



Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: "MyTaxiService"

# **Requirement Analysis and Specifications Document**

Giuseppe Canonaco, Andrea Di Giosaffatte

6 November 2015

# **Table of Contents**

## **1. Introduction**

- 1.1.Purpose
- 1.2.Scope
- 1.3.actors
- 1.4.Goals
- 1.5.Definitions, Acronyms, Abbreviations
  - 1.5.1 Definitions
  - 1.5.2 Acronyms
  - 1.5.3 Abbreviations
- 1.6.Reference Documents
- 1.7.Document Overview

## **2. Overall Description**

- 2.1.Product Perspective
  - 2.1.1. User Interfaces
  - 2.1.2. Software Interfaces
  - 2.1.3. Communication Interfaces
- 2.2.User Characteristics
- 2.3.Assumptions and Dependencies

## **3. Specific Requirements**

- 3.1.Functional Requirements
- 3.2.Non Functional Requirements
- 3.3.Scenarios
- 3.4.UML Models
  - 3.4.1. Use Case Diagram
    - 3.4.1.1. Use Case Description
  - 3.4.2. Sequence Diagrams
  - 3.4.3. Class Diagram
- 3.5. Alloy Model
- 3.6. Used Tools

# 1. Introduction

## 1.1. Purpose

This document addresses the requirement analysis of the application MyTaxiService, which will be developed in order to optimize the taxi service of a large city. No previous version of this application have been developed. The intended audience is formed by the developers of the software, the financial stakeholders and the end users.

## 1.2. Scope

MyTaxiService is an application which will improve the taxi service of a large city both from the taxi drivers and passengers point of view. Passengers will be able to request or reserve a taxi by using the web application or the mobile app. The request is done at the very moment the passenger needs a cab, instead the reservation is done when a passenger knows in advance that he will need to take a cab. On the other hand taxi drivers will be scheduled by the system to the various taxi queues within the city according to the position in which they are when they become available. This will turn out into a more efficient way to handle the queues and a smother access to the service itself by the passengers.

## 1.3. Actors

**USER:** is either a cab catcher or a cab driver.

**CAB CATCHER:** is someone, which is already registered to MyTaxiService system, who may to take a cab via a request or a reservation.

**CAB DRIVER:** is someone which has a taxi license released by the town council and is already registered to the MyTaxiService system.

**API BASED SYSTEM:** is a third part system, which uses the basic functionalities provided by the MyTaxiService system. These functionalities allow the API BASED SYSTEM to request or reserve a taxi cab on behalf of a MyTaxiService cab catcher. More over through them the API BASED SYSTEM can also delete a reservation or a request previously done.

## 1.4. Goals

The system must assure the subsequent goals:

**[G1]** Allow the users to use the service.

**[G2]** Simplify the access of passengers to the taxi service.

**[G3]** Guarantee a fair management of taxi queues.

**[G4]** Offer programmatic interfaces to enable the development of additional services on top of the basic one.

## 1.5. Definitions, Acronyms, Abbreviations

### 1.5.1. Definitions

- **Programmatic Interface:** an object which allows a third part system to perform operations over the MyTaxiService system on behalf of a MyTaxiService cab catcher.
- **Arrangement:** either a request or a reservation.
- **Pending Request:** a request which has not been accepted yet by any cab driver.
- **Application:** by this term we refer both to the mobile application and the web application.
- **Reservation:** the object which contains information about a cab booking, whose ride will be handled in the future.
- **Request:** the object which contains information about a cab booking, whose ride will be handled immediately.

### 1.5.2. Acronyms

- **API:** application program interface.

### 1.5.3. Abbreviations

- **[Gn]** The n<sup>th</sup> goal.
- **[An]** The n<sup>th</sup> dependency.
- **[Dn]** The n<sup>th</sup> domain assumption.
- **[Rm.n]** The n<sup>th</sup> functional requirement related to the m<sup>th</sup> goal.
- **[NFRn]** The n<sup>th</sup> non-functional requirement.
- **[UIn]** The n<sup>th</sup> use case diagram.
- **[SDn]** The n<sup>th</sup> sequence diagram.

## 1.6. Reference Documents

- Assignments 1 and 2 (RASD and DD) which can be retrieved on the beep page of the course.
- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, which can be retrieved on the beep page of the course.

## 1.7. Document Overview

- **Introduction:** this section gives a brief explanation of the system to be and the actors involved in interactions with it. More over the introduction provides both the basic knowledge and the referenced documents to properly keep on reading the document itself.
- **Overall Description:** in this section it will be provided a solid background for the requirement so that they become easier to understand.
- **Specific Requirements:** this section contains all the software requirements associated with UML diagrams which make the requirements themselves easier to understand.

# 2. Overall Description

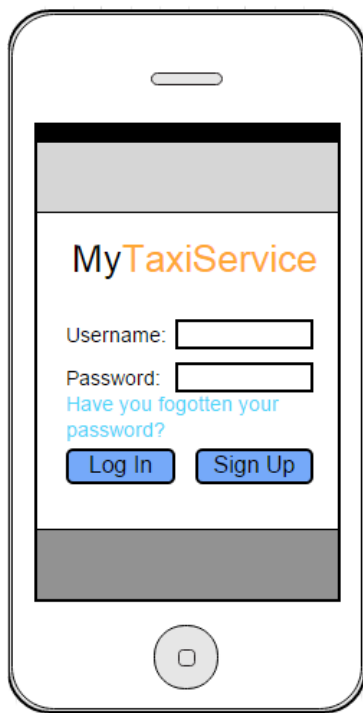
## 2.1. Product Perspective

The system to be is integrated with the municipal database from which it will be retrieved all the information related to the taxi drivers. The municipal database grants access only to the taxi drivers information.

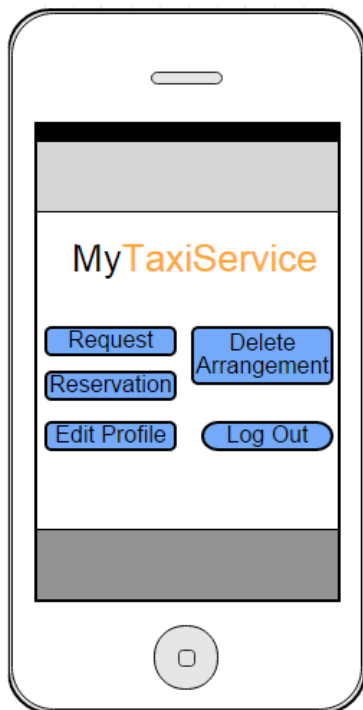
### 2.1.1. User Interfaces

Just to provide a better understanding of what the principal interfaces may be, we displayed them in this section. The choice of representing only the mobile version of the interfaces is due to the fact that the web application interfaces are the same, at least in the essential core.

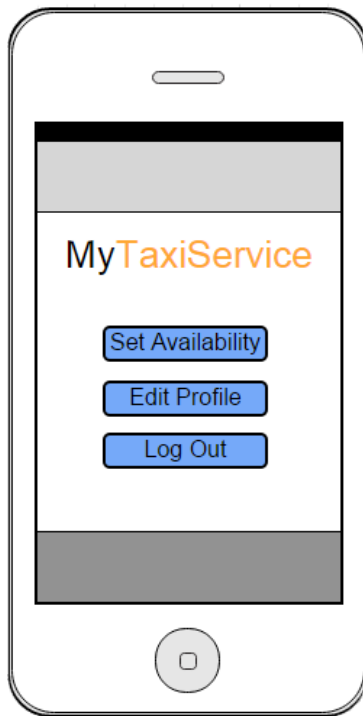
**[UI1]** User Log In.



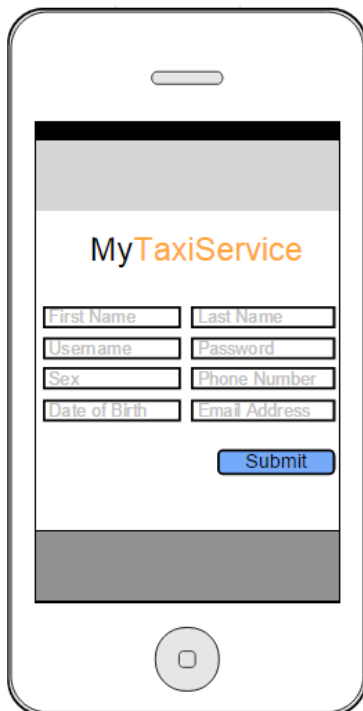
**[UI2]** Cab Catcher Home Page.



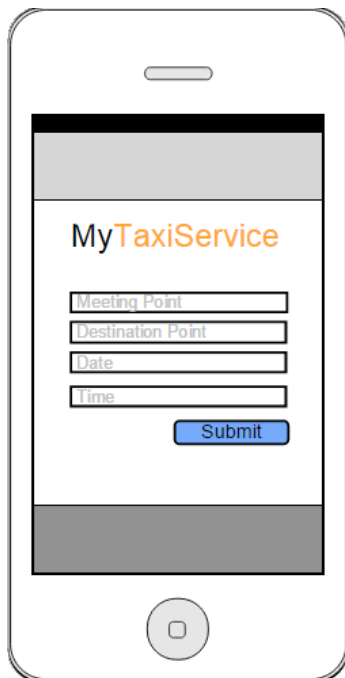
**[UI3]** Cab Driver Home Page.



**[UI4]** Cab Catcher Sign Up.



**[UI5]** Cab Catcher Reservation.



The image shows a mobile app interface for "MyTaxiService". The app has a white background with a grey header and footer. The main content area contains the app name "MyTaxiService" in orange and black text. Below the name are four input fields: "Meeting Point", "Destination Point", "Date", and "Time". A blue "Submit" button is located below the input fields. The app is shown on a white smartphone with a grey home button.

MyTaxiService

Meeting Point

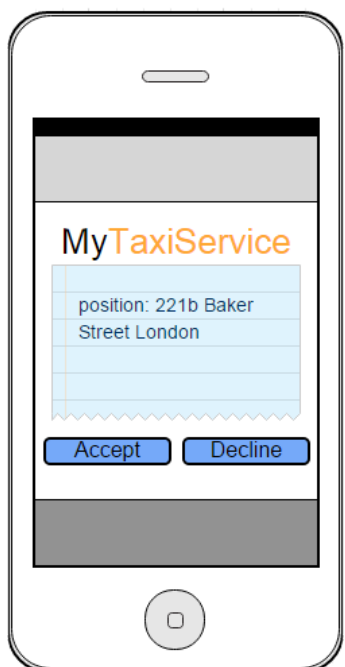
Destination Point

Date

Time

Submit

**[UI6]** Notification of a Pending Request.



The image shows a mobile app interface for "MyTaxiService". The app has a white background with a grey header and footer. The main content area contains the app name "MyTaxiService" in orange and black text. Below the name is a light blue notification box with a wavy bottom edge. The notification text reads "position: 221b Baker Street London". Below the notification box are two blue buttons: "Accept" and "Decline". The app is shown on a white smartphone with a grey home button.

MyTaxiService

position: 221b Baker  
Street London

Accept Decline



### 2.1.2. Software Interfaces

- Municipal Database Management System (Municipal DBMS):
  - Name: MySQL
  - Version: 5.6.21
  - Source: <http://www.mysql.it/>
- Google maps APIs to build a location-aware app and to show the shortest path to a certain location:
  - Name: Google maps APIs
  - Source: <https://developers.google.com/maps/>

### 2.1.3. Communication Interfaces

Protocol	Application	Port
TCP	HTTPS	443
TCP	HTTP	80
TCP	DBMS	3306 (default)

## 2.2. User Characteristics

We expect three kind of user to access our system:

- The cab catcher who only needs to know how to use a browser or a mobile phone.
- The cab driver who only needs to know how to use a browser or a mobile phone.
- The API based system must follow the designated protocol to properly use the API itself.

## 2.3. Assumptions and Dependencies

### Dependencies:

- **[A1]** MyTaxyService system relies on the municipal data base on which there are stored the accounts of all the taxi drivers.

### Assumptions:

- **[D1]** Cab Drivers are already registered to the service.
- **[D2]** The e-mail of each user is supposed to be currently in use.
- **[D3]** The phone number of each user is supposed to be currently in use.
- **[D4]** Each taxi driver has a smartphone in which the application can run.

