# Software Engineering 2 Project: PowerEnJoy
# Integration Test Plan Document

POLITECNICO
MILANO 1863

- Alessio Mongelluzzo
- Michele Ferri
- Mattia Maffioli

# Contents

# 1 Introduction

## 1.1 Purpose and Scope

### 1.1.1 Purpose

This document is the Integration Test Plan Document (ITPD) for the PowerEnJoy software. Its purpose is to determine how to accomplish the integration test of the software, which tools are to be used and which approach will be followed.

### 1.1.2 Scope

PowerEnJoy is a system which supports a car-sharing service employing electric cars.

## 1.2 List of Definitions and Abbreviations

- **RASD**: Requirements Analysis and Specification Document.
- **DD**: Design Document.
- **ITPD**: Integration Test Plan Document (this document).
- **RDBMS**: Relational Data Base Management System.
- **DB**: the database layer, handled by a RDBMS.
- **Application server**: the layer which provides the application logic and interacts with the DB and with the front-ends.
- **Back-end**: term used to identify the Application server.
- **Front-end**: the components which use the application server services, namely the web front-end and the mobile applications.
- **Web server**: the component that implements the web-based front-end. It interacts with the application server and with the users' browsers.
- **JSF**: Java Server Faces.

# 2 Integration Strategy

## 2.1 Entry Criteria

This section describes the prerequisites that need to be met before integration testing can be started.

All the classes and methods must pass thorough unit tests which should reasonably discover major issues in the structure of the classes or in the implementation of the algorithms. Unit tests should have a minimum coverage of 90% of the LOC and should be run automatically at each build using JUnit. Unit testing is not in the scope of this document and will not be specified in further detail.

Moreover, code inspection has to be performed on all the codebase in order to ensure maintainability, respect of conventions and find possible issues which could increase the effort of the testers in the next testing phases. Code inspection must be performed using automated tools as much as possible: manual testing should be reserved for the most difficult features to test.

Finally, the documentation of all classes and functions, written using JavaDoc, has to be complete and up-to-date in order to be used as a reference for integration testing development. In particular the public interfaces of each class and module should be well specified. Where necessary, a formal specification language can be used.

The following documents must be delivered before integration testing can begin:

- Requirement Analysis and Specification Document of PowerEnJoy
- Design Document of PowerEnJoy
- Integration Testing Plan Document (this document)

## 2.2 Elements to be integrated

In the Design Document we outlined four major high-level components, corresponding to the system tiers, which from now on will be referred to as subsystems:

**Database tier**. This is the DBMS: it is not part of the software to be developed, but has to be integrated.

**Business tier**. This subsystem implements all the application logic and communicates with the front-ends.

**Web tier**. The web tier implements the web interface and communicates with the business tier and the browser clients.

**Client tier**. The client tier consists of desktop web browsers and our mobile applications.

The integration process of our software is performed on two levels.

1. integration of the different components (classes, Java Beans) inside the same subsystem;

2. integration of different subsystems.

The first step needs to be performed only for the component which contains the pieces of software that we are going to develop, namely the business tier, the mobile application in the client tier and part of the web tier (which, as stated in the DD, uses JSF to implement the web application).

## 2.3 Integration Testing Strategy

The integration strategy of choice is the bottom-up approach. This choice comes natural since we assume we already have the unit tests for the smaller components, so we can proceed from the bottom.

Moreover, the higher-level subsystems outlined in section 2.2 are well separated and loosely coupled since they correspond to different tiers; they also communicate through well-defined interfaces (REST API, HTTP), so they will not be hard to integrate at a later time. In this way it will be possible also to limit the stubs needed in order to accomplish the integration, because the specific components do not use the general ones, so they do not require stubs.

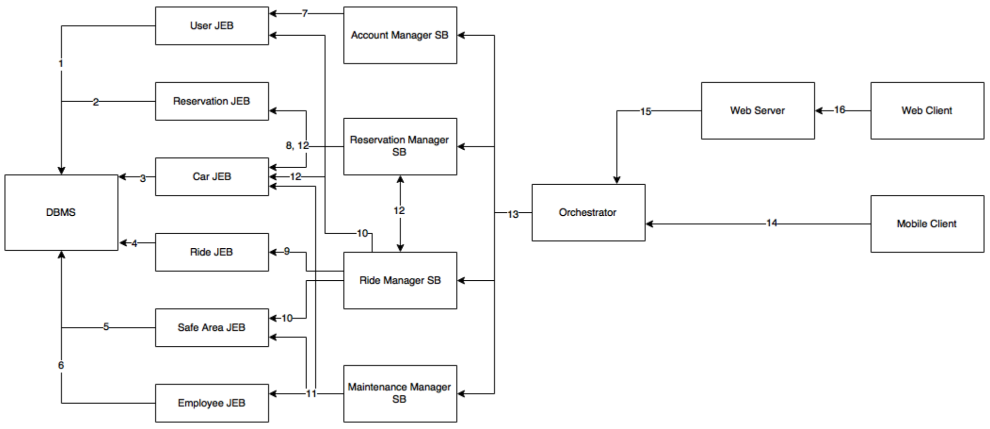

Figure 2.1: Interaction between components in our integration testing plan.

