GROCERY STORE WEBAPP

TEAM MEMBERS:

1. **Boris Wilbert T-Frontend**
2. **Darmeshram R-Testing**
3. **Praveenraj-Admin**
4. **Santhana vignesh S-Backend**

1) PROJECT OVERVIEW

Purpose:

- User Interface (React): Build a responsive and user-friendly front-end interface for browsing menus, managing profiles, placing orders, and tracking order status in real time.

- Backend API (Node.js & Express): Manage business logic, handle API endpoints, authenticate users, process orders, integrate payments, and validate data efficiently.

- Database (MongoDB): Store and manage application data in a flexible, schema-less format to support dynamic and scalable data structures.

- Real-Time Features: Provide instant feedback and updates on order statuses using technologies like WebSockets for seamless real-time interactions.

- Admin Panel: Enable restaurant owners/operators to manage orders, menus, inventory, and customer feedback with ease.

- Deployment and Scaling: Deploy the application for public use and ensure it is optimized to handle real-world traffic and growth effectively.

Goals:

- User Interface (React): Provide an intuitive, visually appealing experience across devices. Enable seamless real-time interactions without page reloads.
- Backend API (Node.js & Express): Ensure data integrity and efficient communication between the front-end and database. Build a scalable backend capable of handling high traffic and real-time updates.

- Database (MongoDB): Facilitate efficient read/write operations during high traffic periods. Support scalability and flexibility for feature enhancements.
- Real-Time Features: Enhance user engagement with live updates. Streamline communication between users, restaurants, and delivery personnel.
- Admin Panel: Improve operational efficiency for restaurant staff. Provide actionable business insights through analytics.
- Deployment and Scaling: Ensure high availability and performance during peak usage. Implement auto-scaling to manage traffic spikes effectively.

2) ARCHITECTURE

Frontend Architecture (React):

Purpose: To build a dynamic, responsive, and interactive user interface using React's component-based architecture for a food ordering application.

Key Elements:

- Component-Based Structure: Reusable components like Header, Menu, Cart, OrderSummary, Login/Register, and Footer enhance modularity and maintainability.
- State Management:
  - Local State: Managed using useState for UI-specific needs.
  - Global State: Managed using Context API or Redux for cart, authentication, and order details.
- Routing: React Router handles navigation between Home, Menu, Cart, and Order Confirmation pages.
- Data Fetching: Axios with useEffect retrieves menu data, user details, and order statuses.
- Authentication: JWT-based login, with private routes for secure access.
- Real-Time Updates: WebSockets or polling for order status updates.

- Build & Deployment: Tools like Webpack, Create React App, and CI/CD pipelines ensure seamless deployment to platforms like Netlify or AWS.
- Error Handling: Error Boundaries and try/catch for robust error management.
- Testing: Jest and React Testing Library for unit and integration tests.

Backend Architecture (Node.js & Express)

Purpose: To manage business logic, API endpoints, and database interactions for the food ordering application.

Key Elements:

- Core Technologies: Node.js runtime and Express.js framework.
- Folder Structure: Organized into /config, /controllers, /models, /routes, and other modular directories for maintainability.
- Routing & Controllers: Routes define endpoints; controllers handle request logic and responses.
- Database Integration: MongoDB models for users, orders, reviews and menu items.
- Middleware: Functions for authentication (JWT), validation, and error handling.
- Authentication & Authorization: JWT for secure, token-based authentication.
- Error Handling: Centralized system for processing and logging errors.
- Logging: Tools like Winston or Morgan for request and error logging.
- API Documentation: Tools like Swagger for auto-generating API docs.
- Performance & Caching: Redis for caching; rate limiting for API protection.
- Testing: Jest, Mocha, and Supertest for backend validation.
- Deployment: Hosted on platforms like Heroku or AWS with Docker and NGINX for scaling.

Database Architecture (MongoDB)

Purpose: To store, retrieve, and manage application data efficiently using MongoDB's flexible schema-less structure.

Key Elements:

- Schema Design: Mongoose schemas enforce structure and validation for collections. Use embedding for related data (e.g., user orders) and referencing for independent updates (e.g., menu items).
- CRUD Operations: Models handle create, read, update, and delete operations.
- Indexes: Indexed fields (e.g., user email) improve query speed.
- Aggregation Framework: Advanced queries for grouping, filtering, and generating reports.
- Transactions: Ensure atomicity for multi-document operations.
- Sharding & Scaling: Distribute data across servers for horizontal scaling.
- Backup & Recovery: Automated backups and failover with MongoDB Atlas.
- Security: RBAC, encryption (at rest and in transit), and authentication mechanisms.
- Data Modeling Patterns: Embed for frequent reads; reference for normalization and flexibility.

3) SETUP INSTRUCTION

I. Prerequisites

Before you begin setting up the application, make sure you have the following software installed on your machine:

- Node.js (version 14 or higher)
  Download Node.js
- MongoDB (version 4.4 or higher)

Download MongoDB

Additionally, you'll need a code editor such as Visual Studio Code to view and edit the project files.

II. Installation

Clone Repository:

- Clone the repo: git clone <repository-url>
- Navigate to project directory: cd <project-directory>

Install Dependencies:

- Frontend (React):
  cd frontend
  npm install
- Backend (Node.js):
  cd backend
  npm install

Set Up Environment Variables:

- Frontend (.env):
  REACT_APP_API_URL=http://localhost:4000
- Backend (.env):
  mongodb+srv://sarakmugdha2910:sara2910@cluster0.tovm3.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0l4pl.mongodb.net/e-commerce>
- PORT=5000
  JWT_SECRET=mySecretKey

MongoDB Setup:

- Ensure MongoDB is running locally or update .env for MongoDB Atlas.

Run Database Migrations:

- Follow backend README or script files for migrations.

4) FOLDER STRUCTURE

Frontend (User)

```
src/
├── components/       # Contains reusable UI components
│   ├── Navbar.js     # Navigation bar for the app
│   ├── Menu.js       # Displays the list of food items
│   ├── Cart.js       # Manages the cart functionality
│   ├── PlaceOrder.js # Handles the order placement process
│   ├── PhotoReview.js # Displays user reviews with photos
├── context/          # Manages global state
│   ├── CartContext.js # Centralized state for cart operations
├── services/         # API services
│   ├── api.js        # Axios setup and backend API calls
├── pages/            # Page-level components
│   ├── Home.js       # Homepage layout
│   ├── AdminDashboard.js # Admin-specific functionalities
├── App.js            # Main component for routing and layout
├── index.js          # Entry point of the React app
├── styles/           # CSS files for styling components
│   ├── Navbar.css
│   ├── App.css
```

Backend (Database)

```
backend/
├── controllers/      # Handles the business logic for routes
│   ├── authController.js   # Handles user authentication
│   ├── foodController.js   # CRUD operations for food items
│   ├── orderController.js  # Handles order processing
│   ├── reviewController.js # Manages photo reviews
├── models/           # Mongoose schemas and models
```

```
│   ├── User.js          # User model
│   ├── FoodItem.js      # Food item model
│   ├── Order.js         # Order model
│   ├── Review.js        # Review model
├── routes/          # API endpoints
│   ├── authRoutes.js      # Routes for user authentication
│   ├── foodRoutes.js      # Routes for food items
│   ├── orderRoutes.js     # Routes for orders
│   ├── reviewRoutes.js    # Routes for reviews
├── middleware/      # Custom middleware
│   ├── authMiddleware.js  # Authenticates user and admin tokens
├── config/          # Configuration files
│   ├── db.js            # MongoDB connection setup
│   ├── dotenv.config    # Environment variables setup
├── server.js        # Entry point for the backend server
├── package.json     # Project dependencies and scripts
```

