Integration Test Plan Document

January 15, 2017

Giacomo Bossi

Marco Nanni

Content

1. INTRODUCTION

1.1 Revision history

Version	Date	Authors	Summary
1.0	15/01/2017	Giacomo Bossi, Marco Nanni	First release

1.2 Purpose and scope

This document is the Integration Test Plan Document of PowerEnJoy system. It describes how we have planned to accomplish the integration test of the created components and it explains to the PowerEnJoy development team what to test, in which sequence, which tools are needed and which drivers need to be developed.

Testing permits to have a greater assurance that the system subcomponents we have developed are working correctly and are doing what they were supposed to do. After unit testing, the modules are grouped in order to verify their functionalities not as a single independent subcomponent, but as a component of a subsystem which has to be integrated with other modules.

In the following sections we are going to provide:

- subsystems and subcomponents that have to be integrated for testing
- integration strategy and integration sequence
- details of subcomponents
- tools and test equipment used
- drivers and data required for testing

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- Subcomponent: a subcomponent is a low level component that provides some functionalities to a subsystem
- Subsystem: a subsystem is a high-level component used to provide a specified functionality to the main system

1.3.2 Acronyms

- RASD: Requirements Analysis and Specification Document
- DD: Design Document
- ITPD: Integration Test Plan Document
- SoB: System on Board (installed inside the car)
- DBMS: Data Base Management System

1.3.3 Abbreviations

- App: Application

1.4 Reference documents

- Assignments 1, 2 and 3 of the specification document
- PowerEnJoy RASD
- PowerEnJoy DD
- http://mockito.github.io
- http://arquillian.org
- http://jmeter.apache.org

2. INTEGRATION STRATEGY

2.1 Entry criteria

We decided to proceed in parallel to the integration testing phase and the system components development. Once several modules strongly related have been developed and unit tested, they should be grouped, integrated and tested together. This allows the development team to save time by reducing possibilities of future errors when the system will be completed.

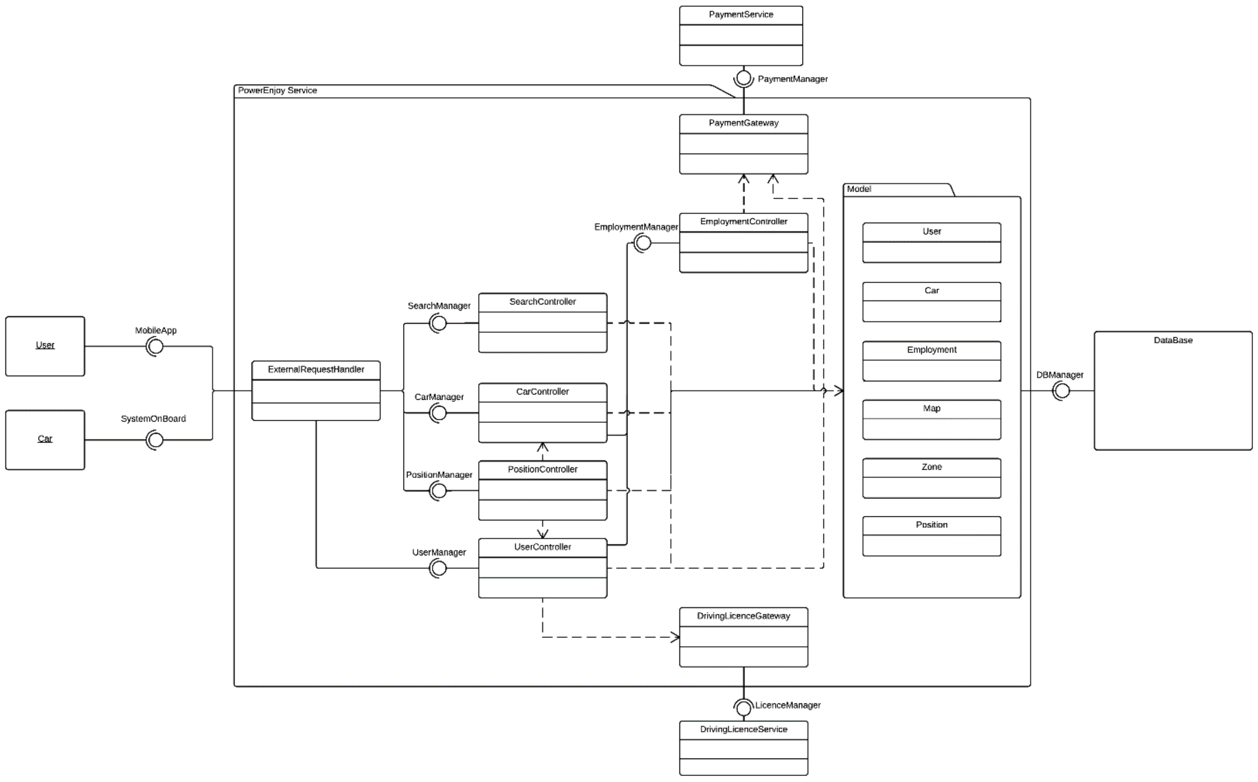
In this way, the Integration Test Plan Document has been written in parallel to the Design Document and after the Requirements Analysis and Specification Document. This has permitted the team to have a full view of the required components and their functionalities.

