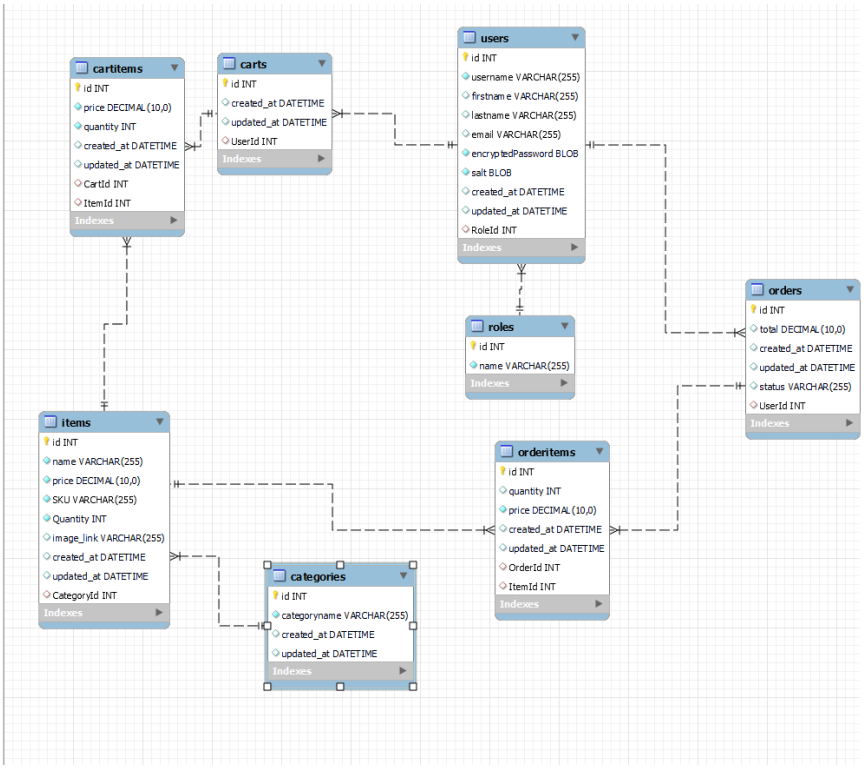
EXAM PROJECT RAPPORT:

The project was designed to connect users to easily purchase products off an items cataloge, this is built for stock-control and sales for a warehouse.

Database:



This is how my database turned out.
8 tables with various relationships:

One to many:

- Role table has a one-to-many relationship with the User table. One role can have many users, but users can only have one role.
- Category and items have a one-to-many relationship. Where each category can belong to many items, but one item may only have one category.
- Items and cartitems have a one-to-many relationship. Where each item may go in many cartitems
- Carts and cartitems have a one-to-many relationship. Where each cart can have multiple cartitems, but one cartitem may only go in one cart.

- Items and orderitems have a one-to-many relationship. Where each item may go in many orderitems
- Order and orderitems have a one-to-many relationship. Where each order can have multiple orderitems, but one orderitem may only go in one order.

One to one:

- Cart table and User table has a one-to-one relationship. Where one user may have one cart, and one cart can only belong to one user.
- Order table and User table has a one-to-one relationship. Where one user may have one order, and one order can only belong to one user.

Many to many:

- Cartitem has a many-to-many relationship with Cart and Item. Where cart can have multiple cartitems and each item can go in multiple cartitems.
- Orderitems has a many-to-many relationship with Order and Item. Where order can have multiple orderitems, and each item can go in multiple orderitems.

