Politecnico di Milano

Scuola di Ingegneria dell'Informazione

Corso di Laurea Magistrale di Computer Science and Engineering

Software Engineering 2 Project

myTaxiService

Part I : RASD (Requirement analysis and specification document)

Principal Adviser:
Prof. Raffaela Mirandola

Authors:

Turconi Andrea ID n. 853589

Raimondi Lorenzo ID n. 859001

Accademic year 2015-2016

Contents

1 Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD). The main goal of this document is to provide the requirement specification that must be met by the system developed. This document presents a description of functional and non-functional requirements, using graphic models, and analyze the real need of the customer to modeling the system. This document is intended for both developers, who have to implement the requirements, and stakeholders. It is also important because it is necessary to establish coherence between the developers and the processed request.

1.2 Scope

The system that will be developed is both an online software application and a mobile application, called MyTaxiService. The purpose of this product is to help people in calling for a taxi service, simplifying the access of passengers to the service, and to help the taxi drivers to manage their service, guaranteeing a fair management of taxi queues. The online platform and the mobile app have to allow a new user registration with all his personal data, the login of an already registered user, the possibility to request a taxi for the users side, accept or reject a call and communicate availability for the taxi drivers side. A registered member can either request an immediate taxi or reserve one for a certain time. If the request is accepted, the system sends a confirmation message to the passenger with the code of the taxi and the waiting time. The application must provide to the user the possibility to see all his information, to edit his data and take a look on the bookings made. The application must also be able to manage queues of taxi drivers based on their availability and their position, which is recovered by the GPS. Each taxi is inserted into the queue of the area in which its position is detected by the GPS. The main goal of the application is to be able to assign the first taxi driver available to the user who makes the request, both reservation and immediate service.

1.3 Actors

- Visitors: all the users who have not an account. They can only view the login page and the registration form, in order to be able to access all the functionality provided by the applications as registered users.
- Registered users: all the users who have an account. They can make requests for taxi service, make reservations, specifying the origin and the

destination of the ride, and manage their data.

- Taxi drivers: people who are authorized to practice that profession. They can accept or reject a request and communicate their availability.

1.4 Goals

- [G1] allow a visitor to become a registered user
- [G2] allow a visitor to log in to the application
- [G3] allow user to make a request of taxi service, both reservations and for immediate service
- [G4] allow taxi drivers to communicate their availability
- [G5] allow taxi drivers to accept or reject a request
- [G6] allow registered members to view their reservations and change them.

1.5 Definitions, acronyms and abbreviations

1.5.1 Definitions

- Request a taxi: send a request to the system of any type of taxi service, both reservations and demand
- Demand a taxi: send a request to the system of an immediate taxi service
- Reservation of a taxi: send a request to the system of a taxi service for a certain time, specified in the form
- Future reservation: a reservation that has not yet been executed
- Pending request: a request that has been sent by the use but not managed yet by the system.
- Accepted request: a request received and managed by the system, which has already accepted it, but has not yet selected a driver for fulfil it.
- Ongoing request: a request that is being accomplished right at this moment; this time span goes from the driver selection instant to the user's arrival in the desired destination.
- Completed request: a past request, that has been completely performed.

1.5.2 Abbreviations

- [Gn] n-goal
- [Rn] n-functional requirement
- [Dn] n-domain assumption

1.6 References

- Specification document: assignment 1 and 2.pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications

1.7 Overview

This document is composed in four different sections. Section 1 is the introduction where it gives the description of the document and some basic information. Section 2 is the overall description of the document, where the constraints and assumptions are defined. Section 3 is the main part of the document, where the list of requirements, scenarios, and use cases are clearly defined. Section 4 is the appendix that contains models realized using Alloy syntax and verifies the consistency with Alloy Analyzer.

2 Overall description

2.1 Procuct perspective

The product we release is a web-based application, which implements a classic client-server model. In order to get as much self-reliance as possible, the server side does not need any administration staff, so its condition depends completely on users. Since there is neither a previous version nor a remaining hardware infrastructure for this system, we do not have to manage integration problems, but to create a brand new environment.

2.1.1 User interfaces

myTaxiService is a multi-platform application, giving to the user the possibility to access the service through a website and a mobile app and with different devices. In order to match with this feature the web page and the mobile application should be adaptable to any resolution and screen size, implementing a responsive design with mobile, normal and wide modes. Of course, website and application's graphic should resemble to the same one, in way to make it easier for the users to get used to the interface.

2.1.2 Hardware interfaces

The application interfaces with device's geolocalization services in order to get the position of the user. Position can be obtained from a GPS module or through a less precise IP location.

2.1.3 Software interfaces

As a base system from which obtain city map, localization, route finding and waiting time myTaxiService relies on Google Maps service. Therefore, we use Google Maps API v3. Instead for managing all the data required from the application has to be used a DBMS.

2.1.4 Communication interfaces

For using the application the device requires an active Internet connection, connected even in LAN or on the Mobile Network.

2.2 Product functions

- Request a Taxi. The system permits the users to call for a taxi by specifying his actual position. The first taxi driver available for the user's zone

will be advised and briefly a taxi will accomplish the task.

- Queue management. Taxi drivers are allocated into a queue, one for each zone of 2 square kilometres, managed as a FIFO structure. When a request from a user arrives at the system the first driver is notified, and can accept or refuse it; in this case, he will be removed and inserted at the end of the queue. As a driver can move along the city without being in service, when a request arrive at the system all the queues are realigned, in order to give to the user the best choice in terms of nearness.

- Booking request. Alongside the simple and immediate taxi request, this service gives to users the choice to book for a ride more than two hours in advance. This booking request will be stored by the system, which 10 minutes before the desired time will advise the most convenient taxi driver.

- Expandability. The system grants access to a series of programming interfaces that permits to developers to extend the service features and to integrate the system in other third party applications.

- Unforeseen management. Even if a taxi driver gets stuck in traffic at rush hour, or, for example, have to manage a flat tire, the system re-allocate the user request to another taxi driver, after an unforeseen notification sent by the driver.

2.3 User characteristics

A typical user expected to use this application is a person who probably use the taxi service regularly, and wants to turn his shifts easier and clever to be booked. The user should be able to access The internet via his personal computer or his smartphone and should be used to booking systems. No particular experience or knowledge is required. Regarding the Taxi Driver instead, he should access the application using his smartphone, keeping it under control very often.

2.4 Constraints

- Taxi's position is computed through GPS localization, which is very sensible to the urban context, giving less accurate or even incorrect data nearby high buildings (e.g. skyscrapers), underground parking or tunnels. According to this, there could be misalignments between the real taxi's position and the one reported by the application.

- In order to make the application available for almost all the possible users, it must work on several web browsers and mobile OS.

- Alongside with the GPS problem, also the Internet connection could be fluctuating, not permitting a stable and consistent connection to the server.

- In order to make the application expandable and reusable by other services, some APIs are given to developers, who can use it for implementing new features and build additional services.

2.5 Assumptions and dependencies

- The first and most important assumption we make declares that once a user submits a request to the system, it is guaranteed that a taxi will fulfil his request. In other words, we assume that a user can never be left stranded; a taxi will always give him a shift.

- In addition, users that booked a ride in advance are ensured to get a taxi right at desired place and time. 10 minutes before the designed time the first taxi in the respective zone is appointed to the requesting user, and by means of assumption number one there always be a taxi to be assigned to the booked ride.

- Along with this web service, a taxi can be taken or booked in "traditional" modes, for example by phone or directly on the road. The application so has to manage this possibility, and while a driver accepts a ride not coming from the system he has to switch his state from "free" to "busy".

- We assume that once the user receives the confirmation notification reporting waiting time and taxi code, he will wait for that taxi, allocated to him.

- Personal information given by the user during the subscription are correct and truthful.

- When an unforeseen occurs, the taxi driver notifies it to the system. Indeed only if the taxi driver does this the system can reassign the request to another driver and update the information for the user.

- The system refuses to manage a new request coming from a user if another request from the same user is pending.

