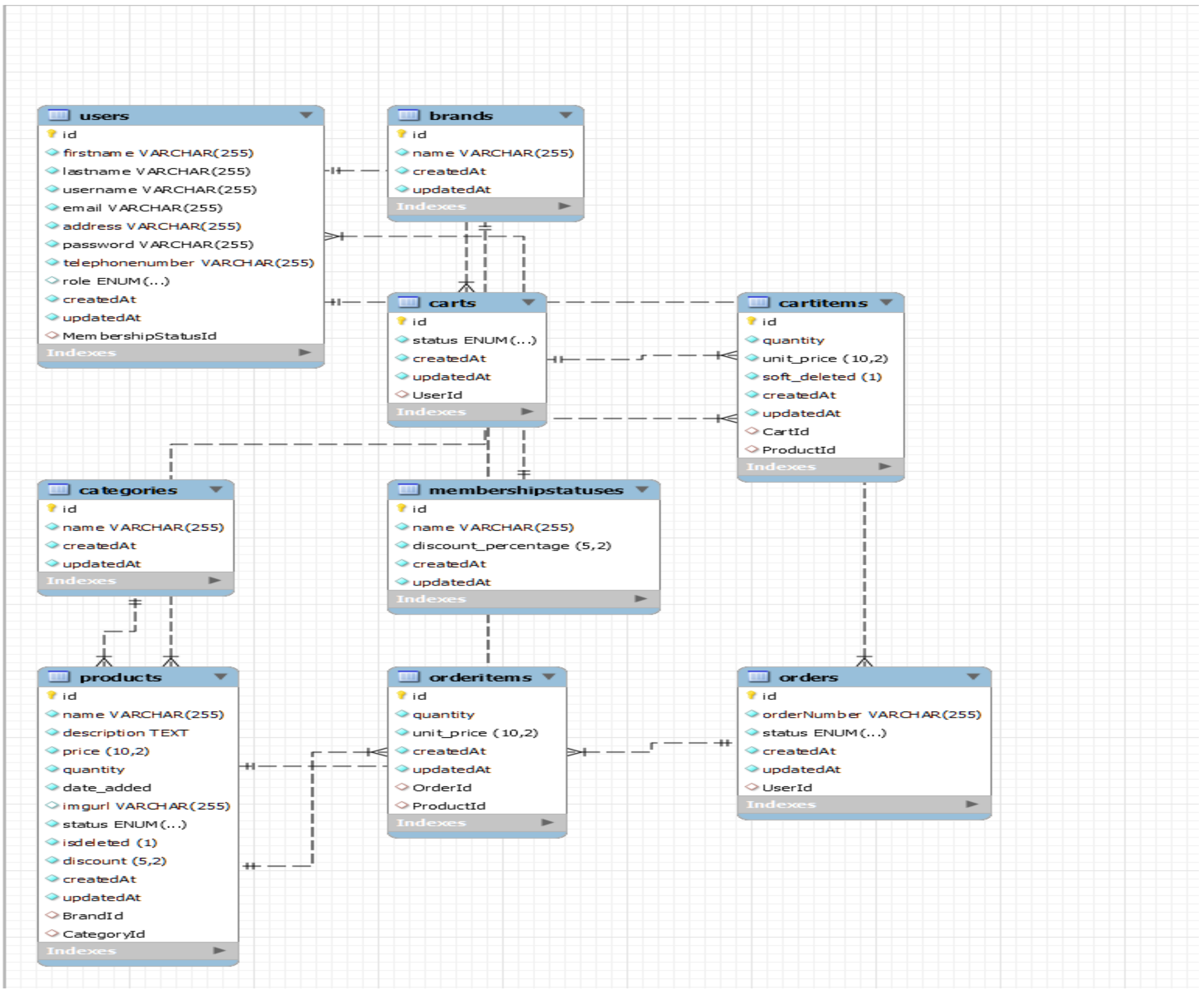
Reflection Report for Noroff E-Commerce Application

I- Introduction

This document reveals development process of the application 'Noroff E-Commerce Application'. It contains the database structure and relationships, Jira Epics and Sprints, the challenges, and issues I faced during the development process and how I dealt with them. The document also covers the references that I used. The appendix section reveals some screenshots of the application. Three 'README.md' files have been added to the repository for explanations. Please check these files for details about the application in each folder.

II- Database Structure and Relationship Between Tables

A- Database ERD Table (all tables and relationships)



B- Relationship between Tables

The relationship between the tables in the "Noroff E-Commerce Database" can be explained as follows:

1- **Brand and Product Tables**

- **Relationship: One-to-Many**: A brand has many products. For example, Brand Apple has many different products. This relationship is done a foreign key in the 'Product' model shows 'Brand' model.

2- **Category and Product Tables**

-**Relationship: One-to-Many**: Same as Brand and Product: many products can have same category. For example, different products can have the same category such as iPhone, Samsung Phones. Product table has reference to the 'Category' it belongs to.

3- Cart and User Tables

- Relationship: Many-to-One: Each 'Cart' belongs to one 'User'. It means a user can have many shopping carts.

4- Cart and CartItem Tables

- Relationship: One-to-Many: A 'Cart' contains many 'CartItems' . A cart can have many CartItems (products).

5- CartItem, Product and Cart Tables:

-Relationship: Many-to-One (for both 'Product' and 'Cart'): Each CartItem is connected to one 'Cart' and one 'Product'. It shows which products are in which cart and in what quantities.

6- Order and User Tables:

-Relationship: Many-to-One: A user can have multiple orders.

7- Order and OrderItem Tables:

-Relationship: One-to-Many: An 'Order' can contain multiple 'OrderItems.'

8- **OrderItem, Product, and Order:**

-Relationship: Many-to-One (for 'Product and 'Order'): Each 'OrderItem' connects to one 'Order' and one 'Product'.

