Reflection Report

Exam Project

Student Name: Farid Vahabzada
Date of Submission: 09 June 2024

Noroff
School of technology
and digital media

Table of Contents

Introduction

This document consists of three sections, **Project Plan**, **Database Design** and **Reflection Report** respectively.

In the first section, **Project Plan**, a brief glimpse into workload compartmentalization with the help of the Jira software is given.

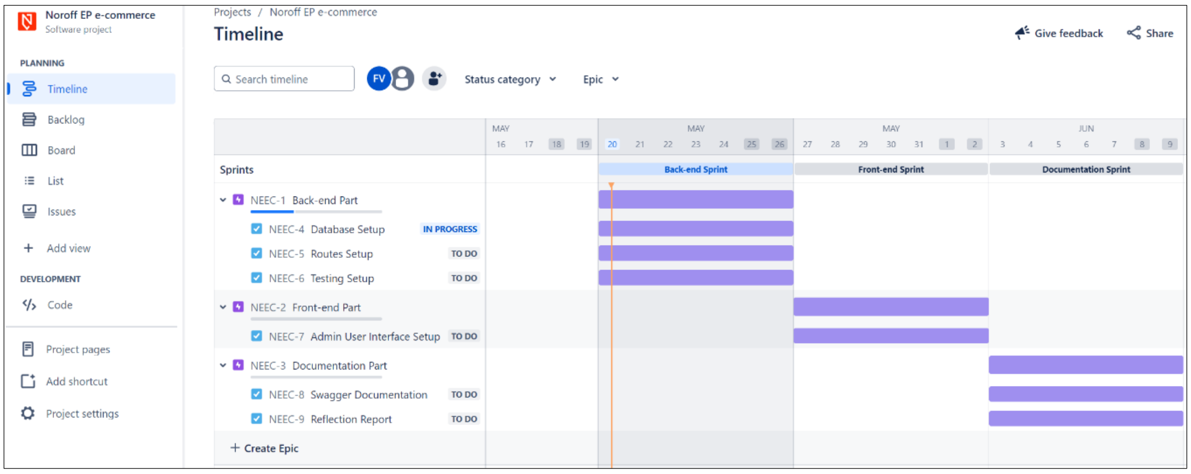
In the second section, **Database Design**, the diagram view of all the tables with the properties and relationships they cover is shown.

The last section, **Reflection Report**, discusses how the project evolved and progressed, and the problems or difficulties experienced throughout the development of the project. And how some of the obstacles were tackled while other issues were left for future versions of the app.

Project Plan

Jira Roadmap

Figure 1 depicts the Jira Timeline which we created as a first step. Here you can find 3 main sprints and 3 epics with all the related tasks.



Noroff EP e-commerce
Software project

PLANNING
- Timeline
- Backlog
- Board
- List
- Issues
- Add view

DEVELOPMENT
- Code
- Project pages
- Add shortcut
- Project settings

Projects / Noroff EP e-commerce

Timeline

Give feedback · Share

Search timeline · FV · Status category · Epic

Sprints		Back-end Sprint	Front-end Sprint	Documentation Sprint
NEEC-1 Back-end Part				
NEEC-4 Database Setup	IN PROGRESS			
NEEC-5 Routes Setup	TO DO			
NEEC-6 Testing Setup	TO DO			
NEEC-2 Front-end Part				
NEEC-7 Admin User Interface Setup	TO DO			
NEEC-3 Documentation Part				
NEEC-8 Swagger Documentation	TO DO			
NEEC-9 Reflection Report	TO DO			

+ Create Epic

Figure 1. Jira Roadmap.

We dedicated our first week solely to material coverage of the previous courses. There are a few vague tasks every epic covers. Eventually, all the tasks were finished, though not exactly within the set time frames.

Database Design

Database ERD

The latest entity relationship diagram of the database is shown in Figure 2.

