

myTaxiService

-

**Requirements Analysis and Specification
Document**

Davide Cremona (matr. 852365), Simone Deola (matr. 788181)

November 6, 2015

Contents

1	Introduction	5
1.1	Purpose	5
1.2	Actual System	5
1.3	Scope	5
1.4	Actors	6
1.5	Goals	6
1.6	Definitions, Acronyms, Abbreviations	7
1.6.1	Definitions	7
1.6.2	Acronyms	7
1.6.3	Abbreviations	7
1.7	Reference documents	8
1.8	Document overview.	8
2	Overall Description	9
2.1	Product perspective	9
2.2	User Characteristics	9
2.3	Constrains	9
2.3.1	Regulatory Policies	9
2.3.2	Hardware Limitations	9
2.3.3	Software Limitations	10
2.3.4	Parallel Operations	10
2.3.5	Documents Related	10
2.4	Assumptions	10
2.5	Future possible Implementations	11
3	Specific Requirements	12
3.1	External Interface Requirements	12
3.1.1	User Interfaces	12
3.1.1.1	Login	12
3.1.1.2	Customer registration	14
3.1.1.3	Taxi Drivers registration	16
3.1.1.4	Customer home page	18
3.1.1.5	Reservation page	20
3.1.1.6	My reservation Page	22

3.1.1.7	Customer notification pop-up	24
3.1.1.8	User page	26
3.1.1.9	Taxi Driver home page	28
3.1.1.10	Taxi Driver notification	30
3.1.1.11	Registered user end ride page	32
3.1.2	Hardware Interfaces	34
3.1.3	Software Interfaces	34
3.1.4	Interfaces to Other Applications	34
3.1.5	Communication Interfaces	35
3.2	Functional Requirements	35
3.2.1	Functional Requirements for Guest Users	36
3.2.1.1	[G.1 - Allow guest users to become customers creating a myTaxiService Account]	36
3.2.1.2	[G.2 - Allow a Guest User to become a Customer using his Facebook Account]	36
3.2.1.3	[G.3 Allow guest users to become a taxi driver]	36
3.2.1.4	[G.4 Allow Guest Users to log in with myTaxiService account.]	37
3.2.1.5	[G.5 Allow Guest Users to log in with Facebook account.]	37
3.2.2	Functional Requirements for Registered Users	37
3.2.2.1	[G.6 Allow a Registered User to view or modify his username and email.]	37
3.2.2.2	[G.7 Allow a Registered User to retrieve his password if he doesn't remember it.]	38
3.2.2.3	[G.8 Allow a Registered User to signal another one if he has made a bad use of the system.]	38
3.2.3	Functional Requirements for Customers	39
3.2.3.1	[G.9 Allow customers to require a taxi.]	39
3.2.3.2	[G.10 Allow customers to reserve a ride.]	39
3.2.3.3	[G.11 Allow customers to delete a previous reservations.]	40
3.2.4	Functional Requirements for Taxi Drivers	40
3.2.4.1	[G.12 Allow Taxi Drivers to accept or decline a ride request.]	40
3.2.4.2	[G.13 Allow Taxi Drivers to notify their availability.]	41
3.3	Scenarios	41
3.3.1	S.1 Registration as a Customer creating a myTaxiService account	41
3.3.2	S.2 Registration as a Customer using a Facebook Account	41
3.3.3	S.3 Registration as a Taxi Driver creating a new Taxi Driver myTaxiService account.	42
3.3.4	S.4 Login with a myTaxiService account.	42
3.3.5	S.5 Customer Login with Facebook	42
3.3.6	S.6 Profile Modification.	43
3.3.7	S.7 Password Retrieval	43

3.3.8	S.8 Reporting Abuses	43
3.3.9	S.9 Require a Taxi	44
3.3.10	S.10 Reserve a Taxi	44
3.3.11	S.11 Delete a Reservation	44
3.3.12	S.12 Taxi Driver Accepts or Decline a Ride Request	45
3.3.13	S.13 Notify Taxi Driver Availability	45
3.4	UML Models	45
3.4.1	Use Case Diagram	45
3.4.2	UML Class Diagram	46
3.4.3	UML Interaction Diagrams	47
3.4.3.1	Registration through the creation of a new my-TaxiService account.	47
3.4.3.2	Registration using a Facebook account.	49
3.4.3.3	Taxi Driver Registration	50
3.4.3.4	Registered User Login	52
3.4.3.5	Customer Facebook Login	53
3.4.3.6	Profile Modification	54
3.4.3.7	Password Retrieval	55
3.4.3.8	Taxi Driver Reporting	57
3.4.3.9	Customer Reporting	57
3.4.3.10	Request a Taxi	58
3.4.3.11	Reserve a Taxi	59
3.4.3.12	Delete a Reservation	60
3.4.3.13	Accept or Decline a Ride Request	62
3.4.3.14	Taxi Driver Availability Notification	63
3.4.4	State Machine Diagrams	64
3.4.4.1	Taxi Request State Machine	65
3.4.4.2	Taxi Driver State Machine	65
3.5	Non Functional Requirements	65
3.5.1	Performance Requirements	66
3.5.2	Design Constraints	66
3.5.3	Software System Attributes	66
3.5.3.1	Adaptability	66
3.5.3.2	Availability	66
3.5.3.3	Compatibility	66
3.5.3.4	Fault-Tolerance	66
3.5.3.5	Maintainability	67
3.5.3.6	Modularity	67
3.5.3.7	Portability	67
3.5.3.8	Simplicity	67
3.5.4	Security	67
3.5.4.1	Application and User Interface Side	67
3.5.4.2	Server Side	68

4	Appendix	69
4.1	Alloy	69
4.1.1	Signatures	69
4.1.2	Facts	72
4.1.3	Predicates and Assertions	74
4.1.4	Generated World	75
4.2	Software and Tools	77
4.3	Hours of Work	77

Chapter 1

Introduction

1.1 Purpose

This document is the R.A.S.D. (Requirement Analysis and Specification Document). The purpose of this document is the description of the "myTaxiService" system. At first, it will provide functional and non-functional requirements, a complete overview of the constraints of the system and its limits. Then it will explain in detail the dynamics of the system using real-life use cases. Finally this document will provide a base for the developers that concretely have to implement the system.

1.2 Actual System

The functionality that the new system will provide is now not supported. So the entire system must be developed without using or modifying existing system.

1.3 Scope

The objective of myTaxiService is to provide an interface between customers and taxi drivers to optimize their interaction and provide a fair management of taxi queues. The users, once registered through the mobile application or the web application, can request a taxi for their travel or reserve one, specifying the origin and the destination. The reservation can be done at least two hour before the ride; if the reservation can take place, the system will allocate a taxi 10 minutes before the meeting time. On the other side, taxi drivers can inform the system that they are waiting for a client and accept or decline a ride request. If the request has been accepted, a notification will be sent to the requesting customer with the identification number of the incoming taxi and the time he has to wait. Otherwise, if the request has been rejected it will be forwarded to the next taxi in the queue. The system has to optimize the management of customers requests giving the rides

to the taxi with the highest priority that has to be evaluated in function of availability and the nearness of the taxi driver.

1.4 Actors

- **Guest User:** guest users are unlogged or unregistered users. They can visit the login page or the registration forms.
- **Registered User:** this kind of user identify either a Guest User or a Taxi Driver.
- **Customer:** this kind of user is the end-user of the service. He can perform request for taxis or reserve a ride. In his personal page he can view his requests and the system responses.
- **Taxi Driver:** this kind of user is composed by the actual taxi drivers that can only see customers requests that has been forwarded by the system. He can accept or decline these requests. Also, he's considered a special kind of user because one can register as a "Taxi Driver" only if he provide a valid Taxi licence.

1.5 Goals

- [G.1] Allow guest user to become a customer creating a myTaxiService Account.
- [G.2] Allow guest user to become a customer using his Facebook Account.
- [G.3] Allow guest user to become a taxi driver.
- [G.4] Allow registered user to log in with myTaxiService account.
- [G.5] Allow customer to log in with Facebook account.
- [G.6] Allow a Registered User to view or modify his username and email.
- [G.7] Allow a Registered User to retrieve his password if he doesn't remember it.
- [G.8] Allow a RegisteredUser to signal another one if he has made a bad use of the system.
- [G.9] Allow customers to require a taxi.
- [G.10] Allow customers to reserve a ride.
- [G.11] Allow customers to delete a previous reservations.
- [G.12] Allow taxi drivers to accept or decline a ride request.
- [G.13] Allow taxi drivers to notify their availability.

1.6 Definitions, Acronyms, Abbreviations

1.6.1 Definitions

- Requesting User: user that requests a taxi ride. This ride can be a Reservation or a Normal ride.
- Reservation (or "Reserve a Ride") : A ride that is reserved for a future journey, the taxi is not called immediately but will be called at the time decided by the user.
- Normal Ride: A ride that is call to be handled as soon as possible. If is not specified, a ride is considered as a normal ride.
- Zone: We decide to divide the city area into zone. this organization is done to optimize the service. The shape and the dimension of this zone is not decided yet.
- Zone Queue: This Queue is composed of taxi drivers. With this queue we can organize the taxi, in order of priority, to optimize the service. Each Zone will have exactly one Queue of TaxiDrivers.

1.6.2 Acronyms

- RASD: Requirement Analysis and Specification Documents.
- DD: Design Document.
- UML: Unified Modeling Language.
- OS: Operative System.
- API: Application Program Interface.
- GPS: Global Positioning System.
- HTTP: Hypertext Transfer Protocol.
- HTTPS: Secure Hypertext Transfer Protocol.

1.6.3 Abbreviations

- G.x is the x-Goal.
- Req.x is the x-Functional Requirement.
- Dom.x is the x-Domain Assumption.
- S.x is the x-Scenario.

1.7 Reference documents

- (IEEE830) IEEE Recommended Practice for Software Requirements Specifications

1.8 Document overview.

Until now, we have given a general explanation about the software functionalities and a brief description of this document. Now we will describe what the rest of this RASD contains.

In Section 2 we will focus more on system constraints and assumptions.

In Section 3 we will describe requirements, typical scenarios and use-cases. In this section there is also a collection of UML diagrams that describes in particular the functionalities of the system.

In Section 4 we will describe the alloy documentation about our system and describe all the software used to make this document.

Chapter 2

Overall Description

2.1 Product perspective

The system will be composed of a web application and a mobile application developed for the three major OS (Apple iOS, Android, Windows 10). The system will provide some API with the purpose of a future connection with another travel planning systems.

2.2 User Characteristics

We suppose that there are two types of users that will use our system. The first type is composed of who want to find a taxi for a travel in the simplest way (customers). They must be able to access to a web browser or download and using a mobile application. The second one is composed of taxi drivers, who want to increment their productivity. In addition to being able to access to a web browser or downloading and using our mobile application, they must have a taxi license.

2.3 Constrains

2.3.1 Regulatory Policies

myTaxiService has to meet regulatory policies about taxis in the countries where it will be used.

2.3.2 Hardware Limitations

The only hardware limitation that the myTaxiService mobile application has to meet is the mobile phones characteristics. the rest of the system will not be affected by particular hardware limitations.

2.3.3 Software Limitations

myTaxiService mobile application has to be compatible with all major mobile operating systems (Android, Apple iOS, Windows 10). Moreover, myTaxiService web application has to be compatible with all major browsers (Chrome, Safari, Firefox, Microsoft Edge).

2.3.4 Parallel Operations

Our system must be able to perform parallel operations on the database to satisfy all the requests from multiple users.

2.3.5 Documents Related

- Requirements and Analysis Specification Document (RASD)
- Design Document (DD)

2.4 Assumptions

- Every taxi driver has equipped a smartphone during working hours.
- Every taxi driver has a unique taxi license.
- Every taxi has a GPS locator to send GPS information to the central server.
- Android, Apple iOS or Windows 10 are available on the registered users smartphones.
- Every registered users can connect to the Internet with a mobile device when outside.
- When a customer require a taxi, the GPS information about his location are automatically sent to the central server.
- The reservation of a ride is made at least two hours before the ride.
- The deletion of a reservation is made at least two hours before the ride.
- The requests of the customers are automatically notified to the first taxi driver in the zone queue.
- If a taxi driver declines a request he will be placed in the bottom of the zone queue.
- If a request is declined it will be forwarded to the next taxi driver in the zone queue.
- If a customer makes a bad use of the taxi request system, he can be reported as a bad customer.

- If a taxi driver notifies his availability, it is because he is actually available
- If a taxi driver notifies his availability, it is because he wants to be notified of customers that needs a ride.
- If a taxi driver accepts a request, the requesting customer will be notified

2.5 Future possible Implementations

A possible future implementation can be a complex feedback system, that allows the customers to leave a comment about the taxi driver and vice versa. For example, taxi drivers can be interested in knowing the punctuality or the behaviour of the customer who requests the ride.

Chapter 3

Specific Requirements

This chapter contains a detailed description of how the application works and its features. It also gives a specification of the functional and quality requirements.

3.1 External Interface Requirements

This section gives a description of the various inputs and relative outputs of the system. It also gives a description of the hardware, software and communication interfaces that are necessary to make the system work. It provides a generic visualization of the user interface in the various user platforms.

3.1.1 User Interfaces

Here we describe in particular how the application should look like either for mobile and web application. To make an easier explanation of the aspect of the various screen of the application we are going to use Mockup.

3.1.1.1 Login

On this page the users can log in with username and password or with Facebook (only if the registered user is a Customer). If the user is not registered yet, he can access to the registration pages (for Customers or for Taxi Drivers). Furthermore, if the user has forgotten the password, he can get it from this page.

