

Champlain College - Lennoxville

Final project: An open-source web application

PROGRAM:	420.B0 Computer Science Technology	
COURSE:	Transactional Web Applications 2	
COURSE CODE:	420-530-LE	
WEIGHT:	20% of the final score	
SEMESTER:	Fall 2023	
INSTRUCTOR:	Francis Gauthier	Office C-239
	fgauthier@crcmail.net	

Project overview

The goal of the project will be to use some code from an open-source web application. The students will install and document the application. After a review of the application, they will come up with a CI/CD pipeline with stages to build, prepare, enhance, and deploy the web application.

This project is meant as a complete review and application of the concepts seen in this course so far. It is also an opportunity to showcase one's ability to understand any web application based on our current knowledge.

Working in Teams

This project should be done in teams of two. If a student wants to work alone on the project, they can, but the scope will be only slightly reduced.

Each team must be approved by the teacher to avoid any problematic collaboration.

Team approbation and application validated by Nov. 27th 2023.

Part 1 - Locate, install, and document the application

Intended timeline: Nov. 20th - Nov. 30th

Locate

Your first locate an open-source web application publicly available online. Verify that the license allows public access.

You will be evaluated on:

- The complexity of the web stack (backend only, frontend only or full-stack, with DB or no DB)
- The complexity of the application chosen (number of features, technologies used)

**Think about part 2 of the project. A simpler application can result in losing points for this task but might make it easier to achieve the part 2 of the project. Choose one project within your reach.*

Install

Next, you will be to clone the application on your machine. Then, you must install and run the application locally.

To do so:

- Follow the documentation provided by the application owners (usually in Readme.md)
- Install the required dependencies
- Provide the environment variables needed (if any)
- Provide a database connection (if any)
- Run the application locally

You will be evaluated on:

- The ability to run the application locally

Document

Next, you should document that the application does.

Using a text editor or any other tools, write a summary of:

- What is the general purpose of the application?
- If the application has a back-end:
 - o The routes available on the API
 - o The purpose of these routes
- If the application has a front-end:
 - o The different views (take screenshots to document them)
 - o What are the actions possible?
 - o Is there authentication required?
- If the application has a database:
 - o The different collections/table names
 - o What are the models used in the database?

You will be evaluated on:

- The thoroughness of the documentation
- The exactitude of the documentation
- The format/organization of the documentation

Part 2 - CI/CD pipeline

Intended timeline: Nov. 30th - Dec. 11th

Each team will then have to build a CI/CD pipeline on GitLab.

To do so:

- Clone the application
- Create a new GitLab project
- Add the remote to your repository cloned (git remote add ...)
- Push on GitLab

Stages required

For a team of two:

- (Required) Install stage (setting up, installing node_modules, etc.)
- At least **two** of:
 - o A Lint stage
 - o A documentation stage (produces automatic documentation based on annotations)
 - o A test stage (unit tests)
 - o A test stage (integration tests)
- (Required) Build stage (preparing resources for the deployment stage)
- (Required) Deploy stage (deploying the web application on AWS)

For students working alone:

- (Required) Install stage (setting up, installing node_modules, etc.)
- At least **one** of:
 - o A Lint stage
 - o A documentation stage (produces automatic documentation based on annotations)
 - o A test stage (unit tests)

- A test stage (integration tests)
- (Required) Build stage (preparing resources for the deployment stage)
- (Required) Deploy stage (deploying the web application on AWS)

More details to come on Nov. 27th...

Part 3 - Presentations

Intended date: Dec. 11th

Each team will do a small presentation to present the result of the application that they worked on.

More details to come...

Part 1 – Documentation of the application

General purpose of the application

PostIt is a comprehensive social media web application built with the MERN stack that provides a platform for users to interact and share content. The main functionalities of PostIt include:

1. **Post Management:** Users can create, read, update, and delete posts. This allows users to share their thoughts, ideas, or any content they wish to disseminate to their network.
2. **Interaction with Posts:** Users can like and unlike posts, providing a way to express their reactions to the content shared by others. This feature enhances user engagement on the platform.
3. **Commenting:** The application supports nested comments, meaning users can reply to comments on a post, fostering deeper discussions and interactions.
4. **Markdown Support:** Posts and comments support Markdown, a lightweight markup language for creating formatted text. This allows users to add styling and formatting to their posts and comments, making the content more readable and engaging.
5. **User Authentication:** The application uses JWT for authentication, ensuring that only registered and authenticated users can perform certain actions like creating a post or commenting.
6. **Private Messaging:** Users can send private messages to other users in real-time using socket.io, facilitating direct and private communication between users.
7. **User Profiles:** Users can view profiles of other users and browse through their posts, liked posts, and comments. This feature allows users to know more about others and their activities on the platform.

