POLITECNICO
MILANO 1863

Software Engineering 2 project: *PowerEnJoy*

A.A. 2016/2017 - Professor E. di Nitto

Design Document

Version **1.0** –11/12/2016

Pietro Avolio Mat 878640
Guido Borrelli Mat 874451

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, acronyms and abbreviations . . . 3
- 1.4 Reference Documents . . . 3
- 1.5 Document Structure . . . 4

2 Architectural Design **5**
- 2.1 High level components and their interaction . . . 5
- 2.2 Component view . . . 6
  - 2.2.1 Relational Database . . . 6
  - 2.2.2 NoSQL Database . . . 6
  - 2.2.3 Application Server . . . 7
  - 2.2.4 Web server . . . 10
  - 2.2.5 Clients . . . 11
  - 2.2.6 External components . . . 11
- 2.3 Deployment view . . . 13
- 2.4 Runtime view . . . 13
- 2.5 Component interfaces . . . 17
  - 2.5.1 Application server to database . . . 17
  - 2.5.2 Application server to clients (REST APIs) . . . 17
  - 2.5.3 Application server to vehicles . . . 21
  - 2.5.4 Web server to clients . . . 22
  - 2.5.5 Application server to external components . . . 22
- 2.6 Selected architectural styles and patterns . . . 22
  - 2.6.1 Passwords and other sensible data . . . 23
  - 2.6.2 Maps . . . 23
  - 2.6.3 Communication with vehicles . . . 23
  - 2.6.4 Reliability and availability . . . 23

3 Algorithm Design **23**
- 3.1 New reservation . . . 23
- 3.2 Unlock Doors . . . 24
- 3.3 Rental ended . . . 25
- 3.4 Fee calculation . . . 26

4 User Interfaces Design **26**
- 4.1 PowerEnJoy Mobile Application . . . 27
- 4.2 PowerEnJoy Control Room . . . 29

5 Requirements Traceability **29**
- 5.1 Functional requirements . . . 29

6 Appendix **31**
- 6.1 Tools used . . . 31
- 6.2 Effort spent . . . 31

1 Introduction

1.1 Purpose

The purpose of this Design Document is to provide a more detailed description of the PowerEnJoy system architecture, to describe main architectural components, their interfaces and interactions. Two sections are also dedicated to algorithms and user interfaces.

This document is thus intended for project manages, developers and testers and it can be considered as a good overview to base the development and the maintenance on.

1.2 Scope

As stated in the RASD document, the PowerEnJoy system is made of three main components: two very thin clients (PowerEnJoy Mobile Application and PowerEnJoy Control Room) and one application server (PowerEnJoy Core).

In this document, we will describe how this components create a four layers architecture, how they interact and communicate one with each other. We will also provide a physical deployment structure.

For the sake of coherence and consistency, we will provide a strong cross-reference to the RASD document, of which this Design Document is a natural follow-up and deepening.

1.3 Definitions, acronyms and abbreviations

In order to avoid ambiguity and possible misunderstandings, this table is an integration to the definitions, the acronyms and the abbreviations defined in the RASD document.

Relational Data	Data structured according to the relational model
NoSQL Data	Data not structured according to the relational model
RDBMS	Relational Database Management Server
NSDBMS	NoSQL Database Management Server
Java EE	Java Enterprise Edition
EJB	Enterprise JavaBeans
JPA	Java Persistence API

1.4 Reference Documents

- Design Document of previous year published on Beep.
- Software Design & Software Architecture slides on Beep
- Architectural Styles on Beep
- Software Architectures in Action on Beep
- Java EE 7 Overview on Beep
- Tips and Trick for Supporting Architecture Description with UML on Beep

1.5 Document Structure

The document is structured in the following chapters:

Introduction This chapter provides general information about this document, its purpose and the intended public.

Architectural Design This chapter provides details about the components of the system, their interactions and interfaces. The dynamic behaviour is provided as well as a focus on design, styles and pattern choices.

Algorithm Design This chapter provides and discusses about the most interesting algorithms to be implemented in the system.

User Interfaces Design This chapter provides some elements mockups for the user interfaces to be developed.

Requirements Traceability This chapter shows how all the functional requirements stated in the RASD document are satisfied by architectural choices.

2 Architectural Design

This section provides a very detailed view over the system architecture and its components, describing them at both the logical and physical level.

Section 2.1 describes high-level components of the system; each of this components will be separately described and detailed in section 2.2; section 2.3 focuses on the deployment of the system on physical tiers; section 2.4 describes the dynamic behaviour of the softwares; section 2.5 focuses on the interface between different components; finally section 2.6 provides a description of the design choices and patterns.

2.1 High level components and their interaction

As stated in the RASD document, the system is composed of three main softwares: two very thin clients, which are PowerEnJoy Mobile Application and PowerEnJoy Control Room, and a server containing all the application logic, which is PowerEnJoy Core. Plus, also company vehicles can be considered as passive clients, since they complete some tasks automatically without human interactions.

The architectural style chosen for the system is a classical client-server architecture, splitted on four logical layers as shown in figure 1.

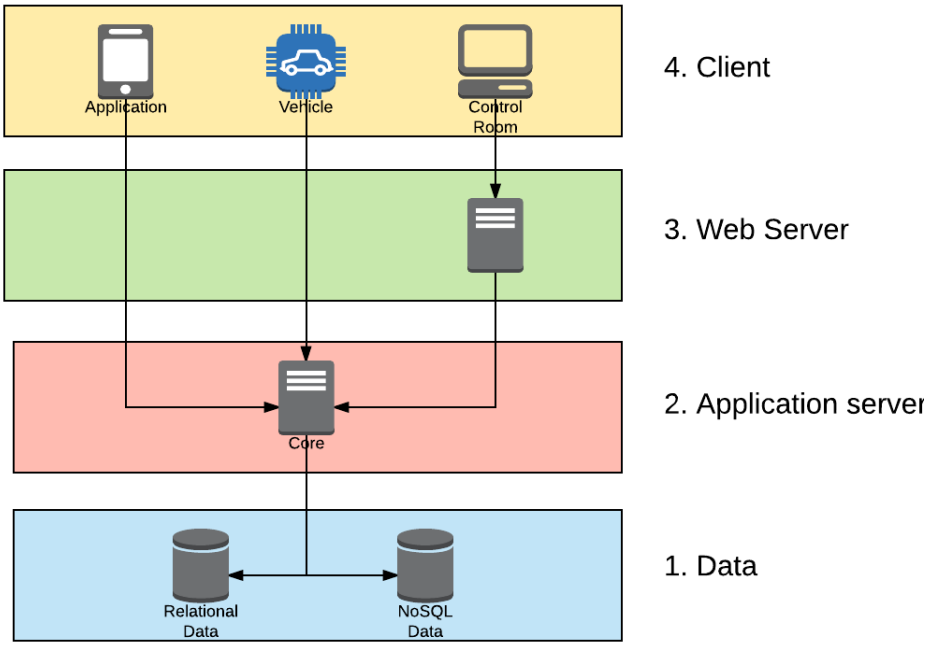

Figure 1: Logical layers

The system also needs to interface with some external components in order to accomplish some tasks. This components are:

- Payment Gateway

- SMS Gateway
- Support Ticket System

A diagram including an high level overview of the system components is shown in figure 2

