A.Y. 2015/2016

POLITECNICO DI MILANO

COMPUTER SCIENCE AND ENGINEERING

SOFTWARE ENGINEERING 2

MyTaxiService

RASD Document

Authors

SARA PISANI – 854223
LEONARDO TURCHI – 853738

6/11/2015

MyTaxiService-RASD.pdf · V.1.0

Table of contents

1. Introduction

1.1 Recipients

This document is applied to professors who teach software engineering 2 in the years 2015/2016, but other users interested in this documents could be the developers of a system that manage requests of taxi rides.

1.2 Purpose

The purpose of this document is the description of a system that offer a taxi service.
The software developed will be a web application that allow user to book a taxi ride, immediately or starting from two hours after, and offers a taxi sharing service.

1.2.1 Stakeholders

- **Passengers**: they can book a taxi ride by inserting a departure and a destination, with the chance to share the ride with other passenger to save money.
- **Drivers**: they make available their taxi on-line and they can accept or decline ride requests from passengers.

1.2.2 System Functionality

- Ride's customization
  Passengers can personalize a ride simply choosing a button: "Ride Request" or "Ride Booking". Once the choice has been taken, passengers have to fill out a form with two required fields (departure and destination) for the immediate booking (ride request), and four required fields (departure, destination, date and time) for the delayed booking (ride booking). The fields “date” and "time" allow to book a taxi after a couple of hour or more.
  Once completed the form, the button “Taxi Sharing” makes the reservation provided for sharing and manages other passengers who are interested in the same ride.

- Consultation of ride’s details
  After an immediate booking (ride request) appears a popup on screen with all ride’s data.

After one or more delayed booking (ride booking), appears a list on screen with all ride's data anticipation. If users want more information about a specific ride, they must click on button "info" and a popup appears on screen with that ride's data.

- Booking management
  The system creates a priority queue of available taxi in each zone, and the current request is sent to the taxi on top of the queue.
  If he accepts, the system enqueues the driver in back of the queue and saves the ride's data.
  If he declines, the system enqueues the driver in back of the queue and iteratively sends the request to the first taxi in the queue.

- Response to reservation
  Drivers on top of the queue receive a popup notification about the ride, with complete details.
  If he accepts the passenger receives a confirm popup notification.

1.2.3 Goals

- The system allows an easy and fast taxi's immediate reservation (ride request).
  The research is optimized by geographic zone.

- The system allows an easy and fast taxi's delayed reservation (ride booking), and a fast ride's details consultation. The research is optimized by geographic zone.

- The system allows to share a ride: passengers could take advantage from this service by saving money and, with a special consideration for the environment,
  This service reduces the number of automobiles on the road and the traffic.

- The system allows drivers to be organized by a taxi queue: they save time because their rides are optimized by geographic zone.

1.3 Definitions and acronyms

1.3.1 Definitions

- **System**: The software that will be developed.
- **Service**: Everything That will be make available to the users by the system.
- **Ride**: The main service, that concerns passenger's mobility.

1.3.2 Acronyms

- **DBMS**: Database Management System
- **J2EE**: Java 2 Enterprise Edition
- **JVM**: Java Virtual Machine

1.4 References

- Assignments 1 and 2.pdf
- Assignments rules and group registration.pdf
- IEEE standard for requirement specification.pdf
- Alloy reference manual.pdf

1.5 Overview

The paper is organized as follows:

- **Chapter 1**: This section explains the target audience, the purpose and the references of the Project.

- **Chapter 2**: This section provides a general description of the system, its features and its requirements.

- **Chapter 3**: This section contains all the necessary software requirements to allows a proper implementation.

- **Chapter 4**: This section contains the appendix: here there are the logic model requirements.

- **Chapter 5**: This section is used for the notes and all the future revision of the document.

2. General Description

2.1 Product perspective

The application that will be released is a web application which is not integrated with other existing system.
That application could have (in future) an internal interface for a call-center administration, but right now it is only user-based.
The application will not provide any interface or public API for integration with other project.

2.2 User characteristics

The user that we expect to use our application is a person who want an easy way to travel in their city. This user must be able to use a web browser and have access to internet.

2.3 Constraints

2.3.1 Regulatory policies

MyTaxiService doesn't have to meet any regulatory policies.

2.3.2 Hardware limitations

MyTaxiService doesn't have to meet any hardware limitations.

2.3.3 Interfaces to other applications

MyTaxiService doesn't have to meet any interfaces to other applications.

2.3.4 Parallel operation

MyTaxiService must support parallel operations from different users when working with database and with all operation done by the user after connection.

2.3.5 Documents related

- Requirements and Analysis Specification Document (RASD)
- Design Document (DD)
- Testing report
- Inspection Document

2.4 Assumptions and Dependencies

- There isn't an administrator or privileged user. Is not necessary a hierarchy of users to keep the system safe.
- Registrations can be done only by users; drivers' data are inserted directly in the database by the company: this to guarantee the registration of only certified drivers.
- Using the simple taxi service there isn't any dependence between users, but using the taxi sharing service there is.
- Ride request's confirm popup notification will be sent only if there's at least an available taxi, otherwise a message or a notice will inform the user.
- If a passenger wants to delete a taxi reservation, he must do it using the popup that compares after the ride customization.
- The choice to set a ride shareable will not be possible to modify later.
- PayPal payment mode note: money transfer will be done only when a driver mark as complete a ride by the popup. In case of troubles passengers can obtain a refund contacting the call center.

2.5 Future possible implementation

- **Increase of customizing filters:**
  MyTaxiService wants to provide more possibility of personalization for the sharing service, so could be inserted more fields to choose the characteristics of the traveling companion (for example smoker or not, chatterbox or not…).
- **Chance to send a feedback:**
  from passengers to the driver and vice versa and from passengers to other passengers who share the same taxi.
- **Provide a calendar to the drivers:**
  with the handy data of every ride that they have confirmed, the organization could be better.
- **Reduce cheating:**
  providing passengers the route and the estimated time of their ride, inefficiency caused by drivers would be reduced.
- **Service scalability:**
  The service is initially available for a single city, but is scalable. System manager can add a new serviced city to extend the working area.

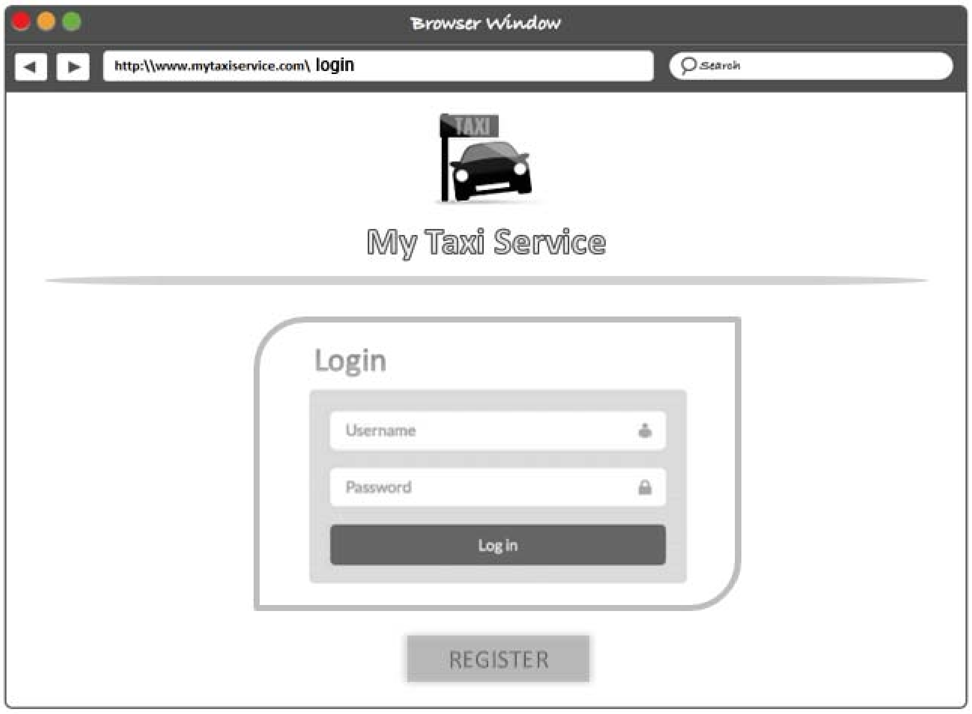
3. Specific Requirements

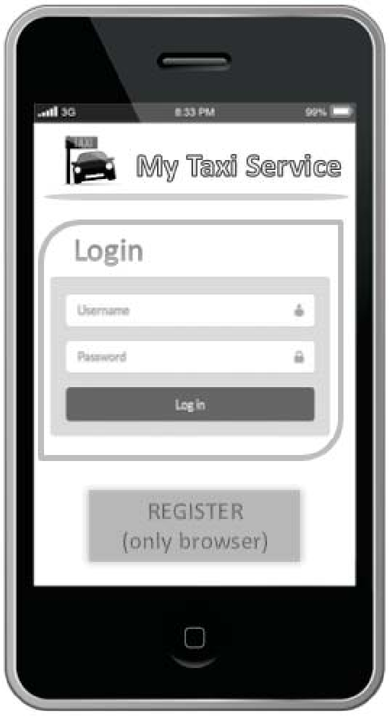
3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.1.1 Login

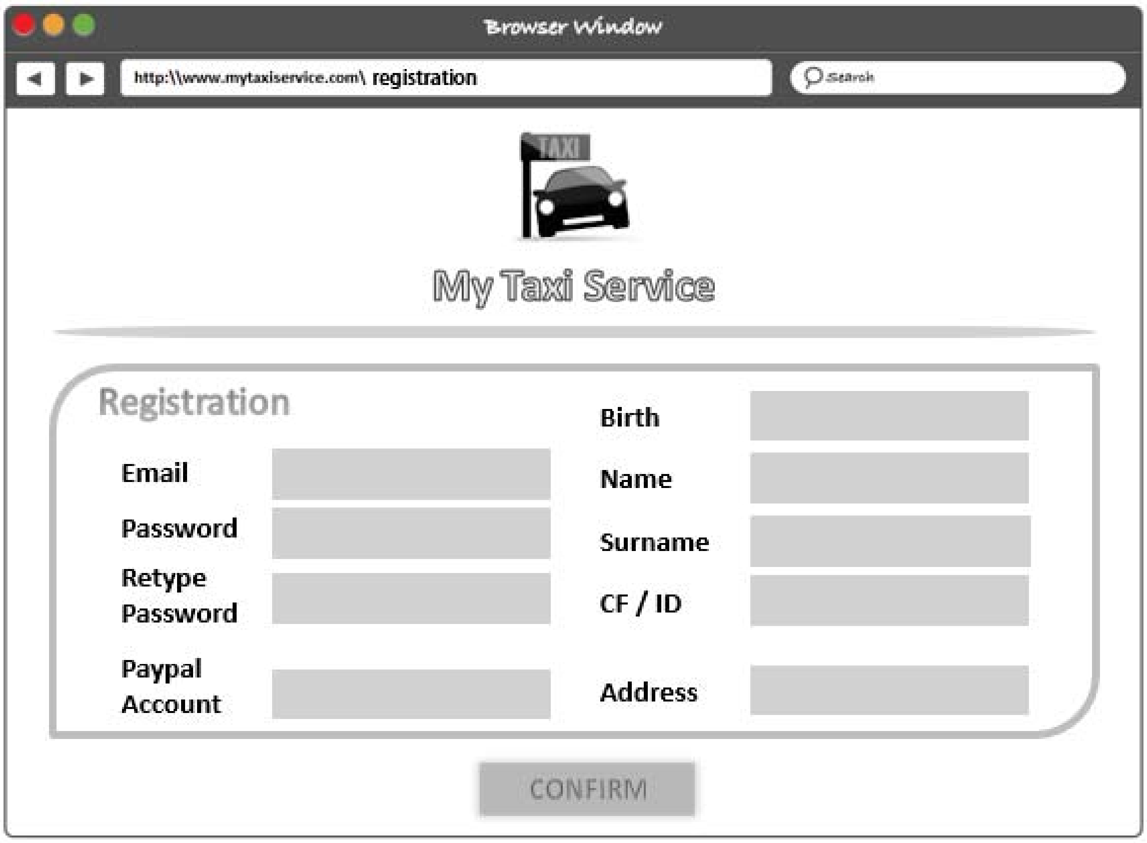
Here is a template of the home page of MyTaxiService.
Users can log in to the site or access the registration page.





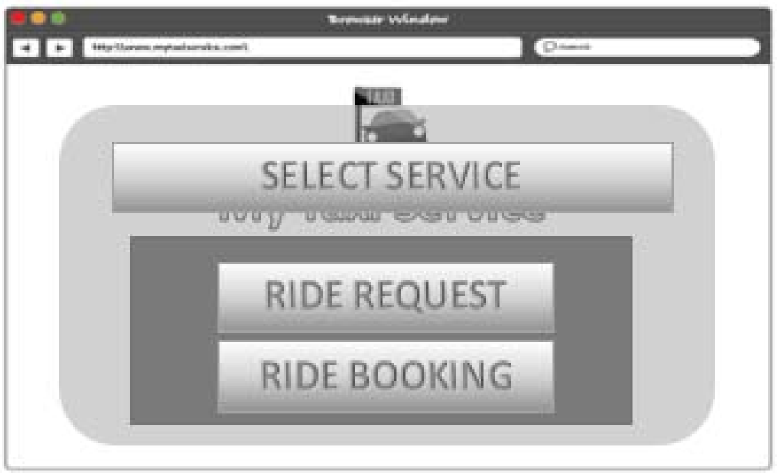
3.1.1.2 Registration form

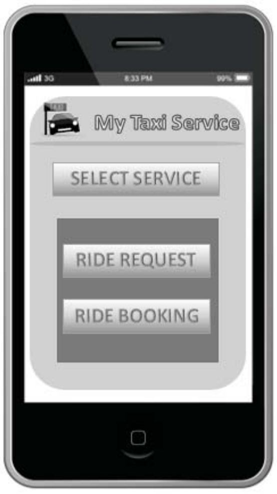
The first registration to the service must be done via browser.
A generic user becomes a registered user after the submission of a form.



3.1.1.3 Service selection

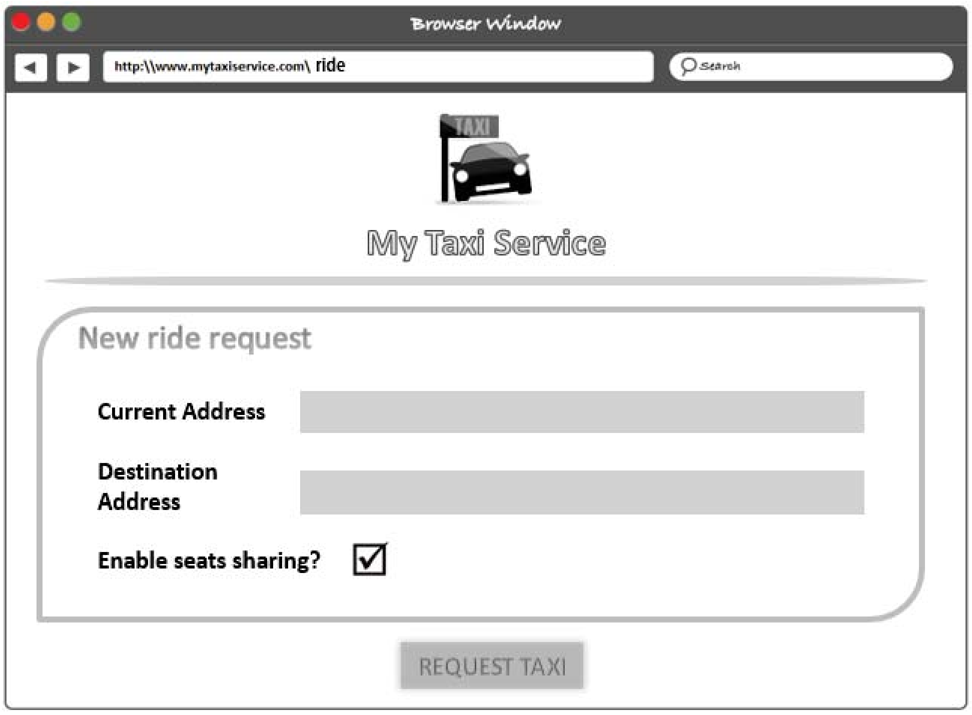
This mockup shows the main services menu of the site/app.
The user has to click to enter in the requested area.

