GROCERY WEBAPP USING MERN STACK

A Project Done By

KAMESHWARAN B	**211121104027**
YUVARAJ N	**211121104059**
KEERTHANA R	**211121104029**
KERAN TABERAH E	**211121104031**
ABIRAMI	**211121104002**

STUDENTS OF

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

MADHA ENGINEERING COLLEGE

KUNDRATHUR

CHENNAI-69

Table of Content

Abstract

The Grocery WebApp is an innovative solution for online grocery shopping, developed using the MERN (MongoDB, Express.js, React, Node.js) stack. This project aims to provide users with a seamless shopping experience while enabling administrators to efficiently manage the platform. Key features for users include a modern interface with product browsing, advanced filtering, a secure authentication system, a dynamic shopping cart, and an intuitive checkout process.

On the administrative side, the platform offers robust tools for managing products, categories, orders, and users, as well as generating reports and monitoring system performance. The application follows a modular architecture, ensuring scalability, maintainability, and responsiveness.

With a strong focus on user experience, security, and performance, the Grocery WebApp is well-suited to meet the demands of the evolving e-commerce landscape, offering a reliable platform for both users and administrators. Future enhancements, including AI-powered recommendations and advanced analytics, promise to further elevate its utility and effectiveness.

List Of Figures

1. INTRODUCTION

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!

1.1 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.

1.2 Key Features:

1. Product Browsing:
- Users can search, filter, and sort products by categories, price, or popularity.
2. User Authentication:
- Secure login and signup functionality with JWT-based authentication.
3. Shopping Cart:
- Real-time cart updates for adding, removing, or modifying items.
4. Order Management:
- Users can view their order history and track current orders.
5. Admin Panel:
- Administrative tools for managing products, monitoring sales, and analysing customer data.

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

2. ARCHITECTURE

2.1 Technical Architecture:

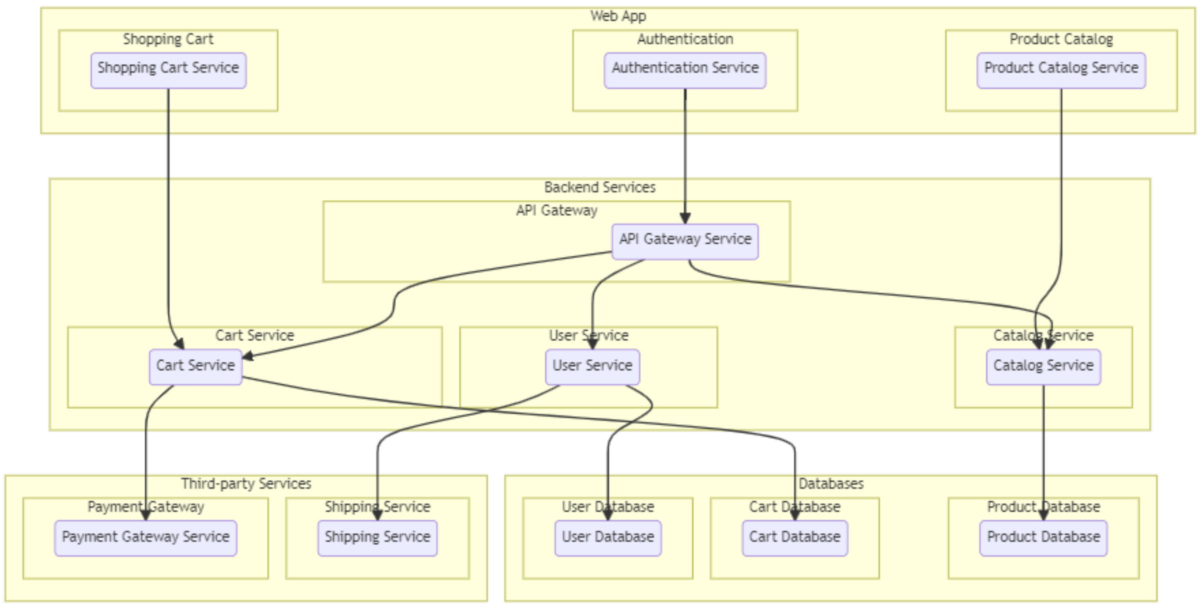


Fig 1.1 Technical Architecture

The technical architecture of an 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.

2.2 ER Diagram:

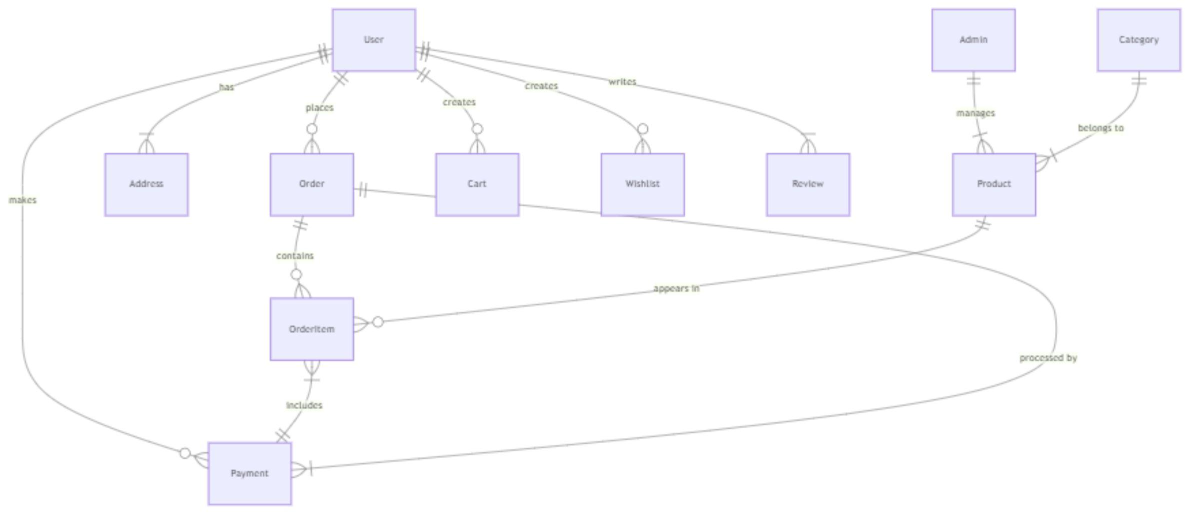


Fig 1.2 ER Diagram

The Entity-Relationship (ER) diagram for an 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.

2.3. 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 ProductCard, 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.

<u>2.4. Backend Architecture (Node.js + Express.js):</u>

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.

2.5. Database Architecture (MongoDB):

MongoDB, a NoSQL database, is used for flexible and scalable data storage.

- **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.

3. SETUP INSTRUCTION

Prerequisites:

- Node.js (v14 or higher)
- MongoDB (locally or via a cloud service like MongoDB Atlas)
- Git
- npm or yarn

Installation:

1. **Clone the repository:**

```
git clone <repository-url>
cd <repository-folder>
```

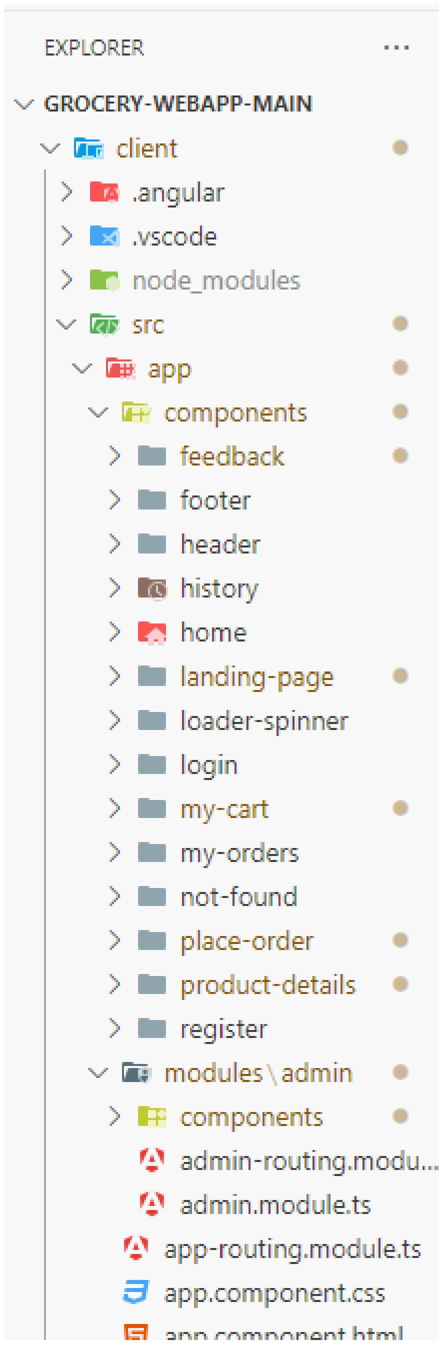
2. **Install dependencies for both frontend and backend:**

```
cd client
npm install
cd ../server
npm install
```

3. **Set up environment variables:**

- Create a .env file in the server directory.
- Add variables such as database connection string, JWT secret, etc.

Project Structure



```
EXPLORER ...
GROCERY-WEBAPP-MAIN
  client
    .angular
    .vscode
    node_modules
    src
      app
        components
          feedback
          footer
          header
          history
          home
          landing-page
          loader-spinner
          login
          my-cart
          my-orders
          not-found
          place-order
          product-details
          register
        modules\admin
          components
          admin-routing.modu...
          admin.module.ts
        app-routing.module.ts
        app.component.css
        app.component.html
```

Fig 1.3 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, placeorder, 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.

<u>Running the Application:</u>

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 se
rver
```

3. Start the backend server:

```
npm start
```

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 directory:

```
cd client
```

3. Start the frontend development server:

```
npm start
```

4. The React application will open in your default browser. If it doesn’t, you can manually open: **http://localhost:3000**

4. API DOCUMENTATION

4.1. Authentication Endpoints

4.1.1. User Signup

- Endpoint: /api/auth/signup
- Method: POST
- Description: Allows new users to create an account.
- **Request:**

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "securepassword"
}
```

- **Response:**

```
{
  "message": "User registered successfully",
  "user": {
    "id": "648c6d8f58b7f8e95b5c",
    "name": "John Doe",
    "email": "john@example.com"
  }
}
```

4.1.2. User Login

- Endpoint: /api/auth/login
- Method: POST
- Description: Authenticates the user and provides a JWT token.

- Request Body:

```
    {
     "email": "john@example.com",
     "password": "securepassword"
    }
```

- Response:

```
{
 "message": "Login successful",
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI..."
}
```

4.2. Product Endpoints

4.2.1. Get All Products

- Endpoint: /api/products
- Method: GET
- Description: Fetches all available products.
- Response:

```
    [
     {
      "id": "648d7c92df7a2b8f3e3a",
      "name": "Apple",
      "price": 2.5,
      "category": "Fruits",
      "stock": 50
     },
     {
      "id": "648d7c93df7a2b8f3e3b",
      "name": "Bread",
      "price": 1.5,
      "category": "Bakery",
      "stock": 30
    }
]
```

4.2.2. Get Product by ID

- Endpoint: /api/products/:id
- Method: GET
- Description: Fetches a specific product by ID.