Admin

```
src/
├── admin/              # Admin-specific components and pages
│   ├── AdminDashboard.js  # Main dashboard displaying statistics
│   ├── ManageItems.js     # CRUD operations for food items
│   ├── ManageOrders.js    # View and update orders
│   ├── ManageReviews.js   # Approve/reject photo reviews
├── components/          # Shared components
│   ├── Navbar.js        # Admin-specific navigation bar
│   ├── Sidebar.js       # Sidebar for admin navigation
├── context/             # Global state for admin
│   ├── AdminContext.js  # Centralized state management for admin
├── services/            # API services for admin
```

```
│   ├── adminApi.js      # API calls for admin actions
├── pages/               # Admin entry point
│   ├── AdminLogin.js    # Admin login page
│   ├── AdminDashboard.js # Main page for admin workflows
├── App.js               # Routes and main app entry
├── styles/              # CSS files for admin UI
│   ├── AdminDashboard.css
│   ├── ManageItems.css
```

5) RUNNING THE APPLICATION

To run the application locally, you'll need to start both the frontend and backend servers. Follow the commands below to launch each part of the application:

Frontend

1. Navigate to the client directory:

   cd frontend

2. Start the React development server:

   npm run dev

This will run the frontend application on http://localhost:5173

Backend (Node.js)

1. Navigate to the server directory:

   cd backend

2. Start the Node.js server:

   Npm run server

This will run the backend server on http://localhost:4000

Admin

1. Navigate to the server directory:

```
cd admin
```

2. Start the Node.js server:

```
npm run dev
```

This will run the backend server on http://localhost:5174

6) API DOCUMENTATION

Sign Up:

Request Body (json):

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

Response Body:

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

User Login:

Request Body (json):

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

Response Body:


```
{
  "message": "Login successful",
  "token": "jwt_token_here"
}
```


Adding new food item:

Request Body (json):


```
{
  "name": "Cheese Burger",
  "description": "A delicious cheese burger with lettuce, tomato, and special sauce",
  "price": 8.99,
  "image": "cheeseburger.jpg",
  "category": "Burgers"
}
```


Response Body :


```
{
  "message": "Food item added successfully",
  "menuItem": {
    "id": "menu_item_id",
    "name": "Cheese Burger",
    "description": "A delicious cheese burger with lettuce, tomato, and special sauce",
    "price": 8.99,
    "image": "cheeseburger.jpg",
    "category": "Burgers",
  }
}
```

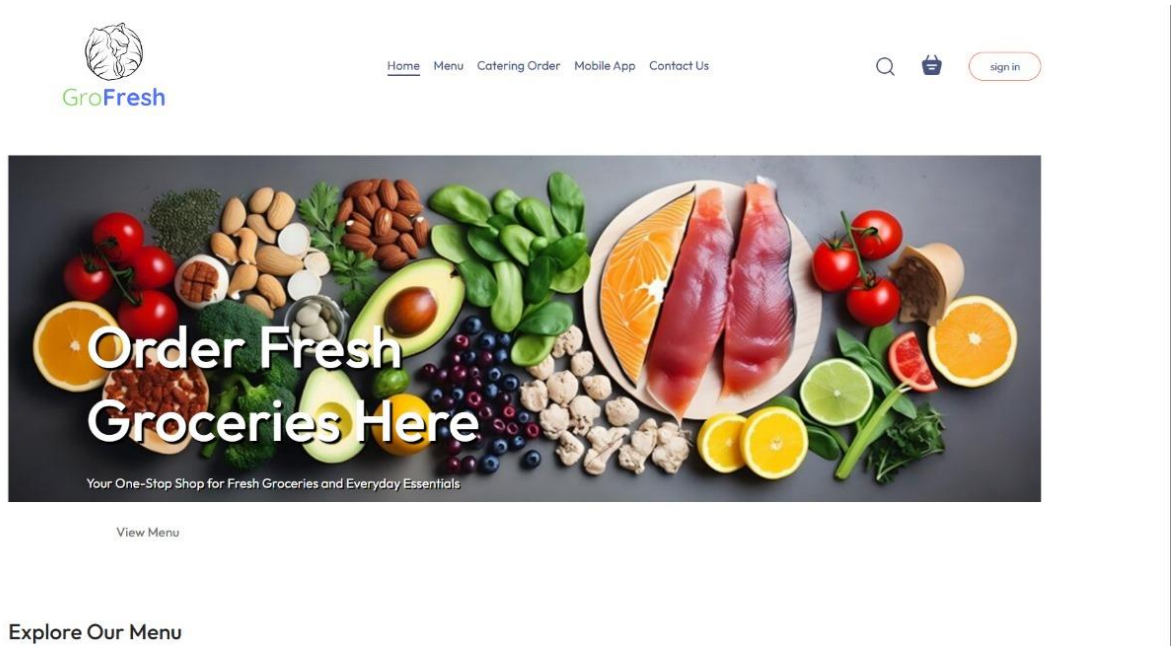
7) AUTHENTICATION

In this App, authentication is implemented using JWT (JSON Web Token) to secure the endpoints.

- Registration: Users register with their name, email, and password. Passwords are hashed before storing in the database.
- Login: Users log in with their email and password. Upon successful authentication, a JWT token is generated and returned.
- Token-Based Access: The client stores the token (usually in localStorage or sessionStorage) and includes it in the Authorization header for protected requests.
- Password Hashing: Hashing passwords before storing them in the database (using bcrypt).

8) USER INTERFACE

Home Page: Displays all available food items with search and category filter options.



localhost:4000 | Grocery Delivery

localhost:5173

GroFresh

GroFresh is a dynamic, innovative company that focuses on providing fresh, locally sourced produce directly to consumers through an efficient delivery system. It connects small-scale farmers with urban customers, ensuring access to high-quality fruits, vegetables, and herbs while promoting sustainable farming practices. With a focus on convenience and health, GroFresh offers customizable produce boxes, subscription services, and same-day deliveries. Its mission is to support local agriculture, reduce food waste, and provide nutritious, farm-fresh options to busy individuals and families who prioritize healthy living.

