SW Engineering CSC648/848 Summer 2022

GATORSHARE

Milestone 04

Team 01, Section 01

Team Members:

Donna Nguyen, Team Lead | unguyen3@mail.sfsu.edu

Estefanos Kebebew, Back-End Lead

Brianna Soukup, Front-End Lead

Mohamed Toure, Github Master

Brian Nguyen, Database Master

Aleksandr Gusev, Front-End

History:

	Date Submitted	**Date Revised**
Milestone 1	June 21, 2022	July 15, 2022
Milestone 2	July 07, 2022	July 18, 2022
Milestone 3	July 19, 2022	-
Milestone 4	July 27, 2022	

Table of Contents

Product Summary

Name of the product:

GatorShare

Product Summary:

Our application is a one-stop website designated for San Francisco State University students, faculty, alumni, and the general public. Our goal is to connect people, allow viewers to explore different social events SFSU has to offer, find Discord servers related to SFSU clubs and courses, and find Tutoring services. Our website also allows Users to post their work and become a part of a community. GatorShare is unique in that our objective is to combine everything into one. Just by accessing GatorShare alone, a Guest will be able to find resources, find Discord channels, view event flyers, read articles written by SFSU peers, and also find Tutors that fit their needs. Registered Users can interact with other SFSU students, comment on posts, and message other peers. Our product helps both new and returning students locating the tools they require to feel welcomed and a member of the SFSU community. We provide an atmosphere in which members of the SFSU community may be connected and discover all the tools they need to support their academic success and social fulfillment on and off campus.

Product URL:

http://gatorshare.com/

Itemized List: Priority 1

Requirements for Website:

1. Search Bar shall be present throughout all pages.
2. Footer shall be present throughout all pages.
3. Logo shall be present throughout all pages.
   - When the Logo is clicked, it shall redirect the User to the Home page.
4. Navigation Bar shall be accessible for all Users.
5. Navigation Bar shall have a tab that redirects to the Home page.
6. Navigation Bar shall have a tab that redirects to the Login page.
7. Navigation Bar shall have a tab that redirects to the Sign Up page.
8. Navigation Bar shall have a tab that redirects to the Posts page.
9. Navigation Bar shall have a tab that redirects to the Create a Post page.
   - Create a Post page shall be visible only for Registered Users.
10. Navigation Bar shall have a tab that redirects to the Messages page.
    - Messages page shall be visible only for Registered Users.
11. Navigation Bar shall have a tab that redirects to the Profile Page.
    - Profile page shall be visible only for Registered Users.
12. Home page shall have a “Most Recent Posts” section.
13. Home page shall have a “Most Popular Posts” section.
14. Home page shall have six tabs, organized by content:
    - “Articles & Essays”, “Art & Film”. “Clubs”, “Discords”, “Tutoring” and “Other”.
15. Footer shall have an About Us link.
16. Footer shall have the Terms & Conditions link.

17. Footer shall have the address of San Francisco State University.
18. Posts shall be public for all Users.
19. Comments shall be public for all Users.
20. Images shall be accessible for all Users.

Requirements for Unregistered User:

1. Unregistered Users shall be able to view all Posts.
2. Unregistered Users shall be able to view all Comments.
3. Unregistered Users shall be able to view all Images.
4. Unregistered Users shall be able to utilize the Search bar.

Requirement for Registered Users:

21. Registered Users shall be able to add a Post onto the platform.
22. Registered Users shall be able to add Comments on another Registered User's Posts.
23. Registered Users shall have a Profile.
24. Registered Users shall be able to send messages.
25. Registered Users shall be able to receive messages.
26. Registered Users shall be able to like a Post.
27. Registered Users shall be able to delete their Post.

Requirements for Posts:

28. Posts shall have an image submission field.
29. Images shall only be accepted in jpg, png, or jpeg format.
30. Posts shall display the Registered User who created the post.
31. Posts shall include a comment box.

32. Posts shall include a “Leave a Comment” submission button.
33. Posts shall display the time of the submission from present time.
34. Posts shall include a like button.
35. Posts shall display the number of likes it has received.

Requirements for Comments:

36. Comments shall display the username of the Registered User who created the comment.
37. Comments shall display the time of the comment submission from present time.
38. Comments shall be text only.

Requirements for Admins:

39. Admins shall be a Registered User.
40. Admins shall be able to Post.
41. Admins shall be able to Comment.
42. Admins shall be able to send messages.
43. Admins shall be able to receive messages.
44. Admins shall be able to like a Post.
45. Admins Users shall be able to receive notifications from Registered Users.

Requirements for Professors:

46. Professors shall be a Registered User.
47. Professors shall be able to receive notifications from other Registered Users.
48. Professors shall have a Profile.
49. Professors shall be able to Comment.
50. Professors shall be able to send messages.

51. Professors shall be able to receive messages.
52. Professors shall be able to like a Comment.
53. Professors shall be able to undo the like on a Comment.

Requirements for Tutors:

54. Tutors shall be a Registered User.
55. Tutors shall be able to receive notifications from other Registered Users.
56. Tutors shall have a Profile.
57. Tutors shall be able to Post.
58. Tutors shall be able to delete their Post.
59. Tutors shall be able to Comment.
60. Tutors shall be able to send messages.
61. Tutors shall be able to receive messages.
62. Tutors shall be able to like a Post.

Usability Test Plan

Purpose -

This test is intended to identify any problems Users may experience while attempting to use our posting feature. This feature includes creating, viewing, searching for, commenting on, and deleting a post.

Problem Statement and Objective -

We have chosen to test these post-related functionalities to ensure that a User is able to navigate around our application effortlessly and efficiently. As we test for creating a post, we want to ensure that a User's post submits correctly and accurately. We want to test to confirm their post is saved into our database which in turn allows their post to be shown on our public feed. Within our public feed, we want to test that posts can be accessed by all Users within our application. When a User searches for a post, we want to guarantee the correct subject is being filtered. Upon clicking on a post, we will be testing to see if each post has a comment box and comments may be posted. Lastly, we want to test if a User can delete their post, and if this deleted post erases from our data system.

