SOCIAL MEDIA AGGREGATOR SERVER

TERM PROJECT BY:

JILLELLA SUSMITHA
REGD.NO: 23BQ1A4266

PROJECT GUIDE :

Mr. M.PARDHA SARADHI
Associate Professor

INTRODUCTION

- Managing multiple social media platforms can be overwhelming for users who need to stay updated and interact seamlessly across networks.
- The **Social Media Aggregator Server** is a solution designed to simplify this process by integrating content from various social media platforms into a single, unified interface.
- Built using **Flask** and the **Requests** library, this project demonstrates how to create a robust backend service capable of fetching, processing, and organizing social media data.
- It supports features like real-time data aggregation, API rate limit management with token rotation, and a basic web interface for displaying consolidated feeds.
- This project offers users a streamlined way to interact with their social media content, enhancing convenience and productivity while showcasing modern web development and API integration principles.

EXISTING SYSTEM AND PROPOSED SYSTEM

Existing System:

- Users must log in and switch between multiple platforms.
- No centralized way to view or manage data.
- Prone to disruptions due to API rate limits.
- No unified user interface; each platform has its own design.

Proposed System:

- Consolidates all social media content into one interface.
- Aggregates and organizes data for easy filtering and querying.
- Manages API rate limits using token rotation.
- Provides a simple, unified, and user-friendly interface.
- Offers real-time updates and is scalable for future platforms.

CONCEPT-APPROACH

Concept :

The project aggregates Twitter data by interacting with Twitter's API through a Flask web server. It fetches tweets from a user’s timeline, handles API rate limits, and presents the data via a simple web interface. The core idea is to manage token rotation to avoid disruptions caused by rate limits while providing a smooth and unified view of social media content.

Approach :

1.Backend Setup:

- **Flask** is used to create a server with two main endpoints: one for the home page (/) and one for retrieving tweets (/get_tweets).

2.API Integration:

- The **Requests** library is used to interact with Twitter's API for fetching user data and tweets.
- Token rotation is implemented by using multiple bearer tokens to handle API rate limits effectively.

3.Fetching and Handling Data:

- The server sends a request to Twitter’s API to get the user ID using the username.
- Once the user ID is obtained, the server fetches tweets from that user's timeline using the API.
- Data is parsed, organized, and returned as a list of tweets, including tweet ID, text, and creation time.

4.Error Handling:

- The server handles common errors like network issues, invalid usernames, and API rate limits with retries and pauses.

5.User Interface:

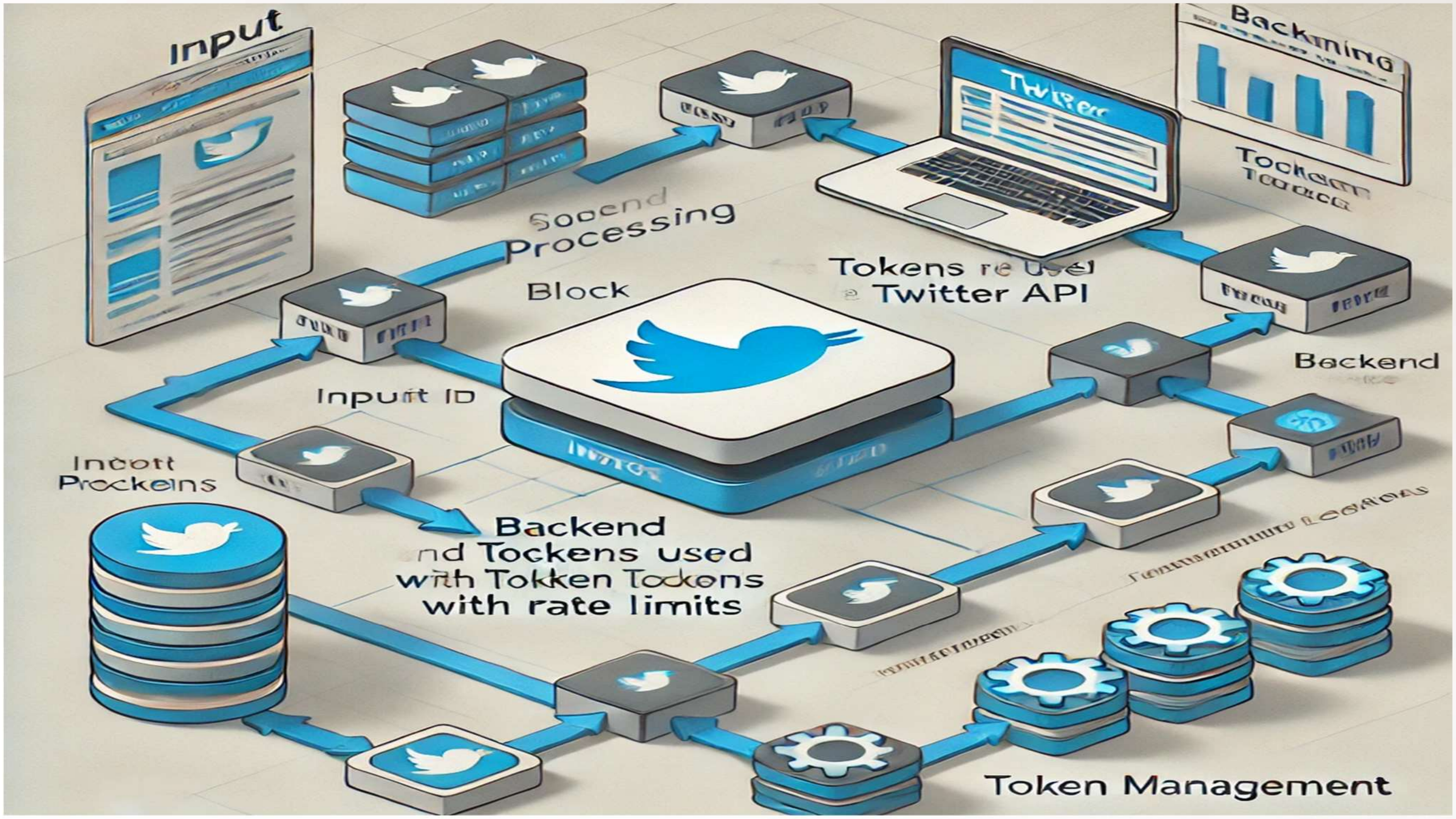
- A basic HTML interface allows users to input a Twitter username and view their aggregated tweets in a structured format.
- The interface also displays a loading indicator while data is being fetched, improving user experience.

