POLYTECHNIC UNIVERSITY OF MILAN

School of Industrial and Information Engineering

Computer Science and Engineering

Project of Software Engineering 2: MyTaxi Service Requirements Analysis and Specification Document

Course Professor: Prof. Elisabetta DI NITTO

Authors:

Mattia CRIPPA 854126

Francesca GALLUZZI 788328

Marco LATTARULO 841399

Academic Year 2015–2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

This document represents the *Requirement Analysis and Specification Document* (RASD). The main goal of this document is to completely describe the system in terms of functional and non functional requirements, analyze the real need of the customer to modeling the system, show the constraints and the limits of the software and simulate the typical use cases that will occur after the development. This document is intended to all developers and programmers who have to implement the requirements, to system analysts who want to integrate other systems with this one, and could be used as a contractual basis between the customer and the developer.

1.2 Scope

The aim of the project is to optimize the taxi service of a city using modern techniques and technologies to better manage the distribution and the availability of the taxicabs, to shorten the waiting time for passengers and to reduce costs for passengers and traffic in the entire city using taxi sharing. The system is implemented as a mobile application associated to an external, high precision GPS receiver for the taxi drivers and as a mobile application or a web interface for the passengers.

1.3 Domain Properties

We suppose that the following properties hold in the analyzed domain:

D01 : Taxicabs are all equals. They have a maximum of 3 passenger seats and they are all owned by a specific company.

D02 : Every taxi driver uses always the same taxicab

D03 : Accurate taxicabs positions are known by the GPS

D04 : Taxi drivers correctly signal their availability

D05 : If a passenger requests or reserves a taxi, he will then take the ride from the specified origin to the specified destination

D06 : A passenger doesn't change the origin or the destination of a ride after the reservation

D07 : If a taxi driver confirms a ride, then he/she will reach the passenger location on time according to the reservation hour or the waiting time calculated by the system

D08 : If a taxi driver confirms a ride, he/she will complete it

D09 : The total fee of a ride is calculated considering only the total kilometers of the ride, given a specific fee per km

1.4 Proposed System

The system will be composed by a server running all the business logic, generating dynamic web pages and managing all the accesses to the data sources, and by a number of clients implemented as mobile application deployed on Android or a web application using the JEE platform. GPS raw data will be provided by a specific GPS receiver, installed in every taxicab and provided by a specialized company.

1.5 Goals

Visitors should be able to:

G01 : Sign up

G02 : Log in

Passengers should be able to:

G03 : Request a taxi

G04 : Reserve a taxi

G05 : Receive information about the incoming taxi (confirmation, taxi ID, waiting time)

G06 : Share the ride

G07 : Receive a receipt after each completed ride

G08 : Update personal account

Taxi drivers should be able to:

G09 : Signal whether they are available or not

G10 : Receive incoming requests

G11 : Accept or decline incoming requests

G12 : Visualize information about the optimal route

G13 : Update personal account

Developers should be able to:

G14 : Add new features

1.6 Assumptions

A01 : Given the collected data about the distribution and the total number of the taxicabs, we assume that the coverage of the taxicabs among the zones is almost uniform; therefore the probability of having an empty queue is reasonably low.

A02 : Given the collected data about the acceptance rate of the taxi drivers, we assume that the probability of reaching the end of the queue without any acceptance is reasonably low.

A03 : We assume that the maximum number of people sharing the same ride is 3.

A04 : We assume that the taxi driver will correctly follow the optimal route suggested by the system.

1.7 Stakeholders Identification

- The government of the city in which the system will be used
- The citizens of the city in which the system will be used
- The taxi drivers of the city in which the system will be used
- A specific IT company which provides all the technology (GPS receivers, smartphones for taxi drivers, mainframes and computational power) in exchange of raw GPS data provided by the receiver that will be used for data mining purposes
- A specific company that loans the taxicabs

1.8 Definitions, acronyms and abbreviations

PASSENGER: is a citizen that benefits of the taxi service

TAXI DRIVER: is the owner of a taxicab, and provides a taxi service to citizens

RIDE: is the act of make use of the taxi service (provided by a taxi driver) by a passenger

REQUEST: is the act performed by a passenger when he/she immediately needs a taxicab in the location in which he/she is

RESERVATION: is the act performed by a passenger when he/she needs a taxicab in a certain future time (at maximum in 2 hours from now) at a certain specific location

ROUTE: is the path through the city from the origin of the specific ride to the destination of the specific ride

1.9 References

- Specification Document: MyTaxiService Project AA 2015-2016.pdf.
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- IEEE Std 1016tm-2009 Standard for Information Technology - System Design - Software Design Descriptions.

1.10 Overview

This document is essentially structured in seven parts:

- **Section 1** → Introduction: it gives a description of the document and gives general information about the software product with more focus about constraints and assumptions.
- **Section 2** → Actors Identification: it gives a description of all actors of the system.
- **Section 3** → Specific Requirements: this part lists all the functional and non functional requirements that are part of the system.
- **Section 4** → Scenario Identifying: it gives a description of typical scenarios.

- **Section 5** → UML Models: to give an easy way to understand all functionality of this software, this section is filled with UML diagrams and it contains also some Use Cases.
- **Section 6** → Alloy Modeling: this part contains the code used for the analysis of consistency of the Class Diagram and the worlds generated by Alloy Analyzer in order to understand if the model is consistent.
- **Section 7** → Other Information: this part contains some information about the software used to realize this document and the total working hours of each group member.

Chapter 2

Actors Identifying

The actors of our system are:

- ***Visitor***: unregistered user that access the application interface in order to sign up or log in as Passenger, Taxi Driver or Developer and start interacting with the system.
- ***Passenger***: this user, after successful log in, depending on his/her needs, access the request interface or the reservation interface. He/she has also access to an information page about all his/her accepted reservations or requests and has access to a page with the chronology of his/her receipts.
- ***Taxi driver***: this user, after successful log in, is enabled to receive reservations or requests from passengers and can decide whether to accept them or not. He/she can access at any time after the acceptance a summary page about the specific ride. This user is also able to signal at any moment his/her availability to accept new reservations or requests.
- ***Developer***: this user, after successful log in, can access a specific interface through which he/she can introduce modifications to the system or access privileged information about the system for maintenance scopes.

Chapter 3

Specific Requirements

3.1 Functional Requirements

Functional requirements should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs.

G01 → To allow a visitor to sign up, the system shall provide functionalities to:

- **R01** Check the validity and correctness of the information provided by the Visitor (personal information, password, payment information)
- **R02** Check if the user is already registered into the system

G02 → To allow a Visitor to log in, the system shall provide functionalities to:

- **R03** Check if username and password provided by the Visitor correspond to an existing user, authorized to use the system
- **R04** Prevent unauthorized or banned users from accessing the system

G03 & **G04** → To allow a Passenger to request and to reserve a taxi, the system shall provide functionalities to:

- **R05** Obtain the Passenger location
- **R06** Access the queue associated to the right taxi zone

R07 Check the availability of Taxi Drivers

R08 Iteratively contact all the Taxi Drivers of the queue starting from the first one until one of them accepts the call

R09 Iteratively search for an available Taxi Driver inside adjacent zones in the case that the right zone has an empty queue or all the contacted Taxi Drivers had declined the request

G05 → To allow a Passenger to receive information about the incoming taxicab, the system shall provide functionalities to:

R10 Obtain the Taxi Driver position and the Passenger position and estimate the time needed by the Taxi Driver to reach the Passenger

R11 Obtain the taxicab unique identifier from the Taxi Drivers database

G06 → To allow a Passenger to share a ride, the system shall provide functionalities to:

R12 Check for each request or reservation if the Passenger has enabled the sharing function

R13 Compare routes that start from the same taxi zone and determine whether or not they can be merged into one, according to specific rules of comparison

R14 Calculate the correct distribution of the fee according to specific rules based on the percentage of the kilometers shared with others or traveled alone

R15 Elaborate an optimal route for taking every passenger to the right destination and show it to the taxi driver

G07 → To allow a Passenger to receive receipts after each completed ride, the system shall provide functionalities to:

R16 Keep track of the actual route followed by the Taxi Driver and keep track of the actual duration of the ride

G08 & **G13** → To allow passengers and taxi drivers to update the personal account, the system shall provide functionalities to:

R17 Update the database information for each user

G09 & **G11** → To allow a Taxi Driver to signal whether he/she is available or not, and to accept/decline an incoming request, the system shall provide functionalities to:

R18 Monitor and collect inputs from Taxi Drivers

G10 → To allow a Taxi Driver to receive incoming requests, the system shall provide functionalities to:

R19 Contact Taxi Drivers and forward them all the information about the proposed request (position of the Passenger, destination of the Passenger, sharing option enabled or not)

G12 → To allow a Taxi Driver to visualize information about the optimal route for the ongoing ride, the system shall provide functionalities to:

R20 Retrieve the taxi location and the locations of all the Passengers of the ride

R21 Access and query the map provider service to obtain an updated map with information about traffic, smashes, road construction sites

G14 → To allow a Developer to add new features, the system shall provide functionalities to:

R22 Update the system code and architecture

3.2 Non Functional Requirements

3.2.1 External Interfaces

Hardware Interface:
Device should be enabled with Internet and GPS receiver.
Software interface:
The user's browser should be HTML5 compatible and the resolution should be at least 1280x720 for a satisfactory user experience.