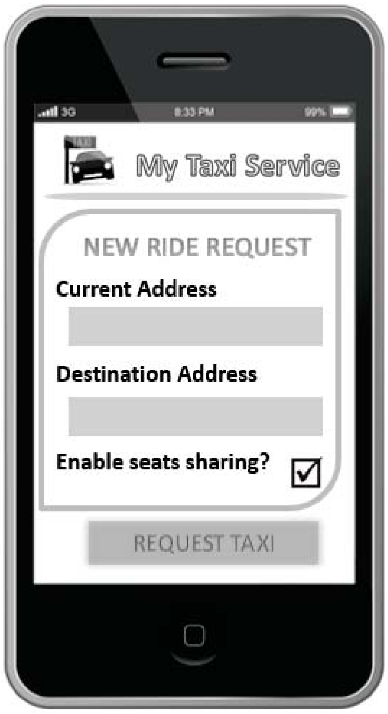




3.1.1.4 Ride Request

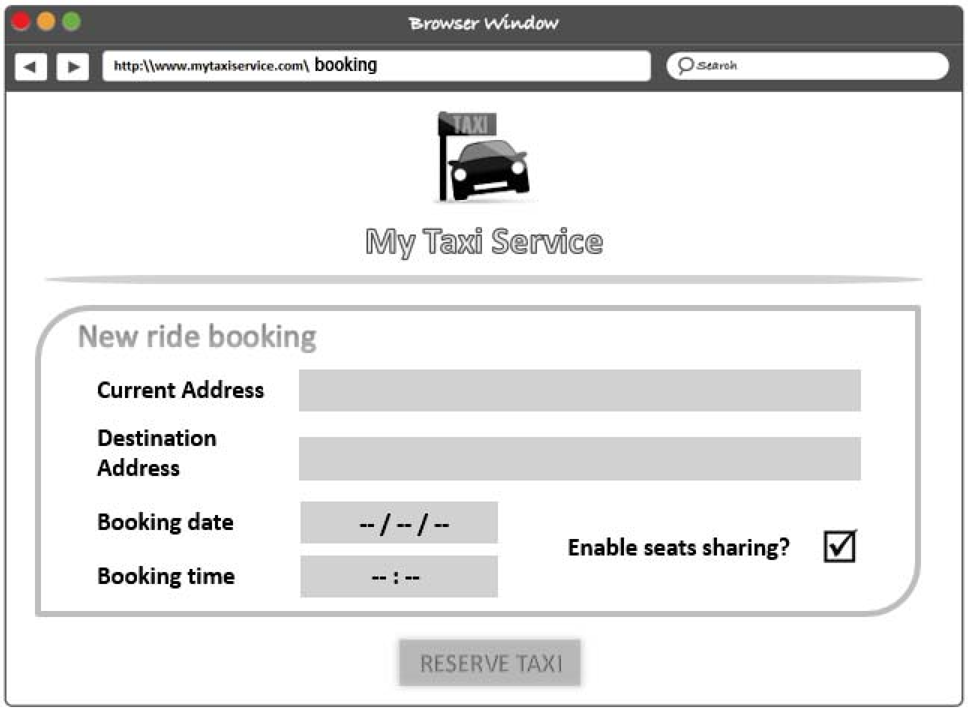
This mockup shows the primary page for a taxi request.
In this case is for a single ride without booking.
In addition the user can choose if other seats are shared with other people or not.

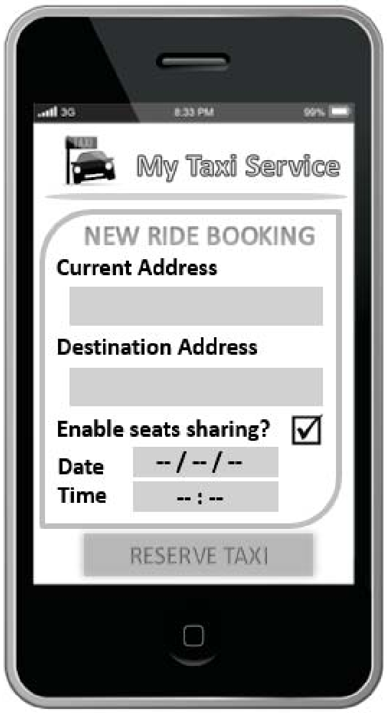




3.1.1.5 Ride Booking

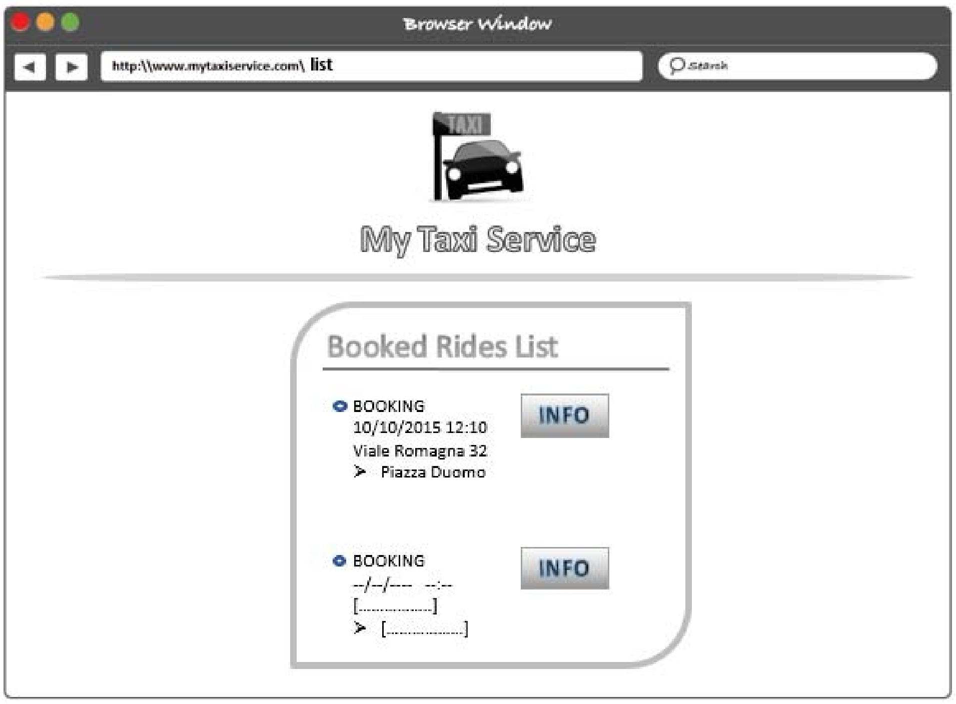
This mockup shows the primary page for a taxi booking request.
This version includes the possibility of a taxi ride in a specific date and hour.
In addition the user can choose if other seats are shared with other people or not.

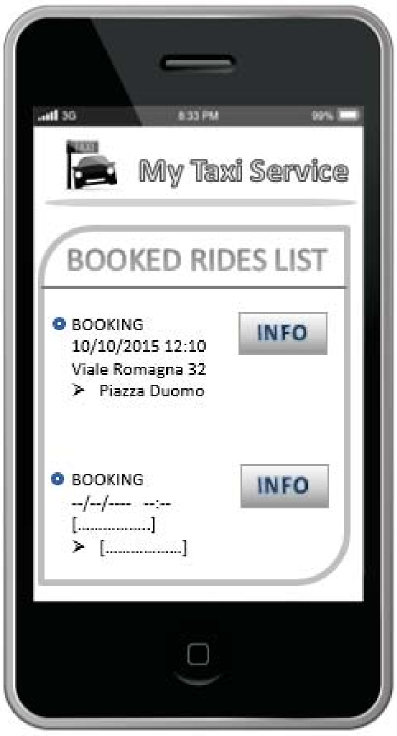




3.1.1.6 Booked Rides List

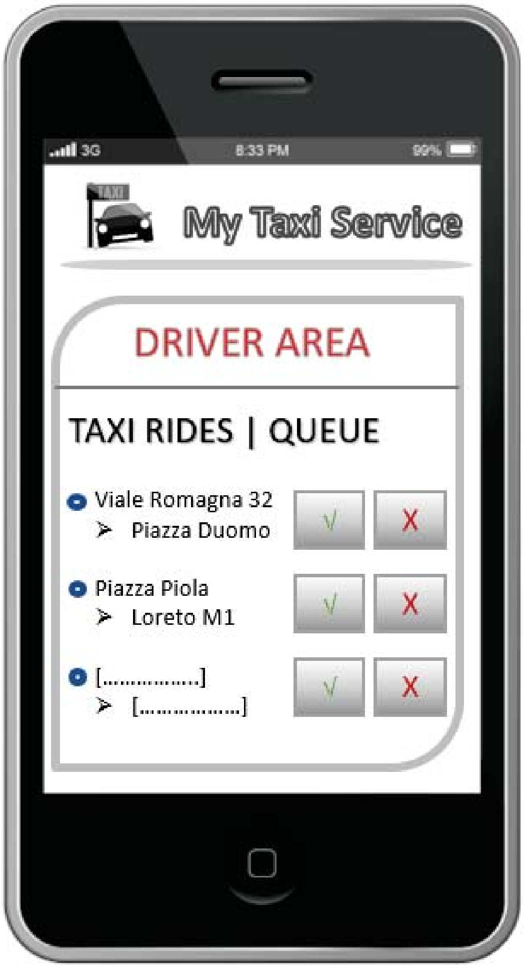
This mockup shows the main page for the booked rides management.
The user can scroll up/down to search for a specific ride (after booking), and click on 'info' button to see the detail's popup.
In addition, on popup the user is able to delete a booked ride.

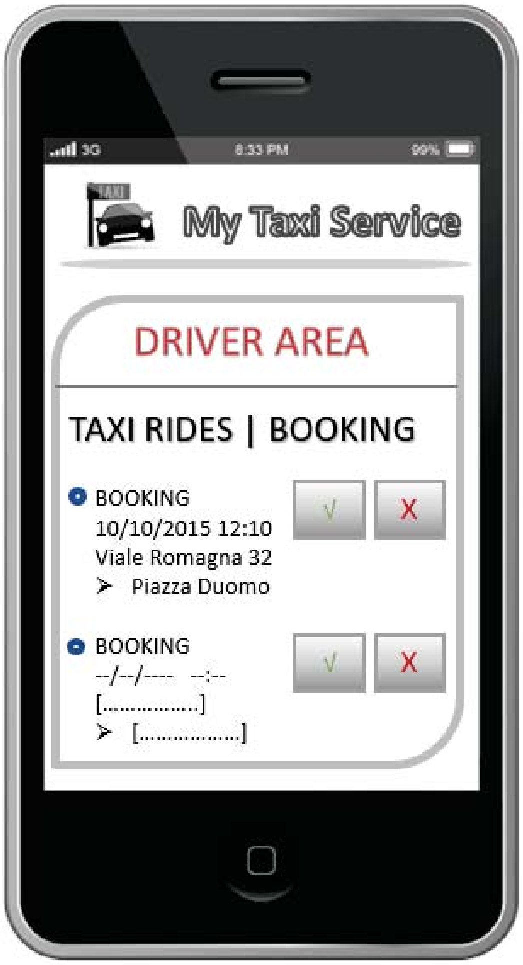




3.1.1.7 Driver area

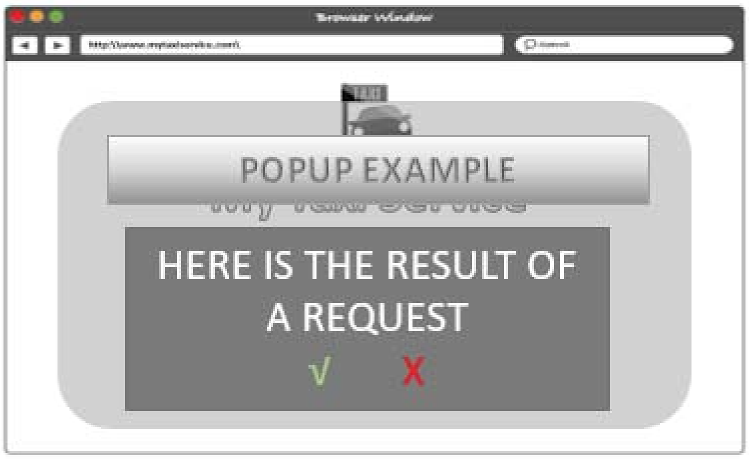
Only for drivers, in the app there is a dedicated area.
In this section drivers are able to see the queue in theirs zone and accept or decline new rides.
With a right-slide are also able to manage the reservations: a list of booking rides is shown, drivers can accept or hide reservations.
This section is designed to be used in mobility, so there is no browser version.

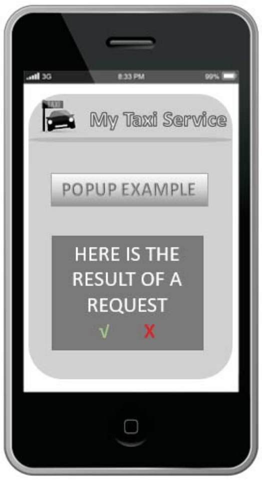


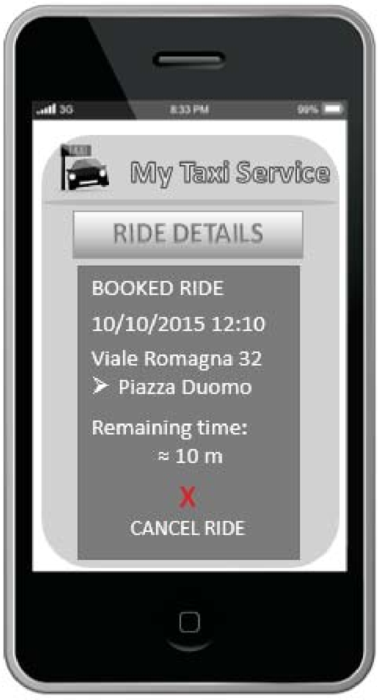


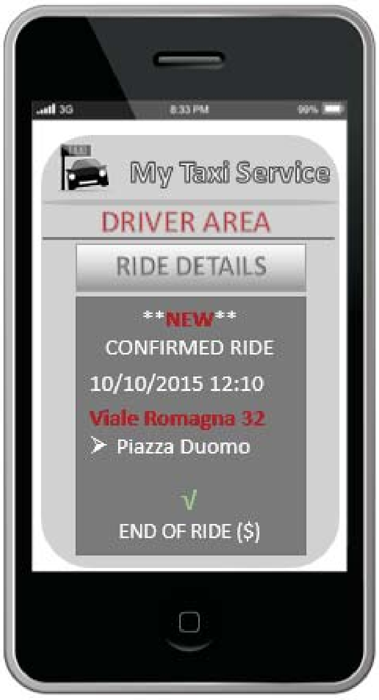
3.1.1.8 Popups

This mock up shows examples of pop-up alert.
Popups are used to inform or alert the user of something.
In addition, popups are used to show to user the result of a request or an action, as consequence of a click.
In some cases are visible (enabled to click) even the buttons, in other not, depending by the action required.
Finally popups are used to show to user details of a ride and allow him/her to manage these ones.









[example of a populated popup user-side] *[example of a populated popup driver-side]*

3.1.2 API Interfaces

To convert addresses to GPS coordinates we use the google Maps Geocoding API.

As well described on the website, Geocoding is the process of converting addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates (like latitude 37.423021 and longitude -122.083739), which you can use to place markers on a map, or position the map. Also Data is available in JSON and XML format.

A future implementation could be to make available to the passenger the smartest route:
It will possible to use Google Maps Roads API.

The Google Maps Roads API allows you to map GPS coordinates to the geometry of the road, and to determine the speed limit along those road segments. The API is available via a simple HTTPS interface, and exposes two services:

- **Snap to roads**: This service returns the best-fit road geometry for a given set of GPS coordinates. This service takes up to 100 GPS points collected along a route, and returns a similar set of data with the points snapped to the most likely roads the vehicle was traveling along. Optionally, you can request that the points be interpolated, resulting in a path that smoothly follows the geometry of the road.

- **Speed limits**: This service returns the posted speed limit for a road segment. The Speed Limit service is only available to Google Maps API for Work customers. If you are an existing customer, you can contact your account manager or file a ticket in the Maps API for Work support portal to enable the Google Maps Roads API.

3.1.3 Hardware Interfaces

This project does not support any hardware interfaces.
The software runs on a server and displays output in remote devices.

3.1.4 Software Interfaces

- Database Management System (DBMS):
    - Name: MySQL
    - Version: 5.7
    - Source: http://www.mysql.it/
- Java Virtual Machine (JVM).
    - Name: JEE
    - Version: 7
    - Source: http://www.oracle.com/technetwork/java/javaee/tech/index.html
- Application server:
    - Name: Glassfish
    - Version: 4.1.1
    - Source: https://glassfish.java.net/
- Operating System (OS).
    - Application must be able to run on any SO which supports JVM and DBMS specified before

3.1.5 Communication Interfaces

Protocol	Application	Port
TCP	HTTPS	443
TCP	HTTP	80
TCP	DBMS	3306 (default)

3.1.6 Memory

The minimum memory requirements are:

- Primary Memory: 2GB+
- Secondary Memory: 32GB+

3.2 Functional Requirements

Acronyms for the correct reading of this section:

- **[F#x]** means 'Functionality #x'
- **[D#x]** means 'Database constraint #x'
- **[G#x]** means 'Goal #x'

3.2.1 Common functionalities (every goal need these):

3.2.1.1 [F1] Allow a visitor to become a registered user.

- [R1] Visitor must not be already registered to perform registration process.
- [R2] Visitor must choose a username not already used by another user.
- [R3] User cannot sign up twice but only once for session.
- [R4] Visitor can just see login page.
- [R5] Visitor can only access to registration form.
- [D1] Email address used for registration must be formally correct.
- [D2] PayPal account is mandatory and must be inserted at the moment of the registration process.
  The system checks the correctness and makes the validation with the PayPal server before allowing user to enjoy services.

3.2.1.2 [F2] Allow user to log in to application.

