Capstone Project, Part 2: Design Report

Leslie Montgomery

McMaster University

BDV 102: Interactivity and Databases

Per Pettersson

November 2025

Introduction

GitHub Repository: https://github.com/montygrl/swiftcart-backend-part2

This project builds on Part 1 by turning the ERD and SQL schema into a working backend API using JavaScript, Express, and Sequelize. The backend supports product listings, shopping-cart actions, and checkout. The main challenges were (1) implementing core functionality in Express and (2) replacing raw SQL with Sequelize ORM. This report summarizes design decisions, endpoints, and key issues resolved.

2. Design Choices

2.1 Technology Stack

I used the following tools:

- **Node.js + Express** for routing
- **PostgreSQL (Neon.tech)** for the database
- **Sequelize ORM** for table interaction
- **dotenv** for configuration
- **VS Code REST Client** for testing

The stack was chosen to keep the backend simple, stable, and easy to test.

2.2 Sequelize Models

My schema used explicit keys (product_id, order_id, etc.), while Sequelize defaults to id. To avoid errors, I rewrote models so field names matched the database exactly and disabled timestamps. This alignment eliminated "*column 'id' does not exist*" errors.

2.3 API Architecture

```
src/
  — models/
  — routes/
        — cart.js
        — checkout.js
        — server.js
```

- **server.js** - sets up Express and routes
- **cart.js** - handles add/list/clear cart actions
- **checkout.js** - creates orders and order_items in a transaction
- **models/** - defines Sequelize models and associations

This kept the code modular and readable.

3. API Endpoints

- **GET /products** - returns all products.
- **POST /cart** - adds an item for user_id = 1.
- **GET /cart** - shows cart contents with product details and totals.
- **POST /checkout** - reads cart, calculates total, creates order and order_items, clears cart, all within a transaction. This was the most complex endpoint.

4. Challenges & Solutions

- REST Client spacing errors: fixed by placing JSON body directly after headers.
- Sequelize id mismatch: solved by disabling timestamps, setting tableName manually, and defining explicit primary keys.
- In the checkout route, after const order = await Order.create(...), Sequelize was returning the order object with .id = null but the real key was .order_id. So I had to use order.order_id when inserting into order_items.
- Testing: once formatting was fixed, all endpoints passed: products loaded, cart worked, checkout created orders and cleared the cart.

5. Conclusion

Part 2 was more technical than Part 1, especially as I introduced Sequelize late. The main lesson was how strict Sequelize is about naming conventions and primary keys. Despite troubleshooting, the final backend:

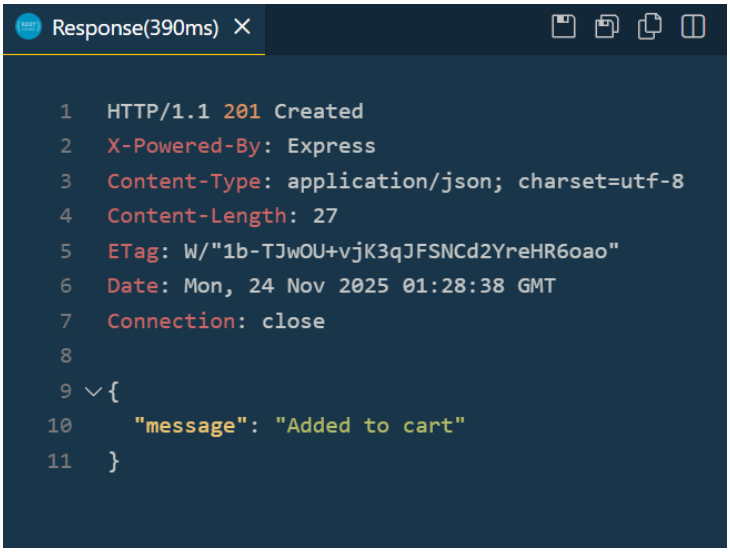
- meets assignment requirements
- connects to Neon
- uses Sequelize for CRUD operations
- passes all REST tests

This project gave me a clear view of backend flow: from database tables to API routes, ensuring everything integrates smoothly. If I were to do this again, I would design the database schema with Sequelize conventions in mind from the beginning (especially since I originally missed the instruction that Sequelize was required, and had to convert everything after finishing the raw-SQL version), because troubleshooting the ORM took more time than the actual logic.

