**ICS 2309:**

**Commercial Programming**

B.Sc. Computer Science

3.2 (January – April/2025)

**Background information**:

We’re working on TESTING a model/system Feeler, a sentiment analysis application.

April, 2025

Software TESTING:

**Authors: Feeler Team, Version: 1.0**

****

Part 1: Overview

1.1 About:

This is what **Feeler** does – it is a sentiment analysis application designed to allow users to analyze the sentiment of text inputs - short texts that take the shape of tweets, in real-time. It provides insights into whether a given text is **positive** or **negative**, and exploring sentiment trends. The app uses **machine learning** models for sentiment analysis.

We will test the system against the requirements laid out earlier on in prior documents.

**DISCLAIMER**: the UI images embedded in this document are from the developed application’s interface and might be slightly different from what is present in prior documents, which were essentially place-holders (e.g logo). The rest of the images are predominantly snapshots of **tests done in command-line, using CURL or a frontend/backend specific testing library done in command-line.**

1.2 Core Features:

The main goal of this write-up is to assess whether or not we have properly implemented these functionalities listed out below:

1. **Authentication**:
   * user registration and login with JWT-based authentication.
   * secure logout functionality to invalidate tokens.
2. **Sentiment Analysis**:
   * analyze text for positive or negative sentiment.
   * display confidence scores.
3. **Mood History**:
   * store and display sentiment analysis history for the logged-in user.
   * Twitter-like user interface for history visualization.
4. **Mood Visualizations**:
   * line charts showing trends of sentiment over time.

Part 2: Technology used and API Endpoints

2.1 Built With:

Our sentiment analysis application depends/runs on the following technologies:

* **Backend**: Django REST Framework with API endpoints for sentiment analysis, user authentication, and data retrieval.
* **Frontend**: React.js application with components for each feature, and global styles applied through styles.css.
* **Database**: PostgreSQL schema to store user details and sentiment history.
* **ML Model**: A trained TensorFlow sentiment analysis model integrated into the backend.

2.2 API Endpoints

The backend is based on the working of a Django RESTful application-programming interface. We have brief descriptions of the API resources below that serve out HTTP requests from the frontend:

|  |  |  |
| --- | --- | --- |
| **ENDPOINT** | **METHOD** | **DESCRIPTION** |
| /api/register/ | POST | user registration. A user signs up to use the system |
| /api/token/ | POST | obtain access and refresh tokens(authentication) for logging into Feeler |
| /api/token/refresh/ | POST | refresh access token to get new ones |
| /api/logout/ | POST | invalidate refresh token |
| /api/predict/ | POST | analyze text sentiment |
| /api/history/ | GET | fetch user's sentiment history |
| /api/visualization/ | GET | fetch data for sentiment graphs |

We developed and implemented a system that works to satisfy the features we listed out earlier and with the information up to this point, we hope we have given enough context for what we are testing.

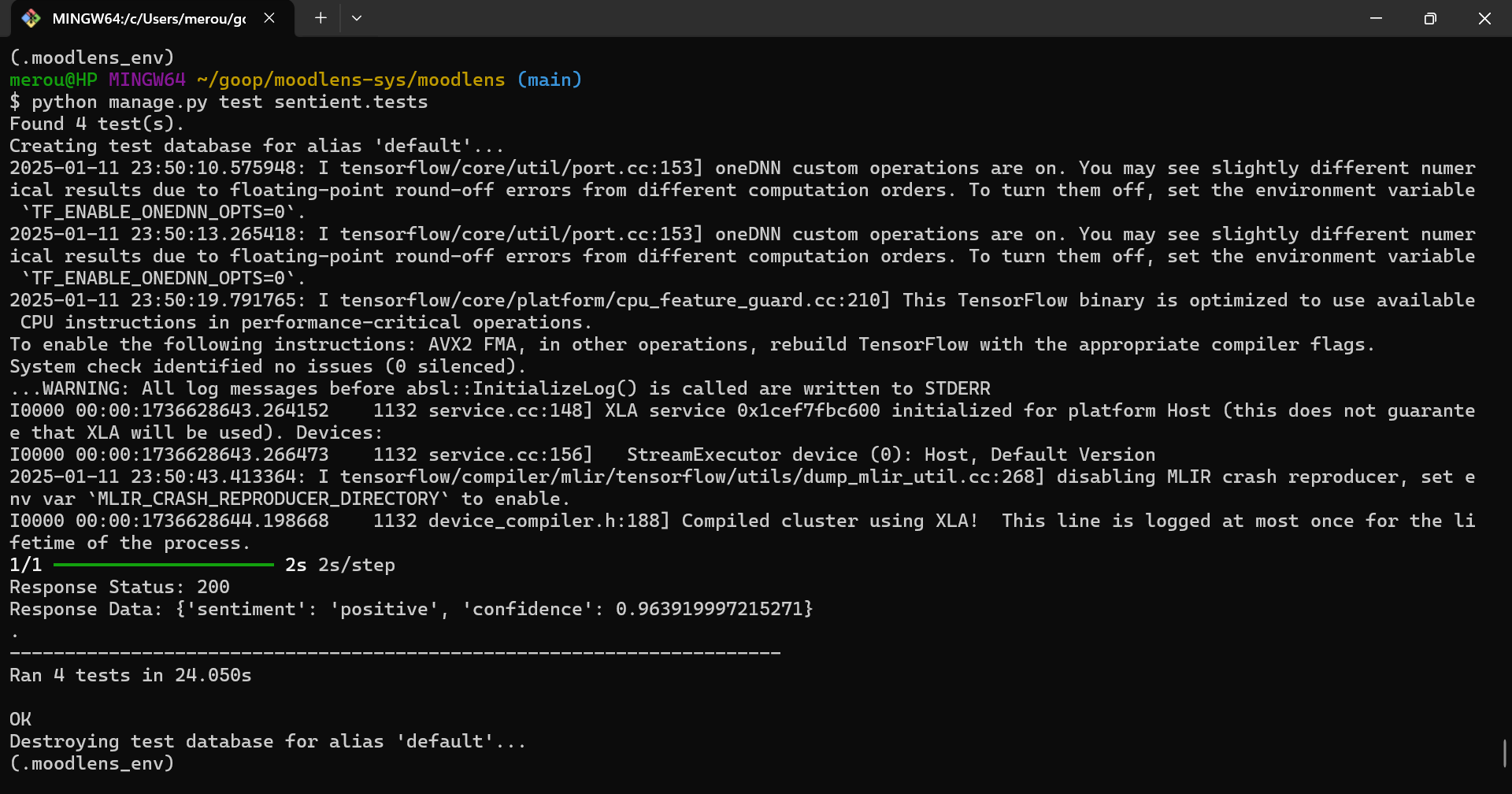
Part 3: Development Testing

In this context, we do testing with the intent to show that our application, Feeler, does what it is intended to do and to discover program defects before it is put into use. There is no correct model or system that is universally acceptable or is correct in the eye of everyone. Regardless, we want to build trust in our app by performing tests on it!

