MyTaxiService

Design Document

Software Engineering 2 Project AA 2015-2016

04/12/2015

Elis Bardhi **790135**

Andrea Cavalli **841512**

Mario Dolci **773705**

1. Introduction

1.1. Purpose

This document intends to explain the general and specific architectures of the system. Its main aims are to describe and justify the architectural decisions that we have taken in the design process. In particular, in this document there will be presented the relations and interactions between the various parts of the system.

1.2. Scope

The scope of the architecture design is based on the RASD document; in this phase we do not provide all the functional requirements already described in the RASD document, but we present the general functions of the system:

- **Manage accesses to the system:**
  MyTaxiService will allow guests to sign in into the system in order to have access to its services;
- **Manage users:**
  MyTaxiService will allow to manage three types of users: guest, client and taxi driver. Each type of user can access some specific services of the system dedicated for it.
- **Manage taxi requests and reservations:**
  MyTaxiService will allow clients to request or reserve a taxi;
- **Manage the taxi sharing option:**
  MyTaxiService will allow clients to choose if to share its ride with other people;
- **Manage the clients' requests:**
  MyTaxiService will allow taxi drivers to accept or refuse clients' requests;
- **Manage the system calculations:**
  MyTaxiService will allow the system to calculate the ride cost and the best route for each ride.

1.3. Definitions, Acronyms, Abbreviations

The following acronyms will be used through this document:

- RASD: Requirements Analysis and Specification Document;
- DD: Design Document;
- SCS: smartphone/computer system;
- LS: listener system;
- DS: dispatcher system;
- LTS: Location-tracking system;
- TS: Taxi system;
- DBS: Database system;
- PHP: Hypertext PreProcessor;
- XML: eXtensible Markup Language;
- JSON: JavaScript Object Notation;
- UI: User Interface;
- SOA: Service-oriented architecture;
- UX: User Experience;
- JSF: JavaServer Faces.

1.4. Reference Documents

The following references were used in order to write this document:

- MyTaxiService RASD document;
- MyTaxiService Project Description and Rules;
- MyTaxiService Assignments 1 and 2 (RASD and DD);
- Structure of the design document;
- Design (Part I & II);
- Architectural Styles (Part I & II);

1.5. Document Structure

The Design Document specifies the architecture of the system from a general view to a more specific one, using different levels of detail.
The document is divided into various sections:

- **Introduction**
  This section presents the purpose of MyTaxiDriver services and its main functionalities. It also proposes a list of definitions and abbreviations in order to give a better reading experience to the reader.
- **Architectural Design**
  This section provides the architectural design of the system starting from a general overview to a more specific analysis. Here the modules of the system and their interaction are presented, and it also contains the styles decision and the pattern that we have decided to use in order to build the system.
- **Algorithm Design**
  This section presents the definitions of the most relevant algorithmic part of the project.
- **User Interface Design**
  This section provides an overview of the UI already amply specified in the RASD document.
- **Requirements traceability**
  This section explains how the requirements already described in the RASD document map into the design elements of the system.
- **RASD Modifications**
  In this section we present the changes that we have introduced in the RASD document.
- **Used Tools**
  This section contains the software programs used in order to write this document.
- **References**
  The section presents a compilation of extra referenced used to write this document.
- **Hours of work**
  It contains the amount of hour for each element of the group.

2. Architectural Design

2.1. Overview

The MyTaxiService system in general will consist of 6 major parts: a smartphone (or a computer) system, a taxi dispatching system, a listener system, a location-tracking system, a database and a taxi system. The combination of these 6 components will realize the full potential of the MyTaxiService system.

2.2. High level components and their interaction

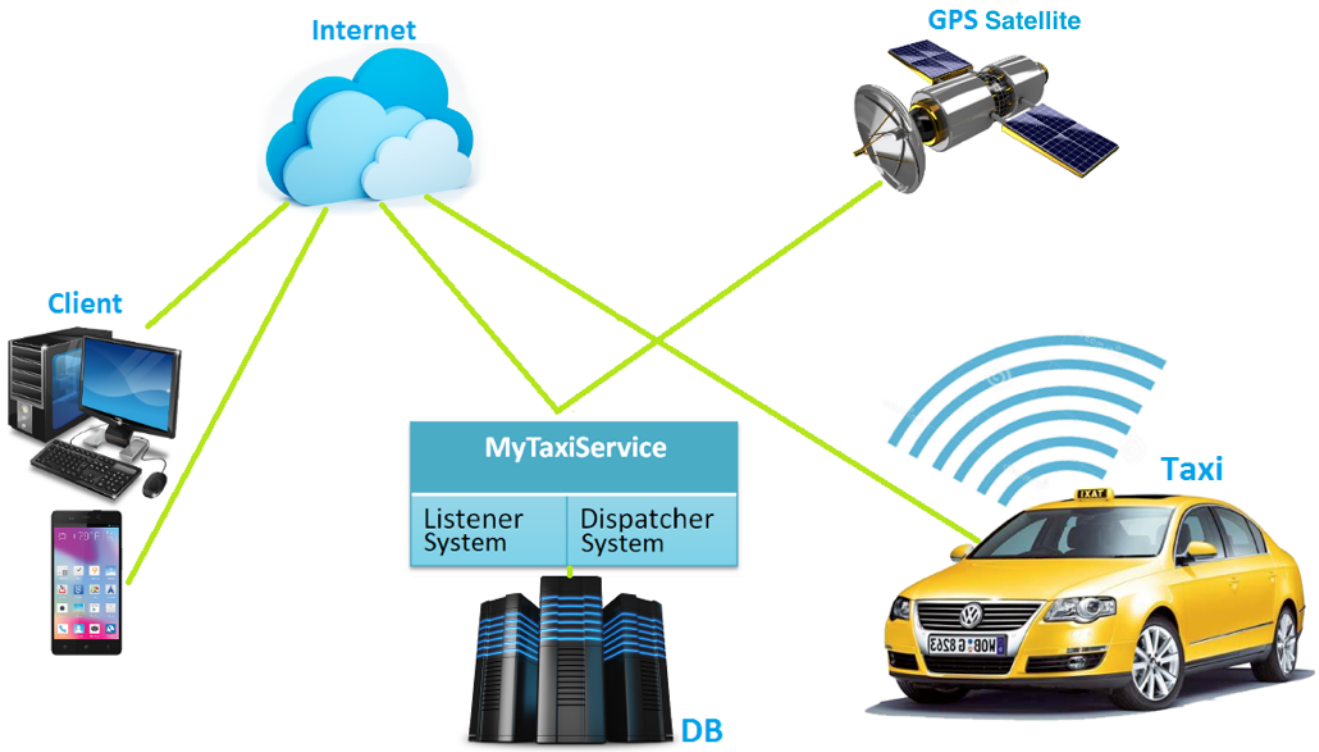

Figura 2.2.0

The smartphone (computer) system SCS will connect clients through to Listener System.
This system is installed into smartphones or it can be accessed via web going to http://www.mytaxiservice.com. Though this system guests or clients can sign up/in, make taxi requests and modify their account's information. This system is also used to display notifications to clients about their rides. The SCS interact only with:

- LS through internet exchanging simple text messages.

The listener system LS is the main part of the system. The main goal of this system is to collect all requests incoming from clients and prepare them to be elaborated. This system has also to check information validity such as addresses existence, to provide suggestions about compiling various forms such as auto completing fields and to assign taxies to requests. Another purpose of LS is to

follow clients during all the process life, from the request until the end. This system has to interact with:

- SCS through internet exchanging messages;
- DBS by queering the database;
- DS through a dedicated interface.

The dispatcher system DS is another important part of our whole system. Its main task is to track all taxies on duty and associate them to zones, also it manages taxi queues and calculates routes. This system has to interact with:

- LS through a dedicated interface;
- LTS through internet by querying the LTS database;
- DBS by querying the database;
- TS through internet exchanging messages.

The location-tracking system LTS is an external (third-party) system that has the capability to localize objects using satellite system. Each taxi will be equipped with a tracking device. The tracking system will provide real-time information about each taxi location. This system interacts only with:

- DS through internet sending query results;
- TS through radio waves incoming from GPS device installed in the taxi.

The taxi system TS is installed in each taxi. A taxi will have a tracking device installed as described in the location-tracking system. Each taxi driver will also have an online terminal, which consists of a smartphone connected to the internet. Communication between the dispatcher system and the taxi system will be through the internet. This system has to interact with:

- LTS through radio waves incoming from GPS satellite;
- DS through internet.

The database system DBS is another important component of our system. This system is used to store all the information about clients, taxi drivers, type of rides, taxies code and other types of information. The DBS interact only with:

- LS by sending results of queries;
- DS by sending results of queries.

Communications: communications between components of the system are assumed to be asynchronous: for instance, MyTaxiService can send messages to clients without their request. Another reason of this decision is that having an asynchronous communication the whole system cannot go in stall.

The system architecture is based on a layer style where the presentation, the business logic and the persistence of the system are clearly separated, as show in the following figure:

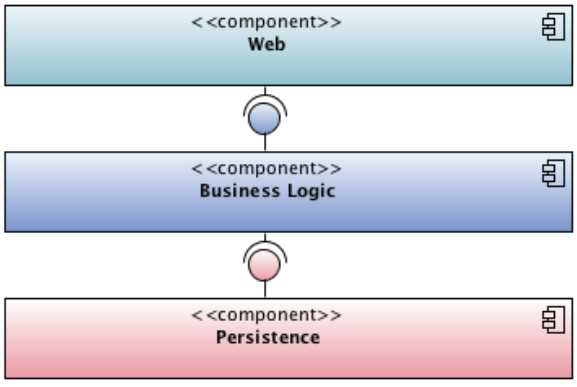

Figura 2.2.2

For our design the previous architecture can be detailed in specific components according to the software requirements.

