POLITECNICO
MILANO 1863

Software Engineering 2 project: *PowerEnJoy*

A.A. 2016/2017 - Professor E. di Nitto

Integration Test Plan Document

Version **1.0** –15/01/2018

Pietro Avolio Mat 878640
Guido Borrelli Mat 874451

Contents

1 Introduction **4**
- 1.1 Purpose and scope . . . 4
- 1.2 Definitions, acronyms and abbreviations . . . 4
- 1.3 Reference Documents: . . . 4
- 1.4 Document Structure . . . 4

2 Integration Strategy **6**
- 2.1 Entry criteria . . . 6
- 2.2 Elements to be integrated . . . 6
- 2.3 Integration testing strategy . . . 7
- 2.4 Integration sequence . . . 7
  - 2.4.1 Sub-components of the core component integration sequence 7
  - 2.4.2 Components integration sequence . . . 9

3 Test Description **11**
- 3.1 Test cases specification . . . 11
  - 3.1.1 Integration test case CI_1 . . . 11
  - 3.1.2 Integration test case CI_2 . . . 11
  - 3.1.3 Integration test case SCI_1 . . . 12
  - 3.1.4 Integration test case SCI_2 . . . 13
  - 3.1.5 Integration test case SCI_3 . . . 13
  - 3.1.6 Integration test case SCI_4 . . . 14
  - 3.1.7 Integration test case SCI_5 . . . 14
  - 3.1.8 Integration test case SCI_6 . . . 14
  - 3.1.9 Integration test case SCI_7 . . . 15
  - 3.1.10 Integration test case SCI_8 . . . 15
  - 3.1.11 Integration test case SCI_9 . . . 16
  - 3.1.12 Integration test case SCI_10 . . . 16
  - 3.1.13 Integration test case SCI_11 . . . 16
  - 3.1.14 Integration test case SCI_12 . . . 17
  - 3.1.15 Integration test case SCI_13 . . . 17
  - 3.1.16 Integration test case SCI_14 . . . 17
  - 3.1.17 Integration test case SCI_15 . . . 18
  - 3.1.18 Integration test case SCI_16 . . . 19
  - 3.1.19 Integration test case E_SCI_1 . . . 19
  - 3.1.20 Integration test case E_SCI_2 . . . 20
  - 3.1.21 Integration test case E_SCI_3 . . . 20
  - 3.1.22 Integration test cases with the Log Manager . . . 20
  - 3.1.23 Integration test cases with the RESTful Api Manager . . . 21
  - 3.1.24 Integration test cases CI_3, CI_4, CI_5 . . . 21
- 3.2 Test procedures . . . 21
  - 3.2.1 Integration test procedure TP1 . . . 21
  - 3.2.2 Integration test procedure TP2 . . . 21
  - 3.2.3 Integration test procedure TP3 . . . 22
  - 3.2.4 Integration test procedure TP4 . . . 22

4 Tools and Test Equipment Required **23**

5 Program Stubs and Test Data Required **24**

6 Appendix **24**
6.1 Tools used 24
6.2 Effort spent 24

1 Introduction

1.1 Purpose and scope

The purpose of this integration test plan document is to provide information about the integration test activities to be performed for the PowerEnJoy project. This includes how to accomplish the integration tests, which approach to follow and which tools to use.

The aim of the integration test activities is to verify that the different components of the system interoperate coherently in order to fulfill the functional and non-functional requirements of the system.
Since some components are made up of several other sub-components, the integration tests must be performed inside those components too.

1.2 Definitions, acronyms and abbreviations

In order to avoid ambiguity and possible misunderstandings, this table is an integration to the definitions, the acronyms and the abbreviations defined in the RASD and DD documents.

