POLITECNICO
MILANO 1863

11 December 2016

PowerEnJoy

Design Document

Blanco Federica 875487
Casasopra Fabiola 864412

Software Engineering 2 Project
2016/2017

Contents

List of Figures

1 Introduction

The main purpose of this document and the project's scope are described in this section. Moreover, we are going to give some definitions which will help the reader to better understand the content of this document and we are going to show the reference documents that have been used to redact this one. At the end, it will be explained the structure of this document.

1.1 Purpose

This document is called *Design Document* and, from now on, we will refer to it using the acronym DD. Its purpose is to provide a complete description of the system specified in the RASD, giving enough technical details to allow the proceeding of the software development. So, we must have a good understanding of which are the components of the system, how they interact, which is their high-level architecture and how they will be deployed, highlighting the design patterns we decided to use. In addition, the RASD is useful for the developers in order to figure out how to implement the entire system, thanks to the general description of the architecture and the design of the system to be built. For this reason, it must be complete and correct as much as possible. The DD contains both narrative and graphical documentation of the software design, including, for example, user experience diagrams, entity-relation diagrams, component diagrams, and other supporting requirement information.

1.2 Scope

The aim of the project PowerEnJoy is to provide a car-sharing service that involves *only* electric cars. In this document, we will give more information about the design choices for the development of this application. In order to have more details about the scope of our project, you may refer to the *Section 1* of the Requirements Analysis and Specifications Document.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

Here we present some fundamental definition:

- A Session Bean encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. To access an application deployed on the server, the client invokes the

session bean's methods. The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server. Moreover, a session bean is not persistent.

- A Stateful Session Bean is a session bean in which the instance variables (that is the state of the object) represent the state of a unique client/bean session. A session bean is not shared: it can have only one client per time. When the client terminates, its session bean appears to terminate and is no longer associated with it. The state is retained only for the duration of the client/bean session. If the client removes the bean, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.
- A Stateless Session Bean does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Because they can support multiple clients, stateless session beans can offer better scalability for applications that require large numbers of clients.
- A Singleton Session Bean is instantiated once per application and exists for the lifecycle of the application. Singleton session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.
- JavaServer Faces technology is a server-side component framework for building Java technology-based web applications. JSF technology provides a well-defined programming model and various tag libraries. The tag libraries contain tag handlers that implement the component tags. These features significantly ease the burden of building and maintaining web applications with server-side UIs.

These definitions are taken from "*Java Platform, Enterprise Edition The Java EE Tutorial, Release 7*", released by Oracle.

1.3.2 Acronyms and abbreviations

Here there is the acronyms and abbreviations list:

DD Design Document

GUI Graphical User Interface

EIS Enterprise Information System

EJB Enterprise JavaBean

ER Entity-Relationship

GPS Global Positioning System

JEE Java Platform, Enterprise Edition

JSF JavaServer Faces

HTTPS HyperText Transfer Protocol over Secure Socket Layer

MVC Model-View-Controller

RASD Requirements Analysis and Specifications Document

REST Representational State Transfer

UI User Interface

1.4 Reference Documents

- Specification document: Assignments AA 2016-2017.pdf
- IEEE Std 1016tm-2009 Standard for Information Technology - System Design - Software Design Descriptions.
- Requirements Analysis and Specifications Document: RASD.pdf (`https://github.com/fabiola-casasopra/sw-eng-2-project/tree/master/RASD/RASD.pdf`)

1.5 Document Structure

Here is presented the structure of the document, with a brief overview of each section. While the RASD is written for a more general audience, this document is intended for only people directly involved in the development of our application, as the software developers, the project consultants and the team managers. So, each person, depending on its role, can go directly and read to the section he finds more relevant.

Section 1 There is an introduction with this document's purpose and other general information about it.

Section 2 There is an overall view of our system, describing all the components from different points of view and highlighting their interaction. Moreover, there is an explanation about the selected architectural system and design pattern.

Section 3 Here there are presented the algorithm we think are more relevant for the development of the application. They are mainly described using a pseudocode implementation.

Section 4 There is a description of all the details about the structure of the Graphical User Interface. This section is useful for the reader to get an idea on how the final application will look like.

Section 5 There is an explanation of how the requirements defined in the RASD map into the design elements that have defined in this document.

Section 6 Here there are given additional information that may be useful to the reader, such as the tools used and the time spent to redact this document.

2 Architectural Design

In this section, we will show the proposed architecture for our system, that allow us to give a more complete and general idea of the entire system and a better view of the relation with external components.

2.1 Overview

Now we are going to present an overall description of the architecture of our system. We propose a 4-tier architecture, following the model of the Java Platform, Enterprise Edition architecture, shown in Figure 1. The **client**, usually a web client or an application client, runs on the client machine. Instead the **business** code, which is logic that solves or meets the needs of a particular business domain (such as banking) runs on the server machine, as it happens for the **Web-tier**. The **Enterprise Information System-tier** includes enterprise infrastructure systems, such as the database and legacy systems, and it runs on a third dedicated machine.
For sake of simplicity, in this document the web client and the mobile application are treated as one entity. For this reason, each communication between server and client will pass through the Web-tier.
JSF technology is a server-side component framework for building Java technology-based web applications and we will use it for the dynamic web pages. As far as the communication with the mobile application is concerned, we will consider an implementation of the REST paradigm.

In the following part, we will give a more accurate and detailed description of this 4-tier architecture.

- **Client-tier**: This tier component are the Application Clients and Web Browsers. They both typically have a GUI, since they interact with the actors.
- **Web-tier**: This tier component has the task to manage the requests sent by the Client-tier and to forward these requests to the Business-tier. In a similar way, the Web-tier elaborates the contents generated by the Business-tier and it sends these contents to the Client-tier in a way that this tier components can render the information received.
- **Business-tier**: This tier represents the core of the whole system. This tier components contain the logic that solves or meets the needs of a particular business domain such as banking, retail, or finance, is handled by Enterprise Java Beans. They receive data from client programs,

processes it, and sends it to the EIS-tier for storage. An Enterprise Java Bean also retrieves data from storage, processes it, and sends it back to the client program. All the application logic resides here under the form of Enterprise Java Beans and Java Entities. This tier is connected to the Database through a Java Persistence API.

– **EIS-tier**: This tier components are usually database and legacy systems, where the entire system stores the needed persistent information.

