Politecnico di Milano

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL’INFORMAZIONE

Computer Science and Engineering

Software Engineering 2 project

myTaxiService Requirements Analysis and Specification Document

	Andrea Autelitano	Matr. 849869
Prof. Elisabetta Di Nitto	Marco De Cobelli	Matr. 858360
	Matthew Rossi	Matr. 858880

Academic year 2015-2016

Table of contents

1. Introduction

1.1 Purpose

The main goal of this document is to completely describe the software “myTaxiService”, which we are going to develop, laying out its functional and non-functional requirements aspects. It includes the description of the scenarios and use cases and the interactions between the users and the software. Moreover, it establishes the basis for the contract between customers and suppliers.
The audience of this software are Taxi drivers and all the common people that will use the service as Clients.

1.2 Scope

The aim of this project is to optimize the taxi service of a large city thanks to the simplification of the access to the service for the users and the fair management of the taxi queues.
The registered users will be able to request or book a taxi from a mobile application or a web site and Taxi drivers will use the mobile application to accept the requests of the users.
The city is divided into zones of approximately 2km^2, each of these with a different queue of taxi.
The taxi chosen from the system will be chosen from the zone that is nearest to the user that has made the request/reservation and usually it’s his own zone.
Furthermore, when a taxi becomes available, either because its taxi driver logs into the system or the related taxi driver reports the termination of a ride with some clients, its identifier will be added at the end of the queue of the current zone in which he is located.
When a request is received by the first taxi driver in a queue, they can freely choose to accept it or to refuse it. In the first case, it’s notified to the client that one taxi driver has accepted his/her request, in the second case the request is sent to the second taxi driver in the queue and so on.
Moreover, there’s the possibility to reserve a taxi from a certain origin to a specific destination within two hours before the ride and in this case the system will allocate a taxi 10 minutes before the ride to the first taxi driver in the queue.
Finally, besides the traditional interfaces, it’s provided programmatic interfaces to enable the development of additional services like the taxi sharing option. This option allows users to share the cost of a ride with other people after specifying the starting point, the destination of the ride, the number of people to carry on and the availability to share the ride.
In any case, the system automatically calculates the route of the taxi toward the destinations using Google Maps and it provides the fee for each person in the taxi and notifies the users and the Taxi driver who has accepted the ride before the departure.

1.3 Definitions, acronyms, and abbreviations

- CLIENT: a person that uses the app to request rides
- TAXI DRIVER: a person that uses their taxi to provide the ride
- USER: either a Client and a Taxi driver
- GUEST: a person that wants to register to the app or a user before login

1.4 References

- Description of the problem: Assignments 1 and 2 (RASD and DD).pdf
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications
- ISO/IEC 2501:2001 Product Quality Model

1.5 Overview

The contents and the organization of this document is:

- Section 1: Introduction.
  It gives a description of the document and some basic information about the software.
- Section 2: Overall description.
  It gives general information about the software with more focus about constraints and assumptions.
- Section 3: Specific requirements.
  It lists functional and non-functional requirements, scenarios and use cases with the related UML diagrams. It also contains the attached .als file (Alloy code).
- Section 4: Appendix.
  It contains information about the used tools, attached documents and hours of work.
- Section 5: Revision.
  It contains details about the changes done after the first release of this document.

2. Overall description

2.1 Product perspective

The application *myTaxiService* is mainly composed of a web site and a mobile application. Both of them will be used by the Clients that want to request or reserve a taxi, while Taxi drivers use the mobile application to accept or refuse these requests according to their availabilities and the web site to only see their rides.
The mobile application and the web site aren't independent: they will exchange data with another part of the developed system, a database that contains information related to taxi, Taxi drivers, Clients and rides (either concluded and pending).

2.1.1 System interfaces

There are interfaces between web sites, mobile applications and the system in which all the data related to the registered users, taxi and rides are stored.
All these data are used during the normal execution of the pieces of the software we will develop.

2.1.2 User interfaces

There are mainly two kinds of user interfaces.

a. **Taxi driver.** There are four main interfaces for the Taxi driver that allow them to:
   - Login into the application/web site
   - Receive and answer to requests coming from users and accept/refuse them
   - See the history of their concluded rides
   - See the state of the pending ride with the possibility to handle it

b. **Client.** There are four main interfaces for the Client that allow them to:
   - Login into the application/web site
   - Request/Book a taxi by choosing the number of people to carry, where to begin and conclude the ride and the departure time. Moreover, the Client can choose if he/she allows to share the ride with other Clients with a piece of the itinerary in common.
   - See the history of their concluded rides
   - See the state of the pending rides with the possibility to handle them

2.1.3 Hardware interfaces

There are no hardware interfaces.

2.1.4 Software interfaces

There are APIs to allow the access to the GPS and to the mobile applications.

2.1.5 Communications interfaces

There are interfaces that allow the communication between the database of the system and the mobile applications/web site.
Moreover, there are the interface to communicate with the national system in which are stored the information about the Taxi driver licenses (for work as a Taxi driver).
Finally, there are programmatic interfaces to allow the development of additional services.

2.1.6 Memory constraints
The main constraint is related to the availability of memory in the main system that store all the information. Moreover, the mobile phone of the Client and of the Taxi driver has to have enough memory to install the application (only for Clients it’s not mandatory to use it).

2.1.7 Operations
A function for the backup of the data is implemented in the central system in order to guarantee the persistency of data.

2.2 Product functions

Name	**[PF00] Client registration**
Description	The system allows the Client to sign-up inserting email, password, name, surname, gender, address and date of birth.

Name	**[PF01] Taxi driver registration**
Description	The system allows the Taxi driver to sign-up inserting email, password, name, surname, gender, address, date of birth, driving license, taxi license and IBAN.

Name	**[PF02] Login**
Description	The system allows the user to login inserting e-mail and password.

Name	**[PF03] Password recovery**
Description	The system allows the user to recover their account's password, first by identifying the account through the e-mail address and then sending a new password (randomly generated) that will allow the user to access to their account to change the password.

Name	**[PF04] Personal information management**
Description	The system allows the users to modify some of the data previously inserted during the registration phase. The Client can only modify his password, the Taxi driver can also modify the driving license, taxi license and IBAN.

Name	**[PF05] Add payment option**
Description	The system allows the Client to memorize one or more credit cards so that once they want to pay a ride with a credit card, instead of inserting it each time, they only need to select it from an existing list.

Name	**[PF06] Delete payment option**
Description	The system allows the Client to delete a previously inserted credit card.

Name	**[PF07] Add new taxi**
Description	The Taxi driver is able to insert information about the taxis they can use.

Name	**[PF08] Taxi driver working state and taxi choice**
Description	The Taxi driver is able to specify, once they are logged in, if they want to start working using a system option selecting the taxi they are going to use (from those previously inserted) from an existing list. Once they want to stop working, the Taxi driver can go back to the not working state.

Name	**[PF09] Request taxi**
Description	The Client can request a taxi by specifying the place where they want to be picked up through the address (starting position of the ride), the number of people who will take the ride, the destination and if they want to share the ride or not.

Name	**[PF10] Reservation of a taxi**
Description	The Client can reserve a taxi; to do that they have to give information about when and where they wants to have the ride specifying date, time and address, the number of people that will take the ride, the destination and if they want to allow the taxi sharing option or not with someone else.

Name	**[PF11] Zone identification**
Description	The system has to identify the zone in which a user is located once it receives their position.

Name	**[PF12] Queue handling**
Description	The system has to provide a fair queue management assigning at each zone a FIFO (First In First Out) queue of the available taxis.

Name	**[PF13] Taxi driver notification**
Description	The system notifies the Taxi driver selected in the queue and sends them the position of the Client, the destination they want to go to and the number of people for that ride. If the ride is shared with other Clients the system will send to the Taxi driver information about each stop, e.g. the position, number of people that will carry with the taxi and their destination. The Taxi driver can accept or refuse the ride.

Name	**[PF14] Taxi driver answer notification**
Description	The Taxi driver can accept or refuse the ride the system has sent to them.

Name	**[PF15] Compute waiting time**
Description	The system sends a request to the Google Maps servers specifying Client position (starting point) and Client position (destination point) and analyses the answer to extract the waiting time of the fastest path.

Name	**[PF16] Compute ride cost**
Description	The system sends a request to the Google Maps servers specifying the ride origin (starting point) and the ride destination (destination point) and analyses the answer to extract the length of the fastest path. Based on that the system computes preventively the ride cost.

Name	**[PF17] Client notification**
Description	The system confirms to the Client that a taxi has been allocated for their call and sends him/her the Taxi diver’s identification code, the waiting time and the ride cost.

Name	**[PF18] Ride payment**
Description	The Client selects a payment option and pays the Taxi driver the owed amount.

Name	**[PF19] Conclude ride**
Description	The Taxi driver communicates the system the successful conclusion of the ride and the system appends the Taxi driver to the queue of the zone they are in.

Name	**[PF20] List of rides**
Description	The system sends the user the list of rides: - the list of pending rides with details (taxi code, time of reservation, ETA, date, origin, destination, price, n° people and acceptance state) - the list of rides already done with details (taxi code, time of request/reservation, Time of arrival, date, origin, destination, price, n° people) and overall statistics

Name	**[PF21] Modify reservation**
Description	The Client can change the reservation information (date, time, destination and shared).

Name	**[PF22] Cancel reservation**
Description	The Client can cancel the reservation.

Name	**[PF23] Ride cancellation**
Description	The Taxi driver has accepted a ride, but they are unable to satisfy the call of the Client and notifies the system.

Name	**[PF24] Ride interruption**
Description	The Taxi driver can interrupt the ride once they have already picked the Client up because: - The taxi or the Taxi driver has problems while carrying out the ride - The Client wants to stop the ride before reaching the destination (e.g. they are close to the destination, there is huge traffic jam and they can get at the destination faster on foot)

Name	**[PF25] Administration panel**
Description	The administrator can access a dedicated interface where they can see overall statistics about the system, the available taxi distribution, all the rides that has been done and a list of errors that occurred in the system usage.

Name	**[PF26] Tutorial**
Description	To help the user understand how the web site and mobile app work, at the first login a tutorial is shown. It will summarize system functionalities and how to access them.

Name	**[PF27] Zone change**
Description	When a Taxi driver changes their zone the mobile application installed on the Taxi driver’s mobile phone communicates the event to the system.

2.3 User characteristics

We can identify two main categories of users: Taxi drivers and Clients.
However, both categories are characterized by heterogeneous levels of education, experience and technical expertise, thus both the web application and the mobile app should satisfy the "Usability" requirement, as specified in section 3.7.4.

2.4 Constraints

The service is subjected to the following constraints:

a. **Regulatory policies**
   HTTP cookies are used both on the web site and the mobile application: therefore, the user must be informed (e.g. using a popup message), in compliance with EU Privacy directives.
   Taxi fares calculated by the system must not exceed the limit determined by the National Fares Association.
b. **Interfaces to other applications**
   The system must be able to interact with the pre-existing central database containing information about Taxi drivers and IDs.
   The system must use the Google Maps APIs to calculate the taxis distribution in each area.
c. **Parallel operations**
   The system must support concurrent access by different users.
d. **Safety and security considerations**
   Sensitive data from both Clients and Taxi drivers are stored in the central database. Therefore, these data must not be accessible from external subjects (e.g. advertising agencies).

2.5 Assumptions and dependencies

- There is a pre-existent system that contains information about Taxi drivers' taxi license. This system is used to verify, during the registration phase, whether the Taxi driver has inserted correct information.
- If a Taxi driver does not answer to a ride notification in 30 seconds they cannot take care of the ride at the moment. This is done to guarantee that a problem on a Taxi driver does not affect the whole system quality by introducing excessive delays and thus reduce the satisfaction of the Clients.
- The fee is calculated by means of the kilometers of the ride. This is done so that the system can pre-calculate the cost of the ride and send it to the Taxi driver when they are reachable. In this way even if the Taxi driver loses GPS signal and/or data connection they can still terminate the ride.
- The Client is sure about the destination they want to reach when they do a standard request (they don't change it).
- Clients and Taxi drivers are disjointed (if a Taxi driver wants to use the system as a Client too he has to sign up with a Client account).
- In each taxi at least one Client has completed the request/reservation. This means that if a group of more than 4 people want to share a ride they have to split it into more requests/reservations.
- If a Taxi driver refuses 3 consecutive calls they don't want to work (the Taxi driver is removed from any queue).
- If a Taxi driver refuses 5 consecutive calls they are trying to interfere with the service quality and they are suspended for 3 days where 3 increase each time this situation occurs.

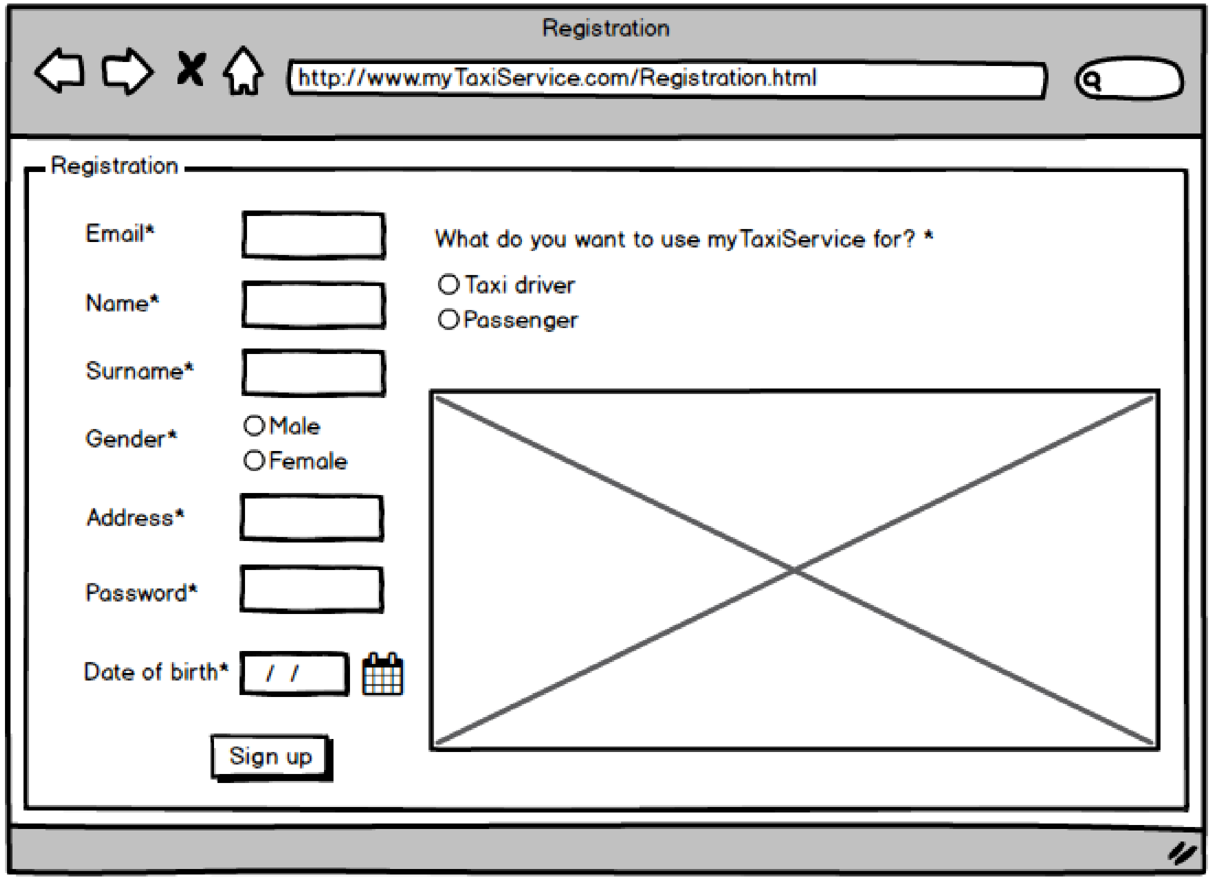
- If the Taxi driver logs out or changes their working state when they have already accepted a request/reservation for a ride they will not get the Client anymore
- Each reservation is allocated to a Taxi driver 10 minutes before the departure time (the Client cannot delete/modify the reservation anymore after this period of time).

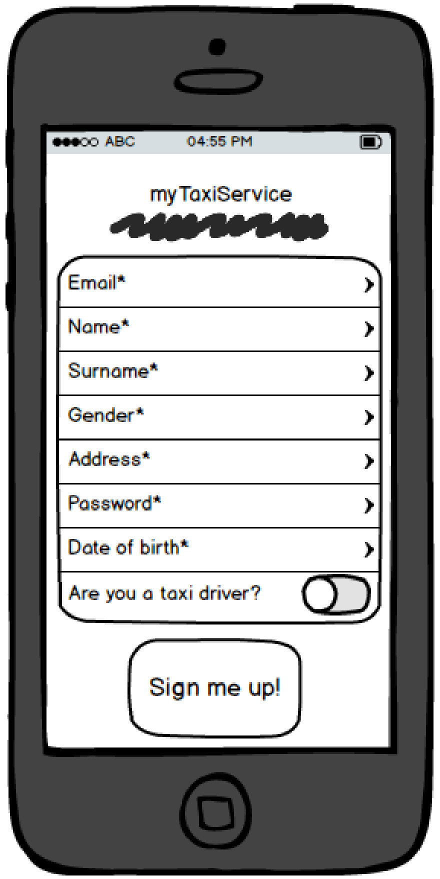
3. Specific requirements

3.1 External interfaces

Registration page

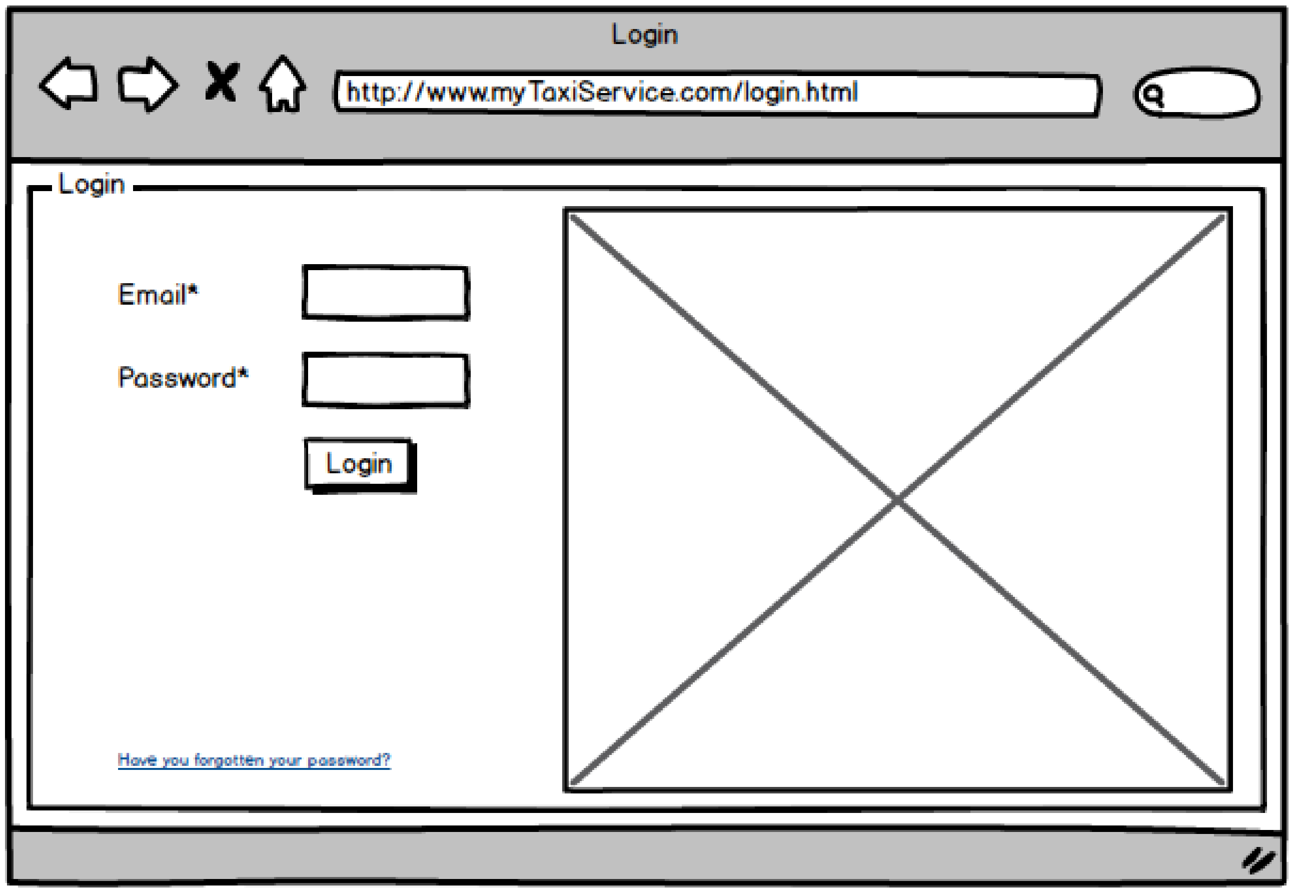
The following images show the registration pages available for a Guest from the web site or from the mobile application.