The components shown in the following figure are the general components needed for implementing MyTaxiService system. In particular, in the next sections and chapters we will describe in more detail design aspects for each component of each layer. Here we explain what components of each layer contains.

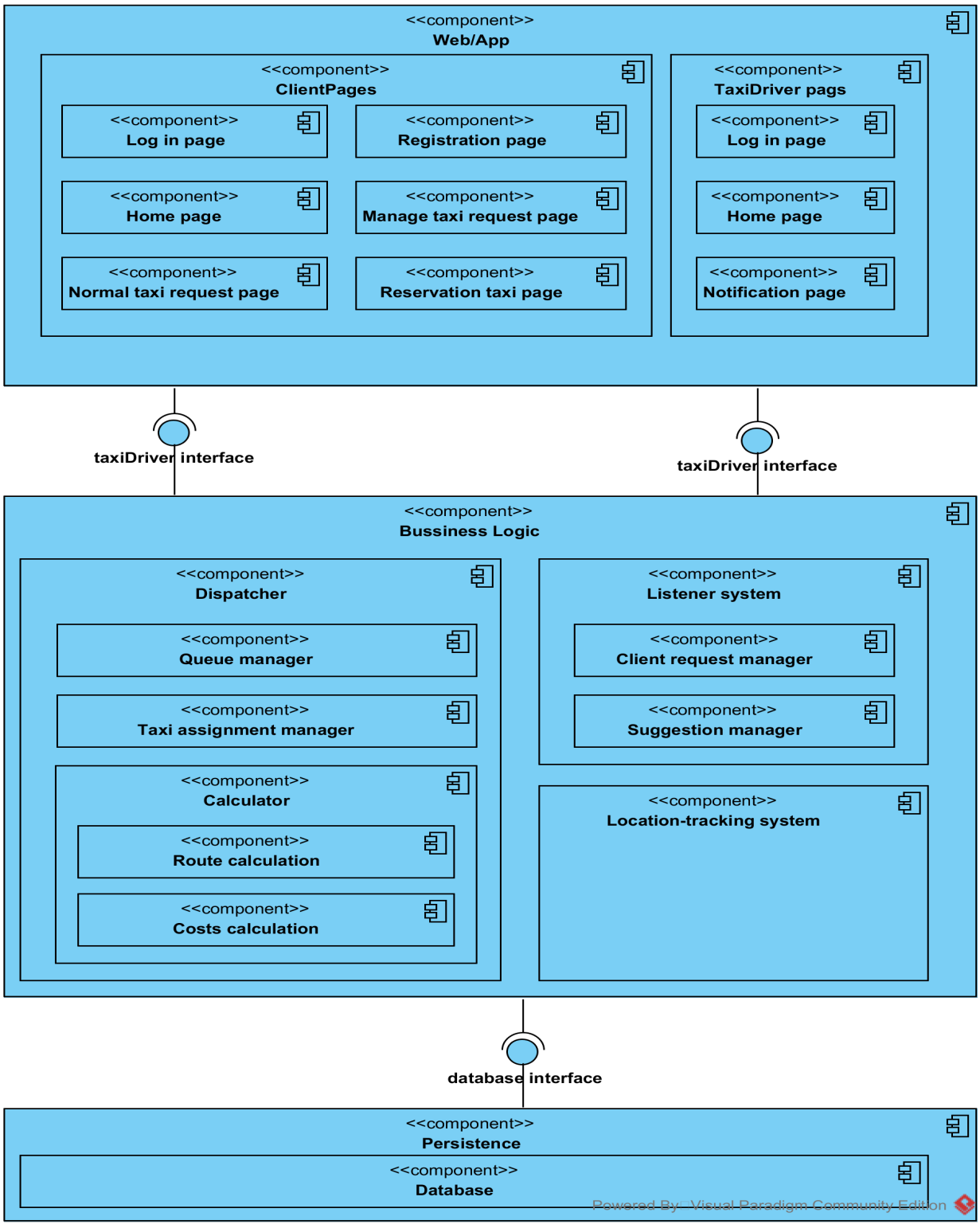

Figura 2.1.3

2.3. Component view

This section explains in detail each component mentioned in the last chapter, specifying their responsibilities, methods, constraints and resources among others. Additionally, in this section we will define the database structure of the system.

2.3.1. Presentation layer

This layer contains all the different kinds of web pages/app UI needed for interacting with each type of system user. The app used by the client and taxi driver is available both for Android smartphones and iPhone and for clients who don't use the app is available a website which offer the same services of the app. Pages are classified into two categories:

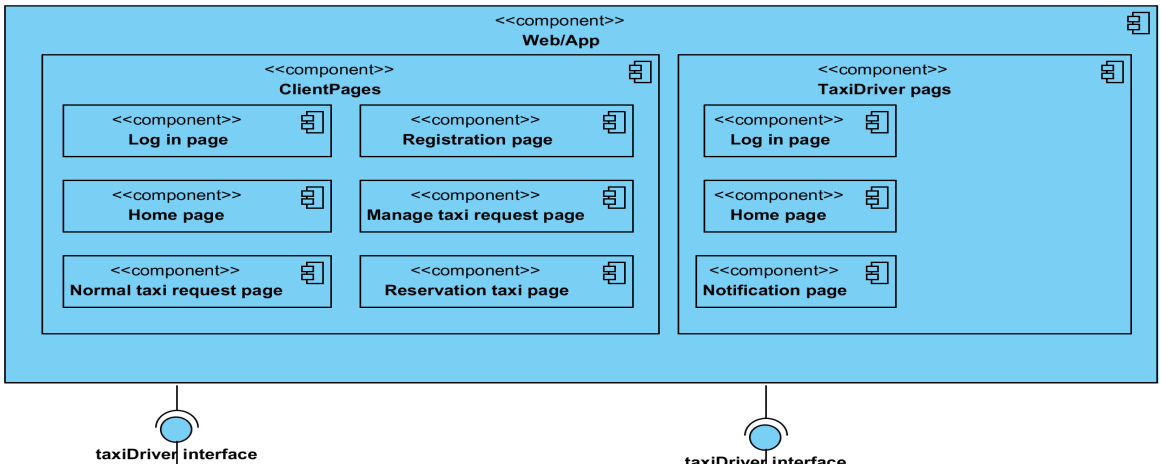

Figura 2.3.1

- **Client Component**: These component is used by clients in order to interact with the main system. This component represents essentially the app which clients use to request, modify and get notifications. When the user accesses to the app, two services are started: Listener/Sender Service. These services are used to send/receive data.

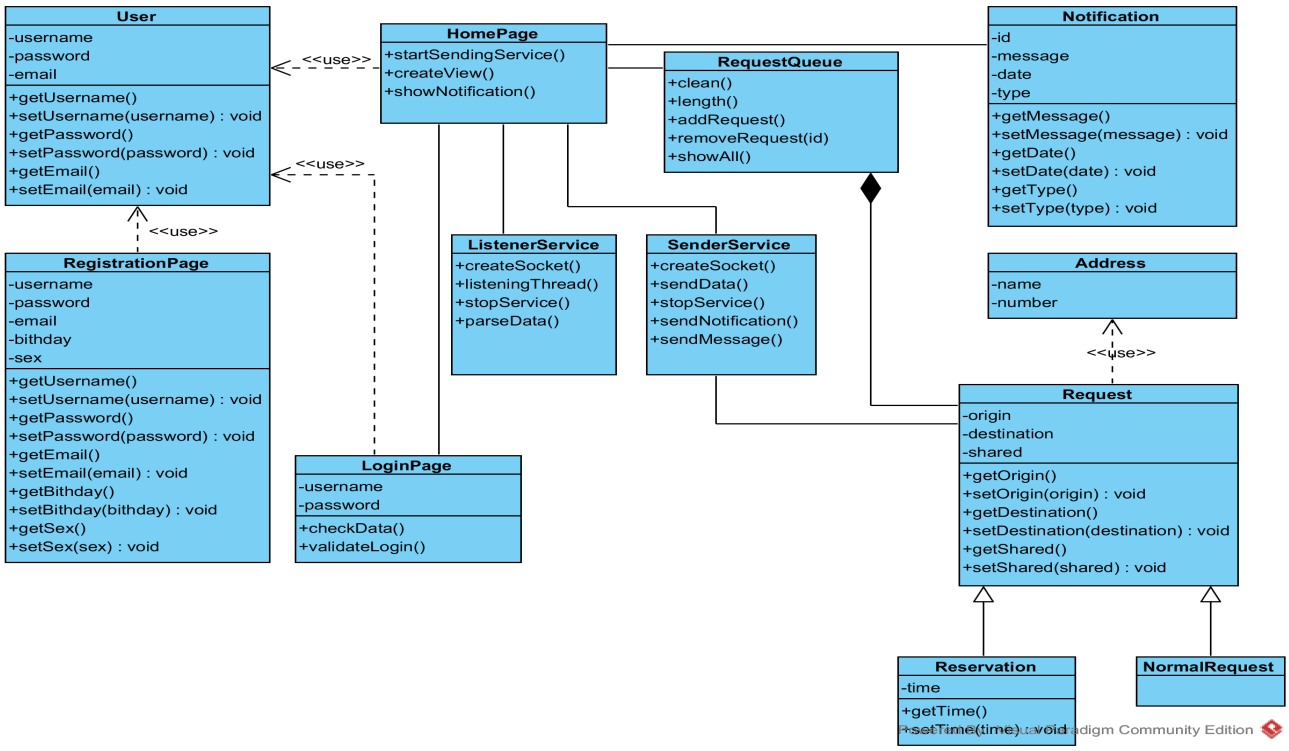

Figura 2.3.2

- **TaxiDriver Component**: This component is used by taxi drivers in order to interact with the system. This component represents essentially the app which a taxi driver use to receive new taxi requests from the main service.

