



Politecnico di Milano

Software engineering 2

myTaxiService

Design Document (DD)

Version 1.0

Release date 4th december 2015

790021 Antenucci Sebastiano

852461 Buonagurio Ilaria

Prof.ssa Di Nitto Elisabetta

Table of contents

| | |
|---|------------------|
| <i>1 Introduction</i> | <i>3</i> |
| <i>1.1 Purpose</i> | <i>3</i> |
| <i>1.2 Scope</i> | <i>3</i> |
| <i>1.3 Definitions, Acronyms, Abbreviations</i> | <i>4</i> |
| <i>1.3.1 Definitions</i> | <i>4</i> |
| <i>1.3.2 Acronyms</i> | <i>5</i> |
| <i>1.4 Reference documents</i> | <i>5</i> |
| <i>1.5 Document structure</i> | <i>6</i> |
| <i>2 Architectural design</i> | <i>7</i> |
| <i>2.1 Overview</i> | <i>7</i> |
| <i>2.2 High level components and their interaction</i> | <i>8</i> |
| <i>2.3 Component view</i> | <i>9</i> |
| <i>2.4 Deployment view</i> | <i>10</i> |
| <i>2.5 Runtime view</i> | <i>11</i> |
| <i>2.6 Component interfaces</i> | <i>19</i> |
| <i>2.7 Selected architectural styles and patterns</i> | <i>19</i> |
| <i>2.8 Other design decision</i> | <i>20</i> |
| <i>3 Algorithm design</i> | <i>21</i> |

| | |
|---|-----------|
| 4 User interface design | 24 |
| 4.1 Client interface design | 25 |
| 4.2 Taxi driver interface design | 27 |
| 5 Requirements traceability | 28 |
| 5.1 Functional requirements | 28 |
| 5.2 Non functional requirements | 29 |
| 6 References | 30 |
| 7 Appendix | 31 |
| 7.1 Changes to RASD | 31 |
| 7.2 Tools | 31 |
| 7.3 Working time | 32 |

1 Introduction

1.1 Purpose

This paper describes the software architecture of the application myTaxiService. The goal of this document is provide the specifications about design information and concern in a sufficient detail to guide the implementation phases.

Is important to keep this document up to date because it plays a pivotal role in the development and maintenance of the software system.

This document is intended mainly for developers who will have to implement the system, which will be required to follow the design described.

It considers both the software and its system context, including the developmental and operational environment.

The description of the system adopts a top-down approach. This choice has been taken in order to realize a system as close as possible to the high-level requirements formalized in the analysis document.

Any change of this document will be noted and described in the Appendix.

1.2 Scope

Foremost the aim of the project is to optimize the taxi service of a large city and in particular it is to simplify passengers' access to the service and to guarantee a fair queue management by creating a new software called myTaxiService.

Users can request a reservation through a web or mobile application and they are notified by the system with the code of the incoming taxi and the waiting time.

There are a two kind of reservation: the reservation, made at least two hours in advance of the meeting time by specifying the departure and arrival of the ride; or the "quick" one, by specifying only the origin of the ride. In the second case is supposed that the taxi is requested immediately.

In support to taxi drivers there is a mobile application that allows them to inform the system about their availability and they can also confirm or dismiss a certain call.

The system also simplifies queues management and taxi distribution on the territory, in order to make the service more efficient and to serve a larger number of passengers.

1.3 Definitions, Acronyms, Abbreviations

In this section we provide definitions, acronyms and abbreviations that can be useful in order to better understand this document.

1.3.1 Definitions

Here we define some terms used in the design document in order to give the reader a better understanding of it.

- Layer: logic level that contains a set of functionalities of the application.
- Tier: a layer in a multitiered software architecture.
- Multitiered: client-server architecture in which presentation, application processing, and data management functions are physically separated.
- Java: general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.
- Java Servlet: Java program that extends the capabilities of a server.
- Top-down: software design strategy which first provides an overview of the system (top) and then goes into the detail of the components that compose the system (down).
- Server: computer program or machine capable that waits for requests from other machines or software (clients) and responds to them. The purpose of a server

is to share data or hardware and software resources among clients.

- Client: piece of computer hardware or software that accesses a service made available by a server.

1.3.1 Acronyms

Here we define the acronyms that will be used in the document.

- JEE: Java Enterprise Edition.
- DD: Design Document.
- RASD: Requirements Analysis and Specification Document
- EIS: Enterprise Information System.

1.4 Reference documents

In order to write this document we consulted the following sources:

- IEEE Std 1016-2009 IEEE Standard for Information Technology - System Design - Software Design Description
<http://ieeexplore.ieee.org/servlet/opac?punumber=5167253>
- SO/IEC/IEEE Systems and Software Engineering - Architecture Description
<http://ieeexplore.ieee.org/servlet/opac?punumber=6129465>
- Specification Document: Assignments 1 and 2 (RASD and DD).pdf
- RASD - S. Antenucci, I.Buonagurio
- Structure of the design document.pdf
- Slides from Software Engineering 2 course of Politecnico di Milano, academic year 2015/2016.

1.5 Document structure

The document is structured as follows: first of all we will give a general overview about the document. Then we will focus on the system and in particular first in the architectural design, showing the implementation of our system, both hardware and software, after that on the algorithms that will be used in the development then on the user interface design and in the end on the requirement traceability.

Moreover the document is divided into the following sections:

- Introduction: section that defines the general aspects related to the design of the system in addition to defining purposes and recipients for this document.
- Architectural design: section that describes in detail the structure, navigation and logic decomposition of the application.
- Algorithm design: section that provides the algorithms used in the implementation of the software.
- User interface design: section that shows how the user interface is designed.
- Requirement traceability: section that assesses the traceability requirements for the system.

2 Architectural design

2.1 Overview

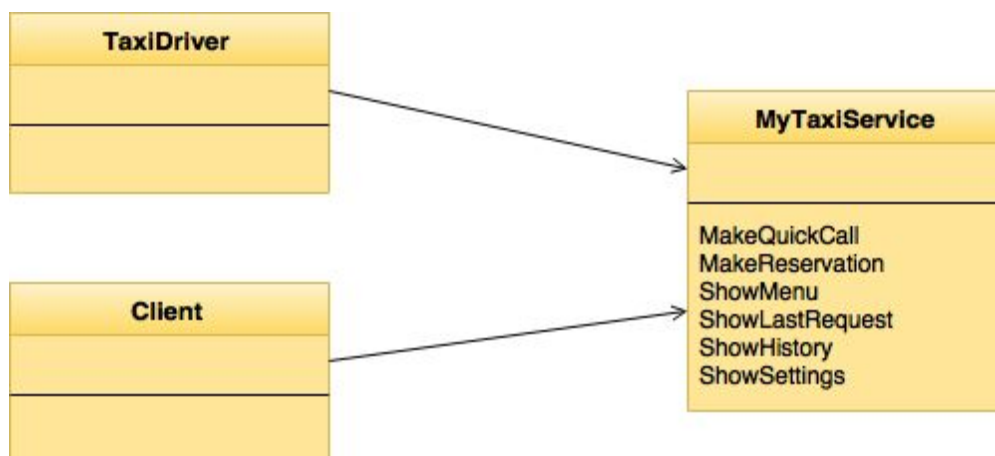
We decided to use JEE architecture, this architecture consist in a 4-layer structure that allow us to handle at the same time web and mobile application;