3.2.2 User Interfaces

Here are presented some mockups that represent an idea of the structure of the application pages:

Log In

The mockup in Figure 3.1 shows the Log In Page of MyTaxiService. Here Visitors can log in into the application.

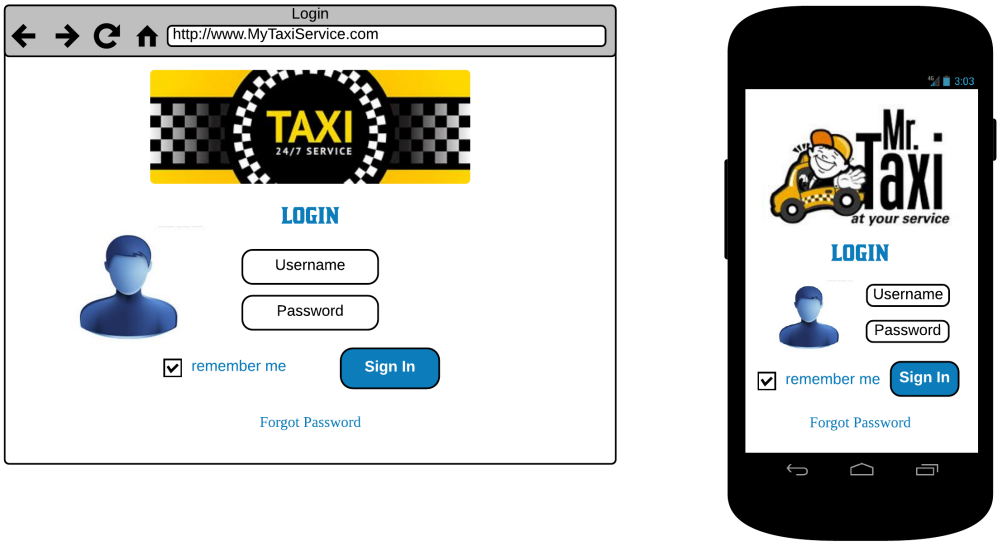

Figure 3.1: Log In Page mockup

Registration Form

The mockup in Figure 3.2 shows the Registration Form Page that Visitors can access.

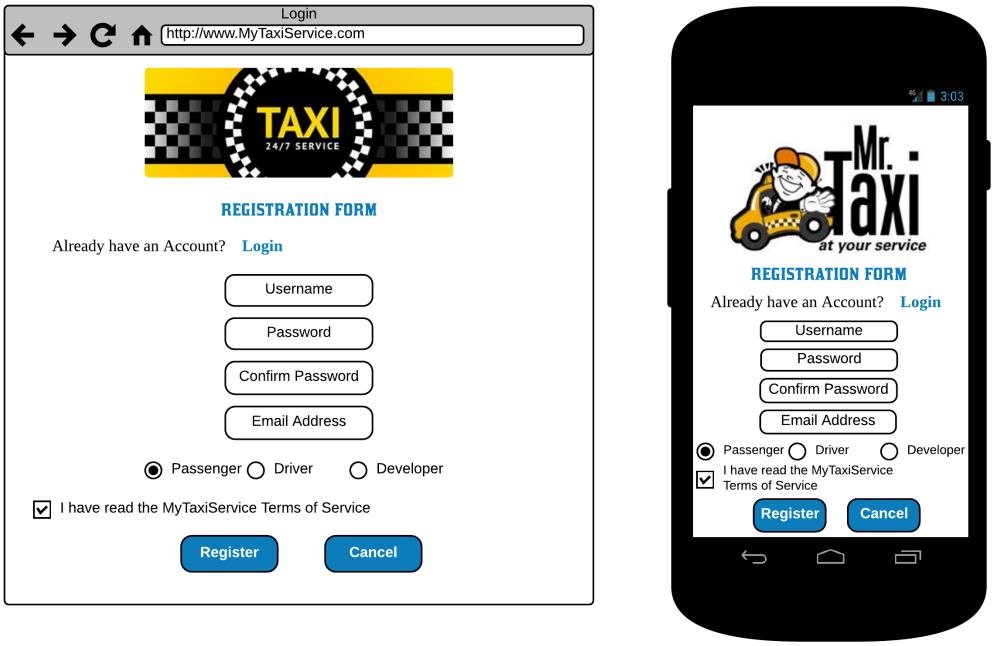

Figure 3.2: Registration Form mockup

Passenger Home Page

The mockup in Figure 3.3 shows the Home Page for a Passenger user.

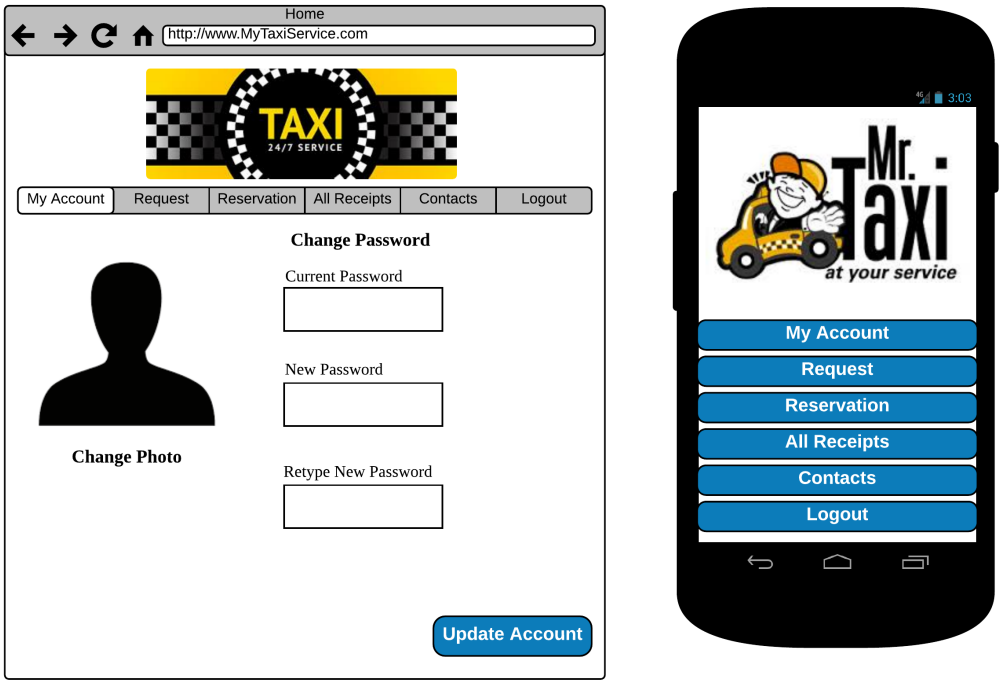

Figure 3.3: Passenger Home Page mockup

Passenger Request Page and Reservation Page

The mockups in Figure 3.4 and Figure 3.5 show the Request and the Reservation Pages for a Passenger user.