6.Testing & Optimization:

- The server checks for errors at each step (e.g., failed API requests, missing user data) and provides appropriate feedback.
- The system can scale to handle multiple users and various Twitter accounts due to its modular structure.

DESIGN-ARCHITECTURE

1. **Client-Server Model**:
   - Frontend (HTML/JS) communicates with the backend (Flask server).
2. **Frontend**:
   - User submits Twitter username.
   - Sends POST request to Flask server to fetch tweets.
3. **Backend (Flask)**:
   - Handles user ID retrieval and tweet fetching using the Twitter API.
   - Implements token rotation for handling rate limits.
4. **External Layer**:
   - Interacts with **Twitter API** using **Bearer Tokens**.
5. **Data Flow**:
   - User input → Flask backend → Twitter API → Response sent to frontend.
6. **Key Technologies**:
   - **Flask**, **Requests**, **JavaScript (AJAX)**, **HTML/CSS**.
7. **Scalability**:
   - Supports additional platforms and scalable token management.

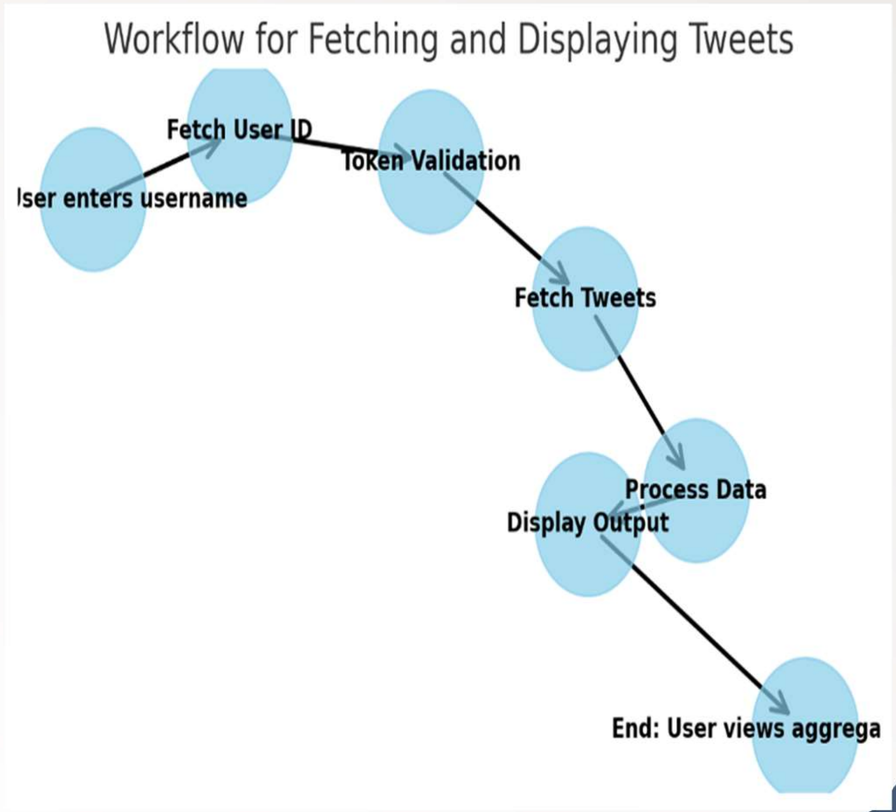

Input
Backining
Tockenn
Soocend
Processing
Tokens re used
Twitter API
Block
Backend
Inpuit ID
Incott
Prockeins
Backend
nd Tockens used
with Tokken Tockens
with rate limits
Token Management

