REQUIREMENTS ANALYSIS AND SPECIFICATION DOCUMENT

myTaxiService

Reference Professor: Elisabetta Di Nitto
POLITECNICO DI MILANO
Computer Science and Engineering
Project of Software Engineering 2

Andrea Binelli
858235

Sommario

1. Introduction

1.1 Description of the given problem

I will project myTaxiService, which is an online system that want to improve and optimize a taxi service.

More specifically, it wants to simplify the access of passengers to the service and guarantee a fair management of taxi queues.

My vision is “a city where taxis goes to the customers”.

The system should be able to register new users with their information (such as name, address, email, phone number...) and once they are logged in permit them to request a taxi service in a specific location.

The system should be able also to manage the queues of free taxis for each zone in a centralized way.

1.2 Goals

The main goal of the system is the optimization of the existent taxi service of the city.

I thought about the possible users and what myTaxiService should provide them, so I planned to give our system these features:

Users should be able to:

- Sign up into the system;
- Log into the system;
- Change user information and password
- Request a taxi service;
- Be notified by the system for a response to the request and the system will give also the number of the incoming taxi and the waiting time;

Taxi drivers should be able to:

- Inform the system about their availability;
- Confirm that they are going to take care of a certain request;

The system should provide:

- A request to a taxi available in that zone;
- A response to the user that request a taxi service.

1.3 Domain Properties

I suppose that the following properties hold in the analysed domain.

- Every address in the city is unique;
- The user that request a taxi had to be in the city;
- Taxis are always connected to the system;
- Taxi drivers have their personal taxi with a unique code.

1.4 Glossary

The following list is an accurate definition of every words that I will use frequently in the documentation in order to avoid misunderstandings.

- USER: is a person already registered in the system. When he sign up the user provides all these information:
    - First name;
    - Last name;
    - Email;
    - Telephone number;
    - Password;
- GUEST: is a person who has not signed up yet. They cannot do anything in the system except signing up to become a user.
- REQUEST: is the event made by the system when the user needs a taxi. It is characterized by the user, the address, the state of the request (waiting for the response, confirmed) and two information that will be available only when the request is confirmed: the code of the taxi that will come and the estimated waiting time.
- TAXI: is the combination of the physical taxi (the car) and the driver. It provides information such as the code of the taxi, the position and other additional information like the complete name of the driver, the model of the car and the number of seats. Some of these information are not strictly required but can be useful for a future upgrade of the application, for example the implementation of the taxi sharing that needs the number of seats. In some parts of the projects is needed the division between the Taxi and the Driver in order to distinguish the actor from the entity (and even from the interface).
- QUEUE/ZONE/AREA: they are three terms related each other. A zone is a rectangular area and there is one and only one for every zone. A queue contains, in an ordinate way, all the taxis available for that zone. In the DD document, I have often collapse these three term in Area.

- SYSTEM: is the automatic part of the project and it will be implemented with two types of clients (one public app or via browser for the user and a private app for the taxi) and a centralized server that will provide the services.

1.5 Assumptions

I want to make some assumptions about points in the document that are not very clear.

- A user cannot do multiple requests (he have to wait that the current request is confirmed before doing another request, even if this case hasn't much sense in the real world).
- A taxi is already registered into the system. I assume that this is a process with operations (economic logics, licenses verifications, registrations of cars…) which falls outside the scope of this system and for this reason it cannot be done like the registration of users (a taxi driver had to present his license, pay for this service…)
- A taxi interacts with the system with a customized smartphone given by the company that commissions the application. The choice of the bespoke device is for many reason:
    - The device must be able to perform quickly the app and they had to be always on;
    - The device must have a GPS and an internet connection that are always on, the company has to stipulate a contract with the most appropriate telephone company for commercial purposes;
    - Personal information of the driver doesn't change, there's no need to change them by the smartphone;
    - Information of the taxi doesn't change often, there's no need to change them by the smartphone;
    - The application driver-side must be private and not available for everyone to be downloaded on a store like App Store for iOs or Play Store for Android, for this reason, Android with a private apk seems to be the best candidate for this purpose.
    - For all the reasons mentioned above a personal device is not the correct solution.
- A taxi cannot accept multiple request. After a confirmation for a request, it become automatically unavailable until the driver turns on manually the availability switch. The system does not have the possibility to control if actually there is no passengers on a taxi that appears available, the correct utilization of the application had to be done by the driver.
- The system will not record any ride and for this reason the gps information of unavailable taxis are useless.
- Every reservation for an event is intended for just one person. The participation of other passengers along with the user is not expected and the decisions is left to the driver.

- The system does not involve about the destination of the ride, which is something that the user speaks directly to the driver.
- When a user request a taxi, he cannot cancel the reservation. I can suppose that the only way to abort the ride is to speak directly to the driver when it comes and pay an extra fee for the inconvenience but the system does not recognize this type of exceptions and the taxi can return available for that zone.
- The system does not keeps track of any commercial transaction, extra fee o tip. The price of the ride is calculated externally to the system.
- The system does not keeps track of the number of hours of work/pause and the faults that every driver do. It is not a purpose of the system to control the work of the taxis.
- An available taxi can goes across the city in every moment. When an available taxi goes across the limit between two zones, the system will automatically transfer the taxi from a queue to another, more precisely to the last position of the new queue.
- A driver can make his taxi unavailable when he wants, for example if he make a pause.
- A driver can also drives his taxi outside the city but the taxi will remain or become automatically unavailable. Examples: a taxi finishes his ride outside of the city -> the taxi remains unavailable; an available taxi goes outside the city -> the taxi becomes unavailable.
- The system is not interested of what happens after a confirmation by a taxi (except for forcing the taxi unavailable). A taxi ride can abort for many things (a user is not in the correct place, the taxi breaks down…).
- If there are no taxis in that zone the system will provide a bad response for the request and invite him to retry in a few minutes. The system will not implement a complex policy like calling a taxi from a neighbouring zone (it can be a future upgrade).
- If the address (of a request) is not in the city the system will provide a bad response to the user and invite him to retry.
- The city is divided into rectangular zones like a chessboard. This facilitates the localization of the taxis by coordinates.
- Four geographical lines uniquely identify a rectangular area: latitude of the upper side, latitude of the lower side, longitude of the left side and longitude of the right side.
- Combining the two last assumptions I assume that division of the city in sub-areas is made by seeing the city from above with the proper orientation (the up coincide with the north).
- I assume that there can be hundreds of zones (a city can easily have more than 100km^2 so more than 100 zones) and starting from this assumption the system, for each zone, has to keep track of the adjacent zones in order to improve the performances.

- According to the assignment, a taxi can deny a request from the system. I want to prevent two exceptional cases:
    - There are some taxis in the queue and they all deny the request. I can prevent this case by adding a timer (when it reach the timeout I stop from calling taxis).
    - There is only one taxi in the queue. I do not want to call it just after it denied the previous request. I can prevent this taxi by stopping the procedure if the queue has only one taxi.

1.6 Identifying stakeholders

The main stakeholder in this project is the professor who gave us the specification of the problem.

This description of the problem can be interpreted, documented and implemented in many ways with the appropriate assumptions and premises.

I will focus on the main functionalities of the system, which can be directly elicited by the specification then I will try to extend the functionalities of the system in order to make the application more suitable for an hypothetical final customer of the application (in this case that is a government of a large city).

2. Actors Identifying

The actors of the system are three:

- Guest: as I already mentioned in the glossary section, is someone who has not signed up yet and has only the capability of signing up.
- User: as I already mentioned in the glossary section is someone who has already signed up. The user can take advantage of the features of our system.
- Taxi driver: is the actor that drives his own taxi (the physical person). He interacts with the system in a different way than the user does.

3. Requirements

I have determined the following requirements according to the Jackson and Zava analysis, so assuming that the domain properties, which are described above, holds for our purposes, I have written the requirements in order to satisfy goals.

- Registration of a guest into the system
    - The system has to provide a sing up functionality for the user.
- Change profile information
    - The system has to provide the user the possibility to change profile information.
- Change password
    - The system has to provide the user the possibility to change the password.
- Make a request
    - The system has to provide a function that allows users to make a request for a taxi at a specific address.
- Managing the request
    - The system has to take care of the requests, search for a taxi and provide a response with the code of the taxi and the waiting time or a bad response.
    - In order to do this the system has to compute the distribution of taxis in the various zones based on the GPS information it receives from each taxi and update the queues.
    - If the proper queue (the queue associated with the zone in which the address is located) is empty, the system will provide a bad response to the user.
    - If the address is not in the city, the system will provide a bad response to the user.
    - Else the system has to request the first taxi in the queue (of the proper zone) to take care of the service.
    - If the taxi confirms, then the system will send a confirmation to the passenger and will make the taxi unavailable. If not, then the system will forward the request to the second in the queue and will, at the same time, move the first taxi in the last position in the queue.
- Update the queues
    - The system has to update the proper queue when a taxi turning available by adding the taxi into the queue[1] corresponding to the zone in which the taxi is located.