Method -

Our method to test our functionalities is to have a new User access our web application. Our User must navigate around our homepage where they can utilize our search bar, click on different tabs, and click on different posts. Our User must also create an account with us in order to test how to create, delete, and comment on a post.

Task list -

We'd want to start by testing our search bar to see if a User can search and find a post. We'd also like to check if a User can click on the post they've searched for. If these two tests are successful, we can confirm that any User can publicly access all postings on our platform. Next, we want to test if a User can create a post, post comments, and delete a post. Because these features are only available to Registered Users, we must create an account in order to test them. If a User can successfully create a post, post a comment, and delete a post, it confirms that all data is being stored in our database, the User can use our platform, and data can be deleted from our database.

Description:

Test Environment/System Set Up -

In order to test our website, our server must first be set up and running. Our database must also be connected to the server in order to store a User's data. When a User first accesses our application, they are able to view and search for a post. These two functionalities are open to both an Unregistered and Registered user. When a User sign's up with our application, their registration data is saved into our database. Upon logging in, a User is now free to create a post, comment, and delete an existing post.

User Profile -

We are testing our application for both an Unregistered User (A User who has not signed up or logged in) and a Registered User (A User who has created an account and has logged in).

Starting Point -

Upon entering our application, our search bar and feed is open to the general public. The User can click on any "Recent Post", "Popular Post", search for a subject in our search bar, and also click any of the category tabs to navigate to different Posts. These features are not limited to any

User. For other components- such as commenting on a post, creating a post, and deleting a post, a User must be registered and logged in. Once logged in, in order to create a post, a User must navigate to the navigation bar and click “+Create a Post”. To comment, a User may click on any submitted Post whether, its by accessing “Recent Posts” or “Popular Posts” from the homepage, opening a category page such as “Discord”, “Tutoring”, “Art & Film”, etc. or searching for a post using our search bar. To delete a post, a user must navigate to their Profile using the navigation bar. From there, a User can delete any of their created posts.

Intended Users -

For students of SFSU to connect with other students, faculty, tutors, alumni, and the general public. Our application is designed to assist SFSU students in communicating with others and finding conveniently available resources that meet their specific needs. They may use our website to acquire tutoring services, locate a discord server to meet new people, join clubs, provide feedback for arts & films, or read articles about SFSU.

URL, What Is Being Measured -

http://gatorshare.com/

Report Content/Evaluation Measures:

Usability Test Table - Measuring Effectiveness -

# of Pages	Test/Use Case	% Completed	Errors	Comments	% of Time to Complete the Task (in seconds)
3	Create a Post	50	Not able to post	You can start the post creation process but nothing happens when you click “Post”	–
	View a Post	100	No error detected	The user can view a post completely with no hassle	06.52
2	Search a Post	90	No error detected	It shows the posts containing the searched words. Though, no recommendation is made yet if it does not find anything.	01.50
3-4	Comment on a Post	100	No error detected	The user can comment on any post after logging in	33.11
3	Delete a Post	0	Not able to test yet	This functionality does not work yet	0

Task Description -

Task	**Description**
Create a Post	From the homepage, the User must first log in. If they are not registered yet, they must create an account and then log in. Once logged in, the User can click "Create Post"

	+ Create Post from the navbar. Once redirected to the Create a Post form, the User must fill out the various fields for the post and, if wanted, upload any desired images or media. When the User is finished, they must click "Post" POST . Once posted, the User is redirected to "All Posts" where they can see their post amongst all other posts.
View a Post	From the homepage, the User can either click on "Posts" Posts in the navbar to view the "All Posts" page or if the User is interested in a certain category of post, they can click on one of the six categories located under the Homepage's hook.
Search a Post	From the homepage, the User can click on the "search bar" and type in what they are looking for. The User can also narrow their search by clicking the "Category" Category dropdown menu next to the bar and selecting a category. Once the User submits their search , the results will be displayed.
Comment on a Post	While viewing a post, a User can leave a comment for that post by entering their comment in the text box below the post's image and description. To post the comment, the User must click "Leave a comment" LEAVE A COMMENT . Once clicked, the User can then see their comment, along with any others, in the provided comment box.
Delete a Post	From the homepage, the User must first log in. Once logged in, the User can click the "Account" Account dropdown from the navbar. In the dropdown, the User must click "Profile" in order to view all their created posts. Once redirected to their Profile, the User can click "Delete" on any of their posts they wish to delete.

User Satisfaction Table

	Strongly Disagree	**Disagree**	**Neutral**	**Agree**	**Strongly Agree**
Create a Post					
Creating a post is fast and requires little effort			X		
The Create a Post form is organized and easy to understand		X			
My post was successfully created with no errors	X				
View a Post					
It is easy to view a post					X
The view a post page contains a title, description, image and comment box					X
I can view any post on the site					X
Search a Post					
The search bar is visible					X
The search function is easy to use					X

I received search results related to what I searched			X		
Comment on a Post					
The comment box is easy to find					X
It is quick and easy to post a comment	X				
I can see all comments posted on a post	X				
Delete a Post (Not implemented yet)					
I have the ability to delete any of my created posts	X				
It is quick and easy to delete a post	X				
I am provided with a popup to confirm deletion	X				

QA Test Plan

Test Objectives:

We would like to test the following components: Comments must have a text limit of 250 characters, passwords shall be encrypted in the database where it's stored, passwords must have very specific criteria when a User signs up, all data shall be stored in the team's SQL database, and a Registered User's session will end once they click log out. We have chosen these components because we want to make sure comments have a text limit to prevent long and unnecessary potential spam. We also want to make sure passwords are encrypted to guarantee a User's password is safe and secure. We require passwords to meet certain criteria to make sure their passwords are not easily guessed. This will prevent hacking measures. Lastly, we want to test if a User's data is stored into our team's database to make sure all posts and comments are properly being collected and saved.

HW and SW setup (including URL):

http://gatorshare.com/

- **Setup of HW:**

In order to have our application accessible, the live version of our website must be deployed to our Cloud services, then a User can access our website from any device of the User's choice. In order to test our website, a User must have a computer, smartphone, or any device that has connection to the internet.

