

ShopSmart: Your Digital Grocery Store Experience

Team ID : LTVIP2025TMID48554

Team Leader : M. Naga Sirisha.

1.Introduction:

ShopSmart is an innovative and robust full-stack grocery shopping platform designed to bring convenience and efficiency to both consumers and sellers in the grocery domain. As modern lifestyles increasingly demand digital solutions, ShopSmart serves as a bridge between traditional grocery shopping and modern e-commerce, providing a smart, user-friendly alternative.

Built with the **MERN stack** — **MongoDB**, **Express.js**, **React.js**, and **Node.js** — the platform integrates powerful technologies to deliver a responsive, scalable, and interactive web application. Each layer of the stack contributes to the seamless flow of data and user actions:

- **MongoDB** is used as the NoSQL database for storing user data, product information, and order histories.
- **Express.js** acts as the backend framework handling API requests and routing.
- **React.js** provides a dynamic and responsive user interface for customers and administrators.
- **Node.js** powers the server-side logic and real-time functionalities.

Core Objectives:

- Enable customers to **browse, search, and purchase** groceries online from a curated selection of categories including fruits, vegetables, dairy, beverages, snacks, and household items.
- Provide sellers with the ability to **add, update, and track** inventory and sales.
- Empower administrators with tools to **monitor platform activity**, manage users and product listings, and maintain overall system health.

Key Highlights:

- **Modular Design:** The codebase is organized by functionality (e.g., authentication, product management, order processing), making it easy to scale and maintain.
- **Real-Time Operations:** Cart updates, order placements, and admin dashboards reflect changes in real time.
- **Role-Based Access Control:** Secure login and access are provided based on user roles — customer, seller, or admin — ensuring each user has access only to relevant features.
- **Responsive UI/UX:** The frontend is built using React with modern UI libraries to ensure mobile and desktop responsiveness.

Scenario: Late Night Shopping, Next Morning Delivery

User: Priya, a busy software engineer in her late 20s.

Time: 10:00 PM, after a long day at work.

Problem:

Priya realizes she's out of daily essentials—**milk, vegetables, and rice**—which she needs the next morning. With local stores closed or inconvenient, she cannot buy them right away.

She's tired, wants to relax, and doesn't want to waste time looking for open stores or standing in queues.

Solution: How ShopSmart Helps

1. Quick Access:

Priya opens the **ShopSmart** mobile/web app from the comfort of her home.

2. Easy Login:

She logs into her existing account. Her delivery address and preferences are already saved.

3. Product Discovery:

She browses through categorized sections:

- **Dairy** → Adds 2L milk
- **Vegetables** → Adds tomatoes, onions, spinach
- **Grains** → Adds 5kg rice

The app's smart filters help her find fresh and available items easily.

4. Cart & Quantity Selection:

She sets the quantity for each item, reviews the prices, and adds them to the cart.

5. Payment Method:

She chooses **Cash on Delivery (COD)** for added flexibility.

6. Order Placement & Notification:

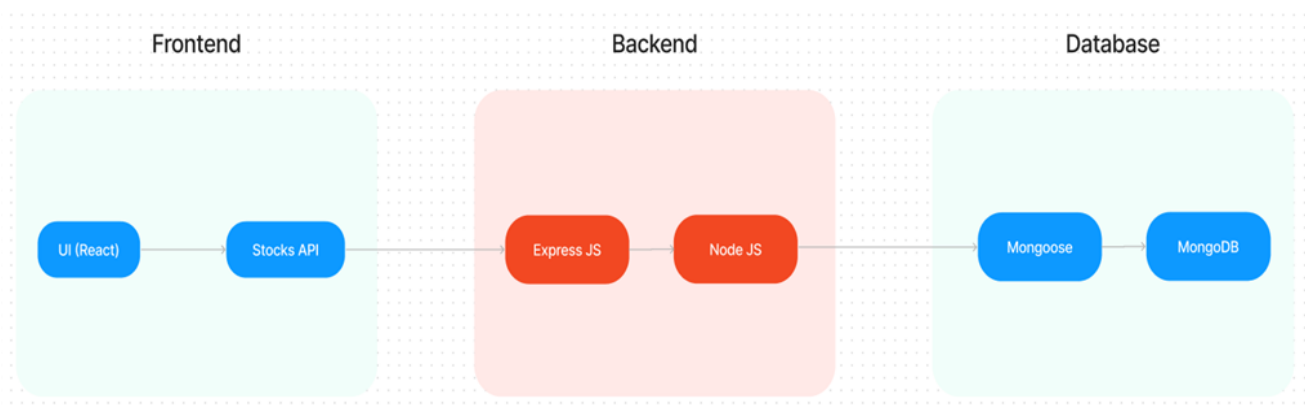
She confirms the order. Instantly, she receives:

- An **SMS** and **email** confirmation
- A tracking link to monitor delivery status

7. Next Morning Delivery:

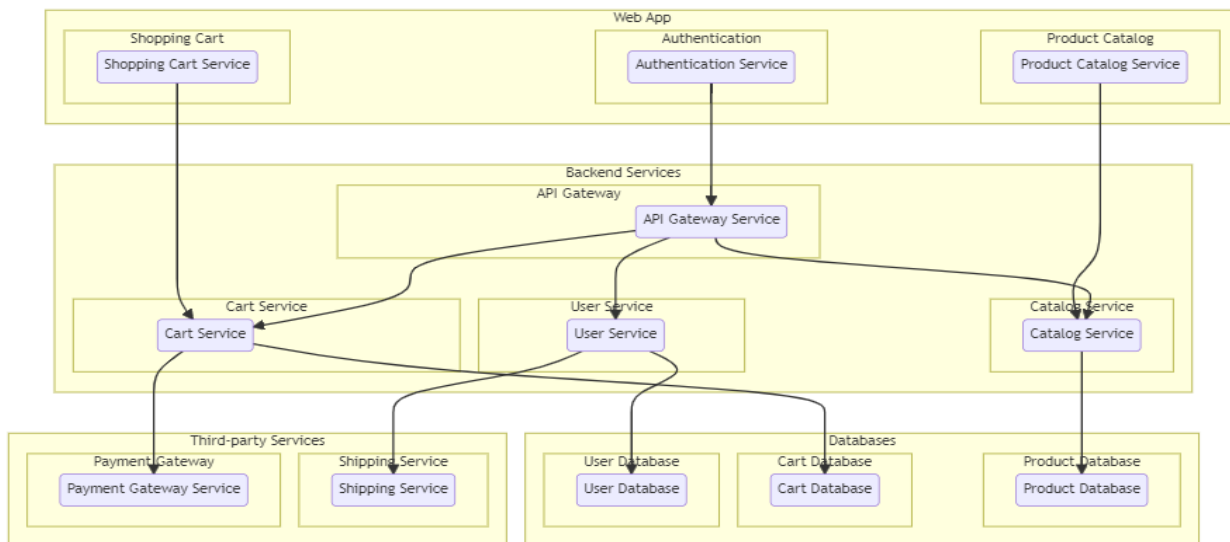
Her groceries arrive early the next day—fresh and on time—while she prepares for work. No hassle, no delay.