BLOCK DIAGRAM

- **Input**: The user provides an input ID, which could be a Twitter username or a similar identifier for the social media platform.
- **Second Processing Block**: After the input is received, a second processing step occurs, likely to handle or validate the input and prepare it for the next steps in the flow.
- **Token Management**: The system utilizes multiple tokens, which are managed through a dedicated "Token Management" block. This part of the system ensures that tokens are rotated correctly to avoid hitting rate limits imposed by Twitter's API. Tokens are used to authenticate API requests to fetch Twitter data.
- **Twitter API**: The tokens are used to authenticate requests to the Twitter API, enabling the aggregator to retrieve data such as tweets, user information, or other relevant social media content.
- **Backend**: The backend processes the responses from the Twitter API, manages the tokens, and sends the aggregated data back to the frontend or user interface for display. This includes handling rate limits by switching tokens when necessary.
- **Backend and Token Rotation**: The backend interacts with the token management system, ensuring that each token is used efficiently and that API rate limits are adhered to. The backend also stores and manages the data retrieved from Twitter.

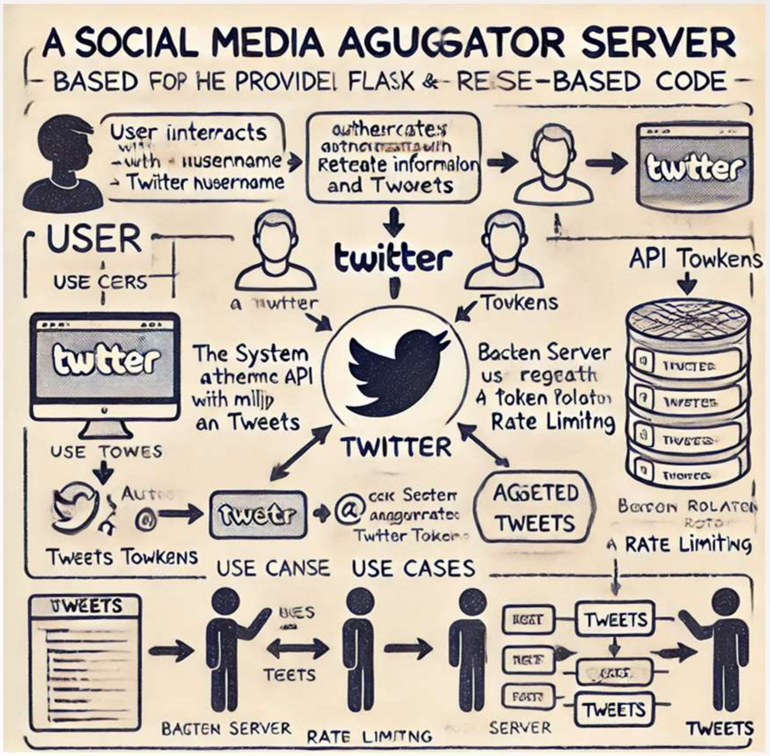
FLOW CHART

User Enters Username:
• The process starts when the user inputs a Twitter username into the system.
Fetch User ID:
• The system queries the Twitter API to retrieve the user ID associated with the given username.
Token Validation:
• The system ensures that the API request is authenticated and valid by checking the token for access to Twitter's API.
Fetch Tweets:
• Once the user ID is retrieved and the token is validated, the system fetches the user's tweets from the Twitter API.
Process Data:
• The system processes the raw tweet data, such as filtering, organizing, and formatting it for display.
Display Output:
• The processed tweet data is then displayed on the front-end interface, allowing the user to view the aggregated tweets.
End:
• The process concludes with the user viewing the aggregated tweets.

