Politecnico di Milano

Software Engineering II project:
PowerEnjoy

Design Document

Gregori Giacomo and Ruaro Nicola

February 5, 2017
Version 1.2

Contents

Abstract

This document provides a more technical description about the PowerEnJoy system adopting the IEEE-1016 standard for DD documentation.
The scope of the Design Document is to discuss our architectural and algorithmic design choices and the user experience that PowerEnJoy should provide. It is based on the Requirement Analysis Specification Document presented in the previous delivery.

Introduction

1.1 Scope of the System

PowerEnJoy is a car-sharing service based on mobile and web applications which should allow users to reserve vehicles and use them.
It will be deployed as a 4-tier JEE client-server application which will be discussed in the following sections.
The application logic must be designed and allocated into components that should improve software maintenability and ease future extensions.

1.2 Document Structure

Introduction: In this chapter an introduction to the system and the Design Document is given.

Architectural Design: In this section an overall description of the architecture is given, it is structured into 7 different parts:

- Overview: High-level components and their interaction
- Component view
- Deployment view
- Runtime view
- Component Interfaces
- Selected architectural styles and patterns
- Other design decisions

Algorithm Design: In this chapter the implemented algorithms are discussed and presented using flow-charts and pseudo-code in order to ease the comprehension and focus on the functionality.

User Interface Design: In this section the main choices in User Interface and User Experience design are discussed.

Requirements Traceability: In this section a clear link between requirements specification (RASD) and design decisions (DD) is created.

Architectural Design

2.1 Overview: High-level components and their interaction

A brief description of the overall design of the system is presented in this section of the DD. Our system will be developed as a 4-tiered JEE application, divided as Client Tier, Web Tier, Business Tier and the EIS Tier. It is distributed between client machines, Java EE server machine and the database.
The mobile and web applications in particular are thin since data operations will be computed by a central server; in this way there is no heavy load on user side clients.
The diagram below provides a better understanding of the components of our system, highlighting the interactions among them:

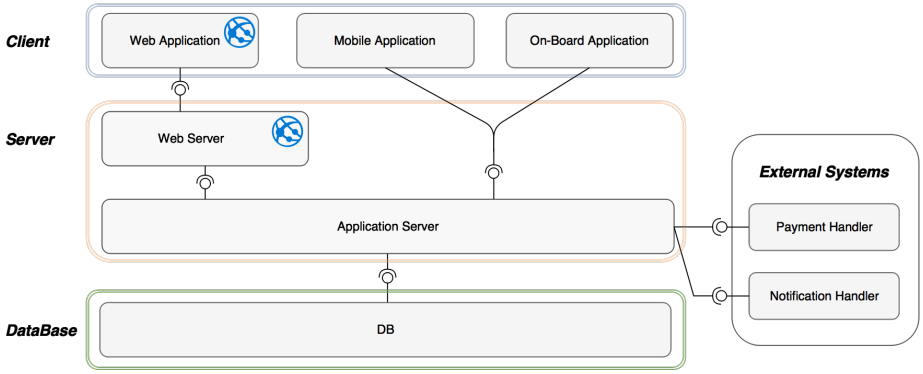

Figure 2.1: System architecture

We can observe that the Web application needs to interact with the Web Server before accessing the Application server, the mobile application on the other side has a direct access to it.

2.2 Component view

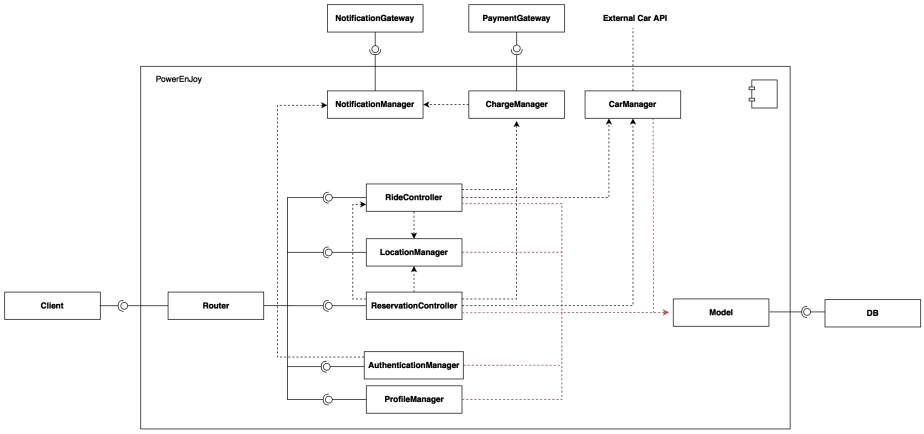

Figure 2.2: Component view for the Application Server

2.2.1 System components

To define and easily understand what kind of functionalities must be implemented in our system we decided to decompose PowerEnJoy logically into components, which are reusable and easily adaptable bricks for our application.
In this section the single components and their interactions are analysed, the router component is not represented for sake of simplicity.

- **AuthenticationManager**: This component provides all the authentication-related functionalities such as registration, login, credentials generation and password recovery. It is important to remark here that a RESTful API is provided and so no session is created, instead a token is provided and used for authentication purposes.
- **ProfileManager**: This component manages all the profile-related functionalities in order to allow informations' editing.
- **LocationManager**: This component handles the logic behind the vehicle/user localization and tracking, it is also responsible for safe-areas and charging-stations' location consistency.
- **ReservationController**: This component manages the reservation logic, it receives informations from the LocationManager, correctly handles the timing for expiration and queries the CarManager component to update the car status (FREE, RESERVED, INUSE, OUTOFSERVICE). It is responsible for the Reservation logic and correctness checking.

