myTaxiService
Requirements Analysis and Specification Document

Belluschi Marco, Cerri Stefano, Di Febbo Francesco

November 5, 2015

Contents

Chapter 1

Introduction

1.1 Purpose

This document represent the Requirement Analysis and Specication Document (RASD). The main goal of this document is to completely describe the system in terms of functional and non-functional requirements, analyse the real need of the customer to modelling the system, show the constraints and the limit of the software and simulate the typical use cases that will occur after the development. This document is intended to all developer and programmer who have to implement the requirements, to system analyst who want to integrate other system with this one, and could be used as a contractual basis between the customer and the developer.

1.2 Scope

The system described in this document is a taxi service for large cities. The main goals of the system are: 1) simplify the access of passangers to the service 2) guarantee a fair management of taxi queues. The system is composed by a web application, a mobile application and a web server.

There are three types of actors that can use the system: visitors, taxi drivers and passengers. Visitors have only two operations allowed: log in or sign in. Passengers can use both the web application and the mobile application to request a taxi. Taxi drivers use only the mobile application to modify their status and to confirm to the system that they are going to take care of a certain request from a certain passenger.

The system, when a passenger request a taxi, informs an available taxi driver (FIFO mode) about the current position of that passenger. At this time the taxi driver has two options:

- accept : the system sends a notification to the passenger with the estimated waiting time

- reject : the system searches for another available taxi driver

The system allows also a passenger to:

- reserve a taxi by specifying the origin and the destination of the ride
- share a taxi with others (if possible) by specifying all the rides that he/she wants to share. In this case the system defines the cost of the ride for each passenger

Besides the specific user interfaces for passengers and taxi drivers, the system offers also APIs to enable the development of additional services on top of the basic one.

1.3 Definitions, acronyms, and abbreviations

Definitions

- User: person that uses the service applications.
- Visitor: user that has not registered nor logged in.
- Registered user: user that has registered to the service.
- Passenger: passenger registered to the service.
- Taxi driver: taxi driver registered to the service.
- System: the union of software and hardware to be developed and implemented.

Acronyms

Abbreviations

1.4 References

- Software Engineering 2 Project AA 2015/2016: Project Description And Rules
- Software Engineering 2 Project AA 2015/2016: Assignments 1 and 2 (RASD and DD)
- Software Engineering 2 Project AA 2015/2016: RASD-meteocal-example1
- Software Engineering 2 Project AA 2015/2016: RASD-meteocal-example2

1.5 Overview

This document is essentially structured in three parts:

- Section 1: Introduction: it gives a description of the document and some basical information about the system. It also identifying the stakeholders and the actors involved.
- Section 2: Overall Description: it gives general information about the software and hardware product, constraints and assumptions.
- Section 3: Specific Requirements: this is the core of the document. It describes the functional and non-functional requirements combined with some scenarios. There is also a class diagram that gives an overall representation of the system.

Chapter 2

Overall Description

2.1 Product perspective

The system is composed by a web application, a mobile application and a web server. The web application runs on most common browsers, namely Chrome, Internet Explorer, Firefox, Safari. It needs a web server that supports PHP. The mobile application needs a platform supporting Android, iOs or Windows Phone. Both applications interact with a DBMS.
Additional functionalities are provided through the use of APIs or interfaces, i.e. taxi reservation and the taxi sharing option.

2.2 Product functions

The system allows different kinds of user to perform different actions. In particular:

- Visitors can simply register or log in, thus becoming either a passenger or a taxi driver user.
- Passengers can request, reserve and share taxi rides.
- Taxi drivers can modify their availability status and respond (accept/refuse) to impeding ride requests.

2.3 User characteristics

Registered users can be either passengers or taxi drivers.
The system wants to give both an easy way to interact, thus optimizing the taxi service. To do so, passengers must be able to install and use the mobile application, or use the web application. On the contrary, taxi drivers can only install and use the mobile app; besides, their cellphone must be provided with a GPS. All users must have access to the Internet.

2.4 Constraints

2.4.1 Regulatory policies

myTaxiService is a service provided by the public company responsible for public transportation in the city. The user, who reaches this service by web or mobile application, has to agree to License Agreement rather than Privacy policy and Terms of use at registration.
The user access and use of the services constitutes his/her agreement to be bound by these Terms, which establishes a contractual relationship between him/her and myTaxiService. If user does not agree to these Terms, he/she may not access or use the services. myTaxiService may immediately terminate these Terms or any services with respect to him/her, or generally cease offering or deny access to the Services or any portion thereof, at any time for any reason.
myTaxiService collects the information provided by the user, for example when creating or making changes to services on demand, through contact with customer service or during other communications. This information may include: name, email, phone number, mailing address, profile picture, payment method, products required (for service delivery), delivery receipts and other information user choose to provide. The personal data will be used only to provide the services requested.
User is responsible for obtaining the data network access necessary to use the services. User mobile network's data and messaging rates and fees may apply if he/she accesses or uses the services from a wireless-enabled device. User is responsible for acquiring and updating compatible hardware or devices necessary to access and use the service and applications and any updates thereto.
myTaxiService does not guarantee that the services, or any portion thereof, will function on any particular hardware or devices. In addition, the services may be subject to malfunctions and delays inherent in the use of the Internet and electronic communications.

2.4.2 Hardware limitations

myTaxiService defines the minimum requirements for using web and mobile applications.

- *Web application*
  Supported minimum version browsers: Chrome 25, Internet Explorer 10, Firefox 20, Safari 25. Other browsers may also work
  Web access at the minimum speed of 1Mbps

- *Mobile application*
  Operating system: Android, iOS, Windows Phone
  Memory: 512MB RAM
  Hard drive: 50MB of free space
  GPS navigation system (only for taxi drivers)
  Web access at the minimum speed of 1Mbps

2.4.3 Interfaces to other applications

myTaxiService is a stand-alone application and does not have to meet any interface to other applications. It is available on web and mobile stores.

