Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 project: MyTaxiService

RASD

(Requirements Analysis and Specification Document)

Alessandro Pozzi (mat. 852358), Marco Romani (mat. 852361)

6 November 2015

Version 2.0

Contents

1. Introduction 3

1.1. Purpose 3

1.2. Scope 3

1.3. Goals 4

1.4. Definitions, acronyms, abbreviations 4

1.4.1. Definitions 4

1.4.2. Acronyms 5

1.4.3. Abbreviations 5

1.5. Identify Stakeholders 5

2. Overall Description 6

2.1. Product perspective 6

2.2. User characteristic 6

2.3. Constraints 6

2.4. Assumptions and dependancies 6

2.5. Future possible implementations 7

3. Specific Requirements 9

3.1. External interface requirements 9

3.1.1. Mobile Application 9

3.1.2. Web Application 14

3.2. Functional requirement 17

3.3. Non Functional requirements 19

3.4. The world and the machine 20

3.5. Scenarios 22

3.5.1. Scenario 1 22

3.5.2. Scenario 2 22

3.5.3. Scenario 3 22

3.5.4. Scenario 4 23

3.5.5. Scenario 5 23

3.6. UML Models 24

3.6.1. Use Case 24

3.6.2. Class Diagram 41

3.6.3. State diagram 42

4. Appendix 44

4.1. Alloy 44

4.1.1. Signatures 44

4.1.2. Facts 46

4.1.3. Predicates 48
4.1.4. Assertions 50
4.1.5. Generated world 51
4.1.6. Comments on Alloy 54
4.2. Software and tools used 54
4.3. Hours of work 54

1. Introduction

1.1. Purpose

This document represents the Requirements Analysis and Specification Document (RASD). Its main aim is to describe the system to be developed in terms of what the software will do and how it will be expected to perform. Functional and non-functional requirements will be comprehensively defined, as well as constraints, assumptions and the system limitations. It will also include a set of use cases and graphical mockups that describe the interactions the users will have with the application, so that the commissioner will have a clear idea of the aspect and functionality of the final product. This document may also be used by developers and programmers in order to extend this system or integrate it with existing ones.

1.2. Scope

The aim of this project is to model a software application whose intent is to provide an easy, comfortable and intuitive access to a city's taxi service. The name of the application is MyTaxiService (MTS).

The front-end layer of the system is composed by a web application, dedicated exclusively to people requesting the service, and a mobile application, dedicated both to taxi drivers and customers. Customers and taxi drivers have access to different functionalities. Access to the actual service on both sides is granted only to registered users, meaning that a registration is required. The only functionality available for guests is to visualize the homepage.

The back-end of the application manages the taxis' distribution around the city via GPS information and the forwarding of incoming requests to near taxis via a queue policy based on a taxi zones division. It also manages the forwarding of notifications to the users.

MTS functionalities on the customer side:

MTS allows customers to request a taxi in a specific location. In such case, the user is given a code and a waiting time estimated by the system. MyTaxiService also allows customers to make a taxi ride reservation for a specific time, origin and destination. The reservation must be done at least two hours before the given time, otherwise the system will send a notification about the rejected reservation's request.

MTS functionalities on the taxi driver side:

MTS allows the user to communicate his availability. If the user is available, the system can send him notifications about incoming requests. The user may or may not confirm his will to take care of the request.

1.3. Goals

[**G1**] Allow customers to access the system's taxi service in any moment, whether they are at home or anywhere else in the city.

[**G2**] Allow customers to request a taxi ride from an arranged location.

[**G3**] Allow customers to reserve a taxi ride at a specific time with a given origin and destination.

[**G4**] Allow taxi drivers to answer a ride request and take care of customers.

[**G5**] A ride request should always be satisfied within a considerable short amount of time, 15 minutes on average.

[**G6**] Allow customers to be notified of any relevant update connected to their requests and reservations.

[**G7**] Customers and taxi drivers must be able to contact each other after the system has paired them.

[**G8**] Allow customers to cancel requests and reservations.

[**G9**] Administrators must be able to assign and manage an account to the taxi drivers hired by the company.

1.4. Definitions, acronyms, abbreviations

1.4.1. Definitions

Customer –Registered user that may demand a taxi ride.

Taxi Driver – Employee of the taxi service.

Reservation – A ride that has been reserved by a customer using the web or mobile application. It becomes automatically a request 10 minutes before the established time.

Request – A customer's demand to be picked up by a taxi, sent using the web or mobile application.

Taxi zones –2km^2 areas in which the city's territory is divided. Each taxi queue is associated to a single taxi zone and vice versa.

Taxi driver availability – The status of a taxi driver. In detail, these are the 2 possibile status of a taxi driver:

- *Available* – When the driver has no customer to pick up and it's ready to accept new requests.
- *Busy* – When the driver is taking care of a request or of a standard customer. Note that the taxi driver is considered "busy" even if he hasn't pick up the customer yet, but has only accepted his request.

Customer and taxi driver pair – Sometimes taxi drivers and customers are said to be "paired": this happens only after a requests or 10 minutes before a reservation, when the system choose a taxi driver to serve the customer. Basically it corresponds to the ride status *Assigned*. This association ends when the taxi drops the customer to the requested destination.

In detail, these are the possible status of a taxi ride:

- *NotAssigned* – This is the default status of a taxi ride requested by customers. It simply means that the request (or reservation) has been accepted by the system but nothing else has been done yet.
- *Assigned* – Appears when the system has assigned a taxi to the customer.
- Completed – When the taxi driver has brought the customer to destination and has declared himself "Available".
- *Annulled* – When the customer has deleted the request (or reservation).

Standard customer – Person who is not using MTS's applications but may still require a ride with the "traditional" method.

1.4.2. Acronyms

MTS - MyTaxiService

1.4.3. Abbreviations

[Gn]: n-goal.

[Rn]: n-functional requirement.

[Nn]: n-non functional requirement.

1.5. Identify Stakeholders

Our main stakeholder is Prof. Raffaella Mirandola of Politecnico di Milano, DEIB. She gave us the delivery of the project. Other hypothetical stakeholders interested in the service offered by myTaxiService are:

- The government of the city, which has come up with the idea to improve the taxi service for its citizens.
- Citizens and tourists, who are the main users of the application.
- The taxi drivers, who are a small subset of users of the application.
- The taxi company, which of course provides the taxis.
- Other entities indirectly touched by the service, such as airports, hotels etc.

2. Overall Description

2.1. Product perspective

The applications to be developed is not completely independent and self-contained: it will rely mainly on the GPS system in order to retrieve the position of all its taxis. Moreover, we will promote possible future integrations and extensions by releasing a set of APIs. We hope that MTS may be exploited by other companies and developers and integrated with similar services (like journey planners or hotel booking services) to make travelling easier. APIs will also provide an easier development of additional functions, like taxi sharing or a SMS-based reservation system.

2.2. User characteristic

MyTaxiService will be used by two types of users: taxi drivers and customers. In both cases, no particular skills or information technology knowledge are required, since our application will be easy to use and accessible by any mobile phone or computer that have access to internet.

2.3. Constraints

- *Regulatory policies*

  MTS doesn't have to meet any regulatory policies.

- *Interfaces to other applications*

  MTS doesn't have to meet any interface to other applications.

- *Parallel operations*

  MTS system must support parallel operations and multiple connections in order to provide a smooth experience and a correct functioning.

2.4. Assumptions and dependancies

- Requests and reservations can be cancelled if and only if no taxi have been assigned to the customer.
- Customers and taxi drivers access to the same mobile app. The available functions will depend by the login phase, which determines the type of user connected.
- A taxi driver account cannot be used as a customer account, and vice versa. This means that if a taxi driver wants to access the customer's services, he will need to create a customer account.
- Taxi drivers will manually update their status (available or busy) with the mobile app every time they pick up or drop off standard customers. On the contrary, when drivers accept a ride request by MTS customers, the "busy"

status will be automatically set up by the system. Anyway, the “available” status must still be selected manually after the customer has been taken to destination.

- Taxi drivers have a fixed amount of time of 1 minute to answer a ride request. If the answer is not received, the system will consider the request as refused and act accordingly.
- Customers are allowed to perform unlimited taxi reservations. However, due to the fact that overlaps between two reservations are not predictable, the system will cancel any impracticable reservation identified at runtime.
- Customers are not allowed to request a taxi ride if any other request made by the same account has not been fulfilled yet.
- If and only if a taxi zone does not have any taxi available to answer a request, the system will search for an available taxi in adjoining zones. Worst case scenario: if there are no taxi available in the adjoined zones, the costumer should be notified and put in hold. During this period of time the costumer should be allowed to cancel the request.
- Taxis are assumed to be well distributed in the city’s taxi zones. This means that taxi zones never happen to be completely empty and the situation in which no taxi drivers are present, either available or busy, cannot occur.
- Payment and specific duties related to the taxi service are not considered and managed by the application. MyTaxiService is meant to be only an interface between customers and taxi drivers.
- The web and mobile registration is intended for customers only. Taxi drivers’ account are created by an administrator when they are hired by the taxi company. Taxi drivers will then receive their username and temporary password, which they will be able to change once logged in the application.
- The system will not accept reservation whose date is latter than a year in respect to the submission date.
- Taxi drivers can log into the web application with theirs driver’s account but cannot access to the customers’ service. The only available function will be the modification of some account information.

2.5. Future possible implementations

- The implementation of a taxi sharing service, which allows customers to share their ride with other users in order to decrease the cost of the ride.
- The implementation of a SMS system, which allows users to request or reserve a ride by sending a SMS. SMS syntax rules may be available online or may be sent by request with a SMS. This function may extend the service to those who does not have always access to an internet connection, like tourists.

- A function that estimates how much time a ride will take and how much it will cost. This function may be accessible even without requesting a ride.
- An improved taxi management system, which redistributes available taxis moving them from high density areas to low density areas.
- The possibility to show the GPS position of the customer's taxi on an interactive map.

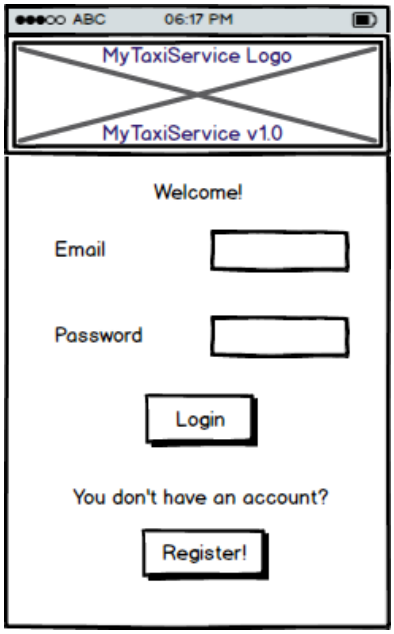
3. Specific Requirements

3.1. External interface requirements

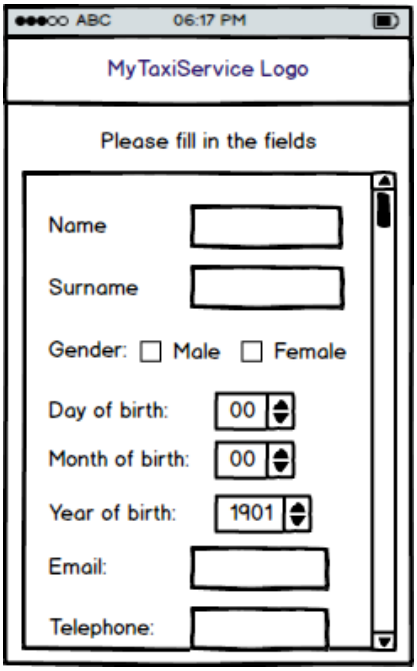
Below are shown some mockups that displays the general aspect, structure and main functionalities of the mobile and web application. Note that these mockups are only indicative, and the final version of the application may differ.

3.1.1. Mobile Application

Home - This is the layout of the main screen of the mobile application:

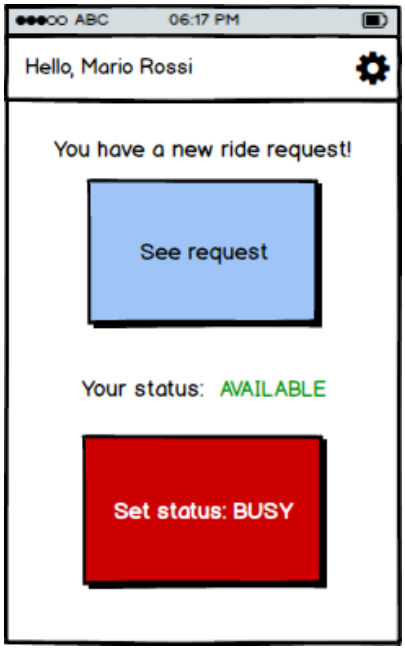


Registration - If the user taps on the "Register!" button from the application home, the following screen will appear:



In the final version some of the fields may be marked as compulsory.

Taxi Driver Home – If a user logs in with taxi driver's credential from the application home, the following screen will appear:

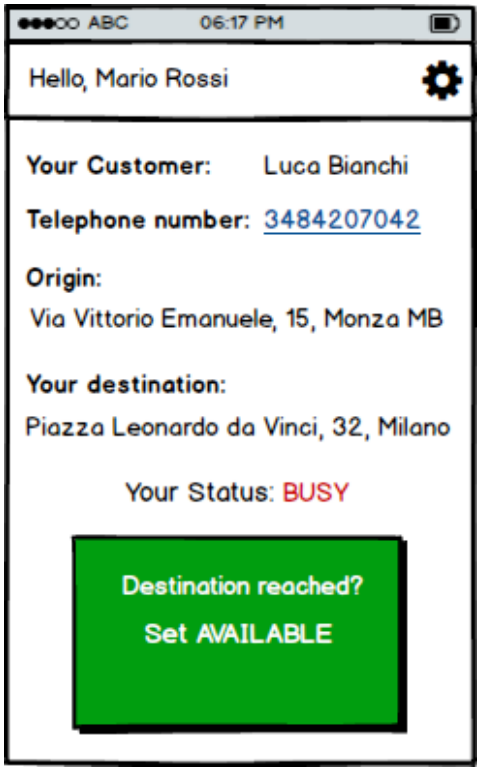


The "See request" button, as long as the "You have a new ride request" text, may not be present if no request have been received.

Taxi driver request details – The following mockup shows what the user will see if he presses the "See request" button from the previous screen.

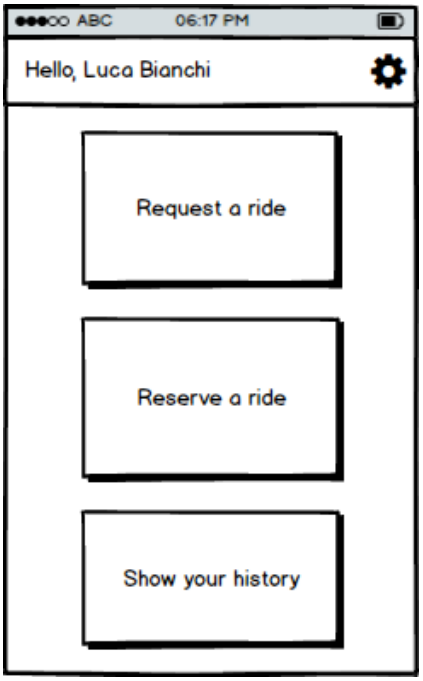
Note that the "Destination" field may be empty if the customer has not compiled it while making the request. If the user presses the "REFUSE" button, the *Taxi driver home* will appear, otherwise the following screen will be loaded:

Taxi driver busy status – This screen may be reached either by pressing the "ACCEPT" button in the *Taxi driver request details* or by setting the busy status in the *Taxi driver home*.

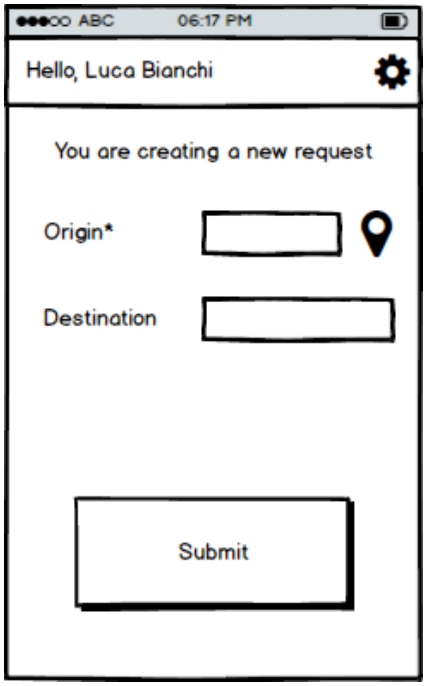


No customer information will appear if the taxi driver picks up a standard customer.

