POLITECNICO DI MILANO
Computer Science and Engineering
Software Engineering 2 Project

MyTaxiService Requirements Analysis And Specification Document

Authors: Greta Ghiotti
Raffaele Malvermi
Mirco Mutti

Reference Professor: Mirandola Raffaela

Sommario

1. INTRODUCTION

1.1 DESCRIPTION OF THE PROBLEM

The government of a large city aims at optimizing its urban taxi service and, to do that, asks our company to improve it by implementing a system able to manage all the interactions between passengers and taxi drivers.
Indeed, the problem focuses on interactions: the aim of the system is to make passengers able to access the service in a simpler and immediate way; but the system has also to guarantee a fair distribution of the requests among the associated taxi drivers, in order to cover all the city zones and maximize the number of served requests.

The system has to allow the passengers to sign up, request a taxi immediately, booking a taxi, manage all the reservations and share rides with other passengers.
On the other hand, it has to allow the confirmation of the requests by the taxi drivers.
The system will guarantee all these functions only as online operations.

1.2 ASSUMPTIONS

These assumptions aim to make the given specifications more complete, in order to justify the proposed system model.
We assume that:

- If the system searches an available taxi in a zone with an empty queue, then it will try in the contiguous zones;
- Users can inform the system about the departure position by sending it or enabling the gps option;
- The system takes care of the service management until the taxi allocation; after this operation, no other actions depend on the system;
- The "same direction" statement is verified in this way:

- The system considers the current route (we call it CR, just to simplify the explanation) and the shortest route from the departure position to the destination of the new request (we call it NR);
- Then it compares NR with the shortest route that reaches the same destination and contains CR (we call it NR');
- If the attended time of NR' is greater than the attended time of NR + 10% of the same value, the request corresponding to NR is not considered "in the same direction" (the system will allocate another shared taxi);
- Otherwise, the system considers NR' as the new current route and is ready to repeat the procedure;

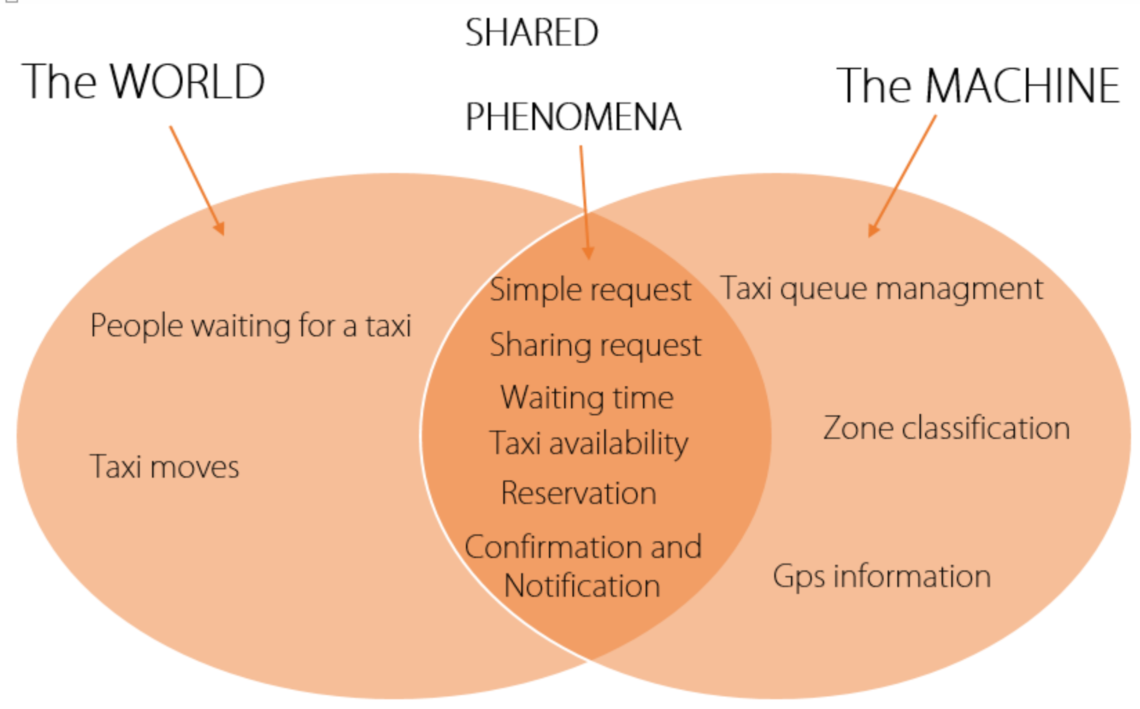
1.3 GLOSSARY

In this paragraph, we propose a list of definitions with the aim of making the dissertation less ambiguous. Indeed, some terms must be explained before their usage in the following pages.

TaxiDriver: he's a taxi driver working at the urban taxi service. He has an ID Code related to his taxi and he hasn't signed up for myTaxiService yet (see chapter 2: "Actors" to analyse his features). After the registration, he changes his status from TaxiDriver to myTaxiDriver
myTaxiDriver: he's a signed up taxi driver who can use the system functionalities (see chapter 2: "Actors" to analyse his features)
Guest: he's a passenger that hasn't signed up for myTaxiService yet. He can only see the sign up page (see chapter 2: "Actors" to analyse his features). After the registration, he changes his status from Guest to User
User: he's a signed up passenger that can use all the offered functionalities (see Chapter 2: " Actors" to analyse his features)
Developer: he's an external person (or company) interested in using system functionalities in other software projects (see Chapter 2: "Actors" to analyse his features)
Taxi: since there's a bijective correspondence between a taxi and its driver, we use this term as a synonymous of TaxiDriver and myTaxiDriver, depending on the context
To pop: we use this verb in its IT meaning. Indeed, is related to the queue behaviour and in particular we use it when the queue extracts the first included taxi (depending on the adopted queueing policy)

ID Code: it’s an alphanumeric code given to a taxi by the urban service. This is unique and since each TaxiDriver has at most one taxi, they have one ID Code
City Zone: to simplify the service management, the city is divided into many zones. Each zone stretches approximately 2 km^2 and has at most one related taxi queue
Queue: we use this term in its IT meaning. Indeed, it’s a data structure abstraction with certain properties affecting the storing management
Request: we mean a specific type of notifications sent by Users to the system. This notification signals User’s needs that the system has to satisfy
Notification: we mean all the messages that the system has to accept, sort and send. There are several types of notifications, that includes requests, confirmations, TaxiDrivers’ notices.
Shared ride: when we use this term, we refer to a ride created just to serve requests with the sharing option enabled. This type of ride is served by a myTaxiDriver who has to go through a specified route that the system has computed before.
Route: it’s the path that links two points in the city. Every myTaxiDriver follows the route defined for a given ride.
Date: it’s part of the request data; it includes the day, the month and the year on which the User wants to take a taxi; it also includes the gathering time
Valid Address: an address is valid if and only if there is a correspondence in the system database
WaitAlgorithm: it’s an algorithm that should compute the waiting time between two points of the city, taking into account several factors (traffic behaviour, route length, average speed)
SharedWaitingAlgorithm: this algorithm should compute the waiting time of all the shared route parts that link the incoming taxi to every User that requests it

1.4 THE WORLD & THE MACHINE



In this paragraph we consider the model “The World & The Machine” by M. Jackson and P. Zave to have a first analysis of our problem domain.
In the world phenomena we have “taxi moves” and “people waiting for a taxi”: these events are related with the problem but they aren’t controlled by the system. Surely, when the System confirms a request, it can imagine that in the world there’s a User (the requestor) who is waiting for a taxi and then a taxi is moving to him, but it doesn’t manage these events.
On the other hand we have “taxi queues management”, “zone classification”, “gps information” that are hidden to all actors in the problem except the Machine itself; “zone classification”, for example, is essential for the System functionalities, but certainly it isn’t significant for the Users.
In the middle, we consider a set of phenomena that we can’t relate only to the World or the Machine exclusively: they are shared phenomena. The events “Simple Request”, “Sharing Request”, “Reservation” occur in the world but, of course, are also observed by the machine. Otherwise, the occurrences of “confirmation and notification” and “waiting time” are controlled by the Machine but Users and MyTaxiDrivers are certainly interested on them, so we could say that the world observes these phenomena.

All these considerations are the base of the formalization we will explain in the following paragraphs in relation with System goals and domain properties, and also with the requirements elicitation.

1.5 GOALS

In the following paragraph, we want to show a list of goals that we have obtained by studying the World & Machine Model.
These goals are the stakeholders' needs that the system has to satisfy; for this reason, they cover all the aspects and represent the system as is to be:

(G1): A Guest can sign up to the system, using the corresponding app;
(G2): a registered Guest (or User) can log into the system;
(G3): a logged User can request a taxi at any time, for a number of passengers included in the range [1,4], always receiving an answer from the system;
(G4): a logged User is able to booking a taxi for a number of passengers contained in the range [1,4], specifying the date, doing this until two hours from the inserted time and always receiving a positive or negative answer from the system;
(G5): a logged User can delete one of his reservations until 10 minutes from the ride time;
(G6): a User is able to ask the system to share the ride with other passengers;
(G7): a TaxiDriver can sign up into the system, using the corresponding app;
(G8): a registered TaxiDriver (or MyTaxiDriver) is able to declare his availability and that he's ready to receive requests from the system;
(G9): a registered TaxiDriver (or MyTaxiDriver) is able to accept or deny any received requests;
(G10): a fair queue management is guaranteed;
(G11): system functionalities should be extendable for other projects

1.6 DOMAIN PROPERTIES

The study of the World & The Machine Model help us to define not only the goals, but also the assumption concerning the properties of the world affected by the system.

These are described in the following lines:

(D1): a Guest fills the form with valid information once during the registration phase: these information include correct personal data and an existing mail address that actually belongs to him;
(D2): a Guest, who wants to sign up, hasn't signed up yet;
(D3): an algorithm that is able to compute the attending time between two given positions (described by their coordinates), considering the traffic conditions and other possible environmental variables, is available;
Notice: with D3, we mean the existence in the world of the knowledge necessary to implement this kind of algorithm and so we think D3 as a domain property and not as a functional requirement
(D4): a User who requests a taxi must wait it, in any case;
(D5): Users must wait requested taxi staying near the inserted departure address or in the current position if the gps option is enabled;
(D6): urban taxi service drivers have cars with 4 seats;
(D7): a User doesn't book two taxi for the same date;
(D8): a User who books a taxi must be in time with respect to reservation;
(D9): Every TaxiDriver has an ID Code, linked to his taxi, and this is unique;
(D10): an available TaxiDriver always waits for a request;
(D11): a TaxiDriver is available if and only if he isn't serving any passengers;
(D12): It is always possible to translate gps data in the corresponding zones;

1.7 PROPOSED SYSTEM

The software that we aim to develop should be a distributed system in which all the functionalities will be accessible from two interfaces: one for passengers, that can be web-oriented or mobile-oriented, and another for drivers, available only on mobile devices.

The system should be able to store many types of requests and generate different pages, depending on the needed functionality.
It also will be able to compute served request, shared routes and shared fees.
The system should modeling the city and the taxi distribution in terms of zones and taxi queue, to be able to manage them in order to guarantee the service fairness.
It also should be able to make the software extendable and to make its functionalities available to other software projects.

1.8 IDENTIFY STAKEHOLDERS

Our main stakeholder is certainly the Professor, who gave us the assignment and who is also the only real recipient of this Requirements Analysis and Specification Document. However the assignment describes a situation in which there is an imaginary committee that asks to our (imaginary) organization to develop an application with some functionalities; in according to this scenario we could assume that the city government is the main stakeholder for the project.
Indeed the government is the committee for our system myTaxiService: it needs to improve the management of its urban taxi service by simplifying the access of the passengers and by guaranteeing a fair distribution of the requests for each taxi driver.
The urban taxi service users are important stakeholders for this project: they asks to our system to be able to access the service in the easy way as possible.
The taxi drivers of the city are also stakeholders: their main need is to have a manager for the taxi queues that guarantees a fair distribution of the potential clients; they asks to myTaxiService to be this manager.
We could also include in the stakeholders set the developers who wants to import the functionalities of our to-be-developed system in their application. The system has to provide a suit of API to satisfy this need.

2. ACTORS

We have considered these five actors interested in then system:

- Guest: is a potential user who hasn't signed up to the application yet. He basically can only access to the application (web or mobile) and fill the registration form to sign up to myTaxiService
- User: is a guest who has just signed up to the application. He has to be able to log in myTaxiService and so to have access to the main functionalities, such as to request a taxi and to reserve a taxi for a specific date.
- TaxiDriver: is someone who has got a license of the urban taxi service, but he hasn't signed up to myTaxiService yet. He can access the application dedicated to taxi drivers and fill the registration form to sign up.
- MyTaxiDriver: is a taxi driver who has just signed up. He can log into taxi drivers application and so he can declare his availability and he can receive request by myTaxiService users from the system.
- Developer: is someone who has interested in import functionalities of myTaxiService in his own application. He has to be able to access a suit of API.

3. REQUIREMENTS

After the presentation of the model that we propose to analyse the context in which the system will work, and after having found the stakeholders' needs that the system has to satisfy, we want to determine the requirements.
We have derived them starting from goals and taking into account the related domain properties.
Therefore, in this paragraph, the requirements are listed with the corresponding goals.
Instead, we classify the requirements in functional, non functional, constraints in the following paragraphs.

1. A Guest can sign up to the system, using the corresponding app
    - D1: A Guest fills the form with valid information once during the registration phase: these information include correct personal data and an existing mail address that actually belongs to him
    - D2: A Guest, who wants to sign up, hasn't signed up yet
    - R1: The System should provide to the Guest a graphic interface containing a sign up form
    - R2: The System should accept a sign up request only if the Guest has filled all the mandatory fields in the sign up form
2. A registered Guest (or User) can log into the system
    - R1: The System should provide to the Guest a graphic interface containing a log in form
    - R2: The System should accept a log in request only if the User has inserted into the log in form a couple (mail, password) that match with the same couple of a corresponding registered User
3. A logged User can request a taxi at any time, for a number of passengers included in the range [1,4], always receiving an answer from the system
    - D3: An algorithm that is able to compute the attending time between two given positions (described by their coordinates), considering the traffic conditions and other possible environmental variables, is available
    - D4: A User who requests a taxi must wait it, in any case
    - D5: Users must wait requested taxi staying near the inserted departure address or in the current position if the gps option is enabled
    - D6: Urban taxi service drivers have cars with 4 seats
    - R1: The System has to be available 24h per day, 7 days per week
    - *Notice: R1 doesn't want to be a requirement about system reliability but it should be seen as an availability specification: it means that, when the system is working, it must accept requests at any time*
    - R2: The System should consider a taxi request only if the User has inserted:
        i. A valid departure address or, in alternative, has enabled the gps option for the current position
        ii. Number of passengers in the range [1,4]
    - R3: The System should give a positive answer to a request only if a MyTaxiDriver has accepted to take care of it

- R4: The System should give a positive answer to a request only if the position of the MyTaxiDriver, that has accepted to take care of it, corresponds to a waiting time less or equal thirty minutes
- R5: The System giving a positive answer to a request has to notify waiting time and ID Code of the coming taxi to the requesting User
- R6: The System should give a negative answer to a request if there isn't a MyTaxiDriver who has accepted to taking care of it
- R7: The System should give a negative answer to a request if the position of the MyTaxiDriver, that has accepted to take care of it, corresponds to a waiting time greater than thirty minutes
- R8: The System that receives a request has to answer in, at least, five minutes

4. A logged User is able to booking a taxi for a number of passengers contained in the range [1,4], specifying the date, doing this until two hours from the inserted time and always receiving a positive or negative answer from the system
    - D7: A User doesn't book two taxi for the same date
    - D8: A User who books a taxi must be in time with respect to reservation
    - D6: Urban taxi service drivers have cars with 4 seats
    - R1: The System should provide a graphic interface that allow Users to request a reservation
    - R2: The System has to consider a reservation request only if the User ha inserted: departure time, departure address, destination address, number of passengers in the range [1,4]
    - R3: The System has to consider a reservation request only if the request has valid departure address and destination address
    - R4: The System should: accept a reservation request only if the request is submitted at least two hours earlier than the departure time
    - R5: The System reserve a taxi ten minutes earlier than the departure time
    - R6: The System that receives a reservation request has to answer in, at least, one minute
5. A logged User can delete one of his reservations until 10 minutes from the ride time
    - R1: The System should provide a graphic interface that allow Users to access his list of hanging reservations
    - R2: The System should allow Users to delete his reservation until 10 minutes before the ride time
6. A User is able to ask the system to share the ride with other passengers

- D6: Urban taxi service drivers have cars with 4 seats;
- R1: The system allows Users to select the sharing option on every type of requests (simple or reservation), in order to enable the sharing of taxi rides
- R2: The system reserves the same taxi for one or more Users having done a request with active sharing option if and only if they want to leave from the same zone and they are headed to destinations that are in the same direction (see the "same direction statement" assumption)
- R3: The system adds further Users to a shared ride if and only if the related taxi has enough free seats
- R4: The system adds further Users to a shared ride if and only if the myTaxiDriver who takes care of it hasn't declared the departure to the system yet
- R5: the system has to compute the route of the shared ride and notifies the myTaxiDriver about it
- R6: the system computes the total cost of the shared ride and all the partial fees that each User has to pay
- R7: the system sends the payment information to the myTaxiDriver and all the Users sharing the ride

7. A TaxiDriver can sign up into the system, using the corresponding app
   - D9: Every TaxiDriver has an ID Code, linked to his taxi, and this is unique
   - R1: the system displays a sign up form and lets the TaxiDriver view it
   - R2: the system approves the registration if and only if the ID Code is valid and it doesn't belong to another TaxiDriver
8. A registered TaxiDriver (or MyTaxiDriver) is able to declare his availability and that he's ready to receive requests from the system;
   - D10: An available TaxiDriver always waits for a request
   - D11: A TaxiDriver is available if and only if he isn't serving any passengers
   - R1: the system provides an interface that allows a myTaxiDriver to signal his availability
   - R2: When a myTaxiDriver declares his availability, the system add his taxi ID Code in the last position of the taxi queue related to the zone in which he's located
9. A registered TaxiDriver (or MyTaxiDriver) is able to accept or deny any received requests
   - R1: the system sends requests only to available myTaxiDrivers

- R2: the system informs the available myTaxiDrivers about new requests by sending notices to them
- R3: the system provides an interface that allows myTaxiDrivers to accept or deny a received request
- R4: the system lets a myTaxiDriver delete a request only if it's not a reservation request

10. A fair queue management is guaranteed
    - D12: It is always possible to translate gps data in the corresponding zones
    - R1: When a myTaxiDriver declares his availability, the system adds his taxi ID Code in the last position of the taxi queue related to the zone in which he's located
    - R2: When a request occurs, the system sends it to the first taxi stored in the queue, where the queue is related to the same zone to which the request belongs; then the system wait for his answer
    - R3: If a myTaxiDriver doesn't answer to the request within 1 minute from its notification, the system moves him to the last position in the queue
    - R4: If a myTaxiDriver confirms a request, the system pops from the queue
    - R5: If a myTaxiDriver rejects a request or doesn't answer to it within 1 minute, the system moves him to the last position in the queue and repeat the procedure (see the requirement R2)
11. System functionalities should be extendable for other projects
    - R1: The system offers a list of APIs that lets every developer add the system functionalities to his software projects

3.1 FUNCTIONAL REQUIREMENTS

- The System should provide to the Guest a graphic interface containing a sign up form
- The System should accept a sign up request only if the Guest has filled all the mandatory fields in the sign up form
- The System should provide to the Guest a graphic interface containing a log in form

- The System should accept a log in request only if the User has inserted into the log in form a couple (mail, password) that match with the same couple of a corresponding registered User
- The System should consider a taxi request only if the User has inserted:
  i) A valid departure address or, in alternative, has enabled the gps option for the current position
  ii) Number of passengers in the range [1,4]
- The System should give a positive answer to a request only if a MyTaxiDriver has accepted to take care of it
- The System should give a positive answer to a request only if the position of the MyTaxiDriver, that has accepted to take care of it, corresponds to a waiting time less or equal thirty minutes
- The System giving a positive answer to a request has to notify waiting time and ID Code of the coming taxi to the requesting User
- The System should give a negative answer to a request if there isn't a MyTaxiDriver who has accepted to taking care of it
- The System should give a negative answer to a request if the position of the MyTaxiDriver, that has accepted to take care of it, corresponds to a waiting time greater than thirty minutes
- The System should provide a graphic interface that allow Users to request a reservation
- The System has to consider a reservation request only if the User ha inserted: departure time, departure address, destination address, number of passengers in the range [1,4]
- The System has to consider a reservation request only if the request has valid departure address and destination address
- The System should: accept a reservation request only if the request is submitted at least two hours earlier than the departure time
- The System reserve a taxi ten minutes earlier than the departure time
- The System should provide a graphic interface that allow Users to access his list of hanging reservations
- The System should allow Users to delete his reservation until 10 minutes before the ride time
- The system allows Users to select the sharing option on every type of requests (simple or reservation), in order to enable the sharing of taxi rides

- The system reserves the same taxi for one or more Users having done a request with active sharing option if and only if they want to leave from the same zone and they are headed to destinations that are in the same direction (see the “same direction statement” assumption)
- The system adds further Users to a shared ride if and only if the related taxi has enough free seats
- The system adds further Users to a shared ride if and only if the myTaxiDriver who takes care of it hasn’t declared the departure to the system yet
- The system has to compute the route of the shared ride and notify the myTaxiDriver about it
- The system computes the total cost of the shared ride and all the partial fees that each User has to pay
- The system sends the payment information to the myTaxiDriver and all the Users sharing the ride
- The system displays a sign up form and lets the TaxiDriver view it
- The system approves the registration if and only if the ID Code is valid and it doesn’t belong to another TaxiDriver
- The system provides an interface that allows a myTaxiDriver to signal his availability
- When a myTaxiDriver declares his availability, the system add his taxi ID Code in the last position of the taxi queue related to the zone in which he’s located
- The system sends requests only to available myTaxiDrivers
- The system informs the available myTaxiDrivers about new requests by sending notices to them
- The system provides an interface that allows myTaxiDrivers to accept or deny a received request
- The system lets a myTaxiDriver delete a request only if it’s not a reservation request
- When a myTaxiDriver declares his availability, the system adds his taxi ID Code in the last position of the taxi queue related to the zone in which he’s located
- When a request occurs, the system sends it to the first taxi stored in the queue, where the queue is related to the same zone to which the request belongs; then the system wait for his answer
- If a myTaxiDriver doesn’t answer to the request within 1 minute from its notification, the system moves him to the last position in the queue
- If a myTaxiDriver confirms a request, the system pops from the queue

