Reflection report

by Martinus Nordgård

Documenting progress and decisions made throughout the final exam project.

Content of this report:

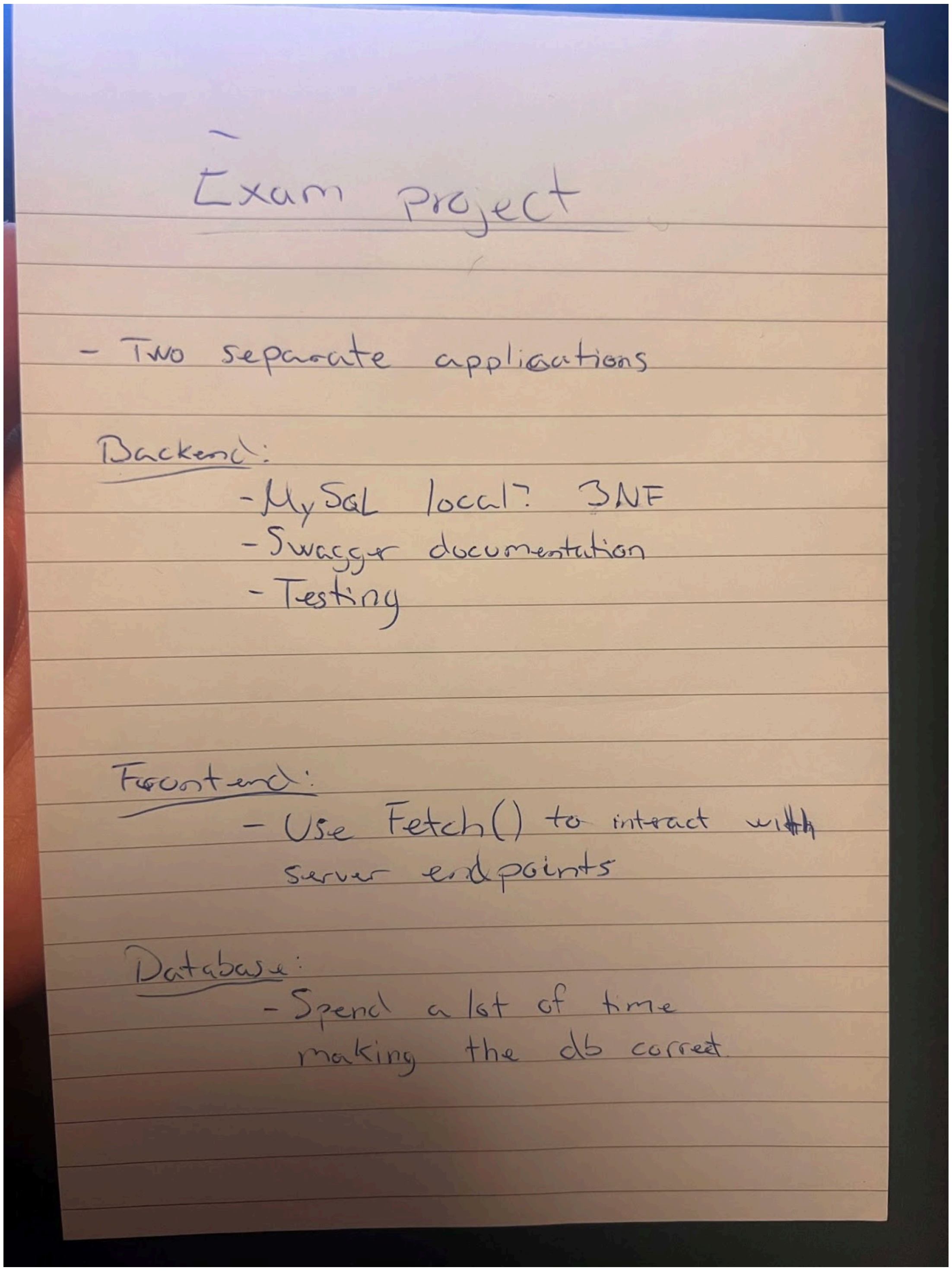
- Sharing personal experiences and thoughts through this project. This will serve as the discussion of progress and challenges encountered.
- Screenshot of the Jira Roadmap.
- Database ERD and Relationship explanation.
- .env example (also included in repo).
- Screenshot of my Postman Collection during development.

I remember well when we started the Semester Project in the middle of this course. I was quite nervous beforehand, and knew it had to be quite a challenge since it took four weeks to complete.

But this time it's different. Throughout this course I've built the confidence needed to take on new challenges, and knowing that they will be solved, one way or another. Each of our CAs have challenged me in different ways, and it's the consolidated knowledge acquired through these challenges have taught me how to handle the work at hand.

On day 1 I jumped right in. I downloaded the starter code and spent almost 2 hours just reading the requirements. My mind quickly got ideas of how each of the challenges could be solved. I revisited my own code from earlier CAs and confirmed that my ideas matched with my earlier solutions.

Parallel to this I noted down some quick thought which looked like this:



Exam project

- Two separate applications

Backend:

- MySQL local? 3NF
- Swagger documentation
- Testing

Frontend:

- Use Fetch() to interact with server endpoints

Database:

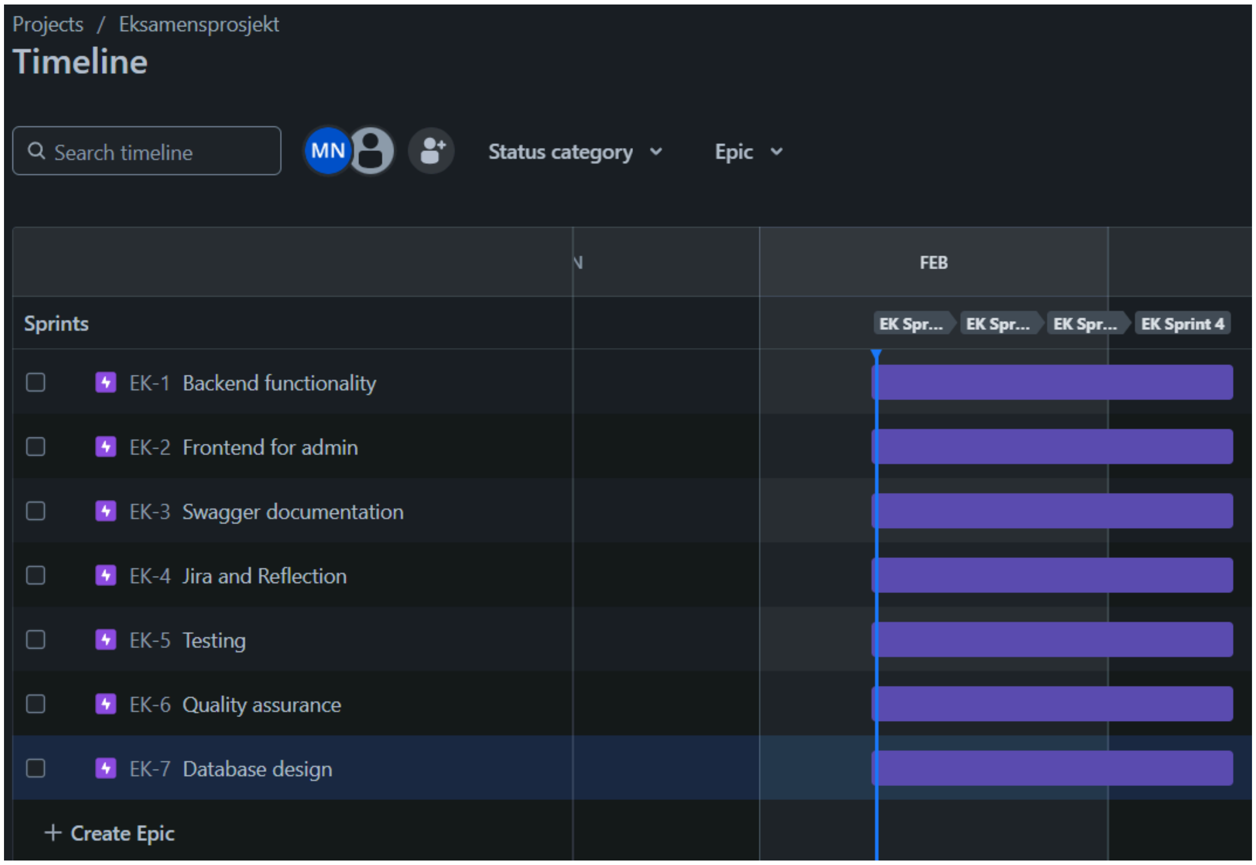
- Spend a lot of time making the db correct

