Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title:** Food Delivery System
- **Team Members:** Mohith Krishna - Frontend Development, Sankalp Sharma- Frontend Development, Mounika Saipu-Database Design, Sunil - Backend Development

2. Project Overview

- **Purpose:** The goal of the Online Food Ordering App project is to create a user-friendly platform that allows users to browse and order food from various restaurants seamlessly. The platform aims to streamline the ordering process by providing detailed information on food items, enabling users to customize their orders, and facilitating secure and efficient payment methods. Additionally, the project seeks to offer restaurant owners an easy way to manage their menus, track orders, and engage with customers. The overarching objective is to enhance the overall dining experience by leveraging modern web technologies and ensuring a smooth, hassle-free interaction for all users involved.

- **Features: User Authentication and Management**

- Secure user registration and login.
- Different user roles: Admin, Restaurant, and Customer.
- Profile management for users.

Restaurant Management

- Restaurant registration and profile setup.
- Approval and rejection of restaurant profiles by admins.
- Management of restaurant information, including name, contact details, and description.

Food Item Management

- Add, update, and delete food items.
- Detailed information for each food item, including name, category, description, price, and photo.

Cart Management

- Add food items to the cart with specified quantities.
- Update and remove items from the cart.
- View cart summary before placing an order.

Order Management

- Place orders directly from the cart.
- Track order status and history.
- Secure payment processing and status tracking.

Admin Panel

- Manage user accounts and roles.
- Approve or reject restaurant registrations.
- Monitor system usage and generate reports.

Responsive Design

- User-friendly interface compatible with various devices (desktop, tablet, mobile).
- Intuitive navigation and seamless user experience.

Notifications

- Email notifications for order confirmations and updates.
- Alerts for restaurants on new orders and status changes.

Search and Filter

- Search functionality to find restaurants and food items easily.
- Filter options based on categories, price range, and popularity.

Reviews and Ratings

- Customers can leave reviews and ratings for restaurants and food items.
- View aggregated ratings to make informed choices.

Analytics and Reporting

- Detailed analytics for restaurants on sales, popular items, and customer feedback.
- Reports for admins on user activity and system performance.

3. Architecture

- **Frontend:** The React frontend project for the food ordering app is designed with a clear separation of concerns, ensuring scalability and maintainability. The architecture comprises multiple directories and components, each responsible for specific functionalities, adhering to best practices in React development.

Directory Structure

1. `frontend`

- **node_modules**: Contains all the npm packages required for the project, generated and updated based on `package.json`.
- **public**: Holds static files such as images, icons, and the main HTML file.
    - `images`: Stores image assets like icons and logos.
    - `favicon.ico`: The icon displayed in the browser tab.
    - `index.html`: The main HTML file serving as the entry point for the React application.
    - `logo192.png`, `logo512.png`: Logo images for different screen resolutions.
    - `manifest.json`: Metadata for the web app, useful for Progressive Web App (PWA) capabilities.
    - `robots.txt`: Instructions for web crawlers on how to index the site.

2. `src`

- **components**: Contains reusable React components.
    - **AdminDashboard**: Components related to admin dashboard functionality.
    - **Payments**: Components handling the payment process.
    - **RestaurantDashboard**: Components for the restaurant's interface.
    - **UserDashboard**: Components for the user dashboard interface.
    - Common components such as `About.js`, `AnimatedFood.js`, `Carousel.js`, `Contact.js`, `Footer.js`, `Header.js`, `Login.js`, `MainPage.js`, `Menu.js`.
- **context**: Contains Context API files to manage global state across the application.
- **styles**: CSS files for styling different components.
    - `login.css`, `App.css`, `index.css`: CSS files for specific pages and the main app.
- **entry point**: The main files initializing and rendering the React application.
    - `index.js`: Entry point for the React application, rendering the App component into the DOM.
    - `App.js`: Main App component that serves as the root of the React application.
    - `App.test.js`: Test file for the App component.
    - `logo.svg`: SVG logo used in the application.
    - `reportWebVitals.js`: Utility for measuring performance metrics.
    - `setupTests.js`: Configuration file for setting up tests.

3. Configuration and Dependencies

- **.gitignore**: Specifies which files and directories to ignore in version control.
- **package-lock.json**: Automatically generated file locking the versions of dependencies for consistency.
- **package.json**: Lists the project dependencies and scripts.

Key Components and Functionality

1. **AdminDashboard, Payments, RestaurantDashboard, UserDashboard**

    - Components specific to different user roles and functionalities within the application.
2. **Common Components**
    - `About.js`, `AnimatedFood.js`, `Carousel.js`, `Contact.js`, `Footer.js`, `Header.js`, `Login.js`, `MainPage.js`, `Menu.js`: Handle common pages and functionalities such as about us, contact, header/footer, login, main page, and displaying the menu.
3. **Global State Management**
    - The `context` directory is used to manage global state using React's Context API, ensuring state is accessible throughout the app.
4. **Styling**
    - CSS files like `login.css`, `App.css`, `index.css` are used to style different components, maintaining a consistent look and feel across the application.
5. **Entry Point**
    - `index.js` initializes and renders the React application, while `App.js` serves as the root component encapsulating the entire app's structure and routing.

Routing

Main Routes

1. **Home Page**
    - **Path:** `/`
    - **Component:** `MainPage`
    - **Description:** The landing page of the application, typically showing an overview of the app and featured items.
2. **About Page**
    - **Path:** `/about`
    - **Component:** `About`
    - **Description:** A page providing information about the application and the team behind it.
3. **Animated Food**
    - **Path:** `/animated-food`
    - **Component:** `AnimatedFood`
    - **Description:** Page showcasing animated food items, possibly for promotions or special effects.
4. **Carousel**
    - **Path:** `/carousel`
    - **Component:** `Carousel`
    - **Description:** A page featuring a carousel/slider of images or items, often used for highlighting specials or new arrivals.
5. **Contact Page**
    - **Path:** `/contact`
    - **Component:** `Contact`
    - **Description:** Page containing contact information and possibly a form for users to reach out to the support team.