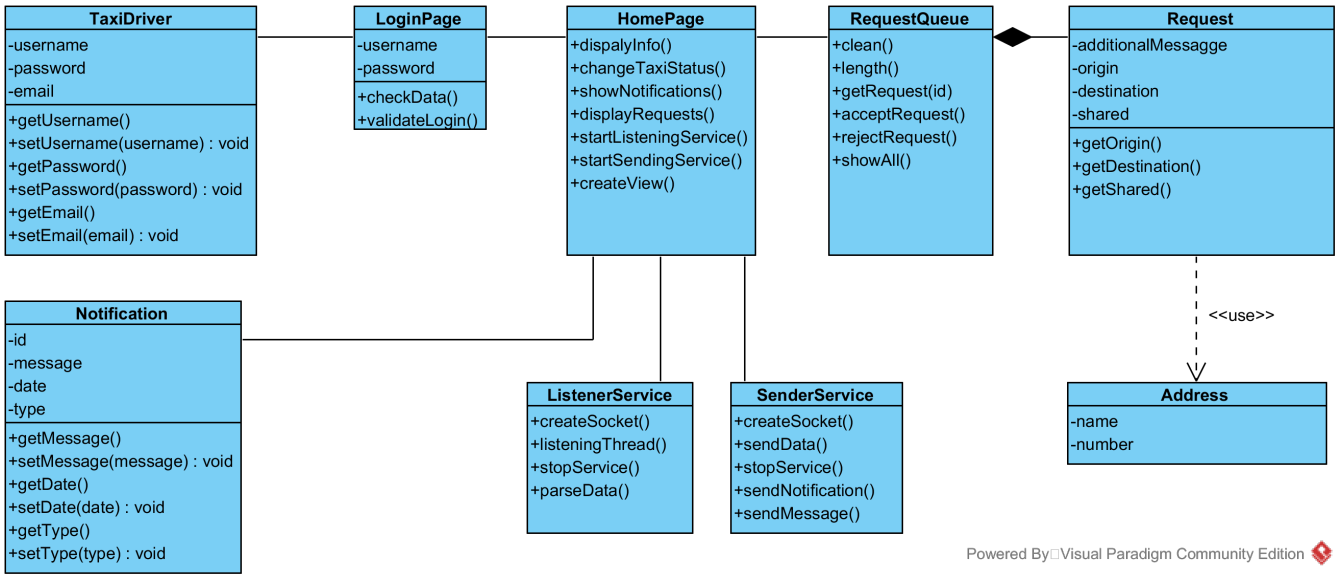

Figura 2.1.3

2.3.2. Business layer

This layer contains the components in charge of the management of the different functionalities of the system. This layer basically represents the server where is implemented all the logic of our system. All the services offered by the server are implemented using PHP.

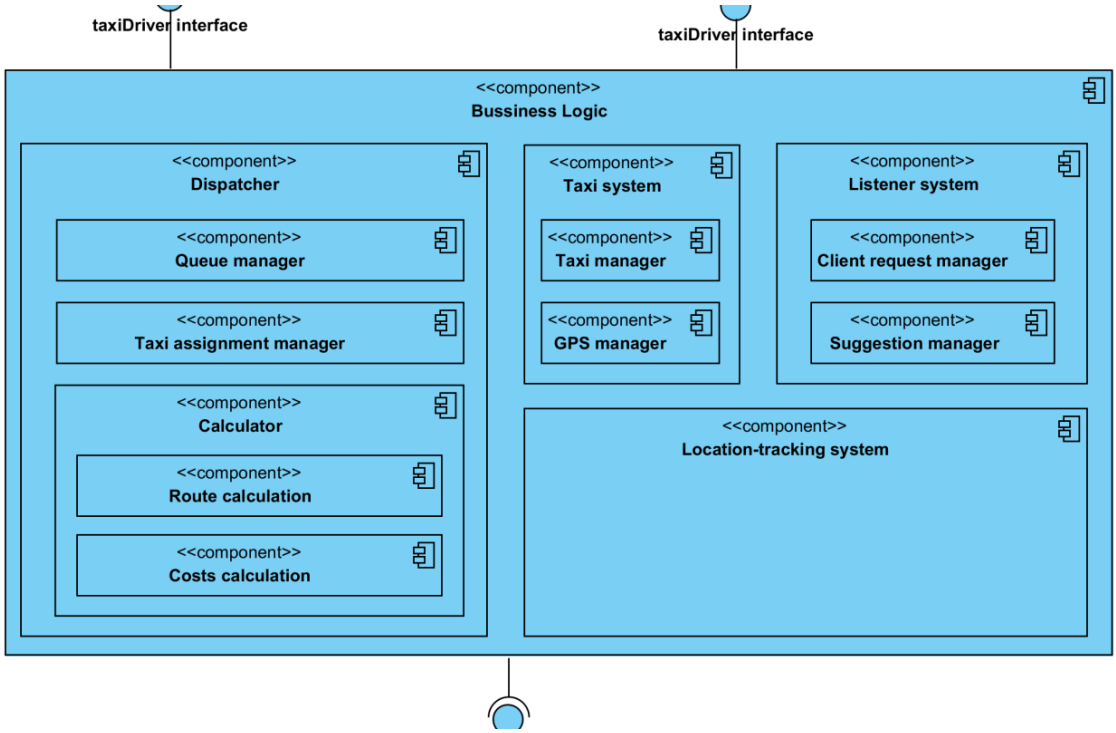

Figura 2.3.4

Dispatcher System: The DS component manages all taxis, routes, costs and assignments. It is connected to LS component from which it gets all the incoming taxi requests and tries to assign to each of them a taxi. This also manages the position of taxies in the queue.

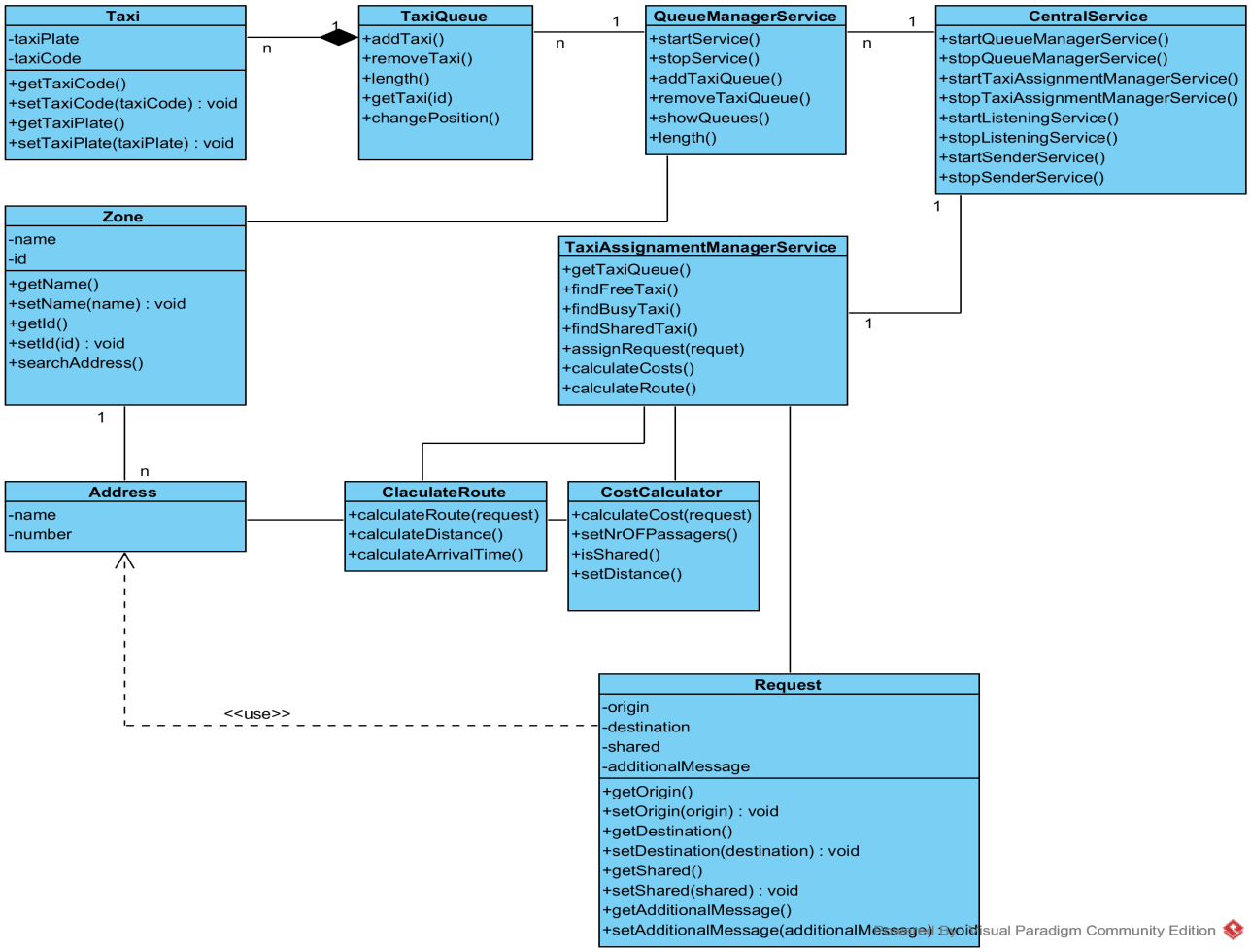

Figura 2.3.5

Listener System: This component is essential for the whole system because it manages all the data that the other components send or receive.