- **RideController**: This component controls the (un)locking of the car, the car status and correctly handles the timing and charges for the ride. It is responsible for the Ride logic and correctness checking.
- **CarManager**: This component is responsible for communications with the on-board computer and for car's status update.
- **ChargeManager**: This component handles the application of charges for rides and reservations, it also process the applications of fees and discounts due to bad/virtuous behaviours. It is responsible to communicate with the PaymentGateway to complete the payment process.
- **NotificationManager**: This component manages the users' notification, in particular regarding charges and payment requests. It communicates with the NotificationGateway to effectively notify the users.
- **PaymentGateway**: This component is responsible for the communication with the external payment handler in order to effectively process the payments(automatic payments are pre-authorized).
- **NotificationGateway**: This component actually creates and send the user notification.
- **Router**: This component is responsible for routing the requests to the correct components.
- **Client**: The actual client device(Mobile/Web application).
- **Model**: The data we interact with, this is an abstraction of the DataBase.
- **DataBase**: The database used to store persistent data.

2.2.2 Database components

The data stored in the database will be split into different subcomponents that identifies the main entities of our system: User,Vehicle, Location, Safe Area, Charging Station, Reservation, Ride, Behaviours and Payment.
The designed model for persistent data is provided here in a ER diagram in order to better analyze the motivations of our design. That's the representation of the database model:

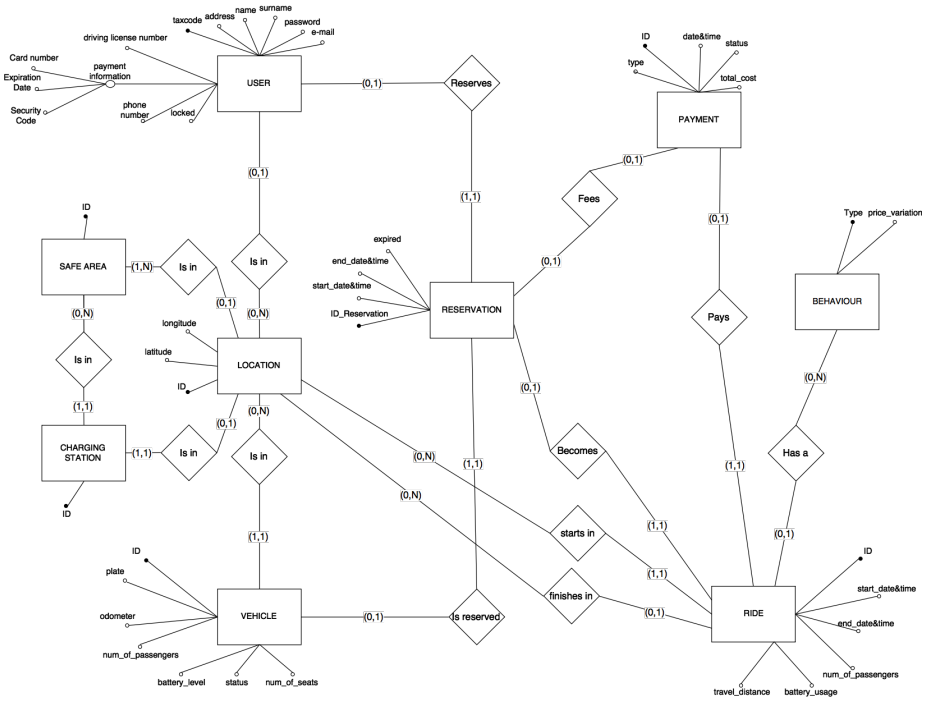

Figure 2.3: ER Diagram

And this is the relation schema associated with the ER diagram.

- User (<u>taxcode</u>, *Location*, e-mail, password, driving_license_number, name, surname, address, phone number, card_number, expiration_date, security_code, locked)
- Vehicle (<u>ID</u>, *Location*, plate, type, odometer, battery_level, status, num_of_seats, num_of_passengers)
- Location (<u>ID</u>), longitude, latitude
- Safa Area (<u>ID</u>, <u>Location</u>)

- Charging Station(ID, *Location*,*ID_SafeArea*)
- Reservation (ID, *ID_User*, *ID_Vehicle*, start_date&time, end_date&time, expired)
- Ride (ID, *ID_User*, *ID_Vehicle*, *ID_Reservation*, *ID_Payment*,*Start_Location*,*End_Location*, start_date&time, end_date&time, num_of_passengers, travel_distance, battery_usage)
- Behaviours(Type, *ID_Ride*, price_variation)
- Payment (ID, *ID_User*, *ID_Reservation*, *ID_Ride*, date&time, total_cost, status, type)

In the User entity there are all the main informations about the user such as credentials and payment method. User lock tag can be either TRUE or FALSE. The user's location is not mandatory.
A Vehicle entity has different attributes, some of them supplied by the On-Board computer. The status here can be FREE, RESERVED, INUSE or OUTOFSERVICE.
The Location entity represent a location provided by the GPS. Safe areas are zones composed by one or more locations disposed in poligons, which can contain Charging stations.
A reservation is associated with a Payment object if and only if it is expired. Reservations are strictly connected with the Ride entity, which can exist only related with it. Each ride has a starting point and when it finishes an end point. Every Ride is associated with a Payment object, in particular the presence of a related Behaviour object implies a variation of the final price.
In the Payment entity there are informations about the transaction from the user to PowerEnJoy, the status indicates if the payments succeeded or is pending.

2.3 Deployment view

The hardware topology is described here, highlighting components and their relationships. The software parts are deployed in order to have the system working.
As previously seen in the Overview the system will accomplish a 4-tier architecture:

- The client device, where an User can interact with the system. There are different GUI that renders the web or mobile pages of our system, differentiating between On-board computer, mobile application and web application.
- The Web Server is needed for those who are connected to the system with a computer. It establishes a secure internet connection through the HTTPS protocol.
- The Application Server is the core of our system. Here we have the Business Logic, where the whole system computation is done.
- In the Database all the informations of the system are stored. It's accessible only by the Application Server that store and retrieve data from there.

