Software Engineering 2

Design Document

Authors
Paolo Paterna, Lara Premi

Politecnico di Milano, Italy
Email: paolo.paterna@mail.polimi.it, lara.premi@mail.polimi.it

4th Dec 2015

Contents

1 Introduction

1.1 Purpose

This document is the Design Document (DD) of a system, called myTaxiService, used to manage a taxi service in a city. The main goal of this document is to specify how the system has to be build in terms of software and hardware architecture, focusing on how the software architecture is built, what kind of styles are used and which kind of tiered architecture is used for the hardware part.

1.2 Scope

The scope of this system is to manage taxis in a city. A town is divided in zone of 2 square kilometers and, for each zone, the system defines a queue, composed by the identifier, which is the vehicle plate, of free taxis in that specific zone. A user can require a taxi ride from a zone, but he/she can also book one for another moment, using the web application or the mobile one. Furthermore, about long-term reservations, after the user has created one, he/she is able to modify the date or the hour or both of his/her booking, and he/she can delete it.

1.3 Acronyms and abbreviations

1.3.1 Acronyms

- RASD: Requirements Analysis and Specification Document from previous delivery

1.3.2 Abbreviations

- Gi: goal number i, that refers to the corresponding goal in the RASD.

1.4 Reference Documents

To redact this document, we used the following other documents as references:

- Requirements Analysis and Specification Document from previous delivery
- Design Document Table of Content

1.5 Document Structure

This document contains the schema that represents the software and the hardware architectures, the sequence diagrams that show how the system works and the design choices made to build the system. After that, there are some implementations in pseudo code, user interfaces and all the requirements present in the RASD, mapped in this document.

2 Architectural Design

2.1 Overview

The main architecture chosen for this system is the client-server one; clients access the service through the web server that is only the view of the system.

All services are inside the application server that exposes all the APIs needed to expand the system in the future.

All the data is stored inside a database that is on another machine.

This modular schema helps the system' maintainability because every component is independent from the other and they can be substituted without replacing other parts.

Furthermore, as architectural pattern, we have used the Model-View-Controller.

2.2 High level components and their interaction

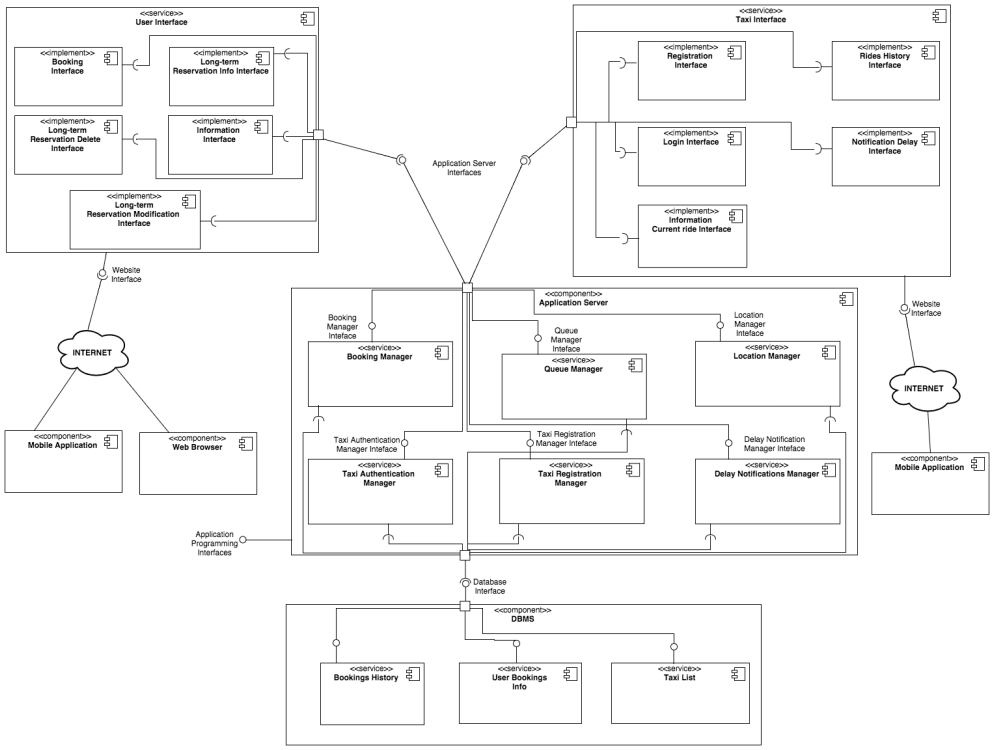
Starting from the user or from the taxi, every request is made to the web server that can serve all the static pages which contain only the basic structure and the graphics of the website. There are two types of interfaces:

- taxis can access to their interfaces to manage their rides;
- users can access only to the booking interfaces, where they can make a new booking or modify/delete a long-term reservation.

To populate all the pages with the needed data, the web server needs to ask the application server, that receives and elaborates the request: if there is the urge, it will query the database server to retrieve or save the needed data for the request.

The database server stores all the data received from the application server and it waits for queries.

2.3 Component view



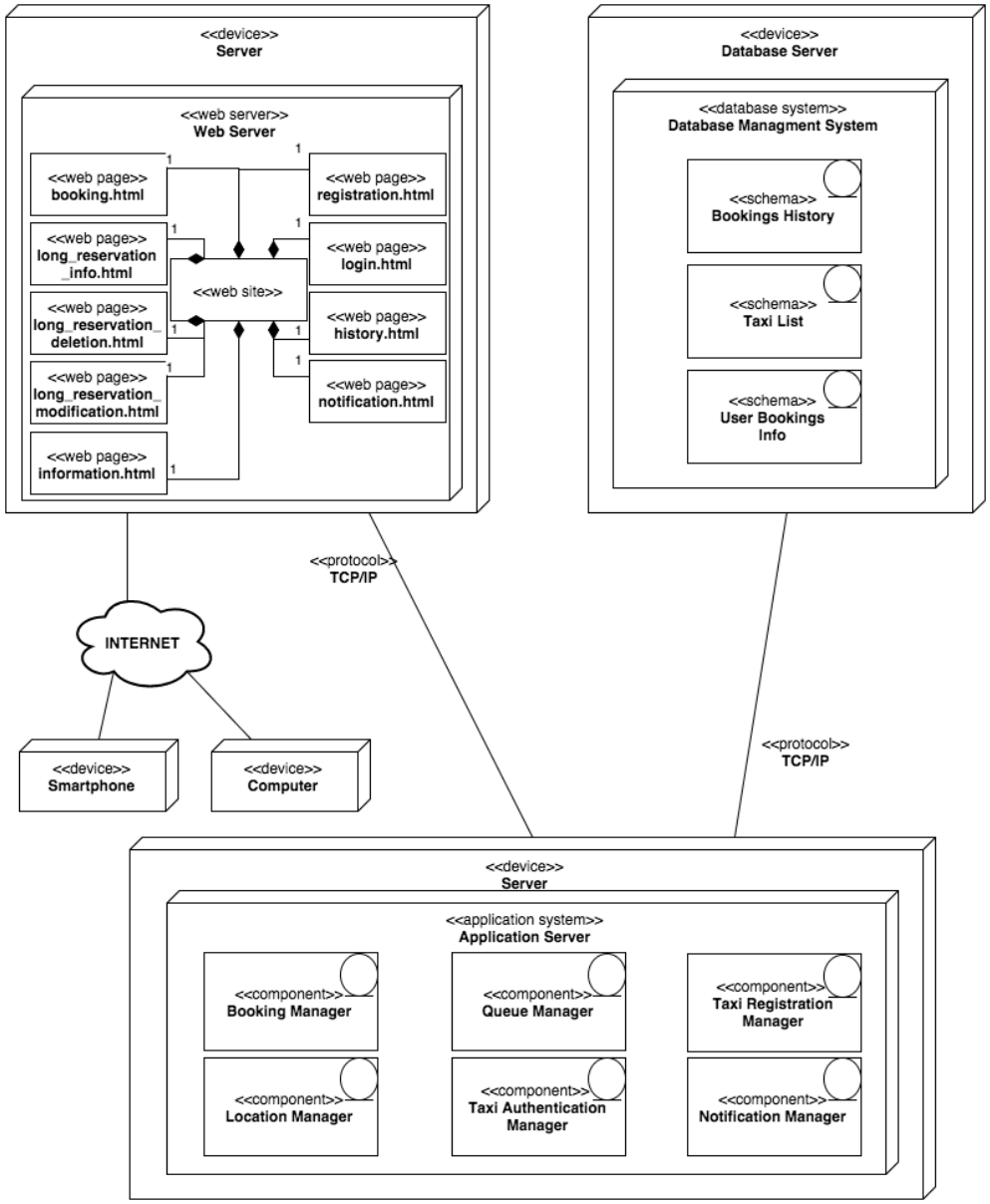
In the above diagram, through the Internet, using a web browser or, in alternative, a smartphone, users can access to their interfaces, that are about the booking, the modification of the long-term reservation, the long-term reservation' information, the service' information and the elimination of the long-term reservation.