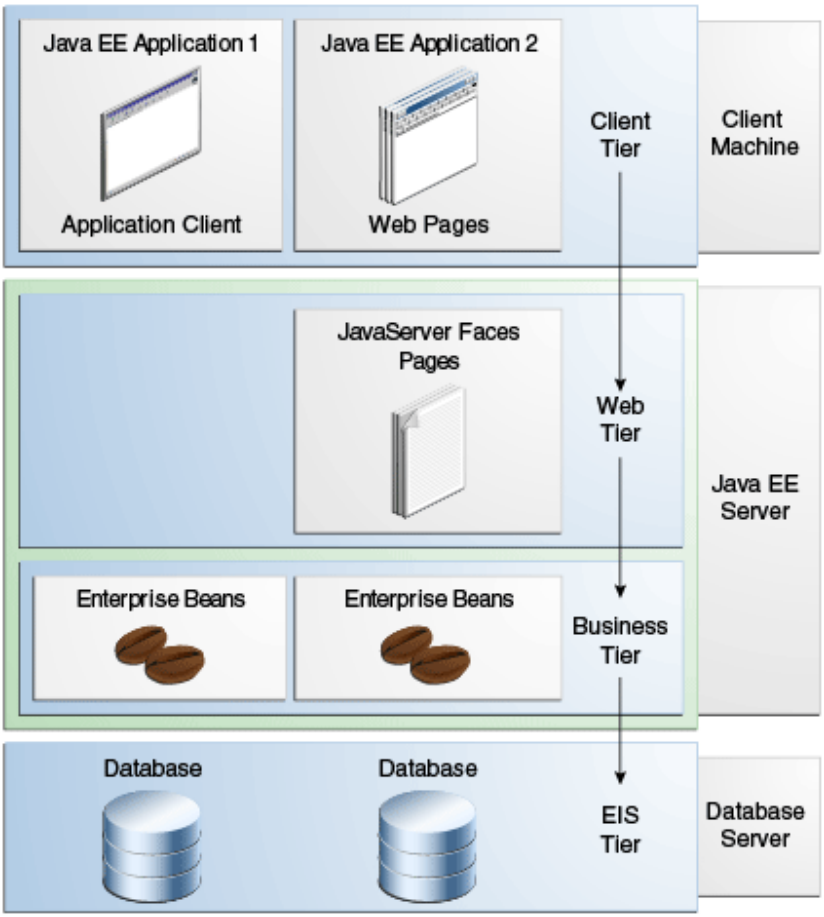

Figure 1: JEE 4-tier architecture

2.2 Interactions with External Components

The diagram in Figure 2 shows the interactions between our system, in paricular the **Business-tier** and all the external components, that are the **Mapping Service**, the **Bank** and the **Car System**. In all the three cases, the communication between the components is done using the HTTPS protocol, in order to ensure the security of the data.

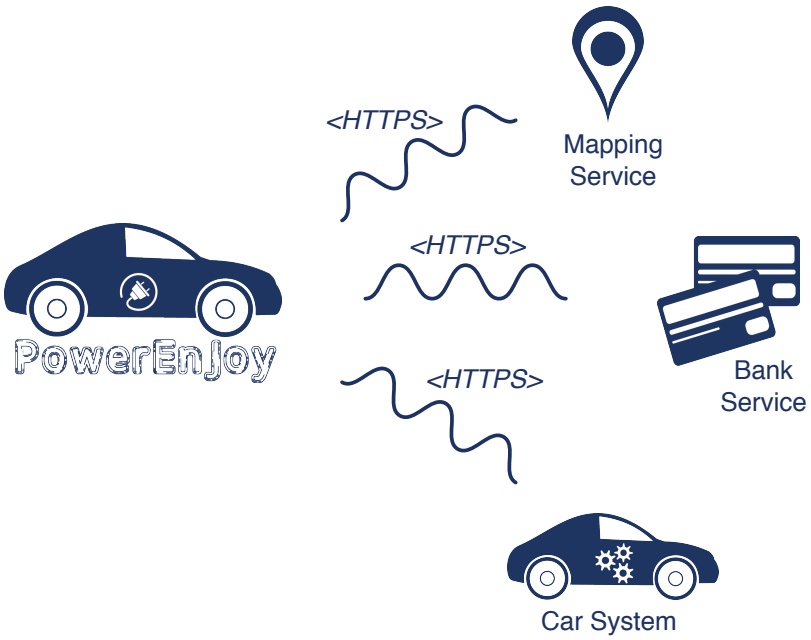

Figure 2: Interactions between PowerEnjoy and the external components

Here, we are going to show a more detailed view, but always remainig at higl-level, on how the interaction with the **Bank** is implemented.

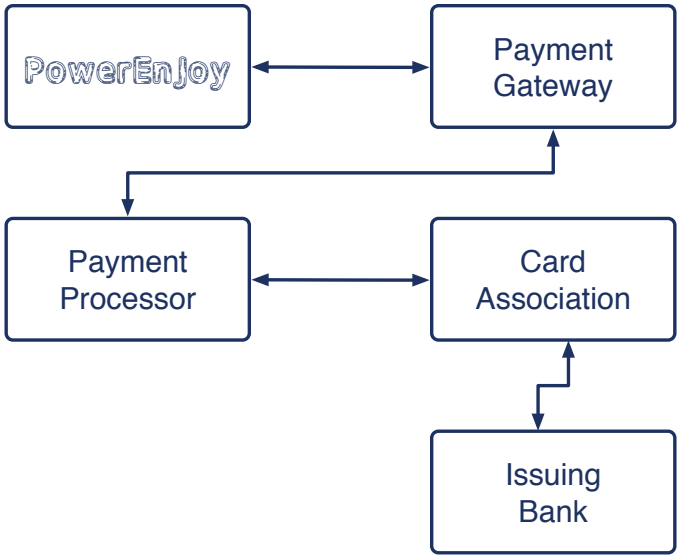

Figure 3: Interactions between PowerEnjoy and the bank

2.3 Internal High Level Components and their Interaction

The diagram in Figure 4 shows the conceptual high level architecture we propose for our system.

