Requirements Analysis and Specification Document
Software Engineering Project

Federico Gatti [Matricola: 852377] e Luca Fochetta [Matricola: 792935]

November 2015

Contents

Chapter 1

Introduction

1.1 Purpose

This document is *Requirement Analysis and Specification Document* for **myTaxiService application**. This document describes the components of the system, functional and non-functional requirements, goals, domain properties and assumptions. It Also provides a description of the system using UML diagrams.
This document is intended to all developers and programmers who have to implement the requirements or who want to integrate other system with this one, and it could be used as a contractual basis between the customer and the developer.

1.2 Present System

Until now any software doesn't exist to manage the taxi requests.
The only existing way to request a taxi is to call the taxi call center that forwards it to an available Taxi.

1.3 Definitions, acronyms and abbreviations

We will speak about the user as a Passenger in this document.
A Passenger can be authenticated or not, so when we refer to him we don't specify it. When we want to refer to authenticated Passenger we specify it.
Taxi and taxi driver will be used as the same subject.

- PMA: Passenger Mobile Application
- PWA: Passenger Web Application
- PA: Passenger Application refers indiscriminately at PMA and PWA
- TMA: Taxi Mobile Application

- RASD: Requirement Analysis and Specification Document
- QTM: Queue and Taxi Manager
- DB: Data Base
- DBMS: Data Base Management System

1.4 Scope

The aim of the project is to create a software, called "myTaxiService", which can manage the queue of the taxi requests in a city.
The system is composed by 4 parts:

- 2 front-end applications used by the passengers called PMA and PWA
- 1 front-end application used by the Taxi Drivers called TMA
- 1 back-end application called QTM

The system will be able to suggest the best distribution of the taxi in the city zone to maximize passengers' satisfaction and the quality of the service.
A passenger who sends a request, using a web application or mobile application, can see from the application the *waiting time* and the code of the taxi that accepted the request. The passenger position can be determinate from GPS or if GPS information are incorrect, or aren't available, the passenger can insert manually the information of the location.
An authenticated passenger can reserve a taxi by specifying the origin and the destination of the ride. In this case the passenger must place the reservation at least two hours before the ride. The reservation request will be sent like a normal request ten minutes before the specified time. An authenticated user can always delete his reservation or modify it.
A Taxi driver can accept or reject a request.

1.5 Actors

- **Passenger** is the most general user of the application. A Passenger can forward a request from the application to the system in 2 different ways: using the Web application or the Mobile application. A passenger can know the *waiting time* and the *taxi ID* that answers positively at the request.
- **Authenticated Passenger** is a particular instance of Passenger. In addition to all Passenger features an authenticated user can place a taxi reservation at least 2 hours before the ride. An authenticated passenger can also deletes or modifies its reservation. There aren't limits to the numbers of reservations that a user can insert.
- **Taxi Driver** answers passengers' requests and he can choose to accept them or not. A Taxi Driver must respect system's decision.

1.6 Goals

1. A passenger can request a taxi by PMA
2. A passenger can request a taxi by PWA
3. A Taxi driver can accept or reject a request by TMA
4. A passenger authenticated can insert a reservation by PMA
5. A passenger authenticated can insert a reservation by Web application
6. A passenger authenticated can delete a reservation by PMA
7. A passenger authenticated can delete a reservation by PWA
8. To inform the taxi about a request
9. To inform passenger about the *Waiting time* and *Taxi ID*
10. A passenger can see all his reservations
11. The system must handle in a fair way taxis queue (the definition is in the next section)
12. The system must handle the distribution of taxis in the city (the definition is in the next section)

1.6.1 Queue management

The request is forwarded to the first taxi in the queue associated to the zone from which the request comes. If there aren't available taxi in the first queue the request is forwarded to the surrounding zones.
Handle taxis' queue in a fair way means that, when a taxi becomes Available, the taxi is inserted in the tail of the queue and when a request is sent to the system it is forwarded to the first taxi in the queue. If the taxi accepts the request it will be deleted from the queue, otherwise it will be deleted from the top and inserted in the tail.

1.6.2 Taxi Distribution manager

The system provides an algorithm to maximize the Taxi distribution in the city. Consider

- N identifies the number of *Available* taxis in the city
- z identifies the zone
- Z identifies the number of zone
- r_z identifies the number of taxis request in the zone z

The number of taxis in the zone z must be $Nr_z / \sum_{z=1}^{Z} r_z$.
The algorithm has a tolerance of 25%, in this way the case that the system must continuously command changes of zone to the taxis is avoided.
When the algorithm finds an incorrect distribution of the taxis in the city it has to inform the taxis that are located at the tails of the queues that have an overflow of the numbers of available taxis, maintaining a balance in the system. So if we identify with $a_{z,i}$ the Taxi *Available* in the zone z in the instant i and we have $a_{z,i} >= r_z$, in the instant $i + 1$ if there will not any requests in the zone z we can't have $a_{z,i+1} < r_z$.
The algorithm considers Taxi in *Transition* (see Functional Requirements) in the correct zone for computes the distribution, but the taxi isn't inserted in the queue until it reaches the competence zone.

1.7 References

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- Specification Document: myTaxiService Software Engineering 2 Project, AA2015-2016

1.8 Overview

This document is essentially structured in four part:

- Section 1: Introduction, it gives a description of the document and some basic information about the software.
- Overall Description, it gives general information about the software product with more focus about constraints and assumptions.
- Section 3: Specific Requirements, this part lists requirements, typical scenarios and use cases. To give an easy way to understand all functionality of this software, this section is filled with UML diagrams.
- Section 4: Appendix, this part contains some information about the attached .als file and some described screen-shot of the software used to generate it.

Chapter 2

Overall description

2.1 Product perspective

The system , as we had previously mentioned, is composed by four parts:

- **Passenger Mobile Application** is the front-end software used by the passenger. Users can install it in any device that respects the constrains (see software and hardware limitations). The Software is supported by iOS, Android and WindowsPhone. With PMA users can place a request, place a reservation, modify and delete a reservation. Passenger can sign up or sign in by the PMA.
  The Sign in is not necessary to use the PMA.
  PMA should be able to communicate with GPS to locate passengers position.

- **Passenger Web Application** is the front-end software used by the passenger. The Software is supported by every browsers that support Java. With PWA users can place a request, place a reservation, modify and delete a reservation. Passenger can sign up or sign in by the PWA.
  The Sign in is not necessary to use the PWA.
  PWA should be able to communicate with GPS to locate passengers position.

- **Taxi Mobile Application** is the front-end software used by the Taxi Driver. A Taxi Driver must be logged into the system to use the Application .
  Taxi driver can't Sign up using the TMA.
  The Software is supported by iOS, Android and WindowsPhone. TMA handles the request that comes from the passengers.

- **Queue and Taxi Manager** is the back-end software. All the requests originated by PMA and PWA are handled by the QTM that, after checking the queue of the corresponding zone, forwards the request to TMA. Also QTM notifies PMA or PWA when receives the answer by TMA.
  QTM computes and notifies to TMA the distribution of the taxis in the city.
  The QTM contains DBMS and Data Base system, so it contains all the data structures and the records of the system.

QTM establishes and communicates to the taxis the best distribution of them int the city.

2.2 User characteristics

There aren't required any specific educational level, experience and technical expertise for using the front-end application. All software interfaces are very user friendly.

2.3 Constrains

In this section are described all the constrains of the application.

2.3.1 Regulatory policies

The Passenger and the Taxi Driver must be agree with the software conditions to use it, in particular Users must be accept that the software can locate the position of the users.
Passenger and Taxi Driver must accept that the software can use cookies to optimize the performance.
Taxi Driver must accept to perform all the system decisions when is logged into the system.

2.3.2 Hardware Limitations

An Internet connection is required to use all the front-end services, in particular:

- A 3G connection for the PMA and TMA
- At least 2Mb/s connection for the PWA

2.3.3 Memory constraints

- PMA requires 128 MB of RAM and 200MB of HD
- PWA requires 256 MB of RAM and no space for the installation (omitting cookies)
- TMA requires 256 MB of RAM and 300MB of HD

2.4 Assumptions and dependencies

1. Accurate taxi locations are known by GPS
2. Accurate passenger locations of request is known by GPS at the moment if it is available

3. The Id Taxi is the same both in the system and in the real world
4. There is only one type of taxi that can carry at maximum 4 people
5. A Taxi driver respects the system decisions
6. A Taxi Driver communicates the status *Available* to the system when a passenger leaves the taxi
7. A Taxi Driver communicates the status *Available* to the system when Taxi Driver doesn't find the passenger
8. A Taxi Driver communicates the status *Out of Service* to the system when Taxi Driver ends the turn
9. A Taxi Driver communicates the status *Busy* to the system if a passenger takes a taxi without using the PA
10. A Taxi Driver communicates the status *Emergency* to the system when the taxi occurs in an accident
11. The system is notified by GPS when the taxi reaches its competence's zone
12. A Taxi drivers has always access to Internet
13. Once a taxi reaches its designated zone, it must not leave it.
14. A Taxi can move freely in its competence zone
15. When a taxi completes a ride it always goes back to its competence zone
16. Every request is correctly forwarded
17. City is divided in zones
18. There is a single queue for each zone
19. The system doesn't forward the request from out of town

2.5 Possible Future Extension

A possible future extension will be able to provide different types of taxis. Users will select the number of passengers at the moment of the request and so the system will be able to forward the request to the taxis that have the requested capacity.
A future extension will be able to provide to insert a field *Telephone Number* in the PA, in this way the taxi will be able to phone if it doesn't find the passenger.
Another extension could be the replacing the variable r_z in the QTM algorithm into $r_z(t)$. This could allow the algorithm to predict a different distribution depending on the hour of the day.
All the requests could be also stored to improve $r_z(t)$ variable.

Chapter 3

Specific Requirements

In this chapter we want to analyze the requirements of our application, to prove that they ensure the satisfaction of the goals introduced in the previous section, according to the domain properties.