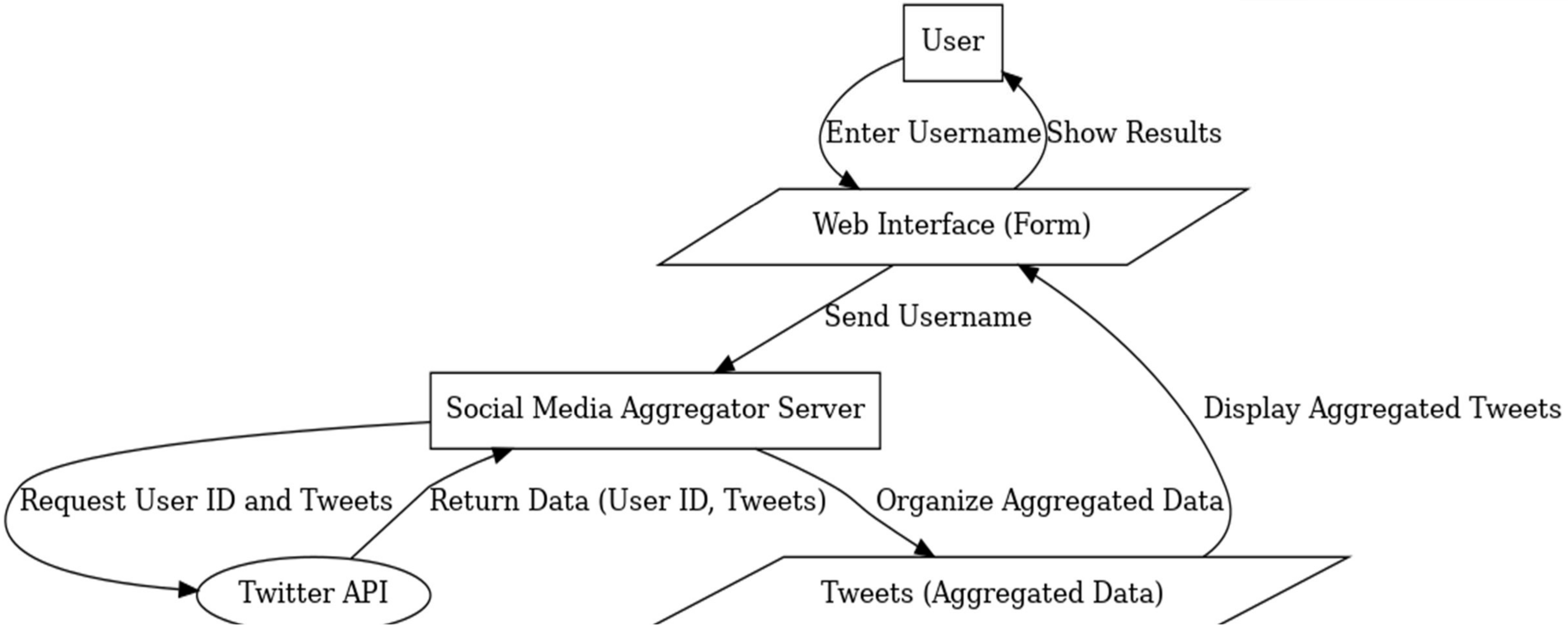


USE CASE DIAGRAM

- **Fetch Tweets**: The server uses API tokens to fetch data while handling rate limits.
- **Error Handling**: Displays messages for invalid usernames or API issues.
- **Tweet Display**: Aggregated tweets are shown in a clean, user-friendly interface.
- **Username Input**: Users provide a Twitter handle.


A SOCIAL MEDIA AGUGGATOR SERVER
BASED FOP HE PROVIDEI FLASK & RE SE-BASED CODE
User iinteracts with usenname Twitter husername
Reteate informalon and Twoets
twitter
USER
USE CERS
API Towkens
Tovkens
The System athemc API with mlljp an Tweets
Bacten Server us regeath A token Polaton Rate Limitng
TWITTER
USE TOWES
AGGETED TWEETS
Tweets Towkens
USE CANSE
USE CASES
A RATE Limiting
TWEETS
TEETS
BACTEN SERVER
RATE LIMITNG
SERVER
TWEETS

DFD DIAGRAM


User
Enter Username
Show Results
Web Interface (Form)
Send Username
Social Media Aggregator Server
Display Aggregated Tweets
Request User ID and Tweets
Return Data (User ID, Tweets)
Organize Aggregated Data
Twitter API
Tweets (Aggregated Data)

BACKEND CODE

1.Fetching User ID by Username:

```
def get_user_id(username, token):
   url = f'https://api.twitter.com/2/users/by/username/{username}'
   headers = {'Authorization': f"Bearer {token}"}
   try:
      response = requests.get(url, headers=headers)
      response.raise_for_status()
      return response.json()['data']['id']
   except requests.exceptions.RequestException as e:
      print(f"Error getting user ID: {e}")
      return None
```