- If a myTaxiDriver rejects a request or doesn't answer to it within 1 minute, the system moves him to the last position in the queue and repeat the procedure (see the requirement R2)

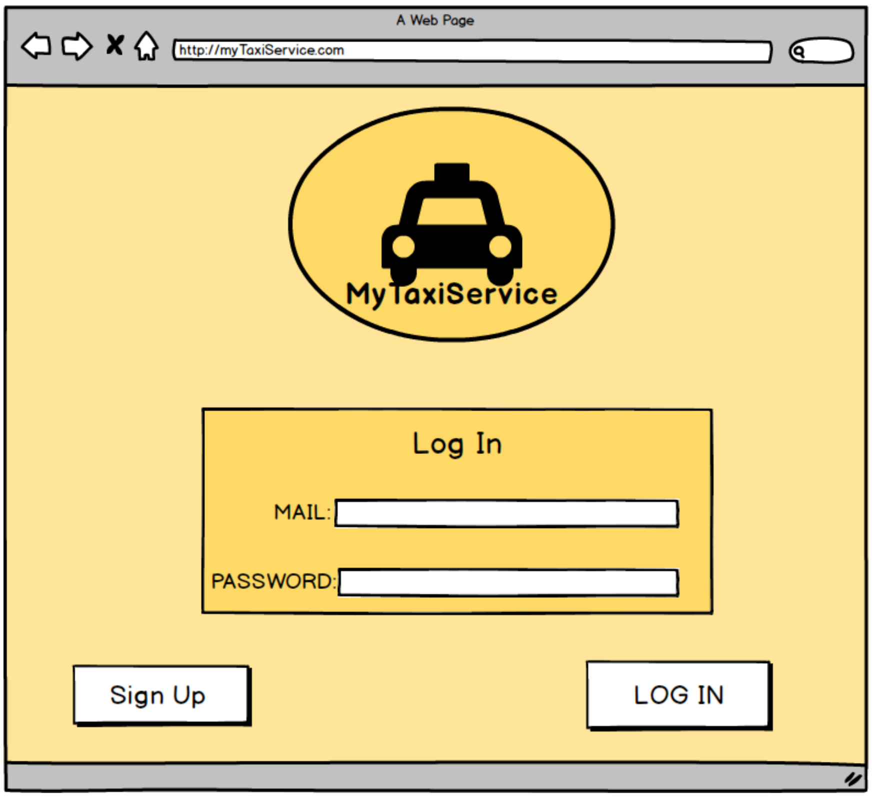
3.2 NON FUNCTIONAL REQUIREMENTS

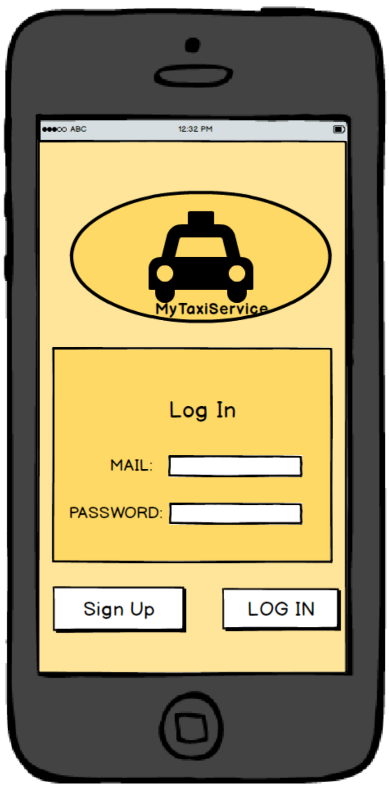
- The System has to be available 24h per day, 7 days per week
- The System that receives a request has to answer in, at least, five minutes
- The System that receives a reservation request has to answer in, at least, one minute

3.2.1 USER INTERFACE

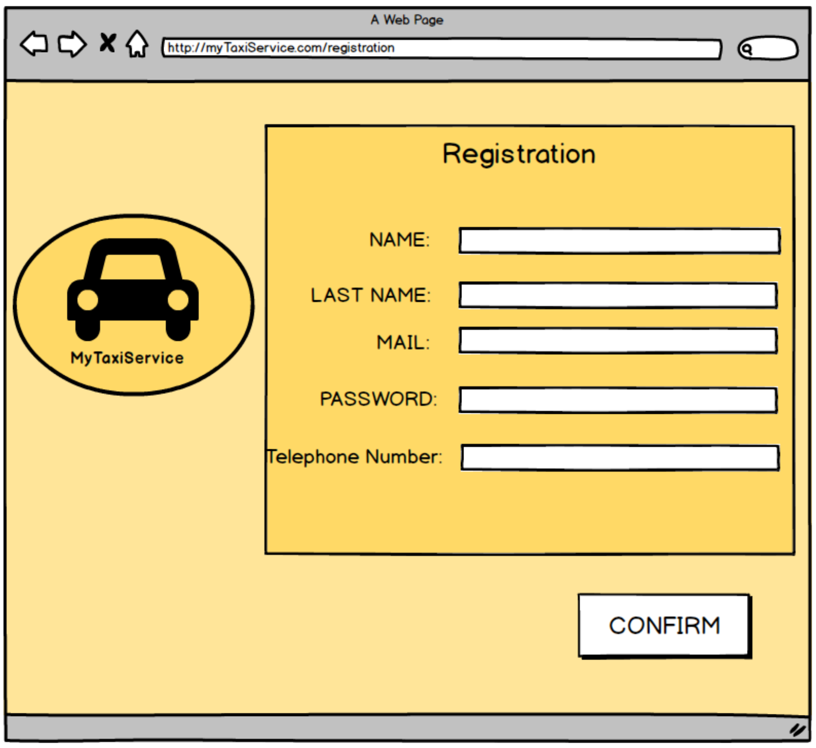
Users can access MyTaxiService through a web Application or a Mobile Application.

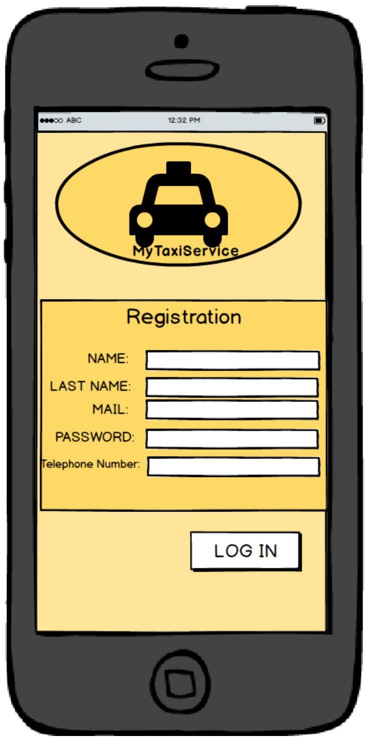
➔ When Users open the application or the web page, the first loaded page is the home Page, where they can access the application and fill the Log In form.



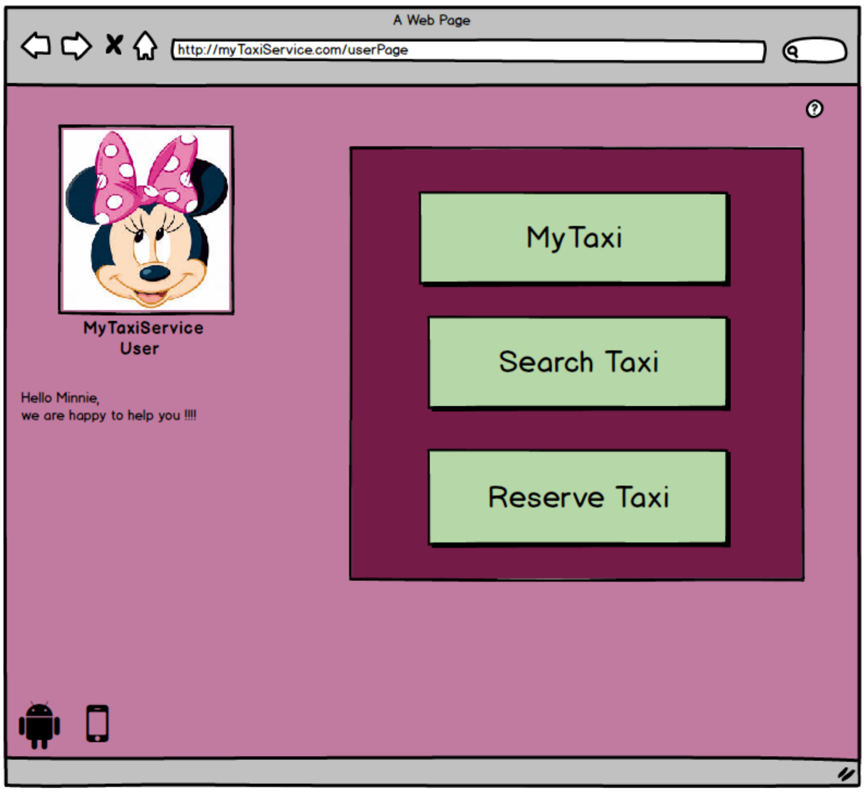


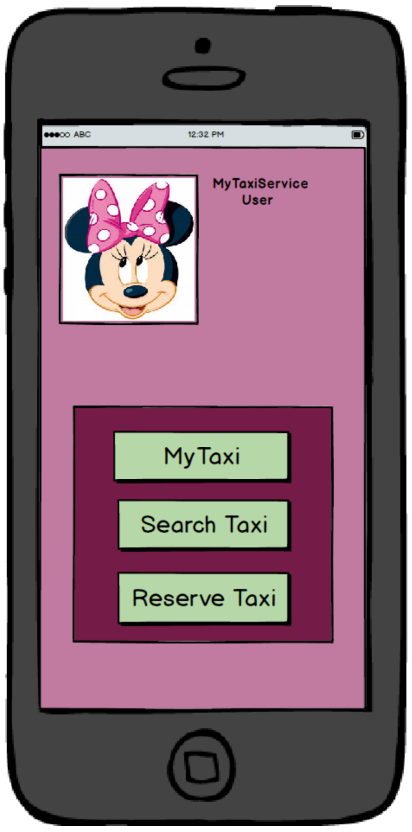
➔ If Users haven't signed up yet, they can open the Sign Up Page by clicking on the Sign Up Button:



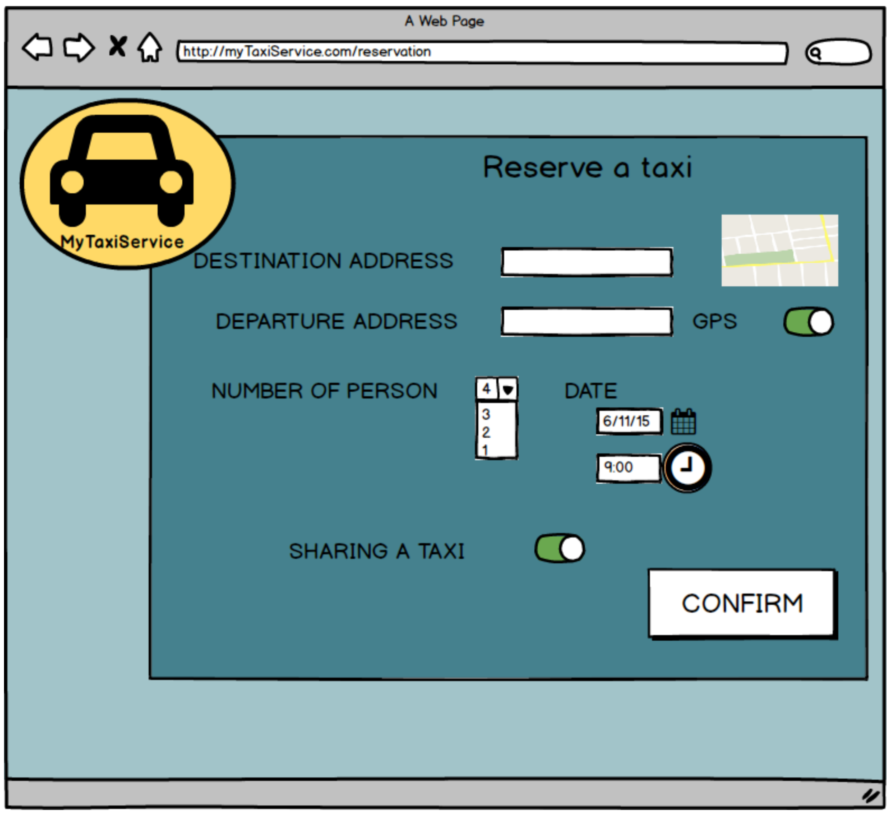


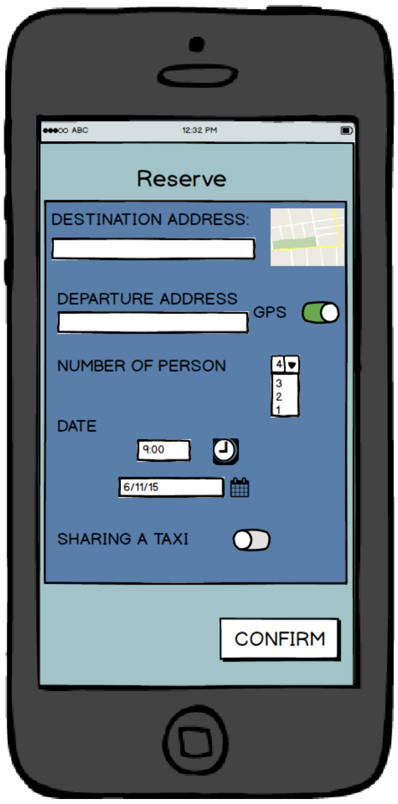
➔ The most important page is the Profile Page, where Users can request or reserve a taxi, and view their reservations.





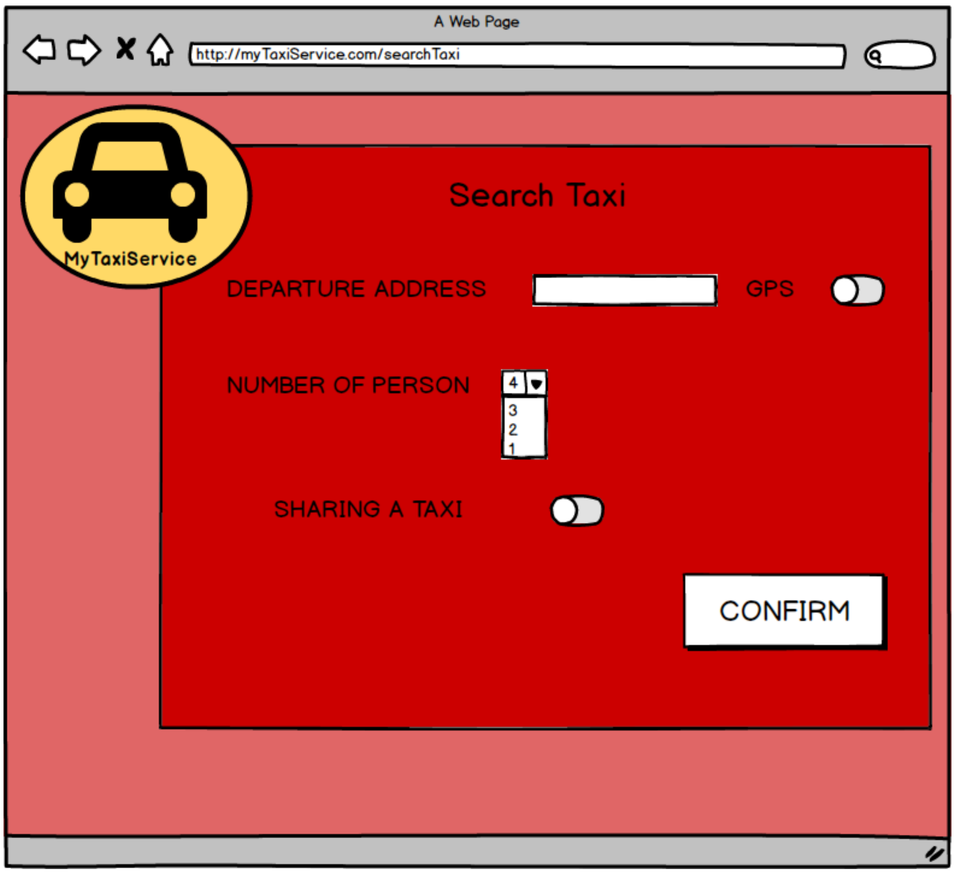
➔ These images show how to reserve a taxi:

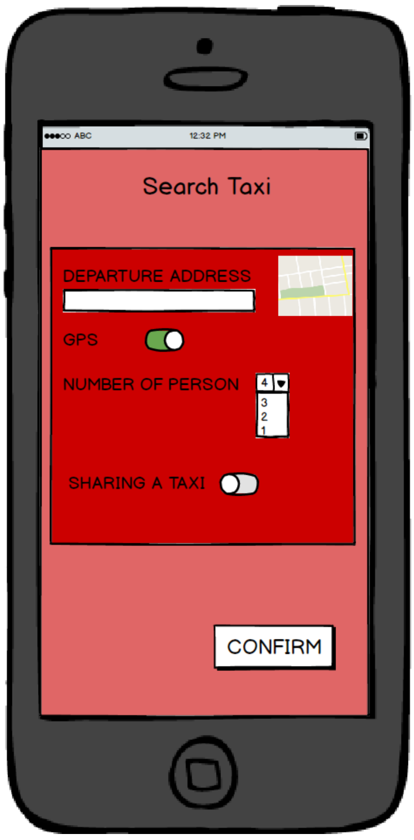




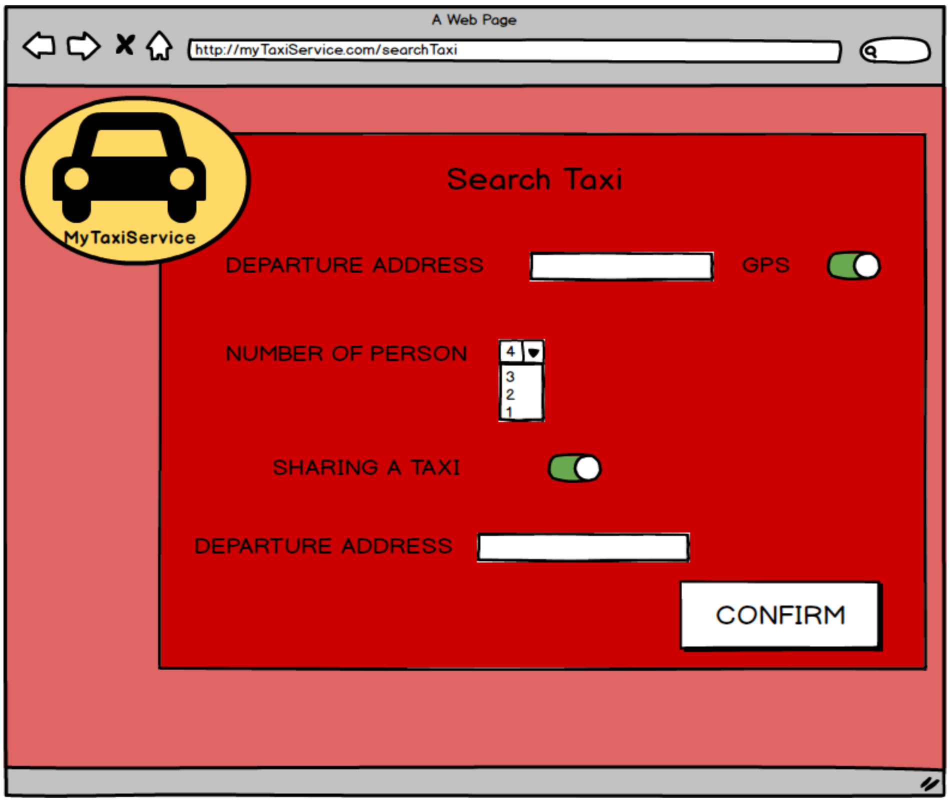
➔ These images show how to search available taxi:

SEARCH A TAXI NO SHARING OPTION





SEARCH A TAXI WITH SHARING OPTION:



A Web Page

http://myTaxiService.com/searchTaxi

MyTaxiService

Search Taxi

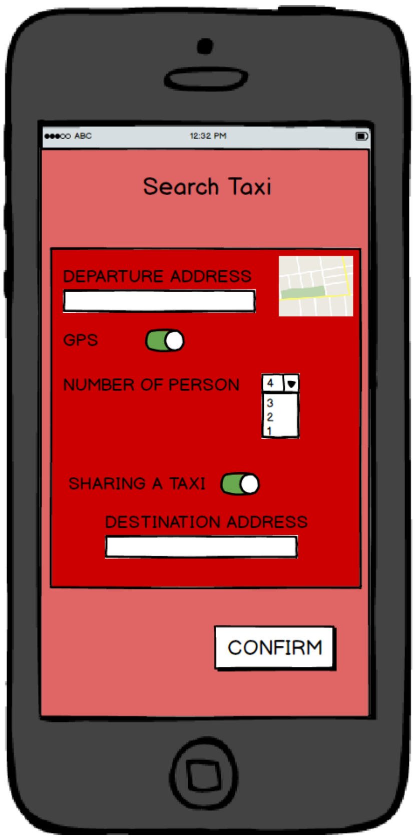
DEPARTURE ADDRESS GPS

NUMBER OF PERSON 4 3 2 1

SHARING A TAXI

DEPARTURE ADDRESS

CONFIRM



●●●○○ ABC 12:32 PM

Search Taxi

DEPARTURE ADDRESS

GPS

NUMBER OF PERSON 4 3 2 1

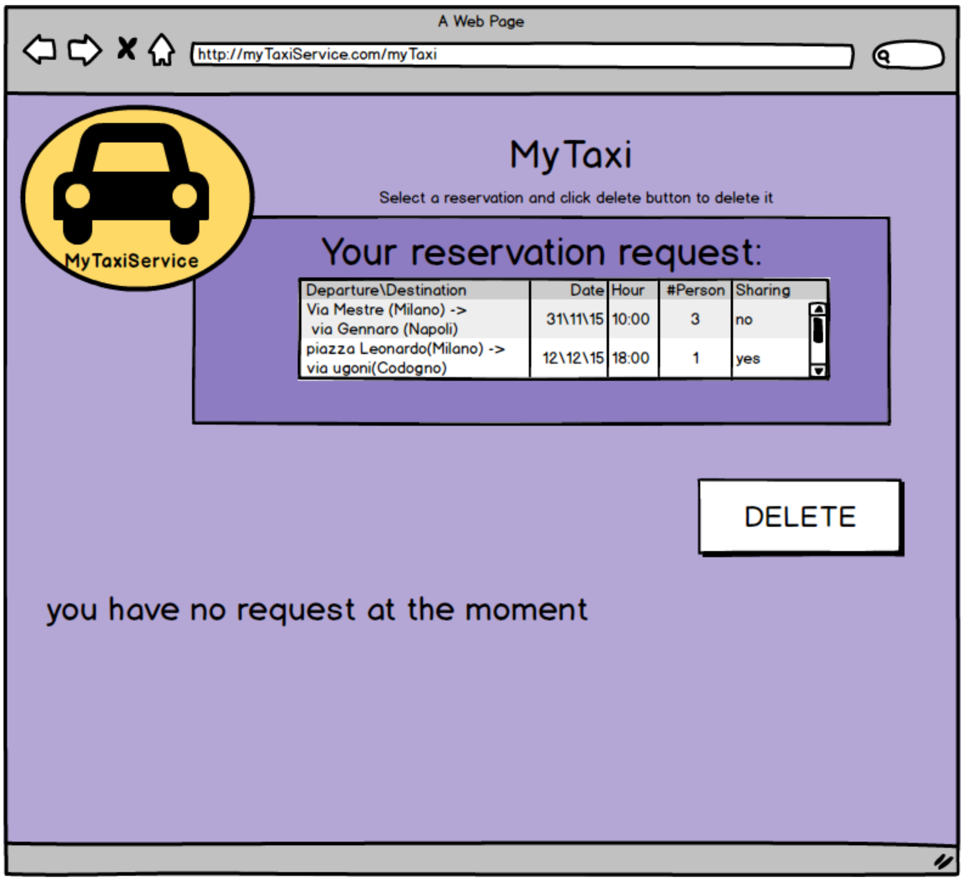
SHARING A TAXI

DESTINATION ADDRESS

CONFIRM

➔ Users can click on the MyTaxi Button to see done reservations.