3.1 External Interface Requirements

3.1.1 User interfaces

Some meaningful MockUps are of PA and TMA are illustrateed in this section

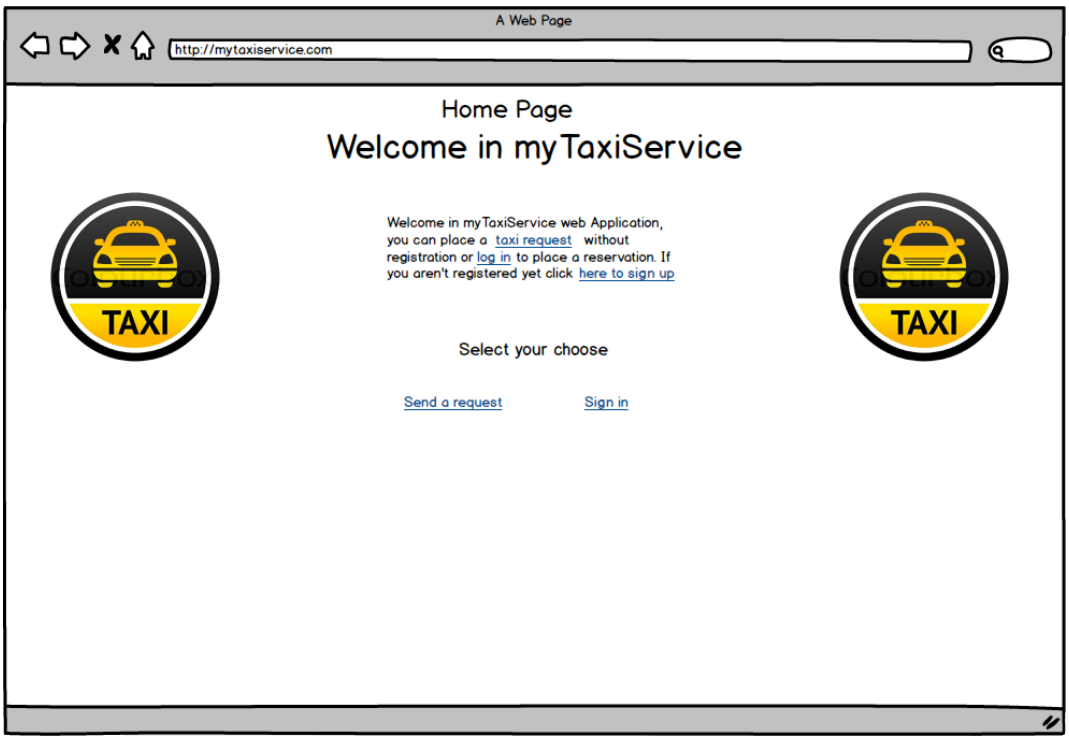
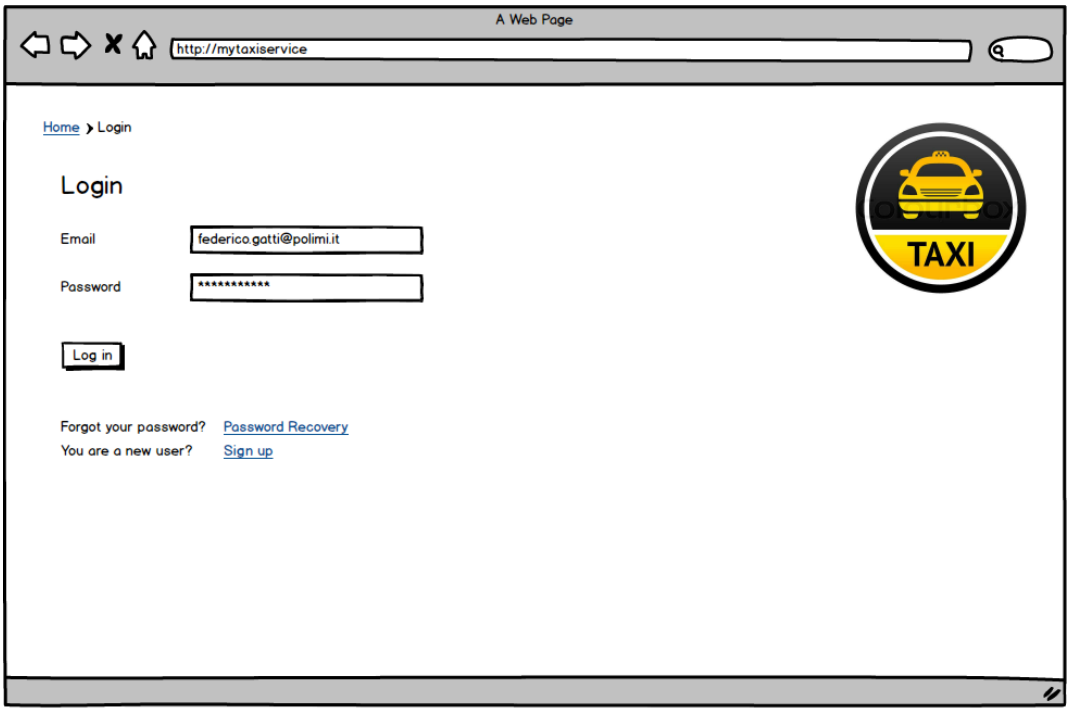


Figure 3.1: Home Page



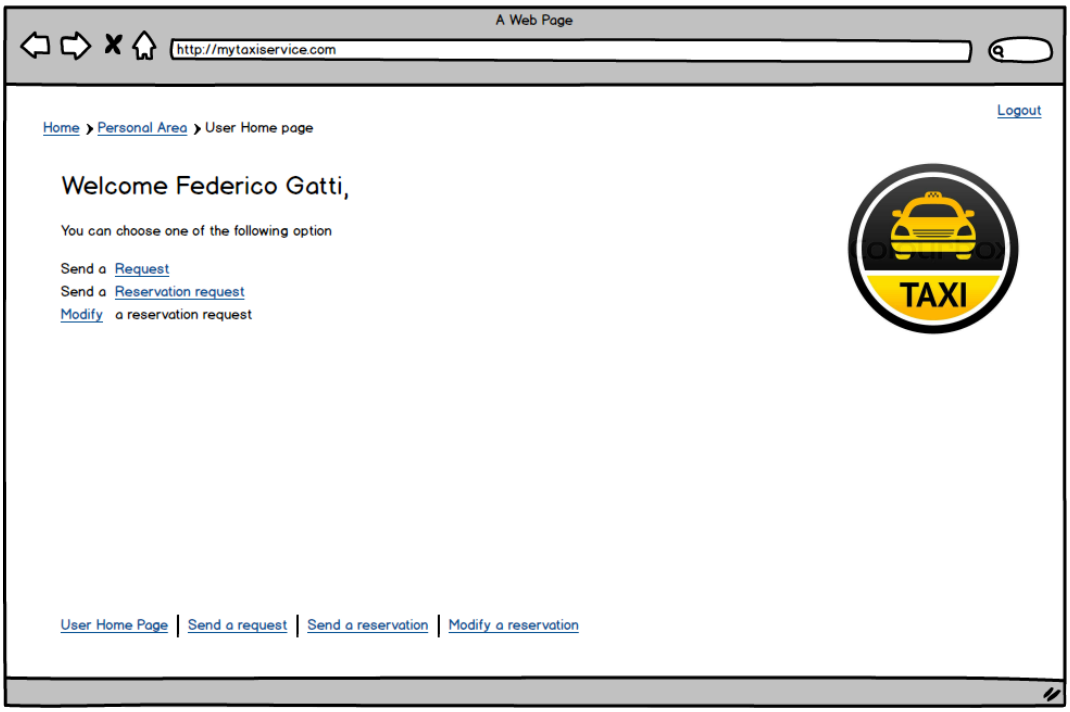
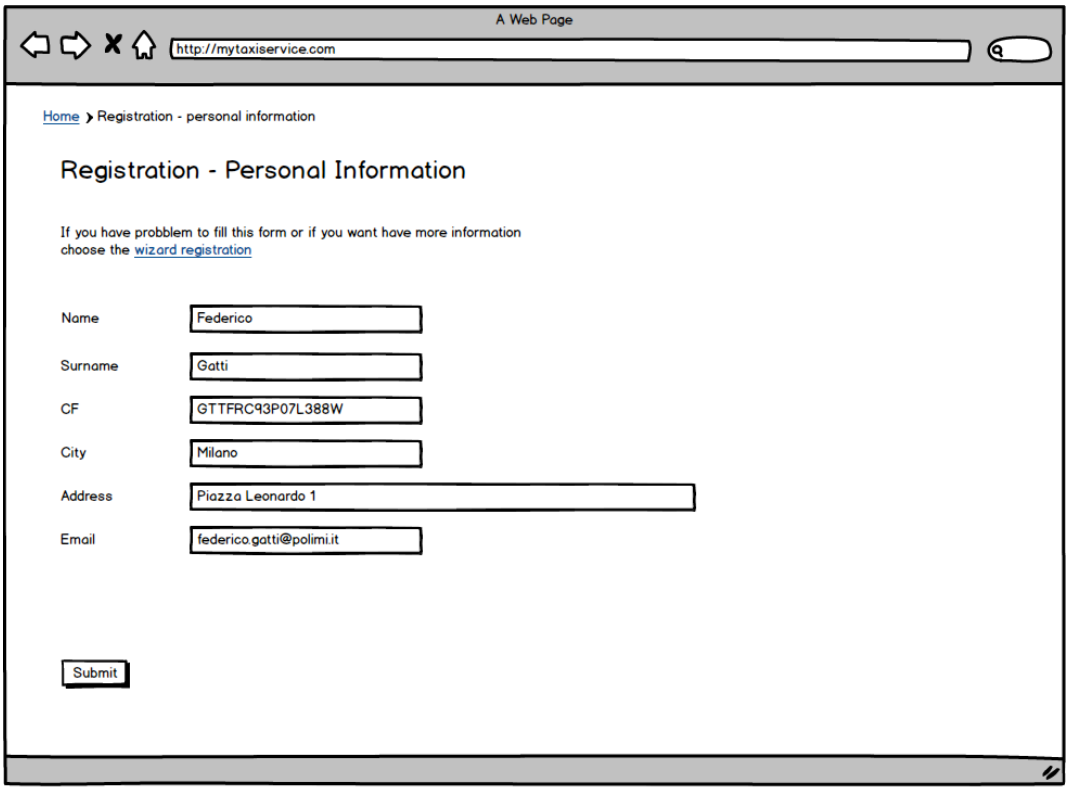