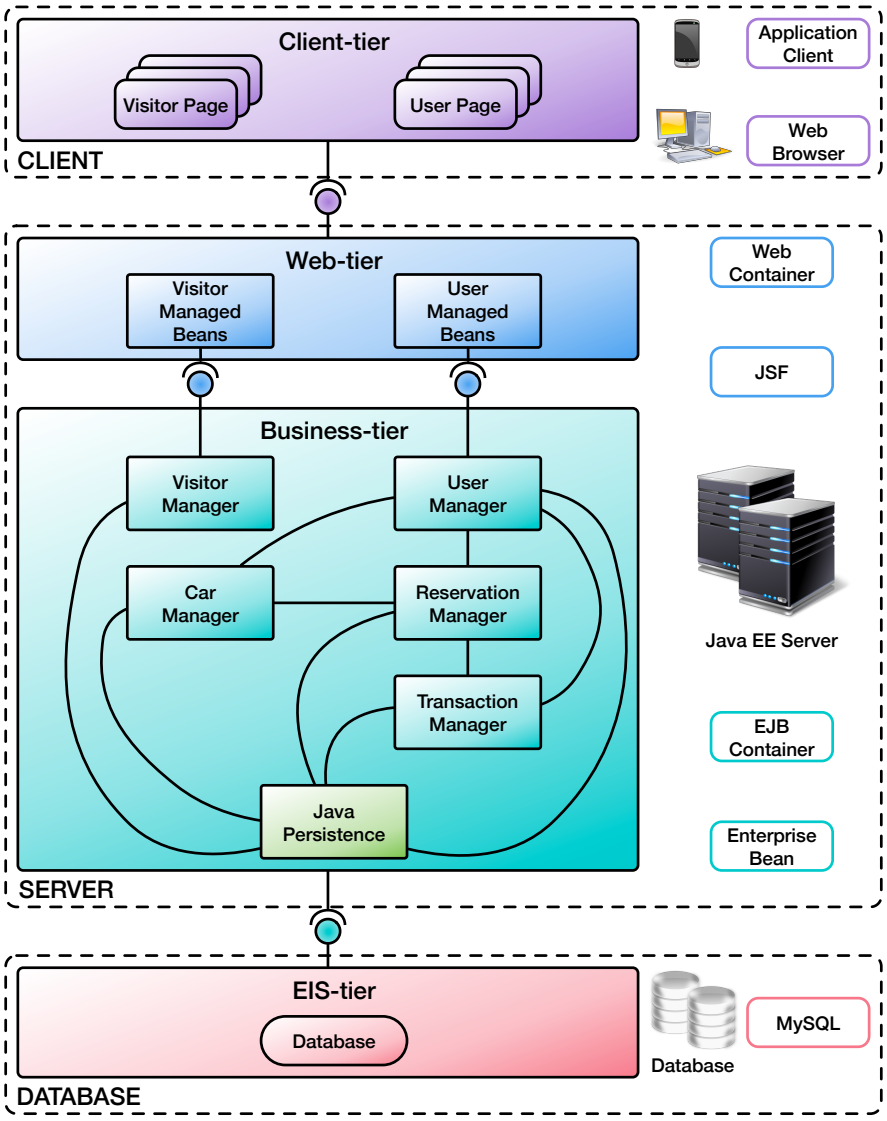

Figure 4: PowerEnjoy High Level Components and their Interactions

2.4 Component view

In this subsection, we are going to analyze more in details the component we have just introduced.

Client component

The first component we are going to analyze is shown in Figure 5: the Client component provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language (HTML, XML, and so on). The clients usually do not query databases, execute complex business rules, or connect to legacy applications: such operations are off-loaded to enterprise beans executing on the server, where they can leverage the security, speed, services, and reliability of Java EE server-side technologies. The Client component present different interfaces: each of them display the user only the pages for which he has permissions. Each interface is a subcomponent of the Client component.

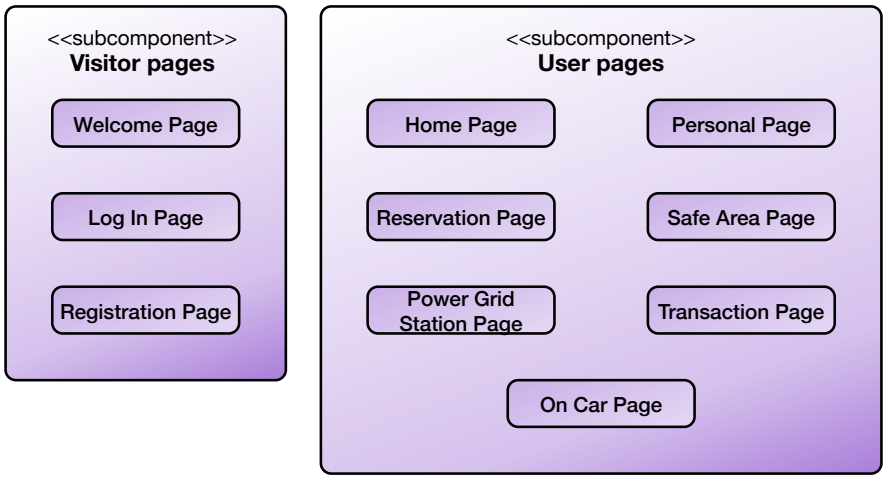

Figure 5: Client component and its subcomponents

Web component

The second component we are going to analyze is shown in Figure 6: the Web component generates dynamic web pages containing XHTML. Moreover, it implements JavaServer Faces technology, a common user interface component framework for web applications. The Web component includes also JavaBeans components, one for each group of pages, in order to manage the user input and then send that input to enterprise beans running in the Business-tier for processing.

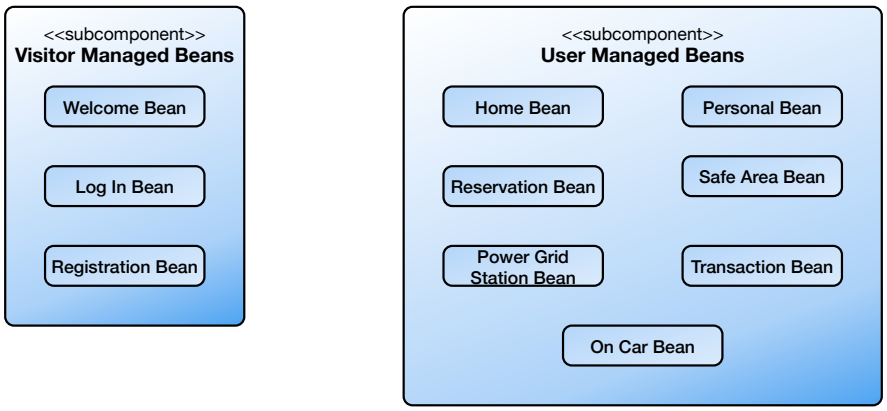

Figure 6: Web component and its subcomponents

Business Logic component

The third component we are going to analyze is shown in Figure 7: the Business component. It includes Enterprise JavaBeans that are a server-side component encapsulating the business logic of an application. An EJB is a body of code that has fields and methods to implement modules of business logic. You can think of it as a building block that can be used to execute business logic on the Java EE server. The **business logic** is the code that fulfills the purpose of the application.

Another task performed by the Business Logic Component is to transfer and processes data between the Client component and the Java Persistence Entity, which holds the information of the system data model and it has to store and retrieve information from the database.

In this document, we focus only on the fundamental elements needed in order to manage the basic functionalities. However, further details and functionality will be added during the development. Here we show the reader some of them.

- The **Visitor Manager** has to perform the following functionalities:
  - check the validity and correctness of user's information;
  - create a new user and save them into the database;
  - check if the Log In is valid; if so, it has to authenticate the user.
- The **User Manager** has to manage the user's profile and the geolocalization services.
- The **Car Manager** has to manage the car status and position.
- The **Reservation Manager** has to allow a user to reserve a free car and it has to keep track of all the related information, for example duration and number of passengers.
- The **Transaction Manager** has to charge the user for its reservation and apply eventual discount or penalties.

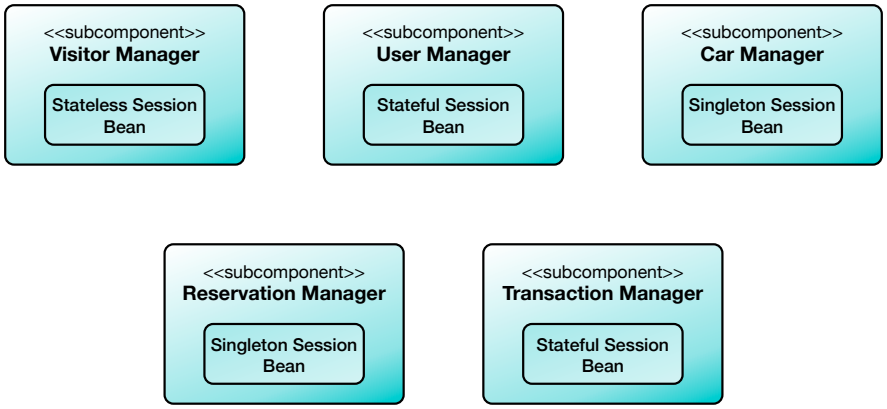

Figure 7: Business Logic component and its subcomponents

Database component

The last component we are going to analyze is shown in Figure 8: the Database component. In order to show the conceptual architecture of this component, we used an Entity-Relationship model, useful to highlight the entities of the system and to specify the relationships that can exist between instances of those entity types.

