# PowerEnJoy
# Integration Test Plan Document

Stefano Brandoli (mat. 875633)
Silvia Calcaterra (mat. 874887)
Samuele Conti (mat. 875708)

January 15, 2017

Version 1.0

# Contents

# 1 Introduction

## 1.1 Revision History

| Version | Date | Summary |
|---|---|---|
| 1.0 | 15-01-2017 | Initial Release |

## 1.2 Purpose and Scope

### 1.2.1 Purpose

This document is the Integration Test Plan Document (ITPD) for PowerEnjoy.

The purpose of this document is to precisely describe the plan identified to perform the integration testing activity of the high level components and subcomponents identified in the DD. It will also highlight the rationales behind the integration test strategy followed and the tools to be used to perform the integration.

The target audience for this document are the developers and specifically the testers.

### 1.2.2 Scope

The scope is to develop PowerEnJoy, a digital management system for a car-sharing service that exclusively employs electric cars. PowerEnjoy will provide all the basic functionalities available in a regular car-sharing service, plus:

- An intelligent management of the distribution of the cars in the city area.
- Users' fee incentives for a proper usage of the cars.

## 1.3 List of Definitions and Abbreviations

### 1.3.1 Definitions

- **System:** the complete PowerEnjoy software to be developed and then deployed on physical machines.
- **Subsystem:** high-level component of the system. Every subsystem corresponds to a software layer (tier) presented in the DD, however, regarding the Client Layer we are considering as subsystems both the Car Application and the Mobile Application.
- **Unit (Subcomponent):** Single low level component inside a subsystem which realizes a functionality of the subsystem.

### 1.3.2 Abbreviations

- RASD = Requirements Analysis and Specification Document.
- DD = Design Document.

- API = Application Programming Interface.
- DBMS = Database Management System.
- JPA = Java Persistence API.
- EJB = Enterprise Java Bean.
- SDK = Software Development Kit.

### 1.4 List of Reference Documents

- Specification Document of PowerEnjoy.
- PowerEnjoy RASD.
- PowerEnjoy DD.
- Integration Test Plan Example: SpinGrid.
- Integration Testing Example Document: myTaxiService.
- jUnit Documentation: `http://junit.org/junit4/javadoc/latest/index.html`
- Mockito Documentation:`http://static.javadoc.io/org.mockito/mockito-core/2.5.5/org/mockito/Mockito.html`
- Arquillian Documentation: `http://arquillian.org/modules/`
- jMeter Documentation: `http://jmeter.apache.org/usermanual/`

# 2 Integration Strategy

## 2.1 Entry Criteria

Before any kind of integration between units and then between subsystems can begin, the following conditions must be met:

- The RASD, DD and this document (ITPD) have been released.
- The estimated completion of each unit should be at least 85% according to the information provided in the DD. This because in this way the testing environment will be closer to the future production environment. The complete completion of the code will also depend on how the integration tests will be performed and on their results.
- Each unit must be unit tested using JUnit with a code coverage of: approximately 80 % for non critical units, approximately 90 % for the critical ones. The critical units regard the management of the cars and of their reservations.
- All the high prioritized bugs in each single unit must be fixed and closed.
- Each unit must be documented (using JavaDocs where possible and markdown for Swift). The level of detail and priority of the documentation depends on both the level of importance and exposure of a functionality.

## 2.2 Elements to be Integrated

In this section we will provide a list of the subsystems and their units that need to be integrated together. The high level components (subsystems) defined in the DD are:

- Car Application
- Mobile Application
- Web Layer
- Application Layer
- Database Layer

Each subsystem is then obtained by the interaction between different units. The integration phase will be done in two steps:

- Integrate the units belonging to the same subsystem. This will not regard the Database Layer, since we only need to interface and configure the DBMS.
- Integrate the subsystems interacting with each other.

## 2.3 Integration Testing Strategy

The integration testing strategy will use a mixed strategy following these approaches:

- **Bottom-Up approach:** we can start first with the integration of the most independent units which depend on few other units to function or on already developed units. In this way:
  - We will limit the number of stubs (mocks) that may be necessary, saving budget and time. However we will need some drivers.
  - The integration phase will follow more closely the development process, in fact the development will start from the simplest and less dependent units.
- **Critical Modules approach:** when the order of integration of components is not much significative by following the previous approach, we can first concentrate on the integration of the riskiest units (functional critical or algorithmically complex), since a bad behaviour of these integrations will strongly compromise the ability of our system to fulfill its goals.