I then made a Jira Project, some Epics and four Sprints. One Sprint for each week.
Each Epic would span across all four sprints, since the progression of the project was still unclear.

The Epics would consist of:

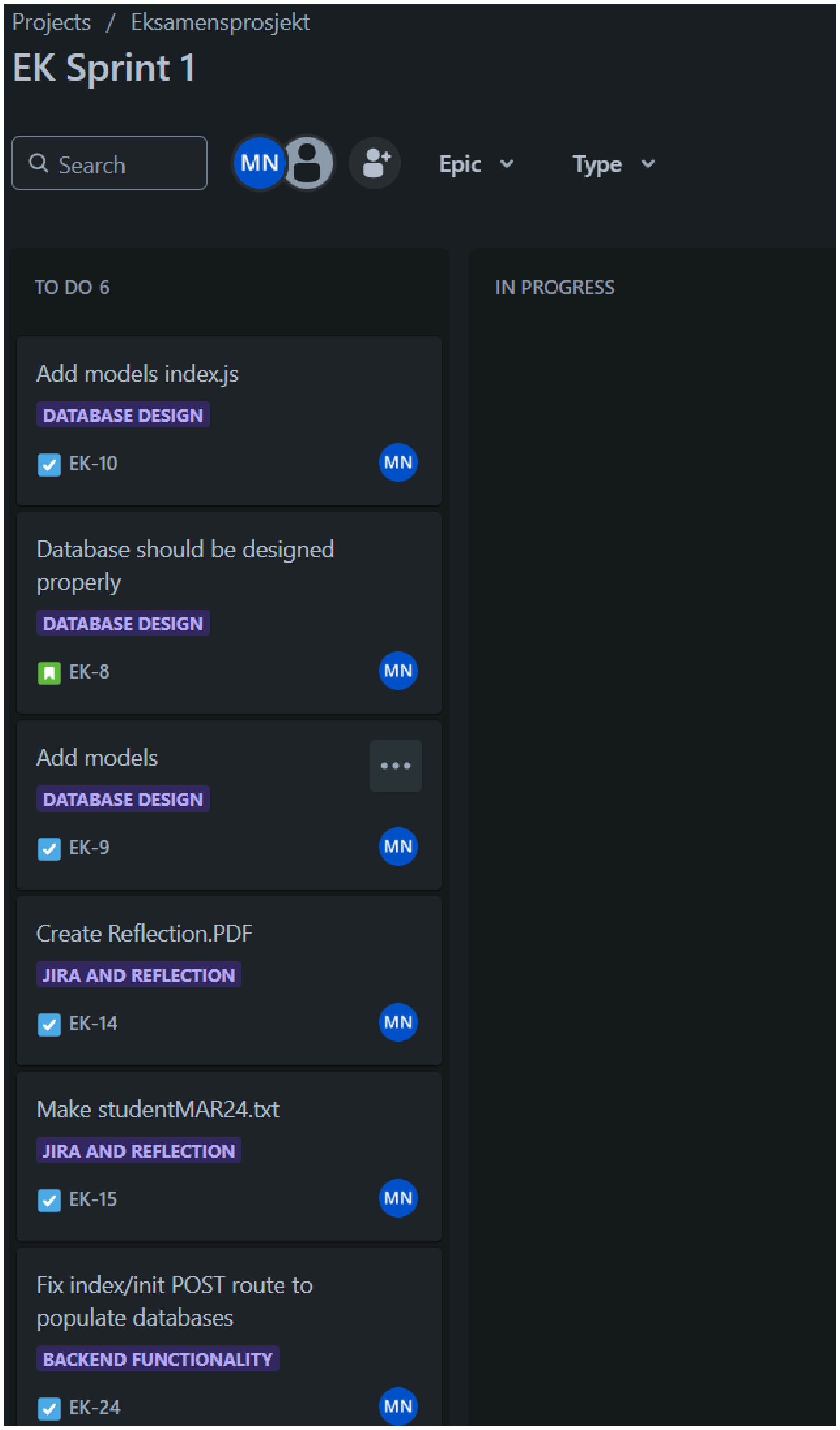
- Backend functionality
- Frontend for admin
- Swagger documentation
- Jira and Reflection
- Testing
- Quality assurance
- Database design

Jira roadmap:



EK-1 Backend functionality
EK-2 Frontend for admin
EK-3 Swagger documentation
EK-4 Jira and Reflection
EK-5 Testing
EK-6 Quality assurance
EK-7 Database design

Added some tasks and stories, and started the first Sprint:



Projects / Eksamensprosjekt

EK Sprint 1

Search

Epic

Type

TO DO 6

Add models index.js
DATABASE DESIGN
EK-10 MN

Database should be designed properly
DATABASE DESIGN
EK-8 MN

Add models
DATABASE DESIGN
EK-9 MN

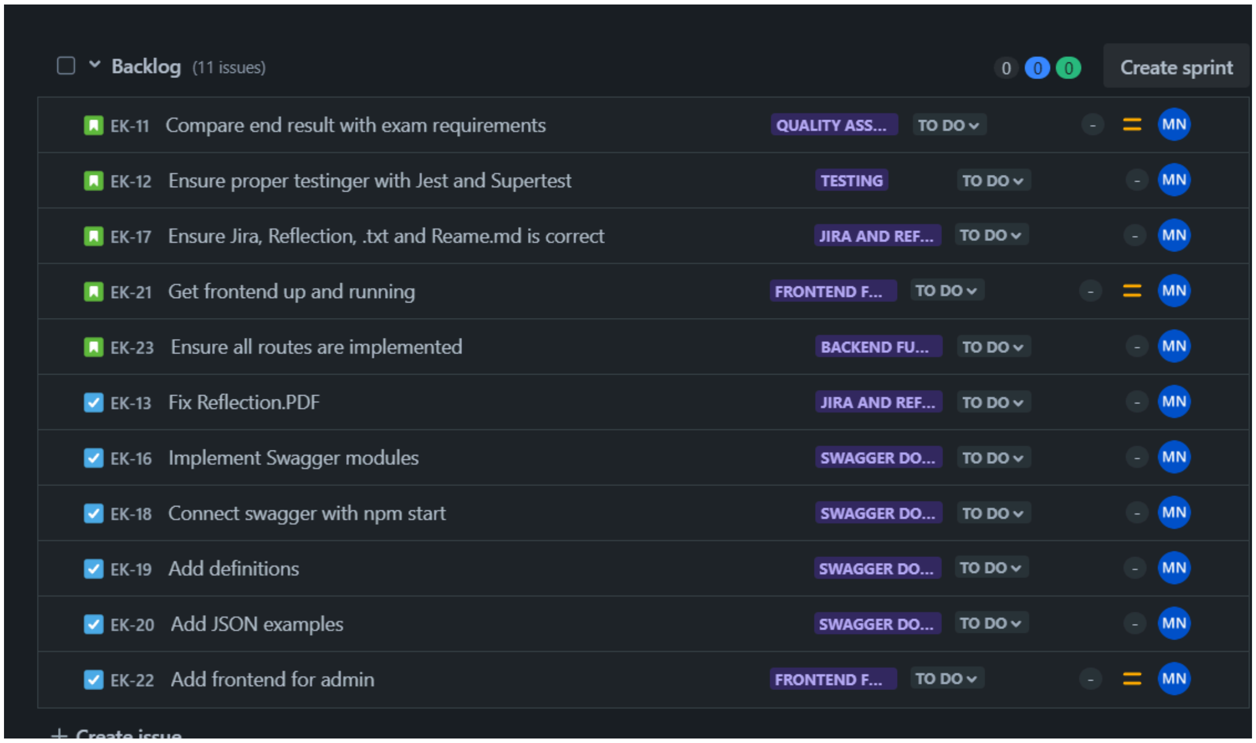
Create Reflection.PDF
JIRA AND REFLECTION
EK-14 MN

Make studentMAR24.txt
JIRA AND REFLECTION
EK-15 MN

Fix index/init POST route to populate databases
BACKEND FUNCTIONALITY
EK-24 MN

IN PROGRESS

With additional tasks and stories left in the backlog:



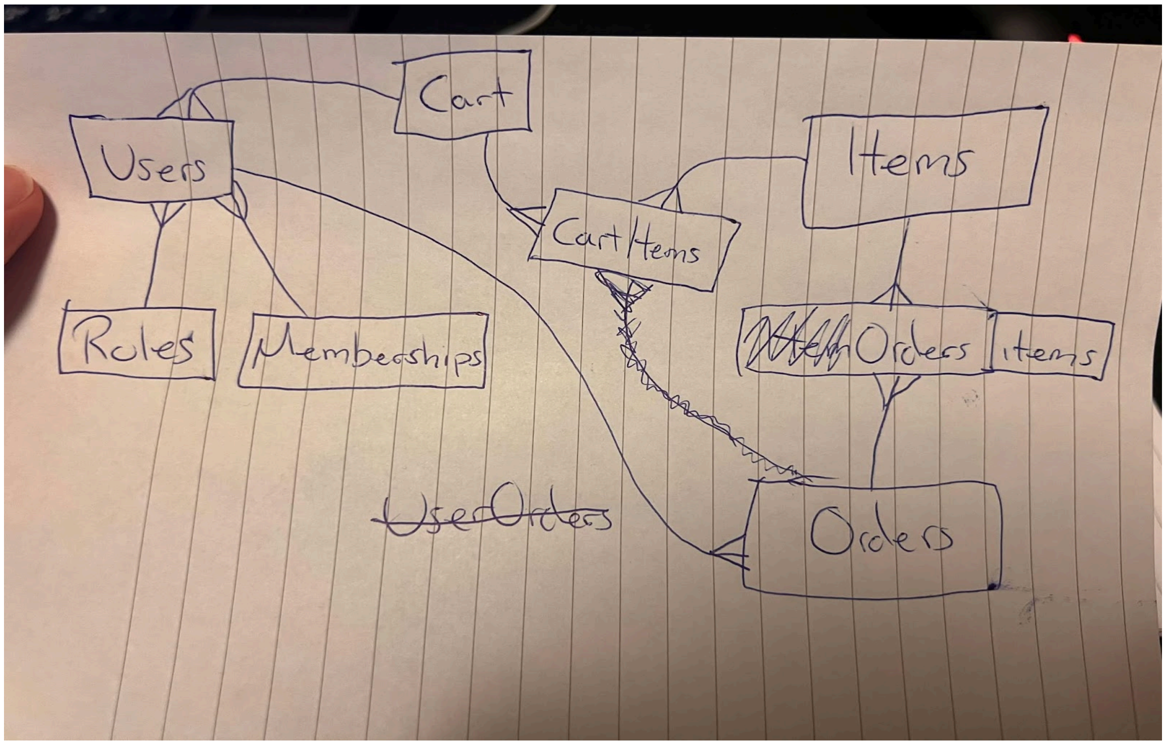
Backlog (11 issues) 0 0 0 Create sprint

EK-11	Compare end result with exam requirements	QUALITY ASS...	TO DO	MN
EK-12	Ensure proper testinger with Jest and Supertest	TESTING	TO DO	MN
EK-17	Ensure Jira, Reflection, .txt and Reame.md is correct	JIRA AND REF...	TO DO	MN
EK-21	Get frontend up and running	FRONTEND F...	TO DO	MN
EK-23	Ensure all routes are implemented	BACKEND FU...	TO DO	MN
EK-13	Fix Reflection.PDF	JIRA AND REF...	TO DO	MN
EK-16	Implement Swagger modules	SWAGGER DO...	TO DO	MN
EK-18	Connect swagger with npm start	SWAGGER DO...	TO DO	MN
EK-19	Add definitions	SWAGGER DO...	TO DO	MN
EK-20	Add JSON examples	SWAGGER DO...	TO DO	MN
EK-22	Add frontend for admin	FRONTEND F...	TO DO	MN

