Politecnico di Milano

PowerEnjoy Service - Design Document

December 10, 2016

Version 1.1

Authors:

- Domenico FAVARO (Mat. 837995)
- Matheus FIM (Mat. 876069)
- Caio ZULIANI (Mat. 877266)

Prof. Elisabetta DI NITTO

Contents

1 Introduction

1.1 Purpose

This Design Document serves the purpose to present to all parties interested the description of the structure for the PowerEnjoy Service. It provides documentation of Software, Architecture and other important aspects of Design to help the understanding and development of the System. Every component that is implemented for the System will be explained as well as the purpose they serve to contribute to the fullfilment of all the project Requirements. All strategies and design decisions will be documented as well. Software Design Description, decisions and key information to be used to communicate the general purpose of the structure to our stakeholders, and serve as detailed documentation for the components of the System for the developers.

1.2 Scope

This Document presents the PowerEnjoy Service System, an electric car sharing service. To better understand the broader scope of the service, it is presented in the RASD Document. This Document will not include detailed information on all the tools and protocols that will be used in the development of the System but rather their purpose and functionality inside the PowerEnjoy System. As example, general knowledge of the Client-Server structure is expected as it will not be rigorously explained but instead how such structure will be used to satisfy our System's Requirements.

1.3 Glossary: Definitions, Acronyms, Abbreviations

- **RASD:** Reqirements And Specifications Document.
- **DD:** Design Document.
- **Java EE:** Java Enterprise Edition. Software Development Platform.
- **App:** Application. Refers to the deployed service as Web or Mobile Application.
- **Component:** Software element that implements and offer functionalities in the System.
- **EJB:** Enterprise Java Beans. Component in the Business Tier for the Application Logic.

- **DBMS:** DataBase Management System.
- **JDBC:** Java Database Connectivity, Java API to connect to DataBases.
- **HTTPS:** Hypertext Transfer Protocol over TLS. Protocol to safefully communicate over the Internet.
- **TLS:** Transport Layer Security that provides communication security.
- **REST/RESTful:** Representational state transfer, architectural style for the System communications.
- **MVC:** Model View Controller, software design pattern for implementing interfaces.
- **API:** Application Programming Interface.

For other concepts concerning the Service definition look in the **Glossary** section of the RASD.

1.4 Reference Documents

- Specification Document: Assignments AA 2016-2017.pdf
- PowerEnjoy Requirements And Specifications Document (RASD)
- IEEE Std 1016-2009 IEEE Standard for Information Technology-Systems Designs-Software Design Descriptions (SDD IEEE 1016-2009.pdf)
- Example Documents:
  - Sample Design Deliverable Discussed on Nov. 2.pdf
  - Software Design Document (SDD) Template (sdd_template.pdf)

1.5 Document Structure

Section 1 - Introduction: This section provides a general description of the purpose and structure of this Design Document.

Section 2 - Architectural Design: This section illustrates a broader to specific view of the components that form part of the System, presenting from the overview of the architecture of the system to a description of how each component will interface inside the structure.

Section 3 - Algorithm Design: Important functionalities of the System that require the development of algorithms will be described in this section.

Section 4 - User Interface Design: All details refering to how the User will interface with the System, from the Web Application to the screens inside the Cars will be shown in this section. As well as general mockups of the Graphical User Interface (GUI) screens.

Section 5 - Requirements Traceability: In this section is explained how the Design decisions and structure help fulfill the Requirements for the System that were defined in the RASD.

Section 6 - Effort Spent: Detailed record of the hours worked for each member of the team is documented in this section.

Section 7 - References: Any reference to additional external sources that can help the better understanding of this Document is documented in this section.

2 Architectural Design

2.1 Overview

As was show in the Proposed System part of the RASD, the PowerEnjoy structure will consist on a 3-tier Client-Server Architecture. The server part built on a JEE Platform with access to the Company's Database Server and the Client consisting of Users, CRM and Cars that will use a browser to access our Web Application and a Mobile Device and Screen inside the Cars will access the Mobile Application. As we need to interface our application with the Car functions (i.e. Lock, Unlock, Battery state) the application must have some Logic implemented client side. This distributed Logic structure will allow GUI to be created on the Client side as well. The Database Server will belong to the Company and will allow registration of User, Reservations, Rides and Payments. Cars, CRM Employees and Parking Locations are asumed to be added outside of this System.

- Client Tier (Web App, Mobile App, Car)
- Server Tier (Java EE Web and Application Server)
- Enterprise Information System (EIS) Tier (DataBase Server)

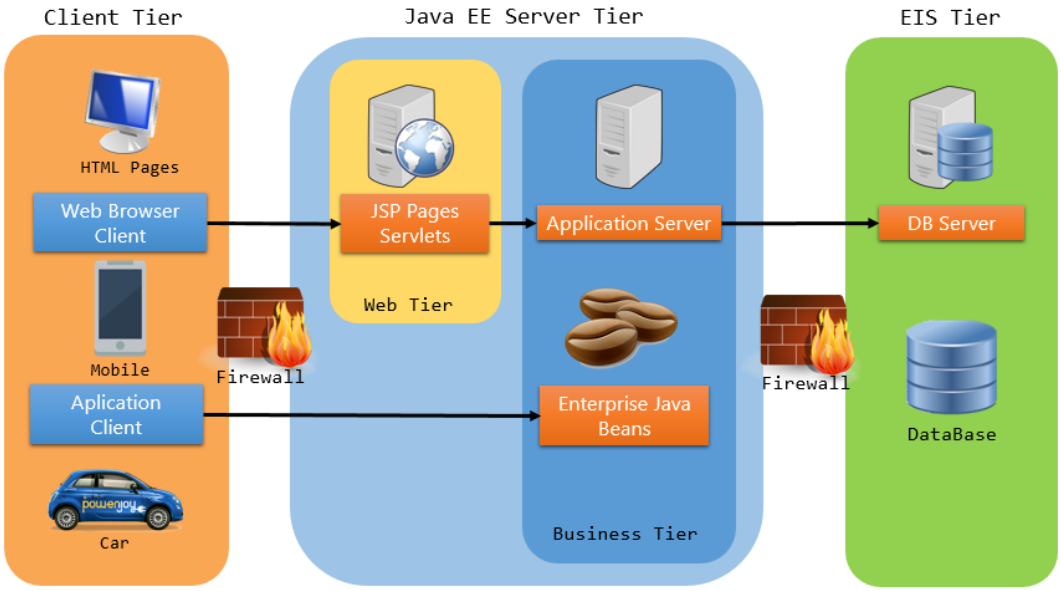


Figure 1: Proposed System Architecture

2.2 High level components and their interaction

Following our Architecture we organize the High level components on the 3 Tiers. This overview shows the interaction between the components and the entities they represent inside the System. Clients will have their User and CRM Components, distributed as Interfaces and Web Pages for the Apps. Inside the Server through Servlets and Session Beans a User and CRM Controllers will provide the services to the clients sending the requests with the respective Controllers. Based in the MVC pattern Controllers will manage the data that will be viewed by the Client. Each Controller will manage and represent the System Entities previously explained in the RASD and will interface with a Persistent DataBase Connection to the DBMS. Helpers and Extra Feature Components are represented as well.

