

|  |
| --- |
| Politecnico di Milano A. A. 2015-2016 |
| Design Document |
| Software Engineering 2: “myTaxiService” |

|  |
| --- |
| Stefano Sartini (Mat. 853697)  04/12/2015 |

**SUMMARY**

[1 Introduction 3](#_Toc436996168)

[1.1 Purpose 3](#_Toc436996169)

[1.2 Scope 3](#_Toc436996170)

[1.3 Definitions, Acronyms, Abbreviations 3](#_Toc436996171)

[1.4 Reference Documents 3](#_Toc436996172)

[1.5 Document Structure 4](#_Toc436996173)

[2 Architectural Design 5](#_Toc436996174)

[2.1 Overview 5](#_Toc436996175)

[2.2 High Level Components and their Interaction 5](#_Toc436996176)

[2.3 Component View 6](#_Toc436996177)

[2.3.1 Component Diagram 6](#_Toc436996178)

[2.3.2 BCE Diagrams 0](#_Toc436996179)

[2.3.3 Entity Relationship 0](#_Toc436996180)

[2.3.4 Registered User BCE 0](#_Toc436996181)

[2.3.5 Taxi Driver BCE 1](#_Toc436996182)

[2.3.6 Upload Position 2](#_Toc436996183)

[2.3.7 Taxi Request BCE 3](#_Toc436996184)

[2.3.8 User and Taxi Notification 0](#_Toc436996185)

[2.4 Deployment View 0](#_Toc436996186)

[2.4.1 Processing Sub-System 0](#_Toc436996187)

[2.4.2 Data Sub-System 0](#_Toc436996188)

[2.4.3 Visitor Sub-System 0](#_Toc436996189)

[2.4.4 Registered User Sub-System 0](#_Toc436996190)

[2.4.5 Taxi Driver Sub-System 1](#_Toc436996191)

[2.5 Runtime View 0](#_Toc436996192)

[2.5.1 Registration 0](#_Toc436996193)

[2.5.2 Log In 1](#_Toc436996194)

[2.5.3 Upload Location 0](#_Toc436996195)

[2.5.4 Request a Taxi 1](#_Toc436996196)

[2.5.5 Notification 2](#_Toc436996197)

[2.6 Component Interfaces. 0](#_Toc436996198)

[2.7 Selected Architectural styles and patters. 1](#_Toc436996199)

[3 Algorithmic Design 2](#_Toc436996200)

[3.1 Finding the correct queue Algorithm. 2](#_Toc436996201)

[3.2 Find Free Taxi in Queue Algorithm 3](#_Toc436996202)

[3.3 Taxi Drivers in Queue Managing Algorithms 4](#_Toc436996203)

[3.3.1 Adding a Taxi Driver Algorithm 4](#_Toc436996204)

[3.3.2 Removing a Driver Algorithm 4](#_Toc436996205)

[3.3.3 Moving a Driver to the End Algorithm 4](#_Toc436996206)

[3.4 Other Algorithms 4](#_Toc436996207)

[4 User Interface Design 6](#_Toc436996208)

[4.1 Registered User Interfaces 0](#_Toc436996209)

[4.2 Taxi Driver Interfaces 1](#_Toc436996210)

[5 Requirements Traceability 0](#_Toc436996211)

[5.1 Visitor Requirements Traceability 0](#_Toc436996212)

[5.2 Registered User Requirements Traceability 0](#_Toc436996213)

[5.3 Taxi Driver Requirements Traceability 0](#_Toc436996214)

[5.4 Non Functional Requirements Traceability 0](#_Toc436996215)

# Introduction

## Purpose

This document will describe the architectural design of the application, analysing the different components and their interactions. The user experience with the application will be an important factor in defining its design and architecture. The main algorithm will also be described to give a rough idea on how the application will work. All this with the Data Base Design will be used to give a consistent description of the application’s design. More precise definitions of the software will be decided during the implementation phase, this is more of a guideline for how the application should be structured.

## Scope

The purpose of this Design Document is to focus on the design of the software to be with regards to the relationship between system and user. The application that will be developed is “myTaxiService”, which will allow registered users to request a taxi through mobile or web application. The user request is forwarded to a taxi driver who can accept or decline the request. For more information regarding the application functionalities refer to the Requirement Analysis and Specification Document.

## Definitions, Acronyms, Abbreviations

Here the main definitions used throughout the document will be illustrated to avoid any misunderstandings.

* CLIENT: Client defines the part or parts of the application that will be running on mobile devices or web application.
* SERVER: Server defines the part of the application that will be running on a single computer in the taxi station, and that will communicate with every other part of the distributed application.
* DATA BASE: The physical hard disk where important and non-volatile information of the application will be stored.
* DATA BASE MANAGEMENT SYSTEM (DBMS): The part of the application in charge of managing the data base, every operation done on the data base is performed by the DBMS.
* ALTRO COL TEMPO

## Reference Documents

* Specification Document: Assignments 1 and 2.pdf
* IEEE standard for Requirement Specification.
* IEEE Standard System and Software Engineering- Architecture Description.
* IEEE Standard for Information Technology-System Design-Software Description.

## Document Structure

The document is composed of the following parts:

1. **Introduction:** The main document characteristics will be described in this section.
2. **Architectural Design:** In this section the document’s design will be analyse in detail, giving a clear explanation of the choices made and the relative impact on the application’s functionalities.
3. **Algorithm Design:** This part will contain the description of the main algorithms that the application will be using to fulfil its goals.
4. **Requirements Traceability:** This last mayor part will describe the relationship between the design choices and the requirements described in the RASD.
5. **References**

# Architectural Design

## Overview

MyTaxiService will be running on a web application and mobile devices, and it has to keep track of all the taxis in the different queues and all the user request. Because of this the chosen architecture is Client/Server and Java EE will be the environment in which it will be developed. Other architectural styles were considered, such as Service-Oriented Architecture or a Repository-Based system, but the client/server style is the one that fits best the application requirements.

## High Level Components and their Interaction

As described in the previous chapter the application will be developed using a client/server architecture based on Java EE. This means it will be a four-tired architecture divided in:

* Client Tier: Running on the client machine.
* Web Tier: Running on the Java EE server.
* Business Tier: Running on the Java EE server.
* Enterprise Information System (EIS): Managing the database and its information.

The client tier is the distributed part of the application which will run the user interface on a web application but also on mobile devices. All the main browsers like Internet Explorer, Safari, Chrome and Firefox must be supported for the web part of the application through the use of beans, while the mobile part will have to be developed for the three main operating systems: IOS, Android and Windows Phone.

The Web Tier is the part of the application that allow the client Tier to communicate with the server through the use of web pages or servlets if the client is running in the browser. For the mobile use of the application, the client tier will communicate directly with the business tier through the Application Clients.

The Business Tier components usually address the need of particular business and are composed of Enterprise Java Beans, which implements the logic, and Java Persistence Entities.