The integration phase can start only when certain percentages of some critical components are reached. Note that the DBMS is an external component produced by another company, so we expected to have a full completion of it before start integrating our components. Also, the Payment Service and the Driving License Service are external components.

Hence, we required that the integration between the subsystems can start when:

- 95% for the Employment System
- 75% for the Car System
- 75% for the User System
- 50% for the Car's SoB
- 60% for the User App

2.2 Elements to be integrated



From our component diagram of the design document, we have identified three subsystems that form the entire PowerEnJoy System, which are the Employment System, the Car Management System and the User Management System. Moreover, it can be seen that we have three external components that should be completed before the integration process begins, which are the DBMS, the Driving License Service and the Payment Service.

The fundamental subsystem of PowerEnJoy is the **Employment system**: it interacts with the DBMS in order to keep track of all cars' and users' information such as positions, statuses and time logs. It also interacts with the Payment Gateway to perform transaction to external payments services, once the user decides to end the ride. It is formed by several components, which are:

- New Employment, Update Employment and End Employment for the employments management
- Update User Status and Update Car Status to keep track in the DBMS of users' and cars' statuses
- Send Payment to interact with the Payment Gateway to perform transactions

Secondly, there is the **Car Management System** which performs the functionalities provided by the System on Board of the car and which communicates to the user and to the internal system. Its subcomponents are:

- Enter PIN Code, Recharge the Car, End the Ride which communicate with the user, the Employment System and the DBMS
- Update Car Position that communicates with the GPS module and the DBMS
- Occupied Seats that communicates with the Camera Sensor Module and the DBMS

Thirdly, the **User Management System** permits to have most of the functionalities provided by our PowerEnJoy system. Its subcomponents are:

- Signup that communicates with the Check Driver License and the Check Credit Card components and the DBMS
- Login, Search Car, Select Car which communicate with the DBMS
- Reserve Car, Open Car which communicate with the DBMS and the Employment System
- Update User Position that communicates with the GPS module and the DBMS

2.3 Integration testing strategy

For testing we have chosen the bottom-up approach, which is an incremental integration approach. This means that the integration testing starts at the bottom level and goes up by grouping different modules that cooperate to form a subsystem. Once a subcomponent is developed and unit tested, we integrate it with another strongly related component. After some integrations, we are able to test an entire subsystem. Once all the subsystems are completely tested, we can integrate all of them and test the entire PowerEnJoy system.

We start from the most critical and important subsystem, which is the Employment System. It provides the main functionalities of the system and it communicates with the DBMS to store information about users and cars. Note that the DBMS and other components are already existent in commerce and we assume that they are already completed.

The second ones are the Car Management System and the User Management System, which are the subsystems that interact with the users and the PowerEnJoy electric cars.

