DESIGN DOCUMENT

myTaxiService

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

Andrea Binelli
858235

Sommario

1. Introduction

1.1 Description of the document

In this document I focused on the Data design and on how the interaction between our system and the users should be.

The scope of this document is to provide the design of the application, describing and justifying the reason of our choices, for example the Data Base schema, the User Experience that our application should provide to our users and the main classes that I will make on the Implementation phase.

The Data Base design is essential to build a consistent Data Base that supports in an efficient way all the queries and the manipulations on the data.

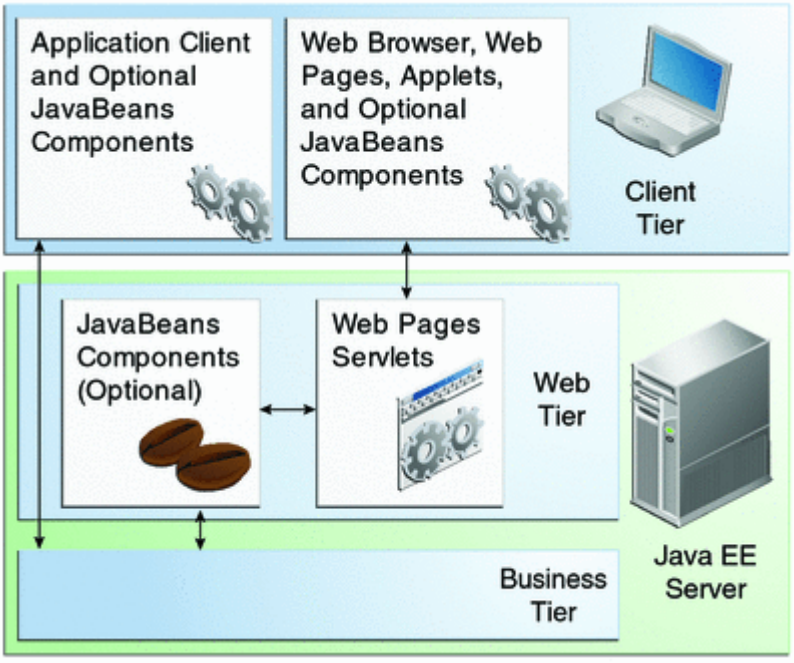
It must be specified that the details of the implementation will be precisely defined on the implementation phase, so what I give is a general idea of my software.

1.2 Architecture Description

For the application I will use the JEE Architecture.

The JEE has a four-tiered architecture divided as:

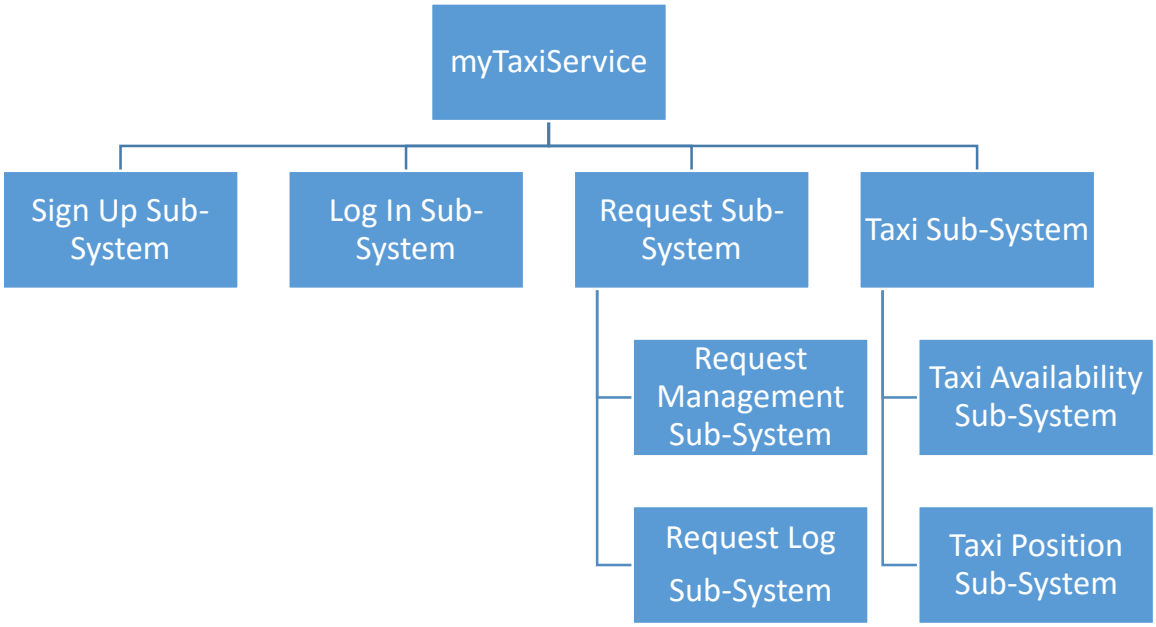
- Client: manages the application User Interface and directly interacts with users and taxi drivers.
- Web/App: manages the communication between the Client tier and the Business tier
- Business logic: manages all the business logic of the application
- EIS: contains the data source and manages the operations of recovering data from the database and writing data on it.



1.3 Identifying Subsystems

I decided to decompose our systems in other sub-systems to figure the functionalities of my system out and to make them clearer. I separate our systems into these sub-systems:

- Sign Up Sub-System
- Log In Sub-System
- Request Sub-System
    - Request Management Sub-System
    - Request Log Sub-System
- Taxi Sub-System
    - Taxi Availability Sub-System
    - Taxi Position Log Sub-System



2. Persistent Data Management

I decided to store application's data in a relational database, because it has to manage persistent information that must be memorized durably. This paragraph explains my choices in the conceptual design of the database, using an ER diagram; then it describes the process I have gone through in order to design the relational model of the database.

2.1 Conceptual Design

In this first phase I have designed the conceptual model of the database, a representation of the main entities and relations that are involved in the application data model.

This section explains in a clearer way how the ER diagram is constructed (only the main aspects and those that are difficult to understand from the diagram itself).

At the end I have reported the complete ER diagram.

The central entity of my application domain is the *Request*. Is it associated to each other by means of three relations:

- *Create*, which represents the creation of a Request by an user
- *Accept*, which represents the acceptance of a request by a taxi
- *Is Located*, which represent the address of the request where the taxi is requested. I have made the choice to have an entity that contains all the addresses of the city (that is periodically updated) in order to associate the request to the proper zone.

