Politecnico di Milano
A.A. 2015-2016
Software Engineering 2: “MyTaxiService”

Requirement **A**nalysis and **S**pecifications **D**ocument

Giuseppe Canonaco, Andrea Di Giosaffatte

6 November 2015 – Version 1.3

Table of Contents

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

MyTaxyService 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^{-th} goal.
- **[UIn]** The n^{-th} user interface.
- **[An]** The n^{-th} dependency.
- **[Dn]** The n^{-th} domain assumption.
- **[Rm.n]** The n^{-th} functional requirement related to the m^{-th} goal.
- **[NFRn]** The n^{-th} non-functional requirement.
- **[UCn]** The n^{-th} use case diagram.
- **[SDn]** The n^{-th} 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

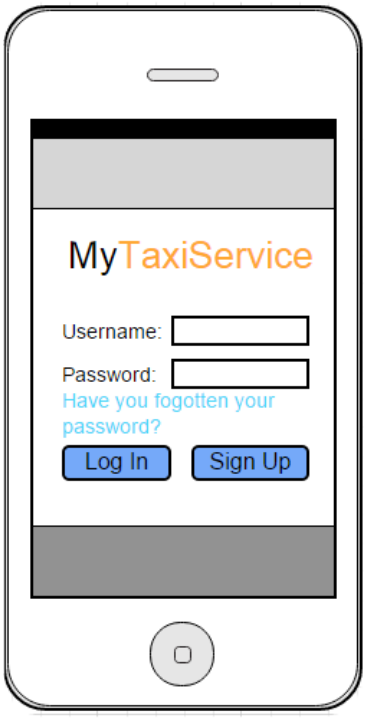
2.1. Product Perspective

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.

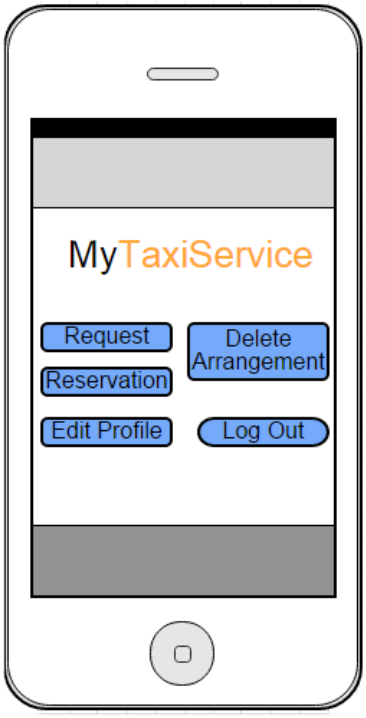
2.1.1. User Interfaces

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.

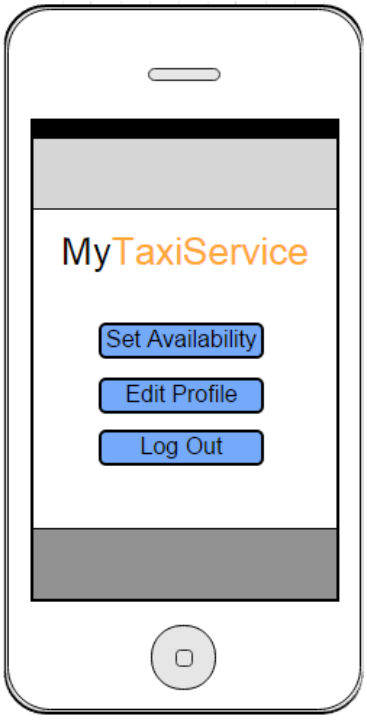
[UI1] User Log In.



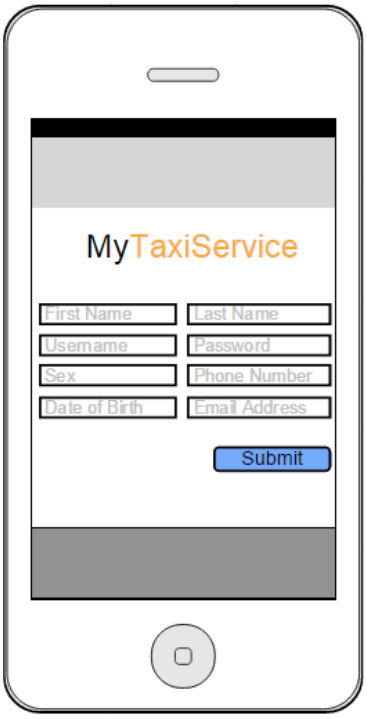
[UI2] Cab Catcher Home Page.



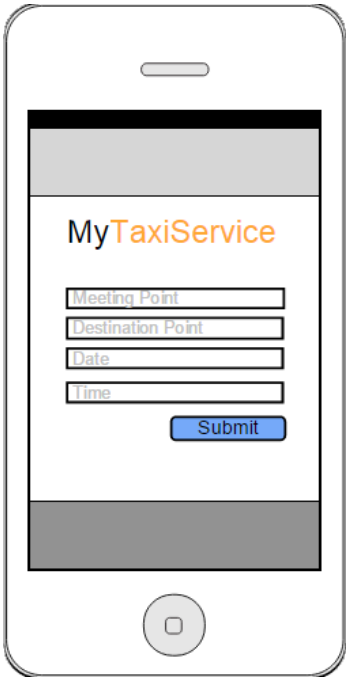
[UI3] Cab Driver Home Page.



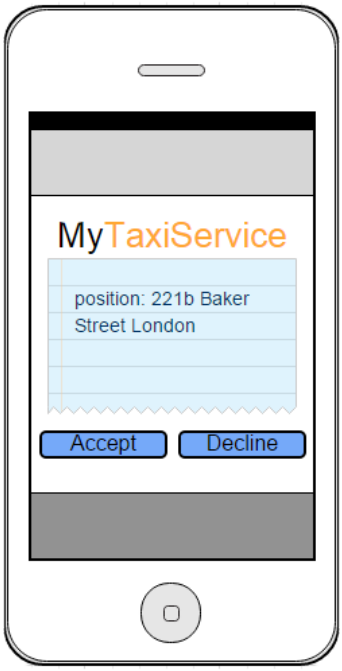
[UI4] Cab Catcher Sign Up.



[UI5] Cab Catcher Reservation.



[UI6] Notification of a Pending Request. This is dual to the Pending Reservation.



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 compute the estimated arrival time 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.

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.

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.
- **[R1.5]** The system has to provide a Change E-mail functionality to allow uers to modify their e-mail.
- **[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 inform the cab catcher about the estimated arrival time and about the code by which he can recognize the car.

- **[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.

[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 Arrangement functionality, which allows cab drivers to decline a taxi arrangement.

- **[R3.3]** The system has to provide an Accept Arrangement functionality, which allows cab drivers to accept a taxi arrangement.

- **[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.

[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.

- **[R4.2]** The system has to allow a third part system to perform a request on behalf of a cab catcher.

- **[R4.3]** The system has to allow a third part system to perform a reservation on behalf of a cab catcher.

- **[R4.4]** The system has to allow a third part system to check the status of a particular arrangement.

- **[R4.5]** The system has to allow a third part system to delete a particular arrangement.

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 OS: 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 2 hours starting from the moment in which the code is generated.

- **[NFR6]** The system, in case of reservation, tries to allocate a taxi to that arrangement 10 minutes before the meeting time with the cab catcher.

- **[NFR7]** The system considers as refused an arrangement notification which is not answered by a cab driver within 1 minute.

- **[NFR8]** The system should be available 99,9% of time over a year.

- **[NFR9]** Every non designated arrangement has a time-out. When this time-out is expired the arrangement can't be designated anymore. This time-out is equal to 20 minute starting from the first attempt of the system to allocate it.

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 taxies when she needs to get to the airport. Recently she heard from a friend that a new taxy 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.

[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 MyTaxyService 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 get 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 request 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.

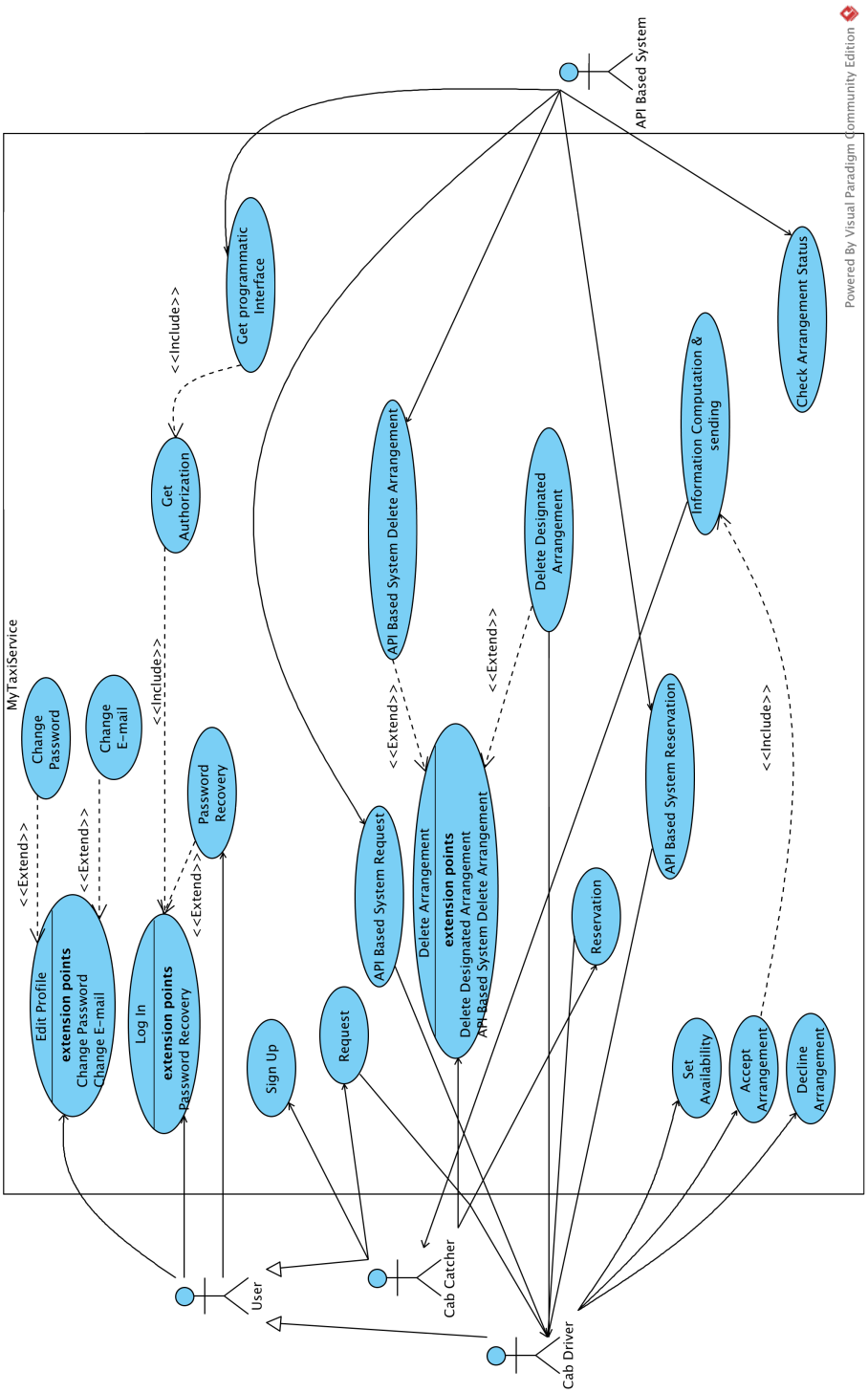
[SC9] API based system request.

