Reflection Report – Fullstack Exam Project

Project Progression

This fullstack e-commerce project was built with a focus on proper backend structure, RESTful APIs, secure authentication, admin functionalities, and a functional admin dashboard frontend. The development process was planned and tracked in Jira, using Epics and Sprints to break the work into manageable tasks.

The backend was built first, using Express.js, Sequelize ORM, and MySQL. I began by defining the database schema, including models for users, roles, memberships, products, categories, brands, carts, orders, and related relationships. After the models were finalized, I implemented authentication using JWT and route protection middleware.

Once the base structure was in place, I developed full CRUD APIs for all entities, starting with brands and categories, then products and users. These endpoints were protected using role-based access control. I used Postman and Jest for testing routes and data handling logic.

Later in the process, I created Swagger documentation for all endpoints, which is available at /doc. It includes summary descriptions, parameters, JSON bodies, and expected responses.

The frontend was developed using EJS and Bootstrap. It focuses on admin operations such as managing products, users, roles, and viewing orders. The dashboard connects to the backend API using Axios and passes the JWT token for authorization.

Finally, I added a /init route that seeds the database with roles, an admin user, memberships, and a product set from an external API. This simplified initial testing.

Challenges Faced

There were several technical and logical challenges during the development process:

1. Sequelize associations and database integrity:
Defining correct one-to-many and many-to-one relationships between entities such as Users → Roles, Products → Categories, and Orders → OrderItems was more complex than expected. Ensuring that foreign keys were populated correctly, especially for role_id and membership_id, required careful seeding and validation logic.

2. Password handling and security:
Initially, passwords were not hashed when users registered via the API. This was fixed by adding a Sequelize hook (beforeCreate) and ensuring consistent hashing during user creation. Login issues were also resolved by comparing hashed passwords with bcrypt.compare().

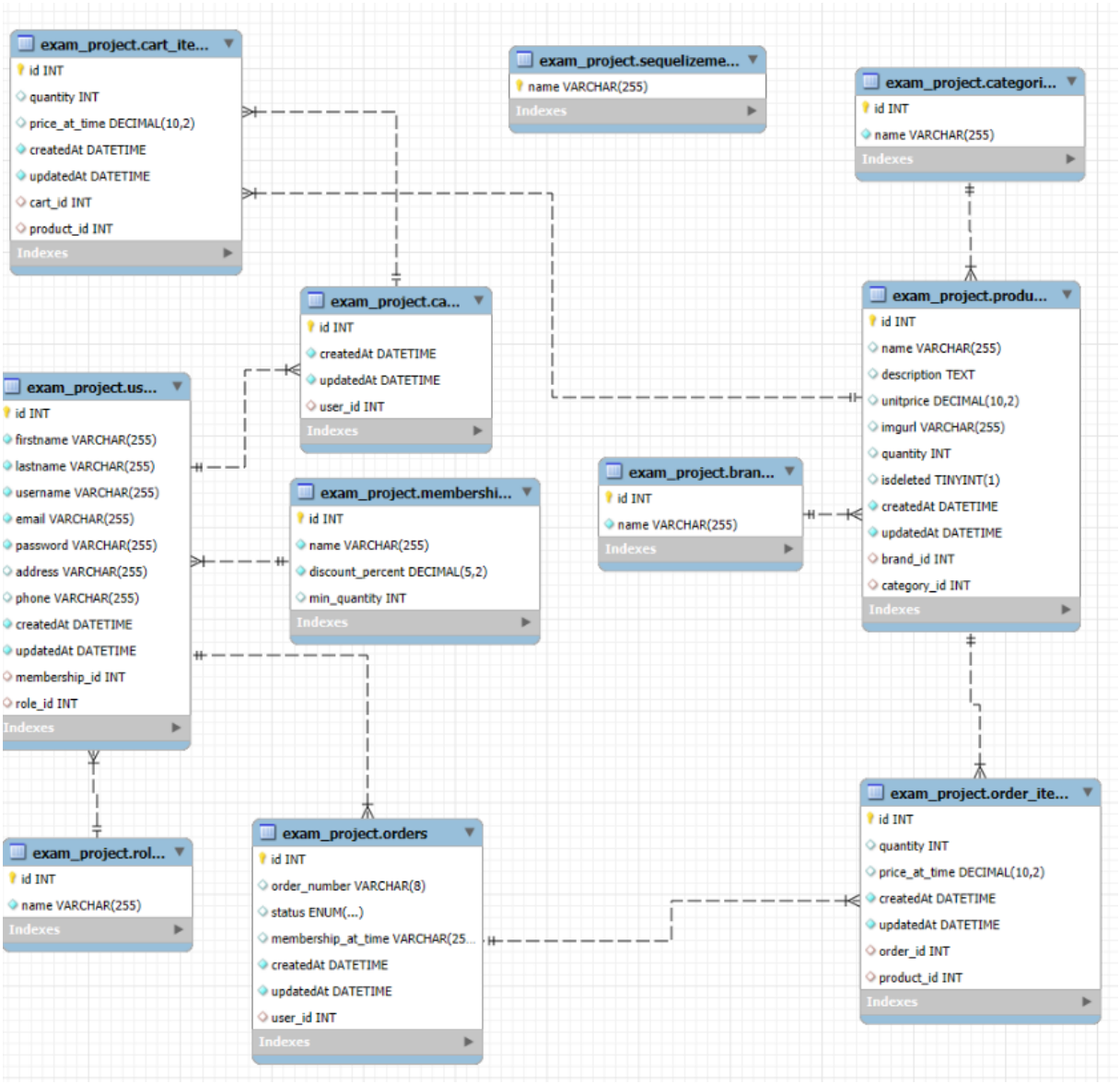
3. Authorization:
JWT token handling in the frontend and backend had to be carefully managed. Tokens were sometimes missing or expired, causing access issues. I added proper middleware and token validation for both general users and admin-only routes.

