Politecnico di Milano

Software Engineering 2 Project Work

PowerEnJoy Project: Software Design Document

Version: 1.0

Autohrs:

Jacopo Fantin (mat. 878723)

Francesco Larghi (mat. 876928)

Professors:

Elisabetta Di Nitto

Luca Mottola

December 11, 2016

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to give more technical details than the RASD about PowerEnjoy system. We want to provide a complete description of the system specified giving more technical details to allow the practical understanding of the software development. We will focus more on the components of the system and how they interact, which is their high level architecture. Then we will specify wich protocols the System will use in order to let the different components or tiers communicate to each other. We have both narrative and graphical documentation of the software design, including High-level diagrams, UX diagrams, Entity-Relation diagrams, Component diagrams, and other information.

1.2 Scope

This Design Document (also called DD) will examine more in depth the project PowerEnjoy, a digital management system for a car-sharing service that exclusively employs electric cars. The aim is to give more information about the design choices compared to the RASD, that you may refer in order to have more details on our scope. However this document will not be too detailed, but will examine architecture and protocols only generally speaking, without explaining each of them.

1.3 Definitions, Acronyms, Abbreviations

- **Java EE:** Java Enterprise Edition 7, a widely used computing platform for web based software like PowerEnJoy
- **DD:** Design Document.
- **RASD:** Reqirements Analysis Specifications Document.
- **Component:** Software element that implements functionalities.
- **RESTful API:** Representational state transfer web services are one way of providing interoperability between computer systems on the Internet.
- **MVC:** Model View Controller, a software design pattern to create well separeted software.
- **API:** Application Programming Interface.
- **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.

1.4 Reference documents

- Our PowerEnjoy Requirements And Specifications Document
- The specification document: Assignments AA'16-'17.pdf
- IEEE Std 1016tm-2009 Standard for Information Technology - System Design - Software Design Descriptions.
- Examples of Design Documents available

1.5 Document structure

Section 1 The introduction with this document's purpose and other general information.

Section 2 The high-level view of our system, describing the components from different points of view and their interaction. Then there is an explanation about the selected architectural system and design pattern.

Section 3 The list of algorithms that should be important for the development of our System. They are described using pseudocode.

Section 4 The details about the GUI (Graphical User Interface) and so how to User interacts with the System. This section is useful to the reader to get an idea on how the final application will look like.

Section 5 The link between requirements defined in the RASD and the design elements that have defined in this document.

Section 6 The amount of hour we spent writing this document.

Section 7 External references, such as tools used to create this document.

2. Architectural Design

2.1 System architecture overview

We figured out the software architecture as a 4-tier structure, according to the Java Enterprise Edition model. Therefore, the software modules are divided into four hierarchical layers:

- The **Client Tier**, that runs on the client machine (web client or mobile application) and provides the direct interaction interface with the user or guest;
- The **Web Tier**, which runs on the JEE Server, hosts the Java Servlets that dynamically receive the client requests and process the corresponding responses;
- The **Business Tier**, also run on the server machine, contains the Enterprise JavaBeans (EJB) that implement the logic required for a particular function, such as payments or reservations;
- The **Enterprise Information System Tier**, that concerns external application needed for the system of interest, such as databases, and is executed on a different server machine.

Our intention is to shift the balance center on the Business Tier so that the majority of the computation is done remotly instead of being up to the local machine. This way, we provide a light client application which doesn't run any peculiar algorithm and assures only the direct approch with the user.