2.4.4 Parallel operation

myTaxiService supports parallel operations cause of the nature of service. Many users can access to the service at same time thus system and database have to work with parallel requests.

2.4.5 High-order language requirements

myTaxiService requires the following high-order languages based on different platforms.

- *Web*
  HTML 5 and CSS 3 standards.
- *Android*
  Java 8
- *iOS*
  Swift 2.0
- *Windows Phone*
  C# 6.0

2.4.6 Reliability requirements

myTaxiService relies on network connections thus reliability issues are equivalent to performance issues. However, the applications should not corrupt server data as a result of its actions. The system has to guarantee whole-time availability.

2.4.7 Criticality of the application

myTaxiService relies on network systems and servers. Scheduled downtime is acceptable. This system requires a generator backup and redundant power in the event of failover.

2.4.8 Safety and security considerations

myTaxiService guarantees secure communications through AES encryption algorithms.

2.5 Assumptions and dependencies

2.5.1 Assumption

- Passenger requests a ride from web or mobile applications
- Passenger sets a correct meeting point
- Passenger sets a correct destination
- Taxi driver reaches the meeting point
- Taxi driver picks up the correct passenger
- Accurate taxi driver's locations are known by GPS
- Taxi driver reports correctly his availability
- The city is divided in taxi zones
- The taxi queue in a zone contains only taxi drivers available in that zone
- Taxi driver confirms or denies a passenger request call

2.6 Apportioning of requirements

Chapter 3

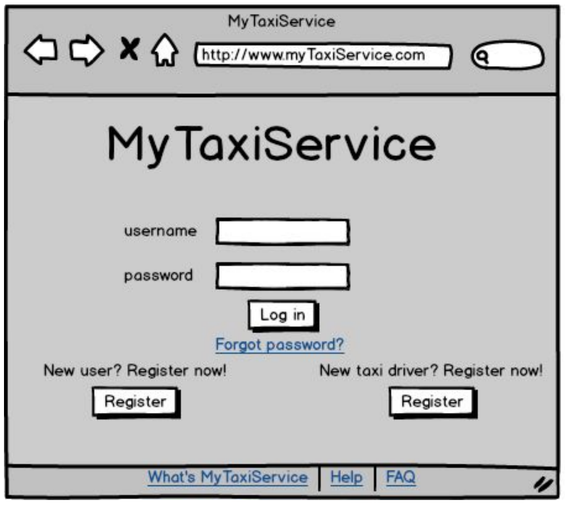
Specific Requirements

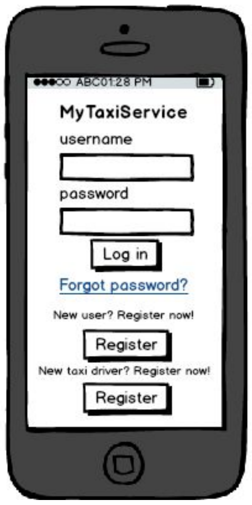
3.1 External interface requirements

3.1.1 User interfaces

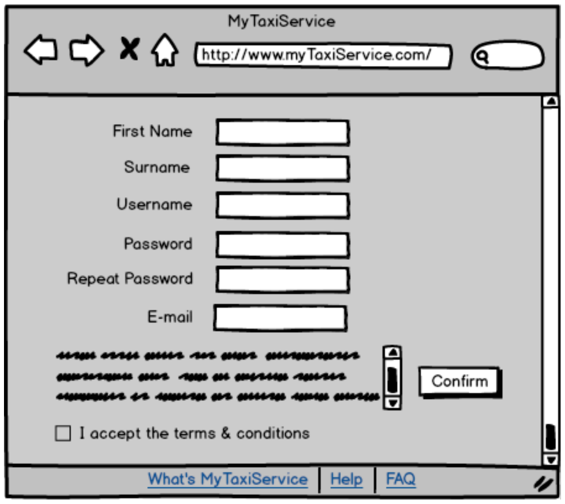
The interface of MyTaxiService can be for web application and mobile application. Here will be presented some of the most important pages and screens of MyTaxiService.

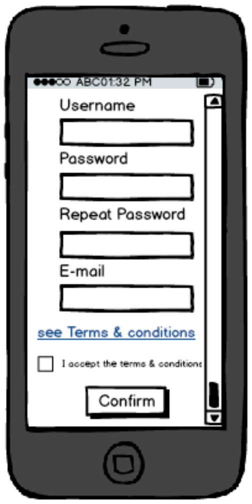
Log in: In the figure below is shown MyTaxiService's homepage



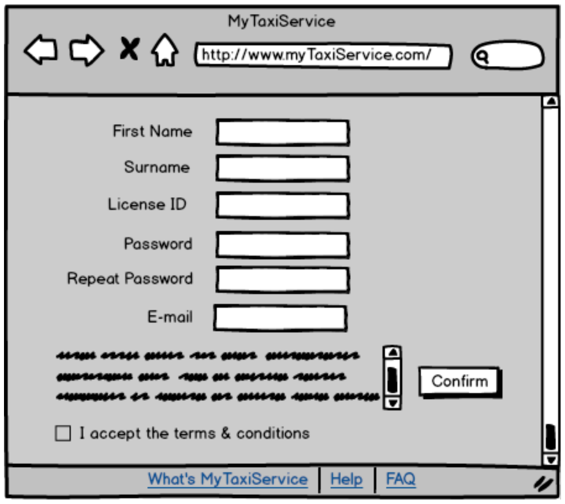


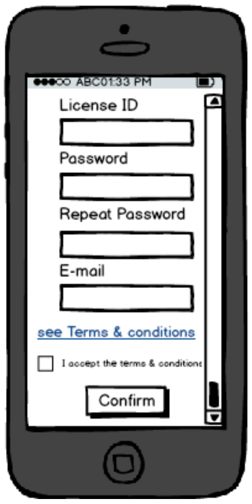
Registration passenger: View of the visitor that wants to register as a passenger



