Politecnico di Milano, A.A. 2016/2017

Software Engineering 2: PowerEnJoy
Integration Test Plan Document

Binosi Lorenzo 876022

January 15, 2017

Contents

1 Introduction **3**
- 1.1 Revision History . . . 3
- 1.2 Purpose . . . 3
- 1.3 Scope . . . 3
- 1.4 Definitions, Acronyms, Abbreviations . . . 3
- 1.5 Reference Documents . . . 4

2 Integration Strategy **5**
- 2.1 Entry Criteria . . . 5
- 2.2 Elements to be Integrated . . . 5
- 2.3 Integration Testing Strategy . . . 7
- 2.4 Sequence of Component/Function Integration . . . 7
  - 2.4.1 Software Integration Sequence . . . 7
  - 2.4.2 Subsystem Integration Sequence . . . 9

3 Individual Steps and Test Description **10**
- 3.1 Integration test cases I01, I02, I03, I04, I05, I06, I07, I08, I09 . 10
- 3.2 Integration test case I10 . . . 11
- 3.3 Integration test cases I11, I12, I13, I14, I15, I16 . . . 12
- 3.4 Integration test case I17 . . . 13
- 3.5 Integration test cases SI1, I18 . . . 13
- 3.6 Integration test case SI1 . . . 14
- 3.7 Integration test case I19 . . . 14
- 3.8 Integration test case I20 . . . 15
- 3.9 Integration test cases SI2, I21 . . . 15
- 3.10 Integration test case SI2 . . . 16
- 3.11 Integration test case I22 . . . 16
- 3.12 Integration test case I23 . . . 17
- 3.13 Integration test case I24 . . . 17
- 3.14 Integration test case I25 . . . 18
- 3.15 Integration test cases SI3, I26 . . . 18
- 3.16 Integration test cases SI3, I27 . . . 19
- 3.17 Integration test case I28 . . . 19
- 3.18 Integration test cases SI4, I29 . . . 20

4 Tools and Test Equipment Required **21**

5 Program Stubs and Test Data Required **22**

A Appendix **23**
A.1 Hours of work 23

1 Introduction

1.1 Revision History

Version of this document: 1.0
Last update: 15/01/2017

1.2 Purpose

This document is the Integration Test Plan Document (ITPD) for the PowerEnJoy software. It describes the testing approach and overall framework that will drive the testing of the software.

The document is written for project managers, developers, testers and quality assurance.

1.3 Scope

PowerEnJoy is a digital management system for a car-sharing service. Its software need to be tested, in a clearly defined and well thought way, in order to avoid any failure or undesired behavior of the system.

1.4 Definitions, Acronyms, Abbreviations

ITPD: Integration Test Plan Document.

DD: Design Document.

RASD: Requirements Analysis and Specification Document.

System: the whole software system to be developed, comprehensive of all its parts.

REST: REpresentational State Transfer. It's an architectural style and an approach to stateless communications, used in the development of client-server systems.

GUI: Graphical User Interface.

API: Application Programming Interface.

EJB: Enterprise Java Bean.

JPA: Java Persistence API.

JSP: JavaServer Pages.

HTTP: HyperText Transfer Protocol.

HTTPS: HyperText Transfer Protocol Secure.

1.5 Reference Documents

This document refers to the following documents:

- Software Engineering 2 project [1].
- ITPD assignement [2].
- RASD of the PowerEnJoy project [3].
- DD of the PowerEnJoy project [4].

2 Integration Strategy

2.1 Entry Criteria

In order to perform an efficient and reliable integration testing of the software architecture, we must define the prerequisites to be achieved before it can be started.

First of all, both the RASD and the DD must be delivered, and tested according to the following criteria:

completeness: whether all components are present and described completely.

consistency: whether all components do not contradict each other.

feasibility: whether the project is feasible considering the given constraints (deadlines, budget and so on).

testability: whether or not a system fulfills its requirements.

Afterwards, the integration testing process must be developed following the defined strategy of Section 2.3.

It's suggested to start the integration process once there is enought code that could be integrated, taking into account the constraints carried by the strategy. For instance, if the development of the mobile application is not started, we could still start the integration process of the web logic. Obviously, it requires the integration of the business logic to be completed and the entire web logic to be written.

2.2 Elements to be Integrated

The elements to be integrated are described in the DD and now recalled, highlighting the subsystem that belong to them, in Table 2.1.