- **[D5]** Each taxi has a unique code.
- **[D6]** The city is divided in taxi zones.
- **[D7]** Each taxi queue is assigned to a zone.
- **[D8]** When available each taxi is assigned to a taxi queue.
- **[D9]** There is always a cab driver which will handle a request from a cab catcher.
- **[D10]** There are always available taxies capable of reaching the origin of a drive associated to a reservation in no more than 10 minutes.
- **[D11]** A cab driver always takes care of a reservation request when proposed by the system.

Reasons which support some assumptions:

- **[D1]:** This assumption is supported by the fact that the taxi licenses are released by the town council, so is the town council itself which gives an account on the system to a person who has just received or has a license.
- **[D10]:** This assumption is supported by the fact that the city is divided into zones of 2km<sup>2</sup> so when every queue is not empty even if there's a lot of traffic the amount of road to travel will be very small. During day the situation just described is pretty realistic. If we think of the night, when we have lots of empty queue, the traffic will be very low so we can travel lots of road in a very small amount of time.

## 3. Specific Requirements

### 3.1. Functional Requirements

**[G1]** Allow the users to use the service.

- **[R1.1]** The system has to provide a Log In functionality to allow users to log into the service.
- **[R1.2]** The system has to provide a Password Recovery functionality to allow users to recover their password.
- **[R1.3]** The system has to provide an Edit Profile functionality to allow users to modify their profile information.
- **[R1.4]** The system has to provide a Change Password functionality to allow users to modify their password.
- **[D1]** Cab Drivers are already registered to the service.
- **[D2]** The e-mail of each user is supposed to be currently in use.

- **[D3]** The phone number of each user is supposed to be currently in use.

**[G2]** Simplify the access of passengers to the taxi service.

- **[R2.1]** The system has to provide a Sign Up functionality, which allows cab catchers to join the service.
- **[R2.2]** The system has to provide a Taxi Request functionality, which lets cab catchers to make an immediate request for a taxi.
- **[R2.3]** The system has to provide a Taxi Reservation functionality, which allows cab catchers to book a taxi.
- **[R2.4]** The system has to provide a Delete Undesignated Arrangement functionality, which allows cab catchers to delete an arrangement that is still pending.
- **[R2.5]** The system has to provide a Delete Designated Arrangement functionality, which allows cab catchers to delete an arrangement that has been assigned to a cab driver.
- **[R2.6]** The system has to be capable of estimating the arrival time of the cab driver to the meeting point.
- **[R2.7]** The system has to inform the cab catcher of the code by which he can recognize its cab.
- **[R2.8]** The system has to inform the cab catcher of the estimated arrival time.
- **[D4]** Each taxi has an integrated GPS system.
- **[D5]** Each taxi has a unique code.
- **[D6]** The city is divided in taxi zones.
- **[D7]** Each taxi queue is assigned to a zone.
- **[D8]** When available each taxi is assigned to a taxi queue.
- **[D9]** There is always a cab driver which will handle a request from a cab catcher.
- **[D10]** There are always available taxies capable of reaching the origin of a drive associated to a reservation in no more than 10 minutes.
- **[D11]** A cab driver always takes care of a reservation request when proposed by the system.

**[G3]** Guarantee a fair management of taxi queues.

- **[R3.1]** The system has to provide a Set Availability functionality, which allows cab drivers to inform the system about their availability.
- **[R3.2]** The system has to provide a Decline Request functionality, which allows cab drivers to decline a taxi request performed by a cab catcher.
- **[R3.3]** The system has to provide an Accept Request functionality, which allows cab drivers to accept a taxi request performed by a cab catcher.
- **[D4]** Each taxi has an integrated GPS system.
- **[D5]** Each taxi has a unique code.
- **[D6]** The city is divided in taxi zones.
- **[D7]** Each taxi queue is assigned to a zone.
- **[D8]** When available each taxi is assigned to a taxi queue.
- **[D9]** There is always a cab driver which will handle a request from a cab catcher.
- **[D10]** There are always available taxis capable of reaching the origin of a drive associated to a reservation in no more than 10 minutes.
- **[D11]** A cab driver always takes care of a reservation request when proposed by the system.

**[G4]** Offer programmatic interfaces to enable the development of additional services on top of the basic one.

- **[R4.1]** The system has to provide an interface, which allows third part systems to perform operations over the basic one.

## **3.2. Non Functional Requirements**

- **[NFR1]** The web application must run at least on the following browsers: Safari, Chrome, Opera, Firefox, Internet Explorer.
- **[NFR2]** The mobile application must run at least on the following browsers: iOS, Android, Windows Phone.
- **[NFR3]** The Cab Catcher can't have more than one active request at a time.

- **[NFR4]** Each reservation has to occur at least two hours before the ride.
- **[NFR5]** The code sent by the system during the password recovery procedure is valid for 40 seconds starting from the moment in which the code is generated.
- **[NFR6]** The system, in case of reservation, has to allocate a taxi to that arrangement 10 minutes before the meeting time with the cab catcher.
- **[NFR7]** The system considers as refused a request notification which is not answered by a cab driver within 1 minute.
- **[NFR8]** The system should be available 97% of time over a year.

### 3.3. Scenarios

Here are some possible scenarios of MyTaxiService:

#### **[SC1]** New user's registration.

Sasha, which is fond of travelling, spends lots of time in searching for taxis when she needs to get to the airport. Recently she heard from a friend that a new taxi booking application service was delivered by the government of her town. So she decided to try this new application considering her next trip which is going to be in less than a few days. She reaches the web site from her default browser and starts the sign-up procedure which consist of fulfilling a form with her first name, last name, date of birth, sex, e-mail, username, password and phone number. She concludes the procedure successfully and confirms her account by clicking on the link contained in the confirmation email which automatically logs her in.

#### **[SC2]** User profile editing.

Samuel wants to edit his password so he logs into the MyTaxiService application. He goes into the profile editing section in which he choose to edit his password. The editing function asks him for the old password and the new one. He enters them and completes the procedure by clicking on the submit button.

#### **[SC3]** Taxy request.

Tony, which is already registered to the system, opens his mobile app which automatically logs him in. He has to reach a restaurant to catch up with his wife. He's terribly late so he decides to use the request functionality to make the system send him a taxi which can drive him to the restaurant. After sending the request, the application sends a text message to his phone number, specified into the profile, the information about the estimated arrival time and the taxi code by which he can recognize the car.

#### **[SC4]** Taxy Reservation.

Tomorrow morning Sarah has to do a business trip but unfortunately the gate closing time is too early to go by public transportation, so she decides to go by taxi cab. She connects her

laptop to the web and reaches the MyTaxiService portal by which she makes a reservation. To complete the reservation procedure she's asked to fulfill a form containing information about the meeting time, the origin and the destination of the ride. Ten minutes before the meeting time a text message containing the taxi code and the estimated arrival time is sent to her phone so she can recognize the assigned car.

**[SC5] Designated arrangement deletion.**

John has just received the text containing the cab code and the estimated arrival time of his drive to the station, but suddenly he receives a phone call. It's his boss who is telling him that the deadline for the presentation of the project John was working on was anticipated to the following day. Unfortunately the project is still to be finished so John has to cancel the arrangement with the cab which was already incoming. Hence he quickly takes his phone, opens his MyTaxiService app and makes the cancellation.

**[SC6] User password recovery.**

Mary wants to access the MyTaxiService web application so she reaches the web site and tries to log in. Unfortunately the authentication fails because the password is wrong. She decides to use the password recovery function to retrieve her password so that she can access the service again. The system asks the user to enter his registration mail then she clicks on the send code button, which makes the system send a code to her phone. Now the system asks the user to enter the received code by which she can change her password. She enters the code and the system generates a new password which replaces the old one. The system sends the new password to her phone number. She has successfully changed the password.

**[SC7] Cab driver request acceptance.**

Rick gets into his taxi cab and logs into the MyTaxiService application to start working. In order to be scheduled by the system he changes his account state to available. He reaches the queue to which was assigned by the application and starts to wait his turn. After about ten minutes he receives his first request which he decides to take care of. Hence his account state becomes not available and he starts driving towards the meeting location. While he's driving Rick receives a notification from the system which states that the request has just been cancelled switching his account state to available. Therefore Rick, which is pretty angry, starts driving back to the new assigned queue.

**[SC8] Cab driver reservation refusal.**

Andrew which has just completed a drive switches his account state to available and starts driving to the designated queue. By the time he arrives to the queue it's already his turn, in fact he receives a request notification which he decides to turn down because he has just noticed a flat tire. He puts his account state to unavailable and calls the mechanic.

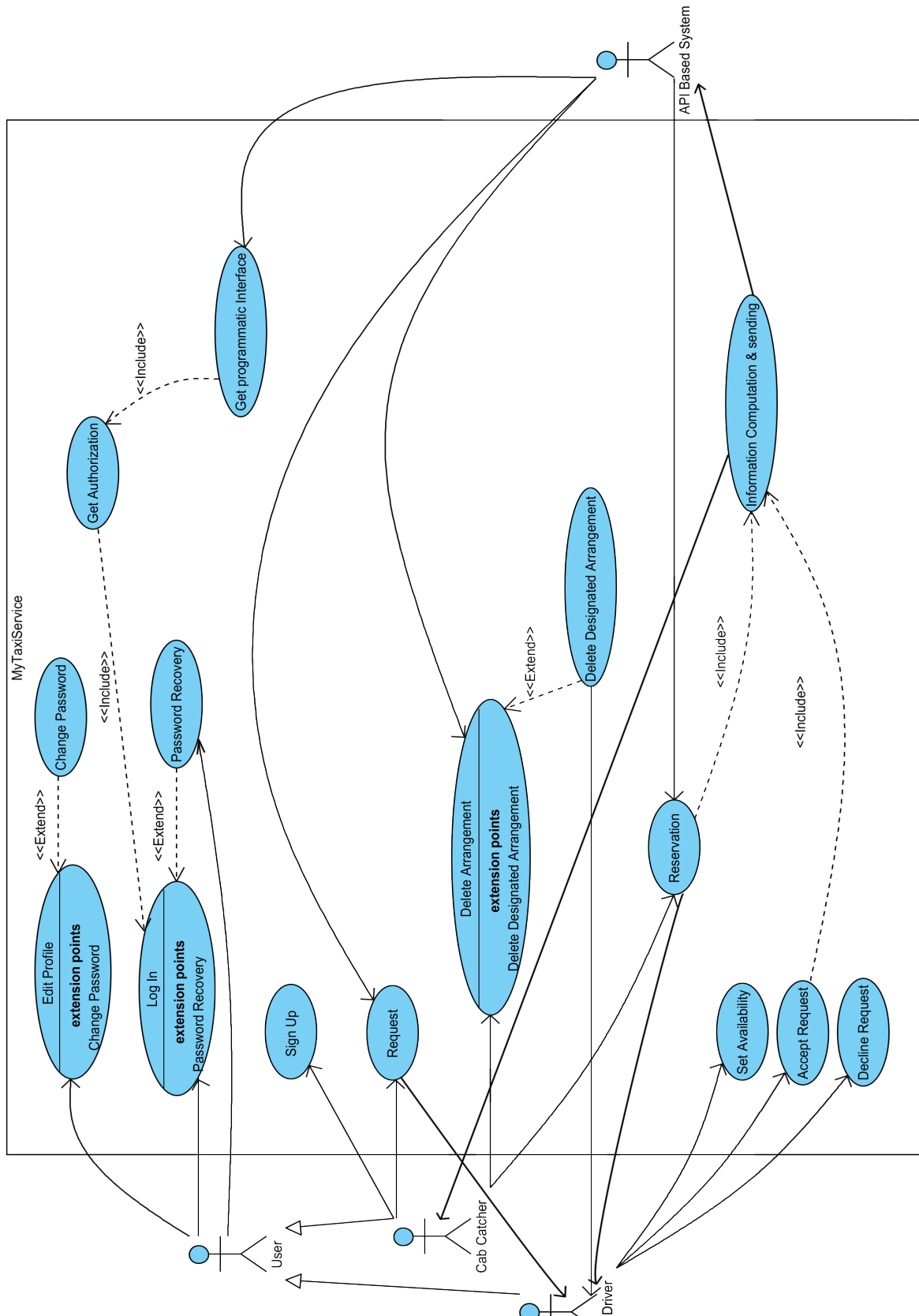
**[SC9] API based system request.**

The MyTaxiSharing application, which is based upon the MyTaxiService application, in order to complete a taxi sharing procedure, has to book the cab which will handle the shared ride. To do so, it uses the reservation functionality provided by the programmatic interface.

