**Politecnico di Milano**

**Software Engineering 2**

**DESIGN DOCUMENT**

**PowerEnjoy**

**Authors:**

**Simone Bruzzechesse**

**Luca Franceschetti**

**Gian Giacomo Gatti**

# Index

Sommario

[Index 3](#_Toc467681136)

[1 Introduction 4](#_Toc467681137)

[1.1 Purpose 4](#_Toc467681138)

[1.2 Scope 4](#_Toc467681139)

[1.3 Definition, Acronyms, Abbreviation 5](#_Toc467681140)

[1.4 Reference Documents 5](#_Toc467681141)

[1.5 Document Structure 6](#_Toc467681142)

[2 Architectural Design 7](#_Toc467681143)

[2.1 Overview 7](#_Toc467681144)

[2.2 Component view 8](#_Toc467681145)

[2.3 Deployment view 11](#_Toc467681146)

[2.4 Runtime view 11](#_Toc467681147)

[2.5 Component interfaces 11](#_Toc467681148)

[2.6 Selected architectural styles and patterns 11](#_Toc467681149)

[2.7 Other design decision 11](#_Toc467681150)

[3 Algorithm design 13](#_Toc467681151)

[4 User interface design 14](#_Toc467681152)

[5 Requirements traceability 15](#_Toc467681153)

[6 Effort spent 16](#_Toc467681154)

[7 References 17](#_Toc467681155)

# Introduction

## Purpose

The purpose of this document is to give a better explanation of the PowerEnjoy system with respect to the RASD document. The aim of this document is to define aspects of the software that are useful for the programmers to develop the system according to the specification that are proposed in this document. The document purpose is to identify:

* High level infrastructure of the system
* The main components of the system and the interfaces between them
* The main interfaces of the system with existing components or 3rd part software
* The runtime behaviour of the system

## Scope

The aim of PowerEnjoy software is to provide a system for a car-sharing service that exclusively employs electric cars, so it should provide the functionality normally provided by car-sharing services. The users must be able to register to the system by providing their credentials and payment information, then they receive back a password that can be used to access the system. This procedure can be done both by the mobile application or web application.

Registered users must be able to find the locations of available cars within a certain distance from their current location or from a specified address. The system provides also the possibility to reserve a single car, but with some constraint: for example, if a car is not picked up, the user must pay a fee. On the other hands, if a user reaches a reserved car, he must be able to tell the system he’s nearby his reserved car, so the car will be unlocked and the user can enter and start his rent.

Car-sharing system initiate the charging of money as soon as the engine ignites, and the system starts charging the user for a given amount of money per minute. Indeed, the user is notified of the current charges through a screen on the car. The system stops charging the user as soon as the car is parked in a safe area and the user exits the car.

The set of safe areas for parking cars is predefined by the management system, so we can contact a database to catch some information about the current position of the car, and then the system can decide if it is parked in a safe area.

Although, the system must be able to define certain user’s behaviour with the car-sharing services and apply some discount (or charging) in consequence of determinate action.

## Definition, Acronyms, Abbreviation

Here there are some definition and acronyms that we will use later on this document:

* **RASD**: requirements analysis and specifications document
* **DD**: design document
* **SMS**: short message service; it is a notification sent to a mobile phone, an SMS gateway is needed to use it.
* **CharginsStationAreas**:
* **SafeAreas**:
* **API**: application programming interface; it is a common way to communicate with another system.
* **MVC**: model view controller
* **URL**: uniform resource locator
* **Push** **notification**: it is a notification sent to a smartphone using the mobile application, so it must be installed.
* **Push** **service**: it is a service that allows to send push notifications with own API
* Matching itineraries:
* **Path**: it’s a structure containing at least 2 positions
* **Sharing** discount percentage: discount percentage applied only if the sharing option is enabled and there is more than one request in the merged request
* **REST**: Representational State Transfer
* **RESTful**: REST with no session
* **ETA**: estimated time available; it is the time the taxi needs to arrive to client starting position.
* **Zone**: it is a zone of approximately 2 kmˆ2, the city is split into these zones. From taxi position the system gets his zone and inserts the taxi into the zone queue. So the system guarantees a fair management of taxi queues.
* **UX**: user experience design
* **BCE**: business controller entity