4. Swagger and OpenAPI:
Documenting the API using Swagger took significant effort. I had to ensure all routes were properly annotated with summaries, schemas, and security definitions. Swagger's strict YAML syntax made this process detail-oriented but rewarding in the end.

5. Frontend data fetching and rendering:
Handling errors and missing fields in Axios calls was tricky, especially when the backend API returned 404 or 500 errors. I added fallback messages and improved error logging to the console for better debugging.

6. Time management:
Staying on track with Jira was essential to avoid being overwhelmed. Breaking down the project into Epics and working on one major feature at a time helped me manage progress and avoid scope creep.

Database ERD



I used MySQL Workbench to design and visualize the ER diagram. It includes:

Users table with foreign keys to Roles and Memberships

Products related to Categories and Brands

Orders linked to Users and OrderItems

Carts associated with Users and CartItems

Relationships Between Tables
A User belongs to a Role and a Membership (many users can share the same role/membership)

A Product belongs to one Category and one Brand

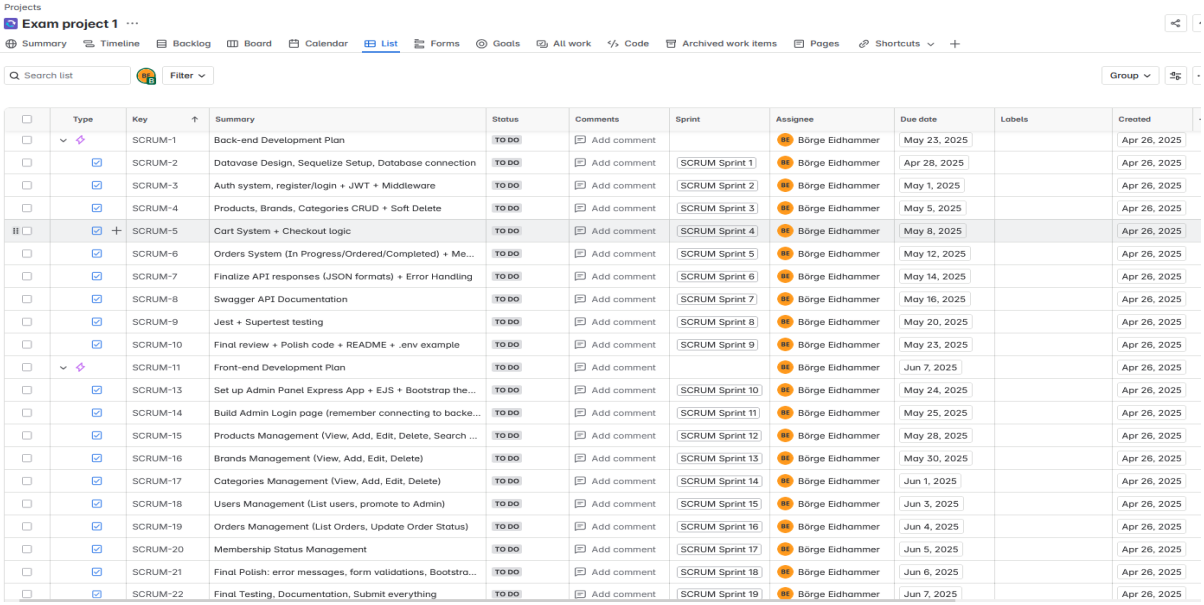
A User can have multiple Orders and Carts

An Order has many OrderItems

A Cart has many CartItems

These relationships follow a one-to-many or many-to-one pattern using Sequelize's association methods.