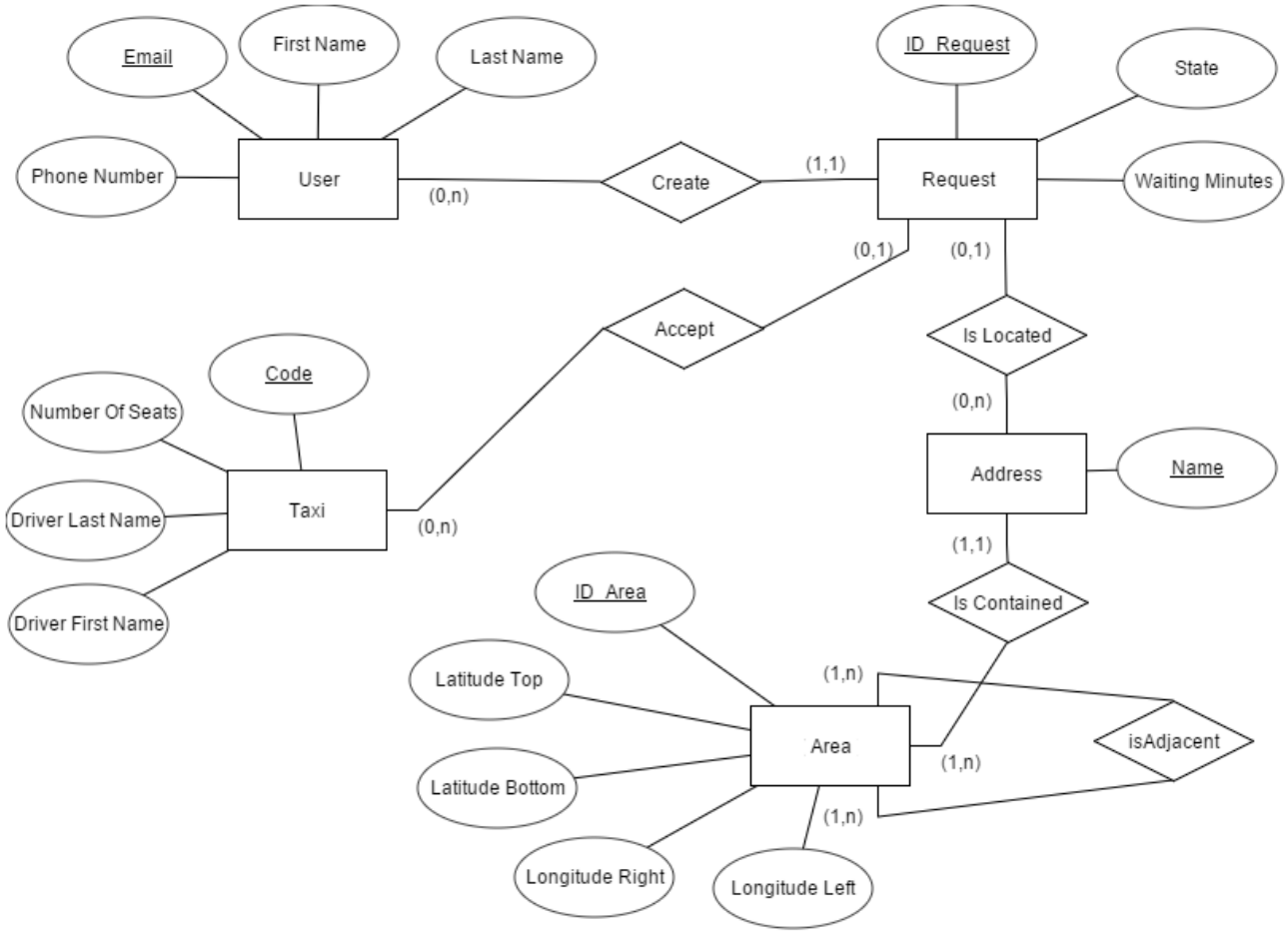
The *User* entity contains all the attributes that describes a user and it is identified via email. The fact that an user can also log in by his phone number does not imply that the User must be identified also with the number (because is a feature that is not mandatory).

Taxi is an entity that contains both the information about the physical car (number of seats and code), the driver (first name and last name).

Area, involves both the queue of free taxis and the area (because there is a 1 to 1 relation between the queue and the zone). With the premise that an area can be only a rectangle (another assumption contained in the RASD), I have to specify four lines (the coordinates) in order to specify a geographical area.

I have made the decision to remove the relation between Taxi and Queue/Area (from the Class Diagram in the RASD) because the managing of the queue is made at runtime and it does not require being stored in a database. Therefore, I have changed the name from Queue to Area for a better denomination.

The complete ER Diagram:



2.2 Logical Design

The aim of this phase is to develop a relational schema of the database starting from the conceptual design. In this section I will discuss the main project decisions I have taken during this step and the final result, which is the schema of the database including the specifications of all the tables (with their fields and fields' types) and the most significant constraints in order to make data consistent with respect to domain properties.

2.2.1 Entities and Relations translation

In our ER schema, the entity user is identified by means of the *Email* attribute, but in the logical schema, I have decided to use an incremental integer identifier. Of course, I force the uniqueness of the *Email* (and the phone number) by declaring it as unique field.

The same as above happens to the entity *Taxi* (by replacing the identifier from *Code* to an incremental identifier) and to the entity *Address* (by replacing the identifier from *Name* to an incremental id).

These choice are made also because the attribute *Code* can change during the life of a *Taxi* (for example when the physical taxi is replaced) and every attributes of the *User* can be changed manually by the application.

The identifiers of the entities become the primary keys of the tables and the attributes of the entities become the fields of the tables.

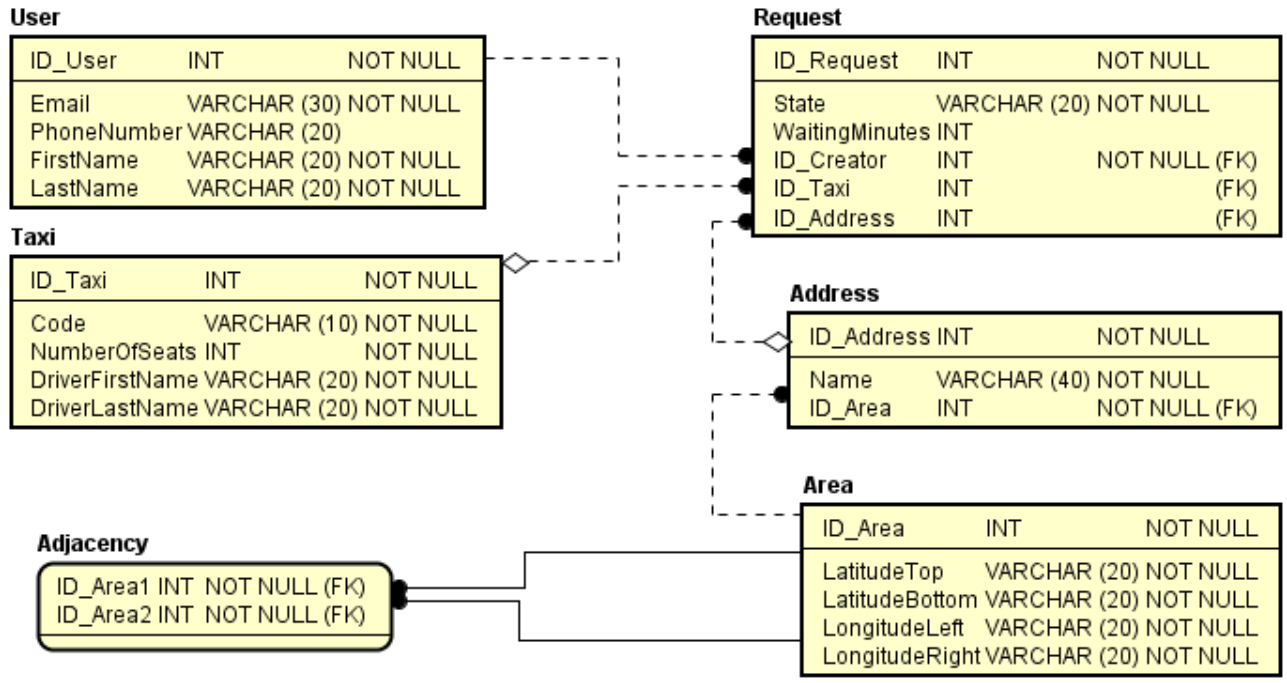
The next step is the translation of the relations; i've translated them as follows:

The *Create*, *Accept*, *Is Located* relations are translated by adding three foreign keys to the table *Request*, named *ID_Creator* (not null), *ID_Taxi* and *ID_Address*, referring respectively to the primary key of the table *User*, *Taxi*, *Address*.

