POLITECNICO
MILANO 1863

Computer Science and Engineering

A.A. 2016/2017

Software Engineering 2 Project:

"PowerEnJoy"

Design **D**ocument

December 11, 2016

Prof.Luca Mottola

Matteo Michele Piazzolla Matr. 878554

Andrea Millimaggi Matr. 876062

Revision History

Revision	**Date**	**Author(s)**	**Description**
1.0	13/11/16	Piazzolla Millimaggi	First document issue

Contents

List of Figures

1 INTRODUCTION

1.1 Purpose

The Software Design Document is a document whose purpose is to provide documentation which will be used to guide the software development, providing all the detailed information for how the software should be built.
This document consists of narrative and graphical documentation of the software design for the project including UML diagrams, object behaviour models, architecture description.

1.2 Scope

The Software design document would demonstrate how the design will accomplish the functional and non-functional requirements captured in the RASD for the PowerEnJoy service. The document describes the high level components and architecture, sub systems, interfaces, database design and algorithm design. This is achieved through the use of architectural patterns, design patterns, and user interfaces.

1.3 Definitions, Acronyms, Abbreviations

- **RASD:** Requirement Analysis and Specification Document
- **ECU:** Electronic Control Unit.
- **REST:** REpresentational state transfer, is a way of providing interoperability between computer systems on the Internet.
- **CRUD:** Create, Read, Update and Delete.
- **JSON:** JavaScript Object Notation, is serialization of a JavaScript data object.
- **JSP:** JavaServer Pages, a presentation technology for web pages
- **JEE:** Java Enterprise Edition, a development platform for the development of enterprise applications

1.4 Reference Documents

- Project Assignment Document: Assignments AA 2016-2017.pdf
- Requirement Analysis Specification Document: RASD.pdf

1.5 Document Structure

In the first part of the document there is a detailed description of the architecture designed for PowerEnjoy service from the logical and physical prospect. All the subsection are a different point of view of the system. In the second part of the document the focus is on the explanation of some of the algorithms used on the service. In the last section there is a Requirements Traceability that link this document to all the RASD requirements.

2 ARCHITECTURAL DESIGN

2.1 Overview: High level components and their interaction

In this section is provided a complete overview of all the system components, from the logical level to the physical level of the PowerEnJoy service. First of all we give a description of the high level components and their interaction

From an high level prospective the service is composed by different modules, that are:

- **Database:** the data layer of the service. All the persistent data will be stored in this layer.
- **Application Server:** this component will manage the Business logic of the system.
- **HTTP Server:** this layer is the interface with the world, so with che Clients. It will manage the requests coming from both application and website.
- **Mobile Application:** this is the interface between the Mobile User and the system. The user will use the application to send requests to the web Server and see the results sent back through his mobile device.
- **Web Site:** this is the interface between the Web User and the system.The user will use it to send requests to the Web Server to get static contents and send requests to the Application Server to use the service.
- **Car OnBoard Device:** this is the module mounted on the car. It will communicate with the Application server and with the car ECU.

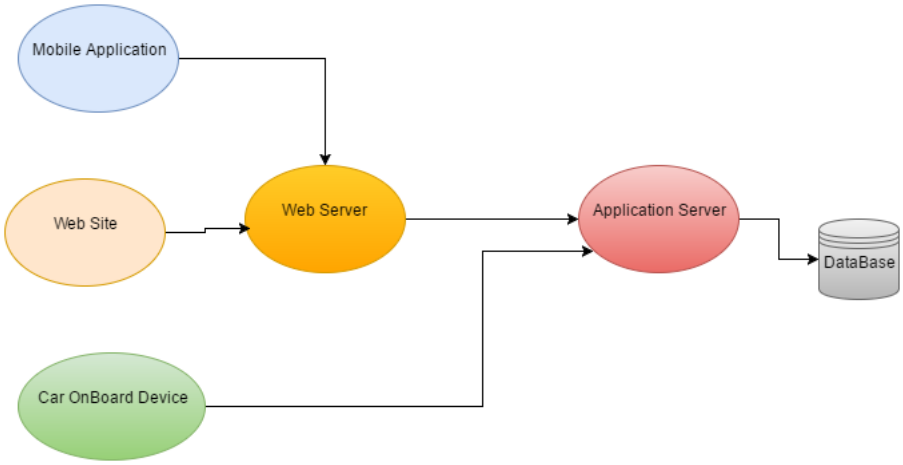

Figure 1: High Level Architecture Diagram

2.2 Component view

From a lower level perspective, the components of the system can be divided in 2 subsystems: the Frontend system and the Backend system

2.2.1 Frontend

In the Frontend there are all those components that will work as interfaces between the user and system and will allow the user to use the functions of the service. So the Frontend is made by three GUIs:

- Web Client GUI: It is the graphic interface used by the user that access the service through the Website
- Mobile Client GUI: It is the graphic interface used by the user that access the service through the Mobile Application.
- Driver Client GUI: It is the graphic interface used by the user when he interact with the system using the OnBoard Device.