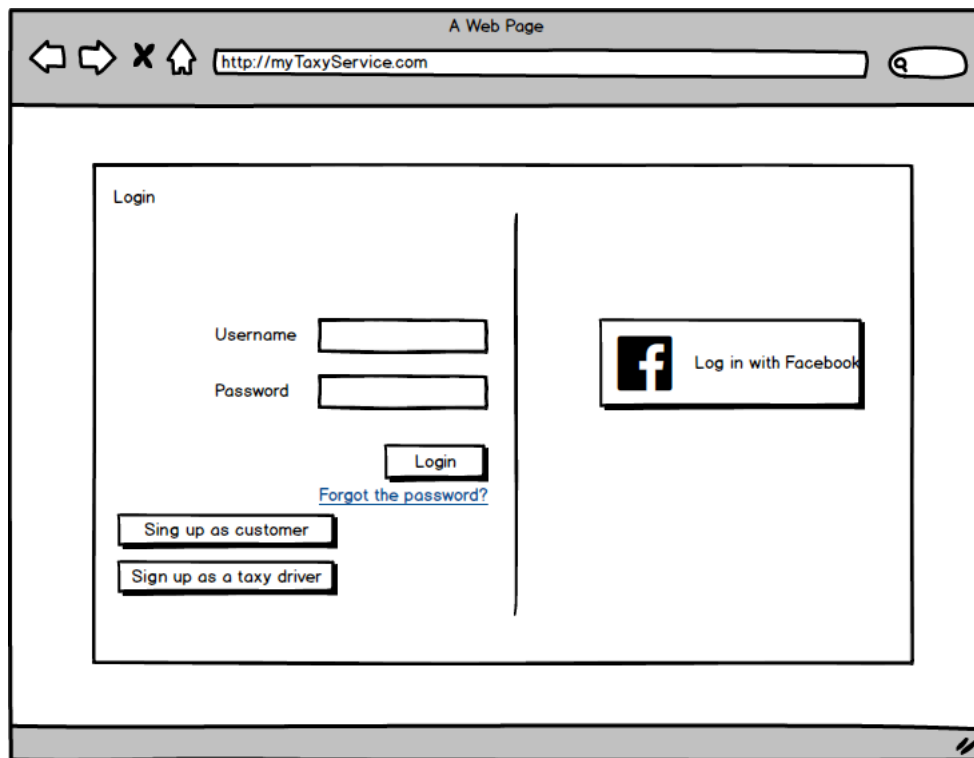


Figure 3.1: Login Page, web version

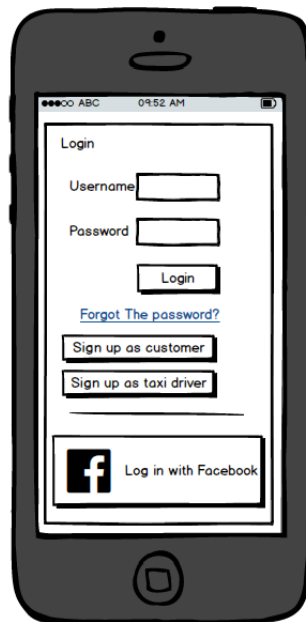


Figure 3.2: Login Page, mobile version

3.1.1.2 Customer registration

On this page the user can register itself. This page must provide two ways of registration: registration through the standard form (e-mail, password and username) or through Facebook API.

A Web Page

http://myTaxiService.com

Customer registration

Username

Password

Repeat password

e-mail


 Log in with Facebook

Figure 3.3: Customer registration Page, web version

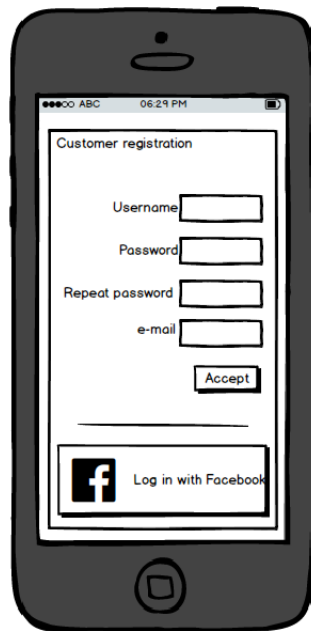


Figure 3.4: Customer registration Page, mobile version

3.1.1.3 Taxi Drivers registration

On this page the user can register itself as a taxi driver. The taxi driver has a special form to be registered because of the additional information the user must provide (taxi license number). Due to the additional information, the registration can be done only through the standard form.

A Web Page

http://myTaxiService.com

Taxi drivers registration

Username

Password

Repeat password

e-mail

license number

taxi number

Accept

Figure 3.5: Taxi Driver registration Page, web version

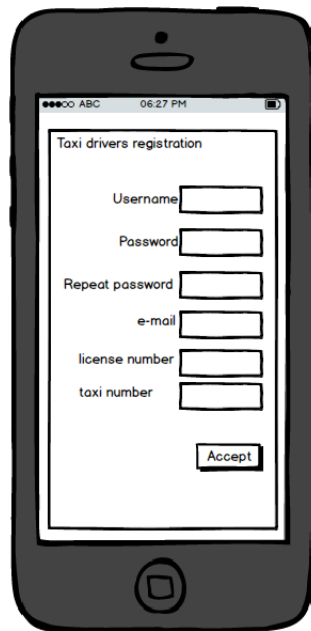


Figure 3.6: Taxi Driver registration Page, mobile version

3.1.1.4 Customer home page

On this page the Customer can see his position (information from his GPS) and perform all the main operations. He can Request a taxi on the showed position, can go to the reservation page (in which can reserve a ride), can visit his personal page (in which can see all the information of his profile), can go to the information page (information about myTaxiService) or go to the 'my reservation page' (in which can manage all the previous reserved rides).



Figure 3.7: Customer home page, web version

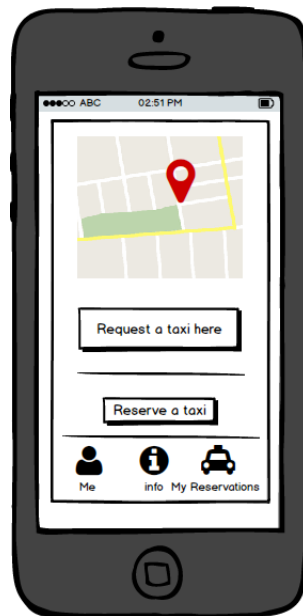


Figure 3.8: Customer home page, mobile version

3.1.1.5 Reservation page

On this page the Customer can reserve a taxi ride. To do this, he must insert a Origin position, a Destination position, a Data and a Time. The reservation must be at least two hours after the current time.



Figure 3.9: Reservation page, web version

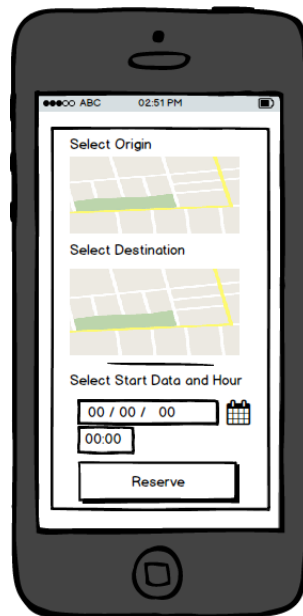


Figure 3.10: Reservation page, mobile version

3.1.1.6 My reservation Page

On this page the Customer can see all the reservations he has already done and add some new reservations. For each previous reservation he can see all the information and, if the Date is after the next two hours, he can delete it.

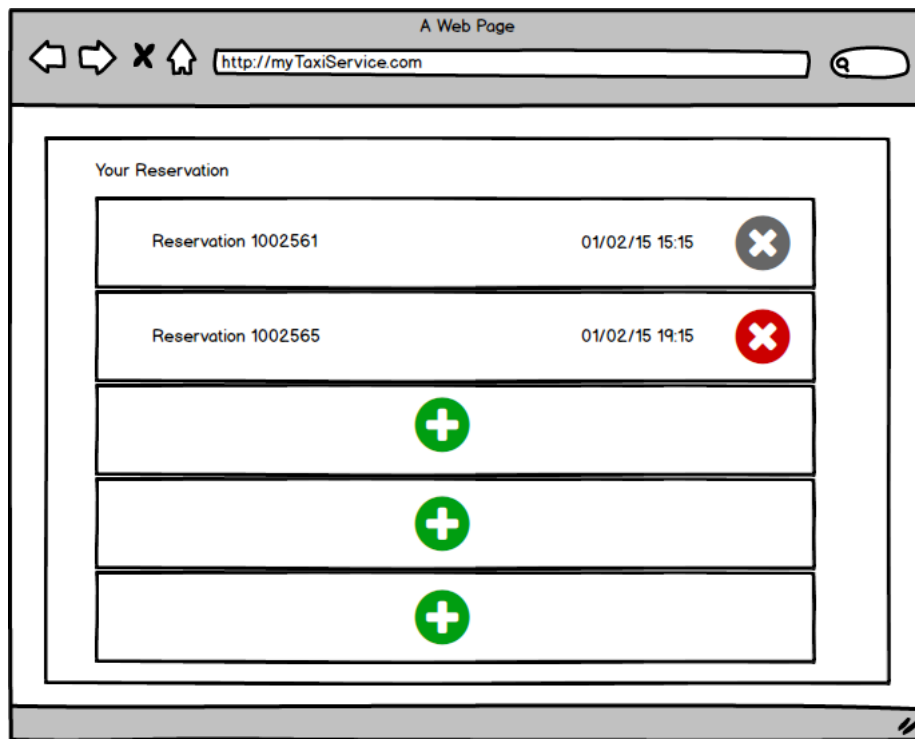


Figure 3.11: My reservation, web version

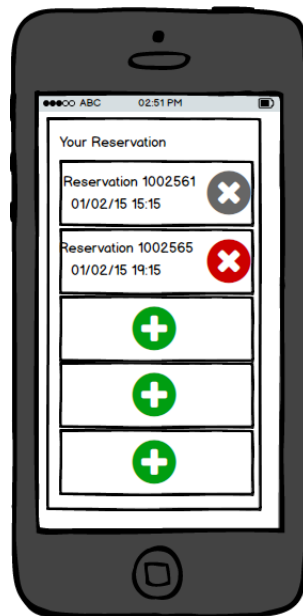


Figure 3.12: My reservation, mobile version

3.1.1.7 Customer notification pop-up

This pop-up is showed to the requesting Customer when his request has been handled. On this pop-up the system shows also the identification number of the incoming taxi and an approssimative waiting time.



Figure 3.13: Customer notification, web version

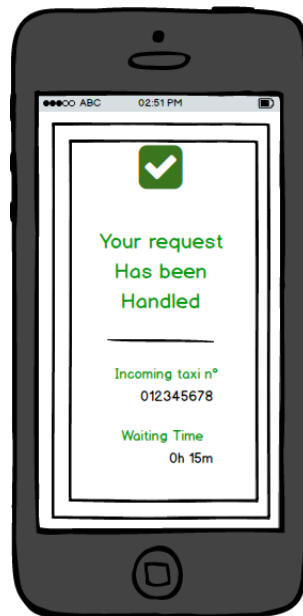


Figure 3.14: Customer notification, mobile version

3.1.1.8 User page

This page shows all the information about your user and show how many time you've been notified as bad user.

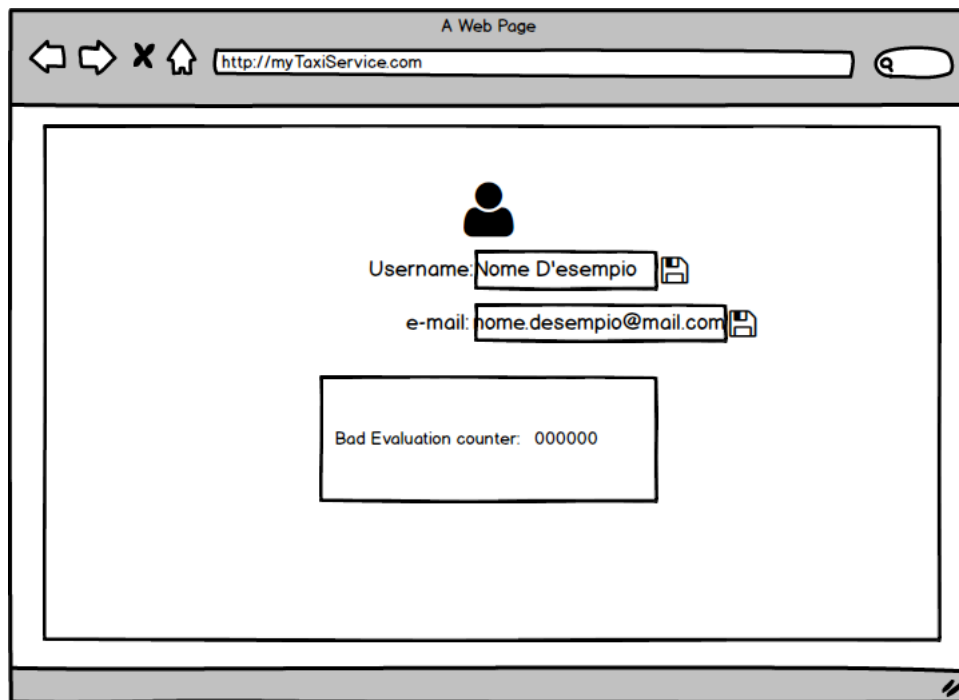


Figure 3.15: User Page, web version

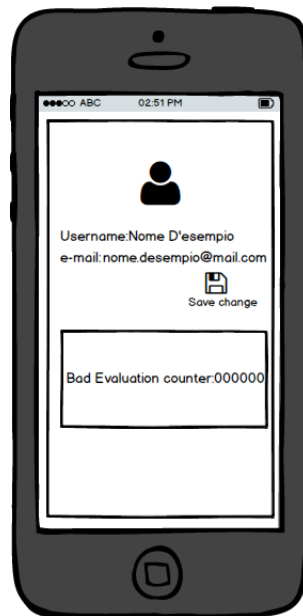


Figure 3.16: User Page, mobile version

3.1.1.9 Taxi Driver home page

On this page the Taxi driver can see his position (information from his GPS) and inform the system about his availability.

