Requirements Analysis and Specification Document

Alberto Mario Pirovano Alessandro Vetere

November 7, 2015

Contents

1 Introduction

What follows is an introduction to *myTaxiService* RASD document.

1.1 Purpose

The purpose of this document is to serve as a guide to software designers, developers and testers who are responsible for the engineering of the *myTaxiService* project.

Through UML use cases and diagrams, it will show the interaction between the different agents involved in the application, to better explain the environment in which the software will work when installed in the real world: this should give the engineers all the necessary informations to design, develop and test the software. In addition, it is an important tool for stakeholders to check whether the product matches their initial demands, and to serve as a further contractual basis.

1.2 Scope

This document provides an overall description of *myTaxiService* software system, showing its functional and non-functional requirements, constraints and external interfaces.

It consists of natural language descriptions, UML diagrams and Alloy models of the different features given.

1.2.1 Actors

Below are listed the two main actors that will interact with the application once deployed:

- **Taxi driver:** Owner of a vehicle who is given the permission to provide the service.
- **Non registered passenger:** A person that needs to move from a position to another one among the city and wants to use *myTaxiService* in order to do so, but has not registered yet to the service.
- **Registered passenger:** A formerly non registered passenger that has registered to *myTaxiService*.

1.2.2 Goals

This new service pretends to achieve various goals, the main ones being:

- **G1:** A non registered passenger can only register once to the service.
- **G2:** A registered passenger is registered to the service.

- **G3:** A registered passenger can login to the service only when not logged in.
- **G4:** A registered passenger can logout from the service only when logged in.
- **G5:** A registered passenger can request a taxi ride only when logged to the service.
- **G6:** A registered passenger can reserve a taxi ride only when logged to the service.
- **G7:** A registered passenger can view his taxi requests when logged to the service.
- **G8:** A registered passenger can cancel a taxi request only if he is viewing it.
- **G9:** A registered passenger can view his taxi reservations when logged to the service.
- **G10:** A registered passenger can cancel a taxi reservation only if he is viewing it.
- **G11:** A registered passenger can handle his profile only when logged to the service.
- **G12:** A taxi driver is registered to the service.
- **G13:** A taxi driver can login to the service only when not logged in.
- **G14:** A taxi driver can logout to the service only when logged in.
- **G15:** A taxi driver can accept to give a ride to a registered user that requested one.
- **G16:** A taxi driver can deny to give a ride to a user that requested one.
- **G17:** A taxi driver can notify the end of a ride.
- **G18:** A taxi driver that denies availability for a ride is put last in his current zone waiting queue.
- **G19:** A registered passenger can only take a ride from a taxi driver who is first in his current zone waiting queue.
- **G20:** A taxi driver belongs to at maximum one waiting queue at a time.
- **G21:** Both a non registered passenger and a registered passenger can use either a web application or a mobile application to access the service.
- **G22:** Further services can be built on the top of the existing one through a set of given APIs.
- **G23:** For a registered passenger that has requested a taxi ride, the system allocates a taxi driver to pick him up 10 minutes before the meeting time.

1.3 Definitions, acronyms, and abbreviations

Here are listed some of the necessary definitions, acronyms and abbreviations to make better understandable the content of the document:

1.3.1 Acronyms

As commonly used, some acronyms are included in the document and here explained:

- **IEEE:** Institute of Electrical and Electronics Engineers.
- **SRS:** Software Requirements Specification, document based on IEEE 830.
- **RASD:** Requirements Analysis and Specification Document, also known as SRS.
- **UML:** Unified Modelling Language.
- **Wi-Fi:** Wireless Fidelity, technology for local area wireless computer networking technology based on IEEE 802.11 standard.
- **WiMAX:** Worldwide Interoperability for Microwave Access, technology based on IEEE 802.16 standard.
- **API:** Application Programming Interface, a set of public accessible functions, protocols and tools provided by a specific application for building software applications.
- **GPS:** Global Positioning System.
- **DBMS:** DataBase Management System.
- **SDK:** Software Development Toolkit.
- **URI:** Uniform Resource Identifier, a string of characters used to identify the name of a resource.
- **URL:** Uniform Resource Locator, a specific type of URI that reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
- **HTTP:** The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems; HTTP is the foundation of data communication for the World Wide Web.
- **HTTPS:** Communication protocol over HTTP within a connection encrypted by Transport Layer Security.
- **GUI:** Graphical User Interface.
- **LAN:** Local Area Network.
- **MAN:** Metropolitan Area Network.

1.3.2 Definitions

What follows is a list of definitions of some terms used in the document:

- **Ethernet:** Family of computer networking technologies for LANs and MANs, first standardized in 1983 as IEEE 802.3.
- **Bluetooth:** Wireless technology standard for exchanging data over short distances, once based on IEEE 802.15.1 standard.
- **Internet:** The Internet is the global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) to link billions of devices worldwide.
- **Service:** The functionalities offered by *myTaxiService* via software applications to passengers and taxi drivers.
- **System:** The whole hardware and software parts that together deliver *myTaxiService* as a service to passengers and taxi drivers.
- **Server:** Also (a bit improperly) referred as **Back-End**, the centralized part of the system installed in a dedicated facility, that communicates with clients via a network.
- **Client:** A part of the system that is not the server, and communicates with the server via network to access a desired service.
- **Smartphone:** A mobile phone with an advanced mobile operating systems, which combines features of a personal computer operating system with other features useful for mobile or handheld use.
- **Application:** A computer program (i.e. a piece of software) that performs a group of coordinated functions, tasks, or activities for the benefit of the user.
- **Smartphone Application:** Also referred as **mobile application**, is an application designed to run on smartphones.
- **World Wide Web:** Also referred as **web**, is an information space where documents and other web resources are identified by URLs, interlinked by hypertext links, and can be accessed via the Internet.
- **Web Browser:** Software application for retrieving, presenting, and traversing information resources on the World Wide Web.
- **Web Page:** A web document that is suitable for the World Wide Web and the web browser.
- **Web Server:** An information technology that processes requests via HTTP and stores, processes and deliver web pages to clients.

- **Web Application:** A client-server software application in which the client (or user interface) runs in a web browser.
- **Controller:** A piece of software that resides on the server and act as an arbiter between incoming data from the Internet and the application data.

1.3.3 Abbreviations

For the sake of brevity, some abbreviations are adopted:

- **RP:** Registered passenger.
- **TD:** Taxi driver.
- **PTRS:** Pending Taxi Ride Set.
- **TRQ:** Taxi Request.
- **TRS:** Taxi Reservation.

1.4 Overview

This document is structured in 3 main parts:

- **Section 1:** Overview of the RASD document. Specifically provides both a description of high-level software functionalities, and a set of information about the organization of the document.
- **Section 2:** Provides a high-level description of the software requirements, not describing specifically the main aspect of them, but only providing a background of those requirements. The main purpose of this section is make the requirements easy to understand.
- **Section 3:** Shows all of software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test the system requirements.

1.5 References

This section contains references to external sources that have been taken into account while writing this document.

- Specification Document: Software Engineering 2 Project, AA 2015-2016, Assignment 1 and 2
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- Applied Software Project Management Andrew Stellman & Jennifer Greene
- Wikipedia, the free encyclopedia

2 Overall Description

The aim of *myTaxiService* is to improve taxi service usage and management in a large city by simplifying the access of passengers to the service and optimizing the management of taxi queue.

Overall, *myTaxiService* will lead to several benefits for taxi drivers, passengers and the government of the city.

2.1 Product Perspective

The software *myTaxiService* is a completely new product, not based on previous ones.

However, it relies on location data received via Internet from each taxi driver smartphone application: all the involved smartphones already have a GPS antenna installed inside, that communicates their position to the service.

Being a partially distributed application, *myTaxiService* requires a fully operative Internet connection in order to work properly, both on server and client side: no service is intended to be provided offline.

This software provides two separate end user interfaces accessible via Internet, and a separate administator interface that is only accessible via a LAN network.

All the data generated by this software are stored in a database, accordingly to current normatives and laws about privacy and personal data management.

In addition, several APIs are provided in order to permit further improvements and expansions of the software: in this way additional services like taxi sharing could be built on the top of the existing ones.

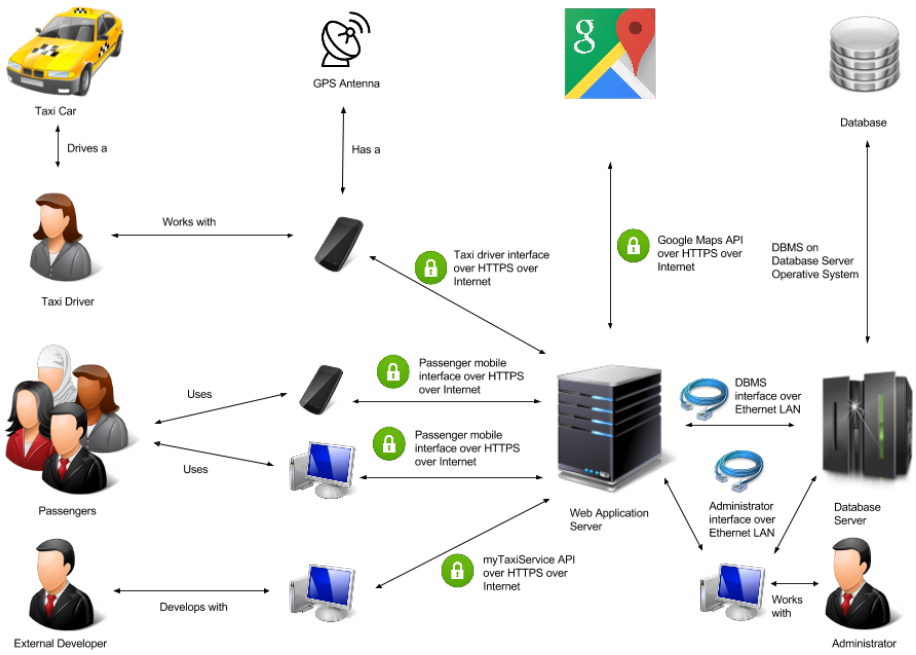

Figure 1: General Diagram

The high-level diagram proposed above is intended to be just a draft of the system to be and to be a general indicator for each type of professionalism that will use this document.

Therefore it aims to provide a high-level overview of the architecture of the software is going to be developed.

In particular, in this architecture are involved four classes of users; Taxi drivers, Passengers, registered or not, external developers and server administrators.

The first three are allowed to use *myTaxiService* through smartphone or personal computer, remotely connected to the Web Application Server using HTTPS Internet navigation protocol.

The Web Application Server, as well as providing the services requested by users and ensuring programmative APIs to external developers, in turn uses the API provided by Google Maps to translate the GPS data sent from Taxi drivers' smartphone in real addresses, in such a way to associate every taxi driver to an area of the city and add it to the relative queue.