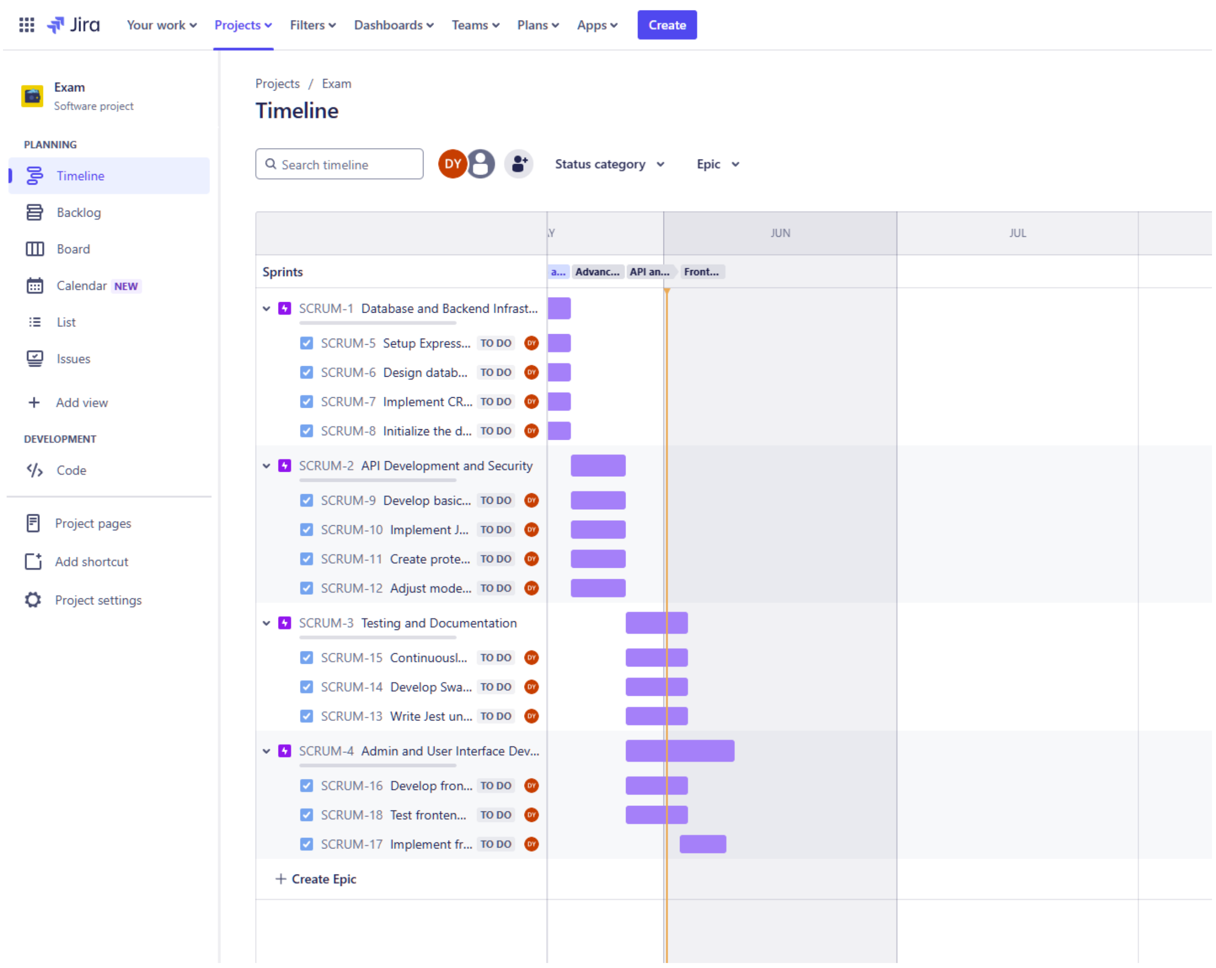
9- **User and MembershipStatus:**

- Relationship: Many-to-One: Each 'User2 has a 'MembershipStatus'.

III -Jira

The figures below show the Jira Epics and Sprints of the Application.