Customer home – This mockup shows the screen that will appear after a customer has finished the login procedure:

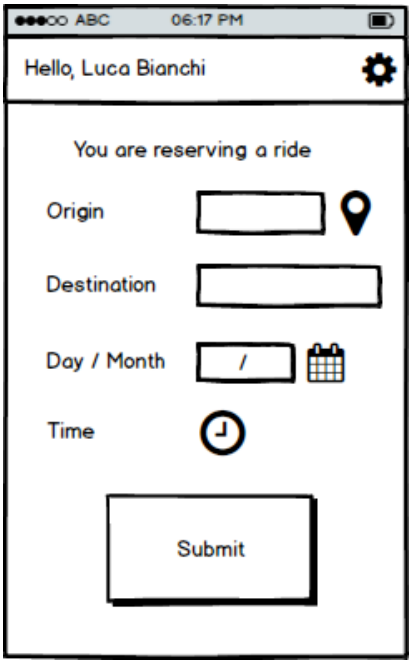


Customer ride request – This screen shows what happens when the “Request a ride” button is pressed in the *Customer home*.

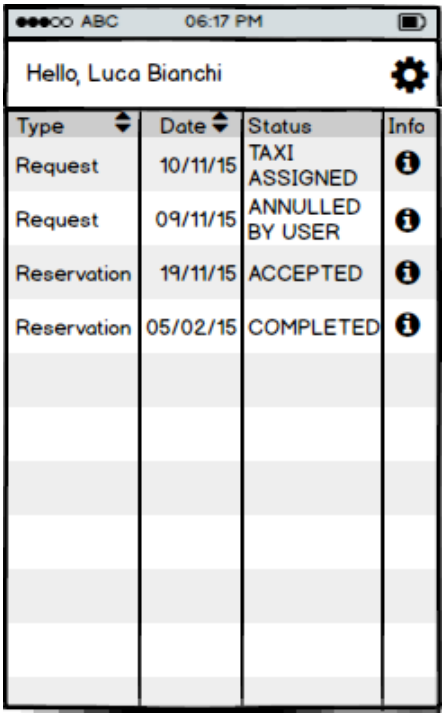


The asterisk indicates that the Origin field is compulsory, while the localization symbol indicates the possibility to locate the origin using the phone’s GPS system (if present).

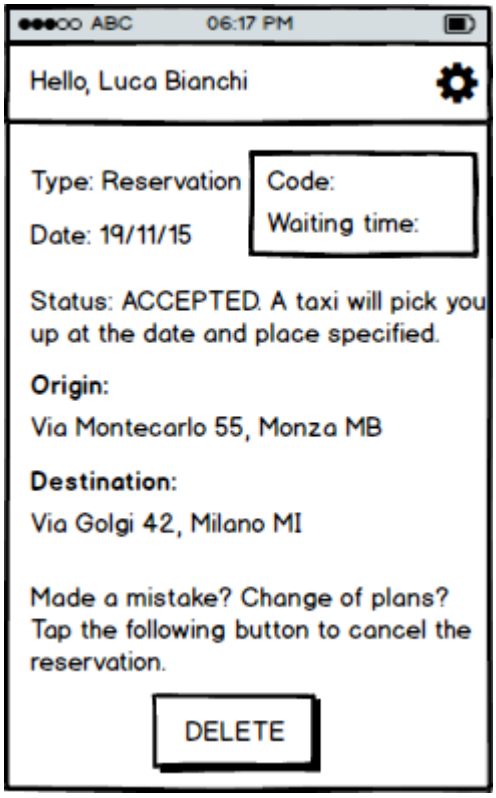
Customer ride reservation - This screen shows what happens when the “Reserve a ride” button is pressed in the *Customer home*.



Customer history – After pressing the “Show your history” button, the following screen will appear:



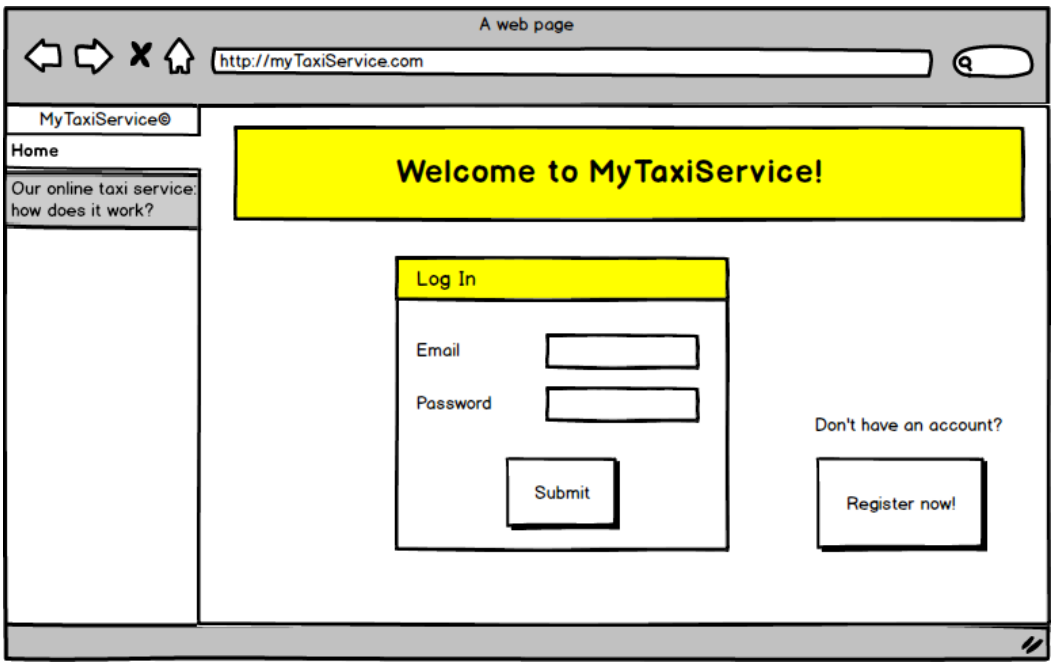
Customer Request/Reservation Info - By pressing the Info symbol in the *Customer history* screen, details about the request or reservation will be shown as follows.



The “Waiting time” and “Code” box will be filled with the time to wait for the taxi to arrive and the correspondent taxi code only when a taxi driver will be assigned to the customer.

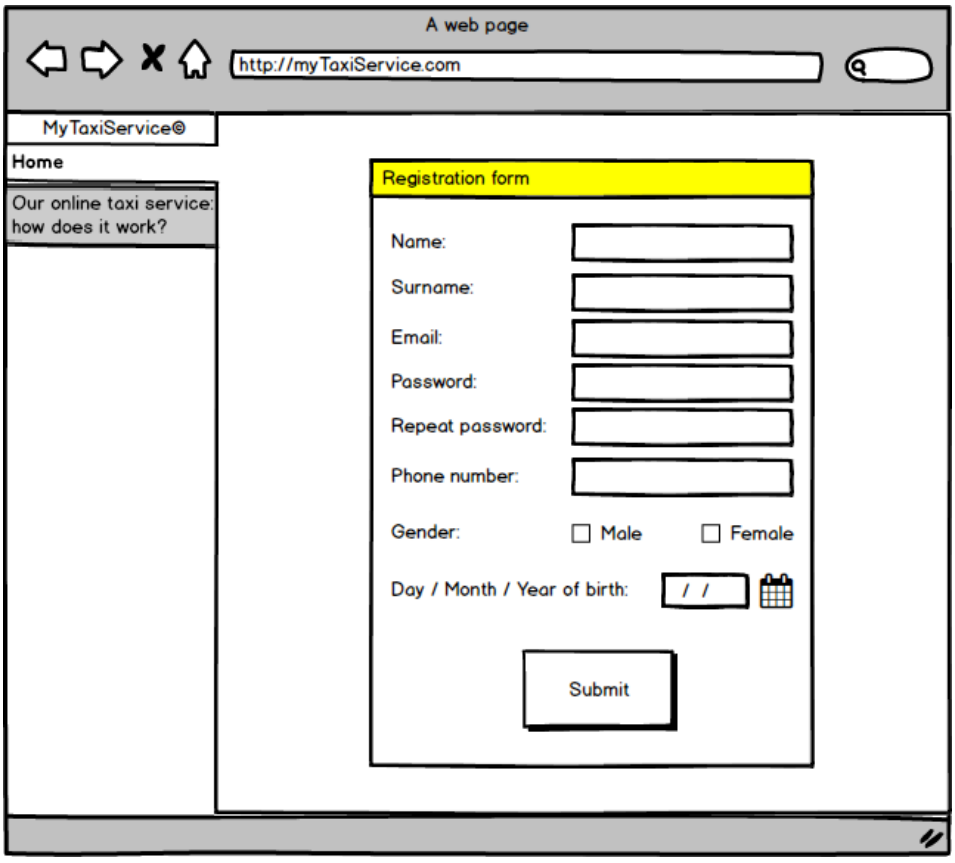
3.1.2. Web Application

Home – This is how the web page home of MTS should appear:

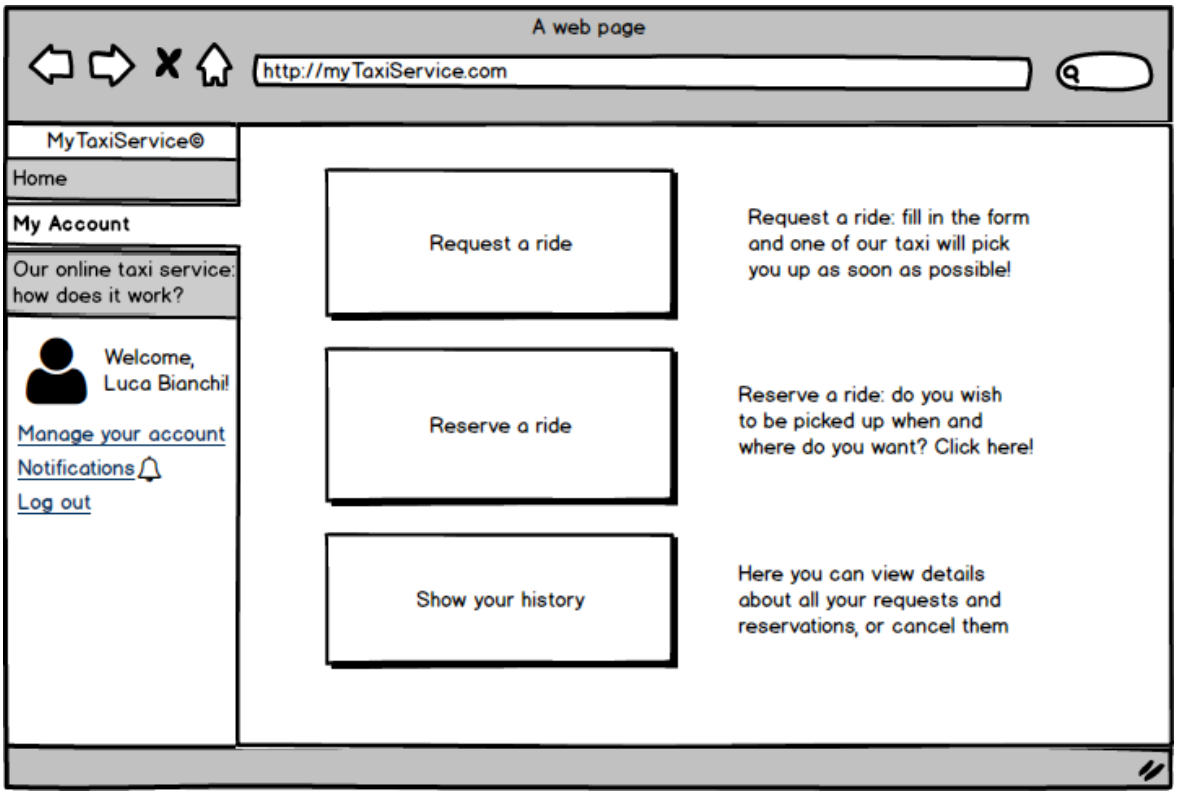


On the left bar there is a possible clickable tab that may explain the service functionalities.

Registration form – This will appear when the "Register now" button is pressed in the home page:

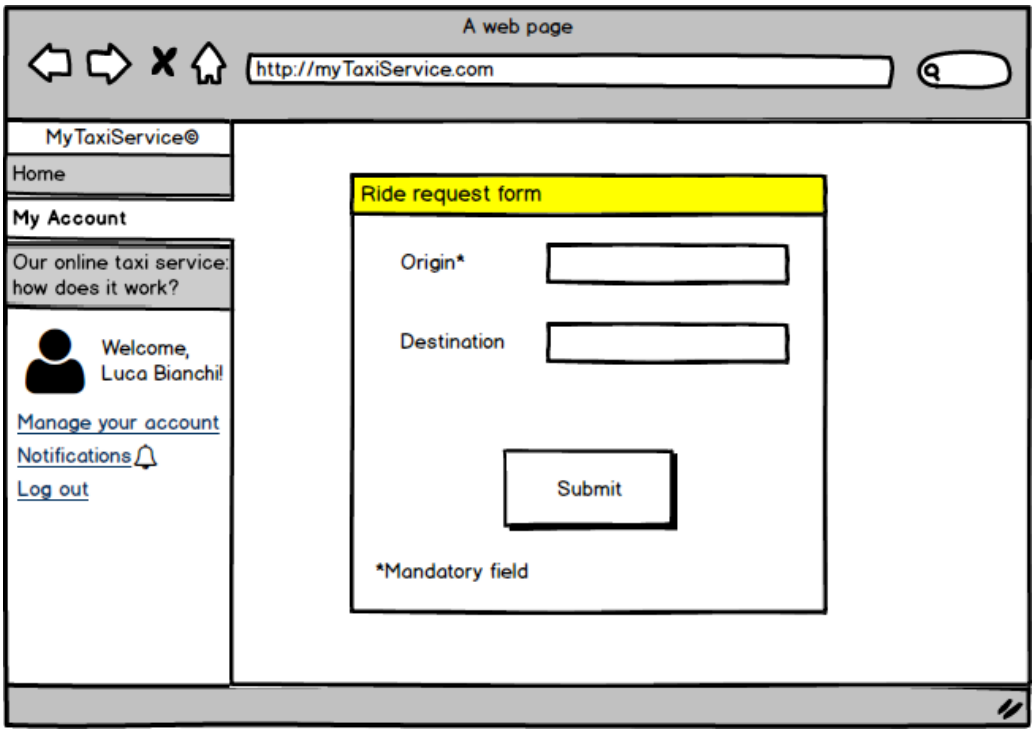


Customer home – When the user has logged in, the following page will load:

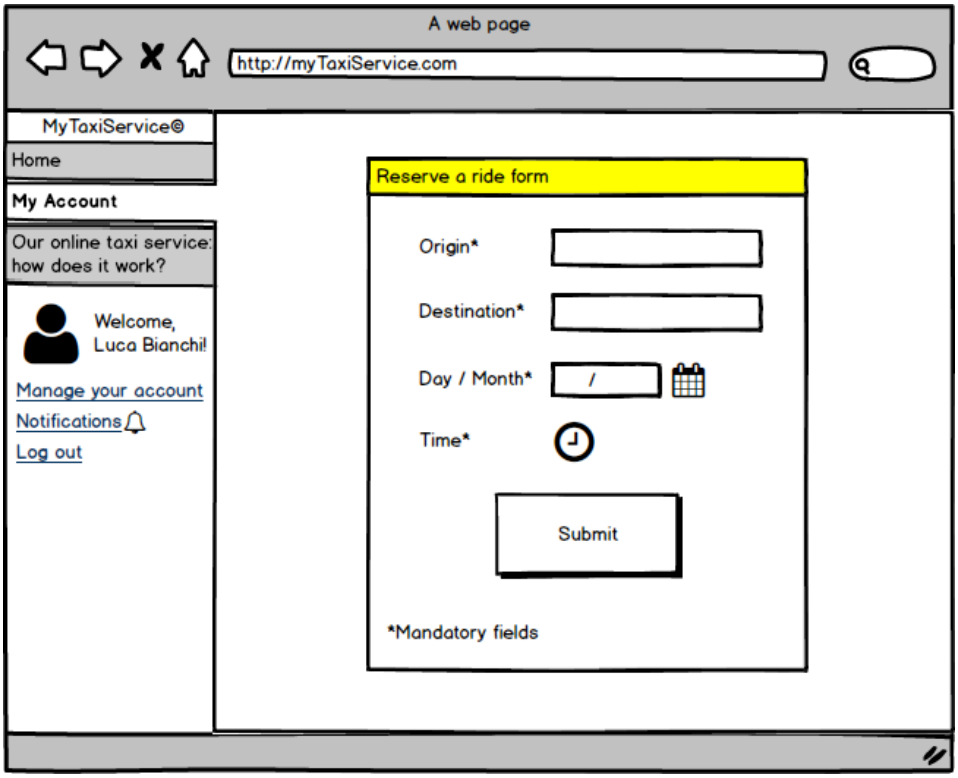


Notice that a taxi driver user won't have any of these functionality available, except for "Manage your account" and "log out". "Notifications" is reserved to customer and allows them to be alerted if some events occurs.