2.4 Sequence of component/function integration

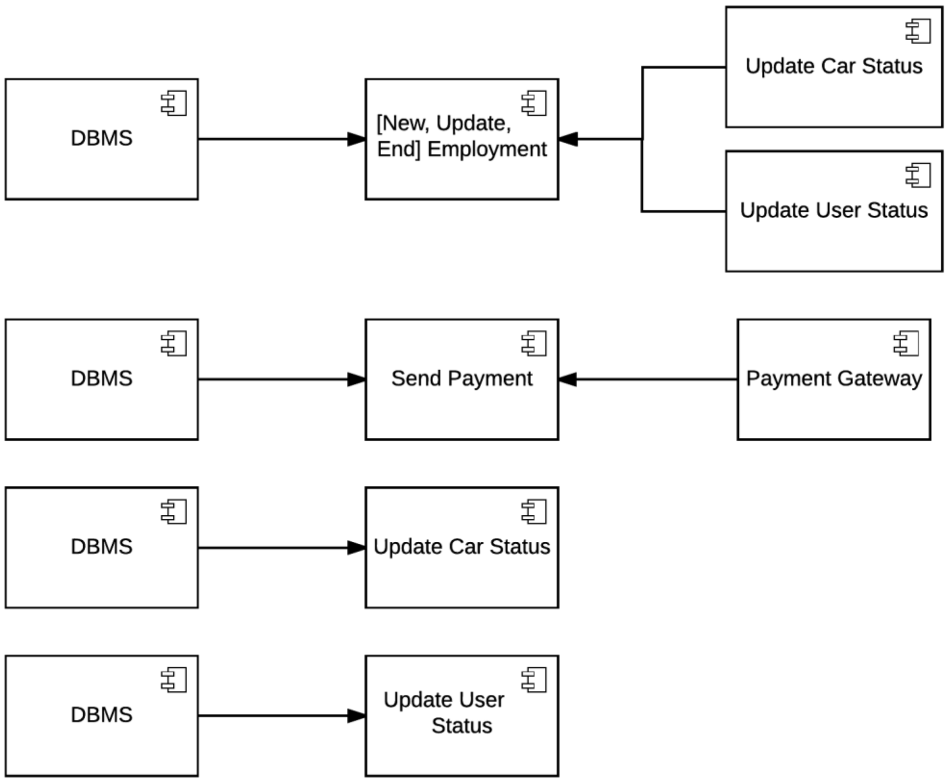
In this part of the document we are going to describe the development of the tests of the subsystems and the linked subcomponents of the PowerEnJoy System. As we have decided to acquire externally the DBMS, we do not need to test its inner component but only the interaction between the DBMS and the PowerEnJoy subsystems.

2.4.1 Software integration sequence

The integration sequence we have selected begins from the analysis of the Employment System, the core of the PowerEnJoy System. It allows the user to rent, use and pay for the employment of the cars.

- **Employment System**

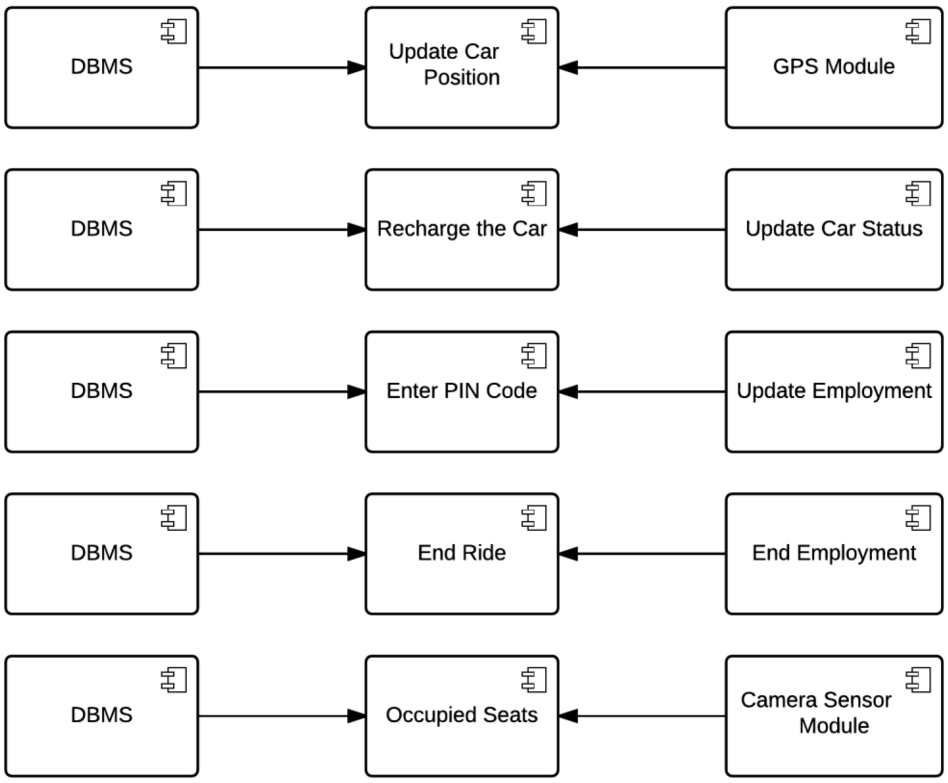
  The Employment System interacts mainly with the DBMS, but it also interacts with other proper subcomponents such as Update Status. This component verifies if the current state can be changed and provide an inner state machine (RASD 5.4.1-2). This subsystem communicates also with the payment gateway, a subcomponent that manages the payment via external bank services.



After the test of the Employment System we are going to test simultaneously the Car and the User Management System. These are two disconnected and independent subsystems, hence there aren't interactions between them.