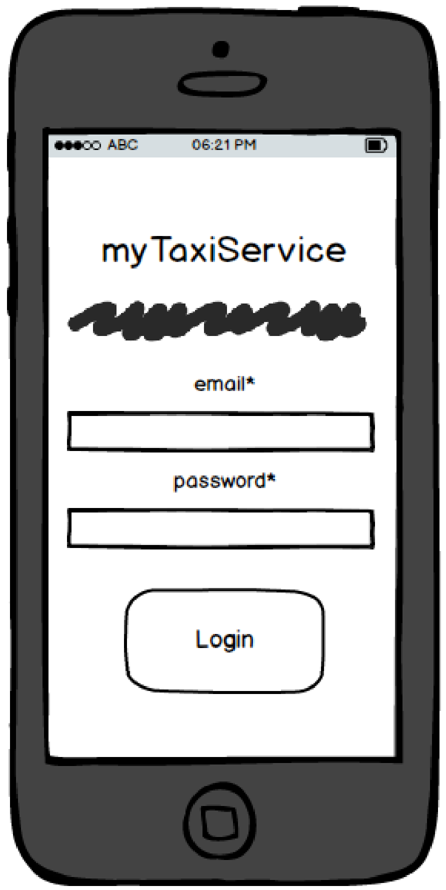




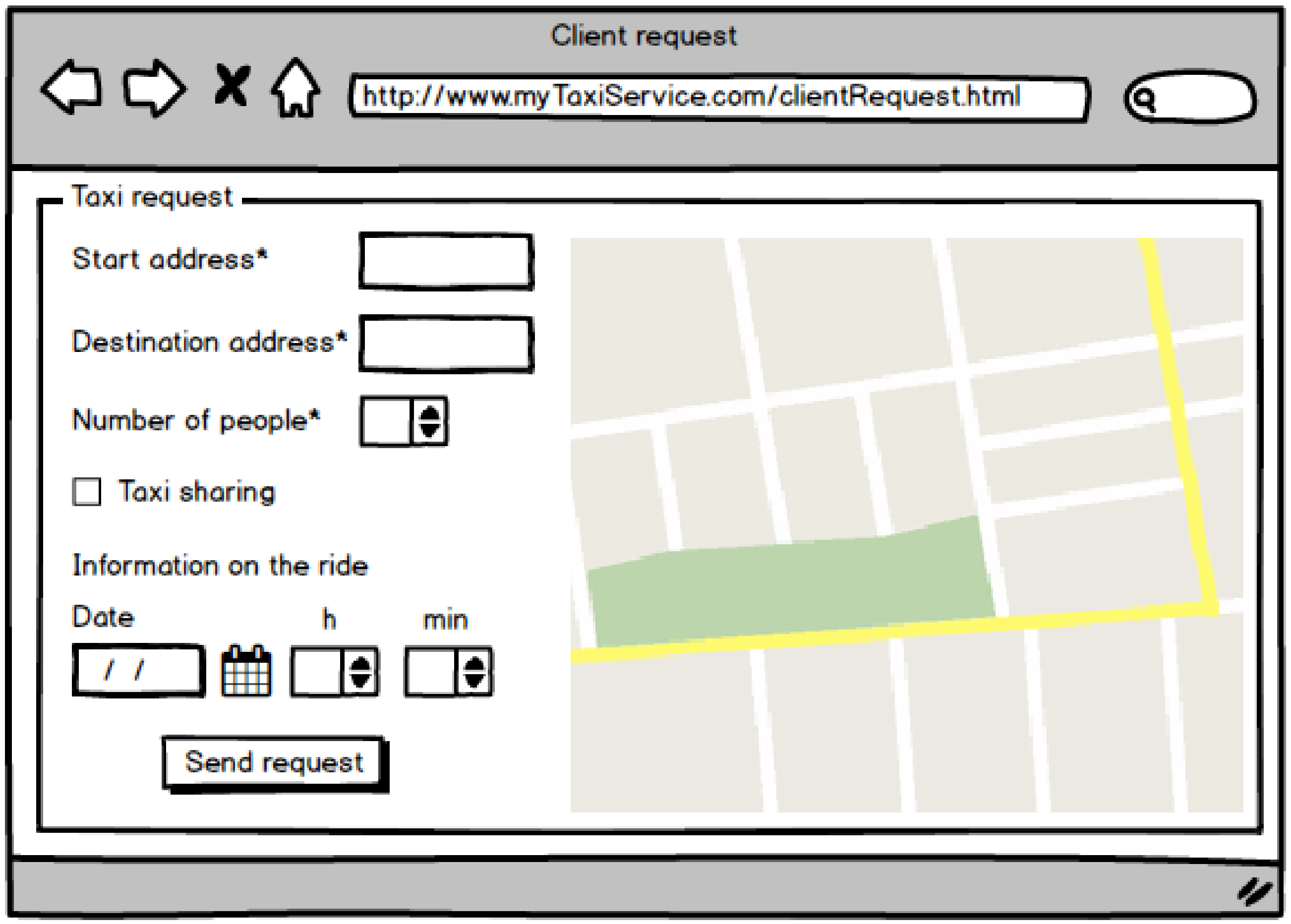
Login page

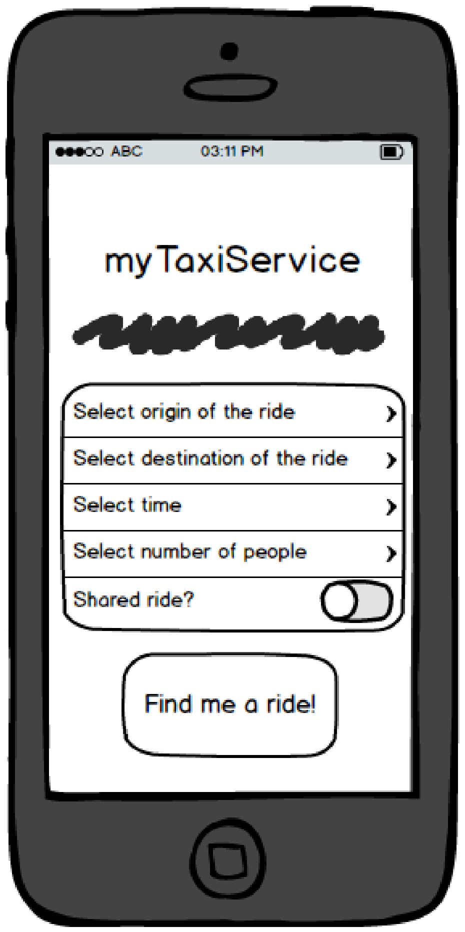
The following images show the login pages available for a Guest from the web site or from the mobile application.


Login
http://www.myTaxiService.com/login.html
Login
Email*
Password*
Login
Have you forgotten your password?



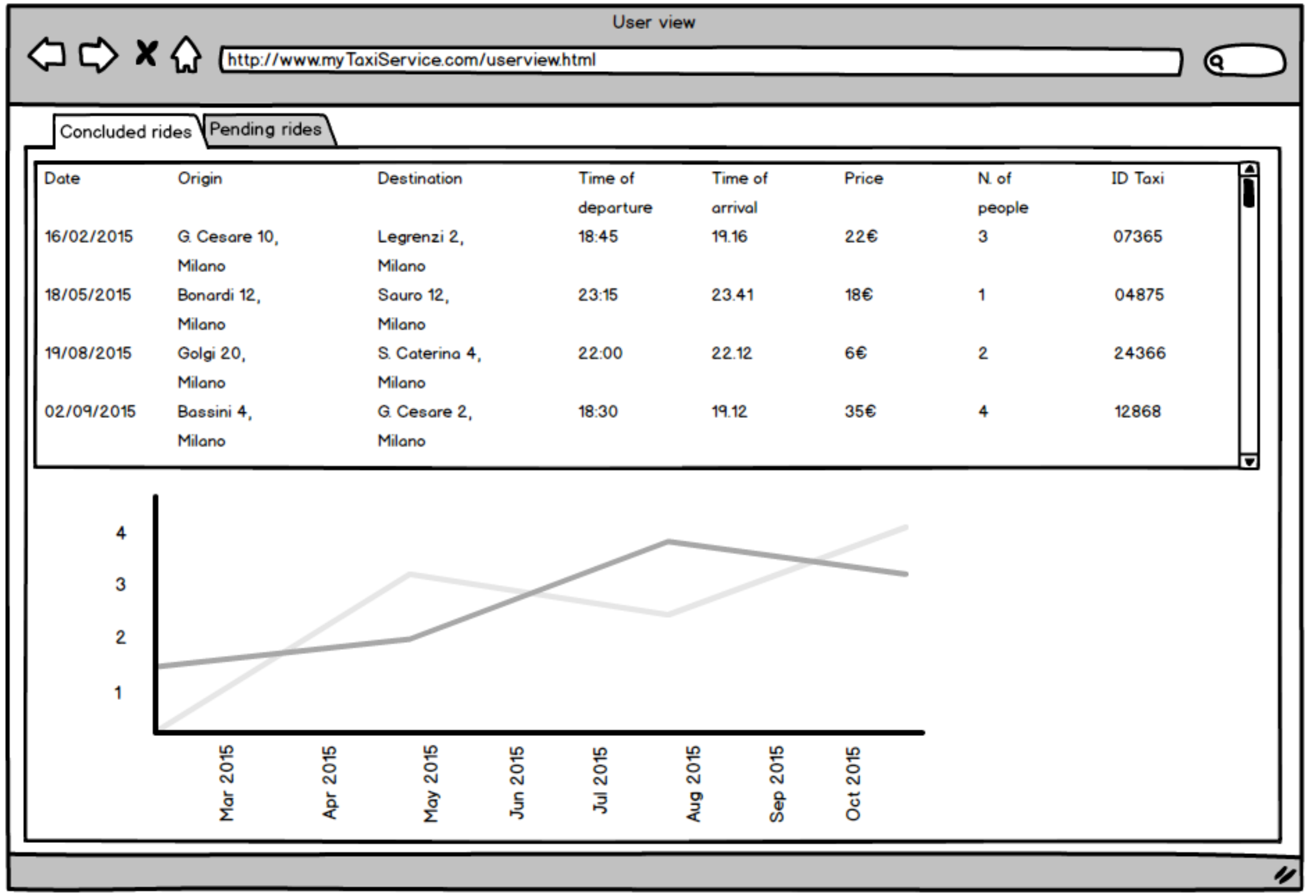
Ride request
The following images show the pages that allow a Client to request a ride right now or to book a new one.



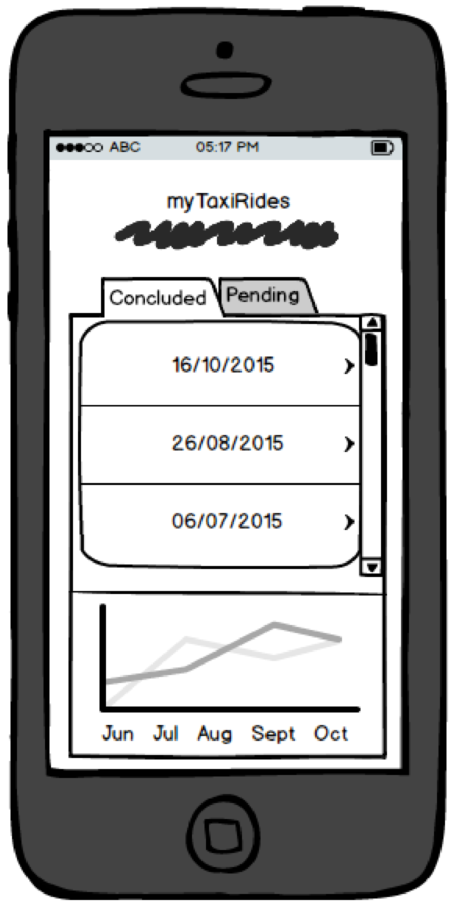


User view – Concluded rides

The following images show the pages that allow a user to see the list of their concluded rides with all the related information and the possibility to see a graph with the number of their concluded rides grouped by a certain period of time.



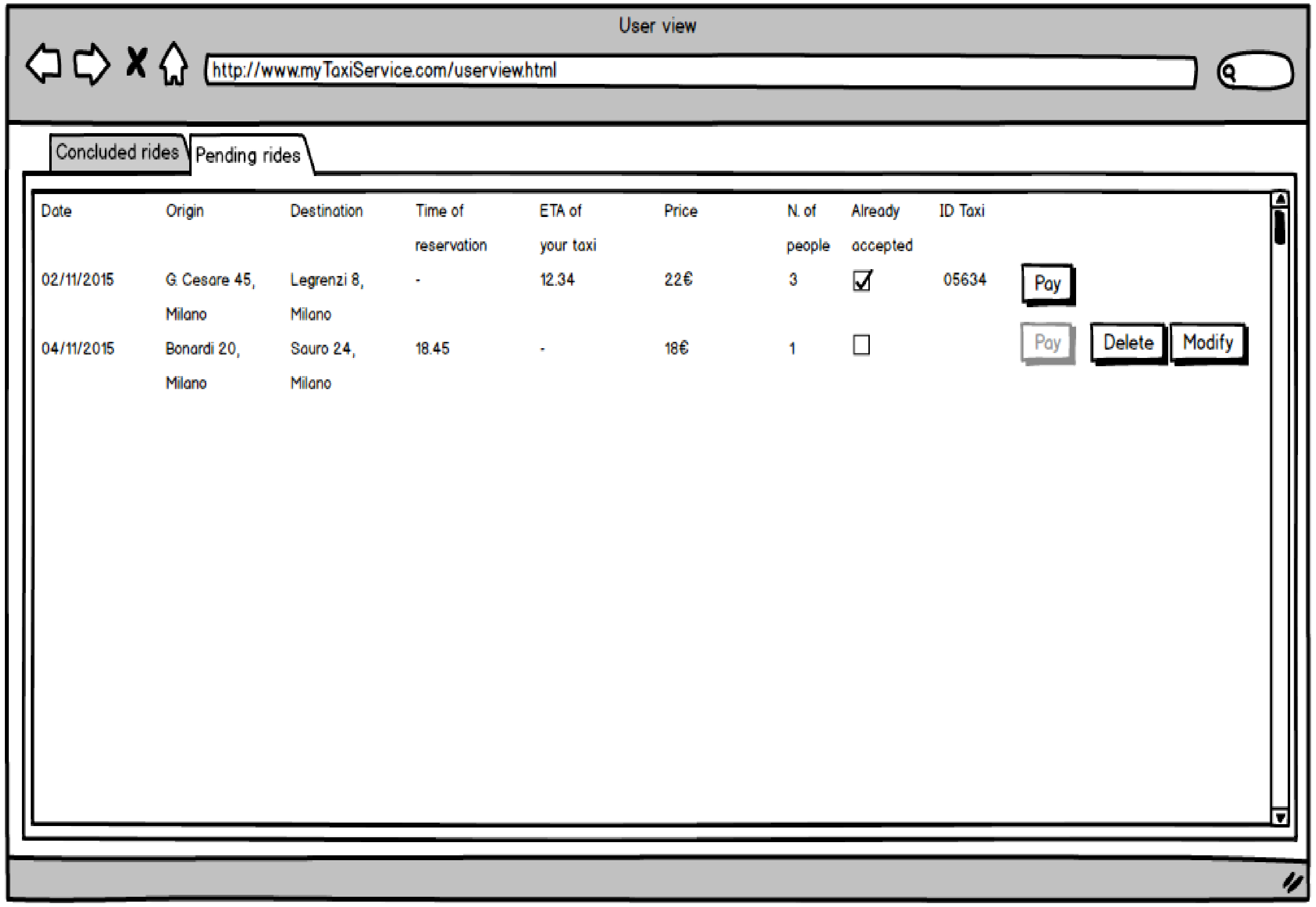
Date	Origin	Destination	Time of departure	Time of arrival	Price	N. of people	ID Taxi
16/02/2015	G. Cesare 10, Milano	Legrenzi 2, Milano	18:45	19.16	22€	3	07365
18/05/2015	Bonardi 12, Milano	Sauro 12, Milano	23:15	23.41	18€	1	04875
19/08/2015	Golgi 20, Milano	S. Caterina 4, Milano	22:00	22.12	6€	2	24366
02/09/2015	Bassini 4, Milano	G. Cesare 2, Milano	18:30	19.12	35€	4	12868



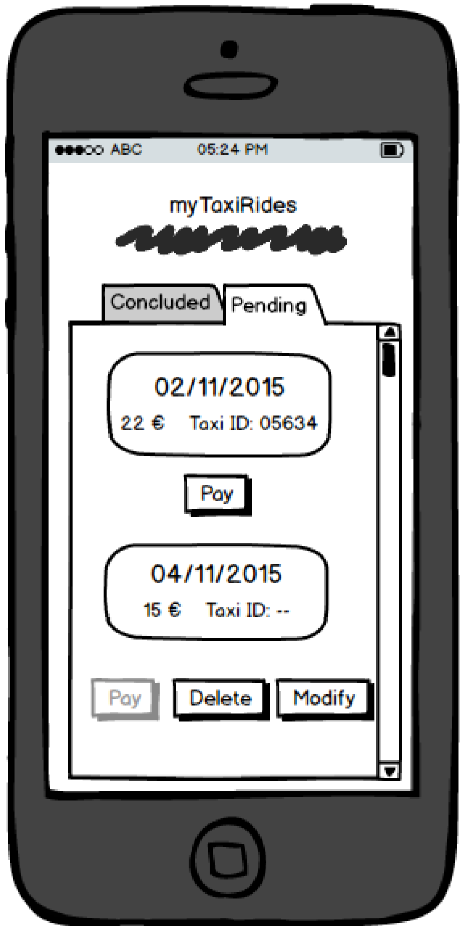
User view – Pending rides

The following images show the pages that allow a Client to see the list of their pending rides with all the related information and the possibility to delete, modify and pay with a credit card.

In the example below you can see the buttons Delete and Modify only related to the booked ride because the others cannot be modified/canceled. Moreover, the button Pay is disabled for the ride booked because is not already accepted from a Taxi driver.



