Project Title: System Verification and Validation Plan for Software Eng 4G06

Team 2, Parnas' Pals
William Lee
Jared Bentvelsen
Bassel Rezkalla
Yuvraj Randhawa
Dimitri Tsampiras
Matthew McCracken

November 2, 2022

1 Revision History

Date	Version	Notes
02-11-2022	1.0	Initial Draft

Contents

List of Tables

This document serves to document the plan place to verify and validate the development of the Olympian Application.

2 General Information

2.1 Summary

The software being tested is a mobile application called Olympian. It is a workout application with a social component. Important components of the software that will be getting tested are:

- The functionality to log in and sign up new users. This will function to allow users to create a customized experience with data being saved over between uses.
- The functionality for users to create and upload their own workouts. Specifically, users should be able to set a workout category, outline steps for their workout plan, and include videos and images to support the exercise.
- The functionality for users to discover new and popular workouts that have been created by other users. Users should be able to sort by new or trending and filter by the muscle groups that they would like to target.

2.2 Objectives

- Demonstrate ease of use for new users to sign in.
- Show off intuitiveness and learnability of the workout creation process.
- Demonstrate usability of workout feed. Newly created workouts must appear in this feed.
- Support the claim that our client shall update the server and vice versa in under 10 seconds under average North American internet speeds.

Team Member	Role
Jared Bentvelsen	Integration Testing
Yuvraj Randhawa	Integration Testing
Bassel Rezkalla	Functional Unit Testing
Matthew McCracken	Non Functional Unit Testing
Dimitri Tsampiras	UI/UX Verification Testing
William Lee	Code Quality and Optimization Verification

Table 1: Team Member Roles

2.3 Relevant Documentation

1. SRS
2. MG
3. MIS

3 Plan

This section contains the criteria for reviewing and verifying the various artifacts involved with the project.

3.1 Verification and Validation Team

3.2 SRS Verification Plan

The SRS will be evaluated by team members and classmates as the project progresses. It will also be verified through testing with a traceability matrix. GitHub issues left by other groups will be carefully considered and addressed. The following checklist will be used:

- ☐ Can the system clearly demonstrate each requirement?
- ☐ Can the system demonstrate each requirement outside of a testing or debugging environment?
- ☐ Does each requirement map to one or more tests?

3.3 Design Verification Plan

The project design will be manually reviewed by each team member and classmates. The following checklist will be used:

- ☐ Does the system design allow for each functional and non functional requirement to be met?
- ☐ Does the chosen architecture(s) make sense for the given use cases? Does it allow for easy scalability?
- ☐ Does the design follow appropriate design principles (e.g. low coupling, high cohesion, separation of concerns)?

3.4 Verification and Validation Plan Verification Plan

The Verification and Validation Plan artifact will be read and reviewed by each team member according to the following checklist:

- ☐ Are there any missing or empty sections?
- ☐ Is each section complete with sufficient detail?
- ☐ Are there any contradictions between sections?
- ☐ Is the table of contents present and reflective of the document's sections?
- ☐ Is the revision history up to date?
- ☐ Are there any typos in the document?

GitHub issues left by other groups will be carefully considered and addressed.

3.5 Implementation Verification Plan

Implementation Verification will be done according to the functional and non-functional tests listed in this document. Code additions will be thoroughly reviewed by team members in feature branches before merging with the main branch. Linters and formatters will be utilized to ensure code consistency.

3.6 Automated Testing and Verification Tools

See Section 3.5 of Problem Statement.

3.7 Software Validation Plan

No external data will be used for validation. In addition to the SRS verification checklist, meetings will be held with stakeholders to ensure requirements cover all desired tests.

4 System Test Description

4.1 Tests for Functional Requirements

The following tests correspond to functional requirements outlined in the SRS.

4.1.1 Workout Routine Tests

1. **test-WR-1**:A Workout Routine can be created

   Control: Manual

   Initial State: Application is running.

   Input: User inputs required data for a workout routine.

   Output: A workout routine is stored in the database and is accessible to the user that created it. If the user determined that the workout to be public then it should be publicly visible.

