Politecnico di Milano

A.A 2016-2017

Integration Test Plan Document

Version 1.0

PowerEnjoy

Instructor : Prof. Di Nitto

Authors:
Amico Simone
Chianella Claudia Beatrice
Giovanakis Yannick

CONTENTS

1 Introduction **4**
- 1.1 Purpose and Scope . . . 4
- 1.2 Definition and Abbreviations . . . 4
- 1.3 Reference Documents . . . 5

2 Integration Strategy **6**
- 2.1 Entry Criteria . . . 6
- 2.2 Elements to be Integrated . . . 6
- 2.3 Integration Testing Strategy . . . 8
- 2.4 Integration Sequence . . . 8
  - 2.4.1 Back-End Tests . . . 10
  - 2.4.2 Client-Server Tests . . . 10

3 Individual Steps and Test Description **11**
- 3.1 Integration test case I1 . . . 11
- 3.2 Integration test case I2 . . . 11
- 3.3 Integration test case I3 . . . 11
- 3.4 Integration test case I4 . . . 11
- 3.5 Integration test case I5 . . . 11
- 3.6 Integration test case I6 . . . 11
- 3.7 Integration test case I7 . . . 11
- 3.8 Integration test case I8 . . . 11
- 3.9 Integration test case I9 . . . 11
- 3.10 Integration test case I10 . . . 11
- 3.11 Integration test case I11 . . . 11

4 Tool and Test Equipment Required **12**
- 4.1 Test Equipment . . . 12
- 4.2 Test Tools . . . 12
  - 4.2.1 Mockito . . . 12
  - 4.2.2 Arquillian . . . 12
  - 4.2.3 Manual Testing . . . 13

5 Program Stubs and Data Test Required **13**

6 Appendices **13**
6.1 References 13
6.2 Effort Spent 13

1. INTRODUCTION

1.1 Purpose and Scope

The purpose of the **Integration Test Plan** is to describe the necessary tests to verify that all of the components of the *PowerEnjoy* platform are properly assembled. Integration testing ensures that the unit-tested modules interact correctly.
The description of the testing process includes:

- A high level specific of the tests
- A testing strategy
- An overview of the testing tools

The document is aimed at stakeholders ,developers in charge of the testing implementation and engineers.
It is important to notice that the focus of the document lies essentially towards **integration** whereas **unit-tests** are ignored and considered as already conducted.

1.2 Definition and Abbreviations

Throughout the document the following *abbreviations* are used and not further explained:

- **RASD**: Requirements And Specifications Document
- **DD**: Design Document
- **ITPD**: Integration Test Plan Document
- **API**: Application Programming Interface
- **RESTful**:REpresentational State Transfer
- **DBMS**: Database Management System

Each **integration test** has a unique identifier that follows the syntax:

$$I[0-9]^{+}$$

Each **test case** has a unique identifier that follows the syntax:

$$I[0-9]^{+}T[0-9]^{+}$$

1.3 Reference Documents

For a full understanding of the content of the ITPD ,it is strongly advised to read the **RASD** and especially the **DD** as they contain more in-depth explanations for the majority of the subjects.
A complete overview about documents and the general system description can be found int the **Assignments AA 2016-2017.pdf** file.

2. INTEGRATION STRATEGY

2.1 Entry Criteria

The **Integration tests** are meant to be developed and conducted only after **single units** have been successfully and thoroughly tested ,with particular regard towards those parts involving intermodule communication.

2.2 Elements to be Integrated

From what we can infer from the previous documents , the to-be tested platform used the **client-server paradigm** as its main architecture with the addition of intra module communication , especially in the **back-end system** where the *business logic* lies, and direct communication happening back and forth on separate channels between the **back-end** and the **client-side applications.**
The following components described in *section 2* of the *Design Document* need to be tested:

- **Server Components**:
  - Ride Manager
  - Notification Manager
  - User Manager
  - Search Manager
  - Position Manager
  - Database Interface
- **Mobile & Web Application**
  - SignIn Action
  - SignUp Action
  - Main Action

- **On-Board Application**
  - SignIn Action
  - Navigation Action
  - EndRide Action