    - The system has to update the queues when an available taxi goes through a border of a zone by removing the taxi from the old queue and adding it to the new one[1].
    - The system has to update the proper queue when a taxi turning unavailable by removing the taxi from the queue corresponding to the zone in which the taxi was located.
    - The system has to update the proper queue when an available taxi goes through the border of the city by removing the taxi from the queue corresponding to the zone in which the taxi was located.

1 - Adding the taxi into the queue means pushing it at the last position of the queue

- Taxi notification
    - Taxi drivers use a mobile application to inform the system about their availability/unavailability.
    - Taxi drivers use a mobile application to confirm to the system that they are going to take care of a certain call (or not).
    - When a taxi driver (available) accepts a request, the system will force his availability switch to off. The driver had to turn on manually the switch when the ride of the last passenger is finished.
    - Taxi drivers have installed a gps locator to inform the system about their position when they are available.

3.1 Functional Requirements

Now that I have defined the main features of myTaxiService, I can find some functional requirements concerning each defined actor:

- Guest: he can
    - Sign up.
- User: he can
    - Log in;
    - Modify his profile information;
    - Require a taxi service;
    - Receive a response from the system.
- Taxi: he can
    - Turn his state into available/unavailable (with the available mode the taxi automatically provides the GPS information to the system);
    - Confirm that they will take care of a request or not.

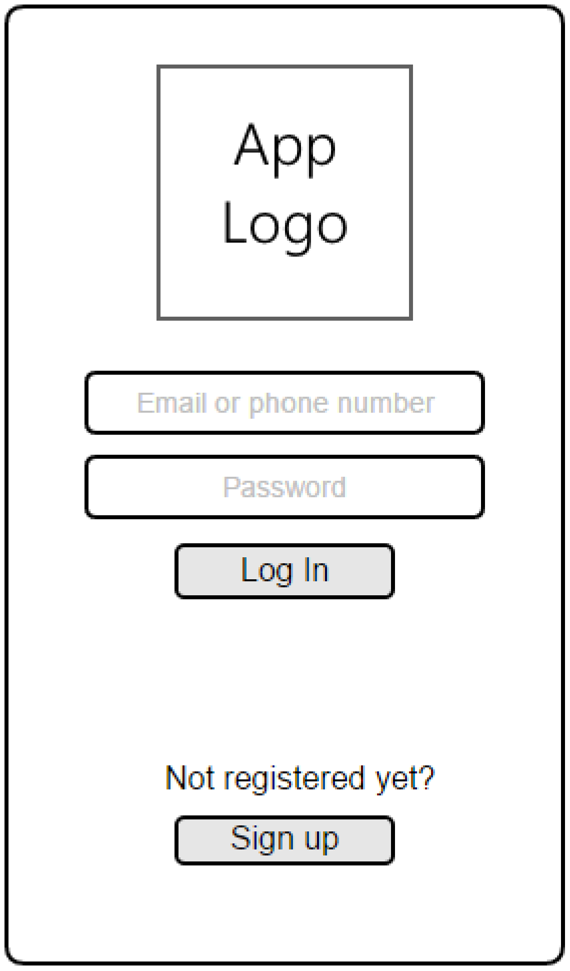
3.2 Non-functional Requirements

3.2.1 User interface

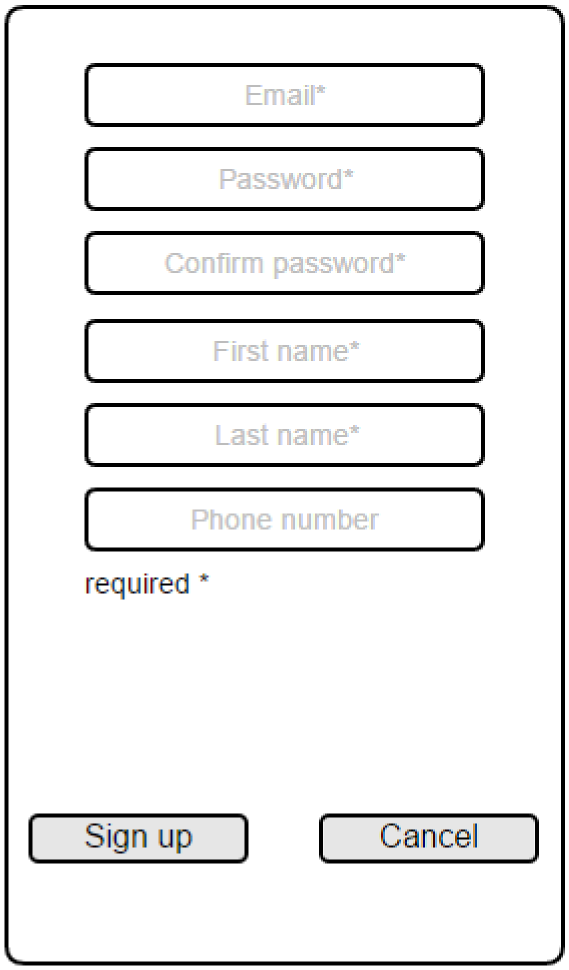
The interface of our application is thought to be used via browser or app. I suppose that the application will be used mainly by mobile devices like smartphone, phablet and more rarely by table or desktop/notebook pcs. Also the interface doesn't require a lot of graphical objects so I've thought to design a simple, flat and smartphone like web page (like Instagram) that could be easily converted to a dedicated app for the most commons mobile os.

These are sketches of the tabs of the application that will be used by the user.

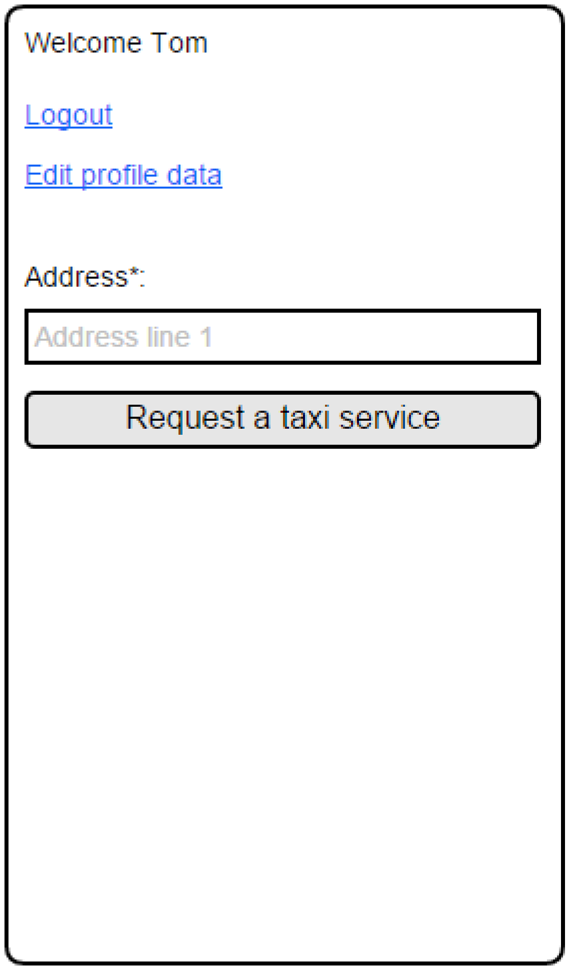
This is the Login tab, where the guest can log in or open a new tab to sign up.