   Test Case Derivation: On a successful creation, the workout routine should be placed into the database.

   How test will be performed: This test can be preformed with a manual insertion of creating a workout routine under a test user. These results can be determined by viewing the database and scope of visibility of the created routine.

   Requirement(s): R1

2. **test-WR-2**: Editing a Workout Routine

Control: Manual

Initial State: Application is running with an existing workout routine.

Input: User edits a workout routine with new values.

Output: A workout routine is updated with new values in the database and new values are visible to accessible users.

Test Case Derivation: On a successful edit, the workout routine should be updated with new values database.

How test will be performed: This test can be preformed with a manual edit of a workout routine under a test user.

Requirement(s): R1

4.1.2 Exercise Tests

3. **test-EX-1**:Adding an Exercise to a Workout Routine

   Control: Manual

   Initial State: Application is running and a workout routine is in the process of being created or edited.

   Input: A user created exercise with the parameters required for the exercise.

   Output: The workout routine should include the added exercise.

   Test Case Derivation: Because exercises are bounded to a specific workout routine, the routine that the exercise was created under should include the new exercise.

   How test will be performed: Given a workout routine, an exercise will be manually added to determine if it is linked and added to the workout routine.

   Requirement(s): R2

4. **test-EX-2**:Removing an Exercise from a Workout Routine

   Control: Manual

   Initial State: Application is running and a workout routine is in the process of being created or edited.

Input: The user chooses to remove an exercise.

Output: Some feedback indicating that an exercise has been removed. The workout routine no longer contains the removed exercise.

Test Case Derivation: On removal of an exercise, there should be an indication towards the user notifying them of the removal. The exercise should also be deleted from the workout routine.

How test will be performed: Given a workout routine, an exercise will be manually removed and checked to ensure that it no longer exists within the given workout routine.

Requirement(s): R2

5. **test-EX-3**:Limiting Exercises to a Workout Routine

   Control: Manual

   Initial State: Application is running and a workout routine is in the process of being created or edited.

   Input: The number of exercises required to reach the limit of exercises per workout routine.

   Output: A notification to the user, before the limiting exercise notifying them of the limit, and preventing them from adding another exercise.

   Test Case Derivation: The user should be aware of the exercise limit per workout routine and should not be able to exceed it.

   How test will be performed: The test will be preformed by manually added exercises to a workout routine until the limit. Then upon adding the limiting input, a notification is expected.

   Requirement(s): R2

4.1.3 Quantifier Tests

6. **test-QT-1**:Adding Quantifiers to an Exercise

   Control: Manual

   Initial State: The application is running and an exercise is in the process of being created or edited.

   Input: A number and unit of measurement describing an exercise.

Output: The exercise now holds the given quantifier and is displayed to the user.

Test Case Derivation: On adding a quantifier to an exercise, there should be a unit of measurement and value associated with it.

How test will be performed: This test will be executed by adding a quantifier to various types of exercises to ensure they correctly measure the exercise.

Requirement(s): R3

7. **test-QT-2**:Removing Quantifiers from an Exercise

   Control: Manual

   Initial State: The application is running and an exercise is in the process of being created or edited.

   Input: Removal of a quantifier.

   Output: The exercise no longer holds a quantifier and is not displayed to the user.

   Test Case Derivation: On removing a quantifier to an exercise, it should no longer be visible to the user.

   How test will be performed: This test will be executed by removing a quantifier from an exercises to ensure it does not display any quantifying information.

   Requirement(s): R3

8. **test-QT-3**:Editing Quantifiers of an Exercise

   Control: Manual

   Initial State: The application is running and an exercise is in the process of being created or edited.

   Input: A number and unit of measurement describing an exercise.

   Output: The exercise now holds the updated quantifier.

   Test Case Derivation: On updating a quantifier to an exercise, there should be a new unit of measurement and/or value associated with it.

   How test will be performed: This test will be executed by editing a quantifier and ensuring the values are updated on completion of the edit.

Requirement(s): R3

4.1.4 Publicity Tests

9. **test-PB-1**:Publicizing a Workout Routine

   Control: Manual

   Initial State: The application is running and a private workout routine is created.