In addition, the Web Application Server has an administrator interface to manage failures and anomalies. To ensure the ACID properties, the database of *myTaxiService* is provided of a database server with a relational DBMS, connected with an Ethernet LAN to the Application Server. As stated above, these indications are just to clarify among the solution space, which solutions are more feasible than others when it come to the deployment of *myTaxiService*.

2.2 Product Features

This project provides interesting features for both passengers and taxi drivers.

Non registered passengers can only register to *myTaxiService*, whereas registered passengers can request a taxi through the system, that answers informing the passenger about the code of the incoming taxi and the waiting time.

On the other hand, taxi drivers should inform the system about their availability and confirm that they are going to take care of a certain call.

The system then guarantees a fair management of taxi queues: when a request arrives from a certain zone, the system forwards it to the first taxi queueing in that zone. If the taxi confirms, then the system will send a confirmation to the passenger. If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.

2.3 User Classes and Characteristics

As already mentioned, this software is dedicated both to passengers and taxi drivers, but the main focus of the software is about passengers' satisfaction, being them end users of the software, whereas taxi drivers are only intermediate users, trained to work with the application provided on their smartphones.

Therefore the graphical interface provided to passengers, both on the web and mobile application, should be more attractive and well-finished, while the taxi drivers interface should more usable, and in general may be less attractive.

Passengers can request a taxi either through a web application or a mobile application.

Taxi drivers use a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call.

Under a certain point of view, we could consider even an additional user class which is the one of the developers that will be using the project API to develop further services based on the provided ones. This class needs an extensive documentation about the programmatic interfaces they will integrate in their development, but no proper graphical or textual user interface is provided for them, so no it will be no mentioned further in the document. Even the system administrators could be considered as a further user class of the system, but that is a bit out of the scope of this document, which is more intended to the explain things around the core business of *myTaxiService*.

2.4 Operating Environment

This software will operate on different machines in different environments.

Its distributed part should be installed on smartphones and be always connected to the Internet.

Taxi drivers must have a GPS capable smartphone because their part of the software requires an always active and updated GPS signal to run properly.

The server runs in a dedicated facility, within a controlled and protected environment, provided with a persistent Internet connection.

2.5 Design and Implementation Constraints

Being the software deployed in a complex and non controllable environment, it could encounter some issues.

The most common issue that could affect *myTaxiService* is an Internet congestion: if the whole network slows down or become unavailable, the service could encounter major problems.

That issue could easily target passengers or taxi drivers when using their mobile application to communicate with the server. For instance, they could be connected to a mobile Wi-Fi hotspot, to a WiMAX hotspot, or to a mobile 2G/3G/4G network cell, without any assurance of the bandwidth availability or network stability, thus leading to slow communication and possible timeouts.

A GPS connection is required only on taxi drivers smartphones, in order for them to communicate their current position, so the strength of the GPS signal on their smartphone is also a possible issue.

No external application is required by *myTaxiService*, nor any interface with external applications is provided.

Object oriented, imperative and declarative programming languages will be used for *myTaxiService* development, as well as markup and modelling languages.

2.6 Assumptions and Dependencies

The software is going to interact with computer browsers and smartphones, smartphone GPS integrated antennas and external ISPs, and we assume that they meet their specification. We are going to assume that Taxi Drivers are registered into *myTaxiService* when hired, and this is done by a dedicated administrator. We are going to assume also that Taxi Drivers don't notify the end of a ride before or after the actual real end of the considered ride.

3 Specific requirements

This section contains the system requirements presented in a a level of detail sufficient to enable the design of such a system.

These requirements have several properties: Correct, Unambiguous, Complete, Consistent, Ranked, Verifiable, Modifiable and Traceable.

3.1 External interface requirements

The software *myTaxiService* provides several external interfaces, both software and hardware.

A detailed coverage over those interfaces requirements is given in the following subsections.

3.1.1 User interfaces

Three different graphical user interfaces are provided: two of them for both registered and non registered passengers and one for the taxi drivers.

One of the two graphical interfaces provided for both kind of passengers is web based and uses web pages to present the content of a web application, and the other one is that of a smartphone application

The GUI provided for taxi drivers is that of a smartphone application.

No additional user interfaces are provided, not graphical nor textual. Below are provided the mockups of the most significant features guaranteed by the application.

Firstly are shown the screens of the passenger interface, putting beside the application and the web-application ones, with the aim of showing an example of the different space management.

3.1.1.1 Passenger mockups

- Homepage

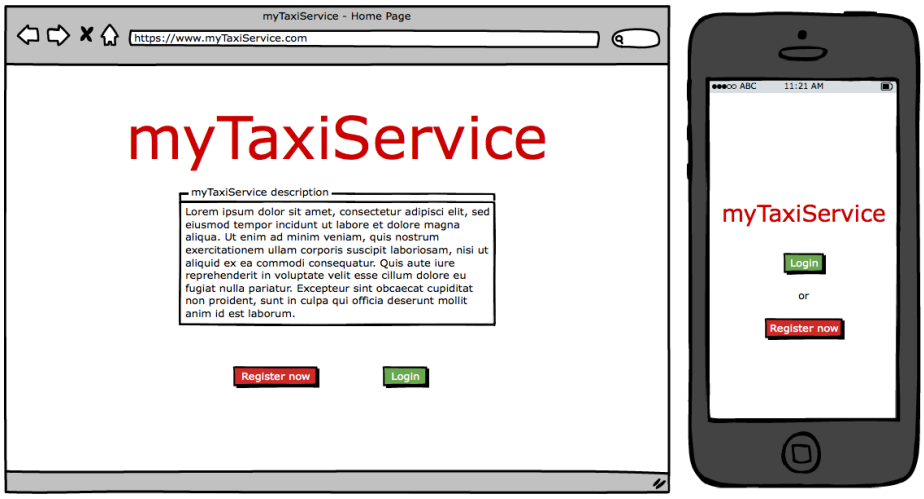

Figure 2: *myTaxiService* homepage.

- Registration

Figure 3: Registration screens.

- Login

Figure 4: Login screens.

- Empty registered passenger home page

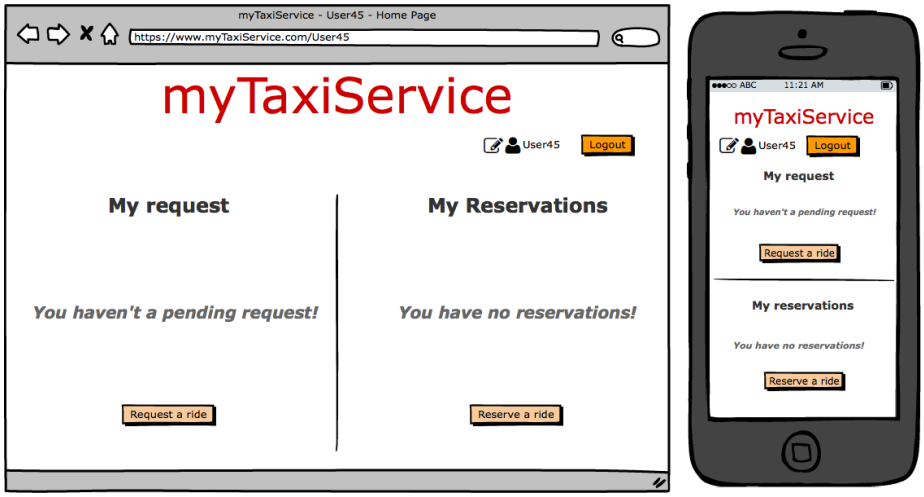

Figure 5: State 1 of RP home page.

- Request a taxi home page

Figure 6: Taxi request.

- Reserve a taxi home page

Figure 7: Taxi Reservation.

- Registered passenger home page

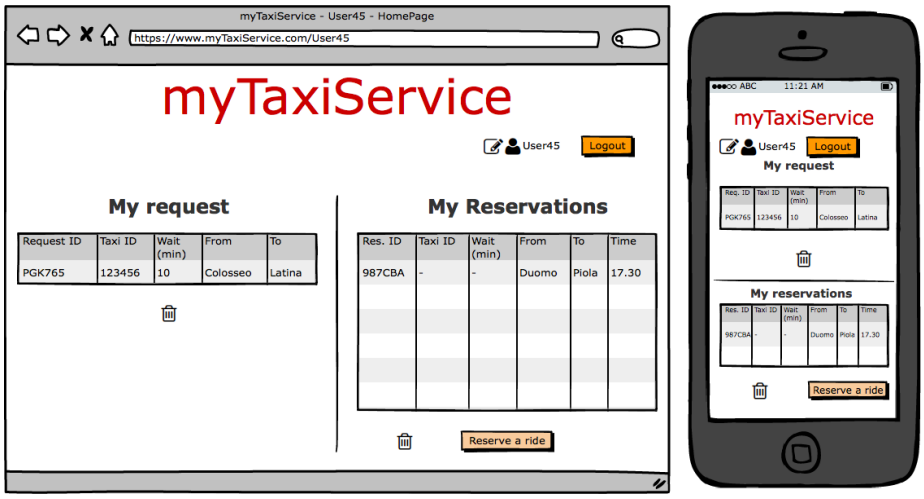

Figure 8: State 2 of RP home page.

- Cancellation of a ride in registered passenger home page

Figure 9: State 3 of RP home page.

3.1.1.2 Taxi Driver mockups

- Homepage

Figure 10: TD homepage.

- Login

Figure 11: TD login page.

- Empty driver home page

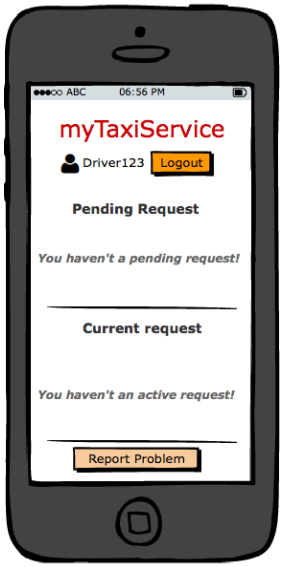

Figure 12: State 1 of TD personal page.

- Taxi driver home page with a pending request

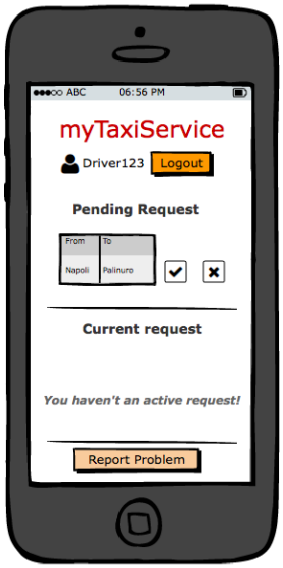

Figure 13: State 2 of TD personal page.