Epics

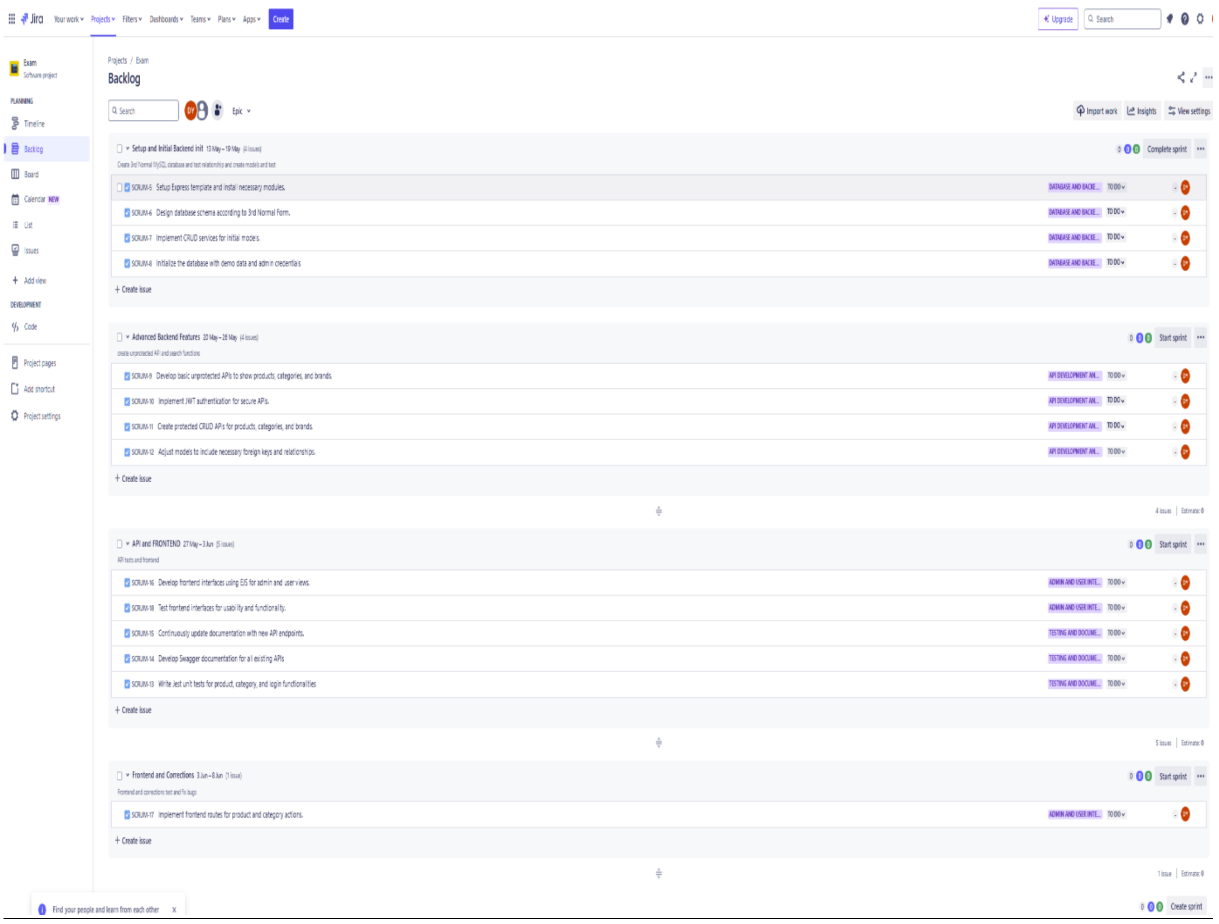


Timeline

Sprints

SCRUM-1 Database and Backend Infrast...
SCRUM-5 Setup Express... TO DO
SCRUM-6 Design datab... TO DO
SCRUM-7 Implement CR... TO DO
SCRUM-8 Initialize the d... TO DO
SCRUM-2 API Development and Security
SCRUM-9 Develop basic... TO DO
SCRUM-10 Implement J... TO DO
SCRUM-11 Create prote... TO DO
SCRUM-12 Adjust mode... TO DO
SCRUM-3 Testing and Documentation
SCRUM-15 Continuousl... TO DO
SCRUM-14 Develop Swa... TO DO
SCRUM-13 Write Jest un... TO DO
SCRUM-4 Admin and User Interface Dev...
SCRUM-16 Develop fron... TO DO
SCRUM-18 Test fronten... TO DO
SCRUM-17 Implement fr... TO DO
+ Create Epic

Sprints


Projects / Exam

Backlog

Setup and Initial Backend init 13 May – 19 May (4 issues)

Create 3rd Normal MySQL database and test relationship and create models and test

SCRUM-5	Setup Express template and install necessary modules.	DATABASE AND BACKE...	TO DO
SCRUM-6	Design database schema according to 3rd Normal Form.	DATABASE AND BACKE...	TO DO
SCRUM-7	Implement CRUD services for initial models.	DATABASE AND BACKE...	TO DO
SCRUM-8	Initialize the database with demo data and admin credentials	DATABASE AND BACKE...	TO DO

Advanced Backend Features 20 May – 26 May (4 issues)

create unprotected API and search functions

SCRUM-9	Develop basic unprotected APIs to show products, categories, and brands.	API DEVELOPMENT AN...	TO DO
SCRUM-10	Implement JWT authentication for secure APIs.	API DEVELOPMENT AN...	TO DO
SCRUM-11	Create protected CRUD APIs for products, categories, and brands.	API DEVELOPMENT AN...	TO DO
SCRUM-12	Adjust models to include necessary foreign keys and relationships.	API DEVELOPMENT AN...	TO DO

4 issues | Estimate: 0

API and FRONTEND 27 May – 3 Jun (5 issues)

API tests and frontend

SCRUM-16	Develop frontend interfaces using EJS for admin and user views.	ADMIN AND USER INTE...	TO DO
SCRUM-18	Test frontend interfaces for usability and functionality.	ADMIN AND USER INTE...	TO DO
SCRUM-15	Continuously update documentation with new API endpoints.	TESTING AND DOCUME...	TO DO
SCRUM-14	Develop Swagger documentation for all existing APIs	TESTING AND DOCUME...	TO DO
SCRUM-13	Write Jest unit tests for product, category, and login functionalities	TESTING AND DOCUME...	TO DO

5 issues | Estimate: 0

Frontend and Corrections 3 Jun – 8 Jun (1 issue)

Frontend and corrections test and fix bugs

SCRUM-17	Implement frontend routes for product and category actions.	ADMIN AND USER INTE...	TO DO

1 issue | Estimate: 0

IV Project Reflection

Discussion (The Progression of the Project, Challenges, and solutions)

In this section, the development process of the application, the problems that emerged during this process, and the solution methods will be discussed together.

Database construction and challenges

I believe the most important phase in developing this application was correctly setting the database and relationships between its tables. Because an incorrect database setup could have caused errors in the next stages. Initially, the relationships between the tables for this app needed to adhere to the 3rd normal form, as expected. For example, we have products categorized by brands, and a user might own multiple products from different categories.

To ensure this, I repeatedly reviewed the information in the exam instructions, then manually drew the database schemas and relationships. My aim was to clearly identify the 'one-to-many' and 'many-to-many' relationships between the tables. During this process, I consulted our course Databases-Modules, and the Database Relations video was particularly helpful. Additionally, Scott Simpson's LinkedIn Learning video provided important insights.

