Requirements Analysis and Specifications Document

myTaxiService

Fabio Catania, Matteo Di Napoli, Nicola Di Nardo

November 6, 2015

Contents

1 Introduction

1.1 Purpose

This Requirement Analysis and Specification Document aims to provide a detailed description of the ICT in case. In particular its main goals are to completely delineate the (functional and non-functional) intended features of the system, to present the constraints under which it must operate and to offer a preliminary glimpse of the software application's User Interface (UI).

1.2 Audience

This document is primarily intended for both the customer for its approval and the developing team as reference material. Furthermore it could be used in the future by system analysts who want to integrate this system with others.

1.3 Current System

At the moment in the city of London taxis are reserved by phone: there is a telephone exchange that receives the calls, an operator manually uses a GPS to locate the nearest to the client available taxi and then communicates the request to the selcted taxi driver, who will take care of the transfer.

1.4 Scope

The software aims at optimizing the taxi service in London. The software is intended to:

- integrate the current telephonic reservation system by offering the possibility to request a taxi through a web application and a mobile app;
- guarantee a fair management of the taxi queues;
- offer the possibility to share the taxi with other people.

1.5 Goals

Main goals of the application are the following:

G1 A user should be able to call a taxi either through the web application or the mobile app and request should be soon accommodated by a nearby taxi.

G2 A user should be able to reserve a taxi by specifying the origin and the destination of the ride at least two hours before the departure.

G3 A user should be able to enable the taxi sharing option. This means that he would be ready to share a taxi with others if possible, thus sharing the cost of the ride.

G4 Taxi drivers should be able, using a mobile application, to inform the system about their availability and to confirm that they are going to take care of a certain call.

G5 The number of trips by every single taxi should be distributed equally.

G6 There should be the possibility to develop and easily integrate in the future additional services on top of the basic one.

1.6 Stakeholders

The ICT is commissioned by the government of London, United Kingdom. It is going to improve the taxi drivers work and provide an easy and immediate access to the service to customers.

At the request's time London is the most populated city of the country (8,539 million inhabitants) and is one of the most important touristic and economic centers in Europe. Its internet-connected inhabitants, tourists and business people are the possible clients for the system.

1.7 Actors

- The passenger: the client who makes the call through the mobile app or the web site and who is carried by the taxi.
- The taxi driver: the person who drives the taxi, who can choose to take care of a certain call through a mobile application.

1.8 Definitions, acronyms and abbreviations

1.8.1 Definitions

- Application Store: An installed application on mobile phone which helps user to find new compatible applications with mobile phone platform and download them from Internet
- Customer: user that interacts with the application to perform requests.
- Regular Call: instantaneous taxi request performed by a single user by the application, to be managed in the shortest time possible.
- Reservation Call: taxi request performed by a single user to be accomplished by the system in a specified time, position and to a specified destination.
- Sharing Call: taxi request performed by a single user to share a taxi with another user of the system. Coupling is provided autonomously by the application.
- Stakeholder: a person who is involved with an organization and therefore has responsibilities towards it and an interest in its success.

- Taxi Driver: licensed taxi driver registered on the application and interacting with it to accomplish customers' requests.
- User: person who interacts with the application.

1.8.2 Acronyms

- API: Application Programming Interface.
- DB: Data Base.
- DBMS: Data Base management system.
- GPS: Global Position System.
- HTML: HyperText Markup Language.
- ICT: Information and Communication Technology.
- IEEE: Institute of Electrical and Electronics Engineers.
- Java EE: Java Enterprise Edition.
- OS: Operating System.
- RASD: Requirements Analysis and Specification Document.
- UML: Unified Modeling Language.

1.8.3 Abbreviations

[Gn]: n-goal.

[Rn]: n-functional requirement.

[Dn]: n-domain assumption.

1.8.4 References

- IEEE Software Engineering Standards Committee *IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications*, October 20, 1998.
- Specification Document: Software Engineering 2 Project, AA 2015-2016 Assignments 1 and 2.pdf

1.9 Document Overview

This document is structured following the IEEE standards to simplify its consultation. It is essentially organized in four part:

- Section 1: Introduction, it gives a description of document and some basic information about software;
- Section 2: Overall Description, 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 functionalities 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 software used to generate it.

2 Overall Description

This section is used to provide a high-level description of the system, as well as identify the users involved and help explain their roles.

2.1 Product Perspective

It will be released a web application that will bring significant changes about requests for a taxi service, not completely automated so far. Specifically, the system will be integrated with the existing one in order to enrich its functionalities without having to replace it entirely. The system will also provide different user interfaces depending on the class of users which they belong to.

Besides these specific user interfaces, the application offers programmatic interfaces to allow the development of other services on top of the basic one, as in particular a taxi reservation option and a taxi sharing one.

2.2 Product Functions

The system shall ensure the taxis are dispatched quickly, in order to reduce waiting time and to increase the ability of the drivers to deal with customers. To facilitate these needs, taxi drivers use a mobile application to inform the system about their availability, their position, through a GPS system, and finally to confirm that they are going to take care of a certain request.

Customers, on the other side, can request a taxi either through a web application or a web app, followed by a reply which defines the incoming taxi code and the waiting time. The system will guarantee a fair management of taxi queues based on the zoning of the city. A customer can, also, reserve a taxi by specifying the origin and the destination of the ride. The system will confirm the reservation to the customer and it will allocate a taxi for the meeting time.

Finally, a customer can make a taxi sharing call in order to share the cost of the ride, when it is possible for the system to couple it with appropriate further sharing calls already made or to be made. In this case, the system arranges the route for the taxi driver, defines the fee for all persons sharing the taxi and informs passengers and the taxi driver.

2.3 User Characteristics

There are four main classes of actors which will use the system. The first group of users is the System Administrators. System Administrators are concerned with data integrity and system stability. These users have the highest computer skill set and are capable of supporting a computer network. Their interaction with the system is very limited and is only necessary for backing up and achieving data from the database or to provide basic computer support to the other users of the system.

The second group is composed by the Management Team. They interact with the system to view reports and statistics about the quality of service their

employees are providing. They have basic computer skills and they spend most of their time ensuring that the incoming requests are computed quickly and that the taxis are allocated in a very efficient way.

The Taxi Drivers team makes up the third group. These users are the only ones which interact in person with customers, when the ride is about to happen. They have basic computer knowledge which allows them to communicate information about their position and their availability in order to have the system working on it.

Customers make up the last group of application users. They can have any experience with the application, any level of computer skills and they may be differently familiar with the city. This group has to deal with different interfaces depending on the type of service they are requesting, both through a web application and a mobile app.

2.4 Constraints

2.4.1 Regulatory policies

myTaxiService doesn't have to meet any regulatory policy.

2.4.2 Software limitations

- System shall need Windows 7 or greater, Mac OS X 10.10 Yosemite or greater or GNU/Linux Ubuntu if the user wants to use it on the web application.
- System shall need Google Chrome 46.0.2490, Mozilla Firefox 41.0.2 or Windows Edge 20.10240 in order to run the application on the web.
- System shall need Android 4.0 or greater or IOS 7 or greater if the user wants to use it on a mobile app.
- The database shall be stored on a Windows, Linux, or Unix server using Apache and MySql, and all the computers shall need to be networked together so that all can have access to the database.
- Both the web portal and the mobile application will be constrained by the capacity of the database. Since the database is shared between both application it may be forced to queue incoming requests and therefor increase the time it takes to fetch data.

2.4.3 Hardware limitations

Both parts, customer users and driver users, must have the access to Internet in order to use the application.

2.4.4 Parallel operation

myTaxyService must support parallel operations from different users and different type of users at the same time, specially when working on the database.

2.4.5 Interfaces to other applications

There is no actual interaction with other external applications, but a developers-oriented interface should be provided to enable further interactions.

2.5 Assumptions

D1 The system assumes that the users have adequate skill with using computers and computer software.

D2 Only the Management Team and the System Administrators, as privileged users, can access to the database. The first group to view and to analyze stored data, the second one for maintenance issues.

D3 Each taxi driver shall need a device capable of running the application in order to forward the necessary information to the system.

D4 Each taxi driver must communicate to the system the required information.

D5 It is assumed that customers have at least a PC or a mobile to use the application.

D6 It is assumed that customers can understand English in order to fill correctly the request.

D7 Each customer can make only one request at a time.

D8 Each customer has to insert a valid telephone number.

D9 Each customer has to use a valid address both for the origin of the ride, when manually inserted, and for the destination, making a reservation request.

D10 If a customer made a reservation request, he must be in the specified place at the specified time.

D11 The city is divided in zones, each one approximately $2km^2$. Each zone is associated to a queue of taxis.

D12 A user should be able to download the mobile application through either an app store or similar service on the mobile phone. The application should be free to download.

D13 When a new/updated version of the software is released the user should check for these manually. The download of the new release should be done through the mobile phone in the same way as downloading the mobile application.

D14 It is assumed that the GPS component works in order to let the application run as intended.

2.6 Apportioning of requirements

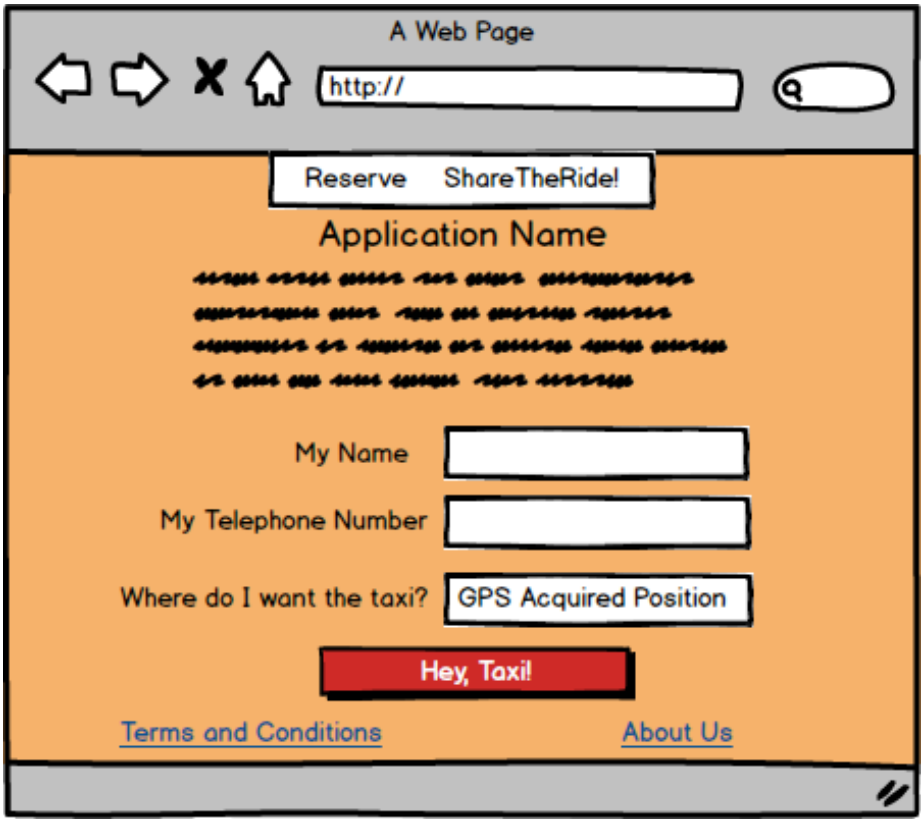
In the case that the project is delayed, there are some requirements that could be transferred to the next version of the application.