   Input: An edit or addition to a workout routine to make it public.

   Output: The workout routine is declared public in the database and is now visible to all users.

   Test Case Derivation: This is the expected output because if the workout routine was not public before this action then it should not be visible to users. Once updating the publicity, the database should be updated and the routine should be publicly available

   How test will be performed: This test will be done by manually changing the state of a workout routine and checking for a database update and viewing the routine under a different test-user.

   Requirement(s): R4

10. **test-PB-2**:Privatizing a Workout Routine

    Control: Manual

    Initial State: The application is running and a public workout routine is created.

    Input: An edit or addition to a workout routine to make is private.

    Output: The workout routine is declared private in the database and is no longer visible to any user except the creator.

    Test Case Derivation: By privatizing a routine, it should no longer be accessible to any other user than the creator. There should also be a database update to signify this.

    How test will be performed: This test will be done by manually changing the state of a workout routine and checking for a database update and viewing the routine under a different test-user.

    Requirement(s): R4

4.1.5 Workout Routine Saving Tests

11. **test-WS-1**:Saving a Public Workout Routine

    Control: Manual

    Initial State: The application is running and there is a public workout routine created by another user.

    Input: The other user workout routine is saved.

    Output: The workout routine is now visible and accessible under the saved workout routines.

    Test Case Derivation: On saving a workout routine, it should be stored somewhere the user can access it for later use.

    How test will be performed: This test will be done by creating a public workout routine on one test-user. On a different test user this routine will be saved. The saved workout routines should then include the newly saved one.

    Requirement(s): R5

4.1.6 Browsing Workout Routine Tests

12. **test-BS-1**:Browsing Workout Routines

    Control: Manual

    Initial State: The application is running and multiple workout routines are created.

    Input: navigation movements to view the public workout routines.

    Output: Multiple public workout routines should be displayed.

    Test Case Derivation: On making a routine public, it should be available to all users and also the browsing page.

    How test will be performed: By first adding multiple public test routines, then checking the public workout routines by browsing. There should exist routines to access and view.

    Requirement(s): R6

13. **test-BS-2**:Search Workout Routines

    Control: Manual

    Initial State: The application is running and multiple workout routines are created.

    Input: Search string inputs to view the public workout routines.

    Output: Public workout routines that match the search criteria should be displayed.

    Test Case Derivation: All workout routines that match a search, should be displayed.

    How test will be performed: By first adding multiple public test routines, then preforming a search, the routines that match the search criteria should be displayed.

    Requirement(s): R6

4.1.7 User Profile Tests

14. **test-UP-1**:Creating a User Profile

    Control: Manual

    Initial State: The application is running.

    Input: The required parameters for creating a profile.

    Output: The created profile is stored in a user database and the user should be logged into their profile.

    Test Case Derivation: On creating a profile the database must store the information. To go along with this the user must maintain the login status of the created profile.

    How test will be performed: On launching the application without a created profile a profile will be created. The database will then be checked to ensure the proper data matches.

    Requirement(s): R7

15. **test-UP-2**:Viewing Other User Profile

    Control: Manual

Initial State: The application is running and there exists another user profile.

Input: Search criteria for the searched user profile.

Output: A user profile is displayed with their public routines, fitness goals, and other public profile data.

Test Case Derivation: On searching for another user the data shown should only be what is made public.

How test will be performed: By creating a test-user profile then searching for it. The profile should display all public information and hide non-public information.

Requirement(s): R8

4.1.8 Fitness Goal Tests

16. **test-FG-1**:Creating a Fitness Goal

    Control: Manual

    Initial State: The application is running and a profile has been created or is in the process of creation.

    Input: A string parameter for a fitness goal and publicity options for the goal.

    Output: The fitness goal for the user is now updated in the database and is saved under their profile.

    Test Case Derivation: The fitness goals are specific to a given user so the data for fitness goals are stored under their profile data.

    How test will be performed: A test user will add a fitness goal and check if it is displayed on their profile.