To confirm the accuracy of these relationships, I wrote Sequelize models and submitted these to CHATGPT. However, I think that such AI tools are not yet fully capable of handling this type of analysis accurately. Despite updating the models with new relationships, CHATGPT confirmed that each was in the correct third normal form.

While learning a new module, a new programming language, AI may provide basic information about templated and their uses. However, I mostly found that this information might be outdated. Still, I can say that AI is useful to create a module template as long as the developer knows how to use this template. In a complex application with many parts, AI cannot resolve relationships between components.

For example, while developing the back-end Swagger modules, I faced issues where some Swagger definitions either didn't work at all or appeared incorrectly in the Swagger documentation. Despite extensive research online, I couldn't find any solution. I asked ChatGPT, providing the relevant routers, and inquiring about the potential errors. However, the solutions ChatGPT offered were not applicable. It suggested changing my models, which were unrelated to the Swagger modules. It then recommended creating Swagger documentation, but I was already using an auto-generation module for this purpose.

Upon further inspection of the Swagger code, I discovered that I had omitted the '#' sign in one crucial line. This experience taught me the importance of meticulously reviewing the code step-by-step as a developer, as even small omissions can lead to significant issues.

I decided to use the following approach to this challenge: During the application development process, I could confirm that the database relationships were correctly defined only if the functions managing these relationships operated correctly. As I will discuss later in this chapter, AI suggests non- solutions that are not effective, when in fact, very simple changes could suffice.

An Easy Yet Powerful Tool for Debugging.

In my experience, a simple but yet powerful tool during application development stage is ` console.log` . It's especially effective for debugging or testing functions, particularly for analyzing if a function is passing values correctly or examining the contents of a JSON structure from an API. I frequently used 'console.log' to solve issues challenges. For instance, when receiving a JSON message, I used it to verify if the products were listed correctly, if a module was fully utilized, or if the output from a function was undefined.

I think 'console.log' is powerful tool for these tests because it shows immediate verification of whether functions are integrating properly with other modules. I used this tool extensively and chose the logs in the code for future development. I may say that this approach might extend the development time, but it ensures each function's integration and functionality can be continuously assessed.

Routers and Controllers Construction

When I was creating back-end APIs, I used a different approach which I have seen in Udemy courses especially at this: (Master Node by building a real-world RESTful API and web app (with authentication, Node.js security, payments & more). The aim was to keep the router files as concise as possible by mainly using controllers. However, due to my unfamiliarity with this approach, I faced some challenges. This method particularly complicated the creation of standalone front-page routers. For instance, in the brand.js router file, the route: ‘ router.get(‘/’, BrandController.getAllBrands); ‘ is managed by ‘BrandController’ . This setup makes the routers cleaner and more organized.

app.js and appTest.js Files

In the development of this application, users or testers will notice that two different app.js files are used. This was strategically implemented to ensure accurate results when conducting tests with Jest's supertest framework. Essentially, there were two feasible approaches to set up the testing environment: one was to recreate the necessary modules and routers specifically for testing; the other, which appeared simpler, was to directly copy the existing app.js.

Initially, I tried using the existing app.js directly for testing. However, every time this version of the app was initiated, it automatically triggered database synchronization. This repeatedly caused failures during testing. After consulting with artificial intelligence for solutions, I adopted the recommended approach: creating a second, nearly identical app.js file, which I named appTest.js. This version allows manual triggering of the setup processes, thereby providing more control and avoiding the automatic database sync issues encountered with the original app.js. This modification successfully facilitated the testing process without unintended interactions with the database.

Front-Page in Back-End Folder and Front-Page in Front-End Folder

The user will see two different front ends within the application setup. The first is an embedded front end located in the Back-End folder, which is designed for future development catering to non-admin users and visitors. It operates at port 3000. The front end in the Front-End folder runs on port 5000 and prioritizes a more independent admin panel.

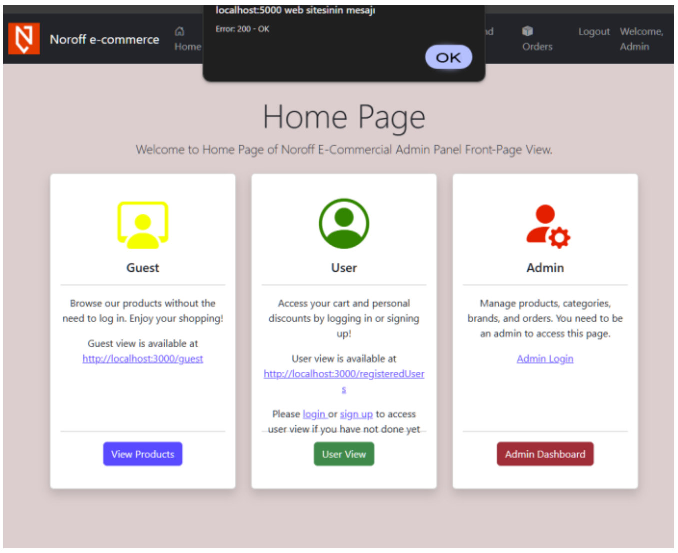
To address potential cross-origin resource sharing issues, the CORS module has been implemented. As detailed in the readme.md files in each folder, the aim was to create a fully independent interface that operates without reliance on Sequelize server modules. This separation provides modularity and allows each part of the application to evolve independently, aligning with different user roles and access levels.

The interface on both sides is nearly identical, created using Bootstrap. For users testing the application, it's crucial to note one important detail, which is highlighted in both readme files: the TOKEN_SECRET in both .env files must be exactly the same. Please ensure this is correctly configured.