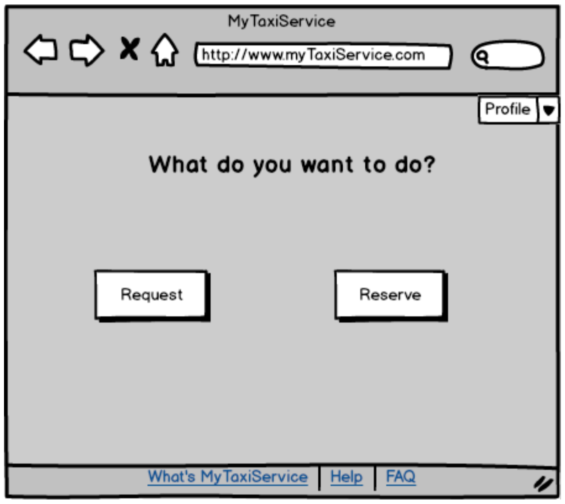


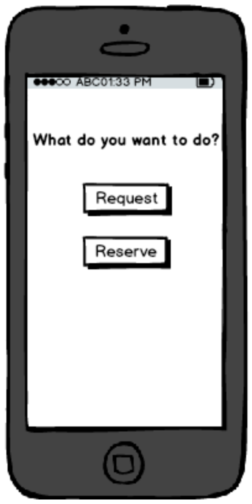
Registration taxi Driver: View of the visitor that wants to register as a taxi driver



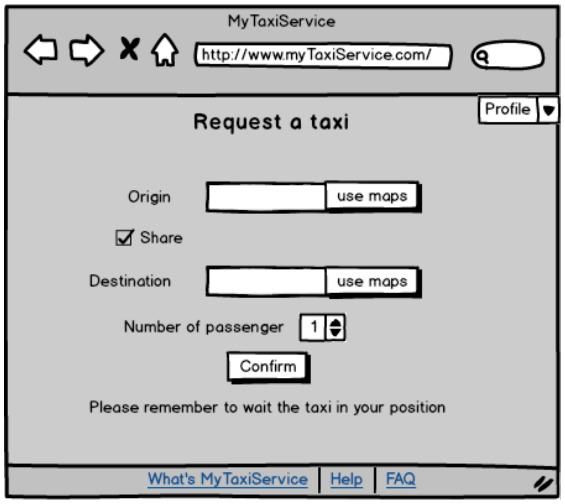


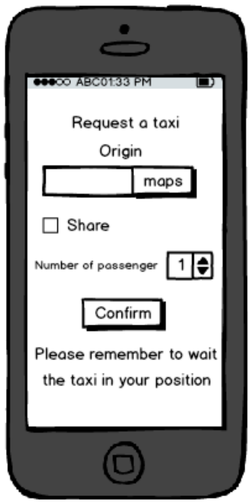
Passenger view: View of the passenger



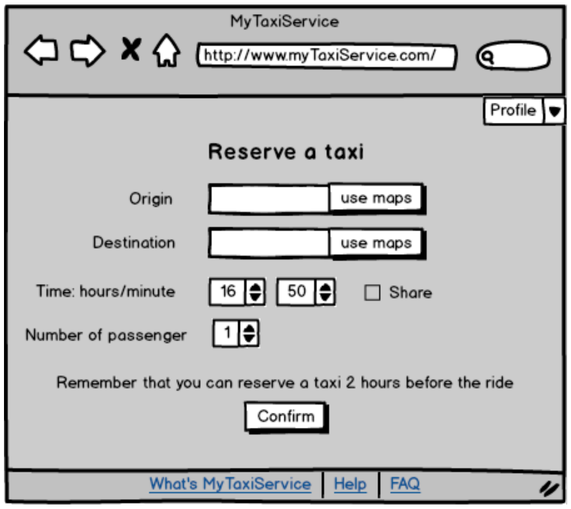


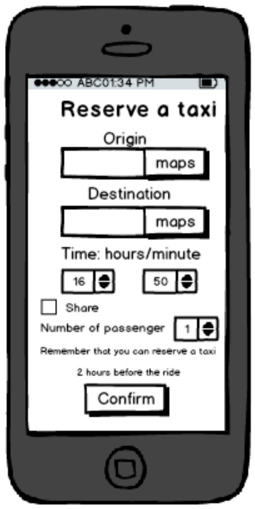
Request a taxi: View of the passenger when he/she requests a taxi



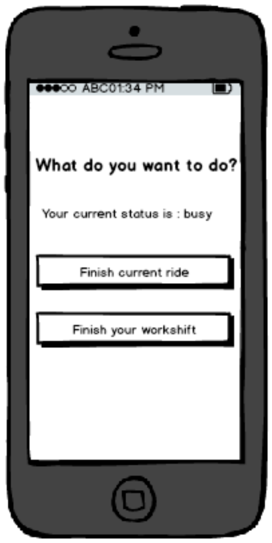


Reserve a taxi: View of the passenger when he/she reserves a taxi

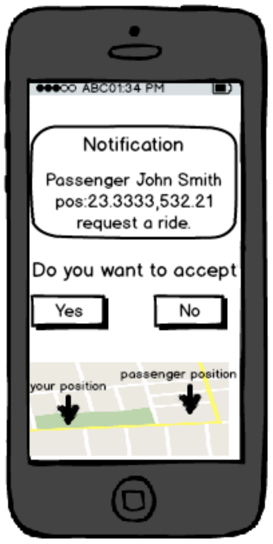




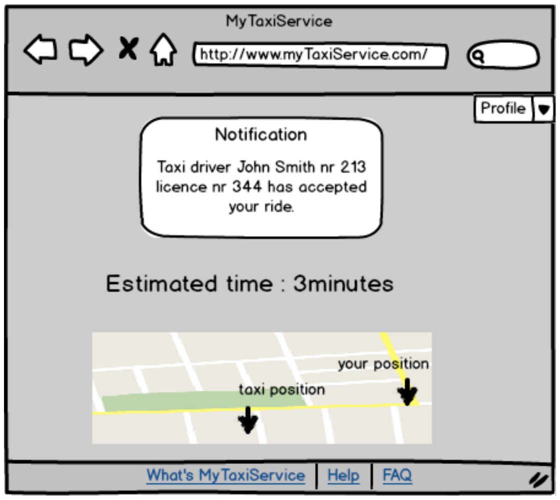
Taxi driver view: View of the taxi driver