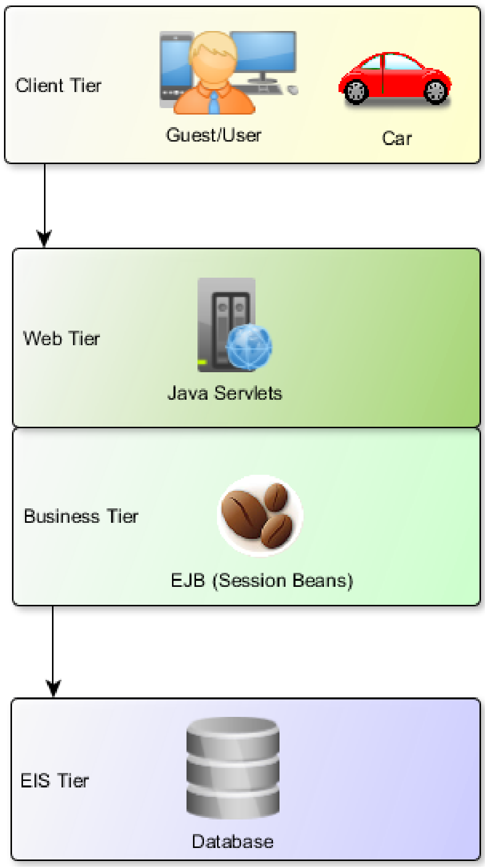


Figure 2.1.1: PowerEnJoy system architecture scheme.

2.2 Overview: High level components and their interaction

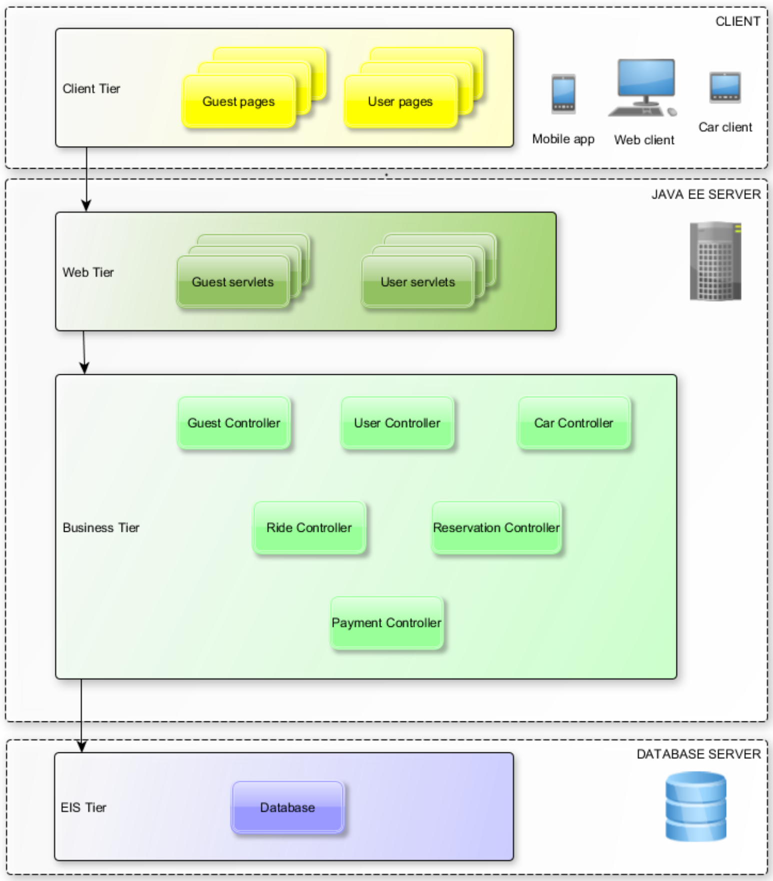


Figure 2.2.2: The High Level Components scheme derived from the system architecture.

2.3 Component view

Client components

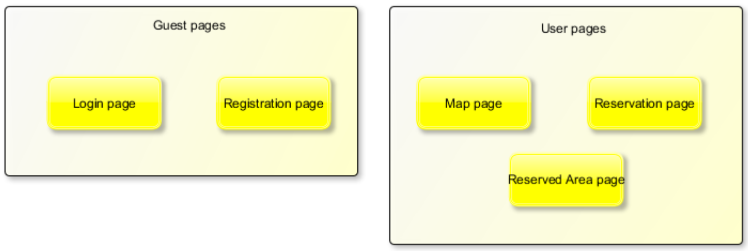


Figure 2.3.3: Client components and their subcomponents.

Web components

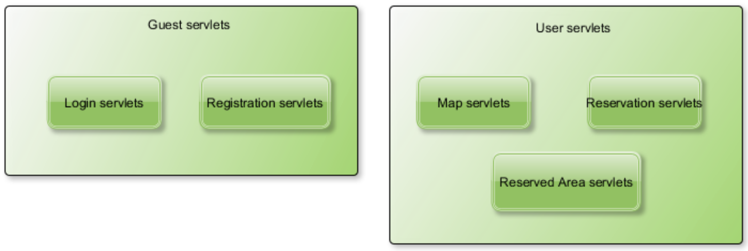


Figure 2.3.4: Web components and their subcomponents.