    Requirement(s): R9

17. **test-FG-2**:Progressing a Fitness Goal

    Control: Manual

    Initial State: The application is running and a fitness goal is created.

    Input: Progress points towards a given fitness goal at a given date.

Output: The progress displayed towards a fitness goal should be visually updated and numerically updated in the database with a date.

Test Case Derivation: The data for a fitness goal must be updated in the database for storage. To follow this, to goal must also be visually updated accordingly to signify progress towards the goal at the given date.

How test will be performed: A test-user will create a fitness goal and add progress points towards the goal. There should be a database update and visual update to signify this change.

Requirement(s): R10, R11

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and Feel Testing

1. **test-LF-1**: Style

   Type: Manual.

   Initial State: The application is running.

   Input/Condition: A survey is conducted with various user interactions with the application.

   Output/Result: The survey results will record the overall style quality targeting areas such as simplicity, cleanliness and aesthetic appeal.

   How test will be performed: This test will be preformed by conducting a survey to determine if test users think that the products appearance is minimal and straight forward.

   Requirement(s): NFR1

4.2.2 Usability and Humanity Tests

2. **test-UH-1**: Text Sizing and Visibility.

   Type: Manual

   Initial State: The application is running.

   Input/Condition: Various application screens are shown to users of diverse demographics.

Output/Result: The survey results will give an indication for how visible application text is, and if any specific text needs adjustment.

How test will be performed: A survey is conducted where users of various demographics interact with the application and are asked how visible application text is.

Requirement(s): NFR2

3. **test-UH-2**: Text Language.

   Type: Manual

   Initial State: The application is running.

   Input/Condition: User click input of language from dropdown menu.

   Output/Result: Language of the application changes to their users' chosen language.

   How test will be performed: Language dropdown will ba manually pressed and a language will be chosen. The application should change all text to the new selected language.

   Requirement(s): NFR3

4. **test-UH-3**: Learnability.

   Type: Manual

   Initial State: The application is running.

   Input/Condition: The system is given to a group of users to use for a predetermined time period.

   Output/Result: The survey will give an indication of how easy or hard the system is to learn.

   How test will be performed: Each user will be tasked with performing a specific task within a set amount of time, and report how easy or difficult the task was to learn to perform.

   Requirement(s): NFR4

5. **test-UH-4**: Understandability.

   Type: Manual

   Initial State: The application is running.

Input/Condition: Various system outputs are shown to a group of users in a survey.

Output/Result: The survey will give an indication of how easy or hard system output is to understand.

How test will be performed: Each user in the survey is shown various system outputs and asked how easy the outputs are to understand.

Requirement(s): NFR5

6. **test-UH-5**: Hearing and Audio considerations.

   Type: Manual

   Initial State: The application is running.

   Input/Condition: Each system sound is played to a group of users in a survey.

   Output/Result: The survey will give an indication on how audible/pleasant the system sounds are.

   How test will be performed: System sounds will be played to users in a survey, who will respond will their perception of the audio's quality.

   Requirement(s): NFR6

7. **test-UH-6**: Use of Colour and Contrast.

   Type: Manual

   Initial State: Application is running.

   Input/Condition: Users are shown images of various application views.

   Output/Result: The survey will give an indication of how visible application views are with respect to colour and contrast.

   How test will be performed: Users will be shown images of various application views in a survey and asked to judge the visibility based on colour and contrast.

   Requirement(s): NFR7

4.2.3 Performance Tests

8. **test-PF-1**: Speed and Latency.

   Type: Dynamic

   Initial State: Application will be running.

   Input/Condition: String representing workout name and workout information.

   Output/Result: Inputted information updated existing program or new program created and displayed.

   How test will be performed: Program will be inputted and updated utilizing a testing framework that will time whether starting to completion took 10 seconds or less.

   Requirement(s): NFR8

9. **test-PF-2**: Accuracy and Precision of Quantifiers.

   Type: Manual

   Initial State: The application will be running and on the post search page.

   Input/Condition: Search string inputs to view public posts.

   Output/Result: Search results with ratings at an accuracy of 2 decimal places.

