**MUNSSN**

**Software Test Plan**

**March 17, 2017**

**Submitted By:**

Floris Bouman,

Josh Davis, Bradley Gavan,  
Matthew Randell,  
Curtis White

**Table of Contents**

[**1**](#_gjdgxs) **Introduction 3**

[1.1 Team Interaction 3](#_30j0zll)

[**2**](#_1fob9te) **Test Objective 3**

[**3**](#_3znysh7) **Process Overview 3**

[**4**](#_2et92p0) **Testing Process 4**

[**5**](#_tyjcwt) **Testing Strategy 5**

[5.1 Unit Testing 7](#_3dy6vkm)

[*5.1.1*](#_1t3h5sf) *White Box Testing 7*

*5.1.2* *Black Box Testing 10*

[5.2 Integration Testing 12](#_4d34og8)

[*5.2.1*](#_2s8eyo1) *Incremental Testing 12*

[5.3 System Testing 12](#_17dp8vu)

[*5.3.1*](#_3rdcrjn) *Function Validation Testing 12*

[*5.3.2*](#_26in1rg) *Performance testing 13*

[**6**](#_lnxbz9) **Entry and Exit Criteria 13**

[6.1 Unit Testing 14](#_35nkun2)

*6.1.1* *Black Box Phase 14*

*6.1.2* *White Box Phase 15*

[6.2 Integration Test 16](#_1ksv4uv)

[*6.2.1*](#_44sinio) *Integration Test Entry Criteria 16*

[*6.2.2*](#_2jxsxqh) *Integration Test Exit Criteria 16*

[6.3 System Test 17](#_z337ya)

[*6.3.1*](#_3j2qqm3) *System Test Entry Criteria 17*

[*6.3.2*](#_1y810tw) *System Exit Criteria 17*

[6.4 Shipping or Live Release 18](#_4i7ojhp)

*6.4.1* *Shipping/Live Release Entry Criteria 18*

*6.4.2* *Shipping/Live Release exit Criteria 18*

[**7**](#_2xcytpi) **Bug Tracking/ Bug Process 18**

7.1 Various Roles in Bug Resolution 19

[7.2 Bug Report Form 20](#_1ci93xb)

[**8**](#_3whwml4) **Roles and Responsibilities 21**

8.1 Development Team 21

8.2 Testing Team 21

[**9**](#_2bn6wsx) **Test Schedule 21**

[**10**](#_qsh70q) **Deliverables 23**

[**11**](#_3as4poj) **References 24**

# Introduction

This document is an overview of our testing strategy for the MUNSSN web application. Its objective is to describe standards for implementing and testing modules. This document will address the different procedures involved in testing the proper functionality of each unit of the MUNSSN. We will test modules as both white boxes and black boxes to ensure that the system works correctly in code, but also works correctly for users who have no understanding of the MUNSSN code. Throughout the testing process we will be applying the test documentation specifications described in the IEEE Standard 829-1983 for Software Test Documentation.

## Team Interaction

The following describes the level of team interaction necessary to have a successful product.

* Team members will work closely with one another to ensure that each module meets specifications for completeness, high code quality, and correct functionality. Each team member should raise quality issues and concerns during meetings to ensure timely corrections.
* Since the application interacts with a back-end system component, the team will need to include a plan for integration testing. Integration testing must be executed successfully prior to system testing.

# Test Objective

The objective of our test plan is to find and report as many bugs as possible to improve the integrity of our program. Each module will be first tested on its own to ensure that it functions correctly as a unit. It will then be integrated into the overall system and tested in a system-wide context to ensure it does not break the integrity of other modules. Our user interface is designed to be user-friendly so that testers can log in to the system quickly and begin using all available modules as a web service. The system will be tested in multiple web browsers so that the user interface can be checked for consistency in different environments.