Business components

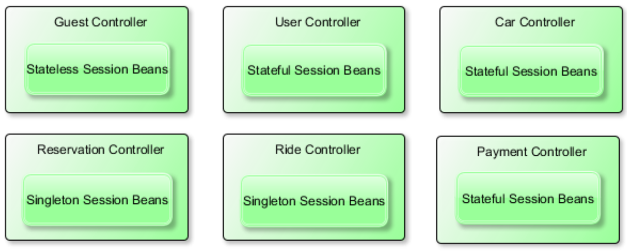


Figure 2.3.5: Business Tier components and their subcomponents.

2.3.1 Database component

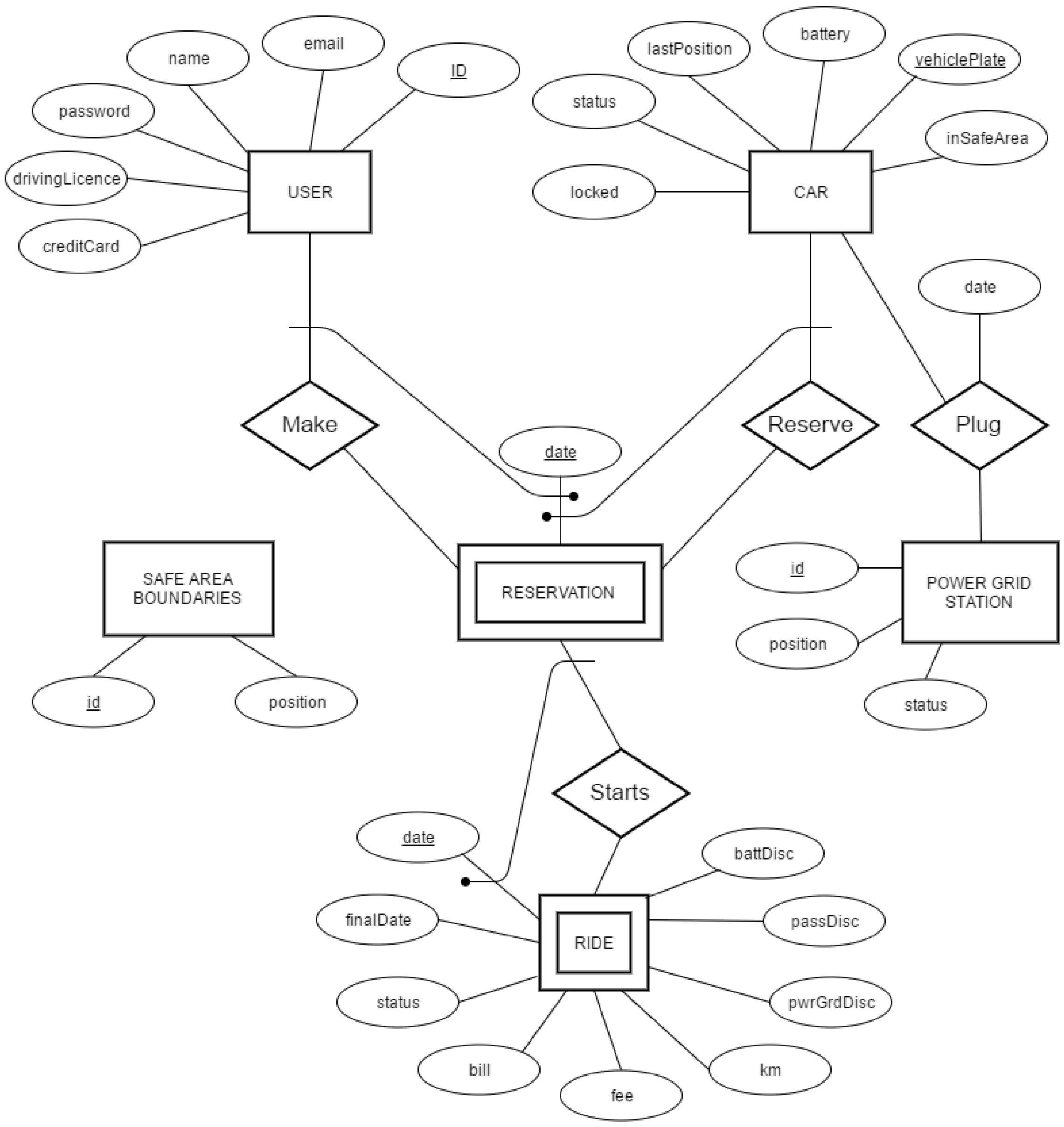


Figure 2.3.6: Database architecture represented through a E/R diagram

2.4 Deployment view

2.5 Runtime view

2.6 Component interfaces

2.7 Selected architectural styles and patterns

2.7.1 Architecture and protocols

Our system will be divided into 3 tiers:

- **Client tier** : the graphic interface for the User to the system
- **Logic tier** : the Business Logic Layer that deals with all the logic and interactions between Client and Database
- **Database tier** : stores all the persistent data

The Client tier will be implemented with an Android application, therefore with Java and Android libraries and SQLite to store few data in the mobile device. However it will be very thin and it will communicate with the Logic tier through RESTful API with JSON. We will use GET, PATCH, PUT and POST methods offered by this protocols. The Logic tier will be a server with Java Enterprise Edition 7 implementing the logic and all the main tasks.. The Database tier (receiving queries by the server with JDBC) will be handled by a MySQL DBMS.

2.7.2 Design patterns

We will use these main design patterns for our System:

- **MVC** Model-View-Controller pattern has been widely in our application.
- **Adapter** Adapters are used in our mobile application to adapt the Driver interface to the RESTful API one.
- **Client-Server** The application is strongly based on a Client-Server communication model. The clients being the taxi drivers mobile application, the client's mobile application and client's web browsers. The clients are thin, thus to let the application run on low-resources devices.

3. Algorithm design

Here we are giving only a general idea of the possible algorithms that could be useful in our PowerEnjoy system. All of these should be implemented in the buisness logic tier, therefore in the Application Server with Java EE.

3.1 Reserve a car

