Capstone Project: Building the Checkout and Order APIs

In this capstone project, you'll build the core of an e-commerce system: the **Checkout** and **Order APIs**. This is your opportunity to apply everything you've learned about authentication, authorization, persistence, and clean API design.

I encourage you to take an hour or two to work through the steps on your own. This will give you hands-on experience and help you identify gaps in your understanding.

Before You Start

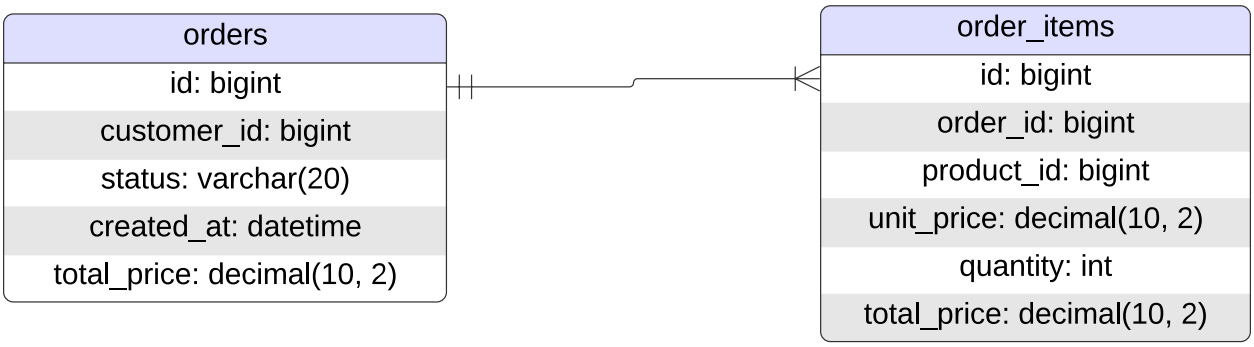
Before you start this project, I recommend increasing your access token expiration time from 15 minutes to 2 hours.

This makes it easier to build and test the feature without having to constantly re-login due to token expiration.

```
spring:
  jwt:
    accessTokenExpiration: 7200 # 2h
```

Step 1: Creating Database Tables

- Below is a database diagram that shows the structure of the tables you'll need for this project.
- Create the tables by writing SQL scripts.
- Apply these scripts using Flyway migrations, then run your application to ensure the tables are created.



Key Design Decisions

We store the total price in both the `order_items` and the `orders` tables because orders represent a historical record of what the user purchased at a specific time. Prices of products can change later—you don't want to re-calculate totals based on the current product prices. That would mess up reports, audits, and customer service interactions. Also, storing the totals improves performance: it avoids having to join tables and re-compute totals every time we need to show order history or generate invoices—something that becomes expensive at scale.

We store the order status as a `VARCHAR(20)` instead of creating a separate statuses table because the set of possible statuses (like PENDING, PAID, FAILED, CANCELLED) is small, fixed, and unlikely to change often. Creating a separate table adds unnecessary complexity without much benefit. Storing the status as a string directly makes the schema simpler and easier to work with, while still allowing you to map it to an enum in your code.

Step 2: Creating Entity Classes

- Now that we have our database tables, create entity classes that map to them.
- Use an `OrderStatus` enum (PENDING, PAID, FAILED, CANCELLED) for the `status` field in the `Order` entity.

Step 3: Checking Out

Example Request:

```
POST /checkout

{
  "cartId": "550e8400-e29b-41d4-a716-446655440000"
}
```

- Only logged-in users should be able to access this endpoint.

Example Response:

```
200 OK

{
  "orderId": 100
}
```

- If cart doesn't exist, return 400 Bad Request
- If cart is empty, return 400 Bad Request.
- Otherwise, create an order, save it, clear the cart, and return 200 OK.

Step 4: Getting All Orders

Example Request:

```
GET /orders
```

- Only logged-in users should be able to access this endpoint.
- Return a list of orders that belong to the currently authenticated user

Example Response:

```
[
    {
        "id": 1,
        "status": "PENDING",
        "createdAt": "2025-03-31T16:34:45",
        "items": [
            {
                "product": {
                    "id": 1,
                    "name": "Product 1",
                    "price": 10.00
                },
                "quantity": 2,
                "totalPrice": 20.00
            }
        ],
        "totalPrice": 20.00
    }
]
```

Step 5: Getting a Single Order

Example Request:

```
GET /orders/{orderId}
```

Example Response:

```
{
    "id": 1,
    "status": "PENDING",
    "createdAt": "2025-03-31T16:34:45",
     "items": [
        {
            "product": {
                "id": 1,
                "name": "Product 1",
                "price": 10.00
            },
            "quantity": 2,
            "totalPrice": 20.00
        }
    ],
    "totalPrice": 20.00
}
```

- If order doesn’t exist, return 404 Not Found.
- If order belongs to another user, return 403 Forbidden.
- Otherwise, return 200 OK with the order in the response.

Reset the Access Token Expiration

Once you're done, make sure to reset the access token expiration to 15 minutes.

```yaml
spring:
  jwt:
    accessTokenExpiration: 900 # 15m
```