Component	A part of the system implementing high-level functionalities
Sub-Component	A part of a component
LOF	Lines Of Code

1.3 Reference Documents:

- Integration Test Plan Documents of the previous years published on BeeP.
- spinGRID integration test plan on BeeP.
- Testing slides on BeeP .
- Tips and Trick for Supporting Architecture Description with UML on BeeP.
- Integration Testing page on Wikipedia (visited 08/01/2017).

1.4 Document Structure

The document is structured in the following chapters:

Introduction This chapter provides preliminary informations about the document.

Integratio Strategy This chapter provides detailed information about the entry criteria to be met before starting the integration test, the elements to be integrated and an outline of the main strategies to follow.

Individual Steps and Test Description This chapter provides details for each test procedure to be performed.

Tools and Test Equipment Required This chapter provides a list of the tools to be used for the integration test activities.

Program Stubs and Test Data Required This chapter summarizes the stubs, the drivers and the test data required to perfor the integration test activities.

2 Integration Strategy

2.1 Entry criteria

In this section we describe the prerequisites needed before the integration tests can be started.

Methods and classes should be locally unit tested and achieve at least an 80% LOF coverage in order to discover major issues in the effective implementation of the algorithms and in the usage of the data structures. Unit tests should be performed during the development.

The project should also have passed through a phase of code inspection in order to ensure maintenability, the respect of the conventions, usage of the best known practices and the avoidance of the bad ones. The code inspection should be automated via automated tools as much as possibile in order to be able to spend manual inspection on the most complex parts.

The following documents describing functional and non functional requirements, behaviour and purpose of the system must be delivered:

- Requirements Analysis and Specication Document (RASD).
- Design Document (DD).
- Integration Test Plan Document (this document).

Finally, an up-to-date and complete documentation of the code, using a common paradigm such as JavaDoc, could be a solid base and a good reference for the integration tests development.

2.2 Elements to be integrated

As stated in the DD document, the system is splitted on four logical layers, each of those containing one or more components:

Client Layer This logical layer contains the PowerEnJoy Mobile Application, the PowerEnJoy Control Room and the vehicles.

Web Server This logical layer contains the web server required to host the PowerEnJoy Control Room.

Business Tier This logical layer contains the PowerEnJoy Core component.

Data Tier This logical layer contains the RDBS and the NoSqlDBMS.

These logical layers and the components they contain communicate using the interfaces described in the DD, which are object of the integration tests.

The PowerEnJoy Core component, which contains all the application logic, is made up of several sub-components grouped in various containers as described in the DD document; it also communicates with some external gateways. The way these sub-components integrate one with the other and the way they communicate with the external gateways is the main part of the integration test activities to be performed.

2.3 Integration testing strategy

In order to prevent all the possibile issues coming from a typical big bang approach, in which all the integration tests are performed at the end of the development process, a slowly incremental approach is the one chosen for the integration tests of the system. The incremental approach facilitates the localization and the recognition of the flaws in the code.

Up to limitate the number of stubs and mockups to be realized, the integration tests should follow a bottom-up flow, starting from the smallest - and often more critical - sub-components of the PowerEnJoy Core to the biggest ones.
The last tests to be performed are the ones concerning the intercommunication between components, which are mainly based on RESTful APIs and HTTPs protocol.

2.4 Integration sequence

In this section is described the sequence in which both the high-level components and the sub-components will be integrated.
The integration tests will proceed following a bottom-up approach: the first integration test activities to be performed are between the PowerEnJoy Core and the database layer; then following integration test activities are between the internal sub-components of the PowerEnJoy core; last activities are about the communication between the clients and the core, thus they are about the RESTful APIs.

2.4.1 Sub-components of the core component integration sequence

The sub-components integration tests sequence is described in table 2 and figure 1. As mentioned in section 2.3, the sequence moves from the most independent to the less independent component through the logical flow of physical events.
As stated in the DD, almost all the components iterate with the Log Manager and the RestAPI Manager: for this reason the integration tests are omitted from both the table and the figure for the sake of simplicity.
The orange arrows correspond to integration tests to be performed towards external components, for this reason an explicit order is not provided and the integration tests can be performed as soon as the component is ready.