Figure 3.2: SignIn



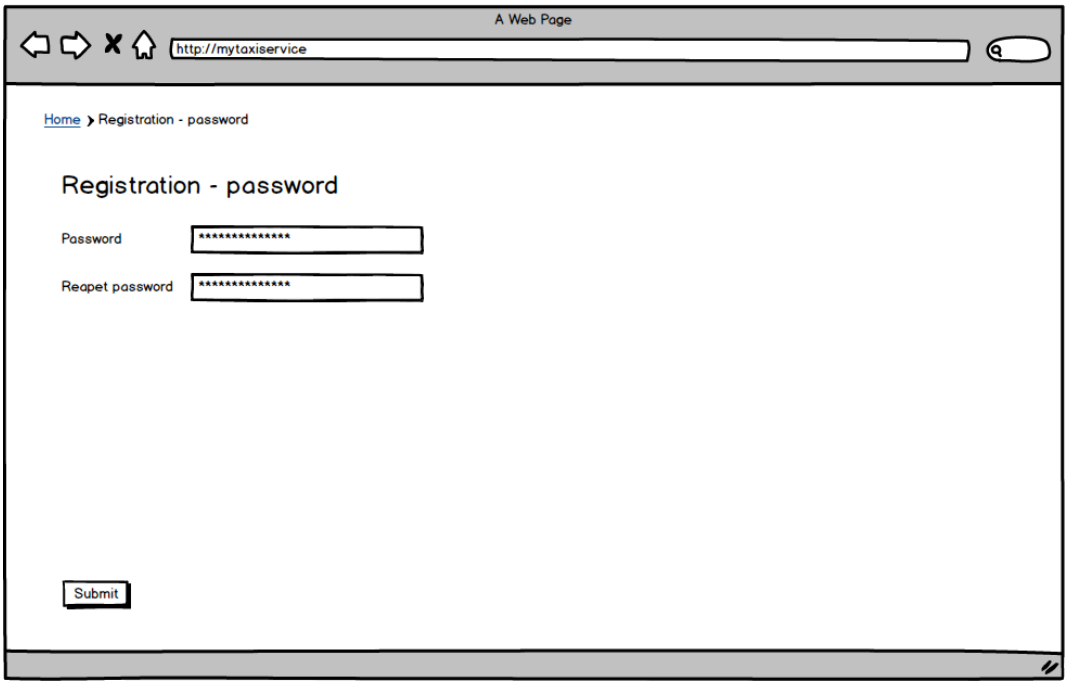
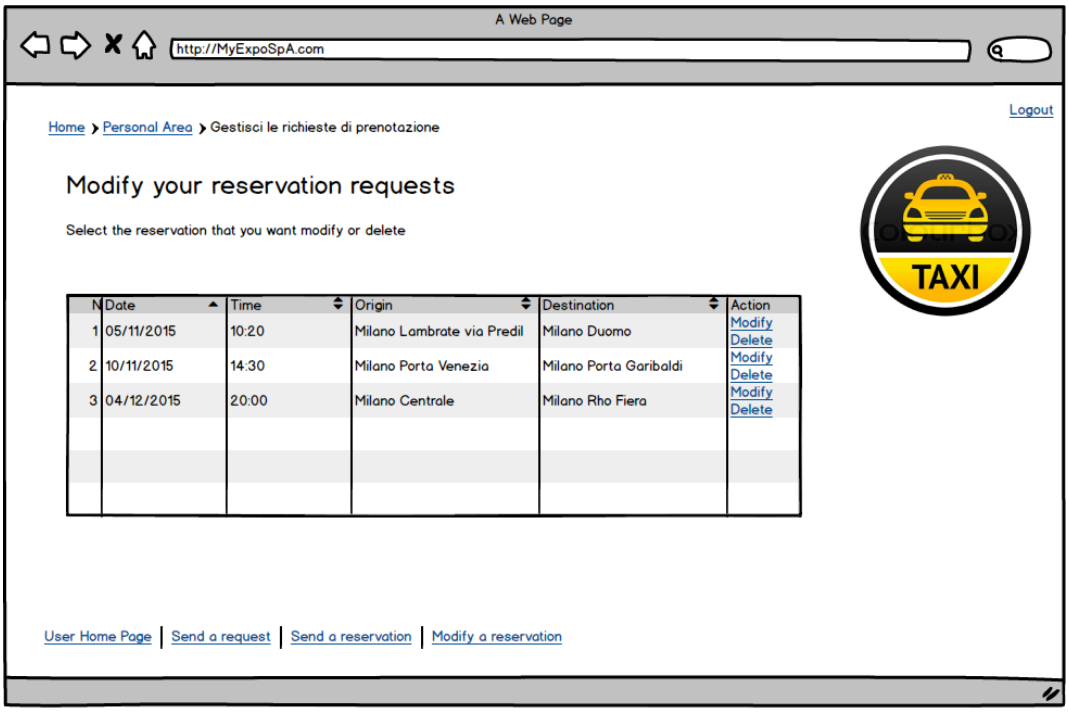


