Date : 19 – 07 – 2024

Team ID : SWTID1720150432

Project Name : EagerEats-Food Ordering App

Full Stack Development with MERN

Project Documentation

.

1. Introduction

- **Project Title**: EagerEats - Food Ordering App
- **Team Members**:
    - Kishore S
    - Sanjay R
    - Praveen S
    - Rishikeshwaran K R

2. Project Overview

- **Purpose**: EagerEats is a food ordering application developed using the MERN stack (MongoDB, Express.js, React, and Node.js). The primary goal of the project is to provide users with a seamless experience for browsing, selecting, and ordering food items online. Users can easily create accounts, add food items to their cart, manage their orders, and view their order history.
- **Features**:
    - Home page: Users can browse available food items categorized by type.
    - User Authentication: Users can sign up for a new account or log in to an existing account.
    - Add to Cart: Users can add food items to their cart with options to adjust the quantity and size (e.g., medium, full).
    - Cart Management: Users can view their cart, update quantities, and remove items.
    - Checkout: Users can proceed to checkout and place their orders.
    - Order History: Users can view their previous orders on the "My Orders" page.

3. Architecture

Frontend

The frontend of EagerEats is developed using React. The main components and their interactions are as follows:

- **Components**:
    - Card.js: Manages the display of individual food items.
    - Carousel.js: Handles the carousel display on the homepage.
    - ContextReducer.js: Manages the state of the application using the context API.
    - Footer.js: Displays the footer section.
    - Navbar.js: Manages the navigation bar, including authentication links and cart display.
- **Screens**:
    - Cart.js: Displays the user's cart and allows for item quantity adjustments and deletions.
    - Home.js: The main landing page displaying available food items.
    - Login.js: Handles user login functionality.
    - MyOrder.js: Displays the user's past orders.
    - Signup.js: Manages user registration.

Backend

The backend is built using Node.js and Express.js. It includes the following main routes and controllers:

- **Models**:
    - FoodItems.js: Defines the schema for food items.
    - User.js: Defines the schema for user data.
- **Routes**:
    - CreateUser.js: Handles user creation and registration.
    - DisplayData.js: Fetches and displays data such as food items.
    - OrderData.js: Manages order-related operations.
    - index.js: The main entry point for the backend application.

Database

The database is managed using MongoDB and includes the following collections and schemas:

- **Collections**:
    - food_items: Stores information about the available food items.

- foodCategory: Categorizes food items.
- signup: Manages user registration data.

4. Setup Instructions

Prerequisites

Ensure you have the following software installed:

- **Node.js** (v14.x or higher)
- **npm** (v6.x or higher) or **yarn** (v1.x or higher)
- **MongoDB** (v4.x or higher)

Installation

Follow these steps to set up the project locally:

1. **Clone the repository**:

   git clone [repository-url]

   cd [repository-directory]

2. **Set up the backend**:

   ```
   cd backend
   npm install
   ```

3. **Set up the Frontend**:

   ```
   cd ../src
   npm install
   ```

5. Folder Structure

Client

The React frontend is organized as follows:

- **public**: Contains static assets such as images and the main HTML file.
  - AuthImage.jpg
  - favicon.ico
  - Header.png
  - index.html
  - Logo.png
  - logo192.png
  - manifest.json

    - robots.txt
- **src**: Contains the source code for the React application.
    - **components**: Contains reusable components.
        - Card.js
        - Carousel.js
        - ContextReducer.js
        - Footer.js
        - Navbar.js
    - **screens**: Contains page components.
        - Cart.js
        - Home.js
        - Login.js
        - MyOrder.js
        - Signup.js
    - App.js: The main application component.
    - App.css: Global styles for the application.
    - index.js: The entry point for the React application.
    - index.css: Global CSS styles.
    - setupTests.js: Configuration for testing.

Server

The Node.js backend is organized as follows:

- **models**: Contains Mongoose schemas.
    - FoodItems.js
    - User.js
- **routes**: Contains route handlers.
    - CreateUser.js
    - DisplayData.js
    - OrderData.js
    - index.js: The main entry point for the backend server.

6. Running the Application

To run the application locally, use the following commands:

Frontend

To start the React frontend server:

```
cd src
npm start
```

Backend

To start the Backend server:

```
cd backend
npx nodemon index.js
```

7. API Documentation

1. Create User

Endpoint: POST /api/createuser

- **Description:** Creates a new user.
- **Request Parameters:**
  - **Body:**
    - email (string): User's email address. Must be a valid email format.
    - name (string): User's full name. Minimum length of 5 characters.
    - password (string): User's password. Must be at least 5 characters long.
    - location (string): User's location (optional, depending on your schema).

