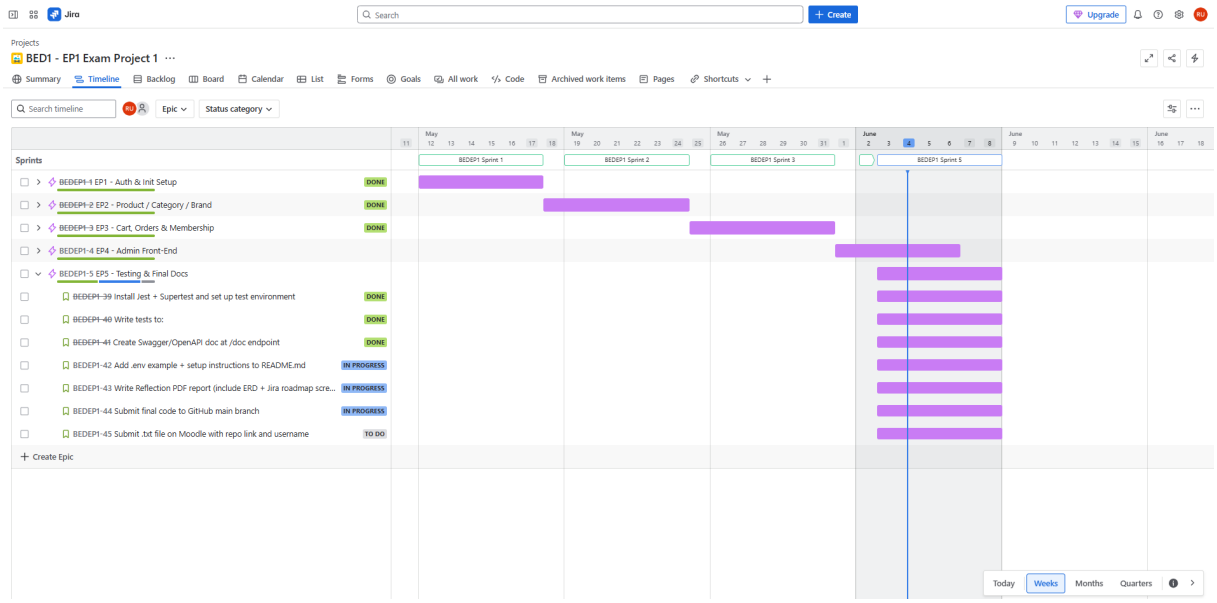
BED1 EP1 CA – Exam Project 1

⚡ PulseMarket – Fullstack E-Commerce Platform

Noroff School of Technology and Digital Media
Back-End Development 1 – June 2025



Timeline (Sprints with Epics) – Screenshot from Jira.