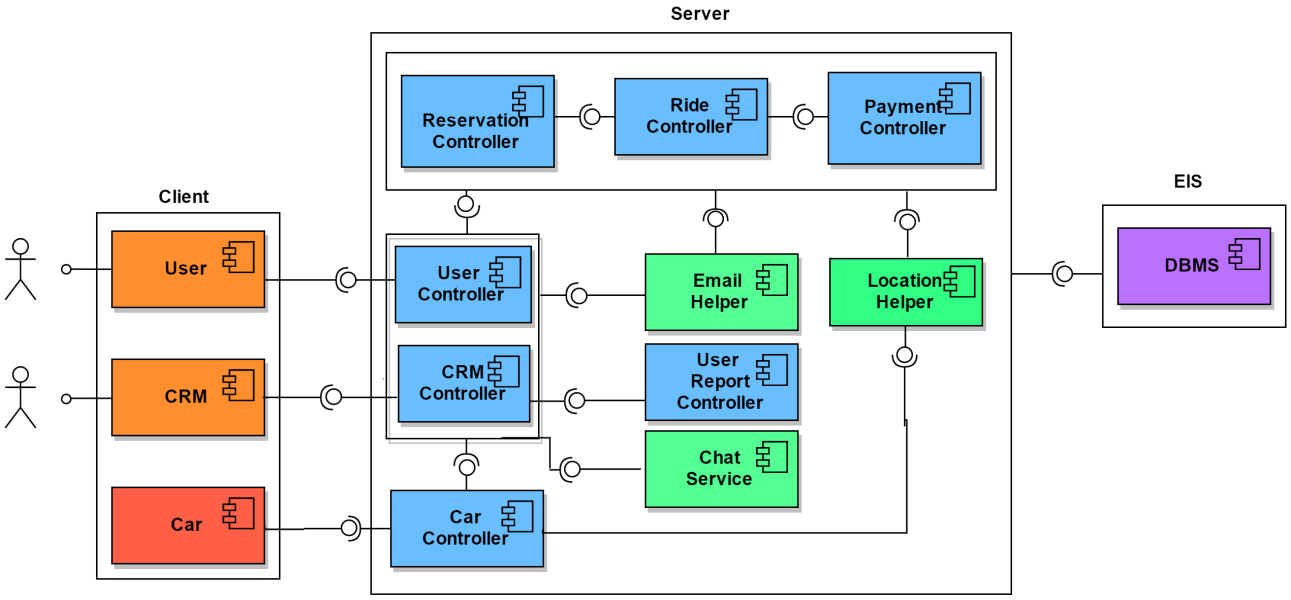


Figure 2: High Level Component Structure

2.3 Component View

2.3.1 User Component

Represents the User in the Client Tier that translates user actions and send requests to the UserController Component, then presents the output to the User. Is composed of several pages that present different types of information. Implement the Service Requests for each Actor, Web Pages in Web Server and Mobile Application and will have to interface with the central Server Component.

2.3.2 CRM Component

The CRM in the Client Tier will access just via browser, web server to the System. Is composed of a different set of pages that perform the requests to the CRM Controller.

Figure 3: Client Components

2.3.3 Car Component

Represents the Car outside of our system that presents an interface to provide information and receive commands from the Car Controller Component.

2.3.4 Reservation Controller

Manages the Reservations made by Users, when a reservation is corfirmed, it will create the correspondant Ride.

2.3.5 Ride Controller

Manages the Active Rides. When a ride is finished, it creates a Payment using the Payment Controller.

2.3.6 Payment Controller

Implements the Logic to make a Payment or Transaction to a User Account, using a PaymentHelper to interface with Credit Card and online Payment. Used by the Ride and the User Report Controllers.

2.3.7 User Report Controller

Used by the CRM Controller to generate User Reports.

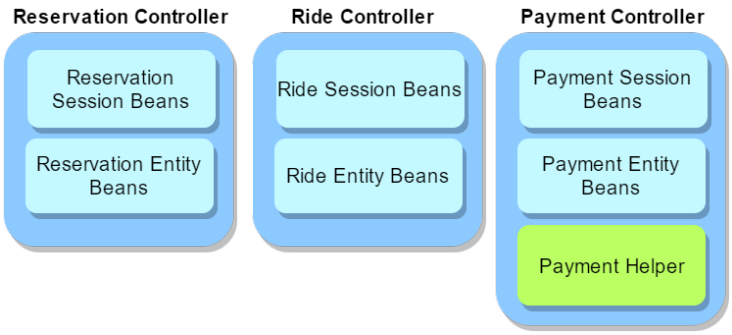


Figure 4: Entity Controllers

2.3.8 User Controller

Will handle the requests coming from the Users. Handle their Login, Registration and Session functionalities and redirect others to their respective controllers.

2.3.9 CRM Controller

Will handle the requests from CRMs. Similar to the UserController it will handle CRM Login and Session functionalities and redirect others to their respective controllers.

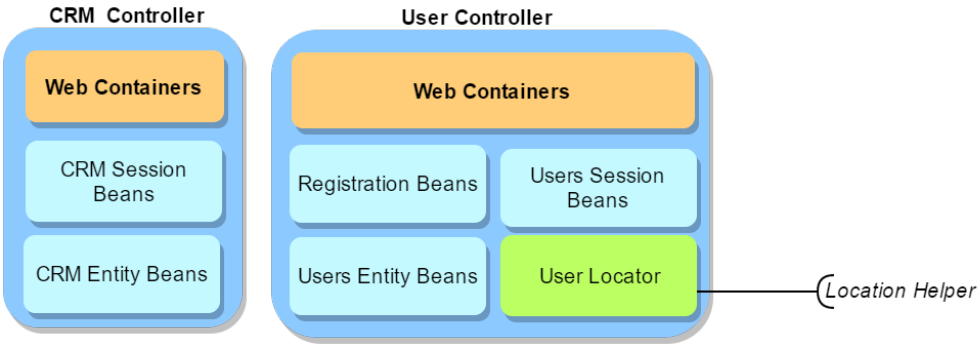


Figure 5: User CRM Controllers

2.3.10 Car Controller

Will manage Cars Status, Locations and interface to Cars to send them instructions.

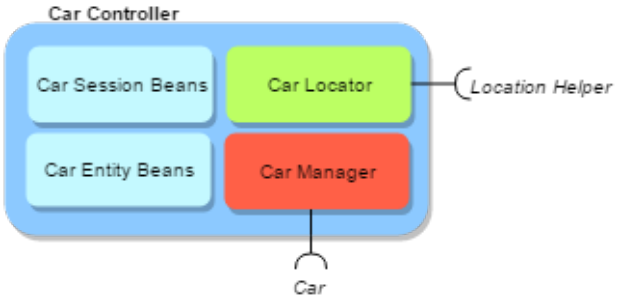


Figure 6: Car Controller

2.3.11 Location Helper

Will provide an interface for Location queries. Can interface to an external Service Provider API (Google Maps).

2.3.12 Chat Service

Will provide the chat platform to provide a channel of communication between Users and available CRMs

2.3.13 Email Helper

Will allow the System to communicate via mail with the Users (password, payment).

2.3.14 DBMS

Each Controller will implement a Persistent Entity for each class that will be mapped on the DB. The Structure of the DB will mirror the Classes in our System.

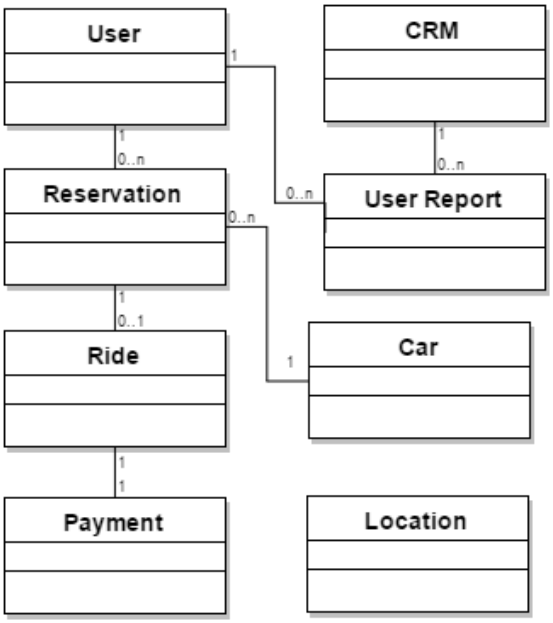


Figure 7: Database Structure

2.4 Deployment View

A first proposal for the Deployment View shows our main components in their deployed hardware as seen in the proposed Architechture. Mainly our Web, Application and DB Servers and the Client Devices. Our proposal for the communication protocols include:

- The Web Clients communicate to the Web Server via HTTPS.
- The Application Server exposes a RESTful API for the Web Server, Mobile Apps and Cars.
- The Application Server communicates with the DB Server using JDBC connection.

