The New and Better Reddit © 2021

Final Report and Documentation

Team Members

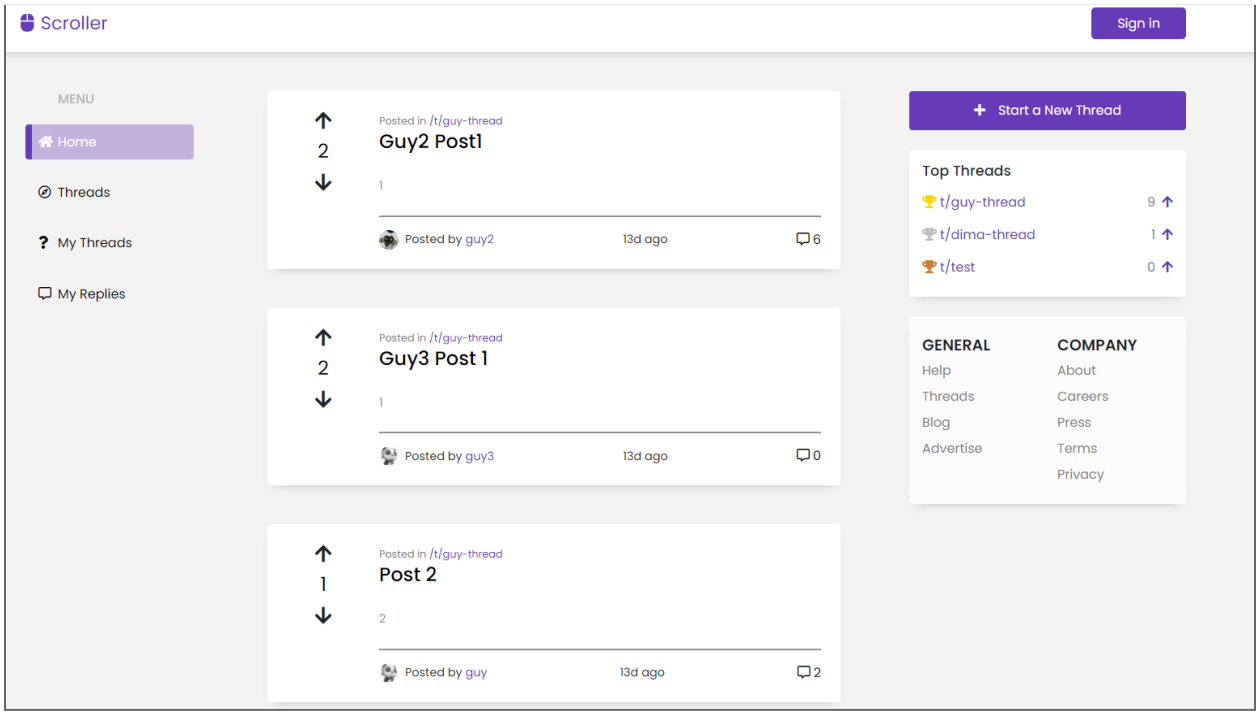
Guy Kaminsky

Dima Zhuravel

Walkthrough

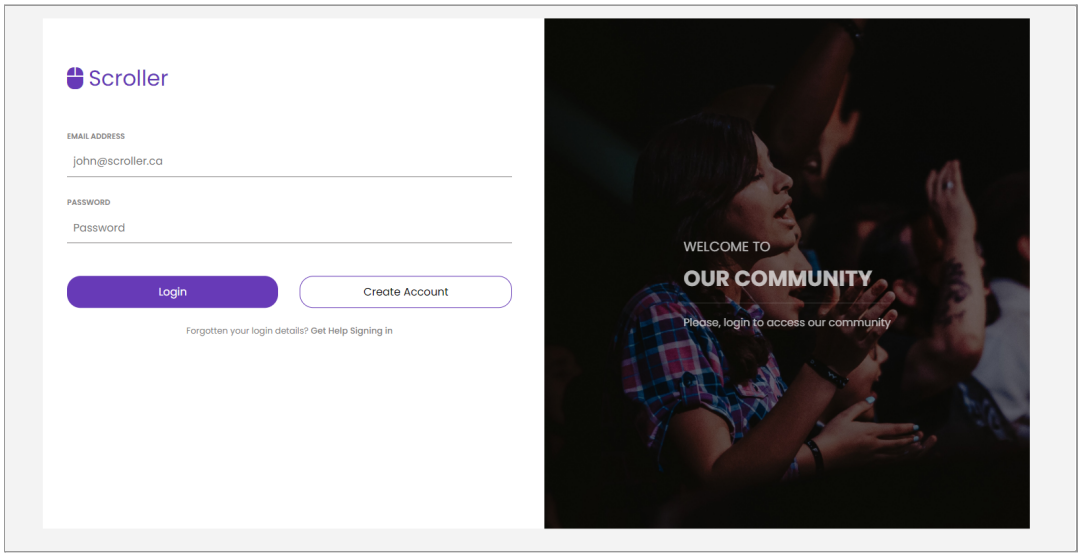
1. Homepage

This is the home page of Scroller. Without needing any login information, an unregistered user can browse through the most popular posts on the home page. A limitation of an unregistered user is that they will be unable to click on a post to load all of its comments, as they will be redirected to the login page. In addition, if an unregistered user clicks on the “Start a New Thread” button, they will be redirected to the login page. To access particular posts, a user must have a registered account. Also, if an unregistered user clicks on any of the menu buttons, all they will be able to do is search through all threads. One thing to note, on the right-hand side, users can find the Top Threads. These are the threads with the most posts in them. Lastly, to create an account or log into one, the unregistered users must click the “Sign in” button located at the top right on the navigation bar.



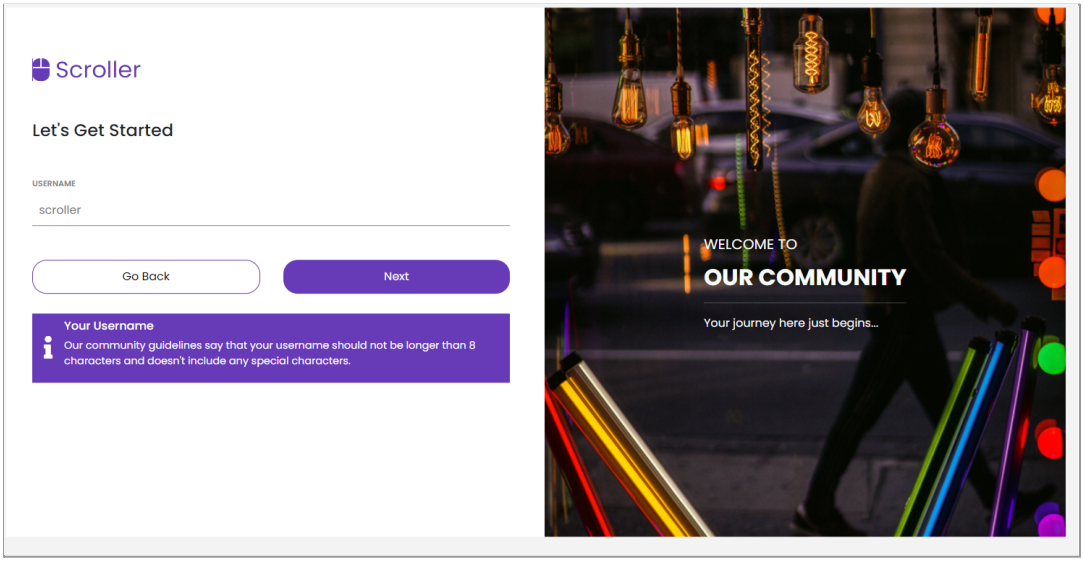
2. Login

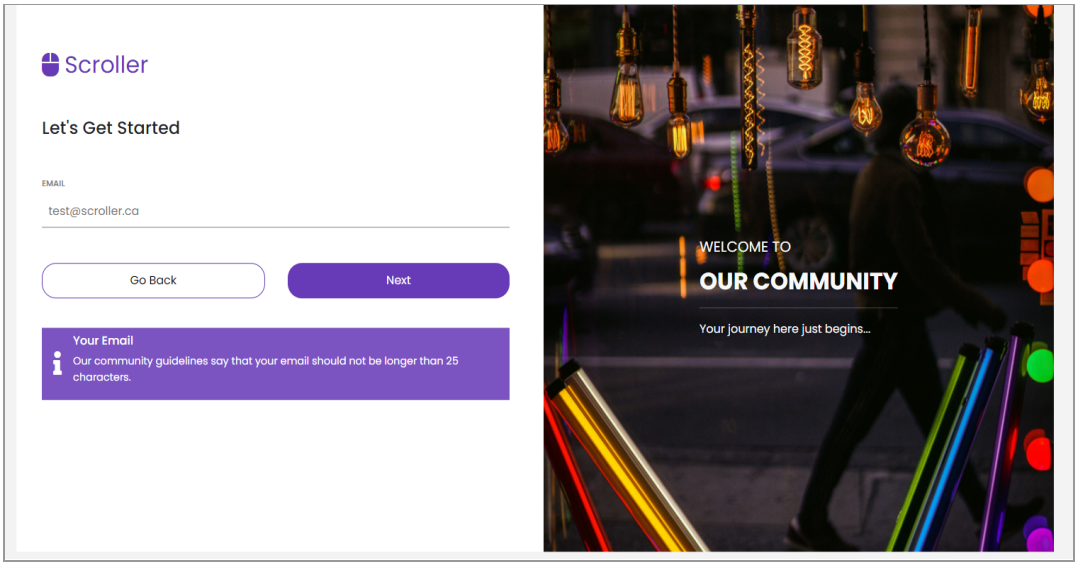
The user is redirected to the Login page after clicking on the “Sign in button” from the home page. If a user has an account, they can input their credentials and click “Login in.” Otherwise, an unregistered user must click the “Create Account” button.