3.1 Unit Testing

With **unit testing**, we want to assess how program components, such as methods or object classes work. This is meant to validate logic and response codes. In this case, we will test for the following each with its test cases:

* **Authentication** Endpoint - Tests: these tests verify that when valid text is provided (e.g., "I love this Feeler!"), the sentiment analysis API responds with a 200 OK status and includes the expected sentiment and confidence fields in the response. Also, they ensure that the API requires authentication. If the token is not provided, the API should return an Unauthorized error (HTTP 401).
* Valid credentials return tokens.
* Invalid credentials return a 401 error.
* **Sentiment** Analysis Endpoint - Tests: these tests check if the sentiment analysis work for edge case inputs, such as very short or unusually structured text, ensuring that the system still provides a valid response
* Valid input returns sentiment and confidence.
* Invalid input (e.g., empty text) returns a 400 error.



Feeler figure 1:unit testing suite(backend)

**Explanation of Output**:

* The tests were executed in 24.050 seconds.
* All tests passed without errors.
* The API responded as expected for both valid and invalid inputs, returning appropriate status codes and response data.
* The **200 OK** status was returned for valid inputs, and **400 Bad Request** or **401 Unauthorized** statuses were returned for incorrect inputs.

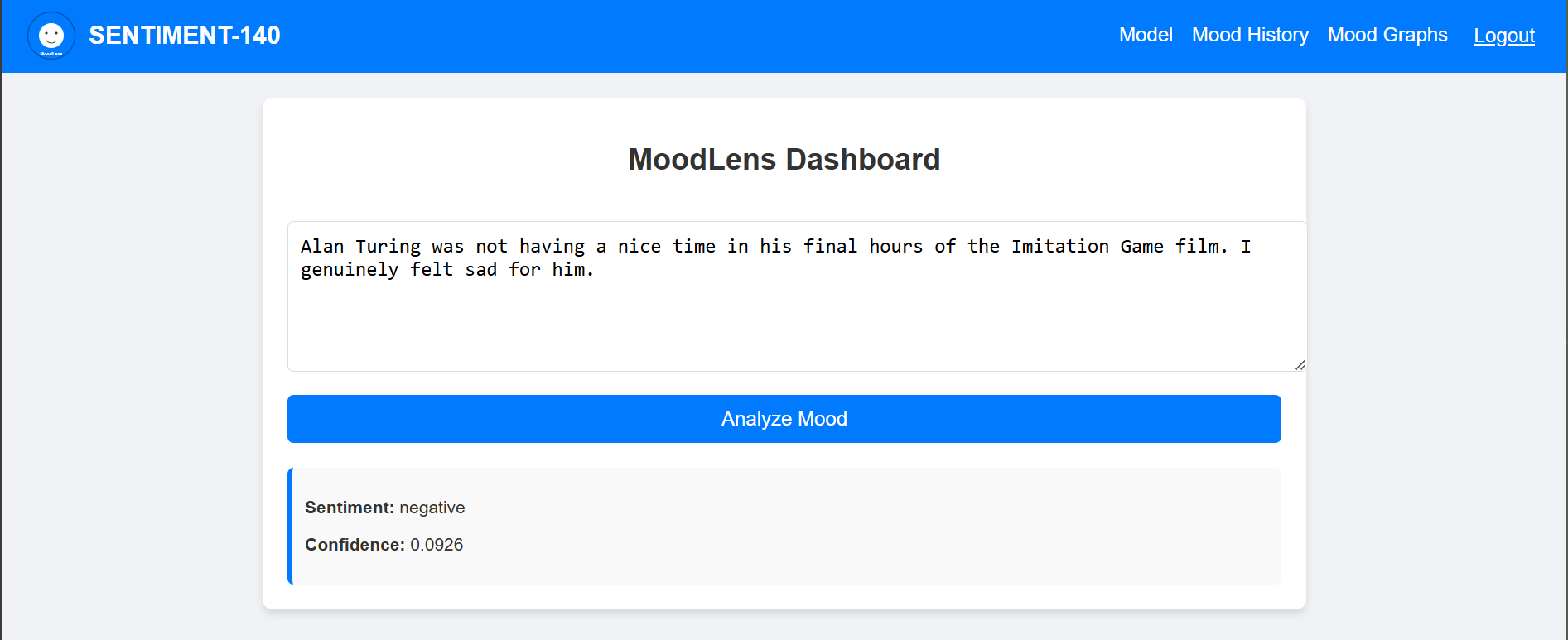
3.2 Integration Testing

The purpose of this was to verify that the backend integrates smoothly with the frontend that consumes data from the backend and to validate that they are communicating correctly. This captures the interaction between the React frontend and Django backend/REST API.

We can use **Postman** client, a graphical user interface for this purpose or **curl**, **a command-line program** that serves the same purpose. We will provide an example of the use curl to make HTTP requests to the following (sampled) endpoints.

