Take-Home Project: Event-Driven Inventory Management System

Project Overview

Design and implement an event-driven inventory management system for a large retail store. The system should handle inventory updates in real-time, ensuring that the stock levels are always accurate. The project should focus on backend functionality, scalability, and reliability.

Key Features

1. Inventory Management:
   - Implement CRUD operations for products (create, read, update, delete).
   - Track stock levels in real-time.

2. Event Handling:
   - Use an event-driven architecture to handle inventory changes.
   - Implement producers and consumers for inventory events.

3. Real-Time Updates:
   - Ensure that stock levels are updated in real-time.
   - Notify relevant systems or components when stock levels change.

4. Reporting:
   - Generate real-time reports on inventory status.
   - Provide endpoints for querying current stock levels and inventory history.

5. Scalability and Performance:
   - Design the system to handle high volumes of transactions.
   - Optimize database queries and minimize latency.

6. Persistence and Storage:
   - Use a combination of SQL (e.g., PostgreSQL) and NoSQL (e.g., MongoDB) databases to store inventory data and events.

7. API Documentation:
   - Document the API using Swagger or a similar tool.

8. Testing:
   - Write unit and integration tests to ensure code quality and reliability.
   - Include tests for event handling and real-time updates.

Technical Stack

- Languages: Java
- Frameworks: Spring Boot (for Java/Kotlin)

- Event Handling: Apache Kafka
- Databases: PostgreSQL (for relational data), MongoDB (for event storage)
- Containerization: Docker
- Orchestration: Kubernetes (optional for advanced deployment)
- CI/CD: Jenkins or GitHub Actions

Project Structure

1. Product Service:
   - Implement CRUD operations for products.
   - Track stock levels for each product.

2. Event Service:
   - Implement event producers to publish inventory events.
   - Implement event consumers to handle inventory events.

3. Inventory Service:
   - Update stock levels based on inventory events.
   - Ensure consistency and accuracy of stock data.

4. Reporting Service:
   - Generate real-time reports on inventory status.
   - Provide endpoints for querying current stock levels and inventory history.

5. API Documentation:
   - Use Swagger to document the API.

6. Testing:
   - Write unit and integration tests.
   - Test event handling and real-time updates.

Deliverables

1. Source Code:
   - Provide a GitHub repository with well-documented source code.

2. API Documentation:
   - Include Swagger documentation or similar.

3. Deployment Instructions:
   - Provide Dockerfiles and Kubernetes manifests (if applicable).
   - Include instructions for setting up and running the project locally.

4. Test Cases:

   - Include test cases and instructions for running them.

5. README:
   - A comprehensive README file explaining the project, how to set it up, and how to use it.

Evaluation Criteria

1. Functionality:
   - The project meets all specified requirements.

2. Code Quality:
   - Code is clean, well-documented, and follows best practices.

3. Scalability:
   - The system handles a high volume of transactions.

4. Performance:
   - Real-time inventory updates work with minimal latency.

5. Testing:
   - Comprehensive tests are provided, covering key functionalities.

6. Documentation:
   - API and project documentation are clear and complete.

---

Good Luck and GodSpeed.

We are rooting for you!