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	**Sign Up**
ACTORS	Cab Catcher
ENTRY CONDITIONS	The Cab Catcher isn't registered to the MyTaxiService application
FLOW OF EVENTS	• The cab catcher open the application • The system shows him the application home page • He clicks on the sign up button • The application shows him the sign-up page • He fulfills the sign up form which requires the following information to be inserted: - First name - Last name - Username - Password - Sex - Phone number - Date of birth - E-mail • He submits the form • He confirms his account by clicking on a link contained in a confirmation e-mail
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	**Log in**
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	• The system shows the log in page • The user inserts his username and his

	password • The user clicks on the log in button
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 **Password Recovery** use case

[UC3] Password Recovery.

NAME	**Password Recovery**
ACTORS	User
ENTRY CONDITIONS	The User has forgotten his password and has clicked on the password recovery button
FLOW OF EVENTS	• The system shows the user the password recovery page • The user fulfills the email record • The user clicks on the send code button • The user receives the security code on his mobile phone • The system shows the user the code checking page to allow him to change his password • The user fulfills the code record with the received code • The user submits it • The system generates a new password and replaces the old one • The system sends the new password to the user phone number
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	**Edit Profile**
ACTORS	User

ENTRY CONDITIONS	The logged User needs to change some information about his profile and clicks the edit profile button
FLOW OF EVENTS	• If the user needs to change his account password or his e-mail, then he respectively clicks on the change password button which starts the **Change Password** use case or on the change e-mail button which starts the **Change E-mail** use case. • Otherwise, the user simply changes the information to be modified • The user submits the updated profile
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	**Change Password**
ACTORS	User
ENTRY CONDITIONS	While the logged User is editing his profile he clicks on the change password button
FLOW OF EVENTS	• The system shows the user the change password page • The user fulfills the records, the first one related to the old password and the second related to the new one • He clicks over the submit button to send the new information
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] Change E-mail.

NAME	**Change E-mail**
ACTORS	User
ENTRY CONDITIONS	While the logged User is editing his profile he clicks on the change e-mail button
FLOW OF EVENTS	• The system shows the user the change e-mail page • The user fulfills the record with his

	new e-mail • He clicks over the submit button to send the new information • The system sends him a confirmation e-mail
EXIT CONDITIONS	The user confirms his new e-mail
EXCEPTIONS	If the new e-mail is not valid the system shows an error message

[UC7] Request.

NAME	**Request**
ACTORS	Cab Catcher, Cab Driver
ENTRY CONDITIONS	The Cab Catcher, who is already logged into the system, wants to take a cab so he clicks on the request button
FLOW OF EVENTS	• The system receives his position • It spots the nearest to the position non-empty queue • If the queue exists then it sends a notification about the pending request to a cab driver which is the first in the found queue
EXIT CONDITIONS	The request is sent to the first in the found queue cab driver
EXCEPTIONS	If the system can't find a non-empty queue, it sends a service unavailable message to the cab catcher

[UC8] API Based System Request.

NAME	**API Based System Request**
ACTORS	API Based System, Cab Driver
ENTRY CONDITIONS	An API Based System, that has already gained the programmatic interface, wants to perform a request on behalf of a Cab Catcher through the programmatic interface itself
FLOW OF EVENTS	• The API based system sends the position of the cab cather to the basic system • The basic system looks for the nearest to the position non-empty queue • If the queue exists the basic system sends a notification about the pending request to the first in the queue cab driver

	• The information about the outcome of the research are stored
EXIT CONDITIONS	The outcome of the request is stored
EXCEPTIONS	If the position of the request is invalid, the basic system sends back an error message.

[UC9] Reservation.

NAME	**Reservation**
ACTORS	Cab Catcher, Cab Driver
ENTRY CONDITIONS	The Cab Catcher, who is already logged into the system, wants to book a cab so he clicks on the reservation button
FLOW OF EVENTS	• The system shows the cab catcher the reservation page • The cab catcher fulfills the records about the meeting time, the origin and the destination of the ride • The cab catcher clicks on the submit reservation button • The system stores the reservation • 10 minutes before the meeting time, the system looks for the nearest to the origin of the ride non-empty queue • If the queue exists the basic system sends a notification about the pending reservation to the first cab driver in that queue • Otherwise the system sends a service unavailable message to the cab catcher
EXIT CONDITIONS	The reservation is sent to the first in the found queue cab driver or a service unavailable message is sent to the cab catcher
EXCEPTIONS	If the information inserted by the user are invalid the system highlight the wrong records

[UC10] API Based System Reservation.

NAME	**API Based System Reservation**
ACTORS	API Based System, Cab Driver
ENTRY CONDITIONS	An API Based System, that has already gained the programmatic interface, wants to perform a reservation on behalf of a Cab Catcher through the programmatic interface

	itself
FLOW OF EVENTS	• The API based system sends the meeting time, the origin and the destination of the ride to the basic system • The basic system stores the reservation • 10 minutes before the meeting time, the basic system looks for the nearest to the origin of the ride non-empty queue • If the queue exists the basic system sends to the first cab driver in that queue the notification about the pending reservation • The basic system stores the outcome of the research
EXIT CONDITIONS	The basic system stores the outcome of the research
EXCEPTIONS	If the sent information are invalid the basic system sends back an error message.

[UC11] Delete Arrangement.

NAME	**Delete Arrangement**
ACTORS	Cab Catcher
ENTRY CONDITIONS	The Cab Catcher, who is already logged into the system, wants to delete an arrangement so he clicks on the delete arrangement button
FLOW OF EVENTS	• The system shows the list of all the active arrangements • The cab catcher chooses the arrangement to erase • If the arrangement is already designated then the **Delete Designated Arrangement** use case starts • Otherwise the system erases the arrangement
EXIT CONDITIONS	The arrangement is deleted
EXCEPTIONS	

[UC12] API Based System Delete Arrangement.

NAME	**API Based System Delete Arrangement**
ACTORS	API Based System
ENTRY CONDITIONS	The API Based System, that has already gained the programmatic interface, wants to delete an arrangement on behalf of a Cab Catcher through the programmatic interface itself
FLOW OF EVENTS	• The API based system sends to the basic system the information that identifies the arrangement to be deleted • If the arrangement is already designated then the **Delete Designated Arrangement** use case starts • Otherwise the system erases the arrangement
EXIT CONDITIONS	The arrangement is deleted
EXCEPTIONS	If the information that identifies the arrangement is invalid, the basic system sends back an error message.

[UC13] Delete Designated Arrangement.

NAME	**Delete Designated Arrangement**
ACTORS	Cab Driver
ENTRY CONDITIONS	The Cab Catcher, which is already logged, or the API Based System, which has already gained the programmatic interface, has chosen to delete an already designated arrangement
FLOW OF EVENTS	• The system deletes the chosen arrangement • The system informs the cab driver who was responsible for the deleted arrangement and it switches his availability
EXIT CONDITIONS	The cab driver has been informed and his availability has been switched
EXCEPTIONS	

[UC14] Set Availability.

NAME	**Set Availability**
ACTORS	Cab Driver
ENTRY CONDITIONS	The Cab Driver, who is already logged into the system, needs to change his availability state
FLOW OF EVENTS	• The cab driver clicks on the change availability button • The system switches the availability account state of the cab driver • If the state switches from not available to available then the system assignes 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	

[UC15] Decline Request.

NAME	**Decline Arrangement**
ACTORS	Cab Driver, Cab Catcher
ENTRY CONDITIONS	The Cab Driver, who is already logged into the system, receives an arrangement notification and wants to decline it
FLOW OF EVENTS	• The cab driver clicks on the decline button • The system puts the cab driver which has just declined the arrangement into the last position of the queue • If the time-out of the arrangement is expired and the arrangement doesn't come from an API based system, then the system sends a service unavailable message to the cab catcher. Otherwise, if the time-out is not expired, the system spots the nearest to the position non-empty queue which doesn't contain in first position the cab driver that has just declined the arrangement. • If that queue exists, the system sends a notification about the pending arrangement to the first cab driver in that queue • Otherwise, the system sends a service unavailable message to the cab catcher
EXIT CONDITIONS	The system stores the result of research

EXCEPTIONS	

[UC16] Accept Request.

NAME	**Accept Arrangement**
ACTORS	Cab Driver
ENTRY CONDITIONS	The Cab Driver, who is already logged into the system, receives an arrangement notification and wants to accept it
FLOW OF EVENTS	• The cab driver clicks on the accept button • The system switches his account state to unavailable • If the arrangement doesn't come from an API based system, the **Information Computation and Sending** use case starts
EXIT CONDITIONS	Either the information about the estimated arrival time and the recognition code are sent to the cab catcher or the arrangement status is updated to accepted
EXCEPTIONS	

[UC17] Information Computation and Sending.

NAME	**Information Computation and Sending**
ACTORS	Cab Catcher, Cab Driver
ENTRY CONDITIONS	The Cab Driver, who is already logged into the system, accepts to handle an arrangement
FLOW OF EVENTS	• The system carries out the estimated arrival time • The system sends the code and the estimated arrival time to the cab catcher itself
EXIT CONDITIONS	The information are sent to the cab catcher
EXCEPTIONS	

[UC18] Get Programmatic Interface.

NAME	**Get Programmatic Interface**
ACTORS	API Based System
ENTRY CONDITIONS	The API Based System wants to obtain the programmatic interface
FLOW OF EVENTS	• The API based system starts the **Get**

	Authorization use case • The API based system uses the authorization obtained to ask for the programmatic interface • The system returns the programmatic interface
EXIT CONDITIONS	The API based system withholds the programmatic interface
EXCEPTIONS	

[UC19] Get Authorization.

NAME	**Get Authorization**
ACTORS	API Based System
ENTRY CONDITIONS	The API Based System needs the authorization to get the programmatic interface
FLOW OF EVENTS	• The API based system asks its user for the authorization to get the programmatic interface by starting the **Log In** 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

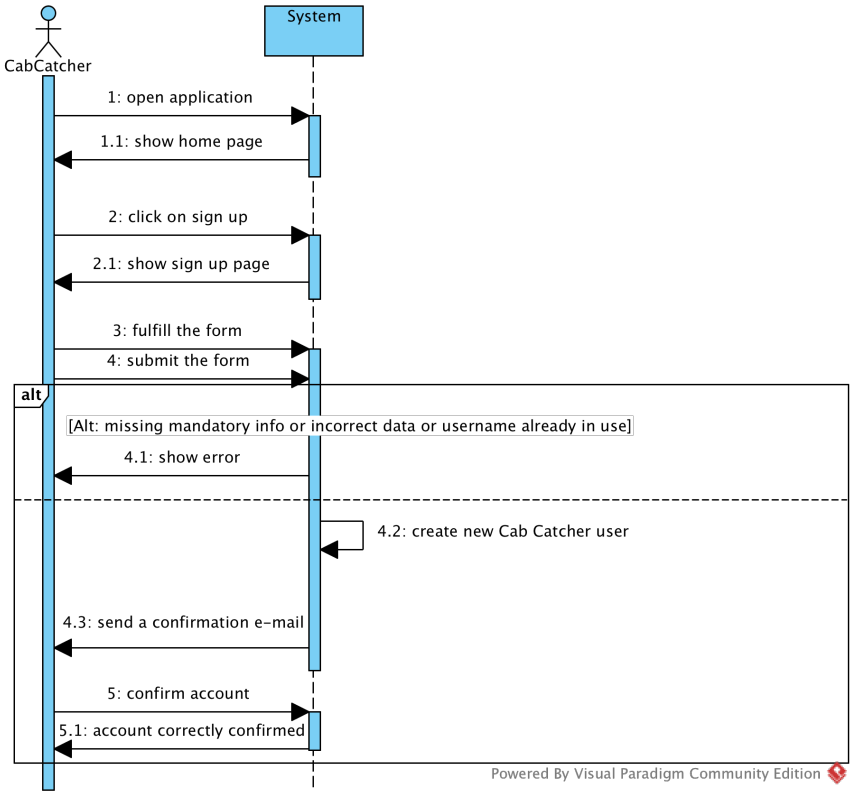
[UC20] Check Arrangement Status.