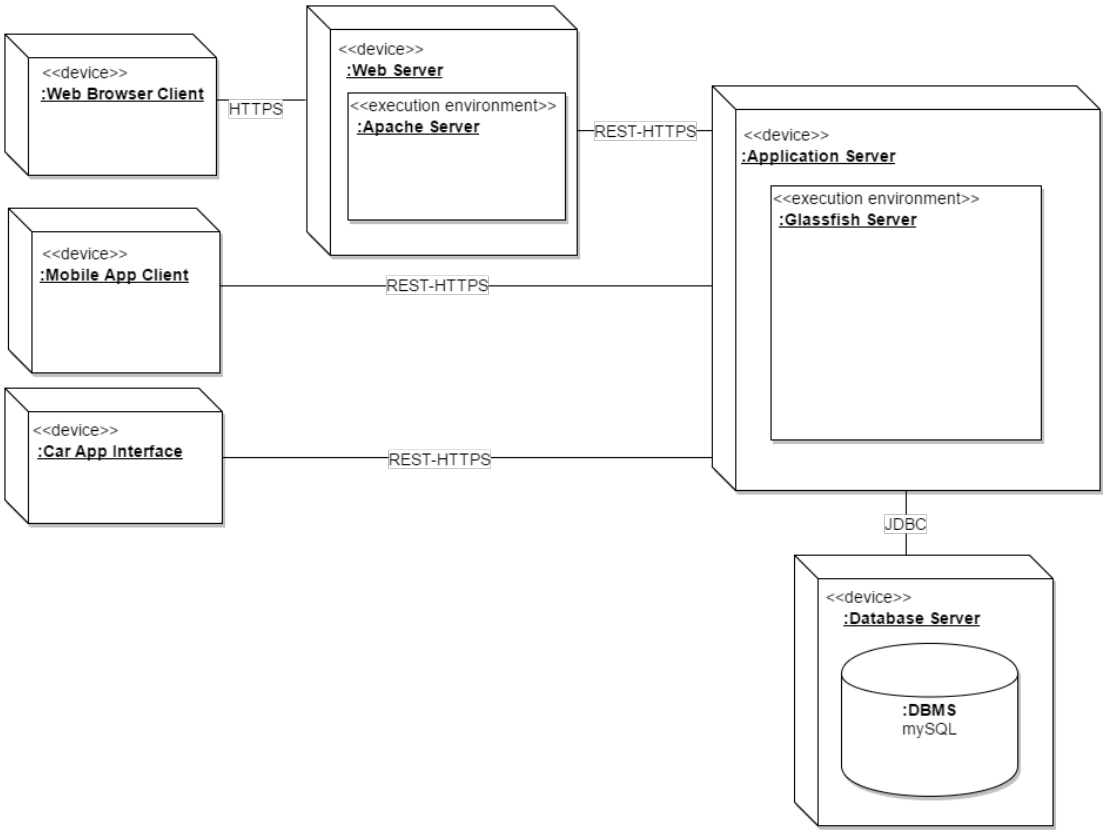
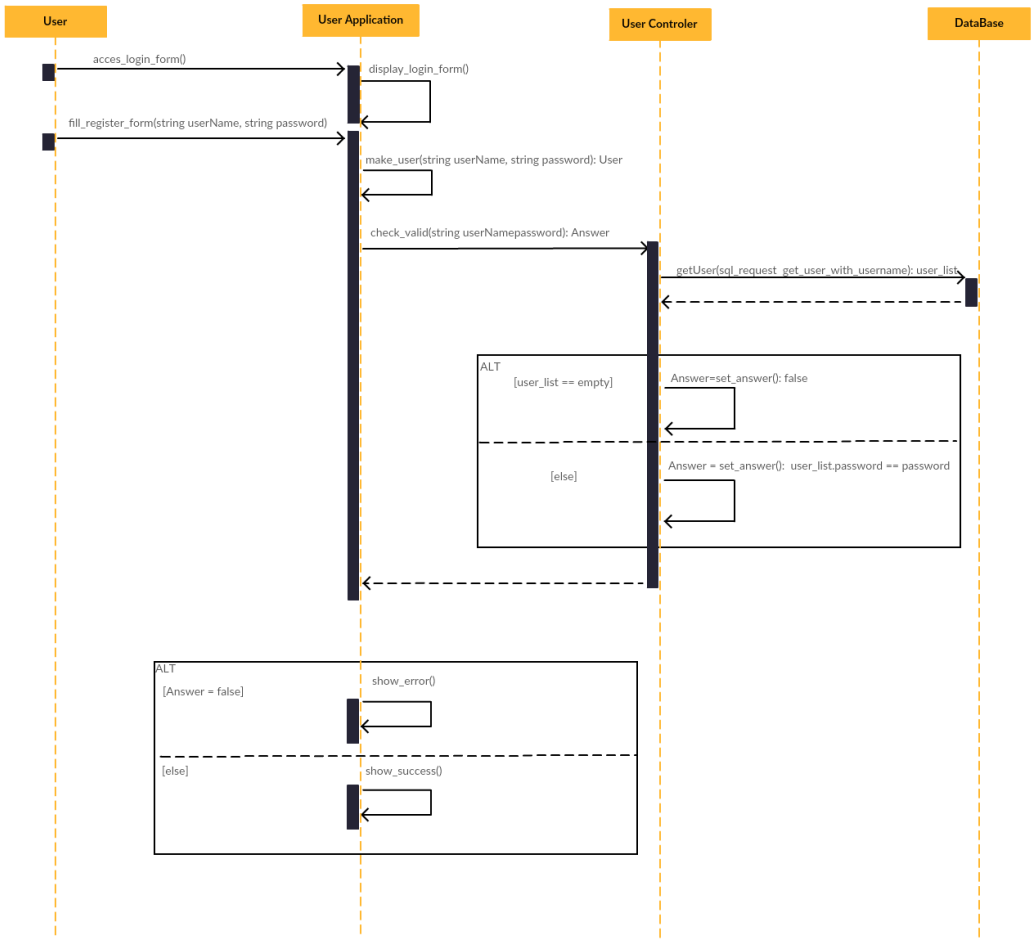