2.Fetching Tweets with Token Rotation:

```
def fetch_tweets_with_retry(user_id, retries=3, delay=10):
   params = {'max_results': 35, 'tweet.fields': 'created_at,id,text'}
   while True:
      for token in BEARER_TOKENS:
         for attempt in range(retries):
            url = f'https://api.twitter.com/2/users/{user_id}/tweets'
headers = {'Authorization': f"Bearer {token}"}
try:
        response =     requests.get(url,headers=headers,
params=params)
       if response.status_code ==200:
          return response.json().get('data', [])
      elif response.status_code == 429:
         break
  except requests.exceptions.RequestException as e:
      print(f"Error fetching tweets: {e}")
time.sleep(delay * (2 ** attempt))
```

3. Endpoint for Fetching Tweets:

```
@app.route('/get_tweets', methods=['POST'])
def get_tweets():
    data = request.get_json()
    username = data.get('username')
    if not username:
        return jsonify({"error": "Username is required"}), 400

    user_id = next((get_user_id(username, token) for token in BEARER_TOKENS), None)
    if not user_id:
        return jsonify({"error": "User not found"}), 404

    tweets = fetch_tweets_with_retry(user_id)
    if isinstance(tweets, dict) and "error" in tweets:
        return jsonify({"error": "API error occurred"}), 500

    return jsonify([
        {'id': tweet['id'], 'text': tweet.get('text', ''), 'created_at': tweet['created_at'], 'username':
username}
        for tweet in tweets
    ])
```

FRONTEND CODE

1. HTML Form for Username Input:

```
<form id="usernameForm" class="form-inline justify-content-center">
   <div class="form-group">
      <input type="text" id="username" name="username" class="form-control"
placeholder="Username" required>
   </div>
   <button type="submit" class="btn btn-primary ml-2">Search</button>
</form>
```

JavaScript for Handling Form Submission:

```
document.getElementById('usernameForm').onsubmit = async function(event) {
    event.preventDefault();
    const username = document.getElementById('username').value;
    const tweetsContainer = document.getElementById('tweetsContainer');
    const errorContainer = document.getElementById('errorContainer');
    const loader = document.getElementById('loader');
    tweetsContainer.innerHTML = '';
    errorContainer.innerHTML = '';
    loader.style.display = 'block';
    try {
        const response = await fetch('/get_tweets', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username })
        });

        if (!response.ok) throw new Error(`Error ${response.status}: ${response.statusText}`);
const data = await response.json();
  loader.style.display = 'none';
if (Array.isArray(data) && data.length > 0) {
            data.forEach(tweet => ){
                    tweetsContainer.innerHTML += `
                        <div class="tweet-card">
                            <div class="tweet-content">
                                <div class="tweet-     username">${tweet.username}</div>
                               <div class="tweet-text">${tweet.text}</div>
                              <div class="tweet-time">${new
          Date(tweet.created_at).toLocaleString()}</div>
                            </div>
                        </div>
                `;
            });
        } else {
            tweetsContainer.innerHTML = `<div>No tweets found for this user.</div>`;
        }
    } catch (error) {
        loader.style.display = 'none';
        errorContainer.innerHTML = `<div>${error.message}</div>`;
    }
};
```

CSS for Tweet Cards:

```
.tweet-card {
    background-color: #ffffff;
    border-radius: 10px;
    padding: 20px;
    margin-bottom: 15px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.tweet-content {
    flex: 1;
}
.tweet-username {
    font-weight: bold;
}
.tweet-text {
    font-size: 1em;
}
.tweet-time {
    font-size: 0.85em;
    color: #777;
}
```

PACKAGES AND TOOLS USED IN MODULES

Backend Tools:

flask:
- A lightweight and easy-to-use web framework for building web applications in Python.
- Provides features like routing, template rendering, and request handling.

requests:
- A powerful library for making HTTP requests in Python.
- Simplifies working with APIs by handling methods like GET, POST, and managing headers.

time:
- A standard Python library for managing time-related functions.
- Used here for delays (e.g., during rate-limiting) and timestamp calculations.

jsonify:
- Converts Python dictionaries into properly formatted JSON responses for APIs.
- Ensures consistent and structured data exchange between the backend and frontend.

Frontend Tools:

Bootstrap:
- Simplifies UI development with pre-designed components like forms, buttons, and grids.
- Ensures responsive and mobile-friendly design effortlessly.

Twitter Widgets.js:
- Embeds tweets, timelines, and other Twitter features without complex code.
- Automatically adjusts content to the user's browser.

Vanilla JavaScript:
- Handles form submissions and asynchronous requests via fetch().
- Dynamically updates the page with tweet cards, enhancing user interactivity.