- **Setup of SW:**

We have chosen to deploy our website using Amazon Web Services (AWS) where we have connected our MySQL database to the cloud. A User must have a working version of any web browser in order to access and test our live application. We have tested our application on both a Mac and Windows device. We have also tested our application on the following web browsers: Google Chrome, Mozilla Firefox, and Safari.

Test Environment, Human Resources Needs, Expected Time to Complete:

In order to test our objectives, we need to access our database. Some of the physical resources we require include having our Database Master guarantee that our database is operational and that all passwords are encrypted for privacy and security.

Risks, Contingencies:

One danger we may face is if our database fails. A User's data cannot be stored if our database fails for any reason, such as when they try to sign up or create a post. Another danger we may face is a large number of spam and/or 'bots' posting on our application. We may need to evaluate how to approach such an issue, such as screening what is spam or how to delete a large amount of spam in one session.

Actual Test Sequence, Test Input and Test Expected Results:

In order for our tests to be successful, our Tester must follow the instructions clearly. In order to create a password that fulfills our requirements, our Tester must have a password that is 8-characters long, must have one uppercase letter and one lower case letter, and must contain a number. To make sure our Tester’s password is encrypted, we have hashed their passwords in order to ensure data integrity.

Feature to Be Tested:

We would like to test the following features: Creating a password that fits our requirements, verifying that passwords are encrypted, and posting a comment with a text limit.

QA Test Plan:

Test Plan #1: Entering a Valid Password

- **Input:** Enter "Abcd123!" as a password when creating an account
- **Output:** Password requirements should turn green. This will indicate that the password created is considered valid.

Test Plan #2: Saving an Encrypted Password

- **Input:** Enter any passwords that fulfills the password requirements. (UserTest1234!)
- **Output:** Password is considered valid and the user's information is now stored in our database system. The User's password is encrypted with a series of letters and numbers.

Test Plan #3: Commenting more than 250 Characters

- **Input:** Type or copy and paste any paragraph that is 250 characters or more.
- **Output:** Comment will still submit, regardless of length.

QA Test Plan Table:

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results (Pass/Fail)
1	Create a password that fulfills requirements	Type "Abcd123!"	Abcd123!	Abcd123! All password requirements should turn green, thus confirming that the password is valid	Pass
2	Password is	Type any	UserTest12	Password is in the	Pass

	encrypted in database	password (that fulfills requirements)	34!	database with a series of scrambled letters and numbers	
3	Posting a comment more than 250 characters	Type or copy and paste any paragraph	Text that has 250 characters or more	Comment will still submit, regardless of length. Our goal is to have a pop up with a text limit error, or a limit that stops the user from continuing to type.	Fail

Code Review

Coding Style:

Our primary goal is to have a code that is clear, organized, well-documented, and clutter-free. PascalCase is used for the names of React UI components (e.g. UserProfile.js). For any other assistance files, CamelCase is utilized (e.g. userAction.js). CSS file names are written in kebab-case (e.g. user-action-button.js). The React naming conventions must be followed for components containing React classes (e.g. className instead of class, marginBottom instead of margin-bottom). Make multiple tiny files rather than one big one. All CSS files are in a single, shared folder (like "./CSS"). Avoided !important and inline css as much as possible. We utilized a prettier plugin to make the code easier to understand and review. Files are named sensibly based on the function they carry out. Organized imports in order by React import, Library imports (Alphabetical order), Absolute imports from the project (Alphabetical order), Relative imports (Alphabetical order), Import * as.

Code Review Feedback from members of Team 1 and Team 3:

Team 1 Feedback:

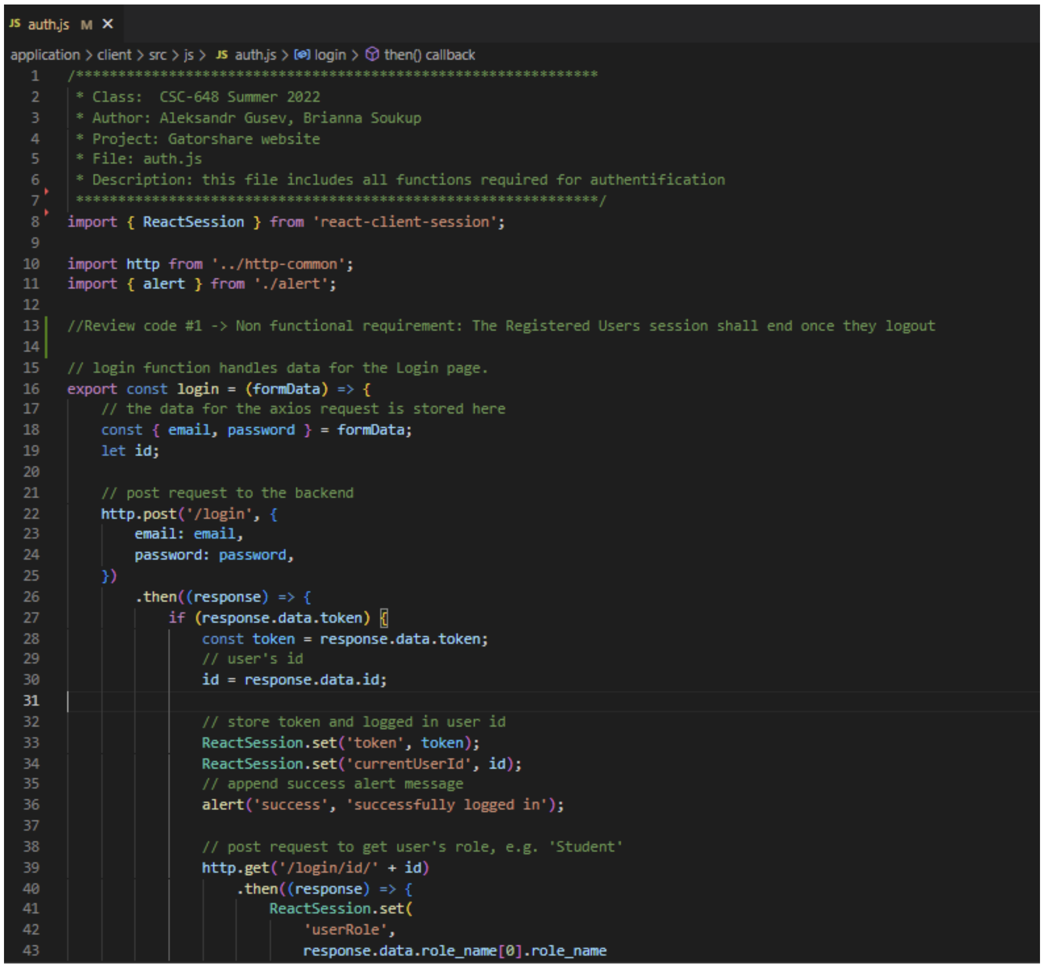
- Short description about each part of the code makes it very direct and straight-forward.
- The white space keeps each part very organized.
- export const login and export const logout are not inline. One is tabbed and one isn't, consider aligning these two.
- Good use of naming conventions.
- Code is very organized and easy to read.