# Process Overview

The following represents the overall flow of the testing process:

1. Identify the requirements to be tested. All test cases are to be derived from the Project Specification.
2. Identify which particular test(s) will be used to test each module.
3. Review the test data and test cases to ensure that the module has been thoroughly verified and that the test data returned from each test is appropriate for the module.
4. Identify the expected results for each test.
5. Describe the test case configuration, test data, and expected results.
6. Perform the test(s).
7. Document the test data, test cases, and test configuration used during the testing process.
8. Successful unit testing is required before the unit is eligible for component integration/system testing.
9. Unsuccessful testing requires that a report be submitted through Git with a pull request so that the issue can be analyzed and corrected. This document shall describe the test case, the problem encountered, its possible cause, and the sequence of events that led to the problem.
10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

# Testing Process

The testing process will follow this schema:

**a.** **Organize Project:** Thisinvolves creating a Test Plan and assigning responsibilities.

**b.** **Design/Build System Tests:** Thisinvolves identifying Test Cases and Expected Results.

**c.** **Design/Build Test Procedures:** Thisincludes setting up procedures such as Error Management systems and Status reporting.

**d.** **Build Test Environment:** Thisincludes coding the modules and setting up the test environment.

**e. Execute System Tests:** The tests identified in the Design/Build Test Procedures will be executed. All results will be documented and Bug Reports will be submitted to Git with a pull request.