TEST RESULTS

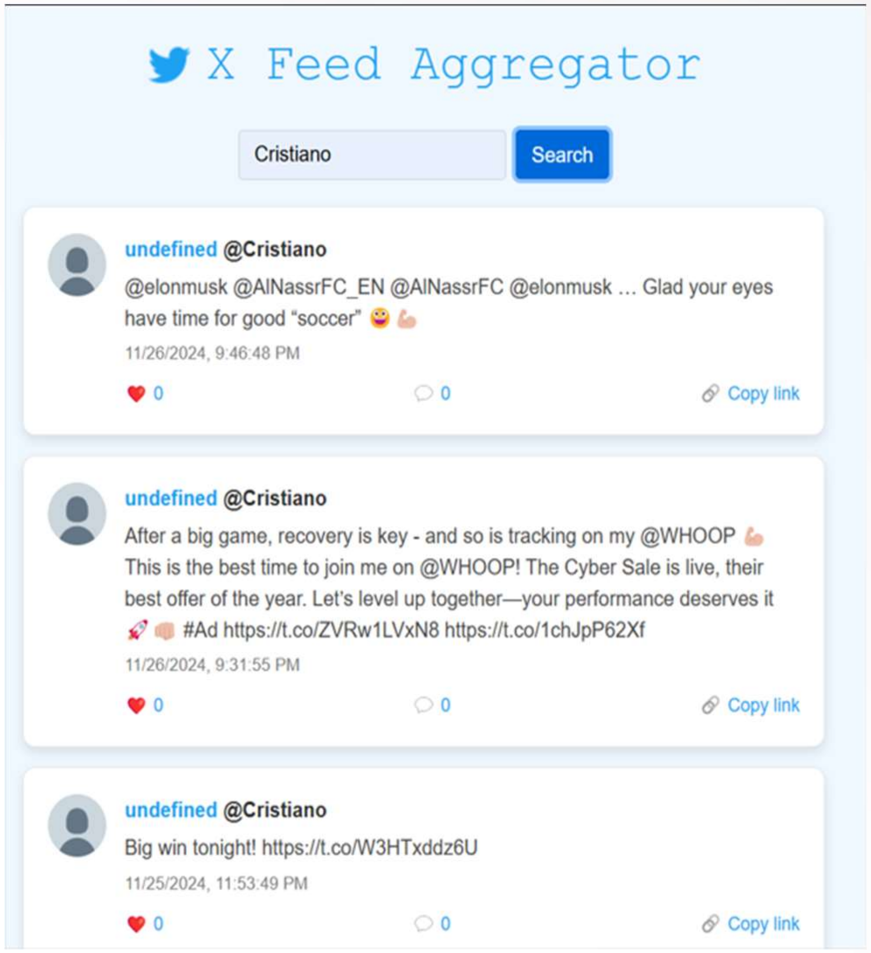
Backend Tests

1. **Test Case**: Valid username with available tweets
   - **Input**: username = "jack"
   - **Output**: List of tweets with ID, text, creation time, and user info.
2. **Test Case**: Valid username with no tweets
   - **Input**: username = "new_user"
   - **Output**: {"error": "No tweets found"}
3. **Test Case**: Invalid username
   - **Input**: username = "invalid_user1234"
   - **Output**: {"error": "User not found"}
4. **Test Case**: Rate limit hit
   - **Simulation**: Exhausted all tokens
   - **Output**: Automatically rotates tokens or delays for reset.

Frontend Tests

1. **Test Case**: User enters valid username
   - **Input**: Username = jack
   - **Output**: Tweets displayed as dynamic cards with text and timestamps.
2. **Test Case**: User enters an invalid username
   - **Input**: Username = invalid_user1234
   - **Output**: Error message displayed: *"User not found."*
3. **Test Case**: No internet connection
   - **Simulation**: Turned off the network
   - **Output**: Error message displayed: *"Unexpected error occurred. Please try again."*