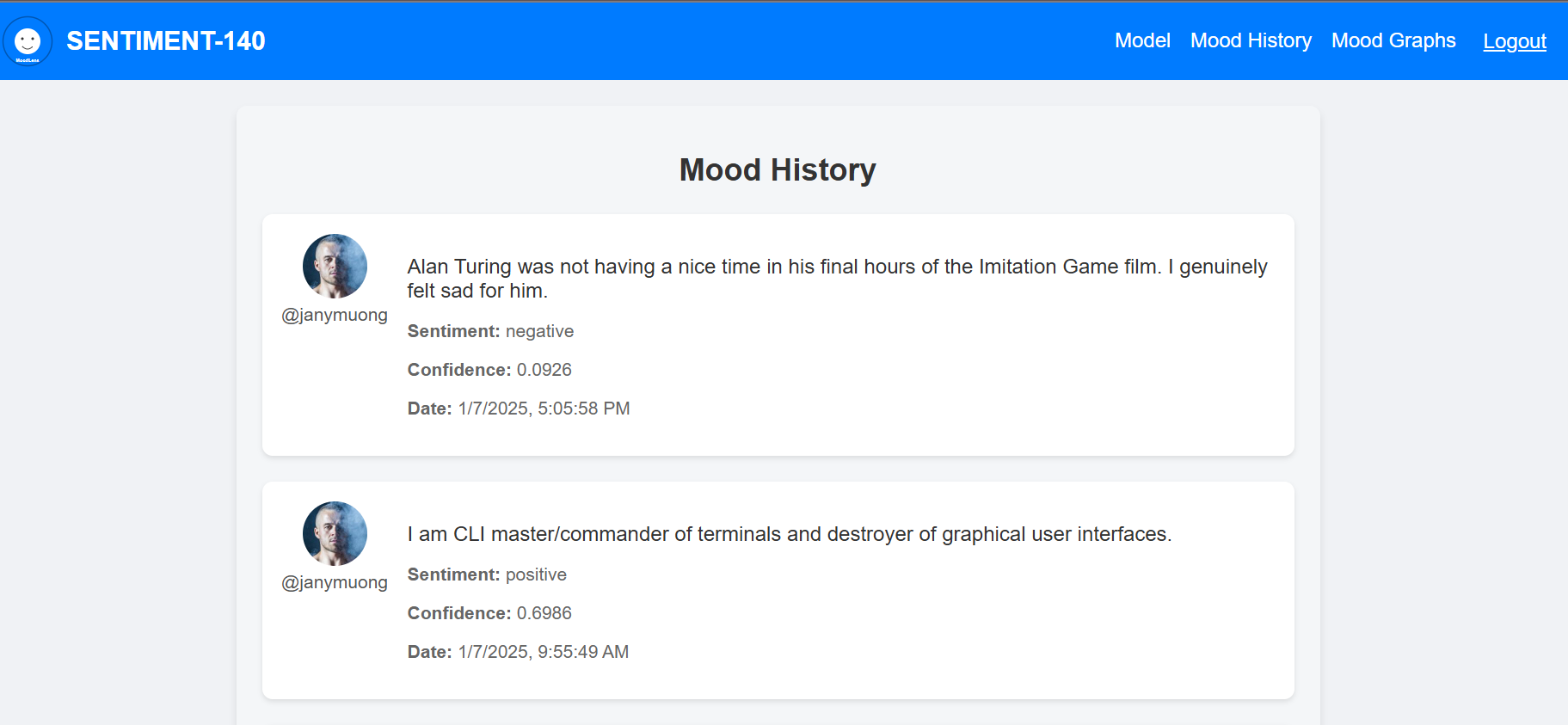
We observed the user interface when it communicates with the following. This was achieved by doing a simple task like entering text and eye inspection.

* Testing /api/predict/ using the frontend form and ensure the response is displayed correctly. This can be seen below:

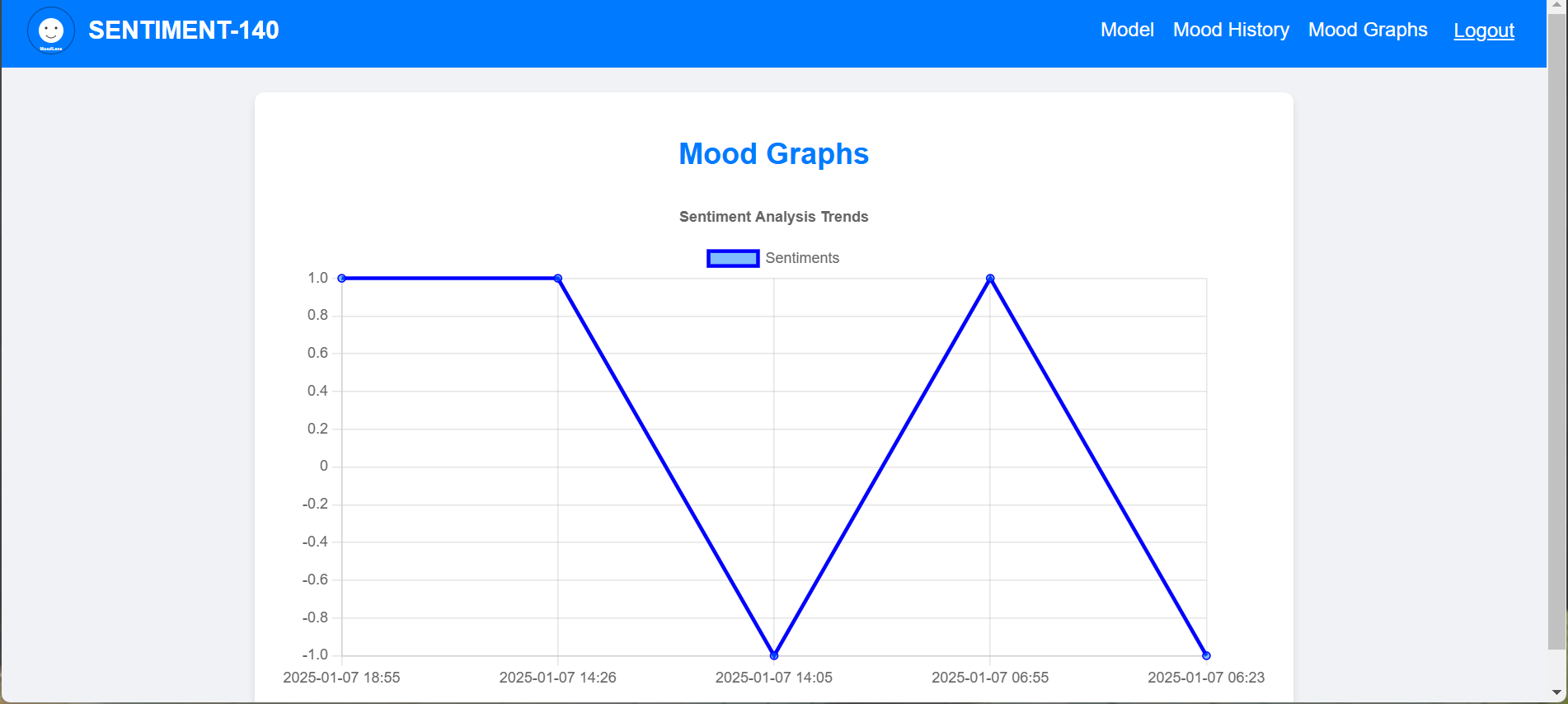


Feeler figure 2: integrating the sentiment analysis endpoint

* Testing /api/history/ by verifying that the history page shows the correct data for the logged-in user.



