11 December 2016

PowerEnJoy
Design Document

Blanco	Federica	875487
Casasopra	Fabiola	864412

Software Engineering 2 Project
2016/2017

Contents

List of Figures

List of Tables

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 defitinition:

– 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 technologyâĂŞbased 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 acronims 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

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 stucture of the documtent, 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 reder to get an idea on how the final application will look like.

Section 5 There is an explaination 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 simpicity, 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 decription 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 contein 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.

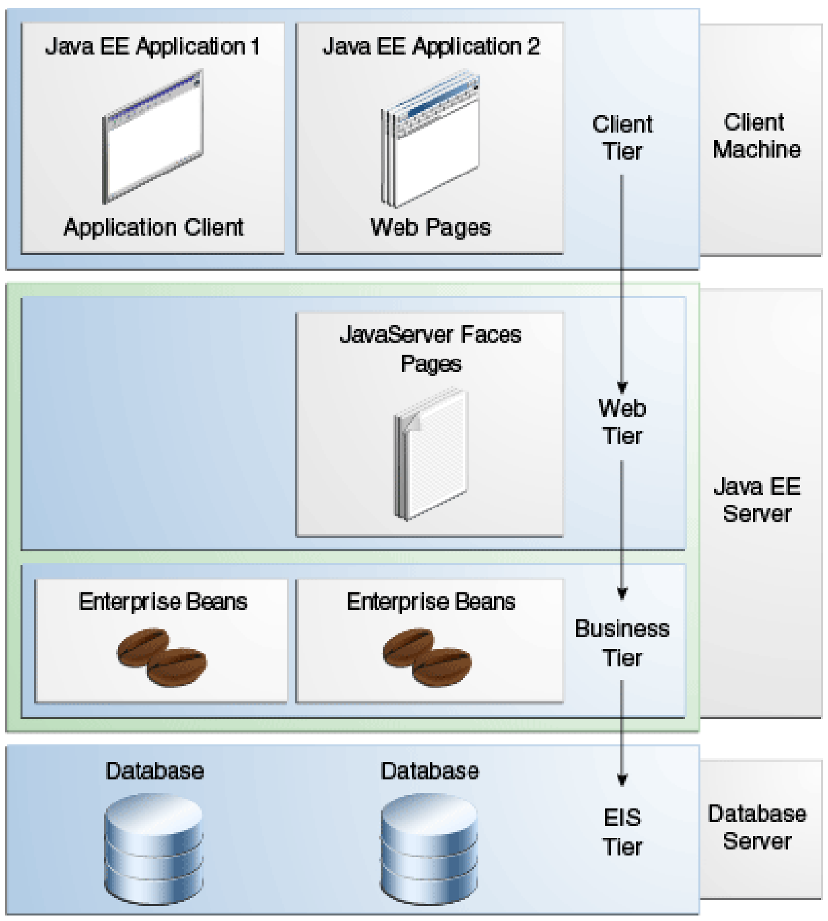


Figure 1: JEE 4-tier architecture

2.2 High Level Components and their Interaction

The diagram in Figure 2 shows the conceptual high level architecture we propose for our system.

