Database 3 normal form:
Junction tables named with merged name of tables relating.

Table	**Table related**	**Relationship**
User hasOne UserMembership	Membership hasMany UserMembership	One user has one Membership, one membership belongs to many users.(many to one)
User hasOne UserRole	Role hasMany UserRole	One user har one Role, role belongs to many users (many to one)
User hasMany UserOrders	Order hasMany UserOrders	Many users have many orders. (many to many)
Cart belongsTo user		One user has one cart (one to one)
Cart belongsToMany Product, through CartProduct	Product belongsToMany Cart, through CartProduct	Many carts have many products. (Many to many)
Product hasOne ProductBrands	ProductBrands belongsTo Brand	One product has one brand (one to one)
Product hasOne ProductCategories	ProductCategories	One product has one category. (one to one)
Order hasOne OrderProgress	OrderStatus hasMany Orderprogress	Many orders have one OrderStatus. (One to many)

Denormalisation on orders as neither productid, price, membership might be valid in future. Only captured for information purposes for customer in future.

Project reflection :

I panicked a little in the beginning and setup the db to just be all many to many relationships(I did not sleep the three first days)
I started my approach to the setup as a tree model– work with one type first, fx sequelize and then onto service files, then endpoints. In retrospective I would have worked in a bush setup, work with one branch/model and then work out from there to the endpoint. I put in 3 days for planning and looking at tables and relationships. I wish I decided variables and naming before starting as this cost me a lot of time in the end where I just did not have consistency to make everything work smoothly in the end phases of the admin endpoints and testing.

Setting up Jira :
Creating rough sections for planning sprints, trying to stay ahead of schedual at all times, just in case.
I forgot to put in the middleware in Jira, this was later added when I started this section.

Sequelize :
I decided to put most weight on the database setup to make everything work “on its own” I also wanted to focus the most on the database section of the app as I felt this is the topic I excelled the least in through my studies.
Tested paranoid tables, instead of creating timestamps manually, however it created a lot of error

messages and I dedicated too much time to work around. Error messages was related to date and node and seem to be a big problem in recent time.
After creating REST API, I changed /init and /cart/checkout/now to transactions as its unfortunate if one part of the endpoint do not work out and I end up with “loose data” in the db, although bulkCreate is a transaction in itself, Its better to secure the entire endpoints data altering. Changing to bulkCreate made me aware of the triggers not working and sorted this accordingly.
I also tried to use upsert(), but found a lack of documentation on how it worked offputting and went for other solutions instead.

Frontend/Endpoints:
Validate as much as possible human related in sequelize(like registration and login), but also require input on all from frontend.
Changing the initial db setup in development of app, I forgot that admin cant “just be put in” the db, because of the salt/hash on login, Admin had to be posted to the endpoint and the be altered to Admin(role 1) after.

Other:
Ive tried to stick to documentation and school notes as much as possible, but I had to cave on a few things along the journey, I like to pretend Im “at work” while doing the exam and do as much as possible on my own. Most of my googling has just ended being my own spelling errors :p

For the future I would like to set clear rules for development variables, and naming.
I would personally like to work more on promises and data manipulation as I feel these are now my weakest sides, not necessarily solving problems with them, but doing it the most efficient and cleanest way – wich I obsess to much over.

My strengths through this exam:
solving problems – not once have I thought : I dont know how to this. I rather think : I dont know wich method to pick.
Fixing issues along the way – I always know where the problems occur and can locate and ifx them fast.
Striving for the best code (this is also a weakness) – I always get exited when I figure out ways to do things better and will go back to alter code to look its best/ cleanest and perform best, however this can also be a weakness as leaving code as is, is a problem for me feeling an intense need to tweak constantly.

My weaknesses:
This reflection report.