Figure 3.3: Registration Page

Figure 3.4: Taxi Request



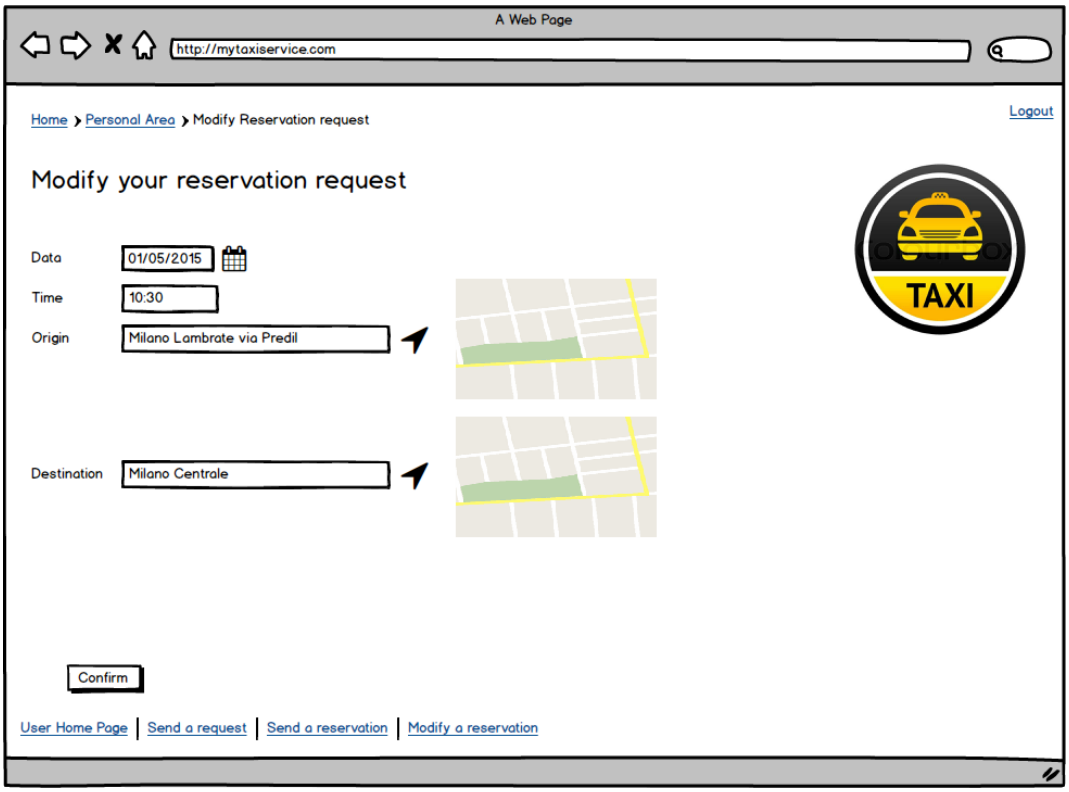


Figure 3.5: Modify Taxi Request

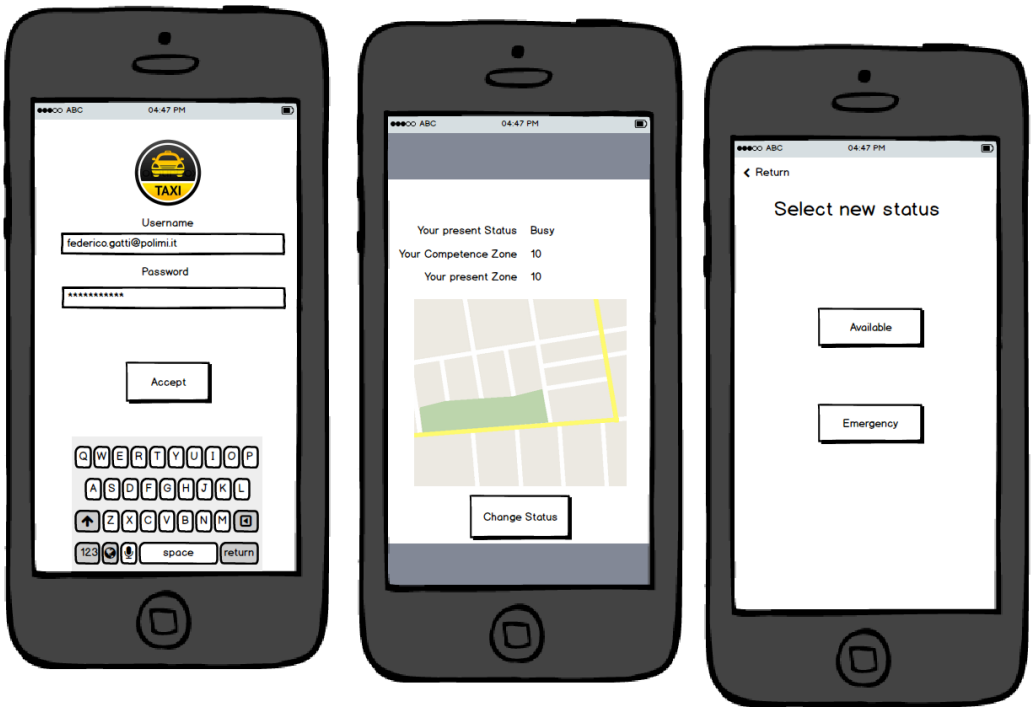


Figure 3.6: MTA

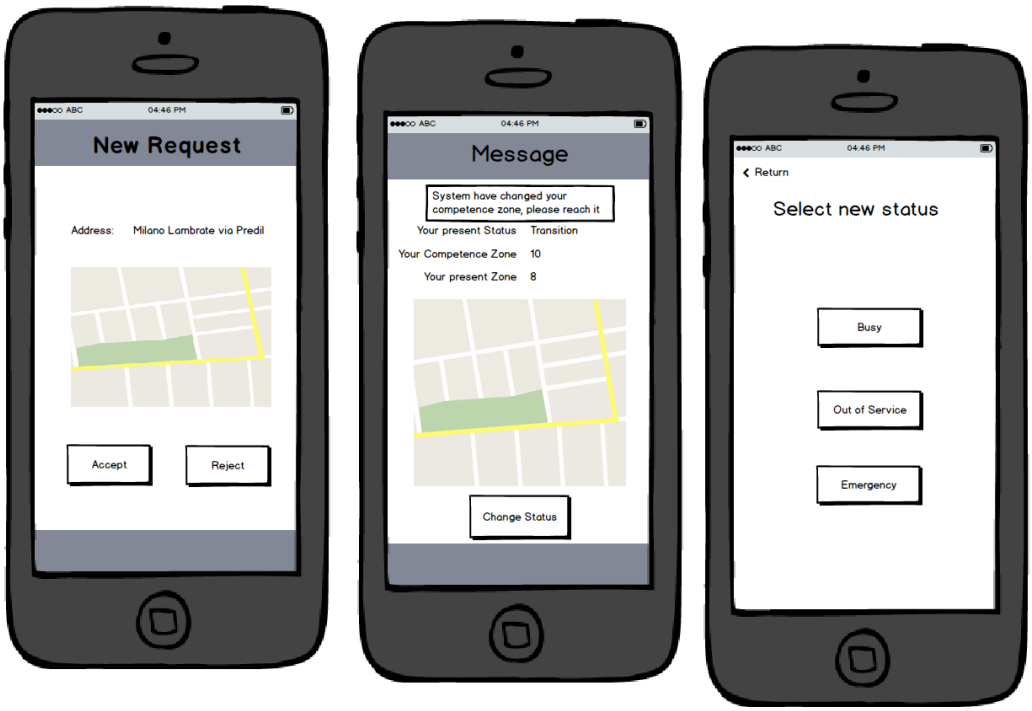


Figure 3.7: MTA1

3.1.2 Software Interfaces

- System must be able to communicate with a GPS system
- System must be able to interconnect with Google service like 'Google Map' or 'Google localization'

3.1.3 Communication Interfaces

The system communicates by using the Internet Transport Control Protocol (TCP-IP)

3.2 Functional Requirements

1. The system shall allow a Passenger to sign up by PA
2. The system shall allow a Passenger to sign in by PA
3. The system shall allow a Taxi driver to sign in by TMA
4. The system shall notify taxi its area of competence by TMA to obtain the best distribution in the city
5. The system shall permit a Taxi to *Accept* or *Reject* a request by TMA
6. The system shall handle the passenger's request, in particular:
   - The request is forwarded to the first taxi in the queue associated to the zone from which the request comes
   - If there aren't taxi in the corresponding zone the system must forward the request to the adjacent zones
   - If there aren't taxi in any zone the request is stored in a requests' queue and forwarded when a taxi becomes *Available*
   - If the Taxi changes its status in *Emergency* the current request is deleted and another one is forwarded
7. The system shall handle the reservation requests from an authenticated passenger, in particular:
   - A passenger can place a reservation at least 2 hours before the ride specifying the origin and the destination of the ride
   - A passenger can delete his reservation requests
   - A passenger can modify his reservation requests
8. The system shall forward a reservation request ten minutes before at the specified time
9. The system provides 5 status for the taxi:
   - *Available*: when the taxi doesn't carry passengers and it is situated in its competence area
   - *Busy*: from request's acceptation by the taxi until the end of the ride

  - *Transition*: when the taxi doesn't carry passengers and it isn't situated in its area of competence. In this case the system algorithm shall consider the taxi in the correct zone when computes the correct taxi distribution, but taxi isn't inserted in the queue.
  - *Out of Service*: when the taxi is not in service
  - *Emergency*: when a taxi occurs in an accident until the taxi solves it and decides to restart its turn

10. System shall assign to all the taxis a single status at a time
11. The system shall handle the taxi status, in particular:
  - When a taxi accepts a request its status changes from *Available* to *Busy*
  - When the system notifies to a taxi that its competence area is changed the system must convert taxi status from *Available* to *Transition*
  - The system changes taxi's status from *Transition* to *Available* when the taxi reaches its new competence area
  - The system provides a command to change taxi's status from *Available/Transition* to *Busy* when the taxi picks up a passenger
  - The system provides a command to change taxi's status from *Available/Transition* to *Out of Service* when a taxi ends its turn
  - The system provides a command to change taxi's status from *Busy* to Available when the passengers leave the taxi
  - The system provides a command to change taxi's status from *Out of Service* to *Available* when the taxi begins its turn
  - The system provides a command to change taxi's status in *Emergency* when the taxi occurs in a accident
  - The system provides a command to change taxi's status from *Emergency* in *Available* when the accident is solved
12. The system shall retrieve automatically passenger's position if GPS is available otherwise users has to insert manually the position.
13. The system shall handle the taxi queue, in particular:
  - When a taxi accepts a request it is deleted from the queue
  - When a taxi rejects a request it is inserted at the end of the queue
  - When a taxi changes its status from *Available* to *Busy*, *Transition* or *Out of Service* the taxi is deleted from the queue
  - When a taxi changes its status from *Transition* to *Available* it is inserted at the end of the queue
  - When a taxi changes its status into *Emergency* it is deleted from the queue
14. A taxi can be available in only one zone at a time
15. The system shall notify the passenger about the *Waiting Time* and the *Taxi ID*
16. The system shall consider *Rejected* a request if the Taxi doesn't answer in 1 minute
17. A Taxi can't reject for 2 times the same request

Note that with this type of requirements the traditional way to make a request, making a call, continues to operate. In fact when a taxi receives a request from the taxi call center it can change its status from *Available* or *Transition* into *Busy*. In this way the taxi is delayed from the queue.

3.3 Performance Requirements

The system must provide a real time interaction between the PA, TMA and QTM, so an interaction at the order of ms is required. For example 100 ms - 500 ms for QTM and 100 ms for TMA, PMA and PWA.

3.4 Software System Attributes

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on the human interaction with the software as a whole.

3.4.1 Availability

The application will be accessible online anytime, 99%. It could be necessary to use a dedicated server to achieve this goal, but all the system could be hosted into a cloud platform to guarantee more availability. This solution gives more scalability to the performance required by the system and could reduce the cost for dedicated server, maintaining an high level of performance especially in case of full load with a lot of connected users. This solution is preferable because the number of users that use the application during a day is very inconstant.
A backup will be programmed for every day.

3.4.2 Maintainability

The application does not provide any specific API, but the whole application code will be documented to well inform future developers of how application works and how it has been developed.

3.4.3 Security

In the system there are expected sensitive and personal data so is required an encrypted communication between the part of the system.
Sensitive data are accessible only to authorized users.

3.5 UseCase Diagram

In the next diagram Taxi must be considered logged into the system.

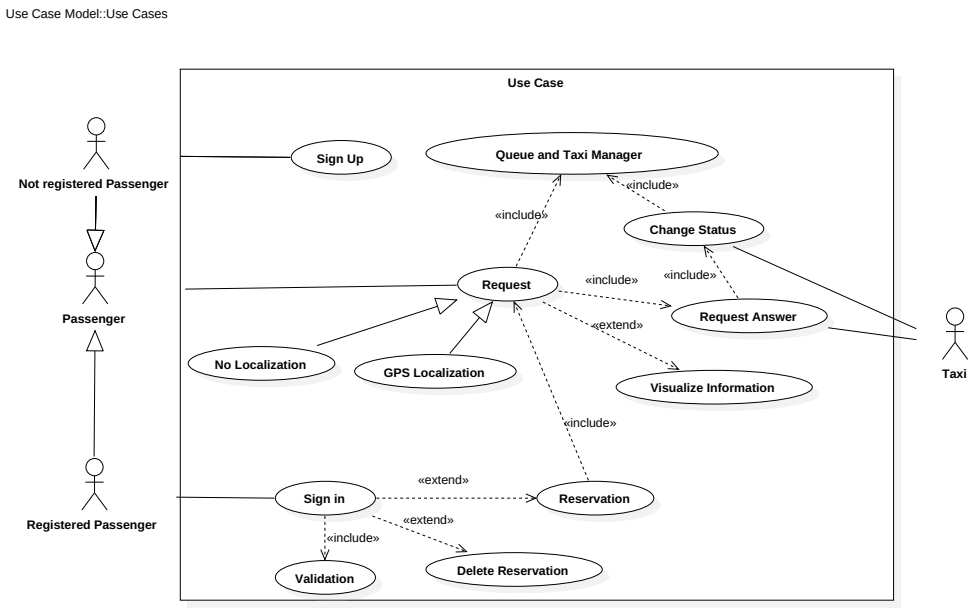


Figure 3.8: UseCase

3.6 Use Case Table and Sequence Diagram

3.6.1 Registration

Name	Sign Up
Actor	Not registered user
Entry Conditions	No conditions
Flow of events	1. The user searches the platform on the web or in the app Store 2. The user opens or installs it 3. The user clicks the 'Sign Up' button 4. The System shows a page which contains necessaries fields for registration 5. The User fills the fields inside the form and then he submit it 6. The System checks the data and if there aren't problems the system registers the user. 7. The System shows a page with a confirm of the registration
Exit conditions	User receives a registration confirm in the specified email
Exceptions	If at least one value is submitted by the user isn't valid, the System will show an alert with wrong fields

