Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Requirements **A**nalysis and **S**pecifications **D**ocument

Davide Azzalini (855185), Fabio Azzalini (855182)

1 Introduction

1.1 Purpose

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

The purpose of this document is to present a detailed description of the web application and mobile app "*myTaxiService*". It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli.

This document is intended for:

- Developers who can review project's capabilities and more easily understand where their efforts should be targeted to improve or add more features to it (design and code the application – it sets the guidelines for future development).
- Project testers can use this document as a base for their testing strategy as some bugs are easier to find using a requirements document. This way testing becomes more methodically organized.
- Project managers who can better measure and control size, cost and schedule of the development process. This document also represents a valid contractual basis between the customer and the developer
- End users of this application who wish to read about what this project can do.

1.2 Scope

The government of a large city aims at optimizing it's taxi service. At this stake myTaxiService will be created, a brand-new application available in two variations: web application and mobile application.

In particular, the attention will be focused on making the access to the service easier from the passengers and on guaranteeing a fairer distribution of the work load among all taxi drivers.

The application will allow passengers to request a ride and taxi drivers to decide weather or not to give the ride. Once a taxi driver has been allocated the passenger will be informed about the waiting time and the code of the incoming taxi.

Furthermore, the application will ensure a more efficient management of the service since each taxi is provided with a GPS tracker and the city is divided in zones, as a result a reduction of waiting times it is expected.

In addition to the standard taxi service (in which the costumer contacts the taxi driver in the exact moment he/she needs the service), will also be possible for the costumers to make reservations for future rides.

Important is also for the system to be developed in such a way that a future addition of some new feature can be made easily and without too much effort.

Moreover, the possibility for payments to be electronic, hence recorded, should help decrease the amount of tax evasion as all transactions can be tracked and accounted for.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Passenger-user: is a registered passenger that can make a request or a reservation for a taxi ride.
- Driver-user: is a taxi driver registered to the service.
- Administrator: System administrator who is given specific permission for managing and controlling the system.
- Web application: is a client-server software application in which the client, user interface, runs in a web browser.
- Mobile app: is a computer program designed to run on a mobile device such as smartphones and tablet computers.
- Request: is a simple call for a taxi ride.
- Reservation: is a ride booked at least two hours before.
- Requests queue: queue associated to zone, for requests.
- Reservations queue: a unique queue for whole city, for reservations.
- Code: the code of the incoming taxi.

- Programmatic Interfaces: is a set of tools that enables the development of additional services.
- Stakeholder: Any person who has interaction with the system who is not a developer.

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document.
- GPS: Global Positional System.
- GUI: Graphic User Interface.
- QoS: Quality of Service.
- DBMS: DataBase management system.
- DB: DataBase.
- OS: Operating System.
- CC: Credit Card.
- UML: Unified Modeling Language.
- JEE: Java Enterprise Edition.
- ES: Environment System.

1.3.3 Abbreviations

- Rat: Rational.
- Desc: Description.
- Dep: Dependency.

1.4 References

- Specification Document: myTaxiService Project AA 2015-2016.pdf.
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- "Articolo 22 codice della privacy".

1.5 Overview

This document is essentially structured in four parts:

- Section 1: Introduction, gives a scope description and overview of everything included in this RASD document. Also, the purpose for this document is described and a list of abbreviations and definitions is provided.
- Section 2: Overall Description, gives an overview of the whole system. The system will be explained in its context to show how the system interacts with other systems and introduce the basic functionality of it. It will also describe what type of stakeholders that will use the system and what functionality is available for each type. At last, the constraints and assumptions for the system will be presented.
- Section 3: Specific Requirements, contains all of the functional and quality requirements of the system. It gives a detailed description of the system and all its features. To give an easy way to understand all functionality of this software, typical scenarios, use cases and UML diagrams are provided.
- Section 4: Appendix A, contains some information about the attached .als file and some described screenshot of software used to generate it.

2 Overall Description

2.1 Product perspective

myTaxiService is a brand new system. It's free and open source.

The system will consist of four parts:

- A mobile app intended for passengers.
- A web application intended for passengers.
- A mobile app intended for taxi drivers.
- A web portal for the administrators.

The two applications meant for passengers offer the user exactly the same functionalities, except for the fact that the mobile one can use the GPS of the smartphone to fix the current location and tell the taxi driver to go there to pick up the passenger.

Both mobile applications will need to communicate to a GPS application within the smartphone or the tablet computer. The GPS will provide the geographical coordinates to the mobile application with locations of the user. The functionality provided by the GPS will be embedded into the application in order for the user to be able to use the functions in the application in a seamlessly manner.
Each application needs to communicate with a central engine consisting in a web server and a DataBase. This entity is also in charge of overseeing the queues and managing the communications between drivers and passengers.

The web portal for the administrator will be very frugal and will provide standard administration features.

The mobile applications are designed to run on the most diffused OS: iOS, Android and Windows Phone, with an active internet connection. The web application is supported by the most popular browsers (Safari, Firefox, Internet Explorer and Google Chrome).

Mobile applications have some restrictions about the resource allocation. To avoid problems with overloading the operating system the applications are only allowed to use 20 megabytes of memory while running the application. The maximum amount of hard drive space is also 20 megabytes.

2.2 Product functions

The applications meant for passenger-user mainly allow them to:

- Register.
- Log in.
- View and modify account information.
- Make requests right away.
- Make reservations for future rides.
- Receive notifications about the waiting time and the taxi code for a pending request.
- Receive notification about a reservation.
- Display, at the end of the ride, all the information about the route and the cost of the service.
- Associate a CC.
- Pay for a ride.

The mobile app meant for taxi drivers allow them to:

- Log in.
- View personal information.
- Set their status as available or not.
- Confirm if they are going to take care of a certain request.
- Confirm if they are going to take care of a certain reservation.
- View the confirmation of a payment.
- Check reservations scheduled.
- Tell the system if the passenger does not show up.
- Display, at the end of the ride, all the information about the route and the cost of the service.

2.3 User characteristics

There are three types of users that interact with the system: passenger-users, driver-users and administrators. Each of these three types of users perform different uses of the system so each of them has their own technical expertise.

Passenger-users have to be able to use an application with a standard GUI that makes use of maps and online payment.

Driver-users have to be able to use an application with a standard GUI that makes use of maps.

Administrators manage the overall system.

2.4 Constraints

The mobile application is constrained by the system interface to the GPS navigation system within the mobile phone. Since there are multiple system and multiple GPS manufacturers, the interface will most likely not be the same for every one of them. Also, there may be a difference between what navigation features each of them provide.

The Internet connection is also a constraint for the application. Since the application fetches data from the database over the Internet, it is crucial that there is an Internet connection for the application to function.

Since a mobile application can not access any content on the cellphone unless you give your permission, our application will have to ask the user for the permission to use Location Services.

According to the european policy "Directive on Privacy and Electronics Communications" end users have to consent for the placement of cookies, and similar technologies for storing and accessing information on users' equipment.

The possibility to pay by credit card imposes to respect the related policies.

2.5 Assumptions and dependencies

- Since the costumer is a governmental authority it is more realistic to assume that taxi drivers cannot register to the system by themselves. An employee of the headquarter, after having run the mandatory checks, will send by mail the log-in information to the taxi driver. The administrator of the system will take care of inserting the new access credentials in the DB.
- When a passenger make a request he/she will need to specify the starting and the end point of the ride and the number of passengers.

- When a passenger make a reservation he/she will need to specify the origin and the destination of the ride, the date and time and the number of passengers.
- By availability we mean that when a taxi driver start working he/she sets the status as available. As soon as a passenger gets in the car the driver sets the status as not available and then again available in the moment the passenger gets out.
- A taxi driver can chose to accept or not a reservation.
- There is a "special" queue for reservations. The reservations queue is unique and doesn't take into consideration of the the different zones in which the city is divided into. When a taxi driver accepts or declines a reservation he/she will be moved in the last position of this queue. We choose to have only one queue for reservations since it's plausible that reservations will be made with a conspicuous advance (even though the specific tells "at least two hours") therefore taxi drivers will have much time to organize themselves.
- When a taxi driver completes a request he/she will stay still in the zone in which he/she is located. If he/she has a reservation scheduled he/she can go there. This assumption has been made in order to prevent the need to check the current position of each available taxi and eventually move that taxi in the tail of a new queue if it has changed zone even though no passengers was on board. A system like that, to be precise and reliable, would have need to check the current position of available taxis very intensively and so would have been too expensive to be achieved.
- When a taxi driver accepts a request he/she will be removed from the queue of that zone and as soon as he/she has completed the request he/she will be inserted in the last position of the queue of the zone in which the ride has ended.
- To avoid conflicts between the two types of queues the following assumption is made. When a taxi driver is on the top of a request queue, before the system assign to him/her a certain request a check has to be performed. The system computes a valuation of the required time for that ride and adds the time that will be required to go from the destination of that request to the origin of the his/her forthcoming scheduled reservation. If this valuation establish that it is possible for the driver to take care of both calls, the system will will forward him/her that request, otherwise that taxi driver will be removed from the queue and he/she will only take care of the reservation.
- It is possible for a taxi driver to have more than one reservation scheduled, but at the moment of allocating a new one the system will check that no

conflict will be created with the others already scheduled. If a conflict will be detected, the taxi driver will not be pushed to the tail but in the second position. We choose to act this way because reservations, unlike requests, are assumed to be more spread over time and so another conflict is less likely to happen.

- Users will use e-mail address as username.
- Reservations have to be paid at the moment of submitting by credit card, and the price will only depend on the distance
- When the system asks a taxi driver if he/she wants to take care of a certain request or reservation, the taxi driver has only a reasonable amount of time to reply.
- It is compulsory for the passenger to associate a credit card to the account. The passenger still can choose whether to pay by credit card or by cash but having a CC linked to the account means that the taxi company is protected if a passenger makes a request an then doesn't show up.

2.6 Apportioning of requirements

- In the 2.0 version of this System will be introduced a taxi sharing service.

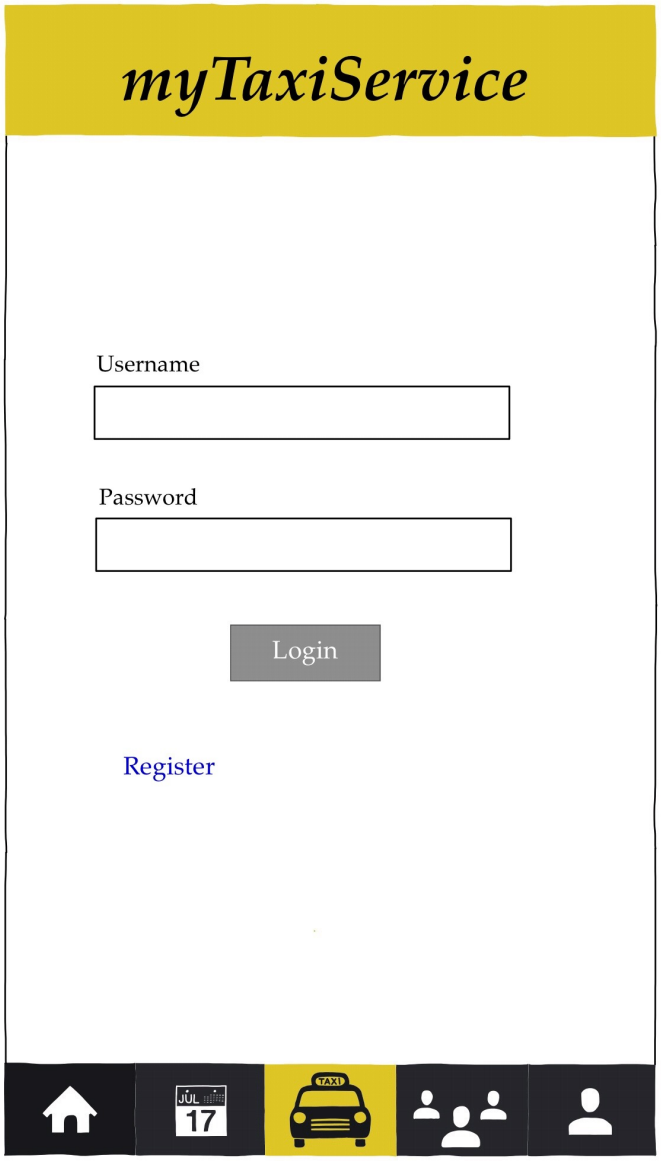
3 Specific requirements

3.1 External interface

3.1.1 User interfaces

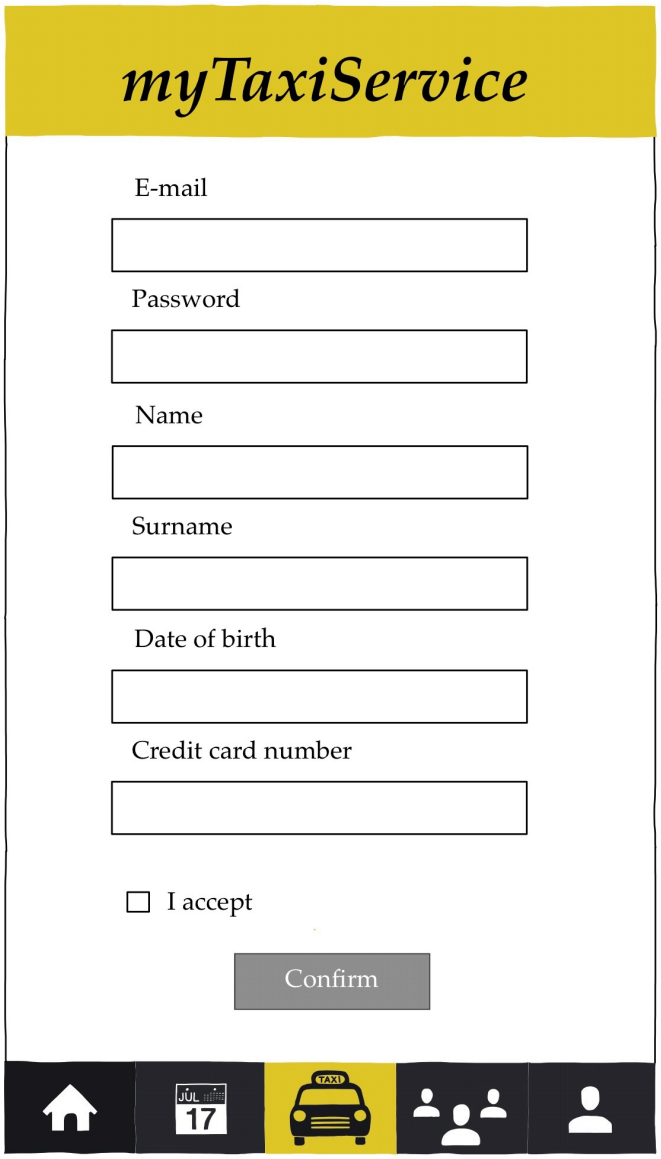
3.1.1.1 Log in passenger App

The mockup below lets registered passenger to login and offers a link to the registration form to those who are not already registered.



