POLITECNICO
MILANO 1863

PowerEnJoy
Integration Test Plan Document

Erba Alessandro
Leveni Filippo
Lodi Luca

A.A 2016/2017

Contents

1 Introduction

1.1 Revision History

1.2 Purpose and Scope

This document will provide the guideline to accomplish the Integration test for the *PowerEnjoy* service, it will provide specifically:

- the identification of the software components that are going to be integrated, with references to the Design Document according to the Requirement Analysis Specification Document.
- the organization of testing on subsystems in which *PowerEnjoy* system is split.
- the schedule and description of the expected integration tests.
- the description of the used test tool.

1.3 List of Definitions and Abbreviations

1.3.1 Rent

A *rent* is the activity beginning with the pick up and ending with the release of a car. It can include some *stopover* .

1.3.2 Car Sharing

car sharing is a model of car rental where people rent cars for short periods of time, often by the hour.

1.3.3 PowerEnJoy

PowerEnjoy is the *car sharing* brand object of this document.

1.3.4 Guest User

A *guest* is a person not logged in to the system.

1.3.5 Registered User

A *registered user* is a person registered to *PowerEnjoy* .

1.3.6 PowerEnJoy User

A *PowerEnjoy User* is a *registered user* can access to the *car sharing* through his smart device.

1.3.7 Staff

staff is the set of people that are *registered user* but can perform special operations. They are divided in three different categories.

- Field *staff*
  - They own a passepartout for cars reporting any issues

    (e.g: cars with low battery).
- Emergency *staff*
  - They manage users issues. They are able to directly contact users.
- Management *staff*
  - They manage the system, configuring its parameters. They can add new *safe areas* , modify *car sharing* fares and create new *staff* accounts.

1.3.8 Third Party Developer

third party developer is a person that can fulfill operations on *PowerEnjoy* system via API.

1.3.9 Service

The *service* is the group of actions that a *PowerEnjoy User* can fulfill trough his smart device.

1.3.10 Safe Area

A *safe area* is a geographical place[1] on the map in which parking is allowed. *safe area* are saved into the system. A *rent* can end parking the car only in safe areas.

1.3.11 Power Plug

A *power plug* is the station that charges the battery of an electric car. All *power plugs* are displayed in the map on-board *PowerEnjoy* cars. Each *power plug* is in a *safe area* . A *PowerEnjoy User* who plugs his car into a *power plug* receives a discount according to *PowerEnjoy* rules.

1.3.12 Power Plug Slot

A *power plug slot* is a parking slot reserved for the *power plug* charge of cars.

[1]defined by a set of geolocation positions

1.3.13 Parking solution

Either a *safe area* or a *power plug slot* .

1.3.14 Reservation

A *reservation* is the action of booking a car, within an hour from its pick up.

1.3.15 Stopover

A *stopover* is a temporary power-off of a rented car. During a *stopover* the *PowerEnjoy User* can leave the car, and re-unlock it later through his smart device. It is subject to a different fare than the rest of rent time.

1.3.16 Fare

A policy to compute price of a rent, complete of involved parameters.

E.g: proportional to time, 0.67€/minute during active rent time

1.3.17 Personal Identification Number (PIN)

personal pin is a number of 5 digits that the user inserts on the car screen to enable driving commands.

1.4 List of Reference Documents

References

[1] Erba A., Leveni F., Lodi L., *Requirement Analysis Specification Document*, 2016.

[2] Erba A., Leveni F., Lodi L., *Design Document*, 2016.

2 Integration Strategy

2.1 Entry Criteria

In order to make the Integration test feasible the following conditions must be verified before it begins.

- Requirement Analysis and Specification Document and Design Document must be complete in each section and already approved.
- All the parts of the component required by an integration test must be implemented.
- All the tested component's sub-parts (related to an integration test) are unit-tested and bug free.

- A risk assessment must already be evaluated (for the Critical modules approach)
- Integration Test data set has already been selected.

