MyTaxiService

Pozzi Matteo
Scandroglio Stefano

INDEX

1 Introduction

1.1 Purpose

This document represents the Requirement Analysis and Specification Document (RASD). The main goal of this document is to describe the system in terms of functional and non functional requirements, identify its goals and show some typical case of use that will occur after the release of the system.

1.2 Scope

The goal of the project is to develop the myTaxiService application, both in terms of mobile application and web application. Once developed, the application will allow users to request a taxi to pick them up as soon as possible, or to reserve a taxi for the future, providing information about the pick up place and the destination of the trip.
The reservation of a taxi has to occur at least two hours before the desired pick up time and may be cancelled or modified up to one hour before the requested time.
The systems will manage different queues of taxis, each one associated to a certain zone of the city, by forwarding requests that come from that zone to the first taxi in the queue of that zone.
Taxi drivers will also use this application to either accept or refuse requests forwarded to them. In case the taxi driver refuses the request, he/she will be moved at the end of the queue.

1.3 Glossary

Brief description of the terms used in the document:

- **User:** a registered person who is able to requests and reserve a taxi
- **Driver:** the person driving the taxi
- **Request:** when a user asks to be picked up by a taxi as soon as possible
- **Reservation:** when a user asks to be picked up in a time and in a location he/she specifies
- **Served request**: a request is served when the user has been taken to the destination
- **Zone:** a 2 km^2 part of the city
- **Queue:** first in first out (FIFO) structure containing available drivers of a certain zone

- **Pick up time:** exact time at which the user wants to be picked up by the taxi driver
- **Pick up location:** location from which the user will be picked up by the taxi driver
- **Destination:** final location to which the taxi driver takes the user
- **Notification:** either a message informing the user about code of the requested/reserved taxi and corresponding waiting time, or a message informing the driver about a new request
- **Available driver:** a driver to which requests can be forwarded
- **Unavailable driver:** a driver who can't receive requests

1.4 Goals

In particular, the services that will be provided are:

(G1) Ability for a guest to register and login

(G2) Ability for a user to request or reserve a taxi

(G3) Ability for a user to either edit or delete a reservation

(G4) Ability for a user to track the taxi and check its code and its estimated time of arrival

(G5) Ability for a driver to log in to his driver account

(G6) Ability for a driver to set his/her availability on his/her account

(G7) Ability for a driver to either accept or refuse a request

(G8) Notifications both to users and drivers

1.5 Domain assumptions

We suppose that the following conditions hold in the analyzed world:

- The email address provided by a user during the registration phase must be valid
- In the sign up phase email address, name, surname, password are mandatory
- If a user requests a taxi, he/she wants to be picked up as soon as possible

- Users will actually get on the taxi they have requested
- Each request/reservation can be fulfilled with the use of just one car
- Users will actually be present in the place from which they want to be picked up
- Once a user makes a request for a taxi, he/she will not request any other taxi until the request has been served
- Users are always logged in from when they made a request until the driver come to the pick up location, so that they can receive notifications and can check taxi informations
- Each driver owns a driver account in the system given by the government
- Only the government is able to create driver accounts
- Once a driver accepts a request he/she will actually pick up the user that has generated that request
- Once a driver declines a request he/she will not pick up the user that has generated that request
- Each available driver is always logged in and will always answer an incoming request
- Payments will happen only when drivers leave their passengers

1.6 Stakeholders

The real stakeholder is our professor who wants to verify if we can properly create every document that is required to properly describe a software application, from its requirement to its design.

In a business context our stakeholders would be the government of the city that wants to release this system, the taxi company and its drivers, who would use this application to interact with their clients, and clients themselves, who will use this system to request and reserve taxis.

1.7 Overview

In the rest of the document we will provide a list of the actors involved in the application, detailed requirements and some possible scenarios of use.
We will then provide some UML diagrams (Use Case, Sequence Diagram, Class Diagram) in order to provide further details on the structure and behaviour of the application.

2 Identifying Actors

We identified 4 actors:

1. **Guest:** a person who is only able to register to the service
2. **User:** a person who can request and reserve taxis
3. **Driver:** a person who drives a taxi and is in charge of picking up users
4. **Government:** the entity who is in charge of distributing driver accounts

3 Specific Requirements

According to the goals and the domain assumptions we can derive the following requirements:

(R1) Registration and login:

The system has to provide sign-in and sign-up functionalities.
The registration step will be required only for normal guests, while taxi drivers will already own an account with a specific username and password provided by the government of the city.

(R2) Requests/reservation management:

The system will allow users to request a taxi to pick them up as soon as possible.
The system will also allow user to reserve a taxi for the future.
Once a request/reservation has been accepted by a driver, the user will be able to see its details (code of the taxi and estimated time of arrival), edit or delete it in specific sections of the web app and of the mobile application.

The reservation must happen at least two hours before the desired pick up time and can be edited or deleted up to 1 hour before the pick up time.

(R3) Queue management:

The system keeps track of the available drivers in a certain zone in a queue structure.
The system sends notifications for requests originating from a certain zone to the first taxi driver in the queue of that zone.
When the system sends a notification to a driver, it deletes him/her from the queue.
Inside the notification the driver will see the details of the request (i.e. pick up place).
If the chosen driver accepts that request, then the system will send the needed informations to the user, mark the driver as unavailable until that request has been served and move to the next request.
If the driver refuses the request, the system will forward the request to the new first driver in the queue, remove the driver who refused the request and then add him/her again to the queue of that zone(so that now he/she is in the last position of the queue).
Drivers are inserted in the queue only if they are available and they are added in the queue of each zone according to the order in which they entered in that zone.
When a driver moves from one zone to another, the system will add him/her to the queue related to that zone.

(R4) Driver availability management:

In a specific section of their account, drivers can specify their availability, so that the system will know which drivers to insert in queue.

(R5) Driver request/reservation management:

When a user send a request for a taxi, the system will notify the taxi driver,while reservations are notificated 10 minutes before the pick up time. Once the driver has received the notification, he/she has the opportunity to accept it or refuse it, directly from the application.

(R6) Driver account distribution

If a driver wants to be part of this service using the application, he/she has to send a request to the government, which is the only entity that can distribute driver account.
Each driver account is associated to a user account so that drivers are able to request a taxi if they need one.

(R7) Notifications

The systems will send notifications to both drivers and users.
Clicking on the notification, drivers are able to see the details of the request (the pick up location and, if it's a reservation, the destination) and either accept or refuse it, instead users can see the code of the taxi, track it and check the estimated time of arrival, but if there are no taxi drivers available the system will notify it to the user.
Then the notification is removed from the menù, but users and drivers can check informations on their account page.

3.1 Non Functional Requirements

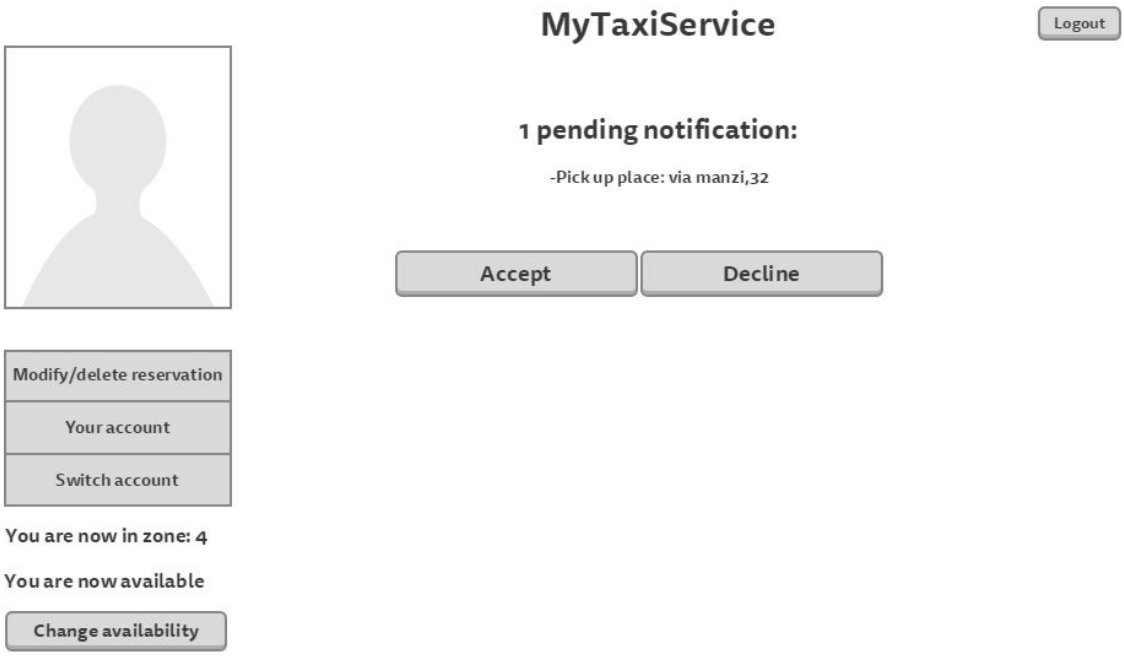
3.1.1 User Interfaces

The system will be implemented both in a Web application and in a Mobile application.
The registration and login functionalities will be provided in a single page which contains only those two forms.