Request Example:

POST /api/createuser

Content-Type: application/json

```
{
  "email": "john@example.com",
  "name": "John Doe",
  "password": "password123",
  "location": "New York"
```

}

Response:

- **Status Code:** 200 OK
- **Body:**


```
{
  "success": true,
  "user": {
    "id": "123",
    "name": "John Doe",
    "email": "john@example.com"
  }
}
```


Status Code: 400 Bad Request

- **Body:**


```
{
  "errors": [
    { "msg": "Invalid value", "param": "email", "location": "body" },
    ...
  ]
}
```


2. User Login

Endpoint: POST /api/loginuser

- **Description:** Authenticates a user and returns a JWT token.
- **Request Parameters:**
  - **Body:**
    - email (string): User's email address. Must be a valid email format.
    - password (string): User's password. Must be at least 5 characters long.

Request Example

POST /api/loginuser

Content-Type: application/json

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

Response:

- **Status Code:** 200 OK
- **Body**

{
"success": true,
"authToken": "your_jwt_token"
}

Status Code: 400 Bad Request

- **Body**

{
"errors": "Try logging with correct credentials"
}

3. Retrieve Food Data

Endpoint: POST /api/foodData

- **Description:** Retrieves food items and categories.
- **Request Parameters:** None
- **Request Example**

POST /api/foodData

Response:

- **Status Code:** 200 OK
- **Body**

{
"food_items": [

```
  { "id": "123", "name": "Burger", "category": "Burgers", "price": 5.99 }
  ...
 ],
 "foodCategory": [
  { "id": "1", "name": "Burgers" },
  ...
 ]
}
```


4. Place Order

Endpoint: POST /api/orderData

- **Description:** Creates or updates an order based on the user's email.
- **Request Parameters:**
  - **Body:**
    - email (string): User's email address.
    - order_data (array): Array of order items, including order date.
    - order_date (string): The date of the order.
- **Request Example**

POST /api/orderData

Content-Type: application/json


```
{
 "email": "john@example.com",
 "order_data": [
  { "id": "123", "name": "Burger", "quantity": 2 }
 ],
 "order_date": "2024-07-19"
}
```


Response:

- **Status Code:** 200 OK

- **Body**


```
{
 "success": true
}
```


5. Retrieve User Orders

Endpoint: POST /api/myorderData

- **Description:** Retrieves orders for a specific user.
- **Request Parameters:**
  - **Body:**
    - email (string): User's email address.

Request Example

POST /api/myorderData

Content-Type: application/json


```
{
 "email": "john@example.com"
}
```


Response:

- **Status Code:** 200 OK
- **Body**


```
{
 "orderData": {
  "email": "john@example.com",
  "order_data": [
   { "id": "123", "name": "Burger", "quantity": 2, "order_date": "2024-07-19" }
  ]
 }
}
```


Status Code: 500 Internal Server Error

- **Body**

```
{
  "message": "Server Error"
}
```

8.Authentication

Authentication Method:

- **Token-Based Authentication:**
    - **JWT Token:** The project uses JSON Web Tokens (JWT) for authentication. Tokens are generated using the jsonwebtoken library.
    - **Secret Key:** Tokens are signed with the secret key praveenmasterofcoding.
    - **Expiration:** Tokens are typically valid for a specific period (e.g., 1 hour), though this may be configured based on your project's needs.

Authorization:

- **Protected Routes:** Certain routes are protected and require authentication. For example:
    - POST /api/orderData
    - POST /api/myorderData
- **Token Verification:** Middleware functions are used to verify the JWT token. The token must be included in the Authorization header of requests in the format Bearer <token>. If the token is missing, invalid, or expired, the request is denied, and an appropriate error message is returned.

Token Handling:

- **Token Storage:** On the client side, the JWT token is typically stored in localStorage. This allows the token to persist across page reloads.
- **Token Transmission:** Tokens are transmitted with requests by including them in the Authorization header as Bearer <token>. This ensures that each request is authenticated.

Sessions:

- **Session Management:** The project does not use traditional server-side sessions. Instead, authentication is managed entirely through JWT tokens, which are validated on each request.