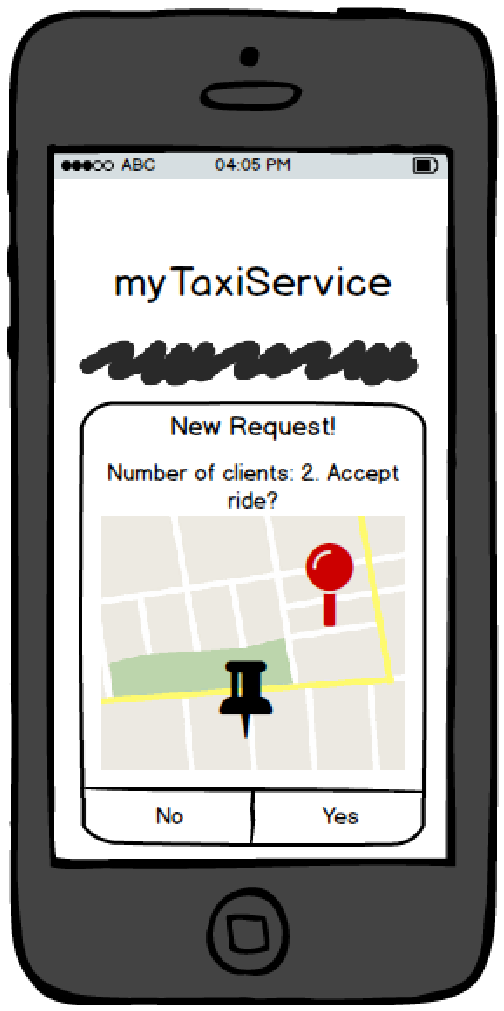
Date	Origin	Destination	Time of reservation	ETA of your taxi	Price	N. of people	Already accepted	ID Taxi	
02/11/2015	G. Cesare 45, Milano	Legrenzi 8, Milano	-	12.34	22€	3	☑	05634	Pay
04/11/2015	Bonardi 20, Milano	Sauro 24, Milano	18.45	-	18€	1	☐		Pay Delete Modify



02/11/2015
22 € Taxi ID: 05634
Pay

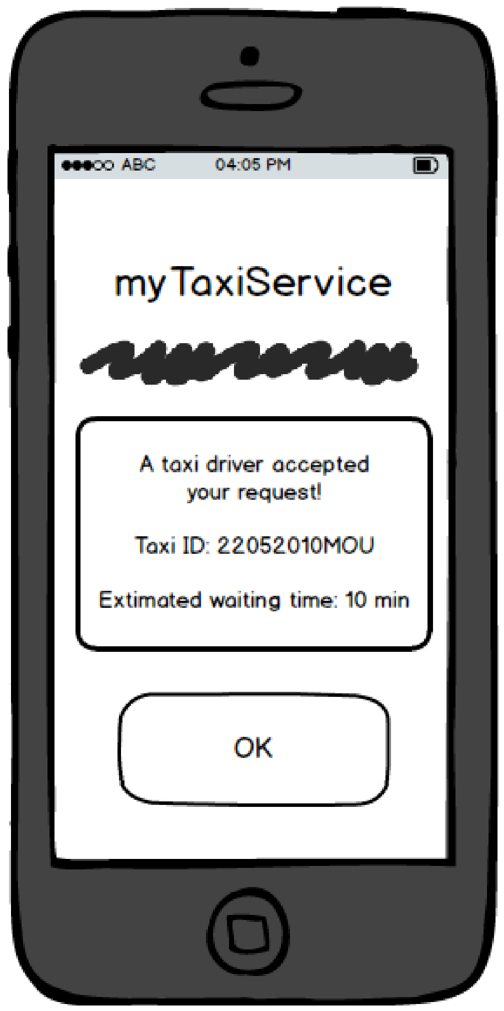
04/11/2015
15 € Taxi ID: --
Pay Delete Modify

Taxi request
The following image shows the notification received by the Taxi driver allowing them to accept/refuse a ride.



Number of clients: 2. Accept ride?

Accepted request
The following image shows the notification received by the Client after a Taxi driver has accepted their taxi request.



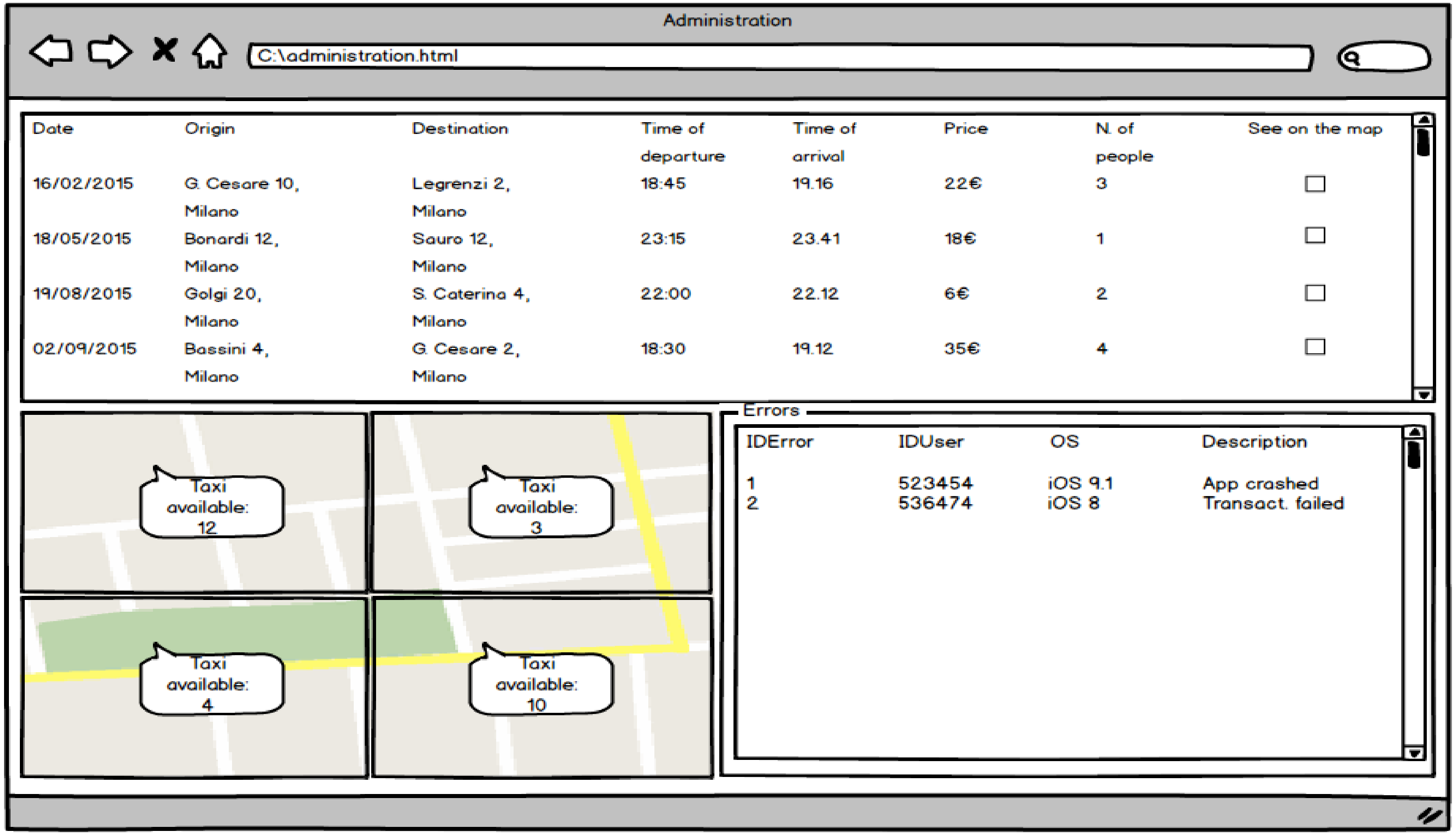
A taxi driver accepted your request!

Taxi ID: 22052010MOU

Extimated waiting time: 10 min

Administration panel

The following image shows the Administration panel (available only for the administrators of the system) with data grouped by tipe of information.



Administration

C:\administration.html

Date	Origin	Destination	Time of departure	Time of arrival	Price	N. of people	See on the map
16/02/2015	G. Cesare 10, Milano	Legrenzi 2, Milano	18:45	19.16	22€	3	☐
18/05/2015	Bonardi 12, Milano	Sauro 12, Milano	23:15	23.41	18€	1	☐
19/08/2015	Golgi 20, Milano	S. Caterina 4, Milano	22:00	22.12	6€	2	☐
02/09/2015	Bassini 4, Milano	G. Cesare 2, Milano	18:30	19.12	35€	4	☐

Taxi available: 12

Taxi available: 3

Taxi available: 4

Taxi available: 10

Errors

IDError	IDUser	OS	Description
1	523454	iOS 9.1	App crashed
2	536474	iOS 8	Transact. failed

3.2 Functions

Name	**[PF00] Client registration**
Requirements	[R0] The Guest must not be already registered [R1] The Guest must insert valid data according to the format of each field [R2] The Guest must choose an email that exists and is not already being used by another user [R3] The user must be able to access their email

Name	**[PF01] Taxi driver registration**
Requirements	[R0] The Guest must not be already registered [R1] The Guest must insert valid data according to the format of each field [R2] The Guest must choose an email, a driving license, a taxi license and an IBAN that exists and is not already used by another user [R3] The user must be able to access their email

Name	**[PF02] Login**
Requirements	[R0] The user must be already signed up [R1] The user must insert valid data according to the format of each field [R2] The user must know their email and password to successfully accomplish the login

Name	**[PF03] Password recovery**
Requirements	[R0] The user must be already signed up [R1] The user must insert a valid email according to its format [R2] The user must know their email to successfully retrieve the password [R3] The user must be able to access their email

Name	**[PF04] Personal information management**
Requirements	[R0] The user must be already logged in [R1] The user must insert valid data according to the format of each field [R2] The user must know their password to successfully change it

Name	**[PF05] Add payment option**
Requirements	[R0] The Client must be already logged in [R1] The Client must insert valid data according to the format of each field [R2] The Client must remember their credit card information

Name	**[PF06] Delete payment option**
Requirements	[R0] The Client must be already logged in

Name	**[PF07] Add new taxi**
Requirements	[R0] The Taxi driver must be already logged in [R1] The Taxi driver must insert valid data according to the format of each field [R2] The Taxi driver must remember their taxi information.

Name	**[PF08] Taxi driver working state and taxi choice**
Requirements	[R0] The Taxi driver must be already logged in [R1] The Taxi driver must know if their want to work [R2] The Taxi driver choose the taxi he/she is going to use [R3] The Taxi driver GPS has to work

Name	**[PF09] Request taxi**
Requirements	[R0] The Client must be already logged in [R1] The Client must insert valid data according to the format of each field [R2] The number of people inserted by the Client has to be smaller or equal than 4

Name	**[PF10] Reservation of a taxi**
Requirements	[R0] The Client must be already logged in [R1] The Client must insert valid data according to the format of each field [R2] The number of people inserted by the Client has to be smaller or equal than 4

Name	**[PF11] Zone identification**
Requirements	[R0] The position to analyze must be inside a defined zone

Name	**[PF12] Queue handling**
Requirements	[R0] At least one Taxi driver has to be available in the whole system if the function is used to assign the taxi to a ride

Name	**[PF13] Taxi driver notification**
Requirements	[R0] The Taxi driver must be already logged in and working [R1] A ride has been requested (or reserved and it has to be allocated to a taxi) by a Client [R2] At least a Taxi driver has to be available in the whole system [R3] The ride cost has been calculated [R4] The Taxi driver is connected at a data network

Name	**[PF14] Taxi driver answer notification**
Requirements	[R0] The Taxi driver must be already logged in and working [R1] The Taxi driver has been notified [R2] The Taxi driver is connected at a data network

Name	**[PF15] Compute waiting time**
Requirements	[R0] The Taxi driver has accepted a ride [R1] The Taxi driver position is known [R2] Google Maps servers are reachable

Name	**[PF16] Compute ride cost**
Requirements	[R0] The Taxi driver has accepted a ride [R1] Google Maps servers are reachable

Name	**[PF17] Client notification**
Requirements	[R0] The Client must be already logged in [R1] The Taxi driver has accepted the Client’s ride [R2] The Client has access at a data network

Name	**[PF18] Ride payment**
Requirements	[R0] The Client must be already logged in [R1] The Taxi driver has accepted the Client’s ride [R2] The Client is connected at a data network

Name	**[PF19] Conclude ride**
Requirements	[R0] The Taxi driver must be already logged in and working [R1] The Taxi driver has accepted the Client’s ride [R2] The Taxi driver has access at a data network [R3] The Client payed with cash/credit card

Name	**[PF20] List of rides**
Requirements	[R0] The user must be already logged in [R1] The user has done at least one ride

Name	**[PF21] Modify reservation**
Requirements	[R0] The Client must be already logged in [R1] The Client has at least one pending reservation [R2] The Client must insert valid data according to the format of each field [R3] At least 10 minutes are left from the starting time of the selected ride

Name	**[PF22] Cancel reservation**
Requirements	[R0] The Client must be already logged in [R1] The Client has at least one pending reservation [R2] At least 10 minutes left from the ride starting time selected by the Client

Name	**[PF23] Ride cancellation**
Requirements	[R0] The Taxi driver has accepted a Client’s ride [R1] The Taxi driver has not already picked up the Client [R2] The Taxi driver is connected at a data network

Name	**[PF24] Ride interruption**
Requirements	[R0] The Taxi driver has accepted a Client’s ride [R1] The Taxi driver has already picked up the Client [R2] The Taxi driver is connected at a data network [R3] Both the Taxi driver and the Client have agreed to terminate the ride before reaching the destination

Name	**[PF25] Administration panel**
Requirements	[R0] The administrator has logged into the server side application

Name	**[PF26] Tutorial**
Requirements	[R0] The user must be already logged in [R1] It’s the first user log in

Name	**[PF27] Zone change**
Requirements	[R0] The Taxi driver must be already logged in and working [R1] The Taxi driver has no ride in progress [R2] The Taxi driver has access at a data network [R3] The Taxi driver location is known [R4] The Taxi driver has changed zone

3.3 Scenarios

Title	**[S01] Requesting a ride**
Related use case	[UC04]
Description	Angelo decides to go seeing his favourite band's concert at Alcatraz, but it is late and no buses can take him there, so he decides to take a taxi. He takes his mobile phone, opens the myTaxiService app, then clicks on the "Find me a ride" button. He selects "Actual position" as origin of the ride, "Via Valtellina 25" as destination and "Right now" as time, clicks "Find me a ride" button and then he waits. After about a minute a notification pops up: Taxi driver 25052005LIV will come to pick him up in 7 minutes. He closes the app and waits patiently for the taxi to arrive.

Title	**[S02] Reserving a ride**
Related use case	[UC05]
Description	It is October 30th and on the 31st at 20.45 there's an important football match between Stefano's supported team and the team that is first of the table. He's already got the tickets, but he doesn't know yet how to go to the stadium. His friend Andrea recommends him to take a taxi and book it on a new website, www.mytaxiservice.com. He likes the idea: he opens the website, signs up and then clicks on "Find me a ride". He selects "Via Golgi 42" as origin of the ride, "Piazzale Angelo Moratti" as destination and 20.10 as time, then clicks on "Send request". A message pops up saying that he has successfully reserved a ride and that he will be notified at 20.00 about his taxi details. He is satisfied by the service and decides to download the app to be sure that he will not miss the notification.

Title	**[S03] Shared ride**
Related use case	[UC06]
Description	Lucia has just landed at Malpensa Airport and wants to meet her friend Renzo, who lives in Corso Vittorio Emanuele. She decides to take a taxi, but she only has 10 € in her wallet. Then she remembers that her friend Agnese told her about the possibility to share rides with other people using the app “myTaxiService”. So she opens the app, clicks on the “Find me a ride” button, then selects “Actual position" as origin of the ride, "Corso Vittorio Emanuele 48" as destination, "Right now" as time and choses the “Shared ride” option, then she clicks "Find me a ride" button and waits. After about two minutes she receives a notification from the app that says that taxi 06062015BAR will arrive in 3 minutes and she will share the ride with two other people: they will spend 8.50 € each.

Title	**[S04] Payment via Credit Card**
Related use case	[UC07]
Description	Maria is waiting for the taxi that she booked using the app “myTaxiService” and that will come to pick her up in fifteen minutes. She decides to pay in advance, so that she will not waste her time when she has to get off the taxi. She opens the app, clicks on “myTaxiRides”, selects the “Pending rides” tab and then clicks on the “Pay” button. She realizes that she has never paid a taxi ride using a Credit Card, so she decides to link her PayPal card to her account: she selects “PayPal”, then waits while she is being redirected to the company’s website. She inserts her details then proceeds to confirm the payment, but she gets a notification: something went wrong during the transaction. She clicks on “Try again” and checks if she wrote her personal detail right. She realizes that she misspelled her name, so she corrects it and confirm the payment. Then she is redirected to the app, and a notification pops up saying that the transaction was successful. Meanwhile Giuseppe, the Taxi driver, receives a notification on his mobile phone: the girl that he is picking up, Maria, has already paid the ride using her Credit Card.

Title	**[S05] Ride in process**
Related use case	[UC08]
Description	Marco is headed to his destination into a taxi. When he arrives to the destination he has to pay and chooses to pay with credit card. Thus, he opens his app and he goes into “myTaxiRides” and then clicks “Pending rides”. At the line of the current ride he finds a button called “Pay” and after clicking it he’s redirected to a page where he chooses from one of his credit cards inserted before or he can insert a new credit card. He decides to choose a credit card previously inserted and then he inserts the CVV of the card. The transaction is confirmed from his credit card’s company. Then the Taxi driver confirms into his app that the ride is to be considered concluded.

Title	**[S06] Ride in process**
Related use case	[UC08]
Description	Roberto is going to work with a taxi. Before arriving to his destination he sees that there’s a lot of traffic on his roads and decides to go on foot from that position. In that case the Taxi driver inserts the current destination into the section of the current rides, after clicking “myTaxiRides” and then “Pending rides”. Consequently, the price is re-calculated and the Client can see it into his app clicking on “myTaxiRides” and then “Pending rides”. He is in a hurry and pays the Taxi driver with cash. The Taxi driver concludes the ride into his app clicking on “Conclude ride” related to that ride.

Title	**[S07] Ride cancelation**
Related use case	[UC09]
Description	Martina made a reservation of a taxi for tonight but this morning she remembered that a friend of her could bring her to the party she wants to go. Then, she decides to call her friend and asking to her if she really could bring her to her destination and her friend confirms so. Thus, Martina decides to open the web site www.myTaxiService.com and she logs into her account with her credentials. She clicks on “myTaxiRides” and then on “Pending rides”. Now she can see the list of all her pending rides and she clicks on the “Delete” button related to the reservation of tonight. The system checks if the ride can be canceled and notifies her of the correct cancelation because at that moment it’s missing 6 hours and 32 minutes to the ride.

Title	**[S08] Ride modification**
Related use case	[UC10]
Description	This morning Paolo made a reservation for a taxi for 8.30pm of today because he has to go to the airport to leave for his holidays. At 7.30pm he receives a notification from the airline company that reports him that his flight is been deleted and postponed to tomorrow morning at 10am. Hence, Paolo decides to modify his reservation and he opens his myTaxiService app, he logs with his account and he clicks on “myTaxiRides” and then on “Pending rides”. Now he can see the list of his pending rides and he chooses the modify option related to the ride reserved for 8.30pm of today by clicking on the button “Modify”. Finally, he can choose the date and time of the new reservation, then he clicks OK. Because there is more than 10 minutes left before the hour of the reservation, the system notifies Paolo of the correct modification of the reservation.