Element	Description	Subsystem
DBMS	The database of the system	Database
SafeArea	The EJB of the safe areas	Business
Car	The EJB of the cars	Business
Rent	The EJB of the rents	Business
RentalEvent	The EJB of the rental events	Business
CarCredential	The EJB of the cars' credential	Business
Driver	The EJB of the drivers	Business
CreditCard	The EJB of the credit cards	Business
Park	The EJB of the parked cars	Business
CarAssistance	The EJB of the cars' issues reports	Business
DriverManager	The session EBJ for the user's operations	Business
DriverRentalManager	The session EBJ for the rental operations	Business
SearchCarManager	The session EBJ for the searching operations	Business
LogBookManager	The session EBJ for the logbook operations	Business
CarHeartbeat	The session EBJ for the car heartbeat management	Business
RentalFee	The singleton EBJ for the rental fee	Business
CarRentalManager	The session EBJ for the car rental information	Business
Servlets	All the servlets that operate using the session EJBs	Business
ServletContainer	The servlet container that invoke the right servlet once called	Business
AndroidUIManager	The Android UI manager	AndroidMobile
iOSUIManager	The iOs UI manager	iOSMobile
AndroidGPSManager	The Android GPS manager	AndroidMobile
iOSGPSManager	The iOS GPS manager	iOSMobile
AndroidController	The Android controller	AndroidMobile
iOSController	The iOS controller	iOSMobile
WebContoller	The web controller that communicates with the application server	Web
JSP	The JSP of the web application	Web
WebContainer	The web container that invoke the right JSP once called	Web

CarApplication	The car application that notifies the system about the car/renta status	Car
CarGUI	The car application that supports several utilities (Radio, GPS Navigator, ...)	Car

Table 2.1: Elements of the PowerEnJoy system to be integrated.

2.3 Integration Testing Strategy

The strategy adopted for the integration testing is the **bottom-up** approach, for the software integration, and the **critical-module-first** approach, for the subsystem integration. The choice of these approaches it's due to the fact that the integration process of a subsystem is automatically completed once the software of the same subsystem is integrated.

Moreover, the performance of the system could be evaluated at each integration step using **JMeter**. In fact if the performance drop down it's probably due to the last integrated element/subsystem.

2.4 Sequence of Component/Function Integration

2.4.1 Software Integration Sequence

The software integration sequence is summarized in Table 2.2.

Choosing the bottom-up approach the elements are tested from the most indipendent to the less one. They require the implementation of some drivers which are developed using **Mockito**, aimed to tests the iteraction between elements of the same subsystem, and **Arquillian**, aimed to tests the iteraction beetwen elements of different subsystem. Both frameworks require **JUnit** for some essential testing functionalities (for instance the *assertEquals* function).

N.	Subsystem	Component	Integrates with
I1	Database, Business	(JEB) Car	DBMS
I2	Database, Business	(JEB) Rent	DBMS
I3	Database, Business	(JEB) Driver	DBMS
I4	Database, Business	(JEB) Park	DBMS
I5	Database, Business	(JEB) SafeArea	DBMS
I6	Database, Business	(JEB) RentalEvent	DBMS
I7	Database, Business	(JEB) CreditCard	DBMS
I8	Database, Business	(JEB) CarAssistance	DBMS
I9	Database, Business	(JEB) CarCredential	DBMS
I10	Business	(SB) RentalFee	Settings.xml
I11	Business	(SB) DriverManager	Driver CreditCard
I12	Business	(SB) DriverRentalManager	Driver Car Rent CarCredential Park
I13	Business	(SB) SearchCarManager	Driver Car
I14	Business	(SB) LogBookManager	Driver Rent
I15	Business	(SB) CarHeatbeat	CarCredential Rent Car CreditCard Park CarAssistance RentalEvent RentalFee
I16	Business	(SB) CarRentalManager	CarCredential Driver SafeArea Park RentalFee RentalEvent
I17	Business	Servlets	DriverManager DriverRentalManager SearchCarManager LogBookManager CarHeatbeat CarRentalManager

I18	Business	ServletContainer	Servlets
I19	Web	WebController	(Business) ServletContainer
I20	Web	JSP	WebController
I21	Web	WebContainer	JSP
I22	Mobile	iOSController	(Business) ServletContainer
I23	Mobile	AndroidController	(Business) ServletContainer
I24	Mobile	iOSGPSManager	iOSController
I25	Mobile	AndroidGPSManager	AndroidController
I26	Mobile	iOSUIManager	iOSController
I27	Mobile	AndroidUIManager	AndroidController
I28	Car	CarApplication	(Business) ServletContainer
I29	Car	CarGUI	(Business) ServletContainer

Table 2.2: Integration of the PowerEnJoy system's elements.

