GROCERY WEB APP
USING MERN STACK SMART INTERNZ (NM)

PROJECT REPORT

Submitted by

BAKIYA BLESSY M	211121205005
KANMANI G	211121205014
LAVANYA J	211121205016
YUVARANI K	211121205030

STUDENT OF

DEPARTMENT OF INFORMATION TECHNOLOGY

MADHA ENGINEERING COLLEGE

KUNDRATHUR :: CHENNAI 600069

NOV 2024

ABSTRACT

This Smart Grocery Shopping App is a web-based Application which will have more additional features for providing the easy and smooth shopping experience to customers The software's and hardware's that we have used to develop this system are easily available and easy to work with. The aim is to add advanced functionalities to the existing shopping system using the prediction module so that the admin can predict the future stock for his market.

Users can buy the groceries as per their requirements and can pay the respective bills with the help of online payment module which will be built-in this system and for the admin, system has the prediction module which will help the admin to predict the future stock for his market based on the previous sales. Basically, the project describes how to manage for good performance and better services for the clients and as well as for the shopkeeper.

Grocery shopping is a regular necessity typically conducted in-store, with available alternatives such as online grocery shopping still not fully established yet. To address challenges experienced by shoppers in a store environment, two supporting modes were derived, implemented in a prototypical smartphone application using a minimal attention user interface and evaluated in an experimental field study. The two supporting modes consist of a novel map representation of shopping lists and a contextual proximity notification system.

The evaluation results show an increase in efficiency in the process of grocery shopping, resulting in reduced shopping time, optimized shopping paths in the store, minimized dependency on external support and preservation of the shopper's cognitive effort. Acceptance of the introduced supporting modes was high which is a crucial indicator for future work**.**

TABLE OF CONTENT

LIST OF FEATURES

1. INTRODUCTION

In today's fast-paced world, grocery store websites have revolutionized the way. My grocery store website aims to bring convenience and variety to user's experiences by offering a seamless platform to explore, order, and enjoy delicious product from their favourite store with a user-friendly interface and robust functionality, this website caters to both food enthusiasts seeking culinary adventures and busy individuals needing quick meal solutions.

The website is designed to provide a hassle-free experience, allowing users to browse an extensive menu of products, filter by preferences, and place orders with ease. A secure login system ensures user data privacy, while advanced search and filtering options make finding the perfect meal straightforward. From appetizers to desserts, users can explore a variety of products from local customers or global chains, ensuring there is something for everyone.

Our platform stands out with features such as real-time order tracking, secure payment gateways, and personalized recommendations based on user preferences. By leveraging responsive design and intuitive navigation, the website is accessible on all devices, ensuring users can order Grocery items anytime, anywhere. Additionally, the integration of reviews and ratings enables users to make informed choices, fostering trust and satisfaction

Our grocery web app, built using the MERN stack (MongoDB, Express.js, React, and Node.js), is designed to transform the way you shop for everyday essentials. With a dynamic and responsive interface powered by React, users can browse an extensive product catalogue, manage their shopping carts, and complete purchases seamlessly.

The backend, driven by Node.js and Express, ensures secure, efficient handling of requests, while MongoDB provides a scalable solution for managing user data and inventory. Features like real-time stock updates, personalized recommendations, and secure payment integration make the app both convenient and user-friendly. Whether you're at home or on the go, our app delivers a fast, reliable, and hassle-free shopping experience tailored to your needs.

2. PROJECT OVERVIEW

Our basic grocery-web app! Our app is designed to provide a seamless online shopping experience for customers, making it convenient for them to explore and purchase a wide range of products. Whether you are a tech enthusiast, a fashionista, or a homemaker looking for everyday essentials, our app has something for everyone.

With user-friendly navigation and intuitive design, our grocery-webapp app allows customers to browse through various categories, view product details, add items to their cart, and securely complete the checkout process. We prioritize user satisfaction and aim to provide a smooth and hassle-free shopping experience.

For sellers and administrators, our app offers robust backend functionalities. Sellers can easily manage their product listings, inventory, and orders, while administrators can efficiently handle customer inquiries, process payments, and monitor overall app performance.

With a focus on security and privacy, our grocery-webapp app ensures that customer data is protected, transactions are secure, and personal information remains confidential. We strive to build trust with our customers and provide a safe platform for online shopping.

We are excited to have you on board and look forward to providing you with a delightful shopping experience. Happy shopping with our grocery-webapp!

3. PURPOSE AND FEATURES

The primary purpose of my Grocery store website is to simplify and enhance the process of ordering grocery item online, servicing to the needs of both users and partners.

It aims to provide a platform where users can explore diverse cuisines, place orders effortlessly, and enjoy their favourite items from the comfort of their homes. By connecting customers with a wide range of supermarkets, the website ensures convenience, accessibility, and an improved users experience.

Additionally, the platform empowers stores owners to reach a larger audience, expand their customer base, and manage orders efficiently through a digital interface.

3.1.PURPOSE:

The application aims to bridge the gap between traditional grocery shopping and the convenience of online platforms. It enables users to save time and effort while offering businesses a robust system to manage inventory, orders, and customer interactions.

The purpose of a grocery web app built using the MERN stack (MongoDB, Express.js, React, and Node.js) is to streamline and modernize the grocery shopping experience by offering users a seamless and user-friendly platform.

This web app aims to provide a comprehensive solution for customers to browse, search, and purchase a wide range of grocery products online. It leverages the scalability and flexibility of MongoDB for efficient data management, Express.js and Node.js for a robust backend, and React for an interactive and responsive user interface.

The app can include features like real-time inventory updates, personalized recommendations, secure payment gateways, and order tracking, making grocery shopping more convenient, efficient, and accessible.

Businesses benefit from improved operational efficiency, customer insights through analytics, and enhanced customer satisfaction.

3.2.KEY FEATURES:

3.2.1. User-Friendly Interface:

- Easy-to-navigate design for browsing menus and placing orders,
- Responsive layout for seamless access on desktops, tablets, and smartphones.

3.2.2. Supermarket Listings and Menu Display:

- Comprehensive listings of stores with detailed menus and pricing.
- Cuisines categorized for quick and easy access.

3.2.3. Secure Login and Payment:

- User authentication for secure access to accounts and order history.
- Multiple payment options, including credit/debit cards, digital wallets, and UPI, ensuring safe and convenient transactions.

3.2.4. Order Tracking and Updates:

- Real-time tracking of orders from shipping to delivery.
- Notifications to keep users informed about order status

3.2.5. Search and Filter Options:

- Advanced search features to find items and supermarkets
- Filters for sorting by ratings, price, distance, or special offers.

3.2.6. Personalized Experience:

- Recommendations based on user preferences and past orders.
- Options to save favourite items and frequent orders for quick access.

3.2.7. Reviews and Ratings:

- Integrated system for users to leave feedback and rate products or supermarket.
- Helps new users make informed choices.

3.2.8. Super market Partner Portal:

- Tools for supermarket owners to manage menus, orders, and customer interactions.
- Analytics to track performance and customer preferences.

3.2.9. Technical Features:

- Use web sockets or socket.io for live notifications and updates.
- Track user behaviour, sales trends, and popular products for business insights.

3.2.10. Additional Features:

- Let us select preferred delivery time.
- Offer points or discounts for repeat customers.

The Grocery Web App not only enhances user convenience but also empowers grocery stores to operate more efficiently in a competitive digital market.

4. TECHNICAL ARCHITECTURE

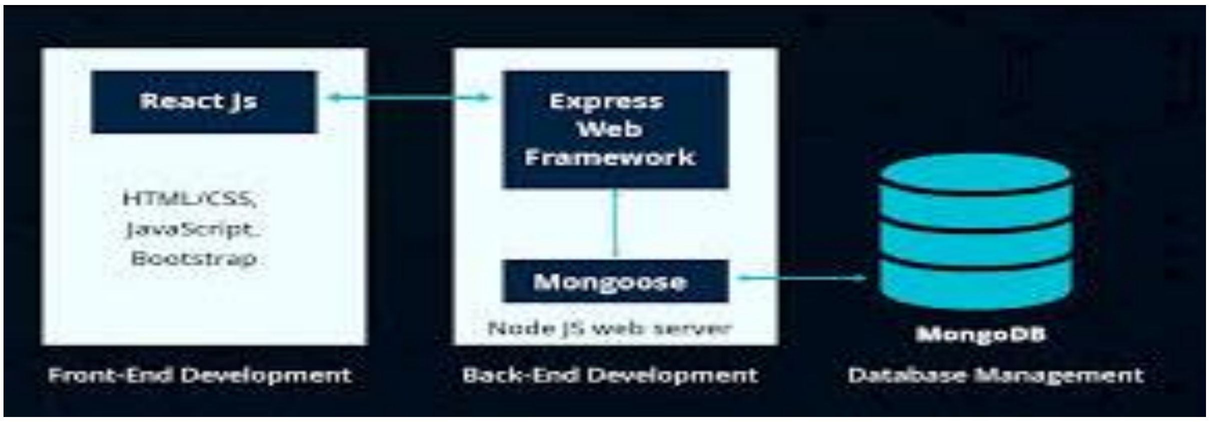


Fig 1.1 Basic Architecture

The technical architecture of a grocery web app built on the MERN stack is designed to ensure scalability, modularity, and efficiency. Below is a breakdown of the architecture and how the components interact.

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.
- The MERN stack's end-to-end JavaScript environment facilitates rapid development, easier debugging, and a unified codebase, making it ideal for projects requiring a modern, responsive, and data-driven approach.
- This architecture supports a modular and maintainable structure, enabling future scalability and adaptability to evolving requirements.

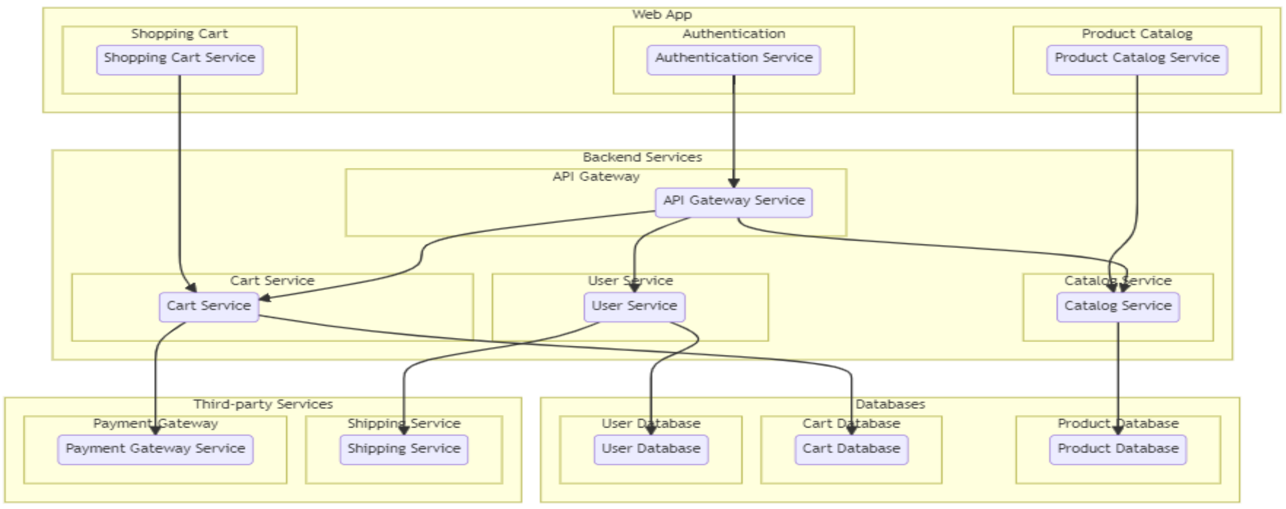


Fig 1.2 Technical Architecture

The technical architecture of a flower and gift delivery app typically involves a client-server model, where the frontend represents the client and the backend serves as the server.

The frontend is responsible for user interface, interaction, and presentation, while the backend handles data storage, business logic, and integration with external services like payment gateways and databases .Communication between the frontend and backend is typically facilitated through APIs, enabling seamless data exchange and functionality.

The technical architecture of a project built using the MERN stack (MongoDB, Express.js, React, and Node.js) provides a robust, scalable, and efficient framework for developing modern web applications.

This architecture is designed to leverage the strengths of each technology in the stack to ensure seamless integration between the front-end and back-end systems. React serves as the front-end library, enabling the creation of dynamic and highly interactive user interfaces.

Node.js, combined with Express.js, forms the server-side backbone, handling API requests, business logic, and secure communication with the database. MongoDB, as a NoSQL database, ensures flexibility and scalability for storing and managing data efficiently.

