**Online Shopping System Project Guide**

**1. Project Overview**

**Goal:**  
Design and implement a modern, scalable online shopping system, focusing on best practices for REST APIs and microservices with .NET and C#. The primary objective is to enable users to browse products, manage shopping carts, place orders, and handle user authentication in a modular, maintainable application.

**Scope:**

* User registration, authentication, and account management
* Product catalog with categories, search, and filtering
* Shopping cart and order management
* Checkout and basic payment simulation
* Order history and tracking
* Admin functionality (for managing products/orders/users)  
  *Not included*: Third-party integrations, advanced analytics, real-world payment processing

**2. Project Goals and Objectives**

* Create a feature-complete RESTful API backend for the shopping system
* Apply microservices architecture to decompose the system into distinct services
* Ensure code readability, maintainability, and testability
* Enable containerized deployment (Docker compatibility; cloud-ready)
* Follow SOLID, Clean Code, and Clean Architecture principles

**SMART Goals**

* **Specific:** Deliver a working online shopping backend with modular microservices (user, product, cart, order)
* **Measurable:** Successful completion defined by passing integration tests for all API endpoints and core business flows
* **Achievable:** Minimum viable product (MVP) ready for manual testing within 6 weeks
* **Relevant:** Focus strictly on modular REST API design and microservices separation
* **Time-bound:** Target major milestones every two weeks

**3. Planned Architecture**

**Architecture Pattern:**

* Microservices-based architecture using .NET 8/9 (C#), Docker containers
* Clean Architecture separation: API, Application (Service), Domain, and Data layers for each microservice[learn.microsoft+3](https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/introduce-eshoponcontainers-reference-app)
* Multiple services:
  + **Product Service:** Handles product catalog CRUD, categories
  + **User Service:** Registration, authentication, profile management
  + **Cart Service:** Shopping cart management, internal user association
  + **Order Service:** Order placement, order tracking
  + **Admin Service (optional):** Manages users/products/orders
* Communication: RESTful HTTP between services, optionally messaging (e.g., using RabbitMQ or Azure Service Bus) for decoupled tasks
* API Gateway to route and secure requests
* Data Storage: Separate databases per microservice (e.g., SQL Server, MongoDB)

**Deployment:**

* Containerized microservices using Docker Compose (for local orchestration)
* Cloud-ready configuration but local focus for starter phase

**4. Functional Requirements**

* RESTful endpoints for all business entities and key actions
* JWT-based stateless authentication; role-based authorization
* Basic request validation and error handling
* OpenAPI/Swagger documentation for each service
* Automated unit and integration tests

**5. Non-Functional Constraints**

* Modular, extensible codebase (easy future feature expansion)
* Use standard .NET dependency injection & configuration conventions
* Keep external dependencies to a minimum
* Adhere to best practices for API error responses, logging, and security

**6. Assumptions & Out-of-Scope Features**

**Assumptions:**

* No live payment integration; simple payment simulation
* No real-time messaging/notifications out of MVP scope
* No advanced analytics, recommendations, third-party integrations

**Explicitly Excluded:**

* UI/Front-end development (unless simple views for testing)
* Social login, third-party carts, multi-tenant architecture

**7. Sample Project Structure**

text

/src

/ProductService

/UserService

/CartService

/OrderService

/AdminService (optional)

/gateway

/docker-compose.yml

/readme.md

Follow separation into API, Domain, Service, and Data folders/layers per microservice.[dev+1](https://dev.to/iamcymentho/building-a-scalable-furniture-e-commerce-web-api-using-net-clean-architecture-and-mongodb-33o7)

**8. Guidance for AI Assistants**

* *Do not* suggest or implement additional features not covered above (“feature creep”)
* Strictly follow the Clean Architecture and microservices separation as documented
* Use latest stable .NET Core (8 or 9) and containerization tools
* Prioritize maintainability, clarity, and direct mapping to described business use cases