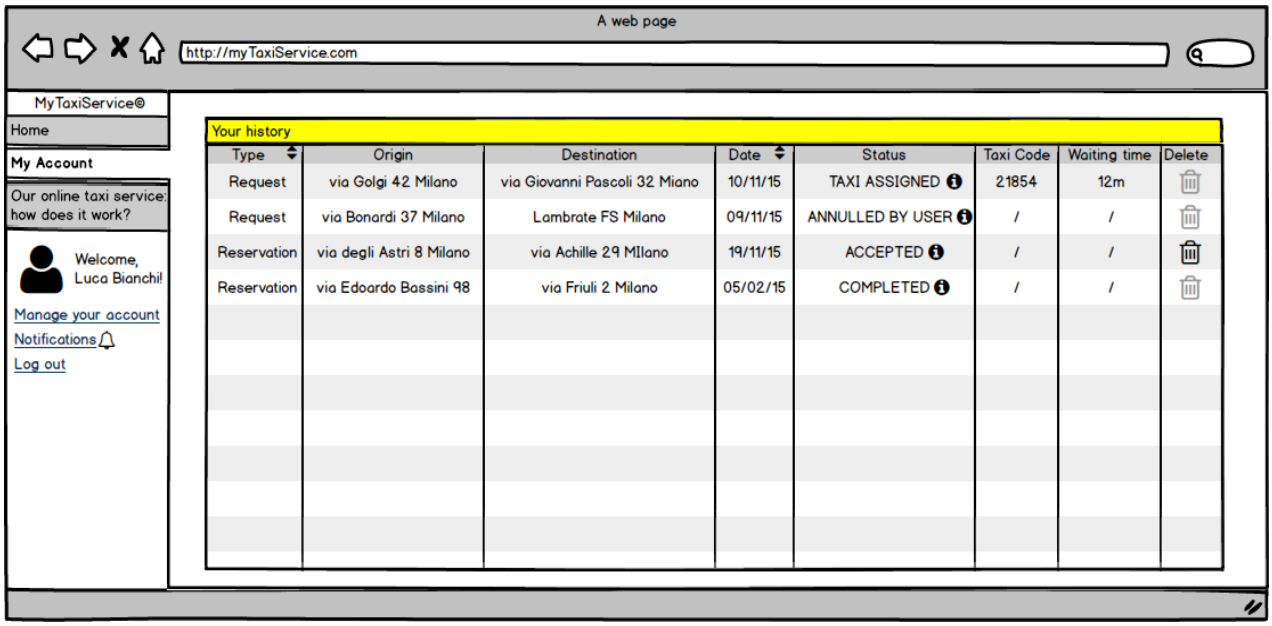
Customer ride request – The following form will appear when "Request a ride" is pressed:



Customer ride reservation – The following form will load when the "Reserve a ride button" is pressed:



Customer history – This page will appear when the "Show your history" button is pressed.



Notice that the "Status" fields will be explained by clicking the info buttons. "Taxi code" and "Waiting time" columns will present a value only when the status field is "Taxi assigned". Finally, the "Delete" icon will be clickable only if a request/reservation has no taxi assigned and has not been competed yet.

3.2. Functional requirement

*[**G1**] Allow customers to access the system's taxi service in any moment, whether they are at home or anywhere else in the city.*

[R1] Customers should be able to access the service through both the web and the mobile application, even at the same time.

[R2] Customers must be able to register to the taxi service from the mobile or web homepage.

[R3] Only registered customers can access MyTaxiService's services.

[R4] The system should allow the log out functionality.

*[**G2**] Allow customers to request a taxi ride from an arranged location.*

[R1] Only registered customers can request a taxi ride.

[R2] Customers must insert a valid origin location in order to request a ride.

[R3] The system will not allow more than a request if the previous one (either request or reservation) has not been accomplished yet.

*[**G3**] Allow customers to reserve a taxi ride at a specific time with a given origin and destination.*

[R1] The system should allow taxi reservations for a specific path communicated by the customer.

[R2] The system must not allow overlaps between reservations (or requests) made by the same customer.

[R3] The system allows reservations only 2 hours before the time and date specified by the customer.

[R4] The system will assign a taxi driver for the reserved ride 10 minutes before the time and date specified by the customer.

*[**G4**] Allow taxi drivers to answer a ride request and take care of customers.*

[R1] Taxi drivers should be able to communicate their current availability state to the system.

[R2] If available, taxi drivers should be able to receive incoming requests.

[R3] After receiving an incoming request, the taxi driver should be able to either confirm or not his intention to take charge of the request.

[R4] Taxi drivers must be able to log in the mobile application with preassigned credential and be identified as drivers.

*[**G5**] A ride request should always be satisfied within a considerable short amount of time, 15 minutes on average.*

[R1] The system should always search an available taxi giving maximum priority to the taxi zone related to the request and lower priority to the immediate near zones. Any other taxi zone should be ignored.

[R2] If no taxis are available in the zones specified in the previous requirement, the system should put the request on hold and periodically check again the taxi availability.

*[**G6**] Allow customers to be notified of any relevant update connected to their requests and reservations.*

[R1] The system should send updates through email and/or in-app notification, as specified by the customer.

[R2] Absence of taxis available, reservations overlaps, taxi average waiting time and taxi assigned to customers are events that must be notified to the customer.

*[**G7**] Customers and taxi drivers must be able to contact each other after the system has paired them.*

[R1] Customers must leave a valid phone number in order to complete the registration phase.

[R2] Taxi drivers must be able to access to the customer's phone number when the system has paired them.

[R3] Customers must receive the taxi drivers' contact number after the system has paired them.

[R4] Customers must receive the taxi code in order to be able to recognize its driver.

*[**G8**] Allow customers to cancel requests and reservations.*

[R1] Customers can cancel a request or reservation only if it has not been assigned to a taxi driver yet.

*[**G9**] Administrators must be able to assign and manage an account to the taxi drivers hired by the company.*

[R1] Administrators must be able to create a taxi driver's account.

[R2] Administrators must be able to delete a taxi driver's account.

3.3. Non Functional requirements

[N1]. MyTaxiService's system should be available 24/7.

[N2]. The waiting time for a taxi to come that is showed to customers should be updated every 30 seconds.

[N3]. Every functionality offered by the mobile or web application after the login should be reachable within three clicks.

[N4]. Mobile applications' button available to taxi drivers should be big enough to be easily recognized and pressed while driving.

[N5]. Code should be well documented in order to facilitate extendibility.

[N6]. Email sent to the user in order to confirm registration should arrive in less than 10 minutes.

[N7]. Web and mobile applications should have a similar graphics so that the correlation between the two would be immediately identified by customers.

[N8]. Mobile's notifications should appear even when the user has not the app opened.

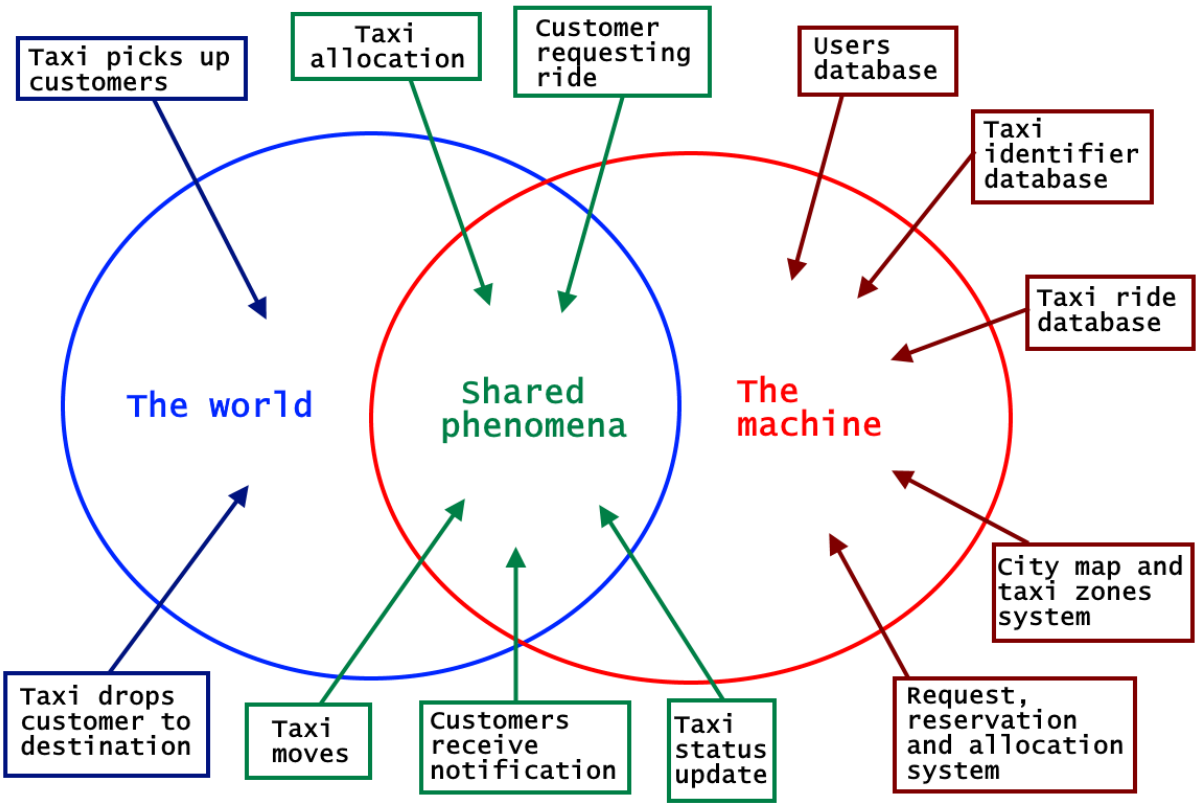
[N9]. The description of the mobile application in the phone app store should point out clearly and briefly its functions, allowing a quick comprehension from any type of user.

[N10]. MyTaxiService's applications should always have imperceptible response time during the navigation within its forms.

3.4. The world and the machine

The model proposed by M. Kackson & P. Zave, "The World & the Machine", allow us to analyze the domain of MyTaxyService application and its relation with the world. This model contemplate the presence of two areas: *The Machine*, which is the portion of the system to be developed, and *The World*, which is the portion of the environment that is affected by the machine. The intersection of the two is called *Shared Phenomena* and is composed by all the entities that are controller by only The World or The Machine, and observed by the other.

The following diagram shows MyTaxiService's *The World and The Machine* analysis.



Entities analysis:

The world

"*Taxi picks up customers*" and "*Taxi drops customer to destination*" are the only two phenomena that happen exclusively in the world. In fact, the machine can observe them only indirectly by monitoring the status of the taxi and its position.

The Machine

The machine is composed by the "*Request, reservation and allocation system*" which manages the main application logic, and is strictly dependent upon the "*City map and taxi zones system*", the "*Taxi identifier database*", the "*Taxi ride database*" and the "*Users database*" entities. In detail, the "*Taxi ride database*" will store all the information regarding actual and past taxi rides; the "*Taxi identifier database*" will contain all the information about taxis and their assigned drivers.

The Shared Phenomena

All the entity in this subset are particularly important because of their nature of "meeting point" between the system and the reality. The correct interpretation of this phenomena is the key on developing a consistent and functional application. Here such entities are briefly described:

"*Taxi moves*" is a shared phenomenon, which is controlled by the world and observed by the machine through the GPS system.

"*Taxi status update*" is also a phenomenon controlled by the world (i.e. the taxi driver that changes their status by picking up and dropping of customers) and observed by the system.

"*Taxi allocation*" is a particular entity: if we consider MTS customers, is observed by the world and controlled by the machine (which sets the taxi's next destination accordingly to the customer requests); if we consider standard customers, then it is observed by the machine (with the GPS and taxi zones system) and controlled by the world (i.e. the standard customer).

Similarly, "*Customers receive notifications*" is machine-controlled, since customers receive updates about their rides by the system.

Instead, "Customer requesting ride" happens in the world and is only observed by the machine, which will react accordingly.

3.5. Scenarios

3.5.1. Scenario 1

Daenerys must reach the city's harbor to get on a ship and cross the narrow sea, where she has an important appointment. Unfortunately, she can't use her car because it is being repaired due to an engine failure. She consider to postpone the appointment, when suddenly she remember about an online taxi service a friend had told her about. She turn on her laptop and after a google search she finds MyTaxiService's website. She clicks on "Register" and fill all the forms. A few minutes later she receive the confirm of the registration by email. Daenerys logs in the service and requests a taxi at her house.

A taxi driver called Jorah, who had just dropped off another client just a few blocks nearby, quickly accepts the request. Less than five minutes later, Jorah arrive at her apartment and call her at the phone number provided by the application. Daenerys answer the call and reaches Jorah's taxi, who finally bring her to the harbor.

3.5.2. Scenario 2

Mr. White needs to take a train in the early morning for a business related matter, but his car is currently not available and his wife is not enthusiastic at all about waking up when not even the sun is out yet. Knowing that, the previous day Mr. White cautiously decided to reserve himself a taxi. Since he is registered to MyTaxiService, he simply turned up his laptop and logged to the web application as he did many times before.

His taxi is scheduled for the 4.30 A.M. and the meeting place is just at the end of the street that leads to his house. Mr. White wakes up, silently takes his breakfast, prepares himself and leaves the house 20 minutes in advance, just to be sure to make it in time. It's very cold out there and there is almost no one around, probably because they are all sleeping in their warm beds. Actually, there is someone around, and Mr. White knows him: he is his neighbor, Mr. Pinkman. After a short talk, it turns out that Mr. Pinkman is going to the station too, and he offers a ride to his friend. Therefore, Mr. White takes out his mobile phone from his pocket and logs in to MTS application in order to check if it is still possible to cancel the reservation. Luckily, the delete button is still available and M. White quickly presses it and jumps into the car.

3.5.3. Scenario 3

It's Thursday afternoon, Lorenzo has just finished an intense day at the Politecnico (of course he is not studying neither Architecture or Engineering Management) and he is on his way to Lambrate station looking forward his comeback home and his Mathematical analysis exercises that waits him on his desk.

There is only a problem: Trenord has decided to completely collapse on itself… actually, it's not big news. Lorenzo's train is scheduled with 120 minutes late. That is not acceptable, his beloved homeworks are waiting him, but Lorenzo has an ace up his sleeve. He takes his mobile phone, taps on MyTaxiService application and clicks on the button "Request a ride". Then he quickly compiles the form and sends his taxi request to the system.

However, there is something he did not take into account: it's a rush hour and he is not the only one affected by train delays. The probability to be picked up in a couple of minutes is lower than usual.

Meanwhile, the taxi driver Carlo has just finished to take care of a ride near Piola subway station and is currently available. He looks at his phone and sees and incoming request notification. He is going to tap on the "Accept request" button but right that moment a customer appears on his road, waiving his hands. Carlo makes him get on the taxi, looks back to his phone and taps on the "Refuse request" button. "Someone else will take care of that request" he thinks.

3.5.4. Scenario 4

Mr. Grimes is a manager of the company "Spice up your life". He is going to have a very full week and he needs the help of MTS application. He seats on his chair and using his laptop he gets access to MTS web application. His intention is to make a series of taxi reservations, one for each day starting from Monday until Friday. However, he accidentally makes two reservations on Wednesday. The two reservations differ of just five minutes.

Monday and Tuesday pass by and now is Wednesday. Mr. Grimes did not spotted the mistake and he is simply going to the meeting point of his reservation. Suddenly, five minutes before the meeting, his cellphone trills.

Mr. Grimes looks at it and sees the notification of the deleted "phantom" reservation. He shakes his shoulders and puts the phone back into the pocket. Anyway, a few minutes later a taxi regularly picks him up.

3.5.5. Scenario 5

It's late evening and the taxi driver Ryan Gosling is waiting in his taxi: just 10 minutes are left before the end of his turn. He is already thinking about the warm bath that awaits him at home, when suddenly a request notification flashes on the screen of his mobile phone. He taps on the "Show request details" button to see the info related to the request. The origin point is not too far from there, but the destination is at the opposite side of the city. Therefore, Ryan decides to refuse the request. "I'm not paid enough for this", he mumbles while making his way home.