2.2.2 Backend

In the Backend there all those component that aren't used directly by the user, but they perform operations that allow to satisfy the requests of the users. The components of the Backend are:

- Data Manager: This component is the one that can access the database. It has the function of inserting, deleting and updating data in the database, when asked by other components. Furthermore, it provides to other components the data needed for their operations.
- User Manager: This component is responsible of managing all the operations concerning the accounts, such as registration, login and logout, modification of personal data.
- Position Manager: This component has the task of managing the position of the cars provided by the GPS and provide it to the other components that need it.
- Travel Manager: This component is responsible of managing all the information about the travel, such as the current current position of the car, the safe areas and the special parking area. It has to communicate all this information to the Driver Client GUI.
- Fee Manager: This component is responsible for the calculation and the application of discounts on the fee of the travels has the task of charging the user. For these reason it is interfaced with an external payment system.
- Reservation Manager: This component is responsible of keeping track the reservations that are still active and of the available cars that can be picked up by the user.

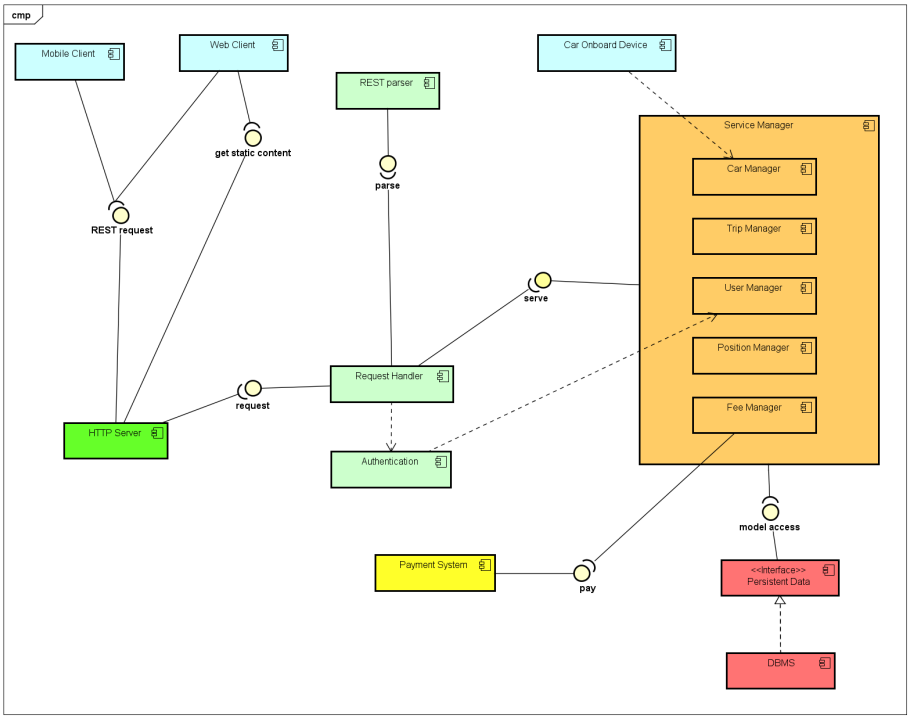

Figure 2: Component Diagram

The communication between the clients (both mobile application and web site) are realized using a request-response methodology. The client asks for information or to make actions related to the server using **REST** requests and obtains a response from the server formatted as a **JSON** object, today the state of the art for web services communications. All the REST requests received by the **HTTP Server** are forward to and, then, processed by the Request Handler component. The Request Handler component is in charge of validating the request integrity, parsing it using the **REST Parser**component and check the authentication for the requests type that need a valid session.

When a requests pass these checks is forwarded to the Service Manager where the application logic is applied. The **Service Manager**, depending on the types of request that are received, processes a response using Car Manager, User Manager, Position Manager, Trip Manager or Fee Manager. Using those component the Service Manager can access the model, manage the request and craft a response.Every request can instantiate different component,

The system status are represented through various **Java Beans** object that stay alive for all the time needed for the request processing.

When a request is received all the element from the model needed for the processing are loaded from the persistent data storage through the **Persistent Data interface**. The abstraction exposed by the Persistent Data interface

allow to adopt different data storage solution in a transparent way to the various component that are using it. All the persistent data are stored through a distributed database that can optimize data access.

