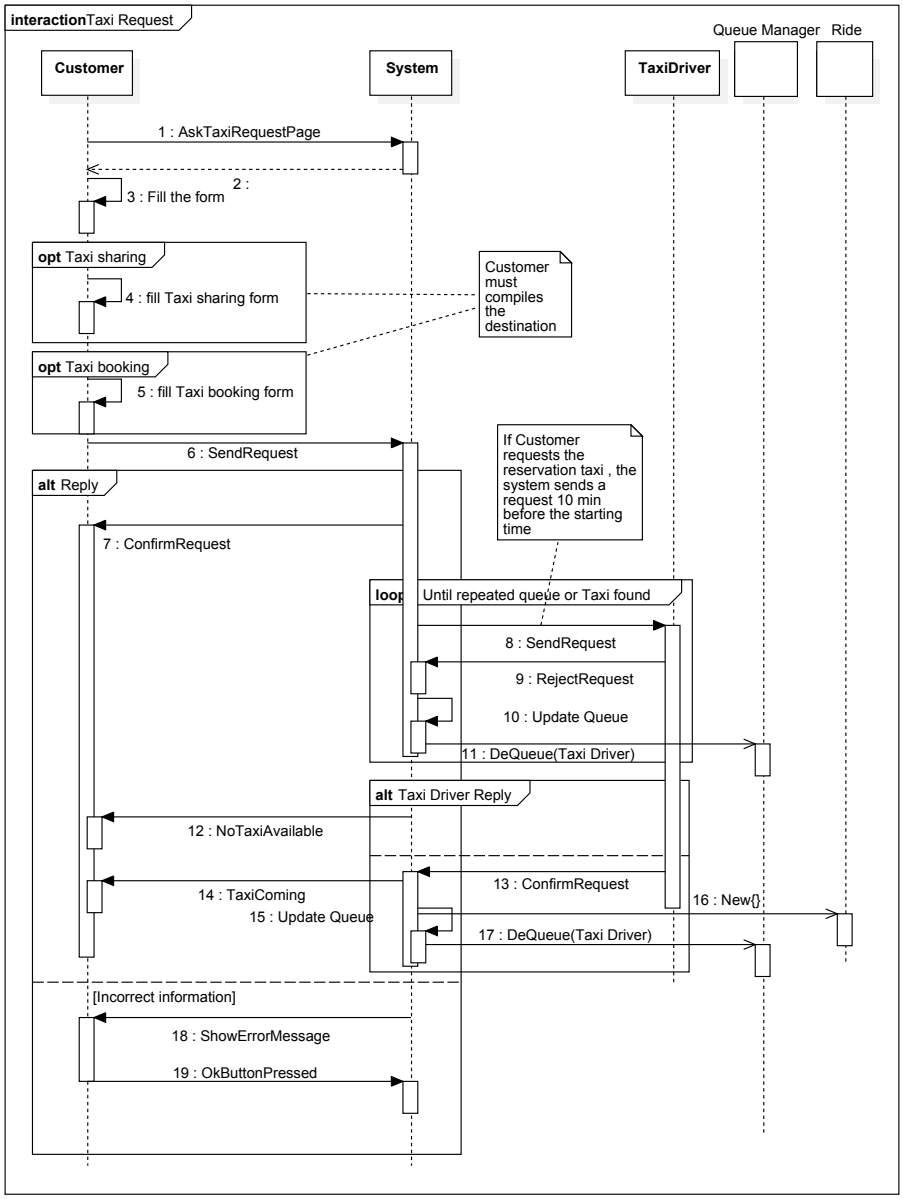
3.6.4 Taxi request

Actors	A2.1 - Customer, A3 - Taxi driver
Preconditions	Both users must be logged in, the taxi driver must be available.

