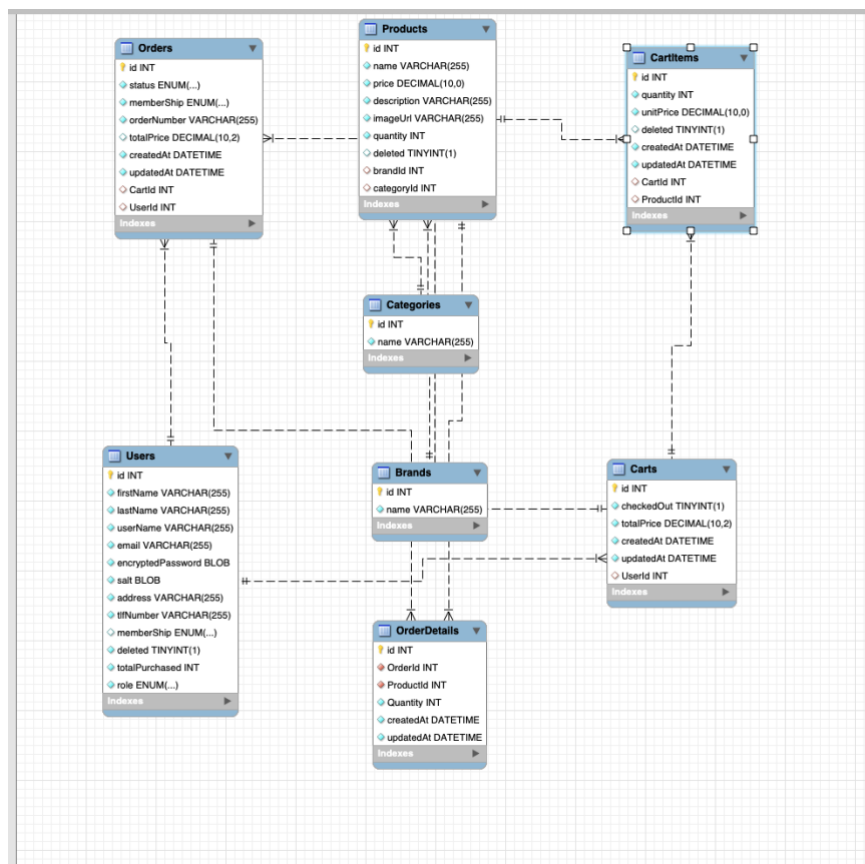


1. Screenshot of the complete Database ERD

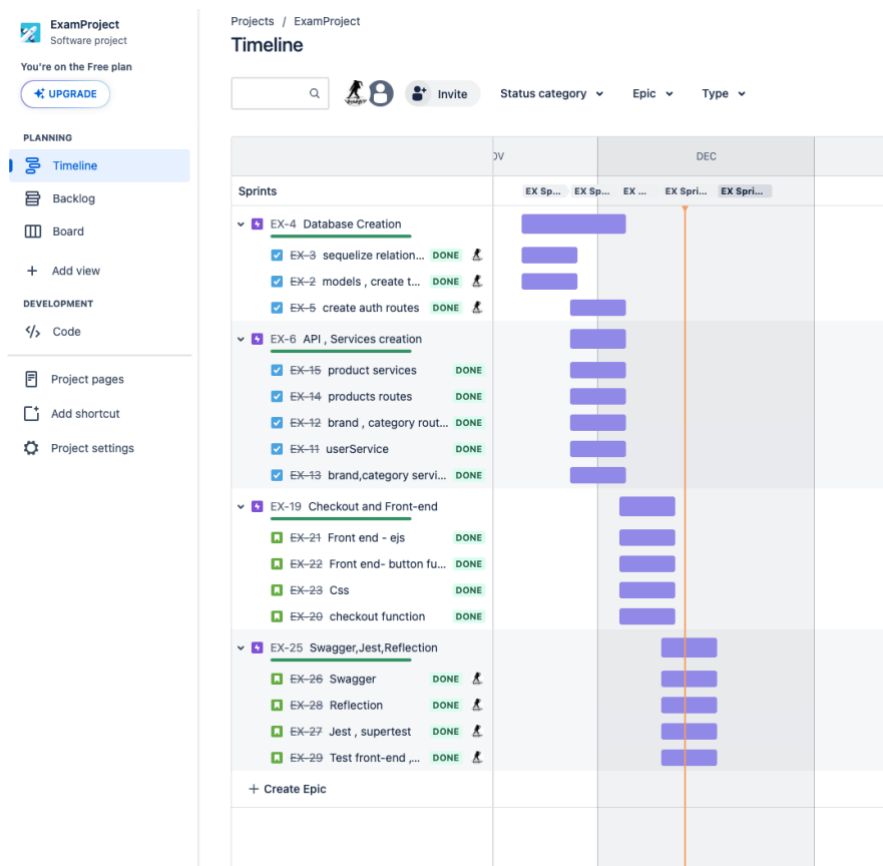


2. An explanation of the relationships between table

- A Brand has many Products, therefore this is a one-to-many relationship.
- A Category includes many Products, therefore this is a one-to-many relationship.
- A Cart has many CartItems, therefore this is a one-to-many relationship.
A Cart has one Order, therefore this is a one-to-one relationship.
A Cart is owned by a User, therefore this is a many-to-one relationship.
- CartItems are part of a Cart and linked to a Product, therefore these are many-to-one relationships.
- An Order is placed by a User and linked to a Cart, therefore these are many-to-one relationships.