This is an algorithm on how to create a new reservation for a generic couple User-Car. This is not a crucial algorithm for its complexity but for the time that it's called. This is the main functionality of the system and so it should be well developed and efficient.

Algorithm 1 Reserve a car

```
function RESERVECAR($User, Car, Reservations, ReservedCars, AvailableCars$)
    if $Car \| User \in Reservations$ then
        ErrorMessage
    else
        $Reservation = NewReservation(Car, User)$
        $Reservations+ = Reservation$
        $ReservedCars+ = Car$
        $AvailableCars- = Car$
    end if
     return $Reservation$
end function
```

3.2 Verify position

This is one of the most important point in our System. Firstly, we want that Users sernely drive without worry about where is the Safe Area. In our system, we only have a big and continous Safe Area and we only want that Users are informed if they are still in the Safe Area or not. There are not buttons or triggers to search it, because they would be useless. The Safe Area is drawn and visible on the car interface map, that is also the gps navigator. Anyway, there is an alarm on the Car interface that will show if you are still in the area or not.
In our System we only have in database the boundaries of our unique Safe Area. This is because we imagine that it is a big area containing a lot of positions and it would be almost impossibile to store all of these positions in our Database. To solve this problem we store only some ordered posistions that will delimit our area: the boundaries. Available cars are supposed to have a fixed posisiton, as Reserved Car or Unavailable. On the other hand, Used Car continue to change their posisiton while moving, so it will be not stored in the database every second, but it will be verified from time to time (we can suppose few seconds, maybe 3 but we should change this value for better performance). If the position is contained in the Safe Area it's ok, otherwise the alarm on the car interface will be triggered and the User will know that he is not more in area where he can leave the car. This is a more complex algorithm.