Execution Flow

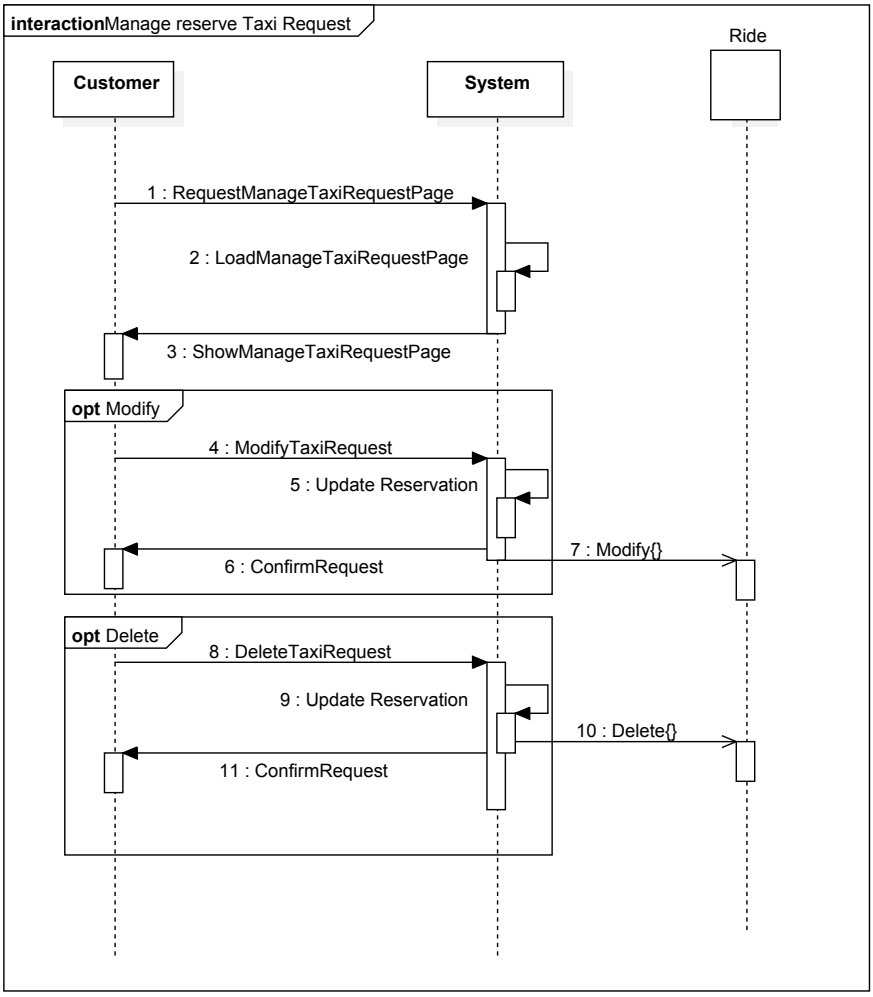
1. The customer requests the *Taxi request* page
2. The customer fills the request form according to their preference and sends the information to the system: they can choose to request a ride for a certain number of passengers, reserve a taxi or enable the *Taxi sharing* option.
3. Based on the type of request that the customer issued, the system generates a request with all the needed data and commits it to the *Queue manager* (entity E3, sec. 3.4.3)
4. The taxi driver can either ignore the request or accept it.
5. 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.
6. 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
Customer must compiles the destination
opt Taxi booking
5 : fill Taxi booking form
6 : SendRequest
If Customer requests the reservation taxi , the system sends a request 10 min before the starting time
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