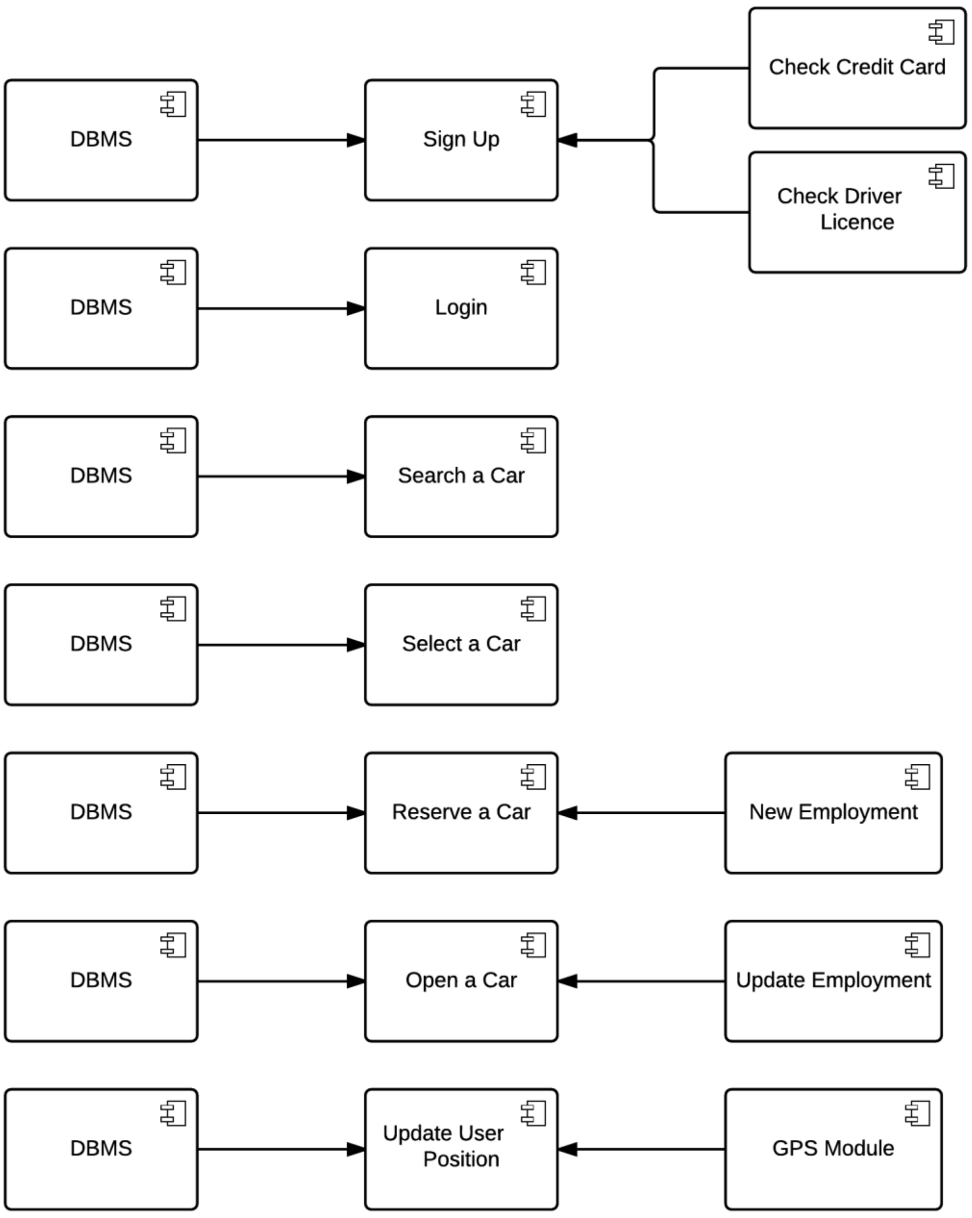
- **Car Management System**

  The Car management system interacts mainly with the DBMS and the Employment System. It interacts with the Employment System in the Recharge the Car, Enter PIN Code and End Ride subcomponents. These actions are implemented by the Car System on Board. Beyond these subcomponents, there is also the Update Car Position subcomponent, a background process that provides a stream of position data update by the DBMS. This subcomponent depends exclusively on the GPS Module, a software that interprets the signal of the GPS antenna. The car also has an internal camera sensor that sends to the DBMS an update of how many passengers are in the car.