## Reference Documents

* Our RASD document
* Specification Document: Assignments AA 2016-2017.pdf
* Structure of the design document.pdf
* Sample Design Deliverable Discussed on Nov. 2

## Document Structure

1. **Introduction**: this section introduces the design document. It contains the purpose and the scope of this document and the parts that are more specified with respect to the RASD document.
2. **Architecture Design**: this section is divided into different parts:
   1. Overview: this part explains the main tier of our application;
   2. High level components and their interaction: this second part gives a high-level view of our architecture and its main components and the interaction between them;
   3. Component view: this sections gives a more detailed view of the components of our applications;
   4. Deploying view: this section explain which components must be deployed to let the application running in the correct way;
   5. Runtime view: “architectural” sequence diagrams are represented in this section to better define the process of the different tasks of our application;
   6. Component interfaces: this section shows the main interfaces between our components and existing software;
   7. Selected architectural styles and patterns: this section explains the architectural choices taken during the creation of the application
   8. Other design decisions
3. **Algorithms Design**: this section describes the main algorithm and the most critical ones that are needed for the application. We use Pseudo code in order to show the main idea of the algorithm and at the same time to hide unnecessary information and keep the algorithm “light” as much as possible.
4. **User Interface** **Design**: this section presents the mockups that have been already seen in the RASD document and present the user experience explained via UX and BCE diagrams.
5. **Requirements Traceability**: this section aims to explain how the decisions taken in the RASD are linked to design elements.

# Architectural Design

## Overview

The overview of our architecture is represented in the diagram below. In particular, we consider the traditional client-server architecture based on three-tier.

As we can see from the diagram this architecture reflects the basic one of the java EE standard, in fact also that one has 3 tier and allow us to abstract from the complexity of the application by organizing it into components and containers. This allow us also to abstract from many issues related to a complex architecture since a lot of services are provided directly from the containers and also the deployment time is reduced (due for example to the use of the annotations). We report also the architecture of the J2EE in order to show the parallelism with our system.



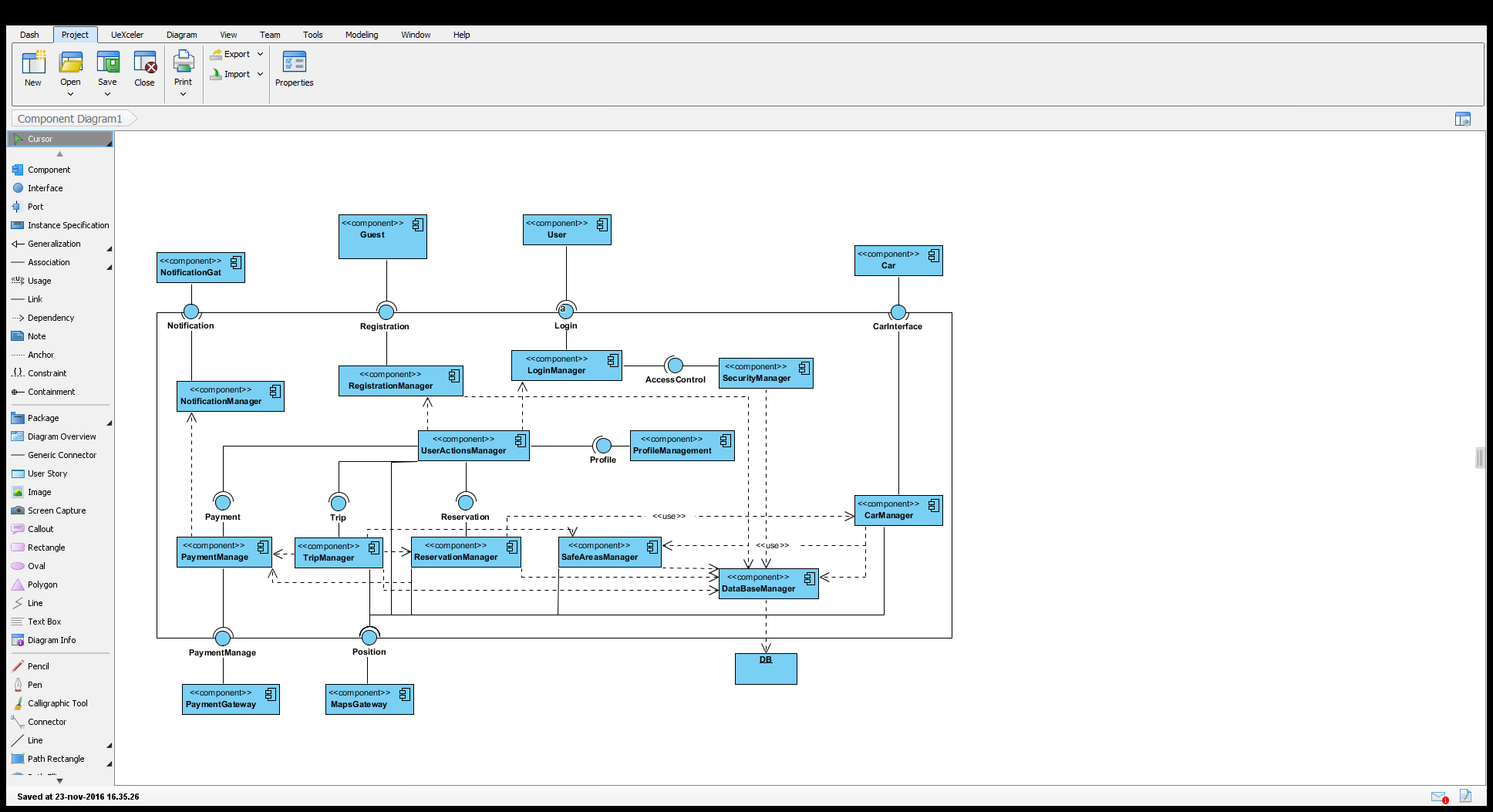
We show the high-level components and their functionalities:

* **Client-tier**: run on the client machine, both on the web browser and mobile application and also on the machine’s tablet;
* **Server-tier**:
  + **Web-tier:** run on the Java EE server; they are organized in Servlets and JSP pages and they will be used to manage the interaction with the web application;
  + **Business-tier**: run on the Java EE server; these components manage the internal logic of the system and they will be analysed later in the Component Diagrams;
* **Enterprise Information System (EIS)-tie**: run on the database server and handle enterprise infrastructure systems, database systems and legacy systems.

## Component view

The main components of our system are listed above, and then thy are represent in a component diagram in order to show the interfaces and the links between them.

* **RegistrationManager**: this component is responsible for the registration of a new user of the system, it uses some service offered by the securityManager handles the registration of a new user into the system, using some interfaces offered by the Security component.
* **LoginManager**: this component is responsible for the login process of a user into the system it will use some interfaces offered by the Security component in order to allow to use the system only to registered users;
* **SecurityManager**: this component is responsible for the security of the system, it can recognize authorized users and let them use properly the system and denied the usage of this one to unregistered users;
* **DatabaseManager**: this component manages the interaction with the database; it is responsible for providing all the information asked by all the others components and also sending new data or modifications to the database;
* **SafeAreasManager**: this component coordinates all the areas of the city, in particular it distinguishes between:
  + SafeAreas
  + ChargingStationAreas
  + Other Areas
* **ReservationManager**: this component is able to manage a reservation, in fact it can perform different operations:
  + Make a new reservation;
  + Delete a reservation;
  + Know if a reservation has expired and ask the PaymentManager to add the fee to the user associated to that reservation;
* **UserAccountManager**: the scope of this component is to manage all the possible actions that the user can perform, for example it allows the user to manage his account, reserve a car or start a trip according to the state of the user. It can perform all this actions using the different operations that other component provides through the interfaces.
* **PaymentManager**: can manage a payment since it can interface with the PaymentGateway who manage the effective payment.
* **PaymentGateway**: is the component that is directly interfaced with another software that manage all the payments of the system.
* **MapsGateway**: the scope of this component is to get the GPS positions of the devices that use it. This component after using it sends the data to the others various components in the system;
* **NotificationManager**: this component can manage all the type of notification that must be sent to the user:
  + TripReview and Cost
  + Fee
  + Confirmation Email
  + Sms message to confirm the smartphone association
* **NotificationGateway**: is linked with the external software that can send sms and email;

 Updates the information of the calls when their status is changed.

## Deployment view

## Runtime view

## Component interfaces

## Selected architectural styles and patterns

## Other design decision

# Algorithm design

# User interface design

# Requirements traceability

# Effort spent

# References