Grocery-Web Application Development with MERN

Project Documentation

1. Introduction

- **Project Title:** Grocery Webapp
- **Team Members:** Abhilash.P(Team leader),Sidharth.C(Team Member),Vignesh.T(Team Member), Ramanathan J(Team Member).

2. Project Overview

- **Purpose:**

  This project is a grocery web application built with the MERN stack (MongoDB, Express, React, Node.js), designed to make online grocery shopping easy, fast, and user-friendly. The app allows users to browse a wide selection of grocery items, add products to their cart, manage their orders, and securely complete purchases. Key features include product search, shopping cart functionality, order tracking, user authentication, and integrated payment options. The app offers a streamlined and convenient shopping experience, providing users with the flexibility to shop for groceries from anywhere.

- **Features:**

  Product Browsing & Search: Users can browse grocery items by categories or search for specific products.

  Shopping Cart: Add, update, or remove items in a virtual cart with a real-time view of total cost.

  User Authentication: Secure sign-up, login, and profile management for personalized shopping.

  Order Management: View and track current orders and review past purchase history.

  Payment Integration: Secure and reliable payment options for a seamless checkout experience.

  Inventory Management (Admin): Admins can add, update, or remove products and monitor stock levels.

  Offers & Discounts: Apply discount codes or view promotional offers on select items.

  Responsive Design: Optimized for both mobile and desktop, providing a smooth experience on any device.

Push Notifications: Notify users about order updates, new offers, or reminders.

Ratings & Reviews: Users can leave feedback and rate products to help others make informed choices.

3. Architecture

- **Frontend:**

  User Interface (UI): Built with React for a dynamic, responsive interface. The frontend handles page layouts, navigation, product listings, shopping cart views, and user interactions.
  State Management: Utilizes React hooks or a state management library (e.g., Redux) to manage application-wide state, like user data and shopping cart content.
  API Requests: Communicates with the backend using RESTful APIs, fetching data for product displays, cart updates, and order processing.

- **Backend:**

  RESTful API: Node.js with Express provides a RESTful API to handle requests from the frontend, including user authentication, product data retrieval, cart management, and order processing.
  Business Logic: Manages core functions like validating orders, updating inventory, handling user roles, and applying business rules for discounts or promotions.
  Authentication & Authorization: User registration, login, and role-based access (user vs. admin) are managed through secure authentication, typically with JWT (JSON Web Tokens).

- **Database:**

  Data Storage: MongoDB stores structured documents for users, products, orders, and reviews. Each document contains the necessary fields to support shopping, ordering, and inventory management.
  Data Models: Uses Mongoose (MongoDB ORM for Node.js) to define schemas for collections, ensuring data consistency and implementing relationships between users, orders, and products.
  Inventory & Order Management: Tracks stock levels and records order details, which can be updated by admins.

4. Setup Instructions

- **Prerequisites:**

- Install **Node.js** and **npm** (Node Package Manager).
- Install **MongoDB** and set up a local instance or use a cloud-based MongoDB service (e.g., MongoDB Atlas).
- Optionally, install **Git** for version control.

1. **Clone the Repository**
   git clone https://github.com/your-username/grocery-web-app.git
   cd grocery-web-app

2. **Install Backend Dependencies**
   cd backend
   npm install

3. Install Frontend Dependencies
   cd ../frontend
   npm install

4. **Configure Environment Variables**

   Backend:

- In the backend folder, create a .env file with:

  PORT=5000

  MONGO_URI=your_mongodb_connection_string

  JWT_SECRET=your_jwt_secret

  PAYMENT_SECRET_KEY=your_payment_gateway_key

  Frontend:

- In the frontend folder, create a .env file with:
  REACT_APP_API_URL=http://localhost:5000

