Course Name : CMPE-272 Sec 48 - Enterprise Software Platforms

Team Name : Algo Architects

Team Members :
Dhruv Khut
Pratik Kamanahalli Mallikarjuna
Linh Tran
Kanwaljeet Kaur

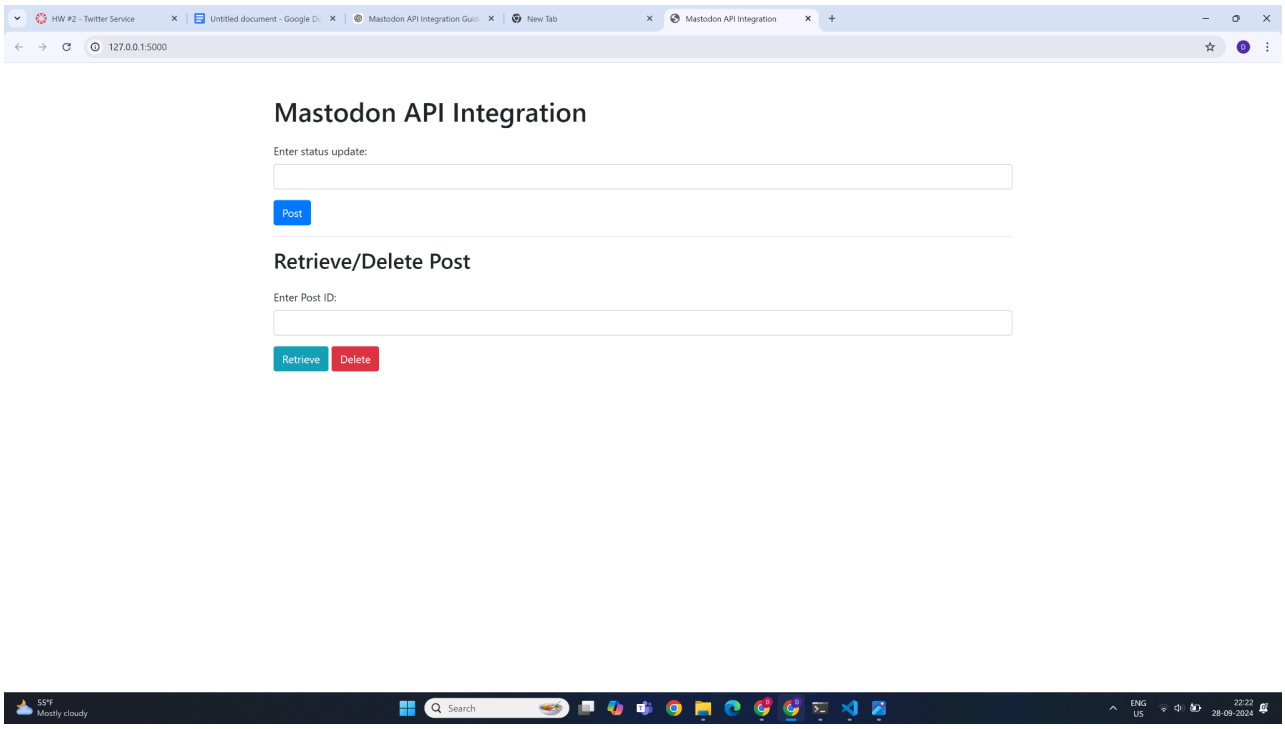
Assignment Name : Twitter Service

Objective: This project demonstrates the integration of the Mastodon API through a simple service and web UI that enables creating, retrieving, and deleting posts programmatically. It includes proper error handling, testing, and a user-friendly interface.