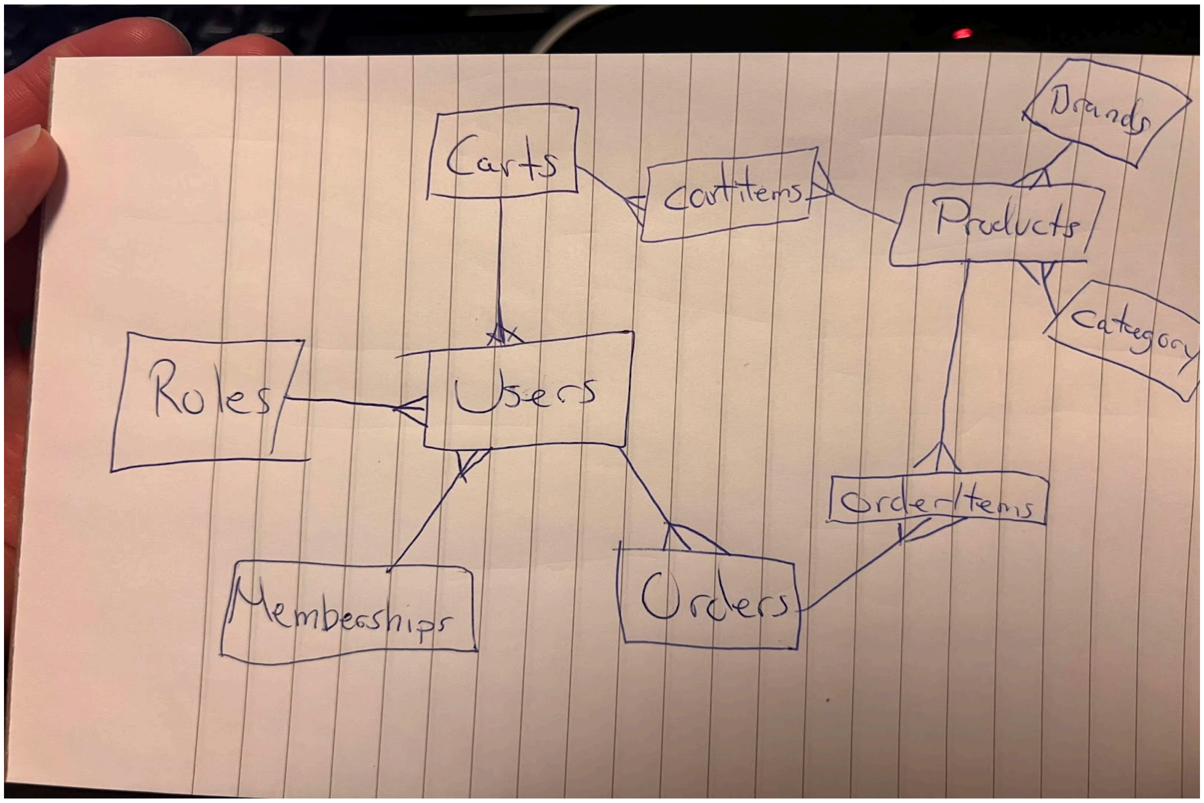
+ Create issue

Started fiddling with some ERDs

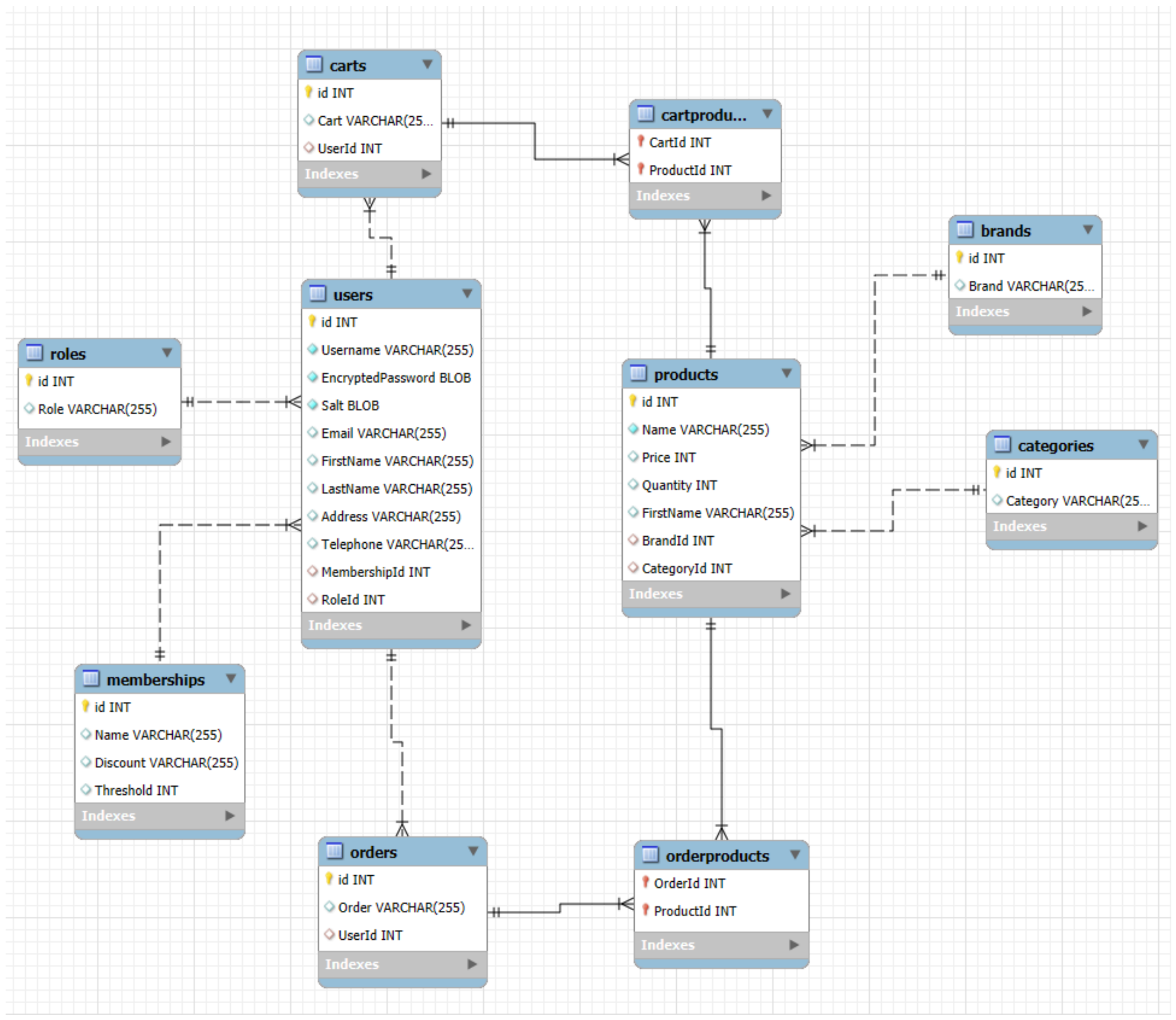
Version 1:



Version 2:

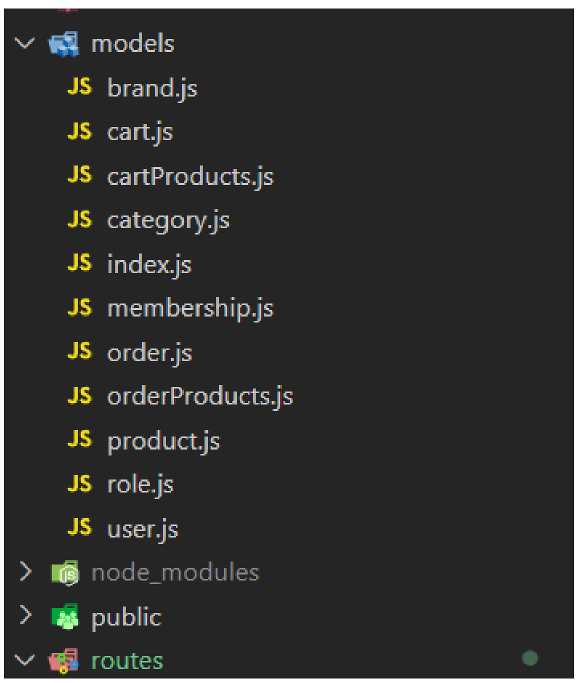


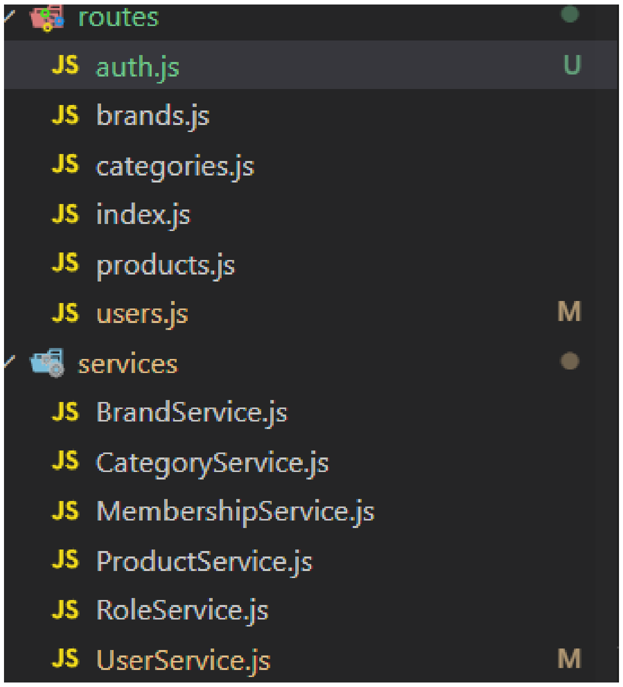
I must say one gets pretty excited when the MySQL ERD also looks like the ERD you sketched out:



The records aren't quite right yet, but the layout looks pretty promising so far.

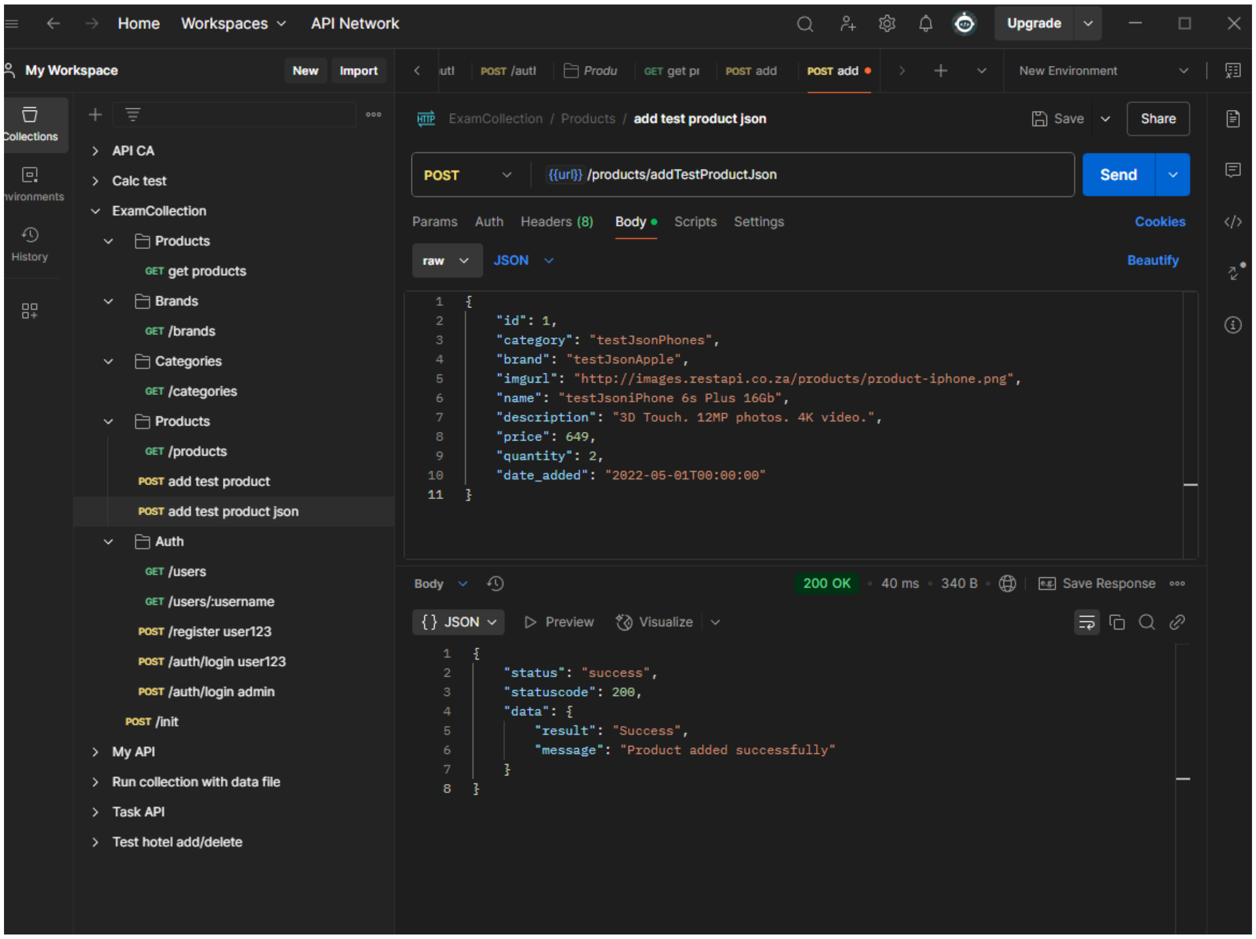
It's the middle of day 3 and I feel like I've make a lot of progress already:


```
∨ models
    brand.js
    cart.js
    cartProducts.js
    category.js
    index.js
    membership.js
    order.js
    orderProducts.js
    product.js
    role.js
    user.js
> node_modules
> public
∨ routes
```


```
routes
    auth.js                U
    brands.js
    categories.js
    index.js
    products.js
    users.js               M
services
    BrandService.js
    CategoryService.js
    MembershipService.js
    ProductService.js
    RoleService.js
    UserService.js         M
```

Many of the necessary files are in place, but with only a handful of endpoints made.

