OrderOnTheGo: Your On-Demand Food Ordering Solution

A Project Report

submitted in partial fulfillment of the requirements

of

FULL STACK DEVELOPER
(MERN Stack)
Internship

with

SMARTBRIDGE
in collaboration **APSCHE**

by

MUTTI REDDY SHALOM VISHAL

Team ID: LTVIP2026TMIDS79872

shalomnani2004@gmail.com

SRI VENKATESWARA COLLEGE OF ENGINEERING,

TIRUPATI

Project Table of Contents

1. INTRODUCTION

1.1 Project Title:

OrderOnTheGo: Your On-Demand Food Ordering Solution.

1.2 Team Members:

Team Size: 05

1. Mutti Reddy Shalom Vishal – Team Lead And Software Tester.
2. Namasani Surya Teja – Frontend Developer (React.js Developer).
3. M S Mahesh – Backend Developer.
4. Konduru Sreeram – Backend Developer.
5. Ontela Bharath – Database Administrator

2. Project Overview

2.1 Purpose

The purpose of *OrderOnTheGo* is to provide a convenient, fast, and reliable online food ordering platform that connects customers with restaurants.

The system aims to:

- Reduce waiting time for customers
- Provide easy access to restaurant menus
- Enable seamless online ordering
- Improve order management for restaurants
- Support digital transformation for small and local food businesses

This project solves the problem of manual order handling, long queues, and lack of real-time order tracking by offering a centralized web-based solution.

2.2 Features

The key features of **OrderOnTheGo** include:

User Features:

- User Registration and Login
- Browse Restaurants
- View Food Items with Prices
- Add to Cart / Remove from Cart
- Update Item Quantity
- Secure Checkout
- Order Confirmation
- View Order History

Admin Features:

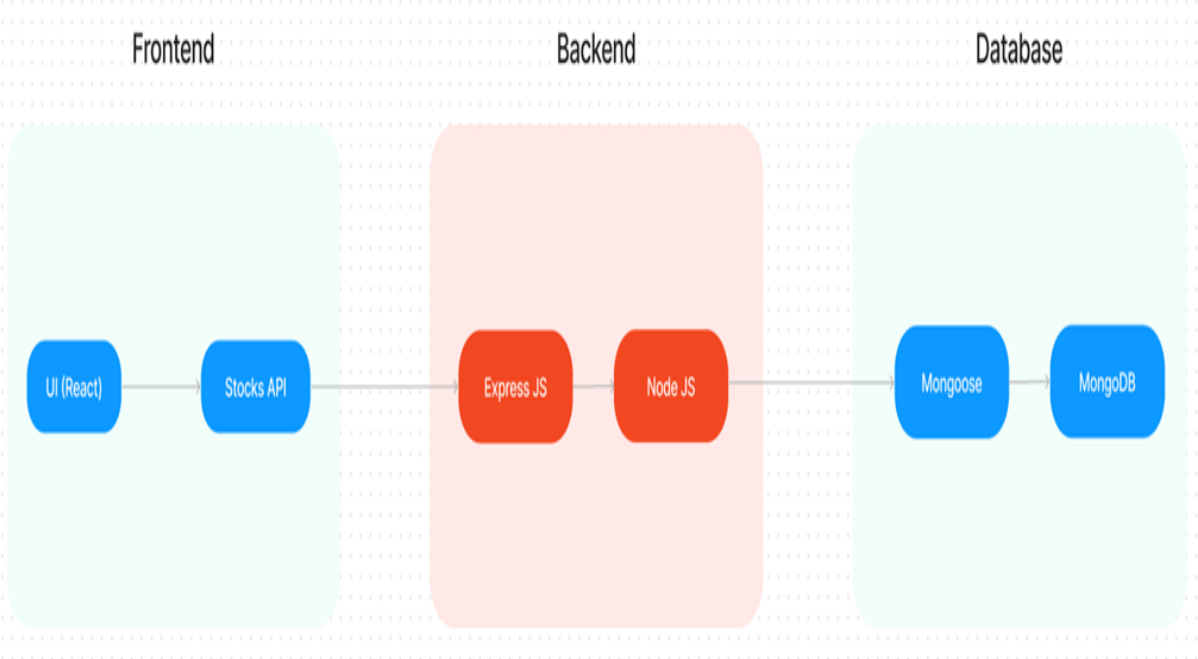
- Admin Login
- Add / Update / Delete Food Items
- Manage Restaurant Listings
- View and Manage Customer Orders