Algorithm 2 Verify position

```
1: function VERIFYPOSITION($Car, SafeAreaBoundaries$)
2:     $pos = Car.position$
3:     $countRightPos = 0$
4:     $countLeftPos = 0$
5:     if $pos \in SafeAreaBoundaries$ then return $true$
6:     else
7:         for $bPos$ in $SafeAreaBoundaries$ do
8:             if $bPos.y = pos.y$ then
9:                 if $bPos.x < pos.x$ then
10:                    $countLeftPos++$
11:                else
12:                    $countRightPos++$
13:                end if
14:            else$a$
15:            end if
16:        end for
17:    end if
18:    if $countLeftPos = 1$ && $countRightPos = 1$ then return $true$
19:    else
20:        if $countLeftPos.isOdd$ && $countRightPos.isOdd$ then return $true$
21:        else return $false$
22:        end if
23:    end if
24: end function
```

3.3 Calculate the final bill

When the ride is finished and Users and his possible passengers close the doors, the System will wait 10 minutes in order to detect eventual Power Grid Station plugged. Then the bill to pay by the User will be calculated starting from the Ride's money amount and applying discounts and fees. Specifically:

- If the system detects the user took at least two other passengers onto the car, the system applies a discount of 10%.
- If the car is left with no more than 50% of the battery empty, the system applies a discount of 20%.
- If a car is left at a Power Grid Station and the user takes care of plugging the car, the system applies 30% discount.
- If the car is left at more than 3km from the nearest Power Grid Station or with more than 80% of the battery empty (less of 20% of the total battery), the system charges 30% more to compensate for the cost required to re-charge the car on-site (handled by Support System).

Algorithm 3 Bill calculus

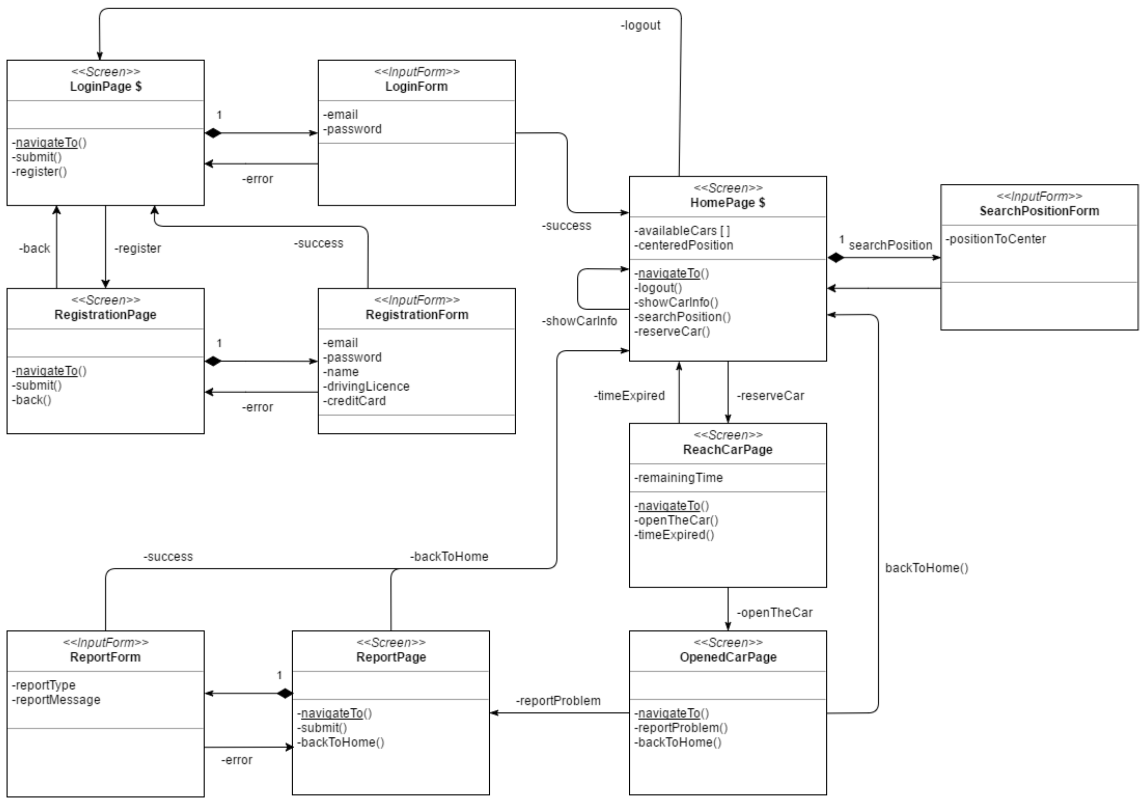
```
function billCalculus($Ride$)
    $timeRide = Ride.date - Ride.finalDate$    ▷ Difference between starting and final dates
    $tempBill = timeRide * PRICE\_PER\_MINUTE$    ▷ The temporary bill. Minute price is pre-defined constant
    $percFactor = 1$    ▷ The final percentage factor to multiply with the temporary bill
    if $Ride.passDisc == TRUE$ then
        $percFactor = percFactor - 0.10$
    end if

    if $Ride.battDisc == TRUE$ then
        $percFactor = percFactor - 0.20$
    end if

    if $Ride.pwrGrdDisc == TRUE$ then
        $percFactor = percFactor - 0.30$
    end if
    if $notEnoughDistanceOrBattery(Ride.Car)$ then    ▷ Check distance to nearest Power Grid Station and remaining battery
        $percFactor = percFactor + 0.30$
    end if
    $Ride.bill = tempBill * percFactor$
    $PaymentController(Ride)$ return
end function
```