## 3.4. UML Models

### 3.4.1. Use Case Diagram

In the section below it's shown a representation of the Use Case Diagram:



### 3.4.1.1. Use Case Description

In the section below it's shown a detailed description of each use case:

#### [UC1] Sign Up.

NAME	<b>Sign Up</b>
ACTORS	Cab Catcher
ENTRY CONDITIONS	The Cab Catcher isn't registered to the MyTaxiService application
FLOW OF EVENTS	<ul style="list-style-type: none"><li>• The cab catcher open the application</li><li>• The system shows him the application home page</li><li>• He clicks on the sign up button</li><li>• The application shows him the sign-up page</li><li>• He fulfills the sign up form which requires the following information to be inserted:<ul style="list-style-type: none"><li>- First name</li><li>- Last name</li><li>- Username</li><li>- Password</li><li>- Sex</li><li>- Phone number</li><li>- Date of birth</li><li>- E-mail</li></ul></li><li>• He submits the form</li><li>• He confirms his account by clicking on a link contained in a confirmation e-mail</li></ul>
EXIT CONDITIONS	Account confirmation procedure successfully done
EXCEPTIONS	In case of missing mandatory information or username already in use or incorrect data then the system shows the cab catcher an error message highlighting the incorrect records

#### [UC2] Log In.

NAME	<b>Log in</b>
ACTORS	User
ENTRY CONDITIONS	Either the user is already registered into the system or an API based system needs to be authorized to execute operations on behalf of a cab catcher



FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The user opens the application</li> <li>• The system shows him the home page</li> <li>• He inserts his username and his password</li> <li>• He clicks the log in button</li> </ul>
EXIT CONDITIONS	Either the user successfully accesses to the service or an authorization is sent back to an API based system
EXCEPTIONS	The user has forgotten his password so the system asks him if he want to recover it by starting the <b>Password Recovery</b> use case

### [UC3] Password Recovery.

NAME	<b>Password Recovery</b>
ACTORS	User
ENTRY CONDITIONS	The user has forgotten his password and has clicked on the password recovery button
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The system shows the user the password recovery page</li> <li>• The user fulfills the email record</li> <li>• The user clicks on the send code button</li> <li>• The user receive the security code on his mobile phone</li> <li>• The system shows the user the show code checking page to allow him to change his password</li> <li>• The user fulfills the code record with the received code</li> <li>• The user submits it</li> <li>• The system generates a new password and replaces the old one</li> <li>• The system sends the new password to the user phone number</li> </ul>
EXIT CONDITIONS	The system send the new password to the user
EXCEPTIONS	If the user enters a non-registered mail or an invalid security code then the recovery procedure must be aborted by the system showing an error message. If there is a connection loss then the procedure is also aborted

**[UC4]** Edit Profile.

NAME	<b>Edit Profile</b>
ACTORS	User
ENTRY CONDITIONS	The user needs to change some information about his profile and clicks the edit profile button
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• if the user needs to change his account password then he clicks on the change password button which starts the <b>Change Password</b> use case</li> <li>• Otherwise, the user simply changes the information to be modified</li> <li>• The user submits the updated profile</li> </ul>
EXIT CONDITIONS	The system successfully stores the information the user has just changed
EXCEPTIONS	If the changed data are not valid the system shows the user an error message highlighting the invalid records

**[UC5]** Change Password.

NAME	<b>Change Password</b>
ACTORS	User
ENTRY CONDITIONS	While the user is editing his profile he clicks on the change password button
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The system shows the user the change password page</li> <li>• The user fulfills the records, the first one related to the old password and the second related to the new one</li> <li>• He clicks over the submit button to send the new information</li> </ul>
EXIT CONDITIONS	The system stores the new password by replacing the old one
EXCEPTIONS	If the old password is wrong or the new one is invalid the system shows the user an error message highlighting the wrong record

**[UC6]** Request.

NAME	<b>Request</b>
ACTORS	Cab Catcher, Cab Driver
ENTRY CONDITIONS	Either the cab catcher needs to take a cab so he clicks on the request button or an API based system wants to perform a reservation on behalf of a cab catcher

FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The system receives his position</li> <li>• It spots the area associated to that position</li> <li>• It sends a notification about a pending request containing the address of the meeting point to a cab driver which is the first in the queue associated to the area to which belongs the position</li> </ul>
EXIT CONDITIONS	The request is sent to the first in the queue cab driver
EXCEPTIONS	If the queue is empty then the system looks for the nearest to the position non-empty queue

### [UC7] Reservation.

NAME	<b>Reservation</b>
ACTORS	Cab Catcher, Cab Driver
ENTRY CONDITIONS	Either the cab catcher needs to book a cab so he clicks on the reservation button or an API based system needs to perform a reservation on behalf of a cab catcher
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The systems shows the cab catcher the reservation page</li> <li>• The cab catcher fulfills the records about the meeting time, the origin and the destination of the ride</li> <li>• The cab catcher clicks on the submit reservation button</li> <li>• The system stores the reservation</li> <li>• Ten minutes before the meeting time the system assigns the reservation to the first cab driver associated to the nearest to the meeting point queue</li> <li>• The <b>Information Computation and Sending</b> use case starts</li> </ul>
EXIT CONDITIONS	The information about the estimated arrival time and the recognition code are sent to the cab catcher
EXCEPTIONS	If the queue to which belongs the meeting point is empty the system looks for the nearest non-empty queue. If the information inserted by the user are invalid the system highlight the wrong records

**[UC8] Delete Arrangement.**

NAME	<b>Delete Arrangement</b>
ACTORS	Cab Catcher
ENTRY CONDITIONS	Either the cab catcher wants to delete an arrangement so he clicks on the delete arrangement button or an API based system wants delete an arrangement on behalf of a cab catcher
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The system shows the list of all the active arrangements</li> <li>• The cab catcher chooses the arrangement to erase</li> <li>• If the arrangement is already designated then the <b>Delete Designated Arrangement</b> use case starts</li> <li>• Otherwise the system erases the arrangement</li> </ul>
EXIT CONDITIONS	The arrangement is deleted
EXCEPTIONS	

**[UC9] Delete Designated Arrangement.**

NAME	<b>Delete Designated Arrangement</b>
ACTORS	Cab Catcher, Cab Driver
ENTRY CONDITIONS	The cab catcher has chosen to delete an already designated arrangement
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The system deletes the chosen arrangement</li> <li>• The system informs the cab driver who was responsible for the deleted arrangement</li> </ul>
EXIT CONDITIONS	The cab driver has been informed
EXCEPTIONS	

**[UC10] Set Availability.**

NAME	<b>Set Availability</b>
ACTORS	Cab Driver
ENTRY CONDITIONS	The cab driver needs to change his availability state
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The cab driver clicks on the change availability button</li> <li>• The system switches the availability account state of the cab driver</li> <li>• If the state switches form not available</li> </ul>

	to available then the system assign the cab driver to the last position in the nearest queue with relation to his gps position;
EXIT CONDITIONS	The cab driver availability state is switched
EXCEPTIONS	

### [UC11] Decline Request.

NAME	<b>Decline Request</b>
ACTORS	Cab Driver
ENTRY CONDITIONS	The cab driver receives a request notification and wants to decline it
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The cab driver clicks on the decline button</li> <li>• The system puts the cab driver which has just declined the request into the last position of the queue</li> <li>• The system forwards the request to the next in the queue cab driver</li> </ul>
EXIT CONDITIONS	The request has been forwarded
EXCEPTIONS	If there isn't a cab driver into the current queue to which the system can forward the request, it looks for the nearest to the request location non-empty queue

### [UC12] Accept Request.

NAME	<b>Accept Request</b>
ACTORS	Cab Driver
ENTRY CONDITIONS	The cab driver receives a request notification and wants to accept it
FLOW OF EVENTS	<ul style="list-style-type: none"> <li>• The cab driver clicks on the accept button</li> <li>• The system switches his account state to unavailable</li> <li>• The <b>Information Computation and Sending</b> use case starts</li> </ul>
EXIT CONDITIONS	The information about the estimated arrival time and the recognition code are sent to the cab catcher
EXCEPTIONS	

**[UC13]** Information Computation and Sending.

NAME	<b>Information Computation and Sending</b>
ACTORS	Cab Catcher, Cab Driver, API Based System
ENTRY CONDITIONS	Either the cab driver accepts to handle a request or the cab driver has been assigned to a reservation
FLOW OF EVENTS	<ul style="list-style-type: none"><li>• The system carries out the estimated arrival time</li><li>• If the reservation or the request has been done by a cab catcher then the system sends the code and the estimated arrival time to the cab catcher itself</li><li>• Otherwise the system sends this information to the API based system</li></ul>
EXIT CONDITIONS	The information are sent to the appropriate user
EXCEPTIONS	

**[UC14]** Get Programmatic Interface.

NAME	<b>Get Programmatic Interface</b>
ACTORS	API Based System
ENTRY CONDITIONS	The API based system wants to obtain the programmatic interface
FLOW OF EVENTS	<ul style="list-style-type: none"><li>• The API based system starts the <b>Get Authorization</b> use case</li><li>• The API based system uses the authorization obtained to ask for the programmatic interface</li><li>• The system returns the programmatic interface</li></ul>
EXIT CONDITIONS	The API based system withholds the programmatic interface
EXCEPTIONS	

**[UC15]** Get Authorization.

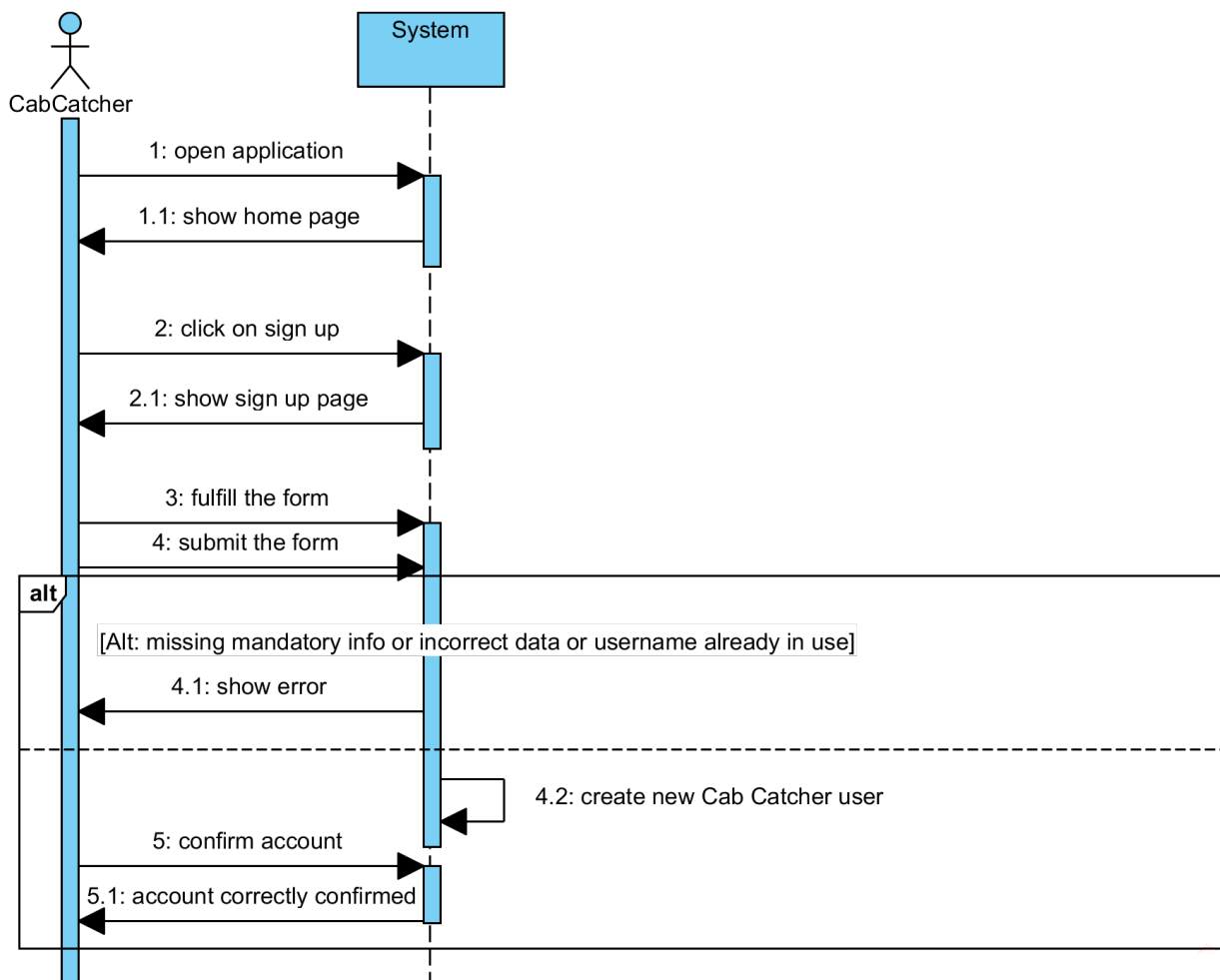
NAME	<b>Get Authorization</b>
ACTORS	API Based System
ENTRY CONDITIONS	The API based system needs the authorization to get the programmatic interface
FLOW OF EVENTS	<ul style="list-style-type: none"><li>• The API based system asks its user for the authorization to get the programmatic interface by starting</li></ul>

	the <b>Log In</b> use case
EXIT CONDITIONS	The API based system has gained the authorization
EXCEPTIONS	The user of the API based system doesn't log into the basic system so the getting authorization procedure is aborted