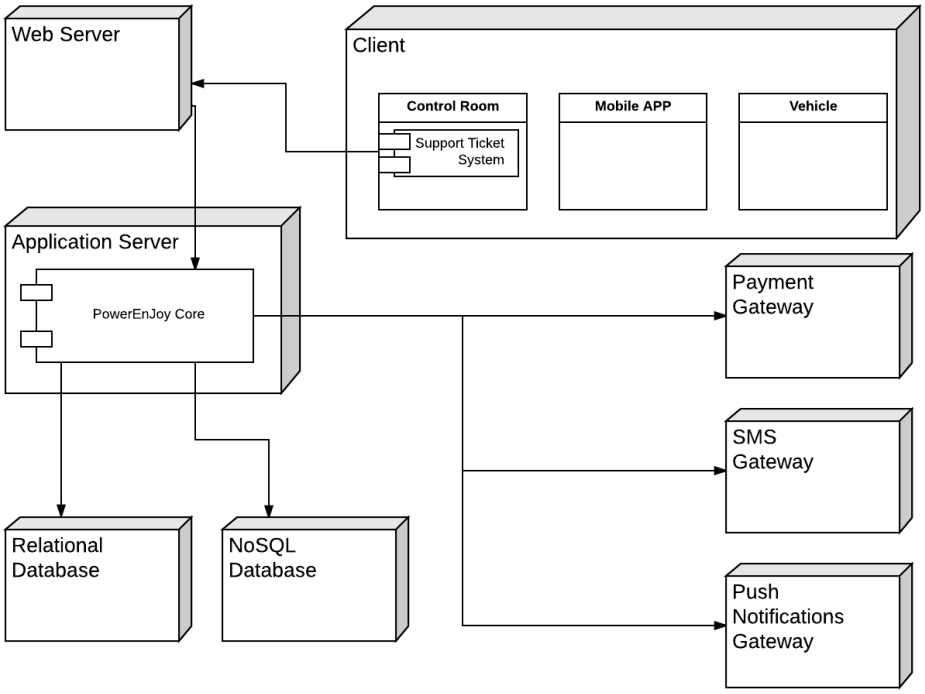

Figure 2: High level components overview

2.2 Component view

2.2.1 Relational Database

The relational database is in charge of storing all the data that well fit the relational model. It runs on a MySQL Community Edition as DBMS and it uses InnoDB as database engine for all the tables. ACID properties must be guaranteed over all the transactions.

The relational database communicates with the business logic tier only: communication goes using the TCP/IP protocol on an arbitrary network port.

The database must be physically protected and duplicated in order to avoid data loss. Communication must be encrypted and different users should be created and assigned to the different softwares that access the database in order to always guarantee the minimum required level of privileges per software.

2.2.2 NoSQL Database

The NoSQL database is in charge of storing all the data generated by the system such as logs and all the kinds of data that will not require any update or delete

during the time and that will be accessed to be consulted only. The decision of using a NoSQL database instead of a traditional relational database is driven by the need of speed in inserting and accessing a big amount of data. For this reason, it's not required to ensure all the ACID properties over transactions.

The NoSQL database runs on MongoDB as DBMS and communicates with the business logic tier using the TCP/IP protocol on an arbitrary port.

All the security concerns stated for the relational database apply to the NoSQL database as well.

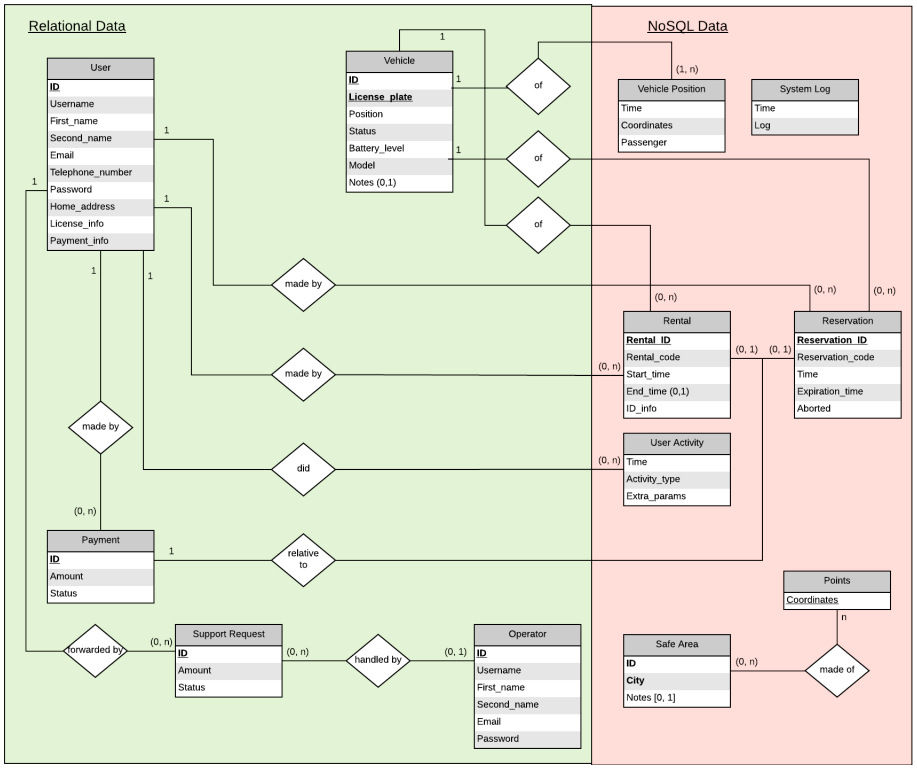

Figure 3: Entity-relational diagram for both the relational and the NoSQL databases

2.2.3 Application Server

The application server is where the PowerEnJoy Core is executed and where all the business logic is. It is implemented using Java EE and it runs on a GlassFish Server.

The business logic is implemented using custom EJBs; the access to the database is implemented using the JPA for both the relational and the NoSQL databases.

In order to expose its functionalities, the PowerEnJoy Core provides RESTful APIs as interface to the clients and to the web server. The detailed behaviour of these interfaces is described in section 2.5.2.

The following list contains all the beans composing the Core application, while

a component diagram is shown in figure 4.

All the beans uses the Log Manager and the REST API Manager, thus there are no shown connections to this beans in the diagram for the sake of readability. All the beans inside the same container can interact one with each other.

User Manager

This bean manages all the features related to users: registration, login, account details update. Furthermore, this bean provides the token used to identify the client once the user logged-in. It validates driving license information and it generates and sends the user's password during the registration process. This bean also handles requests to show available vehicles in a certain area.

Operators Manager

This bean manages all the features related to operators. It mainly communicates with the web-server which hosts the Control Room component and with the Support Requests Handler.

Vehicle Manager

This bean manages all the features and the operations related to vehicles. When the application is started, a stateful bean is instanciated for each company vehicle and it is destroyed only when the application is destroyed. This bean also implements the *hearthbeat* function: it periodically communicates with the vehicle in order to get its information and keep the internal status updated. The update interval is of 30 seconds for rented vehicle, 2 minutes for all the others. It also offers an interface to change the status of the vehicle and it is responsable for sending commands. This bean handles the events sent by vehicles (e.g. when the engine is turned on/off).

Reservations Handler

This bean manages all the aspects relative to reservations: insert, update, delete and expiry. When a reservation is requested over a vehicle it is handled directly by this bean. When a reservation expires, this bean communicates with the Pricing Policies Manager and with the Payment Handler in order to charge the user.

Rentals Handler

Exactly like the Reservations Handler, this bean manages the insertion, the update, the deletion and the expiry of users rentals. It is responsible for handling the user's request to unlock the doors of a vehicle.

Pricing Policies Manager

This bean calculates the final cost of a rental eligible by the company after gathering all the details about the rental. In this beans all the pricing policies stated by the company are applied.

Payment Handler

This bean handles all the payment requests generated by the other beans and it interfaces with the external payment gateway. It also validates payment information when provided by the user during the registration process or when they are updated.