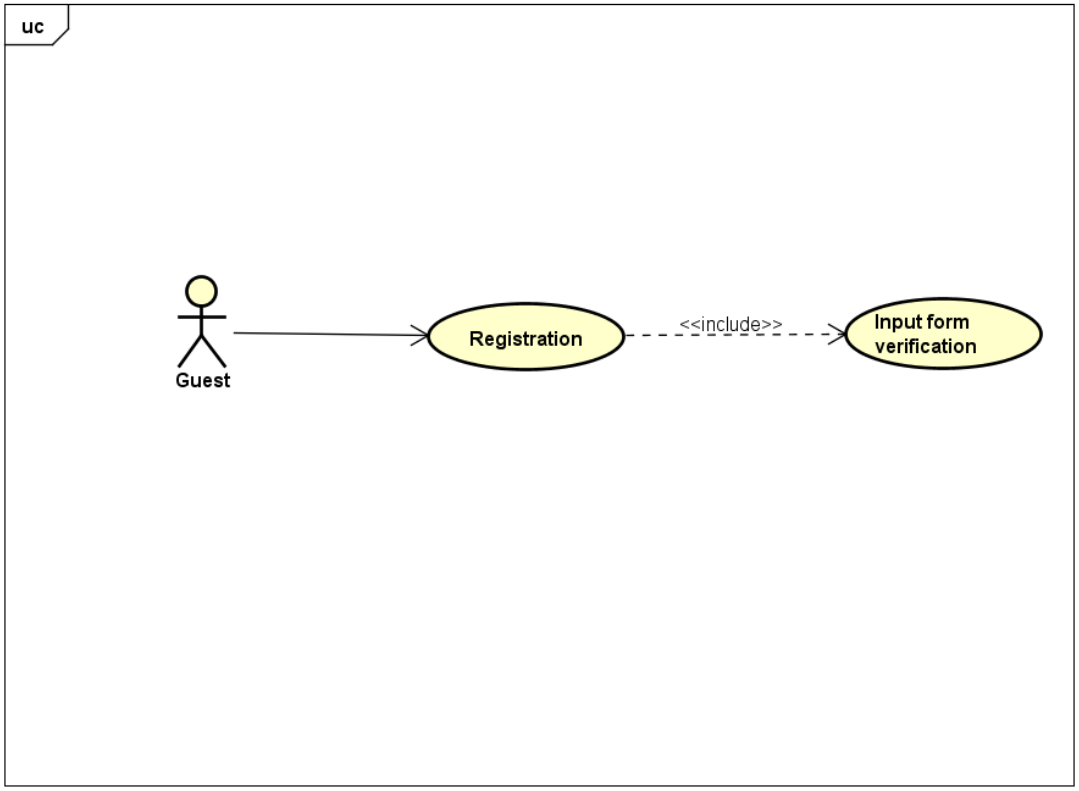
Luckily, the system efficiently and transparently assigns another driver to the request in a couple of minutes.

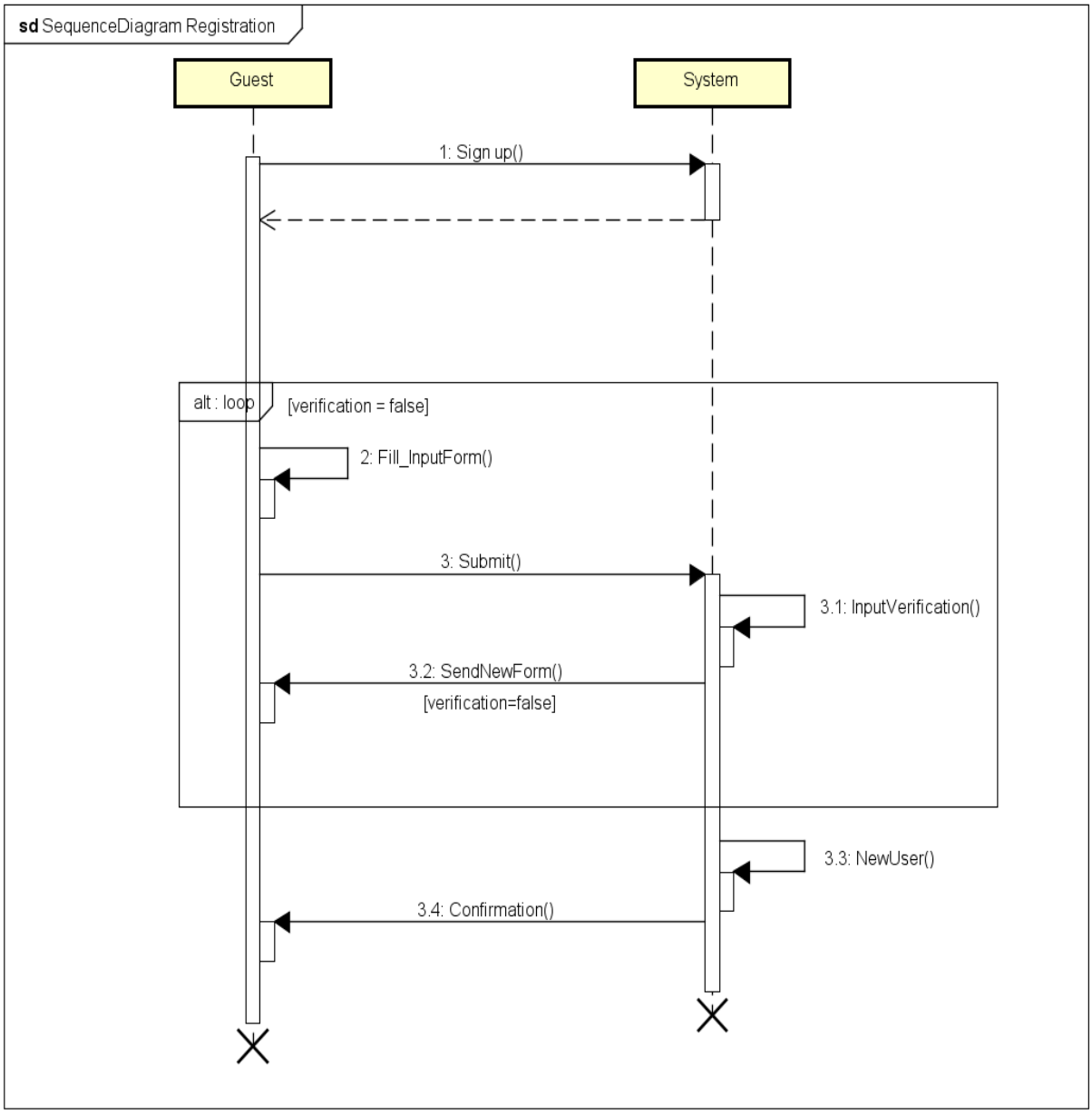
3.6. UML Models

3.6.1. Use Case

3.6.1.1. Registration

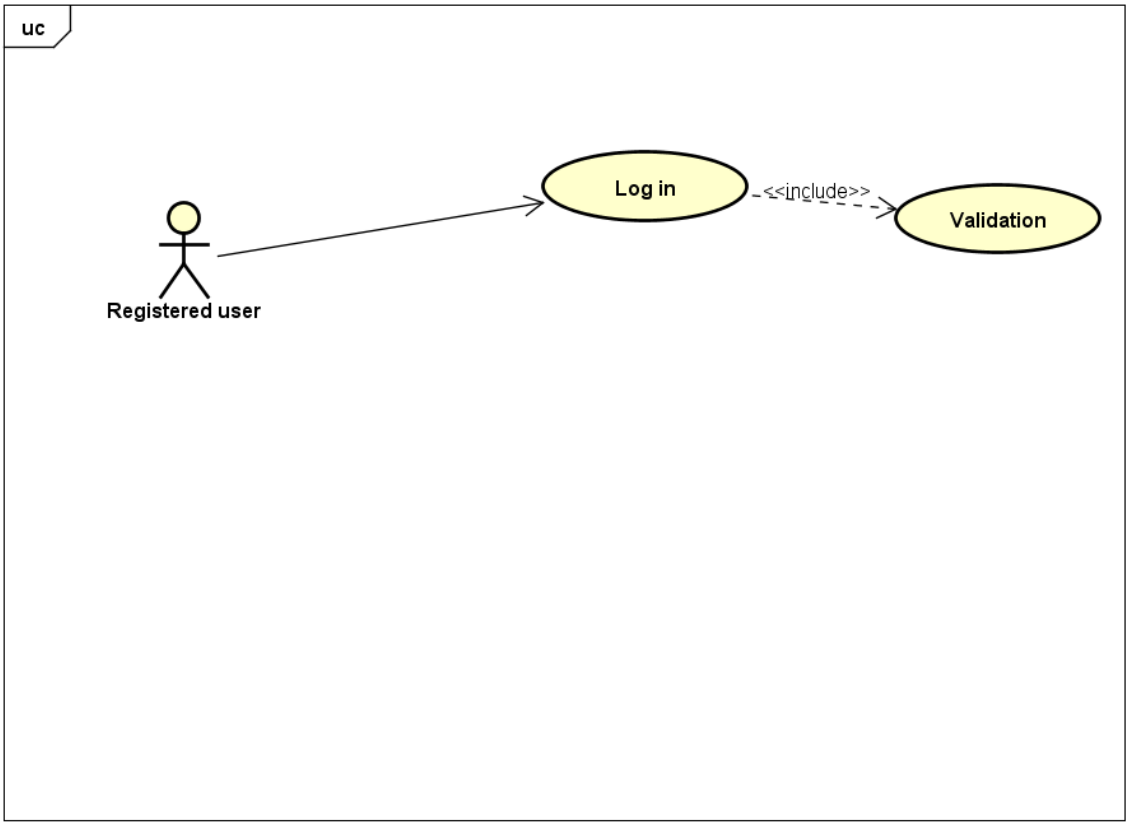
Name	Registration
Actors	Guest
Entry conditions	Null
Event flow	1. Guest clicks on “Register” button on the homepage, either via mobile app or web app. 2. Guest fills the registration form with at least the mandatory fields. 3. Guest clicks on “Submit” button. 4. The system verifies the guest’s inputs 5. The system creates the new customer’s account and sends a confirmation to the new registered user.
Output conditions	Registration done and new customer added to the system.
Exceptions	1. Guest does not fill correctly the fields. 2. Guest is already a user and the account already exists. 3. Mail, username or phone number are not valid or already taken by another user.

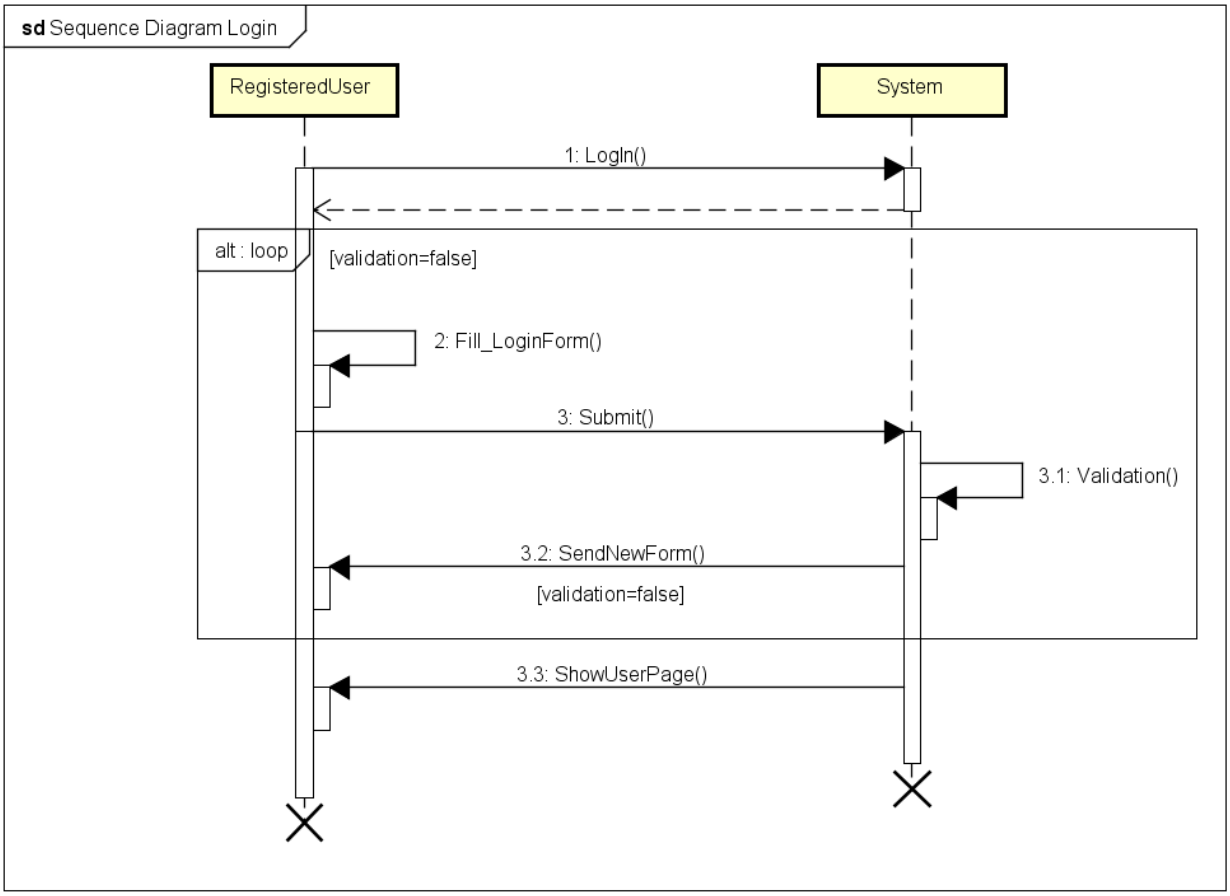

uc
Guest
Registration
<<include>>
Input form verification


sd SequenceDiagram Registration
Guest
System
1: Sign up()
alt : loop
[verification = false]
2: Fill_InputForm()
3: Submit()
3.1: InputVerification()
3.2: SendNewForm()
[verification=false]
3.3: NewUser()
3.4: Confirmation()
powered by Astah

3.6.1.2. Log in

Name	Log in
Actors	Registered user
Entry conditions	User is already registered into the system.
Event flow	1. User clicks on “log in” button on the homepage. 2. User fills the fields of the log in form. 3. The system verifies the fields filled by the user. 4. The system shows the user his personal page.
Output conditions	Registered users are granted access to their personal page.
Exceptions	The user does not fill correctly any field of the log in form and receives another log in form.





powered by Astah

3.6.1.3. Request and reservation

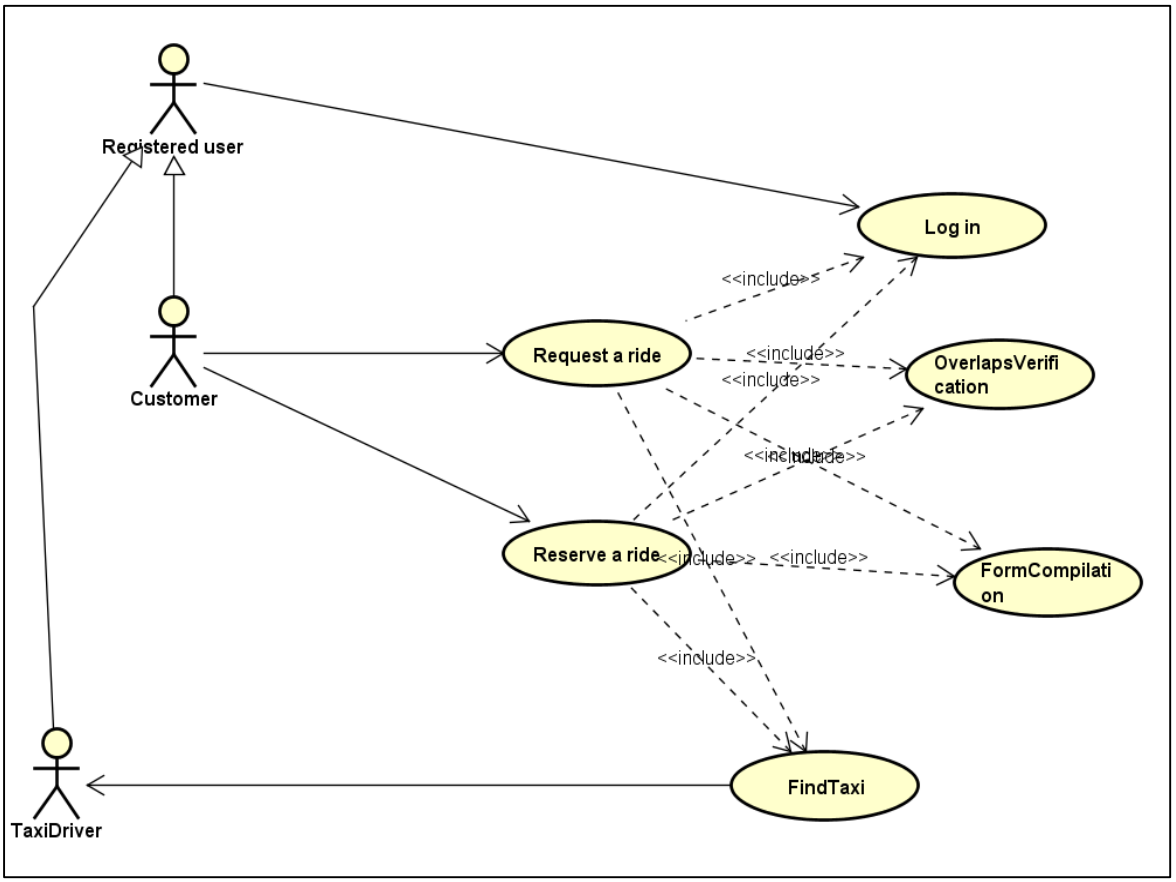
Request a ride table:

Name	Request a ride
Actors	Customer, Taxi Driver
Entry conditions	Customer and Taxi Driver are registered and logged into the system.
Event flow	1. Customer clicks on “Request a ride” button on his user page. 2. Customer fills the input form with at least the Origin field which is mandatory. 3. Customer clicks on “submit” button. 4. The system verifies the validity of the input form. 5. The system checks if the request overlaps with other active requests or reservations. 6. The system sends a confirmation to the customer. 7. The system stars searching for a taxi driver available to fulfill the request. 8. The system finds such a taxi driver and sends him the request. 9. The taxi driver confirms his intention to take care of the request, his status is set to “Busy”. 10. The system sends the waiting time and the taxi code to the user.
Output conditions	The customer successfully receives the taxi ride he asked for.
Exceptions	1. Customer does not fill correctly the request form. 2. Customer’s request overlaps with others active requests or reservations.