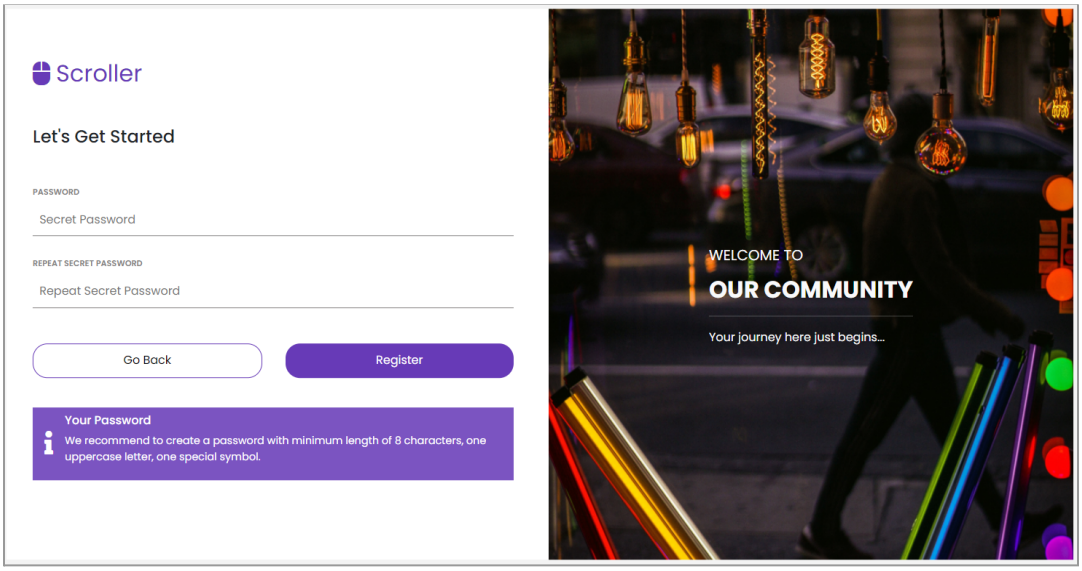


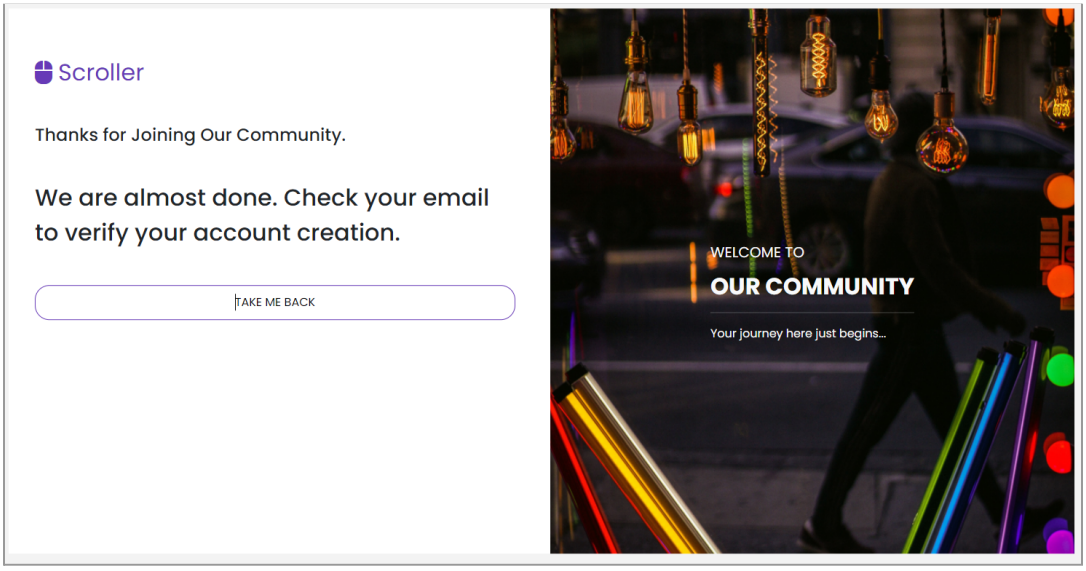
3. Register

After clicking “Create Account,” the user goes to the registration page where they will be asked to input a username, and then when clicking the “Next” button, they can input their email, username, and password. If the user clicks on this page by mistake, they can click the “Go Back” button.

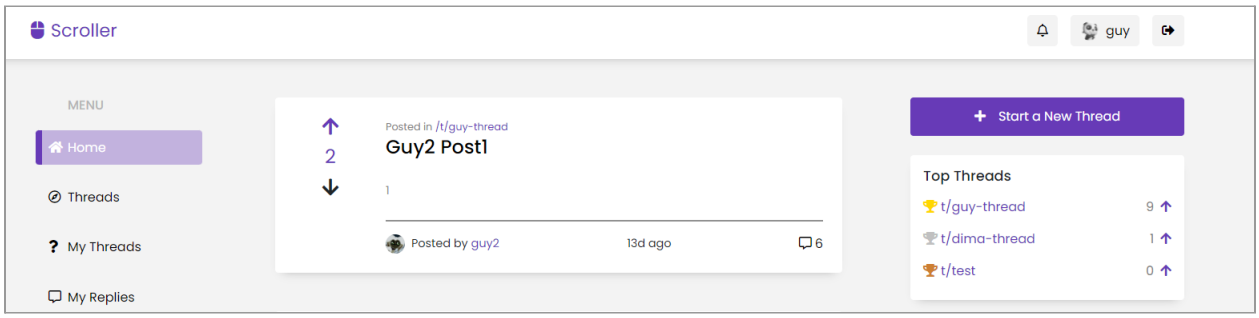



Scroller
Let's Get Started
EMAIL
test@scroller.ca
Go Back
Next
Your Email
Our community guidelines say that your email should not be longer than 25 characters.
WELCOME TO
OUR COMMUNITY
Your journey here just begins...


Scroller
Let's Get Started
PASSWORD
Secret Password
REPEAT SECRET PASSWORD
Repeat Secret Password
Go Back
Register
Your Password
We recommend to create a password with minimum length of 8 characters, one uppercase letter, one special symbol.
WELCOME TO
OUR COMMUNITY
Your journey here just begins...