Security Features:

- JWT-based Authentication
- Role-Based Access Control (User/Admin)
- Input Validation and Error Handling

3. Architecture

The architecture of **OrderOnTheGo : Your On-Demand Food Ordering Solution** follows a three-tier architecture model consisting of Frontend, Backend, and Database layers. This layered approach ensures scalability, maintainability, and secure data handling.



3.1 Frontend

The frontend is developed using **React.js** to provide a dynamic and responsive user interface.

Responsibilities:

- User Registration and Login Interface
- Restaurant and Menu Display
- Cart Management
- Order Placement and Confirmation
- Admin Dashboard Interface
- Form Validation and Error Handling
- API Integration with Backend

Key Characteristics:

- Component-based architecture
- Responsive design for different screen sizes
- Real-time cart updates
- Secure token storage for authentication

The frontend communicates with the backend through RESTful API calls.

3.2 Backend

The backend is developed using **Node.js** and **Express.js** to handle business logic and server-side operations.

Responsibilities:

- Handling API requests and responses
- User authentication using JWT
- Order processing logic
- Cart management logic
- Admin operations (Add, Update, Delete items)
- Error handling and validation

Key Characteristics:

- RESTful API architecture
- Middleware-based request handling
- Secure authentication and authorization
- Modular route structure

The backend acts as a bridge between the frontend and the database.

3.3 Database

The database used is **MongoDB**, a NoSQL document-based database.

Responsibilities:

- Storing user data
- Managing restaurant and food item details
- Storing order information
- Maintaining order history

Collections:

- Users
- Restaurants
- FoodItems
- Orders

Key Characteristics:

- Schema modeling using Mongoose
- Efficient CRUD operations
- Scalable and flexible data structure

4. Setup Instructions

This section describes the prerequisites and installation steps required to set up and run the project **OrderOnTheGo : Your On-Demand Food Ordering Solution**.

4.1 Prerequisites

To develop and execute the full-stack food ordering application, the following software and tools are required:

1. Node.js and npm

Node.js is required to run JavaScript on the server side. npm (Node Package Manager) is used to install project dependencies.

Download: https://nodejs.org/en/download/

Installation Guide: https://nodejs.org/en/download/package-manager/

---

2. MongoDB

MongoDB is used as the database to store user details, restaurant information, and order data.

Download: https://www.mongodb.com/try/download/community

Installation Guide: https://docs.mongodb.com/manual/installation/

MongoDB can be installed locally or accessed using MongoDB Atlas (Cloud Database).

---

3. Express.js

Express.js is a backend framework used for routing and API development.

Installation command:

```
npm install express
```

---

4. React.js

React.js is used for building the frontend user interface.

Installation Guide:

https://reactjs.org/docs/create-a-new-react-app.html

---

5. Mongoose (Database Connectivity)

Mongoose is used to connect the Node.js server with MongoDB and perform CRUD operations.

Installation command:

```
npm install mongoose
```

---

6. Git (Version Control)

Git is used for tracking changes and collaboration.

Download: https://git-scm.com/downloads

7. Development Environment
Recommended code editors:
Visual Studio Code
Sublime Text
WebStorm

4.2 Installation
Follow the steps below to set up and run the project locally.
Step 1: Clone the Repository
Open terminal or command prompt and run:
git clone https://github.com/harsha-vardhan-reddy-07/Food-Ordering-App-MERN

Step 2: Navigate to Project Directory
cd Food-Ordering-App-MERN

Step 3: Install Dependencies
Install required packages:
npm install
If the project contains separate frontend and backend folders, run npm install inside each folder.

