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 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.

Other logic test tables here...

4.2 Results

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 2: HomeTest (wern0096)

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

Table 3: EditorTest (wern0096)

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

Other GUI Test tables here...

5.2 Results

The result of these tests should go here.

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 4: MobWriteClientTest (ratc8795)

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

Table 5: MobWriteServerTest (ratc8795)

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

Table 6: SessionTest (ratc8795)

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

Table 7: UserTest (ratc8795)

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

Table 8: 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)

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.

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.