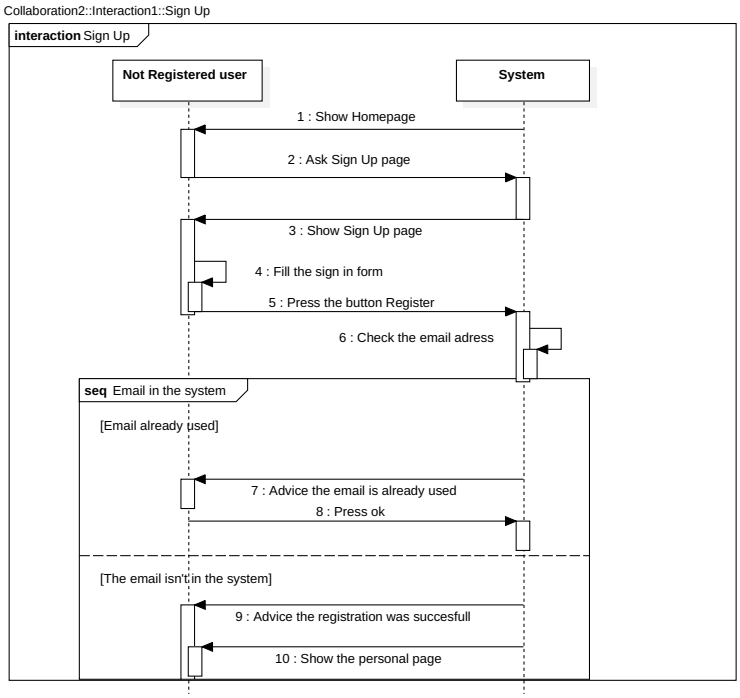


Figure 3.9: Registration Sequence Diagram

3.6.2 Sign In

Name	Sign In
Actor	Registered Passenger
Entry Conditions	The user must be registered to the system
Flow of events	1. The users opens the homepage 2. The system shows him the page 3. The user fills the fields with his username and password 4. The user clicks the *Sign In* button 5. The system checks the entered values 6. The system will show the profile page if the values written by the user are valid
Exit conditions	No Conditions
Exceptions	If the values written by user aren't valid the system will show an alert with an error message

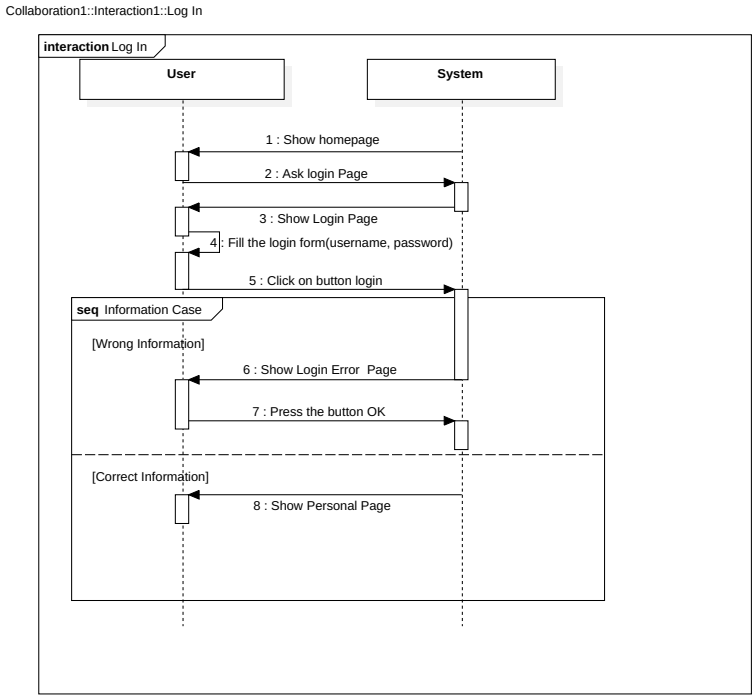


Figure 3.10: Login In Sequence Diagram

3.6.3 Request

Name	Request
Actor	Passenger
Entry Conditions	The user must be agree to the software conditions
Flow of events	1. The user opens the Application 2. The user clicks *Request* choose 3. The user uses GPS for automatic localization or inserts manually the location 4. The user clicks the 'Submit' button
Exit conditions	The system answers to the request by informing the passenger about the code of the incoming taxi and the waiting time.
Exceptions	If there aren't taxi available the system will show an alert with a message

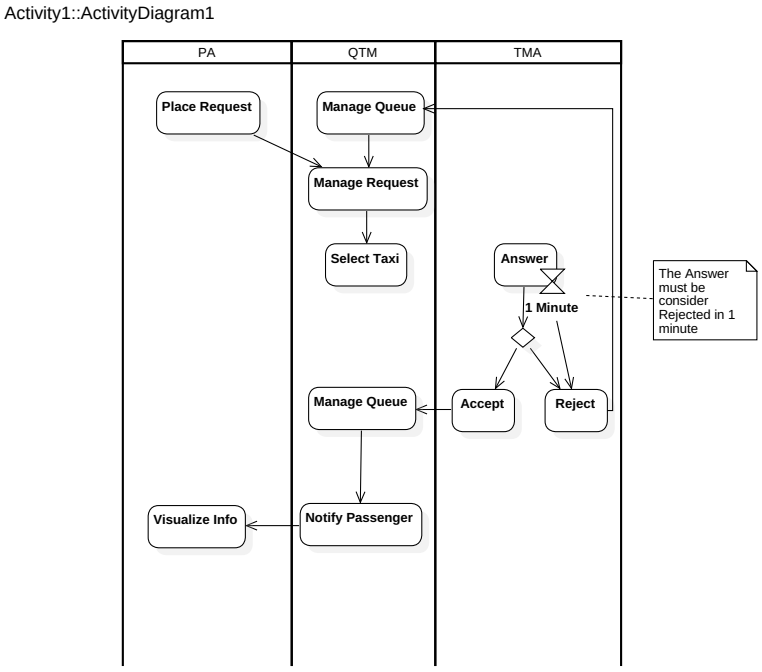


Figure 3.11: Request at high level

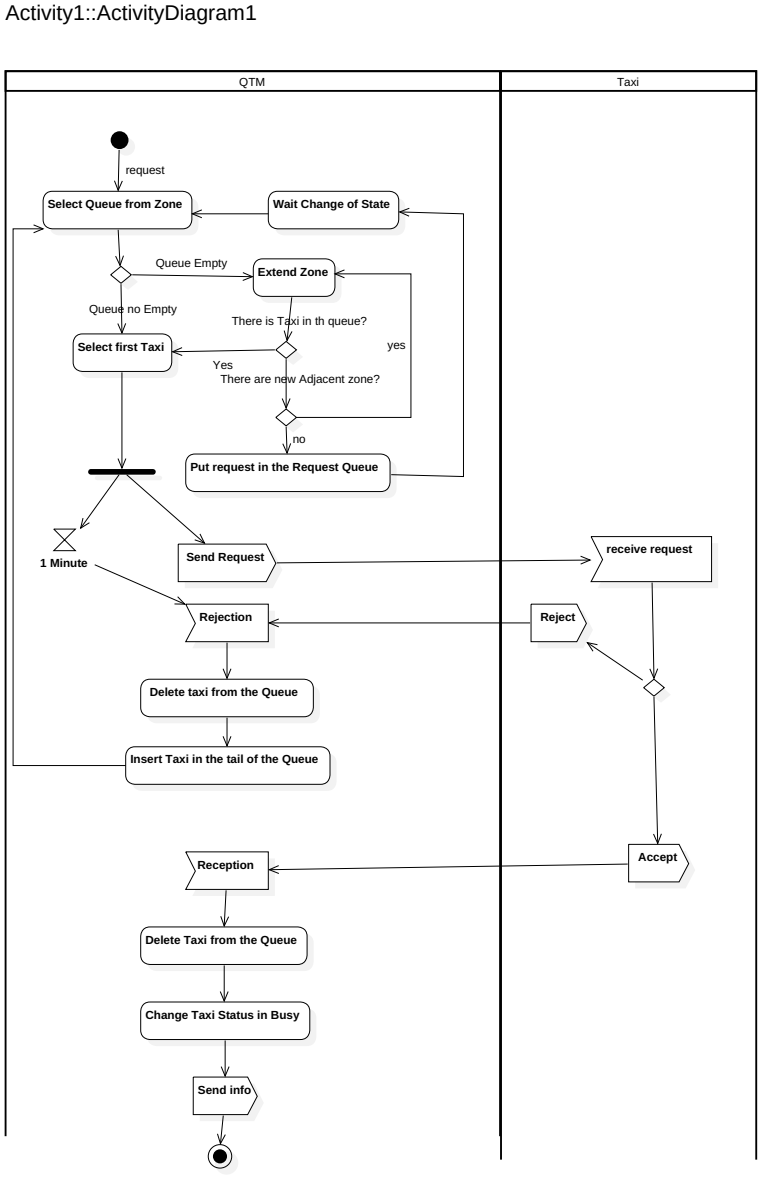


Figure 3.12: Request in details

3.6.4 Place Reservation

Name	Reservation
Actor	Authenticated Passenger
Entry Conditions	The user must be logged to the system
Flow of events	1. The user opens the Application 2. The user clicks *Reservation* choose 3. The user uses GPS for automatic localization or inserts it manually 4. The user inserts the time of the reservation 5. The user clicks the *Submit* button 6. The system confirms the reservation
Exit conditions	The system informs the user that the request was successful
Exceptions	If the reservation is placed less then two hours before the ride the system will show an alert with a message