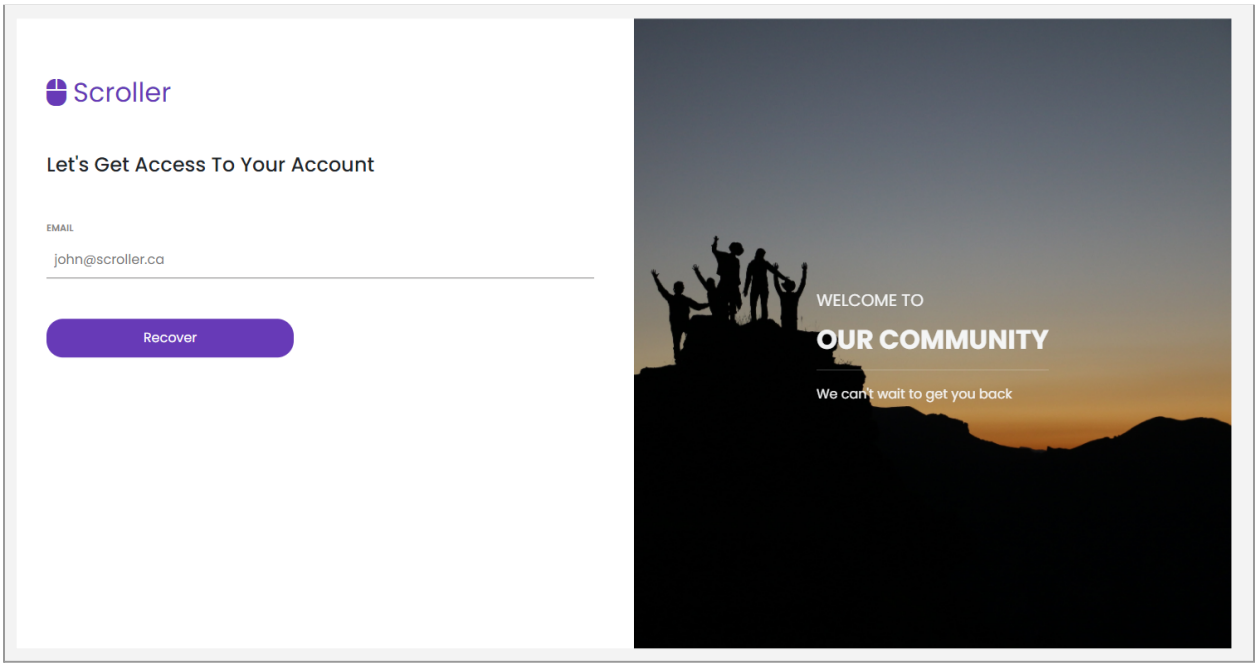

Scroller
Thanks for Joining Our Community.
We are almost done. Check your email to verify your account creation.
TAKE ME BACK
WELCOME TO
OUR COMMUNITY
Your journey here just begins...

After registering, the user needs to confirm their account via email. Then they can click the “TAKE ME BACK” button to be redirected to the login page. On the login page, the user can input their credentials to log into the site. The logged-in user will notice their username appears on the navigation bar.



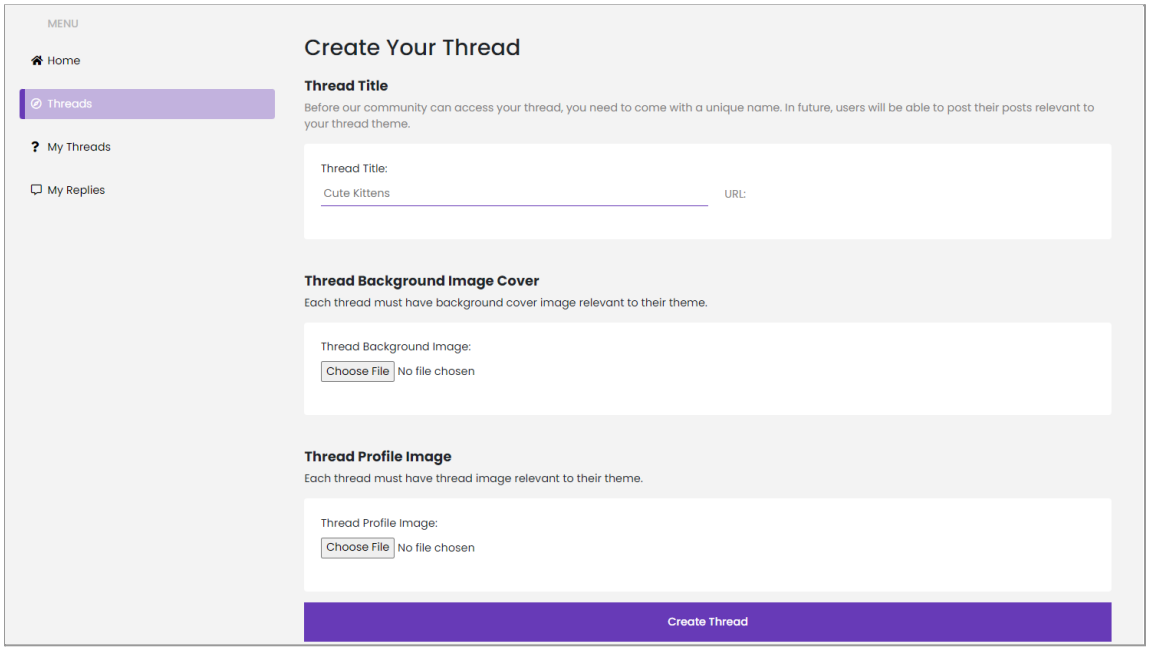
4. Restore Account

If the user forgot their login credentials, they could click on the “Get Help Sign in” button from the login page. Then, the user can input the email they used to register to reset their password.



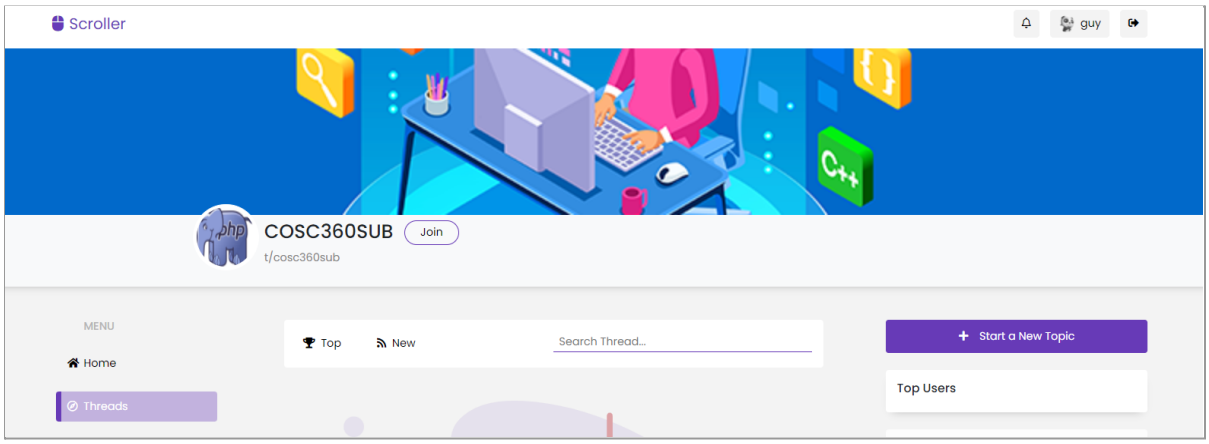
5. Create Thread

After clicking the “Start a New Thread,” the user is redirected to the “Create Thread” page. Here they must input all three fields to create the thread. Once they input all valid fields, they will be redirected to their newly created thread after clicking the “Create Thread” button.



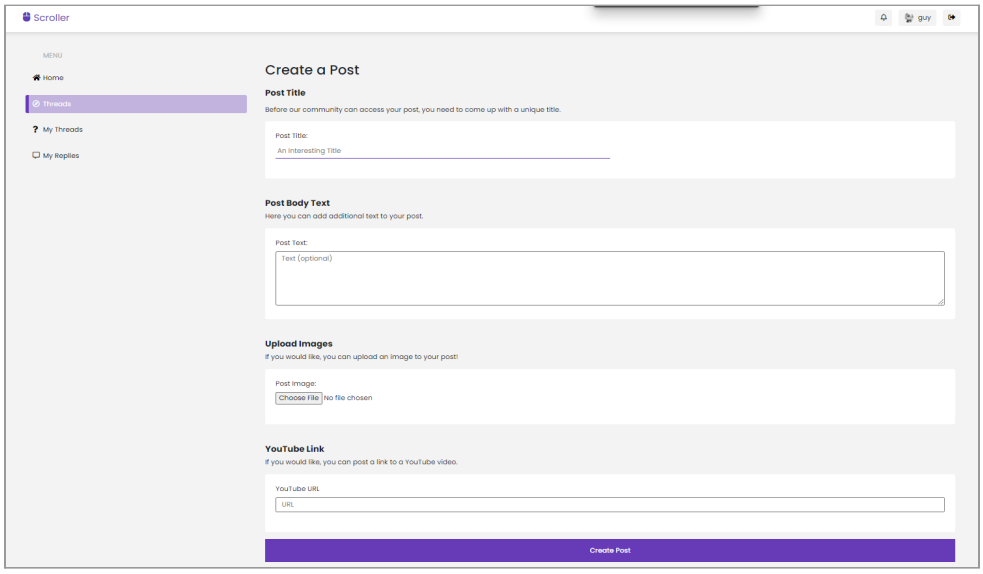
6. Empty Thread Page

Here is an example of a newly created empty thread page. A user can leave or join the thread. The button is updated dynamically, and the user does not need to refresh the page.