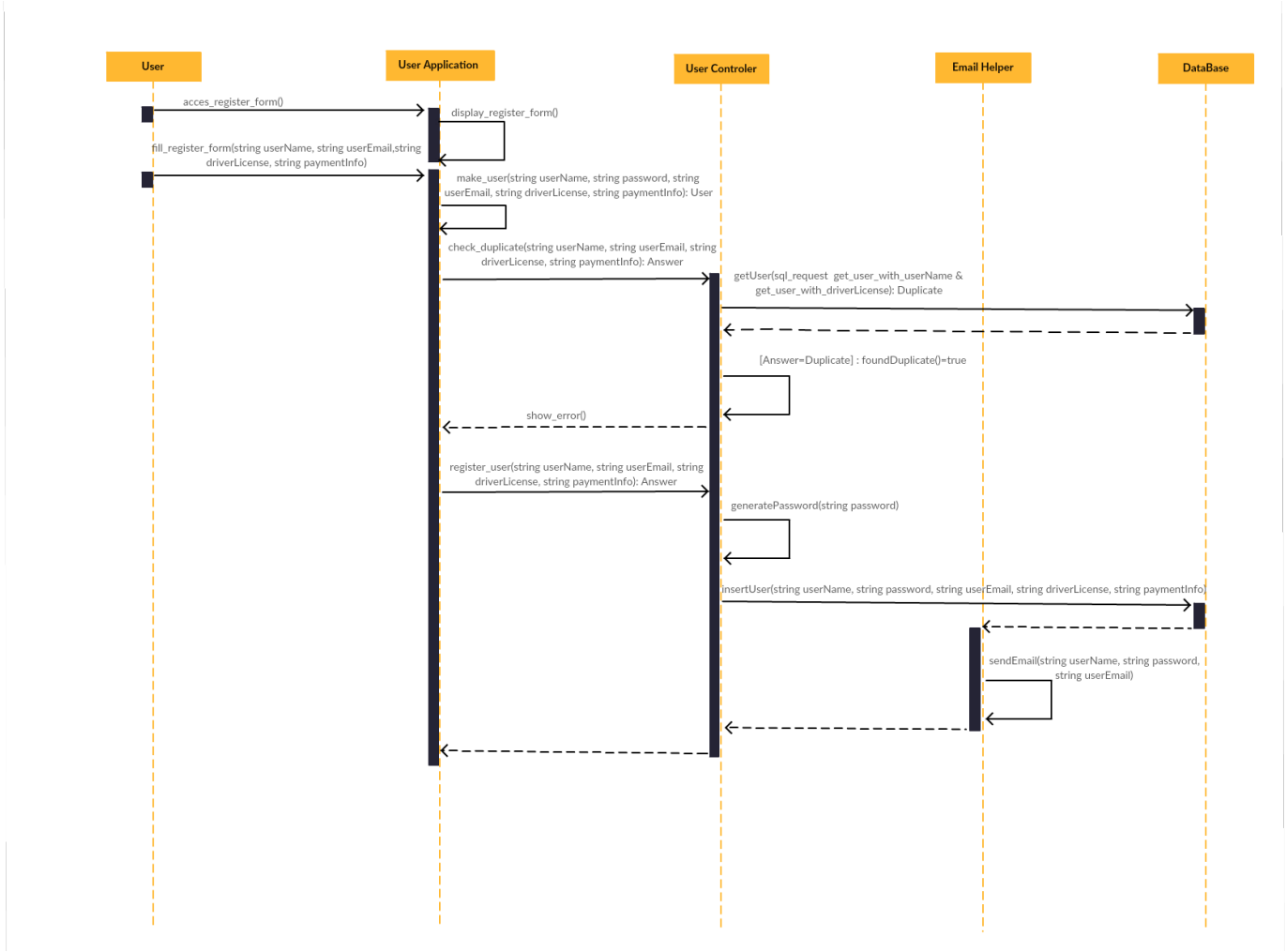
Figure 8: Deployment View

2.5 Runtime View

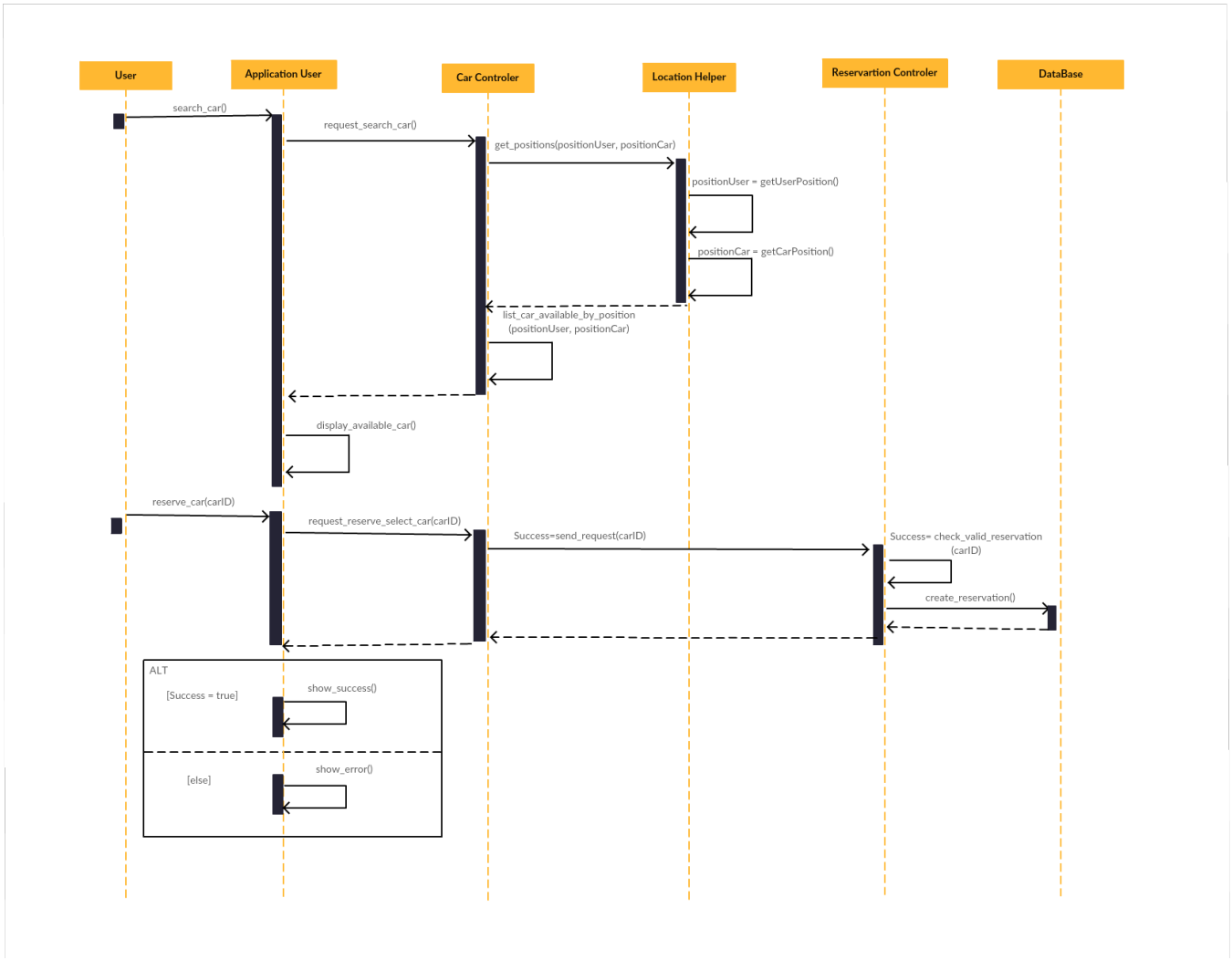
2.5.1 Runtime View Diagram for User's login



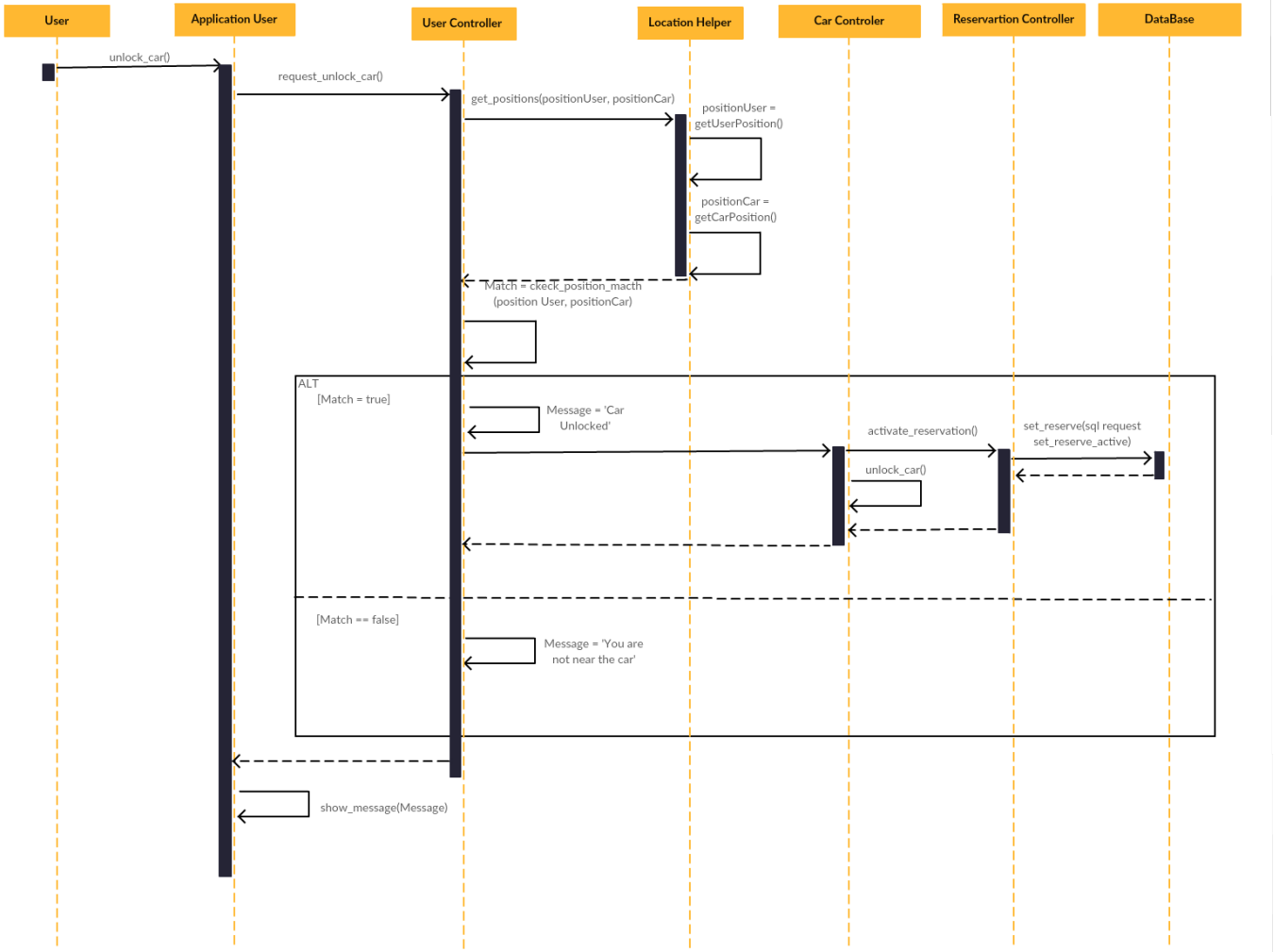
2.5.2 Runtime View Diagram for User's registration