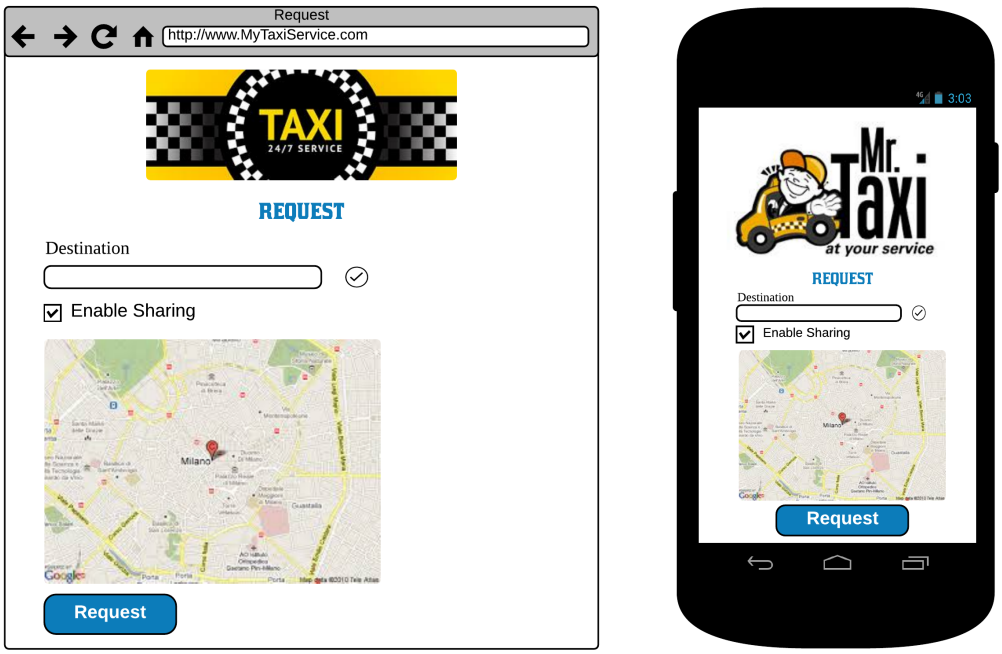

Figure 3.4: Passenger Request Page mockup

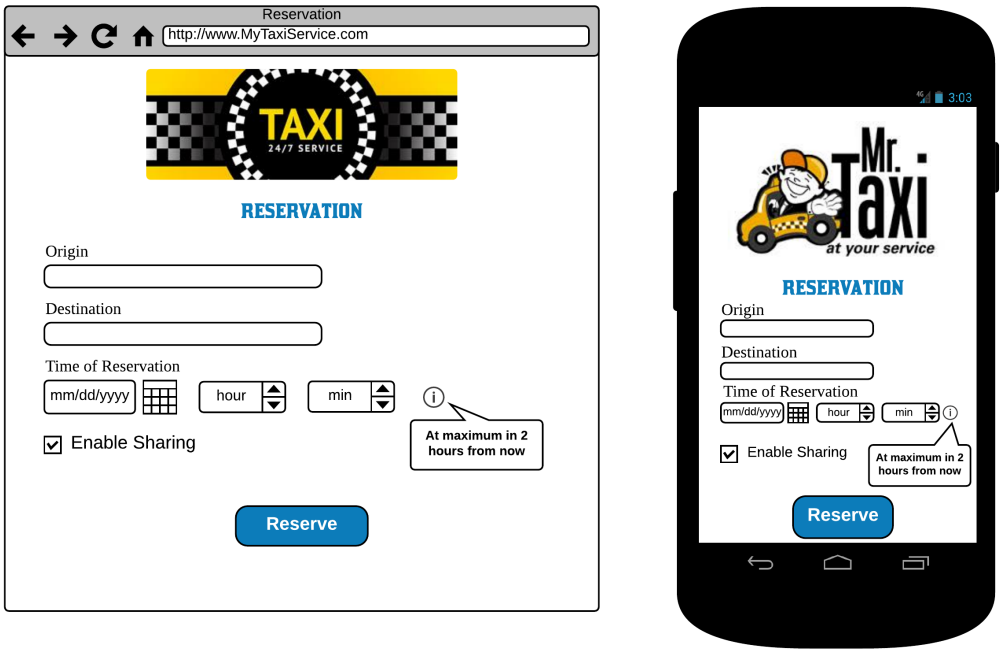

Figure 3.5: Passenger Reservation Page mockup

Taxi Driver Home Page

The mockup in Figure 3.6 shows the Home Page for a Taxi Driver user.

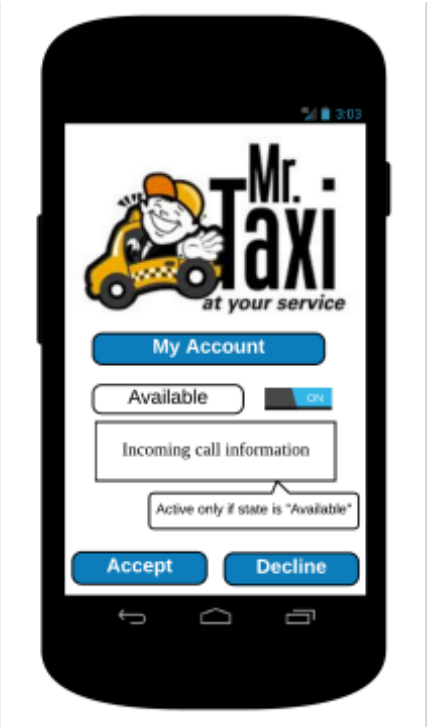

Figure 3.6: Taxi Driver Home Page mockup

Taxi Driver Accepted Call Page

The mockup in Figure 3.7 shows the Page for a Taxi Driver user that accepts a call.

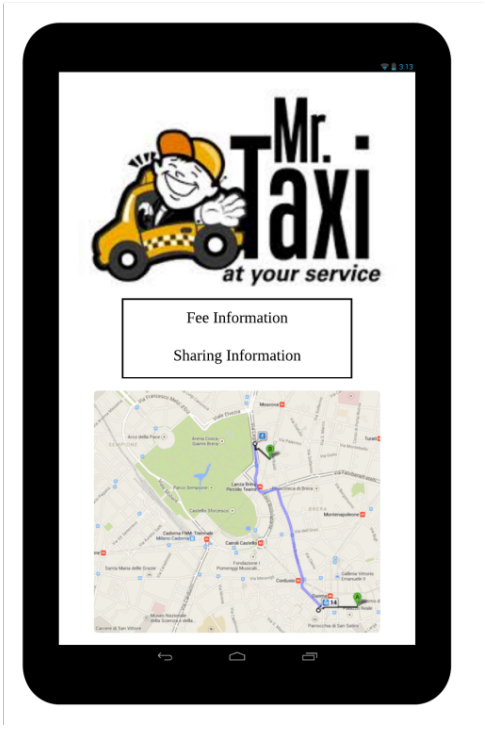

Figure 3.7: Taxi Driver Accepted Call Page mockup

Chapter 4

Scenario Identifying

4.1 Scenario 1

Visitor V wants to sign up into the system as a Passenger
V opens the mobile or the web app and he/she is immediately asked to log in or sign up. V clicks on the button "Sign Up" and is brought to the registration interface. He/She inserts the desired Username, Password and Email address and selects "Passenger" as type of user. Then V clicks on the button "Register" and, if the credentials are correct, he/she receives a confirmation message. The Passenger Home Page now appears on the app.

4.2 Scenario 2

Passenger A requires a standard ride R and driver D accepts
A clicks on the "Request" tab from the Passenger Home Page and the request interface appears on the app. A inserts the destination for his/her desired ride and doesn't check the enable sharing option, then clicks on the button "Request". D receives the incoming request sent by A and decides to accept the call. D receives a message with a summary of R, including indication for an optimal route, and also A receives a confirmation message with information about the taxicab ID of D and the estimated waiting time.

4.3 Scenario 3

Passenger A reserves a standard ride R. Taxi Driver D1 decline the call but second Taxi Driver D2 accepts

A clicks on the "Reserve" tab from the Passenger Home Page and the reservation interface appears on the app. A inserts the origin, the destination, the desired time for R and doesn't check the enable sharing option, then clicks on the button "Reserve". D1 receives the incoming reservation sent by A and decides to decline the call, so the reservation is sent to D2, that decides to accept the call. D2 receives a message with a summary of R, including indication for an optimal route, and also A receives a confirmation message with information about the taxicab ID of D2.

4.4 Scenario 4

Passenger A1 requires shared ride and another Passenger A2 is added to the same ride. The Taxi Driver D accepts

A1 clicks on the "Request" tab from the Passenger Home Page and the request interface appears on the app. A1 inserts the destination for his/her desired ride and checks the enable sharing option, then clicks on the button "Request". A2 does the same thing and, because they are in the same zone and they are going in the same direction, they are put together in the same ride. D receives the incoming request sent by A1 and A2 and decides to accept the call. D receives a message with a summary of R, including indication for an optimal route and the number of passengers, and also A1 and A2 receive a confirmation message with information about the taxicab ID of D the estimated waiting time and the total number of passengers.

4.5 Scenario 5

After a ride, Passenger A wants to see the receipt

A has just completed the ride that he/she requested. A opens the app and, after logging in, clicks on the "All Receipts" tab. A list of all his/her receipts appears in the app, ordered from the most recent to the oldest. A clicks on the

first receipt, which is the one that corresponds to the just completed ride, and sees all the details of the receipt.