- Handling search results is easy to understand.
- There are alerts when errors occur which is good to be able to keep track
- Good indentation which made it easy to follow along with the code
- Remove the //Todo once the section is done
- The readability for both files are proficient and clear

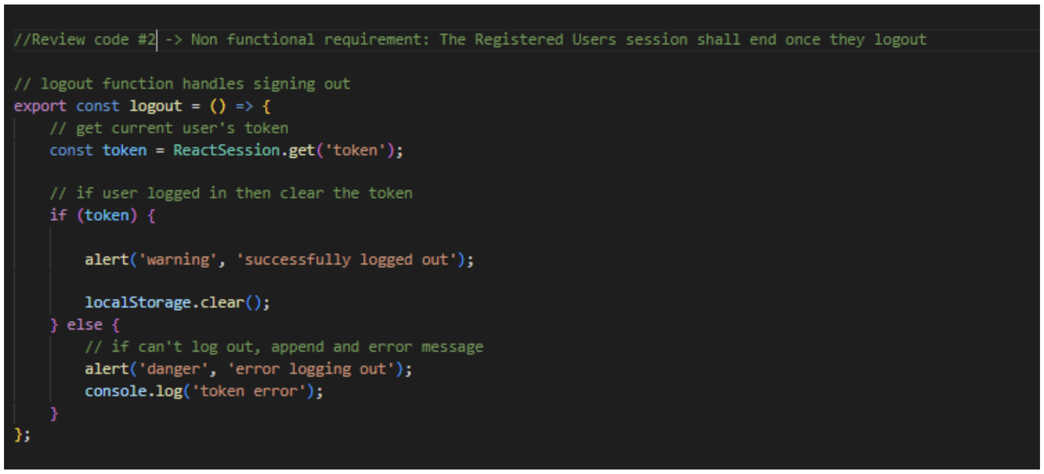
Team 3 Feedback:

- While you log errors to the console, also make sure to render an error page for the user.
- Good consistent code for what is given
- A bit too simple and brief and it only focuses on a basic functionality that is ubiquitous amongst a lot of web applications(basic search and authentication are common functionalities).
- Maybe use better url parameters as opposed to Articles&Essays and Art&Films? I think the ampersand is generally a safe character to use, so it's not a significant issue. But I personally think its better to follow existing uri naming standards.
- The code here is good code, it is organized well and optimized for readability. It is modular enough to be easily inserted into other functionalities.
- There are good comments that make it easier for a non-js developer to read and understand.
- Variable names follow styling guidelines and are pragmatic.

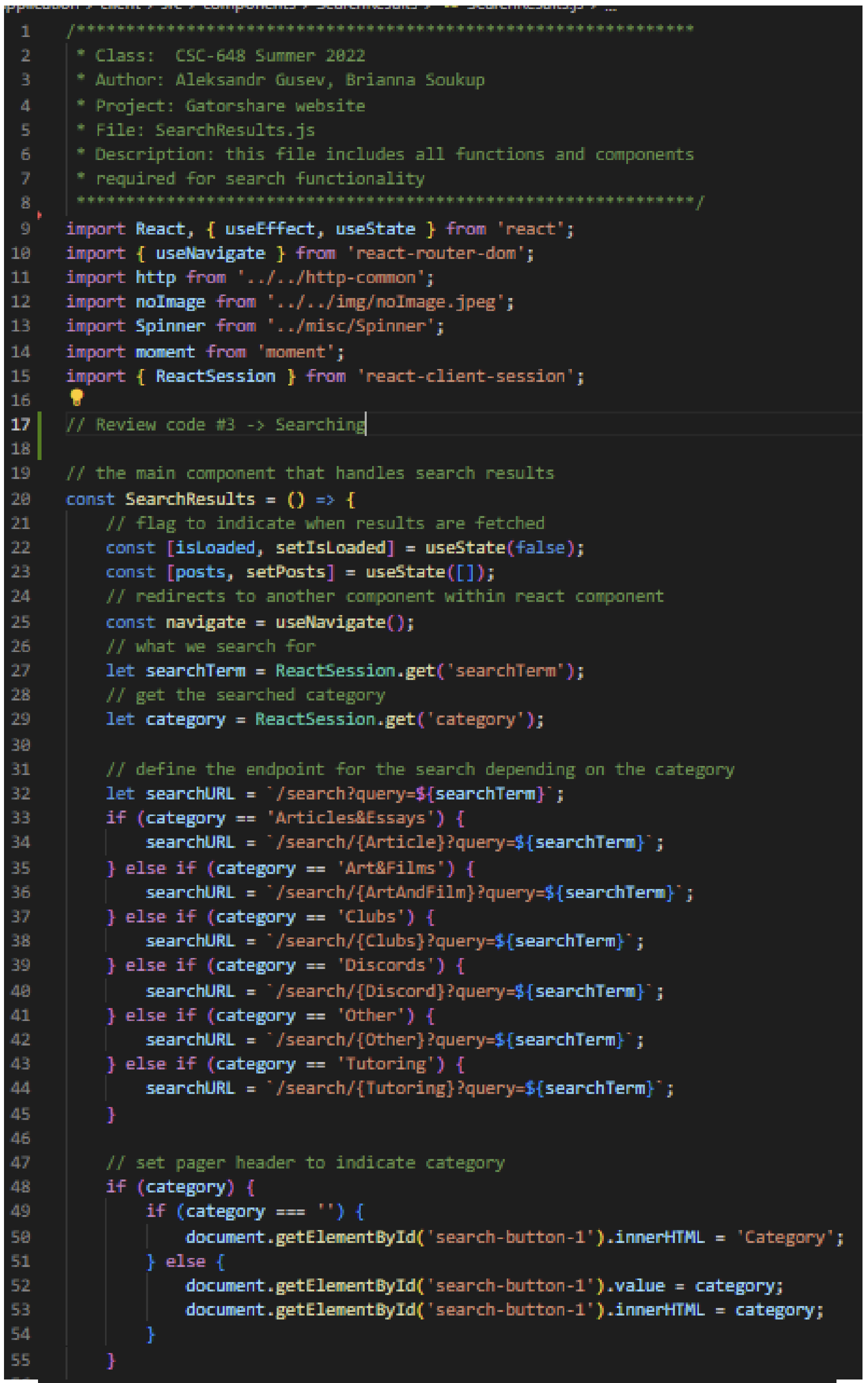
Code for Search and User Session -> Auth.js & SearchResults.js