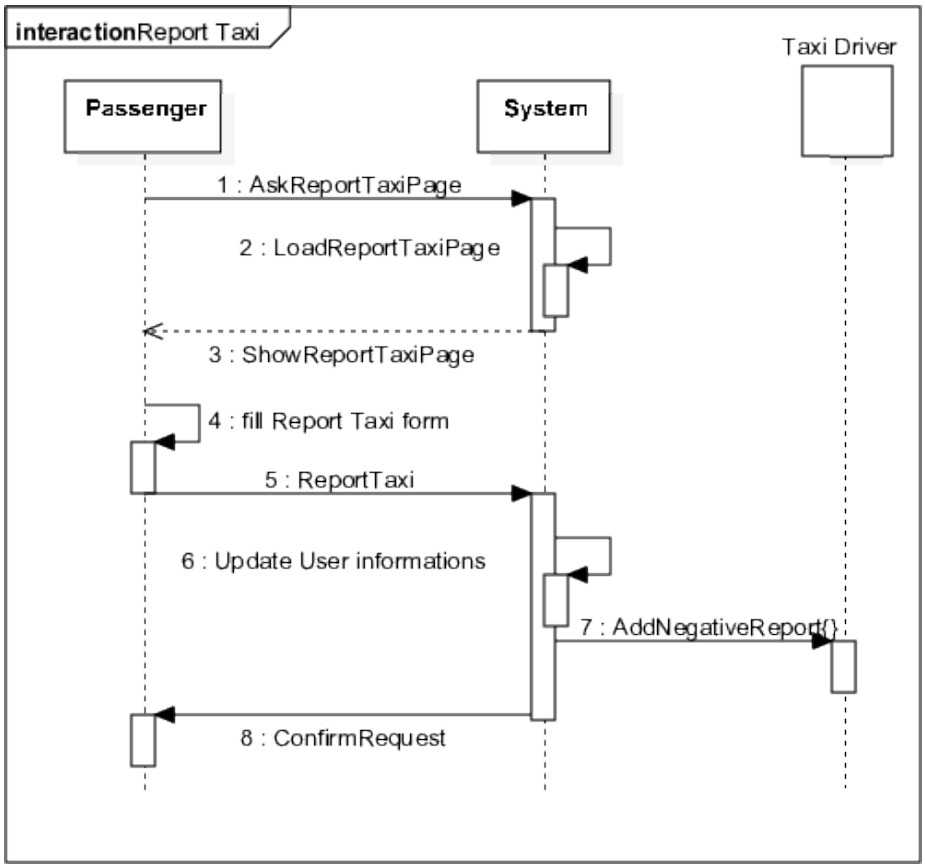
3.6.5 Manage *Reserve taxi* request

Actors	A2.1 - 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 their request too late: an error message is shown and the modification is not allowed.


interaction Manage reserve Taxi Request
Customer
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

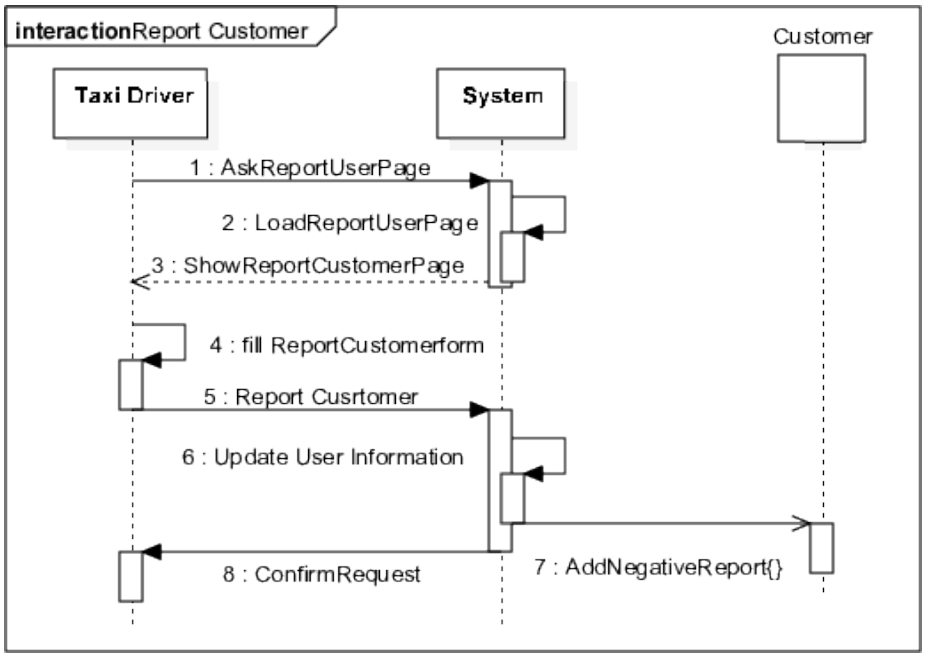
3.6.6 Report taxi

Actors	A2.2 - 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
Passenger
System
Taxi Driver
1 : AskReportTaxiPage
2 : LoadReportTaxiPage
3 : ShowReportTaxiPage
4 : fill Report Taxi form
5 : ReportTaxi
6 : Update User informations
7 : AddNegativeReport()
8 : ConfirmRequest