- Taxi driver home page while is serving a request

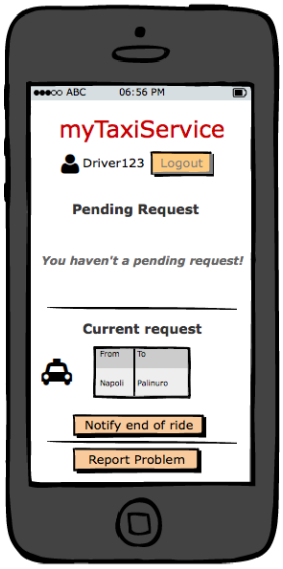

Figure 14: State 3 of TD personal page.

- Report problem

Figure 15: TD report problem screen.

- Report solution

Figure 16: TD report solution screen.

3.1.2 Hardware interfaces

3.1.3 Software interfaces

3.1.4 Communications interfaces

3.2 System Features

The system features are organized by use cases, so each subsection here represent a different use case for *myTaxiService*.

3.2.1 UML Use Case Diagram

The use case considered are represented in this diagram:

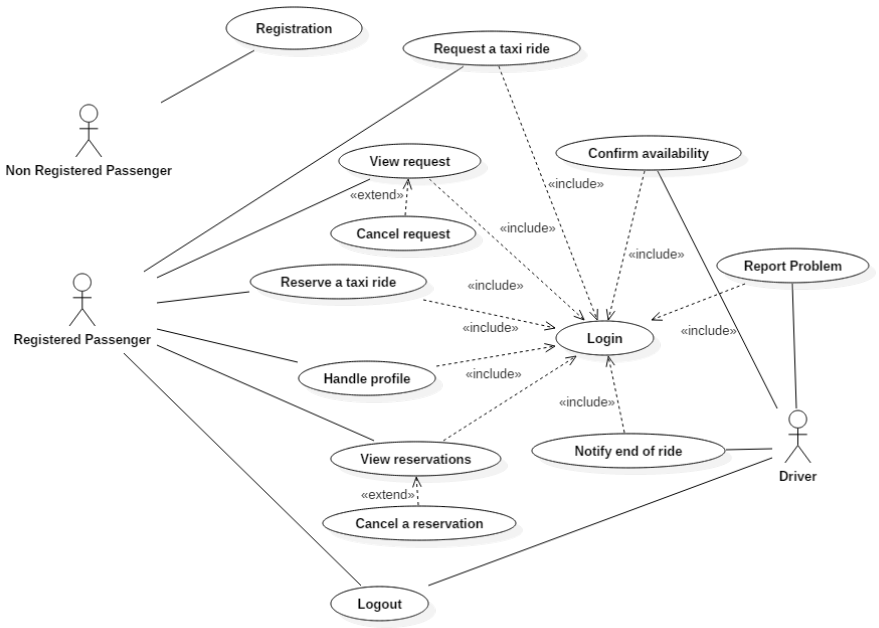

Figure 17: UML Use Case Diagram

3.2.2 Registration

This feature is related to goal **G1**, **G2** and **G12**, and to **Registration** use case.

3.2.2.1 Use Case

- **Actors:** Non registered passenger (referred as Bob for brevity).
- **Pre condition:** Bob is not registered to *myTaxiService* and wants to do so.
- **Post condition:** Bob is now registered to *myTaxiService* and therefore he is a member of registered passenger users class.
- **Event Flow:**
  - Bob reaches either *myTaxiService* web page or application, and selects the registration input control.
  - Bob fills the registration page with his personal data, accepts the treatment of his personal data and selects the confirmation input control.
  - The system acknowledges the registration of Bob to the service and sends him a confirmation email.

- **Exceptions:**
  - The browser or the application is closed at any time: the whole registration process is invalidated.
  - The personal data inserted in the registration page are not valid: on confirmation the system asks Bob to correct the necessary informations.
  - Bob is already registered: on confirmation the system neglects registration.

3.2.2.2 Description and Priority This feature is of high priority, it is essential for the system to be.

3.2.2.3 Functional Requirements

- **FR1:** A non registered passenger can register only once.
- **FR2:** Personal data inserted during registration must be valid.
- **FR3:** In no case can a registration process be resumed after browser or application closure.
- **FR4:** The system must acknowledge the non registered passenger whether the registration process ends successfully or fails.
- **FR5:** The system must send an email to the non registered passenger that has completed the registration.
- **FR6:** After the registration is completed successfully, the non registered passenger involved in the process must be considered a registered passenger.

3.2.2.4 Scenarios

- Carl wants to use the new come *myTaxiService* because his friends told him how useful it is when it comes to move in town via taxi. Therefore he opens the *myTaxiService* application he just installed in his smartphone and presses the registration button. He fill all the required fields with his data, accepts *myTaxiService* conditions and then a dialog confirms his registration to the service. An email also arrives at his address. Being now a registered user, he can start using *myTaxiService* in town.
- Alice is a taxi driver and she got his taxi driver login credentials on the day she was hired, so she don't need to register to *myTaxiService* at all.

3.2.2.5 UML Sequence Diagram

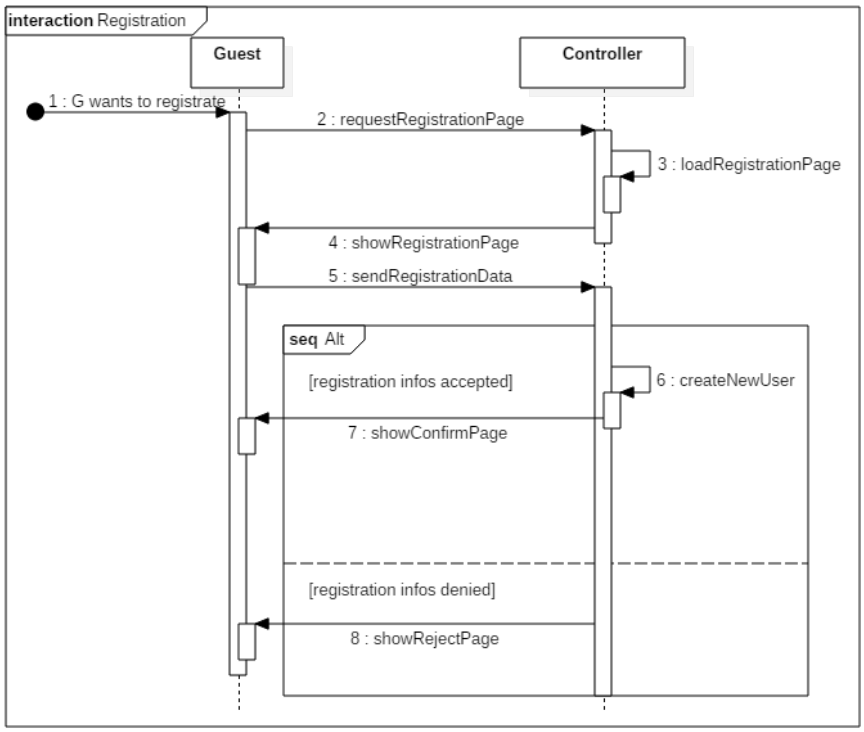

Figure 18: Registration UML Sequence Diagram

3.2.3 Login

This feature is related to goal **G3** and **G13** and to **Login** use case.

- **Actors:** Registered passenger or a taxi driver (referred as Jim for brevity).
- **Pre condition:** Jim is registered but not logged to *myTaxiService*.
- **Post condition:** Jim is logged to *myTaxiService*.
- **Event Flow:**
  - Jim reaches either *myTaxiService* web page or application.
  - Jim fills the login inputs with his credentials, and selects the confirmation input control.
  - The system acknowledges the login of Jim to the service.
- **Exceptions:**
  - The login credentials inserted are invalid: the system shows an error, reject the login and asks for correct credentials.
  - Jim's Internet connection goes down: he is not logged out.

3.2.3.1 Description and Priority This feature is of high priority, it is essential for the system to be.

3.2.3.2 Functional Requirements

- **FR7:** The system must check the correctness of the user's inserted data.
- **FR8:** The system must correctly handle internally the login token assignment to the user.
- **FR9:** The system must notify the user about the result of his login.
- **FR10:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.3.3 Scenarios

- Jim has already registered to *myTaxiService* and wants to use the service on his smartphone. He opens the mobile application and inserts his login credentials. The password he inserted was not the one he chose during registration, because he made a typo using his keyboard. The systems rejects his login and the application shows an error on-screen. Jim then inserts more carefully the correct password and presses the login button: the systems acknowledges his login and Jim can use *myTaxiService* as he wanted to do.

3.2.3.4 UML Sequence Diagram

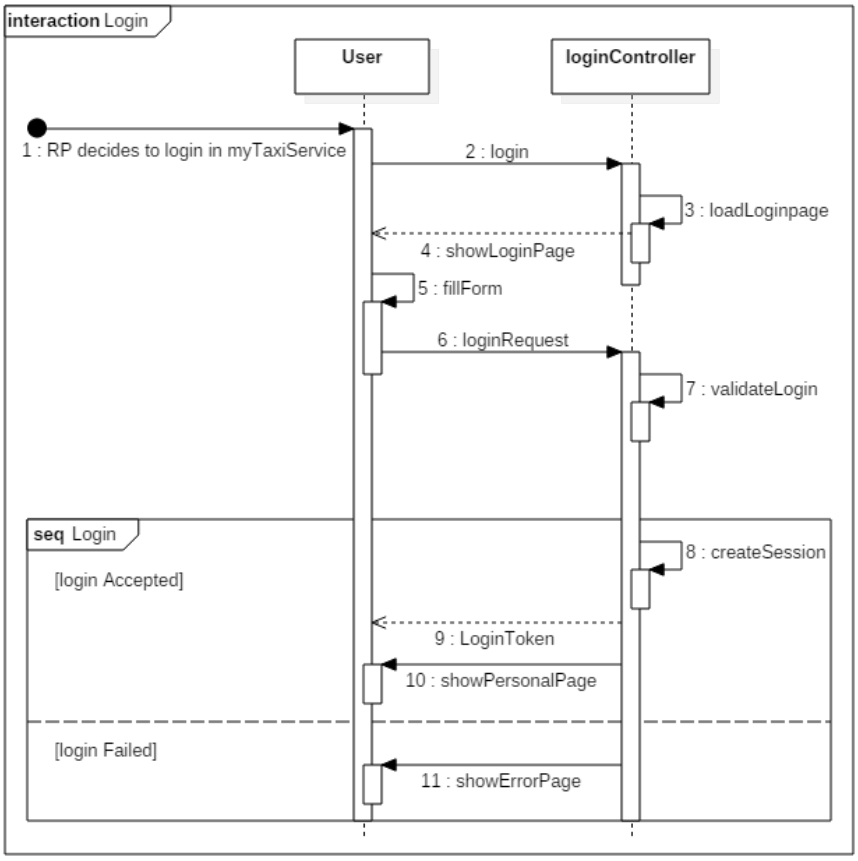

Figure 19: Login UML Sequence Diagram

3.2.4 Logout

This feature is related to goal **G4** and **G14**, and to **Logout** use case.