2. TECHNICAL ARCHITECTURE:

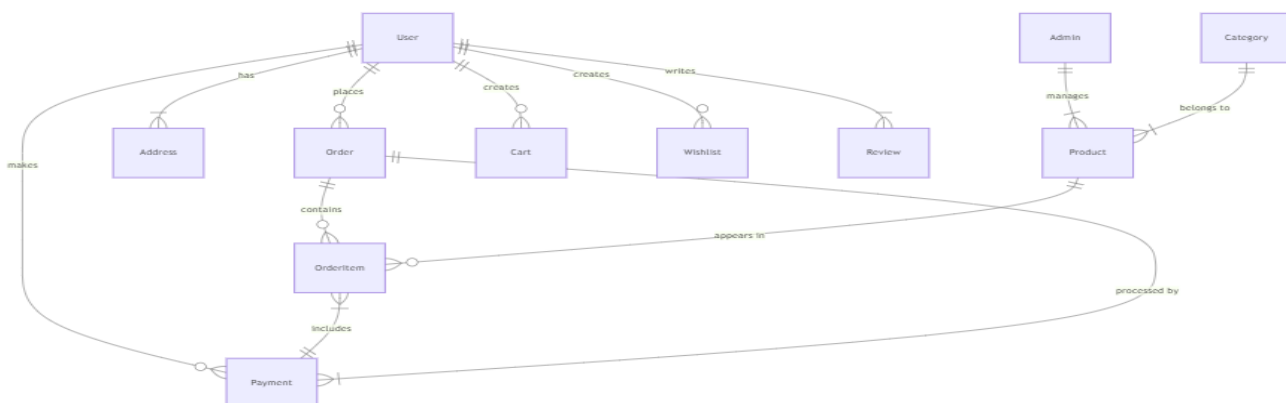


In this architecture diagram:

- The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Cart, Products, Profile, Admin dashboard, etc.,
- The backend is represented by the "Backend" section, consisting of API endpoints for Users, Orders, Products, etc., It also includes Admin Authentication and an Admin Dashboard.
- The Database section represents the database that stores collections for Users, Admin, Cart, Orders, and products.



ER-Diagram



3.Pre-Requisites:

To develop a full-stack food ordering app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side. • Download: <https://nodejs.org/en/download/>

- Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS go through the below provided link:

npm install

Start the Development Server:

- To start the development server, execute the following command:

npm run dev or npm run start

- The e-commerce app will be accessible at **http://localhost:3000** by default. You can change the port configuration in the .env file if needed.

Access the App:

- Open your web browser and navigate to **http://localhost:3000**.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the Shoptmart app on your local machine. You can now proceed with further customization, development, and testing as needed.

4.Application Flow

The application flow is divided among three key user types — **Customers**, **Sellers**, and **Administrators** — each with unique permissions and responsibilities.

Customer Flow:

- A user begins by **registering** with basic information like name, email, and password.

- After successful login, the user is **redirected to the home page** where all grocery items are displayed.
- Customers can:
 - **Browse and search** for products using filters like category and price.
 - **Add products to cart**, adjust quantities, and view subtotal dynamically.
 - **Proceed to checkout**, where they enter their delivery address and select a payment method.
 - **Confirm their order**, after which they receive a **confirmation message**.
- Users can later:
 - **View past orders** in their profile section.
 - **Track delivery status** of current orders.
 - **Update their profile** and personal information.

Seller Flow:

- Sellers register via a **dedicated registration form**.
- Their account status remains **inactive until an admin approves it**.
- Once approved, sellers can:
 - **Log in to their seller dashboard**.
 - **Add new products**, including name, description, image, price, and stock.
 - **Update or remove listings** based on product availability.
 - **Monitor customer orders** and track sales history.

Admin Flow:

- Admins access the application through a **separate login portal**.
- Upon login, they are redirected to an **admin dashboard** with full control over:
 - User accounts (customers & sellers)
 - Product management (add, update, delete)
 - Order status (monitor and process)
 - **Seller approval workflows**
 - Removal of suspicious or inactive accounts

This **role-based access control** ensures each type of user experiences the system based on their responsibilities.

5.Project Setup And Configuration

Install required tools and software:

- Node.js.

Reference Article: <https://www.geeksforgeeks.org/installation-of-node-js-on-windows/>

- Git.

Reference Article: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

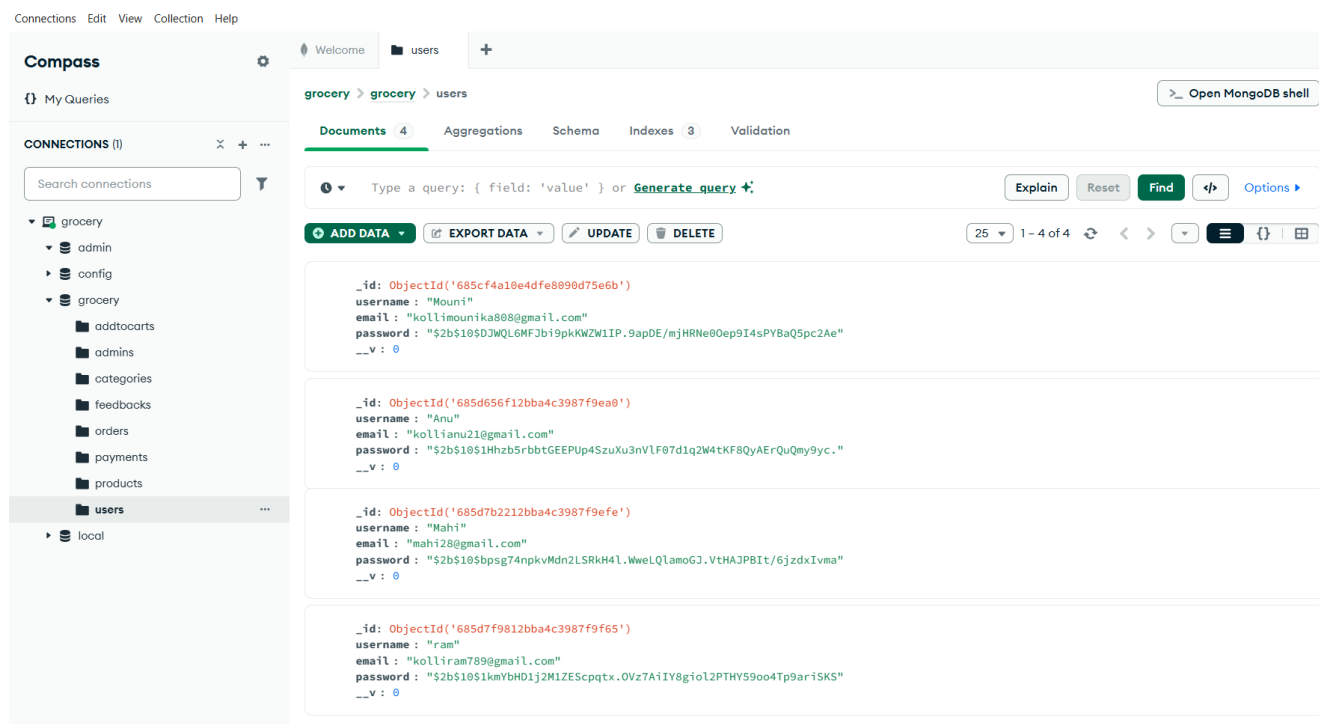
Create project folders and files:

- Client folders.

- Server folders

Create database in cloud

- Install Mongoose.
- Create database connection.



