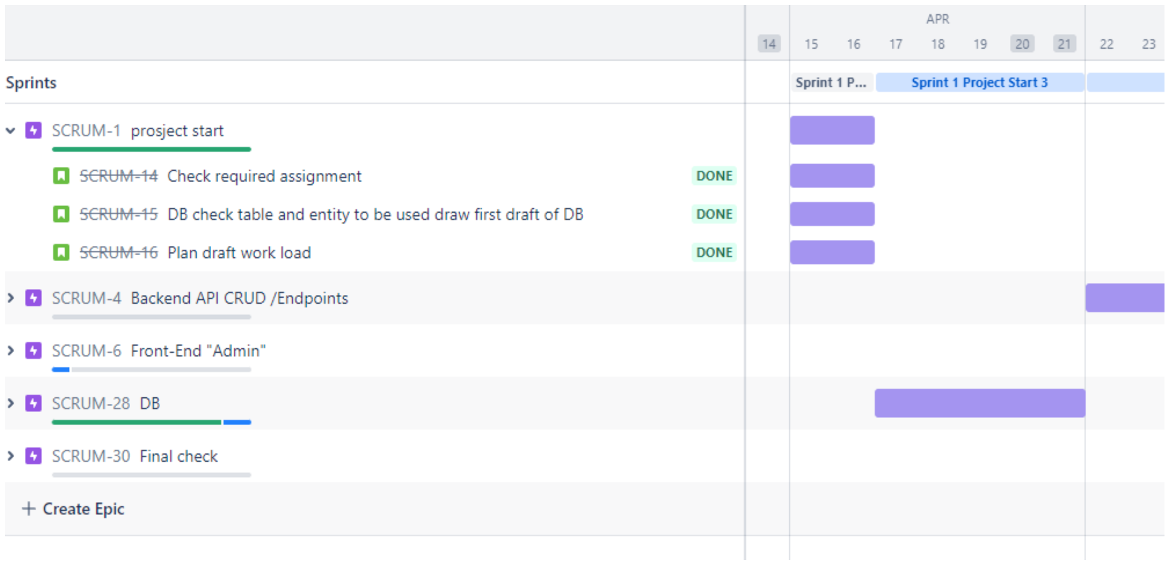
Project Reflection Report

Project in Jira ”aug22pt-ep1-dissolution2” I started the project with 5 epics. First epic and Second will be one small sprint divided by 2 and 5 days, one week total. I used this first sprint to go over the project requirement and plan and test first database configuration, with initializing and authentication JWT token.

Sprint 1 Week 1 (Days 2) Epic Project Start


APR
14 15 16 17 18 19 20 21 22 23

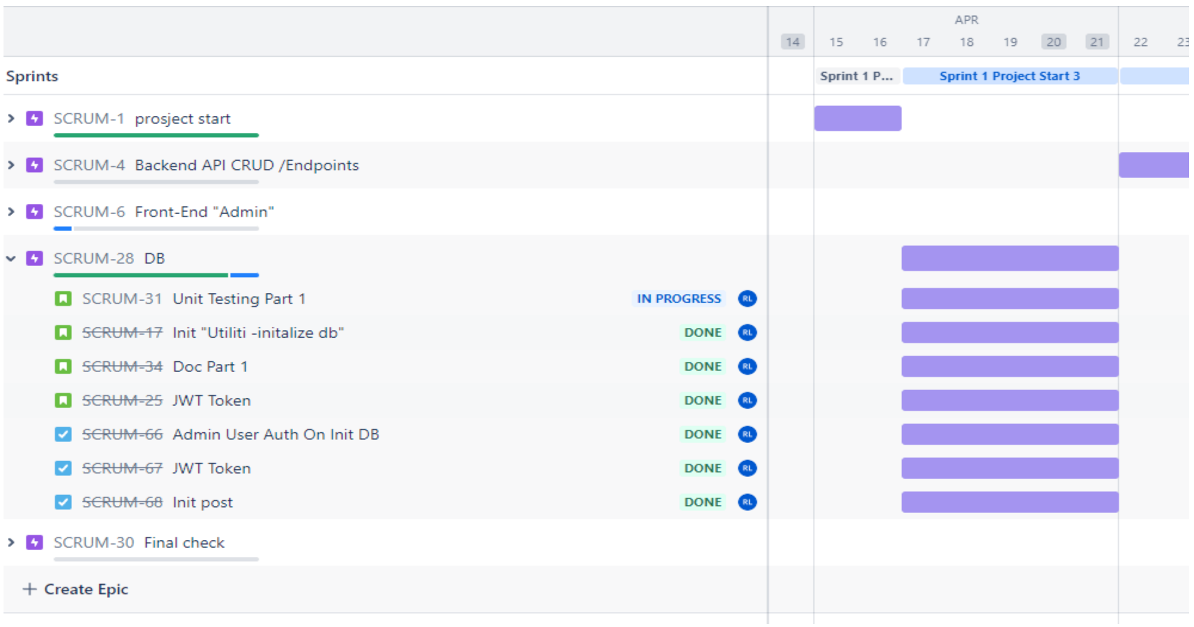
Sprints — Sprint 1 P... — Sprint 1 Project Start 3