6. **Login Page**
   - **Path:** `/login`
   - **Component:** `Login`
   - **Description:** The login page for users to authenticate themselves.
7. **Menu Page**
   - **Path:** `/menu`
   - **Component:** `Menu`
   - **Description:** Page displaying the menu items available for ordering.
8. **Admin Dashboard**
   - **Path:** `/admin-dashboard`
   - **Component:** `AdminDashboard`
   - **Description:** Dashboard for admin users to manage the application, such as viewing orders, managing users, etc.
9. **User Dashboard**
   - **Path:** `/user-dashboard`
   - **Component:** `UserDashboard`
   - **Description:** Dashboard for regular users to view their orders, account details, etc.
10. **Restaurant Dashboard**
    - **Path:** `/restaurant-dashboard`
    - **Component:** `RestaurantDashboard`
    - **Description:** Dashboard for restaurant owners or staff to manage their restaurant's orders, menu items, etc.

Integration with Backend

The frontend of the Food Ordering App communicates with the backend APIs hosted on `[backend URL]`. This integration ensures smooth data exchange and functionality across different modules of the application.

Key Endpoints

1. **User Authentication**
   - **Endpoint:** `POST /api/user/login`
   - **Description:** Handles user login by validating credentials and returning a token for authenticated sessions.
2. **User Registration**
   - **Endpoint:** `POST /api/user/register`
   - **Description:** Registers a new user by saving user details to the database.
3. **Retrieve Orders**
   - **Endpoint:** `GET /api/orders`
   - **Description:** Fetches a list of orders for the authenticated user or admin.
4. **Create Order**
   - **Endpoint:** `POST /api/orders`
   - **Description:** Allows users to create a new order by submitting order details.
5. **Admin Dashboard**
   - **Endpoint:** `GET /api/admin/dashboard`
   - **Description:** Retrieves data and metrics for the admin dashboard.

6. **Restaurant Dashboard**
   - **Endpoint:** `GET /api/restaurant/dashboard`
   - **Description:** Retrieves data and metrics for the restaurant dashboard.

- **Backend:** The backend architecture of the Food Ordering App is designed to handle user authentication, manage data related to users, restaurants, food items, orders, and cart functionalities. It follows a modular approach, ensuring scalability, maintainability, and ease of development. The backend utilizes Node.js with the Express.js framework and MongoDB as the database.

Key Components

1. **Server Setup**
   - **Server Initialization**: Setting up an Express server.
   - **Middleware Configuration**: Configuring middleware for JSON parsing, CORS, authentication, error handling, etc.
   - **Routing**: Defining routes for various endpoints.
2. **Database Configuration**
   - **MongoDB**: NoSQL database to store data related to users, restaurants, food items, orders, and cart.
   - **Mongoose**: ODM (Object Data Modeling) library to interact with MongoDB, define schemas, and perform database operations.
3. **Models**
   - **User**: Schema and model for user data.
   - **Restaurant**: Schema and model for restaurant data.
   - **FoodItem**: Schema and model for food item data.
   - **Cart**: Schema and model for cart data.
   - **Order**: Schema and model for order data.
4. **Controllers**
   - **User Controller**: Handles user-related operations such as registration, login, profile management.
   - **Restaurant Controller**: Manages restaurant data, registration, and approvals.
   - **FoodItem Controller**: CRUD operations for food items.
   - **Cart Controller**: Operations related to the cart, such as adding, updating, and removing items.
   - **Order Controller**: Handles order creation, retrieval, and payment status.
5. **Routes**
   - **User Routes**: Endpoints for user operations.
   - **Restaurant Routes**: Endpoints for restaurant operations.
   - **FoodItem Routes**: Endpoints for food item management.
   - **Cart Routes**: Endpoints for cart operations.
   - **Order Routes**: Endpoints for order management.
6. **Authentication and Authorization**
   - **JWT**: JSON Web Tokens for user authentication.
   - **Middleware**: Middleware functions to protect routes and ensure only authenticated users can access certain endpoints.
7. **Error Handling**

   - **Global Error Handler**: Centralized error handling mechanism to manage and respond to errors consistently.

- **Database: Database Schema and Interactions with MongoDB**

- **Overview**

- The Food Ordering App's backend utilizes MongoDB as its database to store and manage data for users, restaurants, food items, carts, and orders. The database schema is designed to efficiently handle various operations related to these entities, ensuring data integrity and smooth interactions.

1. **User Schema**

   Fields:

- `email`: User's email address (unique).
- `password`: User's password.
- `username`: User's name.
- `userType`: Type of user (e.g., customer, admin).

2. Restaurant Schema

Fields:

- `restaurantName`: Name of the restaurant.
- `ownerName`: Name of the owner.
- `email`: Email address of the restaurant.
- `phone`: Phone number.
- `address`: Physical address.
- `description`: Description of the restaurant.
- `restaurantPicture`: Path to the restaurant's image.
- `approved`: Approval status of the restaurant.
- `rejected`: Rejection status of the restaurant.
- `submissionTimestamp`: Timestamp when the restaurant was submitted for approval.

3. FoodItem Schema

Fields:

- `itemName`: Name of the food item.
- `itemCategory`: Category of the food item (e.g., appetizer, main course).
- `itemDescription`: Description of the food item.
- `itemPrice`: Price of the food item.
- `itemPhoto`: Path to the food item's image.

4. Cart Schema

- **Fields**:
  - `userId`: Reference to the user who owns the cart.
  - `items`: Array of food items in the cart.
    - `foodItem`: Reference to the food item.
    - `quantity`: Quantity of the food item.

5. **Order Schema**

   Fields:

- `userId`: Reference to the user who placed the order.
- `items`: Array of ordered food items.
  - `foodItem`: Reference to the food item.
  - `quantity`: Quantity of the food item.
- `paymentStatus`: Payment status of the order (true for paid, false for unpaid).

4. Setup Instructions

- **Prerequisites:** Node.js and MongoDB
- **Installation:**

  Step 1: Create a folder on your PC and open folder in terminal.