5 . Access the App

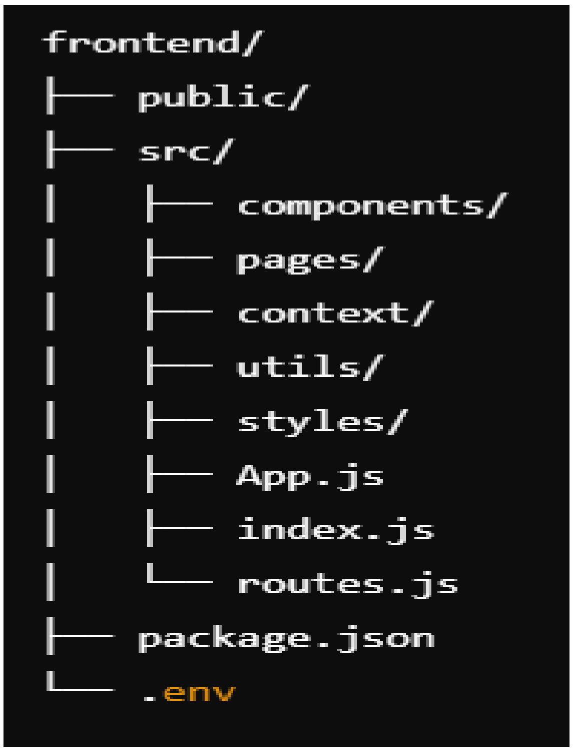
Open your browser and navigate to http://localhost:3000 to access the grocery app.

Your grocery web app is now installed and running!

5. Folder Structure

1. Client (React Frontend)

The React frontend is organized as follows:

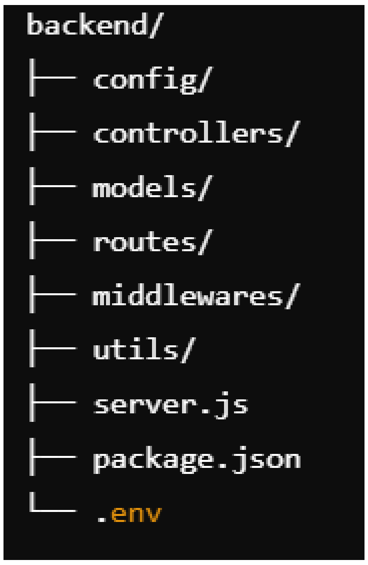


```
frontend/
├── public/
├── src/
│   ├── components/
│   ├── pages/
│   ├── context/
│   ├── utils/
│   ├── styles/
│   ├── App.js
│   ├── index.js
│   └── routes.js
├── package.json
└── .env
```

- The components/ folder contains reusable UI elements like buttons, forms, or headers.
- The pages/ folder includes specific pages, such as HomePage.js, ProductPage.js, and CartPage.js.
- API calls and utilities are abstracted into the utils/ folder.
- The context/ folder manages global states, such as the shopping cart or authentication state.

2. Server:

The Node.js backend is structured as follows:



```
backend/
├── config/
├── controllers/
├── models/
├── routes/
├── middlewares/
├── utils/
├── server.js
├── package.json
└── .env
```

6. Running the Application

Start MongoDB:

Start MongoDB locally or ensure your MongoDB Atlas instance is running.

Start Backend:

From the backend folder

npm start

Start Frontend:

From the frontend folder

npm start

7. API Documentation

1. Authentication Endpoints

POST /api/auth/register

- **Description**: Registers a new user.
- **Request Method**: POST

- **Request Body**:

json

Copy code

```
{
  "name": "John Doe",
  "email": "john.doe@example.com",
  "password": "password123"
}
```

    - name: (string) The user's full name.
    - email: (string) The user's email (must be unique).
    - password: (string) The user's password (min. 6 characters).

- **Response**:

200 OK:

json

Copy code

```
{
  "message": "User registered successfully.",
  "user": {
    "id": "123456",
    "name": "John Doe",
    "email": "john.doe@example.com"
  }
}
```

400 Bad Request: Invalid or missing fields.

json

Copy code

```
{
  "error": "All fields are required."
}
```

POST /api/auth/login

- **Description**: Logs in an existing user and provides a JWT token.
- **Request Method**: POST
- **Request Body**:

json

Copy code

```json
{
  "email": "john.doe@example.com",
  "password": "password123"
}
```

  - email: (string) The user's email.
  - password: (string) The user's password.

