Politecnico di Milano

2015-2016

Software Engineering 2: "MyTaxiService"
Requirements Analysis and Specifications
Document

version 1.3

Author: Nenad Petrovic

12th November 2015

Contents

1 Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD).

The main goal of this document is to completely describe the system in terms of functional and non-functional requirements, analyze the real need of the potential customers and users in order to model the system, show the constraints and the limitations of the software, recognize the actors involved into usage of the application, and describe the typical use cases and scenarios that would occur after finishing the development and during the usage of the system itself.

This document is intended to all developers and programmers who have to implement the requirements into application, to system analyst who want to integrate another system with this one, and could be used as a contractual basis between the customer and the developer.

1.2 Actual System

The software corporation wants to offer a new online taxi scheduling service to the government of a large city that would simplify the access of passengers to the service and guarantee a fair management of taxi queries.

We suppose that until now nothing in this direction has been done and our corporation has to create the entire application from the beginning, without using or modifying a previous system, porting or adapting some existing legacy system.

1.3 Scope

The aim of this project is to create a new online taxi scheduling service that would help the government of a large city to simplify the accesss of passengers to the taxi drivers, and would also guaranteee a fair management of taxi queries at the same time.

In order to use the system, person must be registered (and logged in, after that). User must enter valid information to register. The data needed for registration includes first name, last name, gender fiscal code, phone number, profile picture. Profile picture is optional and is used to help taxi drivers to recognize customers in a traffic. Phone number is needed is used in situations when customers and taxi drivers want to communicate directly, most often in case of some possible problems, as unexpected traffic rush or accident. After registration, user can send requests for a taxi drive by selecting of the desired destination on the map. Optionally, user can enter the maximum waiting time. If maximum waiting time entered by the user is less than estimated waiting time, the request is rejected automatically. Users can access the service both using web appication and mobile application. Taxi drivers use mobile application to receive requests from users. Taxi driver can accept or rejected the drive request. In order to be able to receive the request the taxi driver must be available. After accepting the drive request and starting the drive, the taxi driver becomes

unavailable and can't receive requests. After arrival to the final desired destination by user, the taxi becomes available again. Taxi drivers can also change their availability manually in case of accidents or any other unexpected situation. In order to guarantee a fair management of taxi queries, the city is divided in taxi zones (approximately $2km^2$). Each taxi zone has its own queue of available cars, represented by their numbers. If the request by first taxi from the queue of a certain taxi zone is rejected, the request is forwarded to the next available taxi from the same queue. Taxi is added to a queue when it comes into area belonging to a certain taxi zone and is available at the same time. When drivers accepts request, the car is deleted from queue, as it becomes unavailable. In order to become a taxi driver, user must be approved by one of the administrators (by visiting the administration office), by giving them valid info: car number, car model and driving license number. The system checks the validity of car number and driving license number in order to ensure the eligibility of taxi drivers. To ensure the quality of service, users and taxi drivers can report each other in case of bad behaviour (impoliteness, aggressive and unfair behaviour, customer didn't want to pay etc.) during drive. Each report is described by its reason, which should help system administrators decide to ban the user from system or not. Once the user is banned, he/she can't use the service again- and is prevented from logging in and registering by using the same fiscal code. In case of emergency, both users and taxi drivers are able to send S.O.S signal which will poll the emergency vehicle corresponding to each taxi zone.

1.4   Actors

- Guest: all guests users can only see the login page and complete the registration form to be able to access to all the functionality of the application as registered user. Guests can't use any functionality of the application. This decision is brought in order to avoid system overhead caused by visitors who don't understand how serious could be the effect of sending fake requests and prevent people from possible misuse of the system.

- User: after successful login using mobile or web application, user can send requests for taxi drives and report taxi drivers for bad behaviour. User's fiscal code is known to the system, so he/she can't register twice after being banned (deleted) for bad behaviour by system administrator. In fact, user could be a passenger or a taxi driver when he/she uses application from web or not driving a car.

- Taxi driver: an user approved by an administrator, who has a driving license and a registered car , and is able to both receive (if using mobile application) and send requests for a drive. He/she can respond the request by confirming or rejecting it. Taxi drivers have their availability status. If taxi driver is busy, he/her is unavailable, but after finishing the drive, his/her status is changed to available, although he can change his/her status manually in situations when something goes wrong before the regular drive to the desired destination is over.

- Administrator: the type of user whose username and password are received from the system management department, and has a role of a supervisor. Guests cannot register and become administrators themselves. Administrator has very important roles in this system:1) to promote users to taxi drivers by approving their elegibility (owning valid driving and car license) 2) to view reports by users and taxi drivers 3)to downgrade taxi driver to user 4)to consider bad behaviour

by both users and taxi drivers by viewing reports related to them 5)to delete users that show bad behaviour frequently after considering reports 6)to browse the drive events and see the people that participated in that particular event.

1.5   Goals

List of the goals of MyTaxiService application:

[G1] Only registered users can use the service – high priority

[G2] Registered users can use service if and only if they are logged in – high priority

[G3] User can request taxi to a desired destination using the service – high priority

[G4] Allow user to modify his/her profile – high priority

[G5] Taxi drive negotiation – high priority

[G6] Taxi driver's availability can be changed – high priority

[G7] Allow user and taxi driver to report the other side (user against taxi driver or taxi driver against the user) in case of bad behaviour – medium priority

[G8] Users that show bad behaviour frequently can be removed from system by administration – medium priority

[G9] Only eligible users approved by administration can become taxi drivers – high priority

[G10] Users and taxi drivers can request emergency service in case of an accident during drive – low priority

1.6   Definitions, Acronyms, Abbreviations

1.6.1   Definitions

- Starting point: is a location where the drive should start from, determined by its GPS coordinates.

- Ending point: is a location where the current drive stops, determined by its GPS coordinates. At this point, taxi driver's availability changes automatically to available.

- Request: is a message which consists of user's desired destination for a taxi drive and, optionally, maximum waiting time . The user himself is a „sender" of a request, while the taxi driver who receives the request is called „receiver". If there is no taxi driver who can receive the request, the receiver part of the message is empty (user receives a message stating that there is no taxi available and gives him/her an option to send the same request again or change the desired destination), and in case of forwarding the message to another taxi driver, system changes the receiver to the taxi driver polled from a queue. The request constains starting point-determined by sender's GPS location and ending point- which is, in fact the desired destination selected by user.

- Response: is a message which contains „y"-yes in case where taxi driver accepts the request or „n"-no in case where taxi driver rejects the request by user. This message also contains the estimated time needed for taxi driver to come to the requested starting point and estimated price. In this case, taxi driver has a role of sender, while the user who sent the request is receiver. Receiver part of this message can't be blank. User can accept or reject the drive offer.

- Report:is a message written by user or taxi driver during the drive event, in order to mention bad behaviour of the other side. There is no strict definition for „bad behaviour", so the one who reports has to write reason and describe the situation itself as close as possible. After that, administrators can view the reports and decide to delete user from system or not. User that is banned from system is prevented from using it and can't login or register once again, because his/her fiscal code is on the „black list". Single user can receive many reports. Reports are stored in database and could be viewed by administration.

- Taxi zone: a part of city (approximately 2km^2), which is defined by its center point, and its boundaries are calculated and managed by the system. Each taxi zone has its taxi queue, which consists of car identification numbers of available taxi drivers whose current location belongs to its boundaries. The city has at least one taxi zone (in case of a very small town).

- Availability: could have value „y"(yes)- which means „available" or „n"(no)- which means „unavailable". If taxi driver is available, he/she is placed in taxi queue related to his/her taxi zone and is able to respond to requests belonging to this taxi zone. If taxi driver is unavailable, he/she can't receive requests. When taxi driver accepts the request, his availability switches to „unavailable". After finishing the drive, the availability switches to „available" and is placed in a queue belonging to his/her current taxi zone. In certain cases, when something goes wrong (traffic rush, accident etc.),taxi driver can manually switch to „unavailable". When taxi driver sends S.O.S signal, his/her status changes to „unavailable" automatically.

- Drive event: is a system abstraction of a taxi drive, and consists of request by user, driver's response and reports during a drive. There could be many reports, or no reports at all. Administrations can browse the database of drive events and see the actors of a drive event related to request and response. During active drive event, user can report driver, or driver can report user for bad behavior by describing the reason of report as close as possible. Drive event starts after the user accepts the offer by taxi driver. Sometimes, it is called simply "drive" in this text.

- Estimated waiting time: time needed for a taxi driver to come to the user's current location.

- Maximum waiting time: maximum time that is user ready to spend waiting for a taxi to come.

- Bad behaviour: aggresion, avoiding payment, offensive beahviour etc.

1.6.2   Acronyms

- RASD: Requirements Analysis and Specification Document.

- DB: DataBase.

- DBMS: DataBase management system.

- API: Application Programming Interface.

- Database Management System (DBMS).

- OS: Operating System.

1.6.3   Abbreviations

- [Gn]: n-goal.

- [Rn]: n-functional requirement.

- [Dn]: n-domain assumption.

- [Mn]: n-mock up.

1.7   Reference Documents

- Specification Document: RASD.pdf.

- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

- IEEE Std 1016^{tm}-2009 Standard for Information Tecnology-System DesignSoftware Design Descriptions.

1.8   Document overview

This document is essentially structured in four part:

- Section 1: Introduction, it gives a description of document and some basical information about software.

- Section 2: Overall Description, gives general information about the software product with more focus about constraints and assumptions.

- Section 3: Specific Requirements, this part list requirements, typical scenarios and use cases. To give an easy way to understand all funcionality of this software and describe it as close as possible, this section is filled with UML diagrams: use case diagrams shows what actors could do during the system usage, sequence diagrams describe the most importnat scenarios that could occur, class diagram gives a whole system overview, state diagram describes state transitions in the application and gives a complete view of the application behaviour in possible cases, while the activity diagrams show the description of the activities of utmost importance for understanding how the scheduling system works.

- Section 4: Appendix, this part contains information about the attached .als file and screenshots of software used to generate it.

2   Overall Description

2.1   Product perspective

The application will be released as a both web and mobile application.The application will provide API that could help other developers extend its functionality by developing additional services. Users can access the service both using mobile web application and browser-based web application. In order to use application as a taxi driver, taxi driver user has to login into mobile version of the application (because of the use of GPS sensor). Administrator's application is targeted to be a web browser application, but will have also a mobile counterpart.

2.2   User and stakeholder characteristics

There are different types of possible users and stakeholders.

The first group of users that we expect to use our application are people (passengers) who want to request a taxi drive. They can send a request for a taxi drive to a desired destination.This user must be able to use a web browser and have access to internet or use the mobile application with GPS-enabled handset with active internet conncetion. If taxi driver accpets the request, user gets the notification about the drive offer with price and estimated waiting time and can accept or reject it.

The second group of users are licensed taxi drivers, confirmed by the administrators. They must use mobile version of the application, with GPS-enabled mobile phone and active internet connection. They can accept or reject taxi drive requests. They can be available or unavailable. If taxi driver is available, he/she is eligible to receive a taxi drive request. If taxi driver in unavailable, he/she is not in taxi queue.

The third group of users are administrators. Their profile credentials are given by city's government, and they can give a permission to a user to become a taxi driver. They act as supervisors, and can view bad behaviour reports in order to ban certain users and keep the quality of service as high as possible. They can access service using web application or mobile counterpart.

The fourth group of users are, in fact, potential, unregistered visitors – guests, who can register either using mobile or web application.

The fifth group of users would be developers who would like to extend usage of this system or call its functions by another system using programmatic interface. In this phase, the structure of programmatic interface and the functions themselves are not considered yet and will be considered later.

It should be mentioned that this system is going to be managed by city government, so city government belongs to a group of stakeholders also. City government is repsonsible for the selection of the administrators, providing access to databases that are necessary for data validation (fiscal codes, driving licenses and carid numbers) and also the price regulatory policy which depends on their decisions only.

2.3   Constraints

2.3.1   Regulatory policies

User has to register using valid fiscal code. User is responsible for entered data.

 In order to become a licensed taxi driver, user has to provide a valid driving license number and car identification number.

Validity of data provided is checked by the system, using governmnet's database.

Pricing is regulated by the government's policy. Taxi drivers must behave according to this policy. Not behaving according to pricing policy could lead to writing report and administrators would downgrade taxi driver to user immediately or remove them for system, preventning them to participate in MyTaxiService.

2.3.2   Hardware limitations

Taxi driver has to use mobile version of the application, with GPS-enabled cellphone and active internet connection.

Users can access aplication both using web application or mobile application.

2.3.3   Interfaces to other applications

MyTaxiService will offer API that would enable development of additional services in future, which would extend the possible usage of the system. API will be accessible also by other applications, so they could be extended by MyTaxiService's functions.

2.3.4   Parallel operation

MyTaxiService must support parallel operations from different users at the same time and when working with database.

2.3.5   Documents related

- Requirements and Analysis Specification Document (RASD).

- Design Document (DD). [to be developed]

- User's Manual. [to be developed]

- Testing report. [to be developed]

2.4   Assumptions and Dependencies

2.4.1   Assumptions

- Users can send requests for a drive in which case we have user's current location as starting point.

- Password must be at least 8 characters long and contain at least one capital letter, one number, and one special symbol.

- Users use real fiscal/id codes that belong to them (avoiding fake identity cases).

- Maximum waiting time will be considered as optional value. If estimated time is greater than user's maximum waiting time, than the request is discarded, and forwarded to next car in a queue.

- If user doesn't input maximum waiting time, then this parameter is not considered in scheduling.

- In case of S.O.S signal, the taxi request is sent to an emergency vehicle from corresponding emergency queue of the particular area. If there is no emergency vehicle in current taxi zone of the request sender, then the other taxi zones could be considered. Emergency vehicle is treated as a taxi. The price for any emergency drive is 0.

- If user confirms the drive offer and loses connection after that, the request is still considered as valid and is not discarded.

- If user doesn't respond the offer in 60 seconds, then the offer is considered as rejected.

- When user quits from waiting response page, then the request will be discarded from the system.

- If taxi driver loses internet connection, then he/she is still considered as available in next 60 seconds (to avoid effects of sudden connection loss in unreachable areas) . After that, the taxi driver is considered as unavailable, in order to avoid blocking the next driver in queue in case of an accident (if mobile device is destroyed or damaged).

- System is looking up queues for 180 seconds for each request. After that, the user gets response which states that there is no free taxi available.

- When the taxi driver's working time is finished, he/she switches availability to unavailable.

- Requests can be responded only by taxi drivers corresponding to the user's current location taxi zone.

- If user doesn't accept an offer, then his request is no longer valid and can't make any request in next 10 minutes.

