***PowerEnJoy***



**Integration Test Plan Document**

Authors:

Emanuele Chilà

Giorgio Lazzarinetti

January 11th 2017

SUMMARY

[1. INTRODUCTION 3](#_Toc471582782)

[1.1. PURPOSE AND SCOPE 3](#_Toc471582783)

[1.2. LIST OF DEFINITION AND ABBREVIATIONS 4](#_Toc471582784)

[1.3. LIST OF REFERENCE DOCUMENT 4](#_Toc471582785)

[2. INTEGRATION STRATEGY 4](#_Toc471582786)

[2.1. ENTRY CRITERIA 4](#_Toc471582787)

[2.2. ELEMENTS TO BE INTEGRATED 5](#_Toc471582788)

[2.3. INTEGRATION TESTING STRATEGY 6](#_Toc471582789)

[2.4. SEQUENCE OF COMPONENT/FUNCTION INTEGRATION 7](#_Toc471582790)

[3. INDIVIDUAL STEPS AND TEST DESCRIPTION 13](#_Toc471582791)

[3.1. INTEGRATION TEST CASES 13](#_Toc471582792)

[3.2. TEST PROCEDURES 19](#_Toc471582793)

[4. TOOLS AND TEST EQUIPMENT REQUIRED 21](#_Toc471582794)

[4.1. TOOLS 21](#_Toc471582795)

[4.2. TEST EQUIPMENT 23](#_Toc471582796)

[5. PROGRAM STUBS AND TEST DATA REQUIRED 24](#_Toc471582797)

[5.1. PROGRAM STUBS AND DRIVERS 24](#_Toc471582798)

[5.2. TEST DATA 26](#_Toc471582799)

[6. SUSPENSION, RESTART, AND EXIT CRITERIA 28](#_Toc471582800)

[7. EFFORT SPENT 29](#_Toc471582801)

[8. REFERENCES 30](#_Toc471582802)

## **INTRODUCTION**

### **PURPOSE AND SCOPE**

The purpose of the integration test plan is to describe the necessary test to verify that all the components of PowerEnJoy system are properly assembled. Integration testing ensures that the unit-tested modules interact correctly with no unexpected behaviour.

Power Enjoy allows customers to use a mobile device or a PC to check and then rent an electric car to use for a maximum time of 24h. Customer send information to this system with their mobile device or with their PC, then the system returns other information related to the available car or to the rental activated over a single car.

The Power Enjoy system provide several input methods through proper interfaces, described in the RASD and DD document provided. The aim of this document is to test the interfaces between components.

### **LIST OF DEFINITION AND ABBREVIATIONS**

* **RASD**: requirement analysis and specification document;
* **DD**: design document;
* **ITPD**: integration test plan document;
* **API**: application programming interface, common way to communicate with another system (for example we use this kind of interface to communicate with the Google map services);
* **Car**: we mean, of course, an electric car;
* **Reservation**: It’s a booking made by a user and paid to the use of a certain selected car;
* **Safe Area**: We mean the areas where the user can park the rented car;
* **Special Parking Area**: We mean the areas where the user can both park the rented car and recharge it (all the special parking area are safe area, but not vice versa);
* **Extraordinary situation**: We mean extraordinary events that can happen to the users such as car accident, car’s battery running low etc;
* **Driver:** testing module to perform method invocation of integrated component and so allow the testing of those component;

### **LIST OF REFERENCE DOCUMENT**

* DD produced before 3.0 version
* RASD produced before 2.0 version
* Specification Document: Assignment 1,2 and 3 (RASD, DD and ITPD).pdf

## **INTEGRATION STRATEGY**

### **ENTRY CRITERIA**

Before proceeding with the Integration Test there are some conditions that have to be fulfilled, this in order to produced coherent result. Firstly the RASD and the DD have to be completed as they provide the requirements, the structure of the system and the way the modules interact each other. Before starting the Integration Test, the percentage of the components which must be complete with respect to the functionalities developed and correctly unit tested is:

* **80%** Request Manager, Car proxy, Manager Proxy;
* **50%** Management Application’s components;
* **50%** Car Application’s components;
* **50%** Customer Application’s components;
* **50%** Call Centre Application’s components;

This percentages refer on the status of the project at the beginning of the integration testing phase, the Request Manager, the Car Proxy and the Manager Proxy must be complete at 80% since they are the components that connect all the other and without them we cannot run any of the system integration tasks.

We consider that the remaining components must be completed almost at the 50% in order to let the integration be effective and sensible.

So, the minimum completion percentage in order to consider a component for the integration should be at least 50%, except for the Request Manager, the Car Proxy and the Manager Proxy.