- SCRUM-1 prosject start
  - ~~SCRUM-14~~ Check required assignment — DONE
  - ~~SCRUM-15~~ DB check table and entity to be used draw first draft of DB — DONE
  - ~~SCRUM-16~~ Plan draft work load — DONE
- SCRUM-4 Backend API CRUD /Endpoints
- SCRUM-6 Front-End "Admin"
- SCRUM-28 DB
- SCRUM-30 Final check

+ Create Epic

Epic «Project Start» has three story lines that i use to explaine how the project planning phase would take place. The storys was added to my whiteboard where i examined and drafteted best course for the prosjct starting point.

Sprint 2 Week 1 (Days 5) Epic DB


APR
14 15 16 17 18 19 20 21 22 23

Sprints — Sprint 1 P... — Sprint 1 Project Start 3

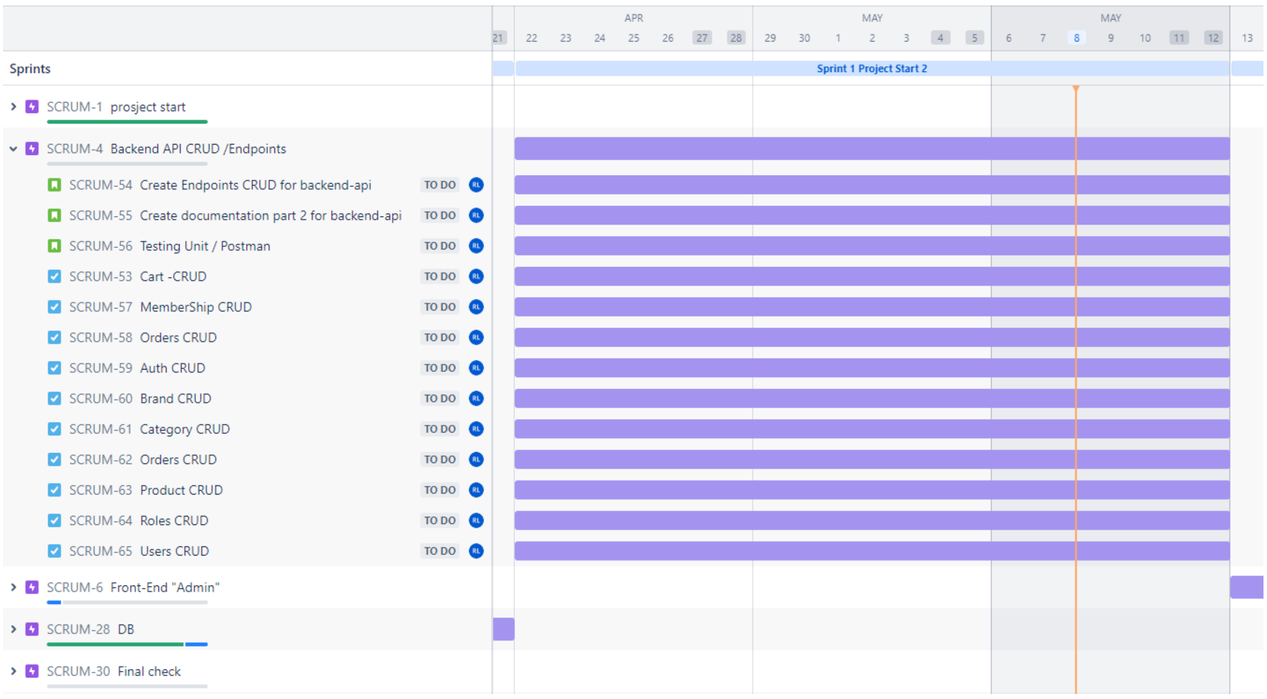
- SCRUM-1 prosject start
- SCRUM-4 Backend API CRUD /Endpoints
- SCRUM-6 Front-End "Admin"
- SCRUM-28 DB
  - SCRUM-31 Unit Testing Part 1 — IN PROGRESS — RL
  - ~~SCRUM-17~~ Init "Utiliti -initalize db" — DONE — RL
  - ~~SCRUM-34~~ Doc Part 1 — DONE — RL
  - ~~SCRUM-25~~ JWT Token — DONE — RL
  - ~~SCRUM-66~~ Admin User Auth On Init DB — DONE — RL
  - ~~SCRUM-67~~ JWT Token — DONE — RL
  - ~~SCRUM-68~~ Init post — DONE — RL
- SCRUM-30 Final check