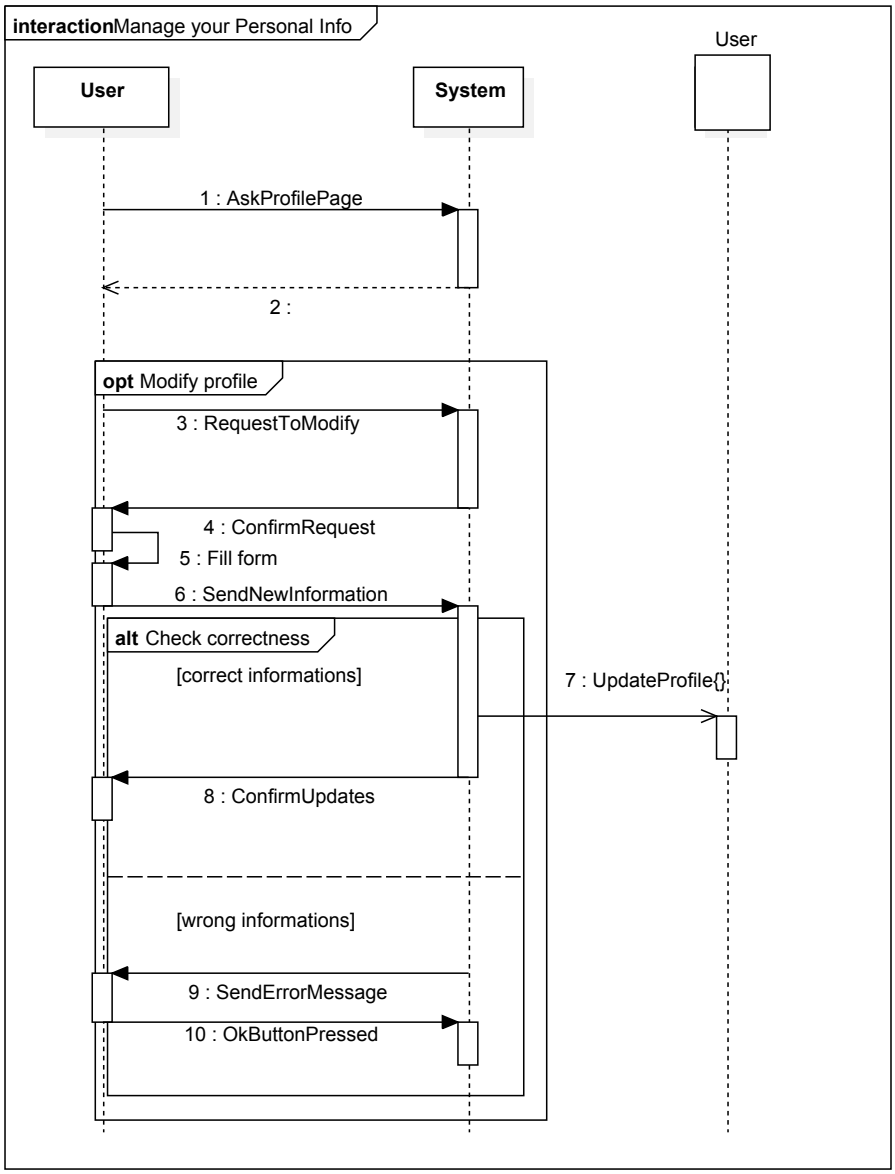
3.6.7 Report customer

Actors	A3 - 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 customer'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