- Client tire: contains web application and mobile application for user and taxiDriver;
- Web tire: contains the web pages servlet which will be used from the web application; it has the task to handle client request to the server and to generate response
- Business tire: will be used either for the app and web application; this layer manage the user session, keeping him logged in into the server while using the application
- EIS tier: represent the database part which contains the user and taxi driver data, it contains also all the information that allow the system to manage the queue, the taxi request and distribution on the territory

2.2 High level components and their interactions

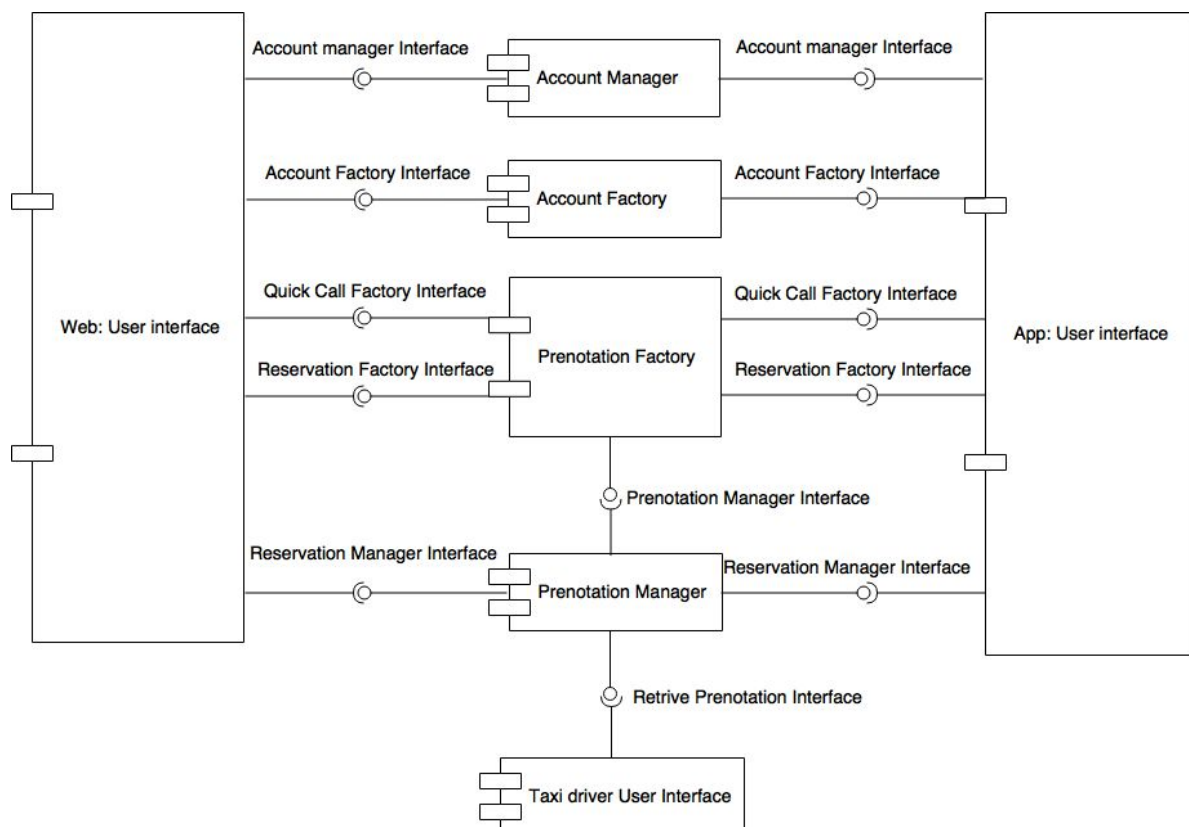
In the next diagram we show our system at a high level, so that we don't focus on how parts manage interaction but on how they interact. It shows the interaction between TaxiDriver, Client and MyTaxiService. We can see that the TaxiDriver and the Client can access to MyTaxiService and can use its functionalities. At this level we are not interested in how they manage it, so the factory and the distinction between users are transparent.

In particular *MakeQuickCall* and *MakeReservation* are used by the Client in order to make a quick call or a reservation; *ShowMenu* is used in order to access to the menu, where a client can manage his profile by accessing to settings, can show the rides' history and can manage the notifications. *ShowLastRequest*, *ShowRides* and *ShowSettings* are used in order to accomplish these dues.



2.3 Component view

In the following diagram we focus on how components interact. We can see that there are two user's interfaces, one for the web application and one for the mobile application. They interact with the system in the same way. Using the account factory interface, implemented by the account factory, the user is able to create an account. With the account manager interface, implemented by the account manager, the user is able to manage his account. In order to make a quick call or a prenotation the user uses the interfaces implemented in the prenotation factory, that deals both with the quick call and with the reservation. The prenotation factory then communicates with the prenotation manager by the prenotation manager interface. The prenotation manager offers methods to both the user and the taxi driver since it sends to both of the notifications and ways to manage the reservations.