- Taxi driver can only be registered with supervision of administrator.

- Taxi driver can be registered only with one car at a moment.

- There are no two taxi drivers that have the same car.

- No taxi drivers with same driving licenses could be registered.

- If user is registered and promoted to taxi driver, and wants to drive, he/she must use the mobile version of the application (GPS-senosr, portability).

- In order to obtain administrator username and password, user must be registered in city's government or administration office. He/she will receive a letter with username and password with credentials to log in as an administrator.

- After downgrading taxi driver to user, his/her car information is deleted and is not longer considered as a part of any queue.

- Car will be in queue only if taxi driver's status is set to available.

- If taxi driver rejects a drive request, than it is forwarded to the taxi driver whose car is next in a queue for that taxi zone.

- When someone sends report, the drive event is over.

- If user rejects drive offer, than nothing changes in terms of taxi availability.

- If there is no taxi in a queue for a current taxi zone, user will get notification stating that there is no available taxi at the moment.

- The system is going to be used for processing drive requests only in one city. No possible drives between distant areas and different cities are considered.

- Report can be written in period from request acceptance by taxi driver until the end of the drive. After that, writing the report is not possible.

- Payment procedure is not supervised nor handled by system. The only payment method available is cash.

- Taxi driver can get only one request at current point of time.

2.5 Future possible implementation

In this part, many possible future implementations are going to be considered. Many of them are with low priority, and are more ideas and suggestions, but programmatic API is something that is going to be considered more detailed in later project phases (to be more price, in design document).

- Since programmatic API is still not described in details, this feature is going to be considered as a highest priority possible future functionality in design phase. Intention is to make a programmatic API that would allow extension of the application and usage of the application by other applications.

- More secure login procedure – captcha for users and face recognition for administrators – high priority.

- Extend the system, so each taxi driver can be registered with multiple cars and same car can be used by multiple taxi drivers – medium priority.

- Payment procedure supervision by system is considered to be developed in future – low priority.

- Electronic money transfer support – low priority.

- Extend system, so it can give ability that multiple passengers can share same taxi according to the car's capacity.

- Adding taxi driver rating system, that would allow administrators to give an award to taxi driver of the month/year award –low priority.

- Adding more detailed log that would give better insight of the system usage – low priority.

3 Specific Requirements

## 3.1 External Interface Requirements

3.1.1 User Interfaces

In what follows, mockups that represent the idea and conceptual design of the application are presented – mobile and web application. The mockups presented may differ from the final product and they are placed in this document for mainly ilustrative purposes – to give you better idea how is the final product going to look. First, the user's point of view is presented, and then, taxi driver's and administrator's, in this order.

User's point of view

3.1.1.1 Login - The mockup bellow shows the home page of MyTaxiService. Here users can log in to the application and guests can access to the regitration form in order to use the service. Here, the mobile application mockup is presented (the web version will look as similar as possible)
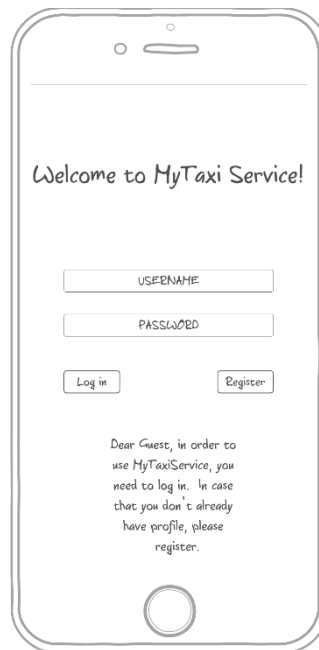
3.1.1.2 Registration form - This mock shows the registration form page on a mobile device. This is form that consists of user-filled textboxes. Profile picture is optional and helps in situations when user/driver needs to recognize another person that is involved into drive process. After pressing the button called „Confirm registration", the guest becomes registered user and can benefit from usage of this system. Profile picture is optional, and user can select a picture from device's storage.
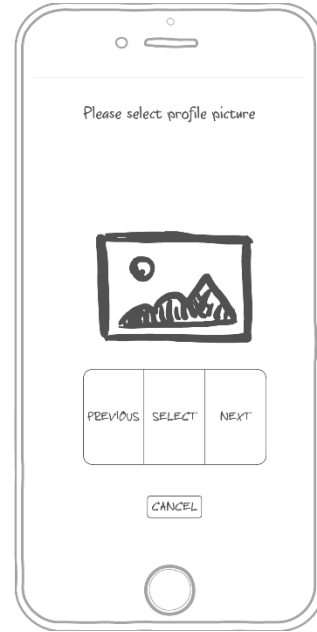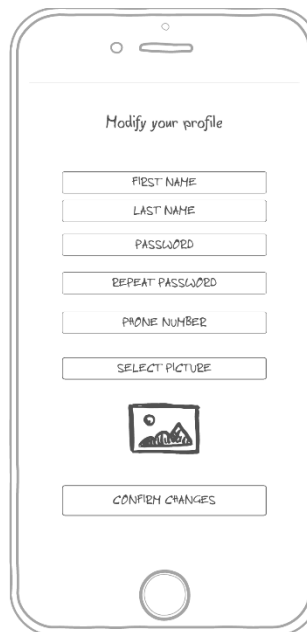
3.1.1.3 User menu – This mockup shows what user can do once logged in – request a taxi, edit profile or log out.

3.1.1.4 Request a taxi - This mock up shows what happens when user wants to send a request for a taxi drive. User selects desired location (left picture), optionally enters maximum waiting time and submits the request (right picture).
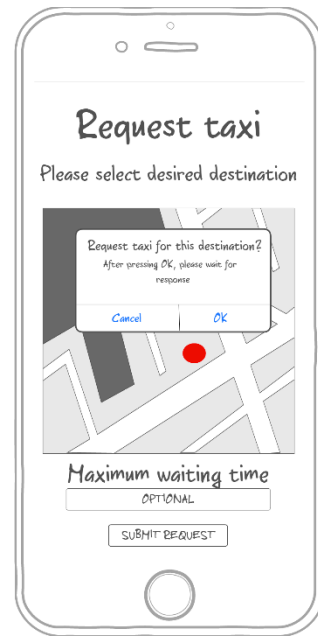
3.1.1.5 Profile modification - This mock up shows the form where user can edit his/her that or change picture, but can't change fiscal code.

3.1.1.6 Waiting for response screen -This mock up shows what happens when the user submits the request for a drive. System responds within 180 seconds, but user can press quit waiting before that and cancel request and return to menu.

3.1.1.7 System response - This mock up shows what user sees when request is accepted by taxi. User can view taxi details (displayed on the right picture), accept or reject the offer. Estimated waiting time and estimated price are shown, so they can help user decide whether to accept the offer or not.
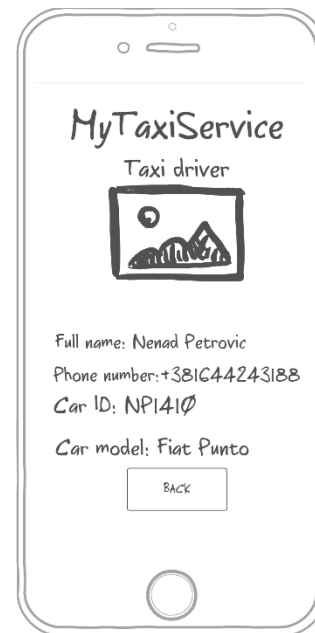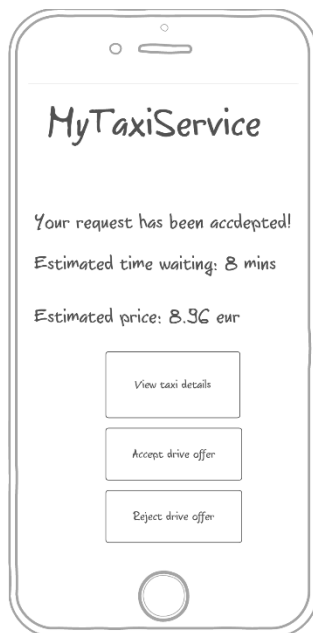
3.1.1.8 No taxi available – This mockup shows the screen displayed to user when there is no taxi available.
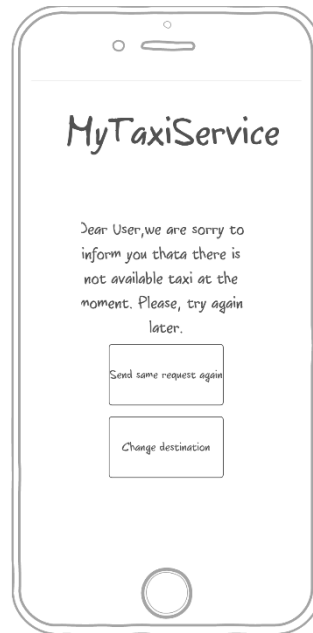

3.1.1.9 Drive progress and report form – This mockup shows the screen displayed to user after accepting the drive offer. User can see the drive progress, estimated time remaining, but can also report driver or send an S.O.S signal. After clicking the Report driver button, the screen on the right side shows up. User has to enter the reason of reporting the driver and confirm report in order to make it valid.

Driver's point of view

3.1.1.10 Driver menu – this mockup is similar to user's menu, but includes availability status button.
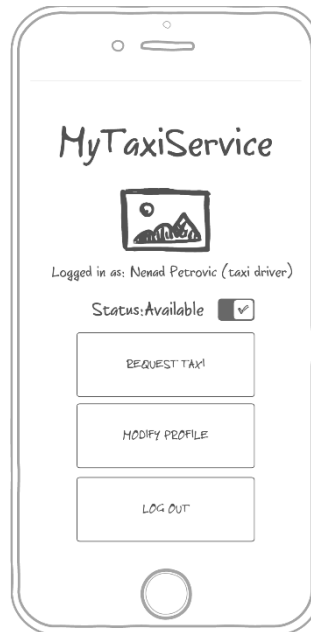
3.1.1.11 User request accept/reject page – Driver can accept or reject user's request for a taxi drive. It is possible to optionally view request summary, as shown on the middle picture. After that, it is possible ro view user profile of the person who sent the request (right side).
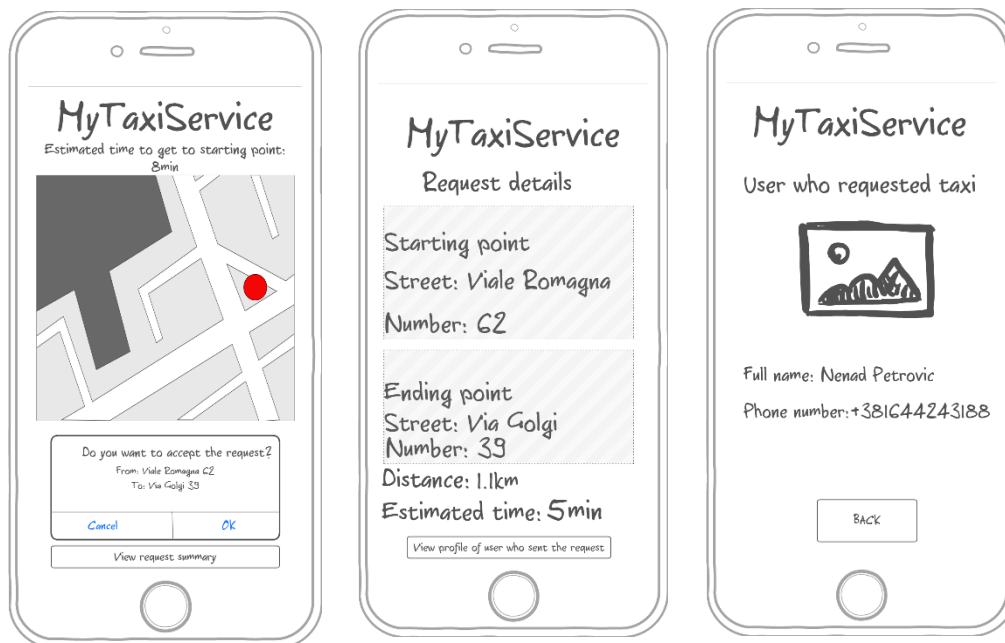
3.1.1.12 Waiting user response – Within 60 seconds, user who requested drive must accept or reject the drive offer and taxi driver has to wait (the left side). After accepting the offer, the similar screen is displayed to a taxi driver like in 3.1.1.9 (the right side). Driver can also report user the same way as in 3.1.1.9.

MyTaxiService

Dear taxi driver, please wait for user's response

MyTaxi Service

Destination:Via Golgi 39

Estimated time remaining

30 sec

Report user   S.O.S.

Administrator's application is optimized for web browser, but will also have a mobile counterpart. In what follows, the web browser version is going to be presented.

3.1.1.13 Login page – In case of using the web browser version, user logs in by entering data in form like shown bellow.
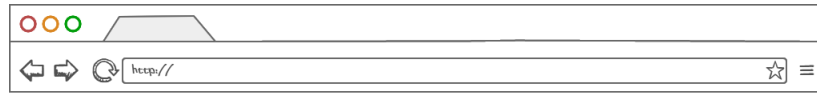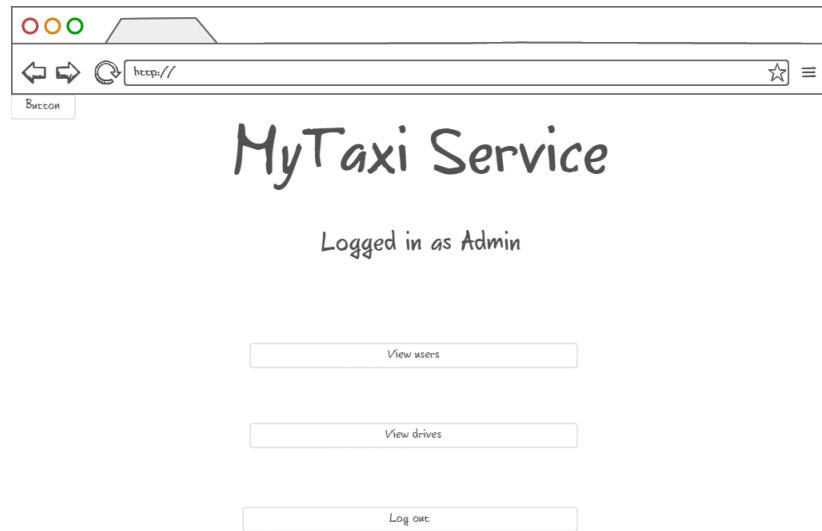

Welcome to MyTaxi Service!

Dear guest, please log in or register in order to use our service.