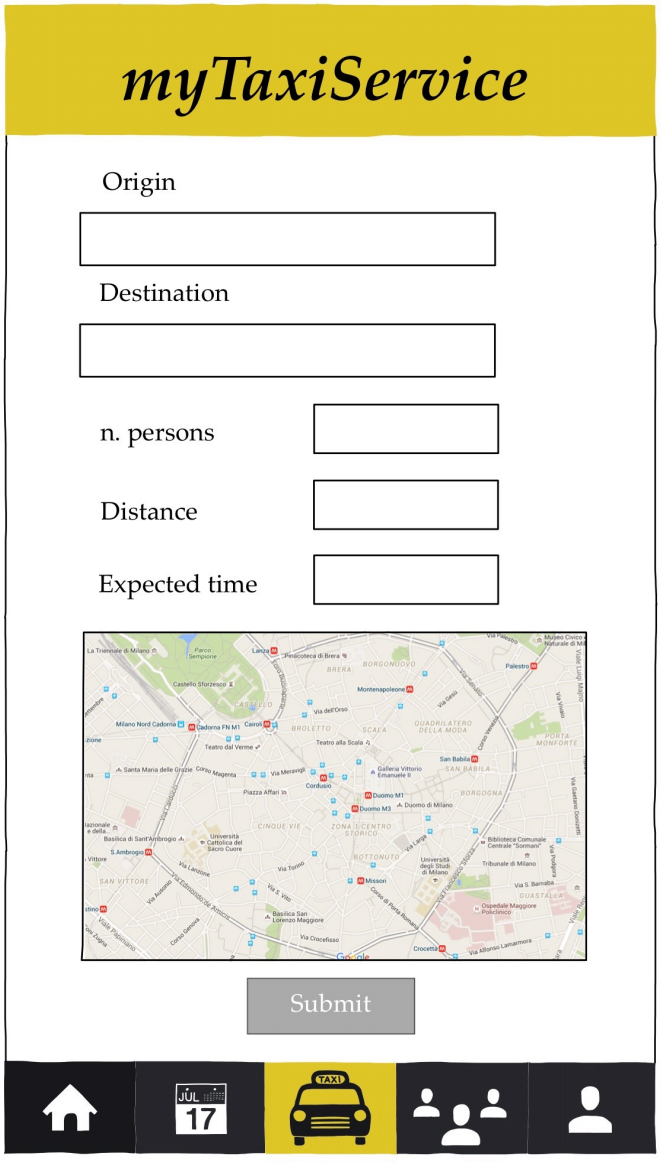
3.1.1.2 Passengers registration form

This mock shows the registration form window.



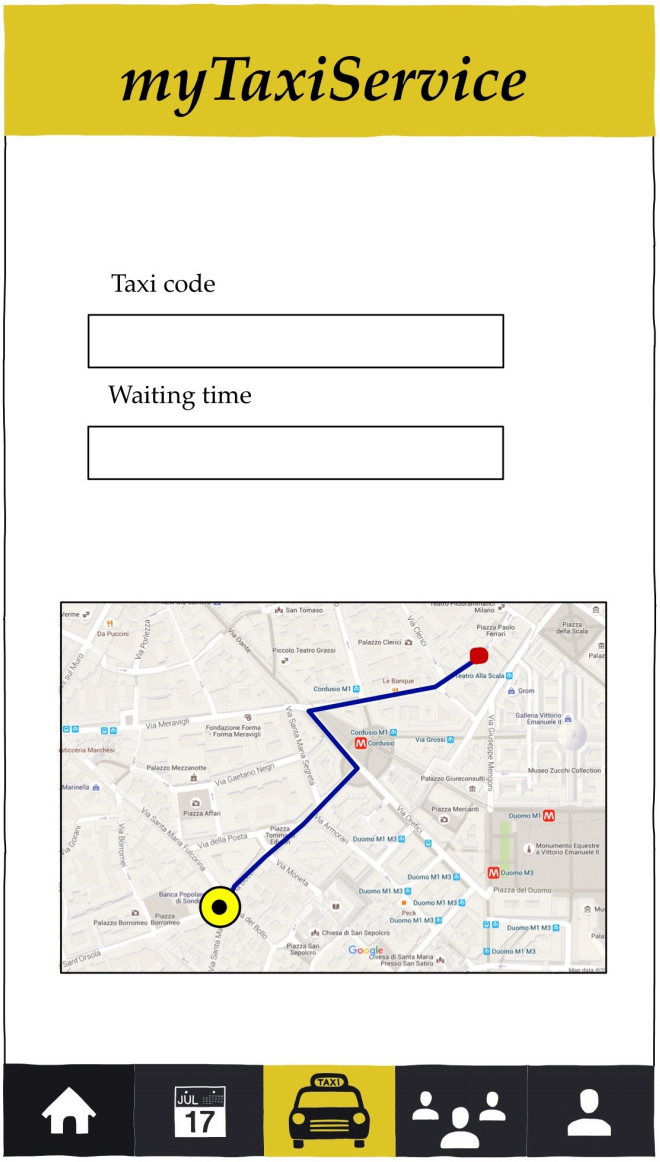
3.1.1.3 New request form

It's possible to reach this window with the yellow button in the middle. After having filled the first three information, the system will suggest a valuation of the expected duration and price for the ride. The passenger can choose his/hers current location as origin simply by selecting "Use current location" that will appear as soon as he/she taps the origin text box.



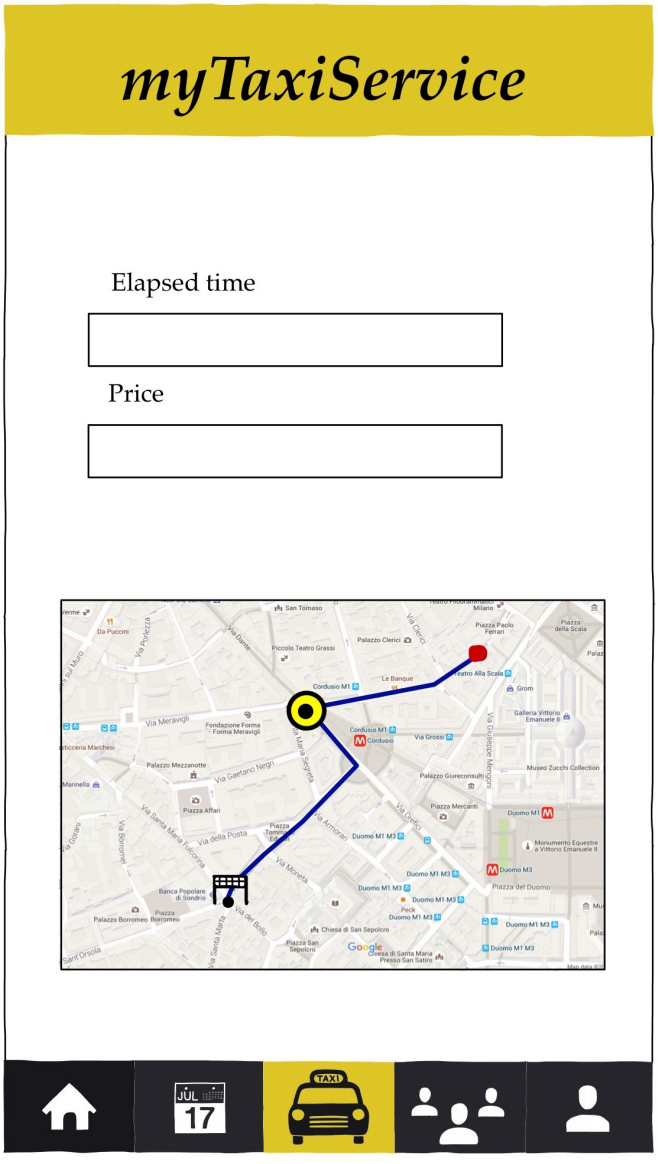
3.1.1.4 Waiting for a taxi

This is the window that automatically appears when a passenger submits his/her request. As soon as a taxi driver is allocated will be shown the code and an esteem of the waiting time. In the map below will be shown a realtime tracking of the taxi as he/she comes closer to the passenger.



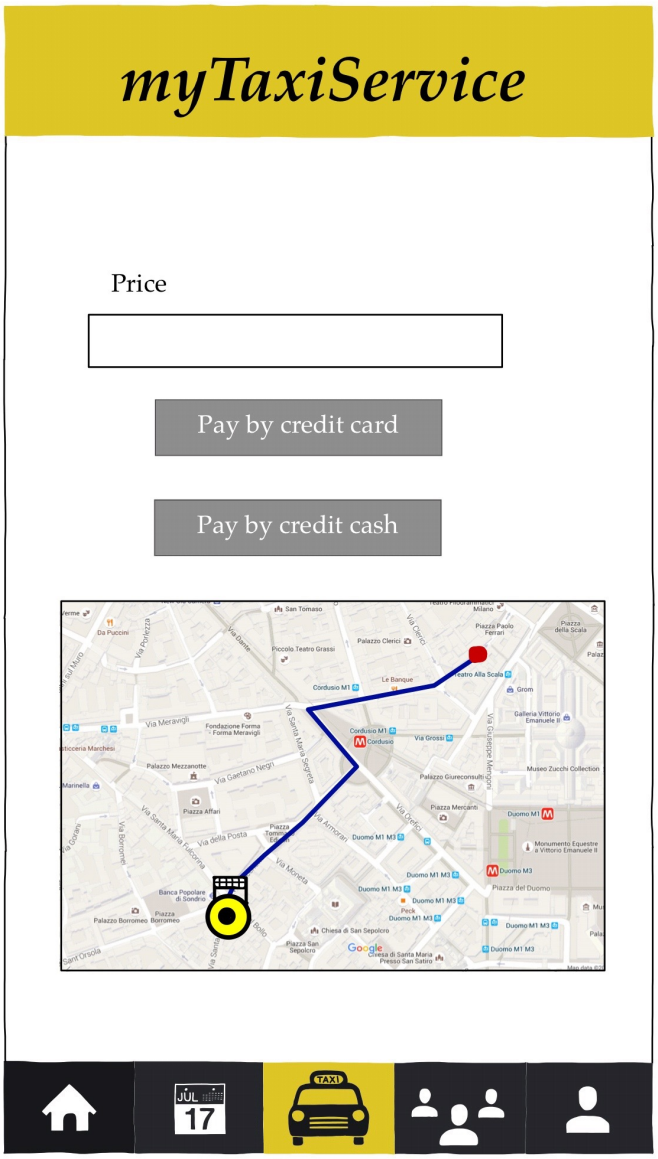
3.1.1.5 The ride

As soon as the passenger gets into the taxi this view automatically appears. The passenger will see the price of the ride increase along with the time. On the map will be shown the current position of the taxi, the origin and the destination.



3.1.1.6 End of a ride

As soon as the taxi driver notifies the system that the destination is reached this window will automatically appear on the passenger smartphone. The passenger can choose weather to pay by the credit card associated to the account or by cash to the taxi driver. Soon this window will disappear and the new request form will be show.

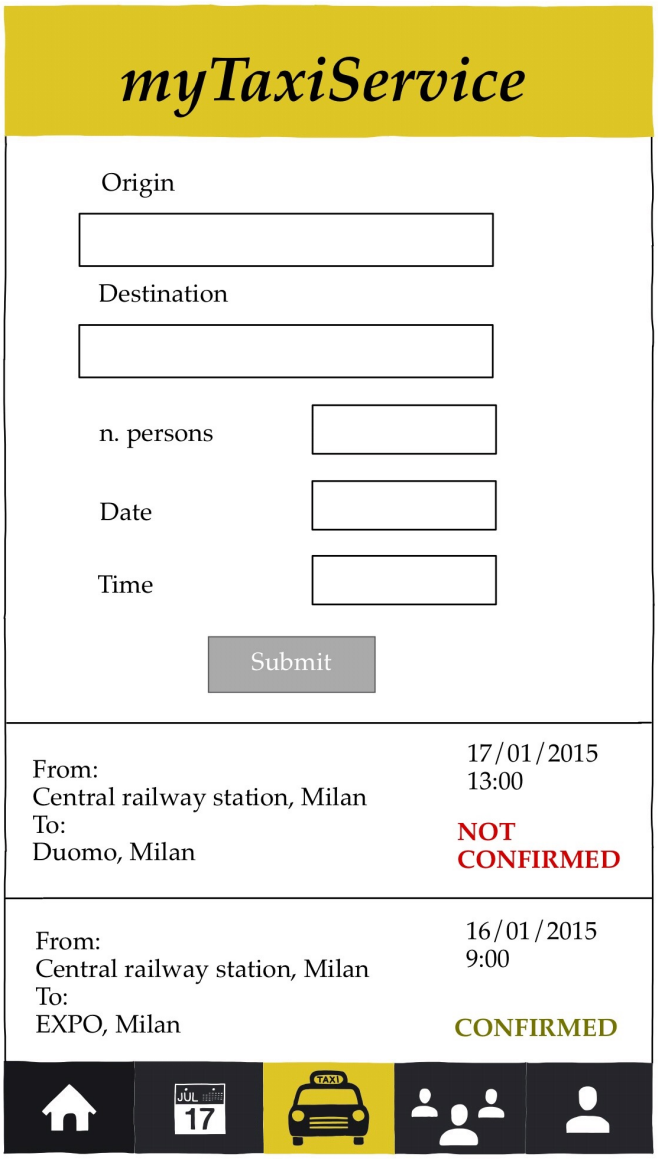


3.1.1.7 Home page

It is possible to reach this window with the first button on the left (the one whit the little house). Here the passenger can find information about the zone in which he/she currently is and the number of taxi driver available in that zone.

3.1.1.8 Reservation form

It is possible to reach this window by pushing the second button on the left (the one with the calendar).



3.1.1.9 Taxi sharing

This feature will be implemented in the second release of the application.

3.1.1.10 Personal profile page

It is possible to view and modify personal information.

It is also possible to log out.

The most recent rides are shown.

3.2 Functions

3.2.1 Functional requirements

3.2.1.1 Allow a passenger to become a registered user (passenger-user) mobile application - web app.

- Passengers must not be already registered to perform registration.
- Users can not log in twice but only one for session.
- Non registered passengers can only see the log in window.
- Non registered passengers can only access the registration form.
- The e-mail address used as username must not already be used by an other user.
- CC data must be correct.
- E-mail used for registration must be formally correct.