**f.** **Signoff:** The project will be signed off when all minimally-required test results have been achieved.

# Testing Strategy

The following outlines the types of testing that will be done for each module, the steps involved and the expected results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Curtis White | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | | 1 | |
| **Test Case Name** | | Registration | |
| **Test Case Description** | | **Attempt to register for an account on the MUNSSN. Ensure that all values are added to the database, and that the system outputs the correct messages for each type of the following scenarios:**   1. **User information not in correct format** 2. **User already exists** 3. **Required values missing** 4. **E-mail invalid** 5. **User successfully added** | |
| **Item(s) to be tested** | | | |
| 1 | User Schema (user.js) | | |
| 2 | Registration module (users.js) | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String name,  String username,  String email,  String password | | | User successfully added to database  If this fails, an error message should be displayed. |
| **Procedural Steps** | | | |
| 1 | Intentionally input correct values, then click “Sign Up.” User should be added to database and “Registration Complete” message displayed. | | |
| 2 | Intentionally input username or e-mail that already exists, then click “Sign Up.” “User already exists” message displayed. | | |
| 3 | Intentionally input invalid e-mail, then click “Sign Up.” “E-mail must be MUN e-mail” message displayed. | | |
| 4 | Intentionally input mismatched passwords, then click “Sign Up.” “Passwords do not match. Try again.” message displayed. | | |
| 5 | Input no values in at least one required field, then click “Sign Up.” “Fields missing” message displayed. | | |
| 6 | Input values with invalid characters, then click “Sign Up.” “Fields must contain only alphanumeric values” message displayed. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Curtis White | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | | 2 | |
| **Test Case Name** | | Login | |
| **Test Case Description** | | **Attempt to login to ensure that users can obtain access to their account and only their account information. Ensure that invalid username and password combinations are properly rejected. Ensure that user is redirected to timeline.** | |
| **Item(s) to be tested** | | | |
| 1 | Login.js | | |
| 2 | Users.js | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String username,  String password | | | Upon accepted login, routing to the user's timeline page which grants access to their stored information. Redirection to timeline.  Upon failure, display an appropriate error message. |
| **Procedural Steps** | | | |
| 1 | Type in correct username and password combination, click login button. User should be directed to timeline. | | |
| 2 | Type in incorrect username and password combination, click login button. Attempt should be rejected and an error message displayed. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Curtis White | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | | 3 | |
| **Test Case Name** | | Logout | |
| **Test Case Description** | | **Ensure that clicking “Logout” closes the database and routes the user to the login page.** | |
| **Item(s) to be tested** | | | |
| 1 | Users.js | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| Logout hyperlink signal. | | | User redirected to login page and database session closed. |
| **Procedural Steps** | | | |
| 1 | Click “Logout” on user timeline. | | |
| 2 | Database should close and user redirected to login page. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Curtis White | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | | 4 | |
| **Test Case Name** | | Timeline | |
| **Test Case Description** | | **Check that timeline appears as it should according to intended CSS. Ensure that expanding timeline size pulls and displays posts from the user's database.** | |
| **Item(s) to be tested** | | | |
| 1 | Index.handlebars | | |
| 2 | Posts.js | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| User account ID  Post | | | Timeline with one empty slot appears when user logs in. One “Add a Post” section appear if no posts exist. As posts are added, timeline can be expanded. |
| **Procedural Steps** | | | |
| 1 | Log in, user should be directed to timeline. If no posts, a single “Add a post” section is available. | | |
| 2 | When more posts are added, click “Expand” to make older posts appear at the bottom of the timeline. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Bradley Gavan | |
| **Test Type** | | Module Testing - White Box | |
| **Test Case Number** | | 5 | |
| **Test Case Name** | | Group Creation | |
| **Test Case Description** | | **Attempt to create a group on the MUNSSN. Ensure that all values are added to the database, and that the system outputs the correct messages for each type of the following scenarios:**     1. **Group information not in correct format** 2. **Group of that name already exists** 3. **Required values missing** 4. **Group successfully created** | |
| **Item(s) to be tested** | | | |
| 1 | Groups.js - createGroup function | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String Group Name  String Course  String Admin  String Privacy | | | The creation of the group in the database with all the values entered correctly  An error if a group of the same name already exists |
| **Procedural Steps** | | | |
| 1 | Create a group, leave out one of the inputs. An error should be thrown indication that a value is missing. | | |
| 2 | Create a group, give all the necessary inputs. The group should now be in the database, a confirmation message should be delivered. | | |
| 3 | Find the group in the database. Check that each of the values in the databases match the inputs given to the constructor | | |
| 4 | Attempt to create another group with the same name as the first group. An error should be generated. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Bradley Gavan | |
| **Test Type** | | Module Testing - White Box | |
| **Test Case Number** | | 6 | |
