Task: Design an Order and Product Management System

Objective: The goal is to design and implement a system that manages products and orders using CQRS and event-driven architecture. The system should be loosely coupled even if the monolith architecture is chosen. Your focus should be on how the system is designed, with implementation being secondary.

Task Requirements:

1. **Architecture:**
    a. Design either a **monolithic** or **microservices** architecture with two main services:
        i. **Order Service**
        ii. **Product Service**
    b. The system should ensure that services are not tightly coupled even if a monolithic design is chosen.
    c. The architecture should support **event-driven communication** between services using **RabbitMQ**.
    d. Implement **CQRS (Command Query Responsibility Segregation)** to separate read and write operations.
2. **Services:**
    a. **Product Service**:
        i. Manages the product catalogue.
        ii. Read requests: 20k requests per day.
        iii. Write requests: 1k requests per day.
    b. **Order Service**:
        i. Manages customer orders.
        ii. Read and write requests: 1k requests per day.
    c. Services should use **RabbitMQ** for communicating **events** (e.g., when a new order is placed or when a product is updated).
3. **Database:**
    a. Use **MSSQL** as the primary database.
    b. Each service (if using microservices) should have its own database schema (for data isolation).
    c. Implement proper indexing and performance considerations, especially for the high-read operations in the Product Service.
4. **Core Functionalities:**

    a. **Product Service**:
        i. Add new products (write).
        ii. Update product details (write).
        iii. Fetch product details (read).
        iv. Fetch a list of products (read).
    b. **Order Service**:
        i. Place an order (write).
        ii. Update order status (write).
        iii. Fetch order details (read).
        iv. Fetch a list of orders (read).
5. **Event-Driven Architecture**:
    a. When an **order is placed**, an event should be published (using RabbitMQ) to inform the Product Service to update inventory.
    b. When a **product is updated**, an event should be published to inform other systems (e.g., for caching or analytics).
6. **Testing:**
    a. Write **unit tests** for core functionalities in both services.
    b. If possible, include basic **integration tests** that simulate real-world scenarios (e.g., an order is placed, and the product stock is updated).
    c. Ensure your tests follow best practices and provide adequate code coverage.
7. **Non-Functional Requirements:**
    a. Ensure the system is designed to handle the given load (20k reads, 1k writes for Product; 1k reads/writes for Order).
    b. Consider scalability, especially for the Product Service due to the high read volume.
    c. Document your design decisions and explain how the architecture would scale if the request load increases.

Evaluation Criteria:

- **System Design**: The candidate's ability to design a system that meets the given requirements while maintaining loose coupling.
- **CQRS and Event-Driven Implementation**: How well the candidate implements CQRS and integrates event-driven communication using RabbitMQ.
- **Code Quality**: The readability, structure, and organization of the code, including testing practices.

- **Scalability Considerations**: How the candidate plans for system scalability given the load.
- **Testing**: The completeness and quality of the unit and integration tests.