**Documentation**

**Project**: ElectroStore – Backend

**Scope**: Product management, basket (cart), and deals/discounts for a web-based electronics store.

**1. Introduction**

**Purpose**

Provide a RESTful backend for an electronics store that manages products, baskets, and promotional deals.

**Objectives**

* Standards-compliant REST APIs.
* Clear, extensible design using Clean Architecture.
* Concurrency-safe operations under multi-user load.
* Atomic transactions with rollback on failure.

**2. Clean Architecture Overview**

**2.1 Principles**

* **Dependency Rule**: Inner layers do not depend on outer layers.
* **Separation of Concerns**: Controllers handle I/O; services orchestrate use cases; domain encapsulates business rules.
* **High Cohesion, Low Coupling**.

**2.2 Layers**

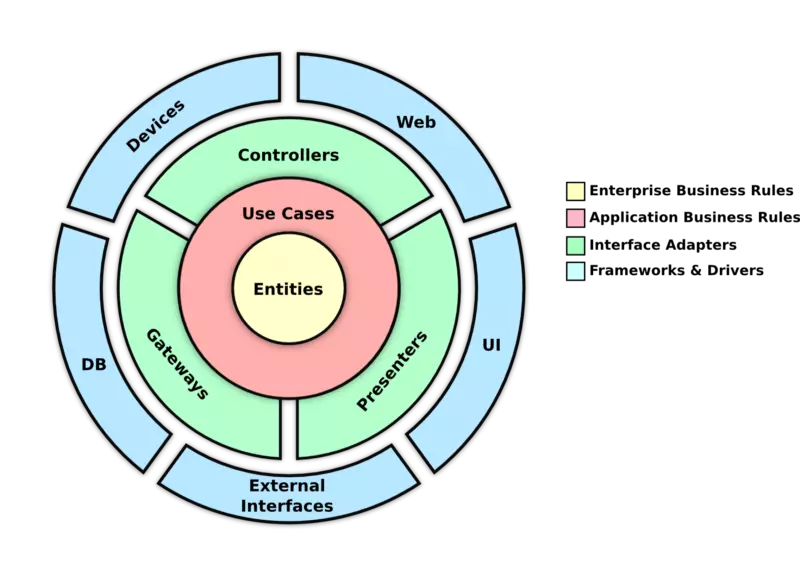
* **API Layer** (Controllers): accepts HTTP requests, validates input, maps DTOs -> domain models.
* **Application Layer** (Services / Use Cases): create product/deal, add to basket, compute receipt, orchestrates transactions and calls domain logic.
* **Domain Layer** (Entities & Business Rules): core entities: Product, Deal, Basket, BasketItem.
* **Infrastructure Layer** (Adapters): persistence (Repositories via JPA), database schema, configuration.

**Project Structure:**

| project  │ └───api // # Presentation - Web Controller │ └─── src │ │ └─── main  │ │ └─── test // # End to End test │ │ pom.**xml** **└─── application** // # Application services (use cases) │ └─── src │ │ pom.**xml** **└─── domain** // # Enterprise business rules - Entities/Aggregates │ └─── src │ │ pom.**xml** **└─── infra** // # Infrastructure │ └─── src │ │ pom.**xml** **pom**.xml |
| --- |

| api/... # Controllers, request/response DTOs application/... # Services, use **case** orchestration domain/... # Entities, value objects, domain services  infra/... # JPA repositories, mappers, DB config |
| --- |

**Diagrams**:



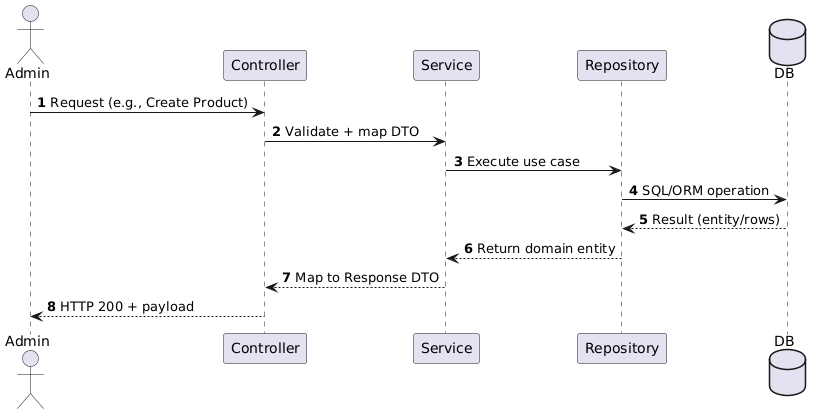
**3. System Flow Diagrams**

**3.1 Product Management (Admin) – Create Product**

**Scenario**: Admin creates a product via POST /api/admin/products.

**Flow**:

1. Controller validates request, maps to domain model.
2. Application service invokes domain validations.
3. Service calls repository to persist product.
4. Return created resource.

