# **ByteStore – Backend Project Documentation**

**Stack:** ASP.NET Core 8 Web API, EF Core 8, SQL Server (or PostgreSQL), Redis, Serilog, FluentValidation, Identity + JWT, Stripe (payments), Docker.

**Architecture:** Onion Architecture (Domain-centric). No CQRS. Service + Repository + Unit of Work. Minimal APIs (or Controllers) with Endpoint grouping.

**Repo layout (monorepo):**

/ByteStore

/src

/Core

/Domain

/Entities

/Enums

/Repositories

/Application

/Services

/DTOs

/Validators

/Mappings

/Exceptions

/External

/Api (ASP.NET Core host)

/Endpoints

/Extensions (DI, Auth, Swagger, HealthChecks)

appsettings.json

Program.cs

/Persistence (Infrastructure)

/Database (DbContext, Configurations)

/Migrations

/Repositories

/UnitOfWork

/Integration (Payment, Email)

/deploy

docker-compose.yml

k8s/ (optional)

/docs

ERD.png / API.postman\_collection.json

## **1) Executive Summary**

ByteStore backend is a modular eCommerce API with a clean domain model, split by concerns using Onion Architecture. The API exposes resources for catalog (products, categories, brands), customer (accounts, addresses, wishlists), shopping flow (cart, orders, payments, shipments), and admin (inventory, pricing, promotions).

## **2) Goals & Non‑Functional Requirements**

* **Scalability:** stateless API, cache-heavy (Redis), DB connection pooling, async I/O.
* **Security:** JWT, refresh tokens, HTTPS, role-based + policy-based authorization (Customer, Admin), data validation and rate limiting.
* **Reliability:** health checks, retries, idempotency keys on payments.
* **Observability:** structured logging (Serilog), request tracing, metrics, audit logs.
* **Performance:** pagination, filtering, sorting; EF compiled queries; output caching for catalog.
* **Maintainability:** SOLID, repository/specification pattern, fluent validation, unit/integration tests.
* **Portability:** Dockerized, environment-driven configuration.

## **3) Onion Architecture Overview**

* **Domain (Core):** business rules, entities, enums, repository interfaces. No external dependencies.
* **Application (Core):** use-case orchestration via services. Defines DTOs, mappings, and validation. Depends only on Domain.
* **Infrastructure (External → Persistence):** EF Core DbContext, repository implementations, integrations (Stripe, SendGrid, storage), Redis cache providers. Depends on Core.
* **API (External → Api):** composition root. Configures DI, middleware, endpoints, Swagger, auth, health checks.

**Request flow:** Api Endpoint → Application Service → Domain Entities (via Repositories & UnitOfWork) → DbContext. Cross-cutting: logging, validation, caching.

## **4) Domain Model**

**Key Entities**

* **User** – Roles: Customer, Admin.
* **CustomerProfile** – name, phone, defaultAddressId.
* **Address** – street, city, country, zip, isDefault.
* **Category** – hierarchical (ParentCategoryId), slug.
* **Brand** – name, slug.
* **Product** – name, slug, sku, description, brandId, categoryId, price, discountPrice, status, stockQuantity, ratingAvg, createdAt.
* **Cart** – customerId, CartItems (productId, qty, priceSnapshot).
* **Order** – orderNo, customerId, status (Pending, Paid, Shipped, Completed, Cancelled, Refunded), totals, shippingAddressId, paymentId, shipmentId.
* **OrderItem** – orderId, product ref, qty, unitPrice, discount.
* **Payment** – provider, amount, currency, status, intentId.
* **Shipment** – carrier, trackingNo, status.
* **Review** – productId, customerId, rating, comment, createdAt.
* **Wishlist** – customerId, items(list of productIds).
* **Promotion** (optional) – code, type (percentage/fixed/freeShip), rules, start/end.

**Aggregates**

* Product, Order, Cart.

**Invariants & Rules Examples**

* Cannot add to cart if stockQuantity < requestedQty.
* Order totals = Σ(items) − discounts + shipping + tax.
* Review allowed only if customer purchased the product.