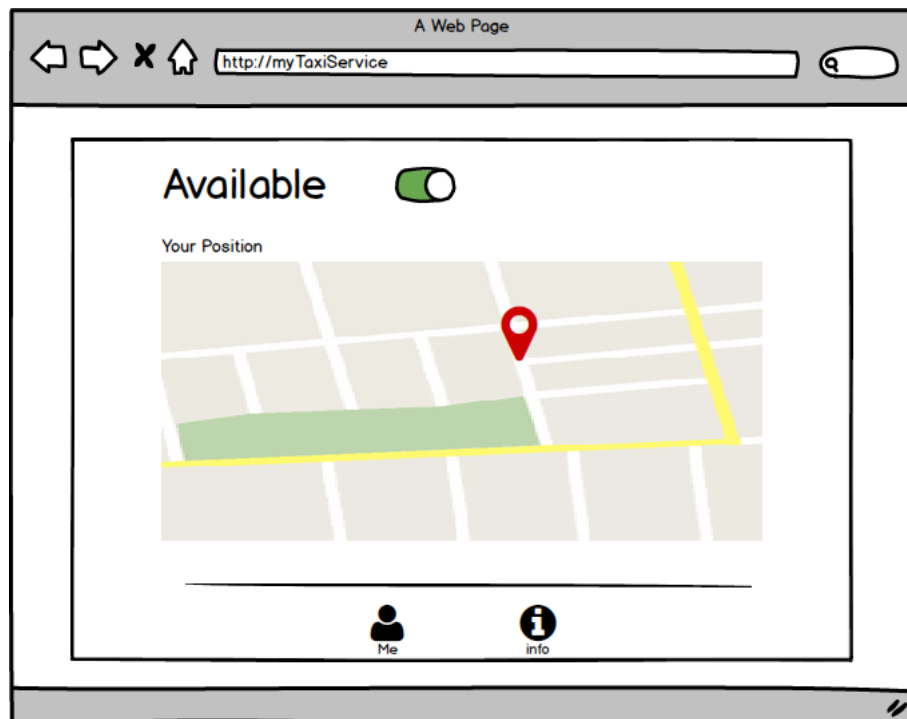


Figure 3.17: Taxi driver home page, web version

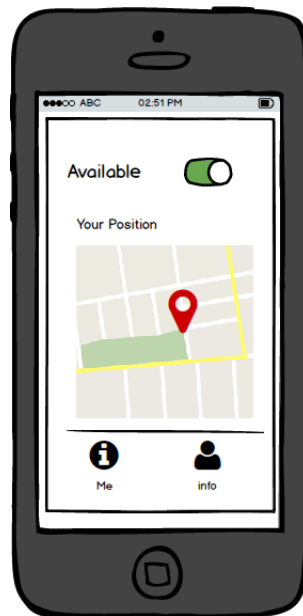


Figure 3.18: Taxi driver home page, mobile version

3.1.1.10 Taxi Driver notification

On this pop-up the taxi driver will be notified of a request that he can handle. On this pop-up are shown also two buttons, with which the taxi driver can accept or decline the request.



Figure 3.19: Taxi driver notification pop-up, web version

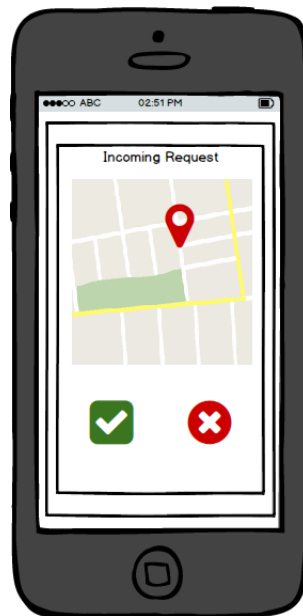


Figure 3.20: Taxi driver notification pop-up, mobile version

3.1.1.11 Registered user end ride page

When a taxi driver accepts a ride, this page is shown up to each registered user involved in this ride. On this page the taxi driver can check the name of the customer and the Origin position of his journey. When the ride is ended each of the registered user can give a bad evaluation to the other or simply end the ride and return to the main page.

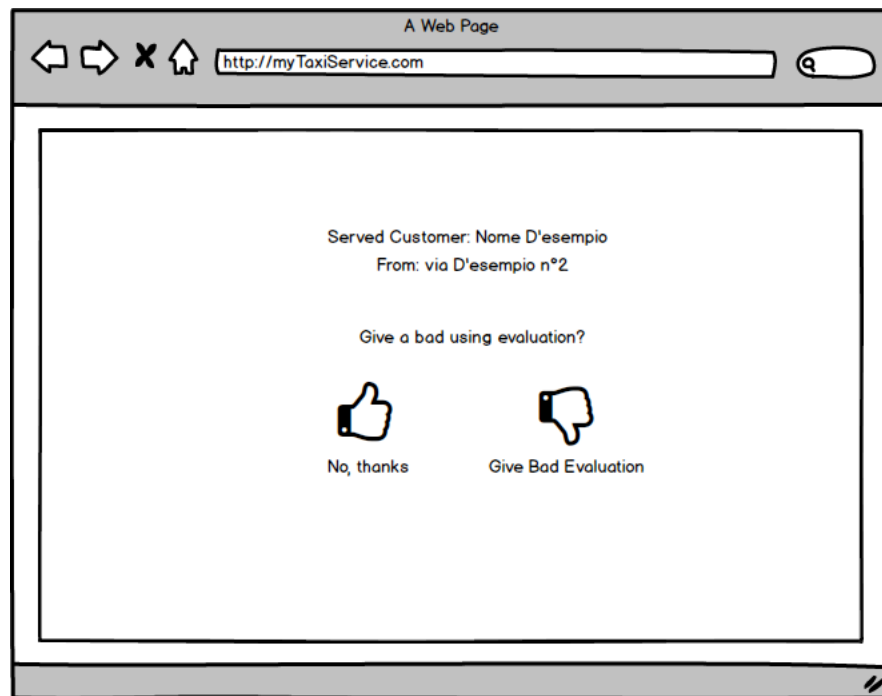


Figure 3.21: On Ride Page, web version

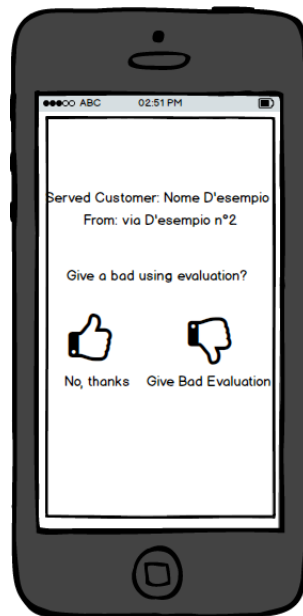


Figure 3.22: On Ride page, mobile version

3.1.2 Hardware Interfaces

Since mobile and web applications don't have any dedicated hardware we have not designed any hardware interfaces for our system. The interaction with the central database is performed by connections handled by the already installed operating system on the mobile devices or the users' computers.

3.1.3 Software Interfaces

- The mobile application communicates with the GPS application in order to get geographical information about the user.
- The web application communicates with the browser in order to get geographical information about the user.
- Mobile and web applications communicate with the database through HTTP requests to the server.

3.1.4 Interfaces to Other Applications

- myTaxiService web application requires that at least one of these browsers is installed on the user's Personal Computer:

Table 3.1: Browsers

Name	Version	Company	Source
Safari	9.0.1	Apple Inc.	Get Safari
Firefox	41.0	Mozilla	Get Firefox
Chrome	46.0.2490	Google	Get Chrome
Microsofr Edge	20.10240.16384.0	Microsoft	Get Edge

- myTaxiService mobile application requires that at least one of these operating systems is installed on the user's Smartphone:

Table 3.2: Mobile Operative Systems

Name	Version	Company	Source
Android	KitKat 4.4W.2 or later	Google	Android Info
iOS	9.1 or later	Apple Inc.	iOS Info
Windows 10	10.0.10572.0 or later	Microsoft	Windows 10 Info

- To give an additional LogIn method, we use also the "LogIn with Facebook" API released by Facebook. Facebook Login for Apps is a fast and convenient way for people to create accounts and log into our system across multiple platforms. It is well described at Facebook Login API Page.

3.1.5 Communication Interfaces

The communication between system pieces is not specified because it is handled by the underlying operating systems for both the mobile application and the web portal.

In particular, the web and mobile applications will communicate with the server through HTTP/HTTPS requests.

- HTTP communicate through the port number 80 and is handled by the operating system.
- HTTPS communicate through the port number 443 and is handled by the operating system.

3.2 Functional Requirements

In this section are described, for every Actor, the Functional Requirements needed to reach the linked Goal.

3.2.1 Functional Requirements for Guest Users

Here are listed all the Functional Requirements referring to the Goals that affect Guest Users.

3.2.1.1 [G.1 - Allow guest users to become customers creating a myTaxiService Account]

To allow the guest user to perform a successful registration, the system has to:

- [Req.1] provide a registration page containing:
 1. A text box where the user can insert his username.
 2. Two text boxes where the user can insert his password (the second one is for security check).
 3. A text box where the user can insert his email.
 4. A button to submit information to the system.
- [Req.2] The registration page shall be the only page accessible by a Guest User.
- [Req.3] All the information submitted by the user must be validated by the system.
- [Dom.1] The email used by the Guest User is a valid one.

3.2.1.2 [G.2 - Allow a Guest User to become a Customer using his Facebook Account]

To allow the guest user to perform a successful registration using Facebook API, the system has to:

- [Req.1] provide a registration page containing:
 1. A button that calls the Facebook Registration API.
- [Req.2] The registration page shall be the only page accessible by a Guest User.
- [Req.3] All the information must be evaluated by the Facebook system.

3.2.1.3 [G.3 Allow guest users to become a taxi driver]

To allow the guest user to become a Taxi Drivers, the system must:

- [Req.1] provide a registration page containing:
 1. A text box where the user can insert his username.

2. Two text boxes where the user can insert his password (the second one is for security check).
 3. A text box where the user can insert his email.
 4. A text box where the user can insert his valid taxi license.
 5. A text box where the user can insert his valid taxi number.
 6. A button to submit information to the system.
- [Req.2] The registration page shall be the only page accessible by a Guest User.
 - [Req.3] All the information submitted by the Guest User must be validated by the system.
 - [Dom.1] The email used by the Guest User is a valid one.

3.2.1.4 [G.4 Allow Guest Users to log in with myTaxiService account.]

To allow a registered user to log in with his myTaxiService account, the system must:

- [Req.1] provide a login page containing:
 1. A text box where the user can insert his username.
 2. A text box where the user can insert his password.
 3. A button to submit information to the system.

3.2.1.5 [G.5 Allow Guest Users to log in with Facebook account.]

To allow a customer to log in with his Facebook account, the system must:

- [Req.1] provide a button that calls the Facebook Login API.
- [Req.2] delegate to the Facebook system all the checks about the existence of the user.

3.2.2 Functional Requirements for Registered Users

Here are listed all the Functional Requirements referring to the Goals that affect Registered Users.

3.2.2.1 [G.6 Allow a Registered User to view or modify his username and email.]

To allow a Registered User to view or modify his username and email, the system must:

- [Req.1] provide an homepage that contains:

1. A button that, if clicked, shows the page that contains the Registered User's information.
- [Req.2] provide a page used to show Registered User's information that contains:
 1. A clickable label showing the username of the Registered User.
 2. A clickable label showing the email of the Registered User.
 3. A box with a label inside that shows how many times the Registered User has been reported as a bad user.
 - [Req.3] The personal Registered User's page shall be only accessible after the login.

3.2.2.2 [G.7 Allow a Registered User to retrieve his password if he doesn't remember it.]

To allow a Registered User to retrieve his password, the system must:

- [Req.1] provide a link, on the log in page, that will show to the Registered User a field where he can write his email to allow the system to send him an email containing the password.

3.2.2.3 [G.8 Allow a Registered User to signal another one if he has made a bad use of the system.]

To allow a Registered User to signal another one, the system must:

- [Req.1] check if the Registered User is correctly logged in.
- [Req.2] for Taxi Drivers, provide an end-ride page that contains:
 1. A label showing the username of the Customer that he has just served.
 2. A label showing the starting location of the current ride.
 3. A label that asks to the Taxi Driver if he wants to report the Customer.
 4. A button that ends the current ride without reporting the Customer.
 5. A button that ends the current ride incrementing the Customer's Bad Evaluation Counter.
- [Req.3] When one of the Taxi Driver end-ride page buttons is pressed, the system must ensure that the page is not reachable anymore and the Taxi Driver must access only to his homepage.
- [Req.4] for Customers, provide an end-ride page that contains:
 1. A label showing the username of the Taxi Driver that has just served him.

2. A label showing the starting location of the current ride.
 3. A label that asks to the Customer if he want to report the Taxi Driver.
 4. A button that ends the current ride without reporting the Taxi Driver.
 5. A button that ends the current ride and increments the Taxi Driver's Bad Evaluation Counter.
- [Req.5] When one of the Customer end-ride page button is pressed, the system must ensure that the page is not reachable anymore and the Customer must access only to his homepage.
 - [Req.6] The end-ride page shall be only accessible after the Registered User login.
 - [Dom.1] The ride is ended after a Taxi Driver or a Customer action.

3.2.3 Functional Requirements for Customers

Here are listed all the Functional Requirements referring to the Goals that affect Customers.

3.2.3.1 [G.9 Allow customers to require a taxi.]

To allow the customers to require a taxi, the system must:

- [Req.1] provide an homepage that contains:
 1. A map showing the current location of the customer.
 2. A button that calls the function of the system to require a taxi (called "require button").
- [Req.2] The require-taxi page shall be only accessible by Customers and after they have performed the login as a Customer.

3.2.3.2 [G.10 Allow customers to reserve a ride.]