## 2.4 Sequence of Component/Function Integration

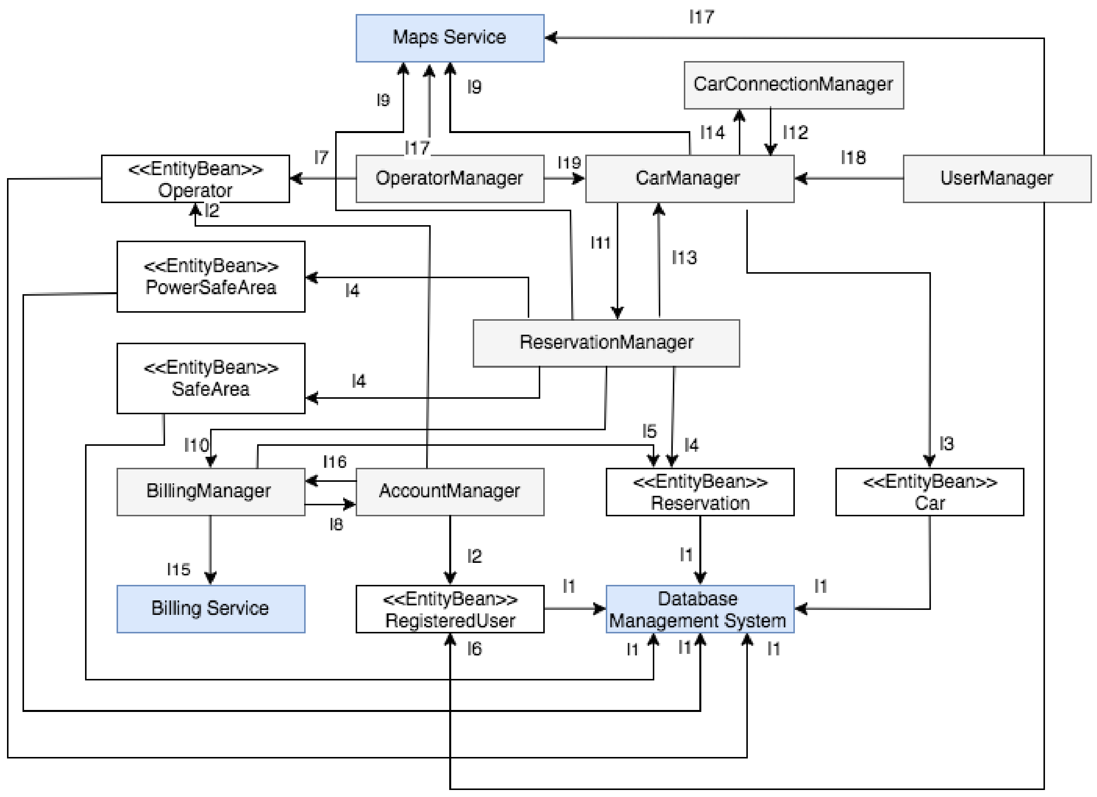
Given two components or subsystems A and B, A -> B means that A relies on at least one functionality provided by B.

### 2.4.1 Software Integration Sequence

In this subsection it is described the order of integration of the different subcomponents inside every subsystem. As stated before, to choose the priority of each subcomponent in the integration ordering, we will follow a bottom-up approach mixed with a critical modules strategy.

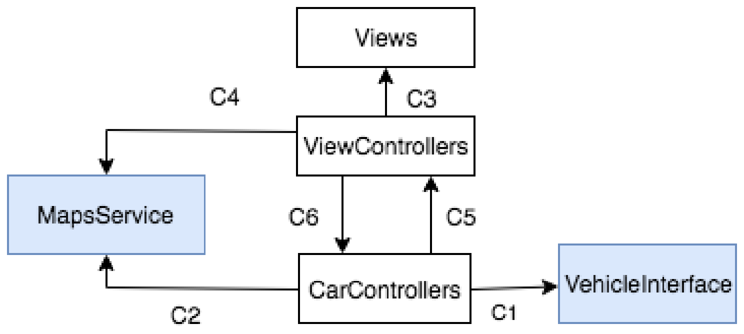
**Application Layer Integration Sequence**

| **ID** | **Integration Test** |
|---|---|
| **I1** | RegisteredUser, Reservation, Car, SafeArea, PowerSafeArea, Operator -> DBMS |
| **I2** | AccountManager -> RegisteredUser, Operator |
| **I3** | CarManager -> Car |
| **I4** | ReservationManager -> SafeArea, PowerSafeArea, Reservation |
| **I5** | BillingManager -> Reservation |
| **I6** | UserManager -> RegisteredUser |
| **I7** | OperatorManager -> Operator |
| **I8** | BillingManager -> AccountManager |
| **I9** | ReservationManager, CarManager -> MapsService |
| **I10** | ReservationManager -> BillingManager |
| **I11** | CarManager -> ReservationManager |
| **I12** | CarConnectionManager -> CarManager |
| **I13** | ReservationManager -> CarManager |
| **I14** | CarManager -> CarConnectionManager |
| **I15** | BillingManager -> BillingService |
| **I16** | AccountManager -> BillingManager |
| **I17** | UserManager, OperatorManager -> MapsService |
| **I18** | UserManager -> CarManager |
| **I19** | OperatorManager -> CarManager |



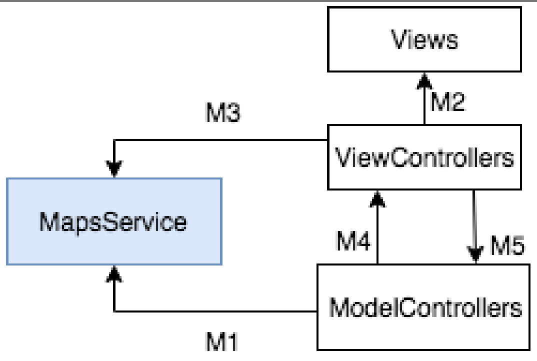
**Car Application Integration Sequence**

| ID | Integration Test |
|---|---|
| C1 | CarControllers -> VehicleInterface |
| C2 | CarControllers -> MapsService |
| C3 | ViewControllers -> Views |
| C4 | ViewControllers -> MapsService |
| C5 | CarControllers -> ViewControllers |
| C6 | ViewControllers -> CarControllers |