Log Manager

This bean satisfies the need of keeping track of everything by registering each system or user action. It is used by almost all the other beans of the system.

CronTasks Manager

This bean manages all the tasks that must be executed periodically, such as to retrieve the users to notify when their documents are about to expire and it generates internal support requests to be handled by operators.

Support Requests Handler

This bean is in charge of the insert, update, and delete of support requests forwarded by customers or internal system notifications. These request are later handled personally by the company operators.

Email Sender

This bean allows the system to send emails to a specific user or to a group of users.

Push Notifications Sender

This bean allows the system to send push notifications to a specific user or to a group of users. It provides an interface to the Push Notifications Gateway.

SMS Sender

This bean allows the system to send SMS to the users on their mobile phone number. It provides an interface to the SMS Gateway.

REST API Manager

This bean is responsible for all the REST APIs provided by the system. It is used by almost all the other beans. It validates incoming requests and builds responses, in order to have a centralized place in which to define a standard for communication.

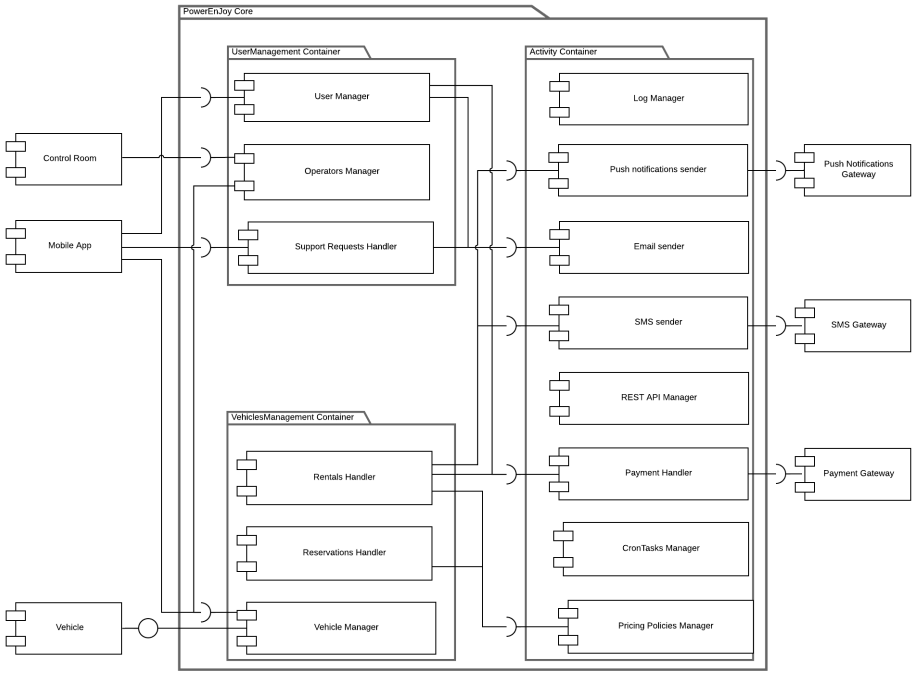

Figure 4: Component diagram of Power Enjoy Core.

2.2.4 Web server

The web server is implemented using Java EE specific HTTP servlet and HTML5, for this reason it runs on a GlassFish Server. The dynamic part of the user interface is realized using jQuery.

This component does not implement any part of the business logic, but it communicates with the application server by using its RESTful APIs. It can be accessed by operators using a standard web browser and the HTTPS protocol.

The web server architecture, as shown in figure 5, is very simple and composed of JavaServer Faces for the view and by a controller which translates user actions into REST requests to the application server.

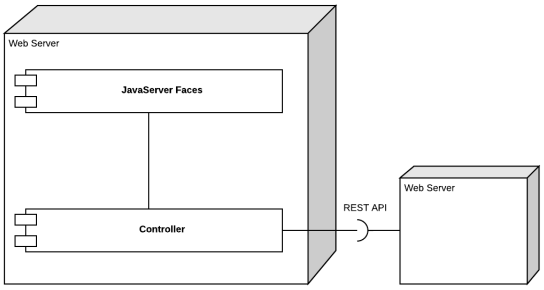

Figure 5: Architecture of the web server

2.2.5 Clients

PowerEnJoy Vehicles

As stated in the RASD document and in the previous sections of this document, we manage PowerEnJoy vehicles as clients. Both the hardware and the software solutions used to implement the information system installed on the vehicles is not part of this project.

The communication with vehicles is implemented using messages and queries: the PowerEnJoy core can query a specific vehicle in order to get information (e.g. position, number of passengers, mechanical status) and send commands (e.g. to unlock doors); vehicles can send messages to the PowerEnJoy core as reaction to some events (i.e. when the rental starts and when the rental ends).

PowerEnJoy Mobile Application

As stated in the RASD document, this mobile application represents the main interface for customers and it should be implemented for both the Android and the iOS platforms.

The main purpose of the mobile application is to retrieve information from the PowerEnJoy core, to transmit user actions and show their results.

The Android application is implemented using Java and all the native components offered by the Android SDK.

The iOS application is implemented using Swing and the UIKit for the interface. A solution involving multi-platform development (e.g. Xamarin) can also be taken into account, even if a native implementation still remains the most encouraged solution in order to offer the best possibile experience to customers. A solution involving WebViews is considered an applicable solution for matters of reliability.

The mobile application directly communicates with the business logic using its RESTful API interfaces.

PowerEnJoy Control Room

As stated in the RASD document, the PowerEnJoy control room is a web-application and it represents the main interface to the system for the company operators. It is implemented using HTML5 and jQuery and it relies on the web server tier.

2.2.6 External components

SMS gateway

This component is used in order to notify users using SMS on their mobile phone numbers. It's not directly implemented by the system but it's used as-a-service from Trendoo[1]. Details on the communication between the system and the gateway are provided in section 2.5.5.

[1] http://www.trendoo.it

Payment gateway

This component is used in order to process users' payments. It's not directly implemented by the system but it's used as-a-service from GestPay provided by Banca Sella[2]. Details on the communication between the system and the gateway are provided in section 2.5.5.

Push Notifications gateway

This component is used in order to send push notifications to the users on their mobile phones. It's not directly implemented by the system but it's used as-a-service from Google Firebase[3]. Details on the communication between the system and the gateway are provided in section 2.5.5.

[2] https://www.gestpay.it/gestpay/index.jsp
[3] https://firebase.google.com/

2.3 Deployment view

The physical deployment of the system is done on 3 layers, as shown in figure 6

- Clients are deployed on different devices according to their finalities.
- The application server and the web server can be deployed on the same physical machine because the web server serves the ControlRoom web-application only.
- The relational and the NoSQL databases can be deployed on the same physical machine. However, since they are completely indipendent one from the other, they could be moved on different machines as soon as it will be required for some reason.