To allow the customers to reserve a ride, the system must:

- [Req.1] provide an homepage that contains:
 1. A button that, if clicked, shows the page used to reserve a ride.
- [Req.2] provide a page used to reserve a ride that contains:
 1. A map where the customer can select the starting location for his ride.
 2. A map where the customer can select the ending location for his ride.
 3. A field where the customer can insert the starting date for the ride.
 4. A field where the customer can insert the starting time for the ride.

- 5. A button that calls the function of the system to reserve a ride (called "reserve button").
- [Req.3] The page used to reserve a ride shall be only accessible after the Customer login.
- [Req.4] "reserve button" must be clickable only if the reserve date and time is at least two hours after the current time. (for instance: if the current date is 10/10/2015 and current time is 10.00, the reservation time must be at least at 12.00 of the 10/10/2015).
- [Req.5] The system must notify the Customer that a Taxi Driver has accepted his reservation at least 10 minutes before the ride.

3.2.3.3 [G.11 Allow customers to delete a previous reservations.]

To allow a customer to delete a previous reservation, the system must:

- [Req.1] provide an homepage that contains:
 1. A button that, if clicked, shows the page that contains all the reservations made by the customer.
- [Req.2] provide a page that contains all the reservations made by the customer:
 1. All reservations that have a difference between the start time and the current time of less than two hours are not erasable.
 2. Other reservations are erasable.
 3. The page also contains a button to add a new reservation. If clicked, that button leads the customer to the page used to reserve a ride (described in the Functional Requirements Req.3 of G.4)
- [Req.3] The page that contains all the reservations shall be only accessible after the Customer login.
- [Dom.1] The reservation is deleted after the Customer action.

3.2.4 Functional Requirements for Taxi Drivers

Here are listed all the Functional Requirements referring to the Goals that affects Taxi Drivers.

3.2.4.1 [G.12 Allow Taxi Drivers to accept or decline a ride request.]

To allow Taxi Drivers to accept or decline ride requests, the system must:

- [Req.1] notify the Taxi Driver (if he is available) that is at the top of the zone-queue that a new taxi-request is made (Req.4 of G.9).
- [Req.2] on the notification screen, show two buttons: one for the acceptance and one for the declination of the taxi-request.

3.2.4.2 [G.13 Allow Taxi Drivers to notify their availability.]

To allow Taxi Drivers to notify their availability, the system must:

- [Req.1] provide an home page that contains:
 1. An ON/OFF button that signals to the system that the Taxi Driver is Available/Not Available.
 2. A map showing his current zone and position.
- [Req.1] The home page should be accessible only if the Taxi Driver is logged in.

3.3 Scenarios

Here are described in natural language some useful scenarios that show how the system should work in different cases:

3.3.1 S.1 Registration as a Customer creating a myTaxiService account

To read the Functional Requirements for this scenario, refer to G.1 Functional Requirements Specification

Aristotle is new in the system and he wants to simplify the way he calls for a taxi. So he downloads the "myTaxiService" application from the Play Store (since he has a smartphone with the fancy Android 5.0 Lollipop Operative System) and he opens it. The application shows to Aristotle the login screen and he taps on the "Sign Up as a Customer" button (see Login Screen) going to the registration page. In that page, he has to fill the form (see Customer Registration Screen (Mobile)) and after that he clicks on the "Accept" button. If the data entered by Aristotle are ok, he is automatically logged in and he can start using myTaxiService. Otherwise, he is redirected to an error page, that describes the problem.

3.3.2 S.2 Registration as a Customer using a Facebook Account

To read the Functional Requirements for this scenario, refer to G.2 Functional Requirements Specifications

Pythagoras is new in the system and he wants to simplify the way he calls for a taxi, but he doesn't want to loose time inserting his credential into a boring registration form, so he decides to register with his Facebook Account. He downloads the application from the Apple App Store (since he has a glorious iPhone 6 Plus) and he opens it. The application shows to Pythagoras the login screen and he taps

on the "Log in With Facebook" button calling the Facebook Login API. If the registration is successful, the Facebook System will reply with the information about the user and the myTaxiService system can create a record in the database with Pythagoras information. Otherwise, he is redirected to an error page, that describes the problem.

3.3.3 S.3 Registration as a Taxi Driver creating a new Taxi Driver myTaxiService account.

To read the Functional Requirements for this scenario, refer to G.3 Functional Requirements Specifications

Thales is a great taxi worker and he wants to improve his service to the customers so he decides to join myTaxiService. First of all he visits the myTaxiService web page from his pc. That page shows the login screen (see Login Screen) and he clicks on the "Sign up as a Taxi Driver" button. The web application redirects him to the Taxi Drivers registration page. Thales fills up the form (see Taxi Driver Registration Screen (Web)) and clicks on the "Accept" button. The system performs a special check for the validity of the given taxi license and taxi number. If the data provided by Thales are ok, he is automatically logged in and he can start using myTaxiService as a Taxi Driver. Otherwise, he is redirected to an error page, that describes the problem.

3.3.4 S.4 Login with a myTaxiService account.

To read the Functional Requirements for this scenario, refer to G.4 Functional Requirements Specifications

Epicurus is a Registered User and he wants to login to use the myTaxiServer Application with his smartphone. So he taps on the application icon and starts using it. myTaxiService Application shows to him the login screen (see Login Screen) and Epicurus fills up the form with his credentials. After that he submits the data by pressing the "Login" button. The Application calls the server function to check the credentials of Epicurus and if they are ok, the server replies to the application with a success message and the homepage is shown to Epicurus. Otherwise, the server replies with an error message that is shown to Epicurus.

3.3.5 S.5 Customer Login with Facebook

To read the Functional Requirements for this scenario, refer to G.5 Functional Requirements Specifications

Solon is a Customer of myTaxiService and he is registered with his Facebook account as described in the scenario S.2. Now he wants to log in and use myTaxiService from his iPhone. First of all he opens the application that shows the

login screen (see Login Screen). In a second moment, he clicks on the "Log in with Facebook" button and he logs in.

3.3.6 S.6 Profile Modification.

To read the Functional Requirements for this scenario, refer to G.6 Functional Requirements Specifications

Zeno is a Registered User but he wants to change his username. He opens his browser and navigates to the myTaxiService Web Application. He logs in to the system and the application shows his homepage (see Homepage). Zeno clicks on the "Me" button to view the profile page (see Profile Page). He clicks on his username to modify it. Once he is satisfied of his new username, he clicks on the "Save" button to save his modifications. Now Zeno is happy.

3.3.7 S.7 Password Retrieval

To read the Functional Requirements for this scenario, refer to G.7 Functional Requirements Specifications

Archimedes is a Registered User and he wants to log in to the myTaxiService Application. Unfortunately, he doesn't remember his password, so he taps on the link in the login page (see Login Screen) and the myTaxiService system sends him an email with the password if the provided email is registered. Otherwise an error message is shown to Archimedes.

3.3.8 S.8 Reporting Abuses

To read the Functional Requirements for this scenario, refer to G.8 Functional Requirements Specifications

Crates is a Customer and he desperately wants to go to the cinema with his friends. He logs in to the myTaxiService Mobile Application and navigates to the homepage. Then he taps on the "Request a Taxi here" button and waits for a taxi (since he receives the acceptance notification). This taxi does not arrive and he can't reach the cinema in time. Crates is very frustrated, so he reports the Taxi Driver that has accepted his taxi request.

Gorgias is a Taxi Driver and he has already logged in to the myTaxiService Mobile Application and signaled his availability. He receives a taxi request from a Customer and he accepts it, but when he goes to the meeting location, the Customer is not there. Gorgias has lost precious time and money so he is very angry and reports the Customer through the application.

3.3.9 S.9 Require a Taxi

To read the Functional Requirements for this scenario, refer to G.9 Functional Requirements Specifications

Plutarch is a Customer who wants to go to his vacation house, but he is already arrived to the city train station and he doesn't have a car, so he decides to call a taxi. He opens myTaxiService and the application shows to him the homepage. First, he checks that the position showed is his current position, so he taps the "Request a taxi here" button and waits for a taxi to accept his request. After 2 minutes a Taxi Driver accepts his request and a pop up notification notifies it to Plutarch. The notification says that a taxi will reach his position after 10 minutes and informs Plutarch about the number of the incoming taxi, so Plutarch can't take the Wrong taxi.

3.3.10 S.10 Reserve a Taxi

To read the Functional Requirements for this scenario, refer to G.10 Functional Requirements Specifications

Proclus is a traveler and he knows that today at 18.00 he has to take a taxi to go to the train station. Proclus is also a myTaxiService Customer, so to reach his goal he logs into the myTaxiService web application from his laptop. The homepage is showed to Proclus and he clicks on the "Reserve a Taxi" button. The web application now shows to Proclus the reservation page (see Reservation Page). Our traveler now inserts the data for his ride and confirms the reservation by clicking on the "Reserve" button. Since the time now is 11.00, the system accepts his reservation and a notification is sent to Proclus making he happy.

3.3.11 S.11 Delete a Reservation

To read the Functional Requirements for this scenario, refer to G.11 Functional Requirements Specifications

Proclus is a Customer that has reserved a ride (see S.10) for this evening, but unfortunately he can not leave today. So he has to delete his taxi reservation. First of all he logs into the myTaxiService web application from his laptop and opens the homepage. He clicks on the "My Reservations" button to access to his reservations page. Here are listed all the reservations that he has made. He now clicks on the "X" at the right of the reservation and the myTaxiService web application tries to delete the reservation trough calling the function of the server. Since now the time is 15.00, the server accepts the deletion and informs the application that notifies the successful operation to Proclus.

3.3.12 S.12 Taxi Driver Accepts or Decline a Ride Request

To read the Functional Requirements for this scenario, refer to G.12 Functional Requirements Specifications

Solon is a Taxi Driver registered to the myTaxiService system. He's currently available on the system and he's at the top of his current zone queue. A customer makes a request (see how in Scenario S.9) somewhere in the city and the system notifies Solon of the incoming request. When Solon sees the notification, he accepts the ride and the system notifies the requesting Customer of the acceptance.

3.3.13 S.13 Notify Taxi Driver Availability

To read the Functional Requirements for this scenario, refer to G.13 Functional Requirements Specifications

Galen is a Taxi Driver registered in the myTaxiService system. He has already performed the login and he is now on the Taxi Driver Homepage. He wants to be notified by the system of the incoming requests for rides but he is currently unavailable on the system. So he taps on the "Available" ON/OFF button in his homepage to signal his availability to the system, that inserts him in his current zone queue. Then the ON/OFF button becomes green to notify to Galen that he is now available.

3.4 UML Models

3.4.1 Use Case Diagram

In the following figure is explained how the Actor interacts with the system and how the various use cases interact with each other.

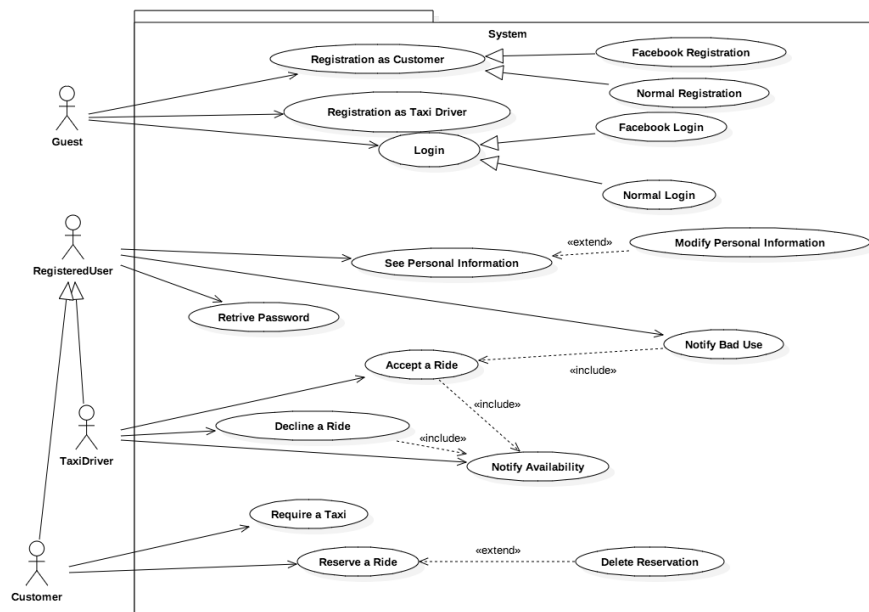


Figure 3.23: Use Case Diagram

3.4.2 UML Class Diagram

In the following figure, is explained how the classes of the system interact each others. Are also indicated some minimal attributes of the system classes.

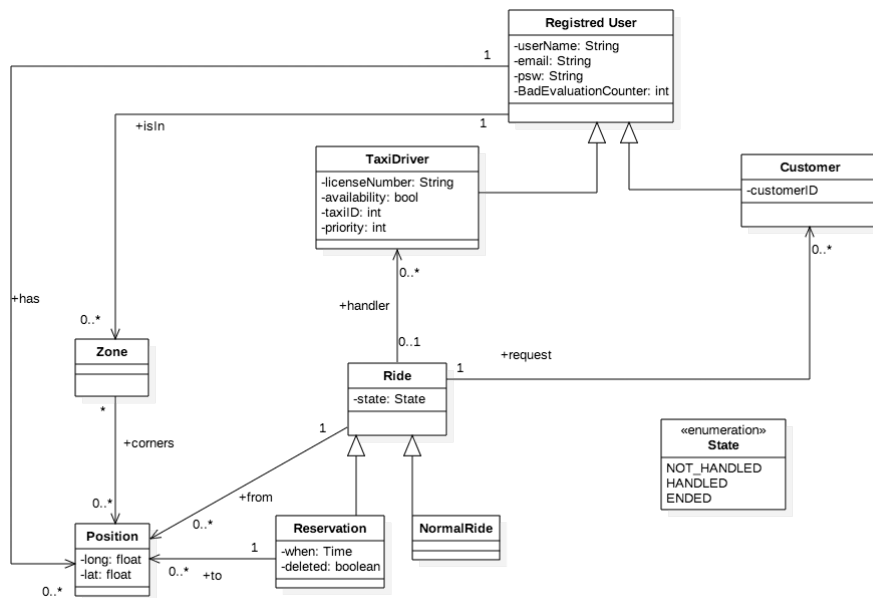


Figure 3.24: Class Diagram

3.4.3 UML Interaction Diagrams

Here are showed the interactions between the various actors in the execution of the scenarios (see Scenarios).