Issues Related to Login

A bug occurs when attempting to access the admin panel through the user interface in the front-end section. Although this bug isn't severe, it was still present when this document was created. If a user who is already logged in as an admin tries to access the admin panel from the homepage, they receive an "Error-200-OK" alert. I suspect this issue might be linked to the mockauth.js middleware, but it does not disrupt functionality. Users or testers can still access the admin panel by clicking the OK button on the alert. Updates will be provided if this error is resolved.

Figure 3- showing bug.



localhost:5000 web sitesinin mesajı

Error: 200 - OK

Home Page

Welcome to Home Page of Noroff E-Commercial Admin Panel Front-Page View.

Guest

Browse our products without the need to log in. Enjoy your shopping!

Guest view is available at http://localhost:3000/guest

User

Access your cart and personal discounts by logging in or signing up!

User view is available at http://localhost:3000/registeredUsers

Please login or sign up to access user view if you have not done yet

Admin

Manage products, categories, brands, and orders. You need to be an admin to access this page.

Admin Login

Final Notes

Apart from these bugs and issues, there were various errors throughout the development process. For a programmer, each stage inevitably introduces unexpected problems and difficulties. Modules and functions that seemed straightforward often led to issues that consumed hours to resolve. As mentioned earlier, a combination of AI, manual debugging, and console messages were employed to address these challenges.

In addition to the Noroff and LinkedIn videos and resources previously described, I referred to an article from Vertabelo (https://vertabelo.com/blog/er-diagram-for-online-shop/) at the project's outset for ideas on model creation. Additionally, the Codeium plugin enhanced the coding experience, making it more comfortable.

V- Swagger

Swagger has also been added to the back-end folder section of the application for API documentation. If the user wishes, he can access this document, which includes all back-end APIs, at http://localhost:3000/doc/. Swagger definitions are also separately written in the 'swaggerdefinitions.js' file located under the Data folder.

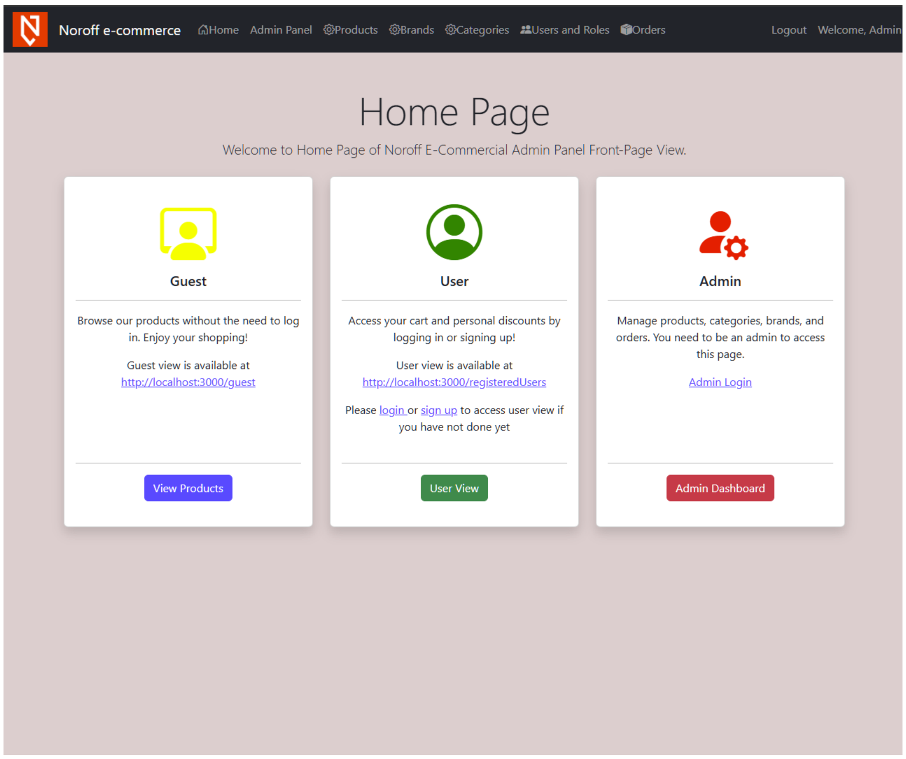
VI- References

1. Backend by Noroff. "Database Module Introduction." YouTube Video. Accessed for learning database fundamentals.
2. ChatGPT, OpenAI. Assistance with database modeling, and idea generation for e-commerce project development.
3. Codeium Plugin. Used for code enhancement and error resolution during development.
4. LinkedIn Learning. "Programming Foundations: Databases - Relationship Rules and Referential Integrity" by Simon Allardice. Course Link. Accessed for understanding database relationships.
5. Noroff Learning Resources. T8CA25f1UKnb6Kt0X9xr. Accessed during the project for foundational learning.
6. Udemy. "Node.js, Express, MongoDB Bootcamp" by Jonas Schmedtmann. Course Link. Accessed for practical development skills in Node.js and MongoDB.
7. Vertabelo Blog. "ER Diagram for Online Shop." Article. Used as a reference for idea generation on commercial site structures.

VII -Appendix Screenshots from Application

Some screenshots from the application:

Homepage (index.ejs)



Home Page

Welcome to Home Page of Noroff E-Commercial Admin Panel Front-Page View.

Guest

Browse our products without the need to log in. Enjoy your shopping!

Guest view is available at http://localhost:3000/guest

View Products

User

Access your cart and personal discounts by logging in or signing up!