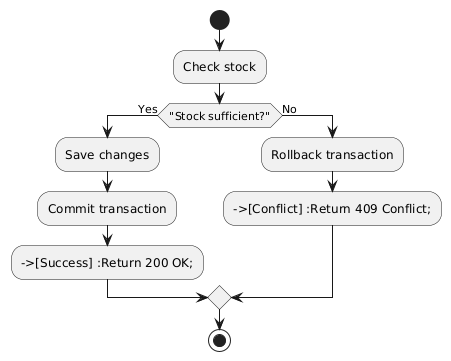


**3.2 Basket Management (Customer) – Add Item to Basket**

**Scenario**: Customer adds a product to a basket via *POST /api/baskets/{id}/items.*

**Flow**:

1. Service loads product & basket within a transaction.
2. Check stock.
3. **If stock sufficient**: create BasketItem, decrement Product.stock, commit.
4. **If insufficient**: rollback and return 409 Conflict.



### Concurrency Handling

### Pessimistic Locking: Main branch → lock product row when reducing stock.

### Optimistic Locking: In-progress branch → use version column + retry on conflict.

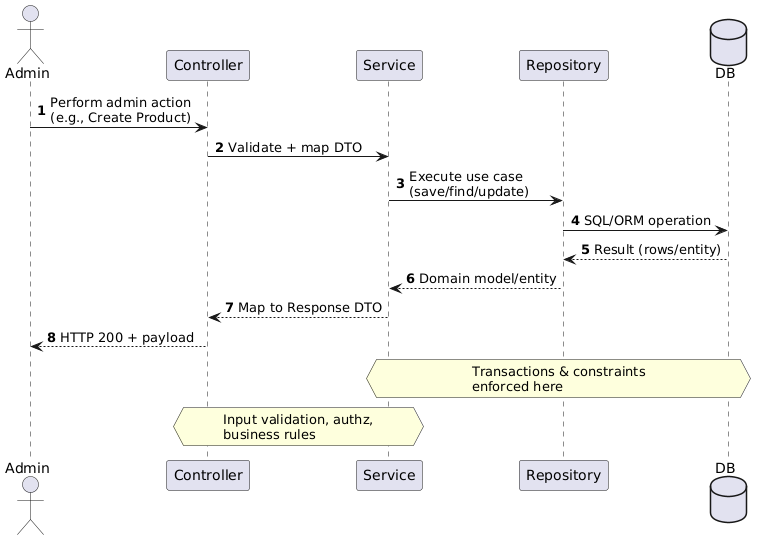
### Idempotency key for repeated client retries: To be handled later.

**3.3 Receipt Calculation**

**Scenario**: *GET /api/baskets/{id}/receipt.*

**Flow**:

1. Load basket + items.
2. DealEngine filters active (non-expired) deals.
3. Apply qualifying deals to items.
4. Compute line totals and grand total.
5. Return receipt JSON.



**4. Data Model**

**4.1 Entities (Fields)**

* **Product**: id, name, category, price, stock, available (boolean), createdAt, updatedAt
* **Deal**: id, productId (FK), dealType (B1G50\_2ND
* , THIRTY\_PERCENT\_OFF), description, expirationDateTime, createdAt, updatedAt
* **Basket**: id, userId, status (e.g., OPEN, CHECKED\_OUT), createdAt, updatedAt
* **BasketItem**: id, basketId (FK), productId (FK), quantity, createdAt, updatedAt