3 Specific Requirements

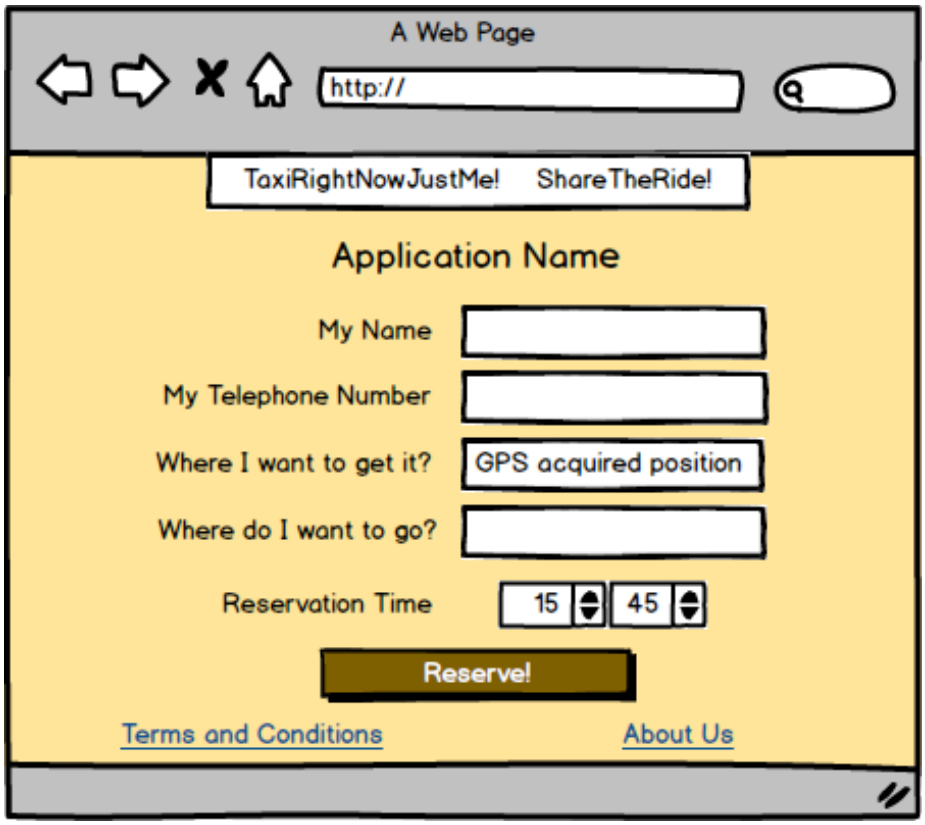
3.1 User Interface Requirements

The following are mockups that represent a possible user interface model.

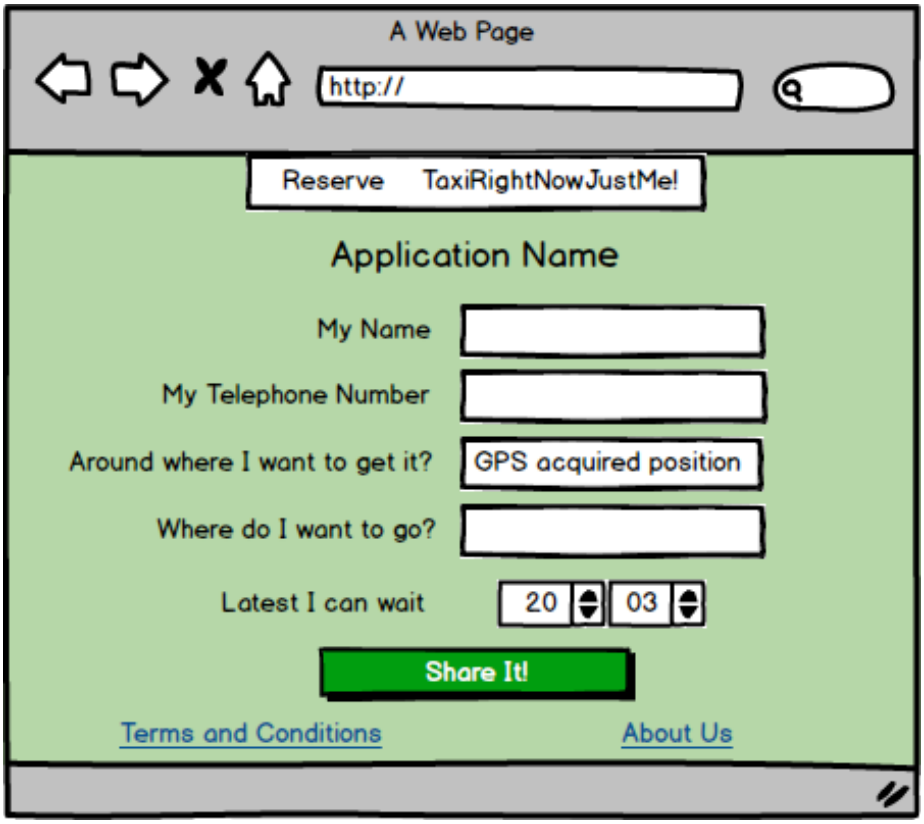
3.1.1 Regular Call Web Interface



3.1.2 Reservation Web Interface



3.1.3 Sharing Call Web Interface

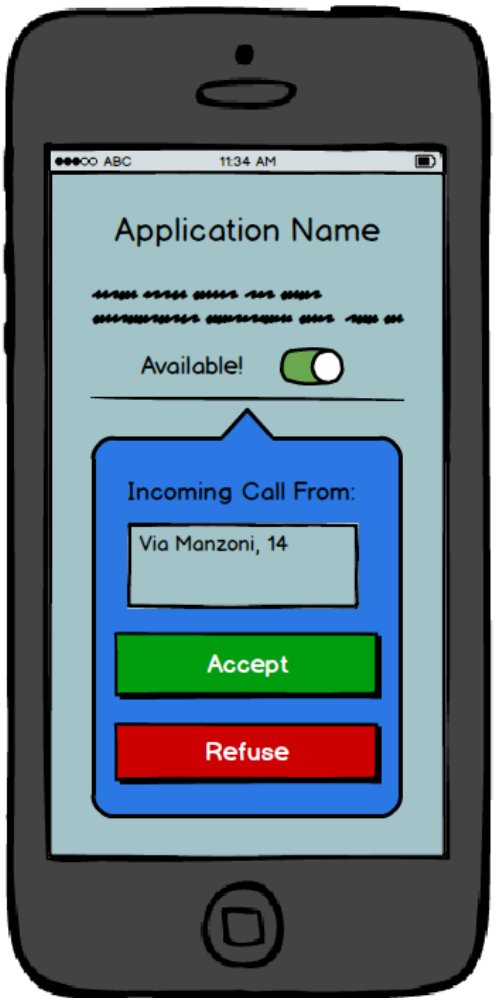


3.1.4 Regular Call Mobile Interface

3.1.5 Reservation Mobile Interface

3.1.6 Sharing Call Mobile Interface

3.1.7 Taxi Driver Mobile Interface



3.2 Functional Requirements

3.2.1 Regular Call and Reservation Functionalities

R1 The system has to provide a sign up functionality to taxi drivers that univocally defines the ID of their vehicle.

R2 The system has to collect and store the telephonic numbers that customers will provide at their first access as contacts for their following requests.

R3 The system has to maintain ordered queues of all the available taxis in a given zone.

R4 The system has to guarantee a FIFO order in the managing of the queues for each city zone: the first taxi driver that gives his availability in a given zone must be the first to be assigned by the system, and so on.

R5 The system has to guarantee that any given taxi identified by its ID can be present only in one zone queue at the same time.

R6 The system has to provide an estimation of the time needed for the selected taxi to get in the required location.

R7 The system has to notify taxi drivers when their taxi is selected for a ride from the queue mechanism and elaborate their answer or notify following taxi driver if the answer doesn't arrive within 1 minute.

R8 The system has to notify a customer if his call has been accepted and provide him the univocal ID of the selected taxi and an estimated waiting time.

R9 The system must notify customers in case after 5 minutes since the request has been done there's no taxi available (because the queue is empty at the moment or because no taxi driver is accepting the request), inviting him to retry in a little time.

R10 The system must be able to accept reservations till two hours before the requested time and allocate a taxi for each reservation 10 minutes before it giving it priority over the regular instantaneous calls.

R11 The system has to notify a customer that his reservation has been accepted, and notify him again 10 minutes before the fixed departure time when a taxi has been allocated for the reservation providing him the univocal ID of the selected taxi.

R12 The system must allow customers to cancel their reservation till 15 minutes before the chosen departure time.

3.2.2 Taxi Sharing Functionalities

R12 In case of call that requires sharing a taxi, system must be able to automatically couple incoming requests in the same zone within the maximum waiting time specified by the customers.

R13 In case of confirmed sharing call, system must be able to calculate weighted fees on the base of the length of any route required by each customer.

R14 In case of confirmed sharing call, system must elaborate a starting point where the taxi has to arrive in order to recollect the passengers, choosing it among a definite set of starting points previously defined for each zone: system has to choose the nearest starting point to all the passengers positions.

R15 The system must notify all the involved customers when a shared call is arranged, providing them the univocal ID of the selected taxi, the estimated waiting time, the starting point, all the expected stop-overs and the amount of the fee entitled to each one.

3.3 Non Functional Requirements

3.3.1 Performance Requirements

Performance must be acceptable to guarantee a good grade of usability: the use of powerful algorithms will guarantee a high efficiency to the system. 95% of the transactions shall be processed in less than 1 second, so that the performance from the user's point of view will essentially be bounded by his/her internet connection.

User should be able to perform any kind of request exploring not more than two layers (on mobile) or two pages (on web app). The system will support 1.000.000 taxi drivers and 10.000.000 clients simultaneously connected, and its DB will be able to store 100.000.000 of users (both taxi drivers and clients).

3.3.2 Design Constrains

The main part of the system will run on the server with Java EE. The mobile apps will be encoded with Java for Android OS and with Swift for IOS, while the web application will be designed with HTML.

The system has to integrate an already existing GPS system that can keep track of the position of any registered vehicle at any moment.

3.3.3 Software System Attributes

Portability If the user's device follows the minimal software prerequisites, the app will run as intended.

Maintainability The system will be highly maintainable thanks to a modular structure and maximum value of method's complexity rate fixed to 10. This application will be documented in detail to help future developers explaining how the application works and how it has been developed.

In addition, the code will be periodically reviewed to keep the maintainability index greater than 60%.

Finally, the system must provide a developers-oriented interface to access its functionalities for further development of additional services without allowing access to implementation details.

Availability The taxi service will be accessible anytime. To achieve this goal could be used a dedicated server or all system could be hosted into a cloud platform. The second solution guarantees more availability and could reduce the cost for the dedicated server maintaining a high level of performance.

The system will take into account situations in which a user loses internet connection or cannot establish a connection to the server. These users will still be able to use the app, but requests and relative transactions posted while disconnected will be cached until the connection is restored.

Security The app is not password-protected regarding customers' use, there is no method to authenticate the actual customer identity.

The systems guarantees the safety of the user's inserted data filtering the requests from the server to the DB with a firewall.

Usability The application's use is easy to learn. The reservation of the taxi will be guided. There will be feedbacks and notifications to inform users of updates and pop-ups to provide users with instructions.

Reliability There is a tolerance of 0% for algorithms' errors.

3.4 Scenarios

The following are examples of possible sceanrios referring to the application:

1. John has been working all day and has totally forgotten his and her wife's anniversary. When he realizes that, it's already almost dinner time. Normally John would take the train to get home, a crowded and slow train, but since he also has to buy her wife a present and has no time left, he thinks that he'll need a taxi to get to the nearest gift shop hoping to find it open. Dressing quickly and leaving the building, in the elevator he opens myTaxiService application from his smartphone and after having inserted his name and his telephone number quickly calls for a taxi just pressing a button. A taxi driver his contacted by the system and he accepts the call by his mobile. One minute after, a notification is sent on his mobile phone saying that a taxi driver will be waiting for him in 4 minutes outside the building.

2. Since the departure time of her flight is in about four hours, Maria knows that she'll need a taxi to get to the airport in about three hours. She has already prepared her luggage the day before (Maria is a quite anxious person) so she's now investing some quality time scrolling a common social network's blue and white main page. In another window of her browser, without losing contact with the social network's precious updates, she opens myTaxiService web app, reaches the reservation page with just one click, fills the format with his name, number and position and puts as location the city's airport. myTaxiService web app notifies her that her reservation has been accepted. She walks down in the street five minutes before the reservation time (because she's a very anxious person), and five minutes later a taxi arrives and takes her to the airport.

3. Last time Leonard has been to the stadium to support his beloved football team, after the game he got stuck in the parking for about an hour because of the traffic generated by all the car-using supporters. The waiting time was even more obnoxious due to the heavy loss his team got that day. So he thinks this Sunday he'll try to get a taxi to reach the stadium, but since he doesn't want to spend much money he smartly decides to try the taxi sharing functionality of his just downloaded myTaxiService app. Since game starts at 15:00, he puts as latest he can wait 14:30, and his address as desired starting point. In about two minutes he receives a notification that informs him that a shared call has been arranged, and that a taxi will wait for him in 8 minutes at the first corner of the nearest main street. myTaxiService also informs him about the fee he has to pay, that is exactly a half of the normal fare since also the person he's sharing the taxi with is directed to the stadium. When he gets the taxi, Leonard discover that Raphael, the other passenger, is a fanatic supporter of the local football team like him: making the taxi driver really happy, they sing along the football team's hymn for all the duration of the ride.