Installation Steps:

Backend Setup

cd server

npm install

Frontend Setup

cd client

npm install

Create a .env file in the backend:

MONGO_URI=your_mongo_connection_string

JWT_SECRET=your_secret_key

PORT=5000

Run the servers:

Backend npm run dev

Frontend npm start

6. Project Structure:

Frontend (client/)

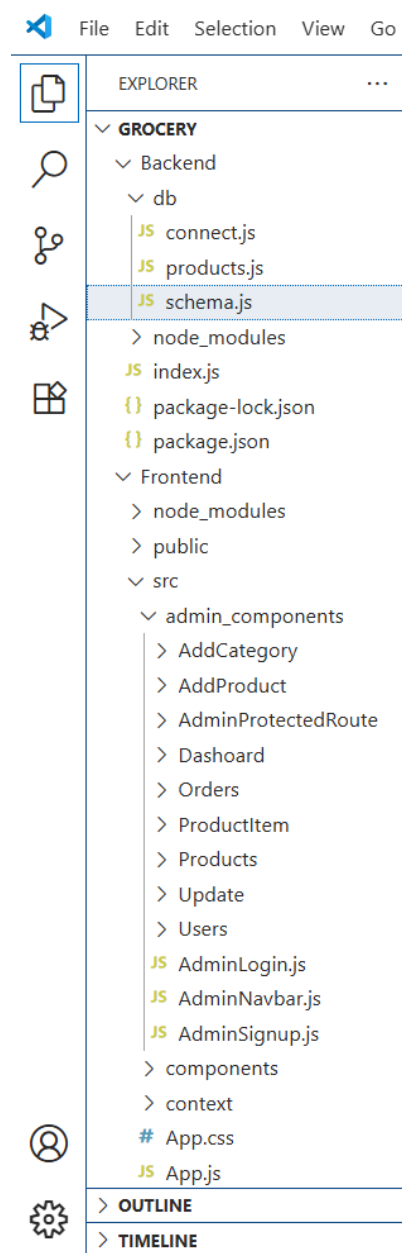
- components/: UI elements like ProductCard, Navbar, CartItem
- pages/: Page components like Home, Login, Register, AdminDashboard
- context/: Global state for auth and cart
- services/: Axios-based API calls
- App.js: Application root and routing setup

Backend (server/)

- models/: Mongoose schemas for User, Product, Cart, Order
- routes/: Express route handlers
- controllers/: Logic for handling requests
- middleware/: Auth and error handlers
- server.js: Main server file connecting all pieces

7. Schema Use-Cases

- **User Schema:**
 - Fields: name, email, password (hashed), role
 - Role determines if the user is a customer or admin
- **Product Schema:**
 - Fields: name, description, price, stock, image, category
 - Represents available grocery items



- **Cart Schema:**
 - Fields: userId (ref), products, quantity, timestamps
 - Temporary container for selected items
- **Order Schema:**
 - Fields: userId, products, shipping address, total, status
 - Stores finalized orders
- **Admin Schema:**
 - Manages credentials and admin-level control

8.Backend Development

Set Up Project Structure:

- Create a new directory for your project and set up a package.json file using the npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

1. Setup express server:

- Create index.js file.
- Create an express server on your desired port number.
- Define API's

2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, restaurants, food products, orders, and other relevant data.

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.

- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like products, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.

- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

9.Frontend development

1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

2.Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3.Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Article Link:

https://www.w3schools.com/react/react_getstarted.asp

Access:

- Frontend: <http://localhost:3000>
- Backend API: <http://localhost:5000>

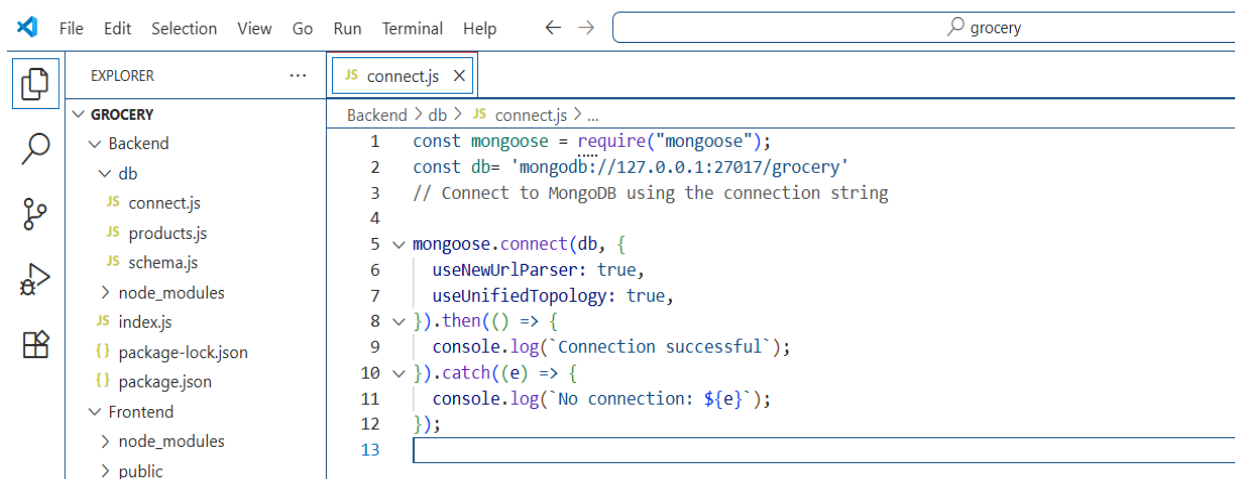
10.Code Explanation:

- JWT token is issued upon successful login/register.
- Stored in localStorage on the frontend.
- Sent with API requests using Axios headers.

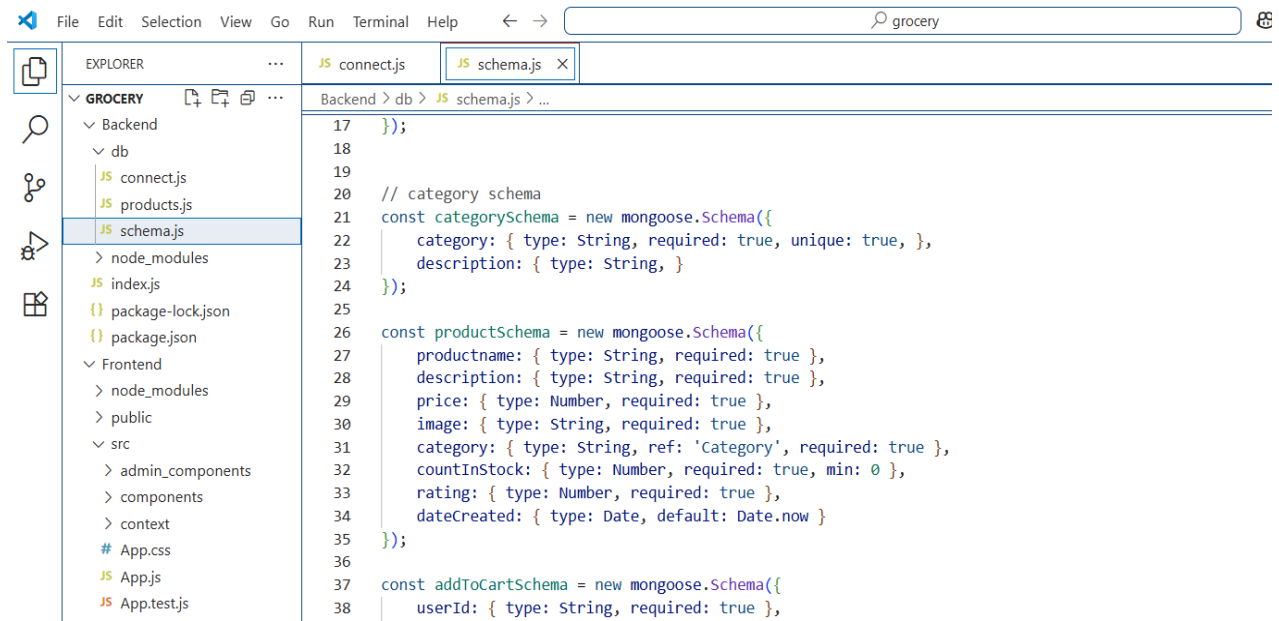
- Middleware validates the token: Passwords are hashed using bcrypt before saving.

```
const auth = (req, res, next) => {  
  const token = req.header("x-auth-token");  
  if (!token) return res.status(401).send("Access Denied");  
  try {  
    const verified = jwt.verify(token, process.env.JWT_SECRET);  
    req.user = verified;  
    next();  
  } catch (err) {  
    res.status(400).send("Invalid token");  
  }  
};
```