After the login phase, users will be taken directly to their user page, drivers to their driver page, but with the ability to switch to their user page.
Here we show a prototype of a possible layout of the user page:

A user will be able to request or reserve a taxi from this page (if no previous request/reservation are active) and he/she will also be able to see the code of the taxi that he/she has requested, together with the estimated time of arrival.
If a reservation is already active he/she will be able to edit that reservation by clicking on the dedicated button, which will take him/her on a different page, which is shown below.

After the login drivers will be able to see a pending request (if they have one) and they'll be able to either accept or refuse that request.
They'll also be able to switch to their passenger account.
A first sketch of the driver's main page is shown in the next page.



From the same page they will also be able modify their availability, from available to unavailable and viceversa.

3.1.2 API Interfaces

The application will use a GPS system to compute the estimated time of arrival of taxis. The application will also provide programming interfaces to allow developers to increment the number of services offered by myTaxiService.

3.1.3 Documentation

The documents that we will write to better the describe the system to be will be:

RASD: Requirement Analysis and Specification Document, to define the given problem and to analyze it in a precise way, providing models and sketches.

DD: Design document to describe the design choices.

4 Scenarios

We now provide some possible examples of use of our system:

Scenario 1 - User requests a taxi

Alice needs to be picked up by a taxi to go home from work because public transport is on a strike.
She logs in the myTaxiService mobile application and requests a taxi to pick her up from her workplace. The systems checks the queue of available taxis in her zone, and sends a notification to the driver of the first taxi in the queue requesting to pick Alice up. The driver accepts the request and so, the system sends a notification to Alice containing the code of the taxi that will pick her up and the estimasted waiting time.

Scenario 2 - User reservation denied

Its 8 pm and Bob knows that he is going to a club this evening with his friends. He knows that he will drink a lot so he will not be able to drive home at the end of the night. He has to meet with his friends at 10 PM so,since he is still at home, he logs in to the mytaxiservice web application and reserves a taxi asking to be picked up at 9:30 at his house.
Unfortunately it's too late to reserve a taxi as there is only one hour and half until the desired pick up time so when he tries to reserve it the web application shows a message saying that his request cannot be fulfilled so he is forced to ask to one of his friends to pick him up.

Scenario 3 - Driver declines a request

Taxi driver Carl has just fullfilled a request for pick up and has just dropped a passenger at his desired place. He decides to stop driving for some time and parks his car in a parking lot to wait for a notification from the myTaxiService application saying that a user requires his services.Unfortunately, to park in that parking lot he left his previous zone and entered a new zone, so the system placed him in the last position of the queue of that zone.
After a long time he receives a request for a ride but notices that the passenger needs to be picked up in a really busy zone of the city, where he prefers not to drive, so he refuses that request and the system moves him again in the last position of the queue of the zone he is currently in.

Scenario 4 - User modifies reservation

Trevor is coming home from holidays in Puglia by train. While he's on the train he realizes that he needs someone to pick him up at the train station and drive him home. Unfortunately all of his friends are unavailable, so he decides to reserve a taxi using the myTaxyService

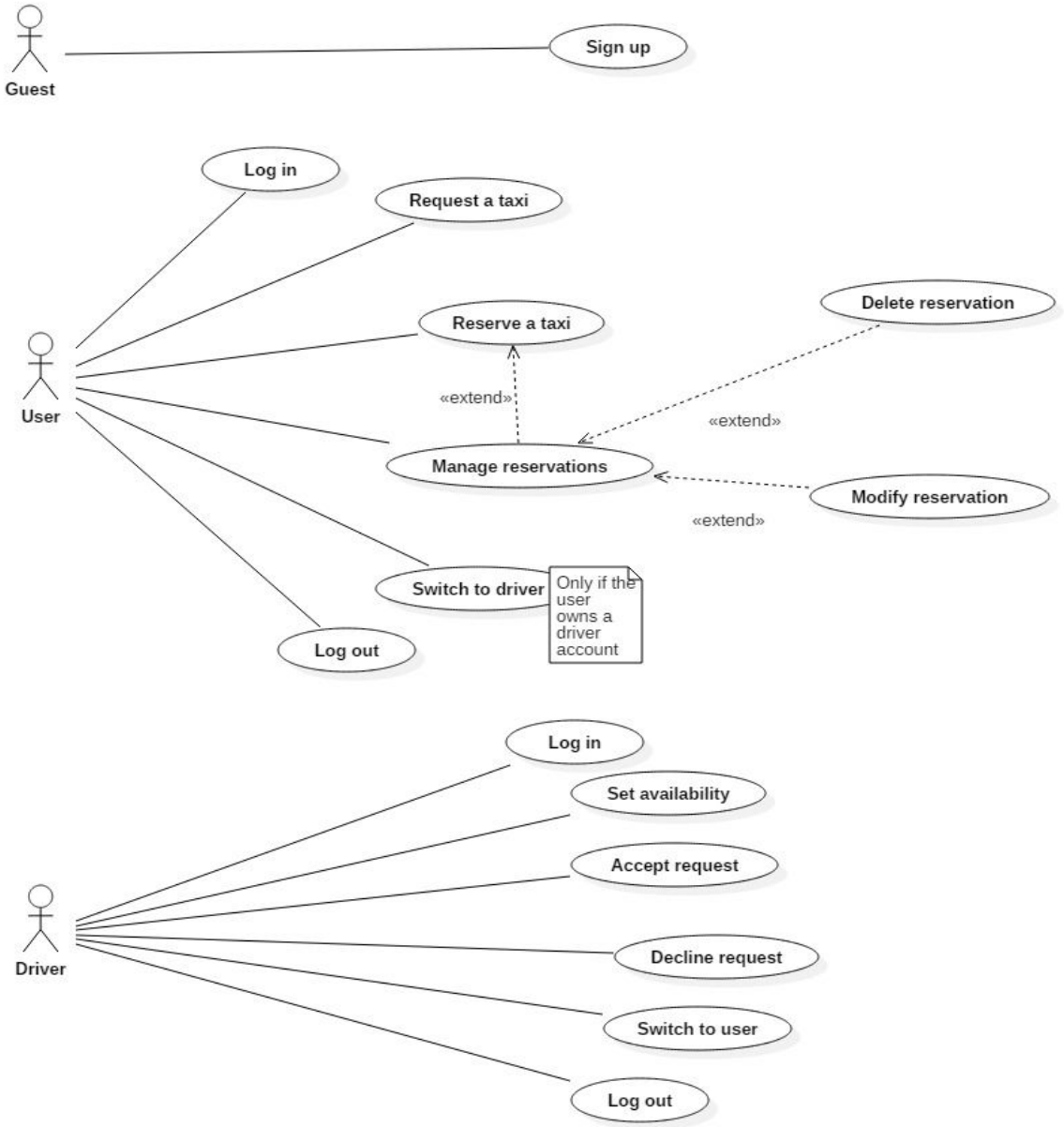
mobile application. He kwows that the train will arrive in Milano at 8 PM so he reserves a taxi to pick him up at Central Station at 8:10 PM.
During the train ride the conductor of the train announces that because of a failure the train will arrive in Milano with a delay of one hour. David therefore, logs in the myTaxiService application and modifies the reservation he has previously made by changing the desired pick up time from 8:10 PM to 9:10 PM.