2.4.2 Subsystem Integration Sequence

As already said, the strategy adopted for the subsystem integration is the critical-module-first approach. For the chosen software architecture, this approach allow us to a better and easier integration. The database requires simple drivers in order to be integrate, instead, the business logic doesn't need them at all. In fact it's possible to test the correct integration of the business subsytem just making API calls in an automated fashion.

The subsystem integration sequence is summerized in Table 2.3.

N.	**Subsystem**	**Integrates with**
SI1	Business	Database
SI2	Web	Business
SI3	Mobile	Business
SI4	Car	Business

Table 2.3: Integration of the PowerEnJoy's subsystems.

3 Individual Steps and Test Description

This section provides a detailed description of the tests to be performed on each elements that have to be integrated.

Test cases whose code starts with **SI** are integration tests between subsystems; test cases whose code starts with **I** are integration tests between components.

3.1 Integration test cases I01, I02, I03, I04, I05, I06, I07, I08, I09

Test Case Identifier	I01T1
Test Item(s)	Car → DBMS
Input Specification	Typical queries on the table Car.
Test Case Identifier	I02T1
Test Item(s)	Rent → DBMS
Input Specification	Typical queries on the table Rent.
Test Case Identifier	I03T1
Test Item(s)	Driver → DBMS
Input Specification	Typical queries on the table Driver.
Test Case Identifier	I04T1
Test Item(s)	Park → DBMS
Input Specification	Typical queries on the table Park.
Test Case Identifier	I05T1
Test Item(s)	SafeArea → DBMS
Input Specification	Typical queries on the table SafeArea.
Test Case Identifier	I06T1
Test Item(s)	RentalEvent → DBMS
Input Specification	Typical queries on the table RentalEvent.
Test Case Identifier	I07T1
Test Item(s)	CreditCard → DBMS
Input Specification	Typical queries on the table CreditCard.

Test Case Identifier	I08T1
Test Item(s)	CarAssistance → DBMS
Input Specification	Typical queries on the table CarAssistance.
Test Case Identifier	I09T1
Test Item(s)	CarCredential → DBMS
Input Specification	Typical queries on the table CarCredential.
Output Specification	The queries return the correct results.
Environmental Needs	GlassFish server, Test Database, drivers for the Java Entity Beans.
Test Description	The purpose of these tests is to check the communication between the EJB and the DBMS. Moreover, the tests must check if the queries are executed correctly.
Testing Method	Automated with JUnit and Arquillian.

3.2 Integration test case I10

Test Case Identifier	I10T1
Test Item(s)	RentalFee → Settings.xml
Input Specification	Methods call from the RentalFee singleton.
Output Specification	The returned value of the method calls contains the fee choosen by the system .
Environmental Needs	GlassFish server, drivers for the singleton.
Test Description	The purpose of these tests is to check if the returned value is correct.
Testing Method	Automated with JUnit.

3.3 Integration test cases I11, I12, I13, I14, I15, I16

Test Case Identifier	I11T1
Test Item(s)	DriverManager → Driver, CreditCard
Input Specification	Methods call from the DriverManager session bean.
Test Case Identifier	I12T1
Test Item(s)	DriverRentalManager → Driver, Car, Rent, CarCredential, Park
Input Specification	Methods call from the DriverRentalManager session bean.
Test Case Identifier	I13T1
Test Item(s)	SearchCarManager → Driver, Car
Input Specification	Methods call from the SearchCarManager session bean.
Test Case Identifier	I14T1
Test Item(s)	LogBookManager → Driver, Rent
Input Specification	Methods call from the LogBookManager session bean.
Test Case Identifier	I15T1
Test Item(s)	CarHeartbeat → CarCredential, Rent, Car, CreditCard, Park, CarAssistance, RentalEvent, RentalFee
Input Specification	Methods call from the CarHeartbeat session bean.
Test Case Identifier	I16T1
Test Item(s)	CarRentalManager → CarCredential, Driver, SafeArea, Park, RentalFee, RentalEvent
Input Specification	Methods call from the SearchCarManager session bean.
Output Specification	Methods are fully executed providing the right functionalities to the system.
Environmental Needs	GlassFish server, drivers for the session beans, stubs for the car application.
Test Description	The purpose of these tests is to ensure that the methods of the session beans provide the right functionalities to the system.
Testing Method	Automated with JUnit and Arquillian.

