



MyTaxi

Design Document

Authors:

Bucci Giovanni

De Togni Riccardo

Summary

1. Introduction	3
1.1. Purpose	3
1.2. Scope	3
1.3. Glossary	3
1.4. References.....	4
1.5. Document Structure	4
2. Architectural Design	5
2.1. Overview	5
2.2. High level components and their interaction	7
2.3. Component View	9
2.4. Component Diagram: Web Service	9
2.5. Component Diagram: User Management	12
2.6. Component Interfaces	13
2.7. Deployment View	15
2.8. Runtime View	17
2.9. Selected architectural styles and patterns.....	21
3. Algorithm Design	23
4. User Interface.....	25
4.1. Design Overview	25
4.2. User Interface and navigation flow	25
4.3. Further preview of the UI	27
4.4. User Experience	28
4.4.1. Taxi Call	29
4.4.2. Sign Up and Reserve a Taxi	29
4.5. Taxi Driver Login and Request Notification	30
5. Entities Architecture	32
5.1. E-R Diagram.....	32
5.2. Relational Model.....	33
6. Requirements Traceability.....	33

1. Introduction

1.1. Purpose

This document describes the specific architecture and design of “MyTaxi” project.

The focus will be on structural and design styles choices, expanding the thread already analysed in RASD document.

The design document in effect, starting from the requirements given in the RASD build a feasible architecture for the application.

1.2. Scope

The document will present different level views in order to describe clearly the architecture of the application. In particular will be presented the component view, both high and low level, the deployment view, the runtime view and a further description of user interface, analysed in its runtime flow.

1.3. Glossary

Below is reported the glossary already inserted in RASD:

- CUSTOMER: a generic person that use any part of the system service, it could be either a user or a guest.
- DBMS: Database Management System, the set of machines and specific operation that allow the right Database working.
- ADMIN/ADMINISTRATOR: it is a particular type of user that has administrative functions.
- GUEST: a person who has not signed up yet. Guests have no power until they sign up with one exception. If a Guest just want to call a taxi, it could simply insert its identification data.
- USER: a person that has already signed up as a customer. It could call a taxi, as guest does, but it also could reserve it in advance, compiling a specific form.
- TAXI DRIVER: a person who has signed up as a taxi driver. In order to complete its registration it has to provide its identification data and its driving license too.
- SYSTEM: the environment formed by the application itself and its features.
- CITY ZONE: each city is divided in zones. Every zone has approximatively the same territorial extension, so a city zone is one of the portions of the metropolitan area.

- QUEUE: an ordered list of taxi drivers that have previously provided their availability.
- CALL A TAXI: the action which can be performed both by guests and user, that consists in asking for a single taxi ride without any advance.
- RESERVE A TAXI: the action that could be performed only by Users. A user can forward the request for a taxi from a specified place to another in advance.
- SERVICE: the service that is provided by the application.
- DENY/DENIAL: when a request is not satisfied. It produce the shifting of the considered taxi driver to the bottom of the queue.
- ACCEPT: when a request, both coming from a reservation or a taxi call, is taken by a taxi driver who assumes the charge to bring passengers to the destination.
- UI: the user interface i.e. the set of web pages that constitute the meeting point for users and system.

1.4. References

- Requirements and Specification Document, RASD
- IEEE Standards for Information Technology Systems, Design Document

1.5. Document Structure

The Design Analysis is based on a Top-Down approach, therefore the Document structure will follow the same path. It will start from the high-level architecture, presenting the main components with their operations and mutual relations (2.2). Then it slides down to a lower level in which the high-level components are decomposed and analysed in detail (2.3).

After that in 2.4 paragraph will be presented the Deployment view of the system that show the execution architecture of the system representing the deployed software and hardware artifacts.

Runtime view presented in 2.5 paragraph will show the behaviour of the system during some typical situation. That will permit to understand in an easier way how the execution flow works.

The lowest level of analysis is reached in paragraphs 2.6 and 2.7 where are described the inner composition of the Interfaces and the Architectural choices which have been made to design the system.

Lastly, will be presented some of the main algorithms that are the fulcrum of the entire system. They will be described through pseudo-code.

Chapter 4 will take the User Interface already presented in RASD and give further information about the design choices. It will also describe some UI flows clearing up the navigation through different web pages or mobile screens. The paragraph about User Experience will get over the mere “Look Requirements” presented in RASD and it will deeply analyse the structure of the Web Application.

The last chapter presents the architectural choices about the Database structure. In particular will be presented the E-R diagram and the correspondent relational model.

2. Architectural Design

2.1. Overview

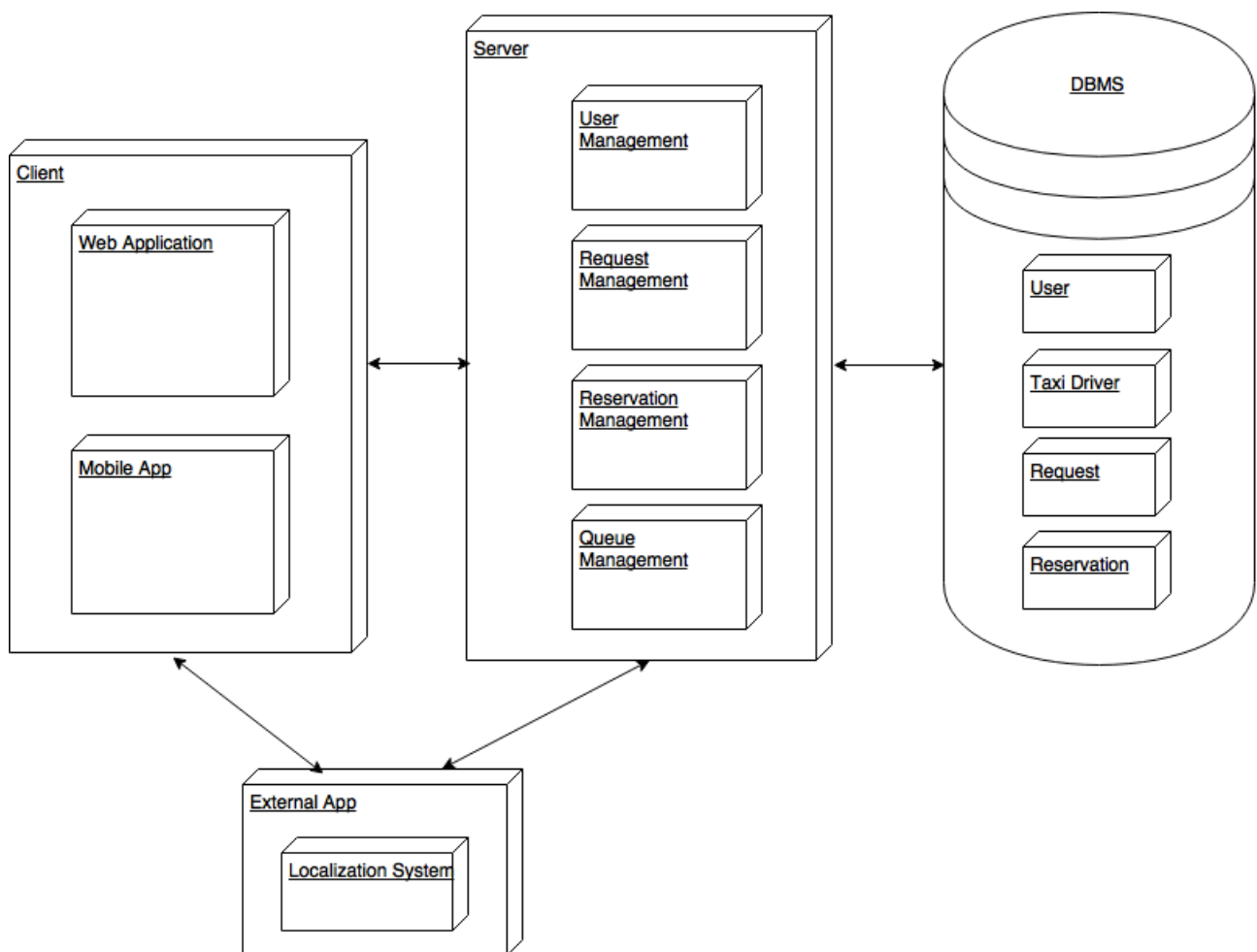
The design of the application is based on a 3-tier distributed system, where the three parts are Client-tier, Business Logic-tier and Entity-tier:

- **CLIENT-tier:** The different type of client application such as Web Application Client (Browser) or mobile app client composes client-tier. It collects the Users’ data and it is responsible of the delivery to the proper control unit that is part of the Business Logic. On the other hand, it is responsible for data receiving that consists in the safe delivery to the client, without loss or steal of data.
- **BUSINESS LOGIC-tier:** as mentioned above Business Logic is the core of the application, it provides the information about how the system objects are related and how they interact, it shape the message format and it also coordinate the intercommunication between clients and entities.
- **ENTITY-tier:** the entity tier contain the information about the system data model. It is responsible of database modification such as the insertion or the retrieving of any kind of data.