- In order to not permit identity theft for taxi drivers, they are provided of a one-time-password that we assume they get by email before application launch; in this way during taxi driver's registration the system can check the visitor identity.

- Because of the immediacy of a demand, we assume that a user would never have to edit or delete this type of request, so these opportunities are given only for reservations.

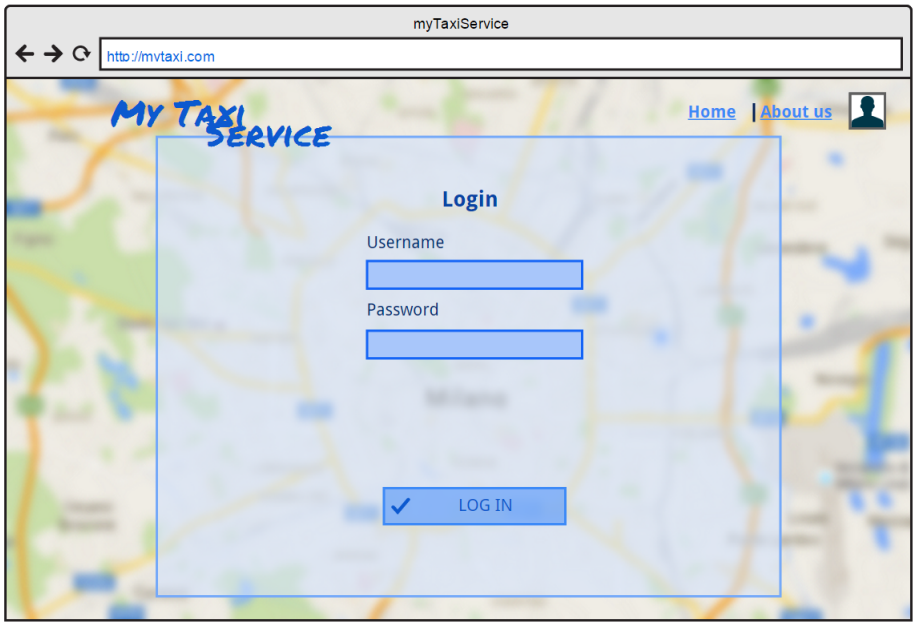
3 Specific Requirements

3.1 External Interface Requirements

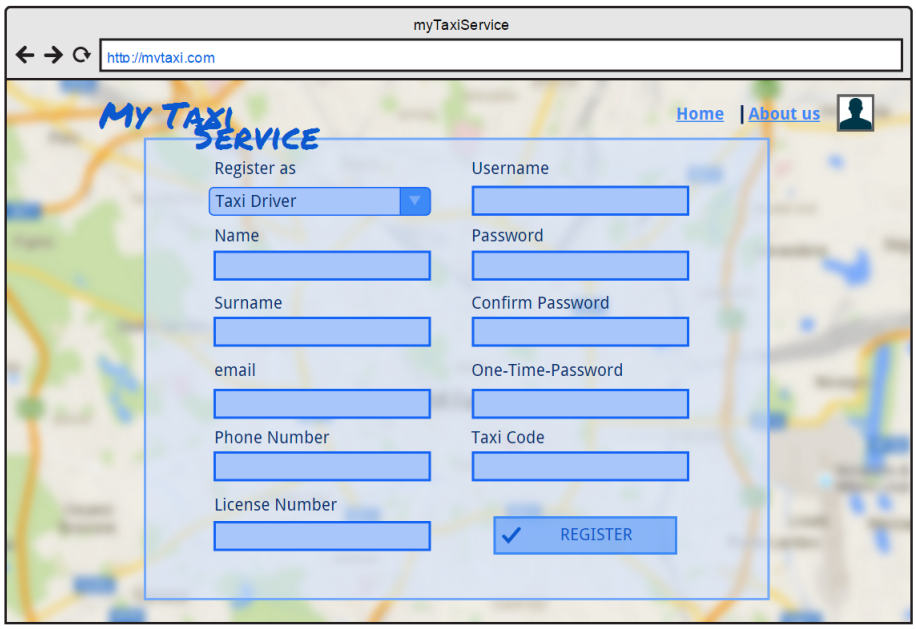
3.1.1 User interfaces

Here are presented some mockups which represent as we projected the Graphic User Interface. Being a multiplatform application the main goal is to preserve a shared graphical language which can guide the user also in different uses and devices.

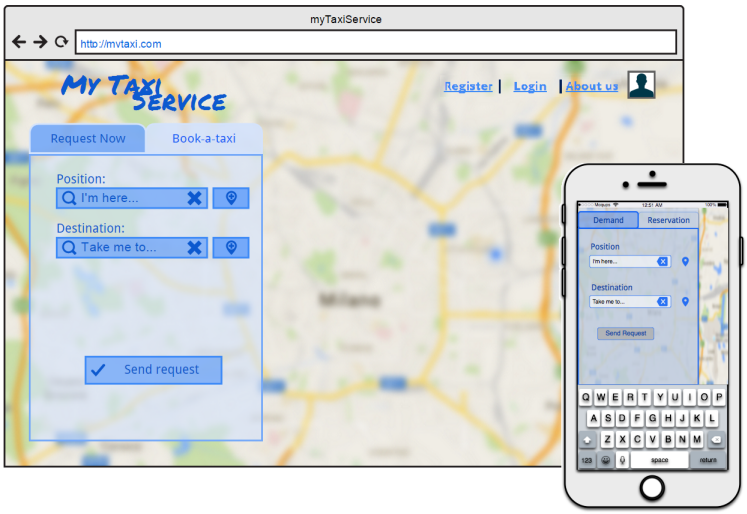
Login In this page a registered user can log in to the application by inserting his credentials. Once logged in, the user is redirected to the service home page.

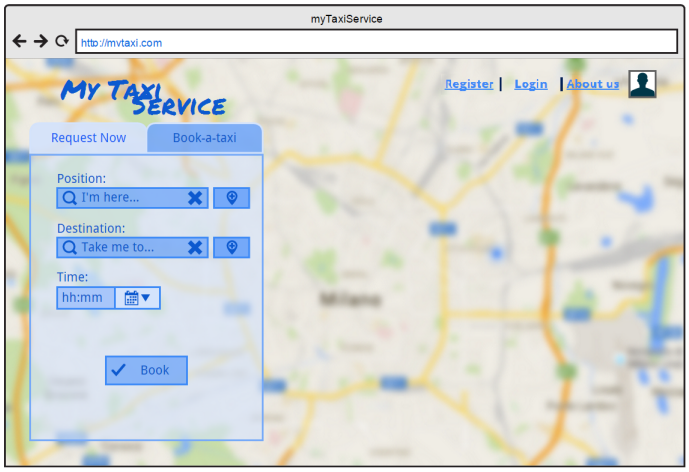


Registration Here we designed a registration form, from where either a client or a taxi driver can subscribe to the service.

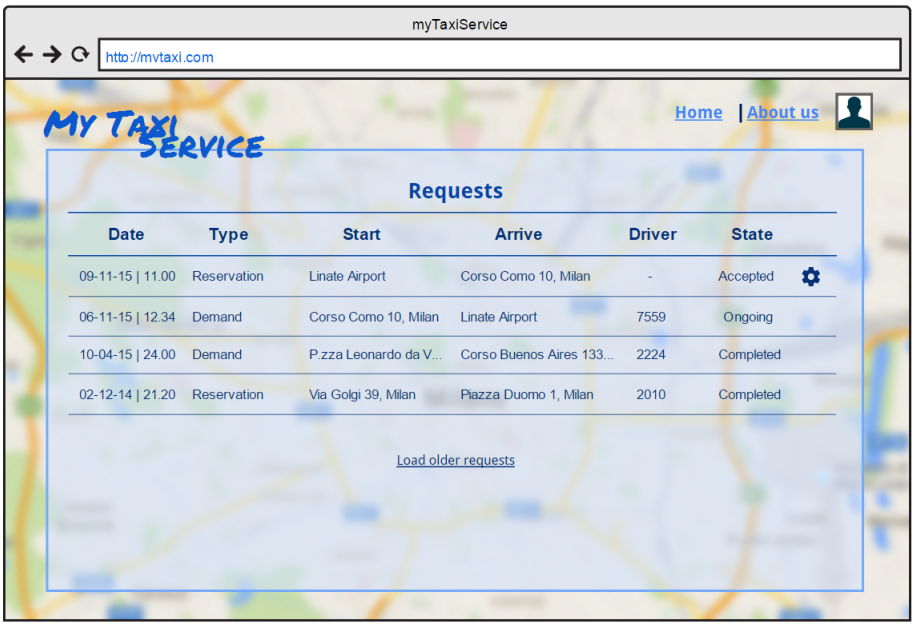


Home page This is how the application looks to a logged user. From here he can forward a request to the system, either a demand or a reservation, by entering all required informations.