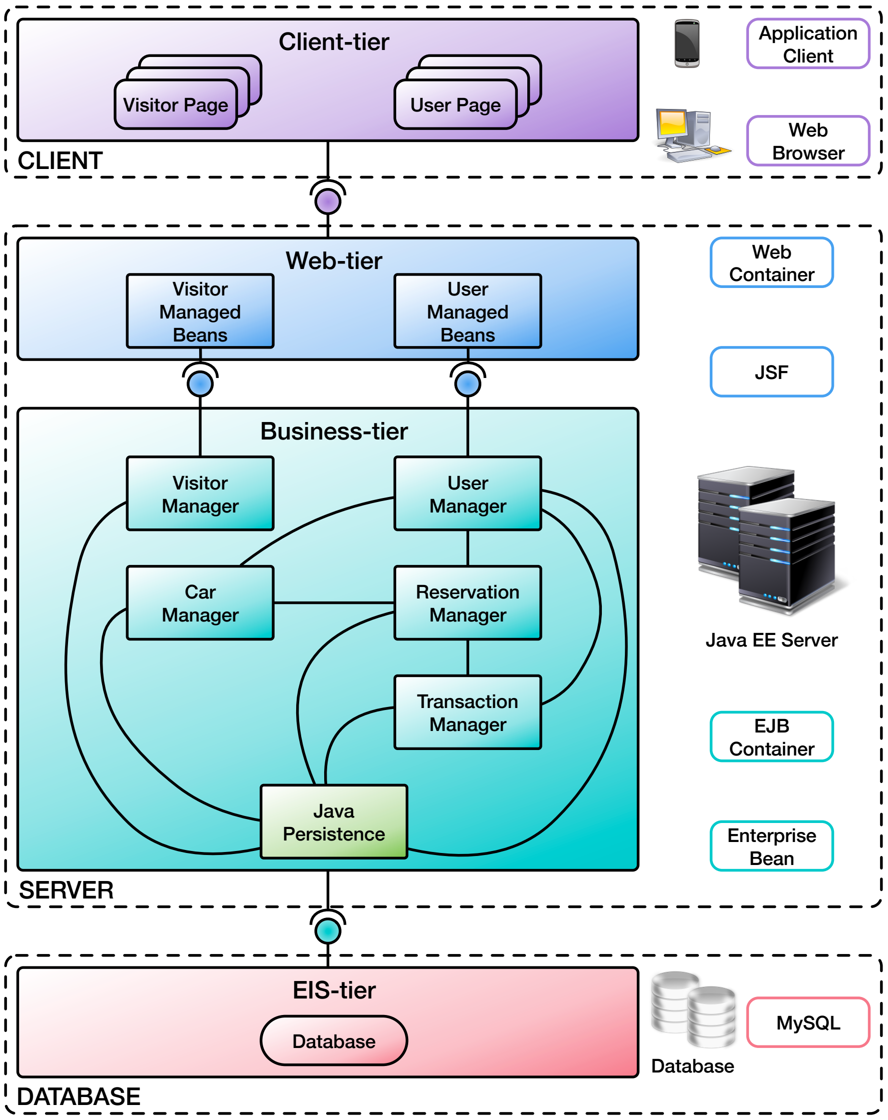


Figure 2: High Level Components and their Interaction of PowerEnjoy

2.3 Component view

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

Client component

The first component we are going to analize is shown in Figure 3: 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.

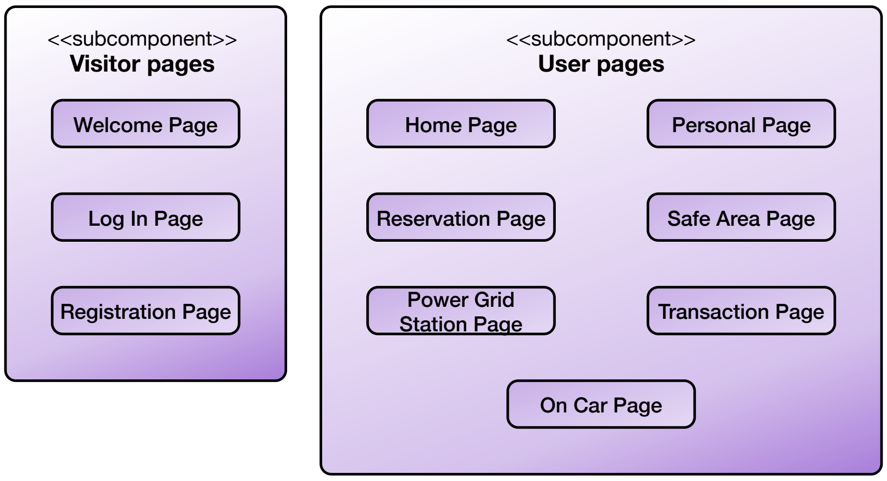


Figure 3: Client component and its subcomponents

Web component

The second component we are going to analize is shown in Figure 4: the Web component generates dynamic web pages containing XHTML. Morover, 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 Busines-tier for processing.