The client-side applications communicate with the *back-end system* through the **RESTful API** and the **Notification Manager**. To simplify the test planning, from now on *Mobile & Web components* and *On-Board components* are grouped together in a single entity called **client-side components**.With this considerations in mind , integration test need to be performed on the following pairs:

- Request Manager → Search Manager
- Request Manager → Ride Manager
- Request Manager → User Manager
- Search Manager → Position Manager
- Position Manager → Vehicle Manager
- Database Interface → DBMS
- RESTful API → Request Manager
- Ride Manager → Notification Manager
- Search Manager → Notification Manager
- Notification Manager → Client-side components
- Client-side components → RESTful API
- **STATION MANAGER!!**

2.3 Integration Testing Strategy

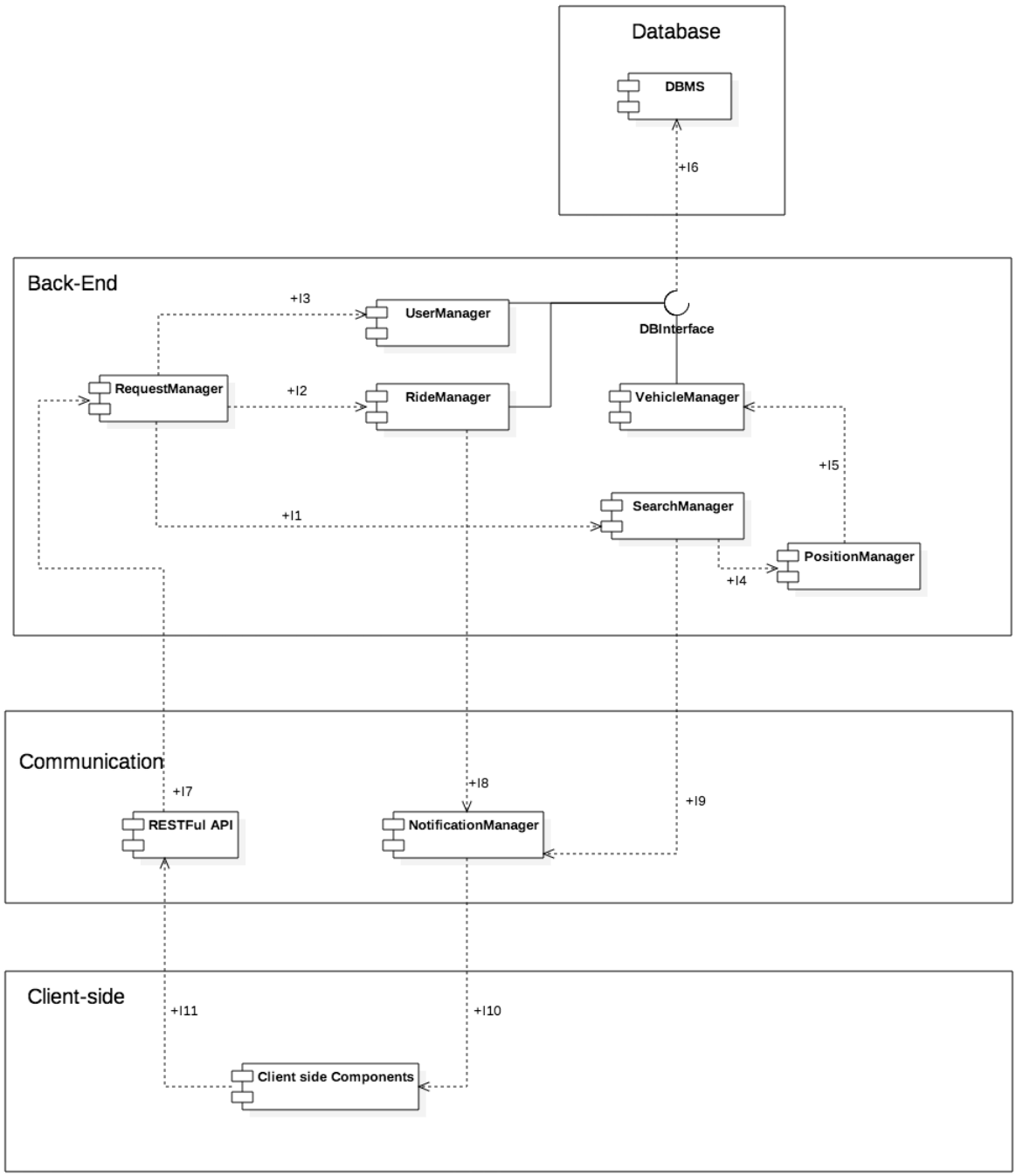
The *PowerEnjoy System* is composed of many components which are subject to a lot of interactions : the system , as already shown in the Design Document, is thus quiet **complex**. Structural testing strategies , such as top-down or bottom-up, are *simpler* whereas more complex strategies provide better process visibility in cases like ours.
The strategy we adopted is the **functional grouping strategy** , a highly *modular* strategy that allows the *separate* development of the various parts of the system.
Moreover the integration test should be performed mostly on *actual code* in order to reduce the number of *stubs and mocks* and reduce the use of *dummy code*.