Given that architecture, it is possible to think that the Business Logics are represented by some Web Servers each one with a specific task. Entity-tier is represented by DBMS.

The situation described above is explained in the following diagram that provide a high-level view of the system structure:

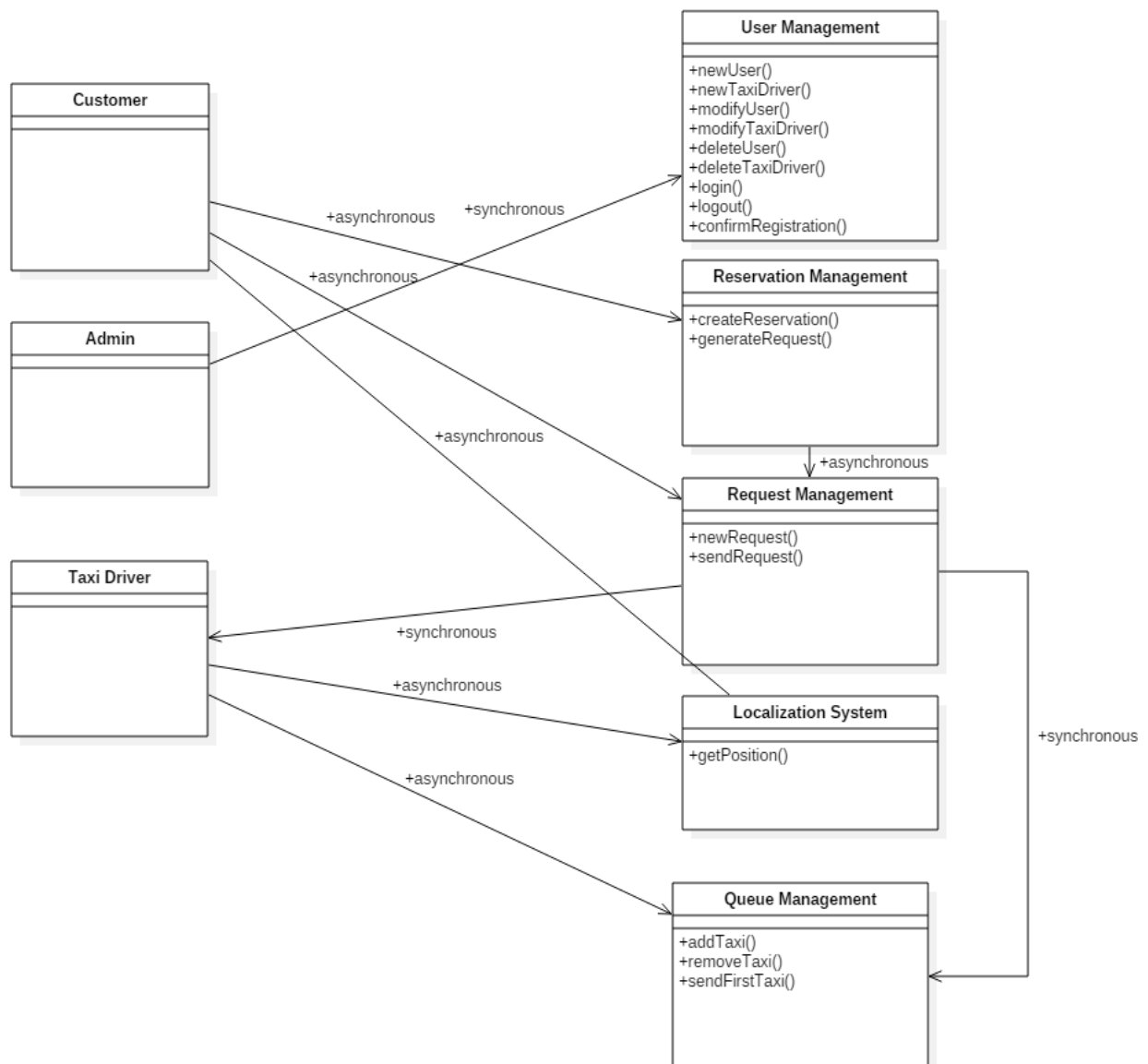
System Overview: 3-tier architecture



2.2. High level components and their interaction

In this paragraph the raw architecture presented above will be decomposed and analysed, focusing on which parts are related and which kind of communication is. The main components will be also provided with the operations that each one can perform.

High-Level Component view



Now every relation will be described in detail:

- **CUSTOMER – REQUEST MANAGEMENT:** a generic costumer may start at any time a communication with the Request Management component, through the provided interface (That part will be described in the next paragraph). The communication is asynchronous since the Request Manager is always listening for new requests and it can handle more than one communication at a time.
- **CUSTOMER – LOCALIZATION SYSTEM:** a customer start an asynchronous communication with the Localization System when it accesses the application.
- **CUSTOMER – RESERVATION MANAGEMENT:** the reservation management system is always ready to accept a new reservation from any customer. To allow that the communication is asynchronous.
- **ADMIN – USER MANAGEMENT:** an administrator can communicate with the User Management Component in order to view, modify or delete any registered user. Since the system permits multiple administrator the communication must be synchronous to grant concurrency.
- **TAXI DRIVER – LOCALIZATION SYSTEM:** Taxi driver implicitly start a communication with Localization System when it provide its Availability. Since the Localization System has to handle multiple communications the message exchange is asynchronous.
- **TAXI DRIVER – QUEUE MANAGEMENT:** As the communication with Localization System this relation starts when the taxi driver provide its availability. Queue Management handles all the taxi drivers so it can accept many request at the same time: the communication is asynchronous.
- **REQUEST MANAGEMENT – QUEUE MANAGEMENT – TAXI DRIVER:** this relation is ternary because it represent a whole communication cycle. The request management has to associate its pending request with a taxi driver who will bring passengers to the destination. In order to find a feasible taxi driver the request management “ask” the queue management who is the first taxi driver in the specific queue, identified by the

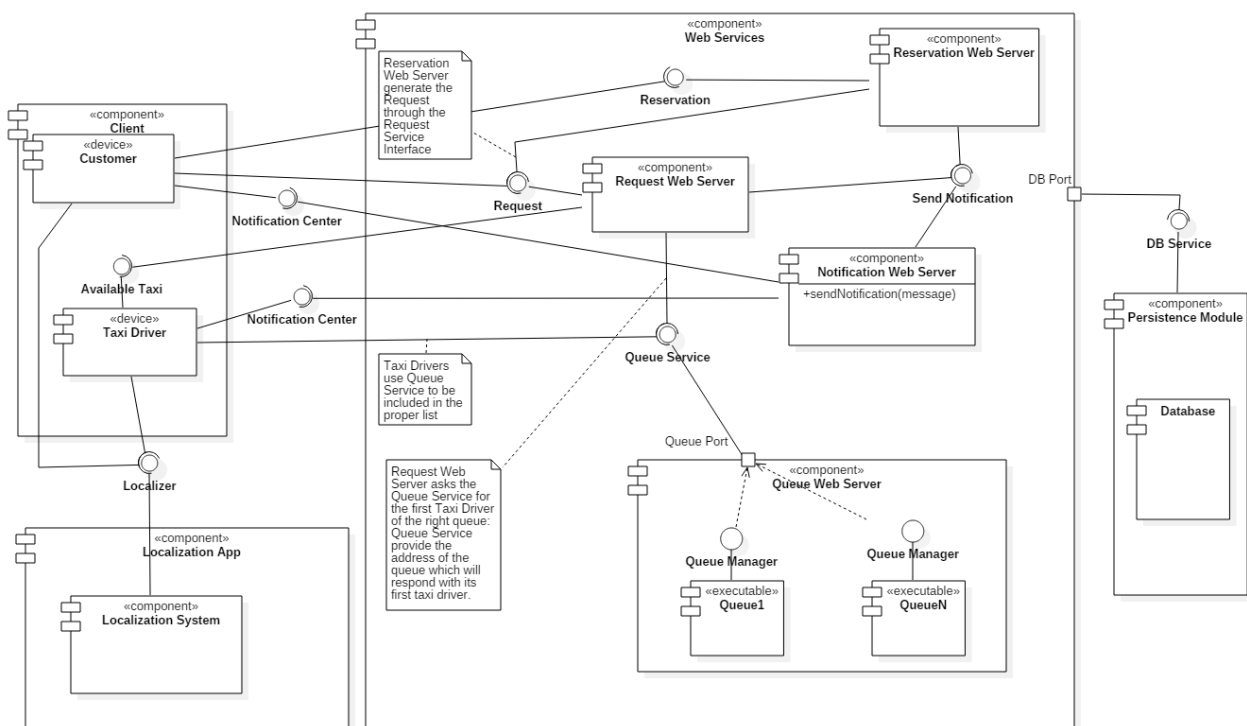
start address of the request. Once the request management receives the taxi driver information, it directly start a communication with Taxi Driver to send it the request. This loops until a taxi driver accepts the request. The communication between Request management and Queue management is synchronous in order to avoid simultaneous requests (Queue management respond a request with the same taxi driver). For similar reasons also the communication between Taxi Driver and Request Management is synchronous. In fact, it will be granted that the request management do not ask for another taxi driver until the former has denied the request.