In the case of taxis, through the Internet, only using a smartphone, they can access to their interfaces, that are about the registration, the history of the rides, the login, the notifications of delay and the information about the current ride.

These interfaces are populated with data, coming from the application server. This is composed of some services that manage the reservations, the queues of taxis, the locations of users and taxis (in the case of users, this manager checks also the validity of the time inserted), the taxis authentication, the taxis registration and finally the delay notifications.

The application server is the only component that can access to the DBMS, which is characterized by three data tables: the history of the booking, the users' bookings and the taxis list.

2.4 Deployment view



In the above diagram, through the Internet, using a web browser and/or a smartphone, users and taxis can access to the server, which contains the web server. Here, there are some pages that represent which stuff, clarified in the component view description, a user or a taxi can do.

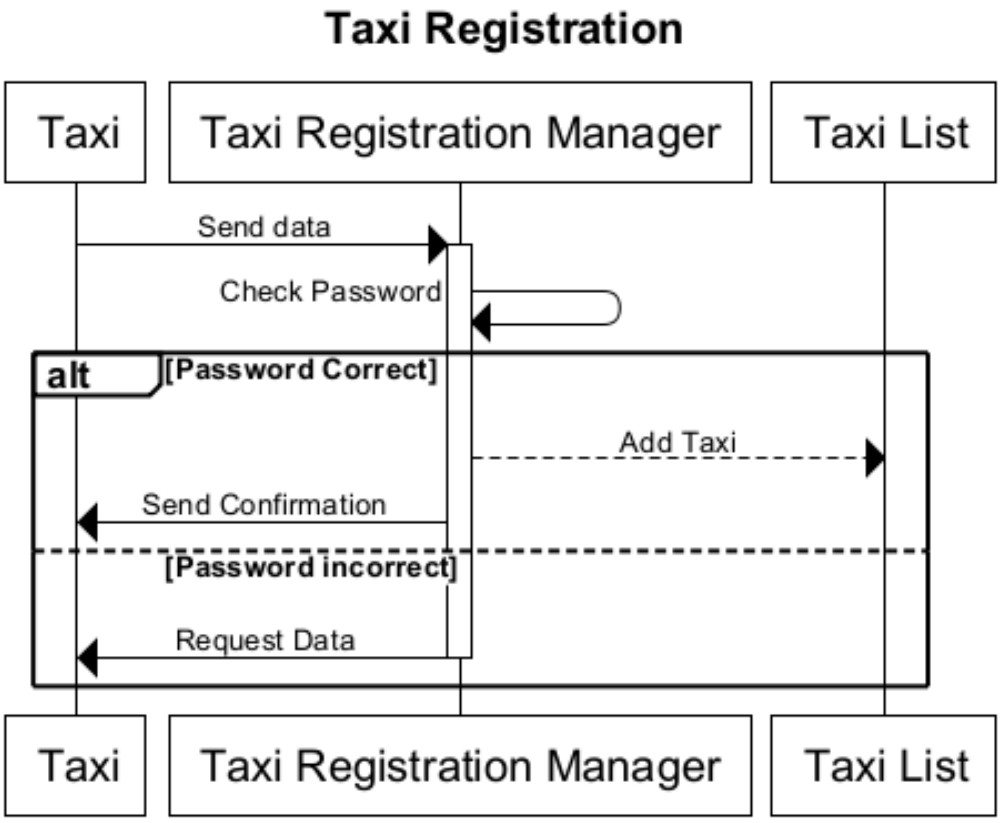
Thanks to the TCP/IP protocol, the server of the web one is connected to the application server: it contains all the application logic, that lets to the pages of the web server to be populated with data.

