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

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 (wern0096)

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 Password-Hash.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.

4.2 Results

4.2.1 PasswordHashTest (wern0096)

PasswordHashTest: 2 total, 2 passed   5.83 s

Collapse | Expand

PasswordHashTest.validatePassword	passed	907 ms
PasswordHashTest.createHash	passed	4.92 s

Generated by IntelliJ IDEA on 4/25/16 9:29 PM

This passed test indicates that we our hashing algorithm successfully validates a user's entered password with their hashed password, and properly creates non-colliding hashes. It also shows us that the createHash method takes 5 seconds to run, indicating a possible place for better optimization.

Table 4: EditorControllerTest (dani2918)

Function Name	Result	Description
testSetupMobWrite()	FAILURE	Attempting to create a new CodeArea, which the setup-MobWrite 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.

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
verifyUiElementsLoaded()	Checks that every UI element loaded properly.	No exceptions thrown by the verifyThat() function calls.

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.

5.2 Results

5.2.1 HomeTest

HomeTest: 1 total, 1 passed   515 ms

Collapse | Expand

HomeTest.verifyUiElementsLoaded   passed   515 ms

Apr 25, 2016 8:54:15 PM javafx.fxml.FXMLLoader$ValueElement processValue
WARNING: Loading FXML document with JavaFX API of version 8.0.65 by JavaFX runtime of version 8.0.20

Generated by IntelliJ IDEA on 4/25/16 8:54 PM

This passed test indicates that all UI elements are loading properly. It does create a warning message about a mismatch of the JavaFX API and the runtime, however, and wee will remedy that next sprint.