   How test will be performed: A string will be used to search public posts and all posts will be checked for two decimal places.

   Requirement(s): NFR9

10. **test-PF-3**: Availability and Uptime

    Type: Manual

    Initial State: A server will be running and hosting our app to mobile clients.

    Input/Condition: Apache JMeter will perform a variety of tasks across multiple clients.

    Output/Result: The application will remain live for %95+ of the duration of the testing.

How test will be performed: Apache JMeter will perform a variety of tasks across multiple users to simulate strain for an extended duration. Uptime.com will be used to check the uptime of the application over the test period.

Requirement(s): NFR10

11. **test-PF-4**: User Capacity.

    Type: Automatic

    Initial State: A server will be running and hosting our app to mobile clients.

    Input/Condition: Apache JMeter will begin running 100 clients and send each client an update.

    Output/Result: The server will receive all responses within 5 seconds.

    How test will be performed: JMeter will perform capacity testing to test if the server can handle up to 100 clients. It will attempt to send an update to each client and receive a response.

    Requirement(s): NFR11

12. **test-PF-5**: Scalability of User Capacity.

    Type: Automatic

    Initial State: A server will be running and hosting our app to mobile clients a year after our release.

    Input/Condition: Apache JMeter will begin running 100 clients.

    Output/Result: All client updates will be successfully performed.

    How test will be performed: JMeter will have all 1000 clients perform simultaneous user updates and tasks and expect task completion within 10 seconds.

    Requirement(s): NFR12

4.2.4 Operational and Environment Tests

13. **test-OE-1**: Supported Systems.

    Type: Manual

Initial State: The application is closed and the mobile device is turned on.

Input/Condition: The device downloads and launches the application. All required hardware features are tested on iOS and Android devices.

Output/Result: There should be complete compatibility for both supported systems without any operation system errors.

How test will be performed: This test will be done by downloading and running all features on both an iOS and Android device ensuring there are no operating system errors.

Requirement(s): NFR13

4.2.5 Maintainability and Support Tests

14. **test-MS-1**: Maintenance.

    Type: Manual

    Initial State: The application is running.

    Input/Condition: The server will be set to activate maintenance mode at a set date (within 24 hours).

    Output/Result: Users currently in the application will see a maintenance alert.

    How test will be performed: A simulated maintenance event will be triggered and application users will verify that a maintenance banner is shown to clients.

    Requirement(s): NFR14

4.2.6 Security Tests

15. **test-SEC-1**: Private and Public Details.

    Type: Manual

    Initial State: The application will be running and logged into a user profile.

    Input/Condition: Navigate to public user profiles.

Output/Result: Other users profiles should not display any private details such as their passwords or email information.

How test will be performed: Search for other user profiles and verify that no private information is being displayed on their profile to other users.

Requirement(s): NFR15

16. **test-SEC-2**: Passwords.

    Type: Manual

    Initial State: User is logged out (i.e. not authenticated and is forbidden from accessing account).

    Input: String parameter as password alongside corresponding account.

    Output: Assuming username exists - inputted password is tested against hashed version of actual user password. If verification is successful, user is logged in.

    Test Case Derivation: The user is redirected to indicate a successful login. User is notified in the case of an unsuccessful login (e.g. wrong password).

    How test will be performed: Test whether or not username exists. Test inputted password against account password. Test if correct error message is presented.

    Requirement(s): NFR16

17. **test-SEC-3**: Client Server Privacy.

    Type: Manual

    Initial State: User request is made.

    Input: User request has authorization token in request header .

    Output: Server process authorization token. If authorized, server request will continue. If not, server returns 403 server error (forbidden).

    Test Case Derivation: Test authencation token against server token secret.

    How test will be performed: Extract authorization token from header. If header exists, run token verification function.

    Requirement(s): NFR17

18. **test-SEC-4**: Data storage and logging.

    Type: Functional

    Initial State: Application is running

    Input/Condition: A valid data entry for a workout routine or user is entered into the database.

    Output/Result: The entry is successfully added and there is a database recorded log of the interaction.