  Step 2: git clone <repository link> run this command in the terminal and project will be cloned into the selected folder.

  Step 3: Open the folder in VS Code or any code editor and open terminal in it.\

  Step 4: Run npm install in both backend and frontend folder to install all the required dependencies.

5. Folder Structure

- **Client:** frontend/
- ├── public/
- │   ├── index.html   # Main HTML file
- │   ├── favicon.ico   # Favicon
- │   └── manifest.json   # Web app manifest
- ├── src/
- │   ├── assets/

- │ │ ├── images/ # Images used in the app
- │ │ └── styles/ # Global styles and CSS files
- │ │ ├── base.css
- │ │ ├── variables.css
- │ │ └── ...
- │ ├── components/
- │ │ ├── common/ # Common reusable components
- │ │ │ ├── Button.js
- │ │ │ ├── Input.js
- │ │ │ └── ...
- │ │ ├── layout/ # Layout components (Header, Footer, etc.)
- │ │ │ ├── Header.js
- │ │ │ ├── Footer.js
- │ │ │ └── Sidebar.js
- │ │ ├── user/ # Components related to user functionalities
- │ │ │ ├── LoginForm.js
- │ │ │ ├── RegisterForm.js
- │ │ │ └── Profile.js
- │ │ ├── restaurant/ # Components related to restaurant functionalities
- │ │ │ ├── RestaurantList.js
- │ │ │ ├── RestaurantDetails.js
- │ │ │ └── AddRestaurant.js
- │ │ ├── food/ # Components related to food item functionalities
- │ │ │ ├── FoodList.js
- │ │ │ ├── FoodDetails.js

- │ │ │ └── AddFoodItem.js
- │ │ ├── cart/   # Components related to cart functionalities
- │ │ │ ├── Cart.js
- │ │ │ └── CartItem.js
- │ │ ├── order/   # Components related to order functionalities
- │ │ │ ├── OrderList.js
- │ │ │ ├── OrderDetails.js
- │ │ │ └── PlaceOrder.js
- │ │ └── ...   # Additional feature-specific components
- │ ├── context/   # Context providers for state management
- │ │ ├── AuthContext.js
- │ │ ├── CartContext.js
- │ │ └── ...
- │ ├── hooks/   # Custom hooks
- │ │ ├── useAuth.js
- │ │ ├── useCart.js
- │ │ └── ...
- │ ├── pages/   # Pages for routing
- │ │ ├── HomePage.js
- │ │ ├── LoginPage.js
- │ │ ├── RegisterPage.js
- │ │ ├── RestaurantPage.js
- │ │ ├── FoodPage.js
- │ │ ├── CartPage.js
- │ │ ├── OrderPage.js

- │   │   └── ...
- │   ├── services/   # API service functions
- │   │   ├── authService.js
- │   │   ├── userService.js
- │   │   ├── restaurantService.js
- │   │   ├── foodService.js
- │   │   ├── cartService.js
- │   │   └── orderService.js
- │   ├── App.js   # Main app component
- │   ├── index.js   # Entry point for React
- │   ├── routes.js   # Route definitions
- │   └── ...   # Additional configuration files
- ├── package.json   # Package configuration
- └── README.md   # Project documentation
-

- **Server:**backend/
- ├── config/
- │   ├── db.js   # Database configuration and connection
- │   └── ...   # Other configuration files
- ├── controllers/
- │   ├── userController.js   # Controller for user operations
- │   ├── restaurantController.js # Controller for restaurant operations
- │   ├── foodItemController.js # Controller for food item operations

- │   ├── cartController.js   # Controller for cart operations
- │   └── orderController.js   # Controller for order operations
- ├── models/
- │   ├── User.js   # User model
- │   ├── Restaurant.js   # Restaurant model
- │   ├── FoodItem.js   # Food item model
- │   ├── Cart.js   # Cart model
- │   └── Order.js   # Order model
- ├── routes/
- │   ├── userRoutes.js   # Routes for user operations
- │   ├── restaurantRoutes.js # Routes for restaurant operations
- │   ├── foodItemRoutes.js   # Routes for food item operations
- │   ├── cartRoutes.js   # Routes for cart operations
- │   └── orderRoutes.js   # Routes for order operations
- ├── middleware/
- │   ├── authMiddleware.js   # Middleware for authentication
- │   └── errorMiddleware.js   # Middleware for error handling
- ├── utils/
- │   ├── generateToken.js   # Utility for generating JWT tokens
- │   └── ...   # Other utility functions
- ├── .env   # Environment variables
- ├── app.js   # Express app setup
- └── server.js   # Server initialization and configuration

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
    - o **Frontend:** `npm start` in the client directory.
    - o **Backend:** `npm start` in the server directory.

7. API Documentation

Backend API Endpoints

User Endpoints

1. Register User

- URL: `/api/users/register`

- Method: `POST`

- Parameters:

- `email` (String, required)

- `password` (String, required)

- `username` (String, required)

- `userType` (String, required)

- Example Response:

```
    {
   "message": "User registered successfully",
   "user": { "id": "user_id", "email": "user@example.com" }
  }
```

2. Login User

- URL: `/api/users/login`

- Method: `POST`
- Parameters:
  - `email` (String, required)
  - `password` (String, required)
- Example Response:

```
     {
   "message": "User logged in successfully",
   "token": "jwt_token"
}
```

Restaurant Endpoints

1. Add Restaurant
   - URL: `/api/restaurants`
   - Method: `POST`
   - Parameters:
     - `restaurantName` (String, required)
     - `ownerName` (String, required)
     - `email` (String, required)
     - `phone` (String, required)
     - `address` (String, required)
     - `description` (String, required)
     - `restaurantPicture` (String)
   - Example Response:

```
 {
 "message": "Restaurant added successfully",
 "restaurant": { "id": "restaurant_id", "name": "Restaurant Name" }
}
```