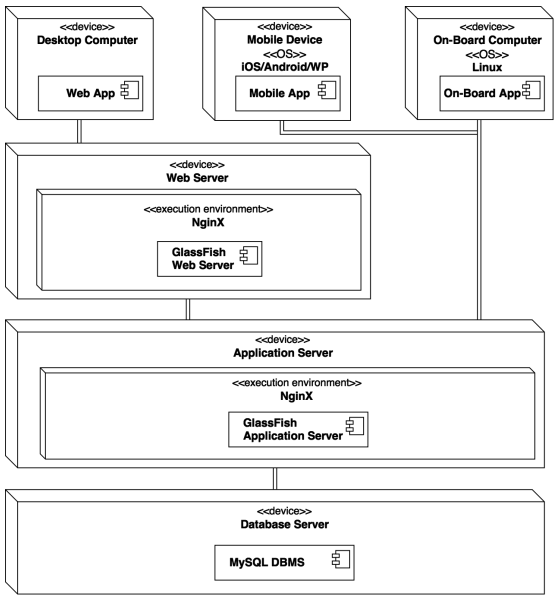

Figure 2.4: Deployment View diagram

2.4 Runtime view

In this section some sequence diagrams are presented to describe the interaction among different system's components.

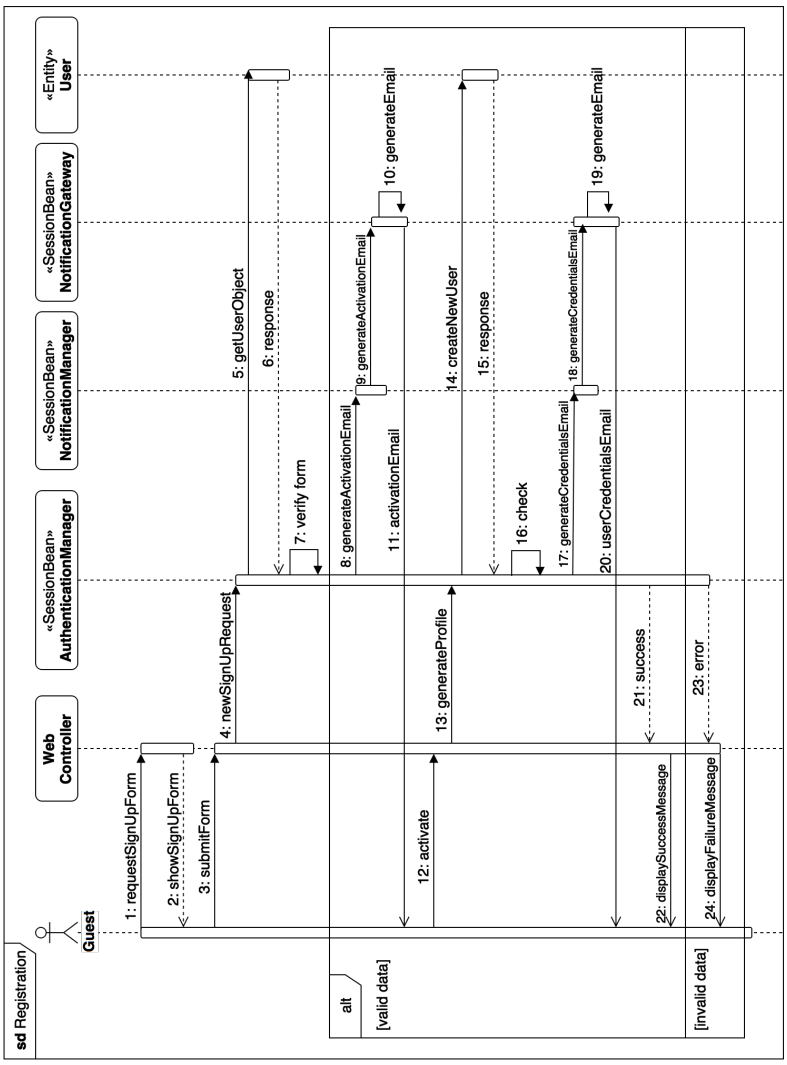

Figure 2.5: Sequence diagram for the registration process using the web application