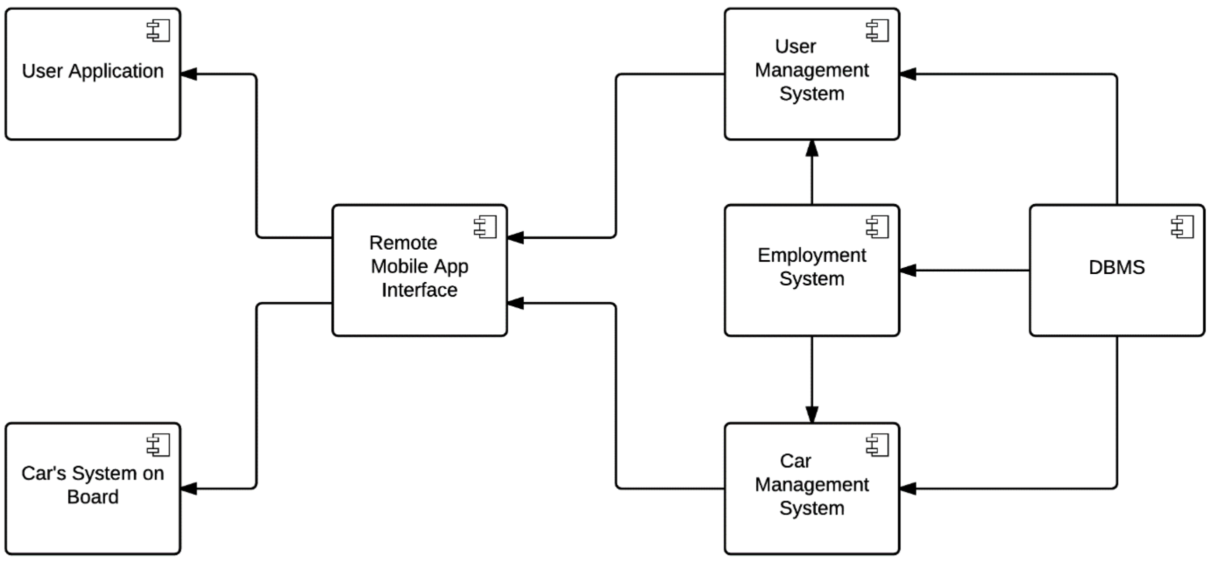
- **User Management System**

The User Management System provides all the mobile app functionalities available to the user. Effectively, through the mobile application, the user can communicate with the DBMS and with the Employment System of the PowerEnJoy Service. The subcomponents of the User Management could be split in two functional subsystems: the account management, in which we can put the Sign Up and the Login subcomponents, and the Car Employment, in which we can put the Search Car, Select Car, Reserve a Car and Open Car subcomponents. The Sign Up subcomponent communicates via the Check Credit Card with the payment gateway, that verifies the availability of the credit card; it also communicates via the Check Driver Licence with the Driver Licence Gateway, a government service that verifies the existence and the validity of the driver licence card equipped by the user. As the Car Management System, it has a GPS Module subcomponent provided by the Mobile Operation System of the smartphone.

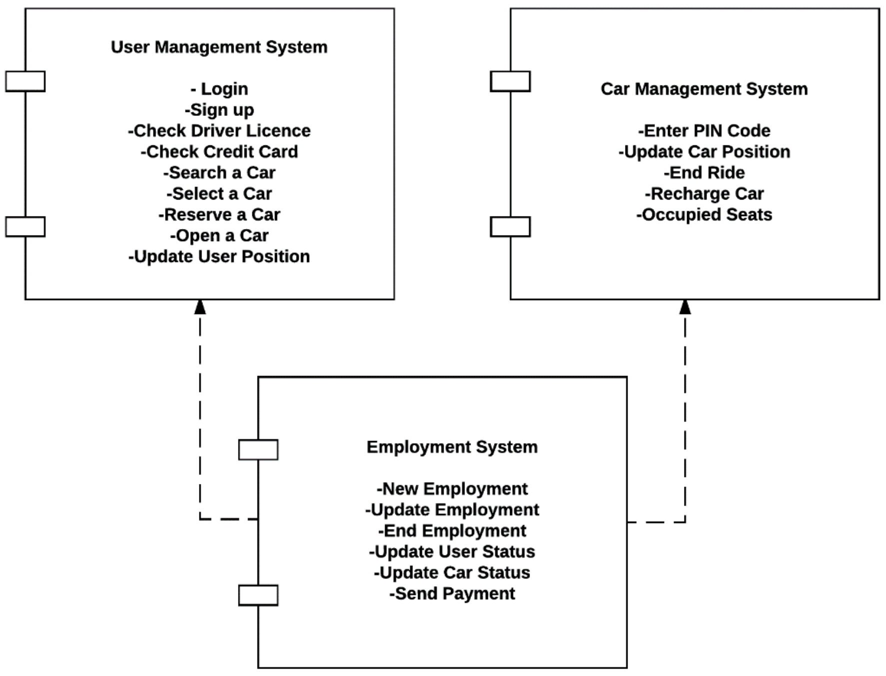


2.4.2 Subsystem integration sequence

In this diagram, we are explaining the order in which subsystems will be integrated during the integration phase. It can be seen that the DBMS is the first component that has to be completed in order to proceed with the Employment System integration. Then the User Management System and the Car Management System can be added. Once the most relevant and critical part of the system is integrated, we can proceed with the Remote Mobile App Interface and then with the System on Board installed inside the PowerEnJoy electric cars and with the User Application.