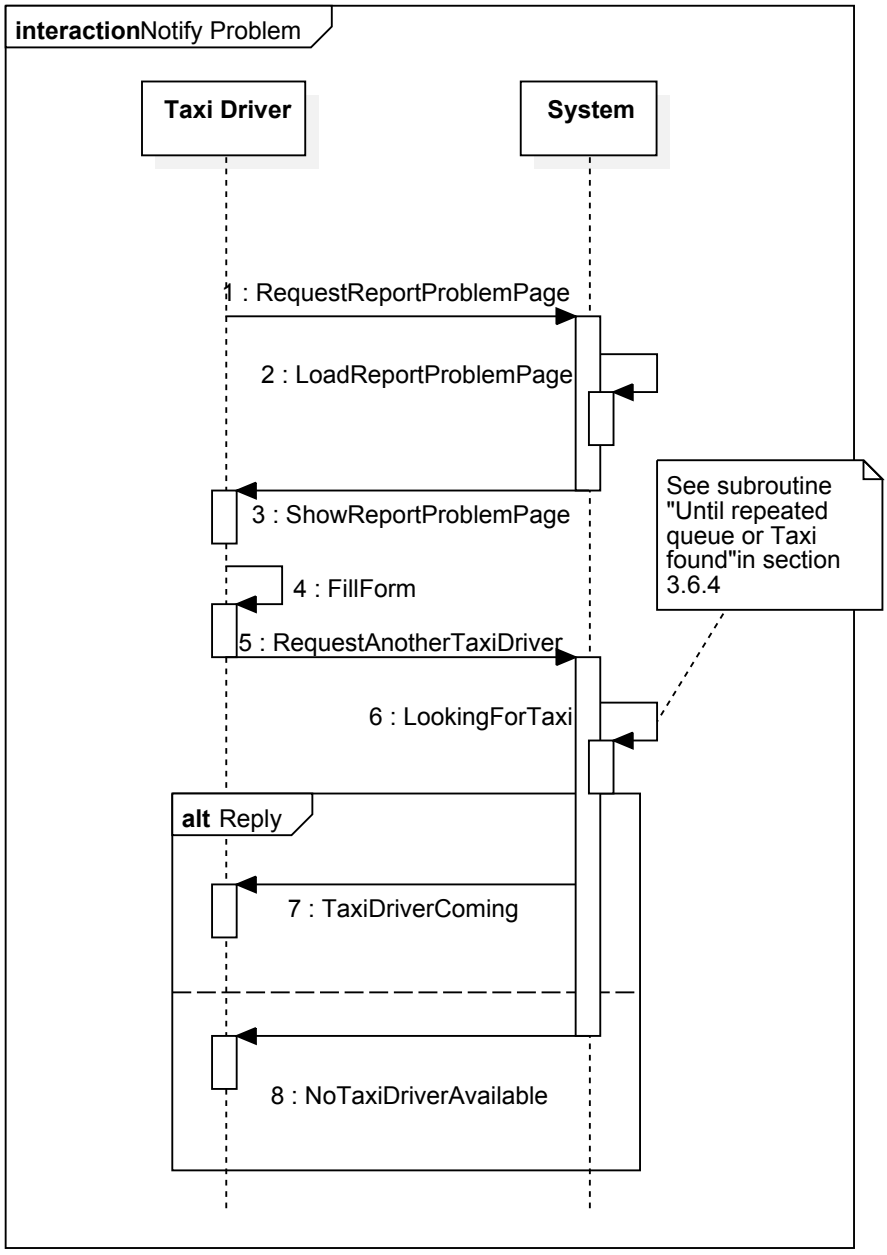
3.6.8 Manage personal information

Actors	A2 - Customer or A3 - 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
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	A3 - 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.


interactionNotify 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 Performance requirements

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

3.7.1 Back-end performance

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.

3.7.2 User-side performance

1. The mobile user-side applications must be as energy efficient as possible, in order to not drain the battery on the users' mobile devices.
2. The users' profile pictures must have a resolution smaller than 1000x1000 pixels, in order to be efficiently stored, loaded into RAM, and transmitted.

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 constraint C3, sec. 2.4.

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 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 sec. 2.1.2, 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 preexisting system). Therefore:

1. The back-end software must be developed in Java Enterprise Edition.
2. Integration with support modules in the back-end 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).