4. Since Luke is a very funny person, he decides to use myTaxiService web app to try to get a ride to the imaginary planet Tatooine from Star Wars. After having filled the form on the myTaxiService internet page with the name Luke Skywalker and his number, he puts as destination the space port of Mos Eisley. The system doesn't recognize it as a valid request and redirect him on the regular call request page again, asking him to refill the form with an existing destination. Luke enjoys the joke a lot and closes the page.

5. Taxi drivers are involved in a major strike because the city government decided to change the official color of the taxis to a bright and fashionable pink, forcing them to repaint their vehicles. Marc isn't aware of the protest and needs urgently a taxi to get to a hotel. He uses his myTaxiService mobile app to perform a regular single taxi call. The system doesn't have available taxis in that zone since the vast majority of the taxi drivers joined the strike. After 5 minutes in which no taxi driver gave his availability,

the system notify to Marc that there's no taxi available and invites him to retry again in a little time. Marc curses the technology that never works when needed and decides to take a bus.

6. Dalila wants to go to the prom party without bothering his parents asking to carry her there. Judith hopes that Sansone would invite her and bring her to the prom, but since he didn't give any sign of life that day she decides to reserve a taxi by myTaxiService mobile app just in case she has to go alone. Her reservation is confirmed by the system. What Dalila doesn't know is that Sansone already invited Jennifer and he's waiting for her answer: in the exact moment Jennifer tells Sansone that she already has a partner, he calls Dalila and invites her to go to the prom with him, since he considers her his safety net. Dalila as happy as ever accepts the invite, so later on she returns on myTaxiService web app and going in the correct section cancels the reservation for the night. The system confirms the reservation cancellation.

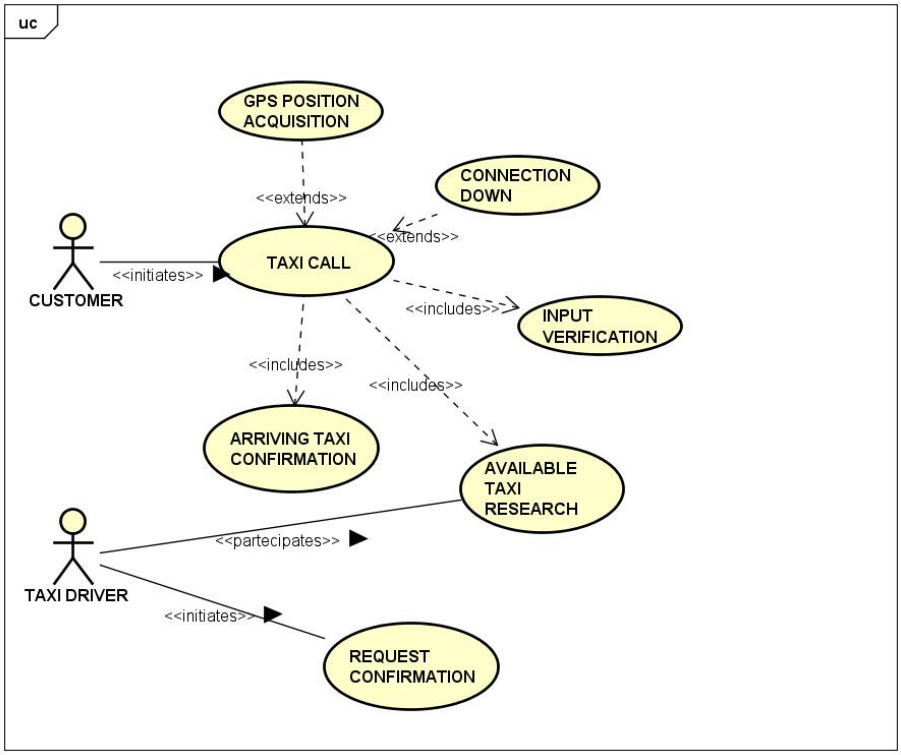
7. Mario is a taxi driver and he's beginning his shift. Some time ago he started using the new myTaxiService application, and he finds it very useful, intuitive and user-friendly. At the beginning of his shift he logs in providing his identifier and his password, and the system successfully add him to the available taxis. At lunch time, he turns off his availability button from the phone and enjoys a very tasty sandwich drinking non-alcoholic beer. The system removes him from the available taxis list.

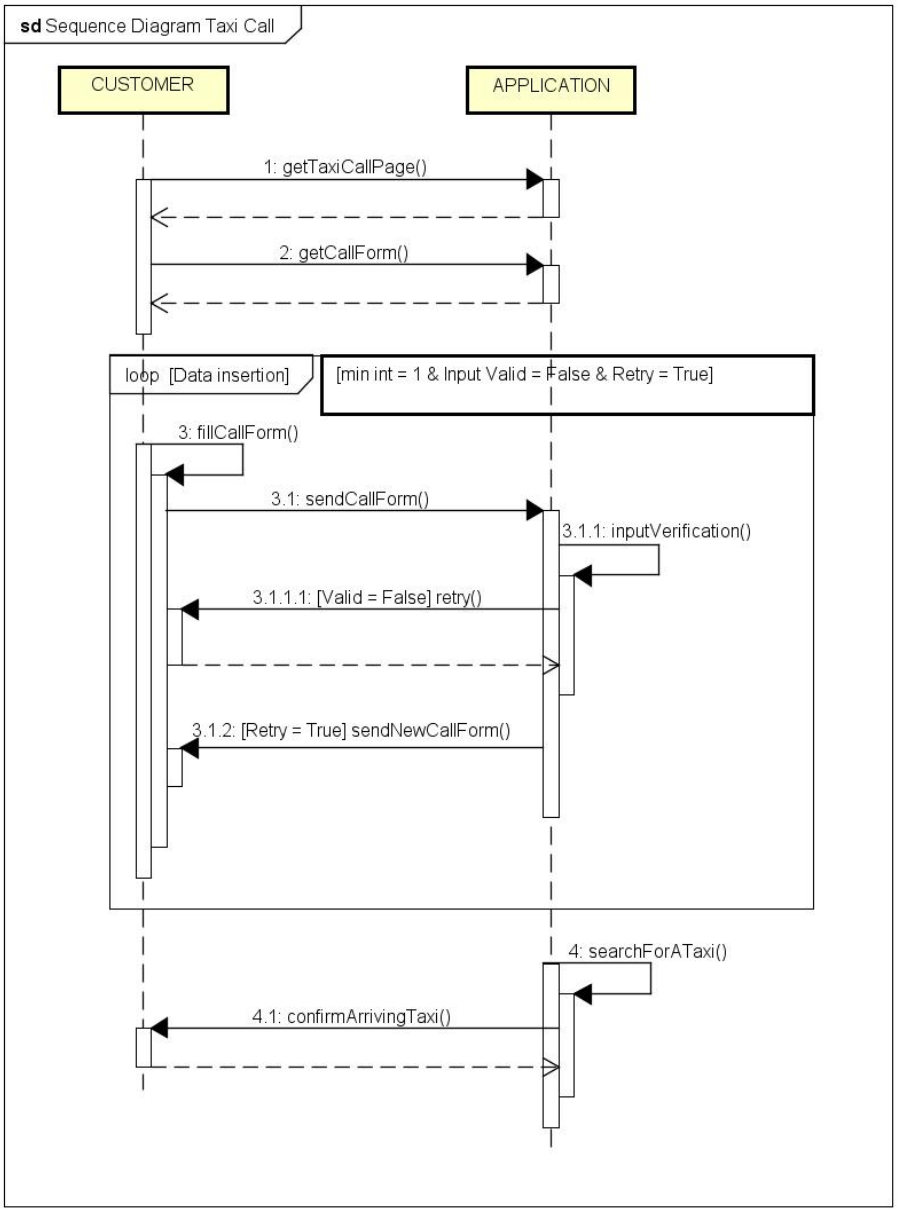
3.5 Use Cases

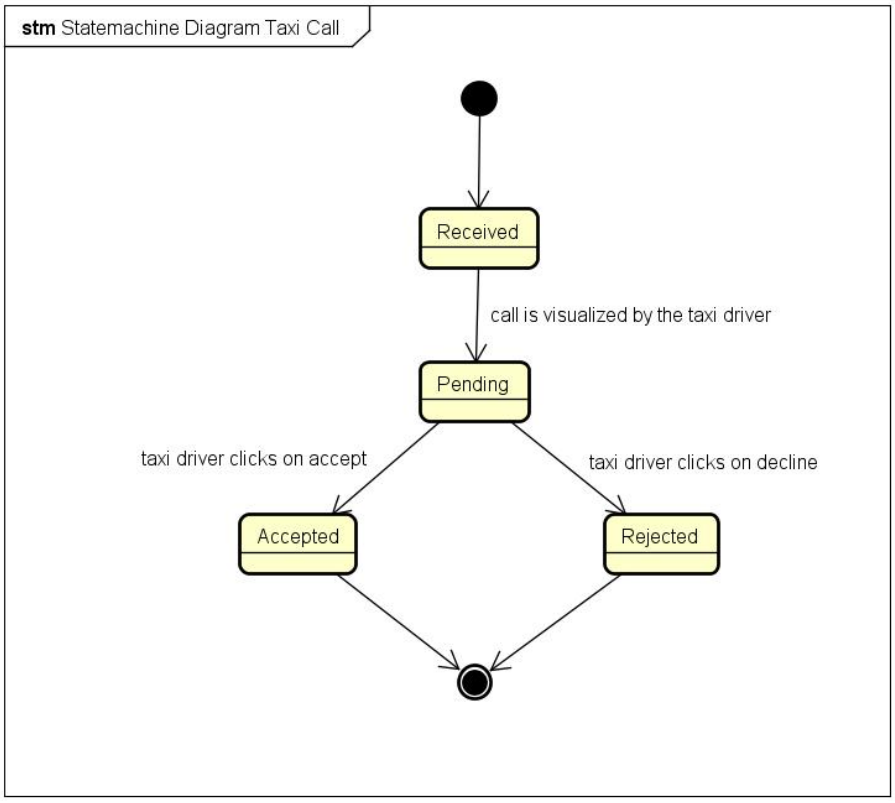
The following analysis will underline possible use cases of the application providing diagrams for each one.

