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:

Member 1- 12031748
Member 2- 13278012
Member 3- 13134885
Member 4- 13049128

September 28, 2016

IMPAKD link

For further references see gitHub. September 28, 2016

Contents

1 Introduction

1.1 Purpose

This document combines the unit test plan and report into a single coherent artefact. Explain the purpose of your system and of this document here. Why is test driven development essential to your system.

1.2 Scope

The scope of this document is structured as follows. The features that are considered for testing are listed in section .... Tests that have been identifed from the requirements are discussed in detail in section .... Furthermore, this document outlines the test environment and the risks involved in the testing approaches that will be followed. Assumptions and dependencies of this test plan will also be mentioned. Section 7.1, 7.1 and 9 outlines, discusses and concludes on the results of the tests, respectively.

1.3 Test Environment

1.4 Assumptions and Dependencies

Unit Test Plan

2 Test Items

3 Functional Features to be Tested

Feature ID	RDS Source	Summary	Test Case ID
1	Implementation.PIO.FrontEnd.app.addProperty	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.FrontEnd.app.addProperty	Passed	06

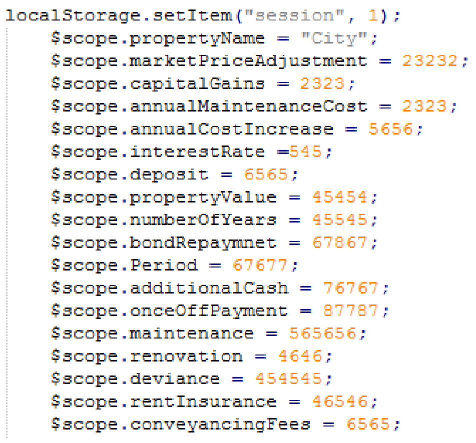
4 Test Cases

4.1 Test Case 1: Test if the encoded string sent to the back-end contains the right information

4.1.1 Condition 1: session is created

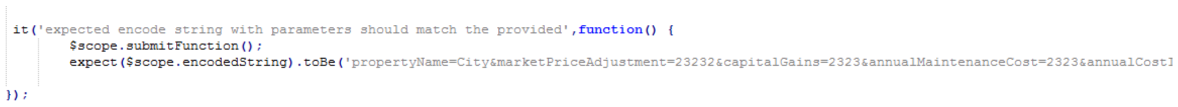
4.1.1.1 Objective: The purpose of this test is to test if the infomation being passed by the $http request is correct

4.1.1.2 Input: The following inputs will be used to test this functionality:



```
localStorage.setItem("session", 1);
    $scope.propertyName = "City";
    $scope.marketPriceAdjustment = 23232;
    $scope.capitalGains = 2323;
    $scope.annualMaintenanceCost = 2323;
    $scope.annualCostIncrease = 5656;
    $scope.interestRate =545;
    $scope.deposit = 6565;
    $scope.propertyValue = 45454;
    $scope.numberOfYears = 45545;
    $scope.bondRepaymnet = 67867;
    $scope.Period = 67677;
    $scope.additionalCash = 76767;
    $scope.onceOffPayment = 87787;
    $scope.maintenance = 565656;
    $scope.renovation = 4646;
    $scope.deviance = 454545;
    $scope.rentInsurance = 46546;
    $scope.conveyancingFees = 6565;
```

4.1.1.3 Outcome: The mocked user input will have to match the equivalent encoded string provided in the expected parameter



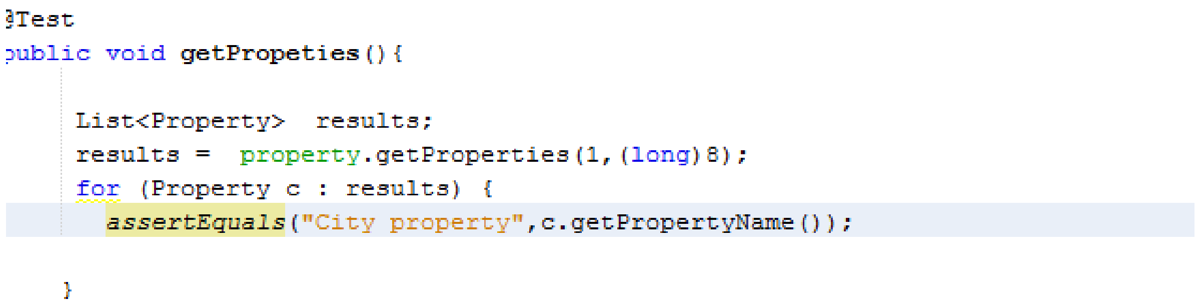
```
it('expected encode string with parameters should match the provided',function() {
        $scope.submitFunction();
        expect($scope.encodedString).toBe('propertyName=City&marketPriceAdjustment=23232&capitalGains=2323&annualMaintenanceCost=2323&annualCostI
});
```