6. Test Results

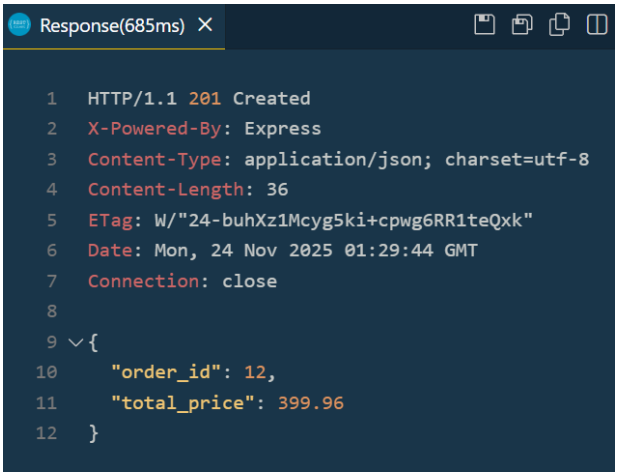
1. **GET /products** > returns all products (200 OK)

2. **POST /cart** > adds item to cart (200 OK)

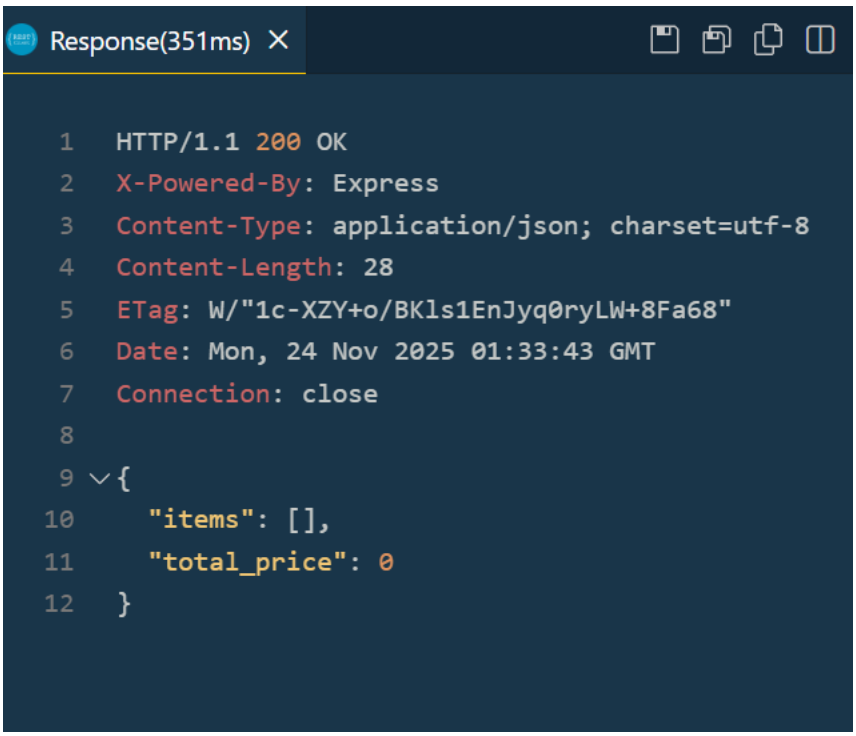


3. **GET /cart** > shows cart items with correct totals

4.**POST /checkout** > creates order and empties cart (201 Created)



5. **GET /cart (after checkout)** > cart is now empty



References

APIDog. (n.d.). *API design tool: OpenAPI documentation & testing*. https://apidog.com/

Dotenv. (n.d.). *Dotenv: Manage environment variables in Node.js*. npm. https://www.npmjs.com/package/dotenv

Express.js. (n.d.). *Express web framework for Node.js*. https://expressjs.com/

McMaster University. (2025). *BDV102: Backend development: Course modules 1–9*. Department of Continuing Education.

Neon. (n.d.). *Neon serverless PostgreSQL*. https://neon.tech/

PostgreSQL. (n.d.). *PostgreSQL official documentation*. https://www.postgresql.org/docs/

Swagger. (n.d.). *OpenAPI specification & tools*. https://swagger.io/tools/open-source/