3.5.1 Taxi Call

Name	Taxi Call
Actors	Customer, Taxi driver
Goal	[G1], [G4], [G5]
Entry condition	This use case starts when a customer decides to call a taxi
Event flow	1. Customer on the home page clicks on the button Taxi right now just me to start the calling process. 2. Customer fills in all mandatory fields. 3. Customer clicks on Hey, Taxi! button to confirm the request. 4. The application will verify the validity of the input data. In case of a positive response, the system will search for an available taxi in the nearby. Otherwise, a new form is sent to the customer. 5. When an available taxi driver takes care of the request, the call will be confirmed to the customer.
Output condition	The use case terminates when the customer successfully ends taxi calling process.
Exception	The customer and the taxi driver are immediately notified if the connection between their terminal and the central is lost

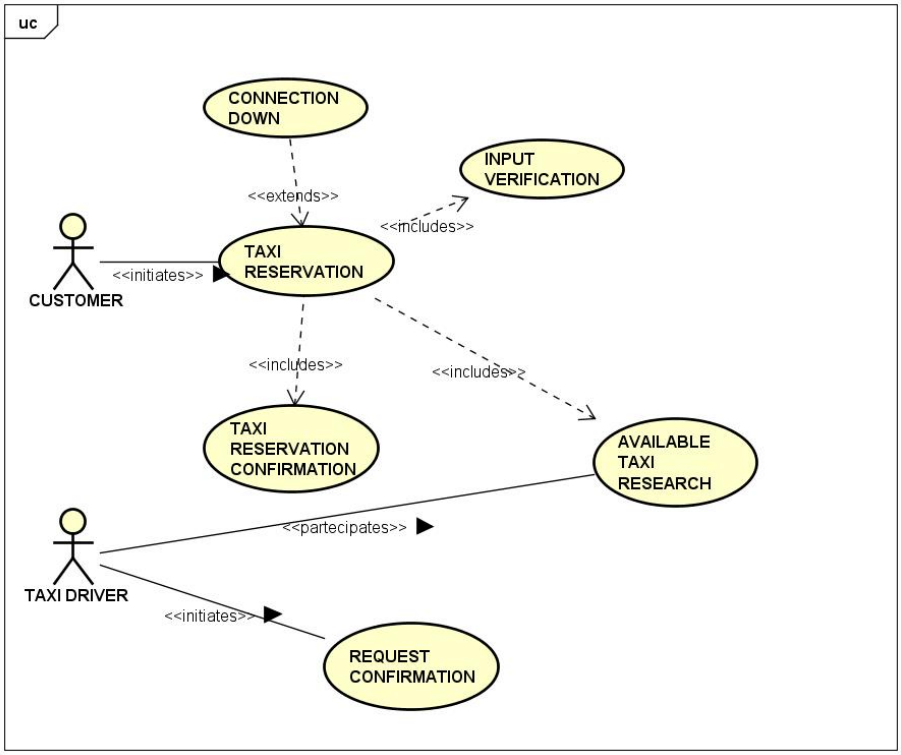

uc
GPS POSITION ACQUISITION
CONNECTION DOWN
<<extends>>
<<extends>>
CUSTOMER
<<initiates>>
TAXI CALL
<<includes>>
INPUT VERIFICATION
<<includes>>
<<includes>>
ARRIVING TAXI CONFIRMATION
AVAILABLE TAXI RESEARCH
<<partecipates>>
TAXI DRIVER
<<initiates>>
REQUEST CONFIRMATION

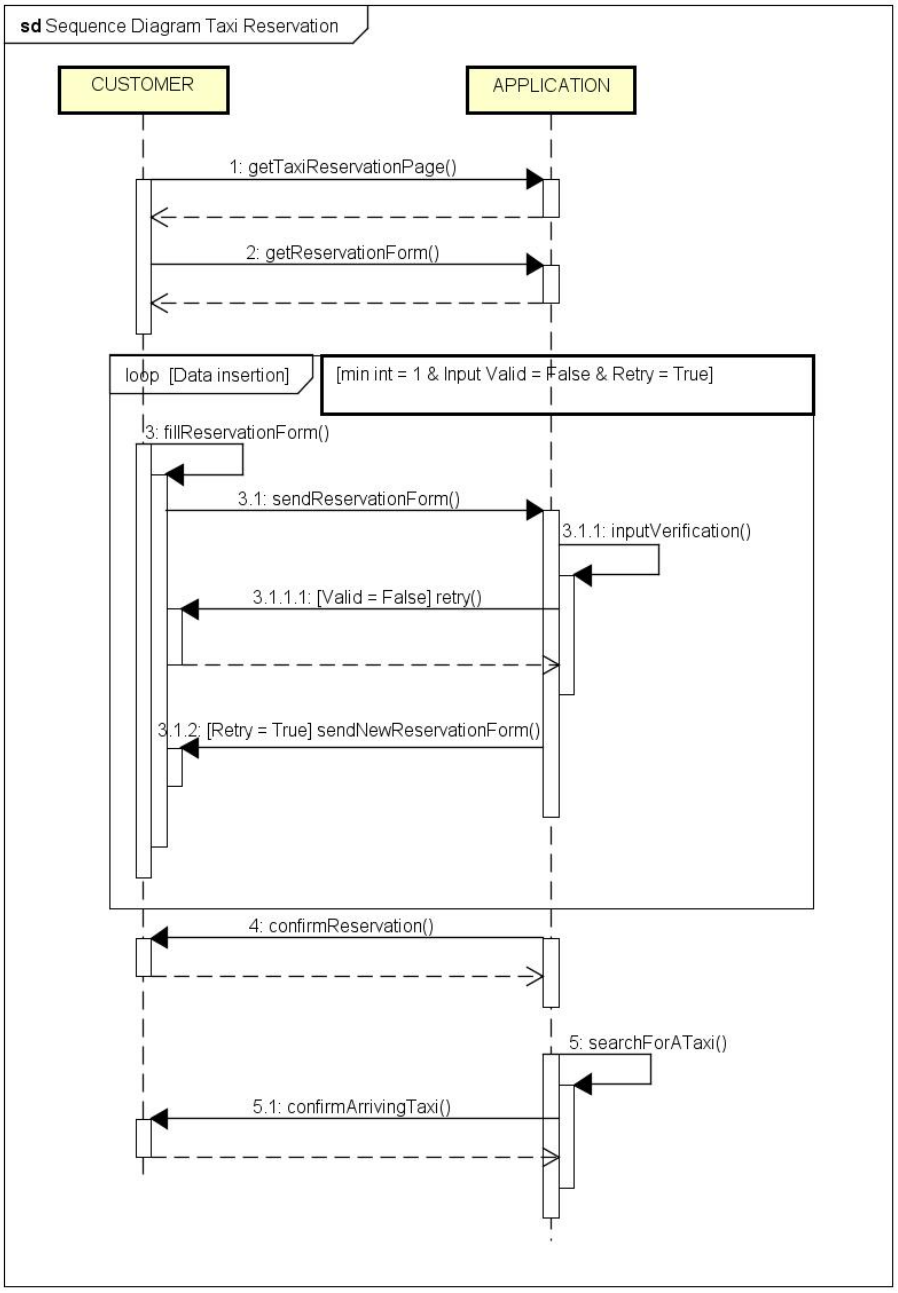

sd Sequence Diagram Taxi Call
CUSTOMER
APPLICATION
1: getTaxiCallPage()
2: getCallForm()
loop [Data insertion]
[min int = 1 & Input Valid = False & Retry = True]
3: fillCallForm()
3.1: sendCallForm()
3.1.1: inputVerification()
3.1.1.1: [Valid = False] retry()
3.1.2: [Retry = True] sendNewCallForm()
4: searchForATaxi()
4.1: confirmArrivingTaxi()

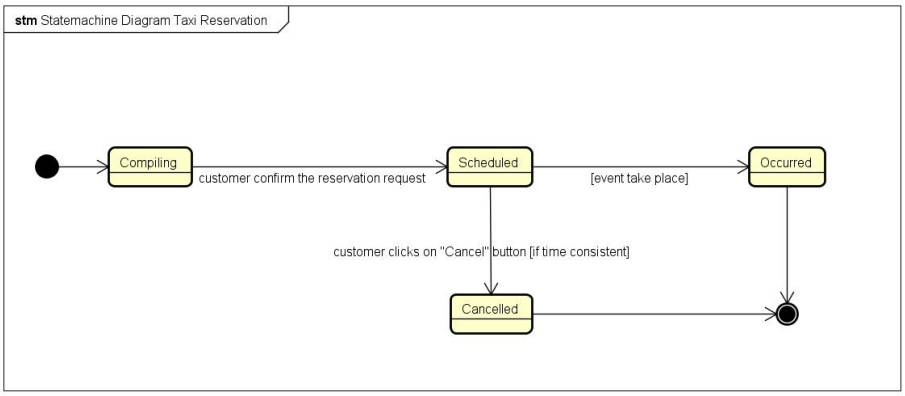

stm Statemachine Diagram Taxi Call
Received
call is visualized by the taxi driver
Pending
taxi driver clicks on accept
taxi driver clicks on decline
Accepted
Rejected

3.5.2 Taxi Reservation

Name	Taxi Reservation
Actors	Customer, Taxi driver
Goal	[G2], [G4], [G5]
Entry condition	This use case starts when a customer decides to reserve a taxi
Event flow	1. Customer on the home page clicks on the button Reserve to start the reserving process. 2. Customer fills in all mandatory fields. 3. Customer clicks on Reserve! button to confirm the request. 4. The application will verify the validity of the input data. In case of a positive response, the reservation will be confirmed to the customer and ten minutes before the fixed time the system will search for an available taxi in the nearby. Otherwise, a new form is sent to the customer. 5. When an available taxi driver takes care of the request, the call will be confirmed to the customer.
Output condition	The use case terminates when the customer successfully ends taxi reserving process.
Exception	The customer is immediately notified if the connection between his terminal and the central server is lost

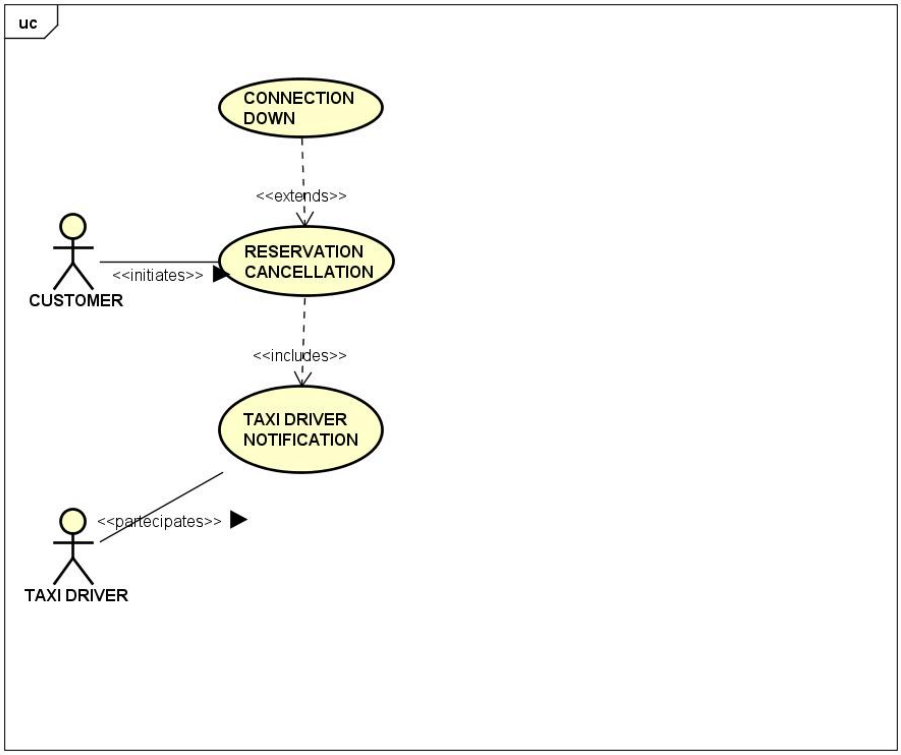

uc
CONNECTION DOWN
INPUT VERIFICATION
<<extends>>
<<includes>>
CUSTOMER
<<initiates>>
TAXI RESERVATION
<<includes>>
<<includes>>
TAXI RESERVATION CONFIRMATION
AVAILABLE TAXI RESEARCH
<<partecipates>>
TAXI DRIVER
<<initiates>>
REQUEST CONFIRMATION