5 UML Models

5.1 Use case models

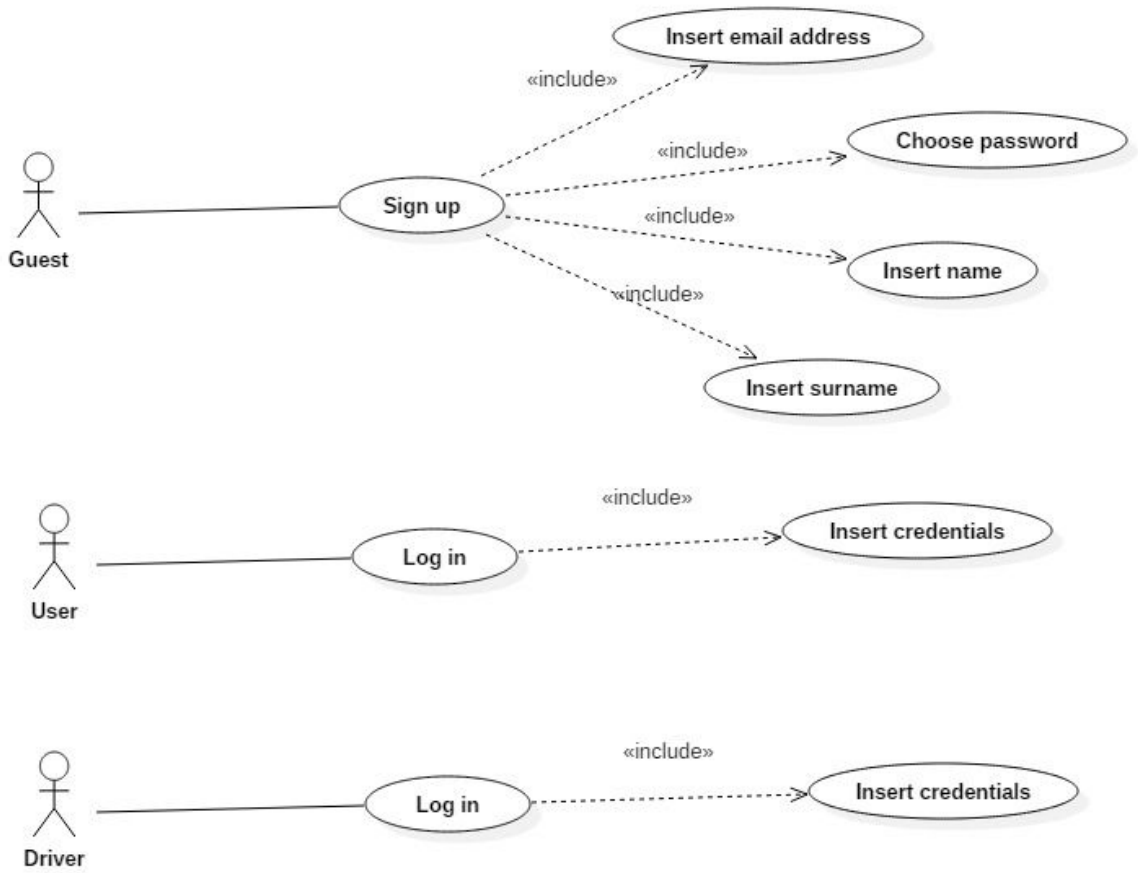
5.1.1 General use case

The diagram in the next page provides a general overview of the use cases of the application, according to the scenarios and the requirements stated above.
We will then deeper analyze some of the functionalities by providing further Use Case diagrams and Sequence diagrams.


Guest
Sign up
User
Log in
Request a taxi
Reserve a taxi
Delete reservation
«extend»
«extend»
Manage reservations
Modify reservation
«extend»
Switch to driver
Only if the user owns a driver account
Log out
Driver
Log in
Set availability
Accept request
Decline request
Switch to user
Log out

5.1.2 Registration and login

Here we analyze both the registration and login phases



- **Registration**

Actor	Guest
Preconditions	• Person who is not yet registered into the system.
Flow	1. Guest connects to the web application or to the mobile application 2. System shows a page containing a 'Sign up' form 3. Guest fills the form with the mandatory informations(e-mail , password, name, surname) 4. Optionally he can choose a profile picture

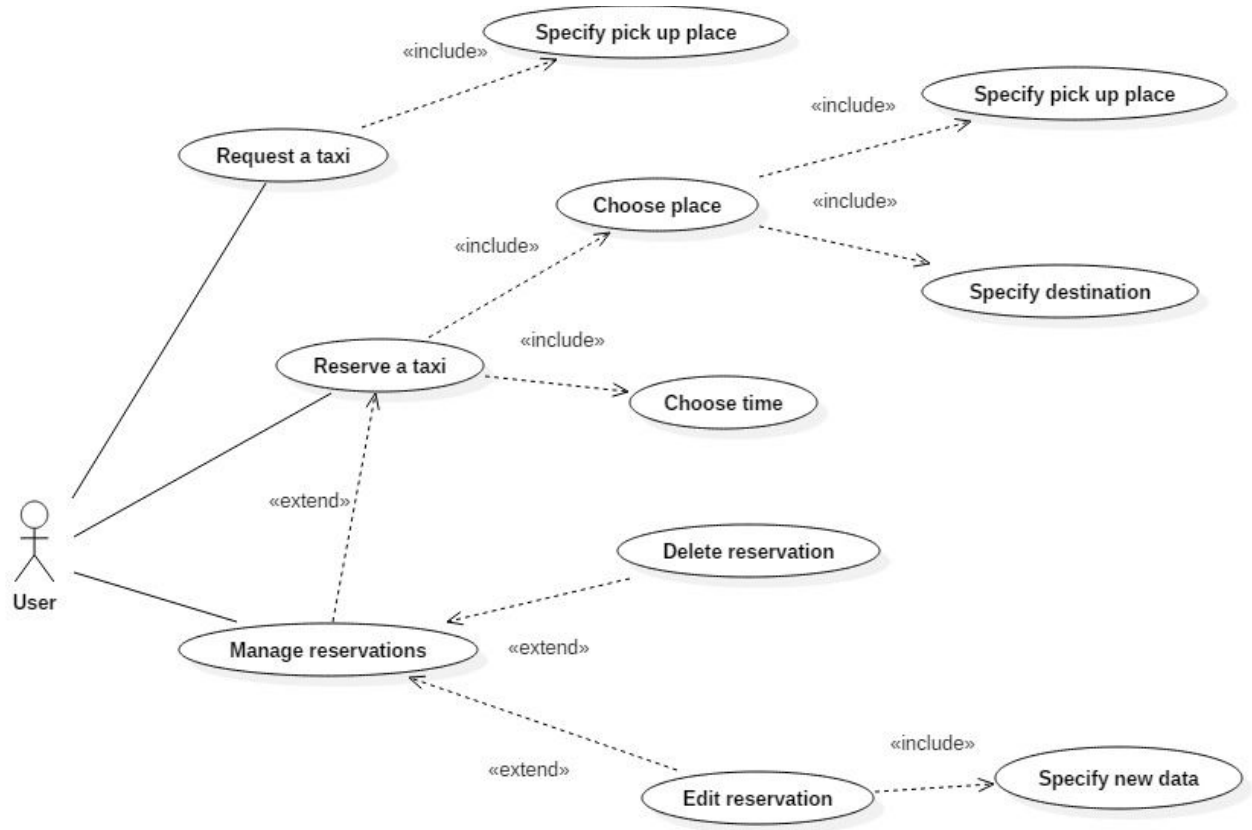
	5. The system checks whether that e-mail address already exists in the system. If it does then an error message is shown asking to input new data. If the e-mail is not in the system, then it is added and the guest is taken to the personal page as a logged user.
Postconditions	● New user data is actually added to the system.