4.2 Test Case 2: Retrieve a property and test if the Property name is as expected

4.2.1 Condition 1: Back-End Server must be running

4.2.1.1 Objective: The purpose of this test is to test the getProperties function is it returns the correct property

4.2.1.2 Input: user profile ID and a property ID , (1 ,8)



```
@Test
public void getPropeties(){

    List<Property>  results;
    results =  property.getProperties(1,(long)8);
    for (Property c : results) {
      assertEquals("City property",c.getPropertyName());

  }
```

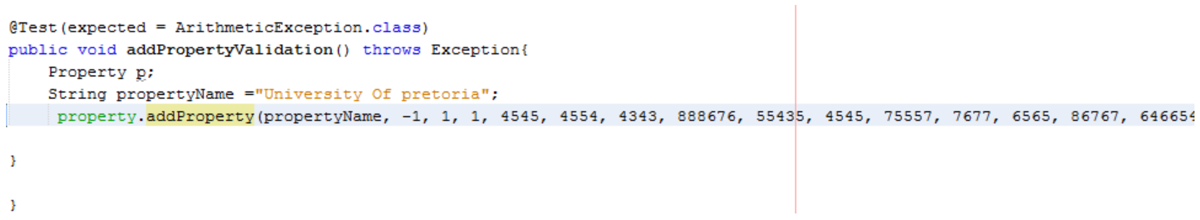
4.2.1.3 Outcome: The function will throw a ArithmeticException

4.3 Test Case 3: Add a property with negative ID

4.3.1 Condition 1: Back-End Server must be running

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

4.3.1.2 Input: Negative values are used as input , -1



```
@Test(expected = ArithmeticException.class)
public void addPropertyValidation() throws Exception{
    Property p;
    String propertyName ="University Of pretoria";
     property.addProperty(propertyName, -1, 1, 1, 4545, 4554, 4343, 888676, 55435, 4545, 75557, 7677, 6565, 86767, 64665

}

}
```

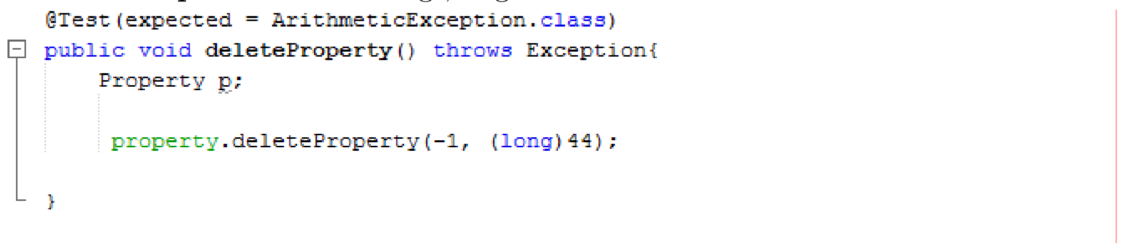
4.3.1.3 Outcome: The function will throw an ArithmeticException

4.4 Test Case 4: Delete a property with an none-existing ID

4.4.1 Condition 1: Back-End Server must be running

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

4.4.1.2 Input: Nonexisting , negative values such -1



```
@Test(expected = ArithmeticException.class)
public void deleteProperty() throws Exception{
    Property p;

    property.deleteProperty(-1, (long)44);

}
```

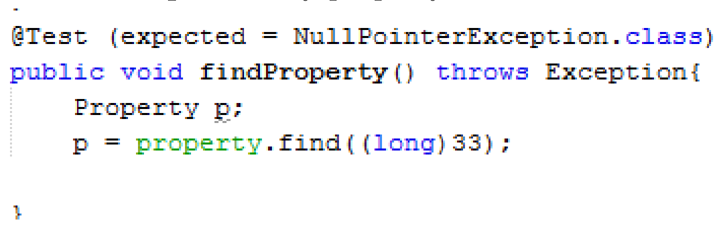
4.4.1.3 Outcome: The function will throw an ArithmeticException