- [R1] User must be already registered to success login process.
- [R2] User must know his username and password used during registration to success login.
- [R3] Username and password insert during login process must be correct.
- [R4] Wrong credentials will not grant access to user to other pages.
- [R5] Visitor can't access to booking page before registration.
- [R6] Application will not implement retrieve password mechanism.

3.2.2 [G1] The system allows an easy and fast taxi's immediate reservation (ride request).

The research is optimized by geographic zone.

3.2.2.1 [F3] Allow user to request an immediate taxi ride.

- [R1] User must be already registered and logged in the application.
- [R2] User must complete booking form and confirm the ride.

3.2.2.2 [F4] Allow user to delete an existing ride.

- [R1] User must be already registered and logged in the application.
- [R2] User must be the owner of the ride he wants to delete.
- [R3] User must delete a ride only by using the popup.
- [R4] Driver must not have accepted the booking.
- [R5] Deleting process is not reversible, all event data will be lost.

3.2.2.3 [F5] Allow user to see their booking data.

- [R1] User must be already registered and logged in the application.
- [R2] Booking must exist, owner of the booking has correctly created it.

3.2.2.4 [F6] After login, application will notify passengers if their booking has been accepted or declined by the taxi driver

- [R1] User must be already registered and logged in the application.
- [R2] Booking must exist, owner of the booking has correctly created it.
- [R3] Application will notify passengers of the taxi only when they perform login.

3.2.3 [G2] The system allows an easy and fast taxi's delayed reservation (ride booking), and a fast ride's details consultation.

The research is optimized by geographic zone.

NOTE:
This goal has the functionality [F5] [F6] of [G1] plus the next:

3.2.3.1 [F7] Allow user to delete an existing ride (in case of multiple bookings).

- [R1] User must be already registered and logged in the application.
- [R2] User must be the owner of the ride he wants to delete.
- [R3] User must delete a ride only by choosing the ride in the list, clicking on the button "Info" and then using the popup.
- [R4] Driver must not have accepted the booking.
- [R5] Deleting process is not reversible, all event data will be lost.

3.2.3.2 [F8] Allow user to request a taxi 'booking' ride.

- [R1] User must be already registered and logged in the application.
- [R2] User must complete booking form and confirm the ride.
- [D1] Time, for a delayed booking, must be included between 00.00 and 23.59.

3.2.3.3 [F9] Allow user to consult ride's details when there are more than one booking

- [R1] User must be already registered and logged in the application.
- [R2] User must have booked some rides.
- [R3] User must choose one ride from the list and must click on the button "Info"

3.2.4 [G3] The system allows to share a ride.

Passengers could take advantage from this service by saving money and, with a special consideration for the environment, this service reduce the number of automobiles on the road and the traffic.

3.2.4.1 [F10] Allow user to share a ride.

- [R1] User must be already registered and logged in the application.
- [R2] User must complete booking form and confirm the ride.
- [R3] User must press the button "Sharing" to divide the cost of the ride and to reduce the number of cars on the road.

3.2.5 [G4] The system allows drivers to be organized by a taxi queue. They save time because their rides are optimized by geographic zone.

3.2.5.1 [F11] Allow user to see the state of their booking queue.

- [R1] User must be already registered and logged in the application.
- [R2] Bookings must exist, owner of the booking has correctly created it.

3.2.5.2 [F12] Allow user to see data of every booking they had accepted.

- [R1] User must be already registered and logged in the application.
- [R2] Bookings must exist, owners of every single booking has correctly created it.

3.2.5.3 [F13] After login, application will notify taxi drivers if there is a taxi booking for their taxi

- [R1] User must be already registered and logged in the application.
- [R2] Booking must exist, owner of the booking has correctly created it.
- [R3] Application will notify taxi drivers only when they perform login.

3.3 Jackson/Zave Model

3.3.1 The World and the Machine

“Domain assumptions bridge the gap between requirements and specifications”
(M. Jackson & P. Zave)

This model is to consider the distinction between domain and machine requirements, emphasized for many years by M. Jackson & P. Zave and called “The World & Machine”.

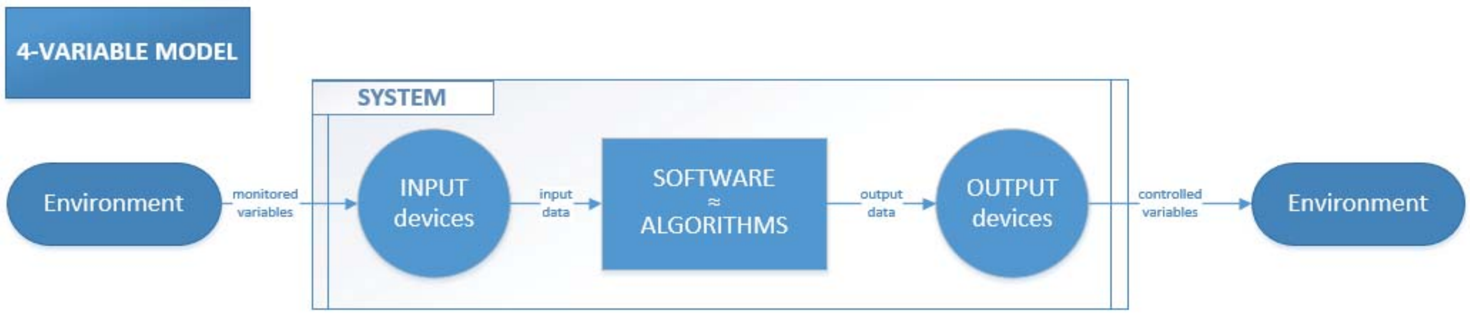
The idea is simple:

- Some requirements elements describe properties of a model of a part of the world, or ‘domain’, in which the system will operate.
- Others describe desired properties of the system, or ‘machine’, that the project wants to build.

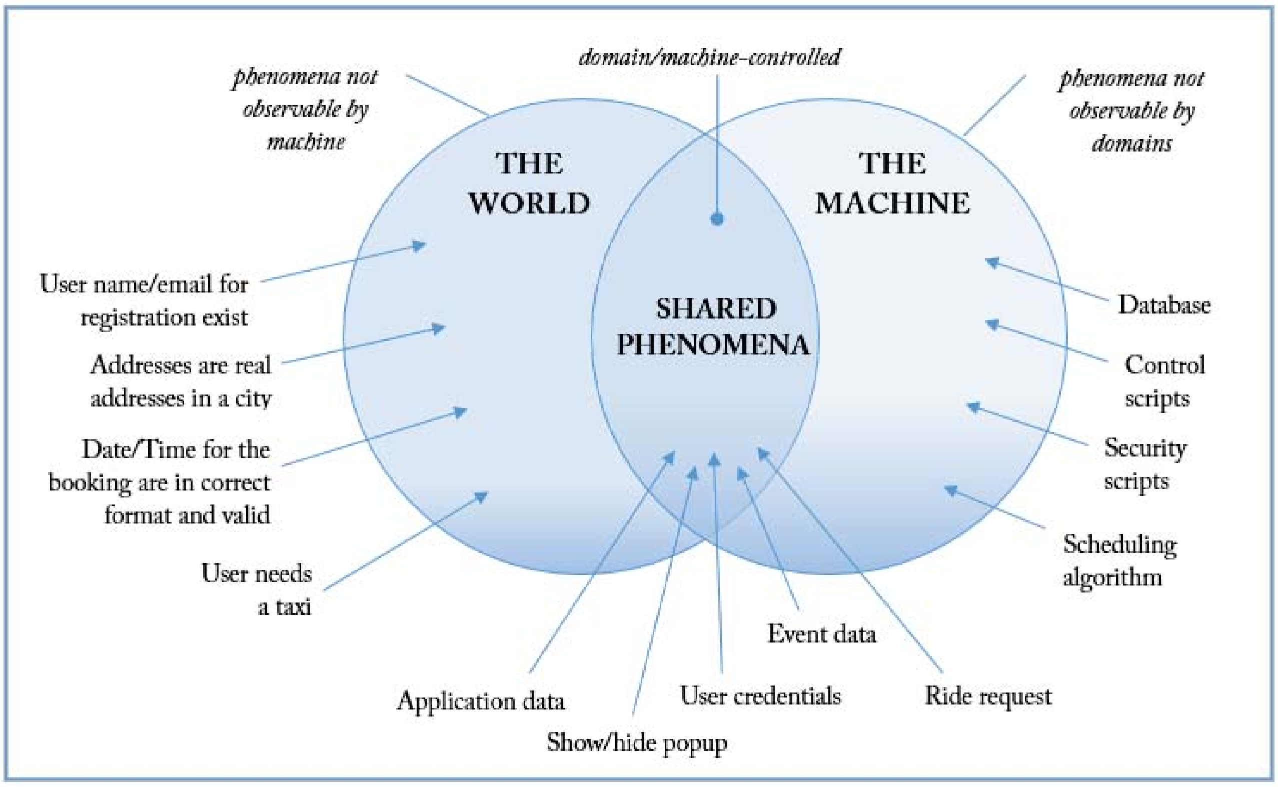
The machine can affect, and be affected by, the environment only because they have some shared phenomena in common. That is, there are some events that are events both in the machine and in the environment; and there are states that are states of both.

This approach let us identify the intersection, or ‘Shared Phenomena’, between the world and the application, that are all world information’s, known or managed directly by the application.

NOTE:
In addition of WM-Model, the 4-variable model let us know how is the behavior of the system in relation of the environment, supposing an input/output data flow.



WM-MODEL


phenomena not observable by machine
domain/machine-controlled
phenomena not observable by domains
THE WORLD
THE MACHINE
SHARED PHENOMENA
User name/email for registration exist
Addresses are real addresses in a city
Date/Time for the booking are in correct format and valid
User needs a taxi
Database
Control scripts
Security scripts
Scheduling algorithm
Event data
Application data
User credentials
Ride request
Show/hide popup

3.4 Scenarios

3.4.1 Scenario 1 »

Bianca has finally finished her afternoon shift at a debt collection agency and she needs a taxi because it's very late for a dinner with some friends.
The restaurant is across town and she wants to save money, because she's a very stingy genovese girl: she decides to enable the sharing service.
Celeste, a Bianca's childhood friend from Genova, lives a block away from Bianca's office and she's late for the dinner too!
Celeste decides to use her new app MyTaxiService, and she finds out the option that allows passengers to share a ride: she could save a lot of money!
Celeste and Bianca have the same destination, so Mr. Rossi with his taxi picks up Bianca and, a few minutes later, Celeste. The ligurian girls arrive on time for the dinner spending less!

3.4.2 Scenario 2 »

Rossella has a job meeting in the company headquarters this afternoon at 2 p.m. o'clock, but has a lunch meeting at 12 a.m.
The restaurant is in the Italian district, but the office that she must reach two hours later is in the financial district: they're too far apart!
The best solution to be punctual is to book a taxi, so she does it while is eating her breakfast.
At 11:30 a.m. she receives an e-mail in which it is written that the meeting has been moved to 15 because her boss has another obligation.She booked the taxi for 1:45 p.m., so she is in time to delete the reservation using the popup, run some errands and make another request for the 14:45 p.m..

3.4.3 Scenario 3 »

Mr. Verdi has just parked his taxi after a stressful ride: the traffic was heavy and the customer was very unlikable, so he decide to take a break with a cappuccino and a croissant.
Soon as he bite the croissant a pop up warns him that he must pick up a passenger near there.He's too nervous and needs at least 10 minutes to cool off, so he selects the option to decline the ride and continues to drink his cappuccino.

3.4.4 Scenario 4 »

Viola and her colleague Rosa have had a little chat on lunch break.
Rosa was very satisfied because she tried MyTaxiService and it has been a very efficient service, so invites Viola to use the same one.

Viola is not very good with technology, but Rosa said that MyTaxiService is simple to use and the registration is a fast process, so she decide to try.
The form that she must compile is intuitive and the fields require information easy to find: her name, her surname, her age, her address, her email, a password, her fiscal code and her PayPal account. Initially she has inserted a not valid e-mail address, so a pop up warns Viola about the error, but after her correction the registration has been a success!

3.4.5 Scenario 5 »

Verdiana must reach her boyfriend's house after dinner. He has moved in her same city just two days ago, but his apartment is not near the metro station and Verdiana is afraid of being out alone in the night. The fastest and safest way to arrive is a taxi, so Verdiana compiles the MyTaxiService's form with her start address and her destination. At this point a pop up warns her about an inexistent destination address: she has inserts the old address of her boyfriend! She corrects the mistake and is be able to book the taxi.

3.4.6 Scenario 6 »

Mr. Bianchi and Mr. Neri must meet themselves for a job lunch. Mr. Bianchi is at home, so use his PC to book a taxi with MyTaxiService: he inserts his address and the destination's address and books the ride. Mr. Neri is in the city center, so use his mobile to book the ride: MyTaxiService's system suggests him the start address because it can find his position with the GPS data. He's very satisfied because he hates to write with the touchscreen keyboard and a simple touch has been sufficient to indicate his position.

3.4.7 Scenario 7 »

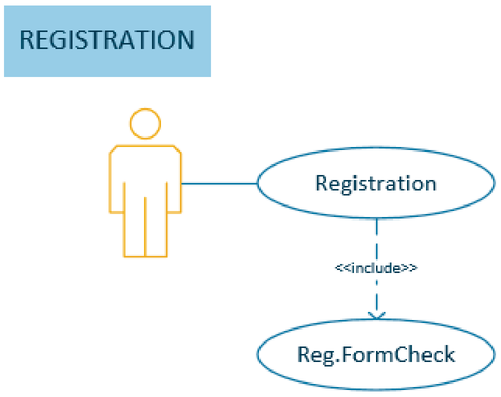
Mr. Marrone has scheduled several appointments for tomorrow.
He's a very precise, organized and a little bit hypochondriac man, so he begins to think about all displacements to do: he must be at 9 a.m at the urologist's office, at 11 a.m at the oculist's office, at 3 p.m at the cardiologist's office, at 5 p.m. at the dermatologist's office and at 19 p.m., finally, at home. Mr. Marrone is afraid to catch cold, because he could take a serious sore throat! He need a safe and fast way to reach all doctors: MyTaxiService is right for him because this service allows users to book more than one taxi and then it will be possible to control all reservations. When the app is open all delayed booking are organized by a list and users can see all rides' details pressing the button "info".

3.5 UML Models

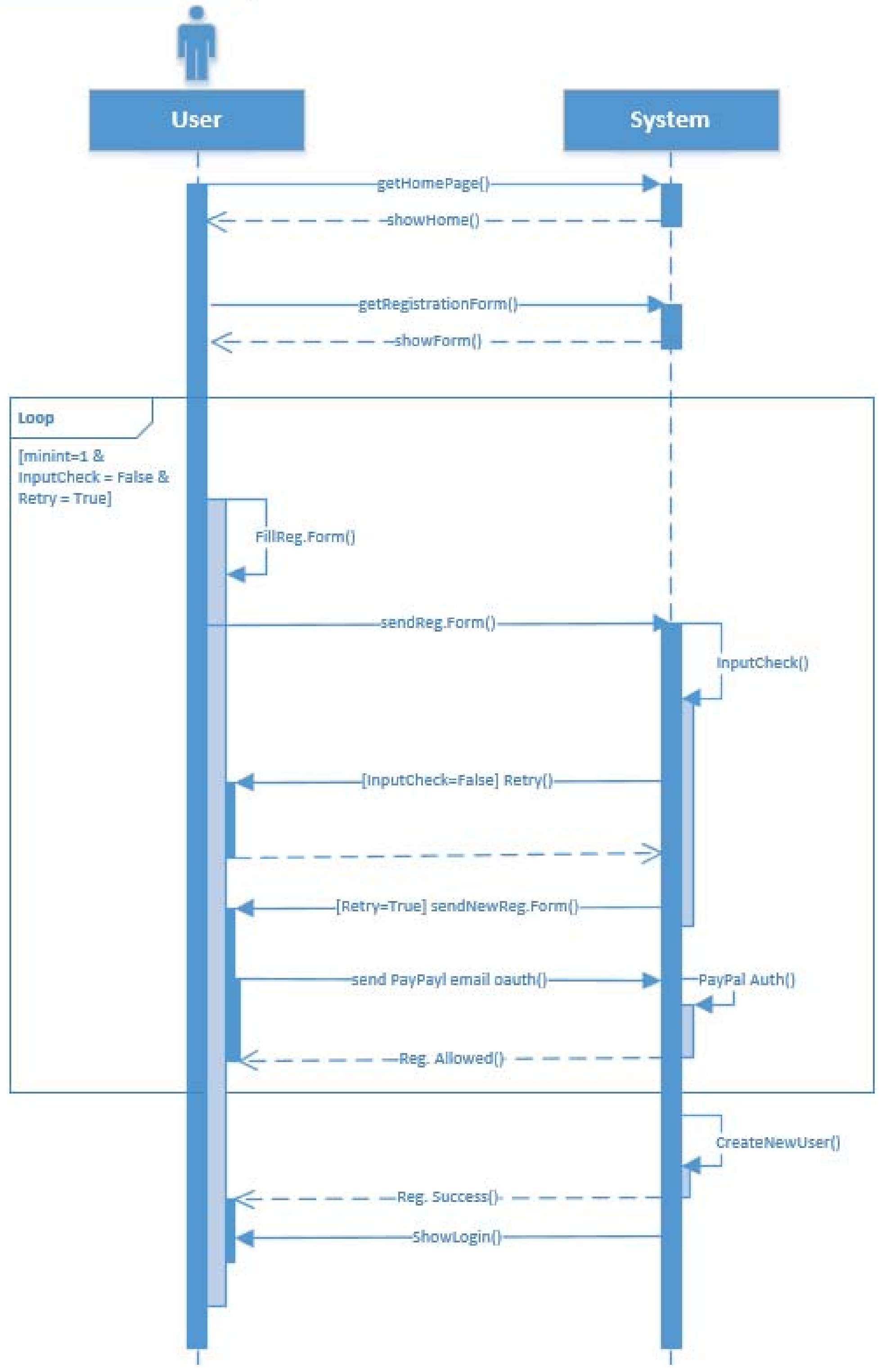
3.5.1 Use Case

3.5.1.1 User registration

Actor	Visitor
Functionality	[F1]
Input Condition	NULL
Event Flow	1. Visitor on the login page clicks on "register" button to reach the registration form. 2. Visitor fills in all mandatory fields. 3. Visitor clicks on "confirm" button. 4. The application will save the credentials in the DB. 5. The database makes check for constraints. 6. A popup inform the user for the registration output.
Output Condition	Visitor successfully ends registration process and become a User. Now he/she is redirected to the login page and can log in to the application using his/her credential.
Exception	1. The visitor is already a user. 2. One or more mandatory fields are not valid. 3. Email chosen is already associated to another user. 4. PayPal account check is failed (account not valid) For every notification or exception a popup is shown over other elements of the page or app.

