POLITECNICO DI MILANO

Computer Science and Engineering

Project of Software Engineering 2: "*myTaxiService*"

Requirements Analysis and Specifications Document

Author: Andrea Maioli (mat. 852429)
Reference Professor: Mirandola Raffaela

Summary

1. Introduction **3**
1.1. Purpose 3
1.2. Actual System 3
1.3. Scope 3
1.4. Actors 4
1.5. Goals 4
1.6. Definitions, Acronyms, Abbreviations 5
1.7. References 6
1.8. Overview 6
2. Overall Description **7**
2.1. Product Perspective 7
2.2. User Characteristics 7
2.3. Constraints 7
2.4. Assumptions and Dependencies 7
2.5. Future possible implementations 8
3. Specific Requirements **9**
3.1. External Interface Requirements 9
3.1.1. User Interfaces 9
3.1.2. Hardware Interfaces 17
3.1.3. Software Interfaces 17
3.1.4. Communication Interfaces 17
3.2. Functional Requirements 18
3.3. Performance Requirements 19
3.4. Software System Attributes 19
3.4.1. Availability 19
3.4.2. Security 19
3.4.3. Maintainability 19
3.5. Scenarios 20
3.6. Use Cases 22
3.7. Class Diagram 27
3.8. UML 28
3.9. Sequence Diagrams 29
3.10. State Chart Diagrams 38
4. Appendix **40**
4.1. Alloy 40
4.1.1. Code 40
4.1.2. Generated World 46
4.1.3. Results of analysis 46
4.2. Software and tools used 47
4.3. Hours of work 47
4.4. Revision 47

1. Introduction

1.1. Purpose

This document represent the Requirement Analysis and Specification Document (RASD) and its main goal is to completely describe the system in terms of functional and non-functional requirements, analyze the real need of the customer, show the constraints and the limit of the software and simulate the typical use cases that will occur after the development.
This document is addressed to:

- Developers and Programmers that have to implement the requirements.
- Testers who have to determine if the requirements have been met.
- Project manager that measures and controls the development process.
- System Analysts and Requirements Analysts that can integrate other systems with the described one.

1.2. Actual System

The only system available, until now, is the call center infrastructure that dispatches all the taxi requests and interacts with drivers, operators and customers:

- Each operator of the call center has at his own disposition a telephone, a computer and a radio device.
  When receiving a call for a trip request the operator insert the pick-up point, provided by the customer, in the computer, which display a map with all the taxi available in a 2km^2 area.
  The choice of the vehicle that will respond to the request is made by the operator, and communicated to the driver using the radio device. If the driver is not available, the request is forwarded to another driver.
- GPS information and taxi status are constantly fetched by a server connected to a radio-antenna that store these informations inside a database that is consulted by each operator's computers.
- Each taxi vehicle is equipped with a radio device and a taximeter that is connected to it.
  The radio devices permits the communication with a call center operator, and sends the vehicle status (busy or free) and the GPS position to the call center infrastructure.
  The taximeter is used for calculating and displaying the trip fees, and is activated by the driver when he accepts a trip request provided by a call center operator.

1.3. Scope

The aim of the project is to create a web application that simplify the access to the taxi service.
The application is used by the registered users to request a trip, by the call center operator to manage trip request calls and special events, and by the taxi driver to accept/decline trip requests and to notify exceptional events.
The city is divided in taxi zone of 2km^2 each and each zone is associated in a queue.
When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone.
If the taxi driver accepts the request, the system will send the confirmation to the passenger. If not, then the system will forward the request to the second in the queue and move the first taxi in the last position of the queue.

1.4. Actors

- Guest: is a person who want to access to myTaxiService application. He/She can sign up to the system.
- Passenger: is a registered user who want to request a taxi for a trip. He/She can perform this request through the web application (using a web browser), the mobile app or by the telephone. All the requests from the application must be made by a registered user, which can also check the status of such request.
- Call center operator: he/she is in charge of taking the requests coming from the telephone and inserting them inside the system, through an apposite section inside the application. He/She can manage particular situations (such as no passenger found in the pick-up point, taxi problems, ecc…). All call center operators are registered users with a special level of clearance.
- Taxi driver: is who bring the passengers from the pick-up point to the destination point. He/She use the web application in order to accept/decline the requests and for reporting special events.

1.5. Goals

1. Simplify the access of passengers to the taxi service, allowing them to:
    a. Register an account using the mobile application or the website.
    b. Log into the system.
    c. Log out from the system.
    d. Request a taxi trip using the web application, the mobile application or the telephone.
    e. Check the status of the current request (only if made using the application).
2. Guarantee a fair management of the taxi queue.
3. Simplify the communications between the driver and the call center infrastructure, allowing the call center operator to:
    a. Log into the system with a special level of clearance.
    b. Log out from the system at the end of the working day.
    c. Insert a request coming from a call in the system, which will manage it.
4. Simplify the methods of access to the taxi trip requests system for the taxi drivers, allowing them to:
    a. Log into the system with a special level of clearance that is different from the one of the call center operator.
    b. Set its status in available or busy.
    c. Log out from the system at the end of the working day.
    d. Get on his/her tablet a request if his/her status is available.
    e. Accept or decline an incoming request.
    f. Report an exceptional event that will be manage from the system.
5. Allow the system to:
    a. Handle requests and their assignment.
    b. Manage each taxi queue.
    c. Handle exceptional events.

1.6. Definitions, Acronyms, Abbreviations

- **API**: stands for *Application Programming Interface* and consist in a set of routines, protocols and tools for building software application. They can facilitate the integration of new features into existing applications, even external from the considered one.
- **Call Center**: is a physical place where customer and other telephone calls are handled by an organization.
- **Call Center Operator**: is a person who works in the call center and answers the phone calls.
- **Credentials**: combination of email and password.
- **Cross Site Scripting**: vulnerability that permits the injection of client-side code into a page.
- **Data Plan**: subscription to a cellular or other wireless carrier for the transfer of data over its network. It may be an unlimited-use plan or based on the actual amount of data transferred.
- **DBMS**: stands for *Data Base Management System* and consist in a system software for creating and managing databases. It provides users and programmers a systematic way to create, retrieve, update and manage data.
- **Drop-off point**: location in which the passenger will be drop off by the taxi. It is specified by the passenger when entering the taxi vehicle.
- **GPS**: stands for *Global Positioning System* and is a system that uses satellites to determine the position of a vehicle.
- **Hashing function**: is a function that can be used to map data of arbitrary size to data of fixed size and doesn't permit to get the correspondent input that produces a certain hashing value, without trying the hashing function on each possible input.
- **HTTP**: stands for *HyperText Transfer Protocol* and is the underlying protocol user by the World Wide Web. It defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands.
- **HTTPS**: stands for *HyperText Transfer Protocol Secure* and basically consists in a HTTP communication over an SSL tunnel, that ensures the security of the communication.
- **Logged user**: a user that has successfully performed the login in the system.
- **NFC**: stands for *Near Field Communication* and is a short-range wireless connectivity standard that uses magnetic filed induction to enable communication between device.
- **Non-logged user**: a generic user not authenticated in the system.
- **Pick-up point**: location in which the passenger will be picked up from the taxi. It is automatically specified by the application or by the passenger (if the request comes from a phone call).
- **Radio Device**: device placed in the taxi vehicle which permits radio communications between the driver and the call center.
- **SQL-Injection**: vulnerability that permits execution of arbitrary sql code.
- **Taximeter**: mechanical or electronic device installed in taxicabs and auto rickshaws that calculates passenger fares based on a combination of distance travelled and waiting time.
- **Taxi Zone**: area of $2km^2$ in which a taxi queue operates.
- **Trip Request**: request for a taxi coming from a user.

1.7. References

- Specification Document: myTaxiService.pdf (part I)
- IEEE Std. 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- “Corriere della Sera” article for taxi density over Milan: http://milano.corriere.it/caso_del_giorno/articoli/2006/03_Marzo/01/caso.shtml
- GitHub Repository: https://github.com/Maiux92/ingsw2project/

1.8. Overview

This document is structured in four parts:

1) Introduction: provides an overview of the entire document and some information about the software.
2) Overall Description: describes general factors that affect the product and its requirements.
3) Specific Requirements: this part list the requirements, typical scenarios and use cases.
4) Appendix: this part contains information about the alloy part and tools used to create this document.

2. Overall Description

2.1. Product Perspective

The application to be created is a web and mobile application which won't interact with the existing infrastructure. It must provide an API for integration with future projects.

2.2. User Characteristics

The users that will use this application must be able to use a web browser, a smartphone or a tablet and must have access to internet.

2.3. Constraints

- myTaxiService web application must fit every screen resolution without compromising its functionalities.
- myTaxiService web application functionalities must work perfectly with every most popular browser such as Internet Explorer, Firefox, Chrome, Opera, Safari. Is also important to consider all the possible restrictions made by the host operative system in which the browser is running such as memory limitations and location access restrictions.
- myTaxiService mobile application must be available on the most used mobile OS (iOS, Android and Windows Phone) on their respective main markets (App Store, Play Store and Windows Store).
- myTaxiService mobile application's size must not exceed 10Mb in order to not discourage users to download it when they are not over Wi-Fi.
- The call center operator must insert all the requests coming from a phone call inside the myTaxiService application, which will manage them properly.

2.4. Assumptions and Dependencies

- In the given documentation is not specified which is the city to consider, so this project works under the assumption that the city has the same size of Milan, that for instance is, approximately, 180km^2.
- It is not specified which is the number of taxi drivers. Considering the number of people lives in Milan area (about 3.1-3.2 Millions), and considering the last analysis of taxi density every 10.000 people made by "Corriere della Sera" in 2006, which is 16, there are approximately 5.000 taxi vehicles.
- It is also not specified which is the frequency of each taxi request. Assuming that each driver can work about 8 hours per day, there are approximately 1600 drivers working at the same time. Considering that each request has and average time of 15-20 minutes to completion, each taxi can manage up to 4 request per hours. In the assumption that the number of free taxi vehicles is never saturated, in the worst case of service congestion, there can be up to 6400 requests per hour to the system, which led about to 110 requests per minutes.
- In the given documentation is not specified which is the number of the call center operators. Considering the assumption that the mean time to complete a call request is 2 minutes and the assumption on the above point, this project will work on the assumption that the number of call center operators is 50.
  This number is calculated considering the worst scenario, in which half of the requests came through via phone call all the same instant, and a user must wait 2 minutes to perform its request (which is a reasonable time).
- In the given documentation is not specified who manage the taxi services, so for simplification this project will assume that all the drivers and call center operators are directly hired from the government and they are considered public employees.
- This project works under the assumption that all the 90 taxi zones, of 2km^2 each, don't overlap and only one queue is assigned to each zone.

- This project works under the assumption that at each taxi can be assigned only one queue.
- In the documentation is not specified which kind of infrastructure is available until now, so this project will assume that the existing infrastructure is the one describe in the section 1.2 called “Actual System”.
- The call center will not be discontinued after the application launch, in order to allow people without an internet connection to access the service. This will also grant to senior citizens, and who don’t possess a smartphone, the access to the taxi system. In order to obtain this, call center operators can directly insert requests inside the application.
- Taxi drivers can obviously pick up passengers even if they are not making a request using the application or even without calling the call center. In order to achieve this, each driver can decline a request, that will be forwarded to the next taxi in the queue.
- Taxi drivers can report exceptional situations using the application, so if they have already accepted a request that they cannot accomplish, it will be forwarded to the next taxi in the queue and the current driver is dismissed from the current request.
- In the given documentation is not specified what the system should do if no taxi is available in a queue. The project will assume that if a request arrives for an area with an empty queue, the request can be forwarded to a taxi in the nearest area’s queue available.
- This project will assume that the government will give a tablet with a monthly-prepayed data plan to each taxi driver, in order to grant internet access to its employees. All the tablets comes with the myTaxyService mobile application pre-installed.
- Each driver must login at the start of his working day and logout at the end.

2.5. Future possible implementations

- Allow passengers to reserve a taxi specifying date, time, origin and destination of the ride.
- Allow passengers to share a taxi in order to pay less for a ride.
- Allow passengers to rate the ride and the service.
- Create a bonus campaign, in which each ride gives a certain amount of credits to a passenger. When the number of collected credits is sufficient, the ride is offered by the government.
- Allow passengers to pay using their telephone with existing technologies (NFC, Apple Pay, Google Wallet).