- Response:

```
• {
•   "id": "648d7c92df7a2b8f3e3a",
•   "name": "Apple",
•   "price": 2.5,
•   "category": "Fruits",
•   "stock": 50
• }
```

4.3. Cart Endpoints

4.3.1. Add Item to Cart

- Endpoint: /api/cart
- Method: POST
- Description: Adds an item to the user’s cart.
- Request Body:

```
    {
      "productId": "648d7c92df7a2b8f3e3a",
      "quantity": 3
    }
```

- Response:

```
{
  "message": "Item added to cart",
  "cart": {
    "id": "648d8c34df7b2f5e2a2c",
    "items": [
      {
        "productId": "648d7c92df7a2b8f3e3a",
        "quantity": 3
      }
    ]
  }
}
```

4.3.2. Get User’s Cart

- Endpoint: /api/cart
- Method: GET
- Description: Fetches the current user’s cart.
- Response:

```
    {
      "id": "648d8c34df7b2f5e2a2c",
      "items": [
        {
```

```
      "productId": "648d7c92df7a2b8f3e3a",
      "name": "Apple",
      "price": 2.5,
      "quantity": 3
    }
  ],
  "totalPrice": 7.5
}
```

4.4. Order Endpoints

4.4.1. Place Order

- Endpoint: /api/orders
- Method: POST
- Description: Places an order for the items in the user’s cart.
- Request Body:

```
{
  "address": "123 Main Street, Cityville"
}
```

- Response:

```
{
 "message": "Order placed successfully",
 "order": {
  "id": "648d9e93df7a2b8f3e4a",
  "items": [
   {
    "productId": "648d7c92df7a2b8f3e3a",
    "quantity": 3
   }
  ],
  "totalPrice": 7.5,
  "address": "123 Main Street, Cityville",
  "status": "Processing"
 }
}
```

4.4.2. Get Order History

- Endpoint: /api/orders
- Method: GET
- Description: Fetches all past orders for the logged-in user.
- Response:

```
[
  {
    "id": "648d9e93df7a2b8f3e4a",
    "items": [
      {
        "productId": "648d7c92df7a2b8f3e3a",
        "name": "Apple",
        "quantity": 3
      }
    ],
    "totalPrice": 7.5,
    "address": "123 Main Street, Cityville",
    "status": "Delivered"
  }
]
```

4.5. Error Handling

In case of errors, the API responds with a descriptive message and status code. Example:

Error Response:

```
{
  "error": "Product not found",
  "statusCode": 404
}
```

4.6 Authentication:

The authentication system of the Grocery WebApp ensures secure access to the platform for users, allowing them to sign up, log in, and maintain their session via JWT (JSON Web Token). The backend manages user authentication using JWT-based token authentication. Here's how it works in detail:

4.6.1. User Authentication Flow

1. **Signup**
   - A new user registers by providing basic details like name, email, and password.
   - The backend validates the provided data, hashes the password, and stores it in the database.
   - A JWT token is generated and returned to the user, allowing them to authenticate future requests.
2. **Login**
   - An existing user provides their email and password.
   - The backend verifies the credentials and, if valid, generates a JWT token.
   - This token is sent back to the client and stored (usually in localStorage or cookies) for future requests.
3. **Protected Routes**

- Certain routes (e.g., cart management, order placement) are protected and can only be accessed if the user is authenticated.
- The client must send the JWT token in the Authorization header of the request to access these routes.
- The backend verifies the token’s validity and grants access if the token is valid.

4.6.2. Backend Authentication Flow

2.1. User Signup

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "securepassword"
}
```

- Endpoint: /api/auth/signup
- Method: POST
- Description: Registers a new user. The server will hash the password and store user details in the database.

- Request Body:

- Response:

```
{
  "message": "User registered successfully",
  "user": {
    "id": "648c6d8f58b7f8e95b5c",
    "name": "John Doe",
    "email": "john@example.com"
  }
}
```

Password Hashing:

Passwords are hashed using **bcrypt** before storing them in the database for security purposes.

2.2. User Login

- Endpoint: /api/auth/login
- Method: POST
- Description: Authenticates a user by comparing the provided credentials with stored ones and returns a JWT token.
- Request Body:

```
{
  "email": "john@example.com",
  "password": "securepassword"
}
```

- Response:

```
{
  "message": "Login successful",
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY0OGM2ZDhkZjc4YzY1Mj
FiYmZmYmNmM2MwZDZkNTExZmQzIiwiaWF0IjoxNjgyNzQ2NjU4fQ.ZJ2XYnk
_bAKjXJJJ8-V9Yo5sM6YB3q6fmIl4HfouZpY"
}
```

JWT Token:

The generated JWT token contains the user’s id and is valid for a set period (e.g., 1 hour). It is signed using a secret key

4.6.3. Token-based Authentication

3.1. JWT Authentication Middleware

To protect routes, a middleware is used to check the validity of the JWT token.

- **Middleware Function:**
  The middleware extracts the token from the **Authorization** header, decodes it, and verifies it. If the token is valid, it allows the request to proceed. If not, it sends an error response.

```
const jwt = require('jsonwebtoken');

const authenticateToken = (req, res, next) => {
  const token = req.header('Authorization')?.split(' ')[1]; // Extract token from
"Bearer <token>"
  if (!token) {
    return res.status(401).json({ error: 'Access denied. No token provided.' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET); // Verify the
token
    req.user = decoded; // Attach decoded user data to the request
    next(); // Proceed to the next middleware/handler
  } catch (error) {
    res.status(400).json({ error: 'Invalid token.' });
  }
};
```

3.2. Using Middleware on Protected Routes

Once the authenticateToken middleware is set up, you can use it to protect routes that require authentication.

- **Example: Protected Cart Route**

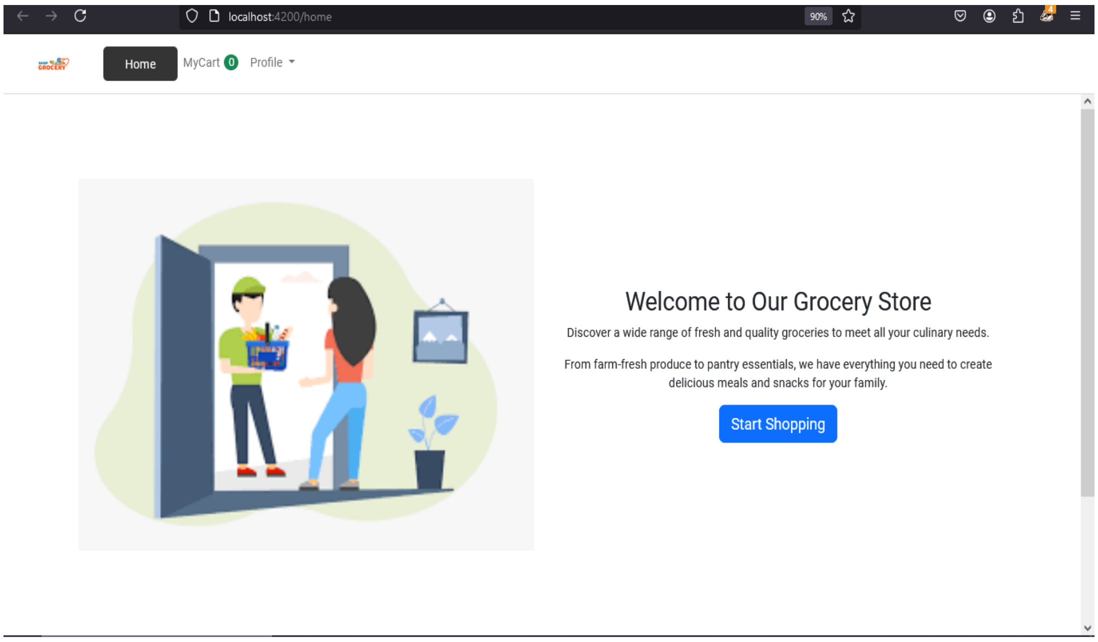
```
const express = require('express');
const { authenticateToken } = require('./middleware/auth');

const router = express.Router();

router.get('/api/cart', authenticateToken, (req, res) => {
  // This route is protected. Only accessible if the user is authenticated.
  res.status(200).json({ message: 'Cart details here' });
});
```

5. 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:

5.1. 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.

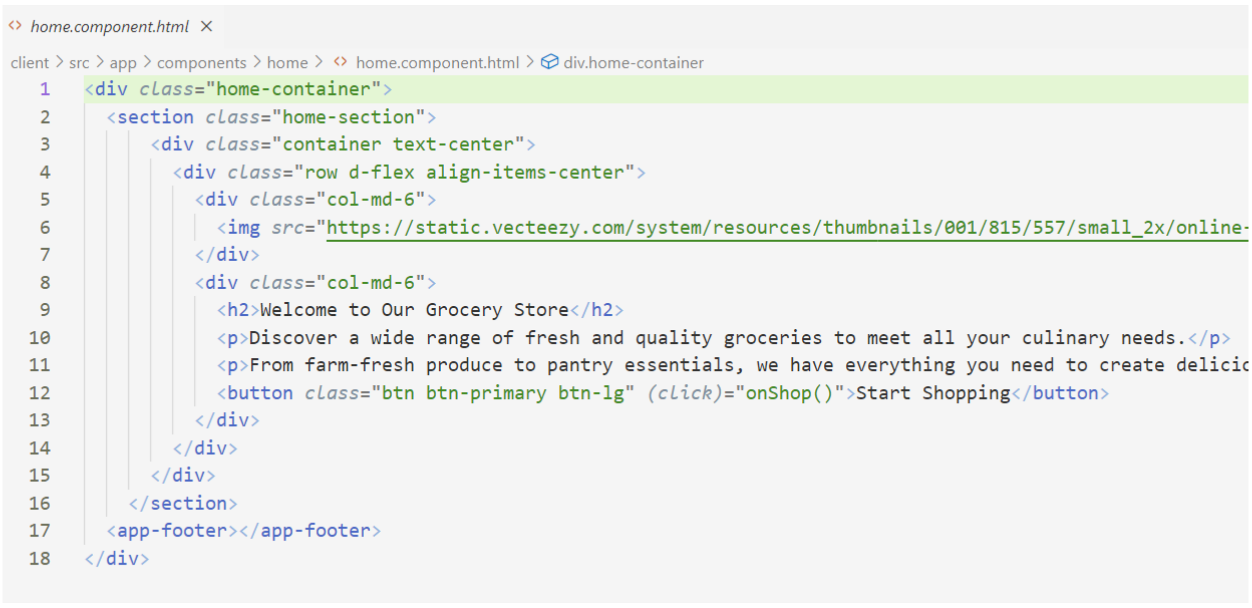


Welcome to Our Grocery Store

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 1.4 Home Page


home.component.html

client > src > app > components > home > home.component.html > div.home-container

```
<div class="home-container">
  <section class="home-section">
    <div class="container text-center">
      <div class="row d-flex align-items-center">
        <div class="col-md-6">
          <img src="https://static.vecteezy.com/system/resources/thumbnails/001/815/557/small_2x/online-
        </div>
        <div class="col-md-6">
          <h2>Welcome to Our Grocery Store</h2>
          <p>Discover a wide range of fresh and quality groceries to meet all your culinary needs.</p>
          <p>From farm-fresh produce to pantry essentials, we have everything you need to create delicio
          <button class="btn btn-primary btn-lg" (click)="onShop()">Start Shopping</button>
        </div>
      </div>
    </div>
  </section>
  <app-footer></app-footer>
</div>
```

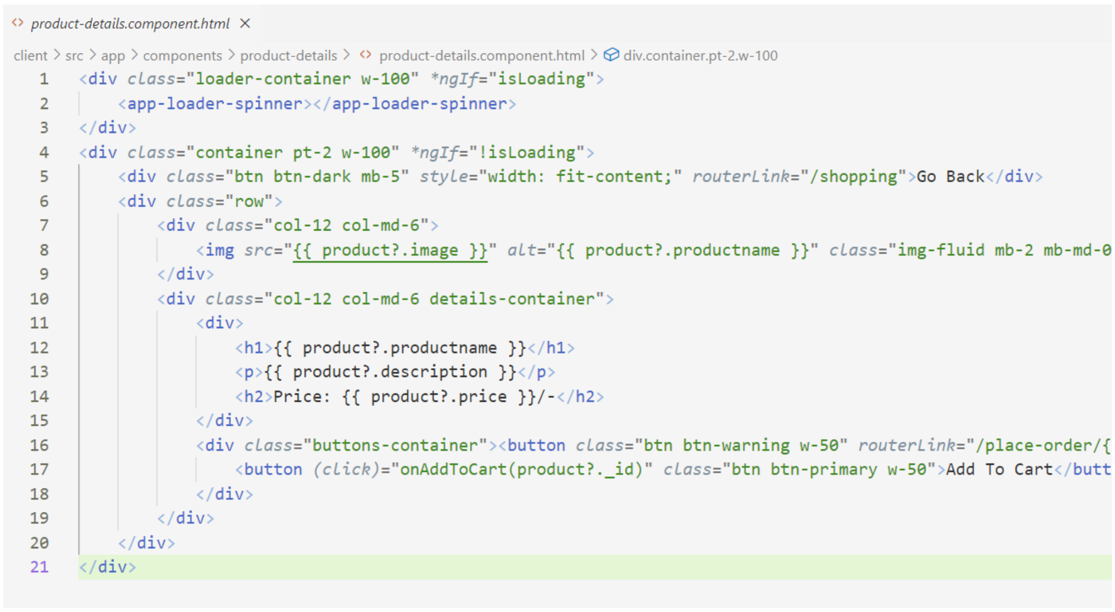
Fig 1.5 Home Page Code

5.2. 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:
  - Product image.
  - Name and price.
  - An "Add to Cart" button.


product-details.component.html

client > src > app > components > product-details > product-details.component.html > div.container.pt-2.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
    <app-loader-spinner></app-loader-spinner>
</div>
<div class="container pt-2 w-100" *ngIf="!isLoading">
    <div class="btn btn-dark mb-5" style="width: fit-content;" routerLink="/shopping">Go Back</div>
    <div class="row">
        <div class="col-12 col-md-6">
            <img src="{{ product?.image }}" alt="{{ product?.productname }}" class="img-fluid mb-2 mb-md-0
        </div>
        <div class="col-12 col-md-6 details-container">
            <div>
                <h1>{{ product?.productname }}</h1>
                <p>{{ product?.description }}</p>
                <h2>Price: {{ product?.price }}/-</h2>
            </div>
            <div class="buttons-container"><button class="btn btn-warning w-50" routerLink="/place-order/{
                <button (click)="onAddToCart(product?._id)" class="btn btn-primary w-50">Add To Cart</butt
            </div>
        </div>
    </div>
</div>
```