8. Infinite Scrolling: This feature allows users to continuously scroll through content, improving the user experience by providing a seamless flow of content.

9. Search and Sorting: Users can search for posts by their title and sort posts by attributes such as like count, comment count, and date created, making it easier to find and organize content.

10. Profanity Filtering and Cooldowns: The application has measures in place to maintain a respectful and safe environment for users, such as profanity filtering and posting/commenting cooldowns.

11. Responsive Layout: The application layout is fully responsive, meaning it adjusts to different screen sizes for optimal viewing and interaction, enhancing the user experience across various devices.

Back End of the application

Routes

Posts

All API requests under posts: /api/posts

1.

Route: GET /api/posts/

Description: Retrieve a paginated list of posts.

Authentication: Optionally verifies the user's token.

2.

Route: POST /api/posts/

Description: Create a new post.

Authentication: Requires a valid user token.

3.

Route: GET /api/posts/:id

Description: Retrieve details of a specific post.

Authentication: Optionally verifies the user's token.

4.

Route: PATCH /api/posts/:id

Description: Update the content of a specific post.

Authentication: Requires a valid user token.

5.

Route: DELETE /api/posts/:id

Description: Delete a specific post.

Authentication: Requires a valid user token.

6.

Route: POST /api/posts/like/:id

Description: Like a specific post.

Authentication: Requires a valid user token.

7.

Route: DELETE /api/posts/like/:id

Description: Unlike a specific post.

Authentication: Requires a valid user token.

8.

Route: GET /api/posts/liked/:id

Description: Retrieve posts liked by a specific user.

Authentication: Optionally verifies the user's token.

9.

Route: GET /api/posts/like/:postId/users

Description: Retrieve users who liked a specific post.

Users

All API requests under users: `/api/users`

1.

Route: `POST /api/users/register`

Description: Register a new user.

Validation: Validates the input for username, email, and password.

2.

Route: `POST /api/users/login`

Description: Log in an existing user.

Validation: Validates the input for email and password.

3.

Route: `GET /api/users/random`

Description: Retrieve a list of random users.

Authentication: Requires a valid user token.

4.

Route: `GET /api/users/:username`

Description: Retrieve details about a specific user.

Authentication: Optionally requires a valid user token.

5.

Route: `PATCH /api/users/:id`

Description: Update user information.

Authentication: Requires a valid user token.

6.

Route: `POST /api/users/follow/:id`

Description: Follow a user.

Authentication: Requires a valid user token.

7.

Route: DELETE /api/users/unfollow/:id

Description: Unfollow a user.

Authentication: Requires a valid user token.

8.

Route: GET /api/users/followers/:id

Description: Retrieve followers of a user.

Authentication: Optionally requires a valid user token.

9.

Route: GET /api/users/following/:id

Description: Retrieve users that a user is following.

Authentication: Optionally requires a valid user token.

Comments

All API requests under comments: /api/comments

1.

Route: PATCH /api/comments/:id

Description: Update a comment.

Access: Private

2.

Route: POST /api/comments/:id

Description: Create a comment.

Access: Private

3.

Route: DELETE /api/comments/:id

Description: Delete a comment.

access: Private

4.

Route: GET /api/comments/post/:id

Description: Get comments for a specific post.

Access: Public

5.

Route: GET /api/comments/user/:id

Description: Get comments made by a specific user.

Access: Public

Messages

All API requests under messages: /api/messages

1.

Route: GET /api/messages/

Description: Retrieve a list of conversations for the authenticated user.

Authentication: Requires a valid user token.

2.

Route: POST /api/messages/:id

Description: Send a message to a specific user/conversation.

Authentication: Requires a valid user token.

3.

Route: GET /api/messages/:id

Description: Retrieve messages for a specific conversation.