| **Test Case Name** | | Add admin | |
| **Test Case Description** | | **Attempt to add an admin to a group on the MUNSSN. Ensure that the new admin is added to the set of admins and that all the previous admins are still in the set. Do not allow for duplicates in the array.** | |
| **Item(s) to be tested** | | | |
| 1 | Groups.js - addAdmin function | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String user | | | The user added to the array of admins for the group.  An error if the user is already an admin |
| **Procedural Steps** | | | |
| 1 | Create a group, verify that it is added to the database. | | |
| 2 | Have a array of unique users, add them as admins. | | |
| 3 | Compare the original array of unique users to the array of admins from the group. Ensure that they are equal. | | |
| 4 | Attempt to add a user as an admin who is already an admin. An error should be generated. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Bradley Gavan | |
| **Test Type** | | Module Testing - White box | |
| **Test Case Number** | | 7 | |
| **Test Case Name** | | Adding and Removing Users to/from a Group | |
| **Test Case Description** | | **Attempt to add and remove user to/from a group on the MUNSSN. Ensure that users are added/removed as specified. Do not allow duplicates to be added. The arrays should not be modified other than the specified additions/removals.** | |
| **Item(s) to be tested** | | | |
| 1 | Groups.js - addUser and removeUser functions | | |
| 2 |  | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String user | | | For addUser, the specified user should be added to the list of users in the database for that group. An error should be generated if that user is already a member of the group.  For removeUser, the specified user should be removed from the array of users in the database for that group. An error should be generated if that user is not a member of the group. |
| **Procedural Steps** | | | |
| 1 | Create a group, verify that it added to the database. | | |
| 2 | Have an array of unique users. Add them, one at a time, to the group array “Users”. | | |
| 3 | Compare the original array of unique users to the array of users from the group. Ensure that they are equal. | | |
| 4 | Attempt to add a user from the list that has already been added. An error should be thrown. | | |
| 5 | Remove all the users, one at a time, from the group array “Users”. Ensure that it is now empty. | | |
| 6 | Attempt to remove a user from the group array “Users”. An error should be thrown. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Bradley Gavan | |
| **Test Type** | | Module Testing - White Box | |
| **Test Case Number** | | 8 | |
| **Test Case Name** | | Privacy Setting Test | |
| **Test Case Description** | | **Attempt to change the privacy setting of a group on the MUNSSN. Ensure that the setting is changed as given and that only the given privacy setting are taken as inputs. Examples of privacy settings are “Open”, “Restricted”, “Friends Only”.** | |
| **Item(s) to be tested** | | | |
| 1 | Groups.js - setPrivacy function | | |
| 2 |  | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String privacySetting - must be an approved privacy setting | | | The privacy value in the database document for the specified is update to the input value.  Error if input is not an accepted value. |
| **Procedural Steps** | | | |
| 1 | Create a group, verify that it added to the database. | | |
| 2 | Use the setPrivacy function to change the privacy setting to an accepted value. Check that the value is updated correctly. Do this for each of the accepted values. | | |
| 3 | Attempt to change the privacy setting to an unaccepted value. Expect to catch an error. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Bradley Gavan | |
| **Test Type** | | Module Testing - White Box | |
| **Test Case Number** | | 9 | |
| **Test Case Name** | | Sending and Deleting Messages | |
| **Test Case Description** | | **Attempt to send messages from one user to another on the MUNSSN. Ensure that the message appears as written in the input in both sender and the receivers inbox.** | |
| **Item(s) to be tested** | | | |
| 1 | Messaging.js - sendMessage()  Messaging.js - deleteConversation() | | |
| 2 |  | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String user - who’s receiving the message  String message - the message to be sent | | | The message pushed to the messages array in the document containing the conversation between the two users. A new document should be created if a conversation does not already exist.  Error if the user does not exist |
| **Procedural Steps** | | | |
| 1 | Create two users in the database, user1 and user2. | | |
| 2 | Send a message from user1 to user2 using the sendMessage(user2, msg) function | | |
| 3 | Retrieve the db.conversations document between user1 and user2. | | |
| 4 | Verify that the message from the conversation document is the same as the input message. Also verify that the sender is user1 and the recipient is user2. | | |
| 5 | Delete the conversation using the deleteConversation(user1, user2) function. Now verify that there is a null result when searching for a conversation between user1 and user2. | | |
|  | Attempt again to delete a conversation between user1 and user2. This time expect an error as a conversation no longer exists. | | |
| 6 | Attempt to send a message from user1 to a non existent user3. An error should be thrown. | | |
| 7 | Attempt to delete a conversation between Us | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Bradley Gavan | |
| **Test Type** | | Module Testing - White box | |
| **Test Case Number** | | 10 | |
| **Test Case Name** | | Lost and Found Posting | |
| **Test Case Description** | | **Attempt to create a post for the Lost and Found group on the MUNSSN. Ensure that the post is added to the Lost and Found group document and verify that the fields of the document are equal to the given inputs.** | |
| **Item(s) to be tested** | | | |
| 1 | LostAndFound.js - makePost function | | |