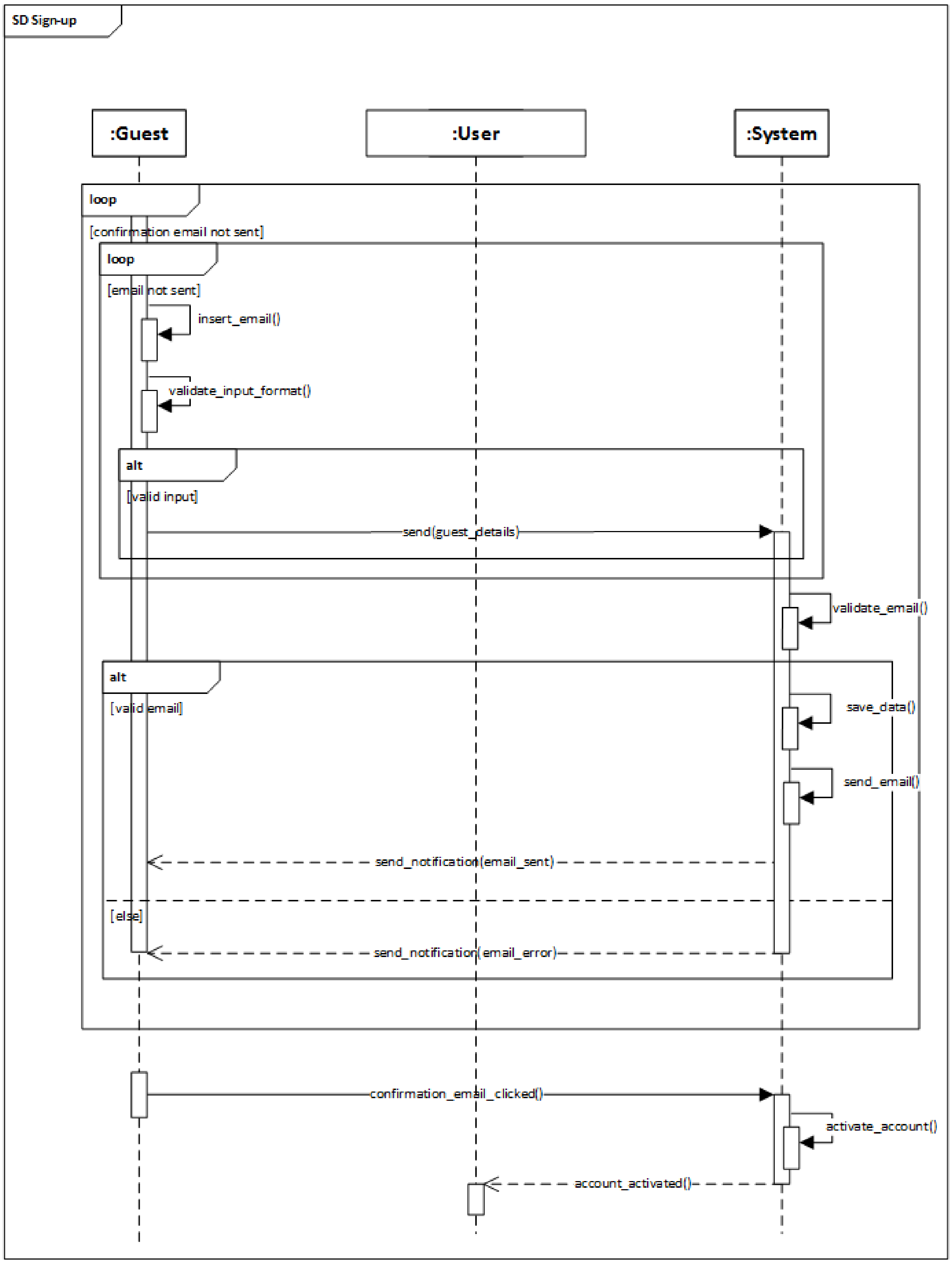
Title	**[S09] Concluded/Pending ride details**
Related use case	[UC11]
Description	Luca is doing a review of his personal calendar of the previous months and he sees that now he’s taking the taxi less and less comparing to the last months, so he has an idea: go into his app myTaxiRides app and have a confirmation of his thoughts. He logs with his credentials into his app and goes to the section “myTaxiRides” and then into the subsection “Concluded rides”. The system provides him a complete list of all his concluded rides and he can see the information related to them grouped by month in a graph and also listed with information like the number of people in the rides, date and time.

3.4 Use cases

3.4.1 Sign-up

This use case describes the process of registration of a Guest.

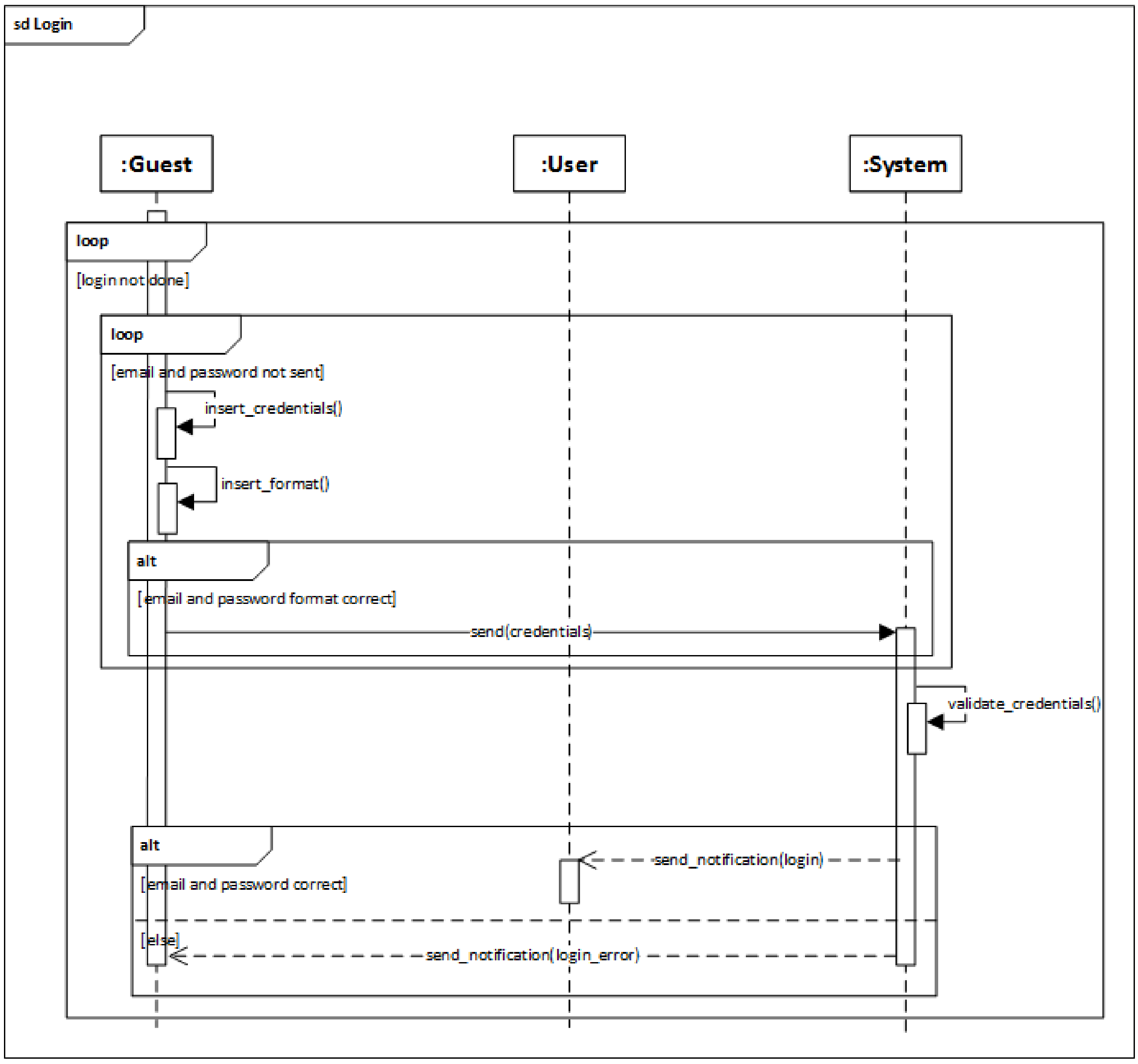
Name of the use case	**[UC01] Sign-up**
Actors	Guest
Entry condition	NULL
Flow of events	1. The Guest clicks on “Sign-up” 2. The Guest decides which kind of user they want to be selecting “Taxi driver” or “Client” (based on this choice different form are shown 3. The Guest fills at least all mandatory fields 4. The Guest clicks on the confirmation button 5. The system saves the data and sends a confirmation email 6. The Guest clicks on the confirmation link 7. The system activates the new account
Exit condition	Guest successfully ends registration process and becomes a Taxi driver or a Client
Exceptions	a. The Guest is already logged-in 1. The Guest is redirected to their personal page b. One or more fields are not valid 4. The Guest is alerted and application goes back to event 3 c. The chosen email or other unique information are already associated to another user 4. The Guest is alerted and application goes back to event 3 d. The email does not exist 5. The Guest is alerted and application goes back to event 3
Special requirements	Once the confirmation email is sent it must be received by the Guest.


SD Sign-up
:Guest
:User
:System
loop
[confirmation email not sent]
loop
[email not sent]
insert_email()
validate_input_format()
alt
[valid input]
send(guest_details)
validate_email()
alt
[valid email]
save_data()
send_email()
send_notification(email_sent)
[else]
send_notification(email_error)
confirmation_email_clicked()
activate_account()
account_activated()

3.4.2 Login

This use case describes the process of login of a User.

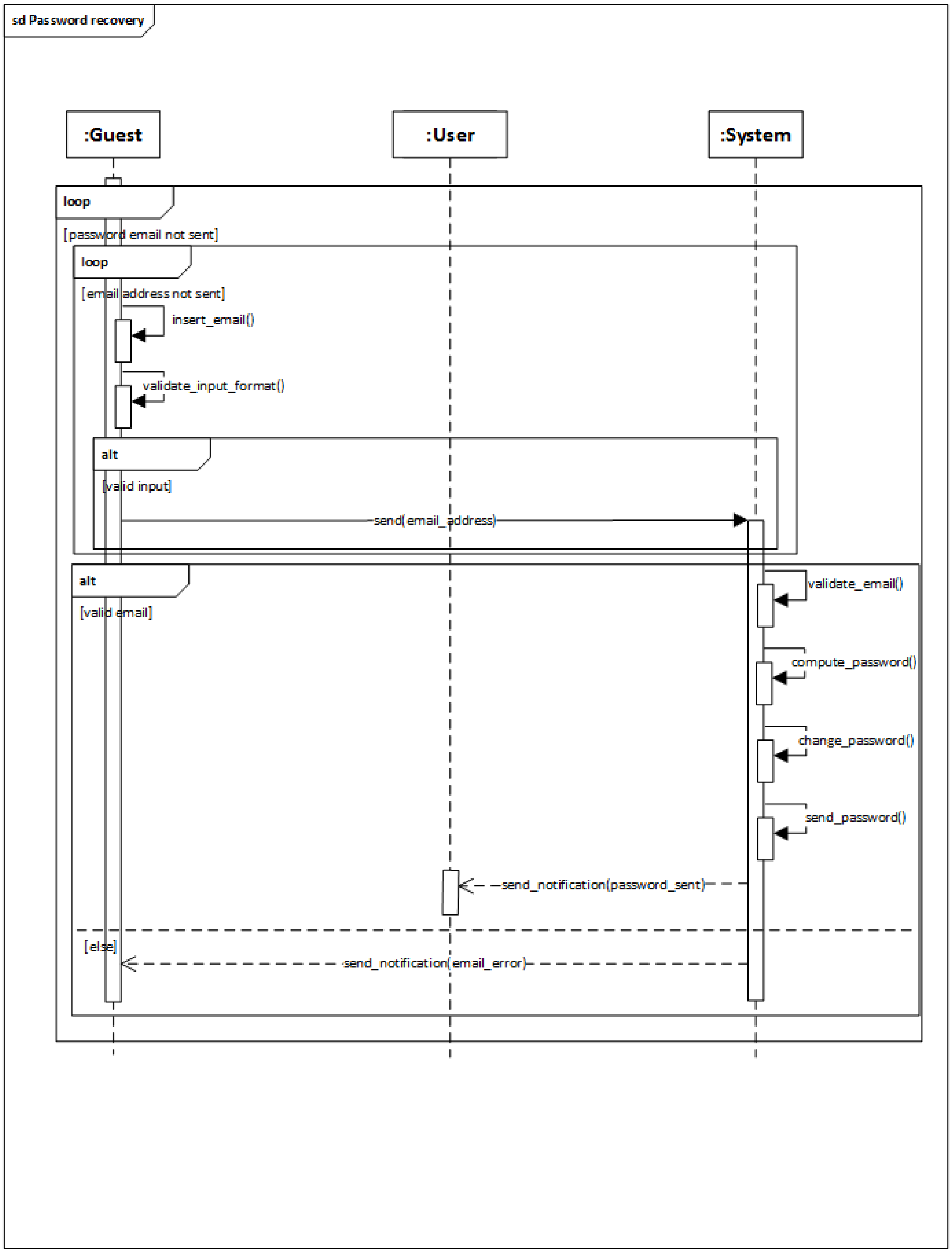
Name of the use case	**[UC02] Login**
Actors	Guest, Client, Taxi driver
Entry condition	The Guest is registered into the system using either the mobile application or the web application
Flow of events	1. The Guest clicks on “Login” 2. The Guest completes the form inserting correct email and password 3. The Guest clicks on the confirmation button
Exit condition	The Guest is promoted to Taxi driver or Client
Exceptions	a. The Guest is already logged-in 1. The Guest is redirected to their personal page b. The email and password are not valid 3. The Guest is alerted and the application goes back to event 2 c. The email and password combination is wrong 3. The Guest is alerted and the application goes back to event 2
Special requirements	-


sd Login
:Guest
:User
:System
loop
[login not done]
loop
[email and password not sent]
insert_credentials()
insert_format()
alt
[email and password format correct]
send(credentials)
validate_credentials()
alt
send_notification(login)
[email and password correct]
[else]
send_notification(login_error)

3.4.3 Password recovery

This use case describes the process of password recovery for a User.

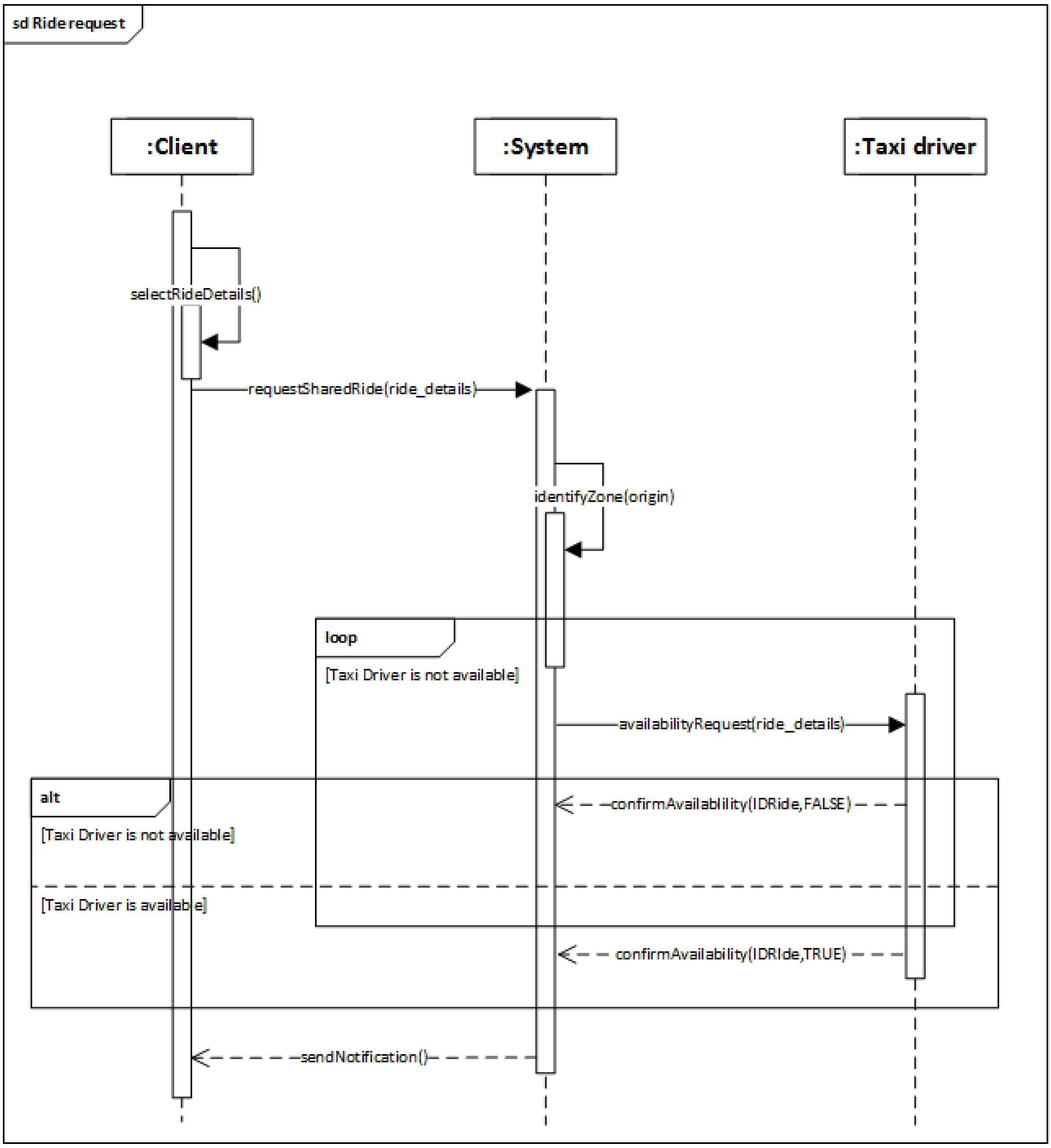
Name of the use case	**[UC03] Password recovery**
Actors	Client, Taxi driver
Entry condition	The Guest is registered into the system using either the mobile application or the web application
Flow of events	1. The User clicks on “Have you forgotten your password?” 2. The User completes the form inserting their registration email 3. The User clicks on the confirmation button 4. The system sends a randomly generated password to the User’s email account
Exit condition	The User receives a temporary password to access their myTaxiService account
Exceptions	a. The User is already logged-in 1. User is redirected to their personal page b. The email is not valid 3. The Guest is alerted and application goes back to event 2 c. The email does not belong to any user 3. The Guest is alerted and application goes back to event 2
Special requirements	Once the confirmation email is sent it must be received by the Guest.