### **ELEMENTS TO BE INTEGRATED**

We are now going to provide the components that are integrated in the system. These are derived from the DD, in which we described the system as composed of different components according to the requirements to be satisfied described in the RASD. The components are the following:

**Customer Application:**

* Form Manager;
* Profile Manager;
* CustomerMessage Manager;

**Server:**

* Request Manager;
* DBQuery Manager;
* Data Model;

**Car Application:**

* Car Manager;
* Car Monitoring;
* Car (component);
* Car Proxy;

**Management Application:**

* Registration Manager;
* Payment Manager;
* Authentication Manager;
* Reservation Manager;
* Car Search Manager;
* Ride Manager;
* Car Listener;

**Call Centre Application:**

* Functionality provider;

**User client**

**Operator client**

**DBMS (this is a commercial component that has already been developed and can be immediately used in a bottom up approach)**

### **INTEGRATION TESTING STRATEGY**

For the integration testing we use the Bottom-up approach. This means to start the aggregation and testing from the leaves of the uses hierarchy. It requires the implementation of drivers to substitute for higher level modules. The purpose of this approach is to better follow the development process in order maximize the parallelism. During the Bottom-up integration approach it will still be given more importance to the leaves that are considered more critical (resulting in an higher priority on the assignment of resources if these will not be sufficient to achieve all tasks in parallel), following so a critical-module-first criteria on the components of the same level of priority defined by the Bottom-up approach.

As already mentioned the DBMS is a commercial component that has already been developed and that can be used immediately with a simple set up. The Car component too is an element that considers all the sensors and the monitor of a car and so they have already been developed and they do not have to be integrated with the system as they present theyr own tested API to the Car Proxy: the integration between the Car Proxy and the car API has already been considered in the unit test.

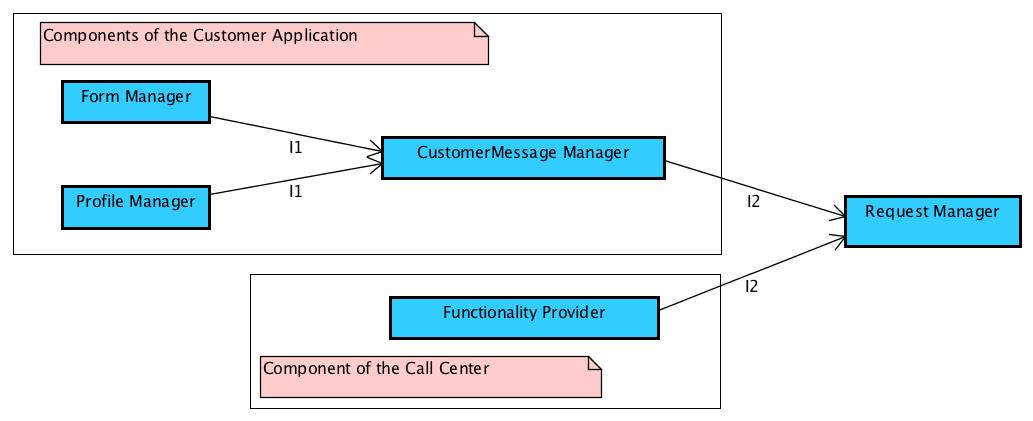
### **SEQUENCE OF COMPONENT/FUNCTION INTEGRATION**

In this section, we are going to identify the sequence in which the software components will be integrated within the subsystem. We refer to the component diagram presented in section 2.2.1 in the DD.

Using the Bottom-up approach we start integrating from the level of the uses hierarchy. We will see how to integrate the component of the Customer Application and of the Call Center Application, of the Car Application and of the Management Application separately, then we will see how to integrate these subsystems, already integrated, with the Server subsystem component. The figures below show the order of the integration, i.e. integration testing of the components.

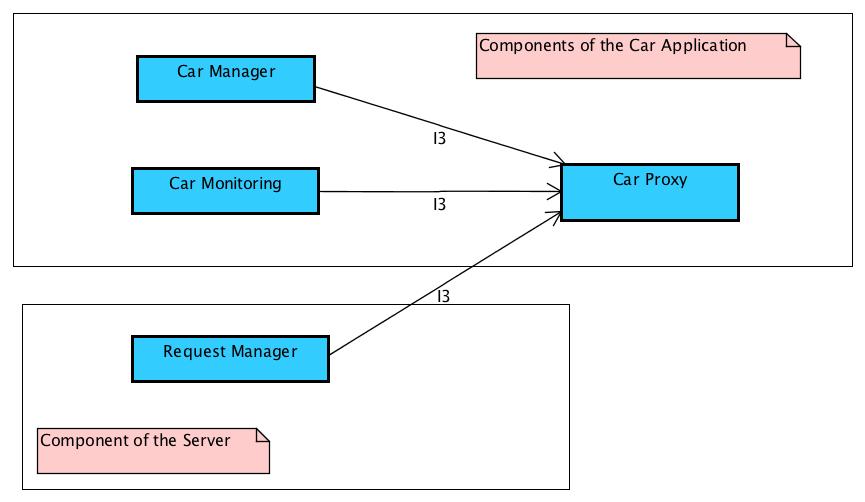
The first integration test we are going to present is the Integration test of the Customer Application. As we can see there are 2 tests to be perform to integrate che Customer Application’s components and the Call Center Application’s component. The first one is necessary to integrate the CustomerMessage manager. The second one is necessary to integrate the Request Manager that is part of the Server component. As we can see the Functionality provider, the Form Manager and the Profile Manager, whose functionalities are described in the DD paragraph 2.2.1, need to be unit tested in order to let the integration test be useful and successfull.

|  |  |
| --- | --- |
| Integration Test of the Customer and Call Center Application | |
| ID | TEST DESCRIPTION |
| I1 | Form Manager, Profile Manager -> CustomerMessage Manager |
| I2 | CustomerMessage Manager, Functionality Provider -> Request Manager |