User Roles and Permissions:

- **Role-Based Access Control:** (If applicable) The current implementation does not include explicit user roles or permissions. All authenticated users have access to the routes they are authorized to use based on token validation.

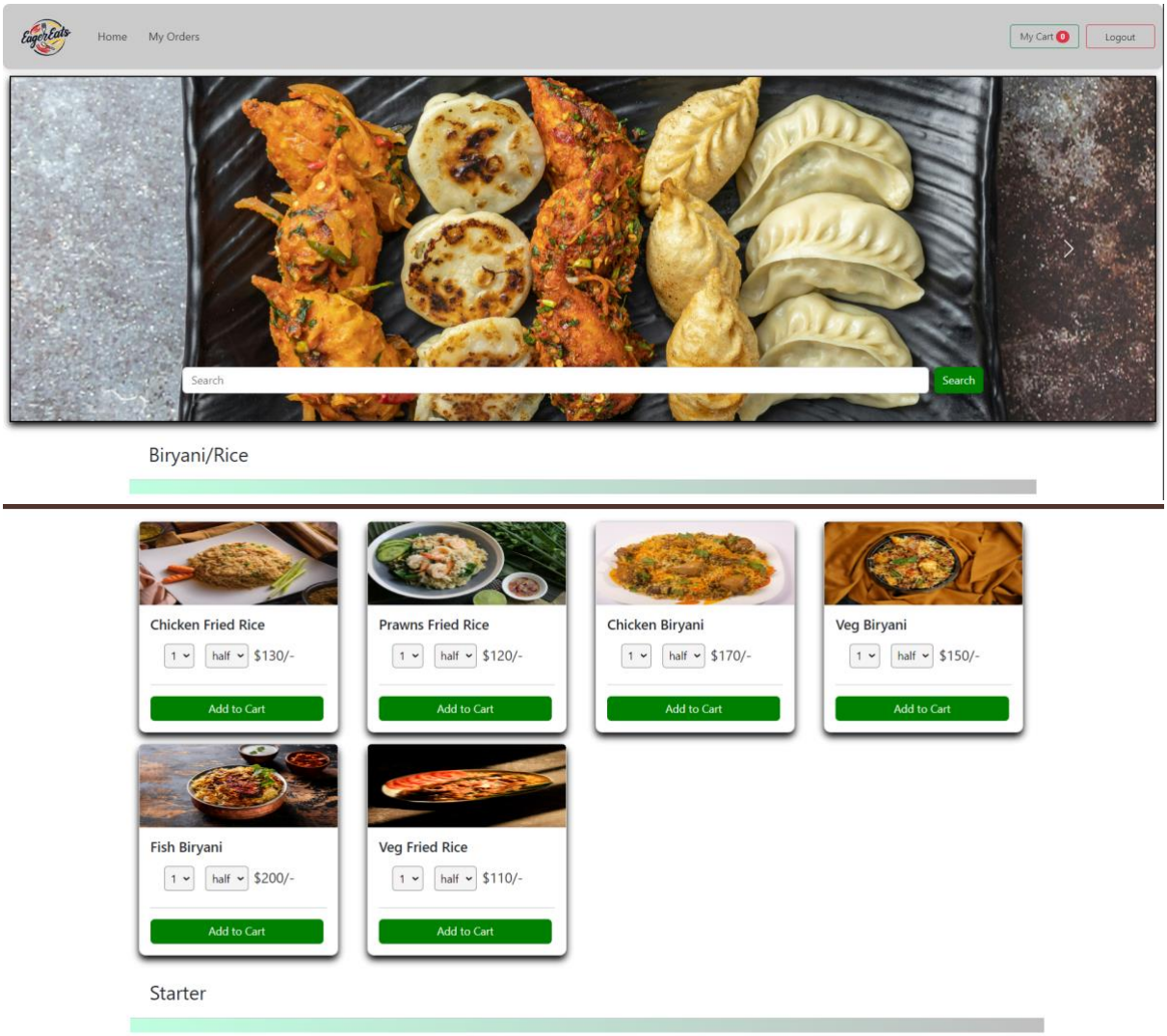
Logout Functionality:

- **Logout Process:** To log out, the client application clears the JWT token from localStorage. Since tokens are not stored server-side, no additional server-side logout logic is required.