2.3 Deployment view

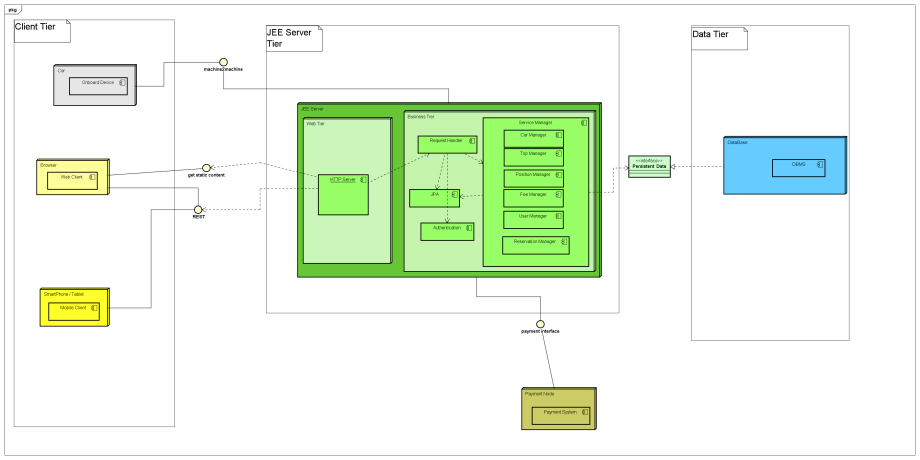

Figure 3: Deployment Diagram

From the deployment perspective, the components described in the Component view will be installed on 3 different machines:

- **JEE Server**: On this machine will be installed a Glassfish server, that is an oper source application server supporting JEE. This Server will contain all the components handling the business and the web logic.
- **Database**: On this machine will be stored all the persistent data. On the database will be installed the latest version (5.6) of MySQL, a relational DBMS distributed by Oracle, that will handle the creation, deleting and updating of the tables of the databases containing the data.
- **Client**: This machine is the one that the user will use to use the Power EnJoy service. It can be a Mobile Device, so a SmartPhone or a Tablet, a PC or the OnBoard Device. On the Mobile Device will run the Mobile Application. If the user uses a PC instead, he will access the service using the WebSite, that runs on any kind of Browser. Lastly, when the user is driving, the Client machine will be the OnBoard Device, that is the device mounted on the Car, on which will run a proprietary software of Power EnJoy.

2.4 Runtime view

In this section we show the runtime behaviour of the system in various situations, in order to show which components have to act and how they interact.

