Author: Domn Werner (wern0096)
Class: CS383-01
Date:04/25/2016
Assignment: HW#7 Test Plan

Contents

1 MTP01 - sQuire Master Test Plan

2 References

This test plan references the following documents:

- sQuire SSRS document.

3 Introduction

The purpose of this test plan is to provide a outline and provide reference for developers testing the complete sQuire program. This document is currently a standalone, but will be integrated with the SSRS document before the final submission. This document covers sQuire's logic tests, GUI tests, back-end tests, coverage tests, and any additional testing methods deemed necessary to ascertain that the complete sQuire software program adheres to our group's acceptable quality standard.

4 Logic Testing

The purpose of this section of tests is to list and describe logic tests in the sQuire program and the required output deemed as a "pass". These include algorithmic functions, arithmetic functions, validator functions, and other pieces of functionality that can easily be decoupled from the main project and/or reused as part of different projects.

4.1 Test Classes

Table 1: PasswordHashTest

Function Name	**Description**	**Pass Criteria**
createHash()	Verifies that the hashing algorithm does not create colliding hashes.	A false assertion that all hashes created inside this function are different.
validatePasswor()	Verifies that the PasswordHash.validatePassword() function correctly authenticates a user based on their hashes password.	A true assertion that the a valid password and hash were validated. A false assertion that an invalid password and hash were validated.

Table 2: EditorControllerTest (dani2918)

Function Name	**Description**	**Pass Criteria**
testSetupMobWrite()	Verifies that we can set up mobwrite components with various names.	Successful creation of mobwrite components with various names constitutes a success.

Table 3: NewProjectControllerTest (dani2918)

Function Name	**Description**	**Pass Criteria**
testInitProjectFields()	Verifies that projects with various names and descriptions (in the form of strings due to the controller class's use of strings from TextFields) are properly created.	Successful creation of project directory inside a test directory constitutes a passing test.
testCopyMainFile()	Verifies that the initial dummy Main "Hello World" class is successfully copied into a directory.	Existence of the file at the specified location constitutes a passing test.

Other logic test tables here...

4.2 Results

Table 4: EditorControllerTest (dani2918)

Function Name	**Result**	**Description**
testSetupMobWrite()	FAILURE	Attempting to create a new CodeArea, which the setupMobWrite function requires as a parameter, was not working in the test class. Further investigation will be required to determine whether the function works properly.

Table 5: NewProjectControllerTest (dani2918)

Function Name	**Result**	**Description**
testInitProjectFields()	FAILURE	The test fails when attempting to create a project based upon the empty string as a title. We will have to implement logic to ensure that a user enters a project title in the appropriate field. This is the only case of those tested which caused a failure.
testCopyMainFile()	PASS	The copied file existed with multiple attempts.

The result of these tests should go here.

5 GUI Testing

This section governs our GUI unit tests and the required output deemed as a "pass". Since we are using the JavaFX framework, every test case requires an initialization step of loading the .fxml file for the GUI scene to be tested. Once it is loaded we perform tests on individual parts of the scene using the TestFX libraries that integrate with JUnit.

5.1 Test Classes

Table 6: HomeTest (wern0096)

Function Name	**Description**	**Pass Criteria**
verifyUiElementsLoaded()	Checks that every UI element loaded properly.	No exceptions thrown by the verifyThat() function calls.

Table 7: EditorTest (wern0096)

Function Name	**Description**	**Pass Criteria**

Table 8: NewProjectTest (dani2918)

Function Name	**Description**	**Pass Criteria**
verifyUiElementsLoaded()	Checks that every UI element loaded properly.	No exceptions thrown by the verifyThat() function calls.

Other GUI Test tables here...

5.2 Results

The result of these tests should go here.

Table 9: NewProjectTest (dani2918)

Function Name	**Description**	**Pass Criteria**
verifyUiElementsLoaded()	PASS	No exceptions were thrown after loading all elements from the FXML file.

6 Back-end Testing

This section governs any tests aimed at our database(s) or server(s) and the required output deemed as a "pass".

6.1 Test Classes

Table 10: MobWriteClientTest (ratc8795)

Function Name	**Description**	**Pass Criteria**

Table 11: MobWriteServerTest (ratc8795)

Function Name	**Description**	**Pass Criteria**

Table 12: SessionTest (ratc8795)

Function Name	**Description**	**Pass Criteria**

Table 13: UserTest (ratc8795)

Function Name	Description	Pass Criteria

Table 14: ProjectDatabaseTest (cart1189)

Function Name	Description	Pass Criteria

6.2 Results

The result of these tests should go here.

7 Coverage Testing

7.1 Methodology

How are we doing it? We should just "Run with coverage" in IntelliJ and go through the program manually.

7.2 Results

Figure out a way of exporting the results from IntelliJ.

8 Software Risk Issues (wern0096)

What are the critical areas of our software?

9 Features To Be Tested

Take the intersection of the SSRS document and what we are testing in our current tests.

10 Features Not To Be Tested

Take the intersection of the SSRS document and what we ARENT testing in our current tests. Make sure to explain WHY we aren't testing those pieces of functionality.