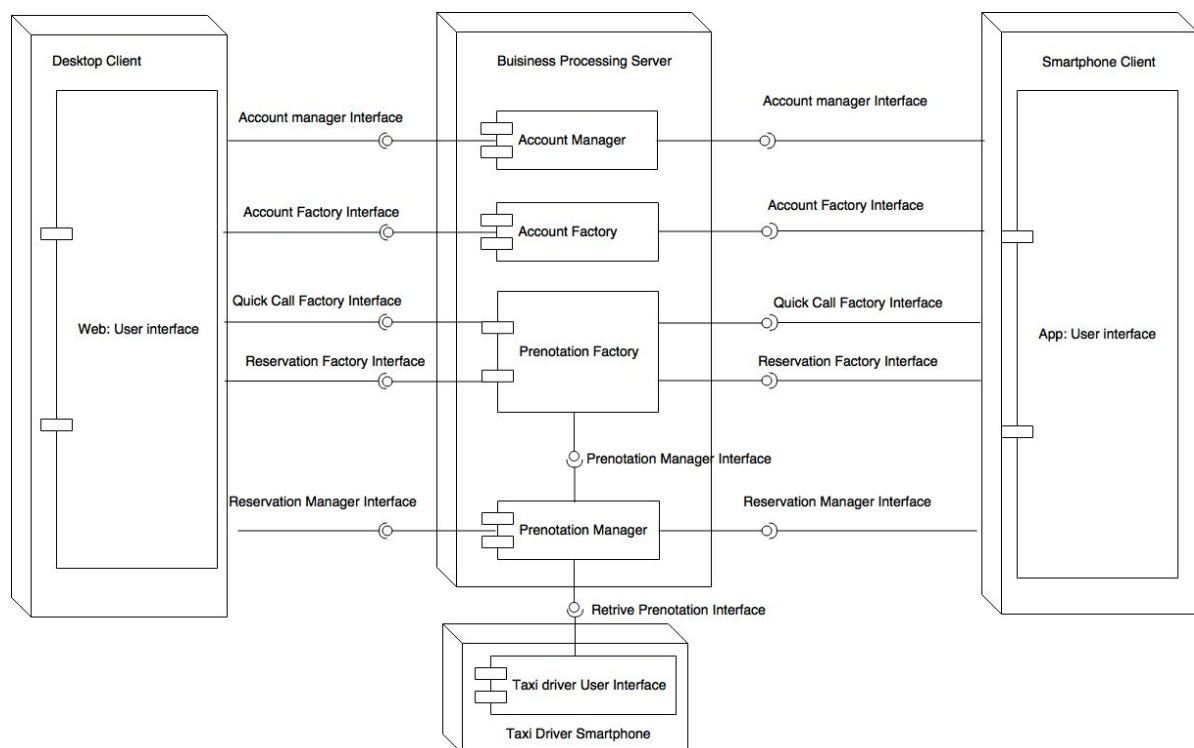


2.4 Deployment view

In this diagram we can see how the different logical component are implemented in different system.

We have Desktop and Mobile Client where the user interfaces are executed, another important part of our system is the Business Processing Server where there are all the server component and the last part of our system is the taxi driver smartphone where is executed the Driver's User Interface.

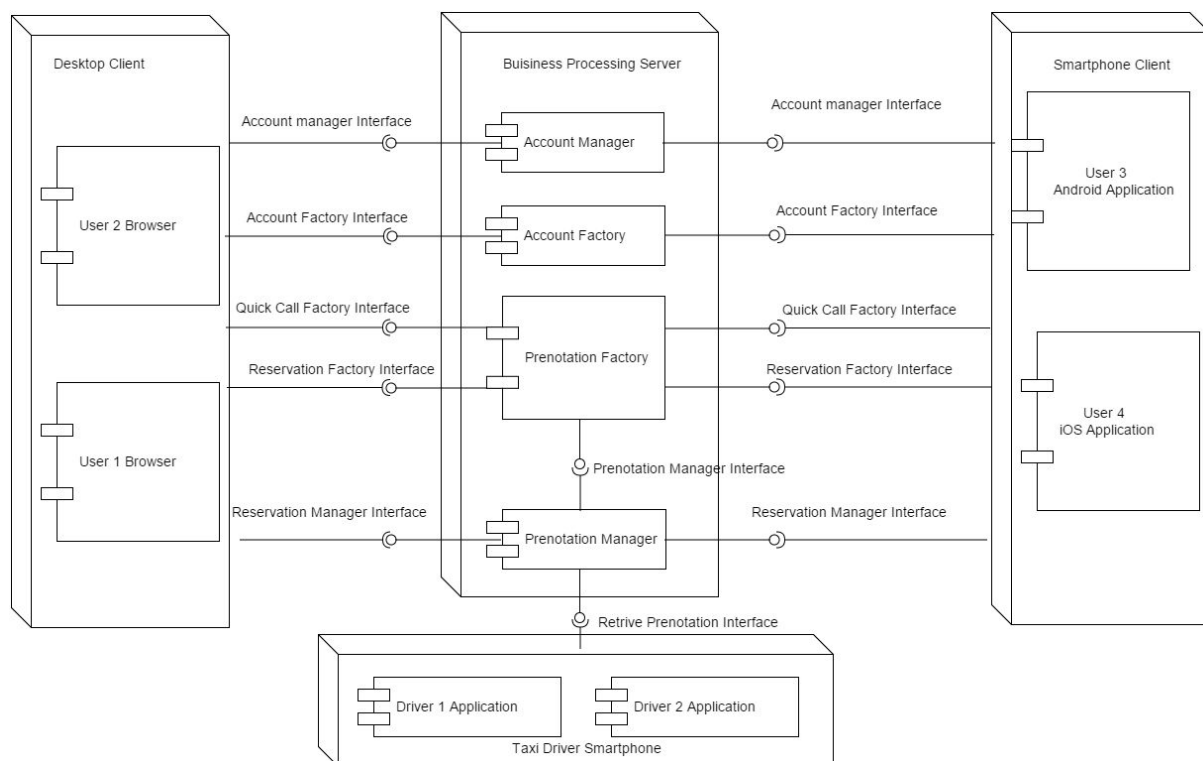
Every UI can communicate with Business Processing Server through different interfaces.



