Excellent — you uploaded the **Staff Backend Software Engineer - JAVA test (Sales Order Management REST API)** document. From the requirements inside that PDF, your backend must expose a **set of REST endpoints** organized around two main resources:

🔹 1. Authentication Endpoints (`/api/v1/auth`)

These endpoints handle user authentication and token management.

HTTP	Endpoint	Role	Description
`POST`	`/api/v1/auth/register`	`ADMIN`	Create a new user (admin or customer). Encrypt password using `BCryptPasswordEncoder`.
`POST`	`/api/v1/auth/login`	Public	Authenticate with username/password. Returns a JWT token containing roles.
`GET`	`/api/v1/auth/me`	Any authenticated	Returns info about the currently logged-in user from JWT.
`POST`	`/api/v1/auth/refresh`	Any authenticated	Generates a new JWT from a valid refresh token.

🔹 2. Catalog Endpoints (`/api/v1/catalog`)

Catalog items (product list) with **price management**.

HTTP	Endpoint	Role	Description
`GET`	`/api/v1/catalog`	`USER`, `ADMIN`	Retrieve all catalog items. Supports pagination and sorting.
`GET`	`/api/v1/catalog/{id}`	`USER`, `ADMIN`	Retrieve a specific catalog item by ID.
`POST`	`/api/v1/catalog`	`ADMIN`	Create a new catalog item (name, price).
`PUT`	`/api/v1/catalog/{id}/price`	`ADMIN`	Update item price (use optimistic locking or `@Version`).
`DELETE`	`/api/v1/catalog/{id}`	`ADMIN`	Remove an item from catalog.

Business rule:
Prices in *existing orders* must stay fixed even if catalog prices are later updated.

🔹 3. Sales Order Endpoints (`/api/v1/orders`)

Handles creation, querying, and cancellation of customer orders.

HTTP	Endpoint	Role	Description
`POST`	`/api/v1/orders`	`USER`, `ADMIN`	Create a new order with: • `customerName` • `items: [ {catalogItemId, quantity} ]`
`GET`	`/api/v1/orders`	`USER`, `ADMIN`	List all orders. Supports pagination, sorting, and filters: `?creationStart=dd/MM/yyyy&creationEnd=dd/MM/yyyy&cancellationStart=&cancellationEnd=`
`GET`	`/api/v1/orders/{id}`	`USER`, `ADMIN`	Retrieve a specific order with all details: customer, items, totals, VAT, status.
`PUT`	`/api/v1/orders/{id}/cancel`	`ADMIN`	Cancel an existing order (set cancellationDate).
`DELETE`	`/api/v1/orders/{id}`	`ADMIN`	Permanently delete order (optional — usually disabled).

Important:
When an order is created, the item prices are **copied from the catalog** into the order's line items table (`order_items`). This preserves historical price integrity.

🔹 4. Admin / Utility Endpoints

HTTP	Endpoint	Role	Description
`GET`	`/api/v1/admin/users`	`ADMIN`	List all users and roles.
`GET`	`/api/v1/admin/health`	`ADMIN`	Simple health check / readiness endpoint.

🔹 Security Rules (Summary)

Resource	Access
`/api/v1/auth/**`	Permit all
`/h2-console/**`	Permit all (for dev only)
`/api/v1/catalog/**`	`GET` allowed to all authenticated; `POST`, `PUT`, `DELETE` restricted to `ADMIN`
`/api/v1/orders/**`	Authenticated users
`/api/v1/admin/**`	`ADMIN` only

🔹 DTO and Entity Layer Overview

- **CatalogItem**
  - `id`, `name`, `price`
- **Order**
  - `id`, `creationDate`, `cancellationDate`, `customerName`, `subtotal`, `vat`, `total`, `List<OrderItem>`
- **OrderItem**
  - `id`, `itemName`, `quantity`, `unitPrice`, `CatalogItem`
- **User**
  - `id`, `username`, `password`, `Set<Role>`
- **Role**
  - `id`, `name`

🔹 Validation Rules

- Quantity ≥ 1 (`@Min(1)`)
- Price > 0 (`@DecimalMin("0.0")`)
- Required fields (`@NotBlank`, `@NotNull`)
- Validation errors handled via `@ControllerAdvice` → return structured JSON error.

🔹 Next Steps for You

1. ✅ Implement `OrderController`, `OrderService`, and `OrderRepository` with methods matching above endpoints.
2. ✅ Fix your `SecurityConfig` to allow `GET /api/v1/catalog/**` for both roles (`ADMIN`, `USER`), and restrict modification endpoints only.
3. ✅ Add proper `JwtAuthenticationFilter` and `JwtTokenUtil` integration (you already have this nearly complete).
4. ✅ Add DTOs for order creation and response (avoid exposing entities).
5. ✅ Create Postman collection covering all endpoints.