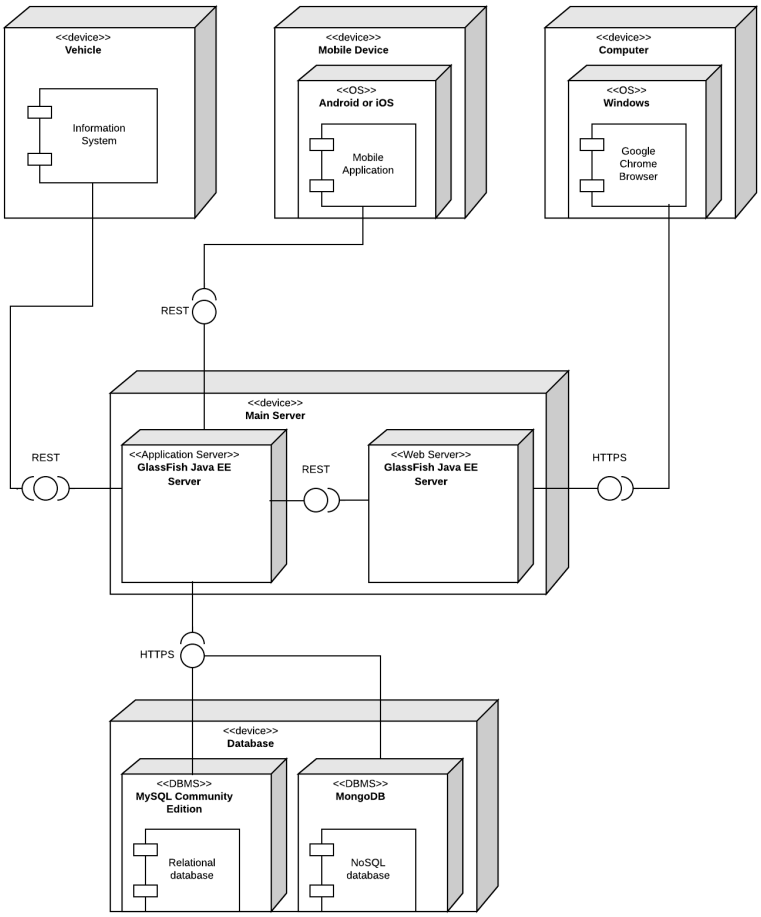

Figure 6: Deployment diagram

2.4 Runtime view

In this section we will expand some scenarios from the RASD document with the purpose of clarifying the dinamic behaviour of the system. Entities of database

are represented using the Java Persistence API entities.

For the sake of simplicity, the log manager bean is shown in every sequence because it is intended to register every action. The Rest API Manager bean is put between transactions that require it.

