Research Mate

Software Test Plan

CSCI-P465/565 (Software Engineering I)

Project Team

<Gulshan Madhwani>

<Jayendra Khandare>

<Shubham Basu>

<Xinquan Wu>

---

1. Overview

1.1 Test Objectives

In Sprint one, we develop login module of ResearchMate, and we host it on silo.soic.indiana.edu. This test is to test the functionality of the login module, to check it's working or not.

Unit Test of ResearchMate[Sprint 1]:

1. Create an account.
2. Verify the email address.
3. Login the account.
4. Log off from the account.

In Sprint two, we refined login and registration module. We have developed features like user profile and have extended database so that it now includes papers published and their information. We are permanently hosting them on silo.soic.indiana.edu:54545 unless testing and debugging is being performed. This test is to test the functionalities of the extended database and verification of the user profile page.

Unit Test of ResearchMate[Sprint 2]:

1. Session string management.
2. User profile setup.
3. Extended database verification.

In Sprint three, we have developed functionalities for search and filter option which will search a specific keyword[string] through all collections, and databases and provide the user with results matching or having that keyword in the content.

Apart from this, we performed major UI changes and cleared up the backlogs from previous Sprint regarding some bugs.

Unit Test of ResearchMate[Sprint 3]:

1. Search in user information collections
2. Search in groups collection
3. Search in skills collection

In Sprint-4, we have developed functionalities suggested by Prof. Adeel. These functionalities include pages for publications,groups and general discussions. Also, we have created functions for presenting the ratings of various publications and finding most liked publications.

Apart from that, we have also made the groups to be public or private and allow access to only public groups and discussion to the user who has not joined that group.

Unit Test for ResearchMate[Sprint 4]:

1. Post a discussion question.
2. Post a discussion reply.
3. Join a group, create a group and view the discussions in a group or public space.

In Sprint-5, we have developed chat functionalities and others like leaving a group and deleting a published paper. This chat functionality covers a lot of functions like storing messages in a database and getting them again, to be displayed when user reconnects the chat.

Apart from that, we have also added validations for various kinds of input to some major functions and tested thoroughly with some real data.

Unit Test for ResearchMate[Sprint 4]:

1. Connect with an user who is in your connections.
2. Leave a group.
3. Delete a paper published by user.

1.2 Test Environment

We are using 3 operating systems to do the testing which are as following:

1. Windows 10 64bit;
2. Ubuntu 64bit;
3. OS-X 64bit;

All use Chrome browser to connect to the web application we have developed. Furthermore, as a requirement this project must be hosted on silo server provided by SICE, hence it is hosted with Jayendra's account.

1.3 Test Personnel

1. Gulshan Madhwani using Windows 10 64bit with Chrome;
2. Jayendra Khandare using Ubuntu 64bit with Chrome;
3. Shubham Basu using Windows 10 64bit with Chrome;
4. Xinquan Wu using OS-X 64bit with Chrome.

1.4 Acceptance Criteria

Sprint 1:

Successfully creating an account, receiving an email to verify the email address, and logging in with the account. Only getting through all these steps that will be consider acceptable. All group members carry out the tests separately.

A complete module of the project will be satisfying if the data provided by the user is accepted and stored at in MongoDB database hosted at the server(silo.soic.indiana.edu).

Sprint 2:

Adding data about the user successfully, verifying the account, maintaining and managing session string using cookies, user being able to add himself in a group, user being able to add information about the papers he has published.

A complete module of the project will be satisfying if the user is able to see and change his personal information and he should be also able to upload information about his research papers and join groups dedicated to various topics.

Sprint 3:

The user will provide a keyword. This keyword has to be searched in user information collections, group information collection, and skill information collection. The relevant information should be displayed on a page. The user should also be able to filter through these results based on some criterion.

This module will be acceptable when the user gets results for his keyword which are displayed on a web page and he is able to sort and filter based on some criterion.

Sprint 4:

The user will be able to join a group, once his request is accepted by the admin / creator of that group. Only a admin will be able to accept requests. Then the accepted user will be able to view questions and answers posted in that group. Anybody will be able to create a group. An user can see which are the most rated publications in the database and he can also provide his own rating.

The work of this sprint will be acceptable when only the admin of any group is able to accept or reject the request of any user who has requested. Also, the questions and answers of someone from a group are shown on the UI.

Sprint 5:

The user will be able to chat in real time with his connections. When he returns to a chat, he should see his previous conversation. He should be able to leave a group which he is a part of. He should be able to delete a paper which he has posted on the website. Also, admin should be able to block some user and re-activate him.