Step 4: Configure Environment Variables
Create a .env file in the backend folder and configure the following:
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_secret_key

Step 5: Start the Application
To start the backend server:
npm start
To start the frontend:
npm run dev

Step 6: Access the Application
Open your web browser and navigate to:
http://localhost:3000
or
http://localhost:5173
If setup is successful, the homepage of **OrderOnTheGo** will be displayed.
.

5. FOLDER STRUCTURE

The project **OrderOnTheGo : Your On-Demand Food Ordering Solution** follows a modular full-stack architecture divided into two main folders:

- **Client (Frontend – React.js)**
- **Server (Backend – Node.js & Express.js)**

5.1 Client

The client folder contains the frontend of the application developed using React.js.

```
client/
│
├── node_modules/
├── public/
├── src/
│   ├── components/
│   │   ├── Footer.jsx
│   │   ├── Login.jsx
│   │   ├── Navbar.jsx
│   │   ├── PopularRestaurants.jsx
│   │   ├── Register.jsx
│   │   └── Restaurants.jsx
│   │
│   ├── context/
│   ├── images/
│   ├── pages/
│   │   ├── admin/
│   │   │   ├── Admin.jsx
│   │   │   ├── AllOrders.jsx
│   │   │   ├── AllProducts.jsx
│   │   │   ├── AllRestaurants.jsx
│   │   │   └── AllUsers.jsx
│   │   │
│   │   ├── customer/
│   │   │   ├── Cart.jsx
│   │   │   ├── CategoryProducts.jsx
│   │   │   ├── IndividualRestaurant.jsx
│   │   │   └── Profile.jsx
│   │   │
│   │   ├── restaurant/
```

```
│   │   │   ├── EditProduct.jsx
│   │   │   ├── NewProduct.jsx
│   │   │   ├── RestaurantHome.jsx
│   │   │   ├── RestaurantMenu.jsx
│   │   │   └── RestaurantOrders.jsx
│   │
│   ├── Authentication.jsx
│   ├── Home.jsx
│   ├── styles/
│   ├── App.css
│   └── App.js
```

🔹 **Description:**

- **components/** – Reusable UI components (Navbar, Footer, Login, Register, etc.)
- **pages/** – Role-based pages (Admin, Customer, Restaurant)
- **context/** – Global state management (if using Context API)
- **images/** – Static assets
- **Authentication.jsx** – Handles login/register logic
- **App.js** – Main React application entry point
- **styles/** – CSS styling files

5.2 Server

The server folder contains the backend developed using Node.js and Express.js.

```
server/
│
├── node_modules/
├── index.js
├── Schema.js
├── package.json
└── package-lock.json
```

🔹 **Description:**

- **index.js** – Main server file, handles API routes and server configuration
- **Schema.js** – Defines MongoDB schema and models
- **package.json** – Contains project dependencies and scripts
- **node_modules/** – Installed backend dependencies

6. RUNNING THE APPLICATION:

This section explains the steps required to run the OrderOnTheGo : Your On-Demand Food Ordering Solution application on a local development environment.

6.1 Start the Backend Server

1. Open terminal or command prompt.
2. Navigate to the server folder:

cd server

3. Install backend dependencies (if not already installed):

npm install

4. Start the backend server:

npm start

or

node index.js

5. The server will start running on:

http://localhost:5000

(default port, configurable in .env file)

6.2 Start the Frontend Application

1. Open a new terminal window.
2. Navigate to the client folder:

cd client

3. Install frontend dependencies:

npm install

4. Start the React development server:

npm run dev

or

npm start

5. The application will run on:

http://localhost:3000

or

http://localhost:5173

(depending on configuration)

6.3 Access the Application

- Open your web browser.
- Navigate to the frontend URL.
- You will see the homepage of OrderOnTheGo.
- Users can register, login, browse restaurants, and place orders.
- Admin users can login to manage products and orders.