+ Create Epic

Epic «DB» has foure story lines and three task at the start of the sprint. The sprint whent over the time line that was sett for the sprint as it was added task: DB Models (basic), DB Service files (basic), JWT Auth register / login (basic).

Both Sprint 1 and 2 was to serve an overall goal of planning the inital requirement of the project and building a basic framework.

Sprint 3 Weeks 3 Epic Crud / Endpoints (server api)



Sprints

SCRUM-1 prosject start

SCRUM-4 Backend API CRUD /Endpoints

SCRUM-54 Create Endpoints CRUD for backend-api TO DO RL
SCRUM-55 Create documentation part 2 for backend-api TO DO RL
SCRUM-56 Testing Unit / Postman TO DO RL
SCRUM-53 Cart -CRUD TO DO RL
SCRUM-57 MemberShip CRUD TO DO RL
SCRUM-58 Orders CRUD TO DO RL
SCRUM-59 Auth CRUD TO DO RL
SCRUM-60 Brand CRUD TO DO RL
SCRUM-61 Category CRUD TO DO RL
SCRUM-62 Orders CRUD TO DO RL
SCRUM-63 Product CRUD TO DO RL
SCRUM-64 Roles CRUD TO DO RL
SCRUM-65 Users CRUD TO DO RL

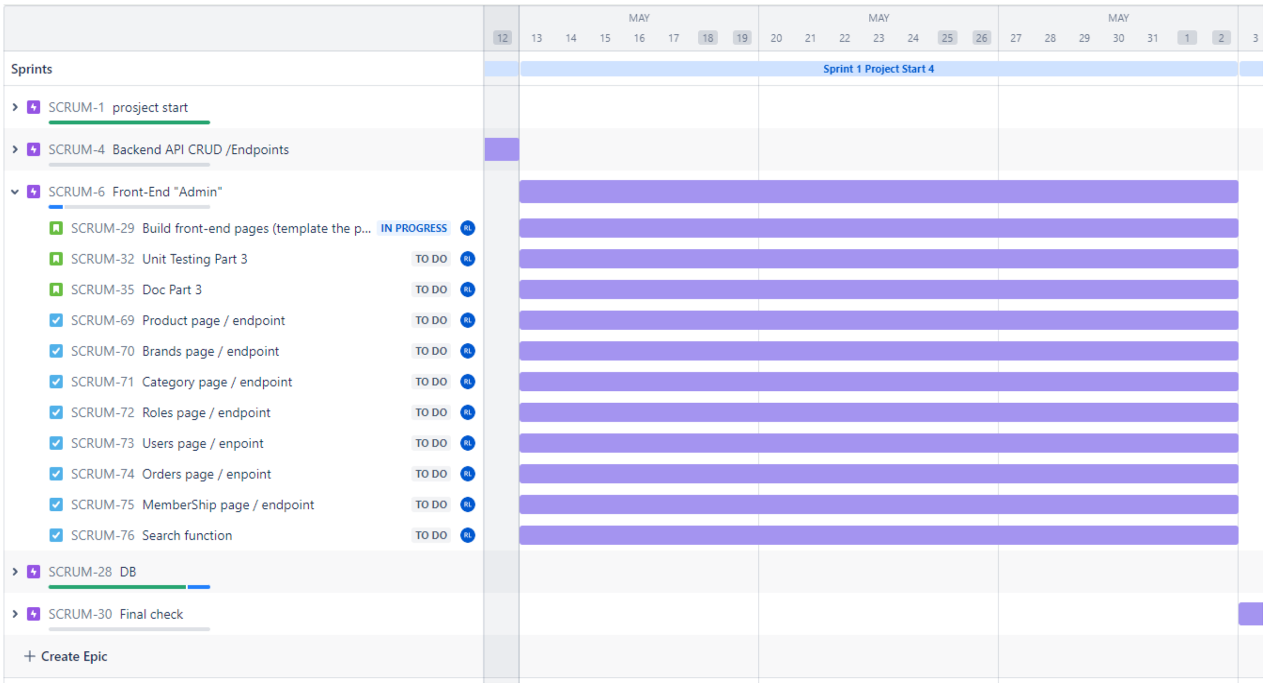
SCRUM-6 Front-End "Admin"