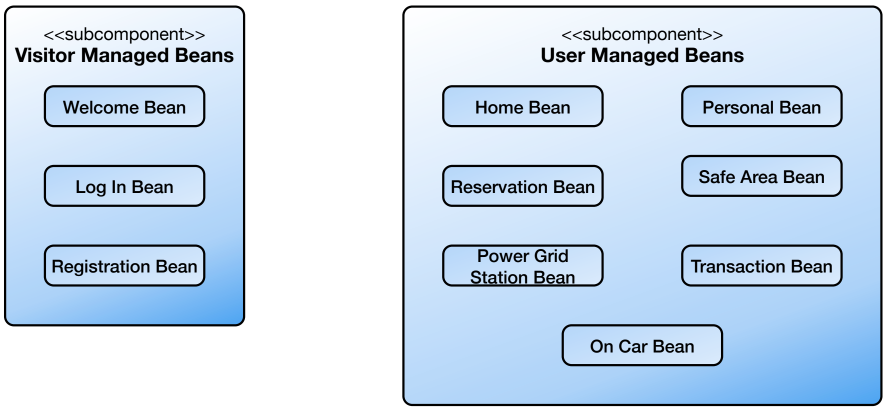


Figure 4: Web component and its subcomponents

Business Logic component

The third component we are going to analize is shown in Figure 5: 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 funtionality 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 users 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.

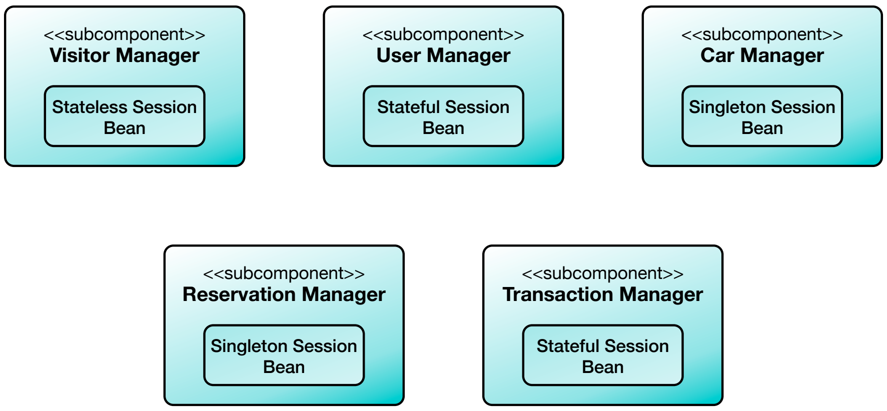


Figure 5: Business Logic component and its subcomponents

Database component

The last component we are going to analize is shown in Figure 6: 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.