- **Login**

Actor	User
Preconditions	● Person who owns an account in the system.
Flow	1. User connects to the web application or to the mobile application 2. Insert his credentials 3. The system checks the validity of the credentials.If those are valid, then the user is taken to his personal page.If those are not valid an error message is shown, asking to input new credentials.
Postconditions	● User is logged in the system and can use the application’s services.

5.1.3 Request and reservation

Here we describe both the request and the reservation of a taxi.
Please note that any mention of “buttons”, “forms” are only hypothesis to make things clear and easy to understand so that the reader may “visualize” what is actually happening, while the actual structure will be defined in the Design Document.



- **Request accepted**

Actor	User
Preconditions	• User is already logged in.
Flow	1. User clicks on the ‘Request’ button 2. The system checks if the user has an already active request or reservation. If he does then the system shows an error message

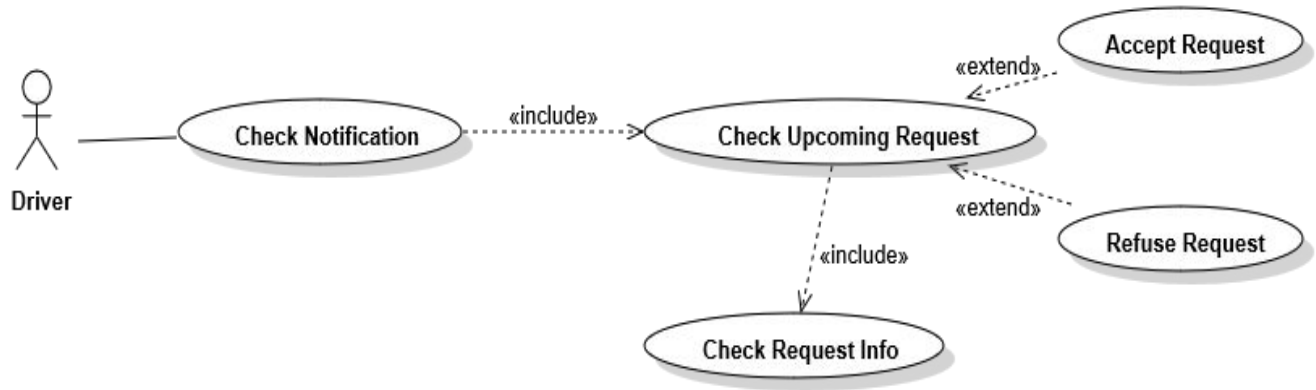
	saying that the user can't request a taxi until the previous request is fulfilled. 3. If the user does not have any active request the system shows a input form asking for the desired pick up location 4. User fills the form and submit the data 5. The system checks the data and if it is not valid asks the user to input valid data. 6. If the data is valid then the system assigns that request to the first driver in the queue of the zone to which the specified pick up location belongs to and then removes the driver from the queue. 7. The driver that has accepted that request is marked as unavailable and the system shows to the user a confirmation message including the code of the taxi and the estimated time of arrival.
Postconditions	● The driver that accepted the request is marked as unavailable. ● The user can see the code of the taxi that will pick him up and its estimated time of arrival. ● The user can't make any further requests until this one has been served.

- **Reservation modification**

Actors	User
Preconditions	● User is logged in. ● User has already made a reservation.
Flow	1. User clicks on the “modify/delete reservation button”. 2. The system checks if the desired pick up time (previously specified) is at least one hour from the current time. 3. If not, it shows an error message saying that is too late to edit the reservation. 4. If the test is passed then an input form is shown asking the user to input new data. 5. User inputs new data and submit it. 6. The systems checks if the data is valid. If it is then a confirmation message is shown to the user. 7. If it is not an error message is shown, asking the user to input new data.
Postconditions	● Data regarding the reservation is updated with the new values.

5.1.4 Notifications

In the following use case diagrams, we analyze both the user and driver notification



- **User notification**

Actor	User
Preconditions	• User is already logged in. • User has requested a taxi.
Flow	1. User clicks on the notification. 2. If a driver has accepted the request, the user checks the taxi info and the waiting time. 3. If there are no available drivers or they all have refused the request, the user reads a warning message.
Postconditions	• Notification is removed from the menù.

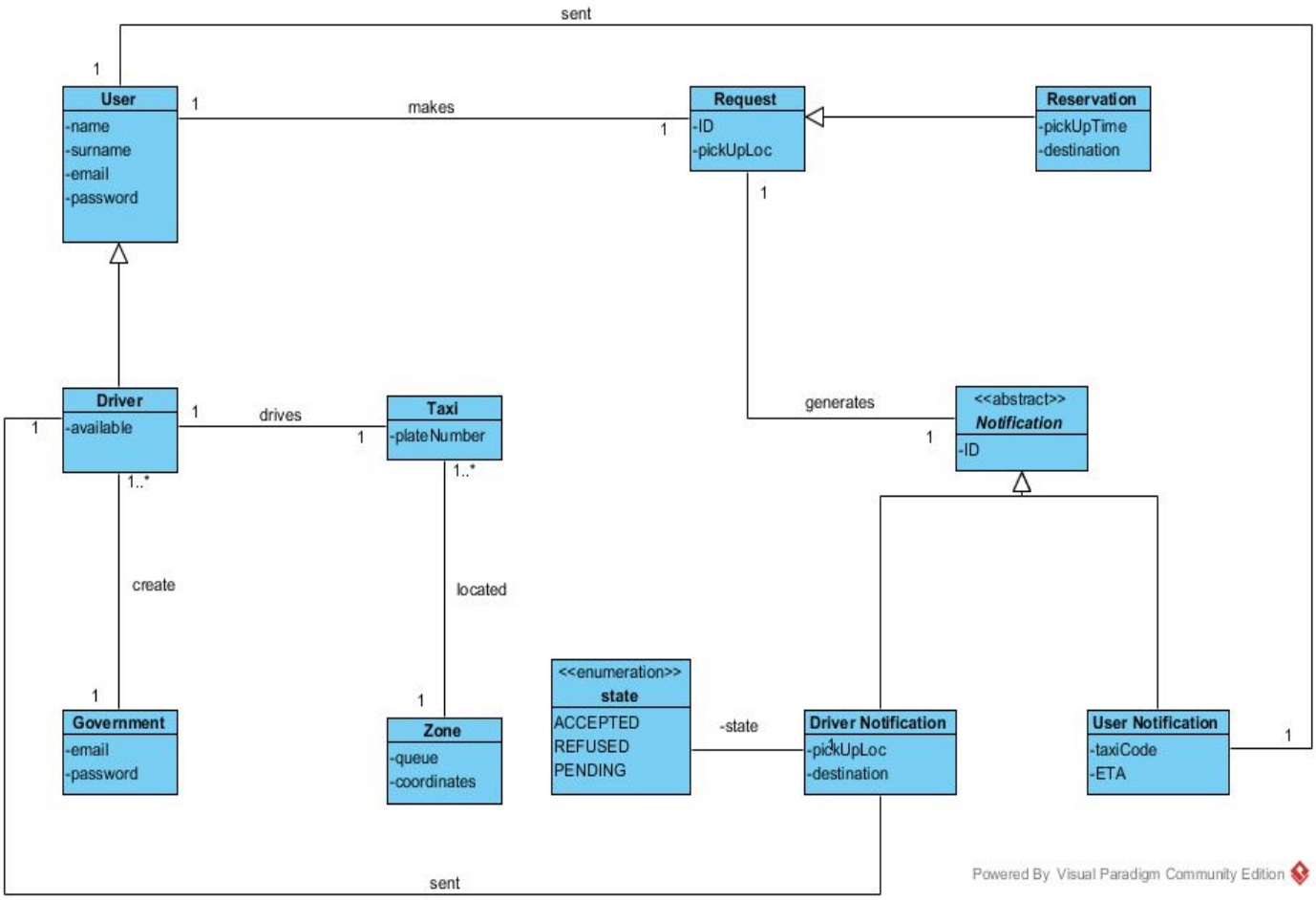
- **Driver notification**

Actor	Driver
Preconditions	● Driver is already logged in. ● Driver is available. ● Driver is on top of the queue related to the zone of the upcoming request. ● A user has requested a taxi.
Flow	1. Driver clicks on the notification. 2. Driver checks the request information and decides whether accept or refuse the request. 3. If the driver decides to accept the request, the system forwards a notification to the user giving the taxi information and the waiting time, while removes the driver from the queue and sets him/her as unavailable. 4. If the driver decides to refuse the request and there are other drivers in the queue, the system forwards the request to the second driver in the queue and moves the driver from the top to the bottom of the queue. 5. If the driver decides to refuse the request and there are no other drivers in the queue, the system

	sends a warning notification to the user.
Postconditions	● Notification is removed from the menù. ● If the driver has accepted the request, he/she is mark as unavailable and removed from. ● If the driver has refused the request, he/she is moved at the bottom of the queue.