Requests In this page a user can access the history of his requests, with all their informations. For the accepted reservations he also can access the editing form, which permits to change the locations and the time gived during the first forwarding.



Notifications Here we present as a notification arrives to the user using the system from mobile. In the first picture we see a notification with all the informations sent to a client that has recently requested a taxi ride. In the second picture instead there are the ones sent to the taxi driver, which can interact with the system by accepting or refusing the request.

3.1.2 API interfaces

In way to implement a map system for the application we use Google Maps API. These permit us to gain a localization service integrated with a lot of additional functions, like traffic state and path planning with ride duration; all these features let the application give sharp waiting time for clients and taxi drivers. More information and documentation about functionalities and quality of service are available at Google Developers website.

3.2 Functional Requirements

3.2.1 [G1] allow a visitor to become a registered user

- [R1] visitors can just see the login page
- [R2] visitors can access only to the registration form
- [R3] if, during registration, the phone number or mail address provided are already present in the database of the application, the registration is denied
- [R4] visitor must complete all the mandatory fields of the registration form
- [R5] the visitor who wants to be registered as taxi drivers must provide their number of taxi license and the one-time password received
- [D1] the phone number and mail address provided must be correct

3.2.2 [G2] allow a visitor to log in to the application

- [R1] misinformation does not allow access to the account
- [R2] visitors cannot request a taxi before the login
- [R3] the application implements the retrieve password mechanism, using mail address provided in the registration
- [D1] Username and password must be inserted correctly

3.2.3 [G3] allow user to make a request of taxi service, both reservations and for demand service

- [R1] user must complete all the mandatory fields of the taxi request form
- [R2] a demand is rejected if there is already another not replied demand, performed by the same user
- [D1] Time must be included between 00.00 and 23.59
- [D2] Starting position and destination provided must exist

3.2.4 [G4] allow taxi drivers to communicate their availability

- [R1] taxi drivers must be in only one status between available and unavailable
- [R2] a taxi driver available that has an unforeseen, must declare himself unavailable without communicating to the system the unforeseen
- [R3] only taxi drivers unavailable but in service must communicate the unforeseen to the system

3.2.5 [G5] allow taxi drivers to accept or reject a request

- [R1] the taxi driver must be available
- [R2] taxi drivers must not both accept and reject a request
- [R3] taxi drivers must not receive more than one requests simultaneously
- [R4] the taxi driver who accepts a request must be set as unavailable

3.2.6 [G6] allow registered members to view their reservations and change them.

- [R1] must be viewed only his own reservations
- [R2] only the future reservations, ie those who have set the date that is later than the current one, can be edited
- [R3] all changes made must comply the requirements previously presented for the reservations of a taxi

3.3 Scenarios

3.3.1 Scenario 1

Mr. John is a businessman who has to move very often for the city of Milan for a job. On the local newspaper notes an advertising of the new app of the local taxi service and decide to download it. Once installed he notes that to make a request for taxis he has to be registered and thus proceeds to the compilation of the required fields. After entering all the mandatory fields, included phone number, mail address, username, and password, he confirms his registration. The system shall check all fields and if mail-address, username, and phone number are not already present in the database, and also if the password is formally correct, send an email of successful registration to the user. By this time, he will be able to log in by entering his username and his password in the login form.

3.3.2 Scenario 2

Jack is a taxi driver who received a mail from the municipality where he works. This mail shows the opportunity to improve the management of his job with the use of the new application launched. He decides to follow this hint and proceed to registration. In the registration form, in addition to the mandatory fields like username, password, mail address and phone number, he inserts the license number and the code received in the mail mentioned. By the monitoring of the inserted fields, the system sends the registration's confirm to Jack's mail address. From this time, he will be able to receive taxi service's requests by the application.

3.3.3 Scenario 3

Margherita is a girl of 55 years old and she is already registered to myTaxiService. She is at home and she wants to check if she has inserted the correct time in her reservation made. Then she goes on the web page of myTaxiService but she has forgotten her password. So she clicks on the button "Recovery Password" and she inserted the mail used in the registration. After few minutes, she received a mail from the system with the password forgotten. At this point, she goes back to the login page and inserts her credentials, to enter in her private page. In this page, she can see all her information. So she selects "Historical reservations" and a page is displayed containing all the reservations made, both the past ones and the future ones. At this point, she can check the time given in the reservation for that afternoon. She realizes that she made a mistake because she inserted 17.00 as starting time instead of 16.00. So she clicks on the button "edit reservations", selects the reservation in question and all the field become editable. She changes the time's field and inserts 16.00, but when she clicks on "confirm", the system denied this modification because it is already 15.00 and changes are available until two hours before the starting time. In the end, she decides to maintain the previous starting time.

3.3.4 Scenario 4

Steph arrives at Garibaldi station at 19.00 and he wants to go to the stadium to see his favorite team play. The subway is temporarily unavailable, so he decide to demand a taxi with the myTaxiService app. After the login, he goes to request's page and inserts "Garibaldi" in the field "Position" and "San Siro stadium" in the field "Destination" and click on the button "send request". The request is sent to the system that, as first operation, reschedules all the queues of taxis based on the position detected by their GPS. Then sends the service's request to the first taxi driver of the queue in the area that contains Garibaldi's station. James, an available taxi driver located near Garibaldi, is the first in the queue and receives the request. He decides to accept it and so he confirms this to the system with the appropriate button. Then the system sends a confirmation message to Steph with the James's taxi's code and a waiting time of three minutes.

3.3.5 Scenario 5

Carlos is a 45 years man who lived in via Monte Rosa, near Lotto. At the morning of the 1st November, his boss informs him that at 13.00 he should participate to a meeting near Expo Rho-Fiera. Considering that he has not a car and he wants to make sure to arrive on time, he decides to reserve a taxi using myTaxiService. After the login, he goes on Reservation's page and inserts "Expo Rho-Fiera" in the field "Destination", "Via Monte Rosa" in the field "position" and selects 12.15 as starting time and at the end presses on "send request" button. The system receives the request and accepts it immediately, sending the confirmation message to Carlos. At 12.05 the system schedules all