- **Response**:

200 OK:

json

Copy code

```json
{
  "token": "your_jwt_token_here"
}
```

400 Bad Request: Invalid credentials.

json

Copy code

```json
{
  "error": "Invalid email or password."
}
```

GET /api/auth/me

- **Description**: Retrieves the current logged-in user's profile using the provided JWT token.
- **Request Method**: GET
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>
- **Response**:

200 OK:

json

Copy code

```
{
  "id": "123456",
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

401 Unauthorized: Token is missing or invalid.

json

Copy code

```
{
  "error": "No token provided. Authorization denied."
}
```

2. Product Endpoints

GET /api/products

- **Description**: Retrieves a list of all products.
- **Request Method**: GET
- **Response**:

200 OK:

json

Copy code

```
[
  {
    "id": "1",
    "name": "Apple",
    "price": 2.5,
    "category": "Fruits",
    "description": "Fresh red apples."
  },
  {
```

"id": "2",

"name": "Banana",

"price": 1.2,

"category": "Fruits",

"description": "Ripe bananas."

}

]

GET /api/products/

- **Description**: Retrieves a specific product by ID.
- **Request Method**: GET
- **Parameters**:
    - id: (string) The ID of the product.
- **Response**:

200 OK:

json

Copy code

```
{
  "id": "1",
  "name": "Apple",
  "price": 2.5,
  "category": "Fruits",
  "description": "Fresh red apples."
}
```

404 Not Found: Product with the given ID not found.

json

Copy code

```
{
  "error": "Product not found."
}
```

3. Cart Endpoints

GET /api/cart

- **Description**: Retrieves the current user's cart.
- **Request Method**: GET
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>
- **Response**:

200 OK:

json

Copy code

```
{
  "items": [
    {
      "productId": "1",
      "quantity": 2,
      "totalPrice": 5.0
    },
    {
      "productId": "2",
      "quantity": 1,
      "totalPrice": 1.2
    }
  ],
  "totalPrice": 6.2
}
```

POST /api/cart

- **Description**: Adds a product to the cart.
- **Request Method**: POST
- **Request Body**:

json

Copy code

{

 "productId": "1",

 "quantity": 2

}

  - productId: (string) The ID of the product to add to the cart.
  - quantity: (integer) The quantity of the product to add.

- **Response**:

200 OK:

json

Copy code

```
{
 "message": "Product added to cart successfully.",
 "cart": {
  "items": [
   {
    "productId": "1",
    "quantity": 2,
    "totalPrice": 5.0
   }
  ],
  "totalPrice": 5.0
 }
}
```

4. Order Endpoints

POST /api/orders

- **Description**: Places an order for the items in the cart.
- **Request Method**: POST
- **Request Body**:

json

Copy code

{

```
  "shippingAddress": "123 Main St, City, Country",
  "paymentMethod": "credit_card"
}
```

  - shippingAddress: (string) The shipping address for the order.
  - paymentMethod: (string) The payment method used for the order (e.g., "credit_card").

- **Response**:

201 Created:

json

Copy code

```
{
  "message": "Order placed successfully.",
  "order": {
    "orderId": "7890",
    "items": [
      {
        "productId": "1",
        "quantity": 2,
        "totalPrice": 5.0
      }
    ],
    "totalPrice": 6.2,
    "status": "pending",
    "shippingAddress": "123 Main St, City, Country"
  }
}
```

GET /api/orders

- **Description**: Retrieves a list of orders placed by the logged-in user.
- **Request Method**: GET
- **Headers**:
  - Authorization: Bearer <JWT_TOKEN>

- **Response**:
    - **200 OK**:

json

Copy code