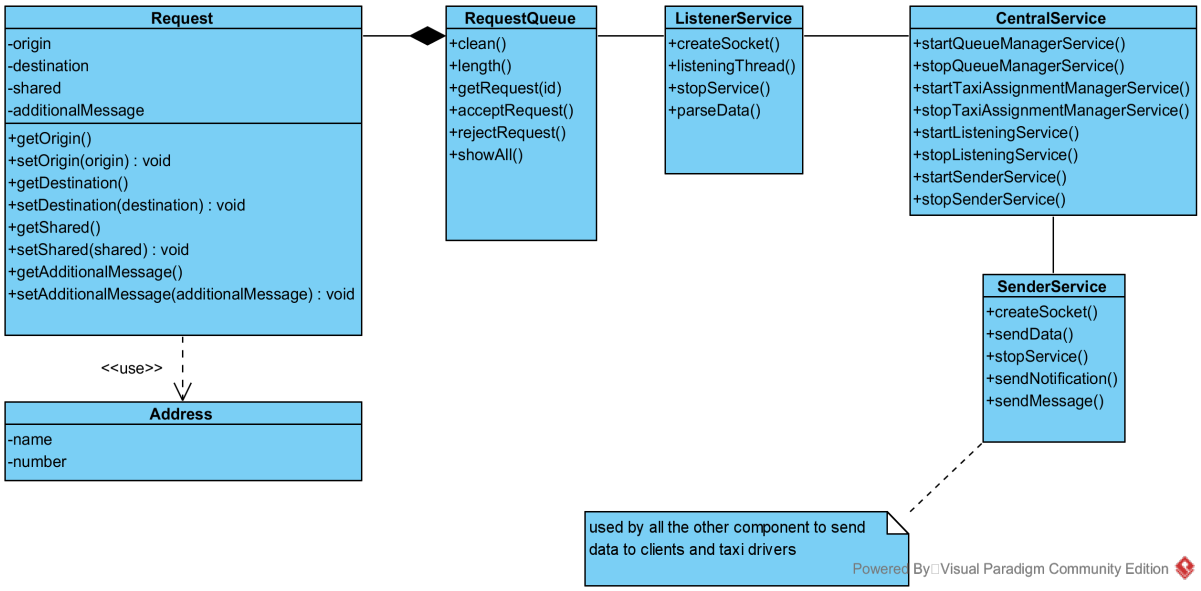

Figura 2.3.6

Location-tracking System: This component is used to take trace of the position of all taxis. It has a service that is linked to an external third-party GPS service from which receive the position of all taxies.

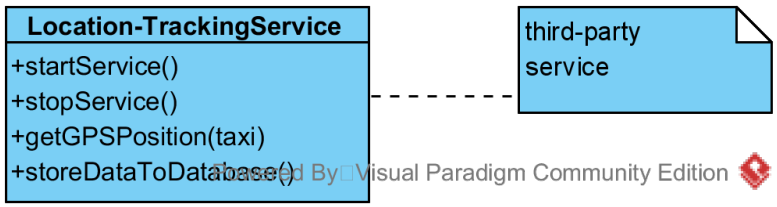

Figura 2.3.7

2.3.3. Database

This subsection contains the database schema in which we are going to design the system entities. The database will interact basically with all sub-components in the business logic component. It will be used to store all clients' information such as profile info and rides, and also to store taxi drivers' information, taxis info etc. Security restrictions will be implemented in order to protect data from unauthorized accesses and in particular the access will be granted only to authorized users.

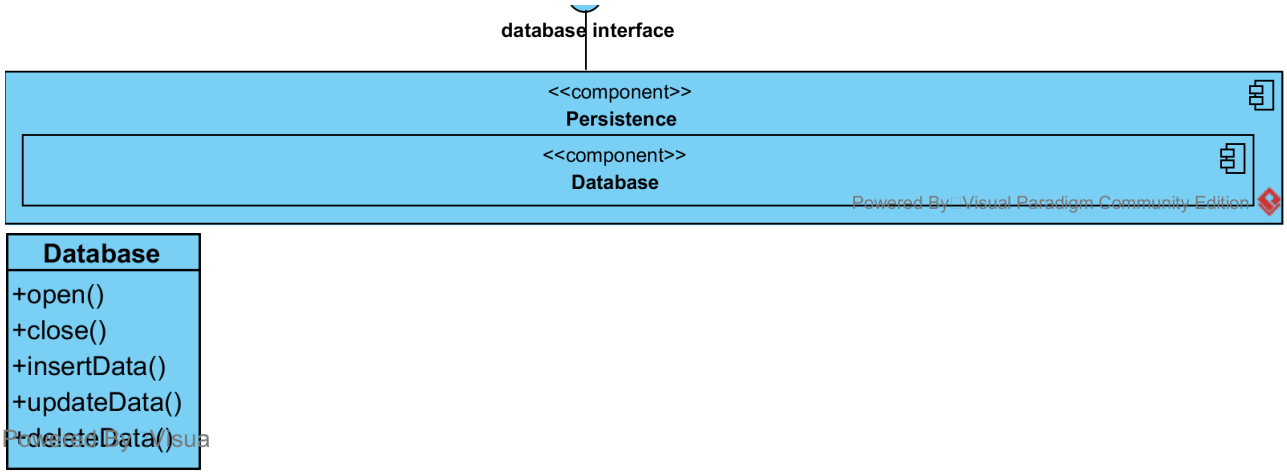

Figura 2.3.8

2.3.3.1. Conceptual Design

The conceptual design is the first step to do in order to create a database. We have focused on the most important data that have to be stored in order to let the service to run.
The conceptual Entity-Relationship diagram can be found below.

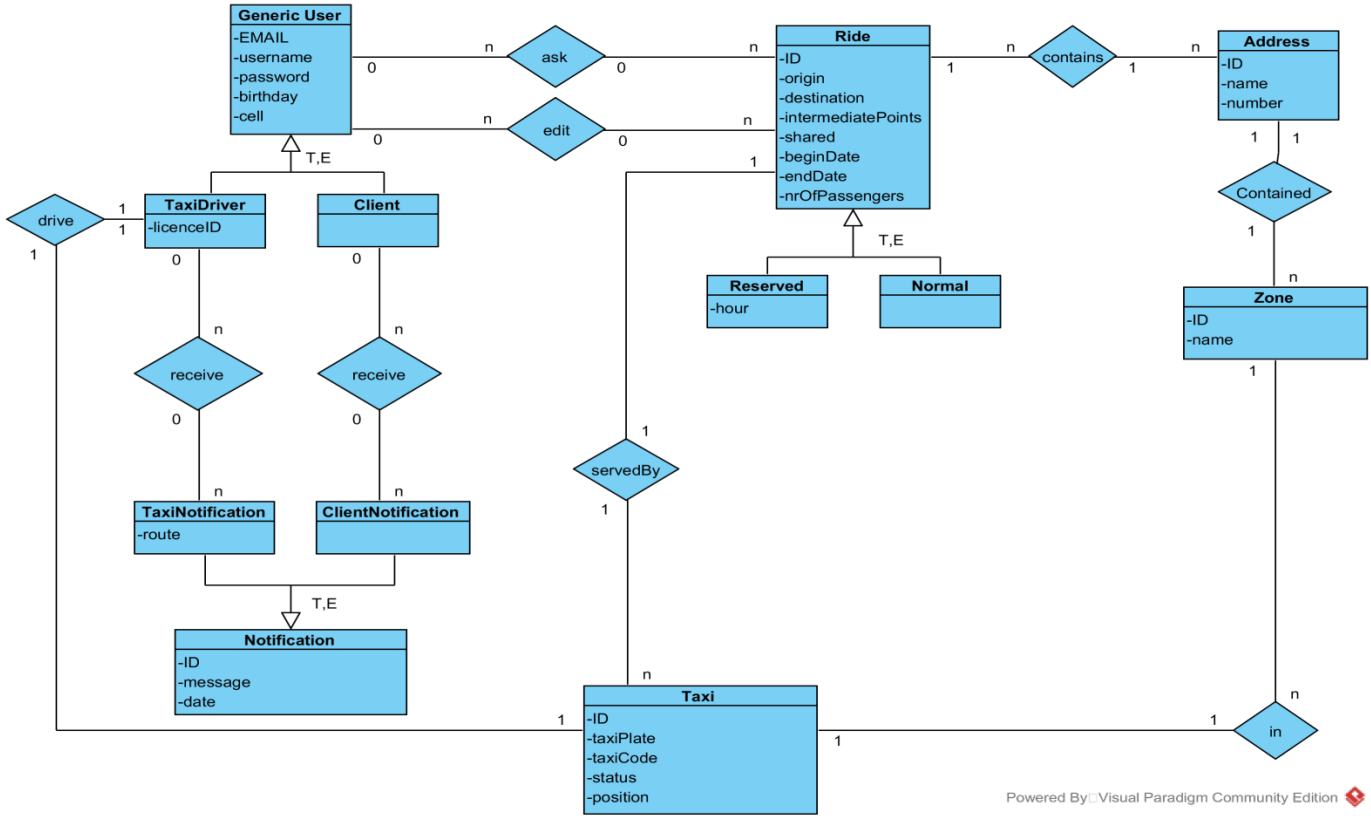

Figure 2.3.9

The keys of the entities are in capital letters.
In our system a user can be “Taxi Driver” or “Client”. These two entities are inherited from the “Generic User” one, since they have a lot of information in common. For the same reason we have the entities “Reserved” and “Normal” rides inherited from “Ride” entity.
A Generic User can ask or edit a Ride; we also have created “Address” and “Zone” entities in order to guarantee more extensibility to the system: if any developer wants to extend our application, he will not need to change the database structure.
The description of all relationships between entities will be provided in the following list:

- **Ask:** between Generic User and Ride, every user can ask several rides;
- **Edit:** between Generic User and Ride, every user has the possibility to edit several rides;
- **Receive:** between Taxi Driver and Taxi Notification, every taxi driver can have different Taxi Notifications and the same Notification can be sent to different taxi drivers;
- **Receive:** between Client and Client Notification, every client can have different Client Notifications and the same Notification can be sent to different clients;
- **Drive:** between Taxi Driver and Taxi, every Taxi Driver must have exactly one Taxi and viceversa;
- **Servedby**: between Ride and Taxi, every Taxi can serve different Rides but specific ride can be served by only one Taxi;
- **In:** between Taxi and Zone, a Taxi must be in a zone and in a zone there is at least one Taxi;
- **Contains:** between Ride and Address, every ride contains different Addresses and each Address can be in different rides;
- **Contained:** between Address and Zone, every Address must be in a Zone and a Zone is composed by different Addresses.

2.3.3.2. Logical Design

The logical design is the second step in order to create the physical database. This subsection contains the database design in which we are going to design the system entities.

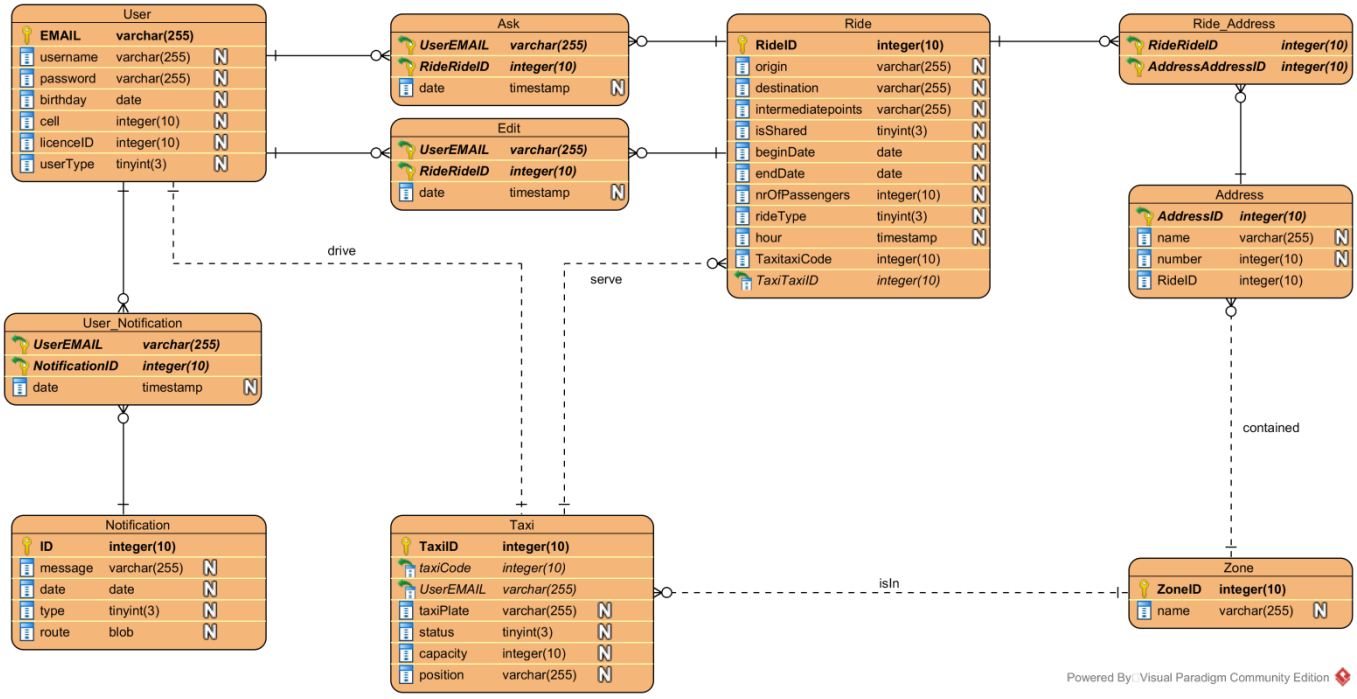

Figure 2.3.10

The database will have the following physical structure:

- **User**: (EMAIL, username, password, birthday, cell, licenceID, userType);
- **User_Notification**: (UserEMAIL, NotificationID, date);
- **Notification**: (ID, message, date, type, route);
- **Ask**: (UserEMAIL, RideID, date);
- **Edit**: (UserEMAIL, RideID, date);
- **Taxi**: (TaxiID, *taxiCode*, *UserEMAIL*, taxiPlace, status, capacity, position);
- **Ride**: (RideID, origin, destination, intermediatePoints, isShared, beginDate, endDate, nOfPassengers, rideType, hour, TaxiCode, *TaxiID*);
- **Ride_Address**: (RideID, AddressID);
- **Address**: (AddressID, name, number, RideID);
- **Zone**: (ZoneID**,** name);

The primary keys are underlined and the foreign keys are written in *italic*.

2.4. Deployment view

An overview of the system architecture is provided with the following schema.

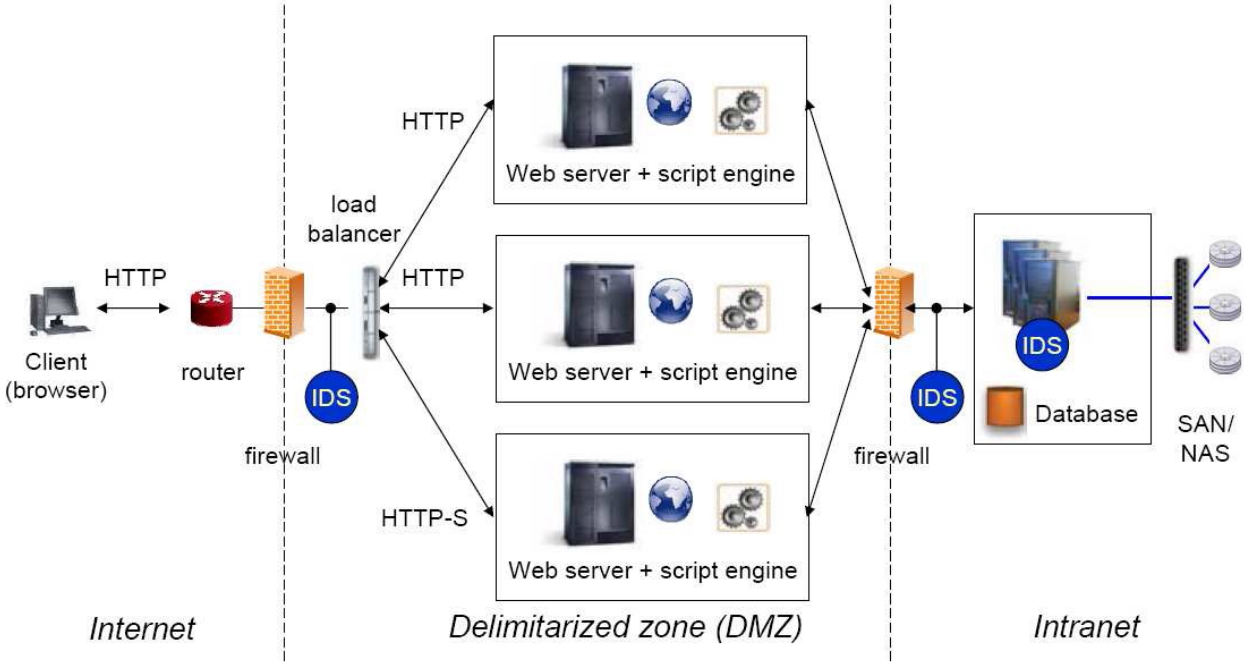

Figure 2.4.1

- **The Web Server** is placed between the end user and the information system. In particular, it is the component that handles HTTP requests from the Internet or Intranet, returning to the applicant of static pages, or through interaction with the script engine, the dynamic result of processing.

- **The Script engine** is a process that runs scripts in order to generate dynamic web pages. Then this pages are sent to the end user by the Web server.

- **DBMS** deals with the management of the database. The script engine must also interact with the DBMS to generate dynamic pages requested by the end user.

- **Firewall** is a network security system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules. A firewall typically establishes a barrier between a trusted, secure internal network and another outside network, such as the Internet, that is assumed to not be secure or trusted.

- **IDS (Intrusion Detection System)** is a system hardware or software that automates the process of monitoring events of a system in order to detect intrusions (attempts to compromise confidentiality, integrity or availability of the system itself).