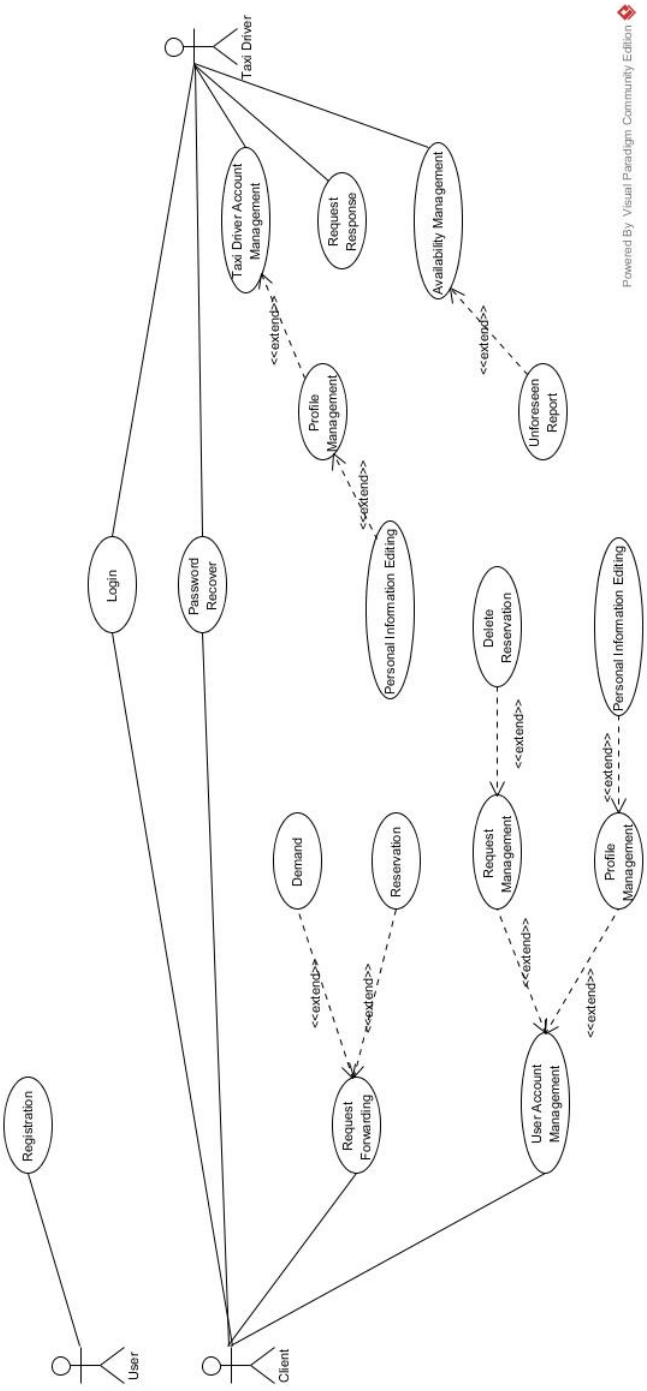
the queue based on taxis' GPS and send the request to the first taxi driver in the queue of the area contains via Monte Rosa. The first taxi driver denied the call because he wants to move in another area soon. So the system moves this taxi driver in the back of the queue and sends the request to the next one. Finally, this taxi driver accepts the request and the system communicate to Carlos 70 as taxi's code and the waiting time of four minutes.

3.3.6 Scenario 6

Mr. Antoine is a taxi driver of Milan who is already registered in myTaxiService. At the morning of the 3rd November start to work. As first operation, he opens myTaxiService's app and he logs in. Then he moves in Lotto's zone and communicates to the system his availability to receive requests using the appropriate button. From this moment, he is available for the system. At 9.30 a demand arrives at the system from a user located in Lotto's zone. The application informs Mr. Antoine, who is the first in the Lotto's zone's queue, who decides to accept it. So he received a message from the system with the client's position. The system also sets him as unavailable. So Mr. Antoine moves to the reported position. During the journey, unfortunately, he blows a tire and he is forced to stop to change it. Understanding that he will never reach the client in a due time, so he decides to inform the system of the impossibility to complete the task, through the appropriate button. The system reschedules the queues and contacts the first taxi driver of Lotto's zone's queue. When he accepts the request, the system communicates the client the unforeseen, sending him a new message with the new taxi number and the new waiting time.