3.4.3.1 Registration through the creation of a new myTaxiService account.

To accompany this diagram, read the Scenario S.1.

Subject	Description
Actors	Guest User, myTaxiService Application, myTaxiService Server
Preconditions	Guest User must not be already registered
Execution	<ol style="list-style-type: none"> 1. Guest User opens the myTaxiService Application. 2. Guest User taps on the "Sign Up as a Customer" button. 3. myTaxiService Application shows the registration page. 4. Guest User fills up the registration form. 5. Guest User taps on the "Submit" button.
Postconditions	The Guest User is now a Customer, he is registered in the database and he is now logged in.
Exceptions	<ol style="list-style-type: none"> 1. The email regex is not optimal (no regex for email is optimal) 2. The connection is lost or the Guest User doesn't complete the registration clicking the button.

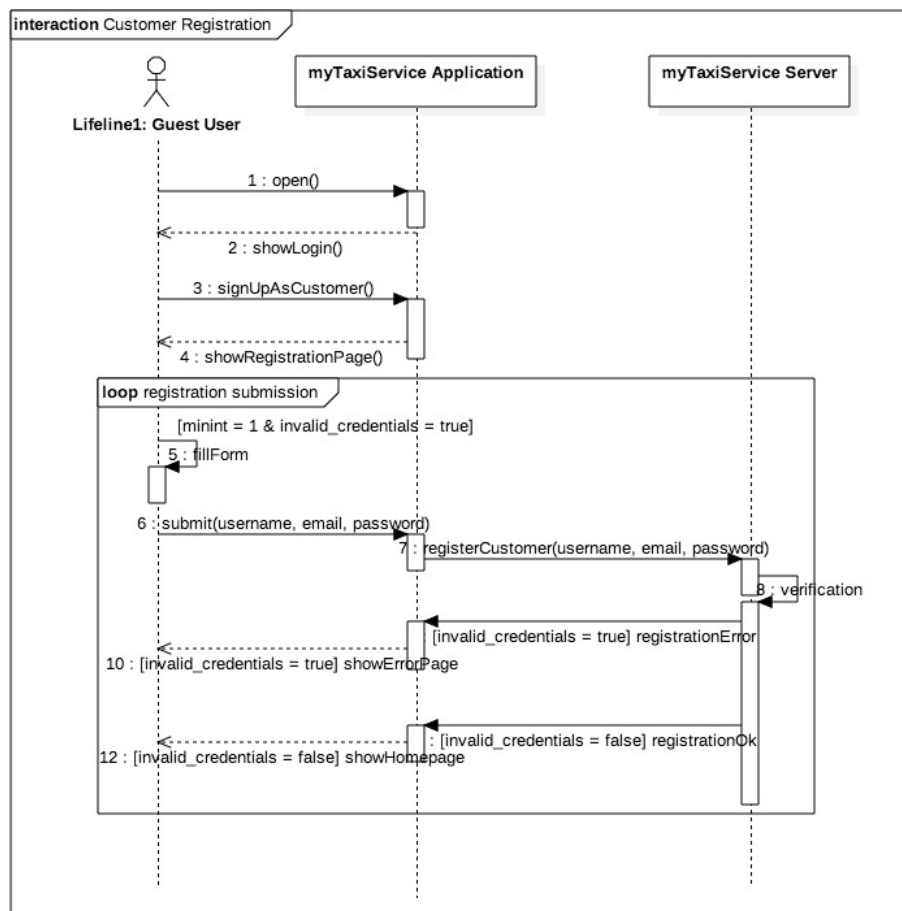


Figure 3.25: Customer Registration Interaction Diagram

3.4.3.2 Registration using a Facebook account.

To accompany this diagram, read the Scenario S.2.

Subject	Description
Actors	Guest User, myTaxiService Application, myTaxiService Server, Facebook Login System
Preconditions	Guest User must not be already registered
Execution	<ol style="list-style-type: none">1. Guest User opens the myTaxiService Application.2. Guest User taps on the "Log in with Facebook" button.3. myTaxiService Application calls Facebook Login API4. Facebook Login System makes some operations5. Facebook Login System replies6. myTaxiService Application shows the result to the Guest User
Postconditions	The Guest User is now a Customer, he is registered in the database and he is now logged in.
Exceptions	<ol style="list-style-type: none">1. The connection is lost or the Guest User doesn't complete the registration clicking the button.

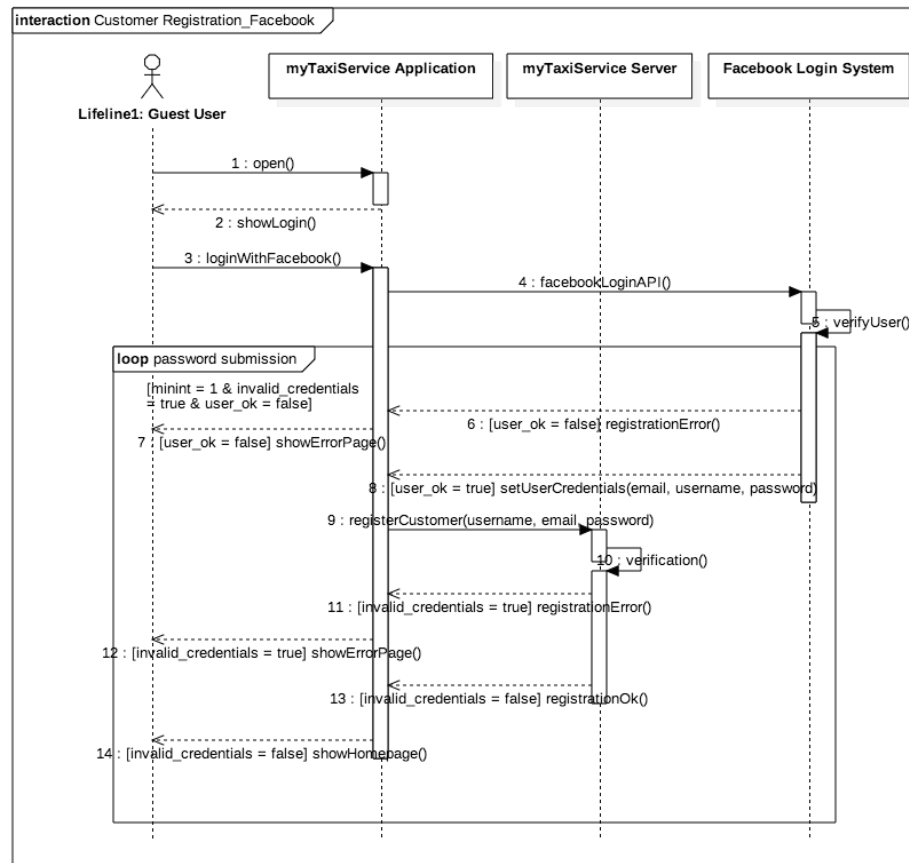


Figure 3.26: Facebook Customer Registration Interaction Diagram

3.4.3.3 Taxi Driver Registration

To accompany this diagram, read the Scenario S.3.

Subject	Description
Actors	Guest User, myTaxiService Web Application, myTaxiService Server
Preconditions	Guest User must not be already registered as a Taxi Driver
Execution	<ol style="list-style-type: none"> 1. Guest User opens the myTaxiService Web Application. 2. Guest User taps on the "Sign Up as a Taxi Driver" button. 3. myTaxiService Web Application shows the registration page. 4. Guest User fills up the registration form. 5. Guest User taps on the "Submit" button. 6. myTaxiService Web Application calls the server registration service. 7. myTaxiService Server verifies the credentials 8. myTaxiService Server replies with an ok or an error 9. myTaxiService Web Application shows to the Guest User the result of his registration.
Postconditions	The Guest User is now a Taxi Driver, he is registered in the database and he is now logged in.
Exceptions	<ol style="list-style-type: none"> 1. The email regex is not optimal (no regex for email is optimal) 2. The connection is lost or the Guest User doesn't complete the registration clicking the button.

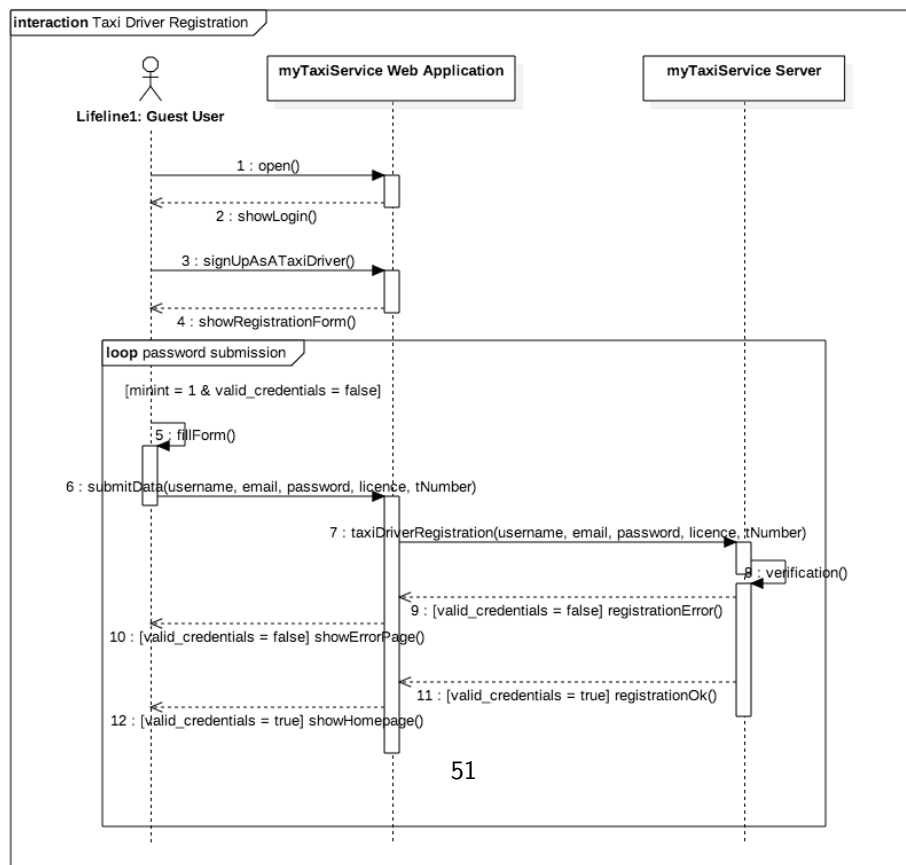


Figure 3.27: Taxi Driver Registration Interaction Diagram

3.4.3.4 Registered User Login

To accompany this diagram, read the Scenario S.4.

Subject	Description
Actors	Guest User, myTaxiService Web Application, myTaxiService Server
Preconditions	1. Registered User must be registered on the system. 2. Registered User must not be already logged in.
Execution	1. Registered User opens the myTaxiService Web Application. 2. Registered User fills the login form. 3. Registered User presses on the "Login" button. 4. myTaxiService Application submits the data to the server. 5. myTaxiService Server checks the data. 6. myTaxiService Server replies with an error or with success. 7. myTaxiService Application shows an error notification or a success one.
Postconditions	The Registered User is now logged in and is able to use the application.
Exceptions	1. The connection is lost or the Guest User doesn't complete the login clicking the button.

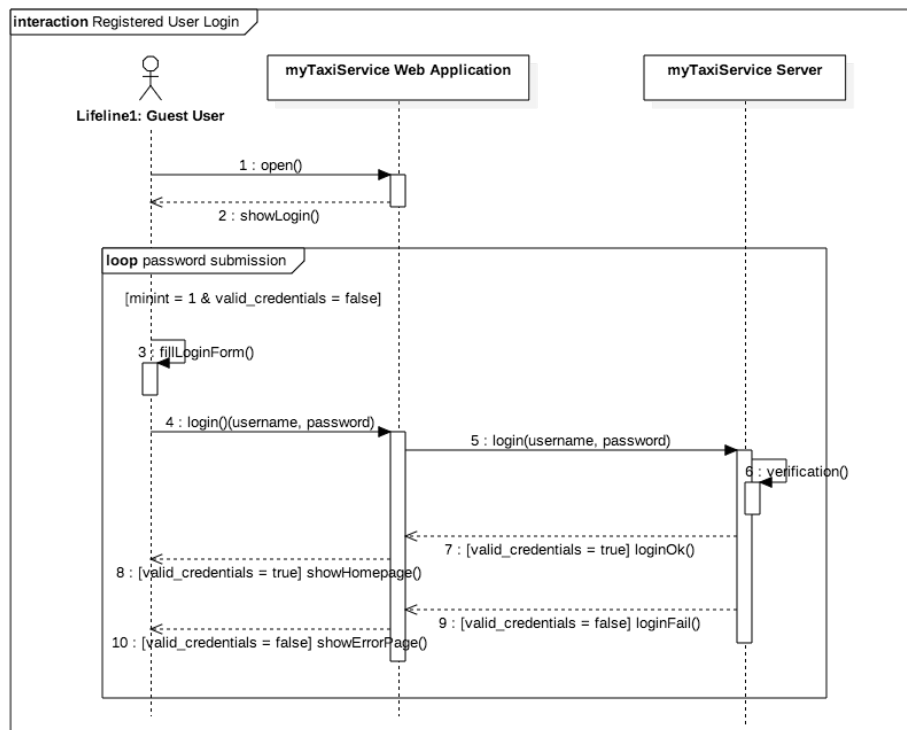


Figure 3.28: Login Interaction Diagram

3.4.3.5 Customer Facebook Login

To accompany this diagram, read the Scenario S.5.