3.2.1.2 Allow a user (passenger-user or driver-user) to login to the application (web application or mobile app for passenger-users).

- User access credentials must exist in the DB in order for the log in process to succeed.
- User must know his/her username and password chosen the registration to success log in.
- Wrong access credential will not grant the access to the application.
- Users can not access to any service before log in.
- The applications will not implement retrieve password mechanism.

3.2.1.3 Allow a passenger-user to make a request for a ride.

- Passengers must be already registered and logged in the application.
- User must fill correctly each mandatory field in order to submit the request.
- User must submit the request.
- In order to select the origin by using the current location provided, the GPS must be enabled.

3.2.1.4 The system notifies the passenger-user code, waiting time and GPS information of the incoming taxi

- Passenger must be already registered and logged in the application.

- User-passenger must have submitted a pending request.
- An available driver-user in that zone must have received and told the system his/her willingness of taking care of that request.

3.2.1.5 Allow a driver-user to receive a request for a ride.

- Taxi driver must be already registered and logged in the application.
- The driver-user must have set his/her status as available.
- The driver-user must have enabled the GPS.
- The driver-user must be in the same zone of the passenger-user that have just submitted the request.
- The driver-user does not have any scheduled reservation that would be conflicting with this request.
- The driver-user must be in the first position of the queue of that zone.

3.2.1.6 Allow a driver-user to inform the system about his/her availability.

- Taxi driver must be already registered and logged in the application.
- The driver-user must have enabled the GPS.

3.2.1.7 Allow a driver-user to inform the system about his/her willingness of taking care or not a certain request.

- Taxi driver must be already registered and logged in the application.
- The driver-user must have just received a notification of a request.

3.2.1.8 Allow a passenger-user to make a reservation for a future ride.

- Passengers must be already registered and logged in the application.
- User must fill correctly each mandatory field in order to submit the request.
- User must submit the request.
- User must have enough money on the CC to pay for the ride.
- In order to select the origin by using the current location provided, the GPS must be enabled.

3.2.1.9 Allow a passenger-user to see a schedule of his/her future reservations and check their status.

- Passengers must be already registered and logged in the application.

3.2.1.10 Allow a driver-user to receive a reservation for a future ride and to decide whether or not to take care of it.

- Taxi driver must be already registered and logged in the application.
- The driver-user does not have any other reservation that would be conflicting with this one.
- A passenger user has just submitted a reservation.
- The driver-user must be in the first position of the queue of the reservations.

3.2.1.11 Allow a driver-user to see a list of his/her scheduled reservations for the future.

- Taxi driver must be already registered and logged in the application.

3.2.1.12 Allow a passenger-user to decide how to pay a request for a ride and eventually to pay by CC.

- Passengers must be already registered and logged in the application.
- The taxi driver must have informed the system that the ride has reached the destination.
- User must have enough money on the CC to pay for the ride if he/she decide to pay by CC.

3.2.1.13 Allow a driver-user to inform the system if the passenger show up or not.

- Taxi driver must be already registered and logged in the application.
- The driver-user is taking care of a request.

3.2.1.14 Allow a passenger-user to see a realtime progression of the ride (only mobile app).

- Passengers must be already registered and logged in the application.
- The driver-user must have enabled the GPS.
- The driver-user must have told the system that the passenger has showed up.

3.2.1.15 Modify personal information.

- Passengers must be already registered and logged in the application.
- Information change must be valid (see registration requirement).

3.2.2 Scenarios

3.2.2.1 Scenario 1

Meredith is an esteemed surgeon of a well known hospital in Seattle. Even if tonight she's not on call the hospital needs her because an old man has just arrived to the ER and she is the only doctor able to perform the procedure needed. Unfortunately that wasn't definitely her plan for the night, in fact she wanted to stay home watching the TV. Since she was sure she wouldn't have needed the car, that morning she had agreed to lend it to her sister in law, Amelia. Luckily for the patient Meredith is a great fan of myTaxiService so she could promptly login and submit a request without loosing too much time. The system choose the first taxi driver in the queue of her zone, Jimmy, that agreed of taking care of her request. After few minutes Jimmy is right outside Meredith's house, she pops in and in 10 minutes the car is at the hospital, Meredith decides to pay by the credit card associated to her account, gets out, and runs toward his patient. If it wouldn't have been for a system so reliable and optimized maybe the poor patient wouldn't have made it.

3.2.2.2 Scenario 2

Jake is a taxi driver member of myTaxiService. He lives in the suburbs of our city, a very peaceful place where everyone is friend with the neighbors, the air quality is better than downtown and children play in the street. Just as he does every morning, today he gets in his taxi, sets his status as available and wait for the first request of the day. The day goes by as every other day with requests and reservations. But today isn't a day like every other, in fact, tonight, Bree, a Jake's neighbor, has invited everyone in the neighbor over for one of her fancy dinners. Everyone in the neighbor knows how good Bree is at cooking and Jake really doesn't want to miss the dinner so today he will go home one hour before. Unfortunately just after the one that should have been the last ride of the day, the system forwards him a request. Bree's cooking is so good that even to miss only the appetizer would be a shame, so Jake decides to reject the request, sets his status as not available and goes home. The passenger will not have to wait long, in fact the system promptly forwards the request to the second taxi driver in the queue that agrees and takes care of it.

3.2.2.3 Scenario 3

Harvey and Louis are colleagues and both work as conselors in the best law firm of the city. They are always in competition and spend their days making the fool out of each other with pranks and jokes. Louis is very proud of myTaxiService, a new application he has been using since a week now. Harvey decides to steal Louis's phone in order to make him a prank, so he opens myTaxiService and makes a request with the intention of not showing up so that Louis will have to pay the fine since he's the one whose credit card is associated to the account. Shortly after a taxi arrives at the meeting point and the driver, not seeing anyone, signals the system that the passenger is missing. The system charges Louis's CC for the minimum fare.

3.2.2.4 Scenario 4

It's almost christmas time and Rebecca, a foreign art student, is going back to her country for the holidays. She has already booked a flight for the next week and now she wants also to arrange also the travel to the airport. Fortunately the night before she went out with some friends for the last goodbye before the holidays. Lisa, one of her closest friend, told her about this new and fantastic application: myTaxiService. So Rebecca decides to register herself and then logs into the application, choses the correct window and makes a reservation. Shortly after the system confirms her reservation. A week later, as planned, a taxi will bring her to the airport perfectly on time for her flight. Tanks to myTaxiService she will be able to spend christmas eve with her beloved family.

3.2.2.5 Scenario 5

Betty works as personal assistant for the editor in chief of a famous fashion magazine. It's September and this one is the more important issue of the year. The September issue is much more full of contents with respect to the other issues, so the staff have needed some extra time to finish the work. But it's not all gone, there is still a chance, if Betty brings the book to the printing press within an hour the magazine will be anyway in the news-stands in time. There is nothing better than a taxi when it comes to fast rides. Betty logs into the application and submits a request, the system promptly finds an available taxi driver and Betty, known to be a very anxious person, checks on the applications the taxi as comes closer and closer.

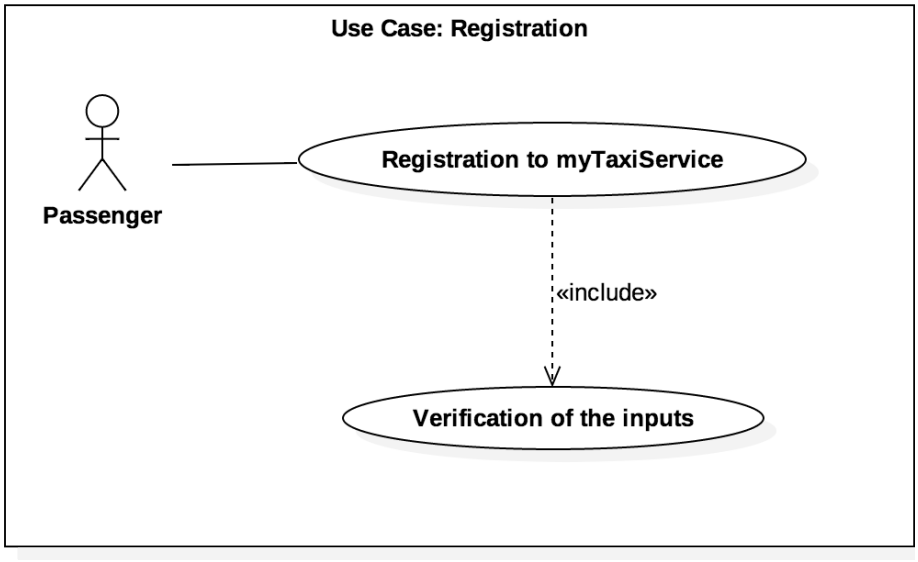
Betty is already on the sidewalk when the taxi arrives, pops in and for all the duration of the ride never looks away from the screen.

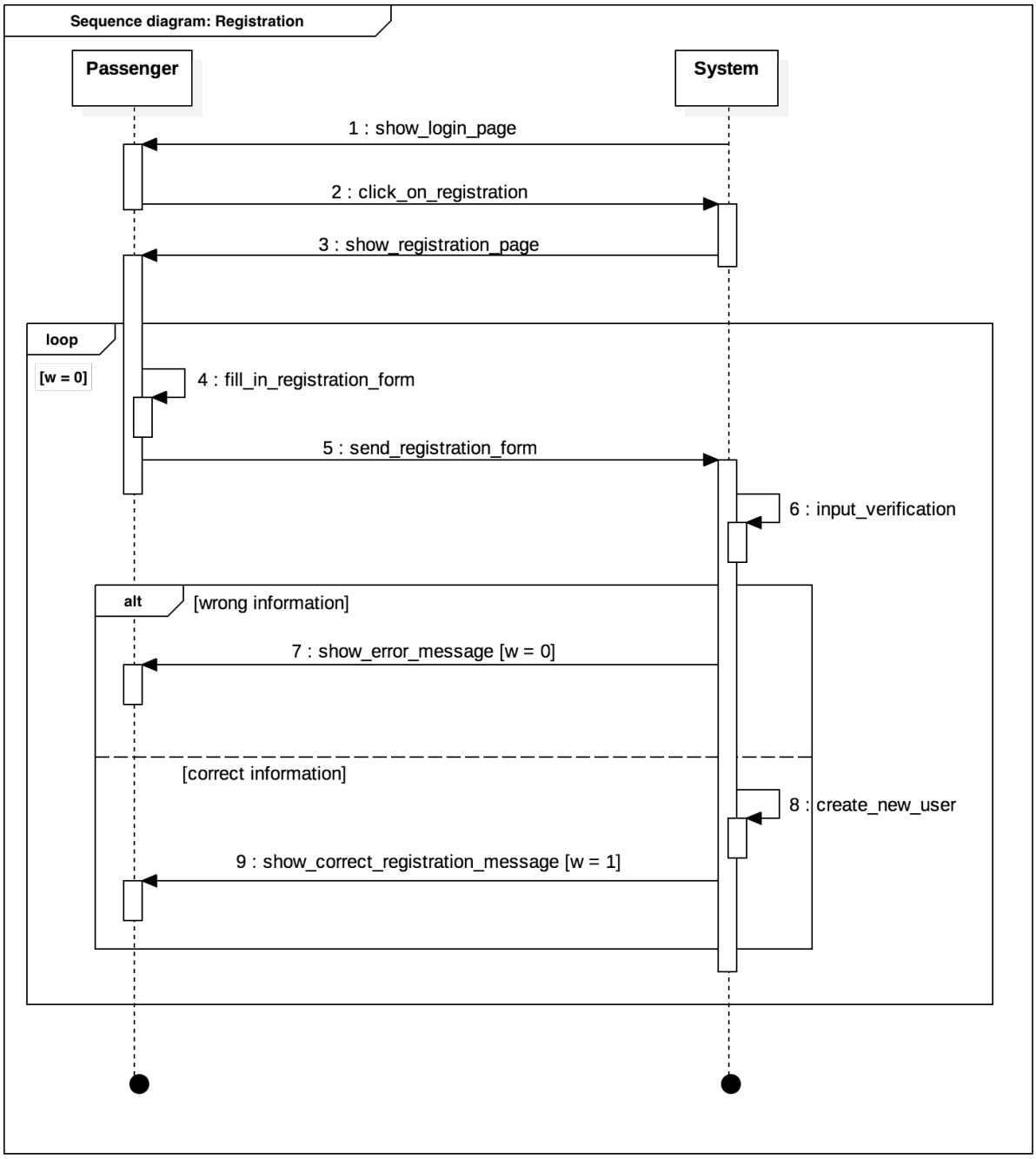
Thanks to myTaxiSerice a little bit of Betty worry has been saved since she could have checked the progression of the journey on the application.

3.2.3 UML models

3.2.3.1 Registration

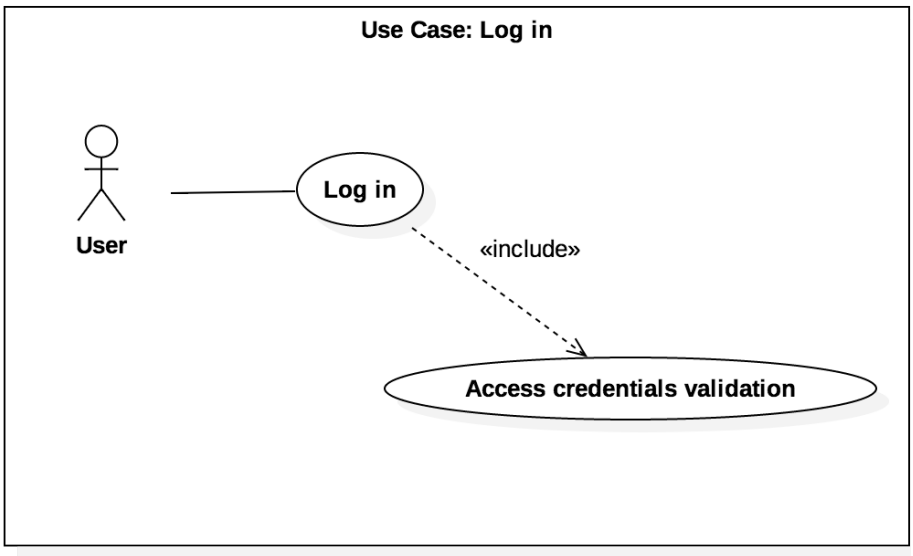
Name	Registration.
Actor	Passenger.
Goal	Allow a passenger to become a registered user (passenger-user) mobile application - web app.
Entry conditions	The passenger is not registered.
Flow of events	1. The passenger, on the log in windows clicks on the registration link. 2. The passenger fills all the mandatory fields. 3. The passenger clicks on the “I accept” check box. 4. The passenger clicks on the “Confirm” button. 5. The system check and save the data about the passenger in the DB.
Exit conditions	Registration successfully done. Now the passenger is a passenger-user. From now on he/she can log in to the application (either web or mobile) using his/her access credentials and start using myTaxiService
Exceptions	1. At least one of the mandatory fields does not contain a valid input. 2. The e-mail address is already associated to an other passenger-user. 3. The passenger has not clicked on the “I accept” check box. All the exceptions are handled informing the passenger about the problem and showing again the registration page.