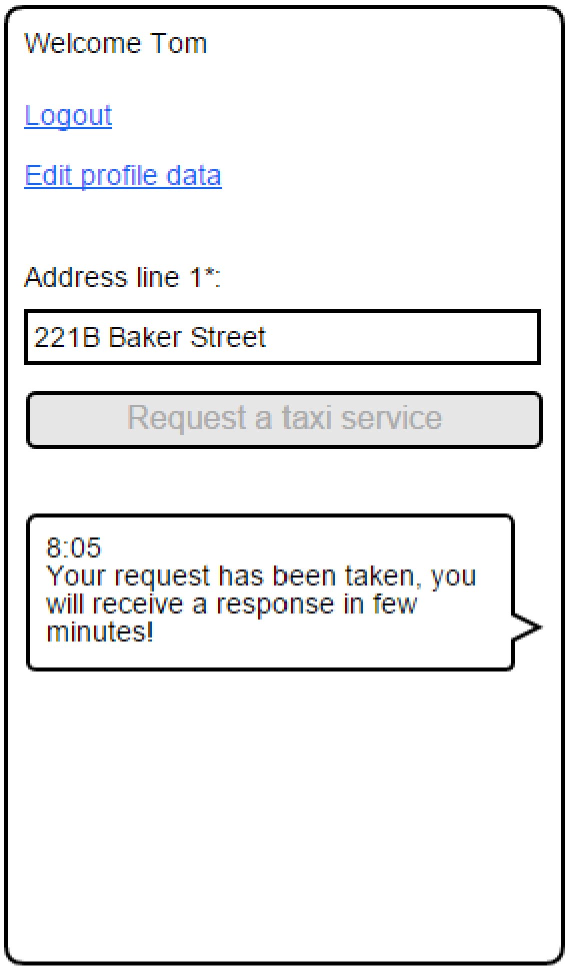
This is the Sing Up tab, where the guest can sing up in order to access the system.



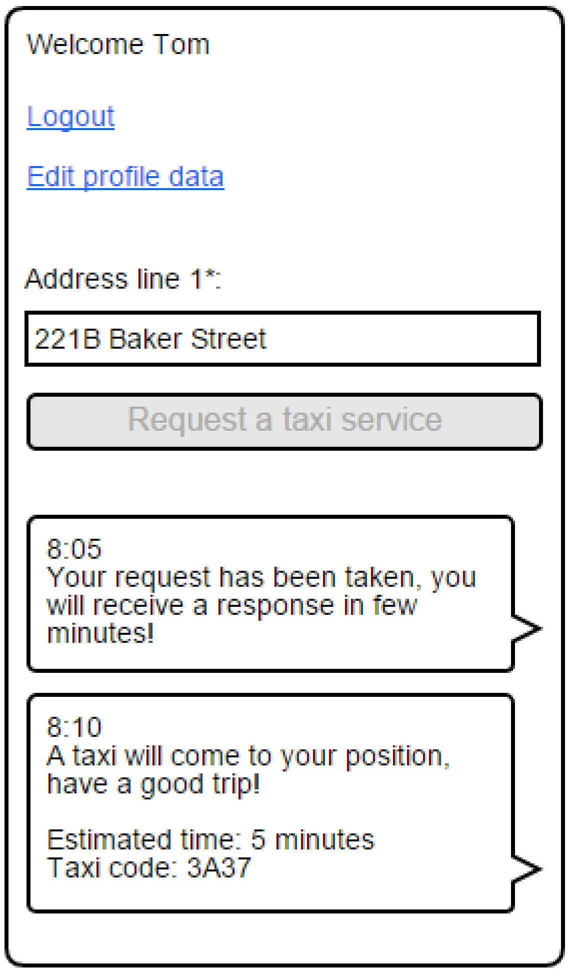
This is the Main tab of the application, where a user can call the taxi service.



In the following two images, we can see a simple and progressive iteration with the interface (in this case calling a taxi and receiving a response from the system).



8:05
Your request has been taken, you will receive a response in few minutes!


Welcome Tom

Logout

Edit profile data

Address line 1*:

221B Baker Street

Request a taxi service

8:05
Your request has been taken, you will receive a response in few minutes!

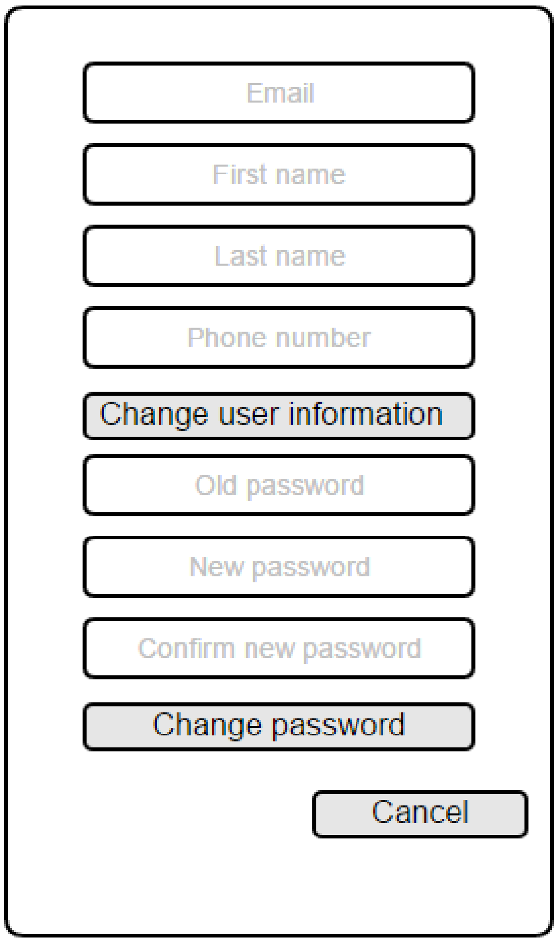
8:10
A taxi will come to your position, have a good trip!

Estimated time: 5 minutes
Taxi code: 3A37

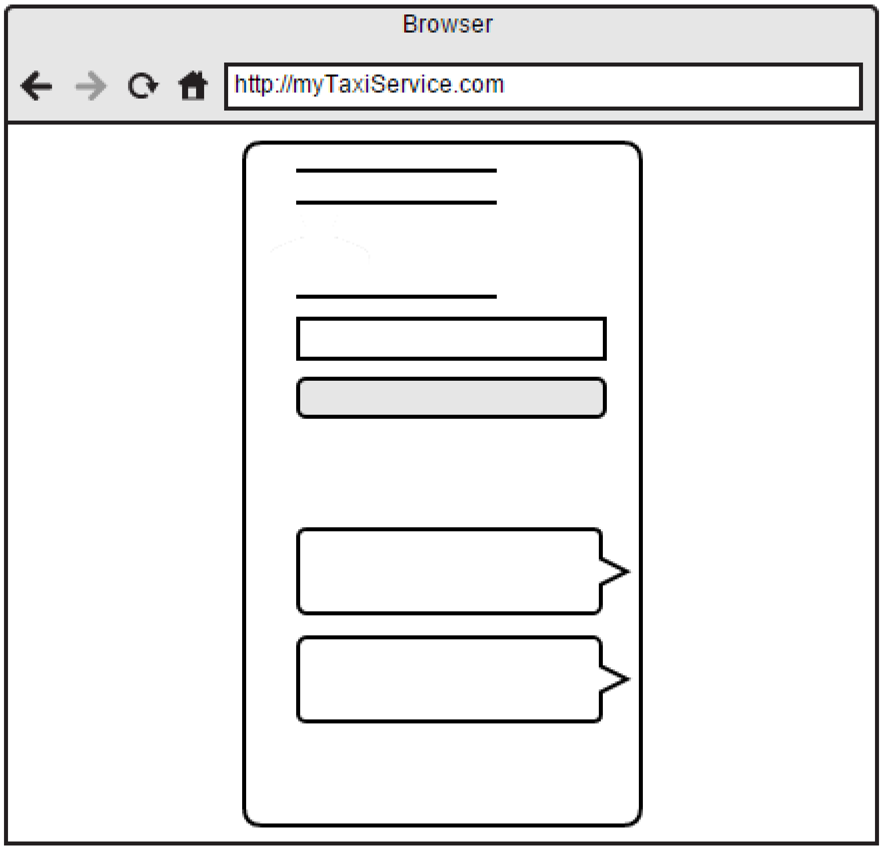
From the main tab, I can either go to the previous Login tab (by pressing logout) or to the Edit Profile tab.

This is the Edit Profile tab. With the cancel button, I return to the Main tab. I assume that the user can change any information about email, first name, second name and phone number by pressing "Change user information" (if something will be entered in the corresponding textbox the system will update the information, it may return an exception popup if for example the email or the phone number is already taken by another user).

The changing of the password is a bit different because all the corresponding fields must be filled (and obviously, there can be errors, for example is the old password is incorrect or the new is too short).