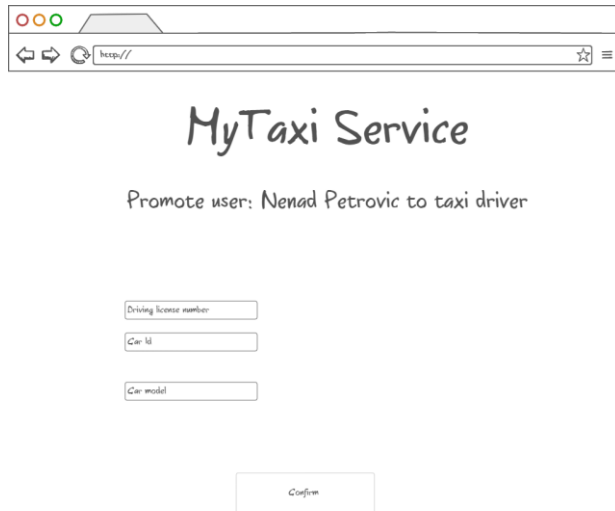| Email | Email |
| Password | Password |

Sign in   Register

3.1.1.14 Administrator menu – Administrator can view users (both tzpical users and taxi drivers) and their reports in order to decide their removal from system or view drives. Of course, there is also a „Log out" button.

3.1.1.15 Browsing users – Administrator can view users and their reports in order to decide if they deserved removal from system, promote users to taxi drivers, downgrade taxi drivers to users and edit user's data.
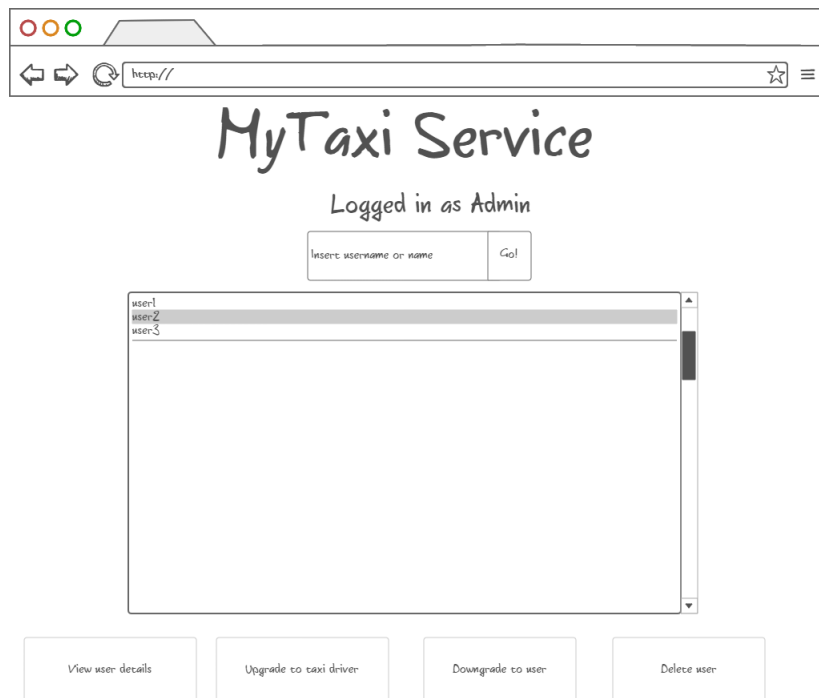
3.1.1.16 Promotion to taxi driver – After user selection, administrator can promote user to taxi driver by entering valid driving license number, car identification number and car model (optional). Data is checked by the system. If data is valid, promotion can be confirmed.
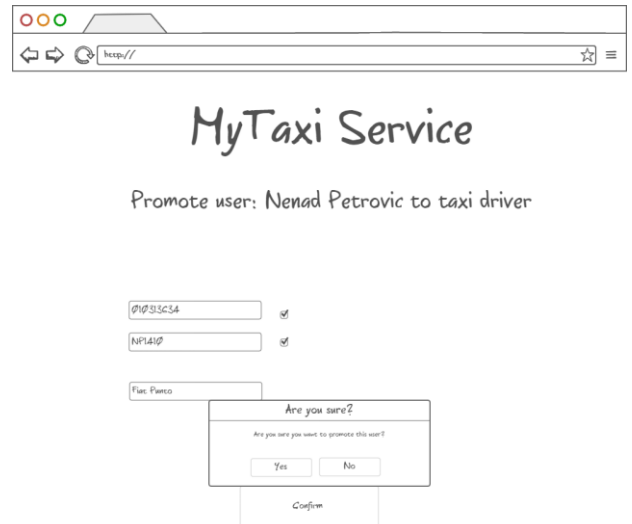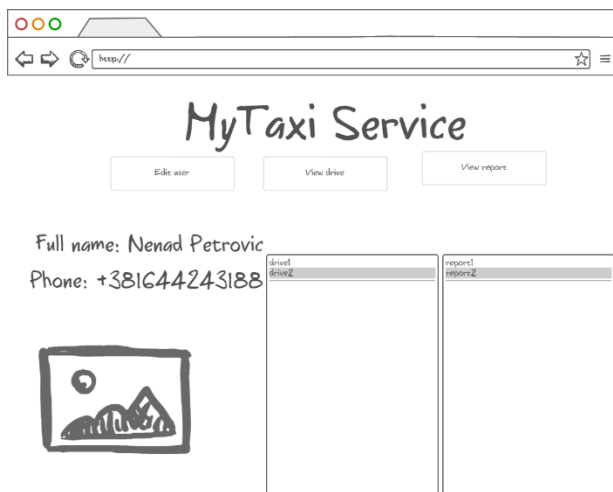

3.1.1.17 Viewing user info, reports and drive event – After user selection, administrator can view user details and all reports and drive events related to a particular user (left picture). It also possible that administrator wants to delete the user after reading the reports. The reports of selected user are shown in a form of table as on the right-side picture. Similar form appears when system administrator is notified about new report which needs an action.
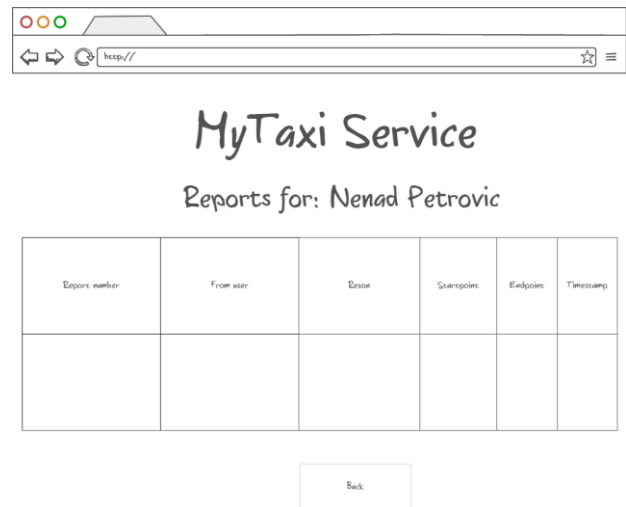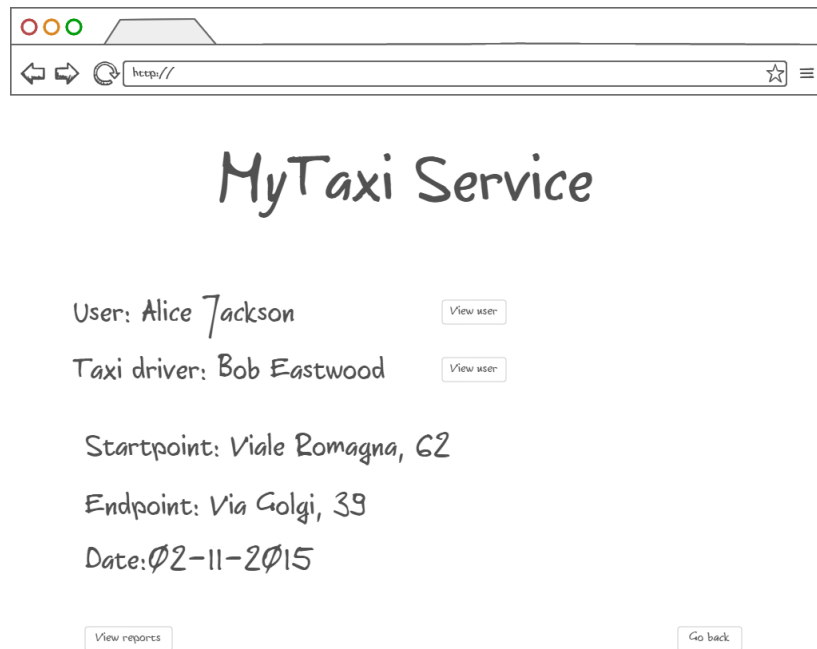
Drive events related to selected user could also be viewed in a form of a short summary with option of viewing details of actors involved in the selected drive event.

3.1.1.17 Deleting selected user – After user selection, administrator can also delete the selected user if he/she deserved. The deleted users can't login nor register anymore. Their fiscal code is blacklisted.

3.1.2   API – Application Programming Interface

Software interface to other applications will be in a form of Application Programming Interface.

MyTaxiService will offer API that targets developers as possible users. Developers will have programmatic interface that will allow them to call MyTaxiService functions inside their application, so they can easily integrate with their apps and extend the extend the usage of MyTaxiService.

API would contain objects and functions that encapsulate MyTaxiService functionalities.

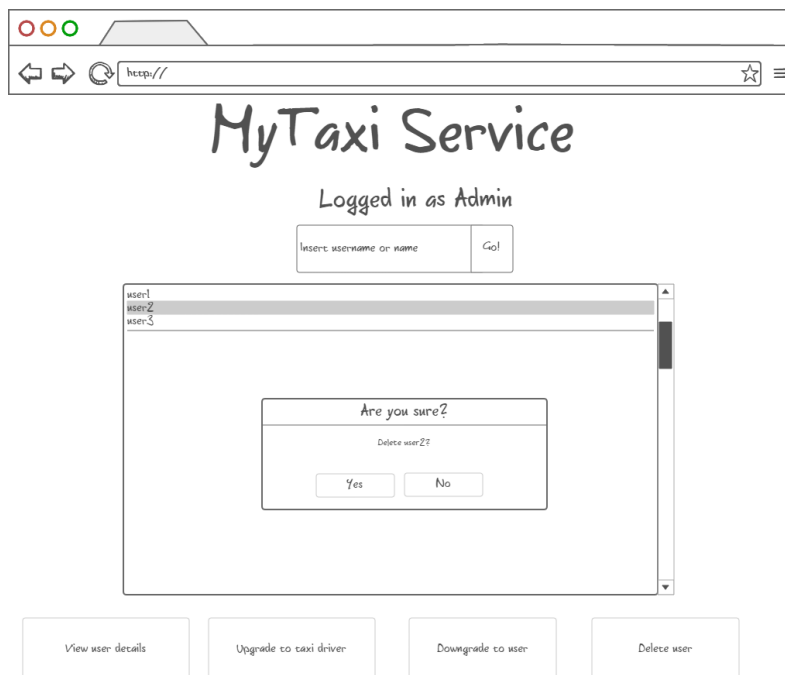Developers will be able to connect their social network-alike applications and easily migrate data to register using already created profiles that have necessary information for registration. For example, in notation of Java-alike object – oriented pseudo-language, it would look like:

```
MyTaxiServiceObject.register(username:      String,      password:String,
fiscalCode: String, phoneNumber: String, gender: Char, picturePath: String)
```

Developers would also have ability to request a taxi inside their application if they have a valid session (previously logged in):

```
MyTaxiSession     session1=MyTaxiServiceObject.login(username:     String,
password:String)

Session1.RequestTaxi( )
```

Functions like this would give developers ability to make more complex systems using MyTaxiService functions as primitives.

3.1.3   Hardware Interfaces

MyTaxiService won't have a specific hardware interface. It will be used on mobile phones and personal computers, so it uses already existing hardware interfaces.

3.2   Functional Requirements

The following requirements are determined according to the Jackson and Zave analysis, so assuming that the domain properties, which are described above, hold for mentioned purposes, we construct the requirements in order to satisfy goals.

3.2.1 [G1] Only registered users can use the service

[R1] Visitor must fill in the form with his/her data

[R2] System checks fiscal code and password for correctness

[R3] Unregistered visitors can only see login page

[D1] Email address used for registration data must be formally correct

[D2] There is no more than one person with same fiscal code

[D3] User is not using someone else's identity

3.2.2   [G2] Registered users can use service if and only if they are logged in

[R1] User has to fill form with credentials

[R2] Username and password entered during login process must be correct and validated by system

[R3] Service becomes available to users once they log in

3.2.3   [G3] User can request taxi to a desired destination using the service

[R1] User selects the desired destination

[R2] User can optionally enter maximum waiting time that he/she is able to wait for taxi to come

[D1] Taxi drive's starting point is user's current location

[D2] User can't request another destination as a starting point

[D3] If user loses connection before the time runs out, the request is discarded from system

3.2.4 [G4] Allow user to modify his/her profile – high priority

[R1] User enters data using the form

[R2] Data validity is checked

[D1] Users can't edit their fiscal code

[D2] Users can't change their gender

3.2.5 [G5] Taxi drive negotiation – high priority

[R1] Taxi driver can see the request information

[R2] Taxi driver can accept or reject request

[R3] User can see the estimated price and time

[R4] User can accpet or reject drive offer

[R5] Request is forwarded to another taxi in same taxi zone if rejected

[R6] Possibility to determine the taxi zone using the current location

[R7] Queue of taxi drivers that can get requests are from same taxi zone as the user who requested a drive

[D1] User's taxi zone corresponds to his/her current location

[D2] Taxi zone boundaries are regulated in advance

[D3] Taxi zones are related to one city only

[D4] Only available taxi drivers are considered in queue

[D5] User has 60 seconds to accept or reject the offer

[D6] One request is valid for 180 seconds. Within 60 seconds,user will get the repsonse if there is taxi available or not.

[D7] Taxi driver has 60 seconds to accept or reject the request

3.2.6 [G6] Taxi driver's availability can be changed – high priority

[R1] Taxi drivers can click on availability slider to change their availability

[R2] System observes drive events and changes the availability according to real word changes

[D1] If driver clicks button and was available before, then becomes unavailable.

[D2] If driver clicks button and was unavailable before, then becomes available.

[D3] When drive starts, driver becomes unavailable.

[D4] When drive finishes, driver becomes available.