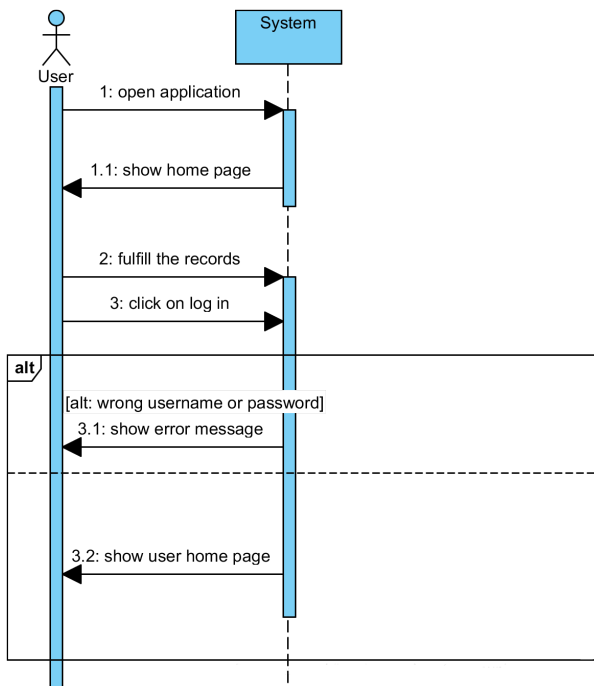
### 3.4.2. Sequence Diagrams

In the section below it's shown a sequence diagram for each use case:

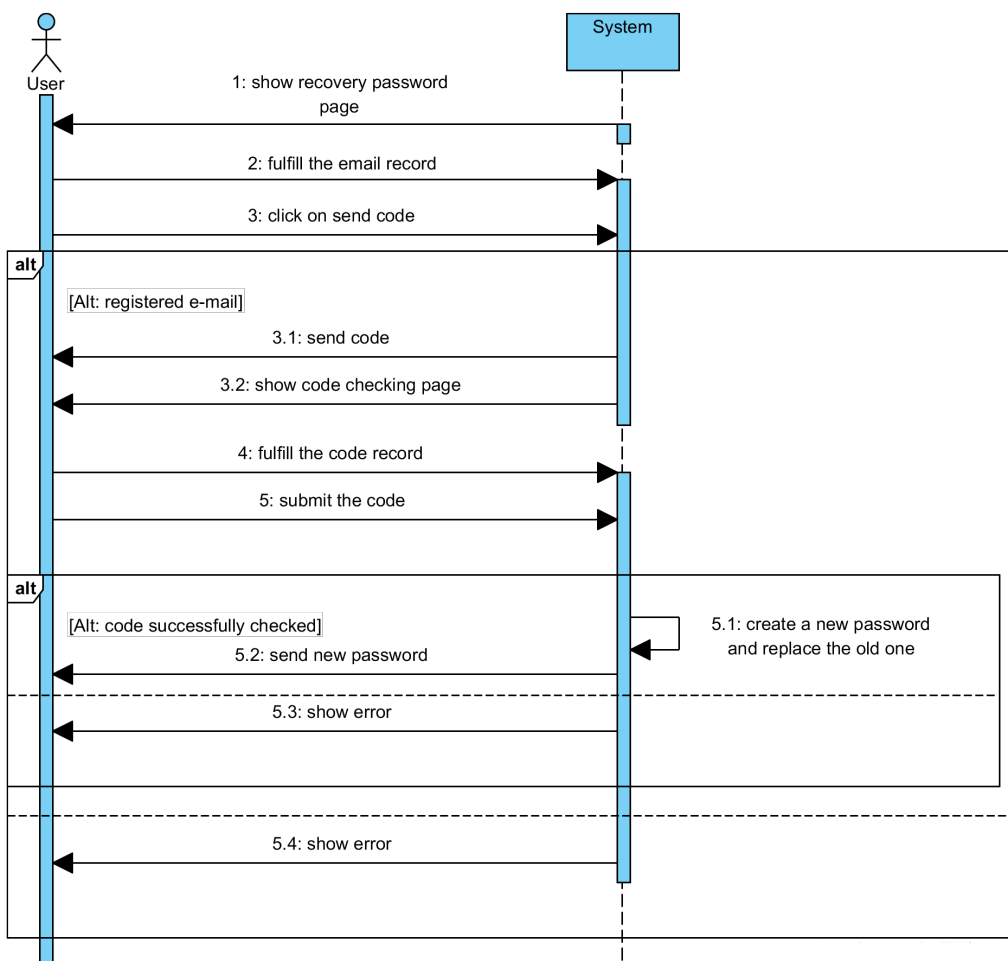
#### [SD1] Sign Up.



## [SD2] Log In.

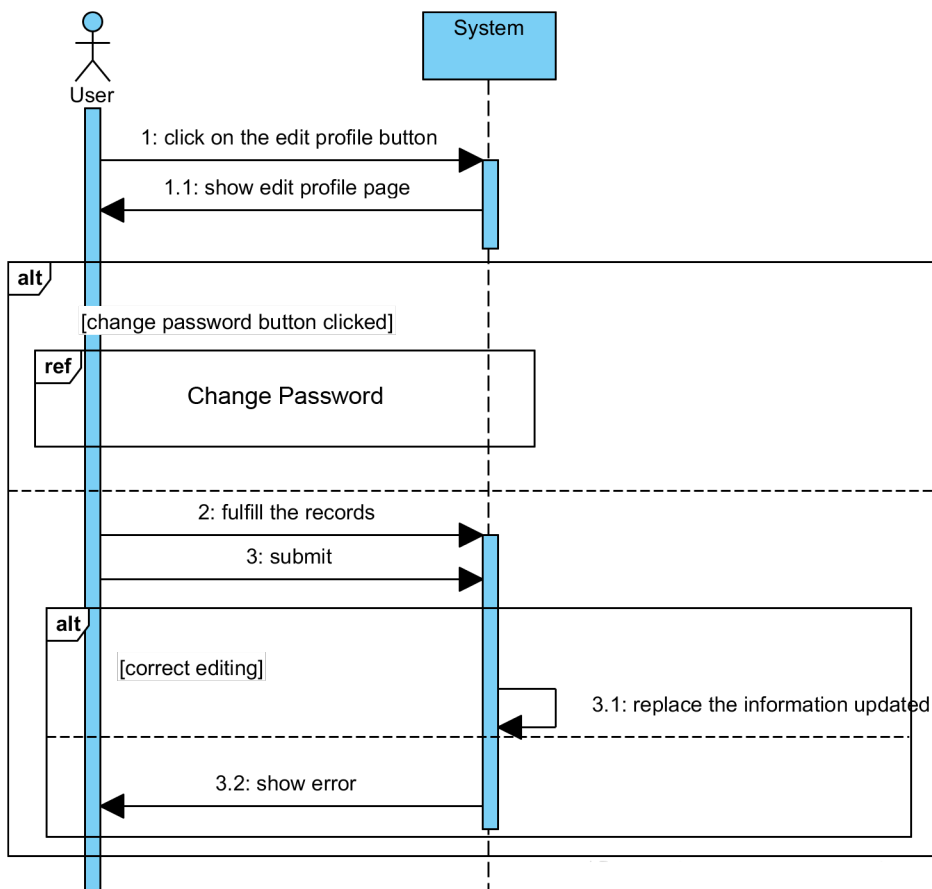


## [SD3] Password Recovery.

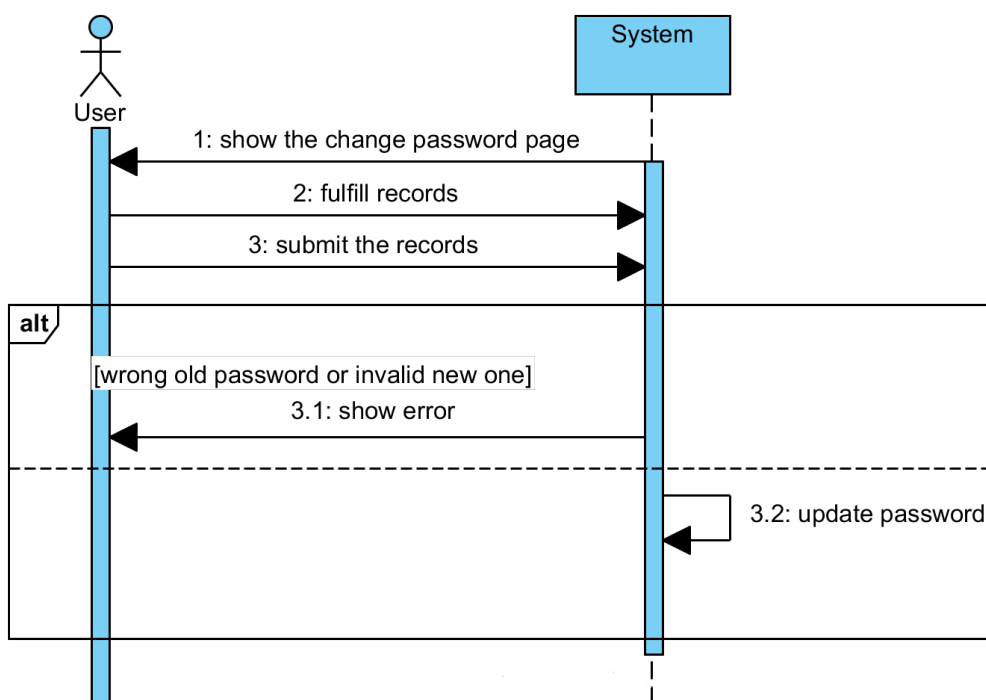




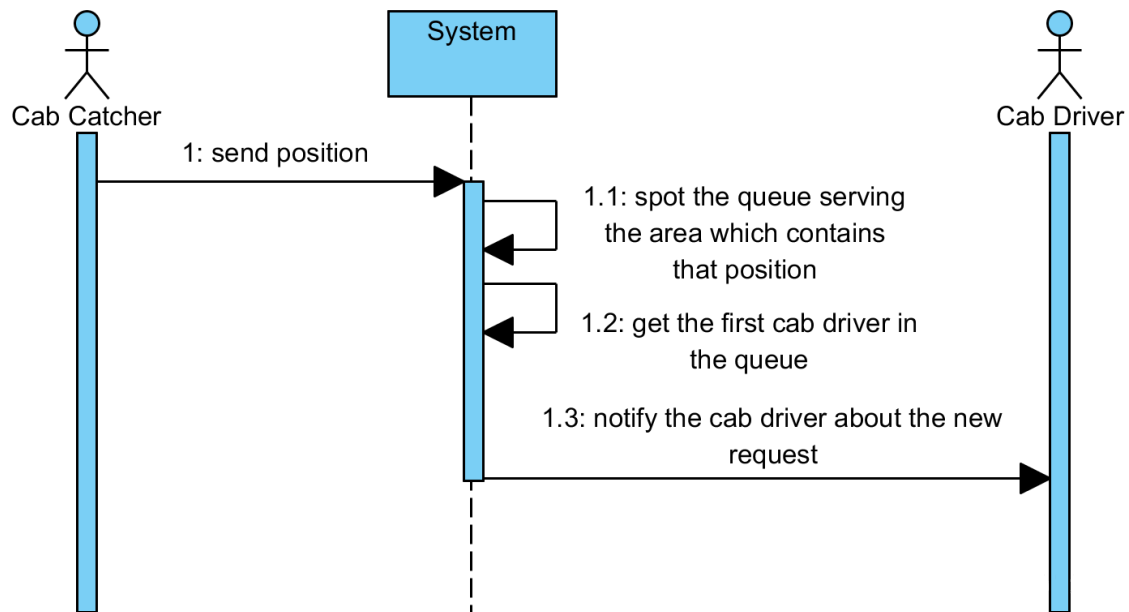
#### [SD4] Edit Profile.