4. User Interface Design

We have inserted an User Experience diagram (UX) for each client interface: 'mobile interface', 'desktop interface' and 'car interface'. Each diagram refers to the corresponding mockup already designed in the RASD.

4.1 UX mobile interface

This diagram shows that with your mobile device you can use all the features offered by the System until you are ready to start the ride. Then you will have to use the car interface.



-logout
<<Screen>> LoginPage $
-navigateTo() -submit() -register()
1
<<InputForm>> LoginForm
-email -password
-error
-success
-back
-register
-success
<<Screen>> RegistrationPage
-navigateTo() -submit() -back()
1
<<InputForm>> RegistrationForm
-email -password -name -drivingLicence -creditCard
-error
<<Screen>> HomePage $
-availableCars [ ] -centeredPosition
-navigateTo() -logout() -showCarInfo() -searchPosition() -reserveCar()
-showCarInfo
1 searchPosition
<<InputForm>> SearchPositionForm
-positionToCenter
-timeExpired
-reserveCar
<<Screen>> ReachCarPage
-remainingTime
-navigateTo() -openTheCar() -timeExpired()
backToHome()
-success
-backToHome
-openTheCar
<<InputForm>> ReportForm
-reportType -reportMessage
1
<<Screen>> ReportPage
-navigateTo() -submit() -backToHome()
-error
-reportProblem
<<Screen>> OpenedCarPage
-navigateTo() -reportProblem() -backToHome()

Figure 4.1.1: UX diagram for mobile User

4.2 UX desktop interface

This diagram is the same as UX mobile, but with Desktop you can't open the Car, neither report a problem. For these tasks you have to use the mobile application.