This application system is composed by the same managers, clarified in the component view description.

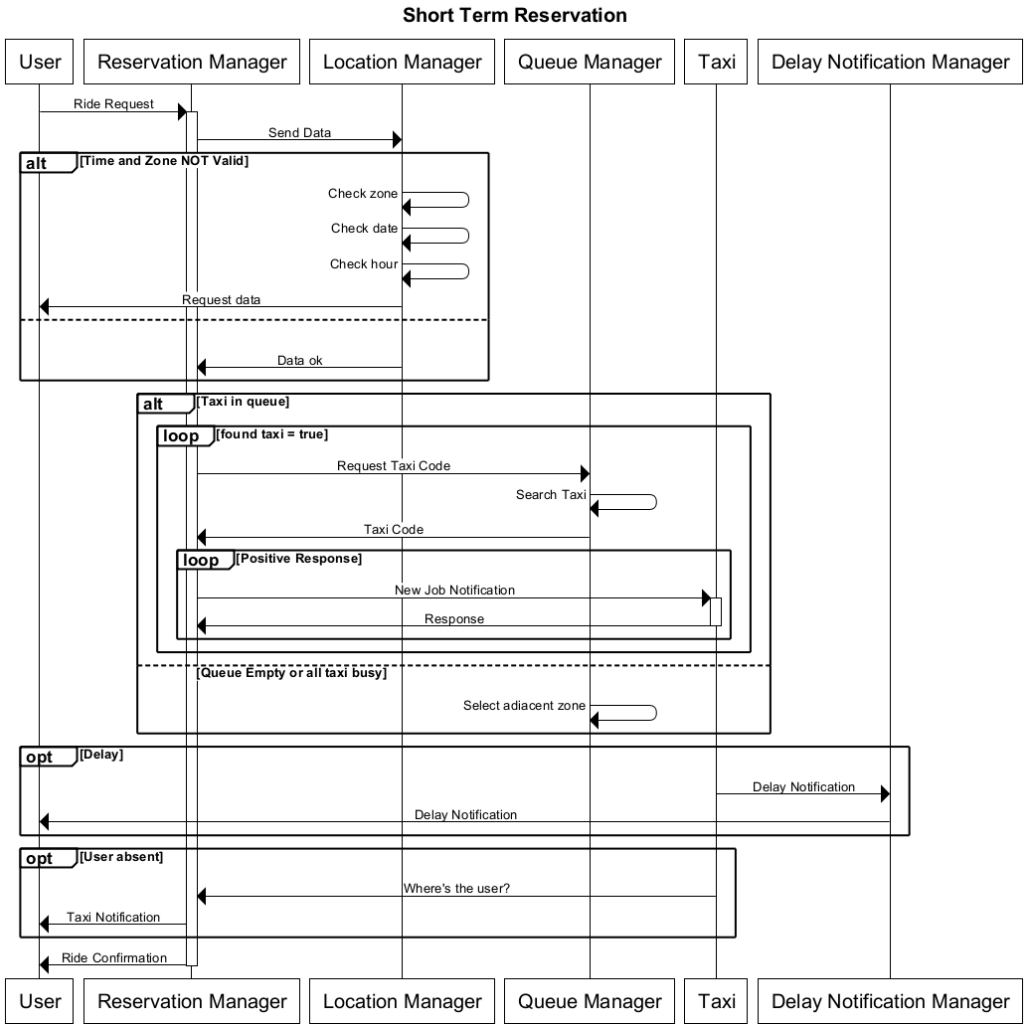
Finally, the server of the application one is linked, thanks to the TCP/IP protocol, to the database server: through this connection, the application server communicates to the web server the data from the database, which contains the same three tables, clarified in the component view description.