Another important thing that we assume as milestone at this point is that all the external services that we interact with are already been tested during unit test. The point is that is impossible to have correctly completed the unit test without interfacing with external services.

Specifically the following components are already integrated: *Push Notification Adapter, Google Maps Adapter, Car navigation Adapter, Car Controller, Document Validation Service Adapter, Payment Service Adapter.*

2.2 Elements to be Integrated

Given the component diagram at section 2.2 of the Design Document every component colored in Orange and Yellowish will be integrated. Specifically the components will be integrated among them following the interfaces that they share.

2.3 Integration Testing Strategy

Among all the different Integration Testing Strategy we have chosen an approach based on the Critical Modules.

Our System is quite complex, we have a lot of features that supports the core business. During our analysis we have decided to give a priority to certain requirements (reference to section 2.1.9 of RASD Document). Following this priority hierarchy we have the need to test integrations of components interfaces related to these requirements.

Following the section 5 (Requirements Traceability) of the Design Document we can easily identify which components interfaces are related to the priorities.

The components (grouped by priority) are ordered by the criticality for the project, following the risk assessment analysis suggestions, from the most risky to the least one.

This means that given all the components of our system, we will sort them following the 2.1.9 RASD Priorities; than each group will be tested using critical modules approach from the riskiest module. We chose this feature-oriented strategy to reduce the risk of project failure, since we have defined a rich set of feature in the previous documents, and also a priority for the introduction of those feature in the software roadmap.

2.4 Sequence of Component/Function Integration

Here we show how the integration test is organized using the strategy chosen above.

We have divided all the components in 3 subsystems. The three subsystems are the one suggested by the priorities of paragraphs 2.1.9 of the RASD Document.

Every subsystem is organized in levels that represents the criticality of the components that it contains. From top to down criticality decreases.

The criticality of each component is related to its complexity and importance. We have carefully analyzed what each component will provide and foresee the effort to fix any possible error.

As already said each module interface is not tested all at once. Each test case will cover certain features. By the end of the integration test phase all the modules will be completely covered.

In the next section we will explain what each test will cover.

2.4.1 PowerEnJoy User related integration test

We have carefully examined the different priorities and we come up with this configuration for the test:

1. **BPM**
2. **Rent manager, Reservation Manager**
3. ***PowerEnjoy* User Manager, Payment Manager**
4. **Map Manager, Service Configuration, Account Manager**
5. ***PowerEnjoy* mobile app**