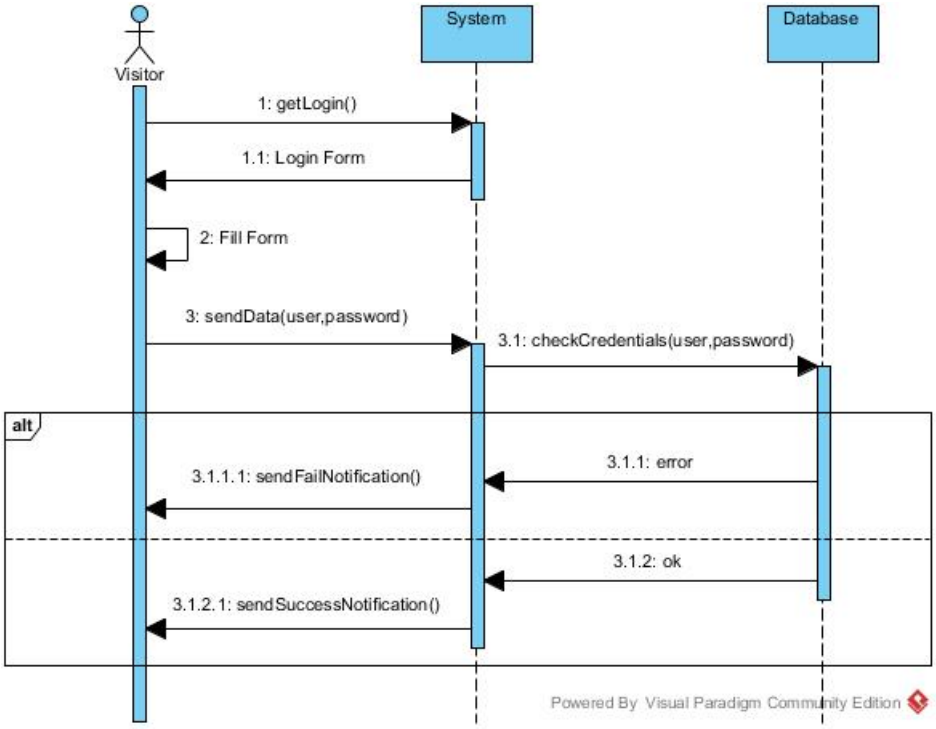
3.4 UML Models

3.4.1 Use case



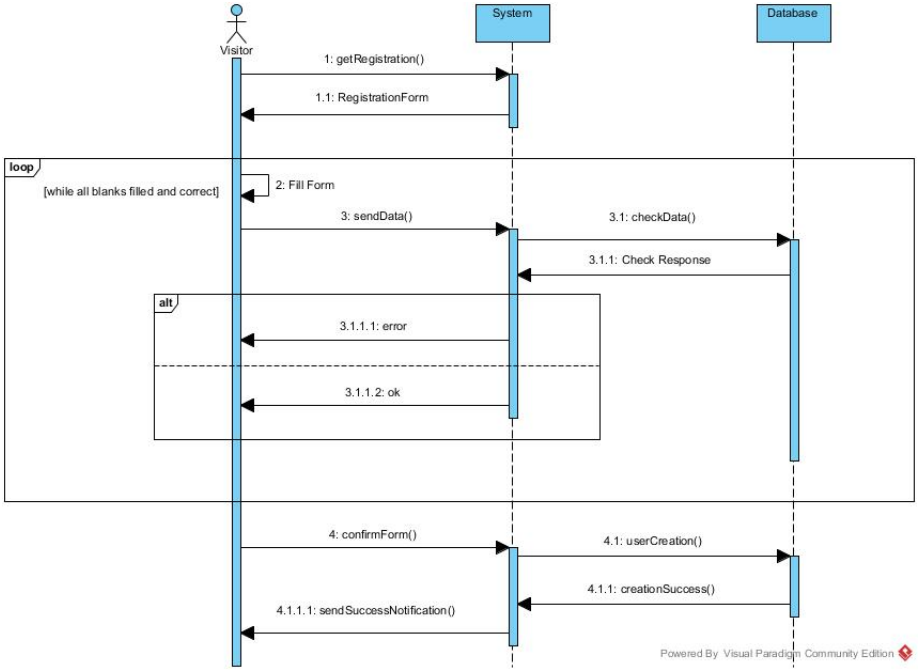
3.4.2 Sequence diagrams

Login



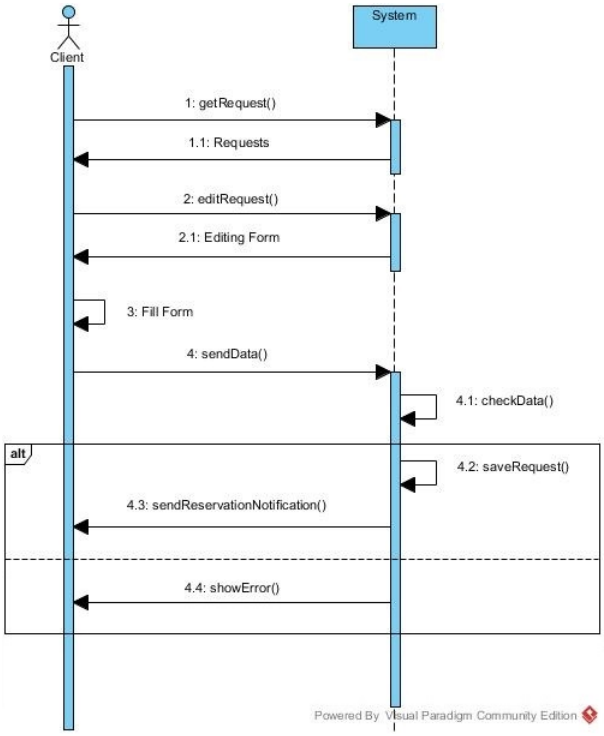
Actors	Registered users
Preconditions	Registered user has successfully signed up to the system
Events flow	1. The registered user opens the home page of the application 2. The registered user inserts his username and his password in the login form 3. The registered user clicks on the button "login" 4. The system redirects the registered user to his private page
Post-conditions	Registered user is logged in
Exceptions	Either username or password is invalid. The system notifies the error and suggests registered user to recovery password

Registration



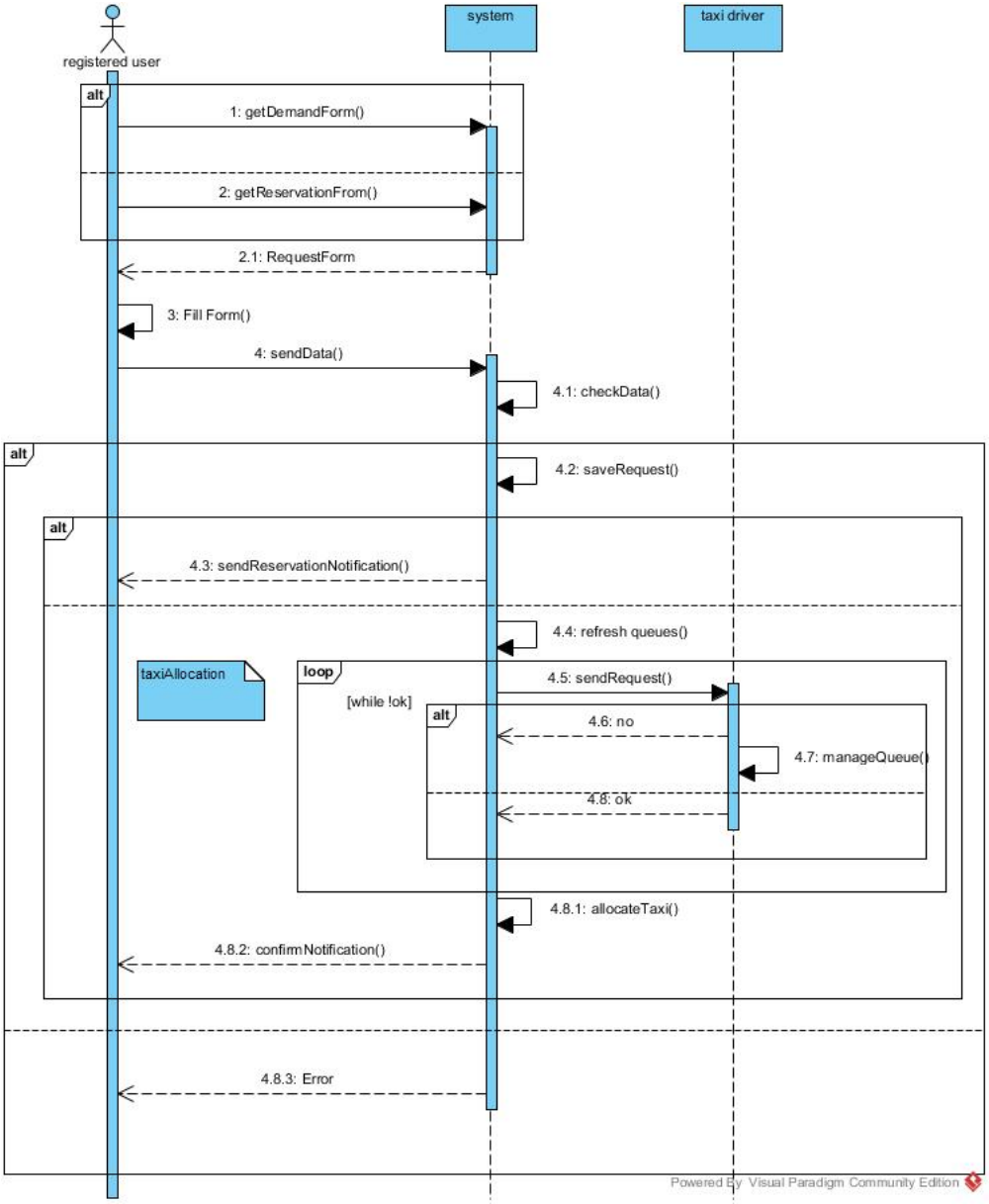
Actors	Visitors
Preconditions	Visitors are not already signed up
Events flow	1. The visitor reaches the registration page 2. The system requires the visitor to fill all mandatory field like name, surname, username, mail address, phone number, password and confirm the password 3. The visitor fills fields required and press on "Confirm Registration" button 4. While entering the fields, the system checks the format and if the information entered is already present in the database 5. The system sends an email to the mail address provided to notify the correct registration 6. The system notifies the registration and allows the new registered user to go to his personal page
Post-conditions	The user is signed up and logged in
Exceptions	The email, username and phone number are already used. The second password does not match to the first one. In this case, the visitor can't complete the registration step. During entering of the information, the system notifies the error alongside the incorrect field. If there is an incorrect field, the visitor can't proceed to confirm registration.

Edit request



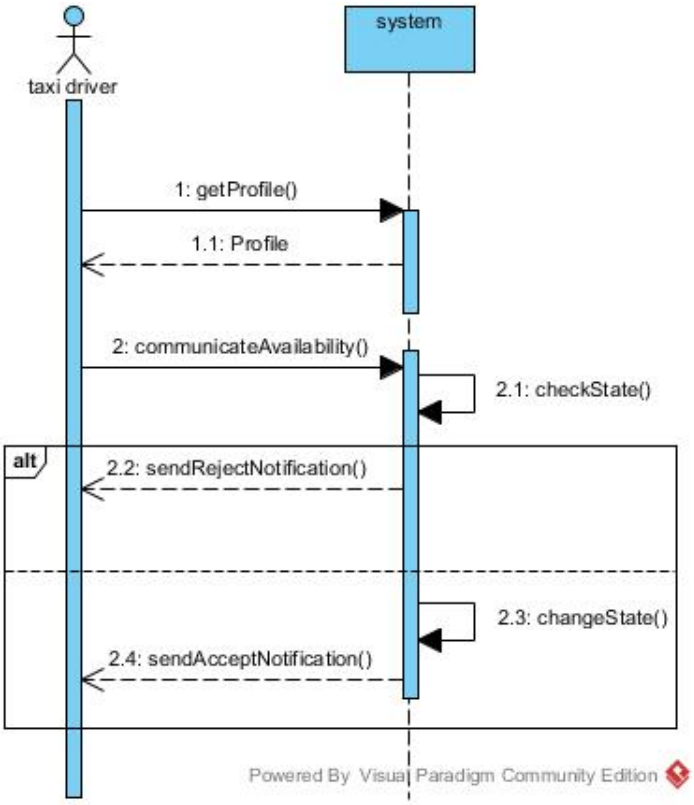
Actors	Registered user
Preconditions	The user is already logged in and he has already made at least one future reservation
Events flow	1. Registered user reach his personal page and selects "Requests" 2. The system shows all requests made, including future reservations 3. Registered user click on the edit button 4. The system redirects to a page where the registered user can change the information about the reservation 5. The user changes the information that he wants to change and click on "Confirm" button 6. The system shows the request page with the updated information.
Post-conditions	The request information has been updated
Exceptions	The registered user changes starting time of the reservation, but he inserts a starting time that is less than two hours ahead of the current time, or even a past time. The registered user inserts a past date. The system notifies the error and doesn't allow to registered user to make those changes