The EIS operates as a DBMS managing all the operations done on persistent data, such as reading and writing from the database.

## Component View

Before analysing the different components that make the application it’s better to study the Sub-Systems, to make clearer the system functionalities and which one are shared across different components. These are the resulting sub-systems:

* Processing Sub-System
  + Log In Sub-System
  + Registration Sub-System
* Registered User Sub-System
* Taxi Driver Sub-System
* Visitor Sub-System
* Data Sub-System

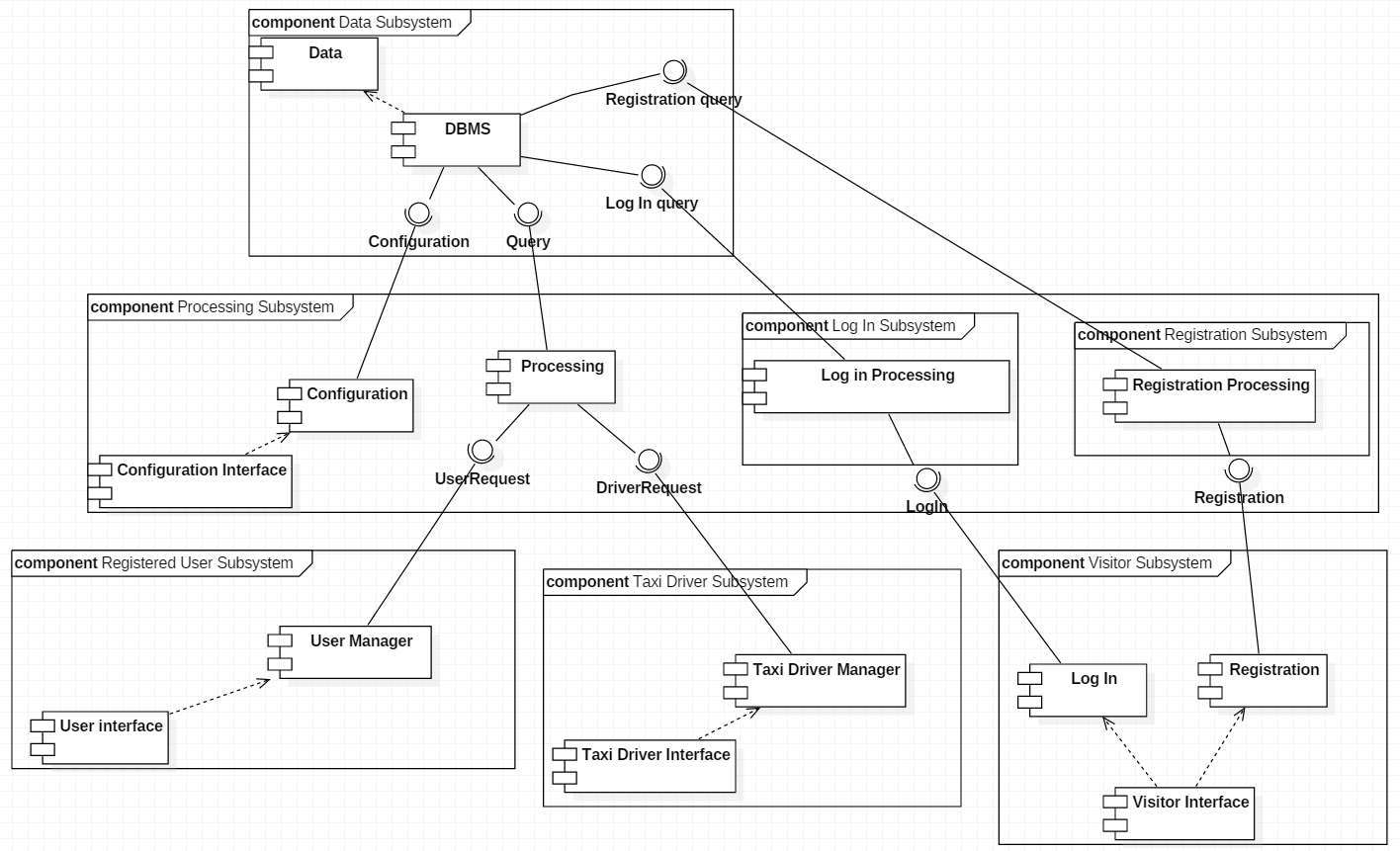
### Component Diagram

The visitor subsystem is the part of the application that runs while the user is still not identified as a registered user or a taxi driver. Once the user is identified one of the two subsystems starts to run. The log in and registration subsystems are part of the processing subsystem since they also have to process the user input and then communicate with the DBMS. In the following page the UML Component Diagram of the different components that are part of the subsystems is shown. Here is a brief description of the different components and how they interact:

The log in and Registration components are the same for every user, in fact a user that hasn’t yet logged in is defined as Visitor and so the components that handle these functionalities are part of their own subsystem. Also the component that displays the visitor user interface is part of this subsystem. Once a user has logged in he is identified as a registered user or taxi driver. Since these two type of users have different functionalities, they are in two different subsystems. Both these two subsystems have a component that handles the logic and the communication with the server and a component that depends from the previous one to display the user interface.

The processing subsystem has two subsystem of his own: the log in subsystem and the registration subsystem are the part of the application that receives input from a user processes it, checks if it is correctly formed and then forwards a query to the DBMS to obtain information about a user, or to write on the DB. This is the main reason why they are two different subsystems with different components: the log in subsystem can only ask the DBMS to read from the DB and then uses this information to verify if the user log in information is correct. The Registration subsystem does a similar thing to check if the input is correct, but then it also has to be able to forward a writing query to the DBMS and so needs different functionalities but also more accurate security implementations. The processing component is the one that has to communicate with taxi drivers, registered users and also the DBMS. This is the heart of the application and it has to implement most of the functionalities, such as taxi and queue handling and user and driver notification. In the processing subsystem, which will be deployed on the server running in the taxi central (more on deployment in the next chapter), there is also a configuration component with its own user interface, which will be used to add taxi driver accounts to the system or to do maintenance on the DBMS.

The last subsystem is the Data subsystem which has two main components: The data component which is the Data Base of the application and contains all the information about accounts, queues and taxi numbers. The Data Base Management System which receives all the queries from other component and reads or writes the Data Base to satisfy the queries.



