**Full Stack Development with MERN**

**Project Documentation format**

# 1. Introduction

* **Project Title:** ShopSmart: Your Digital Grocery Store Experience
* **Team Members:**
* Vullamgunta Gayathri – Database development
* Urumu Mahendra – Frontend Developer
* Tummalagunta Naga Anitha – Backend Developer
* Ummaneni Naga Pravallika – Project Setup & Configuration, Project Implementation & Execution

# 2. Project Overview

* **Purpose:**

The goal of this project is to develop a digital grocery shopping platform where customers can browse, filter, and purchase groceries online. Admins can manage product listings and monitor platform activity.

* **Features:**

**Admin Panel:**

* Secure login/signup
* Add/update/delete grocery products
* View list of customers and orders

**Customer Panel:**

* Browse products by category (e.g., fruits, vegetables, dairy)
* Add to cart and checkout
* View previous orders
* Manage profile

**Product Management:**

* Products include name, description, price, image, category, and stock count.

**Payment:**

* Cash on Delivery (COD) and Card payment options.

# 3. Architecture

# Frontend: Angular (TypeScript), HTML, CSS The frontend is developed using Angular (TypeScript), along with HTML and CSS, to build a responsive and user-friendly interface. It supports role-based access for admins and customers, enabling product management and shopping functionality. Angular services handle API communication, and Angular Material enhances the visual design. The modular structure ensures smooth navigation and maintainability.

# Backend: Node.js with Express.js The backend is powered by Node.js and Express.js, providing RESTful API services to manage the core business logic of the application. It handles user authentication using JWT, secures routes through middleware, and processes requests for products, orders, and users. Data is exchanged in JSON format, and the server environment is configured via dotenv. The backend ensures smooth integration with the frontend and database.

# Database: MongoDB (NoSQL) MongoDB serves as the NoSQL database, storing structured documents for users, products, orders, and cart items. Mongoose is used to define schemas and manage interactions with the database efficiently. Passwords are securely hashed, and validation ensures data consistency. The system is designed to scale easily as features expand.

# 

# 4. Setup Instructions

* **Prerequisites:**
  + Node.js (LTS version recommended)
  + MongoDB (Community Server or a cloud-based service like MongoDB Atlas)
  + Git (for cloning the repository)
* **Installation:**
  1. **Clone the repository:**
  2. git clone < https://github.com/Gayathri14-9/ShopSmart-App>
  3. cd SmartShop-App # Navigate to your project root
  4. **Install Node.js Dependencies:**
  5. npm install
  6. **Set up Environment Variables:** Create or edit the .env file in the project root directory and add:
  7. MONGO\_URI=< "mongodb://127.0.0.1:27017/grocery">
  8. PORT=5000

# 5. Folder Structure

ShopSmart/

├── Backend/

│ ├── connect.js # MongoDB connection setup

│ ├── db/ # Database-related logic

│ │ ├── products.js # Product routes or controller logic

│ │ └── schema.js # Mongoose schema definitions

│ ├── index.js # Entry point of the backend server

│ ├── .env # Environment variables

│ ├── package.json # Backend dependencies and scripts

├── Frontend/

│ ├── public/ # Public assets served as-is

│ │ ├── index.html

│ │ ├── favicon.ico

│ │ ├── logo192.png

│ │ ├── logo512.png

│ │ ├── manifest.json

│ │ └── robots.txt

│ ├── src/ # Source code for the React frontend

│ │ ├── admin\_components/ # Admin-specific React components

│ │ ├── components/ # Shared or user-facing components

│ │ ├── context/ # React context for global state

│ │ ├── App.js # Main App component

│ │ ├── index.js # Frontend entry point

│ │ ├── App.css # App-level styles

│ │ ├── index.css # Global styles

│ │ ├── logo.svg

│ │ ├── setupTests.js # Testing configuration

│ │ └── reportWebVitals.js # Performance reporting

│ ├── package.json # Frontend dependencies and scripts

│ └── package-lock.json # Exact version lock for packages

# 6. Running the Application

To start both the backend server and the frontend application::

**From the project root directory (ShopSmart App/code):**

cd Backend

node index.js

The server will typically run on http://localhost:5000. This handles all API requests such as product listings, cart operations, and order placement.

**In a new terminal, start the frontend:**

cd Frontend

npm install

npm start

The React-based frontend will typically run on:  
[**http://localhost:3000**](http://localhost:3000)

You can then access the full application via your browser, including the customer product view and the admin dashboard.

# 7. API Documentation

All backend endpoints follow a RESTful convention. Below are example routes used in the ShopSmart grocery application. A full Postman collection or Swagger documentation can be prepared separately.