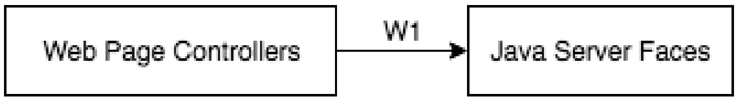
**Mobile Application Integration Sequence**

| ID | Integration Test |
|---|---|
| **M1** | ModelControllers -> Maps Service |
| **M2** | ViewControllers -> Views |
| **M3** | ViewControllers -> Maps Service |
| **M4** | ModelControllers -> ViewControllers |
| **M5** | ViewControllers -> ModelControllers |



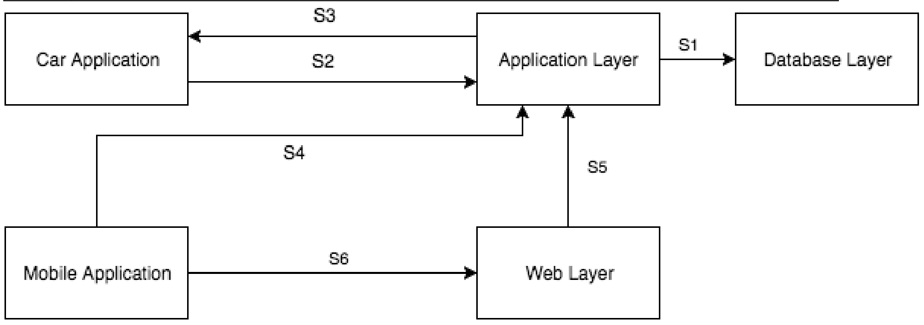
**Web Layer Integration Sequence**

| ID | Integration Test |
|---|---|
| **W1** | WebPage Controllers -> JavaServer Faces |



### 2.4.2 Subsystems Integration Sequence

| **ID** | **Integration Test** |
|---|---|
| **S1** | Application Layer -> Database Layer |
| **S2** | Car Application -> Application Layer |
| **S3** | Application Layer -> Car Application |
| **S4** | Mobile Application -> Application Layer |
| **S5** | Web Layer -> Application Layer |
| **S6** | Mobile Application -> Web Layer |



# 3 Individual Steps and Test Description

## 3.1 Application Layer Integration Sequence

### 3.1.1 Integration test case I1

| **Test case Identifier** | I1T1 |
|---|---|
| **Test Purpose** | Verify that the RegisteredUser entity executes the right queries on the DBMS. |
| **Test Item(s)** | **RegisteredUser -> DBMS** |
| **Input Specification** | Query on the RegisteredUser table. |
| **Output Specification** | Check that the result of the query is correct. |
| **Environmental Needs** | A Test Database is setup and configured in a way very close to the final production environment (see Test Database). |

| **Test case Identifier** | I1T2 |
|---|---|
| **Test Purpose** | Verify that the Reservation entity executes the right queries on the DBMS. |
| **Test Item(s)** | **Reservation -> DBMS** |
| **Input Specification** | Query on the Reservation table. |
| **Output Specification** | Check that the result of the query is correct. |
| **Environmental Needs** | A Test Database is setup and configured in a way very close to the final production environment (see Test Database). |

| **Test case Identifier** | I1T3 |
|---|---|
| **Test Purpose** | Verify that the Car entity executes the right queries on the DBMS. |
| **Test Item(s)** | **Car -> DBMS** |
| **Input Specification** | Query on the Car table. |
| **Output Specification** | Check that the result of the query is correct. |
| **Environmental Needs** | A Test Database is setup and configured in a way very close to the final production environment (see Test Database). |

| **Test case Identifier** | I1T4 |
|---|---|
| **Test Purpose** | Verify that the SafeArea entity executes the right queries on the DBMS. |
| **Test Item(s)** | **SafeArea -> DBMS** |
| **Input Specification** | Query on the SafeArea table. |
| **Output Specification** | Check that the result of the query is correct. |
| **Environmental Needs** | A Test Database is setup and configured in a way very close to the final production environment (see Test Database). |

| | |
|---|---|
| **Test case Identifier** | I1T5 |
| **Test Purpose** | Verify that the PowerSafeArea entity executes the right queries on the DBMS. |
| **Test Item(s)** | **PowerSafeArea-> DBMS** |
| **Input Specification** | Query on the PowerSafeArea table. |
| **Output Specification** | Check that the result of the query is correct. |
| **Environmental Needs** | A Test Database is setup and configured in a way very close to the final production environment (see Test Database). |

| | |
|---|---|
| **Test case Identifier** | I1T6 |
| **Test Purpose** | Verify that the Operator entity executes the right queries on the DBMS. |
| **Test Item(s)** | **Operator -> DBMS** |
| **Input Specification** | Query on the Operator table. |
| **Output Specification** | Check that the result of the query is correct. |
| **Environmental Needs** | A Test Database is setup and configured in a way very close to the final production environment (see Test Database). |