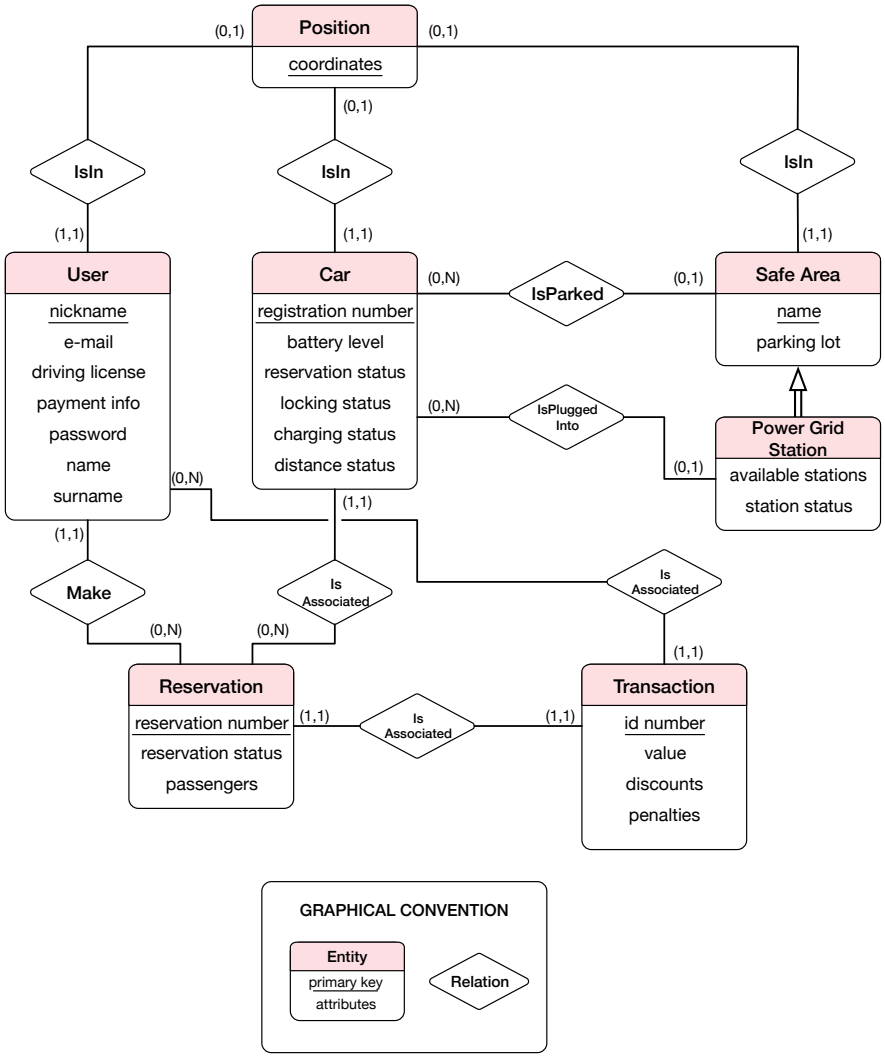

Figure 8: ER model of the Database component

2.5 Deployment view

The diagram in Figure 9 shows the deployment view of the software product. In this early stage, we choose to keep our diagram simple and to only highlight a first distinction between the client machine, the server machine and the database machine. In the next steps of the development, we will detail in a more in specific way the hardware architecture, identifying the hardware components on which the software layer will be deployed.

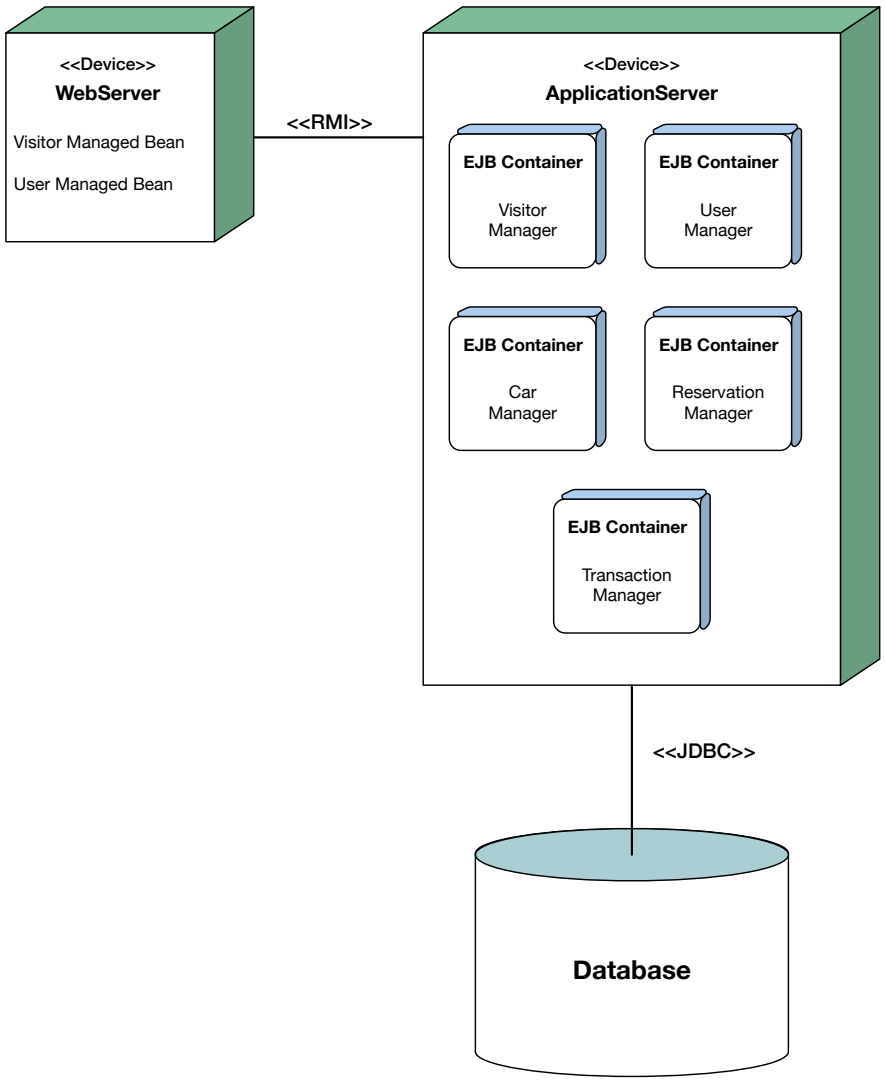

Figure 9: Diagram showing the Deployment view

2.6 Runtime view

In the following subsection, we will show the reader some diagrams that represent the runtime view of PowerEnjoy system: they describe in a simple way the interaction and the behaviour of the component previously described in order to make possible the main functionalities of our system.

Log In and Registration runtime view

The diagram represented in Figure 10 shows the components involved in the login and registration activities and their interactions.