5. ER DIAGRAM

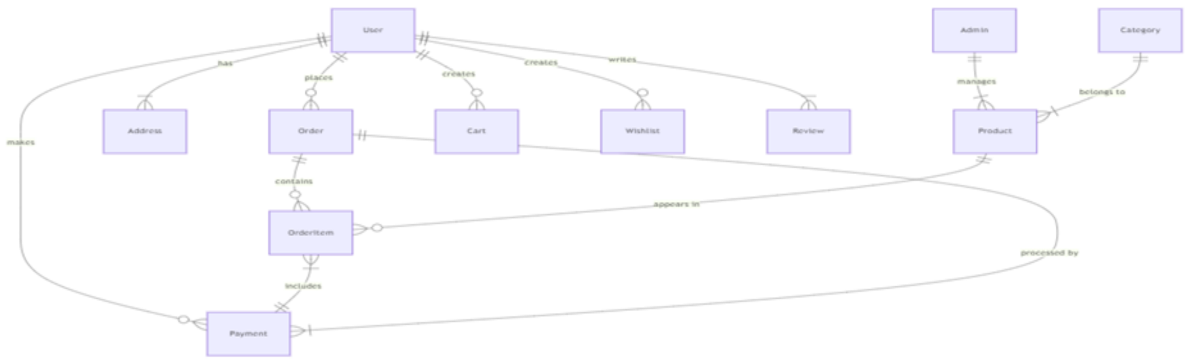


Fig 1.3 ER Diagram

The Entity-Relationship (ER) diagram for a flower and gift delivery app visually represents the relationships between different entities involved in the system, such as users, products, orders, and reviews.

It illustrates how these entities are related to each other and helps in understanding the overall database structure and data flow within the application.

5.1. Frontend Architecture (React.js):

The frontend is designed using React.js, a JavaScript library that allows for building reusable and responsive user interfaces. Key aspects include:

- **Component-Based Structure:**
  UI is divided into reusable components like Product Card, Navbar, and Cart. Each component handles specific tasks, improving maintainability.

- **State Management:**

  Context API or Redux is used for managing application state, such as user authentication and cart data.

- **Routing:**

  React Router manages navigation between pages, such as Home, Product Details, and Checkout, without full-page reloads.

- **Styling:**

  Styling is implemented using CSS, SCSS, or libraries like Material-UI or Bootstrap for responsiveness.

5.2. Backend Architecture (Node.js + Express.js) :

The backend handles business logic, API endpoints, and communication with the database. Key aspects include:

- **RESTFUL APIs:**

  APIs are built using Express.js to handle HTTP requests and responses.

  - **Example:**
    - GET /API/products: Fetch all products.
    - POST /API/orders: Place an order.

- **Middleware:**

  Custom middleware is implemented for tasks like request validation, error handling, and authentication using JWT.

- **Scalability:**

  Node.js's asynchronous and event-driven nature ensures the backend can handle multiple concurrent requests.

5.3. Database Architecture (MongoDB):

MongoDB, a NoSQL database, is used for flexible and scalable data storage. Key aspects include:

- **Collections:** The database consists of the following key collections.

- **Users:** Stores user credentials, profile data, and order history.

- **Products:** Stores product details like name, price, category, and availability.

- **Orders:** Tracks orders, including user ID, product IDs, and order status.

- **Cart:** Temporary storage for a user's cart items.

- **Schema Design:** Documents are structured to allow efficient querying and updates. For instance, orders store product references instead of embedding entire product details to minimize data duplication.

6.PREREQUISITES

To develop a full-stack Grocery Web 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 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.

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

https://www.section.io/engineering-education/nodejs- mongoosejsmongodb

- **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

7. USER & ADMIN FLOW

The user flow and admin flow in a grocery web app built using the MERN stack outline the journey and interactions of customers and administrators. Below is a detailed description of both flows:

7.1. User Flow:

- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform.
- Users can add the products they wish to their carts and orders.
- They can then proceed by entering address and payment details.
- After ordering, they can check them in the profile section

7.2. Store Flow:

- Supermarket start by authenticating with their credentials.
- They need to get approval from the admin to start listing the products.
- They can add/edit the grocery items.

7.3. Admin Flow:

- Admins start by logging in with their credentials.
- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, products, orders, etc.

8. FOLDER STRUCTURE

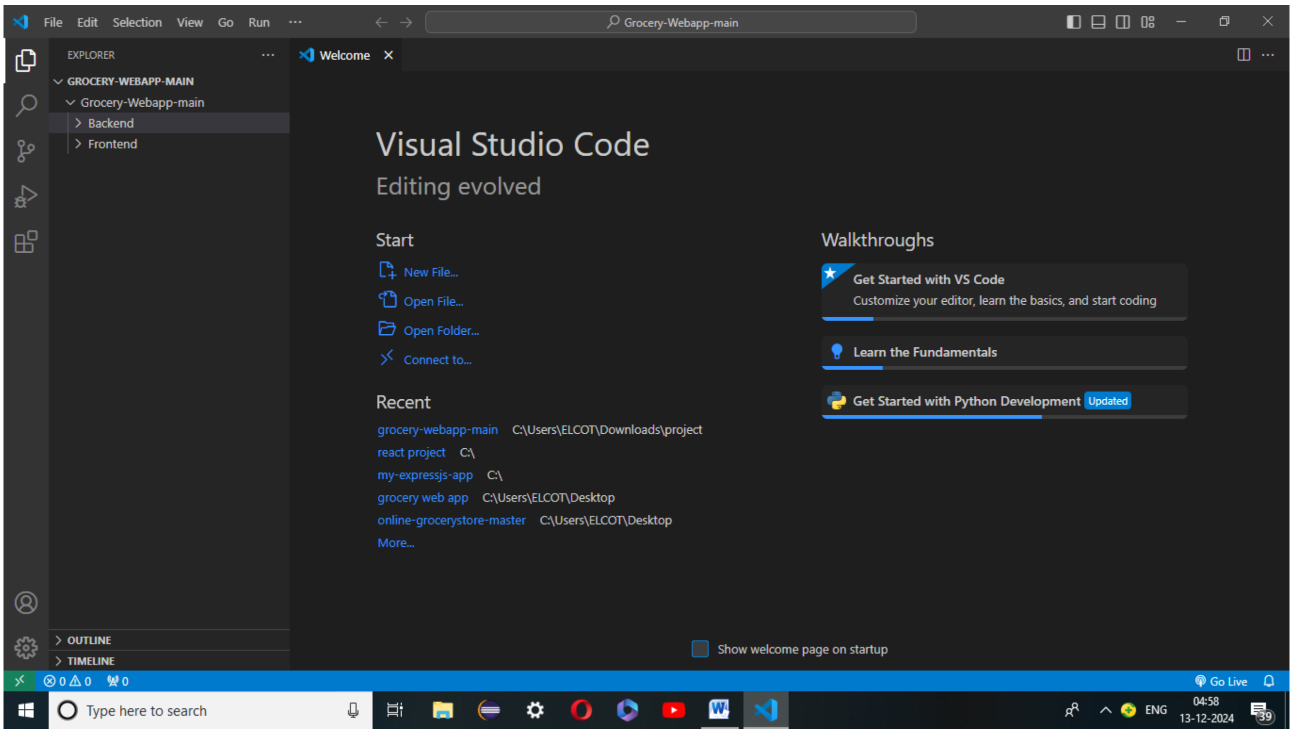
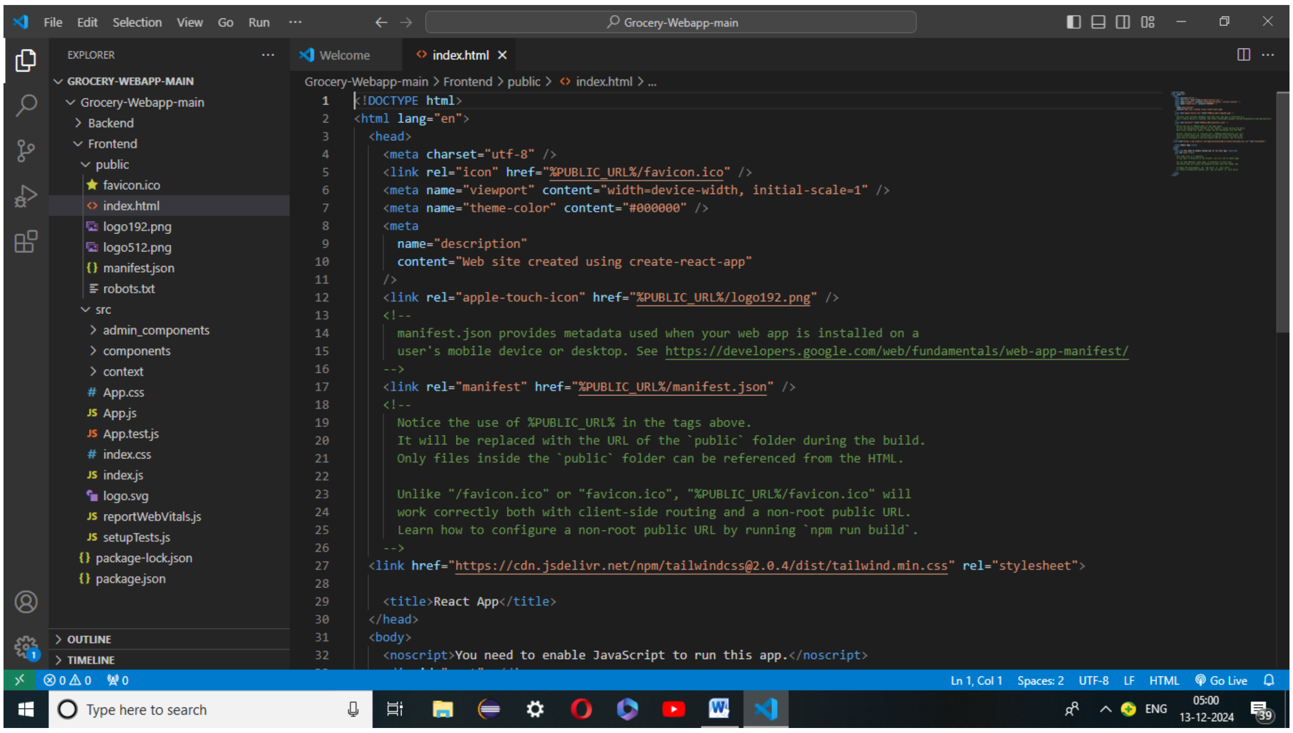


Fig 1.4 Project Structure



```
GROCERY-WEBAPP-MAIN
  Grocery-Webapp-main
    Backend
    Frontend
      public
        favicon.ico
        index.html
        logo192.png
        logo512.png
        manifest.json
        robots.txt
      src
        admin_components
        components
        context
        App.css
        App.js
        App.test.js
        index.css
        index.js
        logo.svg
        reportWebVitals.js
        setupTests.js
      package-lock.json
      package.json
```

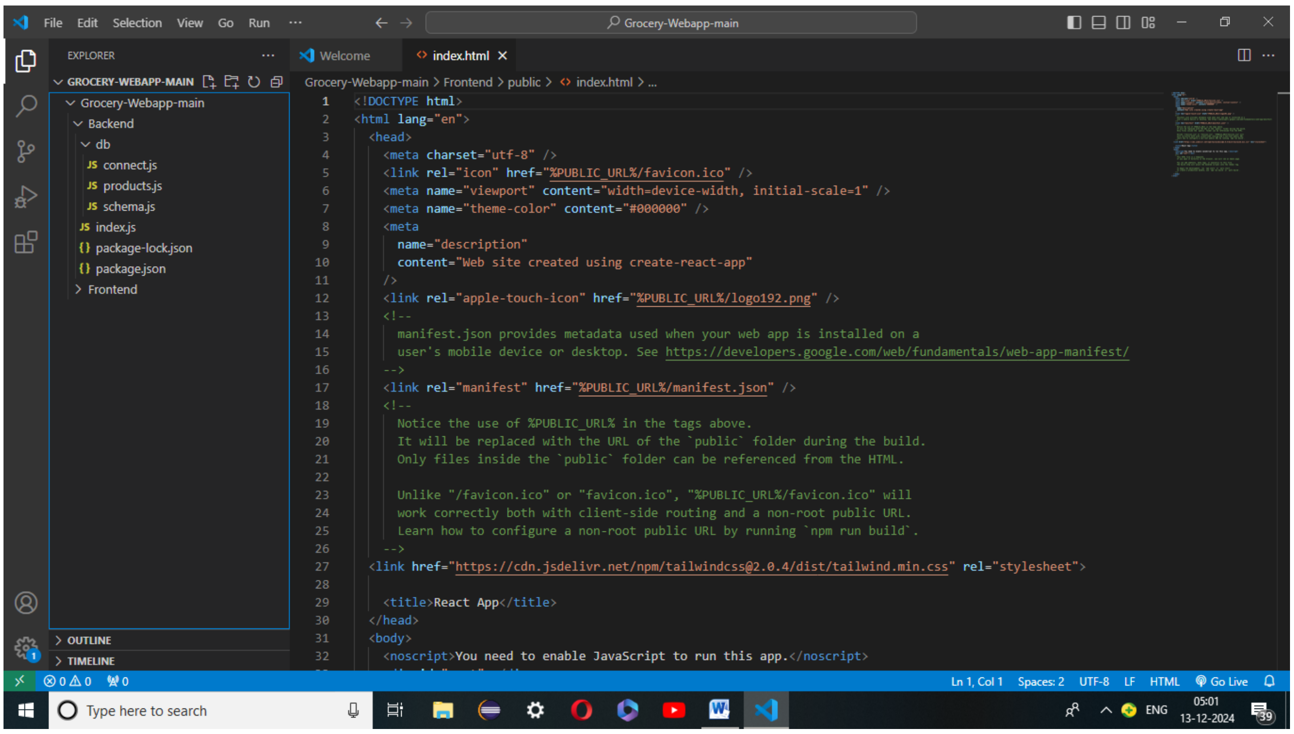
Grocery-Webapp-main > Frontend > public > index.html > ...