Feeler figure 3: integrating history of past sentiment

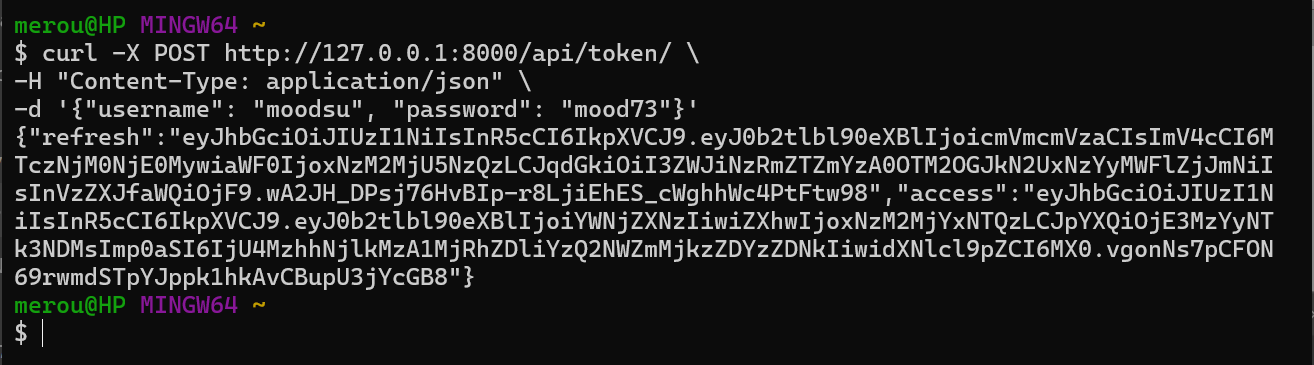
* **Testing** /api/visualization/:the image below also shows integration works when making requests to this API endpoint - from the React front-end (user interface)**:**

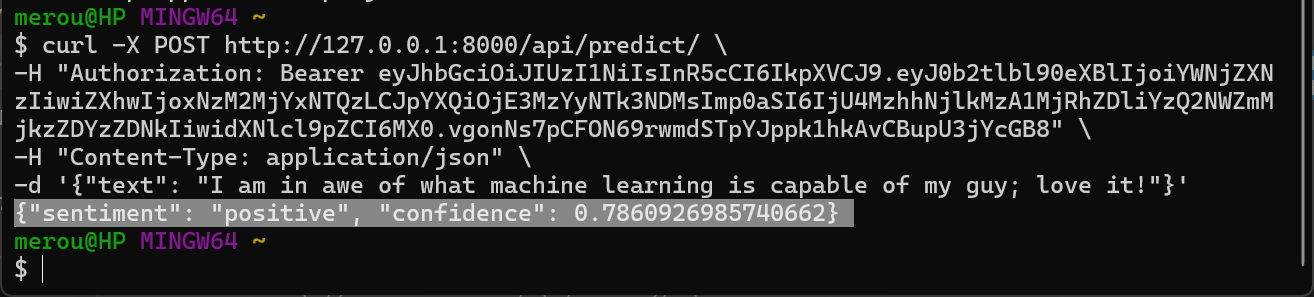
Feeler figure 4: integrating visualizations

**CURL Command-line Utility – Testing Authentication**:

Still following on integration testing, this is what we did in the order they are appearing – using **curl**:

1. using **curl**, we sent a **POST** request to /api/token/ with valid credentials- this gets us a JSON Web Token (JWT) for the user to be authorized to perform the step 2.
2. used the received access token to make a POST request to /api/predict/; this got us an analyzed text from the machine learning/inference endpoint(second figure)





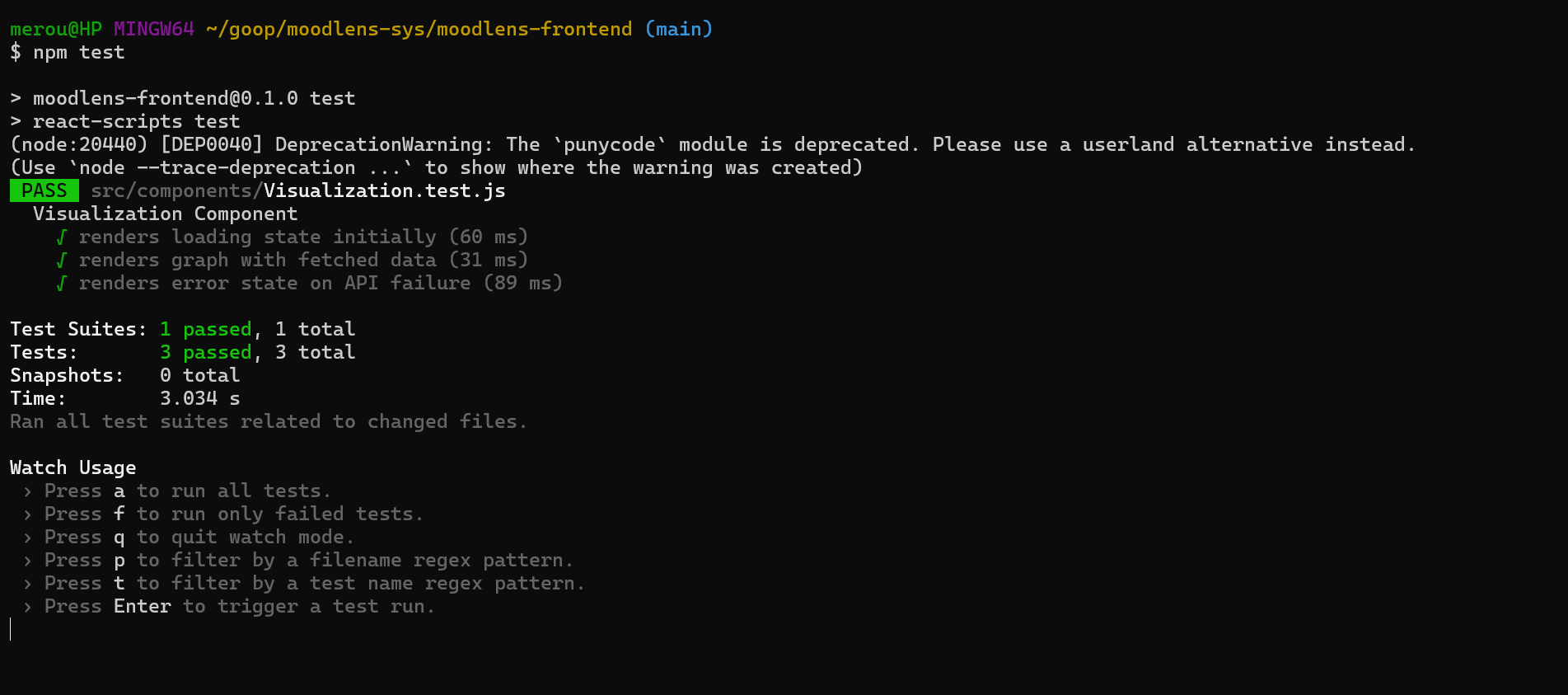
Feeler figure 5: authentication

The verdict is: these tests verify that the backend and the frontend (user interface are communicating well and integrating smoothly).