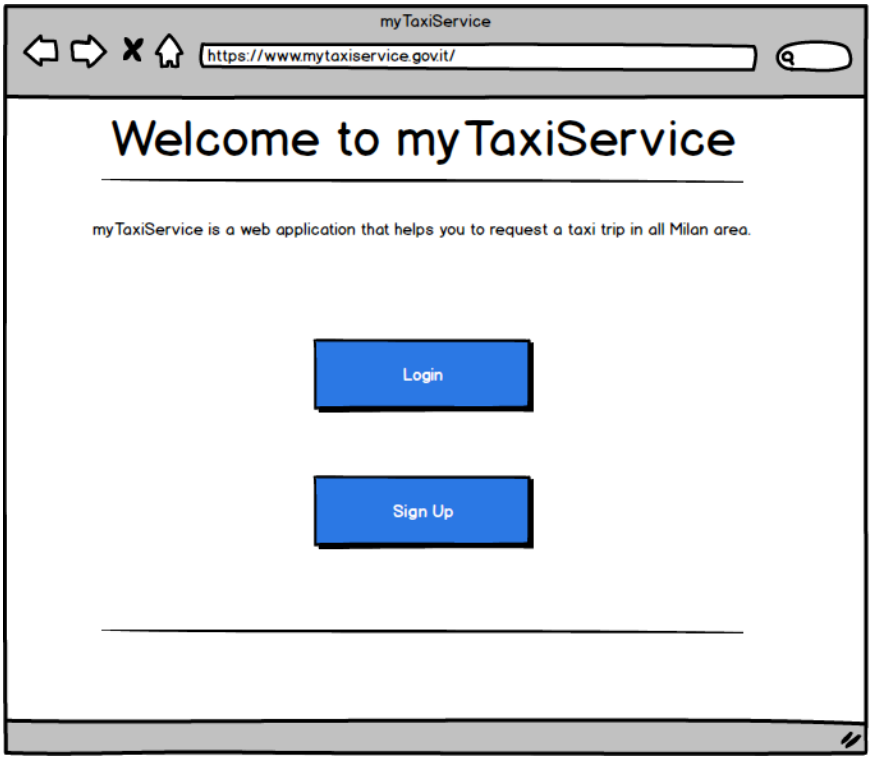
3. Specific Requirements

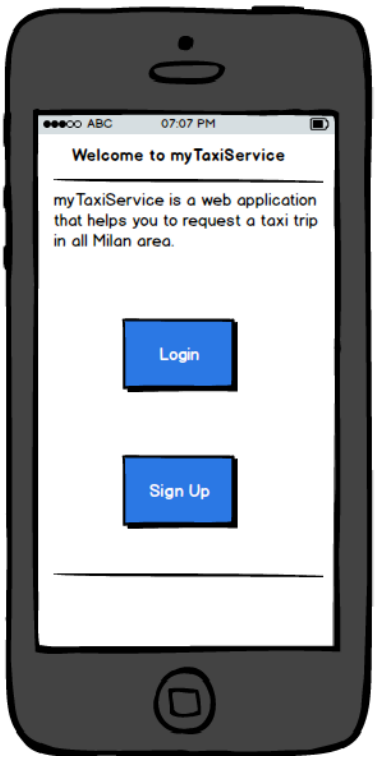
3.1. External Interface Requirements

3.1.1. User Interfaces

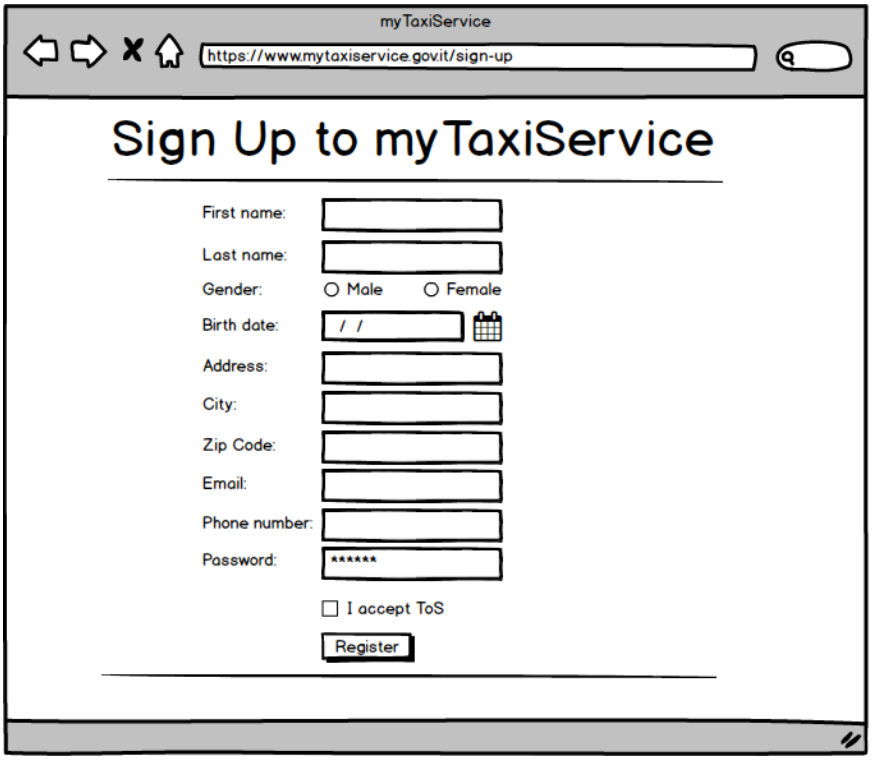
This application can be used via a web browser or a mobile application.

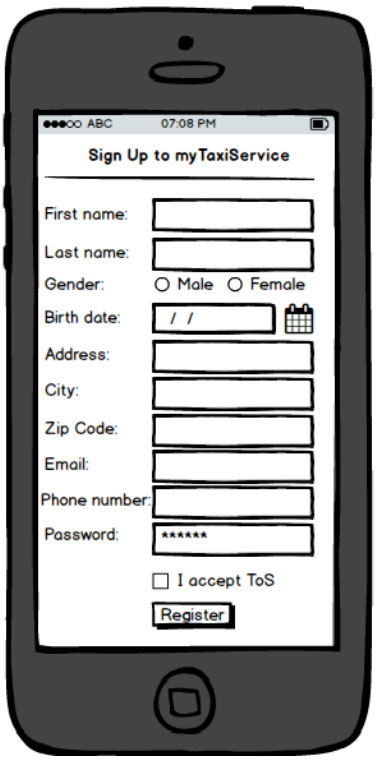
- **Home Page:** this mockup represents the initial page of the application for non-logged users.



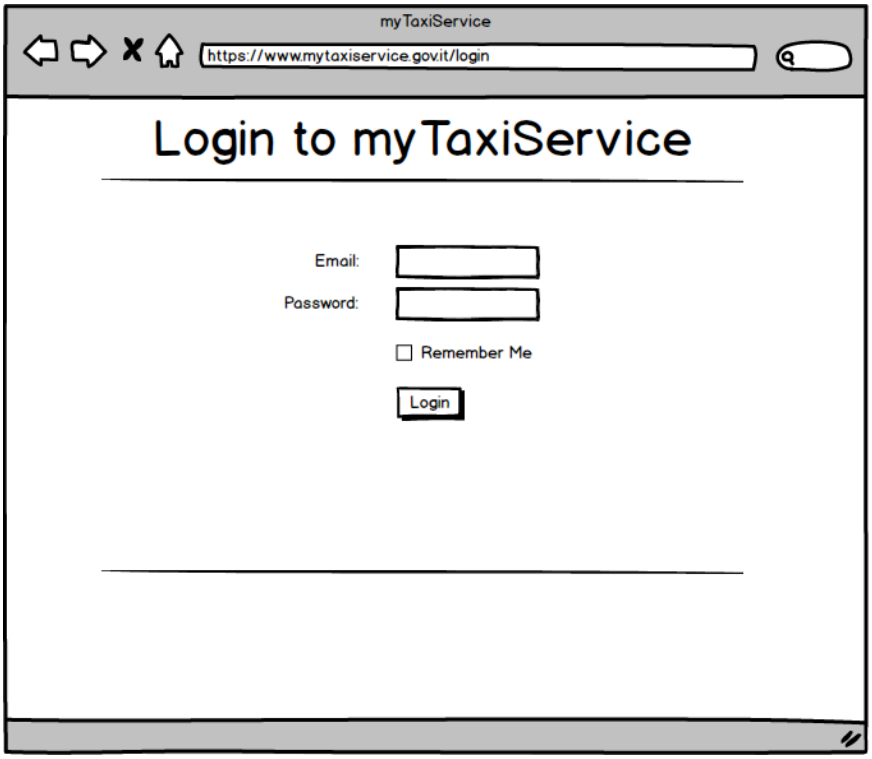


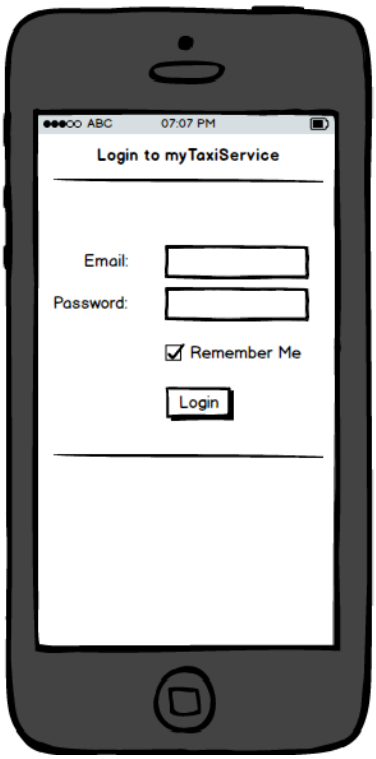
- **Sign Up page:** this mockup represents the form in which a user can sign up into the system.



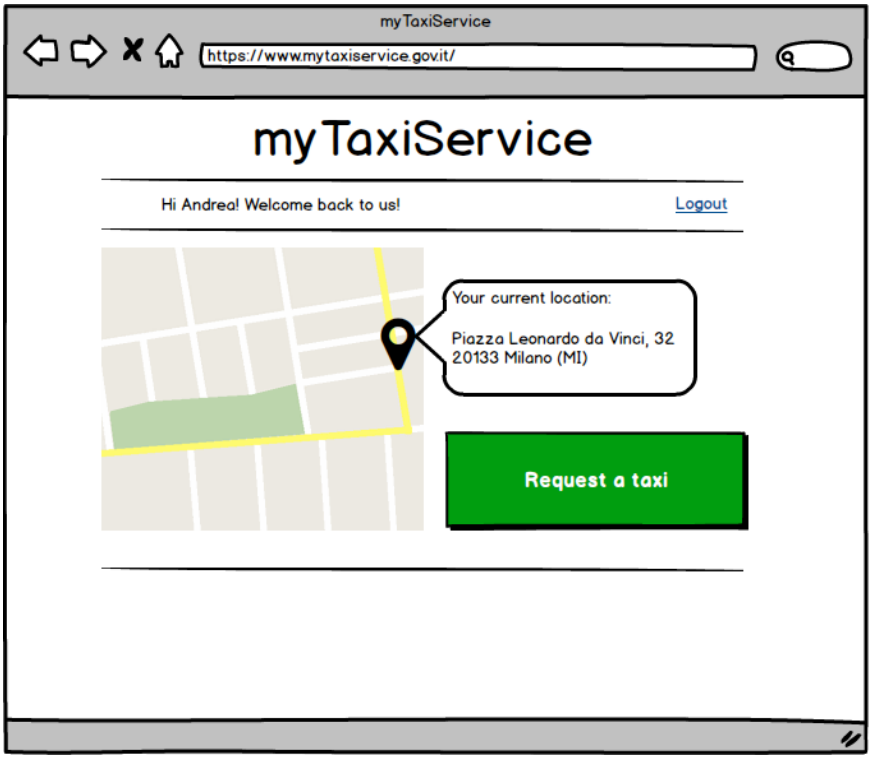


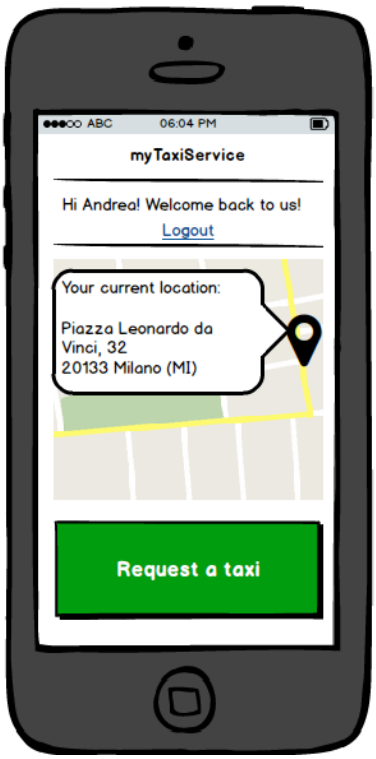
- **Login Page:** this mockup represents the page in which a user can log in to the application.





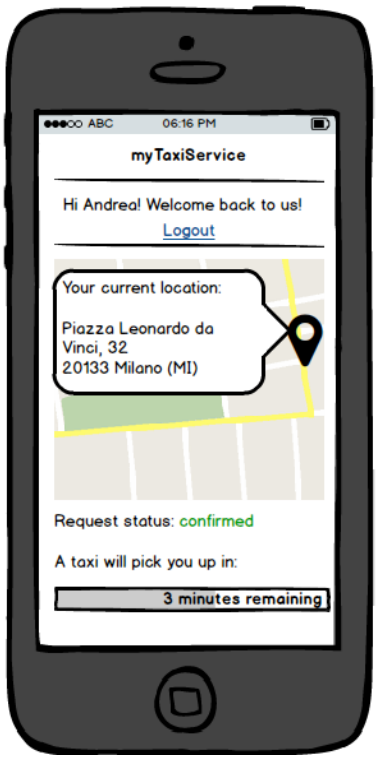
- **Home Page for passengers:** this mockup represents the page that is shown to a passenger after his/her login. Here he/she can make a request for a taxi trip.



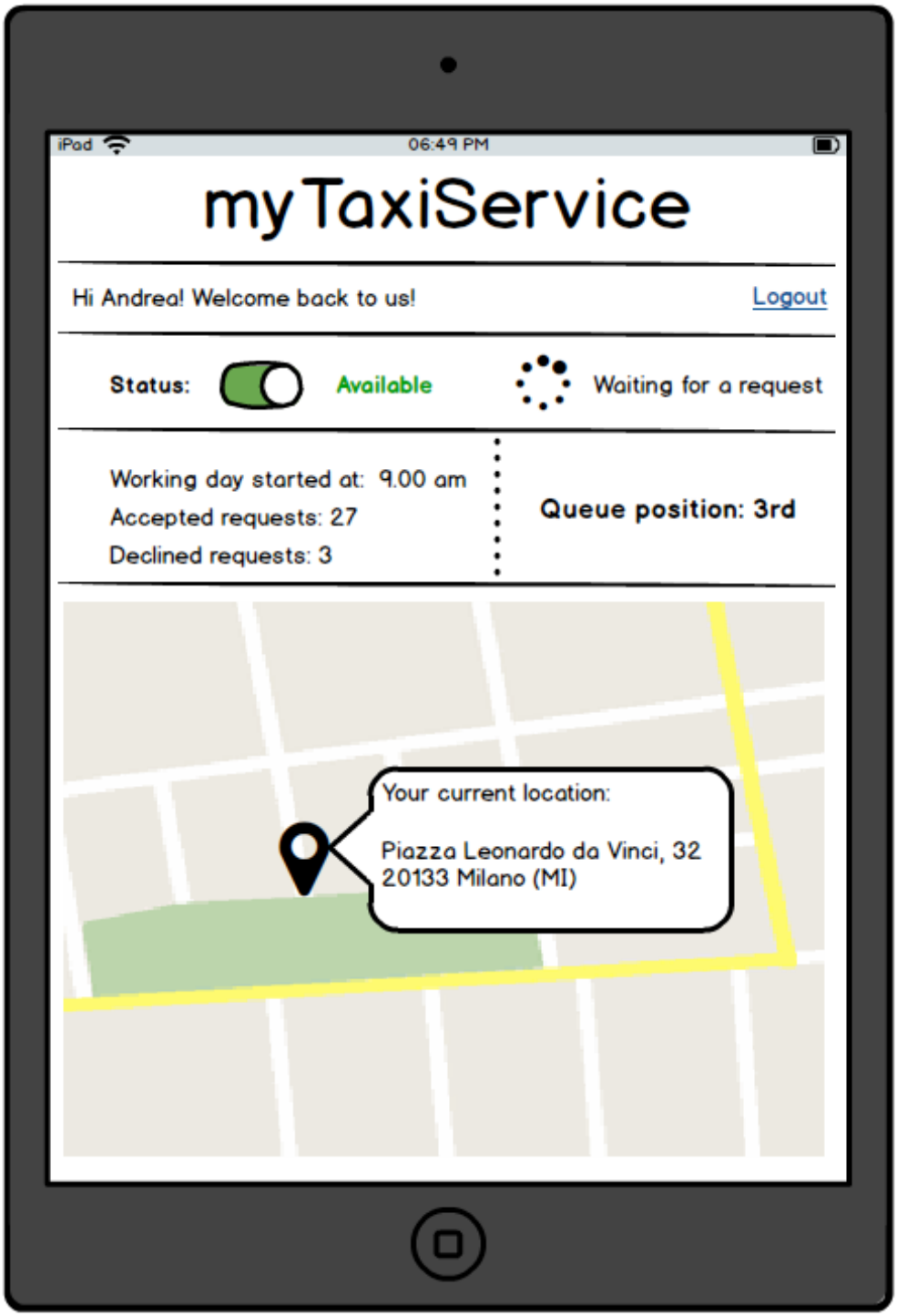


- **Send Request:** this mockup represent the page that is shown to a passenger which is requesting a taxi trip.