2.4 Integration Sequence

The **Integration Test** is meant to be performed in the following order in compliance to the aforementioned strategy:

1. Test and integrate back-end modules
2. Test and integrate client-side components
3. Test and integrate the back-end and clients with the dedicated communication modules

Figure 1: Integration Test Diagram



The following tables show the needed integration tests,highlighting their ID used in this document and the paragraph that describes them.

2.4.1 Back-End Tests

ID	Integration Test	Paragraphs
1	Request Manager → Search Manager	3.1
2	Request Manager → Ride Manager	3.2
3	Request Manager → User Manager	3.3
4	Search Manager → Position Manager	3.4
5	Position Manager → Vehicle Manager	3.5
6	Database Interface → DBMS	3.6

Table 1: Back-End Tests Table

2.4.2 Client-Server Tests

ID	Integration Test	Paragraphs
7	RESTful API → Request Manager	3.7
8	Ride Manager → Notification Manager	3.8
9	Search Manager → Notification Manager	3.9
10	Notification Manager → Client-side components	3.10
11	Client-side components → RESTful API	3.11

Table 2: Client-Server Tests Table

3. INDIVIDUAL STEPS AND TEST DESCRIPTION

For each of the integrations tests mentioned in section 2.4, one or more detailed test cases are defined in the following tables.

3.1 Integration test case I1

3.2 Integration test case I2

3.3 Integration test case I3

3.4 Integration test case I4

3.5 Integration test case I5

3.6 Integration test case I6

3.7 Integration test case I7

3.8 Integration test case I8

3.9 Integration test case I9

3.10 Integration test case I10

3.11 Integration test case I11

4. TOOL AND TEST EQUIPMENT REQUIRED

This section addresses the tools that the testing team should make use of to execute the integrations tests mentioned in section 3.

4.1 Test Equipment

The final deployment of the platform will happen on separate physical machines , as different client-devices need to communicate with the back-end application as stated in DD section 2.4.
In order to correctly test this scenario , during *integration testing* the team must use at least a dedicated **server** running the **back-end application**.

4.2 Test Tools

4.2.1 Mockito

Mockito is a **test framework** usually used to cut out the dependencies during unit testing.
It can nevertheless be used to aid the developers during integration , especially during the early stages of development when it may become useful to mock some **not-yet developed** components in order to perform some initial integration testing.

4.2.2 Arquillian

Arquillian is a **test framework** used to perform testing inside a remote or embedded container, or deploy an archive to a container so the test can interact as a remote client.
Arquillian integrates with other testing frameworks, like **JUnit**, allowing the use of **IDE** and **Maven plugins** and facilitating the developers work.
The use of Arquillian is especially recommended during the testing of **Client-Server** interaction.

4.2.3 Manual Testing

Manual tests should always be kept as a viable alternative,especially when testing **non-standard** implementations of algorithms and interfaces. The testing team should resort to manual testing whenever the aforementioned tools should not be expressive enough to test *specific* aspects of the integrations.

5. Program Stubs and Data Test Required

6. Appendices

6.1 References

The following tools where used in the creation of this document:

- *TexMaker 4.5* as Editor
- *StarUML* for Diagrams

6.2 Effort Spent

- Simone Amico h
- Chianella Claudia Beatrice h
- Giovanakis Yannick h