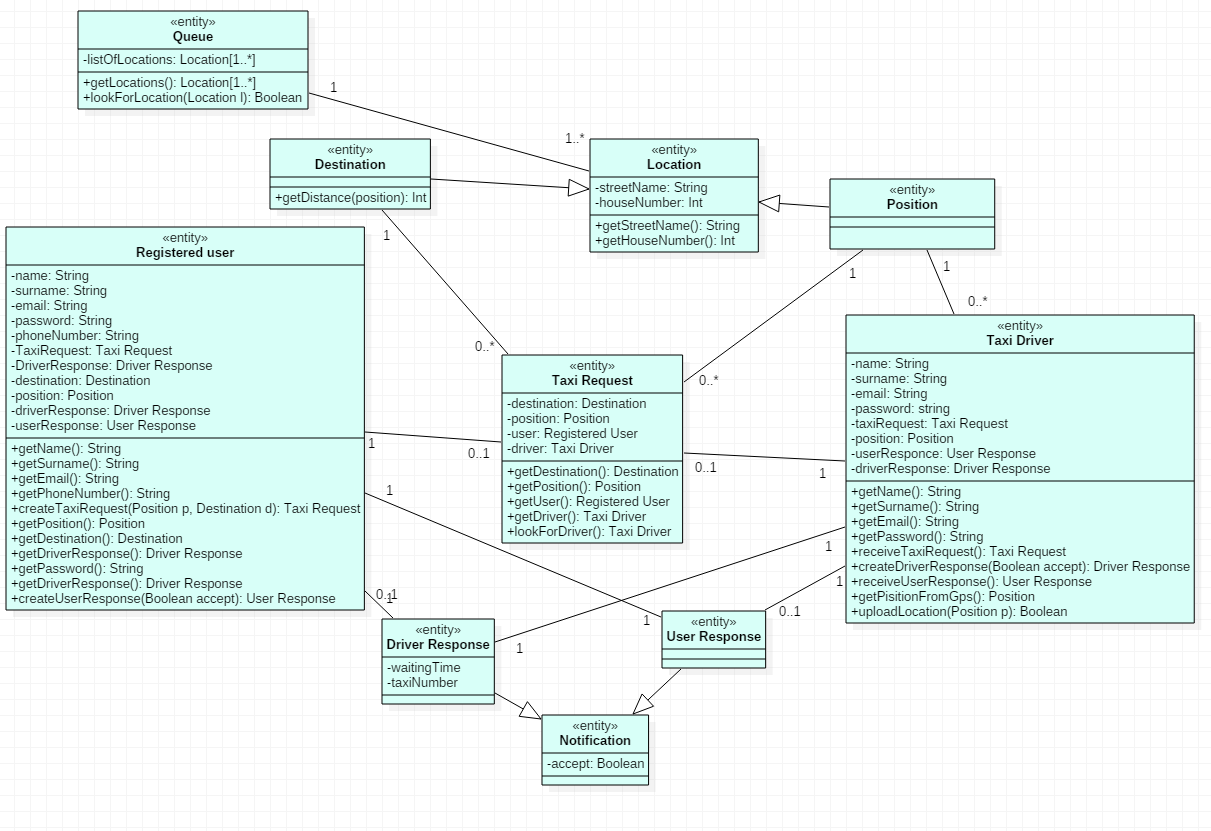
### BCE Diagrams

In order to give a better description of the different components and their interaction in this part of the chapter some Boundary-Control-Entity diagrams will be displayed. BCE is a variant of MVC where Entities are the data model of the application, the boundaries define the view which allows the display of information contained in the entities while the Controllers implement the logic that is used in the system. The proposed diagrams are:

* Entity Relationship
* Registered User BCE
* Taxi Driver BCE
* Upload Location
* Taxi Request
* User and taxi Notification

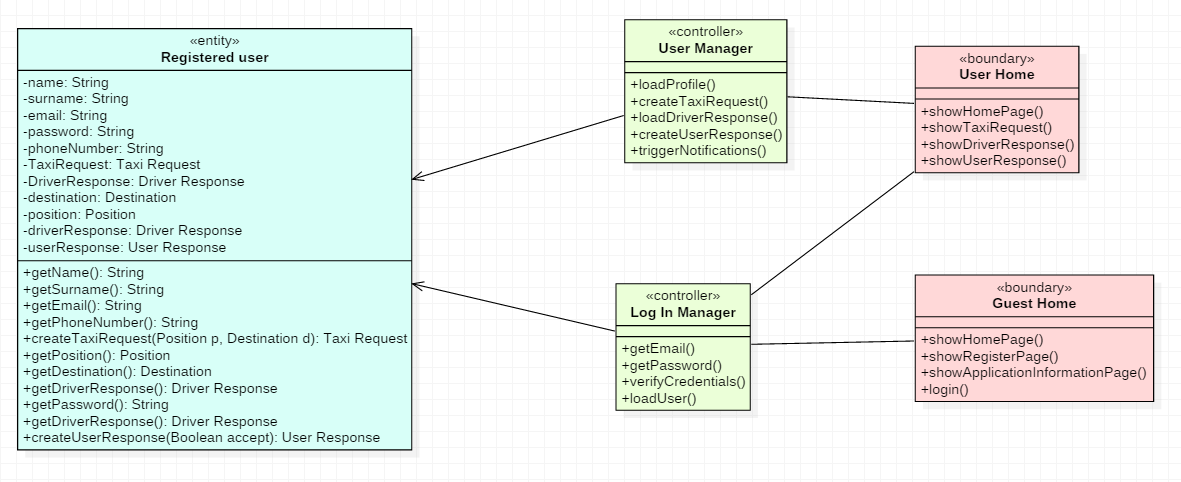
### Entity Relationship

This first diagram show the relationship between the different entities of the system; it is in some ways similar to the class diagram, but it also shows more attributes and more functions of the various entities. In addition it adds the Queue entity and its functions. Position and Destination are derived from Location with the difference that Destination has an operation able to determinate the distance from another Position. Every taxi driver has a Position while the User doesn’t, but the Taxi request made by a user has both a Destination and a Position. One a taxi driver receives a Request, he creates a Driver Response which is received by the User. The Driver response contains the waiting time and the taxi number, but also if the driver has accepted or not the request.



### Registered User BCE

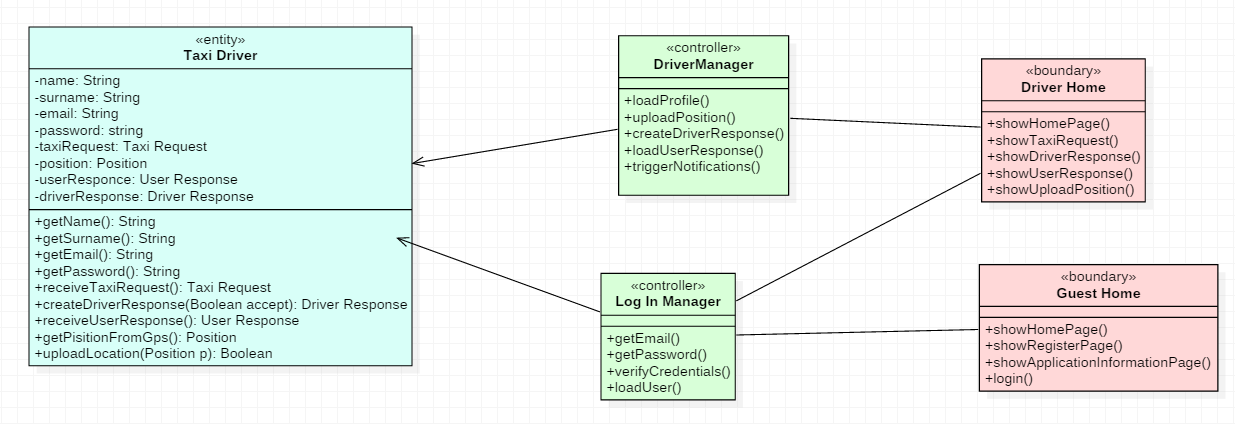
This show what the user has to be able to do and see when he still hasn’t logged in or he has just logged in. The log in manager allows visitors to log in to their personal accounts and display the basic information about the application with the use of the Guest home Boundary while the user manager has to display all the user and application information and then redirect the registered user to the taxi request page. The log in manager also has to check for valid input and only then verify if the email and password submitted are a registered account credentials. Once the user has successfully logged he has full access to the taxi request page through the User Home Boundary.