## 2.4 Sequence of Component / Function Integration

### 2.4.1 Software Integration Sequence

The integration sequence of the components is described in Table 2.1 and in Figure 2.1. The components are tested starting from the most independent one to the less independent one. This gives the opportunity to avoid the implementation of useless stubs, because when less independent components are tested, the components which they rely on have already been integrated. The components are integrated within their classes in order to create an integrated subsystem which is ready for subsystem integration.

| # | Involved subsystems | Component | Integrates with |
|---|---|---|---|
| IT1 | Database, Business | (JEB) User | DBMS |
| IT2 | Database, Business | (JEB) Reservation | DBMS |
| IT3 | Database, Business | (JEB) Car | DBMS |
| IT4 | Database, Business | (JEB) Ride | DBMS |
| IT5 | Database, Business | (JEB) Safe Area | DBMS |
| IT6 | Database, Business | (JEB) Employee | DBMS |
| IT7 | Business | (SB) AccountManager | (JEB) User |
| IT8 | Business | (SB) ReservationManager | (JEB) Car<br>(JEB) Reservation |
| IT9 | Business | (SB) RideManager | (JEB) Ride |
| IT10 | Business | (SB) RideManager | (JEB) Safe Area<br>(JEB) User<br>(JEB) Car |
| IT11 | Business | (SB) MaintenanceManager | (JEB) Safe Area<br>(JEB) Employee<br>(JEB) Car |
| IT12 | Business | (SB) RideManager | (SB) ReservationManager |
| IT13 | Business | Orchestrator | (SB) AccountManager<br>(SB) ReservationManager<br>(SB) RideManager<br>(SB) MaintenanceManager |
| IT14 | Mobile | Mobile Client | Orchestrator |
| IT15 | Web | Web Server | Orchestrator |
| IT16 | Web | Web Client | Web Server |

Table 2.1: Integration order of the system components.

### 2.4.2 Subsystem Integration Sequence

The integration sequence of the subsystems is described in Table 2.2 and in Figure 2.2.

A choice was made to proceed with the integration process from the server side towards the client applications, of which mobile app is integrated before the web tier. The reason to do so is that, first of all, in order to have a functioning client you need to have a working business tier. The business tier, instead, can be tested without any client, by making API calls also in an automated fashion. And secondly, by integrating the mobile application before the web tier, we aim to obtain a fully operational client-server system as soon as possible. The web tier is less essential and can be integrated after the app.

| # | **Subsystem** | **Integrates with** |
|---|---|---|
| ST1 | Business tier | Database tier |
| ST2 | Mobile application | Business tier |
| ST3 | Web tier | Business tier |
| ST4 | Client browser | Web tier |

Table 2.2: Integration order of the subsystem components.

# 3 Individual Steps and Test Description

In this chapter we will analyse the single test cases to be carried out. The integration plan we follow is the one stated in figure 2.1. We first detail the subsystems interaction test cases and then the single components interactions are tested.

## 3.1 Integration test case ST1

| | |
|---|---|
| **Test Case Identifier** | ST1 |
| **Test Items** | Business tier, Database tier |
| **Input** | Business tier method calls of Java Persistence API. |
| **Output** | The DBMS must perform the correct queries and retrieve the requested tuples. |
| **Environmental Needs** | Java Entity Beans and Java Persistence API, driver of a Session Bean that mocks information retrieval request. |
| **Description** | Information retrieval is tested checking if the right information is returned. The DBMS should properly react returning a warning message if it is requested invalid data. |
| **Testing Method** | JUnit, Mockito |

## 3.2 Integration test case ST2

| | |
|---|---|
| **Test Case Identifier** | ST2 |
| **Test Items** | Web tier, Business tier |
| **Input** | Web tier requests for Business tier's offered services through REST API. |
| **Output** | The Business tier must process the Web tier's requests and call the proper services, accordingly to REST API specifications. |
| **Environmental Needs** | The Business tier has to be fully implemented and a Web tier driver is necessary to mock services' requests. |
| **Description** | The objective of this test is to check that the Business tier calls for the correct service corresponding to the Web tier's HTTP request , as stated in the REST API specifications. In case of incorrect requests the Web tier should receive an error code. |
| **Testing Method** | JUnit, Mockito |