### 3.1.2 Integration test case I2

| | |
|---|---|
| **Test case Identifier** | I2T1 |
| **Test Purpose** | Check that the AccountManager is able to create a new RegisteredUser entity and retrieve information from it. |
| **Test Item(s)** | **AccountManager -> RegisteredUser** |
| **Input Specification** | • Create a new RegisteredUser entity<br>• Request the RegisteredUser's email and password<br>• Create a new value for the PIN attribute of a RegisteredUser entity<br>• Set the suspended boolean value of a RegisteredUser entity |
| **Output Specification** | Check that the correct methods are called in the RegisteredUser entity and that a new RegisteredUser entity has been correctly created. |
| **Environmental Needs** | I1T1 succeeded. |

| | |
|---|---|
| **Test case Identifier** | I2T2 |
| **Test Purpose** | Check that the AccountManager is able to create a new Operator entity. |
| **Test Item(s)** | **AccountManager -> Operator** |
| **Input Specification** | Create a new Operator entity. |
| **Output Specification** | Check that a new Operator entity has been correctly created. |
| **Environmental Needs** | I1T6 succeeded. |

#### 3.1.3 Integration test case I3

| | |
|---|---|
| **Test case Identifier** | I3T1 |
| **Test Purpose** | Check that the CarManager is able to retrieve information from the Car entity and to set its attributes. |
| **Test Item(s)** | **CarManager -> Car** |
| **Input Specification** | • Retrieve information about the Car entity, like car position, passengers number, remaining battery percentage, battery charge status, car status, car problem.<br>• Test interactions which are going to modify the car attributes: like update the car with the newest received GPS position, signaling a problem occured to the car, update the car passengers. |
| **Output Specification** | Check that the correct methods are called in the Car entity and that the correct car status is set. |
| **Environmental Needs** | I1T3 succeeded. |

### 3.1.4 Integration test case I4

| | |
|---|---|
| **Test case Identifier** | I4T1 |
| **Test Purpose** | Check that the ReservationManager is able to retrieve information from the SafeArea entity and to change the number of its remaining parking slots. |
| **Test Item(s)** | **ReservationManager -> SafeArea** |
| **Input Specification** | • Retrieve the coordinates of the SafeArea entity and the number of remaining parking slots.<br>• Request the update of the number of remaining parking slots in the SafeArea entity. |
| **Output Specification** | Check that the correct methods are called in the SafeArea entity and that the number of its remaining parking slots is updated correctly. |
| **Environmental Needs** | I1T4 succeeded. |

| | |
|---|---|
| **Test case Identifier** | I4T2 |
| **Test Purpose** | Check that the ReservationManager is able to retrieve information from the PowerSafeArea entity and to change the number of its remaining parking slots and plugs in use. |
| **Test Item(s)** | **ReservationManager -> PowerSafeArea** |
| **Input Specification** | • Retrieve the coordinates of the PowerSafeArea entity and the number of remaining parking slots.<br>• Request the update of the number of remaining parking slots in the PowerSafeArea entity.<br>• Request the update of the number of plugs in use in the SafeArea entity. |
| **Output Specification** | Check that the correct methods are called in the PowerSafeArea entity and that the number of its remaining parking slots and plugs in use are updated correctly. |
| **Environmental Needs** | I1T5 succeeded. |

| | |
|---|---|
| **Test case Identifier** | I4T3 |
| **Test Purpose** | Check that the ReservationManager is able to create a new Reservation entity, to modify its attributes and to retrieve information from it. |
| **Test Item(s)** | **ReservationManager -> Reservation** |
| **Input Specification** | • Create a new Reservation entity<br>• End a reservation (by updating the corresponding boolean attribute)<br>• Check pending reservations<br>• Retrieve the Reservation entity attribute values<br>• Update the Reservation entity attribute values |
| **Output Specification** | Check that the correct methods are called in the Reservation entity, that a new Reservation entity has been correctly created and that its attribute values are updated correctly. |
| **Environmental Needs** | I1T2 succeeded. |

#### 3.1.5 Integration test case I5

| | |
|---|---|
| **Test case Identifier** | I5T1 |
| **Test Purpose** | Evaluate if BillingManager is capable to retrieve information from a Reservation entity, such as the duration of a reservation and the activation of a reservation. |
| **Test Item(s)** | **BillingManager -> Reservation** |
| **Input Specification** | • Request the total duration of a reservation.<br>• Request the pickedUp boolean. |
| **Output Specification** | Check if the correct methods are called in Reservation and if the returned values are as expected. |
| **Environmental Needs** | I1T2 succeeded. |

### 3.1.6 Integration test case I6

| | |
|---|---|
| **Test case Identifier** | I6T1 |
| **Test Purpose** | Check if UserManager can retrieve information from a RegisteredUser entity, such as the User's PIN and location. |
| **Test Item(s)** | **UserManager -> RegisteredUser** |
| **Input Specification** | • Request a User's PIN. <br> • Request a User's position. |
| **Output Specification** | Check if the correct methods are called in RegisteredUser and if the values returned (such as PIN and User's position) have the expected values. |
| **Environmental Needs** | I1T1 succeeded. |

### 3.1.7 Integration test case I7

| | |
|---|---|
| **Test case Identifier** | I7T1 |
| **Test Purpose** | Evaluate if OperatorManager can retrieve information from a Operator entity, such as the car associated to the maintenance performed by that Operator. |
| **Test Item(s)** | **OperatorManager -> Operator** |
| **Input Specification** | Request the car currently associated with the Operator. |
| **Output Specification** | Check if the correct methods are called in Operator and check if the returned value of the attribute carUnderMaintenance is correct. |
| **Environmental Needs** | I1T6 succeeded. |