****

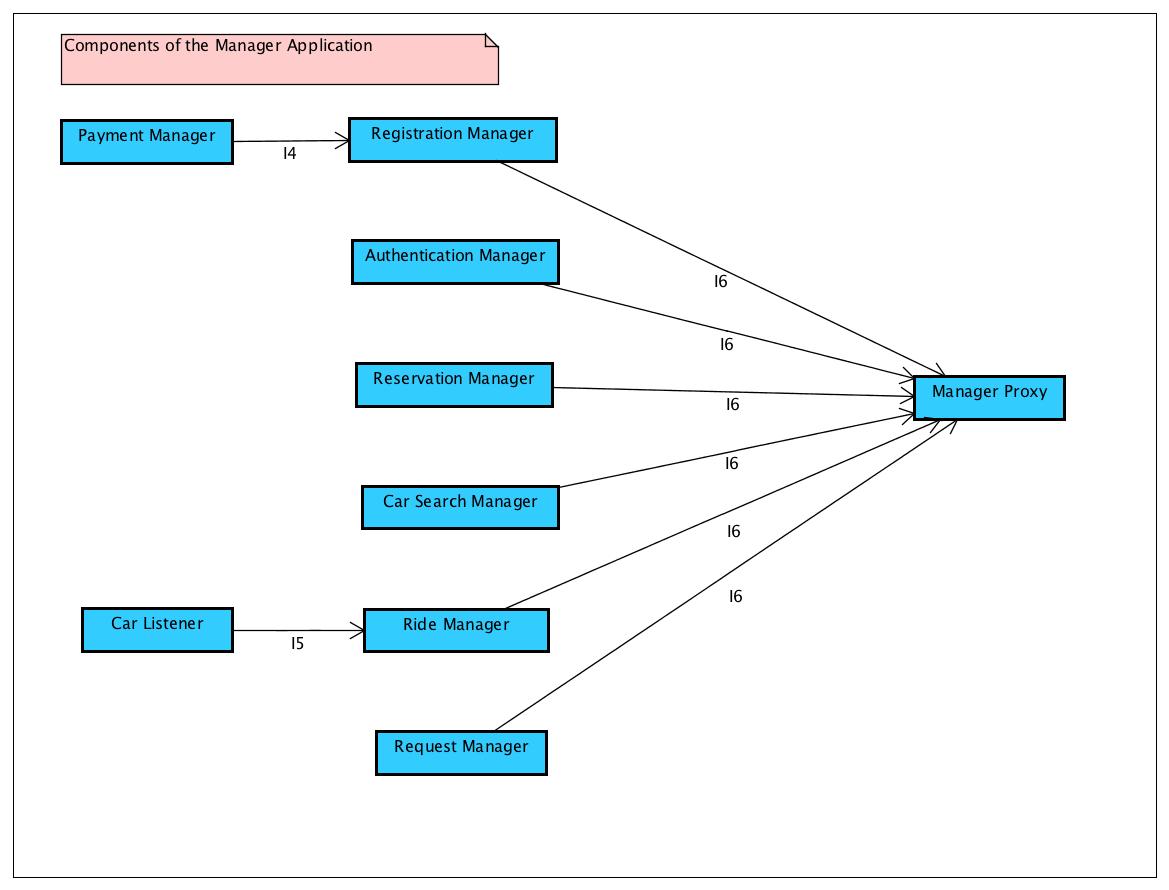
The Bottom-up approach goes on with the testing of the Car Application’s components. Here we have just a single test that includes the integration of all the component of the Car Appliction: Car Manager, Car Monitoring and the Request Manager from the Server’s components

|  |  |
| --- | --- |
| Integration Test of the Car Application | |
| ID | TEST DESCRIPTION |
| I3 | Car Manager, Car Monitoring, Request Manager -> Car Proxy |