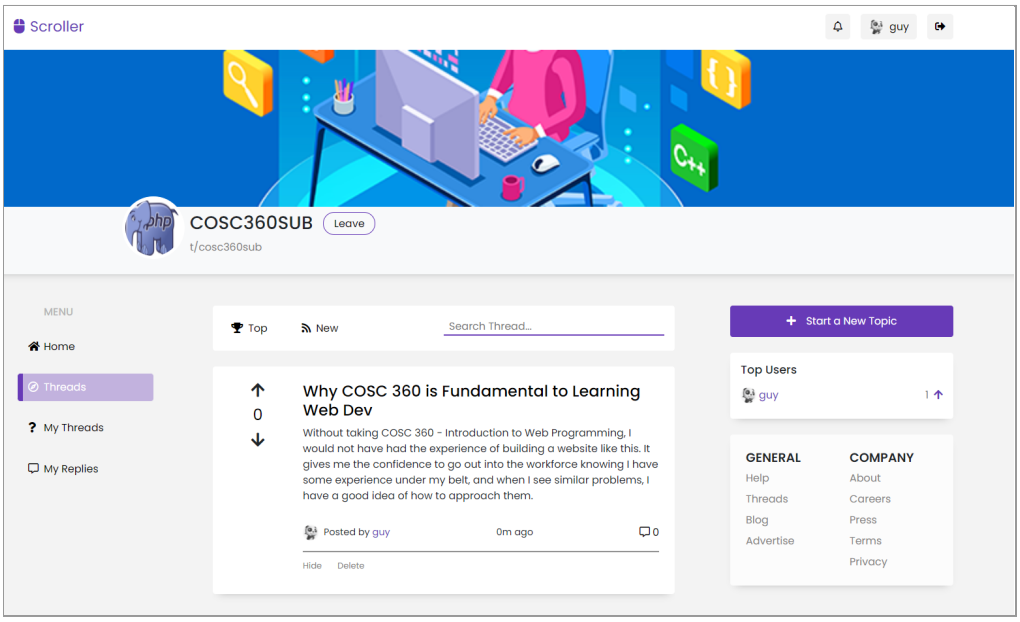


7. Create Post

When users want to create a post for a particular thread, they have to click the “Start a New Topic” button on the thread page. To create a post, the user must input a valid title and at least one of the fields “Post Body Text,” “Upload Images,” or “YouTube Link” to successfully create a post. After inputting their desired fields, the user can click the “Create Post” button. Then, they will be redirected to the thread where the post belongs and see their post.



8. Non-Empty Thread Page



There are a few things to note on the threads page:

- On the right-hand side, there is a “Top Users” tab. Here, the top five users with the most number of posts in the particular thread will appear.

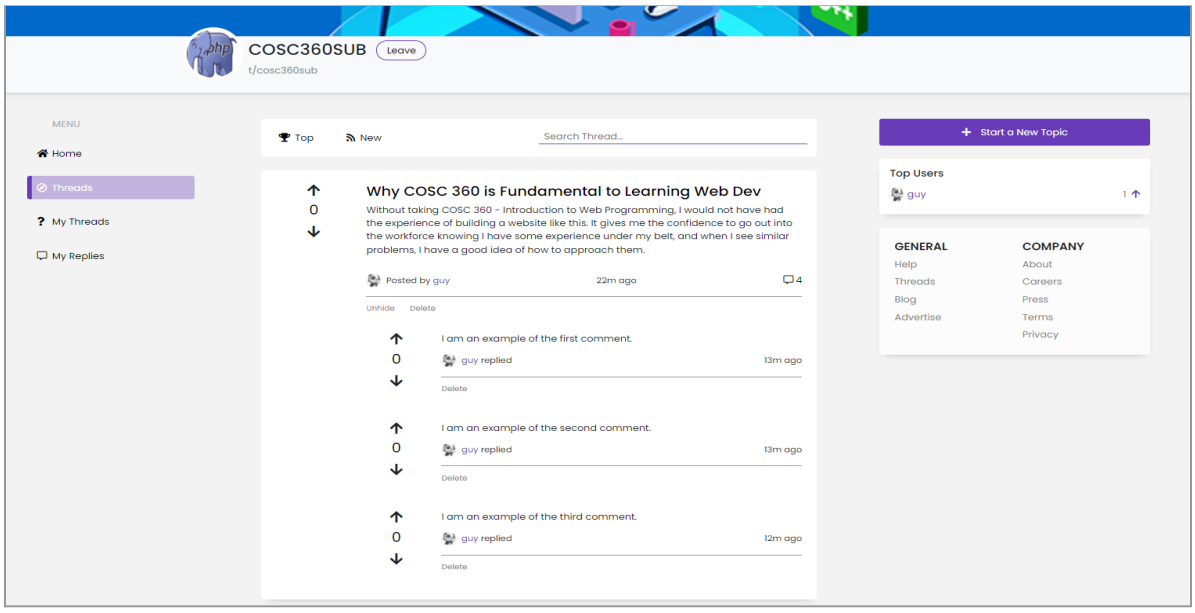
- A user can vote on the post without entering the post.

- The posts can be sorted by “Top” or “New.” When clicking “Top,” the posts are sorted by the most number of votes. When clicking “New,” the posts are sorted based on the time they were created in ascending order. This sort is done dynamically.

- The owner of the post or an admin can “hide” or “delete” the post without entering the post. Deleting a post will remove the post from the thread. Hiding the post will disable the post. Meaning no one can comment on the post or vote on the post or its comments. However, the post is still visible to the users. Hiding and deleting a post can also be done within the post if chosen.

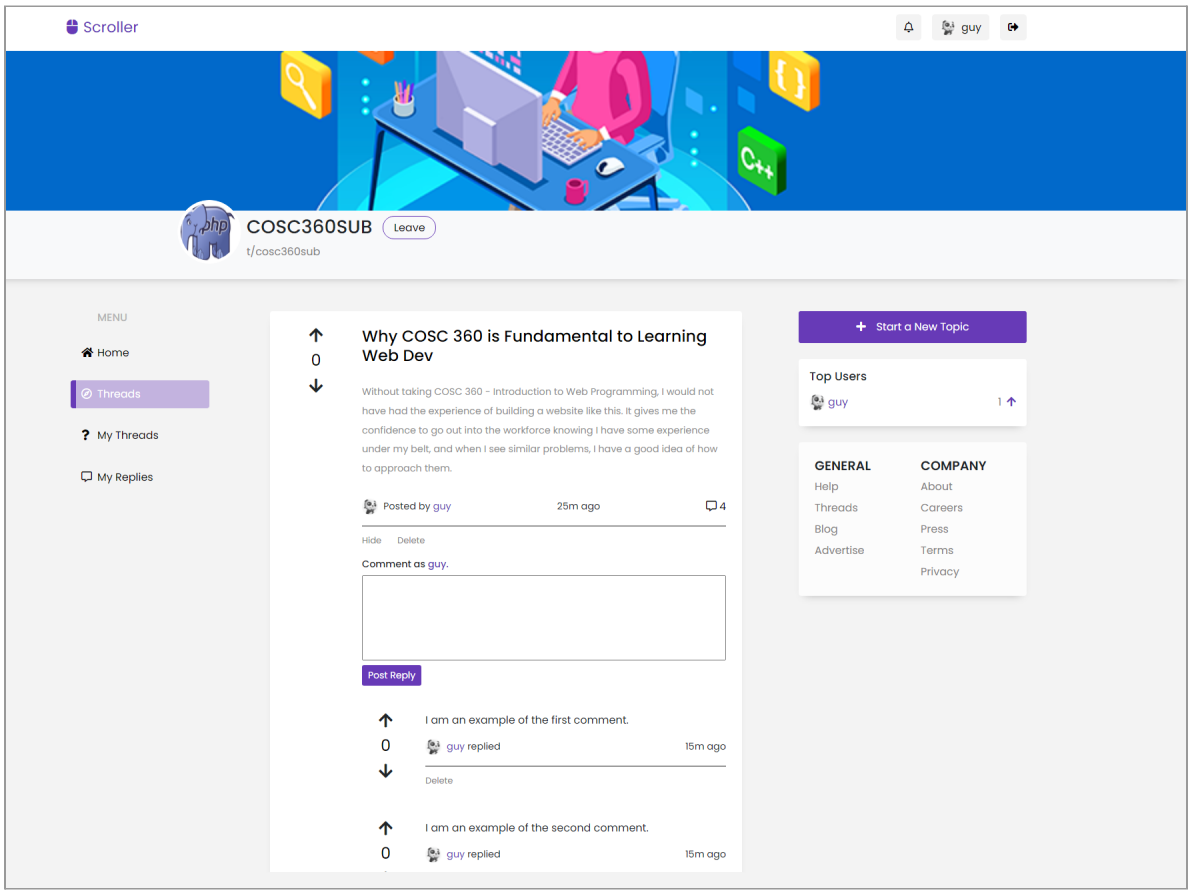
9. Thread Page With Comments

On the threads, the top three comments are also shown and can be voted on. To see all the comments, a user must click on the post. Users can vote on the top three comments without entering the post. The owner of the post or admin can also delete one of the top three comments without entering the post.