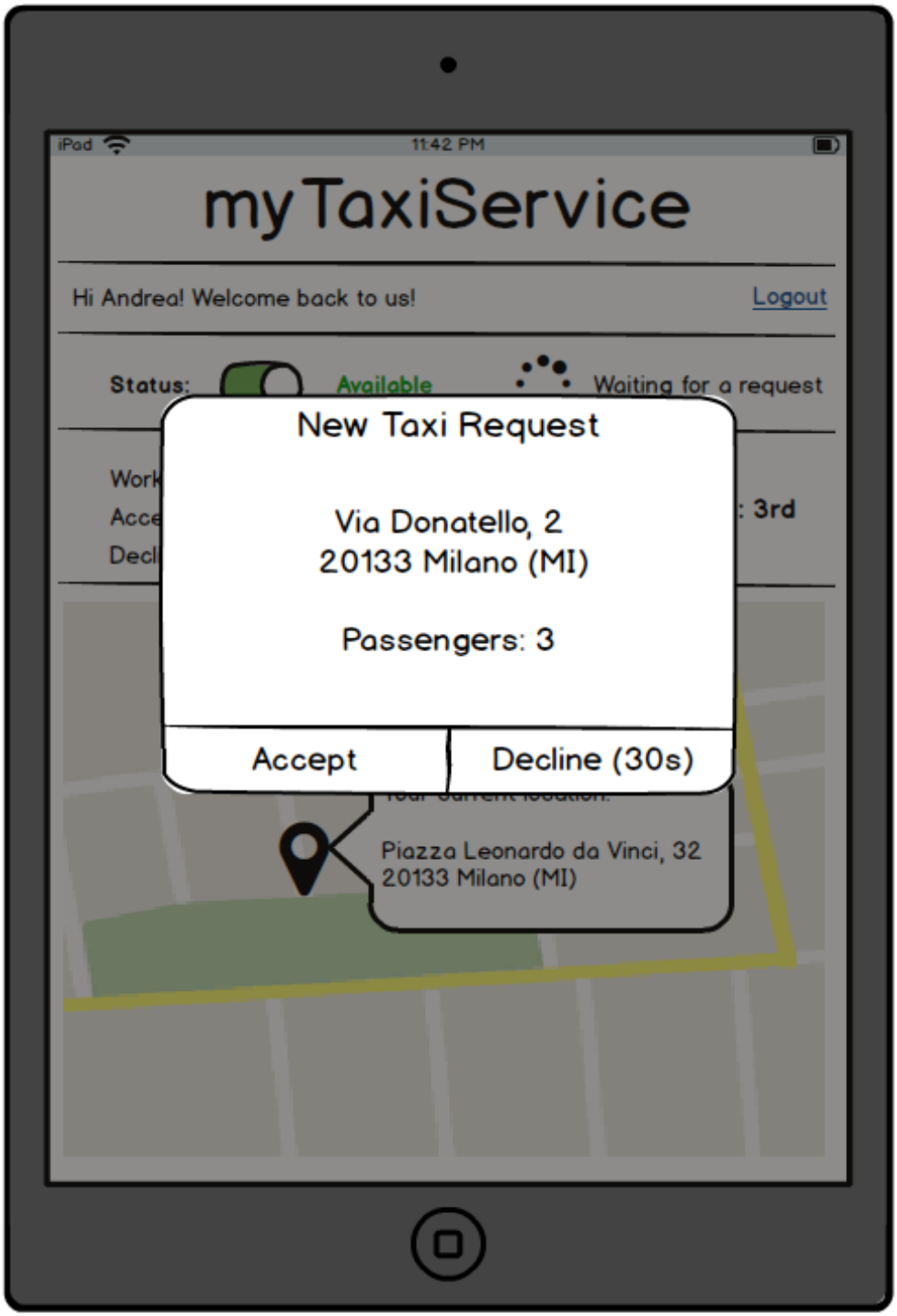
- **Request Status Page:** this mockup represent the page that is shown to a passenger which has sent a request for a taxi trip. Here he/she can see the status of such request.



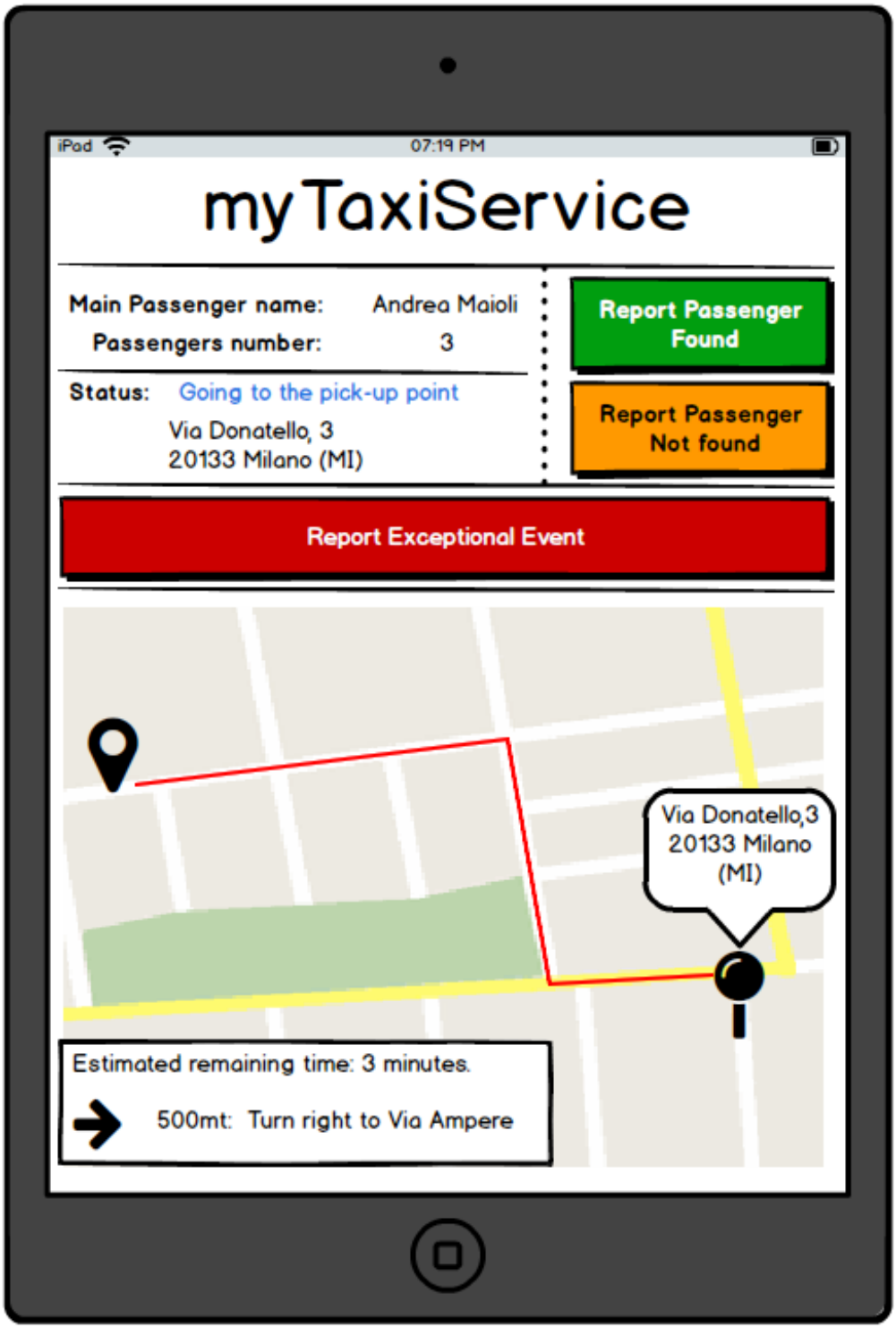
- **Home Page for taxi drivers:** this mockup represents the home page that is shown to a taxi driver. Here he/she can set its status and see some statistics about his/her work day.



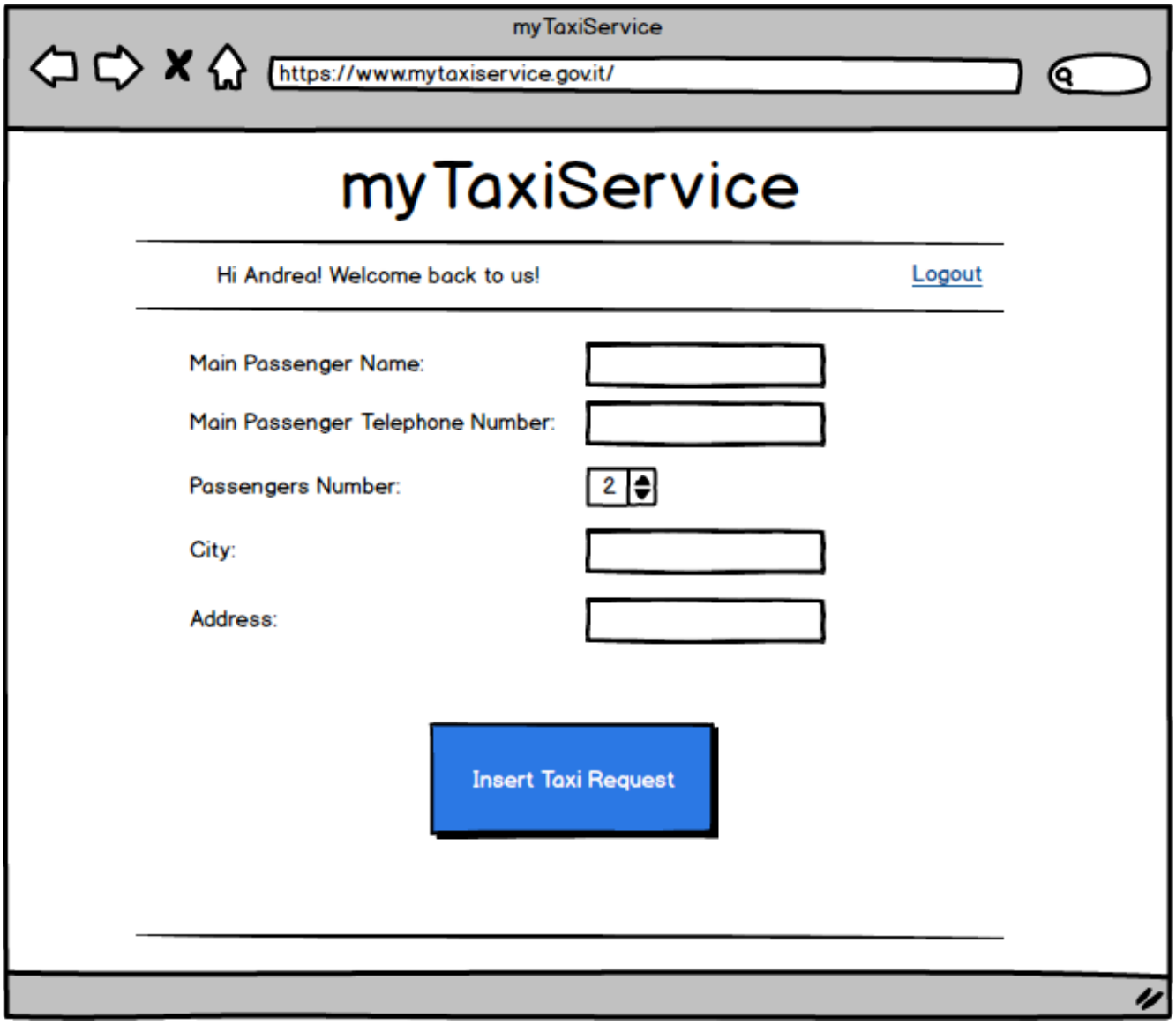
- **Incoming Request:** this mockup represents what is shown to a driver when a request is forwarded to him/her. He/she can accept or decline the request. Is also shown a time in which the request is automatically declined by the application.



- **Request Accepted Page:** this mockup represents the page that is shown to the driver when he/she accept a taxi trip request. Here he/she can see the information of the pick-up point, of the passenger and report some events. Is also shown a map that helps the driver to get to the passenger.



- **Home Page for Call Center Operator:** this mockup represents the page that is shown to the call center operator after his/her login. Here he/she can insert requests coming from a phone call inside the application, allowing it to manage them.



3.1.2. Hardware Interfaces

This project doesn't expect any type of hardware interface.

3.1.3. Software Interfaces

This project needs the following software interfaces:

- Database Management System (DBMS):
  - Name: MySQL Server
  - Version: 5.7
  - Source: http://www.mysql.com/
- Application Server:
  - Name: Glassfish
  - Version: 4.1.1
  - Source: https://glassfish.java.net/
- Web Server:
  - Name: Glassfish
  - Version: 4.1.1
  - Source: https://glassfish.java.net/
- Operative System:
  - Each operative system that can run MySQL and Glassfish without any compatibility issues.

3.1.4. Communication Interfaces

The project needs the following ports for the communication:

- TCP port number 80 for HTTP (used by the Web Server)
- TCP port number 443 for HTTPS (used by the Web Server)
- TCP port number 3306 for the connection with the database (used by MySQL Server)

3.2. Functional Requirements

For each individuated actor, there are reported each action he/she can perform.

- **Guest can**:
  - Sign up

- **Passenger can**:
  - Login both on web application and mobile application
  - Modify his/her profile information
  - Request a taxi
  - Get the status of an active request
  - Get the estimated ETA of an active request
  - Logout from the application

- **Driver can**:
  - Login on the mobile application
  - Set his/her status as Available or Busy
  - See his/her queue position
  - See his/her workday statistics
  - Receive taxi request
  - Accept or Decline a taxi request
  - Report if a passenger is found or not, after getting to the pick-up point
  - Report an exceptional event which prevent him/her to get to the pick-up point
  - Get the estimated arrival time to the pick-up point
  - Get indications on how to get to the pick-up point
  - Logout from the application

- **Call Center Operator can**:
  - Login on the web application
  - Receive calls for taxi requests
  - Insert inside the system a taxi request
  - Get the status of the inserted request
  - Get the estimated ETA of the inserted request
  - Communicate request information to the client through the call
  - Logout from the application

The system is not considered as an actor, but it must be able to:

- Assign a request for an area to the first taxi in the queue of this area.
- Organize drivers in queue.
- Assign exactly one queue for each area.
- Automatically forward a request to the second driver in the queue if the first driver decline it.
- Automatically forward a request to the second driver in the queue if the first driver doesn't accept or decline it within 30 seconds.
- Assign a new driver to an accepted request if the assigned driver report an exceptional events that prevent him/her to get to the passenger.
- If no driver is available in a queue, the system must be able to find the driver that will arrive to a fixed pick-up point in the less possible time. The driver can be one from another queue, or one that will finish a ride in the request's area.

3.3. Performance Requirements

a. Each passenger's request must be confirmed in no more than 2 minutes, and his/her taxi must arrive in no more than 10 minutes, if any available.
b. Each request must be accepted or declined from a driver in less than 30 seconds. After this period, the request must be automatically declined and assigned to the next taxi in the queue, and the driver must be moved in the last position of the queue.
c. The system must be able to compute the response to each request in no more than 500ms.
d. The system must be able to serve simultaneously 1600 taxi drivers, 50 call center operators and 110 passengers with a response time for each action lower than 3 seconds.
e. Given the analysis made in the assumptions, the total number of taxi requests per minutes can be up to 110. The application must be able to handle 200 taxi requests per minutes.

3.4. Software System Attributes

3.4.1. Availability

The application must be accessible 24h per day and 7 days per week and must be always possible to use the service. All the software updates to the main infrastructure must be applied during the night, when the requests are minimum, and the call center must be available.

3.4.2. Security

- User's credential must be stored using a hashing function.
- Users must be able to login only by entering the correct credentials.
- User's password must be at least 8 characters long and a composition of letters and numbers.
- The application server must be the only machine that can communicate with the DBMS.
- All the user inputs must be filtered in order to prevent SQL-Injection, Cross Site Scripting and other type of attacks.
- HTTPS must be enable on the web server and all the connection coming from the HTTP protocol must be redirected to the HTTPS one.