2.5 Runtime view

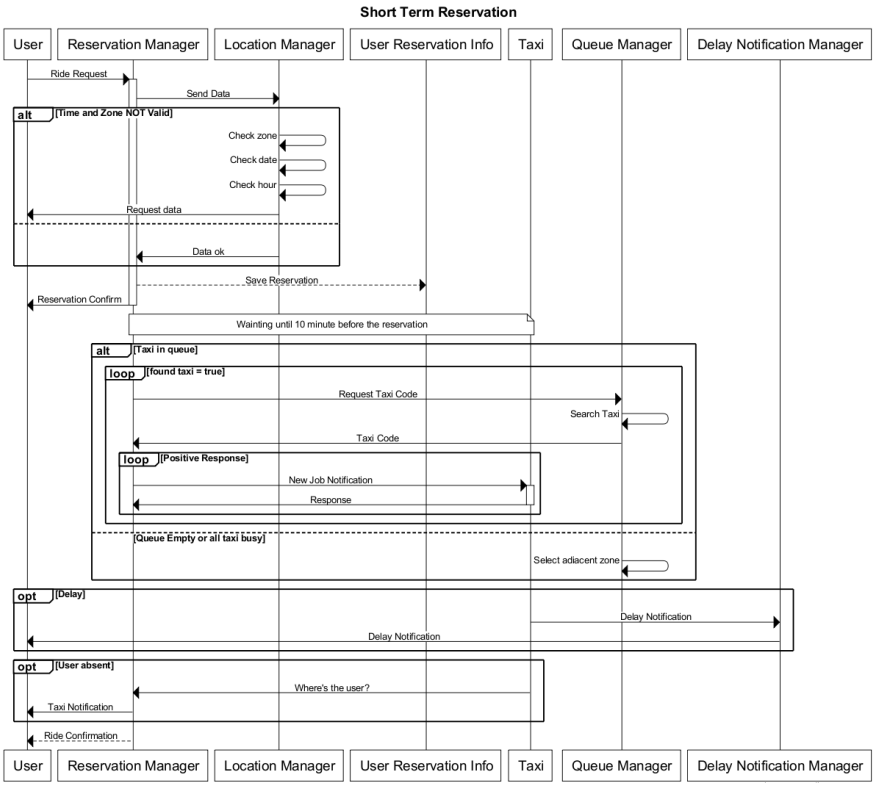
2.5.1 Taxi Registration



2.5.2 Short Term Reservation



2.5.3 Long Term Reservation



2.6 Component interfaces

Every part of the system provides a specific interface to interact with the other parts; in particular the system has:

- **Website Interface:** this interface is used by the clients in order to display the web pages provided by the web server.
- **Application Server Interfaces:** these interfaces are used by the web server to query the application server for the data that is needed in the web pages. Every manager in the application server has its own interface in order to keep, as much modular as possible, the application server in order to make it easier and to scale up it with more services.
- **Database Interface:** this interface, provided by the database, allows to query or modify the database by the application server. Since this is the only interface provided by the database, only the application server can access it, providing data security.

- **Application Programming Interfaces:** these interfaces allow the system to be upgraded and expanded with new functionalities.