4 Alloy Modeling

4.1 Alloy source code

```
/**Signatures**/
//Sig for Users, having username and mail
abstract sig User{
        username : MString,
        mail : MString
}{
        username != mail
}

//Sig to identify Customers
abstract sig Customer extends User{}

//Passenger is a Customer how request a taxi
sig Passenger extends Customer{}
```

```
//Taxi Driver can be Available
sig TaxiDriver extends User{
        license : MString,
        available : Boolean
}

//Ride has: one TaxiDriver, one Starting position, a
   set of Passenger and some destination positions
sig Ride{
        start : one Position,
        destinations : some Position,
        transport : set Passenger,
        hasDriver : lone TaxiDriver,
        pending : Boolean
}{
        #transport >= 1 and #transport <= 4 and
        all d:destinations | different[d, start] =
           True and
        #hasDriver = 1 iff pending = True
}

//Queue for each city Zone
sig Queue{
        contains : set TaxiDriver
}

//Manager of Queues
one sig QueueManager{
        manage : some Queue
}

/**Support Signatures**/
sig MString{}
sig Float{}
```

```
sig Position{
        x : Float,
        y : Float
}

abstract sig Boolean{}
one sig True extends Boolean {}
one sig False extends Boolean {}

/**Facts**/
//All the Passengers are Customers (and also Users)
fact {
        Passenger in Customer
}

//Users are composed by all the TaxiDrivers and
   Customers
fact {
        User = TaxiDriver + Customer
}

//No repeteaded Positions
fact uniquePositions{
        all p,q:Position | p!=q implies different[p,q]
            = True
}

//All the Positions are used in the Ride sig
fact noUselessPositions{
        Ride.start + Ride.destinations = Position
}

//No useless Float sig
fact onlyUsedFloat{
        Position.x + Position.y = Float
}
```

```
//A User has unique username and mail
fact uniqueUsers{
        all u,w : User | (u.username = w.username or u
            .mail = w.mail) iff u=w
}

//A TaxiDriver is identified by his own license
fact uniqueLicense{
        all t1,t2 : TaxiDriver | t1.license = t2.
            license iff t1 = t2
}

//There are no username equals to mail
fact{
        User.username & User.mail = none
}

//Not exists license equals to username and mail
fact{
        (User.username + User.mail) & TaxiDriver.
            license = none
 }

//The String used are used only for username, mail and
    license
fact{
        User.username + User.mail + TaxiDriver.license
             = MString
}

//a User can be transported at most by one pending
   Ride
fact{
        all u:Ride.transport | (lone r:Ride | r.
            pending = True and u in r.transport)
```

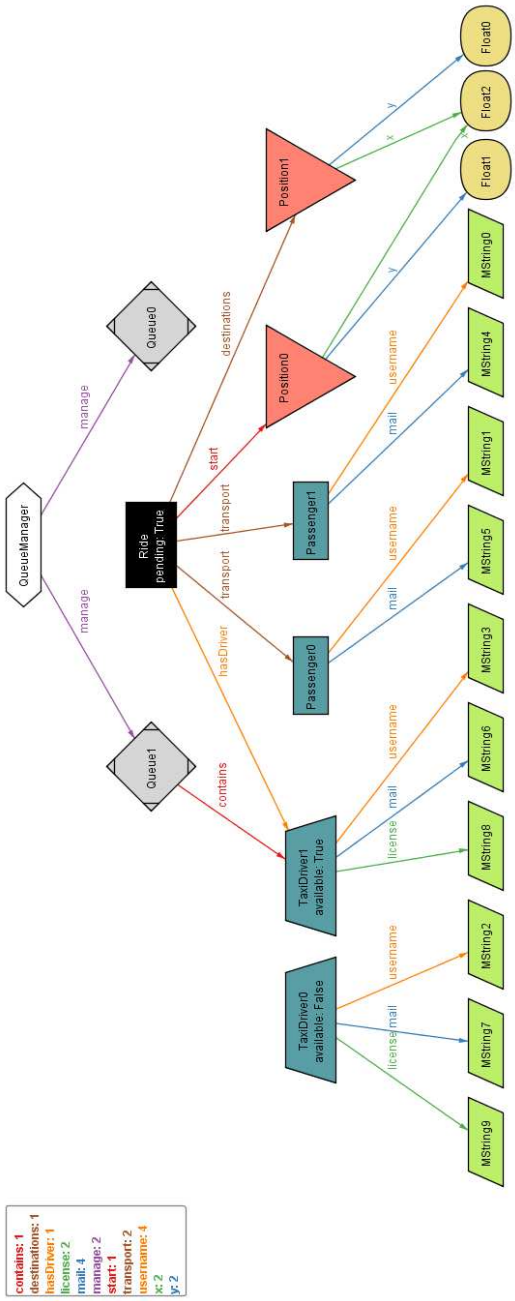
```
}

//If a Taxi Driver is in a Queue implies his
   availability
fact taxiInQueueIsAvailable{
        all t:TaxiDriver | t in Queue.contains implies
            t.available = True else t.available =
           False
}

//Every Taxi Driver belogns at most to one Queue
fact taxiContainedInOneQueue{
        all q,p:Queue | q!=p implies (all t:q.contains
            | !t in p.contains)
}

//Every Queue belongs to the QueueManager
fact exacltyOneManager{
        all q:Queue | (one m:QueueManager | q in m.
           manage)
}

//For each Ride the number of destinations belongs to
   the # of Passenger
fact maxDestinations{
        all r:Ride | #r.destinations <= #r.transport
}

/**Functions **/
//True if the Positions are different, False otherwise
fun different[p1,p2 : Position] : Boolean{
        (p1.x != p2.x or p1.y != p2.y) implies True
           else False
}

/**Assertions**/
```