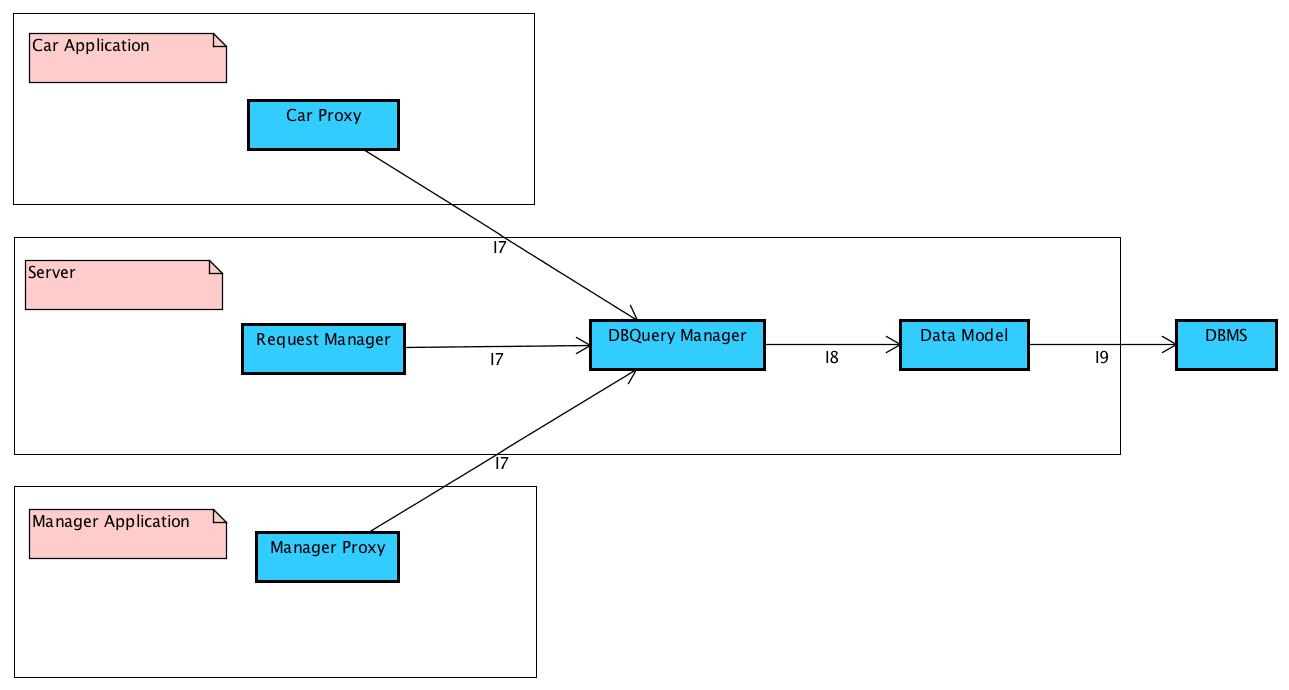
In Parallel with the integration of the Car Application’s component, the integration of the Manager Application’s components could be performed. As already mentioned following a critical-module first criteria on leaves at the same depth, to the Manager Application we should give a higher priority in case of lack of resources that doesn’t allow to perfectly follow the 2 integration tasks in parallel. Here we present the tests of the integration of this subsystem.

|  |  |
| --- | --- |
| Integration Test of the Manager Application | |
| ID | TEST DESCRIPTION |
| I4 | Payment Manager -> Registration Manager |
| I5 | Car Listener -> Ride Manager |
| I6 | Registration Manager, Authentication Manager, Reservation Manager Car Search Manager, Ride Manager, Request Manager -> Manager proxy |