| 2 | setFound()  setLost() | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String title  String description  String location  String picURL  Bool found | | | A post added to the Lost and Found timeline. The values in the document should match those of the given inputs. |
| **Procedural Steps** | | | |
| 1 | Create a post with some given inputs, mark post as lost.. | | |
| 2 | Retrieve the post from the database. | | |
| 3 | Verify that the given inputs match the database values. | | |
| 4 | Mark the post found using the setFound() function. Verify that the post is now found. | | |
| 5 | Mark the post lost using the setLost() function. Verify that the post is now lost. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Joshua Davis | |
| **Test Type** | | Module Testing - White Box | |
| **Test Case Number** | | 11 | |
| **Test Case Name** | | Create Schedule | |
| **Test Case Description** | | **Attempt to input the information and successfully create a schedule. Ensure the schedule has been created and displayed successfully on the page.** | |
| **Item(s) to be tested** | | | |
| 1 | schedule.js | | |
| 2 | inputSchedule(String name, String time, String[] days),  createSchedule() | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String name  String time  String[] days | | | A schedule filled in with the new information that was entered by the user |
| **Procedural Steps** | | | |
| 1 | Input correct information into the input boxes | | |
| 2 | Click create Schedule, verify schedule is being displayed correctly | | |
| 3 | Put incorrect information into input boxes. Verify incorrect information is caught and handled | | |
| 4 | Fill out entire schedule, verify the schedule is outputted correctly | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Joshua Davis | |
| **Test Type** | | Module Testing - White Box | |
| **Test Case Number** | | 12 | |
| **Test Case Name** | | Permissions/Content Visibility | |
| **Test Case Description** | | **Attempt to change the permission option on content posted to user timeline. Ensure that content is being displayed or correctly to specific user-base that was selected.** | |
| **Item(s) to be tested** | | | |
| 1 | permissions.js | | |
| 2 | checkPerm(String choice) | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String choice | | | Content displayed to “choice”  If this fails, an error message should be displayed. |
| **Procedural Steps** | | | |
| 1 | Select each choice, use different test accounts to verify correct output | | |
| 2 | Select list and input no people, should expect warning message to add at least 1 person to list. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Joshua Davis | |
| **Test Type** | | Module Testing -Whitebox | |
| **Test Case Number** | | 13 | |
| **Test Case Name** | | Polling System | |
| **Test Case Description** | | **Attempt to create poll with inputs and have the ability to input a vote and see the results change. Make sure voting works correctly and updates in the database.** | |
| **Item(s) to be tested** | | | |
| 1 | poll.js | | |
| 2 | initializePoll(String title, String question, String[] options)  createPoll()  updateVotes(String output) | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String title  String question  String[] options  String output | | | Expect poll to be created with all inputted options. |
| **Procedural Steps** | | | |
| 1 | Input information to create a poll. | | |
| 2 | Check to make sure poll has been created correctly | | |
| 3 | Test that a option can be inputted and the voting results will be updated correctly | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Matthew Randell | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | |  | |
| **Test Case Name** | | Profile Information | |
| **Test Case Description** | | **Attempt to edit the logged in users profile information.this information includes their name,major and minor. Also test that they can add/edit/delete a schedule or resume object .** | |
| **Item(s) to be tested** | | | |
| 1 | profile.js | | |
| 2 | editName()  editMajor()  editMinor()  add Schedule()  editSchedule()  deleteSchedule()  addResume()  uploadResume()  editResume()  deleteResume() | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String fName,mName,lName  String major  String minor  Schedule() ”created from createSchedule()”  Resume() ” created from createResume()”  file resume.txt | | | The information given in input should be uploaded to become the most current information in the user’s profile |
| **Procedural Steps** | | | |
| 1 | Call editName() with arguments fName,mName,lName | | |
| 2 | Update user's name variable to the given arguments | | |
| 3 | call editMajor() with argument major | | |
| 4 | update users major to the given string | | |
| 5 | call editMajor() with argument major | | |
| 6 | update users minor to the given string | | |
| 7 | Verify that the given input match the database values. | | |
| 8 | call addSchedule() on the given schedule object | | |
| 9 | Verify that the schedule object has been attached to the users profile | | |
| 10 | call editSchedule() | | |
| 11 | change some values in the schedule editor and save them | | |
| 12 | verify the values were changed and saved | | |
| 13 | call deleteSchedule() | | |
| 14 | verify that the schedule object has been removed | | |
| 15 | call addResume() on the given resume object | | |
| 16 | Verify that the resume object has been attached to the users profile | | |
| 17 | call editResume() | | |
| 18 | change some values in the resume editor and save them | | |
| 19 | verify the values were changed and saved | | |
| 20 | call deleteResume() | | |
| 21 | verify that the resumeobject has been removed | | |
| 22 | call uploadResume() with the resume text file as an argument | | |
| 23 | verify that the text file was properly converted into a resume object and uploaded to the users profile | | |
| 24 | call deleteResume() | | |
| 25 | verify that the resumeobject has been removed | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Matthew Randell | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | |  | |
| **Test Case Name** | | Suggested Friends panel | |
| **Test Case Description** | | **Attempt to get a list of 10 suggested friends from the database to present to the user based on match criteria** | |