Connect.js in Backend:



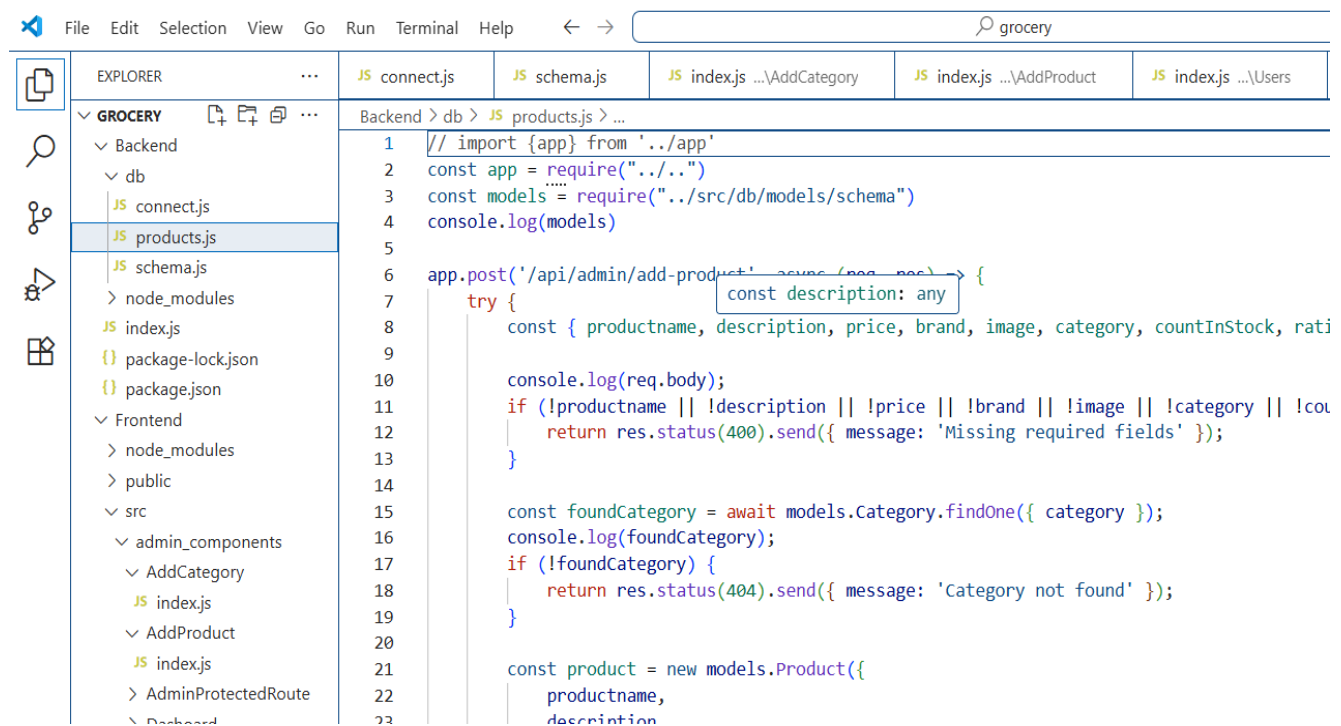
Schema.js in Backend:



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'GROCERY' project is expanded, showing the 'Backend' folder, which contains a 'db' folder. Inside 'db', the 'schema.js' file is selected. The main editor displays the content of 'schema.js', which defines Mongoose schemas for 'category' and 'product', and an 'addToCart' schema. The 'category' schema has fields 'category' (string, required, unique) and 'description' (string). The 'product' schema has fields 'productname', 'description', 'price', 'image', 'category' (reference to 'Category'), 'countInStock', 'rating', and 'dateCreated'. The 'addToCart' schema has a 'userid' field.

```
17 });
18
19
20 // category schema
21 const categorySchema = new mongoose.Schema({
22   category: { type: String, required: true, unique: true, },
23   description: { type: String, }
24 });
25
26 const productSchema = new mongoose.Schema({
27   productname: { type: String, required: true },
28   description: { type: String, required: true },
29   price: { type: Number, required: true },
30   image: { type: String, required: true },
31   category: { type: String, ref: 'Category', required: true },
32   countInStock: { type: Number, required: true, min: 0 },
33   rating: { type: Number, required: true },
34   dateCreated: { type: Date, default: Date.now }
35 });
36
37 const addToCartSchema = new mongoose.Schema({
38   userid: { type: String, required: true },
```

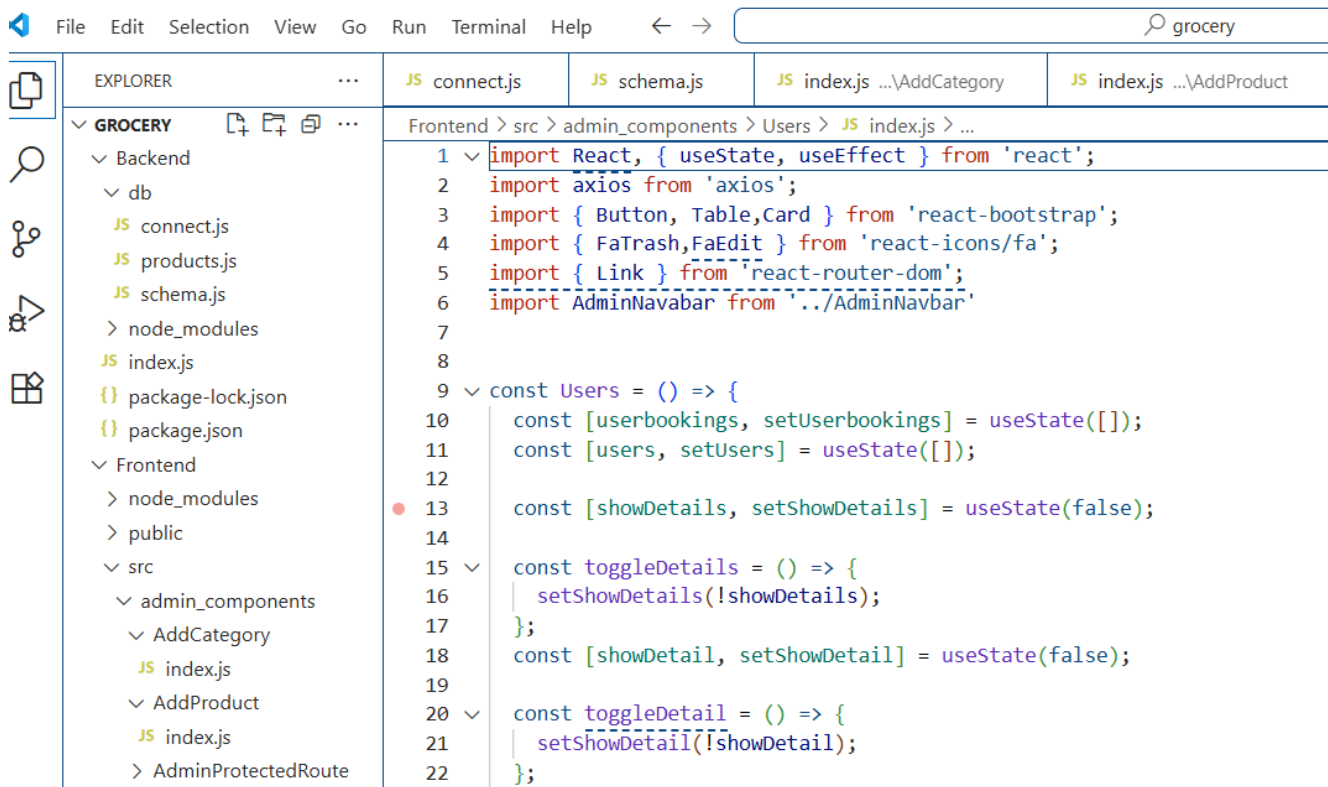
Product.js in Backend:



The screenshot shows the VS Code interface with the Explorer sidebar on the left. The 'GROCERY' project is expanded, showing the 'Backend' folder, which contains a 'db' folder. Inside 'db', the 'products.js' file is selected. The main editor displays the content of 'products.js', which imports the 'app' module and the 'models' module. It defines a POST endpoint '/api/admin/add-product' that takes a request body and returns a response. The endpoint checks for required fields (productname, description, price, brand, image, category, countInStock, rating) and returns a 400 status if any are missing. It also checks if the category exists and returns a 404 status if not found. Finally, it creates a new product and returns it.

```
1 // import {app} from '../app'
2 const app = require("../..")
3 const models = require("../src/db/models/schema")
4 console.log(models)
5
6 app.post('/api/admin/add-product', async (req, res) => {
7   try {
8     const { productname, description, price, brand, image, category, countInStock, rating } = req.body
9
10    console.log(req.body);
11    if (!productname || !description || !price || !brand || !image || !category || !countInStock || !rating) {
12      return res.status(400).send({ message: 'Missing required fields' });
13    }
14
15    const foundCategory = await models.Category.findOne({ category });
16    console.log(foundCategory);
17    if (!foundCategory) {
18      return res.status(404).send({ message: 'Category not found' });
19    }
20
21    const product = new models.Product({
22      productname,
23      description,
```

Users.js in frontend:



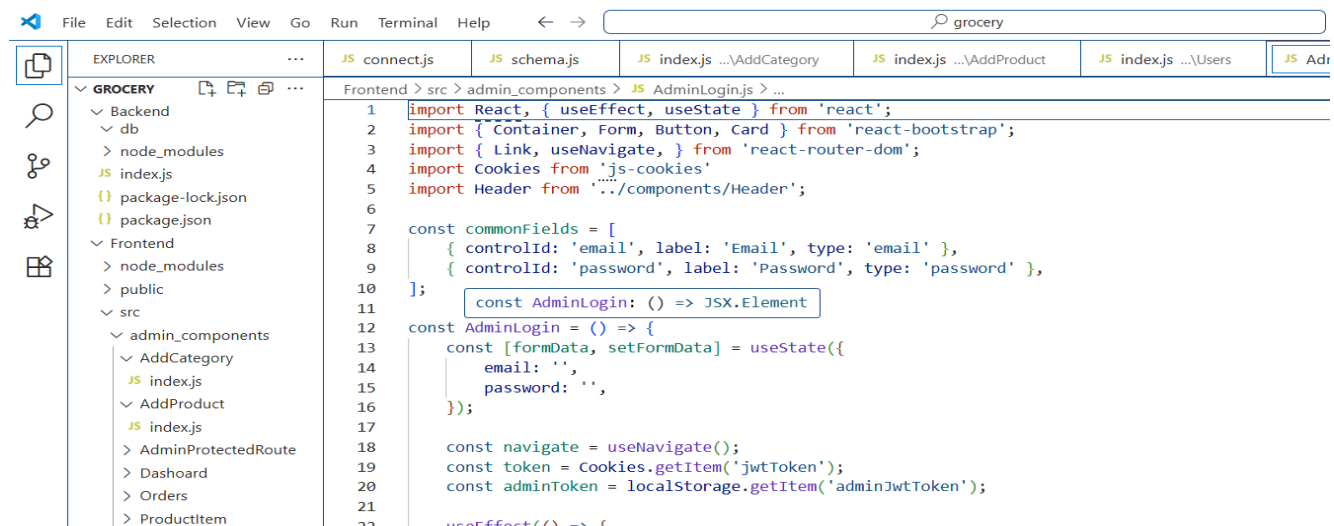
```
File Edit Selection View Go Run Terminal Help ← → grocery
```

EXPLORER ... JS connect.js JS schema.js JS index.js ...\AddCategory JS index.js ...\AddProduct

Frontend > src > admin_components > Users > JS index.js > ...

```
1 import React, { useState, useEffect } from 'react';
2 import axios from 'axios';
3 import { Button, Table, Card } from 'react-bootstrap';
4 import { FaTrash, FaEdit } from 'react-icons/fa';
5 import { Link } from 'react-router-dom';
6 import AdminNavbar from '../AdminNavbar';
7
8
9 const Users = () => {
10   const [userbookings, setUserbookings] = useState([]);
11   const [users, setUsers] = useState([]);
12
13   const [showDetails, setShowDetails] = useState(false);
14
15   const toggleDetails = () => {
16     setShowDetails(!showDetails);
17   };
18   const [showDetail, setShowDetail] = useState(false);
19
20   const toggleDetail = () => {
21     setShowDetail(!showDetail);
22   };
23 }
```

AdminLogin.js in frontend:



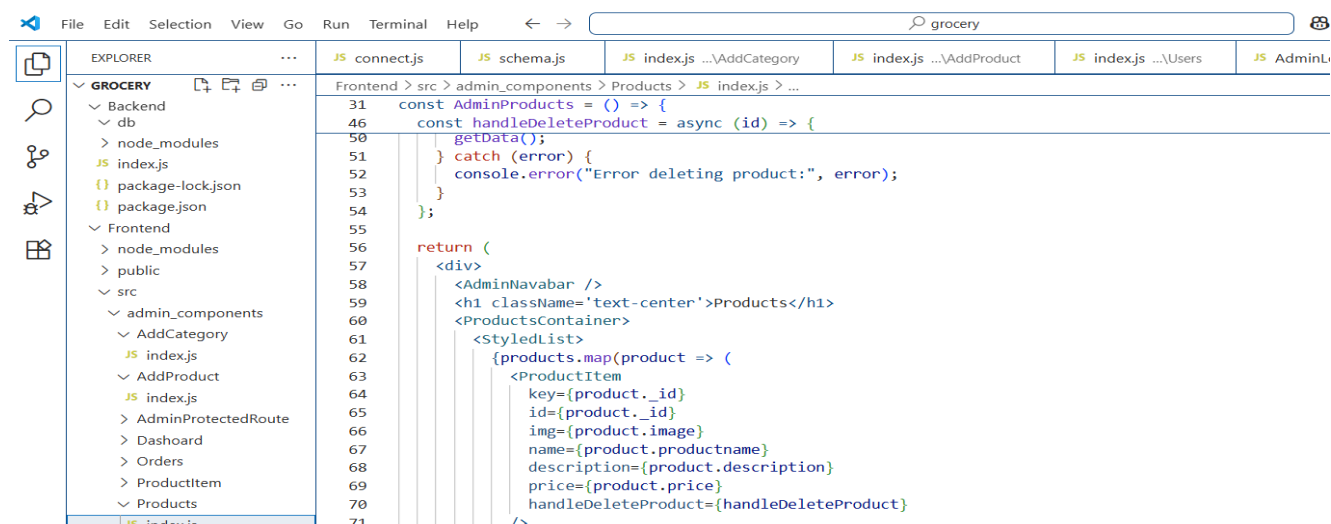
```
File Edit Selection View Go Run Terminal Help ← → grocery
```

EXPLORER ... JS connect.js JS schema.js JS index.js ...\AddCategory JS index.js ...\AddProduct JS index.js ...\Users JS Adr

Frontend > src > admin_components > JS AdminLogin.js > ...