NAME	**Check Arrangement Status**
ACTORS	API Based System
ENTRY CONDITIONS	The API Based System, that has already gained the programmatic interface, wants to check the status of an arrangement previously made, through the programmatic interface itself
FLOW OF EVENTS	• The API based system asks for the status of a particular arrangement • The basic system answers with the current status of the arrangement
EXIT CONDITIONS	The API based system has received the current status of the arrangement
EXCEPTIONS	If the information that identifies the arrangement is invalid, the basic system sends back an error message.

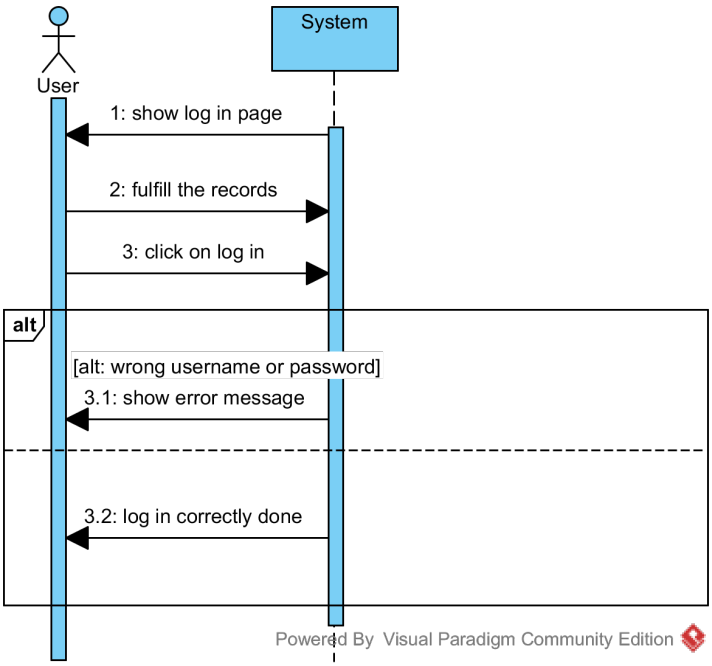
3.4.2. Sequence Diagrams

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

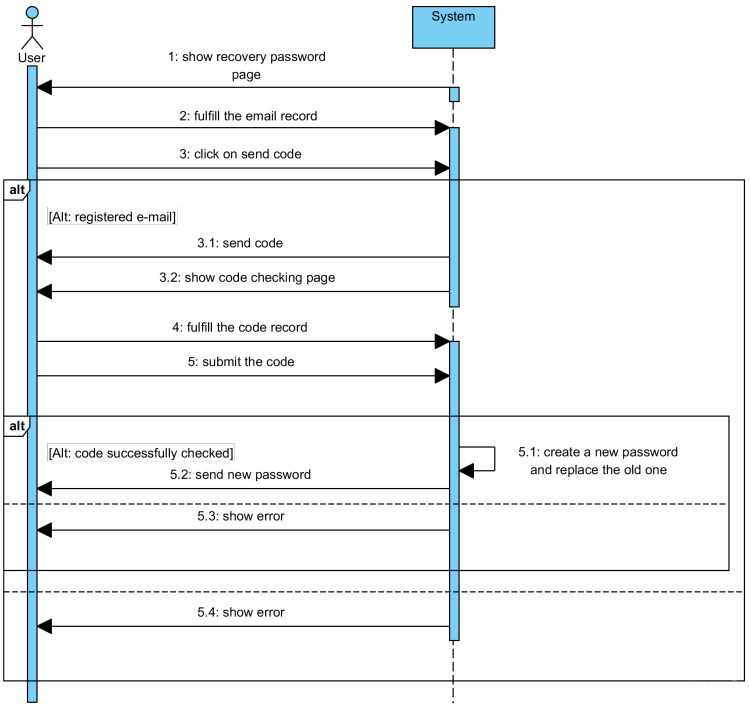
[SD1] Sign Up.



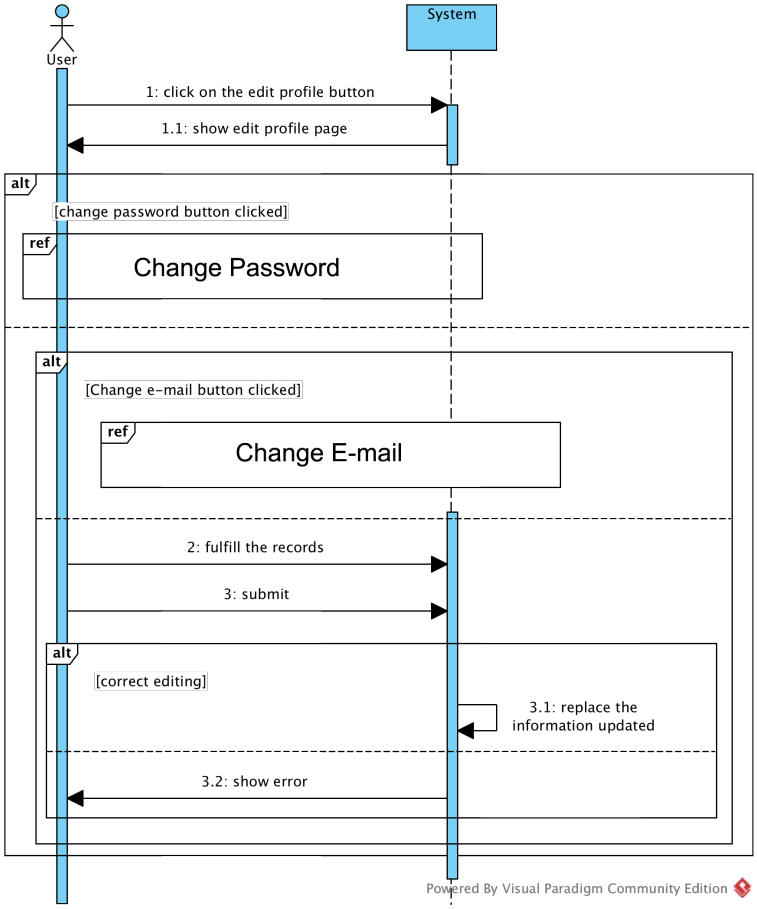
[SD2] Log In.



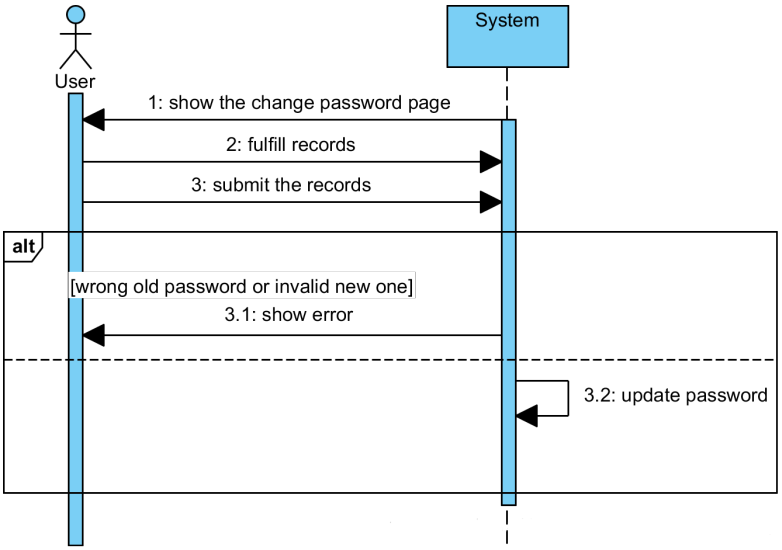
[SD3] Password Recovery.



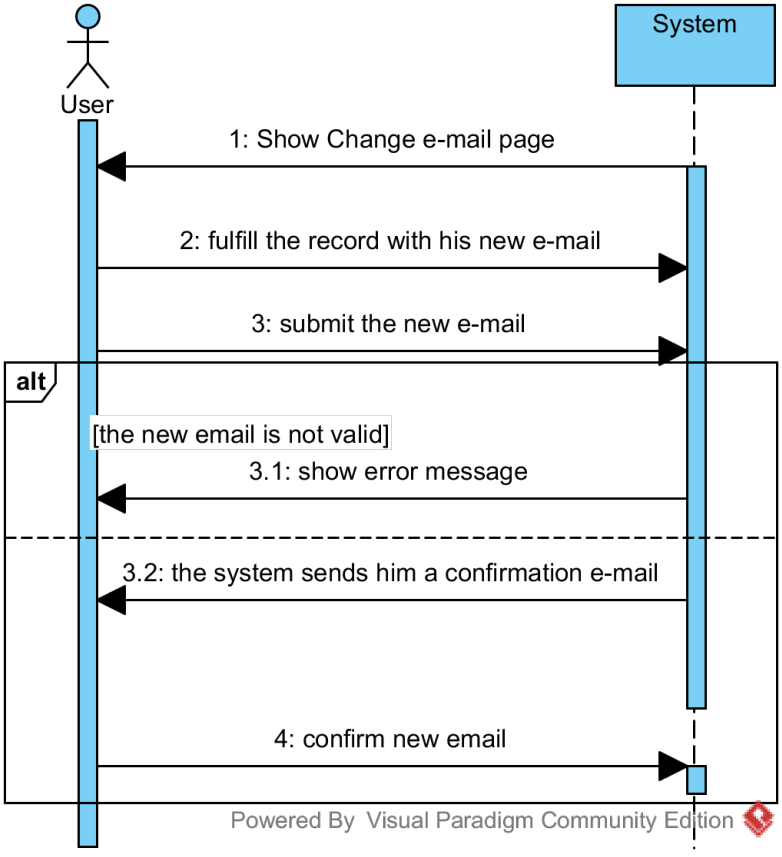
[SD4] Edit Profile.