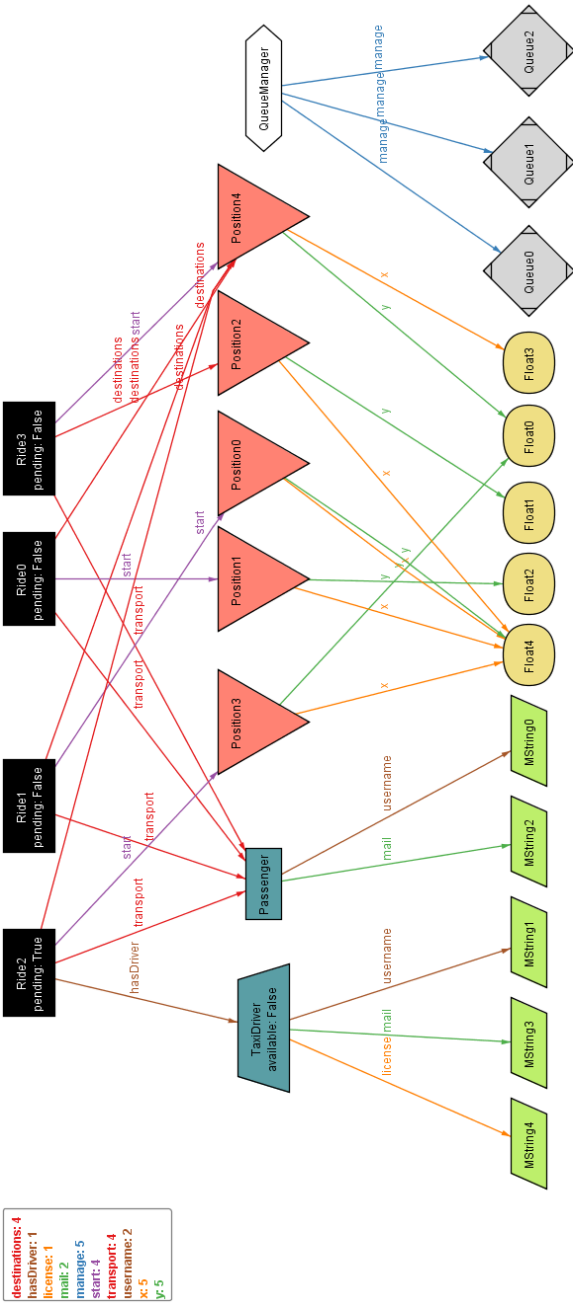
```
assert differentsUsernames{
        no u1,u2:User | u1.username = u2.username and
            u1 != u2
}

assert differentsMail{
        no u1,u2:User | u1.mail = u2.mail and u1 != u2
}

/**Predicates**/
pred taxiDriverAvailable{
        all t:TaxiDriver | t.available = True implies
            (one q:Queue | t in q.contains)
}

pred intrestingPred{
        #Passenger > 1
        #Position < 3
        #Queue = 2
        #Ride > 0 and #Ride < 3
        #TaxiDriver > 1
        one t:TaxiDriver | t.available = True
        one r:Ride | r.pending = True
}

pred show{}

/**Executions**/
run taxiDriverAvailable for 5
```

4.2 Alloy output

The output is shown to give a visual confirmation of the consistency of the model, even if it does not add any further information to the document.