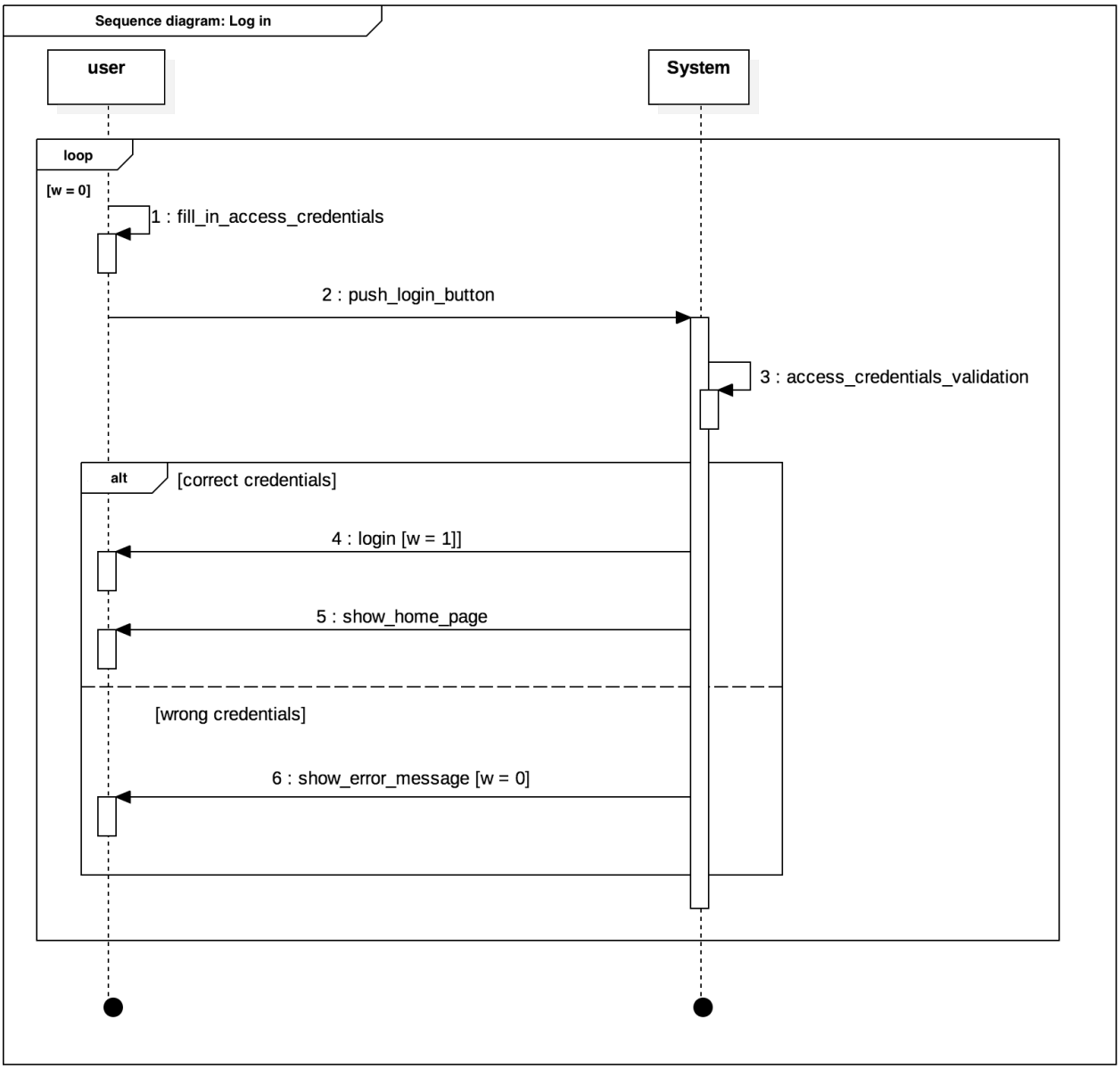

Use Case: Registration
Passenger
Registration to myTaxiService
«include»
Verification of the inputs


Sequence diagram: Registration
Passenger
System
1 : show_login_page
2 : click_on_registration
3 : show_registration_page
loop
[w = 0]
4 : fill_in_registration_form
5 : send_registration_form
6 : input_verification
alt
[wrong information]
7 : show_error_message [w = 0]
[correct information]
8 : create_new_user
9 : show_correct_registration_message [w = 1]

3.2.3.2 Log in

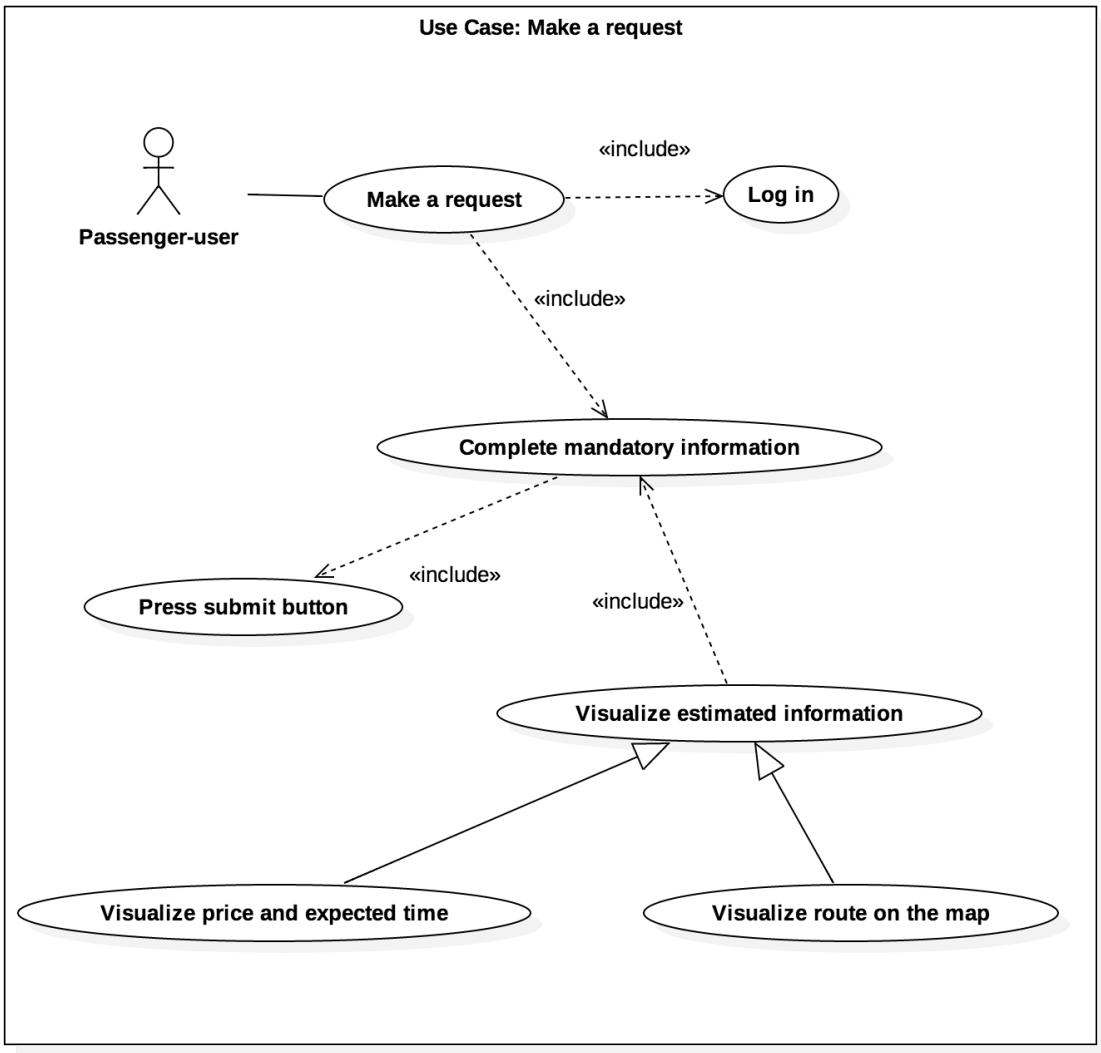
Name	Log in.
Actor	User (passenger-user or driver-user).
Goal	Allow a user (passenger-user or driver-user) to login to the application (web application or mobile app for passenger-users).
Entry conditions	The user is registered to the application.
Flow of events	1. The user is in the log in page. 2. The user complete the form inserting correct username and password. 3. The user clicks on the "log in" button. 4. The system authorize the access if the credentials are correct.
Exit conditions	The home page is shown.
Exceptions	1. The password and/or the username are wrong. The exceptions is handled informing the passenger about the problem and showing again the log in page.

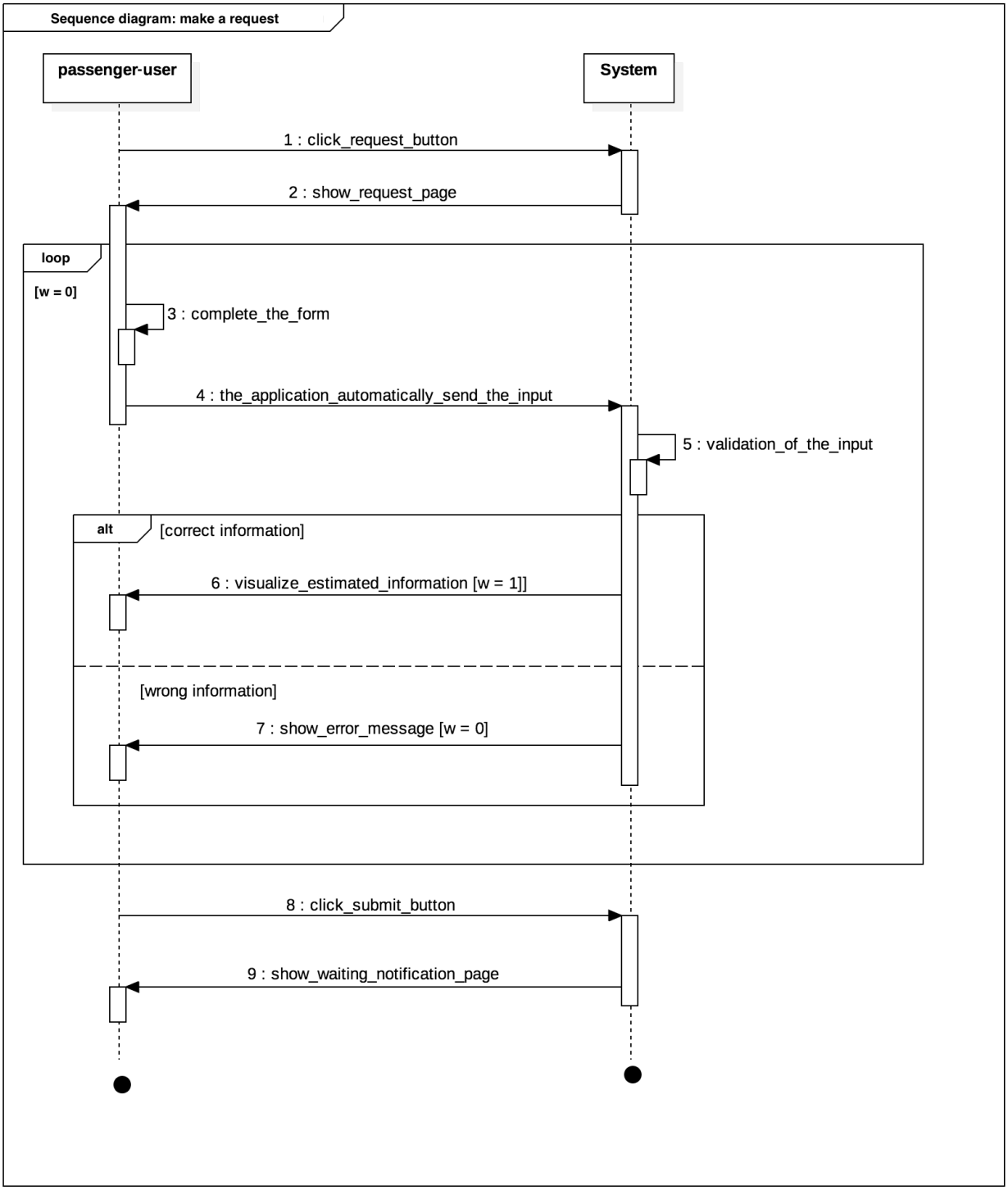



Sequence diagram: Log in
user
System
loop
[w = 0]
1 : fill_in_access_credentials
2 : push_login_button
3 : access_credentials_validation
alt
[correct credentials]
4 : login [w = 1]]
5 : show_home_page
[wrong credentials]
6 : show_error_message [w = 0]