The Is *Contained* relations is translated by adding a foreign key (not null) to the table *Address*, named *ID_Area*, referring to the primary key of the table *Area*.

The isAdjacent many-to-many relation between User and Event is translated by adding a new table to the schema, named Adjacency, which contains two foreign keys, named ID_Area1 and ID_Area2, referring both to the primary key of the table Area.

I have obtained the following relational model:



The final model has the following physical structure:

User(ID_User, Email, PhoneNumber, FirstName, LastName)

Request(ID_Request, ID_Creator, ID_Taxi, ID_Address, State, WaitingMinutes)

Address(ID_Address, ID_Area, Name)

Area(ID_Area, LatitudeTop, LatitudeBottom, LongitudeLeft, LongitudeRight)

Taxi(ID_Taxi, Code, NumberOfSeats, DriverFirstName, DriverLastName)

3.User Experience

In this paragraph I describe the user experience given by my system to the users and taxis by means of a UX (User Experience) diagram.

I have represented that diagram by extending the Class Diagram notation by means of appropriate stereotypes for every class:

- <<screen>> represents a complete tab (or a tab inside a web page) of our application.
- <<screen component>> represents a graphical component inside a page.
- <<input form>> represents a set of input fields or other input components inside a page.

The arrows represent the flow followed by the user in the application by performing a specific operation or by triggering a particular behaviour.

I have used composition and aggregation relations in order to represent the relationship among the components of a page.

I have decided to split my diagram into two parts in order to make it more readable and to focus on each function of our application.

3.1 Home

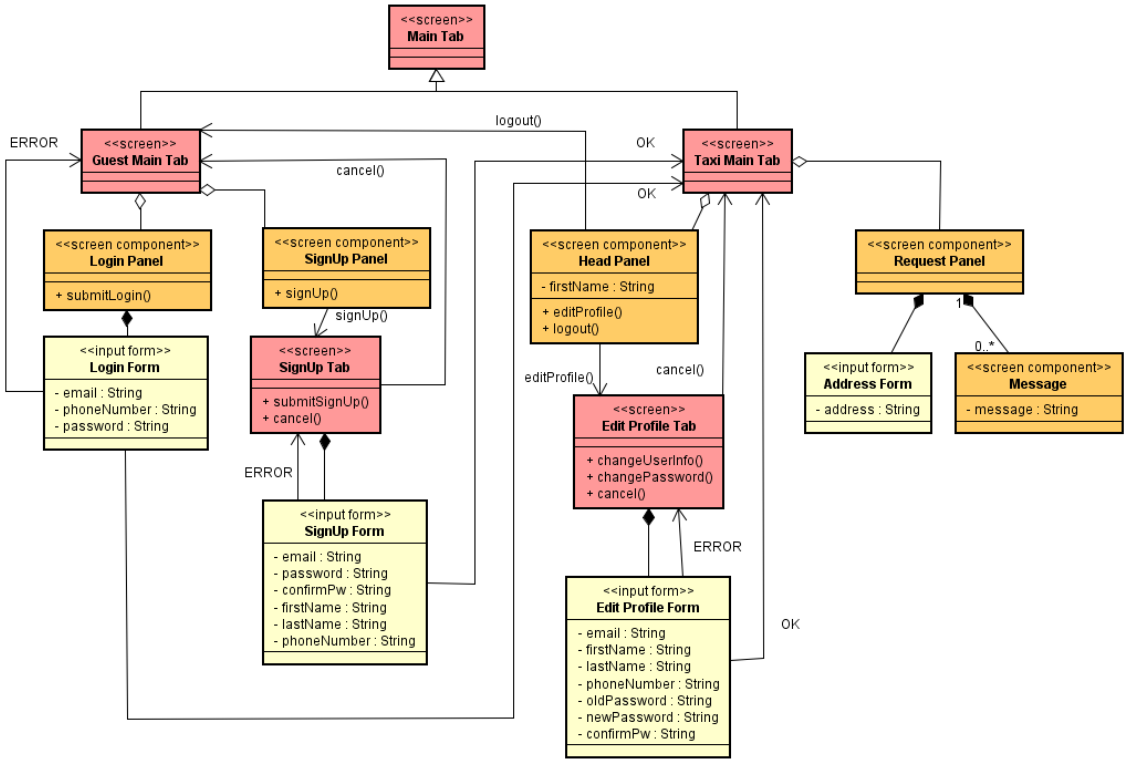
This part of the UX Diagram describes the main tab (or the homepage if it is visualized by browser), which can be shown in two different ways if the user is authenticated or if the user is just a guest.

If the user is not already logged in he sees the Guest Home where he can either sign up or log in. If the user does not have an account, he can click on the sign up button. He has to fill in the Sign Up form, then if every field of the form is correct and the email (and eventually the phone number) is not already used form someone else, the system shows him the User Main Tab. If the user already has an account, he has to fill in the Log In Form, click on the log in button and if his data are correct the system shows him the User Main Tab.

From the User Main Tab a user can make a call for a taxi and see the responses from the system. I suppose that old messages are visible for a while until they are deleted by the app so I represented this by a 0..* relation.

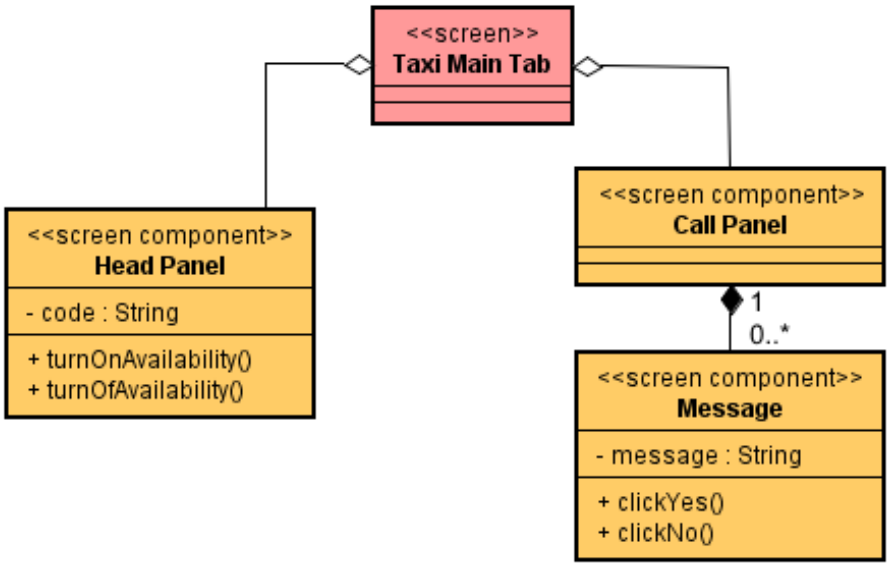
When he clicks on Edit Profile Data, another tab is shown to the user where he can click on Change user information or Change password.

When he clicks on Logout, the user return to the Guest Main Tab.



3.2 Taxi

This part of the UX Diagram describes the Taxi main tab. The taxi can only see his code, set his availability and answer to the calls received from the system in the form of messages (I suppose that old messages are visible for a while until they are deleted by the app so I represented this by a 0..* relation).



4. BCE Diagrams

I decided to go into the details of the structure of our application using class diagrams that represent its components and the relationship among them. In particular, I used the Boundary-Entity-Control (BCE) pattern, which is a variation of the MVC.

Boundaries define the view, so the interaction between the users and the application; they can be associated to the screens defined in UX diagrams, but they may gather more than one screen.