[D5] When report is submitted, taxi drive becomes available

[d6] When SOS is requested, taxi driver becomes available

3.2.7 [G7] Allow user and taxi driver to report the other side (user against taxi driver or taxi driver against the user) in case of bad behaviour – medium priority

[R1] Users and taxi drivers can initiate report clicking on "report" button

[R2] Users and taxi drivers must write provide text (reason) in which they describe the situation

[D1] In case of report, it is assumed that bad behavior occurred during drive event

3.2.8 [G8] Users that show frequent behaviour can be removed from system by administration – medium priority

[R1] Administrator can read reports

[R2] Administrator can see the users related to report

[R3] Administrator can delete user from system

[D1] Administrator decides is report is serious enough to be considered

[D2] Administrator cannot remove any other administrator user from system in any case

[D3] Administrators are employed by city government and have special credentials used for logging in.

3.2.9   [G9] Only eligible users approved by administrationcan become taxi drivers – high priority

[R1] Administrator can promote users in order to become taxi driver clicking on button

[R2] Administrator must find the desired user either scrolling the list or browsing by keywords

[R3] Administrator must enter driver's car ID number and driving license number

[R4] System must check validity of entered driving license and car ID number

[R5] Administrator can downgrade driver to user again

[R6] Administrators can change driver's info (for example, when driver changes car or license number)

[D1] User visits the administrator's office to show real documents

[D2] People are using their own driving licenses

[D3] There are no people with same licenses

[D4] Each taxi driver can be registered with one car at the moment

[D5] Administrator decides to promote or downgrade user

3.2.10 [G10] Users and taxi drivers can request emergency service in case of an accident during drive – low priority

[R1] User or taxi driver sends an S.O.S signal clicking on the S.O.S button

[R2] System dispatches the closest possible emergency vehicle

[D1] Accident occured during drive event

[D2] Emergency vehicle is coming to sender's location

[D3] Driver that responds to emergency group is dispatched, not a regular taxi driver

3.3   The world and the machine

In what follows, will be given an overview of „The world and the machine" (M.Jackson and P. Zove) approach application to MyTaxiService as an example.

 I would clearly mention that the diagram bellow has only illustrative purposes and is not detailed overview, so only the most relevant parts are included, while some parts are omitted in orded to keep picture as understandable as possible.

As it can be seen, there are three main categories:

The World – portion of real world affected by machine

The Machine – portions of system to be developed

Shared phenomena- 1.can be controlled by world and observed by machine or 2. controlled by machine and observed by world and are, in practice, portions of system that have to be implemented.

Considering the results presented in 3.2, we can apply The world and the machine approuch to our system.

The World - As previously said, in set „The world" are placed all these things that are related only to environment and out of machine range. For example, internet connection coverage is related to many environment factors. Payment procedure isn't covered by this system yet and is performed only in world domain. User banning decision is brought by the administrator and is not based on machine algorithm. Accident or breakdown are events that are out of control of the machine and are result of the environment conditions. Maximum waiting time is not a result of any machine calculation and depends only on user's decision and patience. User's behavior is also in this part. Taxi and user movement are in this part. Taxi driver must come to administration office in real world to show driving license and car registration document in order to become eligible.

Shared phenomena – Mainly, here are the actions that users can take that will affect the real world. Taxi requests, accepting and rejecting taxi requests and offer – are used taken actions that are visible by machine and cause changes in real world. Sending a report or S.O.S signal also belongs to this category. When accident or bad behavior occurs in real world, the users react by sending a corresponding message to the system. Taxi drivers can change their availability, which is observed by the machine and affects the scheduling and queues, but is controlled by "The world" and "The Machine", so it is shared phenomenon. Forwarding requests is also in this part, because it is controlled by "The Machine", but affects the real world. Movement of cars corresponds to real world, but is observed by the machine which determines the corresponding taxi zone. GPS coordinates determination is observed by machine, but the coordinates themselves are changed by real world events (user and taxi movement). Registration belongs to this part, because user enters the data, but the process is observed by the machine and affects the database. After registration, user can use the service (the real world phenomena), but it is controlled by the machine itself, which has its own databases where it stores the registered users, so the machine could add new users. Profile modification also belongs to this part – the same reason. GPS coordinates validation (belonging to city or not, or particular taxi zone) also corresponds to this part, along with scheduling algorithm which needs to be implemented, because both of them are done and controlled by machine, but do affect the state and the situation of the real world. Estimation algorithms are also part of the system that has to be developed, and depends on both real world and machine. Pricing regulatory comes from business owners, but, in practice, does affect the algorithm performed by the machine itself. Promotion to a taxi driver or downgrading a taxi driver to user is also shared phenomenon because – in first case, when taxi driver

needs to come to admin's office in order to register, but the admin needs to enter his data in order to promote him to a taxi driver so it will affect the database and the system itself, but the real world also, because the particular user gets taxi driver role. User removal is also shared phenomenon, because it is performed by system on a database level, but it depends on administrative decisions and affects the real world at the same time – users removed from system cannot use the service anymore nor register again. Taxi zone division is done by system but takes some real world but takes into account some government decisions and affects the whole taxi scheduling and dispatching mechanism.

The Machine – This category contains only entities strictly related to computers, programs and communication devices that are necessary along with database with users and other data belongs here to ensure that system can perform its role.

In what follows, the diagram presents the facts previously stated.

3.4   Scenarios

In this part of the document, 5 different real-life scenarios are described in order to illustrate how MyTaxiService could be used in practice and which situations can be covered by this system. Only the most interesting and complex scenarios are selected and presented, because the number of possible scenarios is quite large, but many of them are really trivial, so they are not described here.

3.4.1   Scenario 1

Alice wants to use MyTaxiService. She hasn't previously used the system. First time she started application , she has figured out that she has to register in order to use MyTaxiService. She clicks the „Register" button on the home page and then she is transfered to a registration form. On this form, she enters the required data (fiscal code, username, password, phone number confirms the password, then first name, last name, gender and even selects here profile picture from a mobile phone) and clicks on submit button. But, she gets message that states that her desired username already exists in database, so she needs to select another username. After long consideration, she finds another appropriate username and submits the data. Everything is valid, the database is updated, so she is now a registered user. She is back to a home page and can now enter her credentials, as she is so excited to use the service as soon as possible.

3.4.2   Scenario 2

Alice has entered her credentials and is logged in. From user menu, she selects „Taxi request" button. After that, map is displayed to her. She lives in „Cassa dello Studente – Leonardo da Vinci" and wants to go to Expo. She finds Expo and map, and selects it as a destination. She is so excited and confirms the request, but she forgot to enter maximum waiting time. She is in a hurry and hits the quit button after 2 seconds waiting for response. The system didn't send her response, because she cancelled it before her request was accepted by taxi driver. She still wants to try out this new service, but this time she selects much closer destination - „Via Golgi 39". She enters that she can wait for taxi only 2 minutes After that, she is transfered to response waiting screen. System transfers her request to the first taxi from a queue belonging to her taxi zone and is available. Bob is the taxi driver who received her request, but he rejected it, because he finished his working time and forgot to change his availability status. After that, the queue is checked. The request is forwarded to next taxi driver from her taxi zone. John is polled from the queue, because he needs only 1 minute to come to her place. He accepts Alice's request. Alice gets notification that she has a taxi drive offer with details – the drive would cost 4.21 euros and she will have to wait 1 minute. She accepts the offer. John is coming to her location.

3.4.3   Scenario 3

Alice is in John's car, on the way to the desired destination. John has a bit crush on Alice and starts to behave in a way that Alice didn't like. She told him that she has a boyfriend, but he is still behaving the same way. They are about 400m from the destination. Alice is angry and clicks the „Report" button. She describes the situation into reason field and submits the report. Report and drive event details are inserted into system. Drive vent ends and system destroys the object. John

stops car. Alice exits the car. John's taxi is available again and is available in queue belonging to his current taxi zone.

3.4.4   Scenario 4

Nenad is one of the administrators. He is notified about the report and views it. He decides to see John's profile. Nenad sees that this is not John's first time to be reported for such thing, so the administrator chooses the most severe measure – to delete John from system. He clicks on „Delete" button. John's fiscal code is transferred to a black list and he is no longer able to register, log in or work as a taxi driver using this service.

3.4.5   Scenario 5

One of Nenad's friends comes to MyTaxiService administration office. Friend told him that he had bought a car. Nenad is amazed a bit, because he knew that his friend had difficulties passing driving license exam. Friend told him that he wants to work as a taxi driver. Nenad finds his friend in database and clicks on „Promote" button. After that the form for taxi driver details appears.

-    Which car do you have? - Nenad asks.
-    Zastava 101 ! – friend responds as quick as possible
-    Car registration number? –Nenad asks.

His friend politely reads him car number from a mobile phone. Nenad immediately wants to check the validity of car registration number. It turns out that the number is not valid. Nenad asks again. The friend honestly told him that he tried to lie to him, but didn't know that system checks the data validity using the government database. Nenad is angry and explains his friend that he has luck because the system checks data validity.

-    I suppose that It's better not to ask you if you have a driving license! You could risk your life driving without license. But you don't even have a proper car registration number.

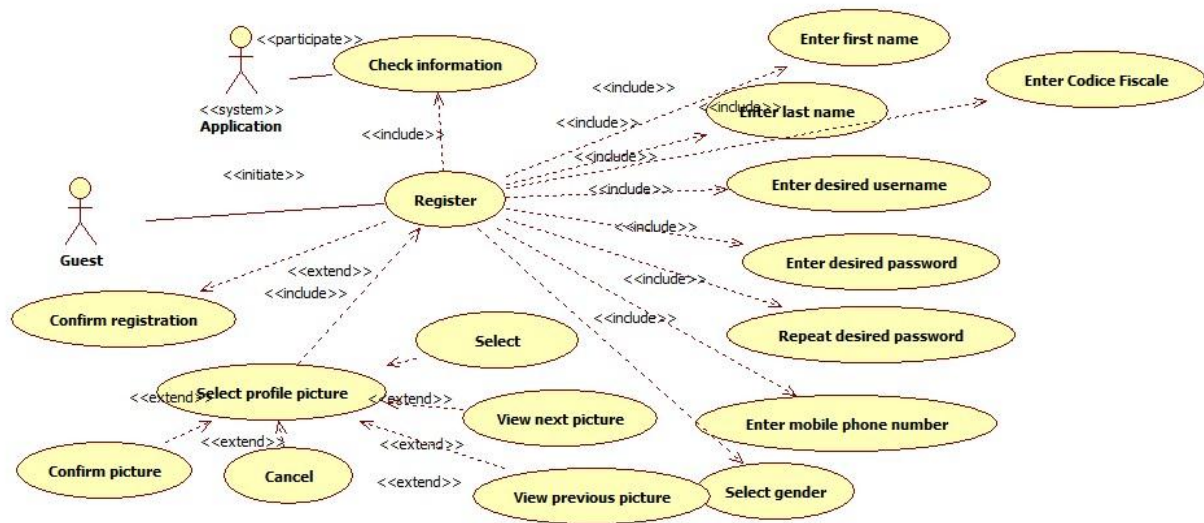The friend lost a good opportunity to find a job, and leaves the office.

3.5   UML Diagrams
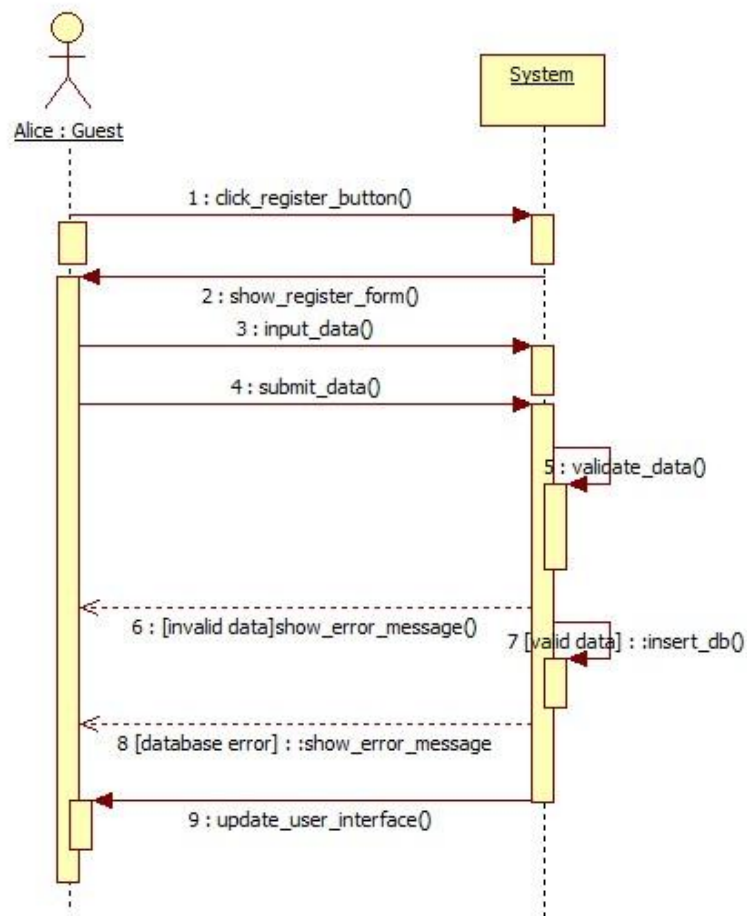
3.5.1   Use Case and Sequence Diagrams

In this part, use cases are given in order to describe what can system do, which are cases when it can be used and how it can be used in these cases.