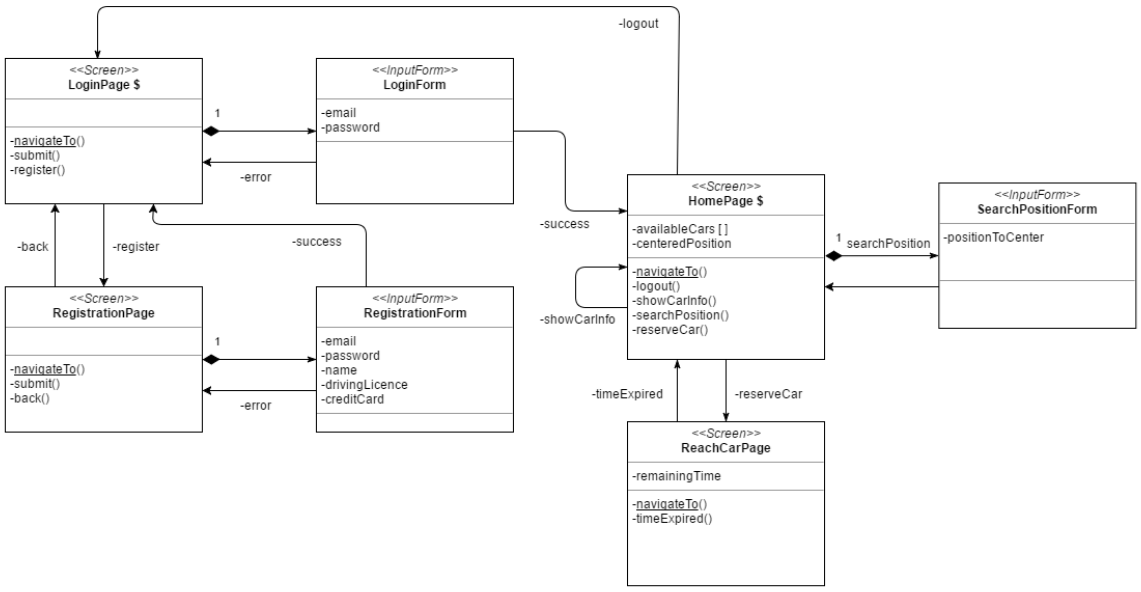


Figure 4.2.2: UX diagram for desktop User

4.3 UX car interface

This diagram shows that we have two car interfaces, one only showing the navigation map integrated in the car and the other to interact with the User. As we sayed in the RASD document, we supposed that the car is provided with 2 screens and an integrated a GPS and a navigation system.

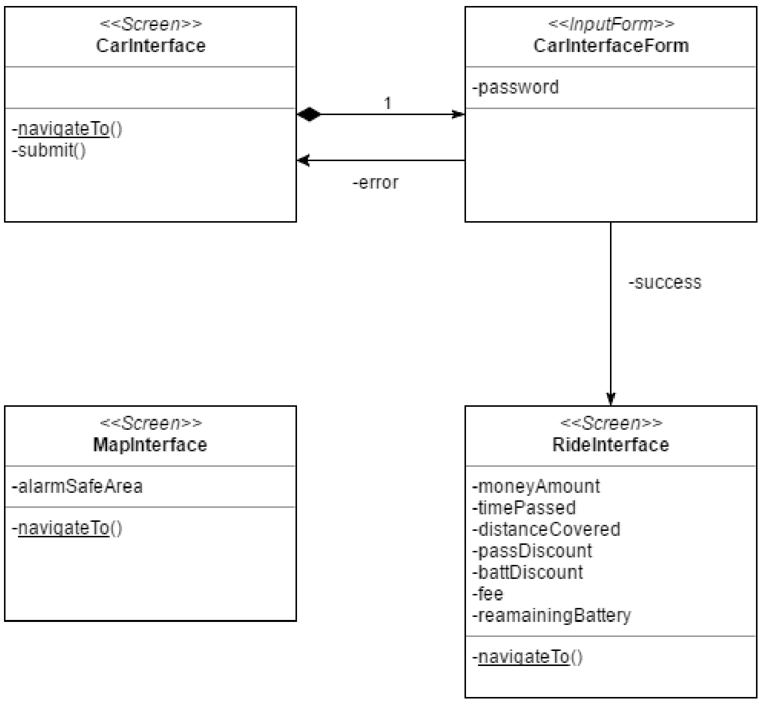


Figure 4.3.3: UX diagram for User using the car interface

5. Requirements Tracebility

G1 Allow the guest to register to the system

- Guest Controller

G2 Send a e-mail to a guest submitting a registration containing a password to access the system

- Guest Controller

G3 Allow the guest to log into the system

- User Controller

G4 Load a map indicating all the Available cars near the detected position or specified adress

- User Controller
- Map Controller

G5 Allow the user to reserve a car up to one hour before it is picked up

- Car Controller
- Reservation Controller

G6 Let the user know how much time left there is in his reservation. After one hour the reservation expires and the system applies a fee of 1 euro to the user

- Reservation Controller

G7 Allow the user open a reserved car near him

- Car Controller
- Reservation Controller

G8 Allow the user starts the car with his personal password

- Car Controller
- Reservation Controller
- Ride Controller

G9 Charge the user for a given amount of money as soon as the engine ignites and show all the details about car, ride and locations through a screen in the car

- Ride Controller
- Car Controller

G10 Lock the car and stop the time count as soon as the user leaves it

- Ride Controller
- Car Controller

G11 Wait 10 minutes to detect possible power grid connection

- Car Controller

G12 Calculate final discounts or fees and detuct money from the account and make it Available again

- Ride Controller
- Car Controller
- Payment Controller

6. Effort Spent

Last Name	First Name	Total Hours
Fantin	Jacopo	h
Larghi	Francesco	20h

7. References

The tools used to create this Design Document are:

- Github: for version controller
- draw.io: to create graphs
- DIA Diagram: to create UML diagrams
- TexMaker: to create pdf documents with LaTex
- yEd Graph Editor: to create graphs