Controllers manage all the business logic of the system and act as a connection between the boundaries and the entities.

Finally, the entities represent the data model of the application. I point out that the entities are slightly different from the ones in the Data Base, because I have decided to give a representation suitable for a class diagram (some terms are changed, for example the IDQueue is replaced by the property queue)

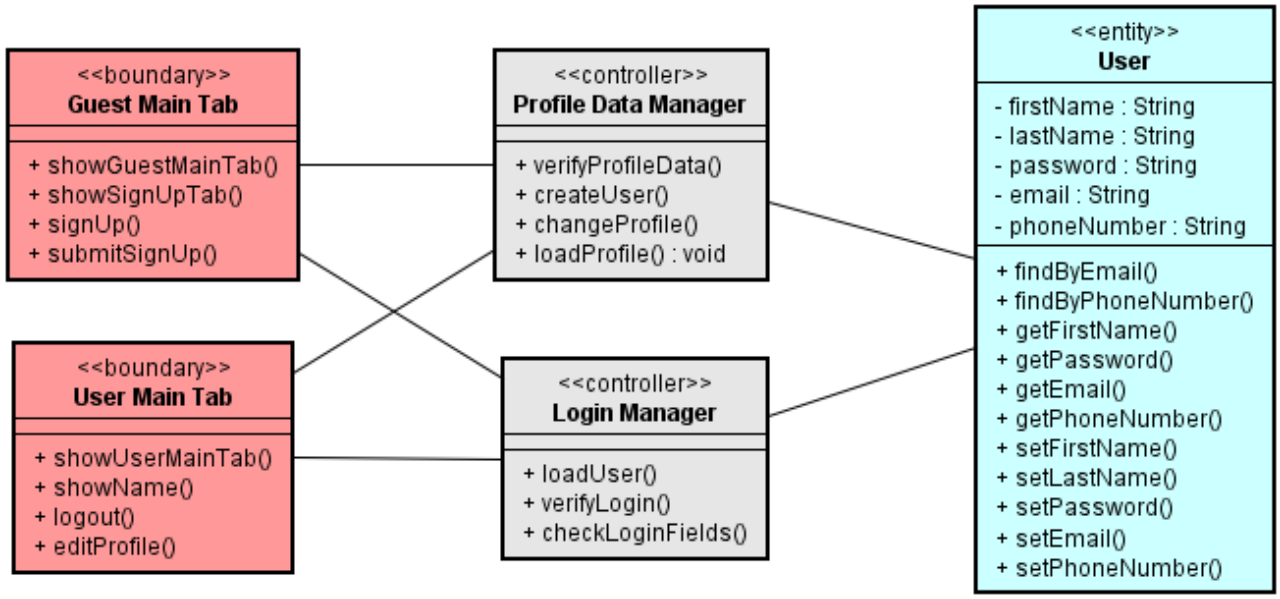
4.1 Sign Up, Log In and Change Profile Data

The two boundaries Guest Main Tab and User Main Tab represent the interfaces between the user and the system; the first one when the user is a guest, the second one when the user is authenticated. They expose to the user the main functionalities that can be activated when the user is the main tab of the application except the request one, which will be discussed in the section 4.2.

In this subsystem there are two main controllers:

- Profile Data Manager: it controls the correctness of the information inserted by the user during the sign up task, it creates new user entities and it manages the modification of profile's information. Also the displayed name on the User Main Tab is retrieved by this manager using the method loadProfile().

Log In Manager: it controls that the username and password inserted by the user in the log in fields are conform with their correct structure (by means of the method checkLogInFields) in order to avoid the access to the DB in the case that information inserted can't be valid. It verifies that the username and password inserted by the user correspond to an existing user, using the method verifyLogIn. It loads the information about the user by means of the method loadU



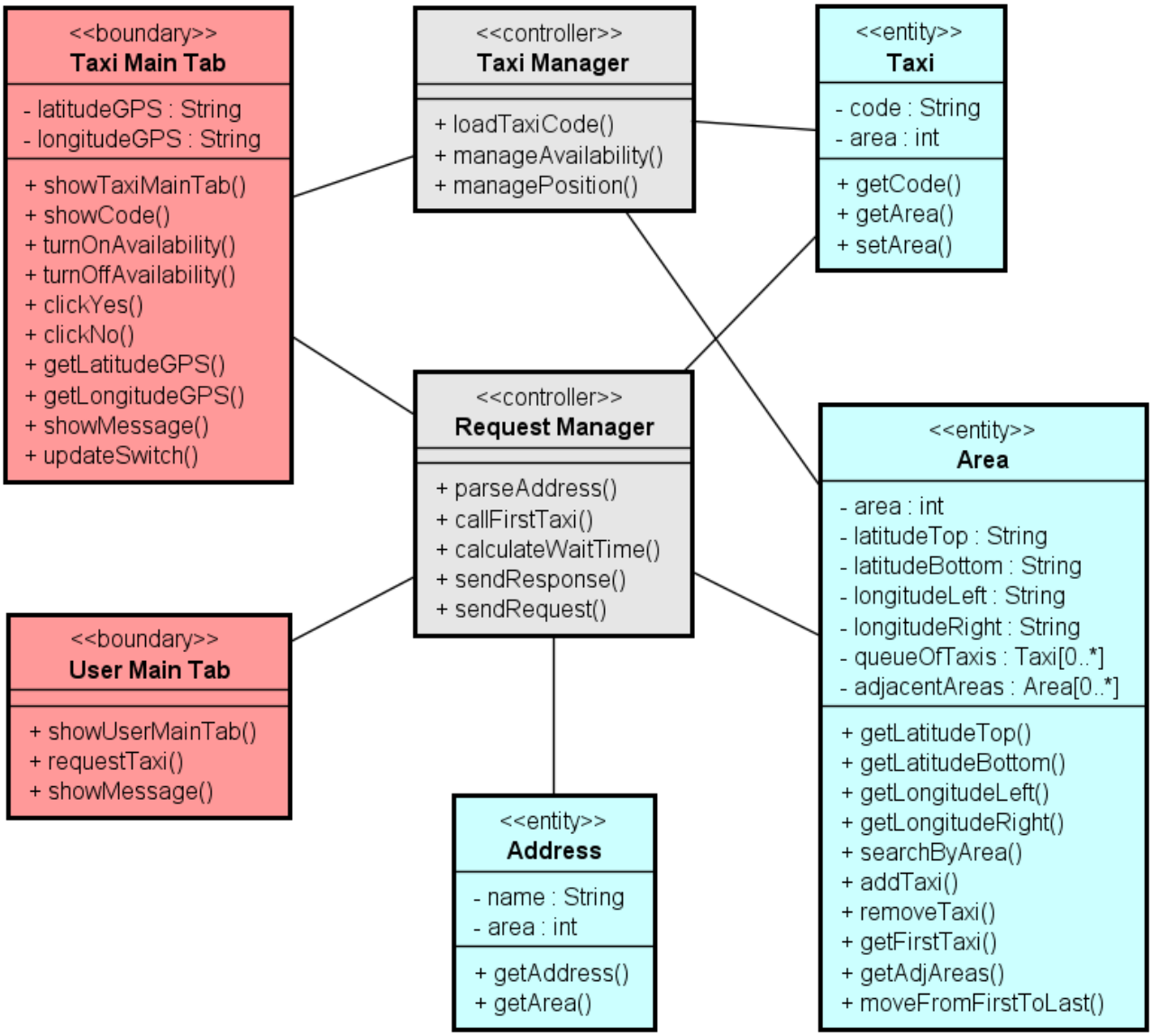
4.2 Taxi Managing and Request Managing

The boundary *User Main Tab* represent the interfaces between the system and the user (only the method requestTaxi() because the others were discussed in the section before)

The *Taxi Main Tab* represent the interfaces between the system and the taxi driver. This boundary keeps track of the GPS coordinates (automatically generated by the GPS module of the mobile device) and provides all the functions that the taxi driver can do.