UI Interaction :

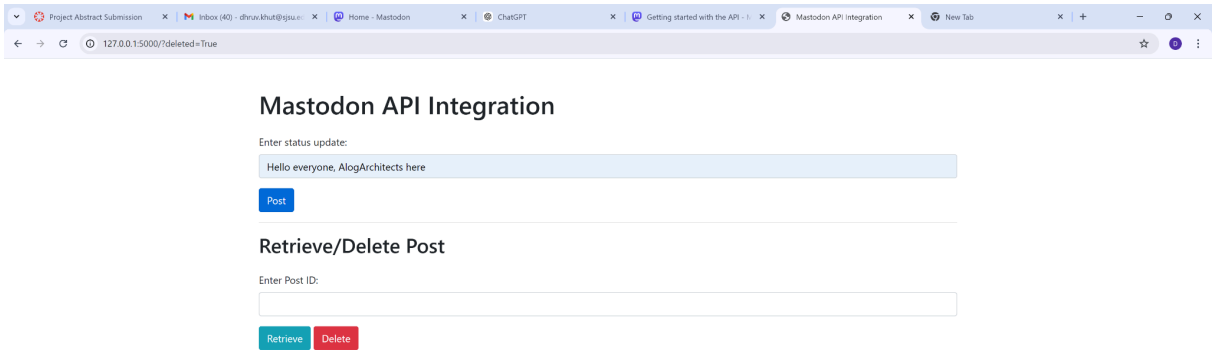
The web UI is designed to allow users to interact with the Mastodon API easily. Below are the functionalities of using the UI :

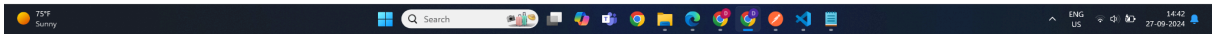
Home Page :

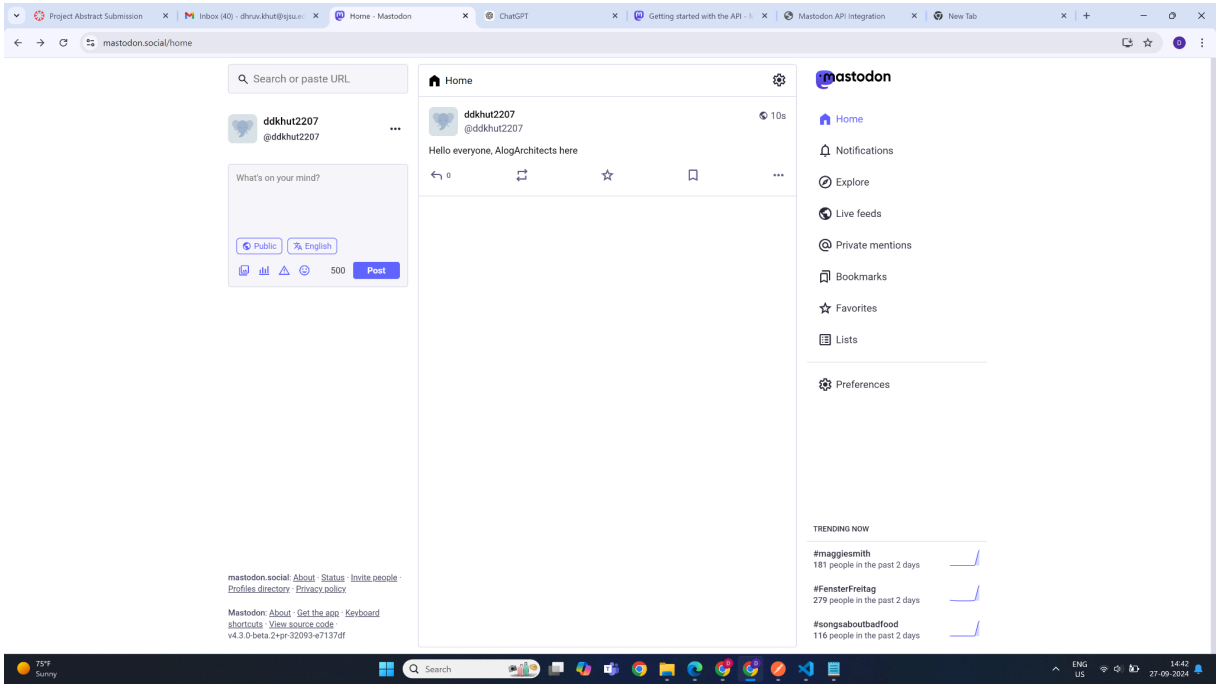


Create a Post:

In the text box labeled "Enter status update," input the content of your post.
Click the "Post" button to submit your update to Mastodon. The post will be displayed below once it is successfully created.