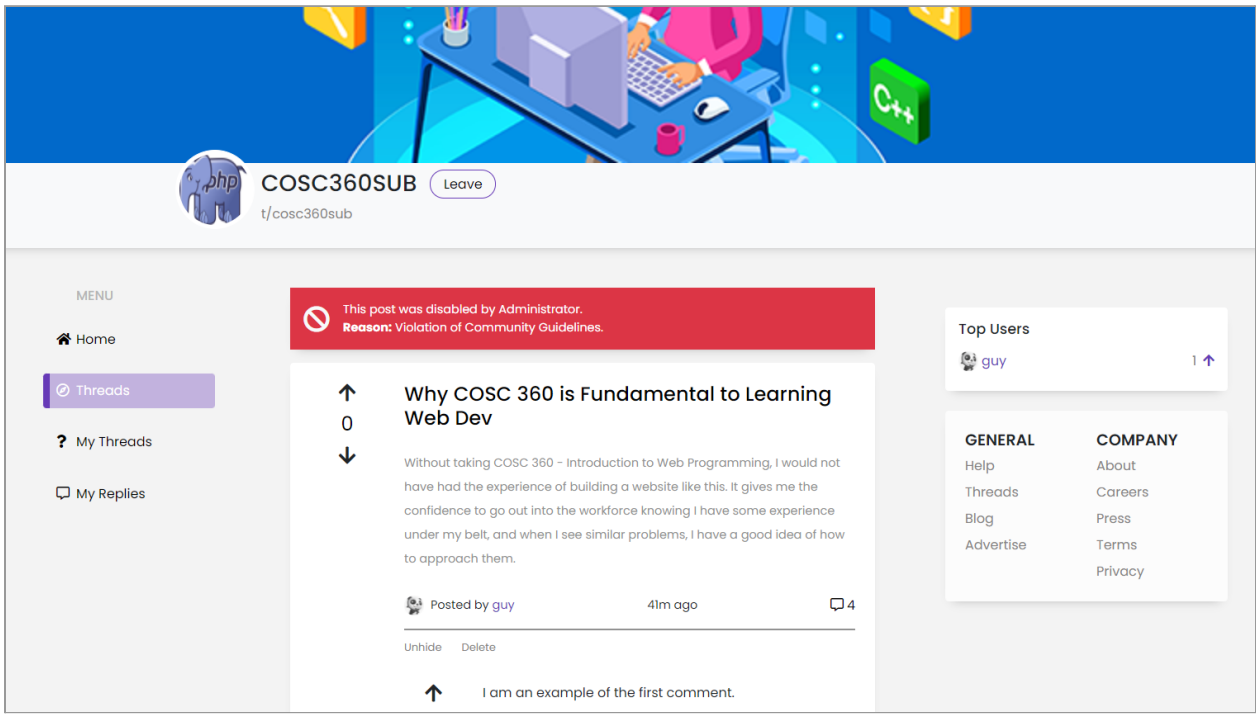
10. Post Page

On the post page, users can vote on the post or the comments of the post. Additionally, users can insert a comment which will appear on the post. Comments are sorted from newest to oldest. If the user clicks on the sub-thread located under the thread title, they will be redirected to the main sub-thread page. Additionally, owners of the post or admins can delete or hide posts and comments from this page.



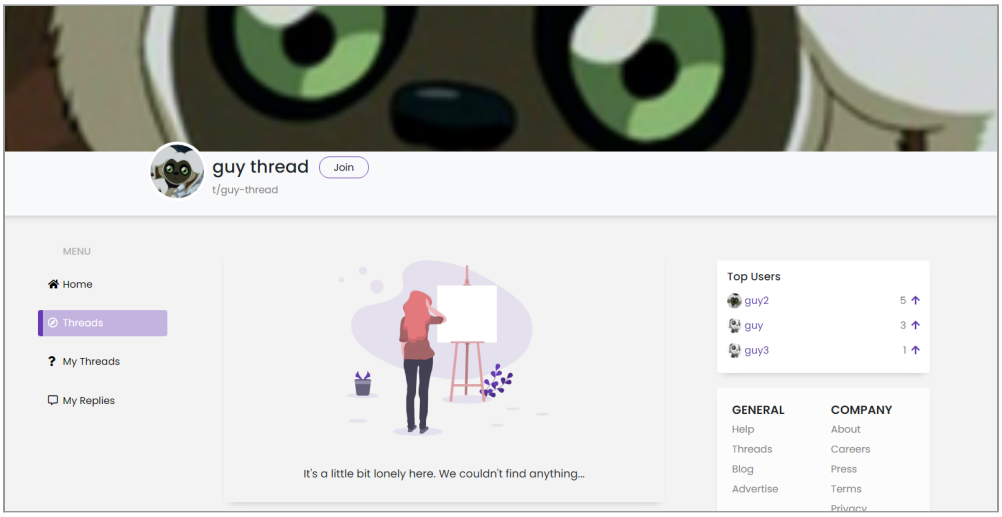
11. Hidden Post Page

The users can still access this post, but they cannot vote on its comments. Additionally, users cannot insert a comment into the post until it is unhidden.



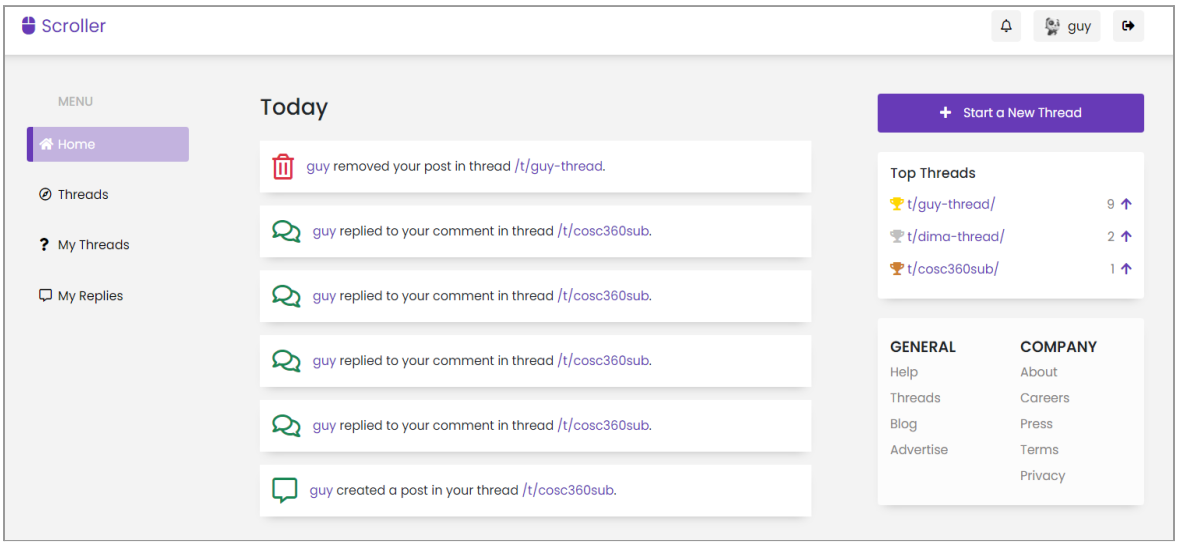
12. Deleting Post From the Post Page

If the owner of the post or admin chooses to delete a post from the post page rather than from the threads, this is how the page will appear.