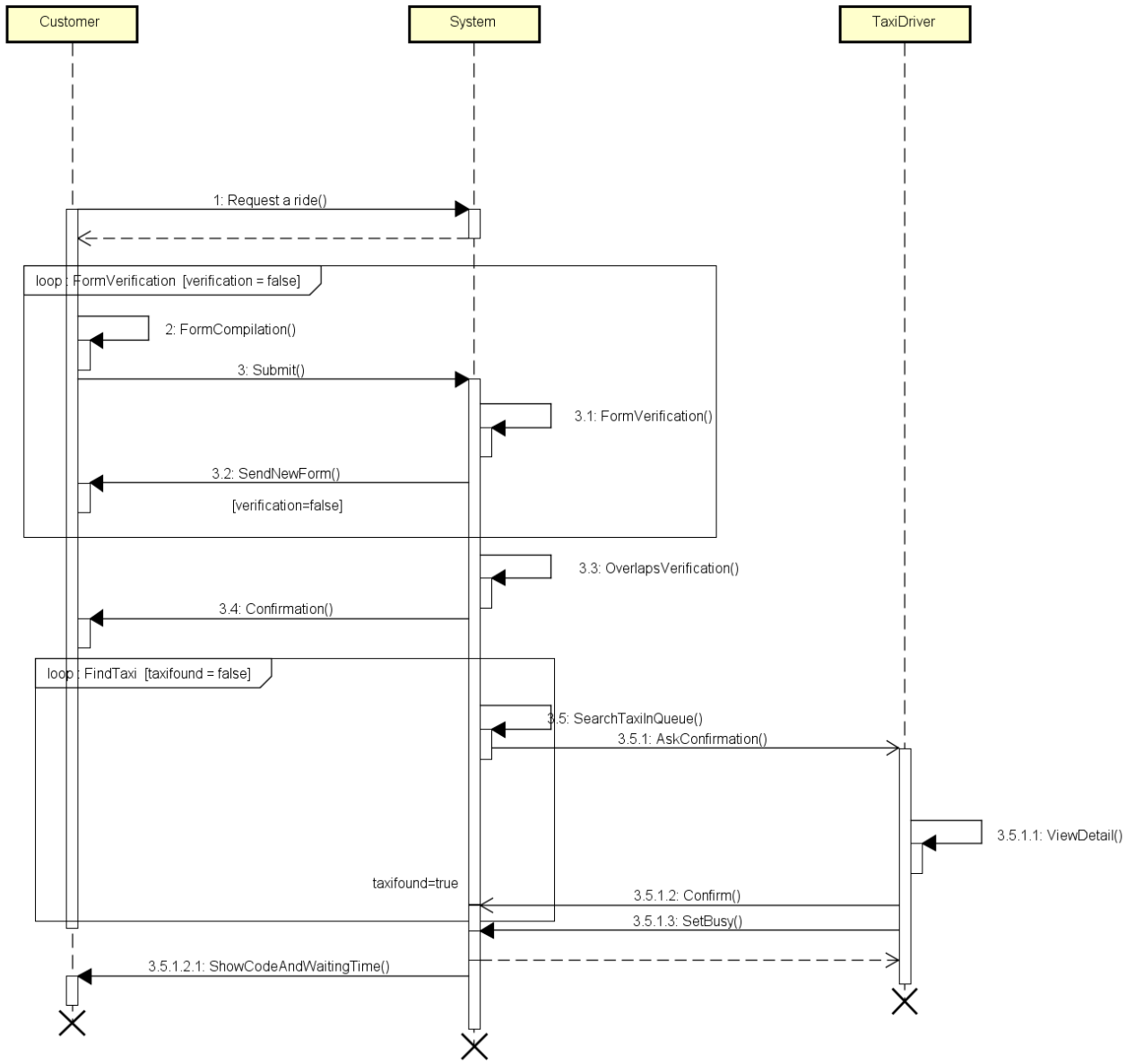
Reserve a ride table:

Name	Reserve a ride
Actors	Customer, Taxi Driver
Entry conditions	Customer and Taxi Driver are registered and logged into the system.
Event flow	1. Customer clicks on “reserve a ride” button on his user page. 2. Customer fills the input form. All the fields are mandatory. 3. Customer clicks on “submit” button. 4. The system verifies the validity of the input form 5. The system sends a confirmation to the customer. 6. 10 minutes before the established time the system checks for overlaps with other active requests or reservations. 7. The system starts searching for a taxi available to take care of the customer. 8. The system finds a taxi and asks his confirmation. 9. The taxi driver confirms his intention to take care of the customer. His status is set to “Busy”. 10. The system sends the taxi code to the customer.
Output conditions	The customer successfully receives the taxi ride he reserved.
Exceptions	1. Customer does not fill correctly the reservation form. 2. Customer reservation overlaps with other active requests or reservations.

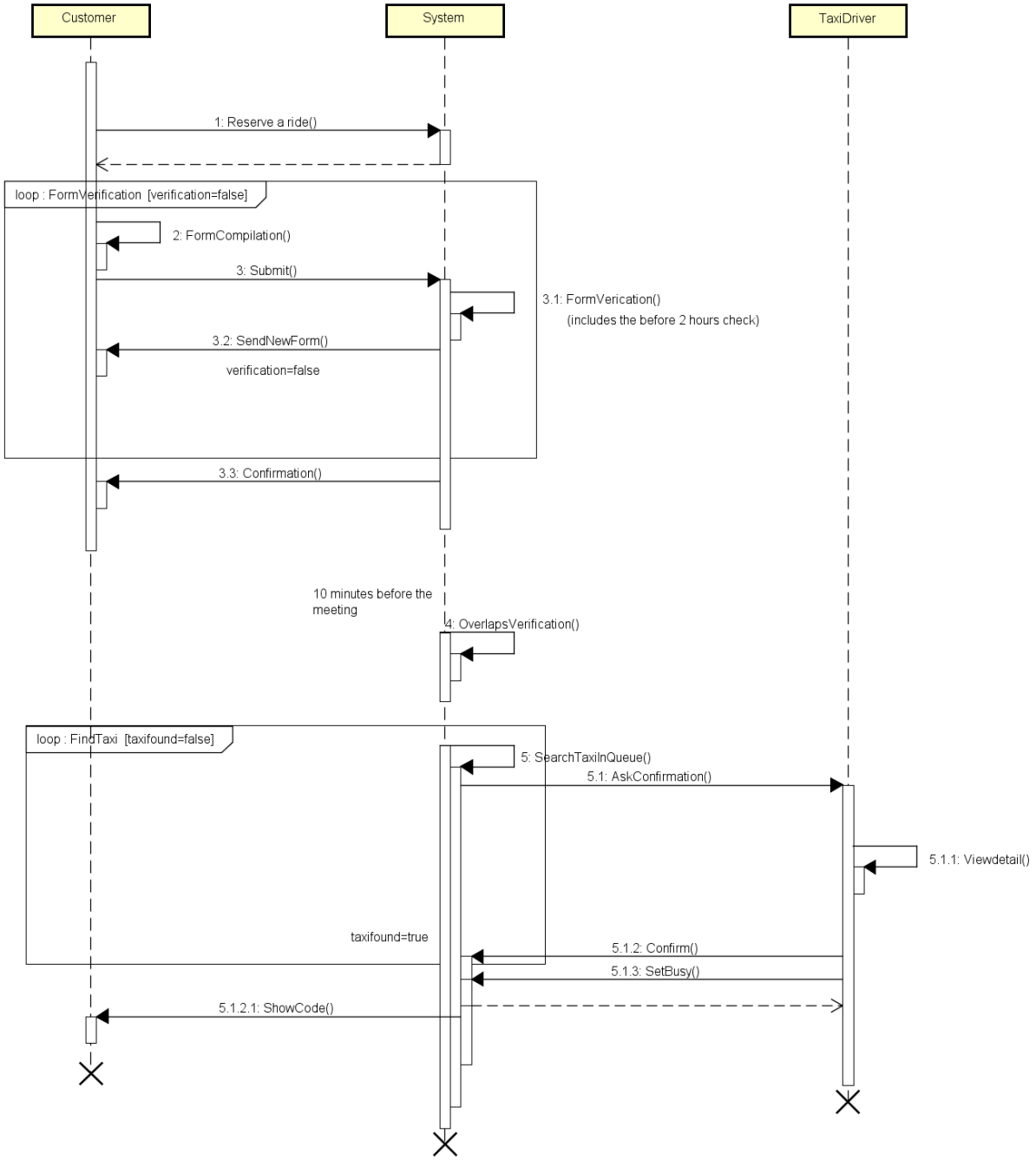
The *Request* and *reserve a ride* use case diagrams are presented together because of their similarities.



Request a ride sequence diagram.

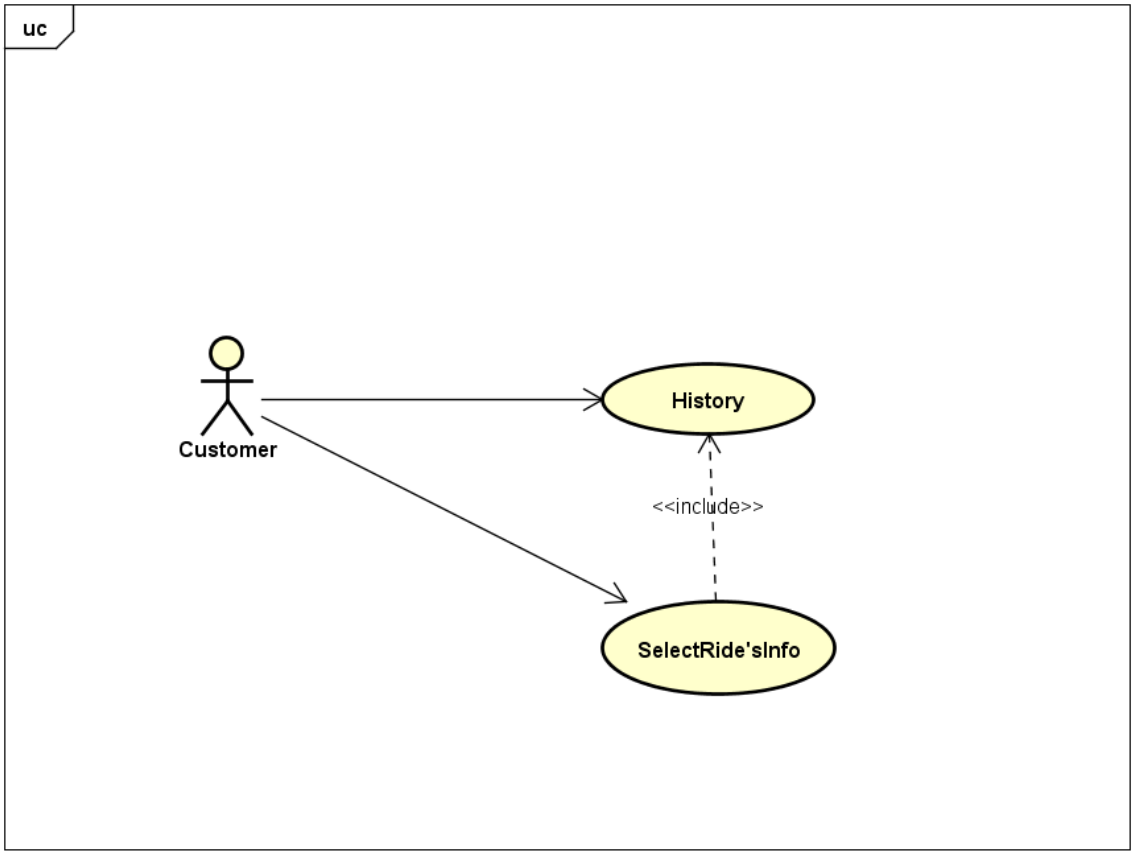


Reserve a ride sequence diagram.



3.6.1.4. History

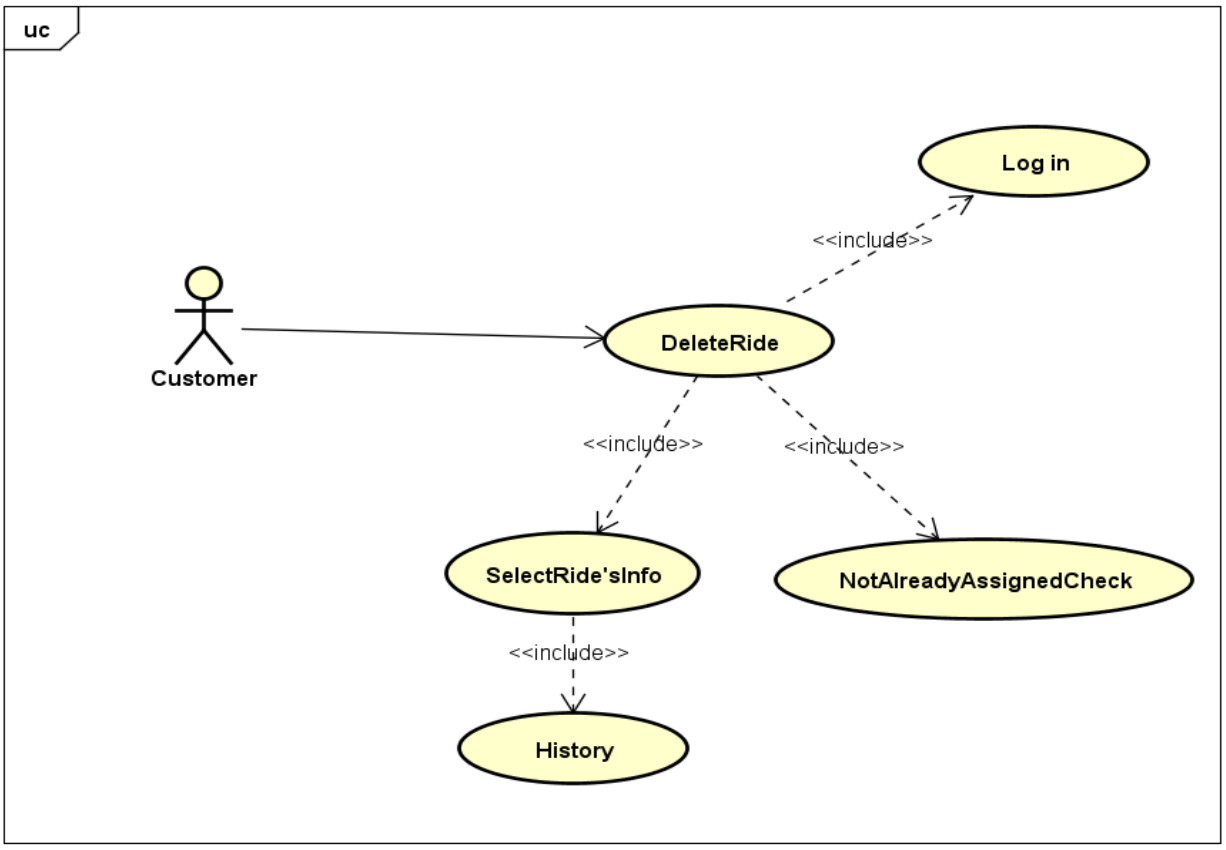
Name	History
Actors	Customer
Entry Conditions	Customer already registered and logged into the application.
Event flow	1. Customer clicks on “history” button. 2. Application shows the history of personal rides. 3. Customer may want to select a specific ride and see the information related.
Output conditions	The application allow the customer to see his personal history of taxi rides.
Exceptions	Null