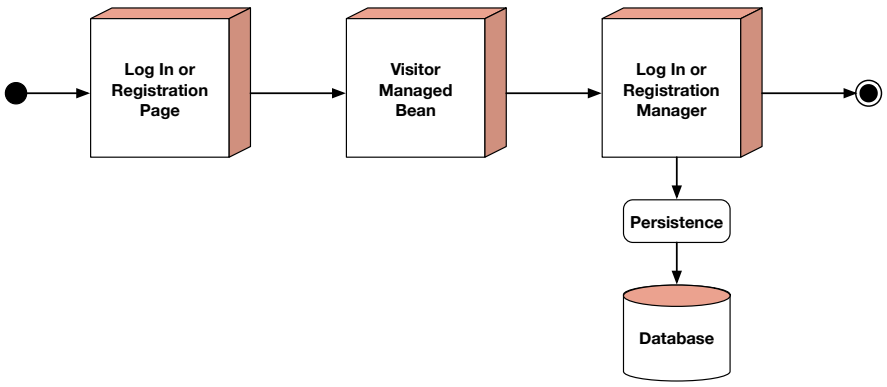

Figure 10: Diagram showing the Log In and Registration runtime view

Modification of User's account runtime view

The diagram represented in Figure 11 shows the components involved in the modification of user's account and their interactions.

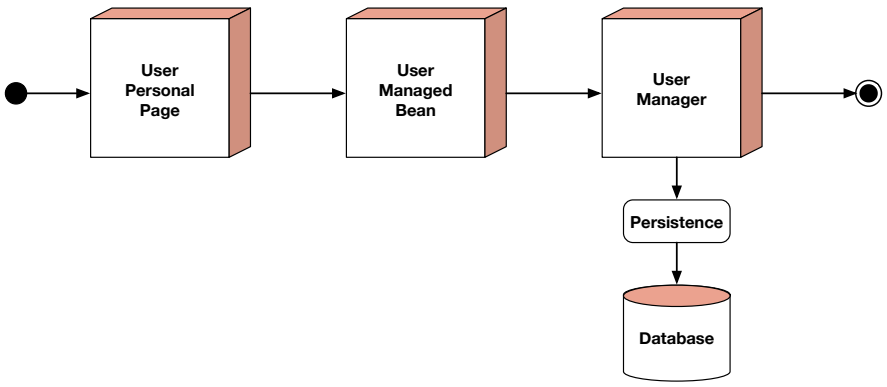

Figure 11: Diagram showing the Modification of User's account runtime view

Car Reservation runtime view

The diagram represented in Figure 12 shows the components involved in the car reservation and their interactions.

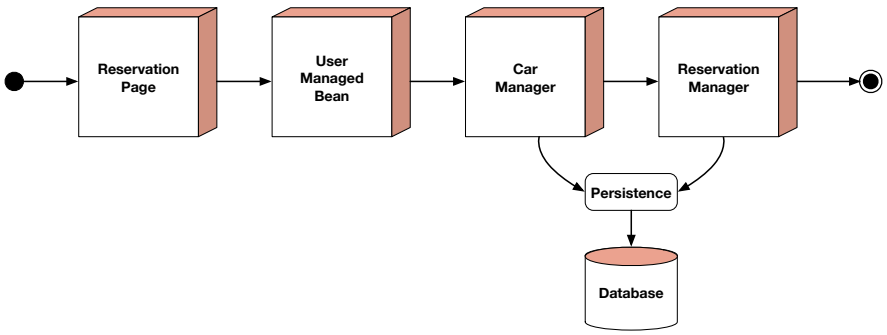

Figure 12: Diagram showing the Car Reservation runtime view

Safe Area and Power Grid Station display runtime view

The diagram represented in Figure 13 shows the components involved in the display of safe areas and power grid stations and their interactions.

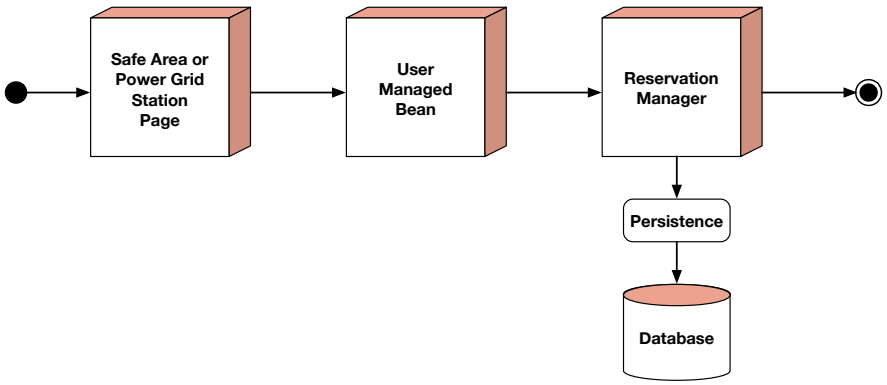

Figure 13: Diagram showing the Safe Area and Power Grid Station display runtime view

Details during the Ride display runtime view

The diagram represented in Figure 14 shows the components involved in the display of the details during the ride and their interactions.

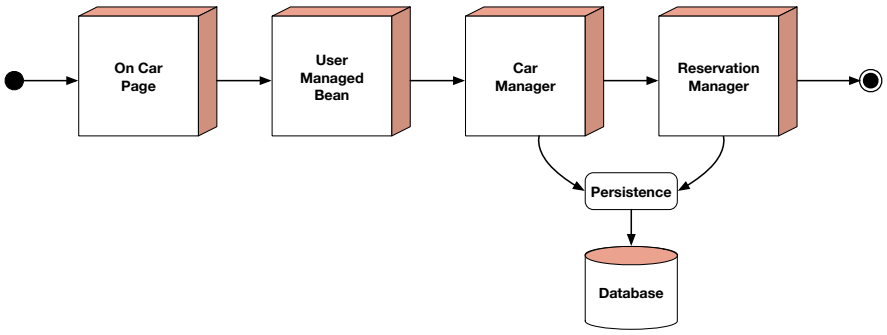

Figure 14: Diagram showing the Details during the Ride display runtime view

Fee computation and display runtime view

The diagram represented in Figure 15 shows the components involved in the computation and display of the fee and their interactions.

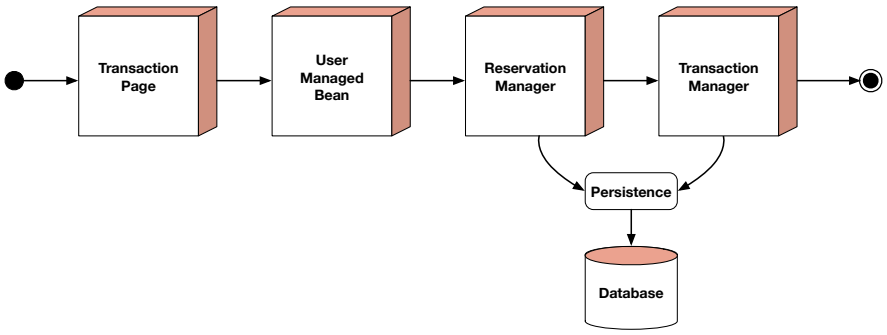

Figure 15: Diagram showing the Fee computation and display runtime view

2.7 Component interfaces

In this subsection, we are going to identify some of the function offered by the Beans that are present in the Business-tier.