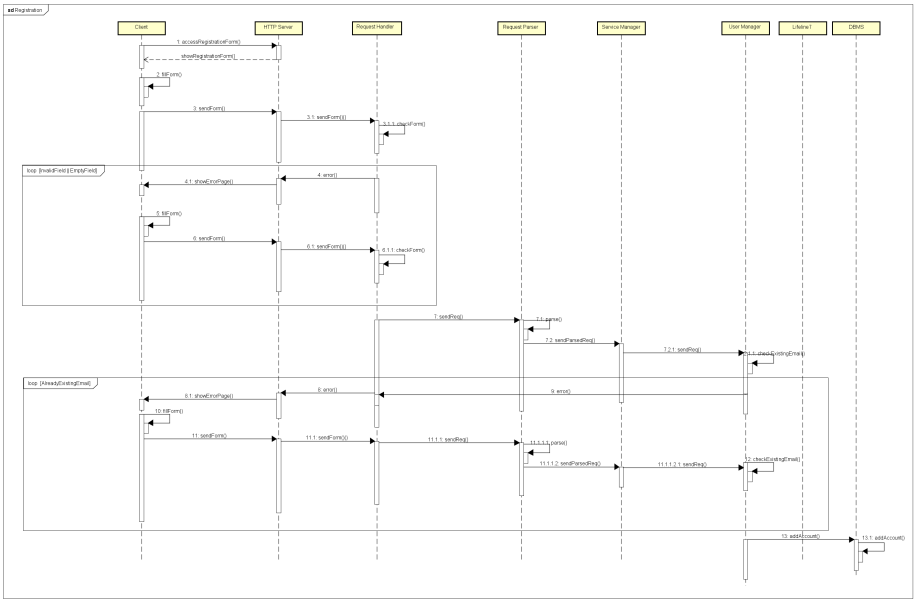

Figure 4: Sequence Diagram of the Registration process

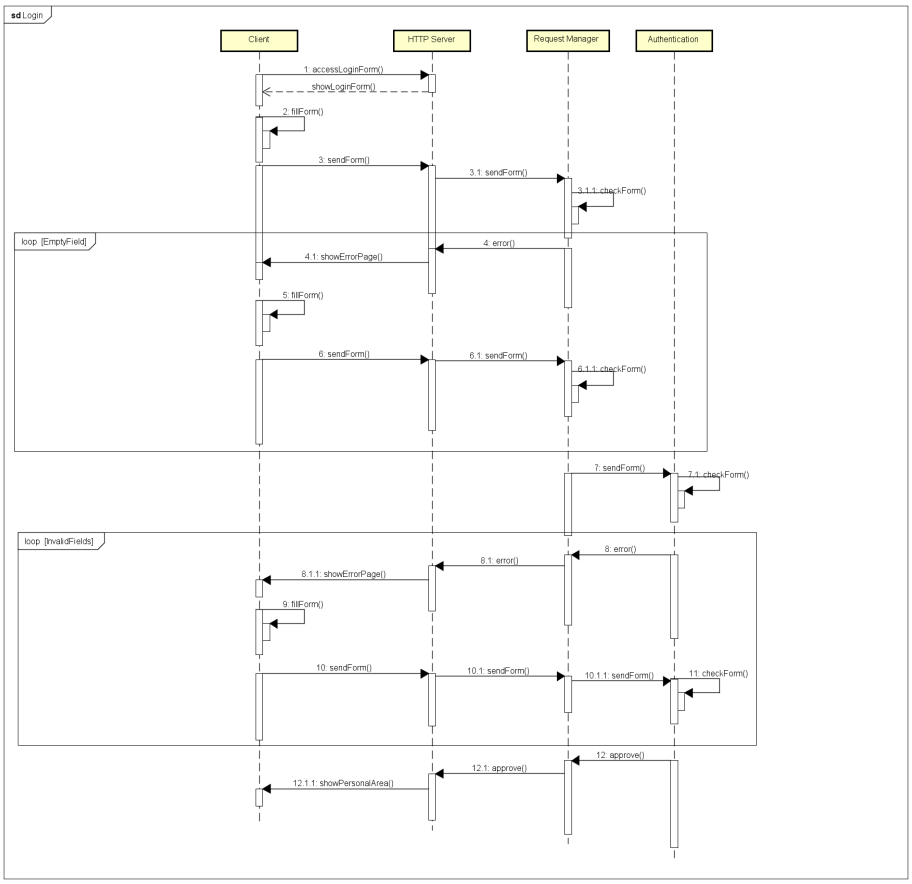

Figure 5: Sequence Diagram of the Login process

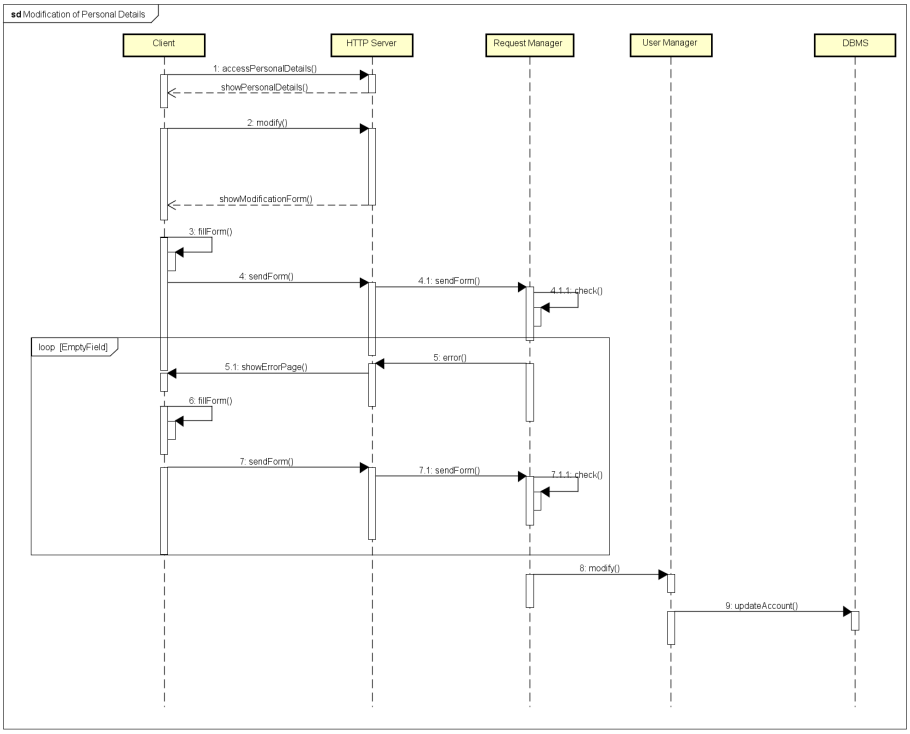

Figure 6: Sequence Diagram of the Modification of Personal Details process

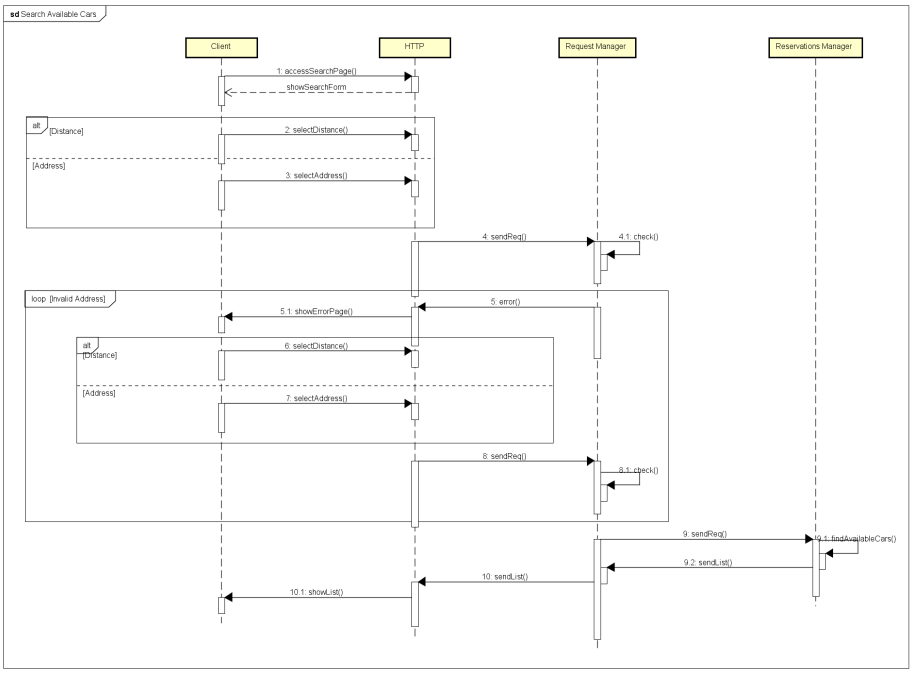

Figure 7: Sequence Diagram of the Search of Available Cars process

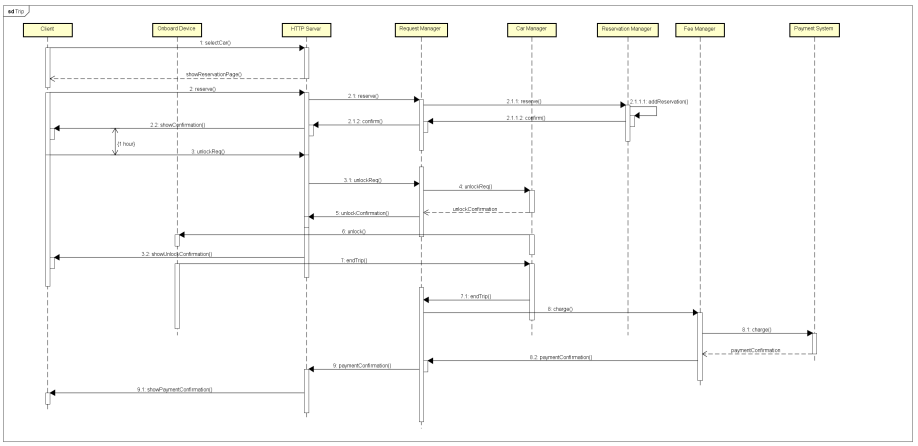

Figure 8: Sequence Diagram of the Trip process

2.5 Component interfaces

2.5.1 REST

As explained in the introduction the communication takes place via REST requests. The REST model establish a relation map between the common CRUD operations (Create, Read, Update and Delete of a resource) and the HTTP operations as explained in the following table.

CRUD	REST Http	example	description
Create	POST/PUT	POST /user/register	Create a resource
Read	GET	GET /cars/zone/12	Retrieve information. Requests must be safe and idempotent.
Update	PUT	PUT /user/gpsposition	Update an existing entity. Request is idempotent.
Delete	DELETE	DELETE /car/request/28	Remove a resource.

Table 2: Rest/CRUD interface mapping.

2.5.2 Database Connectivity

JDBC (Java Database Connectivity) is used as connector between Database and the Application Server. JDBC, in fact, is a software component that allows Java applications to interact with databases. It has been chosen because have support for almost every **DBMS** on the market. To enhance the connection, JDBC requires drivers for each database. These drivers connect to the database and implement the protocol to query and get respective results between the client and database.

ODBC driver is a level of abstraction that allows programmers to make SQL requests to access data from distinct Database without having to know the proprietary interfaces of each DB. In this way it is easier to change the database without altering the application layer.