Fig 1.6 Product Listing Page Code

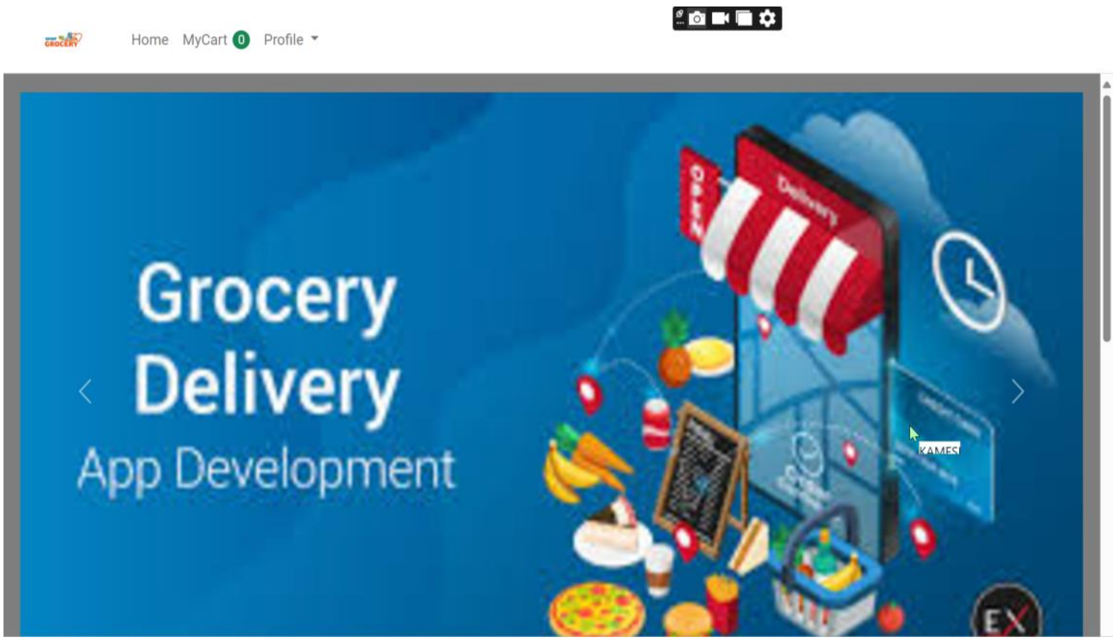
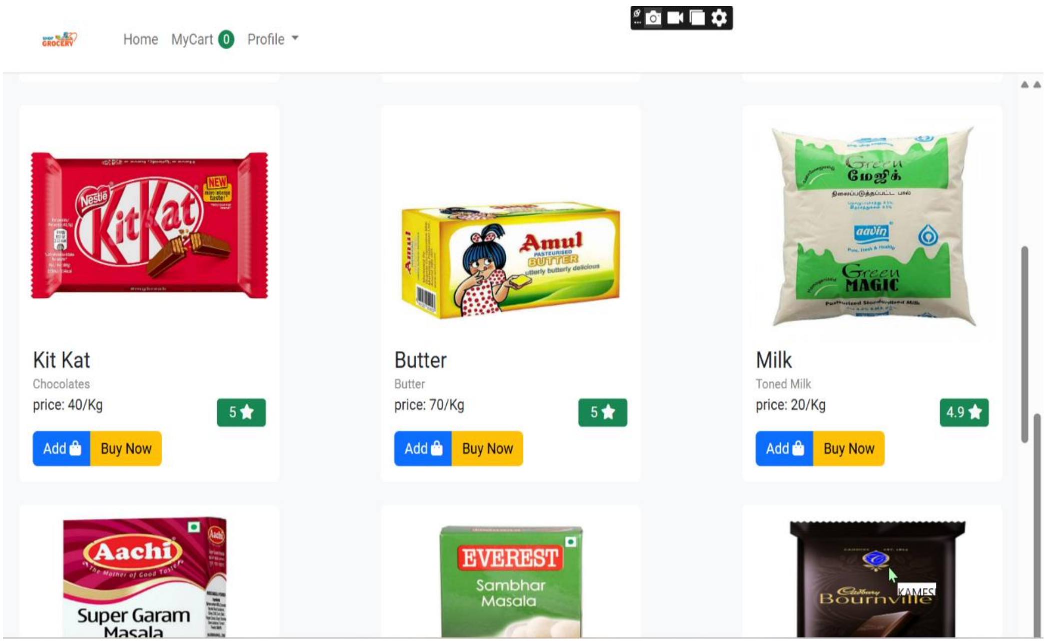


Fig 1.7 a. Product Listing Page

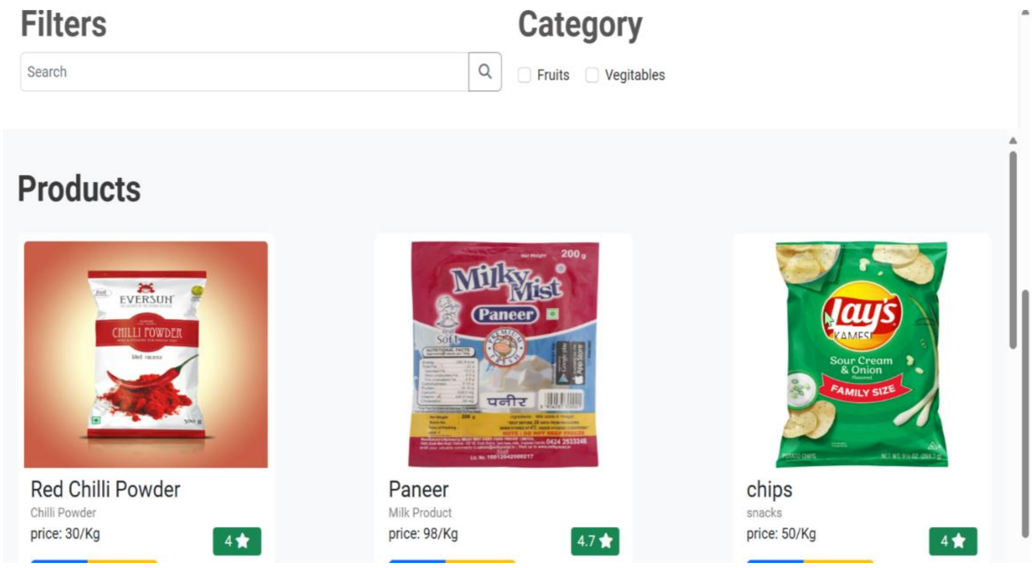


Kit Kat
Chocolates
price: 40/Kg
5 ★
Add | Buy Now

Butter
Butter
price: 70/Kg
5 ★
Add | Buy Now

Milk
Toned Milk
price: 20/Kg
4.9 ★
Add | Buy Now

Fig 1.7 b. Product Listing Page


Filters

Search

Category

Fruits
Vegitables

Products

Red Chilli Powder
Chilli Powder
price: 30/Kg
4 ★

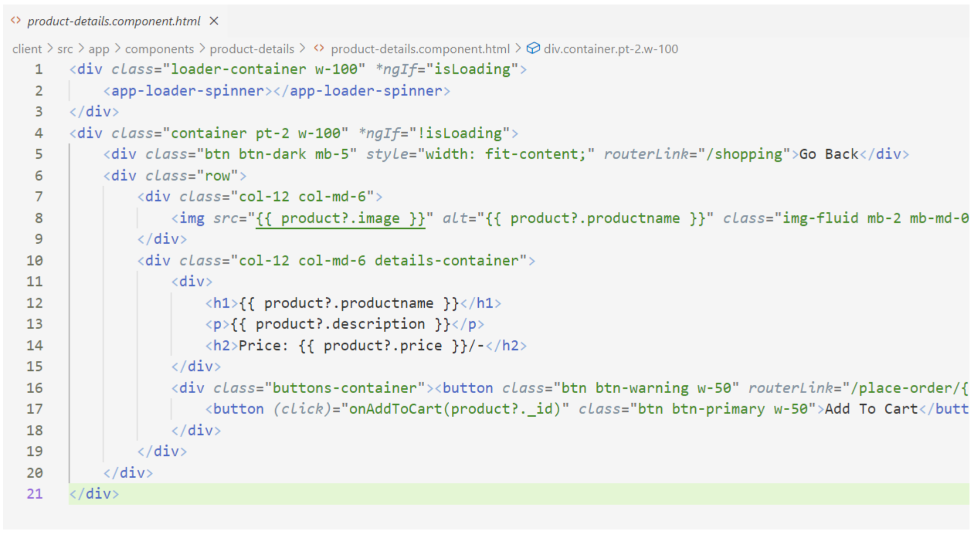
Paneer
Milk Product
price: 98/Kg
4.7 ★

chips
snacks
price: 50/Kg
4 ★

Fig 1.7 c. Product Listing Page

5.3. 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.