#	**SubComponent**	**Integrates with**
SCI_1	Rentals Handler	Vehicle Manager
SCI_2	Rentals Handler	Pricing Policies Manager
SCI_3	Rentals Handler	Payment Handler
SCI_4	Rentals Handler	SMS Sender
SCI_5	Rentals Handler	Push Notifications sender
SCI_6	Reservations Handler	Vehicle Manager
SCI_7	Reservations Handler	Pricing Policies Manager
SCI_8	Reservations Handler	Payment Handler
SCI_9	User Manager	Payment Handler
SCI_10	User Manager	Email sender
SCI_11	User Manager	Vehicle Manager
SCI_12	CronTasks Manager	Email sender
SCI_13	CronTasks Manager	SMS sender
SCI_14	CronTasks Manager	Push Notifications Sender
SCI_15	Operators Manager	Support Requests Handler
SCI_16	Operators Manager	Vehicle Manager
E_SCI_1	Payment Handler	Payment Gateway
E_SCI_2	SMS Sender	SMS Gateway
E_SCI_3	Push Notifications Sender	Push Notifications Gateway

Table 2: Integration of the sub-components of the core component

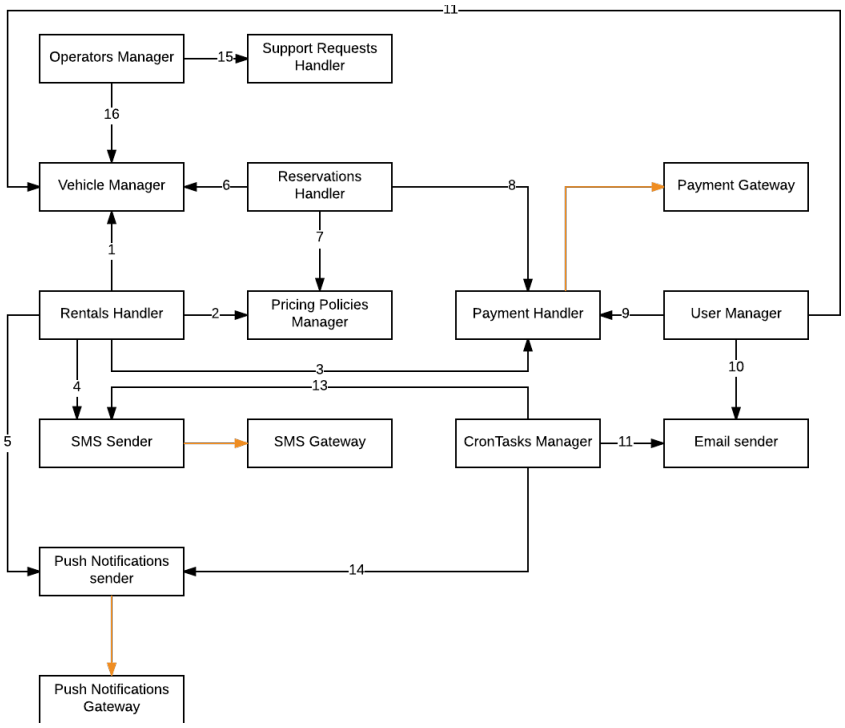

Figure 1: Sequence diagram of the integration of the sub-components of the core component

2.4.2 Components integration sequence

The integration tests sequence of the main components of the system is described in table 3 and figure 2.

The order of the integration tests between the PowerEnJoy Core with the two DBMS as well as the order of the integration tests between the PowerEnJoy Mobile Application and the PowerEnJoy Control Room with the PowerEnJoy Core are arbitrary and they could be reversed for any special needs rising during the development.