In the next diagram, we provide a view of the inner subcomponent of each subsystem in order to have a better representation of the integration between the PowerEnJoy subsystems.



3. INDIVIDUAL STEPS AND TEST DESCRIPTION

In this part of the project we will provide a detailed test explanation of each subsystem component, and for each of them we have set some exception that could be raised. This section is composed by three main parts, one for each subsystem.

3.1 User Management System

- Sign up

signUp(credentials, creditCard, drivingLicence)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Username	An InvalidUsernameException is raised
An already Existent Username	An ExistentUsernameException is raised
An Unregistered User	Return password and PINCode generated by the system

- Check Driving License

checkDriverLicence(credentials, drivingLicence)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Driving License Parameter	An InvalidDrivingLicenseException is raised
An Unmatched Driving License Owner	An UnmatchedOwnerException is raised
A Valid Driving License	Return true

- Check Credit Card

checkCreditCard(credentials, creditCard)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Credit Card Parameter	An InvalidCreditCardException is raised
An Unmatched Credit Card Owner	An UnmatchedOwnerException is raised
A Valid Credit Card	Return true

- Login

login(username, password)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Unregistered User	An UnregistredUserException is raised
A User already logged in the system	An AlreadyLoggedUserException is raised
An Invalid Password	An InvalidPasswordException is raised
A correct and unlogged User and Password combination	Return the session cookie and display the app Home Page

- Search a car

searchCar(position)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Position	An InvalidPositionException is raised
An Unknown Address	An UnknownAddressException is raised
An External-Zone Position	An ExternalZonePositionException is raised
A Valid Position	Return the set of cars located in the Position's Zone

- Select a car

selectCar(car)

Input	Effects
A no longer available Car	A NotAvailableCarException is raised
An Available Car	Show Car's Information

- Reserve a car

reserveCar(user, car)

Input	Effects
A no longer available car	A NotAvailableCarException is raised
An Available Car	Car reserved

- Open car

openCar(user, position)

Input	Effects
An Invalid User Position	A InvalidUserPositionException is raised
An Unrecognized Position	A GPSPositionException is raised
A Valid Position	If the user is located nerby the car, the vehicle will be unlocked

- Update User Position

updatePosition(position)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Unrecognized Position	A GPSPositionException is raised
A Valid Position	Make sure the DBMS modifies the user's position in the DB

3.2 Car Management System

- Enter PIN Code

enterPINCode(code)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid PIN Code	A InvalidPINCodeException is raised
A Valid PIN Code	Give access to the SoB and allow the user to turn on the car engine. Start employment charges

- Update Car Position

updatePosition(car, position)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Unrecognized Position	A GPSPositionException is raised
A Valid Position	Make sure the DBMS modifies the car's position in the DB

- End Ride

endRide(car, position)

Input	Effects
An Unrecognized Car Position	An GPSPositionException is raised
An External-Zone Position	An ExternalPositionException is raised
A Valid Position	End employment charges

- Recharge Car

rechareCar(plugSensor)

Input	Effects
Car Unplugged	Return false
Invalid Measurement	A PlugSensorException is raised
Car Plugged	Return true and starts recharging the car

- Occupied Seats

occupiedSeats(cameraSensorMeasurement)

Input	Effects
Invalid Measurement	A SeatSensorException is raised
Valid Measurement	Return the number of occupied seats in the car

3.3 Employment System

The Employment System is the core of PowerEnJoy service. Each employment matches a user with a car and a time log. This data structure must be valid, in each of its fields. End Employment can be called both from the car SoB and from the Employment System. In this case, it will be called one hour after the time log of the reservation has started, if the user that have requested an employment has not request to open the reserved car.