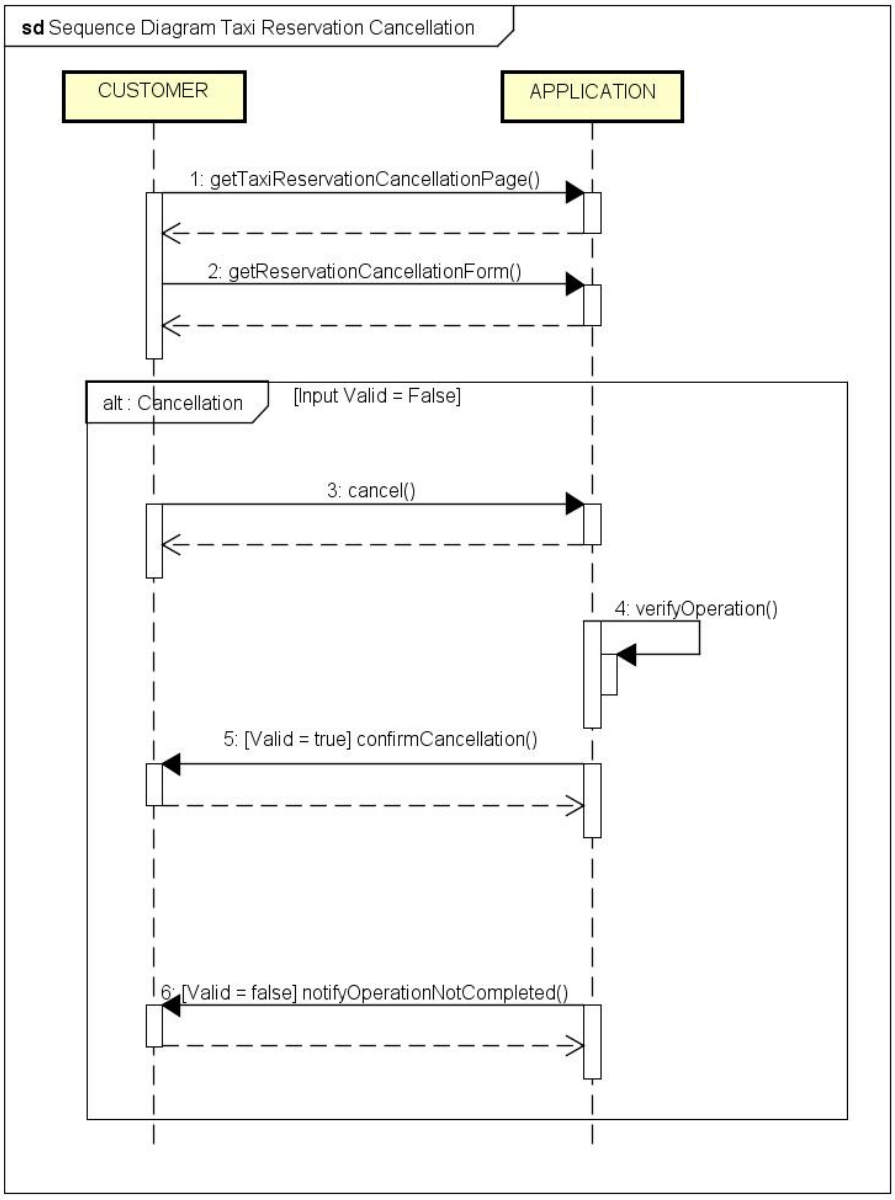

sd Sequence Diagram Taxi Reservation
CUSTOMER
APPLICATION
1: getTaxiReservationPage()
2: getReservationForm()
loop [Data insertion]
[min int = 1 & Input Valid = False & Retry = True]
3: fillReservationForm()
3.1: sendReservationForm()
3.1.1: inputVerification()
3.1.1.1: [Valid = False] retry()
3.1.2: [Retry = True] sendNewReservationForm()
4: confirmReservation()
5: searchForATaxi()
5.1: confirmArrivingTaxi()



3.5.3 Taxi Resevation Cancellation

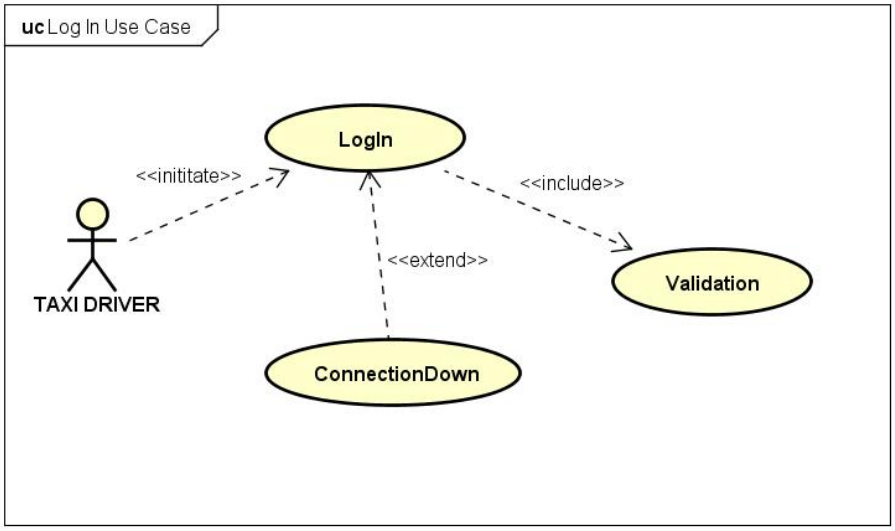
Name	Taxi Reservation Cancellation
Actors	Customer, Taxi driver
Goal	[G2]
Entry condition	This use case starts when a customer decides to cancel a reservation
Event flow	1. Customer on the home page clicks on the button Cancel reservation to start the cancelling process. 2. Customer selects the request in case and clicks on Cancel! button to confirm the request. 3. The application will verify the validity of the application. In case of a positive response, the reservation in case will be cancelled.
Output condition	The use case terminates when the customer successfully ends taxi cancelling process.
Exception	The customer is immediately notified if the connection between his terminal and the central server is lost

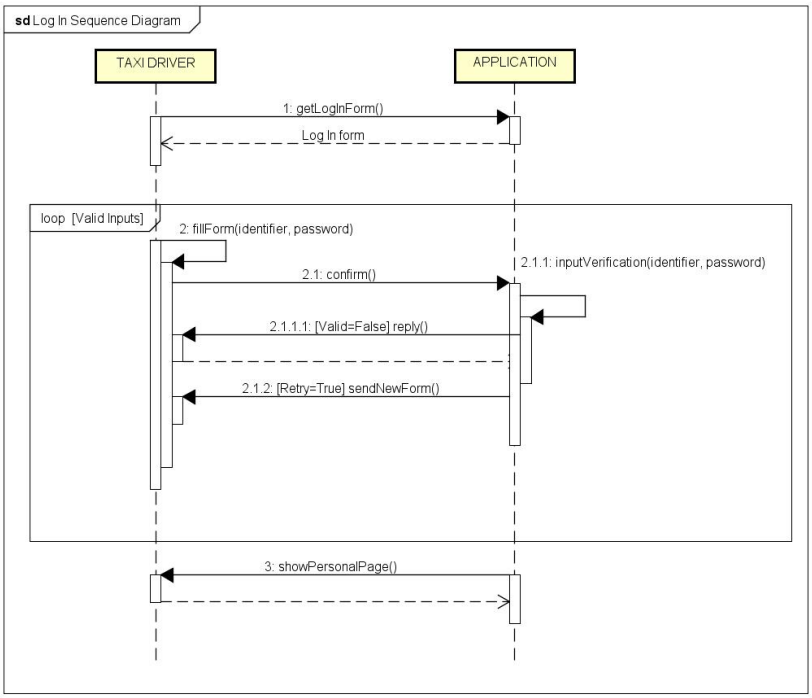

uc
CONNECTION DOWN
<<extends>>
RESERVATION CANCELLATION
<<initiates>>
CUSTOMER
<<includes>>
TAXI DRIVER NOTIFICATION
<<partecipates>>
TAXI DRIVER


sd Sequence Diagram Taxi Reservation Cancellation
CUSTOMER
APPLICATION
1: getTaxiReservationCancellationPage()
2: getReservationCancellationForm()
alt : Cancellation
[Input Valid = False]
3: cancel()
4: verifyOperation()
5: [Valid = true] confirmCancellation()
6: [Valid = false] notifyOperationNotCompleted()

3.5.4 Log In

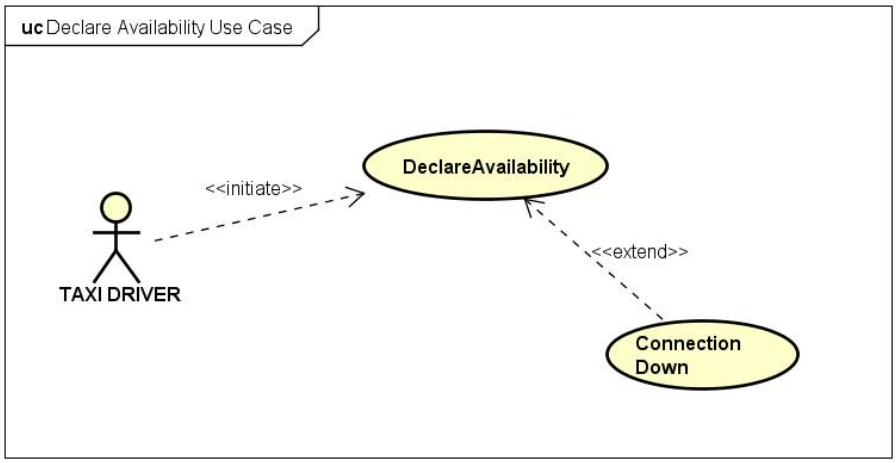
Name	Log In
Actors	Taxi driver
Goal	[G4]
Entry condition	This use case starts when a taxi driver, who is already registered in to the system, begins its shift.
Event flow	1. The taxi driver connects to the application. 2. The taxi driver fills the form entering his identifier and password.
Output condition	The application shows to the taxi driver his personal page.
Exception	If the connection is lost, the system immediately notifies the taxi driver In the case of the password or/and the identifier inserted by the user are wrong, the system shows an error message to the user.

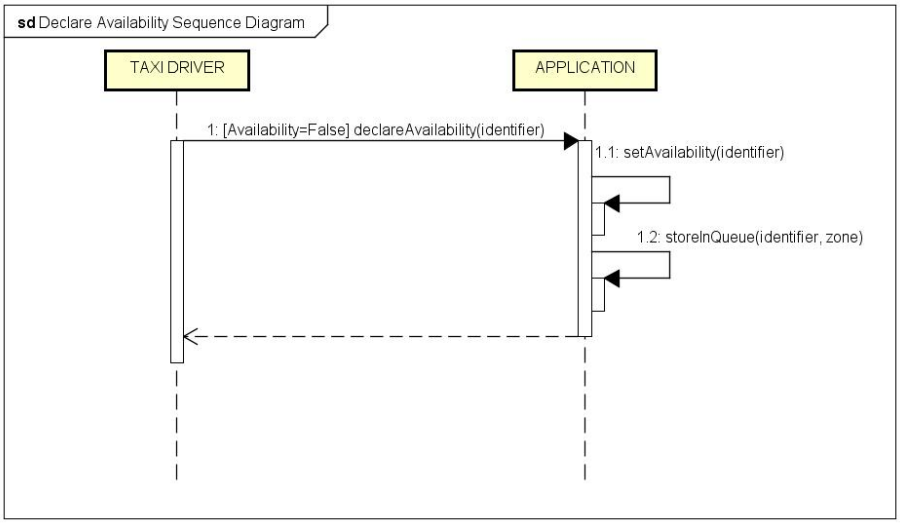


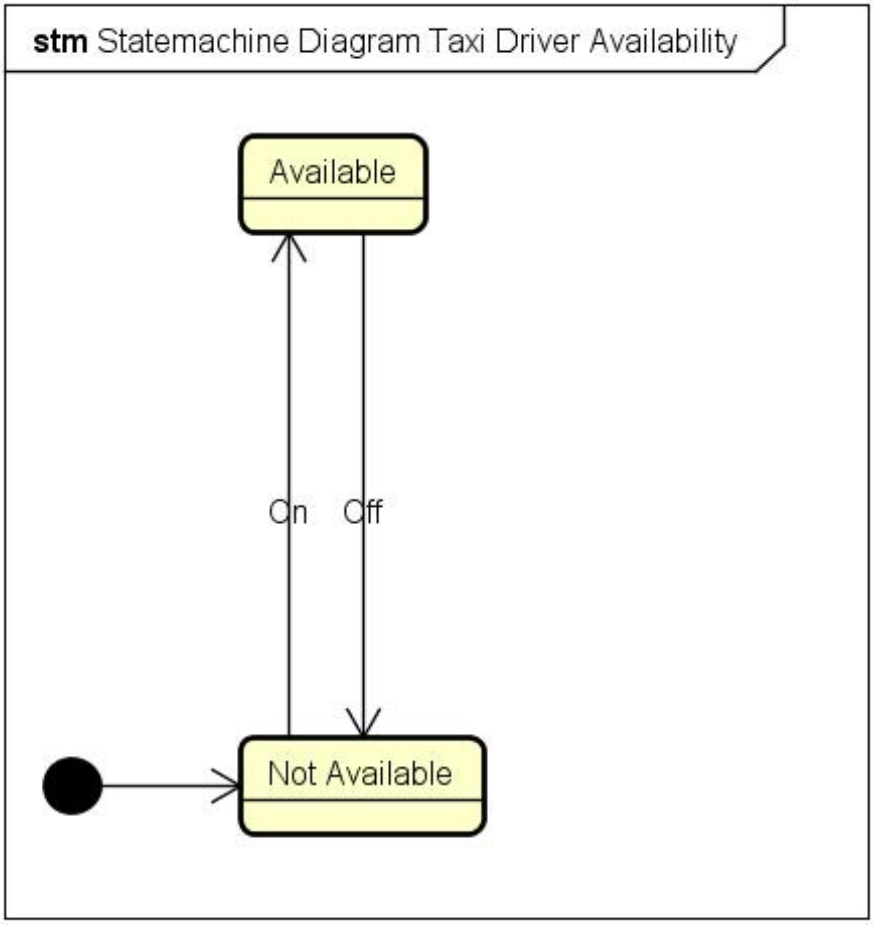


3.5.5 Declare Availability

Name	Declare Availability
Actors	Taxi driver
Goal	[G4]
Entry condition	This use case starts when a taxi driver has completed the login procedure or when he has completed a previous request.
Event flow	1. The taxi driver selects from his terminal the Declare Availability option. 2. The application gets the taxi identifier and automatically associates the taxi to a certain zone based on the GPS information. 3. The application stored the identifier in the queue of taxis in the corresponding zone.
Output condition	The taxi driver is now available to collect new requests from a certain zone.
Exception	The taxi driver is immediately notified whether the connection between his terminal and the central is lost