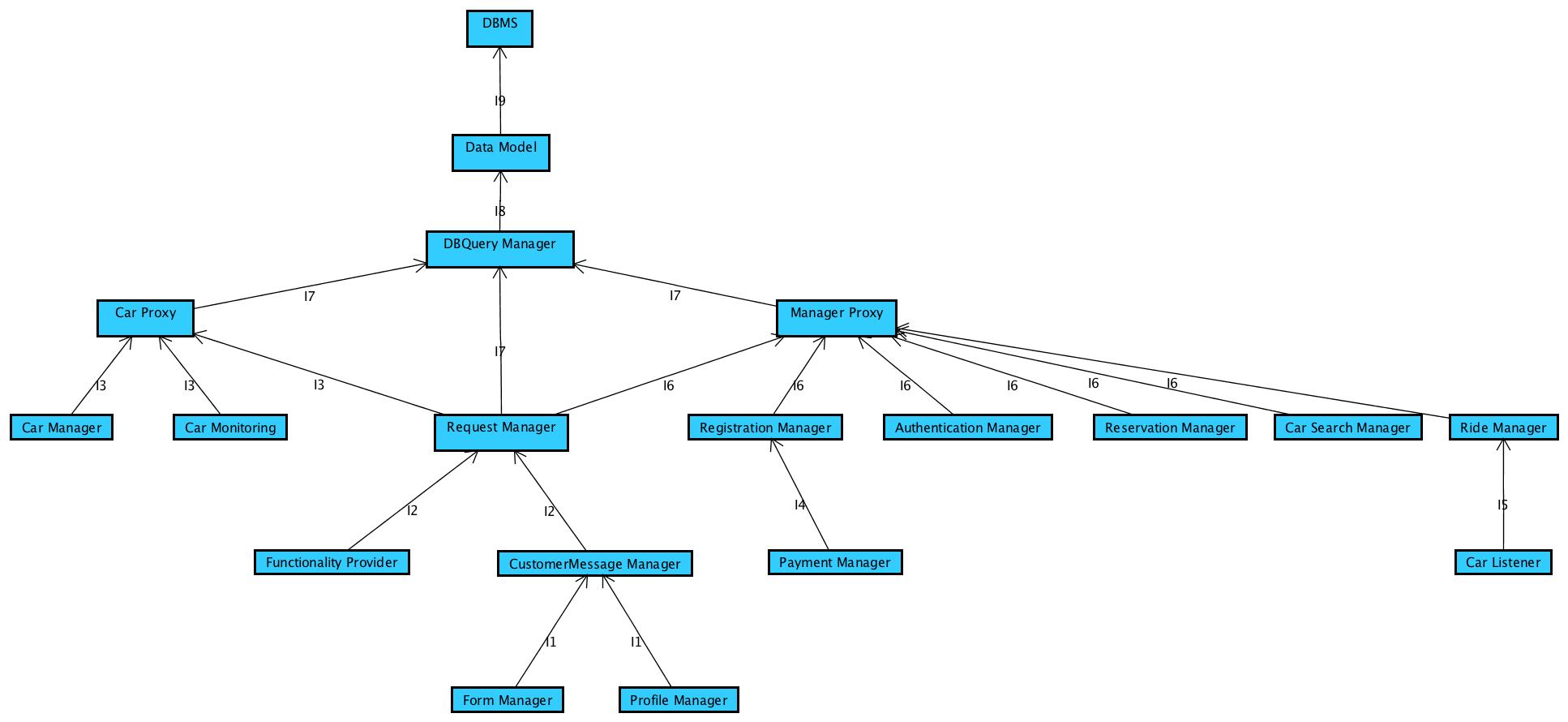


Finally, once the Car Proxy, the Manager Proxy and the Request Manager are tested, we can go on with the Bottom-up approach integrating the other part of the Server Component and of the DBMS. Here we present the lasts tests of the integration plan.

|  |  |
| --- | --- |
| Integration Test of the Server Component | |
| ID | TEST DESCRIPTION |
| I7 | Car Proxy, Request Manager, Manager Proxy -> DBQuery Manager |
| I8 | DBQuery Manager -> Data Model |
| I9 | Data Model -> DBMS |



In conclusion, we present the complete diagram of the integration test in order to better understand the different passages and the meaning of the fact that the Manager Application and the Car Application can be integrated in parallel.



## **INDIVIDUAL STEPS AND TEST DESCRIPTION**

In this section, we are going to describe, for each step of the integration process, the type of test that will be used to verify that the elements integrated in this step perform as expected. We will describe before the expected results of the test set in general and then some protocols to fulfil the goals of the tests in this section.

### **INTEGRATION TEST CASES**

* + 1. **INTEGRATION TEST CASE I1TI**

|  |  |
| --- | --- |
| **Test case identifier** | I1T1 |
| **Test items** | Form Manager->Costumer Message Manager |
| **Input specification** | Create typical Form Manager input |
| **Output specification** | Check if the correct functions are called in the Customer Message Manager |
| **Environmental needs** | Costumer Message Manager Driver |

* + 1. **INTEGRATION TEST CASE I1T2**

|  |  |
| --- | --- |
| **Test case identifier** | I1T2 |
| **Test items** | Profile Manager->Costumer Message Manager |
| **Input specification** | Create typical Profile Manager input |
| **Output specification** | Check if the correct functions are called in the Customer Message Manager |
| **Environmental needs** | Costumer Message Manager Driver |

* + 1. **INTEGRATION TEST CASE I2TI**

|  |  |
| --- | --- |
| **Test case identifier** | I2TI |
| **Test items** | Costumer Message Manager-> Request Manager |
| **Input specification** | Create typical Customer Message Manager input |
| **Output specification** | Check if the correct functions are called in the Request Manager |
| **Environmental needs** | I1 succeeded |

* + 1. **INTEGRATION TEST CASE I2T2**

|  |  |
| --- | --- |
| **Test case identifier** | I2T2 |
| **Test items** | Functionality Provider-> Request Manager |
| **Input specification** | Create typical Customer Functionality Provider input |
| **Output specification** | Check if the correct functions are called in the Request Manager |
| **Environmental needs** | Request Manager Driver |

* + 1. **INTEGRATION TEST CASE I3TI**

|  |  |
| --- | --- |
| **Test case identifier** | I3T1 |
| **Test items** | Car Manager-> Car Proxy |
| **Input specification** | Create typical Car Manager input |
| **Output specification** | Check if the correct functions are called in the Car Proxy |
| **Environmental needs** | Car Proxy Driver |

* + 1. **INTEGRATION TEST CASE I3T2**

|  |  |
| --- | --- |
| **Test case identifier** | I3T2 |
| **Test items** | Car Monitoring-> Car Proxy |
| **Input specification** | Create typical Car Monitoring input |
| **Output specification** | Check if the correct functions are called in the Car Proxy |
| **Environmental needs** | Car Proxy Driver |

* + 1. **INTEGRATION TEST CASE I3T3**

|  |  |
| --- | --- |
| **Test case identifier** | I3T3 |
| **Test items** | Request Manager-> Car Proxy |
| **Input specification** | Create typical Request Manager input |
| **Output specification** | Check if the correct functions are called in the Car Proxy |
| **Environmental needs** | I2 succeeded |

* + 1. **INTEGRATION TEST CASE I4**

|  |  |
| --- | --- |
| **Test case identifier** | I4 |
| **Test items** | Payment Manager->Registration Manager |
| **Input specification** | Create typical Payment Manager input |
| **Output specification** | Check if the correct functions are called in the Registration Manager |
| **Environmental needs** | Registration Manager Driver |

* + 1. **INTEGRATION TEST CASE I5**

|  |  |
| --- | --- |
| **Test case identifier** | I5 |
| **Test items** | Car Listener->Ride Manager |
| **Input specification** | Create typical Car Listener input |
| **Output specification** | Check if the correct functions are called in the Ride Manager |
| **Environmental needs** | Ride Manager Driver |

