Build a Simple Inventory Management System

Objective:

Create a RESTful API for managing an inventory of products. The API should support basic CRUD operations and include some business logic for managing stock levels and creating orders.

Requirements:

- ### Setup:

  1. Use Node.js and Express.js to build the API.
  2. Use a database (e.g., lowdb, mongodb) for data persistence.
  3. Use CQRS pattern.
  4. Include appropriate environment configuration using .env files if necessary.

- ### API:

1. Products**:**

   a. Add new endpoint
      GET /products: Retrieve a list of all products.
   b. Add new endpoint
      POST */products*: Create a new product (fields: name, description, price, stock; all required, max length 50).

2. Stock Management:

   a. Add a new endpoint
      POST /products/:id/restock: to increase the stock level of a product.
   b. Add a new endpoint
      POST /products/:id/sell: to decrease the stock level of a product. Ensure the stock cannot go below zero.

3. Price Validation:

   a. Ensure that the price of a product is always positive.

4. Order Management

   a. Add endpoints to manage orders:
      POST /orders: Create a new order (fields: customerId, products).
   b. Implement logic to update stock levels when an order is placed.
   c. Prevent orders from being placed if there is insufficient stock.

d. Calculate the final order value based on applicable discounts and pricing rules:

   i. Volume-based discount

      - If a customer purchases 5 or more units, apply a 10% discount.
      - If a customer purchases 10 or more units, apply a 20% discount.
      - If a customer purchases 50 or more units, apply a 30% discount.

   ii. Seasonal & promotional discounts

      - Black Friday Sale: Apply a 25% discount on all products.
      - Holiday Sales: Apply a 15% discount on selected product categories (choose two).

   For simplicity, use bank holidays in Poland as the reference dates.

   iii. Location-based pricing

      - United States (US): Standard pricing.
      - Europe: Prices increased by 15% due to VAT.
      - Asia: Prices reduced by 5% due to lower logistics costs.

   Location does not have to be fetched dynamically; it can come from Customer entity.

   iv. Discount application rules

      - Discounts cannot be combined.
      - Apply only the highest applicable discount from the customer's perspective.

• Validation:

1. Implement input validation for all endpoints using a library like Joi or express-validator.

• Error Handling:

1. Implement proper error handling for cases such as invalid input, resource not found, and server errors.
2. Return appropriate HTTP status codes and error messages.

- ## Testing

1. Write unit and integration tests for crucial parts of application

- ## Notes / Assumptions

The candidate must provide a NOTES.md file or a dedicated section in README.md covering the points below:

1. Assumptions & Simplifications
   a. Key assumptions made during implementation.
   b. Elements that were intentionally omitted and the reasoning behind them.
   c. Interpretation of ambiguous parts of the task (e.g. discount rules, customer/location model).

2. Technical Decisions
   a. Justification for:
      - database choice,
      - project structure,
      - CQRS implementation approach.

   b. Brief explanation of command / query separation.

3. Business Logic
   a. How the discount system works (priority, order of application).
   b. How stock consistency is ensured (no negative stock).
   c. Key edge cases that were taken into account.

4. Testing
   a. What is covered by tests and why.
   b. What is not covered, but would be required in a production system.

- ## Trade-offs & Alternatives describing:

1. One concrete design decision you made that you would change if you had more time.
2. One alternative solution you seriously considered but rejected.
3. Why the chosen solution was selected over the alternative, including its downsides.

This section must:

- Refer to specific parts of your implementation (files, modules, or flows).
- Describe real compromises, not theoretical best practices

- ## Submission Guidelines:

1. Provide a link to a GitHub repository containing the code with pr to main branch.

2. Ensure the project is well-structured and follows best practices for Node.js development.