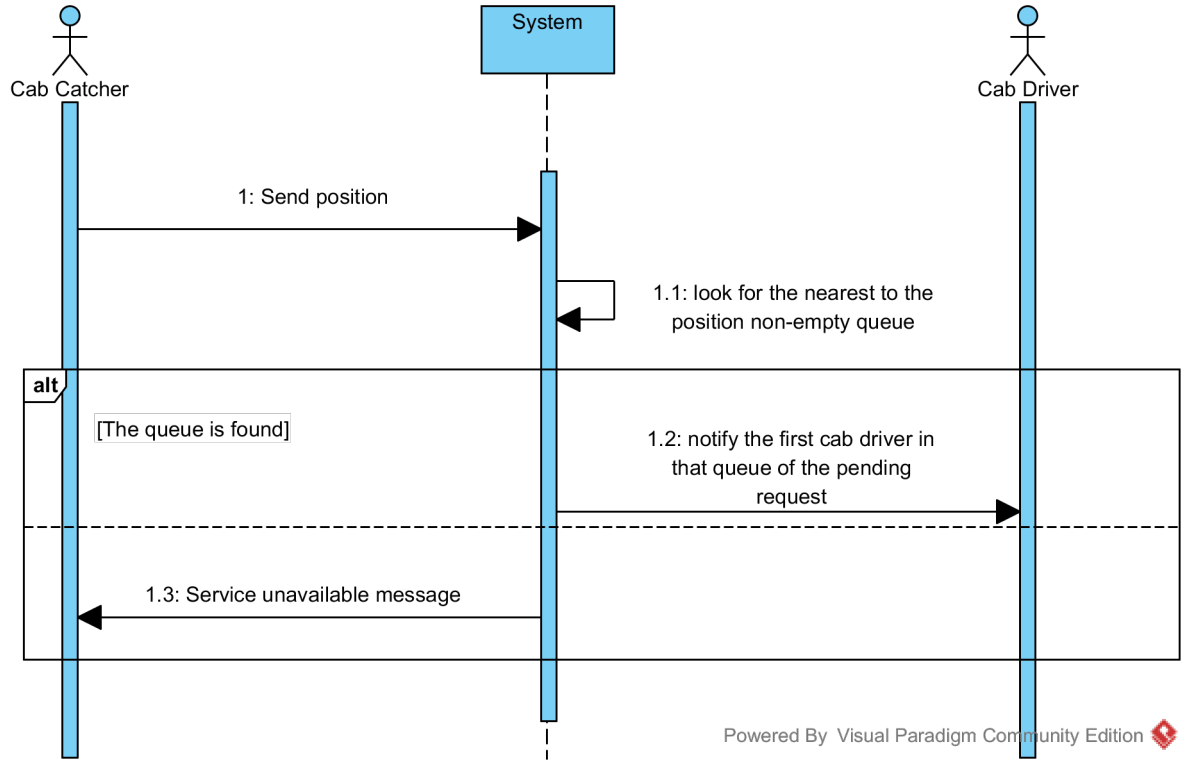
[SD5] Change Password.



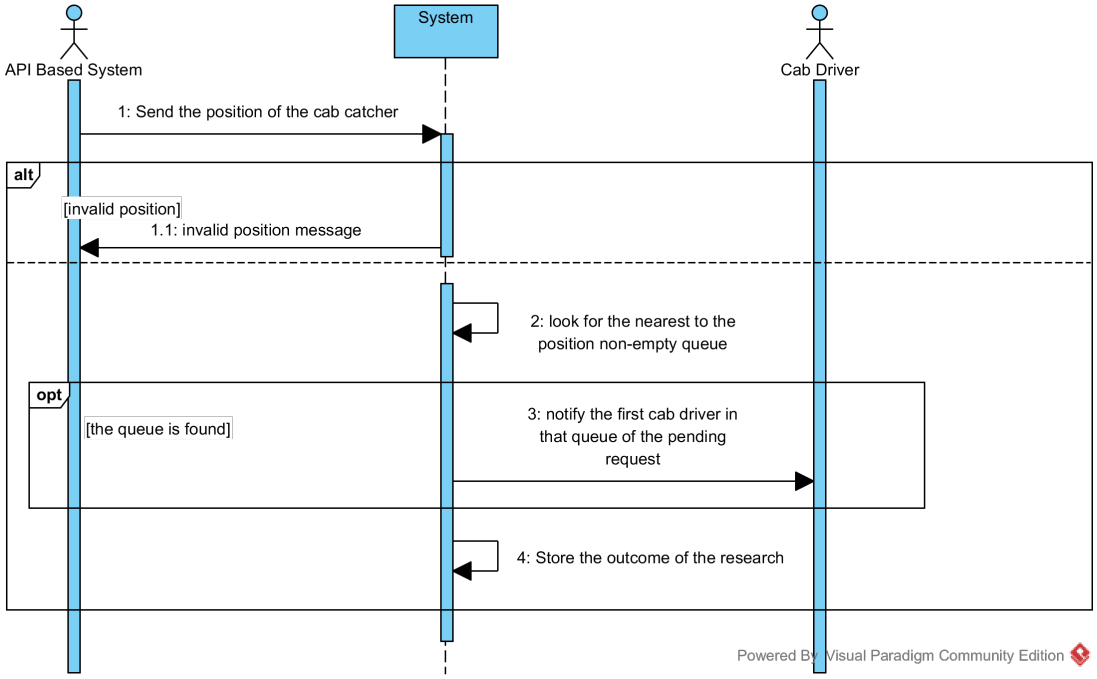
[SD6] Change E-mail.



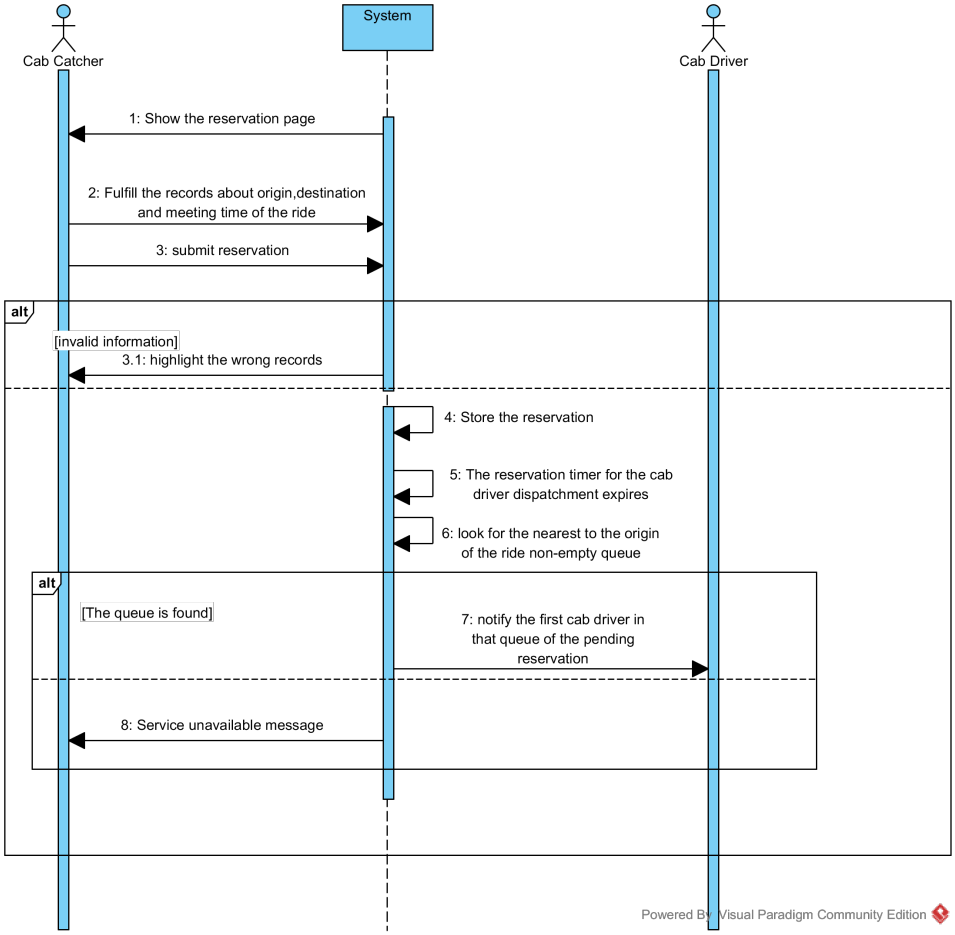
[SD7] Request.



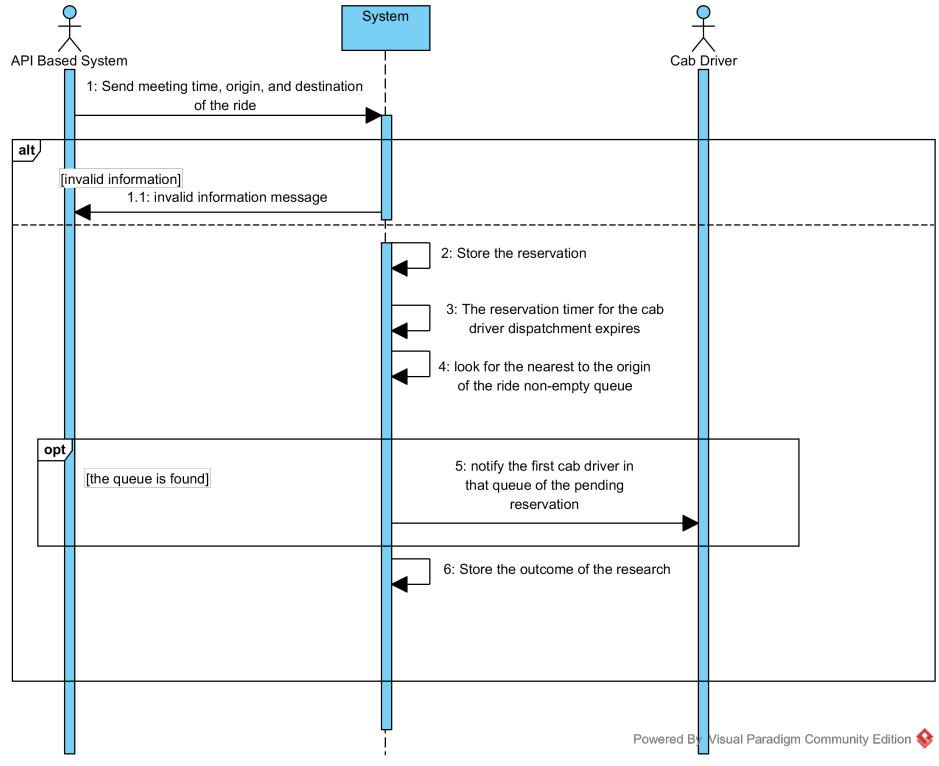
[SD8] API Based System Request.