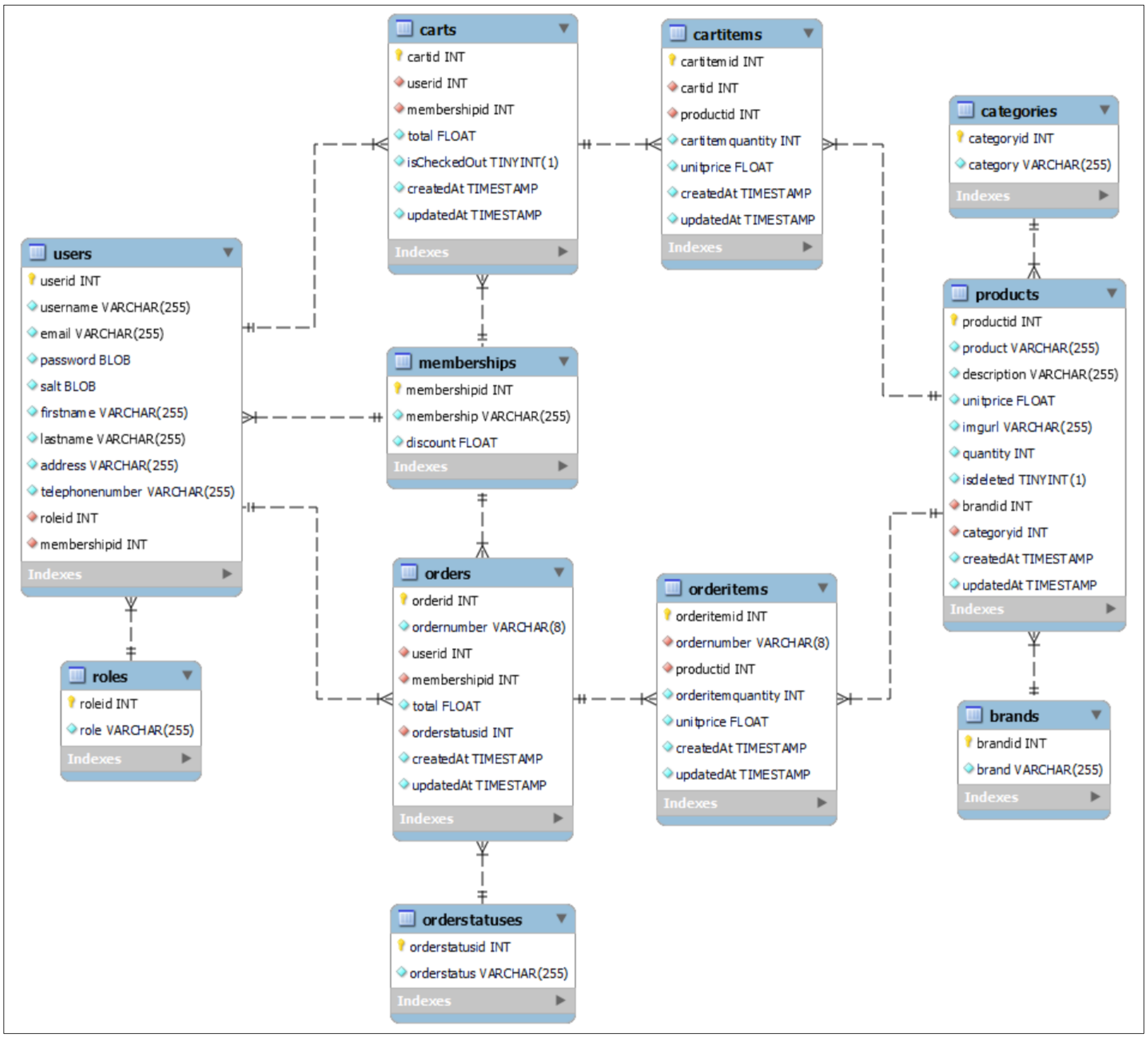


Figure 2. Database ERD.

Relationships

As can be seen from the Database ERD figure, there exist 11 tables in total, each with its own set of properties. To achieve the Third Normal Form (3NF), we separated the order statuses table from the orders table. We could do the same for the isCheckedOut and isdeleted properties as well, but we constrained ourselves from doing that as it would add more complexity to the project.

A user can have multiple carts and multiple orders, while each order or cart can have only one user, that is why we used One-to-Many relationships here.

A user can have only one role and one membership, but many users can get the same role or membership, so the choice was for One-to-Many. The same logic applies to the relationships between orders and order statuses, products and categories/brands, carts/orders and memberships, cart items and carts, and order items and orders.

Originally, we wanted to add the One-to-One relationship between carts and orders, and between cart items and order items too. For the same reasoning of complexity, this would bring to the project, and we later achieved this logic through the lines of codes. It is the same way we can only get one active cart per user at a time.

Reflection Report

Progression

Real work started with creating the database and its tables, and then constructing all the relationships correctly. Next was populating the database and creating the login/registration routes. Many crucial decisions had to be made before reaching future tasks, so it wouldn't cause any interference with their realization. This brought us to choosing the right set of packages.

We started writing the code for the table routes with the categories table. There you can find many different techniques utilized before finding the universal one in many aspects. For example, when it came to validation, if we create a new category then error handling is done through the Sequelize ORM. At the end of the day, we preferred to go with the Ajv package. As it offered us enough flexibility and the least amount of code volume.

We did much extra work and error handling consideration and tried to have every CRUD operation for every table available. After all the back-end routes were finished, we began with the unit testing requirement. It is very important to note that you **must** start the app and populate the database first! Then you can start the tests with the **npm test** command.