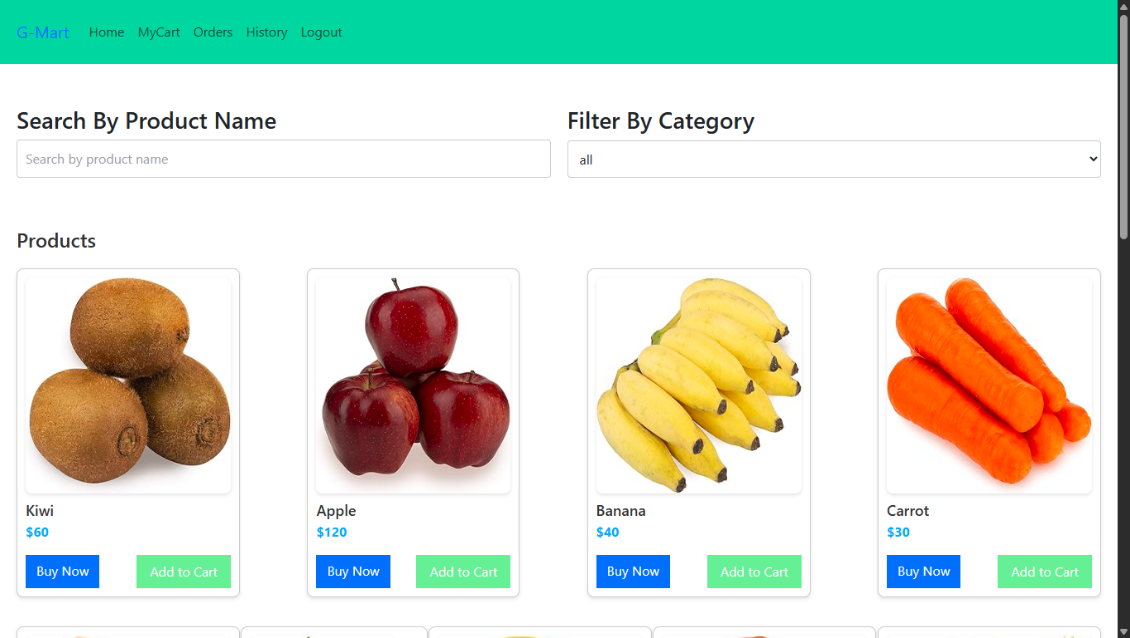
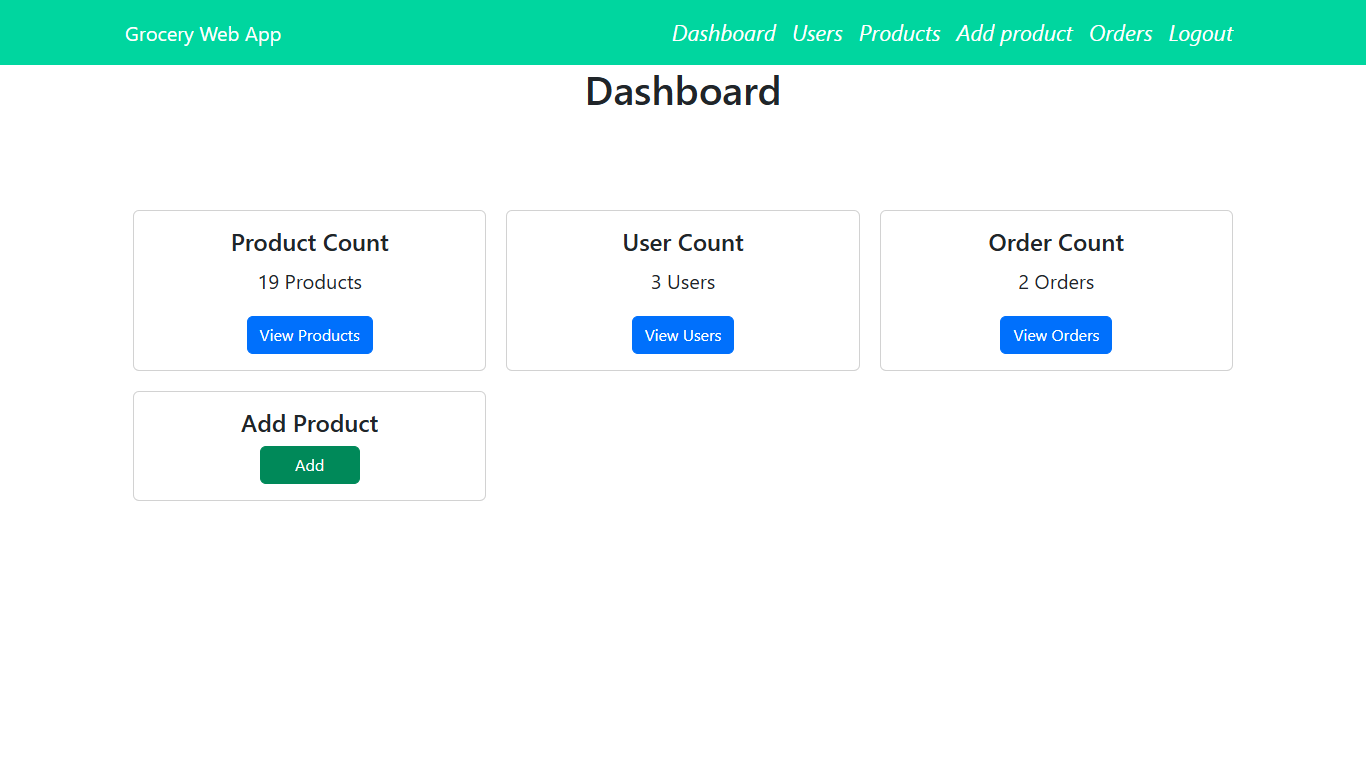
* + **Admin Authentication & Management:**
  + POST /api/admin/login — Admin login and JWT generation
  + GET /api/admin/dashboard — Get admin dashboard metrics (e.g., product counts, users, orders)
  + GET /api/admin/customers — View list of all registered customers
  + POST /api/admin/products — Add a new grocery product
  + PUT /api/admin/products/:id — Update an existing product
  + DELETE /api/admin/products/:id — Delete a product
  + **Product Endpoints (Public & Admin):**
  + GET /api/products — Fetch all available grocery items
  + GET /api/products/:id — Get a single product's details
  + POST /api/products — Add a product (admin only)
  + PUT /api/products/:id — Update product info (admin only)
  + DELETE /api/products/:id — Delete a product (admin only)
  + **Cart & Order Endpoints (Customer Protected):**
  + POST /api/cart/add — Add an item to the cart
  + GET /api/cart — Get current cart contents
  + DELETE /api/cart/:itemId — Remove an item from the cart
  + POST /api/orders — Place a new order
  + GET /api/orders — View customer's past orders
  + PUT /api/orders/:id/status — Update order status (admin only)
  + **Customer Profile & Authentication:**
  + POST /api/auth/register — Register a new customer
  + POST /api/auth/login — Login and get JWT
  + GET /api/auth/profile — Get customer profile (requires JWT)
  + PUT /api/auth/profile — Update customer profile
  + PUT /api/auth/password — Change password

# 8. Authentication

# Authentication in the ShopSmart app is implemented using JSON Web Tokens (JWT) for secure session management. Upon successful login, the backend generates a JWT that contains the user’s ID and role (e.g., admin or customer). This token is sent to the frontend and typically stored in the browser's localStorage for persistent access. For all protected API routes, the frontend includes the token in the Authorization header using the format Bearer <YOUR\_TOKEN>.

# The backend uses middleware to validate and decode the JWT. The auth.js middleware (protect function) ensures the token is valid and attaches the user data to req.user. Additional middleware functions check the user's role and permissions to allow or restrict access to specific routes. Passwords are hashed securely using bcrypt.js before being stored in the database, ensuring data protection even if the database is compromised.