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.0.4/dist/tailwind.min.css" rel="stylesheet">

    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
```

Fig 1.5 Project Structure



```
GROCERY-WEBAPP-MAIN
  Grocery-Webapp-main
    Backend
      db
        connect.js
        products.js
        schema.js
      index.js
      package-lock.json
      package.json
    Frontend
```

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.0.4/dist/tailwind.min.css" rel="stylesheet">

    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
```

Fig 1.6 Project Structure

This structure assumes an Angular app and follows a modular approach. Here's a brief explanation of the main directories and files:

- src/app/components: Contains components related to the customer app, such as register, login, home, products, my-cart, my-orders, place order, history, feedback, product-details, and more.

- src/app/modules: Contains modules for different sections of the app. In this case, the admin module is included with its own set of components like add-category, add-product, dashboard, feedback, home, orders, payment, update-product, users, and more.

- src/app/app-routing.module.ts: Defines the routing configuration for the app, specifying which components should be loaded for each route.

- src/app/app.component.ts, src/app/app.component.html, `src.

9. PROGRAM

9.1 INDEX HTML:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
  <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.0.4/dist/tailwind.min.css"
rel="stylesheet">

    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

9.2 DASHBOARD:

```
import React, { useEffect, useState } from 'react';
import Card from 'react-bootstrap/Card';
import Button from 'react-bootstrap/Button';
import './style.css'; // Import your custom CSS for styling
import axios from 'axios';
import { Link } from 'react-router-dom';
import AdminNavabar from '../AdminNavbar'

const Dashboard = () => {
  const [data, setData] = useState({
    products: 0,
    users: 0,
    orders: 0,
  });

  useEffect(() => {
    const fetchData = async () => {
      try {
        const [usersResponse, productsResponse, ordersResponse] = await
Promise.all([
          axios.get('http://localhost:5100/users'),
          axios.get('http://localhost:5100/products'),
          axios.get('http://localhost:5100/orders'),
        ]);

        setData({
          users: usersResponse.data.length,
          products: productsResponse.data.length,
          orders: ordersResponse.data.length,
        });
      } catch (error) {
        console.error('Error fetching data:', error);
      }
    };

    fetchData();
  }, []);

  return (
    <div>
      <AdminNavabar/>
      <h1 className="text-center">Dashboard</h1>
      <div className="dashboard container">
      <div className="card-container">
        <Card>
          <Card.Body>
            <Card.Title>Product Count</Card.Title>
            <Card.Text>{data.products} Products</Card.Text>
            <Link style={{textDecoration:'none',color:'white'}} to={'/admin/all-
products'}><Button variant="primary">View Products </Button></Link>
          </Card.Body>
        </Card>

        <Card>
          <Card.Body>
            <Card.Title>User Count</Card.Title>
            <Card.Text>{data.users} Users</Card.Text>
            <Link style={{textDecoration:'none',color:'white'}}
to={'/admin/users'}><Button variant="primary">View Users </Button></Link>
          </Card.Body>
        </Card>

        <Card>
          <Card.Body>
            <Card.Title>Order Count</Card.Title>
            <Card.Text>{data.orders} Orders</Card.Text>
            <Link style={{textDecoration:'none',color:'white'}}
to="/admin/orders"><Button variant="primary">View Orders</Button></Link>
          </Card.Body>
        </Card>

        <Card>
          <Card.Body>
            <Card.Title>Add Product</Card.Title>
            <Link style={{textDecoration:'none',color:'white'}} to='/admin/add-
product'><Button style={{width:'100px'}} variant="success">Add </Button></Link>
          </Card.Body>
        </Card>

        {/* <Card>
          <Card.Body>
            <Card.Title>Add Category</Card.Title>
            <Link style={{textDecoration:'none',color:'white',}} to='/admin/add-
category'><Button style={{width:'100px'}} variant="success">Add </Button></Link>
          </Card.Body>
        </Card> */}
      </div>
    </div>
    </div>
  );
};

export default Dashboard;
```

9.3.ORDERS :

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import styled from 'styled-components';
import LoaderSpinner from '../../components/LoaderSpinner';
import AdminNavbar from '../AdminNavbar';

// Styled components
const Container = styled.div`
  text-align: start;
`;

const Heading = styled.h1`
  color: rgb(62, 62, 62);
  font-size: 38px;
  font-weight: bold;
  margin-bottom: 20px;
`;

const OrderCard = styled.div`
  background-color: #fff;
  border: 1px solid #ddd;
  border-radius: 5px;
  padding: 15px;
  margin-bottom: 20px;
  transition: transform 0.3s ease;
`;

const OrderDetail = styled.p`
  margin: 5px 0;
`;

const Button = styled.button`
  background-color: rgb(98, 90, 252);
  color: #fff;
  width: 150px;
  margin-top: 10px;
  padding: 5px 10px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s ease;

  &:hover {
    background-color: rgb(68, 60, 196);
  }

  &:disabled {
    background-color: #ccc;
    cursor: not-allowed;
  }
`;

const Orders = () => {
  const [isLoading, setIsLoading] = useState(true);
  const [data, setData] = useState([]);
  const [isUpdate, setIsUpdate] = useState(false);
  const [selectedOrderId, setSelectedOrderId] = useState('');
  const [statusForm, setStatusForm] = useState({
    status: 'Confirmed', // Default status
  });

  useEffect(() => {
    const timer = setTimeout(() => {
      setIsLoading(false);
    }, 2000); // Simulating 2 seconds loading time
    return () => clearTimeout(timer);
  }, []);

  useEffect(() => {
    getData();
  }, []);

  const getData = () => {
    axios.get('http://localhost:5100/orders')
      .then((response) => {
        setData(response.data);
      })
      .catch((error) => {
        console.error('Error fetching orders:', error);
      });
  };

  const onSubmit = (formData) => {
    axios.put(`http://localhost:5100/orders/${selectedOrderId}`, formData)
      .then(() => {
        setIsUpdate(false);
        getData();
      })
      .catch((e) => {
        console.log(e);
      });
  };

  const onChangeStatus = (orderId) => {
    setIsUpdate(true);
    setSelectedOrderId(orderId);
  };

  return (
    <div>
      <AdminNavbar />
      {isLoading ? (
        <div style={{ height: '100vh', display: 'flex', alignItems: 'center',
justifyContent: 'center' }}>
          <LoaderSpinner />
        </div>
      ) : (
        <Container className="container">
          <h1 className='text-center'>Orders</h1>
          {data.length === 0 ? (
            <div>
              <p>No orders in your shop!</p>
            </div>
          ) : (
            <div>
              {isUpdate ? (
                <div>
                  <form onSubmit={(e) => { e.preventDefault();
onSubmit(statusForm); }}>
                    <div className="form-group">
                      <label htmlFor="statusSelect">Select Status</label>
                      <select className="form-control" id="statusSelect"
value={statusForm.status} onChange={(e) => setStatusForm({ ...statusForm, status:
e.target.value })}>
                        <option value="Confirmed">Confirmed</option>
                        <option value="Shipped">Shipped</option>
                        <option value="Delivered">Delivered</option>
                      </select>
                    </div>
                    <Button type="submit">Save Changes</Button>
                  </form>
                </div>
              ) : null}
              {!isUpdate && data.map((item) => (
                <OrderCard key={item._id}>
                  <OrderDetail><strong>Order ID:</strong> {item._id}</OrderDetail>
                  <OrderDetail><strong>Fullname:</strong> {item.firstname}
{item.lastname}</OrderDetail>
                  <OrderDetail><strong>Phone:</strong> {item.phone}</OrderDetail>
                  <OrderDetail><strong>Product ID:</strong>
{item.productId}</OrderDetail>
                  <OrderDetail><strong>Quantity:</strong>
{item.quantity}</OrderDetail>
                  <OrderDetail><strong>Total price:</strong>
{item.price}</OrderDetail>
                  <OrderDetail><strong>Payment Method:</strong>
{item.paymentMethod}</OrderDetail>
                  <OrderDetail><strong>Address:</strong>
{item.address}</OrderDetail>
                  <OrderDetail><strong>Created At:</strong>
{item.createdAt}</OrderDetail>
```

```
                  <div style={{ display: 'flex', justifyContent: 'space-between',
alignItems: 'center' }}>
                    <OrderDetail><strong>Status:</strong>
{item.status}</OrderDetail>
                    {item.status !== 'Canceled' && item.status !== 'Delivered' &&
<Button onClick={() => onChangeStatus(item._id)} disabled={item.status ===
'Delivered'}>Update Status</Button>}
                    {item.status === 'Canceled' && <Button disabled={item.status
=== 'Canceled'}>Customer Canceled</Button>}
                    {item.status === 'Delivered' && <Button disabled={item.status
=== 'Delivered'}>Delivered</Button>}
                  </div>
                </OrderCard>
              ))}
            </div>
          )}
        </Container>
      )}
    </div>
  );
};

export default Orders;
```

9.4. PRODUCT ITEM :

```
import React, { useState } from "react";
import { Link } from "react-router-dom";
import axios from "axios";
import Cookies from 'js-cookies'
import {
  ProductContainer,
  ProductName,
  ProductDescription,
  ProductPrice,
  ProductImage,
  Button,
  ButtonContainer,
} from "./styledComponents";


const AdminProductItem = ({ id, name, description, price, img
,handleDeleteProduct}) => {

  const handleDelete = async () => {
    handleDeleteProduct(id)
  };
  return (
   <div>
     <ProductContainer>
      <ProductImage src={img} alt={name} />
      <ProductName>{name}</ProductName>
      <ProductPrice>${price}</ProductPrice>
      <ButtonContainer>
        <Link to={`/admin/product-update/${id}`}  className='btn btn-
primary'>Update</Link>
        <Button onClick={handleDelete} className='btn btn-danger' >Delete</Button>
      </ButtonContainer>
      {/* <div>
        <label>Quantity: </label>
        <input
          type="number"
          value={quantity}
          onChange={(e) => setQuantity(e.target.value)}
        />
      </div> */}
    </ProductContainer>
   </div>
  );
};

export default AdminProductItem;
```

9.5 PRODUCTS :

```
import React, { useState, useEffect } from 'react';
import styled from 'styled-components';
import ProductItem from '../ProductItem';
import Cookies from 'js-cookies';
import axios from 'axios';
import AdminNavabar from '../AdminNavbar';

const ProductsContainer = styled.div`
  margin-top: 4vh;
  padding: 20px;
  text-align: start;
`;

const Heading = styled.h2`
  font-size: 24px;
  color: #333;
  margin-bottom: 20px;
  margin-top: 40px;
  text-align: center;
`;

const StyledList = styled.ul`
  list-style: none;
  display: grid;
  grid-template-columns: repeat(4, 1fr); // Display 4 items in each row
  gap: 20px;
  padding: 0;
`;


const AdminProducts = () => {
  const api = 'http://localhost:5100/products';
  const [products, setProducts] = useState([]);

  useEffect(() => {
    getData();
  }, []);

  const getData = () => {
    fetch(api)
      .then(response => response.json())
      .then(data => setProducts(data))
      .catch(error => console.error('Error fetching products:', error));
  };

  const handleDeleteProduct = async (id) => {
    const userId = Cookies.getItem("userId");
    try {
      await axios.delete(`http://localhost:5100/products/${id}`);
      getData();
    } catch (error) {
      console.error("Error deleting product:", error);
    }
  };

  return (
    <div>
      <AdminNavabar />
      <h1 className='text-center'>Products</h1>
      <ProductsContainer>
        <StyledList>
          {products.map(product => (
            <ProductItem
              key={product._id}
              id={product._id}
              img={product.image}
              name={product.productname}
              description={product.description}
              price={product.price}
              handleDeleteProduct={handleDeleteProduct}
            />
          ))}
        </StyledList>
      </ProductsContainer>
    </div>
  );
};