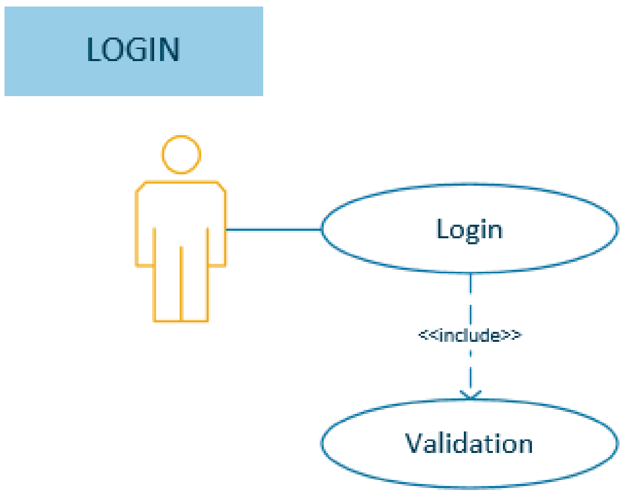


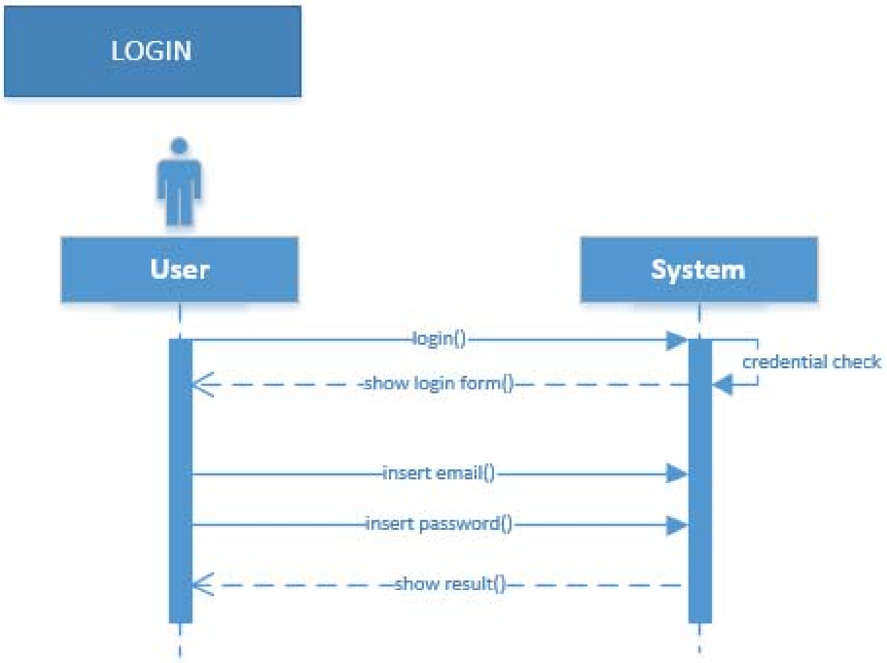
REGISTRATION



3.5.1.2 User login

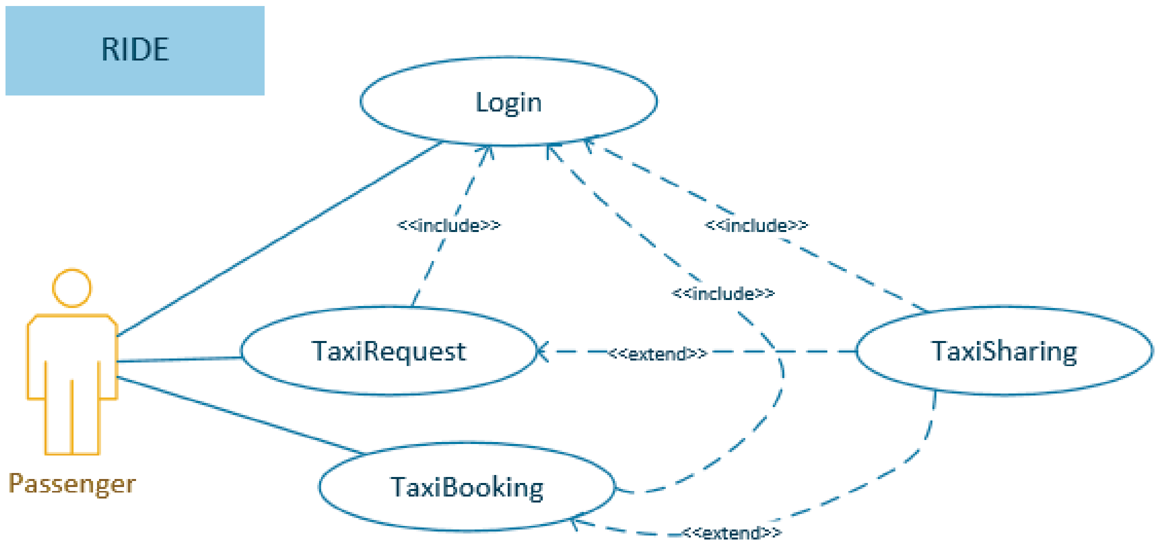
Actor	Visitor, Registered User
Functionality	[F2]
Input Condition	Visitor is registered into the system.
Event Flow	1. The site/app shows the login page. 2. Visitor complete the form inserting a correct email and password.
Output Condition	The site/app verifies the credential of visitor and if these are correct, shows a menu with all services.
Exception	If email and/or password are incorrect the system notifies the visitor with a popup and show a redirect button the login page (to try again).

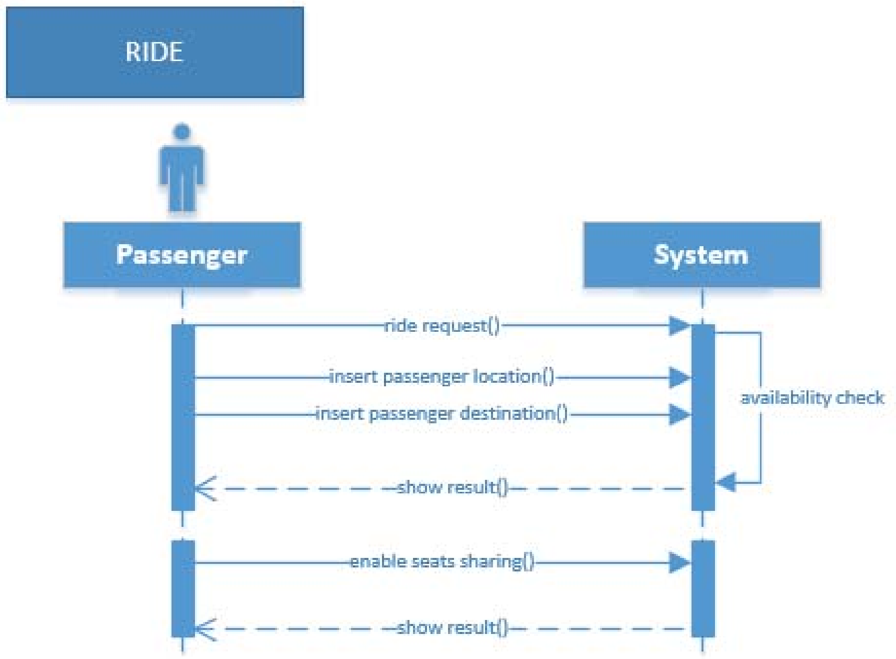




3.5.1.3 User requests for a taxi ride

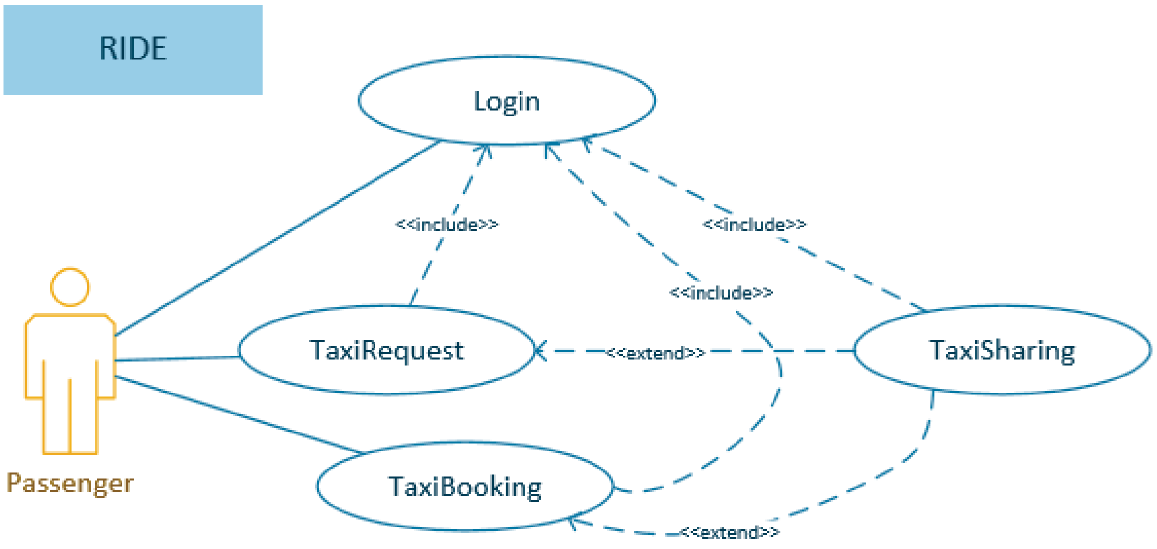
Actor	Registered User
Goal	[G1] [G3]
Input Condition	Registered User is already logged in into the site/app.
Event Flow	1. Registered User navigate to the simple ride request area. 2. User insert current address and destination address. 3. User activate or not the 'sharing' function. 4. User clicks on 'Request Taxi'.
Output Condition	The site/app shows a popup with the result of the request.
Exception	If the User wants to cancel the request process: -can go back in the browser -in the popup can push the X button

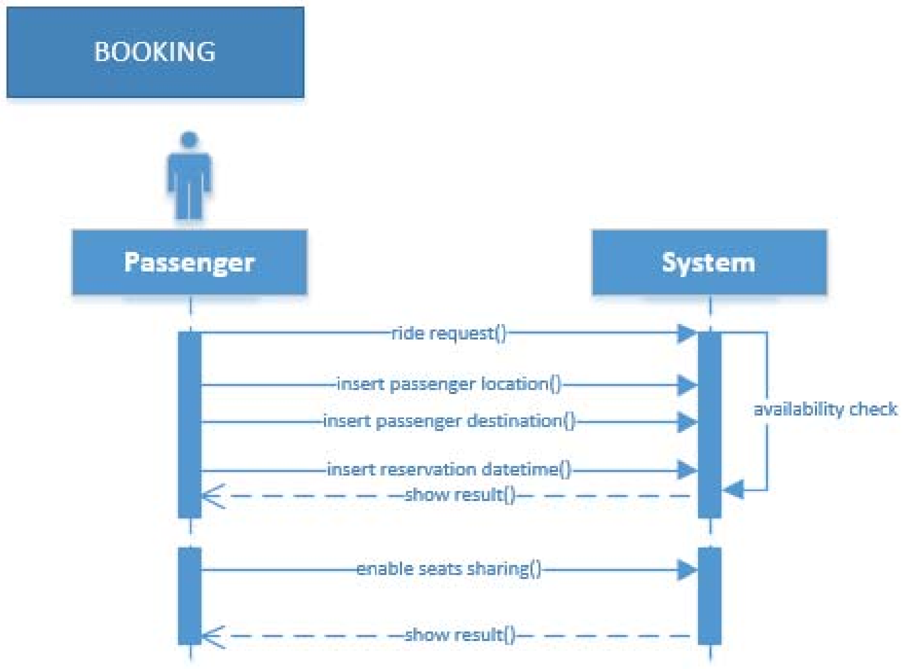

RIDE
Login
<<include>>
<<include>>
<<include>>
TaxiRequest
<<extend>>
TaxiSharing
Passenger
TaxiBooking
<<extend>>


RIDE
Passenger
System
ride request()
insert passenger location()
insert passenger destination()
availability check
show result()
enable seats sharing()
show result()

3.5.1.4 User requests for a 'booking' taxi ride

Actor	Registered User
Goal	[G2] [G3]
Input Condition	Registered User is already logged in into the site/app.
Event Flow	1. Registered User navigate to the 'booking' ride request area. 2. User insert current address and destination address. 3. User select the date and the time for which he/she wants to have the taxi. 4. User activate or not the 'sharing' function. 5. User click on 'Reserve Taxi'.
Output Condition	The site/app shows a page with the list of the requested bookings.
Exception	If the User wants to cancel the reservation process: -can go back in the browser -in the list can open the 'info' popup and can push the X button

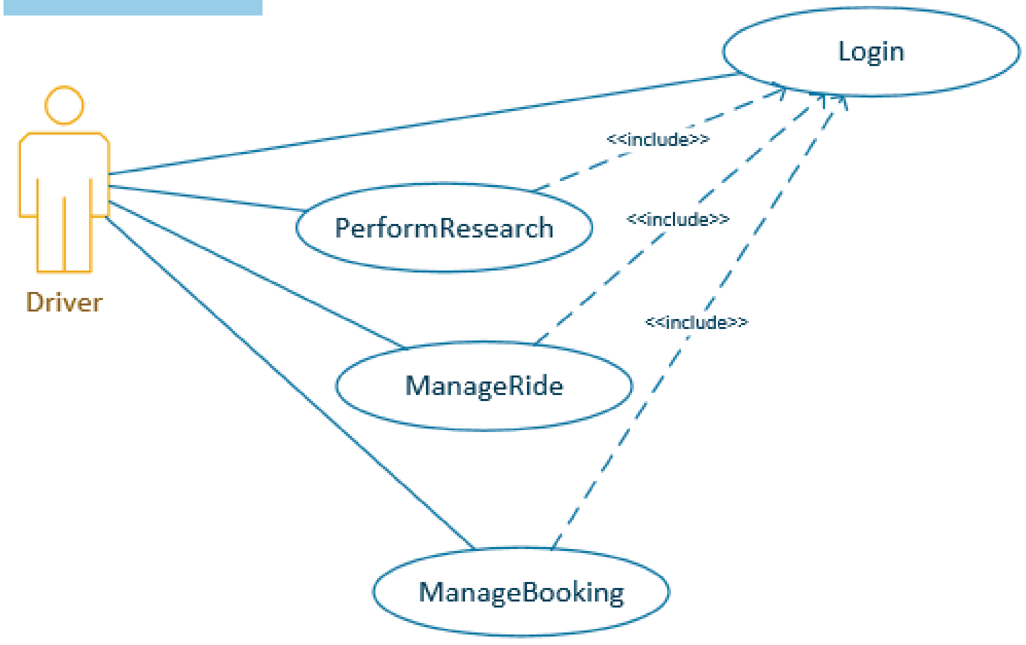




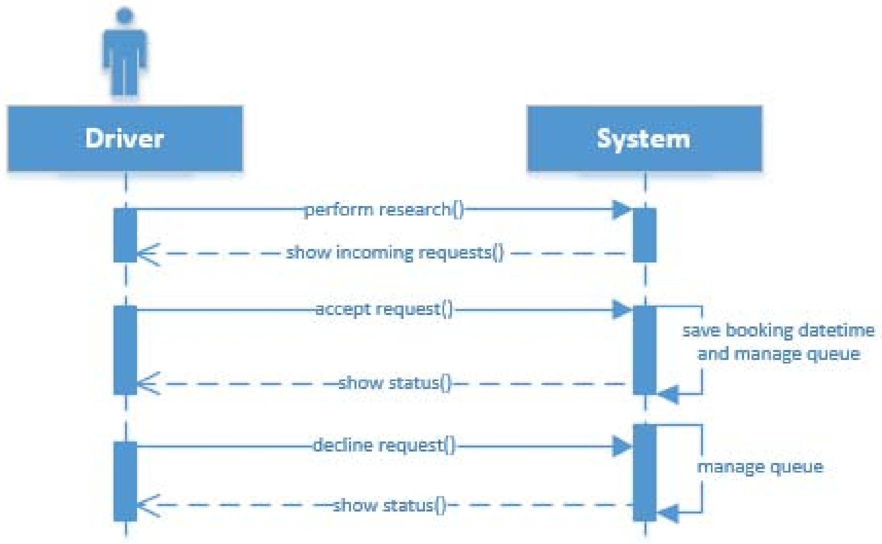
3.5.1.5 Driver management of new rides

Actor	Registered Driver
Goal	[G4]
Input Condition	Driver is already logged in into the app.
Event Flow	1. Driver navigate to the private 'Driver Area'. 2. Driver scroll the page and find a good ride for him/her. 3. Driver select a ride with the tick √ button. 4. Driver can hide from the view a ride, using the close X button.
Output Condition	The site/app shows a popup with the confirmation for the taking charge of the ride.
Exception	A Driver cannot refuse a confirmed ride. If a Driver wants to cancel the charge for a ride: -has to call the call-center that will contact the user at his registered email address