1. **User Interface**



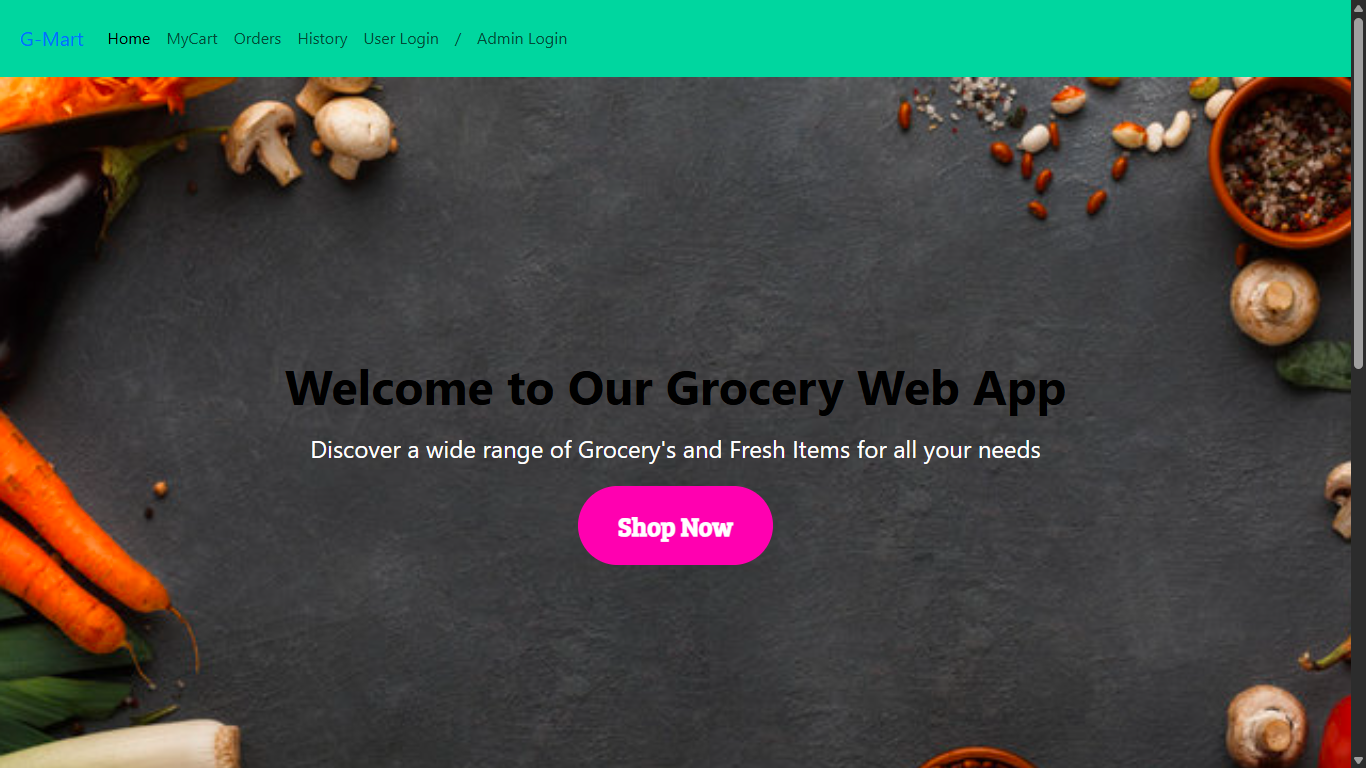
1. **Testing**

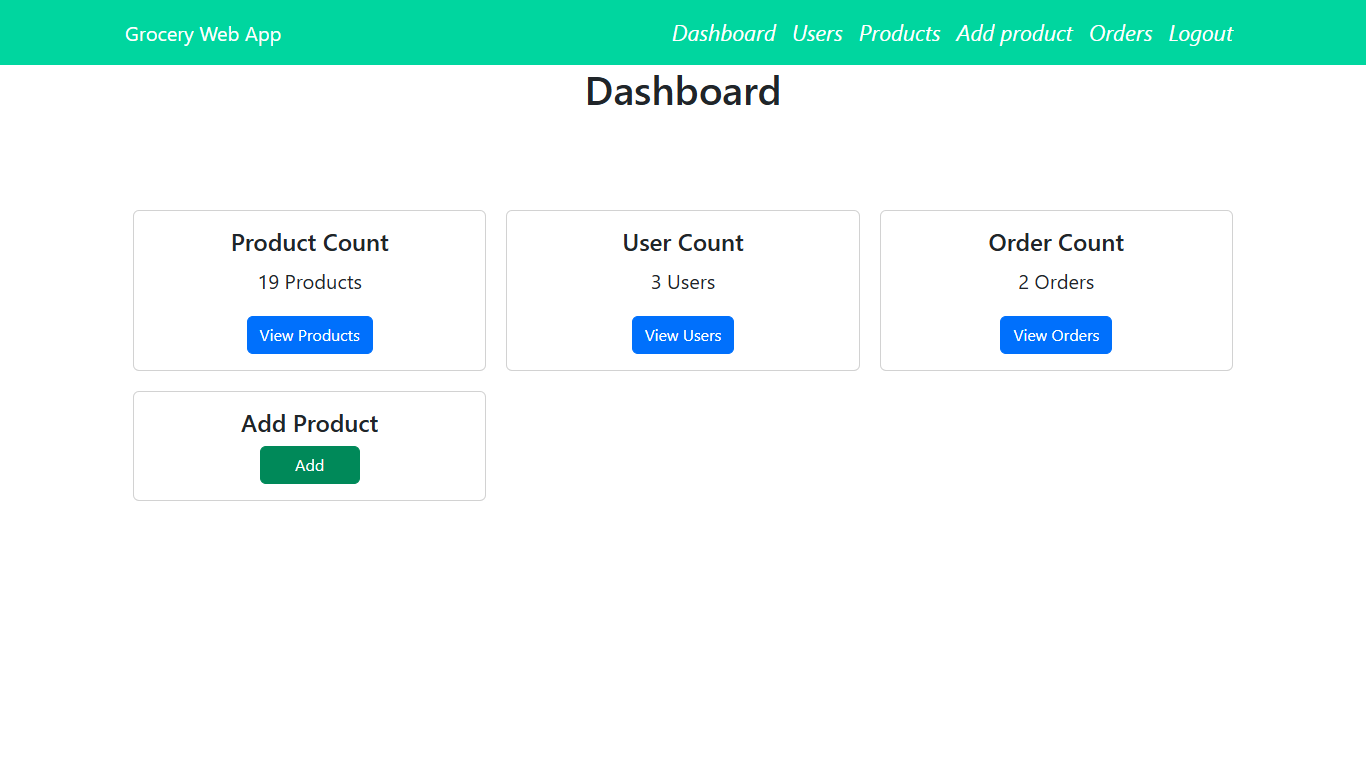
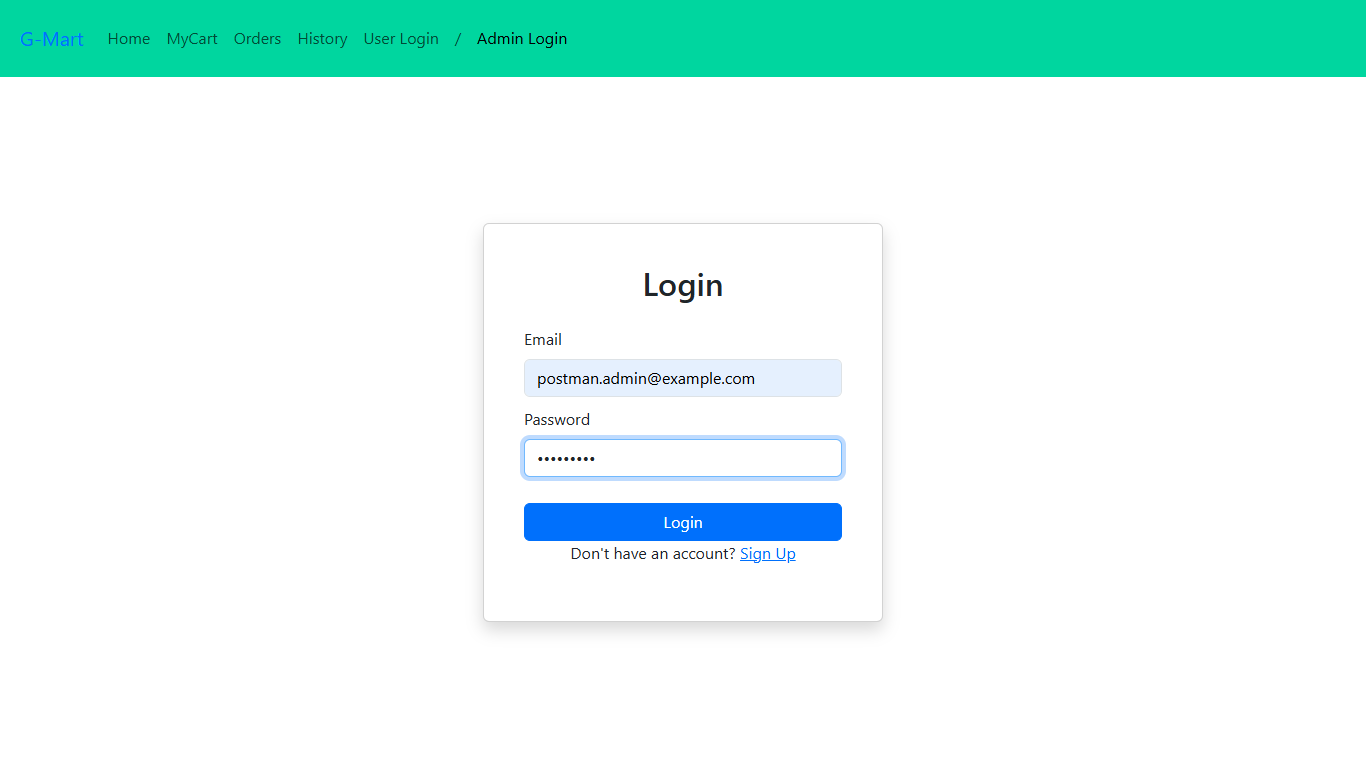
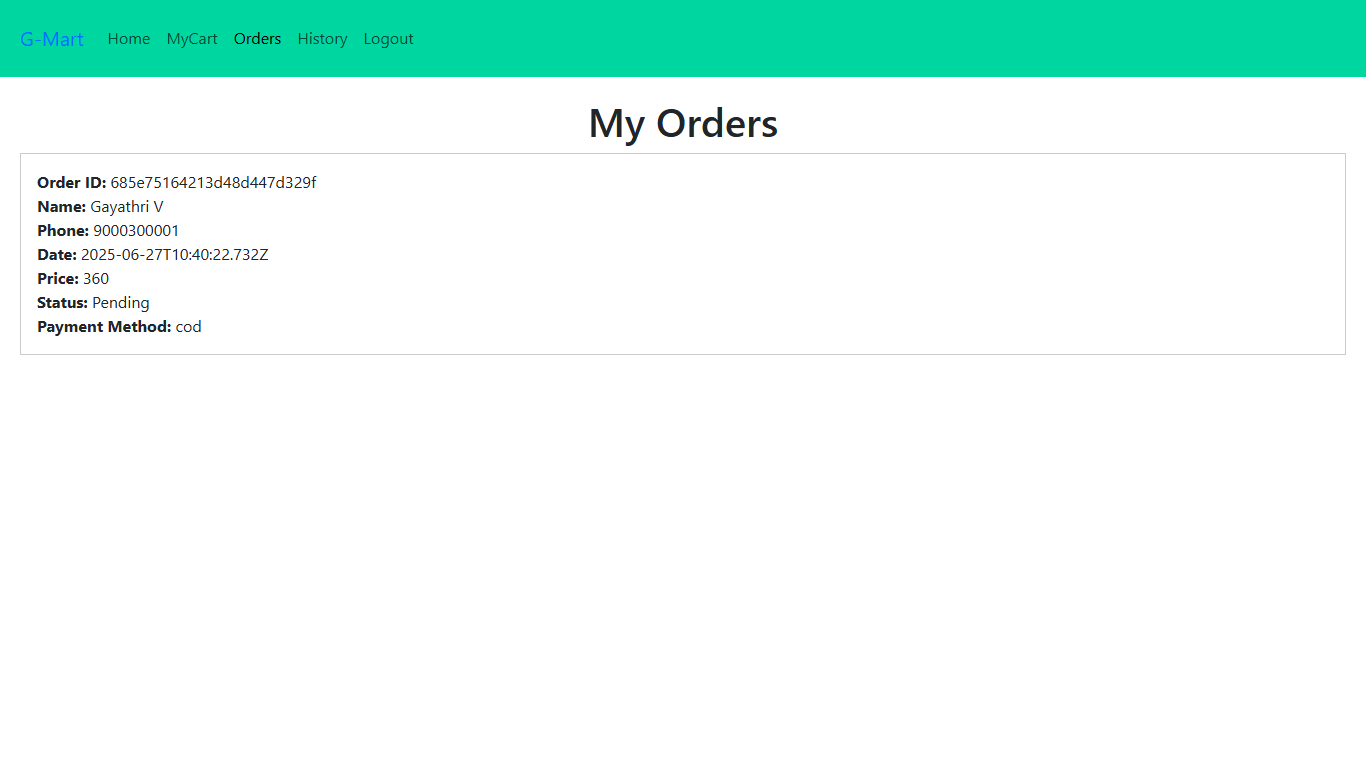