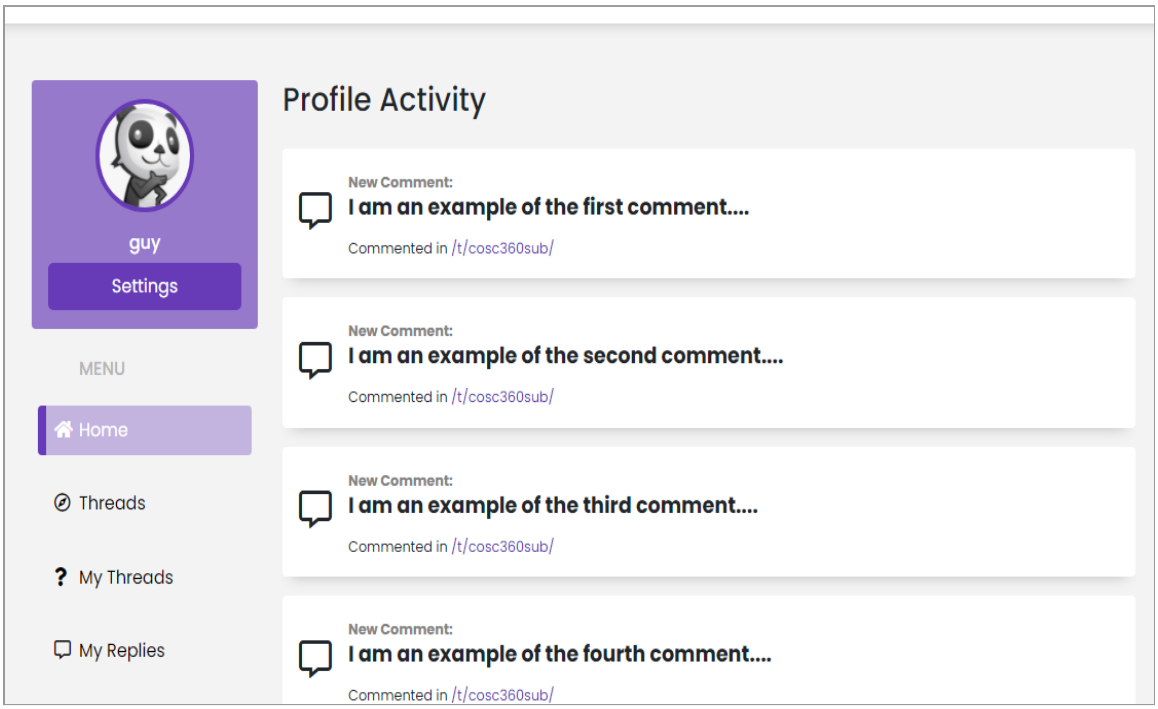
13. User Notifications

When clicking on the bell icon on the notification bar, the user will see different activities in their active sub-threads.



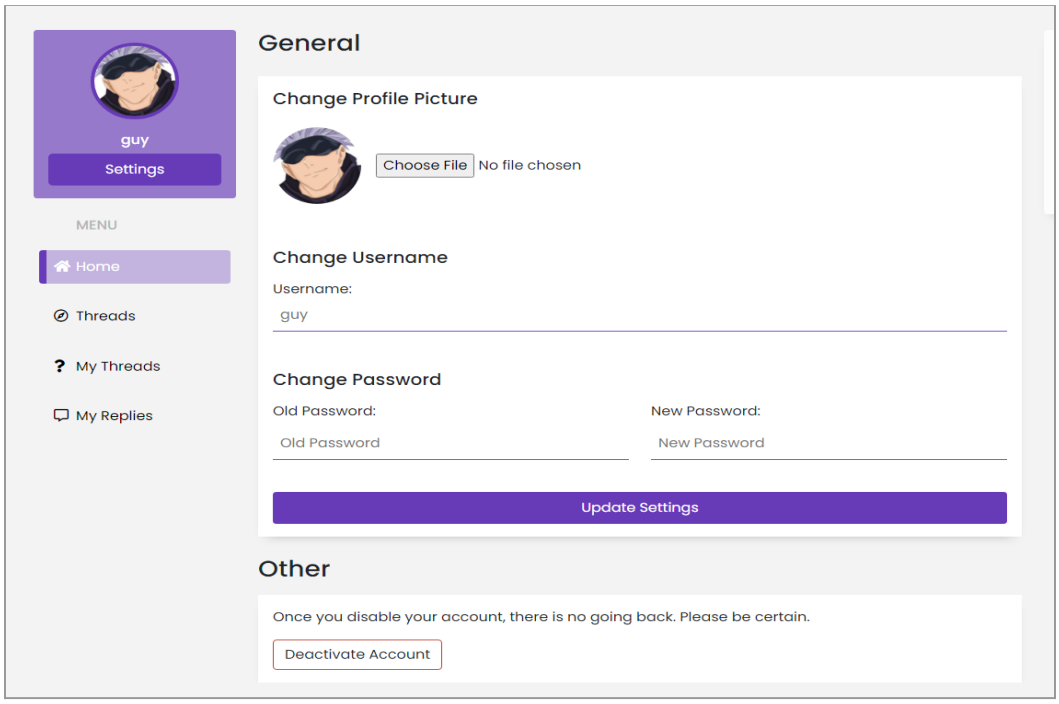
14. Account Page

Here the user can see all of the comments they posted in different threads.



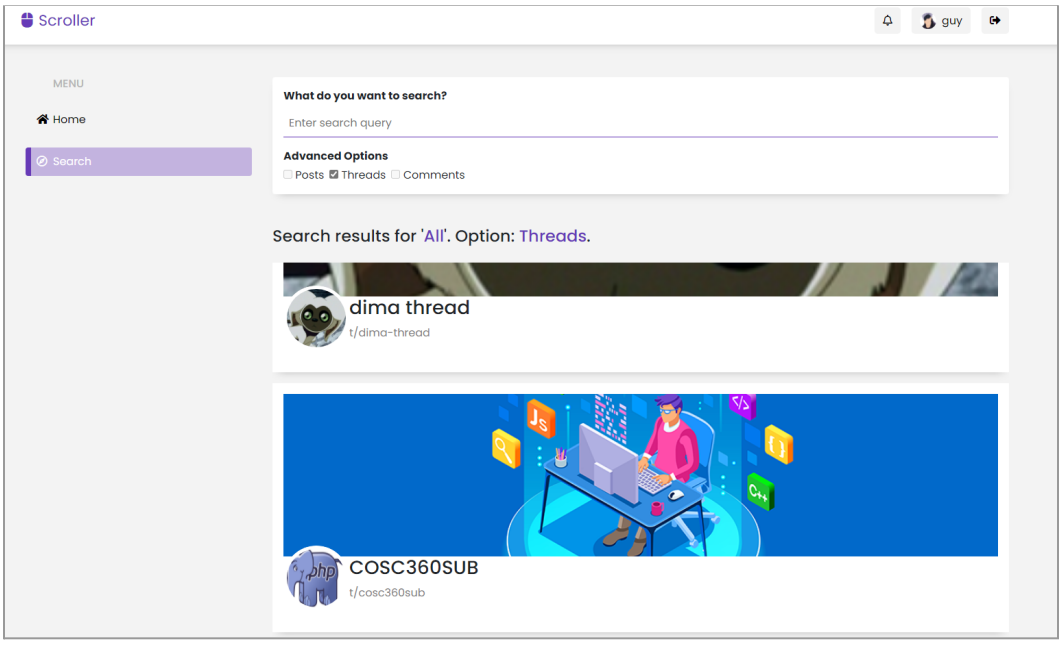
15. Settings Page

The user can change their profile picture, username, password or deactivate their account on this page.

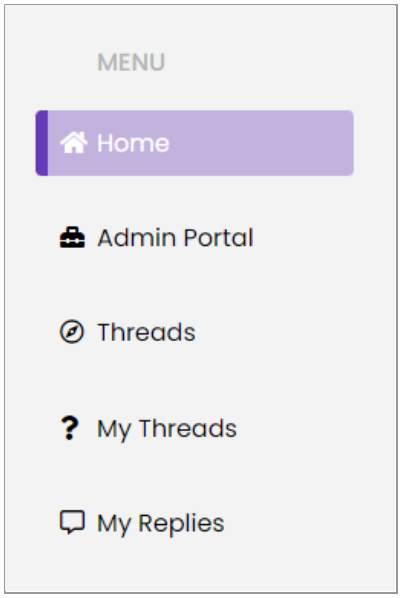


16. Search Page

Clicking on any of the left-hand buttons will lead the user to the search page. On this page, the user can search for specific posts, threads, or comments.



If a user is an admin, they will have the “Admin Portal” appear on their left-hand side. This option allows the extra admin functionality. To log in as an admin, here is the following email and password to use:



ADMIN AND USER CREDENTIALS

For Admin:

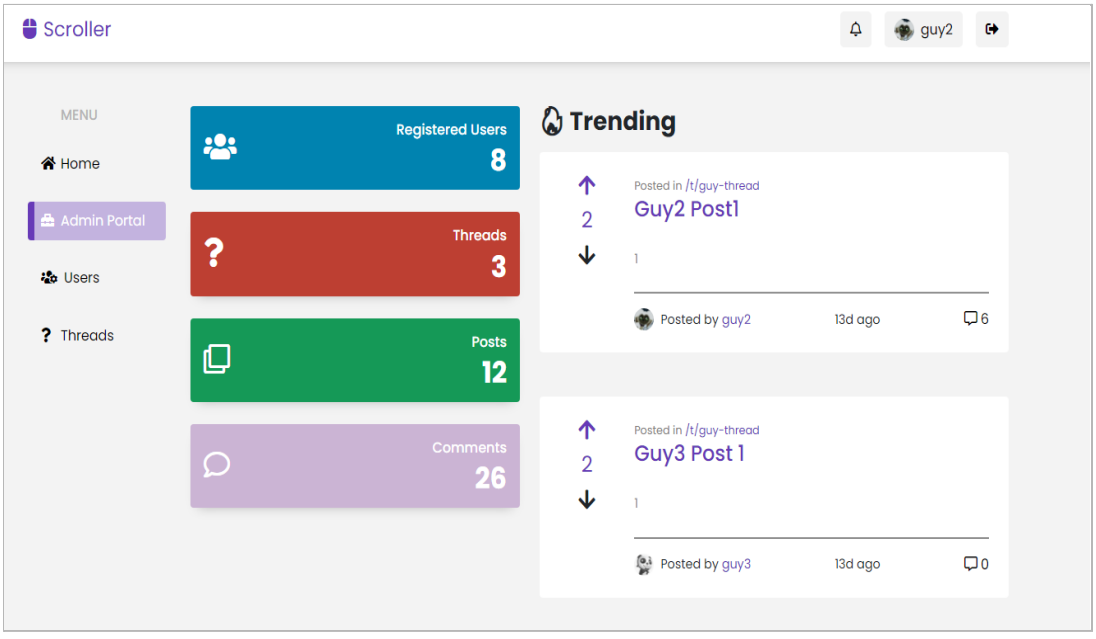
Email: admin@scroller.ca
Password: testtest@L

For User:

Email: scott.fazackerley@ubc.ca
Password: 100perOnProj@Def

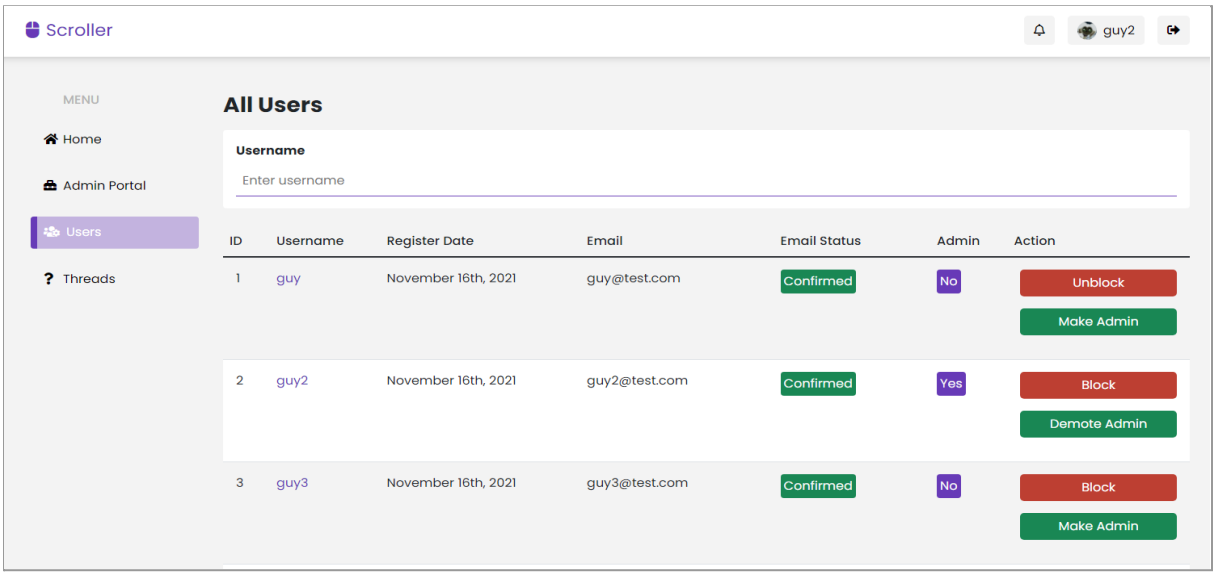
17. Admin Main Page

On this page, the admins can see several metrics and trending posts. Something to point out here is that admins have a new menu bar available to them, giving them the option to go to the home page or see all the users and threads with additional functionalities.



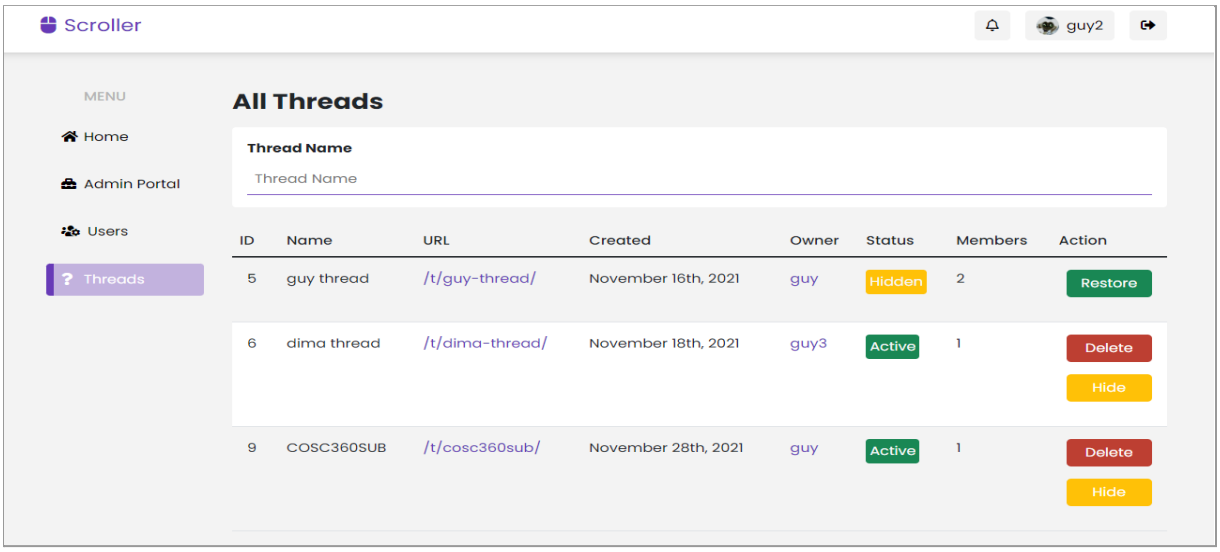
18. Admin Users Page

Here the admins can search for a particular user or just scroll through the user list. They can click on the username and go to their profile. They can also see if a user confirmed the status of their email or if they are an admin. Lastly, admins can make users admins, or they can block or unblock them.



19. Admin Threads Page

On this page, the admins can search for a particular thread name in the search bar. They can click on the thread URL to go to inspect that thread. They can see who created the thread. Additionally, they can see if the thread is active or hidden and the number of members that have joined the thread. Lastly, they have the option to hide, restore, or delete threads.



Implementation from a System and Developer's Perspective

Summary of Implemented Features

Non-User:

- Can view posts on the main page.
- Can search for posts, threads, and comments by clicking any left-hand side menu bar options and then using the search bar.
- Can create an account by clicking on the Sign-in button on the top-right of the navigation bar of the main page.
- Have access to the exact styling that a registered user would.
- Users can recover their passwords through email by clicking on the "Get Help Sign in" button from the login page.

User:

- Create threads with a thread background and profile pictures.
- Create posts on any thread with a body text, an image, or a YouTube URL.
- Can vote on comments and posts.
- Insert comments on any posts of any thread.
- Hide and delete their posts.
- Delete comments from their posts.
- Update their profile picture.
- Update their username, password, or email.
- Can search for posts, threads, and comments by clicking any left-hand side menu bar options and then using the search bar.
- Users can simply log in with an email and password.
- View every comment they have said.
- Maintain state between page navigations. The user maintains a state between page navigation as the session is stored, including keeping the user logged in.
- Receive various notifications:
    - **%username%** created a post in your thread **%thread%**.
    - **%username%** replied to your comment in thread **%thread%**.

    - **%username%** voted down your post in thread **%thread%**.
    - **%username%** voted up your post in thread **%thread%**.
    - **%username%** removed your post in thread **%thread%**.
    - **%username%** removed your thread **%thread%**.

Admin:

- Can search for a user by username from their extra tab.
- They can block, unblock users or make them admins.
- They can hide, restore, or delete threads.
- Delete any comment or post.

Description of PHP and JavaScript Files

PHP:

- **Account-settings:** part of View. Uses UserController class to display current information about the user. Uses Javascript to update the information about the current user dynamically.
- **Account:** part of View. Uses UserController class to display current information about the user and its threads.
- **Admin-threads:** part of View. Uses AdminController class to display all existing threads on the website. Uses Ajax to delete/hide/restore threads on the website dynamically. Only Administrators can view and do any actions on this page.
- **Admin-users:** part of View. Uses AdminController class to display all existing users on the website. Uses Ajax to dynamically delete/make admin/unblock users on the website. Only Administrators can view and do any actions on this page.
- **Admin:** part of View. Uses AdminController class to display all trending threads and statistics of the website. Only Administrators can view and do any actions on this page.
- **Error:** part of View. Works along with Router.class.php to show the error message if the page does not exist on the website.
- **Index:** part of View. The page handles all renderings of Views, titles, and additional security checks to verify that users, admins, guests have access to certain pages.