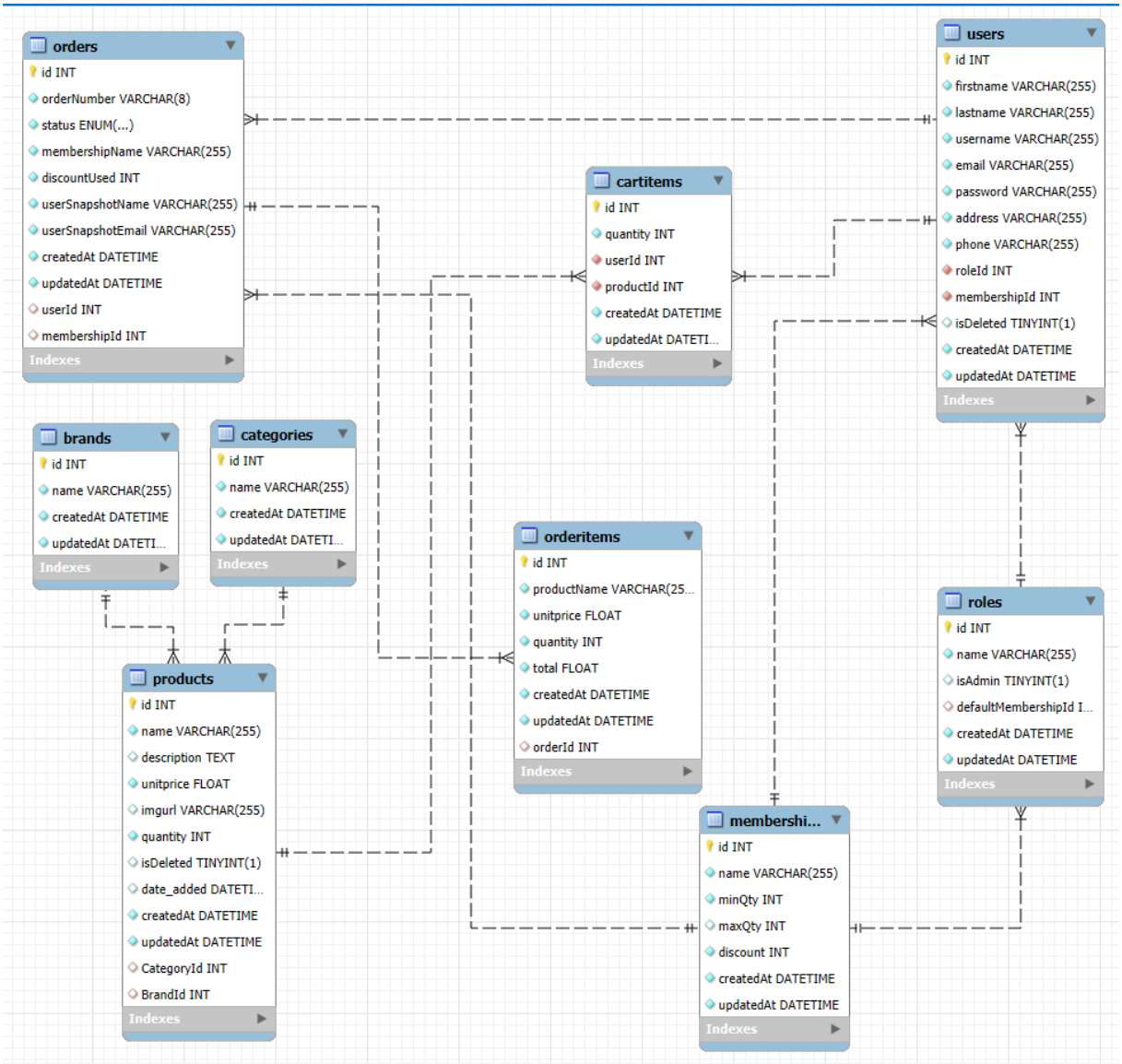
🗺️ Jira Timeline Summary

This timeline outlines the full development process across five distinct sprints. Each sprint targeted a key focus area: authentication and initialization, product and content setup, cart and checkout functionality, admin front-end interface, and final testing and documentation.

All epics were broken into manageable tasks and tracked in Jira to ensure clarity and consistency across the project lifecycle. With most of the features implemented ahead of Sprint 5, the final days were spent polishing features, squashing bugs, and preparing documentation and deliverables.

🌊 .env example

```
PORT=3000
DB_HOST=localhost
DB_PORT=3306
DB_USER=Admin
DB_PASS=0000
DB_NAME=EcommerceDB
TOKEN_SECRET=supersecretkey
```



ERD (Entity-Relationship Diagram) – exported from MySQL Workbench.

Entity Relationship Explanation (ERD Breakdown)

`users`

- **One `user` has one `role`** (`roleId`) → Many-to-One
- **One `user` has one `membership`** (`membershipId`) → Many-to-One
- **One `user` can have many `cartitems`** → One-to-Many
- **One `user` can have many `orders`** → One-to-Many

🧺 `cartitems`

- **Each `cartitem` belongs to one `user`** → Many-to-One
- **Each `cartitem` belongs to one `product`** (`productId`) → Many-to-One

📦 `products`

- **One `product` belongs to one `category`** (`CategoryId`) → Many-to-One
- **One `product` belongs to one `brand`** (`BrandId`) → Many-to-One
- **One `product` can appear in many `cartitems`** → One-to-Many
- **One `product` can appear in many `orderitems`** (via snapshot fields) → One-to-Many (captured via `orderitems.productName` and `unitprice`)

🛒 `orders`

- **One `order` belongs to one `user`** (`userId`) → Many-to-One
- **One `order` has one membership at time of checkout** (`membershipId`) → Many-to-One
- **One `order` can have many `orderitems`** → One-to-Many
- Contains snapshot fields:
  - `userSnapshotName`, `userSnapshotEmail` (capturing user identity at checkout)
  - `membershipName`, `discountUsed` (capturing discount context)

📄 `orderitems`

- **Each `orderitem` belongs to one `order`** (`orderId`) → Many-to-One
- Captures snapshot data from `products`:
  - `productName`, `unitprice`, `quantity`, `total`

🏷️ `categories`

- **One `category` can have many `products`** → One-to-Many

🏷️ `brands`

- **One `brand` can have many `products`** → One-to-Many

`roles`

- **One `role` can be assigned to many `users`** → One-to-Many
- `isAdmin` flag defines admin rights
- `defaultMembershipId` specifies default membership for this role