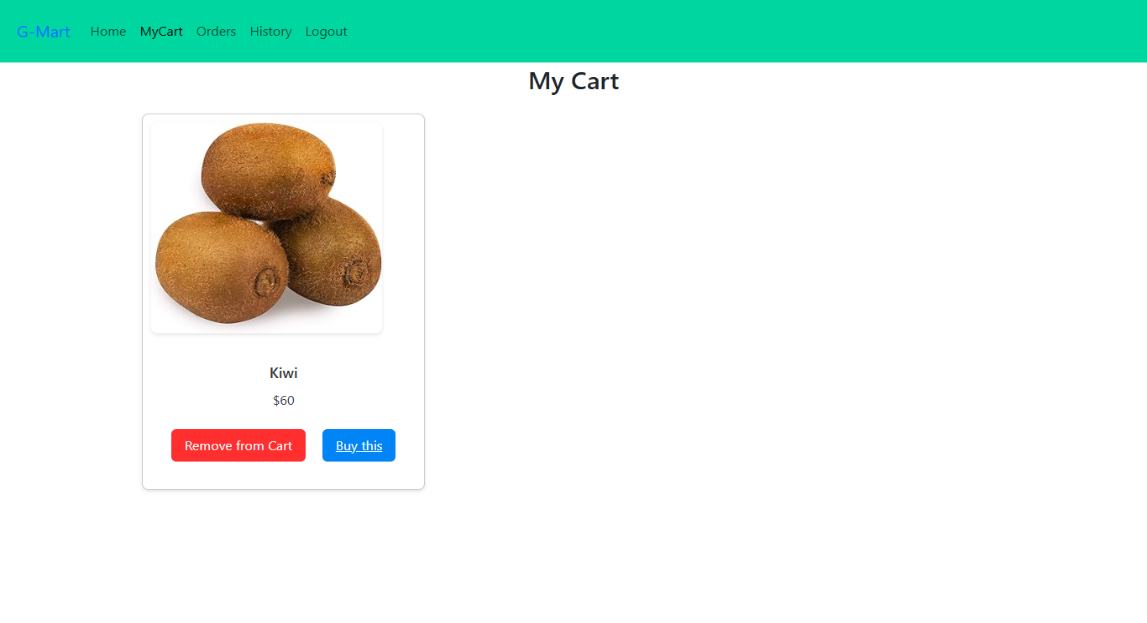
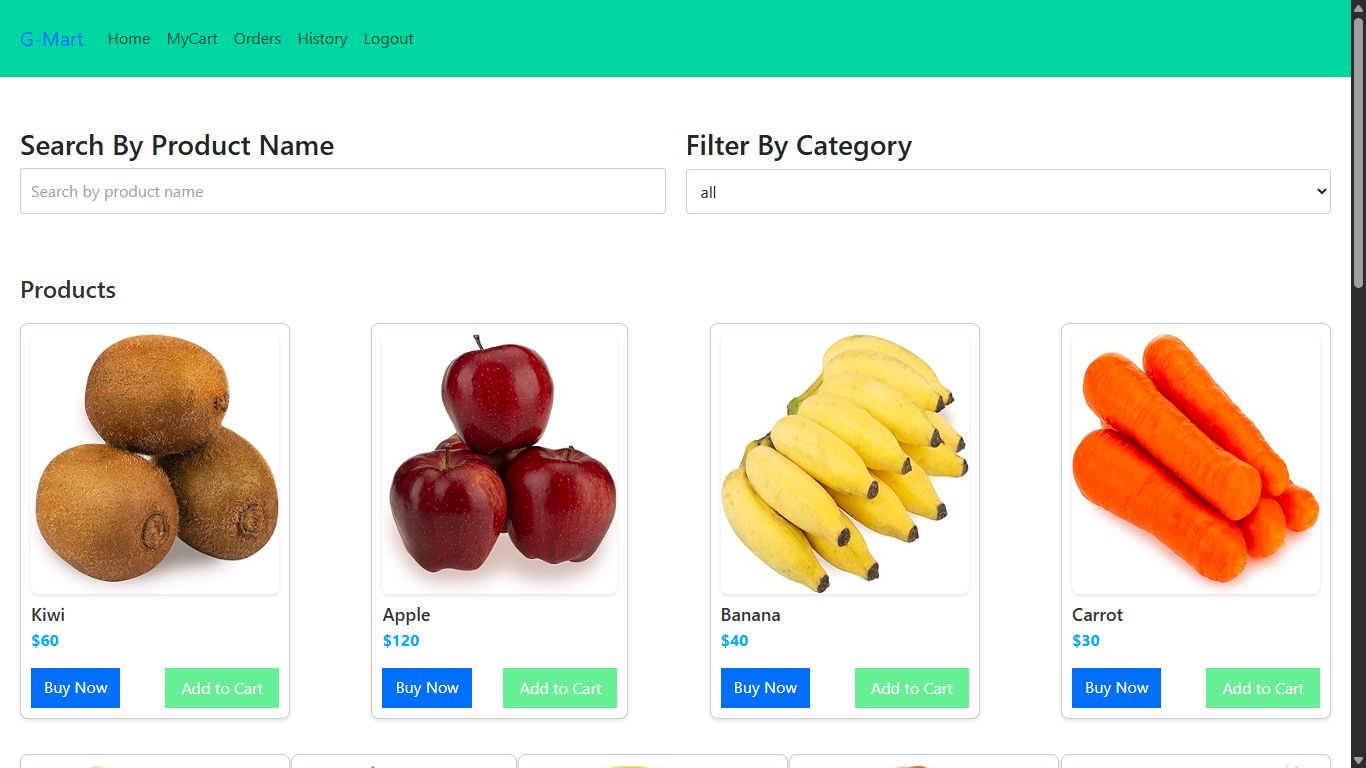
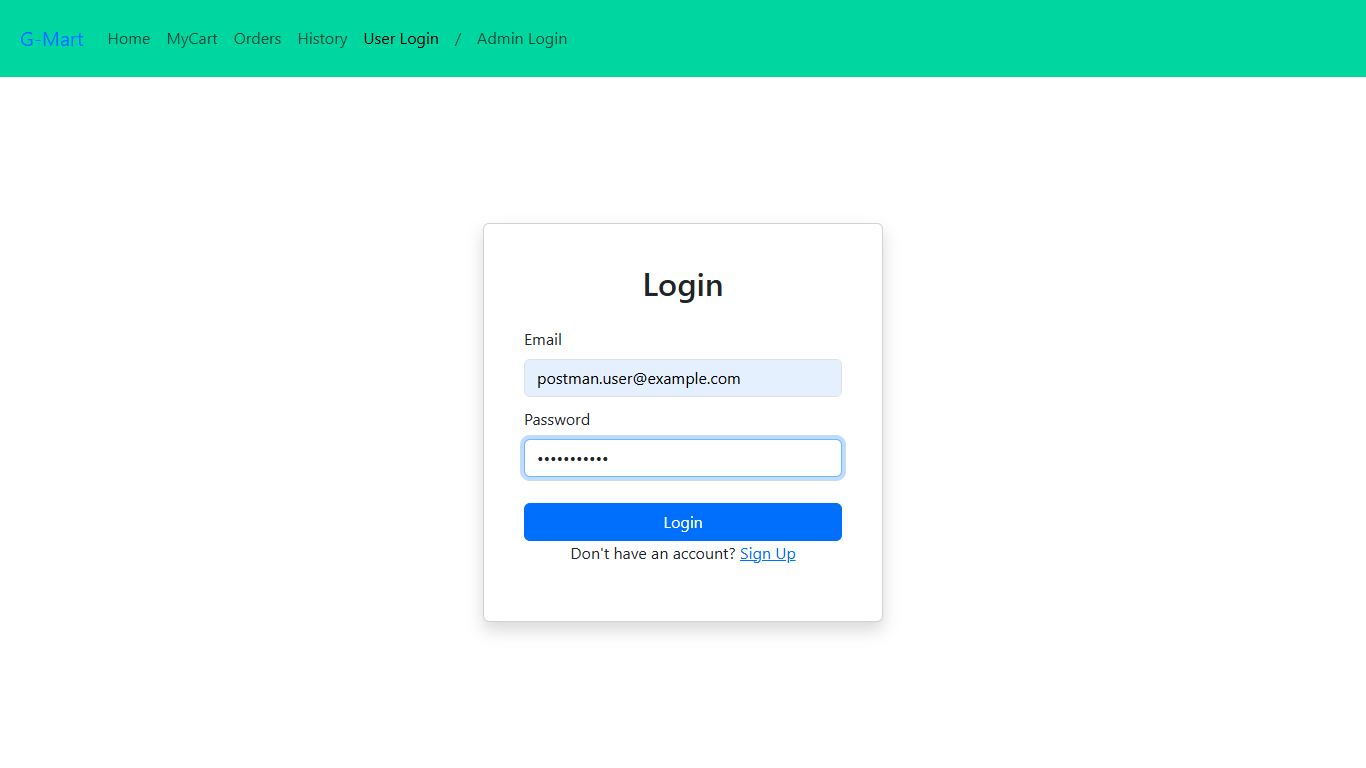
**Testing Strategy:**  
The ShopSmart project employed a combination of **manual testing** and **unit testing** (where applicable) to ensure application functionality, usability, and robustness. Manual testing was carried out across different user flows such as admin login, product management, and customer order placement. Additionally, **User Acceptance Testing (UAT)** was conducted using structured templates aligned with user stories to validate that the system met stakeholder expectations.