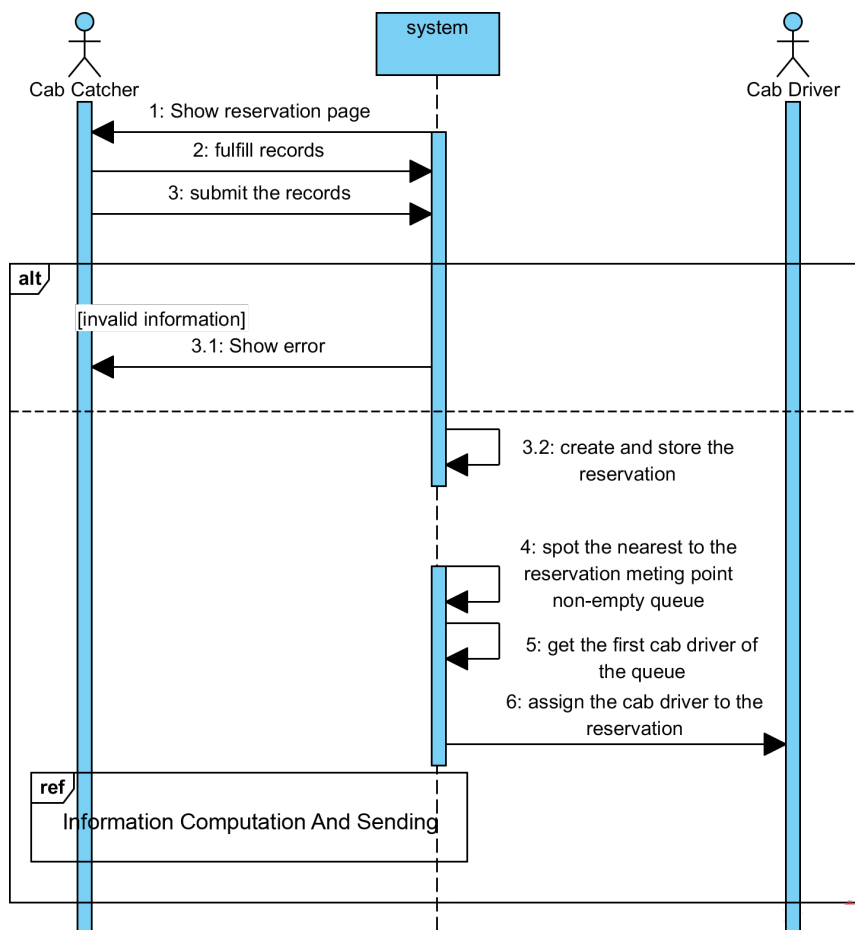
#### [SD5] Change Password.



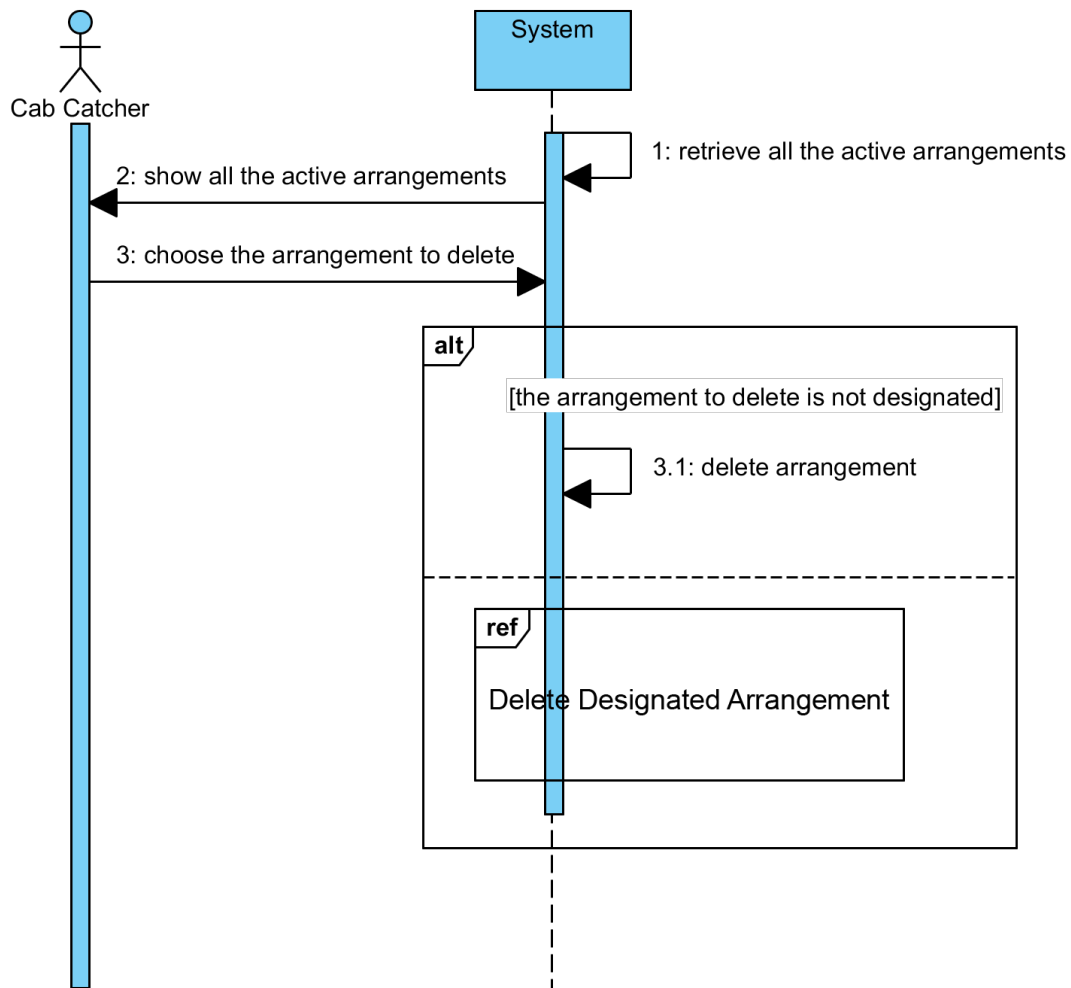
### [SD6] Request.



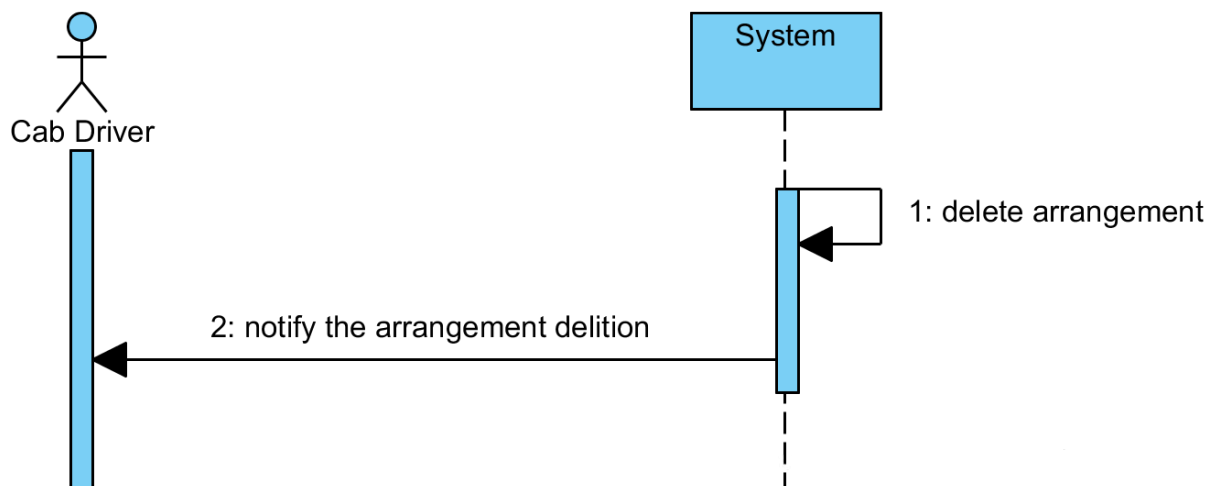
### [SD7] Reservation.



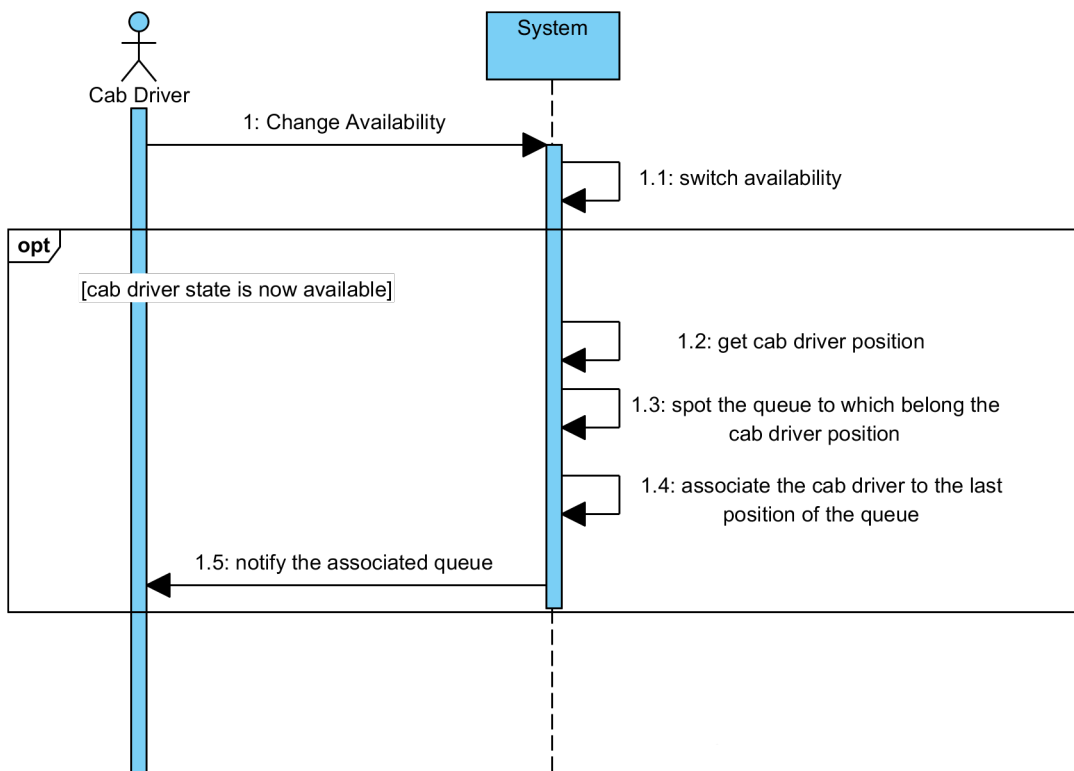
### [SD8] Delete Arrangement.