6.4 Important Notes

- Ensure MongoDB service is running before starting the backend.
- Make sure the .env file contains correct database connection details.
- Backend must be running before frontend for API communication.

---

7. API DOCUMENTATION:

This section describes the RESTful APIs developed for **OrderOnTheGo : Your On-Demand Food Ordering Solution**.
The backend is built using **Node.js and Express.js**, and it communicates with MongoDB to perform CRUD operations.

7.1 Base URL
http://localhost:5000/api

7.2 Authentication APIs

🔹 1. Register User

- **Method:** POST
- **Endpoint:** /auth/register
- **Description:** Creates a new user account.

Request Body:

```
{
  "name": "John",
  "email": "john@example.com",
  "password": "123456",
  "role": "customer"
}
```

Response:

```
{
  "message": "User registered successfully"
}
```

🔹 2. Login User

- **Method:** POST
- **Endpoint:** /auth/login
- **Description:** Authenticates user and returns JWT token.

Request Body:

```
{
  "email": "john@example.com",
  "password": "123456"
}
```

Response:

```
{
  "token": "jwt_token_here",
  "role": "customer"
}
```

7.3 Restaurant APIs

◈ 3. Get All Restaurants

- **Method:** GET
- **Endpoint:** /restaurants
- **Description:** Fetches all restaurants.

◈ 4. Get Restaurant by ID

- **Method:** GET
- **Endpoint:** /restaurants/:id
- **Description:** Fetches details of a specific restaurant.

◈ 5. Add Restaurant (Admin Only)

- **Method:** POST
- **Endpoint:** /restaurants
- **Authorization:** Admin Token Required

7.4 Product (Food Item) APIs

◈ 6. Get All Products

- **Method:** GET
- **Endpoint:** /products

◈ 7. Add New Product (Restaurant/Admin)

- **Method:** POST
- **Endpoint:** /products

◈ 8. Update Product

- **Method:** PUT
- **Endpoint:** /products/:id

◈ 9. Delete Product

- **Method:** DELETE
- **Endpoint:** /products/:id

7.5 Cart APIs

◈ 10. Add to Cart

- **Method:** POST
- **Endpoint:** /cart/add

◈ 11. Get User Cart

- **Method:** GET
- **Endpoint:** /cart/:userId

◈ 12. Remove from Cart

- **Method:** DELETE
- **Endpoint:** /cart/:id

7.6 Order APIs

◈ 13. Place Order

- **Method:** POST
- **Endpoint:** /orders

◈ 14. Get User Orders

- **Method:** GET
- **Endpoint:** /orders/user/:userId

◈ 15. Get All Orders (Admin)

- **Method:** GET
- **Endpoint:** /orders

7.7 Security

- JWT-based authentication
- Role-based authorization (Admin / Customer / Restaurant)
- Password hashing using bcrypt
- Middleware validation for protected routes

8. AUTHENTICATION:

Authentication in OrderOnTheGo : Your On-Demand Food Ordering Solution ensures that only authorized users can access protected features such as placing orders, managing products, and viewing admin dashboards.

The system uses JWT (JSON Web Tokens) for secure authentication and role-based access control.

8.1 Authentication Process

The authentication flow follows these steps:

1. User registers with valid details.
2. User logs in using email and password.
3. Backend verifies credentials.
4. If valid, a JWT token is generated.
5. The token is sent to the frontend.
6. The frontend stores the token (localStorage).
7. The token is included in API requests for protected routes.

8.2 Registration

- Users provide name, email, password, and role.
- Password is hashed using bcrypt before storing in the database.
- Duplicate email validation is implemented.
- After successful registration, user can login.