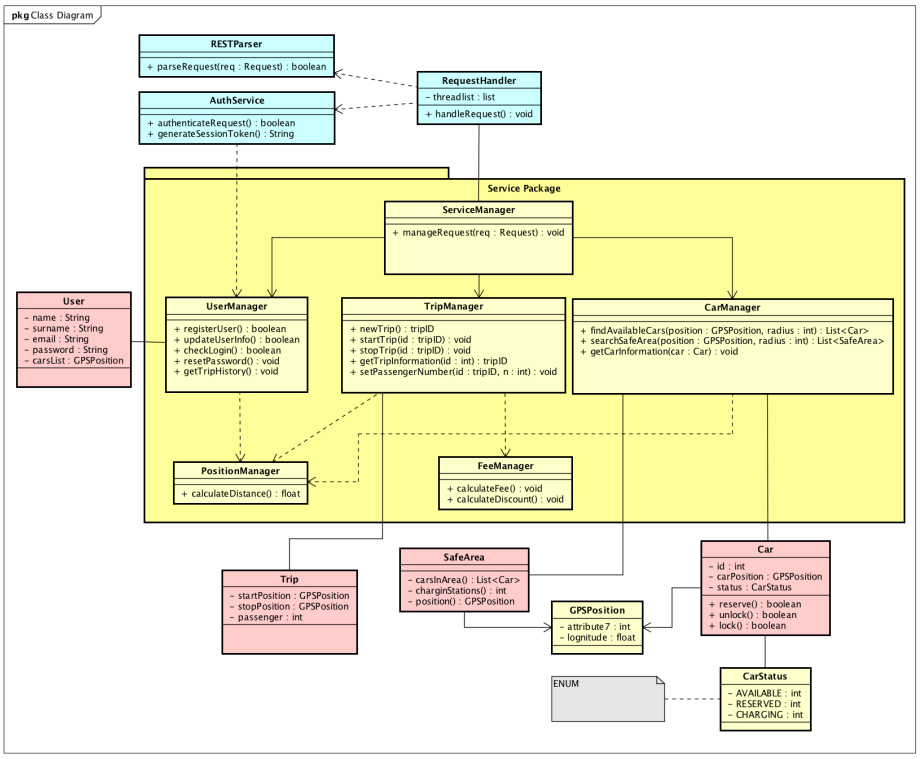

Figure 9: Class Diagram

2.5.3 Backend Component Interface Description

- **ServiceManager:** The Service Manager is instantiated when a new request has been parsed and validate.
  - *manageRequest(Request)*: Receive as input a new request and instantiate the right component to manage it.
- **UserManager:** The User Manager is instantiate when ad User action or an User information are required.
  - *registerUser()*: Check if provided informations are correct and create a new User entity in the model.
  - *updateUserInfo()*: Update the selected User entity in the model.
  - *checkLogin()*: Check if provided informations are correct and allow to create a new Session.
  - *resetPassword()*: Send to the User email a link to reset the account password.
  - *getTripHistory()*: Get a list of the User's past trip.

- **TripManager:** The Trip Manager is instantiate when the User start or end a trip.
  - *newTrip()*: Create a new Trip instance in the Model. Return a tripID.
  - *startTrip(tripID)*: Set the Trip starting position.
  - *stopTrip(tripID)*: Set the Trip ending position.
  - *getTripInformation(tripID)*: Return the information about a specific Trip (past or current).
  - *setPassengerNumber(tripID, int n)*: Set the passengers number for a specific Trip.
- **CarManager:** The Car Manager is instantiate when to manage cars request or get a specific car status.
  - *findAvailableCars(GPSPosition,radius)*: Return a set of cars in a given radius centered on a GPS position.
  - *searchSafeArea(GPSPosition,radius)*: Return a set of SafeArea given radius centered on a GPS position.
  - *getCarInformation(Car)*: Return the status of a specific car (example the charge of the battery).
- **PositionManager:** The Position Manager is used when a calculation between GPS Coordinate is needed.
  - *calculateDistance(GPSPosition,GPSPosition)*: Return a cartesian distance between two coordinate.
- **FeeManager:** The Fee Manager is instantiate to calculate the amount of money that need to charge to the user.
  - *calculateFee(tripID)*: Calculate the normal fee for the trip in base of the time and distance traveled.
  - *calculateDiscount(tripID)*: Apply a discount in base of starting/ending position and state of the car after the travel and the number of passenger.
- **RequestHandler:** The Request Handler select a free thread from an allocated pool to handle an HTTP request.
  - *handleRequest()*: Use the REST parser and the AuthService to validate a request.
- **AuthService:** The Authentication Service check if an User session for the request is valid.
  - *authenticateRequest()*: Check if a valid session token is provided for the request.

  - *generateSessionToken()*: Generate a session token to allow user to authenticate the requests.

- **RESTParser:** The REST Parser check te format of the REST request and discard a request if invalid
  - *parseRequest()*: Check if the request is valid.

2.6 Selected architectural styles and patterns

2.6.1 Architecture

We realized a 3-tiered architecture. In particular there are 3 dedicated machines for:

- Data Layer
- Presentation Layer
- Web Server and Application Server

The reader can see the organization of Tiers in details in 2.3

The reasons why we made this choice concern:

- **Security:** Is possible to setup a DMZ for the webserver and filter malicious requests before they reach the application server.
- **Scalability:** Because of the separation of the layers, developers can modify existing functions or introduce new ones in any level (presentation, logic or data), leaving the other layers untouched.
- **Performance:** The webserver do a massive use of cache and load balancer to serve only static content to minimize resource and traffic
- **Reliability and Available:** because of the partition of tasks in distinct machines, it's easier to check for failures and fix them. So the time the system is down for maintainance is reduced and, as conseguence, the time the system is operating properly and the probability to find a system ready to accomplish users requests is increased. Furthermore, reliability and availability can be further improved further by replicating the servers, so that a failure in a single server doesn't affect the operativity of the system, but only temporary reduces performances.