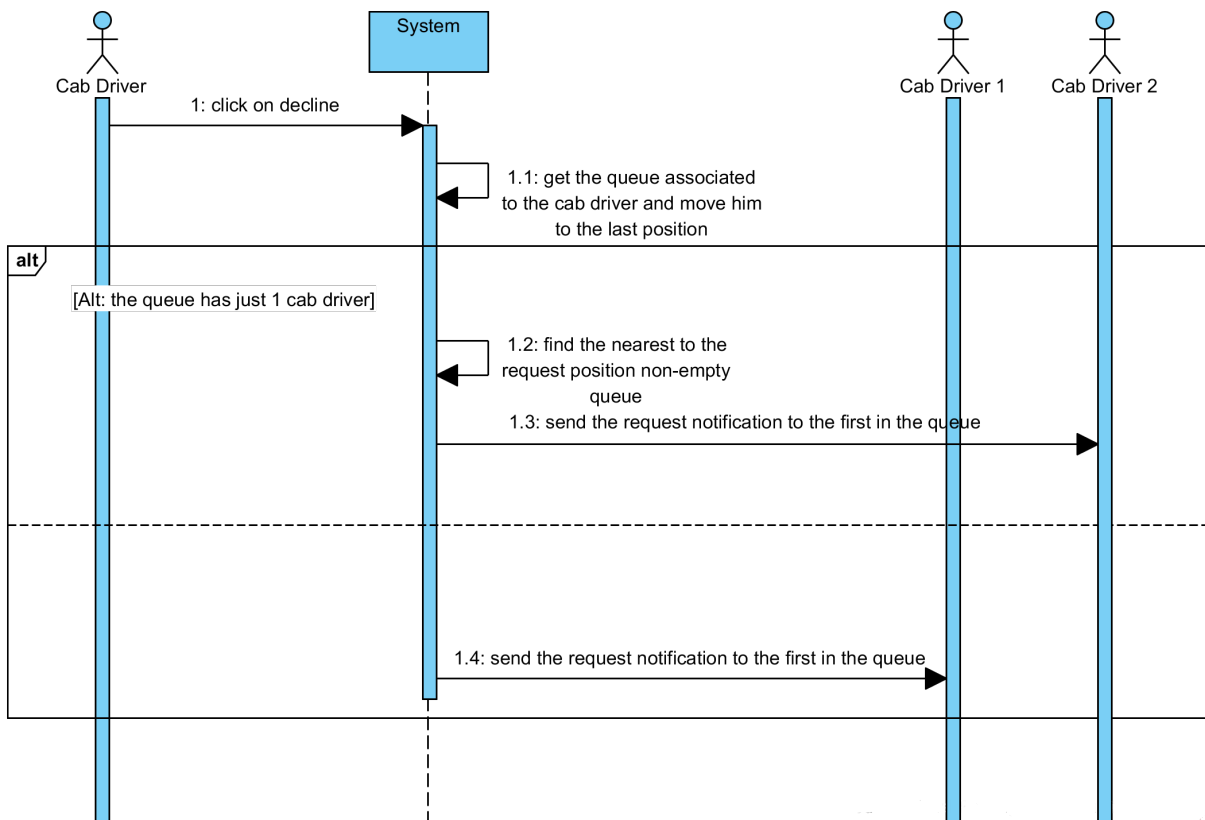
### [SD9] Delete Designated Arrangement.



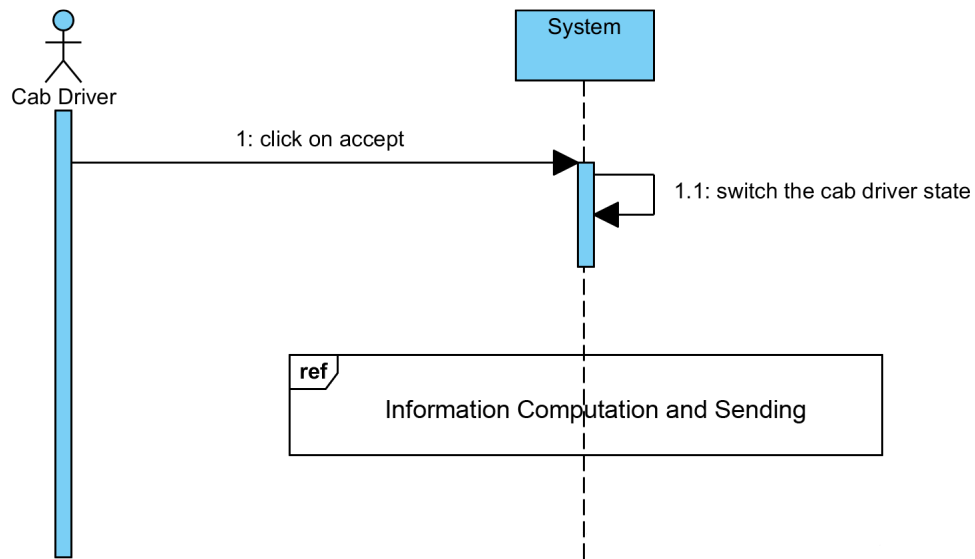
### [SD10] Set Availability.



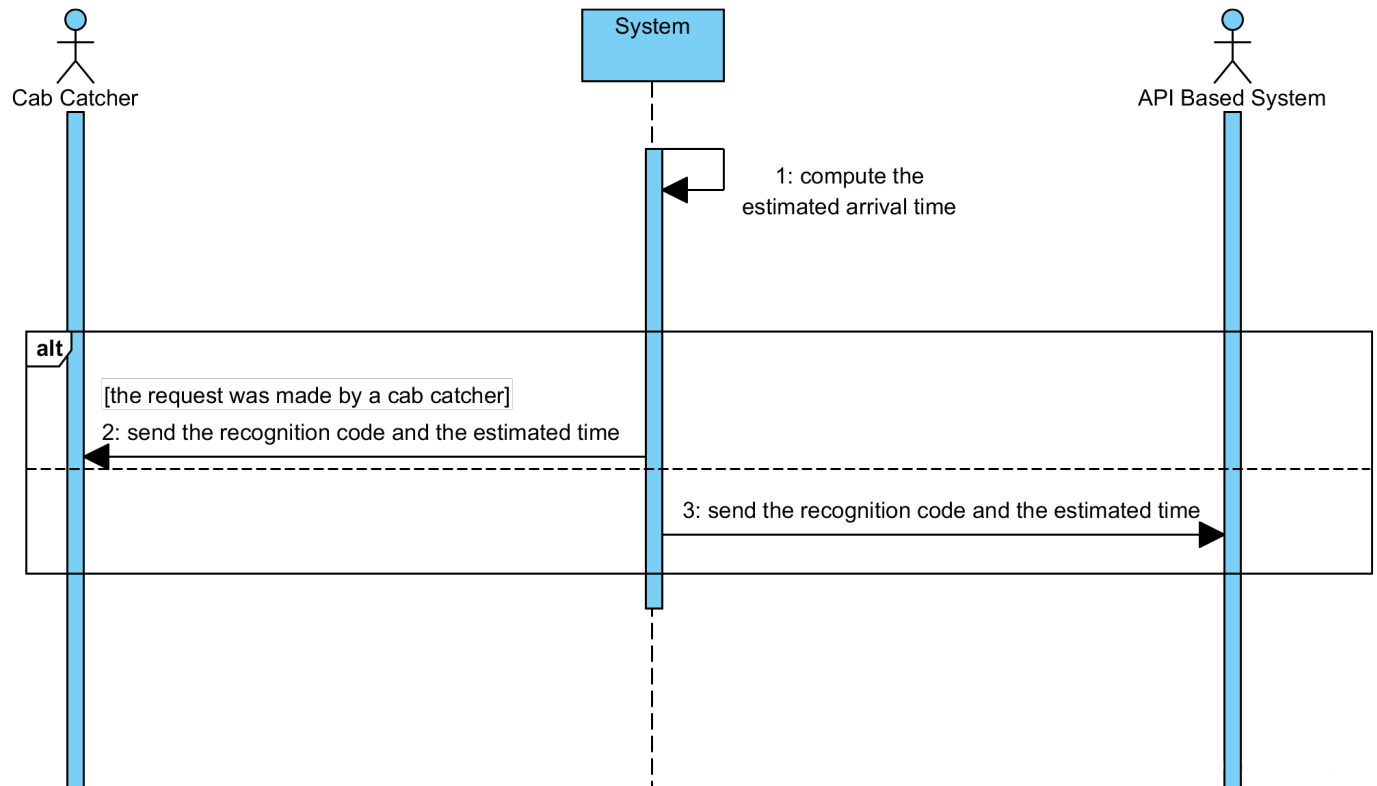
### [SD11] Decline Request.



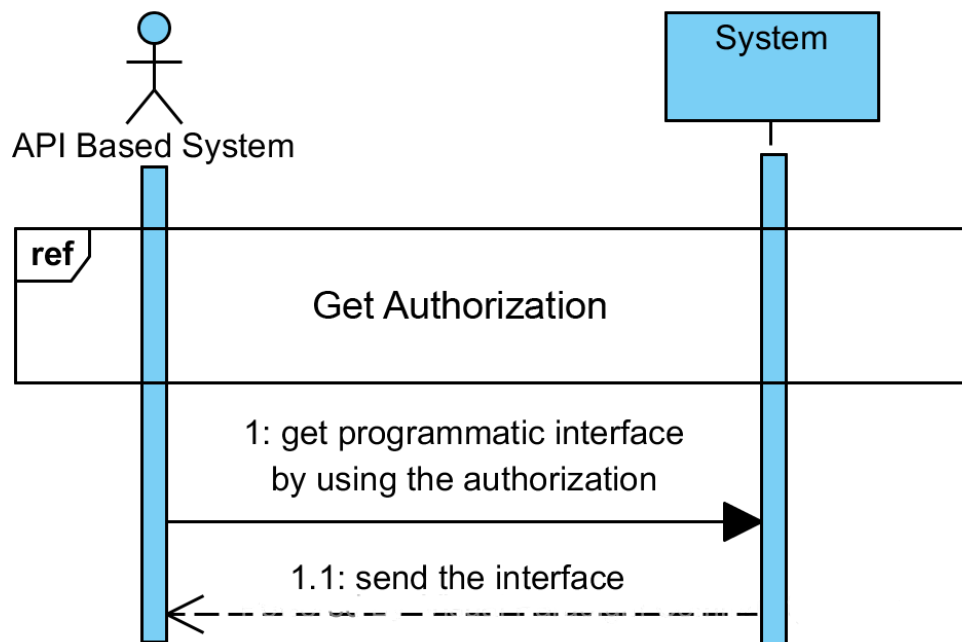
### [SD12] Accept Request.



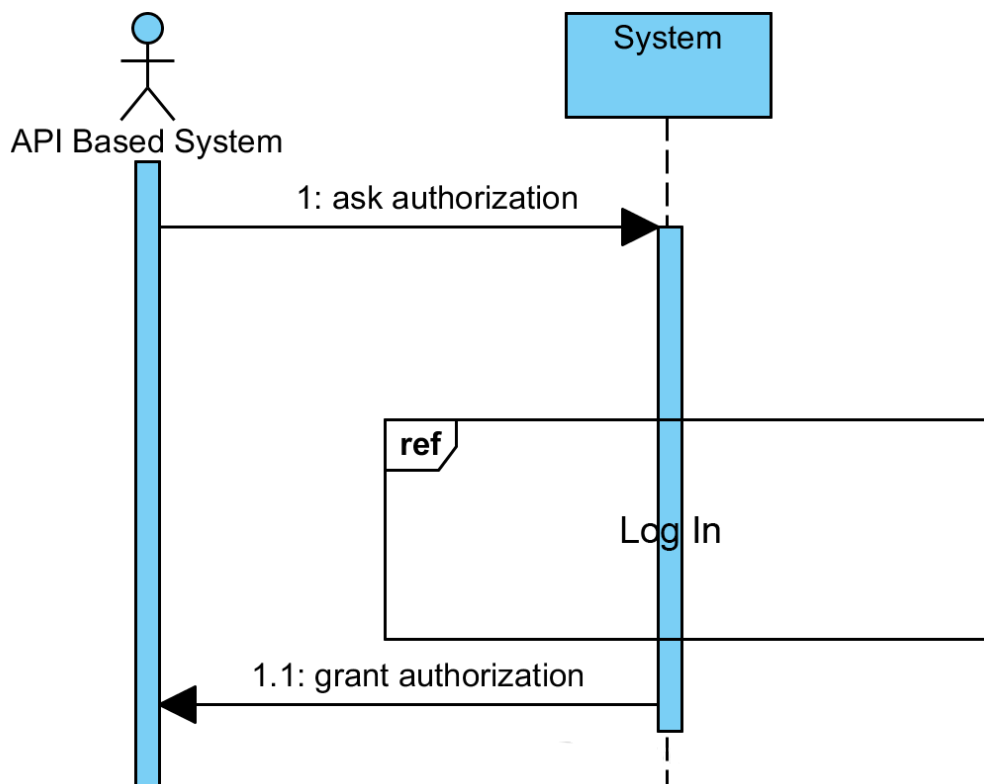
### [SD13] Information Computation and Sending.