Jira Roadmap



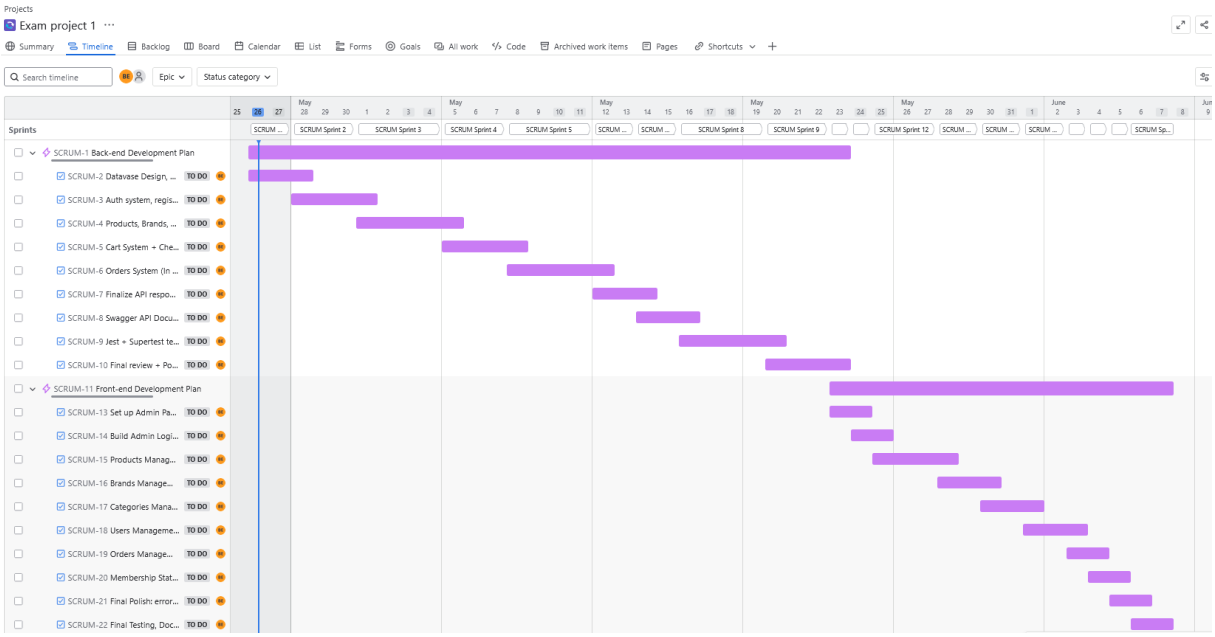
Projects

Exam project 1 ···

Summary | Timeline | Backlog | Board | Calendar | List | Forms | Goals | All work | Code | Archived work items | Pages | Shortcuts