product-details.component.html

client > src > app > components > product-details > product-details.component.html > div.container.pt-2.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
    <app-loader-spinner></app-loader-spinner>
</div>
<div class="container pt-2 w-100" *ngIf="!isLoading">
    <div class="btn btn-dark mb-5" style="width: fit-content;" routerLink="/shopping">Go Back</div>
    <div class="row">
        <div class="col-12 col-md-6">
            <img src="{{ product?.image }}" alt="{{ product?.productname }}" class="img-fluid mb-2 mb-md-0
        </div>
        <div class="col-12 col-md-6 details-container">
            <div>
                <h1>{{ product?.productname }}</h1>
                <p>{{ product?.description }}</p>
                <h2>Price: {{ product?.price }}/-</h2>
            </div>
            <div class="buttons-container"><button class="btn btn-warning w-50" routerLink="/place-order/{
                <button (click)="onAddToCart(product?._id)" class="btn btn-primary w-50">Add To Cart</butt
            </div>
        </div>
    </div>
</div>
```

Fig 1.8 Product Details Page Code

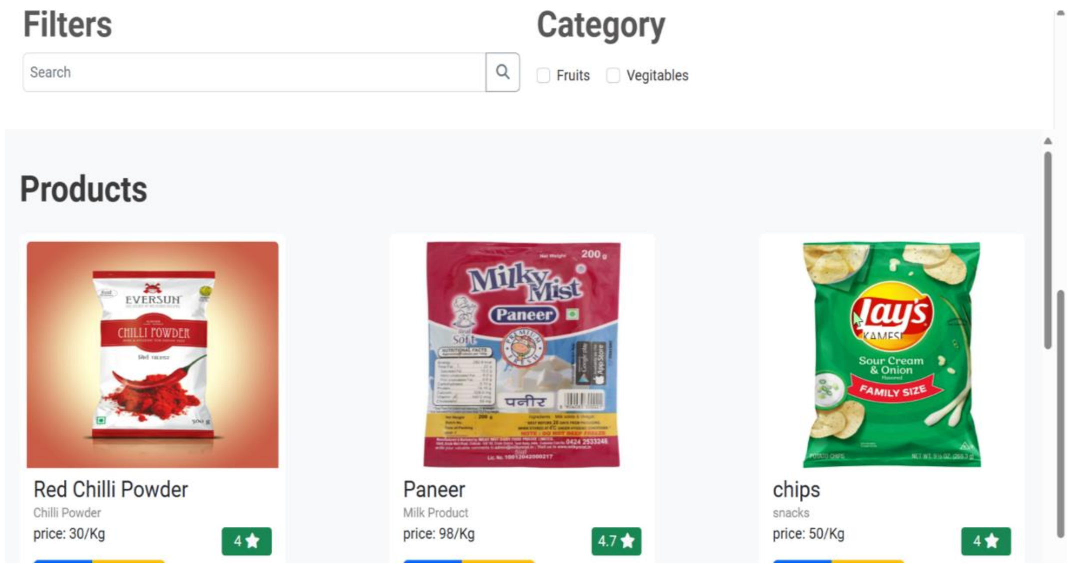
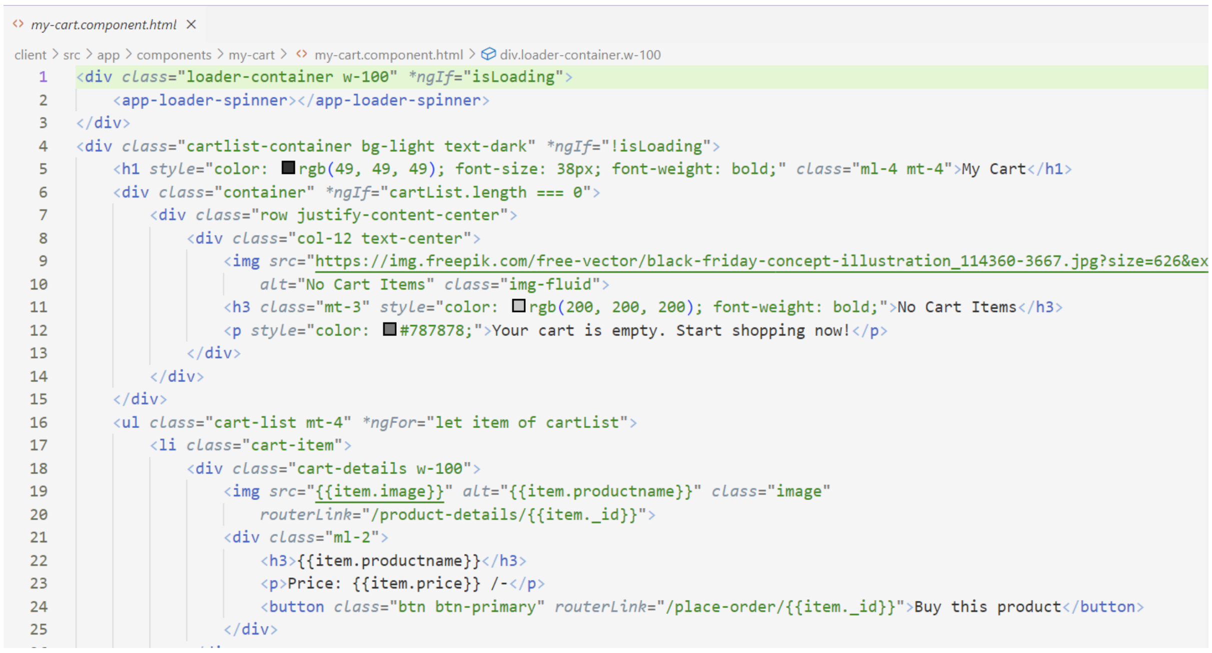


Fig 1.9 Product Details Page

5.4. Cart Page

Features:

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



my-cart.component.html

client > src > app > components > my-cart > my-cart.component.html > div.loader-container.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
    <app-loader-spinner></app-loader-spinner>
</div>
<div class="cartlist-container bg-light text-dark" *ngIf="!isLoading">
    <h1 style="color: rgb(49, 49, 49); font-size: 38px; font-weight: bold;" class="ml-4 mt-4">My Cart</h1>
    <div class="container" *ngIf="cartList.length === 0">
        <div class="row justify-content-center">
            <div class="col-12 text-center">
                <img src="https://img.freepik.com/free-vector/black-friday-concept-illustration_114360-3667.jpg?size=626&ex
                    alt="No Cart Items" class="img-fluid">
                <h3 class="mt-3" style="color: rgb(200, 200, 200); font-weight: bold;">No Cart Items</h3>
                <p style="color: #787878;">Your cart is empty. Start shopping now!</p>
            </div>
        </div>
    </div>
    <ul class="cart-list mt-4" *ngFor="let item of cartList">
        <li class="cart-item">
            <div class="cart-details w-100">
                <img src="{{item.image}}" alt="{{item.productname}}" class="image"
                    routerLink="/product-details/{{item._id}}">
                <div class="ml-2">
                    <h3>{{item.productname}}</h3>
                    <p>Price: {{item.price}} /-</p>
                    <button class="btn btn-primary" routerLink="/place-order/{{item._id}}">Buy this product</button>
                </div>
```

Fig 1.10 Cart Page Code

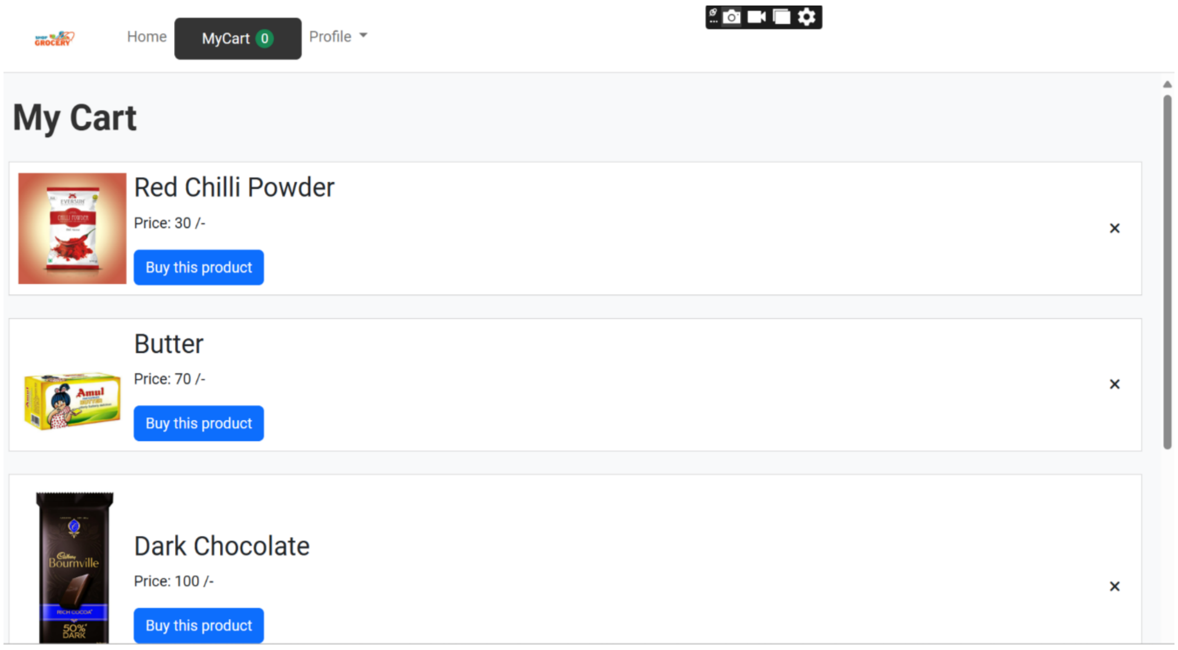
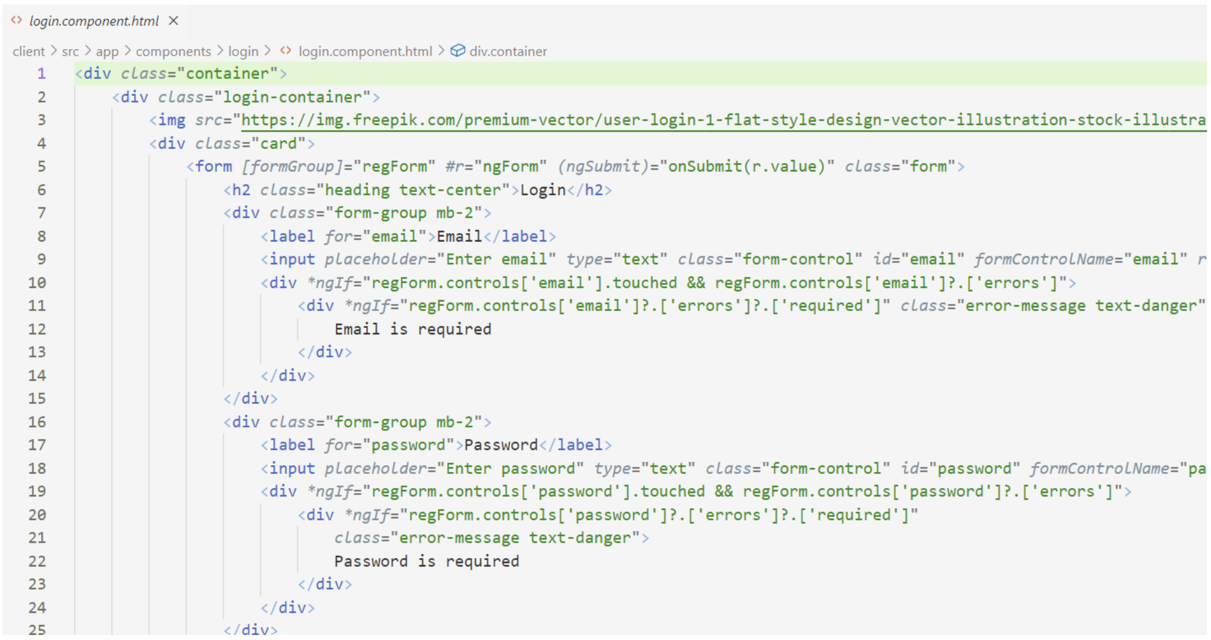


Fig 1.11 Cart Page

5.5. Login/Signup Page

- **Features:**

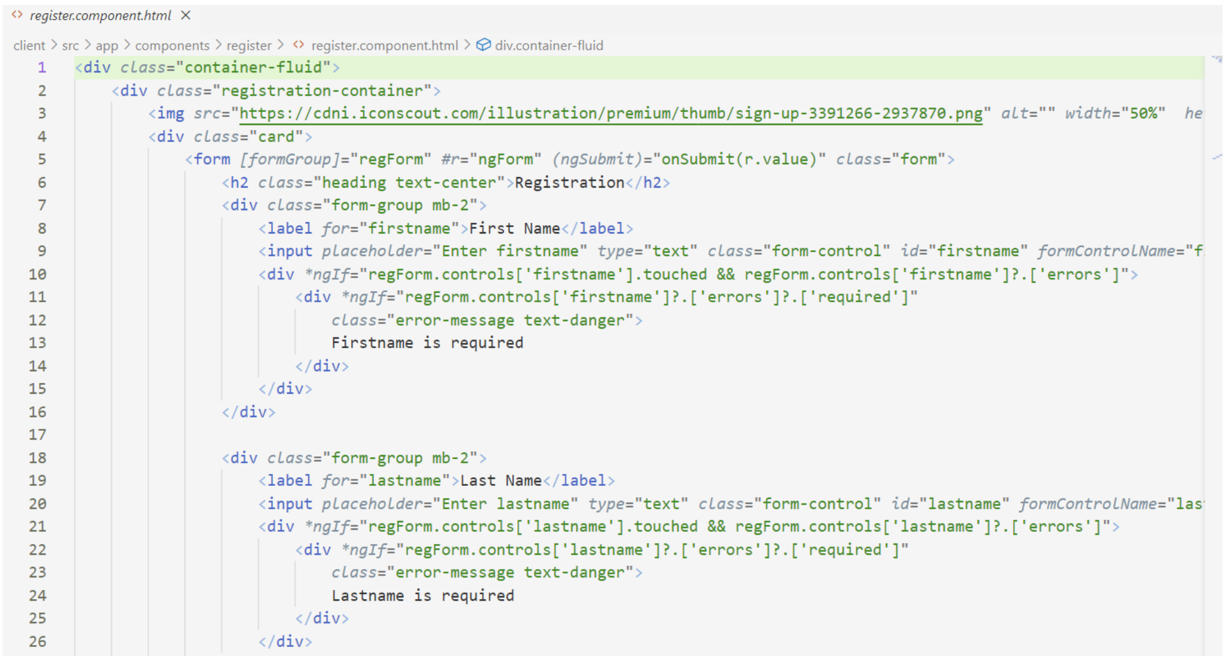
  - **Login Form:** Contains fields for email and password, with a "Login" button.
  - **Signup Form:** New users can sign up with name, email, and password fields, along with a "Sign Up" button.
  - **Validation:** Both forms have validation for required fields and proper format (e.g., email).
  - **Forgot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


login.component.html ×

client > src > app > components > login > login.component.html > div.container

```
<div class="container">
    <div class="login-container">
        <img src="https://img.freepik.com/premium-vector/user-login-1-flat-style-design-vector-illustration-stock-illustra
        <div class="card">
            <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="onSubmit(r.value)" class="form">
                <h2 class="heading text-center">Login</h2>
                <div class="form-group mb-2">
                    <label for="email">Email</label>
                    <input placeholder="Enter email" type="text" class="form-control" id="email" formControlName="email" r
                    <div *ngIf="regForm.controls['email'].touched && regForm.controls['email']?.['errors']">
                        <div *ngIf="regForm.controls['email']?.['errors']?.['required']" class="error-message text-danger"
                            Email is required
                        </div>
                    </div>
                </div>
                <div class="form-group mb-2">
                    <label for="password">Password</label>
                    <input placeholder="Enter password" type="text" class="form-control" id="password" formControlName="pa
                    <div *ngIf="regForm.controls['password'].touched && regForm.controls['password']?.['errors']">
                        <div *ngIf="regForm.controls['password']?.['errors']?.['required']"
                            class="error-message text-danger">
                            Password is required
                        </div>
                    </div>
                </div>
```