#	Component	Integrates with
CI_1	PowerEnJoy Core	Relational DBMS
CI_2	PowerEnJoy Core	NoSQL DBMS
CI_3	PowerEnJoy Vehicle	PowerEnJoy Core
CI_4	PowerEnJoy Mobile App	PowerEnJoy Core
CI_5	PowerEnJoy Control Room	PowerEnJoy Core

Table 3: Integration of the main components of the system

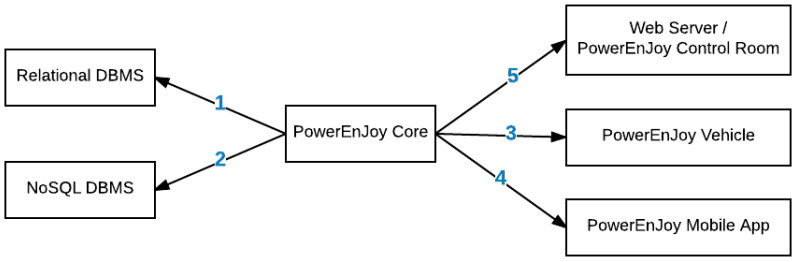

Figure 2: Sequence diagram of the integration of the main components

3 Test Description

3.1 Test cases specification

Here we will schematize and detail every integration step of the integration sequence identified in section 2.4. Each test case is directly mapped to the content of tables 2 and 3.
Test cases are listed in the order in which they will be executed.

3.1.1 Integration test case CI_1

Identifier	CI_1_T1
Items	PowerEnJoy Core -> Relational DBMS
Input specification	Relational DBMS address and port; database user and password
Output specification	-
Description	The test checks the connection between the core and the relational DBMS
Environmental needs	-
Testing method	Arquillian

Identifier	CI_1_T2
Items	PowerEnJoy Core -> Relational DBMS
Input specification	Execution of different queries on the relational data
Output specification	Responses should contain the correct results for each query
Description	The test checks: the correctness of the mapping between the JPA and the datasource, if the privileges granted to the given user are sufficient
Environmental needs	Full mapping of the JPA with the datasource, relational DBMS connection, test relational data, driver to run the JPA
Testing method	Arquillian

3.1.2 Integration test case CI_2

Identifier	CI_2_T1
Items	PowerEnJoy Core -> NoSQL DBMS
Input specification	NoSQL DBMS address and port; database user and port
Output specification	-
Description	The test checks the connection between the core and the relational DBMS
Environmental needs	-
Testing method	Arquillian

Identifier	CI_2_T2
Items	PowerEnJoy Core -> NoSQL DBMS
Input specification	Execution of different queries on the datasource
Output specification	Responses should contain the correct results for each query
Description	The test checks: the correctness of the mapping between the JPA and the datasource; the correctness of the interface between the JPA and a NoSQL DBMS; if the privileges granted to the given user are sufficient
Environmental needs	Full mapping of the JPA with the datasource, NoSQL DBMS connection, test data, driver to run the JPA
Testing method	Arquillian

3.1.3 Integration test case SCI_1

Identifier	SCI_1_T1
Items	Rentals Handler -> Vehicle Manager
Input specification	Method call to get a specific vehicle status
Output specification	Response should contain the correct vehicle status
Description	The test checks the correctness of method interaction between those two sub-components
Environmental needs	Full development of the two components, database communication, test data about the vehicle
Testing method	JUnit

Identifier	SCI_1_T2
Items	Rentals Handler -> Vehicle Manager
Input specification	Method call to change a specific vehicle status
Output specification	Response should contain the correct operation result and the status of the vehicle should be correctly changed in the system and in the database
Description	The test checks the correctness of method interactions between those two sub-components
Environmental needs	Full development of the two components, database communication, test data about the vehicle
Testing method	JUnit

Identifier	SCI_1_T3
Items	Rentals Handler -> Vehicle Manager
Input specification	Method call to send a specific command to a given vehicle
Output specification	Response should contain the correct operation result
Description	The test checks the correctness of method interactions between those two sub-components
Environmental needs	Full development of the two components, database communication, test data about the vehicle, stub for the vehicle as a client
Testing method	JUnit + Mockito