2. Get Restaurants

- **URL: `/api/restaurants`**
- **Method: `GET`**
- **Parameters: None**
- **Example Response:**

```
  [
 {
  "id": "restaurant_id",
  "restaurantName": "Restaurant Name",
  "ownerName": "Owner Name",
  "email": "owner@example.com",
  "phone": "1234567890",
  "address": "Restaurant Address",
  "description": "Restaurant Description",
  "restaurantPicture": "path/to/image",
  "approved": false,
  "rejected": false
 },
  ]
```

Food Item Endpoints

1. Add Food Item

- **URL: `/api/food`**
- **Method: `POST`**
- **Parameters:**
  - **`itemName` (String, required)**
  - **`itemCategory` (String, required)**
  - **`itemDescription` (String, required)**
  - **`itemPrice` (Number, required)**
  - **`itemPhoto` (String)**
- **Example Response:**

```
    {
  "message": "Food item added successfully",
  "foodItem": { "id": "food_item_id", "name": "Food Item Name" }
 }
```

2. Get Food Items

- **URL: `/api/food`**
- **Method: `GET`**
- **Parameters: None**
- **Example Response:**

```
 [
    {
      "id": "food_item_id",
      "itemName": "Food Item Name",
      "itemCategory": "Category",
      "itemDescription": "Description",
      "itemPrice": 10,
      "itemPhoto": "path/to/image"
    },
  ]
```

Cart Endpoints

1. Add to Cart

- **URL: `/api/cart`**
- **Method: `POST`**
- **Parameters:**
  - **`userId` (String, required)**
  - **`items` (Array, required)**
- **Example Response:**

```
   {
    "message": "Items added to cart successfully",
```

"cart": { "id": "cart_id", "userId": "user_id", "items": [...] }

}

2. Get Cart

- URL: `/api/cart/:userId`

- Method: `GET`

- Parameters: `userId` (String, required)

- Example Response:

```
{
  "id": "cart_id",
  "userId": "user_id",
  "items": [...]}
```

Order Endpoints

1. Place Order

- URL: `/api/orders`

- Method: `POST`

- Parameters:

  - `userId` (String, required)

  - `items` (Array, required)

- `paymentStatus` (Boolean, required)

-Example Response:

```
{
  "message": "Order placed successfully",
  "order": { "id": "order_id", "userId": "user_id", "items": [...] }
}
```

2. Get Orders

- URL: `/api/orders/:userId`

- Method: `GET`

- Parameters: `userId` (String, required)

- Example Response:

```
[
  {
    "id": "order_id",
    "userId": "user_id",
    "items": [...],
    "paymentStatus": true,
    "createdAt": "timestamp",
    "updatedAt": "timestamp"
  },
    ]
```

8. Authentication

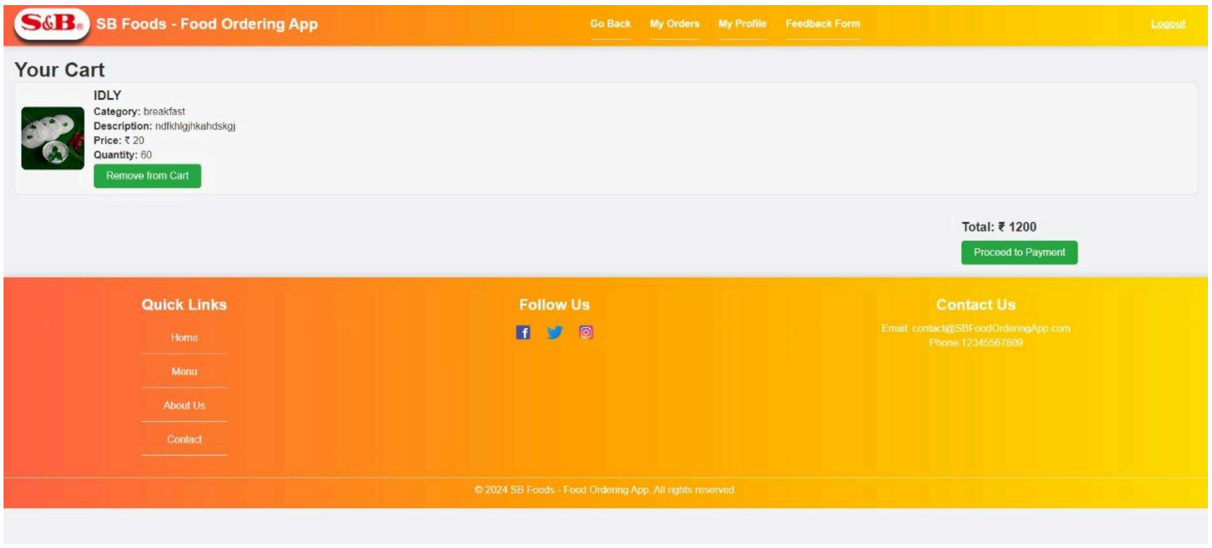
Authentication: Handled using JWT (JSON Web Tokens).

- **Login: User credentials are validated, and a JWT token is issued.**
- **Token Storage: JWT is stored in localStorage or cookies on the client side.**

Authorization:

- **Middleware: Protects routes by verifying the JWT token in requests.**
- **Role-Based Access: User roles (e.g., admin, user, restaurant) are checked to allow or deny access to specific endpoints.**