The work of this sprint will be acceptable if all the functionalities described above work as stated and no errors are shown.

1.5 Noted Omissions

-

2. Test Cases

Sprint 1:
Number: 001
Name: Checking support provided by silo server
Description: This test includes checking whether silo server system supports the tools required for MEAN stack development.
Initial Conditions: No prior test case is needed for this test.
Input Data: Student username and passphrase for accessing any system of IU.
Specifications:

1. **Get access to a silo account.**
2. **Check whether it has MongoDB support.**
3. **Check whether it has Node.js support.**
4. **Check whether it has NPM package manager support.**

Procedure: We used Jayendra Khandare's account to access silo/burrow and do the necessary testing. The testing includes running some basic bash commands like

- **mongod –version**
- **npm –version**
- **node –version**

Number: 002
Name: Hosting the server code on silo
Description: For this test, we had to develop some basic server code using node and run on silo. After that, we had to manage some connectivity issues since 'mongod', which is server for mongodb doesn't give enough rights to students.
Initial Conditions: The access to silo is required for this test. The access is given by default to all students of SICE.
Input Data: server application to be hosted on silo
Specifications:

1. Start an instance of mongod through student account.
2. Run the server code developed by the team.

Procedure: Whenever you run a server on silo, you must reserve a port for your website. This procedure is done through the server code. For this project, we chose 23456 as the port. Hence, the address becomes http://silo.soic.indiana.edu:54545.

As the server code is not running all the time, the availability of this address is ambiguous.

Number: 003
Name: Register & Login test
Description: Create an account, receive an email to verify the email address and account, login with the account created, log out with the account.
Initial Conditions: Satisfying test number 002.
Input Data: First name, Last name, User name, Email address.
Specifications:

1. Create an account;
2. Receive an email from se.researchmate@gmail.com with a link to verify the email address;
3. Login with the account;
4. Log out with the account.

Procedure: The link cannot be provided as we are hosting the project at silo server and don't have any idea how to perform the tests on this environment. We have contacted IU knowledge base regarding this. The will provide us with a definite answer in the next week.

Sprint 2:

Number: 004
Name: Session String Management and Verification
Description: Whenever a user tries to login, after verifications and validations, a random string is returned to the user and it is stored in cookies. This string is used hereafter for security checks.
Initial Conditions: Satisfying test number 003.
Input Data: session string.
Specifications:

1. When login() function is called, check username and paossword.

2. After verification, assign a randomly generated session string in user collection(/table) and send the same string to front end.

3. Front end will store this string in cookies.

4. In each sub-sequential transaction with the server, this string will be provided and server will decide based on that string whether to allow access.

5. This string will be required for all transactions for everyone, admins will have a different session string which they can change(if required).

Procedure: For sessionString generation, we are using a module called randomString. This string is generated and stored in the database when the user logs in. This string is stored in cookies file, hence the user doesn't have to worry about keeping the track of this string. It is updated each time the user logs in. It is stored in database which helps the server to find the userID which is required for most of the functionalities.

Number: 005
Name: Data packet verification and validation
Description: The data packets which are sent to and fro while communicating with the server from front-end have to be checked thoroughly if they have the data required for certain function. We are using JSON format for the project, hence it has become easy to keep track of the data flow and apply necessary validation parameters.
Initial Conditions: JSON formatted data packet.
Input Data: JSON data packets.
Specifications:

1. Whenever any kind of data is sent towards server, it is checked at front-end before sending whether it is in JSON format.

2. Whenever any kind of data is received by the server, server checks again whether data is in JSON format.

3. If the data is not in JSON format, the server rejects such data packet, because it possibly could be a query injection attempt.

Procedure: We didn't stick to any module or package for this validation, because in MEAN stack environment, it is a usual practice to change the signatures of functions and their parameter values often. Although it is good for security, it requires you to change the code every time to accommodate with the shift. In order to avoid that and make it a robust and steady model, we checked the format for each packet before sending[at front-end] and after receiving [at back-end]. Hence, there are no fixed function for that.

Number: 006
Name: User profile maintenance
Description: Every user has a user profile which he can modify as much as he likes. This profile contains his personal information about his research publications, the groups he is following, comments he has made, people he is following and the people that follow him. This information should not be immutable, hence we created functions to access and change this information.
Initial Conditions: user is verified and his session is validated.
Input Data: For accessing, session string. For changing, session string and JSON data.
Specifications:

1. Once the user logs in, he will get the session string that is allocated to him.
2. Using that session string, he can call functions which will give him his personal information.
3. If he wants to change his information, he just needs to be logged in and press submit buttons wherever he feels like changing.
4. Once the user requests to change the data, he provides session string which is needed to locate his ID and JSON data for the elements he wants to change, this data is updated in the respective fields required by user.