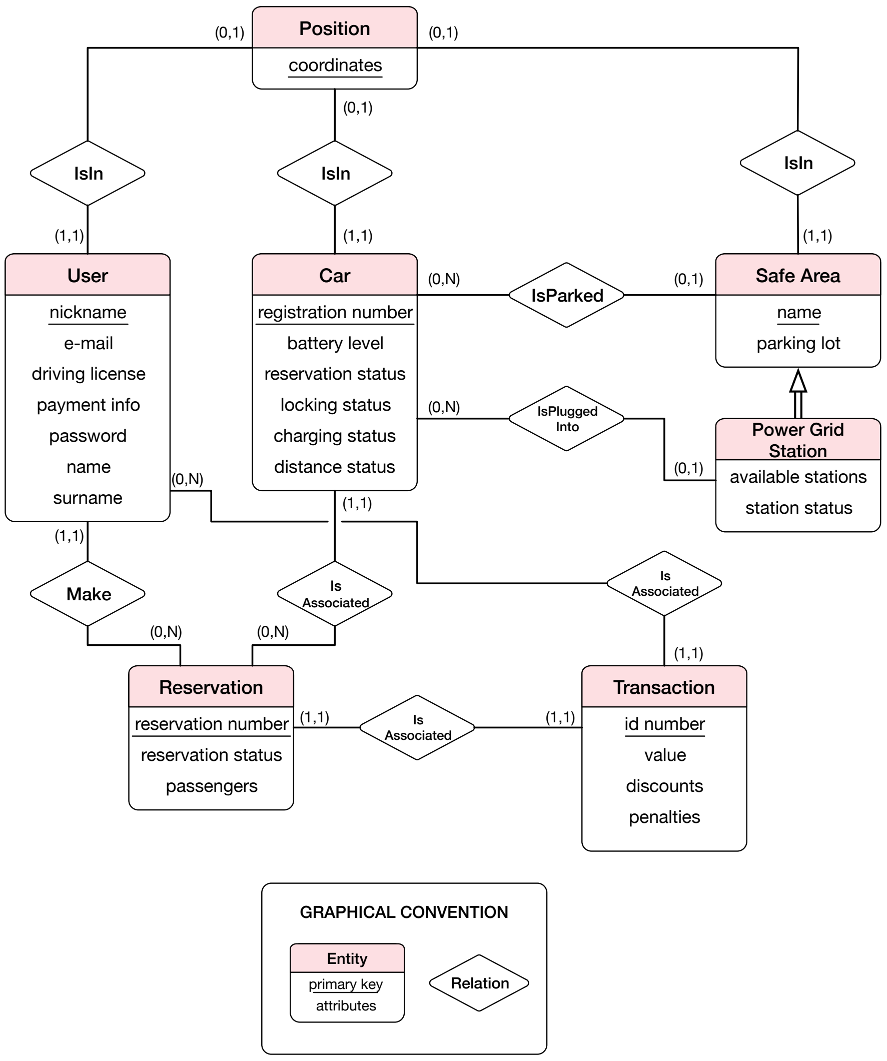


Figure 6: ER model of the Database component

2.4 Deployment view

The diagram in Figure 7 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 details in a more in specific way the hardware architecture, indentifying the hardware components on which the software layer will be deployed.