- New Employment

createEmployment(user, car)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Data Structure	A DataStructureException is raised
An Invalid Parameter X in Data Structure	A InvalidXParameterException is raised
A Valid Data Structure with Valid Paramters	Creates a new Employment with User and Car Data and TimeLog of the Reservation and make sure that DBMS add a new tuple in the Database

- Update Employment

updateEmployment(car), updateEmployment(user)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Data Structure	A DataStructureException is raised
An Invalid Parameter X in Data Structure	A InvalidXParameterException is raised
A Valid Data Structure but no existent Employment	A InvalidEmploymentException is raised
A Valid Data Structure with Valid Parameters	Update the Employment with User and Car Data and TimeLog of the event and make sure that DBMS add a new tuple in the Database referring to the employment event

- End Employment

endEmployment(car), endEmployment(car, user)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Data Structure	A DataStructureException is raised
An Invalid Parameter X in Data Structure	A InvalidXParameterException is raised
A Valid Data Structure but no existent Employment	A InvalidEmploymentException is raised
A Valid Data Structure with Valid Paramters	Update the Employment with User and Car Data and TimeLog of the event and make sure that DBMS adds a new tuple in the Database referring to the end event

- Update User status

updateUserStatus(user, previous, next)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid User ID	An InvalidUserException is raised
Invalid State Change	An InvalidChangeStateException is raised
A Valid User ID	Updates the user's status and makes sure that the DBMS modifies the tuple in the DB referring to that user

- Update Car status

updateCarStatus(user, previous, next)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Car ID	An InvalidCarException is raised
Invalid State Change	An InvalidChangeStateException is raised
A Valid Car ID	Updates the car's status and makes sure that the DBMS modifies the tuple in the DB referring to that car

- Send Payment

sendPayment(creditCard, charge)

Input	Effects
A Null Parameter	A NullArgumentException is raised
An Invalid Credit Card Parameter	An InvalidCreditCardException is raised
An Empty Credit Card	An EmptyCreditCardException is raised
Valid Parameters	Makes sure the PaymentGateway performs the transaction and makes sure the DBMS adds a new tuple in the DB referring to that user

4. TOOLS AND TEST EQUIPMENT

4.1 Tools

Since our RASD has already been written and discussed with customers, we do not need an extremely flexible model; thus, the easy application of the Verification & Validation model satisfies all our requests. Following this model, we are going to use different tools according to the level of the integration. In particular, we are going to use:

- Mockito and JUnit for component unit testing
- Mockito, JUnit and Arquillian to test the subsystems previously identified
- JMeter for the entire system test

Note that Mockito, JUnit and Arquillian are going to test the PowerEnJoy functional requirements, while JMeter is going to test the non-functional ones (RASD - section 3).

Arquillian is an integration testing framework for containers which allows us to execute test cases against our components and their integration. This tool verifies that the components are injected together and if the interaction with the database is right. The DBMS carries out a main role in our system given that most of the functionalities provided require data from the DB.

Apache JMeter is a load testing tool for analysing and measuring performances which permits to test heavy loads on servers, test their strength and analyse performances. With JMeter we can also set up test plans that represents the main functionalities provided to users like signup, login and a simulation of the user experience.

4.2 Test Equipment

Since PowerEnJoy system is composed of a front-end and a back-end infrastructure, we are going to specify two types of equipment used for testing.

For the front-end side, we are going to use:

- an Android 4.4+ smartphone for User Application tests
- an iOS 9.0+ smartphone for User Application tests
- a Windows Mobile 8.1+ smartphone for User Application tests
- 10’’ LCD screen installed inside the electric car for the Car Application tests

The devices previously mentioned have to connect with both a wireless mobile telecommunication service and the Global Positioning System.

For the backend side, we are going to use:

- SUSE Linux Enterprise Server 12 SP2 (release 2016-11)
- Java Enterprise Edition runtime
- Glassfish Java Application Server
- Oracle Database Management System

Other software and components can be used according to the technological changes of the releases.

5. PROGRAM STUBS AND TEST DATA REQUIRED

5.1 Program stubs and drivers

According to the bottom-up approach already explained in the Integration Strategy section, here we present all the drivers we have developed in order to test the components' functionalities and their interactions. *Since we are using a bottom-up approach, no stubs are required to test our system.*

- **Status Management Driver**: this testing module invokes the methods exposed by Update Car Status and Update User Status components

- **New Employment Driver, Update Employment Driver, End Employment Driver**: each of these testing modules invokes the methods exposed by the corresponding component to test its interaction with the DBMS and the Status Management Driver

- **Payment Gateway Driver**: this testing module checks the functionalities of the Payment Gateway

- **Send Payment Driver**: this testing module invokes the methods exposed by the Send Payment component to test its interaction with the DBMS and the Payment Gateway Driver

- **Employment Driver**: this testing module tests the interactions between all its subcomponents and the DBMS

- **Recharge Car Driver, Insert PIN Code Driver, End Ride Driver**: each of these testing modules invokes the methods exposed by the corresponding component to test its interaction with the DBMS and the Employment Driver