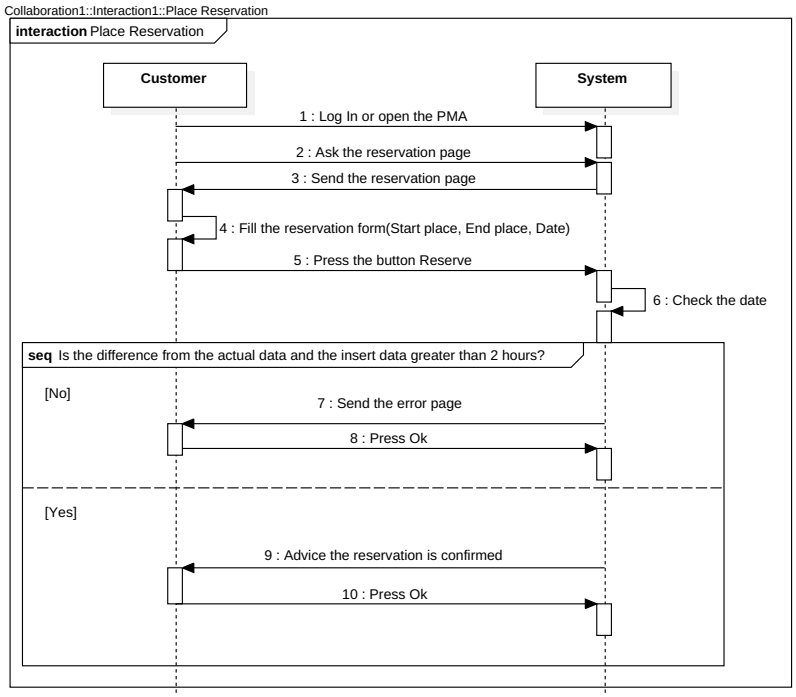


Figure 3.13: Delete Sequence Diagram

3.6.5 Delete Reservation

Name	Delete Reservation
Actor	Authenticated Passenger
Entry Conditions	The user must be logged to the system
Flow of events	1. The user opens the Application 2. The user clicks *Modify Reservation* choice 3. The user selects the reservation to be deleted 4. The user clicks *Delete* button 5. The user clicks the *Confirm* button
Exit conditions	The system informs the user that the operation was successful
Exceptions	No conditions

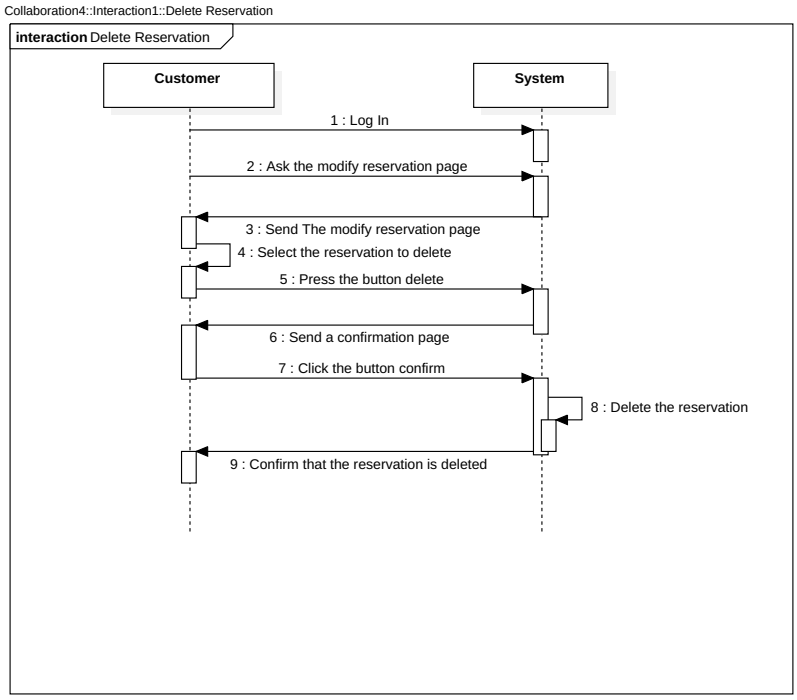


Figure 3.14: Delete Sequence Diagram

3.6.6 Modify Reservation

Name	Modify Reservation
Actor	Authenticated Passenger
Entry Conditions	The user must be logged to the system
Flow of events	1. The user opens the Application 2. The user clicks *Modify Reservation* choice 3. The user selects the reservation to be modified 4. The user clicks *Modify* button 5. The user modifies the fields 6. The user clicks the *Confirm* button
Exit conditions	The system informs the user that the modification was successful
Exceptions	

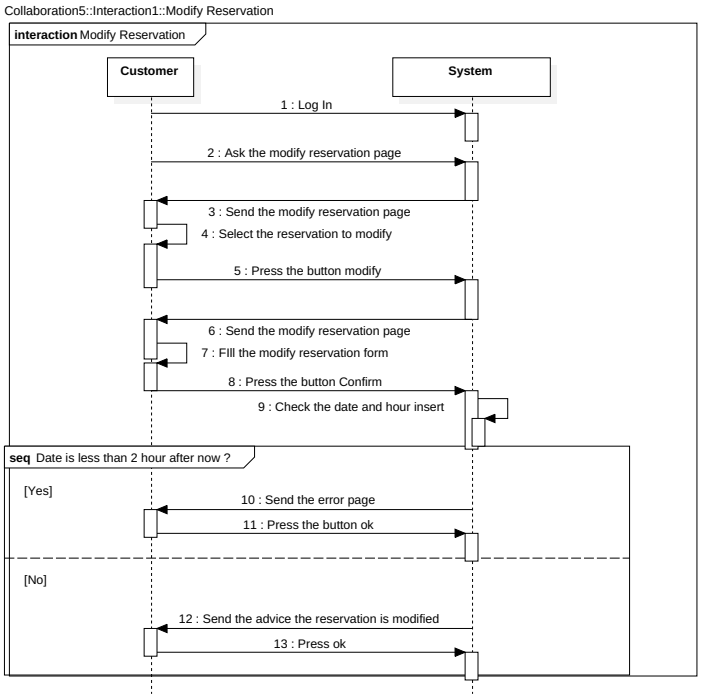


Figure 3.15: MTA

3.6.7 Visualize Information

Name	Visualize Information
Actor	Authenticated Passenger
Entry Conditions	The user must be logged to the system
Flow of events	1. The user clicks *Modify Reservation* button in the homepage 2. The user can see all the reservations' info
Exit conditions	No conditions
Exceptions No conditions	

3.6.8 Accept Request

Name	Accept Request Answer
Actor	Taxi Driver
Entry Conditions	Taxi driver must be logged to the system
Flow of events	1. Taxi Driver is notified 2. Application shows the passenger's location 3. Taxi Driver click *Accept* button
Exit conditions	Passenger is notified about waiting time and taxi ID
Exceptions	If the taxi takes more than 1 minute to answer the request is considered *Rejected*

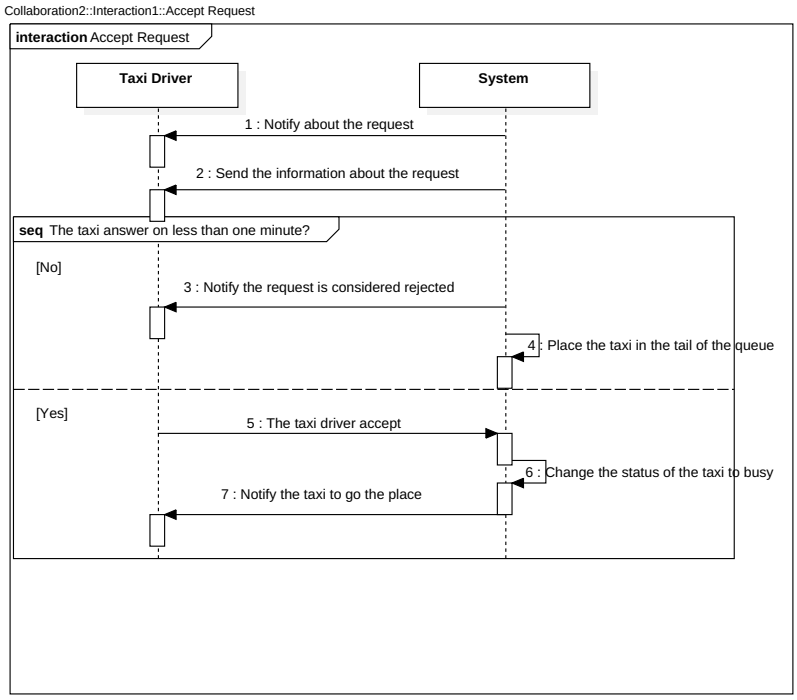


Figure 3.16: Delete Sequence Diagram

3.6.9 Reject Request

Name	Reject Request Answer
Actor	Taxi Driver
Entry Conditions	No conditions
Flow of events	1. Taxi Driver is notified 2. Application shows the passenger location 3. Taxi Driver clicks *Reject* button
Exit conditions	Taxi is deleted from the queue and then it is inserted in the tail of the queue
Exceptions	No conditions