DRIVER

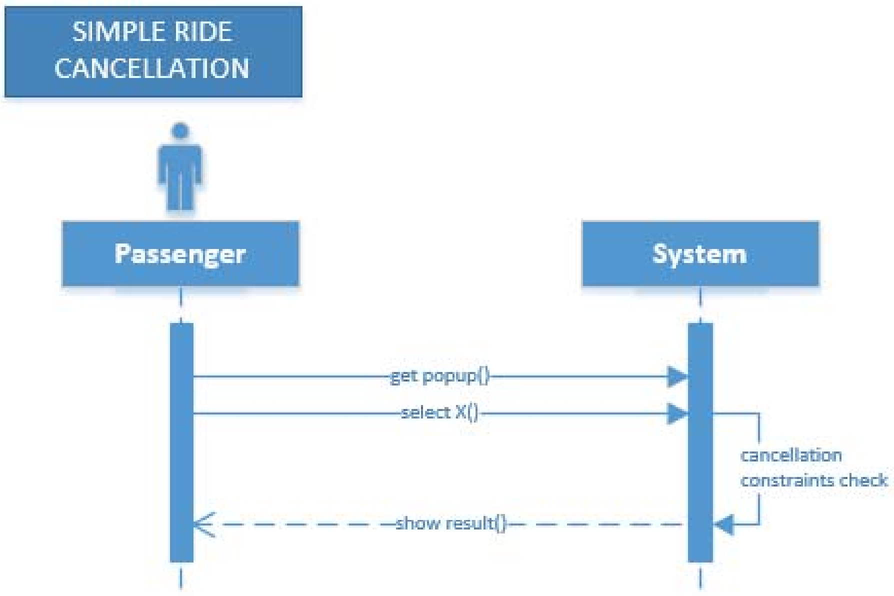


DRIVER



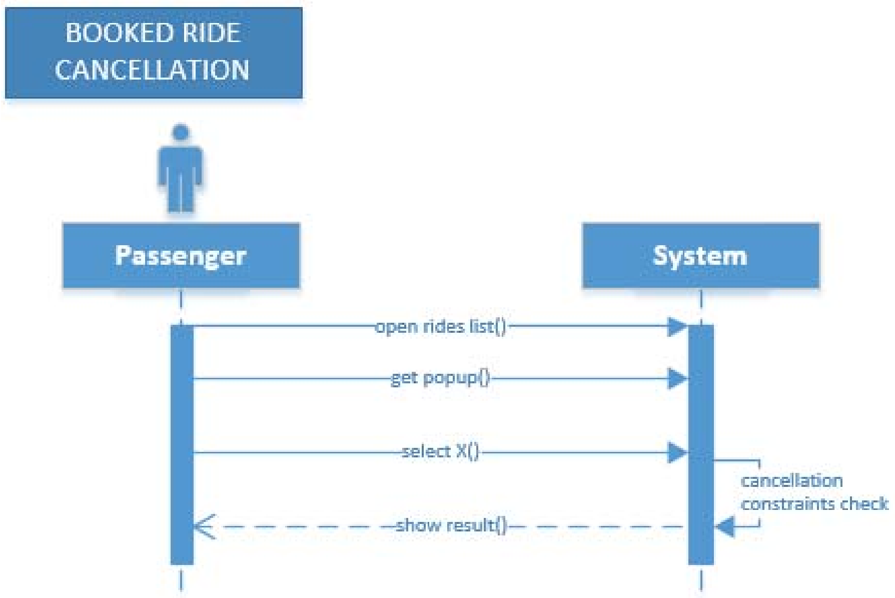
3.5.1.6 User cancellation of a simple ride

Actor	Registered User
Goal	[G1]
Functionality	[F4]
Input Condition	Registered User is already logged in into the site/app. Registered User has already requested a simple ride.
Event Flow	1. Registered User open the popup with details of the just requested ride. 2. User clicks on 'X'.
Output Condition	The site/app shows a page with result of the operation (depending by rides deletion constraints).
Exception	If the User wants to cancel the deletion process: -can go back in the browser

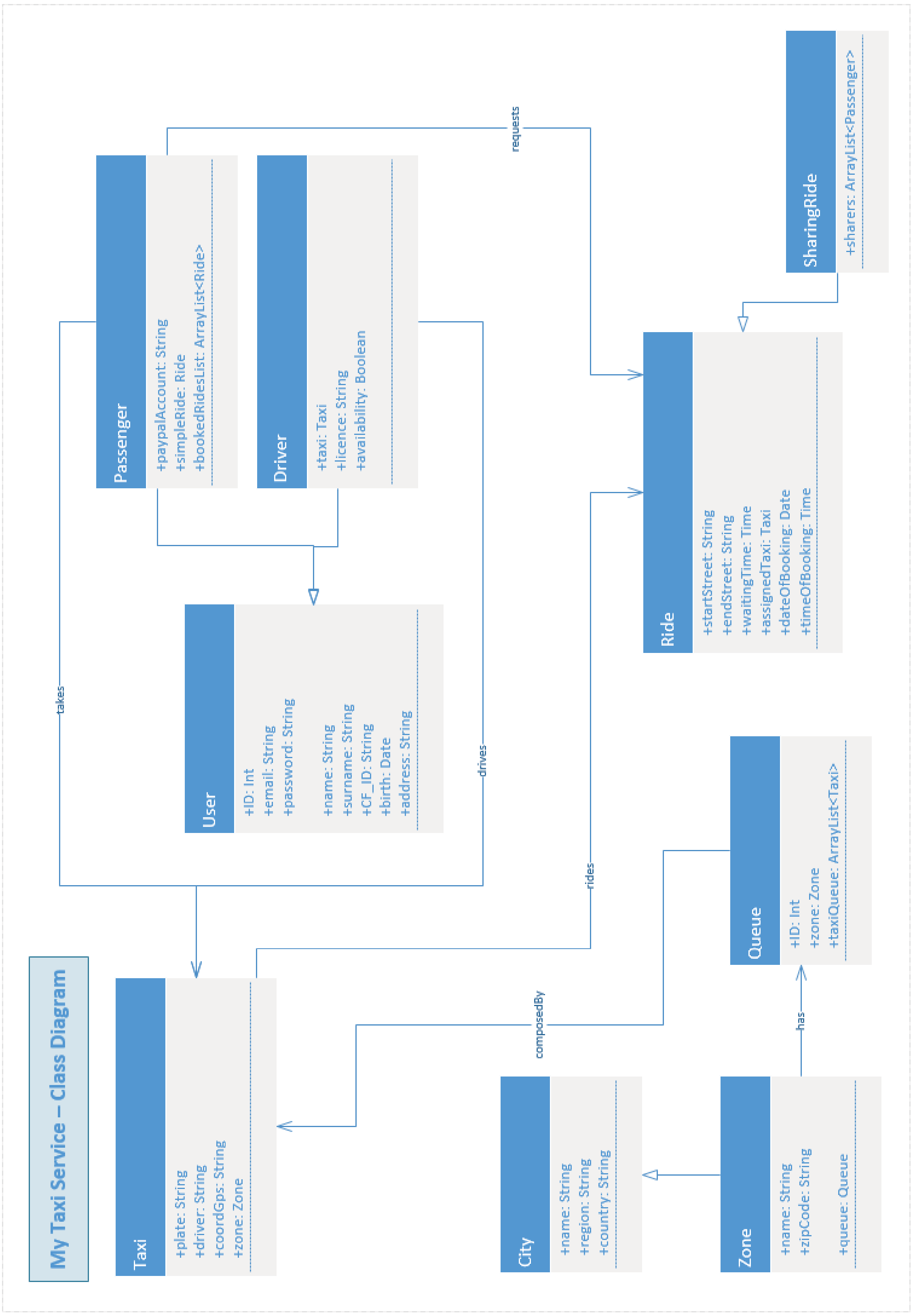


3.5.1.7 User cancellation of a booked ride

Actor	Registered User
Goal	[G2]
Functionality	[F7]
Input Condition	Registered User is already logged in into the site/app. Registered User has already requested at least a ride (booking).
Event Flow	1. Registered User navigate to the 'rides list' area. 2. User scroll and select the ride that wants to delete. (a popup with details of the selected ride is shown) 3. User clicks on 'X'.
Output Condition	The site/app shows a page with result of the operation (depending by rides deletion constraints).
Exception	If the User wants to cancel the deletion process: -can go back in the browser

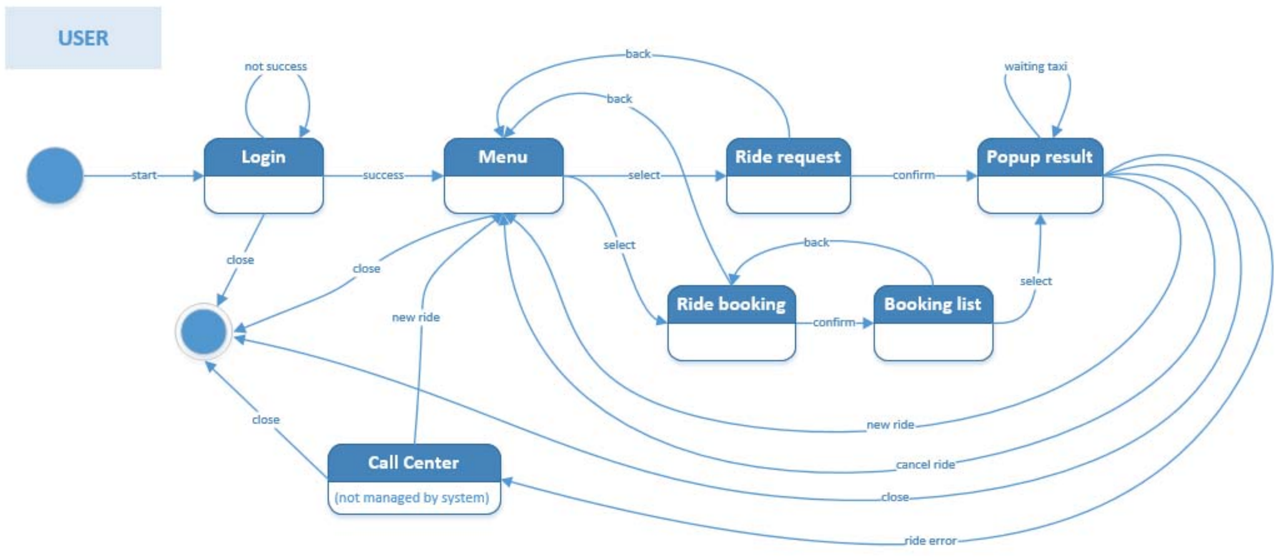


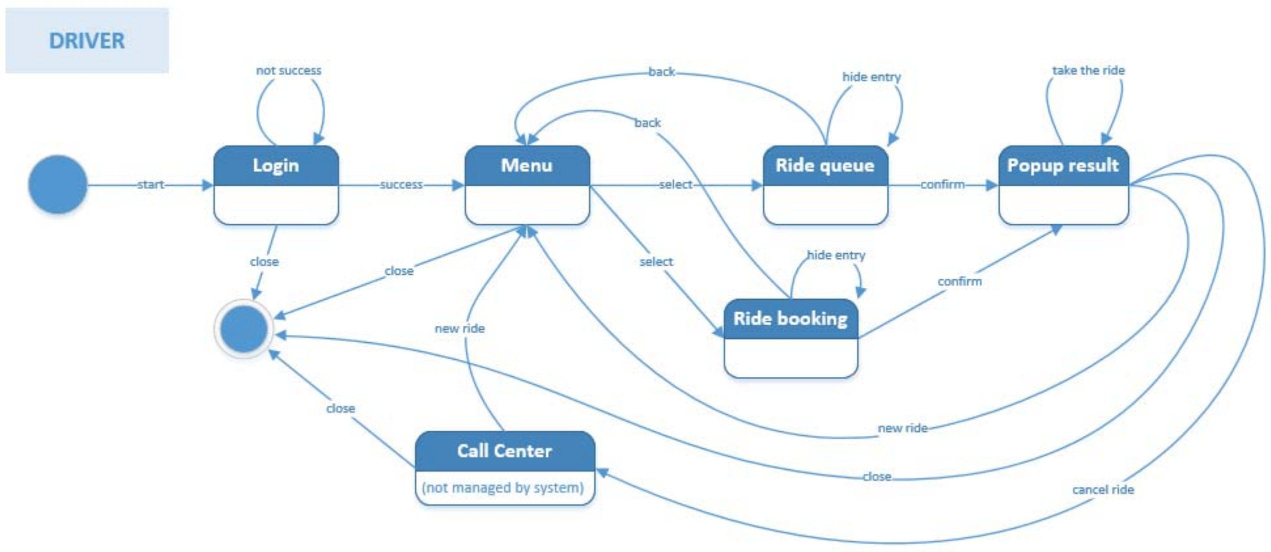
3.5.2 Class Diagram

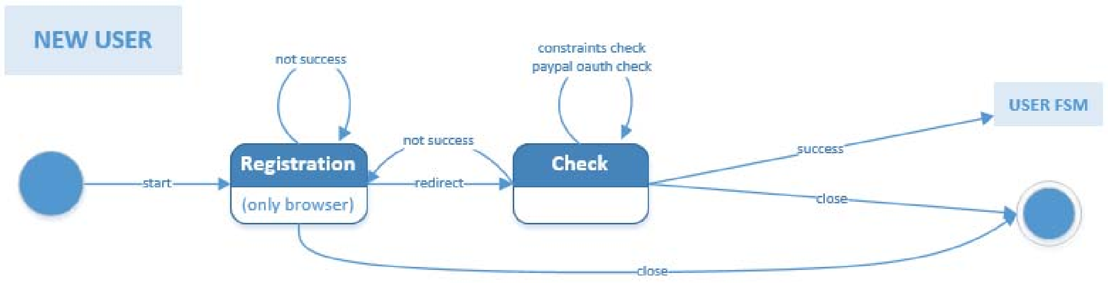


3.5.3 Finite State Machine Diagrams

The following state-machine diagram would give a simplified vision of entire application.







3.6 Non Functional Requirements

3.6.1 Performance Requirements

Performance must be acceptable to guarantee a good grade of usability. We assume the response time of the system is close to zero, so the performance are essentially bounded by users internet connection.

3.6.2 Design Constraints

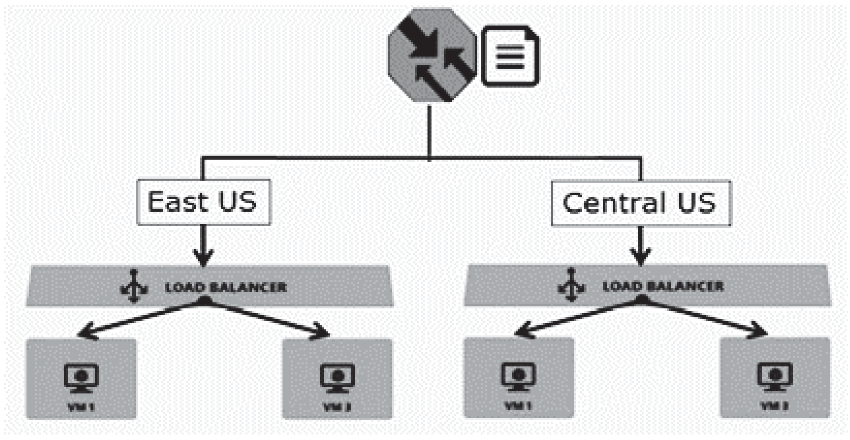
The application will be developed using Java EE so it will inherit all language's constraints.