3.2.3.3 Make a request

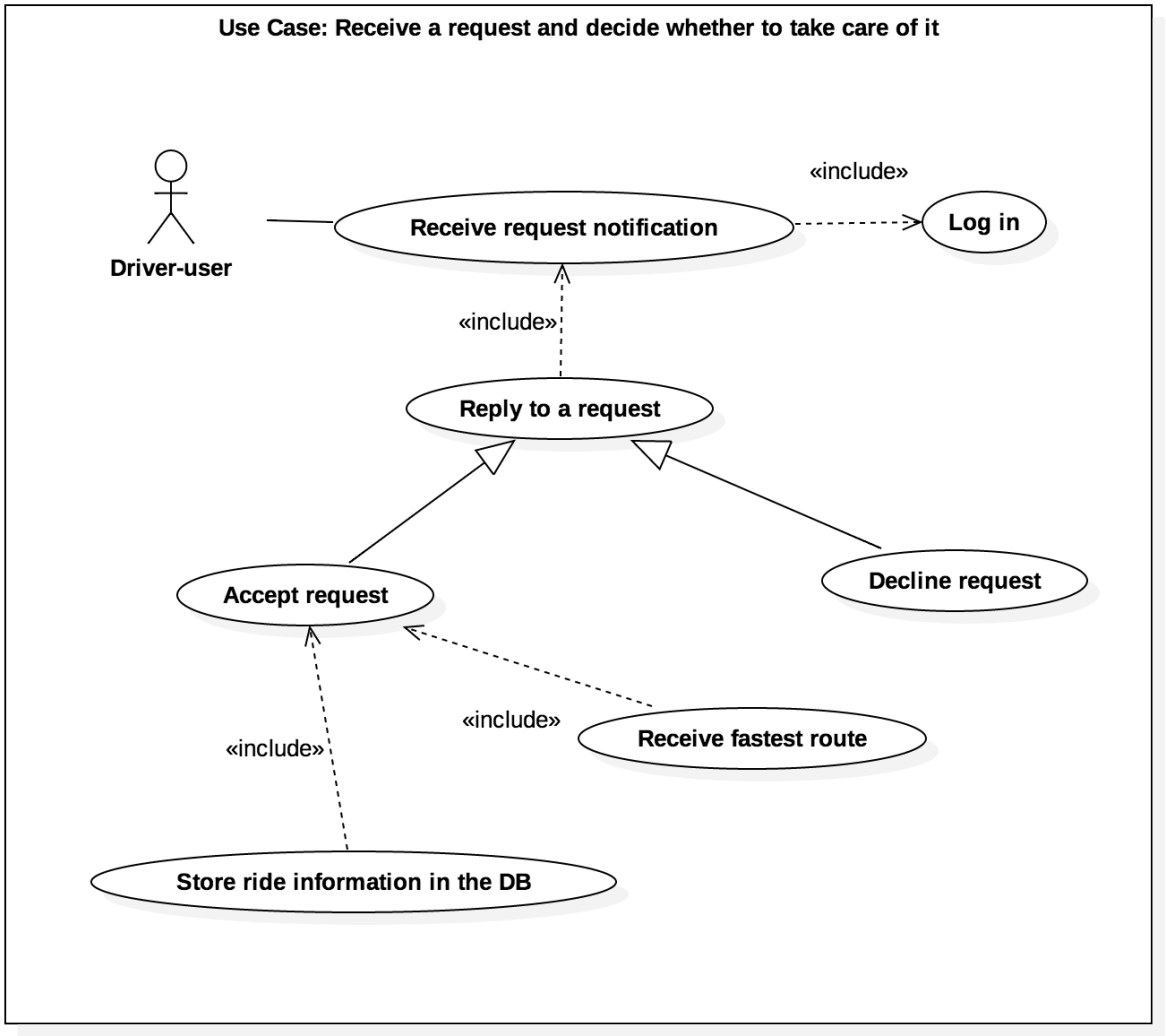
Name	Make a request.
Actor	Passenger-user.
Goal	Allow a passenger-user to make a request for a ride.
Entry conditions	The passenger-user must be already logged in to myTaxiService.
Flow of events	1. The passenger-user clicks on the central button, the one with a taxi on it. 2. The passenger-user fills origin destination and number of persons. 3. The system estimate the price and the expected time for the ride. 4. The system shows the route of the ride on the map. 5. The passenger-user clicks the “submit” button. 6. The system shows a waiting page until a driver-user has been found.
Exit conditions	The waiting page is shown. The passenger-user has to wait for a notification.
Exceptions	1. At least one of the mandatory fields does not contain a valid input. The exceptions is handled informing the passenger about the problem and showing again the request page.

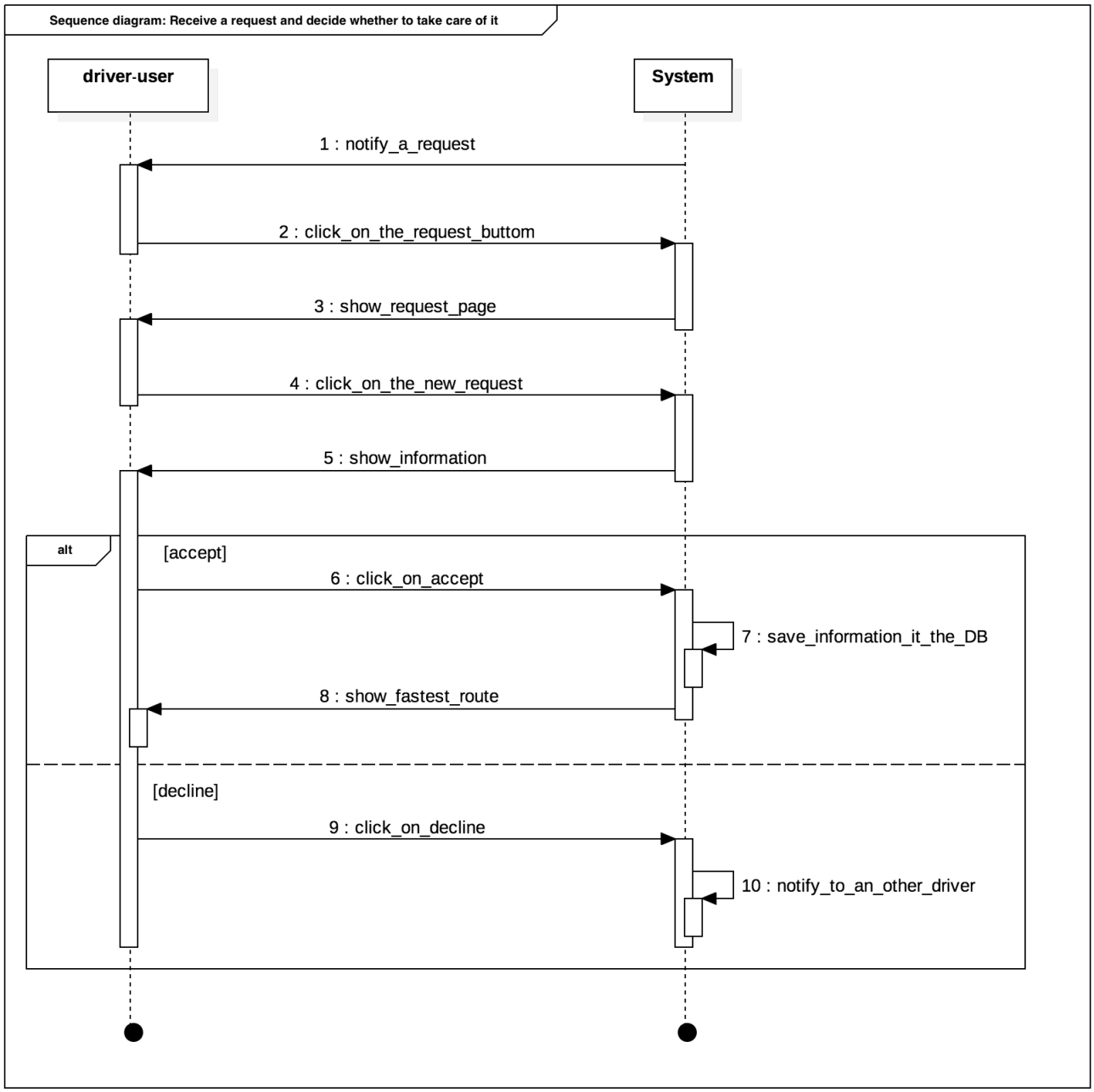

Use Case: Make a request
Passenger-user
Make a request
«include»
Log in
«include»
Complete mandatory information
Press submit button
«include»
«include»
Visualize estimated information
Visualize price and expected time
Visualize route on the map


Sequence diagram: make a request
passenger-user
System
1 : click_request_button
2 : show_request_page
loop
[w = 0]
3 : complete_the_form
4 : the_application_automatically_send_the_input
5 : validation_of_the_input
alt
[correct information]
6 : visualize_estimated_information [w = 1]]
[wrong information]
7 : show_error_message [w = 0]
8 : click_submit_button
9 : show_waiting_notification_page

3.2.3.4 Receive a request and decide whether to take care of it

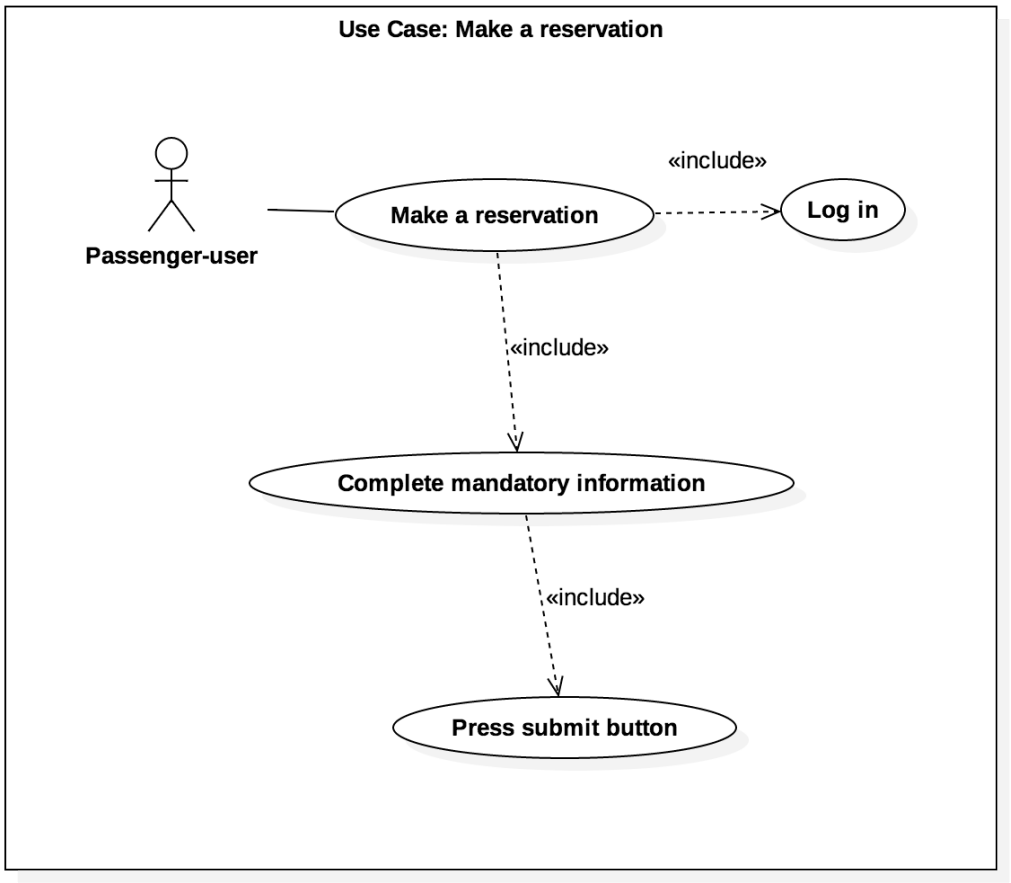
Name	Receive a request and decide whether to take care of it.
Actor	Driver-user.
Goal	Allow a driver-user to receive a request for a ride and to inform the system about his/her willingness of taking care of this request.
Entry conditions	The driver-user must be already logged in to myTaxiService. The taxi-driver must be the first in the queue of that zone.
Flow of events	1. The application notify a new request. 2. The driver-user clicks on the central button, the one with a taxi on it. 3. The driver-user clicks on the new request to see all the information about the ride. 4. The driver-user clicks on the “accept” button if he/she wants to take care of the request. 5. The driver user click on the “decline” button if he/she does not want to take care of the request. 6. The system shows a page with the fastest route to reach the passenger if the driver-user has accepted the request.
Exit conditions	If the driver-user has accepted, he/she and the passenger-user who has made the request are stored in the data in the DB with the information about the ride. If the driver-user has not accepted, the request is sent to the second driver-user in the queue of that zone. The driver-user that has declined the request now is in the last position of the queue of that zone.
Exceptions	None.

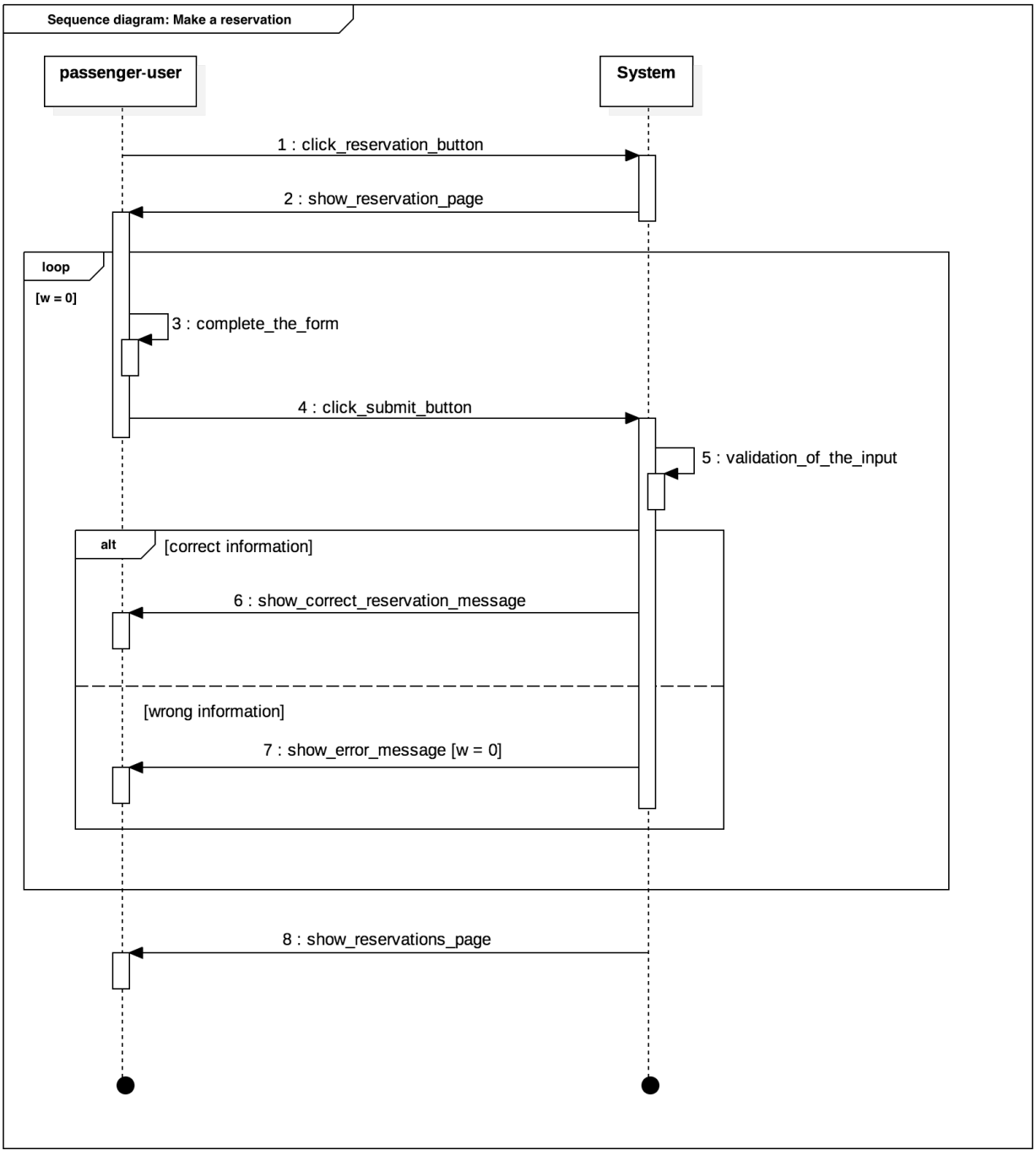

Use Case: Receive a request and decide whether to take care of it
Driver-user
Receive request notification
«include»
Log in
«include»
Reply to a request
Accept request
Decline request
«include»
Receive fastest route
«include»
Store ride information in the DB