### 3.1.8 Integration test case I8

| | |
|---|---|
| **Test case Identifier** | I8T1 |
| **Test Purpose** | Check if BillingManager can obtain from AccountManager the payment token of a User and eventually update it. Check if BillingManager can request and obtain the update of some informations of the User, such as its suspended status. |
| **Test Item(s)** | **BillingManager -> AccountManager** |
| **Input Specification** | • Request the Payment token of a User.<br>• Request the suspended status of a User.<br>• Update the suspended status of a User.<br>• Update the paymentToken of a User. |
| **Output Specification** | Check if the correct methods in AccountManager are called and if the expected informations of the User are correctly returned or updated. |
| **Environmental Needs** | I1T1 succeeded. |

### 3.1.9 Integration test case I9

| | |
|---|---|
| **Test case Identifier** | I9T1 |
| **Test Purpose** | Evaluate if ReservationManager and CarManager can properly interact with the external MapsService. |
| **Test Item(s)** | **ReservationManager, CarManager -> MapsService** |
| **Input Specification** | A pair of coordinates corresponding to the car's position or to the User's position. |
| **Output Specification** | Check whether the returned answers from the MapsService API are as expected. |
| **Environmental Needs** | N/A |

### 3.1.10 Integration test case I10

| | |
|---|---|
| **Test case Identifier** | I10T1 |
| **Test Purpose** | Check that the ReservationManager can correctly interact with the BillingManager to apply discounts and penalties to a user's bill. |
| **Test Item(s)** | **ReservationManager -> BillingManager** |
| **Input Specification** | The specific amount of money representing the original user's bill before discounts and penalties evaluation. |
| **Output Specification** | • Check that all the necessary discounts are applied to the bill<br>• Check that all the necessary penalties are applied to the bill |
| **Environmental Needs** | I8 succeeded. |

### 3.1.11 Integration test case I11

| | |
|---|---|
| **Test case Identifier** | I11T1 |
| **Test Purpose** | Verify that the CarManager can correctly request to the ReservationManager operations related to the management of the User reservation. |
| **Test Item(s)** | **CarManager -> ReservationManager** |
| **Input Specification** | • Request to handle the termination of a Reservation<br>• Request to make a stop<br>• Request to activate the money saving option<br>• Request to handle a car failure |
| **Output Specification** | Check that the correct methods of the ReservationManager have been called. |
| **Environmental Needs** | I4T3, I9, I10 succeeded. |

### 3.1.12 Integration test case I12

| | |
|---|---|
| **Test case Identifier** | I12T1 |
| **Test Purpose** | Check that the CarConnectionManager can correctly communicate with the CarManager. |
| **Test Item(s)** | **CarConnectionManager -> CarManager** |
| **Input Specification** | A String following the structure of the messages to be exchanged on the WebSocket Protocol between the Car Application and Application Layer. |
| **Output Specification** | Check that the String has been correctly retrieved by the CarManager. |
| **Environmental Needs** | I3, I9, I11 succeeded. |

### 3.1.13 Integration test case I13

| | |
|---|---|
| **Test case Identifier** | I13T1 |
| **Test Purpose** | Verify that the ReservationManager can correctly interact with the CarManager to modify the attributes of a Car object and to request an actuation of a command on a PowerEnjoy Car. |
| **Test Item(s)** | **ReservationManager -> CarManager** |
| **Input Specification** | • Request the change of the car status<br>• Request the retrieval of the car status and coordinates<br>• Request the lock and unlock of a car |
| **Output Specification** | Check that the correct methods of the CarManager are called. |
| **Environmental Needs** | I3, I9 succeeded. |

### 3.1.14 Integration test case I14

| | |
|---|---|
| **Test case Identifier** | I14T1 |
| **Test Purpose** | Check that the CarManager is able to communicate correctly with the CarConnectionManager. |
| **Test Item(s)** | **CarManager -> CarConnectionManager** |
| **Input Specification** | A String representing the message that the CarManager needs to send. |
| **Output Specification** | Check that the String has been correctly received by the CarConnectionManager. |
| **Environmental Needs** | N/A |

### 3.1.15 Integration test case I15

| | |
|---|---|
| **Test case Identifier** | I15T1 |
| **Test Purpose** | Check if the BillingManager can correctly interact with the external BillingService to handle users' payments. |
| **Test Item(s)** | **BillingManager -> BillingService** |
| **Input Specification** | • Request a PaymentToken to the BillingService given the payment informations sent by a User<br>• Request to charge the User through its PaymentToken<br>• Check if the user has been correctly charged through the BillingService APIs. |
| **Output Specification** | Check whether all the functionalities related to user's payment have a positive outcome. |
| **Environmental Needs** | An account with the BillingService provider should be set up and put in Testing Mode (see Test Data). |

### 3.1.16 Integration test case I16

| | |
|---|---|
| **Test case Identifier** | I16T1 |
| **Test Purpose** | Evaluate if the AccountManager can request a paymentToken for a User. |
| **Test Item(s)** | **AccountManager -> BillingManager** |
| **Input Specification** | Payment informations provided by the User during the registration procedure. |
| **Output Specification** | Check if a paymentToken is correctly returned and associated to the expected User. |
| **Environmental Needs** | I1T1, I16 succeeded |