`memberships`

- **One `membership` can be assigned to many `users`** → One-to-Many
- **One `membership` can be linked to many `orders`** (as snapshot) → One-to-Many
- Defines discount thresholds using:
  - `minQty`, `maxQty`, and `discount`

Summary of Relationship Types

Relationship Type	Tables Involved
One-to-Many	Users → Orders, Users → CartItems
One-to-Many	Products → CartItems, Products → OrderItems (snapshot)
One-to-Many	Categories → Products
One-to-Many	Brands → Products
One-to-Many	Roles → Users
One-to-Many	Memberships → Users, Memberships → Orders
One-to-Many	Orders → OrderItems
Many-to-One	CartItems → Products, CartItems → Users
Many-to-One	Orders → Users, Orders → Memberships
Many-to-One	Products → Categories, Products → Brands
Many-to-One	OrderItems → Orders

📈 Progression Reflection

The development of PulseMarket followed a structured five-sprint timeline, each focused on distinct functional areas. I began by setting up the Express/Sequelize back-end and connecting it to MySQL. Then I created models, relationships, and core routes for roles, memberships, users, and authentication, securing everything with JWT.

With core auth in place, I implemented CRUD features for products, brands, and categories. I added soft-delete logic for products and active/inactive flags for content. All functionality was tested manually and with Jest/Supertest.

Mid-project, I focused on the cart, checkout, and order flow — the most complex area of the app. This included logic for discounts based on membership, stock validation, order snapshots, and duplicate prevention. Handling edge cases and enforcing rules like "no duplicate cart entries" required careful logic and multiple refinements.

In Sprint 4, I developed a responsive admin front-end using Express, EJS, and Bootstrap 5. Thanks to my prior two years in Noroff's Front-End Development program, I was able to structure clean layouts, apply DRY principles with partials, and use modals effectively. This background also helped with UX decisions, accessibility, and managing Bootstrap styling across views. My front-end knowledge saved time and let me focus more deeply on backend complexity.

In the final sprint, I polished the UI, improved UX details, finalized Swagger documentation, cleaned the codebase, and wrote this report. I also prepared detailed README files, captured screenshots, and reviewed the project against exam requirements.

⚠️ Challenges Faced

One major challenge was designing a normalized database schema that still allowed snapshot logic without violating 3NF. Storing order-time data like unit price and discount involved using duplicated (but justified) fields to preserve historical accuracy.

The cart/checkout system introduced tricky scenarios like quantity limits, out-of-stock logic, and merging cart items. I revised this logic several times to ensure consistency across all flows.

Managing user roles and memberships added another layer of complexity. Roles defined both admin rights and default memberships, so I had to enforce those rules during registration and updates. Ensuring that seeded users like "Staff" and "Support" behaved correctly in all contexts required careful handling.

Creating a DRY and flexible layout with EJS and Bootstrap also took iteration. Implementing live feedback for cart limits, JWT token awareness, and role-based UI changes challenged both front-end and back-end sync logic. Balancing design clarity with dynamic logic became a key learning point.

Lastly, writing documentation while developing was demanding. I found that regularly updating the README and Swagger docs alongside implementation helped keep the project clean and aligned.

Finally, maintaining documentation while actively developing code was difficult at times. I found it helpful to pause periodically to update README content, test routes in Postman, and verify Swagger endpoint examples. In the end, I believe this helped ensure a more complete and maintainable submission.

📌 Final Reflections

This has been one of the most demanding and rewarding projects in my backend journey. It required full control over architecture, data integrity, user roles, and visual presentation.

I'm especially proud of how modular and scalable the final result is. With services, controllers, routes, and validators separated, the code is maintainable and ready for future features like payments or a public-facing store.

My front-end education at Noroff played a major role in delivering a polished admin interface. Being comfortable with Bootstrap, EJS, and layout logic helped me build clean, user-friendly views quickly, and let me focus my learning on backend challenges like token handling, normalization, and test automation.

Beyond the coding, this project gave me valuable experience in planning, documenting, testing, and structuring a real-world application. Tools like Swagger, Jira, Sequelize, and testing libraries became part of my everyday workflow.

This wasn't just a technical test — it also shaped my approach to building complete, scalable, and documented applications. I feel better prepared to contribute as a fullstack developer going forward.

Submitted by:
Rune Unhjem
BED1 – Back-End Development – 1st year (following 2 years of Front-End Development)
Noroff – School of Technology and Digital Media
June 2025