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

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
  - Charging Manager
  - Station Manager
  - Database Interface
- **Client-side Components**
  - All main actions performed by the client side application

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
- Request Manager → Charging Manager
- Search Manager → Position Manager
- Position Manager → Vehicle Manager
- Charging Manager → Station 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

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

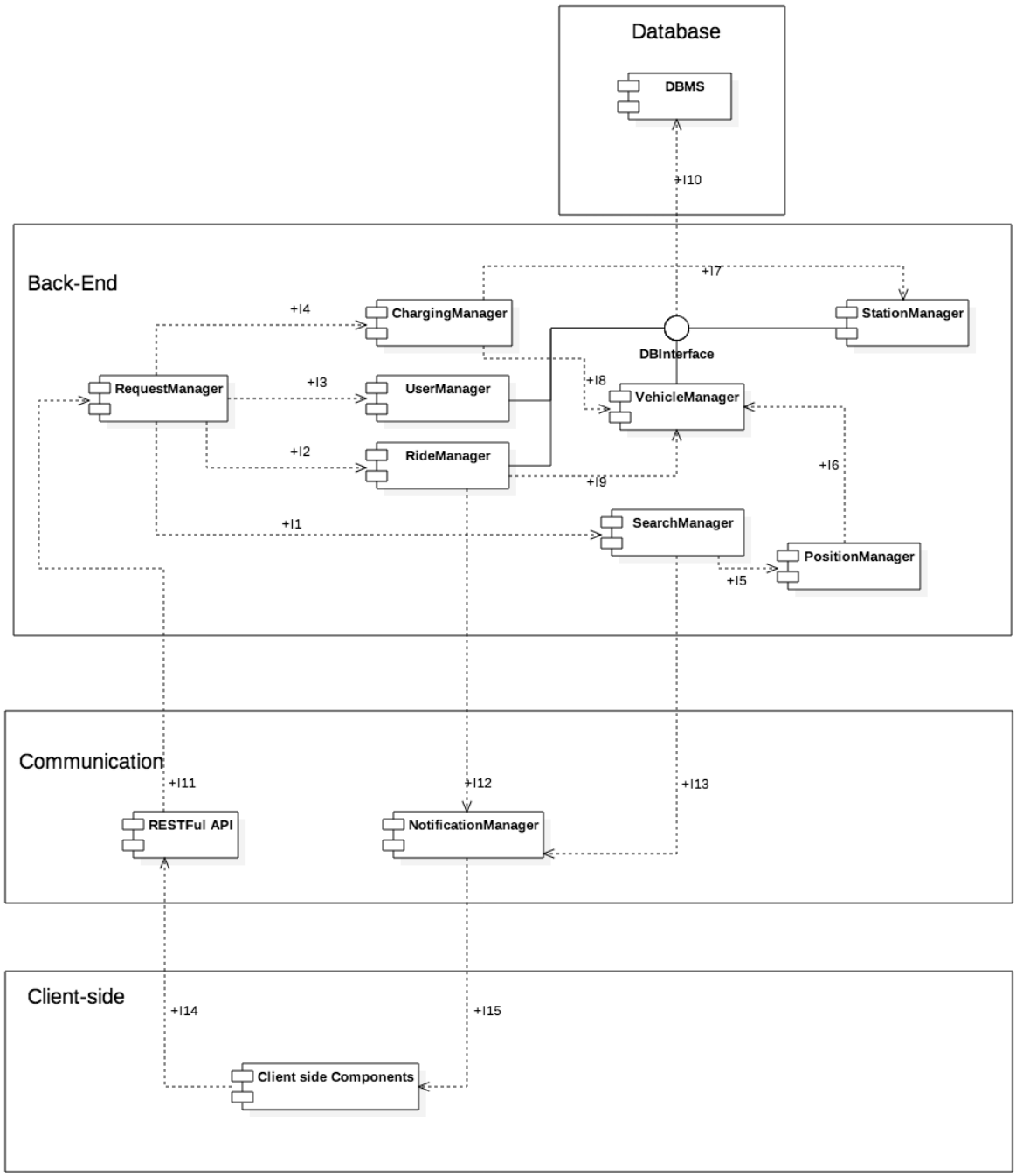
of the system.
Moreover the integration test should be performed mostly on *actual code* in order to reduce the number of *stubs and mocks* and reduce the use of *dummy code*.