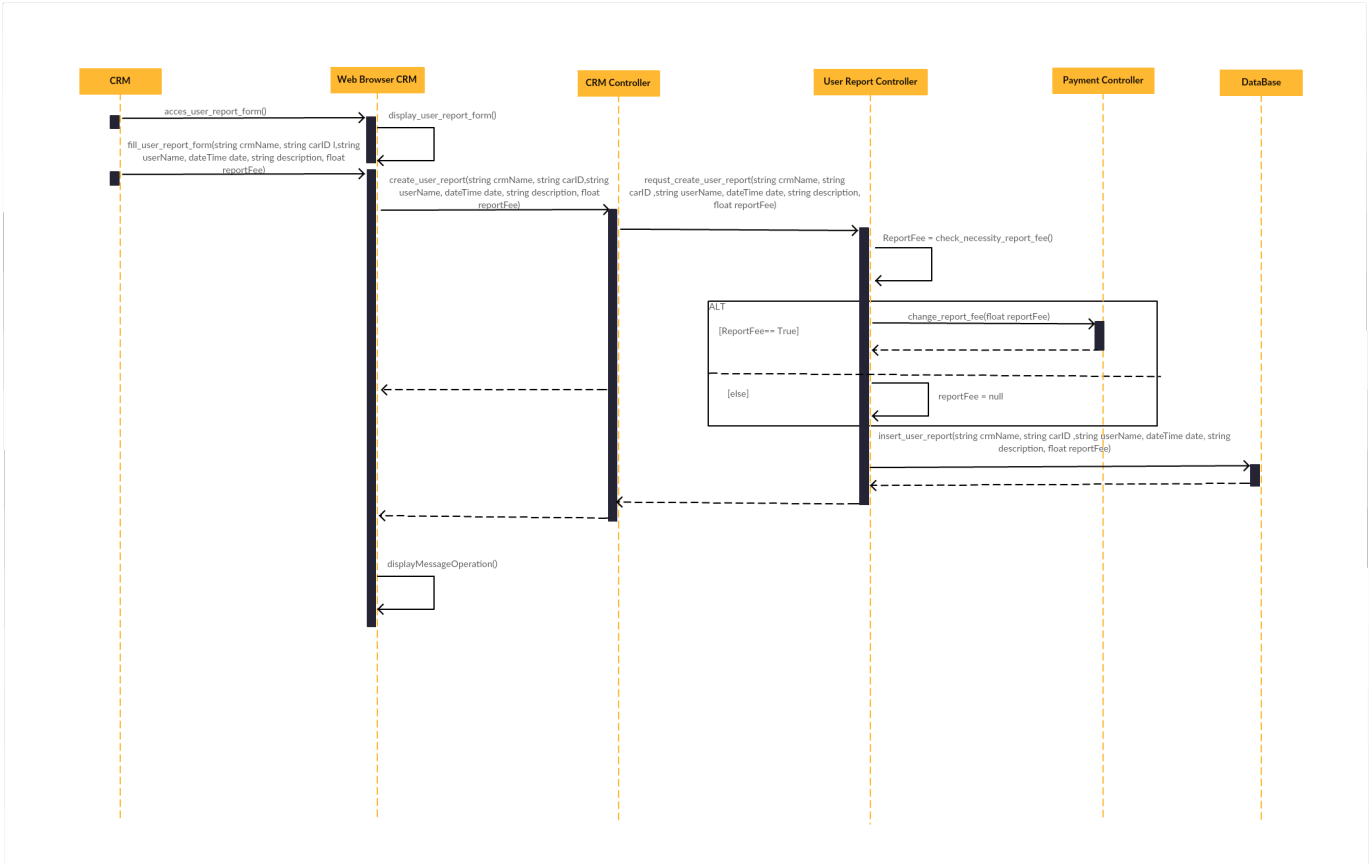
2.5.3 Runtime View Diagram for the reserve of the car



2.5.4 Runtime View Diagram for unlock the car

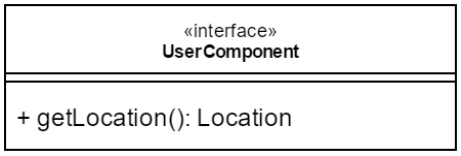


2.5.5 Runtime View Diagram where CRM create a Report of an User

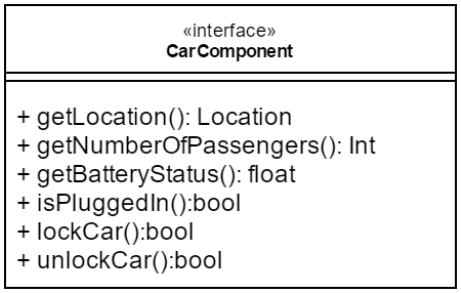


2.6 Component Interfaces

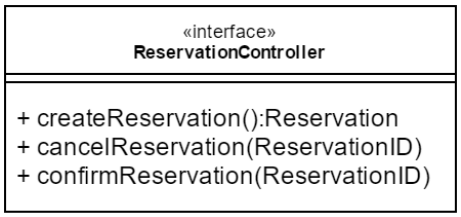
2.6.1 User Component



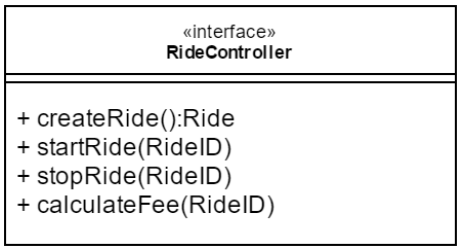
2.6.2 Car Component



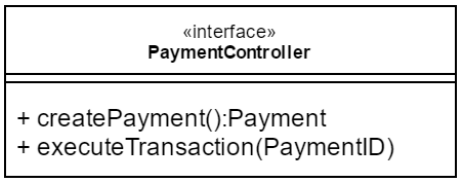
2.6.3 Reservation Controller



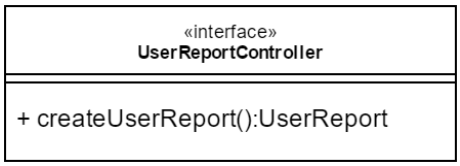
2.6.4 Ride Controller



2.6.5 Payment Controller

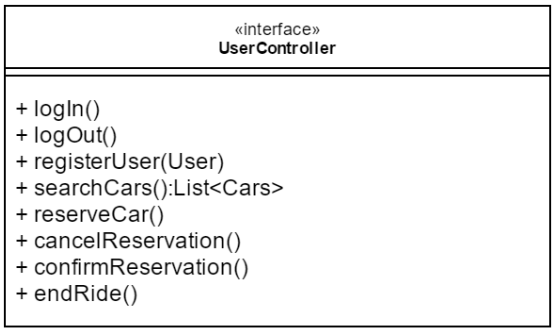


2.6.6 User Report Controller



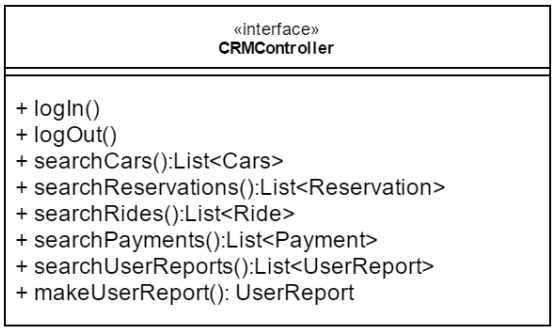
2.6.7 User Controller

Will Interface with the Users and answer requests redirecting them to the corresponding Controller.



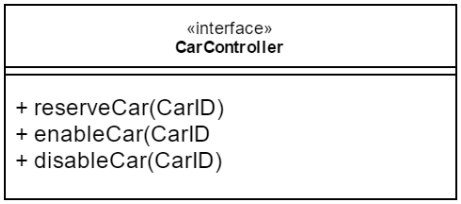
2.6.8 CRM Controller

Will Interface with the CRM and answer their requests.



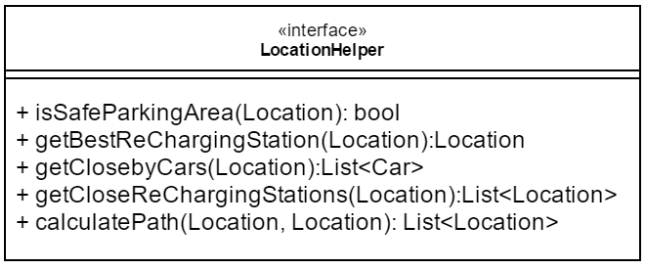
2.6.9 Car Controller

The Car Controller will interface directly to the Car so as well to communicate the Car methods will offer these other methods.

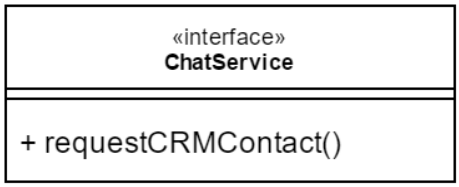


2.6.10 Location Helper

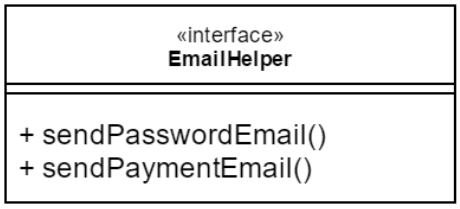
Location Helper will Implement our Location search Algorithms. It will help the System get the best ReCharging Station to ensure the best distribution of Cars in the city, as well as calculate Path and seach for nearby Cars or ReCharging Stations.