3.1.4 Integration test case SCI_2

Identifier	SCI_2_T1
Items	Rentals Handler -> Pricing Policies Manager
Input specification	Method call to calculate a rental charge with current rates
Output specification	Response should contain the correct price for the provided rental
Description	The test checks the correctness of method interactions between those two sub-components
Environmental needs	Full development of the two components, test data about the rental
Testing method	JUnit

3.1.5 Integration test case SCI_3

Identifier	SCI_3_T1
Items	Rentals Handler -> Payment Handler
Input specification	Method call to request a payment of a given amount
Output specification	Response should contain the correct operation result
Description	The test checks the correctness of method interactions between those two sub-components
Environmental needs	Full development of the two components, stub for the payment gateway
Testing method	JUnit + Mockito

3.1.6 Integration test case SCI_4

Identifier	SCI_4_T1
Items	Rentals Handler -> SMS Sender
Input specification	Method call to send an SMS to a given phone number
Output specification	Response should contain the correct operation result
Description	The test checks the correctness of method interactions between those two sub-components
Environmental needs	Full development of the two components, stub of the SMS gateway
Testing method	JUnit + Mockito

3.1.7 Integration test case SCI_5

Identifier	SCI_5_T1
Items	Rentals Handler -> Push Notifications sender
Input specification	Method call to send a Push Notification to a given user
Output specification	Response should contain the correct operation result
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two components, test data about the user, stub for the Push Notifications gateway
Testing method	JUnit + Mockito

3.1.8 Integration test case SCI_6

Identifier	SCI_6_T1
Items	Reservations Handler -> Vehicle Manager
Input specification	Method call to get a specific vehicle status
Output specification	Response should contain the correct vehicle status
Description	The test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two components, database communication, test data about the vehicle
Testing method	JUnit

Identifier	SCI_6_T2
Items	Reservations Handler -> Vehicle Manager
Input specification	Method call to change a specific vehicle status
Output specification	Response should contain the correct operation result and the status of the vehicle should be correctly changed in the system
Description	The test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two components, database communication, test data about the vehicle
Testing method	JUnit

3.1.9 Integration test case SCI_7

Identifier	SCI_7_T1
Items	Reservations Handler -> Pricing Policies Manager
Input specification	Method call to calculate the price of an expired reservation
Output specification	Response should contain the correct price of an expired reservation
Description	The test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two components
Testing method	JUnit

3.1.10 Integration test case SCI_8

Identifier	SCI_8_T1
Items	Reservations Handler -> Payment Handler
Input specification	Method call to request a payment of a given amount
Output specification	Response should contain the correct operation result
Description	The test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two components
Testing method	JUnit

3.1.11 Integration test case SCI_9

Identifier	SCI_9_T1
Items	User Manager -> Payment Handler
Input specification	Method call to request the validation of specific payment informations
Output specification	Response should contain the correct operation result
Description	The test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two components, test data about the payment information, stub of the payment gateway
Testing method	JUnit + Mockito

3.1.12 Integration test case SCI_10

Identifier	SCI_10_T1
Items	User Manager -> Email Sender
Input specification	Method call to send an email to a specific email address
Output specification	Email should be correctly sent to the address
Description	The test checks the correctness of method interactions between those two sub-components
Environmental needs	Full development of the two sub-components, test data about the email address, email server
Testing method	JUnit

3.1.13 Integration test case SCI_11

Identifier	SCI_11_T1
Items	User Manager -> Vehicle Manager
Input specification	Method call to retrieve all the vehicle given the center and the radius
Output specification	Response should contain a list of all the available vehicles in the area
Description	The test checks the correctness of method interfactions between those two sub-components
Environmental needs	Full development of the two sub-components, database connection, test data about the vehicles
Testing method	JUnit