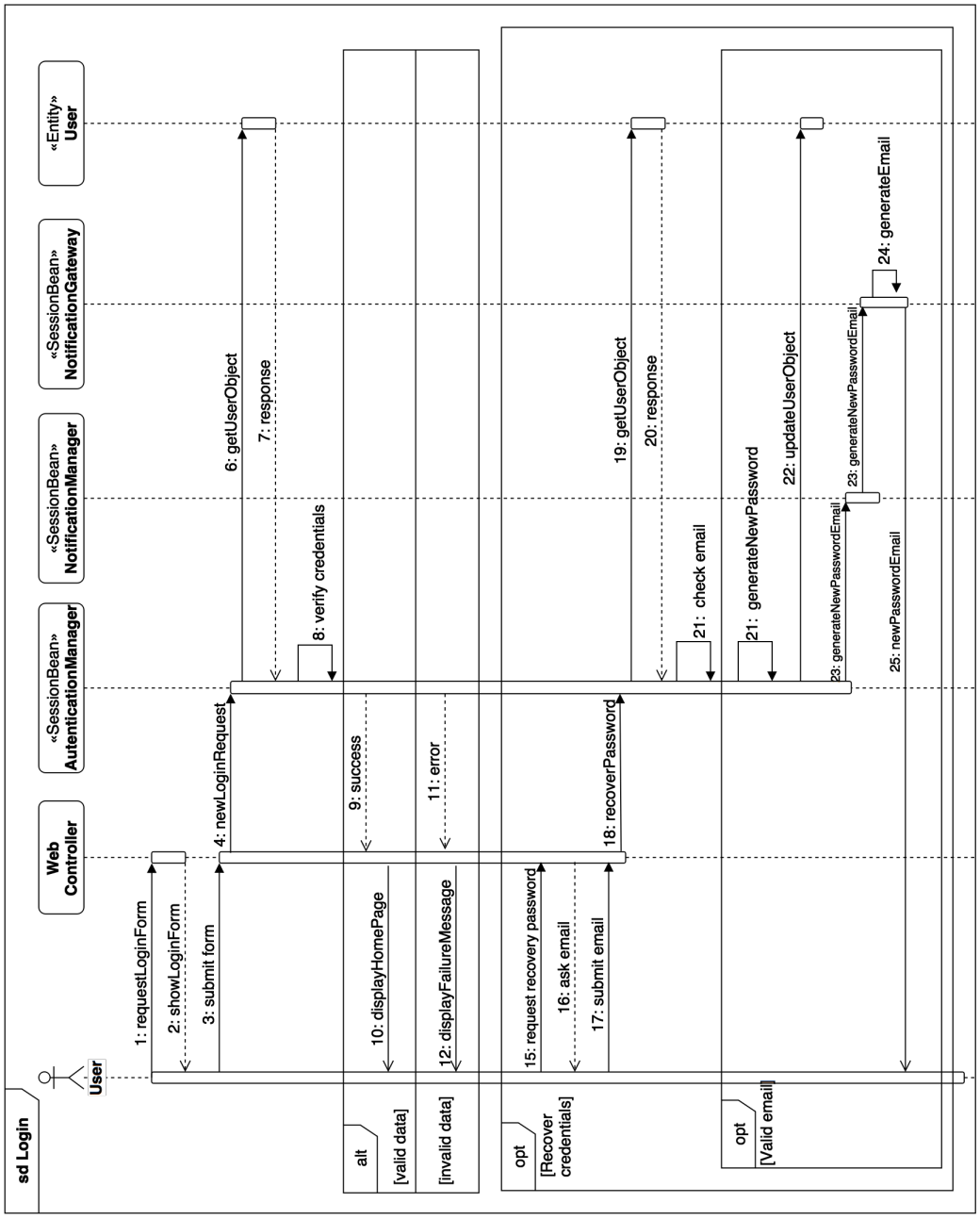

Figure 2.6: Sequence diagram for the login process using the web application

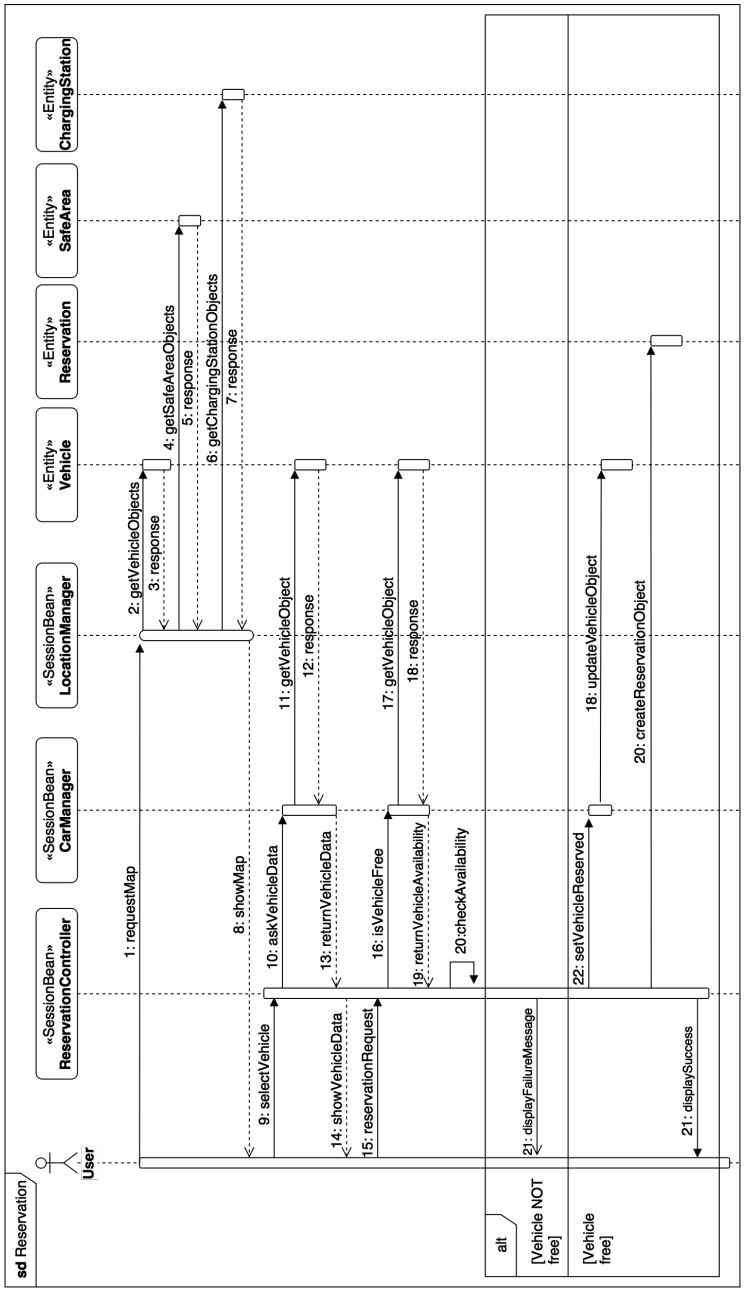

Figure 2.7: Sequence diagram for the reservation process using the mobile application