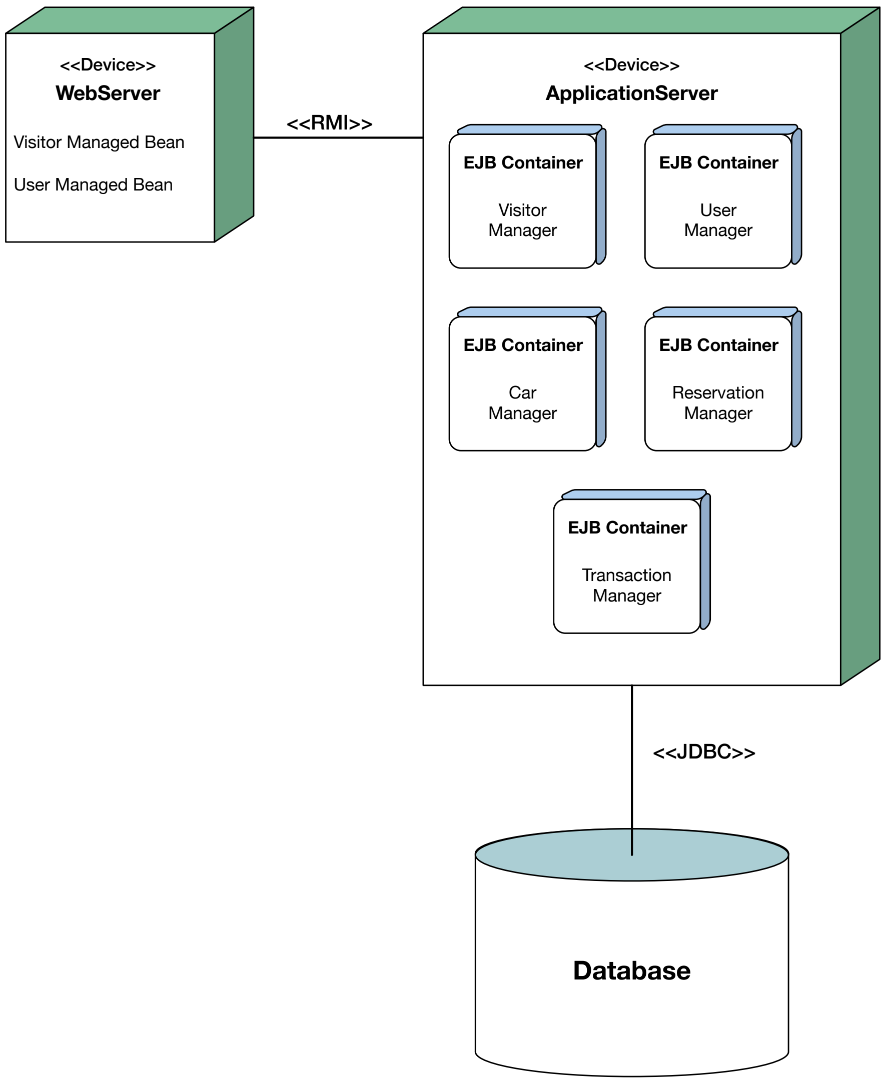


Figure 7: Diagram showing the Deployment view

2.5 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 descibed in order to make possible the main functionalities of our system.

Log In and Registration runtime view

The diagram represented in Figure 8 shows the components involved in the login and registration activities and their interactions.

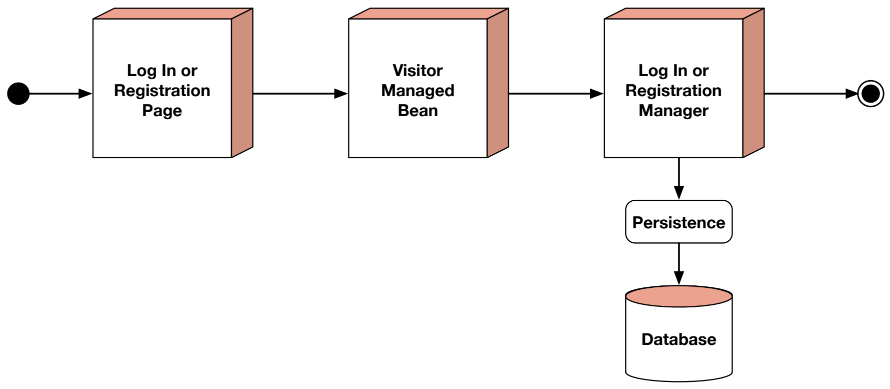


Figure 8: Diagram showing the Log In and Registration runtime view

Modification of User's account runtime view

The diagram represented in Figure 9 shows the components involved in the modification of user's account and their interactions.

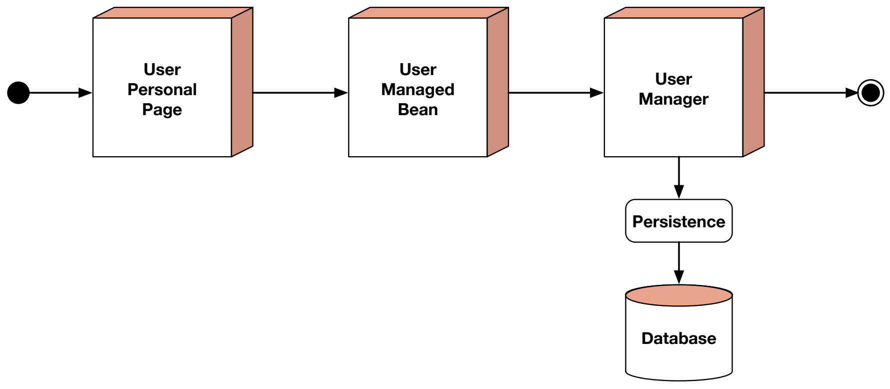


Figure 9: Diagram showing the Modification of User's account runtime view

Car Reservation runtime view

The diagram represented in Figure 10 shows the components involved in the car reservation and their interactions.