Request forwarding



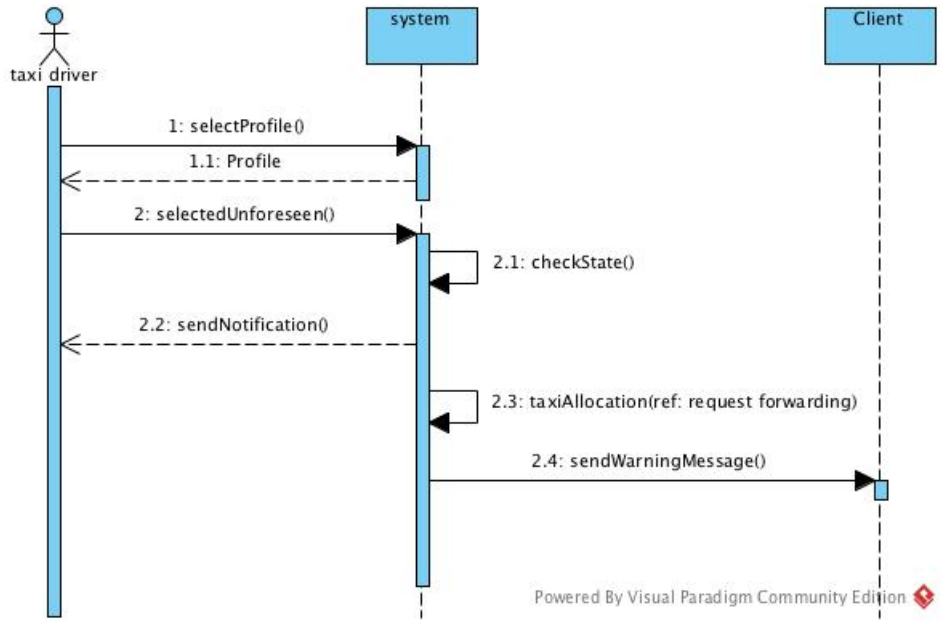
Actors	Registered User and Taxi Driver
Preconditions	Registered user is logged in and taxi driver is logged in and available
Events flow	1. Registered user clicks on "Make a request" button 2. Registered user selects "Reservation" or "Demand" 3. The system shows the required form page showing the fields to fulfil, such as "Position", "Destination" and "Date and Time". This last field is showed only in the reservation form 4. The registered user types the required information 5. Registered user clicks on "Send Request" button 6. The system checks the data 7. If it was a reservation, the system sends a Reservation notification to the user, while if was a demand the system proceeds to allocate a taxi: 1) The system refreshes the queues 2) The system sends a request message to the first taxi driver in the queue of the required zone 3) If taxi driver rejects the request, the system manages the queue, setting him as last in the queue, and redirects the request to the next taxi driver 4) Point 3 has been made until one taxi driver accepts the request 5) The system assigns the taxi to the client and sends him a notification
Post-conditions	Request has been accepted and a taxi driver is assigned to complete this task
Exceptions	"Position" or "Destination" doesn't exist. "Date and Time" is not formally correct. In these cases, the user can't complete the request step. The system notifies the error and prepares the user to reintegrate the information of the request

Taxi Driver availability management



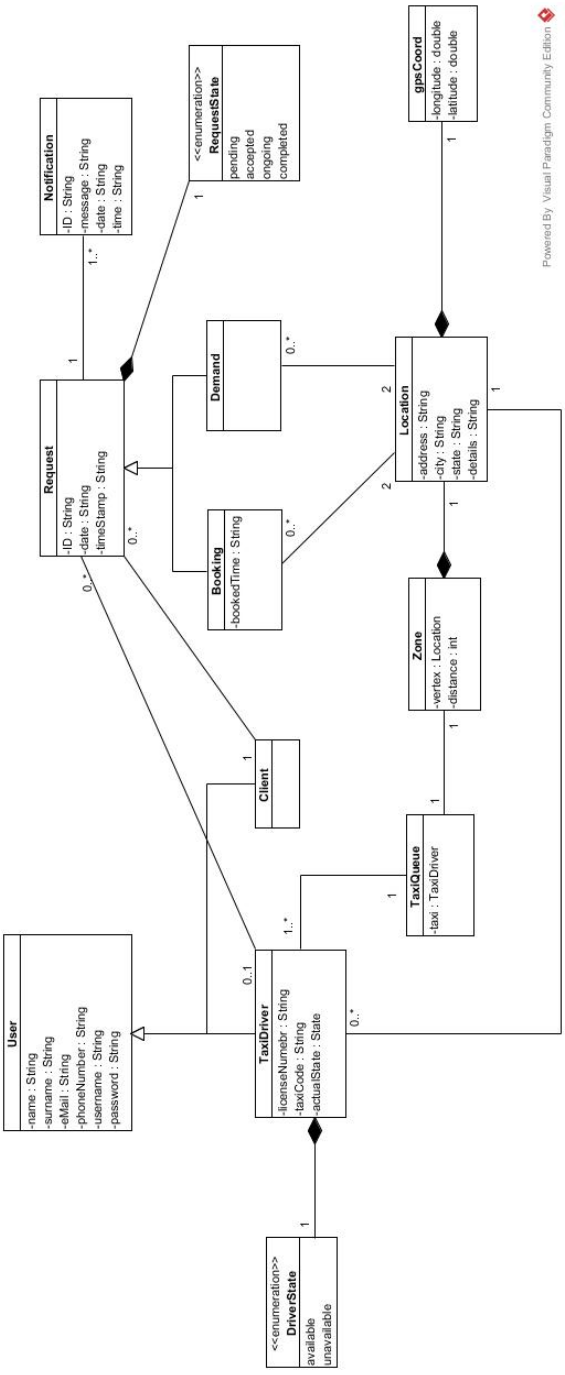
Actors	Taxi Driver
Preconditions	Taxi driver is already logged in
Events flow	1. Taxi driver reaches his own personal page 2. Taxi driver selects "available" button or "unavailable" button 3. The system checks the state 4. The system changes the state of the taxi driver 5. The system sends a confirmation notification
Post-conditions	Taxi driver's availability is changed
Exceptions	If taxi driver selects the same state in which is already in, this change is not applicable

Unforeseen management



Actors	Taxi Driver
Preconditions	Taxi driver is already logged in
Events flow	1. Taxi driver reaches his own personal page 2. Taxi driver clicks on "Unforeseen" button 3. The system sends a notification to the taxi driver 4. The system allocates a new taxi, proceeding as in the "Request Forwarding" diagram 5. The system sends a warning notification to the client
Post-conditions	Taxi driver is relieved from his task and a new taxi driver has been assigned to complete this task
Exceptions	Taxi driver's state is available. Taxi driver is not assigned to a request. The system sends an error notification and invites the taxi driver to change his state without communicating an unforeseen

3.4.3 Class diagram



3.5 Non-Functional Requirements

Non-functional requirements are those not directly related to specific functionalities of the system but refer to its quality, the so-called Quality of Service (QoS) attributes.

3.5.1 Reliability

The system must guarantee all the required functionalities under several different situations. In way to achieve this requirement each operation which need to edit database entry, like requests or operations on user's account, must operate with a transactional model, preventing data loss and bearing connection problems.

3.5.2 Availability

As long as city taxi service is available 24/7, also myTaxiService aims to be available every day and every night, so we claim a 99.9% of uptime. In case of extraordinary maintenance, it would be announced in advance and would be held during minimum flow periods esteemed by users habits.

3.5.3 Performance

As myTaxiService aim to be a clever and swift system to request a taxi ride, rather than a classic phone call, it must ensure a high-performance interaction between user and system. Either users or taxi drivers must be capable of employ any system's functionality and receive an acknowledgement utmost in 10 seconds. Only operations which request a realignment of taxi's queues can take a longer time frame, anyhow not higher than 60 seconds.