5.2.2 EditorTest

EditorTest: 1 total, 1 error   296 ms

Collapse | Expand

| EditorTest.verifyUiElementsLoaded | error   296 ms |

Generated by IntelliJ IDEA on 4/25/16 8:56 PM

This test currently fails due to it requiring initialization with data from another class. This will be better tested with an automation script, but it is still useful to know what data it requires to successfully initialize. Here is the following exception it throws:
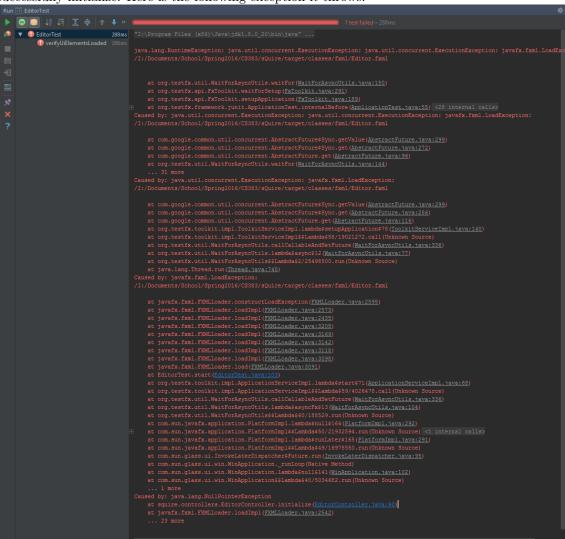
```
Run  EditorTest

EditorTest                288ms    "I:\Program Files (x86)\Java\jdk1.8.0_20\bin\java" ...
    verifyUiElementsLoaded 288ms
                                    java.lang.RuntimeException: java.util.concurrent.ExecutionException: java.util.concurrent.ExecutionException: javafx.fxml.LoadExc
                                    /I:/Documents/School/Spring2016/CS383/sQuire/target/classes/fxml/Editor.fxml

                                        at org.testfx.util.WaitForAsyncUtils.waitFor(WaitForAsyncUtils.java:150)
                                        at org.testfx.api.FxToolkit.waitForSetup(FxToolkit.java:291)
                                        at org.testfx.api.FxToolkit.setupApplication(FxToolkit.java:189)
                                        at org.testfx.framework.junit.ApplicationTest.internalBefore(ApplicationTest.java:55) <28 internal calls>
                                    Caused by: java.util.concurrent.ExecutionException: java.util.concurrent.ExecutionException: javafx.fxml.LoadException:
                                    /I:/Documents/School/Spring2016/CS383/sQuire/target/classes/fxml/Editor.fxml

                                        at com.google.common.util.concurrent.AbstractFuture$Sync.getValue(AbstractFuture.java:299)
                                        at com.google.common.util.concurrent.AbstractFuture$Sync.get(AbstractFuture.java:272)
                                        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:96)
                                        at org.testfx.util.WaitForAsyncUtils.waitFor(WaitForAsyncUtils.java:144)
                                        ... 31 more
                                    Caused by: java.util.concurrent.ExecutionException: javafx.fxml.LoadException:
                                    /I:/Documents/School/Spring2016/CS383/sQuire/target/classes/fxml/Editor.fxml

                                        at com.google.common.util.concurrent.AbstractFuture$Sync.getValue(AbstractFuture.java:299)
                                        at com.google.common.util.concurrent.AbstractFuture$Sync.get(AbstractFuture.java:286)
                                        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:116)
                                        at org.testfx.toolkit.impl.ToolkitServiceImpl.lambda$setupApplication$78(ToolkitServiceImpl.java:140)
                                        at org.testfx.toolkit.impl.ToolkitServiceImpl$$Lambda$58/19021272.call(Unknown Source)
                                        at org.testfx.util.WaitForAsyncUtils.callCallableAndSetFuture(WaitForAsyncUtils.java:336)
                                        at org.testfx.util.WaitForAsyncUtils.lambda$async$12(WaitForAsyncUtils.java:77)
                                        at org.testfx.util.WaitForAsyncUtils$$Lambda$2/25498500.run(Unknown Source)
                                        at java.lang.Thread.run(Thread.java:745)
                                    Caused by: javafx.fxml.LoadException:
                                    /I:/Documents/School/Spring2016/CS383/sQuire/target/classes/fxml/Editor.fxml

                                        at javafx.fxml.FXMLLoader.constructLoadException(FXMLLoader.java:2595)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:2573)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:2435)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:3208)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:3169)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:3142)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:3118)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:3098)
                                        at javafx.fxml.FXMLLoader.load(FXMLLoader.java:3091)
                                        at EditorTest.start(EditorTest.java:103)
                                        at org.testfx.toolkit.impl.ApplicationServiceImpl.lambda$start$71(ApplicationServiceImpl.java:68)
                                        at org.testfx.toolkit.impl.ApplicationServiceImpl$$Lambda$59/4026478.call(Unknown Source)
                                        at org.testfx.util.WaitForAsyncUtils.callCallableAndSetFuture(WaitForAsyncUtils.java:336)
                                        at org.testfx.util.WaitForAsyncUtils.lambda$asyncFx$13(WaitForAsyncUtils.java:104)
                                        at org.testfx.util.WaitForAsyncUtils$$Lambda$60/188529.run(Unknown Source)
                                        at com.sun.javafx.application.PlatformImpl.lambda$null$164(PlatformImpl.java:292)
                                        at com.sun.javafx.application.PlatformImpl$$Lambda$50/21932594.run(Unknown Source) <1 internal calls>
                                        at com.sun.javafx.application.PlatformImpl.lambda$runLater$165(PlatformImpl.java:291)
                                        at com.sun.javafx.application.PlatformImpl$$Lambda$49/16978550.run(Unknown Source)
                                        at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(InvokeLaterDispatcher.java:95)
                                        at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
                                        at com.sun.glass.ui.win.WinApplication.lambda$null$141(WinApplication.java:102)
                                        at com.sun.glass.ui.win.WinApplication$$Lambda$40/5034682.run(Unknown Source)
                                        ... 1 more
                                    Caused by: java.lang.NullPointerException
                                        at squire.controllers.EditorController.initialize(EditorController.java:90)
                                        at javafx.fxml.FXMLLoader.loadImpl(FXMLLoader.java:2542)
                                        ... 23 more
```

Table 9: NewProjectTest (dani2918)

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

7

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".

One of the major libraries we use on the server (MobWrite) contains a full test suite written in JUnit, however, since we use the library as-is, without any modifications, we do not integrate these tests with our test suite. Instead, we just test to see if the client can successfully connect to the server, as that should be the only variable.

6.1 Test Classes

Table 10: MobWriteServerTest (ratc8795)

Function Name	Description	Pass Criteria
connectToServer()	Checks to see if the server can be connected to.	No exceptions thrown by the Connection.connect() function.

Table 11: MobWriteClientTest (ratc8795)

Function Name	Description	Pass Criteria
testComputeSyncInterval()	Tests that the sync interval is calculated correctly.	The sync interal is calculated correctly under a variety of conditions.
testUniqueId()	Tests that unique IDs are created correctly.	The ID generated is 8 characters long, and two seperate IDs are unique.

Table 12: SessionTest (ratc8795)

Function Name	Description	Pass Criteria
isExpired()	Check that the isExpired funciton works correctly.	A not-expired session returns false, and a expired session returns true.
logout()	Check that logout function works correctly.	Once the logout function is called on the session, the session no longer exists.
activeSession()	Check the activeSession function returns an active session given a token.	The correct session is returned for a given token, a not active session won't be returned, and a non exsistant token won't return a session.

Table 13: ChatServerTest (gent7104)

Function Name	Description	Pass Criteria
VerifyConnection()	Verify that the a connection has been made	No exception is thrown when testconnection() is called.