User view is available at http://localhost:3000/registeredUsers

Please login or sign up to access user view if you have not done yet

User View

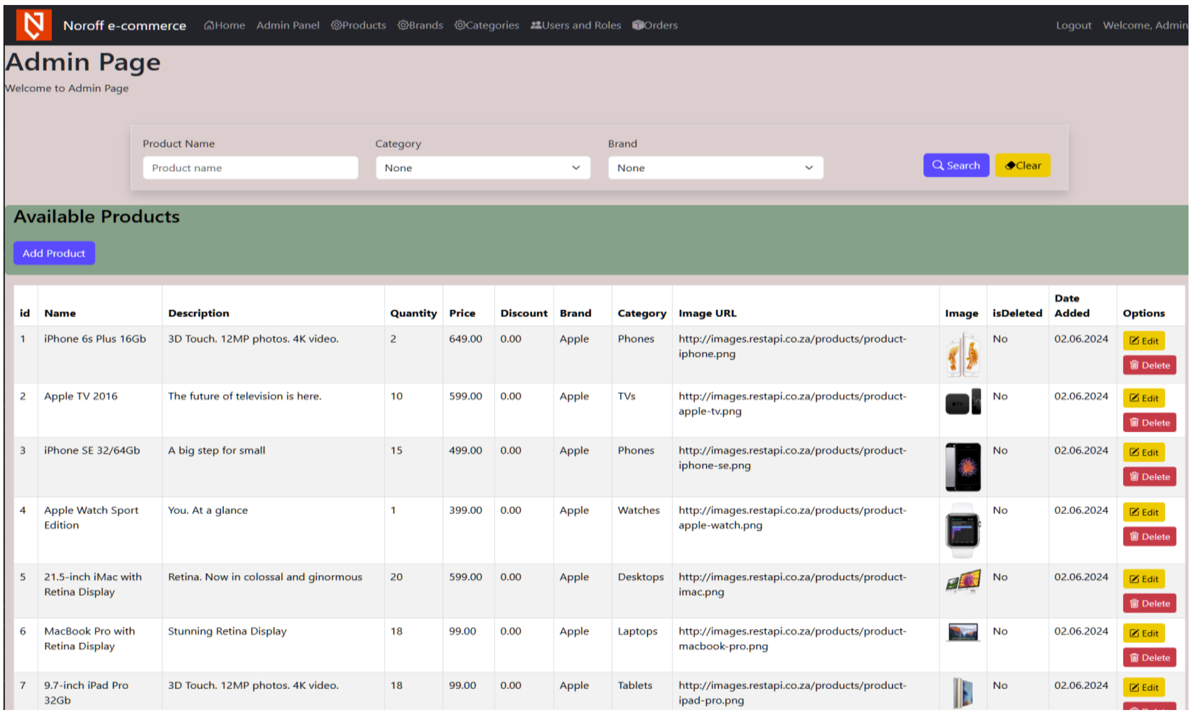
Admin

Manage products, categories, brands, and orders. You need to be an admin to access this page.