3.3 Component Testing

We want to ensure that individual pieces of the application (like React components or backend modules) function as expected in isolation. We will focus on the frontend for this, and show how **visualization** is rendered correctly. Testing is done using the JEST framework for React.JS ie **@testing-library/jest-dom** and **@testing-library/react**:

**What was done**: we made a mock API call to get some data for visualization and ascertained that it is rendering graph correctly. This run against the file that does the visualization. Please see below for reference



Feeler figure 6: testing visualization component(frontend test suite)

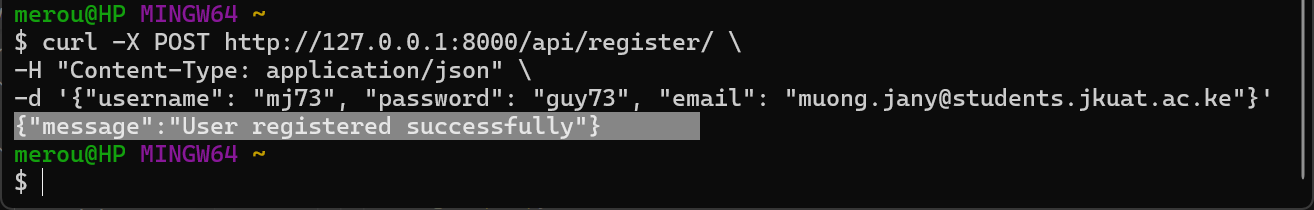
Explanation:

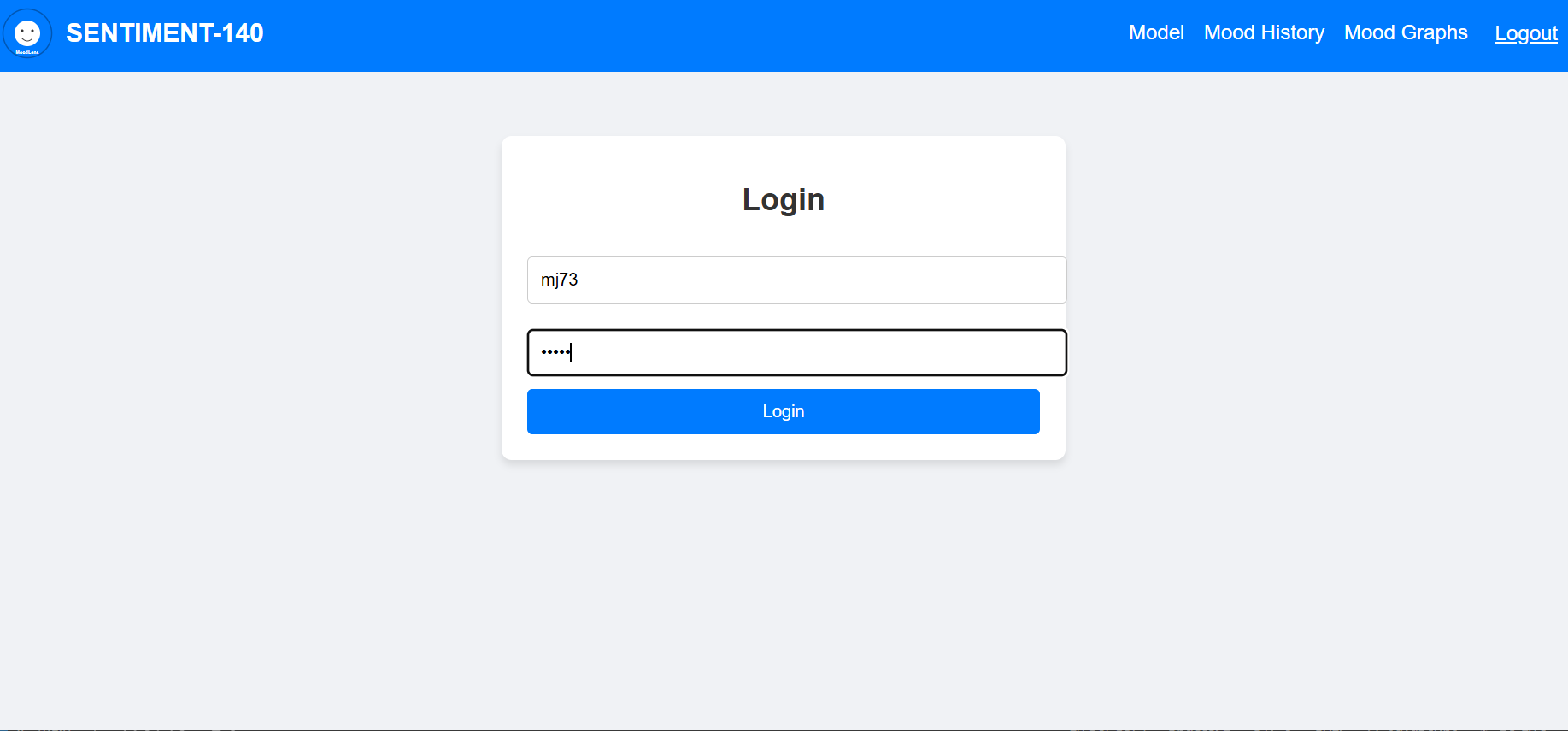
* **Loading state**: confirms that a "Loading..." text is displayed initially when the component is rendered.
* **Graph rendering with data**: mocks the API to return sample data (labels and sentiments), waits for the graph to render and verifies the presence of graph-related text (e.g., "Mood Graphs").
* **Error handling**: simulates a failed API call and ensures an error message ("Error fetching visualization data") is displayed.