- **Actors:** Registered passenger or a taxi driver (referred as Nick for brevity).
- **Pre condition:** Nick is logged to *myTaxiService*.
- **Post condition:** Nick is not logged to *myTaxiService*.
- **Event Flow:**
  - Nick reaches either *myTaxiService* mobile application or web page.
  - The system loads automatically Nick's personal home page.
  - Nick selects the logout input control.
  - The system acknowledges Nick's logout and redirects him to the home page.
- **Exceptions:**
  - Nick's Internet connection goes down: he is not logged out.

3.2.4.1 Description and Priority This feature is of medium priority, it is not essential for the system to be, but can lead to benefit in term of hardware resources utilization.

3.2.4.2 Functional Requirements

- **FR11:** The system must offer the possibility to logout to a given RP.
- **FR12:** After the logout, the system must redirect the RP to the *myTaxiService* home page.
- **FR13:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.4.3 Scenarios

- Foo is using *myTaxiService* in his web browser and decides to end his activity to dedicate to something else. He is a good guy and therefore he wants to cleanly logout from *myTaxiService* before closing the web page. Then, Foo clicks on the logout input control and the system acknowledges his action: he is not logged to *myTaxiService* anymore and closes peacefully the web browser.

3.2.4.4 UML Sequence Diagram

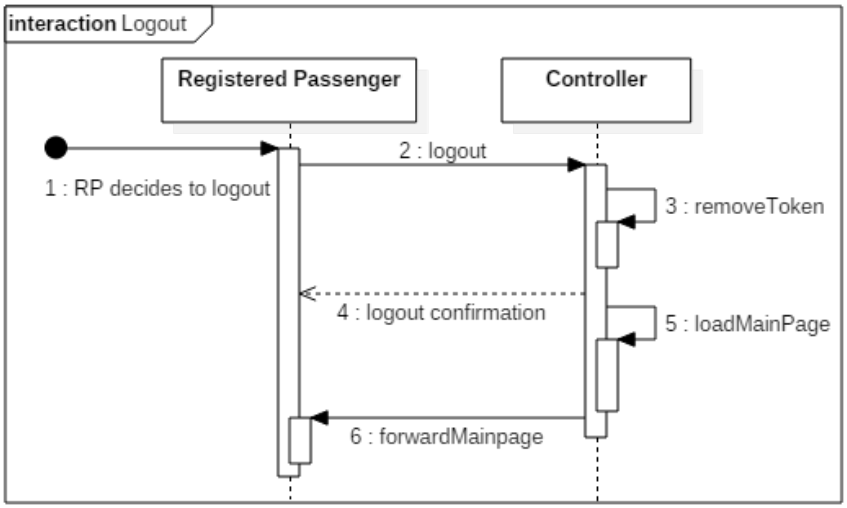

Figure 20: Logout UML Sequence Diagram

3.2.5 View Request and Reservations

This feature is related to goal **G5**, **G6**, **G7** and **G8**, and to **View Request**, **Cancel Request**, **View Reservation**, **Cancel Reservation** use cases.

- **Actors:** Registered passenger (referred as Adam for brevity).
- **Pre condition:** Adam is logged to *myTaxiService*.
- **Post condition:** Adam is still logged to *myTaxiService*.
- **Event Flow:**
  - Adam reaches either *myTaxiService* mobile application or web page and logs in.
  - The system loads automatically into Adam's page all of his taxi request and reservations into two separate lists.
  - Optionally, he can select one of the taxi rides ordered and cancel it through a pop-up on-screen.
- **Exceptions:**
  - Adam's requests list is empty: an info message is displayed by the system in place of the empty list.
  - Adam's reservations list is empty: an info message is displayed by the system in place of the empty list.
  - Adam's Internet connection goes down while the pop-up to cancel a ride is open: he is notified to try again later.

3.2.5.1 Description and Priority This feature is of medium priority, it is not essential for the system to be, but can increase by far the overall usability of *myTaxiService* for registered passengers.

3.2.5.2 Functional Requirements

- **FR14:** The system must be capable of loading a given RP data at any time.
- **FR15:** The system must ask a RP for cancel confirmation before actually cancelling a taxi ride.
- **FR16:** A message must be displayed in place of an empty list in case there's not Taxi Request to display or in case the Taxi Reservations list is empty for a given RP.
- **FR17:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.5.3 Scenarios

- Lucas is using *myTaxiService* in his web browser and has logged in a moment ago. He is now in front of all his requests and reservations, and he can view all the informations he need.

3.2.5.4 UML Sequence Diagram

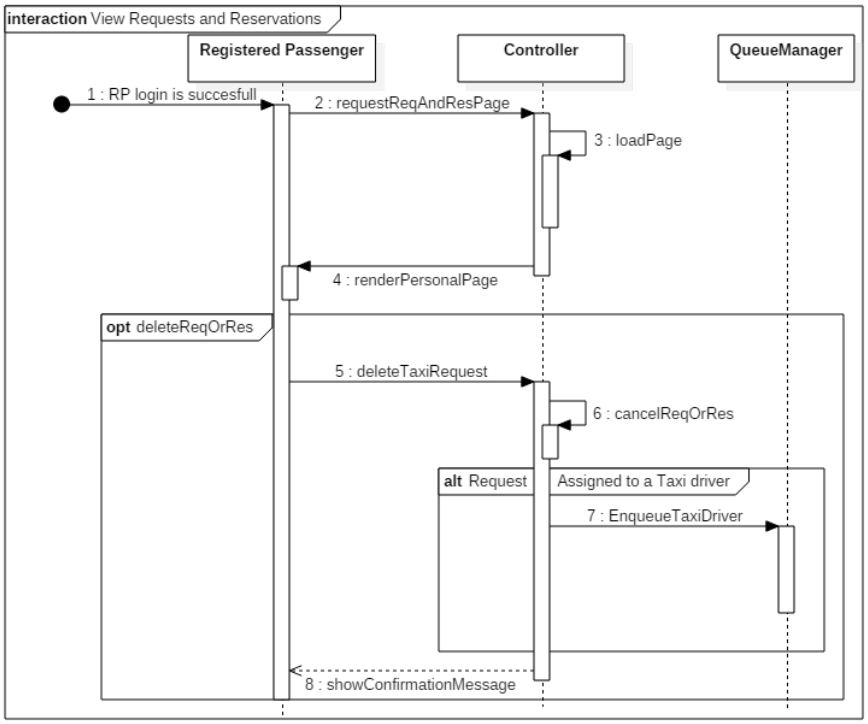

Figure 21: View Requests and Reservations UML Sequence Diagram

3.2.6 Handle Personal Profile

This feature is related to **Handle Profile** use case.

- **Actors:** Registered passenger (referred as James for brevity).
- **Pre condition:** James is logged to *myTaxiService*.
- **Post condition:** James is still logged to *myTaxiService* and his personal information are now updated.
- **Event Flow:**
  - James reaches either *myTaxiService* mobile application or web page and logs in.
  - James decides to modify his personal profile.
  - The system loads James's personal page.
  - James fills the box with the modifications he wants to perform.
  - James holds the "Modify Personal Profile" button.
  - James confirm to update his data.
  - If the modifications are correct the system responds with a confirmation message.
- **Exceptions:**
  - James inserts incorrect informations: the system sends a notification message.
  - James' Internet connection goes down: she is notified to try again later.

3.2.6.1 Description and Priority This feature is of medium priority, it is not essential for the system to be, but can increase by far the overall usability of *myTaxiService* for registered passengers.

3.2.6.2 Functional Requirements

- **FR18:** The system must check the correctness of a RP's inserted data.
- **FR19:** The system must be coherent with the RP inserted data modification, if the modification goes fine.
- **FR20:** The system must inform a RP about the result of his modification.
- **FR21:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.6.3 Scenarios

- James is waiting for his taxi and is navigating *myTaxiService* application.

  He sees the modify profile button and he decides to change his credentials updating his email with one more used.

3.2.6.4 UML Sequence Diagram

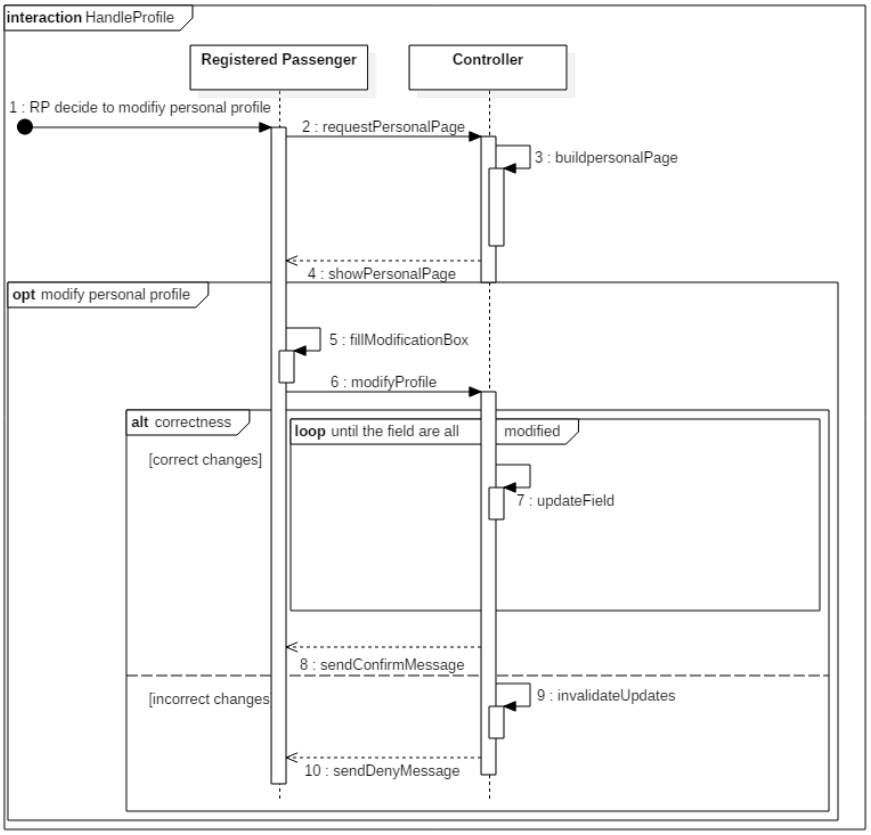

Figure 22: Handle Profile UML Sequence Diagram

3.2.7 Taxi Reservation

This feature is related to goal **G6**, and to **Reserve a taxi ride** use case.