3.4.3. Maintainability

- Each function and class of the application must be well commented in order to allow future developers to understand and modify the code.
- Each update or modification to the application or the infrastructure must be traceable and well documented.
- Each solution to every future problem must be well documented, in order to know how it has solved.
- Each method provided from the API will be documented and some example will be provided.

3.5. Scenarios

Scenario #1: Taxi request using mobile application.
Andrea needs a taxi to return his home, so he open the myTaxiService application on his smartphone and login. After that, he see his current location and click the “Request a taxi” button. The application ask him to insert the number of passengers for this ride, and since he is with her girlfriend, he select two and confirm the request. Now the application shows him that the request is in the processing phase and after one minutes he sees that the request status is “confirmed” and a counter says that a taxi will pick they up in no more than 5 minutes. The taxi arrives in only 3 minutes and brings them home.

Scenario #2: Taxi request through a telephone call.
Giovanni needs a taxi to go to his girlfriend’s home and opens the myTaxiService application. Unfortunately, there is no internet connection available in the current location of Giovanni, but the application shows him the call center number. He decides to call and in less than one minute an operator answer to the phone, which asks Giovanni his name and surname in order to check if he is registered in the system.
Unfortunately Giovanni never registered to myTaxiService, so the call center operator asks him his details and tells Giovanni his new password. Now the call center operator asks his current location and the number of passengers. After getting these informations, in less than 30 seconds the operator tells Giovanni that his taxi will come to pick him up in no more than 3 minutes. The taxi arrives and brings him to his girlfriend’s home.

Scenario #3: Taxi request accepted.
Luca is a driver, and he is waiting in a parking lot for a ride. In his tablet there is the myTaxiService application running, showing that he is the first of the queue. In less than one minutes he hear a “Bling” sound and the application shows Luca that there is an incoming request for a ride, showing also all the details. He accept the request and now on his screen appears a map showing indications on how to get to the passengers.
When Luca arrives to the pick-up point, the passengers enter his car and he click on the “Report Passenger Found” button and sets the drop-off point location. When he arrives to the drop off point, the clients pay him and Luca click on the “Ride Ended” button. Now the application changes page and shows his new queue position.

Scenario #4: Taxi request declined.
Luca is a driver, and he is going to a gas station to refill his car. In his tablet there is the myTaxiService application running, showing that he is the first of the queue and a request arrives. Luca has to decline the request and now the application shows that he is in the position 16 of the queue.

Scenario #5: Taxi request automatically declined.
Luca is a driver, and he is at the gas station refilling his car. In his tablet there is the myTaxiService application running and he forgot to set the “Busy” status on it. The application forward a request to him, but Luca is unable to hear the notification. After 30 seconds the request is automatically declined and Luca is moved on the last position of the Queue.

Scenario #6: Taxi request accepted but driver not able to complete it.
Luca is a driver, and he is waiting in a parking lot for a ride. In his tablet there is the myTaxiService application running, showing that he is the first of the queue. In less than one minutes he hear a “Bling” sound and the application shows Luca that there is an incoming request for a ride, showing also all the details. He accept the request and now on his screen appears a map showing indications on how to get to the passengers.
Unfortunately his car stops due to a flattered tire and Luca tap the “Report exceptional event” button.
He select “Car Problem” and send the report. The application automatically sets his status to “busy” and the system forward the request to another driver.

Scenario #7: Taxi request accepted but no passenger found.
Luca is a driver, and he is waiting in a parking lot for a ride. In his tablet there is the myTaxiService application running, showing that he is the first of the queue. In less than one minutes he hear a “Bling” sound and the application shows Luca that there is an incoming request for a ride, showing also all the details. He accept the request and now on his screen appears a map showing indications on how to get to the passengers.
When Luca arrives to the pick-up point, there is anyone and he clicks on the “Report passenger not found”. The system send to the passenger a message and the application shows Luca a counter of 5 minutes. At the end of this timer, Luca press the “No passenger” button and the system reassign him to the queue of free drivers in the area.

Scenario #8: No taxi available in a queue.
Giovanni want to go home, so he requests a taxi using the myTaxiService mobile application.
Unfortunately in this area there is no taxi available, so the system search for all the busy taxies that will end the ride in the request’s area and compute the estimated time of the end of the request, considering the taxi of John, that has the less timing of arrival. The system also compute the ETA for the arrival of the taxi that is in the first position of the nearest area queue and compare with the ETA of John.
John will end the request in less than one minutes, so the system forward the request to him.

3.6. Use Cases

Considering the documentation provided until now, is it possible describe in a detailed way the use cases.
All the references to the pages and theirs components are made in order to have a better understanding of the described situation. The real structure of the application is not defined in this documentation and will be defined in the Design Document.

Name	Passenger Sign Up
Actors	Guest
Entry Conditions	The guest isn't registered to the application and want to sign up
Flow of Events	1. The guest enters the website or the mobile application. 2. The guest clicks on the "Sign Up" button in the initial page. 3. The guest compile the given form with his/her informations: • First Name • Last Name • Email • Password • Telephone Number • Date of Birth • Address • City • Zip Code • Gender 4. The guest accept the terms of service (ToS) 5. The guest send the form clicking on the "Register" button
Exit Conditions	The registration ended successfully
Possible Exceptions	• Email already used by another user • Phone number already used by another user • Terms of service not accepted • User already registered

Name	Login
Actors	Passenger or Call Center Operator
Entry Conditions	The user has previously signed up to the system
Flow of Events	1. The user enters in the website or mobile application. 2. The user click on the "Login" button in the initial page. 3. The user enters his/her email and password into the login form. 4. The user click on the "Login" button.
Exit Conditions	The system identify the user as a Call Center Operator or Passenger and display the relative page.
Possible Exceptions	• Email and/or password wrong

Name	Taxi Request Using Application
Actors	Passenger, Driver
Entry Conditions	The passenger is logged into the system and want to request a taxi for a ride
Flow of Events	1. The user clicks on the "Request a taxi" button. 2. The user enters the number of passengers for this ride. 3. The application sends to the system the current location of the user. 4. The user send the request for a ride. 5. The system assign the request to the first driver in the area's queue and move him/her to the last position of the queue. 6. If the driver doesn't accept the request, restart from 4 until the request is confirmed from a driver. 7. The status of the driver who accepted the request is removed from the queue.
Exit Conditions	A taxi is assigned to the passenger and the application shows the ETA of the taxi arrival.
Possible Exceptions	• None

Name	Taxi Request through a call
Actors	Passenger, Call Center Operator, Driver
Entry Conditions	The passenger call the call center for requesting a taxi.
Flow of Events	1. A Call Center Operator answer the phone 2. The call center operator asks information about the user and verify if he/she is registered to the system. If not, the call center operator registers the user with his/her details and tells him/her the new temporary password, that is automatically generated by the system. 3. The call center operator asks the number of passengers and the location. 4. The passenger tells the number of passengers and the location. 5. The Call Center Operator insert the request inside the system using the web application. 6. The system assign the request to the first driver in the queue and move him/her to the last position of the queue. 7. If the driver doesn't accept the request, restart from 7 until is confirmed from a driver. 8. The driver who accepted the request is removed from the queue.
Exit Conditions	The call center operator confirm the request made from the passenger and gives him/her the ETA of the taxi arrival.
Possible Exceptions	• Call is interrupted due to a connection error.

Name	Edit Password
Actors	Passenger, Call Center Operator, Driver
Entry Conditions	The user want to edit his/her password.
Flow of Events	1. The user click on the “Edit Password” link in the web application or mobile applicatoin. 2. The user enter the old password and the new password. 3. The user click on the “Confirm” button 4. The system change the user password
Exit Conditions	The user is automatically logged out from the system and now have to login again with his/her new credentials.
Possible Exceptions	• Wrong password • Weak Password

Name	Password Reset
Actors	Passenger, Call Center Operator, Driver
Entry Conditions	The user has forgotten the password.
Flow of Events	1. The user click on the “Password Reset” link in the web application or mobile application. 2. The user insert his/her email address and confirm the request. 3. The system send a code to the email of the user. 4. The user insert the received code in the page the application is showing right now. 5. The application prompt a request for a new password. 6. The user insert a new password and confirm. 7. The system change the user password.
Exit Conditions	a. The user has a new password and now can login. OR b. The email does not exists inside the system.
Possible Exceptions	• Wrong code • Weak password

Name	Incoming Taxi Request
Actors	Driver, Passenger
Entry Conditions	The driver has the application open on his/her tablet and his/her status is set to ‘Available’.
Flow of Events	1. A request for a ride is displayed on the tablet.
Exit Conditions	a. The driver Accept the request OR b. The driver Decline the request
Possible Exceptions	• 30 seconds are run and the request is automatically declined

Name	Accepted incoming taxi request
Actors	Driver
Entry Conditions	The driver has accepted an incoming request.
Flow of Events	1. The system remove the driver from the queue. 2. The application display to the driver the location of the passenger. 3. The driver start the ride to the pick-up point.
Exit Conditions	a. The driver arrives to the pick-up point. OR b. The driver has an inconvenient and click on the “Report Exceptional Event” button.
Possible Exceptions	• None

Name	Declined incoming taxi request
Actors	Driver
Entry Conditions	The driver has declined an incoming request.
Flow of Events	1. The system move the driver to the last position of the queue.
Exit Conditions	The system reassign the request to the driver that now is on the first position of the queue.
Possible Exceptions	• None

Name	Passenger found for a request
Actors	Driver, Passenger
Entry Conditions	The driver has arrived at the pick-up point and the passenger is present.
Flow of Events	1. The driver click on the “Report passengers found” button. 2. The driver communicates to the system the drop-off point. 3. The driver ride the passengers to the drop-off point. 4. The driver click on the “Ride Ended” button.
Exit Conditions	The system insert the driver in the last position of the queue.
Possible Exceptions	• None

Name	Passenger not found for a request
Actors	Driver, Passenger
Entry Conditions	The driver has arrived at the pick-up point and the passenger is not present.
Flow of Events	1. The driver click on “Report passengers not found” button. 2. The system sends a text message to the passenger. 3. The driver wait 5 minutes for the passenger.
Exit Conditions	a. The passenger is found and the new situation is described from the Use Case “Passenger Found”. OR b. The passenger is not found, the request is deleted from the system and the driver is inserted in the last position of the queue.
Possible Exceptions	• None

Name	Driver Login
Actors	Driver
Entry Conditions	The driver is starting his/her working day.
Flow of Events	1. The driver open the myTaxiService application on the tablet. 2. The driver click on the “Login” button displayed on the page. 3. The system insert the driver at the bottom of the queue in his/her area.
Exit Conditions	The driver is inside a queue and is considered from the new incoming requests.
Possible Exceptions	• Email and/or password wrong

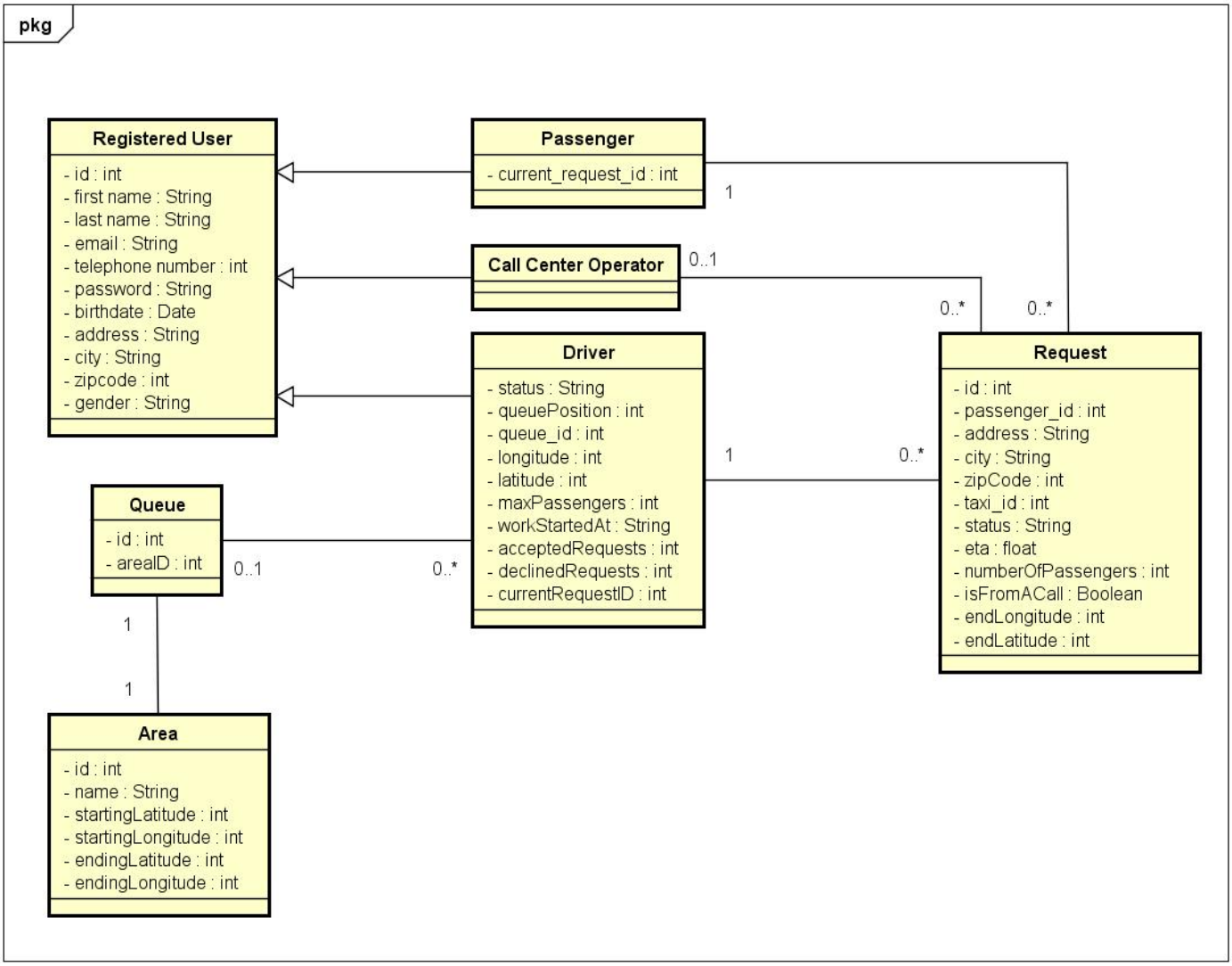
Name	Driver Logout
Actors	Driver
Entry Conditions	The driver has finished his/her working day
Flow of Events	1. The driver click the “Logout” button on the application opened on the tablet. 2. The system remove the driver from the current queue. 3. The driver close the application on the tablet.
Exit Conditions	The driver is no more inside a queue, is not considered anymore from the new incoming requests and has successfully logout from the system.
Possible Exceptions	• None

Name	Driver set status busy
Actors	Driver
Entry Conditions	The driver status is “Available” and want to set his/her status to “Busy”.
Flow of Events	1. The driver click on the status button in the application. 2. The system set the driver status on “Busy” and show it on the page. 3. The system remove the driver from the queue.
Exit Conditions	The driver is no more inside a queue and is not considered anymore from the new incoming requests. Now his/her status is “Busy”
Possible Exceptions	• None