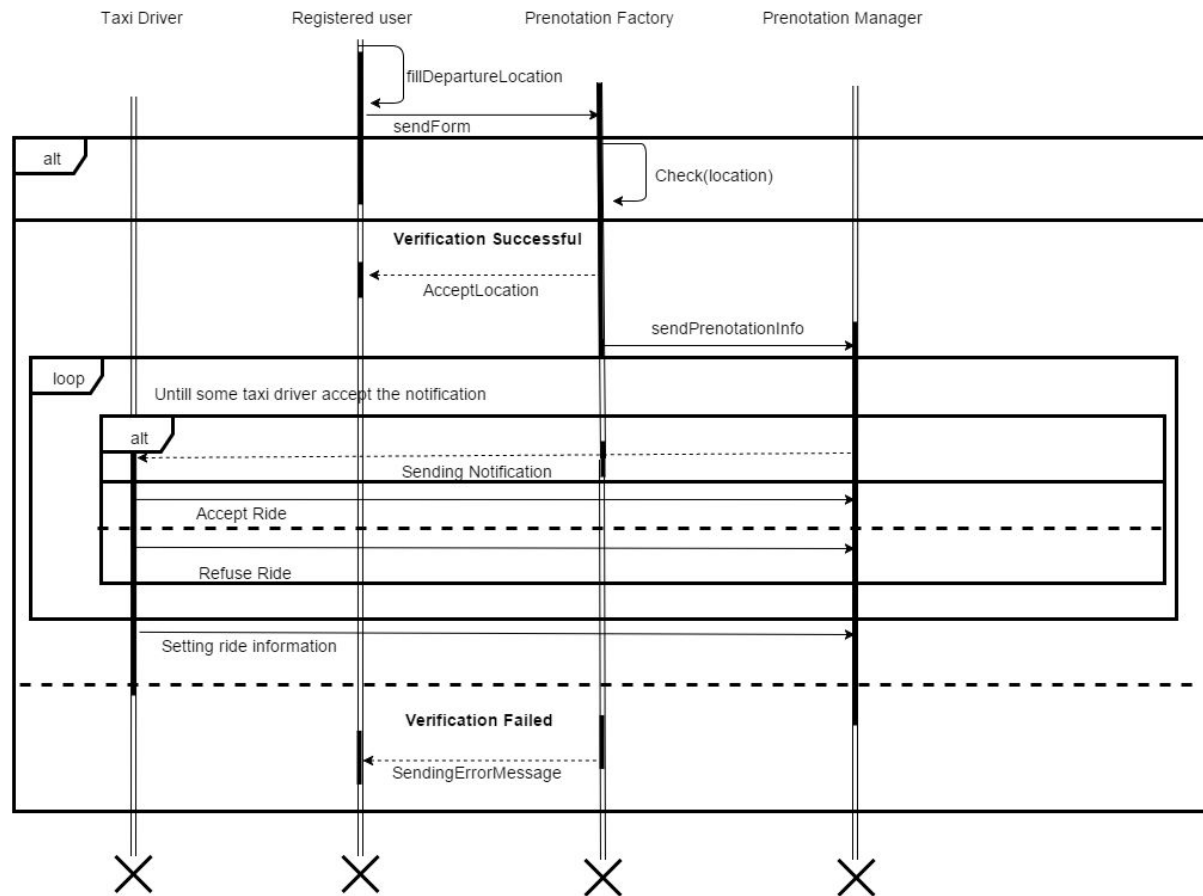
2.5 Runtime view

In the following diagram we took a snapshot during the runtime of our system. In particular we can see an example of execution where there are two users using the web application, user 1 and user 2, two users using the mobile application, user 3 and user 4, and two taxi driver, driver 1 and driver 2.

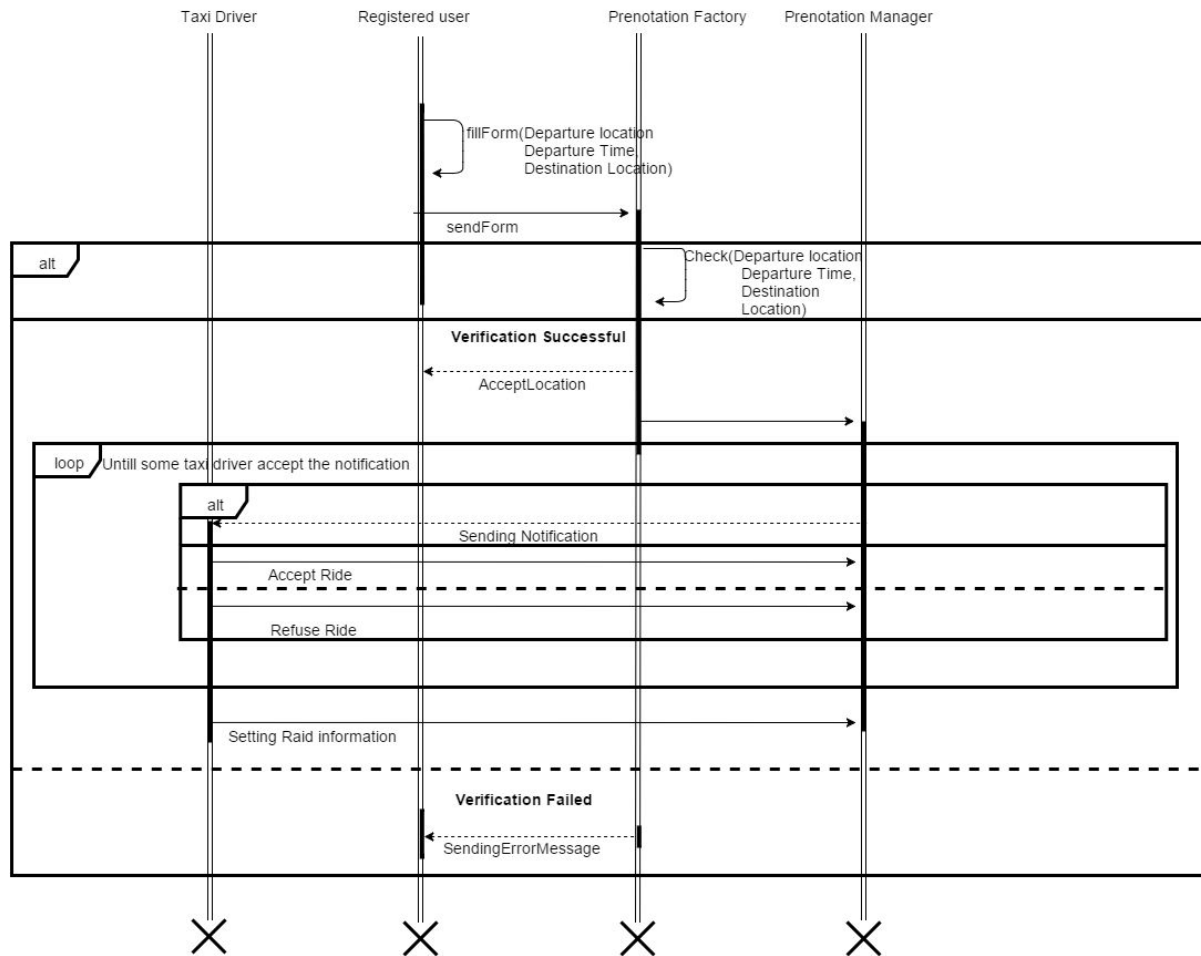
The users can make quick calls and reservations and then the prenotation manager sends notifications to the drivers that are available and in the correct queue.