The system shows this page:



A Web Page

http://myTaxiService.com/myTaxi

MyTaxiService

My Taxi

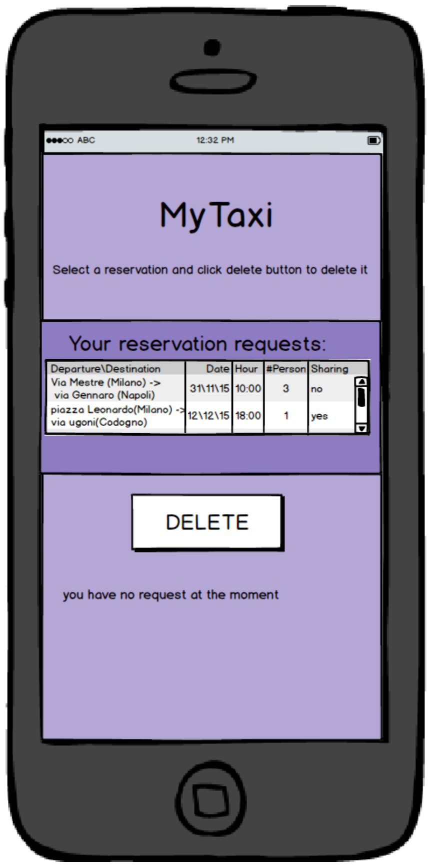
Select a reservation and click delete button to delete it

Your reservation request:

Departure\Destination	Date	Hour	#Person	Sharing
Via Mestre (Milano) -> via Gennaro (Napoli)	31\11\15	10:00	3	no
piazza Leonardo(Milano) -> via ugoni(Codogno)	12\12\15	18:00	1	yes

DELETE

you have no request at the moment



●●●○○ ABC 12:32 PM

My Taxi

Select a reservation and click delete button to delete it

Your reservation requests:

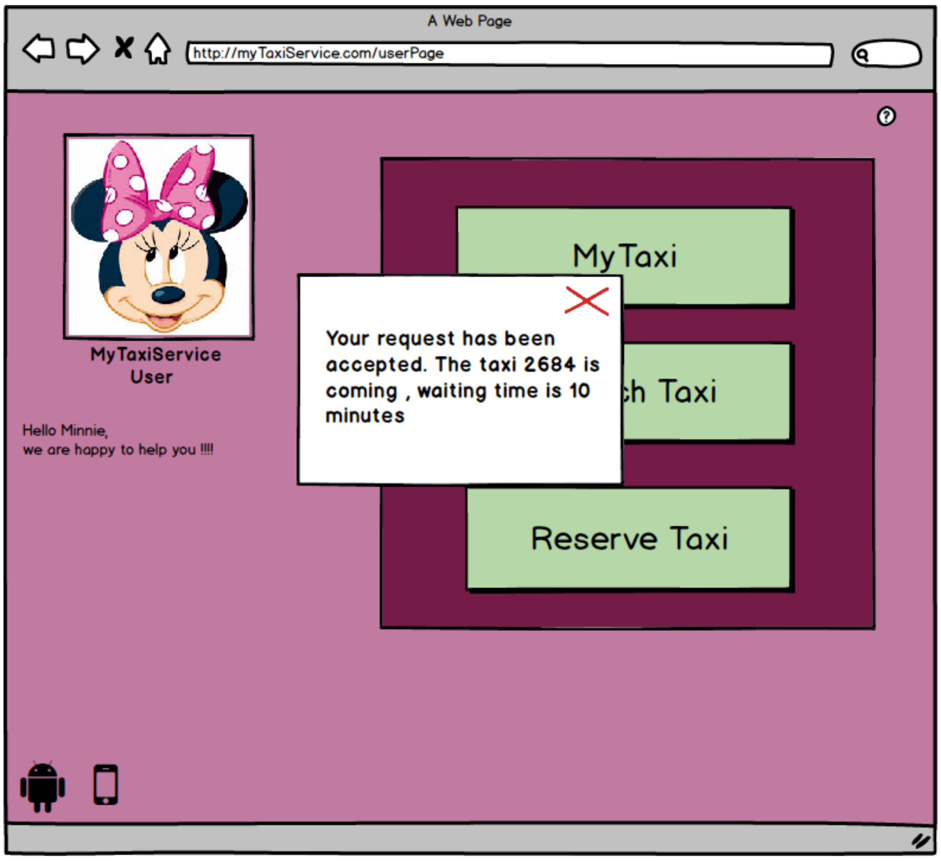
Departure\Destination	Date	Hour	#Person	Sharing
Via Mestre (Milano) -> via Gennaro (Napoli)	31\11\15	10:00	3	no
piazza Leonardo(Milano) -> via ugoni(Codogno)	12\12\15	18:00	1	yes

DELETE

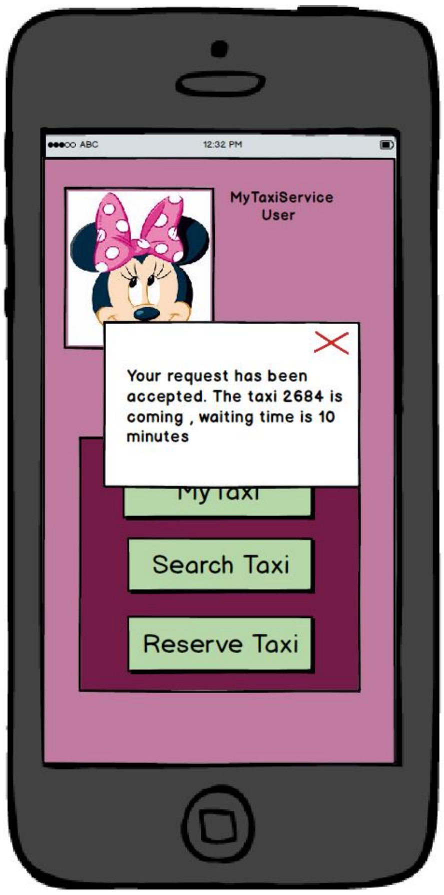
you have no request at the moment

Now we want to analyse notification messages, we decided not to focus on error messages because they appear on the screen in the same way

➔ Notification about confirmation to taxi request:

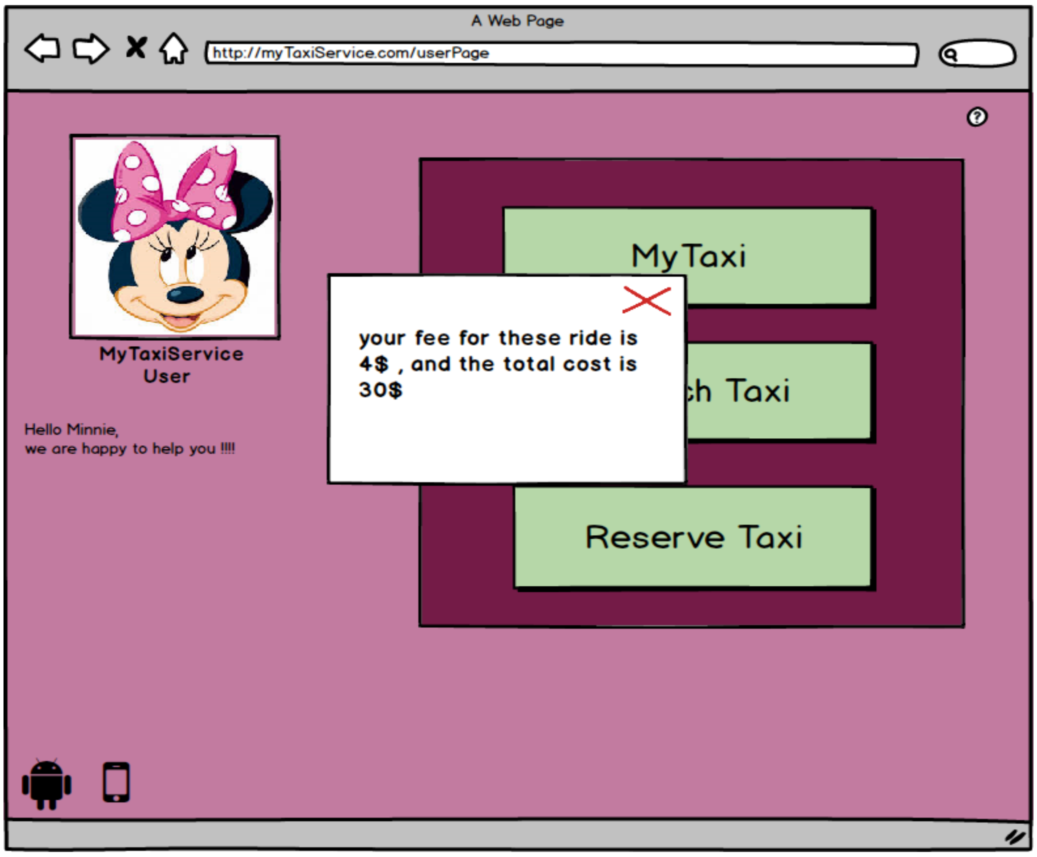


Your request has been accepted. The taxi 2684 is coming , waiting time is 10 minutes



Your request has been accepted. The taxi 2684 is coming , waiting time is 10 minutes

➔ Notification with total cost and fees during sharing ride:



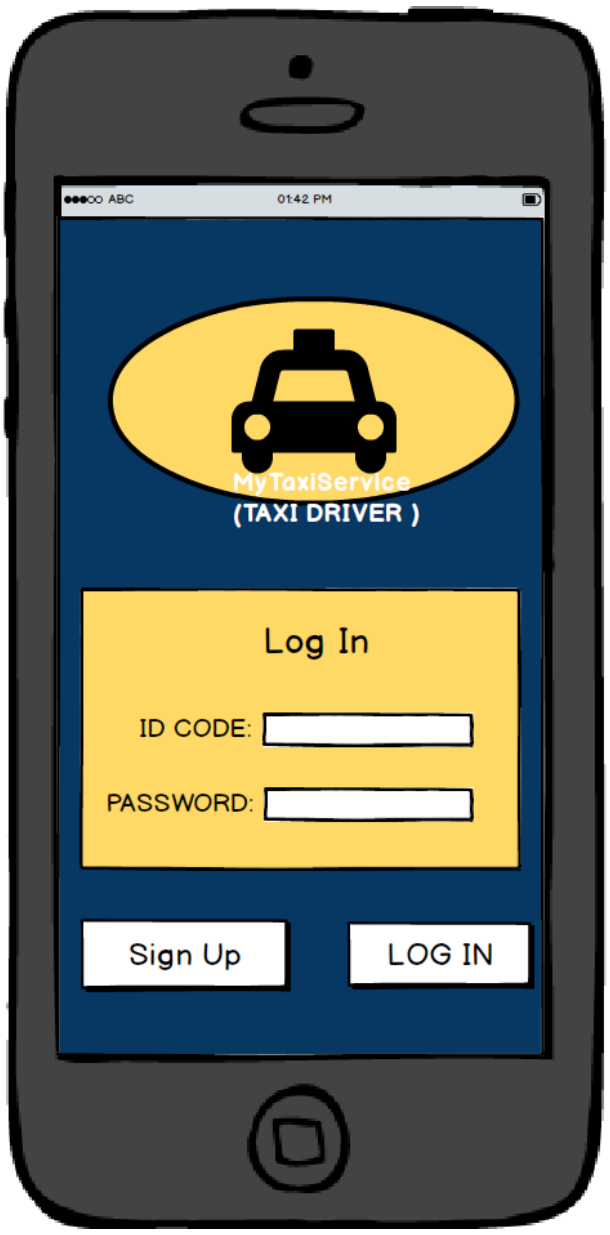
Hello Minnie,
we are happy to help you !!!!

your fee for these ride is 4$, and the total cost is 30$

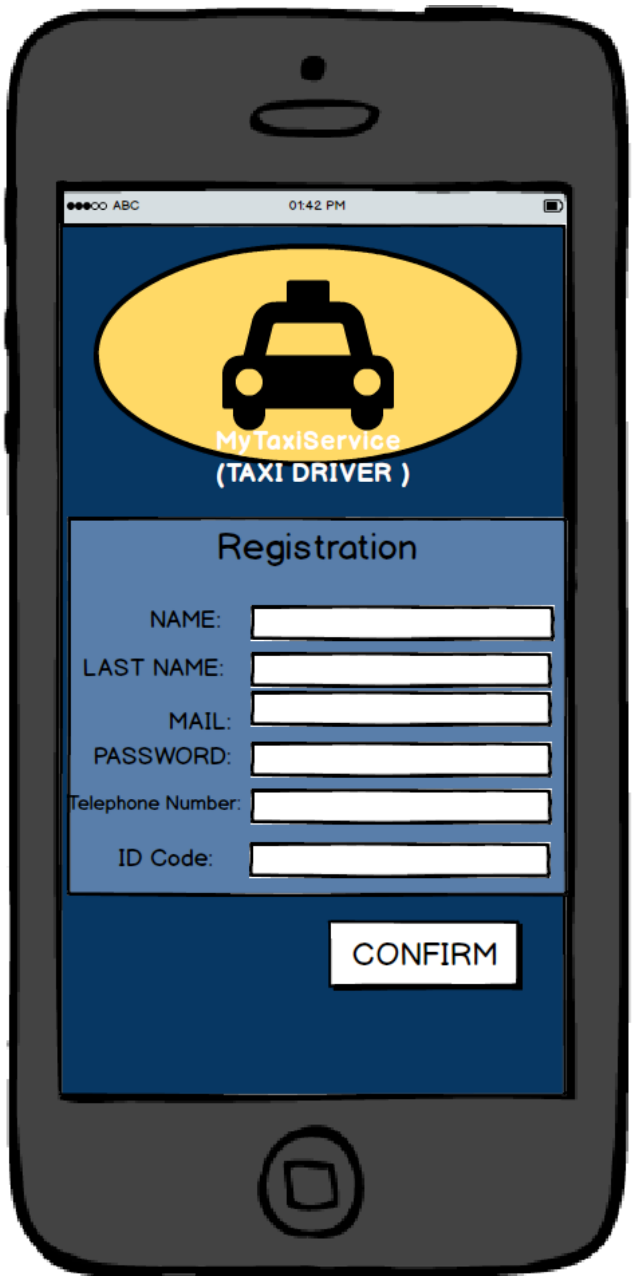
3.2.1 TAXI DRIVER INTERFACE

In These images, we analyse the Taxi Driver Interface;

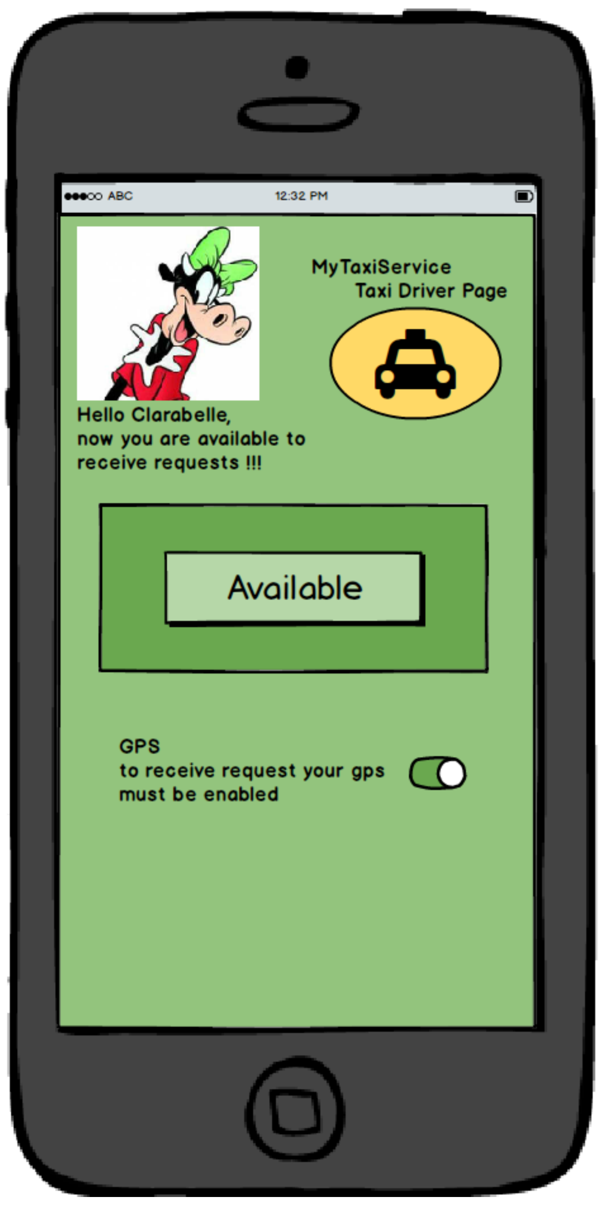
->Log in Page for taxi Driver :



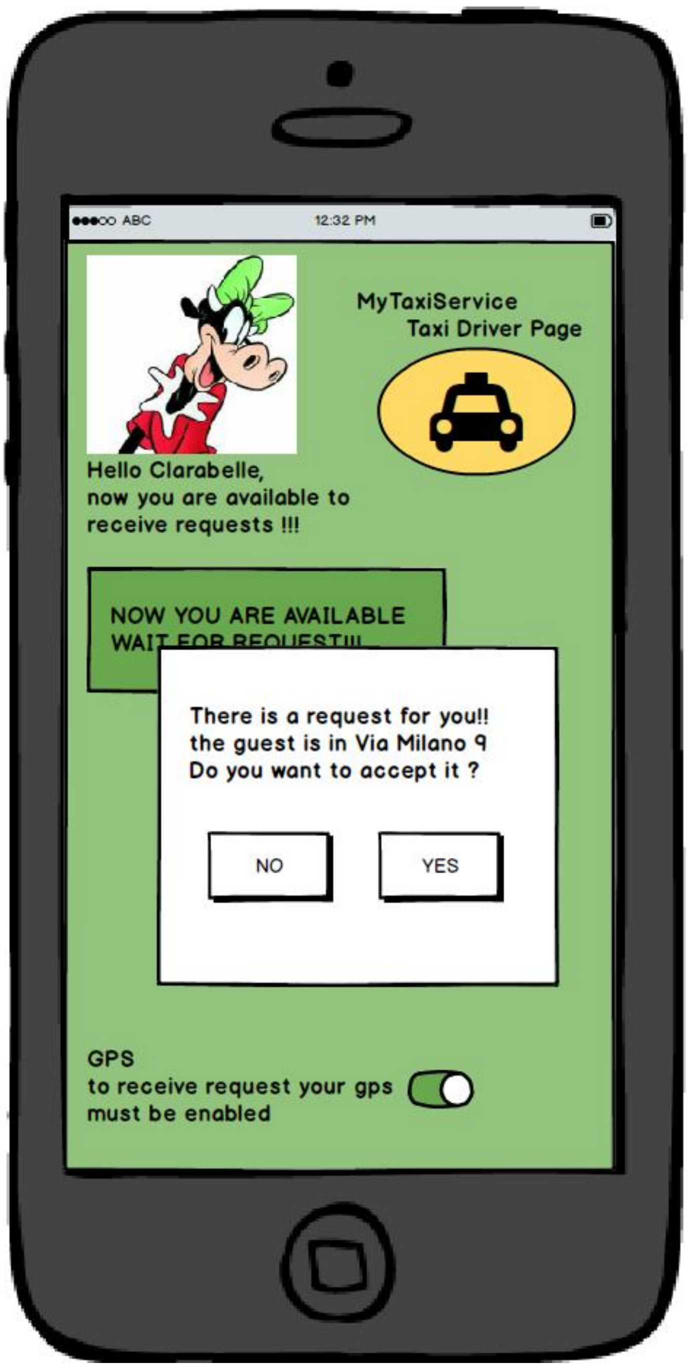
->Sign Up Taxi Driver:



➔ Taxi Driver application Home Page, when he/she is available, he/she clicks the corresponding button



->Notifications appear when myTaxiDrivers receive a request or a reservation

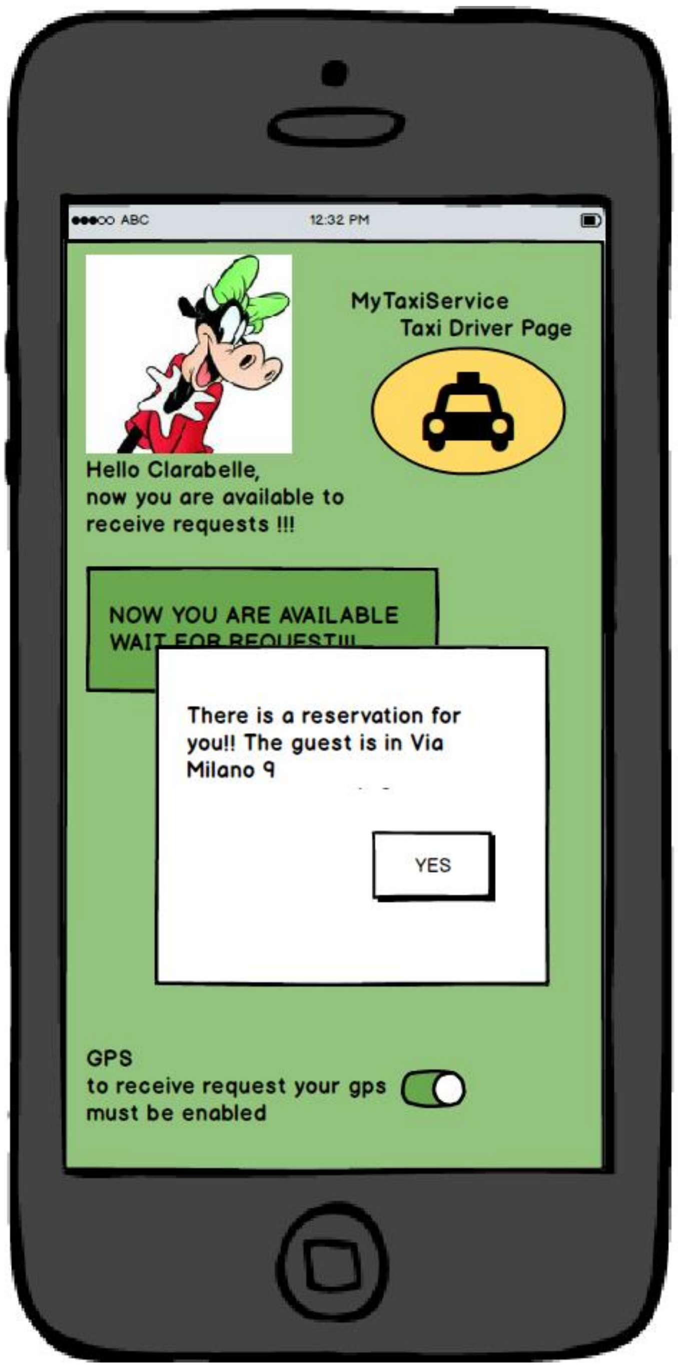


Hello Clarabelle,
now you are available to
receive requests !!!

NOW YOU ARE AVAILABLE
WAIT FOR REQUEST!!!

There is a request for you!!
the guest is in Via Milano 9
Do you want to accept it ?

to receive request your gps
must be enabled



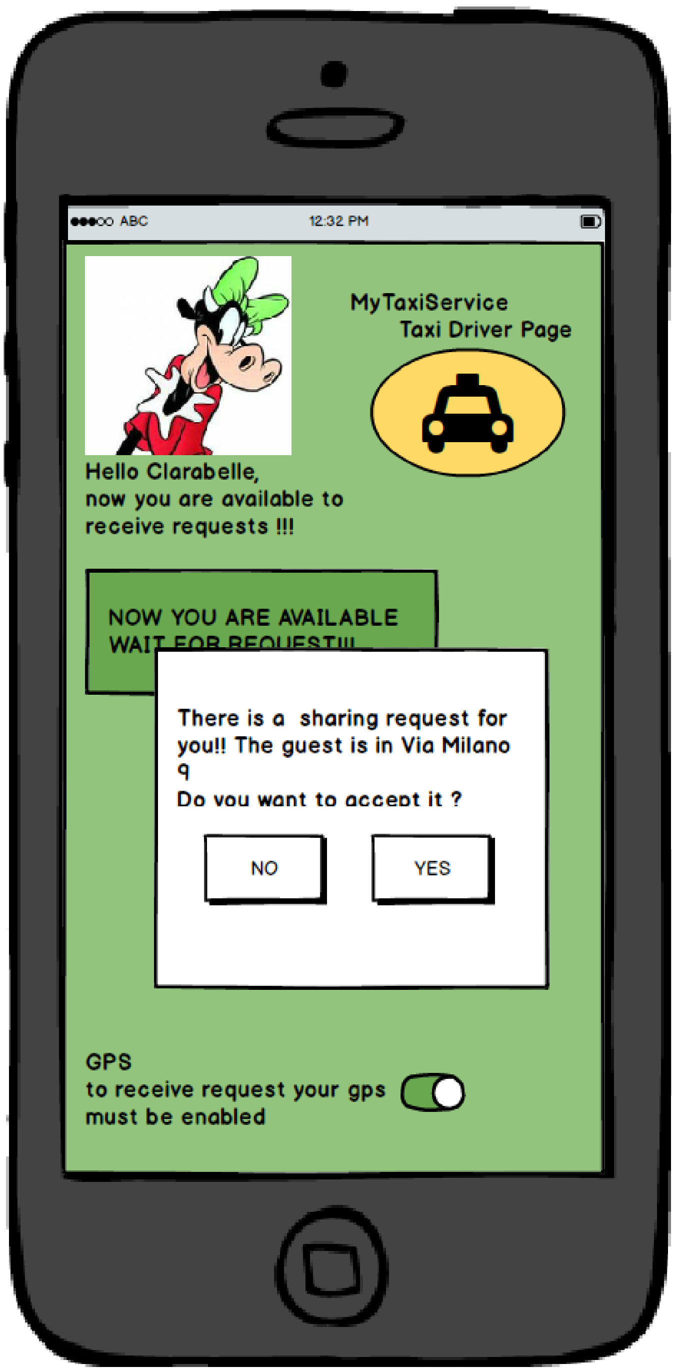
Hello Clarabelle,
now you are available to
receive requests !!!

NOW YOU ARE AVAILABLE
WAIT FOR REQUEST!!!

There is a reservation for
you!! The guest is in Via
Milano 9

to receive request your gps
must be enabled

->when the Taxi driver receives a Sharing request and he accepts it, a page with the map, and the departure button is shown on the screen.

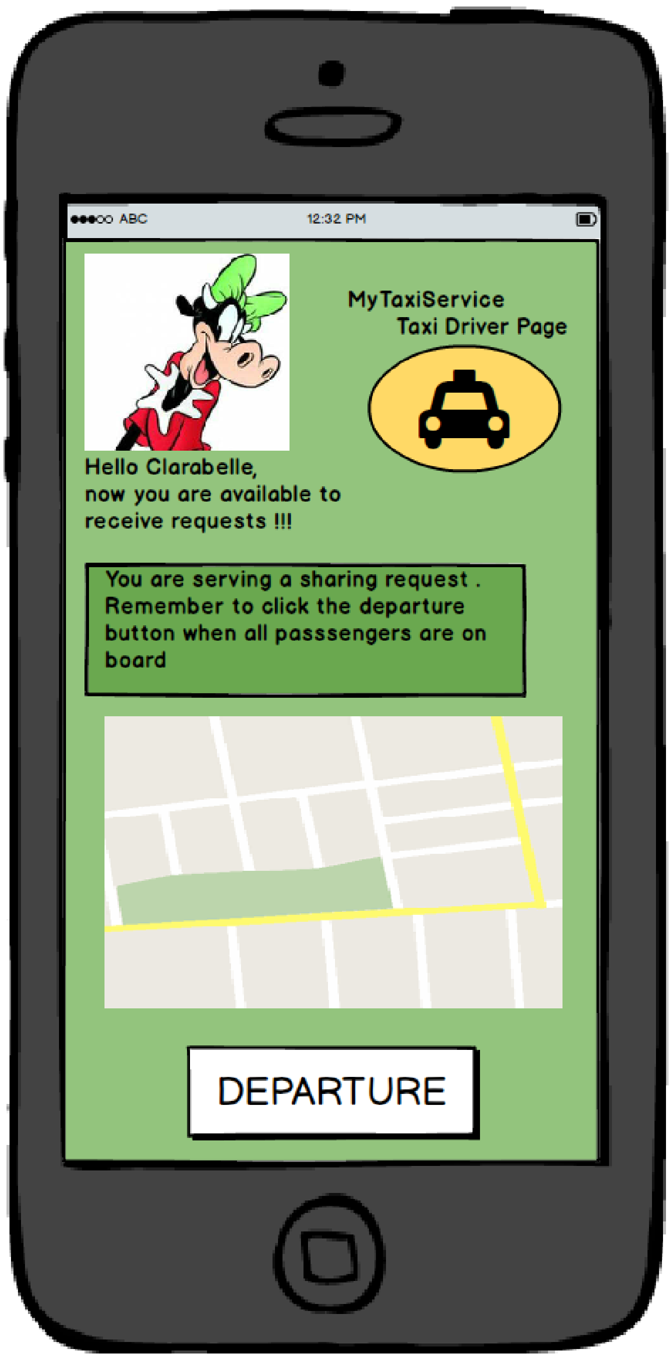


Hello Clarabelle,
now you are available to receive requests !!!

NOW YOU ARE AVAILABLE
WAIT FOR REQUEST!!!

There is a sharing request for you!! The guest is in Via Milano 9
Do you want to accept it ?

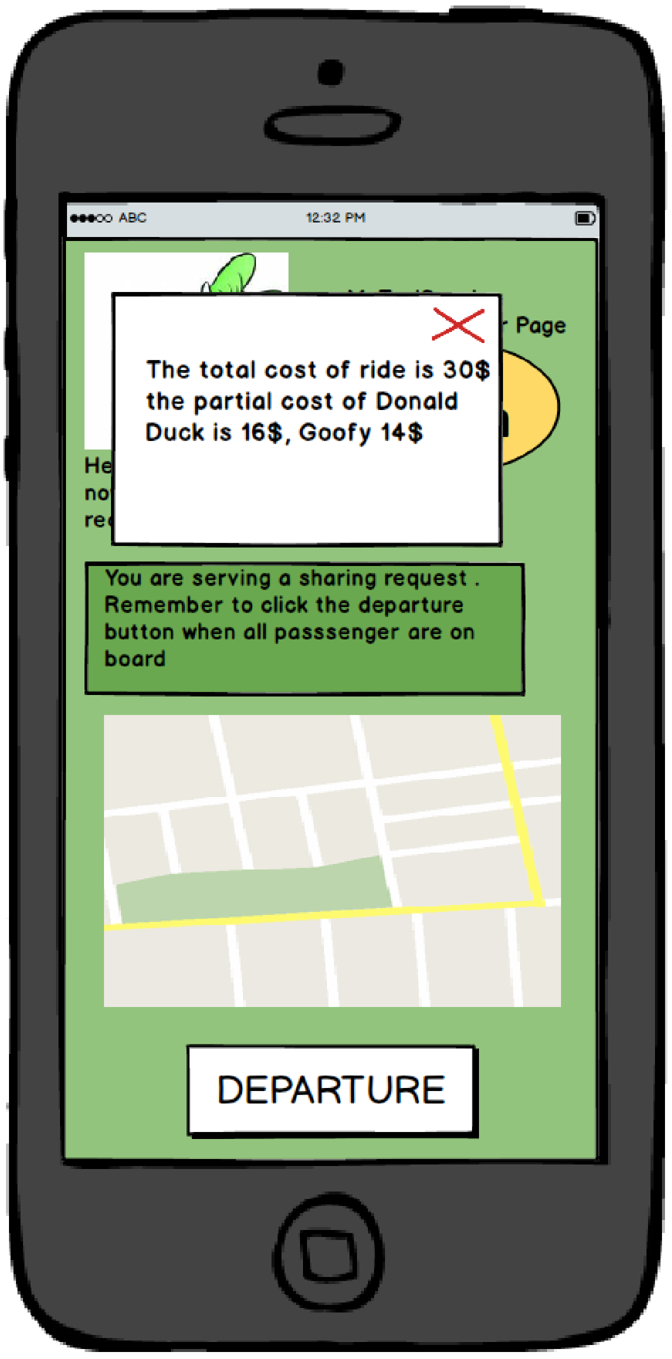
to receive request your gps must be enabled



Hello Clarabelle,
now you are available to receive requests !!!

You are serving a sharing request .
Remember to click the departure button when all passsengers are on board

- ➔ The MyServiceSystem Button is linked with the home Page of Taxi Driver
- ➔ During the sharing ride, before the taxi Driver has clicked the Departure Button , new passengers, with sharing option ,are notified with a message as the request message to the Taxi Driver
- ➔ After the Taxi Driver has clicked the Departure Button, the system notifies the total cost and fees of the ride.



The total cost of ride is 30$ the partial cost of Donald Duck is 16$, Goofy 14$

You are serving a sharing request . Remember to click the departure button when all passsenger are on board

3.3 CONSTRAINTS

- The System should provide a web application that Guests are able to access
- The System should provide a mobile application that Guests are able to access
- The System should provide a mobile application that urban TaxiDriver are able to access
- The system offers a list of APIs that lets every developer add the system functionalities to his software projects

4. SCENARIO:

- Like every Thursday evening, Minnie wants to go to Daisy Duck's home to see the last "Game Of Thrones" episode with her. It's time to go, when Minnie realizes that Mickey Mouse has taken their car and he hasn't come back yet (although it's very late!).
  She needs a lift to reach Daisy Duck's house, so she surfs the net to find any suggestion: she wants to find a taxi in a few minutes. She immediately encounters myTaxiService application (which really has high marks), she downloads it in a few minutes and signs up filling the form with her personal data (name, surname, mail address, phone number). Then she logs into the system and goes on, asking for a taxi to reach her near her house (to do that, she inserts her Mickey Mouse Clubhouse address). After few seconds, she receives a notification message that says: "your request has been confirmed. Taxi with ID Code 3225 will be at your service within 4 minutes." As promised, the taxi arrives on time, Minnie will spend a beautiful evening with Daisy Duck and will certainly post a very positive feedback on the AppStore!

- Mickey Mouse absolutely wants to visit the Expo this weekend, because it will end within few weeks and he certainly doesn't want to be the only one in the Universal Studios who hasn't gone yet! He decides to go to Rho by taxi and to book in advance in order to be sure that he will not have to go there on his own. Since Minnie does not stop repeating how fantastic the new application myTaxiService is, he accesses the myTaxiService webpage (unfortunately, he still does not have a smartphone) and logs into the system using his wife's credentials. At this point, he quickly books a taxi for next Saturday at

7:00 am, inserting his house address as departure position and Padiglione Italia as destination. Convinced by his wife's advice, he books another taxi for the return trip, which will occur the same day at 5:30 pm. The system notifies the booking confirmation with a notice in which it reminds him that a possible deletion of the reservation can be done only until 10 minutes before the appointment. The service works properly, Minnie and Mickey Mouse spend a wonderful day at Expo, but it's 5 o'clock pm when they still are queueing at the Japanese pavilion: they will not succeed in taking the booked taxi! Luckily, the appointment is within 30 minutes, so they are allowed to delete the reservation: they log into the system using Minnie's smartphone, they delete the booked ride and try to reserve another taxi that meets them at 6:30 pm.
Immediately they receive myTaxiService message that says: "the reservation has been rejected; you can book a taxi with at least a two hours advance notice". Minnie and Mickey Mouse decide to reserve a ride for evening at 10 pm, so they will have time to visit the pavilion and have dinner in one of the awesome restaurants located in Piazza Italia.

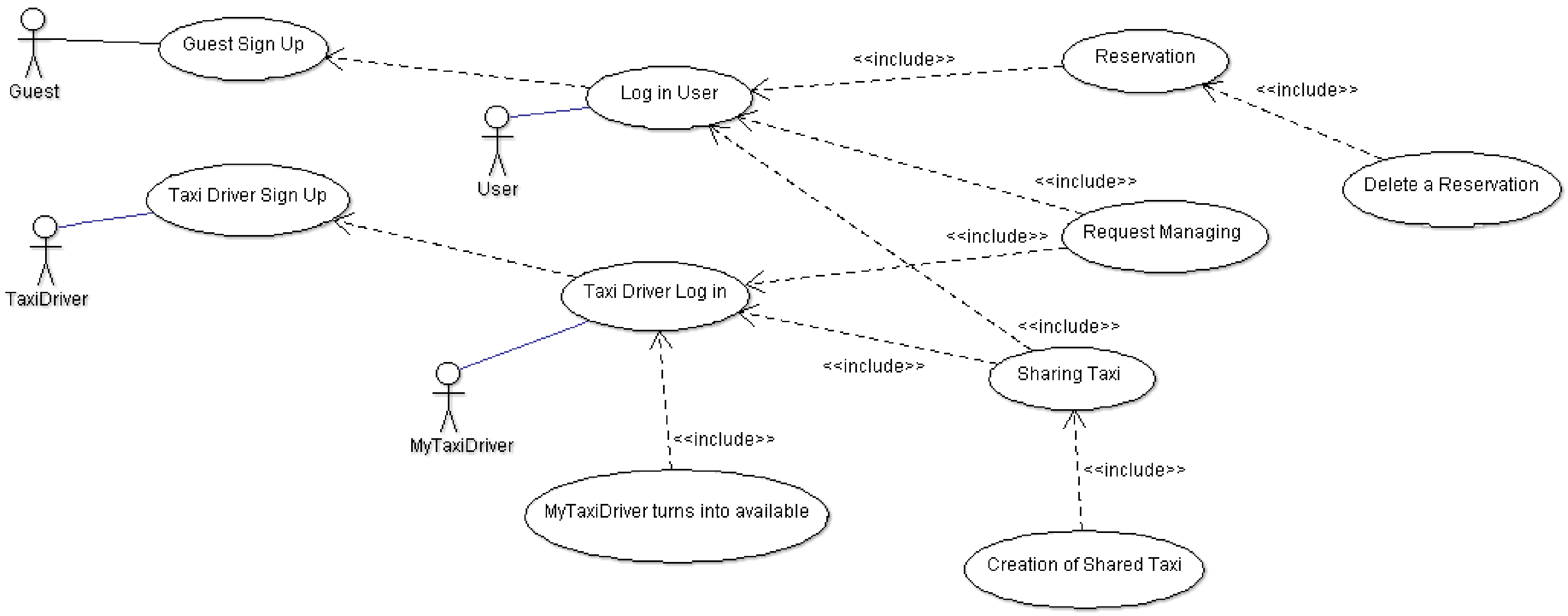
- After a hard working day, Donald Duck runs to the Milano Lambrate Station but he unfortunately misses the last train to Milano Centrale. So he decides to use myTaxiService, the app to which he has just signed up. He logs into the system and asks for a taxi, enabling the sharing option and inserting the destination address; the system confirms him the coming of the taxi 5937 with an arrival time of 15 minutes. After 5 minutes, Goofy and Pluto close their nearby bakery and are going to request a taxi to go back home in Palazzo Reale. They choose the sharing taxi option, too: a so long and boring trip is a good opportunity to make new friends! At that moment, the taxi driver Peg Leg Pete, who is driving the taxi 5937 and is going to reach Donald Duck, receives a message from the application with a new and updated route: he has to serve other two passengers that, luckily, are two minutes away from the first passenger. Meanwhile, Goofy and Pluto receive the confirmation with taxi ID Code 5937 and an estimated time of 12 minutes. When all the passengers are reached, the taxi driver informs myTaxyService that he's ready to leave and receives a notification with the bill and all the partial fees. Goofy, Pluto and Donald Duck receive, in turn, information about their part of the total fee. Donald Duck cannot really complain about the service: the taxi sharing option give him the chance to find two new friends and save a lot of money!

- There’s a new taxi driver working in the Milan taxi service: her name is Clarabelle Cow! She is getting ready for her first working day, all the documents are in place (including her taxi ID Code) so she decides to go on with the registration using the dedicated mobile app. To do that, she fills the form with all her personal data and her taxi ID Code and she enables the gps localisation. Awesome! Now Clarabelle Cow cannot wait to serve her first clients, so she logs into the system and informs it about her availability. Within a few minutes she receives her first request from myTaxiService: the system submits to her a passenger waiting at Via Monte Napoleone, 56. Clarabelle Cow doesn’t think twice, she confirms immediately and she’s going to reach the gathering point as soon as possible. She arrives at the meeting place, finds the client without any problem and takes him to the right destination; now she has just to declare her availability again, relax and wait for a new request: the taxi driver’s life is not so bad if you can trust myTaxiDriver support!