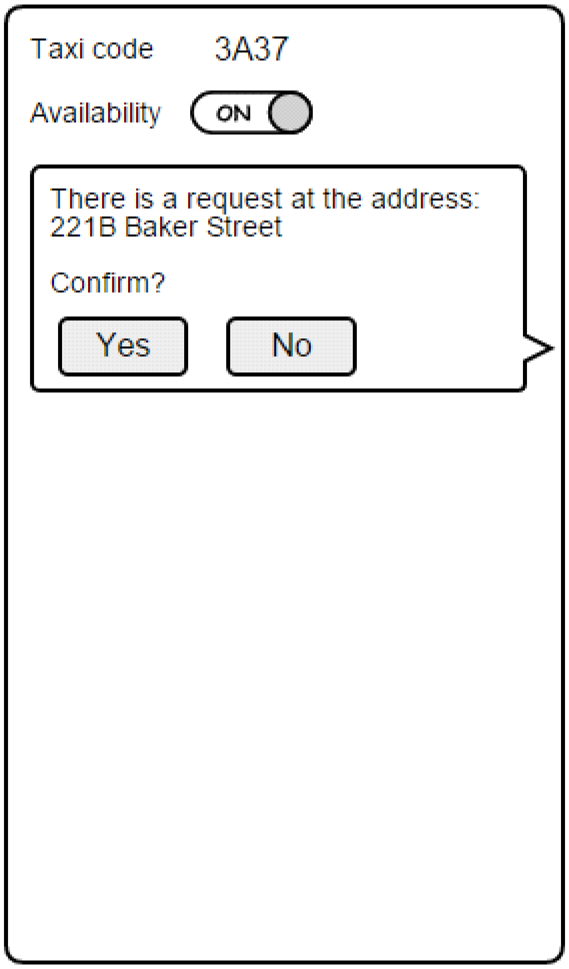
Finally, (for the user side) this is what the application can looks like if it is visualized on a bigger screen with a browser. I have taken inspiration from the site www.instagram.com where a similar interface of the mobile app is visualized on a desktop mode.



The taxi side application is simpler than the user side app because the driver use an early-configured device and can do few things reassumed in just one page.

Like previously, the requests from the system appear like messages. I think that paradigm fits perfectly for the purpose of this application.

The (manual) availability can be easily implemented with a simple switch. When the switch are off no message can be received by the system (because the corresponding taxi disappears from its queue). If the taxi turn down the availability switch before answer the request (and after receiving it), it will appear as a NO answer.


Taxi code 3A37

Availability ON

There is a request at the address:
221B Baker Street

Confirm?

3.2.2 Documentation

I will release the following documents in order to organize our work in each phase of the development process and keep in touch with the stakeholders.

- RASD, Requirement Analysis and Specification Document, which defines our goals and assumptions and contains an overall description of the system (using scenarios and use-cases) and the models describing requirements and specifications. I included in the RASD also the alloy and GUI (graphical user interface) parts.

- DD, Design Document, which contains a functional description of the system using models such as UML diagrams.

4. Scenarios Identifying

Here are some possible scenarios of myTaxiService

- Jack arrive to a new city by train. Out of the station, he sees a welcome billboard made by the government of the city and he discovers the app for the taxi service. He brings his phone and search for it in the app store. Once downloaded he makes the request for signing up by telling his personal information. The system accepts his registration and he can log into the system. Once opened the app he makes the request for the taxi service by entering his position and in few minutes, he receive the confirmation with the code and the waiting time. In this way, Jack could reach the most famous museum in the city without had to stop a taxi on the road.

- Tom has been driving taxi for 20 years and recently has joined this solution. Tom has improved his work because he has much less periods of inactivity between the rides since he joined this solution. Tom is in the middle of the morning when it receive a call by the system on his dedicated smartphone. It appears like a message with the position of the caller, only two blocks from his. He accept the request and after 5 minutes of drive, he see a person who wants to request a taxi. He is Jack, the actor of the first scenario.

- It's one o'clock and Tom needs a rest in a restaurant. Before parking his taxi, he turns his availability switcher from on to off in order to inform the system that he is no more in service. Once finished his break he return in service and he has to turn on the availability in order to receive the requests of the afternoon.

- Jack, yet registered into the system, require a taxi service from his hotel. The system receive the request and contacts the first available taxi on the queue associated with the zone of the hotel. The fist available taxi is driven by Luke but is stuck in traffic. Luke, with his experience, knows that the time to reach the customer can be very long so he deny the request. The system forwards the request to the second taxi in the queue and move the Luke's taxi in the tail of the queue. The second taxi is driven by Tony, which is near the hotel, so he confirms the request and goes to take the customer. Once Jack came on board into the taxi speaks to the driver about the destination. This destination is just outside the city and the driver accepts the run (not always drivers are available to extra urban routes). When Tony has completed the service and let down the user, he had to return to the city in order to turn on his availability.

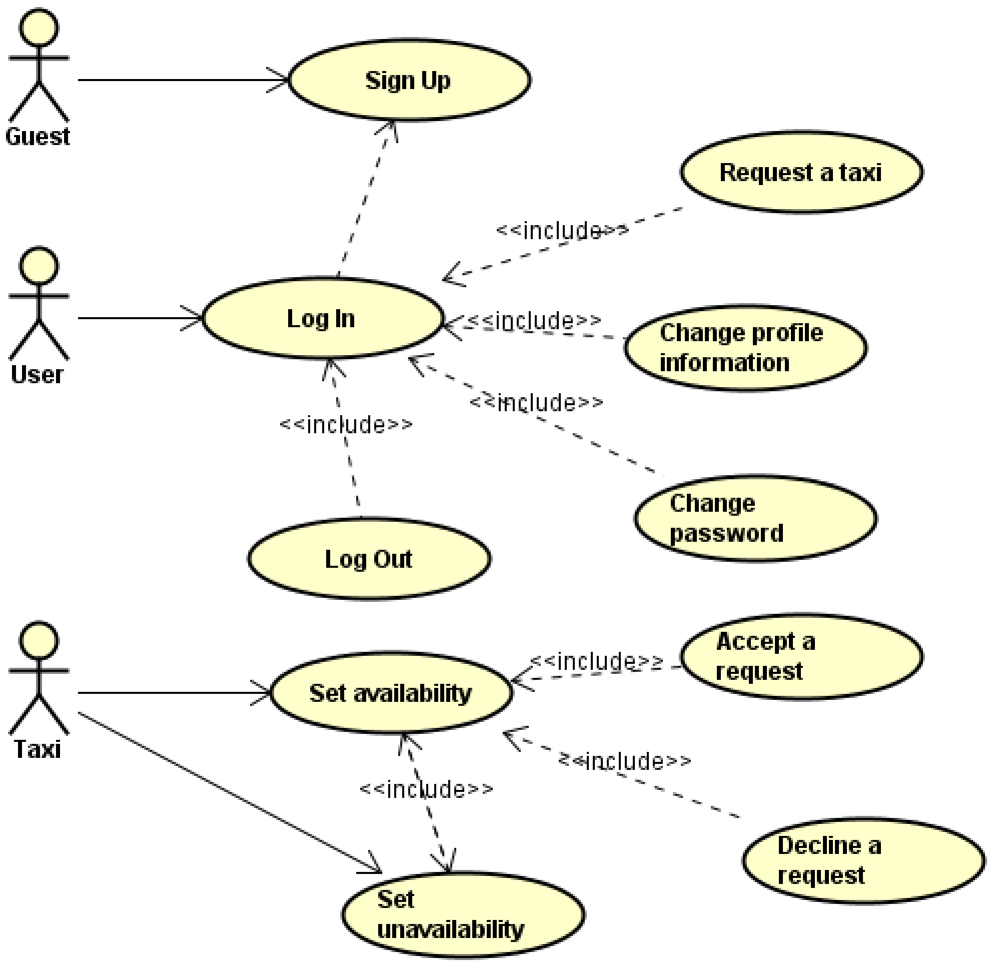
5. UML Models

5.1 Use Case Diagram

I can derive some use cases from the scenarios identified in the previous paragraph:

- Sign up;
- Log in;
- Log out;
- Request a taxi;
- Change profile information;
- Change password
- Accept a request/ Decline a request;
- Set availability/ Set unavailability;

Here it is a Representation of the Use Case Diagram



5.2 Use Case Description

I describe in a detailed way the use cases that I derived from the scenarios.

I try to define them all.