9. User Interface


SB Foods - Food Ordering App

Go Back | My Orders | My Profile | Feedback Form | Logout

Your Cart

IDLY
Category: breakfast
Description: ndfkhlgjhkahdskgj
Price: ₹ 20
Quantity: 60

Remove from Cart

Total: ₹ 1200

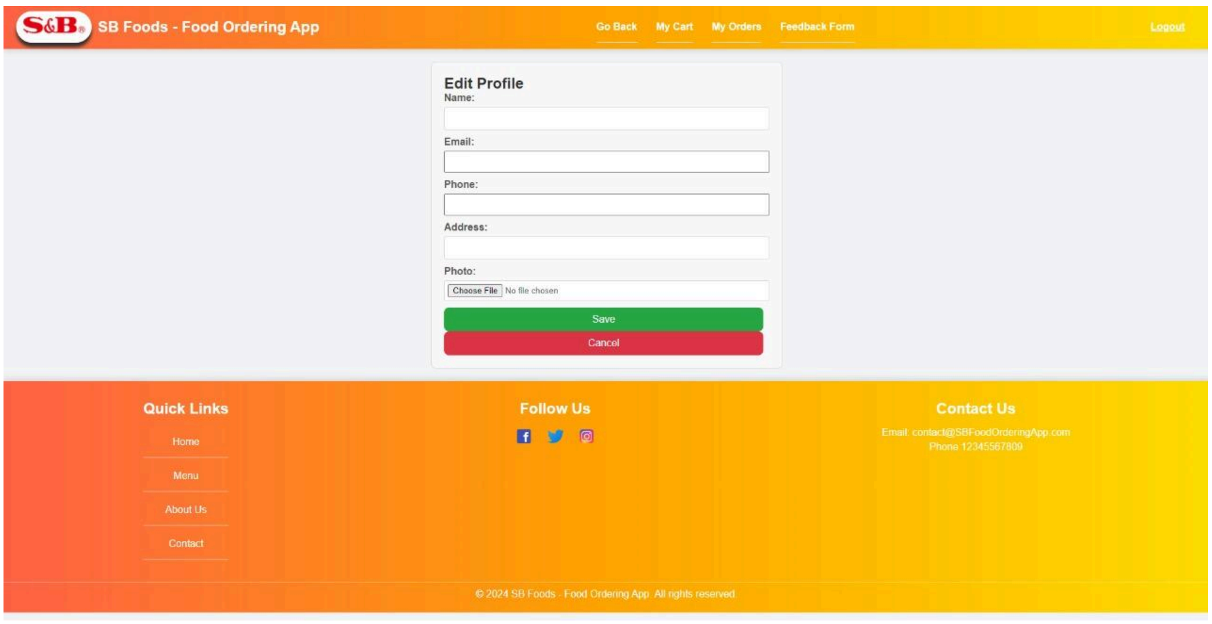
Proceed to Payment

Quick Links: Home, Menu, About Us, Contact

Follow Us

Contact Us
Email: contact@SBFoodOrderingApp.com
Phone 12345567809

© 2024 SB Foods - Food Ordering App. All rights reserved.


SB Foods - Food Ordering App

Go Back | My Cart | My Orders | Feedback Form | Logout

Edit Profile

Name:

Email:

Phone:

Address:

Photo:
Choose File No file chosen

Save

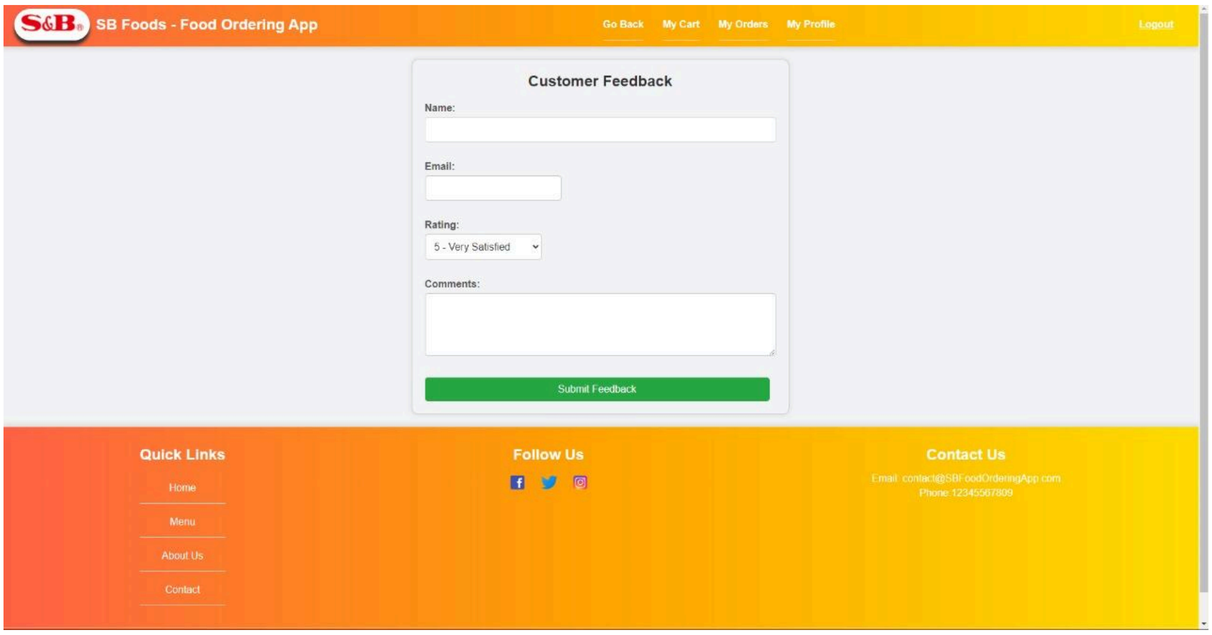
Cancel

Quick Links: Home, Menu, About Us, Contact

Follow Us

Contact Us
Email: contact@SBFoodOrderingApp.com
Phone 12345567809

© 2024 SB Foods - Food Ordering App. All rights reserved.