- **Login:** part of View. The page where users can log in to their account. This page is not accessible if the user is already authorized to the system. Users are not able to log in if the email is not confirmed or the account is disabled. Uses Ajax to dynamically verify user input with the server and set the current session with the proper user credentials.
- **Notifications:** part of View. Uses NotificationController class to display the current activity of other users towards the content that the current users created on the website. Content loads only for the current user.
- **Post-create:** part of View. Uses PostMiddleware and PostController classes to create a post for a respective thread. Uses Ajax and Javascript to communicate with databases and do security checks before saving information in the database/file storage systems(images).
- **Post:** part of View. Uses PostController class to load the information about **ONE** post(and respective information) in the specific thread. Uses Ajax and Javascript to validate and create comments in the particular post and thread. Page is only accessible for users who are logged in. The post is not displayed if the owner or administrator deletes the post. Users are not able to post comments or vote if the post is "hidden."
- **Register-confirm:** part of View. The page where the user enters a randomly generated code to confirm the registration of the account. This page is only accessible for users who have a unique link that contains an attribute in the URL with the name of "token." The token is randomly generated and stored in the database during registration. The length of the token varies to prevent brute-forcing techniques. This security code to enter on the page is sent to the user's email along with the secure link to confirm the registration of the account.
- **Register:** part of View. The page where the user registers the new account. This page is not accessible if the user is logged in. Uses Ajax and Javascript to dynamically update the page's content and do security checks with the database(e.g., check if the username or email exists in the database, username or email field is empty, the password does not match, the password doesn't match RegEx format). Once all the information is confirmed, the account is created in the database, and the link with the token and confirmation code is sent to the user's email to confirm the registration of the account.

- **Restore-confirm:** part of View. The page where the user enters new and confirmation passwords to restore access to the account. The page is not accessible if the URL attribute misses a randomly generated “token” or the token has expired. The token is generated randomly(different symbols and lengths) when the user confirms “recovery of the password” on the restore page of the website.
- **Restore:** part of View. The page where the users enter their email address to restore access to their account(e.g., users forgot their password or been hacked). If email is not found in the database or not confirmed, the user will not be able to recover their page. Otherwise, the secure link will be sent to the user’s verified email address to proceed to the next step of password recovery. Page is not accessible if the user is logged in.
- **Search:** part of View. This page is accessible for all authorized and non-authorized users to search for the information on the website: comments, posts, and threads). Before the search query is sent to the Model, the server sanitizes information for any potential attacks and executes the query. This page uses Ajax and javascript to dynamically validate and update the page’s content based on the search query and the parameters.
- **Thread-create:** part of View. Uses ThreadMiddleware and ThreadController classes to create a thread. Uses Ajax and Javascript to communicate with databases and do security checks before saving information in the database/file storage systems(images).
- **Thread:** part of View. This page is only accessible for authorized users. It displays information about the threads, respective posts, and comments. Users can join/leave the thread, create posts, and vote for the post, comment. This page also shows the top users of the thread based on the number of posts they created in this thread.
- **Main:** part of View. This page is accessible for all authorized and non-authorized users to show the trending posts and threads. This page is also used as the main page of the website.
- **DatabaseConnector.class:** part of Controller. Creates a connection to the database and returns the object of “Connection” so other controllers/models are able to retrieve access to the database. If the connection fails, the website will only show one page, which displays the error that “Website doesn’t work.” For security reasons, detailed information about the error is not displayed.

- **DotEnv.class:** part of Controller. The class that reads ".env" file to access the credentials to the database and pass the information to the DatabaseConnector class.
- **Router.class:** part of Controller. Routing all GET requests to the specific pages returns the title of the page and "path" of the file to be rendered.
- **AdminMiddleware.class:** part of Controller. Adds a similar validation as in the javascript file to confirm that information passed from the user is valid. Based on the request(POST/GET), information is sent to a specific method in the middleware. Once the validations are completed, the server calls the specific method in the AdminController Model to update/delete/create information in the database. Once the operation is completed, the server returns a JSON object with either information or error back to the Ajax method.
- **CommentMiddleware.class:** part of Controller. Adds a similar validation as in the javascript file to confirm that information passed from the user is valid. Based on the request(POST/GET), information is sent to a specific method in the middleware. Once the validations are completed, the server calls the specific method in the CommentController Model to update/delete/create information in the database. Once the operation is completed, the server returns a JSON object with either information or error back to the Ajax method.
- **ThreadMiddleware.class:** part of Controller. Adds a similar validation as in the javascript file to confirm that information passed from the user is valid. Based on the request(POST/GET), information is sent to a specific method in the middleware. Once the validations are completed, the server calls the specific method in the ThreadController Model to update/delete/create information in the database. Once the operation is completed, the server returns a JSON object with either information or error back to the Ajax method.
- **TokenMiddleware.class:** part of Controller. Adds a similar validation as in the javascript file to confirm that information passed from the user is valid. Based on the request(POST/GET), information is sent to a specific method in the middleware. Once the validations are completed, the server calls the specific method in the TokenController Model to update/delete/create information in the database. Once the operation is completed, the server

returns a JSON object with either information or error back to the Ajax method.

- **UserMiddleware.class:** part of Controller. Adds a similar validation as in the javascript file to confirm that information passed from the user is valid. Based on the request(POST/GET), information is sent to a specific method in the middleware. Once the validations are completed, the server calls the specific method in the UserController Model to update/delete/create information in the database. Once the operation is completed, the server returns a JSON object with either information or error back to the Ajax method.
- **Controller.class:** part of Controller. An abstract class that enforces Models to create specific methods get, post, update, delete to interact with database.
- **AdminController.class:** part of Model. This controller executes insertion/deletion based on the information passed from the AdminMiddleware. Also, it executes retrieval information based on the method call from the respective View in the admin pages.
- **CommentController.class:** part of Model. This controller executes insertion/deletion based on the information passed from the CommentMiddleware. Also, it executes retrieval information based on the method call from the respective Views in the thread/comment/post/search pages.
- **NotificationsController.class:** part of Model. This controller executes insertion/deletion based on the information passed from the various middlewares, based on the user's actions on the website. Also, it executes retrieval information based on the method call from the respective View on the notification page.
- **PostController.class:** part of Model. This controller executes insertion/deletion based on the information passed from the PostMiddleware. Also, it executes retrieval information based on the method call from the respective Views in the thread/comment/post/search pages.
- **ThreadController.class:** part of Model. This controller executes insertion/deletion based on the information passed from the ThreadMiddleware. Also, it executes retrieval information based on the method call from the respective Views in the thread/main/admin/search pages.

- **TokenController.class:** part of Model. This controller executes insertion/deletion based on the information passed from the UserController and TokenMiddleware. Also, it executes retrieval and validation of information based on the method call from the respective Views in the register-confirm, restore-confirm pages.
- **UserController.class:** part of Model. This controller executes insertion/deletion based on the information passed from all the middlewares that require verification of the current user(e.g., if the email is confirmed, the user is not blocked, the user is admin, etc.). Also, it executes retrieval information based on the method call from the respective Views in the account, account-settings pages.

Javascript:

- Script.js - handles the dynamic update of website pages using JQuery and Ajax.

How the Website Works at a High Level

Our codebase is divided into two parts: the client and the server.

Client

The purpose of the client part of the code is to make the website dynamic and responsive. We use AJAX to update the styles and information dynamically, so pages do not get refreshed when using a search bar or sorting threads. In addition, we use RESTful API calls to the server to retrieve the correct information.
All CSS styles are defined here as well, and every page of the website is created here.

Server

The server manages the connection to our database and allows the user to access and modify data easily. We created a DatabaseConnector class to call in our controllers to connect to the database consistently.

We created middleware classes to improve security. Any time an API call is made, it first goes through a PHP middleware class to determine if the information being

sent is valid. If the information is invalid, it will send an error. Otherwise, the middleware will send the information to its respective PHP controller.

In the PHP controllers, that is where all our querying is done. We have the following controllers:

1. Admin
2. Comment
3. Notification
4. Post
5. Thread
6. Token
7. User

Each controller is responsible for communicating with the necessary database models defined in our database folder.
Lastly, we have created a PHP router class to properly navigate between pages. The router is where we set the title of the webpage.

Website Limitations

- Not able to send an automatic email with the recovery/register link as we do not have an SMTP server/service. The administrator needs to provide the user with the link from the database manually.
- Users are unable to reply to another user's comment. They can only comment on a post.
- Users cannot edit their comments.