3.5.1.1 Register

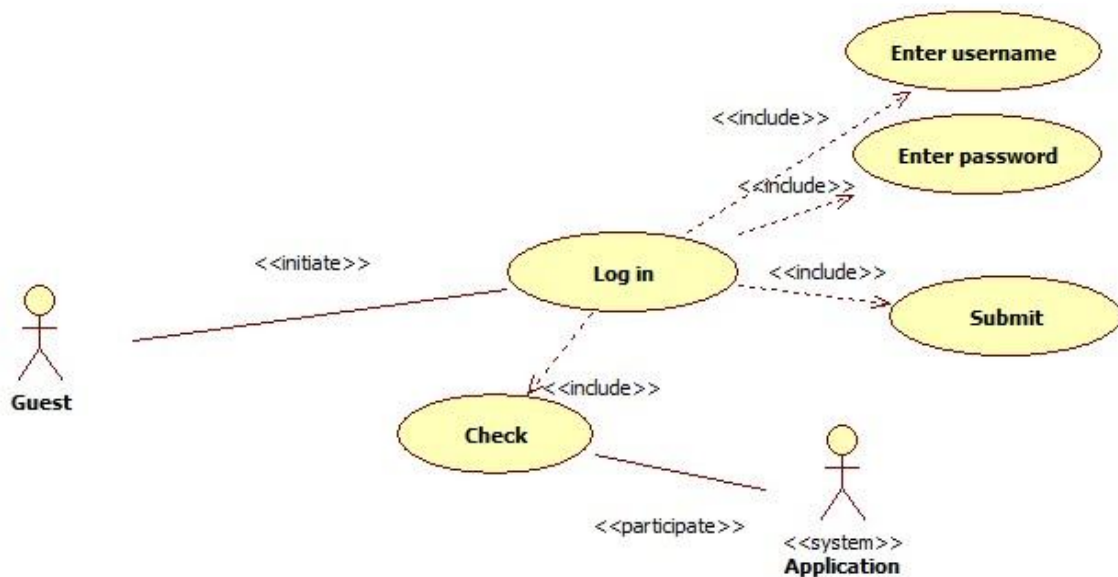| Actor	Guest
Goal	Users are added to database (in order to access the service)
Input Condition	NULL
Event Flow	1. Guest on the home page clicks on "Register". button to start the registration process. 2. Guest fills in at least all mandatory fields and selects picture (optional). 3. Guest clicks on "confirm" button.
Output Condition	Visitor succesfully ends registration process and becomes a Registeref User. From now on he/she can log in to the application using his/her credential and start using MyTaxiService and is added to database.
Exception	1. The visitor is already a user. 2. One or more mandatory fields are not valid. 3. Username choosen is already used by another user. 4. Password isn't according to security policy 5. Fiscal code is not valid, already in use or is blacklisted (banned) System participates in this use case by checking the data validity after pressing the „confirm button". All exception are handle alerting the visitor of the problem and application goes back to point 2 of Event Flow.

The following sequence diagram is based on Scenario 1.

3.5.1.2 Login

Actor	Guest
Goal	Guest that already have profiles log in, so they can access service
Input Condition	Guest is previously registered into the system.
Event Flow	1. MyTaxiService shows the login page to visitor. 2. Visitor completes the form inserting correct username and password.
Output Condition	1. MyTaxiService verifies the credential of visitor and if correct show the menu page with enabled functions (depending on which type of user is logged in). 2. Guest becomes User and can acccess the service.
Exception	1.If username and/or password are incorrect or not exist, MyTaxiService notifies that to guest and shows the login page again. 2. If user is blacklisted, he/she will also get a message that can't use the service. System actively participates in this part, by checking the data validity.

3.5.1.3 Profile modification

Actor	User
Goal	User changes his/her data after registration
Input Condition	Registered User is already logged in into MyTaxiService.
Event Flow	1. Registered User click on „Modify" button 2. User completes form with new data 3. User confirms changes
Output Condition	User's data is changed by new data entered by himself/herself. Database is updated.
Exception	1. New data not is not valid. System participates in this use case by checking the new data entered by user. If it is not valid, user has to do steps 2. and 3. again.
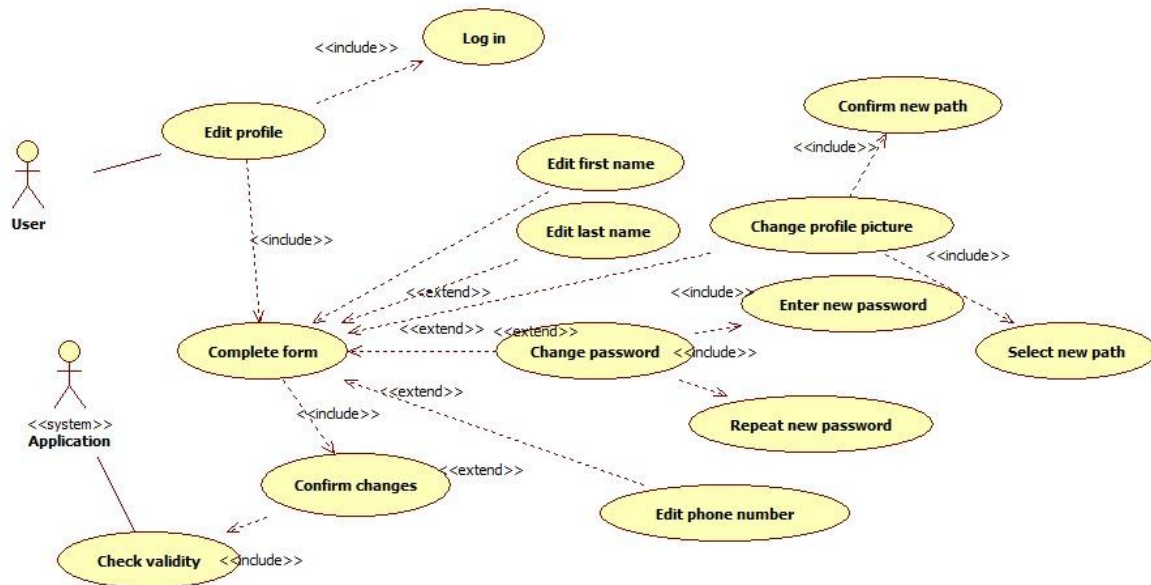
3.5.1.4 User negotiation for taxi request to a desired destination .

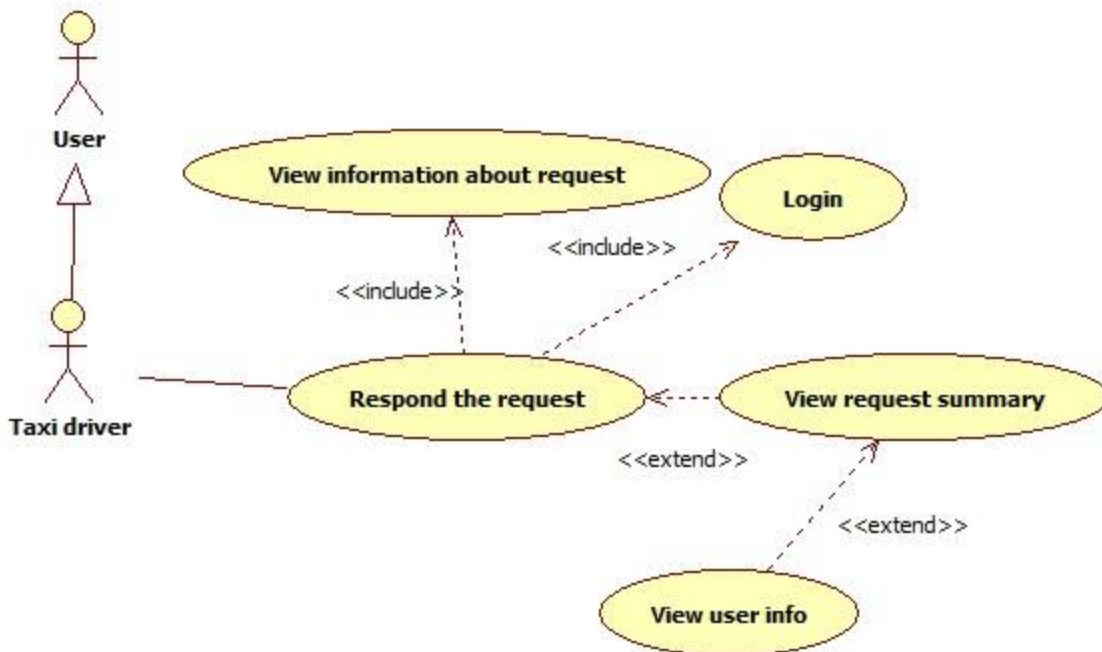| Actor	User, Taxi Driver
Goal	Users and taxi drivers negotiate about taxi drive for a desired destination
Input Condition	1. User and taxi driver are already logged in into MyTaxiService. 2. Taxi driver is using mobile application.
Event Flow	1. User triggers „Taxi request" button 2. User selects destination on the map 3. User optionally enters maximum waiting time 4. User confirms request 5. System finds taxi from queue 6. Taxi driver responds to a request by rejecting or accepting it 7. User gets response with offer if there is available taxi 8. User accepts or rejects the offer.
Output Condition	User gets response and has scheduled taxi drive
Exception	1. Taxi rejects the request – System forwards it to next taxi in queue 2. User rejects the offer – System cancels the request 3. No taxi available – System sends message to user that there is no taxi available 4. User loses internet connection before confirming the offer – Request is deleted from system 5. User loses internet conncetion after confirming offer – Request is considered as still valid
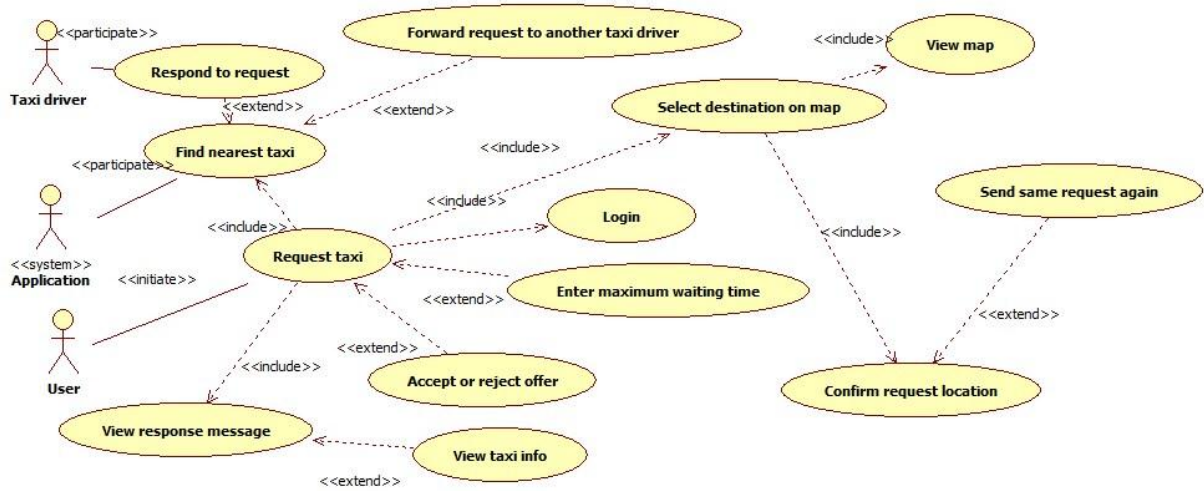
Taxi driver can respond or reject the request. It includes a smaller user case where taxi driver views the request data or, optionally, user info, and decides to accept or reject it.

The following sequence diagram is based on Scenario 2.
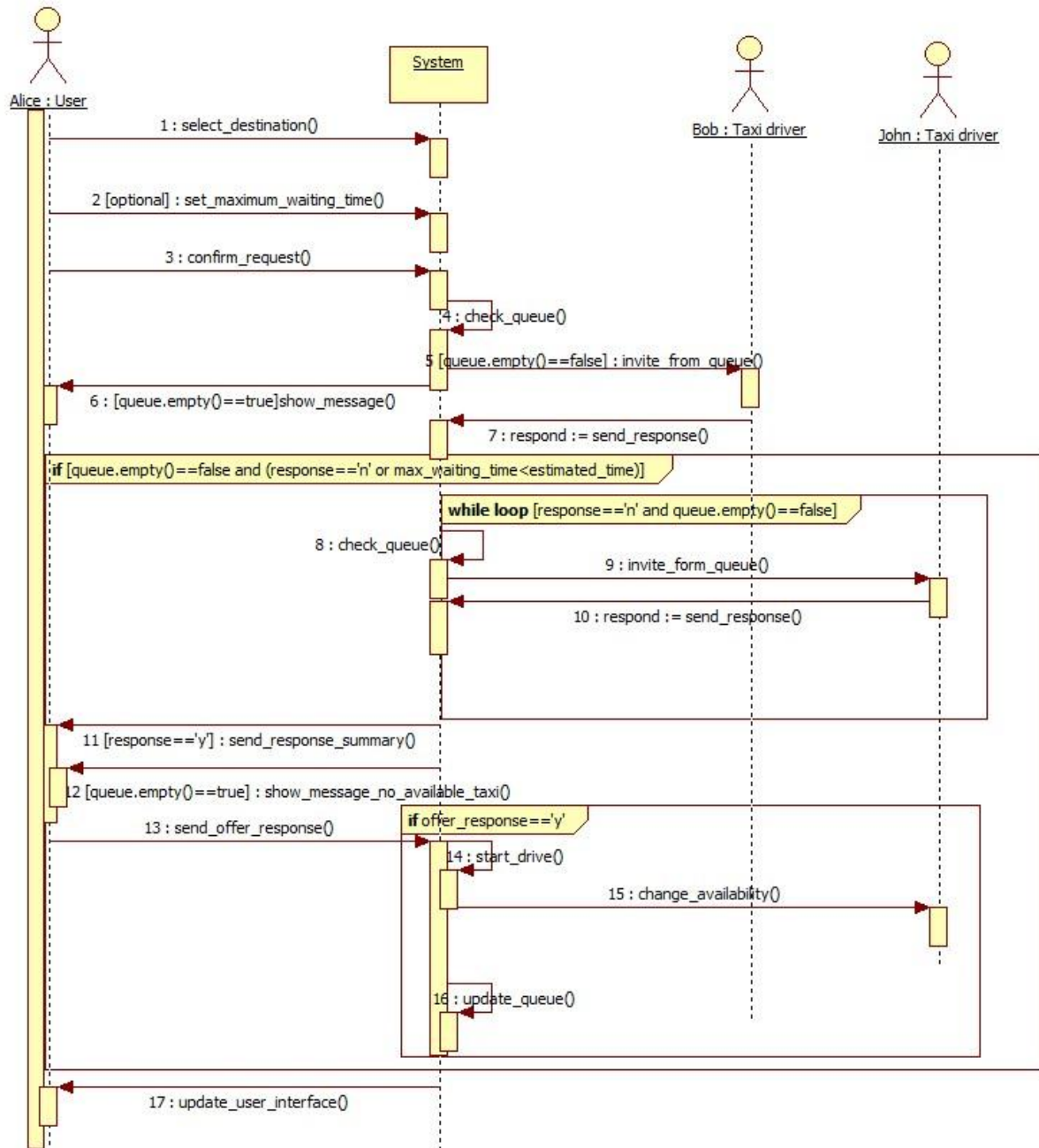
3.5.1.5 Taxi driver availability change.