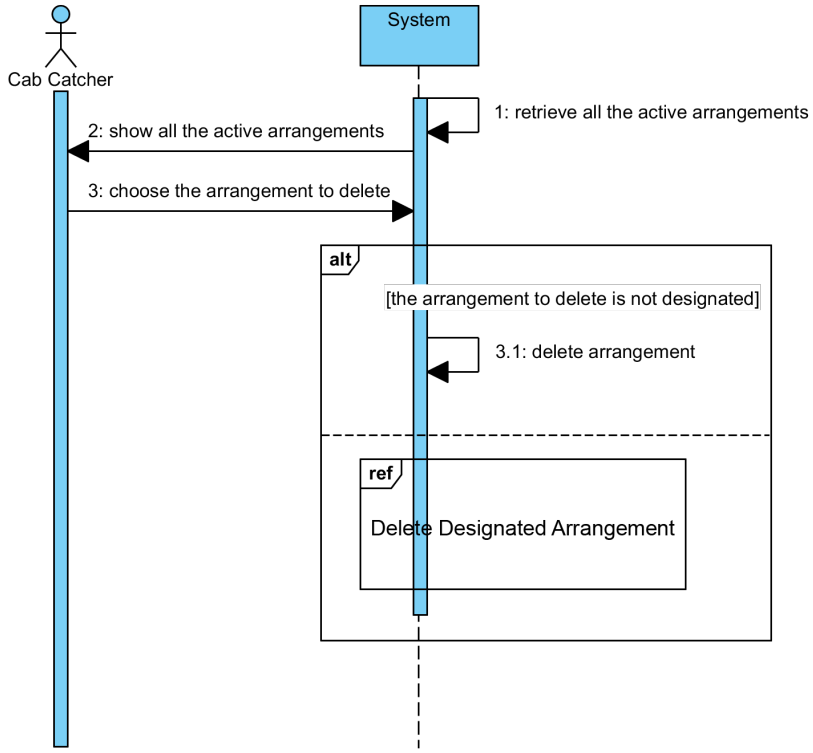
[SD9] Reservation.



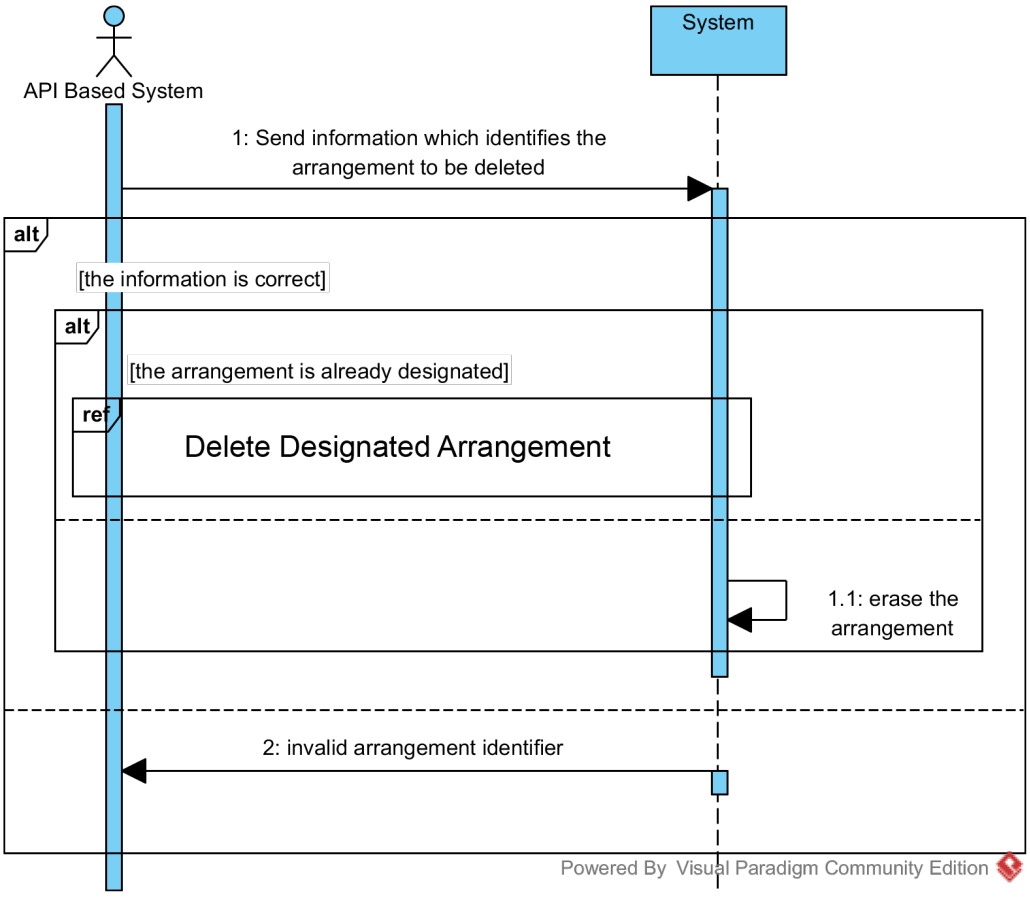
[SD10] API Based System Reservation.



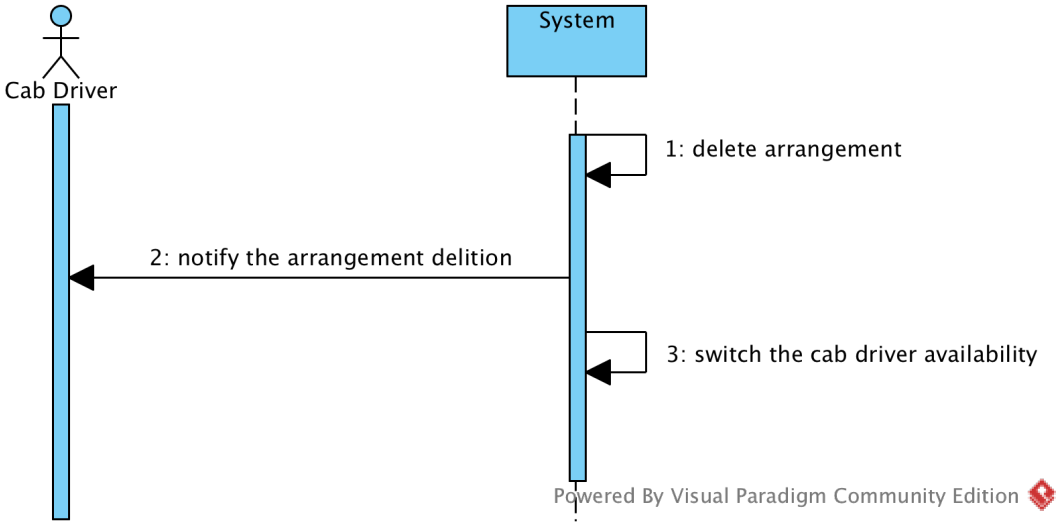
[SD11] Delete Arrangement.



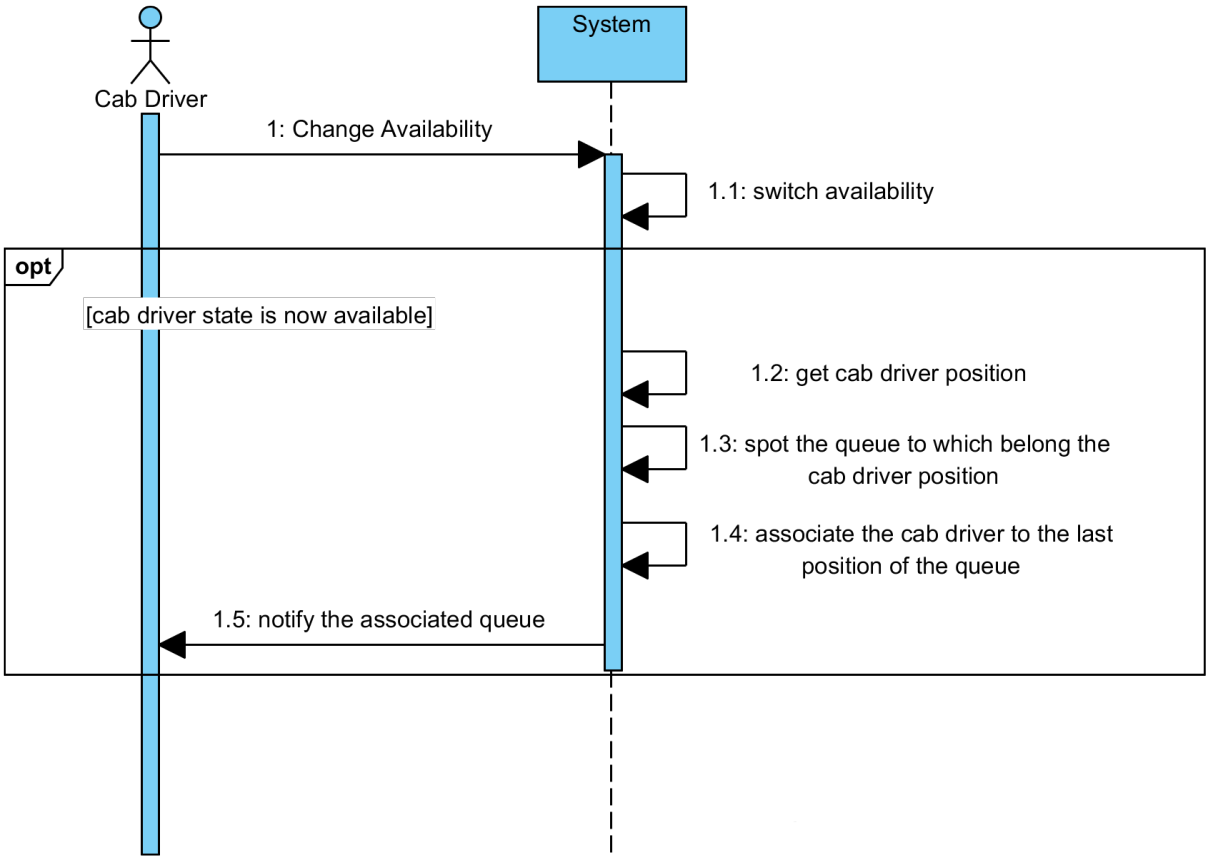
[SD12] API Based System Delete Arrangement.