3.6.3 Software System Attributes

3.6.3.1 Availability

As all distributed sites/apps, MyTaxiService will be accessible online anytime.
The backend uses a variety of services:

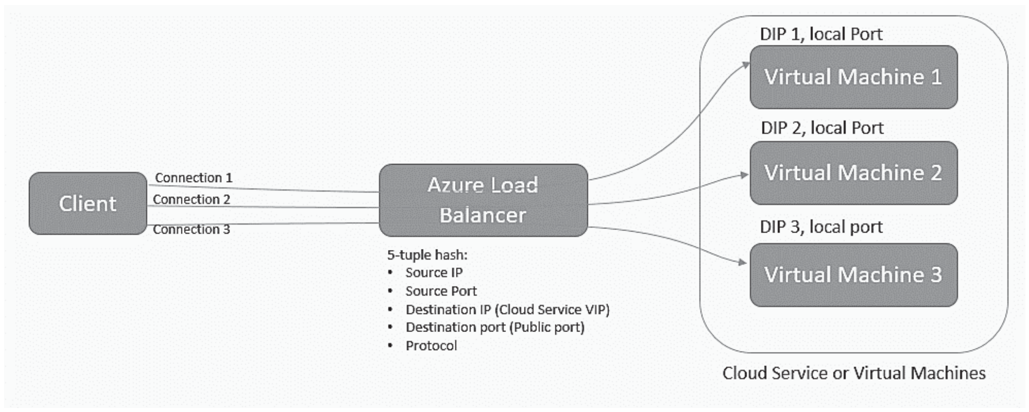
- ✓ hosting machine
- ✓ webserver
- ✓ database server
- ✓ load balancer
- ✓ traffic manager
- ✓ IP failover
- ✓ CDN



[traffic management example]

To guarantee a future scalability of the service, all system could be hosted into cloud platform like Microsoft Azure.

This solution gives more scalability to performance required by the system and could reduce the cost for dedicated server, maintaining an high level of performance especially in case of full load with a lot of requests per second.



[load balancing example]

3.6.3.2 Maintainability

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

3.6.3.3 Portability

The application could be used on any SO which supports JVM and DBMS.

3.6.4 Security

3.6.4.1 External Interface Side

- **Login**
  MyTaxiService application embeds a login authentication to protect the information of users. The connection with the server is encrypted.

- **Passwords**
  Password of user is saved using hashing mechanism like SHA-128.
  This system does not actually check the password length, so could be developed a system that require an 8-character password with number, letter and special character.
  Password also is static, user is not involved in password changing so system will ask to user to change frequently the password for example every 3 months.
  To have a more secure system, it should be implemented a login system with a captcha code, to prevent botnet attack.

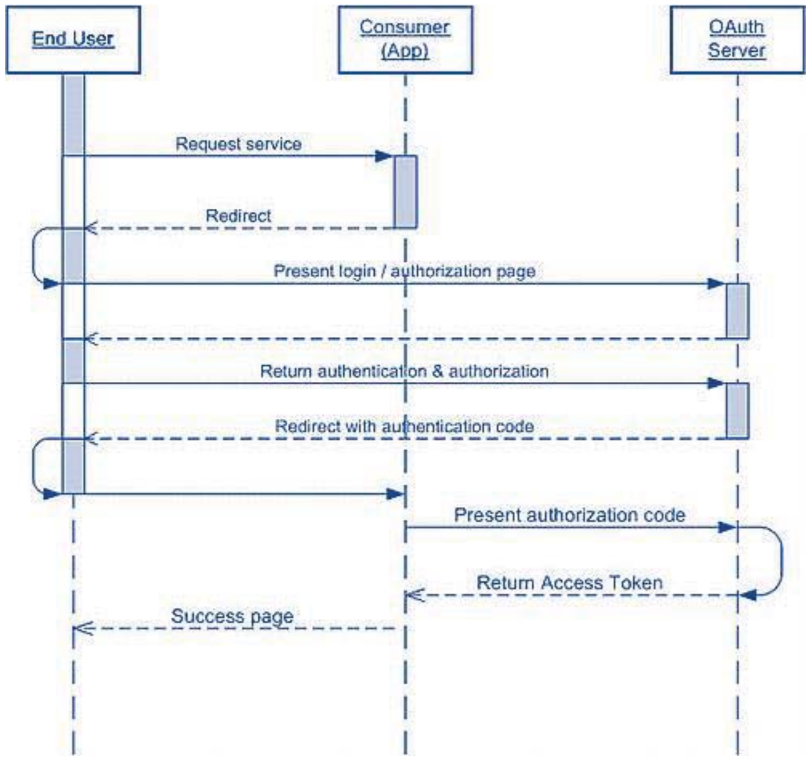
- **Multifactor authentication**

Could also be implemented an authentication system with a mix of these technologies:

✓ **Two-factor** authentication with a code sent by email or SMS to the user

✓ **OTP** one-time password, composed by a card static password list owned by user or a dynamic embedded password generator (ex. using a code-generator app). The system asks for random code on that card or an entire code generated.

✓ **OAUTH** login system: to authenticate a user by other public authentication protocols like Facebook, Twitter, Windows Live etc.. (ex. 'Login with Facebook' Button)
Follows the Active Diagram as example of OAUTH events flow:


End User
Consumer (App)
OAuth Server
Request service
Redirect
Present login / authorization page
Return authentication & authorization
Redirect with authentication code
Present authorization code
Return Access Token
Success page

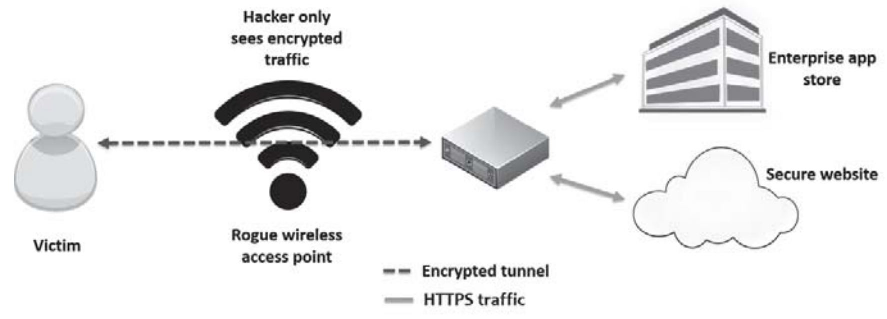
3.6.4.2 Application Side

On the application side could be implemented a security backend to protect the application against multiple attacks

- **SQL Injection**
  Could be implemented a filtering system to all form.
  Malicious user can fill the form with an ad-hoc SQL code to have access to information who normally cannot have access.

- **Buffer overflow**
  Buffer overflows can be triggered by inputs that are designed to execute code, or alter the way the program operates. This may result in erratic program behavior, including memory access errors, incorrect results, a crash, or a breach of system security.
  A simple way to avoid the BOF attack is to constantly check the integrity of the requests, and in case of insanely big memory request, drop it.

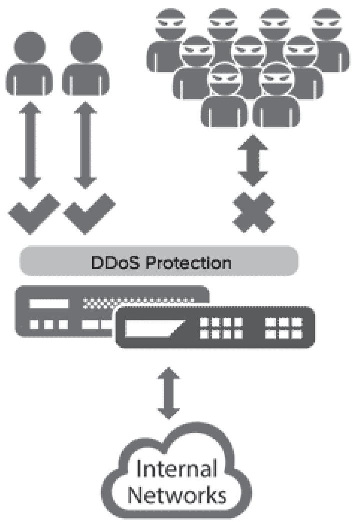
- **MITM Attack**
  Implement the https connection instead of http is a best effort method to guarantee communication confidentiality and integrity and also mutual authentication.
  Moreover the SSL protocol (in common use with a Digital Certificate signed by a Certification Authority(CA)) is resistant to the Man In the Middle attack.



3.6.4.3 Server Side

The server side architecture could be implemented dividing strongly the data from application. In particular, a real threat is the DoS attack.

- **DDoS Attack**

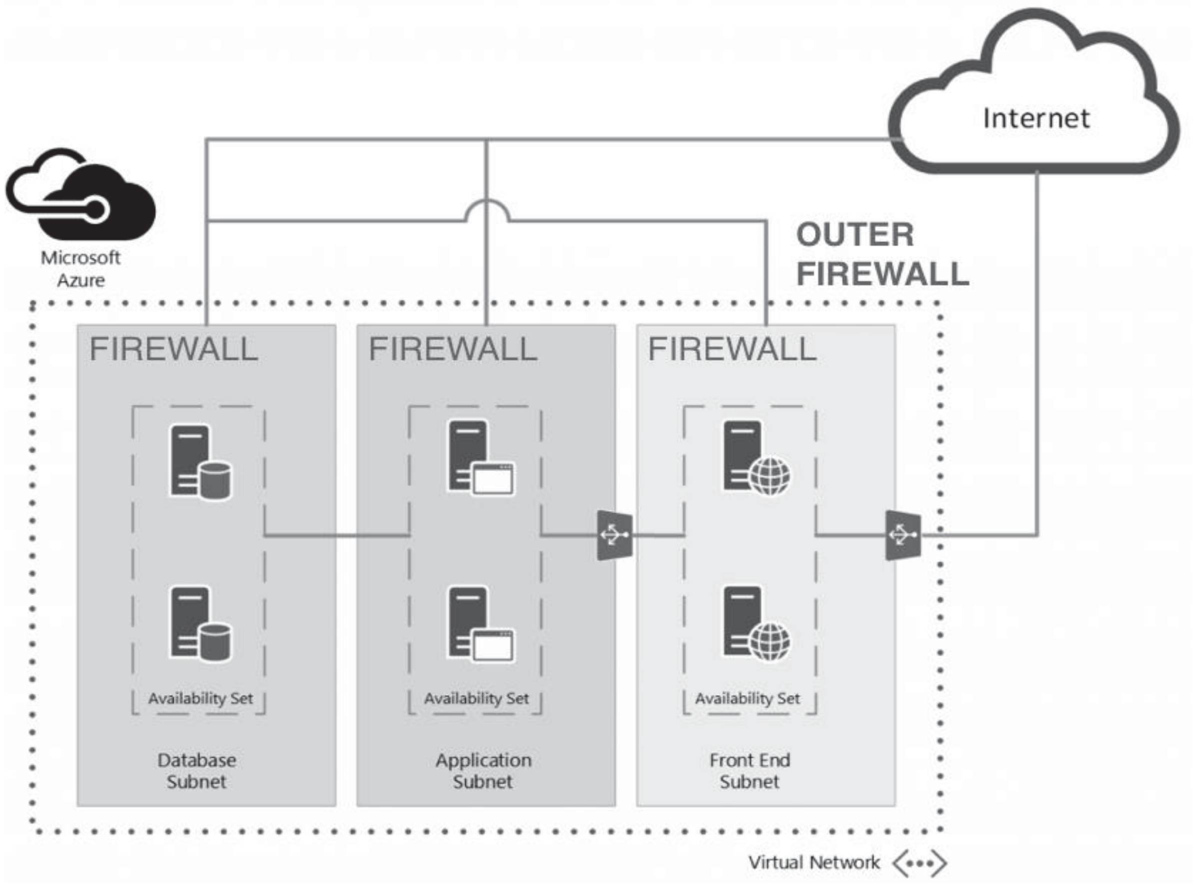


A denial-of-service attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service.

Defensive responses to denial-of-service attacks typically involve the use of a combination of attack detection, traffic classification and response tools, aiming to block traffic that they identify as illegitimate and allow traffic that they identify as legitimate.
This can be done with a use of the cloud based services like traffic managers and virtual routers/virtual switches like the MS Azure ones.

- **DNS Spoofing**
  DNS spoofing (or DNS cache poisoning) is a computer hacking attack, whereby data is introduced into a Domain Name System (DNS) resolver's cache, causing the name server to return an incorrect IP address, diverting traffic to the attacker's computer (or any other computer).
  A protection from this attack could come by services like 'Secure DNS' (*DNSSEC*) that uses cryptographic digital signatures signed with a trusted public key certificate to determine the authenticity of data.

- **Firewall protection**
  An idea of possible secure infrastructure is well represented by the following picture. Application Server is separated from database and from the web server. All zones are divided by firewall. Access to this zone is restricted and forbidden to not authorized user.


Internet
Microsoft Azure
OUTER FIREWALL
FIREWALL
FIREWALL
FIREWALL
Availability Set
Availability Set
Availability Set
Database Subnet
Application Subnet
Front End Subnet
Virtual Network

4. Appendix

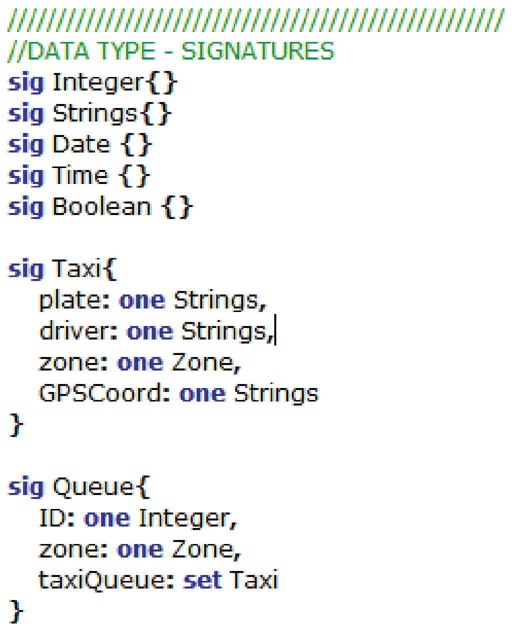
4.1 Alloy

The complete alloy file (*.als*) could be find on GitHub code repository.
The following alloy model presented is created using the class diagram.

We try to divide the code in part dividing signature from fact, assert and predicates. In the last part there are the generated word.

4.1.1 Data Type – Signatures

This is the alloy code of data type:



```
/////////////////////////////////////////////////////
//DATA TYPE - SIGNATURES
sig Integer{}
sig Strings{}
sig Date {}
sig Time {}
sig Boolean {}

sig Taxi{
   plate: one Strings,
   driver: one Strings,
   zone: one Zone,
   GPSCoord: one Strings
}

sig Queue{
   ID: one Integer,
   zone: one Zone,
   taxiQueue: set Taxi
}
```

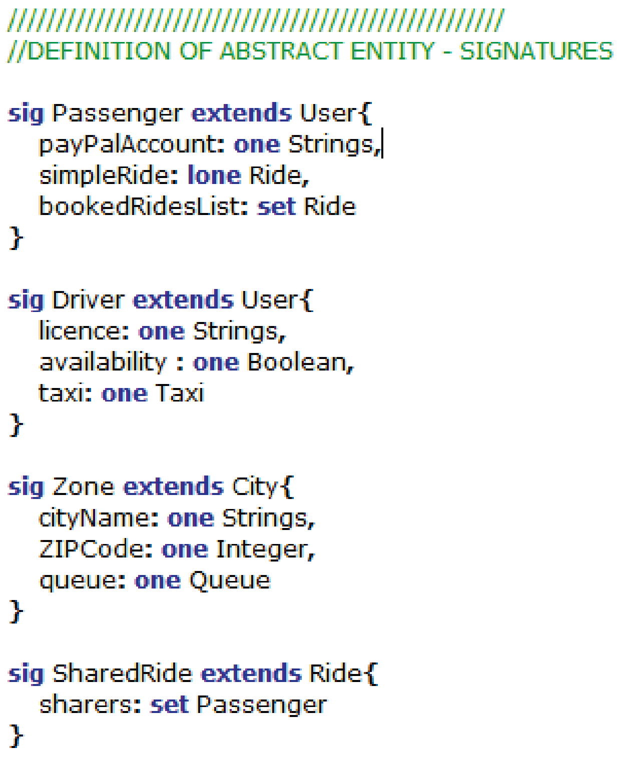
4.1.2 Abstract Entity - Signatures

This is the alloy code of abstract entity:

```
//////////////////////////////////////////////////
//ABSTRACT ENTITY - SIGNATURES

abstract sig User{
   ID: one Integer,
   eMail: one Strings,
   password: one Strings,
   name: one Strings,
   surname: one Strings,
   CF_ID: one Strings,
   birth: one Date,
   address: one Strings
}

abstract sig Ride{
   startStreet: one Strings,
   stopStreet: one Strings,
   waitingTime: one Time,
   assignedTaxi: one Taxi,
   dateOfBooking: one Date,
   timeOfBooking: one Time
}

abstract sig City{
   name: one Strings,
   region: one Strings,
   country: one Strings
}
```

4.1.3 Implementation of Abstract Entity – Signatures

This is the alloy code of the definition of abstract entity:



```
/////////////////////////////////////////////////////
//DEFINITION OF ABSTRACT ENTITY - SIGNATURES

sig Passenger extends User{
   payPalAccount: one Strings,
   simpleRide: lone Ride,
   bookedRidesList: set Ride
}

sig Driver extends User{
   licence: one Strings,
   availability : one Boolean,
   taxi: one Taxi
}

sig Zone extends City{
   cityName: one Strings,
   ZIPCode: one Integer,
   queue: one Queue
}

sig SharedRide extends Ride{
   sharers: set Passenger
}
```