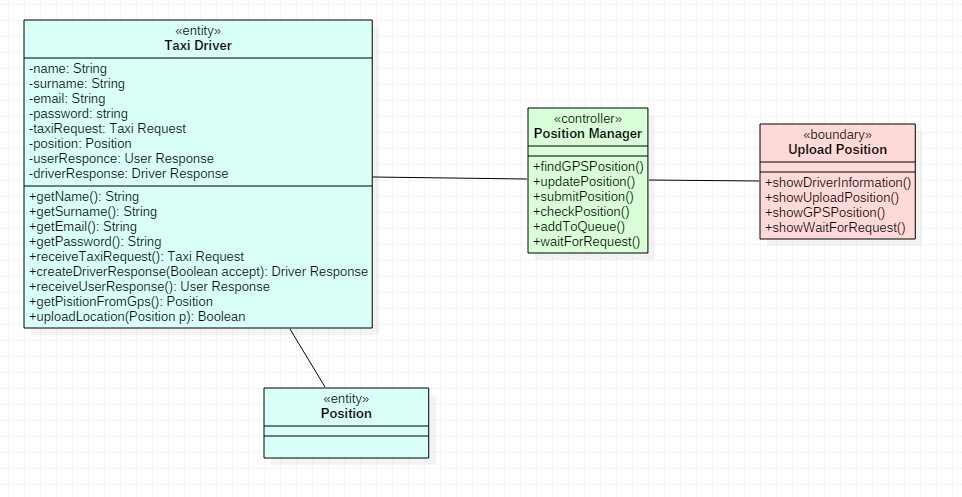
### Taxi Driver BCE

This diagram represents the log in and authentication process for the taxi driver which the same as for the Registered User. The only real difference is in what the taxi driver can do once logged in: While the user manager remains the same different information will be displayed on the screen and a different web page will be available to the taxi driver.



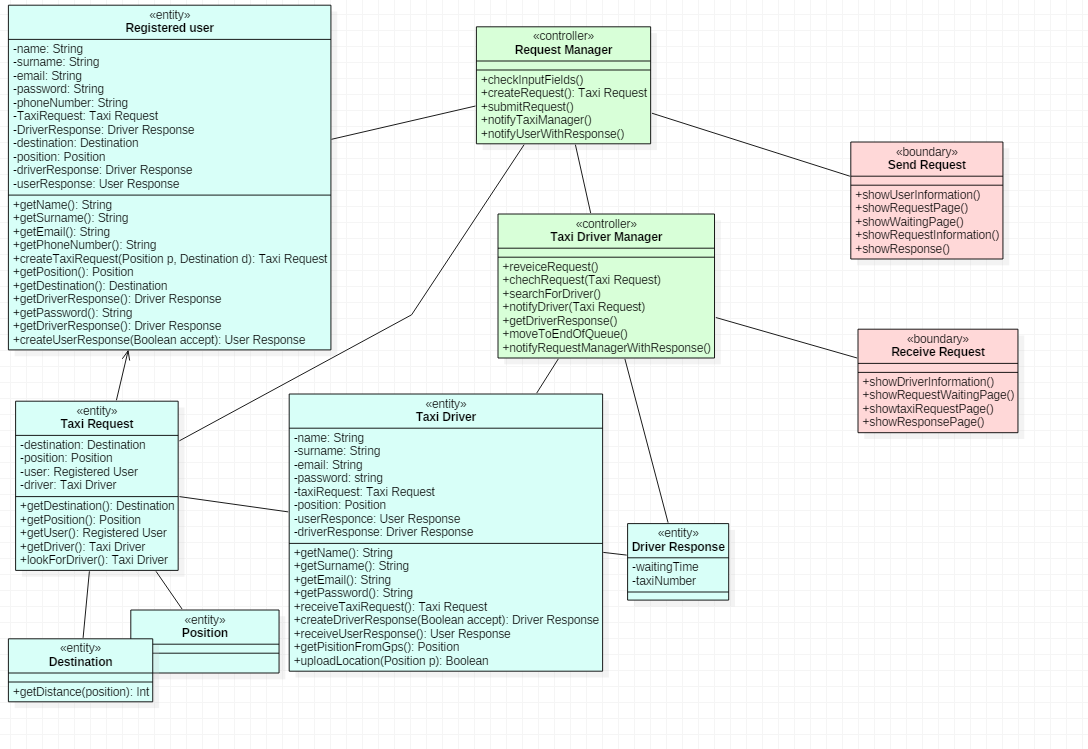
### Upload Position

This diagram in specific for taxi drivers since they are the only ones that upload their position in this way. The Position Manager controller has the task to use the GPS integrated in the mobile devices to get the taxi driver location and change the information in a street name and house number, and then upload the position to the system so that the driver is inserted in a new queue. The upload location boundary uses information given by the controller to display the user interface for this particular part of the application.