in Figure 1 you can see the dependencies between components

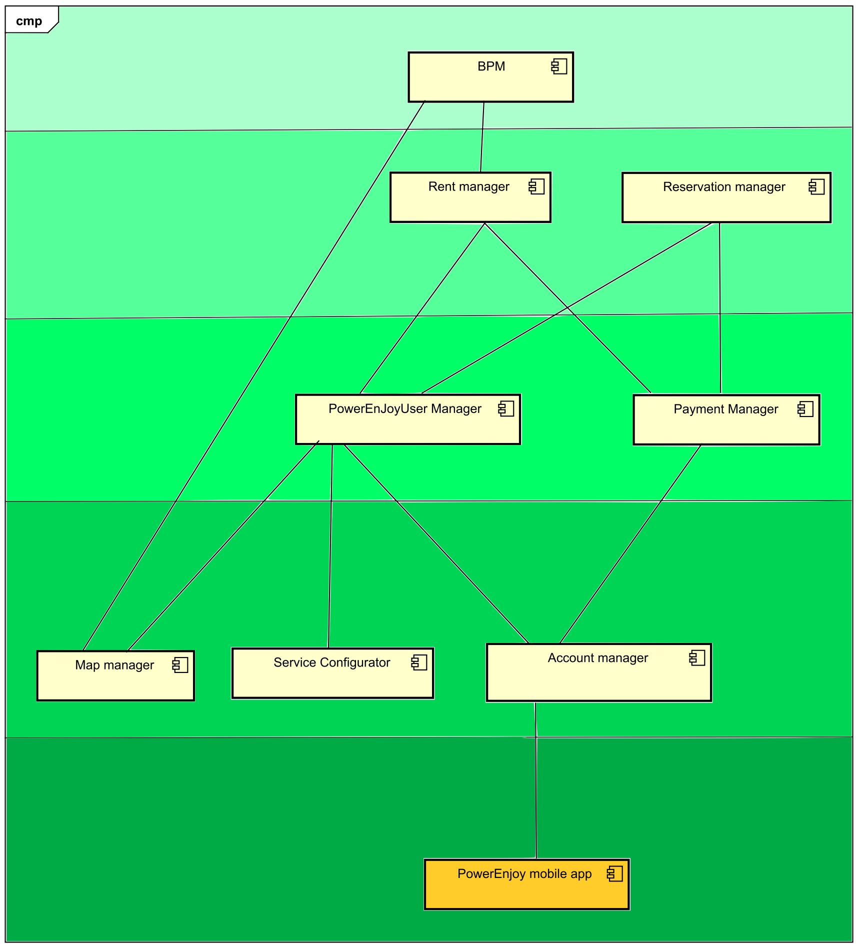

Figure 1: PowerEnJoy User related integration test

2.4.2 Staff and Car related integration test

This integration test is divided in two, one about Emergency Staff, Field Staff and Cars Related Components; the second about Managements staff related components. This split is suggested by the related features of the first group, i.e. there was a high interaction of components of the three categories and it was more easy to group them together.

We have carefully examined the different priorities and we come up with this configuration for the test:

1. **Car Manager**
2. **Emergency Staff manager, Field Staff Manger**

3. **Emergency Staff Web App, Field Staff mobile app, Emergency manager, BPM, Map Manager**
4. **Car App**

in Figure 2 you can see the dependencies between components

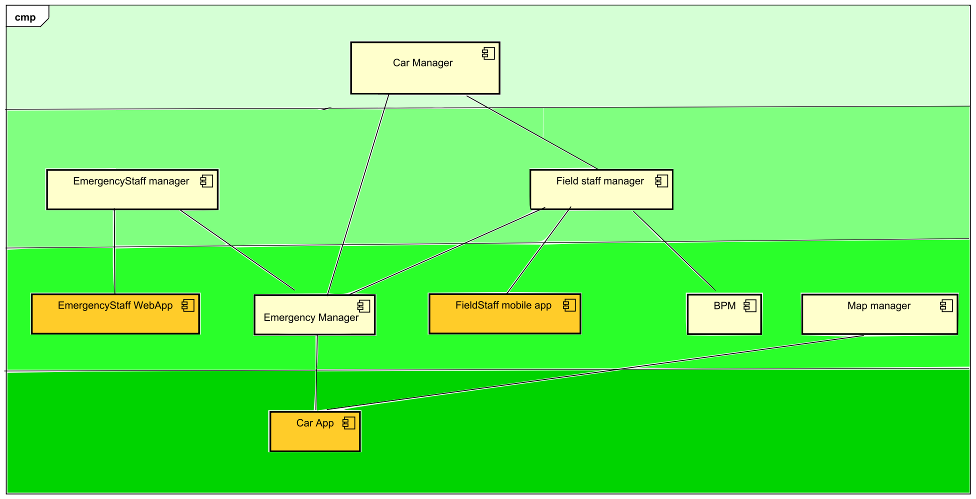

Figure 2: Emergency Staff, Field Staff and Car related integration test

We have carefully examined the different priorities and we come up with this configuration for the test:

1. **Service Configurator**
2. **Management Staff Manager, PowerEnjoy User Manger**
3. **Management Staff Web App, BPM, Rent manager, Reservation Manager**

in Figure 3 you can see the dependencies between components

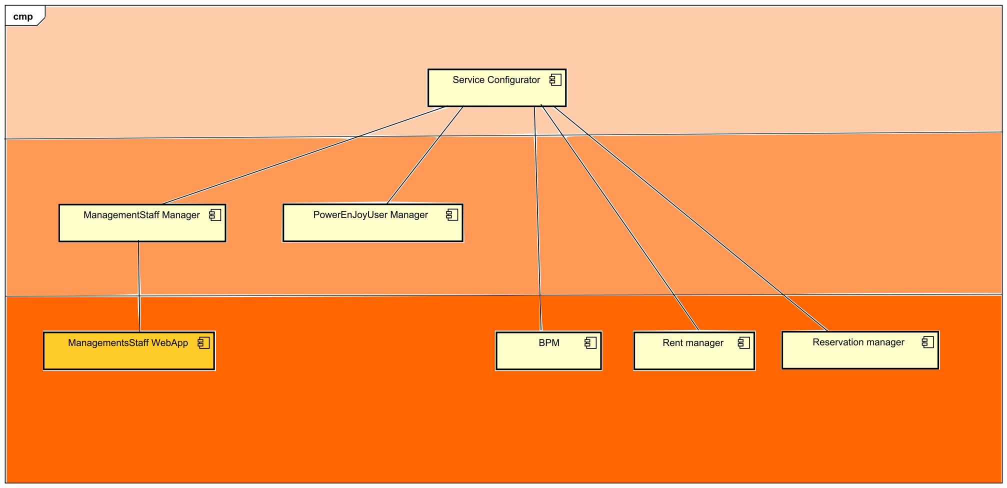

Figure 3: Management Staff related integration test

2.4.3 Third Party related integration test

We have carefully examined the different priorities and we come up with this configuration for the test:

1. **External App Manager**
2. **Account Manager PowerEnjoy User Manager**
3. **Map Manager, Rent manager, Reservation Manager**

in Figure 4 you can see the dependencies between components

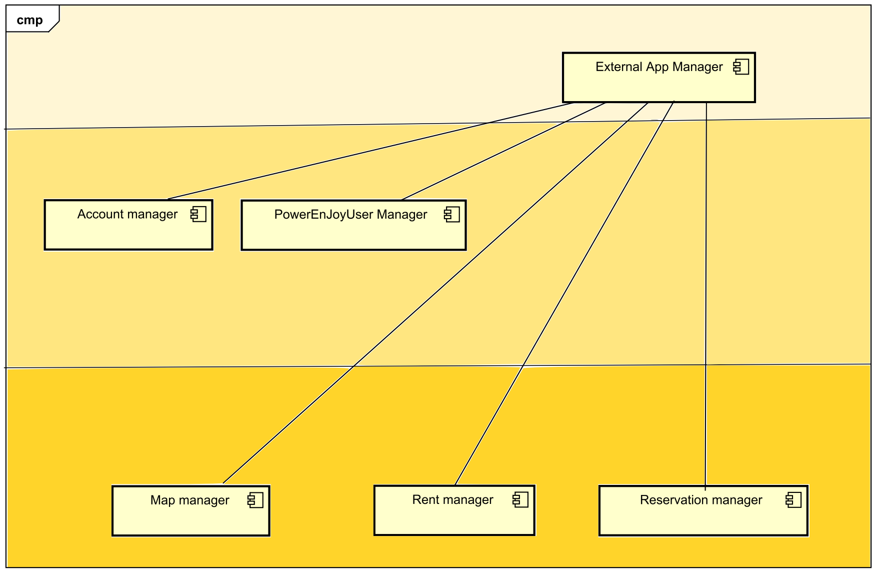

Figure 4: Third Party related integration test

3 Individual Steps and Test Description

In this paragraph we are going to give an high-level description of the integration tests that will be performed on PowerEnJoy system. Since we adopted the critical modules strategy (feature-oriented approach), we do not have to worry about the heavy creation of drivers nor stubs. But we need to "mock" the component's classes not related to the tested interfaces, since they could be not already implemented.