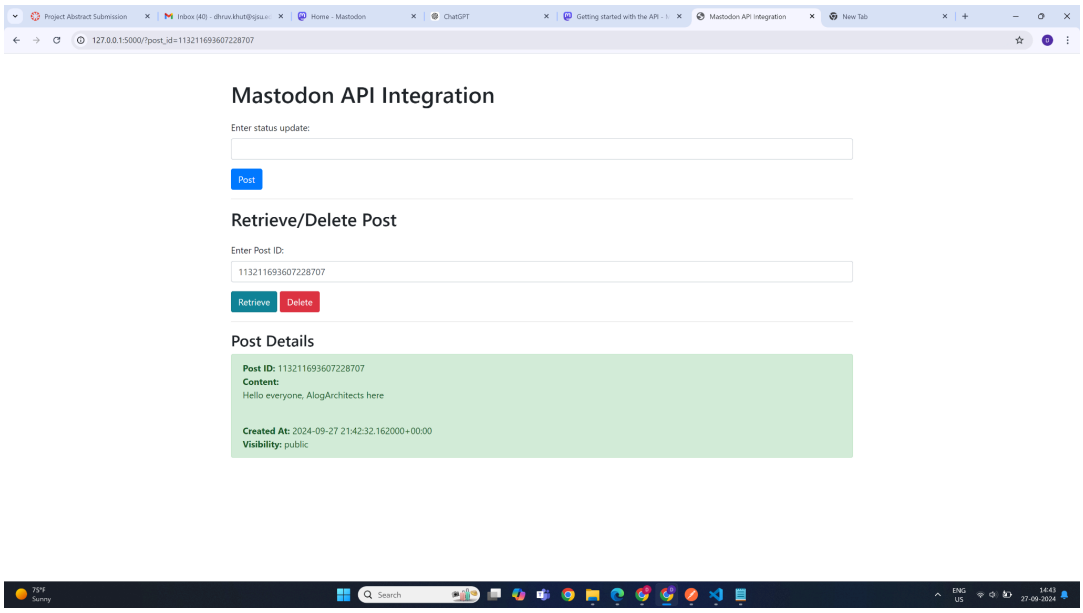






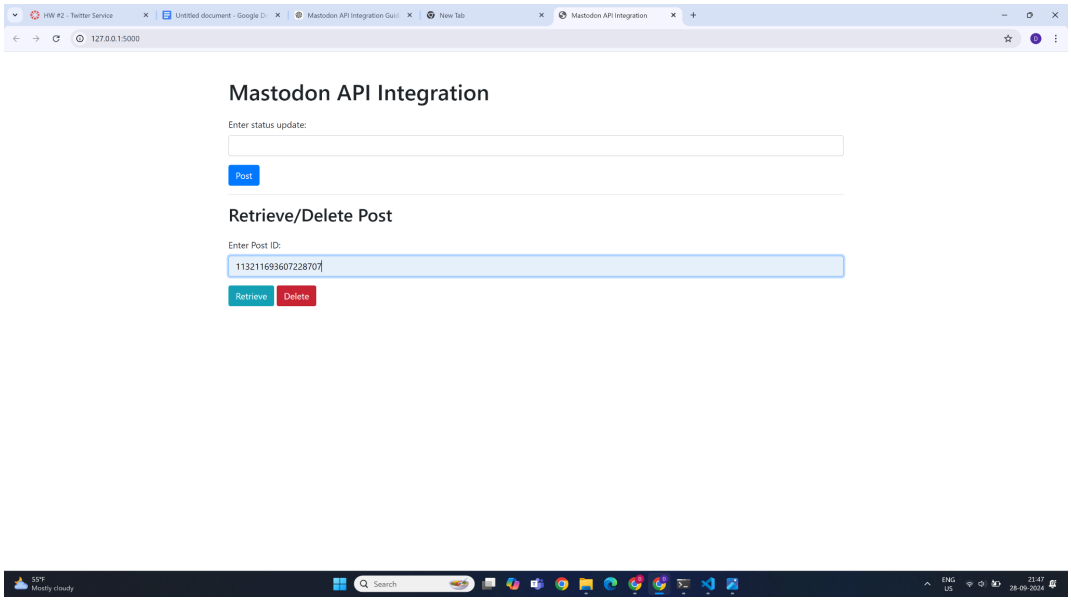
Retrieve a Post:

Click the "Retrieve" button to fetch and display the latest post you created.
The post content and details (e.g., timestamp, ID) will be shown in the display area.

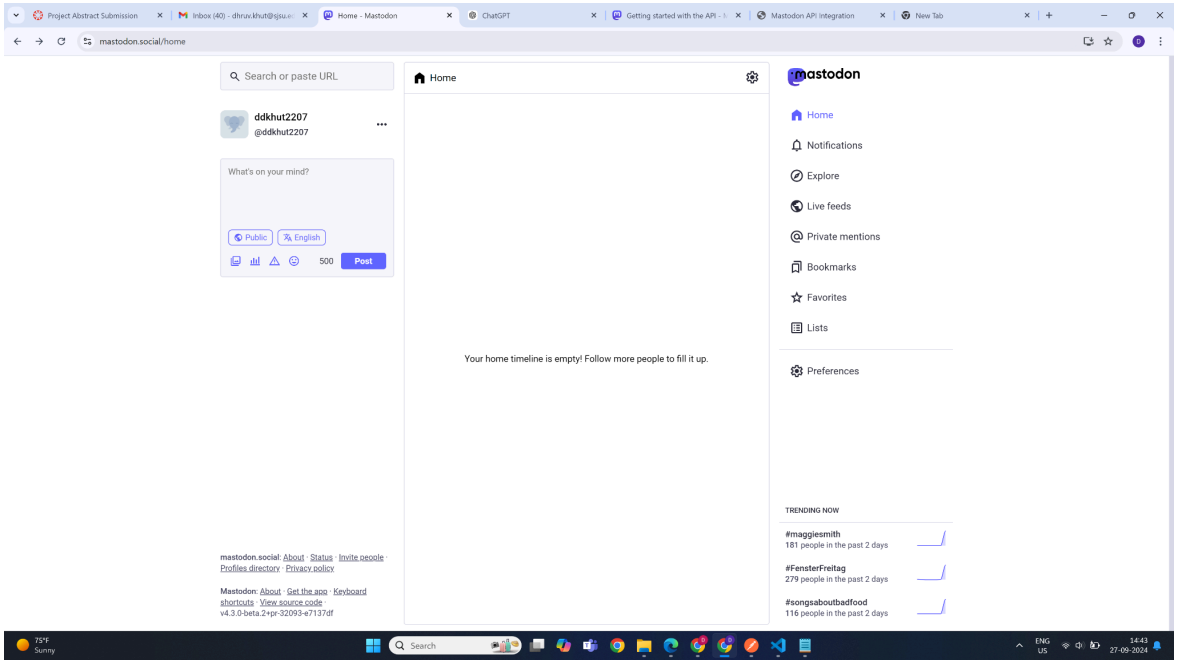


Delete a Post:

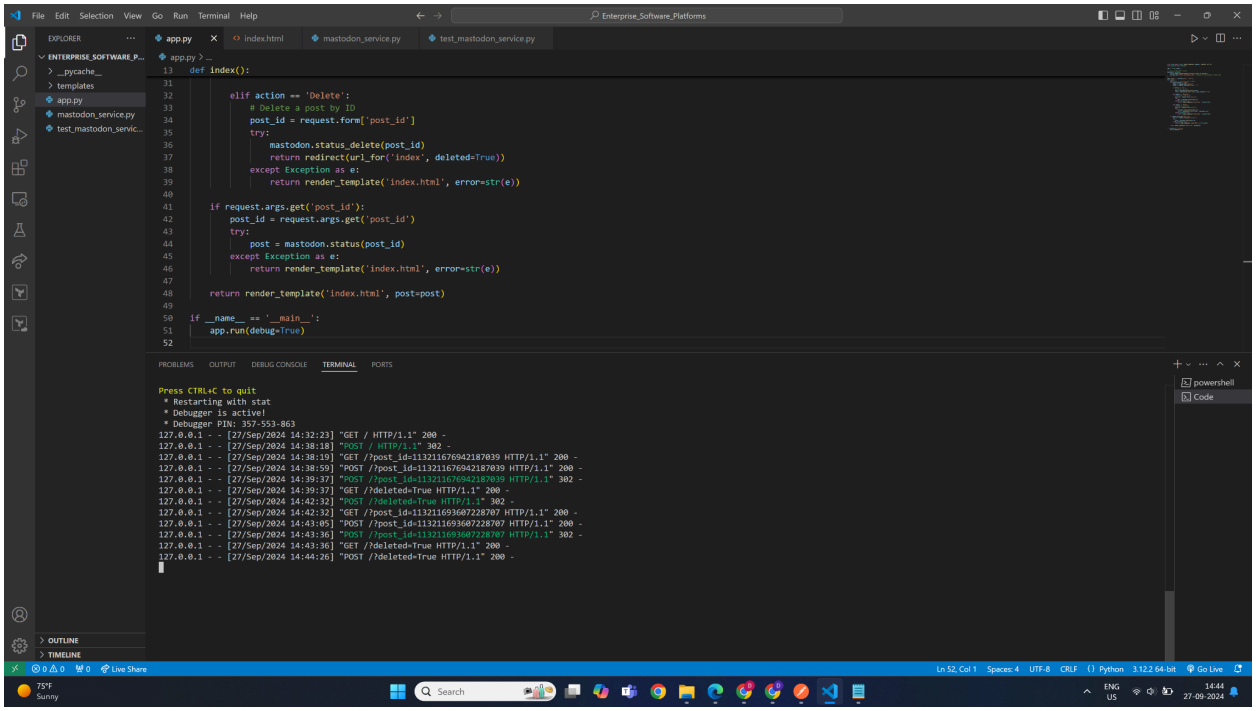
Once the post is created, you can enter the post ID and then click on the "Delete" button to remove the post from Mastodon. After deletion, the UI will update to confirm the post has been successfully deleted.



The post has been deleted from Mastodon :



And this how it looks on the backend and all the functions like create, retrieve and delete are updated on console :



Source Code :

The project comprises two main components: the Mastodon Service and the Web UI. Below I have given a breakdown of the key sections of the source code.

1. Mastodon Service Code (“mastodon_service.py”) :

This file contains all the backend logic for interacting with the Mastodon API. It provides functions for creating, retrieving, and deleting posts on Mastodon. The primary methods are:

- create_post:
  - Description : This function sends a POST request to the Mastodon API to create a new status update.
  - Implementation : The post content is passed as an argument to this function, which uses the `mastodon.status_post()` method to create the post. If successful, it returns the post data, including its ID and URL.
  - Error Handling : If the API call fails due to issues like rate limiting or invalid content, it raises appropriate errors and logs them for debugging.

- retrieve_post:
  - Description : This function retrieves a post using its unique ID.
  - Implementation : It sends a GET request to Mastodon using `mastodon.status()`, which fetches the post data by ID and returns it to the user.
  - Error Handling : The function checks if the post exists and manages errors if the post is not found or if the API returns other exceptions.

- delete_post:
  - Description: This function deletes a specific post using the Mastodon API.
  - Implementation : It sends a DELETE request using `mastodon.status_delete()`, which removes the post by ID.
  - Error Handling : If the post ID is invalid or the post has already been deleted, the function catches the error and raises a user-friendly message.

- Error Handling :
  - Each function is wrapped with proper error handling to deal with API errors like rate limits, invalid inputs, or connection issues. The errors are logged for debugging, and the user is notified of the failure in a clear, user-friendly way.