- **Actors:** Registered passenger (referred as Dylan for brevity).
- **Pre condition:** Dylan is logged to *myTaxiService*.
- **Post condition:** Dylan is still logged to *myTaxiService*, he has received a Notification Message and he is now waiting for the a system message with a Taxi ID and a waiting time.
- **Event Flow:**
  - Dylan reaches either *myTaxiService* mobile application or web page and logs in.
  - The system loads automatically into Dylan's page.
  - Dylan pushes the "RESERVE A RIDE" button.
  - The system loads automatically into "RESERVE A RIDE" page.
  - Dylan inserts the origin, the destination, the date and the time of the Taxi reservation.
  - Dylan pushes the "Reserve" button.
  - The system adds the Taxi reservation to the Pending Taxi Ride Set.
  - Dylan receives a notification message about the correct processing of the reservation.
- **Exceptions:**
  - Dylan provides incorrect information in the reservation form: he has to reinsert the informations.
  - Dylan's Internet connection goes down: he is notified to try again later.

3.2.7.1 Description and Priority This feature is of high priority, it is essential for *myTaxiService* system to be. It is the main service that the application offers.

3.2.7.2 Functional Requirements

- **FR22:** The system must check the correctness of the RP's inserted data.
- **FR23:** The system must correctly handle internally the added taxi reservation.
- **FR24:** The system must notify the RP about the result of his operation.
- **FR25:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.7.3 Scenarios

- Dylan is going to reach a friend by metro and he is taking a look to his e-mails. Scrolling them he sees a reply e-mail from a hiring newsletter asking him to show up to their office in 5 hours.

  So he decides to book a taxi in two hours from his stop to the location of the hiring office using *myTaxiService* dedicated interface, so he can meet his friend and then he can go to he hiring interview.

3.2.7.4 UML Sequence Diagram

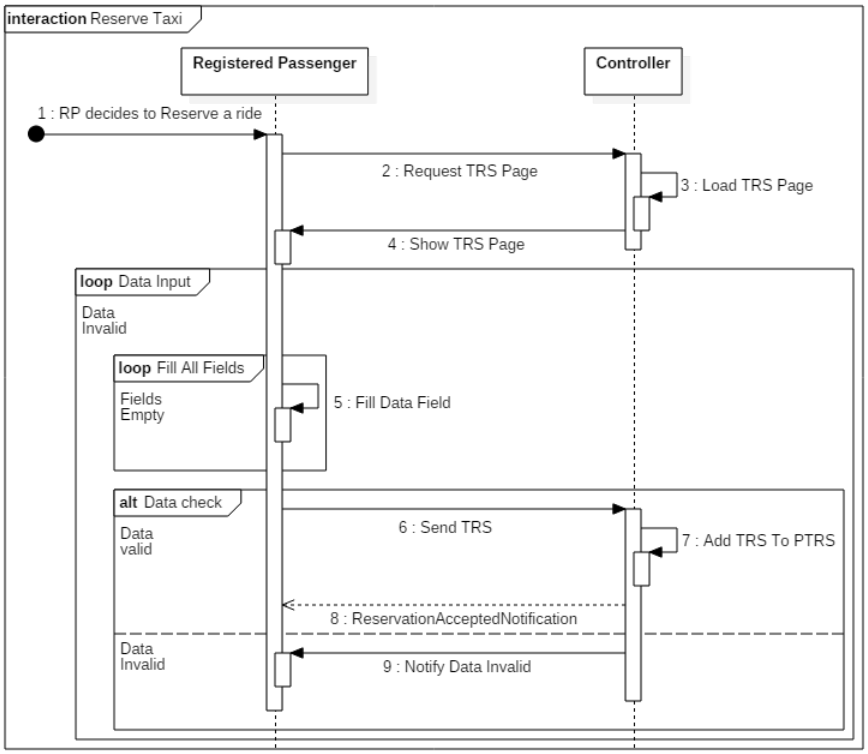

Figure 23: Taxi Reservation UML Sequence Diagram

3.2.8 Taxi Request

This feature is related to goal **G5**, and to **Request Taxi Ride** use case.

- **Actors:** Registered passenger (referred as Sarah for brevity).
- **Pre condition:** Sarah is logged to *myTaxiService*.
- **Post condition:** Sarah is still logged to *myTaxiService*, she has received a Notification Message and she is now waiting for a system message with a Taxi ID and a waiting time.
- **Event Flow:**
  - Sarah reaches either *myTaxiService* mobile application or web page and logs in.
  - The system loads automatically into Sarah's page.
  - Sarah pushes the "REQUEST A RIDE" button.
  - The system loads automatically into "REQUEST A RIDE" page.
  - Sarah inserts the origin and the destination of the Taxi ride.
  - Sarah pushes the "Request" button.
  - The system adds the Taxi request to the Pending Taxi Ride Set.
  - Sarah receives a notification message about the correct processing of the request.
- **Exceptions:**
  - Sarah provides incorrect information in the request form: she has to reinsert the informations.
  - Sarah's Internet connection goes down: she is notified to try again later.

3.2.8.1 Description and Priority This feature is of high priority, it is essential for *myTaxiService* system to be. It is the main service that the application offers.

3.2.8.2 Functional Requirements

- **FR26:** The system must check the data inserted by a RP into the Request a Ride Page and notify the RP in case of problems.
- **FR27:** The system must correctly handle internally a taxi ride that is successfully issued by a RP.
- **FR28:** The RP must be notified about the result of his request.
- **FR29:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.8.3 Scenarios

- Sarah is walking in the main street of the city. With no notice she receives a message from a friend asking her to have a drink in 30 minutes in a remote area 40 minutes walk from where she is.

  It's late and she is scared to take a bus or a metro. So she decides to download *myTaxiService* app thanks to a Wi-Fi area near her. Once downloaded she registers herself and she uses the application for calling a taxi.

3.2.8.4 UML Sequence Diagram

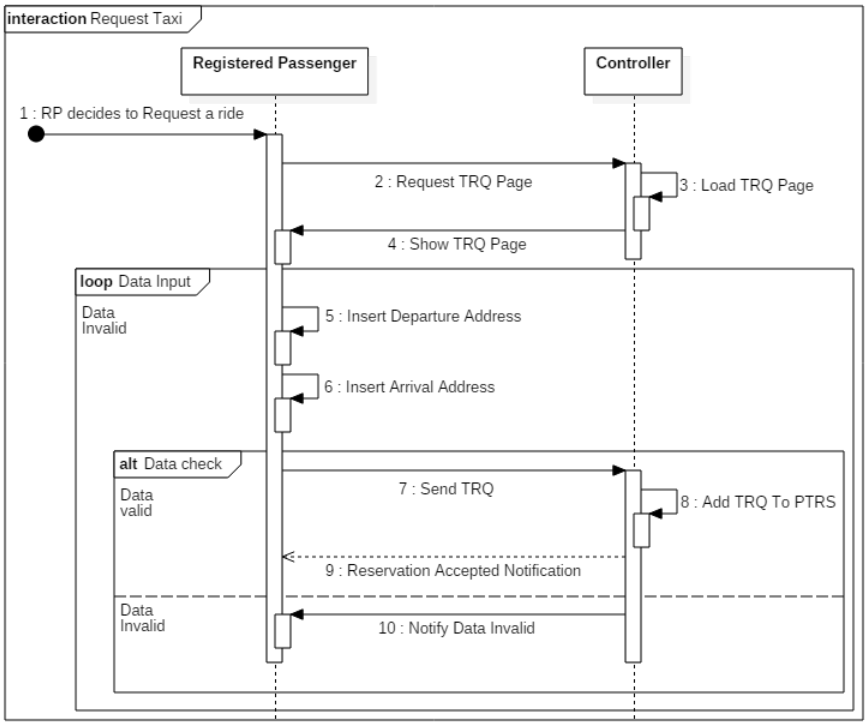

Figure 24: Taxi Request UML Sequence Diagram

3.2.9 Notify Problem

This feature is **Report Problem** use case.

- **Actors:** Taxi Driver (referred as Amy for brevity) and Registered Passenger (referred as Frank for brevity).
- **Pre condition:** Amy is logged to *myTaxiService* and is giving a Taxi Ride to Frank.
- **Post condition:** Amy is still logged to *myTaxiService* and her problem has been solved; Frank will reach his destination with Amy's taxi or another taxi automatically called for him.
- **Event Flow:**
  - Amy opens *myTaxiService* mobile application.
  - The system loads automatically into Amy's page.
  - Amy decides to press the "Report problem" button.
  - The system loads the dedicated screen.
  - Amy describes the problem using the dedicated text area.
  - Amy checks the combo box if the problem can be resolved without changing taxi.
  - Amy sends the report.
  - If the problem is not solvable without changing taxi the system automatically creates a new taxi request for Frank.
  - Else Amy can continue working. EndIf.
  - The system puts the newly created request in the correct queue.
  - The system notify Amy that the service is finished.
  - The system waits for the first available taxi to join the request.
  - If the taxi is found Frank's interfaces is updated.
  - Else if the taxi is not found the system will notify Frank that no taxi can join his request.
  - Else if the system responds with the requested information and the ride can continue. EndIf.
- **Exceptions:**
  - Amy's Internet connection goes down: she is notified to try again later.

3.2.9.1 Description and Priority This feature is of medium-high priority for the system to be. In fact without this feature some taxi rides would not be ended without external human aid, and therefore the efficiency of the system would not be the best one.

3.2.9.2 Functional Requirements

- **FR30:** The system must offer the possibility to report a problem with a dedicated text area and a combo box to specify whether the problem is solvable without interrupting the taxi driver's work.
- **FR31:** The system must be capable of automatically issuing a new taxi request for the passenger that is involved in a non solvable problem during his taxi ride.
- **FR32:** The system must be capable of notifying Amy about the result of her problem reporting.
- **FR33:** The system must be capable of notifying Frank about the evolving of his automatically issued request.
- **FR34:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.9.3 Scenarios

- Amy is taking Frank to the other part of the city. They are talking about Mauro Icardi's last goal when the car stops with no advice.

  Amy tries to turn it on but she can't. Then she takes her phone and reports the problem to the system specifying that the problem is not solvable.

  The report is processed and the system sends Amy a notification saying that a new taxi is coming.

  Amy waits till the taxi arrived, greets Frank and they gather at the stadium. Frank takes the new taxi and heads to his destination.