export default AdminProducts;
```

9.6. UPDATE:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate, useParams } from 'react-router-dom'; // Import useParams to
get the product ID from the URL
import styled from 'styled-components';
import AdminNavabar from '../AdminNavbar'


// Styled components (you can adjust styles as needed)
const Container = styled.div`
  max-width: 700px;
  margin: 5vh auto;
  text-align: start;
  background-color:skyblue;
`;

const Heading = styled.h2`
  font-size: 24px;
  font-weight: bold;
  color: rgb(62, 62, 62);
  margin-bottom: 20px;
`;

const Form = styled.form`
  display: flex;
  flex-direction: column;
`;

const FormGroup = styled.div`
  margin-bottom: 20px;
`;

const Label = styled.label`
  font-weight: bold;
  margin-bottom: 8px;
`;
```

```
const Input = styled.input`
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 16px;
  width: 100%;
`;

const Textarea = styled.textarea`
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 16px;
  width: 100%;
  min-height: 100px;
`;

const Button = styled.button`
  background-color: blue;
  color: white;
  padding: 10px 20px;
  border: none;
  cursor: pointer;
  transition: background-color 0.3s ease;

  &:hover {
    background-color: orangered;
  }
`;

const UpdateProduct = () => {
  const { id } = useParams(); // Get the product ID from the URL

  const [formData, setFormData] = useState({
    productname: '',
    description: '',
    price: '',
    image: '',
    category: '',
    countInStock: '',
    rating: '',
  });

  const navigate = useNavigate()

  useEffect(() => {
    // Fetch the existing product data
    axios.get(`http://localhost:5100/products/${id}`)
      .then((response) => {
        // Populate the form with the existing product data
        setFormData(response.data);
      })
      .catch((error) => {
        console.error('Error fetching product data:', error);
      });
  }, [id]);

  const { productname, description, price, image, category, countInStock, rating }
= formData;

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      const response = await axios.put(`http://localhost:5100/products/${id}`,
formData);

      console.log('Product updated:', response.data);
      navigate('/admin/all-products')

      // Handle any other actions upon successful product update

    } catch (error) {
      console.error('Error updating product:', error);
      // Handle errors here, e.g., show an error message to the user
    }
  };

  return (
   <div>
    <AdminNavabar/>
    <h1 className='text-center'>Update Product</h1>
     <Container>
      <Form onSubmit={handleSubmit} className='shadow p-3'>
        {/* Render form fields with existing data */}
        <FormGroup>
          <Label htmlFor="productname">Product Name</Label>
          <Input
            type="text"
            name="productname"
            value={productname}
            onChange={handleChange}
            placeholder="Enter product name"
          />
        </FormGroup>
        <FormGroup>
          <Label htmlFor="rating">Rating</Label>
          <Input
            type="number"
            name="rating"
            value={rating}
            onChange={handleChange}
            placeholder="Enter product rating"
          />
        </FormGroup>
        <FormGroup>
          <Label htmlFor="price">Price</Label>
          <Input
            type="number"
            name="price"
            value={price}
            onChange={handleChange}
            placeholder="Enter product price"
          />
        </FormGroup>
        <FormGroup>
          <Label htmlFor="image">Image URL</Label>
          <Input
            type="text"
            name="image"
            value={image}
            onChange={handleChange}
            placeholder="Enter image URL"
          />
        </FormGroup>
        <FormGroup>
          <Label htmlFor="category">Category</Label>
          <Input
            type="text"
            name="category"
            value={category}
            onChange={handleChange}
            placeholder="Enter category"
          />
        </FormGroup>
        <FormGroup>
          <Label htmlFor="countInStock">Count in Stock</Label>
          <Input
            type="number"
            name="countInStock"
            value={countInStock}
            onChange={handleChange}
            placeholder="Enter count in stock"
          />
        </FormGroup>
        <FormGroup>
          <Label htmlFor="description">Description</Label>
          <Textarea
            name="description"
            value={description}
            onChange={handleChange}
```

```
            placeholder="Enter product description"
          />
        </FormGroup>
        <Button type="submit">Update Product</Button>
      </Form>
    </Container>
   </div>
  );
};

export default UpdateProduct;
```

9.7.USER :

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Button, Table,Card } from 'react-bootstrap';
import { FaTrash,FaEdit } from 'react-icons/fa';
import { Link } from 'react-router-dom';
import AdminNavabar from '../AdminNavbar'


const Users = () => {
  const [userbookings, setUserbookings] = useState([]);
  const [users, setUsers] = useState([]);

  const [showDetails, setShowDetails] = useState(false);

  const toggleDetails = () => {
    setShowDetails(!showDetails);
  };
  const [showDetail, setShowDetail] = useState(false);

  const toggleDetail = () => {
    setShowDetail(!showDetail);
  };

   useEffect(() => {
    axios.get(`http://localhost:5100/users`)
      .then((response) => {
        setUsers(response.data);
        // setLoading(false);
      })
      .catch((error) => {
        // setError('Failed to fetch projects.');
        // setLoading(false);
      });
}, []);

const deleteData = (taskId) => {
    axios.delete(`http://localhost:5100/userdelete/${taskId}`);
    window.location.assign('/admin/users');
    alert('User is deleted');
  };
  const deleteorder = (taskId) => {
    axios.delete(`http://localhost:5100/userbookingdelete/${taskId}`);
    window.location.assign('/admin/users');
    alert('deleted');
  };


  const fetchUserBikeData = (userId) => {

    axios.get(`http://localhost:5100/getbookings/${userId}`)

    .then((response) => {
      setUserbookings(response.data);
      toggleDetails(); // Show Plan Details when data is fetched
    })
    .catch((error) => {
      console.error('Error fetching user bike data:', error);
    });
  };
  const calculateStatus = (date) => {
    const currentDate = new Date();
    const formattedDeliveryDate = new Date(date);

    if (formattedDeliveryDate >= currentDate) {
      return "Upcomming";
    } else {
      return "completed";
    }
  };

  return (
    <div>
      <AdminNavabar/>
    <br />
    <h1 className='text-center'>Users</h1> <br />
    <div style={{display:"flex",justifyContent:"center"}}>
    <Table striped bordered hover variant="dark" style={{width:"70%"}}>
      <thead>
        <tr>
          <th>sl/no</th>
          <th>UserId</th>
          <th>User name</th>
          <th>Email</th>
          <th>Operation</th>
        </tr>
      </thead>
      <tbody>
        {users.map((item, index) => (
          <tr key={item._id}>
            <td>{index + 1}</td>
            <td>{item._id}</td>
            <td>{item.username}</td>
            <td>{item.email}</td>
            <td>
              {/* <button style={{ border: 'none', background: 'none' }}>
                <Link to={`/useredit/${item._id}`} style={{ color: 'blue',
textDecoration: 'none' }}>
                  <FaEdit />
                </Link>
              </button> */}
              <button onClick={() => deleteData(item._id)} style={{ border: 'none',
color: 'red', background: 'none' }}>
                <FaTrash />
              </button>{' '}
              <Button onClick={() => fetchUserBikeData(item._id)} style={{
marginBottom: '12px' }}>
                view
              </Button>
              <div style={{ display: 'flex' }}>
                {showDetails && (
                  <div className="fixed top-0 left-0 w-screen h-screen flex items-
center justify-center z-50" >
                      <div className="bg-gray-900 bg-opacity-50 absolute inset-
0"></div>
                    <div className="bg-white p-4 rounded-lg z-10 relative" style={{
maxHeight: "80vh", overflowY: "scroll" }}>
                      {/* Rest of your content */}
                      <p className="text-sm text-gray-600">
                        <div className="container mx-auto mt-8"
style={{width:"1350px"}}>
                          <h1 className='text-center text-blue-300'>User
Bookings</h1>
                          {userbookings.map((item) => {
                             const status = calculateStatus(item.date);
                            return (
                                <Card
                                key={item._id}
                                style={{
                                  width: '90%',
                                  marginLeft: '65px',
                                  backgroundColor: '#fff',
                                  boxShadow: '0 2px 4px rgba(0, 0, 0, 0.1)',
                                  borderRadius: '8px',
                                  paddingTop: '15px',
                                  marginBottom: '35px',
                                }}
                              >
                                <div style={{ display: 'flex', justifyContent:
'space-around' }}>
                                  <div >
                                  <img
src={`http://localhost:7000/organizer/${item?.templeImage}`} style={{ height:
"80px",width:"120px" }} />
                                  </div>
                                  <div>
                                    <p>Temple Name:</p>
                                    <p>{item.templeName}</p>
                                  </div>
                                  <div>
                                    <p>Darshan Name:</p>
                                    <p>{item.darshanName}</p>
                                  </div>
                                  <div>
                                    <p>Bookingid:</p>
                                    <p>{item._id.slice(0,10)}</p>
                                  </div>
                                  <div>
                                    <p>Organizer</p>
                                    <p>{item.organizerName}</p>
                                  </div>
                                  <div>
                                    <p>BookingDate</p>
                                    <p>{item.BookingDate}</p>
                                  </div>
                                  <div>
                                    <p>Timings</p>
                                    <p>{item.open}-{item.close}</p>
                                  </div>
                                  <div>
                                    <p>Quantity</p>
                                    <p>{item.quantity}</p>
                                  </div>
                                  <div>
                                    <p>Price</p>
                                    <p>{item.totalamount}</p>
                                  </div>
                                  <div>
                                    <p>Status</p>
                                    <p>{status}</p>
                                  </div>
                                  <button onClick={() => deleteorder(item._id)}
style={{ border: 'none', color: 'red', background: 'none' }}>
                <FaTrash />
              </button>
                                </div>
                              </Card>
                            );
                          })}
```

```
                          </div>
                        </p>
                        <Button onClick={toggleDetails} className="mt-4">
                          Close
                        </Button>
                      </div>
                    </div>
                  )}
                </div>
              </td>
            </tr>
          ))}
        </tbody>
      </Table>
      </div>
    </div>

  )
}

export default Users
```

9.8. ADMIN LOGIN :

```
import React, { useEffect, useState } from 'react';
import { Container, Form, Button, Card } from 'react-bootstrap';
import { Link, useNavigate, } from 'react-router-dom';
import Cookies from 'js-cookies'
import Header from '../components/Header';

const commonFields = [
    { controlId: 'email', label: 'Email', type: 'email' },
    { controlId: 'password', label: 'Password', type: 'password' },
];

const AdminLogin = () => {
    const [formData, setFormData] = useState({
        email: '',
        password: '',
    });

    const navigate = useNavigate();
    const token = Cookies.getItem('jwtToken');
    const adminToken = localStorage.getItem('adminJwtToken');

    useEffect(() => {
        console.log(adminToken)
        if (token) {
            navigate('/'); // Redirect to home if token exists
        } else if (adminToken) {
            navigate('/admin/all-products'); // Redirect to admin if an admin token
exists
        }
    }, [navigate]);

    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            const response = await fetch('http://localhost:5100/login', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(formData),
            });

            if (response.ok) {
                const data = await response.json();
                if (data.token) {
                    Cookies.setItem('jwtToken', data.token, { expires: 30 });
                    Cookies.setItem('userId', data.user._id);
                    Cookies.setItem('userName', data.user.firstname);
                    navigate('/');
                } else if (data.jwtToken) {
                    localStorage.setItem('adminJwtToken', data.jwtToken, { expires:
30 });

                    Cookies.setItem('userName', data.user.firstname);
                    navigate('/admin/dashboard');
                }
            } else {
                alert("Email or Password didn't match");
            }
        } catch (error) {
            alert('Error during login:', error);
        }
    };

    const handleInputChange = (e) => {
        const { name, value } = e.target;
        setFormData((prevData) => ({ ...prevData, [name]: value }));
    };

    const handlelogin=()=>{
        alert('Login Successfull')
        navigate('/admin/dashboard')
    }

    return (
       <div>
```

```
        <Header/>
         <Container className="d-flex justify-content-center align-items-center"
style={{ minHeight: '100vh', paddingTop: '10vh' }}>
             <Card className="shadow p-4" style={{ width: '400px' }}>
                 <Card.Body>
                     <h2 className="mb-4">Login</h2>
                     <Form >
                         {commonFields.map((field) => (
                             <Form.Group style={{ textAlign: 'start', marginBottom:
'10px' }} controlId={field.controlId} key={field.controlId}>
                                 <Form.Label>{field.label}</Form.Label>
                                 <Form.Control
                                     type={field.type}
                                     placeholder={`Enter
${field.label.toLowerCase()}`}
                                     name={field.controlId}
                                     value={formData[field.controlId]}
                                     onChange={handleInputChange}
                                     required
                                 />
                             </Form.Group>
                         ))}
                         <Button onClick={handlelogin} className="btn-primary w-100
mt-3">Login</Button>
                     </Form>
                     <p >
                         Don't have an account? <Link to="/asignup">Sign Up</Link>
                     </p>
                 </Card.Body>
             </Card>
        </Container>

       </div>
          );
};

export default AdminLogin;
```

9.9.ADMIN NAVBAR:

```
// src/components/Navbar.js

import React from 'react';
import { Navbar, Nav, Container } from 'react-bootstrap';
import {Link } from "react-router-dom"

const Unavbar = () => {
  const get=localStorage.getItem('user')
  return (
    <Navbar className='bg-green-400' variant="dark" expand="lg" >
      <Container>
        <Navbar.Brand><Link to='/uhome'
style={{color:"white",textDecoration:"none"}}>Grocery Web App</Link></Navbar.Brand>
        <Navbar.Toggle aria-controls="basic-navbar-nav" />
        <Navbar.Collapse id="basic-navbar-nav">
          <Nav className="ml-auto" >
            <Link to="/Admin/dashboard"
style={{padding:"8px",color:"white",textDecoration:"none",fontSize:"22px",fontStyle
:"italic"}}>Dashboard</Link>
            <Link to="/admin/users"
style={{padding:"8px",color:"white",textDecoration:"none",fontSize:"22px",fontStyle
:"italic"}}>Users</Link>
            <Link to="/admin/all-products"
style={{padding:"8px",color:"white",textDecoration:"none",fontSize:"22px",fontStyle
:"italic"}}>Products</Link>
            <Link to="/admin/add-product"
style={{padding:"8px",color:"white",textDecoration:"none",fontSize:"22px",fontStyle
:"italic"}}>Add product</Link>
            <Link to="/admin/orders"
style={{padding:"8px",color:"white",textDecoration:"none",fontSize:"22px",fontStyle
:"italic"}}>Orders</Link>

            <Link to="/"
style={{padding:"8px",color:"white",textDecoration:"none",fontSize:"22px",fontStyle
:"italic"}}>Logout</Link>
            {/* <h4
style={{color:"white",paddingTop:"0px"}}>({JSON.parse(get).name} )</h4> */}

          </Nav>
        </Navbar.Collapse>
      </Container>
    </Navbar>
  );
};

export default Unavbar;
```

9.10.ADMIN SIGNUP:

```
import React, { useState } from 'react';
import { Container, Form, Button, Card } from 'react-bootstrap';
import { Link, useNavigate } from 'react-router-dom';
import Header from '../components/Header';
```

```
const commonFields = [
    { controlId: "username", label: "UserName", type: "text" },
    { controlId: "email", label: "Email", type: "email" },
    { controlId: "password", label: "Password", type: "password" },
];

const AdminSignup = () => {
    const [formData, setFormData] = useState({
        firstName:'',
        lastName:'',
        username:'',
        email: '',
        password: '',
    });

    const navigate = useNavigate()

    const handleSubmit = async (e) => {
        e.preventDefault();

        try {
            const response = await fetch('http://localhost:5100/register', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(formData),
            });

            if (response.ok) {
                const data = await response.json();
                console.log('Registration successful:', data);
                navigate('/login');
            } else {
                const errorData = await response.json();
                alert('Registration failed: ' + errorData.message);
            }
        } catch (error) {
            console.error('Error during registration:', error);
            alert('Error during registration');
        }
    };


    const handleInputChange = (e) => {
        const { name, value } = e.target;
        setFormData((prevData) => ({ ...prevData, [name]: value }));
    };
    const handlelogin=()=>{
        alert('Registerd Successfull')
        navigate('/admin/dashboard')
    }
    return (
       <div>
        <Header/>
         <Container className="d-flex justify-content-center align-items-center"
style={{ minHeight: '100vh',paddingTop:'10vh' }}>
            <Card className="shadow p-4" style={{ width: '400px' }}>
                <Card.Body>
                    <h2 className="mb-4">Sign Up</h2>
                    <Form onSubmit={handleSubmit}>
                        {commonFields.map((field) => (
                            <Form.Group
style={{textAlign:'start',marginBottom:'10px'}} controlId={field.controlId}
key={field.controlId}>
                                <Form.Label>{field.label}</Form.Label>
                                <Form.Control
                                    type={field.type}
                                    placeholder={`Enter
${field.label.toLowerCase()}`}
                                    name={field.controlId}
                                    value={formData[field.controlId]}
                                    onChange={handleInputChange}
                                    required
                                />
                            </Form.Group>
                        ))}
                        <Button onClick={handlelogin} className="btn-primary w-100
mt-3">Sign Up</Button>
                    </Form>
                    <p>Already have an account? <Link to="/alogin">Log
In</Link></p>
                    {/* <div className="w-100 text-center mt-3">

                    </div> */}
                </Card.Body>
            </Card>
        </Container>
       </div>
    );
};

export default AdminSignup;
```

9.11.HISTORY:

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import Cookies from 'js-cookies';
import styled from 'styled-components';
import Header from '../Header';

// Styled components
const Container = styled.div`
  padding: 20px;
  margin-top: 10vh;
  text-align: start;
`;

const Heading = styled.h2`
  font-size: 24px;
  margin-bottom: 16px;
`;

const OrderList = styled.ul`
  list-style: none;
  padding: 0;
`;

const OrderItem = styled.li`
  border: 1px solid #ccc;
  padding: 16px;
  margin-bottom: 16px;
`;

const Strong = styled.strong`
  font-weight: bold;
`;

const History = () => {
  const userId = Cookies.getItem('userId');
  const [orders, setOrders] = useState([]);

  useEffect(() => {
    axios
      .get(`http://localhost:5100/my-orders/${userId}`)
      .then((response) => {
        // Assuming response.data is an array of orders
        setOrders(response.data);
      })
      .catch((error) => {
        console.error('Error fetching orders:', error);
      });
  }, [userId]); // Include userId in the dependency array to re-fetch orders when
it changes

  return (
    <div>
      <Header/>
      <Container>
      <h1 className='text-center'>My History</h1>
      <OrderList>
        {orders.map((order) => {
          const isDelivered = order.status === 'Delivered' || order.status ===
'Canceled';

          return isDelivered ? (
            <OrderItem key={order._id} style={{ border: order.status ===
'Delivered' ? '1px solid green' : '1px solid red' }}>
              <Strong>Order ID:</Strong> {order._id} <br />
              <Strong>Name:</Strong> {order.firstname} {order.lastname} <br />
              <Strong>Phone:</Strong> {order.phone} <br />
              <Strong>Date:</Strong> {order.createdAt} <br />
              <Strong>Price:</Strong> {order.price} <br />
              <Strong>Status:</Strong> {order.status} <br />
              <Strong>Payment Method:</Strong> {order.paymentMethod} <br />
            </OrderItem>
          ) : null;
        })}
      </OrderList>
    </Container>
    </div>
  );
};

export default History;
```

9.12.MYCART :

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import Cookies from "js-cookies";
import Header from "../Header";
import { Link } from "react-router-dom";
import {
  ProductContainer,
  ProductName,
  ProductDescription,
  ProductPrice,
  ProductImage,
  Button,
  ButtonContainer,
} from "../ProductItem/styledComponents";


const MyCart = () => {
  const userId = Cookies.getItem("userId");
  const [cartData, setCartData] = useState([]);

  useEffect(() => {
    getProductsList();
  }, []);

  const getProductsList = () => {
    axios
      .get(`http://localhost:5100/cart/${userId}`)
      .then((response) => {
        setCartData(response.data);
        console.log(response.data);
      })
      .catch((error) => {
        console.error("Error fetching cart items:", error);
      });
  };

  const handleCancelClick = (productId) => {
    axios
      .delete(`http://localhost:5100/remove-from-cart/${productId}`)
      .then((response) => {
        setCartData((prevCartData) =>
          prevCartData.filter((item) => item.productId !== productId)
        );
        getProductsList();
      })
      .catch((error) => {
        console.error("Error removing product from cart:", error);
      });
  };

  return (
    <div>
      <Header />
      <br/>
      <br/>
      <h1 className="text-3xl font-semibold mt-8">My Cart</h1>

