API - Testing - CI/CD

Final Class Project (part 1)

Build an E-commerce API

Overview

This project requires students to integrate all the concepts they've learned into a single, functional application, moving data through the different layers (Express, Node, Prisma, PostgreSQL).

Create a RESTful API for a basic e-commerce application that allows users to manage products, place orders, and securely access data.

Important Notes;

- **DO NOT** use AI to build this project. Do it yourself.

Project Timelines.

Start date:
09th/Dec/2025

Submission deadline:
20th/Dec/2025 at 8:00 pm

Project Requirements;

These stories define the necessary functionality and user experience details.

API Endpoints Required:

Resource	Method	Endpoint	Description
User	POST	/auth/signup	Create a new user.
User	POST	/auth/login	Log in and receive a **JWT** token.
User	GET	/users/:id	Retrieve a user by their ID. (by CUSTOMER/ADMIN)
Products	GET	/products	Retrieve a list of all products.
Products	POST	/products	Create a new product. (Only ADMIN can do this)
Products	PATCH	/products/:id	Update an existing product. (Only ADMIN can do this)
Products	DELETE	/products/:id	Delete a product. (Only ADMIN can do this)
Orders	POST	/orders	Place a new order, deducting stock. (by CUSTOMER/ADMIN)
Orders	GET	/orders/:id	Retrieve an order by its ID (by CUSTOMER/ADMIN)

Tasks for you to complete:

1. Your data models should be well-defined in **schema.prisma** file so that the structure of the e-commerce data (User, Product, Order, and OrderItem) is clear and relationships are clearly stated. Implement a **Role** field on the **User** model (**CUSTOMER, ADMIN**).

2. Implement **sign-up** and **login** endpoints. The server should return an error if a user with that email already exists.
3. Upon successful login, generate and return a **JWT** for subsequent API calls.

4. Implement **Role-Based Access Control (RBAC)** middleware to ensure only authenticated **ADMIN** users can use **POST**, **PUT**, and **DELETE** on the

/products endpoint.

5. Use the **Joi** package to validate the input payload for all **POST** and **PUT** endpoints (e.g., strong password requirements, valid email format, positive price/stock values).

6. As an **admin**, create, update, and delete products. Ensure a regular user is blocked from these routes.

7. As an authenticated user, create a new order. The application must **verify and deduct sufficient stock** for each product before creating the order.

8. The application must immediately log the successful post creation/deletion to **ecommerce_activity.log**. This log should persist even if the server restarts, using the **Node.js file system module** (`fs`) for file persistence.

9. All data about users, products, and orders should be stored in a **PostgreSQL** database.

10. Implement a comprehensive test suite using **Jest** for unit testing business logic (e.g., price calculation, stock checks) and **Supertest** for integration testing API endpoints (e.g., successful order placement, 401/403 access control checks).

11. Configure Jest to enforce a minimum test coverage of 80% (Line/Statement Coverage). The CI/CD pipeline with GitHub Actions should fail if the minimum coverage is not met.

12. Create a **main.yml** workflow file in **.github/workflows/** that automatically runs on **push** and **pull_request** events.
    - Set up a temporary PostgreSQL service for testing.
    - Run Prisma migrations to set up the test database schema.
    - Execute all **Jest/Supertest** tests

13. Create a production-ready **Dockerfile** that containerizes the entire Node/Express application

14. Deploy the fully containerized application as a **Web Service** on **Render**.

Project Submission Link

Link to be provided