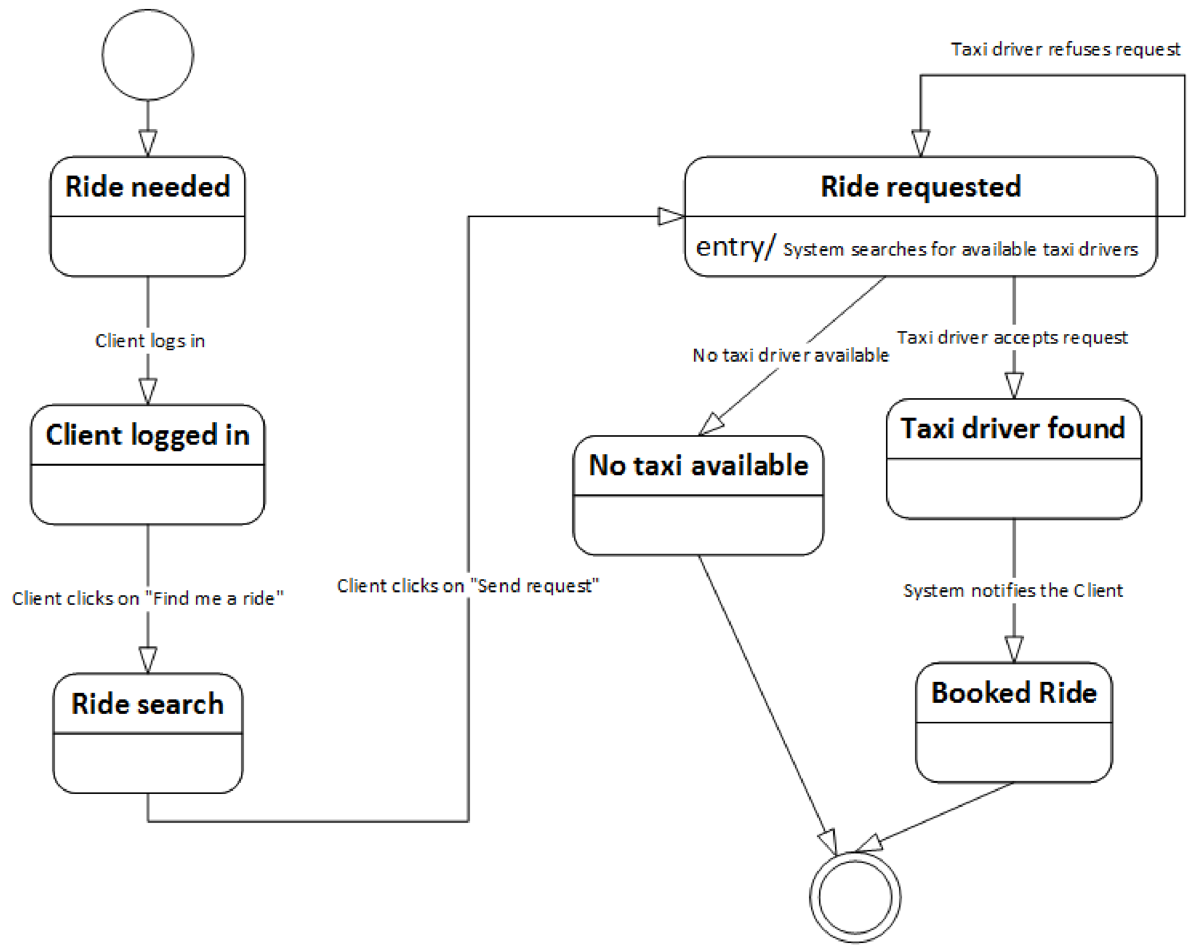

sd Password recovery
:Guest
:User
:System
loop
[password email not sent]
loop
[email address not sent]
insert_email()
validate_input_format()
alt
[valid input]
send(email_address)
alt
[valid email]
validate_email()
compute_password()
change_password()
send_password()
send_notification(password_sent)
[else]
send_notification(email_error)

3.4.4 Requesting a ride

This use case describes the process of requesting of a ride by a Client and the consequent possible actions of a Taxi Driver.

Name of the use case	**[UC04] Requesting a ride**
Actors	Client, Taxi driver
Entry condition	The Client is logged in using either the mobile application or the web application
Flow of events	1. The Client clicks on the “Find me a ride” button 2. The Client selects the origin position, the destination and “Right now” as the time of the ride, then clicks on the “Send request” button. 3. The request is sent to the system. 4. The system queries the database containing taxi queues to find the Taxi driver in the nearest zone compared to the Client. 5. The system sends a notification to the selected Taxi driver to ask them whether they accept the request or not 6. i. The Taxi driver choses to accepts the request ii. The Taxi driver doesn’t accept the request: the system moves the Taxi driver to the last position of the queue and forwards the request to the second taxi in the queue, going back to event 5. 7. The system notifies the Client about the incoming Taxi driver’s ID and the waiting time.
Exit condition	The Client has successfully booked a taxi
Exceptions	-
Special requirements	-

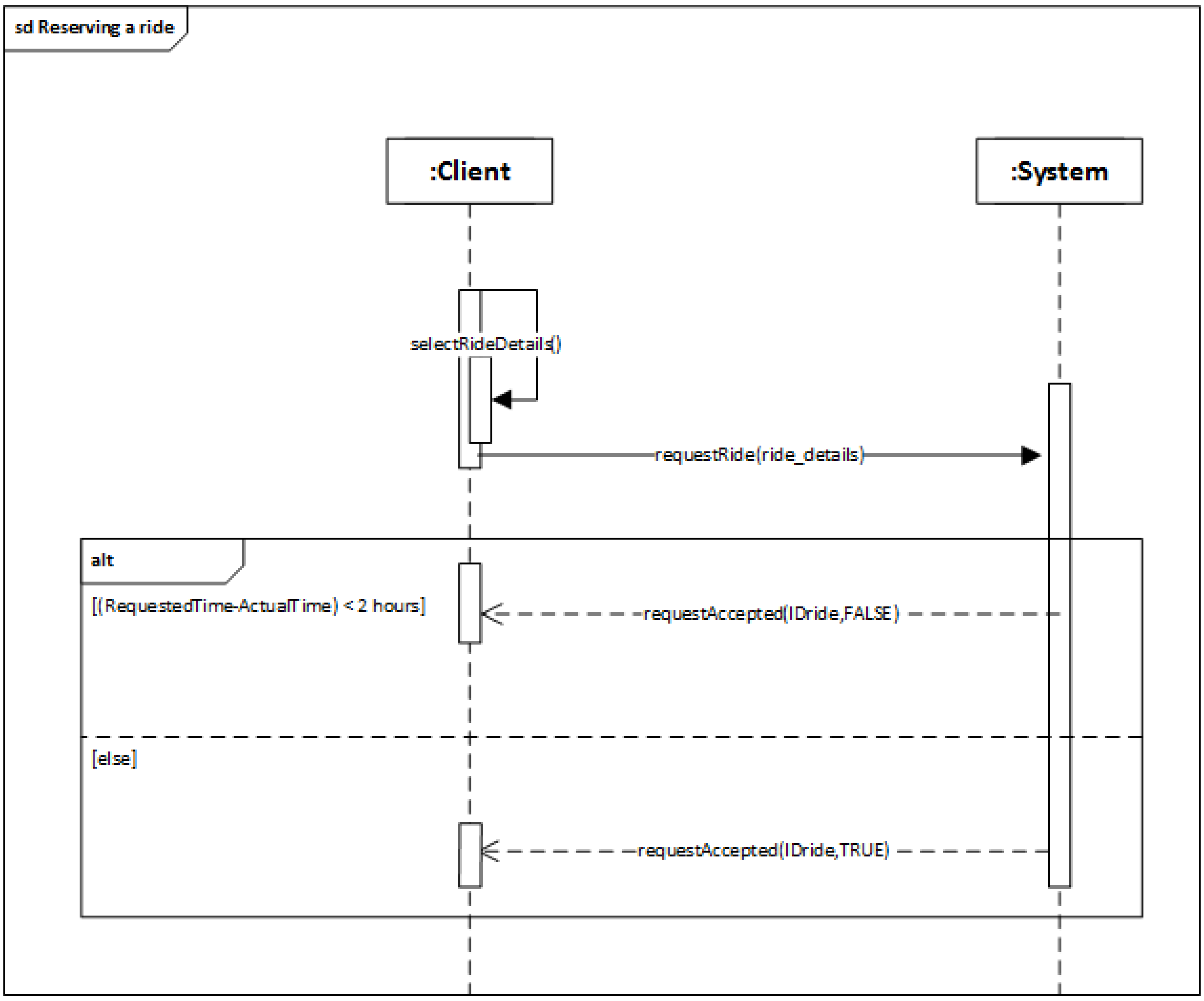

sd Ride request
:Client
:System
:Taxi driver
selectRideDetails()
requestSharedRide(ride_details)
identifyZone(origin)
loop
[Taxi Driver is not available]
availabilityRequest(ride_details)
alt
[Taxi Driver is not available]
confirmAvailablility(IDRide,FALSE)
[Taxi Driver is available]
confirmAvailability(IDRIde,TRUE)
sendNotification()


Taxi driver refuses request
Ride needed
Ride requested
entry/ System searches for available taxi drivers
Client logs in
No taxi driver available
Taxi driver accepts request
Client logged in
Taxi driver found
No taxi available
Client clicks on "Find me a ride"
Client clicks on "Send request"
System notifies the Client
Ride search
Booked Ride

3.4.5 Reserving a ride

This use case describes the process of reservation of a ride by a Client.

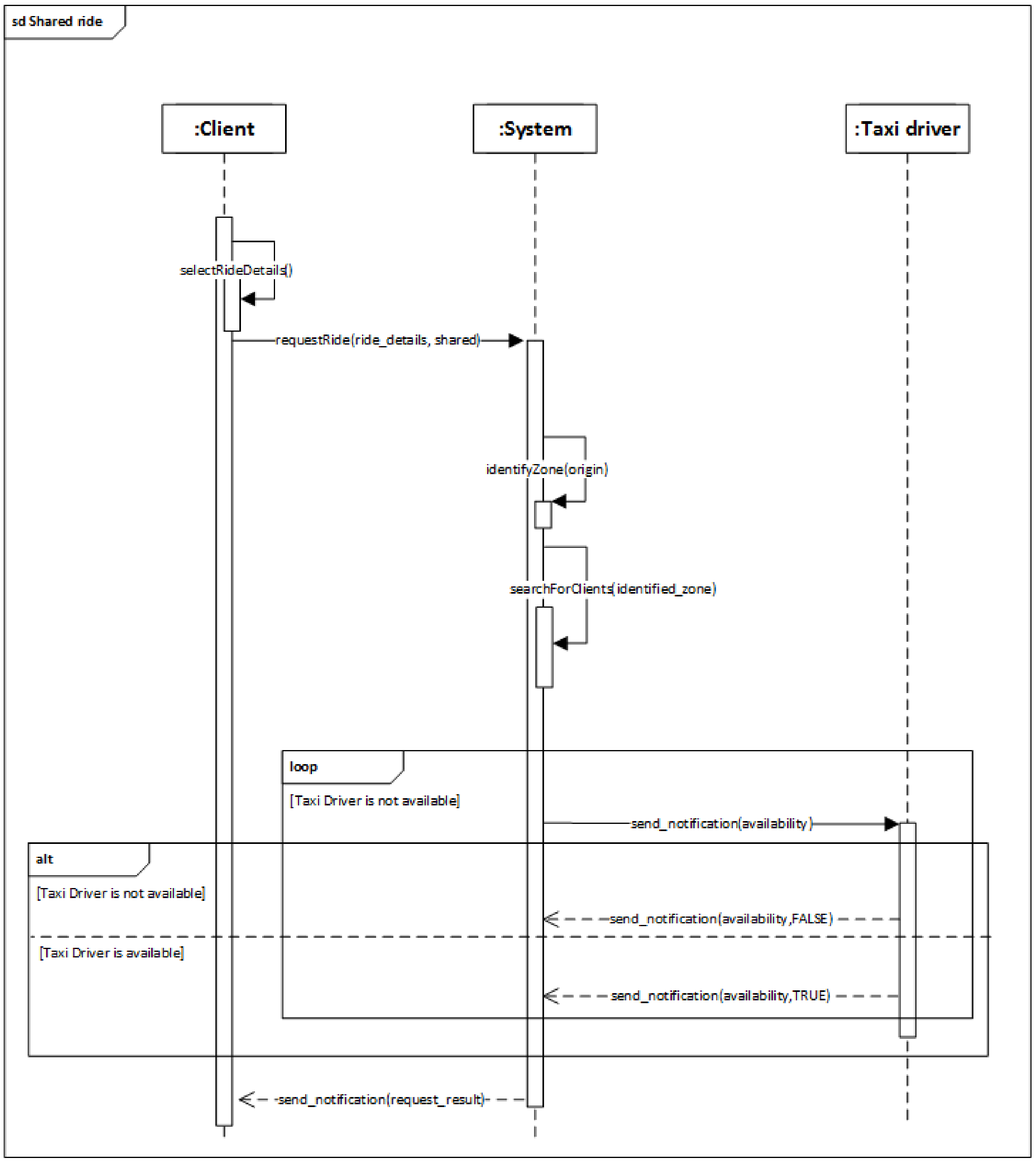
Name of the use case	**[UC05] Reserving a ride**
Actors	Client, Taxi driver
Entry condition	The Client is logged in using either the mobile application or the web application
Flow of events	1. The Client clicks on the “Find me a ride” button 2. The Client selects the origin, the destination and the time of the ride, then clicks on the “Send request” button. The request is sent to the system. 3. The system checks if the specified time of the ride is at least two hours after the current time: if not, the system notifies the Client, who has to change the time. 4. The system notifies the Client that the ride was successfully reserved and that they will be notified 10 minutes before the established time about their taxi details
Exit condition	The Client has successfully reserved a taxi
Exceptions	-
Special requirements	-


sd Reserving a ride
:Client
:System
selectRideDetails()
requestRide(ride_details)
alt
[(RequestedTime-ActualTime) < 2 hours]
requestAccepted(IDride,FALSE)
[else]
requestAccepted(IDride,TRUE)

3.4.6 Shared ride

This use case describes the process of reservation of a shared ride by a Client and the consequent possible actions of a Taxi Driver.

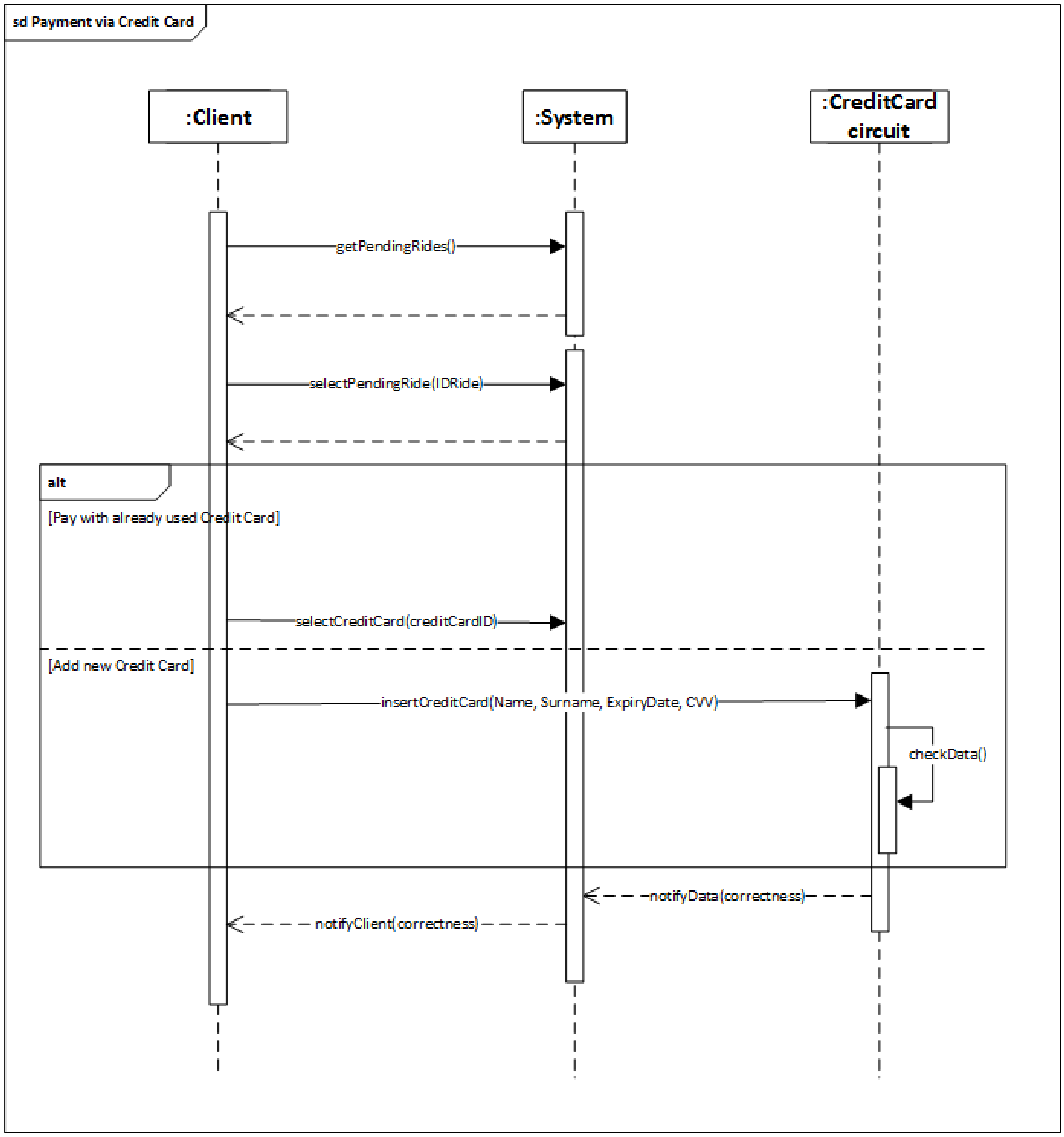
Name of the use case	**[UC06] Shared ride**
Actors	Client, Taxi driver
Entry condition	The Client is logged in using either the mobile application or the web application
Flow of events	1. The Client clicks on the “Find me a ride” button 2. The Client selects the origin, the destination and the time of the ride and enables the “Taxi sharing” option, then clicks on the “Send request” button. The request is sent to the system. 3. The system identifies the zone selected as the origin of the ride and checks if there are other requests coming from that zone. 4. The system queries the database containing taxi queues to find the Taxi driver in the nearest zone compared to the Client. 5. The system sends a notification to the selected Taxi driver to ask them whether they accept the request or not 6. i. The Taxi driver chooses to accepts the request: the system arranges the route for the Taxi driver and computes the fee for all Clients sharing the taxi ii. The Taxi driver doesn’t accept the request: the system moves the Taxi driver to the last position of the queue and forwards the request to the second Taxi driver in the queue, going back to event 5. 7. The system notifies all Clients about the incoming Taxi driver’s ID, the waiting time and the estimated fee..
Exit condition	The Client has successfully booked a taxi and will share the ride with one or more other Clients.
Exceptions	-
Special requirements	-


sd Shared ride
:Client
:System
:Taxi driver
selectRideDetails()
requestRide(ride_details, shared)
identifyZone(origin)
searchForClients(identified_zone)
loop
[Taxi Driver is not available]
send_notification(availability)
alt
[Taxi Driver is not available]
send_notification(availability,FALSE)
[Taxi Driver is available]
send_notification(availability,TRUE)
send_notification(request_result)

3.4.7 Payment via Credit Card

This use case describes the process of payment via Credit Card by a Client.