It is important to understand that all the references to “tabs”, “buttons” or “textboxes” are only hypothesis to make the situation as clear as possible and to help the reader to draw a visual picture in his mind of what I plan to do, but the real structures will be well defined in the Design Document

I refine the use case “Sign up”:

Name	Sign up
Actors	Guests
Entry Conditions	The guest isn’t registered to the system
Flow of events	• The guest enters the app/website • The guest clicks on the “Sign Up” button • The guest fills in the form where he has to write: ○ Email ○ Password ○ Confirm password ○ First name ○ Second name ○ Phone number • The guest clicks the “Sign Up” button • The system shows him the Log In tab
Exit conditions	• Registration successfully done • The system shows the user the login tab
Exceptions	An exception can be caused: • If the email the guest inserts already exists • If the phone number the guest inserts already exists (if it is inserted) • Some required field is not filled • The password is shorter than 8 chars • The confirming password is not matching the password

I refine the use case “Log In”:

Name	Log In
Actors	Users
Entry Conditions	The user has successfully signed up to the system
Flow of events	• The user enters the app/website • The user fills in the textboxes with email (or phone number) and password • The user clicks on the “Log In” button
Exit conditions	The system shows the user the main tab
Exceptions	An exception can be caused: • The email inserted is wrong (it doesn’t match with any yet registered emails) • The phone number inserted is wrong (it doesn’t match with any yet registered phone number) • First or second textbox is empty (or both) • The password inserted is not matching the one associated with the account

I refine the use case “Log Out”:

Name	Log Out
Actors	Users
Entry Conditions	The user has successfully logged in to the system
Flow of events	The user clicks on the “Logout” link
Exit conditions	The system shows the user the login tab
Exceptions	No exceptions

I refine the use case “Request a taxi”:

Name	Request a taxi
Actors	Users
Entry Conditions	The user has successfully logged in to the system
Flow of events	• The user types in the address where he wants to picked up by the taxi • The user clicks on the button “Request a taxi service” • The button will become temporarily greyed out (not clickable) • The user receives a response that the order has been taken in charge • The user receives a response with the code of the taxi and the waiting time
Exit conditions	The system remains on that tab and the button will return clickable
Exceptions	• The system runs out of taxis • The address is not in the city

I refine the use case “Change profile information”:

Name	Change profile information
Actors	Users
Entry Conditions	The user has successfully logged in to the system
Flow of events	• The user clicks on the “Edit profile data” link • The user fills in the textboxes with the new information • The user clicks on the “Change user information” button
Exit conditions	The system shows the user the login tab
Exceptions	• The email or the telephone number is already taken • All fields are empty

I refine the use case “Change password”:

Name	Change password
Actors	Users
Entry Conditions	The user has successfully logged in to the system
Flow of events	• The user clicks on the “Edit profile data” link • The user fills in the textboxes with the old password, the new password and a confirmation of the new password • The user clicks on the “Change password”
Exit conditions	The system shows the user the login tab
Exceptions	• Old password does not match with the current password • New password is shorter than 8 chars • Confirm new password does not match with new password • Missing fields

I refine the use case “Accept a request”:

Name	Accept a request
Actors	Taxi drivers
Entry Conditions	The taxi is available
Flow of events	• The driver receive a request • The driver accept the request by pressing the “Yes” button on the message
Exit conditions	The taxi becomes unavailable
Exceptions	No exceptions

I refine the use case “Decline a request”:

Name	Decline a request
Actors	Taxi drivers
Entry Conditions	The taxi is available
Flow of events	• The driver receive a request • The driver decline the request by pressing the “No” button on the message
Exit conditions	No exit conditions
Exceptions	No exceptions

I refine the use case “Set availability”

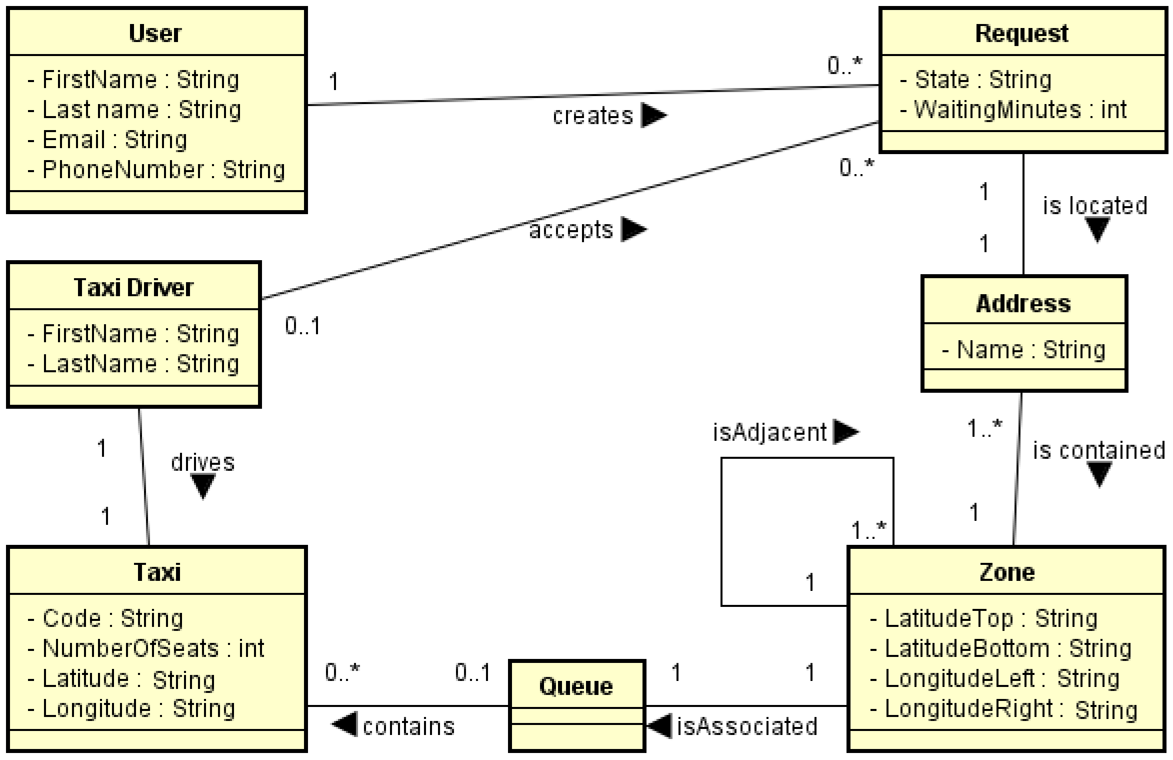
Name	Set availability
Actors	Taxi drivers
Entry Conditions	The taxi is unavailable
Flow of events	The driver turn on the availability switch
Exit conditions	The taxi becomes available
Exceptions	The taxi is outside the city

I refine the use case “Set unavailability”

Name	Set unavailability
Actors	Taxi drivers
Entry Conditions	The taxi is available
Flow of events	The driver turn off the availability switch
Exit conditions	The taxi becomes unavailable
Exceptions	No exceptions