Following the critical module approach, we test the integration of couples of components (the sub-part related to the tested functionality) at a time, in order to find errors as soon as possible in the riskiest modules.

The following tables are structured as:

- **Test Case ID:** a label to identify the integration test under analysis.
- **Test Items:** the couple of involved components; "used by" relationship is represented through an arrow.
- **Input Specification:** the high-level description of the input (coming from the right element of the item's couple previously specified)
- **Output Specification:** the high-level description of the expected output provided the input above described.
- **Environmental Needs:** test execution's conditions, in terms of:

- already performed tests
- already implemented functionalities

NOTE: For example in order to test the integration of BPM and Map manager (they interact through the "Map search API" interface), just a sub-part of those two components need to be already implemented. The unimplemented part of the two components need to be "mocked" (using mockito tool). This apply to all the test cases described in the tables below.

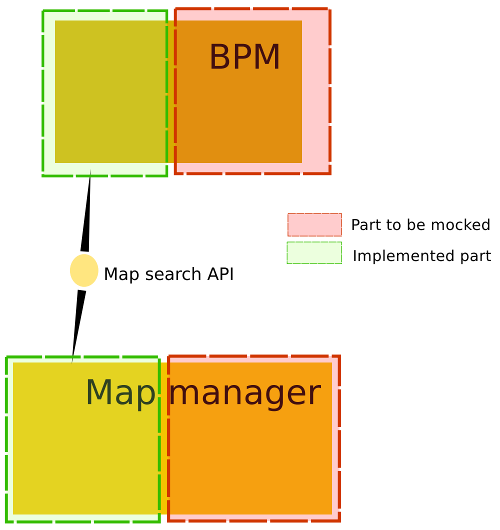

Figure 5: Integration example

3.1 PowerEnJoy User related integration tests

3.1.1 BPM integration test

Test Case ID	I1-T1
Test Items	BPM —> Map manager
Input Specification	a simulation of the typical BPM component input coming from Map manager, attention must be paid to cover the edge or exceptional cases related to the "Map search API" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Map search API" interface must have been already implemented and unit-tested

Test Case ID	I1-T2
Test Items	Rent manager —> BPM
Input Specification	a simulation of the typical Rent manager component input coming from BPM, attention must be paid to cover the edge or exceptional cases related to the “Recommend destination” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Recommend destination” interface must have been already implemented and unit-tested AND I1-T1 must have already been performed

3.1.2 Rent manager integration test

Test Case ID	I2-T1
Test Items	PowerEnJoyUser manager —> Rent manager
Input Specification	a simulation of the typical PowerEnJoyUser manager component input coming from Rent manager, attention must be paid to cover the edge or exceptional cases related to the “Rent API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Rent API” interface must have been already implemented and unit-tested AND I1-T2 must have already been performed

Test Case ID	I2-T2
Test Items	Rent manager —> Payment manager
Input Specification	a simulation of the typical Rent manager component input coming from Payment manager, attention must be paid to cover the edge or exceptional cases related to the “Payment transaction API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Payment transaction API” interface must have been already implemented and unit-tested

3.1.3 Reservation manager integration test

Test Case ID	I3-T1
Test Items	PowerEnJoyUser manager —> Reservation manager
Input Specification	a simulation of the typical PowerEnJoyUser manager component input coming from Reservation manager, attention must be paid to cover the edge or exceptional cases related to the "Reservation API" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Reservation API" interface must have been already implemented and unit-tested

Test Case ID	I3-T2
Test Items	Reservation manager —> Payment manager
Input Specification	a simulation of the typical Reservation manager component input coming from Payment manager, attention must be paid to cover the edge or exceptional cases related to the "Payment transaction API" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Payment transaction API" interface must have been already implemented and unit-tested