Subject	Description
Actors	Customer, myTaxiService Mobile Application, myTaxiService-Server, Facebook System.
Preconditions	Registered User must have a Facebook account registered in the myTaxiService System.
Execution	1. Customer opens the myTaxiService Mobile Application. 2. myTaxiService Mobile Application shows the Login screen. 3. Customer clicks on the "Login with Facebook" button. 4. myTaxiService facebook login function is called and the homepage is showed to the Customer if the Facebook login is successful.
Postconditions	Customer is logged in.
Exceptions	1. The Customer disconnects before the homepage is showed.

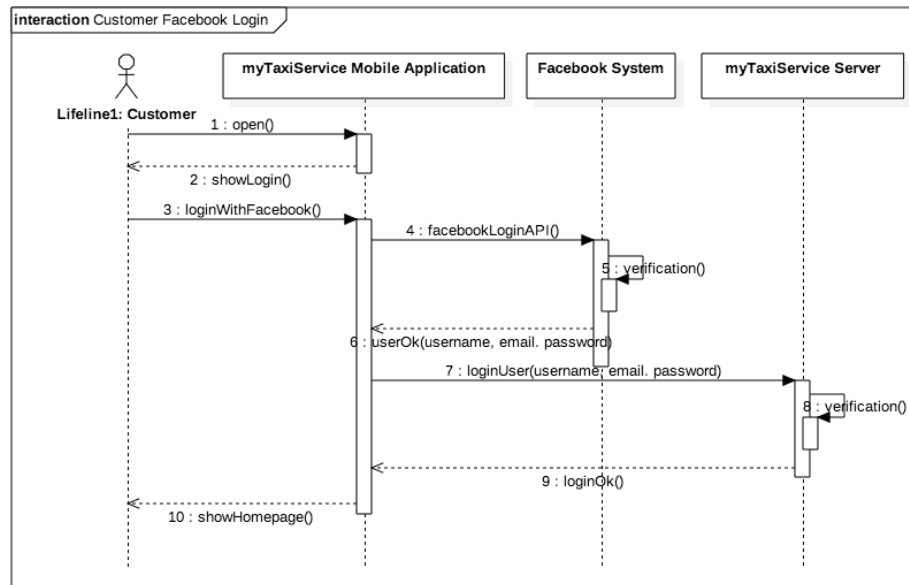


Figure 3.29: FacebookLogin Interaction Diagram

3.4.3.6 Profile Modification

To accompany this diagram, read the Scenario S.6.

Subject	Description
Actors	Registered User, myTaxiService Web Application, myTaxiService Server
Preconditions	Registered User must be logged in.
Execution	1. Registered User opens the myTaxiService Web Application. 2. Registered User taps on the "Me" button. 3. myTaxiService Web Application shows the profile page. 4. Registered User changes his credentials. 5. Registered User taps on the "Save" button. 6. myTaxiService Web Application calls the server modification function. 7. myTaxiService Server verifies the credentials 8. myTaxiService Server replies with an ok or an error 9. myTaxiService Web Application shows to the Registered User the result.
Postconditions	Registered User's credentials are now changed.
Exceptions	1. The Registered User disconnects before saving.

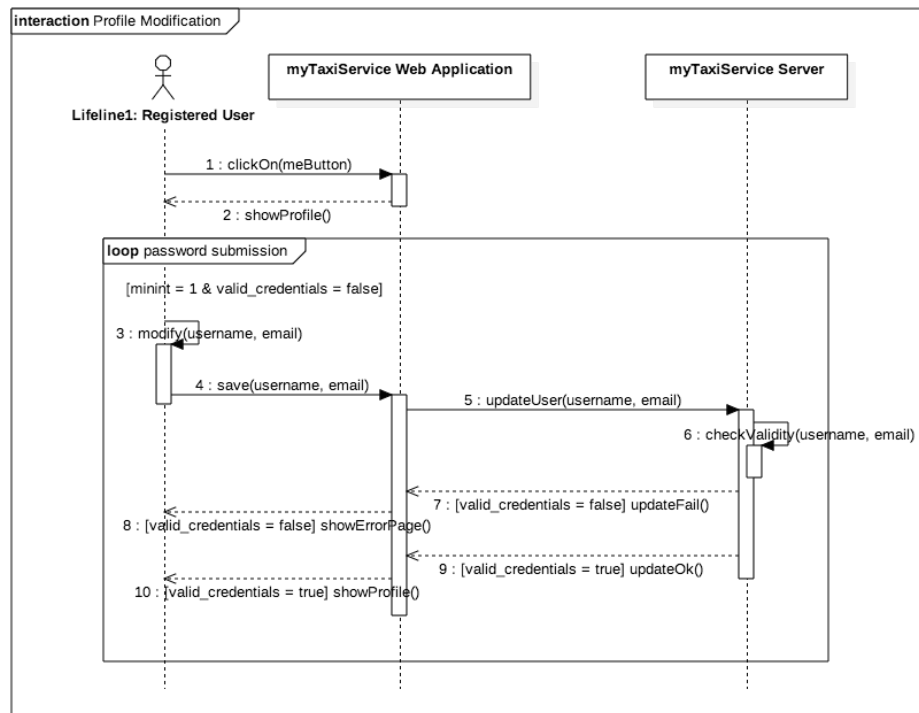


Figure 3.30: Profile Modification Interaction Diagram

3.4.3.7 Password Retrieval

To accompany this diagram, read the Scenario S.7.

Subject	Description
Actors	Registered User, myTaxiService Web Application, myTaxiService Server
Preconditions	Registered User must be actually registered.
Execution	<ol style="list-style-type: none"> 1. Registered User opens the myTaxiService Web Application. 2. Registered User taps on the "Retrieve Password" link. 3. myTaxiService Web Application shows the email field. 4. Registered User inserts his email. 5. Registered User taps on the "Retrieve" button. 6. myTaxiService Web Application calls the password retrieval function. 7. myTaxiService Server verifies the email. 8. myTaxiService Server replies with an ok or an error 9. myTaxiService Web Application shows to the Registered User the result.
Postconditions	The system has sent an email to the Registered User containing his password.
Exceptions	1. The Registered User disconnects before clicking on the "Retrieve" button.

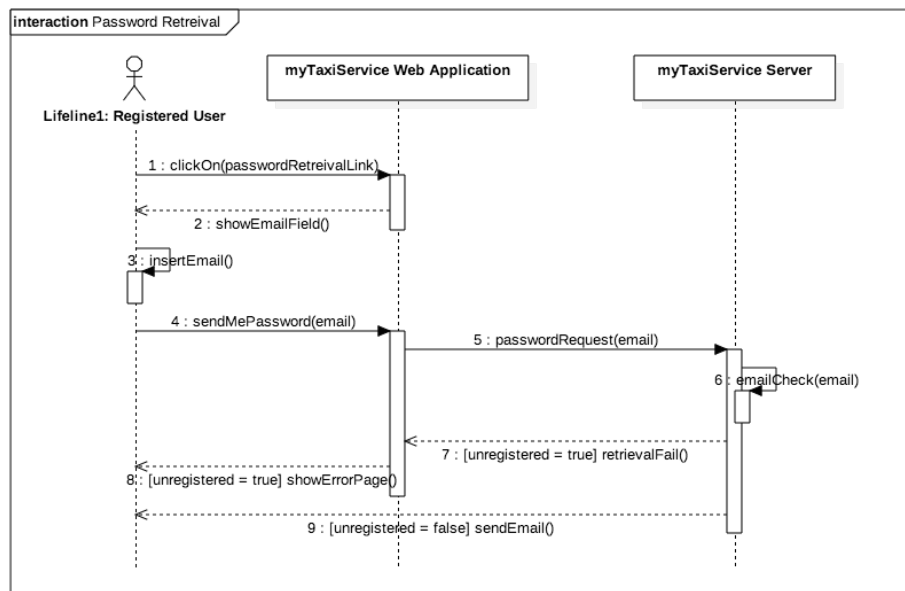


Figure 3.31: Password Retrieval Interaction Diagram

3.4.3.8 Taxi Driver Reporting

To accompany this diagram, read the Scenario S.8.

Subject	Description
Actors	Customer, myTaxiService Mobile Application, myTaxiService Server
Preconditions	Customer must be logged in.
Execution	<ol style="list-style-type: none"> 1. Customer opens the myTaxiService Mobile Application. 2. Customer taps on the "Request Taxi" button. 3. myTaxiService Mobile Application sends the requests to the Server. 4. Customer waits for the acceptance. 5. myTaxiService Mobile Application shows the acceptance notification. 6. Customer waits for the Taxi that does not arrive. 7. Customer ends the ride reporting the Taxi Driver.
Postconditions	The ride is set to ended and the Taxi Driver is reported.
Exceptions	1. The Customer disconnects before reporting.

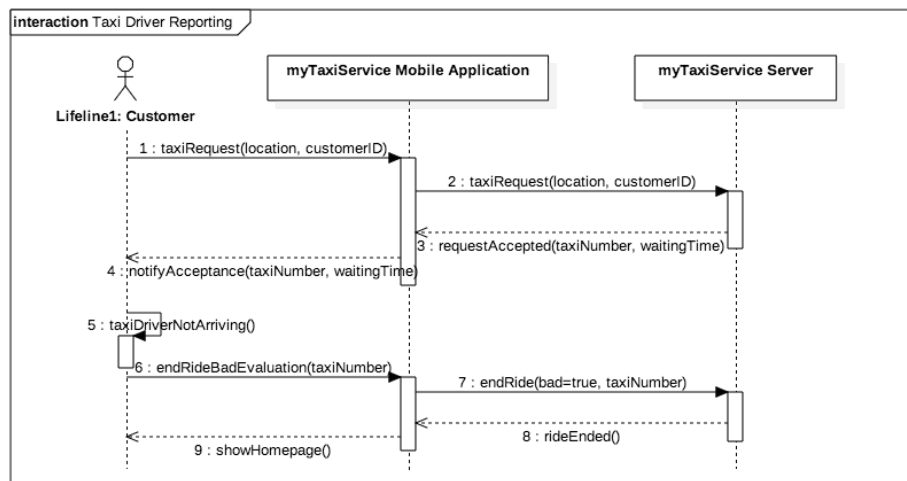


Figure 3.32: Taxi Driver Reporting Interaction Diagram

3.4.3.9 Customer Reporting

To accompany this diagram, read the Scenario S.8.

Subject	Description
Actors	Taxi Driver, myTaxiService Mobile Application, myTaxiService Server
Preconditions	Taxi Driver must be logged in.
Execution	<ol style="list-style-type: none"> 1. Taxi Driver opens the myTaxiService Mobile Application. 2. Taxi Driver receives a ride request notification from the Application. 3. Taxi Driver accepts the ride request. 4. Taxi Driver goes to the meeting location and the Customer is not there. 5. Taxi Driver ends the ride reporting an abuse.
Postconditions	The ride is set to 'ended' and the Customer is reported.
Exceptions	1. The Taxi Driver disconnects before reporting.

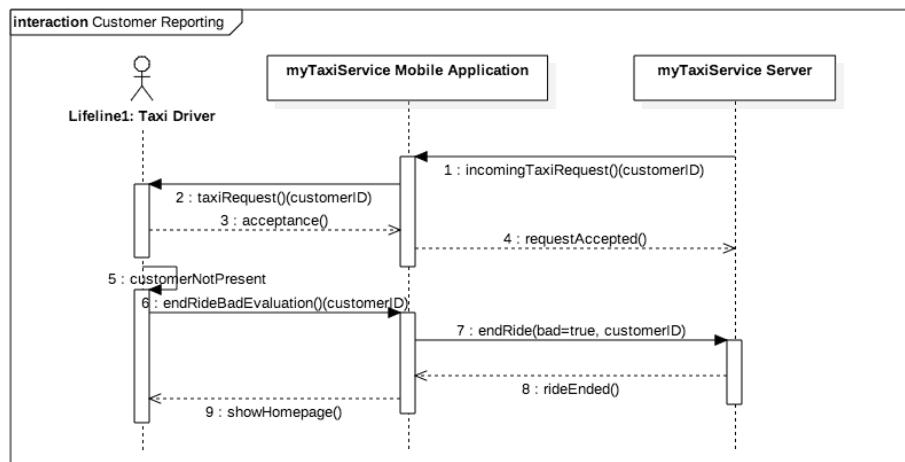


Figure 3.33: Customer Reporting Interaction Diagram

3.4.3.10 Request a Taxi

To accompany this diagram, read the Scenario S.9.

Subject	Description
Actors	Customer, myTaxiService Mobile Application, myTaxiService Server
Preconditions	Customer must be logged in, Customer must be on his homepage (see Customer Homepage).
Execution	2. Customer taps on the "Request Taxi" button. 3. myTaxiService Mobile Application sends the requests to the Server. 4. Customer waits for the acceptance. 5. myTaxiService Mobile Application shows the acceptance notification. 6. Customer waits for the Taxi.
Postconditions	The Customer has request a taxi successfully.
Exceptions	1. The Customer's GPS is off.

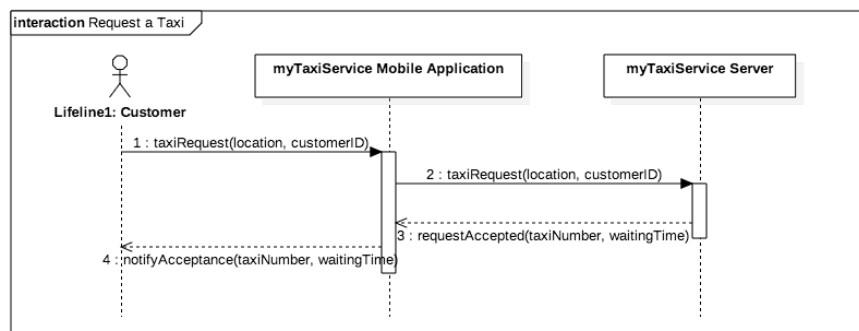


Figure 3.34: Taxi Request Interaction Diagram

3.4.3.11 Reserve a Taxi

To accompany this diagram, read the Scenario S.10.