- Scrooge McDuck has struck by an incredible intuition while he is sitting in a taxi, headed for the Linate airport: why don’t we add the urban taxi service functionalities to my famous “DuckMaps” app? Since he is enthusiastic about myTaxyService treatment, just received (indeed he finds a taxi quickly using this app), he decides to call his team of developers to see if it is possible to find a solution to the problem. His task force manager tells him that there will be no problems: myTaxiService makes the required API available to complete the mapping service with its functionalities. A new profitable partnership between these two mobile applications giants is about to be born!

5. UML MODELS

5.1 USE CASE DIAGRAM



5.2 USE CASE DESCRIPTION AND SEQUENCE DIAGRAMS:

After having supposed some possible scenarios about the desired system behavior, we can generalize them in use cases. In the following paragraph, we want to show, for each use case, a description and the corresponding sequence diagram.
We said that interactions are the main subject of our modelling, and the main feature of the problem to be analyzed. Sequence diagrams are essentials for this reason: every function has to be described during its execution, focusing on the interactions between the affected actors and the system.

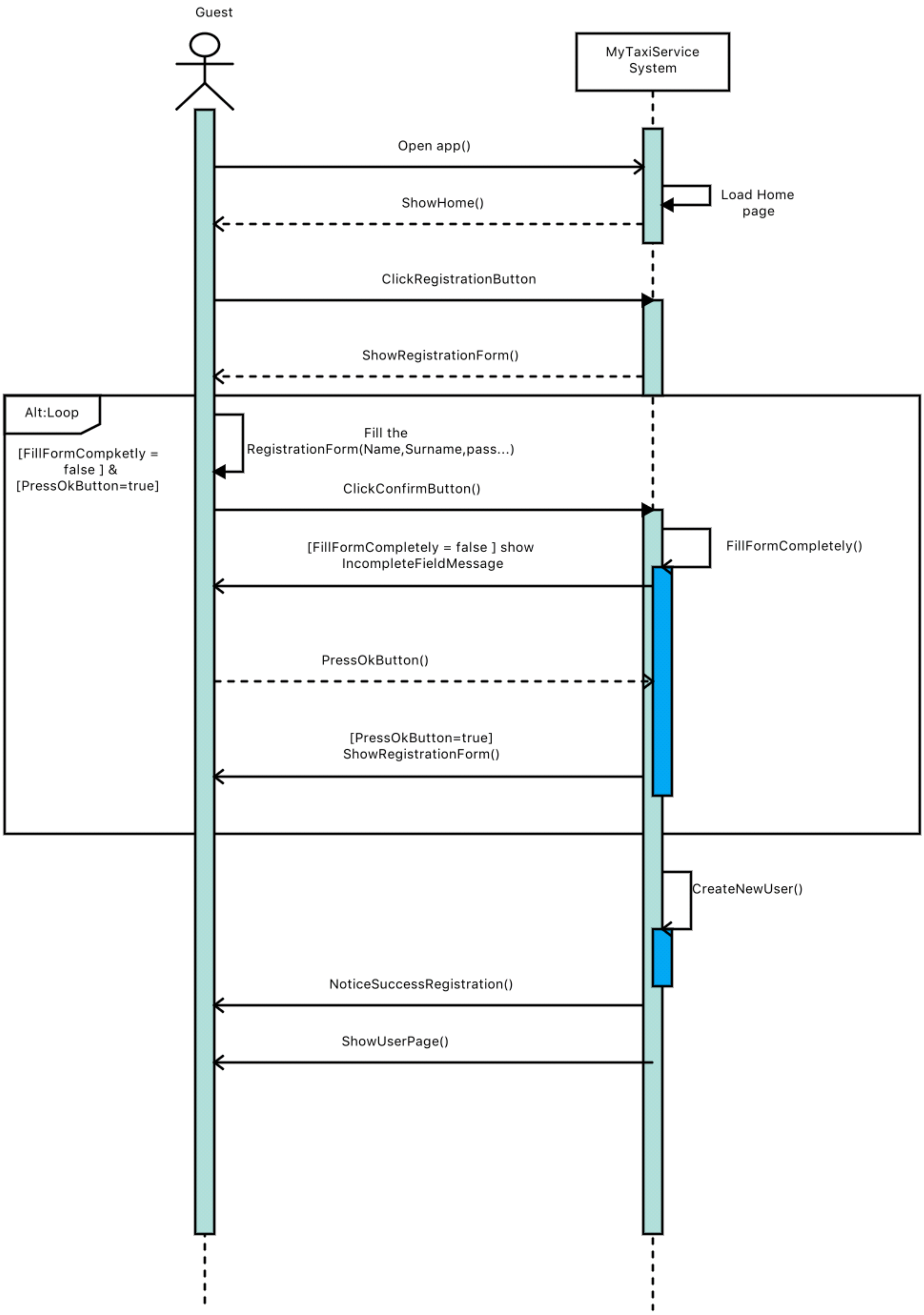
- Guest Sign Up":
- Sign up Taxi Driver
- Log in User
- Taxi Driver Log in
- Request Managing
- Sharing Taxi
- Shared Taxi Creation
- Taxi Available
- Reservation
- Reservation with Sharing option
- Delete a Reservation

We evaluate the use case “Guest Sign Up”:

Name	Guest Sign Up
Actors	Guest
Entry Condition	The guest hasn’t already signed up and he has downloaded the app or he is viewing MyTaxiService Web page. The guest has to fill the form with valid information.

Flow of Events	• Guest clicks on the Sign Up button
	• The system loads a form in which it asks for:
	    o Name
	    o Surname
	    o Mail
	    o Telephone Number
	    o Password
	• The guest completes the form and clicks on “Confirm”
	• The system stores the data, notices the successful registration and loads the Log In page

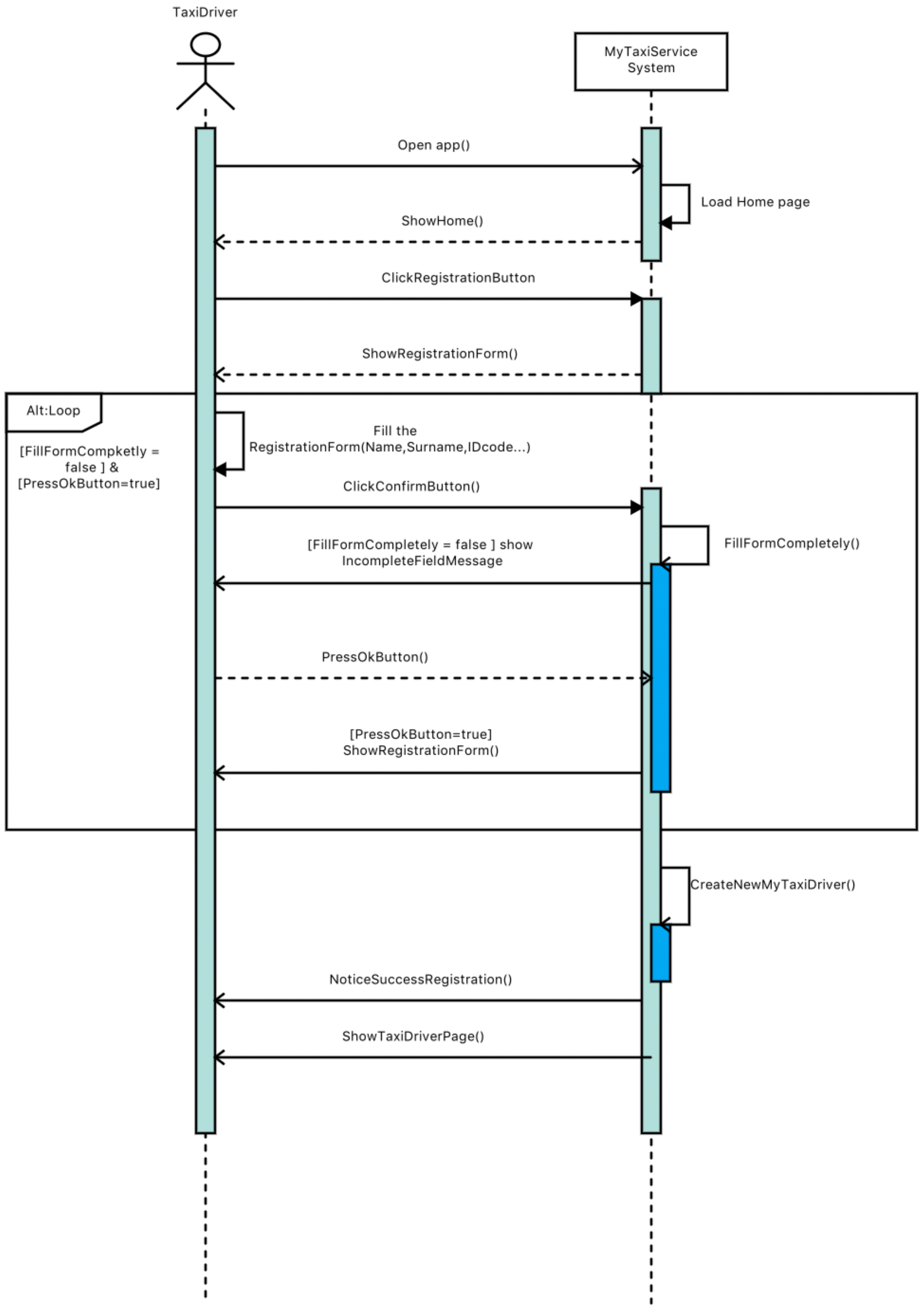
Exit Conditions	The guest data are successfully registered in the system database, and now the guest is able to log into the system
Exceptions	The guest doesn’t fill the form completely: the system sends again the sign up form with a notice of incomplete fields


Guest
MyTaxiService System
Open app()
Load Home page
ShowHome()
ClickRegistrationButton
ShowRegistrationForm()
Alt:Loop
[FillFormCompketly = false] & [PressOkButton=true]
Fill the RegistrationForm(Name,Surname,pass...)
ClickConfirmButton()
[FillFormCompletely = false] show IncompleteFieldMessage
FillFormCompletely()
PressOkButton()
[PressOkButton=true] ShowRegistrationForm()
CreateNewUser()
NoticeSuccessRegistration()
ShowUserPage()

Powered by DrawExpress

We evaluate the use case “Sign up Taxi Driver”:

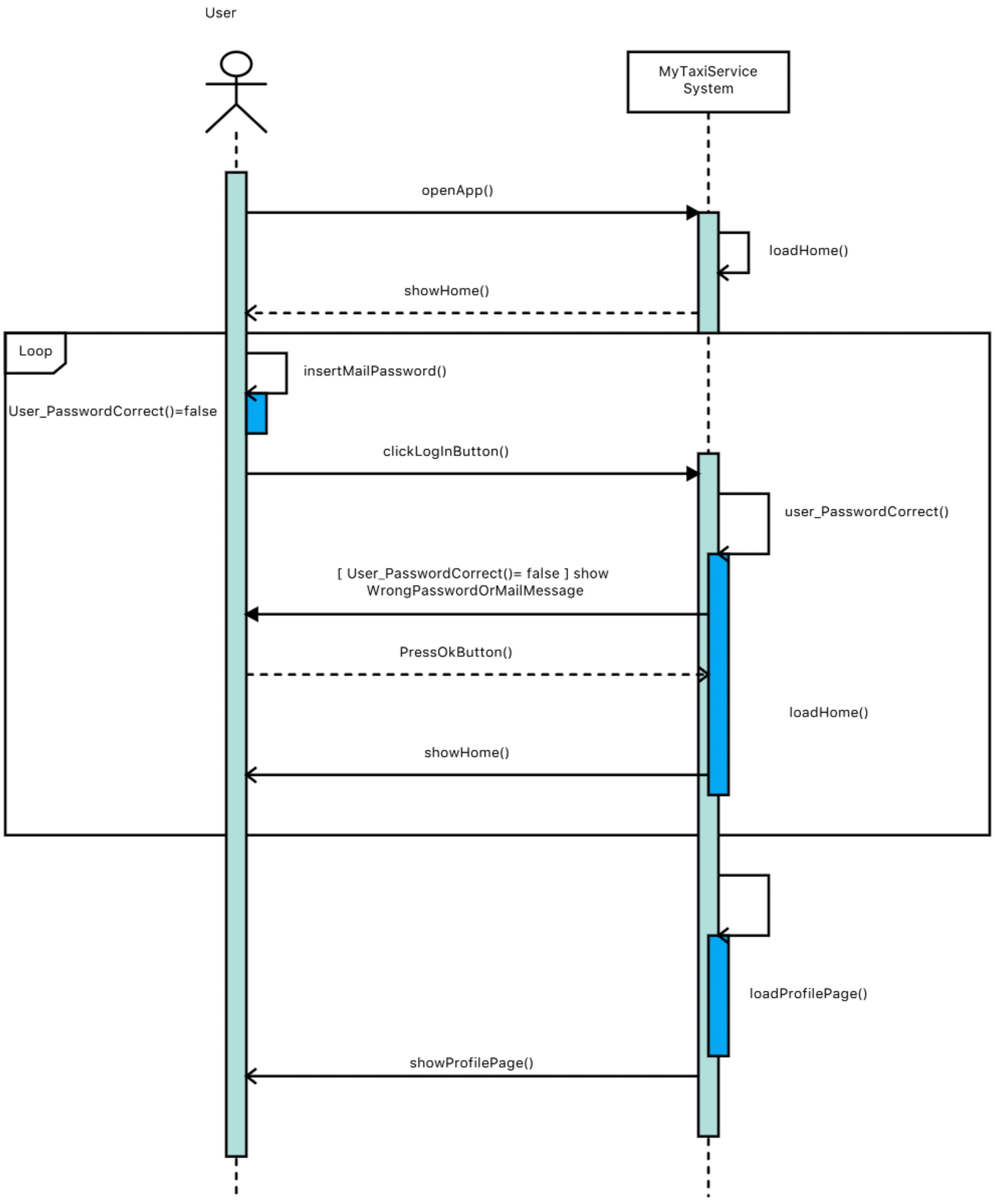
Name	Taxi Driver Sign Up
Actors	Taxi Driver
Entry Condition	The Taxi driver must have a valid ID code, he hasn’t signed up yet and he has downloaded the app. The Taxi Driver has to fill the form with valid information.
Flow of Events	• Taxi Driver clicks on the Sign Up button • The system loads a form in which it asks for: o Name o Surname o Mail o Telephone Number o Password o ID Code • The Taxi Driver completes the form and click on “Confirm” • The system stores the data, it notices successful registration and it loads the Log In page
Exit Conditions	The Taxi Driver data are successfully registered in the system database, and now the Taxi Driver is able to log in in the system
Exceptions	The Taxi Driver doesn’t fill the form completely: the system sends again the sign up form with a notice of incomplete fields


TaxiDriver
MyTaxiService System
Open app()
Load Home page
ShowHome()
ClickRegistrationButton
ShowRegistrationForm()
Alt:Loop
[FillFormCompketly = false] & [PressOkButton=true]
Fill the RegistrationForm(Name,Surname,IDcode...)
ClickConfirmButton()
FillFormCompletely()
[FillFormCompletely = false] show IncompleteFieldMessage
PressOkButton()
[PressOkButton=true] ShowRegistrationForm()
CreateNewMyTaxiDriver()
NoticeSuccessRegistration()
ShowTaxiDriverPage()

Powered by DrawExpress

We evaluate the use case “Log in User”:

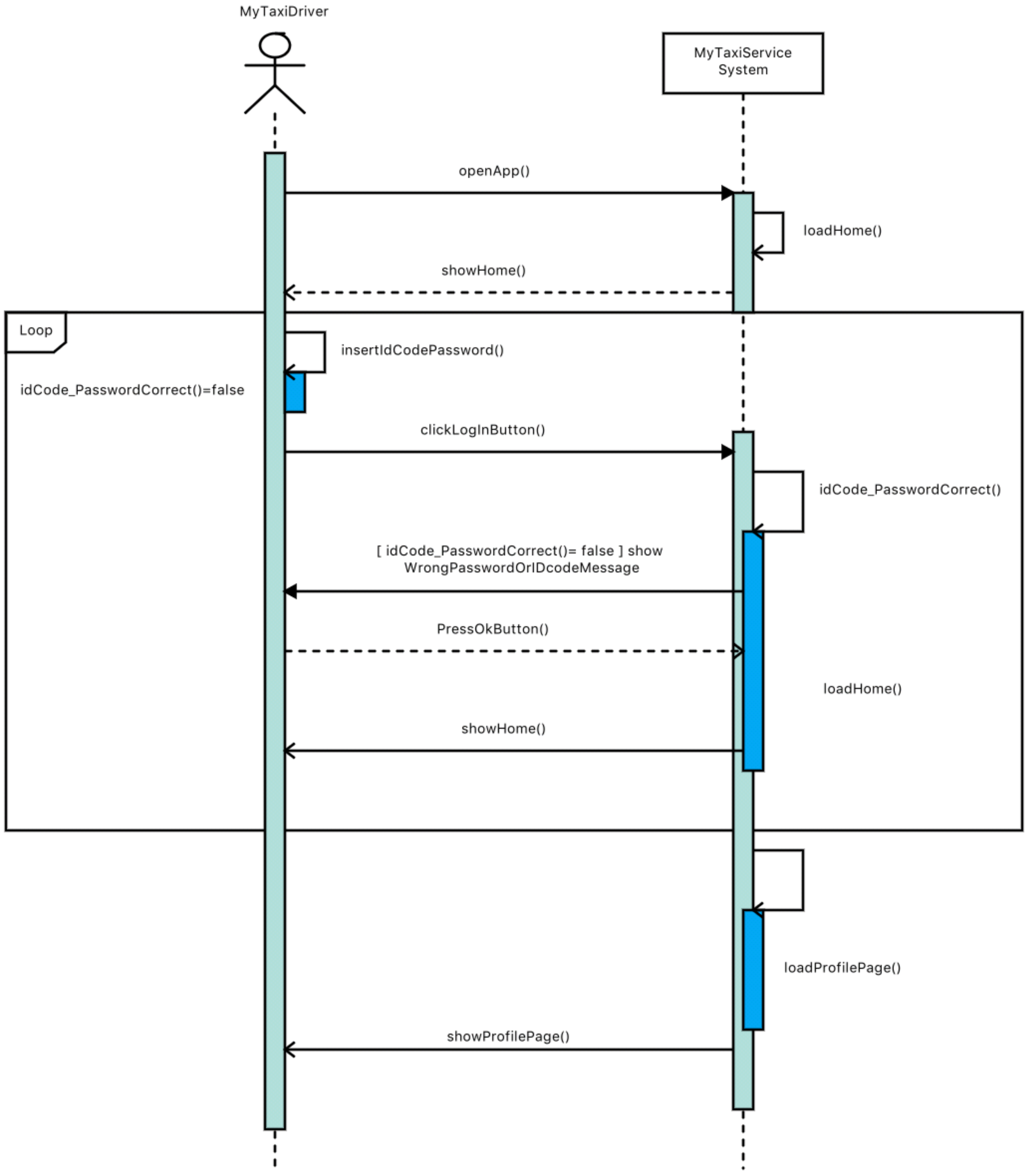
Name	User Log In
Actors	User
Entry Condition	The user has already registered to the system
Flow of Events	• The User accesses the home page or application • System views the log in page • User enters a mail address and a password and clicks on the log In button • The System checks the validity of mail address and password, finds them and loads the profile page
Exit Conditions	Now the user is able to use system functionalities
Exceptions	The system doesn’t find the couple mail address - password, inserted by the user, in the database. In this case the system notifies the incorrect insertion of the password or the mail address and asks for entering the data again



Powered by DrawExpress

We relate to the use case “Taxi Driver Log in”:

Name	Taxi Driver Log In
Actors	MyTaxiDriver
Entry Condition	The MyTaxiDriver has already registered to the system and he has already enabled the gps position
Flow of Events	• The MyTaxiDriver accesses the home page or application • System views the log in page • The MyTaxiDriver enters his taxi ID code and a password and clicks on the log In button • The System checks the validity of the ID code and the password, finds them and loads the profile page
Exit Conditions	Now the MyTaxiDriver is able to receive requests from the system
Exceptions	The system doesn’t find the couple ID code – password, inserted by the MyTaxiDriver, in the database. In this case the system notifies the incorrect insertion of the password or the ID code and asks for entering an ID code and a password again



Powered by DrawExpress

We evaluate the use case “Request Managing”:

Name	Request Managing
Actors	User, MyTaxiDriver
Entry Condition	Both user and MyTaxiDriver are logged in
Flow of Events	<ul><li>The user clicks on the “Search Taxi” Button in his profile page</li><li>The system views the Search Taxi Page, composed by:<ul><li>A field in which the user enters the appointment address</li><li>A field in which the user chooses the number of passengers (with a maximum of 4)</li><li>A check button that enables the gps position of User</li><li>A check button that enables the sharing option</li></ul></li><li>The user enters the information without enabling the sharing option and confirms the request</li><li>The System searches the zone corresponding to the appointment address, sends the request to the first MyTaxiDriver stored in the available taxi queue of the zone and waits 1 minute for the MyTaxiDriver’s answer</li><li>The MyTaxiDriver receives the request and decides to confirm or reject it.</li><li>The System receives the answer from MyTaxiDriver</li><li>If he has confirmed, the system pops him from the top of the queue and sends the confirmation notice to the user with the ID Code corresponding to the MyTaxiDriver who</li></ul>