8.3 Login

- User enters email and password.
- Backend compares password using bcrypt.
- If credentials are valid:
    - JWT token is generated.
    - Role (Admin / Customer / Restaurant) is returned.
- If invalid:
    - Error message is displayed.

8.4 JWT Token Structure

The token contains:

- User ID
- User Role
- Expiration Time

Example payload:

```
{
  "userId": "123456",
  "role": "customer",
  "exp": "expiration_time"
}
```

8.5 Role-Based Authorization

The system supports three roles:

👤 Customer

- Browse restaurants
- Add to cart
- Place orders
- View order history

🏨 Restaurant

- Add/Edit/Delete products
- View restaurant orders

🛠️ Admin

- Manage users
- Manage restaurants
- View all orders
- Monitor system data

Middleware is used in backend to:

- Verify JWT token
- Check user role
- Allow or restrict access accordingly

8.6 Security Measures

- Password hashing using bcrypt
- JWT expiration time
- Protected API routes
- Input validation
- Error handling for unauthorized access

9. USER INTERFACE

The User Interface (UI) of the application is designed to be **simple, responsive, and user-friendly**. It is developed using **React.js** to ensure a smooth and interactive experience.

1. Design Goals

- Clean and minimal layout
- Easy navigation
- Responsive for desktop and mobile
- Fast loading and smooth transitions

---

2. Main Pages

1. Home Page

- Introduction to the application
- Navigation bar with links (Login, Register, Dashboard)
- Brief description of features

2. Registration Page

- Input fields:
    - Name
    - Email
    - Password
- Form validation (required fields, valid email format)
- Submit button
- Success/Error messages

3. Login Page

- Email and Password fields
- Authentication handling
- Error message for invalid credentials
- Redirect to Dashboard after successful login

4. Dashboard

- Displays user-specific data
- CRUD operation buttons:
    - Add
    - Edit
    - Delete
    - View
- Logout button

---

3. UI Components

- Navbar (Navigation bar)
- Forms (Input fields, Buttons)
- Cards / Tables for displaying data
- Modal popup for editing data
- Alerts for success/error messages

---

4. Styling

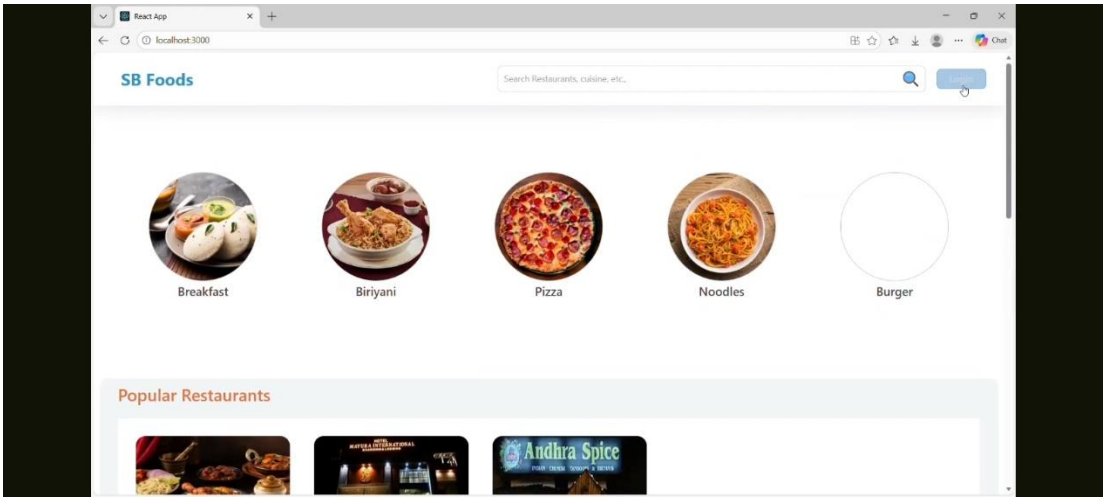
- CSS / Tailwind / Bootstrap (based on project setup)
- Consistent color theme
- Proper spacing and alignment
- Responsive grid layout