2.3. Component View

Now the High-Level Components view presented above will be decomposed and analysed in detail. In particular will be highlighted the subcomponents and how they behave. A component behaviour is described by provided and required interface, and its relation with other components.

2.4. Component Diagram: Web Service

Component Diagram: Web Services



The Component Diagram is divided in four macro components:

- Web Services
- Client
- Localization App
- Persistence Module

Web Services represent the main component. It contains many sub-components, one for each service that the system can provide. Even if it has been used the term “Server”, the sub-components are not independent hardware entities. They are just executable pieces of software contained in the same environment.

- The Request Web Server handle the Request Management already described in the High Level Component View (2.2). It provide a Request Interface available for Clients and for an inner component, the Reservation Web Server, which will be presented soon. It has to be able to communicate with the Queue Manager, from that comes its required interface Queue Service. The use of this interface is explained in a note inside the diagram and it will be deeply analysed in the next paragraph. Further when a request it is accepted by a taxi driver the Request Web Server has to send a notification to the User and to the taxi driver itself. That explain the second required interface: Send Notification.
- The Reservation Web Server handle the Reservation Management, it interact with the Client component through its provided interface, Reservation, and with the Notification Web Server through the required interface Send Notification. It also communicate with the Request Web Server via Request interface, since the Reservation Web Server has to generate a request ten minutes before the related reservation time.
- Queue Web Service hides its inner composition to the rest of the Web Servers since the only way to communicate is to use its provided interface Queue Service. It contains one software component for each queue, so for each city zone (see Glossary). When the Request Web Server ask for the first taxi driver of a particular queue the Queue Web Server transparently redirect the request to the right Queue. Each Queue provide a

Queue Manager Interface that can respond the Request Web Server with the required data just sending it to the Queue Service interface through the Queue Web Sever gateway.

- Notification Web Server handles all the notification functionality. It provide a Send Notification interface through which the other components may communicate their intention to send a notification. Both Request and Reservation Web Servers use this interface. To forward the notification to the clients the Notification Web Server uses the Notification Centre interface provided by both customers and Taxi Drivers.

The second Macro Component represent all the kind of client that can access the service, so it integrate Customer component and Taxi Driver component.

- Customer symbolize a generic utilizer, which may be both a Guest and a User. It requires two different interfaces: one to make a request (Request Interface) and the other one to make a reservation (Reservation Interface).
When it make a Request or a Reservation using the mobile app its position is automatically calculated by a Localization System, asked by the customer through its Interface.
- Taxi Driver component embodies the client-side service provider. The Request Web Server can communicate with it through the Available Taxi Interface. A taxi driver can give its availability to the Queue Web Server through its Queue Service Interface. In order to permit the Queue Web Server to redirect the Taxi Driver to the right queue, jointly with its availability, it has to provide its position too. For this reason, it transparently (without any alert) connects with a Localization System through the specific interface, which will provide the position.

The third macro component is external to the application since it represent a Localization App. Its internal composition is briefly schematized here in order to explain how it is related to the other system components.

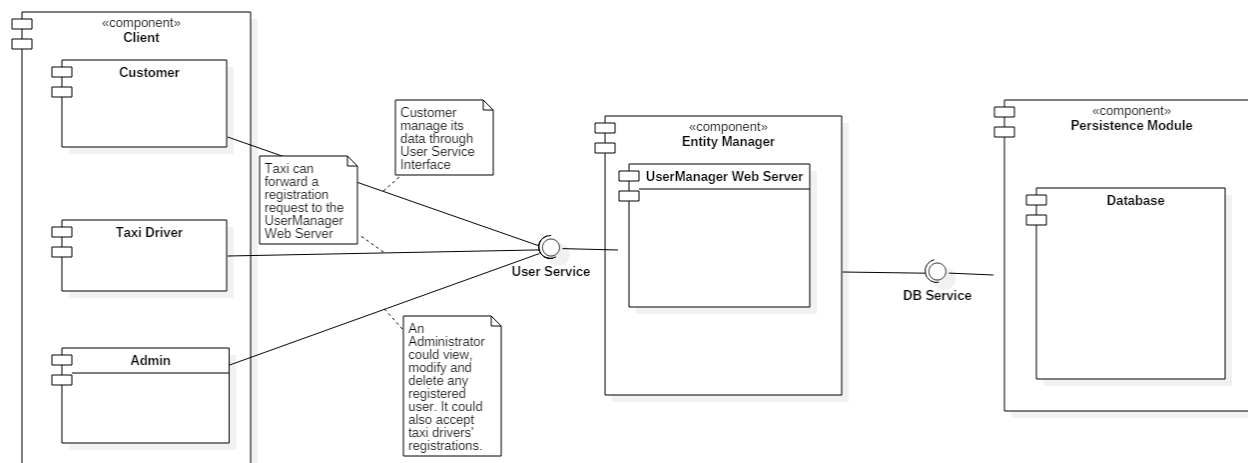
- Localization System is the only subcomponent. It contains all the functionality to calculate the position of a device given its GPS data. This component communicates through a provided interface Localizer.

The fourth component is the responsible for the data storage. All the Web Services are linked to it through its interface DB Service. This interface provides methods that allow to operate over the database. Persistence module and its subcomponent Database form the Database Management System (DBMS) presented in the Overview (2.1).

2.5. Component Diagram: User Management

Another important view concerns what it has already been called “User Management” in the High-Level view. The Client-side components are the same presented above with the addition of Admin, the administrator. Server Side there is only a component: Entity Manager. Here there is a graphical view:

Component Diagram: User Management



The Entity Manager contains a sub-component, User Manager Web Server, which performs the functions of the User Management component. It provides User Service Interface through which all the clients can communicate. The component in the middle receives the requests from the clients and forward them to the Database through the DB Service. The specific use is partially described in the diagrams notes but it will also be analysed in the next paragraph.

2.6. Component Interfaces

Every component needs to interact with other elements of the system. The communication between components is managed using interfaces. Every interface takes a fixed number of arguments, and returning the desired result, so that components can avoid knowing how every operation is implemented, and so reducing coupling.

Here is presented a brief description of every interface presented in the component view, in order to perfectly explain the interaction between components.

▪ Localizer API

- +getPosition (Bin signal : String coordinates)

It is the interface used to calculate the client position using an external GPS service. It receives the signal and provides the coordinates for the client.

▪ Request

- +createRequest (String name, String surname, String email, String phone_number, String position, String address, Int passengers : Int taxi_code, Time expected_time)

A user to ask for a taxi ride uses it. It takes essential data as an input, and after all the procedure to find the taxi, returns the assigned taxi code and an estimated waiting time.

- +viewRequest()

▪ Reservation

- +createReservation (String username, String start_point, String end_point, Int passengers, Date reservation_date, Time reservation_time : Bool confirmation)

This interface allows the user to book a taxi ride in advance; in order to do so the interface only needs the client username (to retrieve user details) and ride details such as start and end addresses, number of passengers and date and time of the ride. The request will be created a

couple of minutes before the indicated time, so the only thing returned to the user is the outcome (positive or negative) of the reservation.

▪ **Available Taxi**

- +Accept (String request_ID, String taxi_code : Bool response)

This interface represent the attempt to assign a taxi to a specific request: it takes the request id to identify univocally the request, and sends the request to the taxi; it then sends back to the system the taxi driver answer.

▪ **Queue Service**

- +insertTaxi (String taxi_code, String position : String zone)

The target of this method is to insert an available taxi in a queue, so it receives the taxi code and its position, sending back the name of the zone in which the taxi is positioned.

- +getTaxi(String request_start_address : String taxi_code)

The request server ask the queue server a possible taxi to forward a specific request, sending the start address indicated in the request. The queue server detects the associated zone queue and query the queue manager for a taxi; as soon as it receives the response, it forwards the taxi code to the request server

▪ **Queue Manager**

- +getTaxiFromQueue(int queue_index : String taxi_code)

The queue manager receives the queue index and so requires the first taxi in the indicated queue. When the taxi is found, its code is sent to the queue manager, ready to be sent to the waiting request

▪ **Send Notification**

- +sendEmail(String email, String event_type, String event_id : void)
- +sendSMS(String phone_number, String event_type, String event_id : void)

- +sendNotification(String username, String event_type, String event_id : void)

This set of methods is created in order to delegate notifications at the specific component. The system sends the data related to the event that has to be notified, and the target media for the notification (email, mobile phone or mobile application). The notification server then provides to send the proper notification to the user.