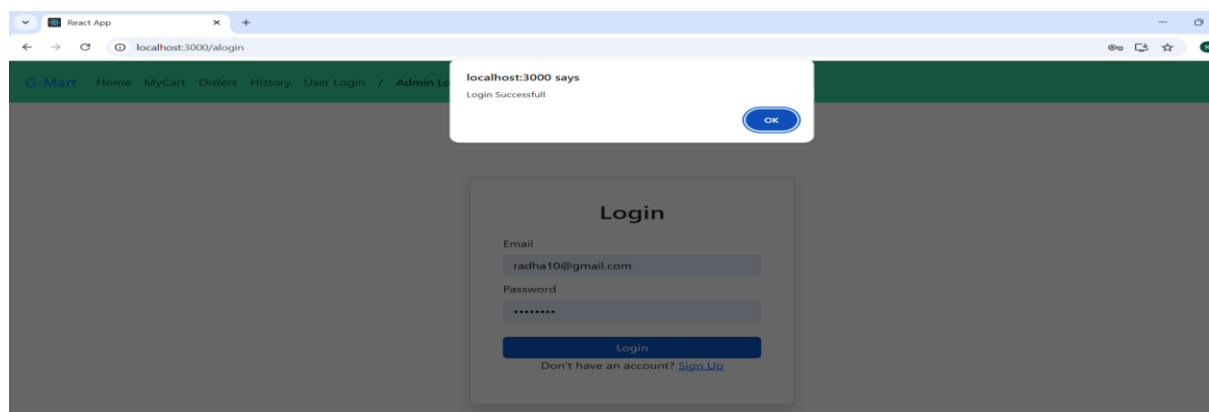
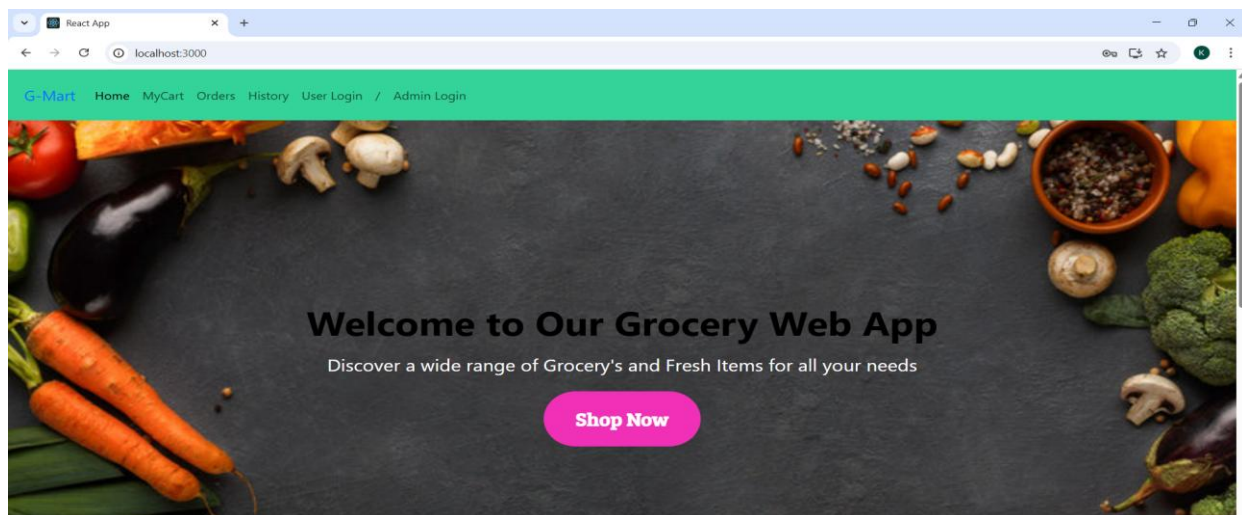
```
1 import React, { useEffect, useState } from 'react';
2 import { Container, Form, Button, Card } from 'react-bootstrap';
3 import { Link, useNavigate, } from 'react-router-dom';
4 import Cookies from 'js-cookies';
5 import Header from '../components/Header';
6
7 const commonFields = [
8   { controlId: 'email', label: 'Email', type: 'email' },
9   { controlId: 'password', label: 'Password', type: 'password' },
10 ];
11 const AdminLogin = () => JSX.Element
12 const AdminLogin = () => {
13   const [formData, setFormData] = useState({
14     email: '',
15     password: '',
16   });
17
18   const navigate = useNavigate();
19   const token = Cookies.getItem('jwtToken');
20   const adminToken = localStorage.getItem('adminJwtToken');
21
22   useEffect(() => {
23     if (token || adminToken) {
24       navigate('/');
25     }
26   }, [token, adminToken]);
27 }
```

Products.js in frontend:

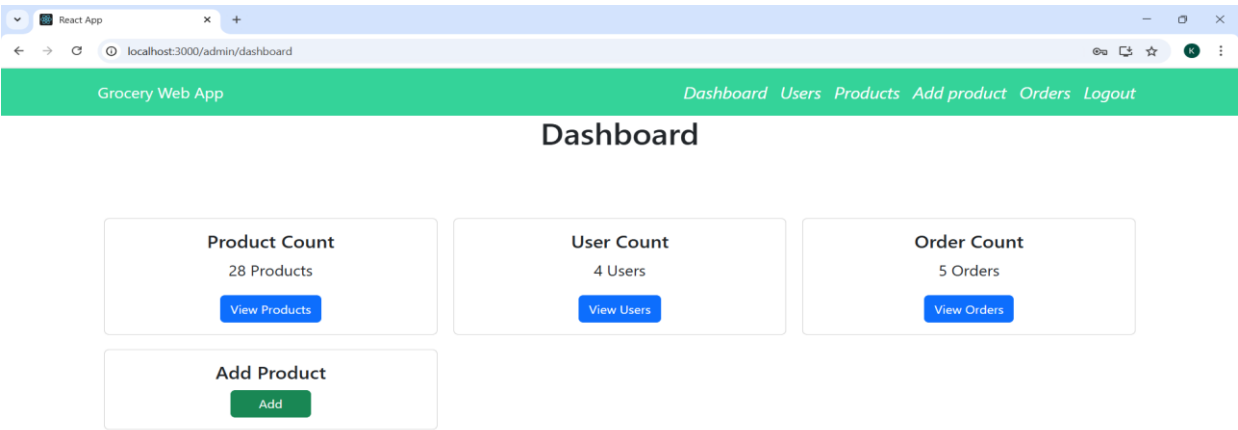


11.Screenshots:

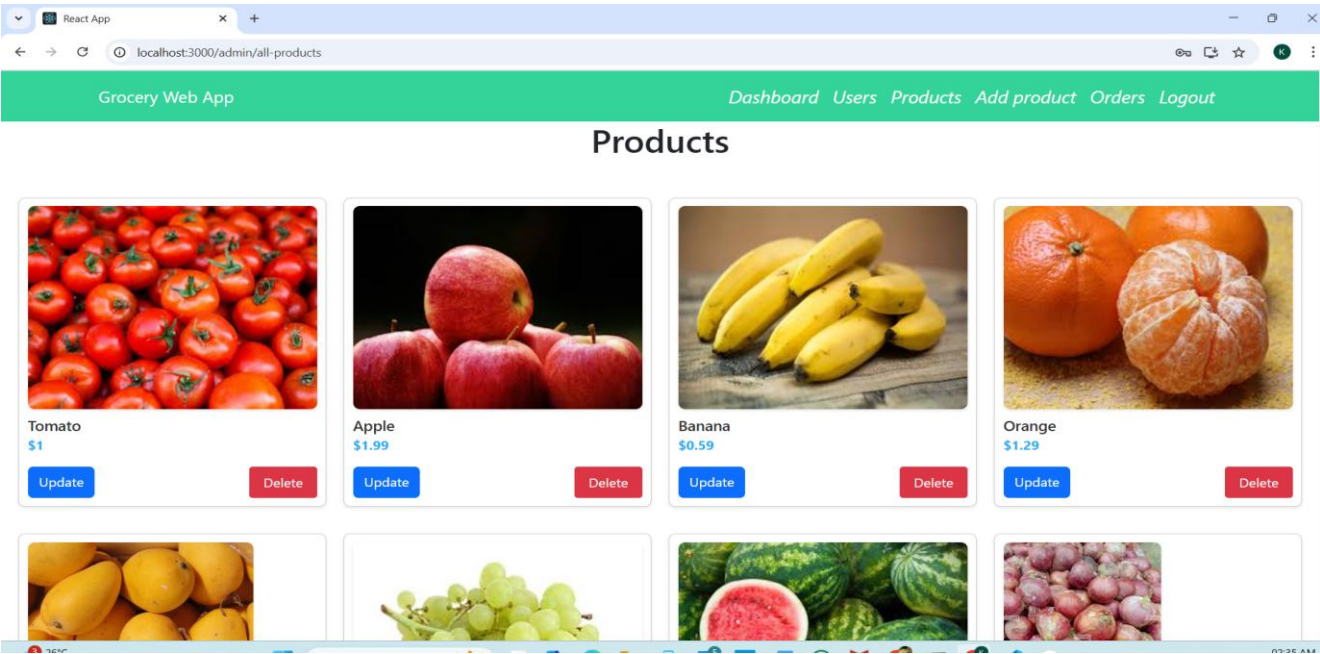
Homepage:



Admin Dashboard:



Poducts in admin page:

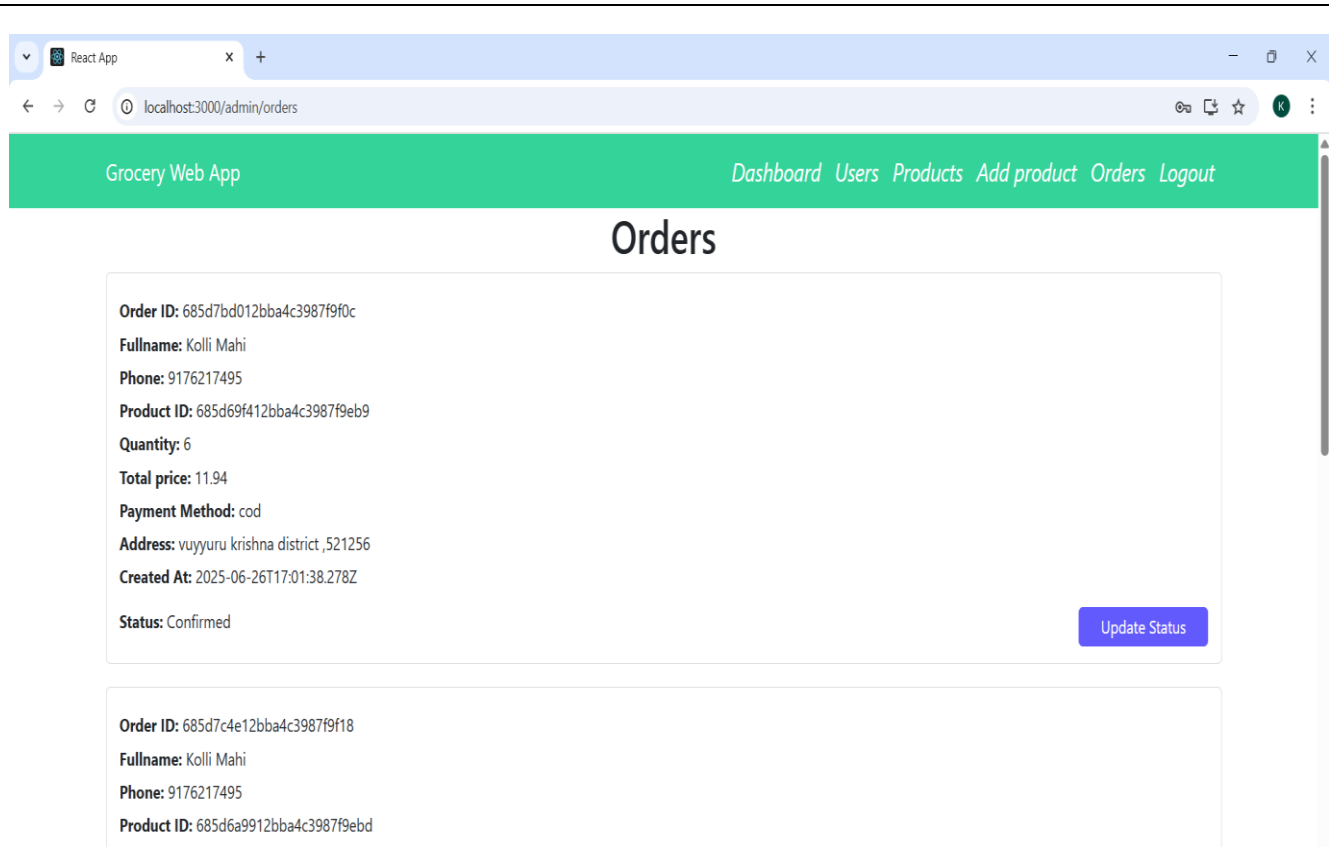


Add product in admin page:

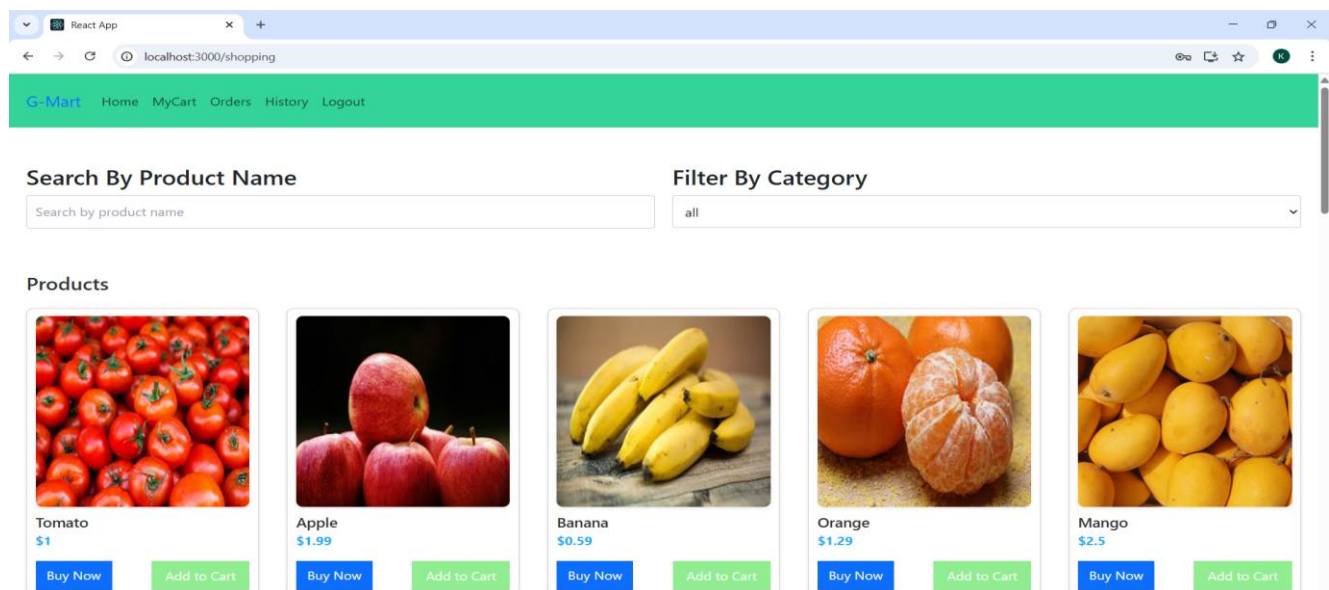
The screenshot shows a web browser window with the address bar displaying 'localhost:3000/admin/add-product'. The page has a green header bar with the text 'Grocery Web App' on the left and a navigation menu on the right containing 'Dashboard', 'Users', 'Products', 'Add product', 'Orders', and 'Logout'. The main content area is titled 'Add Product' and contains a light blue form. The form has the following fields:

| Product Name | Rating | Price | |
|--|--|---|-------------|
| <input type="text" value="Enter product name"/> | <input type="text" value="Enter product rating"/> | <input type="text" value="Enter product price"/> | |
| Image URL | Category | Count in Stock | |
| <input type="text" value="Enter image URL"/> | <input a="" arrow<="" dropdown="" td="" type="text" value="Select Category" with=""/> <td><input type="text" value="Enter count in stock"/></td> | <input type="text" value="Enter count in stock"/> | |
| <th>Description</th> | | | Description |
| <input type="text" value="Enter product description"/> | | | |
| <input type="button" value="Add Product"/> | | | |

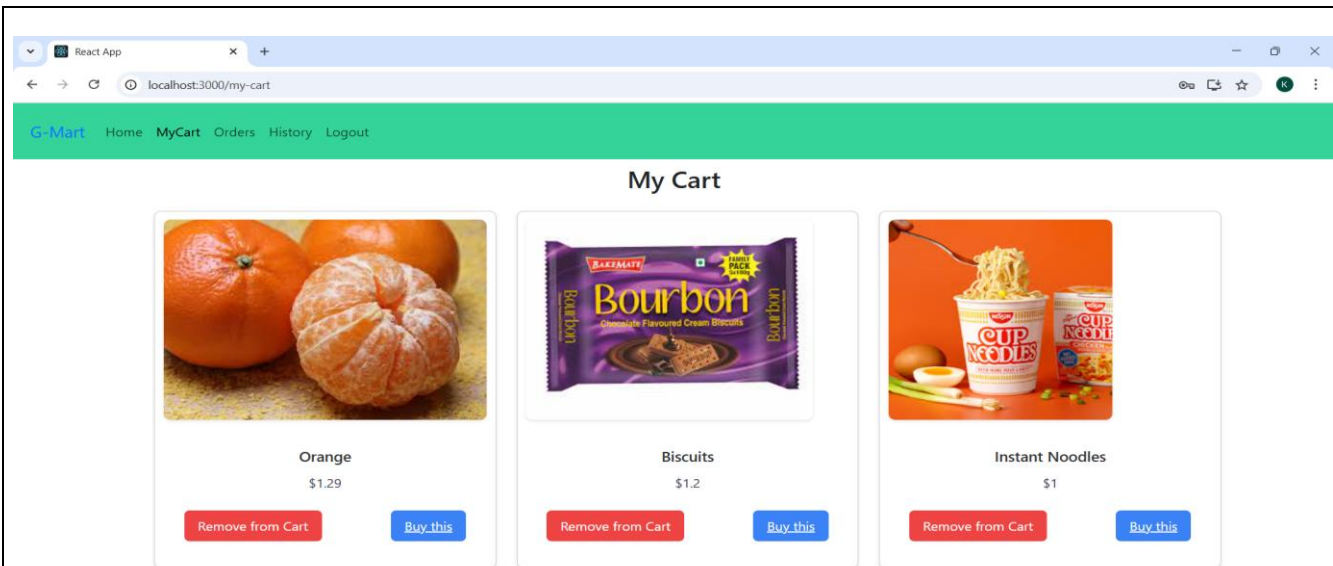
Orders in admin page:



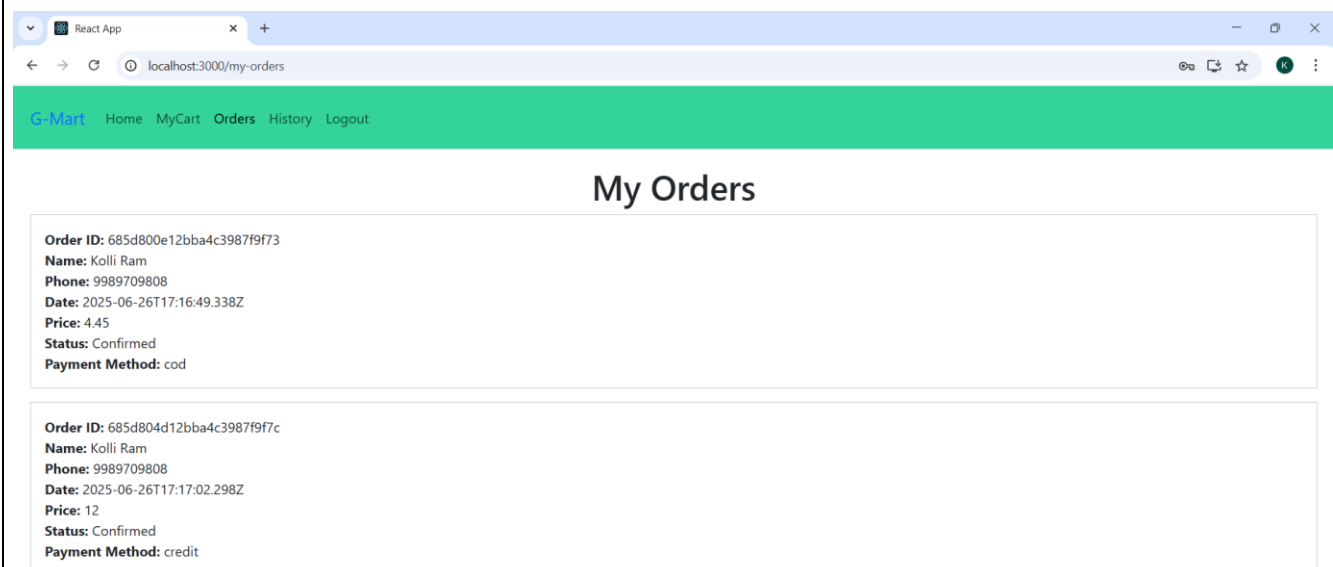
User home page:



User mycart page:



Myorders in user page:



Order details in user page:

React App

localhost:3000/order-details/685d6a9912bba4c3987f9ebd

G-Mart Home MyCart Orders History Logout

Order Details

First Name:
Enter your first name

Last Name:
Enter your last name

Phone:
Enter your phone number

Quantity:
Enter the quantity

Address:
Enter your address

Payment Method:

12.Demo Link:

Google Drive video_demo link for the project:

https://drive.google.com/file/d/1V8MDyKiy4U4Ti0G_zZTxZL7SFuNYIvTa/view?usp=drive_link

13.Future Enhancements:

- Stripe/Razorpay integration for secure payments
- Wishlist and favorites functionality
- Product ratings and reviews
- AI-powered product recommendations
- Email/SMS notifications for order status
- PWA (Progressive Web App) support for offline use
- Analytics dashboard for admin