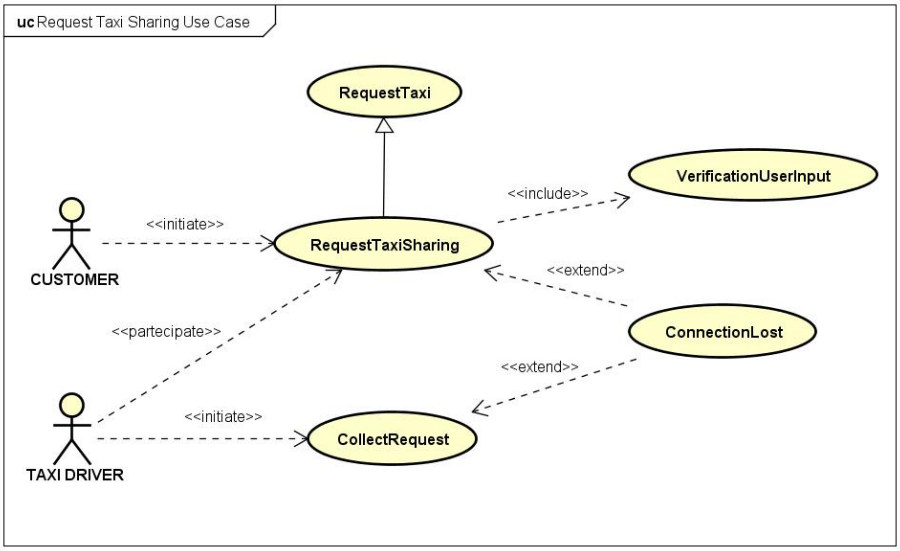

uc Declare Availability Use Case
DeclareAvailability
<<initiate>>
<<extend>>
TAXI DRIVER
Connection Down

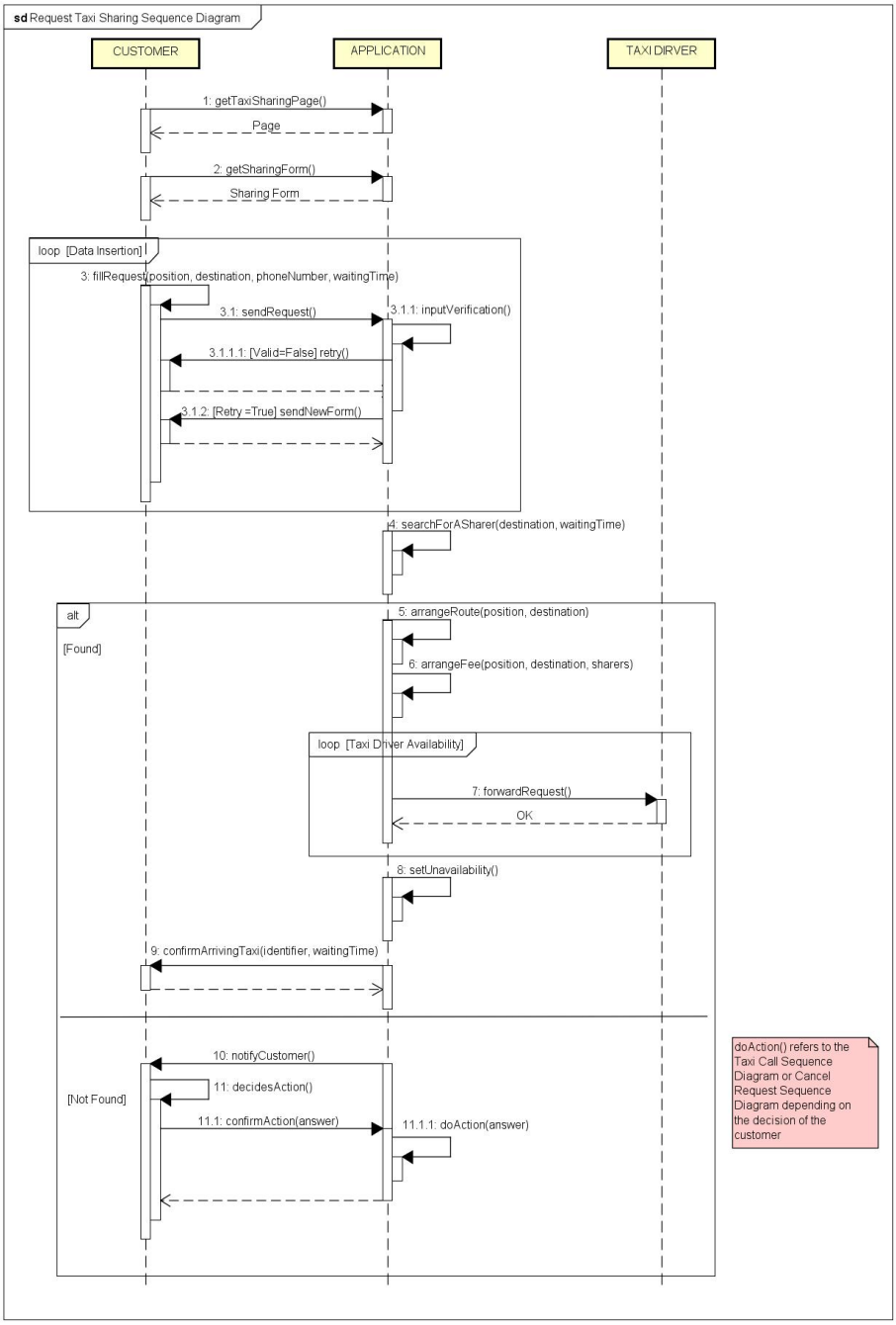

sd Declare Availability Sequence Diagram
TAXI DRIVER
APPLICATION
1: [Availability=False] declareAvailability(identifier)
1.1: setAvailability(identifier)
1.2: storeInQueue(identifier, zone)


stm Statemachine Diagram Taxi Driver Availability
Available
On
Off
Not Available

3.5.6 Taxi Sharing Call

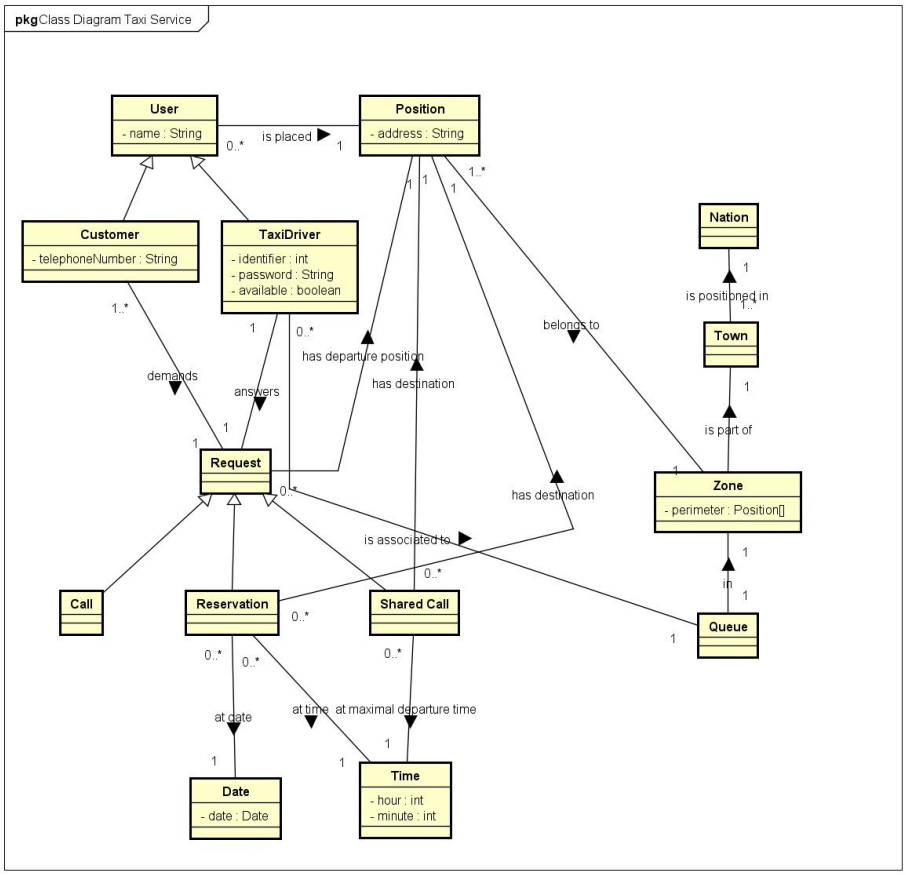
Name	Taxi Sharing Call
Actors	Customer, Taxi driver
Goal	[G3], [G4], [G5]
Entry condition	This use case starts when a customer decides to use the taxi sharing option.
Event flow	1. The customer on the home page clicks on the Taxi Sharing option to begin the process. 2. The customer fills at least all the mandatory fields specifying the destination and the time he is willing to wait to take the ride, while the application gets the actual position through the mobile GPS system. 3. The customer clicks on the Confirm button. 4. The application computes data looking for anyone is willing to share the ride from the same zone going in the same direction. 5. The application arranges the route for the taxi driver and defines the fee for all the persons sharing the taxi. 6. The application informs the taxi driver about the request. 7. The taxi driver confirms to the system that he is going to collect the request. 8. The system set as unavailable the state of the considered taxi. 9. The system informs all the interested passengers that the request has been accepted.
Output condition	The use case terminates when there is an appointed to collect the request.
Exception	The customer and the taxi driver are immediately notified if the connection between their terminal and the central is lost. In case of one or more mandatory field is missed or not valid the system alert the user of the problem occurred and the application goes back to point number 2 of the event flow. In case of the system has not received any other similar request within the stipulated time the customer receives a notice. At this point the customer can either decide to take the ride on his own or to delete the request. If the selected taxi doesn't confirm, the system forward the request to the second of the same queue and moves at the same time the first taxi to the last position of the queue.


uc Request Taxi Sharing Use Case
RequestTaxi
VerificationUserInput
<<include>>
<<initiate>>
RequestTaxiSharing
CUSTOMER
<<extend>>
<<partecipate>>
ConnectionLost
<<extend>>
<<initiate>>
CollectRequest
TAXI DRIVER


sd Request Taxi Sharing Sequence Diagram
CUSTOMER
APPLICATION
TAXI DIRVER
1: getTaxiSharingPage()
Page
2: getSharingForm()
Sharing Form
loop [Data Insertion]
3: fillRequest(position, destination, phoneNumber, waitingTime)
3.1: sendRequest()
3.1.1: inputVerification()
3.1.1.1: [Valid=False] retry()
3.1.2: [Retry =True] sendNewForm()
4: searchForASharer(destination, waitingTime)
alt
[Found]
5: arrangeRoute(position, destination)
6: arrangeFee(position, destination, sharers)
loop [Taxi Driver Availability]
7: forwardRequest()
OK
8: setUnavailability()
9: confirmArrivingTaxi(identifier, waitingTime)
[Not Found]
10: notifyCustomer()
11: decidesAction()
11.1: confirmAction(answer)
11.1.1: doAction(answer)
doAction() refers to the Taxi Call Sequence Diagram or Cancel Request Sequence Diagram depending on the decision of the customer