COMPANY

- Home
- About us
- Delivery
- Privacy Policy

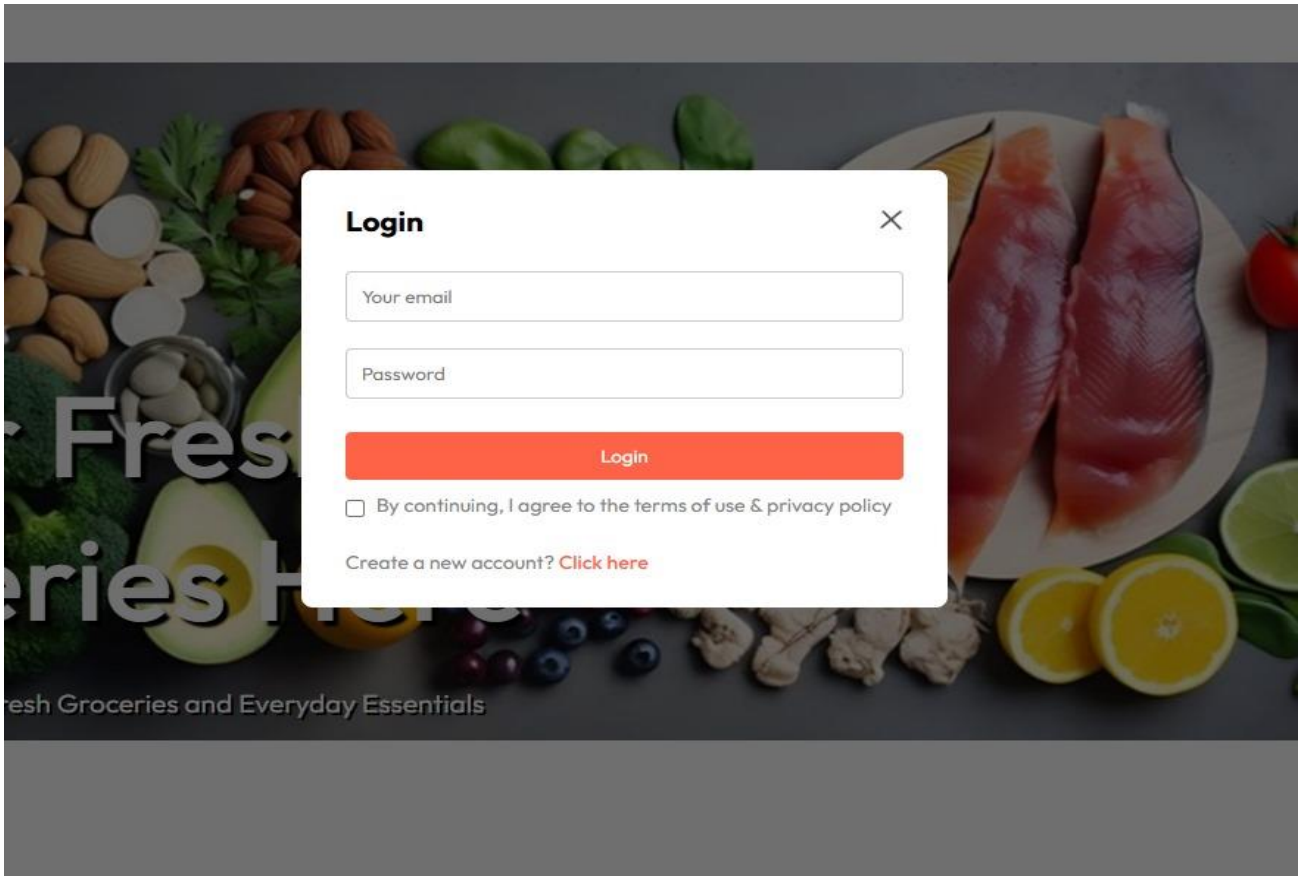
GET IN TOUCH

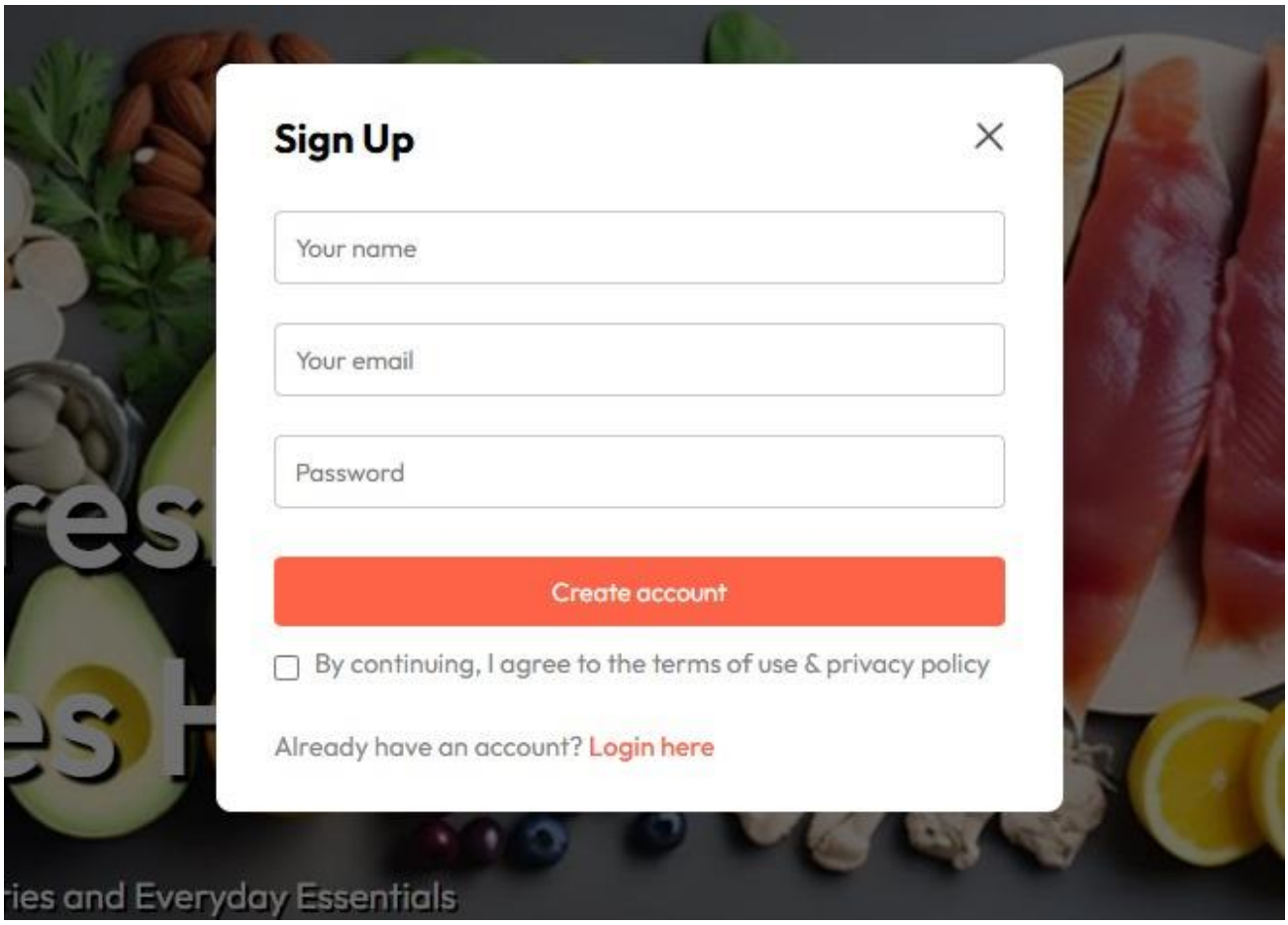
+91 8220034061

grofresh@gmail.com

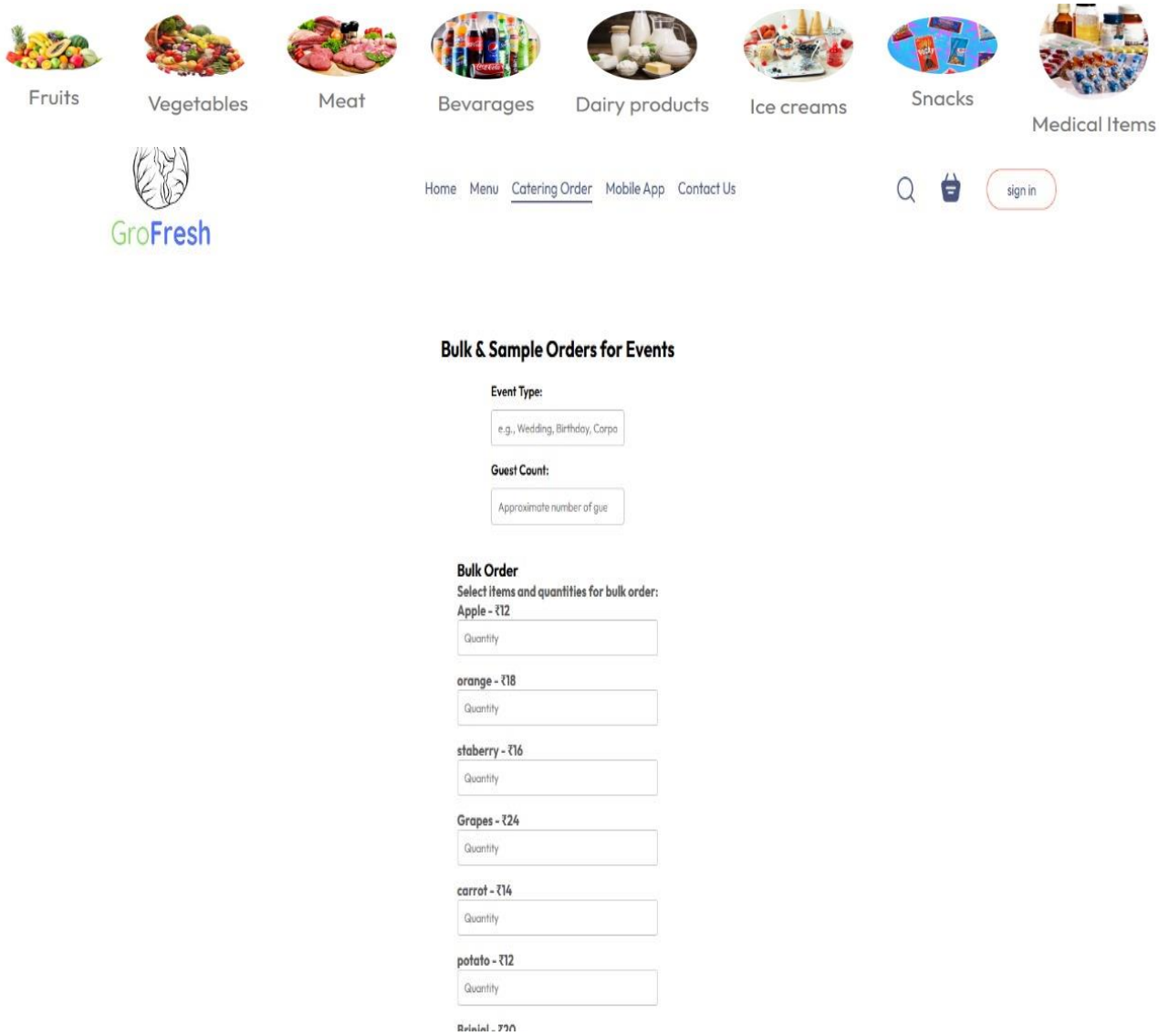
29°C Haze | Search | ENG IN | 11:25 20-11-2024