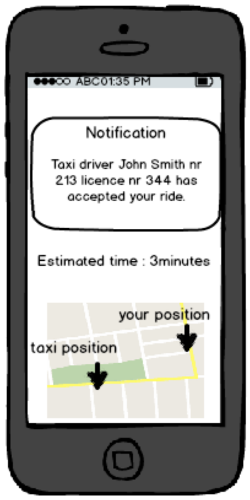


Taxi driver notification: Notification that the taxi driver, choosen by the system, sees when a passenger request a ride.



Passenger notification : Notification that the passenger see when a taxi accept the ride


MyTaxiService
http://www.myTaxiService.com/
Profile
Notification
Taxi driver John Smith nr 213 licence nr 344 has accepted your ride.
Estimated time : 3minutes
your position
taxi position
What's MyTaxiService
Help
FAQ


●●●○○ ABC01:35 PM
Notification
Taxi driver John Smith nr 213 licence nr 344 has accepted your ride.
Estimated time : 3minutes
your position
taxi position

3.1.2 API interfaces

MyTaxiService uses Google Maps APIs in order to show to the passengers and to the taxi driver their position in the city. This API is continuos update and works for all the OS and browsers web supported by MyTaxiService. More information are available on the site: https://developers.google.com/maps/

3.1.3 Hardware interfaces

MyTaxiService doesn't support any hardware interfaces.

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/

3.1.5 Communication interfaces

- Protocol: TCP Service: HTTPS Port : 443
- Protocol: TCP Service: HTTP Port : 80
- Protocol: TCP Service: DBMS Port : 9247

3.2 Functional requirements

3.2.1 Registration

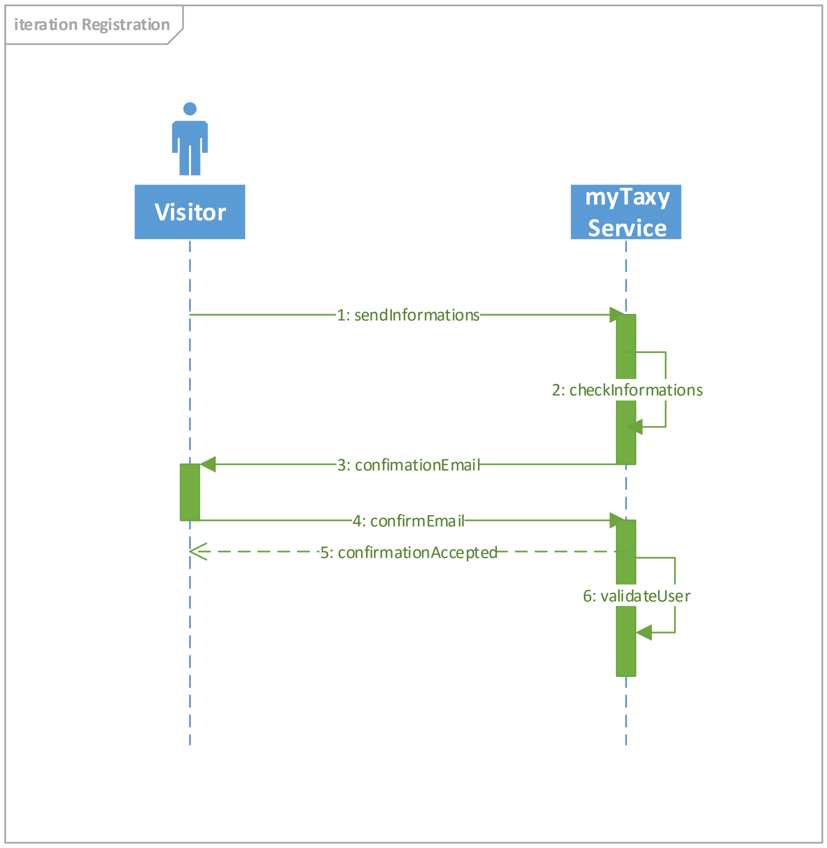
Purpose Visitors can register to myTaxiService through the web or mobile application. They can register either as a passenger or as a taxi driver.
In both cases, this operation requires the visitor to fill a registration form with personal data and accept myTaxiService terms and conditions, including personal data policies, according to local law. In case of registration as a taxi driver, the system requires the visitor more info, including proof of the possession of a valid taxi driver license.
If any of the previous requirements are not met or any input is invalid, the registration fails and the system asks the visitor to repeat the process. Otherwise, a verification email is sent to the provided email address: from that email the visitor can confirm his new account and successfully end the registration process.

Scenarios

1. Alex is a student. He has heard about myTaxiService and, finding it an easy way to travel, wants to subscribe to it.
   Therefore, he access to the homepage of the web application, clicks "Register", then chooses "Passenger". He fulfills the form, accepts the terms and conditions, and clicks "Confirm". However, the system cannot verify Alex's info because the confirmation password does not match with the first one. It therefore asks Alex to write it again. This time Alex fills the form correctly, then clicks "Confirm". The system verifies his info, then sends Alex a verification email to the submitted email address. Alex checks his mailbox, opens the new mail and clicks on the link inside it, redirecting him back to the web application of myTaxiService. The system informs him that the registration has successfully ended. He can now log in as a passenger user.