3.1.4 PoweEnJoyUser manager integration test

Test Case ID	I4-T1
Test Items	PowerEnJoyUser manager —> Map manager
Input Specification	a simulation of the typical PowerEnJoyUser manager component input coming from Map manager, attention must be paid to cover the edge or exceptional cases related to the "Map search API" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Map search API" interface must have been already implemented and unit-tested

Test Case ID	I4-T2
Test Items	PowerEnJoyUser manager —> Service configurator
Input Specification	a simulation of the typical PowerEnJoyUser manager component input coming from Service configurator, attention must be paid to cover the edge or exceptional cases related to the "Notification API" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Notification API" interface must have been already implemented and unit-tested

Test Case ID	I4-T3
Test Items	PowerEnJoyUser manager —> Account manager
Input Specification	a simulation of the typical PowerEnJoyUser manager component input coming from Account manager, attention must be paid to cover the edge or exceptional cases related to the "Registration API" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Registration API" interface must have been already implemented and unit-tested

3.1.5 Payment manager integration test

Test Case ID	I5-T1
Test Items	Payment manager —> Account manager
Input Specification	a simulation of the typical Payment manager component input coming from Account manager, attention must be paid to cover the edge or exceptional cases related to the “Account API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Account API” interface must have been already implemented and unit-tested

3.1.6 Account manager integration test

Test Case ID	I6-T1
Test Items	PowerEnJoyUser webapp —> Account manager
Input Specification	a simulation of the typical PowerEnJoyUser webapp component input coming from Account manager, attention must be paid to cover the edge or exceptional cases related to the “Login API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Login API” interface must have been already implemented and unit-tested

3.2 Staff and Car related integration tests

3.2.1 Car manager integration test

Test Case ID	I7-T1
Test Items	Car manager —> Emergency manager
Input Specification	a simulation of the typical Car manager component input coming from Emergency manager, attention must be paid to cover the edge or exceptional cases related to the “Availability int” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Availability int” interface must have been already implemented and unit-tested

Test Case ID	I7-T2
Test Items	FieldStaff manager —> Car manager
Input Specification	a simulation of the typical FieldStaff manager component input coming from Car manager, attention must be paid to cover the edge or exceptional cases related to the “Car int” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Car int” interface must have been already implemented and unit-tested

3.2.2 EmergencyStaff manager integration test

Test Case ID	I8-T1
Test Items	EmergencyStaff webapp —> EmergencyStaff manager
Input Specification	a simulation of the typical EmergencyStaff webapp component input coming from EmergencyStaff manager, attention must be paid to cover the edge or exceptional cases related to the "EmergencyStaff int" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "EmergencyStaff int" interface must have been already implemented and unit-tested

Test Case ID	I8-T2
Test Items	Emergency manager —> EmergencyStaff manager
Input Specification	a simulation of the typical Emergency manager component input coming from EmergencyStaff manager, attention must be paid to cover the edge or exceptional cases related to the "EmergencyStaff int" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "EmergencyStaff int" interface must have been already implemented and unit-tested

3.2.3 FieldStaff manager integration test

Test Case ID	I9-T1
Test Items	Emergency manager —> FieldStaff manager
Input Specification	a simulation of the typical Emergency manager component input coming from FieldStaff manager, attention must be paid to cover the edge or exceptional cases related to the "Maintenance request" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Maintenance request" interface must have been already implemented and unit-tested AND I3T1 and I4T1 must have already been performed

Test Case ID	I9-T2
Test Items	FieldStaff webapp —> FieldStaff manager
Input Specification	a simulation of the typical FieldStaff webapp component input coming from FieldStaff manager, attention must be paid to cover the edge or exceptional cases related to the "Field staff int" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Field staff int" interface must have been already implemented and unit-tested

Test Case ID	I9-T3
Test Items	BPM —> FieldStaff manager
Input Specification	a simulation of the typical BPM component input coming from FieldStaff manager, attention must be paid to cover the edge or exceptional cases related to the "Relocation request" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Relocation request" interface must have been already implemented and unit-tested