An Order comprises many OrderDetails, therefore this is a one-to-many relationship.

- OrderDetails are part of an Order and linked to a Product, therefore these are many-to-one relationships.
- A Product is categorized under a Category and associated with a Brand, therefore these are many-to-one relationships.
A Product is included in many CartItems and OrderDetails, therefore these are one-to-many relationships.
- A User can have many Orders and has many Carts, therefore these are one-to-many relationships.

3.Screenshot of Jira Roadmap



4.Progression of the project.

First week: Table models, initializing the first data, creating login and signup routes.

Second week: Focus on Service and routes file for both users and admins.

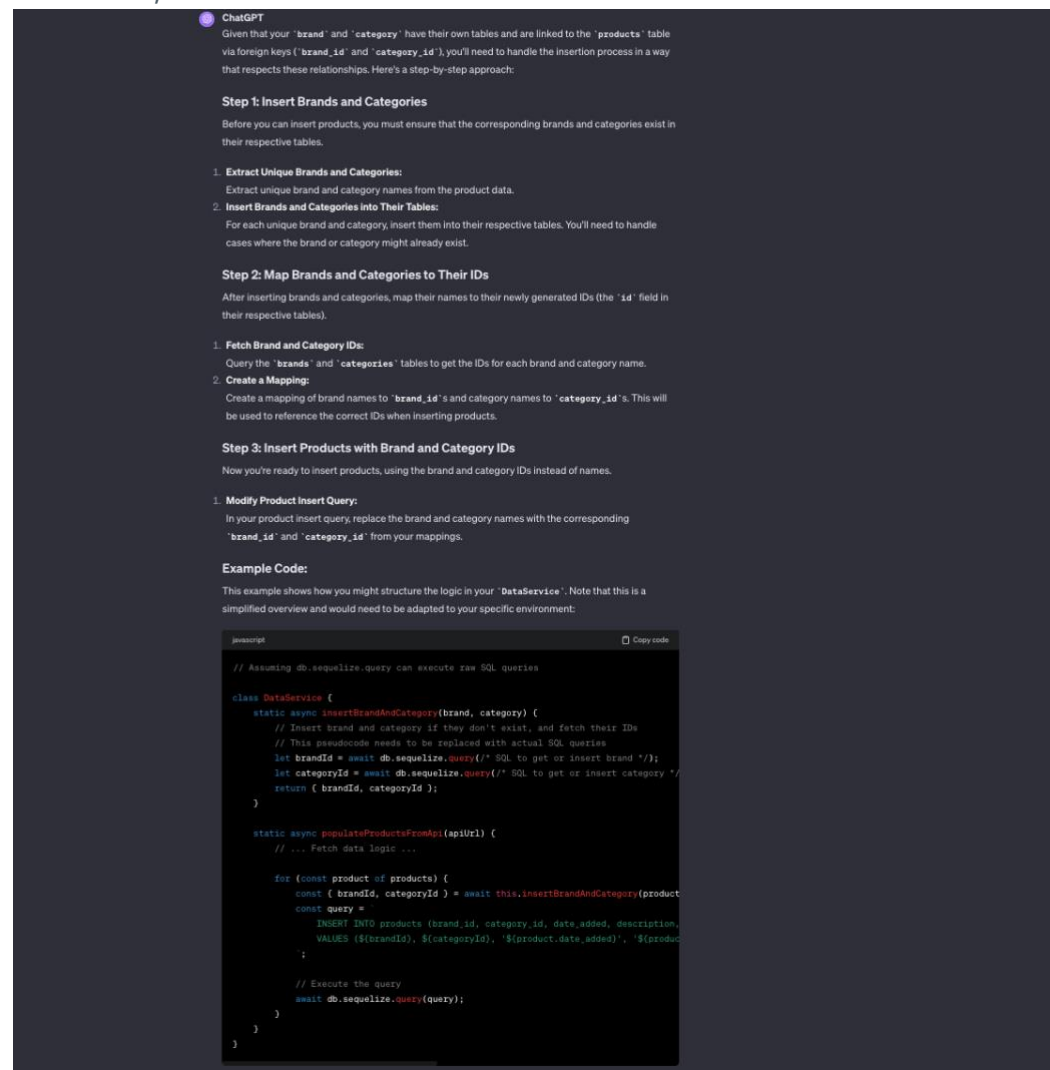
Third week: Front-end design for admin.