Sequence diagram: Receive a request and decide whether to take care of it
driver-user
System
1 : notify_a_request
2 : click_on_the_request_buttom
3 : show_request_page
4 : click_on_the_new_request
5 : show_information
alt
[accept]
6 : click_on_accept
7 : save_information_it_the_DB
8 : show_fastest_route
[decline]
9 : click_on_decline
10 : notify_to_an_other_driver

3.2.3.5 Make a reservation

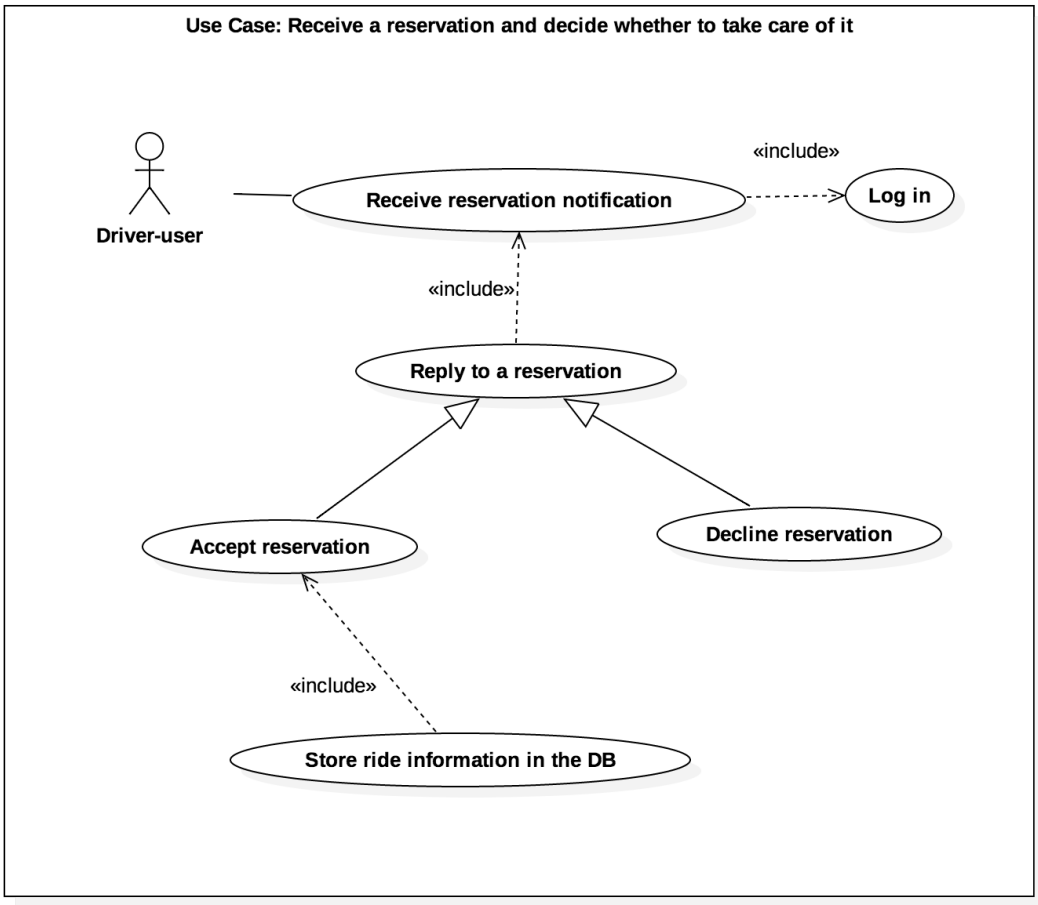
Name	Make a reservation.
Actor	Passenger-user.
Goal	Allow a passenger-user to make a reservation for a ride.
Entry conditions	The passenger-user must be already logged in to myTaxiService.
Flow of events	1. The passenger-user clicks on the second button, the one with a calendar on it. 2. The passenger-user fills origin destination, number of persons, date and time. 3. The passenger-user clicks the “submit” button. 4. The system add the reservation to the reservations page with the status: “not confirmed”.
Exit conditions	The reservations page is shown. The passenger-user has to wait for the confirm of the ride.
Exceptions	1. At least one of the mandatory fields does not contain a valid input. The exceptions is handled informing the passenger about the problem and showing again the reservations page.

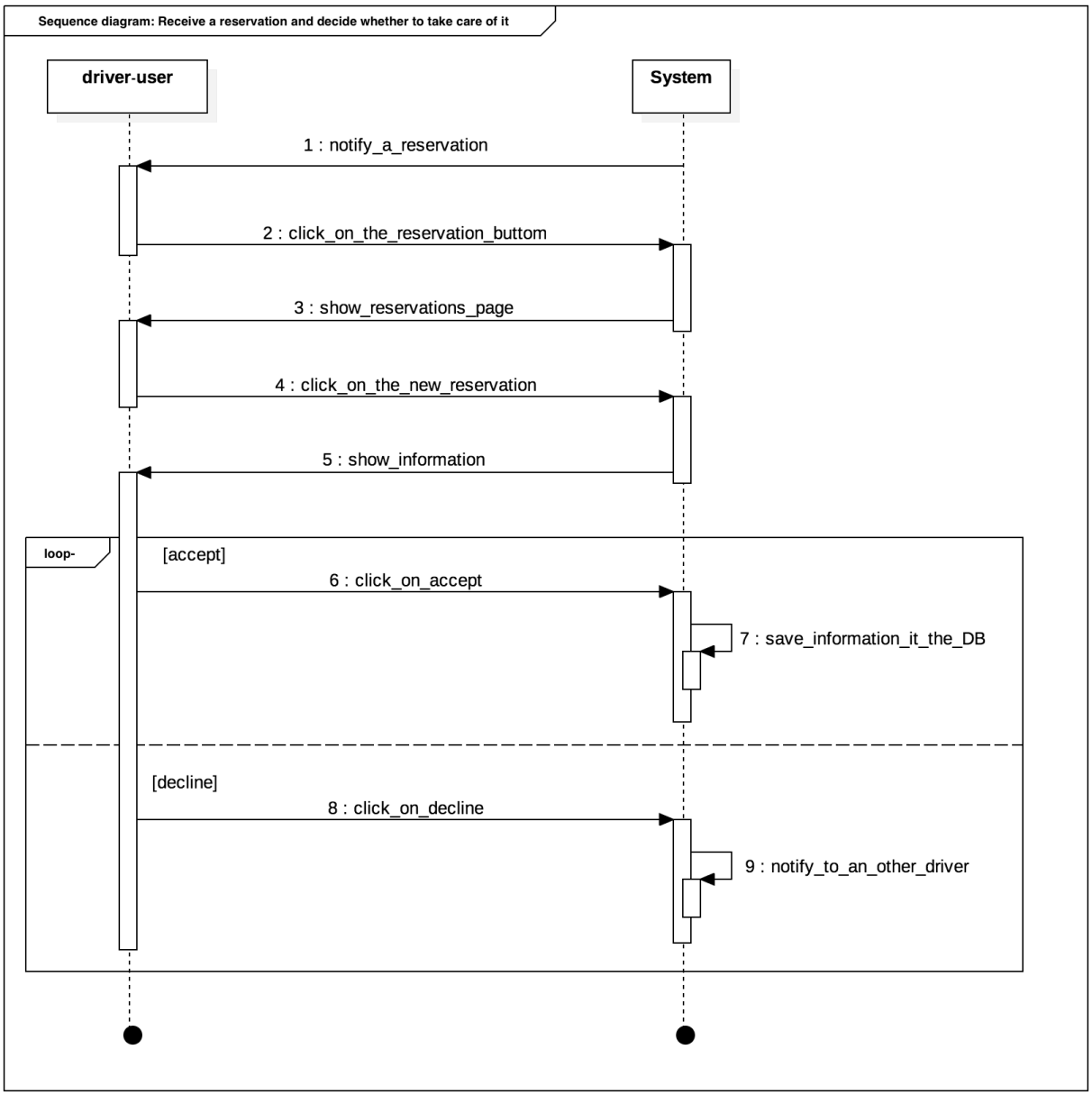

Use Case: Make a reservation
Passenger-user
Make a reservation
«include»
Log in
«include»
Complete mandatory information
«include»
Press submit button


Sequence diagram: Make a reservation
passenger-user
System
1 : click_reservation_button
2 : show_reservation_page
loop
[w = 0]
3 : complete_the_form
4 : click_submit_button
5 : validation_of_the_input
alt
[correct information]
6 : show_correct_reservation_message
[wrong information]
7 : show_error_message [w = 0]
8 : show_reservations_page

3.2.3.6 Receive a reservation and decide whether to take care of it

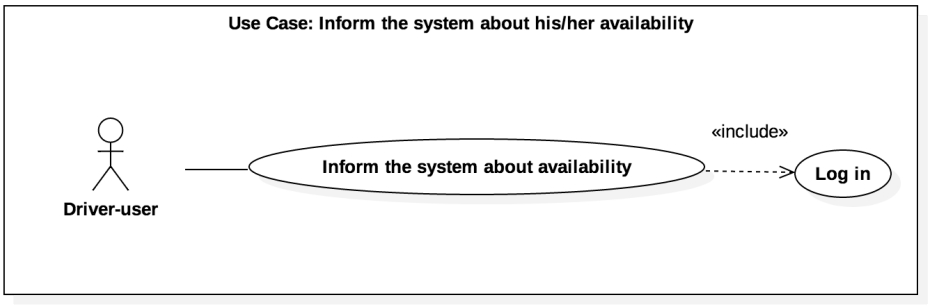
Name	Receive a reservation and decide whether to take care of it.
Actor	Driver-user.
Goal	Allow a driver-user to receive a reservation for a ride and to inform the system about his/her willingness of taking care of this reservation.
Entry conditions	The driver-user must be already logged in to myTaxiService. The taxi-driver must be the first in the reservations queue.
Flow of events	1. The application notify a new reservations. 2. The driver-user clicks on the second button, the one with a calendar on it. 3. The driver-user clicks on the new reservation to see all the information about the ride. 4. The driver-user clicks on the “accept” button if he/she wants to take care of the reservation. 5. The driver user click on the “decline” button if he/she does not want to take care of the reservation. 6. The reservation is saved in the scheduled reservations, if the driver-user has accepted the reservation.
Exit conditions	If the driver-user has accepted, he/she and the passenger-user who has made the reservation are stored in the data in the DB with the information about the ride. If the driver-user has not accepted, the reservation is sent to the second driver-user in the reservation queue. The driver-user that has declined the reservation now is in the last position of the reservation queue.
Exceptions	None.

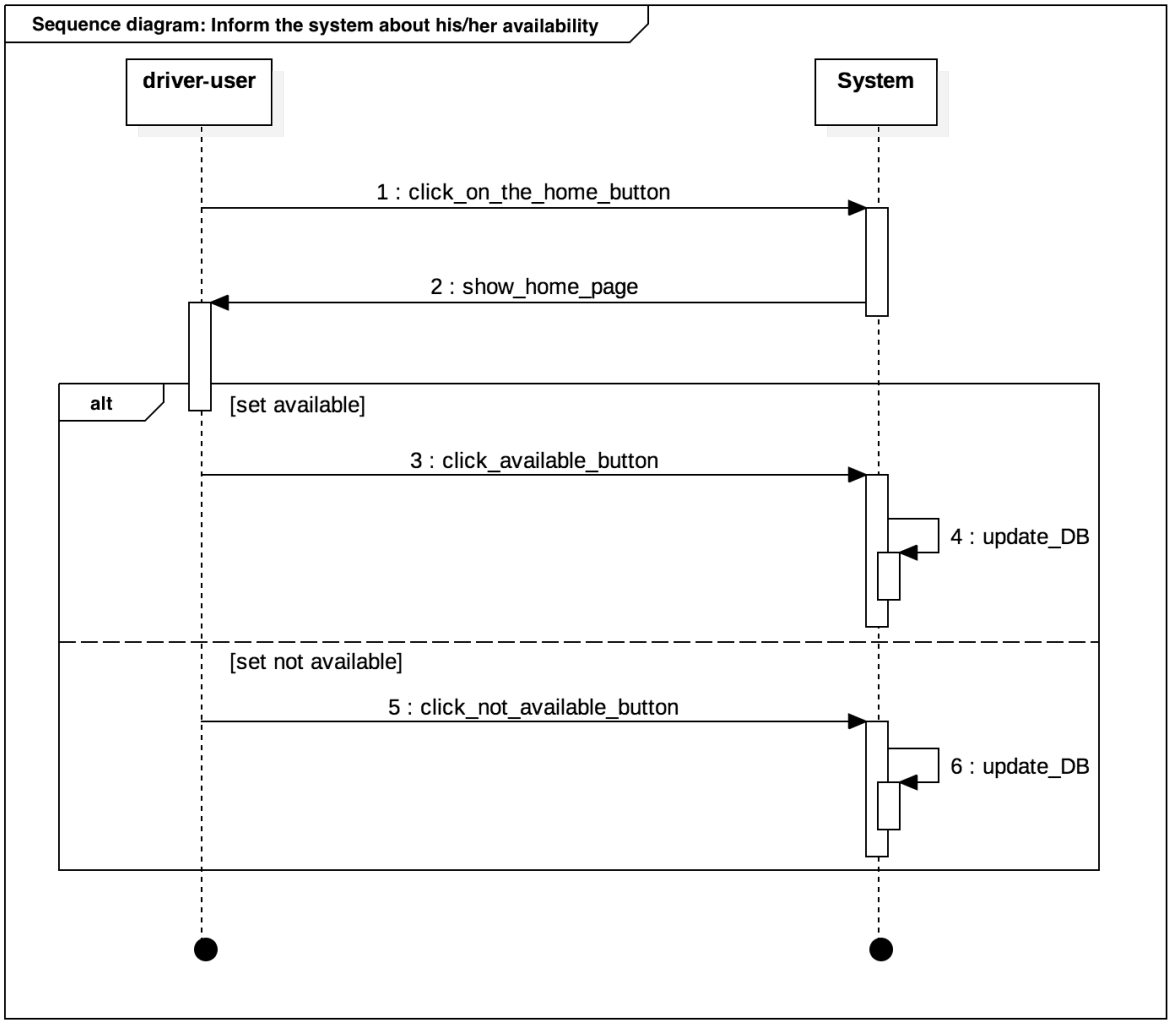

Use Case: Receive a reservation and decide whether to take care of it
Driver-user
Receive reservation notification
«include»
Log in
«include»
Reply to a reservation
Accept reservation
Decline reservation
«include»
Store ride information in the DB