We have decided to opt for a 3 tier server farm architecture for these following reasons:

- Increased availability: if one of this servers falls, the others will continue to provide the service, or at limit the service has a partial degradation;
- Increase scalability: actions can be taken separately on the 2 tiers and sizing the 2 tier is less critical;
- Improve performance;
- This configuration has a better security: DB has double protection, due to the two firewalls;

The 3 layer architecture is shown in the Figure 2.4.2.

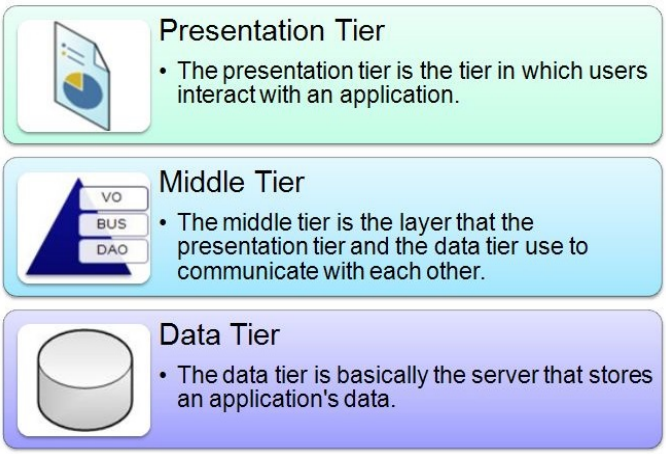

Figure 2.4.2

This architecture responds to the need by different applications to access to the same database.

Since this is a client-server architecture, we have to establish what type of client we want: a **fat or thin client**.

We opted for a thin client which contains only the presentation layer. MyTaxiService clients' interaction is provided through a app or web application, which will run on any modern smartphone and computer, and since all the logic and all the data of the application will usually be located on other machines we don't need to have a fat client. Adopting this strategy, the security will be also improved.

2.5. Runtime view

In RASD document we have already described the flow of events and the processes which occur when a service is called. In this section we will speak about processes more in detail showing how the system works when:

- A client makes a normal taxi request. In this case only two systems are used, Client and LS.

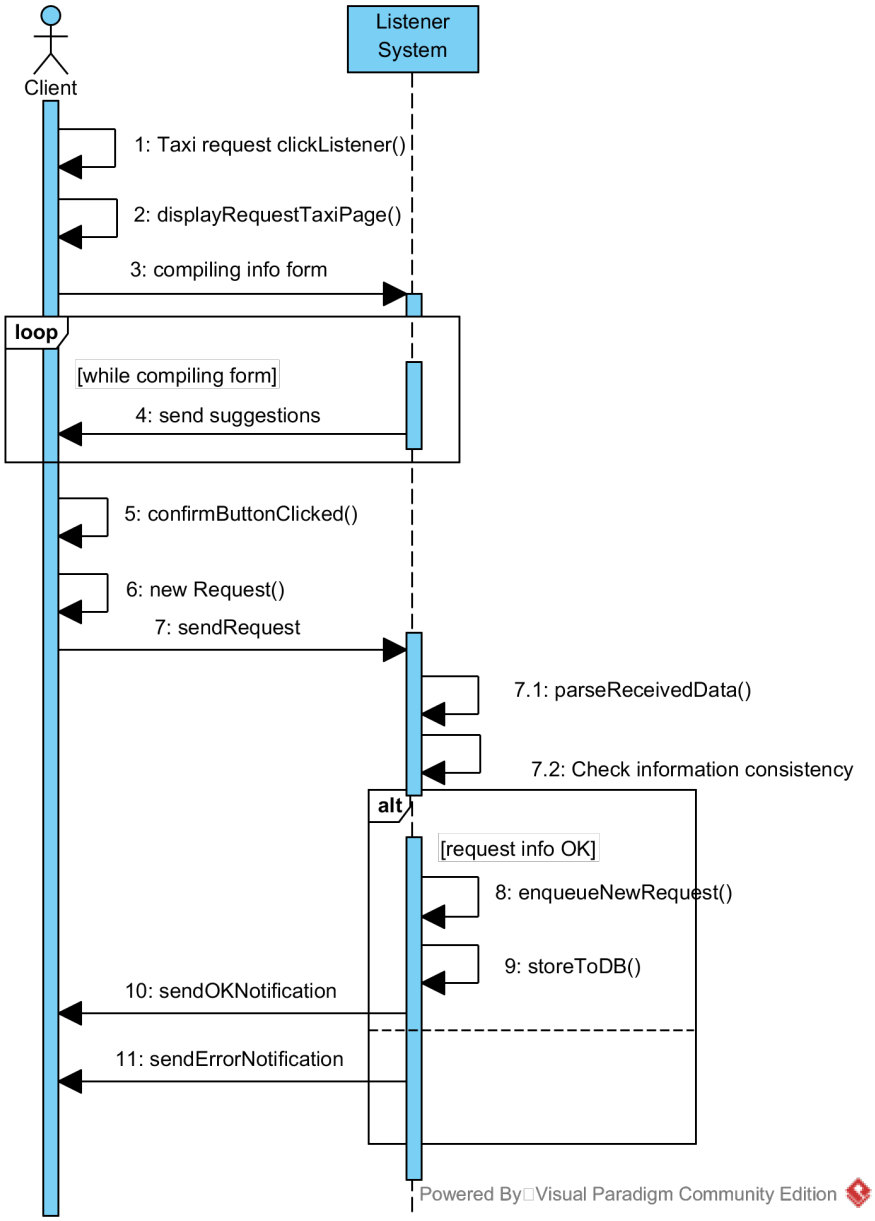

Figure 2.5.1

- The DS receives a new request from the Listener System.