It's day 4 and I've make myself a Postman collection and made a lot of requests to test the API endpoints one by one, and tweaking what need to be tweaked:



POST {{url}} /products/addTestProductJson

```
{
    "id": 1,
    "category": "testJsonPhones",
    "brand": "testJsonApple",
    "imgurl": "http://images.restapi.co.za/products/product-iphone.png",
    "name": "testJsoniPhone 6s Plus 16Gb",
    "description": "3D Touch. 12MP photos. 4K video.",
    "price": 649,
    "quantity": 2,
    "date_added": "2022-05-01T00:00:00"
}
```

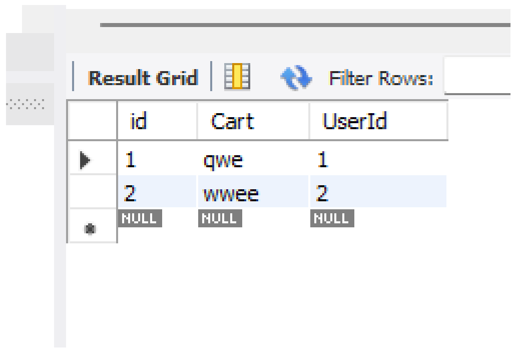
```
{
    "status": "success",
    "statuscode": 200,
    "data": {
        "result": "Success",
        "message": "Product added successfully"
    }
}
```

Day 5: I have ensured proper authentication functionality, and added middleware which differentiates between Admin and User.

Throughout the day, my mind has also been working on the database design. How doesn't one actually make use of carts, cartProducts etc.

Then it suddenly hit me that the concept of carts might be totally redundant. If a user only can have one cart, then the items in their "cart" might as well be linked with the UserId instead.

I added some data manually to the database to observe how it played out, and as you see in the screenshot below, id and UserId matches ("Cart" column is placeholder).



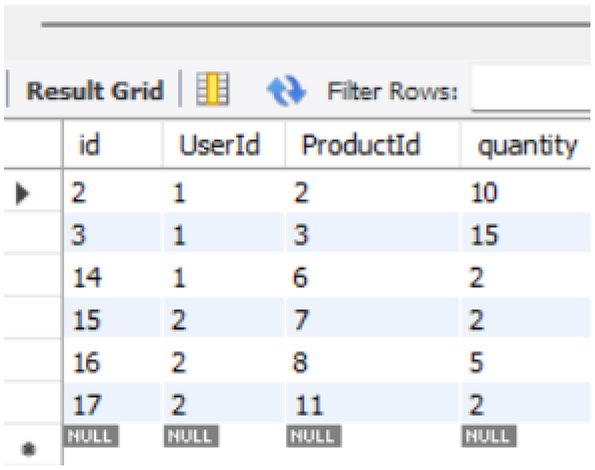
id	Cart	UserId
1	qwe	1
2	wwee	2
NULL	NULL	NULL

Week 4, day 3:
Okay, it's been a long time since I've written anything in this documentation file. After I started theorising about the database, I went deeper and deeper into the rabbit hole, and I've spent all my energy in progressing the project. It's crazy how many ideas you get and trying to keep track of these. I often need to write down ideas, because they might appear when you're not working on the project.

I actually went through with the "cartless" database design, which I believe was a good call. There might not be many pros or cons with having either design, but there is at least one less table to manage. Do not mistake this for not having a "cart", because they definitely do, it's just that their cart items are linked directly with their user ID, which will always be unique.

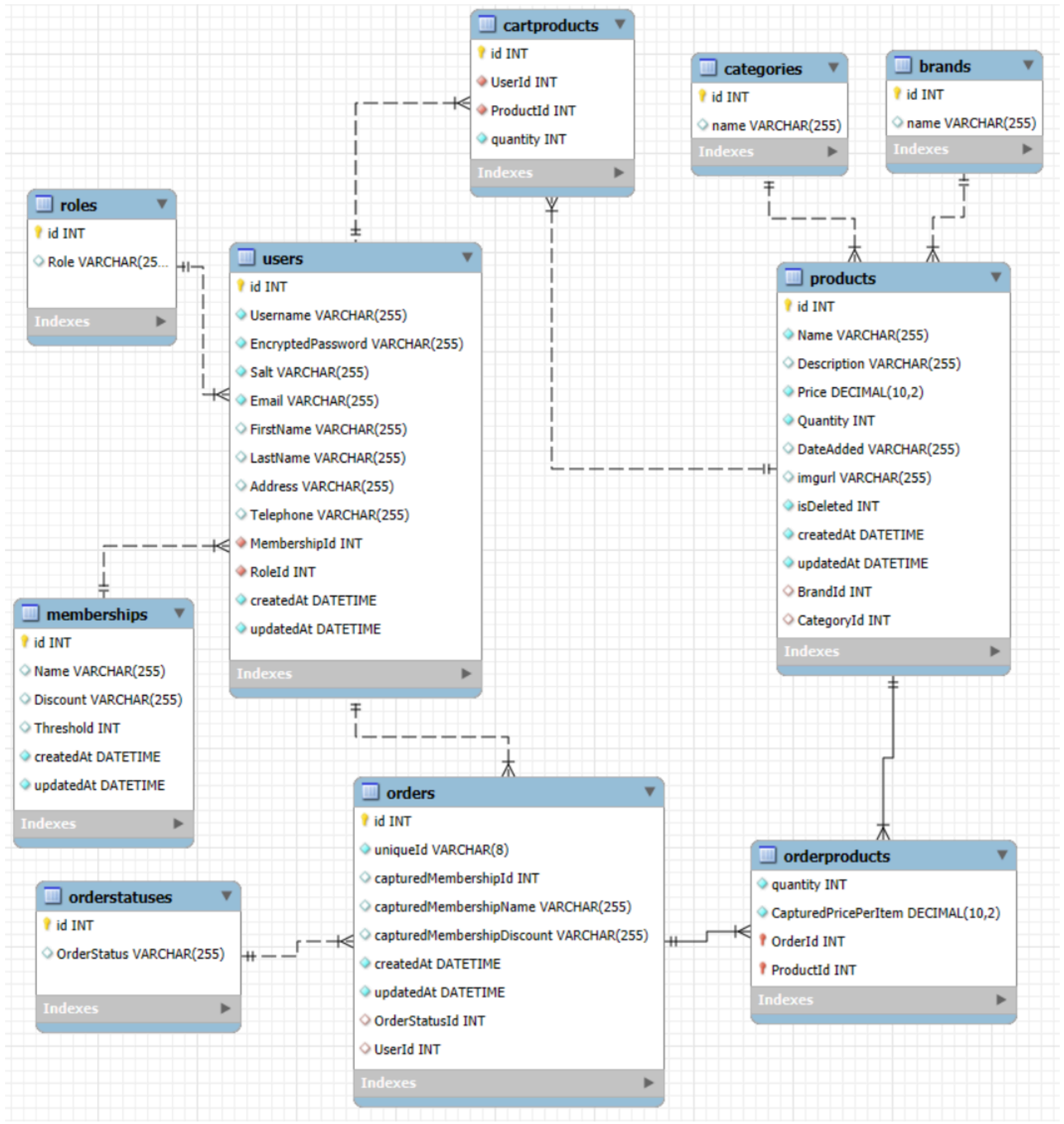
Here is a screenshot of the table “cartproducts”:

- One can see here that the table serves the purpose of a “cart” perfectly on its own. One loses the ability to have multiple carts per user, but I’ve never interacted with a webshop which practices that concept.



id	UserId	ProductId	quantity
2	1	2	10
3	1	3	15
14	1	6	2
15	2	7	2
16	2	8	5
17	2	11	2
NULL	NULL	NULL	NULL

My final database design ended up like this, and I will now try to explain the relationships as clearly as possible:



While designing my database, and the many iterations of it, I had to refer to the documentation of Sequelize multiple times
https://sequelize.org/docs/v6/core-concepts/assocs/

As well as this article about "Defining Many-to-Many Associations in Sequelize"
https://gist.github.com/elliette/20ddc4e827efd9d62bc98752e7a62610

Relationships in the database:

- Roles and users:
    - A user can only have one role, but one role can belong to many users. Therefore this is a one-to-many relationship.