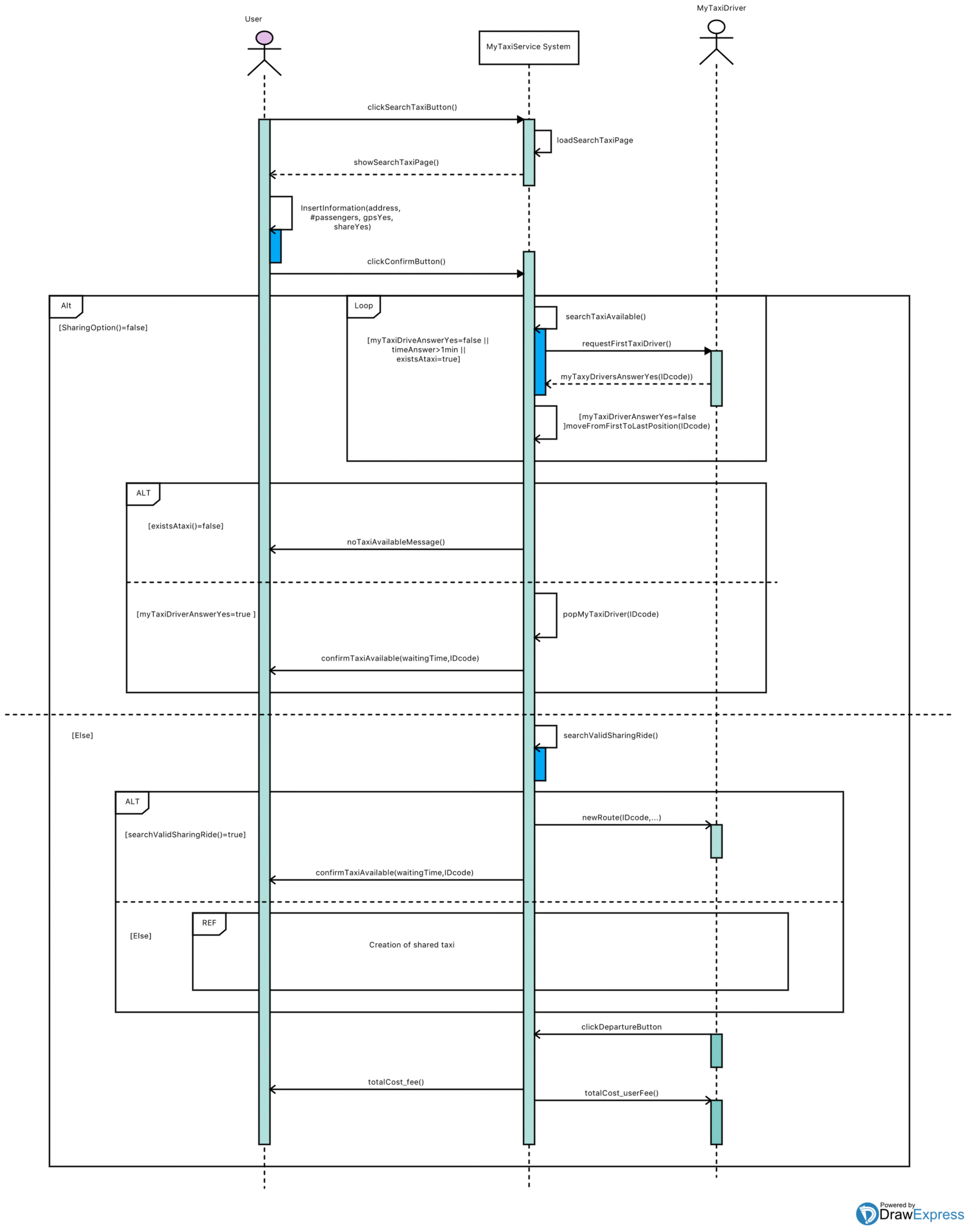
	takes care of him and the waiting time calculated by the WaitAlgorithm • Otherwise the system sends the request to the next available MyTaxiDriver and moves the first in the last position of the queue. It repeats this operation until someone confirms.
Exit Conditions	The user receives a confirmation message that includes the Taxi ID Code and estimated time
Exceptions	• There aren't any available taxi in that zone so the system tries to find an available taxi in the nearest zone until it finds one • The only available taxi in all zones has a waiting time longer than 20 minutes then the system sends a "no found taxi" notice • There aren't available taxi in the city then the system sends a "no found taxi" notice

We evaluate the use case "Sharing Taxi ":

Name	Sharing Taxi
Actors	MyTaxiDriver, User
Entry Condition	MyTaxiDriver and User are logged in
Flow of Events	

	• User clicks on the “Search Taxi” Button in his profile page • The system views the Search Taxi Page, composed by: o A field in which the user can enter the gathering address o A field in which the user can choose the number of passengers (with a maximum of 4) o A check button that enables the gps position o A check button that enables the sharing option • User enters the required information, enables the sharing option, inserts the destination address and confirms the request • The System searches if there is a valid (starting from the same zone corresponding to the appointment address, with the destinations in the same direction and not already departed) sharing ride: • If there’s a taxi, the system controls that the number of passengers, including the new ones, has to be less than 4, then it notifies the MyTaxiDriver the new route and shows the confirmation to the User with the taxi ID code of the MyTaxiDriver who takes care of him and the waiting time calculated by the SharedWaitingAlgorithm • Otherwise the system creates a new shared taxi (see “Creation of shared taxi” use case) • Once the MyTaxiDriver has reached all the passengers, he has to click the Departure button • The system sends to all the involved users the total cost and the corresponding fee, and to the MyTaxiDriver the total cost and all the users’ fees
Exit Conditions	In this case there’s no exit condition
Exceptions	In this case there’s no exception

This diagram describes both requests with and without sharing option:



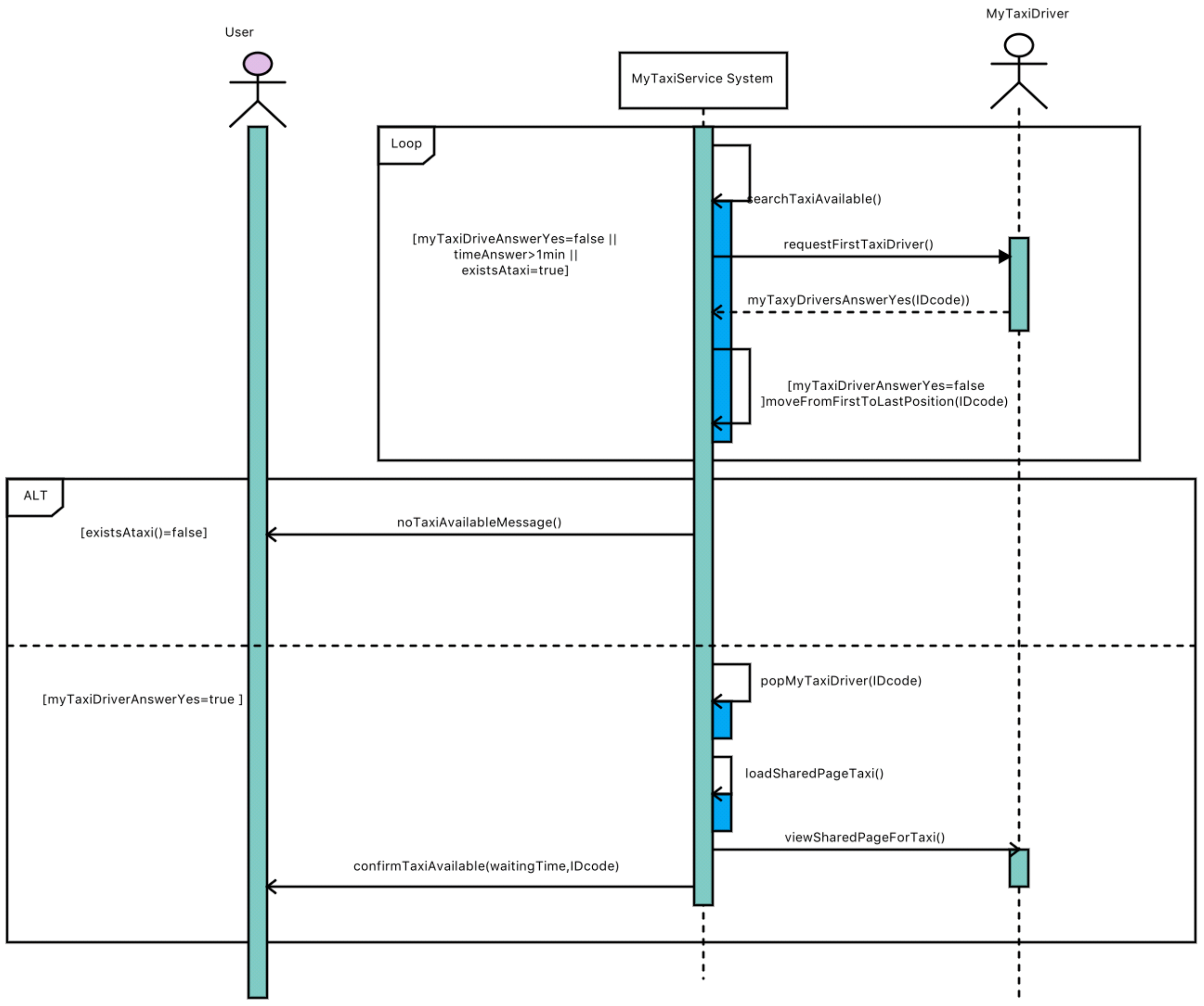
Powered by DrawExpress

We evaluate the use case “Shared Taxi Creation”:

Name	Creation of Shared Taxi
Actors	MyTaxiDriver
Entry Condition	MyTaxiDriver is logged in
Flow of Events	<ul><li>The system sends the Sharing request to the first MyTaxiDriver stored in the available taxi queue of the zone and waits 1 minute for the MyTaxiDriver’s answer.</li><li>The MyTaxiDriver receives the request and decides to confirm or reject it.</li><li>The System receives the answer from the MyTaxiDriver</li><li>If he has confirmed, the system</li><li>pops him from the top of the queue,</li><li>sends the confirmation notice to the user with the taxi ID code of the MyTaxiDriver who takes care of him and the waiting time calculated by the WaitAlgorithm</li><li>views the Shared route Page with the route and the Departure button</li><li>Otherwise the system sends the request to the next available MyTaxiDriver and moves the first in the last position of the queue. It repeats this operation until someone confirms.</li></ul>
Exit Conditions	A new shared taxi has been created and it’s available to serve sharing requests
Exceptions	

	• There aren’t available taxi in that zone so the system tries to find an available taxi in the near zones until it succeeds in finding one • The only available taxi in all zones has a waiting time longer than 20 minutes then the system sends a “no found taxi” notice • There aren’t available taxi in the city then the system sends a “no found taxi” notice

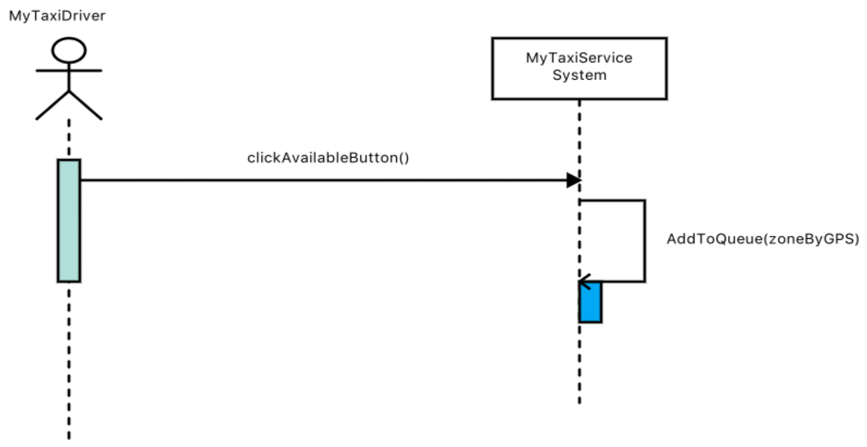
This is the sequence diagram Creation of shared Taxi: we refer to it in the REF of the “search taxi“ sequence diagram.


User
MyTaxiService System
MyTaxiDriver
Loop
[myTaxiDriveAnswerYes=false || timeAnswer>1min || existsAtaxi=true]
searchTaxiAvailable()
requestFirstTaxiDriver()
myTaxyDriversAnswerYes(IDcode))
[myTaxiDriverAnswerYes=false]moveFromFirstToLastPosition(IDcode)
ALT
[existsAtaxi()=false]
noTaxiAvailableMessage()
[myTaxiDriverAnswerYes=true]
popMyTaxiDriver(IDcode)
loadSharedPageTaxi()
viewSharedPageForTaxi()
confirmTaxiAvailable(waitingTime,IDcode)

Powered by DrawExpress

We evaluate the use case “MyTaxiDriver turns into available”:

Name	MyTaxiDriver turns into available
Actors	MyTaxiDriver
Entry Condition	MyTaxiDriver is logged in, he is not serving other Users and he has already enable the GPS information.
Flow of Events	• MyTaxiDriver clicks on the “Available” button • The system determines in which zone the taxi is, based on the GPS information, then stores MyTaxiDriver’s taxi ID code in the corresponding queue and inform the MyTaxiDriver that now he is available
Exit Conditions	MyTaxiDriver is now available and ready to receive requests
Exceptions	In this case, there are no exceptions



We evaluate the use case "Reservation":

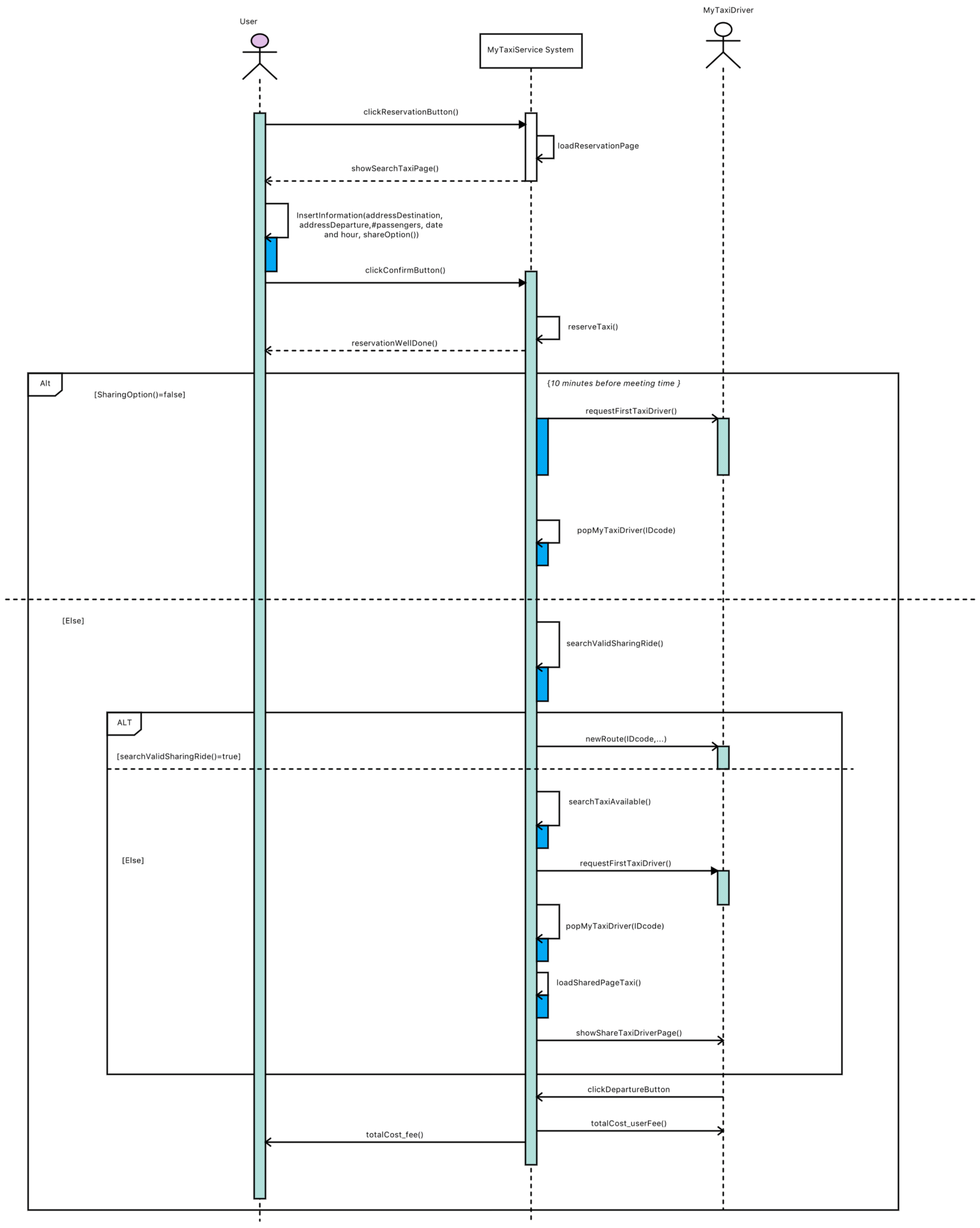
Name	Reservation
Actors	User , MyTaxiDriver
Entry Condition	User is logged in and he hasn't already done a reservation with the same time and date
Flow of Events	• The user clicks on the "Reservation" button • The system views the Reservation Page, composed by: o A calendar that lets the User choose the date o A clock that lets the User choose the hour o A field in which the User can enter the gathering address o A message in which the system informs the user that the registration can be done only two hours before the appointment time • The user enters the required information and clicks on the "Confirm" button • The system stores the reservation in a Database and informs the user that the operation was correctly done • The system reserves a taxi 10 minutes before the meeting time, choosing the first taxi in the queue of the corresponding zone • MyTaxiDriver receives the notification of a Reservation request that he cannot reject.
Exit Conditions	The reservation is properly done

Exceptions	• The user inserts an invalid date so the system notifies an invalid insertion error • The user tries to reserve a taxi less than two hours before the appointment so the system notifies the User that the reservation can be done at least two hours before the appointment and that he can try to repeat the operation with another hour • The queue is empty, in this case the system searches an available taxi in the contiguous zones, until it succeeds in finding one

We evaluate the use case “Reservation with sharing option”:

Name	Reservation with sharing option
Actors	User
Entry Condition	User is logged in
Flow of Events	• User clicks on the “Reservation” button • The system views the Reservation Page, composed by: o A calendar that lets the User choose a date o A clock that lets the User choose the time o A field that lets the User enter the appointment address o A message in which the system informs the user that the registration can be done only two hours before the appointment

	• The user enters the information and clicks on the “Confirm” button • The system stores the reservation information in a Database and informs the user that the operation was correctly done • 10 minutes before the appointment time, the System searches if there is a valid (starting from the same zone corresponding to the appointment address, with the destinations in the same direction and not already departed) sharing ride: • If there’s a taxi, the system controls that the number of passengers, including the new ones, has to be less than 4, notifies the MyTaxiDriver the new route and shows the confirmation to the User with the taxi ID code of the MyTaxiDriver who takes care of him and the waiting time calculated by the SharedWaitingAlgorithm • Otherwise the system creates a new shared taxi (see “Creation of a shared taxi” use case) • The system sends to the User a confirmation notice with the taxi ID code
Exit Conditions	The reservation is correctly done
Exceptions	• The user inserts an invalid date so the system notifies an invalid insertion error • The user tries to reserve a taxi less than two hours before the appointment so the system notifies to the user that the reservation can be done at least two hours before the appointment and that he can try to repeat the operation with another hour

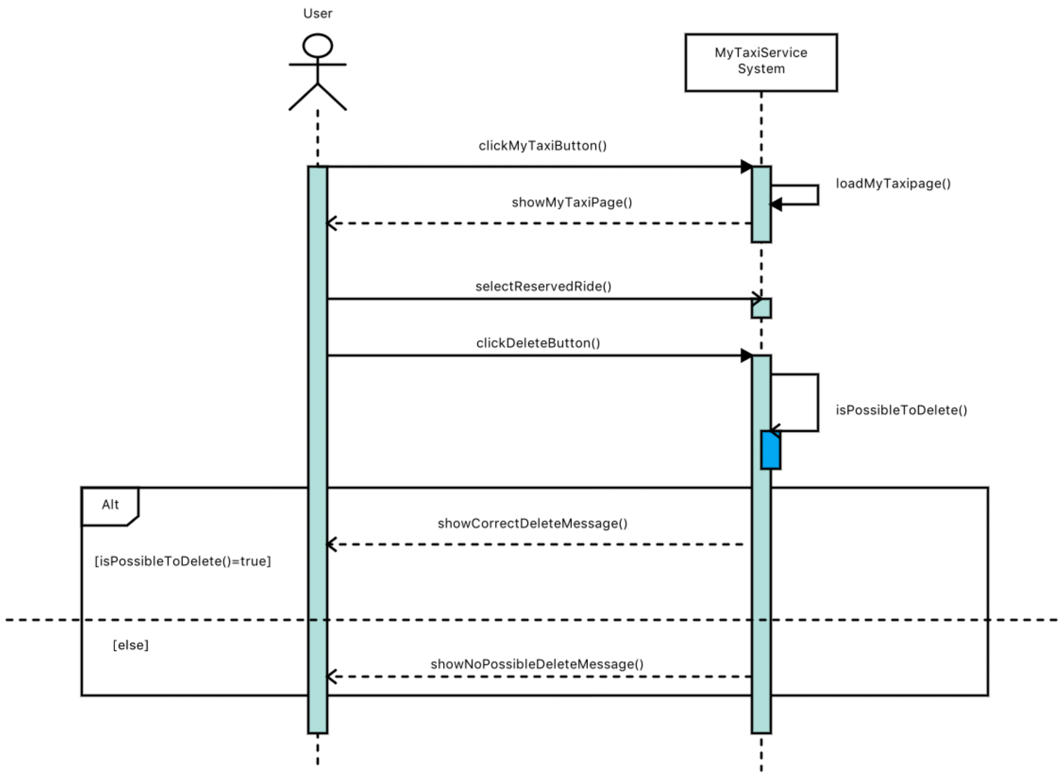


Powered by DrawExpress

The two previous use cases both describe booking operations, so we have combined these to a unique diagram, in order to make the UML model more generalized.