2.6.11 Chat Service



2.6.12 Email Helper



2.7 Selected Architectural Styles and Patterns

2.7.1 MVC 3 Tier Architecture

The 3 tier Architecture will be based in the MVC pattern as the Client represents the View, the Controllers will be implemented in the Application logic and the Model will be present in the Data Layer. This design allow us to present the data to our Users in a consistent and reliable way.

2.7.2 Client-Server Architecture

As the abstraction of our System is providing a service, the logical choice is to present the Client-Server model. The logic of our System implemented in our Servers and the Clients (Users and CRMs) will communicate with them to access our service. The simplicity and effectiveness of this model facilitates the deployment. Contrary to peer-to-peer

2.7.3 Communication: REST, HTTPS, JDBC

- Basing our communication patterns on REST will allow simplicity for the Client requests, mantaining persistence on our data, this helps keeping the communication stateless.
- REST over HTTPS is the most common practice and HTTPS is as well the standard secure protocol of communication on the Internet.
- Since we are deploying our App with Java EE, JDBC is the Java standard of communication with the DBMS.

2.8 Other Design Decisions

3 Algorithm Design

3.1 Making new Reservation

Although this is not a complex algorithm, it is one of the main in the car sharing system. The goal in describing this is to rule out the other possibilities of implementation, in order to set the way smaller functions, that depend on the car reservation to operate, will handle their own tasks. The model that will be shown here has already been tested in the RASD file and was proved solid.
Until the proper function is called the user will have to make a few steps:

- From the home screen, the user will select a car near his position or near the address he/she entered.
- When a car is selected the user will see relevant information of the car, to help him decide whether or not make the reservation.
- After the user has select the car and confirmed the reservation the algorithm that will be displayed is called.

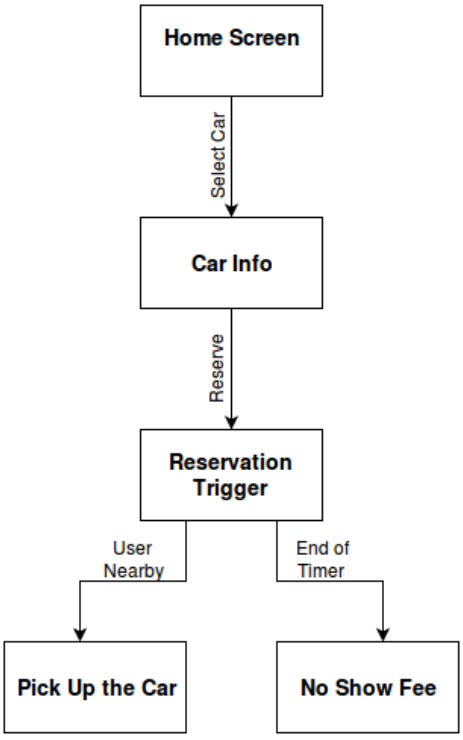


Figure 9: Fluxogram for making a new reservation. "Cancel" steps were omitted but are possible every time.

Algorithm 1 Making a new Reservation

1: **procedure** MAKERESERV($Car, Account, ListOfReservations, listOfAvailableCars$)
▷ no user or car in two reservations
2: **if** $Account \vee Car \subseteq ListOfReservations$ **then**
3: ShowErrorMessage
4: **else**
5: $Reservation \leftarrow CreateReserv(Car, Account)$ ▷ Sets the variables for the data structure
6: $ListOfReservations \leftarrow ListOfReservation + Reservation$
7: $listOfAvailableCars \leftarrow listOfAvailableCars - Car$
8: CounterTrigger($Reservation$) ▷ Starts Counter
9: **end if**
10: **return**
11: **end procedure**

The counter will work with interruptions. We have three main fases:

- User picks up the car in time, counter is ended.
- 45 minutes has passed, notification is sent to the user.
- 1 hour is reached and user is billed.

Algorithm 2 Counter Trigger

```
1: procedure COUNTER(Reservation)
        ▷ Different interruptions will trigger different parts of the function
2:     if 45Min == TRUE then                          ▷ time to issue a warning
3:         ShowWarning
4:         Wait                                                  ▷ system sleep
5:     end if
6:     if TimeFinished == TRUE then                   ▷ 1 hour has passed
7:         ShowWarning
8:         Ride ← createRide(Reservation)
9:         Ride.notPickUp ← 1
10:        FeeCalc(Ride)                  ▷ bill the user for not picking up
11:        return
12:    end if
13:    if PickUp == TRUE then                         ▷ User nearby the car
14:        StartRide(Reservation)          ▷ unlock the car, take off reservation list and put on rides list
15:        return
16:    end if
17:    Wait                                          ▷ system is put to sleep
18: end procedure
```

3.2 Calculating the fee

This algorithm will be called when the car controller pass the signal that the engine is off, the car is parked in a safe area and the user is outside the car.
The amount to be paid is calculated primarily based on the time of usage of the car and then extended for the virtuous behavior rules that add or subtract a percentage of the total amount. These rules are:

- if the system detects the user took at least two other passengers for more than 50% of the trip a discount of 10% will be applied.
- if the car is left with no more than 50% of the battery empty, the system applies a discount of 20%.
- if the car is left more than 3km from the nearest power grid or with more than 80% of the battery empty, the system charges 30% more.
- if a car is left at special parking areas and the user takes care of plugging the car into the power grid, the system applies 30% discount.

If the user parks in a special parking area, where he can plug the car into a power grid, the closing of the invoice will be delayed in five minutes, in order to give time for the user to plug the car and receive the discount.

Algorithm 3 Part 1: Fee calculator

```
1:  procedure FeeCalc(Ride)
2:      if Ride.notPickUp == TRUE then
3:          Ride.Total ← 1                                  ▷ 1 euro fee
4:          Ride.Active ← FALSE                             ▷ Boolean
5:          MarkCarAsActive(Ride.Car)   ▷ car will be available again
6:          PaymentHelper(Ride)                 ▷ Will charge the user
7:          EmailHelper(Ride)             ▷ Sent the invoice by e-mail
8:
9:      else
10:         timeRide ← Ride.TimeOfDropOff − Ride.TimeOfPickUp
    ▷ Time is in minutes
11:         timeFee ← timeRide ∗ pricePerMin
12:
13:         if Ride.PassengerDisc == TRUE then
14:             Disc ← Disc − 0.10                ▷ Disc is initialized as 1
15:         end if
16:
17:         if Ride.BatteryDisc == TRUE then
18:             Disc ← Disc − 0.20
19:         end if
20:
21:         if Ride.RechargingStationDisc == TRUE then
22:             Start5MinCounter()   ▷ the user will be given 5 minutes
    to plug the car
23:             if Ride.Car.plug == TRUE then
24:                 Disc ← Disc − 0.30
25:             end if
26:         end if
```

This algorithm was divided in two parts: In the first the "not picking up the car" condition is evaluated, the time fee is calculated and the discounts are added; In the second it is evaluated the bad usage of the car and the bills are charged.

Algorithm 4 Part 2: Fee calculator

```
27:         EvaluateHarshConditions(Ride) ▷ Check if the car is more than 3 km of the nearest plug or it's with less then 20% of battery
28:
29:         if Ride.HarshConditionsFee == 1 then   ▷ User is charged extra for the trouble of recharging onspot
30:             Disc ← Disc + 0.30
31:             SignalizeCrm(Ride)     ▷ Send message to crm crew that the car needs special attention
32:             Total ← timeFee * Disc
33:             PaymentHelper(Ride)
34:             EmailHelper(Ride)
35:             return
36:         end if
37:
38:         Ride.Total ← TimeFee * Disc
39:         PaymentHelper(Ride)
40:         EmailHelper(Ride)
41:         MarkCarAsActive(Ride.Car)
42:         return
43:     end if
44: end procedure
```

3.3 Money-Saving

When the money-saving option is enabled the system will suggest a power grid for the user to go in order to receive a discount. The algorithm for choosing the best place to send the user will take into account:

- The destination of the user, as no station within more than 1km will be suggested. In case there are no stations inside the 1 km radius, the system will apologize and show the nearest one.
- The suggestion will check if there are power plugs available in the location. In case they become unavailable after the start of the ride, the user will still receive the discount, as it is impossible to predict.
- The algorithm will take in consideration the uniform distribution of the cars in the city.

The uniform distribution can be implemented in various different ways. As in the real world problem, it is impossible to have a uniform distribution (every point of the city has the same probability of having a car). But what can be done is to calculate the standard deviation for all the possible points and choose the destination that will increase it the most,

as this means that the dispersion of the cars among the city are being broadened. And thus, staying a bit closer of the uniform distribution.
As would be slow to calculate correctly the standard deviation every time, samples with a percentage of randomly chosen cars will be used instead.