<u>Login/Sign Up:</u> Here new users can register themselves and already registered users can login

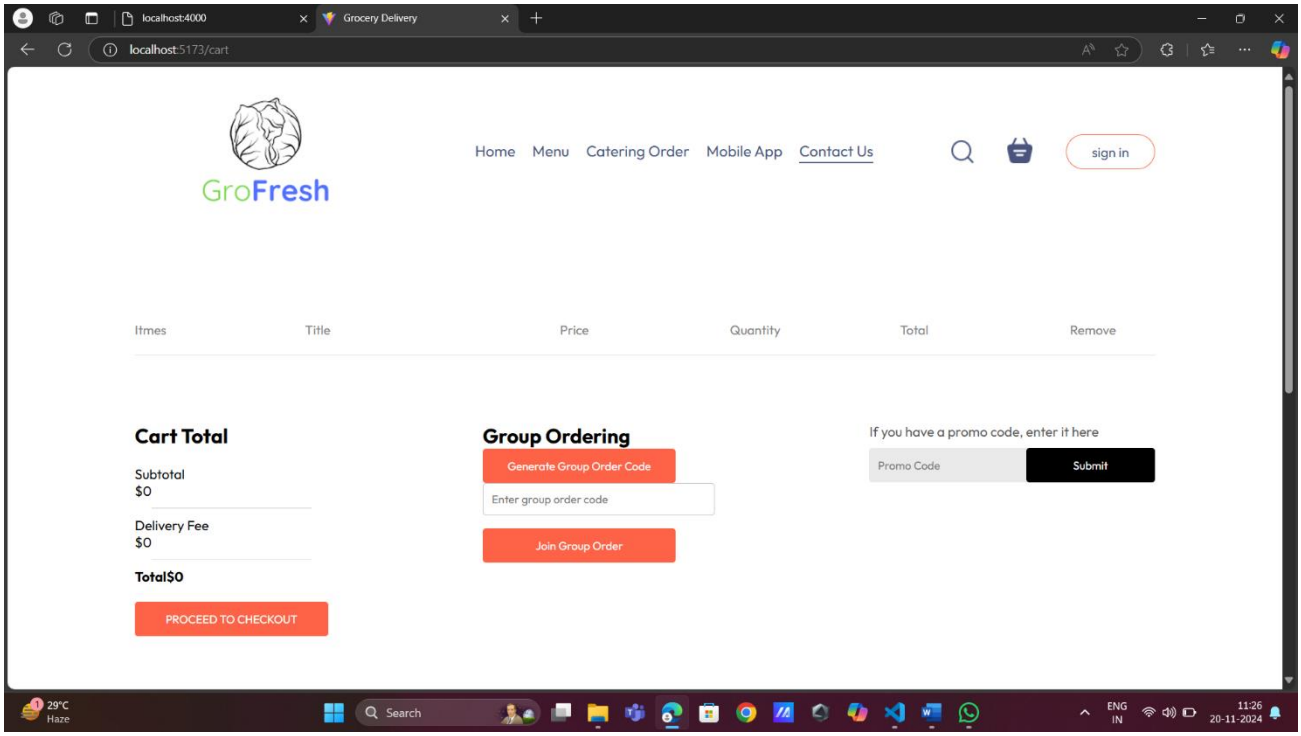



Sign Up
Your name
Your email
Password
Create account
By continuing, I agree to the terms of use & privacy policy
Already have an account? Login here
ies and Everyday Essentials

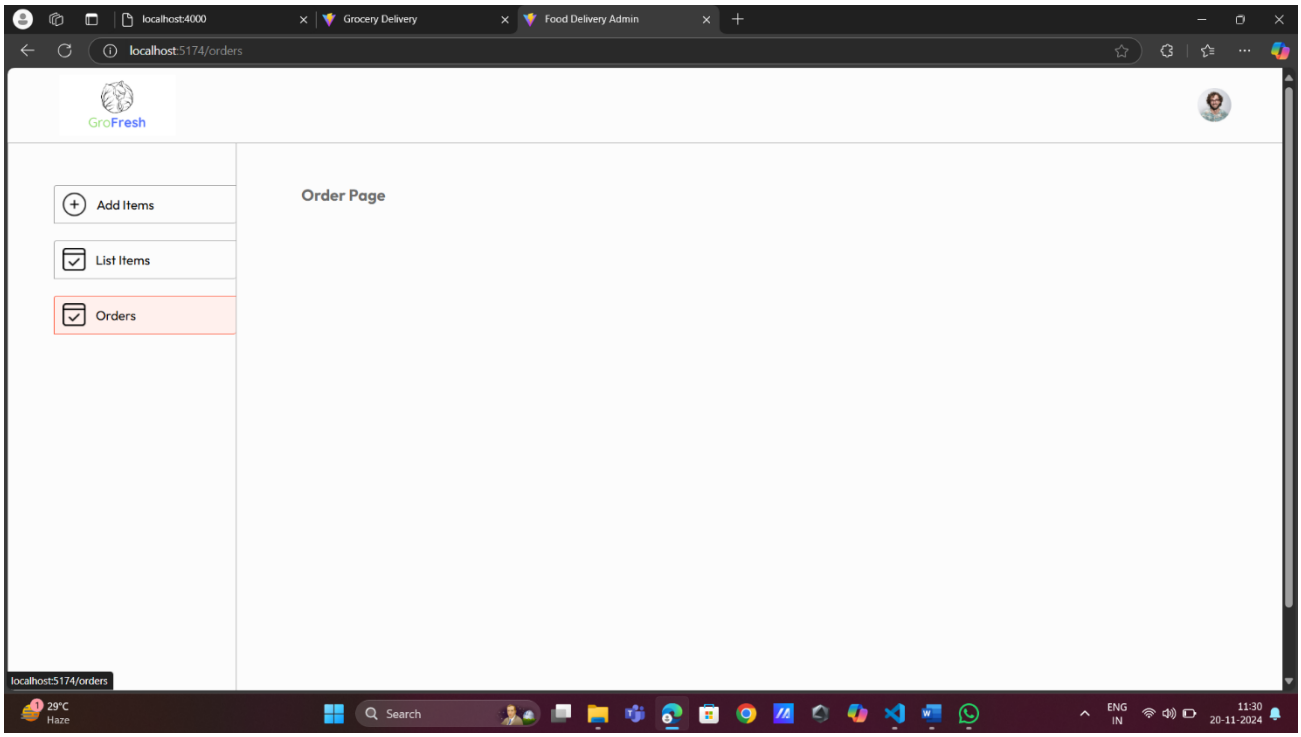
<u>Catering Order Page:</u> Here the users can give a bulk order and order food for events with specific favorable foods.