- **GPS Driver**: this testing module checks the functionalities of the GPS module

- **Camera Sensor Driver**: this testing module checks the functionalities of the Camera Sensor module

- **Update Car Position Driver**: this testing module invokes the methods exposed by the corresponding component and tests its interaction with the GPS Driver and the DBMS

- **Occupied Seats Driver**: this testing module invokes the methods exposed by the corresponding component and tests its interaction with the Camera Sensor Driver and the DBMS

- **Driving License Gateway Driver**: this testing module checks the functionalities of the Driving License Gateway

- **Check Credit Card Driver, Check Driver License Driver**: each of these testing modules invokes the methods exposed by the corresponding component to test its interaction with the Payment Gateway Driver and the Driving License Gateway Driver respectively

- **Sign Up Driver**: this testing module invokes the methods exposed by the corresponding component to test its interaction with the DBMS, the Check Credit Card Driver and the Check Driver License Driver

- **Login Driver, Search Car Driver, Select Car Driver**: each of these testing modules invoke the methods exposed by the corresponding component to test its interaction with the DBMS

- **Reserve Car Driver, Open Car Driver**: each of these testing modules invokes the methods exposed by the corresponding component to test its interaction with the DBMS and the Employment Driver

- **Update User Position Driver**: this testing module invokes the methods exposed by the corresponding component and tests its interaction with the GPS Driver

5.2 Test Data required

To perform the that we have previously mentioned we need different data types. In this paragraph, we will specify the data required for each different test block:

- **User Management System**:
  In this case we need different data type according to the operation performed by the user; for each operation we highlight the data that could expose unexpected behaviours:

  Sign up – A list of valid and invalid candidate user containing instances that exhibit the following exception:
    - Null Parameter
    - Invalid Username
    - Invalid Driver Licence
    - Invalid Credit Card

  Log in – A list of valid and invalid candidate user containing instances that exhibit the following exception:
    - Null Parameter
    - An Invalid Password
    - Already Logged User

  Update User Position – A list of valid and invalid position containing instances that exhibit the following exception:
    - Null Parameter
    - Invalid Position
    - An Unrecognized Position

  Search a Car – A list of valid and invalid position and address containing instances that exhibit the following exception:
    - Null Parameter
    - Invalid Position
    - Invalid Address
    - Unknown Address

  [Select, Reserve] a Car – A list of valid and invalid cars containing instances that exhibit the following exception:
    - A no Longer Available Car

- **Car Management System**:
  In this test block we need different data type according to the operation performed by the car's SoB; for each operation we highlight the data that could expose unexpected behaviours:

  Enter PIN Code – A list of valid and invalid PIN Code containing instances that exhibit the following exception:
    - Invalid Parameter
    - Null Parameter

  Update Car Position – A list of valid and invalid position containing instances that exhibit the following exception:
    - Null Parameter
    - Invalid Position

  End Ride – A list of valid and invalid cars instances that exhibit the following exception:
    - Unrecognised Car Position
    - External-Zone Position

  Recharge Car – A list of valid and invalid cars camera sensors instances that exhibit the following exception:
    - Invalid Measurement

  Occupied Seats – A list of valid and invalid cars camera sensors instances that exhibit the following exception:
    - Invalid Measurement

- **Employment System**:
  In this test block we need different data type according to the operation performed by the Employment System; for each operation we highlight the data that could generate unexpected behaviours and expose the system:

  [New, Update, End] Employment – A list of valid and invalid employment containing instances that exhibit the following exception:

  - Null Parameter
  - Invalid fields
  - Update of a non-existent employment

  Update [User, Car] Status – A list of valid and invalid cars and users containing instances that exhibit the following exception:

  - Null Parameter
  - Invalid ID
  - Invalid user change state

  Send Payment – A list of valid and invalid user’s credit card containing instances that exhibit the following exception:

  - Null Parameter
  - Empty Credit Card
  - Invalid Credit Card

6. EFFORT SPENT

Giacomo Bossi:

27/12: 1.5hrs

28/12: 2.5hrs

29/12: 2.5hrs

02/01: 2hrs

03/01: 2.5hrs

04/01: 3hrs

05/01: 2hrs

10/01: 2hrs

11/01: 3hrs

13/01: 2hrs

14/01: 2hrs

15/01: 1hr

Marco Nanni:

28/12: 2hrs

29/12: 3hrs

02/01: 2.5hrs

03/01: 2.5hrs

04/01: 3hrs

05/01: 2.5hrs

10/01: 2.5hrs

11/01: 3hrs

13/01: 2hrs

14/01: 2hrs

15/01: 1hr