Name	Driver set status available
Actors	Driver
Entry Conditions	The driver status is “Busy” and want to set his/her status to “Available”.
Flow of Events	1. The driver click on the status button in the application. 2. The system set the driver status on “Available” and show it on the page. 3. The system insert the driver at the bottom of the queue in his/her area.
Exit Conditions	The driver is now inside a queue and is considered from the new incoming requests. Now his/her status is “Available”.
Possible Exceptions	• None

3.7. Class Diagram

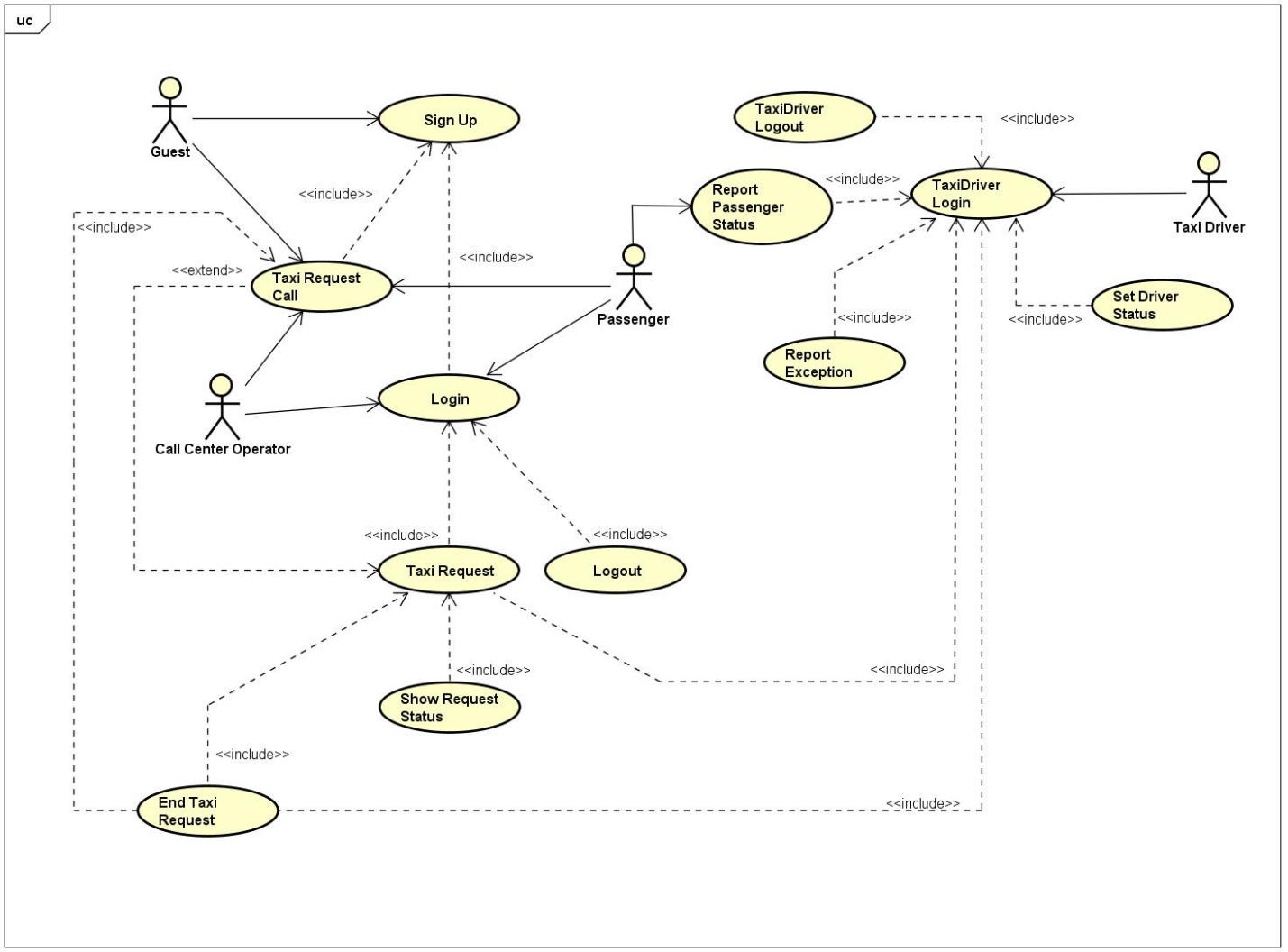
This is the class diagram of the system:



For a better viewing, the Class Diagram can be also consulted from the folder "Class Diagram" inside the github repository.

3.8. UML

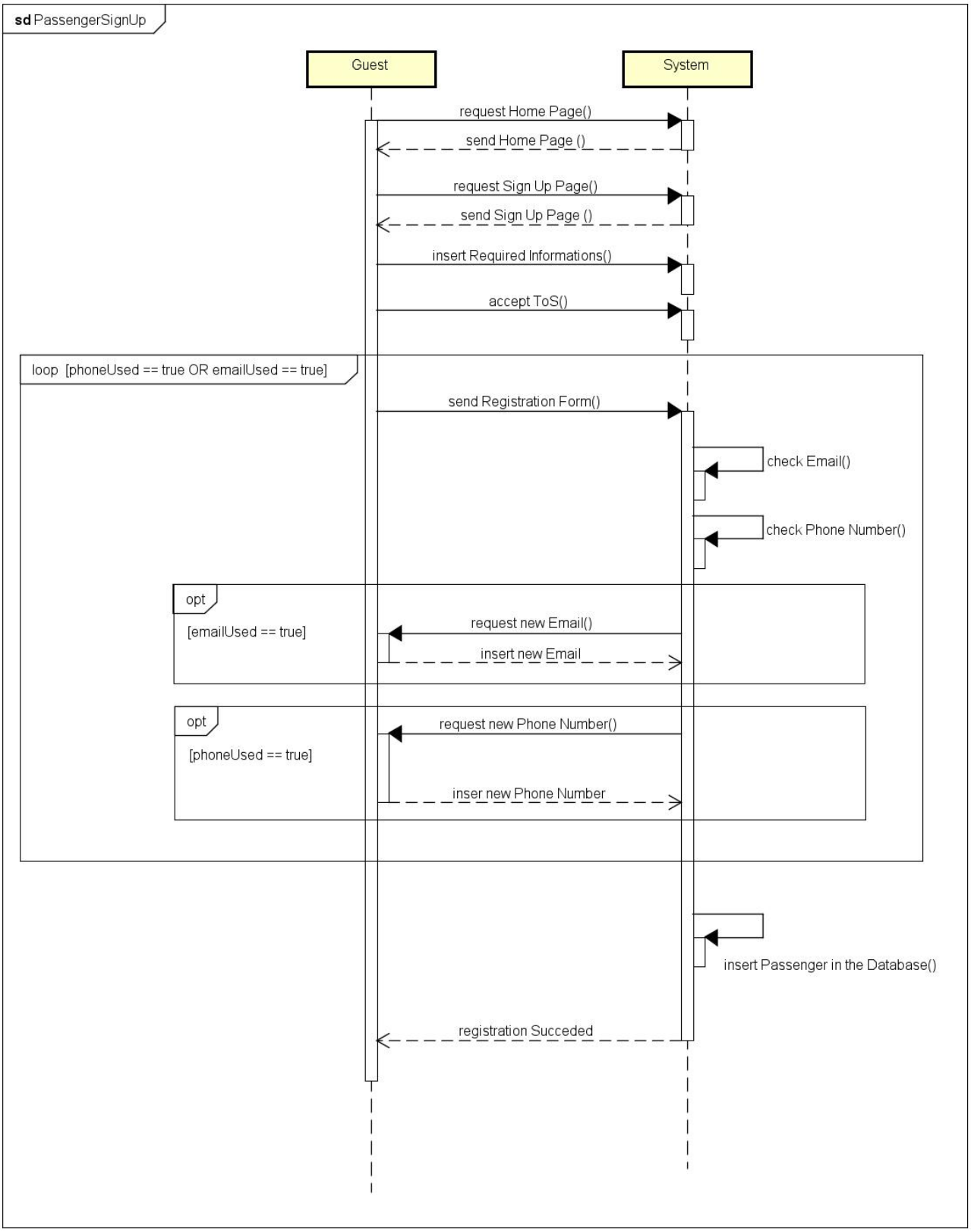
This is the representation of the system, using a Use Case Diagram



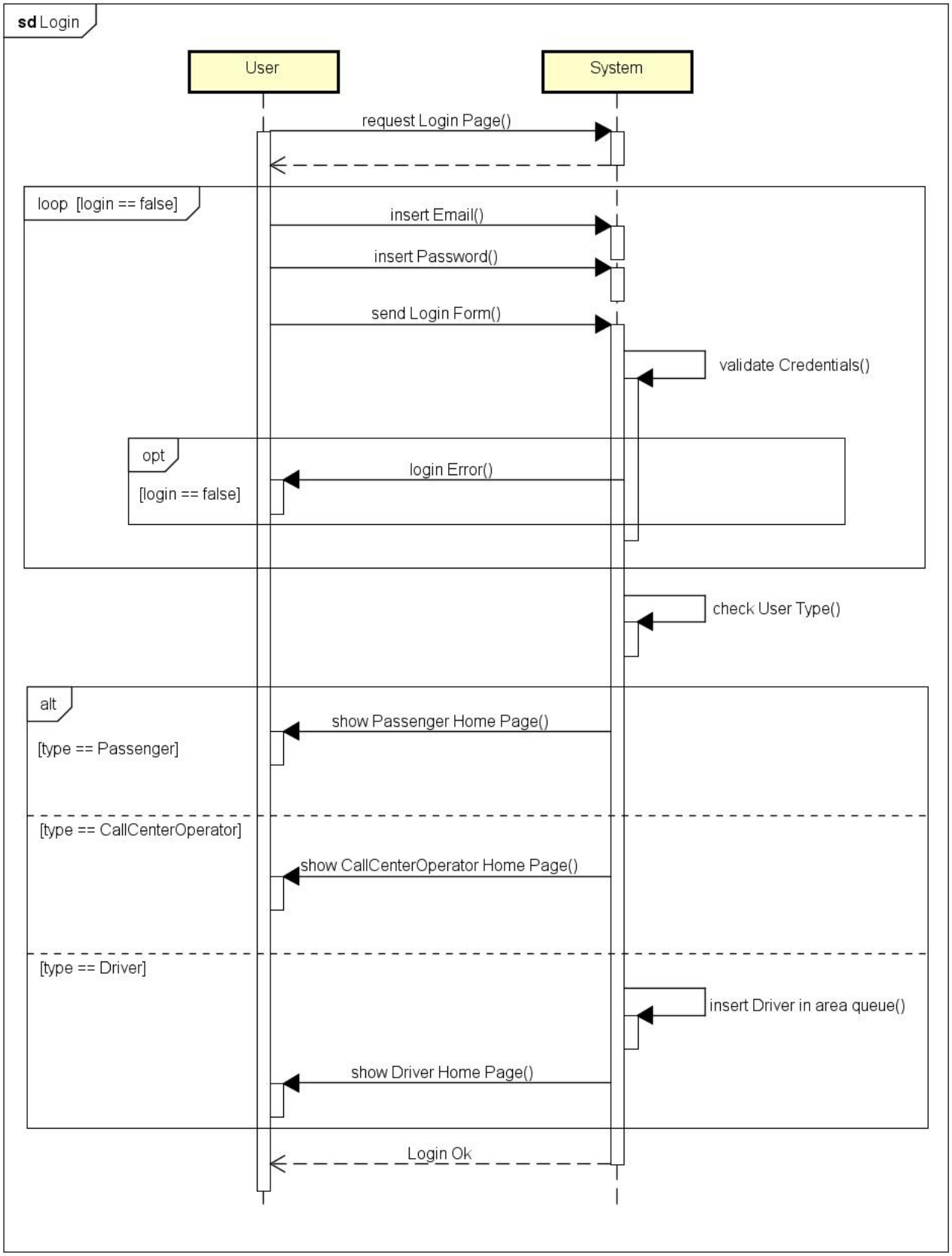
For a better viewing, the Use Case Diagram can be also consulted from the folder "Use Case" inside the github repository.