| **Item(s) to be tested** | | | |
| 1 | displayList | | |
| 2 | readRandom()  checkMajor()  checkMinor()  checkMutual()  checkClasses()  createSuggestions() | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| Sting user | | | A list of 10 non-friend status users that have similarities to the user that called the search |
| **Procedural Steps** | | | |
| 1 | Call createSuggestions() which calls the readRandom() checkMajor(),checkMinor,checkMutual() and checkClasses() FUNCTIONS | | |
| 2 | Call ReadRandom() | | |
| 3 | Verify that 10 users were selected that are not friends with the current user and that they are all unique | | |
| 4 | Call checkMajor() on the user and the current random user | | |
| 5 | Verify that if the majors are the same the value of suggestion level for the random user is updated. | | |
| 6 | Call checkMinor() on the user and the current random user | | |
| 7 | Verify that if the minors are the same the value of suggestion level for the random user is updated. | | |
| 8 | Call checkMutual on the user and the current random user | | |
| 9 | Verify that if the user has a set number of the mutual friends with the random user that the suggestion level of the random user is updated | | |
| 10 | Call checkClasses on the user and the current random user | | |
| 11 | Verify that if the user has a set number of the classes with the random user that the suggestion level of the random user is updated | | |
| 12 | Repeat steps 4 to 11 for each of the 10 random users and store the suggestion levels in a list | | |
| 13 | Sort the suggestion list by suggestion level | | |
| 14 | Verify that the list has been sorted properly | | |
| 15 | Call displayList() with the sorted suggestion list as an argument | | |
| 16 | Verify that the list of suggested friends is now displayed to the user in the proper order | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Floris Bouman | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | |  | |
| **Test Case Name** | | “Accordion” (left side profile) | |
| **Test Case Description** | | **Create a users profile from the GUI, and check that all its parameters can be modified by its owner, this includes:** | |
| **Item(s) to be tested** | | | |
| 1 | app (app.js) | | |
| 2 | user module (userSchema.js) | | |
| 3 | user input (index.ejs) | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| file avatar    String aboutme    String schedule    String resume | | | 'test users “about me” successfully updated'  where aboutme is resume, schedule or avatar, depending on which function |
| **Procedural Steps** | | | |
| 1 | Intentionally input correct values, or a valid image file, then click the “upload”, or “set about me”/schedule/resume | | |
| 2 | Intentionally input an invalid image file. The user should be informed to try again with a valid file (also outlining file requirements: <16mb image size. | | |
| 3 | Intentionally input an invalid form of text into the fields, the user should be informed to try again with valid text (outlining requirements like no illegal chars | | |
| 4 | Intentionally input nothing into the text fields, and press “set about me”/resume/schedule. The user should be notified that this deletes their current content for that parameter, and given an option to proceed or cancel. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Floris | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | |  | |
| **Test Case Name** | | “Commenting” | |
| **Test Case Description** | | **Create a comment, edit that comment, and reply to that comment.** | |
| **Item(s) to be tested** | | | |
| 1 | app (app.js) | | |
| 2 | comment module (commentSchema.js) | | |
| 3 | user inputs (index.ejs) | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String comment (x3) | | | ‘comment succesfully left’  ‘reply sucessfully left’  ‘comment successfully edited’ |
| **Procedural Steps** | | | |
| 1 | Intentionally input a correct string as a comment/reply | | |
| 2 | Intentionally input an empty string as a comment/reply. The user should be notified to please enter text in order to leave a comment or reply. | | |
| 3 | Intentionally input a correct string as an edit | | |
| 4 | Intentionally input an empty string as an edit. The user should be notified this will delete their comment and be given an option to proceed or cancel | | |
| 5 | Intentionally input a comment or reply with invalid characters. The user should be notified they must enter valid characters to comment, and to try again. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Floris Bouman | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | |  | |
| **Test Case Name** | | Searching with nav bar | |
| **Test Case Description** | | **search for groups or users that match the given input in the database** | |
| **Item(s) to be tested** | | | |
| 1 | navbar.js | | |
| 2 | results.ejs | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| String searchCriteria | | | a list of groups and users that match the  given search criteria |
| **Procedural Steps** | | | |
| 1 | Input the searchCriteria string into the search bar on the nav menu | | |
| 2 | query the user and group db’s for the user's input | | |
| 3 | return results of search | | |
| 4 | order results by relevance | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tested By:** | | Floris Bouman | |
| **Test Type** | | Module Testing | |
| **Test Case Number** | |  | |
| **Test Case Name** | | Returning home | |
| **Test Case Description** | | **when the home button is pressed the user should be returned to their profile page** | |
| **Item(s) to be tested** | | | |
| 1 | navbar.js | | |
| **Specifications** | | | |
| **Input** | | | **Expected**  **Output/Result** |
| current user’s profile | | | user is back at the homepage for their own profile |
| **Procedural Steps** | | | |
| 1 | clicking on the home button searches the database for the current user's profile id | | |
| 2 | display the user’s profile | | |
| 3 | verify that the user is on their own profile with regular access to their personal options for it. i.e not viewing it as a guest | | |

