Backend Developer Assignment - Polling System with Kafka, Zookeeper & Real-Time Features

---

Objective:

Build a backend system that supports a high-concurrency polling feature using **Kafka** and **Zookeeper**. The system should allow multiple users to interact with polls simultaneously, ensure resiliency in case of failures, and include **real-time poll updates** using WebSockets along with a **leaderboard feature**. The system must ensure no votes are lost, even in the event of system failures.

Technology Stack:

- **Backend Framework:** Node.js
- **Message Broker:** Kafka (with Zookeeper
- **Database:** PostgreSQL
- **Real-Time Updates:** WebSockets
- **Containerization:** Docker (optional)

Key Functionalities:

1. **Poll Creation:**
    - Users should be able to create polls with multiple options.
    - Polls should be stored in the database.
2. **Poll Participation:**
    - Users can vote on polls, and each vote should be sent to **Kafka** for processing and stored in the database.
3. **Real-Time Poll Updates:**
    - Implement a **WebSocket server** that pushes real-time updates to connected users whenever a new vote is registered.
    - Users should be able to see dynamic poll results without having to refresh the page.
4. **Leaderboard Feature:**
    - Implement a **global leaderboard** that ranks the most popular poll options in real-time.
    - This leaderboard should update dynamically as votes are processed, showing the top poll options across different polls.
5. **Concurrency and Fault Tolerance:**
    - Use **Kafka and Zookeeper** to handle high concurrency and ensure resilience.
    - The system should ensure that no vote is lost in case of Kafka broker failures by leveraging Zookeeper for managing brokers.

Guidelines:

1. **Set Up Kafka with Zookeeper:**
    - Set up Kafka with Zookeeper to manage broker configurations and ensure fault tolerance.
    - Optionally provide Docker setup instructions for Kafka, Zookeeper, and the backend server.
2. **Develop Polling API:**
    - **Create Poll API:** Users should be able to create polls with options (`POST /polls`).
    - **Vote API:** Users can vote on polls (`POST /polls/{id}/vote`), and votes should be sent to Kafka for processing.
    - **Poll Results API:** Users can retrieve the current results for a poll (`GET /polls/{id}`).
3. **Real-Time Poll Updates with WebSockets:**
    - Implement **WebSocket functionality** to send real-time updates to users connected to a poll.
    - Every time a vote is processed by a Kafka consumer, the system should push the updated poll standings to connected clients via WebSockets.
4. **Implement Kafka Producers and Consumers:**
    - **Producer:** The voting API sends votes as messages to Kafka.
    - **Consumer:** A Kafka consumer processes these messages and updates the poll counts in the database.
    - Use **partitions in Kafka** to handle concurrency, ensuring votes are processed accurately.
5. **Leaderboard Feature:**
    - Implement a leaderboard API (`GET /leaderboard`) that ranks the top poll options across all active polls.
    - The leaderboard should also update in real-time, leveraging WebSockets to broadcast any changes in ranking as new votes come in.
6. **Concurrency and Failover Handling:**
    - Demonstrate how the system handles concurrent voting with Kafka partitions.
    - Ensure that the system remains operational and processes votes even if a Kafka broker fails, using Zookeeper for broker management and leader election.

Submission Guidelines:

- **Submit the full source code in a GitHub repository and share it with “chauhan-rakesh”..**
- **Include a README file with setup instructions for Kafka, Zookeeper, the backend, and the WebSocket setup.**
- **Provide clear instructions on how to run and test the real-time poll updates and leaderboard feature.**

Bonus Tasks:

1. **Dockerization:**
    - Provide a Docker setup for Kafka, Zookeeper, and the backend service to make the setup easier.

Additional Considerations:

- Implement proper error handling for scenarios where the specified poll does not exist.
- Ensure that the API response format clearly presents the requested data.

Note:

If you encounter any difficulties or have questions regarding the tasks or requirements at any step of the development process, please don't hesitate to reach out for clarification and assistance. You can send your doubts and questions to **rakesh@pollpe.in** with the subject line **"Backend-Task-Doubt."**