Actor	Taxi driver
Goal	Availability status of taxi driver can be changed
Input Condition	1. Taxi driver must be registered. 2. Taxi driver must be logged in into mobile application.
Event Flow	1.Taxi driver clicks on change availability button.
Output Condition	Previous availability status is changed or corrected by system
Exception	1. Drive event is started or ended System can react and change availability in this cases.
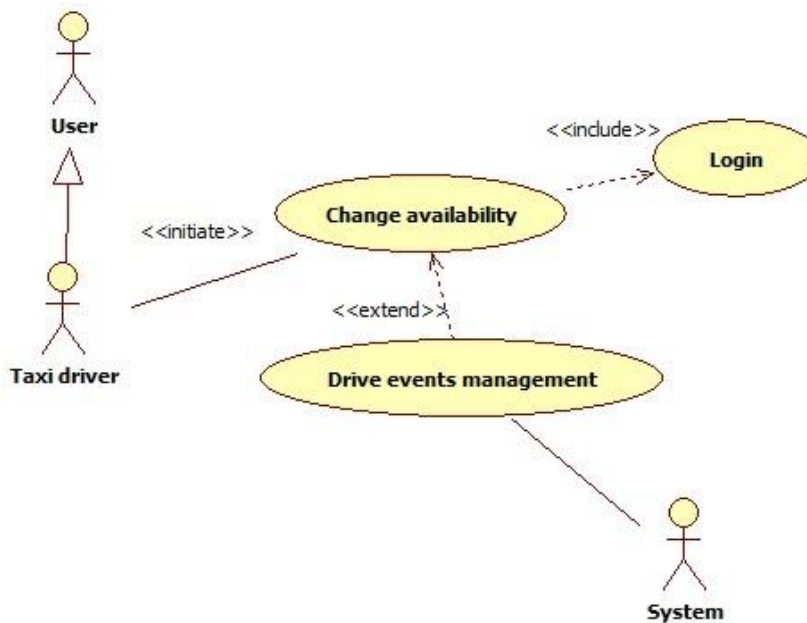
3.5.1.6 Report user

Actor	User
Goal	Report other user for bad behaviour during taxi drive
Input Condition	1. User must be registred. 2. User must be logged in into application. 3. User is in drive event.
Event Flow	1.User pushes report button. 2. User writes reason for reporting. 3. User confirms the report.
Output Condition	Report is added into database, so administration can later review it.
Exception	1. Empty report reson field 2. Internal database error User is returned to report page again.
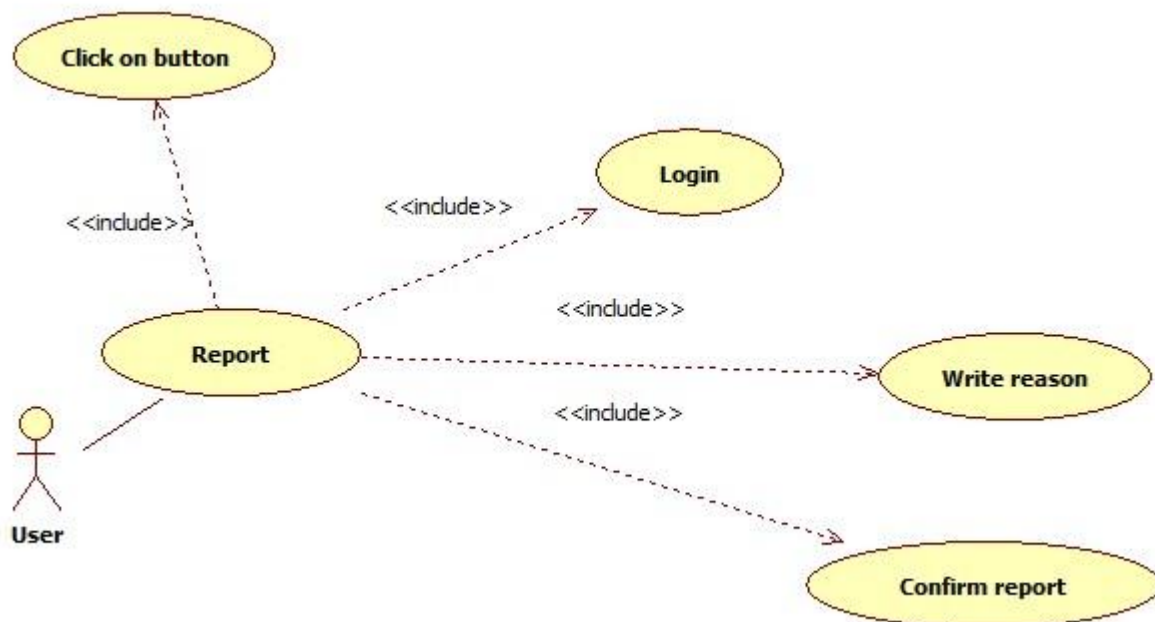
Sequence diagram based on Scenario 3.
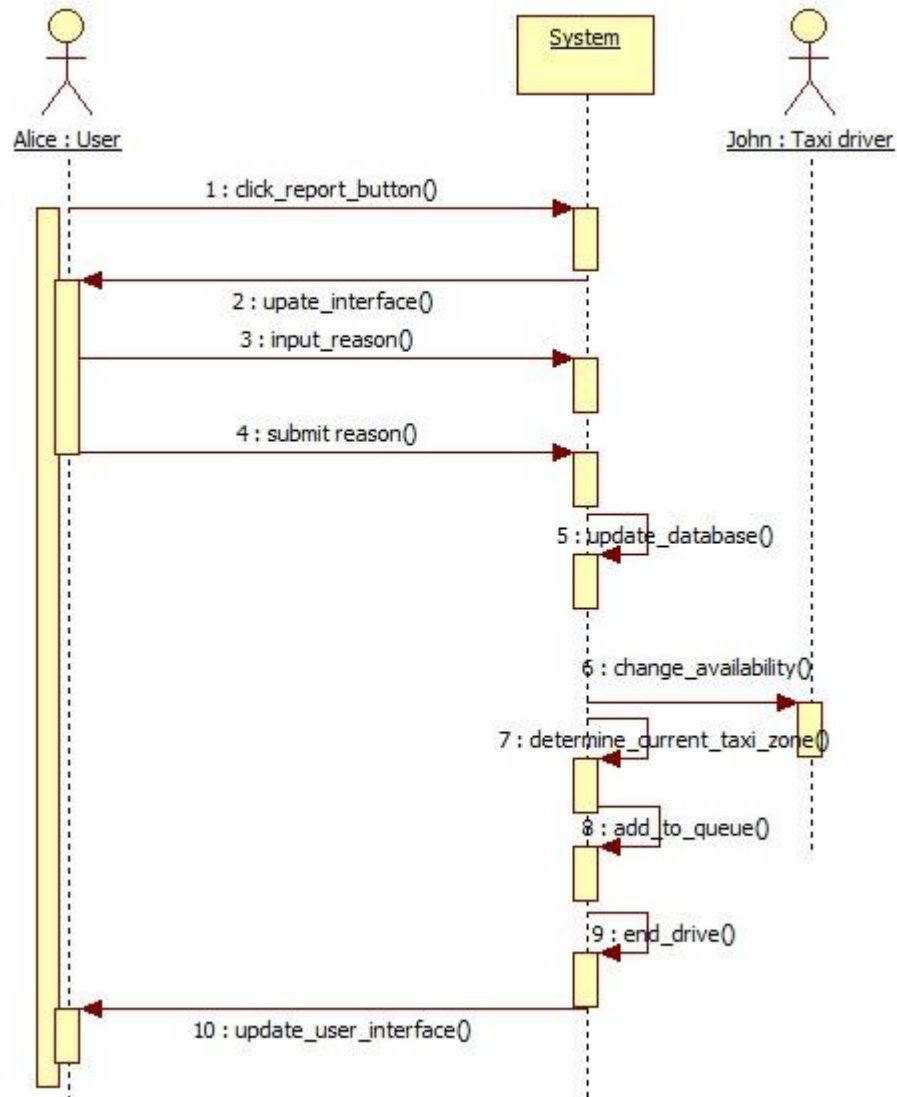
3.5.1.7 Administrator deletes reported users

Actor	Administrator
Goal	Remove users who don't take MyTaxiService serious enough from system
Input Condition	1. Administrator must be registered by government. 2. Administrator must be logged in into application.
Event Flow	1. Administrator browses user database. 2. Administrator selects desired user. 3. Administrator views reports related to user. 4. Administrator views drive events related to user. 5. Administrator clicks on delete burron. 6. Administrator confirms operation.
Output Condition	Database is updated and selected user is prevented from using MyTaxiService.
Exception	1. Internal database error. Exception is handeled alerting the administrator of the problem and application goes back to point 1. of Event Flow

Use case diagram is included as a part of more complex diagram presented in 3.5.1.9., in order to better explain how this case is used as apart of more complex use case.

3.5.1.9 Administrator promotes user to taxi driver

Actor	Administrator
Goal	User becomes taxi driver if eligible
Input Condition	1. Administrator must be registred. 2. Administrator must be logged in.
Event Flow	1. Administrator selects the desired user. 2. Administrator clicks on „Promote to driver" button. 3. Administrator enters the necessary data using form. 4. System checks data. 5. Administrator confirms taxi driver promotion.
Output Condition	Previously selected user is now also a taxi driver, database is updated.
Exception	1. Driving license invalid. 2. Taxi driver with same license already exists. 3. Car identification number invalid. 4. Car with same identification exists. 5. Internal database error. System participates by checking the data. All exception are handled alerting the administrator of the problem and application goes back to step 3 from event flow.

Check car id

Confirm promotion

Check license

<<participate>>

<<system>>
Application

<<include>>

Login

Log in

<<include>>

Insert driving license number

<<participate>>

User manipulation

<<include>>

Select user

Insert search keyword

<<include>>

<<include>>

<<extend>>

Promote user to taxi driver

<<include>>

Delete user

View user details

<<extend>>

View report

Insert car model

<<include>>

Switch taxi driver to user

<<include>>

Confirm action

<<extend>>

<<extend>>

Edit user

Admin

<<include>>

<<include>>

Insert car ID number

View drive

Emergency driver or not

Confirm deletion

<<include>>

Login

Admin

View report

<<include>>

<<include>>

Select user

Select report

View user info

<<include>>

Select drive

Login

<<include>>

View User

<<include>>

Admin

View Drive

<<extend>>

<<extend>>

View Taxi Driver

<<extend>>

View Reports

44

Sequence diagram that explains how the realization in practice is based on Scenario 5 and presented below.
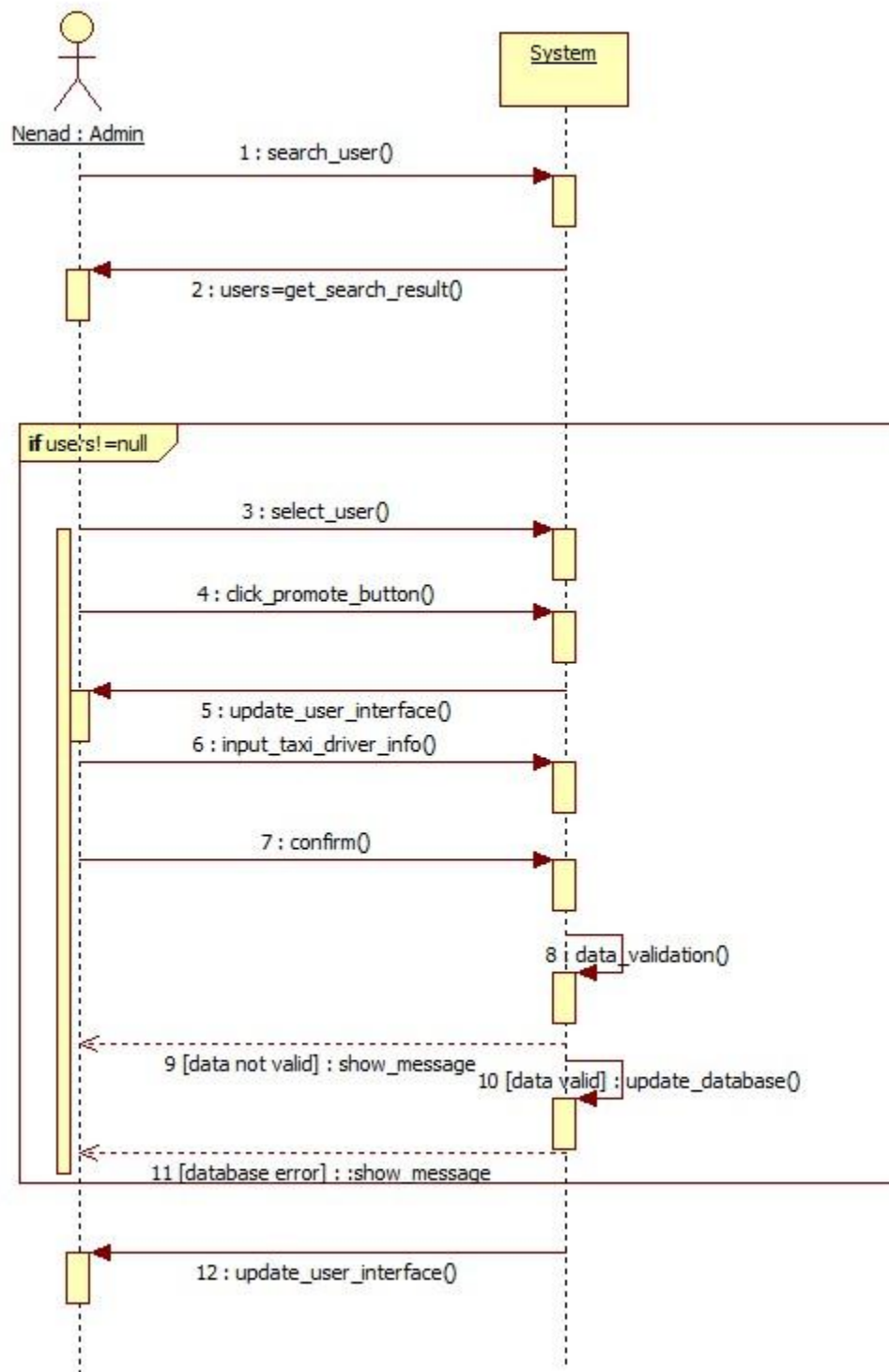
3.5.1.10   User sends S.O.S signal in case of an accident   .