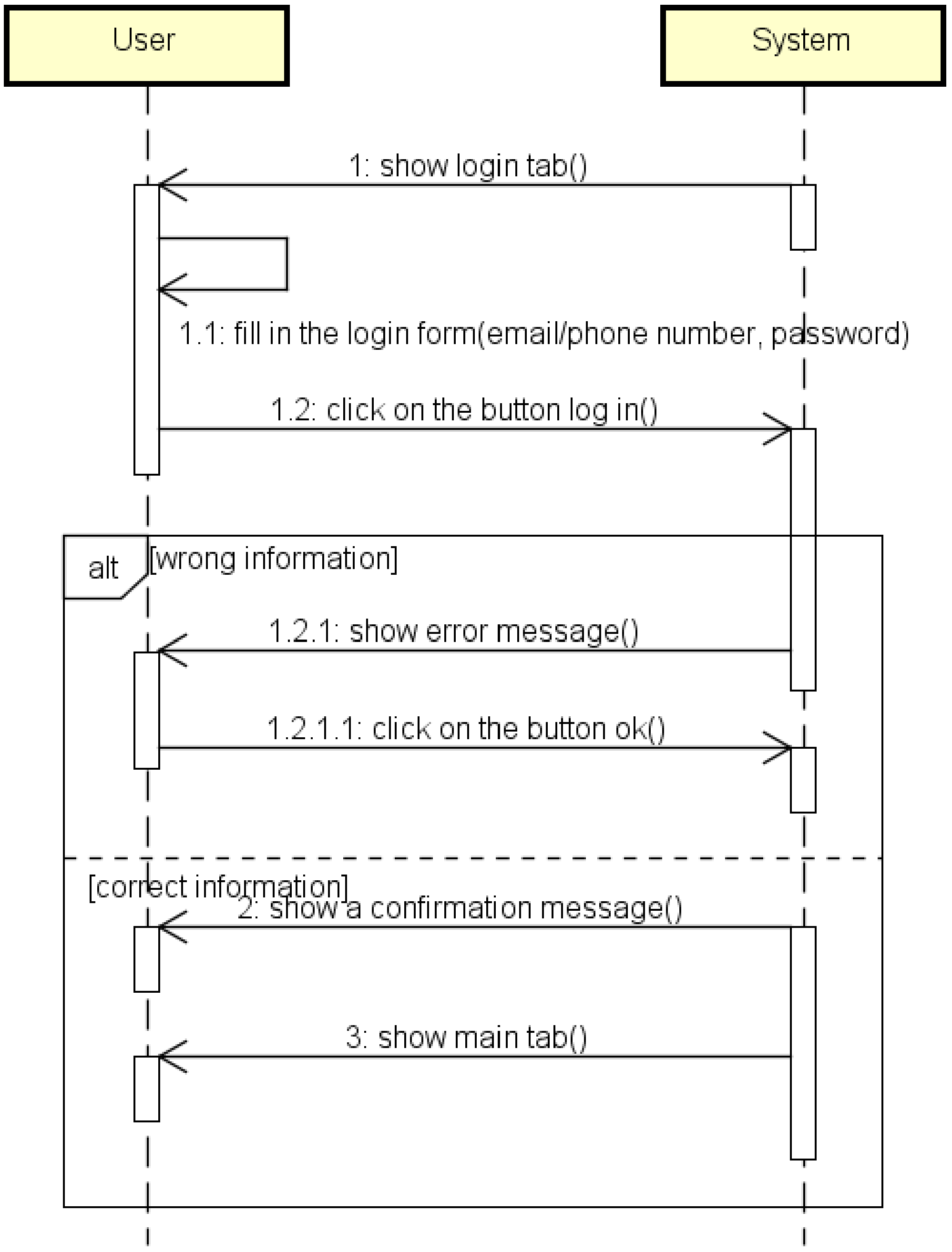
5.3 Class Diagram

Now that I have refined every use case, I can draw an initial sketch of the class diagram of our system:

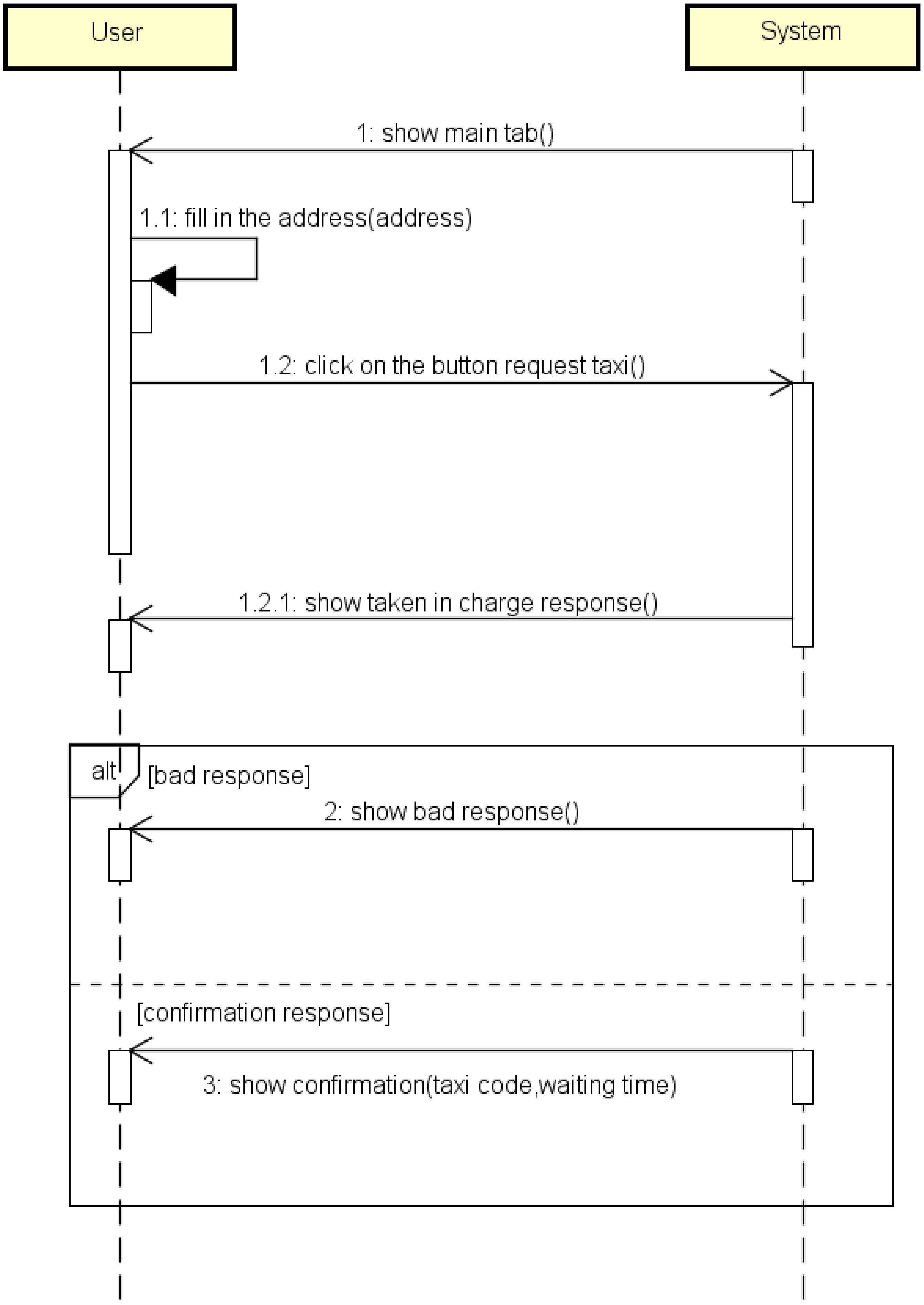


5.4 Sequence Diagrams

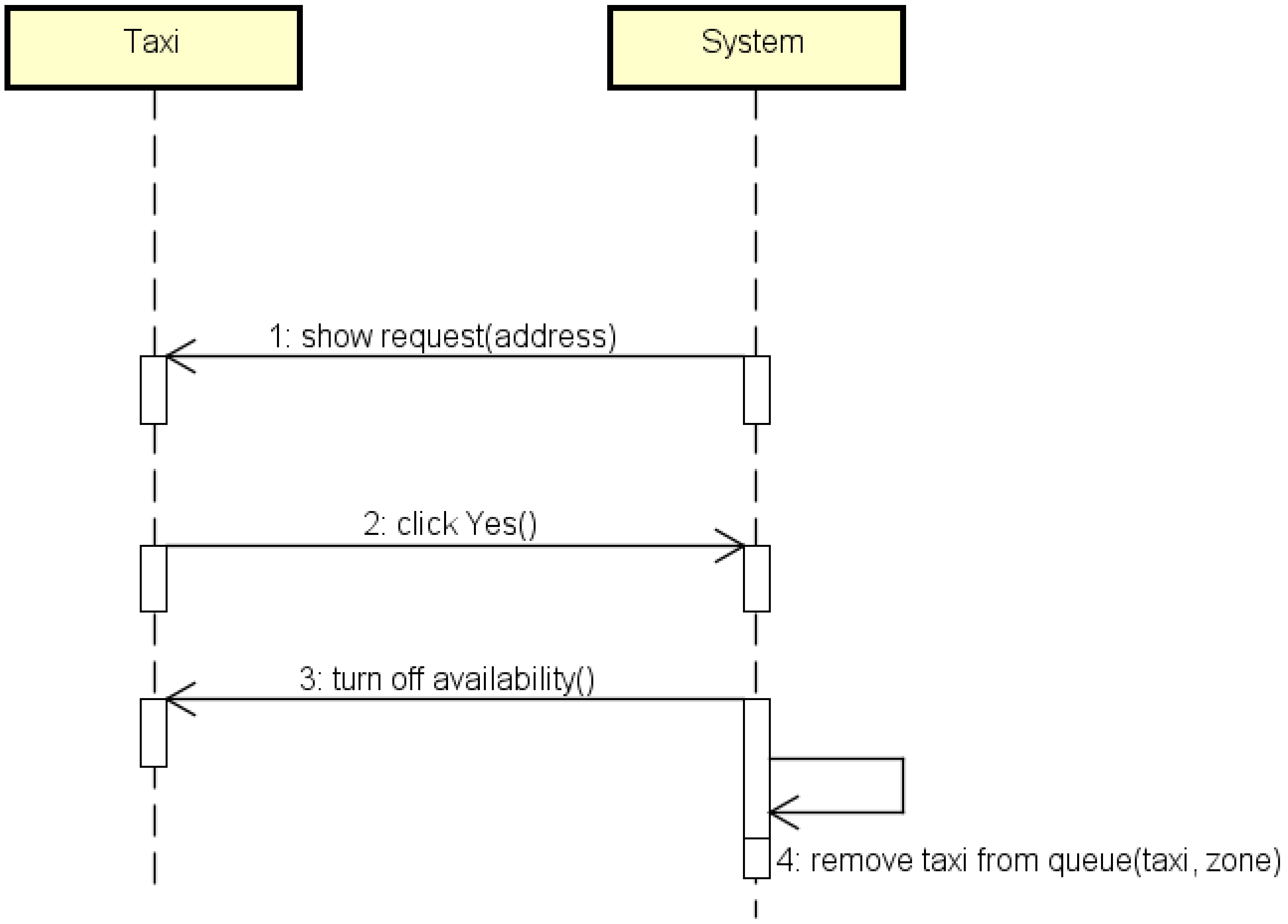
5.4.1 Log In


User
System
1: show login tab()
1.1: fill in the login form(email/phone number, password)
1.2: click on the button log in()
alt
[wrong information]
1.2.1: show error message()
1.2.1.1: click on the button ok()
[correct information]
2: show a confirmation message()
3: show main tab()