2. Bob is a taxi driver. His company recommends him to subscribe to myTaxiService, in order to make his work easier and improve the taxi service. Therefore, he downloads and installs the mobile app of myTaxiService on his mobile phone, then opens it. He taps "Register", then chooses "Taxi Driver". He inputs all the required data, including his driver license ID, accepts the terms and conditions and confirms. The system verifies the submitted info and sends Bob a confirmation email. Bob checks his mailbox, opens the new mail and taps on the link inside. The system informs him that the registration has successfully ended. He can now log in as a taxi driver user.

Diagrams



Functional Requirements

- Visitors can register either as passengers or as taxi drivers.
- Visitors can abort the registration process at any time.
- The link in the confirmation email must be clicked within 1 day, otherwise the registration is deleted along with the visitor's info.
- Registration forms contain the following info (fields):
  - email address
  - username
  - password
  - password confirmation
  - name

    - surname
    - (*) address
    - (*) telephone number
    - (**) taxi license ID
    - (**) taxi plate number
    - (**) taxi code

  All fields must be contain valid inputs.
  Fields marked with (*) are not mandatory.
  Fields marked with (**) are only for taxi driver registrations.

- email address and username cannot be the same as ones from other myTaxiService users.
- password must contain at least 8 characters.
- password and password confirmation must match.

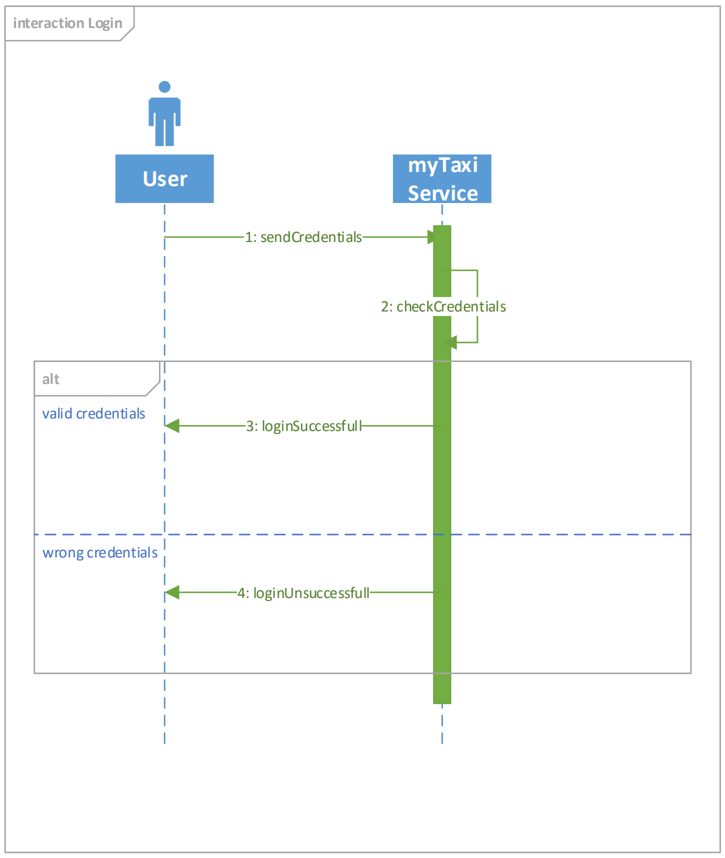
3.2.2 Login

Purpose Visitors on myTaxiService website or mobile application may access to an existing registered user account providing its corresponding username (or email address) and password. In case the submitted info do not match with any existing account info, the system notifies the visitor that the username/email address doesn't exist, or that it exists, but the submitted password is wrong.
In case a user forgets his/her password, the system allows him/her to retrieve it, automatically creating a new password, setting it as the user's one and sending it to the provided email address.

Scenarios

1. Carl is a passenger user. He opens myTaxiService website, but can't remember his password to access the service. Therefore, he clicks on "Forgotten password?". The system asks him for the email address or username he provided at registration. He writes it down and clicks "Confirm". The system verifies the existence of the submitted email address, then creates a new password and sends it in an email to the submitted email address.
2. Daisy is a passenger user, familiar with the myTaxiService website. She wants to use the mobile app, too. Knowing that she can enter either username or email and password, she fills both fields and clicks on "Log in". The system verifies her info: the operation ends successfully, and she gains access to the passenger user homepage.

Diagrams



Functional Requirements

3.2.3 Standard ride request

Purpose Passenger users can request a taxi both through the web or through the mobile application, giving only simple data about the number of passengers and sharing preferences (in case of shared ride, see also ?.?.?, "Shared Ride Request").
In any case, the system will then care about keeping the user informed about all details of his request, i.e. status of the request, estimated time of arrival (ETA) of the incoming taxi, in addition to its taxi code.

Scenarios

1. Elsa wanted to take the bus, but the heavy snow that fell in the last three days caused a lot of traffic problems. Fortunately for her, the taxi service is still functioning, so she opens myTaxiService on her mobile phone, logs in, and chooses "Request". She uses the GPS info to fill the "Origin" field, leaves the "Share" checkbox blank, then "Confirm". In a matter of minutes Frank, a taxi driver in her zone, accepts her request: Elsa is informed that she has to wait approximately 6 minutes for her taxi, encoded 288, to arrive. In the meanwhile, the system give her updates about the taxi position. At the expected time Frank arrives, picks Elsa up and carries her to desired destination.