4.6 Scenario 6

A Developer D access the app to introduce modifications to the system

D opens the mobile or the web app and he/she is immediately asked to log in or sign up. D clicks on the button "Log In" and inserts his/her credentials. If the credentials are correct, D has access to a specific interface dedicated to developers. D introduces modifications to the system and, after checking that everything is correct, submits the modifications. The app restarts and the new modifications are applied. D logs in again and verifies that everything works correctly.

Chapter 5

UML Models

5.1 Use Case Diagram

The Figure 5.1 gives an overview on the main actors involved in the system and the functionalities that they are able to execute.

Figure 5.1: Use Case Diagram

5.2 Use Case Descriptions

Sign Up

Actor	Visitor
Related Requirements	[R01], [R02]
Goal in context	[G01]
Precondition	the Visitor access the initial application interface
Successful end condition	the Visitor is registered on the system as a passenger, taxi driver or developer
Failed end condition	the Visitor is not able to register. He/She is asked to try again
Trigger	the Visitor selects the "Sign up" button
Main Flow	**1.** the Visitor fills in the registration form with his/her personal data **2.** the Visitor selects what kind of user he/she is **3.** the Visitor submits the form **4.** the System saves the new account information in its database **5.** the Visitor receives a confirmation message
Extensions	—

Table 5.1: Sign Up Use Case

Login

Actor	Visitor
Related Requirements	[R03], [R04]
Goal in context	[G02]
Precondition	the Visitor access the initial application interface
Successful end condition	the Visitor can now access his/her profile and interact with the system according to his/her privileges
Failed end condition	the Visitor is not able to login. He/She is asked to try again
Trigger	the Visitor selects the "Log In" button
Main Flow	**1.** the Visitor insert username and password **2.**the System verifies his/her credentials **3.** the Visitor receives a confirmation message
Extensions	—

Table 5.2: Log In Use Case

Request Taxi

Actor	Passenger
Related Requirements	[R05], [R06], [R07], [R08], [R09], [R10], [R11]
Goal in context	[G03], [G05]
Precondition	the Passenger must be logged in
Successful end condition	the Passenger can access an information page with confirmation, taxi code and waiting time
Failed end condition	the Passenger receives an error message
Trigger	the Passenger select the "Request" button to access the request interface
Main Flow	**1.** the System sets the current position of the Passenger as the origin for the ride **2.** the Passenger insert the destination of the ride **3.** the Passenger submits the request to the System **4.** the Passenger receives a confirmation message
Extensions	**3.1** the Passenger selects the "enable sharing" option

Table 5.3: Request Taxi Use Case

Reserve Taxi

Actor	Passenger
Related Requirements	[R05], [R06], [R07], [R08], [R09]
Goal in context	[G04], [G05]
Precondition	the Passenger must be logged in
Successful end condition	the Passenger can access an information page with confirmation, taxi code and summary of the reservation
Failed end condition	the Passenger receives an error message
Trigger	the Passenger select the "Reserve" button to access the reservation interface
Main Flow	**1.** the Passenger insert origin and destination for the ride **2.** the Passenger insert the reservation time **3.** the Passenger submits the reservation to the System **4.** the Passenger receives a confirmation message
Extensions	**3.1** the Passenger selects the "enable sharing" option

Table 5.4: Reserve Taxi Use Case

Enable Sharing Option

Actor	Passenger
Related Requirements	[R12], [R13], [R14], [R15]
Goal in context	[G06]
Precondition	the Passenger must be logged in
Successful end condition	the Passenger can share the ride with other passengers
Failed end condition	there are no other passengers willing to share the ride
Trigger	the Passenger selects the "enable sharing" option from the request or the reservation interface
Main Flow	**1.** the System verifies whether there are other passengers willing to share the ride **2.** the System calculate the fee percentage for the ride **3.** the Passenger receives a message with information about the shared ride
Extensions	—

Table 5.5: Enable Sharing Option Taxi Use Case

See Receipt

Actor	Passenger
Related Requirements	[R16]
Goal in context	[G07]
Precondition	the Passenger has completed the ride
Successful end condition	the Passenger access the receipt of the last ride
Failed end condition	the Passenger receives an error message
Trigger	the Passenger select the "All Receipts" button
Main Flow	**1.** the Passenger access the chronology of his/her rides **2.** the Passenger selects the last ride **3.** the Passenger sees the receipt for the ride
Extensions	**3.1** the Passenger saves the receipt on his/her device **3.2** the Passenger prints the receipt

Table 5.6: See Receipt Use Case

Update Account

Actor	Passenger, Taxi Driver
Related Requirements	[R17]
Goal in context	[G08], [G13]
Precondition	the Passenger/Taxi Driver must be logged in
Successful end condition	the Passenger/Taxi Driver updates the account information
Failed end condition	a System error occurs. An error message is shown to the Passenger/Taxi Driver
Trigger	—
Main Flow	**1.** the Passenger/Taxi Driver selects the "My Account" tab from the home page of the application **2.** the Passenger/Taxi Driver inserts the updated data **3.** the Passenger/Taxi Driver clicks on the "Update Account" button
Extensions	—

Table 5.7: Update Account Use Case

Accept Incoming Requests

Actor	Taxi Driver
Related Requirements	[R18]
Goal in context	[G11]
Precondition	the Taxi Driver has received an incoming request or reservation
Successful end condition	the Taxi Driver accepts the request or reservation
Failed end condition	a System error occurs. An error message is shown to the Taxi Driver
Trigger	—
Main Flow	**1.** the System shows to the Taxi Driver a request or reservation coming from a Passenger **2.** the Taxi Driver clicks on the "Accept" button
Extensions	—

Table 5.8: Accept Incoming Requests Use Case

Decline Incoming Requests

Actor	Taxi Driver
Related Requirements	[R18]
Goal in context	[G11]
Precondition	the Taxi Driver has received an incoming request or reservation
Successful end condition	the Taxi Driver declines the request or reservation
Failed end condition	a System error occurs. An error message is shown to the Taxi Driver
Trigger	—
Main Flow	**1.** the System shows to the Taxi Driver a request or reservation coming from a Passenger **2.** the Taxi Driver clicks on the "Decline" button
Extensions	—

Table 5.9: Decline Incoming Requests Use Case

Receive Incoming Requests

Actor	Taxi Driver
Related Requirements	[R19]
Goal in context	[G10]
Precondition	the Taxi Driver must be logged in
Successful end condition	the Taxi Driver sees the request or reservation details and he/she is asked to click on "Accept" or "Decline"
Failed end condition	a System error occurs. An error message is shown to the Taxi Driver
Trigger	the Passenger sends a request or reservation to the System
Main Flow	**1.** the System receives the reservation or request from a Passenger **2.** the Taxi Driver receives the reservation or request from the System
Extensions	—

Table 5.10: Receive Incoming Requests Use Case

Visualize Information About Optimal Route

Actor	Taxi Driver
Related Requirements	[R20], [R21]
Goal in context	[G12]
Precondition	—
Successful end condition	the Taxi Driver sees the information about the optimal route for the booked ride
Failed end condition	a System error occurs. An error message is shown to the Taxi Driver
Trigger	the Taxi Driver accept a request or reservation
Main Flow	**1.** the System calculates the optimal route for the accepted ride **2.** an information message is shown to the Taxi Driver
Extensions	—

Table 5.11: Visualize Information About Optimal Route Use Case

Introduce Modifications

Actor	Developer
Related Requirements	[R22]
Goal in context	[G14]
Precondition	the Developer must be logged in
Successful end condition	the Developer successfully introduces modification to the System
Failed end condition	a System error occurs. An error message is shown to the Developer
Trigger	—
Main Flow	**1.** the Developer access the "Modification" interface **2.** the Developer introduces modifications **3.** the Developer submits his/her modifications
Extensions	—

Table 5.12: Introduce Modifications Use Case

Access Privileged Information

Actor	Developer
Related Requirements	[R22]
Goal in context	[G14]
Precondition	the Developer must be logged in
Successful end condition	the Developer sees the information the he/she needs
Failed end condition	a System error occurs. An error message is shown to the Developer
Trigger	—
Main Flow	**1.** the Developer access the "Information" interface **2.** the Developer sees all the information about the System
Extensions	—

Table 5.13: Introduce Modifications Use Case

5.3 Class Diagram

Here is presented the UML class diagram in Figure 5.2. This diagram will be updated during the developing process especially by adding all methods:

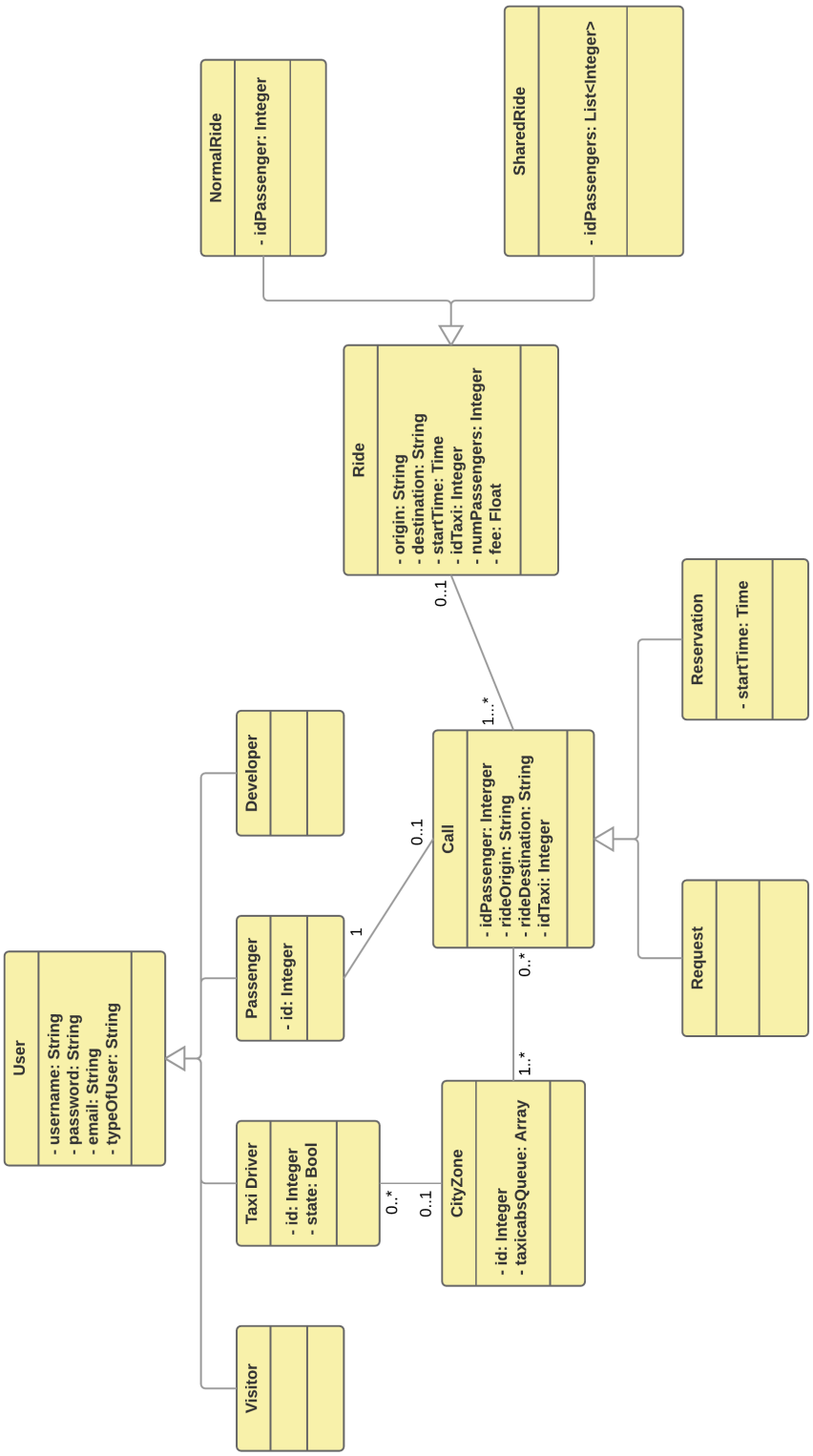

Figure 5.2: Class Diagram

5.4 Sequence Diagrams

Log In

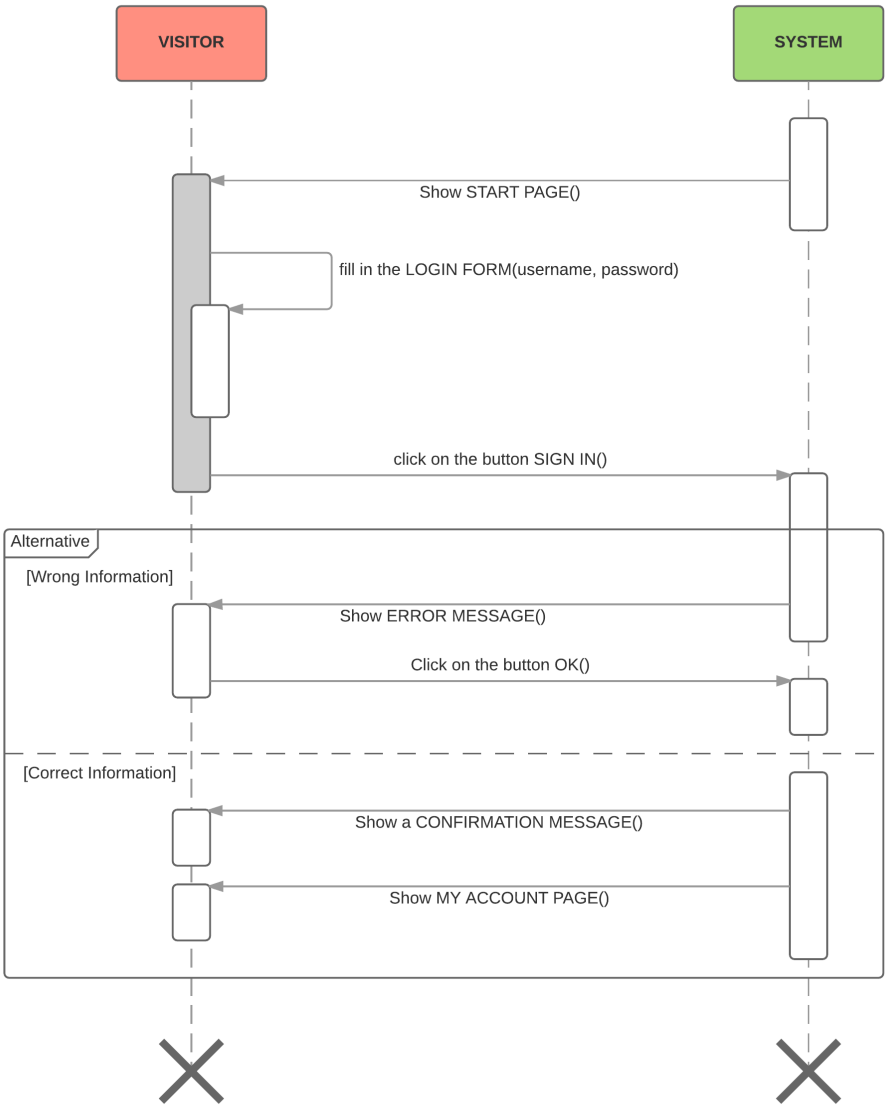

Figure 5.3: Sequence Diagram for Log In

Request a Taxi Ride

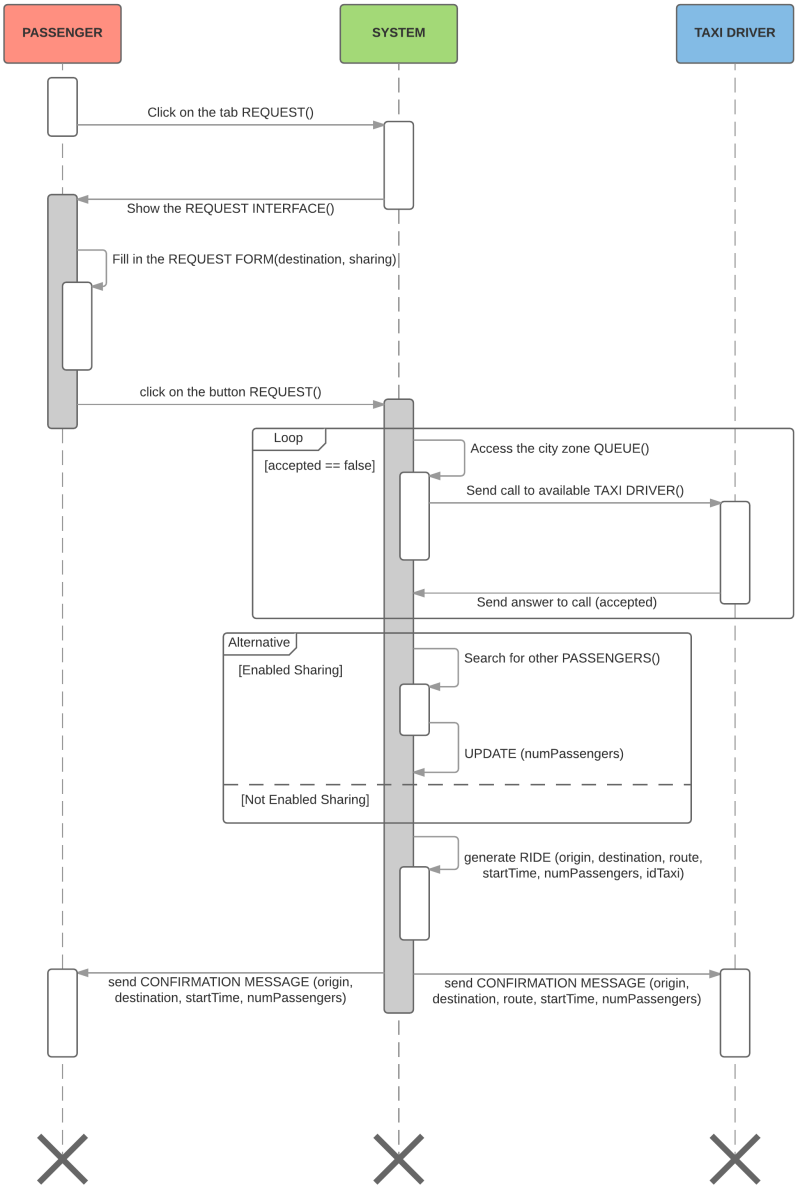

Figure 5.4: Sequence Diagram for Request a Taxi Ride

Reserve a Taxi Ride

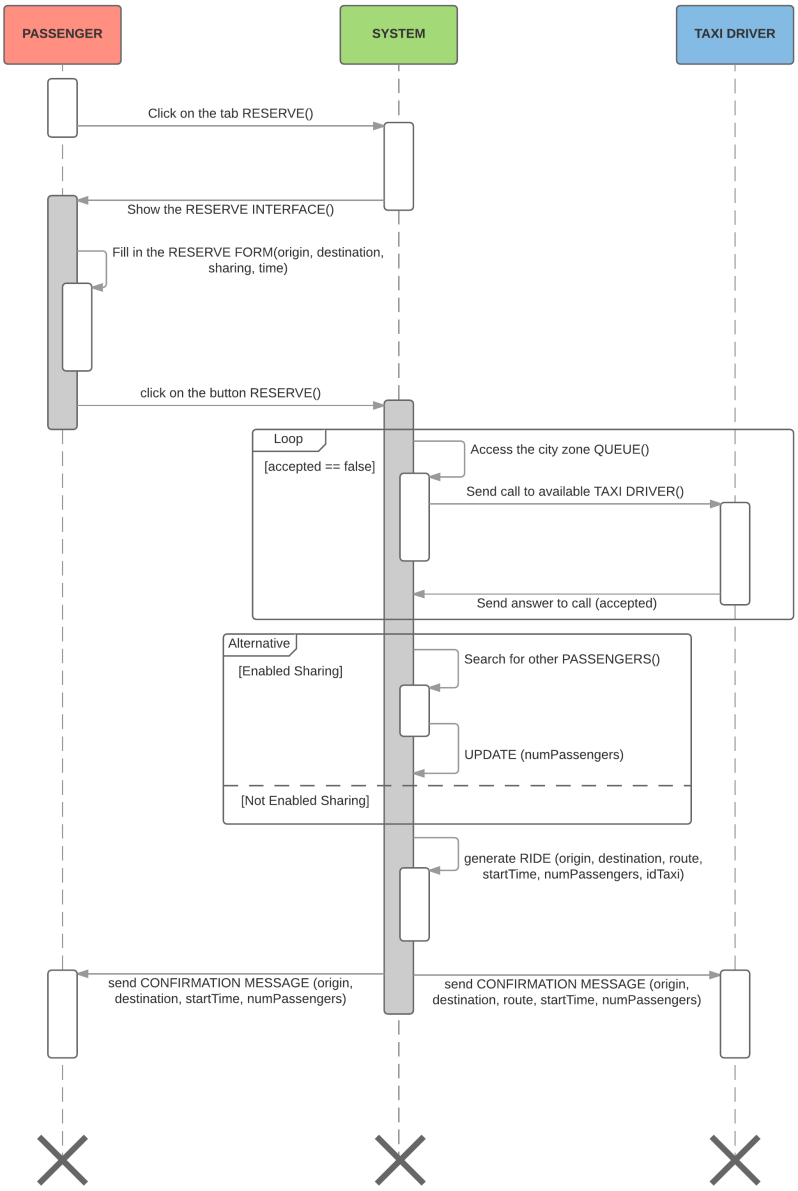

Figure 5.5: Sequence Diagram for Reserve a Taxi Ride