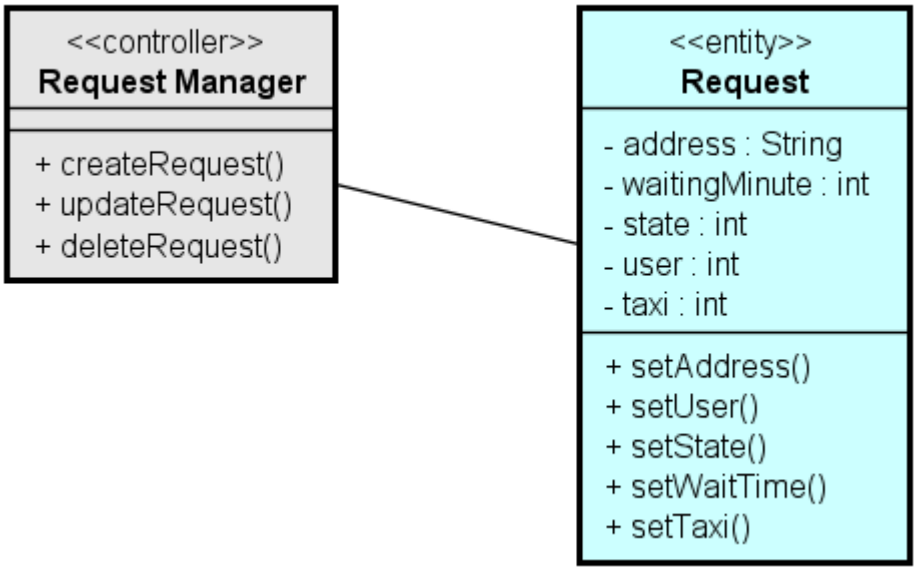
In this diagram I have two controllers:

- Taxi Manager: it controls the position of the free taxis and keeps the integrity of the queues. More specifically it periodically compare the position of a taxi (by its coordinates) with the coordinates of its current queue. If the taxi is no more in that area it the system have to reallocate the taxi in a new queue (by comparing the position of the taxi with the coordinates of all the areas[1]) by adding it at the last position. There is also the possibility that a taxi goes out of the city. For all these possibilities, the Taxi Manager have to update the values of the entity Taxi (bounded with the database). It have also the task of managing the manual availability settled by the taxi driver and retrieving the taxi code.
- The Request Manager: it takes care of all the life of a request, starting with the receiving of a request by an user, parsing the address, calling the taxi, parsing the response from the taxi, updating the queues (and eventually repeating the last three actions), calculating the waiting time and, at the same time, giving feedbacks to the user by messages (or bad responses).


<<boundary>>
Taxi Main Tab
- latitudeGPS : String
- longitudeGPS : String
+ showTaxiMainTab()
+ showCode()
+ turnOnAvailability()
+ turnOffAvailability()
+ clickYes()
+ clickNo()
+ getLatitudeGPS()
+ getLongitudeGPS()
+ showMessage()
+ updateSwitch()
<<controller>>
Taxi Manager
+ loadTaxiCode()
+ manageAvailability()
+ managePosition()
<<entity>>
Taxi
- code : String
- area : int
+ getCode()
+ getArea()
+ setArea()
<<controller>>
Request Manager
+ parseAddress()
+ callFirstTaxi()
+ calculateWaitTime()
+ sendResponse()
+ sendRequest()
<<entity>>
Area
- area : int
- latitudeTop : String
- latitudeBottom : String
- longitudeLeft : String
- longitudeRight : String
- queueOfTaxis : Taxi[0..*]
- adjacentAreas : Area[0..*]
+ getLatitudeTop()
+ getLatitudeBottom()
+ getLongitudeLeft()
+ getLongitudeRight()
+ searchByArea()
+ addTaxi()
+ removeTaxi()
+ getFirstTaxi()
+ getAdjAreas()
+ moveFromFirstToLast()
<<boundary>>
User Main Tab
+ showUserMainTab()
+ requestTaxi()
+ showMessage()
<<entity>>
Address
- name : String
- area : int
+ getAddress()
+ getArea()

4.3 Request Log

This diagram represents the function of logging all the requests in order to fulfil the conceptual design of the database. This function is not mandatory for the tasks showed in the previous BCE diagrams but in perspective of a future upgrade of the application (for example if I want to optimize the allocation of the taxis) I have decided to include it.



5. Sequence diagrams

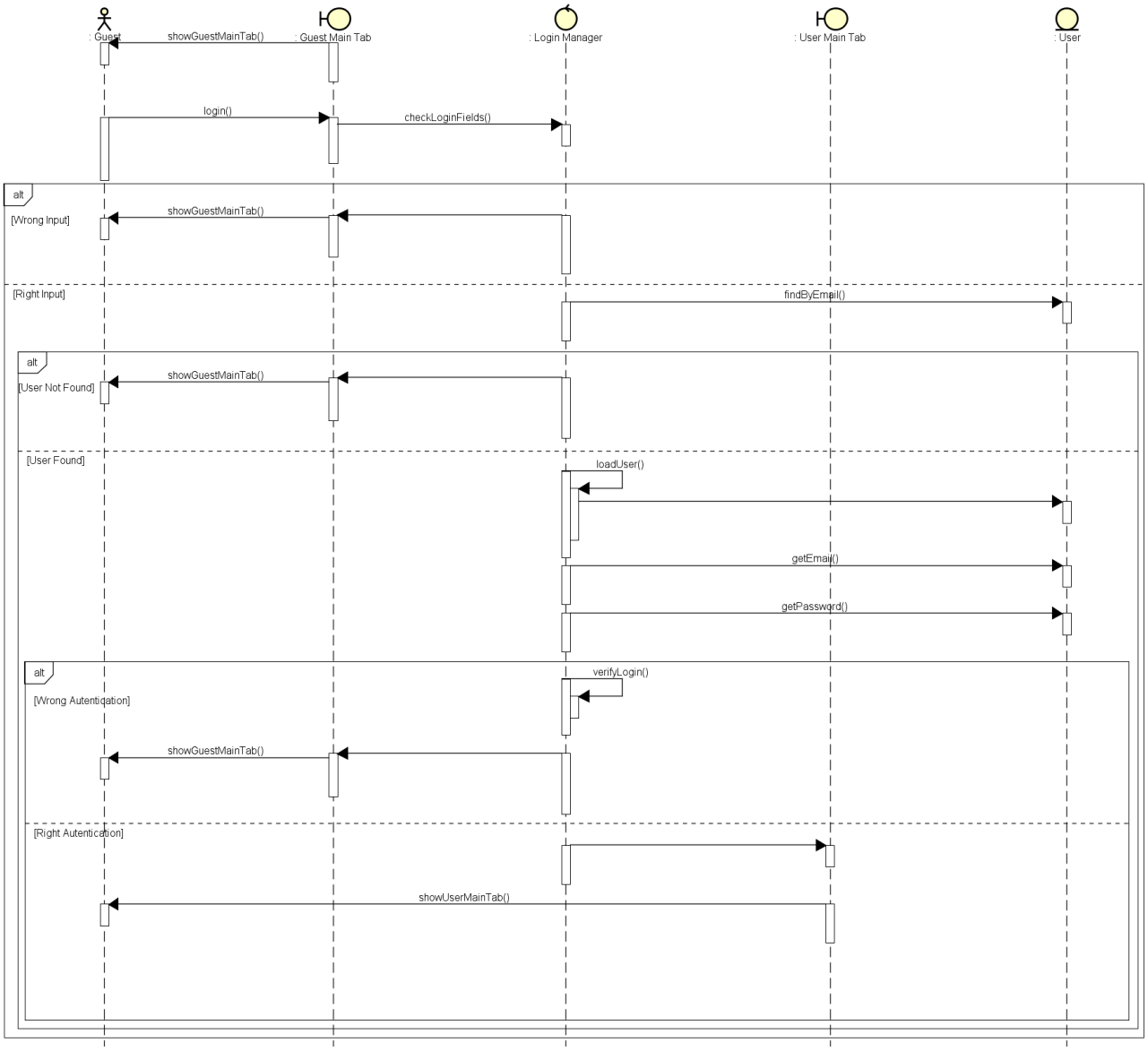
I provide some sequence diagram to let the reader better understand BCE diagrams described above.