3.4 Integration test case I17

Test Case Identifier	I17T1
Test Item(s)	Servlets → DriverManager, DriverRentalManager, SearchCarManager, LogBookManager, CarHeartbeat, CarRentalManager
Input Specification	Servlets' methods call.
Output Specification	Methods are fully executed providing the right functionalities to the system.
Environmental Needs	GlassFish server, drivers for the servlets.
Test Description	The purpose of these tests is to ensure that the methods of the servlets provide the right functionalities to the system.
Testing Method	Automated with JUnit and Mockito.

3.5 Integration test cases SI1, I18

Test Case Identifier	SI1T1, I18T1
Test Item(s)	ServletContainer → Servlet
Input Specification	HTTPS requests to the application server.
Output Specification	The ServletContainer invokes the correct servlets.
Environmental Needs	GlassFish server, plug-in for custom HTTPs requests.
Test Description	The purpose of these tests is to check if the correct servlet has been called.
Testing Method	Automated with JUnit and Arquillian.

3.6 Integration test case SI1

Test Case Identifier	SI1T2
Test Item(s)	Business → Database
Input Specification	Multiple concurrent HTTPS requests to the application server.
Output Specification	The application server must answer the request in a reasonable time with the applied load.
Environmental Needs	GlassFish server.
Test Description	This test case checks if the performance requirements, described in the RASD, are fulfilled.
Testing Method	Automated with Apache JMeter.

3.7 Integration test case I19

Test Case Identifier	I19T1
Test Item(s)	WebController → (Business) ServletContainer
Input Specification	Methods calls from the WebController.
Output Specification	The WebController executes the correct API call.
Environmental Needs	GlassFish server, drivers for the WebController.
Test Description	The purpose of these tests is to check if the WebController executes the correct API call.
Testing Method	Automated with JUnit and Mockito.

3.8 Integration test case I20

Test Case Identifier	I20T1
Test Item(s)	JSP → WebController
Input Specification	Manual interaction with the JSPs.
Output Specification	The JSPs display the web pages correctly according to the injected parameters and execute the right operation.
Environmental Needs	GlassFish server.
Test Description	The purpose of these tests is to check the behavior of the JSP and its graphical components.
Testing Method	Manual testing.

3.9 Integration test cases SI2, I21

Test Case Identifier	SI2T1, I21T1
Test Item(s)	WebContainer → JSP
Input Specification	HTTPS requests to the web server.
Output Specification	The WebContainer invokes the correct JSPs and executes them.
Environmental Needs	GlassFish server, web browser.
Test Description	The purpose of these tests is to check if the correct JSP has been called and the iteractions between JSPs.
Testing Method	Automated with JUnit and Arquillian.

3.10 Integration test case SI2

Test Case Identifier	SI2T2
Test Item(s)	Web → Businnes
Input Specification	Multiple concurrent HTTPS requests to the web server.
Output Specification	The web server must answer the request in a reasonable time with the applied load.
Environmental Needs	GlassFish server.
Test Description	This test case checks if the performance requirements, described in the RASD, are fulfilled.
Testing Method	Automated with Apache JMeter.

3.11 Integration test case I22

Test Case Identifier	I22T1
Test Item(s)	iOSController → (Business) ServletContainer
Input Specification	Methods calls from the iOSController.
Output Specification	The iOSController executes the correct API call.
Environmental Needs	Xcode, iOS Simulator, drivers for the iOSController.
Test Description	The purpose of these tests is to check if the iOSController executes the correct API call.
Testing Method	Automated (iOS testing suite).

3.12 Integration test case I23

Test Case Identifier	I23T1
Test Item(s)	AndroidController → (Business) Servlet-Container
Input Specification	Methods calls from the AndroidController.
Output Specification	The AndroidController executes the correct API call.
Environmental Needs	Android Emulator, drivers for the AndroidController.
Test Description	The purpose of these tests is to check if the AndroidController executes the correct API call.
Testing Method	Automated (Android testing suite).