* + 1. **INTEGRATION TEST CASE I6TI**

|  |  |
| --- | --- |
| **Test case identifier** | I6T1 |
| **Test items** | Registration Manager-> Manager Proxy |
| **Input specification** | Create typical Registration Manager input |
| **Output specification** | Check if the correct functions are called in the Manager Proxy |
| **Environmental needs** | I4 succeeded |

* + 1. **INTEGRATION TEST CASE I6T2**

|  |  |
| --- | --- |
| **Test case identifier** | I6T2 |
| **Test items** | Authentication Manager-> Manager Proxy |
| **Input specification** | Create typical Authentication Manager input |
| **Output specification** | Check if the correct functions are called in the Manager Proxy |
| **Environmental needs** | Manager Proxy Driver |

* + 1. **INTEGRATION TEST CASE I6T3**

|  |  |
| --- | --- |
| **Test case identifier** | I6T3 |
| **Test items** | ReservationManager-> Manager Proxy |
| **Input specification** | Create typical Reservation Manager input |
| **Output specification** | Check if the correct functions are called in the Manager Proxy |
| **Environmental needs** | Manager Proxy Driver |

* + 1. **INTEGRATION TEST CASE I6T4**

|  |  |
| --- | --- |
| **Test case identifier** | I6T4 |
| **Test items** | Car Search Manager-> Manager Proxy |
| **Input specification** | Create typical Car Search Manager input |
| **Output specification** | Check if the correct functions are called in the Manager Proxy |
| **Environmental needs** | Manager Proxy Driver |

* + 1. **INTEGRATION TEST CASE I6T5**

|  |  |
| --- | --- |
| **Test case identifier** | I6T5 |
| **Test items** | Ride Manager-> Manager Proxy |
| **Input specification** | Create typical Ride Manager input |
| **Output specification** | Check if the correct functions are called in the Manager Proxy |
| **Environmental needs** | I5 succeeded |

* + 1. **INTEGRATION TEST CASE I6T6**

|  |  |
| --- | --- |
| **Test case identifier** | I6T6 |
| **Test items** | Request Manager-> Manager Proxy |
| **Input specification** | Create typical RequestManager input |
| **Output specification** | Check if the correct functions are called in the Manager Proxy |
| **Environmental needs** | I2 succeeded |

* + 1. **INTEGRATION TEST CASE I7T1**

|  |  |
| --- | --- |
| **Test case identifier** | I7T1 |
| **Test items** | Car Proxy-> DBQuery Manager |
| **Input specification** | Create typical Car Proxy input |
| **Output specification** | Check if the correct functions are called in the DBQuery Manager |
| **Environmental needs** | I3 succeeded |

* + 1. **INTEGRATION TEST CASE I7T2**

|  |  |
| --- | --- |
| **Test case identifier** | I7T2 |
| **Test items** | Request Manager-> DBQuery Manager |
| **Input specification** | Create typical RequestManager input |
| **Output specification** | Check if the correct functions are called in the DBQueryManager |
| **Environmental needs** | I2 succeeded |

* + 1. **INTEGRATION TEST CASE I7T3**

|  |  |
| --- | --- |
| **Test case identifier** | I7T3 |
| **Test items** | Manager Proxy-> DBQuery Manager |
| **Input specification** | Create typical Manager Proxy input |
| **Output specification** | Check if the correct functions are called in the DBQuery Manager |
| **Environmental needs** | I6 succeeded |

* + 1. **INTEGRATION TEST CASE I8**

|  |  |
| --- | --- |
| **Test case identifier** | I8 |
| **Test items** | DBQuery Manager-> Data Model |
| **Input specification** | Create typical DBQuery Manager input |
| **Output specification** | Check if the correct functions are called in the Data Model |
| **Environmental needs** | I7 succeeded |

* + 1. **INTEGRATION TEST CASE I9**

|  |  |
| --- | --- |
| **Test case identifier** | I9 |
| **Test items** | Data Model-> DBMS |
| **Input specification** | Create typical Data Model input |
| **Output specification** | Check if the correct functions are called in the DBMS |
| **Environmental needs** | Data Model Driver |

### **TEST PROCEDURES**

* + 1. **INTEGRATION TEST PROCEDURE TP1**

|  |  |
| --- | --- |
| **Test procedure identifier** | TP1 |
| **Purpose** | This test procedure verifies whether the Customer Application:   * can handle command-line input; * can handle client input; * can output the different form to the client; * can output requested information to the client;   It verifies also whether the Call Center Application:   * can handle command-line input; * can handle operator input; * can output the requested information to the operator; |
| **Procedure Step** | Execute I2 after I1 |

* + 1. **INTEGRATION TEST PROCEDURE TP2**

|  |  |
| --- | --- |
| **Test procedure identifier** | TP2 |
| **Purpose** | This test procedure verifies whether the Car Application:   * can handle car input; * can handle Server input; * can output information to the Car Proxy; |
| **Procedure Step** | Execute I3 |

* + 1. **INTEGRATION TEST PROCEDURE TP3**

|  |  |
| --- | --- |
| **Test procedure identifier** | TP3 |
| **Purpose** | This test procedure verifies whether the ManagerApplication:   * can handle command-line input; * can handle Server input; * can output requested information to the Manager Proxy; |
| **Procedure Step** | Execute I6 after I4 and I5 |