SB Foods - Food Ordering App

Go Back | My Cart | My Orders | My Profile | Logout

Customer Feedback

Name:

Email:

Rating: 5 - Very Satisfied

Comments:

Submit Feedback

Quick Links

Home

Menu

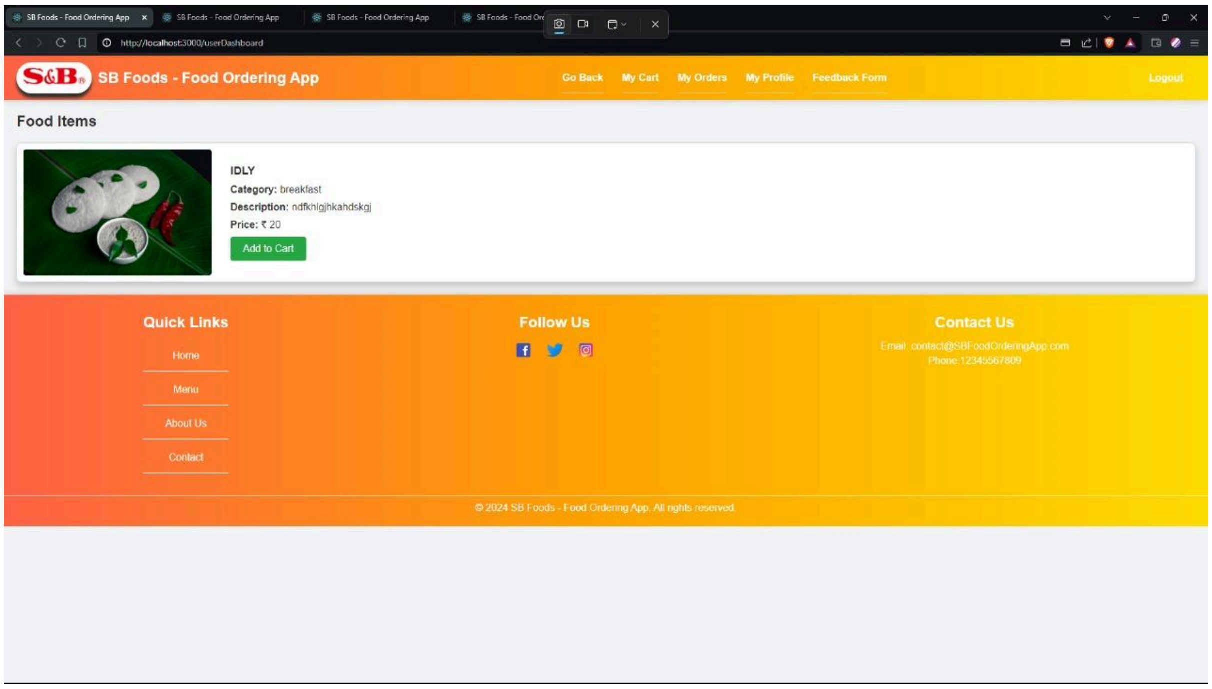
About Us

Contact

Follow Us

Contact Us

Email: contact@SBFoodOrderingApp.com
Phone:12345567809



SB Foods - Food Ordering App | SB Foods - Food Ordering App | SB Foods - Food Ordering App | SB Foods - Food Or