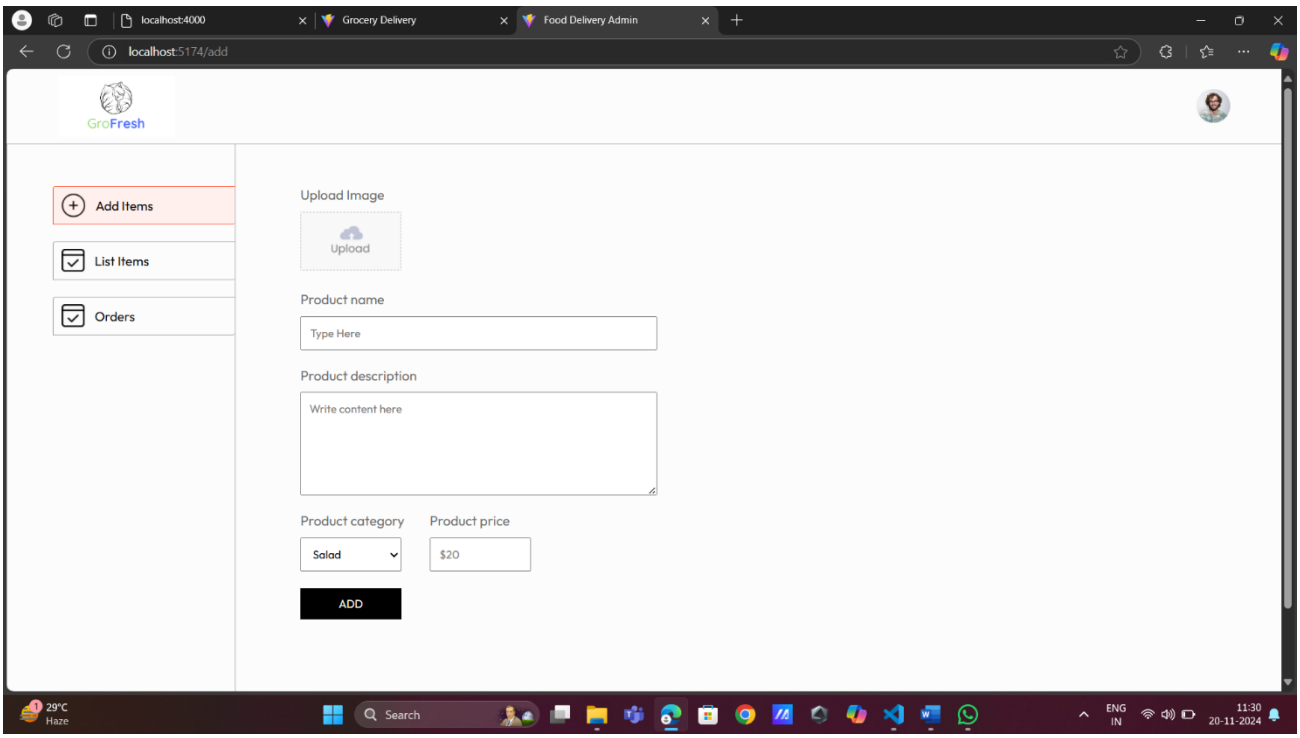
Cart Page: It would display the products under cart with quantity and price breakdown with group ordering feature.

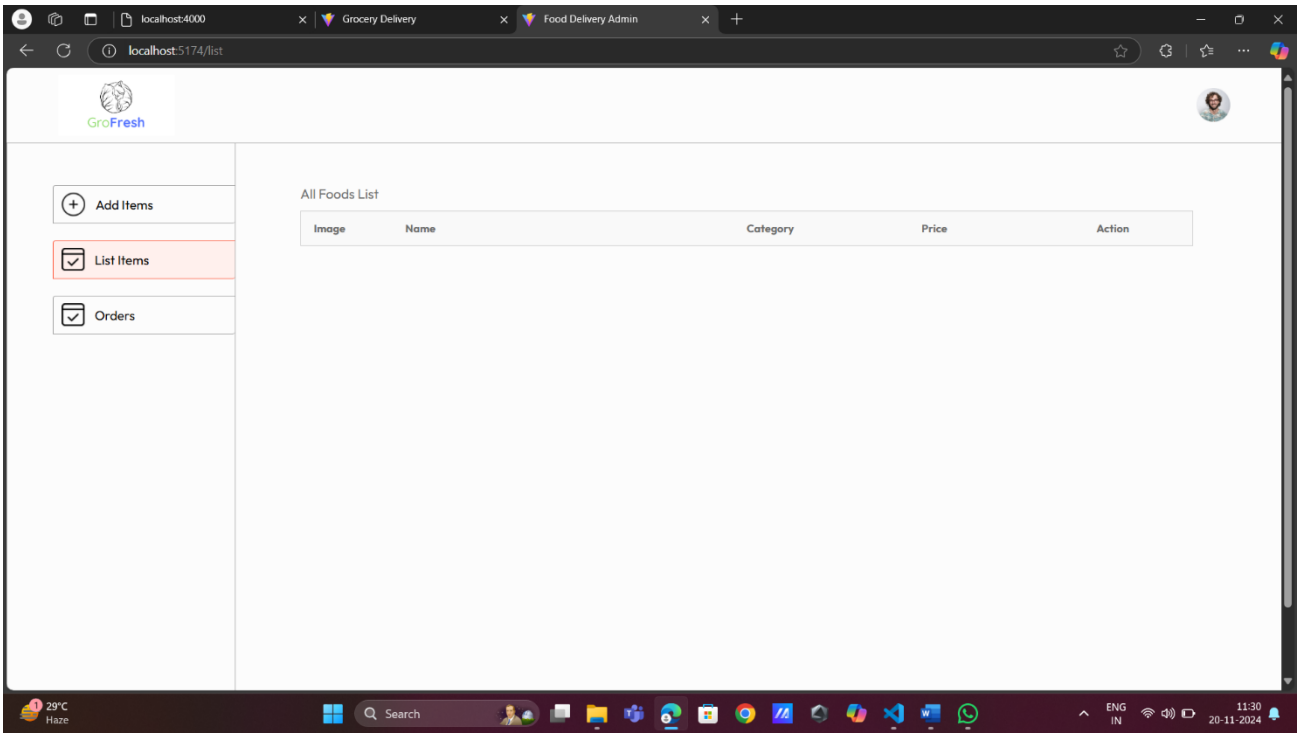


Place Order Page: It would display the products under cart with quantity and price breakdown and also the address and price details.



Admin Page: Here the admin can add, remove and view new food items in the menu and also track and view users' orders.





9) TESTING

Tools and Frameworks

Frontend Testing

- Jest: A testing framework for JavaScript, often used with React.
- React Testing Library: Used for testing React components and their interactions.

Backend Testing

- Mocha: A flexible testing framework for Node.js.
- Chai: An assertion library used with Mocha for readable test cases.
- Supertest: Used for testing HTTP requests to your Express APIs.

End-to-End Testing

- Cypress: For testing user workflows in the browser.
- Postman/Newman: For testing and automating API endpoints.