Fourth week: Testing, Swagger, Reflection

5. Challenges

This part will be associated with the Reference section in the README file .

(1). The first challenge arises when creating raw SQL for populating data into tables. Since the Products table has brandId and categoryId as reference keys to those two tables, determining how to retrieve the corresponding IDs became a challenge for me. After consulting with ChatGPT, I received a solution.



The screenshot shows a ChatGPT interface with a user query and a detailed response. The user query is: "Given that your 'brand' and 'category' have their own tables and are linked to the 'products' table via foreign keys ('brand_id' and 'category_id'), you'll need to handle the insertion process in a way that respects these relationships. Here's a step-by-step approach:". The response is structured into three steps: Step 1: Insert Brands and Categories, Step 2: Map Brands and Categories to Their IDs, and Step 3: Insert Products with Brand and Category IDs. It includes an example code snippet in JavaScript that demonstrates how to insert brands and categories into a database using Sequelize.

```
ChatGPT
Given that your 'brand' and 'category' have their own tables and are linked to the 'products' table via foreign keys ('brand_id' and 'category_id'), you'll need to handle the insertion process in a way that respects these relationships. Here's a step-by-step approach:

Step 1: Insert Brands and Categories
Before you can insert products, you must ensure that the corresponding brands and categories exist in their respective tables.

1. Extract Unique Brands and Categories:
Extract unique brand and category names from the product data.
2. Insert Brands and Categories into Their Tables:
For each unique brand and category, insert them into their respective tables. You'll need to handle cases where the brand or category might already exist.

Step 2: Map Brands and Categories to Their IDs
After inserting brands and categories, map their names to their newly generated IDs (the 'id' field in their respective tables).

1. Fetch Brand and Category IDs:
Query the 'brands' and 'categories' tables to get the IDs for each brand and category name.
2. Create a Mapping:
Create a mapping of brand names to 'brand_id's and category names to 'category_id's. This will be used to reference the correct IDs when inserting products.

Step 3: Insert Products with Brand and Category IDs
Now you're ready to insert products, using the brand and category IDs instead of names.

1. Modify Product Insert Query:
In your product insert query, replace the brand and category names with the corresponding 'brand_id' and 'category_id' from your mappings.

Example Code:
This example shows how you might structure the logic in your 'DataService'. Note that this is a simplified overview and would need to be adapted to your specific environment:

javascript
// Assuming db.sequelize.query can execute raw SQL queries

class DataService {
  static async insertBrandAndCategory(brand, category) {
    // Insert brand and category if they don't exist, and fetch their IDs
    // This pseudocode needs to be replaced with actual SQL queries
    let brandId = await db.sequelize.query(/* SQL to get or insert brand */);
    let categoryId = await db.sequelize.query(/* SQL to get or insert category */);
    return { brandId, categoryId };
  }

  static async populateProductsFromApi(apiUrl) {
    // ... Fetch data logic ...

    for (const product of products) {
      const { brandId, categoryId } = await this.insertBrandAndCategory(product.brand, product.category);
      const query = `
        INSERT INTO products (brand_id, category_id, date_added, description)
        VALUES (${brandId}, ${categoryId}, '${product.date_added}', '${product.description}');
      `;

      // Execute the query
      await db.sequelize.query(query);
    }
  }
}
```


So the solution: insert brands and categories => mapping the key:value pair of each tables, such as id:name=>insert products with brandid and categoryid.