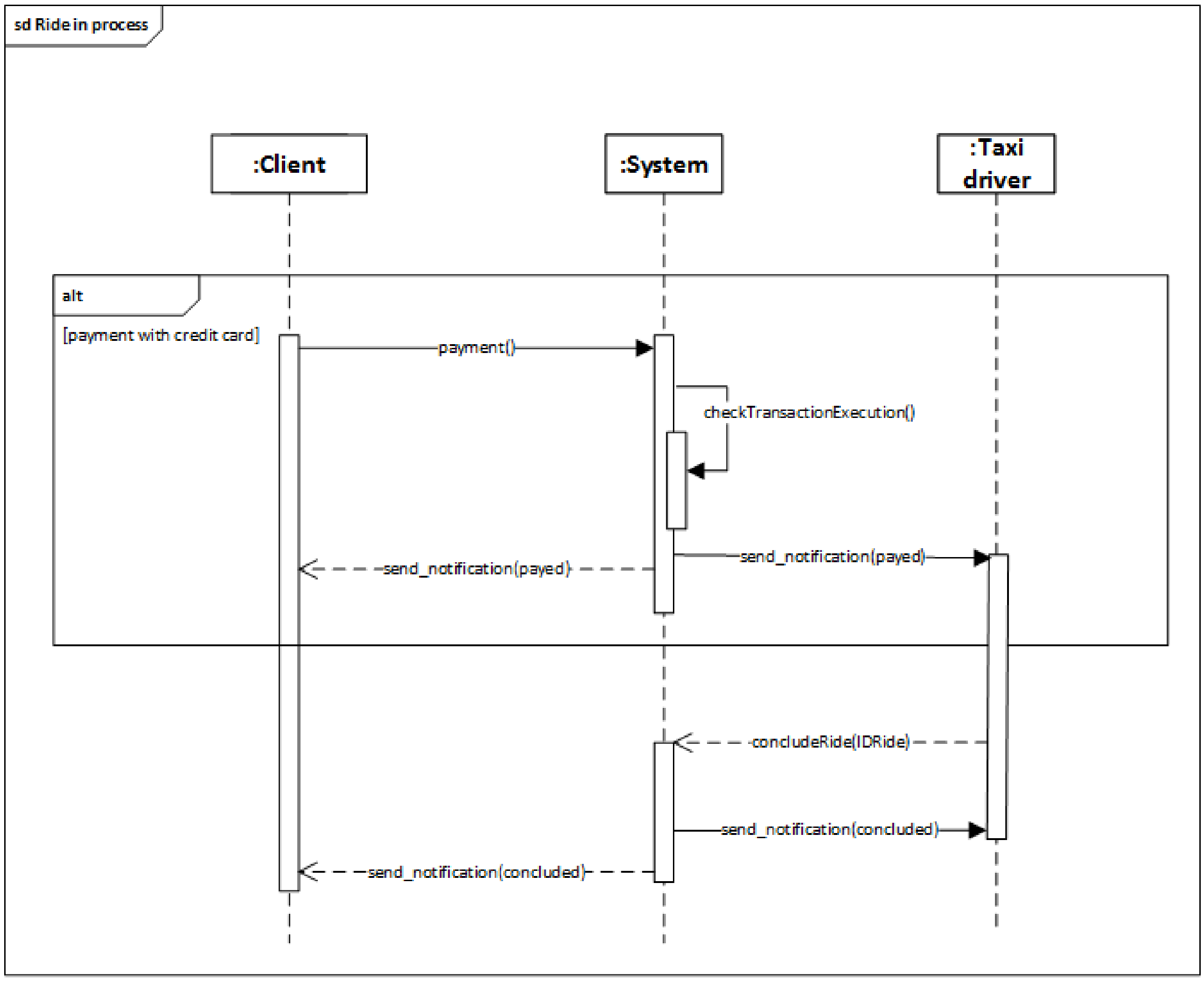
Name of the use case	**[UC07] Payment via Credit Card**
Actors	Client
Entry condition	The Client has successfully booked a ride
Flow of events	1. The Client opens the “myTaxiRides” tab from either the website or the mobile app and choses the “Pending Rides” tab 2. The Client selects the ride that they want to pay for via Credit Card 3. The Client choses whether they want to use a Credit Card that is already in their account or with a new one 4. i. The Client choses the Credit Card they want to pay with ii.1 The Client selects their Credit Card’s company ii.2 The system redirects the Client to their Credit Card company’s website ii.3 The Client inserts the required information (number, expiration date and CVV) about their Credit Card and confirms the payment 5. The system notifies both the Taxi driver and the Client about the successful transaction
Exit condition	The Client has successfully paid for the ride
Exceptions	a. The transaction fails 5. The system notifies the Client about the failure and asks them if they want to try again. If yes, the Client repeats from event 3. If no, the Client is redirected to the home page
Special requirements	The Credit Card needs to be valid

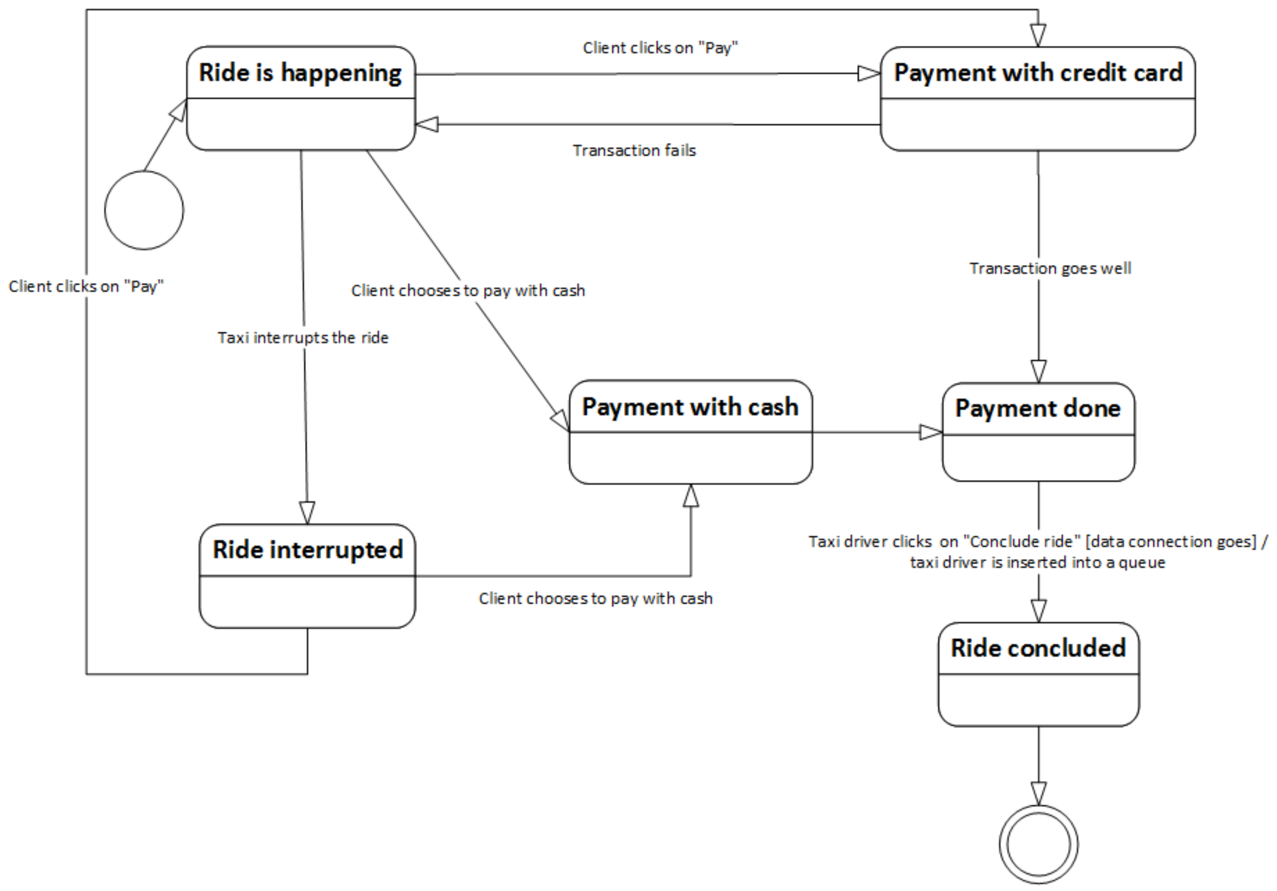

sd Payment via Credit Card
:Client
:System
:CreditCard circuit
getPendingRides()
selectPendingRide(IDRide)
alt
[Pay with already used Credit Card]
selectCreditCard(creditCardID)
[Add new Credit Card]
insertCreditCard(Name, Surname, ExpiryDate, CVV)
checkData()
notifyData(correctness)
notifyClient(correctness)

3.4.8 Ride in process

This use case describes the steps from when a ride is under way to when it ends.

Name of the use case	**[UC08] Ride in process**
Actors	Client, Taxi driver
Entry condition	The Client starts a ride with a Taxi driver
Flow of events	1. The ride proceeds well until its end 2. The Client pays the Taxi driver with cash/credit card 3. The Taxi driver confirms the end of the ride
Exit condition	The ride is shifted into the concluded rides and the Taxi driver into the queue of their current zones
Exceptions	a. The Taxi driver and the Client agree on an interruption of the ride before its end 1. The Taxi driver inserts into the app the current destination of the ride and is recalculates the price to pay b. The Client payment with credit card (see UC07) fails 2. The Client needs to pay with cash c. The data connection of the Taxi driver does not work 3. The system cannot put the Taxi driver into a queue until they confirm the end of the ride. They remain temporally out.
Special requirements	-

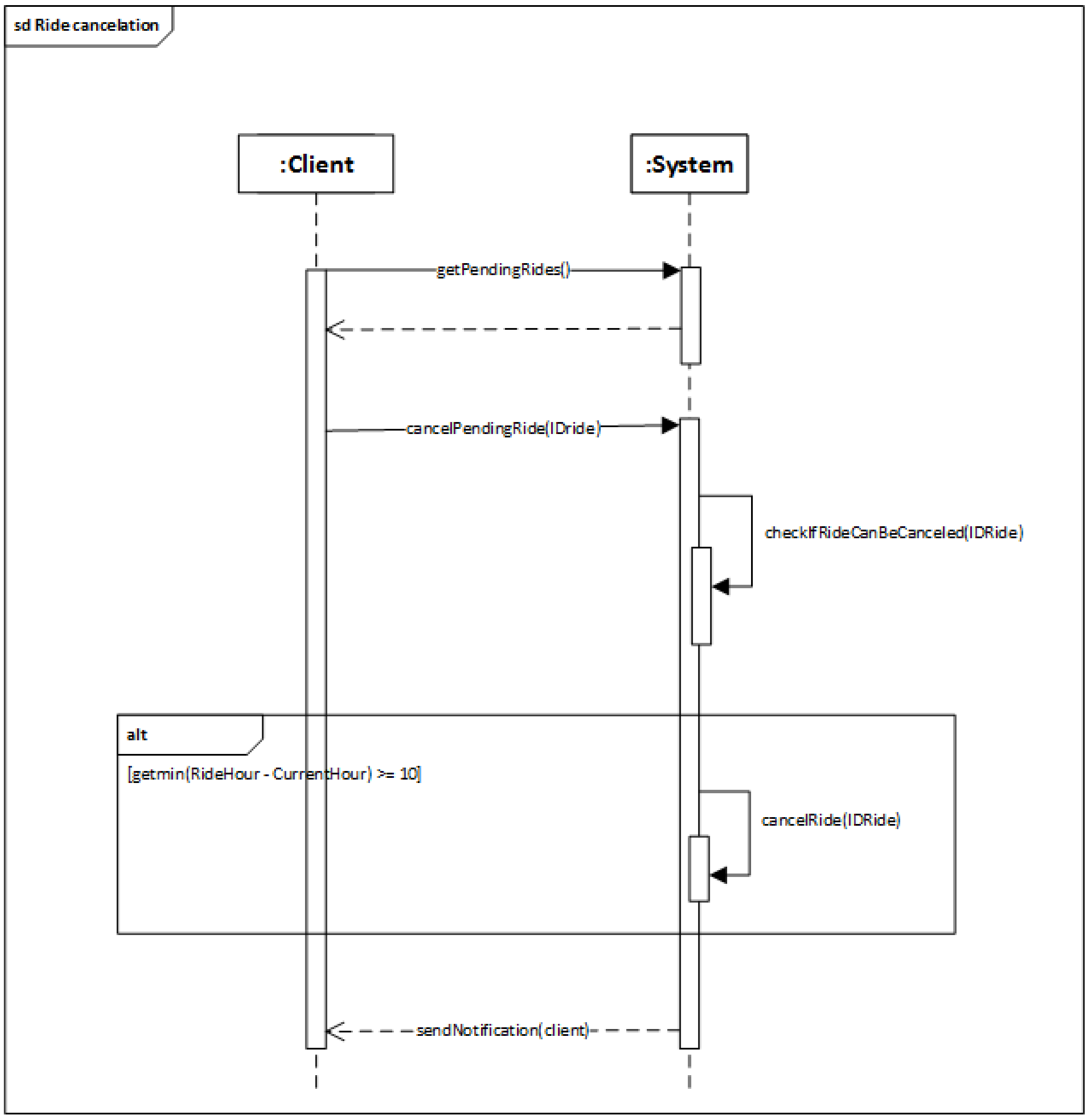

sd Ride in process
:Client
:System
:Taxi driver
alt
[payment with credit card]
payment()
checkTransactionExecution()
send_notification(payed)
send_notification(payed)
concludeRide(IDRide)
send_notification(concluded)
send_notification(concluded)


Ride is happening
Client clicks on "Pay"
Payment with credit card
Transaction fails
Client clicks on "Pay"
Client chooses to pay with cash
Transaction goes well
Taxi interrupts the ride
Payment with cash
Payment done
Ride interrupted
Taxi driver clicks on "Conclude ride" [data connection goes] / taxi driver is inserted into a queue
Client chooses to pay with cash
Ride concluded

3.4.9 Ride cancelation

This use case describes the process of cancelation of a reserved ride by a Client.

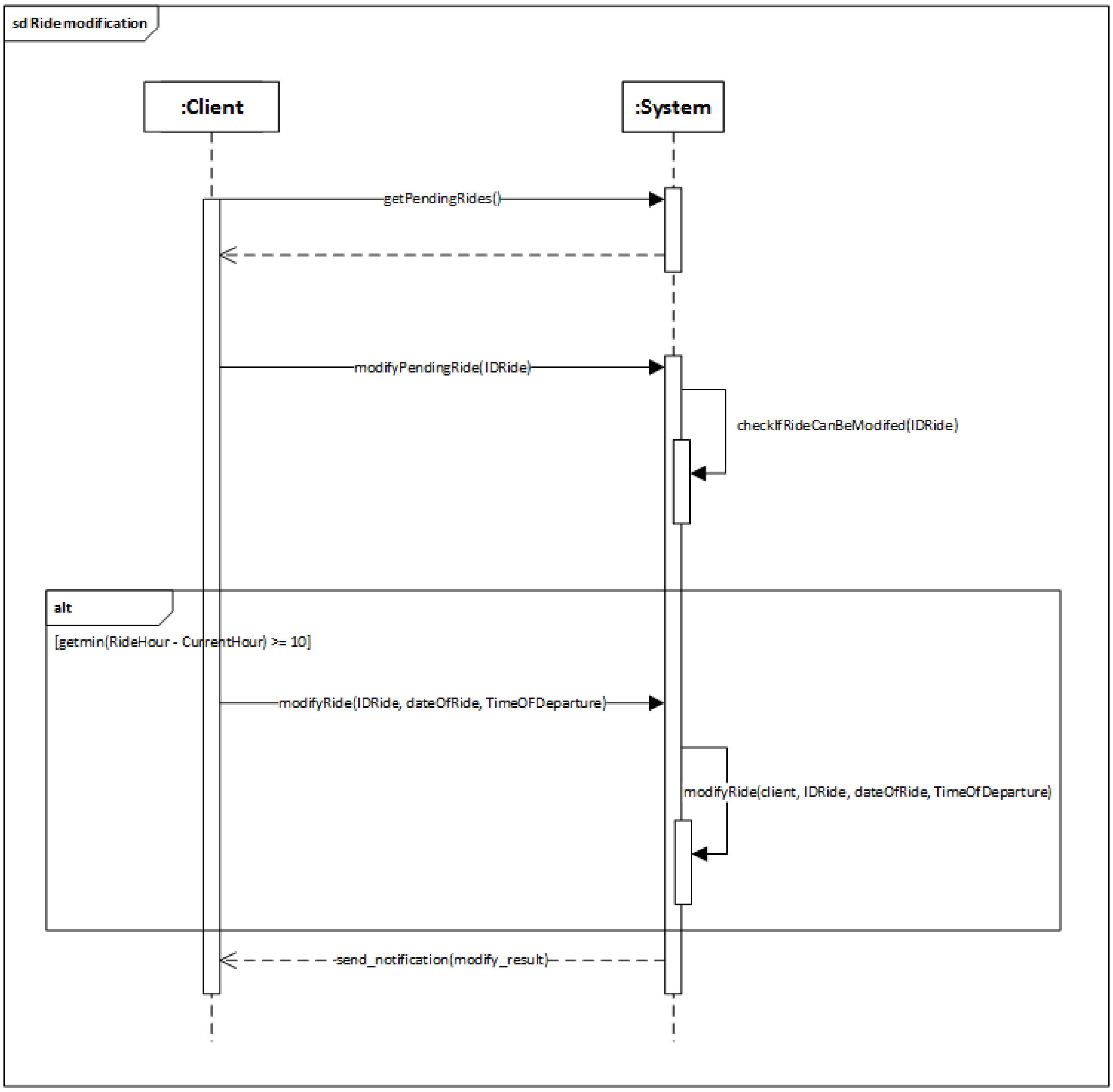
Name of the use case	**[UC09] Ride cancelation**
Actors	Client
Entry condition	The Client is logged in using either the mobile application or the web application
Flow of events	1. The user clicks on the button "myTaxiRides" in the app or in the web site to see the list of their concluded/pending rides 2. The user clicks on "Pending rides" 3. The Client clicks on the button "Delete" related to a pending ride that has been reserved 4. The system checks if that ride can be canceled and allows the Client to cancel it 5. The system notifies the Client of the correct cancelation of the reservation
Exit condition	The Client cancels the ride or does not cancel it because there is less than 10 minutes left before the reservation
Exceptions	a. There's less than 10 minutes left before the reservation 5. The system does not allow the Client to cancel the reservation because there's less than 10 minutes left before the reservation b. An error occurs during the cancelation process 6.i. The system notifies the Client of the non-cancelation of the ride because of an internal error 6.ii. The system notifies the Client of the non-cancelation of the ride because the reservation is in less than 10 minutes
Special requirements	-


sd Ride cancelation
:Client
:System
getPendingRides()
cancelPendingRide(IDride)
checkIfRideCanBeCanceled(IDRide)
alt
[getmin(RideHour - CurrentHour) >= 10]
cancelRide(IDRide)
sendNotification(client)

3.4.10 Ride modification

This use case describes the process of modification of a reserved ride by a Client.