Algorithm 5 Money-Saving

```
1: procedure MoneySaving(Address, listOfAvailableCars)
2:     Candidates ← PossiblePowerGrids(Address)      ▷ will look for power grids within 1km of distance, returns a list
3:     if Candidates == EMPTY then
4:         Suggest ← SearchPowerGrid(Address)      ▷ return nearest Available power grid
5:         return Suggest
6:     else
7:         Suggest ← StdDeviation(Candidates, listOfAvailableCars)      ▷ calculate and return the best
8:     end if
9:     return Suggest
10: end procedure
```

For the standard deviation, the correct formula would be:
$$\sqrt[2]{\frac{\sum((x-\overline{x})^2+(y-\overline{y})^2)}{n-1}}$$
But, as what is important for the decision is only the difference between the values, it's possible to notice in the following algorithm that the formula was simplified.

Algorithm 6 Standard Deviation

```
1: procedure StdDeviation(Candidates, listOfAvailableCars)   ▷ Candidates is a list of all possible locations
2:
3:     sampleSet ← ChooseCars(listOfAvailableCars)   ▷ return a ramdom sample 2d-array with a fixed number of locations.
4:
5:     for loc in sampleSet do
6:         mean_x ← mean_x + loc.x
7:         mean_y ← mean_y + loc.y
8:     end for
9:                 ▷ still needs to divide by the number of elements in the sampleSet
10:    mean_x ← mean_x ÷ |sampleSet|
11:    mean_y ← mean_y ÷ |sampleSet|
12:
13:    for Item in Candidates do
14:        sampleSet ← sampleSet + Item
15:
16:        for loc in sampleSet do
17:            deviation ← deviation + ((loc.x − mean_x)^2 + (loc.y − mean_y)^2)   ▷ only the part that matters from the formula is used
18:        end for
19:
20:        sampleSet ← sampleSet − Item
21:        deviation ← 0
22:
23:                        ▷ savedDaviation starts with negative infinite
24:        if deviation > savedDeviation then
25:            savedDeviation ← deviation
26:            Suggest ← Item
27:        end if
28:    end for
29:    return Suggest
30: end procedure
```

4 User Interface Design

For our User Interface (UI) we'll offer a mobile App for Users and a desktop web App for Users and CRM. They will offer:

- **User experience diagram**

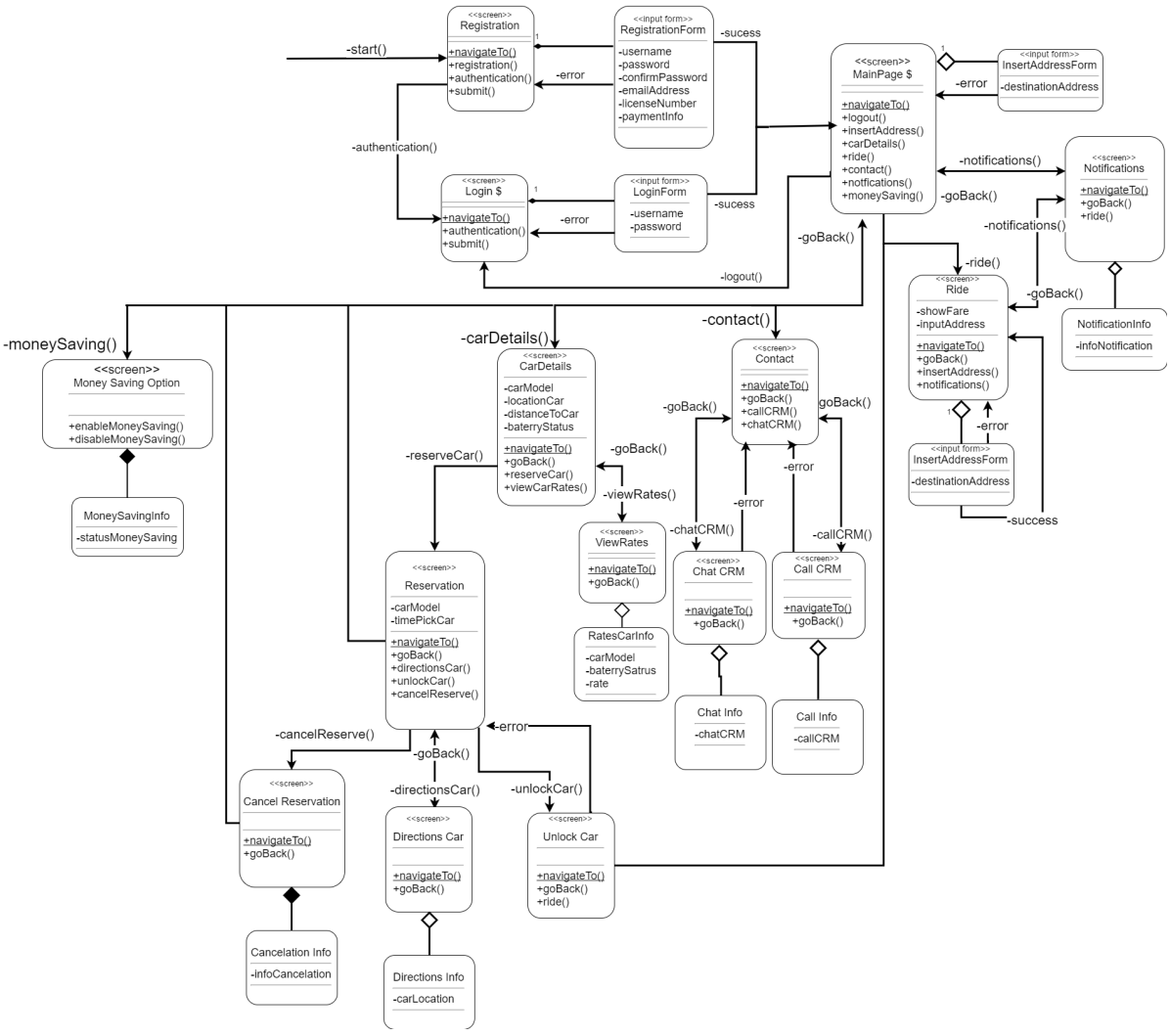


Figure 10: User experience diagram for the User/Registered User.

- **CRM experience diagram**

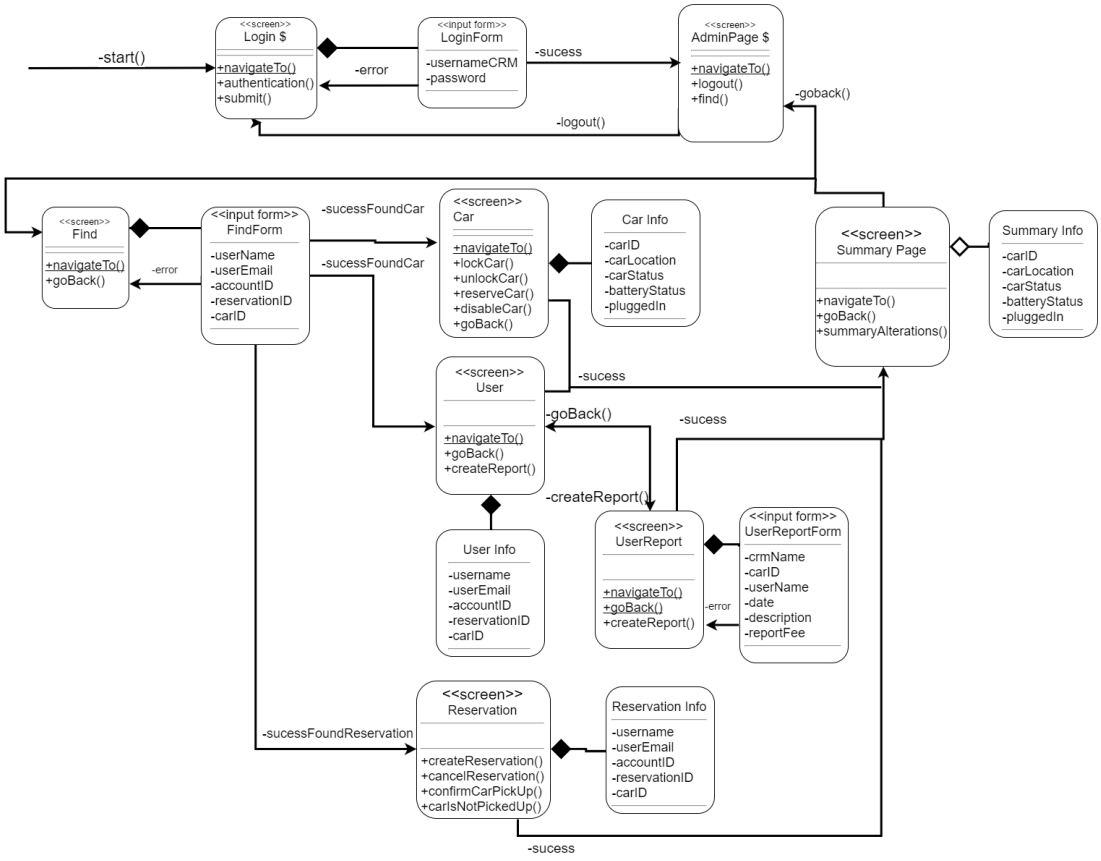


Figure 11: User experience diagram for the CRM.