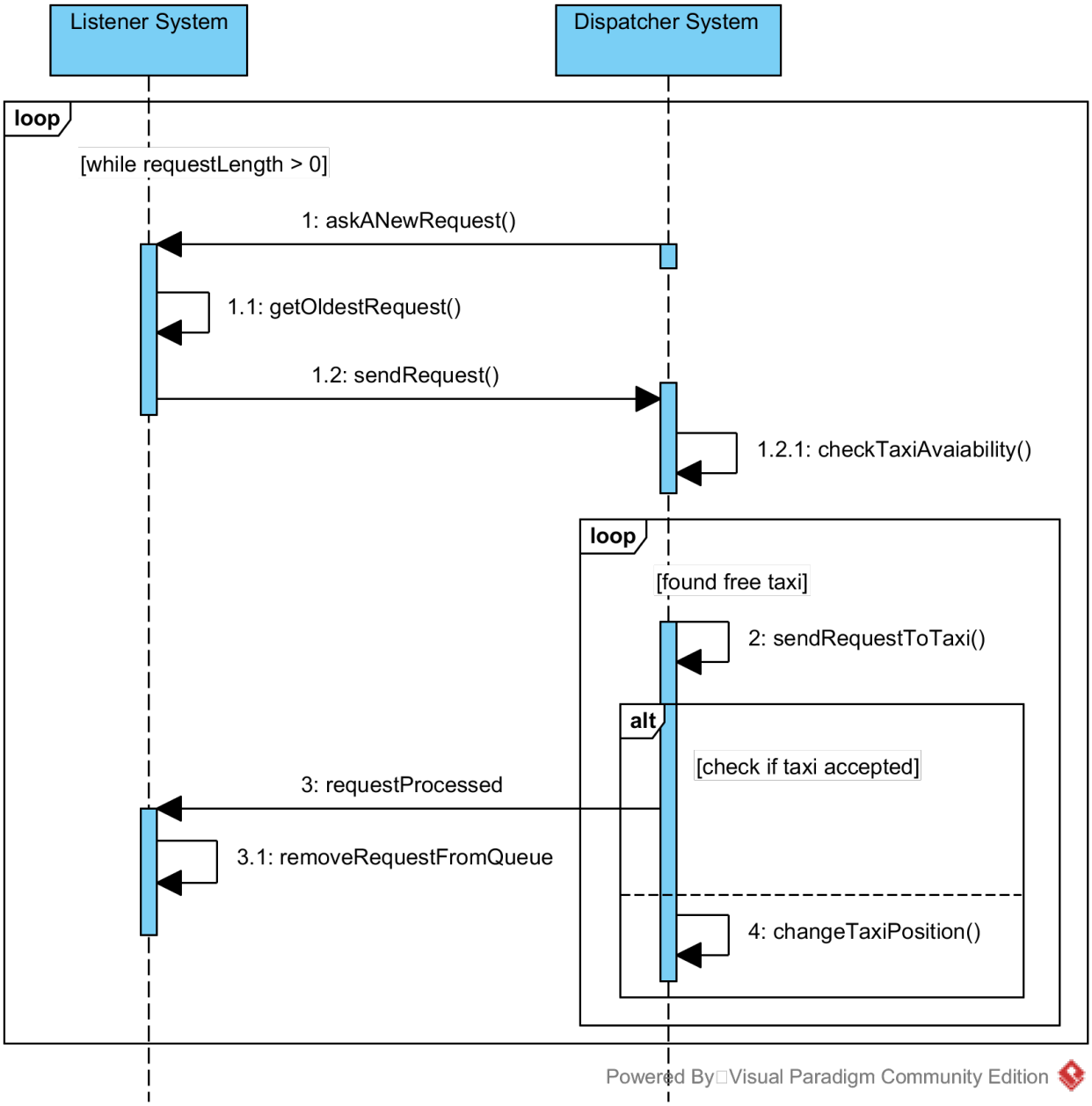

Figure 2.5.2

- The DS sends a request to a Taxi

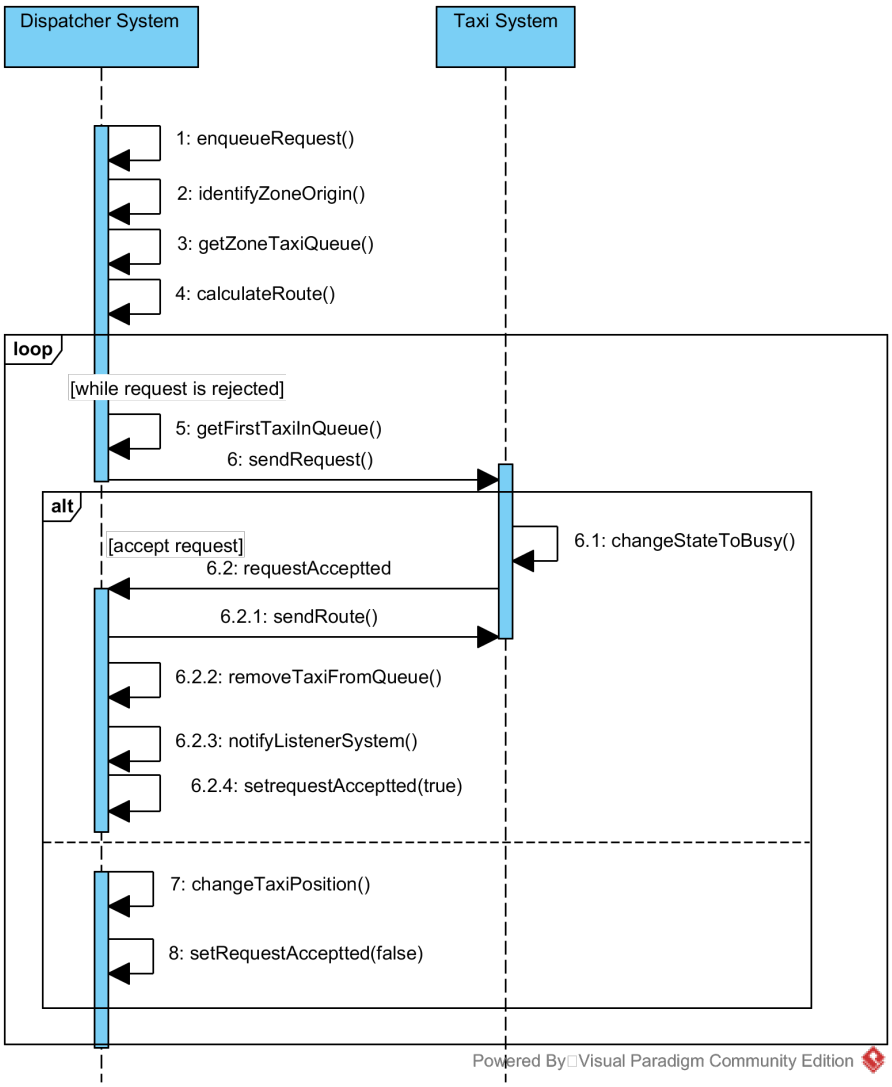

Figure 2.5.3

- The LTS updates the taxi position

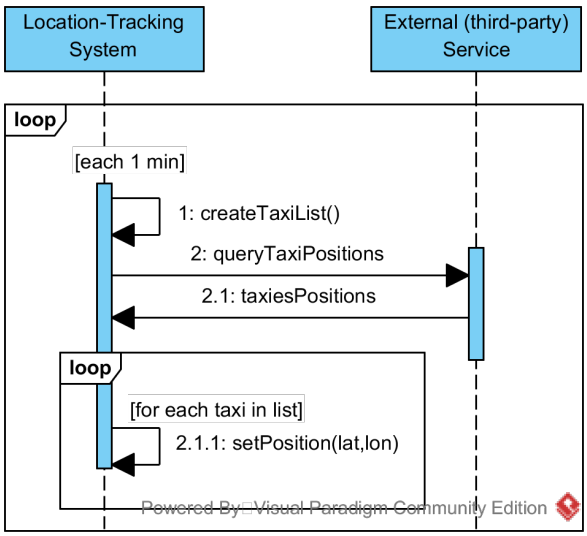

Figure 2.5.4

2.6. Component interfaces

There are only two interfaces that can be detected in our system and they refer to the external services used by the system.

Service interface	Google Maps
Functions offered	displayMap(double latitudine, double longitudine, float radius): MapView getDistance(position origin, position destination):float getDistance(position multiPoints[]):float traceRoute(position origin, position destination):route traceRoute(position multiPoints[]):route travelTime(position origin):int

Service interface	GPS Satellite
Functions offered	getPosition(String taxiCode):position getLastKnownPosition(String taxiCode):position getSpeed(String taxiCode):float

The other components and services do not present an interface because they exchange data only by using HTTP requests. The data structures used to exchange information are XML and JSON.

2.7. Selected architectural styles and pattern

MyTaxiService is designed in order to be a Service-oriented architecture (SOA). Infact, MyTaxiDriver has got the following characteristics:

- Its services are runtime components;
- Its services offer operations (such as a taxi request) that can be invoked by external clients;
- The main service is obtained by assembling small services.

2.8. Other design decisions

The system uses two external third-party services:

- A location tracking service is used to localize all taxies (GPS satellite)
- Google Maps to calculate distances and routes.

We opted for those solutions because they are chipper than buying and managing a GPS satellite and than developing a personal map view of city.

3. Algorithm Design

In order to implement the system, we have supposed that MyTaxiService will be developed using the standards of the Java Platform, Enterprise Edition. Moreover, the mobile app will be developed for iOS (using Swift language) and Android.
In this section we will focus on the possible main algorithms that will characterize MyTaxiDriver system. In particular, we will discuss about three algorithms: the calculation of the best route, the calculation of the ride cost and the management of the zone queues.

1. **Calculation of the best route:** the calculation of the best route for a taxi request will be done by using Google Maps Directions APIs. When a taxi request is done, the system can set a series of at most 23 location waypoints (especially useful for the taxi sharing option) and to plan a route through them. Then, after calculating possible routes, Google Maps Directions APIs are used again in order to estimate travel time of those routes based on historical and current traffic conditions. Finally, the system will select the route with the best combination of distance and travel time.
2. **Calculation of the ride cost:** for calculating a ride cost, the system will consider both the travel distance and the elapsed time in order to cover it. This algorithm is thought especially for the city traffic which causes a relevant amount of wasted time.
   Focusing on the taxi sharing option, the system can fully manage the calculation of each cost for each passenger: for instance, if three people share a taxi and the first has to go from A to D, the second from B to C and the third from B to D, the system will divide the cost for each part of the path according to origins and destinations of each client.
3. **Management of the zone queues:** the system must manage the zone queues in order to be able to satisfy taxi requests. Each zone can be modeled as a queue that contains taxies, and taxies can be enqueued or dequeued according to the incoming requests. The system must guarantee that there will be no empty queues: there must be at least one taxi for each zone ready to satisfy a taxi request.
   If the taxi driver on the top of the queue refuses a request, he will remain in that queue but the system will put him at the bottom of the zone queue.
   A possible implementation in Java language of the management of the queues can be the following.

```
public class Queue{
	private LinkedList list;

	//Queue constructor
	public Queue(){
		list = new LinkedList();
	}

	//A taxi is added to the back of the queue.
	public void enqueue(Taxi taxi){
		list.add(taxi);
	}
```

```
	//The item at the front of the queue is returned and deleted from
	//the queue if the list is not empty. Returns null if the list is
	//empty.
	public Taxi dequeue(){
		Object taxi = list.get(1);
		if(!list.isEmpty()){
			list.remove(1);
		return taxi;
		}
	}
}
```

When a taxi refuses a request, the system will put him at the bottom of the queue: the operation is done by calling first the dequeue function and then the enqueue function.

4. User Interface Design

The UI of MyTaxiService has been already presented in the RASD document. In particular, it contains the mockups of the services pages accessible from a web browser or the mobile application.
In this document we have also presented a possible extension of the system: the specifics of the system say that the taxi driver can access the system only through a mobile application; we have decided to also create the mockups for the taxi drivers' web pages in order to have a more complete system.

In this section we also describe the UX given by our system to users. We realized a UX diagram in order to understand how users experience was thought.
The UX diagram is a class diagram with appropriate stereotypes:

- *<<screen>>* classes represent pages of our system;
- *<<screen compartment>>* classes represent pages which are shared by several pages;
- *<<input form>>* classes represent some input fields that can be fulfilled by a user.

Now we define in detail the main part of the UX:

0. **Login phase:** From the screen "Welcome" a guest can login in or, if he has not registered, he must move to "Sign up" screen in order to register to the system. If the registration is completed without errors, the system redirects to "Welcome" screen, where he can log in. After the login, the system automatically redirects the guest using his email: if he is a taxi driver the system will open "Taxi Driver Home" screen, otherwise it will open "Client Home" screen.

1. **Taxi Driver Home**: the "Taxi Driver Home" screen is composed of three elements:
   1.1. "Notifications" panel: it allows the taxi driver to se all his notifications and, tapping on one of them, the system will open the "Notification" screen so that the taxi driver can see the request more in detail. In this screen he can accept or refuse the request by tapping the dedicated button, or he can tap on the map and a new "Route" screen will be opened in order to let him to see the calculated route on the map.
   1.2. "Settings" Button: tapping settings button the system opens "Settings" screen in order to let the taxi driver to change his credentials.
   1.3. "Log out "button: tapping the logout button the taxi driver will logout from his page and the system will redirect him to "Welcome" screen.
2. **Client Home:** it contains all the services that clients can ask. It is composed of:
   2.1. "Normal taxi request" button: when a client tap this button, the system will open the "Taxi Request" screen to let him to create a new normal request. Then the client can fulfill the text bars with the request information.
   2.2. "Reservation" button: tapping this button, the system will open the "Taxi Reservation" screen so that the client can fulfill the text bars with the reservation information. The only difference between a normal request and reservation is that in the last the client must insert also the hour of the reservation.
   2.3. "Manage a request" button: with this button the client can edit his requests or reservations. "Edit page" screen will be opened and the client can modify the request

information inserted before or he can delete a request/reservation by tapping the dedicated button.