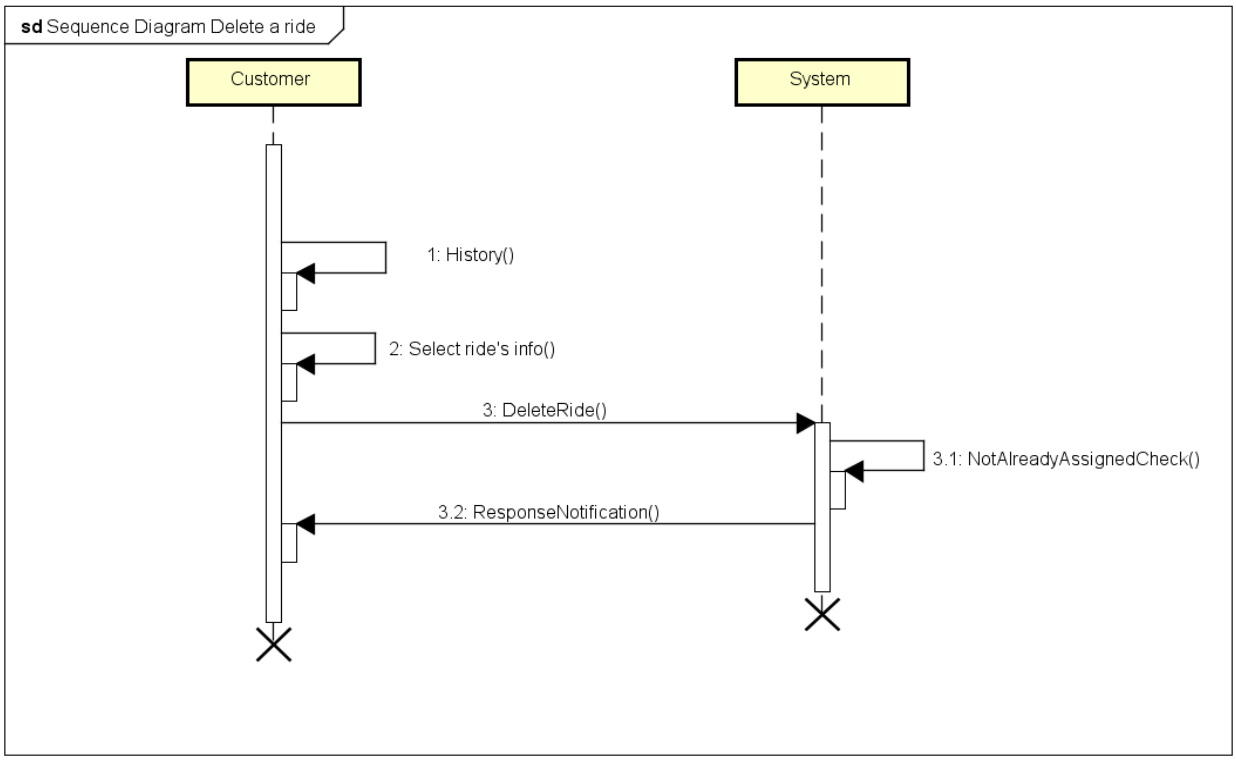
powered by Astah

3.6.1.5. Delete a ride

Name	Delete a ride
Actors	Customer
Entry conditions	1. Customer is already registered and logged into the application. 2. There is an actual request or reservation to delete.
Event flow	1. Customer clicks on “History” button on his user page. 2. Customer selects a request or a reservation from the list of his unaccomplished taxi rides. 3. Customer clicks on “info” button. 4. Customer clicks on “delete” button. 5. System checks that the customer is allowed to delete the ride. 6. The system sends a notification to communicate either the success or fail of the operation.
Output conditions	The customer successfully delete a request or reservation previously made, there is no need to fullfill it anymore.
Exceptions	A taxi has already been assigned to the request/reservation, the customer cannot delete it.



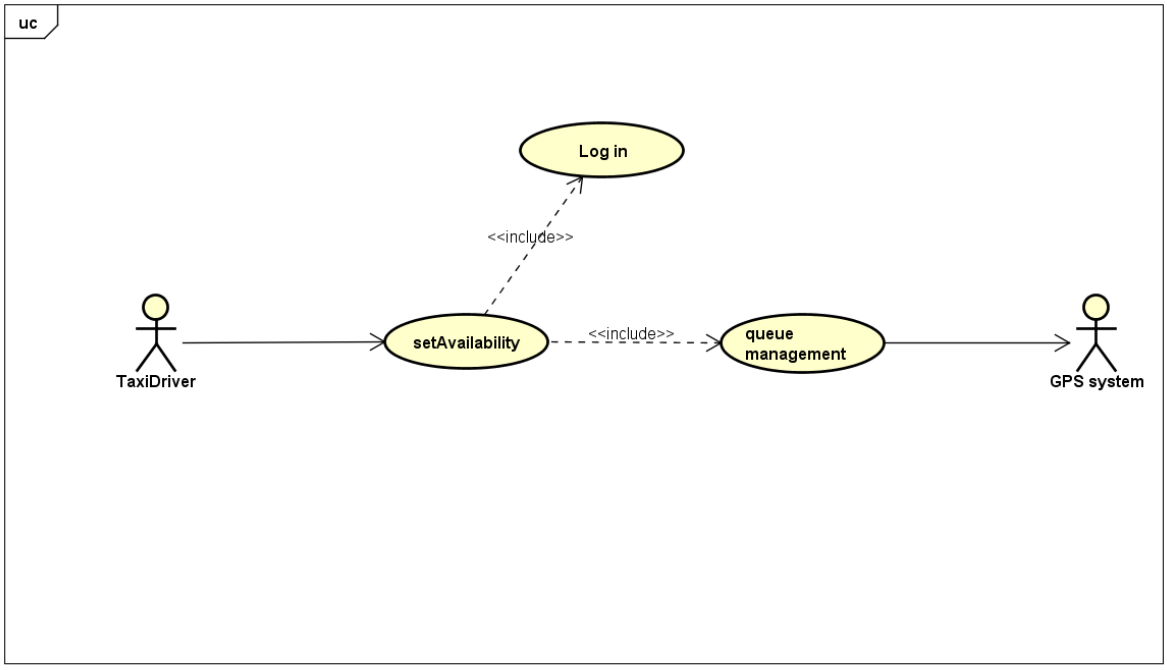
powered by Astah

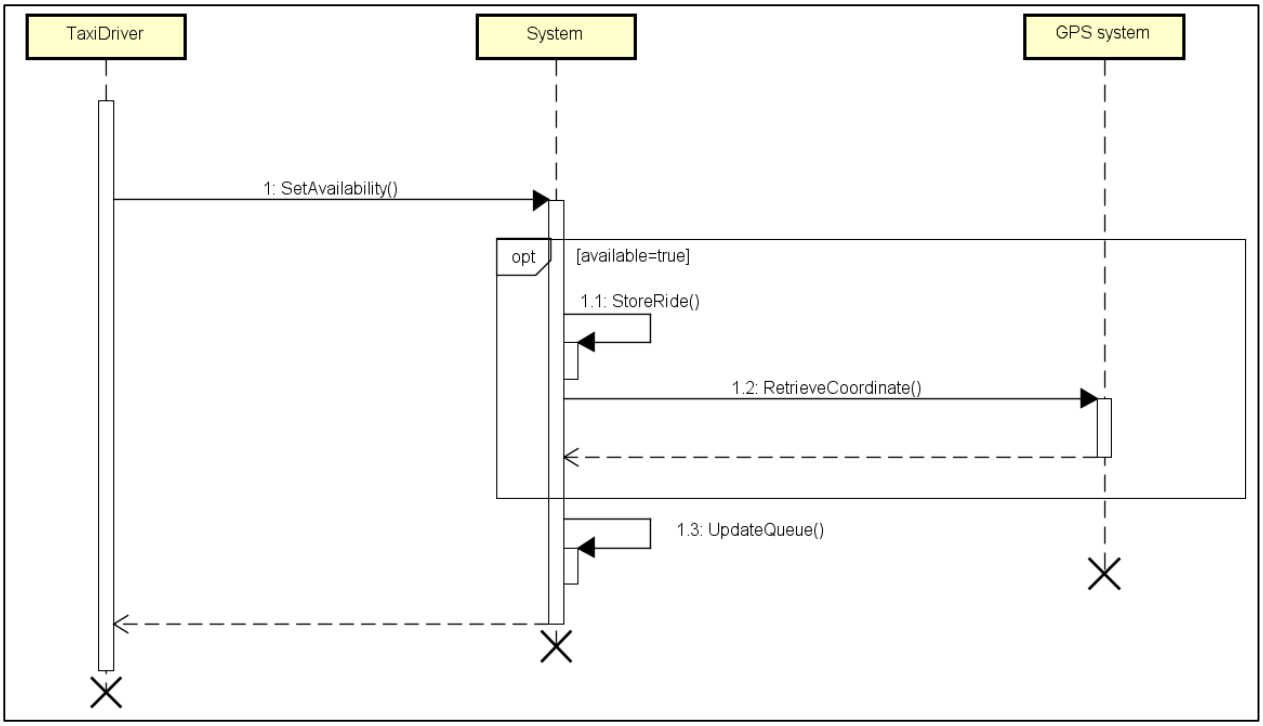


powered by Astah

3.6.1.6. Taxi availability

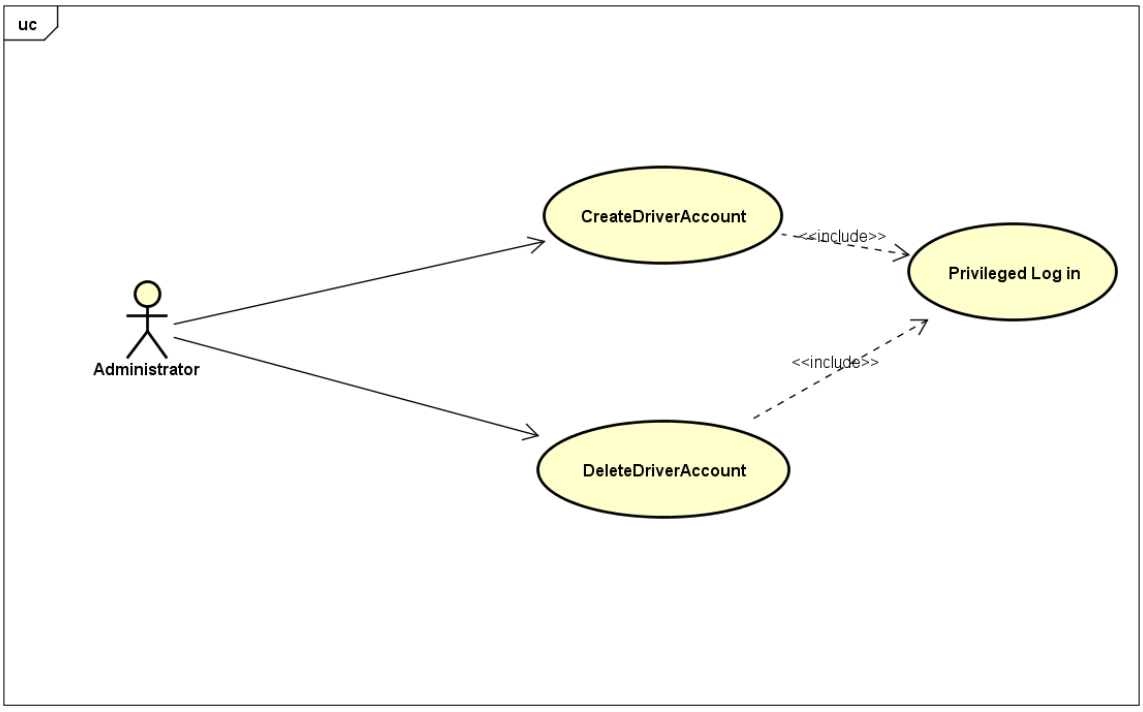
Name	Set availability
Actors	Taxi Driver
Entry conditions	1. The taxi driver is already registered and logged into the application. 2. His status is either “Available” or “Busy”.
Event flow	1. Taxi driver clicks either on “set available” or “set busy” button via his mobile application. 2. If the new status is “available” the system checks for any ride associated to the taxi and stores it among the completed rides. 3. If the new status is “available”, the system retrieves the taxi location via the GPS system. 4. The system updates a taxi queue either removing or inserting the taxi, depending on the new status selected.
Output conditions	The taxi driver correctly change his status and the system reacts according to it by updating his taxi queues.
Exceptions	Null

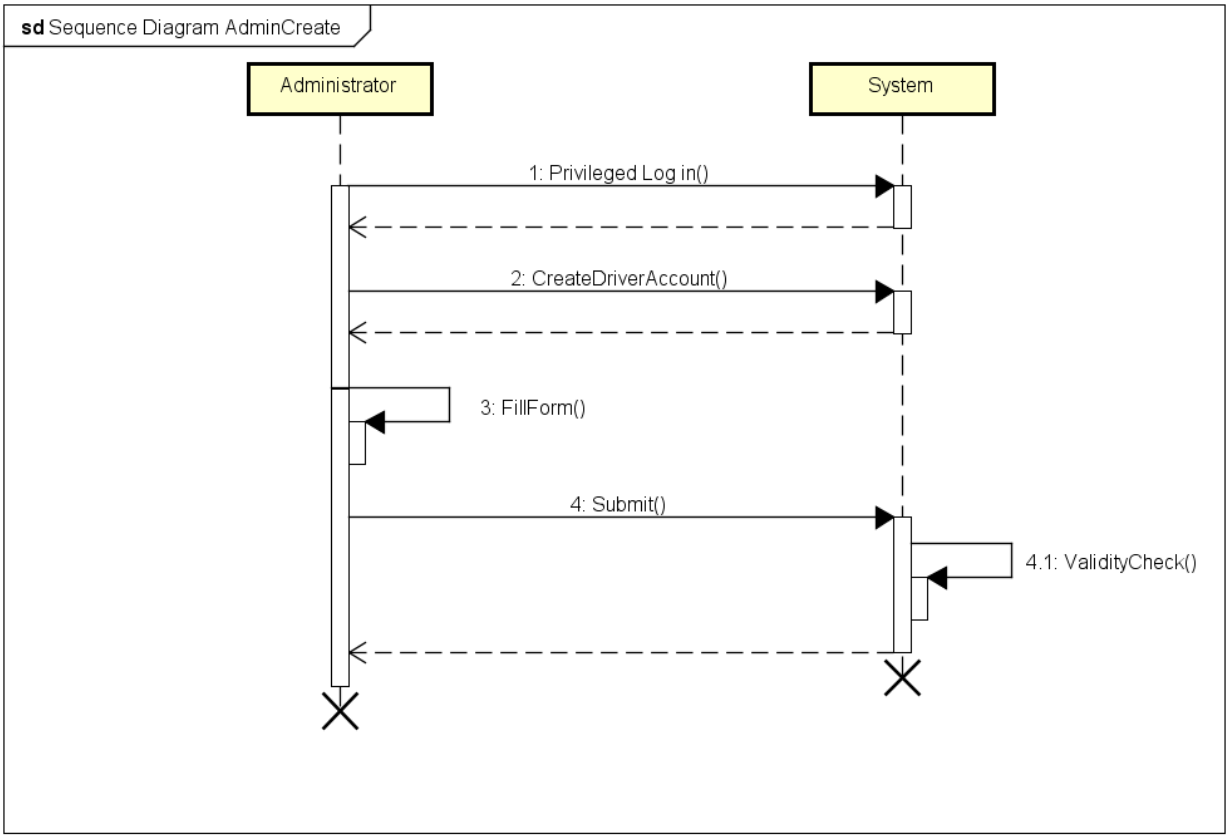



TaxiDriver
System
GPS system
1: SetAvailability()
opt
[available=true]
1.1: StoreRide()
1.2: RetrieveCoordinate()
1.3: UpdateQueue()