3.13 Integration test case I24

Test Case Identifier	I24T1
Test Item(s)	iOSGPSManager → iOSController
Input Specification	Method calls from the iOSGPSManager.
Output Specification	The returned value of the method calls is the correct geographical position.
Environmental Needs	Xcode, iOS Simulator, driver for the iOSGPSManager.
Test Description	The purpose of these tests is to check if the iOSGPSManager returns the correct geographical position.
Testing Method	Automated (iOS testing suite).

3.14 Integration test case I25

Test Case Identifier	I25T1
Test Item(s)	AndroidGPSManager → AndroidController
Input Specification	Method calls from the AndroidGPSManager.
Output Specification	The returned value of the method calls is the correct geographical position.
Environmental Needs	Android Emulator, drivers for the AndroidGPSManager.
Test Description	The purpose of these tests is to check if the AndroidGPSManager returns the correct geographical position.
Testing Method	Automated (Android testing suite).

3.15 Integration test cases SI3, I26

Test Case Identifier	SI3T1, I26T1
Test Item(s)	iOSUIManager → iOSController
Input Specification	Manual interaction with the GUI.
Output Specification	The GUI reacts correctly to the executed operations.
Environmental Needs	Xcode, iOS Simulator.
Test Description	The purpose of these tests is to check the behavior of the iOS GUI and its graphical components.
Testing Method	Manual testing.

3.16 Integration test cases SI3, I27

Test Case Identifier	SI3T2, I27T1
Test Item(s)	AndroidUIManager → AndroidController
Input Specification	Manual interaction with the GUI.
Output Specification	The GUI reacts correctly to the executed operations.
Environmental Needs	Android Emulator.
Test Description	The purpose of these tests is to check the behavior of the Android GUI and its graphical components.
Testing Method	Manual testing.

3.17 Integration test case I28

Test Case Identifier	I28T1
Test Item(s)	CarApplication → (Business) ServletContainer
Input Specification	Runs the CarApplication.
Output Specification	The CarApplication sends the correct information every 10 seconds.
Environmental Needs	Linux.
Test Description	The purpose of these tests is to check if the CarApplication sends the correct information to the system every 10 seconds.
Testing Method	Automated using JUnit, manual testing.

3.18 Integration test cases SI4, I29

Test Case Identifier	SI4T1, I29T1
Test Item(s)	CarGUI → (Business) ServletContainer
Input Specification	Runs the CarGUI application.
Output Specification	The application allows us to interact with some utility tools.
Environmental Needs	Linux.
Test Description	The purpose of these tests is to check if the utility tools of the application work correctly.
Testing Method	Automated using JUnit, manual testing.

4 Tools and Test Equipment Required

The software tools used to automate the integration testing are the following:

- **JUnit**
  JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.
  http://junit.org/junit4/

- **Mockito**
  Mockito is a mocking framework that lets you write tests with a clean & simple API. Mockito doesnt give you hangover because the tests are very readable and they produce clean verification errors.
  http://site.mockito.org/

- **Arquillian**
  Arquillian is a test framework that can be used to perform testing inside a remote or embedded container, or deploy an archive to a container so the test can interact as a remote client.
  http://arquillian.org/

- **Apache JMeter**
  The Apache JMeter is an open source Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.
  http://jmeter.apache.org/

5 Program Stubs and Test Data Required

In order to perform an efficient integration testing we require some drivers aimed to simulate the behavior of the real software.

Drivers for Java Entity Beans: drivers aimed to test the communication between the application server and the database, and the correctness of the queries. They are developed using JUnit and Arquillian.

Drivers for Java Session Beans: drivers aimed to test the correctness of the beans' methods. They are developed using JUnit and Arquillian.

Drivers for Servlets: drivers aimed to test the correctness of the Servlets' methods. They are developed using JUnit and Mockito.

Drivers for containers: drivers aimed to test if the correct component is injected. They are developed using JUnit and Arquillian.

Drivers for controllers: drivers aimed to test the communication between client applications/web server and the application server, and the correctness of the API calls executed. They are developed using JUnit and Mockito.

A Appendix

A.1 Hours of work

This is the time spent redacting the ITPD

- Lorenzo Binosi - 25 hours

References

[1] Software Engineering 2 Project, AA 2016/2017 - *Goal and approach, schedule and rules*

[2] Software Engineering 2 Project, AA 2016/2017 - *Assignments 3*

[3] Lorenzo Binosi, *Software Engineering 2: PowerEnJoy - Requirements Analysis and Specification Document*

[4] Lorenzo Binosi, *Software Engineering 2: PowerEnJoy - Design Document.*