Fig 1.12 a. Login/Signup Page Code



register.component.html

client > src > app > components > register > register.component.html > div.container-fluid

```
<div class="container-fluid">
    <div class="registration-container">
        <img src="https://cdni.iconscout.com/illustration/premium/thumb/sign-up-3391266-2937870.png" alt="" width="50%"  he
        <div class="card">
            <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="onSubmit(r.value)" class="form">
                <h2 class="heading text-center">Registration</h2>
                <div class="form-group mb-2">
                    <label for="firstname">First Name</label>
                    <input placeholder="Enter firstname" type="text" class="form-control" id="firstname" formControlName="f
                    <div *ngIf="regForm.controls['firstname'].touched && regForm.controls['firstname']?.['errors']">
                        <div *ngIf="regForm.controls['firstname']?.['errors']?.['required']"
                            class="error-message text-danger">
                            Firstname is required
                        </div>
                    </div>
                </div>

                <div class="form-group mb-2">
                    <label for="lastname">Last Name</label>
                    <input placeholder="Enter lastname" type="text" class="form-control" id="lastname" formControlName="las
                    <div *ngIf="regForm.controls['lastname'].touched && regForm.controls['lastname']?.['errors']">
                        <div *ngIf="regForm.controls['lastname']?.['errors']?.['required']"
                            class="error-message text-danger">
                            Lastname is required
                        </div>
                    </div>
```

Fig 1.12 b. Login/Signup Page Code

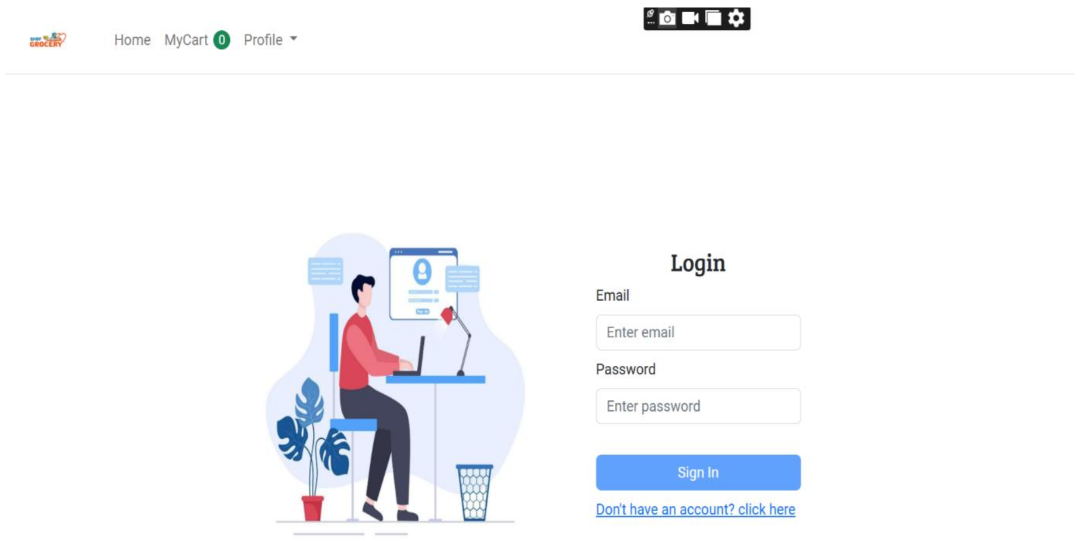
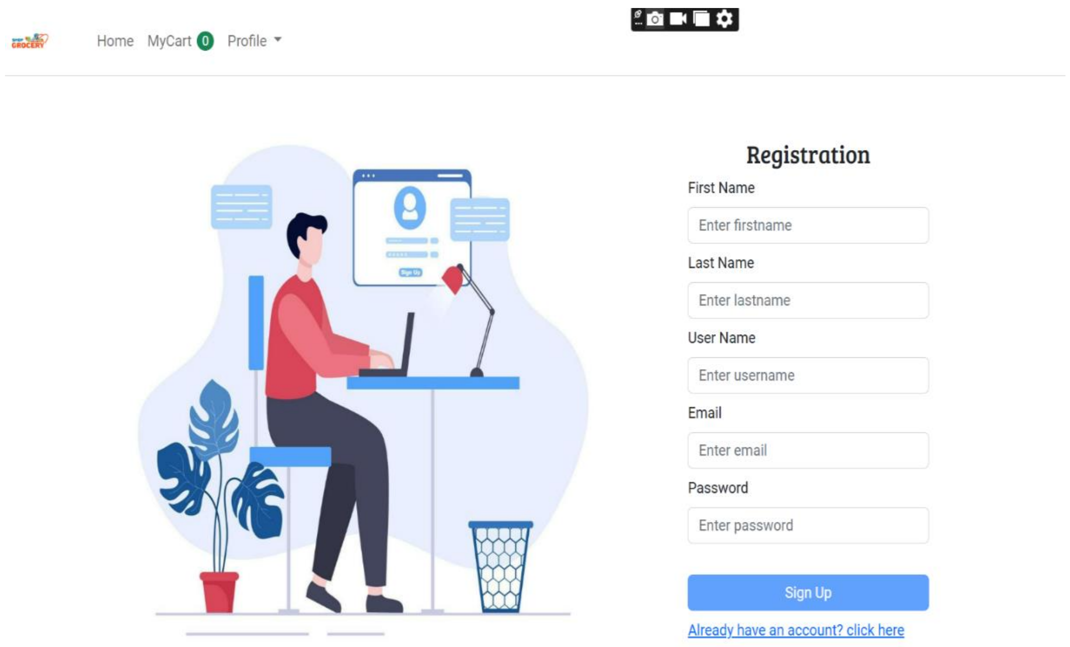


Fig 1.13 a. Login/Signup Page



Registration

First Name

Enter firstname

Last Name

Enter lastname

User Name

Enter username

Email

Enter email

Password

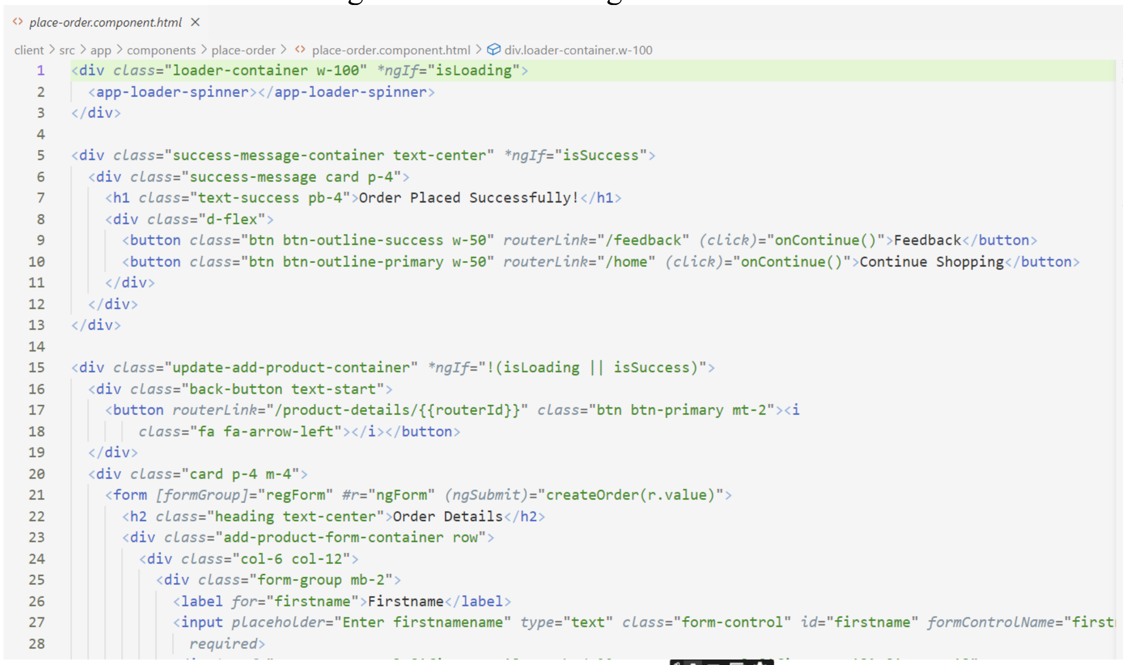
Enter password

Fig 1.13 b. Login/Signup Page

5.6. 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.

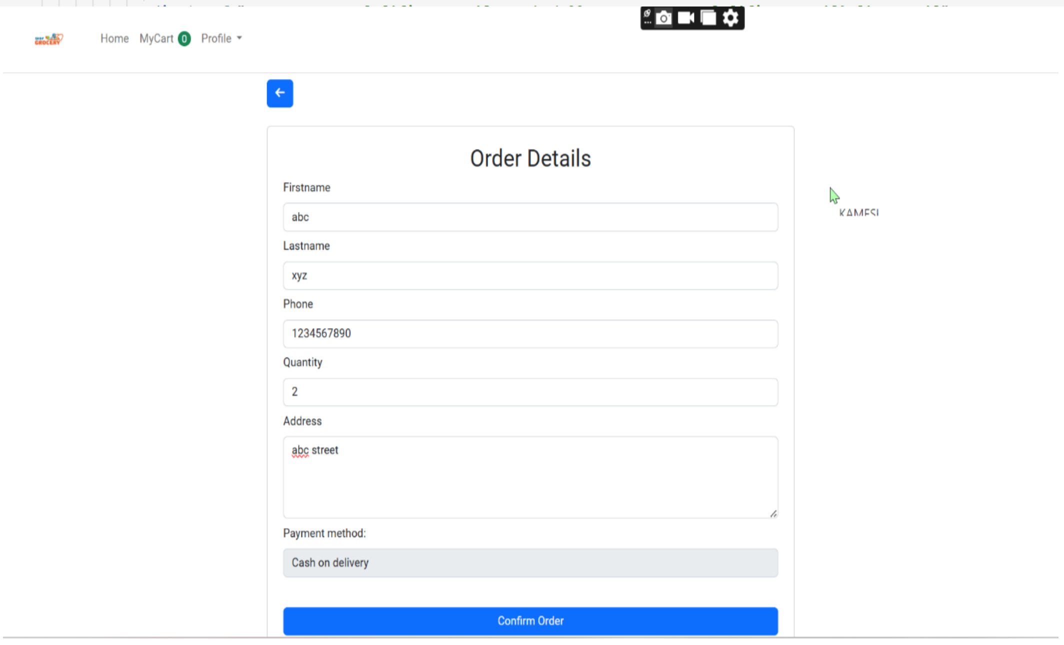
Fig 1.14. Checkout Page Code



place-order.component.html

client > src > app > components > place-order > place-order.component.html > div.loader-container.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
  <app-loader-spinner></app-loader-spinner>
</div>