3.9. Sequence Diagrams

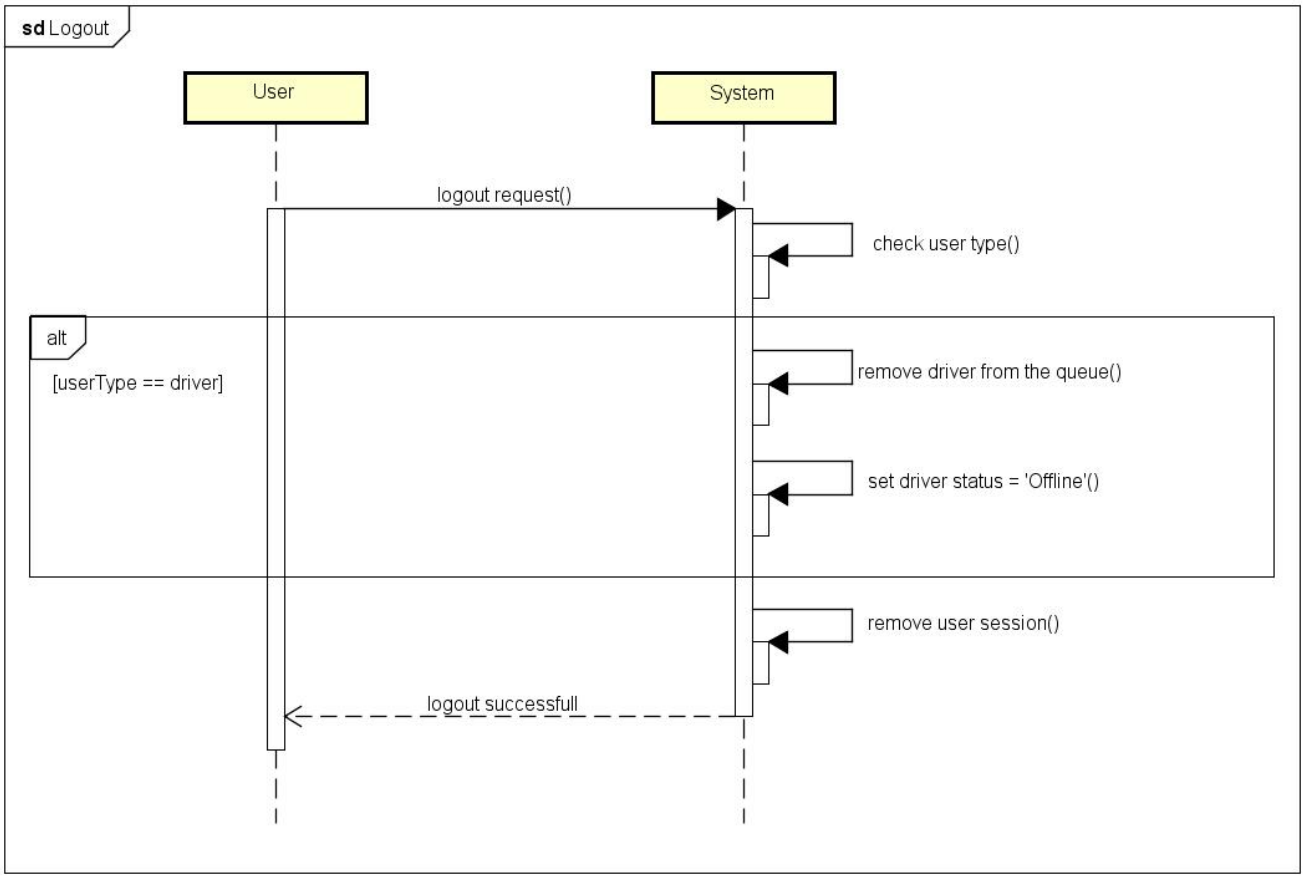
- **Passenger Sign Up**



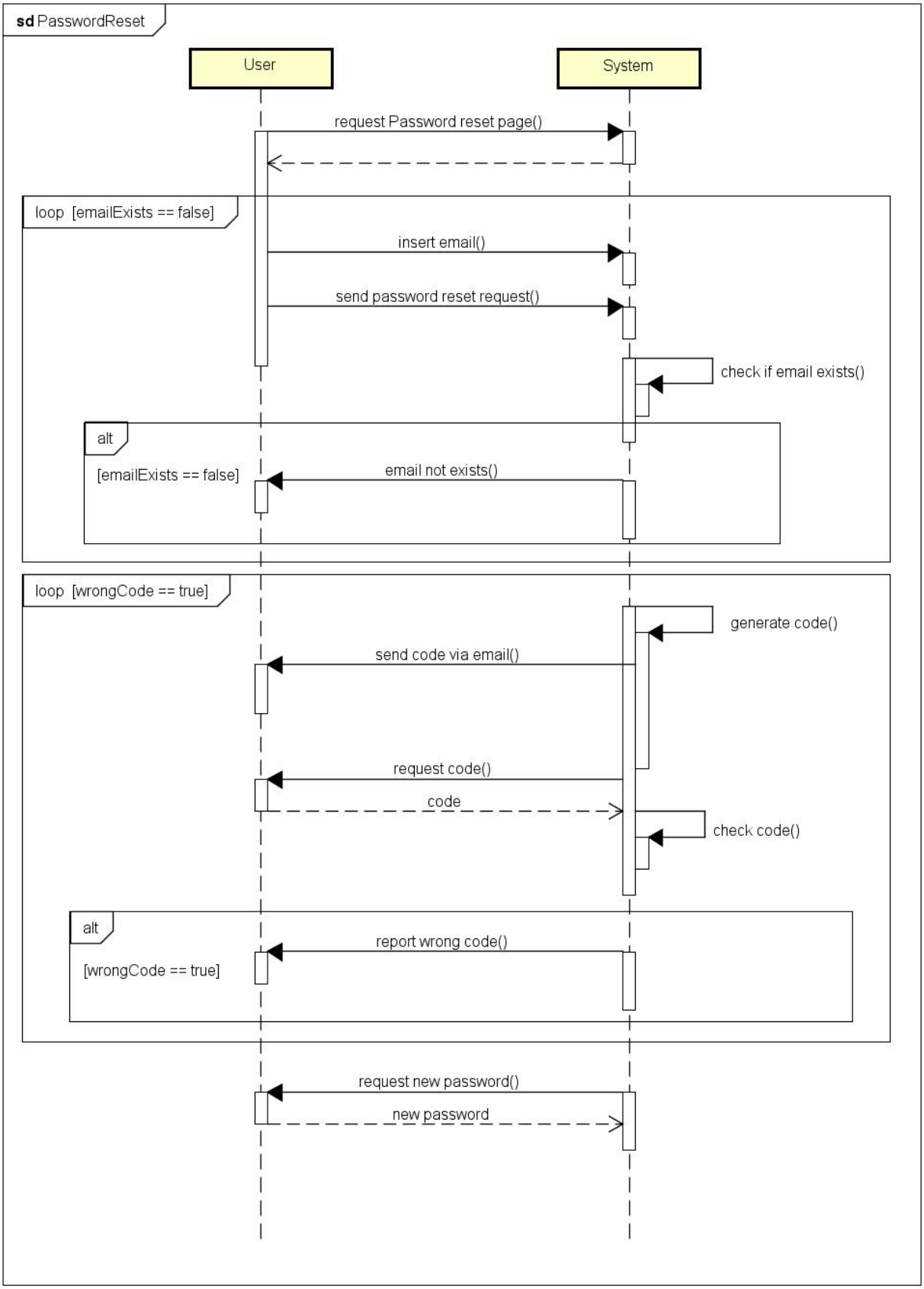
- **Login**


sd Login
User
System
request Login Page()
loop [login == false]
insert Email()
insert Password()
send Login Form()
validate Credentials()
opt
[login == false]
login Error()
check User Type()
alt
[type == Passenger]
show Passenger Home Page()
[type == CallCenterOperator]
show CallCenterOperator Home Page()
[type == Driver]
insert Driver in area queue()
show Driver Home Page()
Login Ok

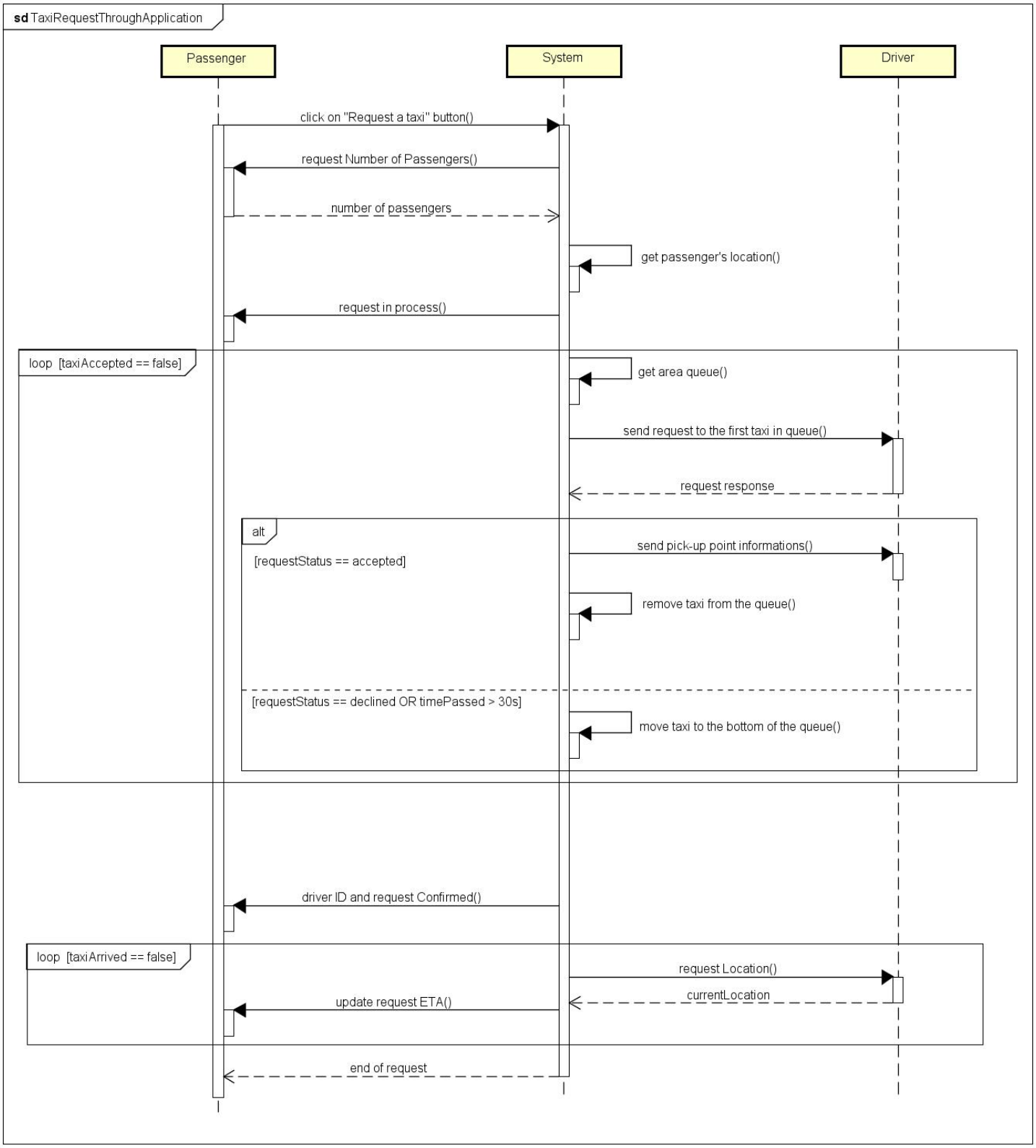
- **Logout**


sd Logout
User
System
logout request()
check user type()
alt
[userType == driver]
remove driver from the queue()
set driver status = 'Offline'()
remove user session()
logout successfull