Search list | Filter | Group

Type	Key	Summary	Status	Comments	Sprint	Assignee	Due date	Labels	Created
	SCRUM-1	Back-end Development Plan	TO DO	Add comment		Börge Eidhammer	May 23, 2025		Apr 26, 2025
	SCRUM-2	Datavase Design, Sequelize Setup, Database connection	TO DO	Add comment	SCRUM Sprint 1	Börge Eidhammer	Apr 28, 2025		Apr 26, 2025
	SCRUM-3	Auth system, register/login + JWT + Middleware	TO DO	Add comment	SCRUM Sprint 2	Börge Eidhammer	May 1, 2025		Apr 26, 2025
	SCRUM-4	Products, Brands, Categories CRUD + Soft Delete	TO DO	Add comment	SCRUM Sprint 3	Börge Eidhammer	May 5, 2025		Apr 26, 2025
	SCRUM-5	Cart System + Checkout logic	TO DO	Add comment	SCRUM Sprint 4	Börge Eidhammer	May 8, 2025		Apr 26, 2025
	SCRUM-6	Orders System (In Progress/Ordered/Completed) + Me...	TO DO	Add comment	SCRUM Sprint 5	Börge Eidhammer	May 12, 2025		Apr 26, 2025
	SCRUM-7	Finalize API responses (JSON formats) + Error Handling	TO DO	Add comment	SCRUM Sprint 6	Börge Eidhammer	May 14, 2025		Apr 26, 2025
	SCRUM-8	Swagger API Documentation	TO DO	Add comment	SCRUM Sprint 7	Börge Eidhammer	May 16, 2025		Apr 26, 2025
	SCRUM-9	Jest + Supertest testing	TO DO	Add comment	SCRUM Sprint 8	Börge Eidhammer	May 20, 2025		Apr 26, 2025
	SCRUM-10	Final review + Polish code + README + .env example	TO DO	Add comment	SCRUM Sprint 9	Börge Eidhammer	May 23, 2025		Apr 26, 2025
	SCRUM-11	Front-end Development Plan	TO DO	Add comment		Börge Eidhammer	Jun 7, 2025		Apr 26, 2025
	SCRUM-13	Set up Admin Panel Express App + EJS + Bootstrap the...	TO DO	Add comment	SCRUM Sprint 10	Börge Eidhammer	May 24, 2025		Apr 26, 2025
	SCRUM-14	Build Admin Login page (remember connecting to backe...	TO DO	Add comment	SCRUM Sprint 11	Börge Eidhammer	May 25, 2025		Apr 26, 2025
	SCRUM-15	Products Management (View, Add, Edit, Delete, Search ...	TO DO	Add comment	SCRUM Sprint 12	Börge Eidhammer	May 28, 2025		Apr 26, 2025
	SCRUM-16	Brands Management (View, Add, Edit, Delete)	TO DO	Add comment	SCRUM Sprint 13	Börge Eidhammer	May 30, 2025		Apr 26, 2025
	SCRUM-17	Categories Management (View, Add, Edit, Delete)	TO DO	Add comment	SCRUM Sprint 14	Börge Eidhammer	Jun 1, 2025		Apr 26, 2025
	SCRUM-18	Users Management (List users, promote to Admin)	TO DO	Add comment	SCRUM Sprint 15	Börge Eidhammer	Jun 3, 2025		Apr 26, 2025
	SCRUM-19	Orders Management (List Orders, Update Order Status)	TO DO	Add comment	SCRUM Sprint 16	Börge Eidhammer	Jun 4, 2025		Apr 26, 2025
	SCRUM-20	Membership Status Management	TO DO	Add comment	SCRUM Sprint 17	Börge Eidhammer	Jun 5, 2025		Apr 26, 2025
	SCRUM-21	Final Polish: error messages, form validations, Bootstra...	TO DO	Add comment	SCRUM Sprint 18	Börge Eidhammer	Jun 6, 2025		Apr 26, 2025
	SCRUM-22	Final Testing, Documentation, Submit everything	TO DO	Add comment	SCRUM Sprint 19	Börge Eidhammer	Jun 7, 2025		Apr 26, 2025



I used Jira to structure my development into:

Epic: Backend setup and API

Epic: Frontend admin dashboard

Each Epic was broken into sprints and individual tickets to track tasks, bugs, and enhancements.

Started with Jira making the development plan. It is easy seeing what needs to be done. What is difficult is to put the time you think you will use on the different tasks. So the time on every sprint is just guessing. I used to much time making a list of tasks. So i relaised that i would generalise the main sprint tasks and have the back-end and front-end part as Epics.

I think if i was in management and had this as my job i would specify more

.env File Example

DB_HOST=localhost
DB_USER=root
DB_PASSWORD=my_password
DB_NAME=exam_project
JWT_SECRET=my_super_secret_key
Note: The actual .env file is excluded from Git using .gitignore.

Swagger API Documentation
The full API documentation is available and accessible via:
http://localhost:3000/doc

It includes all routes for:

Products

Users

Orders

Categories

Brands

Authentication

Roles and Memberships

Each route is described with request parameters, security (JWT), and response schemas.

References
This list is also included in the README.md.

Express.js documentation: https://expressjs.com/

Sequelize ORM documentation: https://sequelize.org/

JWT info: https://jwt.io/

Swagger/OpenAPI reference: https://swagger.io/specification/

Bcrypt hashing: https://www.npmjs.com/package/bcryptjs

Axios HTTP client: https://axios-http.com/

EJS template engine: https://ejs.co/

ChatGPT: Used to debug, validate logic and planning