**[SD14]** Get Programmatic Interface.

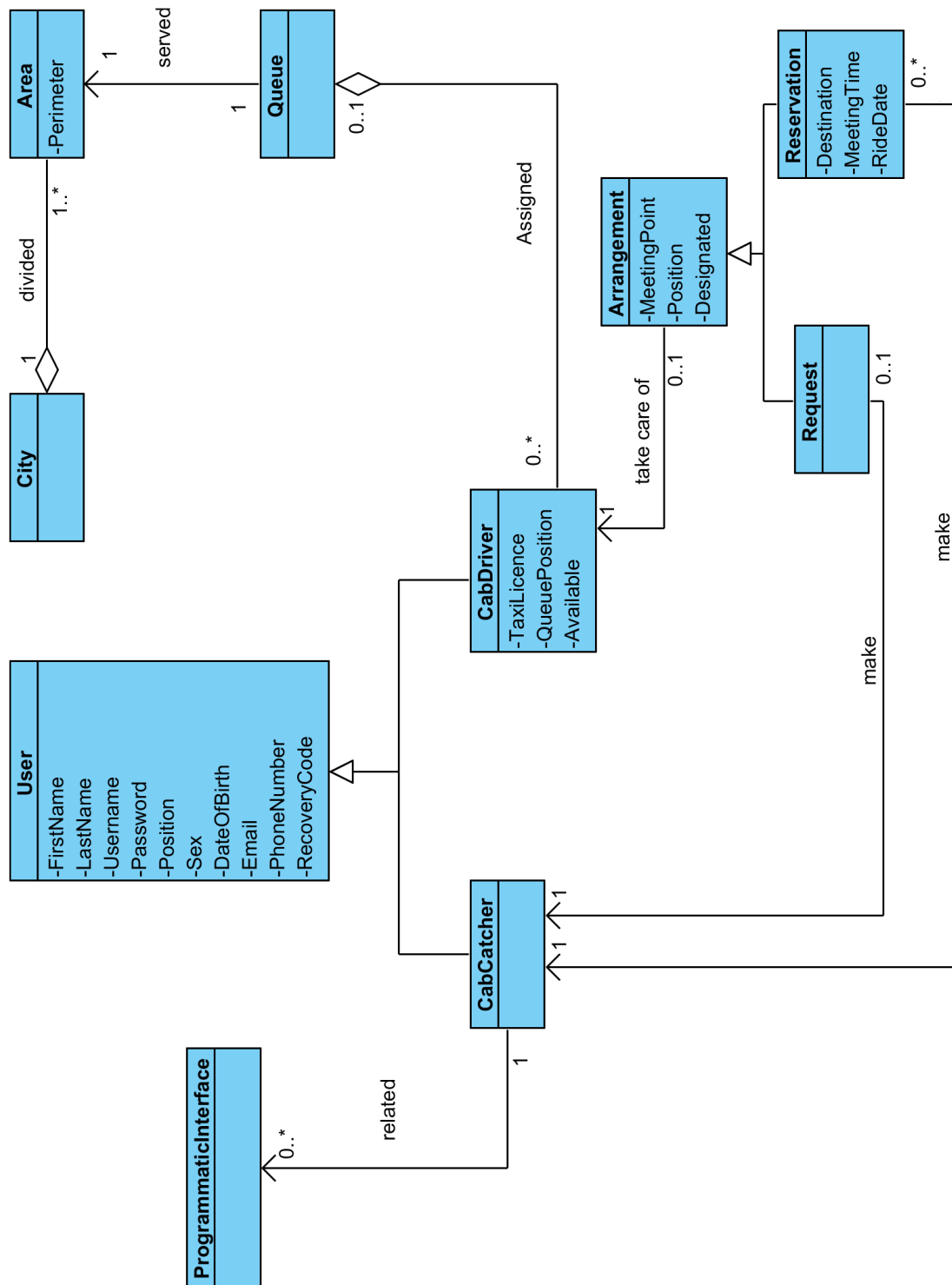


**[SD15]** Get Authorization.



### 3.4.3. Class Diagram

In the section below it's shown a representation of the Class Diagram:



### 3.5. Alloy Model

In the section below it's shown the Alloy Analyzer code used to generate and analyse a model representing the system interaction with the world. This model has been derived from the Class Diagram.

```
//Alloy Model for MyTaxiService

//Datatype representing alphanumeric strings
one sig Strings{}

//Datatype representing dates
sig DateTime{}

//Datatype representing birthdays
one sig DateBirthday{}

//Datatype representing positions of cab catchers
sig Position{}

//Datatype representing boundaries of zones
one sig Perimeter{}

//The zones in which the city is divided
some sig Zone{
    zonePerimeter: one Perimeter
}

//The taxi queues to which zones are related
sig TaxiQueue{
    zone: one Zone
}

//The taxies of the city
some sig Taxi{
    queue: lone TaxiQueue,
    code: one Strings,
    after: lone Taxi
}
```

```
//The taxi drivers
sig CabDriver{
    taxi: one Taxi,
    firstName: one Strings,
    lastName: one Strings,
    username: one Strings,
    password: one Strings,
    phoneNumber: one Strings,
    email: one Strings,
    sex: one Strings,
    dateOfBirth: one DateBirthday
}

//The taxi potential passengers
sig CabCatcher{
    firstName: one Strings,
    lastName: one Strings,
    username: one Strings,
    password: one Strings,
    phoneNumber: one Strings,
    email: one Strings,
    sex: one Strings,
    dateOfBirth: one DateBirthday,
    catcherPosition: one Position
}

//The passenger requests
sig Request{
    requestPassenger: one CabCatcher,
    requestDriver: lone CabDriver
}

//The passenger reservations
sig Reservation{
    reservationPassenger: one CabCatcher,
    reservationDriver: lone CabDriver,
    meetingTime: one DateTime,
    originPoint: one Position,
    destinationPoint: one Position
}
```



```

//Position is a data type associated with cab catchers, it has to be present only if there is at least a cab catcher
fact{
    some CabCatcher implies #Position = 1 else #Position = 0
}

//DateTime is a data type associated with reservations, it has to be present only if there is at least a reservation
fact{
    some Reservation implies #DateTime = 1 else #DateTime = 0
}

//Each cab driver who has taken care of a request is no more related to a taxi queue (unavailable)
fact{
    all r: Request, t: Taxi | (some r.requestDriver and r.requestDriver.taxi = t) implies #t.queue = 0
}

//Each taxi that is not implicated in an arrangement is related to a taxi queue (available)
fact{
    all t: Taxi | (t not in (Request.requestDriver.taxi + Reservation.reservationDriver.taxi)) implies #t.queue = 1
}

//Each cab driver who has taken care of a reservation is no more related to a taxi queue (unavailable)
fact{
    all r: Reservation, t: Taxi | (some r.reservationDriver and r.reservationDriver.taxi = t) implies #t.queue = 0
}

//Each zone has exactly a taxi queue
fact{
    all z: Zone | #z.~zone = 1
}

//Each taxi has exactly a cab driver
fact{
    all t: Taxi | #t.~taxi = 1
}

//The queues are composed of taxis that are linked according the "after" relationship
fact{
    all t: Taxi | t not in TaxiQueue.~queue implies #(t.after) = 0 and #(t.~after) = 0
    all t: Taxi, q: TaxiQueue | t in q.~queue and #(q.~queue) > 1 implies #(t.after + t.~after) >= 1
    all t: Taxi | t->t not in ^after
    all t: Taxi | t.after != t
    all q: TaxiQueue | (q.~queue.after.queue in q) and #(q.~queue - q.~queue.after) = 1
    all q: TaxiQueue | #(q.~queue - q.~queue.~after) = 1
}

//The number of cab drivers must be greater than or equal to the number of requests, due to the domain assumptions
fact{
    #CabDriver >= #Request
}

//Each cab catcher can make only one request per time
fact{
    all c: CabCatcher | #c.~requestPassenger <= 1
}

//Each cab catcher can't have both an allocated reservation and an allocated request
fact{
    all c: CabCatcher | some (c.~reservationPassenger.reservationDriver) implies #(c.~requestPassenger.requestDriver) = 0
}

//Each cab catcher can't have both an allocated reservation and an allocated request
fact{
    all c: CabCatcher | some (c.~requestPassenger.requestDriver) implies #(c.~reservationPassenger.reservationDriver) = 0
}

//Each cab catcher can't have more than one allocated reservation
fact{
    all c: CabCatcher | #(c.~reservationPassenger.reservationDriver) <= 1
}

//Each cab driver can't handle more than one arrangement per time
fact{
    all c: CabDriver | #(c.~requestDriver + c.~reservationDriver) <= 1
}

```

/Predicate to make a request

```
red makeRequest[c, c': CabCatcher, r: Request]{
  c != c'
  #c.~reservationPassenger = #c'.~reservationPassenger
  r.requestDriver = none
  c'.~requestPassenger = c.~requestPassenger + r
}
```

un makeRequest for 3

/Predicate to make a reservation

```
red makeReservation[c, c': CabCatcher, r: Reservation]{
  c != c'
  r.reservationDriver = none
  #c.~requestPassenger = #c'.~requestPassenger
  c'.~reservationPassenger = c.~reservationPassenger + r
}
```

un makeReservation for 5

/Predicate to delete a request

```
red deleteRequest[c, c': CabCatcher, r: Request]{
  r in (c.~requestPassenger)
  c'.~requestPassenger = (c.~requestPassenger - r)
}
```

un deleteRequest for 5

/Predicate to delete a reservation

```
red deleteReservation[c, c': CabCatcher, r: Reservation]{
  r in (c.~reservationPassenger)
  c'.~reservationPassenger = (c.~reservationPassenger - r)
}
```

un deleteReservation for 2

Predicate to allocate a reservation to a cab driver

```
ed allocateReservation[c: CabDriver, r, r': Reservation]{
  r.reservationDriver = none
  r'.reservationPassenger = r.reservationPassenger
  r'.meetingTime = r.meetingTime
  r'.originPoint = r.originPoint
  r'.destinationPoint = r.destinationPoint
  r'.reservationDriver = c
}
```

n allocateReservation for 5

Check that the number of request is lower than or equal to the number of cab catchers

```
sert lowerRequests{
  #Request <= #CabCatcher
}
```

eck lowerRequests for 5

Check the makeRequest predicate

```
sert addRequest{
  all c1, c2: CabCatcher, r: Request | makeRequest[c1, c2, r] implies (r in (c2.~requestPassenger)) and (r not in (c1.~requestPassenger))
  all c1, c2: CabCatcher, r: Request | makeRequest[c1, c2, r] implies c1.~requestPassenger = none
}
```

eck addRequest for 5

Check the makeReservation predicate

```
sert addReservation{
  all c1, c2: CabCatcher, r: Reservation | makeReservation[c1, c2, r] implies (r in (c2.~reservationPassenger)) and (r not in (c1.~reservationPassenger))
}
```