▪ DB Service

- +insertRequest(Request req : void)
- +insertUser(User usr : void)
- +insertTaxi(Taxi taxi_driver, Date Timestamp : void)
- +insertReservation(Reservation reserv : void)
- +insertAdmin(Admin admin : void)
- +modifyUser(User newusr : void)
- +modifyTaxi(Taxi newtaxi : void)

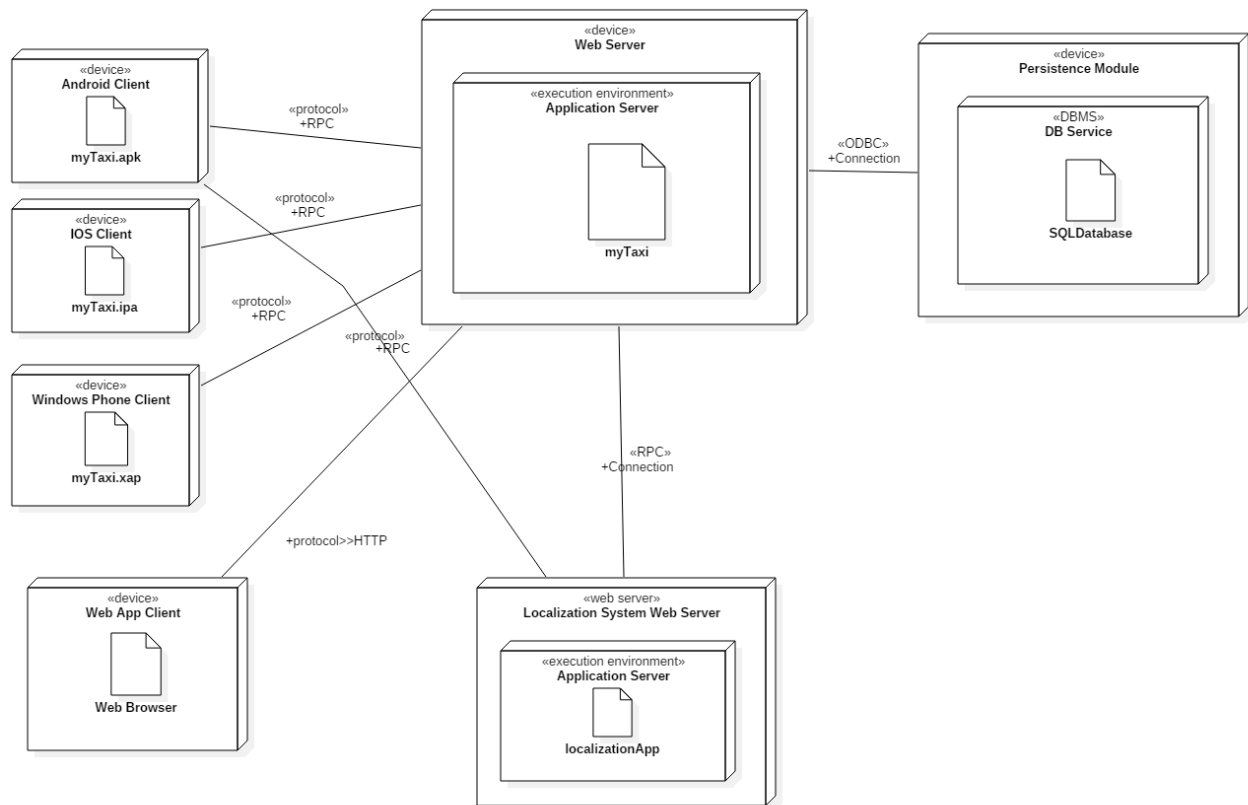
The set of methods defined by this interface is meant to define how entities are inserted or modified in the databases. They take the entities classes, with every detail related, that will be decomposed in basic type pieces of information and inserted in the database.

2.7. Deployment View

This section highlights the real deployment of the different components deepening the presentation already made in the previous paragraph. The focus will be on the type of communication, in terms of real connection and not of logical connection as it was in the case of high-level components. The interfaces that link the various component are omitted to clarify the presentation, since they were already described above. Each group of components is thought as a device, in this way the different types of client embodies a device and also the different services contained in the Web Server. For the first time since the discussion appears the database, here analysed from the point of view of the deployment.

Here the graphical representation:

Deployment Diagram



Since this diagram deepens the analysis of the components, the client side of the deployment diagram presents the differentiation between the mobile or web based device. The three different mobile devices communicate with the Web Server through Remote Procedure Call (RPC in the picture). This kind of communication allow running the desired procedures into the server whenever a client need a service to be performed. Localization system uses the same connection with both Clients and the Web Server. The Web App Client communicate according to the web standards, via HTTP. On the Server side we have a device called Web Server, which contains many services, runnable softwares that share the same processing machine but independent in terms of execution. The execution environment though is shared. It contains the executable application which is reported here without any extension in order to let freedom of implementation.

The device Persistence Module is the responsible for all the operation about stored data. It contains a single subcomponent responsible for DBMS: DBService. This component has to be protected against unauthorized accesses and unintended operations. For this reason is not directly connected with the Web Server but it has a particular interface defined by a “façade”

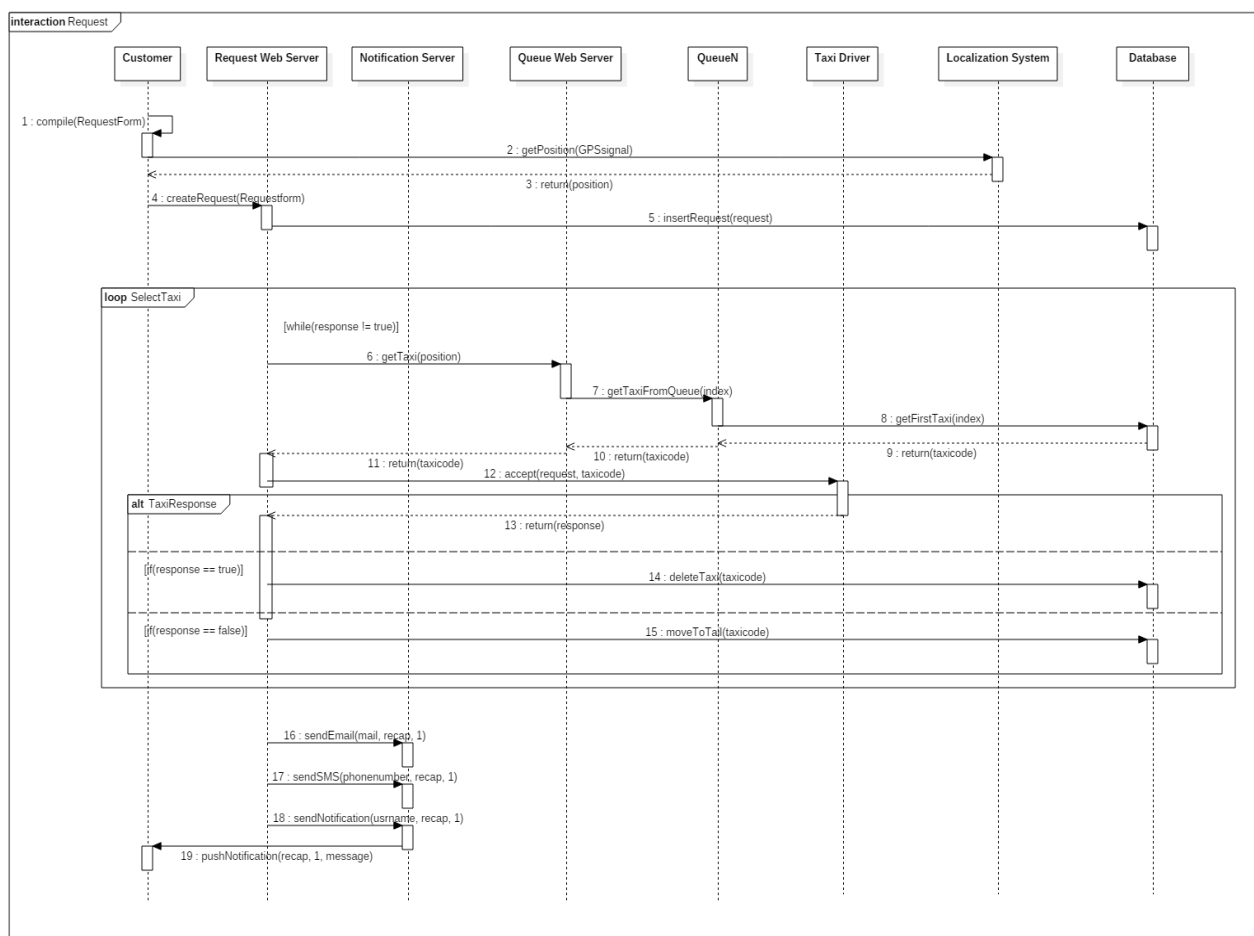
pattern, deeply described in the paragraph 2.9. The connection is based on ODBC, Open Database Connectivity, the driver for the communication with the database.

2.8. Runtime View

This paragraph focuses on the system behaviour during the most important activity such as a Taxi Call or a Reservation. A sequence diagram describes each of them showing which procedures are called and the messages exchange between the components. The components as explained above, communicate through the interfaces presented graphically in the Component Diagram and analytically in the paragraph 2.6. Given that, all the procedures shown in the Sequence Diagrams are methods of specific Interfaces.