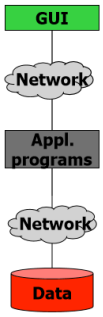
2.7 Selected architectural styles and patterns

The system is organized with a client-server architecture, but the application server is divided in several managers which manage every service.

The GUI part of the system is the website that allows the users and the taxis to access the system, through the internet network, using a browser or the mobile application.

To ensure the maximum possible scalability and expandability, the system is organized in a 3-tiered architecture. Using such architecture makes also the system more maintainable because every part is independent from the other and it can be substituted with no downtime.

Every part can also be replicated one or more times to increase availability of the entire system, guaranteeing more uptime.



2.8 Other design decisions

2.8.1 Design patterns

Taking into account the Controller part of this system, we have thought to use three design patterns:

- **Singleton:** it can be used with reference to the three tables of the DBMS, that are "Bookings History", "User Bookings Info" and "Taxi List".
- **Strategy:** it can be used with reference to the two different bookings, that are the short-term reservation and the long-term one.
- **Observer/Observable:** in this system, we can consider the "City" as the Observable and the Observers are all the classes, from the View

and the Controller, that must be notified about the changes of the Observable element.

3 Algorithm Design

3.1 Queue manager

```
function MANAGEQUEUE(zone)
    queue ← getZoneQueue(zone)
    while queueIsEmpty do
        adjacentZone ← getAdjacentZone(zone)
        queue ← getZoneQueue(adjacentZone)
    end while
    taxiCode ← getTaxi(queue)
    return taxiCode
end function
```

The above pseudo code describes how the Queue Manager organizes the queue of every zone. First of all, the function "manageQueue" will save in the variable "queue" the queue, which will be the return of the function "getZoneQueue". As long as the queue, saved in the variable with the same name, is empty, this function will check the queues of the adjacent zones and it will save in the usual variable the first queue not empty. In this way, the first taxi of the considered queue will be saved in the variable "taxiCode", that will be returned.

3.2 Reservation Manager

```
function MANAGEBOOKING(origin, time, destination)
    if !originIsValid(origin) or !timeIsValid(time) or (!originIsValid(origin)
and !timeIsValid(time)) then
        return error
    end if
    if time = CURRENT_TIME then
        booking ← createShortTermReservation(origin, destination)
        zone ← getZone(origin)
        while !positiveResponse do
            taxi ← manageQueue(zone)
            response ← rideNotification(taxi)
        end while
    else
        booking ← createLongTermReservation(origin, time, destination)
        saveBooking(booking, database)
        return
    end if
    sendConfirmation(booking)
    return
end function
```

The above pseudo code describes how the Booking Manager organizes the bookings. First of all, the function "manageBooking" checks if the origin address and/or the time, inserted by the user, is/are valid or not: in the worst case, there will occur an error. Otherwise, if the time is the current one, in the variable "booking", there will be saved a short-term reservation, which will be the return of the function "createShortTermReservation". In the variable "zone", this function will save the zone in which there is the origin address specified by the user. In the while cycle, that will occur as long as the taxi response will not be positive, thanks to the functions "manageQueue" and "rideNotification", a taxi will be chosen to do the ride.

If the time is not the current one, in the variable "booking", there will be saved a long-term reservation, which will be the return of the function "createLongTermReservation". Then, the information about the booking reserved will be saved in the database, through the function "saveBooking". Finally, a confirmation of the booking realized correctly will be send to the user.

4 User Interface Design

All interfaces are available in the RASD document from section **3.1.1** to section **3.1.3** included.

5 Requirements Traceability

The list below shows how the goals, defined in the RASD, are mapped into the design elements, that we have defined in the upper sections of this document.

5.1 *G1*

- Using the smartphone or the computer, the user clicks on the button, that lets him/her to book a ride.
- The web server loads the web page "booking.html".
- The user can view the booking interface and he/she can ask to a ride.
- Through the TCP/IP protocol, the web server sends the information, inserted by the user, to the application server.
- The application server, using the reservation manager and thanks to the TCP/IP protocol, executes an SQL query with the user's information to the DBMS.
- The DBMS saves these information in the specific table "User Bookings Info".

