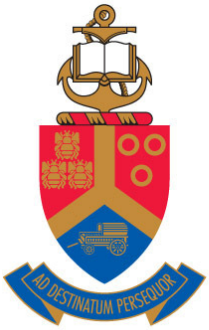

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

IMPAKD

Project COS 301 Main Project

Product Test Plan and Report

Version 1.2

Unit Test Plan and Report

Prepared by:

Sandile Khumalo - 12031748
Kudzai Muranga - 13278012
Diana Obo - 13134885
Priscilla Madigoe - 13049128

September 28, 2016

IMPAKD link

For further references see gitHub. September 28, 2016

Contents

1 Introduction

1.1 Purpose

The Property Investment Optimiser's purpose is to assist the user by optimising their investments in the rental property space. The purpose of this document is to list the tests we have put in place to ensure the functionality of the project. Test driven development was used to ensure that we spend less time debugging the code while creating a detailed specification of what functionality is required for the project.

1.2 Scope

This document will be used to elaborate and substantiate the unit tests conducted for the various use cases of the Property Investment Optimiser project. Unit test, with mock objects, will be used to ensure the correct functionality of each use case's implementation. These unit tests are done on the server side.

1.3 Test Environment

Junit was used to implement the tests for each test case. Also, EJBContainers were used to test functionality that requires JPA and Enterprise Java Beans.

Unit Test Plan

2 Test Items

3 Functional Features to be Tested

Feature ID	RDS Source	Summary	Test Case ID
1	Implementation.PIO.BackEnd.test.Test	Passed	01
2	Implementation.PIO.BackEnd.test.Test	Passed	02
3	Implementation.PIO.BackEnd.test.Test	Passed	03
4	Implementation.PIO.BackEnd.test.Test	Passed	04
5	Implementation.PIO.BackEnd.test.Test	Passed	05
6	Implementation.PIO.BackEnd.test.Test	Passed	06

4 Test Cases

4.0.1 Condition 1:AddProperty

4.1 Test Case 1: Add a property with negative ID

4.1.1 Condition 1: Back-End Server must be running

4.1.1.1 Objective: The purpose of this test is to check how the validation of add property works when invalid values are injected.

4.1.1.2 Input: Negative values are used as input , -1.

4.1.1.3 Outcome: The function will throw a ArithmeticException.

4.2 Test Case 2: Delete a property with negative ID

4.2.1 Condition 1: Back-End Server must be running

4.2.1.1 Objective: The purpose of this test is to check how the validation of add property works when invalid values are injected.

4.2.1.2 Input: Negative values are used as input , -1.

4.2.1.3 Outcome: The function will throw a ArithmeticException.

4.3 Test Case 3: Find a property that does not exist

4.3.1 Condition 1: Back-End Server must be running

4.3.1.1 Objective: The purpose of this test is to test how the function handles a request when an id of a none-existing property is sent

4.3.1.2 Input: Any propertyID Value that is not in the database , 33.

4.3.1.3 Outcome: The function will throw a NullPointerException.

4.4 Test Case 4: Find a profile that does not exist

4.4.1 Condition 1: Back-End Server must be running

4.4.1.1 Objective: The test must return an exception if the given profileID is not found in the database.

4.4.1.2 Input: Any profileID Value that is not in the database , 33.

4.4.1.3 Outcome: The function will throw a NullPointerException.

4.5 Test Case 5: Delete a profile that does not exist

4.5.1 Condition 1: Back-End Server must be running

4.5.1.1 Objective: The test must return an exception if the given profileID to be deleted is not fond in the database.

4.5.1.2 Input: Any profileID Value that is not in the database , 33

4.5.1.3 Outcome: The function will throw a NullPointerException.

4.6 Test Case 6: Delete properties that are associated with a profile that do not exist

4.6.1 Condition 1: Back-End Server must be running

4.6.1.1 Objective: The test must return an exception if the given profileID does not have any properties associated with it.

4.6.1.2 Input:

Any profileID Value that does not have any properties associated with it

4.6.1.3 Outcome:

The function will throw a NullPointerException.

5 Item Pass/Fail Criteria

Each item tested must meet a certain criteria in order to pass. These criteria are as follows:
Back-End

- the back end server must be running

AddPropertyValidation

- the property must not exist in the database
- the input in the parameters must be invalid

DeletepropertyValidation

- the property must not exist in the database
- the input in the parameters must be invalid

FindProperty

- the back end server must be running
- the property must not exist in the database
- the input in the parameters must be valid

FindProfile

- the back end server must be running
- the profile must not exist in the database
- the input in the parameters must be valid

DeleteProfile

- the back end server must be running
- the profile must not exist in the database
- the input in the parameters must be valid

RetrievePropertiesTestValidation

- the back end server must be running
- the profile must not have any properties associated with it in the database
- the input in the parameters must be valid

6 Test Deliverables

Artefacts to be produced as part of unit testing include the following:

- Test Plan
- Test Report
- Link to Test Code

Unit Test Report

7 Detailed Test Results

7.1 Overview of Test Results

The test that were done were mostly on the back-end.This is where we validate our values before they are inserted into the database.all the test we did passed even those that would cause the systems fail. This was done by handling the system failure correctly,anticipating what would go wrong and handling that failure in a correct way through try and catch blocks.The tests expected the system to fail and the error to be handled ,so once it it handled in the expected way, the test pass. Junit spring framework was used for unit testing in the backend.

7.2 Functional Requirements Test Results

The tests we have created for this module are contained within the files PropertyTest and ProfileTest . and gitHub.

7.2.1 Test Case 1 (TC 4.1.1)

The following results were obtained from the tests conducted. The tests to produce the following results have passed/failed and the reasons are stated below.

7.2.1.1 Result: Pass/ Fail.

7.2.2 Test Case 2 (TC 4.2)

The invalid value was handled though a try and catch block by throwing an ArithmeticException

7.2.2.1 Result: Pass.

7.2.3 Test Case 3 (TC 4.3)

The invalid value was handled though a try and catch block by throwing an ArithmeticException

7.2.3.1 Result: Pass.

7.2.4 Test Case 4 (TC 4.4)

The invalid value was handled though a try and catch block

7.2.4.1 Result: Pass.

7.2.5 Test Case 5 (TC 4.5)

The invalid value was handled though a try and catch block

7.2.5.1 Result: Pass.

7.2.6 Test Case 6 (TC 4.6)

The invalid value was handled though a try and catch block

7.2.6.1 Result: Pass.

8 Other

The service contracts ensure that our tests would test that the functionality would return relevant and accurate data.

The different use cases do allow for the application to be deployed into an application server because they cover all the functionality that is required of the application.

9 Conclusions and Recommendations

Test were done on the back-end to test for security features implemented as the application allows users to input data.Parameters are passed though the url to the back-end. So regardless of the validation we have in the font-end, malicious users might pass in malicious code after the validation in the front-end through the url, so it is important for the application to handle both user errors and malicious code, the test have assured that the countermeasures implemented work correctly.