### 3.1.17 Integration test case I17

| | |
|---|---|
| **Test case Identifier** | I17T1 |
| **Test Purpose** | Check if UserManager and OperatorManager can deal correctly with some map functionalities offered through the interaction with the MapsService. |
| **Test Item(s)** | **UserManager, OperatorManager -> MapsService** |
| **Input Specification** | One or more pairs of coordinates that correspond to users' and cars' positions. |
| **Output Specification** | Check whether the expected objects returned by the MapsService APIs are correct. |
| **Environmental Needs** | N/A |

#### 3.1.18 Integration test case I18

| | |
|---|---|
| **Test case Identifier** | I18T1 |
| **Test Purpose** | Evaluate if the UserManager is interacting correctly with the CarManager to provide features such as the location of all the available cars and the unlock of a car given a PIN inserted by the User. |
| **Test Item(s)** | **UserManager -> CarManager** |
| **Input Specification** | • Request the position of all the available cars.<br>• Request the unlock of a given car given a PIN inserted by the User. |
| **Output Specification** | • Check if the returned coordinates from CarManager correspond to the actual positions of all the available cars.<br>• Check if the boolean status of the unlock of the car is correctly updated. |
| **Environmental Needs** | I3, I9, I14 succeeded. |

#### 3.1.19 Integration test case I19

| | |
|---|---|
| **Test case Identifier** | I19T1 |
| **Test Purpose** | Check if the OperatorManager can retrieve through the CarManager the position of the car under the maintenance of an Operator |
| **Test Item(s)** | **OperatorManager -> CarManager** |
| **Input Specification** | Request the given car's position. |
| **Output Specification** | Check if the returned coordinates correspond to the actual car under the maintenance of the given Operator. |
| **Environmental Needs** | I3, I14 succeeded. |

## 3.2 Car Application integration sequence

### 3.2.1 Integration test case C1

| | |
|---|---|
| **Test case Identifier** | C1 |
| **Test Purpose** | Check if the CarControllers are able to retrieve data from the external VehicleInterface. |
| **Test Item(s)** | **CarControllers -> VehicleInterface** |
| **Input Specification** | Request all the kind of data retrievable from the VehicleInterface, such as the battery percentage, the status of the doors etc. |
| **Output Specification** | Check if the interaction goes well and the retrieved data are as expected. |
| **Environmental Needs** | VehicleInterface stub. |

### 3.2.2 Integration test case C2

| | |
|---|---|
| **Test case Identifier** | C2 |
| **Test Purpose** | Check if CarControllers can correctly interact with the external MapsService. |
| **Test Item(s)** | **CarControllers -> MapsService** |
| **Input Specification** | Request a path to a certain destination. |
| **Output Specification** | Check if CarControllers are able to obtain the correct information from MapsService. |
| **Environmental Needs** | N/A |

### 3.2.3 Integration test case C3

| | |
|---|---|
| **Test case Identifier** | C3 |
| **Test Purpose** | Check if ViewControllers are correctly linked to the respective Views and if the Views are updated correctly. |
| **Test Item(s)** | **ViewControllers -> Views** |
| **Input Specification** | • Link to Views.<br>• Update the Views. |
| **Output Specification** | Check if the updates work as expected. |
| **Environmental Needs** | N/A |

### 3.2.4 Integration test case C4

| | |
|---|---|
| **Test case Identifier** | C4 |
| **Test Purpose** | Check if ViewControllers interacts correctly with the external MapsService. |
| **Test Item(s)** | **ViewControllers -> MapsService** |
| **Input Specification** | Request a visualization on the map of some coordinates. |
| **Output Specification** | Check if ViewControllers can retrieve correctly the visualization provided by the MapsService. |
| **Environmental Needs** | N/A |

### 3.2.5 Integration test case C5

| | |
|---|---|
| **Test case Identifier** | C5 |
| **Test Purpose** | Check if CarControllers can request correctly to ViewControllers some modification on the visualization of the data. |
| **Test Item(s)** | **CarControllers -> ViewControllers** |
| **Input Specification** | Provide updated data to ViewControllers. |
| **Output Specification** | Check if ViewControllers correctly receives the updated data and forwards it to the linked Views. |
| **Environmental Needs** | C3 succeeded. |

### 3.2.6 Integration test case C6

| | |
|---|---|
| **Test case Identifier** | C6 |
| **Test Purpose** | Check if ViewControllers can communicate to CarControllers the data inserted by the external User through Views. |
| **Test Item(s)** | **ViewControllers -> CarControllers** |
| **Input Specification** | Forward data inserted by the User through Views. |
| **Output Specification** | Check if CarControllers correctly retrieves the data from ViewControllers. |
| **Environmental Needs** | C3 succeeded. |

## 3.3 Web Layer integration sequence

### 3.3.1 Integration test case W1

| | |
|---|---|
| **Test case Identifier** | W1 |
| **Test Purpose** | Check if the WebPage Controllers are able to handle correctly the JavaServer Faces components. |
| **Test Item(s)** | **WebPage Controllers -> JavaServer Faces** |
| **Input Specification** | Call the different JavaServer Faces needed by the User. |
| **Output Specification** | Check if JavaServer Faces are correctly called and if the visualization is as expected. |
| **Environmental Needs** | N/A |