Then it came to developing the front-end section of the project. We set the token after authorization in the cookies, so we could retrieve and use it. We tried to mimic the front-end mockup as closely as possible. The Bootstrap JavaScript modal was very useful as a plugin. Only necessary functionality and user experience were the goals.

Lastly, we finalized our work with the Swagger documentation endpoint and other small documentation-related tasks.

Challenges

In the beginning, it was hard to formulate any specific tasks. There were many interrelated tasks, and every time one specific task was finished a new unexpected task would reveal itself. Or an issue that would cancel all the progress made.

In the front end, a few challenges worth mentioning, it was quite a challenge to account for all the possible variants of JWT token errors without service files. For instance, what if we got a valid JWT token but its owner no longer has the admin role, then we shouldn't let that token owner access any of the Admin Front-End endpoints, right?

Also, in the mockup we found the soft delete functionality to be excessively accessible at two different locations. Then, having the roles table as a page was unnecessary as all the roles are visible if you edit a user. The same logic makes product names unique. Also, we aren't letting admins alter emails/usernames or memberships.

It was quite a challenge to add a vertical rule to the navigation bar and change it to a horizontal one at some point during zooming. There is a home page now, it is accessed by clicking the Home or the Admin Front-End brand.

We also considered having these tables empty for the pages, so getting some console errors is quite normal. They don’t cause any functionality issues.

If we discuss the Swagger endpoint, then it was a challenge to add the authorization button there. It was done for a better user experience.

In the back end, building flawless e-commerce logic was a real challenge. When to create a cart or an order, when to update the membership or the product quantity, when to recalculate the total cost, or what errors to receive depending on the situation.

Just logging in with both username and email, felt like bending the physics laws. Because of the way MySql functions, we left validating these credentials case insensitive.

The login schema got some comments and there are commented parts all over. In a way, we don’t need validation for the logging-in process.

The way the code was written the brands, categories and order statuses tables aren’t populated in an orderly fashion. This was later accounted for in the front-end part.

We were challenged with the ordernumber property not being considered as a foreign key for the order items table.

When we check if the tables are empty, we check for the root tables where all the main foreign keys are sourced from.

When a product is soft deleted, we get old information about the product in the result, where it states the isdeleted property as false.

The assumptions are that a user does no database manipulation other than through the Postman app or the front-end part, and initially populates the table. Wherever possible we added error handling, but these aforementioned actions can still cause issues with the logic.

There is an assumption that default values cannot be changed or removed, and cannot delete admins, memberships, roles, order statuses, and categories/brands belonging to products.

The meaning of soft delete for carts is defined by the isCheckedOut property. No carts can be deleted. New carts are created on user registration and checkout. Cart items can only be deleted if the cart is active. Neither can be done for orders or order items.

Testing was quite a challenge because of the asynchronous nature of the operations done. Therefore, to get successful test results we changed what the results should look like.

Searching the products returns the count as well. In future app versions, this feature can be added to the front end.

There wasn’t enough time to consider all the possible errors that might occur. For instance, if many check-outs are done simultaneously, or a user checks out while the admin changes the product details.

Overall, this project can be called one big challenge and we discussed only a small fraction of them here. It feels almost dreamlike to me what I accomplished in this Exam Project all by myself only.

References

Get started with bootstrap (no date) · *Bootstrap v5.3*. Available at: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (Accessed: 09 June 2024).

OpenAPI 3.X (no date) *Swagger Autogen*. Available at: https://swagger-autogen.github.io/docs/openapi-3/#google_vignette (Accessed: 09 June 2024).

Basic structure (no date) *Swagger*. Available at: https://swagger.io/docs/specification/2-0/basic-structure/ (Accessed: 09 June 2024).

Getting started · Jest (2024) *Jest RSS*. Available at: https://jestjs.io/docs/getting-started (Accessed: 09 June 2024).

Sequelize V6 (2024) *Sequelize*. Available at: https://sequelize.org/docs/v6/ (Accessed: 09 June 2024).

JSON schema generator (no date) *JSON Schema Tool*. Available at: https://www.jsonschema.net/ (Accessed: 09 June 2024).

MySQL documentation (no date) *MySQL*. Available at: https://dev.mysql.com/doc/ (Accessed: 09 June 2024).

Free stock photos, royalty free stock images & Copyright Free Pictures (no date) *Pexels*. Available at: https://www.pexels.com/ (Accessed: 09 June 2024).

Favicon generator for perfect icons on all browsers (no date) *RealFaviconGenerator.net*. Available at: https://realfavicongenerator.net/ (Accessed: 09 June 2024).

Atlassian (no date) *Bring every team together under one roof*, *Atlassian*. Available at: https://www.atlassian.com/software/jira (Accessed: 09 June 2024).