All the methods used are the methods listed into the BCE in boundaries, controls and entities (some complex routines are simplified because this is not meant to be the final implementation, for example the update of the queue by changing all the positions after the remove of a taxi).

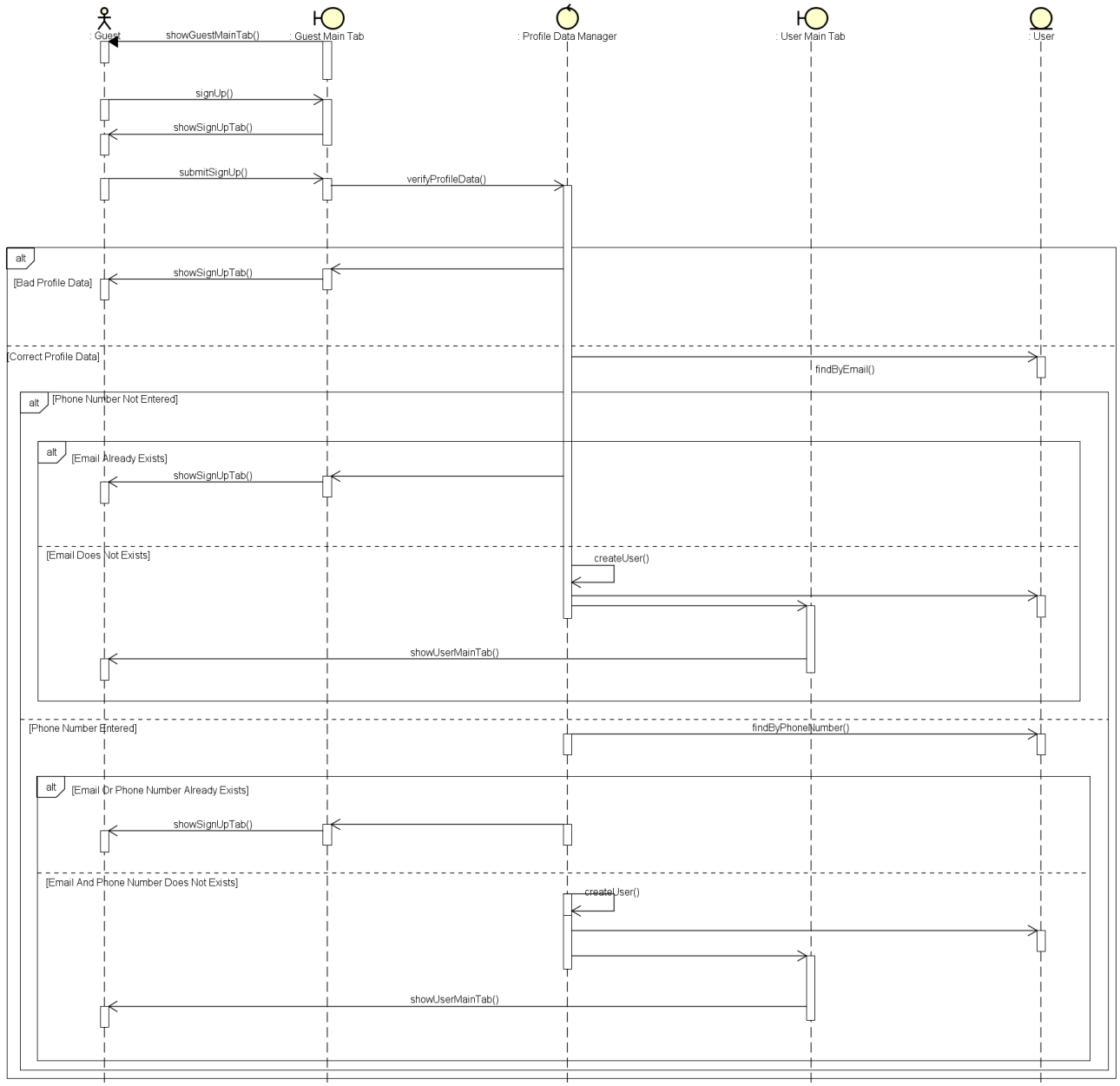
The list of the Sequence Diagram I are going to show is this:

- Log In
- Sign Up
- Initialize Taxi
- Turn Availability On
- Turn Availability Off
- Manage Taxi Position
- Manage Request