3.5.4 Security

myTaxiService uses an encryption system for preventing data stealing, in particular for data exchanged on networks. So user's information are secure, and these could be clear accessed only through username and password. Users can only access to their data and to functionalities regarded to their personal role.

3.5.5 Portability

Portability requirement does with easiness to port the software to other hardware or software systems. This goal, relating back-end layer, is reached by means of a portable language as J2EE; relating the client application portability goal is achieved developing a web based application.

4 Appendix

4.1 Alloy

In this section is presented the Alloy modeling that we used to verify the consistency of our class model. We represented its classes and relations by means of Alloy signatures and facts. In the end of the section are present some worlds generated by the following code; this worlds show up the generated entities and their consistency with our modeling.

4.1.1 Data Types

```
module myTaxiService

//DATA TYPES

sig Text{}

sig Double{}

enum DriverState {
    available,
    unavailable
}

enum RequestState {
    pending,
    accepted,
    ongoing,
    completed
}

sig gpsCoord {
    latitude : one Double,
    longitude : one Double
}
```

4.1.2 Signatures

```
//SIGNATURES

abstract sig User {
    name : one Text,
    surname : one Text,
    email : one Text,
    phoneNumber : one Text,
    username : one Text,
    password : one Text
}

sig TaxiDriver extends User {
    licenseNumber : one Text,
    taxiCode : one Text,
    state : one DriverState,
    position : one Location
}

sig Client extends User {
}

abstract sig Request {
    startingLoc : one Location,
    arriveLoc : one Location,
    ID : one Text,
    date : one Text,
    timeStamp : Text,
    client : one Client,
    driver : lone TaxiDriver,
    state : one RequestState
}

sig Demand extends Request {
}
sig Booking extends Request {
    bookedTime : one Text
}
```

```
sig Notification {
    ID : one Text,
    message : one Text,
    date : one Text,
    time : one Text,
    request : one Request
}

sig Location {
    coord : one gpsCoord,
    zone : one Zone,
    address : one Text,
    city : one Text,
    state : one Text,
    details : one Text
}

sig QueueNode {
    next : lone QueueNode,
    taxi : one TaxiDriver
}

sig TaxiQueue {
    root : lone QueueNode
}

sig Zone {
    vertex : one gpsCoord,
    distance : one Double,
    queue : one TaxiQueue
}
```

4.1.3 Facts

```
//FACTS

//Reject registrations that uses an email, username
// or phone number already registered
fact noDoubleUser {
    (no disj u1, u2 : User | u1.email = u2.email) and
    (no disj u1, u2 : User | u1.username = u2.username) and
    (no disj u1, u2 : User | u1.phoneNumber = u2.phoneNumber)
}

//Avoid creation of a request in which the starting location
//coincide with the arrival one
fact noSamePlaceForRequest {
    all r : Request | r.startingLoc!=r.arriveLoc
}

//ID is a unique value for requests
fact noDoubleRequest {
    no disj r1,r2 : Request | r1.ID=r2.ID
}

//Every notification refers only to one request
fact notifyToOneRequest {
    no disj n1, n2 : Notification | n1.request = n2.request
}

//Unless is pending, every request has at least one notification
fact atLeastOneNotification {
    all r : Request | some n : Notification |
            r.state!=pending and
            r=n.request
}

//Only completed or ongoing requests are related to a driver
fact driverForCompletedAndOngoing {
    all r : Request | (r.state=pending or r.state=accepted)
    iff no r.driver

}
```

```
//A Client cannot be related to more than one ongoing request
fact noClientsUbiquity {
    no disj r1,r2 : Request |
        r1.state=ongoing and r2.state=ongoing and
        r1.client=r2.client
}

//A driver can only be related to one ongoing request
fact noDriverUbiquity {
    all t : TaxiDriver | lone r : Request |
                         r.driver=t and r.state=ongoing

}

//An ongoing request is related to one unavailable driver
fact ongoingRequestOccupiesDriver {
    all r : Request | r.state=ongoing iff r.driver.state=unavailable
}

//Two gpsCoords have the same latitude and longitude value
//if and only are the same one
fact UniqueCoordsUniqueValues {
    all g1,g2 : gpsCoord |
        (g1.latitude=g2.latitude and g1.longitude=g2.longitude)
        iff g1=g2
}

//Two Locations have the same coords if and only are the same one
fact uniqueLocationUniqueCoords {
    all l1,l2 : Location | l1.coord=l2.coord iff l1=l2
}

//Two Zones has the same vertex and the same queue
//if and only are the same one
fact uniqueZoneUniqueQueue {
    all z1,z2 : Zone | (z1.vertex=z2.vertex or z1.queue=z2.queue)
                       iff z1=z2

}
```

```
//Taxi code and license are unique values for taxi drivers
fact noDoubleTaxi {
    (no disj t1,t2 : TaxiDriver | t1.taxiCode=t2.taxiCode) and
    (no disj t1,t2 : TaxiDriver | t1.licenseNumber=t2.licenseNumber)
}

//A Driver is not in a Queue if and only is unavailable,
//else is available and in a queue
fact driverInQueue {
    all t : TaxiDriver | all q : TaxiQueue | (t in q.root.*next.taxi)
                                              iff (t.state=available)

}
//There is always at least a driver available in each queue
fact AtLeastADriver {
    all q : TaxiQueue | some t : TaxiDriver | t in q.root.*next.taxi
}

fact allNodesBelongToOneQueue {
    all n:QueueNode | one q:TaxiQueue | n in q.root.*next
}

//A driver cannot be the next one to himself
fact nextNotReflexive {
    no n : QueueNode | n = n.next
}

//The queue cannot be cyclic
fact nextNotCyclic {
    no n:QueueNode | n in n.^next
}

//Two drivers are in the same node if and only are the same one
fact singleNodeToSingleDriver {
    no disj n1, n2 : QueueNode | n1.taxi = n2.taxi
}
```

```
//Driver position is related to his current zone and queue
fact driverPosition{
    all t : TaxiDriver | lone z : Zone|
        t.state=available iff (t.position.zone=z
                              and t.position.zone.queue=z.queue)
}

pred show{}
run show for 5
```

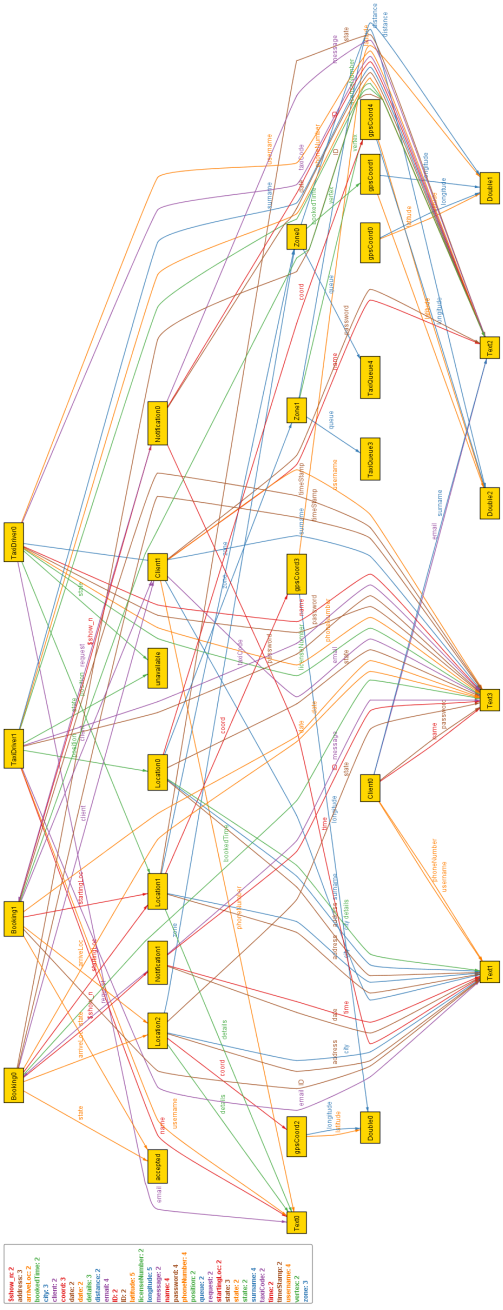
4.1.4 Result

```
Executing "Run show for 5"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   12051 vars. 1040 primary vars. 23068 clauses. 970ms.
   Instance found. Predicate is consistent. 409ms.
```