```
[
 {
  "orderId": "7890",
  "totalPrice": 6.2,
  "status": "pending",
  "shippingAddress": "123 Main St, City, Country",
  "createdAt": "2024-11-15T12:34:56Z"
 }
```

8. Authentication

1. Register:

    The user provides a name, email, and password to register. The server stores the information and sends a confirmation message.

2. Login:

    The user provides valid credentials (email and password). If correct, the server generates and returns a JWT token.

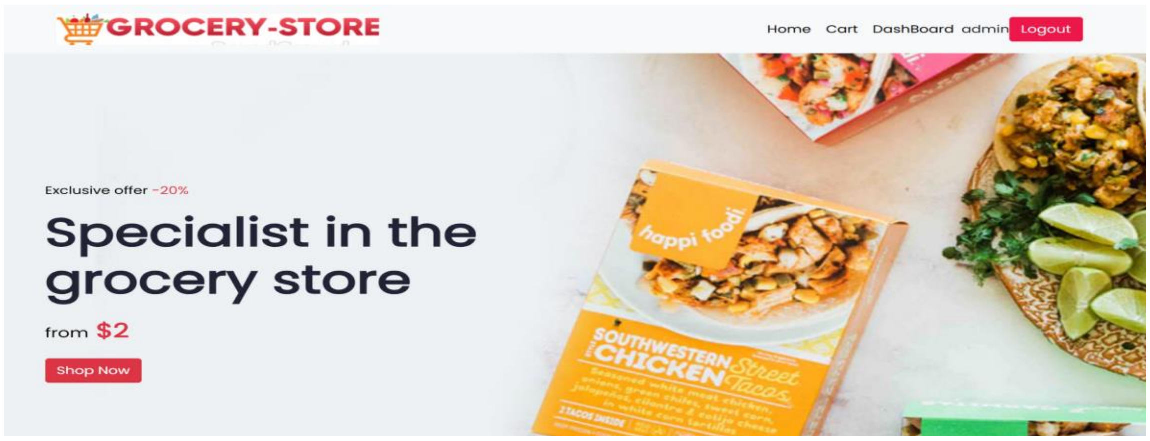
3. Authenticated Routes:

    For all authenticated API routes (like GET /api/auth/me), the user must send the JWT token in the Authorization header. This token is verified by the server, granting access to protected resources.

4. Logout:

    The user can log out by calling the /api/auth/logout endpoint, which invalidates the session (though the token is client-side, so it must be removed from storage).

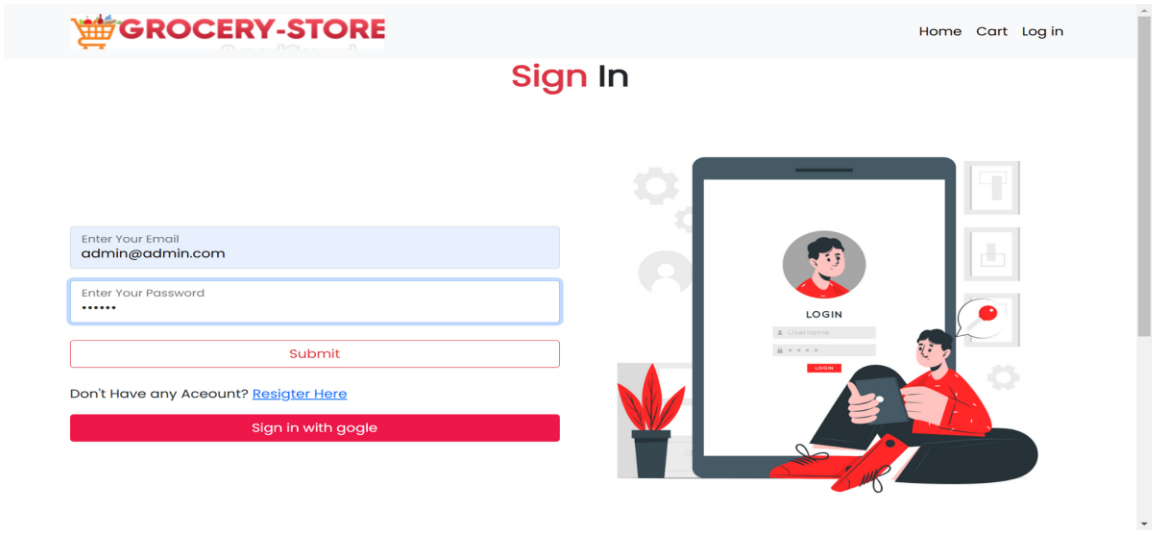
9. User Interface



Exclusive offer -20%

Specialist in the grocery store