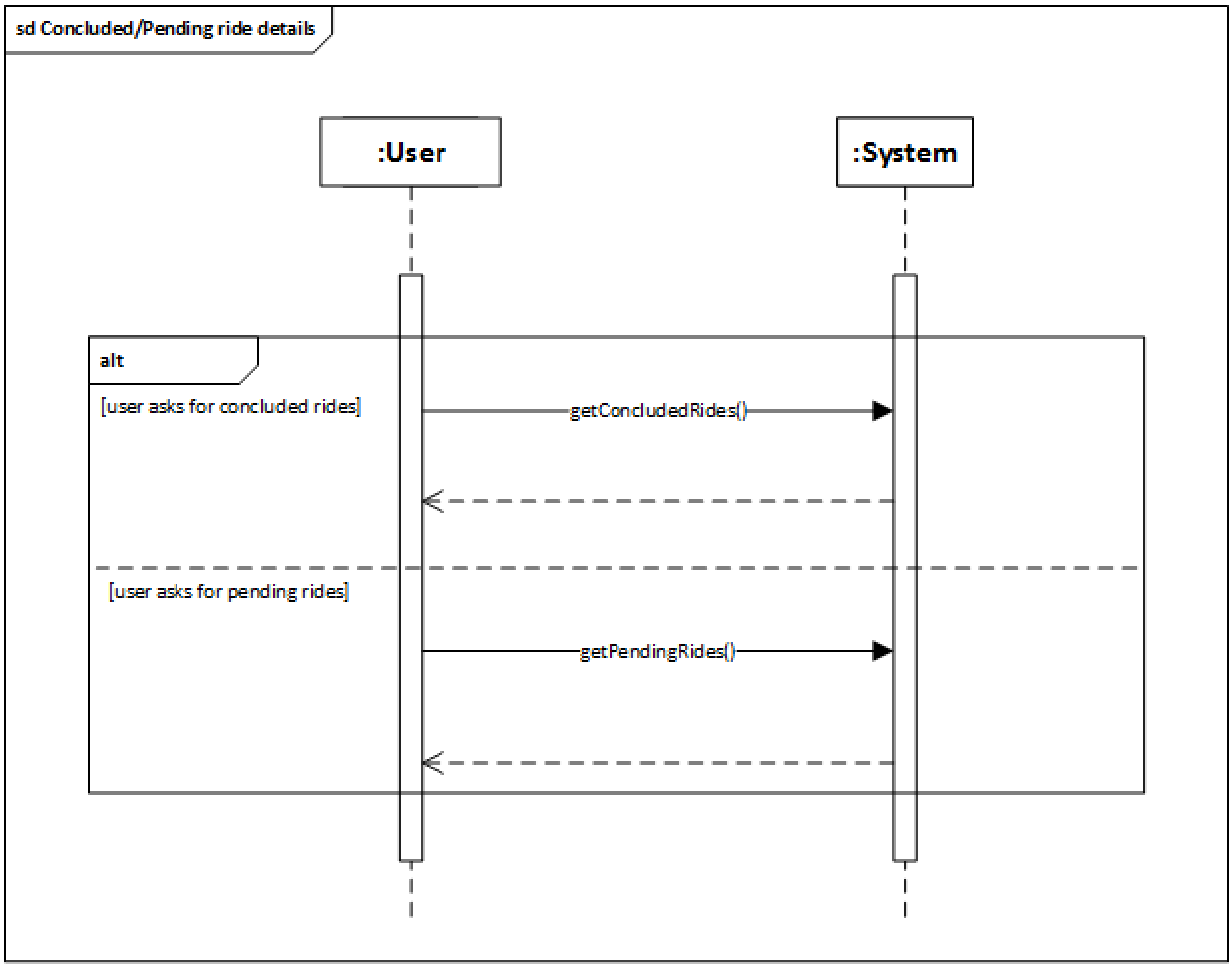
Name of the use case	**[UC10] Ride modification**
Actors	Client
Entry condition	The Client is logged in using either the mobile application or the web application
Flow of events	1. The user clicks on the button "myTaxiRides" in the app or in the web site to see the list of their concluded/pending rides 2. The user clicks on "Pending rides" 3. The Client clicks on the button "Modify" related to a pending ride that is been reserved 4. The system checks if that ride can be modified and allows the Client to modify it 5. The Client inserts into a new page the new desired date and time and clicks on "OK" button 6. The system notifies the Client of the correct modification of the reservation
Exit condition	The Client modifies the ride or does not modify it because there is less than 10 minutes left before the reservation
Exceptions	a. There's less than 10 minutes left before the ride 5. The system doesn't allow the Client to modify the reservation because there's less than 10 minutes left before the reservation b. An error occurs during the modification process 6.i. The system notifies the Client of the non-modification of the ride because of an internal error 6.ii. The system notifies the Client of the non-modification of the ride because the reservation is in less than 10 minutes
Special requirements	-


sd Ride modification
:Client
:System
getPendingRides()
modifyPendingRide(IDRide)
checkIfRideCanBeModifed(IDRide)
alt
[getmin(RideHour - CurrentHour) >= 10]
modifyRide(IDRide, dateOfRide, TimeOFDeparture)
modifyRide(client, IDRide, dateOfRide, TimeOfDeparture)
send_notification(modify_result)

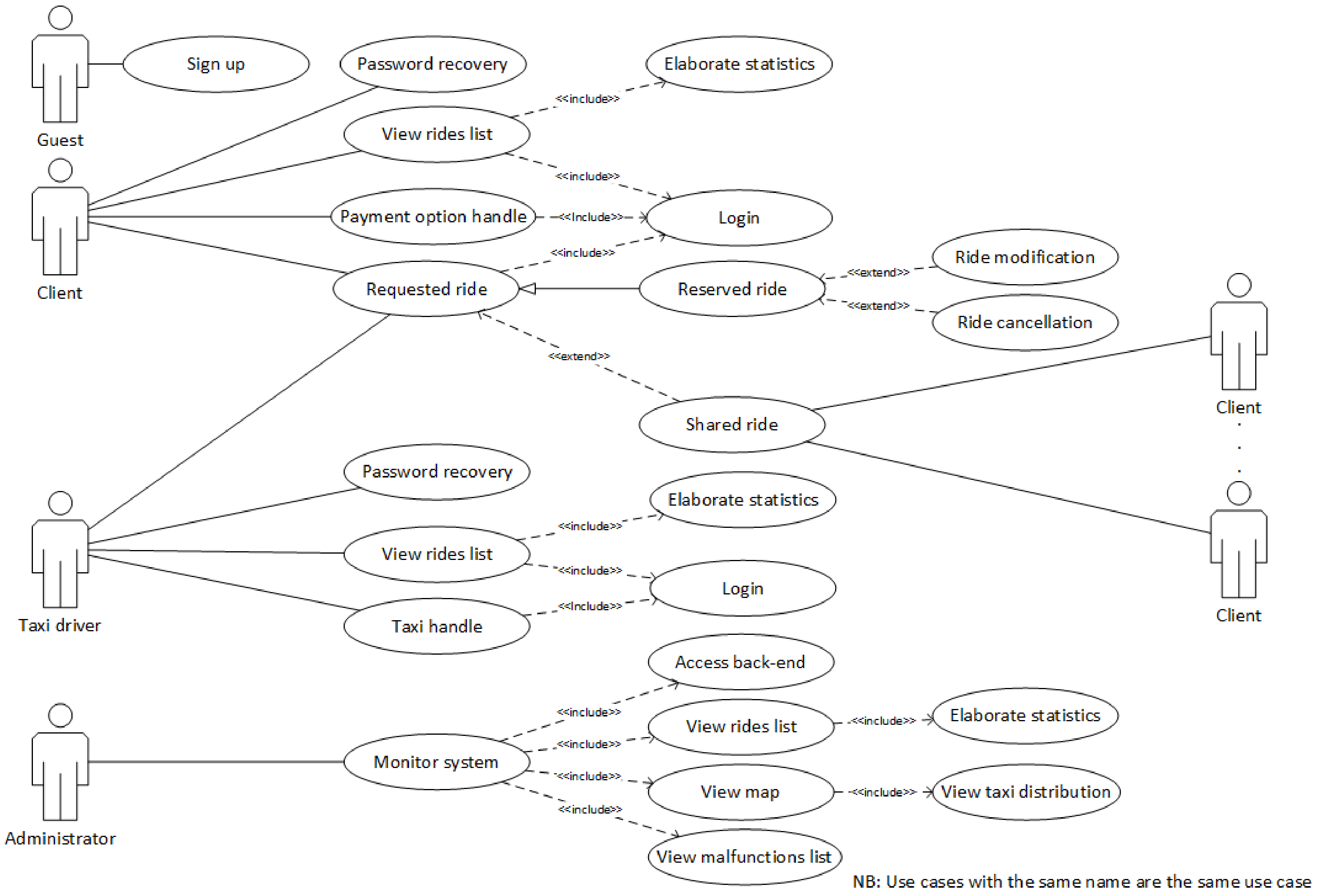
3.4.11 Concluded/Pending ride details

This use case describes the steps to see the list of concluded/pending rides of a User.

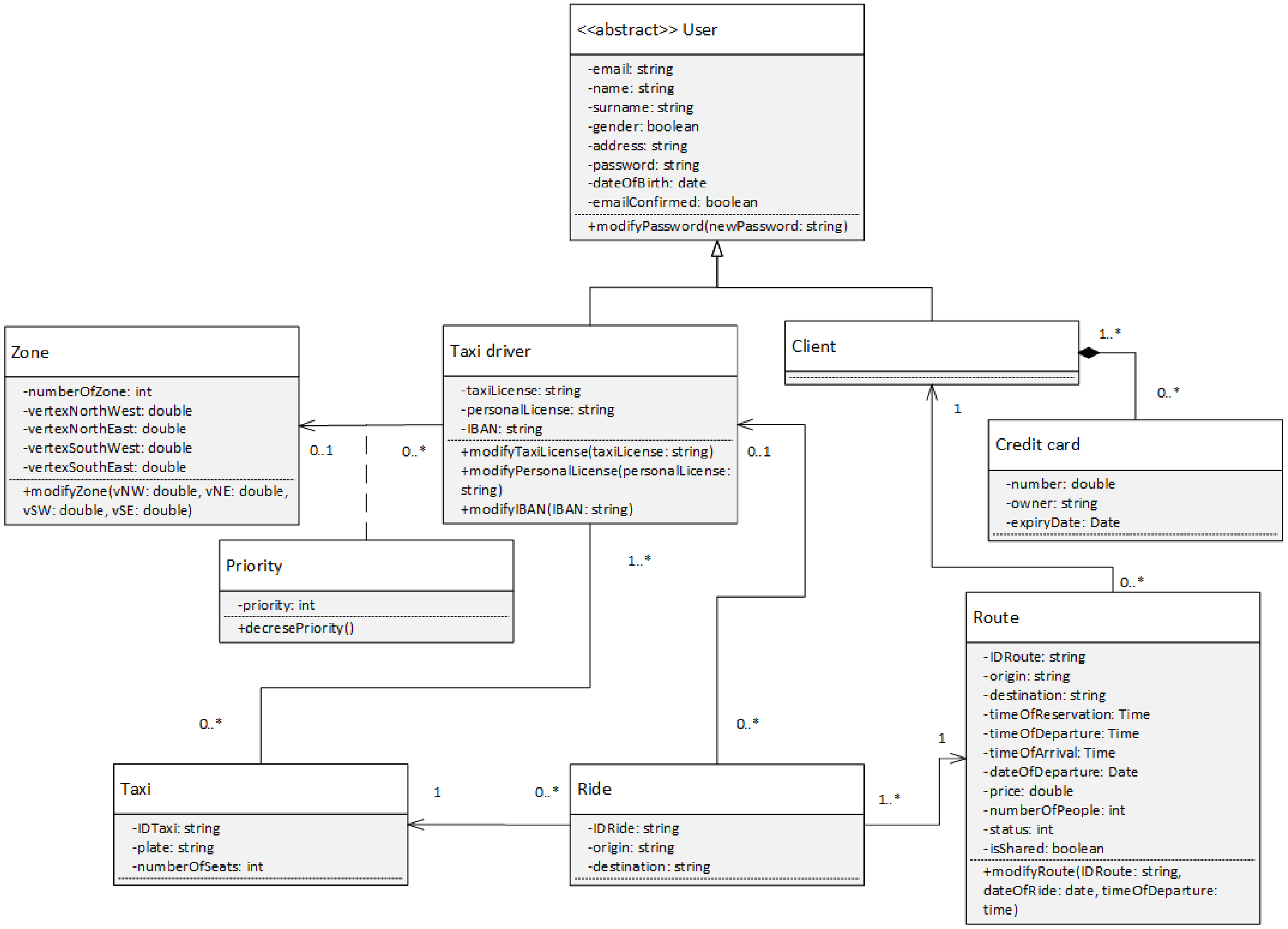
Name of the use case	**[UC11] Concluded/Pending ride details**
Actors	Client, Taxi driver
Entry condition	The User is logged in using either the mobile application or the web application
Flow of events	1. The User clicks on the button “myTaxiRides” in the app or in the web site to see the list of their concluded/current rides 2. The User clicks on “Concluded rides” 3. The system provides the list of the User’s concluded rides with different option 4. The User can see the list of their concluded rides and a summary in a graph of the number of rides grouped by a certain period of time
Exit condition	The system provides the Client or the Taxi driver the list of their concluded/pending rides
Exceptions	a. The User wants to see the “Pending rides” 1. The User clicks on “Pending rides” 2. The system provides the list of the User’s pending rides with different option 3. The User can see their pending rides’ details e.g. estimated arrival time of the taxi, information on the ride and possibility to delete/modify the ride if that is a reservation not yet allocated to a taxi (there’s more than 10 minutes left)
Special requirements	-


sd Concluded/Pending ride details
:User
:System
alt
[user asks for concluded rides]
getConcludedRides()
[user asks for pending rides]
getPendingRides()

3.5 Use case diagram



3.6 Class diagram



3.7 Non Functional Requirements

The following non-functional requirements have been defined with compliance to the ISO/IEC 25010:2011 Product Quality Model.

3.7.1 Functional suitability

Name	**[NFR-FS01] Functional completeness**
Description	The set of functions provided by the system and described in Chapter 3, Paragraph 3.2 “Functions” covers all the goals specified in Chapter 1, Paragraph 1.2 “Scope”.

Name	**[NFR-FS02] Functional correctness**
Description	The system shall provide the correct result 99,99% of the times. An incorrect execution (e.g. A taxi driver of an incorrect queue is chosen) would nullify the advantages that the system provides.

Name	**[NFR-FS03] Functional appropriateness**
Description	The set of functions provided by the system and described in Chapter 3, Paragraph 3.2 “Functions”, facilitates the tasks specified in Chapter 1, Paragraph 1.2 “Scope”.

3.7.2 Performance efficiency

Name	**[NFR-PE01] Time behaviour**
Description	Function [PF12] Queue handling shall be completed in less than 1 second in order to guarantee the ability to answer all requests with the correctness specified in [NFR-FS02] Functional correctness.

Name	**[NFR-PE02] Resource utilization**
Description	The system shall be stress-tested to verify whether it can maintain the level of performance tested in normal conditions.

Name	**[NFR-PE03] Capacity**
Description	The system shall be scaled to support concurrent access to at most 500,000 users

3.7.3 Compatibility

Name	**[NFR-C01] Co-existence**
Description	The system shall not have a detrimental impact on the products it shares the environment with.

Name	**[NFR-C02] Interoperability**

Description	The pre-existent system already exchanges information with other software using JSON and no major problems have been encountered. Therefore, the system shall use the same standard to interact with the pre-existent system. Information about routes and traffic will be obtained using the Google Maps APIs.

3.7.4 Usability

Name	**[NFR-U01] Appropriateness recognizability**
Description	A beta version of the system shall be tested before its release by a random group of users to verify whether it is appropriate for their needs.

Name	**[NFR-U02] Learnability**
Description	The system shall provide a tutorial that covers all the main functions as stated in function [PF26] Tutorial.

Name	**[NFR-U03] Operability**
Description	The system shall provide an administration interface (function [PF25] Administration panel).

Name	**[NFR-U04] User error protection**
Description	Only the main functions shall be shown in the user interface.

Name	**[NFR-U05] User interface aesthetics**
Description	The system shall provide clear user interfaces.

Name	**[NFR-U06] Accessibility**
Description	The system shall provide a tutorial that covers all the main functions as stated in function [PF26] Tutorial and that can be checked at any moment by the user.

3.7.5 Reliability

Name	**[NFR-R01] Maturity**
Description	The system shall be reliable 99.99% of times in normal conditions.

Name	**[NFR-R02] Availability**
Description	The system shall be operational for at least 364 days per year and can be down at most for 6 consecutive hours.

Name	**[NFR-R03] Fault tolerance**
Description	The system shall be installed on replicated servers so that the failure of one can be compensated by the other.

Name	**[NFR-R04] Recoverability**
Description	The system shall be installed on replicated servers so that, in case one encounters a problem, data can be recovered from the other.

3.7.6 Security

Name	**[NFR-S01] Confidentiality**
Description	The system shall use a secure authentication protocol.

Name	**[NFR-S02] Integrity**
Description	The system shall be monitored by a dual firewall which will create a DMZ to prevent unauthorized access to data.

Name	**[NFR-S03] Non-repudiation**
Description	The system shall record all accesses and operations in a log file.

Name	**[NFR-S04] Accountability**
Description	The system shall keep track of users' actions using a unique identifier.

3.7.7 Maintainability

Name	**[NFR-M01] Modularity**
Description	Each software module shall interact with other software modules using lists of parameters so that the coupling level is minimal and a change to one component has minimal impact on other components.

Name	**[NFR-M02] Reusability**
Description	Function [PF12] Queue handling can be used in other systems since it only requires information about taxis' positions (obtained using GPSs) and geographical data (obtained using Google Maps API).

Name	**[NFR-M03] Analysability**
Description	The system will receive log files about failures of the application containing information required that will help identifying which component caused the failure in at most 1 hour for the 95% of the errors that will occur (e.g. OS and operations that causes the mobile app crash).

Name	**[NFR-M04] Modifiability**
Description	The system shall be modifiable without introducing defects or degrading its quality before the change, as granted by [NFR-M01] Modularity.

Name	**[NFR-M05] Testability**
Description	Test case shall be specified (as comments) in each component of the source code.

3.7.8 Portability

Name	**[NFR-P01] Adaptability**
Description	The mobile application shall be developed for the three major mobile OSs (Android, iOS and Windows Phone). The website shall be easily accessed from the main web browsers (Google Chrome, Internet Explorer, Microsoft Edge, Mozilla Firefox, Safari, Opera).

Name	**[NFR-P02] Installability**
Description	The mobile application should be available in the Google Play Store, Apple Store, Windows Phone Store from which is possible to install it.

Name	**[NFR-P03] Replaceability**
Description	The software could potentially replace the already existent application *mytaxi*.