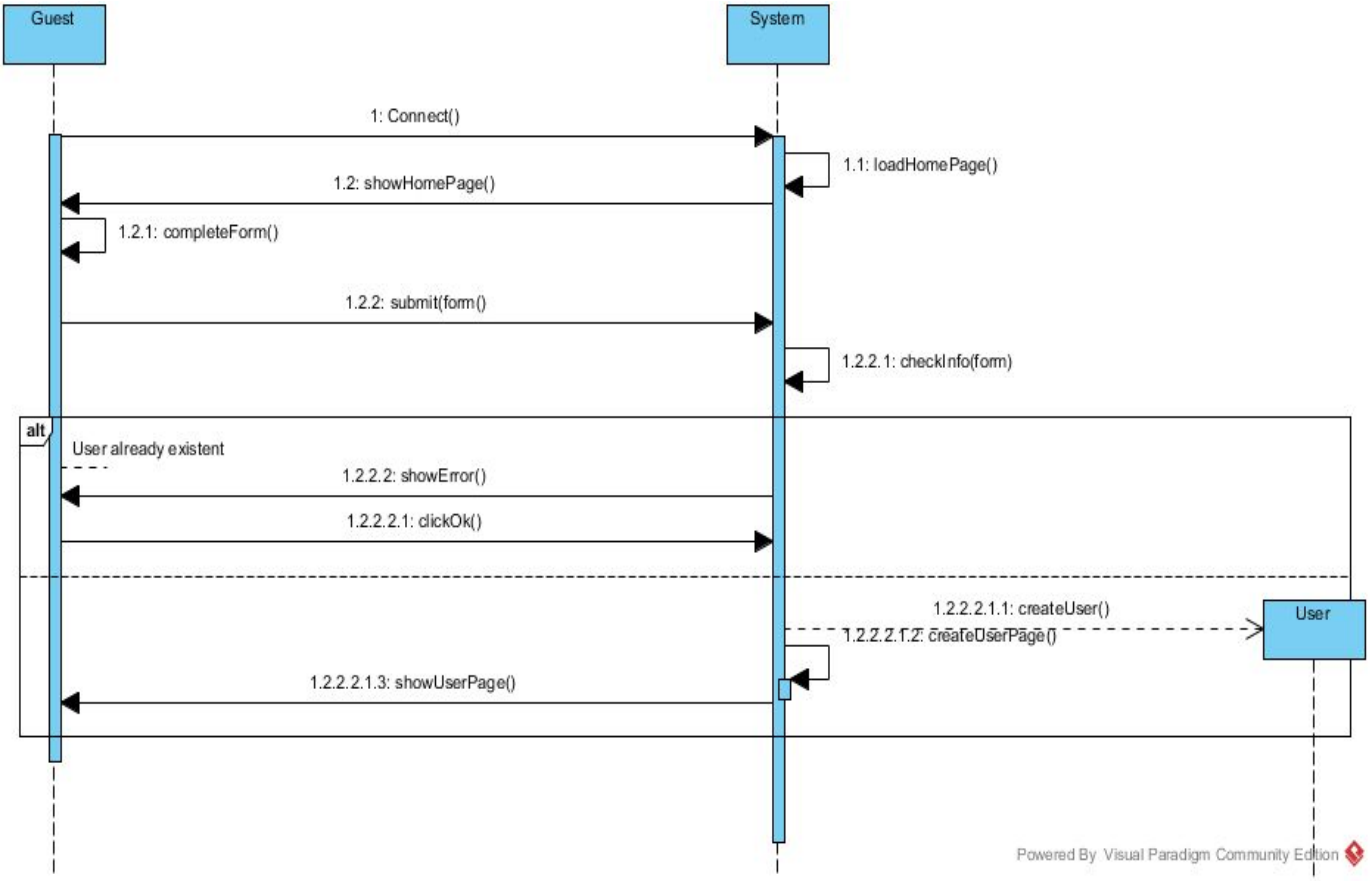
5.2 Class diagram

Now that we have defined some possible use cases we can derive a possible Class Diagram.
Keep in mind that we defined just the main attributes of each class, just to have an idea of the general structure, without being too specific on methods and further attributes of each class that will be defined further in the development.