9 .User Interface

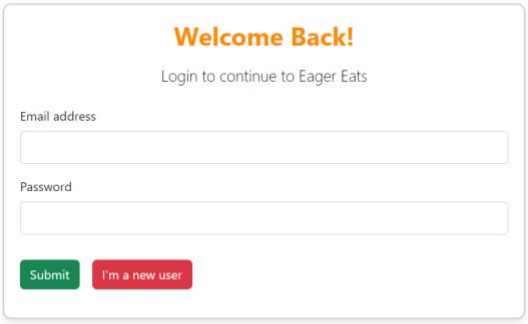
1. Home Page

- **Description: The main landing page of the application showcasing featured food items and categories.**
- **Screenshot:**



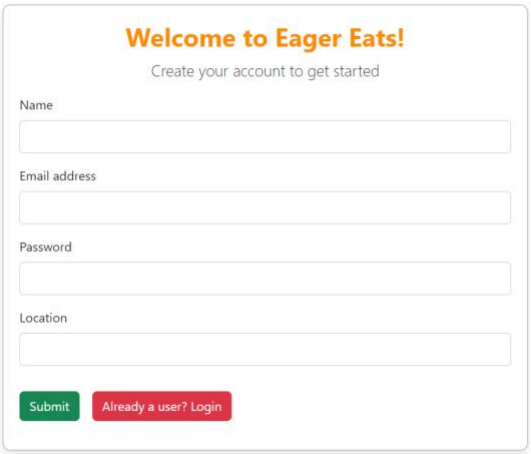
2. Login Page

- **Description: The user login interface where users can enter their credentials to access their account.**
- **Screenshot:**



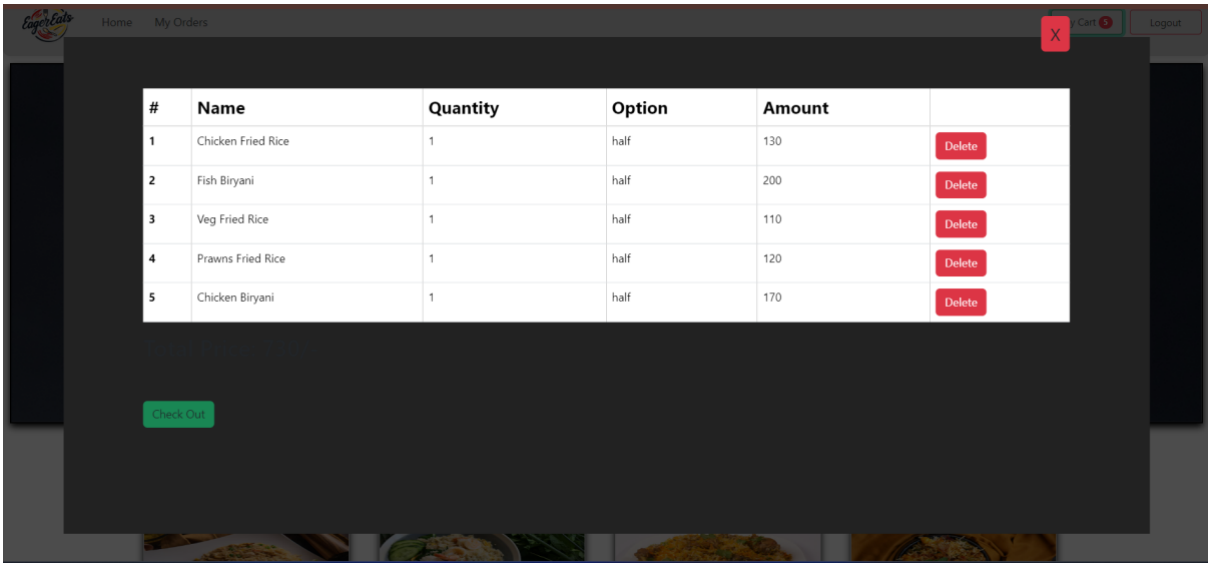
3. Signup Page

- **Description: The registration page where new users can create an account.**
- **Screenshot:**