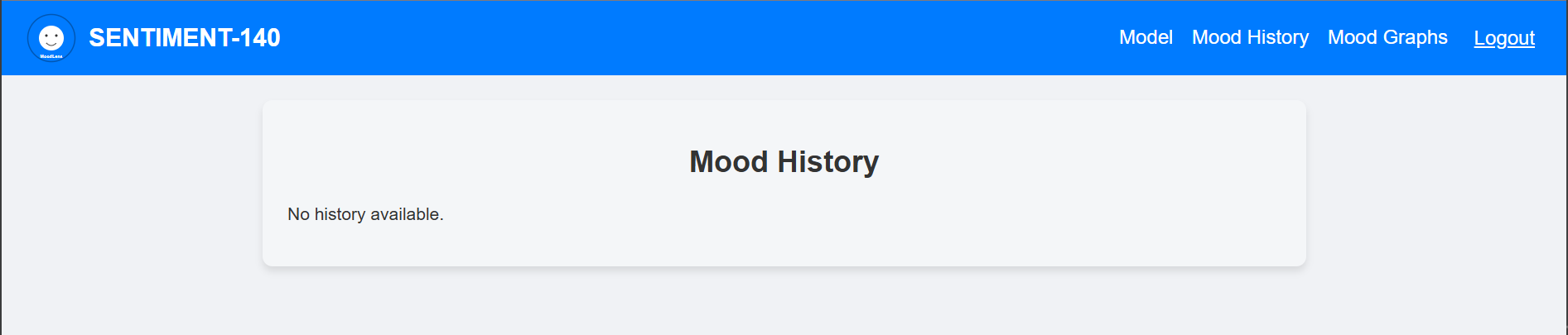
**3.4** System Testing and Functional Testing

We validated end-to-end workflows. The primary motivation here is integrating components to create a version of Feeler with the components together and then testing the integrated system. We checked that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.

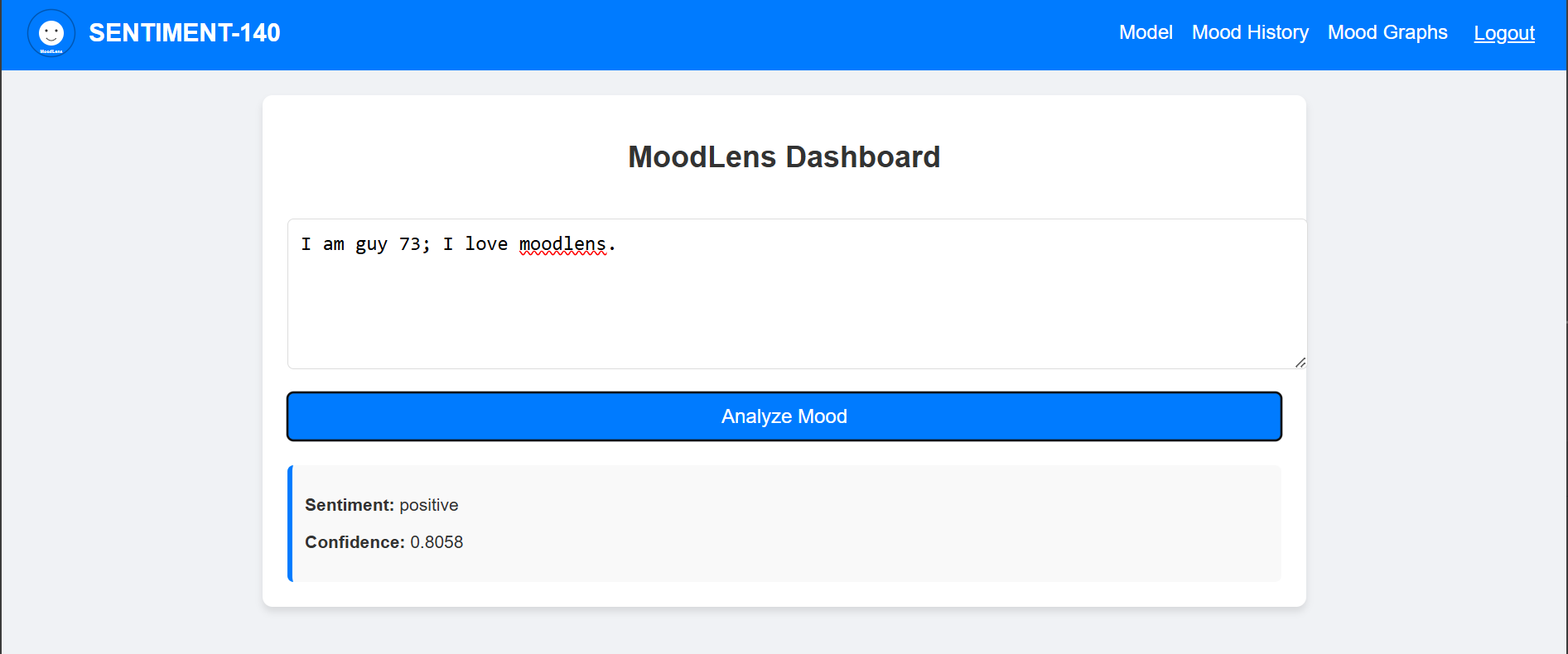
**What was done/WorkFlow**: User registration → Login → Analyze sentiment → Save history → Logout