Test Cases

Frontend Test Cases

1. Component Tests:
    - Verify that the Navbar renders correctly with navigation links.
    - Ensure that the Cart component displays the correct number of items.
    - Check that Home page fetches and displays restaurant data.

```
cd frontend
npm test
```

2. Form Validation:
    - Test form inputs for login and registration (e.g., invalid emails, missing fields).

Backend Test Cases

1. API Endpoint Tests:
    - Verify /api/users registers a new user.
    - Test /api/auth/login for valid and invalid credentials.

```
cd backend
npm test
```

2. Database Tests:
    - Ensure Mongoose models validate schema correctly.

End-to-End Test Cases

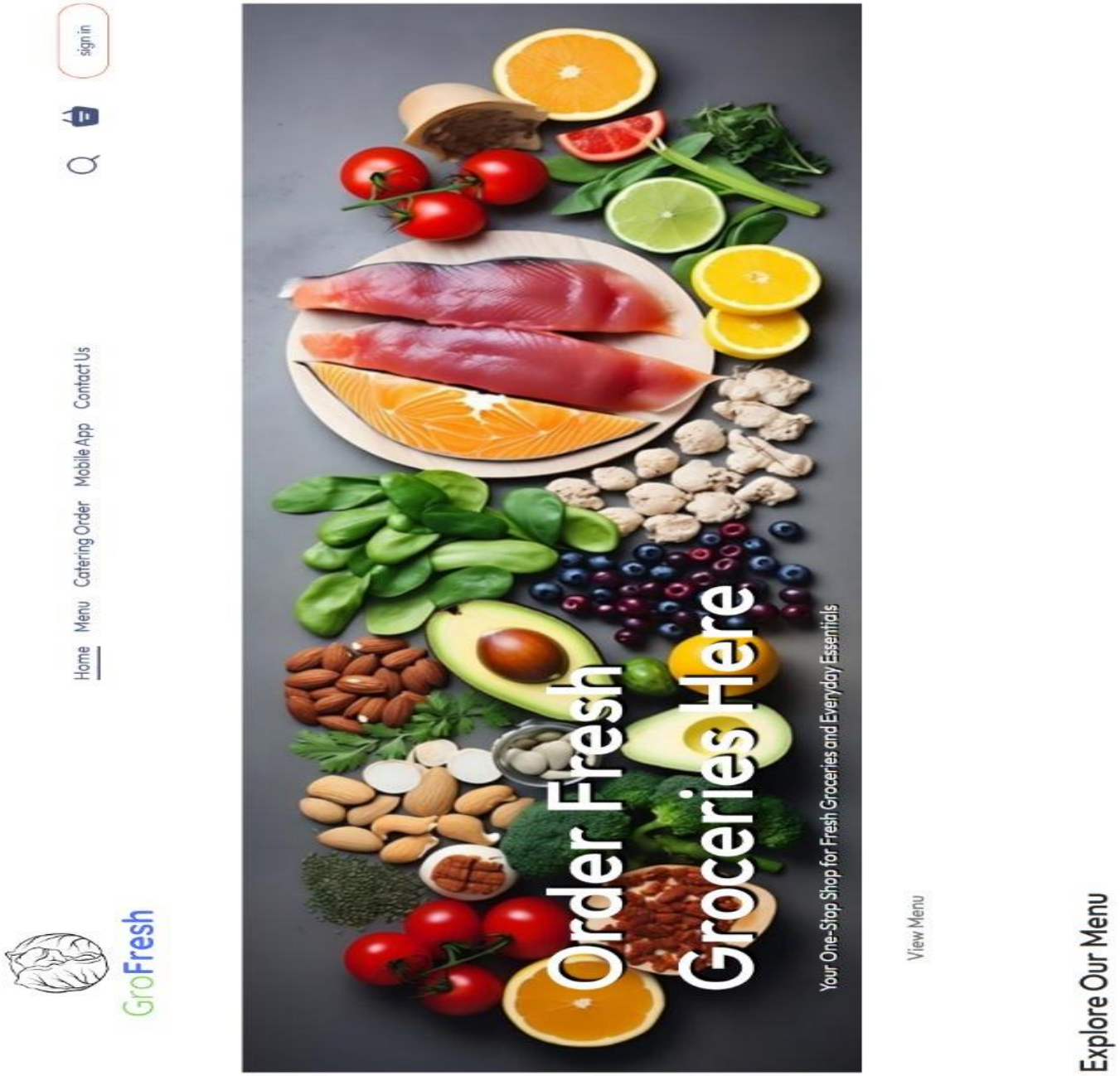
1. User Workflow:
    - Simulate a user registering, logging in, adding items to the cart, and placing an order.
    - Use Cypress to verify UI elements and flows.

```
npx cypress open
```