3.6.1.7. Create/Delete drivers.

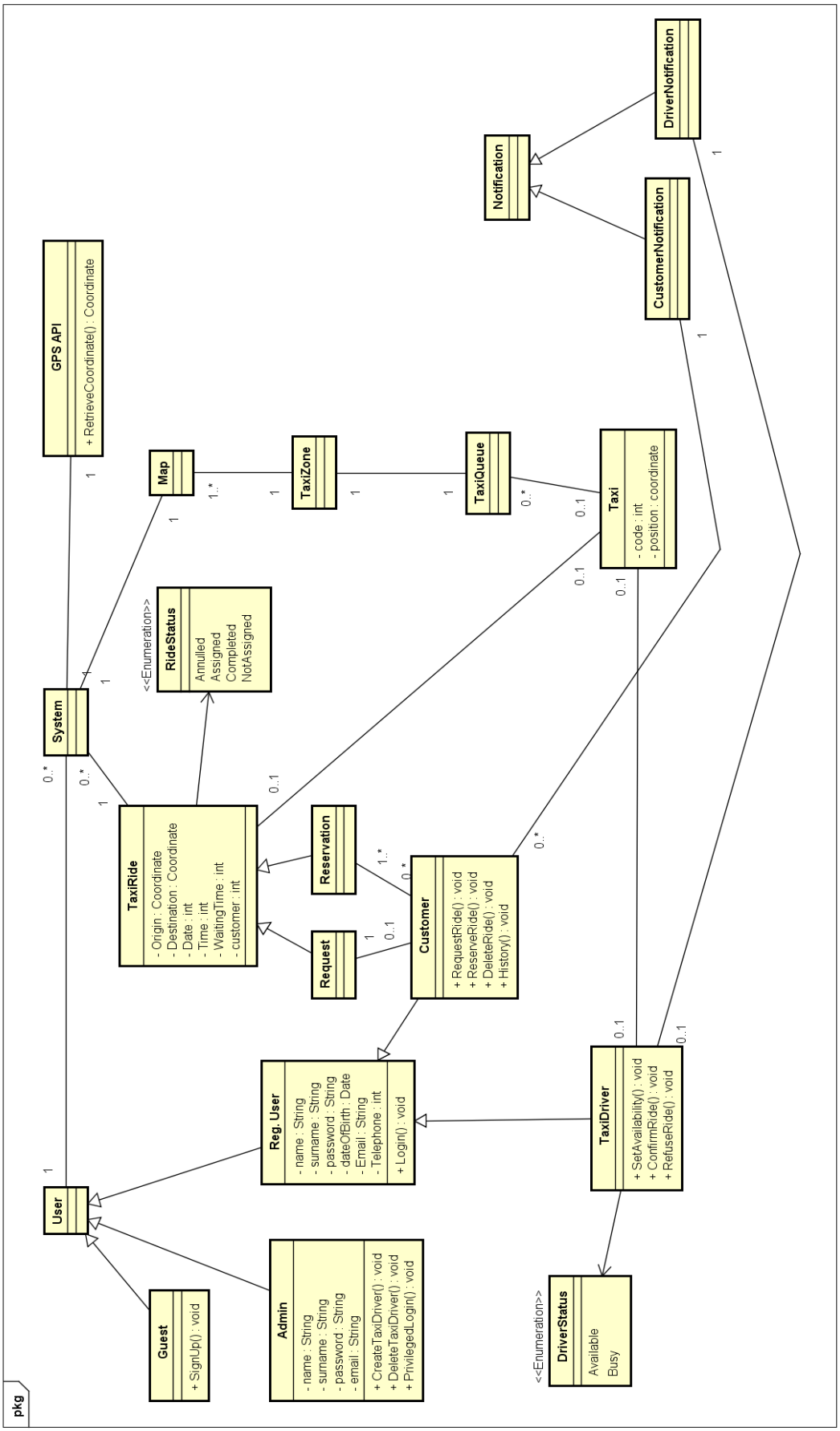
Name	Create/Delete taxi driver's account.
Actors	Administrator
Entry conditions	1. If "delete", there must be an account to delete. 2. The administrator must be logged into the system.
Event flow	1. Administrator clicks on "create account" / "delete account" button. 2. The administrator fills a form with data about the account to be created/deleted. 3. Administrator clicks on "submit" button. 4. The system checks the validity of the operation. 5. The system creates/delete the specified account.
Output conditions	Taxi account created/deleted.
Exceptions	1. Administrator does not fill correctly the form. 2. No such an account to be deleted exists. 3. There is already the account to be created.





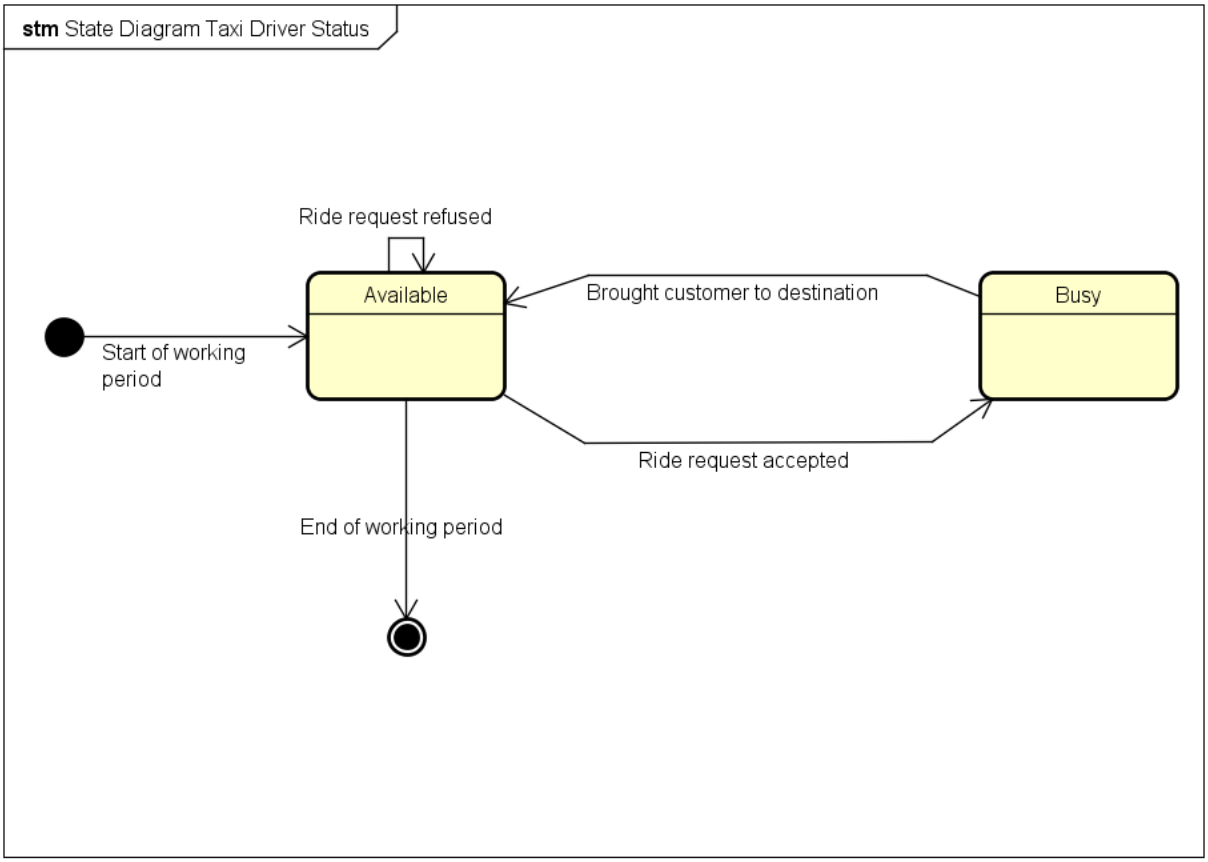
powered by Astah

3.6.2. Class Diagram

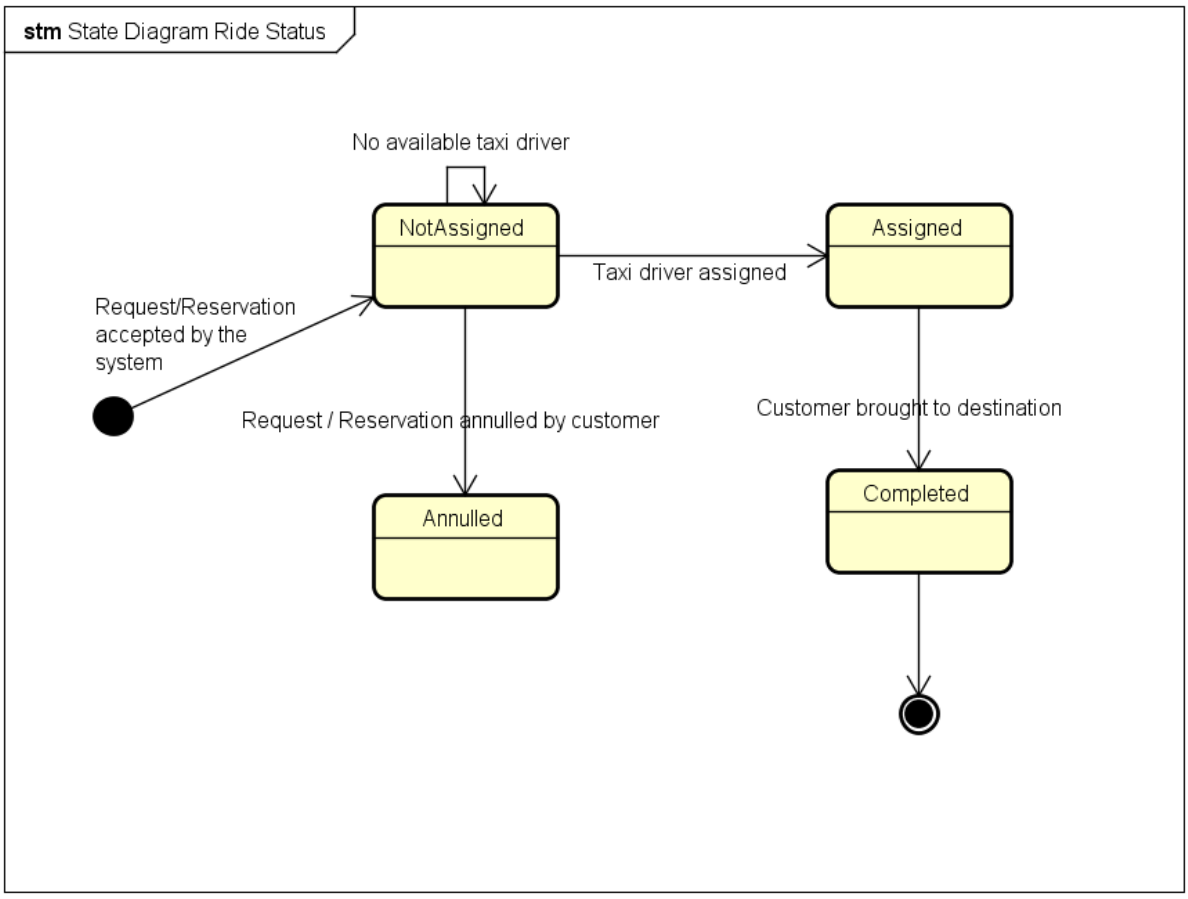


3.6.3. State diagram

The aim of this class diagram is to clarify the two Taxi Driver's status: *Busy* and *Available*.



This diagram is intended to elucidate the four possible status of a *ride* (either *request* or *reservation*).



4. Appendix

4.1. Alloy

4.1.1. Signatures

```
//*****SIGNATURE*****//

sig Integer {}
sig Strings {}
sig Date {}
sig Time {}
sig Coordinate{}

enum DriverStatus {Busy, Available}

enum RideStatus{Annulled, Assigned, Completed, NotAssigned}

abstract sig User {}

sig Guest extends User {}

sig Admin extends User{
  name: one Strings,
  surname: one Strings,
  email: one Strings,
  password: one Strings
}

abstract sig RegUser extends User{
  name: one Strings,
  surname: one Strings,
  password: one Strings,
  birth: one Date,
  email: one Strings,
  telephone: one Strings
}

sig Customer extends RegUser{}

sig TaxiDriver extends RegUser{
  status: one DriverStatus,
  taxi: one Taxi
}
```

```
sig Taxi {
  code: one Integer,
  position: one Coordinate
}

sig TaxiQueue {
  hasTaxi: set Taxi
}

sig TaxiZone {
  hasTaxiQueue: one TaxiQueue
}

one sig Map {
  hasZone: some TaxiZone
}

one sig System {
  map: one Map,
  users: set User,
  taxiRide: set TaxiRide
}

abstract sig TaxiRide {
  origin: one Coordinate,
  destination: lone Coordinate,
  date: one Date,
  time: one Time,
  waitingTime: lone Integer,
  taxi: lone Taxi,
  rideStatus: one RideStatus,
  hasCustomer: one Customer
}

sig Request extends TaxiRide {

}

sig Reservation extends TaxiRide {

}
```

4.1.2. Facts

```
//*****FACTS*****//

//No users have the same email and telephone number (i.e. there are no duplicate users)
fact noDuplicateUsers{
  no disj u1,u2: RegUser| (u1.email = u2.email) or (u1.telephone = u2.telephone)
  no disj a1,a2:Admin | (a1.email = a2.email)
  no disj a1:Admin, u1:RegUser | (a1.email = u1.email)
}

//Every taxi has always exactly one driver
fact taxiOneDriver{
  no disj d1,d2:TaxiDriver | d1.taxi = d2.taxi
  #TaxiDriver = # Taxi
}

//Every taxi queue has always exactly one taxi zone
fact queueOneZone{
  TaxiZone <: hasTaxiQueue in TaxiZone one -> TaxiQueue
}

//Every taxi can be in only one queue
fact taxiOneQueue{
 all t1:Taxi | no disj q1,q2:TaxiQueue | (t1 in q1.hasTaxi and t1 in q2.hasTaxi)
 all t2:Taxi | lone q3:TaxiQueue | t2 in q3.hasTaxi
}


//The map has all the taxi zones
fact taxiZoneInMap{
  TaxiZone in Map.hasZone
}


//No more than 1 "Assigned" taxiRide for customer and
//no more than 1 request for costumer
fact RideLimit{
  //No 2 rides Assigned
  no disj r1,r2:TaxiRide | (r1.rideStatus = r2.rideStatus) and
                           (r2.rideStatus = Assigned) and
                           (r1.hasCustomer = r2.hasCustomer)
  //No 2 requests NotAssigned
  no disj r1,r2:Request | (r1.rideStatus = r2.rideStatus) and
                          (r2.rideStatus = NotAssigned) and
                          (r1.hasCustomer = r2.hasCustomer)
  //No 1 Assigned request and 1 NotAssigned request
  no disj r1,r2:Request | (r1.rideStatus = Assigned) and
                          (r2.rideStatus = NotAssigned) and
                          (r1.hasCustomer = r2.hasCustomer)
//No 1 NotAssigned request and 1 Assigned reservation
  no disj r1:Request, r2:Reservation| (r1.rideStatus = NotAssigned) and
                                      (r2.rideStatus = Assigned) and
                                      (r1.hasCustomer = r2.hasCustomer)
}
```

```
//Taxis (or taxi drivers) have no more than 1 taxiRide with assigned status
fact driverOneRide{
  no disj r1,r2:TaxiRide | (r1.taxi = r2.taxi) and (r1.rideStatus = Assigned) and (r2.rideStatus = r1.ride
}

//There are no taxi paired with a taxi ride with "not assigned" status
fact noTaxiNotAssigned{
  all r1:TaxiRide | (r1.rideStatus = NotAssigned) implies (r1.taxi = none)
}

//Assigned and completed rides must be bound to a taxi
fact TaxiRideStatus{
    all r1:TaxiRide | (r1.rideStatus != NotAssigned and r1.rideStatus!= Annulled) implies (#r1.taxi=1)
}

//Systems must have the reference to all users and rides
fact systemUserRide{
  all u1:User | u1 in System.users
  all r1:TaxiRide | r1 in System.taxiRide
}

//Annulled rides must not link to any taxi
fact AnnulledNoTaxi{
  all r:TaxiRide | r.rideStatus = Annulled implies r.taxi = none
}

//Busy taxi must not be in a queue
fact taxiBusyQueue{
 all d1:TaxiDriver | ((d1.status = Busy) implies
                        (no q1:TaxiQueue | d1.taxi  in  q1.hasTaxi))
}

//Available taxi must be in a queue
fact taxiAvailableQueue{
  all d1:TaxiDriver | ((d1.status = Available) implies
                        (some q1:TaxiQueue | d1.taxi  in  q1.hasTaxi))
}

//Origin cannot be equal to Destination
fact originDifferentDestination{
  no r1:TaxiRide | (r1.origin = r1.destination)
}

//Assigned taxi ride implies Busy taxi driver
fact assignedEqualBusy{
  no t:Taxi |  taxi.t.rideStatus in Assigned  and taxi.t.status=Available
}

//Available taxidriver implies NotAssigned  taxiRide
fact availableEqualnotAssigned{
 all t:Taxi,d:TaxiDriver | (d.taxi = t and d.status = Available) implies
                            (no r:TaxiRide | r.taxi = t and r.rideStatus = Assigned)
}
```