* + 1. **INTEGRATION TEST PROCEDURE TP4**

|  |  |
| --- | --- |
| **Test procedure identifier** | TP4 |
| **Purpose** | This test procedure verifies whether the Server:   * can handle command-line input; * can handle Client Application input; * can handle Car Application input; * can handle Manager Application input; * can output the information to the DBMS; * can output requested information to the client; * can output requested information to the operator; * can output requested information to the Car Proxy * can output requested information to the Manager Proxy |
| **Procedure Step** | Execute I8-I9 after I7 |

## **TOOLS AND TEST EQUIPMENT REQUIRED**

### **TOOLS**

In order support the phase of integration on the various component on PowerEnJoy we are going to make usage of the following automated tools:

* **Arquillian integration testing framework for Java EE:** a container-oriented test framework specifically ideated to make integration test. It allows us to execute tests on a container and so verify the correct integration between a component and his environment. It also allows us to verify that the integrated component doesn’t have any unexpected behaviours.
* **JUnit:** Even if we are in the integration phase we could still need a unit test tool to verify some component that may expose unexpected behaviours only on this phase, and in general because it is safer to verify again the single unit after it has been integrated (check the correctness of the parameter passed and the object returned by the interaction between two integrated component). Another point in favour is that checking again the single component after the integration is not very expensive because the unit tests are already written and they just have to be rerun.
* **JMeter:** Even if the main function of this tool is to make load test and performance test on the complete system, they may be carried out tests on some subsystem in order to get preliminary information about this topic.

We will use this kind of tool to make a preliminary performance analysis on the customer application in particular on the Web Page that is going to run on a computer.

* We are going to use some specific tools for the mobile application that are going to be used alongside JMeter to make some performance test on the Customer Application specifically on the Web Page (both Android and iOS version obviously) to make sure that that the application can be used by a reasonable broad range of cell phone.

In more specific detail we are going to use:

* + **On Android**: The Memory Proﬁler, Memory Monitor and Allocation Tracker tools to monitor main memory usage; the Traceview Walkthrough to monitor method execution time and the Battery Proﬁler to monitor energy consumption.
  + **On iOS**: The full suite of performance analysis tools provided by the Xcode IDE. MallocDebug to ﬁnd memory leaks, Activity Monitor and BigTop to monitor the system statistic.

We define as a benchmark the following fact:

* **For iOS**: the application must be able to run on cell phone of generation at least above the iPhone4s.
* **For Android**: the application must be able to run on cell phone of generation above the Samsung Galaxy S2 (all kind of cellphone that use android and have equal or higher hardware features)

However, this benchmark could be reconsidered in a more advanced stage of the project and they must only be considered as points of reference as no specification was indicated in the requirement specification document (RASD) of the project regarding this topic.

We have to take into account that these automated testing tool still need some manual operations such as initial setup and and the definition of a fair and exhaustive set of testing data.

### **TEST EQUIPMENT**

All the integration testing activities have to be performed within a speciﬁc testing environment. We need the following devices in order to perform the various integration test for what concern the mobile area:

* At least one Android smartphone for each display size from 3” to 6” at steps of 1”, plus a Samsung Galaxy S2 to verify the fixed benchmark.
* At least one Android tablet for each display size from 7” to 12” at steps of 1”.
* At least one iOS smartphone from the iPhone 4 generation to the most recent one.
* At least one iOS tablet for each display size (iPad Pro, Mini, Air).
* An electric car of PowerEnJoy equipped with all the sensors present in the models available to the public.

These devices will be used to test both the native mobile applications (Web Page) and the mobile versions of the web applications.

We are going to test the desktop web application using a set of normal desktop/notebook computers.

## **PROGRAM STUBS AND TEST DATA REQUIRED**

### **PROGRAM STUBS AND DRIVERS**

As already mentioned in the Integration Testing Strategy (in point 2.3) we are going to adopt a bottom up approach for the integration and the testing of the component.

As a result of this choice we are going to need some drivers to perform the method invocation of the integrated components in order to test them.

Here follows the list of all drivers that will be developed to make the integration tests over all the integration phase.

* **Customer Message Manager Driver**: this testing module is responsible to check the correct interaction and integration between **Form Manager** and **Profile Manager** by calling the method exposed by the **Customer Message Manager** component.
* **Request Manager Driver**: this testing module is responsible to check the correct interaction and integration between **Functionality Provider** and **Customer Message Manager** by calling the method exposed by the **Request Manager** component.
* **Car Proxy Driver:** this testing module is responsible to check the correct interaction and integration between **Car Manager** and **Car Monitoring** by calling the method exposed by the **Car Proxy** component and sub-components.
* **Manager Proxy Driver:** this testing module is responsible to check the correct interaction and integration between **Registration Manager**, **Authentication Manager**, **Reservation Manager**, **Car Search Manager**, **Ride Manager** by calling the method exposed by the **Manager Proxy** component and sub-components.
* **DB Query Manager Driver:** this testing module is responsible to check the correct interaction and integration between **Car Proxy**, **Manager Proxy**, **Request Manager** by calling the method exposed by the **DB Query Manager Driver** component and sub-components.
* **Data Model Driver:** this testing module will invoke the methods exposed by the **DB Query Manager** component in order to test its interaction with the **DBMS**.
* **Registration Manager Driver:** this testing module will invoke the methods exposed by the **Payment Manager** component in order to test its interaction with the **Manager Proxy**.
* **Ride Manager Driver:** this testing module will invoke the methods exposed by the **Car Listener** component in order to test its interaction with the **Manager Proxy**.

