Social Post Recommendation

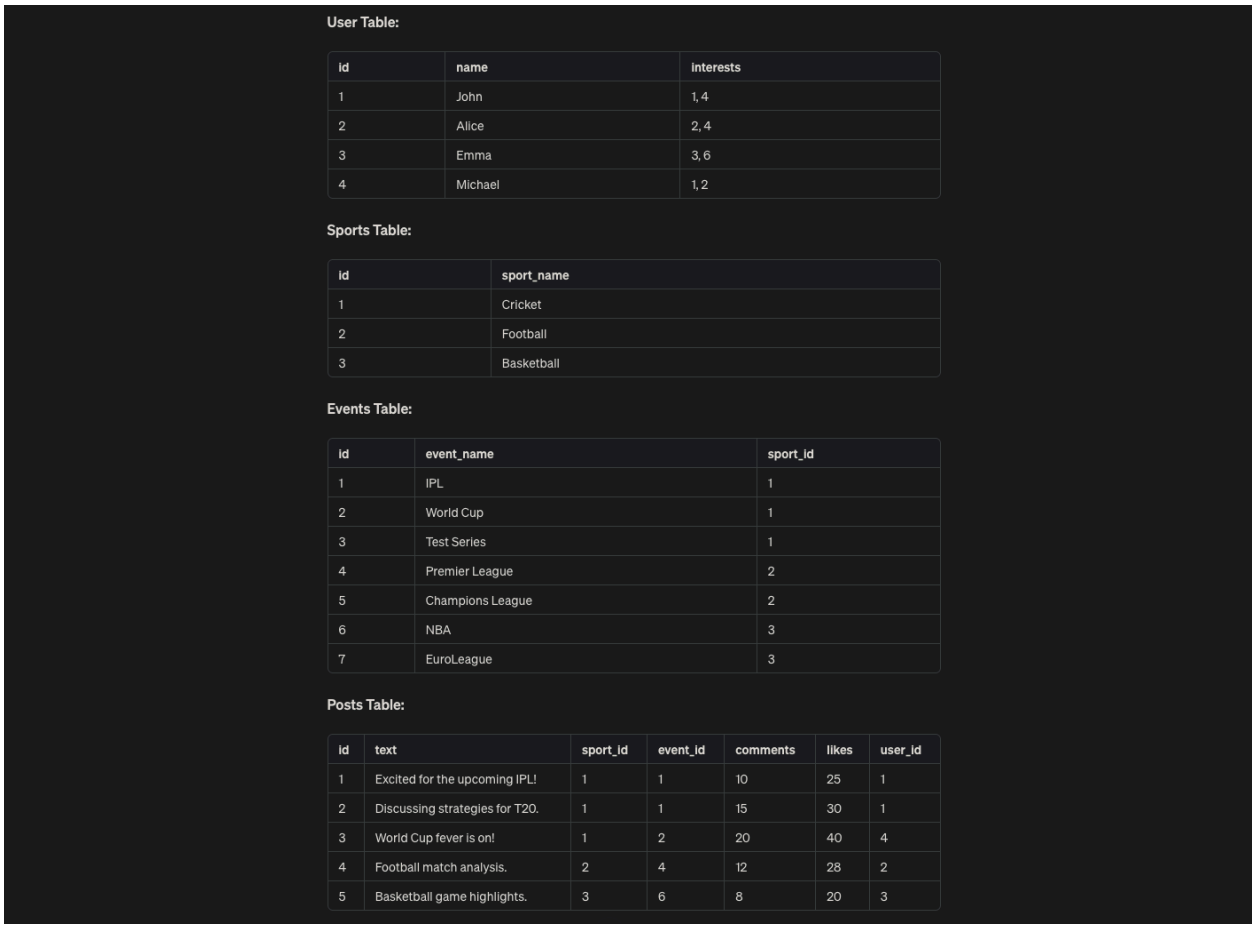
Task Duration: 3 days

Description

You are tasked with creating a recommendation system for a social media platform to recommend users to others based on their similar interests and popularity. The system should utilize data stored in the database, where each post has a sport and event associated with it, to infer user interests and calculate popularity based on engagement metrics.

Reference Data Details:

The provided database design tables for users, posts, sports, and events. Each post should reference a specific sport and event, allowing the system to infer user interests based on their posts.