Diagrams

Functional Requirements

- The system allows taxi ride requests if and only if the passenger accepts to give info about his/her location, either through GPS or directly writing down a valid location.
- The system allows taxi ride requests if and only if the passenger can be located in some definite position of some definite taxi zone.
- The system uses default values for the number of passengers and sharing preferences of a ride (1 person, no sharing), unless the passenger does specify them.
- The system uses a FIFO policy to manage forwarding of pending ride requests.
- The system uses a FIFO policy to manage the order of taxi drivers in queues to send notifications to.
- The system forwards a ride request to the first taxi driver in the considered zone queue if and only if he/she has a sufficient number of free seats available in his/her vehicle.
- The system keeps the passenger(s) notified about the status of the ride request he/she sent.
- Once a ride request has been accepted by some taxi driver, the system changes the request status from "Pending" to "Accepted".
- Once a ride request has been accepted by some taxi driver, the system calculates the ETA of the incoming taxi based on the distance between the taxi and the passenger(s), and the current traffic.
- Once a ride request has been accepted by some taxi driver, the system notifies the passenger(s) about the ETA of the incoming taxi.

- Once a ride request has been accepted by some taxi driver, the system keeps the passenger(s) notified about the current location of the incoming taxi, showing its position on a map.
- Once a ride request has been accepted by some taxi driver, the system prevents the passenger(s) to make a new ride request until the taxi driver changes the status of the ride to "Completed".

3.2.4 Reserved ride request

Purpose Passenger users can request to reserve a taxi for some definite future ride. The operation can be done both through the web or through the mobile application, and requires information about the location and exact date and time of the meeting point, the destination, the number of passengers and the sharing preferences (in case of shared ride, see also ?.?.?, "Shared Ride Request").
In any case, the system will then care about sending a taxi to the given location at the given date and time. Reservation requests must occur at least two hours before the ride meeting time.

Scenarios

1. George has an important meeting tomorrow morning, but his car suddenly broke. He decides he will take a taxi. Therefore, he opens the homepage of myTaxiService web application on his laptop, logs in as a passenger user, then clicks "Reserve". He selects "use maps" for both position fields, and pinpoints his home and the location of the meeting as "Origin" and "Destination", respectively. He selects "7.15" as the meeting time, leaves the "Share" checkbox blank, then clicks "Confirm".
   The next day, at 7.05, a reserved ride requests is received by Harry, the first taxi driver in the queue of the taxi zone where George's meeting point is located. Harry decides to refuse the request, though. The request is then forwarded to Isabelle, which was the second taxi driver in queue at the time of Harry's refusal. She accepts George's request, and at the given time arrives at his house. She picks him up and brings him to the meeting.

Diagrams

Functional Requirements

3.2.5 Shared ride request

Purpose

Scenarios

Diagrams

Functional Requirements

3.2.6 Request notification and response

Purpose

Scenarios

Diagrams

Functional Requirements

3.2.7 Availability settings

Purpose Taxi drivers are able to notify the system about their status through the mobile application at any moment, as long as they're logged in. In particular, the status can be either "Ready", "Busy" or "Offline".
Whenever a taxi driver logs in, the system automatically sets his/her status from "Offline" to "Ready" and put him/her on the bottom of its current taxi zone queue, based on GPS info.
When he/she accepts a taxi ride, the status is automatically updated to "Busy": the system then removes him/her from the queue, preventing the arrival of other ride requests.
Similarly, when the ride is over, the taxi driver has to notify the system that the ride has ended: the system automatically changes the status back to "Ready" and puts him/her back on the bottom of the current taxi zone queue, thus waiting for a new ride request.
Finally, when the taxi driver finishes his workshift, he may inform the system, or simply log off. In both cases, his/her status automatically switches to "Offline".

Scenarios Albert is a taxi driver subscribed to myTaxiService. He logs in through his mobile phone and his status changes from "Offline" to "Ready". The system receives info from the GPS and puts Albert on the bottom of the taxi zone he's currently in. After a while, his phone notifies him about a new ride request: he decides to accept it and his status changes to "Busy". He's no longer in the taxi queue. Albert goes to the start location, picks up Barbara and takes her to her destination. When they arrive, Albert informs the system that he has concluded the ride: his status changes to "Ready". The system puts him on the bottom of his current taxi zone queue. Later on, he receives another ride requests, but this time he decides to refuse it: its status remains unchanged as "Ready", but he loses all his positions in the queue. A few hours later, Albert finishes his worktime and logs off. The system sets his status to "Offline" and removes him from any queue.

Diagrams

Functional Requirements

- The system uses a FIFO policy to manage taxi zone queues.
- The system uses info provided by the GPS to locate taxis and decide their respective queues.
- The system automatically inserts taxi drivers in queues when their status changes to "Ready".
- The system automatically removes taxi drivers from queues when their status changes to "Busy" or "Offline".
- The status automatically changes to "Busy" when the taxi driver accepts a ride request.
- The status automatically changes to "Ready" when the taxi driver notifies the end of a ride.
- When status is "Ready", the application notifies about ride requests.
- When status is "Ready", the application enables the taxi driver to accept/refuse requests.
- When status is "Busy", the application prevents ride requests notifications.
- When status is "Busy", the application enables the taxi driver to notify the end of the current ride.

3.2.8 Account Settings

Purpose The system allows registered users to view and modify their profiles at any moment, as long as they're logged in. Usernames cannot be modified, while modified email addresses, taxi license IDs and taxi codes must not match with the ones of other users, otherwise the system denies the modification request. In case of modified email address, the system sends a confirmation email to the new address. Modification will succesfully ends when the user clicks the link in the sent email.

Scenarios Alex uses to periodically change his account password, in order to increase protection. To do so, every 3 months, he opens myTaxiService on his mobile phone, chooses "Profile", then "Modify". He selects the password field, writes down a new one, then writes it again in the "Confirm password" field. Finally, he clicks "Confirm": the system informs him that his account password has succesfully been updated.

Diagrams

Functional Requirements

3.3 Performance Requirements

myTaxiService will perform 95% of the operations within 4 second; the total amount of the operations within 10 seconds. The system should ensure at least 2000 passangers connected and 500 taxi drivers connected.