Authentication: Requires a valid user token.

Database

Here is how it looks.

Test

Comments

Conversations

Follows

Messages

Postlikes

Posts

Users

Here are the models for the database

CommentSchema

Field	Type	Ref	Required	Default	Description
commenter	ObjectId	user	true	-	ID of the user who made the comment.
post	ObjectId	post	true	-	ID of the post the comment belongs to.
content	String	-	true	-	Content of the comment.
parent	ObjectId	comment	-	-	ID of the parent comment (if it exists).
children	Array of ObjectId (comment)	comment	-	-	IDs of child comments.
edited	Boolean	-	false	false	Indicates whether the comment was edited.
timestamps	Date	-	-	-	Auto-generated timestamps for creation and last update.

ConversationSchema

Field	Type	Ref	Description
recipients	Array of ObjectId (user)	user	IDs of users participating in the conversation.
lastMessageAt	Date	-	Timestamp indicating the last message in the conversation.
timestamps	Date	-	Auto-generated timestamps for creation and last update.

FollowSchema

Field	Type	Ref	Description
userId	ObjectId (user)	user	ID of the user who is following.
followingId	ObjectId (user)	user	ID of the user being followed.
timestamps	Date	-	Auto-generated timestamps for creation and last update.

MessageSchema

Field	Type	Ref	Description
conversation	ObjectId (conversation)	conversation	ID of the conversation associated with the message.
sender	ObjectId (user)	user	ID of the user who sent the message.
content	String	-	Content of the message.
timestamps	Date	-	Auto-generated timestamps for creation and last update.

PostSchema

Field	Type	Ref	Description
poster	ObjectId (user)	user	ID of the user who posted the content.
title	String	-	Title of the post.
content	String	-	Content of the post.
likeCount	Number	-	Number of likes the post has received.
commentCount	Number	-	Number of comments the post has received.
edited	Boolean	-	Indicates whether the post has been edited.
timestamps	Date	-	Auto-generated timestamps for creation and last update.

PostLikeSchema

Field	Type	Ref	Description
postId	ObjectId (post)	post	ID of the post being liked.
userId	ObjectId (user)	user	ID of the user who liked the post.
timestamps	Date	-	Auto-generated timestamps for creation and last update.

UserSchema

Field	Type	Description
username	String	Unique username with length between 6 and 30 characters, no spaces allowed.
email	String	Unique email address, validated using the isEmail function.
password	String	Password with a minimum length of 8 characters.
biography	String	User's biography with a maximum length of 250 characters. Default is an empty string.
isAdmin	Boolean	Indicates whether the user is an administrator. Default is false.
timestamps	Date	Auto-generated timestamps for creation and last update.

Front End of the application

Login view.

PostIt

Login

Don't have an account yet? [Sign Up](#)

LOGIN

Copyright © 2022 [PostIt](#)

View Name:

LoginView

Purpose:

The LoginView component is the view that users see when they want to log into the platform. It consists of a form where users can enter their email and password.

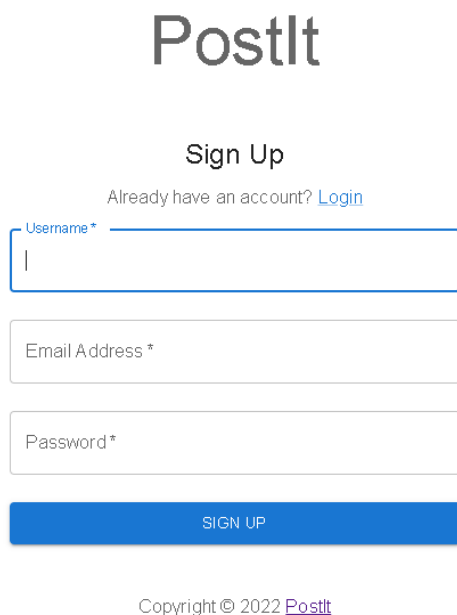
Possible Actions:

- Users can enter their email and password to log in.
- Users can navigate to the home page by clicking on the "PostIt" link.
- Users can navigate to the signup page by clicking on the "Sign Up" link.

Authentication:

This view is used for authentication. Users enter their email and password to log in to the application. For all the other views the authentication is managed from here on. The users have access to the main PostIt page, can look at other users' profiles and can see the current posts but cannot comment, like posts or message other users unless they are logged in.