3.1.14 Integration test case SCI_12

Identifier	SCI_12_T1
Items	CronTasks Manager -> Email Sender
Input specification	Method call to send an email to a specific email address
Output specification	Email should be correctly sent to the address
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Test data about the email address, email server
Testing method	Arquillan

3.1.15 Integration test case SCI_13

Identifier	SCI_13_T1
Items	CronTasks Manager -> SMS Sender
Input specification	Method call to send an SMS to a given phone number
Output specification	Response should contain the correct operation result
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Stub of the SMS gateway
Testing method	JUnit + Mockito

3.1.16 Integration test case SCI_14

Identifier	SCI_14_T1
Items	CronTasks Manager -> Push Notification Sender
Input specification	Method call to send a Push Notification to a given user
Output specification	Response should contain the correct operation result
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Test data about the user, stub for the Push Notifications gateway
Testing method	JUnit + Mockito

3.1.17 Integration test case SCI_15

Identifier	SCI_15_T1
Items	Operators Manager -> Support Request Handler
Input specification	Method call to get all the support requests given a set of filters (could be empty)
Output specification	Response should contain all the support requests that match the given filters
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Database connection, test data about the support requests
Testing method	Arquillan

Identifier	SCI_15_T2
Items	Operators Manager -> Support Request Handler
Input specification	Method call to send a reply to the support request
Output specification	The reply should be correctly delivered
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Database connection, test data about the support request, stub for the user's client
Testing method	JUnit + Mockito

Identifier	SCI_15_T3
Items	Operators Manager -> Support Request Handler
Input specification	Method call to change the status of a support request
Output specification	The status of the support request should be correctly changed into the given one
Description	Test checks the correctness of method interactions between those two sub components; the change should be correctly notified to the user
Environmental needs	Database connection, test data about the support request, stub for the user's client
Testing method	JUnit + Mockito

3.1.18 Integration test case SCI_16

Identifier	SCI_16_T1
Items	Operators Manager -> Vehicle Manager
Input specification	Method call to get the status of a given vehicle
Output specification	Response should contain the correct status of the given vehicle
Description	Test check correctness of method interactions between those two sub components
Environmental needs	Full development of the two components, database connection, test data about the vehicles
Testing method	JUnit

3.1.19 Integration test case E_SCI_1

Identifier	E_SCI_1_T1
Items	Payment Handler -> Payment Gateway
Input specification	Payment request over a credit card with enough credit available
Output specification	The response should be positive and the payment should be correctly handled
Description	The test checks the behaviour of the system when a payment is succesfull
Environmental needs	Full development of the payment handler, payment gateway connection, development tools and development account for the payment gateway
Testing method	JUnit

Identifier	E_SCI_1_T2
Items	Payment Handler -> Payment Gateway
Input specification	Payment request over a credit card with not enough credit available
Output specification	The response should be negative and a notification should be issued towards the operators and towards the user
Description	The test checks the system behaviour when a payment is not successfully
Environmental needs	Full development of the payment handler, payment gateway connection, development tools and development account for the payment gateway, test data about the user
Testing method	JUnit

3.1.20 Integration test case E_SCI_2

Identifier	E_SCI_2_T1
Items	SMS Sender -> SMS Gateway
Input specification	SMS send request to a specific number with special characters (è, à, ù, ì)
Output specification	The SMS should be correctly sent
Description	The test checks both the connection with the SMS gateway and the possibile issues with special characters
Environmental needs	Full development of the SMS sender, SMS gateway connection
Testing method	JUnit

3.1.21 Integration test case E_SCI_3

Identifier	E_SCI_3_T1
Items	Push Notification Sender -> Push Notification Gateway
Input specification	Push Notification send request to a specific user's device
Output specification	The notification should be correctly issued
Description	The test checks the connection with the push notifications gateway
Environmental needs	Full development of the Push Notifications sender, Push Notifications Gateway connection
Testing method	JUnit

3.1.22 Integration test cases with the Log Manager

As stated in other parts of this document and in the DD, almost all the sub-components of the Core component interact with the LogManager.

