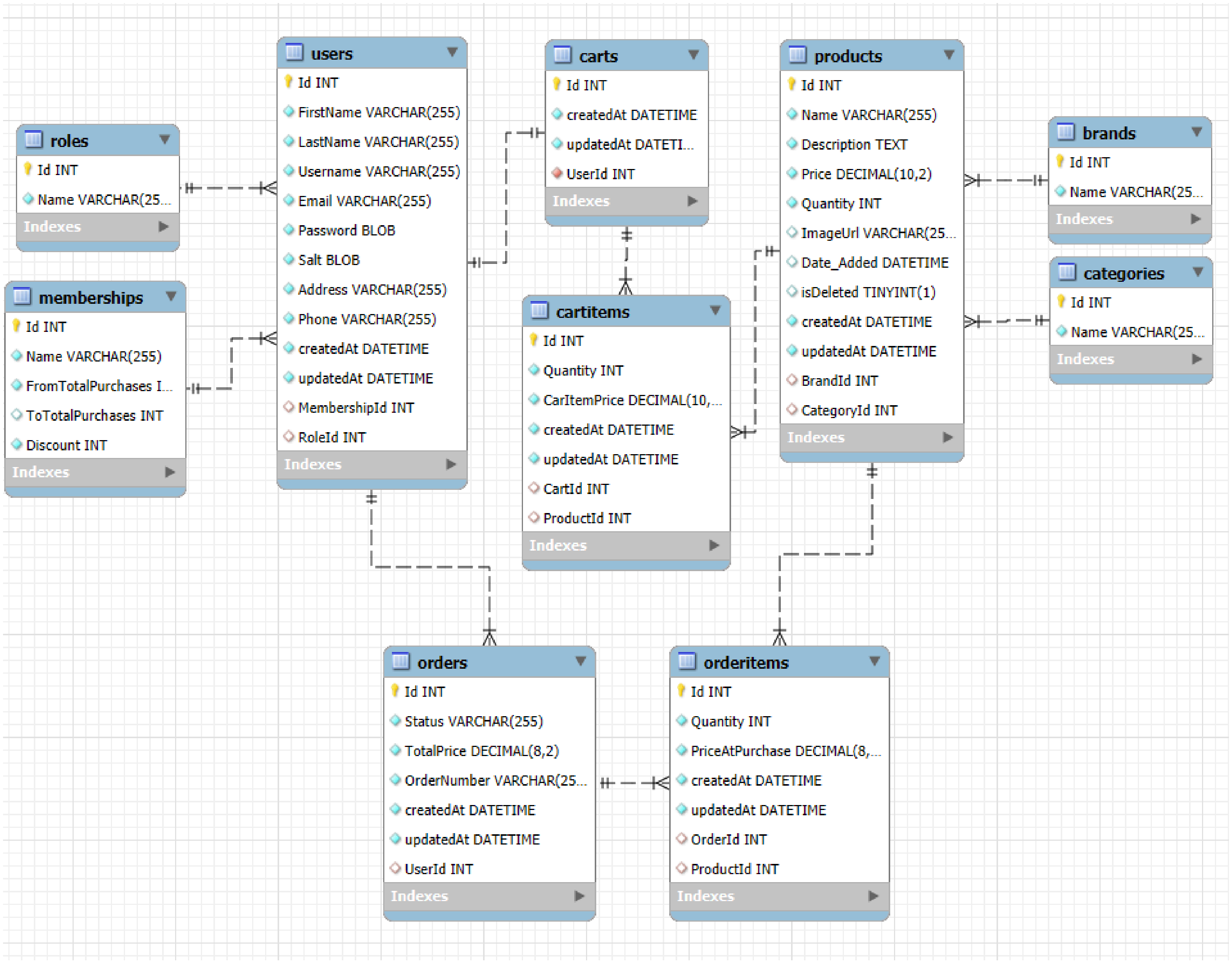
Reflection Report

JAN24FT EP1 Geir-Anders Haugen

Relationships



In the table roles, one role has many users, and a user belongs to one role, therefore this is a one-to-many relationship.

In the memberships table, one membership has many users, and a user belongs to one membership, therefore this is a one-to-many relationship.

The table users:

1. A user belongs to one role, but because many users can belong to the same role, this is a many-to-one relationship.
2. A user belongs to one membership, but because many users can belong to the same membership, this is also a many-to-one relationship.
3. A user has one cart, and a cart belongs to one user, therefore this is a one-to-one relationship.
4. A user can have many (has many) orders, and an order belongs to one user, therefore this is a one-to-many relationship.

The table carts:

1. A cart belongs to one user, and a user has one cart, therefore this is a one-to-one relationship.
2. A cart has many cart items, and a cart item belongs to one cart, therefore this is a one-to-many relationship.

The table cartitems:

1. A cart item belongs to a cart, and a cart has many cartitems, therefore this is a many-to-one relationship.
2. A cart item belongs to a product, and a product belongs to many carts through cart items, therefore this is a many-to-one relationship.

The table products:

1. A product belongs to many carts through cart item, and a cart item belongs to product, therefore this is a one-to-many relationship. However, many products can belong to many carts, meaning that the relationship between products and carts is many-to-many.
2. A product belongs to a brand, and a brand has many products, therefore this is a many-to-one relationship.
3. A product belongs to a category, and a category has many products, therefore this is a many-to-one relationship.
4. A product has many order items, and an order item belongs to products, therefore this is a one-to-many relationship.

In the tables brands/categories, a brand/categories has many products, and a product belongs to a brand/category, therefore they have a one-to-many relationship with products.

The table orders:

1. An order belongs to a user, and a user has many orders, therefore this is a many-to-one relationship.
2. An order has many order items, and an order item belongs to an order, therefore this is a one-to-many relationship.