Procedure: At this moment, we have used one function each for accessing and modifying user data. These functions accept session string as validation tokens and locate the userID from users collection(/table), this userID is always required if we want to make any changes. At this stage, we have not included data from all the collections(/tables) in user profile, but that will be done as soon as we populate the database.

Sprint 3:

Number: 007
Name: Search for user information
Description: If the user knows first name or last name or user name of any other person, he should be able to check whether that person has an account in ResearchMate. If so, he should be able to get access to profile page of that person. This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account.
Input Data: searchString[keyword to search]
Specifications:

1. A user/visitor will provide a searchString which relates somehow to some user's name or username.

2. Once that string is added, it will be sent to back-end.

3. The results will be displayed on a page.

4. He can also filter the results based on parameters like first name, last name, etc.

Procedure: The searchString sent from the front-end will be searched though user collections and then user information collection to find matches. If at least one match is found, the results will be sent to front-end by means of a JSON object. This object will be parsed at front-end and the results will be displayed on a new page in readable format. For filtering the results, we are using a angular directives or 'ngfilters'.

Number: 008
Name: Search for any information
Description: If the user knows any skill, he should be able to check whether any person has that skill in ResearchMate. If the user knows any group or description keywords of any group, he should be able to check whether any person has joined that group in ResearchMate. If so, he should be able to get access to profile page of those people. This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account.

Input Data: searchString[keyword to search]
Specifications:

1. A user/visitor will provide a searchString which relates somehow to some group or skill.

2. Once that string is added, it will be sent to back-end.

3. The results will be displayed on a page.

Procedure: The searchString sent from the front-end will be searched though skill and group information collections and then user information of the people who have that skill or who have joined some group relating to that keyword will be sent to front-end. If at least one match is found, the results will be sent to front-end by means of a JSON object. This object will be parsed at front-end and the results will be displayed on a new page in readable format. For filtering the results, we are using a angular directives or 'ngfilters'.

Number: 009
Name: Security measures
Description: If the user send some wild card characters like " / " or " * ", they should not be processed on the collections and databases. Also SQL injections have to be checked before firing the queries on database. This possesses a major security risk. All these kind of entries have to be omitted by server and the user will be not be provided with results for his query. This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account.
Input Data: None
Specifications:

1. Checks are performed at back-end for wild card characters like " / " or " * ".

2. Checks are performed at back-end for SQL queries.

Procedure: The searchString sent from the front-end will be checked for wild card characters and SQL queries before processing them on database for a results. If any such values are found in the searchString, the user won't be provided with any results.

Spint 4:

Number: 010
Name: Post a question
Description: An user can post a question in a group he has joined or on an open discussion page. This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account.
Input Data: postString[question or post in text], groupID or -1 for public discussion.
Specifications:

1. The user should be logged in so that he has his own sessionString.

2. The user has to be a member of some group for him to be able to post a question there.

3. Otherwise, he can post a question at public discussion page.

Procedure: The user will provide the front end with a questionString or postString which will have his question. From front-end, that string and groupID and sessionString of that user is sent to back-end. Specifics of that user are found based on his sessionString. If the groupID is -1, the question will be posted in public discussions, otherwise it will be posted in specific group discusssions.

Number: 011
Name: Post a reply.
Description: An user can post a reply to a question in a group he has joined or on an open discussion post . This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account.
Input Data: replyString[reply to a post], postID.
Specifications:

1. The user will have the postID for which he is posting the comment / answer.

2. As he is logged in, front-end will have his sessionString.

3. The answer / comment will be taken from user at front-end and passed to back-end.

Procedure: The user will provide the front end with a replyString which will have his answer / comment. From front-end, that string and postID and sessionString of that user is sent to back-end. Specifics of that user are found based on his sessionString. Based on the information, that reply will be stored in discussionreplies collection.

Number: 012
Name: Join a group.
Description: The user knows how many groups are there and their information. If he wants to join some private group, he should be able to send a request to the admin of that group.
Initial Conditions: The user should be logged into his own account and knows which group he wants to join.
Input Data: groupID [of the group he wants to join], sessionString
Specifications:

1. The user wants to join a group that he has not joined before.
2. He knows the details of that group.
3. If that group is open / public, he should be able to join it directly.
4. If that group is private, he will send a request to the group admin.