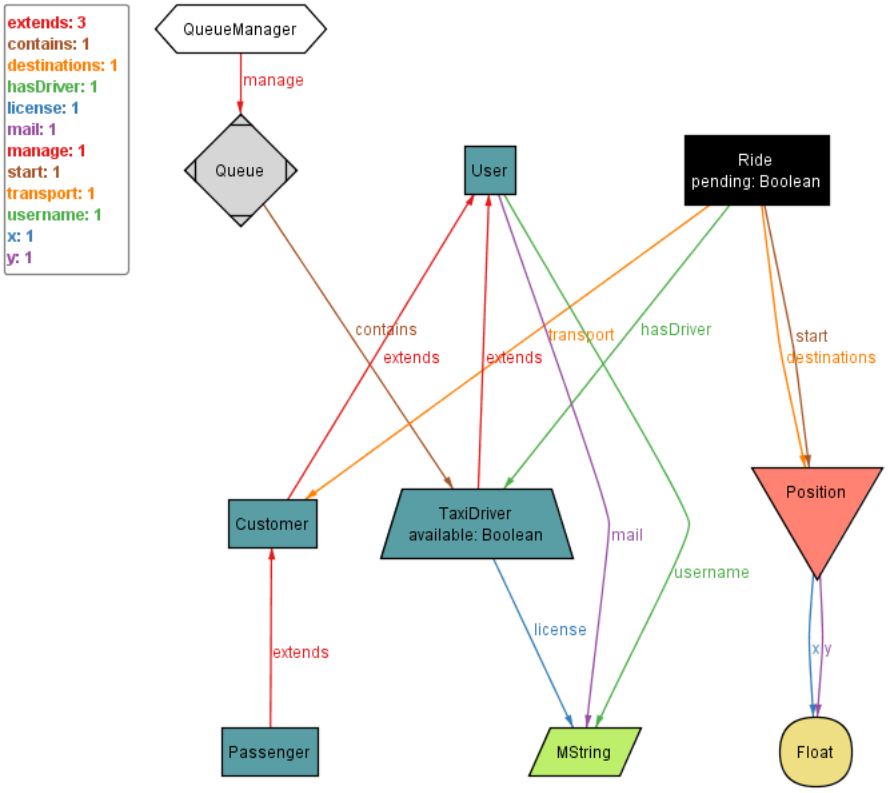
Note that passengers and taxi drivers can be related to more than one ride; this is done intentionally, as ride objects do not cease to exist once the ride is completed, but are stored and are associated with the user's history.


QueueManager
manage
manage
Queue0
Queue1
Ride
pending: True
contains
hasDriver
start
destinations
transport
transport
Position0
Position1
Passenger0
Passenger1
TaxiDriver0
available: False
TaxiDriver1
available: True
license
mail
username
mail
username
mail
username
mail
username
x
y
x
y
Float0
Float1
Float2
MString0
MString1
MString2
MString3
MString4
MString5
MString6
MString7
MString8
MString9
contains: 1
destinations: 1
hasDriver: 1
license: 2
mail: 4
manage: 2
start: 1
transport: 2
username: 4
x: 2
y: 2


QueueManager
Queue2
Queue1
Queue0
manage
Position4
Position2
Position0
Position1
Position3
Ride3
pending: False
Ride0
pending: False
Ride1
pending: False
Ride2
pending: True
destinations
start
transport
hasDriver
Passenger
TaxiDriver
available: False
x
y
Float3
Float0
Float1
Float2
Float4
username
mail
license
MString0
MString2
MString1
MString3
MString4
destinations: 4
hasDriver: 1
license: 1
mail: 2
manage: 5
start: 4
transport: 4
username: 2
x: 5
y: 5

4.3 Alloy metamodel

The following platform metamodel was derived from the Alloy code in section 4.1 and is a further expansion of the class diagram in section 3.4.2.



5 Additional Comments

The production of this document has been a joint effort of all the authors, with a fair distribution of the mansions which caused each member of the group to work on all the parts of the document.

The production has been carried out between 16/10/2015 and 6/11/2015 for a total time expense of:

- **Group work**: 28 hours
- **Individual work**:

Daniele Grattarola (Mat. 853101)	10.5 hours
Ilyas Inajjar (Mat. 790009)	9 hours
Andrea Lui (Mat. 850680)	10 hours