auth.js M ×
application > client > src > js > auth.js > login > then() callback

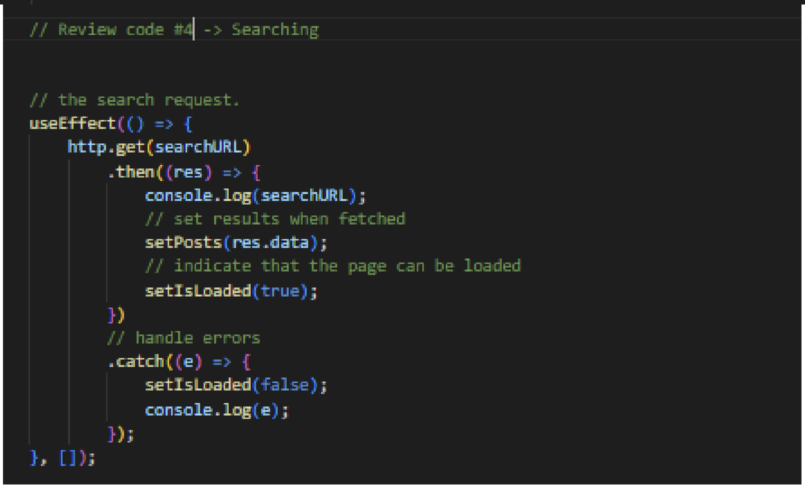
```
/****************************************************************
 * Class:  CSC-648 Summer 2022
 * Author: Aleksandr Gusev, Brianna Soukup
 * Project: Gatorshare website
 * File: auth.js
 * Description: this file includes all functions required for authentification
 ****************************************************************/
import { ReactSession } from 'react-client-session';

import http from '../http-common';
import { alert } from './alert';

//Review code #1 -> Non functional requirement: The Registered Users session shall end once they logout

// login function handles data for the Login page.
export const login = (formData) => {
    // the data for the axios request is stored here
    const { email, password } = formData;
    let id;

    // post request to the backend
    http.post('/login', {
        email: email,
        password: password,
    })
        .then((response) => {
            if (response.data.token) {
                const token = response.data.token;
                // user's id
                id = response.data.id;

                // store token and logged in user id
                ReactSession.set('token', token);
                ReactSession.set('currentUserId', id);
                // append success alert message
                alert('success', 'successfully logged in');

                // post request to get user's role, e.g. 'Student'
                http.get('/login/id/' + id)
                    .then((response) => {
                        ReactSession.set(
                            'userRole',
                            response.data.role_name[0].role_name
```



```
//Review code #2 -> Non functional requirement: The Registered Users session shall end once they logout

// logout function handles signing out
export const logout = () => {
    // get current user's token
    const token = ReactSession.get('token');

    // if user logged in then clear the token
    if (token) {

        alert('warning', 'successfully logged out');

        localStorage.clear();
    } else {
        // if can't log out, append and error message
        alert('danger', 'error logging out');
        console.log('token error');
    }
};
```



```javascript
/*****************************************************************
 * Class:  CSC-648 Summer 2022
 * Author: Aleksandr Gusev, Brianna Soukup
 * Project: Gatorshare website
 * File: SearchResults.js
 * Description: this file includes all functions and components
 * required for search functionality
 ****************************************************************/
import React, { useEffect, useState } from 'react';
import { useNavigate } from 'react-router-dom';
import http from '../../http-common';
import noImage from '../../img/noImage.jpeg';
import Spinner from '../misc/Spinner';
import moment from 'moment';
import { ReactSession } from 'react-client-session';

// Review code #3 -> Searching

// the main component that handles search results
const SearchResults = () => {
    // flag to indicate when results are fetched
    const [isLoaded, setIsLoaded] = useState(false);
    const [posts, setPosts] = useState([]);
    // redirects to another component within react component
    const navigate = useNavigate();
    // what we search for
    let searchTerm = ReactSession.get('searchTerm');
    // get the searched category
    let category = ReactSession.get('category');

    // define the endpoint for the search depending on the category
    let searchURL = `/search?query=${searchTerm}`;
    if (category == 'Articles&Essays') {
        searchURL = `/search/{Article}?query=${searchTerm}`;
    } else if (category == 'Art&Films') {
        searchURL = `/search/{ArtAndFilm}?query=${searchTerm}`;
    } else if (category == 'Clubs') {
        searchURL = `/search/{Clubs}?query=${searchTerm}`;
    } else if (category == 'Discords') {
        searchURL = `/search/{Discord}?query=${searchTerm}`;
    } else if (category == 'Other') {
        searchURL = `/search/{Other}?query=${searchTerm}`;
    } else if (category == 'Tutoring') {
        searchURL = `/search/{Tutoring}?query=${searchTerm}`;
    }

    // set pager header to indicate category
    if (category) {
        if (category === '') {
            document.getElementById('search-button-1').innerHTML = 'Category';
        } else {
            document.getElementById('search-button-1').value = category;
            document.getElementById('search-button-1').innerHTML = category;
        }
    }
```



```javascript
// Review code #4 -> Searching


// the search request.
useEffect(() => {
    http.get(searchURL)
        .then((res) => {
            console.log(searchURL);
            // set results when fetched
            setPosts(res.data);
            // indicate that the page can be loaded
            setIsLoaded(true);
        })
        // handle errors
        .catch((e) => {
            setIsLoaded(false);
            console.log(e);
        });
}, []);
```