4.5 Test Case 5: Find a property that does not exist

4.5.1 Condition 1: Back-End Server must be running

4.5.1.1 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

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



```
@Test (expected = NullPointerException.class)
public void findProperty() throws Exception{
    Property p;
    p = property.find((long)33);

}
```

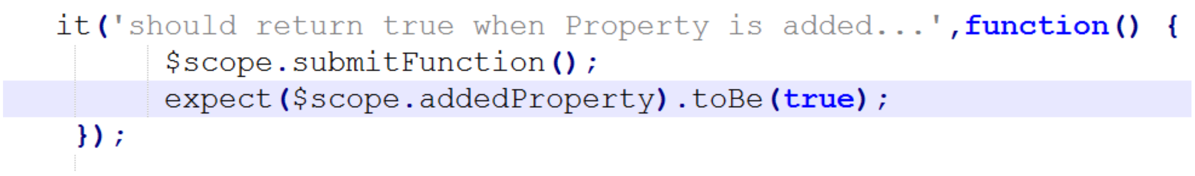
4.5.1.3 Outcome: The function will throw a NullPointerException

4.6 Test Case 6: Set AddedProperty boolean to true if property has been inserted

4.6.1 Condition 1: Back-End Server must be running

4.6.1.1 Objective: The purpose of this function is to test in the front-end application whether the property was successful sent to the back-end and persisted in the database, the $http request's success function initialises the boolean to true if it was successful otherwise it is initialised to false by the error function

4.6.1.2 Input: endcoded String



```
it('should return true when Property is added...',function() {
        $scope.submitFunction();
        expect($scope.addedProperty).toBe(true);
  });
```

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:

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

Test the encoded String

- the must be values in the string

Add Property front-end

- the must be values in the string
- session ID must be initialised

Retrieve a property

- the back-end server must be running
- the property must not exist 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 normally 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. Junit spring framework was used for unit testing in the back-end. The front-end test where done to the test the system in a wider scope because the front-end is dependent on the back-end.when a test that sends a request to the back-end passes, it also assures that the function in the back-end are working correctly.We used Karma and Jasmine framework to conduct unit test in javascript.

7.2 Functional Requirements Test Results

The tests we have created for this module are contained within the files PropertyTest, Font-End Test and gitHub.

7.2.1 Test Case 1 (TC 4.1)

The correct information was passed to the $http request

7.2.1.1 Result: Pass.

7.2.2 Test Case 2 (TC 4.2)

The property list was returned and the name was extracted

7.2.2.1 Result: Pass.

7.2.3 Test Case 3 (TC 4.3)

An ArithmeticException was thrown and the invalid value was handled though a try and catch block

7.2.3.1 Result: Pass.

7.2.4 Test Case 4 (TC 4.4)

An ArithmeticException was thrown and 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

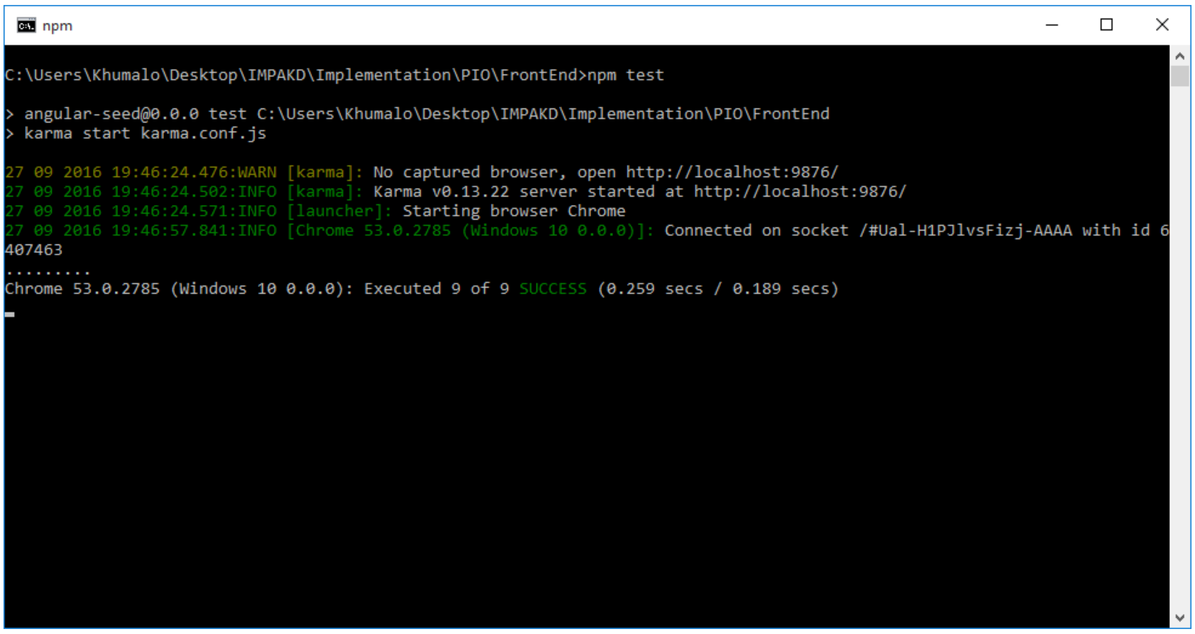
7.2.5.1 Result: Pass.