      <div className="container mx-auto px-4 my-4" >
        <ul className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-4">
          {cartData.map((product) => (
            <ProductContainer key={product._id} >
              <ProductImage src={product.image} alt={product.productname} />
              <div className="p-4">
                <ProductName className="text-xl font-semibold mb-
2">{product.productname}</ProductName>
                <p className="text-gray-700">${product.price}</p>
                <div className="mt-4 flex justify-between">
                  <button
                    onClick={() => handleCancelClick(product._id)}
                    className="px-3 py-2 bg-red-500 text-white rounded-md hover:bg-
red-600"
                  >
                    Remove from Cart
                  </button>
                  <Link
                    to={`/order-details/${product._id}`}
                    className="px-3 py-2 bg-blue-500 text-white rounded-md
hover:bg-blue-600"
                  >
                    Buy this
                  </Link>
                </div>
              </div>
            </ProductContainer>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default MyCart;
```

10. RUN THE APPLICATION

To run a grocery web app using the MERN stack, you need to ensure the proper setup for each component: MongoDB, Express.js, React, and Node.js. Follow these steps:

10.1. Start the Backend :

The backend server is powered by Node.js and runs on the port specified in the .env file (default:5000).

1. Open a terminal
2. Navigate to the server directory: cd server
3. Start the backend server: NPM start

10.2. Start the frontend:

The frontend is built using React.js and runs on port 3000 by default.

1. Open a new terminal
2. Navigate to the client director: cd client
3. Start the frontend development server: NPM start
4. The React Application will open in your default browser If it does not you can manually open : https://localhost:3000

11. API DOCUMENTATION

11.1. User Registration:

- User sends a request to register with their details (name, email, password, etc).
- The system validates the input data.
- If validation passes, a new user account is created.
- The system responds with a success message and user ID

11.2. User Login:

- User sends login credentials (email and password).
- The system verifies the credentials.
- If valid, a token (JWT or similar) is generated and returned.
- User uses the token for authenticated requests.

11.3. Get User Profile:

- User sends a request with the authentication token.
- The system verifies the token and retrieves user details.
- The system returns the user's profile information.

11.1. ADMIN API STEPS

11.1.1. Admin Login:

- Admin provides credentials to log in.
- The system verifies the credentials.
- If valid, a token is issued for admin-level access.

11.1.2. View All Users:

- Admin sends a request with their authentication token.
- The system verifies the token and fetches all user records.
- The system returns the list of users.

11.1.3. Delete a User:

- Admin sends a request to delete a specific user, providing the user ID.
- The system verifies the admin token and checks if the user exists.
- If valid, the user is deleted, and the system responds with a confirmation.

11.1.4. Update User Role:

- Admin sends a request to update a user's role, specifying the user ID and new role.
- The system verifies the admin token and checks the validity of the new role.
- If valid, the user's role is updated, and the system responds with a success message.

12. AUTHENTICATION MECHANISM

12.1. User/Admin Registration:

- Collect user/admin details (e.g., email, password) and store the password securely

12.2 .Login:

- User/admin provides email and password for authentication.

12.3. Credential Verification:

- Validate the email and password by comparing the provided password (hashed)

12.4. Token Generation:

- On successful login, generate a secure token (e.g., JWT) containing user/admin request if authorized

details and expiry.

12.5. Token Usage:

- Client includes the token in the Authorization header for subsequent requests.

12.6. Token Verification:

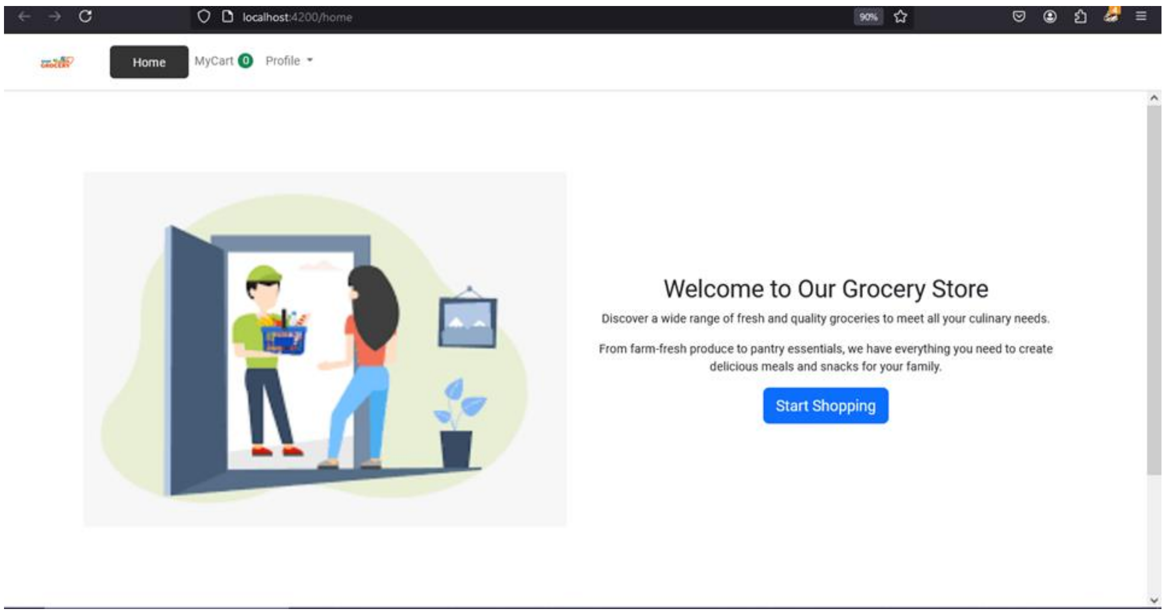
- Server verifies the token, checks user/admin permissions, and processes the requested.

13. USER INTERFACE & SCREENSHOTS

The Grocery WebApp provides an intuitive, user-friendly interface that allows customers to browse products, add items to their cart, and place orders. Below is a detailed description of the key UI features:

13.1. Home Page:

- **Features:**
  - **Header Section:** Includes the logo, navigation links (Home, Products, Cart, Login/Register).
  - **Product Categories:** Displays various product categories (e.g., Fruits, Vegetables, Bakery) that users can click to browse relevant items.
  - **Featured Products:** A carousel or grid layout of popular or featured products.
  - **Search Bar:** Allows users to search for products by name or category.



Welcome to Our Grocery Store

Discover a wide range of fresh and quality groceries to meet all your culinary needs.

From farm-fresh produce to pantry essentials, we have everything you need to create delicious meals and snacks for your family.

Fig 1.7 Home Page

13.2. Product Listing Page:

- **Features:**
  - **Category Filters:** Users can filter products by category, price range, or brand.
  - **Sort Options:** Sort products by price (ascending/descending) or popularity.
  - **Product Cards:** Each product is displayed in a card format with an image, name, price, and "Add to Cart" button.
  - **Pagination:** If there are many products, pagination is used to navigate through pages.