## Module Testing

Module Testing is done at the code level to check for language-specific errors such as syntax errors, logic errors, and to ensure that particular functions work as expected.

### White Box Testing

In white box testing, the UI is bypassed. We will submit inputs directly into the code and outputs are tested directly at the code level and the results are compared against specifications.

## Integration Testing

### Incremental Testing

There are two primary modules that will need to be integrated: the Graphic User Interface module and the Tree Repository module (back-end). The two components, once integrated, will form the complete Binary Search Tree Application. The following describes these modules as well as the steps that will need to be taken to achieve complete integration. We will be employing an incremental testing strategy to complete the integration.

**Graphic User Interface (GUI)**

The User Interface provides users with hyperlinks and graphical fields to manipulate data. This will be tested to ensure all links, fields, and page structures look and behave as expected. Any structural errors, missing links, or non-functioning input-fields can be corrected before implementing the entire system.

**Other Modules**

All other modules rely on the GUI to display their outputs. Tests on each modules will ensure that outputs are placed into their proper web page sections and that the content does not cause the web page's structure to break.

## System Testing

System testing attempts to find any errors that are only revealed once the entire system is integrated. System testing is concerned with areas such as performance, security, and input validation.

### Function Testing

The database has a number of functions which must work correctly for large sections of the system to work. These should be tested separately from the other modules to ensure that integration testing can proceed without issue.

|  |  |
| --- | --- |
| **Function** | **Expected Behavior** |
| Load | Pull data from the database and display on a webpage |
| Register | Add a user to the database |
| Insert | Insert data to the database |
| Delete | Remove data from database so it can no longer be accessed |
| Search | Locate data based on certain criteria and display the data on a webpage |
| Erase Account | Remove a user so that all data inserted by that user is also removed. |

### Performance testing

Performance will be tested using black-box testing method. We will check the following areas:

* Inserting a large number of user data into the database and observing how the application performs.
* Trying to store new data and check if it overwrites the existing ones.

# Exit Criteria

This section describes the criteria by which testing will be completed.

## Module Testing

Module Testing is done at the code level to check for language-specific errors such as syntax errors, logic errors, and to ensure that particular functions work as expected.

* Each module must be coded. Errors should be fixed. Each module should meet the main functional requirements.
* The GUI should be structurally strong so that it does not break when content is inserted.
* Account security must be ensured. Login attempts must work as expected. No user should be able to break the system to gain access to another user's account.
* Performance should be smooth and consistent for a large number of database values.

## Integration Testing

The HTML user interface and the other modules will be integrated to ensure that content is displayed correctly. There are several entry and exit criteria to allow the testing process to progress smoothly.

### Integration Test Entry Criteria

The Integration Test Entry Criteria will rely on all modules being operational. The criteria are as follows:

* All database functions must be coded and functioning.
* The Web User Interface must be coded.
* Black Box Testing of each module should be completed.
* White Box Testing should have begun.

### Integration Test Exit Criteria

The Integration Test Exit Criteria will rely on all modules being operational. The criteria are as follows:

* All bugs that are exposed are corrected.
* All modules interact correctly with the Graphical User Interface Module. Any bugs are corrected.
* All Modules are ready for System Testing.
* Black Box Testing is completed.
* White Box Testing should be completed.