Frontend Result



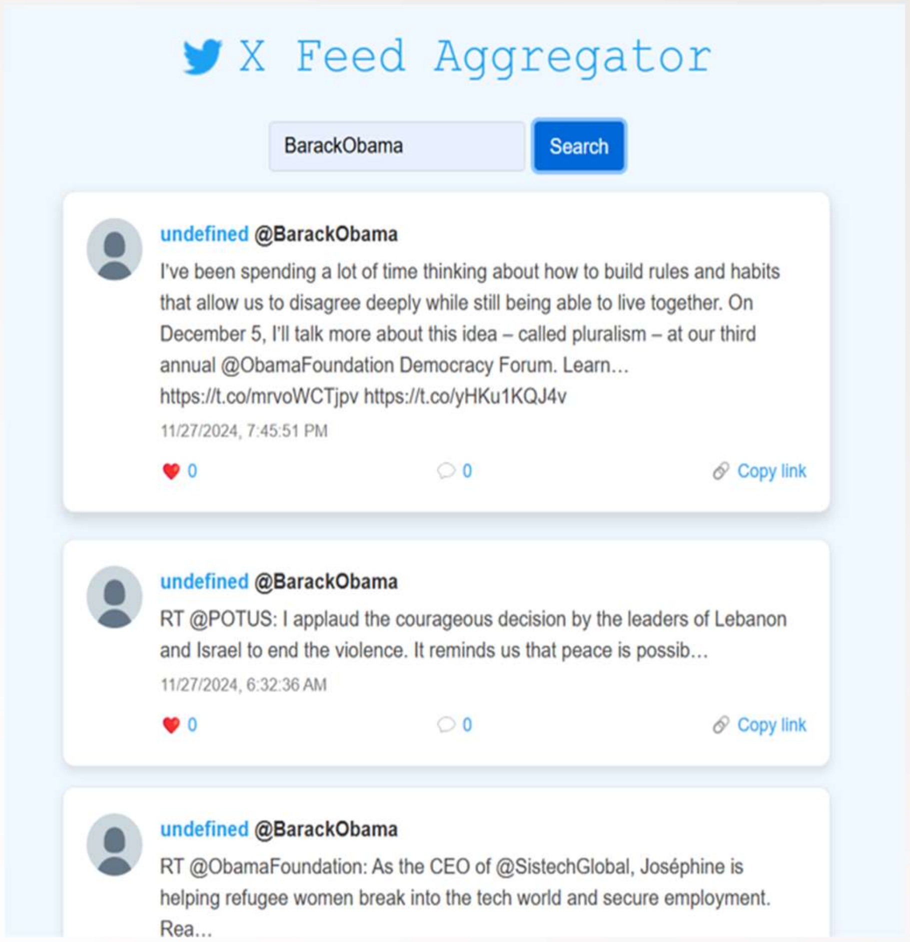
X Feed Aggregator

Cristiano | Search

undefined @Cristiano
@elonmusk @AlNassrFC_EN @AlNassrFC @elonmusk … Glad your eyes have time for good "soccer" 😜 💪
11/26/2024, 9:46:48 PM
❤️ 0 💬 0 Copy link

undefined @Cristiano
After a big game, recovery is key - and so is tracking on my @WHOOP 💪 This is the best time to join me on @WHOOP! The Cyber Sale is live, their best offer of the year. Let's level up together—your performance deserves it 🚀 👊 #Ad https://t.co/ZVRw1LVxN8 https://t.co/1chJpP62Xf
11/26/2024, 9:31:55 PM
❤️ 0 💬 0 Copy link

undefined @Cristiano
Big win tonight! https://t.co/W3HTxddz6U
11/25/2024, 11:53:49 PM
❤️ 0 💬 0 Copy link