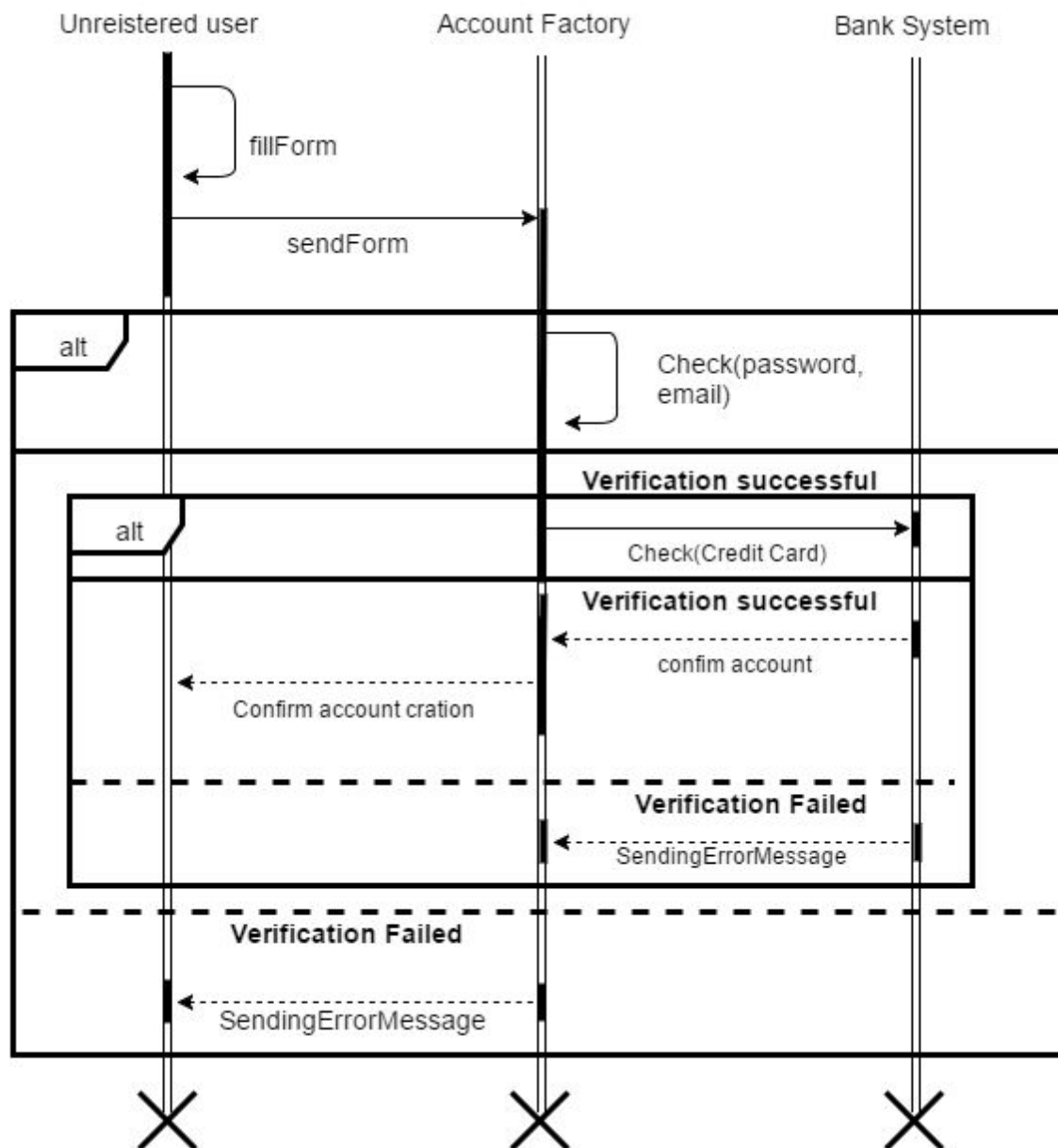
| | |
|------------------|---|
| Name | Quick call |
| Actors | Authenticated user |
| Entry conditions | The User has to be logged in and has to be in his home page. The user has to plan to do a quick call for a taxi ride. |
| Flow of Events | <ul style="list-style-type: none">• The user insert the departure location;• the request is send to all the available taxi driver until someone accept the request;• The user receive a notification about the successfully of the call;• After the ride the taxi driver send to the prenotation manager the ride information;• The ride is paid with the user's credit card; |
| Exit Conditions | The ride is complete and the payment is done. |
| Exceptions | The ride departure location does not exist. An error message is shown. |



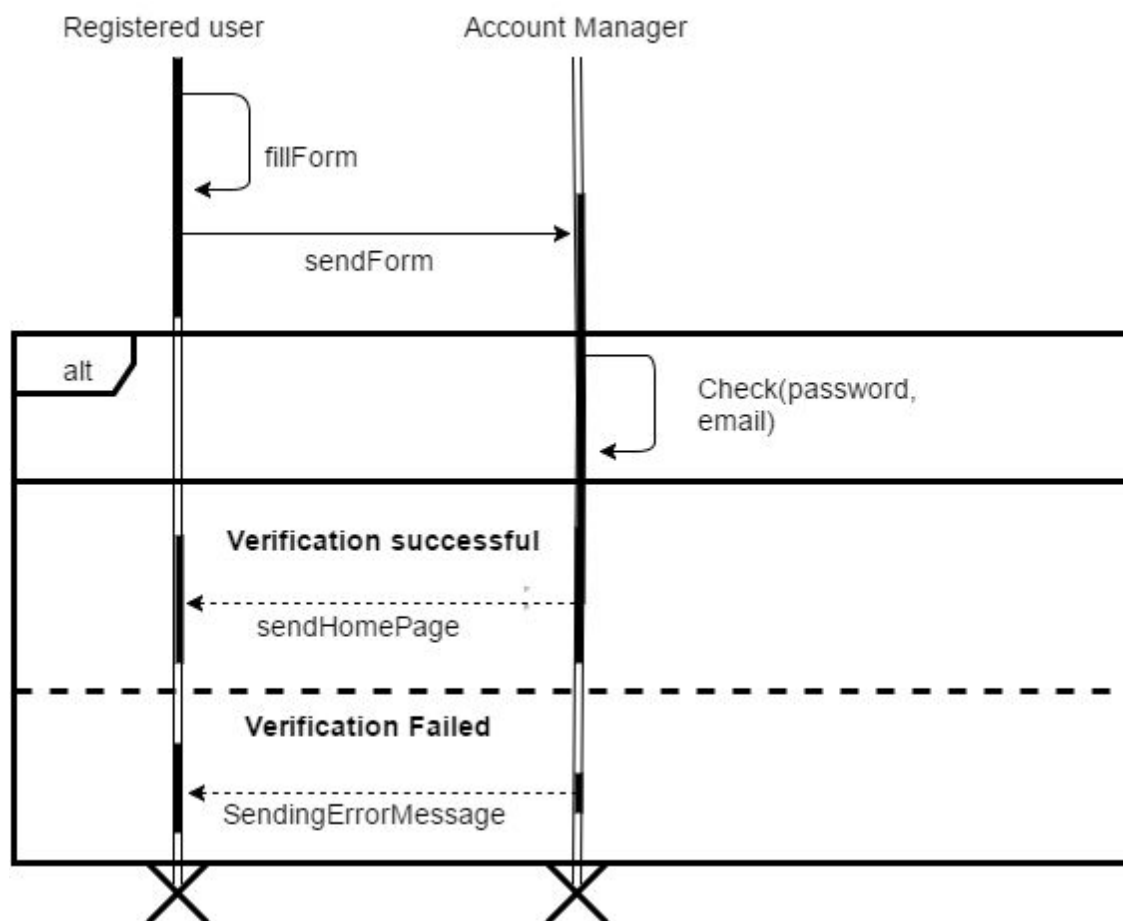
| | |
|------------------|--|
| Name | Reservation |
| Actors | Authenticated user |
| Entry conditions | The User has to be logged in and has to be in his home page. The user has to plan to do a reservation for a taxi ride. |
| Flow of Events | <ul style="list-style-type: none"> • The user insert the time and the departure location; • The user insert the location of the destination; • the request is send to all the available taxi driver until someone accept the request; • The user receive a notification about the success of the call; • 10 minutes before the ride a notification arrive to taxi driver and user; • After the ride the taxi driver send to the prenotation manager the ride information; • The ride is paid with the user's credit card; |
| Exit Conditions | The ride is complete and the payment is done. |
| Exceptions | The ride time or departure location is not valid or the destination location is not valid. An error message is shown. |