4.1.3. Predicates

These predicates have been used with some assertions and in the *show()* predicate in order to generate some specific worlds.

```
//*****PREDICATES******
//Add an assigned ride
pred addAssignedRide(s1,s2:System){
  one r1:TaxiRide | r1.rideStatus = Assigned  and s2.taxiRide=s1.taxiRide + r1
}

//Add an annulled ride
pred addAnnulledRide(s1,s2:System){
  one r1:TaxiRide | r1.rideStatus = Annulled  and s2.taxiRide=s1.taxiRide + r1
}

//Add a completed ride
pred addCompletedRide(s1,s2:System){
   one r1:TaxiRide | r1.rideStatus = Completed  and s2.taxiRide=s1.taxiRide + r1
}

//Add a completed ride
pred addCompletedRide(s1,s2:System){
   one r1:TaxiRide | r1.rideStatus = Completed  and s2.taxiRide=s1.taxiRide + r1
}

//Add 2 taxi drivers
pred add2Driver (s1,s2:System, d1,d2:TaxiDriver){
  s2.users= s1.users + d1 + d2
}
```

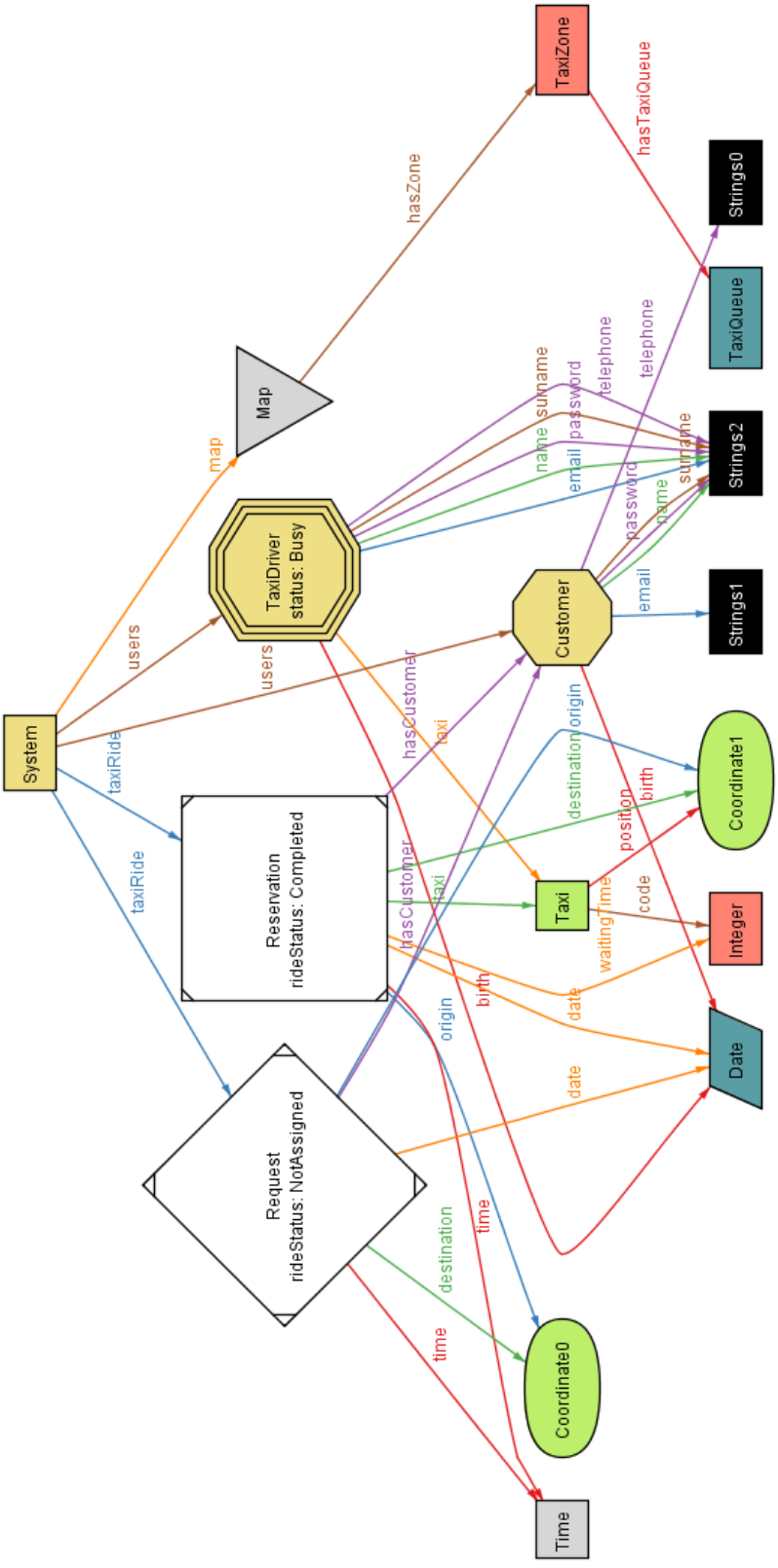
```
//Add 2 customers
pred add2Customer(s1,s2:System,c1,c2:Customer){
 s2.users=s1.users + c1+c2
}
//Add 2 reservations
pred add2Reservation(s1,s2:System, res1,res2:Reservation){
 s2.taxiRide=s1.taxiRide + res1 + res2
}

//Add 2 requests
pred add2Request(s1,s2:System, req1,req2:Request){
 s2.taxiRide=s1.taxiRide + req1 + req2
}

//Add 1 busy and 1 available taxi driver
pred atleast1busy1available{
   some d:TaxiDriver | d.status = Busy
   some d:TaxiDriver | d.status = Available
}

//Add 1 busy driver without a ride (standard customer)
pred addBusyWithoutRide(s1,s2:System){
   one d:TaxiDriver | d.status = Busy and no r:TaxiRide | r.taxi = d.taxi   and s2.users = s1.users + d

}
```

4.1.4. Assertions

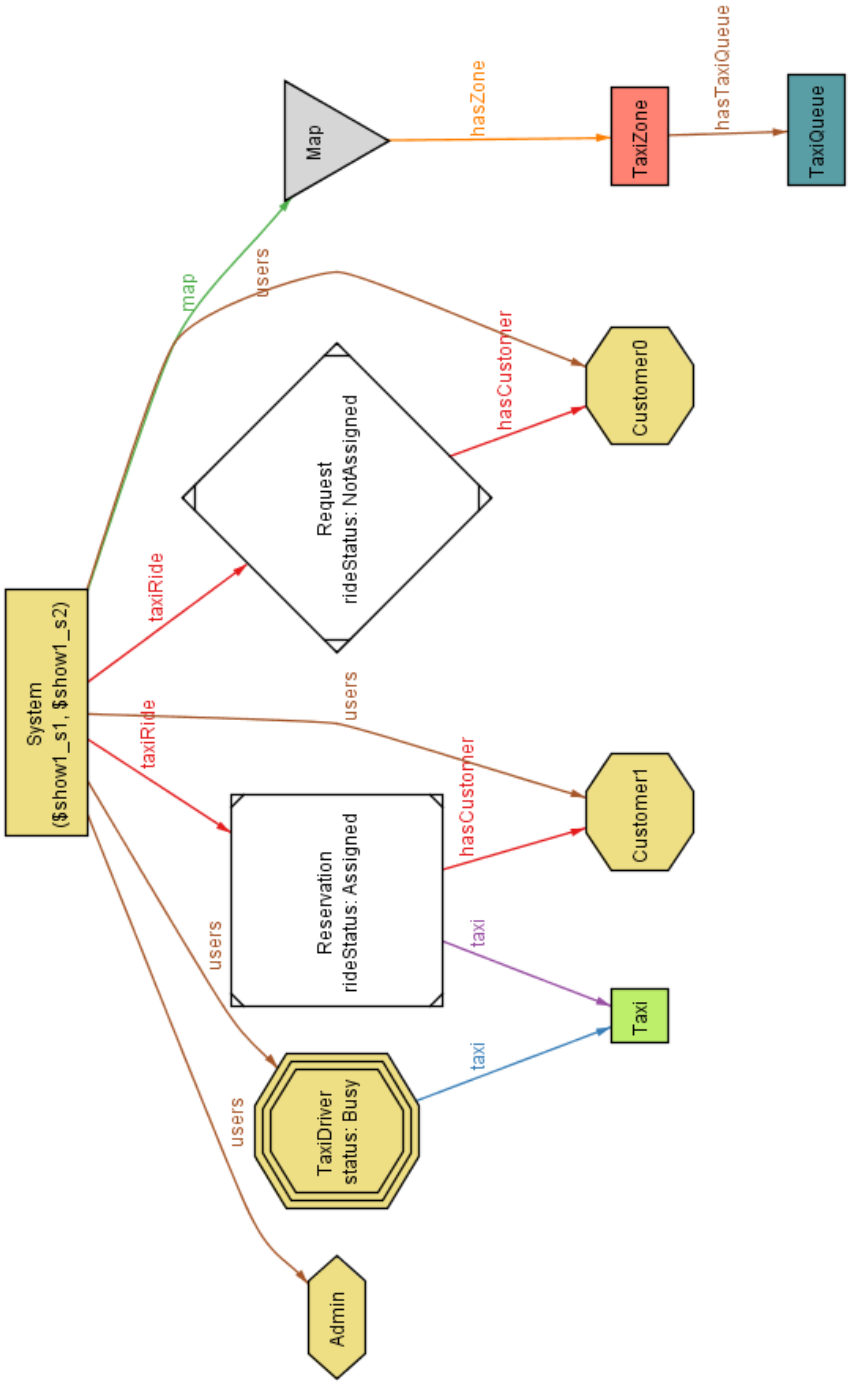
```
//*****ASSERTIONS*****

//All these cardinality equivalences must be satisfied
assert numbersEquivalence{
  #Taxi = #TaxiDriver
  #TaxiZone = #TaxiQueue
  #System.users = # User
  #System.taxiRide = #TaxiRide

  //The num. of available taxi must be equal to the num. of taxi in all the queues:
 # (status.Available).taxi = #TaxiQueue.hasTaxi
}

//Busy taxi drivers may not be assigned to a ride, because of the possibility of standard customers
assert driversStandardCustomer{
    all s1,s2:System | addBusyWithoutRide[s1,s2] implies
                        !(status.Busy  in (TaxiDriver <: taxi).(TaxiRide.(TaxiRide <:taxi)))
}

//All available taxi belong to a taxi zone
assert allAvailableInOneZone{
  all t:Taxi | ((TaxiDriver <: taxi).t).status = Available implies
                                        one z:TaxiZone | t in z.hasTaxiQueue.hasTaxi
}

//There are no taxi without drivers
assert noTaxiWithoutDrivers{
  all t:Taxi | one d:TaxiDriver | d.taxi=t
}

//There are no drivers without taxi
assert noDriversWithoutTaxi{
  all d:TaxiDriver | one t:Taxi | d.taxi=t
}

// no ride without customers
assert noRideWithoutCustomer{
  all r:TaxiRide | one c:Customer| r.hasCustomer = c
}
```

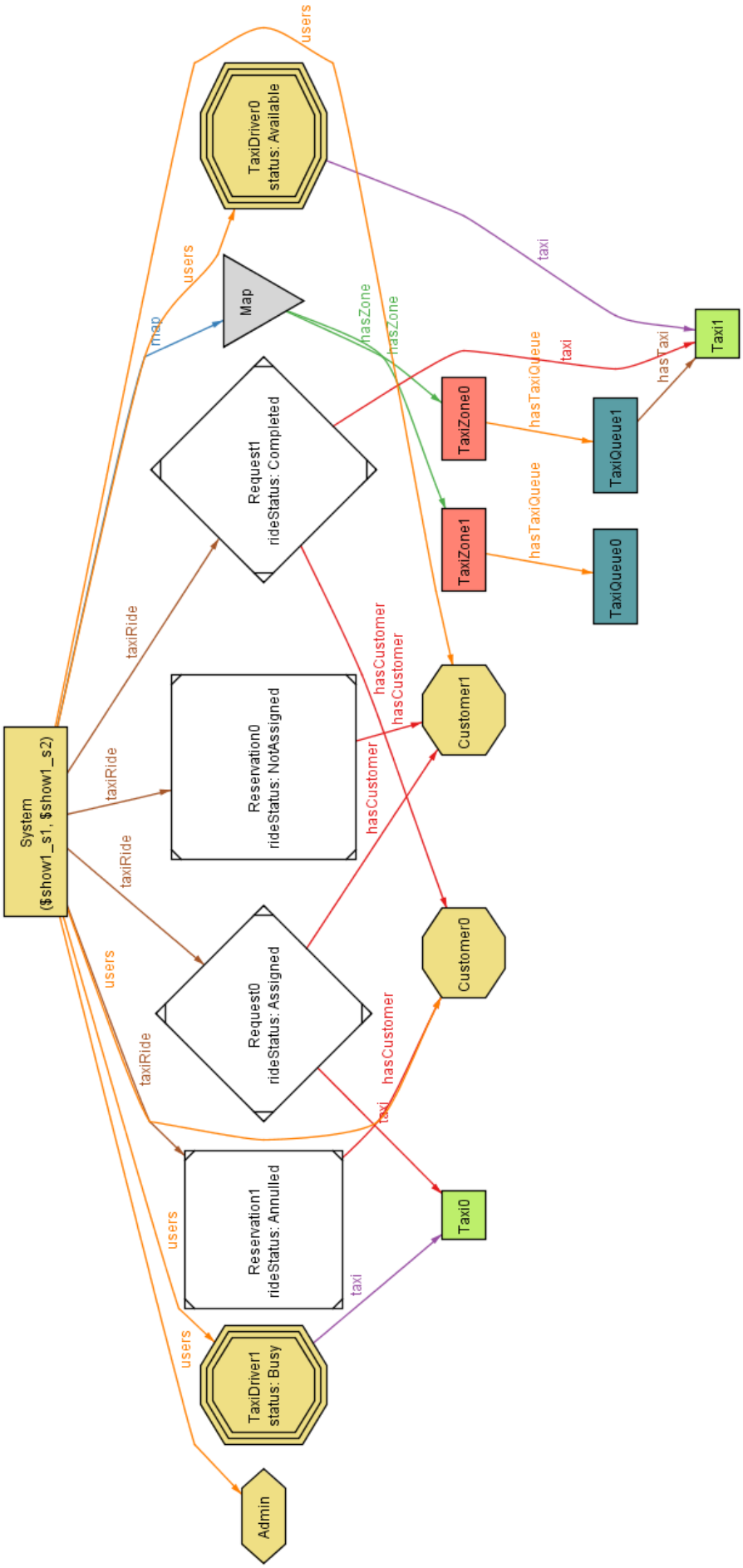
4.1.5. Generated world

This is a possible world with two taxi rides, a single taxi driver and a customer. Note that the entity String has the main purpose to distinguish between user's email and telephone. In fact, it is possible to notice that different users have different Strings as range of the relation "email" and "telephone".



In the following alternative worlds, Strings and other non-essential signatures have been removed in order to clarify the relation between the main entities.




System
($show1_s1, $show1_s2)
Admin
TaxiDriver1
status: Busy
TaxiDriver0
status: Available
Map
Request1
rideStatus: Completed
Request0
rideStatus: Assigned
Reservation0
rideStatus: NotAssigned
Reservation1
rideStatus: Annulled
TaxiZone0
TaxiZone1
TaxiQueue1
TaxiQueue0
Taxi1
Taxi0
Customer1
Customer0
users
map
taxiRide
hasZone
hasTaxiQueue
taxi
hasTaxi
hasCustomer

4.1.6. Comments on Alloy

Reasoning with Alloy has been useful in order to exploit weakness and mistakes in the Class Diagram and in the general behavior of the system, although its analyzer's limited range have sometimes caused misunderstanding about the correctness of the model.

4.2. Software and tools used

- Microsoft Word (https://products.office.com/it-it/word) to redact and to format this document.
- Astah Professional (http://astah.net/) to create Use Cases Diagrams, Sequence Diagrams, Class Diagrams and State Diagrams.
- GitHub (https://github.com) to share the working material of this project.
- Balsamiq Mockups (https://balsamiq.com/products/mockups): to create mockups.
- Alloy Analyzer (http://alloy.mit.edu/alloy/): to prove the consistency of our model.

4.3. Hours of work

Alessandro Pozzi: ~35 hours

Marco Romani: ~35 hours

4.4. Revision

- Removed requirement [R3] in goal [G3]
- World and the machine analysis updated
- Taxi allocation description changed in The world and the machine
- Added use case and table for “History” function