Subject	Description
Actors	Customer, myTaxiService Web Application, myTaxiService Server
Preconditions	Customer must be on the homepage and is already logged in.
Execution	<ol style="list-style-type: none"> 1. Customer taps on the "Reserve a Taxi" button. 2. myTaxiService Mobile Application shows the reservation page. 3. Customer fills the reservation form. 4. Customer submits the data by pressing "Reserve" button. 5. myTaxiService Mobile Application calls the server function to reserve a taxi. 6. myTaxiService Mobile Application shows a success notification.
Postconditions	The Customer has reserved a taxi successfully.
Exceptions	<ol style="list-style-type: none"> 1. The Customer's GPS is off. 2. The Customer disconnects before reservation. 3. The current time is less than two hours before the reservation time.

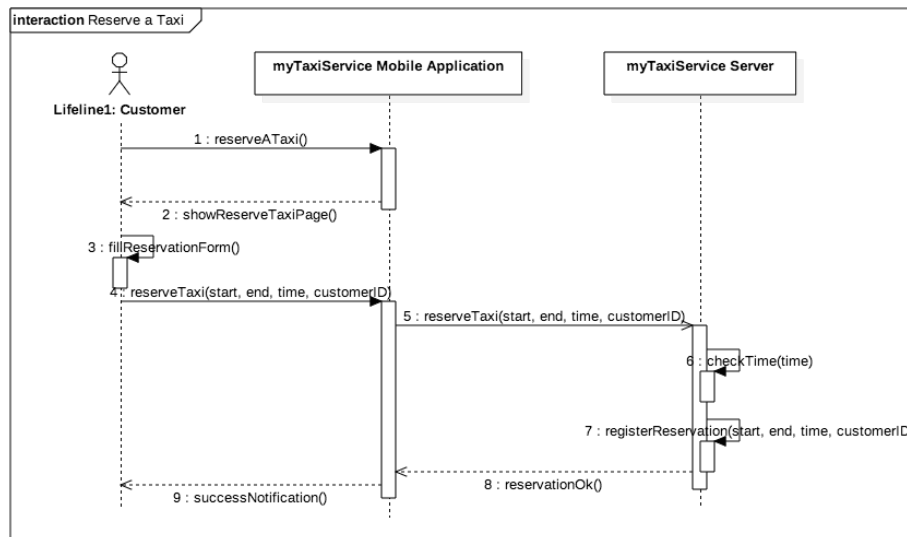


Figure 3.35: Taxi Reservation Interaction Diagram

3.4.3.12 Delete a Reservation

To accompany this diagram, read the Scenario S.11.

Subject	Description
Actors	Customer, myTaxiService Web Application, myTaxiService Server
Preconditions	Customer must be on the homepage and is already logged in.
Execution	<ol style="list-style-type: none"> 1. Customer taps on the "My Reservations" button. 2. myTaxiService Mobile Application downloads the reservation list. 3. myTaxiService Mobile Application shows the reservation list page. 4. Customer selects the reservation to delete 5. Customer deletes the reservation by pressing the "X" button. 6. myTaxiService Mobile Application calls the server function to delete a reservation. 7. myTaxiService Mobile Application shows a success notification.
Postconditions	The Customer has deleted his reservation successfully.
Exceptions	<ol style="list-style-type: none"> 1. The Customer GPS is off. 2. The Customer disconnects before the reservation.

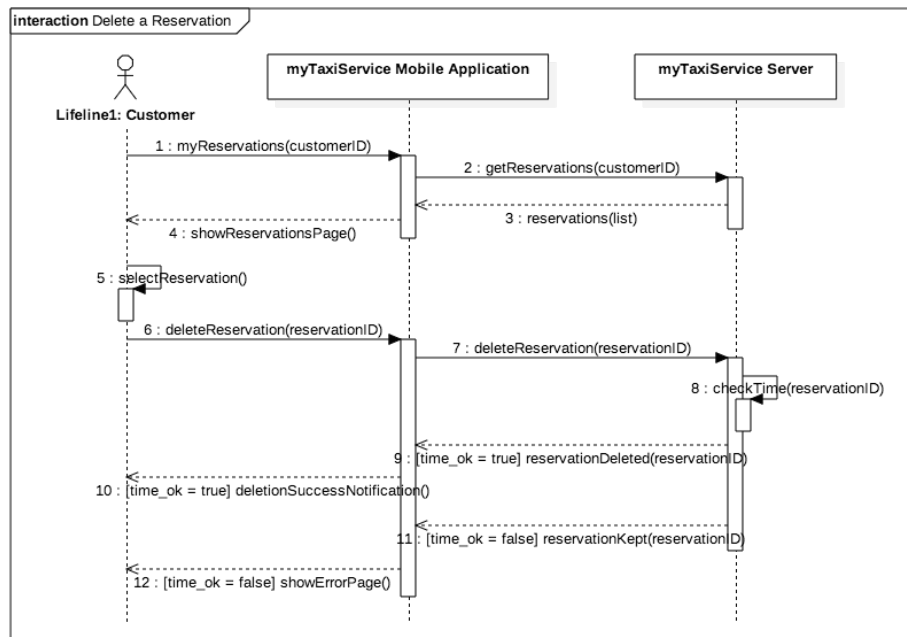


Figure 3.36: Reservation Deletion Interaction Diagram

3.4.3.13 Accept or Decline a Ride Request

To accompany this diagram, read the Scenario S.12.

Subject	Description
Actors	Taxi Driver, myTaxiService Mobile Application, myTaxiService Server
Preconditions	Taxi Driver must be on the homepage and must be already logged in. Taxi Driver must be available.
Execution	<ol style="list-style-type: none">1. myTaxiService Server notifies the Taxi Driver application of an incoming taxi request.2. myTaxiService Mobile Application notifies the Taxi Driver of the incoming request.3. Taxi Driver decides if he want to accept or if he want to decline tapping on the relative button.4. The myTaxiService Mobile Application calls the relative server function.
Postconditions	The Taxi Driver has accepted or declined a request.
Exceptions	<ol style="list-style-type: none">2. The Taxi Driver disconnects before the acceptance or the declination of the request.

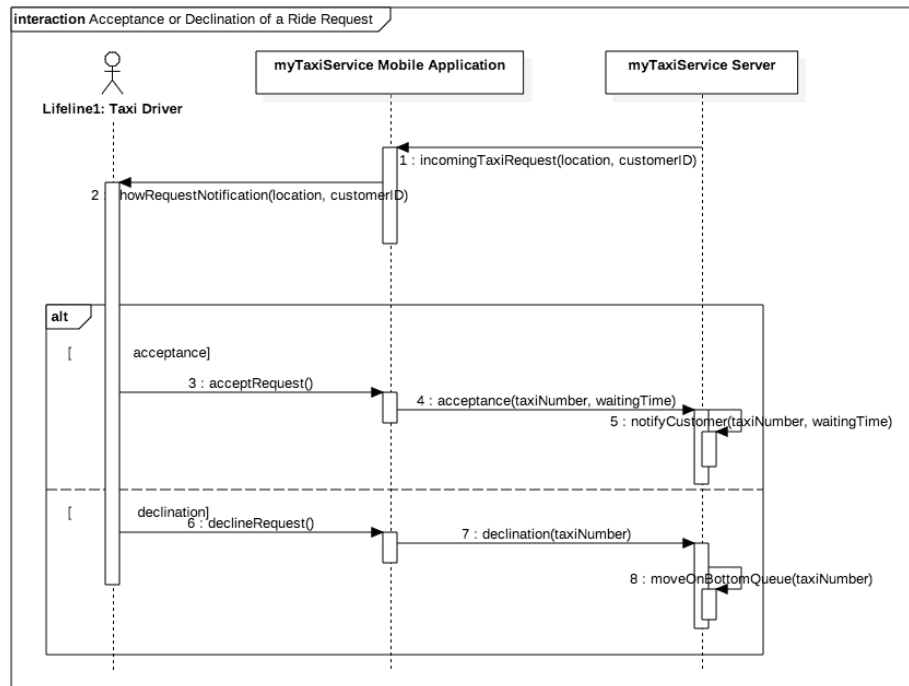


Figure 3.37: Reservation Deletion Interaction Diagram

3.4.3.14 Taxi Driver Availability Notification

To accompany this diagram, read the Scenario S.13.

Subject	Description
Actors	Taxi Driver, myTaxiService Mobile Application, myTaxiService Server
Preconditions	Taxi Driver must be on the homepage and must be already logged in. Taxi Driver must not be already available on the system.
Execution	<ol style="list-style-type: none"> 1. Taxi Driver taps on the "Available" ON/OFF button. 2. myTaxiService Mobile Application calls the relative server function. 3. myTaxiService Server sets the Taxi Driver to 'available' and adds him to the zone queue. 4. myTaxiService Server notifies the Application of the result. 5. myTaxiService Mobile Application shows to the Taxi Driver that he is now available on the system.
Postconditions	The Taxi Driver is now available for the system.
Exceptions	2. The Taxi Driver disconnects before the server response.

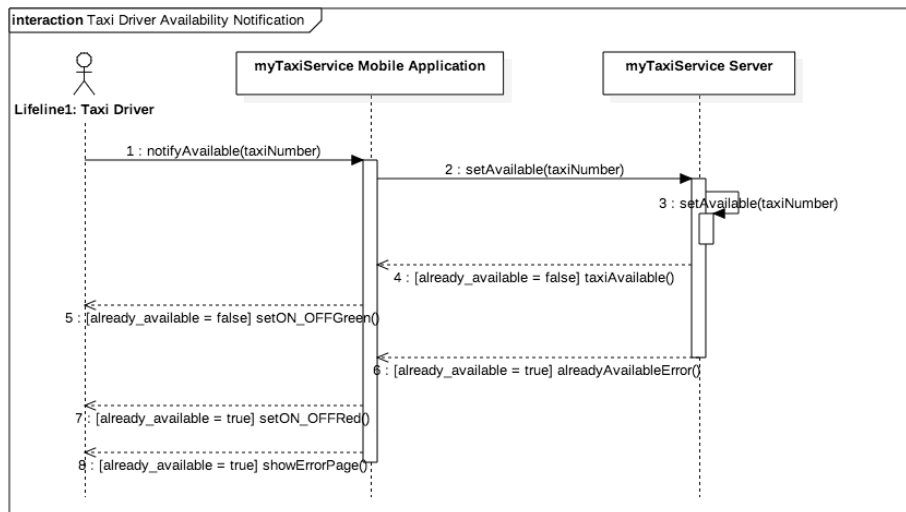


Figure 3.38: Reservation Deletion Interaction Diagram

3.4.4 State Machine Diagrams

Here is specified the behaviour of some classes presented in the Class Diagram Figure.

3.4.4.1 Taxi Request State Machine

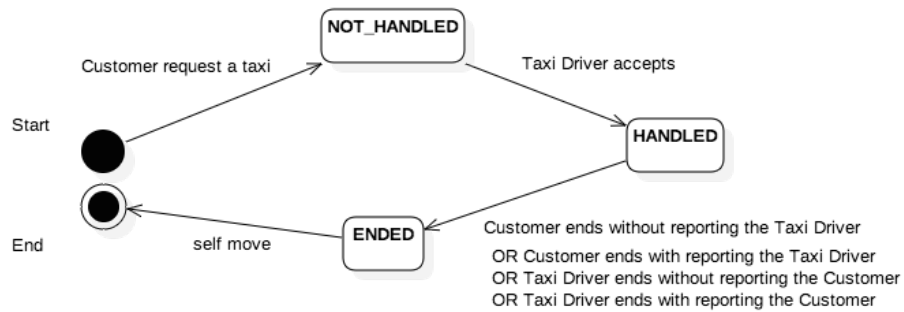


Figure 3.39: Taxi Request State Machine Diagram

3.4.4.2 Taxi Driver State Machine

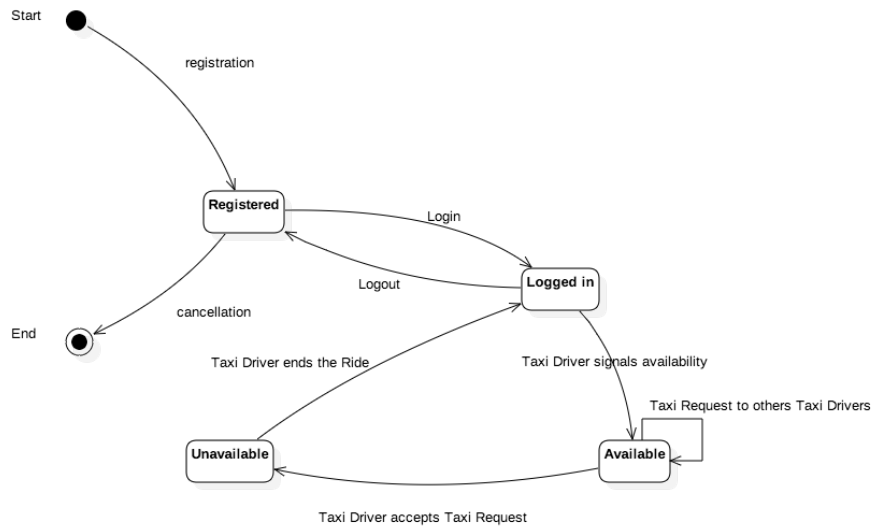


Figure 3.40: Taxi Driver State Machine Diagram

3.5 Non Functional Requirements

Here is described how the system should work to guarantee the Functional Requirements.

3.5.1 Performance Requirements

To give the best user experience when using my Taxi Service web and mobile applications, the system must guarantee enough computation performances for at least the population of the big city in which the service is implemented. Overall, it must be considered that the performances seen by the user are strictly related to his internet connection and also, indirectly, all the system performances.