| | |
|-----------------|---|
| Name | Sign In |
| Actors | Unregistered User |
| Entry condition | The user want to Sign in into the system |
| Fow of event | <ul style="list-style-type: none"> • The user open the Sign in page; • The system shows him that page; • The user enters his email, password, name, surname and credit card information in the input form provide; • The user click the button create; • the Account Factory control all the data; • The Account Fcatory shows the user successfully sign in page |
| Exit Conditions | The user is registered into the system. |
| Exceptions | The information inserted in the form are wrong, or he doesn't fill some mandatory fields. An error message is shown without showing to user own home page |



| | |
|-----------------|--|
| Name | Log In |
| Actors | Unauthenticated User |
| Entry condition | The user is correctly registered in the system and want to access. |
| Fow of event | <ul style="list-style-type: none"> • The user enters his email and password in the input form provide; • The user click the button log in; • The Account Manager shows the user home page |
| Exit Conditions | The user is authenticated in the system. |
| Exceptions | The information inserted in the form are wrong, or he doesn't fill some mandatory fields. An error message is shown without showing to user own home page |



2.6 Component interfaces

- Account Factory Interface: allow the user to create a new account from the different user interfaces
- Account Manager Interface: allow the user to login into the system and to manage account information
- Quick Call Factory Interface: allow the user to make a quick call specifying departure location
- Reservation Factory Interface: allow the user to make a reservation specifying departure time and location and arrival location
- Prenotation Manager Interface: allow the reservation factory components to confirm and insert the prenotation into the system
- Reservation Manager Interface: allow the user to manage reservation info like departure time and location and arrival location
- Retrieve Prenotation Interface: allows the taxi driver to retrieve information about the prenotation done by the user

2.7 Selected architectural styles and patterns

In the creation of the myTaxiService system we used some common famous patterns.

- Factory pattern: we used it in all the creational processess, like the creation of an user, of a ride, of a notification and so on.
- Builder pattern: we used it in order to compose a reservation: it composes destination, departure location, departure time, departure date.
- Adapter pattern: we used it in order to treat in the same way a quick call and a reservation.
- Decorator pattern: we used it in order to create quick calls and reservations.

- Observer pattern: we used it in order to manage the input made by the driver and the users.
- Model View Control pattern: we used it to manage the system. The application installed on the driver and user smartphone and the web app are the view, the model is in the database and the control is in the server.

2.8 Other design decision

There are no particular other design decision at the current version of this document.

3 Algorithm design

In this section we will focus on the definition of the most relevant algorithmic part of the project.

Since we are dealing with lists in many fields like managing the queues, adding a new request or cancelling an existing one and creating or erasing a new client or taxi driver, we decided to focus on the lists. In particular we will explain the algorithm for adding an element to the head of the list, adding an element in order in the list, delete an element from the bottom of the list and erasing a specific element.

For each algorithm we specify the pseudo-code, the description of it and the use of the algorithm in our system.

- Algorithm 1 shows how to deal in order to delete the last element of a list. We used it in order to manage the taxi queues, that follows a FIFO logic. It is invoked every time the systems sends a notification to a taxi driver, whether he accepts it or not. It is also used when a taxi driver changes zone, in order to delete it from the old zone.

Algorithm 1 dequeue (list)

Require: $n \geq 0$

```

1:  $t \leftarrow head$ 
2: while  $t.hasNext()$  do
3:    $t \leftarrow t.next()$ 
4: end while
5:  $t.next() \leftarrow NULL$ 

```

The requirements for this algorithm are that the list must not be empty, so it must contain at least one element.

In line 1 we assign to a temporary node t the head of the list. Then, in line 2 and 3, we perform a cyclic condition: we read all the list while the element that we are considering has a successor.

When we have found the second to last element of the list, we change its successor to $NULL$, as seen in line 5, so that the last element of the list is no more reachable from the head of the list.

- Algorithm 2 deals with the operation of adding an element as a head of a list, so at the beginning of a list. It is used with algorithm 1 in order to manage FIFO queues. It is invoked every time a taxi driver ends his duty and is still available or when a taxi driver switch from unavailable to available. It is also used when a taxi driver changes zone, in order to add it to the new zone.

Algorithm 2 enqueue (list, elem)

```

1: elem.next()  $\leftarrow$  head
2: head  $\leftarrow$  elem

```

Even if they are not specified on the algorithm, the requirements are that the element to be inserted must belong to the list type. In line 1 we attach the head of the list as the element successor. Then we set the head of the list at the element, as seen in line 2, so when we have to access to the list we accede at the element.

- Algorithm 3 explains how to insert elements in a list in a orderly way. We assume that each element is identified with an identifier, in order to make the algorithm more general we also assume that it can be not unique. Another assumption that we make is that the elements are listed in ascending order, from the lowest identifier to the greatest one. We use this algorithm when we want to store the information about rides, reservations, clients, drivers and so on.

Algorithm 3 insert (list, elem)

```

1: while t.next.id()  $\leq$  elem.id() do
2:   t  $\leftarrow$  t.next()
3: end while
4: elem.next()  $\leftarrow$  t.next()
5: t.next()  $\leftarrow$  elem

```

The requirements for this algorithm are that the element must belong to the list type and that the list is in ascending order. The first thing to do with this algorithm is to check if the next element on the list has a greater id that the element that has to be inserted, as shown in lines 1 and 2. When we have found the element after which insert the new one we attach to the element that has to be insert successor's

the successor of the found element, as written in line 4. Then, in line 5, we change the successor of the found element to the new element that has to be insert, in order to attach it to the list.

- Algorithm 4 is about deleting a certain element from an ascending order list. All the assumptions made in the algorithm 3 are still valid. We use this algorithm when we want to delete an element from a list, for example when a ride is completed that ride is removed from the pending rides, or when a payment is executed the client id is removed from the debtors list.

Algorithm 4 delete (list, elem)

```
1: while  $t.next().id() \neq elem.id()$  do
2:    $t \leftarrow t.next()$ 
3: end while
4:  $t.next() \leftarrow t.next().next()$ 
```

The requirements for this algorithm are that the element that has to be deleted is already present in the list, and so that the list contains at least that element.

In order to find the element that has to be deleted, for each element we check whether the identifier of the next element is equal to the identifier of the element that has to be deleted, as shown in line 1 and 2. When we have found the predecessor of the element that has to be deleted we simply change its successor to its successor's successor.