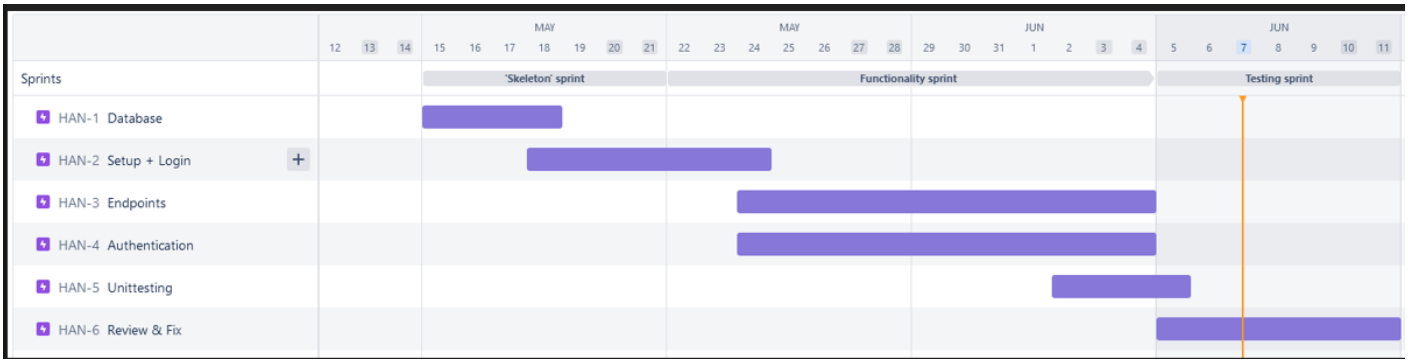
Progression:

For this project I decided to split it into 6 epics:

- Database
- Setup + Login
- Endpoints
- Authentication
- Unittesting
- Review & Fix

Each epic contained various numbers of issues, which i separated into problems that allowed me to make a chunk of code that was easily done in one go, whilst still making sense as a stand lone piece of code. The code blocks in these issues would also be short enough that problem solving on them should be fairly doable when a problem arose.

For a total of 3 sprints:



Skeletonsprint, Functionality sprint and Testing sprint.

I chose this sprint setup as it allowed me to make the base of the project first, before moving on to the core functionality part of the project. Rounding it off with tests, small fixes and anything that I would deem to be an improvement on my code.

Skeleton Sprint:
Starting off with setting up the database tables, so I could move on to the setup to populate said database. This allowed me to have, test and look for improvements in my table relationships.
This sprint caused no major problems, only a few smaller ones including:
One mistake here which was to not include image_link in the original items database, which caused me to having to add it later on and forcing me to rewrite a bit of code.

Looking back I would probably spend more time getting the database correct from the start, I made a very open ended database where not everything was connected where it should be, which caused me to have to rewrite and retest various functions more than once.

Functionality Sprint:
This is where the main functionality of the project was done. Here a few errors occurred which took me a while to figure out a solution for, amongst other I had to look at external sources(links in readme) to create the search endpoint, how do use findOrCreate (used in both order/id endpoint and post cart_item endpoint).

For the post order/:id endpoint where Items need stock change, order needs price update and cartitem should be removed I wanted that either none or all went through so I used sequelize transaction.

Looking back I am quite happy how I went about this challenge, the issues I had was either where I had to do more research or where I had to fix my database setup. So this sprint itself, whilst being the hardest one, was planned fairly well.

Testing Sprint:
The final sprint of the project which I honestly expected to use less time on, the unittesting was quickly set up but I ran into a few potential hickups when trying to run the unittests after I had done a lot of

update and adding into the database so I had to revist the tests to add various redundancies should something fail. The reason why I chose to go back and add a few extra steps is because I wanted to make sure the tests would be usable even after its launched. I did not find a way to make the search testing more proof as with manipulation the database could be both greater or lesser than initial number.

This is also the sprint where I did my documentation in postman, which had the positive sideeffect of me being able to pick up slightly “niche” bugs.

- Image_link not present in my post item endpoint.
- No stock check on item_cart post when adding the same item more than once
- Missing errorhandling in Search endpoint
- Could delete catagories where items used that PK.

Plus a few more.

Looking back at it now I think my setup was mostly correct, however I would have done it slightly differently now. For one I would do the documentation as I moved along with the project, putting it off til the end forced me to find errors late that could’ve taken less time to fix if I catched them earlier.

Overall im pleased with how this project turned out. There is a few things I would like to fix that I didn’t end up doing such as:
Converting all singular endpoints into the index route, which would see me end up with these routes: ALLCART + ALLORDER + CATEGORIES + ITEMS + ORDERS into index.js, leaving me with:

auth.js
cart_item.js
cart.js
category.js
index.js
item.js
order.js
setup.js

This would reduce my number of routes.js files from 13 to 8 which would reduce clutter.

I also started to move updated_at to +2 (to match my timezone) but I did not have time to finish it, leaving item and categories updated_at to be 2 hours infront of everyting else. With more time that would be one to fix.