TDD Kata: Sweet Shop Management System

Objective

The goal of this kata is to design, build, and test a full-stack Sweet Shop Management System. This project will test your skills in API development, database management, frontend implementation, testing, and modern development workflows, including the use of AI tools.

Core Requirements

1. Backend API (RESTful)

You are to build a robust backend API that will serve as the brain of the application.

- **Technology** : Using **Node.js/TypeScript** (with Express/NestJS)
- **Database:** The application must connect to a database (MongoDB). An in-memory database is not sufficient.
- **User Authentication:**
  - Users must be able to register and log in.
  - Implement token-based authentication (e.g., JWT) to secure certain API endpoints.
- **API Endpoints:**
  - **Auth:** `POST /api/auth/register`, `POST /api/auth/login`
  - **Sweets (Protected):**
    - `POST /api/sweets`: Add a new sweet.
    - `GET /api/sweets`: View a list of all available sweets.
    - `GET /api/sweets/search`: Search for sweets by name, category, or price range.
    - `PUT /api/sweets/:id`: Update a sweet's details.
    - `DELETE /api/sweets/:id`: Delete a sweet (Admin only).
  - **Inventory (Protected):**
    - `POST /api/sweets/:id/purchase`: Purchase a sweet, decreasing its quantity.
    - `POST /api/sweets/:id/restock`: Restock a sweet, increasing its quantity (Admin only).

Each sweet must have a unique ID, name, category, price, and quantity in stock.

2. Frontend Application

You must build a modern, single-page application (SPA) to interact with your backend API.

- **Technology:** You must use a modern frontend framework like **React** or **Vue**
- **Functionality:**
  - User registration and login forms.
  - A dashboard or homepage to display all available sweets.
  - Functionality to search and filter sweets.
  - A "Purchase" button on each sweet, which should be disabled if the quantity is zero.
  - (For Admin Users) Forms/UI to add, update, and delete sweets.
- **Design:** This is a chance to show your creativity. The application should be visually appealing, responsive, and provide a great user experience.

Process & Technical Guidelines

1. Test-Driven Development (TDD)

Write tests *before* implementing functionality. We expect to see a clear "Red-Green-Refactor" pattern in your commit history, especially for the backend logic. Aim for high test coverage with meaningful test cases.

2. Clean Coding Practices

Write clean, readable, and maintainable code. Follow SOLID principles and other best practices in software design. Your code should be well-documented with meaningful comments and clear naming conventions.

3. Git & Version Control

Use Git for version control. Commit your changes frequently with clear, descriptive messages that narrate your development journey.

- **README Documentation:** Your README.md file must include a detailed section titled **"My AI Usage"**. In this section, you must describe:
  - **Which AI tools you used** (e.g., GitHub Copilot, ChatGPT, Gemini, etc.).
  - **How you used them** (e.g., "I used Gemini to brainstorm API endpoint structures," or "I asked Copilot to generate unit tests for my service layer").

Deliverables

1. **A public Git repository link** (e.g., on GitHub, GitLab).
2. **A comprehensive** README.md **file** that includes:
   a. A clear explanation of the project.

    b. Detailed instructions on how to set up and run the project locally (both backend and frontend).
    c. **Screenshots** of your final application in action.
    d. The mandatory **"My AI Usage"** section.
3. **A test report** showing the results of your test suite.
4. **(Optional - Brownie Points)** A link to the deployed, live application on a platform like Vercel, Netlify, Heroku, or AWS.