User Table:

id	name	interests
1	John	1, 4
2	Alice	2, 4
3	Emma	3, 6
4	Michael	1, 2

Sports Table:

id	sport_name
1	Cricket
2	Football
3	Basketball

Events Table:

id	event_name	sport_id
1	IPL	1
2	World Cup	1
3	Test Series	1
4	Premier League	2
5	Champions League	2
6	NBA	3
7	EuroLeague	3

Posts Table:

id	text	sport_id	event_id	comments	likes	user_id
1	Excited for the upcoming IPL!	1	1	10	25	1
2	Discussing strategies for T20.	1	1	15	30	1
3	World Cup fever is on!	1	2	20	40	4
4	Football match analysis.	2	4	12	28	2
5	Basketball game highlights.	3	6	8	20	3

Task Details:

1. Generate Mock Data:

- Create a script or implement functionality to generate thousands of mock data rows in the posts table.
- The mock data should include random text for the posts, along with random values for sport_id, event_id, comments, and likes.
- Ensure that the generated mock data covers a variety of sports and events to simulate realistic user activity.
- Consider using libraries or utilities to efficiently generate large volumes of mock data.
- Integrate this script or functionality into your solution to populate the posts table with realistic data for testing and evaluation.

2. User Recommendation API:

- Endpoint: `/api/recommendations/users`
- Method: GET
- Description: Retrieve recommended users for a given user based on similar **interests and popularity**.
- Request Parameters:
    - `userId` (number): The ID of the user for whom recommendations are being retrieved.
- Response:
    - An array of recommended user objects, each containing the following fields:
        - `id` (number): The ID of the recommended user.
        - `name` (string): The name of the recommended user.
        - `interests` (array): An array of inferred interests based on the sports and events of the posts made by the recommended user.
        - `popularityScore` (number): The popularity score of the recommended user, calculated based on engagement metrics such as the number of comments, likes, etc. on their posts.

3. Data Analysis and Inference:

- Utilize the sport and event fields of the posts stored in the database to infer user interests.
- Map the sports and events to user interests and store them in the user profile.

4. Popularity Score Calculation:

- Calculate the popularity score of each user based on engagement metrics such as the number of comments, likes, etc. on their posts.

- Define a formula or algorithm to aggregate these engagement metrics into a single popularity score for each user.

5. Cache Management:

- Implement a caching mechanism to store recommended users data.
- The cache should be refreshed periodically, ideally every 15 minutes, to ensure the recommendations are up-to-date.
- Use an efficient caching strategy to minimize the need for repeated data analysis and improve response times.

Technical Requirements:

- Implement the recommendation API using **Node.js and Express.js**.
- Utilize data stored in the database, where each post has a sport and event associated with it, to infer user interests.
- Calculate the popularity score of each user based on engagement metrics stored in the database.
- Implement a periodic cache refresh mechanism to update recommendations every 15 minutes.
- Use any database of your choice to store user data, posts, sports, and events.
- Ensure efficient handling of database queries to minimize resource usage.
- Dockerize the solution to containerize the application.
- Deploy the Dockerized solution to a free hosting service such as Heroku or AWS Free Tier.
- Write end-to-end tests to validate the functionality of the recommendation API.
- Use Git for version control and make frequent commits to demonstrate the evolution of your solution.
- Include instructions on how to build the Docker image, run the container locally, and deploy to the hosting service.

Additional Notes:

- The mock data generation script should adhere to the schema of the posts table, including non-null constraints and foreign key references to the sports and events tables.
- Generate a sufficient amount of mock data to adequately stress-test the recommendation system and evaluate its performance under realistic conditions.
- Verify that the generated mock data accurately reflects the expected distribution of posts across different sports and events.
- Use efficient database insert operations to optimize the data generation process and minimize execution time.
- Document the mock data generation process and any relevant parameters or configurations used in the script or functionality.
- Focus on clean code practices, efficiency, and scalability in your implementation.

- Document any design/architecture decisions and potential improvements in a separate file.
- If you have any questions or need clarification on the task requirements, feel free to reach out.

Submission & Deliverables:

Once finished, kindly deploy the application on Netlify and share the URL with us. Upload the code on GitHub and provide us with the repository link. Additionally provide necessary documentation mentioned in the task.

Task Duration: 3 days