from $2

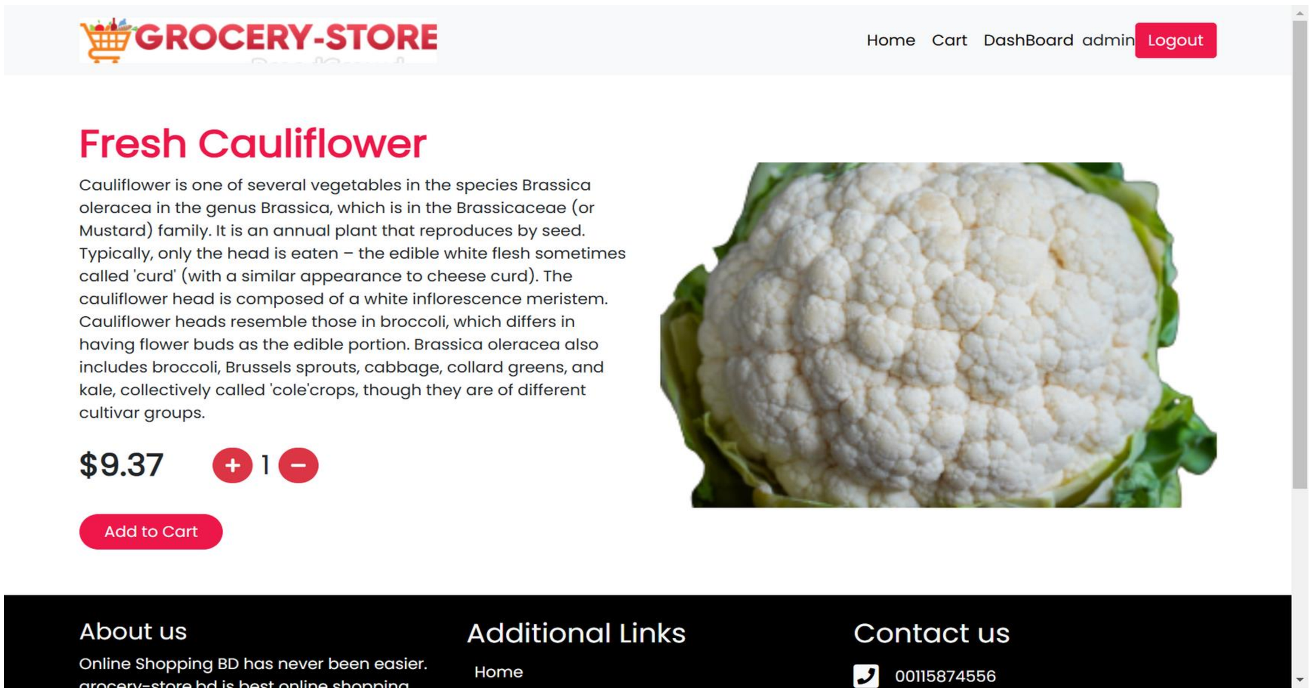


Sign In

Enter Your Email
admin@admin.com

Enter Your Password
••••••

Don't Have any Aceount? Resigter Here



Fresh Cauliflower

Cauliflower is one of several vegetables in the species Brassica oleracea in the genus Brassica, which is in the Brassicaceae (or Mustard) family. It is an annual plant that reproduces by seed. Typically, only the head is eaten – the edible white flesh sometimes called 'curd' (with a similar appearance to cheese curd). The cauliflower head is composed of a white inflorescence meristem. Cauliflower heads resemble those in broccoli, which differs in having flower buds as the edible portion. Brassica oleracea also includes broccoli, Brussels sprouts, cabbage, collard greens, and kale, collectively called 'cole'crops, though they are of different cultivar groups.

$9.37 + 1 −

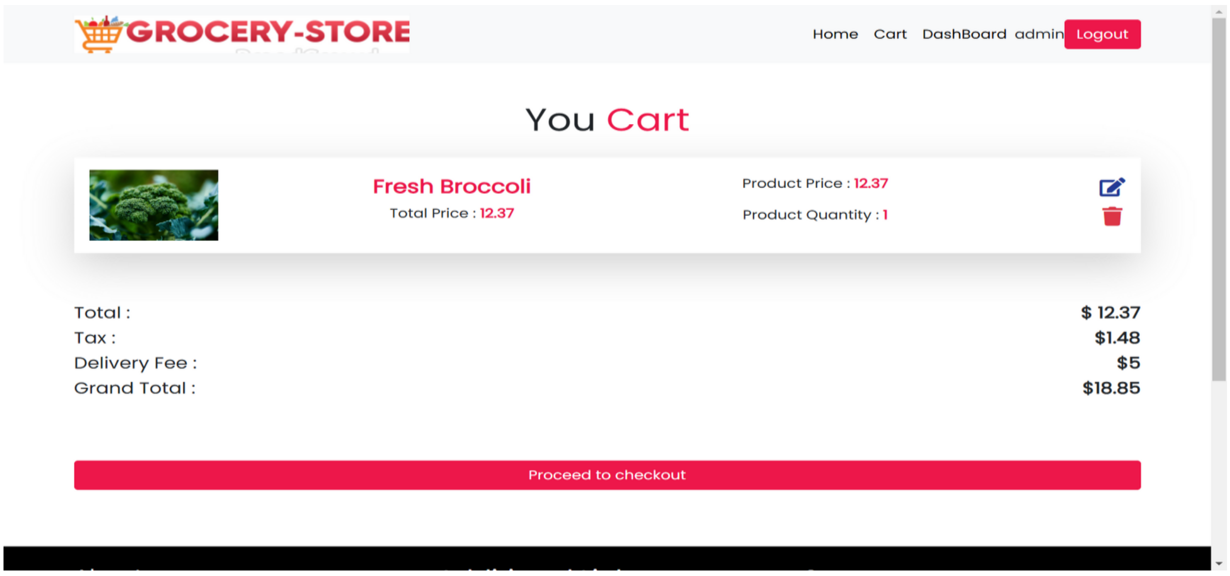
About us

Online Shopping BD has never been easier.

Additional Links

Contact us

00115874556

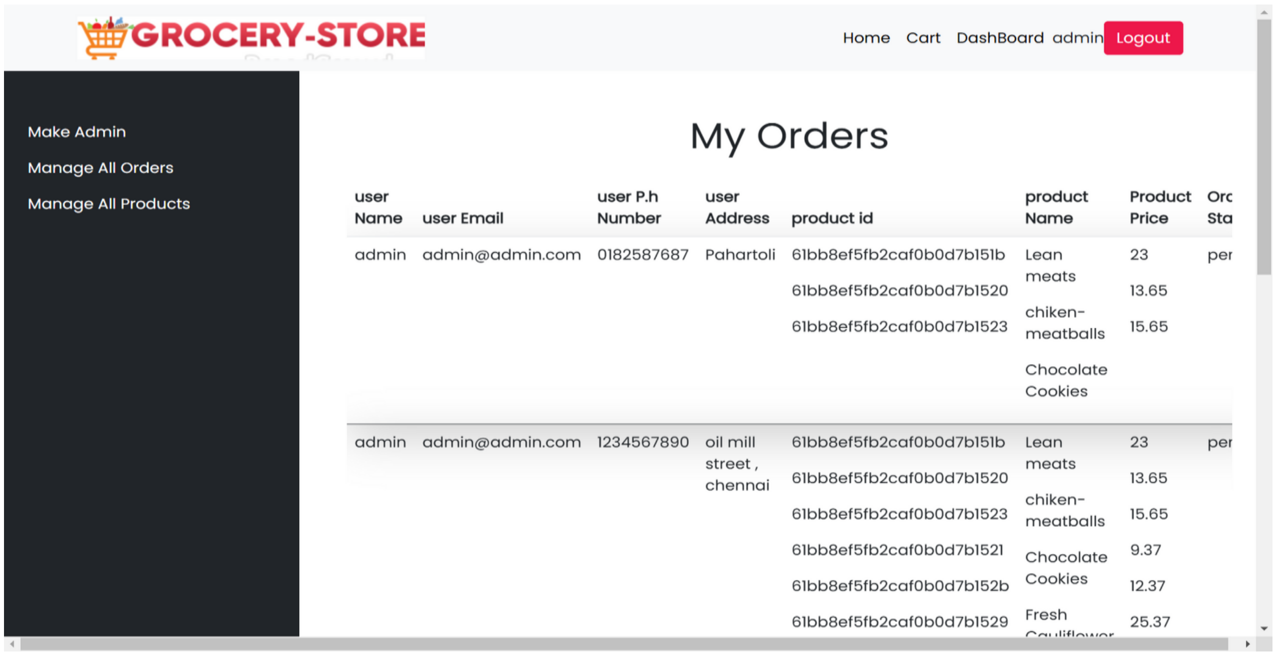