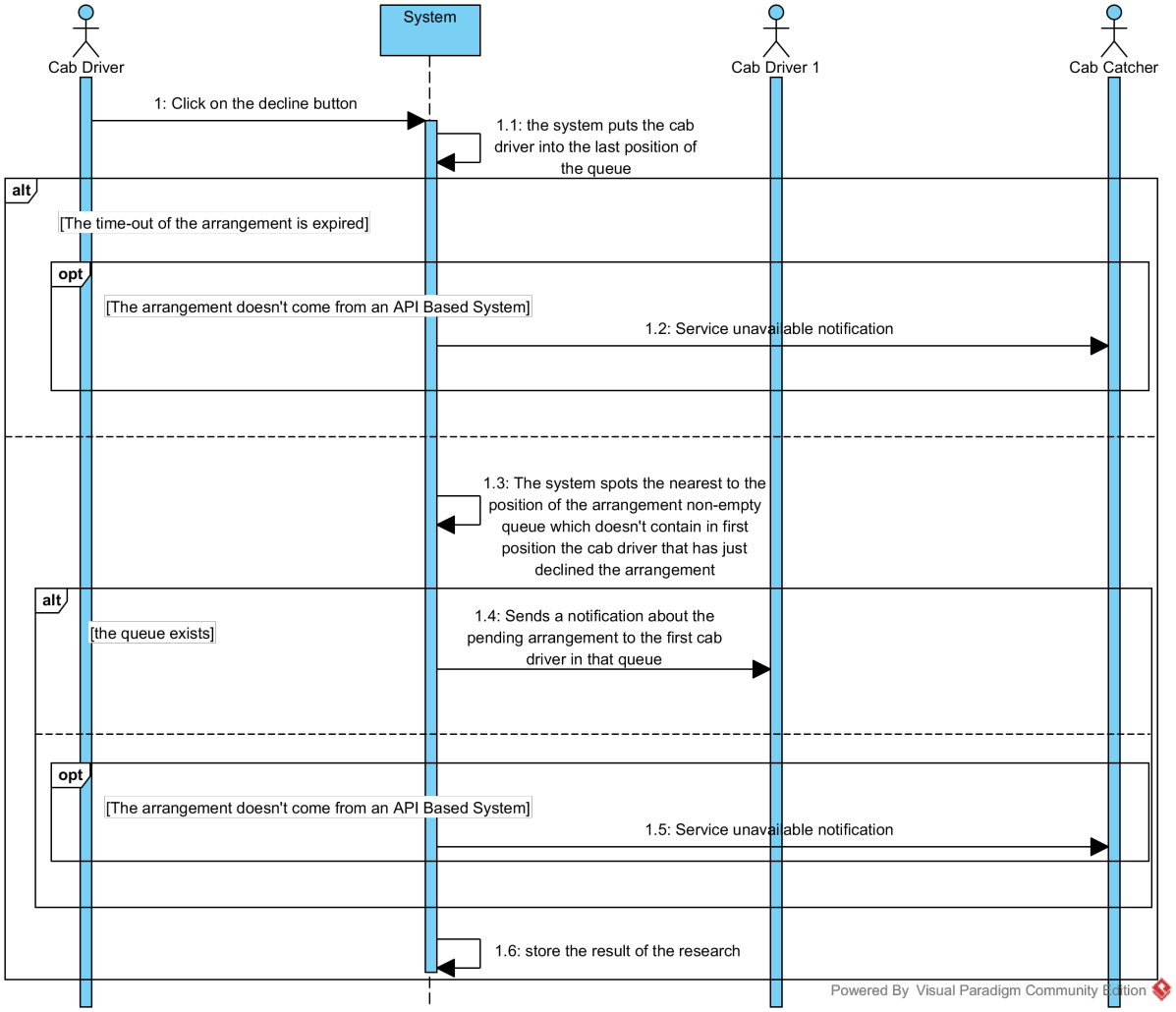
[SD13] Delete Designated Arrangement.



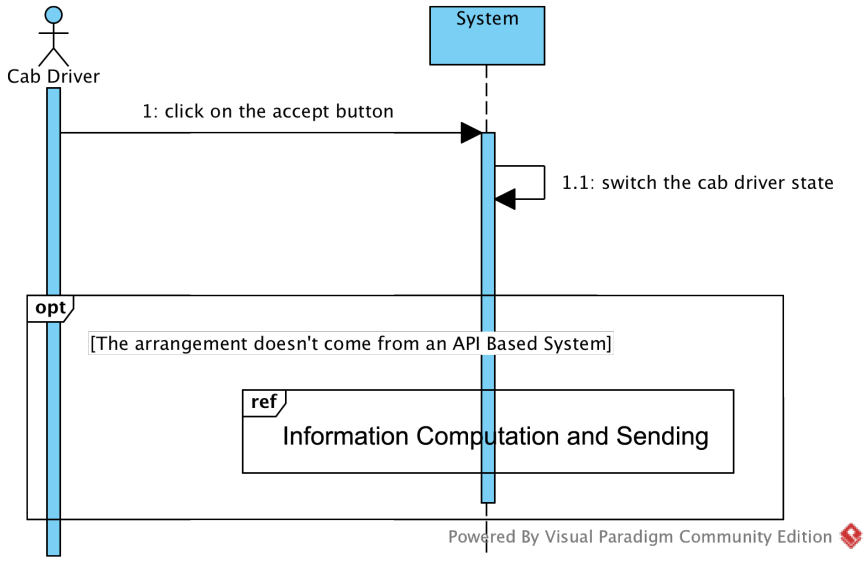
[SD14] Set Availability.



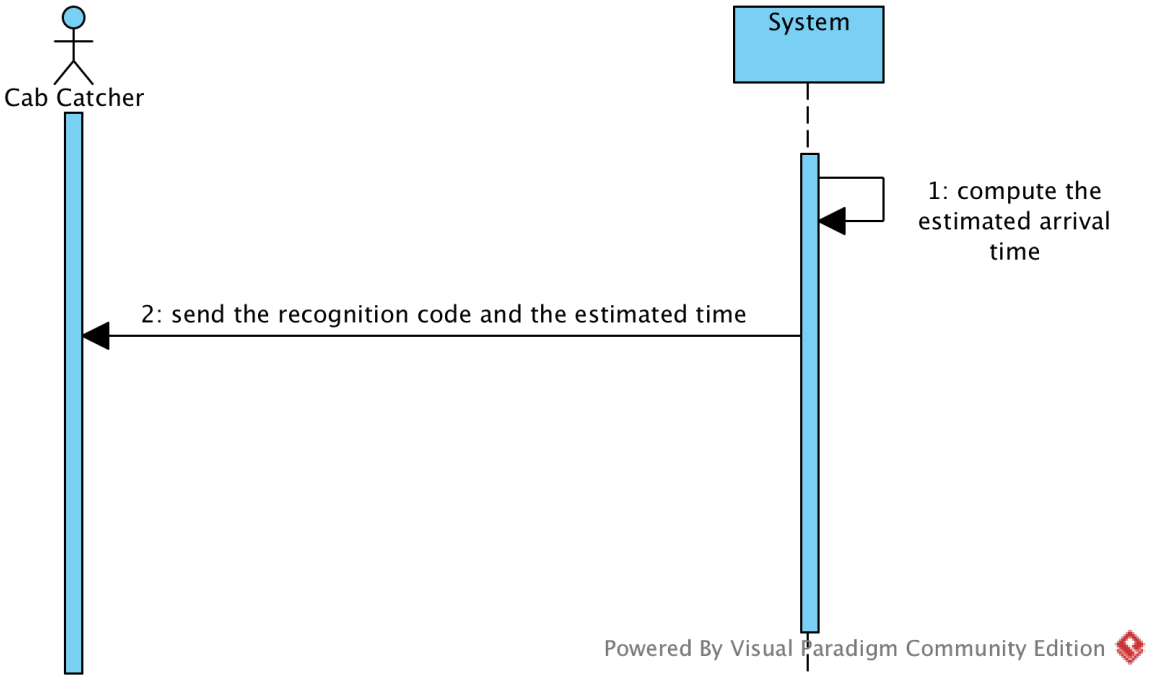
[SD15] Decline Arrangement.



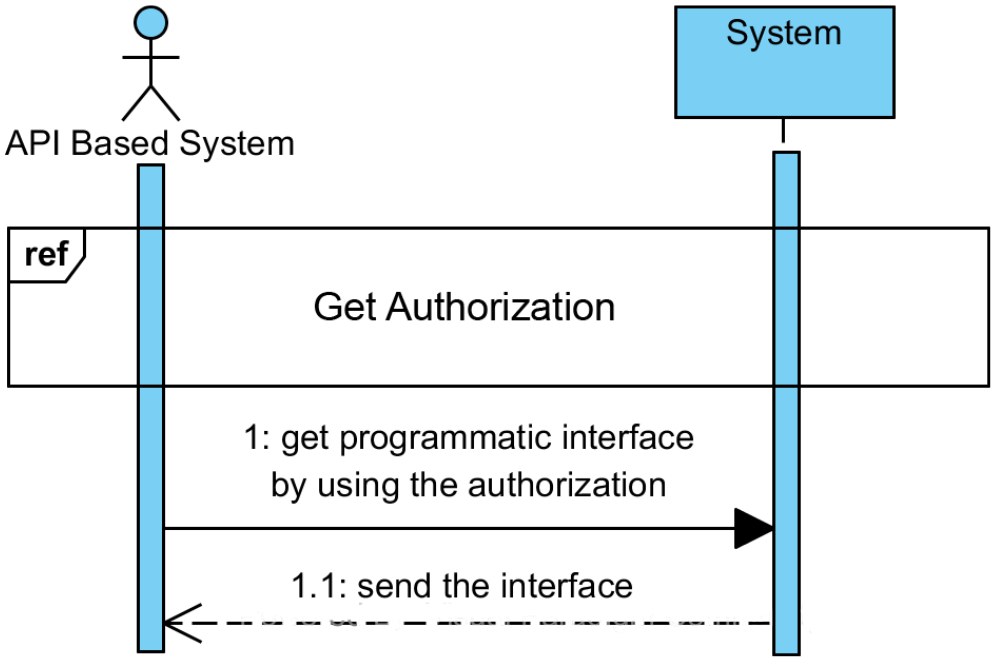
[SD16] Accept Arrangement.



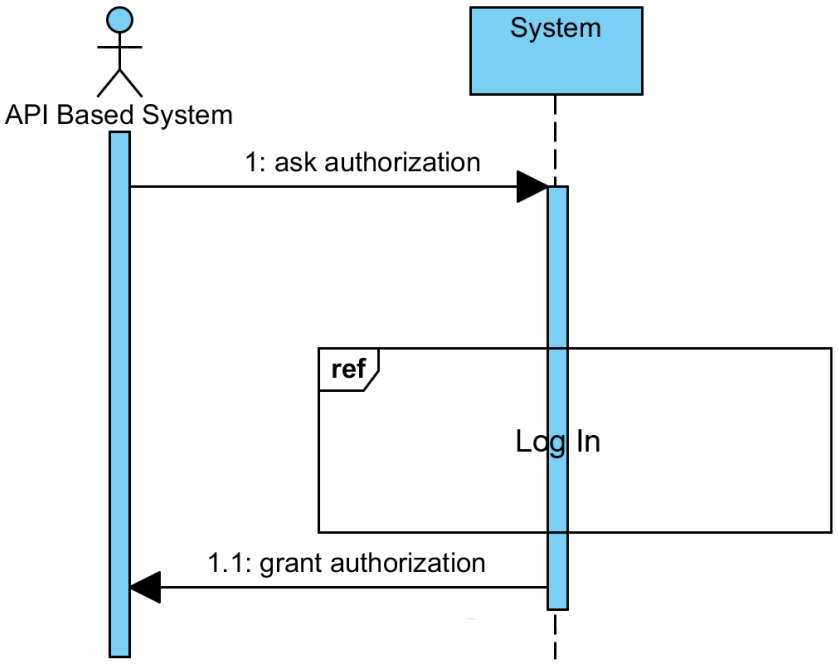
[SD17] Information Computation And Sending.