## System Test

The System Test criteria apply for assessing overall quality of the entire system. All modules are integrated and tested together. A system test focuses on performance, reliability, and security.

### System Test Entry Criteria

Once all modules have been independently developed and tested, the system test may begin.

* The Graphical User Interface, database, and all other modules are completed.
* All code must be tested and signed off.
* All Black Box testing must be complete and exposed bugs must be corrected.
* All White Box testing must be complete and exposed bugs must be corrected.
* Integration Testing must be complete and exposed bugs must be corrected

### System Exit Criteria

To consider the project complete, all modules of the project should be properly integrated and tested according to the requirements of the project.

* All modules and database functions must have tested successfully and accurately.
* The system should remain consistently high in performance, security and reliability.
* The Graphical User Interface behaves correctly and is simple to navigate.
* All input fields on the Graphical User Interface are working correctly.
* Any remaining issues must be deemed acceptable to release to the general user.

## Live Release

Once all bugs have been fixed and a minimum of exit criteria have been achieved. The system may now be launched to a web browser via a server for full use.

# Bug Tracking/ Bug Process

# 

During testing, the team members normally encounter behavior that does not meet specifications. Upon finding an error, that error should be documented on GitHub for other team members to assess and assist in correcting it.

**Reporting Expectations:**

* Document which branch and version the bug was found in
* Determine if bug has already been reported
* Indicate the steps needed to reproduce the bug
* Report exact results of the bug
* Describe what results should have occurred in place of the bug
* Describe how the bug affects project performance
* Describe the seriousness of the bug. Is it minor, or does it cause a module to stop functioning?

# Roles and Responsibilities

**Module Programmer & Tester (Registration, Login/Logout, Timeline) – Curtis White**

* Ensure users can properly register and access account content.
* Ensure exit criteria are achieved for the database and user account based modules.
* Regularly review testing progress.
* Raise and manage issues/risks relating to modules.
* Review and sign off test approach for respective modules.

**Module Programmer(Schedule, Create Poll, Permissions) – Joshua Davis**

Ensure users can create and view the schedule they created.

Ensure users can input their class information correctly and easily.

Ensure all users are capable of setting permissions correctly and that these permissions are followed for other users viewing the content.

Ensure all polls create correctly and that they update when being voted on.

Ensure the modules are cohesive with the overall design of the software.

Maintain the usability of these modules through software changes.

# Test Schedule

The section contains the overall project schedule. It discusses the project milestones and testing goals.

|  |  |  |  |
| --- | --- | --- | --- |
| **Program Milestones** | **End Date** | **Notes** | **QA Deliverables/Roles** |
| Milestone One | 02/06/2017 | At this Milestone, requirements and specifications are planned. Modules are outlined and functions are devised. | Requirements Document (Software Requirements Specification) |
| Milestone Two | 02/26/2017 | At this stage, the system architecture is advanced based on the requirements reported in the previous milestone. Diagrams and schema are devised to visualize the project's design. | Architecture Document  Modules should be started and some basic elements established. |
| Milestone Three | 03/12/2017 | In this milestone, modules are outlined further and the testing procedures for ensuring proper functionality are described. | Modules Document  Modules should have some functionality. System Testing should be close to meeting entry criteria. |
| System Demonstration | 04/06/2017 | The system should now be integrated and meet a minimum of functionality according to requirements.  The demonstration will showcase the release version of the project. | All previous documents.  All modules should be completed, integrated and all testing passed on the modules and the system. |

# Deliverables

* Software Requirements Specification
* Architecture Document
  + - Use Cases
* Modules Documentation
* Testing Procedures

Module Test Plan

System Test Plan

Test Analysis

* Program Source Code

Bug Reports

# References

Pressman, Roger S. Software Engineering - A Practitioner's Approach. Fifth edition. The McGraw-Hill companies, Inc.

Kaner, C., Falk, J., Nguyen, H.-Q. Testing Computer Software. Wiley Computer Publishing, 1999.