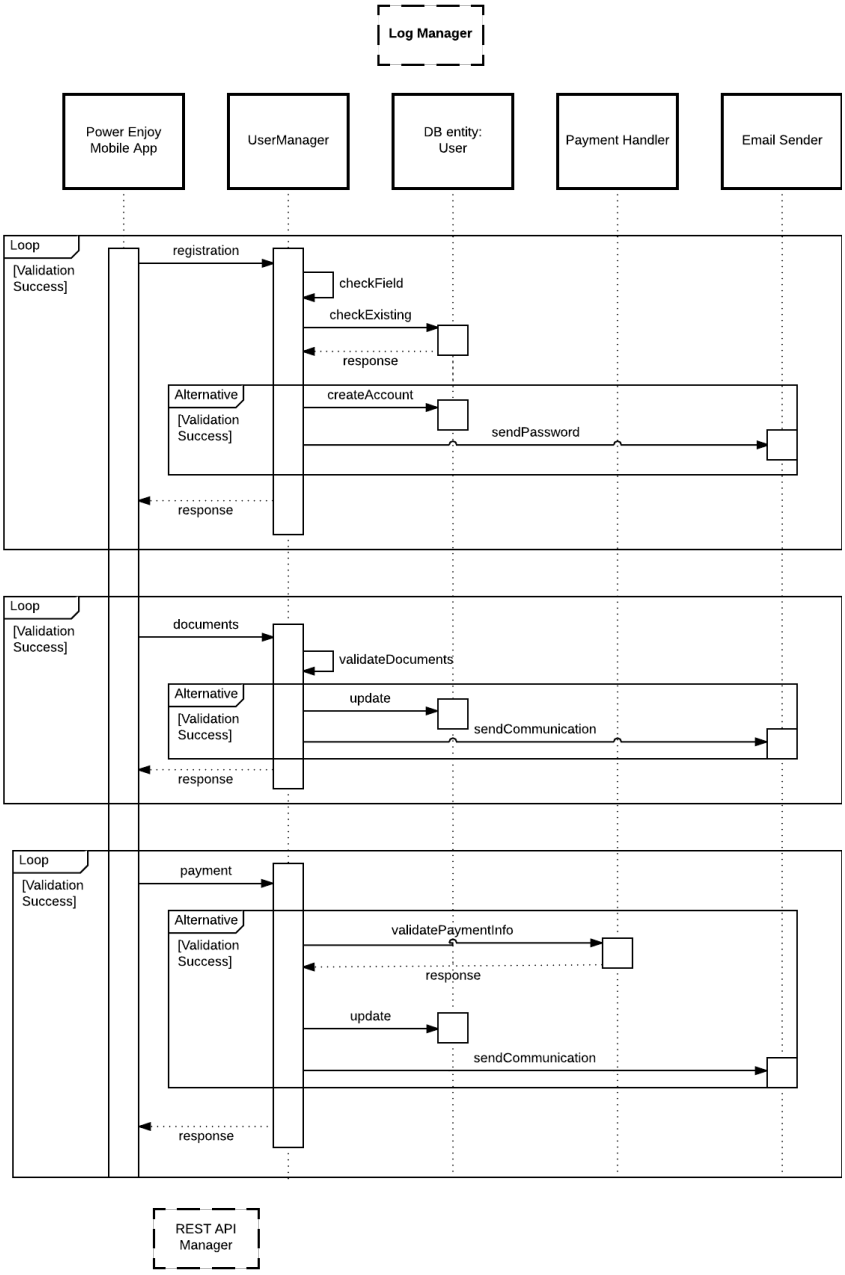

Figure 7: Account registration - Sequence diagram

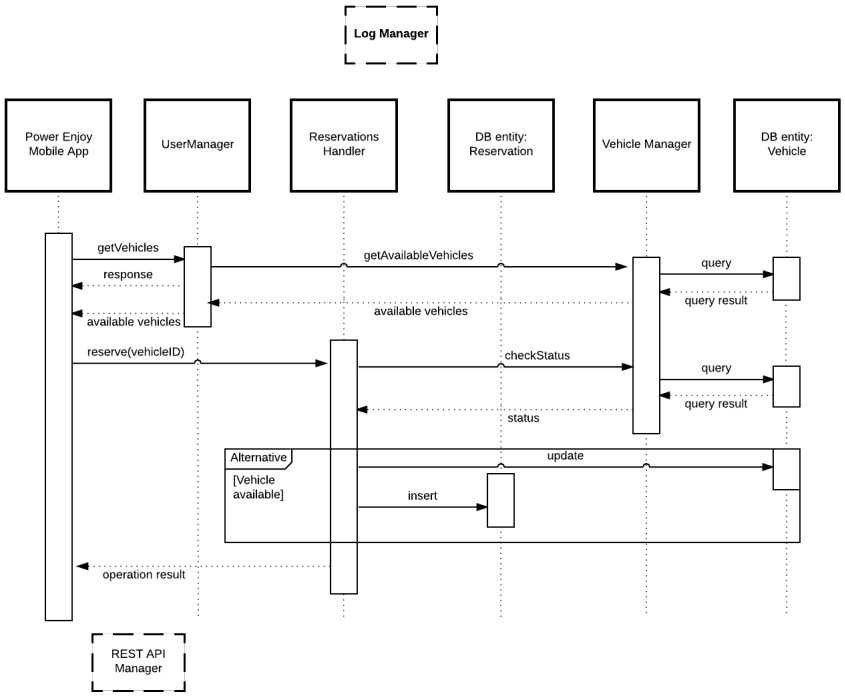

Figure 8: Vehicle research and reservation - Sequence diagram

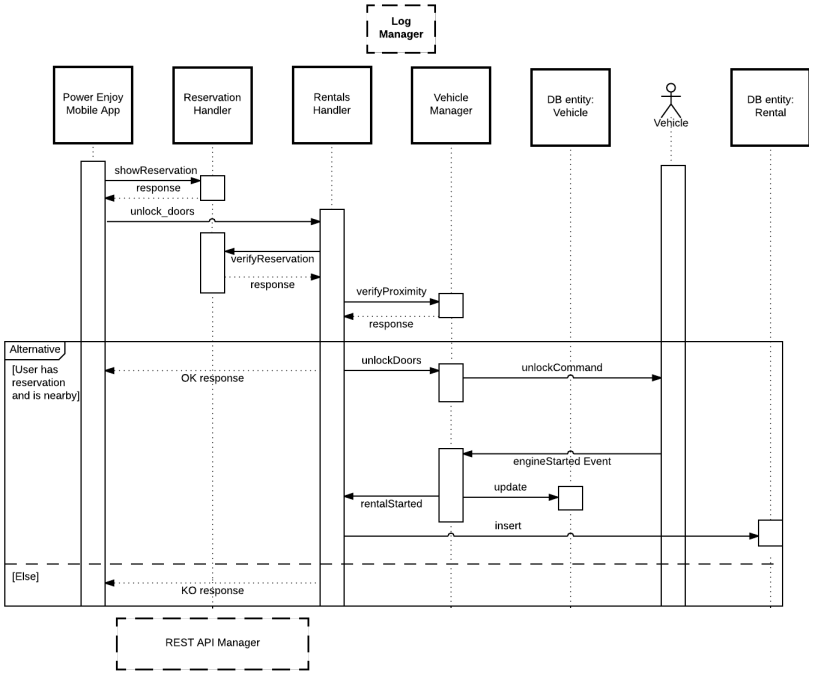

Figure 9: Rental beginning process - Sequence diagram

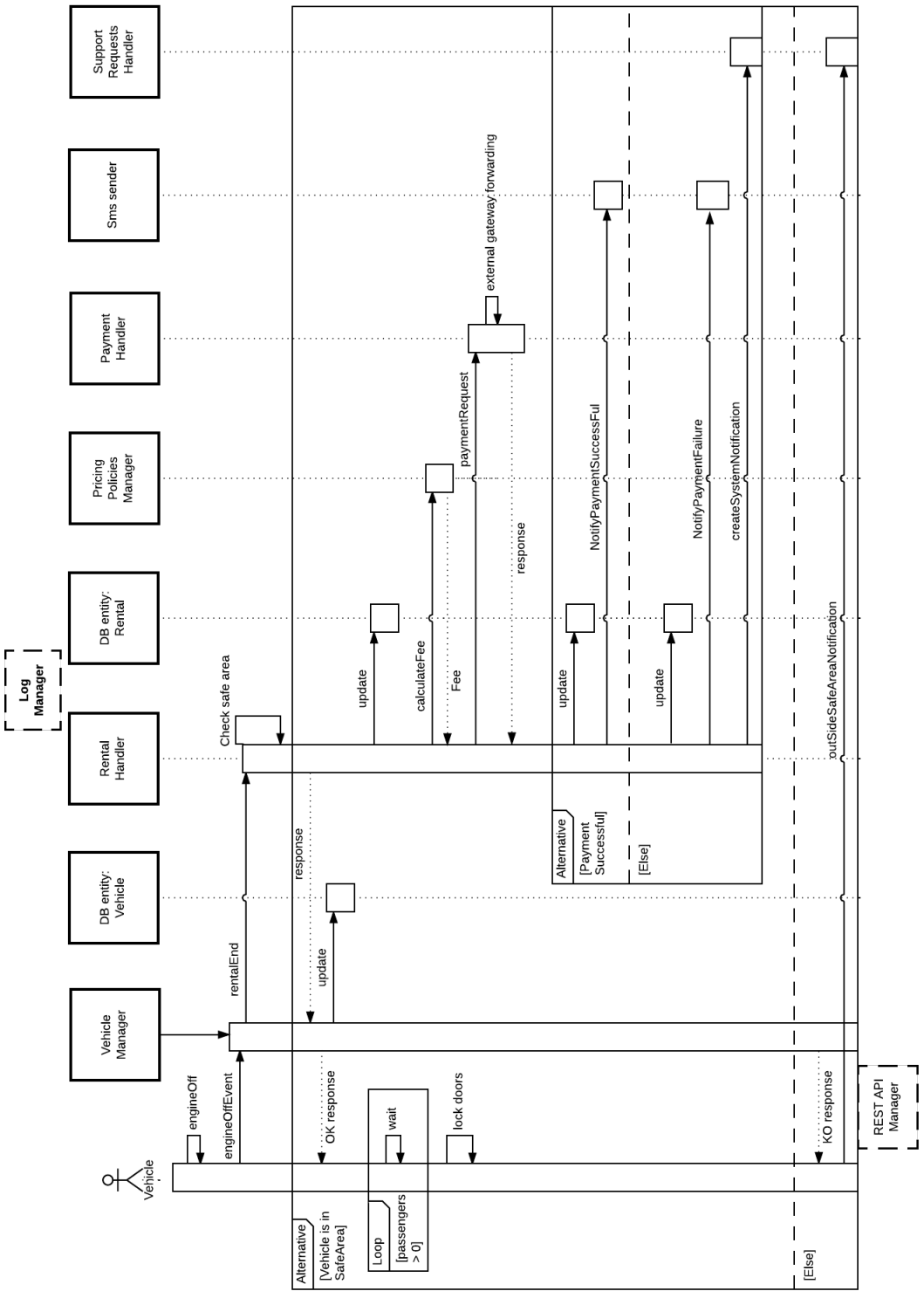

Figure 10: Rental finished - Sequence diagram

2.5 Component interfaces

2.5.1 Application server to database

The communication between the application server and the two databases goes using standard TCP/IP protocol on two different network ports. Both the relational and the NoSQL databases are abstracted in the application core by using the Java Persistence APIs, which use the correct SQL dialect in order to interact with the MySql Community Edition DBMS and the MongoDB DBMS.

Since the two databases can not directly communicate one with the other, if a query requires information from the databases the merge should be done at an application level by merging two different queries results.

2.5.2 Application server to clients (REST APIs)

The clients and the application core communicate by using REST APIs over the HTTPS protocol. Requests are forwarded using POST parameters, while responses are encoded in the JSON format.

In the following pages, we list and detail the REST APIs implemented by the beans. Parameters marked with an *O* are optional.

User Manager

login This interface point allows registered user to log in using username and password. If the credentials are correct, the response contains an authentication_token to be sent as parameter in further communications.

registration This interface point allows a customer to register and create an account. If the registration process is successful, an authentication_token is returned and an email containing a password is sent to the user.

documents This interface point allows a logged-in user to associate its document to the account. This interface is used both during the registration and when the user updates its account information.

payment This interface point allows a logged-in user to associate its payment information to the account. This interface is used both during the registation and when the user updates its payment method. The provided information are validated, as stated in the RASD document, by applying a fee of 0,01 EURO.

get vehicles This interface point allows a logged-in user to retrieve available vehicles in a certain area.

Service	***Input***	***Output***
login	• user_email • user_password	• Login result code • [O] authentication_token
registration	• full_name • username • email_address • home_address • phone_number	• Registration result code • [O] authentication_token
documents	• authentication_token • identity_card • id_expiration • driving_license • dl_expiration	• Operation result code
payment	• authentication_token • credit_card_number • credit_card_expiry • credit_card_cvv • credit_card_owner	• Operation result code
get_vehicles	• authentication_token • position_lat • position_long	• A list of available vehicles in a certain range from the given position, together with their positions and details.