Example UI (Product Listing):

- A grid layout with product cards.
- Each card includes:

  1.Product image.

  2. Name and price.

  3. An "Add to Cart" button

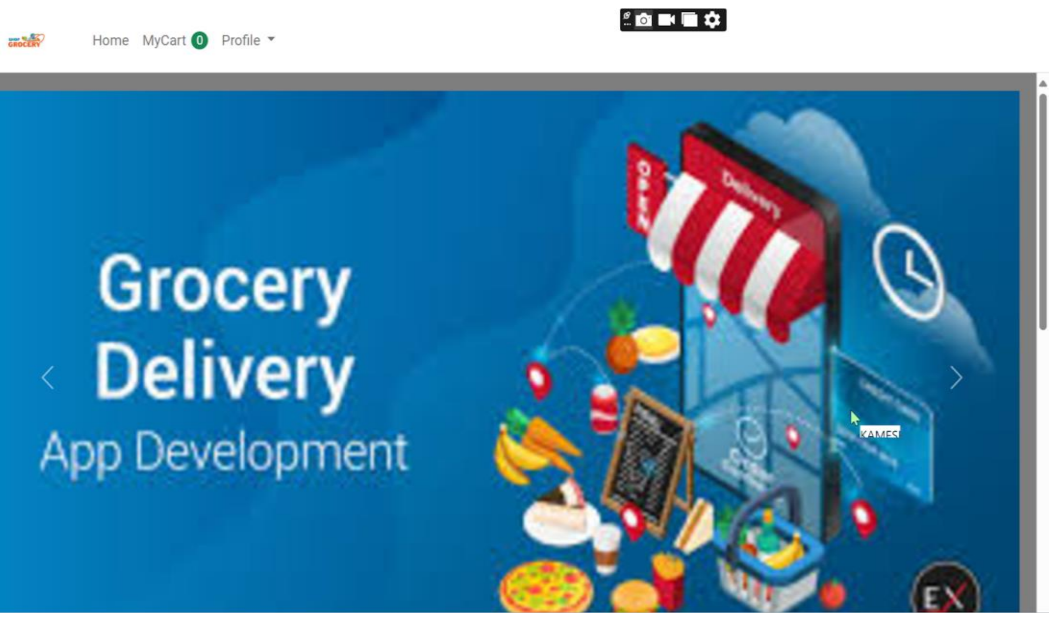
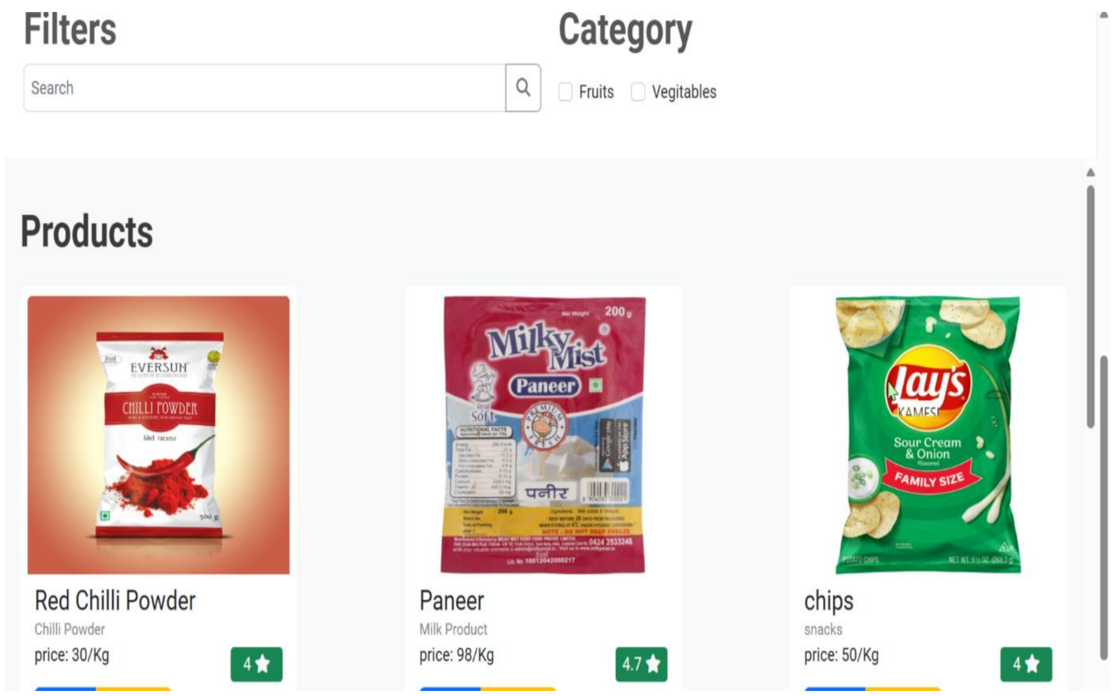
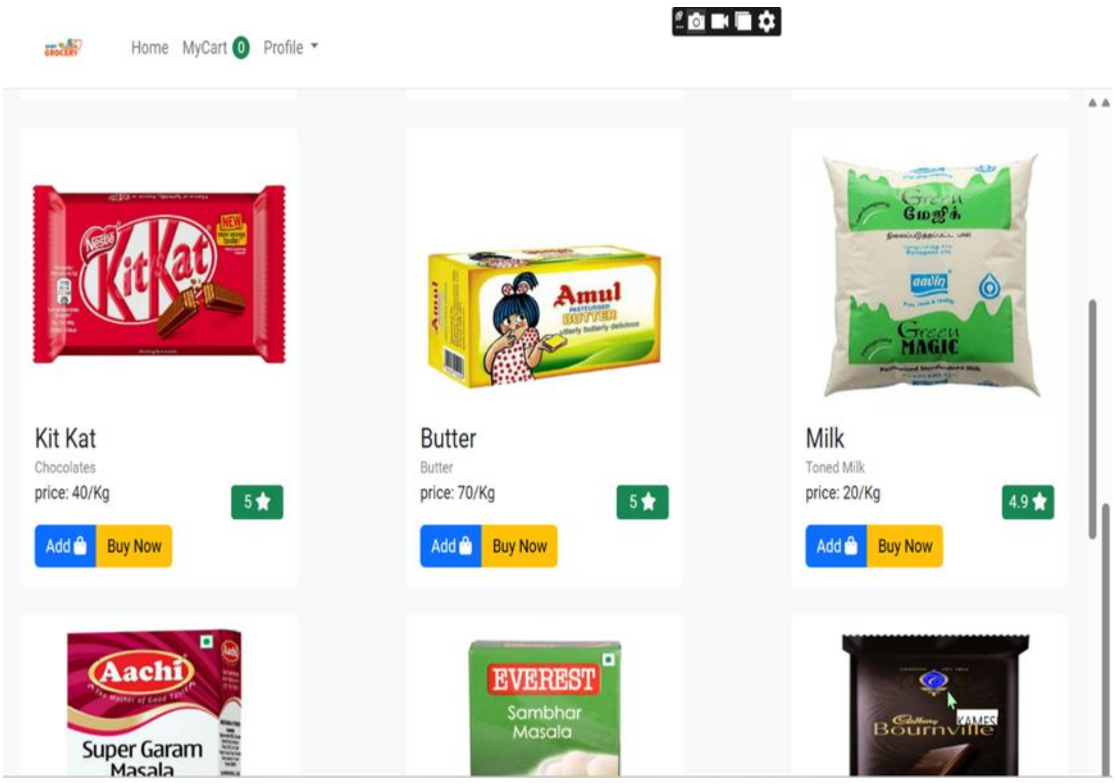


Fig 1.8 (A) Product Listing page



(B) Product Listing Page



(C) product Listing Page

13.3. Product Details Page:

- **Features:**
  - **Product Image:** A large, high-quality image of the product.
  - **Product Information:** Detailed information about the product such as description, price, available stock, etc.
  - **Quantity Selector:** Users can select the quantity they want to add to the cart.
  - **Add to Cart Button:** Allows users to add the selected quantity of the product to their cart.
  - **Reviews Section:** Shows reviews and ratings from other customers.

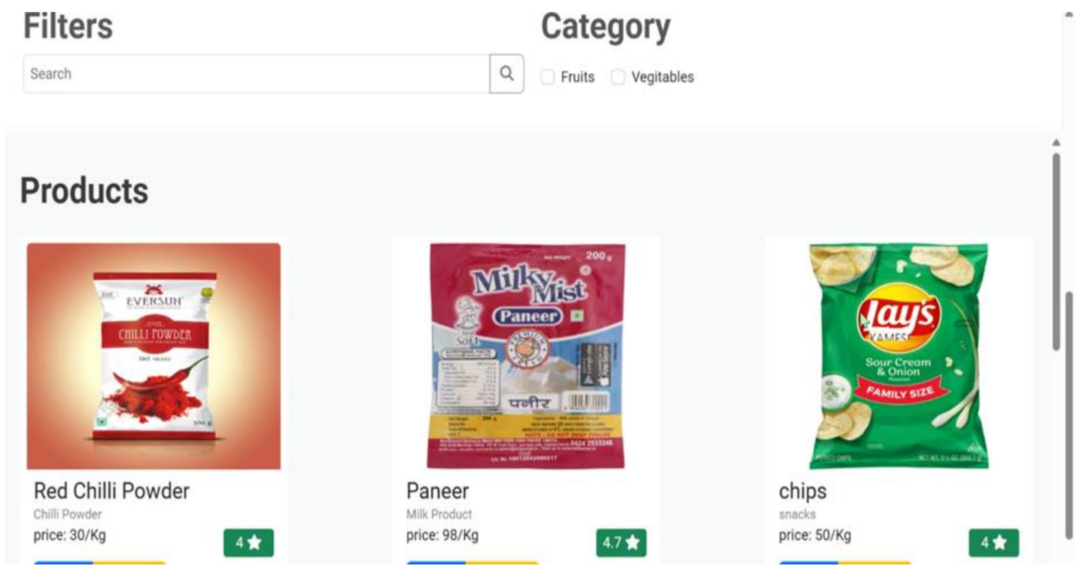


Fig 1.9 Product Detail Page

13.4.Cart Page:

- **Features:**
  - o **Cart Items List:** Displays all the products added to the cart with images, names, quantities, and prices.
  - o **Quantity Editing:** Users can change the quantity of each product or remove it from the cart.
  - o **Total Price Calculation:** Shows the total price of all items in the cart.
  - o **Checkout Button:** Allows users to proceed to checkout and confirm their order.

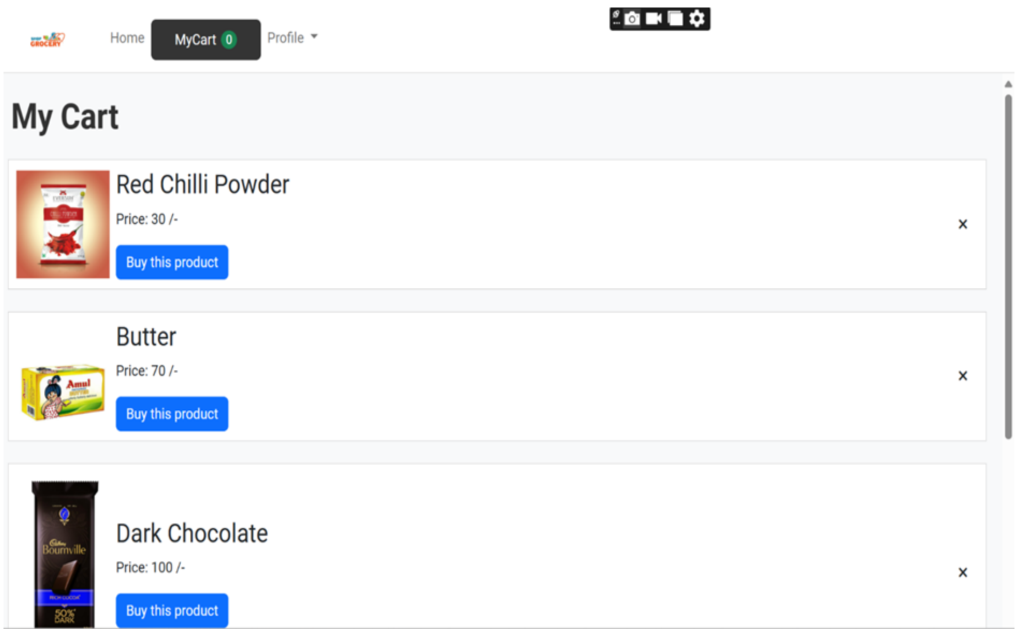


Fig 1.10 Cart Page

12.5.Login/Signup Page:

- **Features:**
  - o **Login Form:** Contains fields for email and password, with a "Login" button.
  - o **Signup Form:** New users can sign up with name, email, and password fields, along with a "Sign Up" button.
  - o **Validation:** Both forms have validation for required fields and proper format (e.g., email).
  - o **Forgot Password Link:** A link for users to reset their password if they forget it.

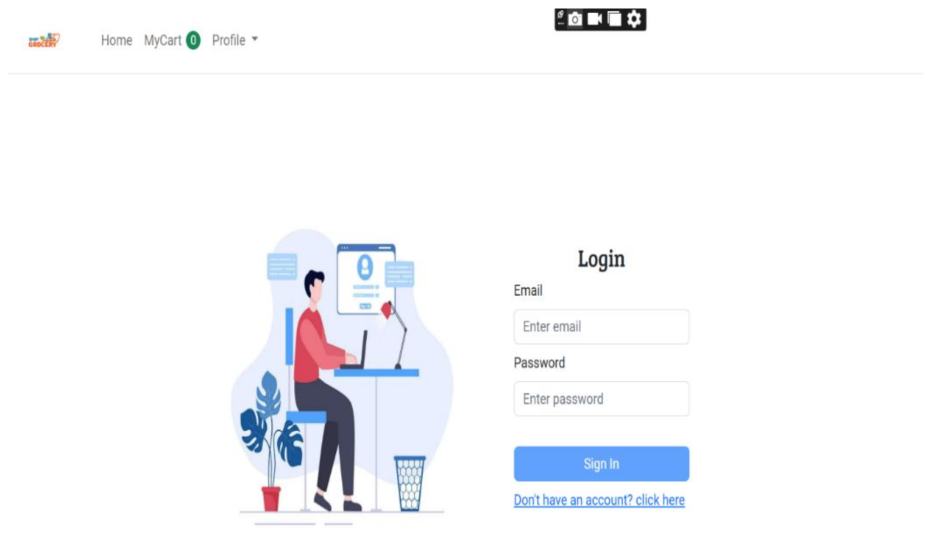
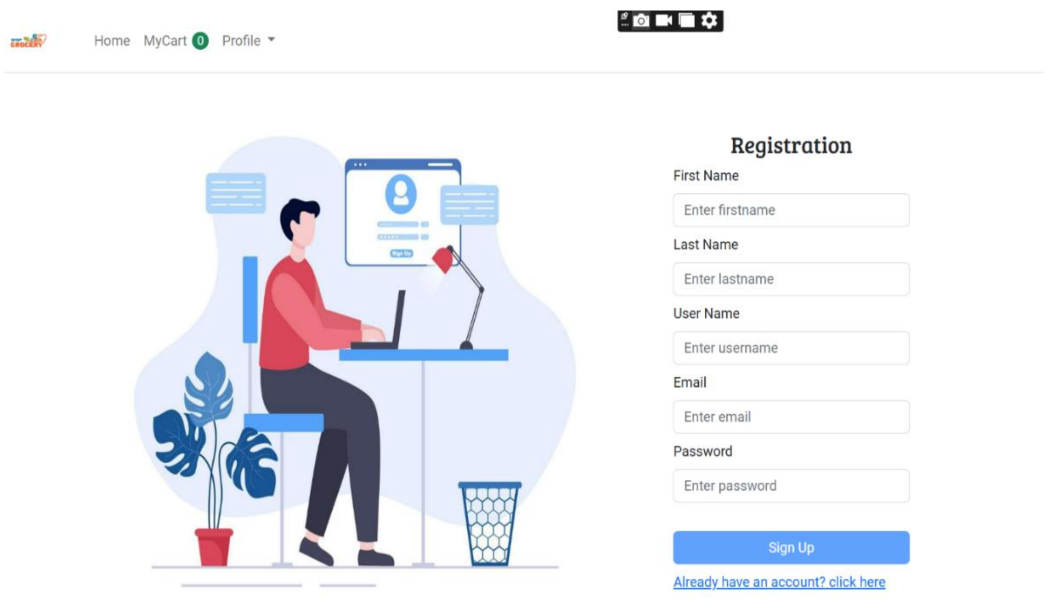


Fig 1.11 Login/Sign Up Page



Registration

First Name

Enter firstname

Last Name

Enter lastname

User Name

Enter username

Email

Enter email

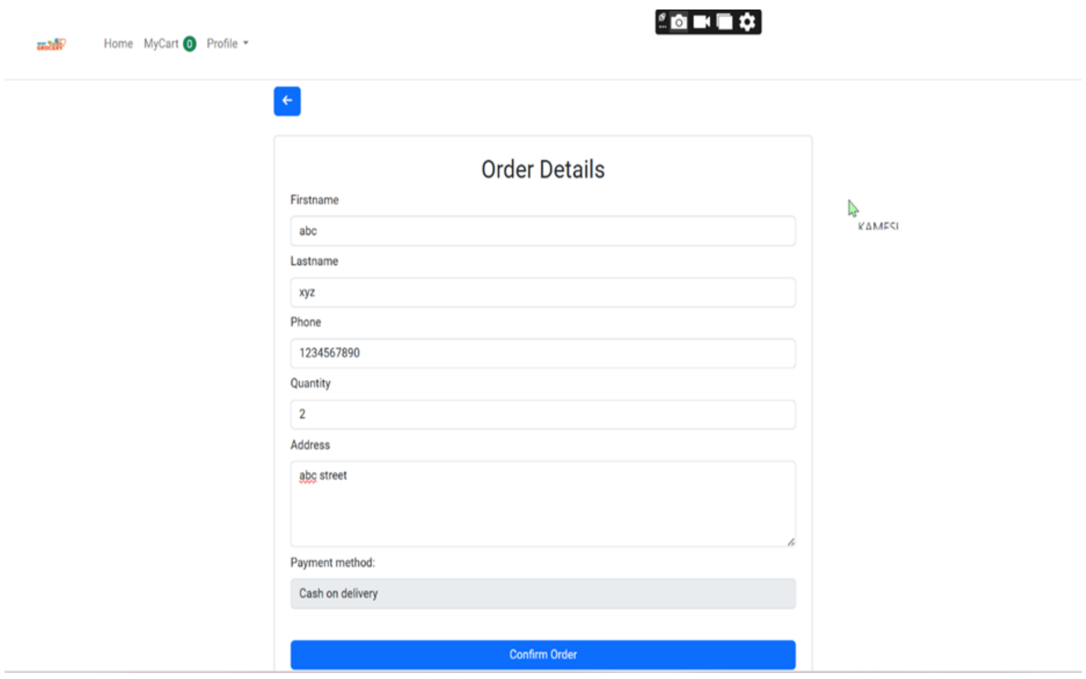
Password

Enter password

Fig 1.11 Login/Sign Up Page

13.6. Checkout Page:

- **Features:**
    - **Shipping Information Form:** A form to enter shipping details like name, address, and contact information.
    - **Order Summary:** Displays the items in the cart along with their prices and total.
    - **Payment Integration:** Options to choose the payment method (e.g., Credit Card, PayPal).
    - **Place Order Button:** A button to finalize and place the order


Home MyCart 0 Profile

Order Details

Firstname
abc

Lastname
xyz

Phone
1234567890

Quantity
2

Address
abc street

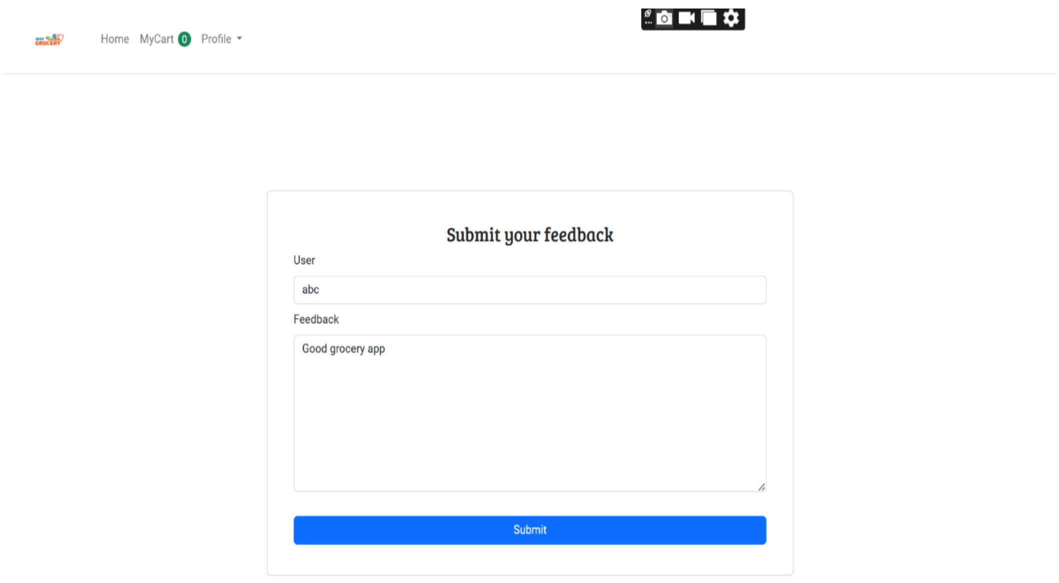
Payment method:
Cash on delivery

Confirm Order

Fig 1.12 Checkout Page

13.7. Confirmation Page (Order Success):

- **Features:**
  - **Order Confirmation:** Displays a message confirming that the order has been placed successfully.
  - **Order Details:** Provides the user with details of the order, including the products purchased, total price, and shipping address.
  - **Next Steps:** Information about the order processing or estimated delivery time.



Submit your feedback

User

abc

Feedback

Good grocery app

Submit

Fig 1.13 Confirmation Page

13.8. Admin Dashboard:

- **Features:**
  - **Overview Widgets:** Key metrics such as total sales, active users, total orders, and inventory status displayed in widgets.
  - **Recent Orders:** A table or list of the most recent orders with order ID, customer name, date, and status.
  - **Quick Links:** Easy navigation to manage products, categories, orders, and users.

- **Charts:** Graphical representations of sales trends, user growth, or product performance.

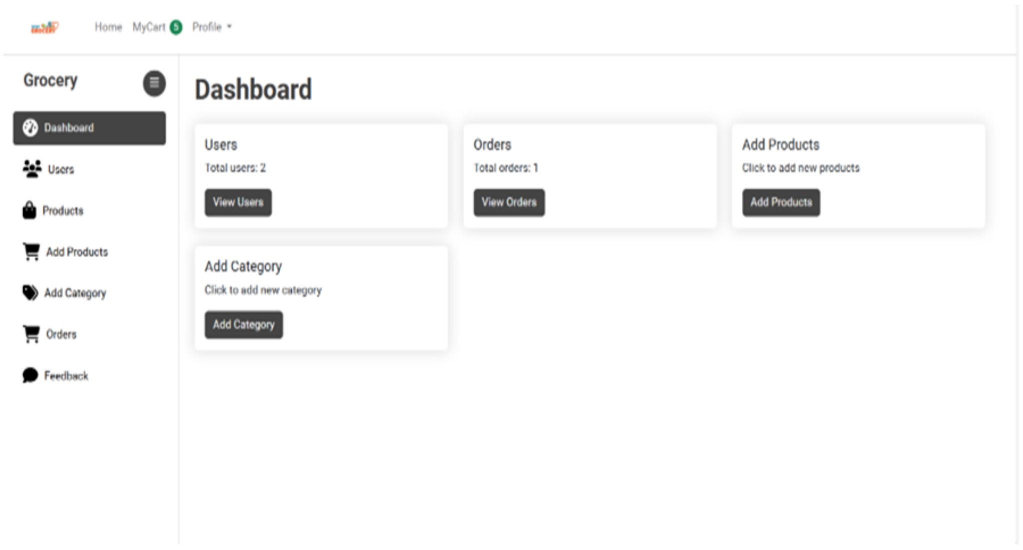


Fig 1.14 Admin Dashboard code

13.9. Product Management:

- **Features:**
  - **Product List:**
    - A table displaying all products with columns for name, category, stock, price, and actions.
    - Pagination for better navigation if there are many products.
  - **Add Product:**
    - A form to add a new product with fields for name, category, price, description, stock, and product image.
  - **Edit Product:**
    - Allows admins to update details for an existing product, such as price, stock, or description.
  - **Delete Product:**

- Remove products from the catalogue with confirmation prompts to avoid accidental deletions.

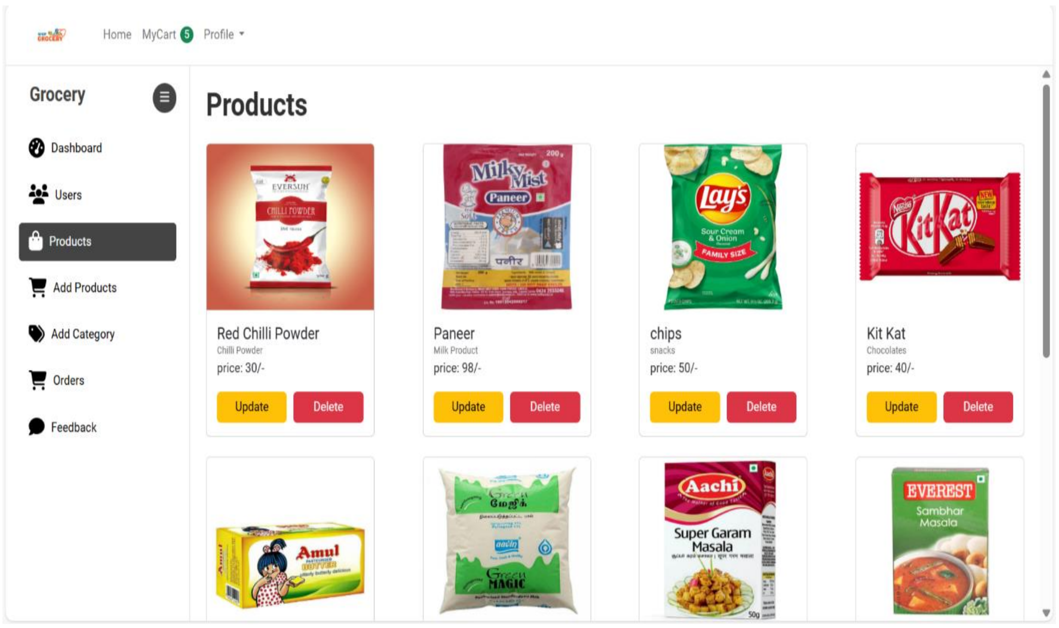
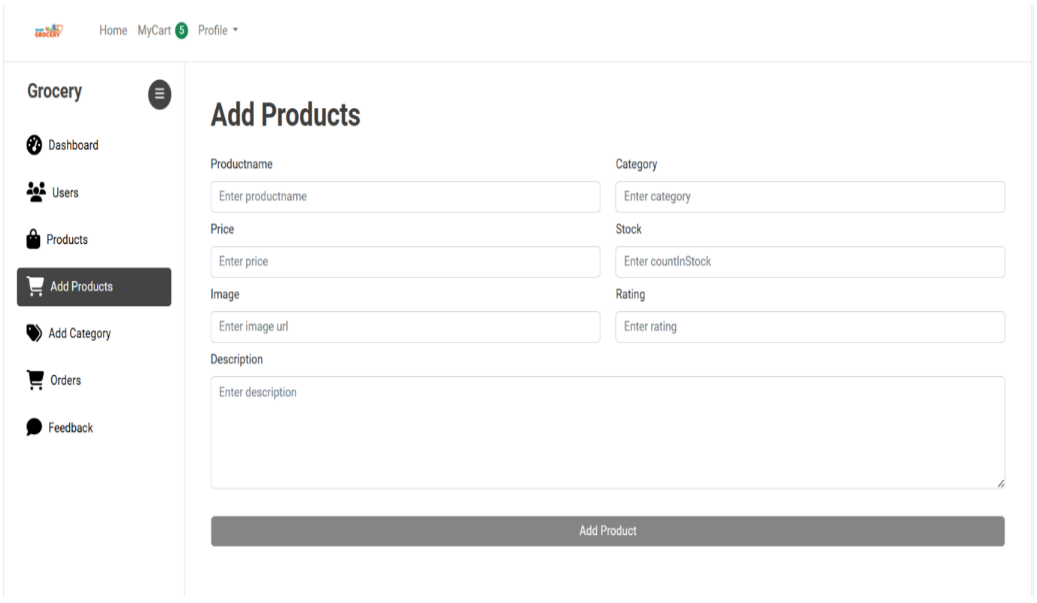


Fig 1.15 (a) Product Management Page



(b) Product Management Page

13.10. Category Management:

- **Features:**
    - **Category List:**
        - View all categories in a table format.
        - Includes options to edit or delete categories.
    - **Add Category:**
        - A form to add new categories with fields for name and description.
    - **Edit Category:**
        - Update existing categories' names or descriptions

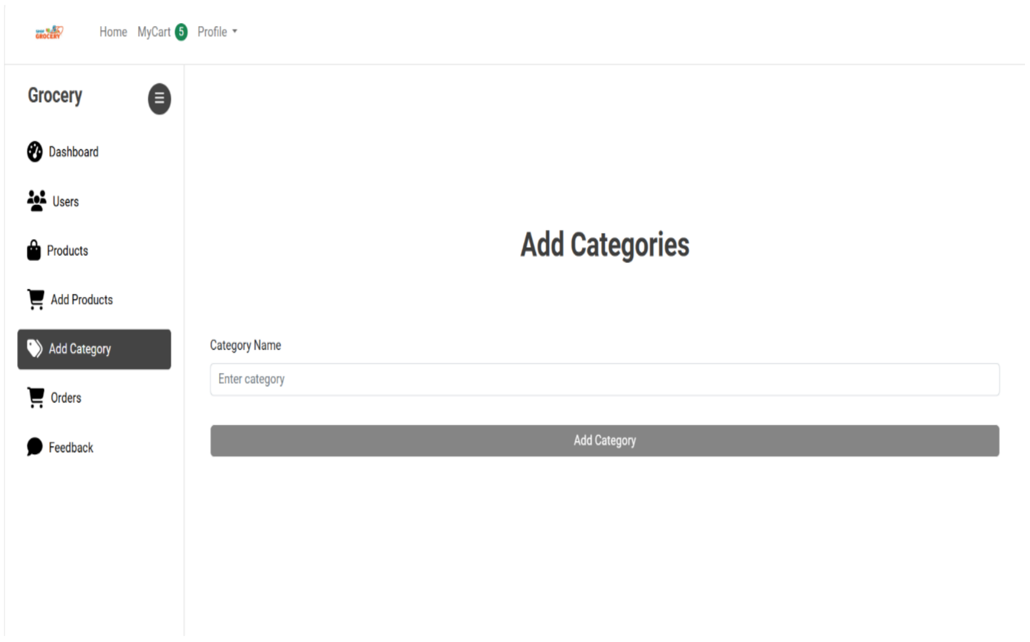
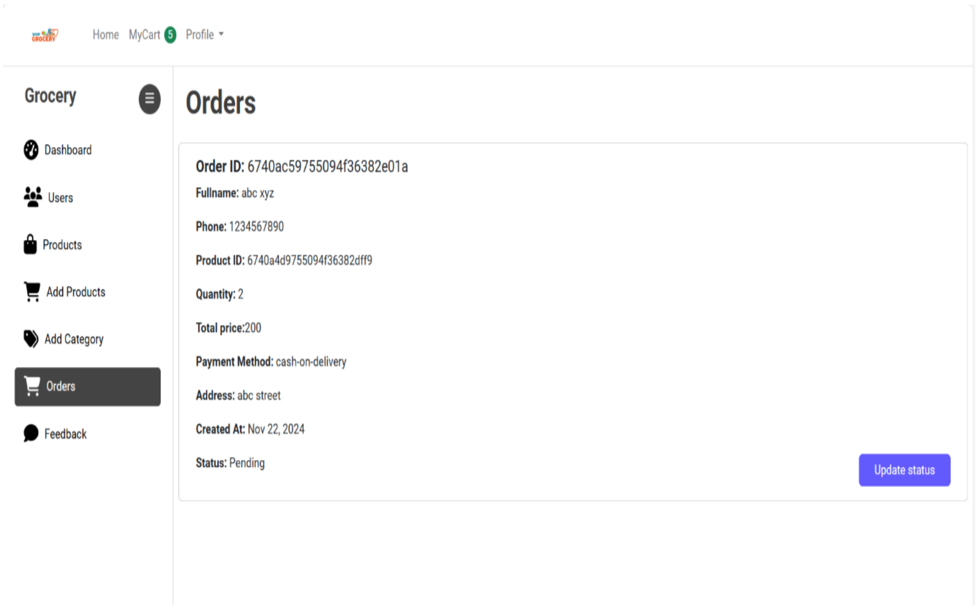


Fig 1.18 Category Management Page

13.11. Order Management:

- **Features:**
  - **Order List:**
    - A table showing all orders with details such as order ID, customer name, order date, status, and total amount.
    - Includes search and filter options (e.g., filter by date, status, or customer).
  - **Order Details:**
    - View detailed information about an individual order, including:
    - Products purchased (name, quantity, and price).
    - Shipping details (name, address, contact information).
    - Payment status (paid/pending).
  - **Update Order Status:**