Admin Login

Admin Dashboard

Admin Panel



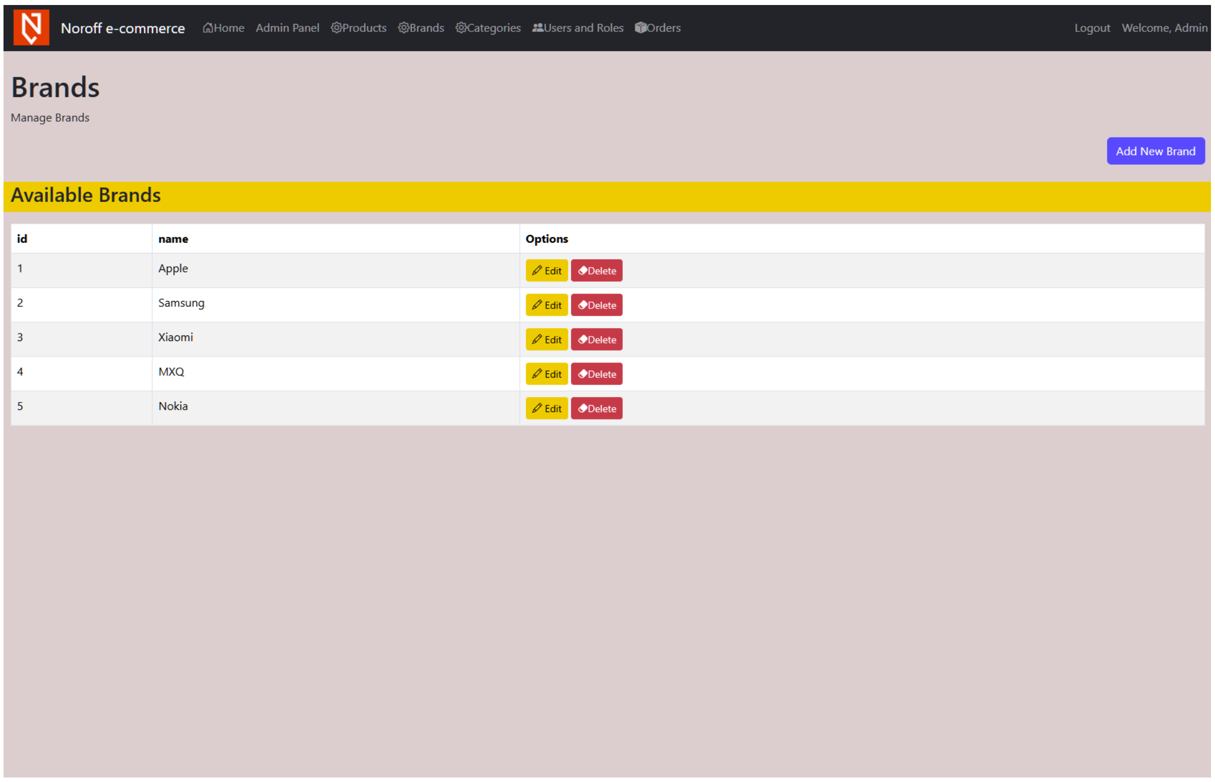
Admin Page

Welcome to Admin Page

Available Products

id	Name	Description	Quantity	Price	Discount	Brand	Category	Image URL	Image	isDeleted	Date Added	Options
1	iPhone 6s Plus 16Gb	3D Touch. 12MP photos. 4K video.	2	649.00	0.00	Apple	Phones	http://images.restapi.co.za/products/product-iphone.png		No	02.06.2024	Edit Delete
2	Apple TV 2016	The future of television is here.	10	599.00	0.00	Apple	TVs	http://images.restapi.co.za/products/product-apple-tv.png		No	02.06.2024	Edit Delete
3	iPhone SE 32/64Gb	A big step for small	15	499.00	0.00	Apple	Phones	http://images.restapi.co.za/products/product-iphone-se.png		No	02.06.2024	Edit Delete
4	Apple Watch Sport Edition	You. At a glance	1	399.00	0.00	Apple	Watches	http://images.restapi.co.za/products/product-apple-watch.png		No	02.06.2024	Edit Delete
5	21.5-inch iMac with Retina Display	Retina. Now in colossal and ginormous	20	599.00	0.00	Apple	Desktops	http://images.restapi.co.za/products/product-imac.png		No	02.06.2024	Edit Delete
6	MacBook Pro with Retina Display	Stunning Retina Display	18	99.00	0.00	Apple	Laptops	http://images.restapi.co.za/products/product-macbook-pro.png		No	02.06.2024	Edit Delete
7	9.7-inch iPad Pro 32Gb	3D Touch. 12MP photos. 4K video.	18	99.00	0.00	Apple	Tablets	http://images.restapi.co.za/products/product-ipad-pro.png		No	02.06.2024	Edit

Brands



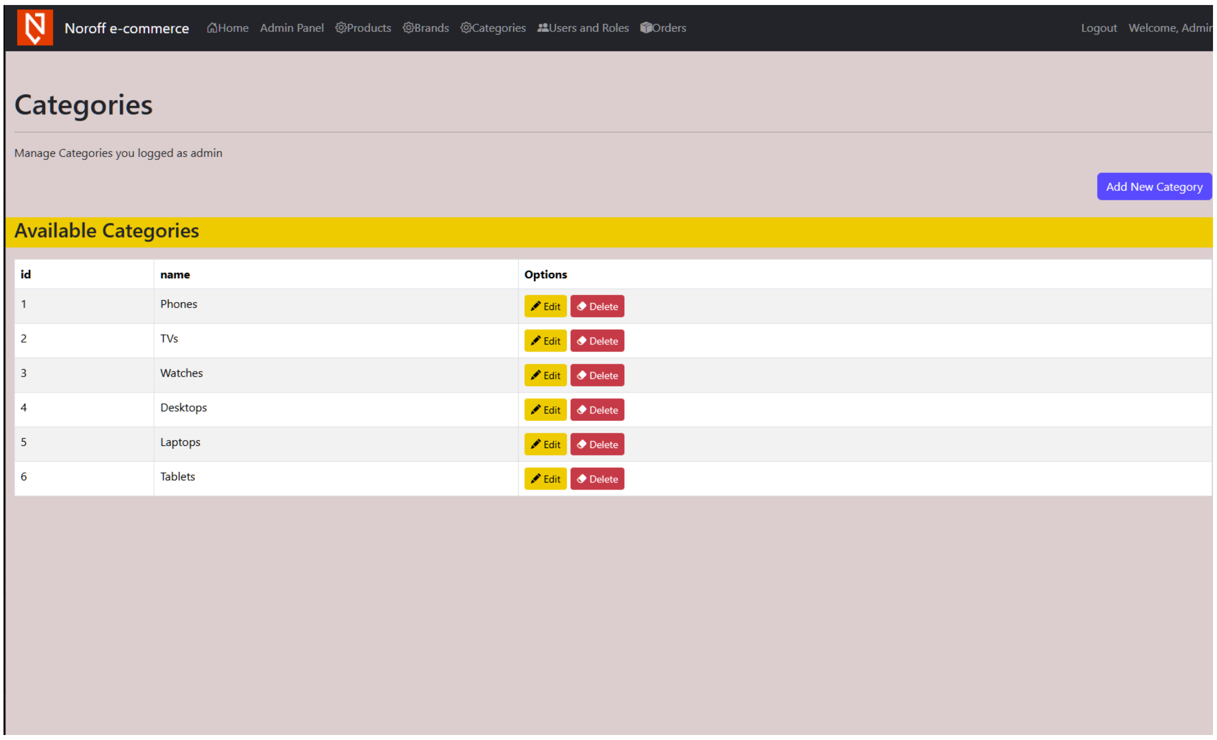
Brands

Manage Brands

Available Brands

id	name	Options
1	Apple	Edit Delete
2	Samsung	Edit Delete
3	Xiaomi	Edit Delete
4	MXQ	Edit Delete
5	Nokia	Edit Delete