Operators Manager

login This interface point allows operators to log-in using their id and password. If the authentication is successful an operator_authentication_code is returned to be used in furhter communications.

support requests This interface point allows operators to retrieve support requests forwarded by users or automatically generated by the system requiring an operator action.

handle request This interface point allows operators to handle a support request by sending a message or changing the request status.

get user info This interface point allows operators to get all the information about a specific user such as account information and activity log.

edit user This interface point allows operators to manipulate or disable an user account.

charge driving loan This interface point allows operators to charge an user for a loan he took during a rental. Operators just need to provide driving loan details and the system automatically detects which user was driving the car at the moment of the infraction.

get vehicles This interface point allows operators to retrieve all the vehicles in a specific area.

Service	***Input***	***Output***
login	• operator_id • operator_password	• Login result code • [O]: op_authentication_token
support_requests	• op_auth_token • [O] order_by • [O] from_date • [O] to_date • [O] by_user • [O] status	• A list of support requests
handle_request	• op_auth_token • request_id • [O] new_status • [O] text_answer	• Operation result code
get_user_info	• op_auth_token • [O] user_email	• Informations about the user
edit_user	• op_auth_token • [O] account_status	• Operation result code
charge_driving loan	• op_auth_token • amount • driving_loan_details	• Operation result code

get_vehicles	• op_auth_token • position_lat • position_lon • range	• A list of all the vehicles in the given range from the given position, together with their position and details

Support Requests Handler

send request This interface point allows a logged-in user to forward a support request or to forward a system notification.

send message This interface point allows a logged-in user to reply to a still open support request.

Service	***Input***	***Output***
send_request	• authentication_token • request_details	• Operation result code
send_message	• authentication_token • request_id • message_text	• Operation result code

Reservations Handler

reserve This interface point allows a logged-in user to reserve a vehicle. If the operation is successful, a reservation code is returned.

cancel reservation This interface point allows a logged-in user to cancel an active reservation.

show reservation This interface point allows a logged-in user to get an active reservation, if any.

Service	***Input***	***Output***
reserve	• authentication_token • vehicle_id	• Operation result code • [O] Reservation code
cancel_reservation	• authentication_token • vehicle_id	• Operation result code
show_reservation	• authentication_token	• [O] User's active reservation

Rentals Handler

unlock doors This interface point allows a logged-in user with an active reservation over a vehicle to unlock the doors of the vehicle. The operation is successful only if the user is nearby ($<$ 100 mt) the vehicle.

Service	***Input***	***Output***
unlock_doors	• authentication_token • vehicle_id • user_position_lat • user_position_lon	• Operation result code

---

Vehicle Manager

event This interface point is used by vehicles to send a status update.

Service	***Input***	***Output***
event	• vehicle_token • event_type [ENGINE_ON | ENGINE_OFF] • status	• Operation result code

---

2.5.3 Application server to vehicles

As stated in the RASD and in this document, we assume that the information system installed onboard of company vehicles as given and not part of this project.

For the way we structured the whole architecture, we just require that the vehicles expose an unique interface point, using the RESTful approach encoded using the JSON format:

Service	*Input*	*Output*
status		• battery_level • battery_charging • location_lat • location_lon • passenger • speed • mechanical_status
command	• type [LOCK_DOORS | UNLOCK_DOORS | RENTALEND_OK | RENTALEND_KO]	• response_code

2.5.4 Web server to clients

The web server interfaces with browsers only: the communication goes using the standard HTTPS protocol.

2.5.5 Application server to external components

Application server to SMS Gateway

Trendoo SMS Gateway provides a wide range of communication protocols. The most suitable for our purposes is HTTPS.

Application server to Payment Gateway

GestPay by BancaSella Payment Gateway provides a complete SDK. Communication works using HTTPS.

Application server to Push Notifications Gateway

Firebise provides a complete SDK. Communcation works using HTTPS.

2.6 Selected architectural styles and patterns

The main architectural style for the system is the client-server one, splitted on four logical layers and three physical layers.

The main design pattern to adopt is the well known MVC, suggested for every object oriented project. In this way we can easily separate the model, whose main purpose is to manage the data, from the view, which is the presentation layer, and from the controller, whose main purpose is to manage the other two components.

2.6.1 Passwords and other sensible data

The system collects users' passwords and other kind of sensible data (e.g. mobile phone number, email address, home address, payment information) and a series of logs about the vehicles that could be used to track the users. For this reason, the database must be protected both physically and logically in order to avoid any kind of data loss.

Passwords should be salted and encrypted using the most advanced algorithms.

2.6.2 Maps

The system massively uses maps as an immediate way to visualize the vehicles. Also geocoding and distances calculation are used in all the components of the system. *GoogleMaps* is the most suitable external service to satisfy all these needs on both the client and the server side.

2.6.3 Communication with vehicles

To communicate with the vehicles is a key requirement of the system, mainly for what concerns security. In order to avoid that malicious users can take control over the vehicles - and maybe unlock them - the on board information system should accept commands from the PowerEnJoy Core only. This can be achieved by using TLS certificates and encrypted communications.

2.6.4 Reliability and availability

As stated in the RASD document (sections 3.2.1 and 3.2.2), reliability and availability are core requirements for the system. In order to achieve them, a combination of redundancy and parallelization appears as the most suitable solution. The deployment architecture described in section 2.3 is engineered in order to easily support scalability and parallelism.

3 Algorithm Design

In this section we want to provide an outline of the main and the more interesting algorithm used by the system. Algorithms are detailed using java-like pseudocode and textual descriptions because of the reasonable tradeoff between expressivity and simplicity. They will represent high-idea procedures in order to not give any restrictions for future coding implementations.

3.1 New reservation

This algorithm represents the procedure issued when the user wants to make a new reservation over a vehicle. This algorithm is part of the Reservations Handler bean.

```
Reservation newReservation(Vehicle v, User u){
    /* We first check that the vehicle is available and that
       the user can make a reservation*/
```

```
    if(v.getStatus() == STATUS_AVAILABLE && u.isEnable() &&
        this.getReservation(u) == NULL && RentalsManager.
        getRental(u) == NULL){
        /* We create a reservation object containing the
            vehicle, the user, the start time and the
            expiration time */
        Reservation r = new Reservation(v, u, System.
            currentTime(), System.currentTime()+
            RESERVATIONS_EXPIRY_TIME);

        /* We change the status of the vehicle using the
            VehicleManager */
        VehicleManager.getForVehicle(v).changeStatus(
            STATUS_RESERVED);

        /* We store the reservation in the database using the
            JavaPersistenceApi*/
        JPA.storeReservation(r);

        /* We schedule a task to handle the expiry of the
            reservation */
        new Handler().executePostDelayed(this.
            reservationExpiry(r), RESERVATION_EXPIRY_TIME);

        /* We add something in the log related to user's
            activity */
        LogManager.log(LOG_USER_ACTIVITY, u, "Reserved " + v);

        /* We return the newly created reservation */
        return r;

    }else{
        /* If the reservation can not be created we simply
            return NULL */
        return NULL;
    }
}
```

3.2 Unlock Doors

This algorithm represents the procedure issued when the user with an active reservation over a vehicle wants to unlock the doors of the vehicle. This algorithm is part of the Rentals Handler bean.

```
boolean unlockDoors(Vehicle v, User u, Coordinates
    userPosition){
    /* We first check if the user has an active reservation
        over the vehicle, then we check if he is nearby */
    if(v.getStatus == STATUS_RESERVED && this.getReservation(u
        ).getVehicle().equals(v)
        && Math.euclideanDistance(userPosition, VehicleManager
            .getForVehicle(v).getPosition()) <
            MAX_DISTANCE_FROM_VEHICLE_TO_UNLOCK){
```

```
            /* We use the vehicle manager to send the command to
                the vehicle */
            VehicleManager.getForVehicle(v).unlockDoors();

            /* We log the activity */
            LogManager.log(LOG_VEHICLE_ACTIVITY, "Doors unlocked
                by" + u);

            return TRUE;
        }

        return FALSE;
    }
```