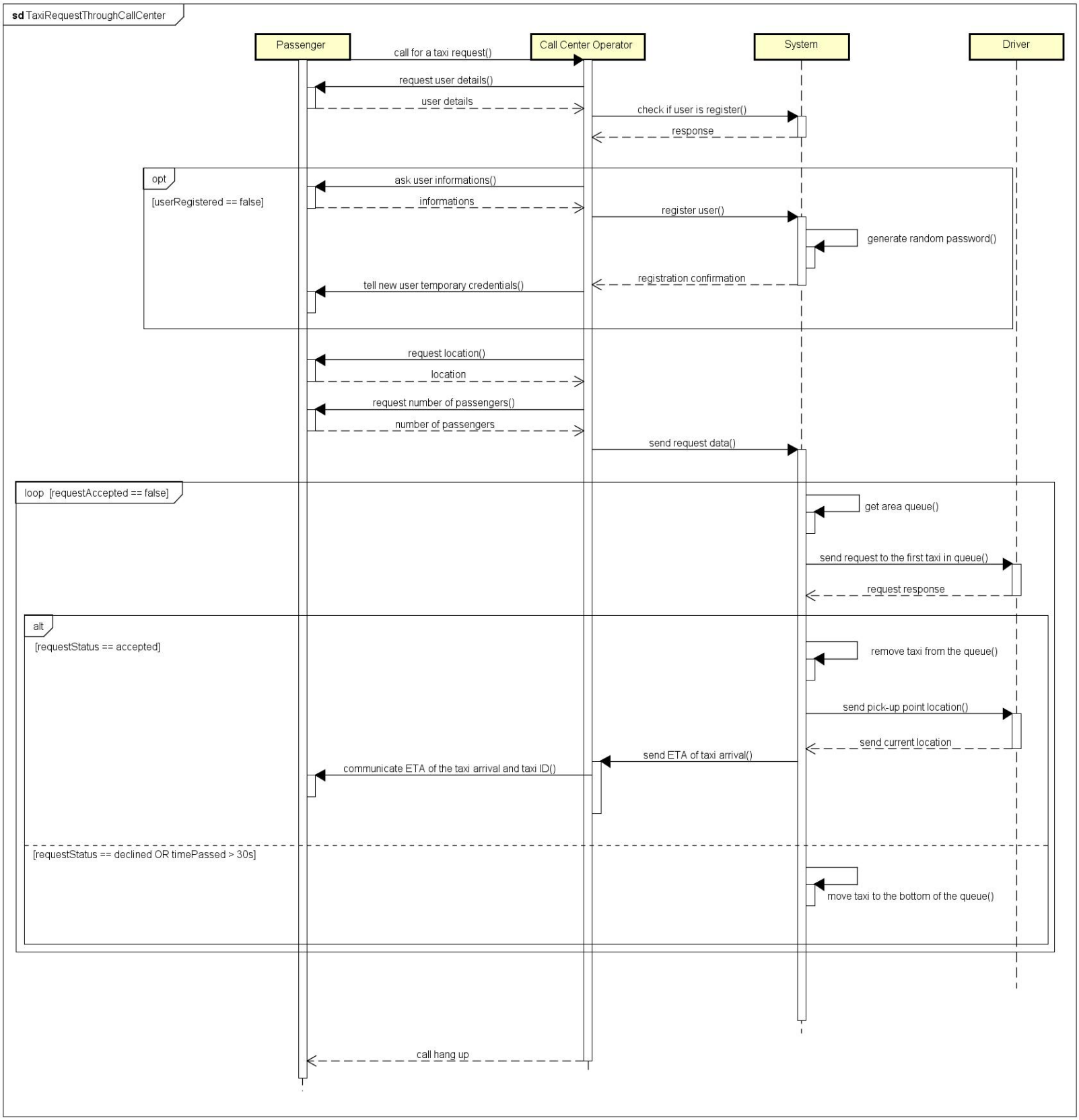
- **Password Reset**



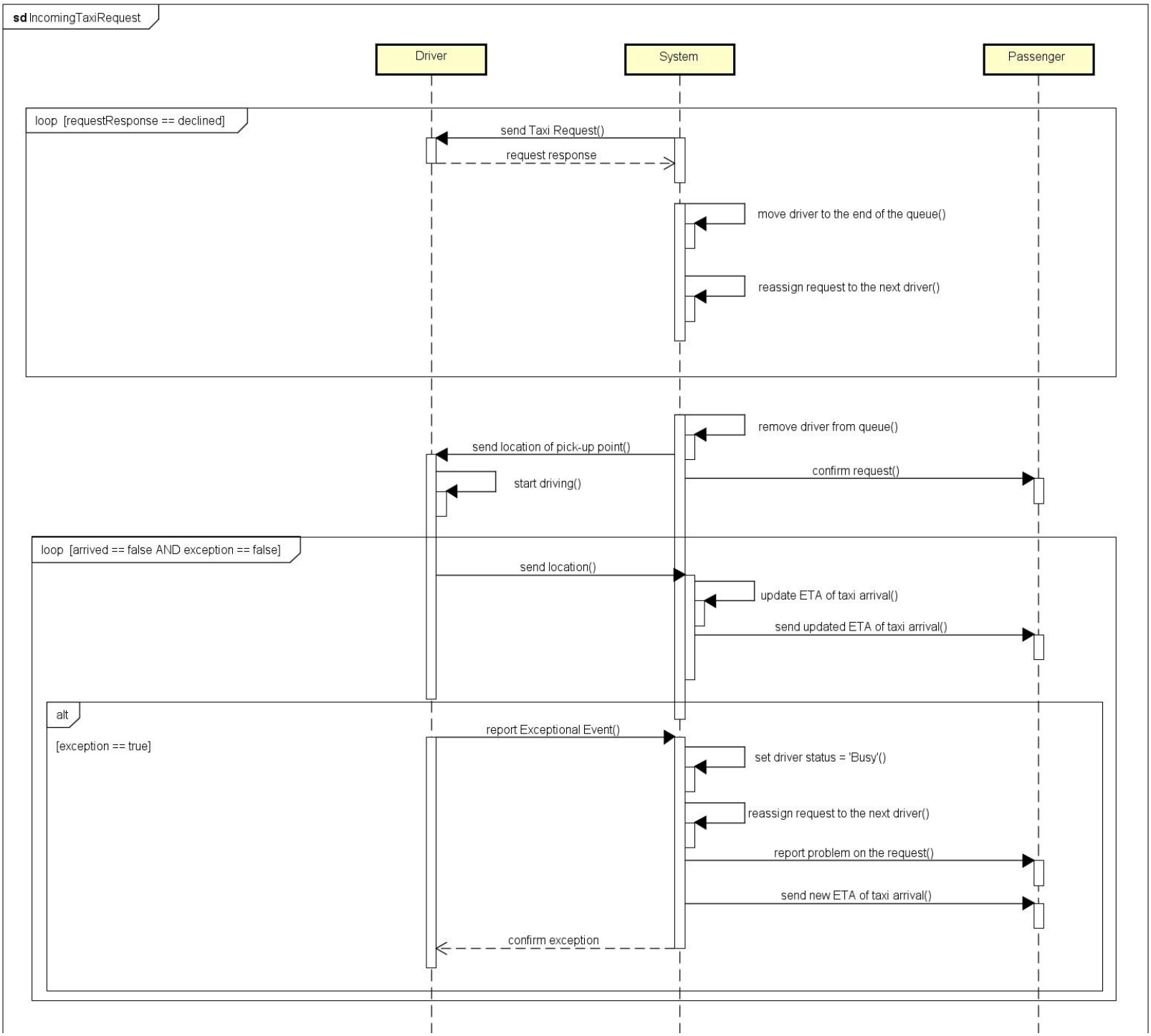
- **Taxi Request Through Application**

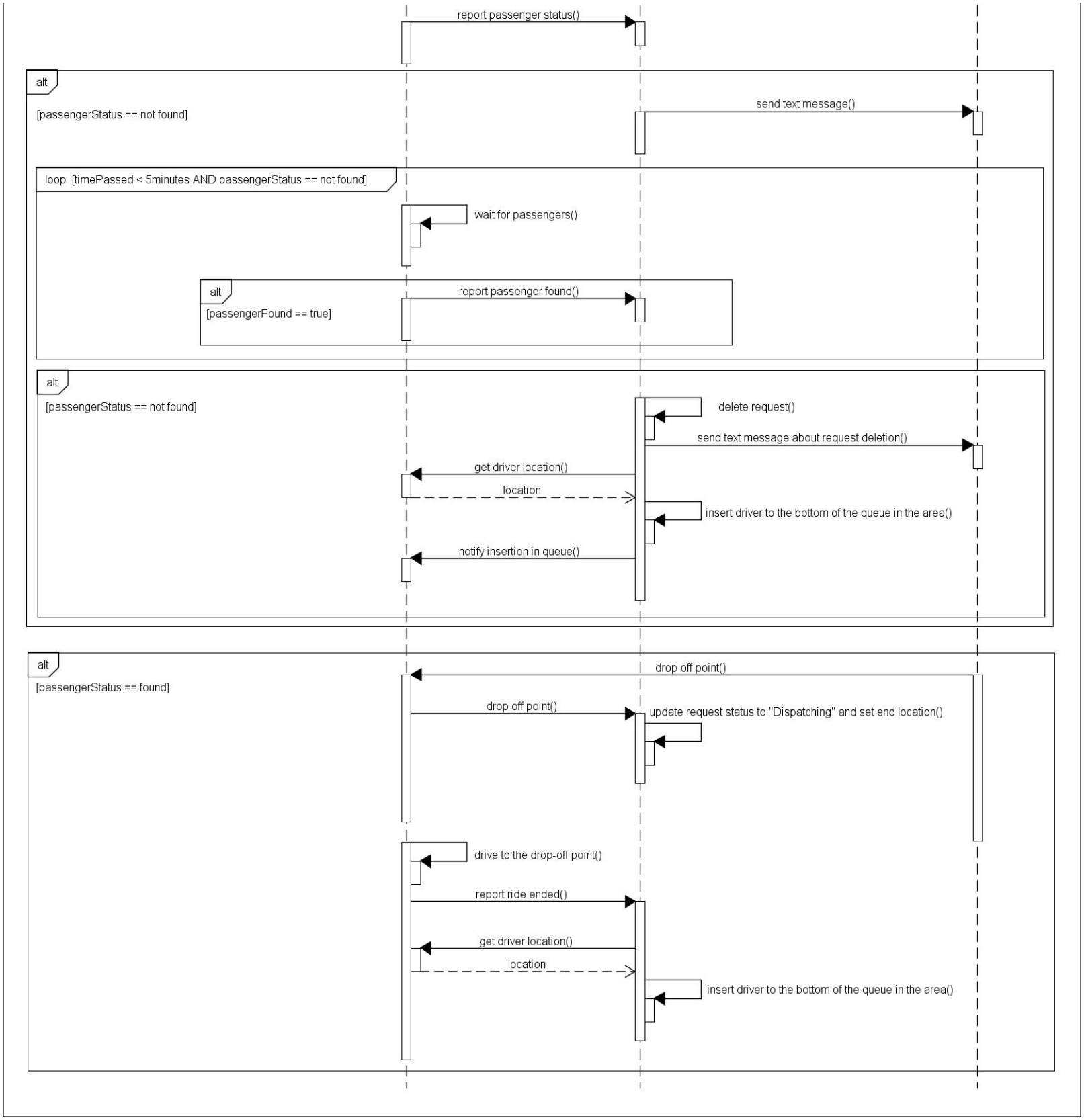


- **Taxi Request Through Call Center**

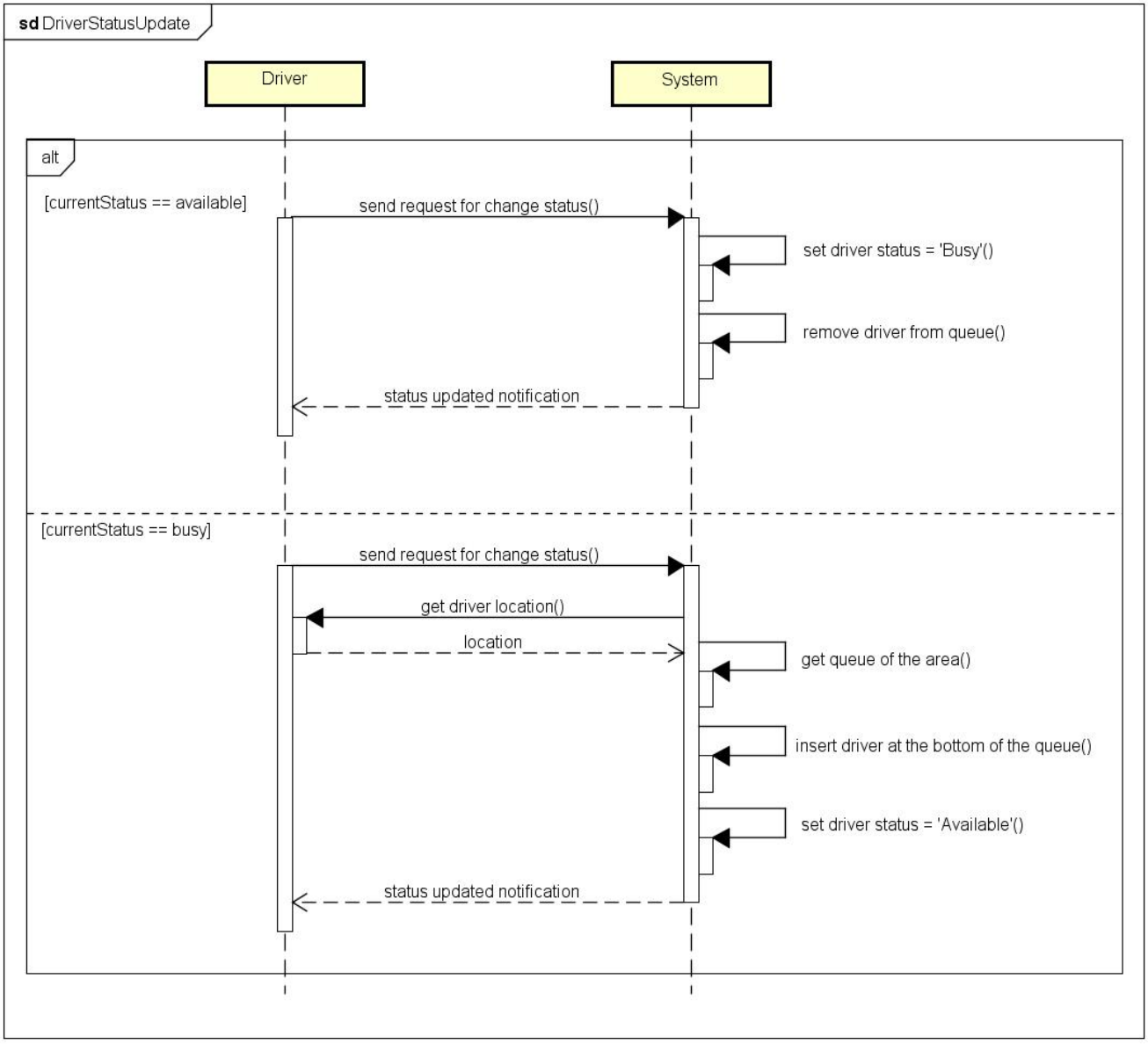


- **Incoming Taxi Request**




report passenger status()
alt
[passengerStatus == not found]
send text message()
loop [timePassed < 5minutes AND passengerStatus == not found]
wait for passengers()
alt
[passengerFound == true]
report passenger found()
alt
[passengerStatus == not found]
delete request()
send text message about request deletion()
get driver location()
location
insert driver to the bottom of the queue in the area()
notify insertion in queue()
alt
[passengerStatus == found]
drop off point()
drop off point()
update request status to "Dispatching" and set end location()
drive to the drop-off point()
report ride ended()
get driver location()
location
insert driver to the bottom of the queue in the area()

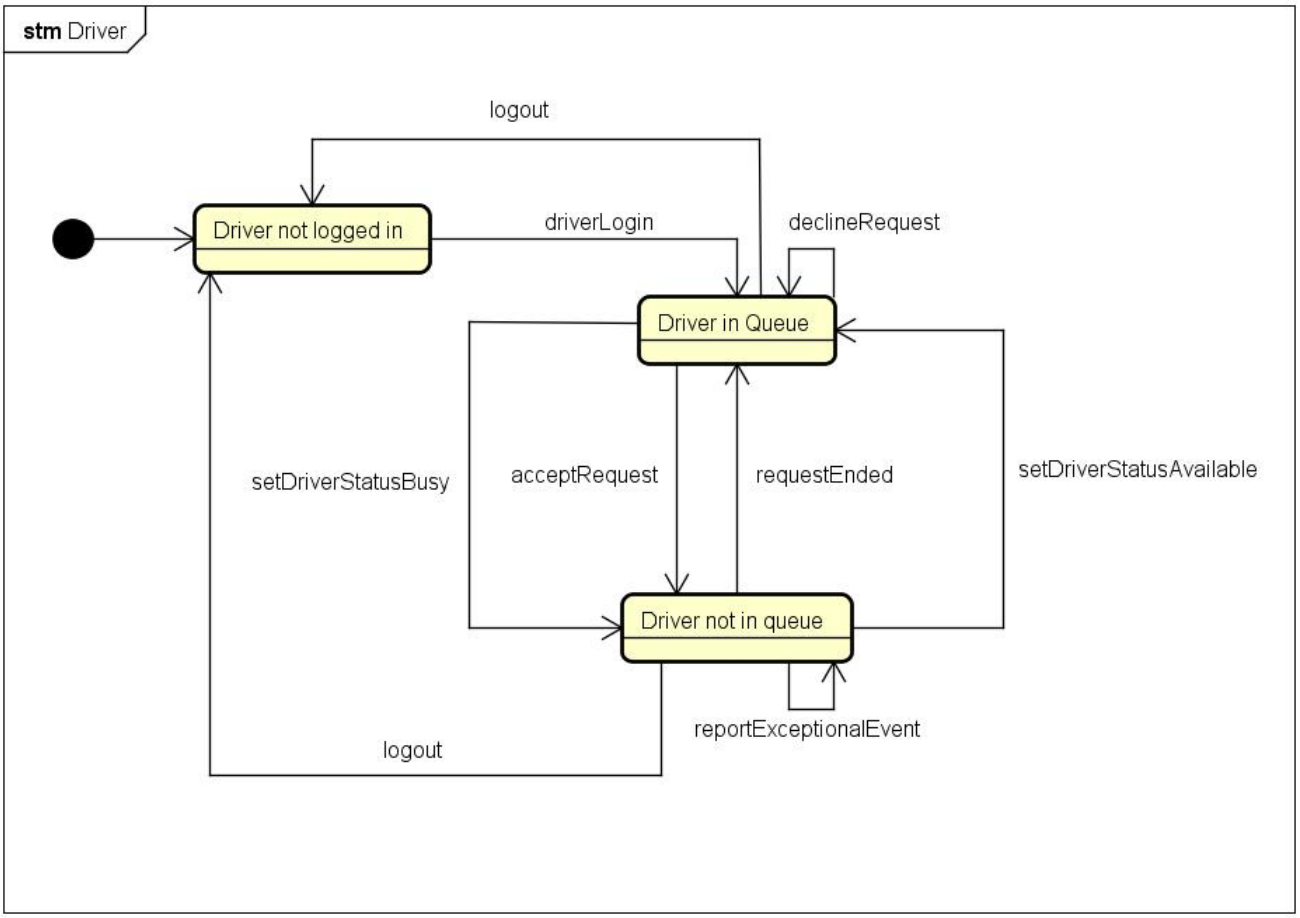
- **Driver Status Update**



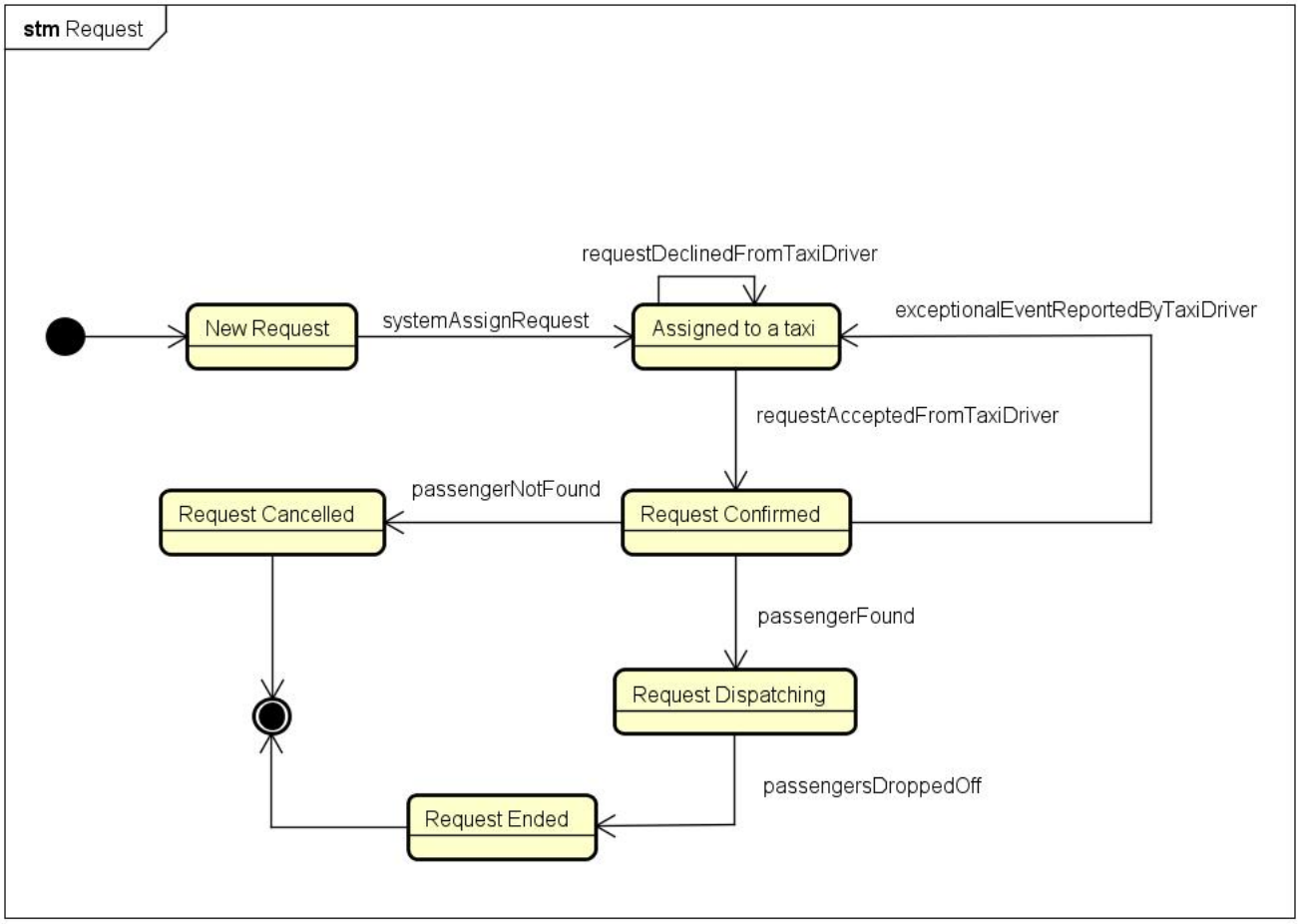
For a better viewing, all the Sequence Diagrams can be also consulted from the folder "Sequence Diagram" inside the github repository.