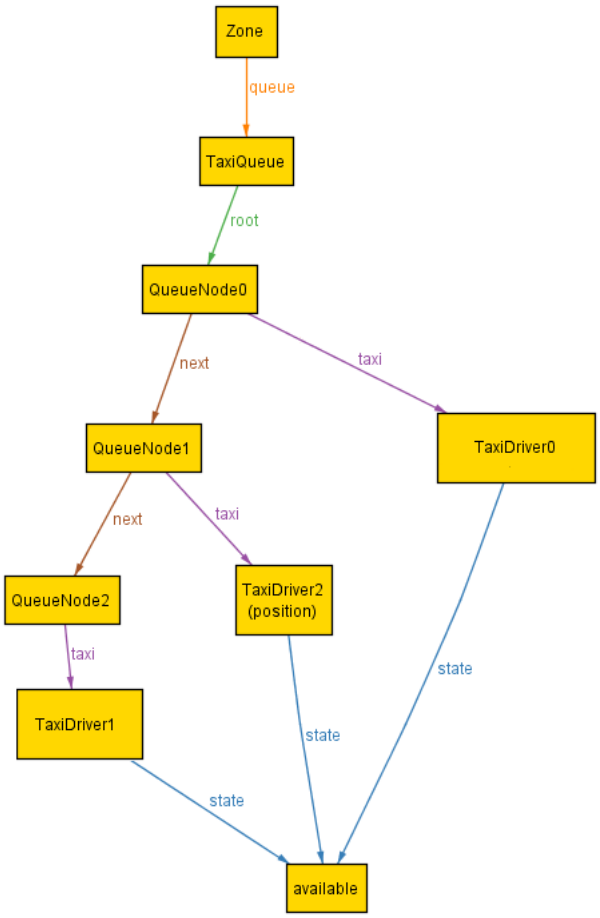
The result confirms the validity of our model. We insterted all the possible assertions we thought in the alloy code and we noticed that we didn't do some assumptions that are essentials for the consistency of our model

4.1.5 Generated worlds

Complete world



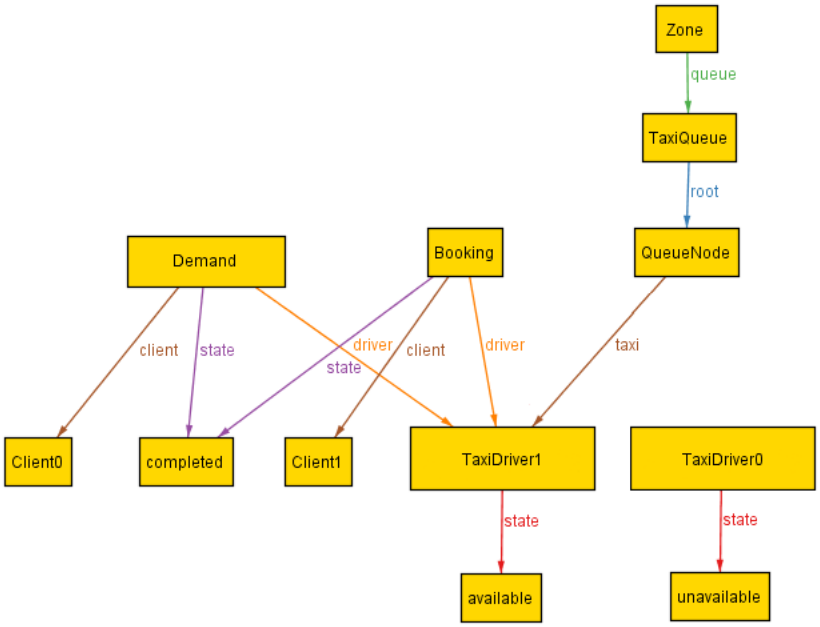
World 1



In this world we focused on the structure of a queue. In this world there are:

- 1 unique zone;
- 1 unique queue related to the zone. It is composed of 3 different nodes;
- 3 taxi drivers related to the 3 different nodes. Each taxi driver is available, in fact all the unavailable taxi driver are not present in any queue.

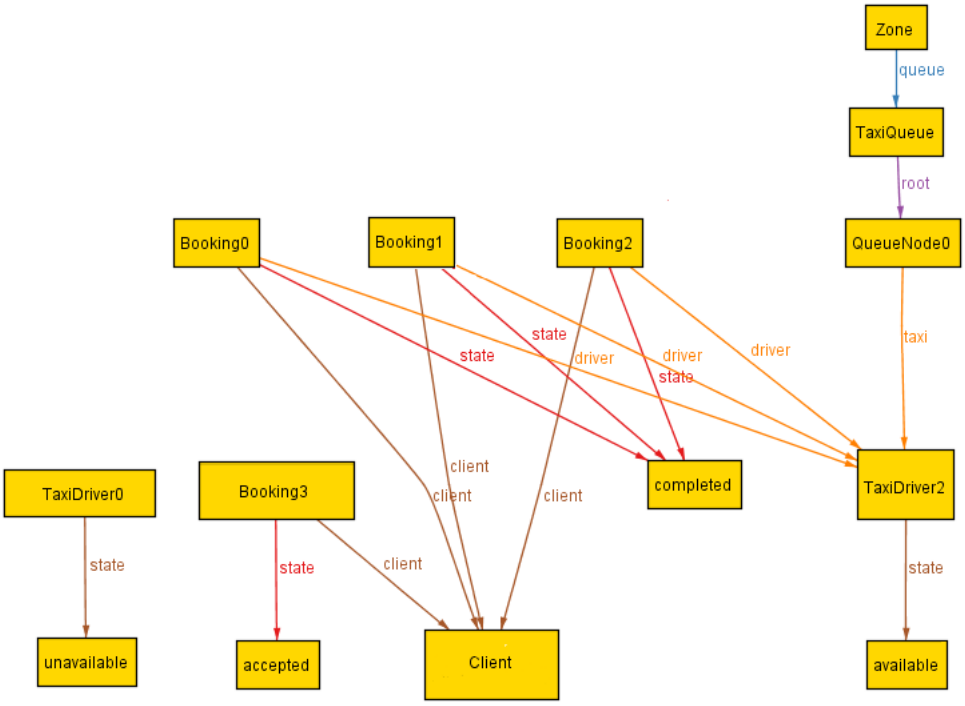
World 2



In this world we focused on the basic functions of the system and in particular on demand and reservation completed by the same taxi driver. In this world we have:

- 2 taxi drivers, which one available and the other unavailable;
- 2 different clients;
- 1 Demand related to a client and 1 reservation related to the other client. Both requests are completed by the same taxi driver, who is available now;
- 1 zone with its relative queue, that has the available taxi driver as first element. The second taxi driver doesn't belong to a queue because he is unavailable.

World 3



In this worlds we focused on numerous reservations management with different state. In this world there are:

- 1 zone with its queue, in which there is only one taxi driver, who is available;
- 4 different bookings, which 3 of them has been completed by the taxi driver contained in the queue, and 1 is only accepted;
- 1 client, who is related to all the bookings;
- 2 taxi drivers, which one of them belongs to a queue, and the other one doesn't belong to any queue, because his state is unavailable.

The accepted reservation is not related to any taxi driver because the taxi allocation phase as not started yet

4.2 Used tools

- *LyX:* to reduct and format this document
- *Alloy Analyzer 4.2:* to prove consistency of our model and related worlds
- *Visual Paradigm:* to create Use Case, Class and Sequence diagrams
- *moqups.com:* to create application's GUI mockups

4.3 Hours of work

Time spent for redacting this specification document:

- Andrea Turconi: ~29 hours
- Lorenzo Raimondi: ~29 hours