---

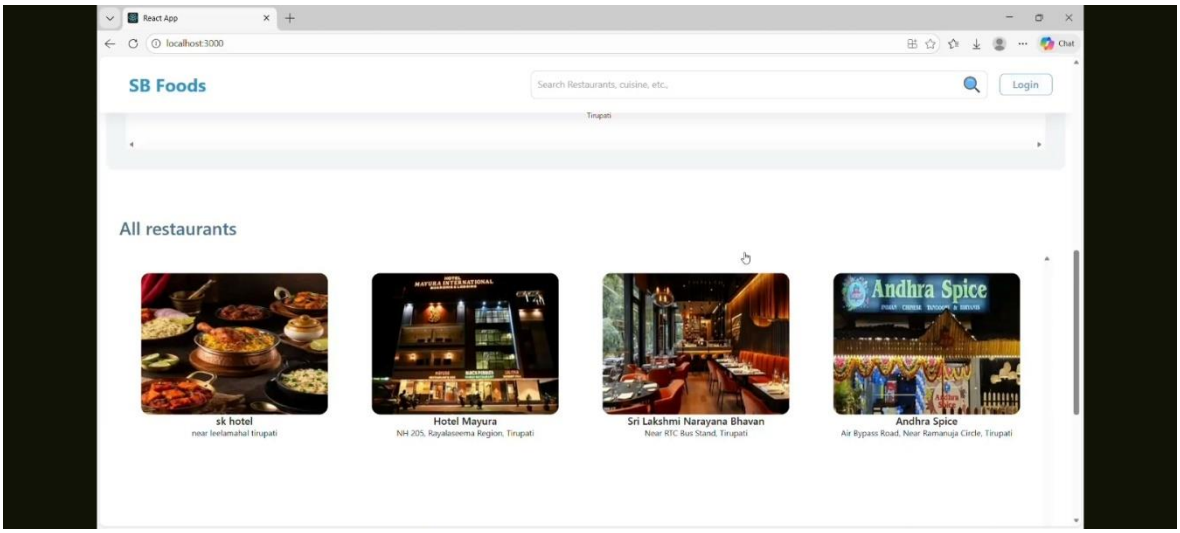
5. User Experience Features

- Client-side validation
- Loading indicators during API calls
- Confirmation dialog before delete
- Secure password masking
- Protected routes for authenticated users

USER INTERFACE FOR FOOD:



USER INTERFACE OF RESTAURANTS:



10. TESTING

Testing ensures that the application works correctly, securely, and efficiently. In our MERN Stack (MongoDB, Express, React, Node.js) project, testing was performed at multiple levels to validate both frontend and backend functionalities.

11.1 Types of Testing Performed

1. Unit Testing

- Individual components and functions were tested separately.
- Backend API routes were tested to verify correct request and response handling.
- Form validation logic was tested to ensure proper error handling and input validation.

2. Integration Testing

- Verified communication between React frontend, Express/Node.js backend, and MongoDB database.
- Ensured smooth data flow between client and server.
- Confirmed that APIs correctly interact with the database.

3. Functional Testing

The following user workflows were tested end-to-end:

- User Registration
- User Login
- Adding new records
- Viewing records
- Updating records
- Deleting records

Each functionality was tested with valid and invalid inputs.

4. UI Testing

- Verified responsive design across different screen sizes.
- Checked correct rendering of components.
- Validated form fields and error messages.

5. Security Testing

- Verified password encryption using bcrypt.
- Tested JWT-based authentication.
- Ensured protected routes are accessible only after login.