The first Sequence Diagram represent the situation when a customer forwards the request for a taxi:

Runtime Behaviour: Request



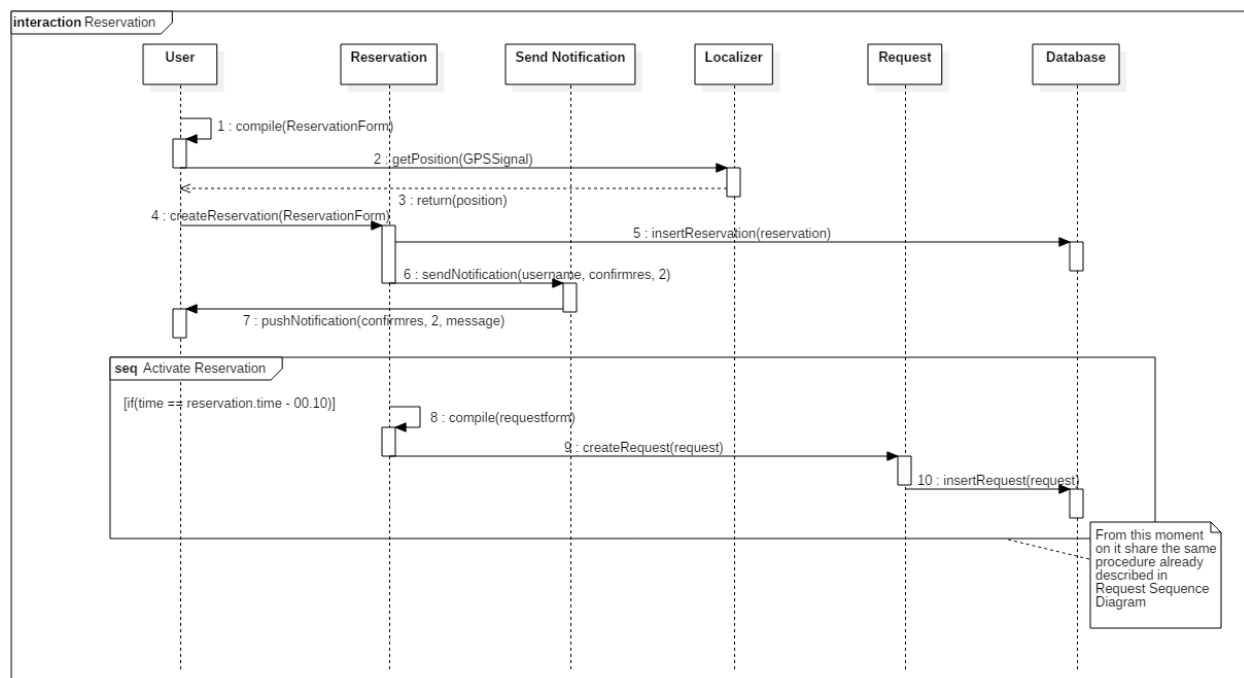
The diagram is divided in three parts delimited by the Loop Frame. In the first part the generic customer compiles the request form and forward it to the server, specifically to the Request Web Server, which analyses it and sends to the Database the insert command.

After that, in the second part, the Request Web Server search for an available taxi that will carry out the request. In order to do that it asks the Queue Web Server the first taxi. The Queue Web Server redirect the request to the right Queue, identified by the position, which is one of its internal components. The Queue communicate directly with the Database where it extracts the first taxi driver in the queue relation (the one with the oldest timestamp) and it sends its taxicode to the Request Web Server. Now the Request Web Server asks the Taxi Driver to confirm its availability. If it denies the procedure restarts and the loop goes on until a taxi accepts the request.

After that the Request Web Server communicate the success to the customer through the Notification Server which sends an email, an SMS and a push Notification.

The second view represents the Reservation i.e. the action that a registered user has to do in order to book a taxi in advance. Here the sequence diagram:

Runtime Behaviour: Reservation



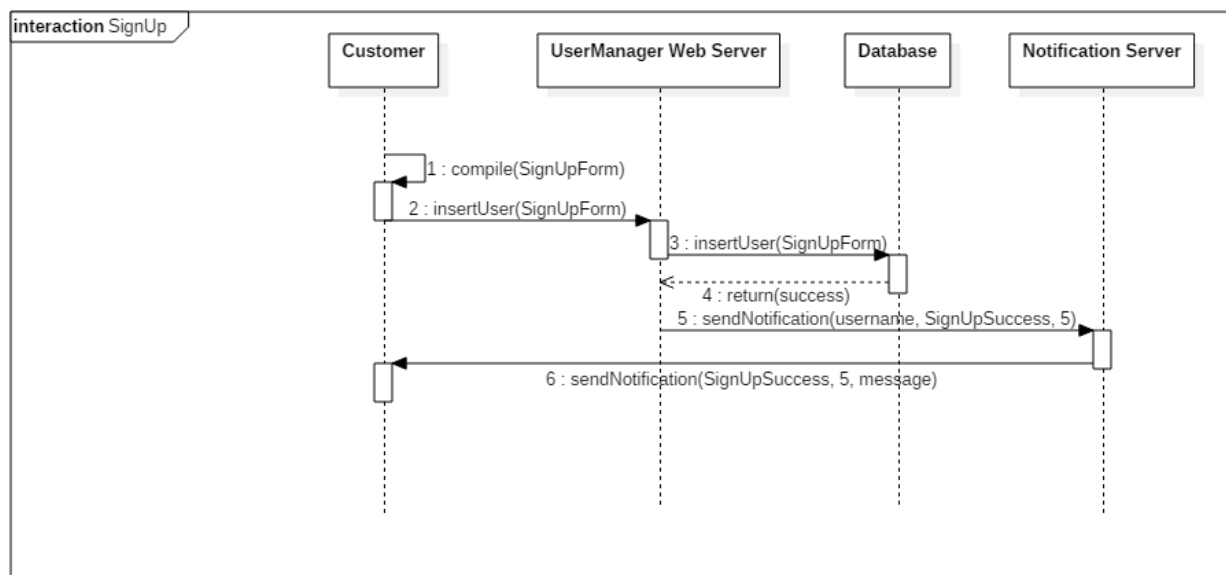
The user compiles the reservation form, filling in every necessary information, and right after that he requests his position to the localization service. With all these data, a reservation is created.

The reservation is inserted in the database, and if the procedure ends correctly, it will ask the notification center to send a message to the user with the result of the operation.

After that, ten minutes before the reservation time, the system activates the request procedure, generating a request with the reservation data. The request will be inserted in the database and managed as a standard request.

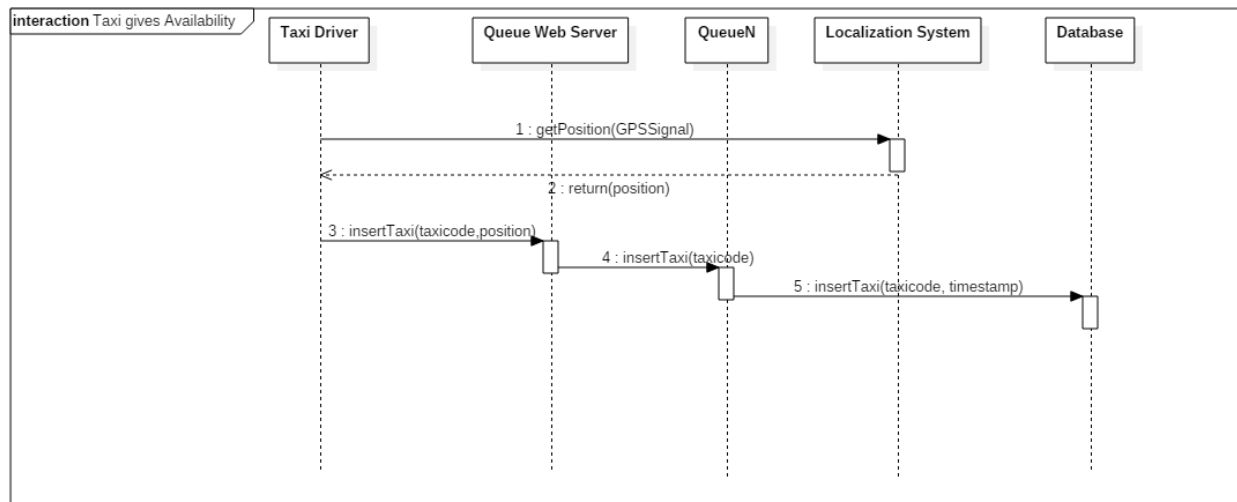
Another action already described in RASD is the Sign Up of a new User:

Runtime Behaviour: Sign Up



The last situation describes the insertion of a taxi driver in the queue. It is verified when it provide its availability:

Runtime Behaviour: Taxi Driver gives Availability



A taxi driver ready to work announces his availability using the dedicated button. This triggers a request to the localization system that sends him his position. The taxi data and the position are then sent to the queue web server with an insertion request. The server then inserts the taxi in the last position of the right queue. Every queue manager is in charge to save the taxi data and position in the queue in the database; the taxi driver is now ready to be called when his turn comes.