3.2.9.4 UML Sequence Diagram

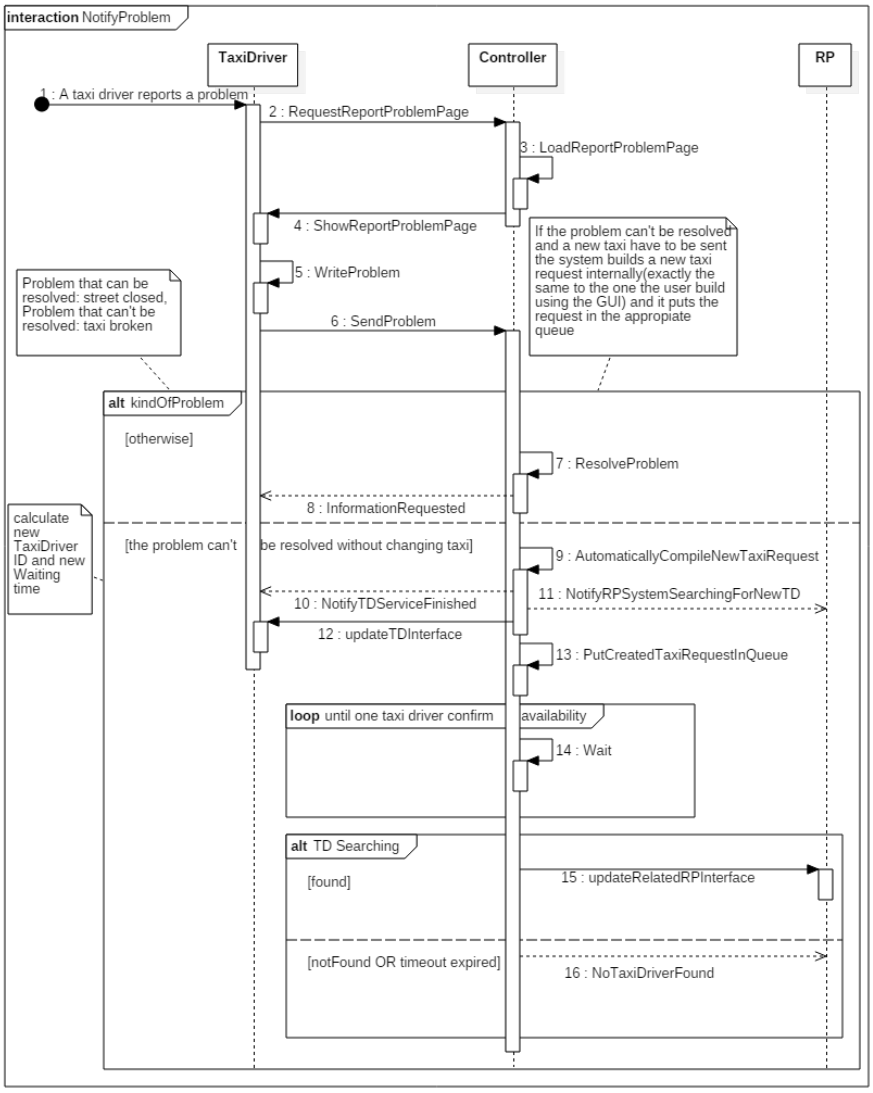

Figure 25: TD Notify Problem UML Sequence Diagram

3.2.10 End Of Ride

This feature is related **Notify End of Ride** use case

- **Actors:** Taxi Driver (referred as Bill for brevity).
- **Pre condition:** Bill is logged to *myTaxiService* and has finished his taxi ride.
- **Post condition:** Bill is still logged to *myTaxiService* and is able to offer a new taxi ride.
- **Event Flow:**
  - Bill *myTaxiService* mobile application is open and the "Current Ride" area is populated.
  - Bill pushes the "Notify End Of Ride" button.
  - The system marks the ride as terminated.
  - The system enqueues Bill to the queue of the zone in which the destination address is located, using the Location Manager service.
  - Bill receives a message about the confirmation of the end of the ride.
- **Exceptions:**
  - Bill's Internet connection goes down: he is notified to try again later.

3.2.10.1 Description and Priority This feature is of high priority, it is essential for the system to be: in fact without this feature the ride can't be completely handled and the system will never know when a ride is finished.

In addition Bill wouldn't be able accept another ride if he doesn't complete his current one.

3.2.10.2 Functional Requirements

- **FR35:** A taxi driver must be capable of notifying the end of a taxi ride he is serving.
- **FR36:** The system must automatically re-enqueue the taxi driver that notifies the end of his current taxi ride in the queue of his current zone.
- **FR37:** A notification must be sent from the system to the taxi driver to inform him about the result of his action.
- **FR38:** The system must peacefully handle a connection error and notify the user of the problem.

3.2.10.3 Scenarios

- Bill is arrived at the destination of his current ride. So the RP goes down from the taxi and, after he has paid, Bill has to notify he has completed his current request.

  So he holds his smart phone, he selects *myTaxiService* mobile application and once he is logged in he holds the "Notify End Of Ride" button.

  He waits some seconds and the system responds him with the confirmation message.

3.2.10.4 UML Sequence Diagram

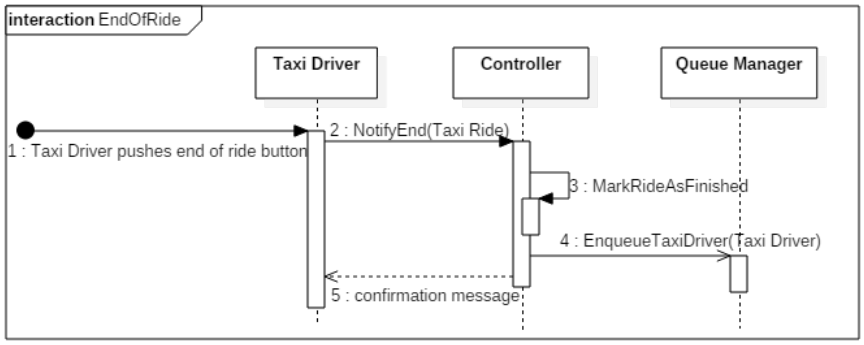

Figure 26: End Of A Ride UML Sequence Diagram

3.2.11 UML Class Diagram

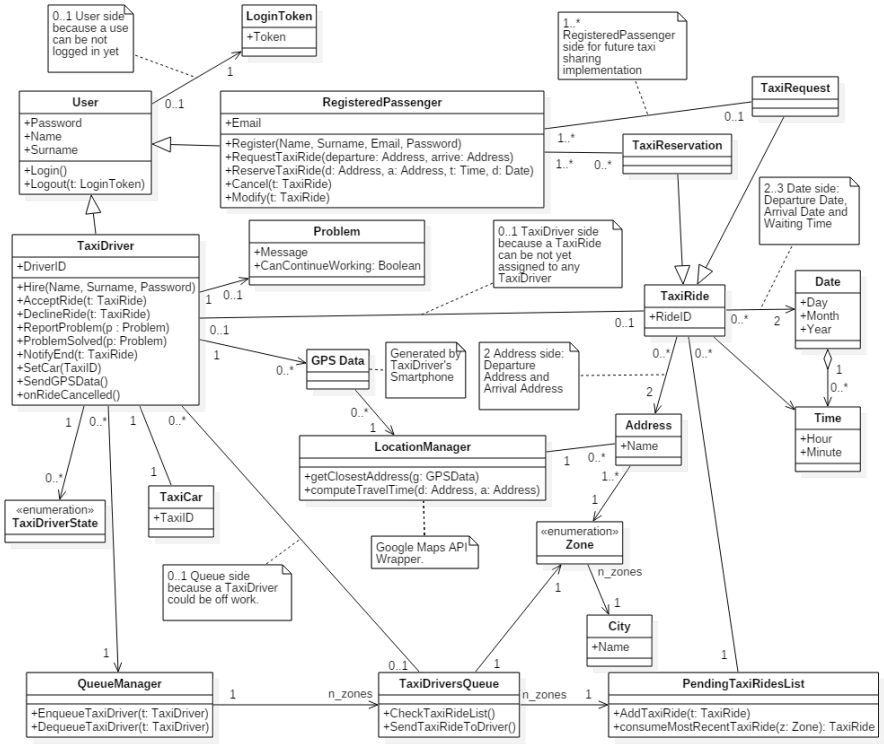

Figure 27: UML Class Diagram

3.2.12 State Chart Diagram

- **Taxi Driver State Chart:**

  The state chart diagram proposed below has te aim of better explaining the different states in which the TD can be during the use of *myTaxiService*.

  Moreover there are two particular states, "TD Not Able To Work Connection Error" and "TD Working Connection Error", that are included with the aim of better showing the software behaviour when some unexpected problem occurs.

  In these cases TD is redirected to a login screen in which he remains till the problem is resolved(for example the connections returns up).

  In particular the system is built to store and periodically update a session object in order to restore the application when TD logs in once the problem is resolved.

  In this way the login procedure brings the TD exactly in the same state in which he was after the problem.

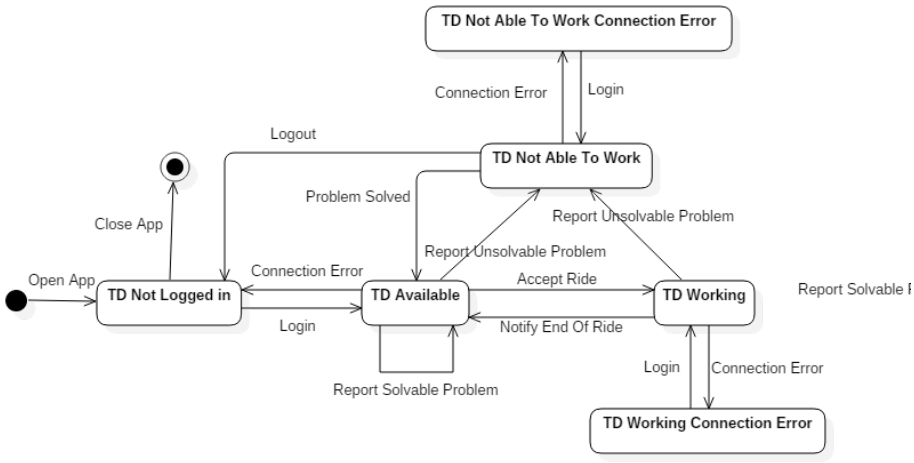

Figure 28: Taxi Driver State Chart Diagram

- **Passenger Mockup State Chart:**

  The state chart diagram proposed below is a complete description of the navigation into *myTaxiService* application.

  In fact the mockups proposed in section [3.1.1] are not completely mapping this diagram.