    How test will be performed: As a data point gets added, there should exist an output log for each interaction, this will be checked by viewing database logs. This will only look at valid data entries because these should get filtered out via unit testing.

    Requirement(s): NFR18

19. **test-SEC-5**: Data Backups.

    Type: Functional, Manual

    Initial State: Application is running

    Input/Condition: Data entries exist in the database and the database fails.

    Output/Result: There should exist a backup of the database from a recent saved state.

    How test will be performed: This can be preformed manually by shutting down the database and checking if the backup is used. This can also be preformed by checking if the backup exists and is refreshed occasionally.

    Requirement(s): NFR18

4.2.7 Cultural Requirements Tests

20. **test-CR-1**: Profanity and Inappropriate Language.

    Type: Functional

    Initial State: The application is running.

    Input/Condition: A user posts content which contains profanities.

Output/Result: The profanity is filtered out by the system, and the user is suspended from the application if they are a repeat offender.

How test will be performed: The action of posting a piece of content will be manually simulated, and content will be posted containing profanities. The content will be checked to verify filtering of profane words, and the user's account status will be checked to verify suspension on a repeat offense.

Requirement(s): NFR19

21. **test-CR-2**: Reporting Offensive Language.

    Type: Manual

    Initial State: The application is running.

    Input/Condition: A user is viewing the context options for interacting with a piece of recommended content on their feed (which they may find offensive).

    Output/Result: The user is able to report the piece of offensive content, removing it from their feed. The content should no longer be visible on the user's feed.

    How test will be performed: The action of interacting with context options for a piece of recommended content will be manually simulated, and an attempt will be made to report the content, making it no longer visible on the user's feed.

    Requirement(s): NFR20

4.2.8 Legal Requirements Tests

22. **test-LR-1**: Age and Gender Use.

    Type: Manual

    Initial State: The application is running.

    Input/Condition: The user is personalizing their profile.

    Output/Result: The user has the ability to specify their gender, age, and race.

How test will be performed: The profile personalization process will be manually simulated, and an attempt will be made to specify a sample gender, age, and race.

Requirement(s):NFR21

23. **test-LR-2**: Data Protection.

    Type: Static

    Initial State: The application database contains sensitive user data.

    Input/Condition: N/A

    Output/Result: All usage and storage of user data is compliant with the Data Protection Act.

    How test will be performed: The logic used to store data will be analyzed to ensure that no user data is stored unlawfully or unfairly, data will be kept only as long as the user consents, and that no sensitive data is transferred across networks in an unencrypted state.

    Requirement(s): NFR22

4.3 Traceability Between Test Cases and Requirements

5 Traceability Matrices and Graphs

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
WR1	X										
WR2	X										
EX1		X									
EX2		X									
EX3		X									
QT1			X								
QT2			X								
QT3			X								
PB1				X							
PB2				X							
WS1					X						
BS1						X					
BS2						X					
UP1							X				
UP2								X			
FG1									X		
FG2										X	X

Table 2: Functional System Tests to Functional Requirement Matrix

6 Unit Test Description

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

6.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

	NFR1	NFR2	NFR3	NFR4	NFR5	NFR6	NFR7	NFR8	NFR9	NFR10	NFR11
LF1	X										
UH1		X									
UH2			X								
UH3				X							
UH4					X						
UH5						X					
UH6							X				
PF1								X			
PF2									X		
PF3										X	
PF4											X

Table 3: Non-Functional System Tests to Non-Functional Requirement Matrix

	NFR12	NFR13	NFR14	NFR15	NFR16	NFR17	NFR18	NFR19	NFR20	NFR21	NFR22
PF5	X										
OE1		X									
MS1			X								
SEC1				X							
SEC2					X						
SEC3						X					
SEC4						X					
SEC5							X				
CR1								X			
CR2									X		
LR1										X	
LR2											X

Table 4: Non-Functional System Tests to Non-Functional Requirement Matrix

6.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

6.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

6.2.2 Module 2

...

6.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

6.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage etc. You should look to identify at least one item for each team member.