4 User interface design

In this section we want to show the navigation paths provided by the graphical interface for the end user, whether it is a client or a taxi driver. For this purpose there are appropriate navigation models (User eXperience) organized into groups of user, in order to better understand the whole diagram. The screens refer to the mockups provided with the RASD diagram. The User eXperience model or UX-model is characterized by a serie of screens (defined by the stereotype "screen") connected together by methods or functionality. In each screen there is always the static method "navigateTo" that allows the navigation between pages. The screen can also contain form that allows the user to enter data (defined by the stereotype "input form"). Another type of screen is represented by the stereotype "screen compartment" that represents parts of the page that can be shared with others and are easily accessible by a bar or a reserved part on the screen. The last type of screen that we use in this diagram is the class non stereotyped, that are class that only contains attributes and are used in order to make more clear the schema and also to share attributes among screens. The cardinality that bounds the class non stereotyped to the screens may be not specified when dealing with input form because with them the user can only provide one value for each attribute in the non stereotyped class. Moreover the notation "\$", present in some screens, denotes the fact that the screen annotated in this way it is always accessible by the other screen of the diagram; the notation "+" identifies a scrollable screen.

In the following diagrams we have assumed that it is always possible to go back to the previous page. The system that we are going to analyze is the same who it was described in the previous paragraph, but now we will enter into the detail with the UX-model.

Some names may not exactly match the names used in the implementation phase. This choice has been used in order to allow a better understanding of the diagrams and not strongly constrain the choice of names in the implementation. However it will be easy to recognize the name used in the diagrams with those used in the implementation phase.

The graphic convention used to represent the diagram is the following:

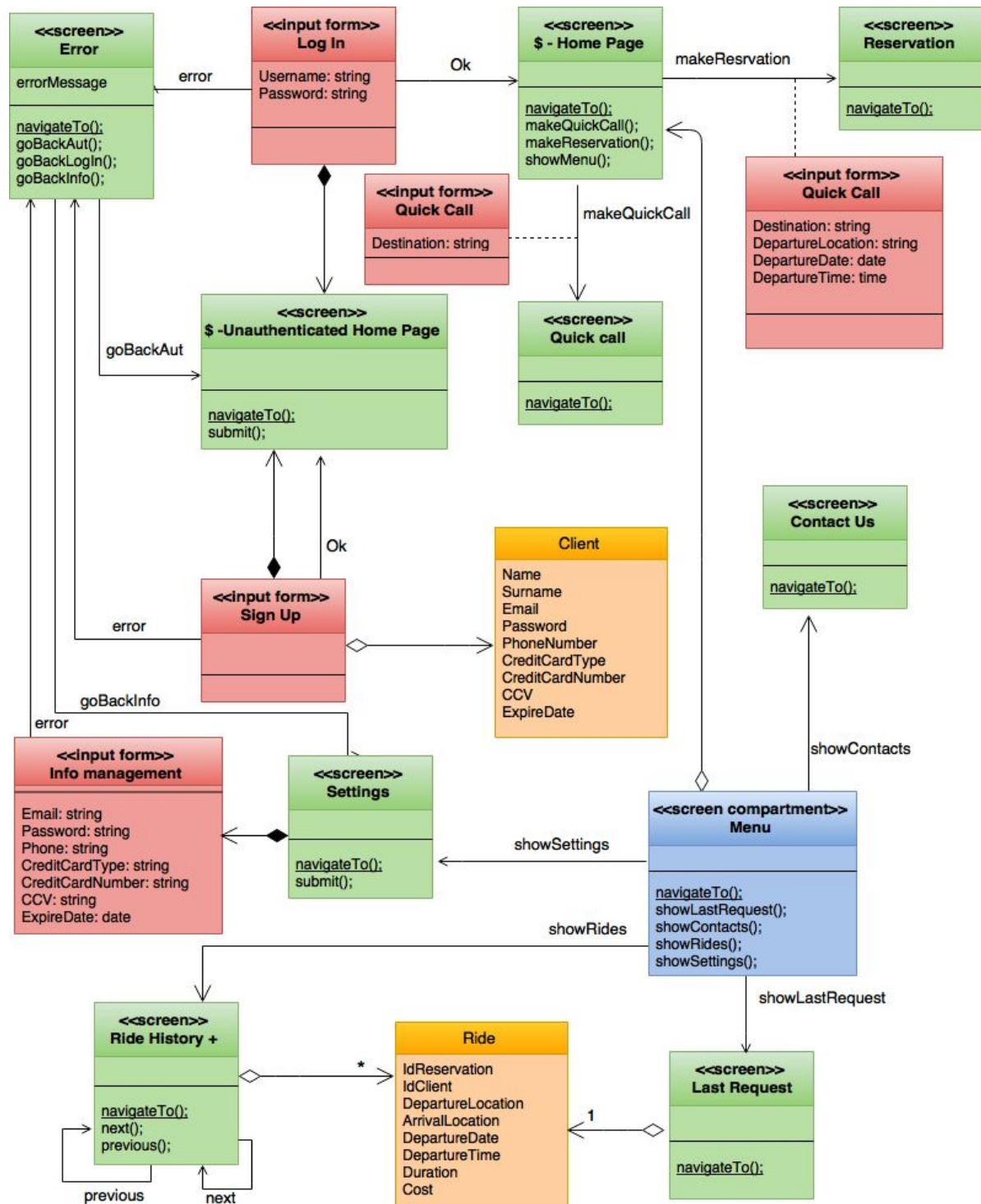
-  = input form
-  = screen
-  = compartment screen
-  = class not stereotyped

4.1 Client interface design

In the diagram below we provide information about the screen visible to an unauthenticated and authenticated user. It is important to point out that this diagram is valid both for the mobile application and for the web one, even if in the implementation the two applications may be implemented in two different ways. Another important thing is that the home page screen and the menu screen are only visible after having completed a successfully log in.

Another assumption of this diagram is that the ride history screen also provides the notifications about each ride.

In the end, the last assumption we made is that the destination location, the departure location, the arrival date and time are all correct, in according to the assumption also made in the RASD.