3.6 Class Diagram

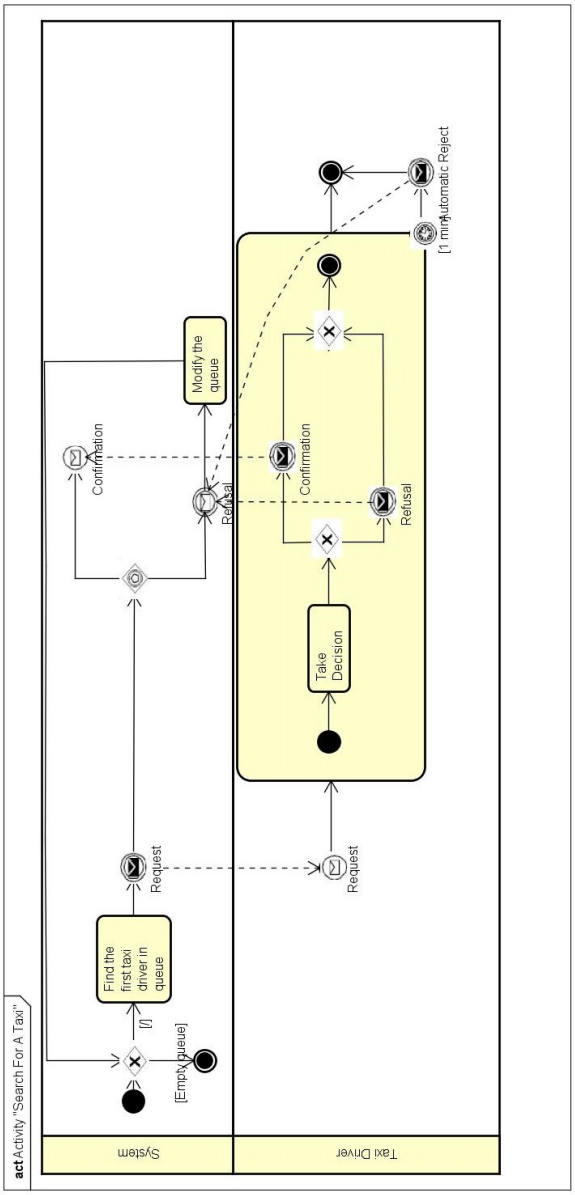
The following diagram represent a possible sketch of the class isolation of the application.



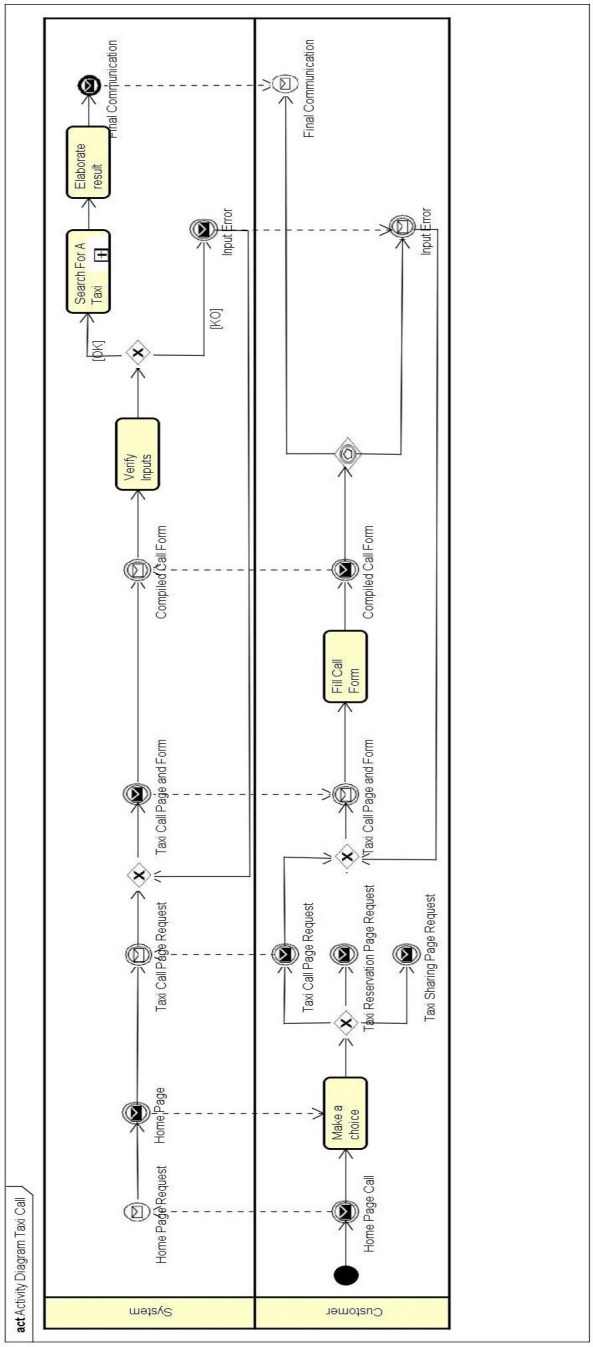
3.7 Activity Diagrams

The following Activity Diagrams would describe the process flow of the taxi call, taxi reservation and taxi sharing requests.