We evaluate the use case “Delete a Reservation”:

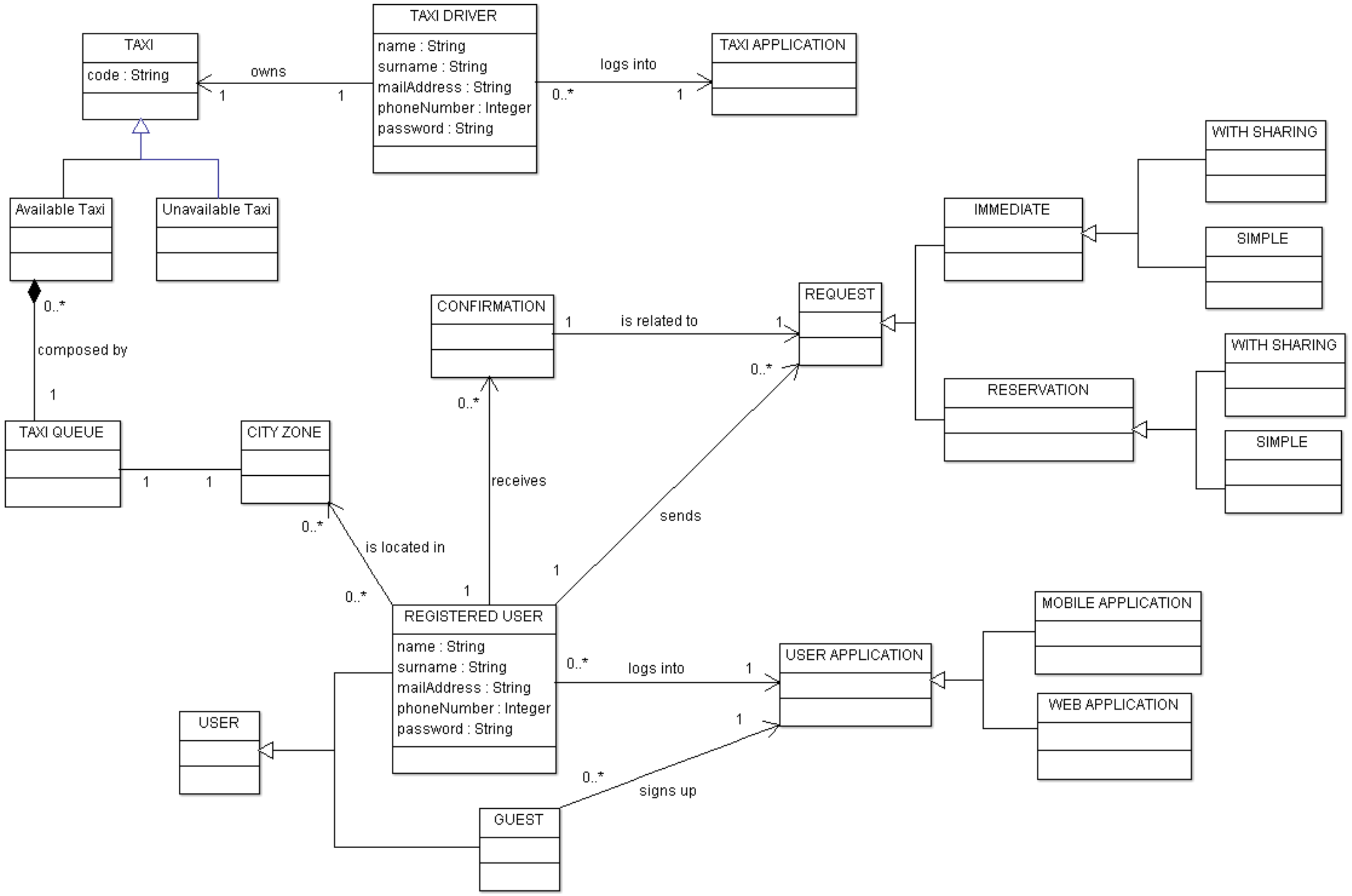
Name	Delete a Reservation
Actors	User
Entry Condition	User is logged in and he has already done a reservation
Flow of Events	• The user clicks on the “MyTaxi” button • The system views the MyTaxi page in which there is the list of his taxi reservations with all the details about the time and the date and the list of taxi requests • The user selects a reservation and clicks on the delete button • The system deletes the reservation from the database and notifies that the operation is correctly done
Exit Conditions	The reservation is correctly deleted
Exceptions	• The user tries to delete a reservation less than 10 minutes before the appointment time so the system notifies him that the taxi is reserved by now and it is impossible to delete the reservation • The user who wants to delete a request cannot do it because there’s no delete button in the request view



Powered by DrawExpress

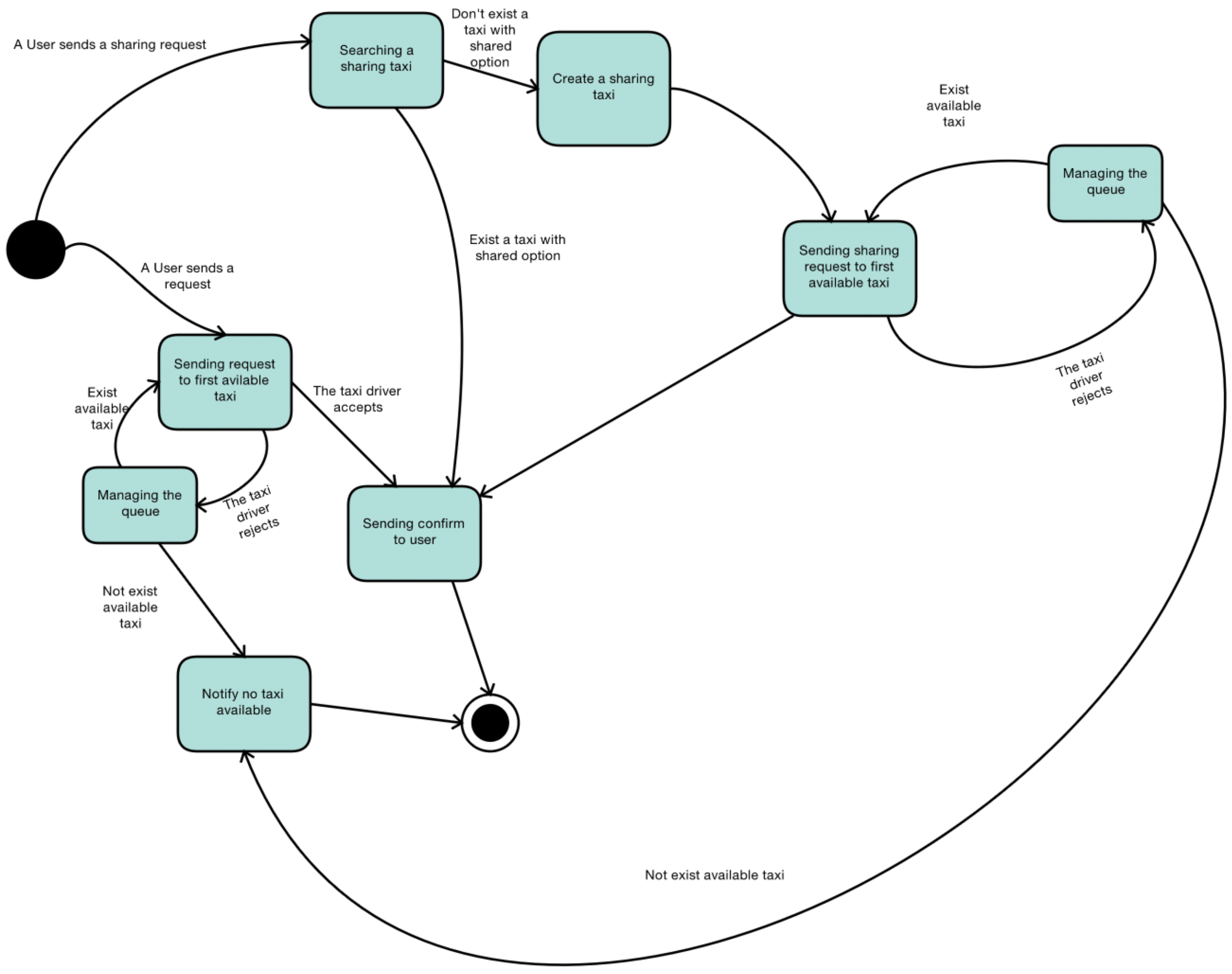
5.3 CLASS DIAGRAM:

Now we describes the class diagram of our system, it helps to understand the use cases and to write the alloy model

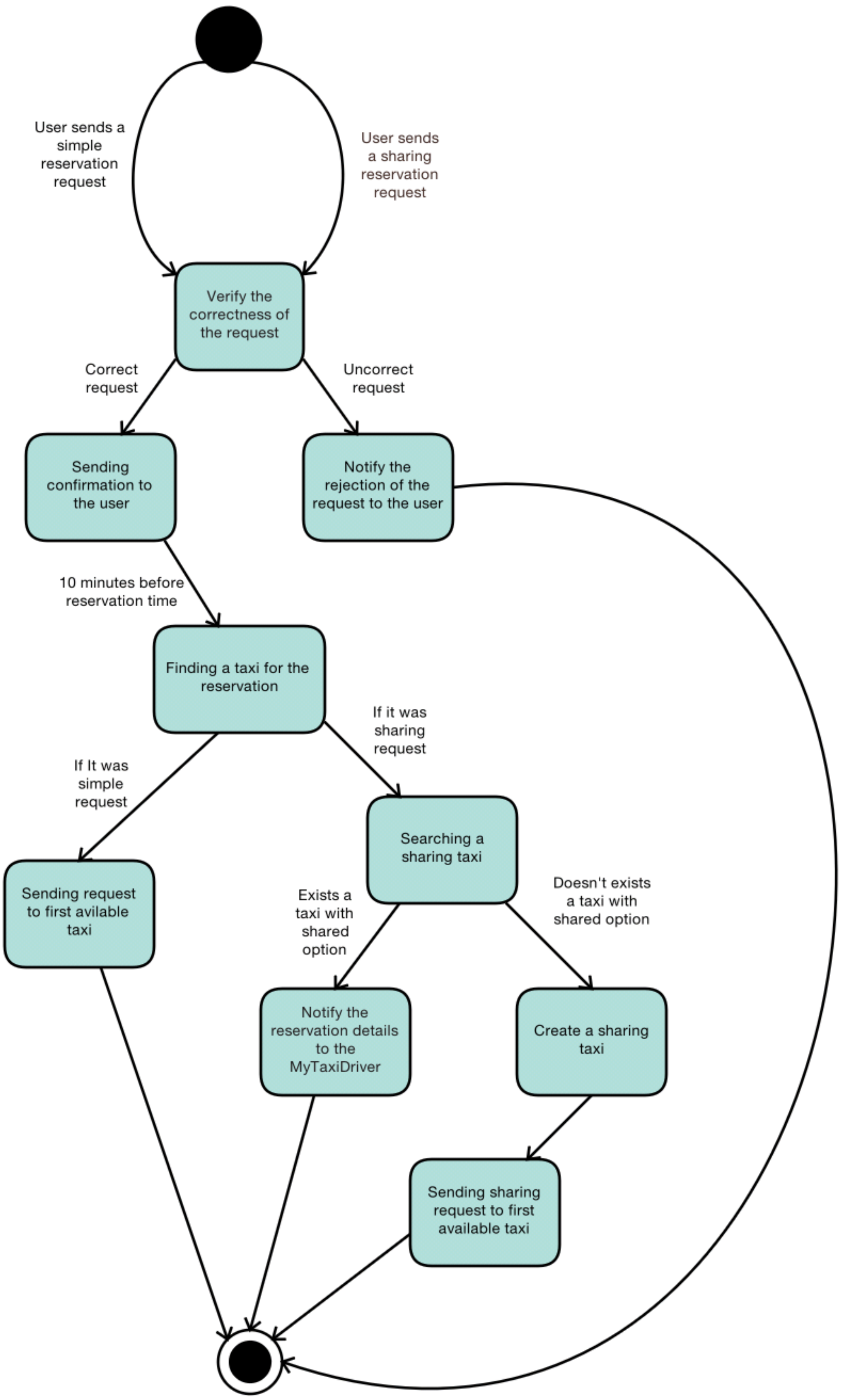


5.4 STATE CHART DIAGRAMS:

5.4.1 Request Event:



5.4.2 Reservation Event:

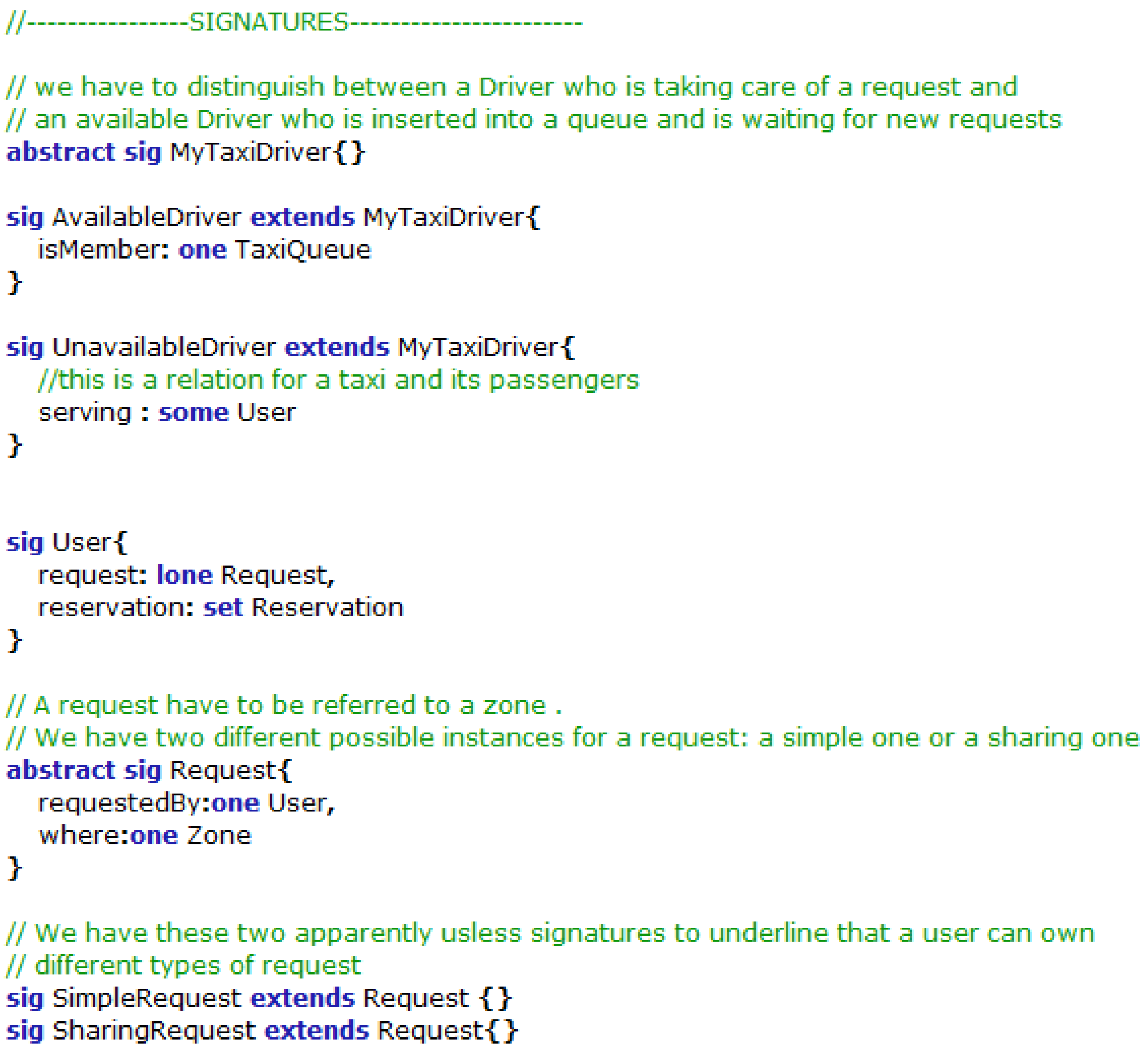

User sends a simple reservation request
User sends a sharing reservation request
Verify the correctness of the request
Correct request
Uncorrect request
Sending confirmation to the user
Notify the rejection of the request to the user
10 minutes before reservation time
Finding a taxi for the reservation
If it was sharing request
If It was simple request
Searching a sharing taxi
Sending request to first avilable taxi
Exists a taxi with shared option
Doesn't exists a taxi with shared option
Notify the reservation details to the MyTaxiDriver
Create a sharing taxi
Sending sharing request to first available taxi

6 ALLOY

We have used the Alloy Analyzer to consider the consistency of our possible system configurations. Using Alloy modelling we have also tried to make more explicit for the reader the properties of these configurations and so we have included in this document some visualizations of the Alloy generated worlds.
We have started considering the Class Diagram (see chapter 5.3) to generate the Alloy code, which we report in the following paragraphs together with the execution results and the generated worlds.

6.1 CODE

We report signatures and facts of our Alloy model:



```
//----------------SIGNATURES-----------------------

// we have to distinguish between a Driver who is taking care of a request and
// an available Driver who is inserted into a queue and is waiting for new requests
abstract sig MyTaxiDriver{}

sig AvailableDriver extends MyTaxiDriver{
   isMember: one TaxiQueue
}

sig UnavailableDriver extends MyTaxiDriver{
   //this is a relation for a taxi and its passengers
   serving : some User
}


sig User{
   request: lone Request,
   reservation: set Reservation
}

// A request have to be referred to a zone .
// We have two different possible instances for a request: a simple one or a sharing one
abstract sig Request{
   requestedBy:one User,
   where:one Zone
}

// We have these two apparently usless signatures to underline that a user can own
// different types of request
sig SimpleRequest extends Request {}
sig SharingRequest extends Request{}
sig Reservation{
    reservedBy: one User,
    date: one Date,
    where: one Zone
}

// We define this signature to represent date and time of a reservation
sig Date{}

// Each zone must have only one queue, and this may contains some available taxi
// We define a contiguity relation between zones
sig TaxiQueue{
    zone : one Zone,
    waiting : set AvailableDriver
}


sig Zone{
    queue : one TaxiQueue,
    contiguous : some Zone
}




//-------------------FACTS----------------------

// Constraints on drivers:
fact unavailableDriverFeatures{
    // the impossibility of having a user served by two distinct taxi,
    all u : User | no disj ud1, ud2 : UnavailableDriver | (u in ud1.serving) && (u in ud2.serving)
    // we impose the maximum of four passengers for each taxi
    all u: UnavailableDriver | #u.serving<5
}

    // if a user has requested a taxi, the taxi has received a request from the user
    all u :User | all r: Request  | (r in u.request) <=> (u in r.requestedBy)
    // if a taxi is unvailable cannot receive other requests
    no u:User, t:UnavailableDriver,r:Request | (u in t.serving) && (u in r.requestedBy)
}

//constraints on Reservation
fact reservationFeatures{
    // there's a bijective relation beetwen the reservation and the reserver user
    all u :User | all r: Reservation | (r in u.reservation) <=> (u in r.reservedBy)
    // a user cannot have two reservation in the same date
    no u:User,r,r':Reservation |r!=r'&&r in u.reservation && r' in u.reservation && r.date=r'.date
    // a reservation must be related to one and only one user
    no u,u':User| u!=u' && u.reservation = u'.reservation && #u.reservation!=0
    // cannot exists a date not related with a reservation
    all d: Date, r: Reservation | d=r.date
}
```

```
fact availableDriverFeatures{
    // the impossibility of having the same available taxi in two distinct queues
    all a : AvailableDriver | no disj q1, q2 : TaxiQueue | (a in q1.waiting) && (a in q2.waiting)
    // an avilable taxi must be inserted in a queue
    no a : AvailableDriver | #a.isMember=0
}
```

```
//Constraints on zones:
fact zoneFeatures{
    //we impose the maximum of 8 zone contigous to one
    all z : Zone | #z.contiguous < 8
    //a zone can't be contigous of itself
    no z : Zone | z in z.contiguous
    //if a zone is contigous to one other, this other is contigous to the previous one
    all z, z' : Zone | (z' in z.contiguous) <=> (z in z'.contiguous)
}
```

We report asserts and check statements:

```
//-------------------ASSERTIONS----------------
//Surely this set of assertions are insufficient for a nearly complete verfication
//of all the main properties of the generated world;
//this is only an exemplification of how we could use assertions to verify
//the presence of any counterexamples

// We want to verify that a user can't request a taxi while he's passenger on a taxi
assert NoRequestFromUserServed{
   no u:User, t:UnavailableDriver,r:Request | (u in t.serving) && (u in r.requestedBy)
}
check NoRequestFromUserServed

// We want to verify that a taxi can't have more than four passengers
assert NumberOfPassengers{
   no u: UnavailableDriver | #u.serving>4
}
check NumberOfPassengers
```

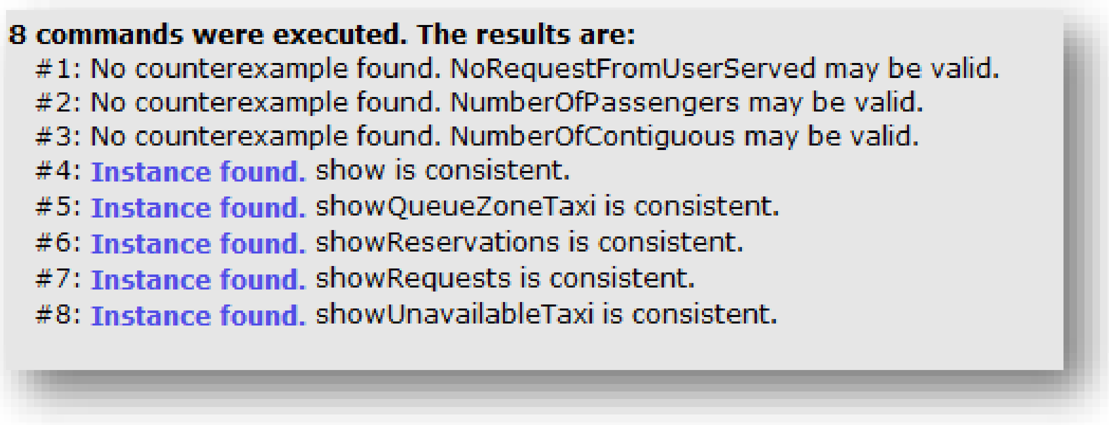
```
// We want to veirfy that a zone can't have more than eight contiguous
assert NumberOfContiguous{
    no z: Zone | #z.contiguous>8
}
check NumberOfContiguous
```