2. Web UI Code (“app.py”) :

The web interface is built using the Flask framework. It communicates with the Mastodon Service to provide the user with options to create, retrieve, and delete posts. Key components include:

- HTML Form for Input :
  - Description : The form on the web page allows users to input the content for their post. It includes buttons to post, retrieve, and delete posts.
  - Implementation : This is rendered using HTML, with input fields for status updates and buttons that trigger the corresponding Mastodon API functions.

- Flask Routes :
  - `/post` : This route handles the "Post" button. When clicked, it sends the content from the input box to the `create_post` function and displays the created post on the page.
  - `/retrieve` : This route fetches the most recent post using the `retrieve_post` function and displays it.
  - `/delete` : This route deletes the created post using the `delete_post` function.
  - Error Handling : Each route includes basic error handling to manage common issues like empty input or network errors.

3. Unit Tests (“test_mastodon_service.py”) :

This file includes tests to ensure that the Mastodon API service works correctly. These tests mock API calls to avoid making real network requests. Key tests include:

- Test Create Post :
  - This test simulates the creation of a post by mocking the API call and asserting that the post data returned is as expected.

- Test Retrieve Post :
  - This test mocks the retrieval of a post using a post ID and ensures that the retrieved post matches the expected data.

- Test Delete Post :
  - This test mocks the deletion of a post and asserts that the deletion is successful and handled correctly, including proper error management when the post is not found.

4. HTML File (app.html) :

The app.html file serves as the user interface for interacting with the Mastodon service. It provides a simple and intuitive layout for API operations:

- Text Box: This input field allows users to type in a status update that will be posted to Mastodon.

- Buttons:
  Post: Sends the entered status to the Mastodon API to create a new post.
  Retrieve: Fetches the most recent post and displays it.

Delete: Deletes the displayed post from Mastodon.
Display Section: A designated area where the posted or retrieved status is shown for users to view.

- The layout is user-friendly, ensuring a smooth experience for interacting with the Mastodon API while demonstrating the core functionalities of the service.

Unit Tests :

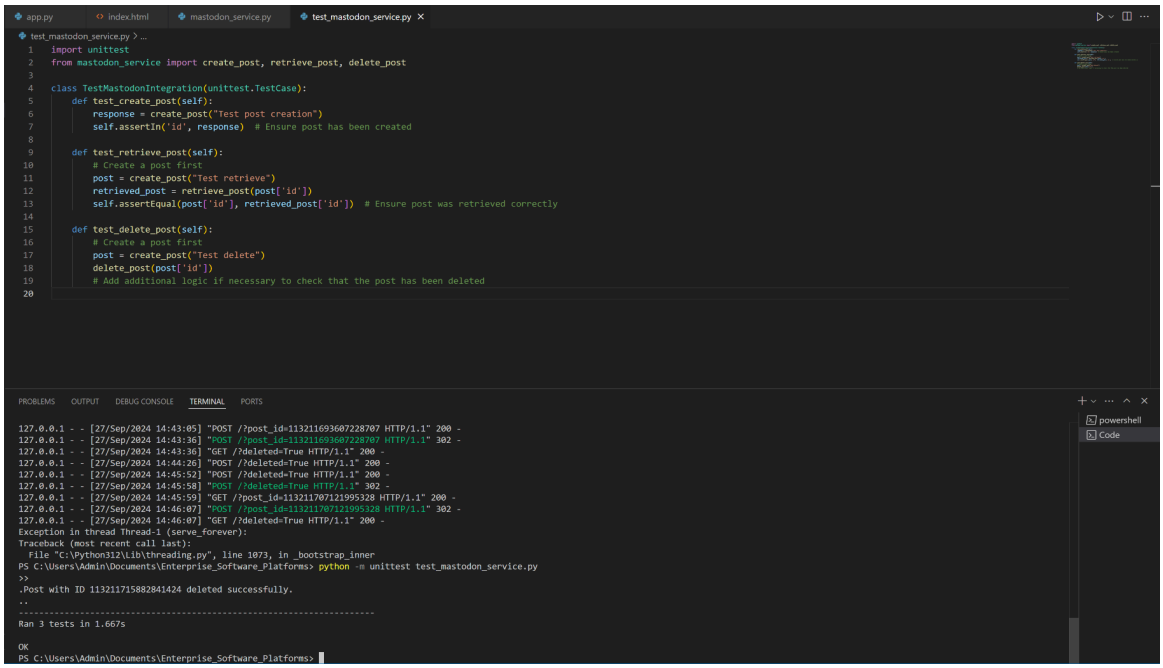
The project includes unit tests to ensure the functionality of each API interaction:

- Create Post Test:
  Verifies the creation of a post using mocked API requests.

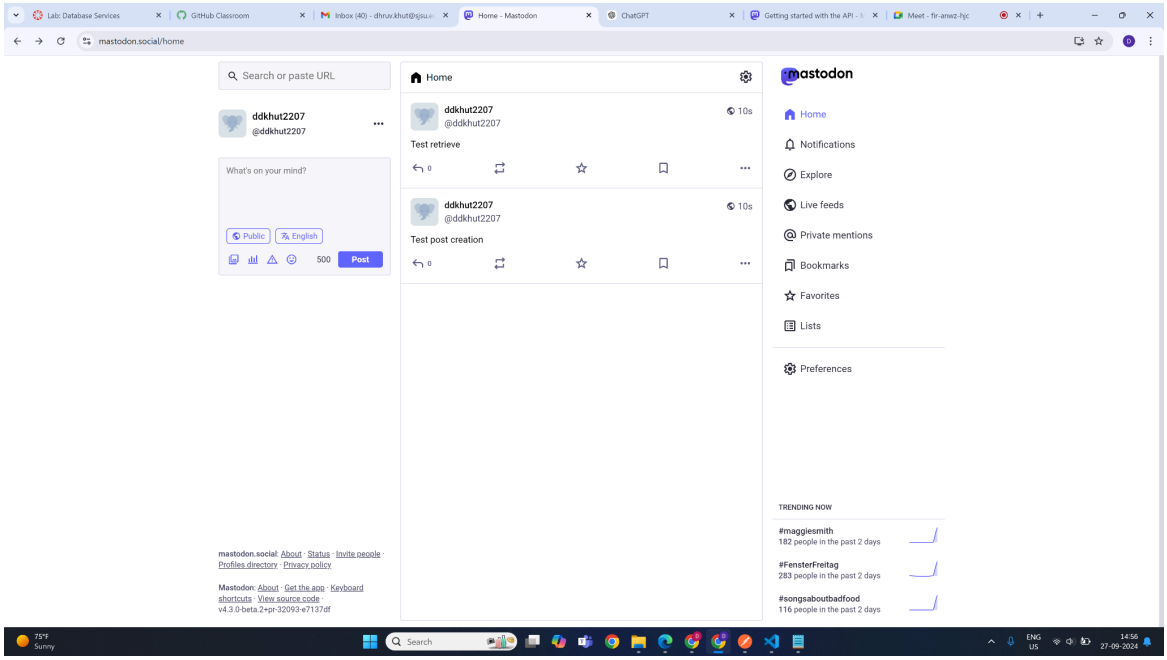
- Retrieve Post Test:
  Ensures the post retrieval works correctly, even without actual network calls.

- Delete Post Test:
  Confirms that the delete functionality works by mocking the deletion request.

Output on the console showcasing that the 3 test cases have been run properly which includes create, retrieve and delete posts :