## 3.3 Integration test case ST3

| | |
|---|---|
| **Test Case Identifier** | ST3 |
| **Test Items** | Mobile application, Business tier |
| **Input** | Mobile application requests for Business tier's services through REST API. |
| **Output** | The Business tier must process the requests and offer the correct service to the Mobile tier . |
| **Environmental Needs** | The Business tier has to be fully implemented and a mockup of the Mobile application is needed to send the requests. |
| **Description** | The Business tier's called services are checked to test whether REST API works correctly and the selected resources correspond to the ones the Mobile application asked for. In case of incorrect requests the Mobile application should receive an error code. |
| **Testing Method** | JUnit, Mockito |

## 3.4 Integration test case ST4

| | |
|---|---|
| **Test Case Identifier** | ST4 |
| **Test Items** | Client browser, Web tier |
| **Input** | HTTPS pages requests by Client browser. |
| **Output** | The Web tier must show to the Client browser the requested pages. |
| **Environmental Needs** | The server side has to be fully implemented, included the Web tier in order to process the web pages requests; a Client browser driver capable of sending HTTP requests is required for the client side. |
| **Description** | This test has to ensure that the HTTP requests are correctly handled by the server, which has to show the requested web pages to the client, while in case of incorrect HTTP requests the result has to be an error code sent to the client. |
| **Testing Method** | JUnit, Mockito |

## 3.5 Integration test case IT1

| | |
|---|---|
| **Test Case Identifier** | IT1 |
| **Test Items** | User Java Entity Bean, DBMS |
| **Input** | JEB queries on User relation. |
| **Output** | Expected query result. |
| **Environmental Needs** | Testing database to perform queries on, JEB stubs to access Java Persistence API, GlassFish server. |
| **Description** | This test aims at checking that the queries performed through the JPA retrieve the requested information and the modifications to the database are committed and persistent. If the Entity bean queries the database for invalid information or tries to illegally modify some values, then the DBMS should return a warning message. |
| **Testing Method** | JUnit, Mockito |

## 3.6 Integration test case IT2

| | |
|---|---|
| **Test Case Identifier** | IT2 |
| **Test Items** | Car Java Entity Bean, DBMS |
| **Input** | JEB queries on Car relation. |
| **Output** | Expected query result. |
| **Environmental Needs** | Testing database to perform queries on, JEB stubs to access Java Persistence API, GlassFish server. |
| **Description** | This test aims at checking that the queries performed through the JPA retrieve the requested information and the modifications to the database are committed and persistent. If the Entity bean queries the database for invalid information or tries to illegally modify some values, then the DBMS should return a warning message. |
| **Testing Method** | JUnit, Mockito |

## 3.7 Integration test case IT3

| | |
|---|---|
| **Test Case Identifier** | IT3 |
| **Test Items** | Reservation Java Entity Bean, DBMS |
| **Input** | JEB queries on Reservation relation. |
| **Output** | Expected query result. |
| **Environmental Needs** | Testing database to perform queries on, JEB stubs to access Java Persistence API, GlassFish server. |
| **Description** | This test aims at checking that the queries performed through the JPA retrieve the requested information and the modifications to the database are committed and persistent. If the Entity bean queries the database for invalid information or tries to illegally modify some values, then the DBMS should return a warning message. |
| **Testing Method** | JUnit, Mockito |

## 3.8 Integration test case IT4

| | |
|---|---|
| **Test Case Identifier** | IT4 |
| **Test Items** | Ride Java Entity Bean, DBMS |
| **Input** | JEB queries on Ride relation. |
| **Output** | Expected query result. |
| **Environmental Needs** | Testing database to perform queries on, JEB stubs to access Java Persistence API, GlassFish server. |
| **Description** | This test aims at checking that the queries performed through the JPA retrieve the requested information and the modifications to the database are committed and persistent. If the Entity bean queries the database for invalid information or tries to illegally modify some values, then the DBMS should return a warning message. |
| **Testing Method** | JUnit, Mockito |

## 3.9 Integration test case IT5

| | |
|---|---|
| **Test Case Identifier** | IT5 |
| **Test Items** | Safe Area Java Entity Bean, DBMS |
| **Input** | JEB queries on Safe Area relation. |
| **Output** | Expected query result. |
| **Environmental Needs** | Testing database to perform queries on, JEB stubs to access Java Persistence API, GlassFish server. |
| **Description** | This test aims at checking that the queries performed through the JPA retrieve the requested information and the modifications to the database are committed and persistent. If the Entity bean queries the database for invalid information or tries to illegally modify some values, then the DBMS should return a warning message. |
| **Testing Method** | JUnit, Mockito |