- MockUps for the MobileApp UI are presented in the RASD in section **4.2.1**.
- Following some WebApp UI examples not yet presented.

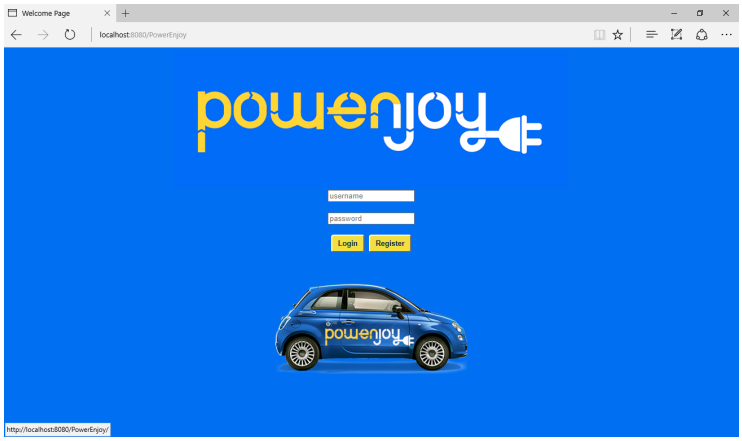


Figure 12: Browser LogIn Page

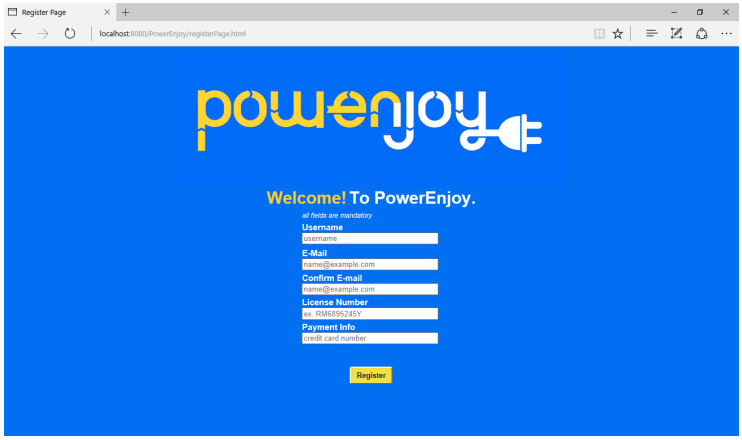


Figure 13: Browser Register User Page

5 Requirements Traceability

5.0.1 User Requirements:

G.1) First-Time Users must be able to register to the System creating an Account.

- UserController will answer for the Registration Request andl add the User Account in the DB.
- EmailHelper will send the Email with the Password to the User Email.

G.2) Registered Users must be able to login to their Account at any time they want.

- UserController will allow the Login Request checking the DB for the correct credentials.

G.3) Users can save his/hers credentials in the System.

- UserComponent in the Client side can be set to send the LogIn request automatically to the UserController whenever the App is opened.

G.4) A Registered User will be able to make a Reservation of any available Car near his/her current location or from an address that she/he can specify.

- UserController will answer the Search Car Request sending it to the CarController.
- CarController will check all the available Cars and using the LocationHelper it will answer with the ones close to the User Location.
- A Reservation Request will be forwarded to the Reservation-Controller that will create it and add it to the DB.

G.5) Users that have made a Reservation must be able to notify the System when they are nearby the Reserved Car so the System can unlock it.

- UserController will control with the LocationHelper if the User and Car Location match.
- ReservationController will be asked to activate the Reservation.
- CarController will unlock the Car.

G.6) A User that has made a Reservation must be able to cancel it before 1 hour starting from the time when the Reservation was made.

- UserController will forward the Reservation Cancelation request to the ReservationController.

G.7) In case an User hasn't started using the Reserved Car at 45 minutes after the Reservation was made, he/she will be notified that either if the Reservation is not canceled or the Car is not used in 15 minutes, the Reservation will be automatically canceled and a 1 Euro fee will be charged to her/his Account.

- ReservationController will control for pending Reservations the start time. If it surpass the 45 minutes it will notify the EmailHelper to send the User a notification.

G.8) When Users start using their Reserved Car, they must be able to see their current expenses on the service through a System screen inside the Car.

- The CarComponent Screen will request the RideController the status of the Ride to show the expenses.

G.9) The User must be able to know where the safe parking areas are nearby his/her current location or any address that she/he can specify.

- UserController using the LocationHelper will request for Safe Parking Areas new the User.

G.10) Users must be able to finish their use of the Car when leaving it in a Safe Parking Area and exiting the car. The User will then be charged for the use of the service. The used Car will be locked and freed for Users to be reserved.

- The CarController is notified when the Car is stopped, ask the LocationHelper if it is in a Safe Parking Area.
- If the Car sends the End Ride Request, the RideController will the end the Ride and ask the PaymentController to generate the Payment.

G.11) The User will always be notified when any Transaction is made on his Account.

- PaymentController when executes the transaction, asks the EmailHelper to send the User the Payment details.

G.12) Notify the Users that are currently using a Car of any available discounts on their ride if they abide by the 'Virtuous Behaviour Rules' and of the extra fee in case of not respecting the facilitation of the re-charging of the Car on site. These extra discounts/charges will be applied on the Total fee at the end of the ride.

- The CarComponent will present this options to the User.
- The RideController will calculate if any of them are applicable.
- CarController will check for Car passengers, battery state and if it is plugged in.
- LocationHelper will provide the Location for ReCharging Stations.

G.13) Users can activate the 'Money Saving'a option on their Ride to be notified of any nearby Re-Charging station on their arrival destination. Leaving the car at the end of the ride at this station and plugging it will register as a 'Virtuous Behavior' and will apply and extra discount when charging the User.

- LocationHelper will search for the best ReCharging Stations considering a uniform distribution of parked cars and provide their Locations.
- CarController Screen will show this ReCharging Stations nearby to the User's final destination.

G.14) Users must have the option to contact a CRM at any moment, the System must provide a Customer Service area that offers a Chat feature within the App or a phone number.

- ChatService will provide this feature.

5.0.2 CRM Requirements:

G.15) CRM must be able to log in into the System and see a list of all the Cars available, reserved, in use or unavailable.

- CRMController will send the LogIn Request to the CRM Controller to check correct credentials.

G.16) CRM has to be able to receive User Reports via chat or outside the app via phone call and be able to register it to the System.

- ChatService will find available CRM and contact it in cases of an User Request

G.17) CRM upon request form an User or any major cause can Reserve or cancel any active Reservation.

- ReservationController will provide different searchs for CRM Requests.
- A Reservation Cancelation Request can be sent from a CRM to the ReservationController.

G.18) CRM must be able to change the Status of a Car given a User Report and tag it, in case it's unavailable, if it's due to Re-Charge, Fix, or Removal.

- CarController will offer this option to CRMs.

G.19) Upon the completion of an User Report CRM must be able to apply extra fees or refund the payment to any User in case the situation demands it.

- UserReportController will allow the Creation of a User Report in the DB.
- If necessary the PaymentController will make a transaction linked with the User Report.

6 References

6.1 Used Tools

Keeping track of the Tools used during develop the DD document were:

- **GitHub:** for Version Control
- **Dia Diagram Editor:** for UML Diagrams
- **TeXworks:** for LaTex editing of this Document

6.2 Effort Spent

Date	Domenico	Caio	Matheus
27/11/16	1h	1h	1h
28/11/16	2h	-	-
29/11/16	2h	-	2h
30/11/16	3h	-	1h
31/11/16	1h	-	2h
01/12/16	-	1h	-
02/12/16	5h	2h	-
03/12/16	2h	4h	-
04/12/16	2h	4h	-
05/12/16	3h	-	-
06/12/16	-	-	-
07/12/16	-	-	8h
08/12/16	-	-	6h
09/12/16	-	4h	-
10/12/16	-	2h	-
11/12/16	-	-	-

7 Changelog

As the project and design decisions may change during the development this document is also prone to change. We'll document every version in this part.

- **Version 1.1:** 11/12/2016