3.8 Alloy

3.8.1 Code

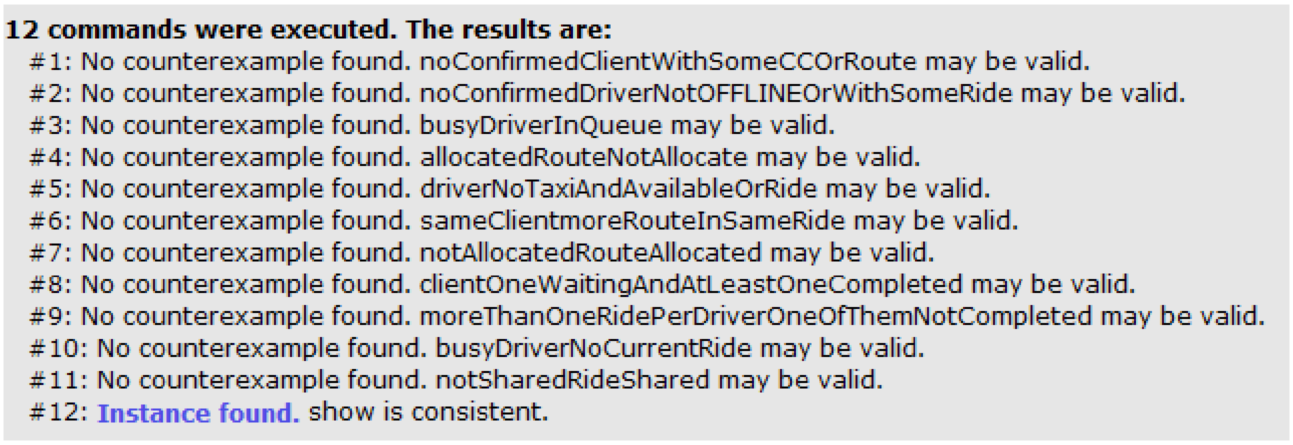
```
// Defines Bool, True, False
open util/boolean

//SIGNATURES
abstract sig User {
    emailConfirmed: one Bool
}

sig Client extends User {}

sig CreditCard {
    owner: some Client
}

sig TaxiDriver extends User {
    status: one TaxiDriverStatus
}

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

sig Zone {
    queue: one TaxiQueue
}

sig TaxiQueue {
    zone: one Zone,
    drivers: set TaxiDriver,
    taxis: set Taxi
}{
    //A taxi per driver
    #drivers = #taxis
}

sig Ride{
    driver: one TaxiDriver,
    taxi: one Taxi,
    routes: some Route,
}

sig Route{
    client: one Client,
    ride: lone Ride,
    isShared: one Bool,
    isReserved: one Bool,
    status: one RouteStatus
}

abstract sig RouteStatus {}
sig NOT_ALLOCATED extends RouteStatus {}
sig WAITING extends RouteStatus {}
sig ON_BOARD extends RouteStatus {}
sig COMPLETED extends RouteStatus {}
sig Taxi {
    drivers: some TaxiDriver
}

//**********************************************FACTS***********************************
***********
//USER REGISTRATION STATUS CHECK
//A client can do something only if confirmed
fact activeClientIsConfirmed{
    all c: Client |
        c.emailConfirmed = False
        implies
        ((no cc: CreditCard | cc.owner = c) and (no rt: Route | rt.client = c))
}

//A taxi driver can do something only if confirmed
fact activeTaxiDriverIsConfirmed{
    all d: TaxiDriver |
        d.emailConfirmed = False
        implies
        ((no t: Taxi | d in t.drivers) and (no r: Ride | r.driver = d) and d.status =
        OFFLINE)
}

//DRIVER STATUS CHECK
//OFFLINE taxi driver are not in a queue
fact offlineDriverNotInQueue{
    all d: TaxiDriver | all q: TaxiQueue |
        d.status = OFFLINE
        implies
        d not in q.drivers
}

//BUSY taxi driver are not in a queue
fact busyDriverNotInQueue{
    all d: TaxiDriver | all q: TaxiQueue |
        d.status = BUSY
        implies
        d not in q.drivers
}

//A BUSY driver has active route
fact busyDriverHasACurrentRide{
    all d: TaxiDriver, r: Ride, rt:Route |
        (d.status = BUSY and r.driver = d and rt in r.routes)
        implies
        (rt.status = WAITING or rt.status = ON_BOARD)
}

//A BUSY driver has at least a ride
fact busyDriverOneRide{
    no d: TaxiDriver | d.status = BUSY and (no r: Ride | r.driver = d)
}
```

```
//AVAILABE taxi driver are in a queue
fact availableDriverInQueue{
    all d: TaxiDriver | one q: TaxiQueue |
        d.status = AVAILABLE
        implies
        d in q.drivers
}

//Taxi driver in a queue are AVAILABLE
fact driverInQueueAvailable{
    all d: TaxiDriver | one q: TaxiQueue |
        d in q.drivers
        implies
        d.status = AVAILABLE
}

//ROUTE STATUS CHECK
//A route in RESERVED_NOT_ALLOCATED state can't be connected to a ride
fact notAllocatedRouteNoRide{
    all rt: Route | no r: Ride | rt in r.routes and rt.status = NOT_ALLOCATED
}

//A route in WAITING, ON_BOARD or COMPLETED state is connected to a ride
fact allocatedRouteRide{
    all rt: Route |
        (rt.status = WAITING or rt.status = ON_BOARD or rt.status = COMPLETED)
        implies
        (one r: Ride | rt in r.routes)
}

// If a taxi driver has not completed a ride, he is busy
fact busyDuringRide {
    all d: TaxiDriver, r: Ride, rt: Route |
        (r.driver = d and rt in r.routes and rt.status != COMPLETED)
        implies
        d.status = BUSY
}

//SHARED ROUTE CHECK
//All clients that take part in a shared ride want to do a shared ride
fact isSharedConsistency{
    no r: Ride | some disj rt1, rt2: Route | rt1 in r.routes and rt2 in r.routes and
    rt1.isShared = False
}

//In a shared ride a route can't be WAITING if the others are RESERVED_NOT_ALLOCATED
//It's obvious since a RESERVED_NOT_ALLOCATED route can't be connected at a ride

//In a shared ride a route can't be ON_BOARD if the others are RESERVED_NOT_ALLOCATED
//It's obvious since a RESERVED_NOT_ALLOCATED route can't be connected at a ride

//In a shared ride a route can't be COMPLETED if the others are RESERVED_NOT_ALLOCATED
or WAITING
//It's obvious if RESERVED_NOT_ALLOCATED route can't be connected at a ride, but not
for WAITING
```

```
fact sharedRideWithOneCompleted{
    all disj rt1, rt2: Route | no r: Ride |  rt1 in r.routes and rt2 in r.routes and
    rt1.status = COMPLETED and rt2.status = WAITING
}

//TAXI CHECK
//Two working driver cant use the same taxi
fact workingNoSameTaxi{
    all d: TaxiDriver |
        (d.status = BUSY or d.status = AVAILABLE)
        implies
        (no disj r1,r2: Ride | r1.taxi = r2.taxi)
}

//A taxi can't be used by a driver in a queue and a ride not completed
fact taxiUsedOnlyByOneDriver{
    all r: Ride, rt: Route, q: TaxiQueue | no t: Taxi | t in q.taxis and r.taxi = t
    and rt in r.routes and (rt.status = WAITING or rt.status = ON_BOARD)
}

//If a taxi driver use a taxi in a ride that taxi has to be associated to the taxi
driver
fact driverUsesOnlyHisTaxis{
    //Each driver has used in rides only his/her taxi
    all r: Ride | no t: Taxi | r.taxi = t and r.driver not in t.drivers
    //Each available driver has at least one of his/her own taxi in the queue he/she
    is in
    all q: TaxiQueue | no d: TaxiDriver | d in q.drivers and (no t: Taxi | t in q.
    taxis and d in t.drivers)
}

//ROUTE CHECK
fact routeConsistency{
    //A route can be connected at only one ride
    all disj r1,r2: Ride | no rt: Route | rt in r1.routes and rt in r2.routes
    //A route is connected at a ride iff it appears in his routes
    all  r: Ride, rt: Route |
    rt in r.routes
    <=>
    rt.ride = r
    //Only a route can be associated to the same client in a ride
    all r: Ride | all disj rt1, rt2: Route | no c: Client | rt1 in r.routes and rt2 in
     r.routes and rt1.client = c and rt2.client = c
}

//A client can't have more than one current route
fact clientOneCurrentRoute{
    all disj rt1, rt2: Route | no c: Client | rt1.client = c and rt2.client = c and
    (rt1.status = WAITING or rt1.status = ON_BOARD) and (rt2.status = WAITING or rt2.
    status = ON_BOARD)
}

//A driver cant have more than one current ride
fact driverOneCurrentRide{
    all disj r1, r2: Ride | all rt1, rt2: Route | no d: TaxiDriver | rt1 in r1.routes
    and rt2 in r2.routes and r1.driver = d and r2.driver = d and (rt1.status = WAITING
    or rt1.status = ON_BOARD) and (rt2.status = WAITING or rt2.status = ON_BOARD)
}

//ZONE-QUEUE CHECK
//Each zone is connected with the queue that is related to it
fact zone_queue_relation {
    all z: Zone, q: TaxiQueue |
        z.queue = q
        iff
        q.zone = z
}

//Each driver can be in only one queue
fact driver_queue_relation{
    all disj q1,q2: TaxiQueue | no d: TaxiDriver | d in q1.drivers and d in q2.drivers
}

//Each taxi can be in only one queue
fact taxi_queue_relation{
    all disj q1,q2: TaxiQueue | no t: Taxi | t in q1.taxis and t in q2.taxis
}

//A taxi can be in a queue only if one of its owners is in it
fact taxi_driver_queue_relation{
    all q: TaxiQueue | no t: Taxi | t in q.taxis and (no d: TaxiDriver | d in q.
    drivers and d in t.drivers)
}

//***********************************************ASSERTIONS*****************************
**************
assert noConfirmedClientWithSomeCCOrRoute{
    no c: Client | c.emailConfirmed = False and ((some cc: CreditCard | cc.owner = c)
    or (some rt: Route | rt.client = c))
}
check noConfirmedClientWithSomeCCOrRoute

assert noConfirmedDriverNotOFFLINEOrWithSomeRide{
    no d: TaxiDriver | d.emailConfirmed = False and (d.status = AVAILABLE or d.status =
     BUSY or (some r: Ride | r.driver = d))
}
check noConfirmedDriverNotOFFLINEOrWithSomeRide

assert busyDriverInQueue{
    no d: TaxiDriver | d.status = BUSY and (one q: TaxiQueue | d in q.drivers)
}
check busyDriverInQueue

assert allocatedRouteNotAllocate{
    no rt: Route | (rt.status = WAITING or rt.status = ON_BOARD or rt.status =
    COMPLETED) and (no r: Ride | r = rt.ride)
}
check allocatedRouteNotAllocate

assert driverNoTaxiAndAvailableOrRide{
    all d: TaxiDriver |
        (d.status = AVAILABLE or d.status = BUSY or ((some r: Ride | r.driver = d)))
        implies
        (some t: Taxi | d in t.drivers)
}
check driverNoTaxiAndAvailableOrRide

assert sameClientmoreRouteInSameRide{
    all r: Ride | all disj rt1,rt2: Route |no c: Client |  rt1 in r.routes and rt2 in
    r.routes and rt1.client = c and rt2.client = c
}
check sameClientmoreRouteInSameRide

assert notAllocatedRouteAllocated{
    no rt: Route | rt.status = NOT_ALLOCATED and (one r: Ride | r = rt.ride)
}
check notAllocatedRouteAllocated

assert clientOneWaitingAndAtLeastOneCompleted{
    all r: Ride | all disj rt1,rt2: Route | no c: Client | rt1 in r.routes and rt2 in
    r.routes and rt1.client = c and rt2.client = c and rt1.status = WAITING and rt2.
    status = COMPLETED
}
check clientOneWaitingAndAtLeastOneCompleted

assert moreThanOneRidePerDriverOneOfThemNotCompleted{
    all disj r1,r2: Ride | all rt1, rt2: Route | no d: TaxiDriver |  rt1 in r1.routes
    and rt2 in r2.routes and r1.driver = d and r2.driver = d and (rt1.status = WAITING
    or rt1.status = ON_BOARD)
}
check moreThanOneRidePerDriverOneOfThemNotCompleted

assert busyDriverNoCurrentRide{
    no d: TaxiDriver | some r: Ride, rt: Route | d.status = BUSY and r.driver = d and
    rt in r.routes and rt.status = NOT_ALLOCATED and rt.status = COMPLETED
}
check busyDriverNoCurrentRide

assert notSharedRideShared{
    all r: Ride | all disj rt1,rt2: Route|
        (rt1 in r.routes and rt2 in r.routes)
        implies
        rt1.isShared = True
}
check notSharedRideShared

// COMMANDS
pred show(){
    //At least one shared ride
    some r: Ride | some disj rt1, rt2: Route | rt1 in r.routes and rt2 in r.routes
    #{rt: Route | rt.status = NOT_ALLOCATED} > 0
    #{rt: Route | rt.status = WAITING} > 0
    #{rt: Route | rt.status = ON_BOARD} > 0
    #{rt: Route | rt.status = COMPLETED} > 0
    #{d: TaxiDriver | d.status = AVAILABLE} > 1
    #{d: TaxiDriver | d.status = BUSY} > 1
    #{d: TaxiDriver | d.status = OFFLINE} > 0
}
run show for 7
```

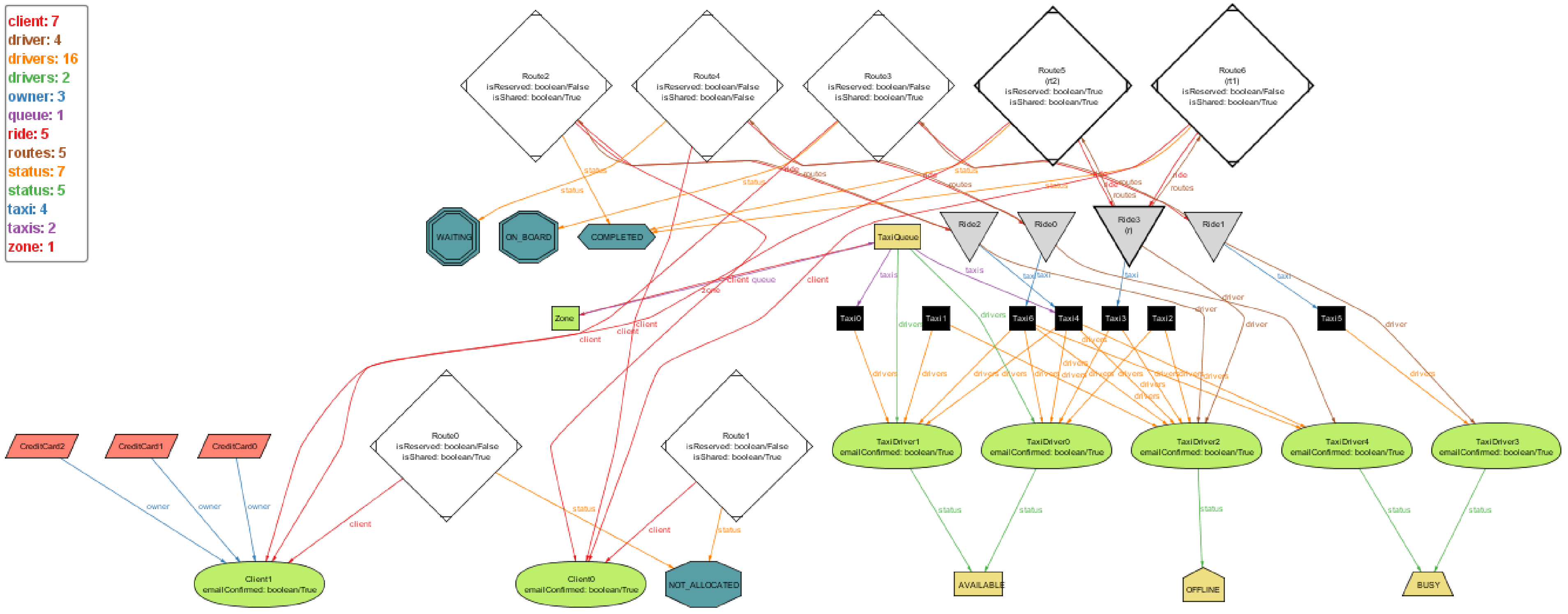
3.8.2 Results

This screenshot of the Alloy Analyzer software shows the consistence of the model with respect to the assertions that have been written.



```
12 commands were executed. The results are:
  #1: No counterexample found. noConfirmedClientWithSomeCCOrRoute may be valid.
  #2: No counterexample found. noConfirmedDriverNotOFFLINEOrWithSomeRide may be valid.
  #3: No counterexample found. busyDriverInQueue may be valid.
  #4: No counterexample found. allocatedRouteNotAllocate may be valid.
  #5: No counterexample found. driverNoTaxiAndAvailableOrRide may be valid.
  #6: No counterexample found. sameClientmoreRouteInSameRide may be valid.
  #7: No counterexample found. notAllocatedRouteAllocated may be valid.
  #8: No counterexample found. clientOneWaitingAndAtLeastOneCompleted may be valid.
  #9: No counterexample found. moreThanOneRidePerDriverOneOfThemNotCompleted may be valid.
  #10: No counterexample found. busyDriverNoCurrentRide may be valid.
  #11: No counterexample found. notSharedRideShared may be valid.
  #12: Instance found. show is consistent.
```

3.8.3 Generated world



4. Appendix

4.1 Used tools

- Microsoft Word 2013 to write this document
- Microsoft Visio 2013 to create UML diagrams
- Balsamiq Mockup 3 to create mockups of the mobile application and of the web site
- Alloy Analyzer 4.2 to describe the structures of the system and explore them

4.2 Attached documents

4.2.1 Client’s document

The problem: myTaxiService

Part I

The government of a large city aims at optimizing its taxi service. In particular, it wants to: i) simplify the access of Clients to the service, and ii) guarantee a fair management of taxi queues.

Clients can request a taxi either through a web application or a mobile app. The system answers to the request by informing the Client about the code of the incoming taxi and the waiting time.

Taxi drivers use a mobile application to inform the system about their availability and to confirm that they are going to take care of a certain call.

The system guarantees a fair management of taxi queues. In particular, the city is divided in taxi zones (approximately 2 km^2 each). Each zone is associated to a queue of taxis. The system automatically computes the distribution of taxis in the various zones based on the GPS information it receives from each taxi. When a taxi is available, its identifier is stored in the queue of taxis in the corresponding zone.

When a request arrives from a certain zone, the system forwards it to the first taxi queuing in that zone. If the taxi confirms, then the system will send a confirmation to the Client. If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.

Besides the specific user interfaces for Clients and Taxi drivers, the system offers also programmatic interfaces to enable the development of additional services (e.g., taxi sharing) on top of the basic one.

Part II

A user can reserve a taxi by specifying the origin and the destination of the ride. The reservation has to occur at least two hours before the ride. In this case, the system confirms the reservation to the user and allocates a taxi to the request 10 minutes before the meeting time with the user.

Part III

A user can enable the taxi sharing option. This means that he/she is ready to share a taxi with others if possible, thus sharing the cost of the ride. In this case the user is required to specify the destination of all rides which he/she wants to share with others. If others are willing to start a shared ride from the same zone going in the same direction, then the system arranges the route for the Taxi driver, defines the fee for all persons sharing the taxi and informs the Clients and the Taxi driver.

4.3 Hours of work

Each member of the group has worked for 47 hours.

5. Revision

This is the 1.1 version of the RASD that contains updates of the document. Below are listed the differences from the previous version.

5.1 Reference errors

- In the paragraph 2.3 User characteristics there was an error in a reference to another subparagraph. The reference to the subparagraph 3.5.4 has been changed with 3.7.4.
- There was a mistake in the number of the subparagraph 3.6.1. We have replaced it with the correct number 3.8.1.
- There were some mistakes in the codes [NFR-M01], [NFR-M02], [NFR-M03] in the subparagraph 3.7.8 Portability. We have replaced them with the correct codes [NFR-P01], [NFR-P02], [NFR-P03].

5.2 Changes in the Scope

- We have replaced the sentence “The taxi chosen from the system will be the nearest to the user that has made the reservation and it’s always chosen starting from the user zone.” with the sentence “The taxi chosen from the system will be chosen from the zone that is nearest to the user that has made the request/reservation and usually it’s his own zone.”

5.3 Changes in the Use cases

- [UC04] and [UC06].
  We have replaced the point “4. The system queries the database containing taxi queues to find the nearest Taxi driver to the Client.” with “4. The system queries the database containing taxi queues to find the Taxi driver in the nearest zone compared to the Client.”.

5.4 Changes in Non Functional Requirements

The following changes have been made to describe in a more objective way some non-functional requirements.

- [NFR-F02].
  We have replaced the sentence “The system shall provide the correct result 99,99% of the times. An incorrect execution (e.g. the farthest taxi is assigned to a Client even though a closer one is available) would nullify the advantages that the system provides.” with “The system shall provide the correct result 99,99% of the times. An incorrect execution (e.g. A taxi driver of an incorrect queue is chosen) would nullify the advantages that the system provides.”.
- [NFR-M03].
  We have replaced the sentence “The system will receive log files about failures of the application containing information necessary that will easily help identifying which component caused the failure (e.g. OS and operation that caused the mobile app to crash).” with “The system will receive log files about failures of the application containing

information required that will help identifying which component caused the failure in at most 1 hour for the 95% of the errors that will occur (e.g. OS and operations that causes the mobile app crash).”.

- [NFR-P02].
  We have replaced the sentence “The mobile application shall be easily installed and uninstalled regardless of the mobile operating system (Android, iOS and Windows Phone) using the proprietary application store.” with “The mobile application should be available in the Google Play Store, Apple Store, Windows Phone Store from which is possible to install it.”.
- [NFR-P03].
  We have replaced the sentence “The software can easily replace the already existent application “mytaxi”.” with “The software could potentially replace the already existent application *mytaxi*.”.

5.5 Changes in Alloy

- The Alloy code has been modified to better show the relations between entities, discarding details about values of fields. The assertions have been rewritten to verify in a more suitable way the model.
- The paragraph 3.8.2 Results has been added to show the results of the assertions.
- The generated world has been modified accordingly to the Alloy code modifications.