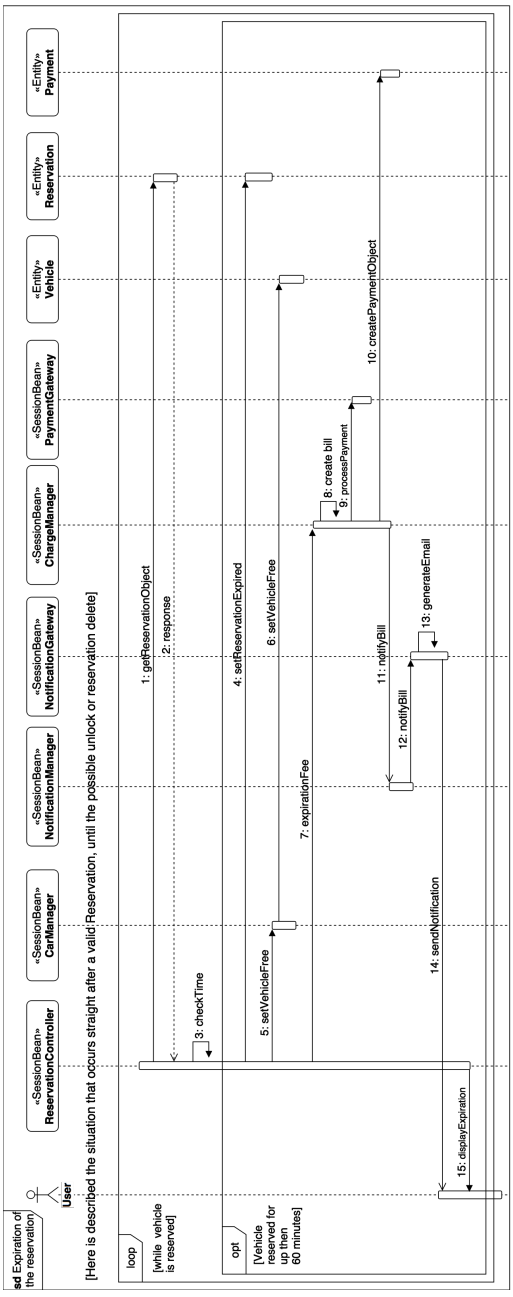

Figure 2.8: Sequence diagram for the reservation's expiration

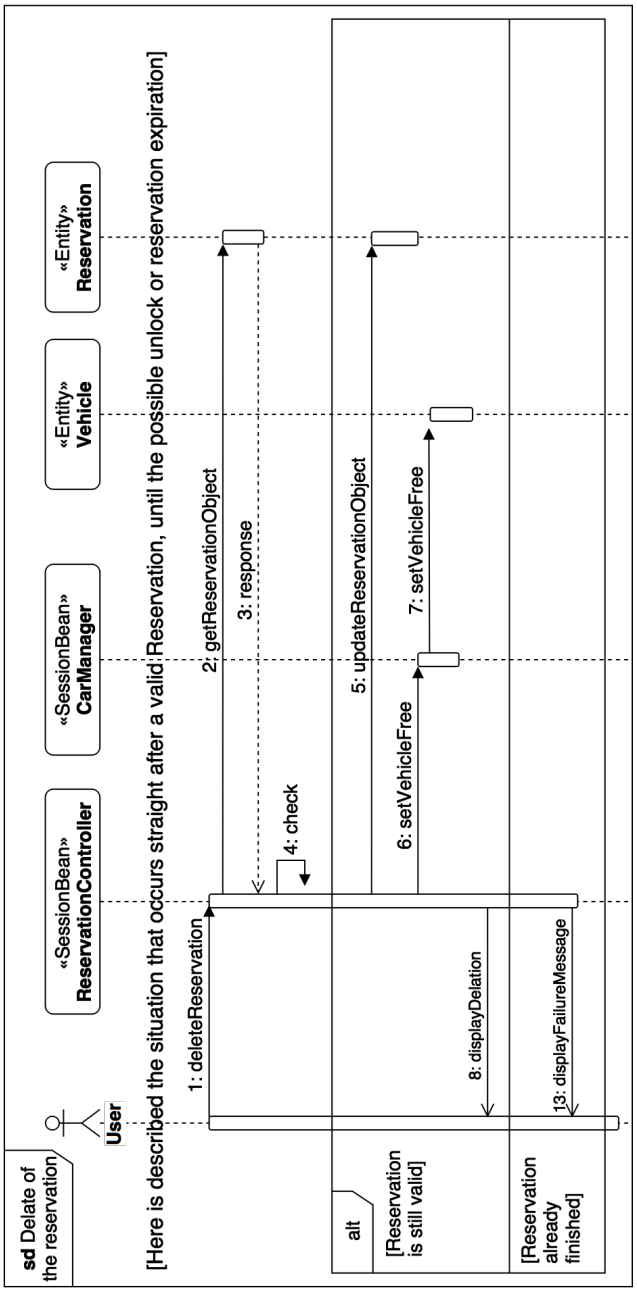

Figure 2.9: Sequence diagram for the reservation's deletion using the mobile application

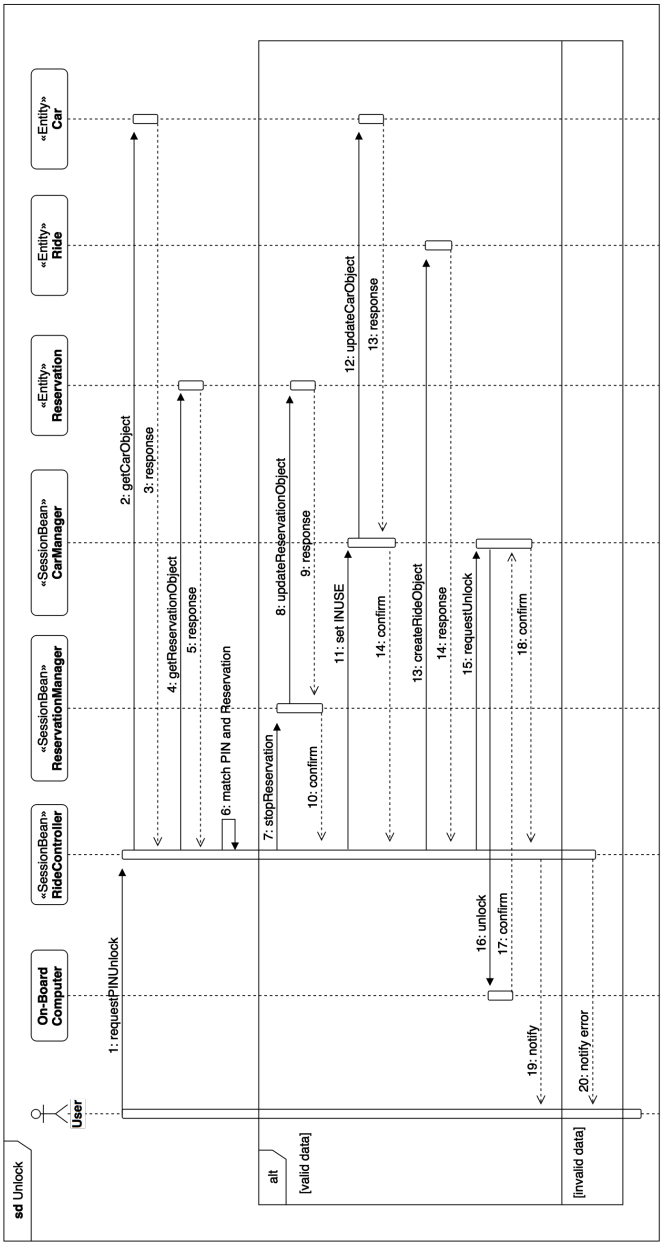

Figure 2.10: Sequence diagram for the car un-lock process. The unlockRequest (as specified in the RASD document) must contain the car's PIN code