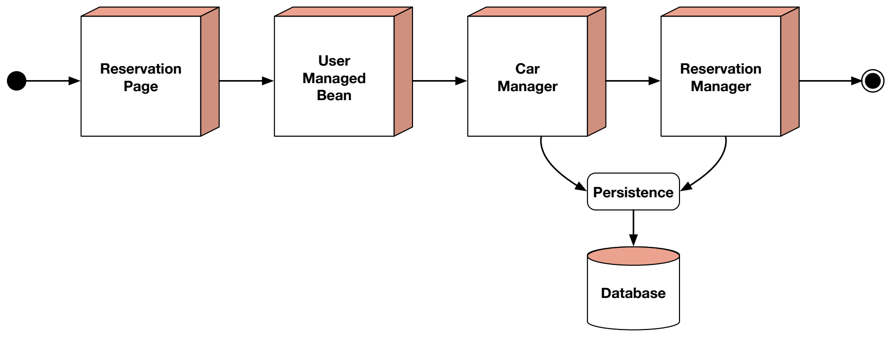


Figure 10: Diagram showing the Car Reservation runtime view

Safe Area and Power Grid Station display runtime view

The diagram represented in Figure 11 shows the components involved in the display of safe areas and power grid stations and their interactions.

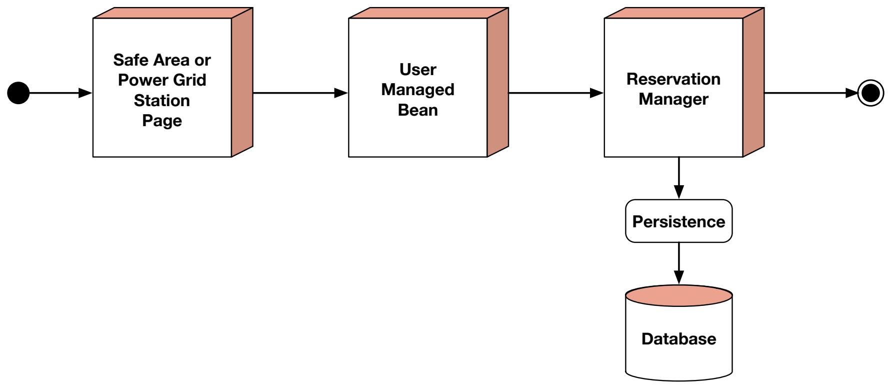


Figure 11: Diagram showing the Safe Area and Power Grid Station display runtime view

Details during the Ride display runtime view

The diagram represented in Figure 12 shows the components involved in the display of the details during the ride and their interactions.

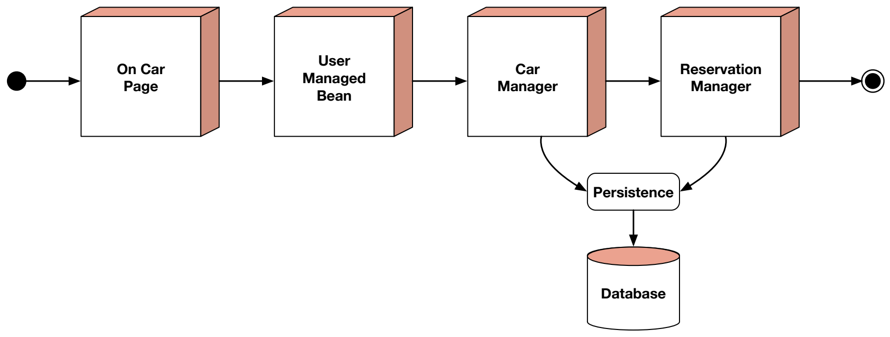


Figure 12: Diagram showing the Details during the Ride display runtime view

Fee computation and display runtime view

The diagram represented in Figure 13 shows the components involved in the computation and display of the fee and their interactions.