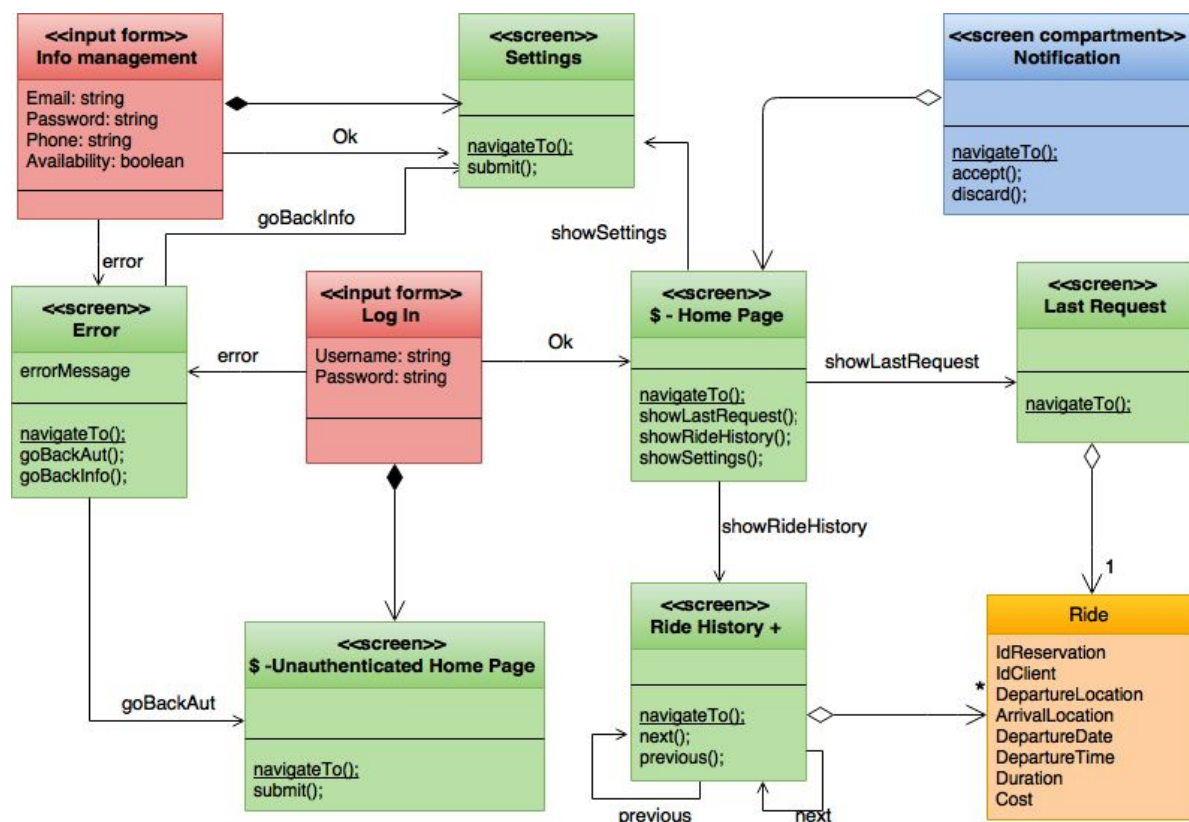


4.2 Taxi driver interface design

The following diagram shows the possible screens visible to a taxi driver in his mobile reserved application.

When the application is opened the company logo will appear and then will ask the driver to log in, if he hadn't already logged in and had chosen the remember me option. If he succeeds in logging in he is redirected to the home page, where he can access all the functionalities provided with the application and already described in the RASD document.

In this diagram is important to specify that the authenticated home page screen and the notification screen are only reachable after having successfully logged in. Moreover, we must underline that some of the field of the ride class may have default values, and this part is managed in the implementation phase.



5 Requirements traceability

5.1 Functional requirements

The system must provide different functionality depending of the type of end user for both the unauthenticated taxi driver and user the system provides the authentication form. For the unauthenticated user the system also provides the sign up functionality.

Authenticated users can make a quick call by providing the location where they want to be picked up, or make a reservation by providing departure date and time, departure location and arrival location. In order to do so the system shows them an appropriate input form.

The system also offers a delete form that is shown only if the operation can be done in time of consistency. Authenticated users can also accede the screens that shows them the last 10 notifications and allows them to manage the last unfulfilled request, in order to check the taxi arrival time, the taxi identification code and the reservation number. Authenticated users can also have access to their profile in order to check information or modify it, and this is managed by showing them an appropriate screen. Another feature to which authenticated users can be admitted is the rides' history, where they can check the reservation number, the taxi identification code, the price, the date, the departure location and time and the arrival location and time of the rides they have made.

When dealing with authenticated taxi drivers, myTaxiService offers the possibility to check their calls list in order to manage salary and shifts, inform the system about their availability or business and, when the system sends them a call notification, to decide whether to take the call or not.

5.2 Non functional requirements

In order to guarantee the reliability of the system, in our design we have chosen an appropriate server that uses a backup recovery in at maximum 5 minutes. This is a consistent backup and every monday night it is saved in the society server farm. Moreover, the server also has an user backup system that is updated for every screen displayed, in order to resume operations from the last screen showed. We use mirroring and RAID to provide this functionality. This backup is a volatile one, so when an user aborts the operation all the data are deleted. The portability is assessed by the use of large compatible components: they can interface with any browser and any operative system.

Last, the database system is protected by a firewall and sensitive data are encrypted with a symmetric encryption. It also filters the data inserted in order to avoid vicious or unwanted modification in the database (e.g. SQL injection).

Concluding, the system also shows only accessible screens, in the sense that an unauthenticated user can't access to authenticated users' pages, and an authenticated user can't access to unauthenticated users' pages.

6 References

We were helped in the creation of the diagrams by some useful resources :

- <http://www.uml-diagrams.org/component-diagrams.html> shows how component diagrams should be built
- <http://agilemodeling.com> helped us build most of the diagrams.
- <https://beep.metid.polimi.it/web/3343933/forum> the beep's forum, where we found useful answers in order to better structure our diagrams.

For the algorithmic part we used this useful guides

- <https://en.wikibooks.org/wiki/LaTeX/Algorithms>
- <http://tex.stackexchange.com/questions/214315/writing-an-algorithm-in-latex>

7 Appendix

7.1 Changes to RASD

At the actual state of thing, first version of the design document, there are no changes to the RASD.

7.2 Tools

The tools we used to create this design document are:

- Google documents: to redact and write this document;
<https://docs.google.com/>
- StarUML: to create User Experience Diagrams and Component Diagrams;
<http://staruml.io/>
- Visio 2013: to create Sequence Diagrams;
<https://products.office.com/en/Visio/visio-pro-for-office-365-online-diagram-software>
- MacTeX: to write the algorithms
<http://www.tug.org/mactex>

7.3 Working time

Group total amount : 99

- Antenucci Sebastiano 790021

Total amount : 49

| Date | Worked hours |
|--------------------|--------------|
| 17th November 2015 | 3 |
| 18th November 2015 | 2 |
| 19th November 2015 | 3 |
| 20th November 2015 | 3 |
| 21st November 2015 | 4 |
| 22nd November 2015 | 3 |
| 23rd November 2015 | 2 |
| 24th November 2015 | 2 |
| 25th November 2015 | 3 |
| 26th November 2015 | 3 |
| 27th November 2015 | 3 |
| 28th November 2015 | 4 |
| 29th November 2015 | 3 |
| 30th November 2015 | 4 |
| 1st December 2015 | 3 |
| 2nd December 2015 | 3 |
| 3rd December 2015 | 2 |

- Buonagurio Ilaria 852641

Total amount : 50

| Date | Worked hours |
|--------------------|--------------|
| 17th November 2015 | 4 |
| 18th November 2015 | 4 |
| 19th November 2015 | 3 |
| 20th November 2015 | 3 |
| 21st November 2015 | 3 |
| 22nd November 2015 | 5 |
| 23rd November 2015 | 2 |
| 24th November 2015 | 2 |
| 25th November 2015 | 3 |
| 26th November 2015 | 4 |
| 27th November 2015 | 4 |
| 28th November 2015 | 5 |
| 29th November 2015 | 2 |
| 30th November 2015 | 2 |
| 1st December 2015 | 2 |
| 2nd December 2015 | 3 |
| 3rd December 2015 | 3 |