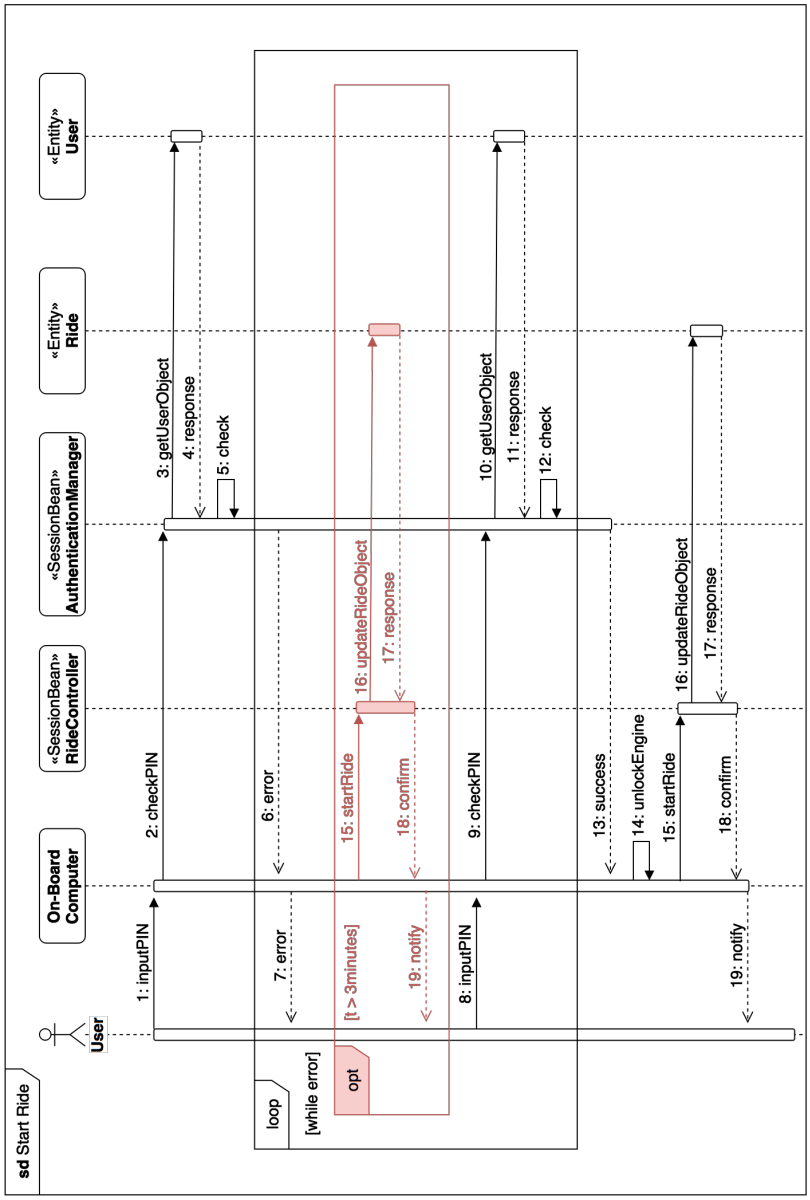

Figure 2.11: Sequence diagram for the start-ride process. It is important to highlight an abuse of terminology: the PIN keyword is used here representing the car-code (which is visible on the vehicle's windscreen) and in fig:2.10 representing the user's personal code

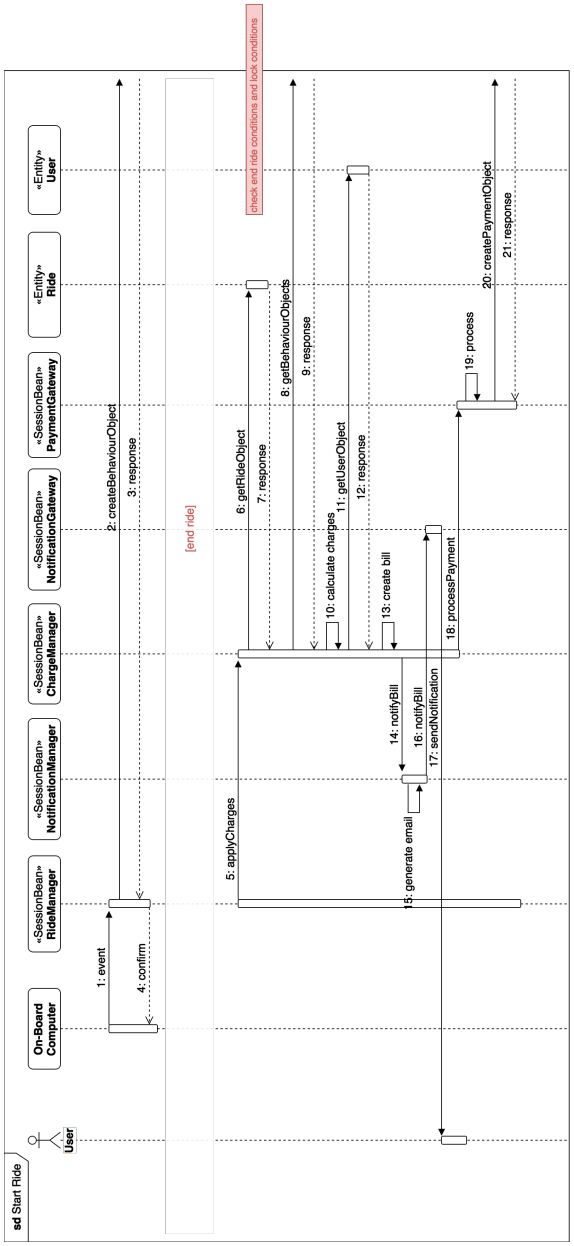

Figure 2.12: Sequence diagram for the charges computation and application. We decided to adopt an event-driven approach for the behaviour detection

2.5 Component Interfaces

In our system there are three main interfaces: the first one between the Database and the Application server, the second one between the Application server and the Web server/Mobile application/On-Board application; finally the third interface is between the Web application and the Web server.