This test should be perfomed for each component and it should be executed as soon as the component is ready.

Identifier	LM_TX
Items	All the sub-components -> Log Manager
Input specification	Execution of different log requests with different messages and log level (e.g. warn, err, debug)
Output specification	Message should be correctly added
Description	Test checks the correctness of method interactions between those two sub components
Environmental needs	Full development of the two sub-components
Testing method	JUnit

3.1.23 Integration test cases with the RESTful Api Manager

As stated in other parts of this document and in the DD, many sub-components of the Core component interact with the RESTful Api Manager.

Identifier	RAM_TX
Items	All the sub-components -> RESTful Api Manager
Input specification	All the parameters needed to build the response in JSON format
Output specification	The response is encoded in JSON format and includes all the inputs
Description	The test checks the correctness of the responses built using the RestFul Api Manager
Environmental needs	Full development of the two sub-components
Testing method	JUnit

3.1.24 Integration test cases CI_3, CI_4, CI_5

Identifier	CI_3, CI_4, CI_5
Items	PowerEnJoy Mobile App, PowerEnJoy Control Room, PowerEnJoy Vehicle -> PowerEnJoy Core
Input specification	Different RESTful APIs calls
Output specification	Responses should be appropiate to the given request, security measures should be satisfied
Description	The test checks the correctness of the communication between the clients and the PowerEnJoy Core through RESTful APIs
Environmental needs	Full development of the two components
Testing method	Arquillan

3.2 Test procedures

In this section we list some of the main procedures that the systems is going to offer and that should be checked in order to verify interoperability between the components and to check that the system achieves the goals specified in the RASD.

3.2.1 Integration test procedure TP1

Procedure identifier	TP1
Purpose	Test the procedure to create a new reservation
Procedure steps	SCI_6

3.2.2 Integration test procedure TP2

Procedure identifier	TP2
Purpose	Test the procedure to start a rental
Procedure steps	SCI_1 after SCI_6

3.2.3 Integration test procedure TP3

Procedure identifier	TP3
Purpose	Test the procedure to forward a payment request after a rental is terminated
Procedure steps	SCI_3 after SCI_2 after SCI_1

3.2.4 Integration test procedure TP4

Procedure identifier	TP14
Purpose	Test the payment transaction through the payment gateway
Procedure steps	E_SCI_1 after SCI_3

4 Tools and Test Equipment Required

The following softwares are identified to assist the integration activities of the system:

- Mockito: it is an open-source test framework useful to generate mock objects, stubs and drivers.
- Arquillian: it is a test framework which can also manage the test of the containers, their integration with JavaBeans and dependency injections too.
- JUnit: it is the most used framework for unit testing in Java, expecially for unit tests. Mainly used in this project to test the integration of the sub-components.

The test equipment regarding real tests used should be very heterogeneous:

- Various type of Android smartphones/tablets with different screen resolutions and hardware features.
- Different PCs to run the PowerEnJoy Control Room on.
- At least one fully-equipped vehicle.
- Safe area should be tested for each city in which the service will be activated.

A good practice would be to enroll some testers and perform a beta-testig phase in order to check that all the goals and the functionalities are correctly provided and in order to have a customer experience feedback.

5 Program Stubs and Test Data Required

Using an incremental approach instead of a 'big bang' approach, different stubs, drivers and test data are required during the different integration test activities. Stubs, drivers and test data are indicated for each test case in section 3.1.22.

6 Appendix

6.1 Tools used

- LyX as LaTeX editor.
- Lucidchart for diagrams.
- Github as version controller.

6.2 Effort spent

Pietro Avolio	11
Guido Borrelli	12