Sign up view.



The image shows a web form for signing up on a platform called "PostIt". The form is centered on the page. At the top, the word "PostIt" is displayed in a large, dark grey font. Below it, the text "Sign Up" is centered. Underneath "Sign Up", there is a link that says "Already have an account? [Login](#)". The form itself consists of three input fields stacked vertically: "Username*", "Email Address*", and "Password*". Each field has a light blue border and a small blue asterisk indicating it is required. Below the input fields is a blue button with the text "SIGN UP" in white, uppercase letters. At the bottom of the form, there is a copyright notice: "Copyright © 2022 [PostIt](#)".

View Name:

SignupView

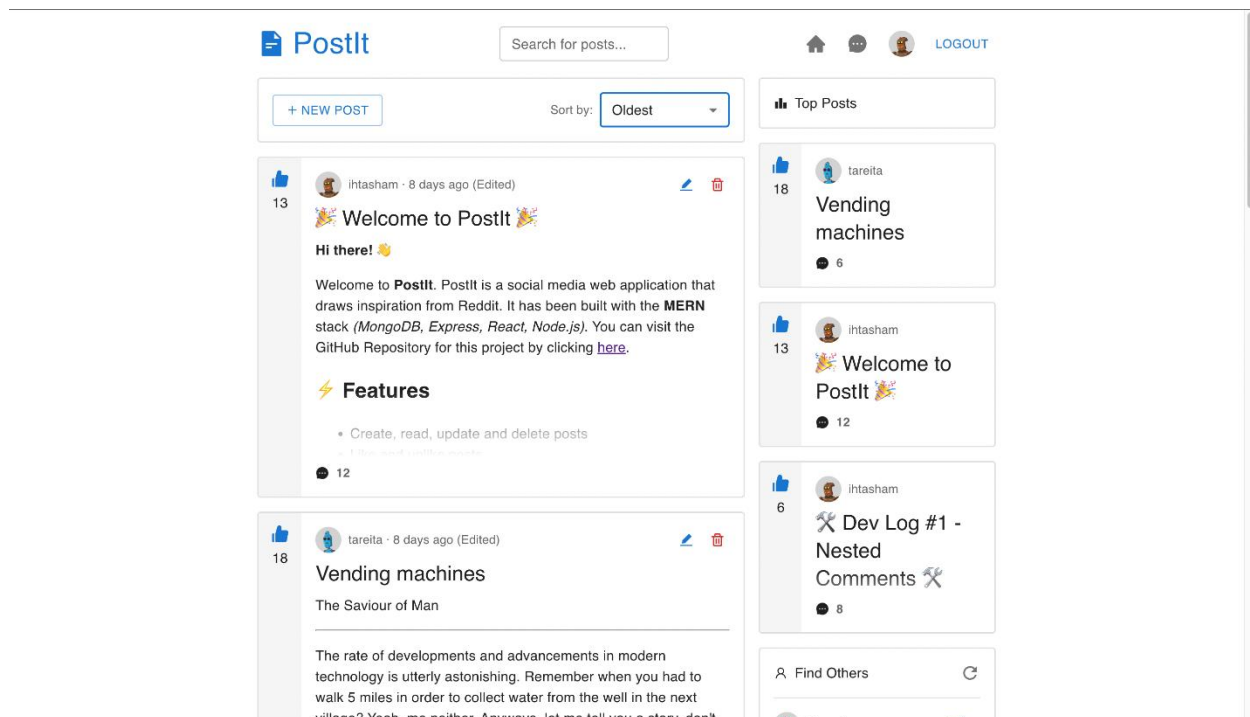
Purpose:

The SignupView component is the view that users see when they want to sign up for the platform. It consists of a form where users can enter their username, email, and password.

Possible Actions:

- Users can enter their username, email, and password to sign up.
- Users can navigate to the home page by clicking on the "PostIt" link.
- Users can navigate to the login page by clicking on the "Log In" link.

Explore view.