- Memberships and users:
    - Same as roles. A user can only have one membership, while many users can have and share the same membership. Therefore this is a one-to-many relationship.

- Users and orders:
    - A user can have many orders, but an order can only belong to a specific user. An order cannot have many users. Therefore this is a one-to-many relationship.

- Order statuses and orders:
    - Each order has one specific status. Each status can belong to many different orders at the same time. Therefore this is a one-to-many relationship.

- Products and orders:
    - One product can belong to many different orders. One order can have many different products in it. Each order can contain many products and each product can belong to many different orders. Therefore this is a many-to-many relationship.

- Categories and products:
    - A product can have one category, and every category can belong to many products. Therefore this is a one-to-many relationship

- Brands and products:
    - Same as categories. Each product can have a brand, and only one brand, but each brand can be shared between many products. Therefore this is a one-to-many relationship.

- Users and products:
    - Each user can claim a product. They can in fact claim many products. Each product can be claimed by many different users. A user can claim multiple products and a product can be claimed by multiple users. “Claim” can be replaced with “Wish to buy” or “Make demand of”. “User” can be replaced with “Cart”. Therefore this is a many-to-many relationship

---

Back to the progression of the project.

At the end of the first week, I had made substantial progress with backend and database design. It was time to take a look at the frontend of the project.

I was confident, but also happily unaware of the work ahead. I dived into the front-end part thinking it would be pretty fast and easy. Boy was I wrong. Establishing the front-end side offered such a wide variety of different issues to solve and solutions to find.

I had already decided that I wouldn’t spend much time making it look nice, but rather focus on the functionality and make sure everything works according to the requirements. Since the requirements asked of us to use Bootstrap, I went with the vanilla bootstrap design, which actually looks quite pleasing. “Primary” and “Light” were my main colors of choice.

After completing the login page, the users would be able to log in. On successful login, the user would be redirected to the /admin/products view. On unsuccessful login, the user would not be redirected, but rather get an alert telling him the login was unsuccessful.

Okay, so what hinders the unauthorized user from just accessing the the /products view by just typing it in the url? Nothing!

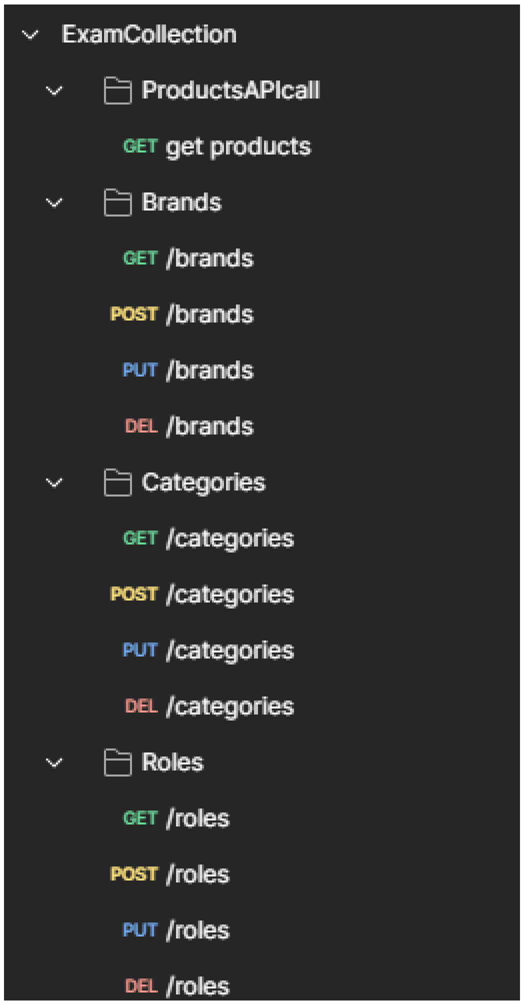
This needed to change, but how? The login endpoint returns a token, so I would have to make use of this to authorize. Let me make it a requirement to have a token to access the view. This of course resulted in a new problem, which allowed any registered user to access the view, since all authorized users get a token.

Now I needed to find out how I could differentiate between admins and normal users. I knew the answer lied in the token itself, because it was encoded with information. But this meant I would have to decode it, to get information out of it. And the frontend doesn't have access to the middleware on the backend, so this surely wasn't the path to go?

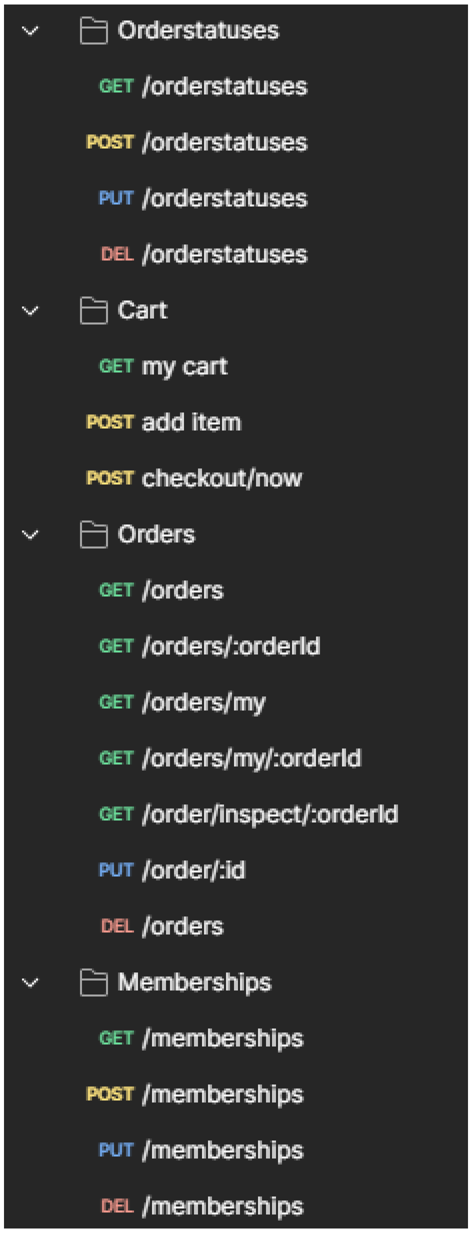
I theory crafted a bit, wondering if I could set a new cookie in the browser. This cookie would for example have the property of "isAdmin" and have a boolean value, and be set to 1 if the login was successful and the login was an admin. This could be achieved with the res.cookie('isAdmin", "1") method. I quickly realised how much of a security risk this would be. A user could just edit his own cookie to his/her liking, and gain full access to the admin dashboard. So this solution was binned pretty fast.