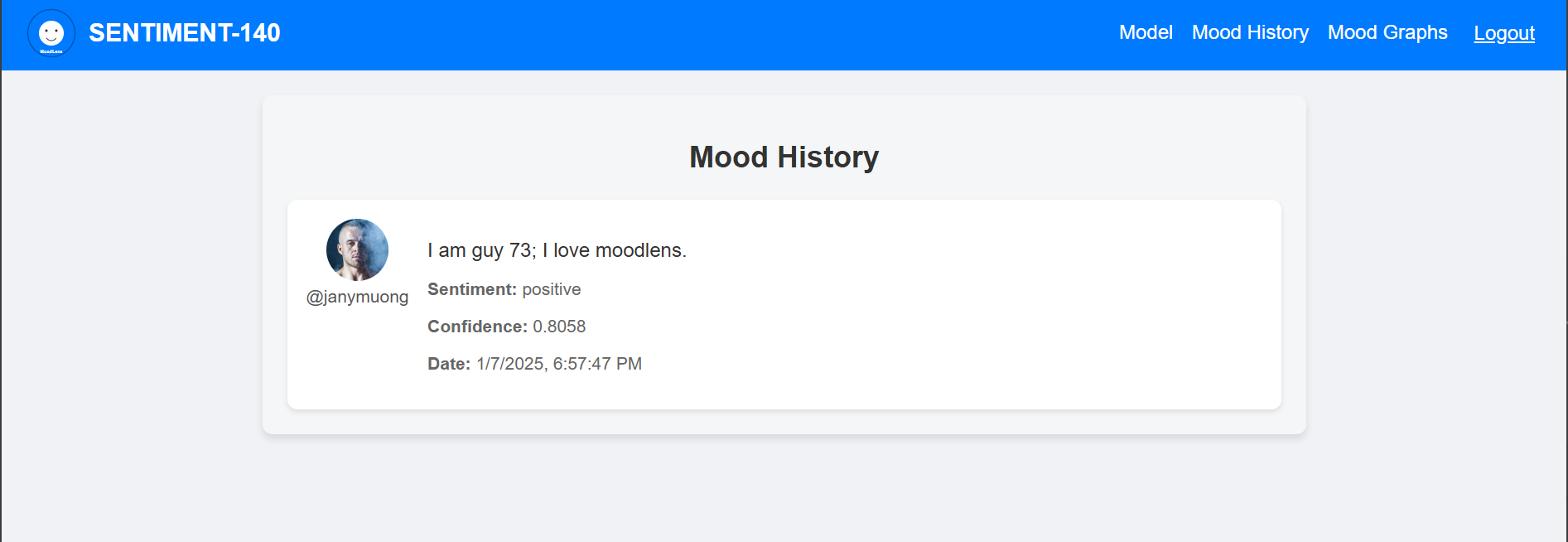
1. Register a user(used **curl** to make registration of new user faster/ we will use the UI for the next steps)
2. Log in using valid credentials and test access to protected routes (we used the history page, as can be seen no data yet).



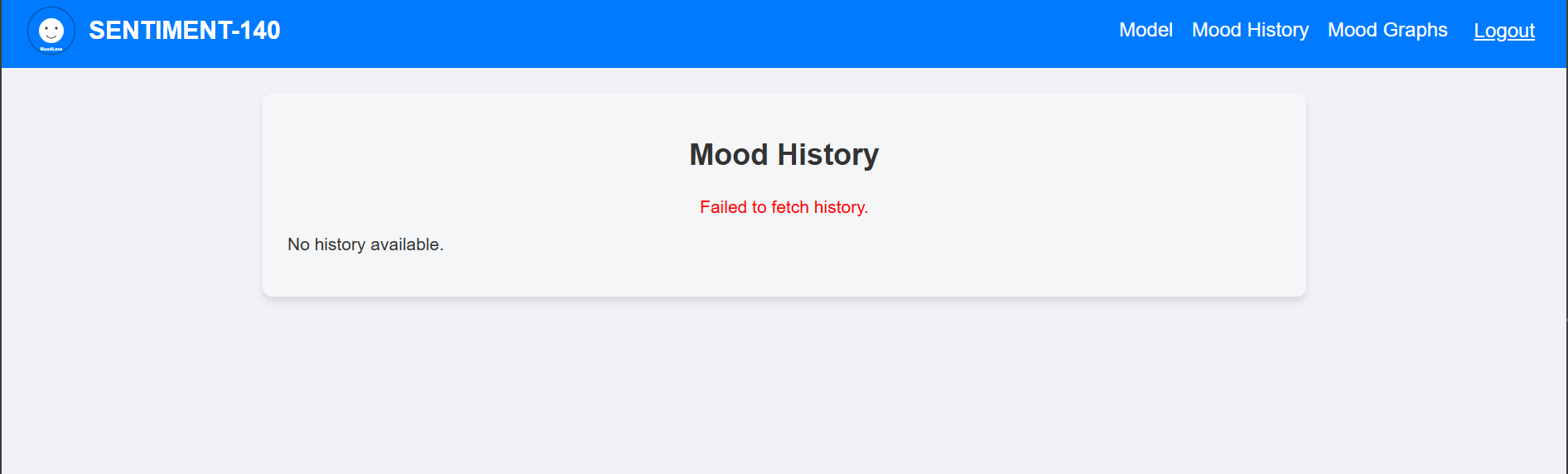


1. Analyze text sentiment, save the result, and view it in history.





1. Log out and ensure tokens are invalidated(the user that viewed the history above cannot view the per-user history because the session is terminated by logging off)

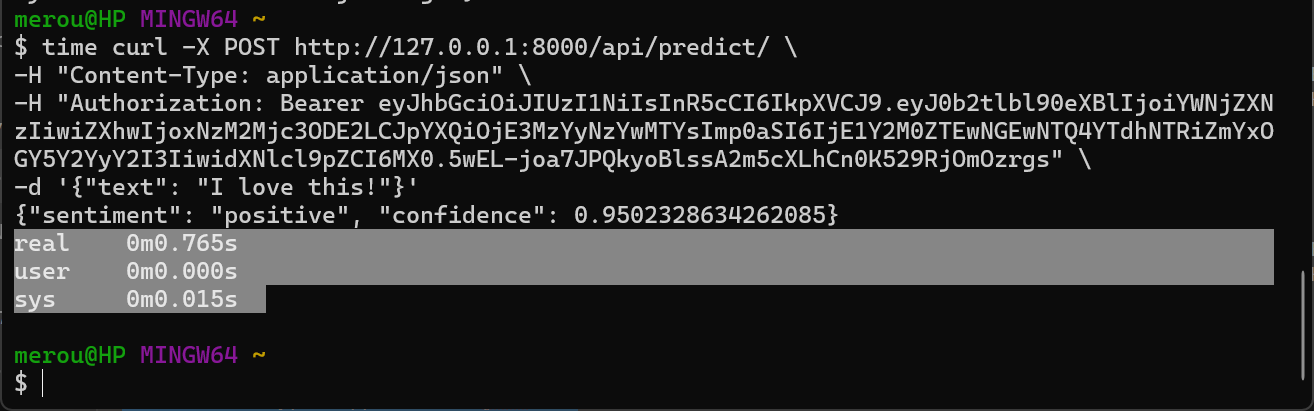


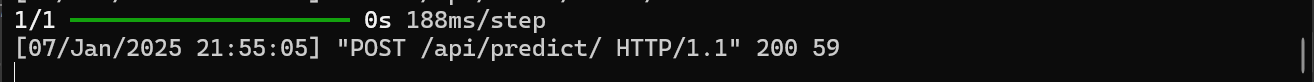
Feeler figure 7: workflow/system testing

3.5. Performance Testing

The **purpose** is to gauge whether API response times are acceptable. E.g prediction endpoint responds within 1 second. We use tools like time in curl.