Why this code:

The code in auth.js and SearchResults.js was chosen for code review as it relates to the search feature tested in the Usability Test and the non functional requirement for User sessions in the QA Test Plan. These two files effectively show how the team uses proper naming conventions, maintains clean, well-organized code, and provides helpful comments and headers.

Self-Check on Best Practices for Security

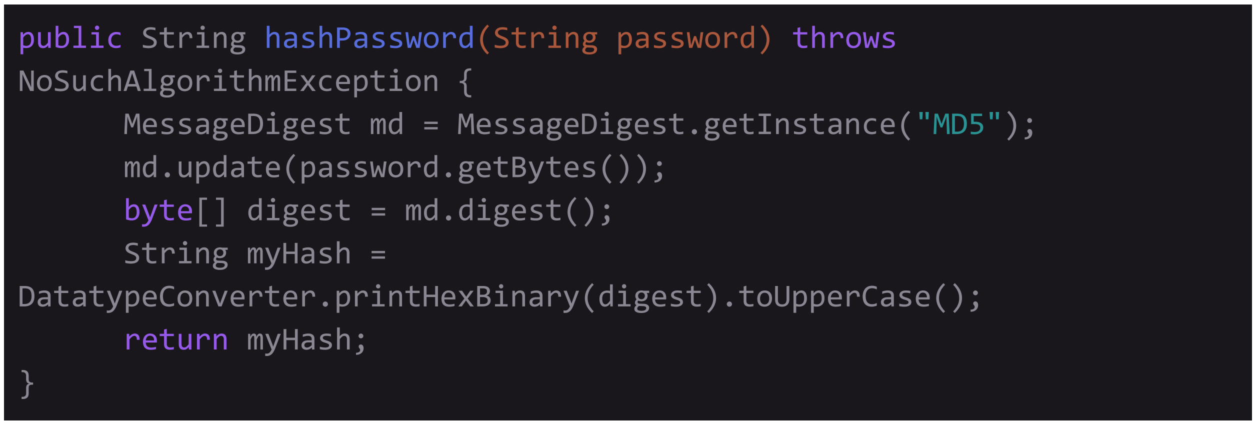
Major Assets that are Protected:

- User’s personal information
- User’s email address
- User’s password

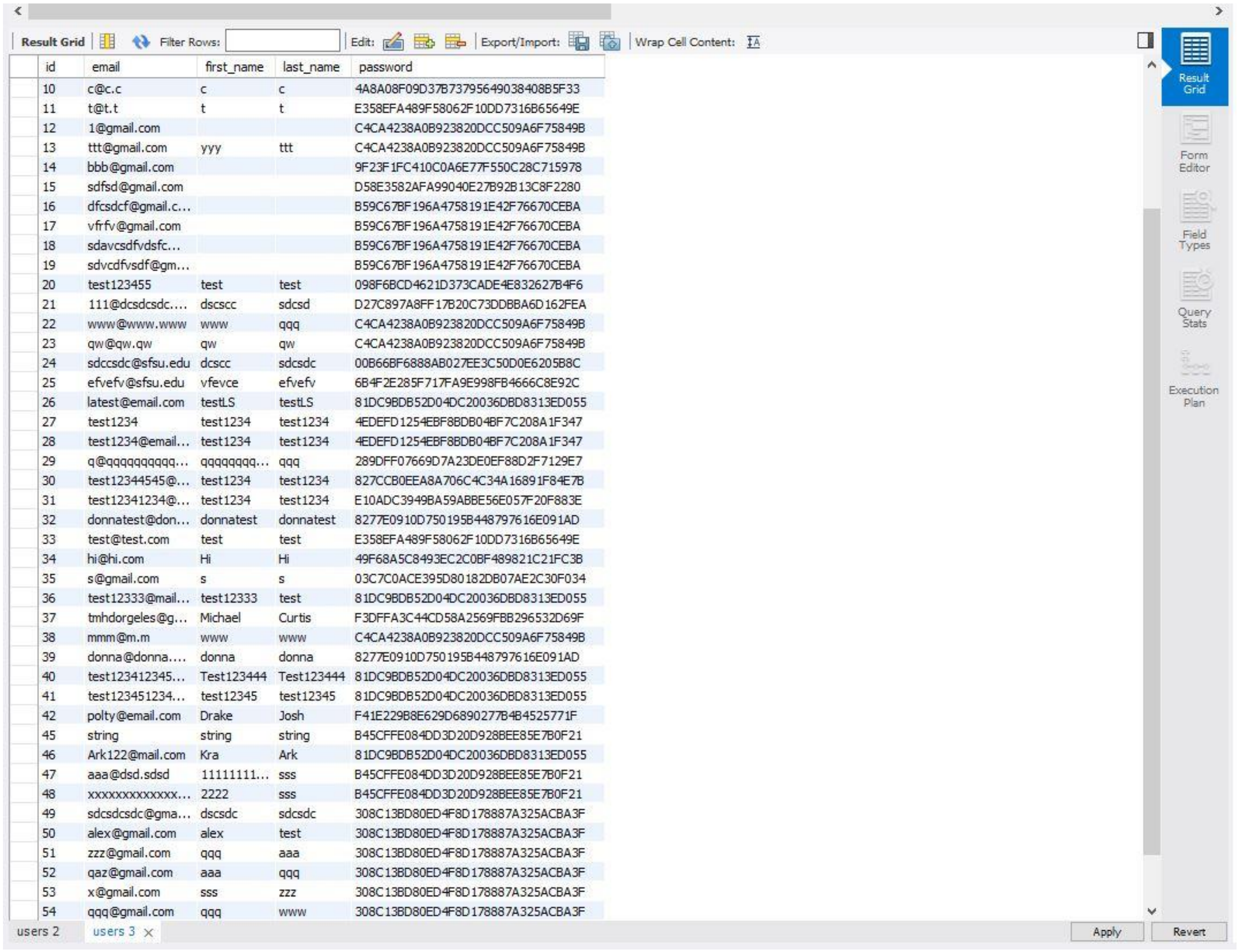
Confirm that you encrypt PW in the DB (with process and screenshots):

