Back-End Social Analytics Screener

Project Overview:

For this back-end development screener at NCRI (National Center for Research and Innovation), you are tasked with creating a robust API that will support a front-end dashboard for analyzing social media data. This API will facilitate the collection, processing, and retrieval of simulated Twitter data, enabling real-time analytics and interactive visualizations. The focus is on creating efficient and scalable services.

Goals

1. Develop an API that provides essential functionalities required for the front-end dashboard to perform data filtering, analytics, and visualizations effectively.
2. The API should handle large volumes of data efficiently and provide fast responses to front-end requests.

Core Features:

1. **Data Management:**
    - Use the attached Twitter, read into a Database (Postgres or Mongo)
    - Provide robust data insertion and querying capabilities to support real-time analytics.
2. **API Endpoints:**
    - **Data Filtering:** Endpoints to filter data by time (day, week, month) and content type (threatening, non-threatening, hateful, neutral).
    - **Analytics:** Endpoints to calculate and return statistics such as the increase in specific types of tweets over time, key users, and keywords.
    - **Visualization Data:** Provide data suitable for generating line graphs, bar graphs, and heatmaps.
3. **Security and Authentication:**
    - (Optional) Implement secure access to the API with authentication mechanisms like OAuth or JWT (JSON Web Tokens) to ensure that only authorized users can access the data.
4. **Performance Optimization:**

- (Optional) Use caching mechanisms to enhance the performance of the API by reducing database load.
- Implement rate limiting to prevent abuse and ensure equitable resource usage among consumers.

Technical Requirements:

- **Back-End Framework:** Preference for FastAPI, Django REST framework, or Flask.
- **Database:** Use SQL (e.g., PostgreSQL) or NoSQL (e.g., MongoDB) databases based on the complexity and nature of data.
- **Authentication:** Implement authentication and possibly authorization with JWT, OAuth, or similar technologies.
- **Testing:** Write unit and integration tests using frameworks like Jest (for Node.js) or PyTest (for Python).
- **Documentation:** Document the API endpoints with tools like Swagger or Postman. Or FastAPI.

Design Considerations:

- Follow RESTful practices or GraphQL for API development to facilitate easy integration and scalability.
- Design the API for high availability and fault tolerance.
- Deployment and scaling

Evaluation Criteria:

- **Functionality:** Completeness of API functionalities and their correct implementation.
- **Performance:** Efficiency in handling data operations, response times, and ability to handle concurrent requests.
- **Code Quality:** Organization, readability, and maintainability of code.
- **Documentation:** Clarity and completeness of API documentation.

Submission Guidelines:

- Provide a GitHub repository link with all source code, along with detailed README instructions for setting up and testing the API.
- Include a collection of API requests in Postman or similar tool to demonstrate how the API endpoints work.

Tweets can be found here:

Data Dictionary

1. **lang**: The language code representing the language in which the content is written.
2. **retweet_count**: The number of times this content has been retweeted by other users.
3. **retweeted**: A boolean (true or false) indicating whether the content has been retweeted by the user.
4. **created_at**: The date and time when the content was originally posted.
5. **full_text**: The complete text of the content as it was posted.
6. **reply_count**: The number of replies the content has received from other users.
7. **id**: A unique identifier for the content.
8. **author**: The username or identifier of the person who posted the content.
9. **author_created_utc**: The date and time when the author's account was created, often in UTC time zone.
10. **text**: Typically, this would represent the raw or unprocessed text of the content.
11. **len_filter**: A metric or filter based on the length of the content, potentially used for processing or analysis.
12. **clean_text**: The text of the content after undergoing cleaning processes, like removing URLs, special characters, or normalization.
13. **datetime**: A datetime stamp associated with the content, often used for sorting or time series analysis.
14. **year**: The year extracted from the datetime when the content was posted.
15. **month**: The month extracted from the datetime when the content was posted.
16. **day**: The day extracted from the datetime when the content was posted.
17. **hour**: The hour extracted from the datetime when the content was posted.
18. **minute**: The minute extracted from the datetime when the content was posted.
19. **second**: The second extracted from the datetime when the content was posted.

20. **year_month**: A combined field of year and month, useful for monthly trend analysis.
21. **year_month_day**: A combined field of year, month, and day, useful for daily trend analysis.
22. **follower_count**: The number of followers the author of the content has.
23. **threat_level**: A classification or assessment of the potential threat level of the content, perhaps based on its content or context.
24. **hateful**: A boolean (true or false) indicating whether the content is considered hateful based on certain criteria.
25. **zip**: This could refer to a zip code associated with the author or content, or potentially a zipped aggregation of data.