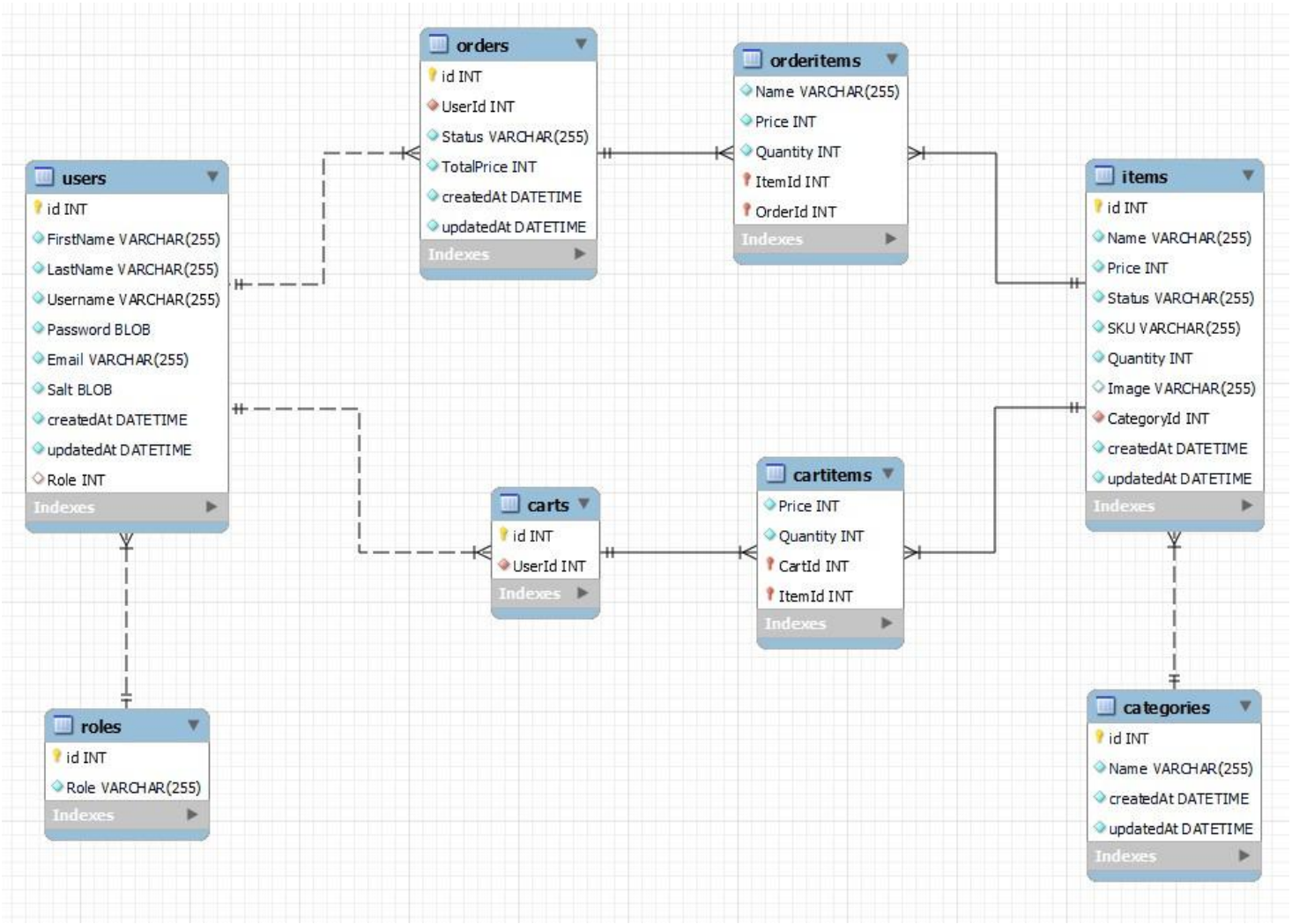
Postman documentation

https://documenter.getpostman.com/view/25181873/2s93mBwyw9

The pdf format isn't showing all the examples (apparently they can't be printed as pdf), but is included as requested.