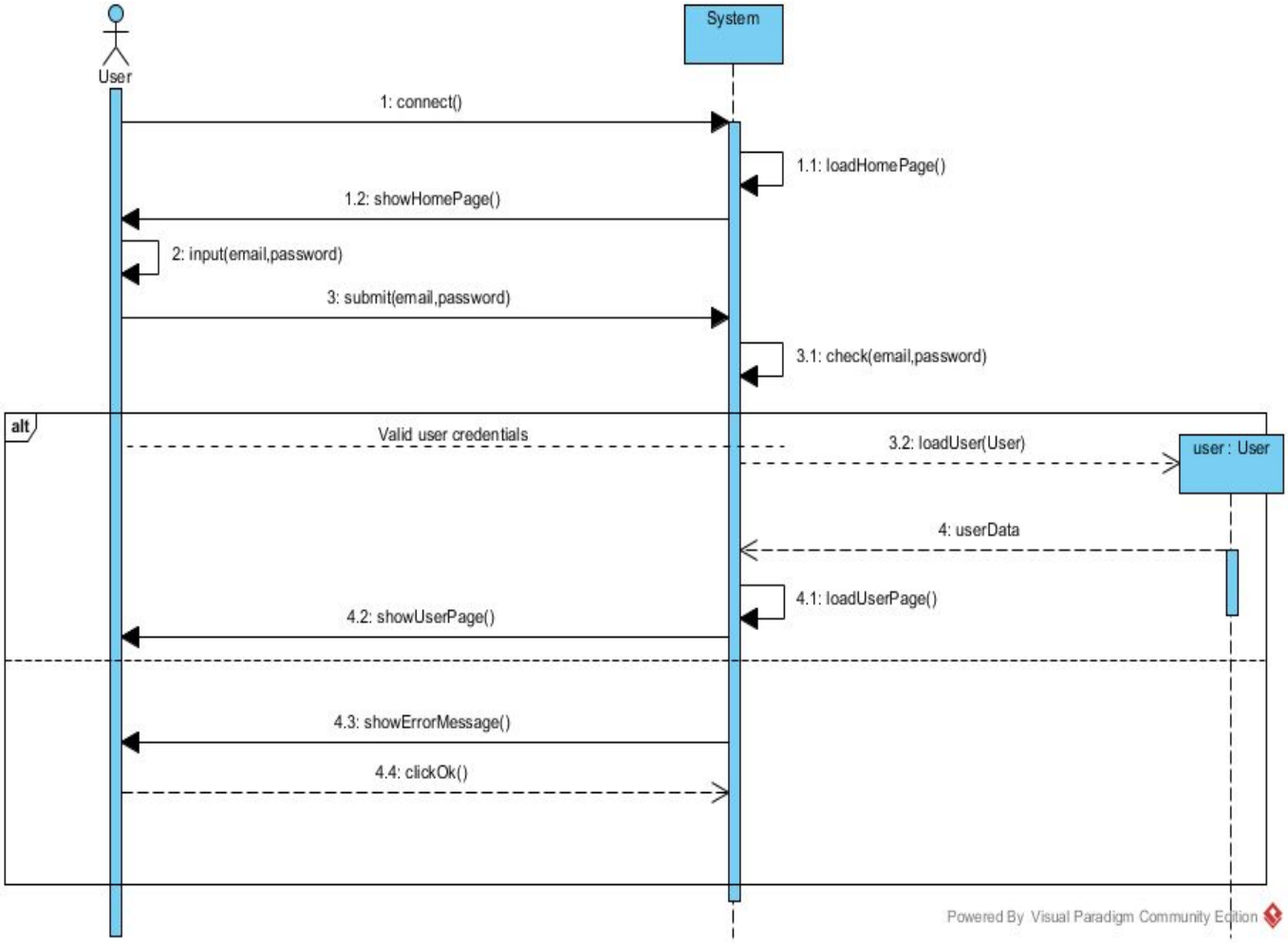


5.3 Sequence diagrams

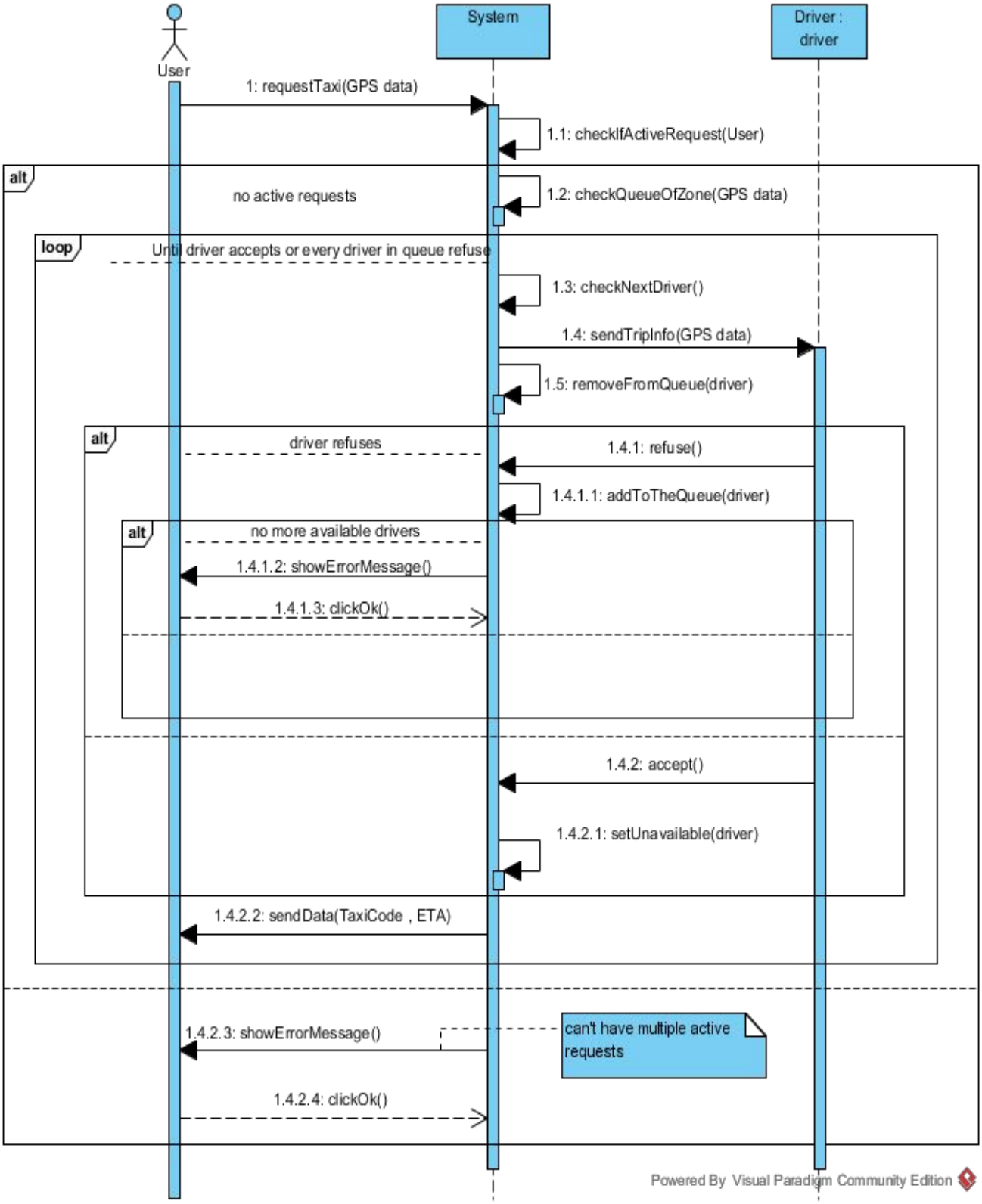
5.3.1 Registration



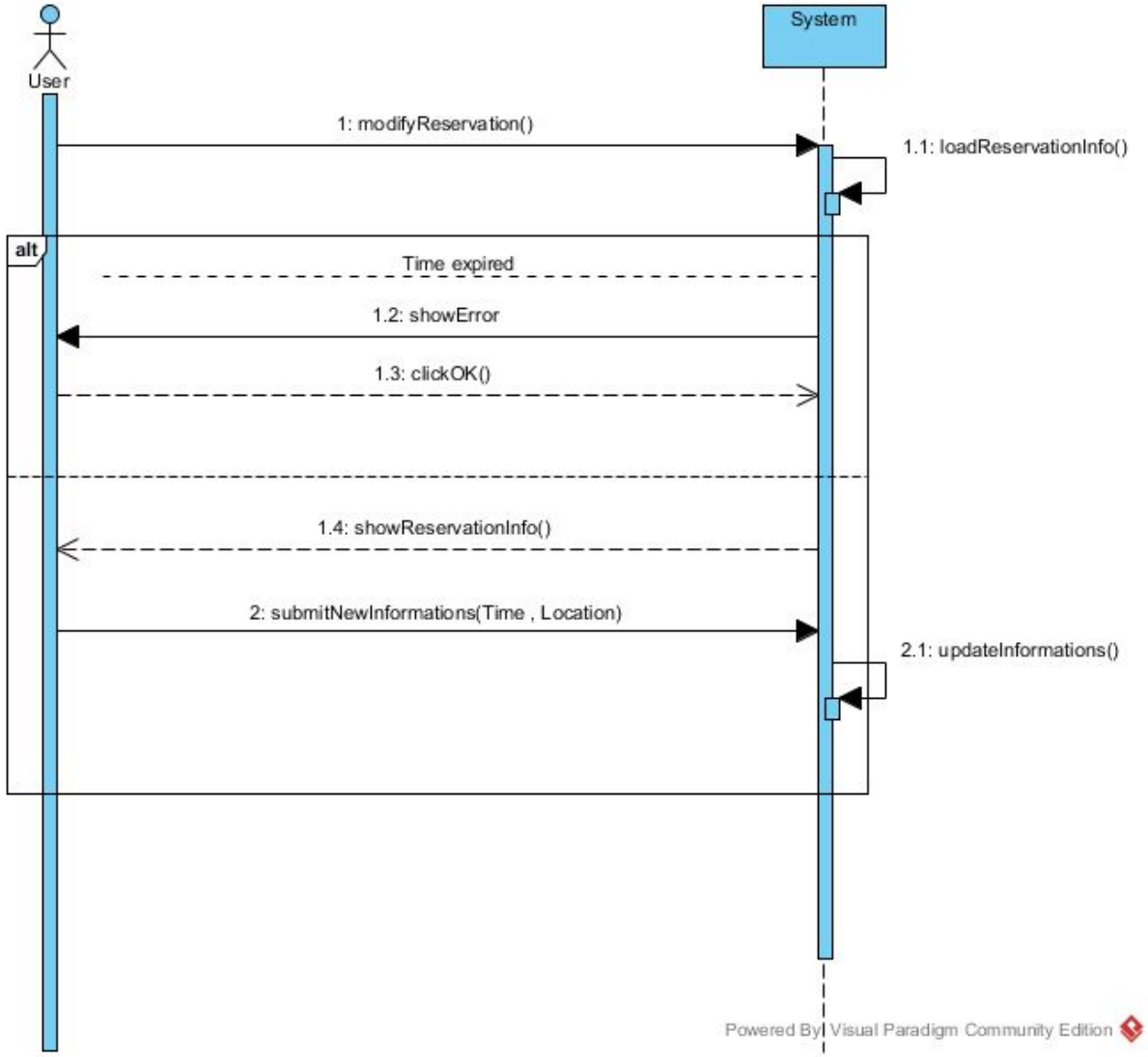
5.3.2 Login


User
System
1: connect()
1.1: loadHomePage()
1.2: showHomePage()
2: input(email,password)
3: submit(email,password)
3.1: check(email,password)
alt
Valid user credentials
3.2: loadUser(User)
user : User
4: userData
4.1: loadUserPage()
4.2: showUserPage()
4.3: showErrorMessage()
4.4: clickOk()
Powered By Visual Paradigm Community Edition