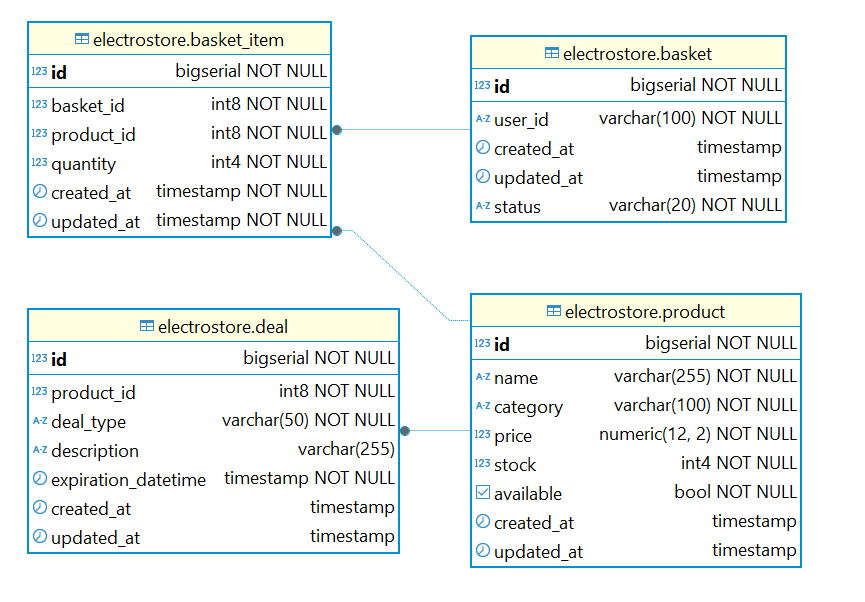
### 4.2 Relationships (ERD)

This project uses an **H2 in-memory** database

### **Product** 1 – N **BasketItem**

### **Basket** 1 – N **BasketItem**

### **Deal** N - 1 **Product** (each deal belongs to a product)



**4.3 Constraints & Validation**

* BasketItem.quantity > 0
* Product.stock >= 0 (enforced by business logic + DB checks if applied)
* Deal.expirationDateTime must be greater than **current time** for the deal to be valid

**5. API Design**

**5.1 Admin Endpoints**

| **Method** | **Path** | **Purpose** |
| --- | --- | --- |
| POST | /api/admin/products | Create product |
| DELETE | /api/admin/products/{id} | Remove product |
| GET | /api/admin/products?page={page}&size={size}&name={name}&category={category}&minPrice={minPrice}&maxPrice={maxPrice} | List/filter products (paginated) |
| GET | /api/admin/products/all | List all products (testing) |
| GET | /api/admin/products/{productId} | Get product by ID (testing) |
| POST | /api/admin/deals | Create deal |
| DELETE | /api/admin/deals/{dealId} | Delete deal |
| GET | /api/admin/deals?page={page}&size={size} | List all deals (paginated) |

**5.2 Customer Endpoints**

| **Method** | **Path** | **Purpose** |
| --- | --- | --- |
| GET | /api/customer/products Query: name,category,minPrice,maxPrice,available,page,size | Browse catalog with filters (paginated) |
| POST | /api/customer/basket/items | Add product to basket (decrement stock)  **Header**: X-Customer-ID |
| DELETE | /api/customer/basket/items/{productId} | Remove product from basket **Header**: X-Customer-ID **Path**: productId |
| GET | /api/customer/basket | View current basket **Header**: X-Customer-ID |
| GET | /api/customer/receipt | Generate receipt for current basket **Header**: X-Customer-ID |

**5.3 Error Model (standardized)**

* **409 Conflict** → Insufficient stock.
* **422 Unprocessable Entity** → Invalid/expired deal or business rule violation.
* **404 Not Found** → Resource missing.
* **400 Bad Request** → Validation errors.

**5.4 Pagination & Sorting**

* Query: page (0-based), size (default 10), sort (e.g., price,asc).
* Response envelope: { "content": [...], "page": 0, "size": 10, "totalElements": 123, "totalPages": 13 }.

**6. Run & Usage**

**Requirements**

* JDK 21
* Maven 3.6+
* Docker (optional)

There are 3 ways to run:

1. **Run (Maven, module api)**

| # Windows .\mvnw.cmd -f .\api\pom.**xml** **spring-boot**:run  # macOS/Linux ./mvnw -f ./api/pom.**xml** **spring-boot**:run |
| --- |

1. **Package & Docker (optional)**

| ./mvnw -DskipTests package  docker build -t electrostore-**app** . docker **run** --**rm** -p 8080:8080 electrostore-**app** |
| --- |

1. **Docker Compose**

| docker compose up -d --build |
| --- |

**Base URL**: [*http://localhost:8080*](http://localhost:8080)

**Test (run all tests in api)**

| ./mvnw -**pl** api **test** |
| --- |

**7. Concurrency & Transactions**

* **Atomicity**: Basket updates and stock decrements occur within a single DB transaction; on failure, rollback all changes.
* **Isolation**: Use pessimistic locking for hot rows (stock decrements) or optimistic locking with retry to prevent oversell.
* **Idempotency**: Optional idempotency keys for create/update endpoints to handle client retries safely.
* **Validation**: Re-read stock before commit or use DB constraints to ensure stock >= 0.