Since we are using a bottom-up approach we shouldn’t have the need to introduce any stub, indeed to simulate and test the system’s core functionality in a quick and easy way we will use a stub that simulates the Customer Application. This stub has the only functionality to make requests to the Server component, and save in a file the answers it gets from it so that they can be carefully checked (in that way we can test in a quicker way the functionality exposed by the Server side without relying on all the longer procedures which may occur during the use of the real client).

### **TEST DATA**

In order to perform all the tests specified in the chapter 3 of this document we are going to need:

* A list of both valid and invalid candidate users profile to test the **Customer Message Manager** component. The set should contain instances exhibiting the following facts:
  + Null object;
  + Null fields;
  + User rental information;
  + User profile information;
  + Information about the data base’s status (e.g. available cars, safe areas position…)

By null object and null fields, we intend the call of any method for which are passed object null pointer or null parameters, the purpose of these kind of test is to see the component react well to the error, report it and is able to recover well.

* A list of both valid and invalid candidate requests made by the users to test the **Request Manager** component. The set should contain instances exhibiting the following facts:
  + Null object;
  + Null fields;
  + Set of request that imply the interaction between Functionality Provider and Customer Message Manager;
* A list of both valid and invalid car requests, actions and sensor triggered actions to test the **Car Proxy** component. The set should contain instances exhibiting the following facts:
  + Null object;
  + Null fields;
  + Show message on the car’s monitor;
  + All kind of sensor detection coming from the car;
* A list of both valid and invalid management request to test the **Manager Proxy** component. The set should contain instances exhibiting the following facts:
  + Null object;
  + Null fields;
  + Lock or unlock command on a car that is not rented;
  + Search for available car that doesn’t find any car available;
  + Reservation request by a user with an unsolved payment;
  + Bad credential to make a login;
  + Login to a user profile already logged;
  + Payment information;
* A list of both valid and invalid request made by the Car Proxy, Request Manager and Manager Proxy to test the **DB Query Manager** component. The set should contain instances exhibiting the following problem:
  + Null object;
  + Null fields;
  + Set of request that imply the interaction between Car Proxy, Request Manager and Manager Proxy;
* A list of both valid and invalid access action to the data model structure, to test the **Data Model** component. The set should contain instances exhibiting the following problem:
  + Null object;
  + Null fields;
  + Access to a not existing field;
  + Access to a not existing element;
* A list of both valid and invalid set of user’s credentials, payments credentials (credit card) to test the **Registration Manager** component. The set should contain instances exhibiting the following problem:
  + Null object;
  + Null fields;
  + Invalid Credit Card Credential;
  + Invalid email address;
  + Invalid mobile phone number;
* A list of both valid and invalid lock and unlock command made with different parameters concerning the situation in which they are called to test the **Ride Manager** component. The set should contain instances exhibiting the following problem:
  + Null object;
  + Null fields;
  + Lock car while car is already locked;
  + Unlock car while car is already unlocked;
  + Lock command too far away from the vehicle;
  + Unlock command too far away from the vehicle;
  + Last Lock command but car not situated in a safe area ;

## **SUSPENSION, RESTART, AND EXIT CRITERIA**

In this section, we are going to define the criteria for suspending/restarting the test before the completion and the criteria to determine the completion of the integration tests.

* **Suspension Criteria**: If for a given input the integrated subsystem doesn’t have the correct behaviours the testing is suspended. If any of the functionalities provided by the integrated components do not work, testing is suspended. System downtime or environment downtime. In both cases the problem is passed to the development team.
* **Restart Criteria**: Once the development team has resolved the problem the tests team tests again the functionalities that failed and if they work well then, the testing process can restart from the point it was stopped.

This is also known as the “**Test Stop Criteria**” and it has to be taken in account that the testing process should be stopped trying to group up minor error and asap if the error is a critical one.

If there are more testing task in parallel the partial or complete suspension depends on the severity of the incident or causes in terms of effort, project time lines, rework and waste of labour and it cannot be defined with precision at this stage of the project.

* **Exit Criteria**: Once all the system has been integrated and all the integration tests have been run successfully the integration testing can conclude.

## **EFFORT SPENT**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time spent [hours]** | |  |
| **Activities** | **Emanuele Chilà** | **Giorgio Lazzarinetti** | **Total** |
| Introductions | 2 | 2 | 4 |
| Integration Strategy | 1 | 4 | 5 |
| Individual steps and test description | 1 | 4 | 5 |
| Tools and test equipment required | 3 | 1 | 4 |
| Program stubs and test data required | 4 | 2 | 6 |
| Suspension, Restart, and Exit Criteria | 3 | 1 | 4 |
| Overall document formatting | 1 | 1 | 2 |
|  | **Total ITPD** | | 30 |

## 

## 8**. REFERENCES**

The used tool used to create the ITPD are:

* Astah professional: to write UML diagrams;
* Microsoft Word 2011: to write the document;