3.3 Rental ended

This algorithm is applied in the Rentals Handler when a rental is ended.

```
    boolean rentalEnded(Rental r){
        Vehicle v = r.getVehicle();
        User u = u.getUser();

        /* The rental can be terminated only inside a safe area */
        if(v.isInSafeArea()){
            /* We calculate the fee and we charge the user */
            double rentalFee = PricingPolicyManager.calculateFee(r
                );
            PaymentHandler.chargeUser(u, rentalFee);

            /* We update the rental object and the database */
            r.setEnd(System.currentTime());
            JPA.updateRental(r);

            /* Based on the battery level, we change the status of
                the vehicle */
            VehicleManager vehicleManager = VehicleManager.
                getForVehicle(v);
            if(v.getBatteryLevel() > 20){
                vehicleManager.changeStatus(STATUS_AVAILABLE);
            }else{
                vehicleManager.changeStatus(STATUS_UNAVAILABLE);
            }

            /* We log the operation */
            LogManager.log(r + "terminated");

            return TRUE;
        }else{
            /* We notify the user via sms too */
            SmsSender.send(u, OUTSIDE_SAFE_AREA);
            return FALSE;
        }
    }
```

3.4 Fee calculation

This algorithm is part of the Pricing Policies Manager bean. This algorithm is more interesting from the design structure of the code rather then the code complexity itself. Pricing policies should be implemented as objects implementing the PricingPolicy interface.

```
double static calculateFee(Rental r){
    /* We first calculate the base price by multiplicand the
       duration of the rental and the price per minute */
    double price = r.getTime() * PRICE_PER_MINUTE;

    /* We first apply all the discounts */
    for(PricingPolicy pp : discounts){
        price = pp.apply(r, price);
    }

    /* We then apply all the surcharges */
    for(PricingPolicy p : surcharges){
        price = pp.apply(r, price);
    }

    return price;
}

interface PricingPolicy{
    double static apply(Rental r, double currentFee);
}
```

4 User Interfaces Design

In this section we include some mockups of the mobile application. This mockups are styleless and does not refer to any specific platform: the purpose of this section is to provide details about the structural elements of the user interface but not their style.

