RESTful API for E-Commerce Platform

Objective:
Develop a **production-grade** RESTful API for a simple e-commerce platform. This platform should allow users to view available products, add new products, and place orders. Your solution should be production grade and must include **exception handling**, **comprehensive test cases**, and a **dockerized container** for easy deployment.

Requirements

Endpoints

1. **GET /products**
   - Retrieve a list of all available products.
2. **POST /products**
   - Add a new product to the platform. Each product should have an ID, name, description, price, and stock quantity.
3. **POST /orders**
   - Place an order for a list of selected products, with validation to ensure sufficient stock is available for each product.

Data Models

Define the following data models:

1. **Product**
   - Fields: `id` (integer, unique), `name` (string), `description` (string), `price` (float), `stock` (integer)
2. **Order**
   - Fields: `id` (integer, unique), `products` (list of product IDs and quantities), `total_price` (float), `status` (string, with possible values: `pending`, `completed`)

Business Logic & Constraints

- **Stock Management:**
  - When an order is placed, validate the stock levels for each product in the order.
  - Deduct the appropriate quantities from stock if the order is successful.
- **Order Validation:**

- Ensure sufficient stock is available before confirming an order. If stock is insufficient, return an appropriate error response.

Expected Solution

1. **Production-Grade Code**
   - Write clean, modular code adhering to best practices for production environments.
   - Implement **exception handling** to capture and manage errors gracefully. Return meaningful error responses for issues like insufficient stock or invalid data.
2. **Testing**
   - Write **comprehensive test cases** for each endpoint, covering both successful scenarios and edge cases (e.g., handling insufficient stock when placing an order).
   - Include **unit tests** for individual functions and **integration tests** for API endpoint behavior.
3. **Dockerization**
   - Dockerize the solution with a `Dockerfile` that builds and runs the application in a containerized environment.
   - Optimize the container configuration for production, including any necessary environment variables, network configurations, and dependencies.
4. **Documentation**
   - Include a **README** file with clear instructions on how to build, run, and test the application within the Dockerized environment.

Submission Requirements

1. **Source Code**
   - Submit all source code, organized for clarity and maintainability, including the Dockerfile in a github repo.
2. **Documentation**
   - Include a README with instructions for building, running, and testing the application, as well as environment configuration details.