2.6.2 Patterns

- **Client-Server:** According to this pattern, in the system there should be 2 entities: a Client, that sends requests, and a Server, that elaborates requests and sends back results to the client. In our architecture there 3 Clients: the Mobile Client, the Web Client and the Driver Client. These all three clients send requests to the ~~Web Server~~HTTP server on the Application server, that takes care of forward it to the Request Handler and send back and show to the clients the results.
- **Model-View-Controller:** In the MVC pattern there are 3 entities: The Model, that contains methods to access the data, the View, that shows data to the user and allow him to perform actions, and the Controller, that

elaborates the commands sended by the view manipulating and modifying the data of the model. In our system the View is the Client, the Model is the Data Manager and the Controller is the Service Manager. Furthermore, between the View and the Controller we have put 2 middle components, to improve the quality and the performance of the system. The Web Server act as a request receiver and sends the commands of the View to the Request Handler. The Request Handler act as validator and activates the forwarding of the command to the Controller only if the request respects integrity and authentication constraints.

- **Resource Injection:** Because of the amount of users , it is useful to have a mechanism that can inject objects in an application at runtime, so that the application hasn't to care about the creation of object everytime he needs it. So we decided to use JNDI service to name and lookup for resources and the Resource Injection mechanism provide by JEE, that allows to inject any resource in the JNDI into an object.

3 ALGORITHM DESIGN

In this section we present some algorithm that we consider relevant for the work of the developers.

3.1 Find cars near the user

Both Users and Cars's positions are identified in the system by GPS coordinates (Latitude and Longitude).
When a User u ask for a list of cars the application send to the server his GPS position.

$$\vec{u_p} = \begin{bmatrix} user\_latitude \\ user\_longitude \end{bmatrix} \tag{1}$$

The shortest distance (the geodesic) between two given points $P1 = (lat1, lon1)$ and $P2 = (lat2, lon2)$ on the surface of a sphere with radius R is the great circle distance.

It can be calculated using the following formula:

$$D = \arccos\Big(\sin(lat1)\cdot\sin(lat2) + \cos(lat1)\cdot\cos(lat2)\cdot\cos(lon1 - lon2)\Big)\cdot R \tag{2}$$

The first step is to convert latitude and longitude in radians:

$$lat_{radians} = lat_{degree}\cdot\frac{\pi}{180} \tag{3}$$

In order to utilize indices on Lat and Lon we can use a method like the bounding rectangle method in Cartesian space.
We need to define r as the angular radius of a circle in which our cars coordinate will be using R as the Earth radius (6371 km) and d as the maximum distance selected by the user.

$$r = \frac{d}{R} \tag{4}$$

for example

$$\frac{5km}{6371km} = 0.000784806$$

Now we need to compute the Minimum and the Maximum Latitude and Lon-

gitude:

$$lat_{min} = lat - r \tag{5}$$
$$lat_{max} = lat + r \tag{6}$$
$$\Delta lon = \arcsin(\frac{\sin(r)}{\cos(lat)}) \tag{7}$$
$$lat_T = \arcsin(\frac{\sin(lat)}{\cos(r)}) \tag{8}$$
$$lon_{min} = lon_{T1} = lon - \Delta lon \tag{9}$$
$$lon_{max} = lon_{T2} = lon + \Delta lon \tag{10}$$

After that we have computed the bounding coordinates we can use them to query a DB.

```
SELECT * FROM Places
WHERE
(Lat => $lat_min AND Lat <= $lat_max) AND (Lon >=
   $lon_min AND Lon <= $lon_max )
HAVING
acos(sin($user_lat) * sin(Lat) + cos($user_lat) *
   cos(Lat) * cos(Lon - ($user_lon))) <= r;
```

The Java-like code for the managing of the search by the Request Handler:

```
public List search(String address, Position pos){
        List result = new List();

        if(address != null){
        Position position =
           extractCoordinates(address);
        result =
           reservationManager.getAvailableCars(position);
        }

        else{
        result =
           reservationManager.getAvailableCars(pos);
        }

        return result;
}
```

3.2 Reservation

This is a Java-like code for the algorithm implemented in the Reservation Manager for the addition of a reservation and the management of the 1 hour timer of the reservation:

```
public List reserve(Car c, User u,  String address,
   Position pos){

 availableCars.delete(c);

 reservations.add(new Reservation(u,c);

 LocalTime begin = LocalTime.now()
 int beginHour = begin.getHour();
 int BeginMin = begin.getMinute();
 boolean ko = false;

 while(!ko){
        int hourDiff = LocalTime.now().getHour() -
           beginHour;

        if(hourDiff == 1){
                int minuteDiff=
                   LocalTime.now().getMinute() -
                   beginMin;
                if(minuteDiff > 0) {
                ko = true;
                }
        }
 }

notifyOberservers(``time_expired'');
}



annullaResSeE'FinitoIlTempo();

}
```

3.3 Payment, Discounts and Penalties

This is a Java-like code for the algorithm implemented in the Fee Manager for the payment and the application of discounts and penalties:

```
public void getMoney(Travel t, int
    distanceFromStation){

        //the parameter distanceFromStation is the
            distance in km from the parking position
            to the nearest charging station,
            calculated by the Trip Manager

        int discount = 0;
        int penalty = 0;
        int fee = t.getFee();

        if(t.passengers >= 2){
        discount = discount + 10;
        }

        if(t.car.batteryLevel > 50){
        discount = discount + 20;
        }

        if(t.car.isPlugged){
        discount = discount + 20;
        }

        if(distanceFromStation > 3){
        penalty = penalty + 30;
        }

        fee = fee - ((fee*discount)/100) +
            ((fee*penalty)/100);

        pay(t.user.getpaymentMethod(), fee); //method
            that will send a request for the payment
            to the payment node, according to payment
            method of the user


}
```