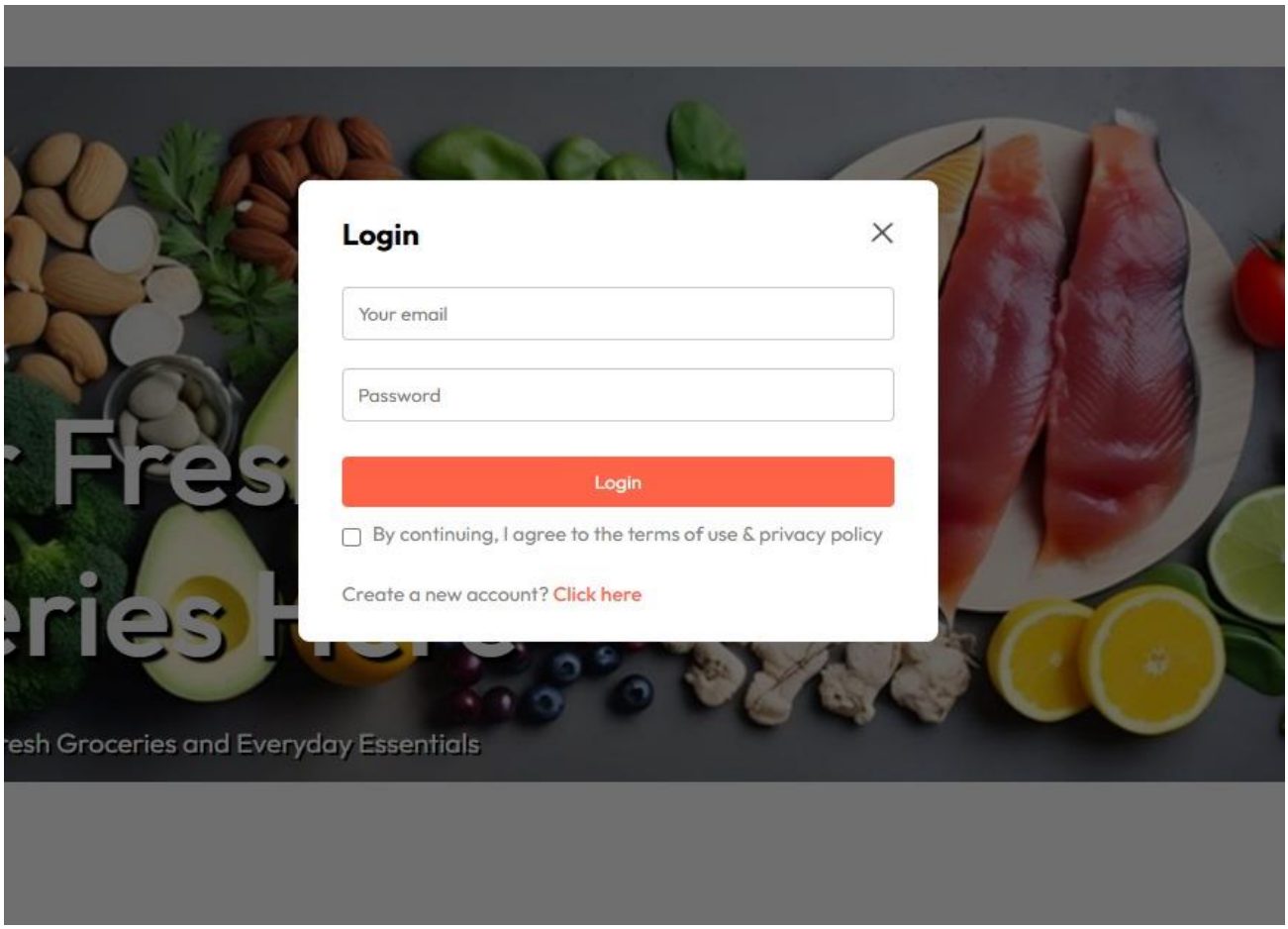
10) SCREENSHOTS

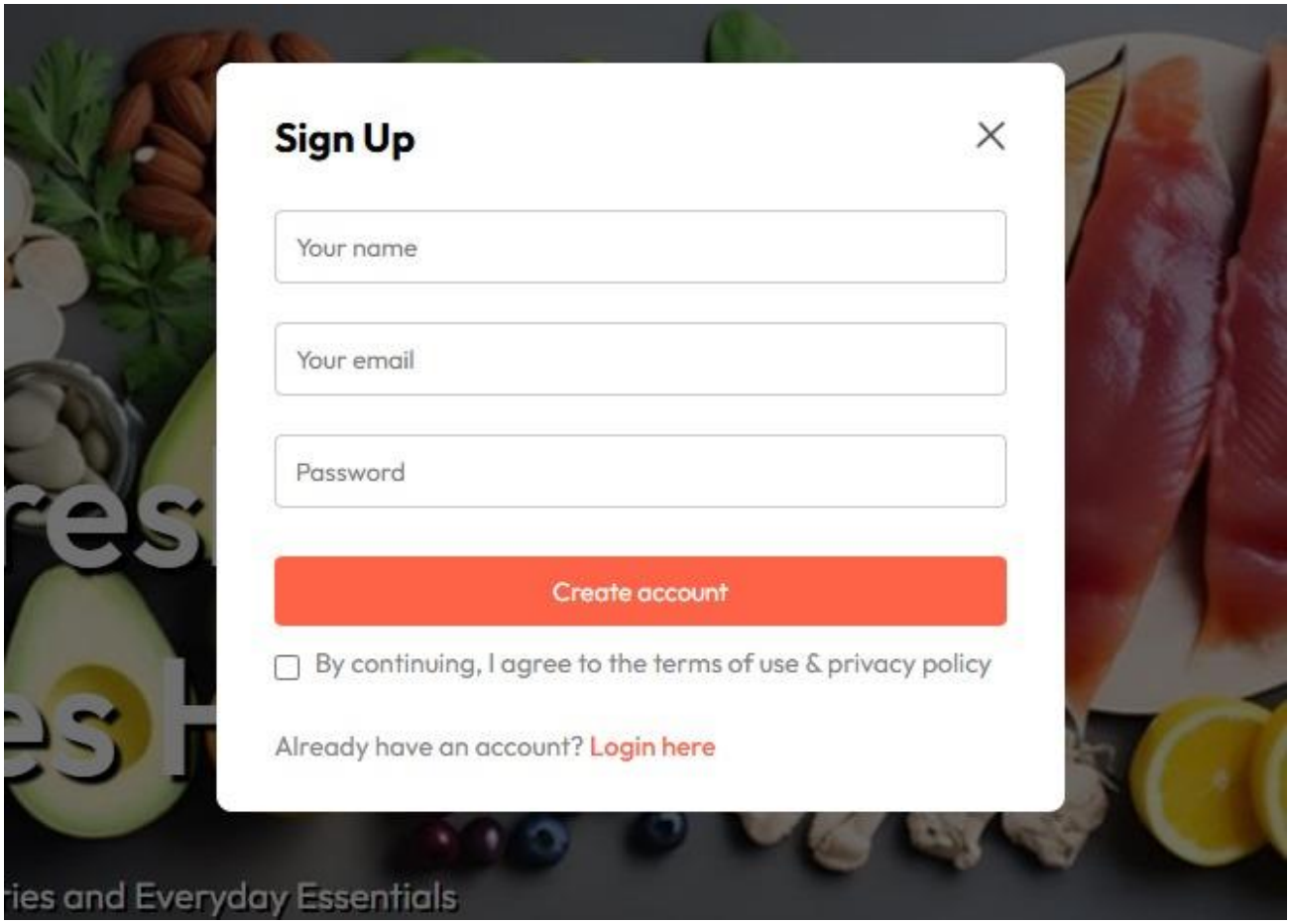
Frontend Screenshots:

Home Page

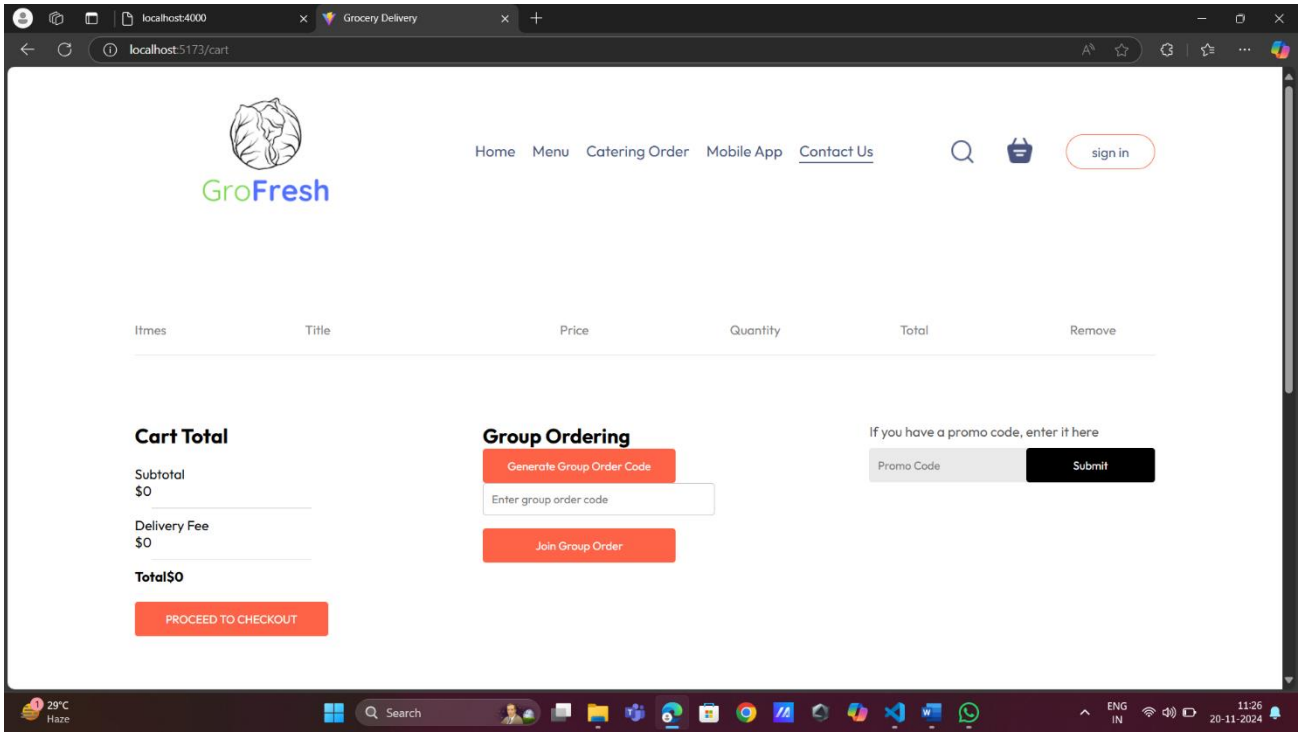


Login/Sign Up:

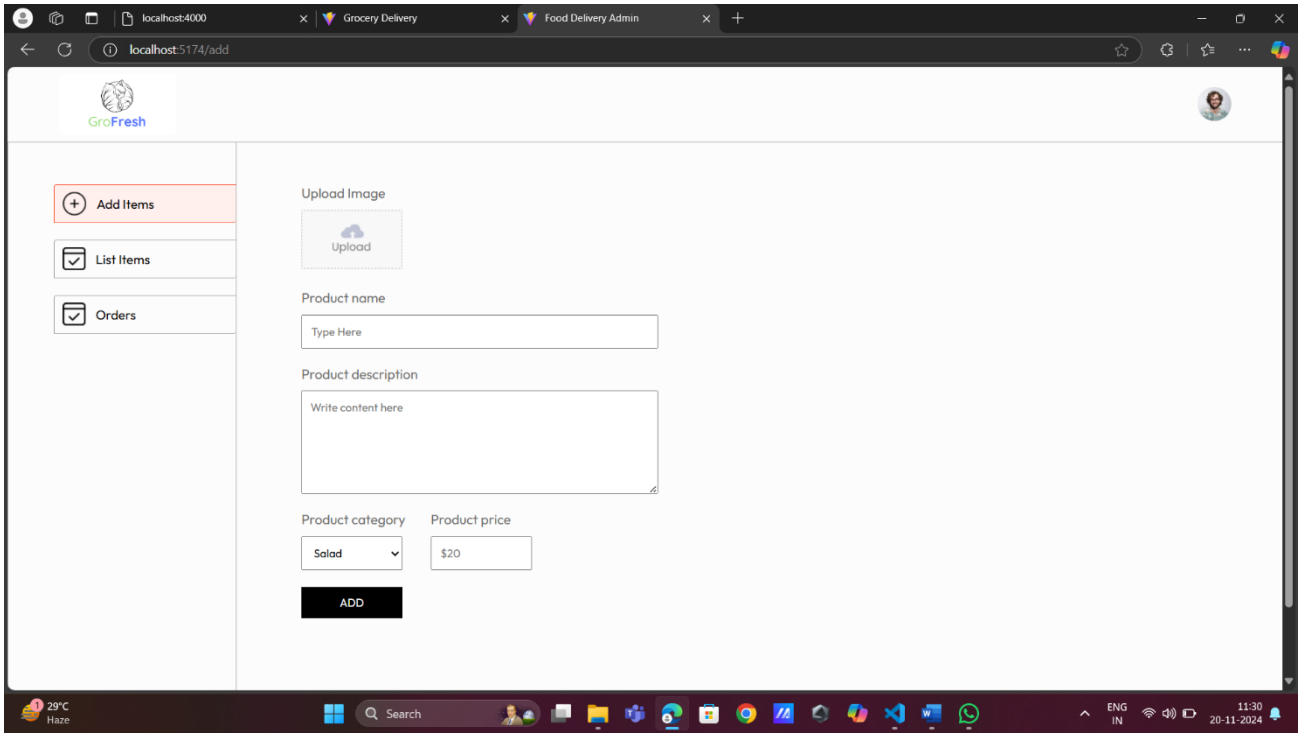


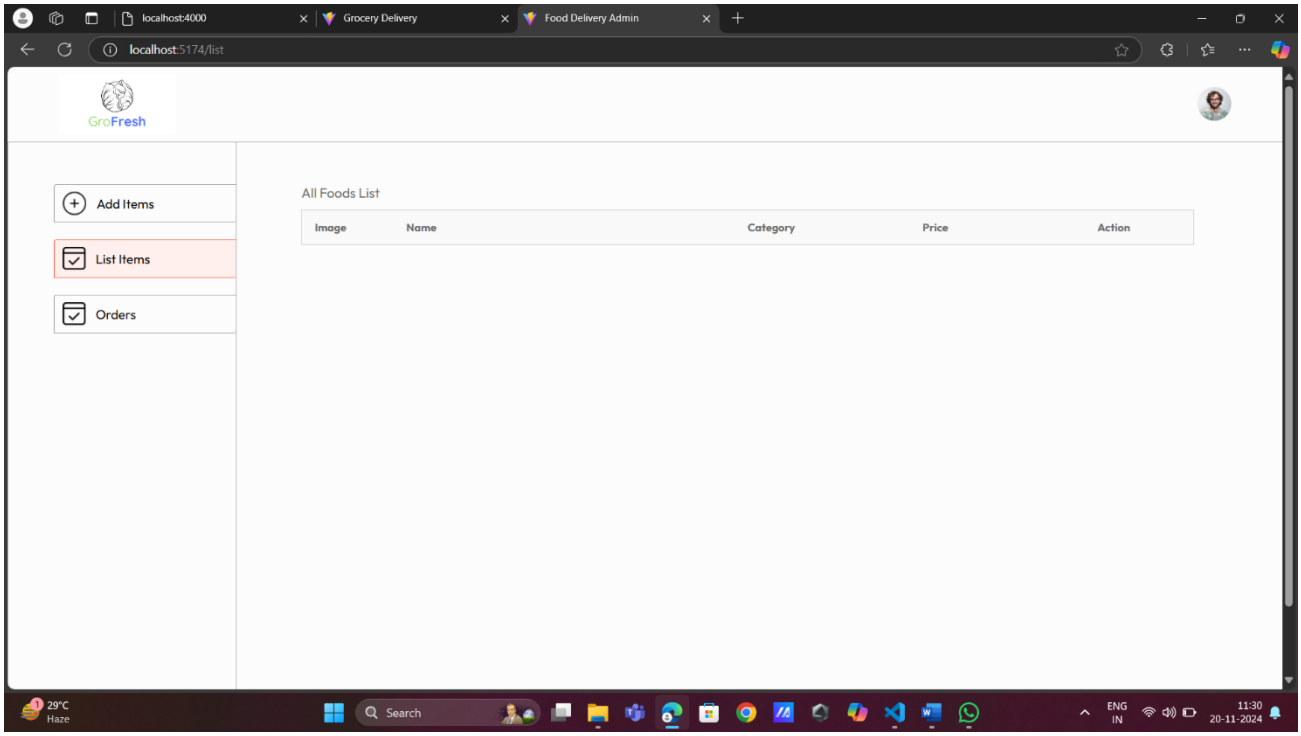

Sign Up
Your name
Your email
Password
Create account
By continuing, I agree to the terms of use & privacy policy
Already have an account? Login here
ies and Everyday Essentials

Cart Page:


localhost:4000
Grocery Delivery
localhost:5173/cart
GroFresh
Home
Menu
Catering Order
Mobile App
Contact Us
sign in
Itmes
Title
Price
Quantity
Total
Remove
Cart Total
Subtotal
$0
Delivery Fee
$0
Total$0
PROCEED TO CHECKOUT
Group Ordering
Generate Group Order Code
Enter group order code
Join Group Order
If you have a promo code, enter it here
Promo Code
Submit

Admin Page:


localhost:4000
Grocery Delivery
Food Delivery Admin
localhost:5174/add
GroFresh
Add Items
List Items
Orders
Upload Image
Upload
Product name
Type Here
Product description
Write content here
Product category
Product price
Salad
$20
ADD


localhost:4000
Grocery Delivery
Food Delivery Admin
localhost:5174/list
GroFresh
Add Items
List Items
Orders
All Foods List
Image
Name
Category
Price
Action

- **KNOWN ISSUES:**

  - Performance: Slow response times for API requests since Database queries not optimized or unnecessary re-renders in the frontend.
  - Deployment: Environment variables or configurations are exposed in the deployment.
  - JWT Token Expiry Handling: Expired tokens not detected, causing authorization errors.

- **FUTURE ENHANCEMENTS:**

  - Notifications: Add email and push notifications to notify users of order confirmations, status changes, and promotional offers.
  - Multi-Tenant Support: Support multiple restaurant owners with separate dashboards which allows each restaurant to manage their menu, orders, and analytics independently.
  - Restaurant Analytics: Provide analytics for restaurants which show metrics like sales trends, popular dishes, and peak order times.
  - Dynamic Pricing: Enable dynamic pricing for restaurants which will allow restaurants to offer discounts during off-peak hours or surge pricing during busy periods