(2).Insert admin table, my original code was

```
//insert Admin
var (password)=req.body
var salt=crypto.randomBytes(16)
crypto.pbkdf2(
  password,
  salt,
  310000,
  32,
  "sha256",
  function (err,hashedPassword){
    if(err){
      return next(err)
    }

    db.sequelize.query(`
      INSERT INTO Users(id,firstName,lastName,userName,email,encryptedPassword,salt,address,tlfNumber,totalPurchased,role)
      VALUES
      (1, 'Admin', 'Support', 'Admin', 'admin@noroff.no', '${hashedPassword}','${salt}', 'Online', 911,0,'Admin')
    `)
  }
)
```

The problem occurs when I log in with my admin account; the password doesn't work. I discovered that the populated hashed password has a length of 64 bytes when it should be 32 bytes. After discussing this with ChatGPT again, it seems that directly inserting raw data can sometimes lead to incorrect records

 ChatGPT

Given your specific situation and the lengths of `'hashedPassword'` and `'salt'` being stored as 64 bytes when directly inserted as BLOBs in your SQL query, it's clear there's an issue with how the Buffer data is being handled in the SQL query. Let's address this step-by-step:

Handling Buffers in SQL Query

When you interpolate a Buffer directly into a SQL string (using template literals), it might not be stored as expected. JavaScript implicitly calls the `toString()` method on the Buffer, which could result in an incorrect format for storage.

Solution: Binary Data Insertion in SQL

To ensure that the binary data is correctly inserted into your database, you should use parameterized queries. This approach prevents the accidental conversion of the Buffer into an unexpected string format.

Here's how you can modify your admin creation script:

```
javascript Copy code

var { password } = req.body;
var salt = crypto.randomBytes(16);
crypto.pbkdf2(password, salt, 310000, 32, "sha256", async function (err, hashedPassword) {
  if (err) {
    return next(err);
  }

  // Use parameterized queries to insert binary data
  try {
    const insertQuery = `
      INSERT INTO Users (id, firstName, lastName, userName, email, encryptedPassword, salt, address, tlfNumber, totalPurchased, role)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    `;
    await db.sequelize.query(insertQuery, {
      replacements: [1, 'Admin', 'Support', 'Admin', 'admin@noroff.no', hashedPassword, salt, 'Online', 911, 0, 'Admin'],
      type: db.sequelize.QueryTypes.INSERT
    });
  } catch (insertErr) {
    return next(insertErr);
  }
});
```

So the solution is to use parameterized queries.

(3) Checkout and AddItems function

These two functions are the most complex ones I have written for this project. I spent two whole days figuring out the steps and the logic within. After reading several articles, I realized that transactions are a good solution when dealing with functions that involve many logics and steps. This is because if one of the steps is incorrect, the entire transaction will be reversed, and nothing will be recorded in the database. However, I failed to implement transactions in the AddItems function. I found the reason but not the solution. The cartItems with a cartID reference key always return undefined, even though a cart was already created in an earlier step. When I write the same logic without a transaction, the function works as expected.

(4).Login form -Frontend

I wanted to add an alert to my login page but failed to extract the values from the form and use them in my JavaScript function, like so:

```
var username = $('#login').val();  
var password = $('#password').val();
```

However, this didn't work, so I asked ChatGPT again.

The code is slightly different from the one I provided in the actual project. The JWT token is stored as cookies, not in local storage, and this is done in the backend

```
javascript Copy code

document.getElementById('loginForm').addEventListener('submit', function(e) {
    e.preventDefault();

    const formData = new FormData(this);
    const formObject = {};
    for (let [key, value] of formData.entries()) {
        formObject[key] = value;
    }
    console.log(formObject);

    fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(formObject)
    })
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok');
        }
        return response.json();
    })
    .then(data => {
        if (data.status === "fail") {
            alert("Right username/password required");
        } else if (data.status === "success") {
            // Save the token to localStorage for future use
            localStorage.setItem('token', data.token);

            // Check the role of the user
            if (data.role === "Admin") {
                // Redirect to the admin main page
                window.location.href = '/admin';
            } else {
                alert("You are not authorized to view this page");
            }
        }
        console.log(data);
    })
    .catch(error => {
        console.error('Error:', error);
        alert('An error occurred during login');
    });
});
```

Thanks for reading !