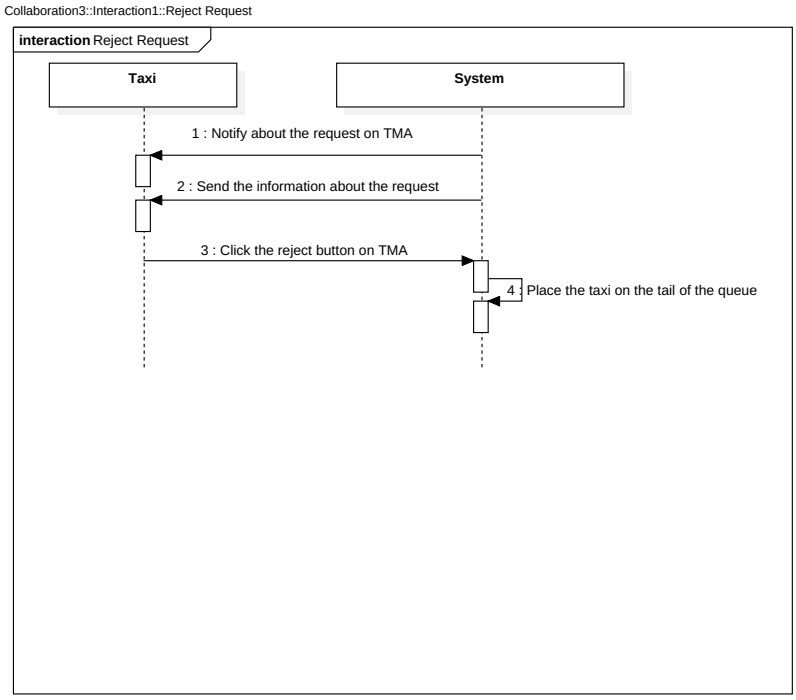
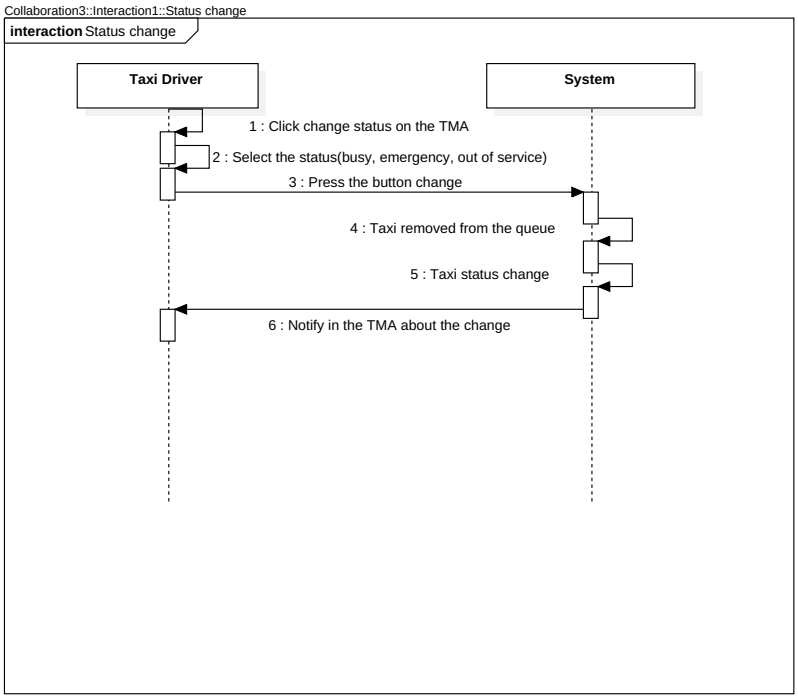


Figure 3.17: Delete Sequence Diagram

3.6.10 Change Status

Name	Change Status
Actor	Taxi Driver
Entry Conditions	No conditions
Flow of events	1. Taxi Driver clicks *Change Status* button in the app 2. Taxi Driver clicks *Busy*, *Out of Service* or *Emergency* choice
Exit conditions	Taxi is deleted from the queue and its status change. Taxi is notified of the change
Exceptions	No conditions


Collaboration3::Interaction1::Status change
interaction Status change
Taxi Driver
System
1 : Click change status on the TMA
2 : Select the status(busy, emergency, out of service)
3 : Press the button change
4 : Taxi removed from the queue
5 : Taxi status change
6 : Notify in the TMA about the change

3.6.11 Change Status from Emergency

Name	Change Status
Actor	Taxi Driver
Entry Conditions	No conditions
Flow of events	1. Taxi Driver clicks *Change Status* button in the TMA 2. Taxi Driver clicks *Available*
Exit conditions	Taxi status changes in *Available* or in *Transition* if it isn't in its competence zone. Taxi is notified of the change
Exceptions	No conditions

3.7 State Diagram

The next diagram represents the possible changes of taxi's status in the system
The status *Emergency* is omitted to make more readable the diagram, be enough remember that the *Emergency* Status is reachable from any other status and from *Emergency* is reachable only the status *Available*

StateMachine1::StatechartDiagram1

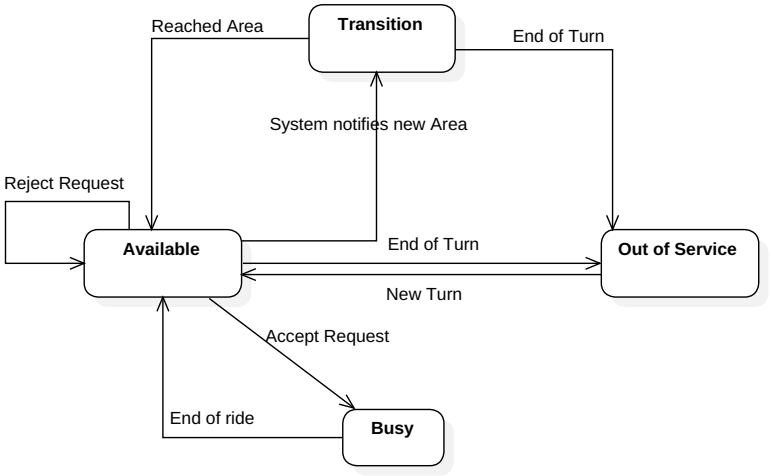


Figure 3.18: Taxi Status

3.8 Class Diagram

In this section we will show you the class diagram of the system. We decide not insert more details, but the necessary that is useful for the Alloy Model. So the entity hasn't all the filed but only the most important.

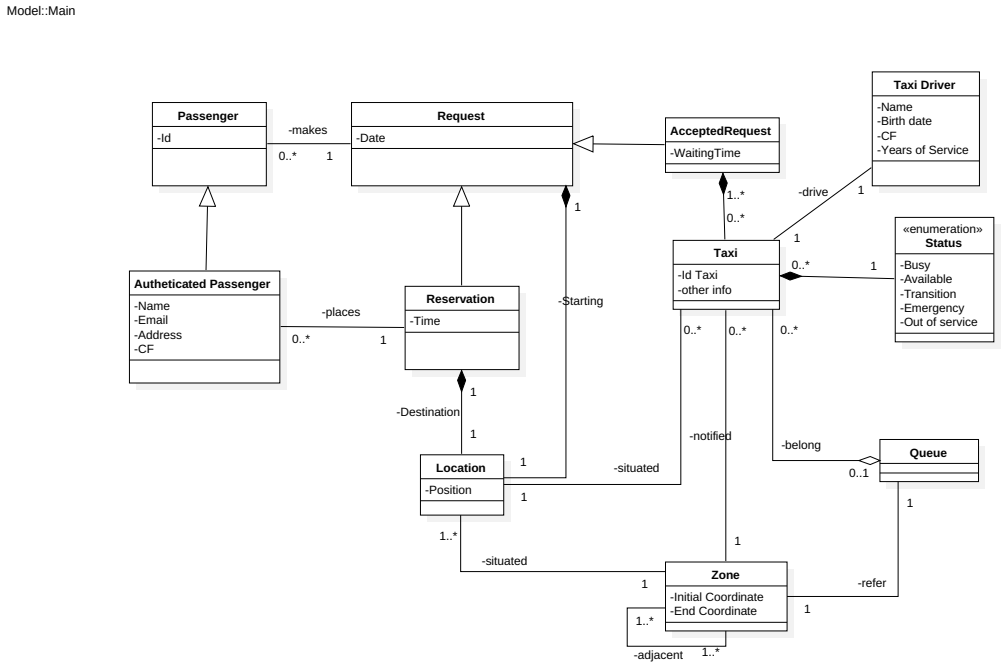


Figure 3.19: Class Diagram

Chapter 4

Scenarios Identifying

In this section we describe some scenario by the point of view of the customers and the taxi driver.

4.1 Scenario 1

James is a business manager and, because of that, he has to travel a lot of times on different cities. During one of his trips, while he is searching for a taxi on the Internet, he discovers MyTaxiService and starts reading the homepage of the site, where there are all the information about what this service is and how it works, and, interested by that, he decides to try it. He makes a reservation trough the website, where he sends his position. At the same time the system allocates the coordinates on one of the zones the city is divided in; after that, the system sends the request to the first free taxi in the zone's queue. The taxi driver receives a notification on his TMA about the request and he decides to accept that. So James receives a notification about the waiting time and the taxi's ID. In ten minutes after the reservation, the taxi arrives and takes James to his partner's office.

4.2 Scenario 2

James is very satisfied of MyTaxiService so he decides to register an account on the application, also because he wants to use the features that allow him to reserve a taxi for a chosen place-to-place path which is, in this case, the path from the partner's office to the airport. He inserts in the Registration page the personal information like name, surname, mail, birth date, permanent address and CF. After the registration, James, using his PMA, decides to reserve a taxi that brings him from his partner's office to the airport at half past two PM on that day. Ten minutes before the specified time the System contacts Marc, the first taxi driver on the queue list of the zone and he receives the request for the service, but unfortunately he isn't able to accept that so he *Rejects*

it and the request is sent to Jacob, the second taxi driver on the queue that accepts the Request. Then James receives the ID number of Jacob's taxi. At the established hour he finds the reserved taxi and he makes the trip. He is so satisfied of the service that he decides to let a positive review on the site.

4.3 Scenario 3

Jeremy lives in an isolated area, and he typically uses his car to go to work. Unfortunately, today he has a problem with the car, and nobody can give him a ride, so he decides to use MyTaxiService to move. He asks on the PWA for a taxi, but unfortunately the zone has not any taxi in the area, so the PA sends the request to the adjacent areas and it's sent to the first free taxi. Luke, the taxi driver accepts the request, so the system sends to the user the Taxi id and the Waiting time which are sent to to the user. Unfortunately, during the travel an animal crosses the road and trying to avoid it, Luke has an accident. Luke isn't hurt but the taxi is broken, so he changes his status to *Emergency* because now he can't go to Jeremy's place. So the system searches for another taxi to satisfy the request. This time, there is a taxi in the Jeremy's area and he receives the token request. The taxi driver accepts and the users receives an update with the ID of the new taxi. After ten minutes the taxi arrives and so the customer is taken from his house to his workplace.