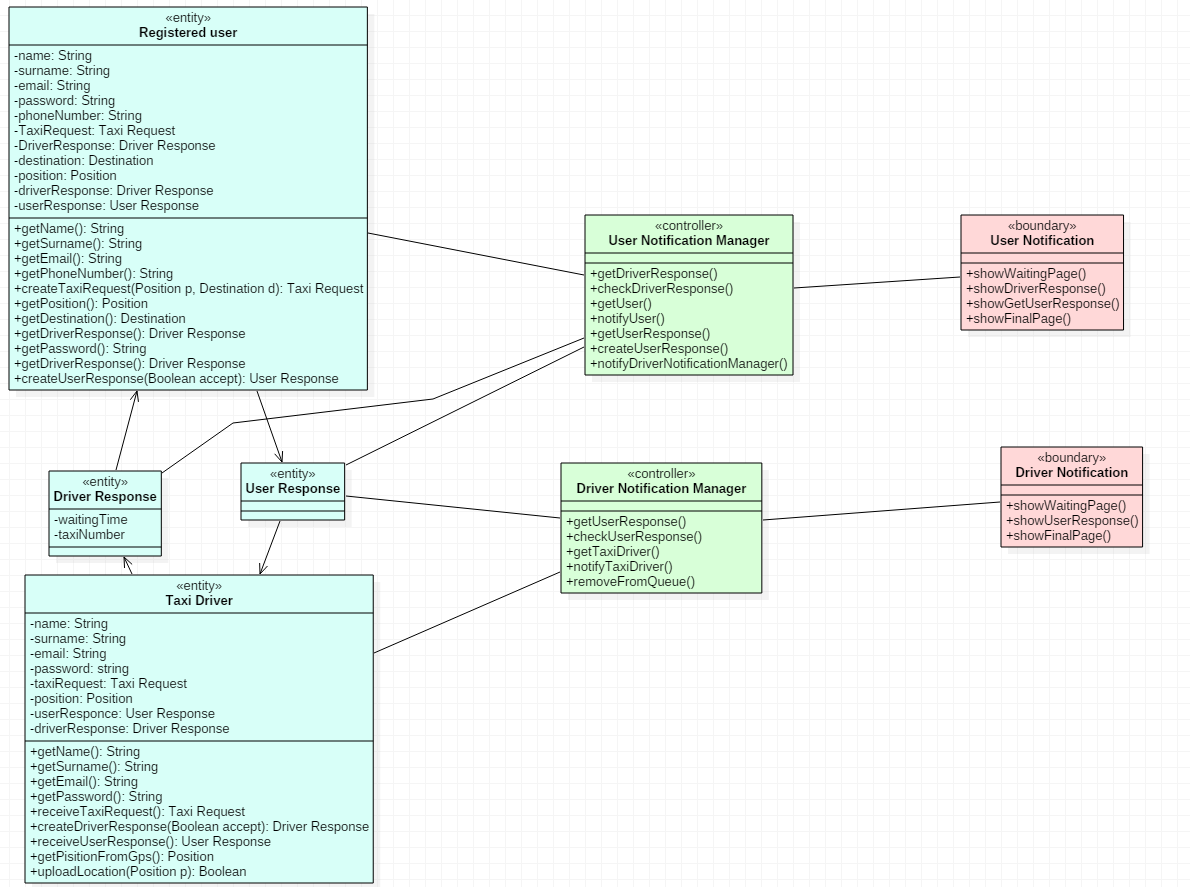
### Taxi Request BCE

This model shows how taxi requests and driver response are handled by the request manager and the taxi driver manager. The request manager has to check the input field the user insert to prevent errors in more crucial phases of the application. It also has to create the request once the fields are verified, inform the system and once the system has found a taxi it has to inform the taxi driver manager. This controller will check the request he received and communicate it to the taxi driver through the user interface. In a future phase it also has to collect the driver response.



### User and Taxi Notification

Once the driver has given his response two more controller will check the correct work of the application. The user notification manager will collect the driver response and forward it to the user that made the original request. Once the user has answered it will store the answer and notify the taxi driver notification manager, which will consequently notify the taxi driver of the user decision. In case of negative response by the taxi driver, the user notification centre will also tell the system to remove the taxi driver from the queue.



## Deployment View

In this chapter the different components that are described in the previous chapter will be described in terms of their deployment on the different hardware parts the application will be running on. Using the Java EE multi-tiered division of the hardware, there are four different hardware categories on which the application will be spread:

* Client Tier.
* Web Tier.
* Business Tier.
* Enterprise Information System (EIS).

Every subsystem will be located in one of these tiers and how it interacts with the other subsystems will be defined.

### Processing Sub-System

This part of the application has to be running all the time on a computer in the taxi station, it is part of the business tier and the web tier and it is deployed on a computer in the taxi station. The Configuration component is part of the business tier since it has direct access to the DB, while the other component are part of the Web Tier. Its main tasks are to keep track of all the different taxis in the queues, receive taxi request from users and pair requests with taxis. In order to do this it has to be able to communicate with every other subsystem: it will communicate with both web and mobile devices and also with the DB through the DBMS. The communication with mobile devices and web application is going to be TCP/IP since it is a reliable and cheap way to connect the clients with the server. Since the DB will be running SQL, the processing subsystem will use this language to communicate with the DB. This subsystem has two minor subsystems, the registration and log in subsystem. These two will as well be running on the same computer but in parallel with the main processing subsystem to allow it to never stop its task to verify a user log in or to register a new user. This should guarantee a better and more fluid experience for registered user and taxi drivers since the main processing subsystem isn’t influenced by the load of registering or logging in users.

### Data Sub-System

The data subsystem and its components are part of the Enterprise Information System and as such they will be deployed on a computer in the taxi station. This can be a different computer from the one used for the processing Subsystem or it can also be the same. Obviously a different computer would allow a better overall experience and a faster data analysis and access. Anyhow the usage of only one computer for both these two subsystem and all their component should be enough to guarantee a good experience. As stated before, only the DBMS component communicates with other processing components through SQL.

### Visitor Sub-System

This subsystem is part of the client tier and as such is deployed on web application and mobile devices. It will communicate with the log in and registration components through TCP/IP and will not have direct access to the DB, but only through the log in and registration components.

### Registered User Sub-System

Just like the Visitor Subsystem this is part of the client tier and deployed on both web application and mobile devices. It communicates directly with the Processing component. This part of the application