## 3.10 Integration test case IT6

| | |
|---|---|
| **Test Case Identifier** | IT6 |
| **Test Items** | Employee Java Entity Bean, DBMS |
| **Input** | JEB queries on Employee relation. |
| **Output** | Expected query result. |
| **Environmental Needs** | Testing database to perform queries on, JEB stubs to access Java Persistence API, GlassFish server. |
| **Description** | This test aims at checking that the queries performed through the JPA retrieve the requested information and the modifications to the database are committed and persistent. If the Entity bean queries the database for invalid information or tries to illegally modify some values, then the DBMS should return a warning message. |
| **Testing Method** | JUnit, Mockito |

### 3.11 Integration test case IT7

| | |
|---|---|
| **Test Case Identifier** | IT7 |
| **Test Items** | AccountManager Session Bean, User Java Entity Bean |
| **Input** | AccountManager calls for JPA methods to modify user tuples. |
| **Output** | JEB confirmation message of successful modification over user information . |
| **Environmental Needs** | JPA fully implemented, AccountManager driver that calls for JPA methods, GlassFish server. |
| **Description** | Check that AccountManager session bean successfully calls for JPA to modify database tables. |
| **Testing Method** | JUnit, Mockito |

### 3.12 Integration test case IT8

| | |
|---|---|
| **Test Case Identifier** | IT8 |
| **Test Items** | ReservationManager Session Bean, Reservation Java Entity Bean, Car Java Entity Bean |
| **Input** | ReservationManager calls for JPA methods to reserve a car. |
| **Output** | A reservation must be created for the requested car and the reserved car state has to be updated. |
| **Environmental Needs** | JPA fully implemented, ReservationManager driver that calls for JPA methods, GlassFish server. |
| **Description** | Check that JPA calls made by the ReservationManager succesfully lead to the creation of a tuple for the reservation in the database and that it is referred to the correct user and to the requested car; verify that the car state is coherently updated to RESERVED. |
| **Testing Method** | JUnit, Mockito |

## 3.13 Integration test case IT9

| | |
|---|---|
| **Test Case Identifier** | IT9 |
| **Test Items** | RideManager Session Bean, Ride Java Entity Bean |
| **Input** | RideManager calls for JPA methods to modify ride table. |
| **Output** | Ride JEB modifies ride relation and returns an outcome message. |
| **Environmental Needs** | JPA fully implemented, RideManager driver that calls for JPA methods, GlassFish server. |
| **Description** | Check that JPA calls made by the RideManager succesfully lead to the creation of a tuple for the ride in the database and that it is referred to the correct user and to the selected car; verify that the car state is coherently updated to RUNNING. |
| **Testing Method** | JUnit, Mockito |

## 3.14 Integration test case IT10

| | |
|---|---|
| **Test Case Identifier** | IT10 |
| **Test Items** | RideManager Session Bean, Safe Area Java Entity Bean, User Java Entity Bean, Car Java Entity Bean |
| **Input** | RideManager calls for JPA methods to conclude a ride. |
| **Output** | JEBs return an outcome message and changes to the database are made persistent. |
| **Environmental Needs** | JPA fully implemented, RideManager driver that calls for JPA methods, GlassFish server, stub mocking payment interaction. |
| **Description** | RideManager can retrieve the information to correctly compute a possible discount or additional charge and coherently updates the database; the car state is coherently updated to AVAILABLE; the ride state is coherently updated to COMPLETED; the user table is updated with a 'banned' flag if the payment is not successful. |
| **Testing Method** | JUnit, Mockito |

## 3.15 Integration test case IT11

| | |
|---:|---|
| **Test Case Identifier** | IT11 |
| **Test Items** | MaintenanceManager Session Bean, Safe Area Java Entity Bean, Employee Entity Bean, Car Entity Bean |
| **Input** | MaintenanceManager methods call to modify the database. |
| **Output** | Java Entity Beans return a message stating the outcome of the operations. |
| **Environmental Needs** | JPA fully implemented, MaintenanceManager driver to mock methods call to JEBs, GlassFish server. |
| **Description** | Check that Cars, Safe Areas and Employees information stored in the database can be modified. |
| **Testing Method** | JUnit, Mockito |