Introduce Modifications in the System

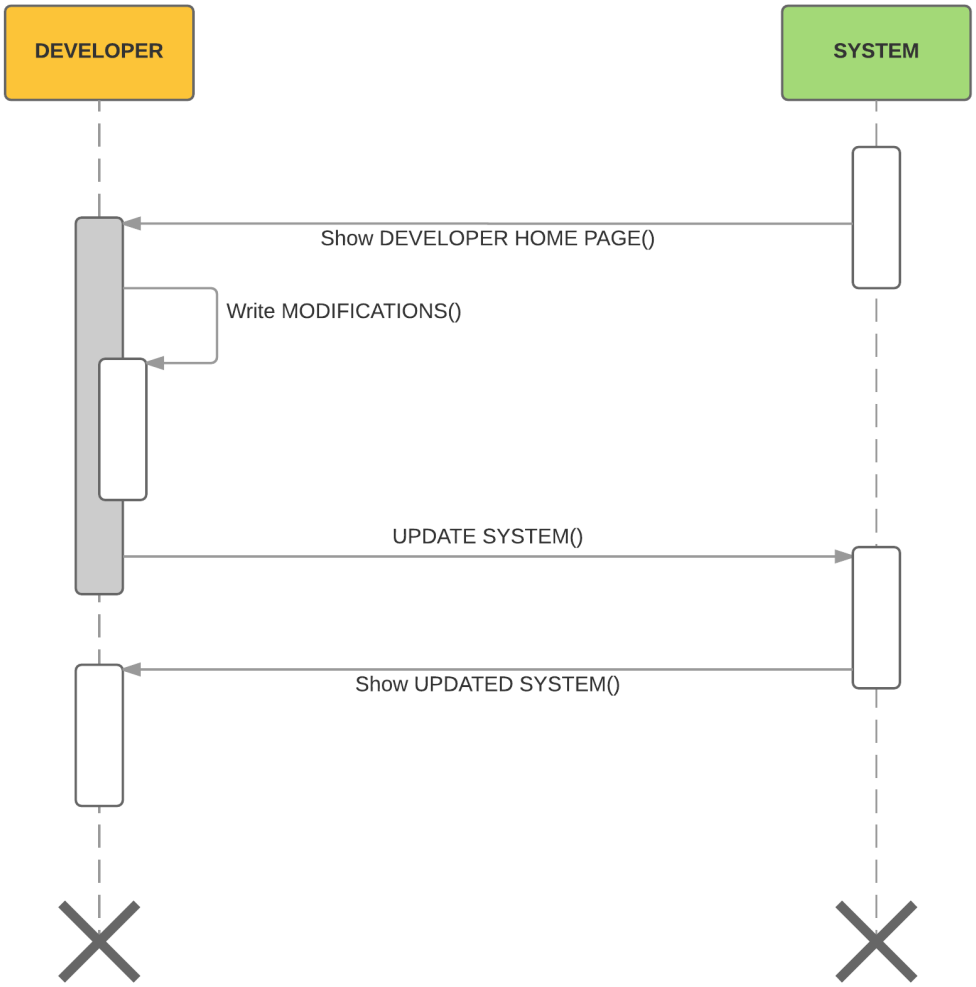

Figure 5.6: Sequence Diagram for Introducing Modifications

5.5 State Chart Diagrams

In this section the behavior of some entities presented in Figure 5.2 is exposed using UML state chart diagrams. The following state chart diagrams will give a simplified vision of entire application:

State Chart Diagram for Passengers

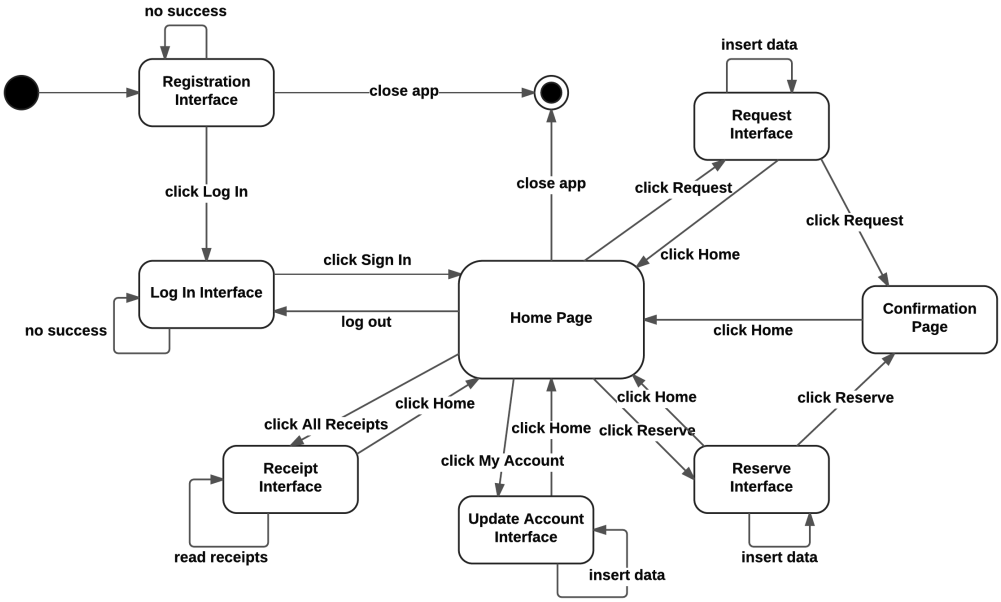

Figure 5.7: State Chart Diagram for Passengers

State Chart Diagram for Taxi Drivers

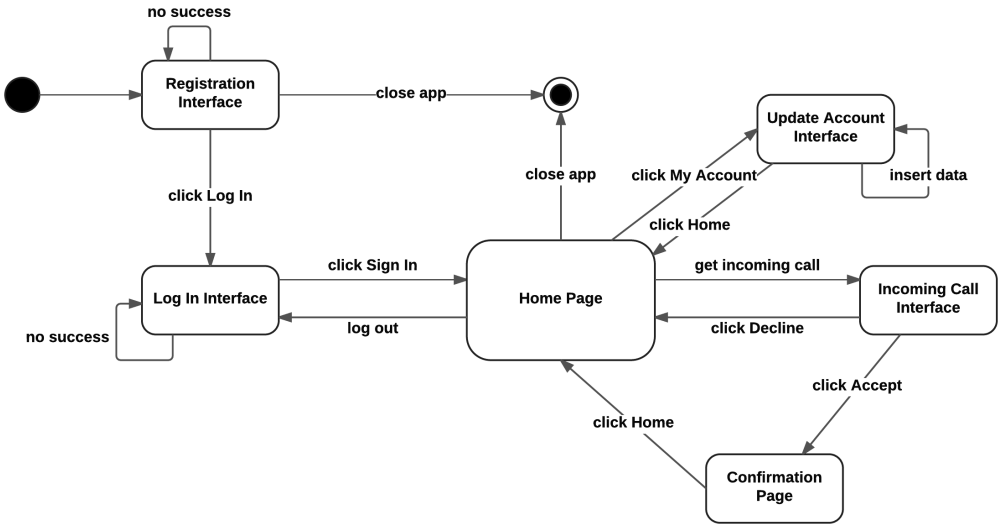

Figure 5.8: State Chart Diagram for Taxi Drivers

Chapter 6

Alloy Modeling

6.1 Alloy Code

```
/*************** CLASSES ***************/

sig Queue {
        first: one AvailableTaxiDriver,
        associatedArea : one TaxiArea
}

abstract sig TaxiDriver {}

sig AvailableTaxiDriver extends TaxiDriver {
        next: lone AvailableTaxiDriver
}

sig BusyTaxiDriver extends TaxiDriver {}

sig Passenger {
}

abstract sig Ride {
        driver : one BusyTaxiDriver,
        origins : some Location,
        passengers : some Passenger,
        destinations : some Location
} {
        origins not in destinations
}

sig SharedRide extends Ride {
} {
        #passengers < 4 ∧
        #passengers > 1 ∧
        #passengers = #destinations ∧
```

```
        #passengers = #origins
}

sig SingleRide extends Ride {
}{
        #passengers = 1 ∧
        #destinations = 1 ∧
        #origins = 1
}

sig Location {}

sig TaxiArea {
        locations : some Location
}

sig Request_Reservation {
        requester : one Passenger,
        requesterArea : one TaxiArea,
        associatedQueue : one Queue
}


/*************** CONSTRAINTS ***************/

// ensures that no taxi driver is the subsequent of himself inside a queue
fact nextNotReflexive { no n:TaxiDriver | n = n.next }

// ensures that each taxi driver is associated to only one queue
fact onlyOneQueuePerAvailableTaxiDriver {
        all d:AvailableTaxiDriver | one q:Queue | d in q.first.*next
}

// ensures that the last taxi driver of the queue exists and is unique
fact notCyclicQueue {
        no d:TaxiDriver | d in d.^next
}

// ensures that all taxi drivers marked as busy are making only one ride
fact AllBusyTaxiDriverAreInARide {
        all d:BusyTaxiDriver | one r:Ride | r.driver = d
}

// ensures that all passengers in a ride are in only one ride
fact OnlyOneRidePerTimePerPassenger {
        all disj r1, r2:Ride | disj[r1.passengers, r2.passengers]
}

// ensures that every area has an associated queue
fact OneQueuePerLocationandViceversa {
        all disj q1, q2 : Queue | q1.associatedArea ≠ q2.associatedArea
}
```

```
//ensures that every area contains different locations and a location is
    ↪ inside only one area
fact DisjointAreasContainsDifferentLocations {
        all disj a1, a2 : TaxiArea | disj[a1.locations, a2.locations]
}

// ensures that exists one and only one queue for every taxi area
fact OneQueuePerArea {
        all a:TaxiArea | one q: Queue | a = q.associatedArea
        all disj q1, q2 : Queue | q1.associatedArea ≠ q2.associatedArea
}

// ensures that in a shared ride all origins belong to the same area and all
    ↪ destinations belong to the same area
fact OriginAndDestinationConsistency {
        all r: SharedRide | one a: TaxiArea | r.origins in a.locations
        all r: SharedRide | one a: TaxiArea | r.destinations in a.locations
}
// ensures that every passenger is associated to at most one active request or
    ↪  reservation
fact AtMostOneRequest_ReservationPerPassenger {
        no disj re1, re2 : Request_Reservation | re1.requester = re2.requester
}

//ensures that every request or reservation is associated to the queue
    ↪ belonging to the requester area
fact RightQueueForRequest_Reservation {
        all re:Request_Reservation | re.requesterArea = re.associatedQueue.
            ↪ associatedArea
}


/*************** ASSERTIONS ***************/

// check that every taxi driver belong to one queue only
assert UniqueQueuePerTaxiDriver {
        no disj q1, q2:Queue | one d:AvailableTaxiDriver | (d in q1.first.*
            ↪ next and d in q2.first.*next)
}
check UniqueQueuePerTaxiDriver

// check that every busy taxi driver is making only one ride
assert OnlyOneRidePerTaxiDriver {
        no disj r1, r2 : Ride | r1.driver = r2.driver
}
check OnlyOneRidePerTaxiDriver

// check that every passenger in a ride is in only one ride
assert OnlyOneRidePerPassenger {
        no disj r1, r2 : Ride | not disj[r1.passengers, r2.passengers]
}
```

```
check OnlyOneRidePerPassenger

// check that an area corresponds to only one queue
assert OneAreaPerQueue {
        no disj q1,q2 : Queue | q1.associatedArea = q2.associatedArea
}
check OneAreaPerQueue

// check that disjoint areas involves different locations
assert OneAreaPerLocation {
        no disj a1,a2 : TaxiArea | not disj[a1.locations, a2.locations]
}
check OneAreaPerLocation

//check the corrispondency between taxi areas and queues
assert OneQueuePerArea {
        no a:TaxiArea | all q : Queue | a ≠ q.associatedArea
}
check OneQueuePerArea

// check that for every shared ride, all the origins and destinations belong
    ↪ to the same 2 taxi zones
assert OriginsAndDestinationsAreConsistent {
        no disj a1, a2 : TaxiArea | one r:SharedRide | r.origins in (a1 + a2).
            ↪ locations
}
check OriginsAndDestinationsAreConsistent


/*************** PREDICATES ***************/
pred show() {
        #Queue > 1 ∧
        #SharedRide > 1 ∧
        #SingleRide > 1 ∧
        #Request_Reservation > 1
}

run show for 6


pred showRides() {
        #Passenger>1 ∧
        #TaxiArea > 1
}

run showRides for 9 but exactly 2 SharedRide
```