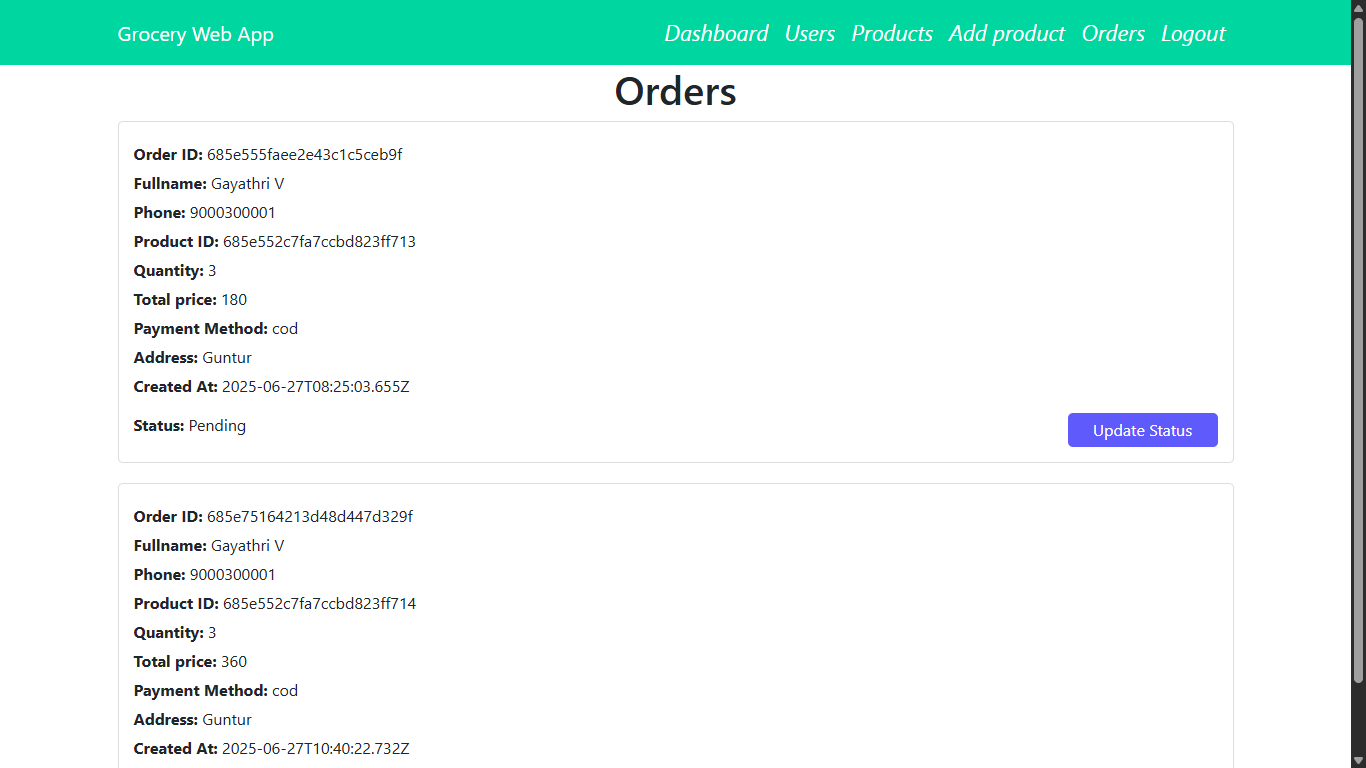
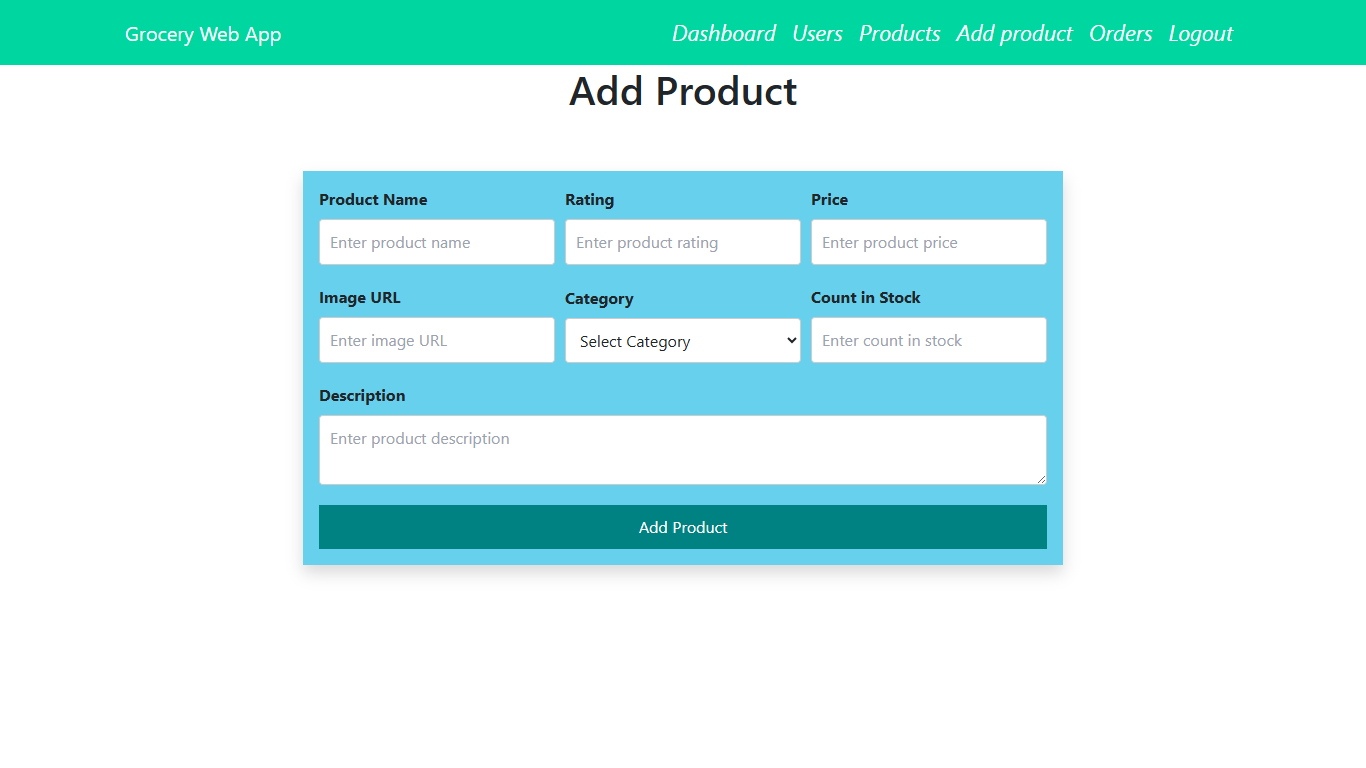
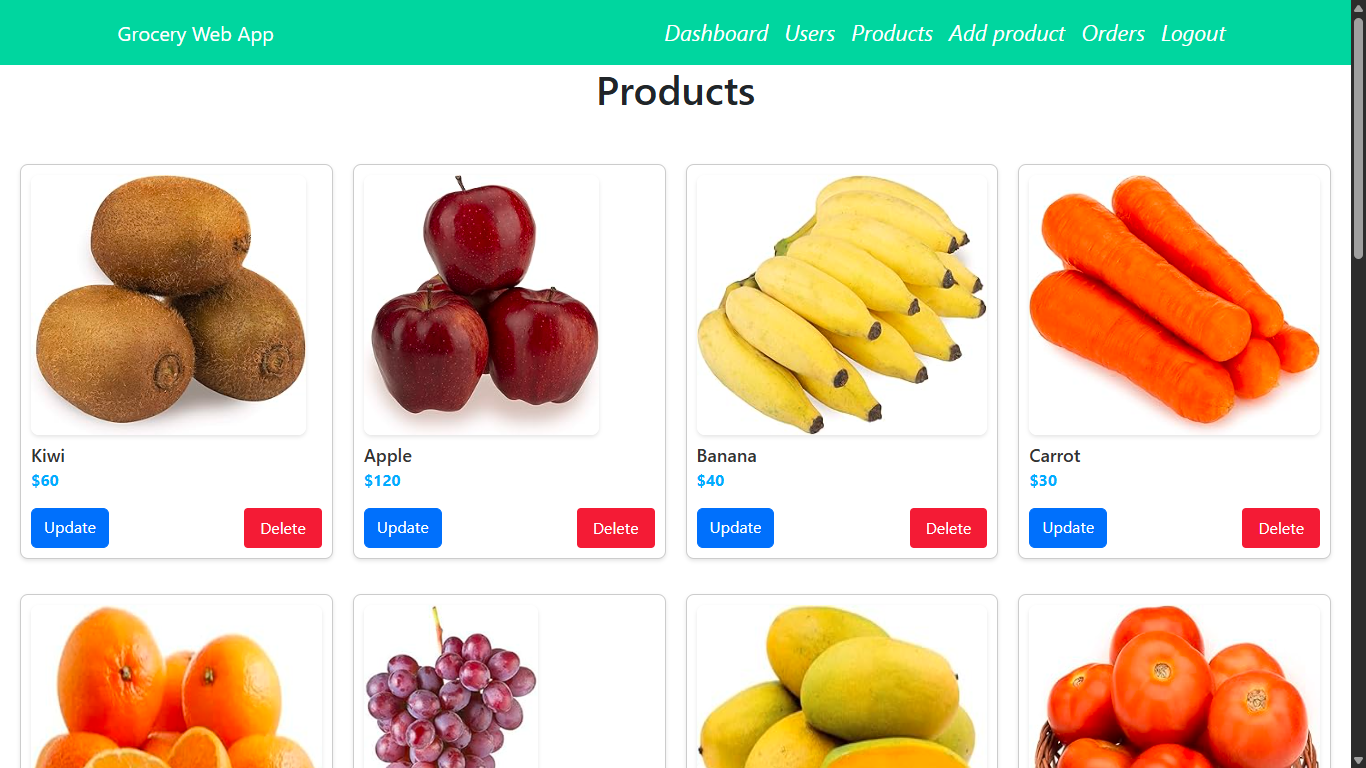
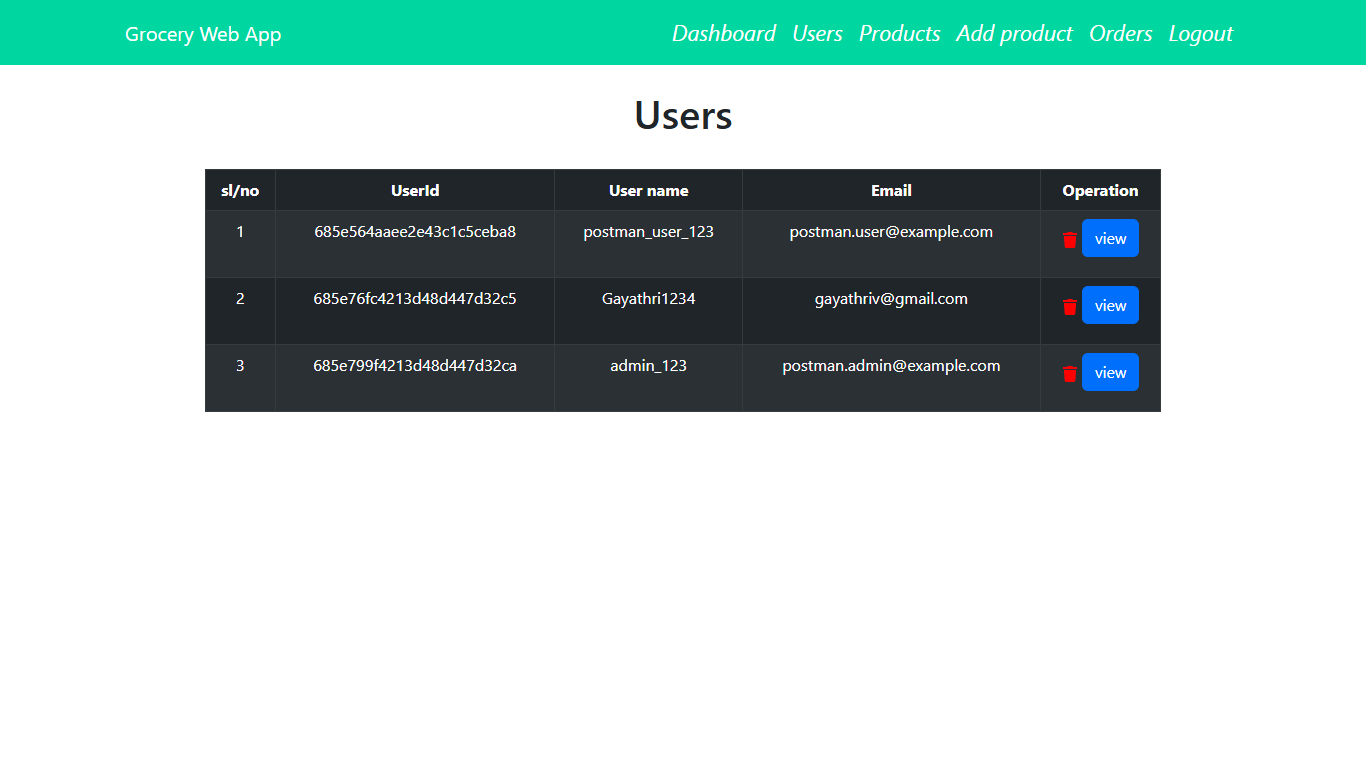
**Tools Used:**

**Postman** was used to test and verify backend API endpoints,**Browser Developer Tools** (Chrome DevTools) were used to inspect frontend behavior and debug UI issues,**Jest** or other JavaScript-based test frameworks may be integrated for unit testing React components or backend logic, though not mandatory in the current version.

1. **Screenshots or Demo**





****

**12.** **Known Issues**

Minor UI inconsistencies appear on some mobile screens. In a few cases, form validations may not trigger instantly during quick submissions. Product images can load slowly on weak networks. The order history may sometimes require a manual refresh to display recent orders.

1. **Future Enhancements**

Future updates to ShopSmart may include advanced search and filtering options, such as filtering products by brand, price range, or dietary preference. Real-time delivery tracking using APIs like Google Maps can enhance user experience. Integrating push notifications or WebSocket-based alerts will keep users updated on order status. Online payment gateways such as Stripe or Razorpay can be added to support digital transactions. Additional features like product reviews, discount campaigns, and performance monitoring tools (e.g., Grafana) will increase reliability and engagement.