SCRUM-28 DB

SCRUM-30 Final check

Epic «Endpoints Server API» has three story lines and then issues.

Sprint 4 Weeks 3 Front-End “Admin”



Sprints

SCRUM-1 prosject start

SCRUM-4 Backend API CRUD /Endpoints

SCRUM-6 Front-End "Admin"

SCRUM-29 Build front-end pages (template the p... IN PROGRESS RL
SCRUM-32 Unit Testing Part 3 TO DO RL
SCRUM-35 Doc Part 3 TO DO RL
SCRUM-69 Product page / endpoint TO DO RL
SCRUM-70 Brands page / endpoint TO DO RL
SCRUM-71 Category page / endpoint TO DO RL
SCRUM-72 Roles page / endpoint TO DO RL
SCRUM-73 Users page / enpoint TO DO RL
SCRUM-74 Orders page / enpoint TO DO RL
SCRUM-75 MemberShip page / endpoint TO DO RL
SCRUM-76 Search function TO DO RL

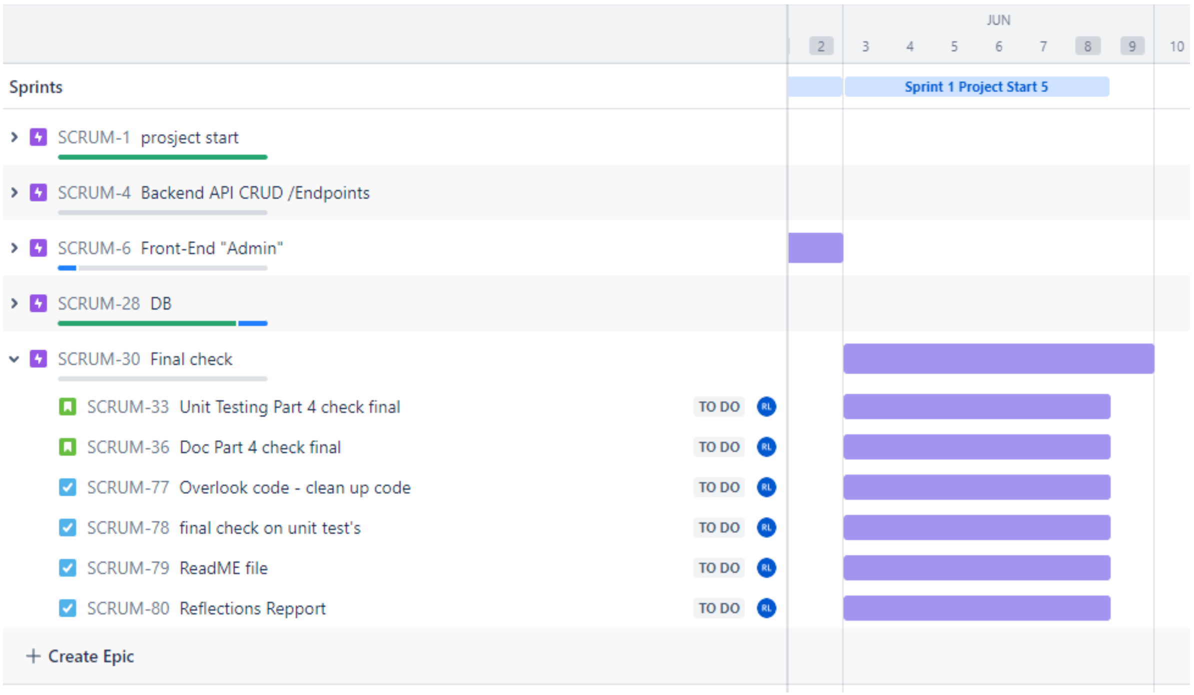
SCRUM-28 DB

SCRUM-30 Final check

+ Create Epic

Epic «Front-End Admin» has three story lines and eight issues.

Sprint 4 Weeks 1 Final Checks


JUN

2 3 4 5 6 7 8 9 10

Sprints — Sprint 1 Project Start 5

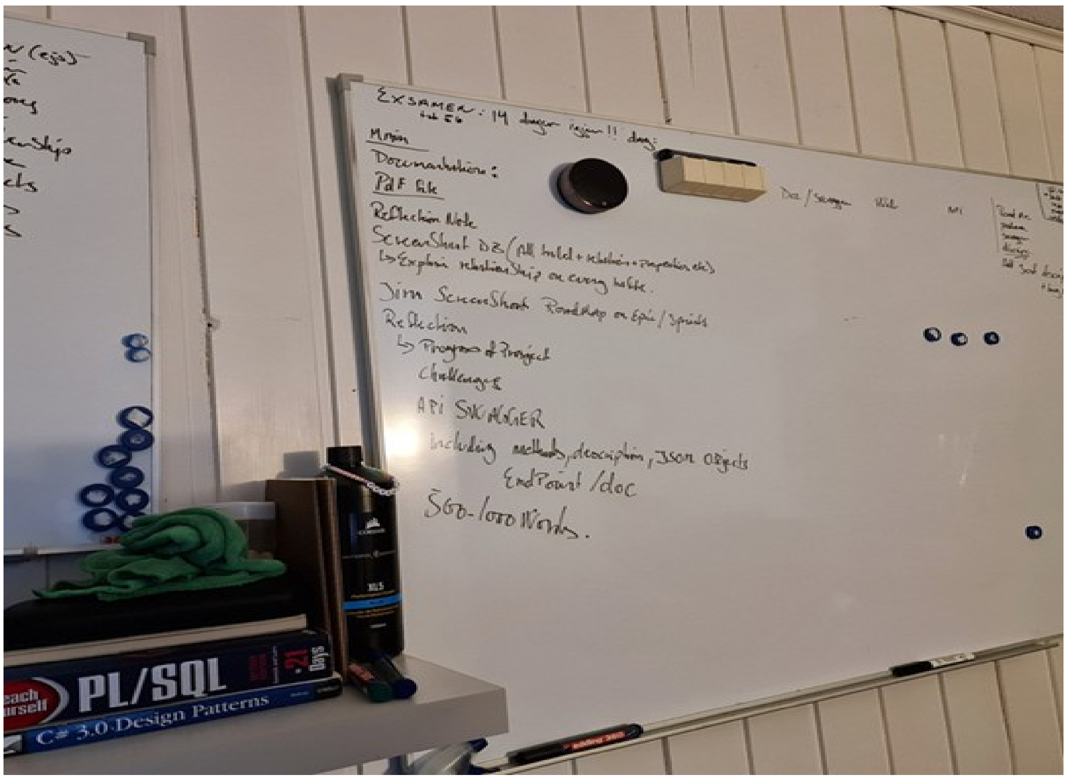
- SCRUM-1 prosjekt start
- SCRUM-4 Backend API CRUD /Endpoints
- SCRUM-6 Front-End "Admin"
- SCRUM-28 DB
- SCRUM-30 Final check
  - SCRUM-33 Unit Testing Part 4 check final — TO DO — RL
  - SCRUM-36 Doc Part 4 check final — TO DO — RL
  - SCRUM-77 Overlook code - clean up code — TO DO — RL
  - SCRUM-78 final check on unit test's — TO DO — RL
  - SCRUM-79 ReadME file — TO DO — RL
  - SCRUM-80 Reflections Repport — TO DO — RL