Sequence diagram: Receive a reservation and decide whether to take care of it
driver-user
System
1 : notify_a_reservation
2 : click_on_the_reservation_buttom
3 : show_reservations_page
4 : click_on_the_new_reservation
5 : show_information
loop-
[accept]
6 : click_on_accept
7 : save_information_it_the_DB
[decline]
8 : click_on_decline
9 : notify_to_an_other_driver

3.2.3.7 Inform the system about his/her availability

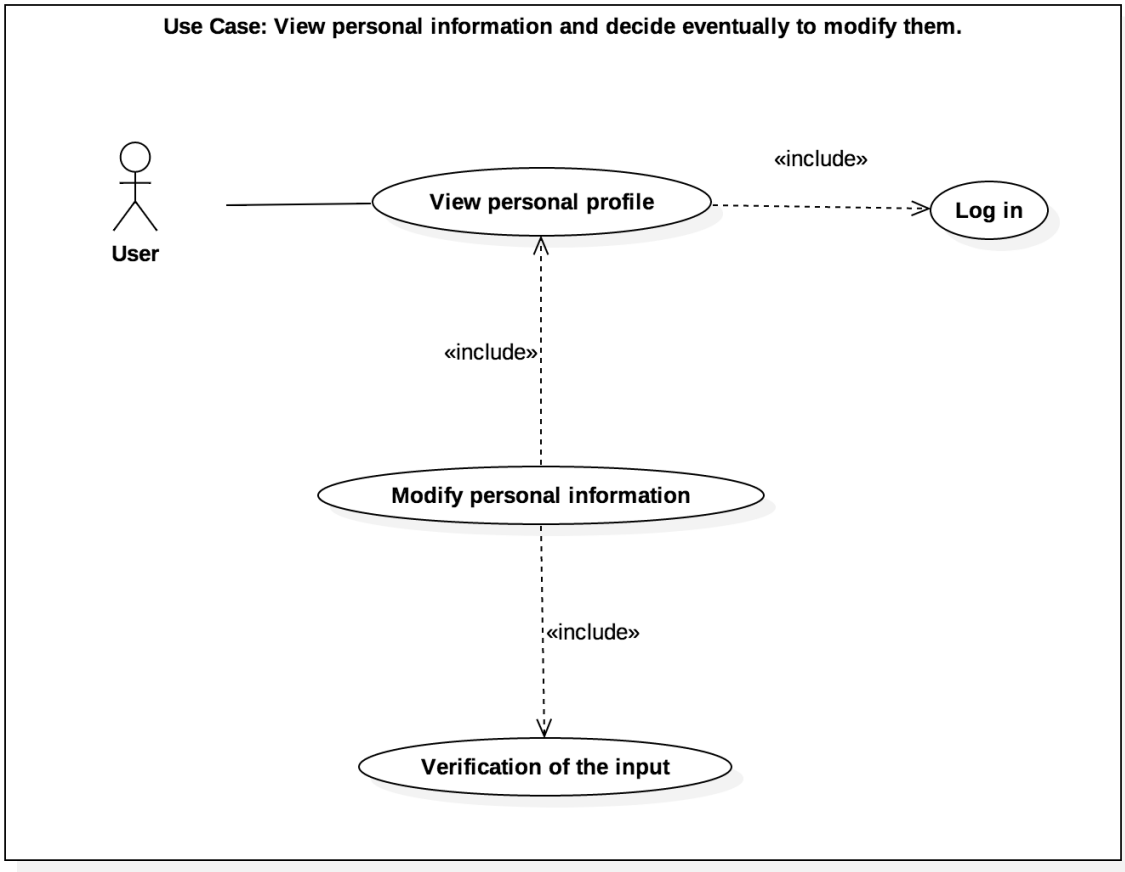
Name	Inform the system about his/her availability.
Actor	Driver-user.
Goal	Allow a driver-user to inform the system about his/her availability.
Entry conditions	The driver-user must be already logged in to myTaxiService. The taxi-driver must not be busy in a reservation or a request.
Flow of events	1. The driver-user clicks on the first button, the one with a hose on it. 2. The driver-user clicks on the “available” button if he/she wants to inform the system about his/her availability. 3. The driver-user clicks on the “not available” button if he/she wants to inform the system that he/she is not available.
Exit conditions	If the driver-user is available, he/she can be part of the reservation queue hence he/she can receive reservation. At the same time can ben in the queue of the zone in which currently is located and therefore receive request. If the driver-user is not available he/she can not be part of any queue and therefore can not receive request or reservation.
Exceptions	1. The driver-user is available and press the “available” button. 2. The driver-user is not available and press the “not available” button. In both cases the system block the action. The driver-user remains on the home page.

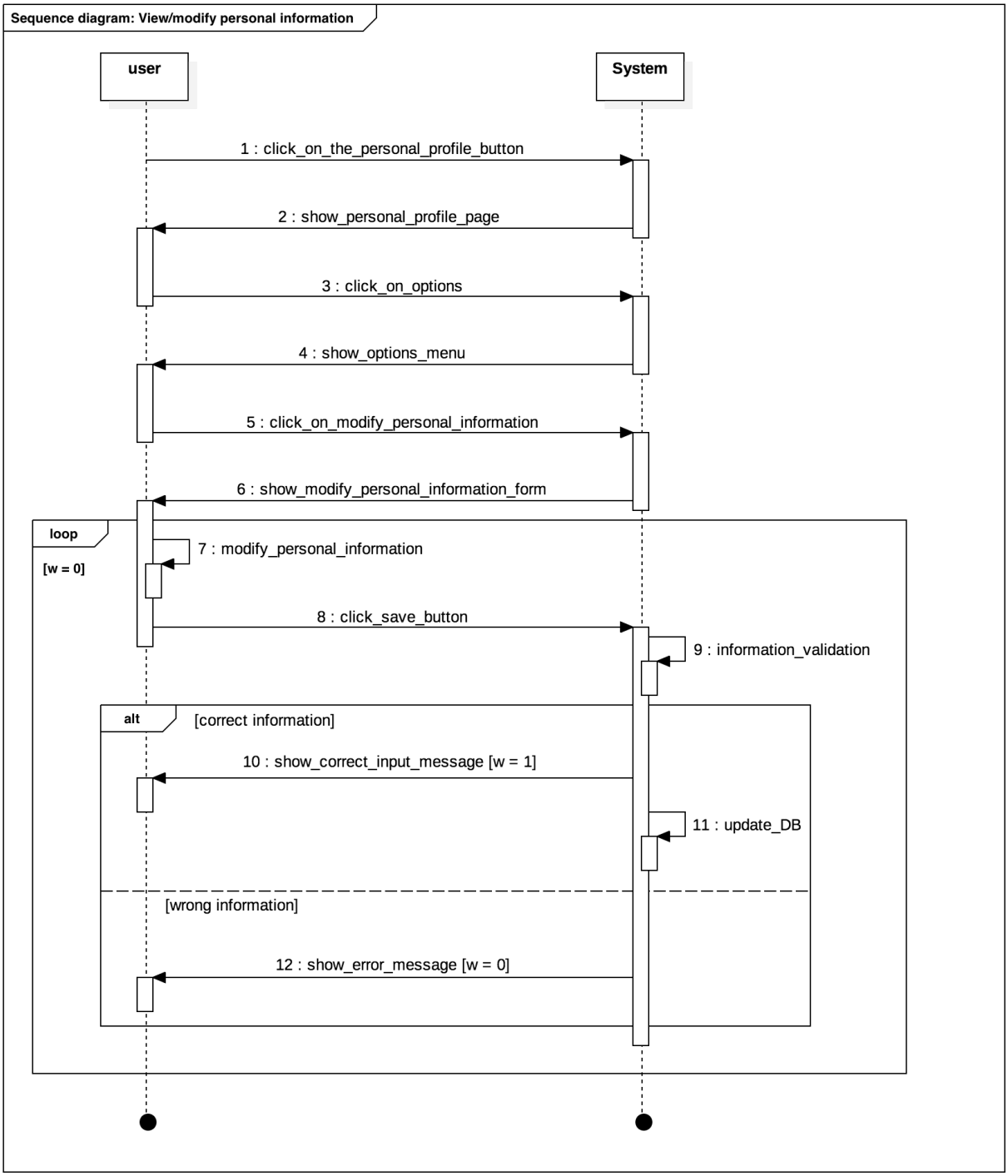

Use Case: Inform the system about his/her availability
Driver-user
Inform the system about availability
«include»
Log in


Sequence diagram: Inform the system about his/her availability
driver-user
System
1 : click_on_the_home_button
2 : show_home_page
alt
[set available]
3 : click_available_button
4 : update_DB
[set not available]
5 : click_not_available_button
6 : update_DB

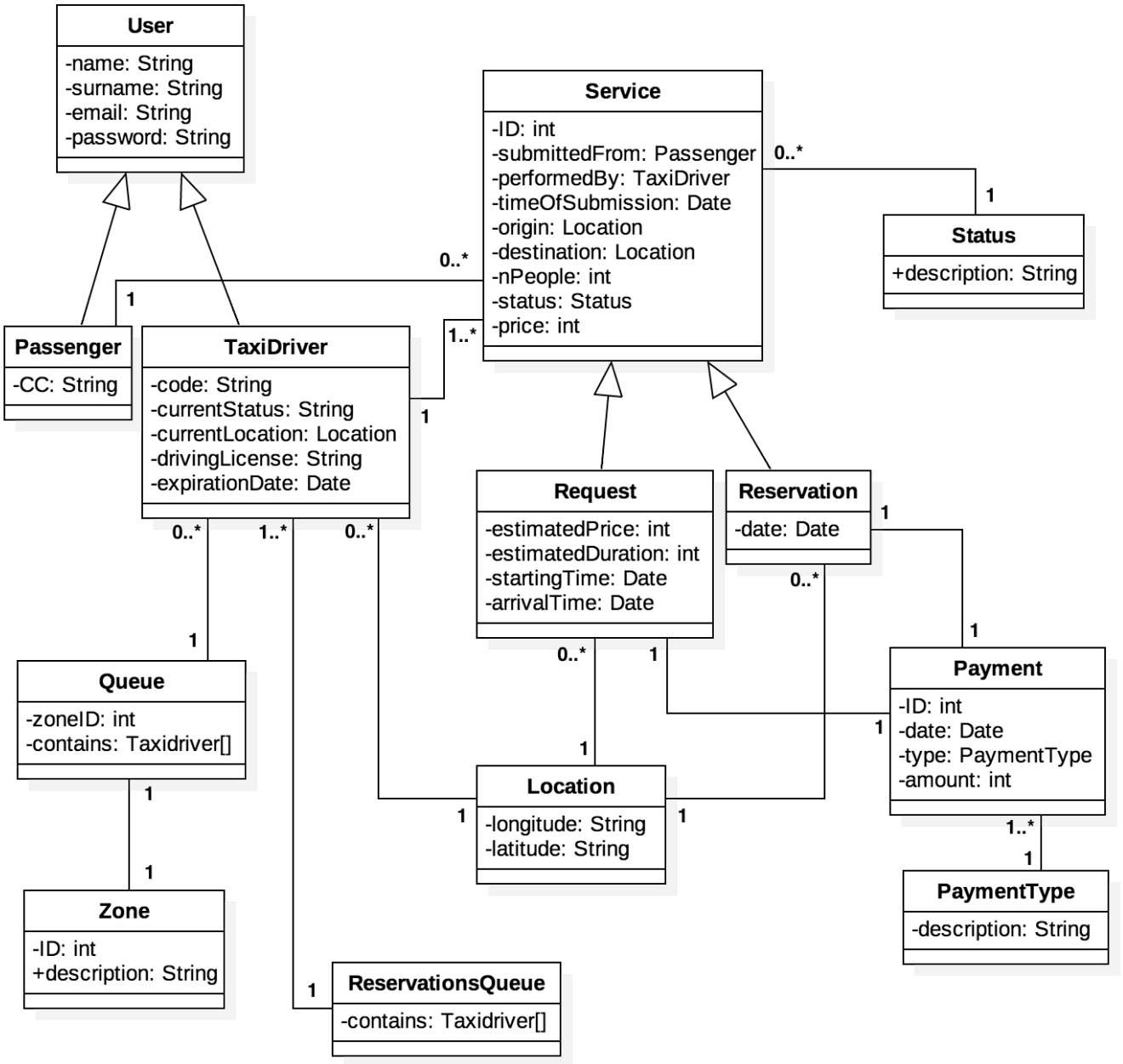
3.2.3.8 View personal information and decide whether to modify them

Name	View personal information and decide eventually to modify them.
Actor	User (passenger-user or driver-user).
Goal	Modify personal information.
Entry conditions	The user is registered to the application.
Flow of events	1. The user clicks on the last button, the one with a person on it. 2. The user clicks on the “options” link. 3. The user clicks on “modify persona information”. 4. The user modify the fields he/she prefers. 5. The user clicks on the “save” button.
Exit conditions	The personal information are changed. The system after a validation of the new information update the DB.
Exceptions	1. At least one of the fields changed contains non valid information. The exceptions is handled informing the passenger about the problem and showing again the form for modifying personal information.