http://localhost:3000/userDashboard

SB Foods - Food Ordering App

Go Back | My Cart | My Orders | My Profile | Feedback Form | Logout

Food Items

IDLY
Category: breakfast
Description: ndfkhlgjhkahdskgj
Price: ₹ 20
Add to Cart

Quick Links

Home

Menu

About Us

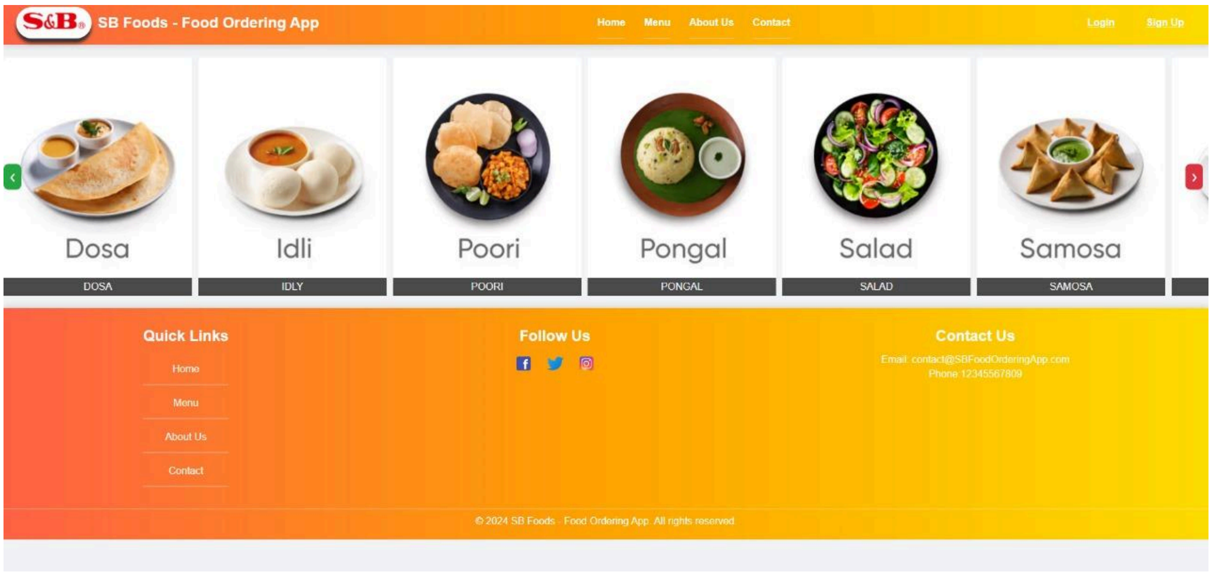
Contact

Follow Us

Contact Us

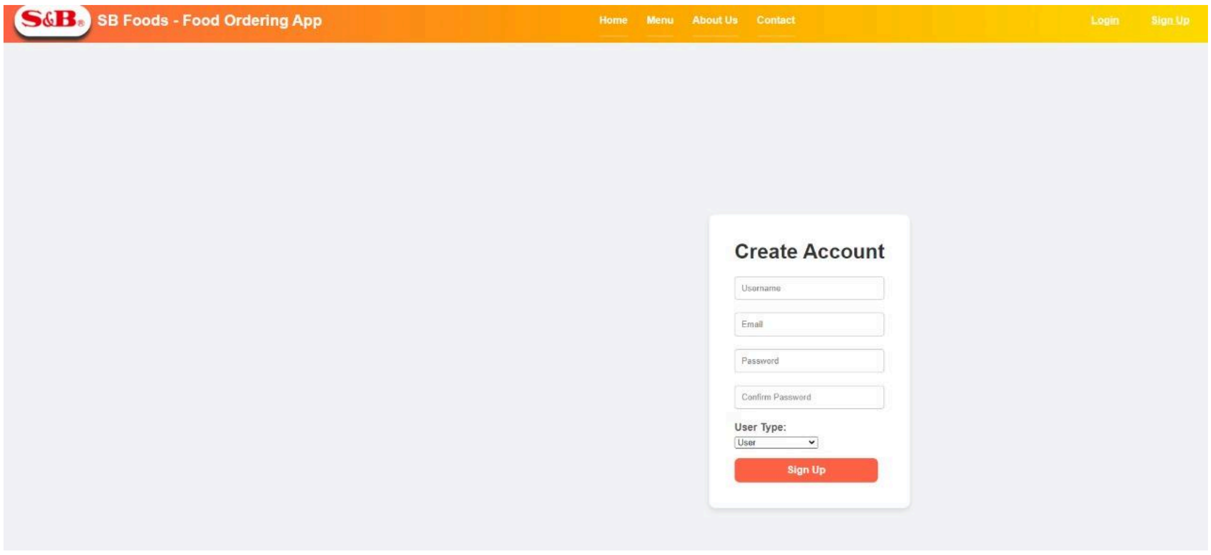
Email: contact@SBFoodOrderingApp.com
Phone:12345567809

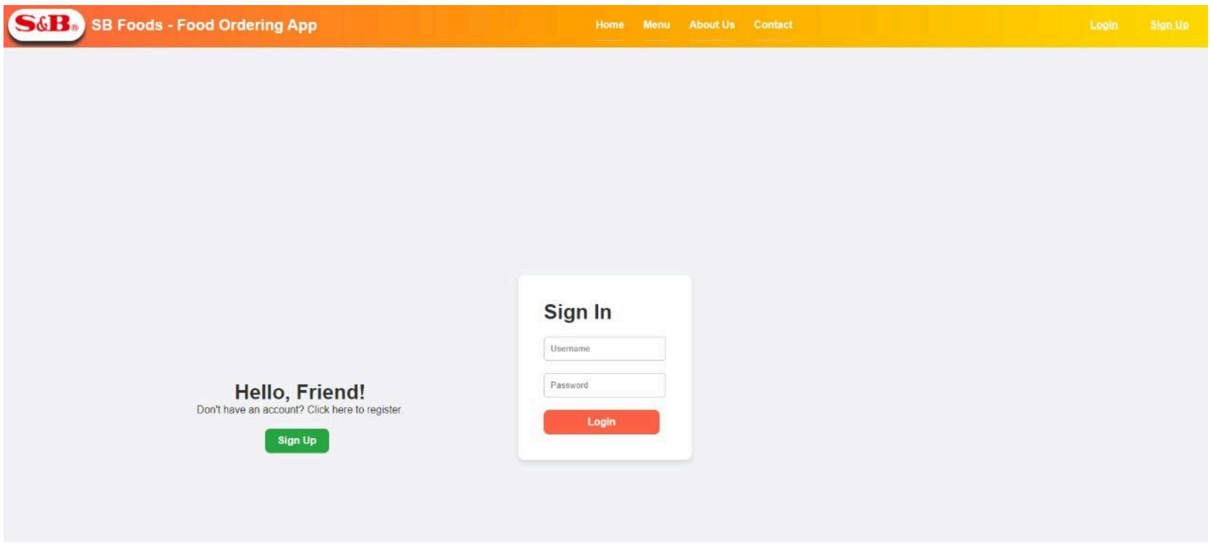
© 2024 SB Foods - Food Ordering App. All rights reserved.



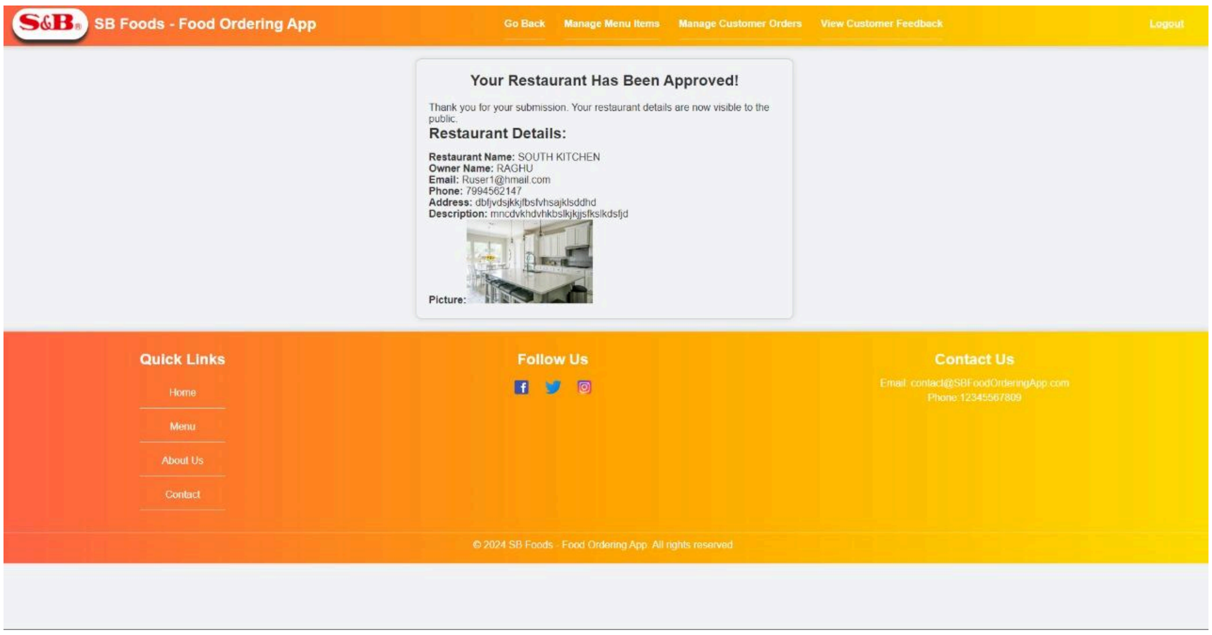
Email: contact@SBFoodOrderingApp.com
Phone 12345567809

© 2024 SB Foods - Food Ordering App. All rights reserved.