2.4. “Settings” button: the same “Settings” screen described before will be opened so that the client can change his credentials.

2.5. “Log out” button: tapping it, the client will be log out from his page and the system will redirect him to the “Welcome page”.

2.6. “Notifications” panel: it contains all the notifications that the client receives, with all the information so that he does not need to tap on a notification in order to have more details of it.

2.7. “Last taxi requests” panel: it contains the details of the last taxi request or reservation that a client has asked.

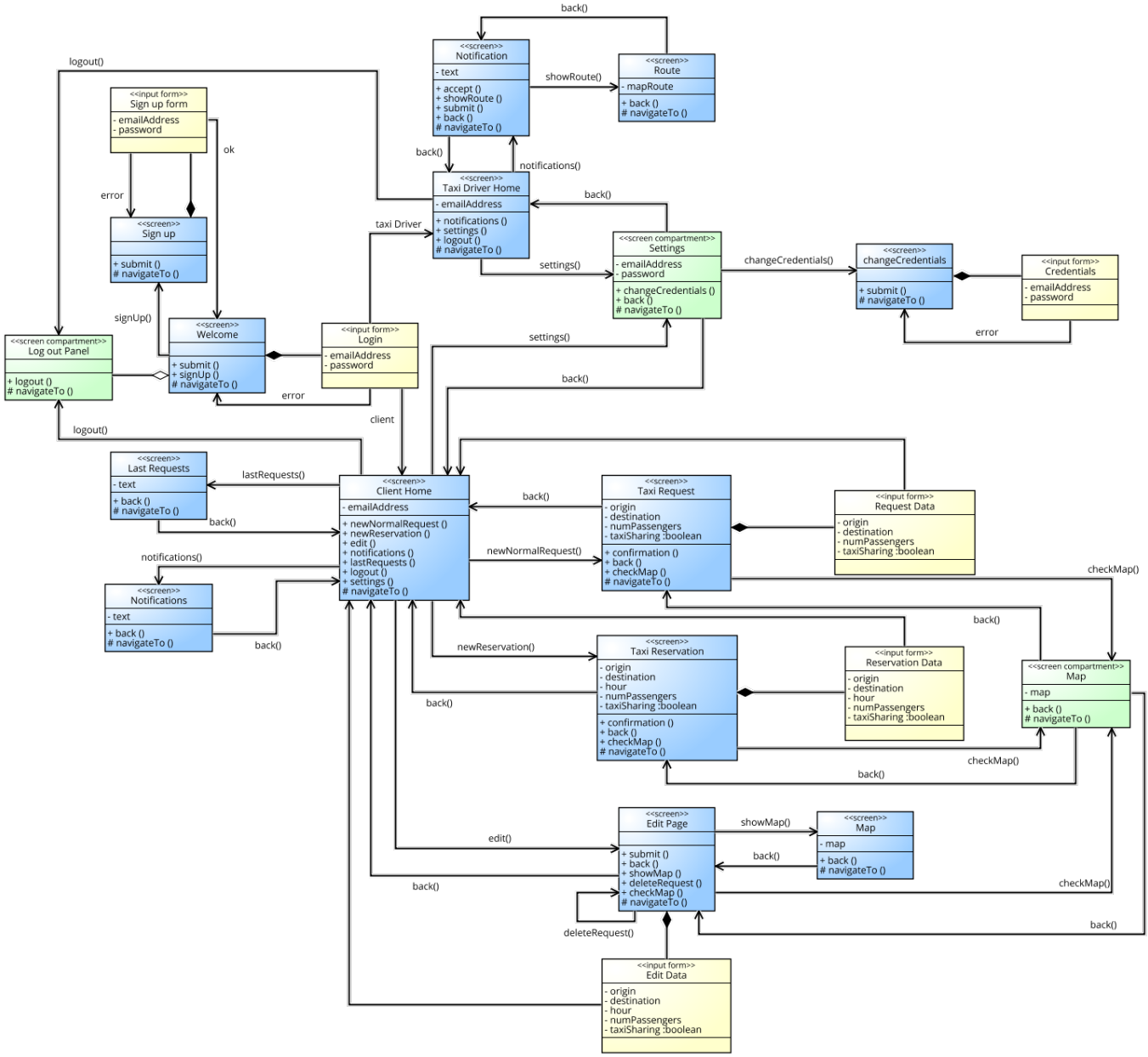
back()
<<screen>>
Notification
- text
+ accept ()
+ showRoute ()
+ submit ()
+ back ()
# navigateTo ()
showRoute()
<<screen>>
Route
- mapRoute
+ back ()
# navigateTo ()
logout()
<<input form>>
Sign up form
- emailAddress
- password
ok
error
back()
notifications()
<<screen>>
Taxi Driver Home
- emailAddress
+ notifications ()
+ settings ()
+ logout ()
# navigateTo ()
back()
taxi Driver
<<screen>>
Sign up
+ submit ()
# navigateTo ()
<<screen compartment>>
Settings
- emailAddress
- password
+ changeCredentials ()
+ back ()
# navigateTo ()
settings()
changeCredentials()
<<screen>>
changeCredentials
+ submit ()
# navigateTo ()
<<input form>>
Credentials
- emailAddress
- password
error
signUp()
<<screen>>
Welcome
+ submit ()
+ signUp ()
# navigateTo ()
<<input form>>
Login
- emailAddress
- password
<<screen compartment>>
Log out Panel
+ logout ()
# navigateTo ()
error
settings()
back()
client
logout()
<<screen>>
Last Requests
- text
+ back ()
# navigateTo ()
lastRequests()
back()
<<screen>>
Client Home
- emailAddress
+ newNormalRequest ()
+ newReservation ()
+ edit ()
+ notifications ()
+ lastRequests ()
+ logout ()
+ settings ()
# navigateTo ()
back()
<<screen>>
Taxi Request
- origin
- destination
- numPassengers
- taxiSharing :boolean
+ confirmation ()
+ back ()
+ checkMap ()
# navigateTo ()
<<input form>>
Request Data
- origin
- destination
- numPassengers
- taxiSharing :boolean
checkMap()
notifications()
newNormalRequest()
<<screen>>
Notifications
- text
+ back ()
# navigateTo ()
back()
back()
newReservation()
<<screen>>
Taxi Reservation
- origin
- destination
- hour
- numPassengers
- taxiSharing :boolean
+ confirmation ()
+ back ()
+ checkMap ()
# navigateTo ()
<<input form>>
Reservation Data
- origin
- destination
- hour
- numPassengers
- taxiSharing :boolean
<<screen compartment>>
Map
- map
+ back ()
# navigateTo ()
back()
checkMap()
back()
<<screen>>
Edit Page
+ submit ()
+ back ()
+ showMap ()
+ deleteRequest ()
+ checkMap ()
# navigateTo ()
showMap()
edit()
<<screen>>
Map
- map
+ back ()
# navigateTo ()
back()
back()
checkMap()
deleteRequest()
back()
<<input form>>
Edit Data
- origin
- destination
- hour
- numPassengers
- taxiSharing :boolean

Figure 4.1

5. Requirements traceability

The decisions that we have adopted in the DD are based on the functional requirements that we have already presented in the RASD document, as you can see in the following table.

Section	Functional Requirements
Chapter 4, Subsection 0	Guest: • Sign up
Chapter 4, Subsection 1	Client: • Log out; • Modify his credentials; • Asking a normal taxi request; • Requesting a taxi; • Editing taxi requests; • Receive notifications;
Chapter 4, Subsection 2	Taxi Driver: • Log out; • Modify his credentials; • Manage clients' requests
Chapter 3, Subsection 1,2,3	Dispatcher System: • Calculation of the costs for each passenger; • Calculation of the best route; • Management of the taxi queue

6. RASD Modifications

Here are explained the modifications that we have added into the RASD document:

- In the assumptions, we have specified that taxi drivers always answer instantly to a system' notifications;
- Into functional requirements section we have changed the clients' and taxi drivers' requirement "modify his profile information" into "modify his credentials";
- Into functional requirements section we have deleted the "recover password" option;
- Into functional requirements section we have added the managing of taxi queues as a system requirement, although we have already worked assuming it.
- We have moved the working hour into a dedicated section.

7. Used Tools

We have used the following programs in order to write this document.

- Microsoft Word 2016 for Mac and Windows: to write and format the document;
- Signavio Academic: to create the class diagram and use case diagrams;
- Visual Paradigm 12.2: to create diagrams in the document.

7. References

In order to write this document, we have consulted the following bibliography.

- Sistemi informative basati sul web – B.Pernici;
- The Google Maps Directions API - https://developers.google.com/maps/documentation/directions/intro#traffic-model
- IEEE Standard For Requirement Specification.

8. Hours of work

Each component of the group has worked the following number of hours in order to write this document:

- Elis Bardhi: 20 hours;
- Andrea Cavalli: 19 hours;
- Mario Dolci: 12 hours.