6.2 Worlds generated

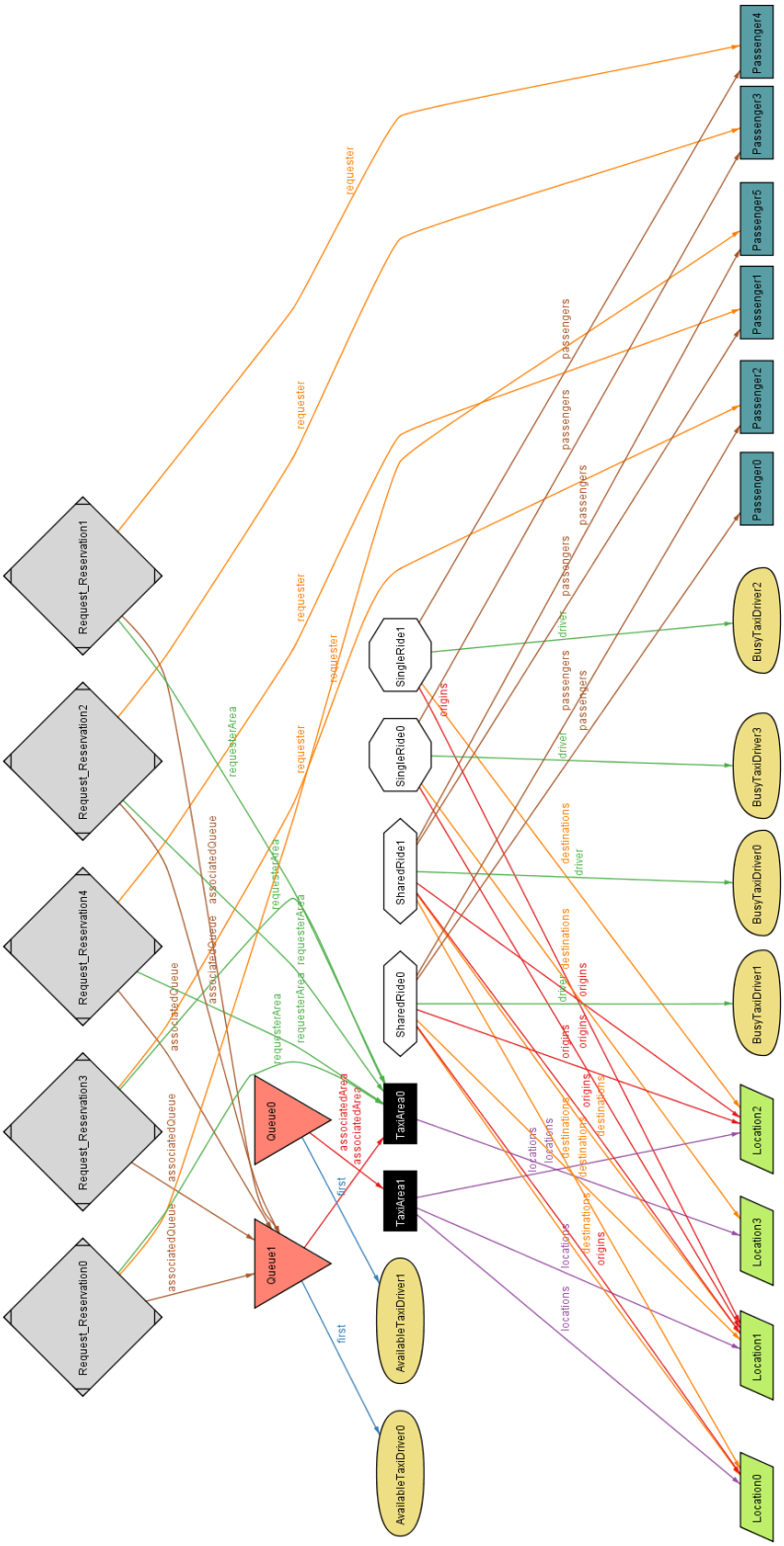

Figure 6.1: World generated from the execution of predicate show()

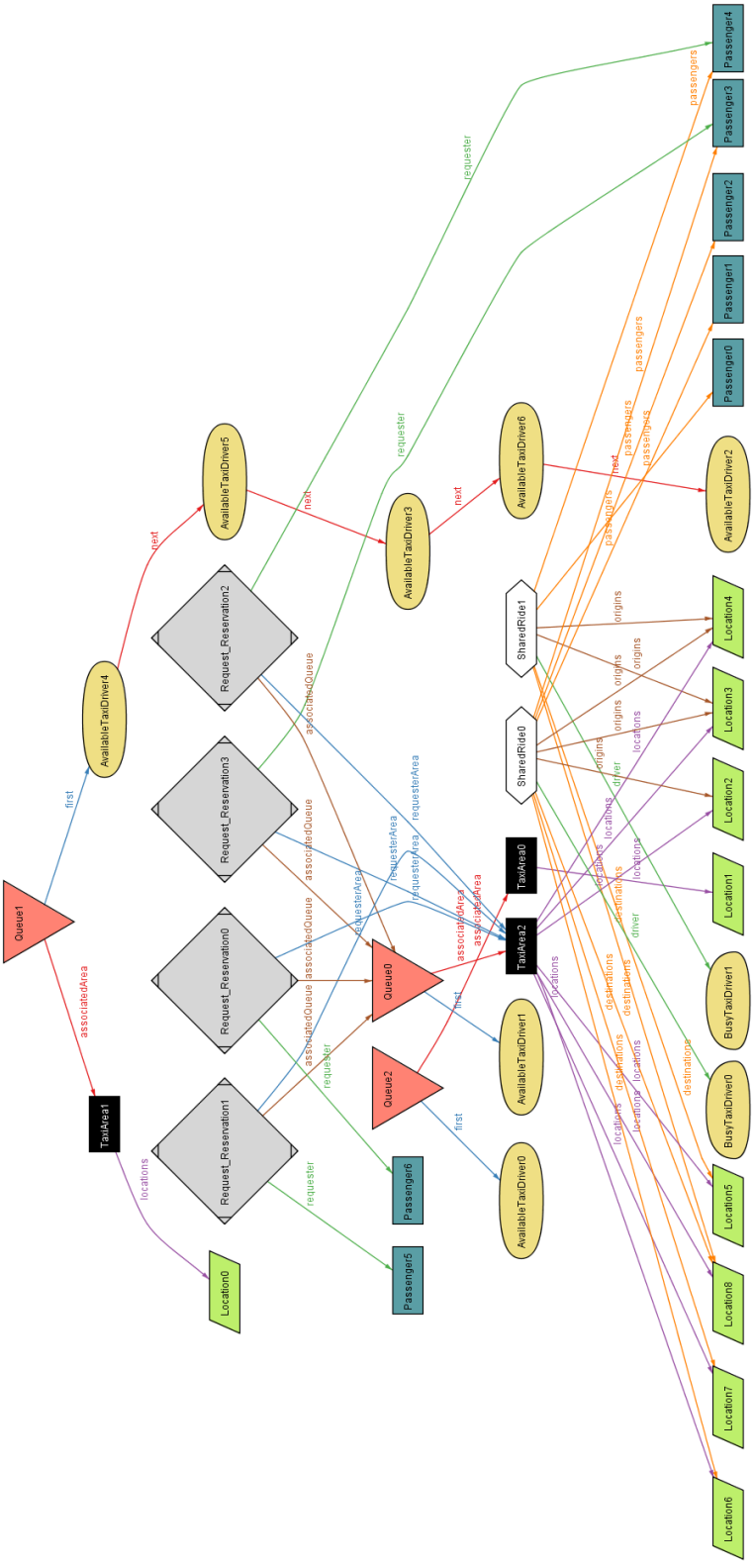

Figure 6.2: World generated from the execution of predicate showRides()

Chapter 7

Other Information

This chapter contains information about the used tools and the hours of work done by the members of the group.

7.1 Used Tools

In this first requirements study phase the following tools were used:

- LaTeX and TeXMaker editor: to redact and to format this document
- Lucidchart (https://www.lucidchart.com/): to create the State Charts, the Class Diagram, the Sequence Diagrams, the Use Case Diagram and the mockups
- Alloy Analizer (http://alloy.mit.edu/alloy/): to prove the consistency of our model

7.2 Working Hours

First Name	Last Name	Total Hours
Mattia	Crippa	25h
Francesca	Galluzzi	22h
Marco	Lattarulo	24h