<div class="success-message-container text-center" *ngIf="isSuccess">
  <div class="success-message card p-4">
    <h1 class="text-success pb-4">Order Placed Successfully!</h1>
    <div class="d-flex">
      <button class="btn btn-outline-success w-50" routerLink="/feedback" (click)="onContinue()">Feedback</button>
      <button class="btn btn-outline-primary w-50" routerLink="/home" (click)="onContinue()">Continue Shopping</button>
    </div>
  </div>
</div>

<div class="update-add-product-container" *ngIf="!(isLoading || isSuccess)">
  <div class="back-button text-start">
    <button routerLink="/product-details/{{routerId}}" class="btn btn-primary mt-2"><i
        class="fa fa-arrow-left"></i></button>
  </div>
  <div class="card p-4 m-4">
    <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="createOrder(r.value)">
      <h2 class="heading text-center">Order Details</h2>
      <div class="add-product-form-container row">
        <div class="col-6 col-12">
          <div class="form-group mb-2">
            <label for="firstname">Firstname</label>
            <input placeholder="Enter firstnamename" type="text" class="form-control" id="firstname" formControlName="first
              required>
```



Order Details

Firstname
abc

Lastname
xyz

Phone
1234567890

Quantity
2

Address
abc street

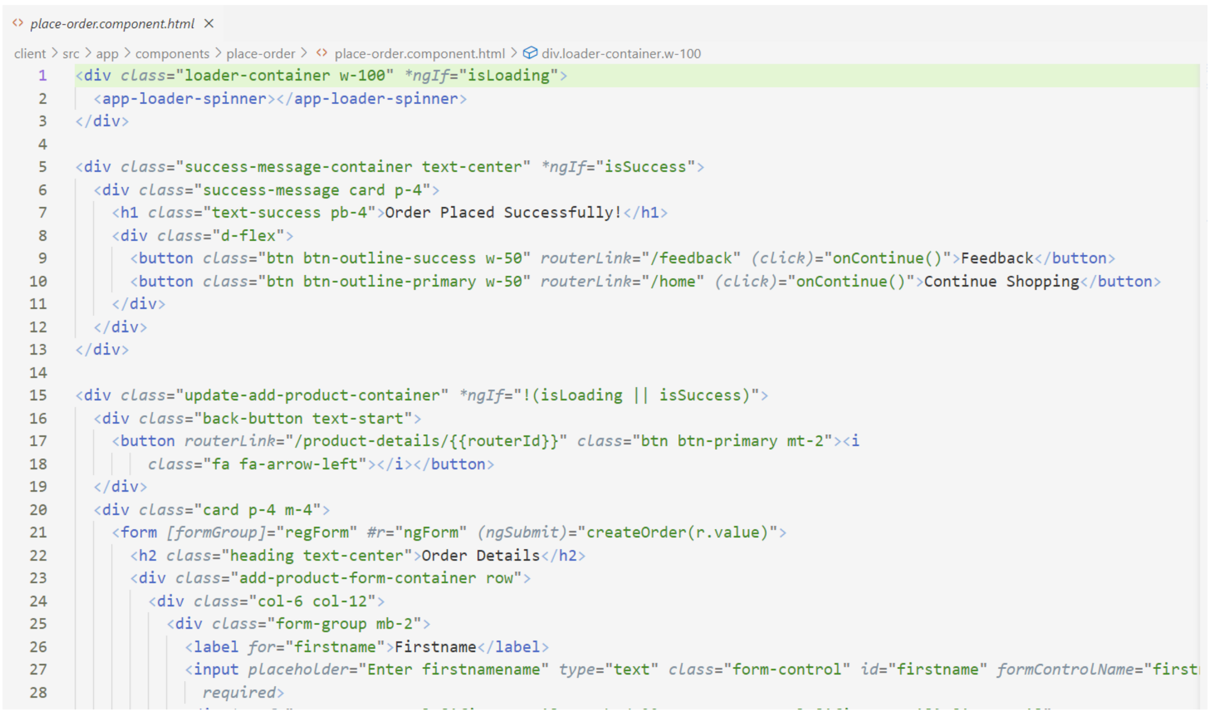
Payment method:
Cash on delivery

Confirm Order

Fig 1.15. Checkout Page

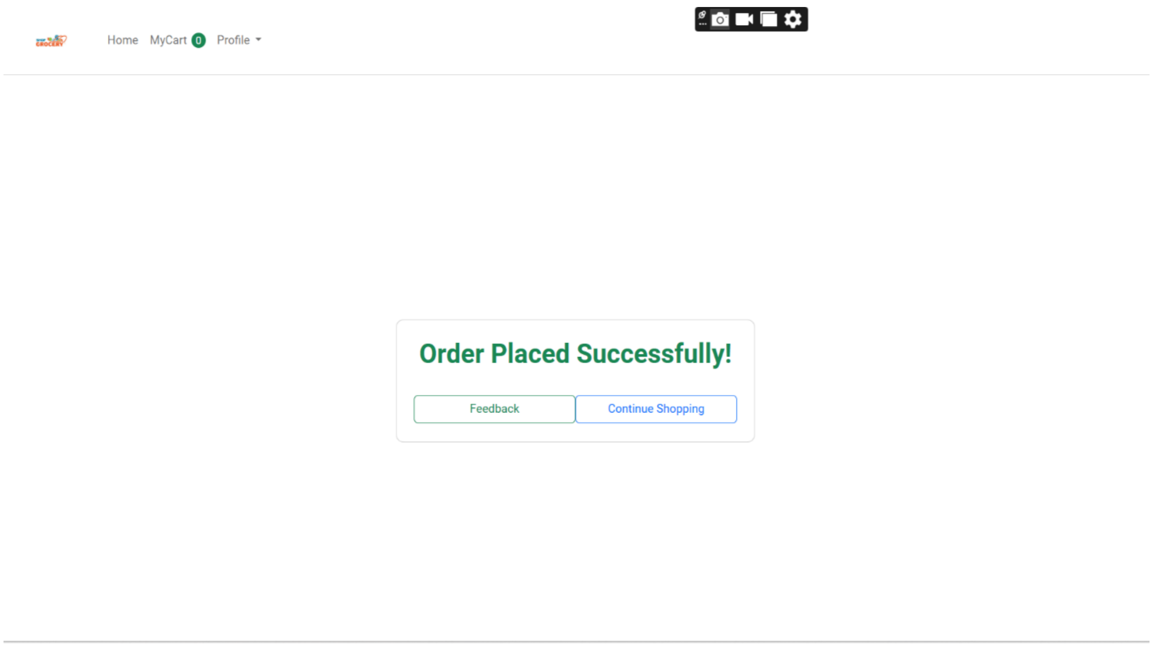
5.7. Confirmation Page (Order Success)

- **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.


place-order.component.html
client > src > app > components > place-order > place-order.component.html > div.loader-container.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
  <app-loader-spinner></app-loader-spinner>
</div>

<div class="success-message-container text-center" *ngIf="isSuccess">
  <div class="success-message card p-4">
    <h1 class="text-success pb-4">Order Placed Successfully!</h1>
    <div class="d-flex">
      <button class="btn btn-outline-success w-50" routerLink="/feedback" (click)="onContinue()">Feedback</button>
      <button class="btn btn-outline-primary w-50" routerLink="/home" (click)="onContinue()">Continue Shopping</button>
    </div>
  </div>
</div>

<div class="update-add-product-container" *ngIf="!(isLoading || isSuccess)">
  <div class="back-button text-start">
    <button routerLink="/product-details/{{routerId}}" class="btn btn-primary mt-2"><i
        class="fa fa-arrow-left"></i></button>
  </div>
  <div class="card p-4 m-4">
    <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="createOrder(r.value)">
      <h2 class="heading text-center">Order Details</h2>
      <div class="add-product-form-container row">
        <div class="col-6 col-12">
          <div class="form-group mb-2">
            <label for="firstname">Firstname</label>
            <input placeholder="Enter firstnamename" type="text" class="form-control" id="firstname" formControlName="first
              required>
```

Fig 1.16. Confirmation Page Code



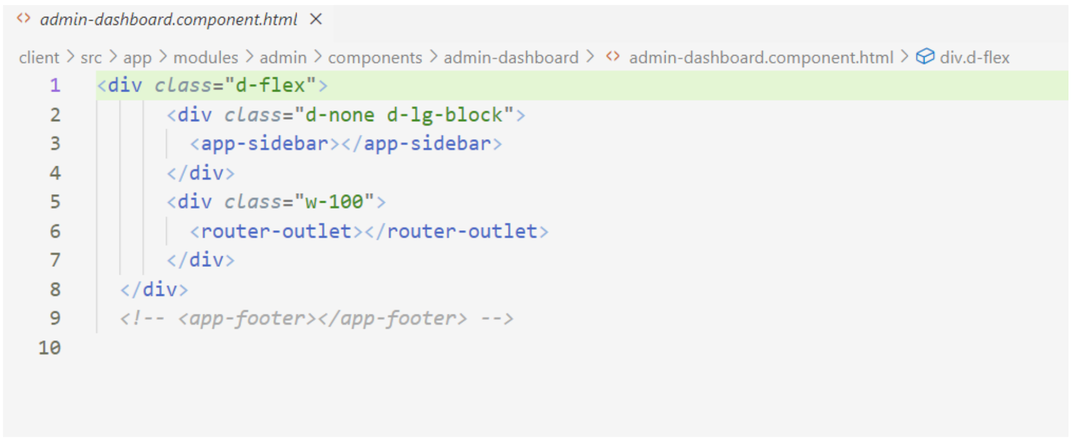
Order Placed Successfully!

Fig 1.17. Confirmation Page

6. ROLES MANAGEMENT

6.1. 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.

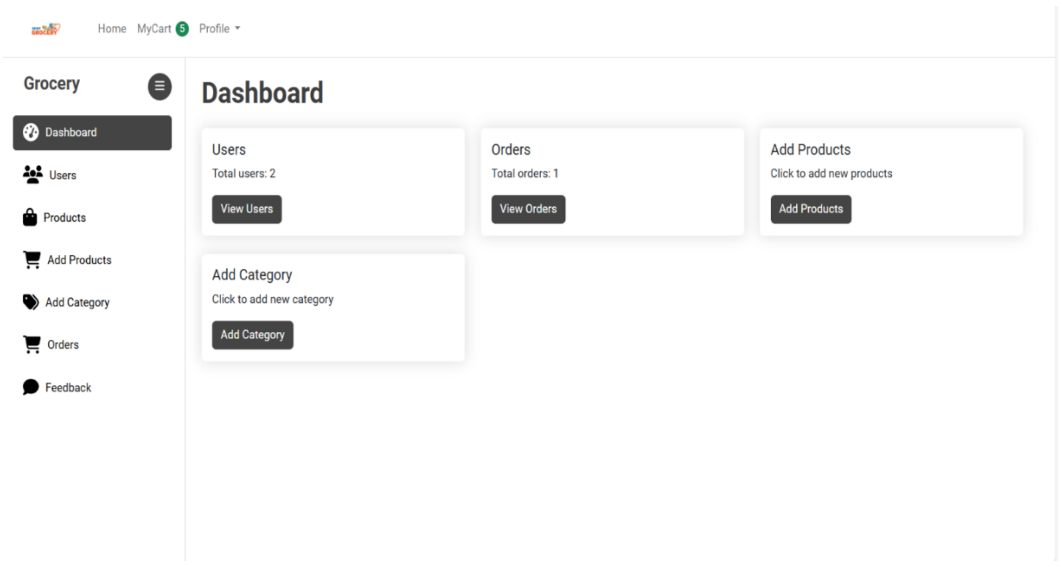


admin-dashboard.component.html

client > src > app > modules > admin > components > admin-dashboard > admin-dashboard.component.html > div.d-flex

```html
<div class="d-flex">
    <div class="d-none d-lg-block">
      <app-sidebar></app-sidebar>
    </div>
    <div class="w-100">
      <router-outlet></router-outlet>
    </div>
  </div>
  <!-- <app-footer></app-footer> -->