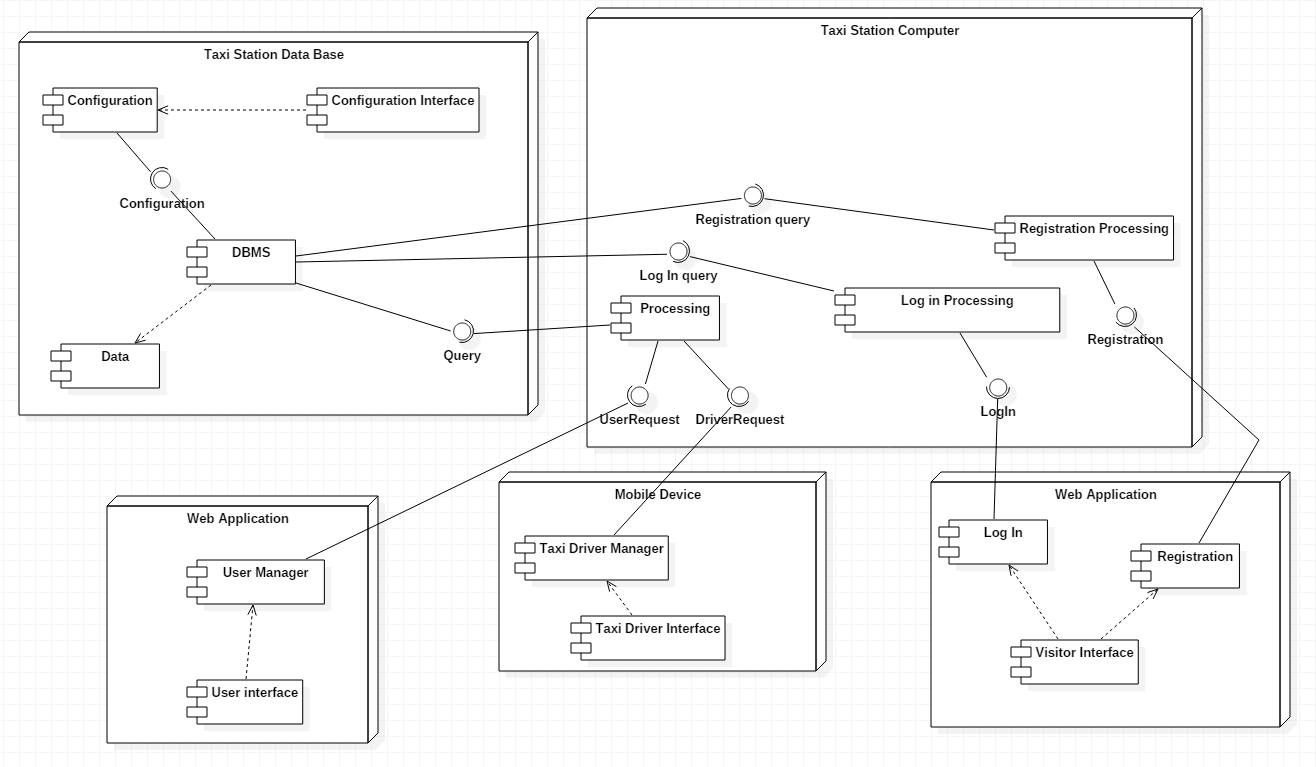
### Taxi Driver Sub-System

This subsystem is deployed only on mobile application, since it is impossible for taxi drivers to have access to a computer and a web application. It has to be able to communicate with the GPS component integrated in the mobile phone and have internet access since the communication with the processing component is done through TCP/IP.

In the next page there is a deployment diagram to give better understanding of the deployment of the different components. It was considered that the Data Base and the processing subsystem are on two different computers. Since there are two different computer for those two subsystems, it is better to deploy the configuration and the configuration interface components on the same computer as the DBMS so that these components can work and manage the DB more efficiently.

The next page diagram describes the distribution of the components on the different hardware, but it does not take in consideration the BCE diagrams, so a few words must be spent here. All the boundaries will be deployed on the mobile devices of web application, and will communicate with the controllers, which are part of the Processing subsystem and as such they will be deployed on a computer in the taxi station. The entities are data structures, and as such will be stored in the data component and deployed on a separate computer in the taxi station.

On second thoughts the User manager and Driver Manager Components could probably be deployed on the same computer as the processing subsystem, since they do most of the processing for the user functionalities. In this way we achieve a thinner client, with only user interfaces, and that will allow it to be installed on more devices. Furthermore it will higher the security level since all the important communication with the DB is done from a computer in the taxi station.



## Runtime View

In this chapter of the document different sequence diagrams will be presented. Some sequence diagrams were already presented in the RASD, but in this chapter the focus will be not on the single component, but on how different components interact with each other. The use cases taken from the RASD that will be analysed are the following:

* Registration
* Log in
* Request a Taxi
* Upload location
* Notification

A few things have to be clarified about the following diagrams, starting from the fact that the BCE diagrams of the previous chapters are the base line for the definition of the sequence diagrams of this chapter. The components of the component diagram could have been used, but in the BCE more function were already defined so those were used in this sequence diagram to give continuity to these chapters.

Some other used functions are not specified in the BCE models, such as the error messages, because they are considered to be implemented in every class of the system.

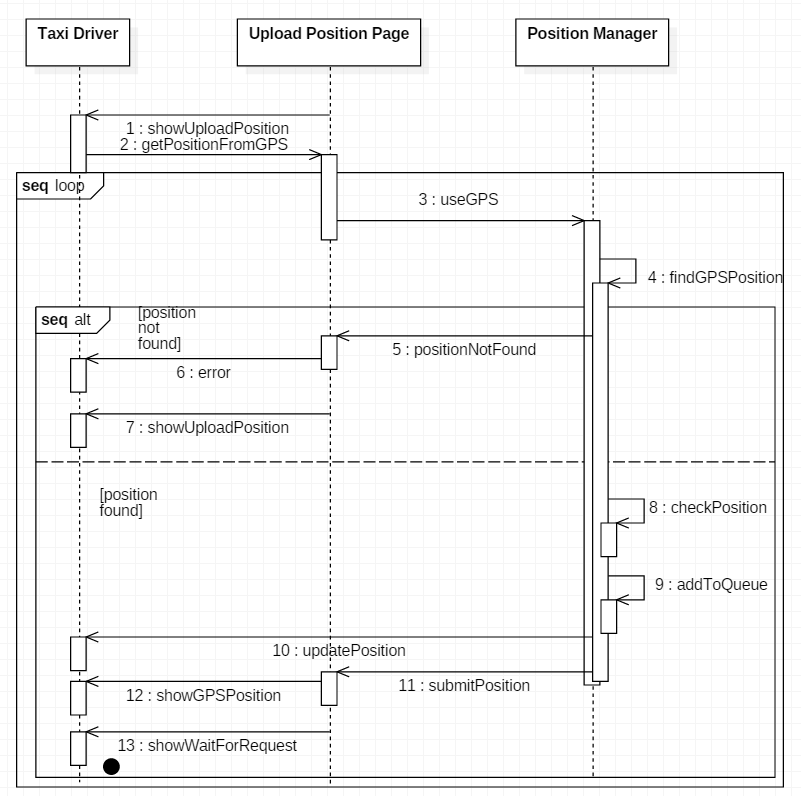
In the Log In sequence diagram there is only one user, the Visitor, which should not have access to functionalities, but once he has logged in he is considered to be a Registered User or a Taxi Driver, and so the two different boundaries display to him the web pages not anymore as if he is still a visitor.

The request a taxi sequence diagram comprehends all events up until the driver is notified by the system of a taxi request, accepts or declines the request, and the User is notified with the final decision of the taxi driver. The notification sequence diagram starts from here, when the system notifies the user with the driver response and describes what happens when the user finally accept or declines the taxi and the driver is notified.