4 USER INTERFACE DESIGN

4.1 UX Diagram

In this section we provide a description of how the User Interface should look like through a UX diagram:

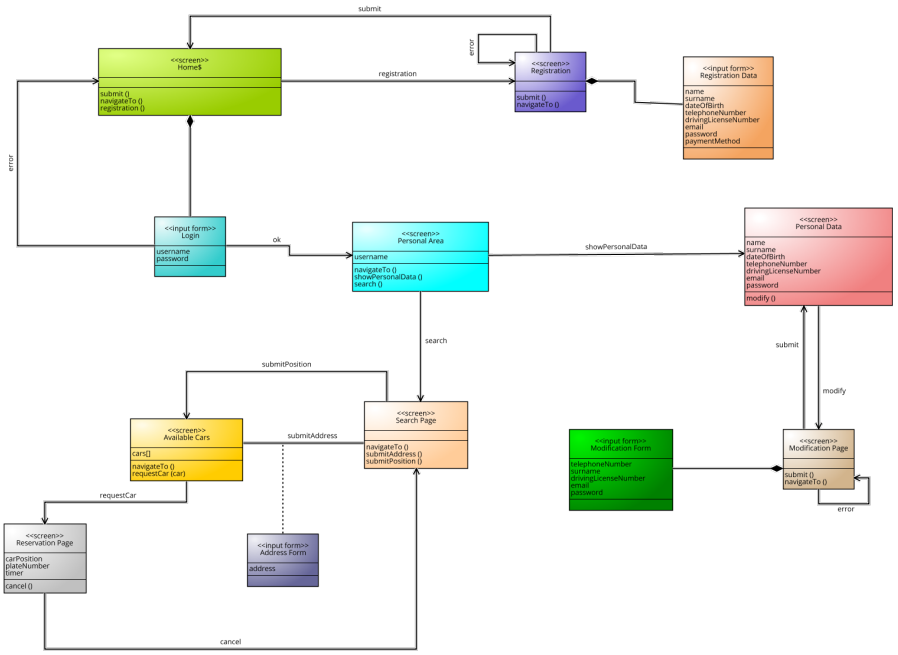

Figure 10: UX Diagram

5 REQUIREMENTS TRACEABILITY

[R-2], [R-3], [R-4], [R-5], [R-6], [R-7], [R-9], [R-12], [R-13], [R-16], [R-18]	Website and Mobile Application
[R-15], [R-21], [R-23], [R-24]	ServiceManager
[R-1], [R-6], [R-8],[R-9], [R-10],[R-14]	UserManager
[R-19], [R-30], [R-34], [R-37],[R-38]	TripManager
[R-16], [R-17], [R-21], [R-27], [R-28], [R-32]	CarManager
[R-20], [R-26] , [R-31], [R-33], [R-35]	FeeManager
[R-17], [R-22], [R-24], [R-29], [R-36]	PositionManager
[R-7],[R-11], [R-25]	RequestHandler
[R-6], [R-8], [R-9],	AuthService

6 Appendix

6.1 Used software

1. **TeXstudio:** `http://www.texstudio.org/` to redact this document in LaTeX format.
2. **Astah:** `http://astah.net` to draw all the UML diagrams presented in this document.
3. **Signavio:** `http://academic.signavio.com/` to draw the UX diagram.
4. **draw.io:** `https://www.draw.io` Online diagram software
5. **Balsamiq Mockups 3:** `https://balsamiq.com` to draw both mobile and web mockups of the service.
6. **Alloy Analyzer:** `http://alloy.mit.edu/alloy/`to test the world model in compliance with the requirements.

6.2 Time effort

List of estimated hours spent day by day to redact this document by each components of the group:

18/11/16	Matteo Michele Piazzolla 3h Andrea Millimaggi 2h
21/11/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 2h
23/11/16	Matteo Michele Piazzolla 1h Andrea Millimaggi 1h
24/11/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 3h
25/11/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 2h
26/11/16	Matteo Michele Piazzolla 1h Andrea Millimaggi 1h
29/11/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 2h
30/11/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 3h
2/12/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 1h
3/12/16	Matteo Michele Piazzolla 1h Andrea Millimaggi 1h
4/12/16	Matteo Michele Piazzolla 1h Andrea Millimaggi 1h
5/12/16	Matteo Michele Piazzolla 1h Andrea Millimaggi 1h
6/12/16	Matteo Michele Piazzolla 3h Andrea Millimaggi 3h
8/12/16	Matteo Michele Piazzolla 2h Andrea Millimaggi 3h
3/12/16	Matteo Michele Piazzolla 1h Andrea Millimaggi 1h