EER of tables and their relationships



Role has many User (One-to-many);
Category has many Item, Item belongs to Category (One-to-many);
Cart belongs to User (One-to-one)
User has many Order, Order belongs to one User (One-to-many);
Super many-to-many relationships:
1- Item/Cart:

- Item belongs to many Cart (through CartItem) , Cart belongs to many Item (through CartItem); (many-to-many)
- Cart has many CartItem, CartItem belongs to Cart; (One-to-many)
- Item has many CartItem, CartItem belongs to Item; (One-to-many)

2- Item/Order:

- Item belongs to many Order(through OrderItem), Order belongs to many Item(through OrderItem);
- Item has many OrderItem, OrderItem belongs to Item; (One-to-many)
- Order has many OrderItem, OrderItem belongs to Order; (One-to-many)

As the Sequelize documentation explains, a super many to many relationship is the best of two worlds and allows for any kind of eager loading and deeply nested includes.
(Don't worry, I've used the raw queries where requested).
Plus, it kind of was set to be like this,as the project document requests orderitems/cartitems itemId to be referencing the items ids from the item table.

ROADMAP with Epics and stories

Sprint

- ESSP-1 As the owner of the stock-control system I want a relational based database, so that data can be stored and easily retrhieved
  - ESSP-2 Create main backend fundaments of the application such as tables and their relationships (sequelize ORM)
  - ESSP-3 Implement utility /setup endpoint for the first population of roles, admin user, categories and items from Noroff API
- ESSP-4 As a user I want to be able to interact with the data, so that I can rethrieve or modify it
  - ESSP-5 Implement authorization with JWT and signup login endpoints
  - ESSP-6 Implement category and item endpoints
  - ESSP-7 Implement Cart and cartItem endpoints
  - ESSP-8 Implement Order and orderItems endpoints
- ESSP-9 As the developer of the stock-control system I want to implement testing, so that I can make sure that all works as intended
  - ESSP-10 Do some testing of all endpoints manually, triple check that all requirements are met
  - ESSP-11 Implement unit testing with Jest and Supertest as per requirement
- ESSP-12 As a developer of the stock-control system I want to document the system and its api endpoints, so that it easily understandable how it works
  - ESSP-13 Add detailed postman documentation with examples for each endpoint available + a detailed readme for installation and usage
  - ESSP-14 Write a retrospective report of progression and challenges met with the project, how the relationships of the tables are set and work and ss of the roadmap

Retrospective on progression

My planning through scrum sprints was in the beginning a bit messy, as I attempted to make it a structured plan with all the requirements from the project documentation as tasks belonging to stories. But after some thorough thinking I landed on a simpler set up which was easier to follow, and as per the requirements, I just went back reading the project documentation thoroughly to spot every single requirement that has been spread around various parts of the document.
The progression went well, the second sprint was started earlier than it was set for, but took a little longer, in general though I managed to respect my 3 weeks time limit that I set with 1 week timeframe per sprint.

Challenges faced during the development of the project

One first challenge was, as earlier described, laying properly out a progression plan.

The next challenge met thereafter, was fully understanding, once again, the 3N form and how to not overthink it at the same time. I found a good and simple article describing in detail how to achieve the 3n form (it is linked underneath this section in Sources used for this project).

Setting up the simplest of signup/logins was also a bit challenging, as the documentation hinted multiple times at tokens being provided upon signup, which does not make much sense since they expire after 2 hours, so I've set them to be created and provided upon a successful login.
My login method is as simple as it gets, other than the express bundled crypto package no other packages were implemented for it. I took such a decision since no sessions and session storing was required from the customer. I deemed it then to be a good choice to use cryptos simple functionalities to create and store, upon signup, hashed/salted passwords and upon login, reverse check the encryption through timingSafeEqual and Buffer.from (this was the part I had to read a bit about, link is added in sources).
I think the result is exactly what the customer would request and I believe that should sessions or a more complex login system be implemented, this could be set with a new request from the customer. I would also dare think this to be a frontend part of the job as the secure creation, storage and retrieval of login information is already in place.