5.2 *G2*

- Using the smartphone or the computer, the user clicks on the button, that lets him/her to modify the date and/or the hour of his/her long-term reservation.
- The web server loads the web page "long reservation modification.html".
- The user can view the long-term reservation modification interface and he/she can ask to modify his/her ride.
- Through the TCP/IP protocol, the web server sends the information, inserted by the user, to the application server.

- The application server, using the reservation manager and thanks to the TCP/IP protocol, executes an SQL query with the user's information to the DBMS.
- The DBMS changes the old information with the new ones, in the specific table "User Bookings Info".

5.3 *G3*

- Using the smartphone or the computer, the user clicks on the button, that lets him/her to delete a long-term reservation.
- The web server loads the web page "long reservation deletion.html".
- The user can view the long-term reservation delete interface and he/she can ask to cancel his/her ride.
- Through the TCP/IP protocol, the web server sends the alphanumeric code, inserted by the user, to the application server.
- The application server, using the reservation manager and thanks to the TCP/IP protocol, executes an SQL query with the user's alphanumeric code to the DBMS.
- The DBMS deletes the long-term reservation information from the specific table "User Bookings Info".

5.4 *G4*

- Before to accept the user's request for a booking, the application system uses the location manager.
- The application server, in this way, thanks to the GPS information, can check if the origin address, inserted by the user, is valid or not.

5.5 *G5*

- The application server uses the location manager.
- In this way, the application server, thanks to the GPS information, identifies the current position of the taxi.

5.6 *G6*

- After the identification of the taxi position, the application server uses the queue manager to understand which is the queue relative to the zone where there is the taxi in that moment.
- The application server inserts the taxi identifier, that is the vehicle plate, in this zone queue, if the taxi is free.
- The application server deletes the taxi identifier, that is the vehicle plate, from this zone queue, if the taxi is not yet available.
- The application server assigns to the first taxi in the queue the request of a user, if he/she has requested a taxi in this zone.
- If the zone queue is empty, the application server, through the location manager, checks which four zones are adjacent of the requested zone. Then, the application server does the same things, already described previously.

5.7 *G7*

- Using the smartphone, the new taxi clicks on the button, that lets him/her to register in the system.
- The web server loads the web page "registration.html".
- The new taxi can view the registration interface.
- Through the TCP/IP protocol, the web server sends the information, inserted by the new taxi, to the application server.
- The application server, using the taxi registration manager and thanks to the TCP/IP protocol, executes an SQL query with the new taxi's information to the DBMS.
- The DBMS inserts these information in the specific table "Taxi List".

5.8 *G8*

- The application server, using the notification manager, sends an SMS, after the user changed his/her long-term reservation.

5.9 *G9*

- The application server uses the location manager and, thanks to the GPS information, it identifies the position of the user.
- The application server sends a notification to the first taxi available in the queue of the zone, from where the user has requested a taxi.

6 References

- Requirements Analysis and Specification Document from previous delivery;
- Design Document Table of Content;
- Slides "Software architectures and styles.pdf" of professor Tamburri;
- `http://www.uml-diagrams.org/component-diagrams.html` to understand how to create the component view diagram;
- `http://www.uml-diagrams.org/deployment-diagrams.html` to understand how to create the deployment view diagram.

7 Appendix

7.0.1 Software and Tools Used

- ShareLatex `http://www.sharelatex.com`, TeXstudio `http://www.texstudio.org/` to redact this document;
- Draw.io `http://www.draw.io` to draw the diagrams of component view and deployment view;
- Web Sequence Diagram `http://www.websequencediagrams.com/` to draw the sequence diagrams.

7.0.2 Hours of Work

In order to write this document, we have done the following hours of work:

- Paolo Paterna: 25 Hrs
- Lara Premi: 25 Hrs