5.3.3 Taxi request



5.3.4 Reservation modification



6 Alloy

6.1 Introduction

To give a more precise idea of the world we are representing, we implemented an alloy model showing a possible state of a simplified version of the system.
This model is useful to represents constraints and verify if they can actually be satisfied, in order to see whether our design of the application is actually feasible or not.

6.2 Code

```
/*
We have decided to develop the main functionalities of our application, not
considering less important parts like the availability of the drivers and the fact that a
user can ask for a request in a zone without drivers. This decision was taken because
Alloy doesn't provide the possibility to "manage the time" and to better understand
the main functionality of our project.
*/

//SIGNATURE

sig Str{}

sig User{
        email: one Str,
}

/*
The Driver is also a User, but we have decided to not use "extends" to focus better on
the differences between them and to create worlds with number of User greater than
the number of Driver.
*/
sig Driver{
        email: one Str,
        working: one Int,
}{
        working >= 0
        working <= 1
} // working = 0 : means that the driver is not working, he/she is simply a user in the
                   world
  // working = 1 : means that the driver is working and so can serve a request
```

```
sig Request{
	user: one User,
	driver: one Driver,
	zone: one Zone,
}

/*
To simplify the management of the zone, we have decided to specify only the driver on the top of the queue of the zone (head) and a list of all drivers in the zone, including the head (drivers).
*/
sig Zone{
	head: one Driver,
	drivers: set Driver,
}

/*
There are two notifications, one for users and one for drivers. We could use only one notification, having both User and Driver in the signature, but to be coherent with our specifications, we have decided to have it separated.
*/
// user notification
sig UserNot{
	user: one User,
	request: one Request,
}

// driver notification
sig DriverNot{
	driver: one Driver,
	request: one Request,
}



//FACTS

fact numStr{

	#Str = #User
}
```

```
fact driverWorking{

        all d: Driver | d.working = 0 || d.working = 1
}

fact mailConstraints{

        all u1: User | all u2: User | (u1 != u2) => (u1.email != u2.email)

        all d1: Driver | all d2: Driver | (d1 != d2) => (d1.email != d2.email)

        all d: Driver | one u: User | d.email = u.email
}

fact zoneConstraints{

        all z: Zone | z.head in z.drivers

        all z: Zone | (#z.drivers = 1) => (z.head = z.drivers)

        all z: Zone | all d: z.drivers | d.working = 1

        all d: Driver | one z: Zone | (d.working = 1) <=> (d in z.drivers)
}

fact userNotDriverInReq{

        all r: Request | r.user.email != r.driver.email
}

fact requestConstraints{

        all d: Driver | all r: Request | (d.working = 1) => (r.user.email != d.email)

        all u: User | lone r: Request | r.user = u

        all d: Driver | lone r: Request | (d.working = 1) <=> (d = r.driver)

        all r: Request | r.driver = r.zone.head
}

fact notificationConstraints{

        all n: DriverNot | n.driver.working = 1

        all disj n1 , n2: UserNot | n1.user != n2.user
```

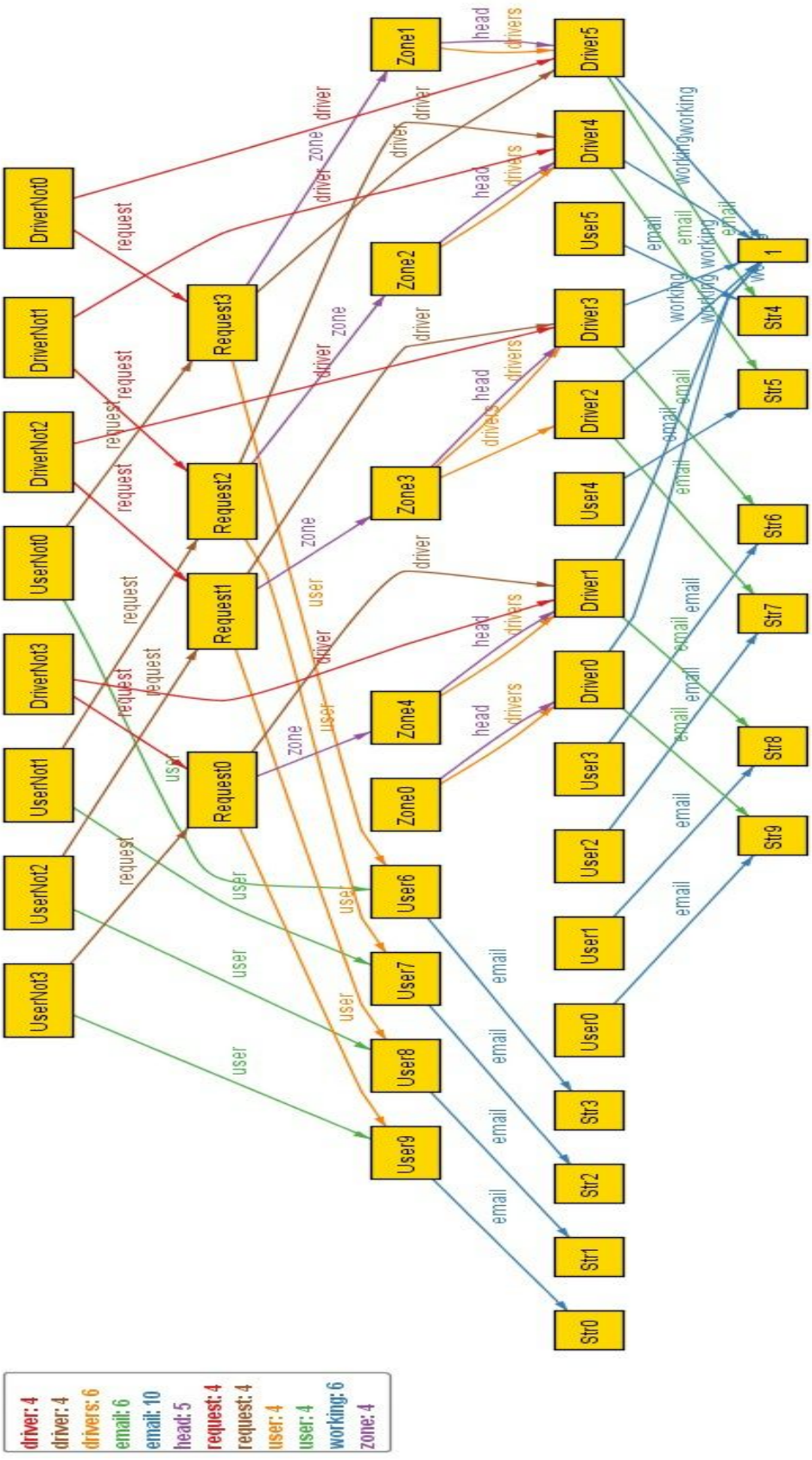
```
        all disj n1 , n2: DriverNot | n1.driver != n2.driver

        all n: UserNot | one r: Request | n.request = r

        all n: DriverNot | one r: Request | n.request = r

        all r: Request | one n: UserNot | r = n.request

        all r: Request | one n: DriverNot | r = n.request
}

fact reqNotConsistency{

        all n: UserNot | n.user = n.request.user

        all n: DriverNot | n.driver = n.request.driver
}


//ASSERTIONS

assert reqDriverInHead{

        all n: DriverNot | n.driver = n.request.zone.head
}

assert notOnlyToReqUser{

        all n: UserNot | n.user = n.request.user
}

assert userDriverNotEqualInNot{

        all un: UserNot | all dn: DriverNot | (un.request = dn.request) =>
                                        (un.request.user.email != dn.request.driver.email)
}

assert noNotifyNotWorkingDriver{

        no n: DriverNot | n.driver.working = 0
}
```

```
assert noNotifyUnrequestedDriver{

	all r: Request | all n: DriverNot | all d: Driver | (d not in r.driver) <=> (d not in
n.driver)
}



//PREDICATES

/*
The first predicate "show" describes a general world, while, in the next ones, we will
focus only on some single parts:

    • "showUserDriver" shows the relations between User, Driver and Zone (without
      Requests);

    • "showZone" shows the characteristics of a zone, having only one or more
      working drivers;

    •  "showNotificationRequest" shows some interesting facts, like that a
      not-working driver can request a taxi or that a User can request no taxis.
*/

pred show{

	#Driver = 6
	#Zone > #Request
}
run show for 10 but exactly 4 Request

pred showUserDriverZone{

	#Request = 0
	#Driver = 4
	#User > #Driver
}
run showUserDriverZone for 5 but exactly 1 Zone

pred showZone{

	#Request = 0
	#Driver = 5
	#Driver.working > 0
}
run showZone for 5 but exactly 2 Zone
```

```
pred showNotificationRequest{

        #Driver = 2
        #User > #Request
        #User > #Driver
}
run showNotificationRequest for 5 but exactly 1 Request

check reqDriverInHead

check notOnlyToReqUser

check userDriverNotEqualInNot

check noNotifyNotWorkingDriver

check noNotifyUnrequestedDriver
```