+ Create Epic

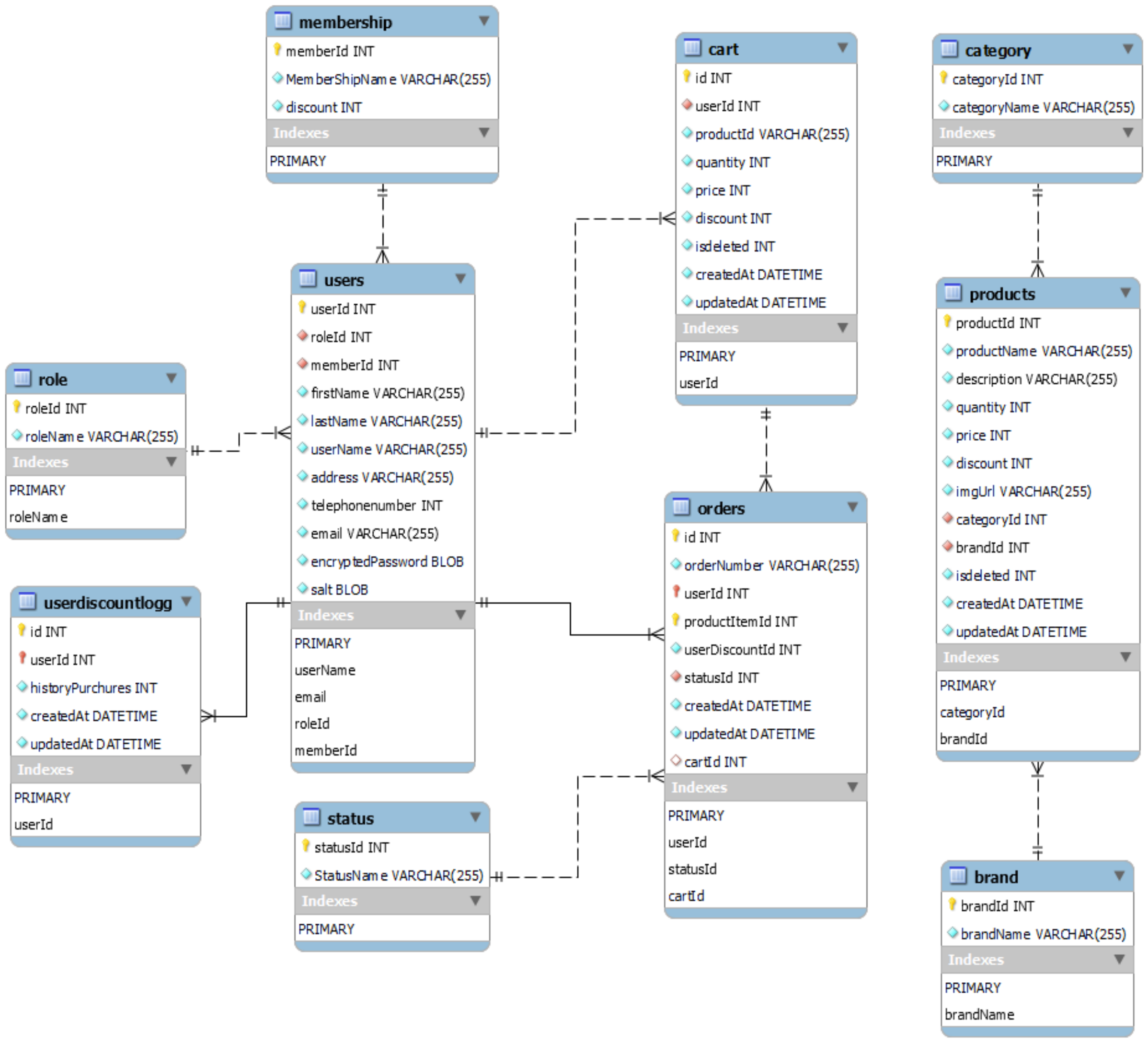
Epic «Final Checks» has two story lines and four issues.