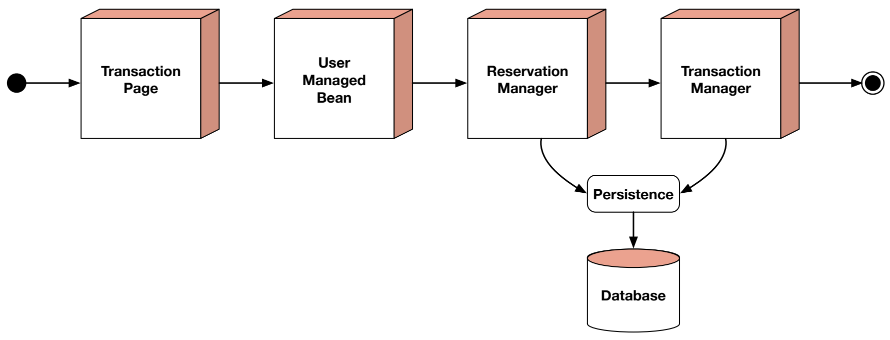


Figure 13: Diagram showing the Fee computation and display runtime view

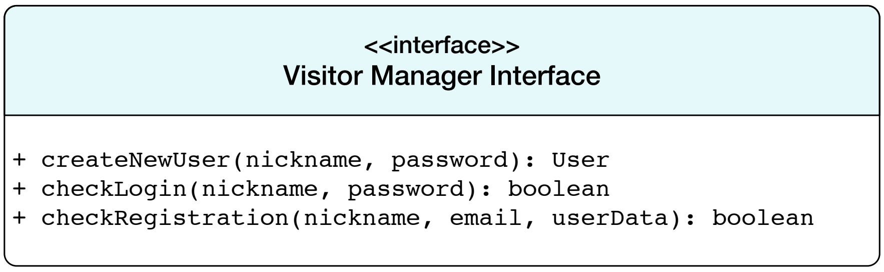
2.6 Component interfaces

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 14 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.



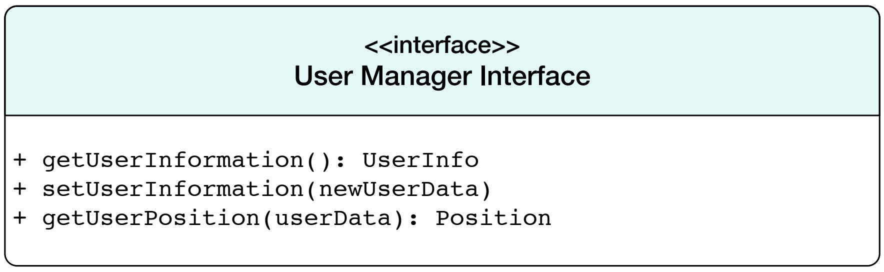
```
+ createNewUser(nickname, password): User
+ checkLogin(nickname, password): boolean
+ checkRegistration(nickname, email, userData): boolean
```

Figure 14: Visitor Manager Interface

User Manager

The Figure 15 shows the interface of the User Manager, that we think it should expose some methods like:

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



```
+ getUserInformation(): UserInfo
+ setUserInformation(newUserData)
+ getUserPosition(userData): Position
```

Figure 15: User Manager Interface

Car Manager

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

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

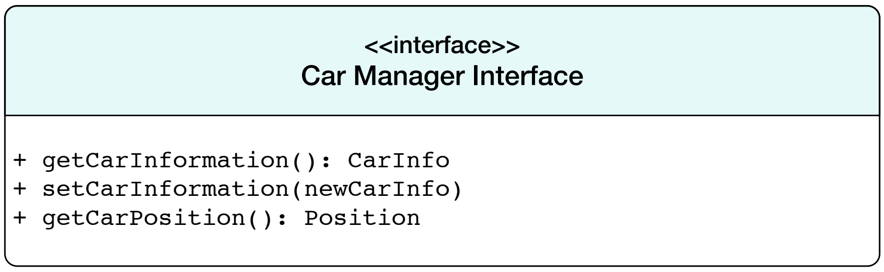
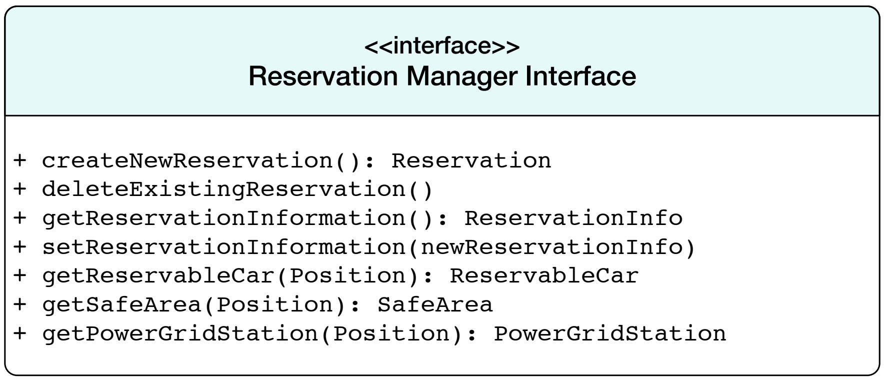