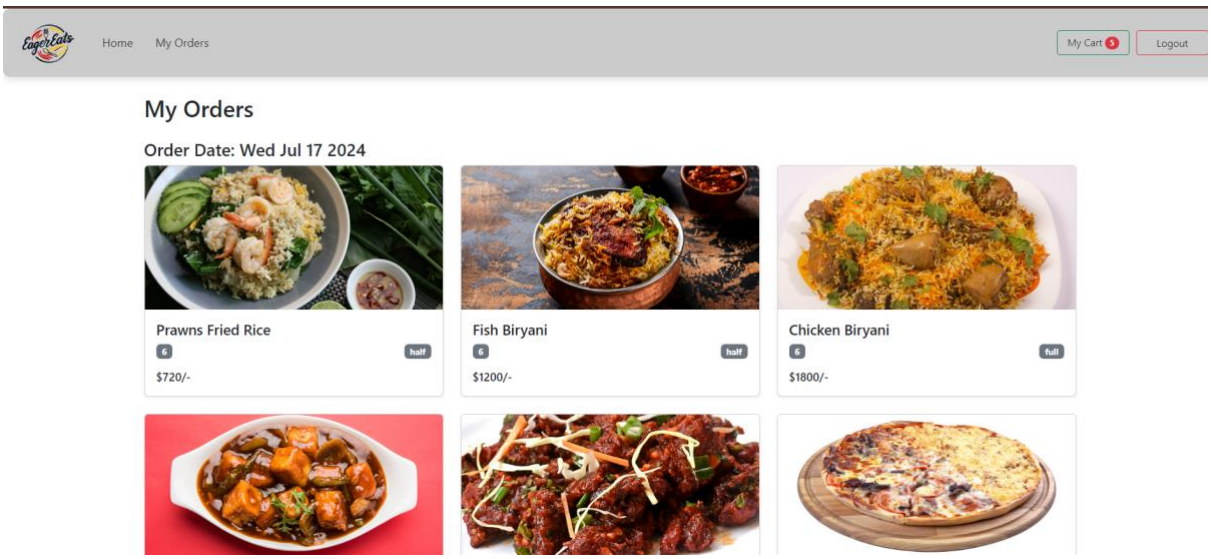
4. Cart Page

- **Description: The page where users can view and manage the items they have added to their cart.**
- **Screenshot:**