Table 14: ChatClientTest (gent7104)

Function Name	Description	Pass Criteria
VerifySocektConnection()	Verify that the client was able to connect with the specified host and and port number	No exception is thrown when connecting to the socket.

6.2 Results

7 Coverage Testing (wern0096)

7.1 Methodology

The coverage test of sQuire was performed by running the program with the coverage testing option enabled in IntelliJ IDEA and a developer manually navigating through all of the functionality in the program.

7.2 Results

The following report was then generated, showing usability per package:

[all classes]

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	58.2% (32/ 55)	44% (147/ 334)	34.9% (942/ 2699)

Coverage Breakdown

Package ▼	Class, %	Method, %	Line, %
squire.controllers	90.9% (10/ 11)	87.7% (57/ 65)	83.1% (375/ 451)
squire.chatserver	0% (0/ 3)	0% (0/ 9)	0% (0/ 136)
squire.Users.query.assoc	0% (0/ 4)	0% (0/ 16)	0% (0/ 24)
squire.Users.query	25% (1/ 4)	16.7% (2/ 12)	20% (4/ 20)
squire.Users	100% (9/ 9)	45.5% (30/ 66)	40.4% (61/ 151)
squire.Projects	66.7% (2/ 3)	32.1% (9/ 28)	25% (18/ 72)
squire	66.7% (2/ 3)	51.6% (16/ 31)	56.3% (49/ 87)
google.mobwrite	44.4% (8/ 18)	30.8% (33/ 107)	24.7% (435/ 1758)

generated on 2016-04-25 21:15

As indicated, most of the squire.controllers package was hit by our coverage test, indicating very little extraneous UI code. The chat server was not tested with this run, and the rest of the packages had about half of their functionalities hit, due to being heavily in development at the moment. As an example of the granularity of this report, let's examine the squire.controllers package to see what code didn't run during this test:

Table 15: ProjectDatabaseTest (cart1189)

Function Name	Description	Pass Criteria
testAddProject()	Checks the ability to create and save new Project objects to their database table.	No exceptions thrown during Project initialization or database save(). The new Project entry is in the table during the test.
testRemoveProject()	Checks the ability to remove Project object entries from their database table.	No exceptions thrown during Project initialization or database save(). The Project in question is removed from the table by the the end of the test.
testGetSetTextFields()	Checks ability to set each string field of a Project database entry, and retrieve those fields from the database.	No exceptions thrown by the test. Each retrieved value matches the value it was set to.
testLoadFile()	Checks ability to upload and download raw files into the ProjectFile table	No exceptions thrown by the test. Retrieved copy of uploaded file matches the original version.

Table 16: MobWriteServerTest (ratc8795)

Function Name	Result	Description
connetToServer()	PASS	

[all classes] [squire.controllers]

Coverage Summary for Package: squire.controllers

Package	Class, %	Method, %	Line, %
squire.controllers	90.9% (10/ 11)	87.7% (57/ 65)	83.1% (375/ 451)

Class ▼	Class, %	Method, %	Line, %
SettingsDialogController	100% (1/ 1)	100% (5/ 5)	100% (17/ 17)
RegisterDialogController	100% (1/ 1)	100% (5/ 5)	100% (15/ 15)
ProjectBrowsingController	100% (1/ 1)	100% (2/ 2)	83.3% (10/ 12)
PreferencesDialogController	100% (1/ 1)	100% (4/ 4)	100% (10/ 10)
NewProjectController3	100% (1/ 1)	100% (10/ 10)	92% (80/ 87)
LogInDialogController	100% (1/ 1)	80% (4/ 5)	67.9% (19/ 28)
HomeController	100% (1/ 1)	88.9% (8/ 9)	75% (75/ 100)
EditorController	75% (3/ 4)	76% (19/ 25)	81.9% (149/ 182)

generated on 2016-04-25 21:

Delving further, let's see what code in the EditorController didn't run:

[all classes] [squire.controllers]

Coverage Summary for Class: EditorController (squire.controllers)

Class	Method, %	Line, %
EditorController	100% (14/ 14)	93.8% (121/ 129)
EditorController$1	100% (2/ 2)	100% (17/ 17)
EditorController$TextFieldTreeCellImpl	42.9% (3/ 7)	36.7% (11/ 30)
EditorController$TextFieldTreeCellImpl$1	0% (0/ 2)	0% (0/ 6)
total	76% (19/ 25)	81.9% (149/ 182)

Lastly, we can get very granular and see exactly which lines of code didn't run. Here's an example of the updateItem() function's code and how the IntelliJ Coverage Report displays code that was hit with a green outline and code that wasn't hit with a red outline:

Table 17: MobWriteClientTest (ratc8795)

Function Name	Result	Description
testComputeSyncInterval()	PASS	
testUniqueID()	PASS	

Table 18: SessionTest (ratc8795)

Function Name	Result	Description
isExpired()	PASS	
logout()	PASS	
activeSession()	PASS	

```java
@Override
public void updateItem(String item, boolean empty)
{
    super.updateItem(item, empty);

    if (empty)
    {
        setText(null);
        setGraphic(null);
    }
    else
    {
        if (isEditing())
        {
            if (textField != null)
            {
                textField.setText(getString());
            }
            setText(null);
            setGraphic(textField);
        }
        else
        {
            setText(getString());
            setGraphic(getTreeItem().getGraphic());
        }
    }
}
```

Table 19: UserTest (ratc8795)

Function Name	Result	Description
getSetUsername()	PASS	
setCheckPassword()	PASS	
authenticate()	PASS	
userQuery()	PASS	

8 Software Risk Issues (wern0096)

8.1 High Risk

The following items are deemed high risk to the security of users and to the usability and quality of the sQuire software:

- **MobWrite Server and Client**

  Our collaborative editing relies on Google's MobWrite software library. We are rating it high risk because it is the core of the collaborative editing functionality, and it is also a third-party tool with a high impact on the functionality of our product. This module also incorporates the diff-match-patch algorithm running on clients' machines that talks to our Azure server that broadcasts text changes from one client to the rest of the clients. Since it will be sending code over the internet to and from the server, the security implications for all machines involved are very serious. As such, we plan to have extensive testing and review of code using this module.

- **Chat Client**

  Our chat client is simply a Java application running indefinitely in the same Azure server as the MobWrite server. Nevertheless, it is broadcasting possibly confidential data through the internet and should have the proper encryption and security reviews as the rest of the high risk items.

- **JavaFX**

  The JavaFX framework is also core to our program. Since it has been used to build most of the user interface, it will require great usability and coverage testing to make sure that it is friendly and intuitive to our users. The breakdown of the user interface would essentially render the rest of the program useless. Thus, it receives a high risk rating.

- **Database Credentials**

  The project uses a SQLite database for storing of user credentials and project information. We rate the security of user and project credentials in the database as high risk. In order to protect the confidentiality of user credentials and integrity of user projects, we will have to adhere to database security best practices such as salting and hashing stored credentials, encrypting traffic over the internet, and ensuring SQL-injection attacks are mitigated.

8.2 Medium Risk

The following items are deemed medium risk to the security of users and to the usability and quality of the sQuire software:

- **Database Storage and Ebean ORM**

  The non-confidential data stored in the database is rated as medium risk. This is because project data is also stored locally, and breakdown of the database storage functionality would

not severely reduce core functionality of the sQuire software. However, it would still significantly hinder collaborative functionality. To address this, we plan on extensively testing database commands with unit tests and code reviews.

- **Code Compilation**

  Whenever code compilation is involved, security becomes a concern. However, there is not code being executed remotely, so the security risks become much smaller, thus earning this module a medium rating. Compilation in sQuire will rely on the user reviewing their code for malicious intent before executing. We plan on implementing tools for the user to easily track changes to the code in order to aid in the review process if we have time. Also, we will have unit tests making sure that the user is properly notified of code errors and their location(s).

- **Testing Modules**

  We are putting testing modules themselves as a medium risk item because of the sheer complexity of testing the user interface with JavaFX and TestFX, another open source framework. Since the user interface is rated as a high-risk item, we believe that properly testing it is at least a medium risk item to the proper functionality of our program. We didn't rate it as high-risk, because there are many ways of testing the user interface that are less complex but take more time.

8.3   Low Risk

The following items are deemed low risk to the security of users and to the usability and quality of the sQuire software:

- **Editor Features**

  This module features some complicated code that will come from third-party open sources such as search/replace, syntax highlighting, and auto-complete. However, failure of this module does not severely endanger users' security or the core functionality of the software. As such, it earns a low risk rating. This module will require extensive unit testing, however.

- **Local File Structure**

  The local file structure stores user's projects locally in the form of a folder for each project. We rate this module as a low risk module because even if the local files were corrupted or deleted, the user's projects/files can still be restored from the database, and the program has code to handle such a case. Nevertheless, we plan to have various unit tests documenting valid/invalid file structures that we can then create code to handle such cases.

- **Program Settings**

  This module involves settable user preferences that should persist between runs of sQuire. We plan to have the user's settings be saved locally as well as on the database, so that upon login, sQuire will update to conform to the user's preferences. Since there is redundancy to this module and its breakdown does not constitute a large hit to the functionality of sQuire, we deem it low risk.