**What was done**: measured response time for /api/predict/ and ensured it is under 1 second.





Feeler figure 8: performance testing

From the snaps above the time command outputs three time metrics:

* **real**: total time taken (wall clock time).
* **user**: time spent in user-mode (CPU).
* **sys**: time spent in kernel-mode (CPU).

**In this case specifically,** real 0m0.765s means the model inference endpoint responded in **765 milliseconds** (well under the 1-second requirement).

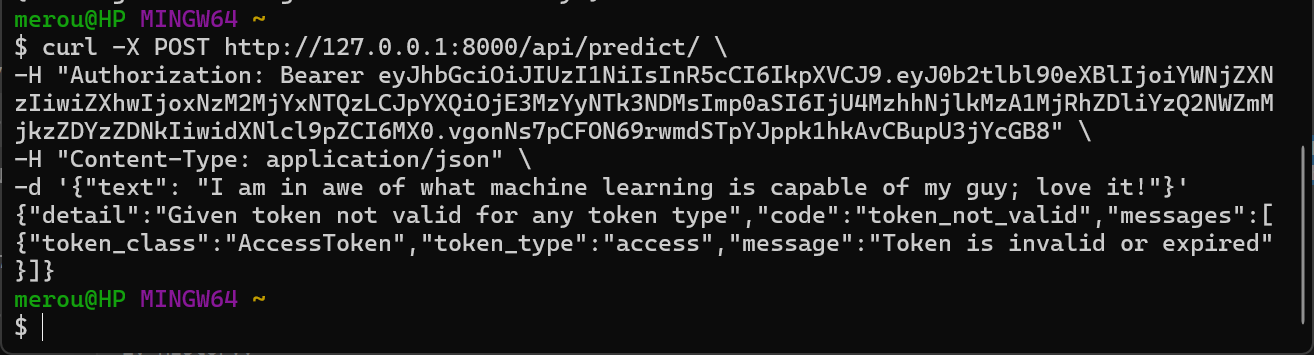
**3.6.** Regression Testing

We had a test suite developed incrementally as our program was developed. With this we ensured new changes (e.g., updated UI) did not break existing features. This was predominantly just observing the UI and ascertaining that new changes did not brake existing functionality as we add them. We corrected them when they did. We did run the systems after modifications or updates. This was iterative and for lack of enough space we left out all our struggles on it, fixing when we broke part of the working of our application :)

**3.7.** Security Testing

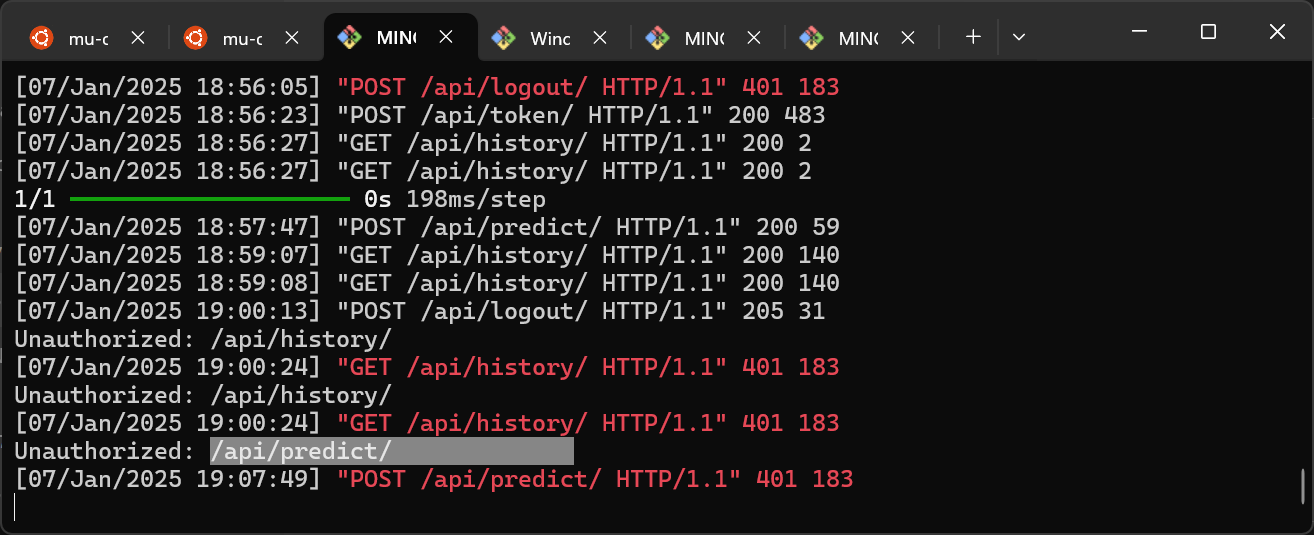
We are testing JWT expiration and invalidation on logout. This ensures the app is secure from token misuse.

**What was done**: we tested expired and invalid tokens against protected routes. This is because logout invalidates JWT tokens.



Feeler figure 9: testing invalidated JWTs

From the **backend logs** below, we can see that the request did reach the API endpoint but is unauthorized: as seen below the token was valid for any request made before logout action.



Part 4: Testing Results

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Expected Outcome** | **Result** |
| Registration with valid inputs | User is successfully registered | ✅ Pass |
| Login with invalid credentials | Displays "Invalid credentials" error | ✅ Pass |
| Sentiment analysis workflow | Prediction and confidence displayed | ✅ Pass |
| History retrieval | Displays the logged-in user's history | ✅ Pass |
| Visualization chart | Renders mood trends correctly | ✅ Pass |
| Logout | Invalidates tokens and redirects to login | ✅ Pass |

Conclusion and The Future

We deliver to you Feeler; a robust sentiment analysis application, complete with secure authentication, user-friendly UI, and real-time data visualization. We can say the the extensive testing we did in here ensures reliability and usability. Also, we can state that we have implemented most of what we set out to achieve within the stipulated time in terms of how the application works. However, we have not gone as far as to implement a genuinely **good user management** feature that would allow a person to update the **profile pictures, resetting passwords and adding advanced security features like 2-factor authentication**. Also we look forward to implanting more features in the mix like **weekly and monthly *sentiment trends***, and a **recommendation feature** for when to ease out on your emotion by doing for instance, exercise and taking short walks in the neighborhood.

End