3.2.4 Emergency manager integration test

Test Case ID	I10-T1
Test Items	Emergency manager —> Car app
Input Specification	a simulation of the typical Emergency manager component input coming from Car app, attention must be paid to cover the edge or exceptional cases related to the "Sensor data" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Sensor data" interface must have been already implemented and unit-tested

3.2.5 Map manager integration test

Test Case ID	I11-T1
Test Items	Car app —> Map manager
Input Specification	a simulation of the typical Car app component input coming from Map manager, attention must be paid to cover the edge or exceptional cases related to the "Map updates" interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The "Map updates" interface must have been already implemented and unit-tested AND I3T1 and I4T1 must have already been performed

3.2.6 Service configurator integration test

Test Case ID	I12-T1
Test Items	ManagementStaff manager —> Service configurator
Input Specification	a simulation of the typical ManagementStaff manager component input coming from Service configurator, attention must be paid to cover the edge or exceptional cases related to the “Configuration int” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Configuration int” interface must have been already implemented and unit-tested

Test Case ID	I12-T2
Test Items	Service configurator —> PowerEnJoyUser manager
Input Specification	a simulation of the typical Service configurator component input coming from PowerEnJoyUser manager, attention must be paid to cover the edge or exceptional cases related to the “Notification API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Notification API” interface must have been already implemented and unit-tested

Test Case ID	I12-T3
Test Items	BPM —> Service configurator
Input Specification	a simulation of the typical BPM component input coming from Service configurator, attention must be paid to cover the edge or exceptional cases related to the “Calculate price” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Calculate price” interface must have been already implemented and unit-tested

Test Case ID	I12-T4
Test Items	Rent manager —> Service configurator
Input Specification	a simulation of the typical Rent manager component input coming from Service configurator, attention must be paid to cover the edge or exceptional cases related to the “Calculate price” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Calculate price” interface must have been already implemented and unit-testedd

Test Case ID	I12-T5
Test Items	Reservation manager —> Service configurator
Input Specification	a simulation of the typical Reservation manager component input coming from Service configurator, attention must be paid to cover the edge or exceptional cases related to the “Calculate price” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Calculate price” interface must have been already implemented and unit-tested

3.2.7 ManagementStaff manager integration test

Test Case ID	I13-T1
Test Items	ManagementStaff webapp —> ManagementStaff manager
Input Specification	a simulation of the typical ManagementStaff webapp component input coming from ManagementStaff manager, attention must be paid to cover the edge or exceptional cases related to the “Management int” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Management int” interface must have been already implemented and unit-tested

3.3 Third Party related integration tests

3.3.1 External app manager integration test

Test Case ID	I14-T1
Test Items	External app manager —> Account manager
Input Specification	a simulation of the typical External app manager component input coming from Account manager, attention must be paid to cover the edge or exceptional cases related to the — interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The — interface must have been already implemented and unit-tested AND I3T1 and I4T1 must have already been performed

Test Case ID	I14-T2
Test Items	External app manager —> PowerEnJoyUser manager
Input Specification	a simulation of the typical External app manager component input coming from PowerEnJoyUser manager, attention must be paid to cover the edge or exceptional cases related to the “PowerEnJoyUser int” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “PowerEnJoyUser int” interface must have been already implemented and unit-tested

Test Case ID	I14-T3
Test Items	External app manager —> Map manager
Input Specification	a simulation of the typical External app manager component input coming from Map manager, attention must be paid to cover the edge or exceptional cases related to the “Map search API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Map search API” interface must have been already implemented and unit-tested

Test Case ID	I14-T4
Test Items	External app manager —> Rent manager
Input Specification	a simulation of the typical External app manager component input coming from Rent manager, attention must be paid to cover the edge or exceptional cases related to the “Rent API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Rent API” interface must have been already implemented and unit-tested

Test Case ID	I14-T5
Test Items	External app manager —> Reservation manager
Input Specification	a simulation of the typical External app manager component input coming from Reservation manager, attention must be paid to cover the edge or exceptional cases related to the “Reservation API” interface
Output Specification	Check whether the correct methods are invoked, along with the correctness of the parameter types and values
Environmental Needs	The “Reservation API” interface must have been already implemented and unit-tested

4 Tools and Test Equipment Required

We decided to automatize the execution of the test and the result's collection through the Arquillian tool. In fact it automatically haldles the initialization of the test environment (Java EE containers and dependency java beans, in our case) easing the test execution.