3.10. State Chart Diagrams

- **Driver**



- **Request**



For a better viewing, all the State Charts Diagrams can be also consulted from the folder "State Charts" inside the github repository.

4. Appendix

4.1. Alloy

4.1.1. Code

```
// for Boolean type
open util/boolean

sig Strings{}

// date and time for requests
sig DateTime {
		day: one Int,
		month: one Int,
		year: one Int,
		hour: one Int,
		minute: one Int
}

// location for requests
sig Location {
		address: one Strings,
		city: one Strings,
		zipCode: one Int
}

// base user data
abstract sig RegisteredUser {
		id: one Int,
		email: one Strings,
		telephoneNumber: one Int
}

// passenger
sig Passenger extends RegisteredUser {
		currentRequest: lone Request
}

// driver
sig Driver extends RegisteredUser {
		busy: one Bool,
		currentRequest: lone Request,
		maxPassengers: one Int
}

// call center operator
sig CallCenterOperator extends RegisteredUser { }
```

```
// request
sig Request {
		id: one Int,
		fromCall: one Bool,
		assignedOperator: lone CallCenterOperator,
		passenger: one Passenger,
		driver:  lone Driver,
		location: one Location,
		ended: one Bool,
		numberOfPassengers: one Int,
		date: one DateTime
}

// queue
sig Queue {
		id: one Int,
		drivers: set Driver,
		area: one Area
}

// area
sig Area {
		id: one Int,
		queue: one Queue
}

// ID must be unique between users
fact noDuplicateUserId {
		no disj u1, u2: RegisteredUser | u1.id = u2.id
}

// ID must be unique between users, independently from their type
fact noDuplicateRoles {
		no p: Passenger, d: Driver, c: CallCenterOperator | p.id = d.id or p.id = c.id or d.id = c.id
}

// Email and phone number must be unique
fact noDuplicateUserData {
		no disj u1, u2: RegisteredUser | (u1.email = u2.email) or (u1.telephoneNumber = u2.telephoneNumber)
}

// Each request ID must be unique
fact noDuplicateRequestId {
		no disj r1, r2: Request | r1.id = r2.id
}

// Each request must be unique
fact noDuplicateRequest {
		no disj r1, r2: Request | (r1.passenger = r2.passenger) and (r1.driver = r2.driver) and (r1.date = r2.date)
and (r1.location = r2.location) and (r1.ended = False or r2.ended = False)
}
```

```
// Each queue ID must be unique
fact noDuplicateQueueId {
		no disj q1, q2: Queue | q1.id = q2.id
}

// Each area ID must be unique
fact noDuplicateAreaId {
		no disj a1, a2: Area | a1.id = a2.id
}

// Each area can have only one queue
fact noDuplicateAreaInQueueu {
		no disj q1, q2: Queue | q1.area = q2.area
}

// Each queue can have only one area
fact noDuplicateQueueInArea {
		no disj a1, a2: Area | a1.queue = a2.queue
}

// If a request is made through a telephone call, it must be insterted by an operator
fact requestMadeByCallHasOperator {
		all r: Request | (r.fromCall = True) => (one c: CallCenterOperator | r.assignedOperator = c)
}

// If a driver is not busy, must be in a queue
// AND
// If a driver is in a queue, must be not busy
fact driverNotBusyIsInAQueueAndViceVersa {
		all d: Driver | one q: Queue | (d.busy = False) <=> (d in q.drivers)
}

// If a driver is busy, must not be in a queue
// AND
// If a driver is not in a queue, must be busy
fact driverBusyNotInQueueAndViceVersa {
		all d: Driver | d.busy = True <=> (no q: Queue | d in q.drivers)
}

// Each taxi can be in only one queue max
fact noDuplicateTaxiInQueue {
		all disj q1, q2: Queue, d: Driver | (d in q1.drivers) <=> (d not in q2.drivers)
}
// Each passenger can have only one active simultaneous request
fact noMultipleActiveRequestForPassenger {
		no disj r1, r2: Request | r1.passenger = r2.passenger and r1.ended = False and r2.ended = False
}

// If a request is not ended, its driver is busy
fact requestNotEndedImpliesDriverBusy {
		all r: Request | (r.ended = False) => (r.driver.busy = True)
}
```

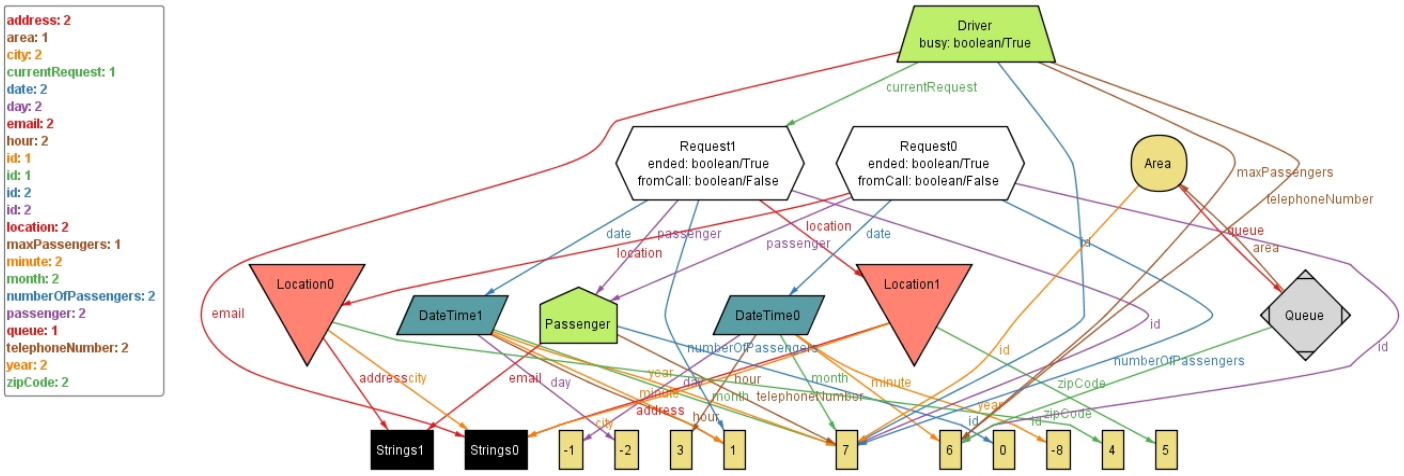
```
// Each driver can have max one active request
fact oneDriverPerActiveRequest {
		no disj r1, r2: Request | r1.driver = r2.driver and r1.ended = False and r1.ended = False
}

// Each driver that have an active request is busy
fact driverInNotEndedRequestIsBusy {
		all d: Driver, r: Request | (r.driver = d and r.ended = False) => d.busy = True
}

// Check that no passenger can have more than one active request
assert noMultipleRequestActiveForPassenger {
		no disj r1, r2: Request | r1.ended = False and r2.ended = False and r1.passenger = r2.passenger
}
check noMultipleRequestActiveForPassenger

// Check that no driver can be assigned to more than one active request
assert noMultipleRequestActiveForDriver {
		all d: Driver | all disj r1, r2: Request | (r1.driver = d and r2.driver = d) => (r1.ended = True or r2.ended =
True)
}
check noMultipleRequestActiveForDriver

// Check that each driver can be at maximum in a queue
assert driverInMaxOneQueue {
		all d: Driver | all disj q1, q2: Queue|  not( (d in q1.drivers) and (d in q2.drivers) )
}
check driverInMaxOneQueue

// Check that all drivers in a queue are not busy
assert noDriverInQueueAndBusy {
		all d: Driver | all q: Queue | (d.busy = True) => (d not in q.drivers)
}
check noDriverInQueueAndBusy

// Check that all drivers not busy are in a queue
assert driverNotBusyIsInAQueue {
		all d: Driver | d.busy = False => (one q: Queue | d in q.drivers)
}
check driverNotBusyIsInAQueue

// Check that all drivers assigned to a request not ended, are busy
assert driverInRequestNotEndedIsBusy {
		all r: Request | r.ended = False => r.driver.busy = True
}
check driverInRequestNotEndedIsBusy

// Check that all request made using a telephone call has an operator assigned
assert requestMadeByCallHasAnOperator {
		all r: Request | r.fromCall = True => (one c: CallCenterOperator | r.assignedOperator = c)
}
check requestMadeByCallHasAnOperator
```

```
// Support function that adds a driver to a queue
pred addDriverToQueue [q1, q2: Queue, d: Driver] {
		q2.drivers = q1.drivers + d
}
run addDriverToQueue

// Support function that removes a driver from a queue
pred removeDriverFromQueue[q1, q2: Queue, d: Driver] {
		q2.drivers = q1.drivers - d
}
run removeDriverFromQueue

// Support function that assigns a request to a driver
pred addDriverToRequest[r: Request, d: Driver] {
		r.driver = d
}
run addDriverToRequest

// Check that adding a driver to a queue and removing he/she, the queue ramins not changed
assert addDriverToQueueUndoesRemoveDriverFromQueue {
		all q1, q2, q3: Queue, d: Driver | (d in q1.drivers and removeDriverFromQueue[q1, q2, d] and
addDriverToQueue[q2, q3, d] ) => q1.drivers = q3.drivers
}
check addDriverToQueueUndoesRemoveDriverFromQueue

// Check that removing a driver from a queue and adding he/she, the queue ramins not changed
assert removeDriverFromQueueUndoesAddDriverToQueue {
		all q1, q2, q3: Queue, d: Driver | (d not in q1.drivers and addDriverToQueue[q1, q2, d] and
removeDriverFromQueue[q2, q3, d] ) => q1.drivers = q3.drivers
}
check removeDriverFromQueueUndoesAddDriverToQueue
// Check if addDriverToQueue works
assert addDriverToQueueIsWorking {
		all d: Driver, q1, q2: Queue | ( (d not in q1.drivers) and d.busy = False and addDriverToQueue[q1, q2, d])
=> (d in q2.drivers)
}
check addDriverToQueueIsWorking

// Check if removeDriverFromQueue works
assert removeDriverFromQueueIsWorking {
		all d: Driver, q1, q2: Queue | (d in q1.drivers and removeDriverFromQueue[q1, q2, d]) => (d not in
q2.drivers)
}
check removeDriverFromQueueIsWorking

// Check if addDriverToRequest works
assert addDriverToRequestIsWorking {
		all r: Request, d: Driver | (r.driver = none and d.busy = False and (one q: Queue | d in q.drivers) and
addDriverToRequest[r, d]) => (r.driver = d and (no q: Queue | d in q.drivers) and d.busy = True)
}
check addDriverToRequestIsWorking
```

```
// Show a world with some requests not ended
pred showRequest {
		some r: Request, d: Driver, p: Passenger | r.driver = d and r.passenger = p and d.busy = True and r.ended
= False
}
run showRequest for 5

// Show a world with drivers available or busy with a request assigned
pred showDrivers {
		some d: Driver | d.busy = False => (no r: Request | r.driver = d)
}
run showDrivers for 5

// Show a world with request made from telephone with an operator assigned
pred showRequestFromCallCenter {
		some r: Request, c: CallCenterOperator | r.fromCall = True and r.assignedOperator = c
}
run showRequestFromCallCenter for 5

// Show a world with drivers in a queue
pred showQueue {
		some d: Driver, q: Queue | d.busy = False => d in q.drivers
}
run showQueue for 5

// Show a world with a certain number of components
pred show {
		#Area = #Queue
		#Area > 0
		#Driver > 0
		#Passenger > 0
		#Request  > 0
}
run show for 5
```

The above code can be also consulted from the folder "Alloy" inside the github repository and the source code file is "souce.als".

4.1.2. Generated World



For a better viewing, the Generated World can be also consulted from the folder "Alloy" inside the github repository.

4.1.3. Results of analysis

20 commands were executed. The results are:

#1: No counterexample found. noMultipleRequestActiveForPassenger may be valid.
#2: No counterexample found. noMultipleRequestActiveForDriver may be valid.
#3: No counterexample found. driverInMaxOneQueue may be valid.
#4: No counterexample found. noDriverInQueueAndBusy may be valid.
#5: No counterexample found. driverNotBusyIsInAQueue may be valid.
#6: No counterexample found. driverInRequestNotEndedIsBusy may be valid.
#7: No counterexample found. requestMadeByCallHasAnOperator may be valid.
#8: **Instance found.** addDriverToQueue is consistent.
#9: **Instance found.** removeDriverFromQueue is consistent.
#10: **Instance found.** addDriverToRequest is consistent.
#11: No counterexample found. addDriverToQueueUndoesRemoveDriverFromQueue may be valid.
#12: No counterexample found. removeDriverFromQueueUndoesAddDriverToQueue may be valid.
#13: No counterexample found. addDriverToQueueIsWorking may be valid.
#14: No counterexample found. removeDriverFromQueueIsWorking may be valid.
#15: No counterexample found. addDriverToRequestIsWorking may be valid.
#16: **Instance found.** showRequest is consistent.
#17: **Instance found.** showDrivers is consistent.
#18: **Instance found.** showRequestFromCallCenter is consistent.
#19: **Instance found.** showQueue is consistent.
#20: **Instance found.** show is consistent.

For a better viewing, the Results can be also consulted from the folder "Alloy" inside the github repository.

4.2. Software and tools used

- ***Microsoft Word 2013*** to redact and format this document.
  - Link: https://products.office.com/it-it/word/
- ***Astah Professional*** to create Use Case Diagrams, Sequence Diagrams, Class Diagram and State Machine Diagram.
  - Link: http://astah.net/editions/professional/
- ***Balsamiq Mockups*** to create user interfaces mockups.
  - Link: https://balsamiq.com/products/mockups/
- ***Alloy Analyzer*** to analyze the consistency of the model.
  - Link: http://alloy.mit.edu/alloy/

4.3. Hours of work

The redaction of the entire document took about 65 hours of work.

4.4. Revision

- **Revision 2:**
  The following Software Interfaces (section 3.1.3) were modified during the creation of the Design Document:
  - The Application Server will be implemented using Glassfish instead of php.
  - The Web Server will be implemented using Glassfish instead of Apache.
- **Revision 3:**
  - Correction of some typographical errors.
  - Reorganization of some requirements.
  - New policy for request assignment in the case of empty queue.