X Feed Aggregator

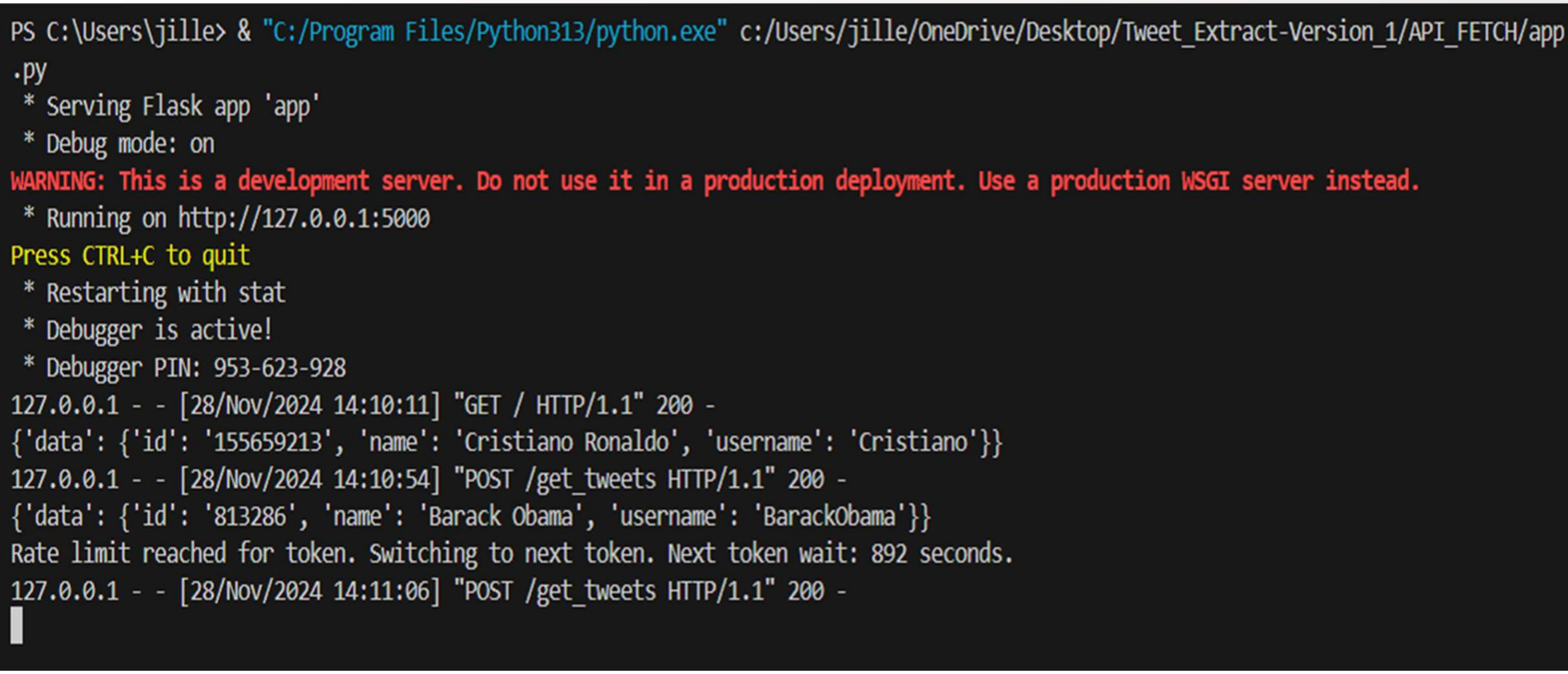
BarackObama | Search

undefined @BarackObama
I've been spending a lot of time thinking about how to build rules and habits that allow us to disagree deeply while still being able to live together. On December 5, I'll talk more about this idea – called pluralism – at our third annual @ObamaFoundation Democracy Forum. Learn… https://t.co/mrvoWCTjpv https://t.co/yHKu1KQJ4v
11/27/2024, 7:45:51 PM
❤️ 0 💬 0 Copy link

undefined @BarackObama
RT @POTUS: I applaud the courageous decision by the leaders of Lebanon and Israel to end the violence. It reminds us that peace is possib…
11/27/2024, 6:32:36 AM
❤️ 0 💬 0 Copy link

undefined @BarackObama
RT @ObamaFoundation: As the CEO of @SistechGlobal, Joséphine is helping refugee women break into the tech world and secure employment. Rea…

BACKEND RESULT



```
PS C:\Users\jille> & "C:/Program Files/Python313/python.exe" c:/Users/jille/OneDrive/Desktop/Tweet_Extract-Version_1/API_FETCH/app
.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 953-623-928
127.0.0.1 - - [28/Nov/2024 14:10:11] "GET / HTTP/1.1" 200 -
{'data': {'id': '155659213', 'name': 'Cristiano Ronaldo', 'username': 'Cristiano'}}
127.0.0.1 - - [28/Nov/2024 14:10:54] "POST /get_tweets HTTP/1.1" 200 -
{'data': {'id': '813286', 'name': 'Barack Obama', 'username': 'BarackObama'}}
Rate limit reached for token. Switching to next token. Next token wait: 892 seconds.
127.0.0.1 - - [28/Nov/2024 14:11:06] "POST /get_tweets HTTP/1.1" 200 -
```

CONCLUSION

- This project demonstrates the effective integration of a robust backend and a user-friendly frontend to aggregate Twitter feeds.
- It efficiently handles API rate limits using token rotation, ensuring uninterrupted data retrieval.
- The application supports dynamic tweet rendering with features like timestamps and media handling, providing a seamless and interactive experience.
- This implementation highlights the potential for scalable and real-time social media integration, laying the groundwork for advanced functionalities and future expansions.

FUTURE SCOPE

1. **Enhanced Features**:
   - Add support for fetching user profile details, follower counts, and trends.
   - Include sentiment analysis for tweets.
2. **Real-time Updates**:
   - Enable auto-refresh for fetching new tweets.
   - Integrate WebSockets for live updates.
3. **Extended API Coverage**:
   - Include hashtag-based tweet searches.
   - Support fetching tweets from multiple users simultaneously.
4. **Improved UI/UX**:
   - Add dark mode and customizable themes.
   - Provide options to save favorite tweets locally or share directly.
5. **Scalability**:
   - Use caching (Redis or similar) to optimize API usage.
   - Host on scalable platforms like AWS or Google Cloud for high traffic.

REFERENCES

1. **Flask Documentation**
   - Official Flask documentation for routing, templates, and request handling.
2. **Twitter API Documentation**
   - Details on Twitter API endpoints, rate limits, and authentication.
3. **Bootstrap Documentation**
   - Guide for building responsive and visually appealing frontend components.
4. **Requests Library**
   - Python HTTP library for API calls and error handling.
5. **JavaScript Fetch API**
   - For handling asynchronous requests and responses in the frontend.
6. **Twitter Widgets.js**
   - Embedding tweets and integrating additional Twitter features.

THANKS!