Arquillian defines two styles of container, remote and embedded. A remote container resides in a separate JVM from the test runner. Its lifecycle may be managed by Arquillian, or Arquillian may bind to a container that is already started. An embedded container resides in the same JVM and is mostly likely managed by Arquillian.

Arquillian represents an optimal solution in a such context where many components have to be integration tested, thanks to its capability to atomatize the setup of the test environment including server, test containers and retrieve the output of the tests.

Therefore Arquillian perfectly meets our requirements since we chose to adopt the Java EE framework, which requires the initialization of its structured environment, step that Arquillian automatizes for us, drastically reducing the test design effort.

Figure 6: Arquillian logo

In order to support the testing of incomplete modules (according to our approach) we rely on Mockito tool for the creation of mocks of the unimplemented classes. Those mocks' purpose is to supply the lack of implementation of classes not related to the components integrated in the test.

Figure 7: Mockito logo

4.1 Operational Testing Environment

Finally we are going to give a brief list of the equipment require in the testing enviroment:

- **Java EE** As described in the design document.
- **GlassFish Server Open Source Edition**
- **Eclipse (or Netbeans)** The IDE for source development.
- **Database** We expect a dedicated machine will run oracle DBMS with an appropriate licence in order to execute local communication testing with the database.
- **Client Browser** For web-application testing. We suggest: Google Chrome, Mozilla Firefox and Safari.
- **Mobile Client** For mobile app client testing: Android, iOS and Windows Phone.

5 Program Stubs and Test Data Required

In order to verify correctness of integration test we should use some test data. These data will be samples created by a script. e.g. cars, users, field staff positions. These data will build a coherent working simulation of which we know the solution.

E.g. for a relocation request we know a priori the field staff user that will be exchanged of this work so if the system assign the job to the expected person, we can assume that test is passed. All these data allow to have a sort of stubs (created using Mockito internally at the same component that allow to simulate all features not already implemented.

The mocks required by first priority test cases are:

I1-T1	for BPM (relocation code) and Map Manager (user search code)
I1-T2	for BPM (relocation code) and Rent manager (payment, car and user code)
I2-T1	for Rent manager (payment and car code) an PowerEnJoyUser manager (*)
I2-T2	for Rent manager (car code) an Payment manager (account suspension code)
I3-T1	for Reservation manager (payment code) and PowerEnJoyUserManager (*)
I3-T2	for Payment manager (user suspension code)
I4-T2	for PowerEnJoyUser manager (*) and Service configurator (configuration code)
I4-T3	for Account manager (register, login code)
I5-T1	for Account manager (register, login code)

The mocks required by second priority test cases are:

I7-T1	for Car manager (remote control code) and Emergency manager (*)
I7-T2	for FieldStaff manager (app interface code)
I8-T1	for EmergencyStaff manager (task assignment code)
I8-T2	for Emergency manager (* code)
I9-T1	for Emergency manager (*) and FieldStaff manager (app interface code)
I10-T1	for Car app (position update code)

The mocks required by third priority test cases are:

I12-T1	for ManagementStaff manager (app interface code) and Service configurator (prices / discounts / fees DSL code)

NOTE: External app manager does not require any stub nor mock since it is the last feature to be implemented, thus all the interacting components are already have been implemented.

6 Effort Spent

- Alessandro Erba ≈ 19h
- Filippo Leveni ≈ 17h
- Luca Lodi ≈ 15h