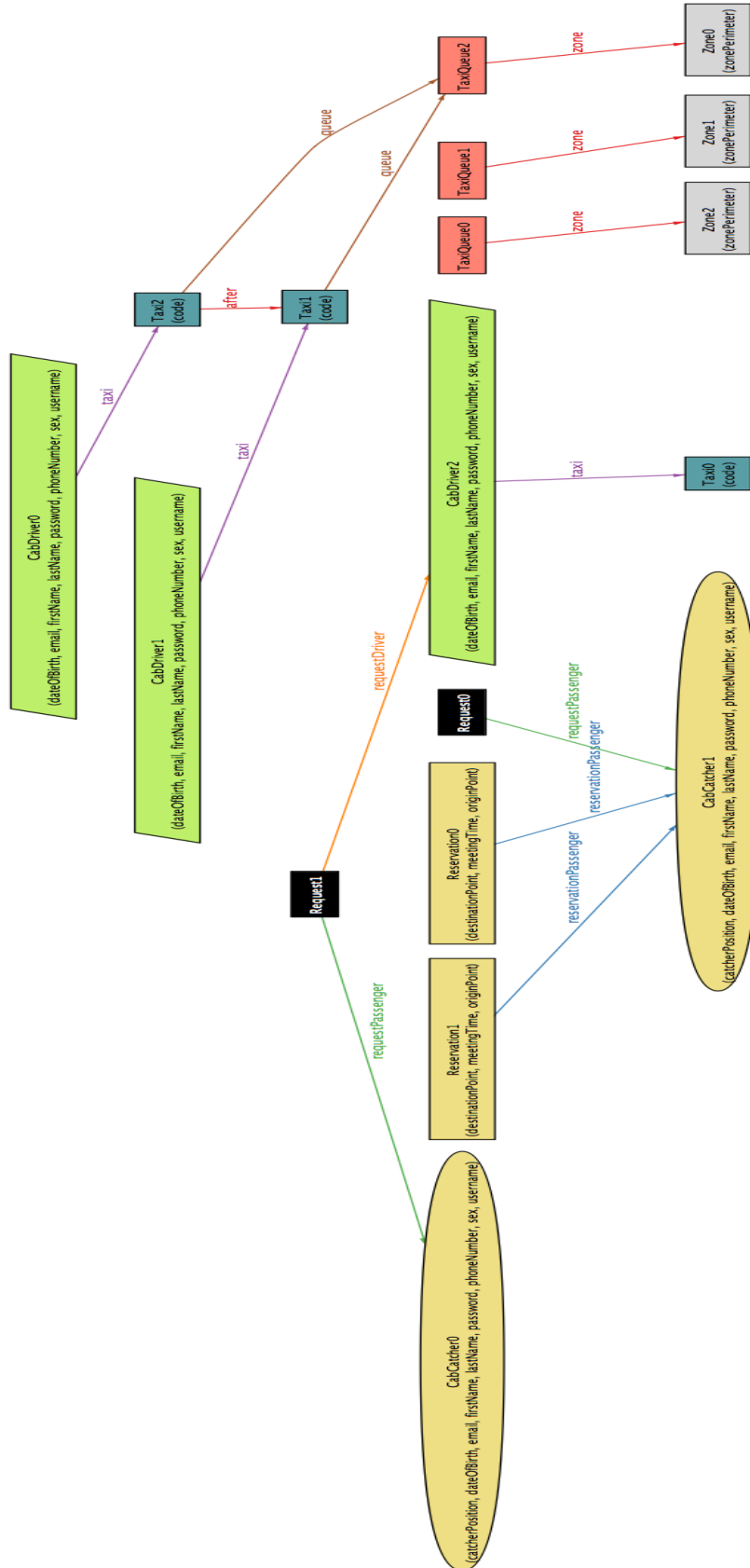
eck addReservation for 5

Check the deleteRequest predicate

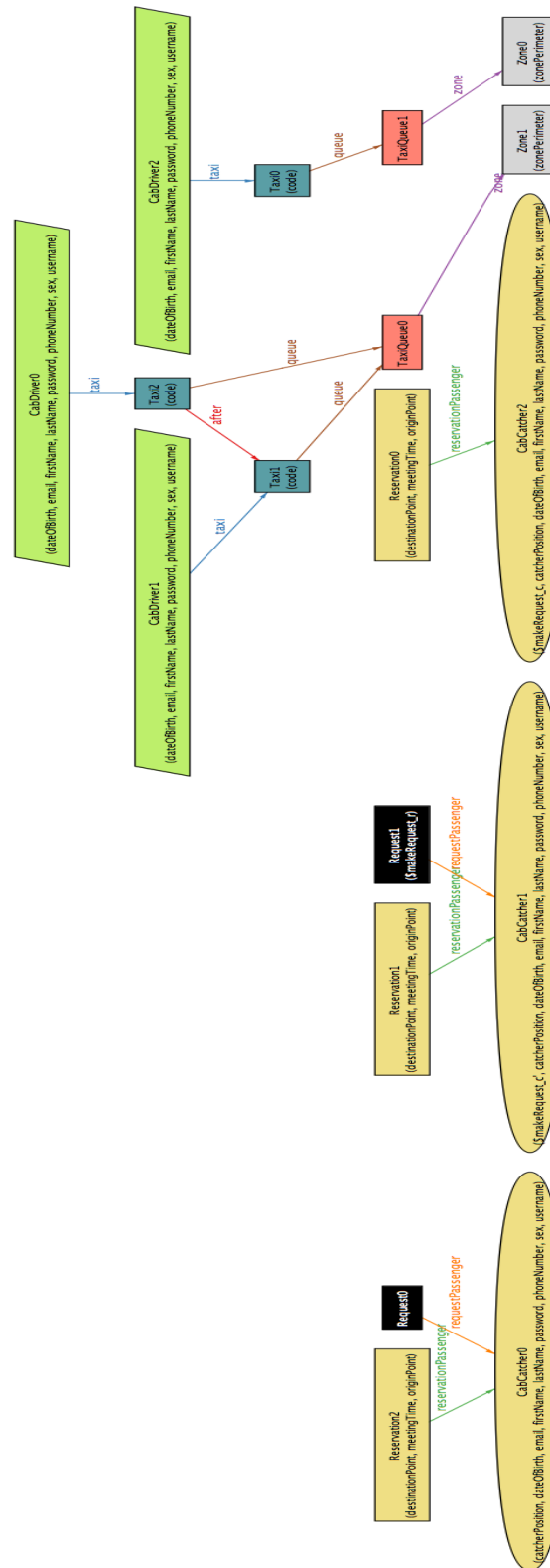
```
sert cancelRequest{
  all c1, c2: CabCatcher, r: Request | deleteRequest[c1, c2, r] implies (r not in (c2.~requestPassenger)) and c1 != c2
}
```

eck cancelRequest for 5

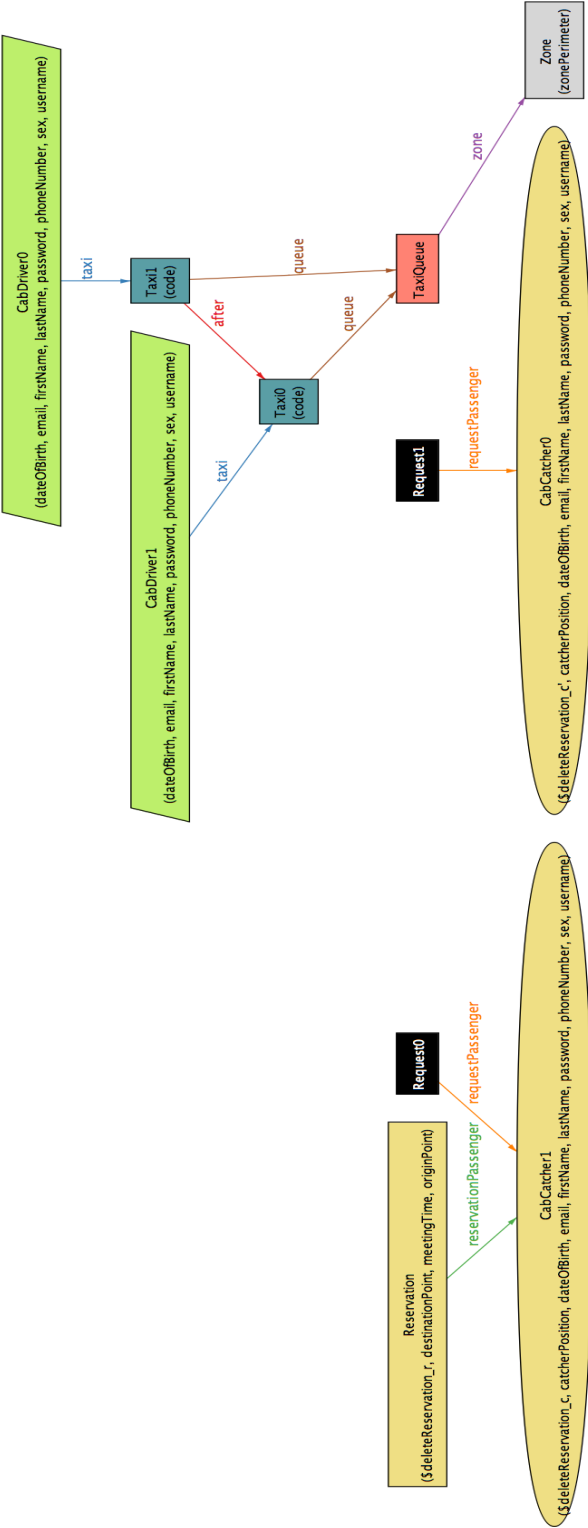
The following screenshot represents the world generated by using the **run showWorld** command:



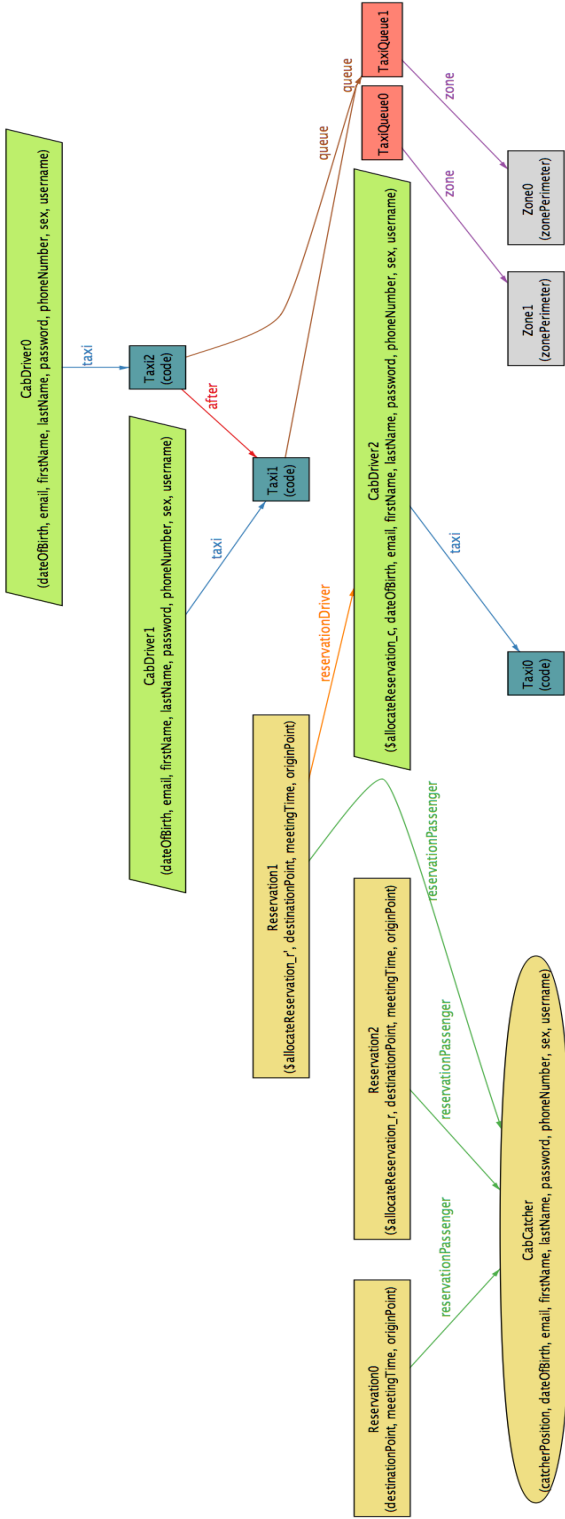
The following screenshot represents the world generated by using the **run makeRequest** command:



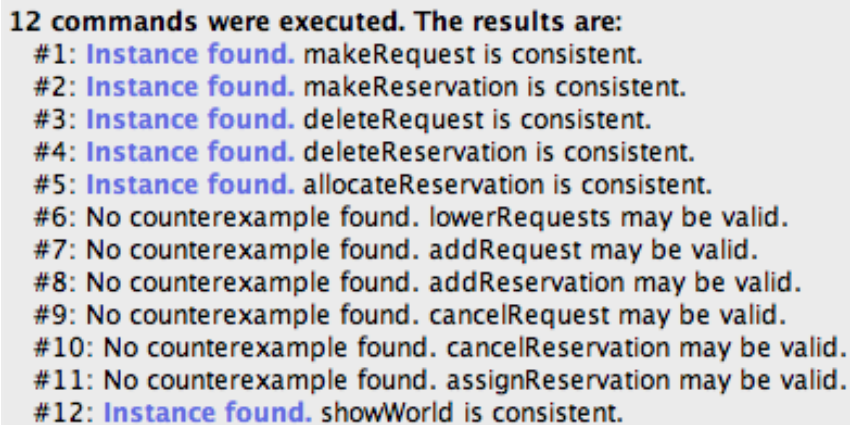
The following screenshot represents the world generated by using the **run deleteReservation** command:



The following screenshot represents the world generated by using the **run allocateReservation** command:



The following screenshot represents the analysis result:



```
12 commands were executed. The results are:
#1: Instance found. makeRequest is consistent.
#2: Instance found. makeReservation is consistent.
#3: Instance found. deleteRequest is consistent.
#4: Instance found. deleteReservation is consistent.
#5: Instance found. allocateReservation is consistent.
#6: No counterexample found. lowerRequests may be valid.
#7: No counterexample found. addRequest may be valid.
#8: No counterexample found. addReservation may be valid.
#9: No counterexample found. cancelRequest may be valid.
#10: No counterexample found. cancelReservation may be valid.
#11: No counterexample found. assignReservation may be valid.
#12: Instance found. showWorld is consistent.
```

### 3.6. Used Tools

The tools used to create the RASD document are:

- Microsoft Office Word 2011: to redact and to format this document.
- Visual Paradigm: to create the Class Diagram, the Sequence Diagrams and the Use Case Diagram.
- Alloy Analyzer 4.2: to create a model of the and prove its consistency.

For redacting and writing this document we have spent **30 hours** per person.