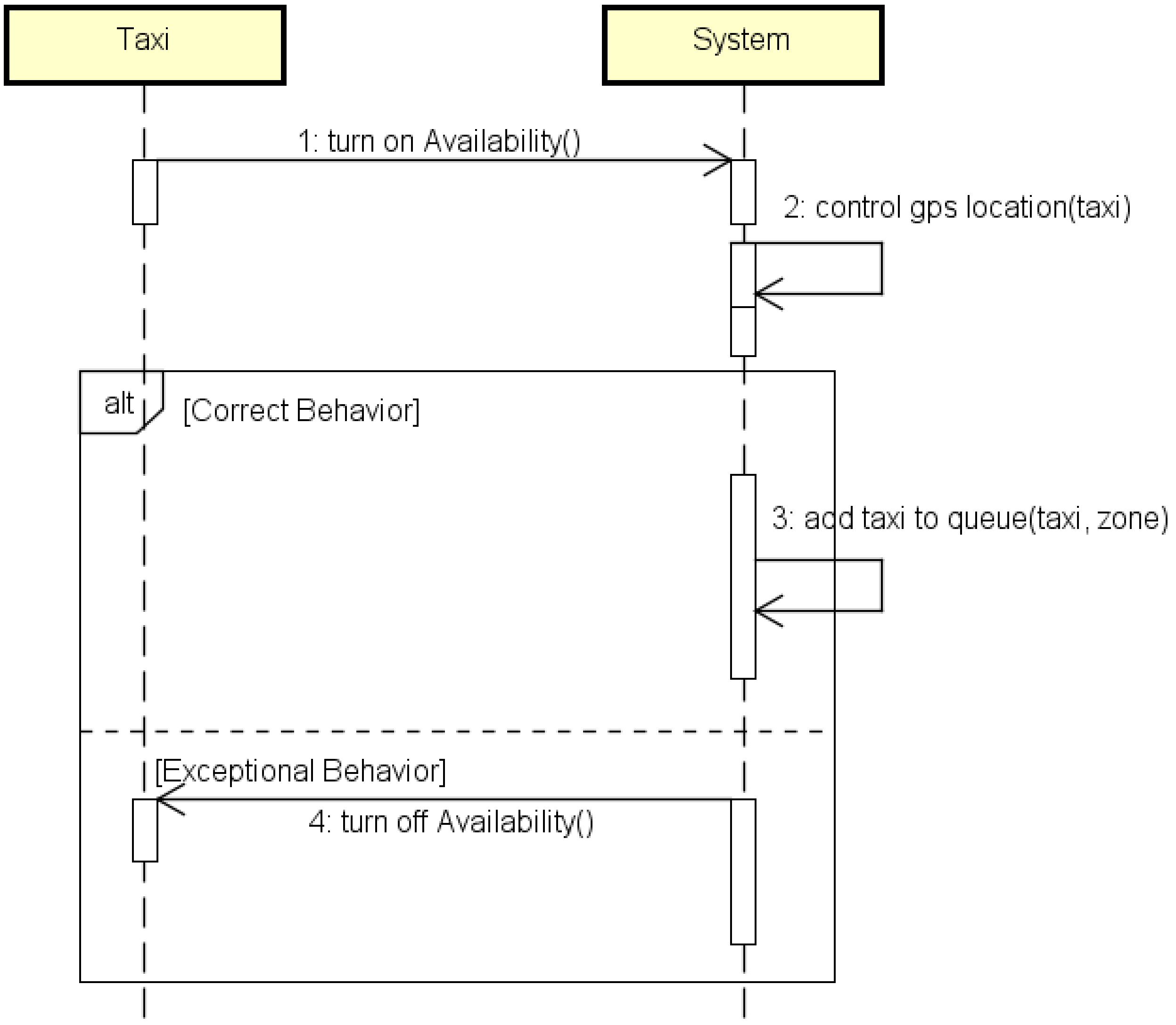
5.4.2 Request a taxi


User
System
1: show main tab()
1.1: fill in the address(address)
1.2: click on the button request taxi()
1.2.1: show taken in charge response()
alt
[bad response]
2: show bad response()
[confirmation response]
3: show confirmation(taxi code,waiting time)

5.4.3 Accept a request

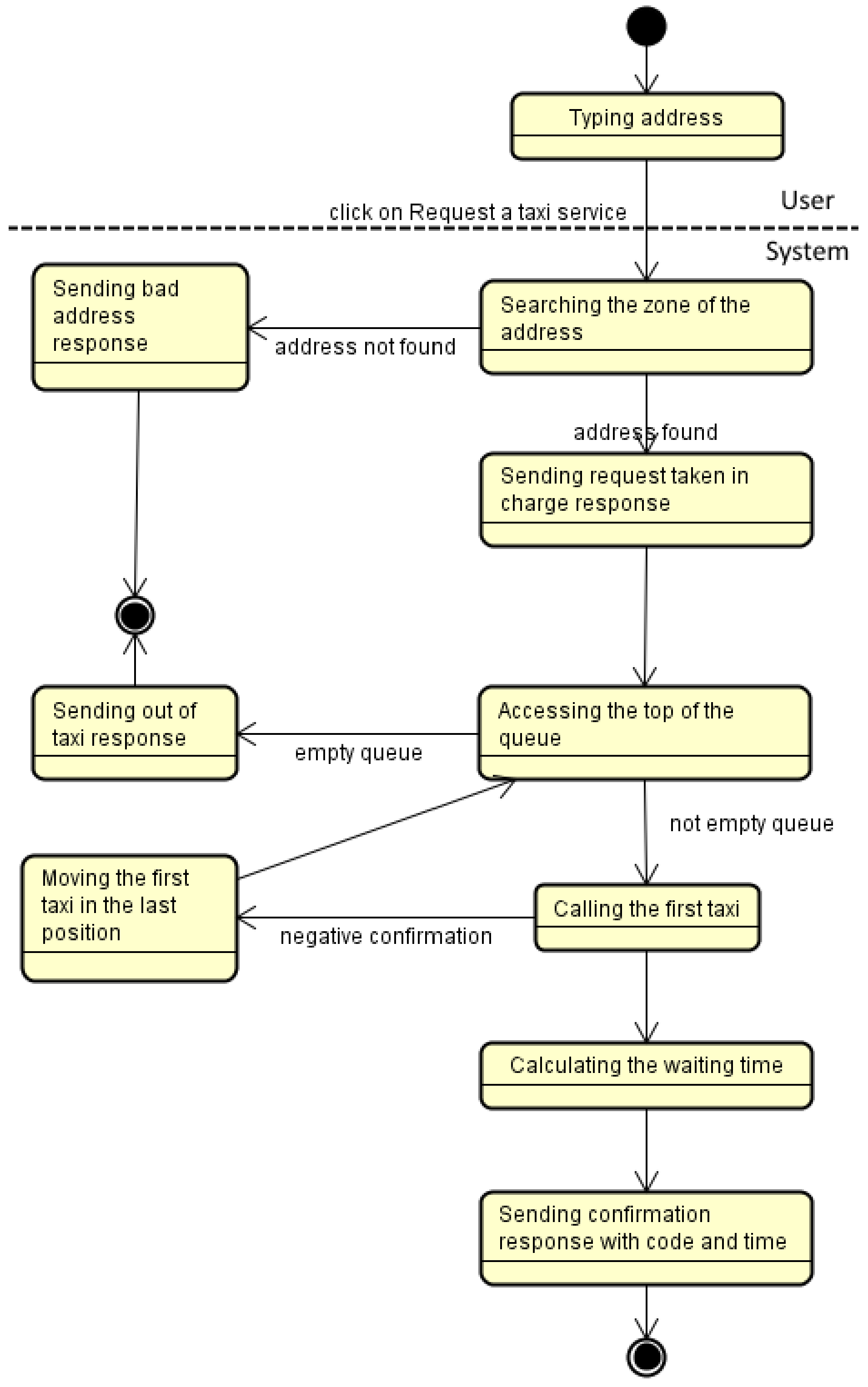

Taxi
System
1: show request(address)
2: click Yes()
3: turn off availability()
4: remove taxi from queue(taxi, zone)