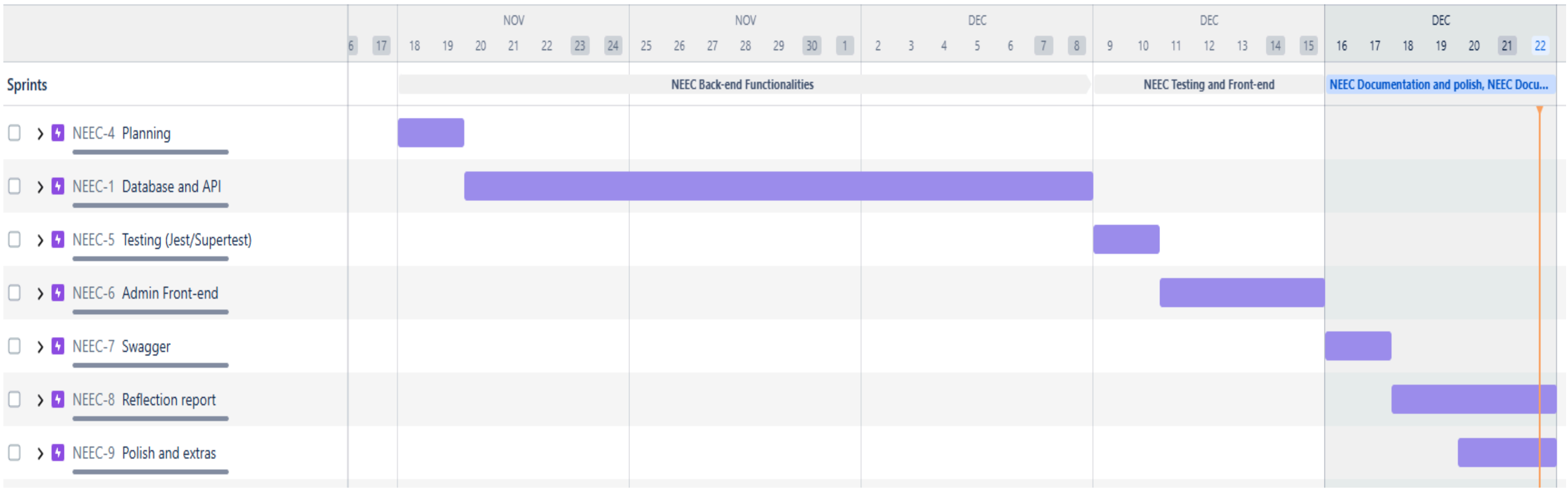
The table orderitems:

1. An order item belongs to an order, and an order has many order items, therefore this is a many-to-one relationship.
2. An order item belongs to a product, and a product has many order items, therefore this is also a many-to-one relationship.

Project Reflection Write-Up

Progression of the Project

The progression of this project was a big journey that involved consistent learning, debugging, and adapting to unexpected challenges that made this way tougher than anticipated. The project requirements were thoroughly analyzed over and over to understand the functional and technical needs.



(The original planned roadmap)

After planning, I started the project with database design, where creating a jira and an ERD sketch allowed me to visualize the progress and the relationships between tables like users, products, orders, and cart items. Setting up these relationships was an important first step as it dictated how the back-end logic would operate.

Moving to the back-end development phase I started with the initialization of the database as I considered this the core of the project. The development of the back-end progressed more smoothly than anticipated which was a great success in itself.

On the front-end, I created a simple, user-friendly admin interface while trying to make it look similar to the front-end example. Features like product addition and displaying product data required multiple iterations to ensure alignment with the back-end. Debugging integration issues between the back-end and front-end was a recurring task, but these moments deepened my understanding of API communication quite a lot. Sadly, time caught up with me so I couldn't finish all of the front-end.

Challenges Faced During Development

Quite many challenges were faced during this exam, but I will mention the major ones below.

One of the significant challenges I faced during this project was becoming very sick during the first four/five days of the exam. So I was unable to work. This unexpected delay caused me to get behind schedule, forcing me to adapt my workflow. It taught me valuable lessons in time management, staying focused under pressure, and maintaining motivation.

However, major challenges didn’t end there. Four days before the deadline of the exam, my fairly new computer’s main drive crashed and malfunctioned. I lost all documents on the computer. Luckily I have been taking back-up every day of my exam project, so it was restored. However, this took a long time to fix, and took away another two days of precious time before getting a setup to work on the exam again.

Another significant hurdle towards the end was handling cross-origin requests from the front-end to the back-end API. Initially, CORS errors blocked interactions, but this wasn’t detected until towards the end of the exam when I finally detected the blockage in the developer tool. Then I could configure middleware to handle these requests. Additionally, implementing JWT-based authentication in the front-end required careful attention to ensure tokens were passed correctly between sessions. I solved this by looking into how to extract the token when the admin logs in.

Lessons Learned

Through this project, I gained a deeper appreciation for the complexities of back-end and front-end development. I learned more than ever the importance of breaking tasks into manageable pieces. Debugging and testing became second nature, as every issue resolved brought a clearer understanding of the underlying systems. Though it was very tough at times. The ability to debug and adapt was crucial in overcoming obstacles and reaching my best potential in this exam project.

.env:

Back-end:

```
HOST="localhost"

ADMIN_USERNAME="your_admin_username"

ADMIN_PASSWORD="your_admin_password"

DATABASE_NAME="your_database_name"

DIALECT="mysql"

PORT="3001"

TOKEN_SECRET="your_token_secret"
```

Front-end:

```
BACKEND_API_URL = "http://localhost:3001"
PORT = "3000"
```