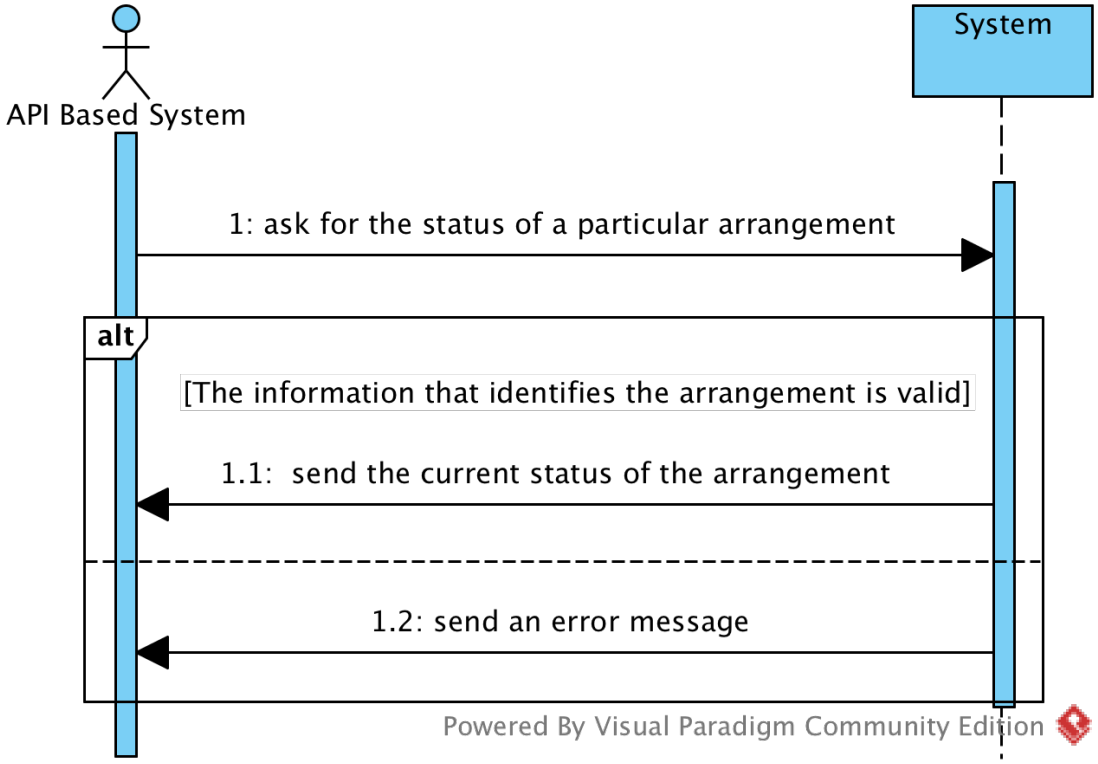
[SD18] Get Programmatic Interface.



[SD19] Get Authorization.

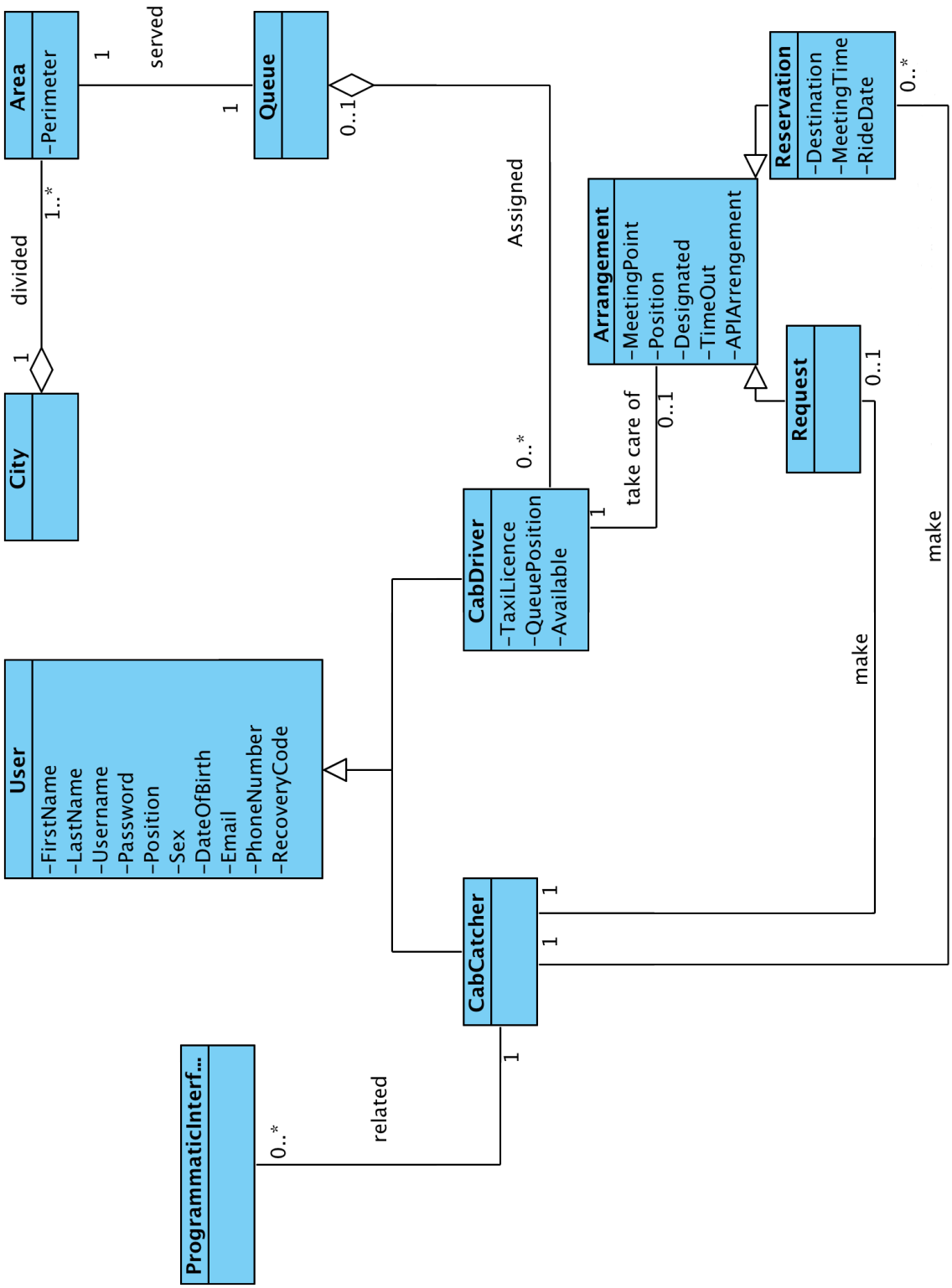


[SD20] Check Arrangement Status.



3.4.3. Class Diagram

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



3.5. Alloy Model

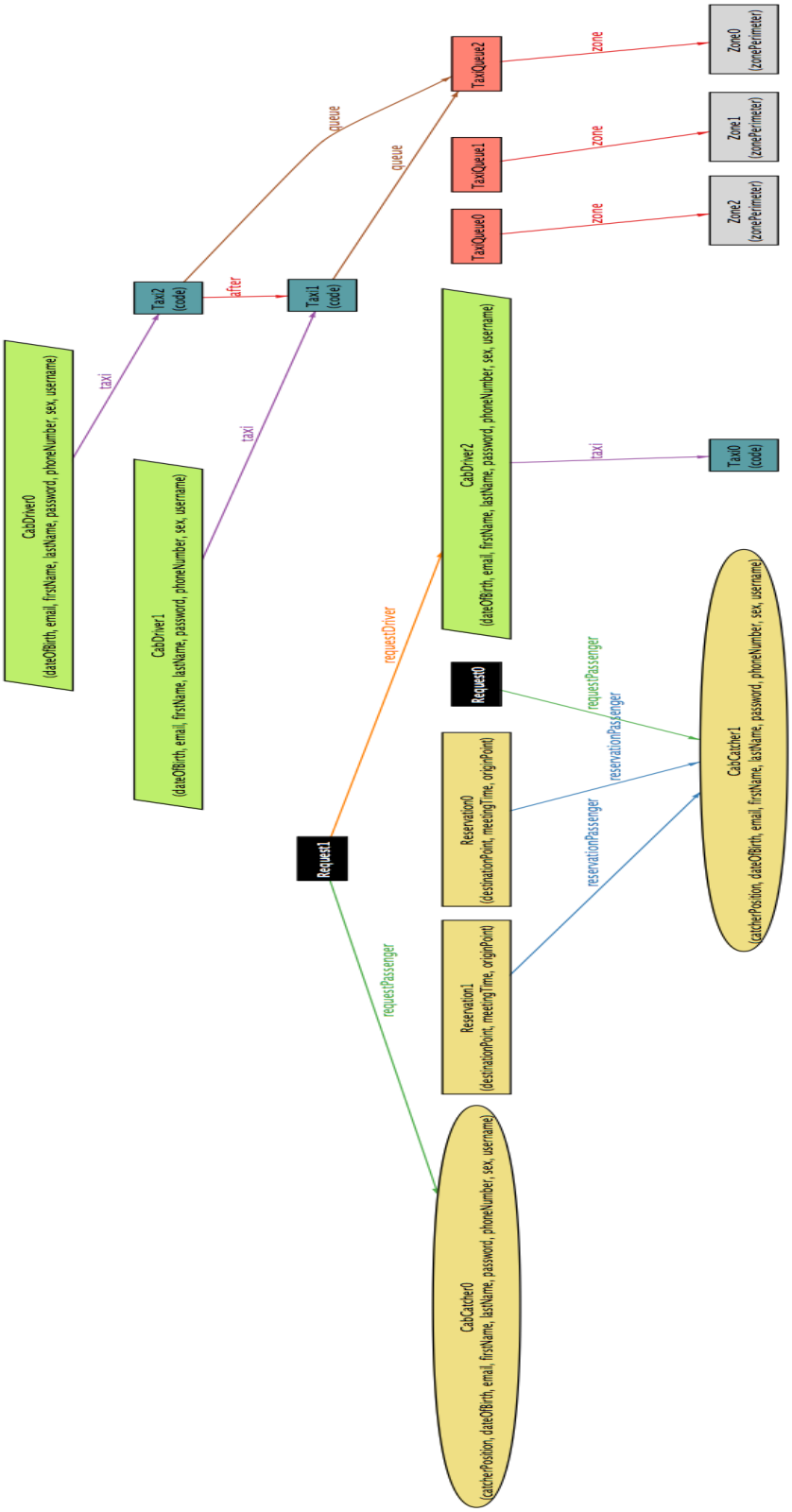
In the section below it's shown the Alloy Analyzer code used to generate and analyse a model representing the system interaction with the world. This model has been derived from the Class Diagram.

```
//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,
      requestAPI: one Boolean,
      timeOut: one DateTime
      }

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

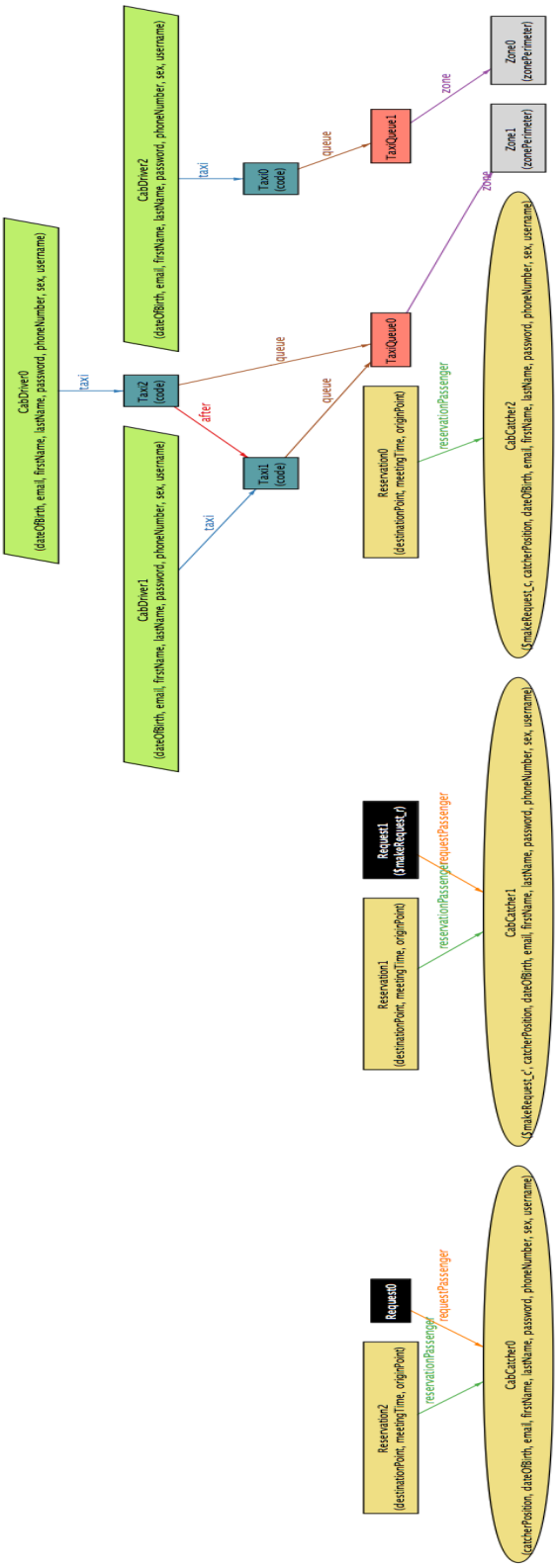
```
//Position is a data type associated with cab catchers, it has to be present only if there is at least a cab catcher
fact{
	some CabCatcher implies #Position = 1 else #Position = 0
	}