```

Fig 1.18. Admin Dashboard Code



Dashboard

Users
Total users: 2

Orders
Total orders: 1

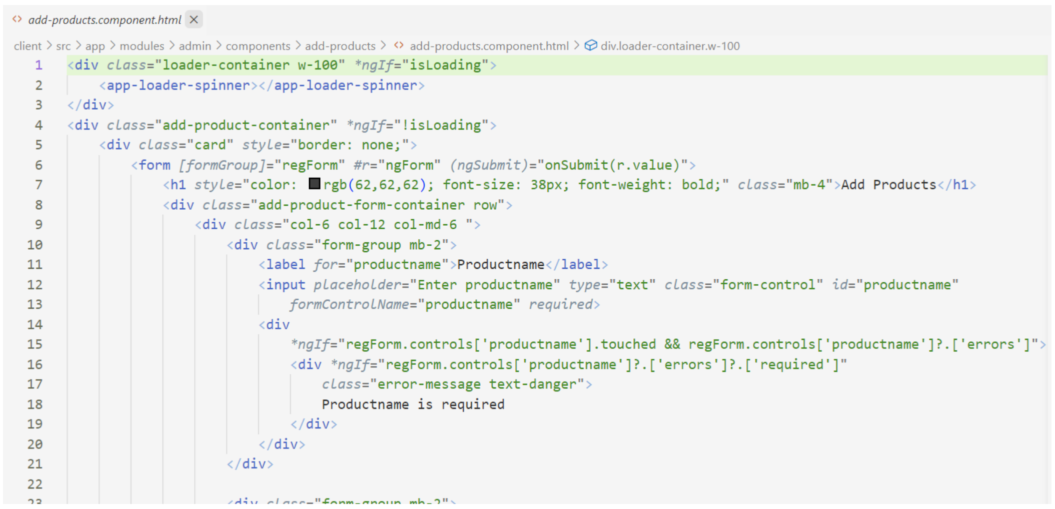
Add Products
Click to add new products

Add Category
Click to add new category

Fig 1.19. Admin Dashboard

6.2. 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 catalog with confirmation prompts to avoid accidental deletions.



add-products.component.html

client > src > app > modules > admin > components > add-products > add-products.component.html > div.loader-container.w-100

```
<div class="loader-container w-100" *ngIf="isLoading">
    <app-loader-spinner></app-loader-spinner>
</div>
<div class="add-product-container" *ngIf="!isLoading">
    <div class="card" style="border: none;">
        <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="onSubmit(r.value)">
            <h1 style="color: ■rgb(62,62,62); font-size: 38px; font-weight: bold;" class="mb-4">Add Products</h1>
            <div class="add-product-form-container row">
                <div class="col-6 col-12 col-md-6 ">
                    <div class="form-group mb-2">
                        <label for="productname">Productname</label>
                        <input placeholder="Enter productname" type="text" class="form-control" id="productname"
                            formControlName="productname" required>
                        <div
                            *ngIf="regForm.controls['productname'].touched && regForm.controls['productname']?.['errors']">
                            <div *ngIf="regForm.controls['productname']?.['errors']?.['required']"
                                class="error-message text-danger">
                                Productname is required
                            </div>
                        </div>
                    </div>

                    <div class="form-group mb-2">
```

Fig 1.20. Product Management Code

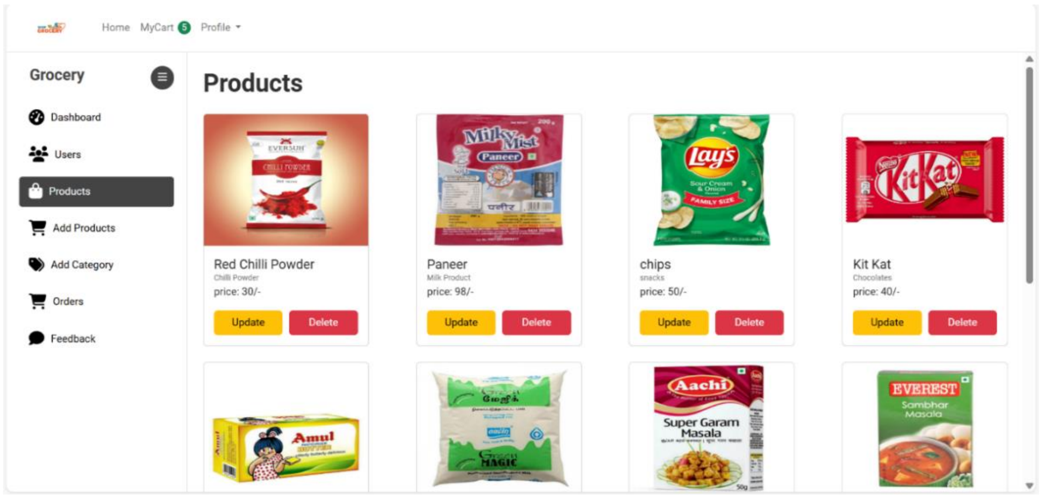
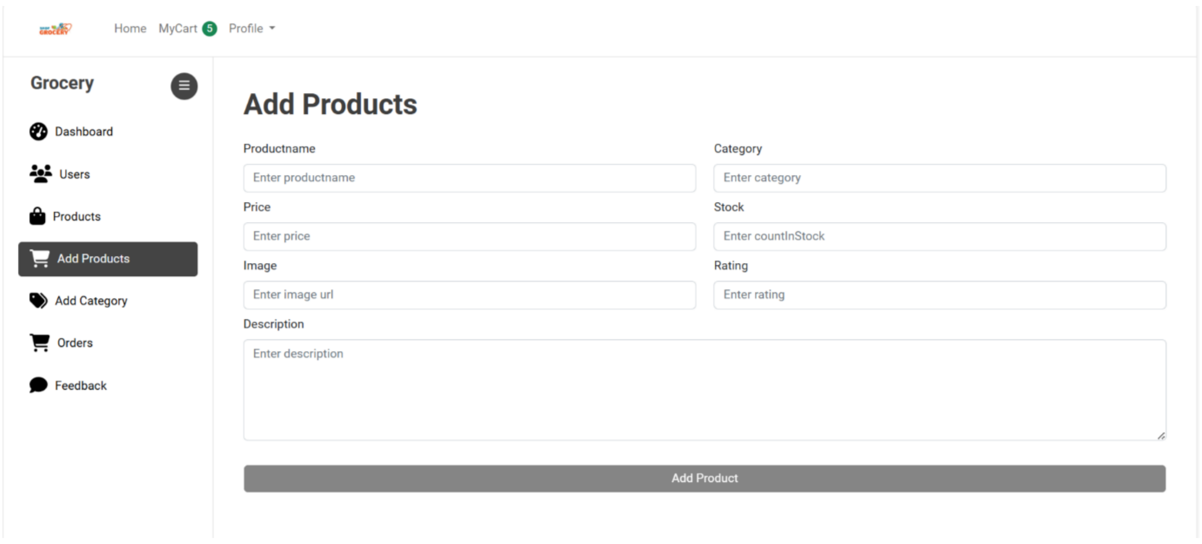


Fig 1.21. a. Product Management


Home MyCart 5 Profile

Grocery

- Dashboard
- Users
- Products
- Add Products
- Add Category
- Orders
- Feedback

Add Products

Productname: Enter productname	Category: Enter category
Price: Enter price	Stock: Enter countInStock
Image: Enter image url	Rating: Enter rating

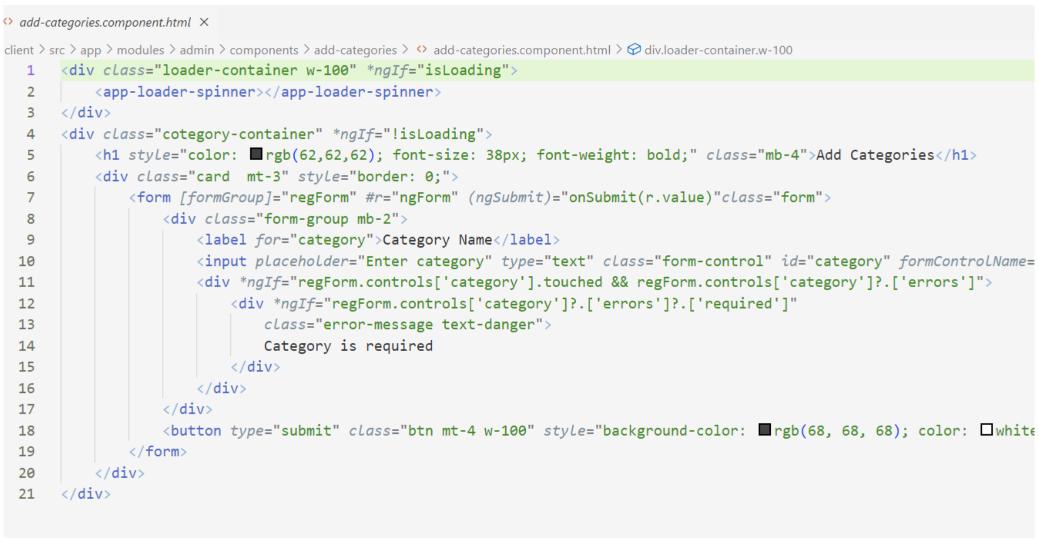
Description: Enter description

Add Product

Fig 1.21. b. Product Management

6.3. 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.