Another fun but a bit challenging part, as I hadn't dealt too often with such an issue previously, was the use of the reduce method to rearrange the arrays' results retrieved from the raw queries of allcarts and allorders.
Once understood and achieved the desired result though it felt easy and straightforward to use such a method. (sources for this part are also linked underneath).

Not a challenge, but parts that I went researching (refreshing my memory) on, in order to fully understand the best practices of them, were the use of associations, hooks and operators in sequelize. (also linked in the sources part underneath).

To conclude, my main biggest challenge was probably having to interpret the document, its wording and all its entangled requirements.
Some requirements are contradictory to one another, others are specific in some parts and vague in others parts.
There are multiple incongruences between what is stated in the whole documentation with what is required through the API requirements.

To make an example of what I mean with those comments, I could reference for example the users should see complete orders only (and orders history), in 2 of the sentences you refer to this very specifically but then in the last sentence, you state that Users should see all their orders. I decided to

interpret that last sentence as, users should be shown order status, but only admin (a type of user as well) should see all types of orders and their relative status.

Another good example of vaguely set specifications was the POST order/:id which didn't really give any hint on what that parameter was meant to relate to, it actually took several students asking about this on chat channels to actually have this outlined and explained.
For instance, it couldn't possibly be an order id, as this isn't created anywhere else prior to that endpoint. Furthermore, as I explain in my readme and documentation, I find it kind of weird that this should be set up as specified by the teacher in the chat channels.
Why set up a backend that requires another developer working on the frontend of the system to set a loop on an endpoint call, when the same results could be achieved by having the backend developer setting the endpoint to automatically do a complete checkout/order of all the items in a cart, with one simple call.

I understand this, as it has been echoed by the teachers several times, is an attempt at creating a "real-world" type of scenario for us students, where a customer might or might not be a developer and might or might not be completely clear with the requests in the project specifications.
But, as this exam document both gives very strict rules on how some parts should be built and at the same time is vague and at times even has requirements contradicting each other, it doesn't really fit into one nor the other type of "customer" one might meet under professional work circumstances.
If the customer hasn't got development experience, there probably wouldn't be such strict requirements. On the other hand, if the customer is actually a developer, I believe the requirements wouldn't have the prior mentioned flaws.

Please look at this as constructive feedback, I mean no disrespect with what I'm stating here.
I, as a student, was looking forward to having the opportunity to showcase how much I've learned during this year of studies, hoping to be given some main project requirements, such as the ones that an internship project would have had, and then build a personal project based on those main requirements, build it to be as efficient and well structured as I possibly could, while maintaining the main requirements and using concepts and methods/technologies that were thought during this year.

Instead I'm left with a feeling of being given a project that wasn't fully quality assured, restricts my possibilities to showcase, at full, my acquired knowledge (since most requirements points straight to how and what to do) and rather sets too much weight on the students capability of interpreting the requirements given and understand what the "customer" really meant/desired with them.

I understand that this is a vital skill to have as a developer, but I believe that in an exam like this one, the evaluation of my developer knowledge and skills, gathered through the year, should be prioritized above all other types of soft skills that a developer should have and obtain through work experience and practice.

Sources used for the project

- Various module code examples from BED courses
- For proper understanding of the 3N form: https://simplesqltutorials.com/third-normal-form/
- Associations and hooks:
    - https://sequelize.org/docs/v6/core-concepts/assocs/
    - https://sequelize.org/docs/v6/other-topics/hooks/
- Use of operators: https://sequelize.org/docs/v6/core-concepts/model-querying-basics/#examples-with-opand-and-opor
- Understanding the buffer for deciphering passwords on login: https://stackoverflow.com/questions/66226092/how-to-use-buffer-from-with-crypto-timingsafeequal
- Supertest basics: https://github.com/ladjs/supertest
- Reducing results from raw queries:
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce
    - https://stackoverflow.com/questions/47840445/js-reduce-array-to-nested-objects
- My brain and a lot of trial and error.