//DateTime is a data type associated with reservations, it has to be present only if there is at least a reservation
fact{
	(some Reservation or some Request) implies (#DateTime = 1 and #Boolean = 1) else (#DateTime = 0 and #Boolean = 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 taxies 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
pred makeRequest[c, c': CabCatcher, r: Request]{
	c != c'
	#c.~reservationPassenger = #c'.~reservationPassenger
	r.requestDriver = none
	c'.~requestPassenger = c.~requestPassenger + r
	}

run makeRequest for 3

//Predicate to make a reservation
pred makeReservation[c, c': CabCatcher, r: Reservation]{
	c != c'
	r.reservationDriver = none
	#c.~requestPassenger = #c'.~requestPassenger
	c'.~reservationPassenger = c.~reservationPassenger + r
	}

run makeReservation for 5

//Predicate to delete a request
pred deleteRequest[c,c': CabCatcher, r: Request]{
	r in (c.~requestPassenger)
	c'.~requestPassenger = (c.~requestPassenger - r)
	}

run deleteRequest for 5

//Predicate to delete a reservation
pred deleteReservation[c,c': CabCatcher, r: Reservation]{
	r in (c.~reservationPassenger)
	c'.~reservationPassenger = (c.~reservationPassenger - r)
	}

run deleteReservation for 2


//Predicate to allocate a reservation to a cab driver
pred 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
	}

run allocateReservation for 5

//Check that the number of request is lower than or equal to the number of cab catchers
assert lowerRequests{
			#Request <= #CabCatcher
		}

check lowerRequests for 5

//Check the makeRequest predicate
assert 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
		}

check addRequest for 5

//Check the makeReservation predicate
assert addReservation{
		all c1, c2: CabCatcher, r: Reservation | makeReservation[c1,c2,r] implies (r in (c2.~reservationPassenger)) and (r not in (c1.~reservationPassenger))
		}

check addReservation for 5

//Check the deleteRequest predicate
assert cancelRequest{
			all c1, c2: CabCatcher, r: Request | deleteRequest[c1,c2,r] implies (r not in (c2.~requestPassenger)) and c1 != c2
			}

check cancelRequest for 5
```

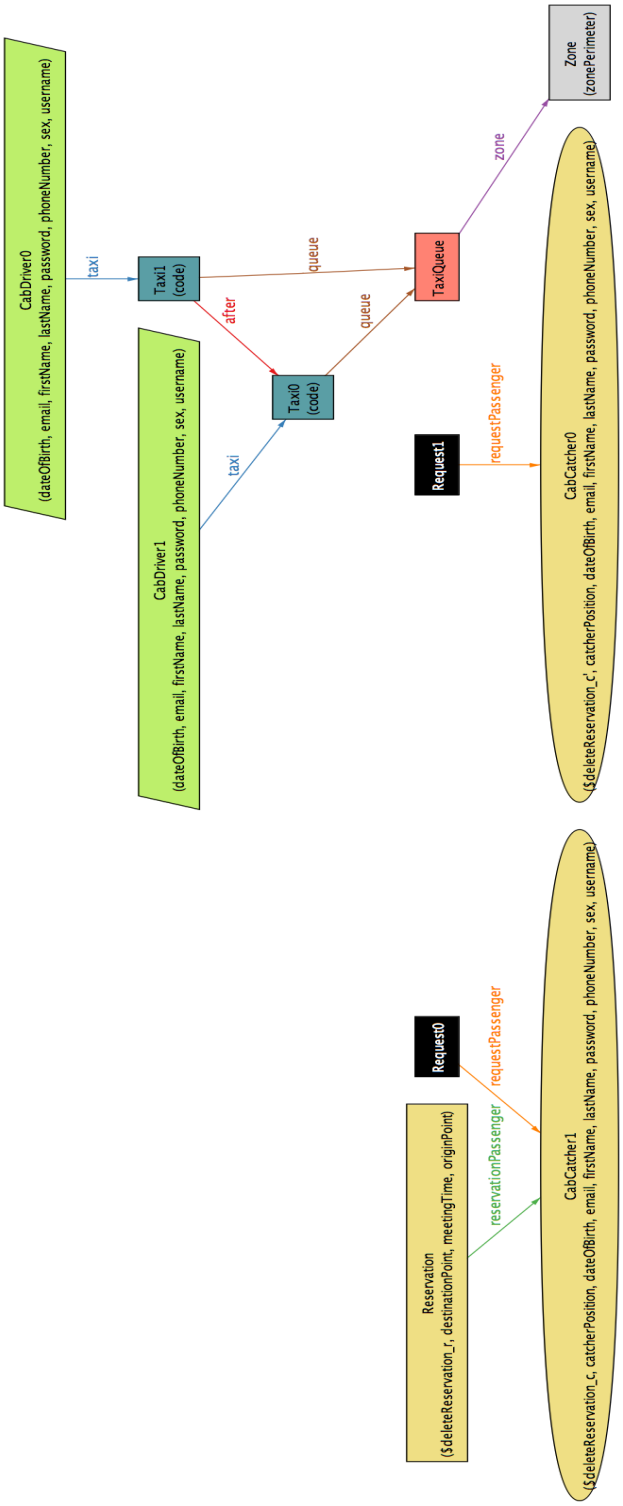
The following screenshot represents the world generated by using the **run showWorld** command:



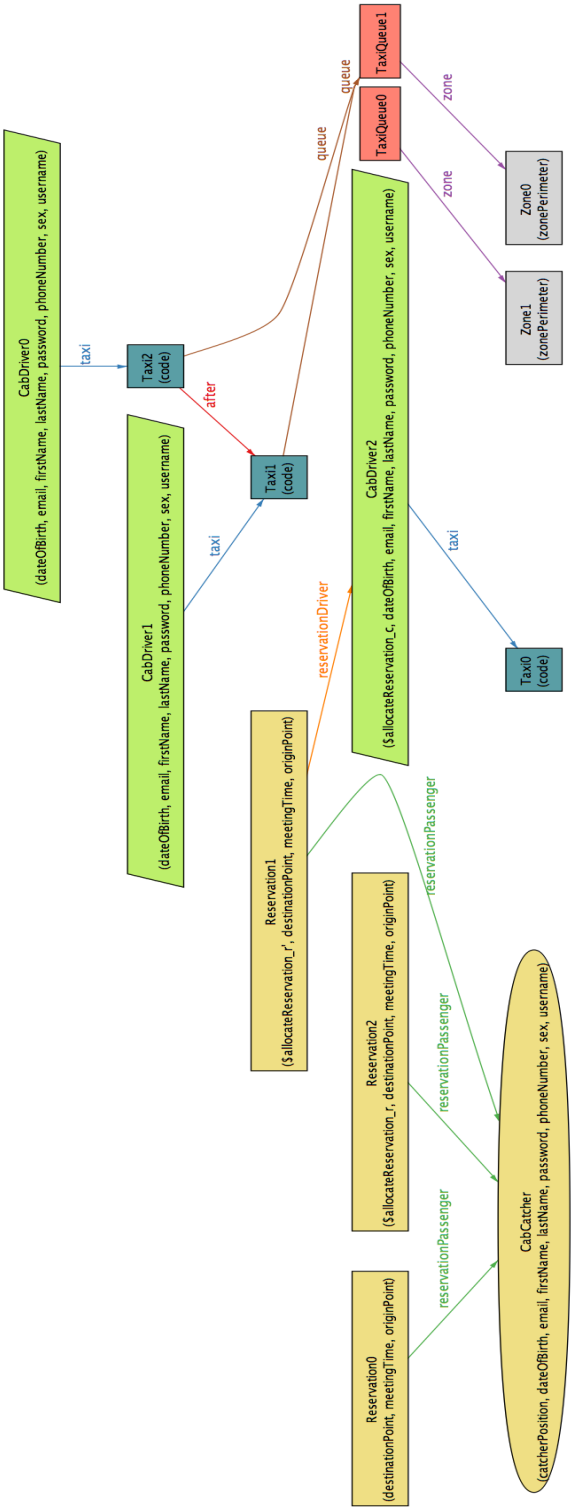
The following screenshot represents the world generated by using the **run makeRequest** command:



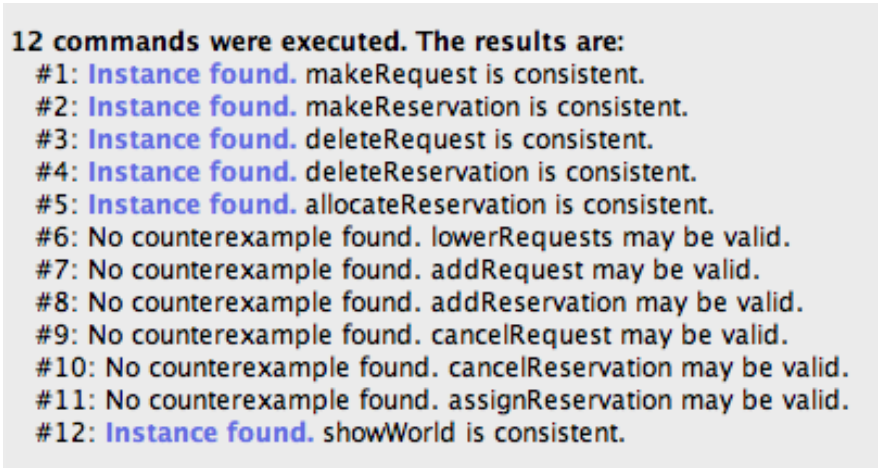
The following screenshot represents the world generated by using the **run deleteReservation** command:



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



The following screenshot represents the analysis result:



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.