## **5) Persistence & Data Modeling**

**EF Core** with Fluent configurations.

* Soft-delete via IsDeleted + global query filters.
* Decimal precision for Money fields.
* Indexes: Product(slug), Product(categoryId, brandId), Order(orderNo).

**Unit of Work**

* IUnitOfWork with SaveChangesAsync() wrapping DbContext transaction.

**Repositories**

* IProductRepository, ICartRepository, IOrderRepository, ICustomerRepository, IReviewRepository.
* Query via **Specification pattern** (e.g., ProductsByFiltersSpec).

**Caching**

* Redis for product lists, product details, categories tree.
* Cache invalidation on writes via events.

## **6) Application Layer**

**Services (Use-cases)**

* CatalogService: list/filter/paginate products, product details, categories/brands.
* CartService: create/get cart, add/update/remove item, merge anonymous → customer.
* OrderService: checkout, place order, apply promotion, calculate totals.
* PaymentService: create payment intent (Stripe), confirm, webhook handling.
* AccountService: register, login, manage addresses, tokens.
* ReviewService: add/update/remove review.
* AdminService: CRUD products, categories, brands, inventory movements.

**Validation** with FluentValidation on DTOs.

**Mapping** DTOs ⇄ Entities via AutoMapper/Mapster.

## **7) API Design (Minimal APIs or Controllers)**

**Conventions**

* RESTful resource endpoints under /api/v1/…
* JSON, UTC times, RFC 7807 ProblemDetails for errors.
* Pagination: pageNumber, pageSize; Sorting: sort; Filtering by query params.

**Endpoints (selected)**

* **Auth & Account**: register, login, refresh, account info, addresses.
* **Catalog**: products list/detail, categories tree, brands, reviews.
* **Cart**: get cart, add/update/remove items.
* **Orders**: place order, list orders, order details.
* **Payments**: payment intent, webhook.
* **Admin**: CRUD products/categories, manage inventory.

## **8) Security & Identity**

* ASP.NET Core Identity with JWT auth.
* Password hashing, email confirmation, lockout policy.
* **Roles:** Customer, Admin.
* Refresh tokens stored with rotation.
* Rate limiting on auth and product search endpoints.

## **9) Cross‑Cutting**

* **Global Exception Handling** → ProblemDetails mapper.
* **Validation Pipeline**.
* **Logging** → Serilog.
* **Caching** → Redis.
* **Email** → SendGrid/SMTP service abstraction.

## **10) Environment & Configuration**

* appsettings.json + environment-specific configs.
* **Required settings:**
  + ConnectionStrings:Default
  + Jwt:Issuer, Jwt:Audience, Jwt:Key
  + Redis:Connection
  + Stripe:ApiKey, Stripe:WebhookSecret
  + Serilog configuration

## **11) Build & Run**

**Backend**

cd src/External/Api

dotnet ef database update --project ../Persistence

dotnet runDocker Compose (/deploy/docker-compose.yml)

## **12) Testing Strategy**

* **Unit Tests** (Domain, Application): xUnit + FluentAssertions.
* **Integration Tests** (API): WebApplicationFactory, Testcontainers for SQL + Redis.
* **Contract Tests**: Verify DTOs/OpenAPI.
* **Load Tests**: k6 or JMeter.

## **13) Observability & Ops**

* **Health Checks:** /health, /health/ready.
* **Metrics:** /metrics (Prometheus).
* **Dashboards:** Grafana + Loki/Seq for logs.

## **14) Coding Guidelines**

* Naming: PascalCase for types, camelCase for locals.
* Controllers/Endpoints thin; logic in services.
* DTOs versioned under Contracts/V1.
* Use async/await everywhere.

## **15) Delivery Roadmap**

1. Foundation: skeleton, DI, EF, Identity, JWT, Serilog.
2. Catalog: products/categories/brands APIs.
3. Cart: anonymous cart, cookie, add/update/remove.
4. Account: register/login/addresses.
5. Checkout: order placement, Stripe intent, webhooks.
6. Admin: product/category management, inventory.
7. Reviews & Wishlist.
8. Caching & Performance.