4.1.4 Facts

This is the alloy code of the facts, that defines the global constraints of the class:

```
/////////////////////////////////////////////////////////////////////
//FACTS

// Impose a non empty User
fact noEmptyUser{
   all u: User | (#u.ID=1)
            and (#u.eMail=1)
            and (#u.password=1)
            and (#u.name=1)
            and (#u.surname=1)
            and (#u.CF_ID=1)
            and (#u.birth=1)
            and (#u.address=1)
}

// Impose a non empty Passenger
fact noEmptyPassenger{
   all p: Passenger | (#p.payPalAccount=1)
}

// Impose a non empty Driver
fact noEmptyDriver{
   all d: Driver | (#d.licence=1) and (#d.availability=1) and (#d.taxi=1)
}

// Impose a non empty Taxi
fact noEmptyTaxi{
   all t: Taxi | (#t.plate=1) and (#t.driver=1) and (#t.zone=1) and (#t.GPSCoord=1)
}

// Impose a non empty City
fact noEmptyCity{
   all c: City | (#c.name=1) and (#c.region=1) and (#c.country=1)
}

// Impose a non empty Zone
fact noEmptyZone{
   all z: Zone | (#z.name=1) and (#z.ZIPCode=1) and (#z.queue=1)
}

// Impose a non empty Queue
fact noEmptyQueue{
   all q: Queue | (#q.ID=1) and (#q.zone=1) and (#q.taxiQueue=1)
}

// Impose a non empty Ride
fact noEmptyRide{
   all r: Ride |  (#r.startStreet=1)
                  and (#r.stopStreet=1)
                  and (#r.waitingTime=1)
                  and (#r.assignedTaxi=1)
                  and (#dateOfBooking=1)
                  and (#timeOfBooking=1)
}
```

```
// Impose a non empty shared ride
fact noEmptySharedRide{
   all r: SharedRide | (#r.sharers>=0 and #r.sharers<=4)
}

// Impose no duplicate mail and no duplicate ID
fact noDuplicateUser{
   no disj u1, u2: User | (u1.eMail = u2.eMail) and (u1.ID = u2.ID)
}

// Impose no duplicate payPal account
fact noDuplicatePassenger{
   no disj p1, p2: Passenger | (p1.payPalAccount = p2.payPalAccount)
}

// Impose no duplicate licence
fact noDuplicateDriver{
   no disj d1,d2: Driver | (d1.licence = d2.licence)
}

// Impose no duplicate plate
fact noDuplicateTaxi{
   no disj t1, t2: Taxi | (t1.plate = t2.plate)
}

// Impose no duplicate ZIPCode and Queue
fact noDuplicateZone{
   no disj z1,z2: Zone | (z1.ZIPCode = z2.ZIPCode) and (z1.queue = z2.queue)
}

// Impose no duplicate ID, zone and taxiQueue
fact noDuplicateQueue{
   no disj q1,q2: Queue | (q1.ID = q2.ID) and (q1.zone = q2.zone) and (q1.taxiQueue = q2.taxiQueue)
}
// Impose that cannot be deleted a not simple booked ride
fact noInexistentSimpleRideDeletion{
   all r: Ride, p: Passenger |
                    (
                      (
                        (#r.dateOfBooking = 0) or (#r.timeOfBooking = 0)
                      )
                      and
                      (
                        (#r.startStreet = 0) or (#r.stopStreet = 0)
                      )
                      implies (p.simpleRide not in r)
                    )
}

// Impose that cannot be deleted a not booked ride
fact noInexistentBookedRideDeletion{
   all r: Ride, p: Passenger |
                    (
                      (
                        (#r.dateOfBooking = 0) or (#r.timeOfBooking = 0)
                      )
                      and
                      (
                        (#r.startStreet = 0) or (#r.stopStreet = 0)
                      )
                      implies (p.bookedRidesList not in r)
                    )
}

// Impose that only registered user can book a ride
fact noUnregisteredUserMakeRequests{
   all p: Passenger | (#p.ID = 0) implies (#p.simpleRide = 0 and #p.bookedRidesList = 0)
}
```

4.1.5 Predicates – Functions

This is the alloy code for predicates and functions (predicates could result true or false):

```
///////////////////////////////////////////////////////////////////////
//PREDICATES - FUNCTIONS

//Show a world in which there are following rules
pred show(){

   //rides
   #Queue = 1
   #Ride >= 1
   #Taxi >= 1

   //cities
   #City = 1
   #Zone = 1

   //users
   #Passenger = 2
   #SharedRide = 1

   //superclass show constraints
   #Integer = 1
   #Boolean = 0
   #Date = 1
   #Time = 1
}
run show for 2

//Show a world in which there is a booked ride
pred showBookedRide(){
   #startStreet = 1
   #stopStreet = 1
   #waitingTime = 1
   #assignedTaxi = 1
   #dateOfBooking = 1
   #timeOfBooking = 1
}
run showBookedRide for 1

//Show a world in which there is a driver confirming ride
pred showDriverAcceptingRide(){
   all r: Ride, d: Driver | (r.assignedTaxi = d.taxi)
}
run showDriverAcceptingRide for 3

//Predicate for a new ride request
pred rideRequest(r:Ride, p1,p2: Passenger){
   r not in p1.simpleRide implies p2.simpleRide = p1.simpleRide+r
}
run rideRequest for 2
```

```
//Predicate for a new ride booking request
pred rideBookingRequest(r:Ride, p1,p2: Passenger){
   r not in p1.bookedRidesList implies p2.bookedRidesList = p1.bookedRidesList+r
}
run rideBookingRequest for 1

//Predicate for a ride deletion
pred deleteRideRequest(r:Ride, p1,p2: Passenger){
   r in p1.simpleRide implies p2.simpleRide = p1.simpleRide-r
}
run deleteRideRequest for 1

//Predicate for a booked ride deletion
pred deleteBookingRideRequest(r:Ride, p1,p2: Passenger){
   r in p1.bookedRidesList implies p2.bookedRidesList = p1.bookedRidesList-r
}
run deleteBookingRideRequest for 1
```

4.1.6 Assertions

This is the last part of the alloy code that is used to verify the consistency of the model:

```
//////////////////////////////////////////////////////////////////////
//ASSERTIONS

//Check that there cannot exist a ride without a taxi
assert noRideWithoutTaxi{
   no r: Ride | (no t: Taxi | r.assignedTaxi = t)
}
check noRideWithoutTaxi

//Check that there cannot exist a ride without a user
assert noRideWithoutUser{
   all p: Passenger |(#p.payPalAccount=0) implies (#p.simpleRide=0) and (#p.bookedRidesList =0)
}
check noRideWithoutUser
```

(continue..)

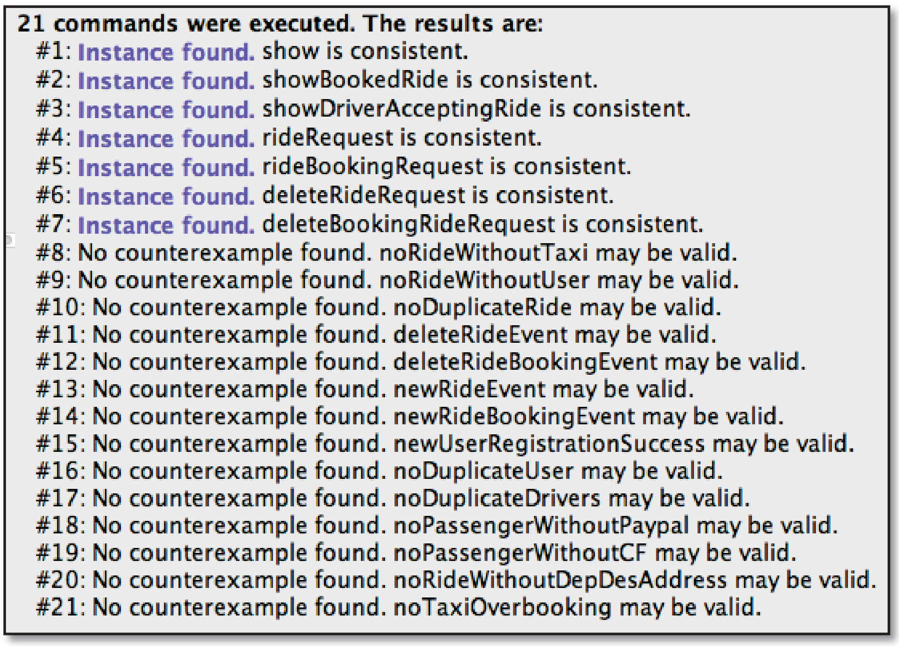
```
//Check that there cannot exist a duplicate ride
assert noDuplicateRide{

   no disj r1, r2: SharedRide, p1, p2: Passenger |

      //two equal rides
      (
         r1.startStreet = r2.startStreet and r1.stopStreet = r2.stopStreet
         and
         r1.dateOfBooking = r2.dateOfBooking and r1.timeOfBooking = r2.timeOfBooking
      )
      and
      //with same sharers
      (
         p1 in r1.sharers
         and
         p2 in r1.sharers
         and
         r1.sharers = r2.sharers
      )
      and
      //and each user has same ride type
      (
         p1.simpleRide = p2.simpleRide
         and
         p1.bookedRidesList = p2.bookedRidesList
      )
}
check noDuplicateRide

//Check the deletion for simple ride event predicate
assert deleteRideEvent{
   all r1: Ride, p1, p2: Passenger |
      (
         (
            p1.simpleRide = r1
            and
            p2.simpleRide = r1
            and
            deleteRideRequest[r1,p1,p2]
         )
         implies
         (
            p2.simpleRide not in r1
         )
      )
}
check deleteRideEvent

//Check the deletion for booking ride event predicate
assert deleteRideBookingEvent{
   all r1: Ride, p1, p2: Passenger |
      (
         (
            p1.bookedRidesList = r1
            and
            p2.bookedRidesList = r1
            and
            deleteBookingRideRequest[r1,p1,p2]
         )
         implies
         (
            p2.bookedRidesList not in r1
         )
      )
}
check deleteRideBookingEvent
```

```
//Check the simple ride request predicate
assert newRideEvent{
   all r1: Ride, p1, p2: Passenger |
      (
         (
            p1.simpleRide = r1
            and
            p2.simpleRide = r1
            and
            deleteRideRequest[r1,p1,p2]
         )
         implies
         (
            p2.simpleRide = r1
         )
      )
}
check newRideEvent

//Check the ride booking request predicate
assert newRideBookingEvent{
   all r1: Ride, p1, p2: Passenger |
      (
         (
            p1.bookedRidesList = r1
            and
            p2.bookedRidesList = r1
            and
            rideBookingRequest[r1,p1,p2]
         )
         implies
         (
            p2.bookedRidesList = r1
         )
      )
}
check newRideBookingEvent
//Check that a new user is added
assert newUserRegistrationSuccess{
   all p: Passenger |
      #p.ID=1
      implies
      #p.payPalAccount=1
}
check newUserRegistrationSuccess

//Check that there aren't two equal users
assert noDuplicateUser{
   no disj u1, u2 : User |
            (u1.ID=u2.ID) and (u1.eMail=u2.eMail) and
            (u1.password=u2.password) and (u1.name=u2.name) and
            (u1.surname=u2.surname) and (u1.CF_ID=u2.CF_ID) and
            (u1.birth=u2.birth) and (u1.address=u2.address)
}
check noDuplicateUser
```

```
//Check that there aren't two equal drivers
assert noDuplicateDrivers{
   no disj d1,d2: Driver |  (d1.licence=d2.licence) and (d1.availability=d2.availability) and (d1.taxi=d2.taxi)
}
check noDuplicateDrivers

//Check that cannot exist a passenger without a paypal account
assert noPassengerWithoutPaypal{
   no p: Passenger | (#p.payPalAccount =0)
}
check noPassengerWithoutPaypal

//Check that cannot exist a passenger without a CF_ID
assert noPassengerWithoutCF{
   no p: Passenger | (#p.CF_ID=0)
}
check noPassengerWithoutCF

//Check that cannot exist a ride without a destination or departure address
assert noRideWithoutDepDesAddress{
   no r: Ride | (#r.startStreet =0) and (#r.stopStreet =0)
}
check noRideWithoutDepDesAddress

//Check that in a shared ride cannot be reserved more than 4 seats
assert noTaxiOverbooking{
   no s: SharedRide | #s.sharers > 4
}
check noTaxiOverbooking
```

4.1.7 Output result

This is the output of the Alloy Analyzer Software that shows the consistence of the model in all its parts:

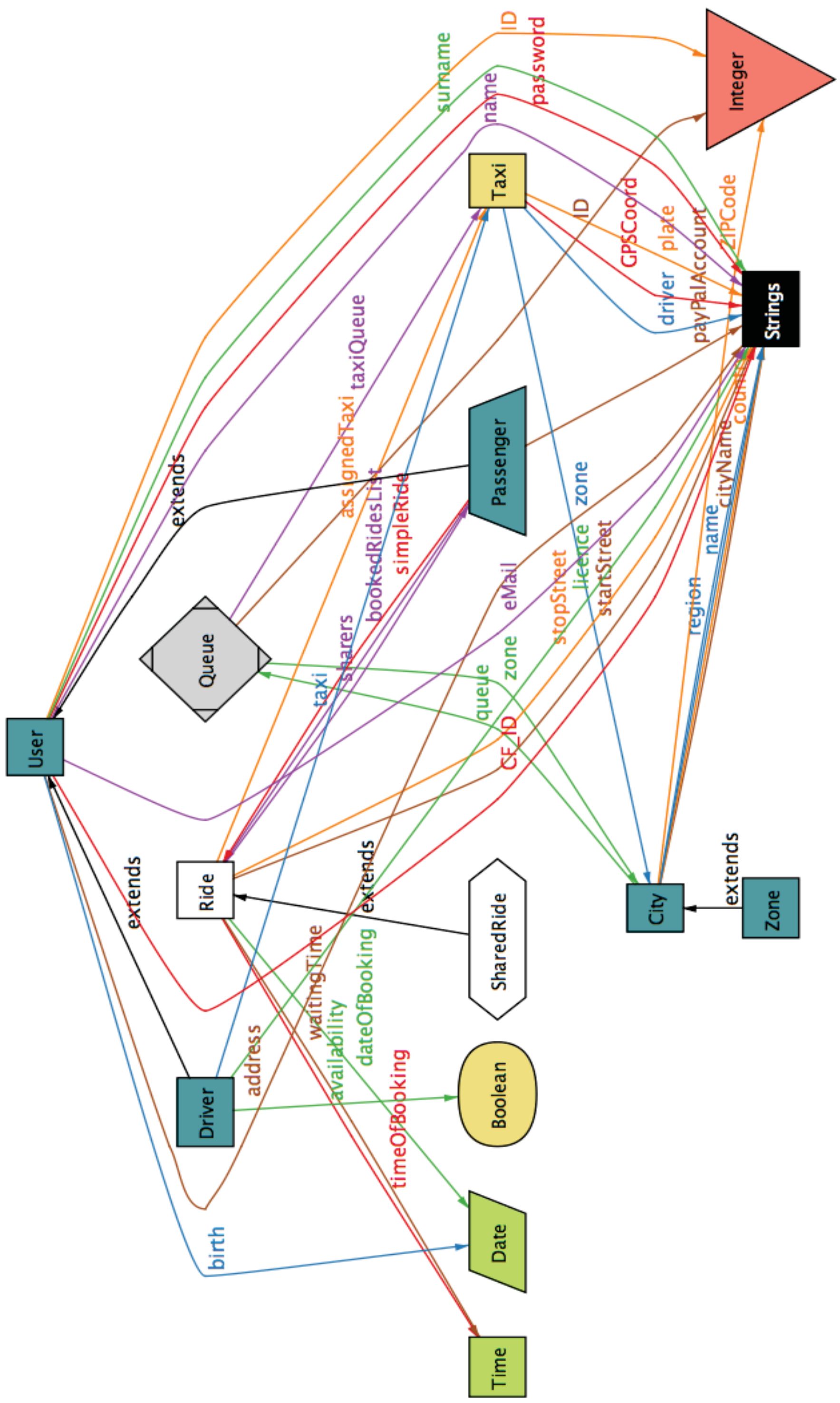


```
21 commands were executed. The results are:
  #1: Instance found. show is consistent.
  #2: Instance found. showBookedRide is consistent.
  #3: Instance found. showDriverAcceptingRide is consistent.
  #4: Instance found. rideRequest is consistent.
  #5: Instance found. rideBookingRequest is consistent.
  #6: Instance found. deleteRideRequest is consistent.
  #7: Instance found. deleteBookingRideRequest is consistent.
  #8: No counterexample found. noRideWithoutTaxi may be valid.
  #9: No counterexample found. noRideWithoutUser may be valid.
  #10: No counterexample found. noDuplicateRide may be valid.
  #11: No counterexample found. deleteRideEvent may be valid.
  #12: No counterexample found. deleteRideBookingEvent may be valid.
  #13: No counterexample found. newRideEvent may be valid.
  #14: No counterexample found. newRideBookingEvent may be valid.
  #15: No counterexample found. newUserRegistrationSuccess may be valid.
  #16: No counterexample found. noDuplicateUser may be valid.
  #17: No counterexample found. noDuplicateDrivers may be valid.
  #18: No counterexample found. noPassengerWithoutPaypal may be valid.
  #19: No counterexample found. noPassengerWithoutCF may be valid.
  #20: No counterexample found. noRideWithoutDepDesAddress may be valid.
  #21: No counterexample found. noTaxiOverbooking may be valid.
```