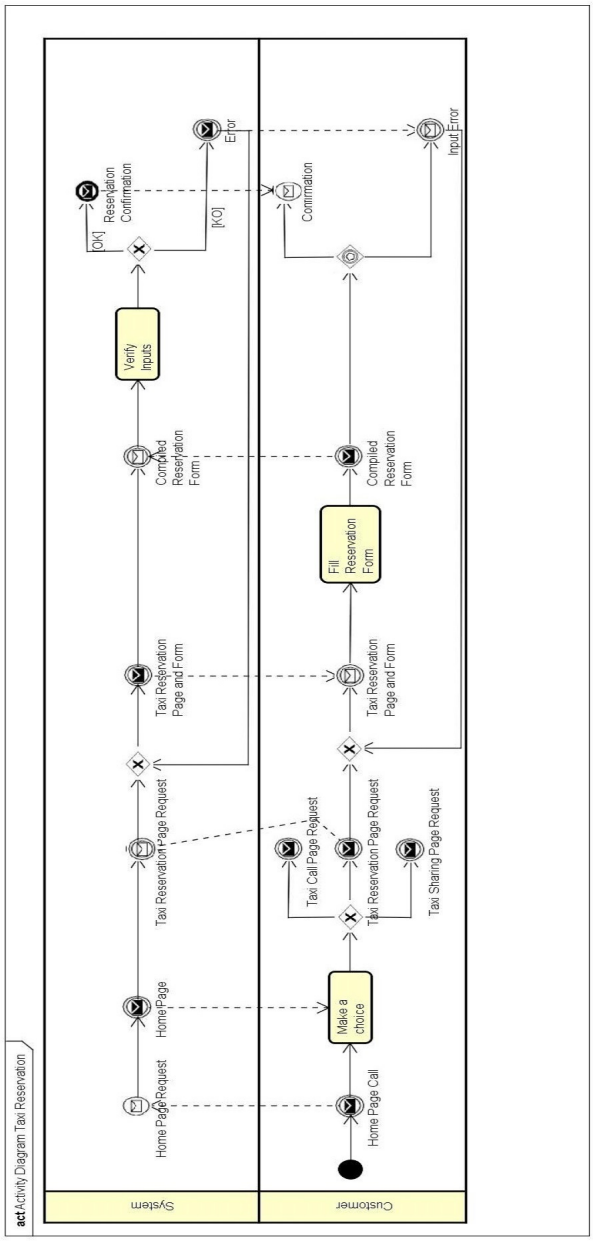
3.7.1 Search For A Taxi



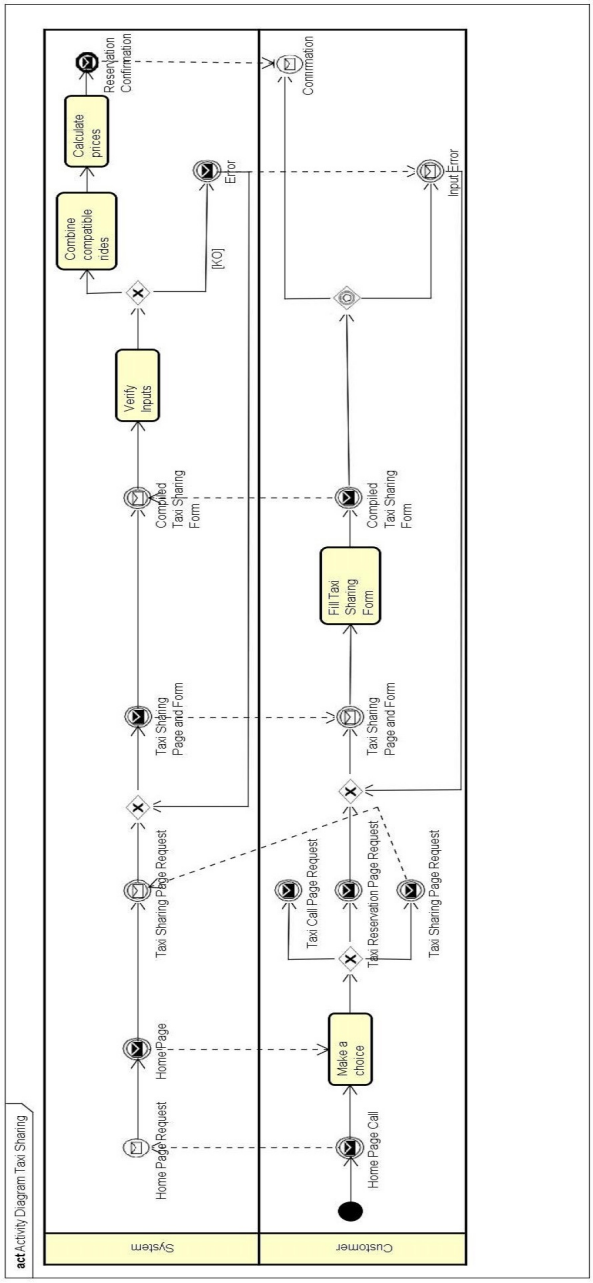
3.7.2 Taxi Regular Call



3.7.3 Taxi Reservation



3.7.4 Taxi Sharing



4 Appendix

4.1 Alloy

4.1.1 Alloy Code

The following alloy model is created from the class diagram. The code provided here is divided in four sections: signatures, facts, asserts and predicates.

```
//SIGNATURES

sig Strings{}

sig Integer{}

sig Boolean{
        value: one Int,
}
{value=1 or value=0}

abstract sig User{
        name: one Strings,
        hasPosition: one Position,
}

sig Customer extends User{
        phoneNumber: one Strings,
        demands: one Request,
}

sig TaxiDriver extends User{
        identifier: one Integer,
        password: one Strings,
        available: one Boolean,
        answer: set Request,
}

sig Nation{}

sig Town{
        locatedIn: one Nation,
}

sig Zone{
    perimeter: set Position,
        partOf: one Town,
}
//{#perimeter > 4}

sig Queue{
        positionedIn: one Zone,
        has: set TaxiDriver,
}

sig Position{
        address: one Strings,
        belongsTo: one Zone,
}

abstract sig Request{
        hasDeparture: one Position
}

sig Call extends Request{}

sig Reservation extends Request{
```

```
        hasDestination: one Position,
        hasDate: one Date,
        hasTime: one Time
}

sig SharedCall extends Request{
        isAssociated: one Position,
        hasWaitingTime: one Time
}

sig Date{
        year: one Integer,
        month: one Integer,
        day: one Integer
}

sig Time{
        hour: one Integer,
        minute: one Integer,
        date: one Date
}



//FACTS

fact OneZonePerQueue{
        //Each zone has exactly one queue
        all z: Zone | (one q: Queue | q.positionedIn = z)
}

fact OneCustomerPerRequest{
        //Each request is demanded by exactly one customer
        all r: Request | (one c: Customer | r in c.demands)
}

fact LoneCallPerTaxiDriver{
        //If a taxi driver answers to a regular call, he will answer only to that single request
        all t:TaxiDriver | (all c: Call | (c in t.answer) implies #t.answer=1)
}

fact LoneReservationTaxiDriver{
        //If a taxi driver answers to a reservation, he will answer only to that single request
        all t:TaxiDriver | (all r: Reservation | (r in t.answer) implies #t.answer=1)
}

fact MoreSharedCallPerTaxiDriver{
        //A taxi driver can answer to more shared calls
        all t:TaxiDriver | (all sc:SharedCall | (sc in t.answer) implies #t.answer≥1)
}

fact LoneTaxiDriverPerRequest{
        //Each request is answered by at most one taxi driver
        no disj t1,t2:TaxiDriver | (some r:Request |( r in t1.answer and r in t2.answer))
}

fact OneZonePerPosition{
        //Each position belongs to exactly one zone
        no disj z1,z2: Zone | (some p: Position |( p in z1.perimeter and p in z2.perimeter))
}

fact OneRequestForOneCustomer{
        //Each request is made by exactly one customer
        no disj c1,c2:Customer | (some r: Request | (r in c1.demands and r in c2.demands))
}

fact SameZoneTaxiDriverRequest{
        //The zone of the
```

```
        // position of the request is the same of the zone of taxi driver's position
    all t:TaxiDriver | (all r:Request | ((r in t.answer) implies
          (t.hasPosition.belongsTo = r.hasDeparture.belongsTo)))
}

fact LoneQueuePerTaxiDriver{
        //Each taxi driver has at most a queue
        all t:TaxiDriver | (lone q:Queue | t in q.has)
}

fact TaxiDriverInQueueIffAvailable{
        //Each taxi driver is in queue iff is available
        all t:TaxiDriver | (one q:Queue | t in q.has iff t.available.value = 1)
}

fact TaxiDriverNotAvailableCall{
        //If a taxi driver answers to a regular call or to a reservation is not available
        all t:TaxiDriver | (all c: Call | ( all r: Reservation | ((c in t.answer or r in t.answer)
        iff t.available.value = 0)) )
}

fact SameZoneTaxiDriverQueue{
        //The zone of the queue
        //is exactly the zone of the taxi driver's position
        all t:TaxiDriver | (all q:Queue | (t in q.has) implies
        (q.positionedIn = t.hasPosition.belongsTo ))
}

fact noDuplicatePhoneNumber{
        //There is no duplicate phone number
        no disj c1,c2:Customer | (c1.phoneNumber = c2.phoneNumber)
}

fact noDuplicateIdentifier{
        //There is no duplicate identifier
        no disj t1,t2:TaxiDriver | (t1.identifier= t2.identifier)
}

fact noDuplicateAddress{
        //There is no duplicate address
        no disj p1,p2:Position | (p1.address= p2.address)
}

fact noDuplicateDate{
        //There is no duplicate date
        no disj d1,d2:Date | (d1.year = d2.year and d1.month = d2.month and
        d1.day = d2.day)
}

fact noDuplicateTime{
        //There is no duplicate time
        no disj t1,t2:Time | (t1.date = t2.date and t1.hour= t2.hour and t1.minute =
t2.minute)
}

fact sameTaxiDriverPerSharedCall{
        //If some requests are coupled (and thus they are shared calls), they are coupled in the
        //same taxi driver
        all disj c1,c2:Customer | (lone t:TaxiDriver | (c1.demands in t.answer)
and  (c2.demands in t.answer))
}

fact sameStartingZonePerSharedCall{
        //If some shared calls are answered by the same taxi driver,
        //they have the same departure zone
        all disj sc1, sc2: SharedCall | (all t: TaxiDriver | (sc1 in t.answer and sc2 in t.answer)
        implies (sc1.hasDeparture.belongsTo = sc2.hasDeparture.belongsTo))
}
```

```
//ASSERTIONS

assert oneZonePerPosition{
        //Each position belongs to exactly one zone
        all z1, z2: Zone | (all p: Position | (z1 in p.belongsTo and z2 in p.belongsTo)
        implies (z1 = z2) )
}
check oneZonePerPosition

assert sameZoneTaxiDriverRequest{
        //There is no taxi driver taking care of a request coming from another zone
        no t:TaxiDriver | (some r:Request | r in t.answer and (t.hasPosition.belongsTo ≠
         r.hasDeparture.belongsTo))
}
check sameZoneTaxiDriverRequest

assert twoTaxisForTheSameRequest{
        //There is no way to have two taxi drivers answering to the same request
        no disj t1,t2:TaxiDriver | (some r:Request | (t1.answer = r and t2.answer =
r))
}
check twoTaxisForTheSameRequest

assert twoQueueForTheSameTaxiDriver{
        //There is no way to have two taxi drivers answering to the same request
        no disj q1,q2:Queue | (some t:TaxiDriver | (t in q1.has and t in q2.has))
}
check twoQueueForTheSameTaxiDriver

assert sameZoneQueueTaxiDriver{
        //There is no queue containing a taxi driver corresponding to another zone
        no q:Queue | (some t:TaxiDriver | t in q.has and (q.positionedIn ≠
        t.hasPosition.belongsTo))
}
check sameZoneQueueTaxiDriver

assert validDate{
        //There is no invalid date
        no d:Date | (#d.year > 1 or #d.month > 1 or #d.day > 1)
}
check validDate

assert validTime{
        //There is no invalid time
        no t:Time | (#t.hour>1 or #t.minute>1)
}
check validTime

assert noDuplicatePhoneNumber{
        //If two customers have the same phone number then the customers are the same
        all c1,c2:Customer | (c1.phoneNumber = c2.phoneNumber implies c1 = c2)
}
check noDuplicatePhoneNumber

assert sameTaxiDriverPerSharedCall{
        //If there are two customers demanding a shared call from two different zone or
        //to two different zone then the taxi driver can take care of only one of the
        //two requests
        no disj sc1,sc2:SharedCall | some t:TaxiDriver | (sc1 in t.answer and sc2 in t.answer and
        sc1.hasDeparture.belongsTo ≠ sc2.hasDeparture.belongsTo)
}
check sameTaxiDriverPerSharedCall
```

```
//PREDICATES

pred show1(){
        #Nation = 1
        #Town = 1
        #Zone = 2
        #Queue >0
        #Position = 4
        #Boolean = 2
        #Date = 4
        #Time = 6
        #Customer=2
        #TaxiDriver = 3
        #Call = 1
        #Reservation = 1
        #answer = 2
}
run show1 for 10


pred show2(){
        #Nation = 1
        #Town = 1
        #Zone = 2
        #Queue >0
        #Position = 4
        #Boolean = 2
        #Date = 4
        #Time = 6
        #Customer=2
        #TaxiDriver = 2
        #SharedCall = 2
        #answer = 2
}
run show2 for 10
```

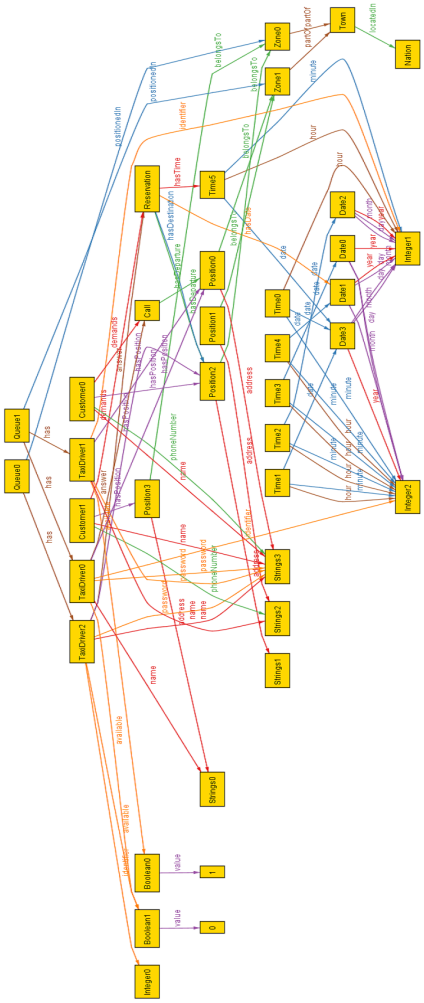
4.1.2 Alloy Analysis Summary

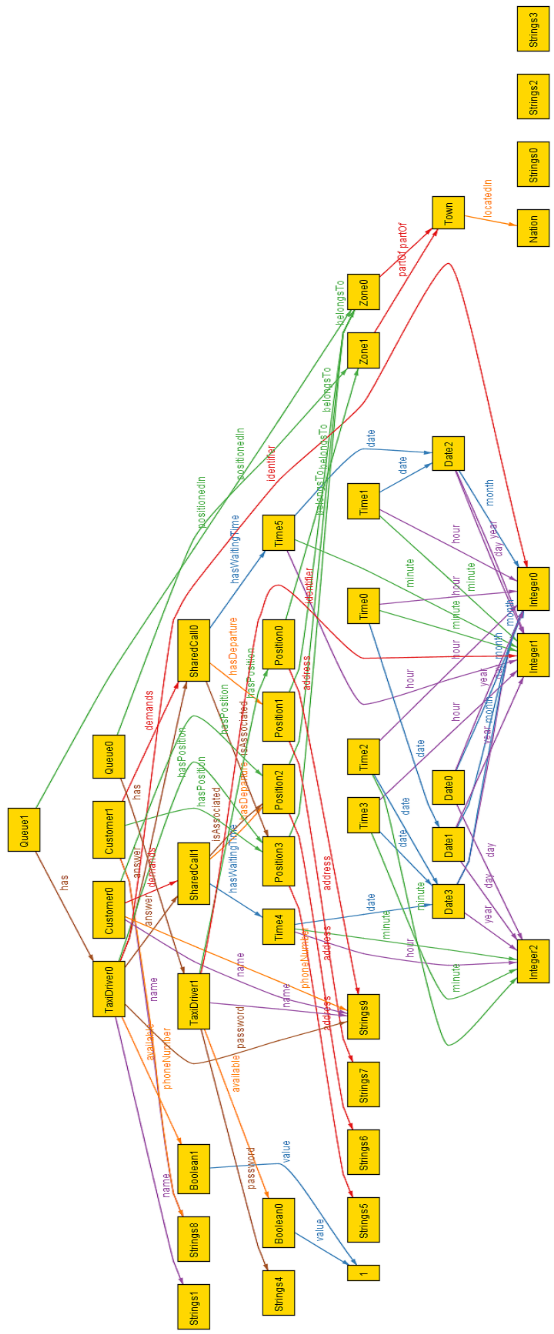
This is the result of the analysis of the given model

11 commands were executed. The results are:
- #1: No counterexample found. oneZonePerPosition may be valid.
- #2: No counterexample found. sameZoneTaxiDriverRequest may be valid.
- #3: No counterexample found. twoTaxisForTheSameRequest may be valid.
- #4: No counterexample found. twoQueueForTheSameTaxiDriver may be valid.
- #5: No counterexample found. sameZoneQueueTaxiDriver may be valid.
- #6: No counterexample found. validDate may be valid.
- #7: No counterexample found. validTime may be valid.
- #8: No counterexample found. noDuplicatePhoneNumber may be valid.
- #9: No counterexample found. sameTaxiDriverPerSharedCall may be valid.
- #10: **Instance found.** show1 is consistent.
- #11: **Instance found.** show2 is consistent.

4.1.3 Alloy Generated Worlds

Here are presented the worlds generated by the alloy analysis software using the given show methods.




Queue1
Queue0
Customer1
Customer0
TaxiDriver0
TaxiDriver1
SharedCall0
SharedCall1
Position0
Position1
Position2
Position3
Time0
Time1
Time2
Time3
Time4
Time5
Zone0
Zone1
Town
Nation
Date0
Date1
Date2
Date3
Integer0
Integer1
Integer2
Strings0
Strings1
Strings2
Strings3
Strings4
Strings5
Strings6
Strings7
Strings8
Strings9
Boolean0
Boolean1
1

4.2 Software Employed

The following is a list of the softwares used to redact the present document:

- *MiKTeX 2.9*: main LaTeX environment.
- *TeXstudio* : LaTeX editor used to redact the document.
- *BalsamiqMockups 3*: to create interface mockups.
- *Astah Professional*: to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Machine Diagrams
- *Alloy Analyzer*: to create and analyze a valid Alloy model.

4.3 Working Hours

Fabio Catania: ca. 30 hours.

Matteo Di Napoli: ca. 30 hours.

Nicola Di Nardo: ca. 30 hours.