7.3 Required Skills

- Skill: **Static Testing**
  Rationale: Static testing will be a very important part of successfully completing the verification and validation plan for many reasons. Static testing involves matching the requirements to the code. It also involves looking into code error and structure. Having a good structure to the code will help with matching requirements stated in the SRS and validation plan resulting in a more sound product.
  Team member: William Lee
- Skill: **Functional Testing**
  Rationale: Functional testing is a crucial component in a verification and validation plan. Functional testing is used as a form of black-box testing to verify that the system's components provide the expected output, as per the requirements set forth in the software requirements specification.
  Team member: Bassel Rezkalla
- Skill: **Integration Testing**
  Rationale: Integration testing is a quintessential process in any software application. Integration testing tests the interaction between components in a system as they are actually deployed. These tests will be especially important for testing user requests and database transactions.
  Team member: Jared Bentvelsen

- Skill: **Dynamic Testing**
  Rationale: Dynamic testing is vital to the verification and validation plan. Dynamic testing is executing test cases to determine and analyze how the system and variables that are not constant and change with time react.
  Team member: Yuvraj Randhawa
- Skill: **Automated Load Testing**
  Rationale: Setting up automated tests to load/capacity test the mobile application is crucial in achieving verification and validation because it is needed for meeting capacity requirements.
  Team member: Matthew McCracken
- Skill: **UI/UX Testing**
  Rationale: UI/UX testing is a very overlooked aspect of testing. It ensures that the UI renders as intended, at the correct times, and with the correct data. Testing UI can help verify that all component states (pending, error, complete, etc.) are accounted for and function correctly.
  Team member: Dimitri Tsampiras

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

7.4 Acquiring Knowledge and Mastering skills

- Skill: **Static Testing Knowledge**
  Approach 1: Research and contrast various static testing methods such as walkthroughs, code reviews and inspections.
  Approach 2: Refer to previous notes on Software Testing 3S03 at McMaster University regarding static testing methodology.
  Verdict: William will use approach 1 to research what methods are the best catered to this project because often tutorials for testing methods go in depth on strategy for approaching static testing. With approach 2, there is only a surface knowledge level of static testing learned from the course. However approach 2 can be used as a good starting point.

- Skill: **Functional Testing Knowledge**
  Approach 1: Review and practice black-box testing and functional testing techniques from SWFRENG 3S03: Software Testing.
  Approach 2: Examine the functional testing techniques used by other social applications that share similarities in feature functionality and requirements with our application.
  Verdict: Bassel will use Approach 2 to learn about writing and implementing functional tests as it may provide information that is more relevant and specifically applicable to an application such as ours.
- Skill: **Dynamic Testing Knowledge**
  Approach 1: Consult notes from previous courses that went over software testing (SFWRENG 3S03: Software Testing) and revisit the relevant course assignments.
  Approach 2: Utilize online resources to learn and improve dynamic testing skills and abilities.
  Verdict: Yuvraj will use Approach 1 to improve his Dynamic Testing knowledge. He has chosen this strategy because the course provided relevant and detailed notes from a reputable source with a lot of background information to aid in the learning.
- Skill: **Integration Testing Knowledge**
  Approach 1: Review and practice integration testing from SWFRENG 3S03: Software Testing.
  Approach 2: Read online articles on integration testing and consult integration tests in well known open source products.
  Verdict: Jared will use Approach 2 to learn about integration testing since SFWRENG 3S03 did not cover it in sufficient detail to expand his skills.
- Skill: **Automated Load Testing Knowledge**
  Approach 1: Consult former co-workers who worked on mobile application load testing to gather best practices and tips.
  Approach 2: Engage with online articles and forums that discuss using the specific load testing tools that we will use.
  Verdict: Matthew will use Approach 2 to learn about load testing because this will give a broad range of knowledge to complete many different types of mobile tests.
- Skill: **UI/UX Testing Knowledge**

Approach 1: Utilize user testing to determine if actual user experience is as intended. Tools such as surveys and questionaires can be used.
Approach 2: Use UI testing libraries learn how to verify and validate the system's UI components.
Verdict: Dimitri will use Approach 2 to ensure proper data and component states are being displayed correctly, and at correct times.