We report predicates and run statements:

```
//------------------PREDICATES---------------

//Show the entire world
pred show{}
run show for 6


//Show a world that contains TaxiDrivers associated to their Queues and then to their Zones
pred showQueueZoneTaxi{
   #MyTaxiDriver>1
   #TaxiQueue>1
}
run showQueueZoneTaxi for 7 but 0 UnavailableDriver, 0 User, 0 Request, 0 Reservation, 0 Date


//Show a world that contains at least one reservation and the associated elements
pred showReservations{
   #Reservation>0
}
run showReservations for 5 but 0 Request, 0 MyTaxiDriver


//Show a world that contains the different types of request and their related users
pred showRequests{
   #SimpleRequest=2
   #SharingRequest=1
   #User=3
}
run showRequests for 10 but 0 Reservation, 0 Date, 0 MyTaxiDriver


//Show a world that contains unavailable taxis which are serving some users
pred showUnavailableTaxi{
   #UnavailableDriver>1
}
run showUnavailableTaxi for 10 but 0 Request, 0 Reservation, 0 Date, 0 Zone, 0 TaxiQueue
```

6.2 EXECUTIONS

We report the execution result for each check and run statement:

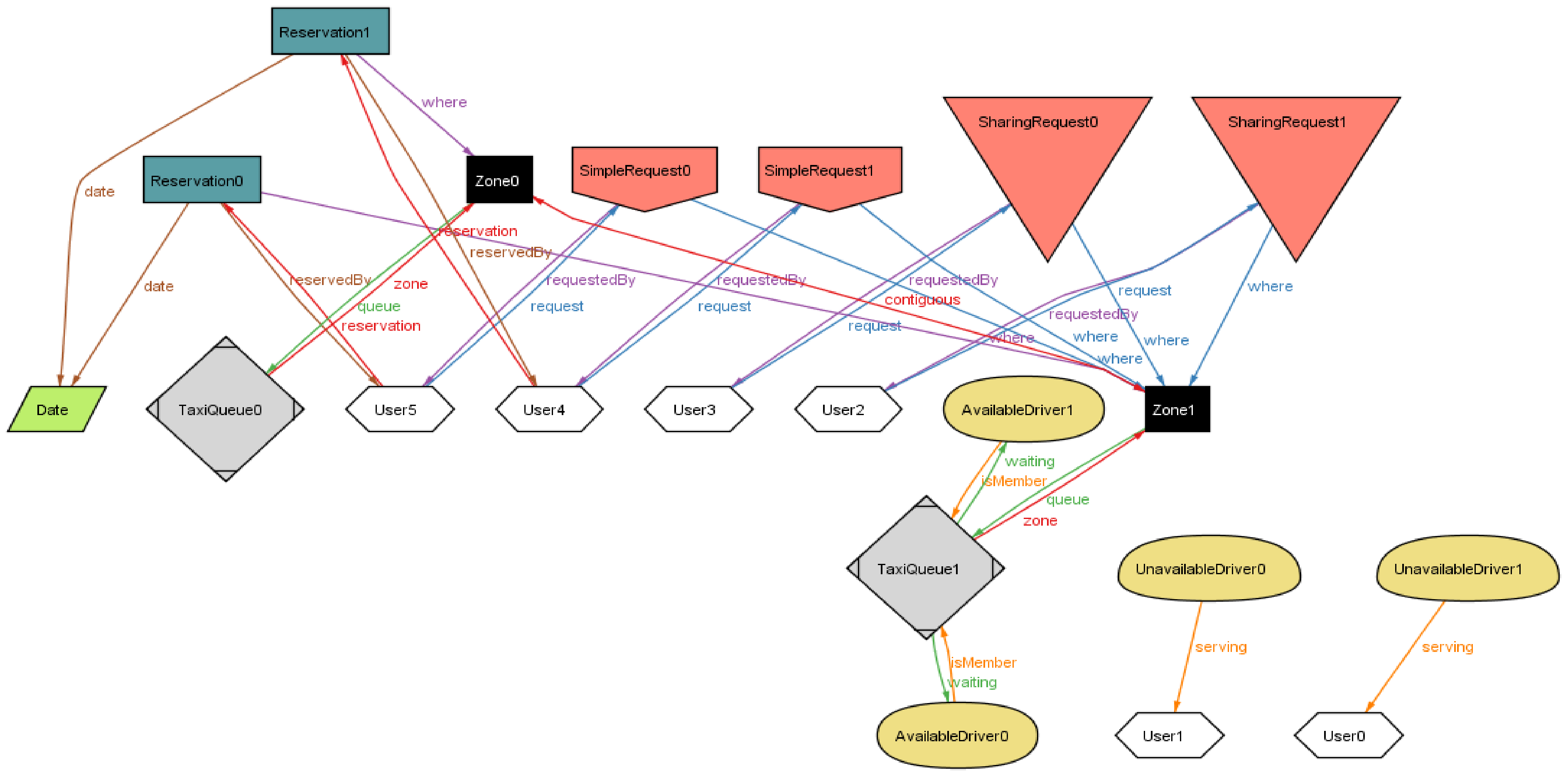


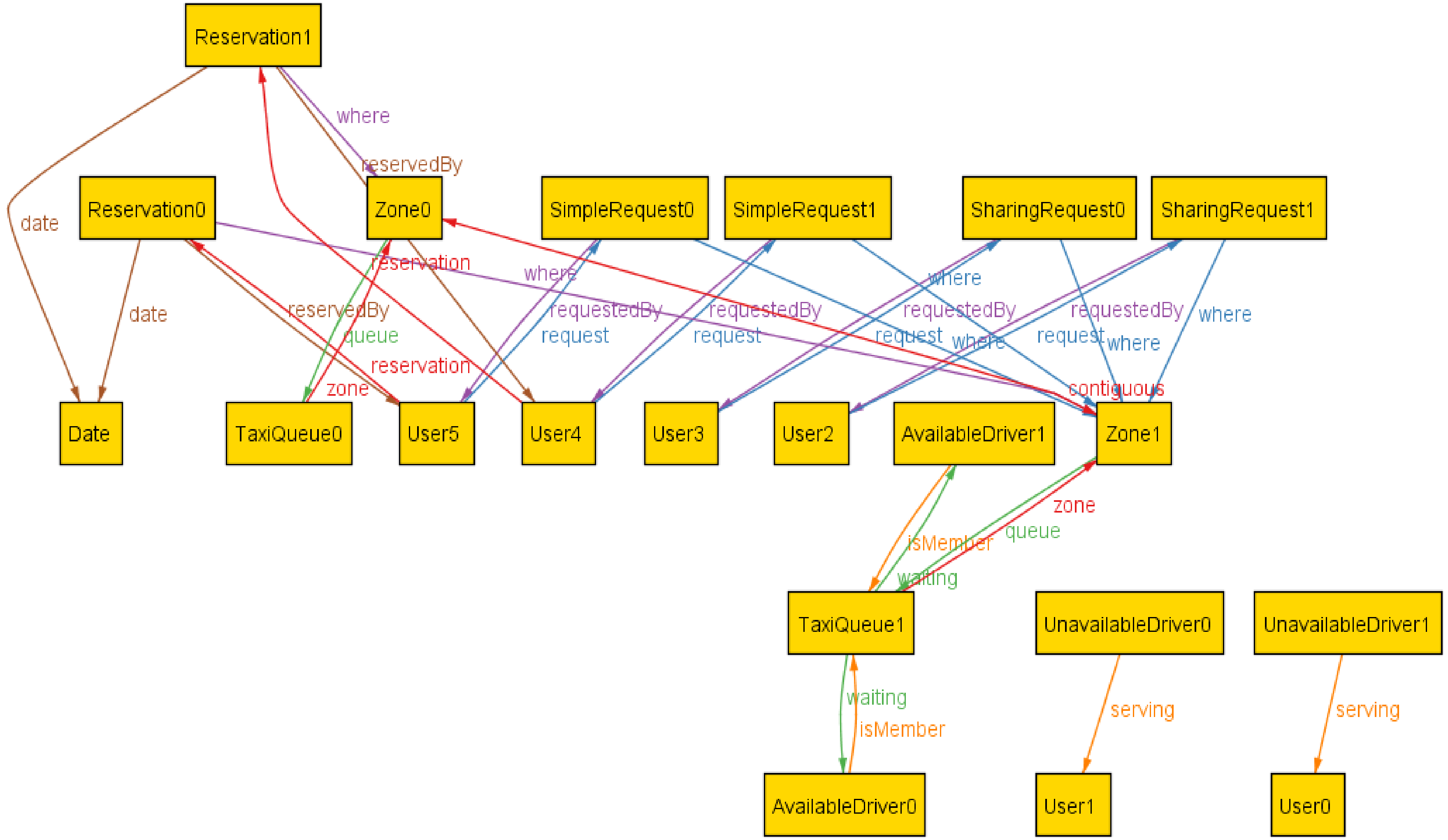
```
8 commands were executed. The results are:
  #1: No counterexample found. NoRequestFromUserServed may be valid.
  #2: No counterexample found. NumberOfPassengers may be valid.
  #3: No counterexample found. NumberOfContiguous may be valid.
  #4: Instance found. show is consistent.
  #5: Instance found. showQueueZoneTaxi is consistent.
  #6: Instance found. showReservations is consistent.
  #7: Instance found. showRequests is consistent.
  #8: Instance found. showUnavailableTaxi is consistent.
```

6.3 WORLDS

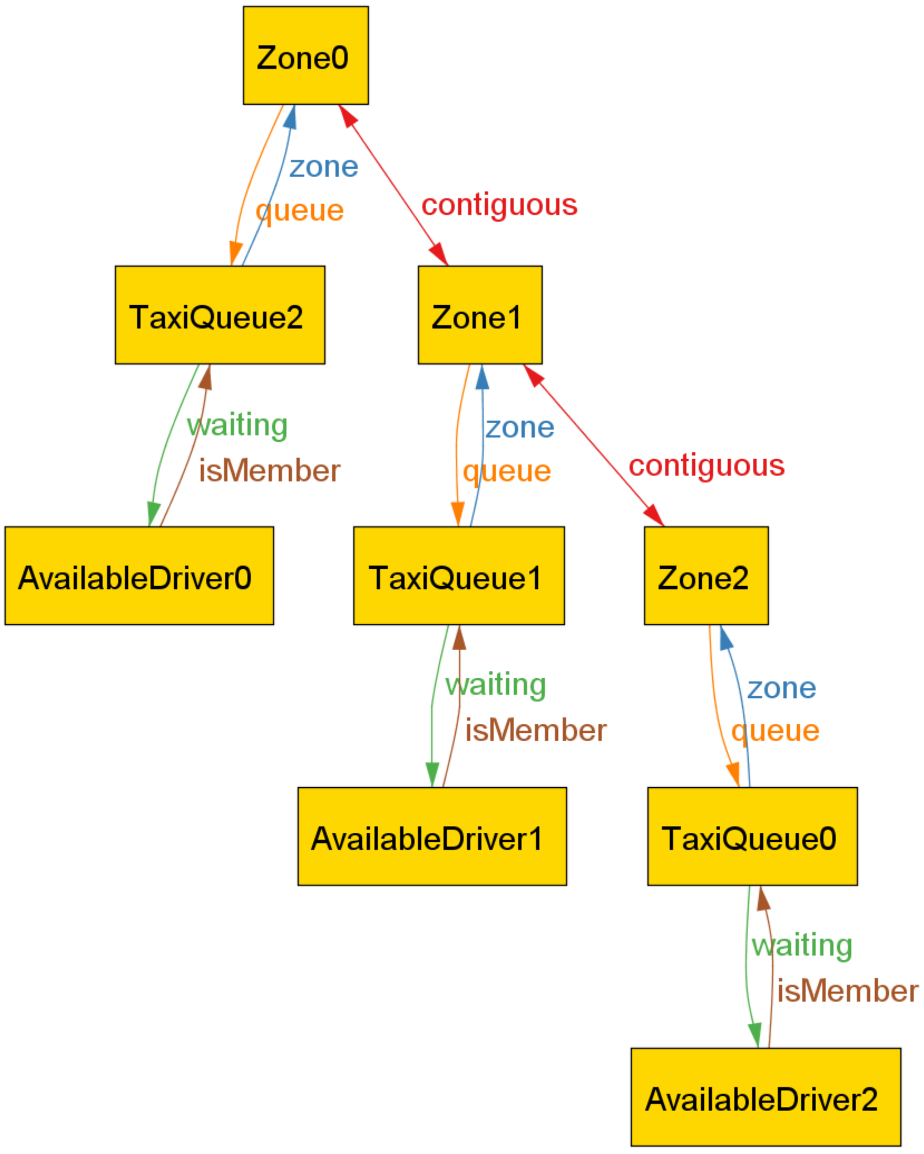
We report a significant sample of the Alloy generated worlds.

First of all, we report two snapshots of the entire world (generated from the execution of the predicate show):



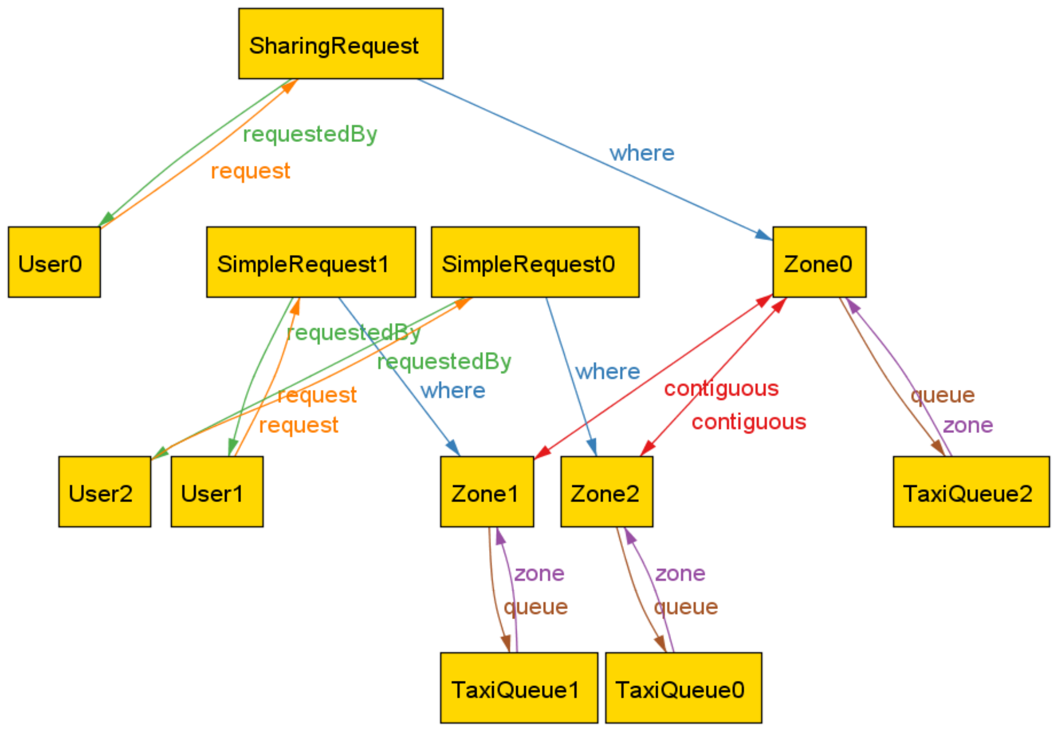

Reservation1
where
reservedBy
date
Reservation0
Zone0
SimpleRequest0
SimpleRequest1
SharingRequest0
SharingRequest1
date
reservation
where
requestedBy
request
requestedBy
request
where
requestedBy
request
where
requestedBy
request
where
where
reservedBy
queue
reservation
zone
contiguous
Date
TaxiQueue0
User5
User4
User3
User2
AvailableDriver1
Zone1
zone
queue
isMember
waiting
TaxiQueue1
UnavailableDriver0
UnavailableDriver1
waiting
isMember
serving
serving
AvailableDriver0
User1
User0

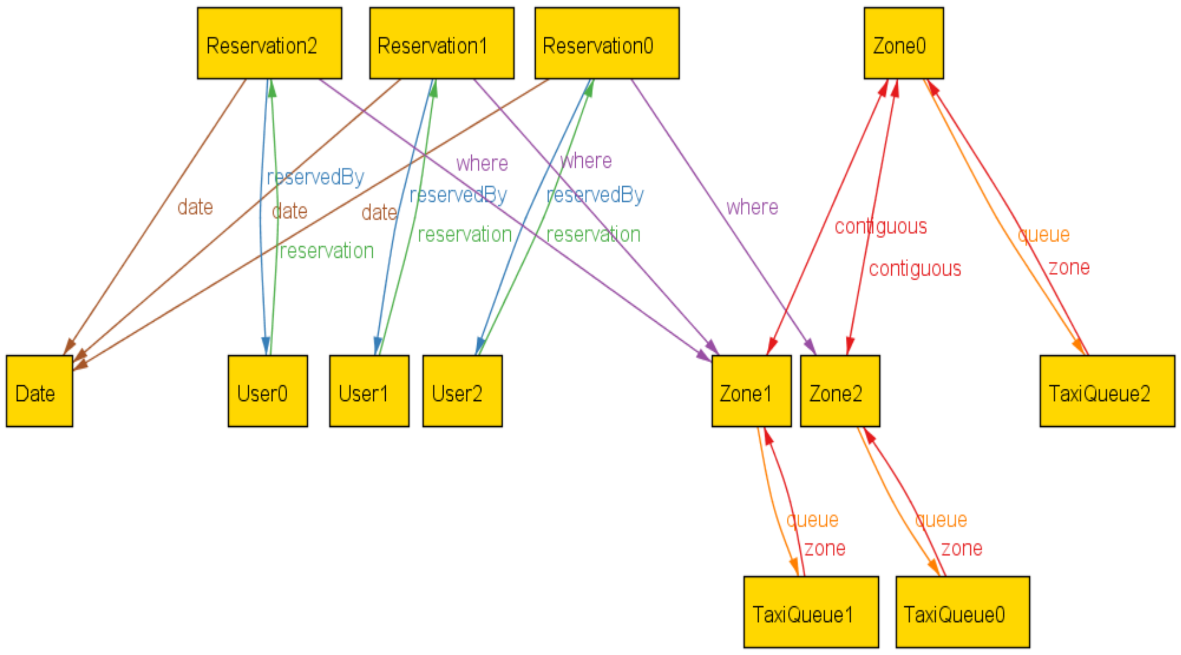
We report now a snapshot of a world generated running showQueueZoneTaxi. In this world are explicit the relation between the taxi queues and the corresponding zones, and also between available drivers and zones:



```
//Show a world that contains TaxiDrivers associated to their Queues and then to their Zones
pred showQueueZoneTaxi{
    #MyTaxiDriver>1
    #TaxiQueue>1
}
run showQueueZoneTaxi for 7 but 0 UnavailableDriver, 0 User, 0 Request, 0 Reservation, 0 Date
```

We report two snapshots of world generated form showReservation and showRequest execution, in which we underline the relation between a requestor and his request, a request and a zone, a zone and its queue:

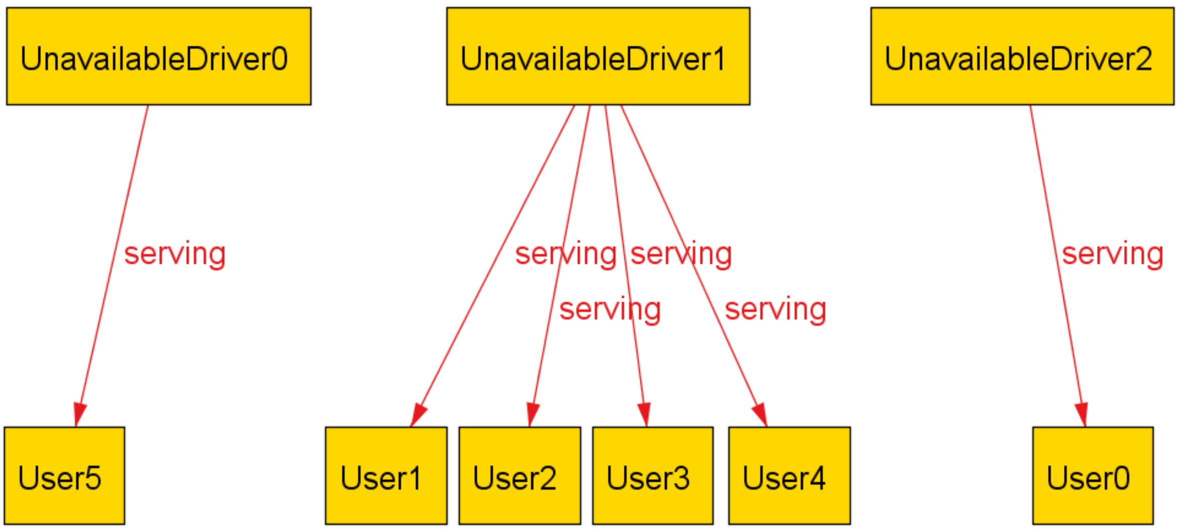




```
//Show a world that contains at least one reservation and the associated elements
pred showReservations{
   #Reservation>0
}
run showReservations for 5 but 0 Request, 0 MyTaxiDriver
```

```
//Show a world that contains the different types of request and their related users
pred showRequests{
   #SimpleRequest=2
   #SharingRequest=1
   #User=3
}
run showRequests for 10 but 0 Reservation, 0 Date, 0 MyTaxiDriver
```

Considering the showUnavailableTaxi execution we have a clear description of the relation between each unavailable driver and its passengers:



```
//Show a world that contains unavailable taxis which are serving some users
pred showUnavailableTaxi{
   #UnavailableDriver>1
}
run showUnavailableTaxi for 10 but 0 Request, 0 Reservation, 0 Date, 0 Zone, 0 TaxiQueue
```

7. USED TOOLS

The tools we used to create the RASD document are:

- Microsoft Office Word 2013: to release and format document
- Alloy 4.2 : to generate the world model
- Argo UML: to draw the class diagram
- DrawExpress app: to draw Sequence Diagram and State chart
- Balsamic Mockups3: to draw interface