## 3.16 Integration test case IT12

| | |
|---:|---|
| **Test Case Identifier** | IT12 |
| **Test Items** | RideManager Session Bean, ReservationManager Session Bean, Reservation Java Entity Bean, Car Java Entity Bean |
| **Input** | RideManager interacts with ReservationManager to retrieve information necessary to initialize a ride. |
| **Output** | ReservationManager provides the correct information needed for the ride to be initialized; Car JEB returns a confirmation message and updates on Car entity are made persistent. |
| **Environmental Needs** | JPA fully implemented, RideManager driver that interacts with ReservationManager and calls for JPA methods, GlassFish server. |
| **Description** | RideManager can retrieve the correct data in order to create a ride from a pending reservation; verify that the car state is coherently updated from RESERVED to RUNNING. |
| **Testing Method** | JUnit, Mockito |

## 3.17 Integration test case IT13

| | |
|---|---|
| **Test Case Identifier** | IT13 |
| **Test Items** | Orchestrator, AccountManager Session Bean, ReservationManager Session Bean, RideManager Session Bean, MaintenanceManager Session Bean |
| **Input** | Requests from the Orchestrator for the functionalities offered by Session Beans. |
| **Output** | The Orchestrator has to be able to provide the right functionality carrying out the proper request to the Session Beans. |
| **Environmental Needs** | GlassFish server. |
| **Description** | Ensure that the Orchestrator is able to provide the functionalities of the system offered by the Session Beans. |
| **Testing Method** | JUnit |

## 3.18 Integration test case IT14

| | |
|---|---|
| **Test Case Identifier** | IT14 |
| **Test Items** | Mobile Client, Orchestrator |
| **Input** | The mobile client calls for REST API to exploit business functionalities. |
| **Output** | The orchestrator processes the request and calls the correct REST resource methods. |
| **Environmental Needs** | Driver for client APIs, fully developed business layer, GlassFish server. |
| **Description** | Verify that the mobile client can successfully access to the business functionalities. |
| **Testing Method** | JUnit, Mockito |

### 3.19 Integration test case IT15

| | |
|---:|---|
| **Test Case Identifier** | IT15 |
| **Test Items** | Web Server, Orchestrator |
| **Input** | The Web Server calls for REST API to exploit business functionalities. |
| **Output** | The orchestrator processes the request and calls the correct REST resource methods. |
| **Environmental Needs** | Driver for the Web Server, fully developed business layer, GlassFish server. |
| **Description** | Verify that the Web Server can successfully access to the business functionalities the web client has asked for. |
| **Testing Method** | JUnit, Mockito |

### 3.20 Integration test case IT16

| | |
|---:|---|
| **Test Case Identifier** | IT16 |
| **Test Items** | Web Client, Web Server |
| **Input** | Web client browser connects to the web application and sends HTTP requests to the web server. |
| **Output** | The web server processes the HTTP request and displays to the client the correct dynamic web page. |
| **Environmental Needs** | Fully developed Web Server, driver for client APIs , GlassFish server. |
| **Description** | Verify that the web server can display the correct web pages to the web client browser. |
| **Testing Method** | JUnit, Mockito |

## 4 Tools and Test Equipment Required

The tools and software we plan to use to perform the integration testing are the following:

- **JUnit:** JUnit is a unit testing and integration testing framework for the Java programming language. We plan to use it both to perform the unit testing and the integration testing together with other tools.
- **Mockito:** Mockito is an open source testing framework for Java. We use it to create stubs and drivers to mock the behaviour of called and caller functions respectively.
- **GlassFish:** GlassFish is an open-source application server project for the Java EE platform. We use it as the server for our project.

## 5 Program Stubs and Test Data Required

We use stubs and drivers to mock the behaviour of the software components that don't exist yet and to test the parts interacting with them. In order to perform integration testing without having developed the entire system first, we need to use stubs and drivers to take the part of the software components that still don't exist and test the others.

**Client**: in order to test the REST API of the business tier without the actual client application, a simple API client which interacts with the business tier by simple HTTP requests is needed.

Stubs of the **Business** components: used to provide a minimum set of data to test the web tier when the business tier is not fully developed.

Drivers for the **Java Entity Beans**: they are used to test the Java Entity Beans when the Business Tier is not fully developed. They call the relevant methods of the EJBs to test the correctness of the queries.

Testing **Database**: the test data contained in this database includes a reduced set of instances of all the entities described in the Entity-Relation diagram of the Design Document.

## A Appendix A

### A.1 Software and tools used

- LaTeX for typesetting the document.
- LyX as a document processor.
- GitHub for version control and teamwork.

### A.2 Hours of work

- Alessio Mongelluzzo: 10 hours
- Michele Ferri: 8 hours
- Mattia Maffioli: 9 hours