Procedure: If the group is public, the user is added to it right away. Otherwise, his request is stored in “GroupJoinRequest” table. Actually, the userID of the requesting user and date of the request is stored in this table. Later, admin of that specific group can see those requests and decide whether to allow the user or not.

Number: 013
Name: Accept or reject requests to join a group.
Description: Some groups are private and the creator of those groups is the admin for those groups. If someone wants to join one of these groups, he make a request and then admin decides whether to allow them or not.

Initial Conditions: The request is made to join some group.
Input Data:
Specifications:

1. When some admin wants to check how many pending requests are there in his group, he will call function getAllPendingRequests() with the help of front-end functionalities.

2. He can see the name and join request date of users who requested.

3. He can allow them to join or reject their requests.

Procedure: When some admin wants to allow someone entry into his group, or reject it. For allowing, the request is deleted from GroupJoinRequest table and an entry is added in UserGroupMapping table. If allowed, that user will be able to view the contents, posts and replies in that group.

Spint 5:

Number: 014
Name: Delete a paper published by own.
Description: An user can delete a paper he has posted on the website earlier. This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account and has a paper posted.
Input Data: sessionString of the user, and publicationID of the paper he wants to delete.
Specifications:

1. User logs into his account which gives him sessionString which is used for all the authorizations.

2. He can go to his published paper and decide to delete or let it be.

3. If he wants to delete it, remove that from website.

Procedure: The user will provide the front end with publicationID he wants to delete. From front-end, user's sessionString and publicationID is sent to back-end.

At back-end, that publication is deleted and all entries of it in userPublication table will be deleted.

Number: 015
Name: Leave a group which the user is a member of.
Description: An user can leave a group he has joined before on the website earlier. This functionality is supported by this sprint's deliverable.
Initial Conditions: The user should be logged into his own account and has a group joined.
Input Data: sessionString of the user, and groupID of the group he wants to leave.
Specifications:

1. The user has joined some group which he don't want to be a member of.
2. He will see all of his groups and decide which group to leave.
3. There is no request if you want to leave a group in this scenario.

Procedure: The user will provide the front end with groupID of the group he wants to leave. From front-end, user's sessionString and groupID is sent to back-end. At back-end, that userGroup entry is deleted from userGroupInfo table.

Number: 016
Name: Deactivate a user.
Description: An admin can deactivate the account of some user. This functionality is supported by this sprint's deliverable.
Initial Conditions: None.
Input Data: admin credentials and username of the user to be deactivaed.
Specifications:

1. Once the superuser or admin logs into his account, he will be shown a page redirections link which eventually shows him information about all the users that have created an account on the website.
2. He can deactivate a user by clicking on the button for deactivation.

Procedure: After successful login of a superuser or admin, he will have a link which will show him information of all user. If he decides to deactivate some user, the username of that user will be sent to back-end and his verification flag will be set to -1, which will give him message like “User Blocked. Contact admin.” when he tries to login.

Number: 017
Name: Activate a user.
Description: An admin can activate the account of some user. This functionality is supported by this sprint’s deliverable.
Initial Conditions: some deactivated user.
Input Data: admin credentials and username of the user to be activaed.
Specifications:

1. Once the superuser or admin logs into his account, he will be shown a page redirections link which eventually shows him information about all the users that have created an account on the website and their status.

2. He can activate a user by clicking on the button for activation.

Procedure: After successful login of a superuser or admin, he will have a link which will show him information of all user. If he wants to activate some user, the username of that user will be sent to back-end and his verification flag will be set to 1, which will give him access by providing a sessionString, when he tries to login.

Revision History

Revision	Date	Change Description
Sprint 1	10/01/2017	Test plan for sprint 1: login and sign up module
Sprint 2	10/15/2017	Added test objectives, 3 test cases, and acceptance criteria specific for Sprint 2
Sprint 3	10/29/2017	Added test objectives, 3 test cases, and acceptance criteria specific for Sprint 3
Sprint 4	11/12/2017	Added test objectives, 4 test cases, and acceptance criteria specific for Sprint 4
Sprint 5	03/12/2017	Added test objectives, 4 test cases, and acceptance criteria specific for Sprint 5
Summary	03/12/2017	The plan has 17 test cases, acceptance criteria for 5 sprints and unit test details for 5 sprints.

Page Author/Creator: Adeel Bhutta
Last Modified: 8/23/2016