4.1 PowerEnJoy Mobile Application

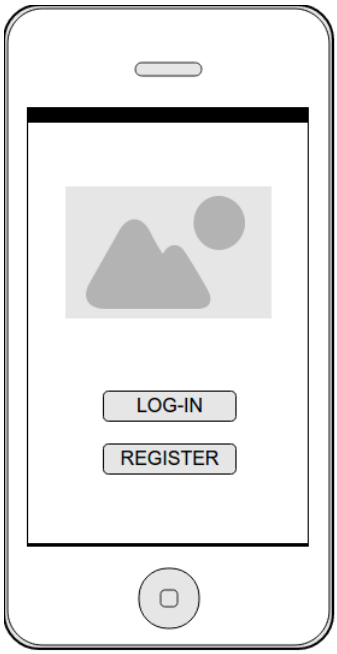

Figure 11: PowerEnJoy Mobile Application mockup 1

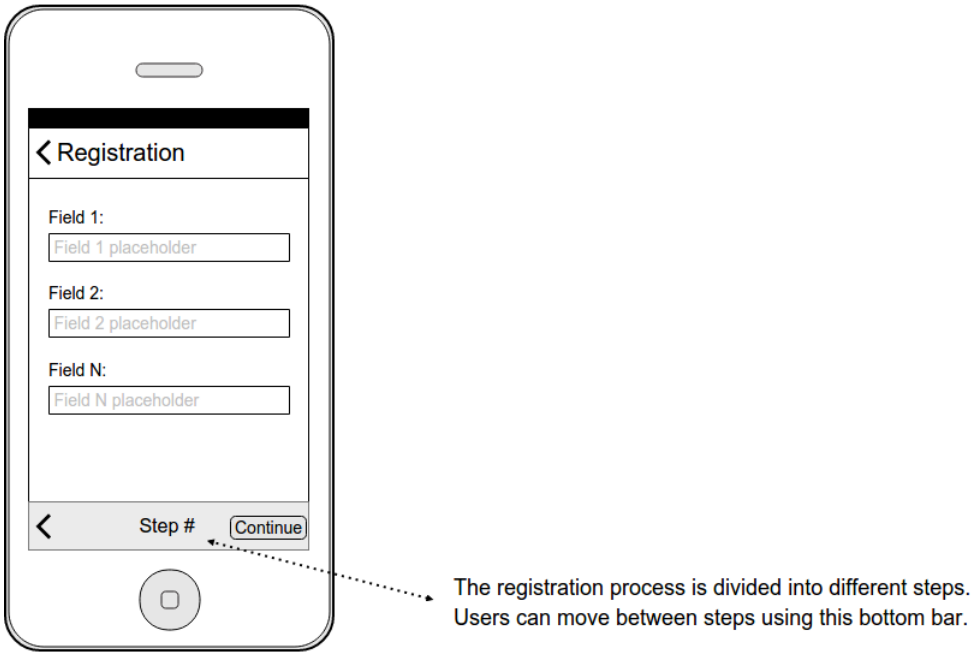

Figure 12: PowerEnJoy Mobile Application mockup 2

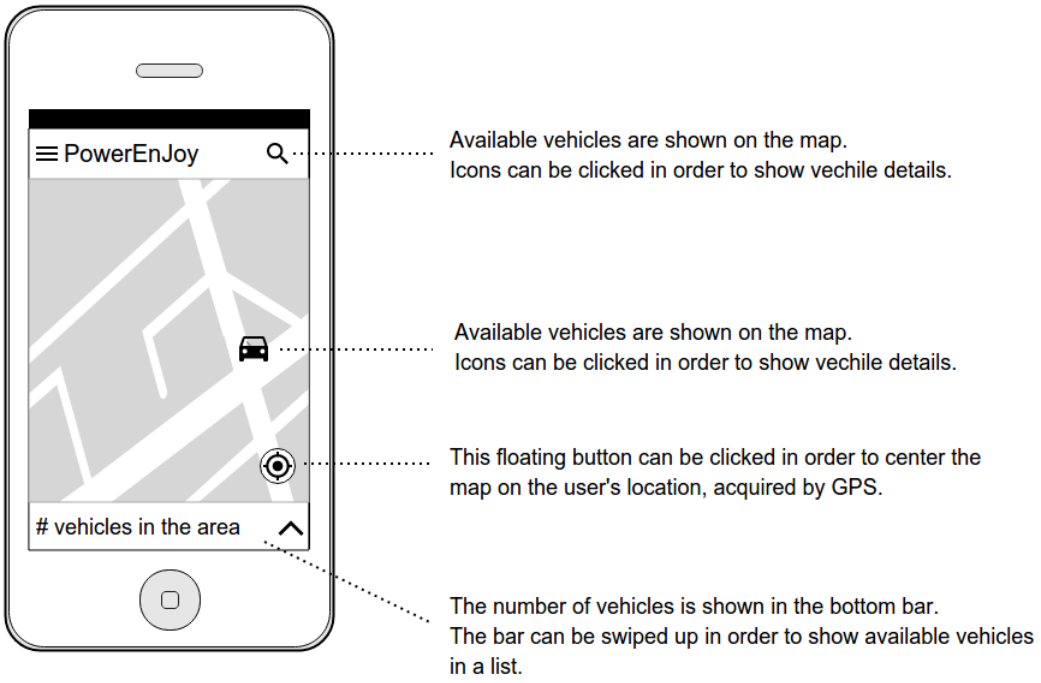

Figure 13: PowerEnJoy Mobile Application mockup 3

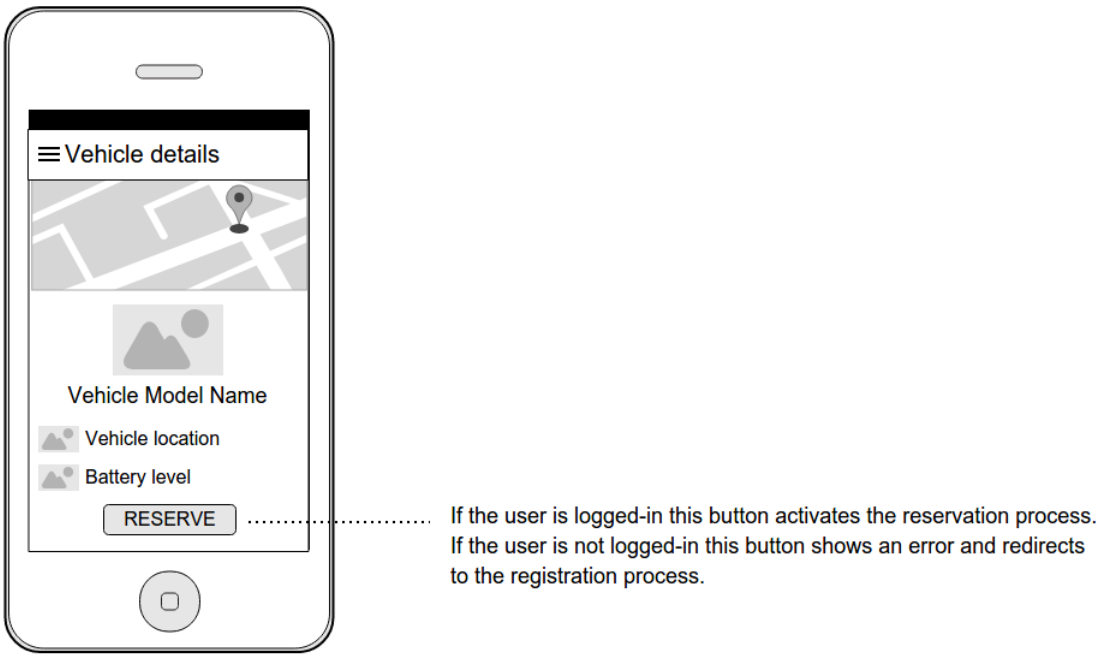

Figure 14: PowerEnJoy Mobile Application mockup 4

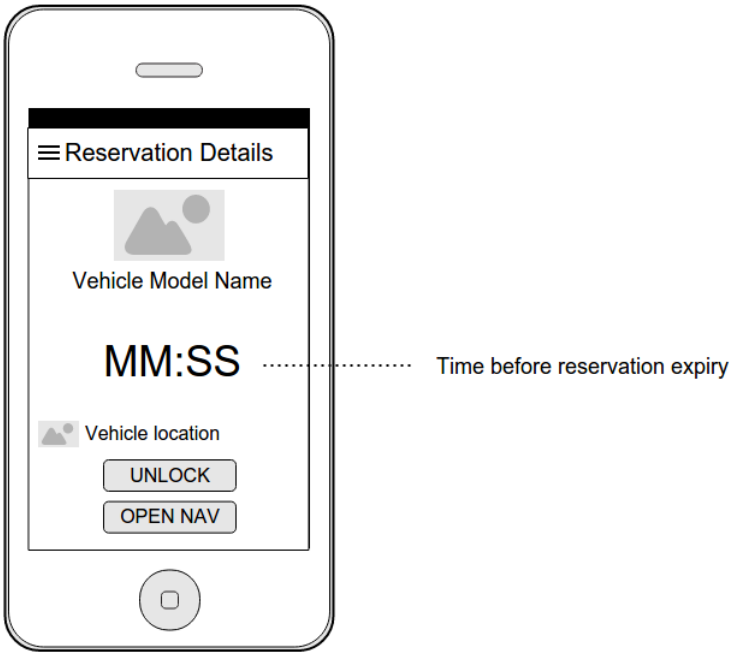

Figure 15: PowerEnJoy Mobile Application mockup 5

4.2 PowerEnJoy Control Room

5 Requirements Traceability

The aim of this document is to optimally design the requirements and goals shown in the RASD document.

Here we logically match them with feature explained in this Design Document.

5.1 Functional requirements

Component (DD)	Functional requirements (RASD document, section 2.2.1)
User Manager	Users can register.
User Manager	Users can log-in.
User Manager	Logged-in users can edit their account information and payment method.
User Manager	Users can see available vehicles on a map.
User Manager	Users can search for available vehicles in a specific area using their GPS position or inserting an address.
User Manager	Users can see specific vehicle information (i.e. vehicle model, battery percentage, estimated kilometers of autonomy).
Reservations Manager	Logged-in users can make a vehicle reservation.

Reservations Manager	Logged-in users can cancel an existing vehicle reservation.
Rentals Manager	Logged-in users can unlock a vehicle they have reserved when they are nearby.
Reservations Manager	Logged-in users can not have a reservation and a rental at the same time.
External - Google Maps	Logged-in users can be guided by a navigator to their reserved vehicle.
Support Requests Handler	Logged-in users can send a support request.
Operators Manager	Operators can log-in.
Operators Manager	Operators can see on a map the position of all the vehicles together with their information.
Operators Manager	Operators can see users' account information.
Operators Manager	Operators can disable or enable a user's account.
Operators Manager	Operators can see user's last activity.
Operators Manager	Operators can see users' support requests.
Operators Manager	Operators can change the status of a user's support request into OPEN;HANDLING or CLOSED.
Operators Manager	Operators can see system notifications.
Operators Manager	Operators can reply to a user's support request.
Operators Manager	Operators can change the status of a system notication in OPEN; HANDLING, CLOSED or CRYTICAL.
Operators Manager	Operators can change the status of a vehicle into AVAILABLE or NOT AVAILABLE.
Operator Manager	Operators can charge users for driving fines.
Vehicle Manager	The component can handle rent events sent from vehicles (i.e. RENT STARTED, RENT TERMINATED).
Rentals Manager	The component can send commands to vehicles (e.g. UNLOCK DOORS).
Payment Handler	The component can charge users on their payment method applying company pricing policies.
Payment Handler	The component can handle payment failures.
All	The component can access and manage the company database.
Email sender	The system can send communications to users via email on their email address.
SMS sender	The system can send communications to users via SMS on their mobile phone number.

Push notification sender	The system can send communications to users via push notifications on their smartphone.
CronTask Manager	The system can generate monthly invoices and sent them to the users on their email address.
Vehicle Manager	The system can detect when a vehicle reservation expires.
Vehicle Manager	The system can mark a vehicle as NOT AVAILABLE when its battery level is less than 20% and it's not rented by any user.
Vehicle Manager	The system can mark a vehicle as AVAILABLE when its status is NOT AVAILABLE and its battery level is higher than 20%.

6 Appendix

6.1 Tools used

- LyX as LaTeX editor.
- Moqups.com for mockups.
- Lucidchart for diagrams.
- Github as version controller.
- Google docs for beta documents sharing.

6.2 Effort spent

Guido Borrelli	35
Guido Borrelli	35