Power EnJoy

Integration Test Plan Document

Version 1.0.0

Redaelli Marco 877622 Zanoli Francesco 877471

07/01/2017

Contents

Figure Contents

Chapter 1

Introduction

This section contains a brief introduction to the **I**ntegration **T**est **P**lan **D**ocument.

1.0.1 Purpose and Scope

This document is mainly based on the **D**esign **D**ocument. In fact the purpose of the **I**ntegration **T**est **P**lan **D**ocument is to clearly the order in which the software components identified in the **Component View** of the **DD** have to be integrated one with each other. It is also used to guarantee a well tested final software. Following the exposed procedure ensures that all the software components explained in the **DD** will communicate and cooperate in the proper way.

1.1 List of definitions and abbreviations

1.1.0.0.1 Definitions In the document are often used some specific terms whose definitions are reported bellow:

- Server Database: data layer
- Server Application: application layer
- Client: client layer
- Mobile App: *PowerEnJoy* mobile application, in Client
- Web App: *PowerEnJoy* web application, in Client
- System: the union of software and hardware to be developed and implemented
- Integration Test Case An atomic procedure done to test the integration of a component on the top of another one.
- Integration Test Suite A collection of **Integration Test Cases**.

- See the correspondent section in the **RASD** and the **DD** for more definitions.

1.1.0.0.2 Acronyms

- **RASD:** Requirements Analysis and Specification Document
- **DD:** Design Document
- **API:** Application Programming Interface
- **DBMS:** DataBase Management System
- **ITPD:** Integration Test Plan Document.
- **In:** Integration Test Suite number n.
- **InTm:** Integration Test Case number m of the Integration Test Suite number n.
- **JS:** JavaScript.
- **UI:** User Interface.
- See the correspondent section in the **RASD** and the **DD** for more acronyms and abbreviations.

1.2 List of reference documents

- Software Engineering 2 Project AA 2016/2017: Project Description And Rules and Assignment 4 - integration test plan
- PowerEnJoy's Requirement Analysis and Specification Document (RASD)
- PowerEnJoy's Design Document (DD)

Chapter 2

Integration Strategy

2.1 Entry Criteria

Before starting the integration testing of any software component that has been designed for myTaxiService system, the internal functions of the considered component (i.e. public or protected methods that are exposed within the package of the component but are not part of any external public interface) must be unit tested using an appropriate framework.

2.2 Elements to be integrated

PowerEnJoy as shown in the **DD** is a three-tier system composed by:

- **DBMS:** The memory of *PowerEnJoy* entire system
- **PowerEnJoy System:** The main server and the main core composed by:
    1. Data Manager
    2. Account Manager
    3. Ride Manager
    4. Bill Manager
    5. Map Services
    6. Notification
    7. Car Manager
    8. Zone Manager
    9. Problem Manager
- **Client Application:** Subdivided in **Car System** and **Mobile Application**

Moreover we assume that that Google Maps API and Paypal are well tested by their owner and thus we can use them without testing any further.

2.3 Integration testing strategy

The integration testing strategy, conducted in this project, is a **bottom-up** approach. This strategy tests the lower level components and start testing a way upwards to higher level components. The advantage of this strategy is that allow us to maintain the code easier, smaller modules have unit tests and there is a clearer structure of how to do things. The disadvantage is that when releasing a prototype it's impossible to see a working prototype until nearly all the program has been completed so that may take a long time before this happens. In early development, testing tools as Mockito and Arquillian (described in Chapter 4) allow us to test components which depend on incomplete ones through stubs and drivers (Chapter 5). The usage of the selected approach will create a robust application with efforts concentrated in testing the Server parts before all.

2.4 Sequence of component/function integration

2.4.1 Software integration sequence

The following diagram illustrates the integration sequence of the various components, following the integration testing strategy described above. This means that in each subsystem, components are integrated starting from the most independent to the less independent, in order to prompt the chosen approach and improving modularity. The colors chosen for the block are standardise and refer to the legend bellow:

- **Green:**
- **Yellow:**
- **Orange:**
- **Red:**

2.4.2 Subsystem integration sequence

The following diagram illustrates the integration sequence of the various subsystems, following the integration testing strategy described above. In particular, the Server Database is integrated before the Client, because the former does not need an actual functioning system in order to be tested efficiently, contrary to the latter.

Chapter 3

Individual steps and test description

Chapter 4

Tools and test equipment required

The following part of the document contains a set of recommended software that can be used to implement the concrete procedure of testing. Moreover, because the high-level architecture proposed in the **DD** is designed using a **Java-based** style, the programming language that better adapts to this style is **Java**, but a lot of other emerging languages can be used in order to build a proper software, like **Swift**.

If it is decided to use **Java**, the proposed and well-known tools are:

- **JUnit:** Unit testing framework.
  - http://junit.org/
- **Mockito:** Another unit testing framework.
  - http://site.mockito.org/
- **Arquillian:** Integration testing framework.
  - http://arquillian.org/
- **Espresso:** Android UI testing automation.
  - http://developer.android.com/training/testing/ui-testing/espresso-testing.html

Chapter 5

Program stubs and test data required

Appendix A

Appendix

A.1 Tools

- **TeXstudio:** LaTeX editor used to write the document.
- **StarUML:** Used to build UML Class Diagram and UML Use Case Diagrams.
- **draw.io WebSite:** Used to design the UML Sequence Diagram and the Mockup (wep portal and smartphone app).

A.2 Hours of work

In the following are listed the hours of work that each member of the group did:

1. Marco Redaelli: 41 *hours*
2. Francesco Zanoli: 41 *hours*

A.3 Version History

In the following are listed the differences between versions:

1. **11/12/2016:** First version