While being kind of tunnel visioned on the "access" part of the admin, I later realised that token decoding on the front-end was the way to go. It was easy to implement, and it could just reuse the code from the back-end.

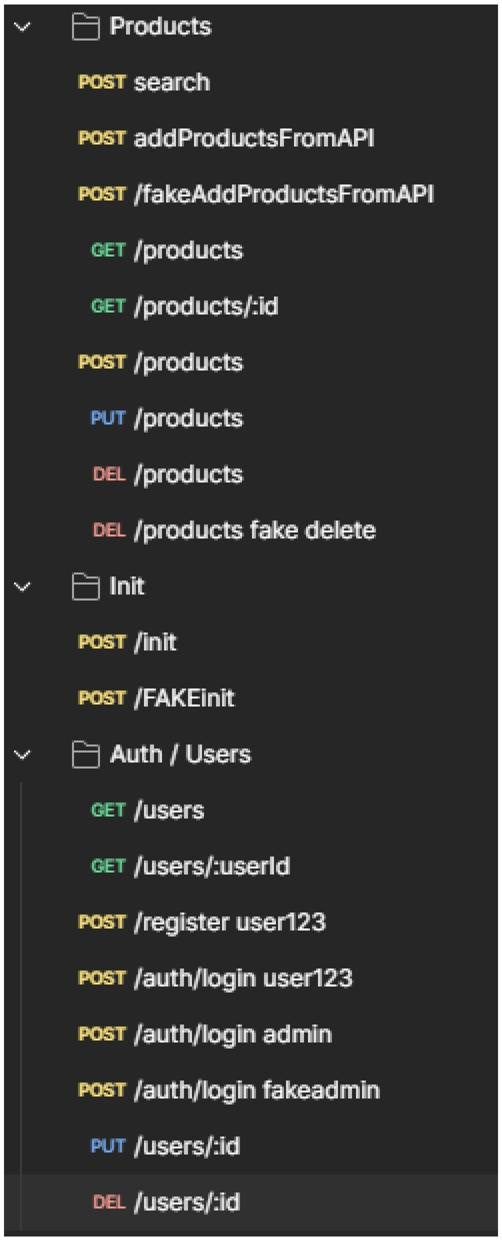
I made a comprehensive collection in Postman for testing all the endpoints in the API during development. The final edition looked like this:



ExamCollection
- ProductsAPIcall
  - GET get products
- Brands
  - GET /brands
  - POST /brands
  - PUT /brands
  - DEL /brands
- Categories
  - GET /categories
  - POST /categories
  - PUT /categories
  - DEL /categories
- Roles
  - GET /roles
  - POST /roles
  - PUT /roles
  - DEL /roles



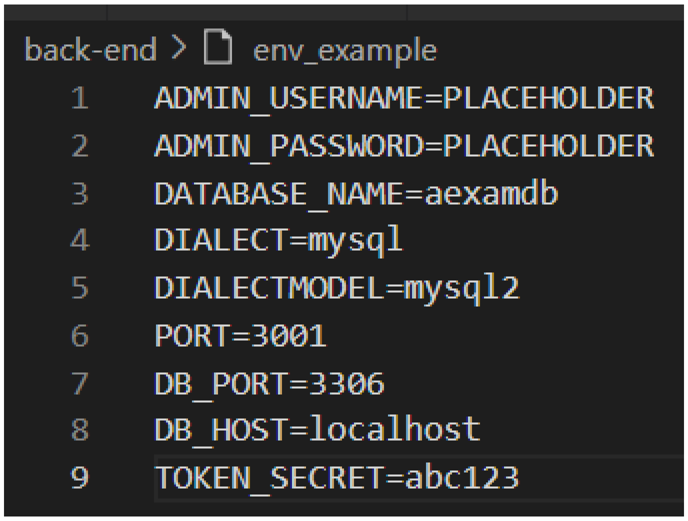
```
Orderstatuses
    GET /orderstatuses
    POST /orderstatuses
    PUT /orderstatuses
    DEL /orderstatuses
Cart
    GET my cart
    POST add item
    POST checkout/now
Orders
    GET /orders
    GET /orders/:orderId
    GET /orders/my
    GET /orders/my/:orderId
    GET /order/inspect/:orderId
    PUT /order/:id
    DEL /orders
Memberships
    GET /memberships
    POST /memberships
    PUT /memberships
    DEL /memberships
```



Products
POST search
POST addProductsFromAPI
POST /fakeAddProductsFromAPI
GET /products
GET /products/:id
POST /products
PUT /products
DEL /products
DEL /products fake delete
Init
POST /init
POST /FAKEinit
Auth / Users
GET /users
GET /users/:userId
POST /register user123
POST /auth/login user123
POST /auth/login admin
POST /auth/login fakeadmin
PUT /users/:id
DEL /users/:id

Example of .env files.

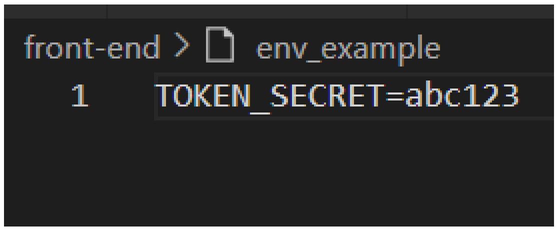
Example of .env file for back-end folder:



back-end > env_example

```
ADMIN_USERNAME=PLACEHOLDER
ADMIN_PASSWORD=PLACEHOLDER
DATABASE_NAME=aexamdb
DIALECT=mysql
DIALECTMODEL=mysql2
PORT=3001
DB_PORT=3306
DB_HOST=localhost
TOKEN_SECRET=abc123
```

Example of .env file for front-end folder:



front-end > env_example

```
TOKEN_SECRET=abc123
```

TOKEN_SECRET in both .env files need to match. This is to ensure correct decoding of tokens.

Final thoughts:

- This project was quite a ride. The difficulty was as I expected. Not too hard, but not too easy either. A lot of logic problems had to be solved, systems needed to work well together, and errors made in the early stages in development manifested and punished you later in the project.

- The sheer amount of “grunt work” required in the project was also an insightful experience. Creating this many routes, service files and models, as well as all the front-end views, routes and javascript files, demanded a lot of work. It teaches one to become “one” with the code and really understand the system on a deeper level.

- The concept of project management and time investment is also an interesting topic of reflection. Having such a big task and four weeks to complete it, it’s no easy task to spend the time wisely and keep track of how far you’ve come. Throughout this project I’ve always felt comfortable and ahead of schedule, but at the time of writing this, week 4, day 5, shows that projects take longer than you think.

- The project itself has definitely been an enjoyable experience. I’ve felt a lot of mastery, but also frustration. Solving big problems turns all the helplessness into pure happiness and it’s such a good feeling.

- A lot of the work is actually done while not sitting with the computer. Ideas and solutions often come to mind when you’re driving or taking a walk. Who hasn’t experienced an epiphany in the midst of a shower? Staring at the screen and struggling isn’t always the right medicine.

- I could have written a lot more about the project decision and reflection, but it’s wise to moderate and keep it within reasonable bounds.

Thank you for taking the time to grade my exam project.

Best regards,
Martinus Nordgård