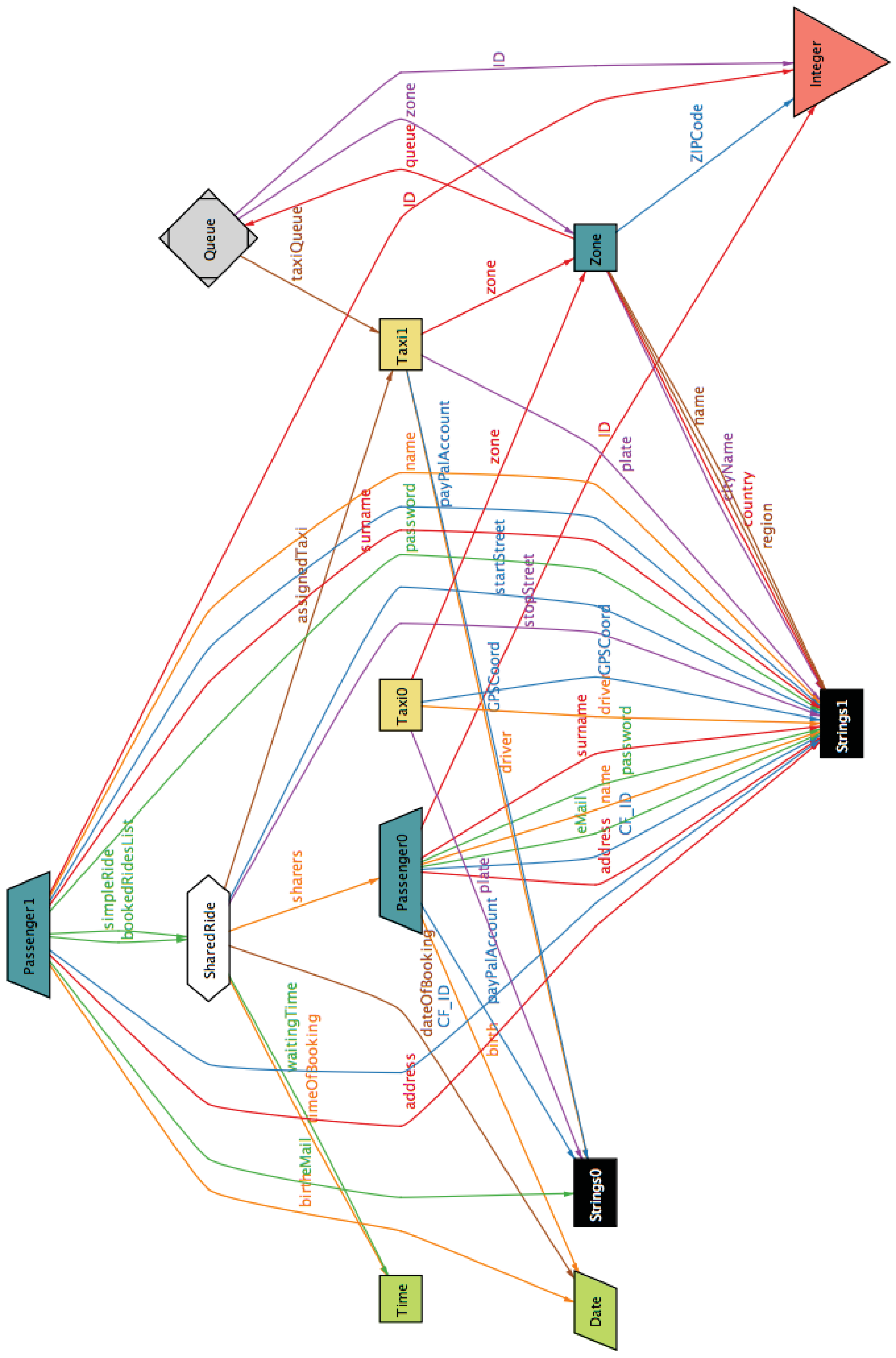
4.1.8 Generated worlds

Follows the screenshot of the generated worlds made using the Alloy Verification software.

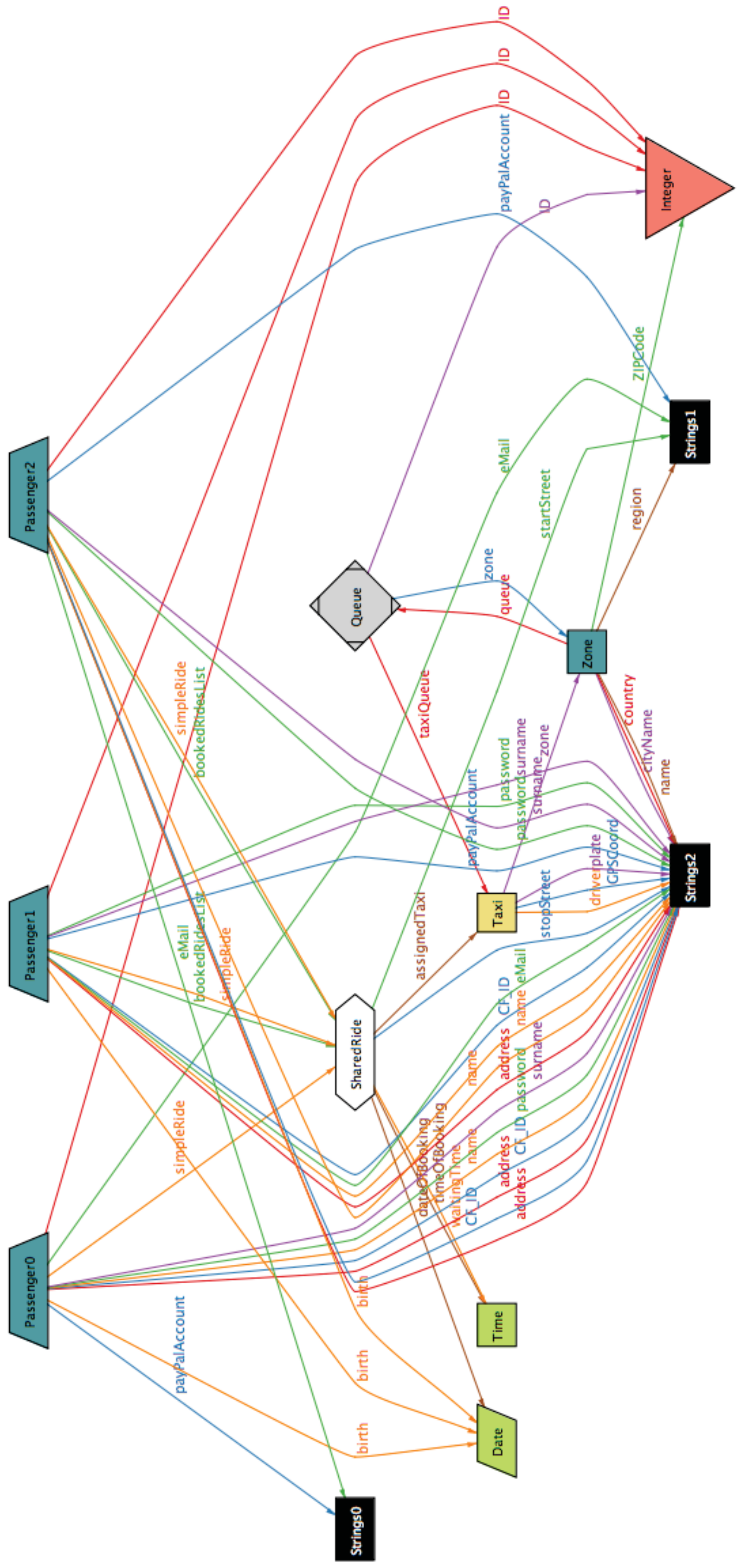
4.1.8.1 Diagram 1 – Metamodel



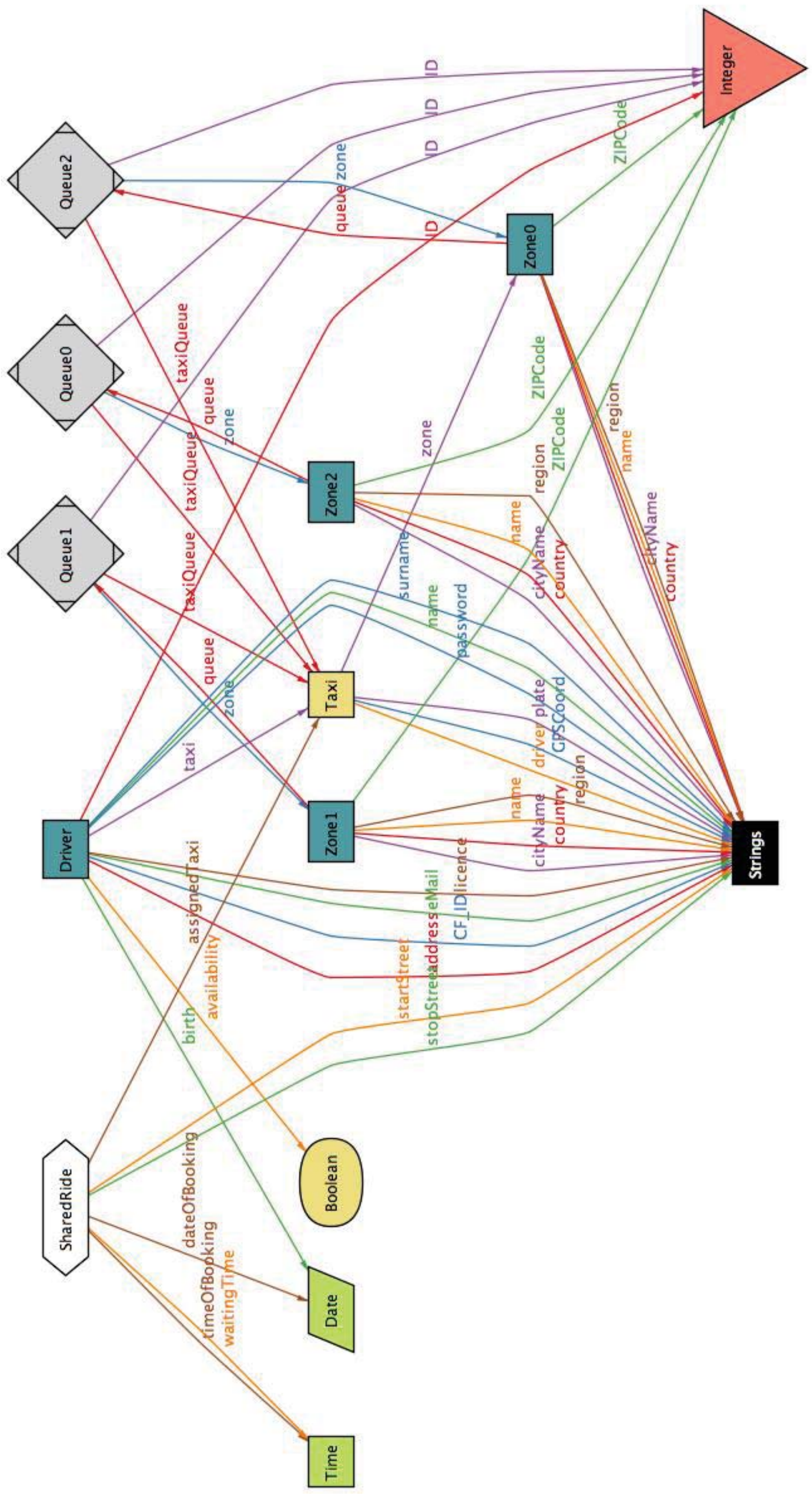
4.1.8.2 Diagram 2 – Predicate Show() for 2 cases (2 passengers / 2 taxies)



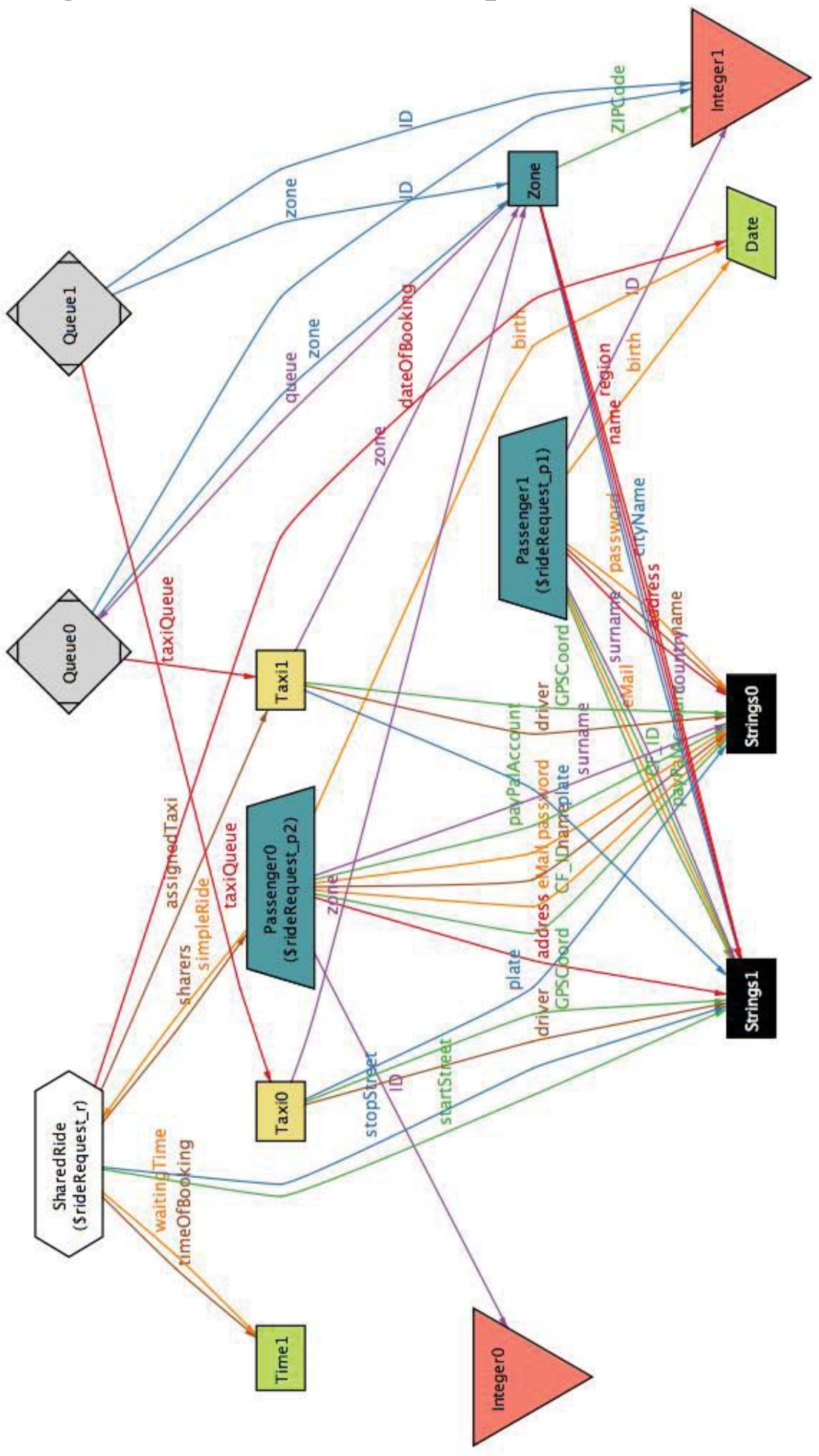
4.1.8.3 Diagram 3 – Predicate Show() for 3 cases (3 passengers / 1 taxi)



4.1.8.4 Diagram 4 – Predicate ShowDriverAcceptingRide() (1 driver in 3 queues)



4.1.8.5 Diagram 5 – Predicate RideRequest() for 2 cases



NOTE
We decided to not attach further worlds generated by more cases because the resulting model will be more and more complex and difficult to read

4.2 Software and tool used

- Microsoft Word 2016 (http://www.office.com/): to redact and to format this document; the 2016 version allow to share the document between users and co-work in a real-time streaming.

- Microsoft Visio Professional (http://www.office.com/): to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Machine Diagrams

- Alloy Analyzer (http://alloy.mit.edu/alloy/): to prove the consistency of our model.

- Gimp (http://www.gimp.org/): to modelling some image.

- Balsamiq Mockups (http://balsamiq.com/products/mockups/): to create mockups.

4.3 Hours of works

Here is the time spent for redact this document:

[sum of hours spent by team's members]
+4h (19/10)
+3h (20/10)
+3h (21/10)
+4h (22/10)
+5h (25/10)
+8h (26/10)
+4h (27/10)
+8h (28/10)
+10h (29/10)
+8h (30/10)
+8h (2/11)
+16h (3/11)
+20h (4/11)
+10h (5/11)
+1h (6/11)

TOTAL ~ 112 hours
- Leonardo Turchi: ~ 56hours
- Sara Pisani: ~ 56hours

5. Revision

This is 2.0 version of RASD that contains update of the document after the entire development of the application.
In the follow are listed the difference between the previous version:

5.1 Modified Assumptions

[...]

5.2 Modified Goal and Functionality

[...]

5.3 Modified Functional Requirements

[...]

5.4 Modified Scenarios

[...]

5.5 Modified Diagrams

[...]

(ONLY FOR FUTURE USE)