<<interface>>
Car Manager Interface
+ getCarInformation(): CarInfo
+ setCarInformation(newCarInfo)
+ getCarPosition(): Position

Figure 16: Car Manager Interface

Reservation Manager

The Figure 17 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 retrives all the reservation information stored in the database;
- `setReservationInformation`: this methods update the reservation information stored the database;
- `getReservableCar`: this method retrieves the reservable 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.



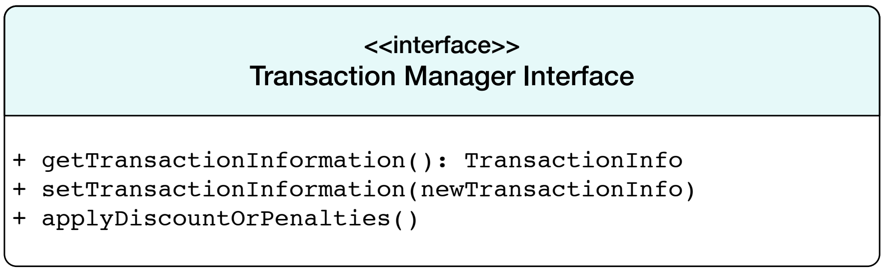
```
+ createNewReservation(): Reservation
+ deleteExistingReservation()
+ getReservationInformation(): ReservationInfo
+ setReservationInformation(newReservationInfo)
+ getReservableCar(Position): ReservableCar
+ getSafeArea(Position): SafeArea
+ getPowerGridStation(Position): PowerGridStation
```

Figure 17: Reservation Manager Interface

Transaction Manager

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

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



```
+ getTransactionInformation(): TransactionInfo
+ setTransactionInformation(newTransactionInfo)
+ applyDiscountOrPenalties()
```

Figure 18: Transaction Manager Interface

2.7 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 architecure 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 detalied one later in the development.

3 Algorithm Design

In the following section, we are going to show the reader some algorithms thet 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 this 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

```
P: position given by the user
L: linked list in which the safe areas are saved
function Look for near Safe Areas(P, L)
    $NearL \leftarrow$ linked list in which the near safe areas are saved
    for $i = 0 \rightarrow L.size()$ do
        if $L.get(i).near(P)$ then
            $NearL.add(L.get(i))$
    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:     $NearS \leftarrow$ linked list in which the near power grid stations are saved
5:     for $i = 0 \rightarrow S.size()$ 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 reservable 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 reservable car, we add them to the near-car-list. Otherwise, if there are no PowerEnjoy cars or they havel 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:     $NearC \leftarrow$ linked list in which the near safe areas are saved
5:     for $i = 0 \rightarrow L.size()$ do
6:         if $L.get(i).near(P)$ then
7:             $NearA \leftarrow L.get(i)$
8:             for $j = 0 \rightarrow NearA.ParkedCar.size()$ 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 Managment

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 have 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 penalties of 30%.
- If the related car is left with more than 80% of the battery empty, there is a penalties 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 Managment

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

4 User Interface Design

5 Requirements Traceability

6 Appendix

6.1 References

6.2 Effort Spent