MACHINE TEST - L1

==========

E-commerce Application

Objective:
Build a robust E-commerce Backend API using .NET Core to manage products, customers, orders, and authentication with admin management.

Functional Requirements:

1. Entities
  i. User / Customer:
- Id, Name, Email, Password (hashed), Role (Admin / Customer), CreatedDate

 ii. Product:
- Id, Name, Description, Price,categoryId, StockQuantity, IsActive, CreatedDate, UpdatedDate

 iii. Order:
- Id, CustomerId, List of OrderItems (ProductId, Quantity, PriceAtOrder), OrderDate, TotalAmount, Status (Pending, Completed, Cancelled)

 iv. Category:
- Id, Name, Description, CreatedDate, UpdatedDate

2. Authentication & Authorization
- JWT-based authentication
- Roles:
  - Admin: Can manage products, categories, and view all orders
  - Customer: Can place orders, view order history
- Endpoints:
  - POST /api/auth/register → Customer registration
  - POST /api/auth/login → Login and receive JWT
  - POST /api/auth/refresh-token → Refresh JWT token

3. Product & Category Management (Admin APIs)
- CRUD for Products (/api/admin/products)
- CRUD for Categories (/api/admin/categories)
- Validation rules:
  - Prevent creating products with duplicate names in the same category
  - Stock quantity cannot be negative
- Soft delete for products (mark as inactive rather than permanent deletion)

4. Customer APIs

- View/update profile (/api/customers/me)
- Search products (/api/products/search?name=shirt)
- View product list and details (/api/products)
- Place order (POST /api/orders)
  - Validate stock availability
  - Calculate total amount
  - Reduce stock quantity automatically
- Cancel order (POST /api/orders/{id}/cancel)
  - Only pending orders can be cancelled
  - Restore stock quantities

5. Admin Order Management APIs
- View all orders (/api/admin/orders)
- Update order status (/api/admin/orders/{id}/status) → Pending / Completed / Cancelled
- View order details including customer information

6. Additional Functional Requirements
- Validation:
  - Email format validation
  - Password strength (minimum length, alphanumeric)
  - Prevent ordering products with insufficient stock
- Audit Fields:
  - CreatedDate, UpdatedDate, DeletedDate (for soft delete)

7. Bonus Features
- Async database operations using async/await
- In-memory caching for products and categories
- Pagination and sorting for product and order lists
- Logging using Serilog or NLog
- Optional: Email notifications on order placement or cancellation

8. Tech Stack Recommendations
- Backend: .NET Core Web API
- Database: SQL Server
- Authentication: JWT
- ORM: Entity Framework Core
- Optional: AutoMapper for DTOs

9. Submission Requirements
- Submit your solution as a GitHub repository
- Include README with:
  - Setup instructions (database migrations, seeding sample data)
  - Sample API requests
  - Short demo video showing API execution

This test evaluates:
- Backend API design and best practices
- Authentication & Authorization
- CRUD operations with validation
- Soft delete and stock management
- Admin vs customer role management
- Optional: async operations, caching, logging

Submission:
Please submit your completed solution as a GitHub repository link. Additionally, include a README file with setup instructions and a short recording demonstrating your code setup and execution process and send to tech.codeandcraft@gmail.com.