Don't have an account? Click here to register.



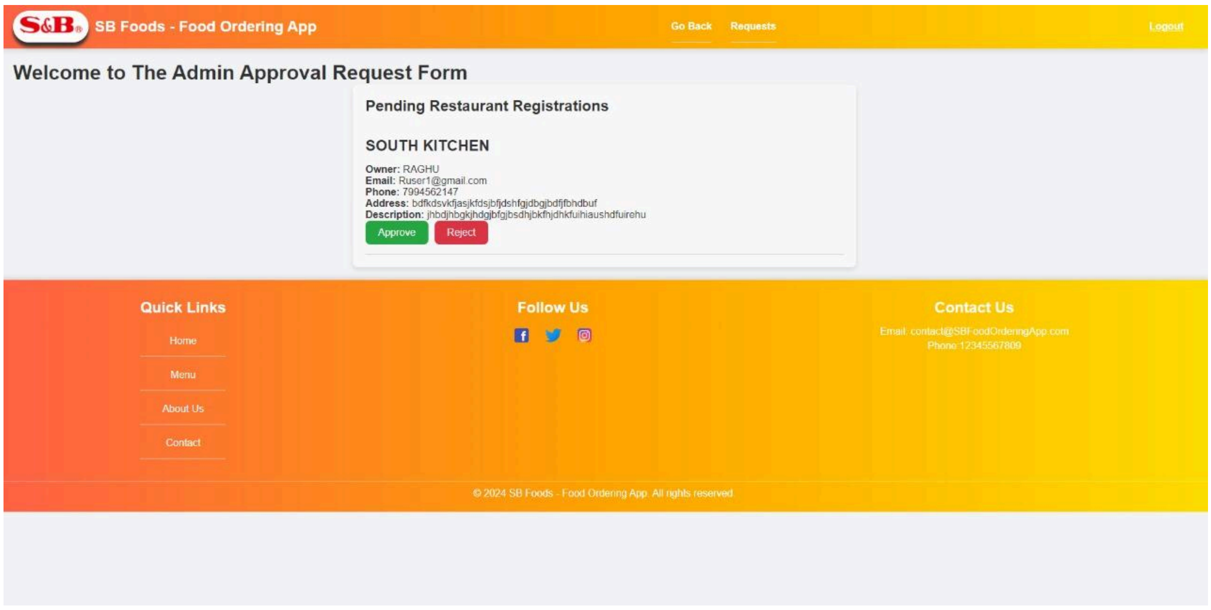
Your Restaurant Has Been Approved!

Thank you for your submission. Your restaurant details are now visible to the public.

Restaurant Details:

Restaurant Name: SOUTH KITCHEN
Owner Name: RAGHU
Email: Ruser1@hmail.com
Phone: 7994562147
Address: dbfjvdsjkkjfbsfvhsajklsddhd
Description: mncdvkhdvhkbslkjkjjsfkslkdsfjd
Picture:

Quick Links
Home
Menu
About Us
Contact
Follow Us
Contact Us
Email: contact@SBFoodOrderingApp.com
Phone:12345567809
© 2024 SB Foods - Food Ordering App. All rights reserved.


SB Foods - Food Ordering App
Go Back
Requests
Logout

Welcome to The Admin Approval Request Form

Pending Restaurant Registrations

SOUTH KITCHEN

Owner: RAGHU
Email: Ruser1@gmail.com
Phone: 7994562147
Address: bdfkdsvkfjasjkfdsjbfjdshfgjdbgjbdfjfbhdbuf
Description: jhbdjhbgkjhdgjbfgjbsdhjbkfhjdhkfuihiaushdfuirehu

Approve
Reject
Quick Links
Home
Menu
About Us
Contact
Follow Us
Contact Us
Email: contact@SBFoodOrderingApp.com
Phone:12345567809
© 2024 SB Foods - Food Ordering App. All rights reserved.

10. Testing

<u>Strategy</u>

1. Unit Testing: Test individual units/components of the application for expected functionality.
2. Integration Testing: Test the interaction between different parts of the system to ensure they work together as expected.
3. End-to-End (E2E) Testing: Simulate real user scenarios to test the entire application flow from start to finish.

4. Manual Testing: Perform exploratory testing to identify any issues not covered by automated tests.

Tools

1. **Jest**: Used for unit and integration testing in the backend.
2. **Mocha & Chai**: Sometimes used for more complex backend testing scenarios.
3. **Supertest**: For testing HTTP endpoints in the backend.
4. **React Testing Library**: Used for unit and integration testing of React components.
5. **Cypress**: For end-to-end testing, simulating user interactions in a real browser environment.
6. **Postman**: For manual API testing and creating automated tests for API endpoints.

11. Screenshots or Demo

Demonstration Of Food Ordering App-20240721_210037-Meeting Recording.…

12. Known Issues

- None

13. Future Enhancements

AI-Powered Recommendations:

- **Personalized Suggestions:** Use machine learning algorithms to suggest dishes based on user preferences, order history, and dietary restrictions.
- **Predictive Analytics:** Predict peak order times and adjust delivery resources accordingly to reduce wait times.

Enhanced User Experience:

- **Voice Command Integration:** Allow users to place orders and navigate the app through voice commands.
- **Augmented Reality (AR) Menus:** Use AR to visualize dishes and ingredients before ordering.

Improved Delivery Logistics:

- **Dynamic Routing:** Implement real-time traffic data to optimize delivery routes and reduce delivery times.

- **Automated Delivery:** Explore the use of drones or autonomous vehicles for faster and more efficient deliveries.

Advanced Payment Options:

- **Cryptocurrency Payments:** Enable payments using cryptocurrencies for tech-savvy users.
- **Flexible Payment Plans:** Offer installment payment options for larger orders or subscriptions.

-