Project: "Order Processing System"

Overview

Develop a **microservices-based** order processing system where:

- **Frontend (Angular):** Allows users to create and view orders.
- **Backend (Spring Boot + gRPC):** Handles order management and communicates between microservices.
- **Docker:** Containers for all services.
- **Git:** Version control.

Project Requirements

1. Microservices Architecture

- Create **two Spring Boot microservices**:
    1. **Order Service:** Handles order creation and retrieval.
    2. **Inventory Service:** Manages product stock levels.
- Communication between microservices **must use gRPC** instead of REST.

2. Angular Frontend

- Simple UI with:
    - A **form** to create orders (select product, quantity).
    - A **table** to display orders.
- Use **Angular services** to call the backend via HTTP.

3. gRPC Communication

- Define a **protobuf file** (`order.proto`) for Order & Inventory services.
- Implement **gRPC client** and **server** communication between microservices.

4. Docker Integration

- Write a `Dockerfile` for each microservice.
- Create a `docker-compose.yml` to orchestrate services.

5. Git Best Practices

- Use GitHub/GitLab repository.
- Require **at least one pull request (PR)** before merging code.
- Write clear commit messages.

Project Breakdown (Tasks & Expected Outcome)

Task 1: Setup Order Service (Spring Boot + gRPC)

- Create a Spring Boot app with:
    - `Order` entity (ID, product, quantity, status).
    - gRPC server to process order requests.

✅ **Expected Outcome:**

- API to create an order (`POST /orders`).
- gRPC service to communicate with Inventory Service.

Task 2: Setup Inventory Service (Spring Boot + gRPC)

- Create a Spring Boot app with:
    - `Product` entity (ID, name, stock quantity).
    - gRPC server to check stock availability.

✅ **Expected Outcome:**

- gRPC method to check & update stock levels.

Task 3: Implement gRPC Communication

- Define `order.proto` file:
  proto

```
syntax = "proto3";

service OrderService {
  rpc CreateOrder(OrderRequest) returns (OrderResponse);
}

message OrderRequest {
  string product = 1;
  int32 quantity = 2;
}

message OrderResponse {
  string status = 1;
}
```

- Implement gRPC client in **Order Service** to talk to **Inventory Service**.

✅ **Expected Outcome:**

- Order Service checks stock via gRPC before creating an order.

Task 4: Develop Angular Frontend

- Create a simple **Angular app** with:
  - **Order form** (Dropdown: Products, Input: Quantity, Button: Submit).
  - **Order list** (Displays existing orders).

✅ **Expected Outcome:**

- UI that interacts with backend via HTTP API.

Task 5: Dockerize the Services

- Write `Dockerfile` for Order Service & Inventory Service.

- Create `docker-compose.yml` to start both services.

✅ **Expected Outcome:**

- Running the command `docker-compose up` starts all services.

Task 6: Version Control & Collaboration

- Use Git for source control.
- Developers must:
  - Work on feature branches.
  - Use **PRs for code reviews**.
  - Write **clear commit messages**.

✅ **Expected Outcome:**

- Organized GitHub repository with a structured commit history.

Evaluation Criteria

Skill	What to Look For
Java Spring Boot	Clean architecture, good use of gRPC, REST API.
Angular	Proper component structure, form validation, API calls.
gRPC	Correct usage of `.proto` files & inter-service communication.
Microservices	Decoupled services with clear responsibility.
Docker	Working `Dockerfile` and `docker-compose.yml`.
Git	Good commit history, feature branching, PRs before merging.

You have 5 working days from when you get the assignment to complete this tasks
Additional features like websocket for real time update, unit test will give you competitive edge