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	Request Manager → Charging Manager	3.4
5	Search Manager → Position Manager	3.5
6	Position Manager → Vehicle Manager	3.6
7	Charging Manager → Station Manager	3.7
8	Charging Manager → Vehicle Manager	3.8
9	Ride Manager → Vehicle Manager	3.9
10	Database Interface → DBMS	3.10

Table 1: Back-End Tests Table

2.4.2 Client-Server Tests

ID	Integration Test	Paragraphs
11	RESTful API → Request Manager	3.11
12	Ride Manager → Notification Manager	3.12
13	Search Manager → Notification Manager	3.13
14	Notification Manager → Client-side components	3.14
15	Client-side components → RESTful API	3.15

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

Test Case ID	I1T1
Test Item	Request Manager → Search Manager
Input Specification	Create a Request Manager that provides a Request to be managed by the Search Manager
Output Specification	Check that the correct methods are called and that the received object by the Search Manager is consistent.
Environmental Needs	A running Server

3.2 Integration test case I2

Test Case ID	I2T1
Test Item	Request Manager → Ride Manager
Input Specification	Create a Request Manager that provides a Request to be managed by the Ride Manager
Output Specification	Check that the correct methods are called and that the received object by the Ride Manager is consistent.
Environmental Needs	A running Server

3.3 Integration test case I3

Test Case ID	I3T1
Test Item	Request Manager → User Manager
Input Specification	Create a Request Manager that provides a request to be managed by the User Manager
Output Specification	Check that the correct methods are called and that the received object by the User Manager is consistent.
Environmental Needs	A running Server

3.4 Integration test case I4

Test Case ID	I4T1
Test Item	Request Manager → Charging Manager
Input Specification	Create a Request Manager that provides a request to be managed by the Charging Manager
Output Specification	Check that the correct methods are called and that the received object by the Charging Manager is consistent.
Environmental Needs	A running Server

3.5 Integration test case I5

Test Case ID	I5T1
Test Item	Search Manager → Position Manager
Input Specification	Correctly create a Search Request with valid parameters to query the Position Manager
Output Specification	Check that the correct methods are called and that the received object by the Position Manager is consistent.
Environmental Needs	A running Server and Test Data

3.6 Integration test case I6

Test Case ID	I6T1
Test Item	Position Manager → Vehicle Manager
Input Specification	Correctly create test vehicle data with relative vehicle managers.
Output Specification	Correctly gather all relevant data from all vehicle managers inside the Position Manager
Environmental Needs	A running Server and Test Data

3.7 Integration test case I7

Test Case ID	I7T1
Test Item	Charging Manager → Station Manager
Input Specification	Correctly create test stations with its Station managers.
Output Specification	Look for the right station according to the data in the request. Check if the station has an empty spot and if so retrieve the empty position .
Environmental Needs	A running Server and Test Data

3.8 Integration test case I8

Test Case ID	I8T1
Test Item	Charging Manager → Vehicle Manager
Input Specification	Correctly create test vehicle data with relative vehicle manager.
Output Specification	Add the car data to the station and change car status
Environmental Needs	A running Server and Test Data

3.9 Integration test case I9

Test Case ID	I9T1
Test Item	Ride Manager → Vehicle Manager
Input Specification	Correctly create test vehicle data with relative vehicle managers along with a Ride object and its Ride Manager
Output Specification	Correctly update vehicle information according to the data in the Ride Manager
Environmental Needs	A running Server and Test Data