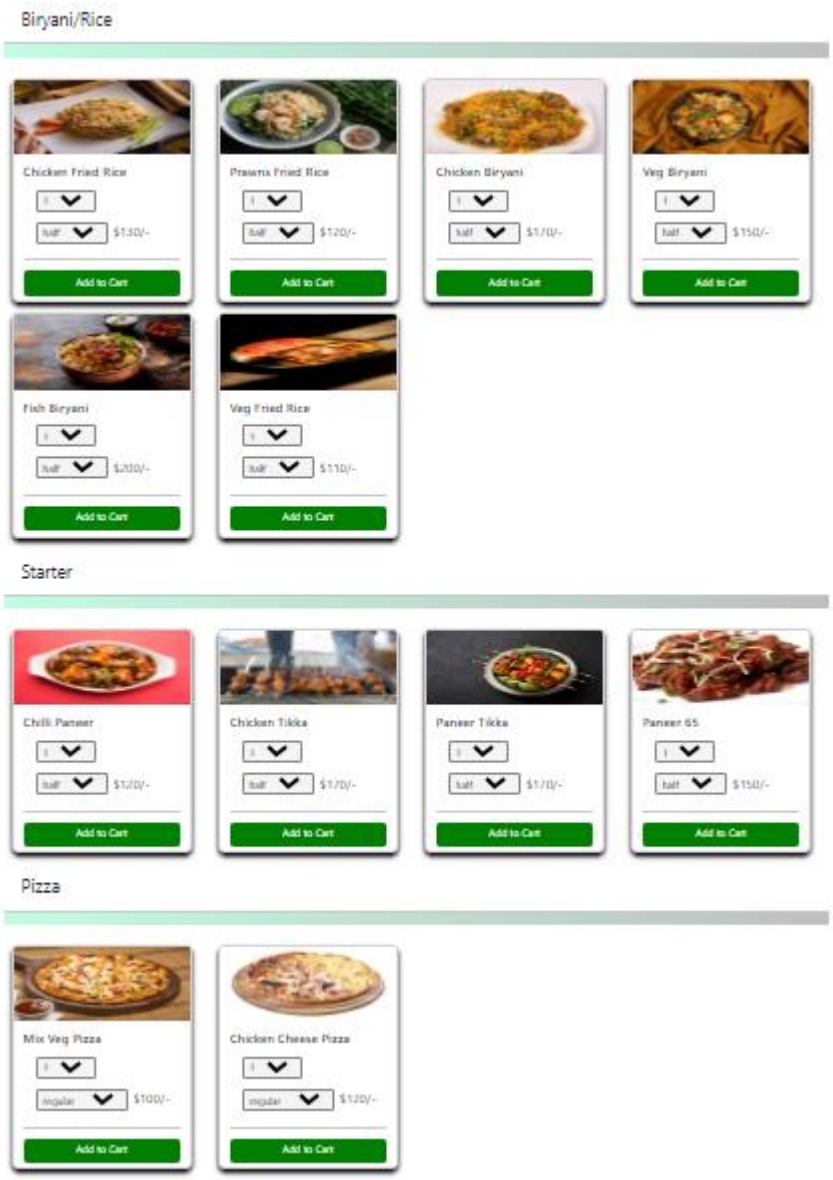
5. My Orders Page

- **Description: A page where users can view their previous orders and track order history.**
- **Screenshot:**



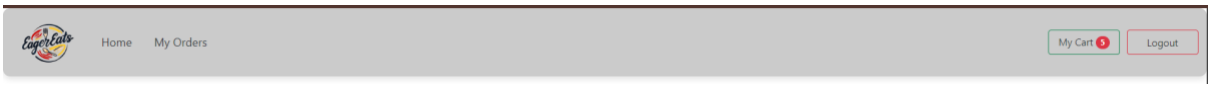
6. Food Items Display

- **Description: Shows how food items are presented to users, including categories and item details.**
- **Screenshot:**



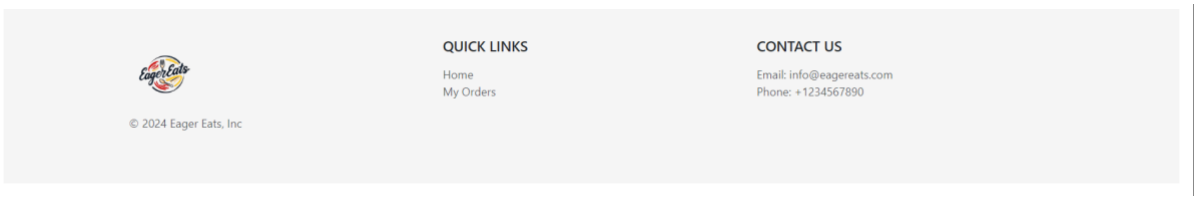
7. Navbar

- **Description: The navigation bar with links to different sections of the application.**
- **Screenshot:**



8. Footer

- **Description: The footer section containing contact information, links, and other relevant details.**
- **Screenshot:**



10. Testing

Testing Strategy

1. Unit Testing: Tests individual components and functions to ensure they work correctly.
2. Integration Testing: Ensures that different parts of the application work well together, including the frontend and backend.
3. End-to-End Testing: Simulates real user scenarios to ensure the whole application works from start to finish.
4. User Acceptance Testing (UAT): End-users test the application to make sure it meets their needs and requirements.
5. Regression Testing: Ensures that new code changes do not break existing features.

Tools Used

1. React Testing Library: For testing React components.
2. Thunderclient: For testing API endpoints. Thunderclient allows creating and sending HTTP requests to the backend server and verifying the responses.
3. MongoDB Compass: For verifying data stored in MongoDB. Compass is a graphical interface to explore and interact with MongoDB data.
4. Manual Testing: Testers perform manual testing to ensure the application works as expected, using predefined test cases.

11.Demo Link:

Demon Link

12. Known Issues

- **Login Session Expiry:** Users may need to log in again if their session expires while they are still using the app.
- **Cart Persistence:** Items in the cart are lost if the page is refreshed.
- **Responsive Design:** Some parts of the UI may not display correctly on smaller screens like mobile phones.

- **Order History:** The order history page can be slow if there are many past orders.
- **Form Validation:** The signup and login forms could better prevent incorrect data from being submitted.

13. Future Enhancements

- **Persistent Cart**: Save cart items even if the page is refreshed or the user logs out and logs back in.
- **Enhanced Security:** Add multi-factor authentication (MFA) for better account security.
- **Responsive Design Improvements:** Make sure all parts of the UI work well on all device sizes.
- **Order Tracking:** Allow users to track their orders in real-time, from preparation to delivery.
- **User Profile Management:** Add a page where users can update their information, change passwords, and view detailed order history.
- **Advanced Search:** Implement a search function with filters to help users find specific food items quickly.
- **Notifications:** Add push notifications to inform users about order status, promotions, and updates.
- **Review and Rating System:** Allow users to leave reviews and ratings for food items to improve quality feedback.