3.5.2 Design Constraints

The three mobile applications will be developed with the programming languages for Android, iOS and Windows 10 that are respectively Java (for Android), Swift and C#. The web application will be developed using Javascript Frameworks like JQuery for the client side and PHP or others server-side languages. These programming languages will inherit all common languages constraints. The system also needs a DBMS to work properly and save all the users information.

3.5.3 Software System Attributes

These are the quality-attributes used to evaluate the performance of myTaxiService system.

3.5.3.1 Adaptability

The developed system must be adaptable to other Software Engineering cycles (because of updates or revisions). To accomplish this goal, it must be developed following the major software design patterns like MVC. Using these design patterns, the system is more adaptable for example in the case of adding some users types or database entities.

3.5.3.2 Availability

The system must be available every time, so it must be hosted into cloud platforms and, maybe, even in the backup server farm to ensure data redundancy.

3.5.3.3 Compatibility

Given that the mobile applications are developed in the natives programming languages of the various operative systems, the web application must be compatible (in style and functioning) with the major browsers listed in the Browsers Table.

3.5.3.4 Fault-Tolerance

The system should not crash or get into infinite loops during the wait for the user's response. To ensure this goal, for every message that will be sent to an user, there will be a Timer-Thread that will be waiting for a limited amount of time and after that the system will have to consider the user disconnected.

3.5.3.5 Maintainability

The system should be maintainable over time to perform updates or add new features. To accomplish this goal, the code must be properly documented even if there are no public API.

3.5.3.6 Modularity

To ensure more maintainability, the system must be developed in a modular way. Every functionality should depend as least as possible by the others and to do this it's recommended to use software design patterns to develop the system.

3.5.3.7 Portability

The server side of the system must be portable through different server operating systems and to do this it will be necessary to not use proprietary programming languages.

3.5.3.8 Simplicity

The application should be simple and accessible to novice users. To ensure this goal, the application interface must be as simple as possible and the main functions have to be directly accessible from the homepage.

3.5.4 Security

This subsection is intended to give a description on how the system should be developed to ensure security against accidental or malicious accesses, unauthorized uses or modifications of the system itself. myTaxiService is a Three-Tier architecture and the Application Side should be considered as the Presentation Tier, the Server Side instead, contains the Logic Tier and the Data Tier.

3.5.4.1 Application and User Interface Side

Here is implemented the Presentation Tier of the system. On the application side should be implemented a first security check on the credentials that the user has to insert. In particular:

- Passwords must be at least 8 characters long and must contain digits and alphabetic characters.
- The emails must match the regular expression:
“`^\w+@[a-zA-Z_]?\. [a-zA-Z]{2,3}$`”.

These controls are made in real time by the application to give a better user's experience avoiding the reload of the application pages, others checks are delegated to the server side to avoid users to exploit the security system. If the user's data are not valid for this layer, then the application should not call for server functions. To

call remote functions, the system must implement HTTP-POST requests instead of HTTP-GET requests.

3.5.4.2 Server Side

The server side is divided in two tiers: Logic Tier and Data Tier.

Logic tier is the one that receives application requests and queries the data tier. In this tier are implemented all the checks about the user credentials that have been sent by the application side. In particular:

- In the registration procedure, the username and the email must not be already registered into the database; passwords are encrypted with MD5 hashing;
- In the login procedure, the username must be registered and the related password must be equal to the given one to give a positive reply.

Data tier is the one that memorizes all the system's data and makes them available to the logic tier (DBMS).

To improve the security of our system, the server side is divided in two servers:

- Web Server, that contains the web application and the server application.
- Database Server, that contains the interface to the database.

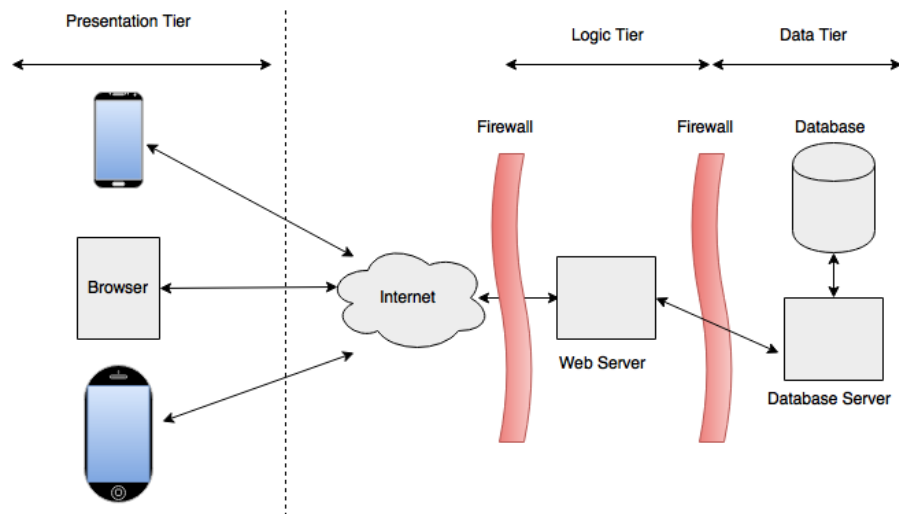


Figure 3.41: Server Architecture

Chapter 4

Appendix

4.1 Alloy

This section is intended to present a model of the system and his constraints using the Alloy modeling language. We have divided the Alloy code in four main sections:

- Signatures, here are modeled the entities of our system. In the first part there are primitive entities like boolean or strings and in the second part there are the system entities.
- Facts, here are modeled all the constraints for our model.
- Predicates and Assertions, here are modeled all the checks to control the consistency of our model.
- Generated World, here there is an example of entities of our system generated by the "show" predicate in the Predicates and Assertions section.

4.1.1 Signatures

Here there are attached screenshots of the Alloy code about entities:

```

module myTaxiService

/*****
/*          SETS (ENUMERATIONS)          */
*****/
enum RideState { NOT_HANDLED, HANDLED, ENDED }

/*****
/*          UTILITY SIGNATURES          */
*****/

abstract sig boolean{}
one sig True extends boolean {}
one sig False extends boolean {}

abstract sig Time{}

sig Integer{}
sig Strings{}

/*****
/*          MODEL CLASSES          */
*****/

/* Position has */
/* No constraints */
sig Position{
    long : one Integer,
    lat  : one Integer,
    isIn : one Zone
}

/* Zone is a piece of the city */
/* it is a quadrilateral shape, so it's characterized by 4 corners */
sig Zone{
    upperLeftCorners : one Position,
    upperRightCorners : one Position,
    lowerLeftCorners : one Position,
    lowerRightCorners : one Position
}

```

Figure 4.1: Signatures 1

```

/* RegisteredUser is the basic user class in the model. */
/* it has an username */
/* it has a password */
/* it has an email */
/* it has a position */
/* it has a zone */
/* FACTS: */
/* - email must be unique */
/* - username must be unique */
/* - currentPosition of the user is in a certain zone if the user is in that zone. */
abstract sig RegisteredUser{
  userName      : one Strings,
  email         : one Strings,
  psw           : one Strings,
  badEvaluCount : one Integer,
  currentPosition : one Position,
  currentZone    : one Zone
}

/* customer is the specialization of registered user.
it has a customerId

FACTS
customerId must be unique
*/
sig Customer extends RegisteredUser{
  customerId : one Integer
}

/* taxi driver is the specialization of registered user.
it has licenseNumber
it has availability
it has taxiId
it has a priority

FACTS
- license number must be unique
- taxiId must be unique
*/
sig TaxiDriver extends RegisteredUser{
  licenseNumber : one Integer,
  availability   : one boolean,
  taxiId        : one Integer,
  priority       : one Integer
}

```

Figure 4.2: Signatures 2


```

/*
  it has a "from" position

  FACTS
  - if state is NOT_HANDLED taxiDriver is empty
  - if state is HANDLED or ENDED taxiDriver is not empty
*/
abstract sig Ride{
  from      : one Position,
  state     : one RideState,
  requestingUser : one Customer,
  taxiDriver : lone TaxiDriver,
}

/*
  normal ride is a taxi request made by a customer
  FACTS
  - from position must be the requestingUser current position.
  - the number of normal rides associated to a requestingUser with state equal
    to NOT_HANDLED is only one.
*/
sig NormalRide extends Ride{
}

/* Reservation is a ride programmed for the future
  it has a position fo finish
  when

  FACTS
  - if deleted is true, then state must be NOT_HANDLED
  - "from" must be different from "to"
  - two different reservation with the same "when" must have different drivers
    and customer
*/
sig Reservation extends Ride{
  to      : one Position,
  when    : one Time,
  deleted : one boolean
}

```

Figure 4.3: Signatures 3

4.1.2 Facts

Here there are attached screenshots of the Alloy code about rules:

```

/*****
/*          FACTS          */
*****/

//
//REGISTERED USER FACTS
// - username must be unique
fact userNameUnicity{
  no disj user1, user2 : RegisteredUser | user1.userName = user2.userName
}

// - email must be unique
fact eMailUnicity{
  no disj user1, user2 : RegisteredUser | user1.email = user2.email
}

/** - currentPosition of the user is in a certain zone if the user is in that zone. */
fact positionInZone{
  all r : RegisteredUser | some p : Position |
    p = r.currentPosition implies p.isIn = r.currentZone
}

//
// CUSTOMER FACTS
// - customerID must be unique
fact customerIDUnique{
  no disj c1, c2 : Customer | c1.customerId = c2.customerId
}

//
// RIDES FACTS
// - if state is NOT_HANDLED taxiDriver is empty
fact RideStateNotHandled{
  all r : Ride | r.state = NOT_HANDLED implies no r.taxiDriver
}

// - if state is HANDLED or ENDED taxiDriver is only one
fact RideStatHandledEnded{
  all r : Ride | r.state = HANDLED or r.state = ENDED implies #r.taxiDriver = 1
}

// - if state is HANDLED then taxiDriver availability is FALSE
fact taxiDriverNotAvailableIfHandled{
  all r : Ride | r.state = HANDLED implies r.taxiDriver.availability = False
}

// - if two rides has the same TaxiDriver and one is HANDLED then the second is ENDED
fact taxiDriverNotInTwoTaxi{
  all disj r1, r2 : Ride |
    (r1.taxiDriver = r2.taxiDriver and r1.state = HANDLED) implies r2.state = ENDED
}

// - if the ride is HANDLED driver and customer are in the same zone
fact taxiDriverHandledZone{
  all r : Ride |
    r.state = HANDLED implies r.taxiDriver.currentZone = r.requestingUser.currentZone
}

```

Figure 4.4: Facts 1

```

//
// NORMAL RIDES FACTS
// - from position must be the requestingUser current position if the ride is not handled.
fact rideStartIsCustomerCurrentPosition{
  all r : NormalRide | all c : Customer |
    (r.requestingUser = c and r.state = NOT_HANDLED) implies r.from = c.currentPosition
}
// - the number of normal rides associated to a requestingUser with state equal to NOT_HANDLED
// is only one.
fact onlyOneForCustomerNotHandled{
  no disj r1, r2 : NormalRide |
    r1.requestingUser = r2.requestingUser
    and (r1.state = NOT_HANDLED or r1.state = HANDLED)
    and (r2.state = NOT_HANDLED or r2.state = HANDLED)
}

//
// RESERVATION RIDES FACTS
// - if deleted is true, then state must be NOT_HANDLED
fact deletedIsNotHandled{
  all r : Reservation | r.deleted = True implies r.state = NOT_HANDLED
}
// - "from" must be different from "to"
fact fromDifferentTo{
  all r : Reservation | r.from != r.to
}
// - two different reservation with the same "when" must have different drivers and customers
fact differentReservations{
  all r1 : Reservation | all r2 : Reservation |
    (r1 != r2 and r1.when = r2.when)
    implies ((r1.taxiDriver != r2.taxiDriver) and (r1.requestingUser != r2.requestingUser))
}

//
// TAXI RELATED FACTS
// - Taxi license must be unique
fact licenseNumberUnicity{
  no disj user1, user2 : TaxiDriver | user1.licenseNumber = user2.licenseNumber
}
// - Taxi number must be unique
fact taxiIDUnicity{
  no disj user1, user2 : TaxiDriver | user1.taxiId = user2.taxiId
}

```

Figure 4.5: Facts 2

4.1.3 Predicates and Assertions

Here there are attached screenshots of the Alloy code about checks:

```

/*****
/*          PREDICATES          */
*****/

pred show{
  #Reservation >= 3
  #NormalRide >= 3
  #TaxiDriver >= 2
  #Customer >= 2
}

run show for 10

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

//OK
//check checkState for 5
assert checkState{
  all r : Ride | #r.taxiDriver = 1 implies r.state = HANDLED or r.state = ENDED
}

//OK
//check checkZone for 5
assert checkZone{
  all r : NormalRide | r.state = HANDLED
    implies r.taxiDriver.currentZone = r.requestingUser.currentZone
}

//OK
//check tDriverAvailability for 5 but 1 NormalRide
assert tDriverAvailability{
  all t : TaxiDriver | all r : NormalRide | all c : Customer |
    (r.taxiDriver = t and r.state = HANDLED and r.requestingUser = c)
    implies c.currentZone = t.currentZone
}

//OK
//check deletedCannotBeEnded for 5
assert deletedCannotBeEnded{
  no r : Reservation | r.deleted = True and r.state = ENDED
}

```

Figure 4.6: Predicates and Assertions 1

4.1.4 Generated World

Here is the attached a screenshot of the entities generated by the "show" predicate:



4.2 Software and Tools

To create this document we have used some common software:

- StarUML ([link](#)) used to create UML models.
- Alloy Analyzer 4.2 ([link](#)) used to model our system with Alloy language.
- Balsamiq Mockups ([link](#)) used to create mockups images.

To write LaTeX code, we have used different software:

- Simone Deola: TexShop, provided with the MacTex package ([link](#))
- Davide Cremona: Sublime Text editor ([link](#)) with LaTeXTools ([link](#)) and the Basic Package of MacTex ([link](#))

4.3 Hours of Work

- Simone Deola: ~38 hours
- Cremona Davide: ~38 hours