Categories



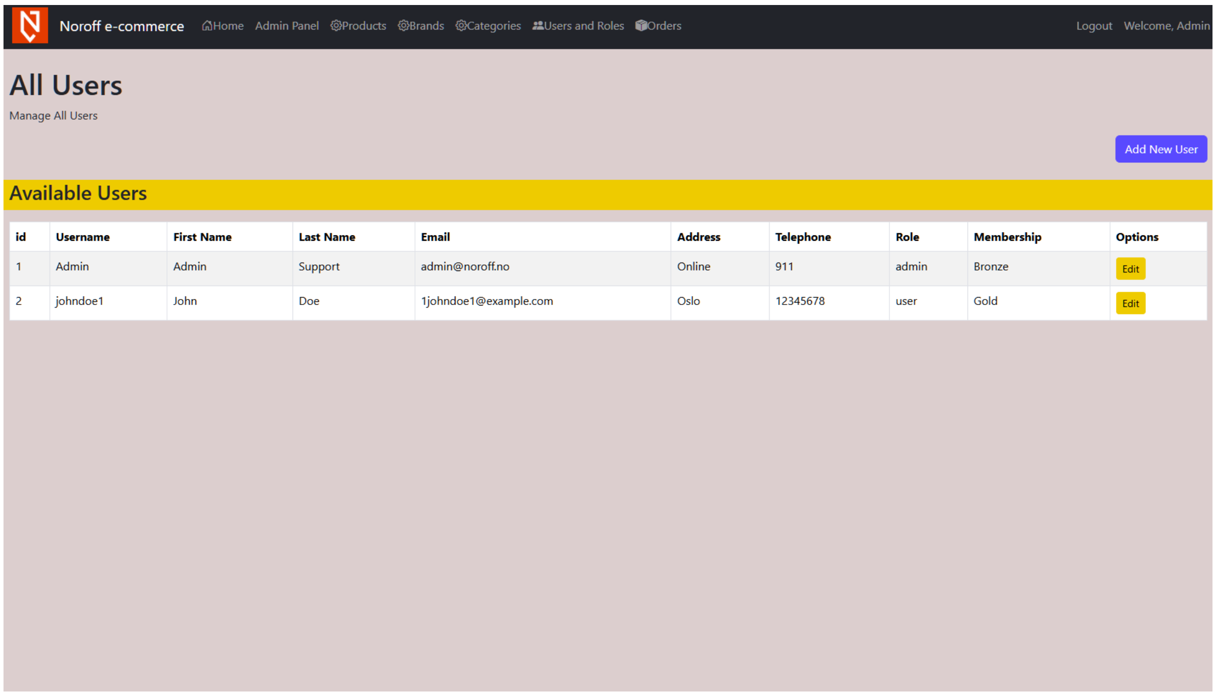
Categories

Manage Categories you logged as admin

Available Categories

id	name	Options
1	Phones	Edit Delete
2	TVs	Edit Delete
3	Watches	Edit Delete
4	Desktops	Edit Delete
5	Laptops	Edit Delete
6	Tablets	Edit Delete

Users



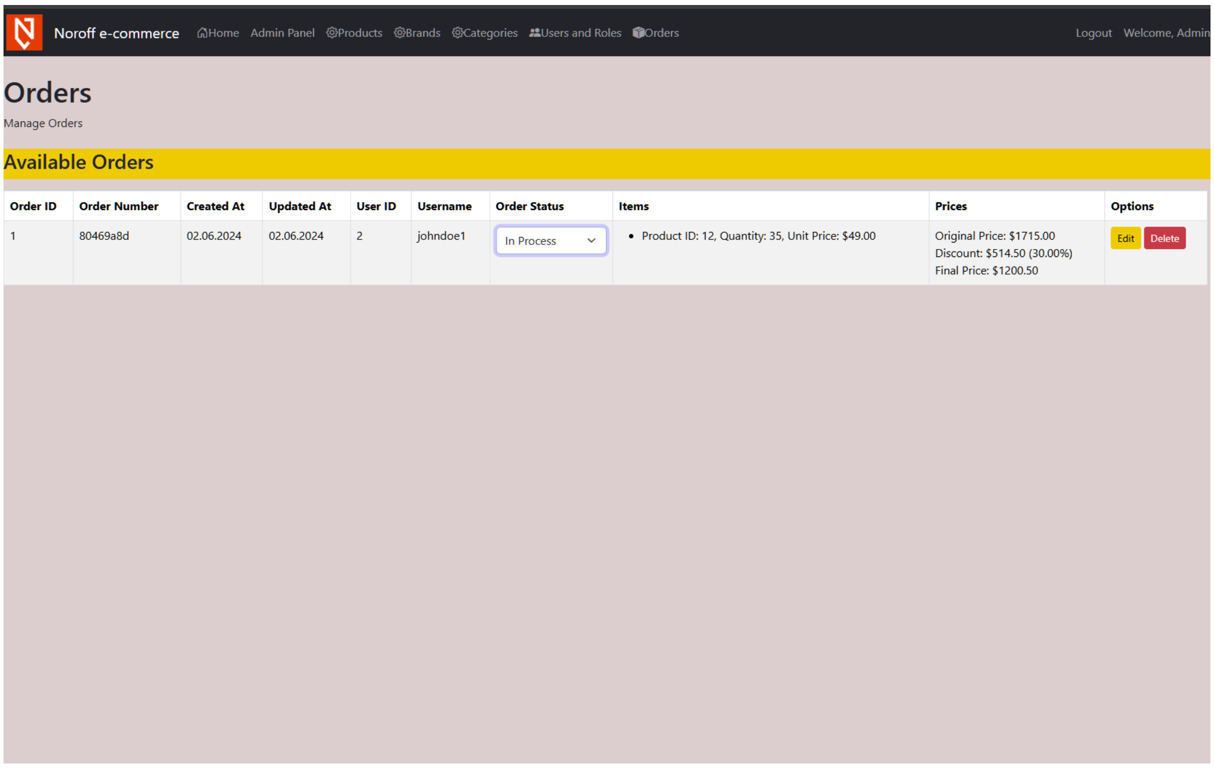
All Users

Manage All Users

Available Users

id	Username	First Name	Last Name	Email	Address	Telephone	Role	Membership	Options
1	Admin	Admin	Support	admin@noroff.no	Online	911	admin	Bronze	Edit
2	johndoe1	John	Doe	1johndoe1@example.com	Oslo	12345678	user	Gold	Edit

Orders



Orders

Manage Orders

Available Orders

Order ID	Order Number	Created At	Updated At	User ID	Username	Order Status	Items	Prices	Options
1	80469a8d	02.06.2024	02.06.2024	2	johndoe1	In Process	• Product ID: 12, Quantity: 35, Unit Price: $49.00	Original Price: $1715.00 Discount: $514.50 (30.00%) Final Price: $1200.50	Edit Delete