2.5.1 Java Persistence API

The Java Persistence API (JPA) is the interface that describes the management of relational data in PowerEnJoy. It is used by the Application server to directly communicate with the Database.

2.5.2 RESTful API

A RESTful API is used by clients to interact with the Application server. A stateless architecture is achieved, in particular, using the JAX-RS API.

2.5.3 HTTPS

The web application interacts with the application server passing through the web server; it uses the HTTPS protocol enforced with a RESTful approach (Query Authentication is discussed in subsection 2.7.1).

2.6 Selected architectural styles and patterns

Building this application different architectural styles and patterns have been used.

2.6.1 4-tier JEE client-server architecture

This architectural style has been used to separate efficiently the different levels of execution. The 4 tiers componing the application are:

- **Client tier**: is the layer that interact with the users, it runs on the client machine(On-Board computer, Mobile application, Web application).
  The Mobile application interacts with the Application server using the RESTful API and will be deployed for three different architectures: iOS, Android and WP.
  The On-Board application interacts with the vehicle's sensors through the vehicle-specific API. It also interacts with the Application server using the RESTful API.
  Finally the Web application interacts with the Web server using the RESTful API too.

- **Web tier**: is implemented using JEE 7 and GlassFish Server in order to avoid conflicts. It contains Servlets, JavaServer Pages (JSP) and exposes a RESTful API. The MVC pattern is implemented in the web logic thanks to the JSP. The Servlets are useful only in specific interactions while the RESTful Api is used in the interaction with the Application server.

- **Business tier**: is implemented using JEE 7 and GlassFish Server, commonly used in large-scale projects. It contains Enterprise Java Beans(EJB), Java Persistence API (JPA) and JAX-RS which exposes a RESTful API to interact with clients and the Web tier. The data are processed (if necessary) using the EJBs and then sent to the Database by the JPA that access the DB and executes the object-relation mapping. RESTful APIs are used to interact with external systems.

- **Enterprise Information System tier**: runs EIS software and includes the enterprise infrastructure systems. It represents the data layer, where the data are stored and retrieved by the Application server. MySQL is the relational DBMS (rDBMS) chosen for the creation and the maintenance of all the application data.

2.6.2 Client-Server

The client-server communication model is highly used in this application.
The On-Board computer and the Mobile application are clients with respect to the Application Server. The Web application consist of a Web browser that is a client with respect to the Web Server. The Web server is also a client with

respect to the Application server. The Database is a server with respect to the Application server that act as a client.

2.6.3 Thin client

In order to avoid the involvement of the client machine in any logical decision all the computations is done in the Application server. This comports different important advantages for the client tier: the client application has lower operational cost for the device, a superior security is obtained, the data are synchronized and the system is highly reliable. In addition this makes the application independent from the number of clients connected.

2.6.4 MVC

The Model-View-Controller pattern has been used in this application during the implementation of the client tier. In this way we separated the model (which represent the knowledge) the view (which is a visual representation of the model) and the controller (which is the link between a user and the system).

2.7 Other design decisions

2.7.1 Authentication

Our application exposes a RESTful API, this means that the server implementation is state-less.
A RESTful authentication and a true-stateless architecture are achieved using **Query Authentication**: the log-in request generates an API token which is stored by the client and every request must be authenticated using the private token as the signing key.

Algorithm design

3.1 Computation of additional charges and discounts

One of the main requirements for the PowerEnJoy system, as discussed in the RASD, is the correct computation of charges/discounts for bad/virtuous users. We assumed that there are precedence rules regarding the fee/discount application and here a computation pseudo-algorithm is presented:

1. The On-Board computer detects a virtuous/bad behaviour
2. The On-Board computer updates the Application Server and the behaviour-detection is added to the ride object
3. When the ride ends the Application Server calculates the total charges:

```
if #detections = 0
        pass;
elif #detections.bad > 0
        apply(detections.bad.get_higher)
else
        apply(detections.virtuous.get_higher)
```

4. The Application Server notifies the user

User interface design

4.1 Mockups

Mockups for the mobile and web application have been presented and discussed in the RASD.

4.2 UX Diagrams

In this section User eXperience diagrams are presented with the intent of defining UI's screens and their interactions.
As stated in the RASD web and mobile application are almost identical and will be treated as a unique application from the UX point-of-view.

4.2.1 On-Board application

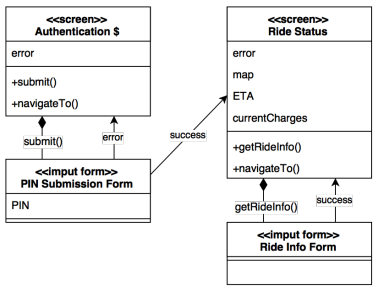

Figure 4.1: UX scheme for the on-board application

After authenticating using his Personal Identification Number, a screen is displayed to the user containing the ride-status informations (which are continuously refreshed through 'getRideInfo').