4.4 Scenario 4

Lucas, a taxi driver for *MyTaxiService*, has just ended the transport of a client and he switched is status to *Available*. At the same time the system discovers that Avenue Park has a number of taxis that is lower than the expectation, so it decides to move a taxi from a near area to solve this problem. The system notifies to Lucas to drive to the new competence area, so the System changes Lucas' taxi status into *Transition* and Luca starts goes to Avenue Park and when he arrives the System changes again his status into Available. Almost immediately, he receives a request from a client that is close to the park, so he immediately accepts the request and he goes to take the customers. He arrives after five minutes and transports the client to the airport.

4.5 Scenario 5

Lori works for a very important bank, and she makes the choice to not have a car for herself, so she usually takes the taxi when she has to move by car. One day, a friend speaks her about *myTaxiService* and how it works and she decides to try it without registering on the site. She downloads TMA on her mobile phone and she makes a request for a taxi. The system sends her the id of the chosen taxi and the waiting time, and she starts waiting. During the waiting time, she receives a message from her boss that ask her to hurry up, so when she doesn't see any taxi, she opens the app to ask

for another taxi. Also this time, the system receives the request and sends a new taxi to take her. Finally, the first taxi shows up and picks up Lori. In five minutes, also the second Taxi arrives, it waits for around five minutes, and not seeing anyone, decides to advice the system that the client is no more there and changes his status to *Available*.

Chapter 5

Alloy

The following alloy model is created using the class diagram. The code is divided in parts: signature, fact, assert and predicates. In the last part there is the world generated.

5.1 General Model

5.1.1 Data Type

```
module mytaxiservice/queue

// DATA TYPE
sig TaxiDriver{}

sig Taxi{
        driver: one TaxiDriver,
        status: one TaxiState,
        id: one Int,
}{id >=1}

sig Queue{
        taxis: some Taxi,
}

sig Request{
        passenger: one Passenger,
        date: one Date,
        time: one Time,
        location: one Location,
}
```

```
sig ConfirmedRequest extends Request{
        waitingTime: one Time,
        taxis: some Taxi
}

sig Reservation{
        passenger: one AuthenticatedPassenger,
        startingPlace: one Location,
        endingPlace: one Location,
        date: one Date,
        time: one Time,
        request: one Request
}{passenger = request.passenger}

sig Date{}

sig Time{}

sig Zone{
    queue one Queue,
}

sig Location{
        zone : lone Zone,
}

sig AuthenticatedPassenger extends Passenger{}
sig UnauthenticatedPassenger extends Passenger{}

one sig Available extends TaxiState{}
one sig Busy extends TaxiState{}
one sig OutofService extends TaxiState{}
one sig Transition extends TaxiState{}
one sig Emergency extends TaxiState{}

//ABSTRACT SIG
abstract sig Passenger{}

abstract sig TaxiState{}
```

5.1.2 Fact

```
//FACT

//each taxis have at maximum one TaxiDriver
fact oneTaxiForEachTaxiDriver{
        all disj t1,t2 : Taxi | t1.driver != t2.driver
}

// two taxis don't have the same id
```

```
fact noTaxiWithSameId{
        no t1,t2: Taxi | t1 != t2 and t1.id = t2.id
}

//Passenger can place only a request  at time
fact oneRequestAtTimePassenger{
        all disj r1, r2 : Request | sameTime[r1,r2] implies r1.passenger != r2.passenger
}

//Taxi can satisfy only a request at time
fact oneRequestAtTimeTaxi{
        all disj r1, r2 : ConfirmedRequest | sameTime[r1,r2] implies r1.taxis!= r2.taxis
}

//A Reservation is associated at most one Request
fact oneRequestIsAssociatedToOneReservation{
        all r : Request | lone rs: Reservation | rs.request = r
}

//A Taxi must belong at most one queue and its status must be Available
fact OneQueuePerTaxi{
        all t : Taxi | one q: Queue | t in q.taxis and t.status = Available
}

//A Queue must belong to exactly one zone
fact OneQueuePerZone{
        all q: Queue | one z: Zone | q in z.queue
}
//A Busy taxi accepted at least a request
fact allBusyTaxiHaveAtLeastAConfirmedRequest{
        all t:Taxi  | some cr:Request | t.status in Busy implies t in cr.taxis
}

//All Taxi in a Queue must be in the same Zone that queue refers
fact TaxiAvailableZoneIsEqualToTheQueueZone{
        all t:Taxi | one z:Zone | one q:Queue | t.status in Available implies t in z.queue.taxis
}


}
```

5.1.3 Predicate

```
//PRED
pred sameTime[r1, r2: Request]{
        r1.date = r2.date and r1.time=r2.time
}
pred show{}

run show for 3
```

Executing "Run show for 3" Solver=sat4j Bitwidth=4 MaxSeq=3 SkolemDepth=1 Symmetry=20 3055 vars. 252 primary vars. 6223 clauses. 14ms. . found. Predicate is consistent. 15ms.

5.1.4 Assert

```
assert oneTaxiMustBeInService{
        one Taxi implies Taxi.status in (Available + Busy + Transition)
}

check oneTaxiMustBeInService

assert noConfirmedRequestAtTheSameTime{
                  all td: TaxiDriver | all disj cr1,cr2: ConfirmedRequest
| td in (cr1.taxis.driver & cr2.taxis.driver) implies not sameTime[cr1,cr2]
}
```

Executing "Check oneTaxiMustBeInService" Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 3081 vars. 252 primary vars. 6249 clauses. 15ms. No counterexample found. Assertion may be valid. 8ms.

5.1.5 World

In this section is described the world created by Alloy. We choose the two most meaningful.

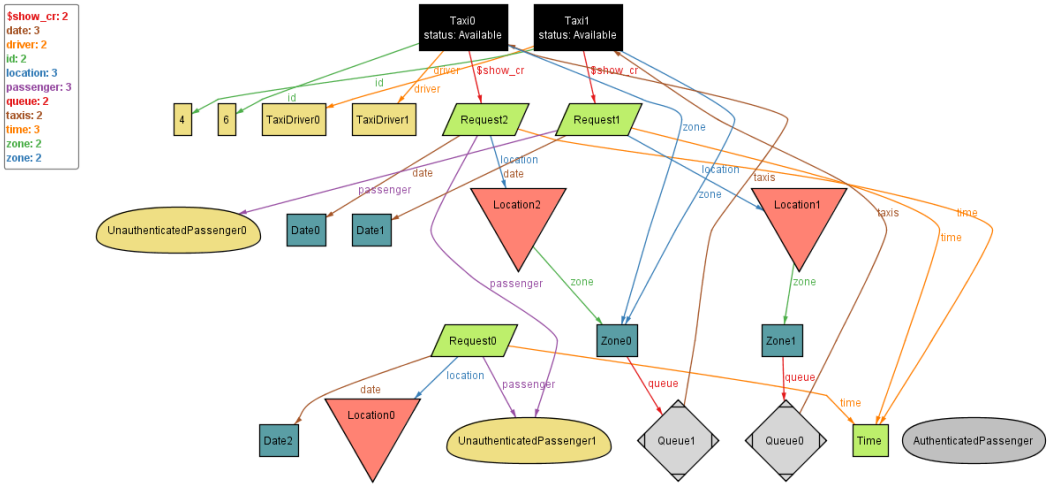


Figure 5.1: Alloy World

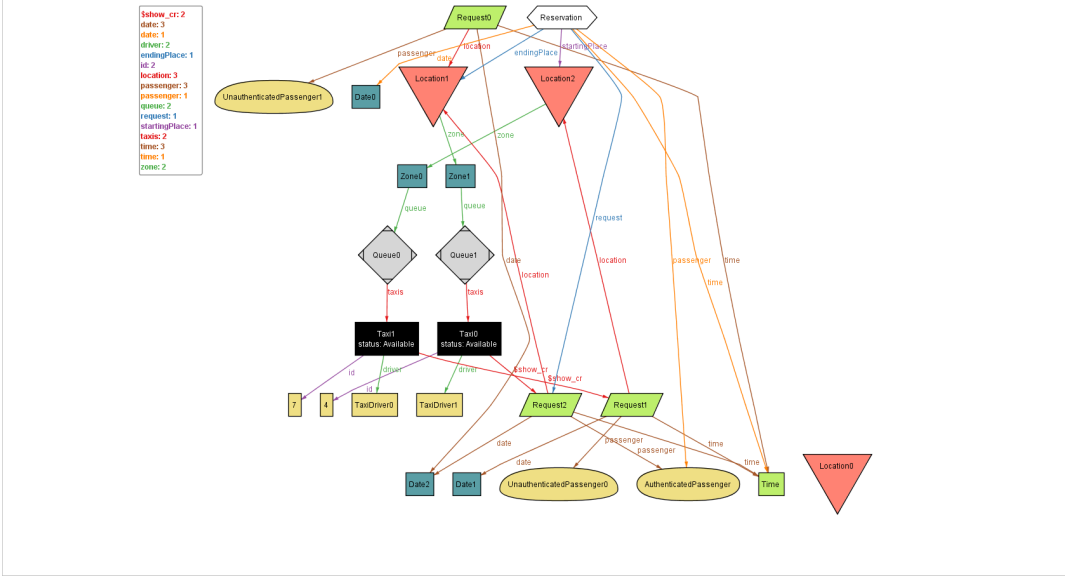


Figure 5.2: Alloy World 2

Chapter 6

Used Tools

The tools used to create the RASD documents are:

- ShareLatex: To redact and to format the document, very helpful because it works completely online.
- StarUml: To create all the sequence diagrams, the class diagram and the state diagram.
- Balsamiq: To realize the Mockup both for the web and mobile application.
- Alloy Analizer 4.2: To realize the model and to check its consistency.