7.2.6 Test Case 6 (TC 4.6)

The function returned true, hence it successfully persisted in the database

7.2.6.1 Result: Pass.

8 Other

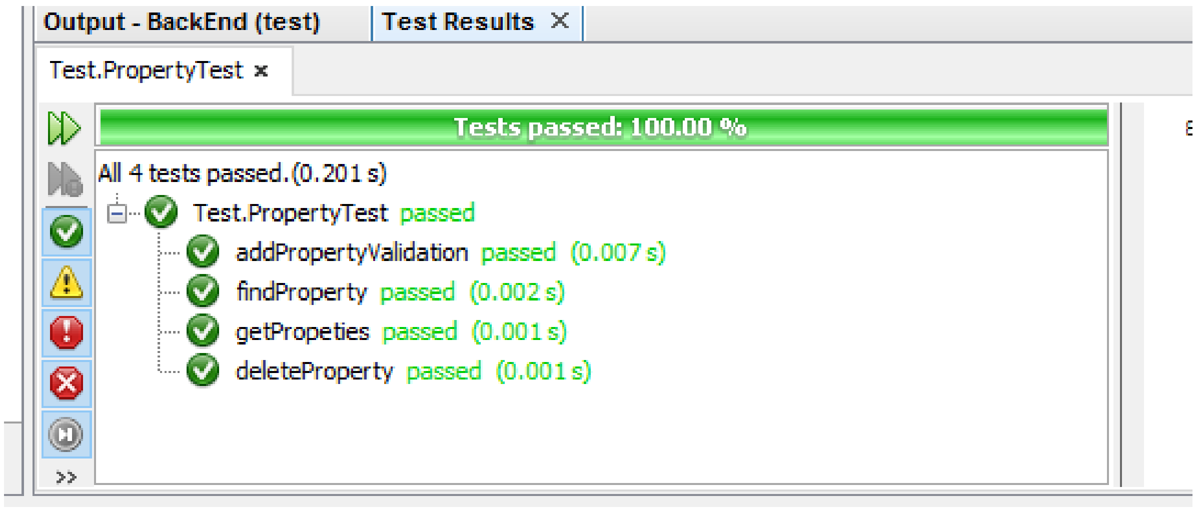


```
C:\Users\Khumalo\Desktop\IMPAKD\Implementation\PIO\FrontEnd>npm test

> angular-seed@0.0.0 test C:\Users\Khumalo\Desktop\IMPAKD\Implementation\PIO\FrontEnd
> karma start karma.conf.js

27 09 2016 19:46:24.476:WARN [karma]: No captured browser, open http://localhost:9876/
27 09 2016 19:46:24.502:INFO [karma]: Karma v0.13.22 server started at http://localhost:9876/
27 09 2016 19:46:24.571:INFO [launcher]: Starting browser Chrome
27 09 2016 19:46:57.841:INFO [Chrome 53.0.2785 (Windows 10 0.0.0)]: Connected on socket /#Ual-H1PJlvsFizj-AAAA with id 6
407463
.........
Chrome 53.0.2785 (Windows 10 0.0.0): Executed 9 of 9 SUCCESS (0.259 secs / 0.189 secs)
```

Front-End Test results



```
Tests passed: 100.00 %
All 4 tests passed.(0.201 s)
Test.PropertyTest passed
    addPropertyValidation passed (0.007 s)
    findProperty passed (0.002 s)
    getPropeties passed (0.001 s)
    deleteProperty passed (0.001 s)
```

Back-End Test results

9 Conclusions and Recommendations

Test were done in 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.