4.2.2 Mobile/Web application

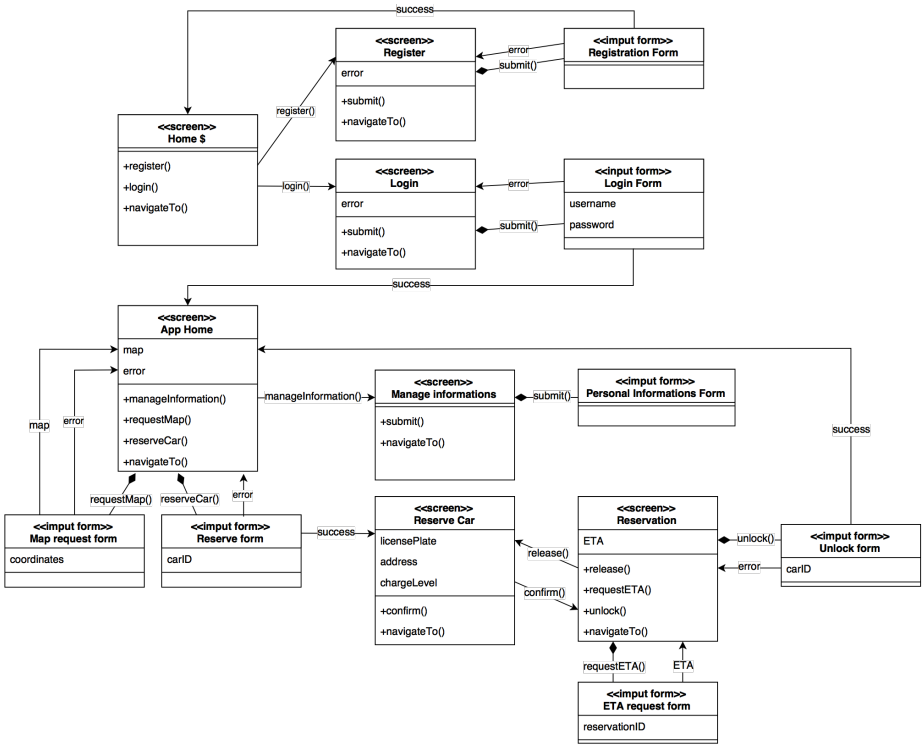

Figure 4.2: UX scheme for the mobile and web applications

After the registration/login the app home page is presented: this screen comprehend a map where all the available cars, safe area and charging stations (which are contained in the 'map' object and continuously refreshed through 'requestMap') are displayed. The user can either navigate to the 'Manage Informations' screen (where he can consult and edit his personal informations) or reserve a car after selecting it on the map. The car is tagged as 'RESERVED' after the user's confirmation and then a 'Reservation' screen is displayed: a timer shows up and the user can Release the reservation or proceed with the Unlock.
When the ride ends the payment is automatically authorized and processed without any user action.

Requirements Traceability

RASD Goals	RASD Functions	DD Component
G1, G2	Registration Login	**AuthenticationManager**
G3	Account Management	**ProfileManager**
G4	Create reservation	**LocationManager**
	Use car	
G5	Create reservation Delete reservation Reservation expiration	**ReservationController**
G6	Use car	**RideController**
G5, G6	Use car	**CarManager**
	Create reservation Delete reservation Reservation expiration	
G1, G7	Charge ride Registration	**NotificationManager**, **NotificationGateway**
G7	Charge ride Discounts & fees	**ChargeManager**, **PaymentGateway**

Appendix A: Used Tools

A.1 LaTeX

Used to format and redact this document

A.2 *g*it

Used as version control system in order to lead development

A.3 *d*raw.io

Used to draw mockups and diagrams

Appendix B: Hours of work

These are the hours of work spent by each group member in order to redact this document:

- Ruaro Nicola: 20 hours
- Gregori Giacomo: 20 hours
- Total worktime: 40 hours

Appendix C: Revisions

These sections will be eventually redacted during future post-release updates in order to approach the DD modifiability providing a comfortable and higly effective way to trace changes:

C.1 Glossary

- **v1.1** MVC and JAX-RS entries have been fixed
- **v1.1** Add Notification Handler in architecture diagram
- **v1.2** Update the ER schema and the relation schema associated with it
- **v1.2** Figure 2.8:"Sequence diagram for the reservation's expiration" has been fixed
- **v1.2** Figure 2.12:"Sequence diagram for the charges computation and application" has been fixed

Glossary

Enterprise Java Bean Enterprise Java Beans (EJB) is a development architecture for building highly scalable and robust enterprise level applications to be deployed on J2EE compliant Application Server.

free A car is tag free when no user is using it or has reserved it. It can also be defined available.

in use A car is tag in-use when an user is using it, from the unlock to the lock of the car.

Java Persistence API The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

JAX-RS Java API for RESTful Web Services (JAX-RS) is a Java programming language API spec that provides support in creating web services according to the REST architectural pattern.

Model-View-Controller Model–view–controller (MVC) is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

out of service A car is tag out of service when it is parked outside a safe area or when it is left with low battery.

reserved A car is tag reserved when an user did a reservation on that car.

RESTful A system adopting the REST approach.

Acronyms

ACID Atomicity, Consistency, Isolation and Durability.

DB Database.

DBMS Database management system.

DD Design Document.

EIS Enterprise Information System.

EJB Enterprise Java Beans.

ER Entity-Relationship Diagram: diagram that shows the relationships of entity sets stored in a database..

GPS Global Positioning System.

GUI Graphical User Interface.

HTTP Hypertext Transfer Protoco.

HTTPS HTTP over TLS, HTTP over SSL and HTTP Secure.

IEEE Institute of Electrical and Electronics Engineers.

JAX-RS Java API for RESTful Web Services.

JEE Java Enterprise Edition.

JPA Java Persistence API.

JSP JavaServer Pages.

MVC Model-View-Controller.

PIN Personal identification number.

RASD Requirements Analysis and Specification Document.

RDBMS Relational database management system.

REST REpresentational State Transfer.

RESTful REST with no session.

SQL Structured Query Language.

UX user experience design.

WP Windows Phone.

Bibliography

[1] IEEE Std 1016, *Recommended Practice for Software Design Specifications*, 2009

[2] Luca Mottola and Elisabetta Di Nitto, *Software Engineering 2: Project goal, schedule and rules*, 2016

[3] Nicola Ruaro and Giacomo Gregori, *RASD: Requirements Analysis and Specification Document*, 2016

[4] Oracle, *https://docs.oracle.com*