3.10 Integration test case I10

Test Case ID	I10T1
Test Item	Database Interface → DBMS
Input Specification	Typical SQL Query
Output Specification	Output is consistent with the query
Environmental Needs	A running Server , a running Database and Test Data

Test Case ID	I10T2
Test Item	Database Interface → DBMS
Input Specification	Typical SQL Query
Output Specification	Database tables are updated as expected
Environmental Needs	A running Server , a running Database and Test Data

3.11 Integration test case I11

Test Case ID	I11T1
Test Item	RESTful API → Request Manager
Input Specification	Create a RESTful API interface that redirects the request to the Request Manager
Output Specification	Check that the forwarded request is consistent
Environmental Needs	A running Server and Test Data

3.12 Integration test case I12

Test Case ID	I12T1
Test Item	Ride Manager → Notification Manager
Input Specification	Create a Ride Manager and Ride object along with a Client message to be send to the Notification Messanger
Output Specification	Check that the message is sent to the correct Client
Environmental Needs	A running server and client application

3.13 Integration test case I13

Test Case ID	I13T1
Test Item	Search Manager → Notification Manager
Input Specification	Create a Search Manager that handles some search data along with a Client message to be send to the Notification Messanger
Output Specification	Check that the message is sent to the correct Client
Environmental Needs	A running server and client application

3.14 Integration test case I14

Test Case ID	I14T1
Test Item	Notification Manager → Client-side application
Input Specification	Create a Notification manager object that sends a message to the client
Output Specification	Check that the notification is received by the client application
Environmental Needs	A running server and client application

3.15 Integration test case I15

Test Case ID	I15T1
Test Item	Client-side application → RESTFul API
Input Specification	Create a Mock Client Application that sends information and requests through RESTful API (Sign in ,Sign up, Manage user data)
Output Specification	Check that the correct methods are called and unexpected behavior is handled correctly
Environmental Needs	Mock client application and test data

Test Case ID	I15T2
Test Item	Client-side application → RESTFul API
Input Specification	Create a Mock Client Application that sends a Search Request through RESTful API
Output Specification	Check that the correct methods are called and unexpected behavior is handled correctly
Environmental Needs	Mock client application and test data

Test Case ID	I15T3
Test Item	Client-side application → RESTFul API
Input Specification	Create a Mock Client Application that sends a Ride Request through RESTful API
Output Specification	Check that the correct methods are called and unexpected behavior is handled correctly
Environmental Needs	Mock client application and test data

Test Case ID	I15T4
Test Item	Client-side application → RESTFul API
Input Specification	Create a Mock Client Application that sends a Unlock Request through RESTful API
Output Specification	Check that the correct methods are called and unexpected behavior is handled correctly
Environmental Needs	Mock client application and test data (test vehicle included)

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

5.1 Program Stubs

- If not already present, a test database consistent with a future implementation, should be created as to test read/write operations to and from a database.
- In the test suite I11 a **code stub** that simulates a Client's application's behaviour is required as to send valid and consistent HTTPS requests to the RESTful API.
- Code stubs must be used each time that involves a module which is not fully developed yet.

5.2 Test Data

- I1, I2 and I3 need a some **Request** to test different scenarios.
- I4 and I5 need a valid **search request** with valid parameters to be handled.
- I5 needs a valid **Search** request with correct parameters to be interpreted by the Position Manager.
- I6 needs some test vehicle data to be collected by the Position Manager.
- I7 and I8 need charging request data.
- I9 needs some Ride data to update the Vehicle Manager.
- I10 needs some **test objects** to input to the database through queries
- I11 needs a Request to be send to the Request Manager.
- I12 needs a Ride Object able to send out messages to a client mock.

- I13 needs Search data that must be interpreted by the Position manager and send out a response to the client mock through the Notification Manager.
- I14 needs a fictional message object to send to the mock client
- I15 needs different type of requests with consistent and proper data according to the type of request.

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 12h
- Chianella Claudia Beatrice 12 h
- Giovanakis Yannick 12 h