Visitor Manager

The Figure 16 shows the interface of the Visitor Manager, that we think it should expose some methods like:

- `createNewUser`: this method adds a new user to the database;
- `checkLogin`: this method checks if the nickname and password, provided during the Log In, match and are correct;
- `checkRegistration`: this method checks if the personal information, provided during the Registration, are valid.

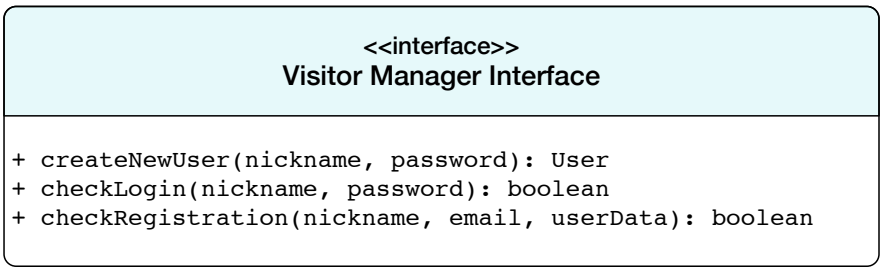

Figure 16: Visitor Manager Interface

User Manager

The Figure 17 shows the interface of the User Manager, that we think it should expose some methods like:

- `getUserInformation`: this method retrieves all the user information stored in the database;
- `setUserInformation`: this method update the user information stored the database;
- `getUserPosition`: this method returns the user position from its smarphone GPS.

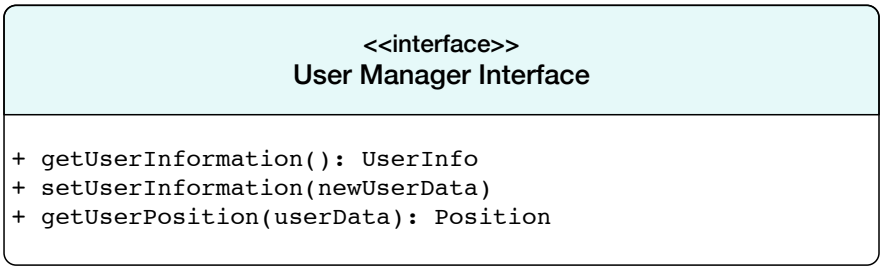

Figure 17: User Manager Interface

Car Manager

The Figure 18 shows the interface of the Car Manager, that we think it should expose some methods like:

- `getCarInformation`: this method retrieves all the car information stored in the database;
- `setCarInformation`: this method update the car information stored the database;
- `getCarPosition`: this method returns the car position from its GPS receiver.

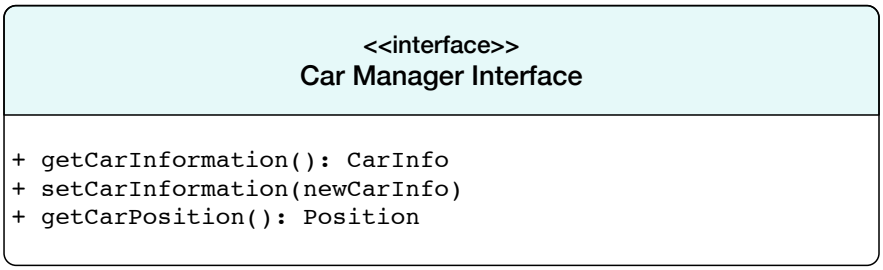

Figure 18: Car Manager Interface

Reservation Manager

The Figure 19 shows the interface of the Reservation Manager, that we think it should expose some methods like:

- `createNewReservation`: this method creates a new reservation;
- `deleteExistingReservation`: this method deletes an existing reservation;
- `getReservationInformation`: this method retrieves all the reservation information stored in the database;
- `setReservationInformation`: this method update the reservation information stored the database;
- `getReservableCar`: this method retrieves the bookable car near the position given by the user.
- `getSafeArea`: this method retrieves the safe area near the position given by the user.
- `getPowerGridStation`: this method retrieves the power grid station near the position given by the user.

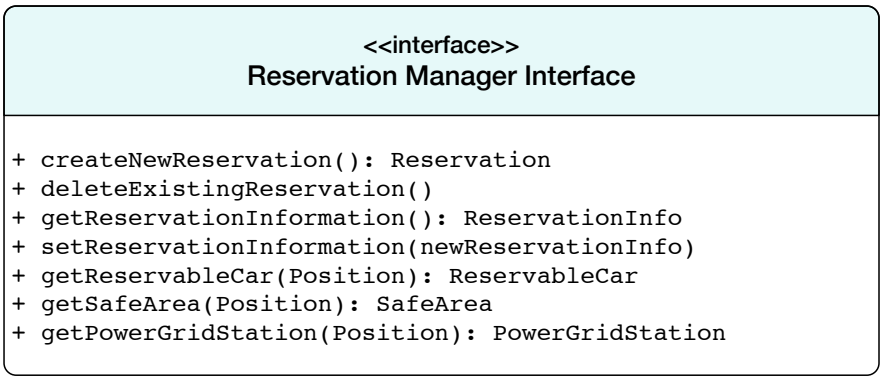

Figure 19: Reservation Manager Interface

Transaction Manager

The Figure 20 shows the interface of the Transaction Manager, that we think it should expose some methods like:

- `getTransactionInformation`: this method retrieves all the transaction information stored in the database;
- `setTransactionInformation`: this method update the transaction information stored the database;
- `applyDiscountOrPenalties`: this method, after having computed the eventual discount or penalties, applies them.

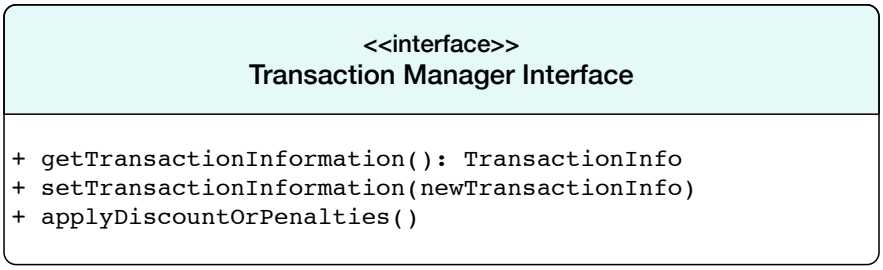

Figure 20: Transaction Manager Interface

2.8 Selected architectural styles and patterns

In this document, we give the reader a first overview on the architecture of our system and a general and simplified description of its behaviour. As far as the architecture is concerned, we choose as a model the one of Java Platform, Enterprise Edition 7. We think that a 4-tier architecture like this, that also relies on the MVC pattern, is the one that best fits our system, since this way we can obtain the performance, scalability, reliability, availability and security requested for PowerEnjoy, a car sharing application. Moreover, we decide to keep the initial model simple in order to leave the space for further improvements and a more detailed one later in the development.

3 Algorithm Design

In the following section, we are going to show the reader some algorithms that we think they will be useful for the implementation of PowerEnjoy. In this document, only a first overview on the step of each algorithm will be presented, without further detailing the implementation aspects. Moreover, we are aware that optimization of the algorithm we are going to present are possible. However, we will discuss about these aspects in the next steps of the development of our system.