The User password is, indeed, encrypted. Without an RSA private key, unauthorized individuals cannot ssh into our database. When our Users create an account, they must enter their created password twice. When they input their password the second time, we check to see if it is the same as the first password input. Afterward, it is stored in our database. MD5 hashing is used to encrypt data in our database. We employ tokens (JWT) in addition to the hashing process to ensure that the user's information is valid and matches our database.

Here's is the method we use to hash the user password:



```
public String hashPassword(String password) throws
NoSuchAlgorithmException {
      MessageDigest md = MessageDigest.getInstance("MD5");
      md.update(password.getBytes());
      byte[] digest = md.digest();
      String myHash =
DatatypeConverter.printHexBinary(digest).toUpperCase();
      return myHash;
}
```



id	email	first_name	last_name	password
10	c@c.c	c	c	4A8A08F09D37B73795649038408B5F33
11	t@t.t	t	t	E358EFA489F58062F10DD7316B65649E
12	1@gmail.com			C4CA4238A0B923820DCC509A6F75849B
13	ttt@gmail.com	yyy	ttt	C4CA4238A0B923820DCC509A6F75849B
14	bbb@gmail.com			9F23F1FC410C0A6E77F550C28C715978
15	sdfsd@gmail.com			D58E3582AFA99040E27B92B13C8F2280
16	dfcsdcf@gmail.c...			B59C67BF196A4758191E42F76670CEBA
17	vfrfv@gmail.com			B59C67BF196A4758191E42F76670CEBA
18	sdavcsdfvdsfc...			B59C67BF196A4758191E42F76670CEBA
19	sdvcdfvsdf@gm...			B59C67BF196A4758191E42F76670CEBA
20	test123455	test	test	098F6BCD4621D373CADE4E832627B4F6
21	111@dcsdcsdc....	dscscc	sdcsd	D27C897A8FF17B20C73DDBBA6D162FEA
22	www@www.www	www	qqq	C4CA4238A0B923820DCC509A6F75849B
23	qw@qw.qw	qw	qw	C4CA4238A0B923820DCC509A6F75849B
24	sdccsdc@sfsu.edu	dcscc	sdcsdc	00B66BF6888AB027EE3C50D0E6205B8C
25	efvefv@sfsu.edu	vfevce	efvefv	6B4F2E285F717FA9E998FB4666C8E92C
26	latest@email.com	testLS	testLS	81DC9BDB52D04DC20036DBD8313ED055
27	test1234	test1234	test1234	4EDEFD1254EBF8BDB04BF7C208A1F347
28	test1234@email...	test1234	test1234	4EDEFD1254EBF8BDB04BF7C208A1F347
29	q@qqqqqqqqqq...	qqqqqqqq...	qqq	289DFF07669D7A23DE0EF88D2F7129E7
30	test12344545@...	test1234	test1234	827CCB0EEA8A706C4C34A16891F84E7B
31	test12341234@...	test1234	test1234	E10ADC3949BA59ABBE56E057F20F883E
32	donnatest@don...	donnatest	donnatest	8277E0910D750195B448797616E091AD
33	test@test.com	test	test	E358EFA489F58062F10DD7316B65649E
34	hi@hi.com	Hi	Hi	49F68A5C8493EC2C0BF489821C21FC3B
35	s@gmail.com	s	s	03C7C0ACE395D80182DB07AE2C30F034
36	test12333@mail...	test12333	test	81DC9BDB52D04DC20036DBD8313ED055
37	tmhdorgeles@g...	Michael	Curtis	F3DFFA3C44CD58A2569FBB296532D69F
38	mmm@m.m	www	www	C4CA4238A0B923820DCC509A6F75849B
39	donna@donna....	donna	donna	8277E0910D750195B448797616E091AD
40	test123412345...	Test123444	Test123444	81DC9BDB52D04DC20036DBD8313ED055
41	test123451234...	test12345	test12345	81DC9BDB52D04DC20036DBD8313ED055
42	polty@email.com	Drake	Josh	F41E229B8E629D6890277B4B4525771F
45	string	string	string	B45CFFE084DD3D20D928BEE85E7B0F21
46	Ark122@mail.com	Kra	Ark	81DC9BDB52D04DC20036DBD8313ED055
47	aaa@dsd.sdsd	11111111...	sss	B45CFFE084DD3D20D928BEE85E7B0F21
48	xxxxxxxxxxxxx...	2222	sss	B45CFFE084DD3D20D928BEE85E7B0F21
49	sdcsdcsdc@gma...	dscsdc	sdcsdc	308C13BD80ED4F8D178887A325ACBA3F
50	alex@gmail.com	alex	test	308C13BD80ED4F8D178887A325ACBA3F
51	zzz@gmail.com	qqq	aaa	308C13BD80ED4F8D178887A325ACBA3F
52	qaz@gmail.com	aaa	qqq	308C13BD80ED4F8D178887A325ACBA3F
53	x@gmail.com	sss	zzz	308C13BD80ED4F8D178887A325ACBA3F
54	qqq@gmail.com	qqq	www	308C13BD80ED4F8D178887A325ACBA3F

Confirm Input data validation (list what is being validated and what code you used):

Articles & Essays, Art & Film, Clubs, Discords, Tutoring, Posts, and Other are what we look for in the search box input. To do a search, we utilize SQL query. Each search, including the criteria, is not hard programmed.