## 3.4 Mobile Application

### 3.4.1 Integration test case M1

| | |
|---|---|
| **Test case Identifier** | M1T1 |
| **Test Purpose** | Check that the ModelControllers are able to interact correctly with the external MapsService. |
| **Test Item(s)** | **ModelControllers -> MapsService** |
| **Input Specification** | • Request to mark certain positions as safe areas or power safe areas<br>• Request to mark the positions of the cars in the city area |
| **Output Specification** | Check that the MapsService can correctly mark the positions inside the city. |
| **Environmental Needs** | N/A |

### 3.4.2 Integration test case M2

| | |
|---|---|
| **Test case Identifier** | M2T1 |
| **Test Purpose** | Check if ViewControllers are correctly linked to the respective Views and if the Views are updated correctly. |
| **Test Item(s)** | **ViewControllers -> Views** |
| **Input Specification** | • Link to Views<br>• Update the Views |
| **Output Specification** | Check that the Views are correctly linked to the respective ViewControllers and correctly updated. |
| **Environmental Needs** | N/A |

### 3.4.3 Integration test case M3

| | |
|---|---|
| **Test case Identifier** | M3T1 |
| **Test Purpose** | Check that the ViewControllers are able to interact correctly with the external MapsService. |
| **Test Item(s)** | **ViewControllers -> MapsService** |
| **Input Specification** | Request the visualization of a map with some coordinates and/or items |
| **Output Specification** | Check that the ViewControllers can correctly retrieve the visualization provided by the MapsService. |
| **Environmental Needs** | N/A |

### 3.4.4 Integration test case M4

| | |
|---|---|
| **Test case Identifier** | M4T1 |
| **Test Purpose** | Check that the ModelControllers can correctly request to ViewController some modifications on the visualization of data. |
| **Test Item(s)** | **ModelControllers -> ViewControllers** |
| **Input Specification** | Provide updated data to ViewController. |
| **Output Specification** | Check that the ViewController correctly updates the visualization of the data. |
| **Environmental Needs** | M2, M3 succeeded. |

### 3.4.5 Integration test case M5

| | |
|---|---|
| **Test case Identifier** | M5T1 |
| **Test Purpose** | Check that the ViewControllers can communicate to ModelControllers the data inserted by an external user through the Views. |
| **Test Item(s)** | **ViewControllers -> ModelControllers** |
| **Input Specification** | Send to ModelControllers the data inserted by an user through the Views. |
| **Output Specification** | Check that the ModelControllers can correctly retrieve the data sent by the ViewControllers. |
| **Environmental Needs** | M1, M4 succeeded. |

## 3.5 Subsystems Integration Sequence

### 3.5.1 Integration test case S1

| | |
|---|---|
| **Test case Identifier** | S1 |
| **Test Purpose** | Check if the Application Layer can correctly interact with the Database Layer by inserting, updating, deleting and querying data. |
| **Test Item(s)** | **Application Layer -> Database Layer** |
| **Input Specification** | Create new Entities, update, delete and execute queries on them. |
| **Output Specification** | Check if all the interactions of the Application Layer with the Database Layer are correctly executed. |
| **Environmental Needs** | I1 succeeded. |

### 3.5.2 Integration test case S2

| | |
|---|---|
| **Test case Identifier** | S2 |
| **Test Purpose** | Check if the CarApplication can correctly interact with the Application Layer, using drivers and stubs to simulate the network between them. |
| **Test Item(s)** | **Car Application -> Application Layer** |
| **Input Specification** | Simulation of typical communication messages from the Car Application. |
| **Output Specification** | Check if the methods are called correctly and return meaningful results. |
| **Environmental Needs** | CarApplication driver. Stub for the network classes in the Application Layer. |

### 3.5.3 Integration test case S3

| | |
|---|---|
| **Test case Identifier** | S3 |
| **Test Purpose** | Check if the Application Layer can correctly interact with the Car Application. |
| **Test Item(s)** | **Application Layer -> Car Application** |
| **Input Specification** | Simulation of typical communication messages from the Application Layer. |
| **Output Specification** | Check if the methods are called correctly and return meaningful results. |
| **Environmental Needs** | Application Layer driver. Stub for the network classes in the Car Application. |

### 3.5.4 Integration test case S4 and S5

| **Test case Identifier** | S4, S5 |
|---|---|
| **Test Purpose** | Check if the Mobile Application can correctly interact with the Application layer. |
| **Test Item(s)** | **Mobile Application, Web Layer -> Application Layer** |
| **Input Specification** | Test the functionality of the methods that will be exposed as RESTful serviced once in the production environment, and also all their related methods. |
| **Output Specification** | Check if the future exposed methods return the expected content in JSON files and if the methods related to the exposed ones return correct objects. |
| **Environmental Needs** | Front End driver. S1, S3 succeeded. |

### 3.5.5 Integration test case S6

| **Test case Identifier** | S6 |
|---|---|
| **Test Purpose** | Check if the Mobile Application can correctly interact with the JSF in the Web Layer through HTTPS requests. |
| **Test Item(s)** | **Mobile Application -> Web Layer** |
| **Input Specification** | Typical and atypical HTTPS requests to the Web Layer. |
| **Output Specification** | Check if the Web Layer is dealing correctly with the HTTPS requests, also the malicious ones, and if it provides an adequate degree of performance as stated in the RASD. |
| **Environmental Needs** | S5 succeeded. |

# 4 Tools and Test Equipment Required

## 4.1 Tools

Some software tools will be used in order to automate as much as possible the execution of the test cases written in the previous section and to obtain from them more significant results.