### Registration

### Log In

### Upload Location



### Request a Taxi

### Notification

## Component Interfaces.

Using the component diagram the main interfaces through which the components interact can be defined:

* Configuration: This interface allows communication between the DBMS and the configuration component which is used to check and maintain the DBMS. The configuration component uses SQL queries to have access to the DB and modify it. It has full access to all SQL functionalities and the security is granted from the fact that this component is deployed on a computer in the taxi station, so who wants to modify the DB has to physically be in front of the computer.
* Query: This interface is used by the processing component to access the DBMS and is based on SQL. The processing component has to be able to both write and read from the DB since it will tell the DB when a driver should be added to a queue and when to remove him or move him to the end of it.
* Log In Query: Used by the log in processing component this interface is based on SQL with the difference that this can only read form the DB, since the log in component only has to check if the inputted credentials are correct, but doesn’t have to modify the DB structure in any way.
* Registration Query: This interface is used by the registration component and uses SQL queries to both write and read from the DB since it has to be able to check if and Email has already been used in the system, and otherwise save all the user information for further log ins.
* User Request: This interface allow the communication between the processing component and the User Manager. Through the TCP/IP standard the user manager send the request to the processing component, and the processing component will send back the driver response and information containing waiting time and taxi number.
* Driver Request: This interface is similar to the previous one since it uses TCP/IP, but in this case what is transmitted is the user request to the taxi driver, and the driver response to the processing component.
* Log in; the communication between the log in component on the client and the log in processing is simply the pair email and password, the processing component for the log in will then ready from the DB and give access or not to the user.
* Registration: Similar to the previous interface, in this case all the information needed to register are transferred through TCP/IP: email, name, surname, phone number and password.

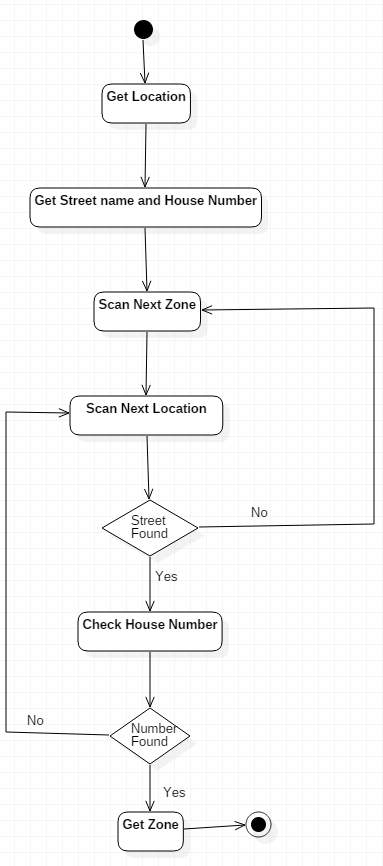
## Selected Architectural styles and patters.

The main architectural style used in this document is Client-Server. The client application on the user and driver mobile devices invokes the server for its main functionalities such as adding a driver to a queue or submitting a request. The server part of the application will handle all communication with the data base and will do most of the processing. The main reason for this choice is that this allows the application to be developed using thin clients, which will help the application to run on as many devices or browsers as possible, while the fat servers in the taxi station will do most of the complex and more sensible computation. Other styles were considered, such as Service-Oriented Architecture or a Repository-Based system, but no real advantage would have been gained from those patters. Moreover Client-Server is a very common and spread style, which almost every developer know how to implement. The Java EE environment will be used, so as described in the high level components and their interactions chapter the application will be divided in four tiers: Client tier, Web Tier, Business tier and Enterprise Information System. The Client tier is the one in charge of the presentation, while the other tiers handle the other functionalities of the application.

# Algorithmic Design

In this third chapter of the Design Document the main algorithm of the application will be described using both natural language and flow charts, to show how the application will make decisions and how it will be able to fulfil its functionalities. There are three main algorithms which complexity and workflow should be described prior to the development of the application: The first one is the one that starting from a taxi request selects the correct queue from which selecting the taxi driver. The second one is the one that is executed right after the first one, as it checks every taxi member from the queue and selects the first available one and notifies him of the request. This second algorithm also has to remove the taxi driver from the queue once he has accepted a customer or move him to the end if he declines. In other words we have two main algorithms: one for finding the queue, one for finding the taxi driver from the queue. The Third algorithm is the one that handles all the adding and removing of taxi drivers from the queues. All other operation done by the application do not present the necessary complexity to study them in details. For a guide line on how the rest of the application should work, some state chart diagrams were presented in the RASD document.

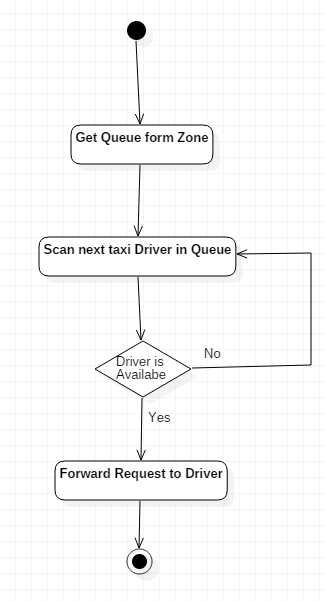
## Finding the correct queue Algorithm.



This first algorithm that will be analysed has the task to determinate the zone in which the taxi request has to be forwarded. This same algorithm is also used when a taxi driver uploads his location to the system and the system has to identify the correct zone in order to put the driver in that zone’s queue. In both cases the system has to determinate a zone from a location, which is made of a house number and a street name. Every zone has a list of locations so the algorithm will have to search in every list for the corresponding location. If *n* is the number of zones saved in the Data Base and *m* the average number of street names per zone and *k* is the number of house numbers per street name, the algorithm will a complexity of O(*nmk)*.As specified in the reference document, every zone is approximately 2 square kilometres, since in 2 kilometres square so for a city such as Milano there would be 90 zones while the streets are almost 4000. Down here a flow chart of the algorithm workflow will be presented. No functions of this part of the system were already presented so natural language will be used in the diagram. There is a different way to solve this problem, and it is by adding a zone attribute to every Location, so the information will be contained in the location itself, without having to look for it, but this would mean a bigger memory needed to story redundant data.

## Find Free Taxi in Queue Algorithm

As mentioned before this algorithm usually runs right after the previous one and looks for the first free taxi driver in the queue. If there are *t* taxi driver in the queue, it’s complexity is, in the worst case O(*t*), but in most cases the first taxi in the queue could also be the first free taxi and so the complexity is lower. An explanation has to be given before proceeding describing the algorithm: Never in the RASD or in this document is described what happens to the taxi driver in the queue while he is communicating with a user, because the taxi driver is removed from the queue only when the user accepts the taxi and the waiting time, so until that moment the driver is still in the queue. The easiest proposed solution is to use the variable “available” associated to the Taxi Driver entity. This variable is Boolean, and true when the driver is waiting for a request and as soon as a request is presented to the driver, this variable is set to false. The system then, when searching with this algorithm for a taxi driver inside a queue always checks the first one and if he is available it forwards the request, otherwise it jumps to the second one and so on. This justifies the O(*t*) complexity, since in the worst case all taxis but the last one could have already received a request but are waiting for a user response. The diagram here show the flow of the algorithm and how it looks for a free driver. The queue used in this system is FIFO, first in first out, since the first taxi driver to be added to a queue is also the first to be notified by a taxi request, but it might happen that the first taxi driver was already notified so the next one will be considered and so on. This is not a very complicated algorithm but since it deals with an aspect of the application that was never described before it was included in this section.