Scrum planning with whiteboard

Under project duration I used my whiteboard to update an plan issues that presented itself witch help to organize and keep track of all the task on the to-do list, and those already completed.



Database



User Model:

- A user has a one-to-one relationship with Role, meaning a user can have only one role. This is represented in Sequelize by the association: Users.belongsTo(models.Role, { foreignKey: 'roleId' }).

Order Model:

- An order belongs to a Cart, meaning a many-to-one relationship. This indicates that multiple orders can belong to the same cart, but each order is associated with only one cart. In Sequelize, this is established through Orders.belongsTo(models.Cart, { foreignKey: 'cartId' }).

- An order also belongs to a User, meaning a many-to-one relationship. Multiple orders can be placed by the same user, but each order is associated with only one user. This relationship is represented by Orders.belongsTo(models.Users, { foreignKey: 'userId' }).

- Additionally, an order belongs to a Status, meaning a many-to-one relationship. Each order has a status associated with it, but each status can be associated with multiple orders. This is achieved in Sequelize using Orders.belongsTo(models.Status, { foreignKey: 'statusId' }).

Cart Model:

- A cart belongs to a User, meaning a many-to-one relationship. This means that multiple carts can be associated with the same user, but each cart is associated with only one user. This relationship is established by Cart.belongsTo(models.Users, { foreignKey: 'userId' }).

- Furthermore, a cart can have many orders, indicating a one-to-many relationship. This implies that each cart can contain multiple orders, but each order is associated with only one cart. In Sequelize, this is represented by Cart.hasMany(models.Orders, { foreignKey: 'cartId' }).

Products Model:

- A product belongs to a Category, representing a many-to-one relationship. This means that each product is associated with one category, but each category can have multiple associated products. In Sequelize, this relationship is defined by Products.belongsTo(models.Category, { foreignKey: 'categoryId' }).

- Similarly, a product belongs to a Brand, also a many-to-one relationship. Each product is associated with one brand, but each brand can have multiple associated products. This relationship is established through Products.belongsTo(models.Brand, { foreignKey: 'brandId' }).

Progression tracking of issues and solutions to challengers

Back-end:

- The initial problem to solve was the database, and over time, it underwent changes as the project requirements where interpreted differently and the solution's perspective changed. I utilized various sources to decide how to structure the database, which are referenced in URLs, with Associations being the most notable (How I decided to structure the response on Orders). SQL was initially tested in MySQL Workbench to achieve the desired view. [GROUP_CONCATE(DISTINCT...)]

- * All tables that are crated must have CRUD endpoints * this was hard to interpreted as, example order status is done on database initializing. And Orders POST / DELETE (CRUD), I took some leeway made CRUD endpoint with info about not implemented in the endpoint but with, example: Orders POST used a version of '/cart/checkout/' endpoint etc.
- role example - DELETE / PUT. We would break the application, and no other role should be added.
- users example - POST, this is implemented in route '/Auth/Registrate' endpoint.

Front-end:

- I used the assignment video as a reference model for the front en's appearance. Beginning with basic HTML, CSS, and JQuery. After some basic research on elements like pop-ups and the window object, I concluded that using Modals was the best approach.

- Implementing Modals with forms posed some challenges. I began by creating individual Modals for update, add, and delete functions on the products page. However for smaller pages like Category etc I decided to combine these functions and use scripting with the Modals to disable or activate form submit buttons.

CSS – Scripting:

- I started in the beginning to use “<%- include('./partials/navbar.ejs') %>” to include style and scripting. But found it as the scripting got a bit more complicated that I wanted to have them in plain view to better organize my code.

Naming convention:

- During the course of the project and especially during code clean-up Sprint 4 phase, I realized that I should have adopted a more read friendly (shorter words, capital letter, underscore etc) naming convention. This serves as a note for future projects.

JIRA PLANNING:

- When dealing with issues, updates and planning where being done on the whiteboard, causing the Jira project to fall behind on sprint 2 and not reflect the new tasks that was found to be needed. (Epic 2)

- Initially, Jira was the main use for project planning, in combination with my whiteboard, but as the project progressed the whiteboard took over.

- For the remaining epics, tasks were clearer, resulting in minimal deviation from Jira planning to whiteboard.