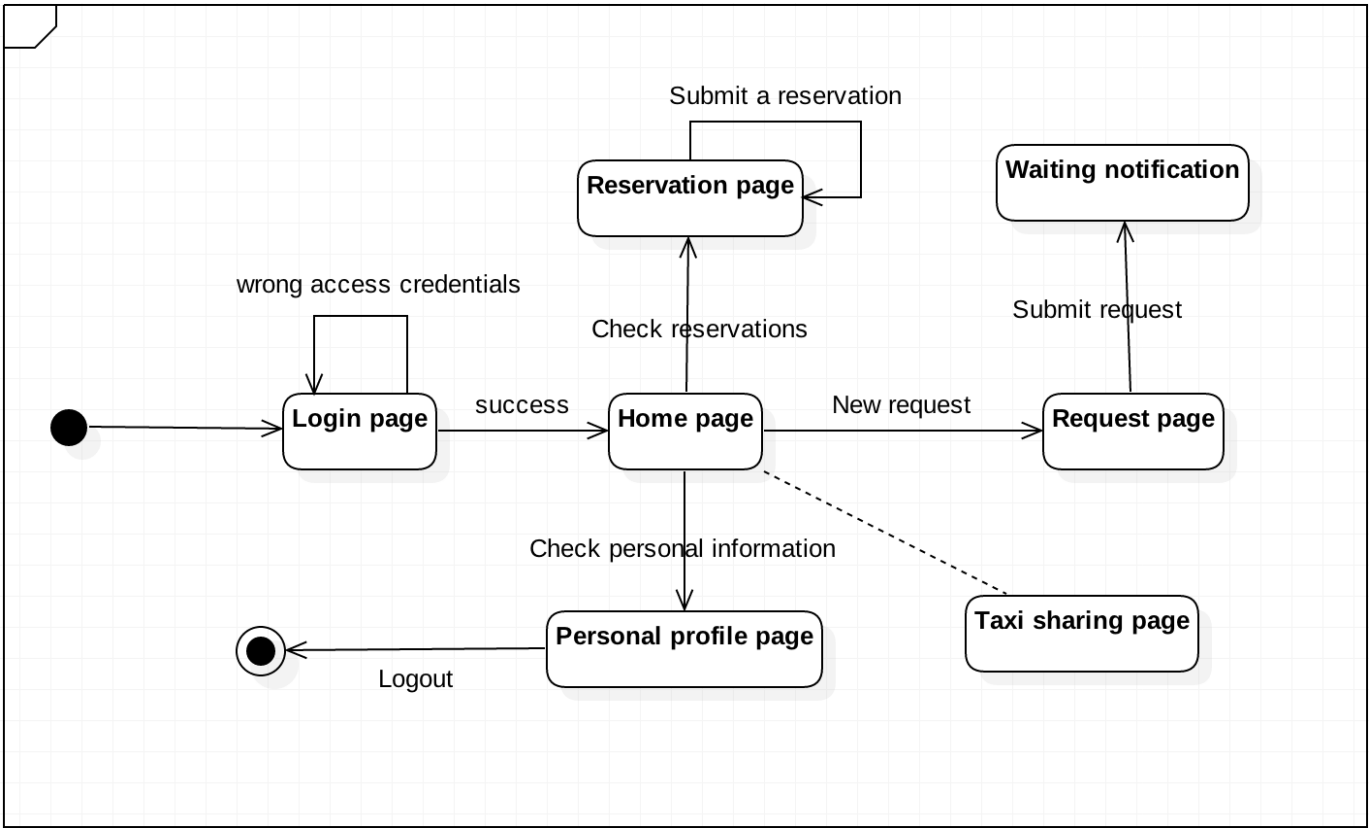

Use Case: View personal information and decide eventually to modify them.
User
View personal profile
«include»
Log in
«include»
Modify personal information
«include»
Verification of the input


Sequence diagram: View/modify personal information
user
System
1 : click_on_the_personal_profile_button
2 : show_personal_profile_page
3 : click_on_options
4 : show_options_menu
5 : click_on_modify_personal_information
6 : show_modify_personal_information_form
loop
[w = 0]
7 : modify_personal_information
8 : click_save_button
9 : information_validation
alt
[correct information]
10 : show_correct_input_message [w = 1]
11 : update_DB
[wrong information]
12 : show_error_message [w = 0]

3.2.4 Class diagram



3.2.5 State machine diagram



3.3 Non functional requirements

3.3.1 Performance Requirements

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

3.3.2 Design Constraints

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

3.3.3 Design Constraints

3.3.3.1 Availability

The wish availability is 100%. A satisfying availability would be 99%. The minimum level acceptable is 98%. The application must be to the internet and to the GPS device.

3.3.3.2 Maintainability

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

3.3.3.3 Portability

The web application will run on the most used browser. The two mobile app will be available for iOS, Android and Windows phone.

3.3.4 Security

MyTaxiService applications implement a login authentication to protect the information of users. Password of user is saved using hashing mechanism but could not be enough since are not available advanced authentication mechanisms.

A lot of improvement could be possible: strangeness requirement, non static passwords, suggest the users of to build secure passwords, captcha test to prevent brute force attacks, https connections and filters against sql injection.

Security can be furtherly improved by the use of firewalls.

4 Appendix

4.1 Alloy

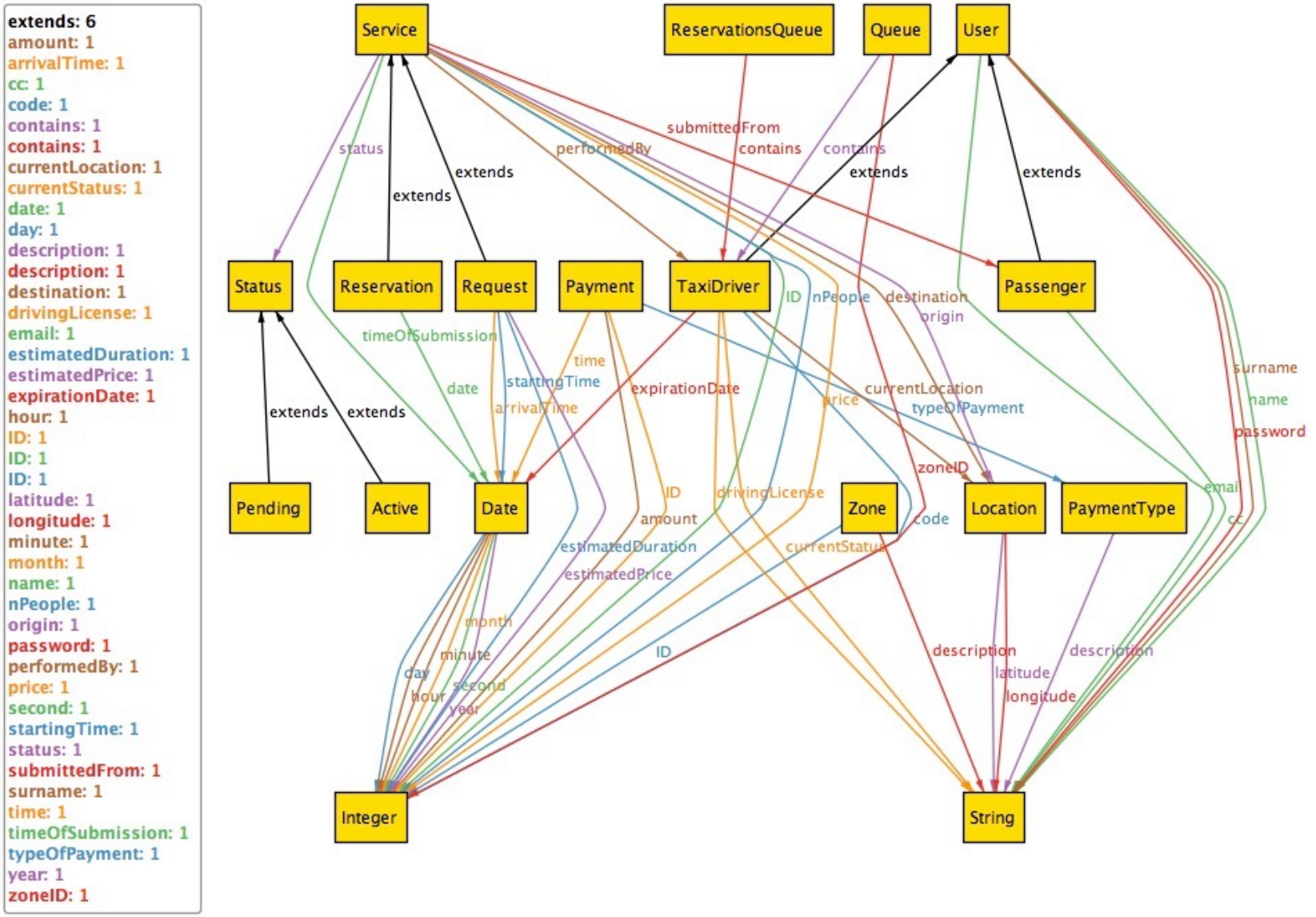
```
sig Integer{}

sig Date{
year: one Integer,
month: one Integer,
day: one Integer,
hour: one Integer,
minute: one Integer,
second: one Integer
}

sig Location{
longitude: one String,
latitude: one String
}

abstract sig User{
name: one String,
surname: one String,
email: one String,
password: one String
}

sig Passenger extends User{
cc: one String
}

sig TaxiDriver extends User{
code: one Integer,
currentStatus: one String,
currentLocation: one Location,
drivingLicense: one String,
expirationDate: one Date
}

abstract sig Status{}
one sig Pending extends Status{}
one sig Active extends Status{}

abstract sig Service{
ID: one Integer,
submittedFrom: one Passenger,
performedBy: one TaxiDriver,
timeOfSubmission: one Date,
origin: one Location,
```

```
destination: one Location,
nPeople: one Integer,
status: one Status,
price: one Integer
}

sig Request extends Service{
estimatedPrice: one Integer,
estimatedDuration: one Integer,
startingTime: one Date,
arrivalTime: one Date,
}

sig Reservation extends Service{
date: one Date
}

sig PaymentType{
description: one String
}

sig Payment{
ID: one Integer,
time: one Date,
amount: one Integer,
typeOfPayment: one PaymentType
}

sig Zone{
ID: one Integer,
description: one String
}

sig Queue{
zoneID: Integer,
contains: set TaxiDriver
}

sig ReservationsQueue{
contains: some TaxiDriver
}

fact noEmptyDate{
all d : Date | (#d.year = 1) and (#d.month = 1) and (#d.day = 1) and (#d.hour = 1)
and (#d.minute = 1) and (#d.second = 1)
```

```
fact noEmptyLocation{
all l : Location | (#l.longitude = 1) and (#l.latitude = 1)
}

fact noEmptyRequest{
all r : Request | (#r.ID = 1) and (#r.submittedFrom = 1) and (#r.performedBy = 1)
and (#r.timeOfSubmission = 1) and (#r.origin = 1) and (#r.destination = 1) and (#r.nPeople = 1)
and (#r.status = 1) and (#r.estimatedPrice = 1) and (#r.estimatedDuration =1)
}

fact noEmptyReservaion{
all r : Reservation | (#r.ID = 1) and (#r.submittedFrom = 1) and (#r.performedBy = 1)
and (#r.timeOfSubmission = 1) and (#r.origin = 1) and (#r.destination = 1) and (#r.nPeople = 1)
and (#r.status = 1) and (#price = 1) and (#r.date = 1)
}

fact noEmptyPayment{
all p : Payment | (#p.ID = 1) and (#p.time = 1) and (#p.amount = 1) and (#p.typeOfPayment = 1)
}

fact noDuplicaetUser{
no disj u1, u2: User | u1.email = u2.email
}

//taxi drivers have different codes
fact driversDifferentCodes{
   no disj d1,d2:TaxiDriver | d1.code = d2.code
}

//Passengers cannot make more than one request at a time
fact noMoreThanOnePendingRequest{
no disj r1, r2: Request | (r1.submittedFrom = r2.submittedFrom) and r1.status=Pending
and r2.status=Pending
}

//Passengers cannot be on two different taxis at the same time
fact noMoreThanOneActiveRequest{
no disj r1, r2: Request | (r1.submittedFrom = r2.submittedFrom) and r1.status=Active
and r2.status=Active
}

//Passengers cannot be on two different taxis at the same time
fact noMoreThanOneActiveReservation{
no disj r1, r2: Reservation | (r1.submittedFrom = r2.submittedFrom) and r1.status=Active
and r2.status=Active
```

```
//A taxi driver cannot take care of two requests at the same time
fact noMoreThanOneActiveRequests{
no disj r1, r2: Request | (r1.performedBy = r2.performedBy) and r1.status=Active
and r2.status=Active
}

//A taxi driver cannot take care of two reservations at the same time
fact noMoreThanOneActiveReservations{
no disj r1, r2: Reservation | (r1.performedBy = r2.performedBy) and r1.status=Active
and r2.status=Active
}

//Each request or reservation has a unique ID
fact twoServicesDifferentID{
no disj s1, s2: Service | (s1!=s2) and (s1.ID=s2.ID)
}

//A taxi driver can be only in one request queue at a time
fact noMoreThanOneQueue{
no t: TaxiDriver | some q1, q2: Queue | q1!=q2 and (t in q1.contains and t in q2.contains)
}

//All taxi driver are always somewhere in the reservations queue
fact allTaxiDriverInReservationsQueue{
all t: TaxiDriver | one q: ReservationsQueue | t in q.contains
}

//If a taxi driver is performing a request he/she is not in a request queue
fact ifDrivingNotInQueue{
all d: TaxiDriver | some r: Request | (r.status=Pending || r.status=Active) && r.performedBy=d
implies no q:Queue | d in q.contains
}

//A request must be ended befor a passenger can make a new one
assert noActiveAndPendingRequests{
no disj r1, r2: Request | (r1.submittedFrom = r2.submittedFrom) and r1.status=Pending
and r2.status=Active
}check noActiveAndPendingRequests

//Show
pred show{
   #Request = 3
   #Reservation = 3
   #TaxiDriver = 3
   #Passenger = 3
}run show for 4
```

```
Executing "Check noActiveAndPendingRequests"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   3490 vars. 327 primary vars. 6337 clauses. 344ms.
   No counterexample found. Assertion may be valid. 30ms.
```


extends: 6
amount: 1
arrivalTime: 1
cc: 1
code: 1
contains: 1
contains: 1
currentLocation: 1
currentStatus: 1
date: 1
day: 1
description: 1
description: 1
destination: 1
drivingLicense: 1
email: 1
estimatedDuration: 1
estimatedPrice: 1
expirationDate: 1
hour: 1
ID: 1
ID: 1
ID: 1
latitude: 1
longitude: 1
minute: 1
month: 1
name: 1
nPeople: 1
origin: 1
password: 1
performedBy: 1
price: 1
second: 1
startingTime: 1
status: 1
submittedFrom: 1
surname: 1
time: 1
timeOfSubmission: 1
typeOfPayment: 1
year: 1
zoneID: 1
Service
ReservationsQueue
Queue
User
Status
Reservation
Request
Payment
TaxiDriver
Passenger
Pending
Active
Date
Zone
Location
PaymentType
Integer
String
status
extends
extends
extends
submittedFrom
performedBy
contains
contains
extends
extends
timeOfSubmission
date
time
startingTime
arrivalTime
expirationDate
ID
nPeople
destination
origin
price
currentLocation
typeOfPayment
zoneID
drivingLicense
amount
estimatedDuration
estimatedPrice
currentStatus
code
ID
month
minute
day
hour
second
year
ID
description
latitude
longitude
description
surname
name
password
email
cc

4.2 Software and tool used

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

4.3 Hours of work

- Azzalini Davide: 60
- Azzalini Fabio: 60