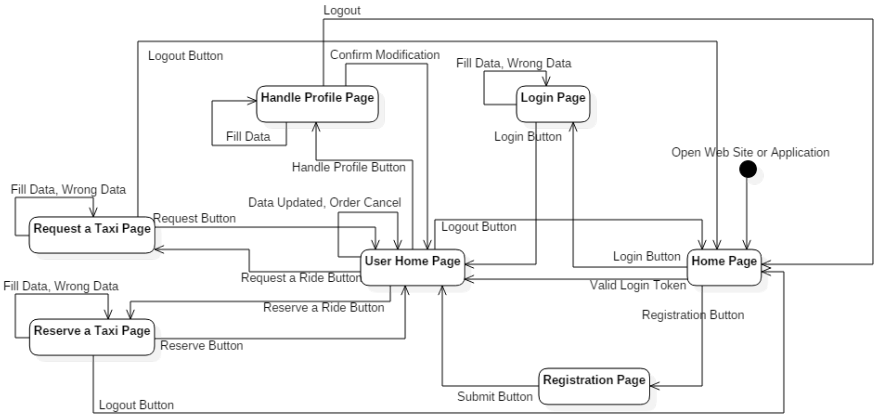

Figure 29: Passenger Mockup State Chart Diagram

3.2.13 Alloy Model

- **Alloy Signatures:**

```
/** === SIGNATURES === **/

abstract sig Boolean {}

lone sig True extends Boolean {}

lone sig False extends Boolean {}

abstract sig TaxiRide
{
   departureAddress : one Address,
   arrivalAddress : one Address
}

sig TaxiReservation extends TaxiRide {}

sig TaxiRequest extends TaxiRide {}

sig Username {}

abstract sig User
{
   username : one Username
}

sig RegisteredPassenger extends User
{
   taxiRequest: lone TaxiRequest,
   taxiReservations: set TaxiReservation
}

sig TaxiDriver extends User
{
   taxiCar : one TaxiCar,
   queueManager : one QueueManager,
   taxiRide : lone TaxiRide,
   problem : lone Problem
}

sig Problem
{
   canContinueWorking : one Boolean
}

sig TaxiCar {}

sig TaxiDriversQueue
{
   taxiDrivers : set TaxiDriver,
   zone : one Zone
}

sig Zone
{
   city : one City
}
```

Figure 30: Alloy Signature 1.

```
one sig City {}

sig Address
{
   zone : one Zone
}

one sig QueueManager
{
   taxiDriversQueue : set TaxiDriversQueue
}

lone sig PendingTaxiRides
{
   taxiRides : set TaxiRide
}
```

Figure 31: Alloy Signature 2.

- **Alloy Facts:**

```
/** === FACTS === **/

fact
{
  //No useless Boolean
  Problem.canContinueWorking = Boolean

  //No useless Username
  User.username = Username

  //No useless Problem
  TaxiDriver.problem = Problem

  //Problem are unique among TaxiDriver
  all td1, td2 : TaxiDriver | (#(td1.problem) > 0 and #(td2.problem) > 0 and td1 != td2) implies td1.problem != td2.problem

  //No useless TaxiCar
  TaxiDriver.taxiCar = TaxiCar

  //No useless Address
  TaxiRide.departureAddress + TaxiRide.arrivalAddress = Address

  //At least two Address in a city
  #Address > 1

  //At least a Address per zone
  all z : Zone | (some a : Address | a.zone = z)

  //No useless City
  Zone.city = City

  //No useless QueueManager
  TaxiDriver.queueManager = QueueManager

  //No useless TaxiDriversQueue
  QueueManager.taxiDriversQueue = TaxiDriversQueue

  //Unique username for all users
  all u1, u2 : User | u1 != u2 implies u1.username != u2.username

  //Every zone is associated to a to at least one TaxiDriversQueue
  all z : Zone | z in TaxiDriversQueue.zone

  //Different TaxiDriversQueue are associated to different Zone
  all tdq1, tdq2 : TaxiDriversQueue | tdq1 != tdq2 implies tdq1.zone != tdq2.zone

  //Two different TaxiDriversQueue have no TaxiDriver in common
  all tdq1, tdq2 : TaxiDriversQueue | tdq1 != tdq2 implies #(tdq1.taxiDrivers & tdq2.taxiDrivers) = 0

  //If a TaxiDriver is not doing a TaxiRide and has no Problem, he is in a TaxiDriversQueue
  all td : TaxiDriver | (#td.problem = 0 and #td.taxiRide = 0) implies (one tdq : TaxiDriversQueue | td in tdq.taxiDrivers)

  //If a TaxiDriver is doing a TaxiRide, he is in no TaxiDriversQueue
  all td : TaxiDriver | #td.taxiRide > 0 implies not (some tdq : TaxiDriversQueue | td in tdq.taxiDrivers)

  //Each TaxiDriver must drive a different TaxiCar
  all td1, td2 : TaxiDriver | td1 != td2 implies td1.taxiCar != td2.taxiCar
```

Figure 32: Alloy Facts 1.

```
//Different TaxiDriversQueue are associated to different Zone
all tdq1, tdq2 : TaxiDriversQueue | tdq1 != tdq2 implies tdq1.zone != tdq2.zone

//All TaxiRide have the DepartureAddress different from the ArrivalAddress
all tr : TaxiRide | tr.departureAddress != tr.arrivalAddress

//No TaxiRide is at the same time in the PendingTaxiRidesList and served by a TaxiDriver
no tr : TaxiRide | tr in PendingTaxiRides.taxiRides and tr in TaxiDriver.taxiRide

//All TaxiDriver that are serving a TaxiRide are also driving a TaxiCar
all td : TaxiDriver | #td.taxiRide > 0 implies #td.taxiCar > 0

//All TaxiDriver that have a Problem and that problem cannot let them continue working, must be serving no TaxiRide
all td : TaxiDriver | (#td.problem > 0 and td.problem.canContinueWorking = False) implies (#td.taxiRide = 0)

//Al TaxiDriver that have a Problem and are also serving a TaxiRide, must have a Problem that let them continue working
all td : TaxiDriver | (#td.problem > 0 and #td.taxiRide > 0) implies (td.problem.canContinueWorking = True)

//A TaxiRide is either in PendingTaxiRidesList or being served by a TaxiDriver
all tr : TaxiRide | (tr in PendingTaxiRides.taxiRides) iff (not (tr in TaxiDriver.taxiRide))

//No useless PendingTaxiRide
TaxiDriver.taxiRide = TaxiRide implies #(PendingTaxiRides) = 0

//Every TaxiRide is either served or pending
TaxiDriver.taxiRide + PendingTaxiRides.taxiRides = TaxiRide

//Every TaxiRide is either a requested or reserved
RegisteredPassenger.taxiRequest + RegisteredPassenger.taxiReservations = TaxiRide

//Different TaxiDriver are associated to different TaxiRide
all td1, td2 : TaxiDriver | td1 != td2 implies td1.taxiRide != td2.taxiRide

//Different TaxiReservation cannot be served at the same time to some RegisteredPassenger
no tr1, tr2 : TaxiReservation | tr1 != tr2 and (tr1 in TaxiDriver.taxiRide) and (tr2 in TaxiDriver.taxiRide) and
            (some rp : RegisteredPassenger | (tr1 in rp.taxiReservations) and (tr2 in rp.taxiReservations))

//No RegisteredPassenged can have served at the same time a TaxiRequest and a TaxiReservation
no tres : TaxiReservation, treq : TaxiRequest |  (tres in TaxiDriver.taxiRide) and (treq in TaxiDriver.taxiRide) and
            (some rp : RegisteredPassenger | (tres in rp.taxiReservations) and (treq in rp.taxiRequest))

//If a TaxiDriver has a Problem that makes him unable to continue working, he must be outside every TaxiDriversQueue
no td : TaxiDriver | #(td.problem) > 0 and td.problem.canContinueWorking = False and (td in TaxiDriversQueue.taxiDrivers)

//If there are TaxiRide in PendingTaxiRides for a given Zone, then there must be no TaxiDriver in that Zone related TaxiDriversQueue
all tr : TaxiRide | (tr in PendingTaxiRides.taxiRides) implies
            (no tdq : TaxiDriversQueue | tdq.zone = tr.departureAddress.zone and #(tdq.taxiDrivers) > 0)

//If a RegisteredPassenger has a TaxiReservation being served, he can have no TaxiRequest
all rp : RegisteredPassenger | some tr : TaxiReservation | ((tr in rp.taxiReservations) and
            (tr in TaxiDriver.taxiRide)) implies #(rp.taxiRequest) = 0
}
```

Figure 33: Alloy Facts 2.

- **Alloy Assertions:**

```
/** === ASSERTIONS === **/

//A TaxiCar is driven by only a TaxiDriver
assert TaxiCarOnlyDrivenByOneTaxiDriver
{
   all tc : TaxiCar | one td : TaxiDriver | td.taxiCar = tc
}

//No TaxiDriver can have a serious Problem and be serving a TaxiRide
assert NoTaxiDriverWithSeriousProblemCanBeServingATaxiDrive
{
   no td : TaxiDriver | #(td.problem) > 0 and td.problem.canContinueWorking = False and  #(td.taxiRide) > 0
}

//All TaxiRides are RegisteredPassenger's generate (by at least one RegisteredPassenger, given the possibility of having TaxiSharing)
assert RideAreIssuedByAtLeastARegisteredPassenger
{
   all tr : TaxiRide | some rp : RegisteredPassenger | (tr in rp.taxiRequest) or (tr in rp.taxiReservations)
}

//There are as many Username as TaxiDriver plus RegisteredPassenger
assert AsManyUsernameAsManyAsUsers
{
   #Username = #TaxiDriver + #RegisteredPassenger
}
```

Figure 34: Alloy Assertions.

- **Alloy Predicates:**

```
/** === PREDICATES === **/

pred SimpleWorld
{
   #TaxiDriver = 2
   #TaxiDriver.taxiRide = 1
   #TaxiDriver.problem = 0
   #RegisteredPassenger = 2
   #RegisteredPassenger.taxiRequest = 1
   #TaxiRide = 3
   #Zone = 2
}

pred RealWorld
{
   #TaxiDriver > 1
   #Problem > 1
   #RegisteredPassenger > 1
   #TaxiRide > 1
   #Zone = 6
}
```

Figure 35: Alloy Predicate.

- **Alloy Executions:**

```
/** === EXECUTIONS === **/

check TaxiCarOnlyDrivenByOneTaxiDriver for 6

check NoTaxiDriverWithSeriousProblemCanBeServingATaxiDrive for 6

check RideAreIssuedByAtLeastARegisteredPassenger for 6

check AsManyUsernameAsManyAsUsers for 6

run SimpleWorld for 6

run RealWorld for 6
```

Figure 36: Alloy Execution.

