Manage Project Homework: MPH Server Testing Developer Guide

Riccardo Ancona - 782025
Alessandro Ditta - 781482

January 24, 2012

Contents

1 Introduction

1.1 Purpose

This testing developer guide describes how the testing for the MPH Server Module was done by the developer team.

1.2 Scope

The software system to be produced is a projects management tool which will be referred to as Manage Project Homework (MPH) throughout this document.

MPH will allow professors to publish a project description and to define the set of the corresponding deliverables. It will allow students to join project teams and submit deliverables by uploading them into the system.

The professor will also be able to evaluate the project deliverables assigning a score to them and leaving to the system the computation of the final score based on the average of the individual scores. MPH will also provide some information sharing functionalities among different teams.

1.3 Definitions, acronyms, and abbreviations

MPH: Manage Project Homework, the software system to be produced.

Team: a set of 1, 2 or 3 students who work together on the same project.

Admin: the system administrator

Deliverable: the model of a tangible object produced as a result of a specific phase of the project.

Artifact: the effectively tangible object associated to a deliverable.

Deadline: the date by which the artifact associated to a deliverable must be delivered.

RASD: Requirements Analysis and Specification Document

DD: Design Document

DBMS: DataBase Management System

GUI: Graphical User Interface

JEE: Java Enterprise Edition

1.4 References

- Description of the project: *http://corsi.metid.polimi.it*
- MPH Requirements Analysis and Specification Document
- MPH Design Document
- http://www.junit.org/

2 Getting Help and Giving Feedback

2.1 Do you need help?

If you experience difficulty with a procedure described in this documentation, please feel free to send an email to axxo@hotmail.it. Don't forget to include in the email all the data needed to get support, like a list of action performed and screenshot that shows the error.

2.2 We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit an email to axxo@hotmail.it. If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

3 Testing Activity

3.1 Introduction

All the tests were created and executed in the project "MPH-Server-Test": in this way, it was possible to test the server in a more easily way, without creating some extra adapter classes, since testing a JEE project from the JEE projects itself is in general different from others simple Java projects. The developers used the JUnit framework, even if it was not specifically created to test JEE Entities or Session Beans. Before trying to run the tests, you need to have a CLEAN installation of the MPH Server module, since some of the tests will introduce new data in it. To run a test, select the correspondent Junit class inside Eclipse and run JUnit on that class.

3.2 Unit Testing

The developers of the server module performed unit testing in order to test all the server modules singly. All the unit test use a black box method of testing, and are contained in the package "com.raaxxo.mph.testing.cases".

3.3 Integration Testing

The developer team performed this kind of test extensively throughout the development, from the early start. This tests suites creates some data in the database using all the shared methods and interfaces: in this way, the test is performed as there is some "dummy client" that performs operations and sends commands to the server. In this way, the functionalities of the server module could be tested even if there was no GUI client at all. In all the tests that were developed, the test criteria used was "edge coverage" from a functional point of view: there were created test suites for a specific functionality implemented. Each method inside a functionality was tested with a large number of possible executions, both the ones that executed successfully and the ones that failed and throwed an exception. All the integration testing are contained in the package "com.raaxxo.mph.testing.suites".

3.4 System Testing

The developer team performed this kind of test extensively throughout the development to test the GUI and the integration between server and client module. The system testing was done running and using the client module of the MPH software: the developer team decided to perform the test this way because the JUnit framework was unsuitable to test the GUI and to test deeply some functionalities of the server.