View Name:

ExploreView

Purpose:

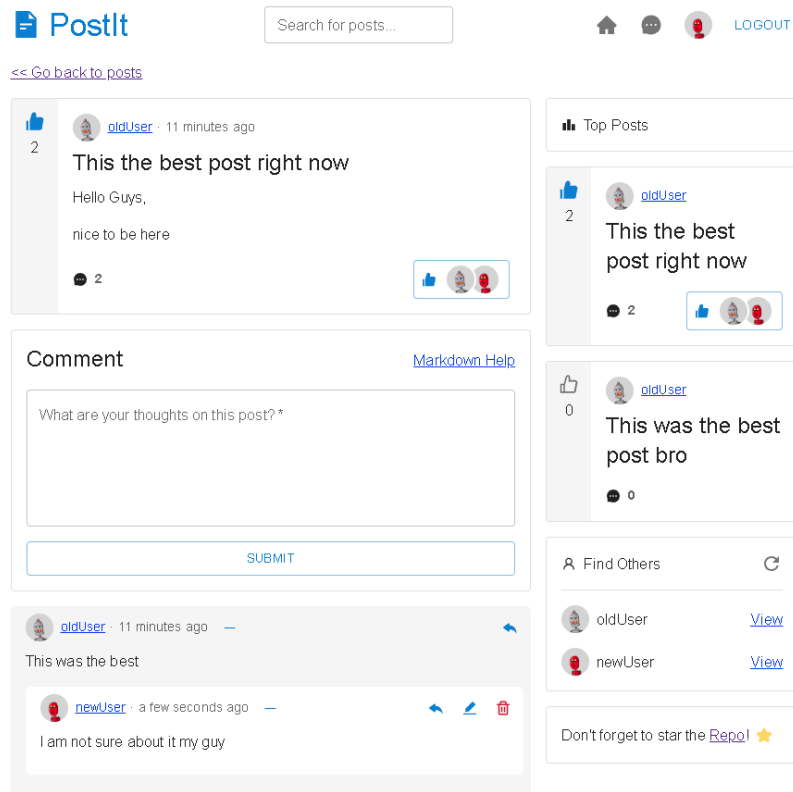
The ExploreView component serves as the primary interface for users to explore posts on the platform. It comprises a navigation bar, a post browser, and a sidebar. The navigation bar provides functionalities, a search bar to look for posts, navigation to the MessengerView, the user's ProfileView, and a logout link. This main component presents an infinite scroll view of all

current posts. The sidebar displays top posts, provides a user search feature, and includes a reminder to star the GitHub repository. Additionally, there is a button for creating new posts, which redirects the user to the CreatePostView. A dropdown menu is also available, allowing users to sort posts by date, number of likes, or number of comments. The NavBar and SideBar are integral to all other views, excluding the MessengerView, and hence, will not be reiterated in subsequent explanations.

Possible Actions:

- Users can browse through posts with an infinite scroll feature.
- Users can search for posts and users using the search bar in the NavBar.
- Users can navigate to the home page, MessengerView, ProfileView, or perform logout using the links in the NavBar.
- Users can view top posts, find users, and star the GitHub repository via the Sidebar.
- Users can create new posts by clicking on the new post button.
- Users can sort posts by date, number of likes, or number of comments using the dropdown menu.

Post view.



View Name:

PostView

Purpose:

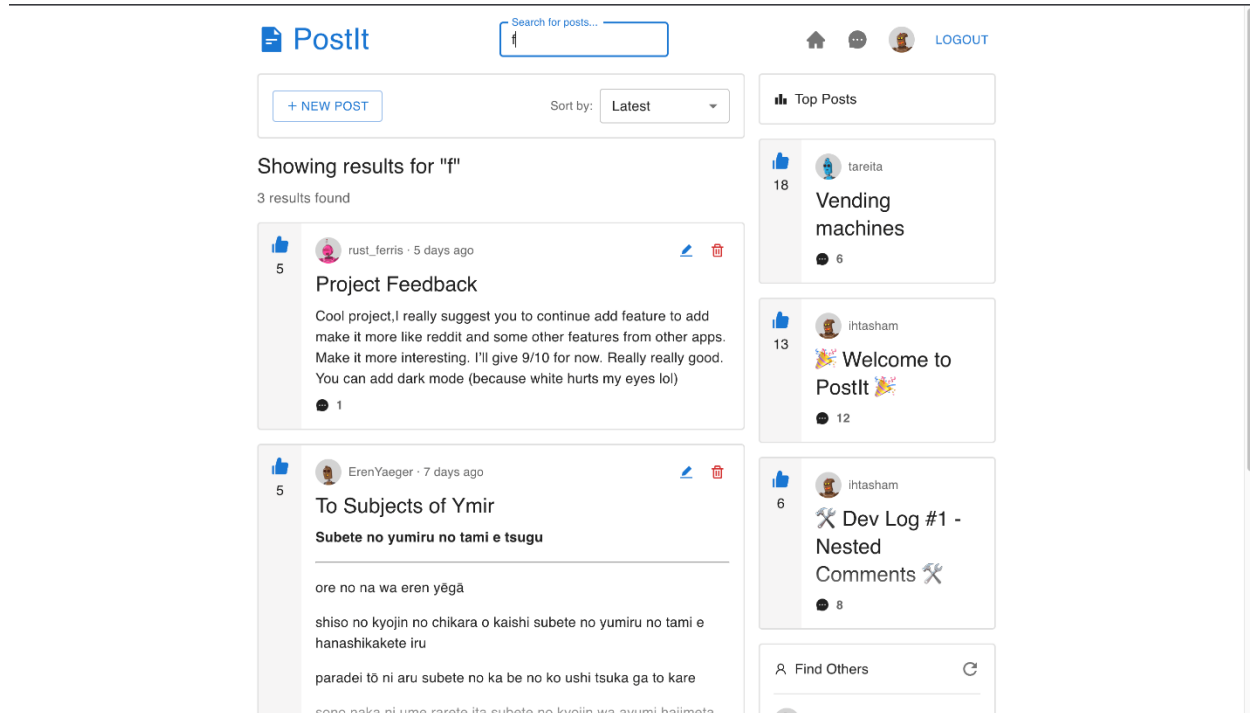
The `PostView` component is the view that users see when they want to view a specific post in detail. It includes a go back button, a link to the user who created the post, a "Like" button, the post content, a section for users to add comments to the post, and a display of comments that were previously added. The comment section also supports nesting of comments, enabling the creation of discussion threads where users can interact with each other.

Possible Actions:

- Users can view the post content.
- Users can like the post.

- Users can navigate back from the current view using the `GoBack` button.
- Users can navigate to the profile of the user who created the post.
- Users can add comments to the post.
- Users can view comments that were previously added to the post.
- Users can participate in nested comment discussions.

Search view.



View Name:

SearchView

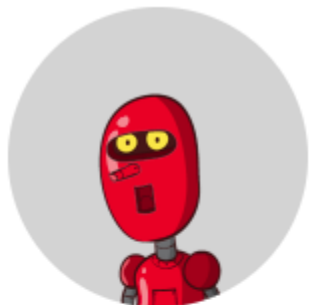
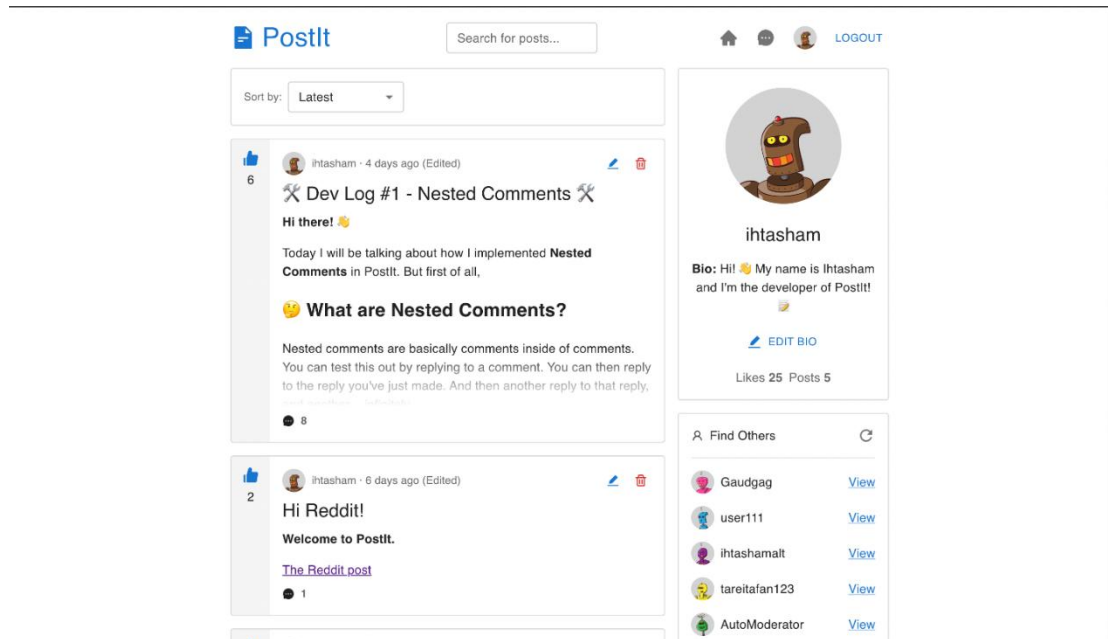
Purpose:

The `SearchView` component is the view that users see when they want to search for posts. When writing a search parameter in the search bar and pressing on enter all the posts that contain that value appear in the main component. There is still the option to sort the posts by date, amount of likes and comments.

Possible Actions:

- Users can enter a search parameter in the search bar in the Navbar.
- Users can view the search results in the main component.
- Users can sort the search results by date, amount of likes and comments.

Profile view.



newUser

Bio Information|

UPDATE

 CANCEL

Likes 0 Posts 0

View Name:

ProfileView

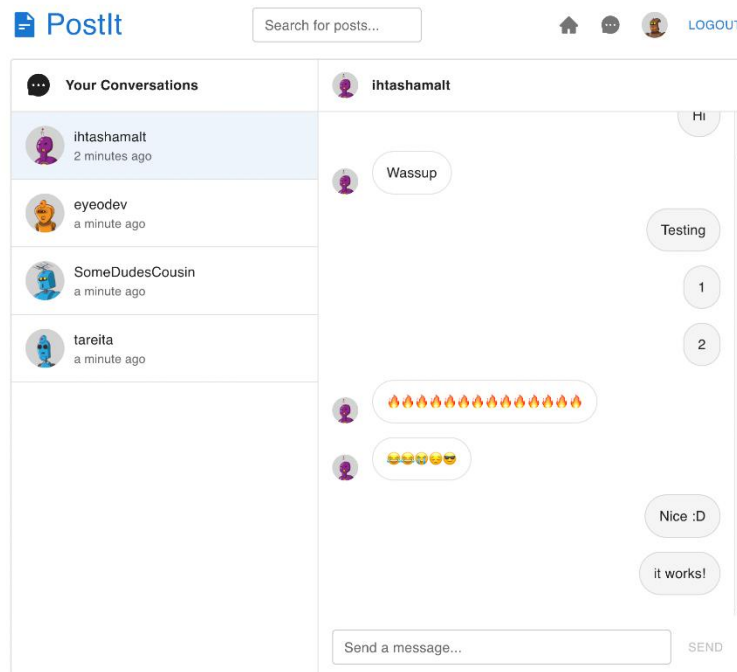
Purpose:

The `ProfileView` component is the view that users see when they want to view a specific user's profile. It also includes all the posts that the user in question created, the amount of likes and the amount of posts. Additionally, when on their own profile page, users can update their bio by clicking on the Edit Bio button and save the changes by clicking Update.

Possible Actions:

- Users can view the user's profile information.
- Users can browse through the user's posts and comments.
- Users can update their bio when on their own profile page.

Real-time private messenger.



View Name:

MessengerView

Purpose:

The `MessengerView` component is the view that users see when they want to use the messaging feature. It includes a list of users with whom the user has had conversations, a view that displays the message history, a text entry to enter text, and a send button.

Possible Actions:

- Users can select one of the users with whom they had a conversation and display a history of the messages sent.
- Users can send new messages in a selected conversation by writing a message in the text-entry part of the view and clicking on the Send Button.

Part 2 - CI/CD pipeline

For a team of two:

- (Required) Install stage (setting up, installing node_modules, etc.)
- At least **two** of:
 - o A Lint stage
 - o A documentation stage (produces automatic documentation based on annotations)
 - o ~~A test stage (unit tests)~~
 - o ~~A test stage (integration tests)~~
- (Required) Build stage (preparing resources for the deployment stage)
- (Required) Deploy stage (deploying the web application on AWS)