3.4 Design constraints

myTaxiService wants to reach most of taxi drivers and passengers, requiring minimum specifications for devices. Taxi drivers, registered to the system, have to use their own devices provided with GPS navigation system to perform the service. Mobile applications have to offer backward compatibility.

3.5 Software system attributes

Reliability The mean time between failures (MTBF) shall exceed 3 months.

Availability In order to maintain the system up-to-date and secure, myTaxiServer schedules downtime periods where will be executed routine operations. The service should be available 99% of the time.

Security myTaxiService to ensure service availability and data protection use:

- AES cryptography algorithm on network operations
- Data are encrypted and stored in backup drives to prevent system failure
- Login authentication. Users, after the registration, have to confirm their e-mail with the security code sent to the e-mail write in the registration form
- SQL injection detection

Server architecture will be implemented separating data from application. Application server must be separated from database and from the web server. All architectures are divided by firewalls.

Maintainability To ensure an easy maintenance of the software, it must be well-documented and written following coding patterns.

Portability Web programming ensures a wide target of browser. Mobile applications instead, cause of different languages and devices, have to be written following coding patterns for easy portability. Availability of the service is ensured by hardware and software limitations in Section 2.4.2.

3.6 Other requirements

Appendix A

Appendix

A.1 Actors

There are three types of actors that use MyTaxiService:

- visitors: they can only log in or sign up
- passengers: they can request,reserve or share a taxi
- taxi drivers: they can accept/deny a ride and modify their status

A.2 Identifying stakeholders

The main stakeholder of the project is the government of a large city. The government, with the help of the transport council, decided to improve the actual taxi service with MyTaxiService. With MyTaxiService the stakeholders want to:

- symplify the access of passangers to the service
- guarantee a fair management of taxi queues
- give the possibility to the passengers to reserve a taxi
- give the possibility to a passenger to share a taxi with other passengers