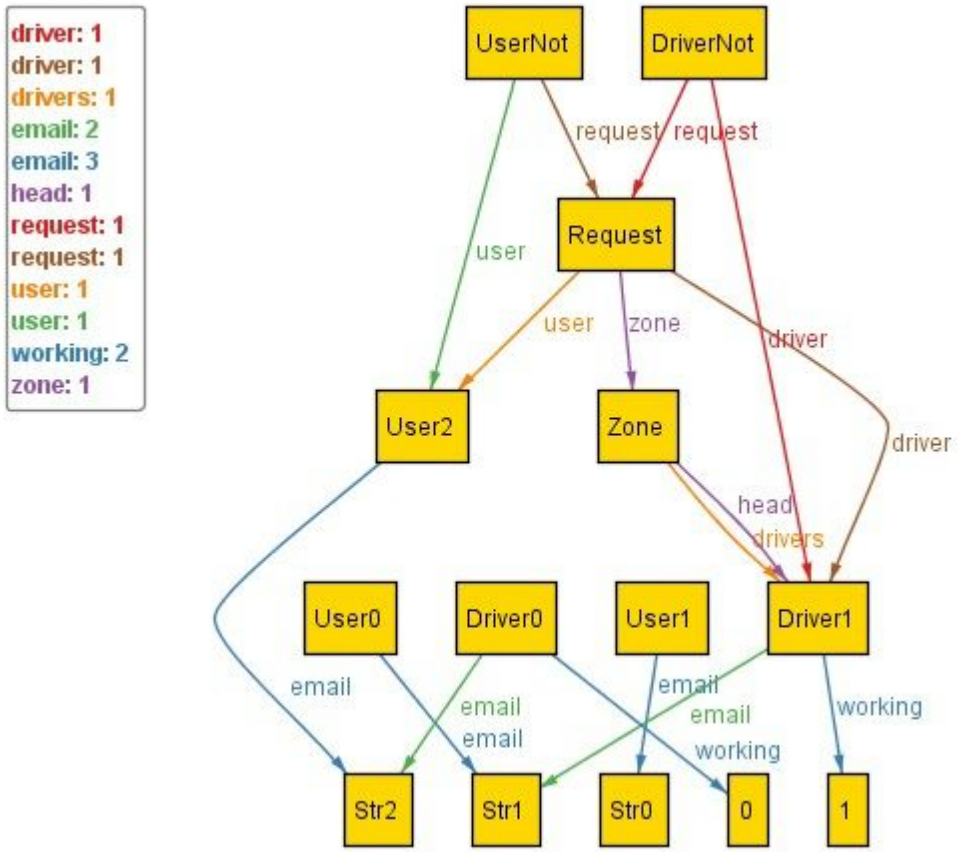
6.3 Generated world

We now show the world obtained throught the use of the predicate show():



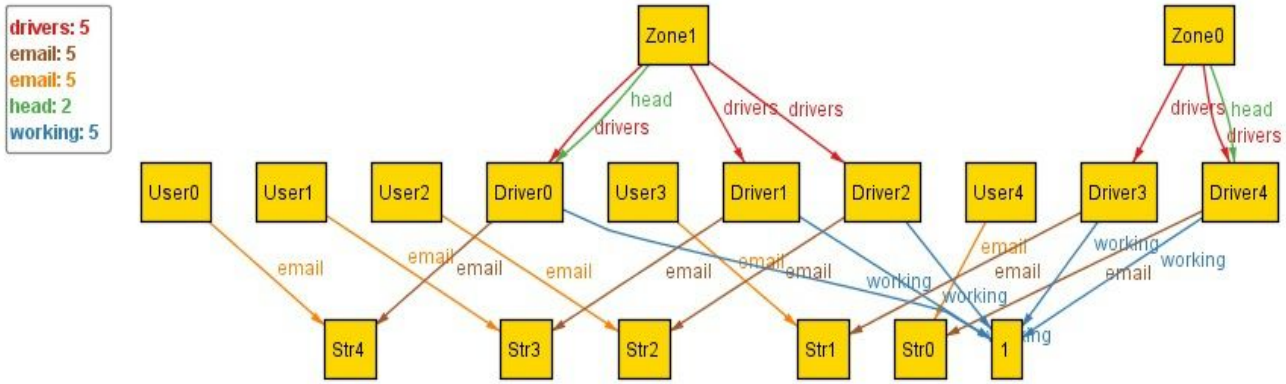
Since that graph it's not so easy to understand we created a few more graphs that show particular aspects of our world.

This is the graph obtained through the showNotificationRequest() predicate:



As we can see a request is associated to user (which is the one that generated that request) and to a driver (which is the one in the head of the queue of the zone from which the request was generated).
In addition, two notifications for that request are generated, one that is sent to the user that generated the request and the other that is sent to the driver that accepted the request.
Notice that the driver that accepted the request has the attribute working equals to one, meaning that he is actually allowed to receive requests, while the other driver (Driver0) has working=0, so he is not able to receive requests.

In the following graph we show the relations among users and drivers and the distribution of drivers among different zones of the city:



As we can see each driver is related to one user meaning that, each driver owns a user account in the system, while there are users who do not own a driver account.
Available drivers are also distributed among the 2 zones and, for each zone, only one of its drivers is marked as the head of the queue and is able to receive requests.

7 Conclusion

To redact this document we used the following tools:

- Google Docs (https://docs.google.com/) to redact and format the document
- Mockingbird (https://gomockingbird.com) to create mockups of UI
- Star UML (http://staruml.io/) to design Use Case diagrams
- Visual Paradigm (http://www.visual-paradigm.com/) to design Sequence and Class diagrams
- Alloy Analyzer (http://alloy.mit.edu/alloy/) to prove the consistency of our model

To create this document we spent approximately 30 hours each.

8 Revision

We have decided to remove the possibility for users to track an incoming taxi and substitute that functionality with the possibility for both users and drivers to edit their profile settings.
Taxi tracking is a functionality that may be added in successive releases of the application.