    - Change the status of an order (e.g., Pending, Shipped, Delivered, Cancelled).



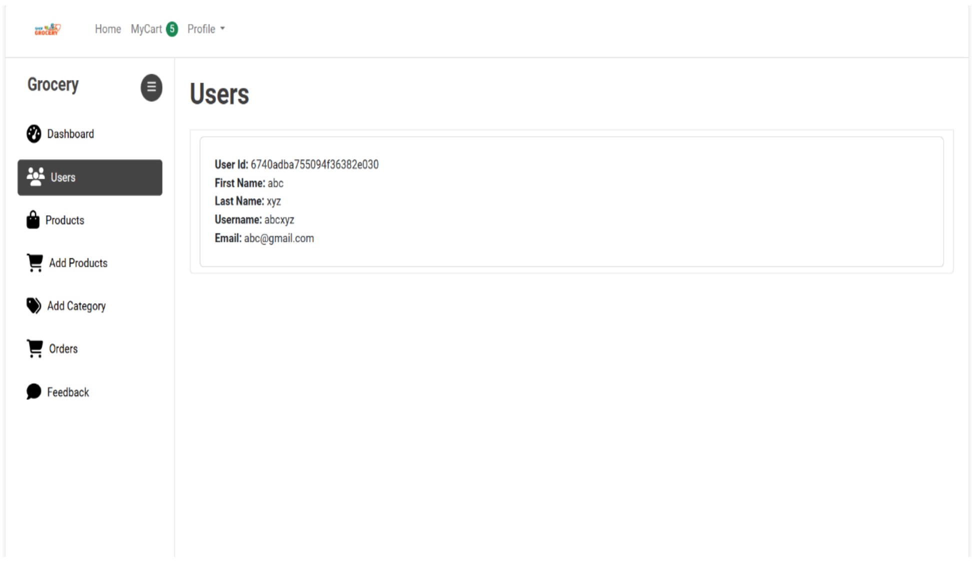
Orders

Order ID: 6740ac59755094f36382e01a
Fullname: abc xyz
Phone: 1234567890
Product ID: 6740a4d9755094f36382dff9
Quantity: 2
Total price:200
Payment Method: cash-on-delivery
Address: abc street
Created At: Nov 22, 2024
Status: Pending

Fig 1.17 Order Management page

13.12. User Management:

- **Features:**
    - **User List:**
        - View all registered users in a table format, including details like name, email, registration date, and status.
    - **Edit User Role:**
        - Assign or update user roles (e.g., customer, admin).
    - **Activate/Deactivate User:**
        - Temporarily disable or enable user account



Users

User Id: 6740adba755094f36382e030
First Name: abc
Last Name: xyz
Username: abcxyz
Email: abc@gmail.com

Fig 1.19 User Management page

14. TESTING STRATEGY

A comprehensive testing strategy for a grocery web app built using the MERN stack (MongoDB, Express.js, React, Node.js) should cover both frontend and backend aspects, ensuring a seamless user experience and reliable performance. Here's a detailed strategy:

13.1. Functional Testing:

- Purpose: Verify that all core features, like order placement, payment, and item selection, work as expected.
- Tools: Selenium, Postman (for API testing), Cypress.

14.2. Security Testing:

- Purpose: Check for vulnerabilities like SQL injection, XSS, and ensure secure handling of sensitive data (e.g., passwords and payment details).
- Tools: OWASP ZAP, Burp Suite.

14.3. Performance Testing:

- Purpose: Test how the system performs under heavy traffic, checking for slowdowns or crashes.
- Tools: Apache JMeter, LoadRunner, Gatling

14.4. Unit Testing:

- Purpose: Test individual components, functions, and modules in isolation.

- Tools: Jest, React Testing Library, Mocha, Chai, Sino

15. KNOWN ISSUES

A grocery web app built using the MERN stack (MongoDB, Express.js, React, and Node.js) may encounter various issues during development and deployment. Here are some common problems:

15.1.Backend Issues:

1. Database Performance:

Inefficient MongoDB queries can lead to slow response times, especially when dealing with a large number of products or user data. Lack of proper indexing in MongoDB.

2. API Design Flaws:

Missing or inconsistent API endpoints can lead to errors in the front-end. Poorly documented APIs can make debugging and future development harder.

15.2. Frontend Issues:

1. State Management:

Mismanagement of application state in React (e.g., Redux, Context API, or Zustand) can lead to bugs like incorrect cart updates or inconsistent UI states.

2. Component Rendering:

Inefficient rendering of components causing performance bottlenecks. Overloading the UI with unnecessary re-renders due to improper use of React lifecycle hooks.

16. FUTURE ENHANCEMENTS

Enhancing a grocery web app built with the MERN stack can make it more user-friendly, feature-rich, and scalable. Here are some potential future enhancements categorized by functionality:

16.1. Mobile App Integration:

- Enhancement: Develop a mobile application for iOS and Android to reach a broader audience and improve user experience.
- Tools: React Native, Flutter, Xamarin

16.2. AI and Personalization:

- Enhancement: Implement AI-driven features like personalized recommendations, predictive text for faster ordering, and user behaviour analysis for targeted promotions.
- Tools: TensorFlow, Python, Google Cloud AI, Firebase ML

16.3. Real-time Order Tracking:

- Enhancement: Add real-time order tracking so customers can monitor their order status and delivery progress.
- Tools: Firebase Realtime Database, Web Sockets, Google Maps API.

16.4. User Experience (UX) Enhancements:

- Implement a recommendation engine using machine learning to suggest products based on user purchase history, preferences, or trends.
- Add voice search functionality. Provide advanced filtering options like dietary preferences, brands, or organic products.

17. CONCLUSION

The **Grocery WebApp** is a full-featured platform designed to streamline online grocery shopping for users and empower administrators to manage the system efficiently.

Built with the MERN stack, the app leverages React for a dynamic user interface, Node.js and Express.js for a robust backend, and MongoDB for scalable data storage.

Key features like user authentication, cart management, and seamless checkout ensure an excellent user experience, while the admin panel offers tools for effective product, order, and user management.

Through rigorous testing and thoughtful design, the application demonstrates reliability, usability, and scalability, making it suitable for real-world deployment.

The project not only showcases technical expertise in full-stack development but also emphasizes modern development practices, including modular architecture, RESTful API design, and responsive UI development.

Future enhancements, such as integrating AI-powered recommendations or advanced analytics, can further enrich the app’s functionality and provide added value to users.

18. REFERENCE

1. **React Documentation** - https://reactjs.org/docs

2. **Node.js Documentation** - https://nodejs.org/en/docs

3. **Express.js Guide** - https://expressjs.com/en/guide

4. **MongoDB Documentation** - https://www.mongodb.com/docs

5. **JWT Authentication** - https://jwt.io/introduction

6. **Cypress Testing** - https://www.cypress.io

7. **Bootstrap (for responsive design)** - https://getbootstrap.com/docs

8. **Postman (for API testing)** - https://www.postman.com

9. **MERN Stack Tutorials** -https://www.freecodecamp.org/news/mern-stack-tutorial