A.3 Alloy

Here the Alloy code for create MyTaxiService model

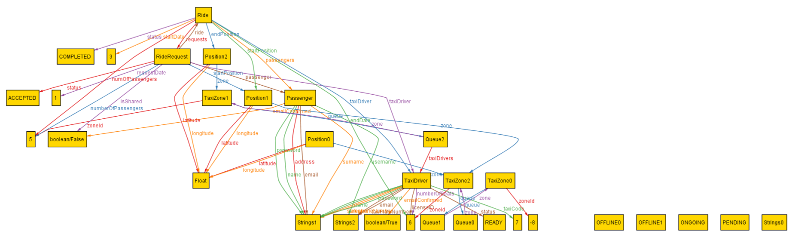
```
// ALLOY CODE FOR MYTAXSERVICE
module MyTaxiService
// Defines Bool, True, False
open util/boolean
```

```
// Dates are expressed as the number of seconds from 1970-01-01

//SIGNATURES
sig Strings{}

abstract sig User {
        email : one Strings,
        emailConfirmed: one Bool,
        username: one Strings,
        password: one Strings,
        name: one Strings,
        surname: one Strings,
        address: lone Strings,
        telephoneNumber: lone Strings
}

sig Passenger extends User{

}

sig TaxiDriver extends User{
        licenseID: one Int,
        taxiPlateNumber: one Strings,
        taxiCode: one Int,
        numberOfSeats: one Int,
        status: one TaxiDriverStatus
}
{
        taxiCode > 0
        licenseID > 0
        numberOfSeats > 0
}

abstract sig TaxiDriverStatus {}
sig READY extends TaxiDriverStatus {}
sig BUSY extends TaxiDriverStatus {}
sig OFFLINE extends TaxiDriverStatus {}

sig Float{
}

sig Position {
        latitude: one Float,
        longitude: one Float,
        zone: one TaxiZone,
}

abstract sig RideStatus{}
sig ONGOING extends RideStatus {}
sig COMPLETED extends RideStatus {}

sig Ride {
        startPosition:one Position,
        endPosition:one Position,
        startDate:one Int,
        endDate:lone Int,
        status: one RideStatus,
        taxiDriver: one TaxiDriver,
        passengers: some Passenger,
        numOfPassengers: one Int,
    requests: some RideRequest
}
{
        #requests > 0
        startDate > 0
        startDate < endDate
        startPosition≠endPosition
        numOfPassengers ≤ taxiDriver.numberOfSeats
        #passengers ≤ numOfPassengers
        #endDate=0 iff status= ONGOING
        #endDate=1 iff status= COMPLETED
}

abstract sig RideRequestStatus{}
sig PENDING extends RideRequestStatus {}
sig ACCEPTED extends RideRequestStatus {}

sig RideRequest{
        startPosition: one Position,
        endPosition: lone Position,
        requestDate: Int,
        ride: lone Ride,
        //passenger that requests the ride
        passenger: one Passenger,
        //additional passengers specified in the request
        numberOfPassengers: one Int,
        taxiDriver: lone TaxiDriver,
        status: one RideRequestStatus,
        isShared: one Bool
}
{
        endPosition ≠ startPosition
    isShared = False implies #endPosition= 0
        (#ride=0 or #taxiDriver=0) iff status = PENDING
        (#ride=1 and #taxiDriver=1) iff status = ACCEPTED
        requestDate>0
        numberOfPassengers > 0
}

sig ReserveRideRequest extends RideRequest{
        startDate: one Int
}
{
        #endPosition=1
}

sig TaxiZone{
        zoneId: Int,
        queue: one Queue,
        positions: set Position
}

sig Queue {
        zone: one TaxiZone,
        taxiDrivers: set TaxiDriver
}

// FACTS

// users must not have same username or same e-mail
fact UniqueUser{
        no u1, u2: User | (u1 ≠ u2 and (u1.username = u2.username or u1.email =
u2.email))
}

// taxi drivers must not have same licenseID or same taxiCode
fact UniqueTaxiDriver{
        no t1, t2: TaxiDriver | (t1 ≠ t2 and (t1.licenseID = t2.licenseID or t1.taxiCode=t2.taxiCode))
}

//zones must not have same zoneId
fact UniqueTaxiZone {
        no z1, z2: TaxiZone |( z1 ≠ z2 and z1.zoneId = z2.zoneId)
        queue = ~zone
}
```

```
//if a taxi driver has the status READY, he/she has to put into some queues
fact QueuesForReadyTaxiDriver{
        all t: TaxiDriver | ((t.status = READY)  iff (some q: Queue | t in q.taxiDrivers))
}

//if a taxi is in a queue must be only in one of them
fact TaxiDriverInOnlyOneQueue {
        all t: TaxiDriver | (lone q: Queue | t in q.taxiDrivers)
}
//a taxi must be BUSY during the time of the ride
fact BusyDuringRide {
        all t: TaxiDriver, r: Ride| (r.taxiDriver = t and #endDate=0)
                implies (t.status= BUSY)
}

//a passenger cannot take two ride at the same time
fact noPassengerOverlapRide {
        all p: Passenger, r1, r2: Ride | (p in r1.passengers and p in r2.passengers and r1 ≠
r2)
                implies (r1.endDate < r2.startDate or r2.endDate < r1.startDate)
}

//a taxi driver cannot take two ride at the same time
fact noTaxiDriverOverlapRide {
        all t: TaxiDriver, r1, r2: Ride | (t in r1.taxiDriver and t in r2.taxiDriver and r1 ≠
r2)
                implies (r1.endDate < r2.startDate or r2.endDate < r1.startDate)
}

// only ACCEPTED Ride Request can have a Ride
fact RideWithOnlyAcceptedRideRequest{
        all r: Ride, rr: r.requests| rr.ride = r and rr.status = ACCEPTED
}

//a Ride Request cannot be in two different Ride
fact RideWithOnlyAcceptedRideRequest{
         no  r1,r2 :Ride | r1≠r2 and (r1.requests=r2.requests)
}

//A ride that has more than one RideRequest must have only RideRequest shared
fact RideWithRequestsSharing {
        all r: Ride| (#r.requests>1)
                iff (all rr:r.requests|(rr.isShared = True ))
}

//if a ride refers to a ride request the taxi driver must be the same
fact taxiDriverUniqueRideRefersRideRequest {
        all rr: RideRequest | rr.ride.taxiDriver = rr.taxiDriver
}

//if a ride refers to a ride request the passenger of the Ride Request must be in the passenger of the Ride
fact passengersUniqueRideRefersRideRequest {
        all rr: RideRequest , r:Ride|  rr.ride = r implies rr.passenger in r.passengers
}

//if a ride refers to a ride request the start destination must be the same
fact destinationUniqueRideRefersRideRequest {
        all rr: RideRequest , r:Ride|  rr.ride = r implies rr.startPosition =
r.startPosition
}

//the number of passengers in the request refers to the corrispondent ride must be the same
fact correspondentNumberOfPassengers {
        all rr: RideRequest , r:Ride|  rr.ride = r implies rr.ride.numOfPassengers =
sum ( r.requests.numberOfPassengers)
}

//a passenger cannot take another request when is in a ongoing ride
```

```
fact noPassengerOverlapRideRequest {
        all p: Passenger, r1, r2: RideRequest | (p = r1.passenger and p = r2.passenger and r1 ≠
r2)
                implies (r1.ride.endDate < r2.ride.startDate or r2.ride.endDate <
r1.ride.startDate)
}

//the request date of a request ride must be before the start date of a ride
fact requestDateBeforeRide{
        all r: RideRequest | r.requestDate<r.ride.startDate
}


//the reserve date of a reserve ride must be before the start date of a ride and after the request date
fact reserveDateBeforeRide{
        all r: ReserveRideRequest | r.startDate<r.ride.startDate and r.startDate>r.requestDate
}


// ASSERTION

//all taxi in at maximum one queue
assert TaxiDriverInOneQueue {
        all t: TaxiDriver | (lone q: Queue | t in q.taxiDrivers)
}

//check TaxiDriverInOneQueue
//No counterexample found. Assertion may be valid

//No another ride if the taxi driver is busy
assert noAnotherRideIfTaxiDriverBusy {
        all  r1, r2: Ride | (r1.taxiDriver=r2.taxiDriver and r1 ≠ r2)
                implies (r1.endDate < r2.startDate or r2.endDate < r1.startDate)
}

//check noAnotherRideIfTaxiDriverBusy
//No counterexample found. Assertion may be valid

//No another ride if the passenger is going in another ride
assert noAnotherRideIfPassengerIsGoingInAnotherRide {
        all  r1, r2: Ride | (r1.passengers=r2.passengers and r1 ≠ r2)
                        implies (r1.endDate < r2.startDate or r2.endDate < r1.startDate)
}
//check noAnotherRideIfPassengerIsGoingInAnotherRide
//No counterexample found. Assertion may be valid


// PREDICATES


pred showNormalRequest(){
        #Passenger =1
        #Ride = 1
        #TaxiDriver = 1
}

run showNormalRequest for 3

pred show(){
        #Passenger ≥ 2
        #Ride ≥ 2
        #TaxiDriver ≥ 2
    #{x: Ride| #x.requests>1} ≥1
        #{x: RideRequest | x.isShared = True} > 1
}

run show for 4
```

Here the world generated by the command (run showNormalRequest for 3):



A.4 Software and tool used

- LaTeX (http://www.latex-project.org/) : to redact and to format this document
- Balsamiq Mockups (http://balsamiq.com/products/mockups/): to create mockups
- Alloy Analyzer 4.2 (http://alloy.mit.edu/alloy/): to prove the consistency of the model

A.5 Hours of works