2.9. Selected architectural styles and patterns

This application is structured as a classical client-server system, and more specifically this is the case of three-tier architecture (client, server and database) composed by a thin client (the group of pages of the website, they only need to send information, no operations are executed on the client side) and a fat server (the whole management system and every operation are executed inside the server). Following the basic principles of software engineering, the ideas behind the project are the “divide et impera” concept and the code reuse. Every macro component exchanges information with others using RPCs (Remote Procedure Calls) following the Single Responsibility Principle: every component is in charge of the operations related to a single task, asking other components for data not related to his specific role. Every macro component is then divided in many sub-components, in order to create atomic blocks of code ready to be used for different purposes, reducing work and complexity; every sub-component is linked to others by internal sockets.

During the application design, it was necessary to decide some specific architecture constraints in order to make everything work as planned. In this process design patterns are very useful, because they are structures that represent ready solutions to common problems.

In this document the choice is to present every solution adopted grouped by pattern type:

- **Thread Pool**

In every service in which is possible to make multiple concurrent request it is necessary to insert a Thread Pool pattern in order to manage the requests, and to maximize the occupation of every service “worker” to obtain the best performance.

In this particular project this kind of pattern is necessary to manage the multiple taxi requests or reservations from users, the procedure of taxi assignement preformed by the queue manager, and data access and modification by multiple users.

- **Façade**

The Façade pattern is used to hide the details of complex blocks of components with different interfaces in order to simplify the communication between these blocks and the rest of the architecture.

In the project this pattern is used to allow the communication between the application and the database, solving possible compatibility problems, to simplify the taxi assignment hiding the division of queues, and to allow the use of localization system without knowing implementation details.

- **Observer**

This pattern is the best way to trace the changes of a particular component; it is composed by an object called “subject” related to an object called “observer”: every change in the subject state is recorded by the observer, often triggering actions.

In this case the Observer is used by the notification component that, as an observer, is ready to listen to every event generated by requests, and send proper notifications to the involved users.

- **Read-Write Lock**

Another useful pattern to manage concurrency: it is used in the database to control multiple access on data and preventing errors due to data corruption or simultaneous operations on the same record.

3. Algorithm Design

In the project the majority of the operations are simple functions, the only complexity is given by communication between them. However, there are a few algorithms that are worth mentioning besides the classic sort and search processes:

- **Queue Management**

The queue system is based on a FIFO rule: every taxi is inserted in the queue in availability order; the first to give availability is the first in the queue. For every request the system asks the first taxi in the queue: if the taxi driver accepts, he serves the customer and is removed from the queue; otherwise if he refuses or does not answer, he is put in the last position of the queue after a fixed time.

Pseudo-code:

```
* received request *

while (!accept){
    taxi_code=getFirstTaxi();
    accept=sendRequest(requestId, taxi_code);
    while(!accept && timeout!=0){
        wait;
    }
    if (accept){
        return 0;
    }
    else taxiNotReady(taxi_code);
}
return -1;
```

- **Expected Time Computation**

This procedure is thought to show a user an estimated time to wait before the arrival of the requested taxi.

It asks the GPS service to calculate the distance between taxi and user, and using this data make a valuation of the remaining time.

Pseudo-code:

```
distance=calculateDistance(taxi_pos, user_pos);  
expected_time=distance/average_speed + costants;  
return expected_time;
```

- **Expected Fare**

In view of a possible future implementation, it is possible to think of an algorithm that gives the user a possible fare for the current ride. It obtains, using GPS, the actual distance between start and end points, calculates an estimated travel time and by retrieving the average cost of fuel and considering some additional factors like traffic or service it provides a plausible fare (subject to changes) to give an indication to the passenger.

Pseudo-code:

```
distance=calculateDistance(taxi_pos, user_pos);  
expected_fare=(distance/avg_taxi_consume) x avg_fuel_cost + costants;  
return expected_fare;
```


4. User Interface

4.1. Design Overview

The idea, always present within the project, is to create a tool that is easy and immediate for the user, and the user interface is designed to meet these requirements.

From the mock-ups already presented in the RASD, it is in fact clear that the aim is to present the users a neat and minimal interface, in order to make the procedures intuitive and as quick as possible.

4.2. User Interface and navigation flow

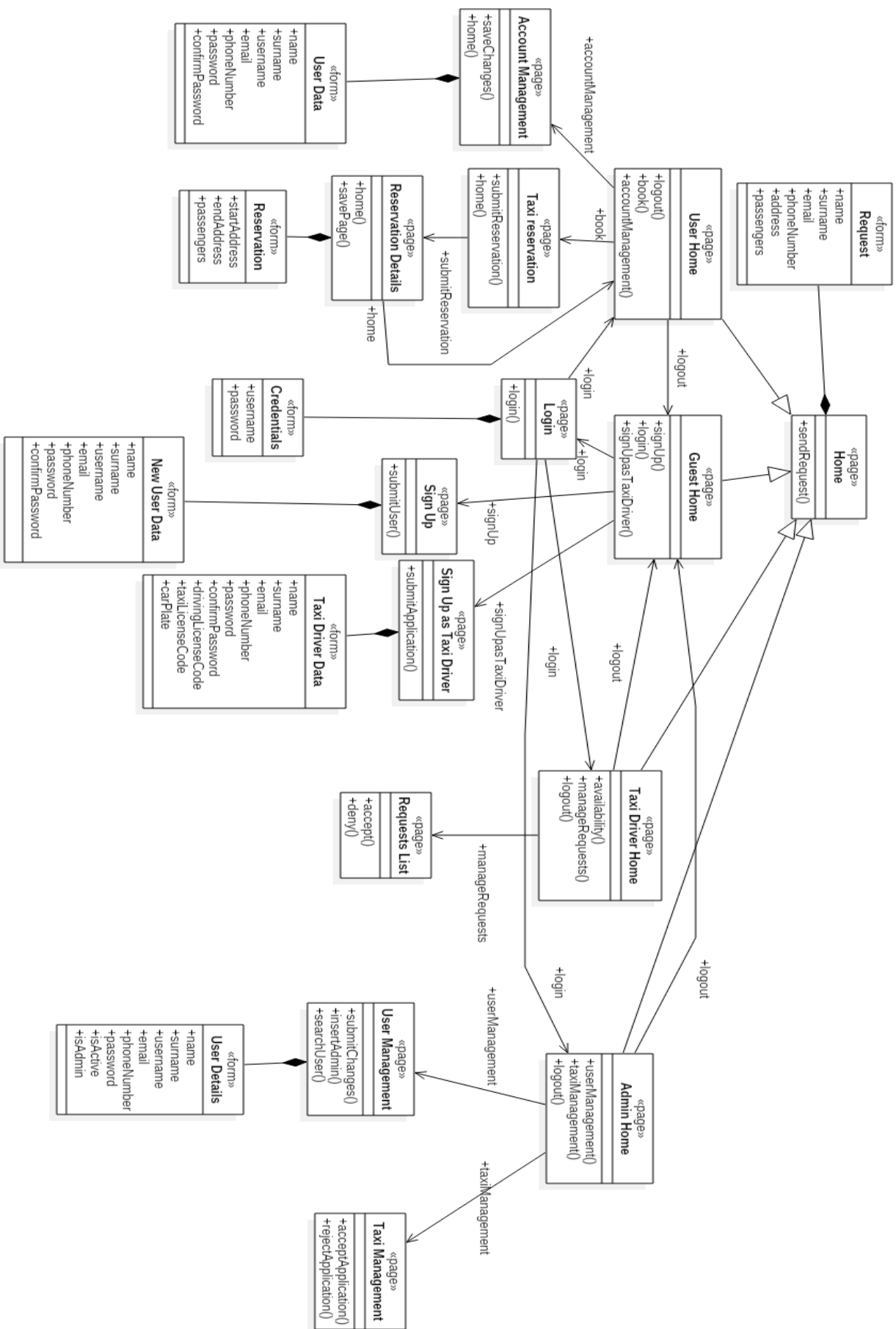
Here it is presented a diagram that, with the proper stereotypes ("page" and "form"), shows how pages are related, which important components are present, every input form, and how the navigation through the website is structured.

The graph is presented as a class diagram, and the symbols have the same meaning as if they were used in that kind of diagrams; it is important to notice that <<page>> means that the class represents a web page, and <<form>> identifies an input form contained in a specific page.

The home page structure is the same for all the users, focused on the fast request for a taxi; every user home page is then developed from this point, adding links and features associated to the relative user.

Following these links, every customer can navigate through the pages, but only through those for which he has permission.