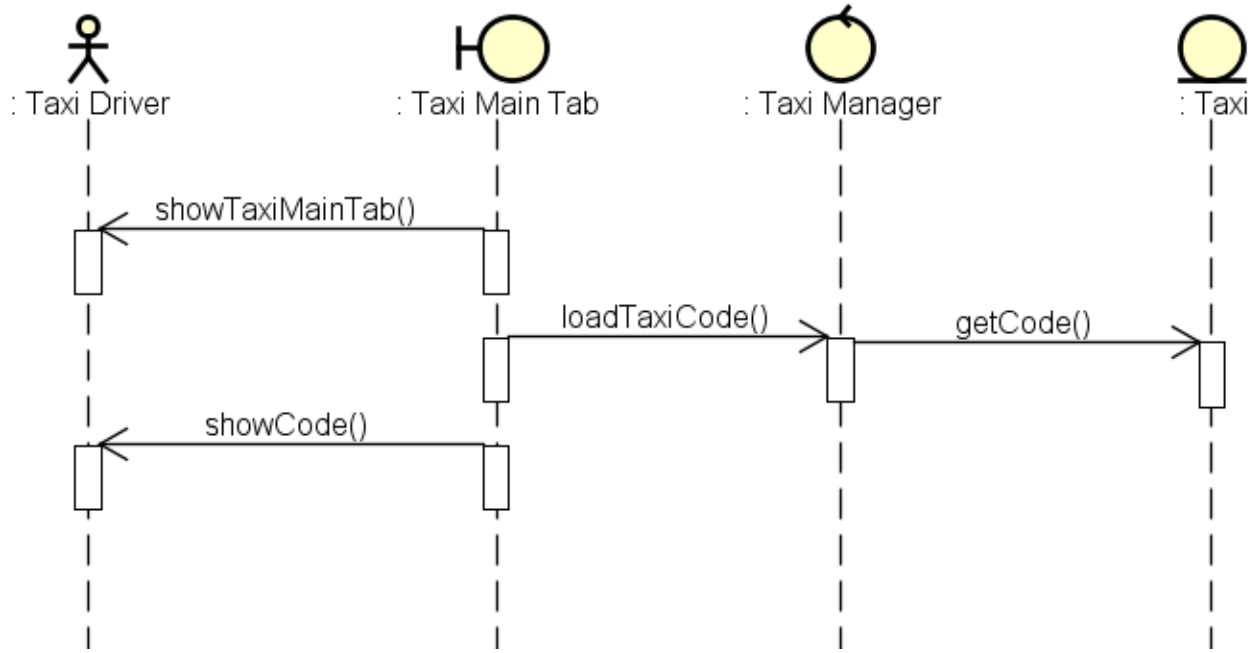
5.1 Log in Sequence Diagram



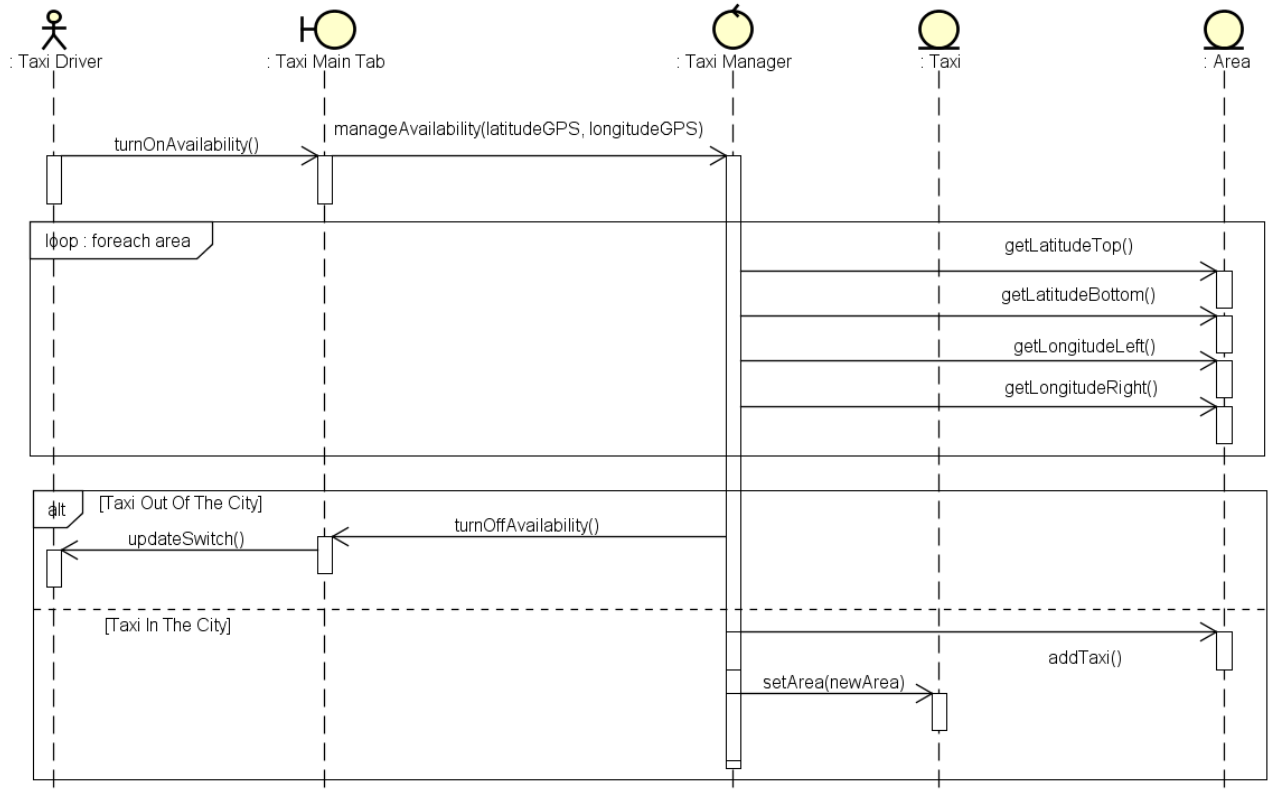
5.2 Sign Up Sequence Diagram



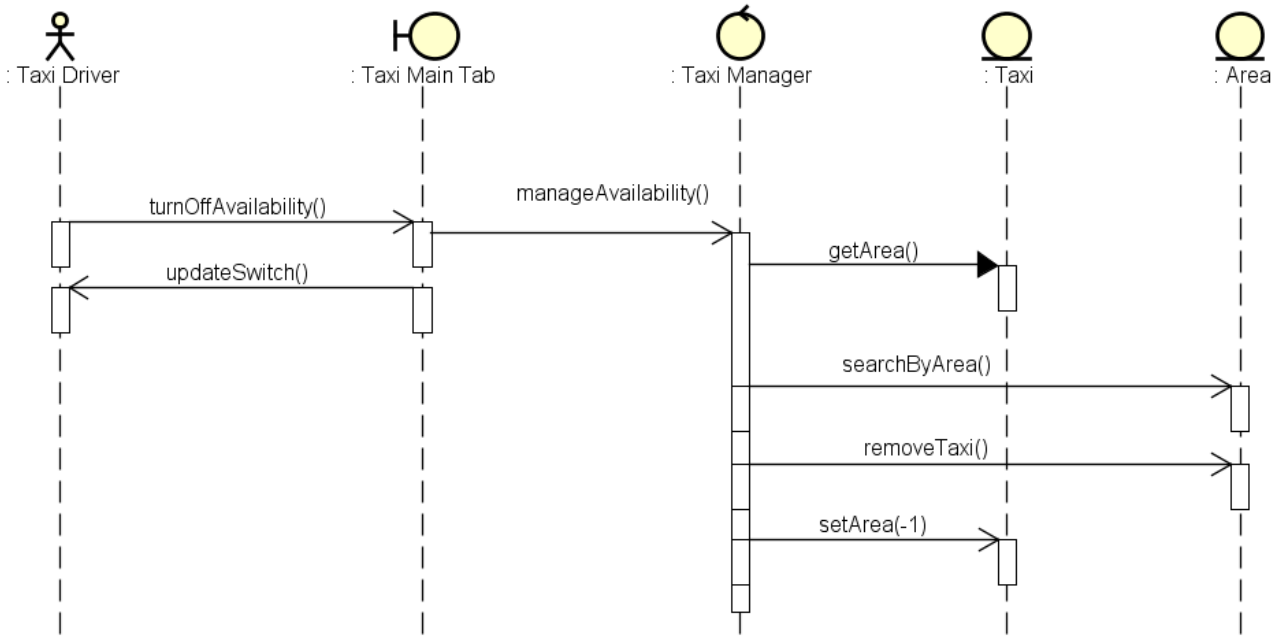
5.3 Initialize Taxi Sequence Diagram



5.4 Turn Availability On Sequence Diagram

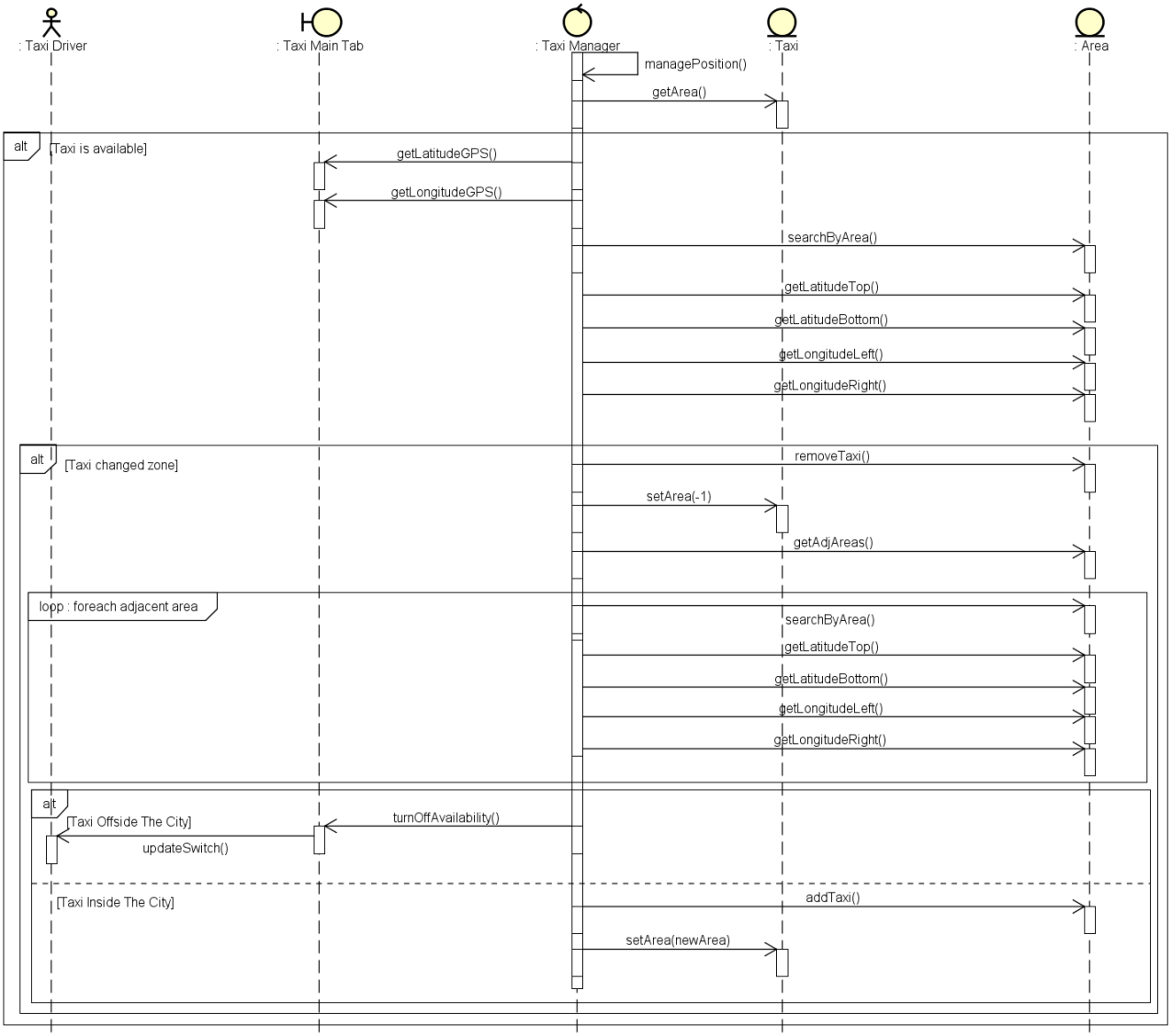


5.5 Turn Availability Off Sequence Diagram

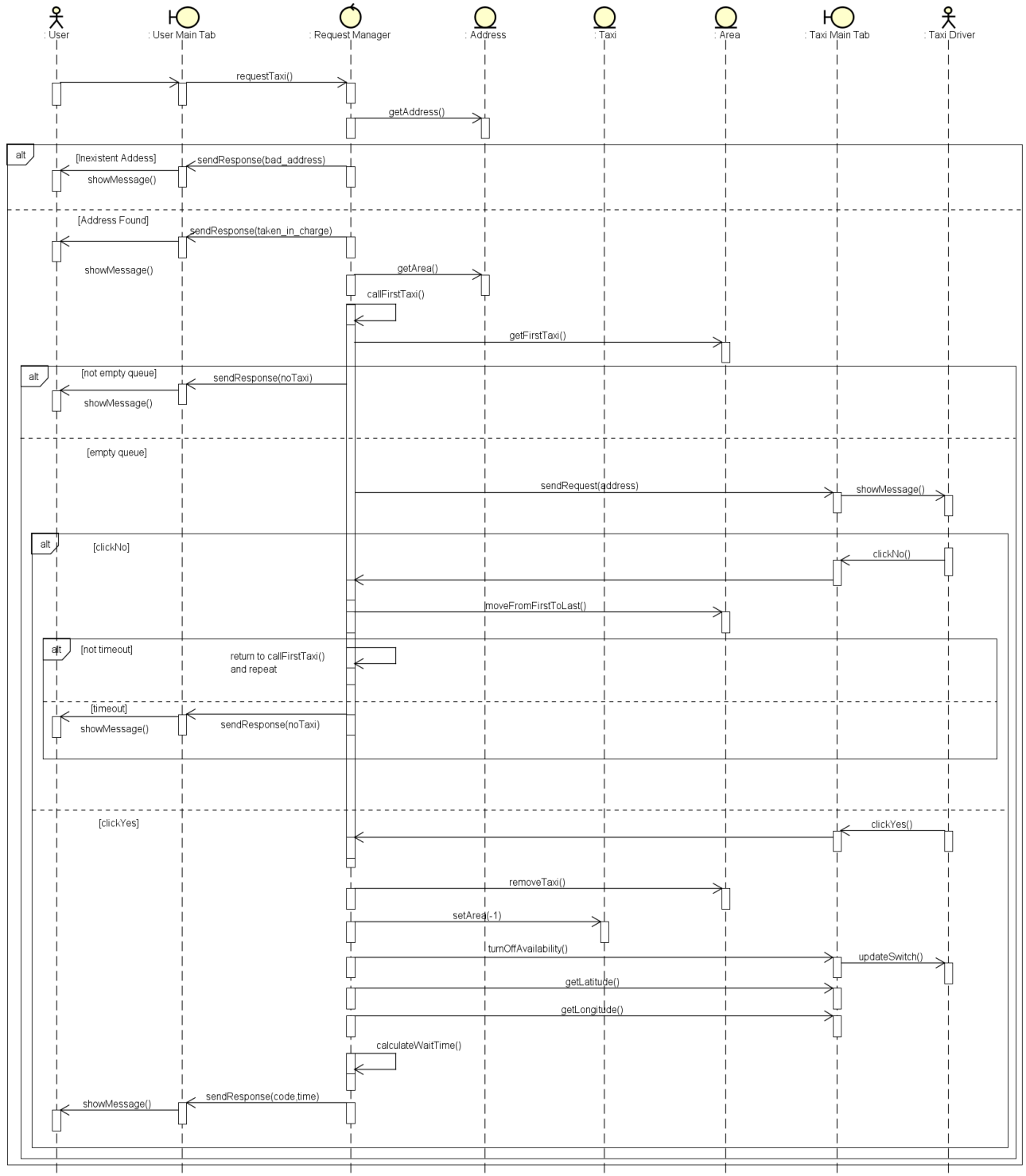


5.6 Manage Taxi Position Sequence Diagram

This procedure is scheduled and executed every minute by the taxi manager associated to each taxi.



5.7 Manage Request Sequence Diagram



6. User Interface Design

The design of the application is entirely based on the simple user interface described in the RASD document.

7. Requirement Traceability

All the requirements of the RASD are fulfilled in the previous diagrams.

8. References

The tools I used to create the RASD document are:

- Microsoft Office Word 2013: to redact and to format this document;
- Astah Professional: to create the diagrams and the ER tables model;
- https://erdplus.com: to create the ER conceptual diagram
- Paint.NET: to adjust the images.

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