5.4.4 Change availability


Taxi
System
1: turn on Availability()
2: control gps location(taxi)
alt
[Correct Behavior]
3: add taxi to queue(taxi, zone)
[Exceptional Behavior]
4: turn off Availability()

5.5 State Chart Diagram

5.5.1 Request


Typing address
click on Request a taxi service
User
System
Searching the zone of the address
address not found
Sending bad address response
address found
Sending request taken in charge response
Accessing the top of the queue
empty queue
Sending out of taxi response
not empty queue
Moving the first taxi in the last position
Calling the first taxi
negative confirmation
Calculating the waiting time
Sending confirmation response with code and time

6. Alloy Modelling

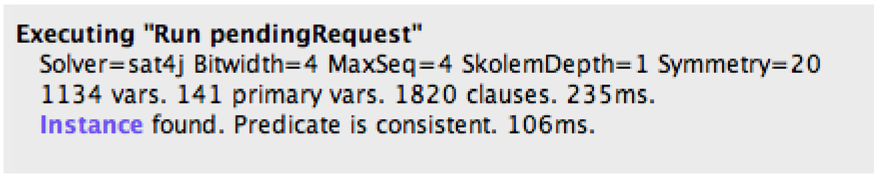
I have used Alloy Analyser to specify the properties of our application, starting from the representation of the Class Diagram. In this paragraph I have reported the alloy code used for the analysis.

6.1 Alloy Code

```
//signatures
sig User
{
	creates:set Request
}

sig Request
{
	isLocated: one Address
}

sig Address
{
	isContained:one Zone
}

sig Taxi
{
}

sig TaxiDriver
{
	drives:one Taxi,
	accepts:set Request,
	NumberOfSeats:Int
}

sig Zone
{
	isAdjacent:set Zone,
	isAssociated:one Queue
}

sig Queue
{
	contains:some Taxi
}
```

```
//FACTS
fact TaxiInOnlyOneQueue
{
	//Taxi can't be contained in two different queues
	all t:Taxi|(no disj q1,q2: Queue|(t in q1.contains and t in q2.contains))
}

fact RequestAcceptedByOnlyOneTaxiDriver
{
	//Request can't be accepted from two different taxi drivers
	all r:Request|(no disj td1,td2: TaxiDriver |(r in td1.accepts and r in td2.accepts))
}

fact RequestMadeByOneAndOnlyOneUser
{
	//Request Made By One And Only One User
	(all r:Request|(one u: User|(r in u.creates)))
}

fact zoneNotAdjacentToItself
{
	//A zone can’t be adjacent to itself
	no z:Zone|(z in z.isAdjacent)
}

fact zoneAdjacentSymmetricRelation
{
	//If a zone is adjacent to another it must be true also in reverse
	all z1:Zone|(all z2:Zone|(z2 in z1.isAdjacent =>(z1 in z2.isAdjacent)))
}


//PREDICATES
pred pendingRequest
{
	//Show a world in which there is request that is waiting a taxi
	one r:Request|(no t:TaxiDriver|r in t.accepts)
}

run pendingRequest

//Show a world in which there is zone that runs out of taxi
pred zoneOutOfTaxi
{
	one q:Queue|(no t:Taxi|t in q.contains)
}

run zoneOutOfTaxi
```

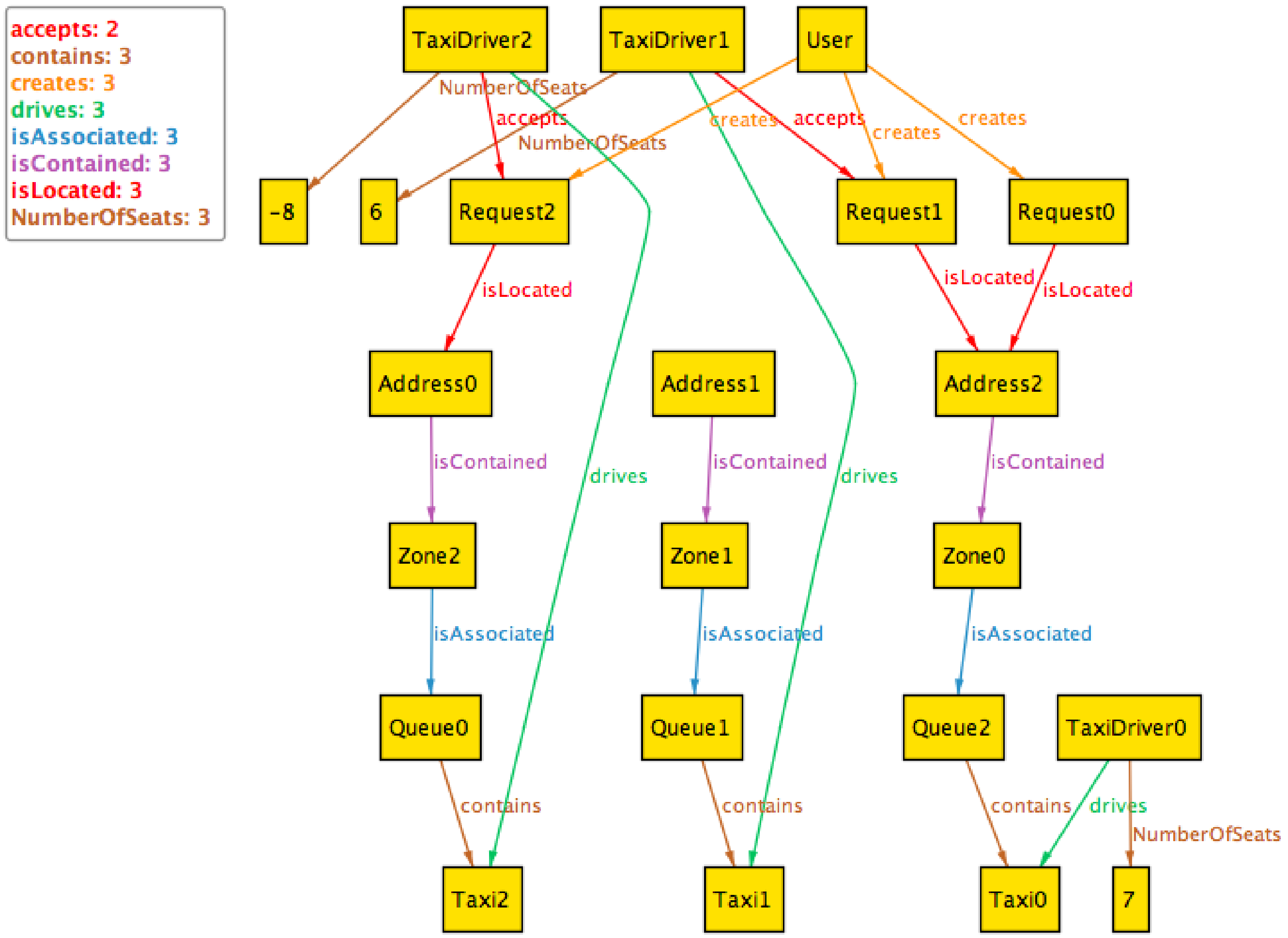
Here result of the analysis.



```
Executing "Run pendingRequest"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   1134 vars. 141 primary vars. 1820 clauses. 235ms.
   Instance found. Predicate is consistent. 106ms.
```

6.2 Alloy Worlds

6.2.1 General world



7. Used tools

The tools I used to create the RASD document are:

• Microsoft Office Word 2013: to redact and to format this document;

• Astah: to create the State Charts, the Class Diagram, the Sequence Diagrams and the Use Case Diagram;

• Alloy Analizer 4.2: to prove the consistency of our model.

• moqups.com: to create the sketch of the applications.

• Paint.NET: to adjust the images.

For redacting and writing this document I spent about 30 hours.