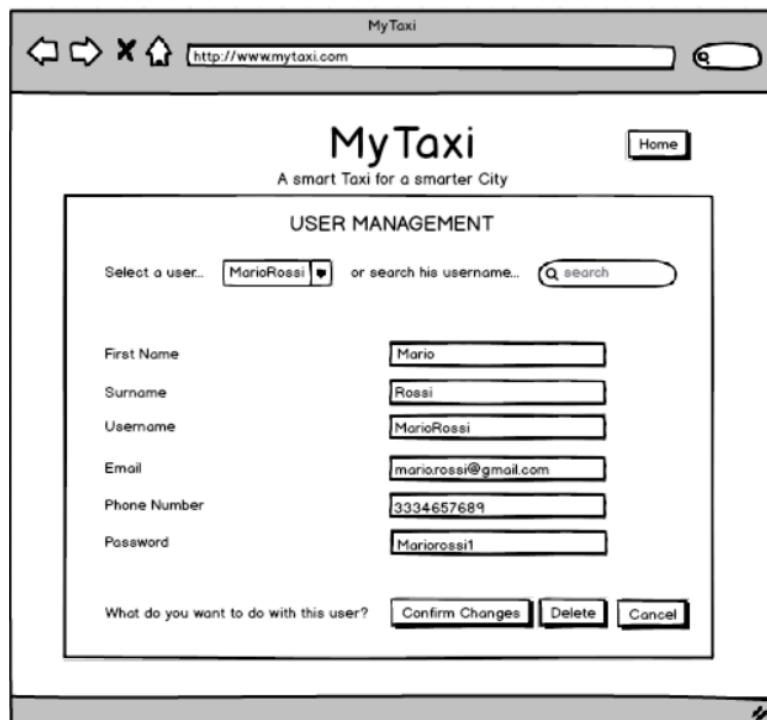
Here are presented only direct flows, associations representing cancellations or links to previous or home pages are omitted in order to simplify the reading of the graph.



4.3. Further preview of the UI

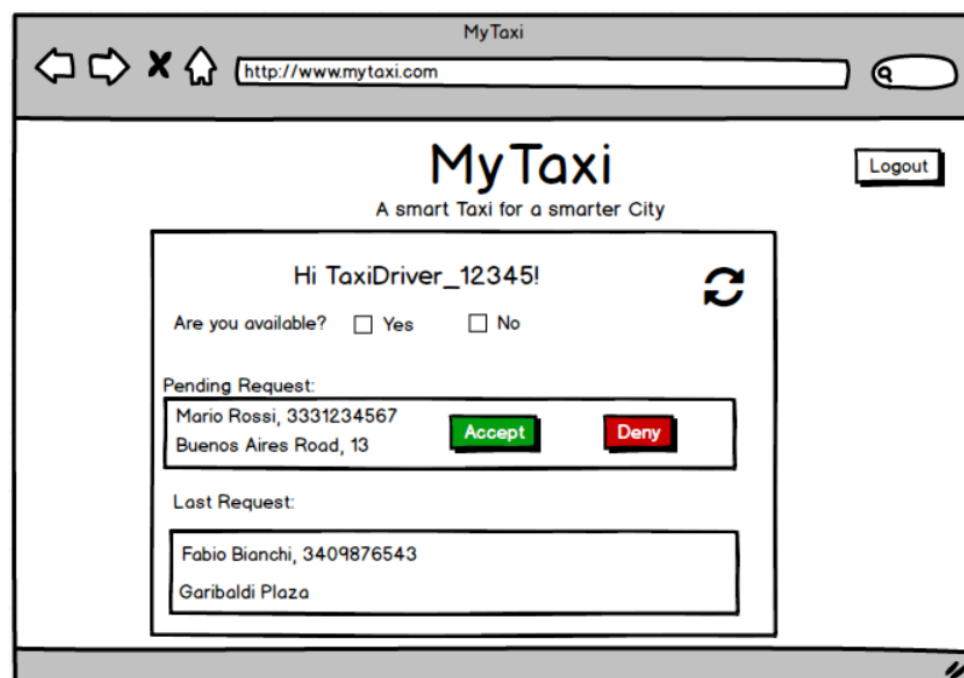
In order to clarify the style of the application, here are presented some of the pages not already contained in the RASD, such as the taxi driver's home page, and the pages used by the administrator to manage users and taxi applications.

User Management



The screenshot shows a web browser window titled "MyTaxi" with the URL "http://www.mytaxi.com". The page header includes the "MyTaxi" logo, the tagline "A smart Taxi for a smarter City", and a "Home" button. The main content area is titled "USER MANAGEMENT". It features a dropdown menu for "Select a user.." with "MarioRossi" selected, and a search bar labeled "or search his username..." with a "search" button. Below this, there are input fields for "First Name" (Mario), "Surname" (Rossi), "Username" (MarioRossi), "Email" (mario.rossi@gmail.com), "Phone Number" (3334657689), and "Password" (Mariorossi1). At the bottom, there are three buttons: "Confirm Changes", "Delete", and "Cancel".

Taxi Home Page



The screenshot shows a web browser window titled "MyTaxi" with the URL "http://www.mytaxi.com". The page header includes the "MyTaxi" logo, the tagline "A smart Taxi for a smarter City", and a "Logout" button. The main content area is titled "Hi TaxiDriver_12345!". It features a "Are you available?" section with "Yes" and "No" radio buttons. Below this, there is a "Pending Request:" section with a box containing "Mario Rossi, 3331234567" and "Buenos Aires Road, 13", and two buttons: "Accept" (green) and "Deny" (red). At the bottom, there is a "Last Request:" section with a box containing "Fabio Bianchi, 3409876543" and "Garibaldi Plaza".

The screenshot shows a web browser window with the address bar displaying "http://www.mytaxi.com". The page title is "MyTaxi" with the tagline "A smart Taxi for a smarter City". A "Home" button is located in the top right corner. The main content area features a "SIGN UP" form with the following fields and values:

Field	Value
First Name	Mario
Surname	Rossi
Username	TaxiDriver1
Email	mario.rossi@gmail.com
Phone Number	3334567890
Password	Mariorossi1
Driving License	M11234567G
Taxi License	12345

At the bottom of the form are two buttons: "Accept" and "Reject".

4.4. User Experience

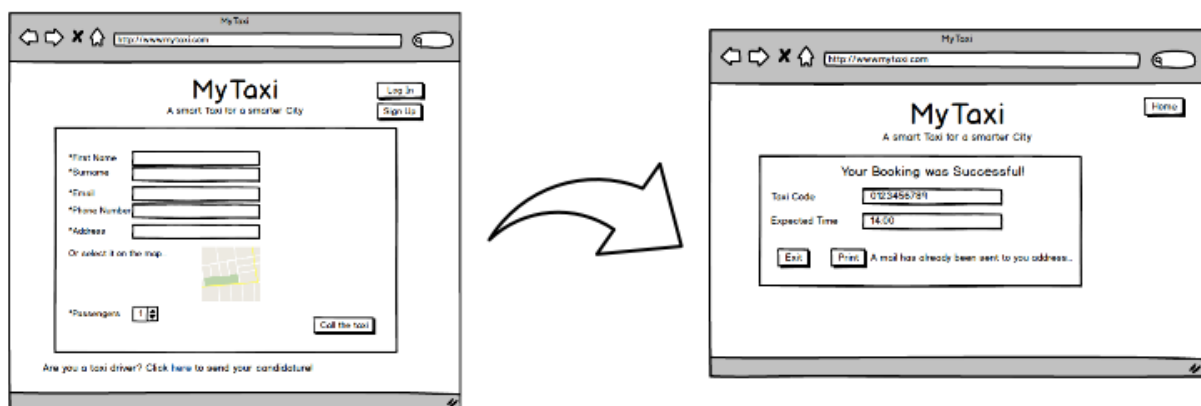
In order to make the service as immediate as possible, the home page of the application is composed of the module to send a rapid request or a taxi, inserting the indispensable data like name and phone number. On the side there are buttons to sign up and login. After the login every kind of user has a personal home page: the registered user has the options to book a taxi in advance and modify his data; taxi drivers can give/remove availability and accept requests; administrators have access to users data and to taxi driver applications.

Every option is fully described in the RASD section related to Use Cases, and the navigation uses only buttons to navigate through the pages, and forms to insert data: in this way the user can understand how to use the application at the first use.

Here are shown some page flows based on the mockups already presented, in order to fully describe how pages are related, and in which way the navigation is structured

4.4.1. Taxi Call

A guest already finds on the home page the form to call a taxi, when he has filled in the fields, he waits for the taxi, and when the call is accepted, the guest is brought on the confirmation page, with the call details.



4.4.2. Sign Up and Reserve a Taxi

If a guest needs to make a reservation, from the home page he has to click on the Sign In button: in the following page he will find the registration form; once he submit the data, he will be redirected on the home page, but this time as a user, and thus allowed to make a reservation. He finds a form very similar to the simple call one, that allows him to book a taxi. After the process a confirmation page will be shown, where the user can read a resume and save the page, or go back to the home page.

MyTaxi
A smart Taxi for a smarter City

Log in Sign Up

*First Name
*Surname
*Email
*Phone Number
*Address
Or select it on the map

*Passengers 1

Call the taxi

Are you a taxi driver? Click here to send your candidature!



MyTaxi
A smart Taxi for a smarter City

Home

SIGN UP

*First Name
*Surname
*Username
*Email
*Phone Number
*Password
*Confirm password

Marked elements to have to be filled

Register



MyTaxi
A smart Taxi for a smarter City

Home

Your Reservation was Successful!

A mail has already been sent to you address.