Here's is the query that was used to do filter and search from the post:

```
("SELECT p FROM Post p WHERE p.Title LIKE CONCAT('%',:query, '%') Or p.Description
LIKE CONCAT('%', :query, '%')")
```

```
@Query("SELECT p FROM Post p where p.Tag LIke '%Article%'")

@Query("SELECT p FROM Post p where p.Tag LIke '%Essay%'")

@Query("SELECT p FROM Post p where p.Tag LIke '%ArtAndFilm%'")

@Query("SELECT p FROM Post p where p.Tag LIke '%Discords%'")

@Query("SELECT p FROM Post p where p.Tag LIke '%Tutoring%'")

@Query("SELECT p FROM Post p where p.Tag LIke '%Clubs%'")

@Query("SELECT p FROM Post p where p.Tag LIke '%others%'")

@Query("SELECT p FROM Post p ORDER BY p ASC")

@Query("SELECT p FROM Post p ORDER BY p.photo_Like DESC")
```

Self-Check: Adherence to Original Non-Functional Specs

System requirements:

1. The website application shall be optimized for selected browsers such as Safari, Google Chrome, and Mozilla Firefox. **-DONE**
2. The website application shall display posted media in full width of the screen. **-DONE**
3. Images posted on the application shall be in the format of jpg, png, or jpeg only. **-DONE**

Performance requirements:

4. The website application shall be optimized for desktop/laptop usage. **-DONE**
5. Loading time for the website shall be 5 seconds or less for any page or screen. **-IN PROGRESS**

Storage, security, environmental requirements

6. The website shall handle at least 50 users simultaneously. **-IN PROGRESS**
7. Data shall be stored in the team's SQL database. **-DONE**
8. React UI component's names shall be PascalCase. **-DONE**
9. All other helper files shall be camelCase. **-DONE**
10. Inline styles shall be camelCase. **-DONE**
11. CSS files shall be named using kebab-case. **-DONE**
12. React classes within components shall follow react naming conventions. **-DONE**
13. Multiple files of code shall be written. **-DONE**
14. All CSS files shall be in one common folder. **-DONE**
15. All files shall be named logically according to the job that they perform. **-DONE**

16. Props shall be destructured in order to make the code cleaner and more maintainable. **-DONE**
17. All imports shall be in an order: React import, Library imports (Alphabetical order), Absolute imports (Alphabetical order), Relative imports (Alphabetical order), Import * as. **-IN PROGRESS**
18. There shall be no unneeded comments in code. **-DONE**
19. All debugging console.log() shall be removed in code. **-DONE**
20. .js extensions shall be used for React components. **-DONE**

Marketing, legal requirements (logos, branding, licensing)

21. Specific logos shall be created for this project only. **-DONE**
22. Some images and logos shall be used from San Francisco State University's Website. **-DONE**
23. All icons and frameworks used shall be from a free, open-source library. **-DONE**

Content (size, formats…)

24. Comments shall have a text limit of 250 characters. **-IN PROGRESS**

Privacy (what data is collected, how is it used…)

25. The login shall be required for all Registered Users (Approved Users and Admin). **-DONE**
26. Passwords shall be encrypted in the database where it is stored. **-DONE**
27. The Registered Users session shall end once they click "Log Out". **-DONE**

Optimization & Performance:

28. The website application language shall be in English. **-DONE**

29. The application will upload User's posts and comments in real time. **-DONE**
30. The website application shall be deployed from the team's Amazon Web Services account. **-DONE**
31. The website application shall store the Users data in the team's MySQL database. **-DONE**
32. User information shall not be duplicated in the database. **-DONE**

Security:

33. Users data shall be securely stored in the backend database. **-DONE**
34. Passwords shall be protected. **-DONE**
35. Passwords shall not be stored in the database as plain text. **-DONE**
36. When an Unregistered User signs up: **-DONE**
    - Passwords shall be at least 8 characters. **-DONE**
    - Passwords shall require at least one uppercase letter. **-DONE**
    - Passwords shall require at least one lowercase letter. **-DONE**
    - Passwords shall require at least one number. **-DONE**
    - Passwords shall require at least one special character. **-DONE**

List of Contributions to the Document

Estefanos (Backend Lead) -

Participated in team meetings and discussions. Worked on the comment section. Helped implement and finalize messaging API. Worked on posts. Implemented group chat. Completed all features besides the comment section. Created new tables to have working back end and front end. Worked on group messaging, post, and comment end points. Wrote chatroom.js, group message dto, dao and controllers. Helped Database team with Best Practices for Security. Helped draft Best Practices for Security. Introduced sockjs and STOMP for web-socket purposes.

Brianna (Frontend Lead) -

Contributed to discussions on Discord. Attended all team meetings. Drafted, reviewed, and finalized Code Review. Provided Team 3 with the code for Code Review. Reviewed Team 3 code. Designated M4 editor. Added last bits of CSS styling. Reviewed and revised Usability Test Plan. Reviewed and Revised QA Test Plan. Redid User Satisfaction Table for posting. Finalized all sections of the M4 Doc before submission.

Brian (Database Master) -

Participated in team meetings and discussions. Helped contribute to Product Summary by drafting. Helped create and finalize user test plan questionnaires (user satisfaction table). Drafted, reviewed, and finalized Best Practices for Security. Helped with Code Review by giving feedback. Reviewed all sections of the Doc.

Mohamed (GitHub Master) -

Participated in team meetings and discussions. Brainstormed on the final modifications and input ideas for improvement on the platform. Performed a series of testing for the platform and submitted feedback describing the task performed. Worked on the project descriptions. Had a Tester for our application. Drafted and filled out our Usability Test Table. Drafted and finalized the Task Description. Finalized and tested our User Satisfaction Table.

Aleksandr (Front End) -

Contributed to discussions on Discord. Attended all team meetings. Contributed to code review. Created ‘edit’ post functions. Created access to other user’s profile pages. Edited the ‘messages’ text box. Fixed search bar functionalities. Added categories to the search bar. Added sorting to the posts. Refactored the code to send fewer requests. Updated links to search bar. Prepared code for review and slightly helped review the code for the other team. Reviewed requirements priorities.