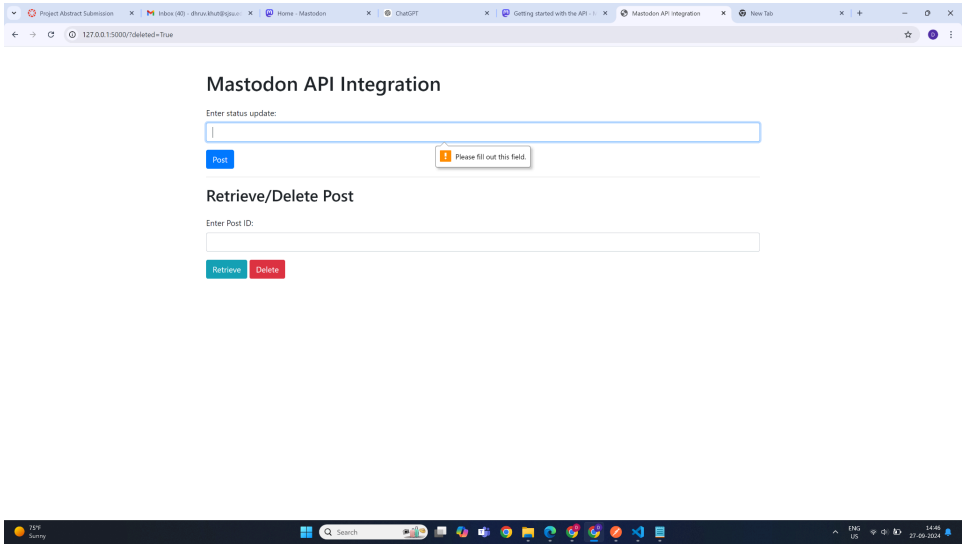
Output on the Mastodon :



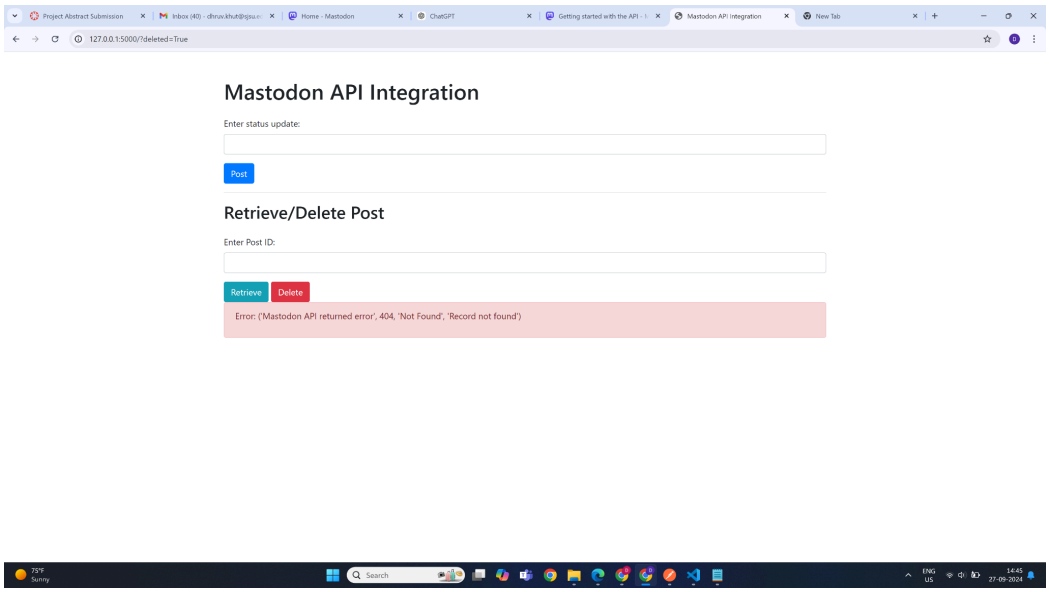
Error Handling :

I have used the library “MastodonRatelimitError’ to handle API rate limits and it is also mentioned in the code. And to handle invalid inputs, it will update on the website that input that user put is invalid and would return an error on the page or direct the user to fill the particular field.

In this instance, I attempted to create an empty post by submitting a blank input. The website promptly displayed an error message, instructing me to fill out the text field before proceeding:



In this instance, I entered an incorrect post ID while attempting to delete a tweet. The website displayed an error message, indicating that the tweet with the given ID does not exist :



Github Repository : https://github.com/dk22072002/AlgoArchitects_Twitter_Service
Commit Hash : b0b0cabb548b1a7772bb3040e65769b44d82be9f