You will receive a message about 10 minutes before the meeting time with your Taxi Code

Exit



MyTaxi
A smart Taxi for a smarter City

Currently Logged as User123

Home EasyCall

Taxi Reservation

*Starting Point
Or select it on the map

*Arrival Point
Or select it on the map

*Passengers 1

*Hour 8:00

*Date 1/1

Book the taxi


Marked elements to have to be filled

4.5. Taxi Driver Login and Request Notification

A taxi driver is ready to work, so on the home page he clicks on the Login button. In the following page he inserts his username and password, and he will be redirected to the home page dedicated to taxi. This page is comprehensive of everything he needs: in fact here he can see his state and his rides. In a dedicated box he will find new requests, ready to be accepted or rejected just by clicking on the specific button.

MyTaxi
A smart Taxi for a smarter City

[Log In](#)
[Sign Up](#)

*First Name:
*Surname:
*Email:
*Phone Number:
*Address:
Or select it on the map: 
*Passengers:
[Call the taxi](#)

Are you a taxi driver? [Click here](#) to send your candidature!



MyTaxi
A smart Taxi for a smarter City

Welcome back!

Username:
Password:
☐ Accept the [Terms of Use](#) and the [Privacy Policy](#)
[Login](#)



MyTaxi
A smart Taxi for a smarter City

[Logout](#)

Hi TaxiDriver_12345!

Are you available? ☐ Yes ☐ No [Refresh](#)

Pending Request:

Phone Row, 3331234567
Buenos Aires Road, 12 [Accept](#) [Decline](#)

Last Request:

Patric Branchi, 3409876543
Starbuck Plaza

5. Entities Architecture

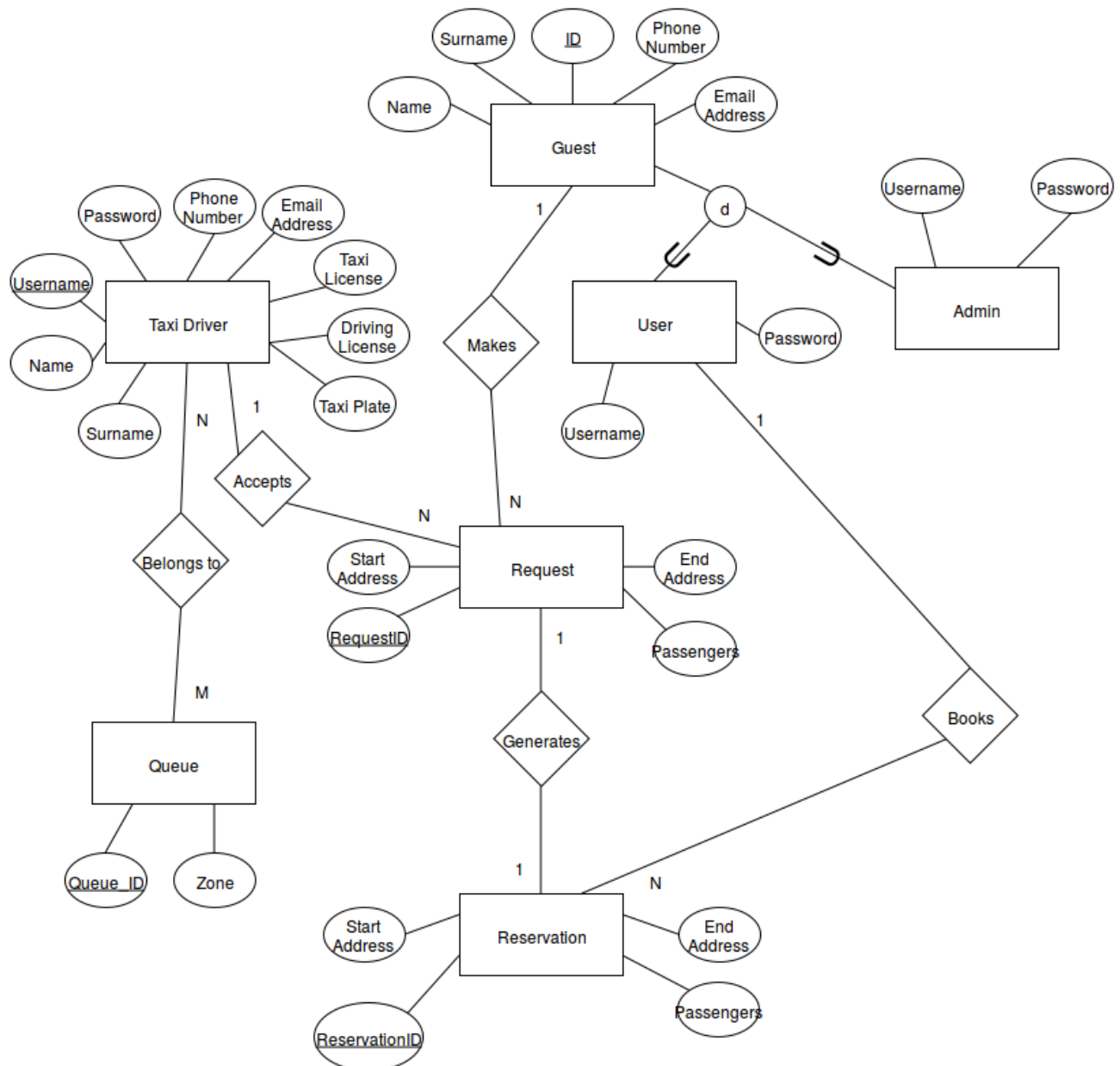
5.1. E-R Diagram

In order to clarify the data structures used in the application, here is reported the ER (Entity-Relationship) diagram for the application database.

Every entity has a primary key, a field already suitable or inserted that is fundamental to identify every record of the database; in this way there are no weak entities.

Every relation between entities is represented with its cardinality, and the diagram uses the UML standard for entities and attributes, becoming clear to read and self-explaining.

Entity Relationship Diagram



5.2. Relational Model

The final relational model for the database is easily designed from the ER diagram as follows, where *italic* is used to denote a foreign key, and underlined fields are primary keys.

- Customer (User_Type, Name, Surname, ID, Username, Password, Phone_Number, Email_Address)
- Taxi_Driver (Name, Surname, Taxi ID, Phone_Number, Email_adress, Username, Password, Driving License, Taxi License, Taxi Plate)
- Request (Start_Address, End_Address, Request ID, Passengers)
- Reservation (Start_Address, End_Address, Reservation ID, Date, Time, Passengers)
- Queue (Queue ID, Zone)
- UserRequestTaxi (User ID, Request ID, Taxi ID)
- ReservationGeneratesRequest (Reservation ID, Request ID)
- UserMakesReservation(User ID, Reservation ID)
- TaxiBelongsToQueue (Taxi ID, Queue ID)

The model is built following the normal forms principles: every entity has a unique primary key that identifies uniquely every other attribute, and every relation is identified by the foreign keys belonging to the entities involved.

6. Requirements Traceability

In this chapter all the requirements presented in RASD section 2.1 (Functional Requirements) will be mapped into the design element(s) which actually perform them.

To link this document to the RASD it is necessary to recall every requirement presented and show how they are managed at the component design level, and so what component is responsible for which function.

Using the notation presented in RASD here is the requirements map:

[G1] Registration

[R1], [R2]: Web Services: the web page present the form that will be filled in. The data will be saved in the database using the specific interfaces. The admin is in charge to check personal data inserted and eventually accept or delete users, and in particular taxi drivers.
[R2.1], [R1.3]: Notification Centre that will send the emails after the users registrations.

[G2] Taxi Queues Management

[R1.1]: Queue Web Server, that receives the availability from the taxi driver and forwards the data to the specific Queue Manager.

[R1.2]: granted by the external Localization System, that calculates and provides the position of the taxi to the Queue Service.

[R1.3], [R1.4]: assigned to the Queue Web Server that, managing every queue, is able to insert the taxi in the right queue and make sure that the taxi rotation is fair and effective.

[G3] Calling a Taxi

[R1]: Request Manager which permit customers to forward a taxi call Request.

[R1.1]: Customer component, which represents both the unregistered user and the registered one.

[R1.2], [R1.3], [R1.4]: This feature is described in the Runtime Behaviour: Request where we can see that the customer has to fill a form before making a taxi call.

[R1.5], [R1.6]: The position is calculated by an external app represented with the Localization System.

[R2]: Taxi Driver component provide an interface through which Request Manager can forward it a request.

[R2.1]: Queue Web Server and its internal Queue components provide the functionality that permit to retrieve from the database the first taxi driver in the queue. That behaviour is described in the specific sequence diagram.

[R2.2], [R2.2.1], [R2.2.2]: A taxi driver can accept or deny a request and communicate it through its interface Available Taxi. Queue Manager changes the position in the queue of the taxi according to these requirements.

[G4]: Book a Taxi in advance

[R1]: Reservation Manager and Client component satisfy this requirement.

[R1.1], [R1.2], [R1.3]: Sequence Diagram: Reservation. The components that participate are Reservation Manager, Client and Queue Manager.

[R1.3.1]: This requirement has been modified since the concurrency is managed by the use of a thread pool pattern which permit simultaneous requests.

[R1.3.2]: deleted.