This however doesn't exclude manual testing, which will mostly be performed when the development will be closer to a production release, to test the most critical integrations and in particular the integration of the Mobile Application with Application and Web Layer and the integration of the Car Application with the Application Layer.

### 4.1.1 jUnit

Besides being a tool mainly created to perform unit testing activities (which must be done before integration testing, see Entry Criteria), this tool will be used also to test if the integration between components is producing correct behaviours. These behaviours include:

- Test if the values and object types returned by method calls involved in integration tests are as expected.
- Test if the proper exceptions are raised when invalid parameters are used in method calls and when invalid objects are returned from method calls.

This tool will be used in the Application Layer, Web Layer, Mobile Application, Car Application.

### 4.1.2 Arquillian

This tool will be used to check if the integration between every component and the particular container in which it resides is behaving as expected. Since both the Application Layer and the Web Layer will run on GlassFish Server, we will use the Arquillian GlassFish Container Adapter.

- **Application Layer:**
  - Test whether the dependency injection of JavaBeans is working properly in the EJB Container.
  - Test whether the interaction between the Entity Beans and the EJB Container is working properly.
- **Web Layer:**
  - Test whether the managed beans implementing the WebPageControllers and the JSF are interacting correctly with the Web Container.

#### 4.1.3 Mockito

Mockito is a framework used for integration testing, to create stubs and drivers to support the scaffolding operations. We will mostly use Mockito to create some stubs which are needed to simulate the bidirectional network interaction between the Car Application and the Application Layer, so we don't have to deal with a real network in between while testing.

#### 4.1.4 Apache jMeter

This tool will be used for performance testing, following most of the performance requirements and part of the security requirements stated in the RASD (see Non Functional Requirements).

In particular it will be used with:

- **Application Layer:** jMeter can be used to simulate a heavy load on the Application Server, in particular on the exposed RESTful APIs, to see whether the system is robust enough to fulfill a specified number of requests in parallel.
- **Web Layer:** jMeter can be used to simulate an high load of HTTPS requests to the Web Server.

### 4.2 Test Equipment Required

Some specific testing equipment will be needed to perform part of the integration testing activity, since some tests are useful only if performed in their specific testing environment.

All the devices must be compliant with the characteristics stated in the RASD (see Product Perspective->User Interfaces, and Constraint->Hardware Limitations in section 2).

So we will basically need:

- At least an Android smartphone with a display of 5 inches.
- At least an Android tablet with a display >= 7 inches.
- At least an iOS smartphone with a display >= 5 inches.
- At least an iOS tablet with a display >= 7 inches

The Android tablet can be used both for testing the Car Application and the Mobile Application. No other devices are needed since we can use software emulation using the tools provided inside Android Studio and xCode respectively for Android and iOS.

# 5 Program Stubs and Test Data Required

## 5.1 Program stubs required

The program stubs we are going to need in order to perform the integration testing are essentially used for the Subsystems Integration Testing.

- **Network stub:** It will be used for the following integration tests:
  - Car Application -> Application Layer
  - Application Layer -> Car Application.

  A network stub is a suitable choice for testing the integration between these two subsystems, since it allows the testers to avoid any unwanted intervention of the network and to fully focus on the system logic and functionalities. Of course the network stubs won't substitute a testing with real internet connections, which will happen once the team is closer to a production release.

- **Vehicle Interface Stub:** This stub will be used to simulate the behaviour of the library used to interact with the PowerEnjoy cars' hardware. In this way we don't have to install the Car Application in a PowerEnjoy Car in order to test the integration of the Car Application and the other subsystems.

## 5.2 Drivers required

Besides the multiplicity of drivers which may be needed by using a bottom up approach, we will use in particular:

- **Car Application driver:** for the integration test Car Application -> Application layer.

- **Application Layer driver:** for the integration test Application Layer -> Car Application.

  Both of these first two drivers are used to simulate the method calls of the two subsystem, in order to proceed with the testing phase when they're not necessarily both complete and without having to worry about the network in the middle.

- **Front End Driver:** this driver will be used to simulate the calls of the methods that will need to be exposed as RESTful services, from the Mobile Application and from the Web Layer. Using this driver, we can concentrate initially on the business logic without having to worry about connections, which will be tested in more deep also with Performance Testing.

## 5.3 Test Data

The testing environment must also include the following components:

- **Testing Database:** it will be created inside the DBMS layer and it will be configured as close as possible to the production environment.
- **Stripe** already provides, for the owner of the opened account, an option that allows him to test the integration of the payment service with his code; without having to deal with real payments. We will use it in order to test the integration between the external Stripe Billing Service with the Application Layer, without having to develop additional stubs.

## 6 Software and tools used

- Git (https://github.com/) : for the version controlling of files shared between the team.
- Slack (https://slack.com/): used for group communication.
- GoogleDocs (https://www.google.it/intl/it/docs/about/): to write this document.
- Draw.io (https://www.draw.io/): to create the graphs.
- Lyx (http://www.lyx.org/): to format this document.

## 7 Effort spent

Total hours of work for the ITPD creation:

- Stefano Brandoli: 11 hours
- Silvia Calcaterra: 7 hours
- Samuele Conti: 7 hours