- **Alloy Console:**

```
Alloy Analyzer 4.2 (build date: 2012-09-25 15:54 EDT)

Warning: Alloy4 defaults to SAT4J since it is pure Java and very reliable.
For faster performance, go to Options menu and try another solver like MiniSat.
If these native solvers fail on your computer, remember to change back to SAT4J.

Executing "Check TaxiCarOnlyDrivenByOneTaxiDriver for 6"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  10620 vars. 575 primary vars. 21652 clauses. 160ms.
  No counterexample found. Assertion may be valid. 37ms.

Executing "Check NoTaxiDriverWithSeriousProblemCanBeServingATaxiDrive for 6"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  10676 vars. 575 primary vars. 21841 clauses. 55ms.
  No counterexample found. Assertion may be valid. 22ms.

Executing "Check RideAreIssuedByAtLeastARegisteredPassenger for 6"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  10610 vars. 575 primary vars. 21471 clauses. 47ms.
  No counterexample found. Assertion may be valid. 7ms.

Executing "Check AsManyUsernameAsManyAsUsers for 6"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  0 vars. 0 primary vars. 0 clauses. 39ms.
  No counterexample found. Assertion may be valid. 0ms.

Executing "Run SimpleWorld for 6"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  10639 vars. 569 primary vars. 21883 clauses. 56ms.
  Instance found. Predicate is consistent. 41ms.

Executing "Run RealWorld for 6"
  Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
  10611 vars. 569 primary vars. 21717 clauses. 47ms.
  Instance found. Predicate is consistent. 66ms.

6 commands were executed. The results are:
  #1: No counterexample found. TaxiCarOnlyDrivenByOneTaxiDriver may be valid.
  #2: No counterexample found. NoTaxiDriverWithSeriousProblemCanBeServingATaxiDrive may be valid.
  #3: No counterexample found. RideAreIssuedByAtLeastARegisteredPassenger may be valid.
  #4: No counterexample found. AsManyUsernameAsManyAsUsers may be valid.
  #5: Instance found. SimpleWorld is consistent.
  #6: Instance found. RealWorld is consistent.
```

Figure 37: Alloy Console.

- **Alloy Meta Model:**

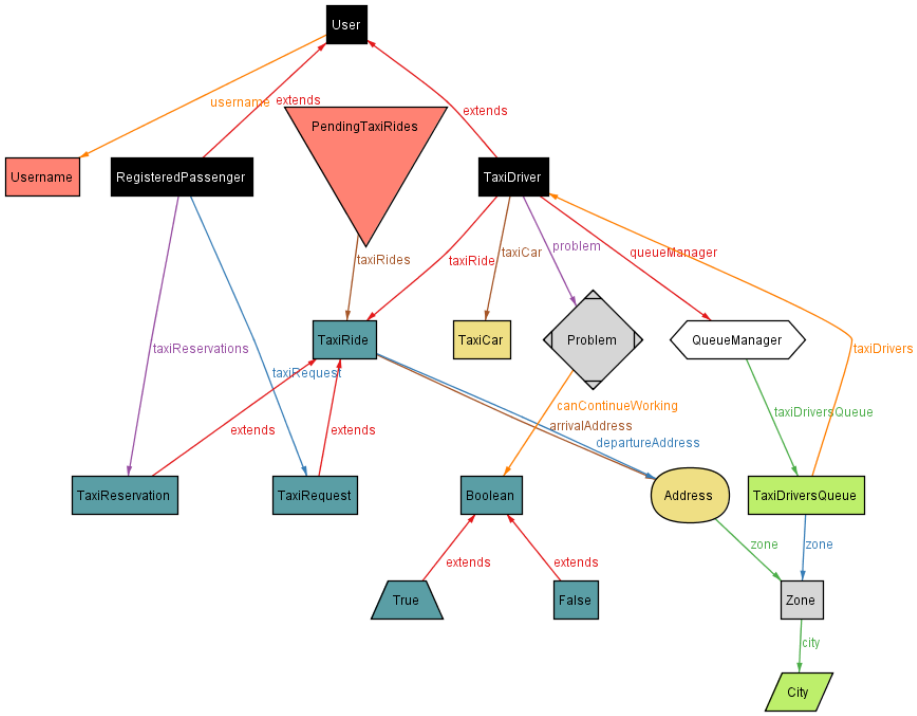

Figure 38: Alloy Meta Model.

- **Alloy Simple World:**

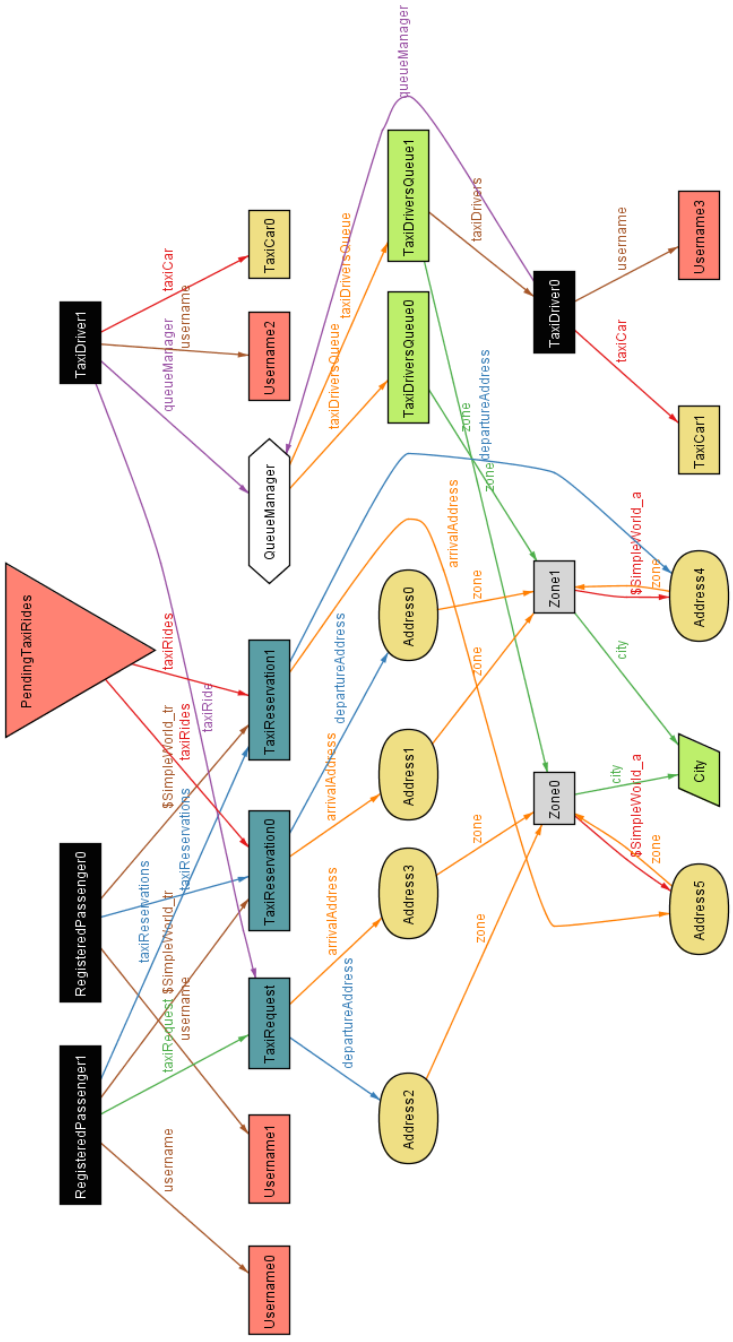

Figure 39: Alloy Simple World.

- **Alloy Real World:**

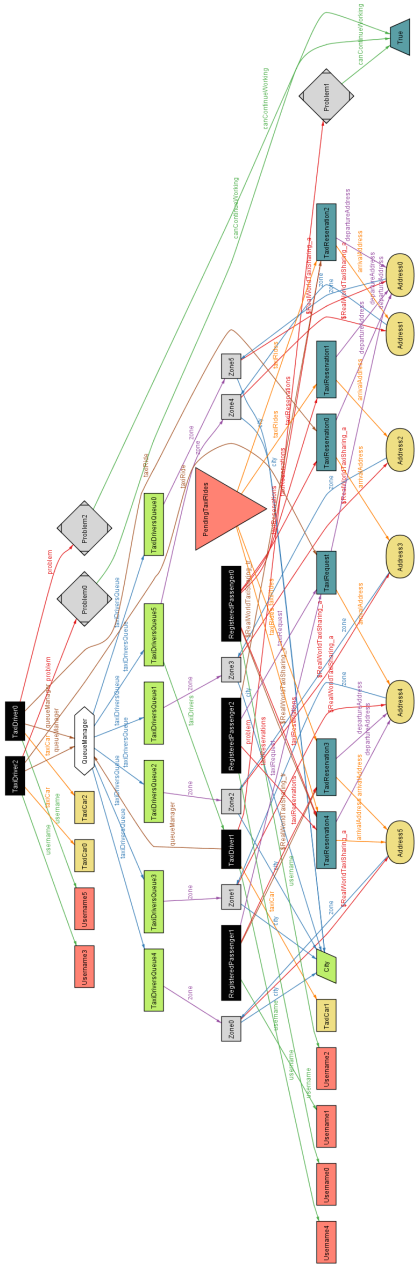

Figure 40: Alloy Real World.

3.3 Performance requirements

In this subsection are specified the most important non functional requirement.

- The system must perform every operation in less than 10 seconds.
- The system must handle 1000 users and 400 taxi drivers connected at the same time.
- The system must handle a number of Registered Passenger equals to 10 time the number of the taxi drivers in the town.

3.3.1 Reliability

The service must be available 7 days a week 24 hours a day.

The server must support parallel processing in order to guarantee a fast and continuous service.

3.3.2 Availability

The service must be up with a probability of failure less than 0.01 percent and must be able to resist to 9 security attack out of 10.

3.3.3 Security

Internet connection must be encrypted with a recognized SSL certificate (HTTPS protocol). The handling of users data must be protected by SQL data stealing.

Moreover the DMBS must guarantee a Serializable SQL isolation level.

3.3.4 Mantainability

The software must be build accompanied with proper documentation and it must comply with Model-View-Controller programmative pattern. In fact using this pattern ensures great opportunities of extension.

3.3.5 Portability

The system must support the integration between different OS, OSX, Windows, Ubuntu.

Moreover the user experience must be the same if user decide to access the service via IE, Opera,Chrome and Firefox. In the case the user decides to use the mobile application the system must guarantee the same feature in Android base system and IOS base system.

4 Appendix

4.1 Tools

- **Texstudio:** LaTeXeditor for building RASD document.
- **AlloyAnalizer 4.2:** Used to check Alloy model consistency.
- **StarUML:** Used to build UML Sequence Diagrams, UML State Chart Diagrams, UML Use Case Diagrams and UML Class Diagrams.
- **SourceTree:** Used to allow team work and synchronize Github repository.
- **Blasamiq Mockups 3.0:** Used to build web-application and smart phone application mokcups.

4.2 Hours Of Work

- **Alessando:** 39
- **Alberto Mario:** 39