add-categories.component.html

client > src > app > modules > admin > components > add-categories > add-categories.component.html > div.loader-container.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
    <app-loader-spinner></app-loader-spinner>
</div>
<div class="cotegory-container" *ngIf="!isLoading">
    <h1 style="color: rgb(62,62,62); font-size: 38px; font-weight: bold;" class="mb-4">Add Categories</h1>
    <div class="card  mt-3" style="border: 0;">
        <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="onSubmit(r.value)"class="form">
            <div class="form-group mb-2">
                <label for="category">Category Name</label>
                <input placeholder="Enter category" type="text" class="form-control" id="category" formControlName=
                <div *ngIf="regForm.controls['category'].touched && regForm.controls['category']?.['errors']">
                    <div *ngIf="regForm.controls['category']?.['errors']?.['required']"
                        class="error-message text-danger">
                        Category is required
                    </div>
                </div>
            </div>
            <button type="submit" class="btn mt-4 w-100" style="background-color: rgb(68, 68, 68); color: white
        </form>
    </div>
</div>
```

Fig 1.22. Category Management Code

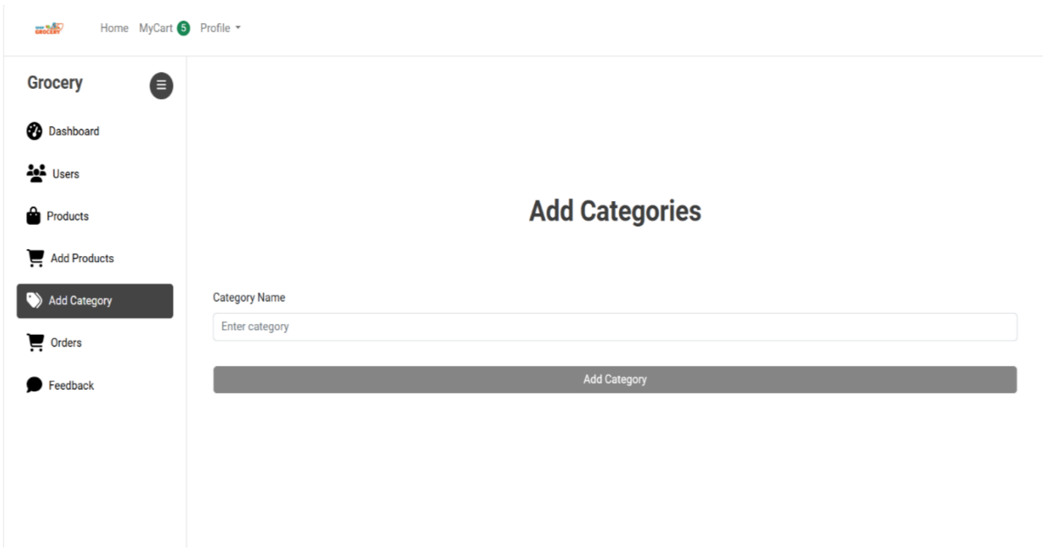
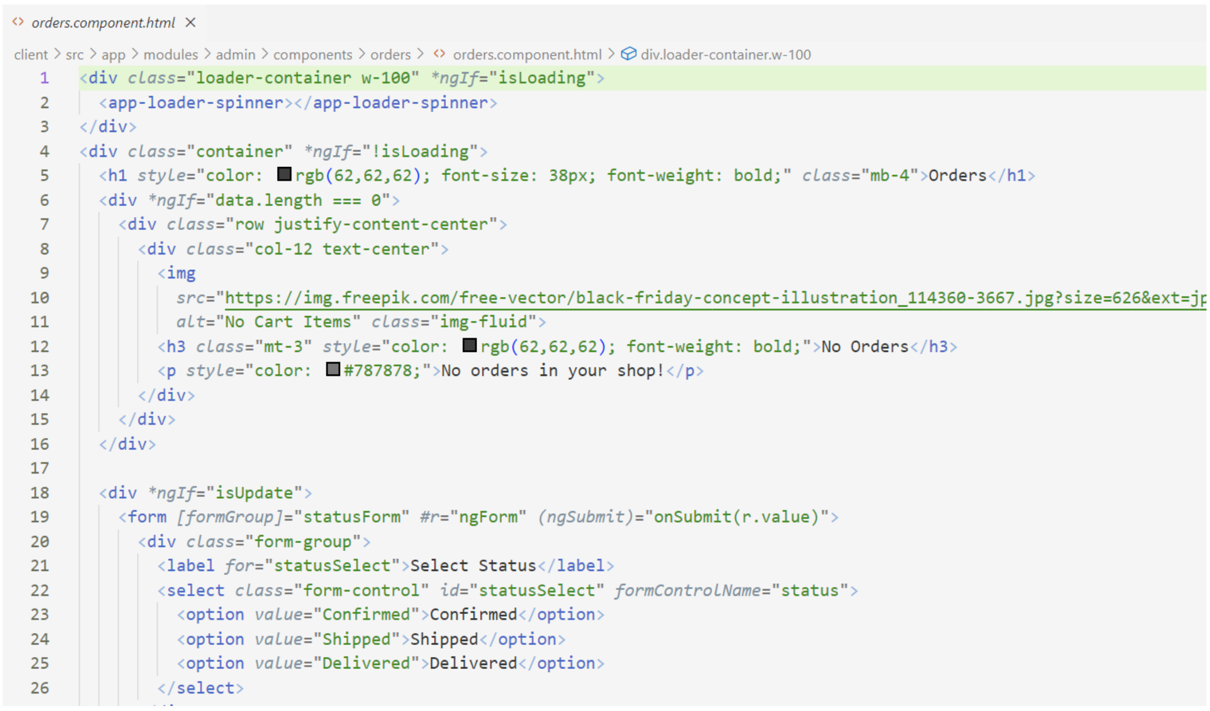


Fig 1.23. Category Management

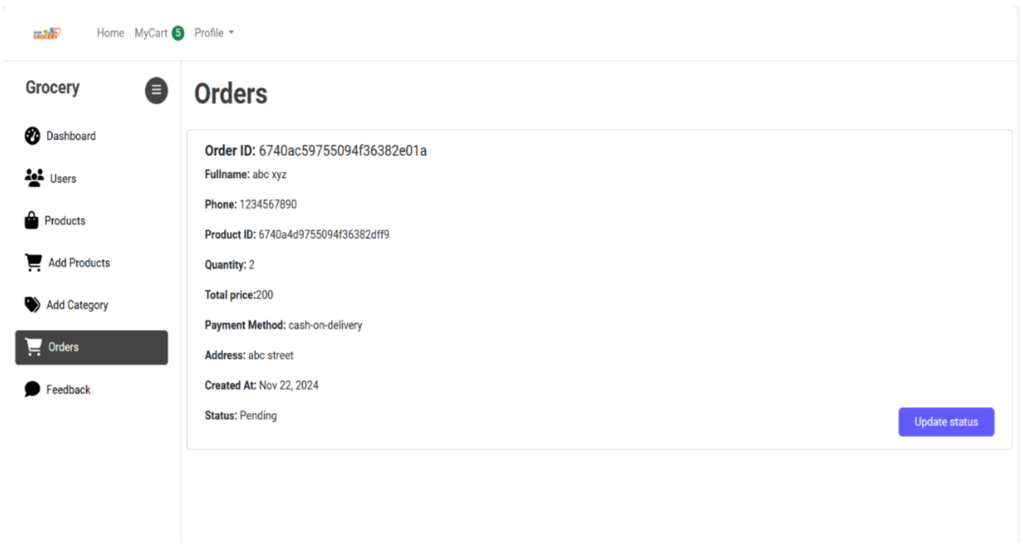
6.4. Order Management

- **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.component.html
client > src > app > modules > admin > components > orders > orders.component.html > div.loader-container.w-100

```html
<div class="loader-container w-100" *ngIf="isLoading">
  <app-loader-spinner></app-loader-spinner>
</div>
<div class="container" *ngIf="!isLoading">
  <h1 style="color: rgb(62,62,62); font-size: 38px; font-weight: bold;" class="mb-4">Orders</h1>
  <div *ngIf="data.length === 0">
    <div class="row justify-content-center">
      <div class="col-12 text-center">
        <img
          src="https://img.freepik.com/free-vector/black-friday-concept-illustration_114360-3667.jpg?size=626&ext=jp
          alt="No Cart Items" class="img-fluid">
        <h3 class="mt-3" style="color: rgb(62,62,62); font-weight: bold;">No Orders</h3>
        <p style="color: #787878;">No orders in your shop!</p>
      </div>
    </div>
  </div>

  <div *ngIf="isUpdate">
    <form [formGroup]="statusForm" #r="ngForm" (ngSubmit)="onSubmit(r.value)">
      <div class="form-group">
        <label for="statusSelect">Select Status</label>
        <select class="form-control" id="statusSelect" formControlName="status">
          <option value="Confirmed">Confirmed</option>
          <option value="Shipped">Shipped</option>
          <option value="Delivered">Delivered</option>
        </select>
```

Fig 1.24. Order Management Code



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

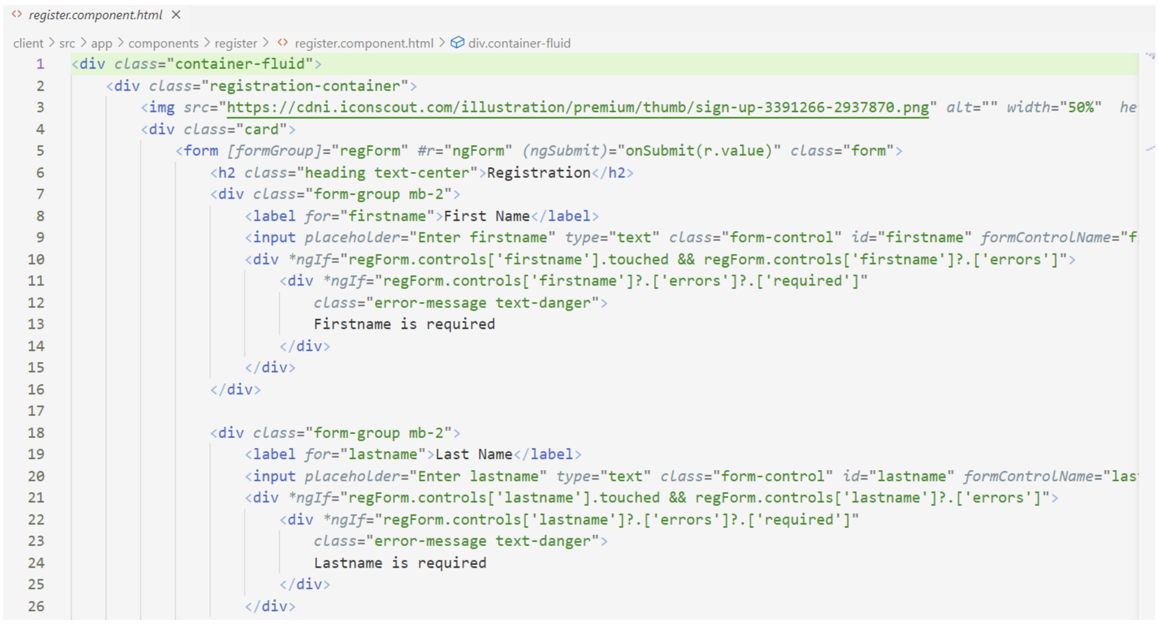
Created At: Nov 22, 2024

Status: Pending

Fig 1.25. Order Management

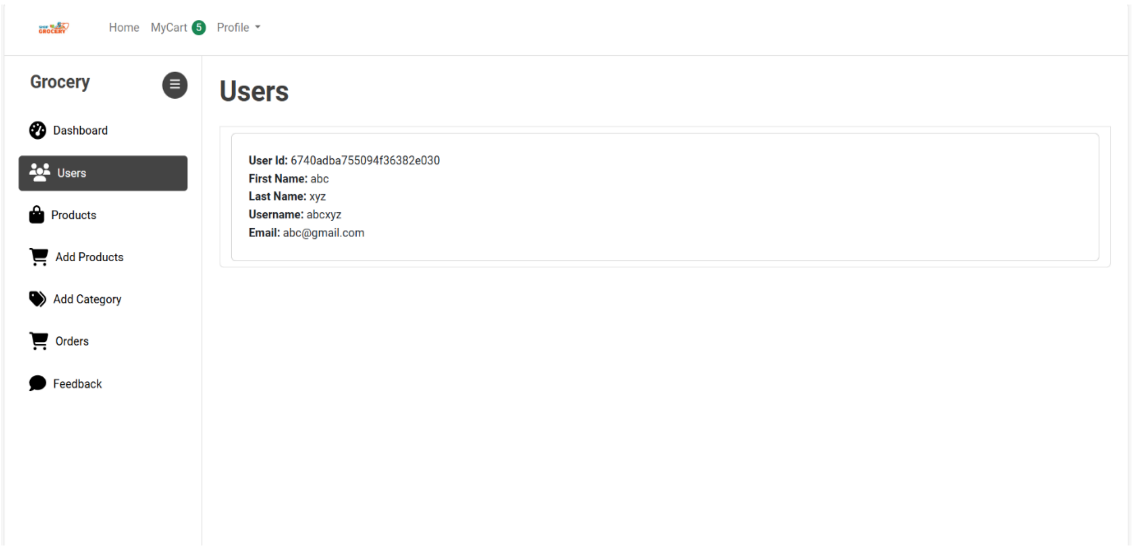
6.5. User Management

- **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 accounts.


register.component.html
client > src > app > components > register > register.component.html > div.container-fluid

```html
<div class="container-fluid">
    <div class="registration-container">
        <img src="https://cdni.iconscout.com/illustration/premium/thumb/sign-up-3391266-2937870.png" alt="" width="50%"  he
        <div class="card">
            <form [formGroup]="regForm" #r="ngForm" (ngSubmit)="onSubmit(r.value)" class="form">
                <h2 class="heading text-center">Registration</h2>
                <div class="form-group mb-2">
                    <label for="firstname">First Name</label>
                    <input placeholder="Enter firstname" type="text" class="form-control" id="firstname" formControlName="f
                    <div *ngIf="regForm.controls['firstname'].touched && regForm.controls['firstname']?.['errors']">
                        <div *ngIf="regForm.controls['firstname']?.['errors']?.['required']"
                            class="error-message text-danger">
                            Firstname is required
                        </div>
                    </div>
                </div>

                <div class="form-group mb-2">
                    <label for="lastname">Last Name</label>
                    <input placeholder="Enter lastname" type="text" class="form-control" id="lastname" formControlName="las
                    <div *ngIf="regForm.controls['lastname'].touched && regForm.controls['lastname']?.['errors']">
                        <div *ngIf="regForm.controls['lastname']?.['errors']?.['required']"
                            class="error-message text-danger">
                            Lastname is required
                        </div>
                    </div>
```

Fig 1.26. User Management Code



Users

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

Fig 1.27. User Management

7. 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.

8. REFERENCES

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
10. **Full-Stack React, TypeScript, and Node" by David Choi**
11. **MERN Quick Start Guide" by Eddy Wilson Iriarte Koroliova**
12. **Learning React: Modern Patterns for Developing React Apps" by Alex Banks and Eve Porcello**
13. **Node.js Design Patterns" by Mario Casciaro**
14. **MongoDB: The Definitive Guide" by Kristina Chodorow and Michael Dirolf**
15. **Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node" by Vasan Subramanian**