Actor	User
Goal	Request emergency vehicle to a destination where the accident is reported.
Imput Condition	1. User is registered and logged in. 2. User is into drive event.
Event Flow	1. User reports energency by clicking on S.O.S button.
Output Condition	Emergency vehicle is dispatched from queue.
Exception	1. No vehicle for taxi zone available. System searches for a vehicle in other taxi zones until a vehicle is available
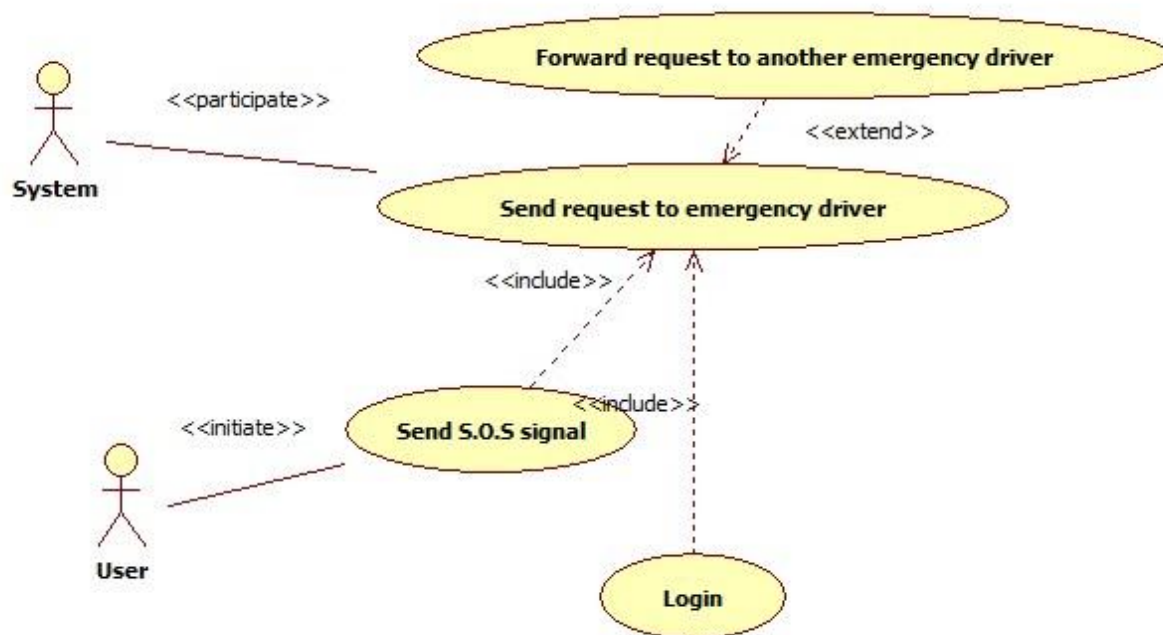
3.5.1.11 Log out

Actor	User
Goal	Log out from system and lose access to service features
Imput Condition	1. User already registered and logged in.
Event Flow	1. User click on „Log out" button
Output Condition	User is back to home page and must log in again in order to use MyTaxiService.
Exception	NULL
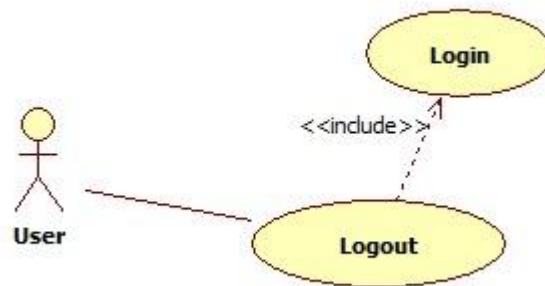
3.5.2 Class Diagram

In this part, the class diagram is presented. This diagram will be updated during the design and development process, because all the operations are missing at the moment, as it can be seen. Mostly, the requirements-level related entities as a part of the application are presented.

3.5.3   State Machine Diagram

Application state diagram

In this part, the state machine diagram is presented. The following state machine diagram gives a simplified vision and overview of the entire application. Together with Mockups from 3.1.1, this diagram gives a better introduction to how system resonds to user action in different cases, corresponding to different events from real world and user actions.

Availability change state diagram

This state diagram shows in which cases the taxi driver is considered available and in which cases is considered as not available. So, this are the facts presented here and the situations in which the change occurs are described.
As it can be seen, availability could be changed manually, but a part of the system monitors the drive events and changes the availability in case of some events.
In order to be a part of the queue, taxi driver needs to be „available".
After the beginning of drive event, taxi driver becomes automatically „unavailable".
When the drive ends, the taxi driver becomes „available" again.
After the report of any side, the drive ends and taxi becomes „available".
When SOS signal is sent, the drive ends and taxi becomes available, but taxi driver can still manually set availability. This could be very useful in case when the drive is ended and taxi is not damaged and user can continue drive.
But, in case when taxi driver can't countinue his/her job, he/she is given ability to manually change the availability to „unavailable".
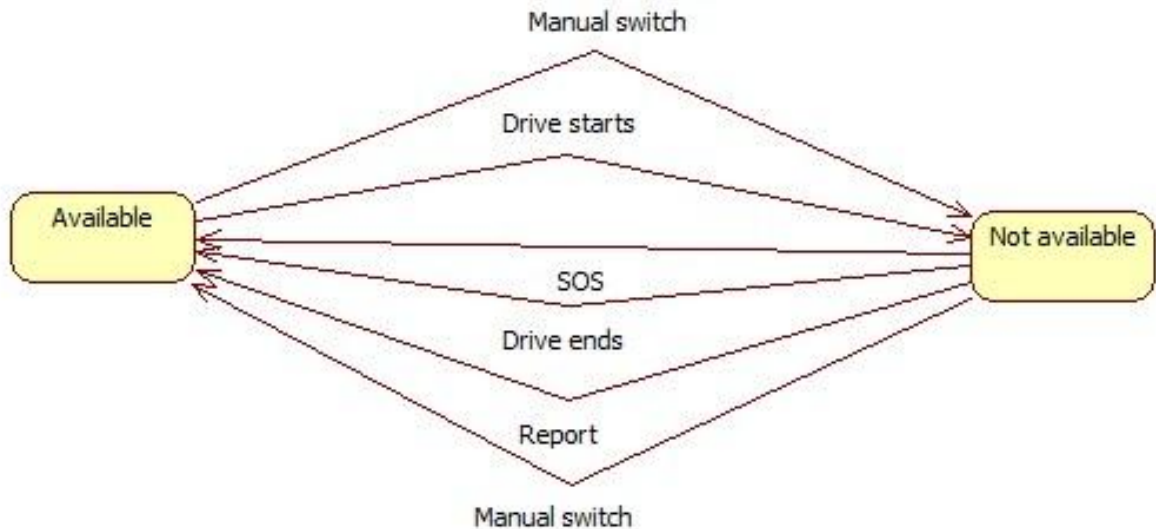
3.5.3   Activity Diagram

In this part, the activity diagram for taxi scheduling is presented. The following activity diagram gives an overview which activities are performed by the system, and the idea how the algorithm for scheduling works and behaves in certain conditions – and will support the development of the algorithm in design phase.
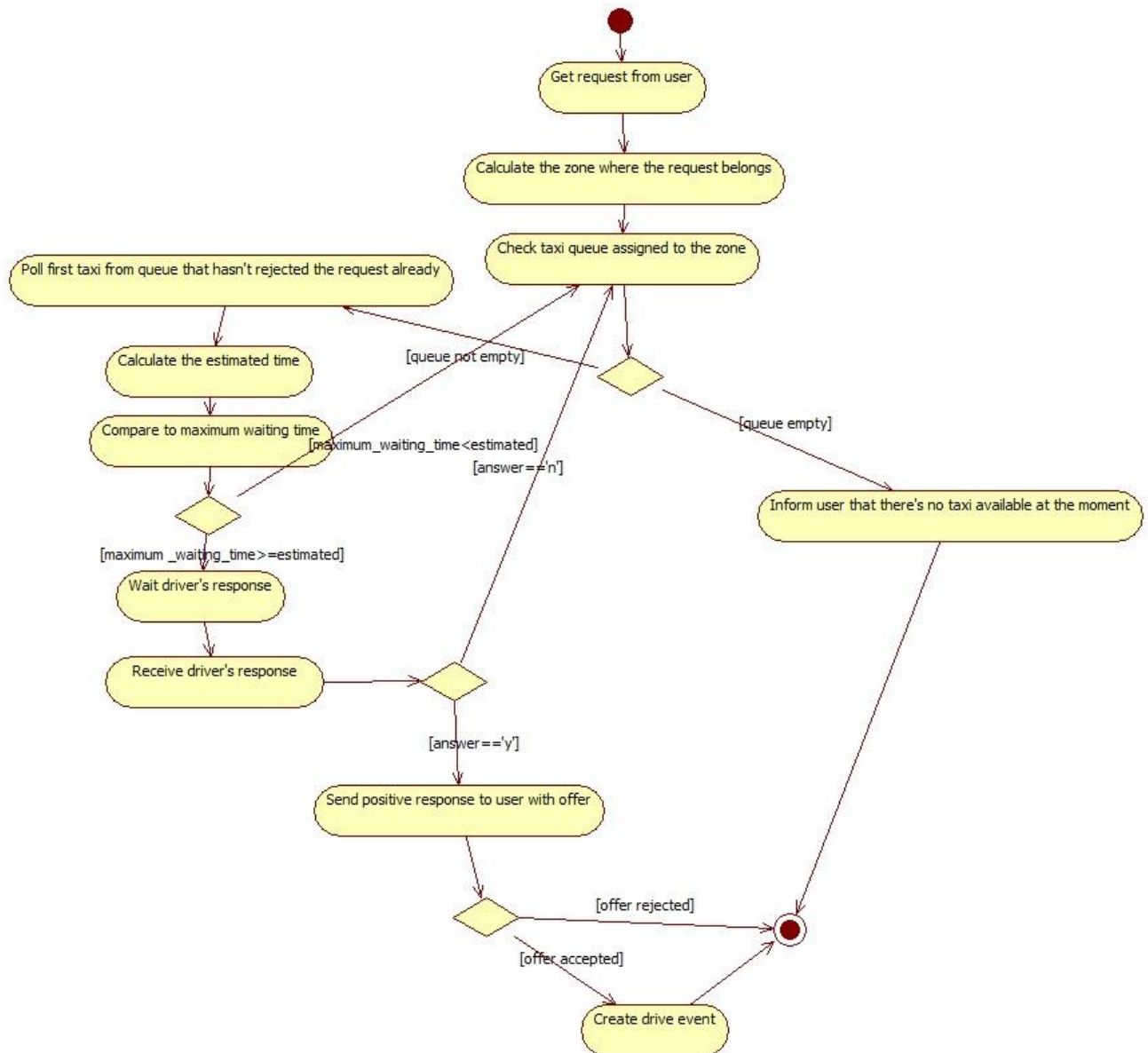
3.6 Non Functional Requirements

3.6.1 Performance Requirements

We assumed that user will get answer within maximum 180 seconds (searching for taxi is limited to maximum 180 seconds per request in order to load balance), but, if taxi chages availability during that period, system needs to respond in real time to that changes. So, the response time of the system, to be usable in real time should be less than 1 second per action, because unstability of internet connection in certain areas is assumed.

In terms of maximum active users at certain point in time, considering that this application will be used in a city environment, the maximum number of simultaneus users should be at least 5 000.

3.6.2 Minimum software, hardware and communication interface requirements

3.6.2.1 Server-side application

Software

OS	Any OS that supports Java Virtual Machine, DBMS and Application Server
DBMS	MySQL 5.6 or newer
Java Virtual Machine	7.0 or greater
Application server	GlassFish 4.1 or greater

Hardware

Memory	8 GB or more
Free storage space	20 GB or more

Communication

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

3.6.2.2 Client-side web-browser application

Software

Web browser	Any

Hardware

Display resolution	1024 x 600 or greater

Communication

Active Internet connection.

3.6.2.2 Client-side mobile application

Software

OS	Android 4.0 or greater

Hardware

Any GPS-enabled Android 4.0 or greater device, with resolution at least 800x480.

Connection

Active Internet connection and GPS turned on.

3.6.3   Software System Attributes

3.6.3.1 Availability: The application would be accessible online 24/7. To achieve this it is necessary to use a dedicated server, but, in order to guarantee even better availability, the whole system could be hosted on already available cloud platform. System downtime should be less than 30 minutes per month.

This solution give more scalability to performance required by the system and could reduce the cost for a dedicated server, mantaining a high level of performance especially in case of full load (advanced load balancing).

3.6.3.2 Maintainability: Regular updates are planned, and the change of princing estimation should be as flexible as possible, because each city's government could change pricing policy any time and each city could have different taxi policy. API is provided that could be used to extend system by other developers and system will be documented in details. So, in case of critical situations or other cases, system could be modified or extended.

3.6.3.3 Scalability: The system should be expandable to handle greater number of active users and also will be extendable – so the new functionalities could be added in future aas described in 2.5.

3.6.3.4 Design constraints: The system will inherit all the design constraints of the programming languages (Java) , frameworks used (Glassfish,Java Eclipse, Android Studio) and DBMS (MySQL). The part of the system that deals with maps and GPS coordinates will rely on Google Maps API as an external API developed by Google and API provided in Android Studio that deals with coordinates and maps.

3.6.4   Security

3.6.4.1 External interface side of MyTaxiService application implements a login authentication to protect the information of users.

There are no special requirements for the user side of the application, but user password has to be at least 8 characters long, containing at least one capital letter, one special symbol and one number. User can change password anytime using the profile modification page. In future, captcha verification could be added in order to prevent attacks.

In future, face recognition based authentication system will be developed in order to identify administrators, together with password. Combination of these two technologies will defend system from being managed by unauthorized people and make system more durable to attacks.

3.6.4.2 Application side would implement defense from SQL injection. Also, the application would use HTTPS instead of HTTP connection, in order to guarantee communication integrity and confidentiality.

3.6.4.3 Server Side – will use separated web, application and database part, with firewall between each of them to prevent unauthorized users from access. Because cloud platform use is still considered (Amazon Services), there might be different solution, offered by Amazon.

One of the possible solutions is to use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3) , where each object is encrypted with a unique key employing strong multi-factor encryption. As an additional safeguard, it encrypts the key itself with a master key that it regularly rotates. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-

bit Advanced Encryption Standard (AES-256), to encrypt data.

4   Appendix

4.1   Alloy

 In this part of the document, Alloy model is presented – its signatures, facts, asserts, predicates and along with its results. The whole model file (Alloy.als) could be found under Deliveries folder in my github repository.

4.1.1   Signatures

In this part, definition of datatypes are given – from common types to domain entities.

Common data types found in almost every programming language, necessary for this model

```
sig Strings{}
sig Char{}
sig Integer{}
```

Auxiliary data types used in this problem – related to space and time.

```
sig Time{
hours: one Integer,
minutes: one Integer
}

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

sig TimeDate{
timeStamp: one Time,
dateStamp: one Date
}

sig Location {
coordinates: one Strings,
streetName: one Strings,
streetNumber:one Integer
}

sig CarNumber{
id: one Strings
}
```

Entity signatures

Signatures of all the entities that appear in this problem.

First, the signatures of actors are given (left). These, are in fact, abstractions of a people who use the application.