3.1 Look for near Safe Areas

In this subsection, we present an algorithm that describes how the system manages the research of safe areas near a certain position given by the user. We make the hypothesis that the safe areas are saved in a linked list. We check each element: if it is near the position of the user, we add it to the near-safe-area list and we keep searching, otherwise we go on without doing anything. We also suppose to have a function `near(P)` that returns `true` if the position of a safe area is near the one given by the user, `false` otherwise.

Algorithm 1 Look for near Safe Areas

```
1: P: position given by the user
2: L: linked list in which the safe areas are saved
3: function Look for near Safe Areas(P, L)
4:     NearL ← linked list in which the near safe areas are saved
5:     for i = 0 → L.size() do
6:         if L.get(i).near(P) then
7:             NearL.add(L.get(i))
8:     return NearL
```

3.2 Look for near Power Grid Stations

In this subsection, we present an algorithm that describes how the system manages the research of power grid stations near a certain position given by the user. We make the hypothesis that the power grid stations are saved in a linked list. We check each element: if it is near the position of the user, we add it to the near-power-grid-stations list and we keep searching, otherwise we go on without doing anything. We also suppose to have a function `near(P)` that returns `true` if the position of a power grid stations is near the one given by the user, `false` otherwise.

Algorithm 2 LOOK FOR NEAR POWER GRID STATIONS

```
1: P: position given by the user
2: S: linked list in which the power grid stations are saved
3: function LOOK FOR NEAR POWER GRID STATIONS(P, S)
4:     NearS ← linked list in which the near power grid stations are saved
5:     for i = 0 → S.size() do
6:         if S.get(i).near(P) then
7:             NearS.add(S.get(i))
8:     return NearS
```

3.3 Look for near Car

In this subsection, we present an algorithm that describes how the system manages the research of bookable car near a certain position given by the user. We first search the safe areas near the given position: for each safe area, if there are one or more bookable car, we add them to the near-car-list. Otherwise, if there are no PowerEnjoy cars or they have been already reserved, we go on to the next near safe area.

Algorithm 3 LOOK FOR NEAR CAR

```
1: P: position given by the user
2: L: linked list in which the safe areas are saved
3: function LOOK FOR NEAR CAR(P, L)
4:     NearC ← linked list in which the near safe areas are saved
5:     for i = 0 → L.size() do
6:         if L.get(i).near(P) then
7:             NearA ← L.get(i)
8:             for j = 0 → NearA.ParkedCar.size() do
9:                 if NearA.ParkedCar.get(j).isReserved() = False then
10:                    NearC.add(NearA.ParkedCar.get(j))
11:    return NearC
```

3.4 Discounts and Penalties Management

In this subsection, we present an algorithm that describes how the system manages the discounts and the penalties that may be applied to a reservation fee. This function takes as input the reservation we want to be analyzed.

- If the system has detected at least two passengers onto the car, there is a discount of 10%.
- If the related car is left with at least 50% of the battery charge, there is a discount of 20%.
- If the related car is plugged into a power grid, there is a discount of 30%.
- If the related car is left at more than 3 Km from the nearest power grid station, there is a penalty of 30%.
- If the related car is left with more than 80% of the battery empty, there is a penalty of 30%.

We suppose that the discount and penalties can be combined, following the order presented above. Moreover, we make the hypothesis to have the function `isFar()`, that returns `true` if the distance between the car and the nearest power station is more than 3 Km, `false` otherwise.

Algorithm 4 Discounts and Penalties Management

```
1: R: we need to compute the final fee of reservation R
2: procedure Discounts and Penalties Management(R)
3:     if R.passengers > 2 then
4:         R.fee ← R.fee − 0.1R.fee
5:     if R.reservedCar.battery > 50 then
6:         R.fee ← R.fee − 0.2R.fee
7:     if R.reservedCar.isPlugged() then
8:         R.fee ← R.fee − 0.3R.fee
9:     if R.reservedCar.isFar() then
10:        R.fee ← R.fee + 0.3R.fee
11:    if R.reservedCar.battery < 0.2 then
12:        R.fee ← R.fee + 0.3R.fee
```

4 User Interface Design

In this section is represented the UX Diagram for the User application with the purpose of showing a schema quite detailed about it. Because in our project from each page the user can go to all the other pages, to make this diagram more readable we have insert a "General Page" that incorporates all the functionalities to go to any page. For the mockups refer to the RASD.

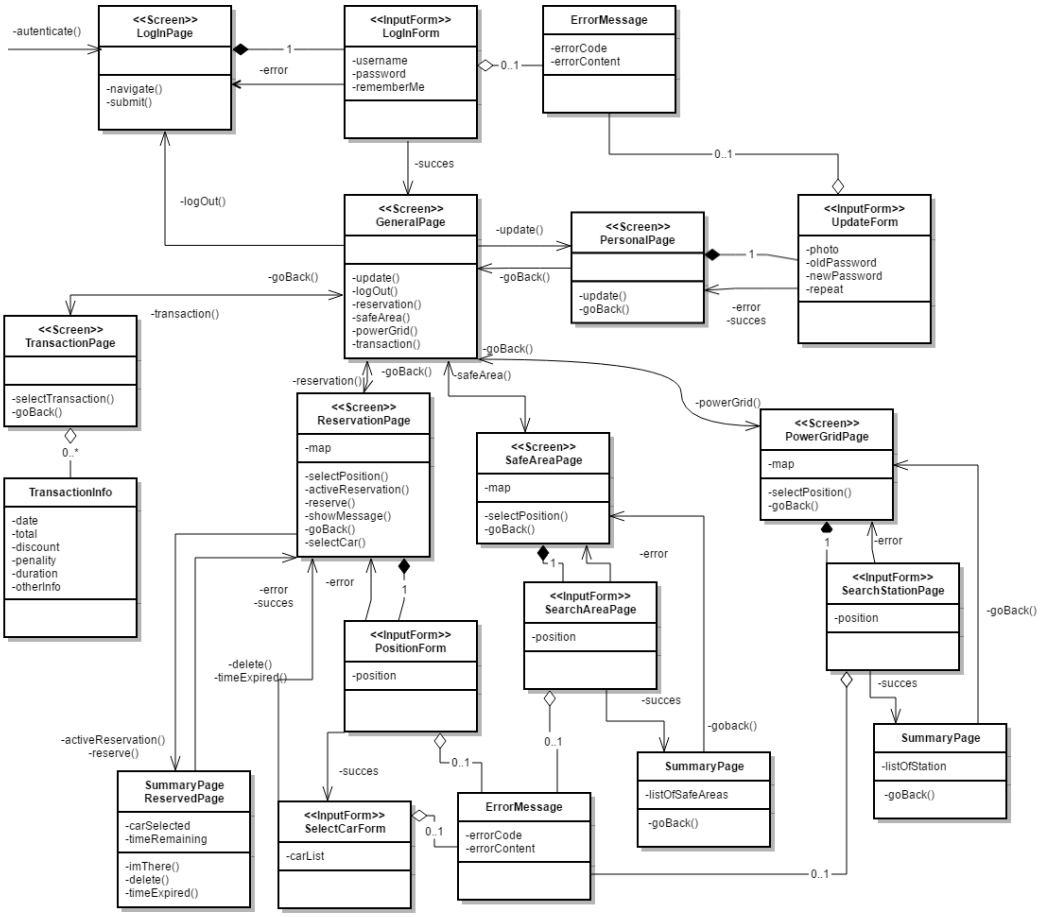

Figure 21: Diagram of the user interface

5 Requirements Traceability

G1.R1 During the registration, visitors can't choose an username already in use.