## Taxi Drivers in Queue Managing Algorithms

This third algorithm has to manage three actions on the queues: Adding a taxi driver, removing a taxi driver and finally moving a taxi driver to the end of the queue.

### Adding a Taxi Driver Algorithm

Adding a taxi driver is not a very complicated task, since the algorithm only has to pass through all the queue positions and then once it has reached the end, add the taxi driver, just like a normal FIFO queue. In this case the implementation of the queue doesn’t have an impact on the complexity since it will always be O(*n*), the time it takes to go down all the queue.

### Removing a Driver Algorithm

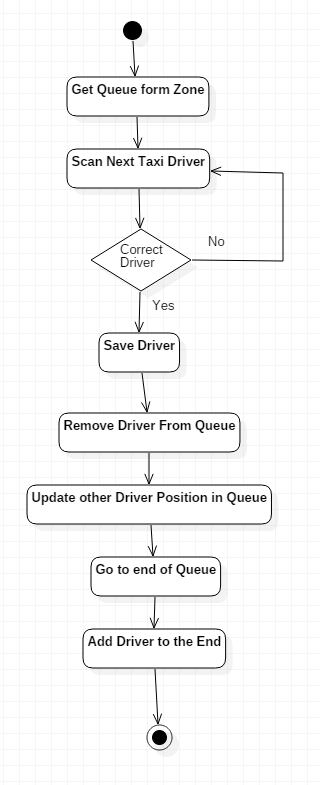
As previously stated the queue is FIFO. The problem is that when a driver is notified with a request, he is not removed from the queue, but only when a customer accepts, the driver is removed. In this sense it is not a proper FIFO queue, since if a customer takes a long time to answer, another taxi driver associated to another customer might be removed from the queue. The flow of this algorithm really depends on how the queue is implemented. For example if the queue is a table with only a single line in the Data Base, the algorithm, once it has found the driver to remove, has to then pass through every other taxi in the queue and move them up one position. In this the complexity is, if *n* number of taxi drivers, in the worst case is O(*n*) if the removed taxi is the first and then we have to move all the others. If we are using Java object so that every taxi driver has a pointer no another taxi driver in the queue, we just have to make it so that the pointer to the removed driver now points to the next one in the queue. In this case the complexity only depends on finding the taxi driver not removing him.

### Moving a Driver to the End Algorithm

This algorithm could actually be considered as a consequent execution of the previous ones explained in the chapter: the driver in fact has to be found, remove from queue and then added to the end, and for this reason a flowchart of this algorithm will be reported to give an idea at the complete functionality of the Queue Managing Algorithms. The only difference is that in this case the algorithm is looking for a specific driver, not for the first free one as described in chapter 3.2.

## Other Algorithms

A few other algorithms could be describer for the system, for example the algorithm that computes the waiting time, given the user position and the driver position. At this stage of the development the real implementation of this part of the application isn’t known, in fact there are two ways to implement this algorithm: the first way is to use an existing online map such as Google Maps to compute the waiting time, this will also take in consideration the traffic. The other option, which will require more development effort, is to store the speed limits and length of each road, then create a graph and use an algorithm to compute the shortest path from two points. The cost of this shortest path will be the waiting time. This solution however is more expansive and does not take in consideration traffic. Another algorithm could be used to check the input from the clients, in order to avoid forwarding queries to the DBMS if the input is obviously wrong (for example if the input requires an email and no emails were inserted by the user).



# User Interface Design

Some aspects of the user interface design were already presented in the RASD document, with focus on the web application. In this section some mobile versions of the user interfaces will be presented, and the interaction between the different pages and its components will be highlighted. This chapter will focus on the following User Interfaces:

* Visitor Home Page and Registration page
* Registered User Taxi Request
* Taxi Driver Upload Position
* User Notification
* Driver Notification

The diagram used in the following pages is a class diagram with the stereotypes <<screen>> for real pages visible by the user, <<screen compartment>> and <<Input form>>. The $ symbol next to a screen name means that ii is reachable from any other page of the system. The arrows in the graph indicate the flow of the interfaces seen by the user. Two diagrams will be proposed, one for users and one for drivers.

## Registered User Interfaces

## Taxi Driver Interfaces

# Requirements Traceability

In this last part of the document the requirements described in the RASD will be linked to specific parts of the application described in this document. Since the requirements were dived based on the different users, the same thing will be done here, starting from visitors.

## Visitor Requirements Traceability

As described in the RASD the visitor should be able to view the application information, log in or register to the application. All this is well described in the previous chapter about the interfaces, but also in the BCE diagrams with de Guest Home Boundary. In the BCE diagram also the controls done on the visitor input to register or login are show. The Log in and Registration components will in fact handle this verification part of the application, before communicating with the DBMS.

## Registered User Requirements Traceability

As shown in the input form of the user interface diagram of the previous chapter, the registered user has to insert destination and position to make a taxi request. These field will be checked by the processing component before it asks the DB for the list of taxi driver in order to associate the user request to a driver. The processing component, and in particular the User Manager will handle the first part of the communication between user and system. The User Notification Manager will handle all the notification received by the user such as the driver response. So the interface requirements for the registered user are satisfied by the User home, Send request and user notification boundaries, while the User manager, and request manager and user notification manager controllers handle all the verification on the input and what to show the user.

## Taxi Driver Requirements Traceability

The flow of the interfaces the driver sees is shown in the Taxi Driver Interfaces diagram with the corresponding screens and input forms. As for the user all the verification on the input is done but the processing component and in particular the Position Manager for uploading the location, Driver Manager for the response to request and finally by the driver notification manager which shows the driver the user final decision. In addition the Position manager also has to communicate with the GPS in the driver mobile device to be able to upload the position to the system. The controllers from the BCE diagram are the ones in charge of verifying all the driver input and allowing him to communicate with the system. These controllers are part of the processing component.

## Non Functional Requirements Traceability

The performance requirements are verified in the algorithm description of this document. Since the application has to be as real time as possible all the proposed algorithm don’t have a high execution time. The division of the Data Base and the processing component on two different computers will allow for a faster system that meets the requirements. Security will be handled by all the components of the process subsystem which will control the input from user before asking anything to the database. The addition of this subsystem between the DBMS and the user and taxi driver Manager is there to guarantee a more secure system. The system might lose in performance but it will gain in security. In this way only the processing subsystem in authorized to communicate with the DBMS, which will then manipulate the DB. Furthermore the addition of the configuration component which is deployed on the same computer of the processing component will make it so that anyone who wants to actively modify the DB has to be physically present in the taxi station to do so.