Then, communication entity signatures, of that are used in protocol are given (right).

```
//People that use application
abstract sig Visitor{}

sig Guest extends Visitor{}

sig User extends Visitor{
firstname: one Strings,
lastname: one Strings,
username:one Strings,
password:one Strings,
mobilephoneNumber: one Strings,
gender: one Char,
picturePath: one Strings,
currentLocation:one Location,
currentZone:one TaxiZone,
codiceFiscale: one Strings
}


sig TaxiDriver extends User{
carId:one CarNumber,
licenseNumber:one Strings,
carModel:one Strings,
availability:one Char,
emergency:one Char
}

sig Admin extends Visitor{
username: one Strings,
password: one Strings
}
```

```
//Communication entities

sig Message{
startpoint:one Location,
endpoint: one Location,
timestamp: one TimeDate,
sender: one User,
receiver:one User
}


sig Request extends Message{
maximumWaitingTime: lone Time
}


sig Response extends Message{
accepted:one Strings,
estimatedTimeWaiting:one Time
}

sig Report extends Message{
reason: one Strings,
id:  one Strings
}
```

And finally, the last tree signatures are written :

```
sig Drive{
driveId: one Strings,
userRequest: one Request,
taxiResponse:one Response,
reports: set Report
}
```

```
//System-related entities
sig TaxiZone{
zoneId: one Strings,
requestQueue: set Request,
carIdQueue: set CarNumber,
centerPoint: one Location,

}
```

-Drive – an abstraction of a drive event

-TaxiZone – an abstraction of taxi zone

-Scheduler – part of the system which deals with taxi zones and their queues

4.1.2 Facts

Fact is used to *force* something to be true of the model. Facts are always „on".

A trivially false fact results in there being no solutions to the model no matter what the rest of the model looks like. This is an extreme version of overconstraining your model. Overconstraint is dangerous, since it means that any other assert statement that you check will trivially have no solutions, thus potentially masking both errors in your model and (worse) failings in the subject matter you are modeling.

In what follows, facts about MyTaxiService are written, after consideration of domain rules, constraints and properties.

```
//noEmpty-alike facts

fact noEmptyTime{
  all t:Time | (#t.hours=1) and (#t.minutes=1)
}


fact noEmptyDate{
  all d: Date | (#d.day=1) and (#d.month=1) and (#d.year=1)
}

fact noEmptyTimeDate{
  all td:TimeDate | (#td.timeStamp=1) and (#td.dateStamp=1)
}

fact noEmptyLocation{
  all l:Location | (#l.coordinates=1) and (#l.streetName=1) and
(#l.streetNumber=1)
}

//User mandatory fields
fact noUserMissingInfo{
        all u:User | (#u.username=1) and (#u.password=1)
}
```

```
//User mandatory fields
fact noSenderMissing{
        all m:Message | (#m.sender=1)
}



//Admin mandatory fields
fact noUsernameMissingAdmin{
        all a:Admin | (#a.password=1)
}

fact noPasswordMissingAdmin{
        all a:Admin | (#a.password=1)
}
//



//Scheduler must have at least one zone
fact minZones{
  all s:Scheduler| (#s.zones>1)
}




//Taxi driver is eligible if and only if has a car and license number
fact noUnlicensedAndCarlessTaxiDriver{
  all t:TaxiDriver | (#t.carId=1) and (#t.licenseNumber=1) and
(#t.carModel=1)
}



//No duplicate and overlap-alike facts



//No duplicate usernames
fact noDuplicateUser{
  no  u1,u2: User |(u1!=u2) and  (u1.username=u2.username)
}

//No possibility for one user to register more than once with a same
fiscal code
```

```
fact noFakeProfiles{
  no u1,u2: User | (u1!=u2) and (u1.codiceFiscale=u2.codiceFiscale)
}


//



//No self-communication allowed - all the requests,responses and report
are towards differnt users

fact noSelfCommunication{
  all u:User, m:Message | not ( (m.sender=u) and (m.receiver=u) )
}

//No sending requests, responses or reports to same location

fact noSameStartAsEnd{
  all m:Message | not ( m.startpoint=m.endpoint )
}



//No taxi zones with same center points

fact noSameZones{
  no z1,z2: TaxiZone | (z1!=z2)  and (z1.centerPoint=z2.centerPoint)
}

//No taxi zones with same ids

fact noSameIdZones{
  no  z1,z2: TaxiZone | (z1!=z2) and  (z1.zoneId=z2.zoneId)
}



//No car number belonging to different owners at the same time
//fact noSameCarOwner{
  //no  cn1, cn2: CarNumber | (cn1!=cn2) and  (cn1.owner=cn2.owner)
//}

fact noSameCarNumbers{
  no  cn1, cn2: CarNumber | (cn1!=cn2) and (cn1.id=cn2.id)
}
```

```
//No two taxi drivers with same car
fact noSameDriverCars{
  no td1, td2: TaxiDriver | (td1!=td2) and  (td1.carId=td2.carId)
}

//No taxi drivers with same license id allowed
fact noSameLicense{
  no  td1,td2: TaxiDriver | (td1!=td2) and
(td1.licenseNumber=td2.licenseNumber)
}

//No two drives with same ids

fact noSameDriveId{
  no d1,d2: Drive | (d1!=d2) and (d1.driveId=d2.driveId)
}


//A taxi drivers can respond to requests that are from users in that taxi
zone

fact responseCondition1{
no r:Request | r.sender. currentZone!=r.receiver.currentZone
}

fact responseCondition2{
no r:Response | r.sender. currentZone!=r.receiver.currentZone
}

//No two requests from same sender can be in one queue of request per
taxi zone

fact noTwoZonesSameSender{
 no r1,r2:Request | some z:TaxiZone|
 r1!=r2 and (r1 in z.requestQueue) and
 (r2 in z.requestQueue) and (r1.sender=r2.receiver)
}


//One request can't belong to queues of two different taxi zones at the
same time

fact noTwoZonesSameRequest{
 no r: Request | some z1, z2:TaxiZone |
 z1!=z2 and (r in z1.requestQueue) and
 (r in z2.requestQueue)
}
```

```
//Are always the right actors reported in a drive
fact driveReportsRule{
all d:Drive,r:Report| (r in d.reports) and (   (r.sender
=d.userRequest.sender) or (r.sender=d.userRequest.receiver)    )
}

//The condition when a drive is considered correct
fact correctDrive{
all d:Drive| (d.userRequest.sender = d.taxiResponse.receiver)
}




//One taxi can't belong to queues of two different taxi zones
fact oneTaxiCanOnlyBeInOneTaxiZone {
 no t: TaxiDriver | some z1, z2:TaxiZone |
 z1!=z2 and (t.carId in z1.carIdQueue) and
 (t.carId in z2.carIdQueue)
}


//All zones belong to one scheduler per city

fact allBelong{
all t:TaxiZone| some s:Scheduler| (t in s.zones)
}
```

4.1.3 Asserts

While a fact is used to force something to be true of the model,
an assert is a claim that something must already be true due to the rest
of the model.


```
//Checking if there are  same requests in two zones
assert NoTwoZonesSameRequest{
 no r: Request | some z1, z2:TaxiZone |
 z1!=z2 and (r in z1.requestQueue) and
 (r in z2.requestQueue)
}
check NoTwoZonesSameRequest for 5


//Checking if there is no self-message

assert NoSelfMessage{
 no m: Message | m.sender=m.receiver
}
```

```
check NoSelfMessage for 5




//Checking add request
assert addRequest{
all r:Request, t1:TaxiZone,t2:TaxiZone | (r not in t1.requestQueue) and
addRequestToTaxiZone[r,t1,t2] implies (r in t2.requestQueue)
}

check addRequest for 5

assert deleteRequest{
all r:Request, t1:TaxiZone,t2:TaxiZone | (r  in t1.requestQueue) and
removeRequestFromTaxiZone[r,t1,t2] implies (r not in t2.requestQueue)
}
check addRequest for 5

//Are always the right actors reported in a drive
assert driveReports{
all d:Drive,r:Report| (r in d.reports) and (    (r.sender
=d.userRequest.sender) or (r.sender=d.userRequest.receiver))
}
check driveReports for 5




//DriveRule
assert driveRule{
all d:Drive| (d.userRequest.sender = d.taxiResponse.receiver)
}
check driveRule for 5




//Should find counterexample!- Simulating a situation where two taxi
drivers register with same car
//A trivially false assert will result in counterexamples being found
when it is checked.
//Since anything makes such an assertion false, any example of the system
will be a counterexample.
assert assignSame1{
all cn:CarNumber, t1,t2,t3,t4:TaxiDriver| AssignCarNumber [cn, t1, t2]
and AssignCarNumber [cn,t3,t4]
}
check assignSame1 for 10
```

```
//Shouldn't find a counterexample this time
assert assignSame2{
all cn:CarNumber, t1,t2,t3,t4:TaxiDriver| AssignCarNumber [cn, t1, t2]
and AssignCarNumber [cn, t3, t4] implies (t4=t2)
}
check assignSame2 for 10

//Should illustrate that it is not possible to send message with same
sender and receiver
//It prevents respond the request of yourself

assert selfMessage{
all m1,m2,m3:Message,u:User | AssignReceiver[m1,m2,u] and
AssignSender[m2,m3,u]
}
check selfMessage for 5




assert RegisterUser1{
all u1,u2,u3,u4: User,name:Strings | RegisterUser[u1,u2,name] and
RegisterUser [u3,u4,name]
}
check RegisterUser1 for 5




assert RegisterUser2{
all u1,u2,u3,u4: User,name:Strings | RegisterUser[u1,u2,name] and
RegisterUser [u3,u4,name] implies (u4=u2)
}
check RegisterUser2 for 5
```

4.1.4   Predicates

This is the part where the predicates that are used are presented with the previous assert to verify the model.

```
//To genrate world example
pred show(){
#User=2
#TaxiDriver=2
#Guest=2
#Drive=2
#Strings=2
#Char=2
#Integer=2
#Time=2
#Date=2
#TimeDate=2
#Request=2
#Response=2
```

```
#Location=3
#Scheduler=1
#TaxiZone=2
#CarNumber=2
}
run show for 5




//Adding request to a taxi zone
pred addRequestToTaxiZone(r:Request, t1,t2:TaxiZone)
{
r not in t1.requestQueue implies t2.requestQueue=t1.requestQueue+r
}
run addRequestToTaxiZone for 5

//Removing request from a taxi zone
pred removeRequestFromTaxiZone(r:Request, t1,t2:TaxiZone)
{
t2.requestQueue=t1.requestQueue-r
}
run removeRequestFromTaxiZone for 5


//Assigning taxi driver a car number
pred  AssignCarNumber(cn:CarNumber, td1,td2:TaxiDriver)
{
td2.carId=td1.carId+cn
}
run AssignCarNumber for 5


//Simulating how system assign recevier from schedule
pred AssignReceiver(m1,m2:Message,u:User)
{
m2.receiver=m1.receiver+u
}
run AssignReceiver for 5



//Simulate sending of the message
pred AssignSender(m1,m2:Message,u:User)
{
m2.sender=m1.sender+u
}
run AssignReceiver for 5
```

```
//Registration of the user
pred RegisterUser(u1,u2:User,name:Strings)
{
u2.username=u1.username+name
}
run RegisterUser for 5
```

4.1.5   Results

The screenshot of the Alloy Analizer software is presented.

```
#1: No counterexample found. NoTwoZonesSameRequest may be valid.
#2: No counterexample found. NoSelfMessage may be valid.
#3: No counterexample found. addRequest may be valid.
#4: No counterexample found. addRequest may be valid.
#5: No counterexample found. driveReports may be valid.
#6: No counterexample found. driveRule may be valid.
#7: Counterexample found. assignSame1 is invalid.
#8: No counterexample found. assignSame2 may be valid.
#9: Counterexample found. selfMessage is invalid.
#10: Counterexample found. RegisterUser1 is invalid.
#11: No counterexample found. RegisterUser2 may be valid.
#12: Instance found. show is consistent.
#13: Instance found. addRequestToTaxiZone is consistent.
#14: Instance found. removeRequestFromTaxiZone is consistent.
#15: Instance found. AssignCarNumber is consistent.
#16: Instance found. AssignReceiver is consistent.
#17: Instance found. AssignReceiver is consistent.
#18: Instance found. RegisterUser is consistent.
```

As it can be seen, three counterexamples are found, as it was expected in asserts.
A trivially false assert will result in counterexamples being found when it is checked. Since anything makes such an assertion false, any example of the system will be a counterexample. As a result, the solution returned may appear completely unrelated to the assertion it falsifies.

Alloy software generates graphical view of the world generated, as it can be shown for command #12.
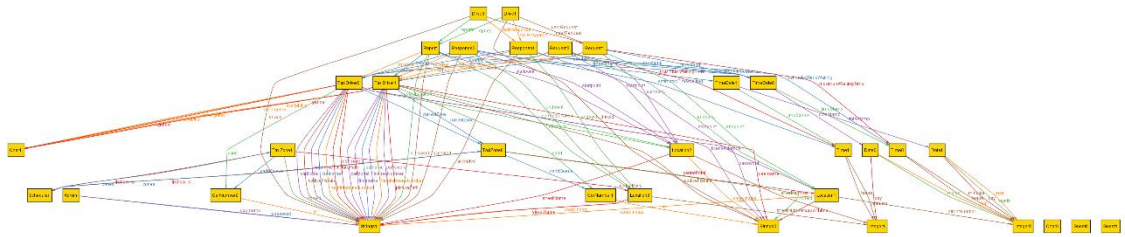
4.2   Software and tools used

- Microsoft Office Word 2013: To create and redact this document

- StarUML (http:// http://staruml.io/): to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams, State Machine Diagram and Activity Diagram

- Alloy Analyzer 4.2 (http://alloy.mit.edu/alloy/): to prove the consistency of the presented model.

- NinjaMock (https://ninjamock.com/): to create mockups for both mobile and web variants of the application.

- Github   (reporsitory: https://github.com/penenadpi/Software-Engineering-2-Project/Deliveries)

4.3   Hours of works

The time spent for constructing this document and Alloy model:

- Nenad Petrovic: ~ 52 hours.

5   Revision

Version 1.3

1. Statechart diagram is added related to change of taxi driver availability taxi drivers in certain situations in order to describe it better.
2. Minor corrections in The World and the Machine part of the document – both the picture and page.
3. Maintainability – added a fact that price estimation could be changed, so the system should be modifiable in terms of pricing.
4. Goals 5 and 6 are updated with explicitly stated all the domain assumptions.
5. Design constraints and Scalability sections updated.
6. Several typos are corrected and missing words are added to several sentences related to goals.

7. Simpliefied sequence diagrams, more suitable for RASD documents are added. More complex variants from version 1.0 are going to be expanded for next phase and are exchanged with simpler variants.