You Cart

Fresh Broccoli
Total Price : 12.37

Product Price : 12.37
Product Quantity : 1

Total :	$ 12.37
Tax :	$1.48
Delivery Fee :	$5
Grand Total :	$18.85



My Orders

user Name	user Email	user P.h Number	user Address	product id	product Name	Product Price	Orc Sta
admin	admin@admin.com	0182587687	Pahartoli	61bb8ef5fb2caf0b0d7b151b 61bb8ef5fb2caf0b0d7b1520 61bb8ef5fb2caf0b0d7b1523	Lean meats chiken-meatballs Chocolate Cookies	23 13.65 15.65	per
admin	admin@admin.com	1234567890	oil mill street , chennai	61bb8ef5fb2caf0b0d7b151b 61bb8ef5fb2caf0b0d7b1520 61bb8ef5fb2caf0b0d7b1523 61bb8ef5fb2caf0b0d7b1521 61bb8ef5fb2caf0b0d7b152b 61bb8ef5fb2caf0b0d7b1529	Lean meats chiken-meatballs Chocolate Cookies Fresh	23 13.65 15.65 9.37 12.37 25.37	per

11. Screenshots or Demo

https://drive.google.com/drive/u/0/folders/192AvnZ8fvdk47fbxgdpxb9sds91iV2Ym

12. Future Enhancements

To continually improve the functionality, user experience, and scalability of the Grocery Web App, the following enhancements are planned:

1. Mobile Application

- Develop native mobile applications for **iOS** and **Android** using technologies like **React Native** or **Flutter**.
- Include push notifications for order updates, promotional offers, and reminders for reordering frequently purchased items.

2. Subscription and Loyalty Programs

- Introduce a subscription model for regular customers (e.g., weekly or monthly deliveries for essentials).
- Implement a loyalty rewards system:
  - Earn points for every purchase.
  - Redeem points for discounts or free products.

3. Voice Search and Assistant Integration

- Add voice search functionality to enable users to find products using voice commands.
- Integrate with virtual assistants like **Google Assistant** or **Alexa** for hands-free shopping and order tracking.

4. Enhanced Analytics for Admins

- Develop a dashboard with advanced analytics to help administrators track:
  - Sales trends.
  - Inventory levels.
  - Customer behavior and preferences.
- Use predictive analytics to forecast demand and optimize inventory management