G1.R2 During the registration, visitors can't choose an e-mail already in use.

G1.R3 During the registration, visitors must insert valid credit card's information.

G1.R4 During the registration, visitors must insert a valid driving license.

G1.R5 During the registration, visitors must agree with the privacy conditions to use the system.

- All these requirements are satisfied by **Visitor Manager** which provides the functionality of check the validity and correctness of user's information.

G2.R1 The registration must be successfully completed.

G2.R2 The user must insert the correct old password.

G2.R3 The user must repeat twice the new password.

G2.R4 The user must upload a valid picture.

- All these requirements are satisfieds by **User Manager** which provides the functionality of manage the user's profile. In the UX Diagram we can see it in the "Personal Page" where there is a form to allow the user to make some profile changes. If there is something wrong it shows an error message.

G3.R1 The user must be registered at the system.

G3.R2 In the log in form the user must insert the correct username.

G3.R3 In the log in form the user must insert the correct password.

G3.R4 If the visitor insert the wrong data, the system shows an error and return to the log in page.

G3.R5 If the user doesn't remember the password he can receive it through a special command.

- All these requirements are satisfied by **Visitor Manager** which provides the functionality of check if the Log In is valid, if so it has to authenticate the user. In the UX Diagram there is the "Log in Page" with a form that check the validity of username and password: if they are correct the application shows the "General Page", otherwise it shows an error message.

G4.R1 The user must insert a valid address or give to the system his position.

G4.R2 All cars position must be known.

G4.R3 All cars are set as available or not.

G4.R4 The list of nearest cars is made through calculate the distance between car and user's position.

- All these requirements are satisfied by **Car Manager** which provides the functionality of manage the cars position and it is obtained allowing the Car Manager to query the Database. But first of all they are satisfied thanks to the use of GPS. We can also see the algorithm to look for a near car in the specific page. We can see that in the "Reservation Page" in the UX Diagram, first of all the user must insert a correct position: if it is valid the page shows the available cars, otherwise it makes an error message. Only if the user insert a correct position he can select one car from a list in the map.

G5.R1 The user must be correctly logged in.

G5.R2 The user must select which car he wants.

- All these requirements are satisfied by **Reservation Manager** which provides the functionality of allow users to reserve a free car and, querying the Database, it can keep track of all the aspects such as the duration. We also use the Google Maps API to show the map.

G6.R1 The position of the user and the car must be the same.

G6.R2 The reservation must not be expired.

G6.R3 The system must recognize the correct car reserved by a specific user to unlock the correct one.

- All these requirements are satisfied by **Reservation Manager** which provides the functionality of query the Database to know all about a reservation. There is also the help of **User Manager** to know the user's position that is inserted in the DB. When there is an active reservation in the UX Diagram from the "Reservation Page" is showed the summary in which there is the functionality "I'm there".

G7.R1 The car's engine must ignites.

G7.R2 An amount of money per minute is set.

G7.R3 The final charge is based on the duration of the car use.

G7.R4 The system must shows all user's transactions.

G7.R5 The user must select a specific transaction to see its details.

- All these requirements are satisfied by **Transaction Manager** which provides the functionality of charge the user and communicates with the Database to show all the details for all the transaction of a user, but also to apply the correct price to the reservation if there are discounts or penalties. There is also the help of an external service to verify the user's credit card information. We can see the algorithm for this management in the specific page. In the UX Diagram we can see the "Transaction Page" that contains the info, such as date, duration, amount, of all the user's transactions.

G8.R1 There must be a display on the car that communicates with the system.

G8.R2 The charge is update every minute.

- All these requirements are satisfied by **Car Manager** and **Reservation Manger** which, querying the Database, can show to the user all the details in the car's display.

G9.R1 It must be passed one hour from the registration.

G9.R2 Is applied a fee of 1 euro to the user involved.

G9.R3 The reservation is deleted.

G9.R4 In the page there is a timer.

- All these requirements are satisfied by **Reservation Manager** which knows the reservation time and ask to **Transaction Manager** to apply the fee to the user. In the UX Diagram if there is already an active reservation the application shows all the summary of the reservation with the timer.

G10.R1 The user must be correctly logged in.

G10.R2 The user must go to the reservation page.

G10.R3 Mustn't be passed one hour from the reservation.

- All these requirements are satisfied by **Reservation Manager**. When there is an active reservation in the UX Diagram from the "Reservation Page" is showed the summary in which there is the functionality "Delete".

G11.R1 The reservation is correctly deleted by a user or it is expired.

G11.R2 The car is set available in the list.

- All these requirements are satisfied by **Reservation Manager** which knows the reservation time or that it is deleted and communicates to **Car Manager** to set the car as available.

G12.R1 A list of safe areas must be available to the user.

G12.R2 The car must be left in a safe area.

G12.R3 The user and all any passengers must exit from the car.

- All these requirements are satisfied by **Car Manager** which knows all the car's details.

G13.R1 The car must be used by a user.

G13.R2 Discounts are in a list where is defined all details.

G13.R3 In the car there is a sensor which counts how many people there are in the car, if they are three or more a discount is applied.

G13.R4 In the car there is a sensor which controls how many battery is empty, if it is no more than 50 per cent the system applied a discount.

G13.R5 If the user takes care to link the car to recharge the car he will have a discount.

- All these requirements are satisfied by **Transaction Manager** that can query the Database to see the discounts.

G14.R1 The car must have been recently used.

G14.R2 Most of 80 per cent of the battery is empty.

G14.R3 The car is left far away from a safe area.

- All these requirements are satisfied by **Transaction Manager** that can query the Database to see the penalties.

G15.R1 The user must go to the specific page.

G15.R2 The user must insert a correct position.

- All these requirements are satisfied by **Reservation Manager** that can query the Database to see the positions. We can see that in the specific algorithm. We can see that in the UX Diagram where, if the user is logged in, he can go to the "Safe Area Page"

and insert a position in the form: if the position isn't correct it shows an error message, otherwise it shows a summary with all the safe areas on the map.

G16.R1 The user must go to the specific page.

G16.R2 The user must insert a correct position.

- All these requirements are satisfied by **Reservation Manager** that can query the Database to see the positions. We can see that in the specific algorithm. We can see that in the UX Diagram where, if the user is logged in, he can go to the "Power Grid Page" and insert a position in the form: if the position isn't correct it shows an error message, otherwise it shows a summary with all the power grid stations on the map.

6 Appendix

In this section, we will give the information about the used tools, the hours of work done by the members of the group.

6.1 Used Tools

In this first phase of the project, the following tools have been used:

- LaTeX and TeXMaker editor: to redact and to format this document
- Omnigraffle (`https://www.omnigroup.com/omnigraffle`): to make some graphs
- Gliffy (`https://www.gliffy.com`): to make the UX Diagram

6.2 Working Hours

Last Name	First Name	Total Hours
Blanco	Federica	25 h
Casasopra	Fabiola	25 h

6.3 Modifications

- Add the subsection 2.2