11.2 Sample Test Scenarios

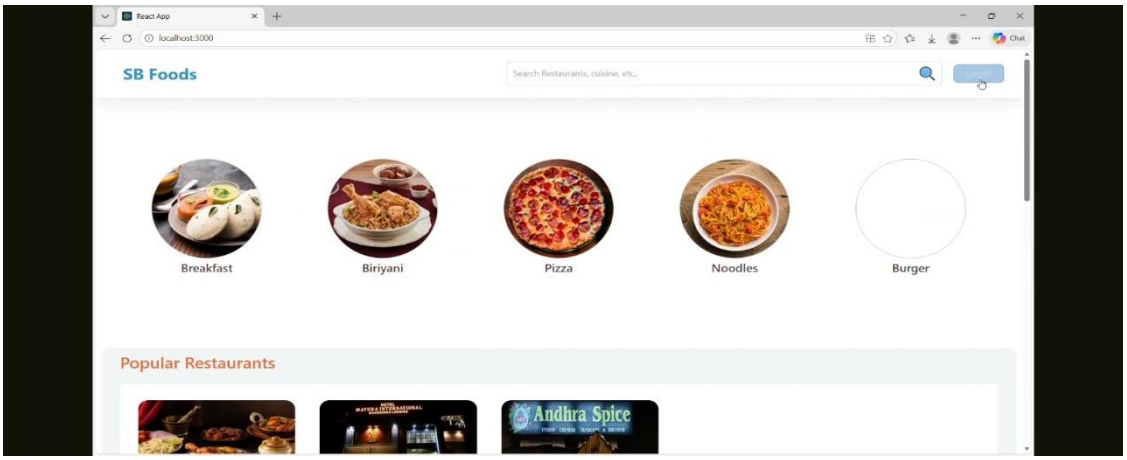
- When valid user details are entered during registration, the account should be created successfully.
- When correct login credentials are entered, the user should be redirected to the dashboard.
- When incorrect login credentials are entered, an appropriate error message should be displayed.
- When valid data is submitted, the record should be stored in the database.
- When a record is deleted, it should be removed permanently from the database.
- All test cases were executed, and the results matched the expected outcomes.

11.3 Tools Used for Testing

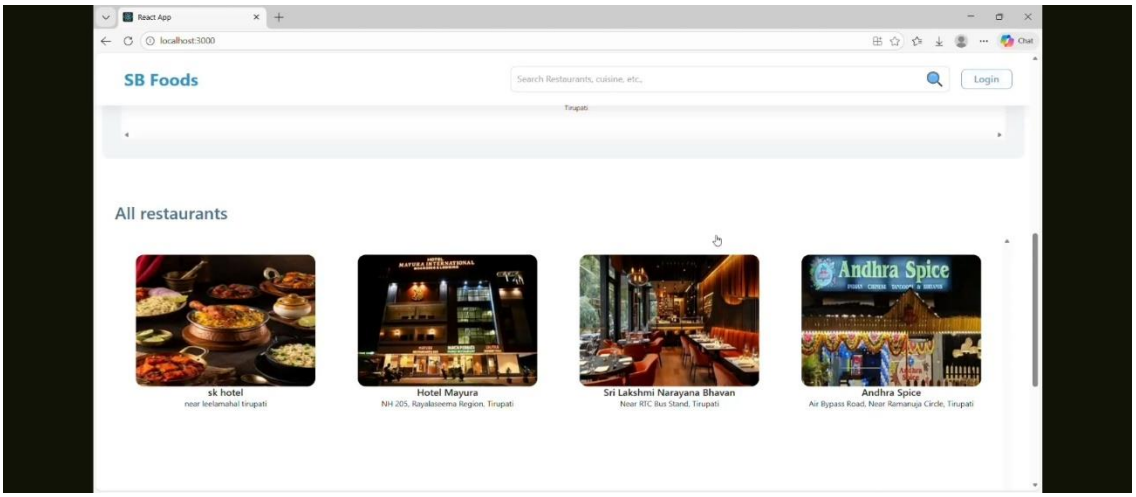
- Browser Developer Tools for frontend testing
- Postman for API testing
- MongoDB Compass for database verification

11. SCREENSHOTS

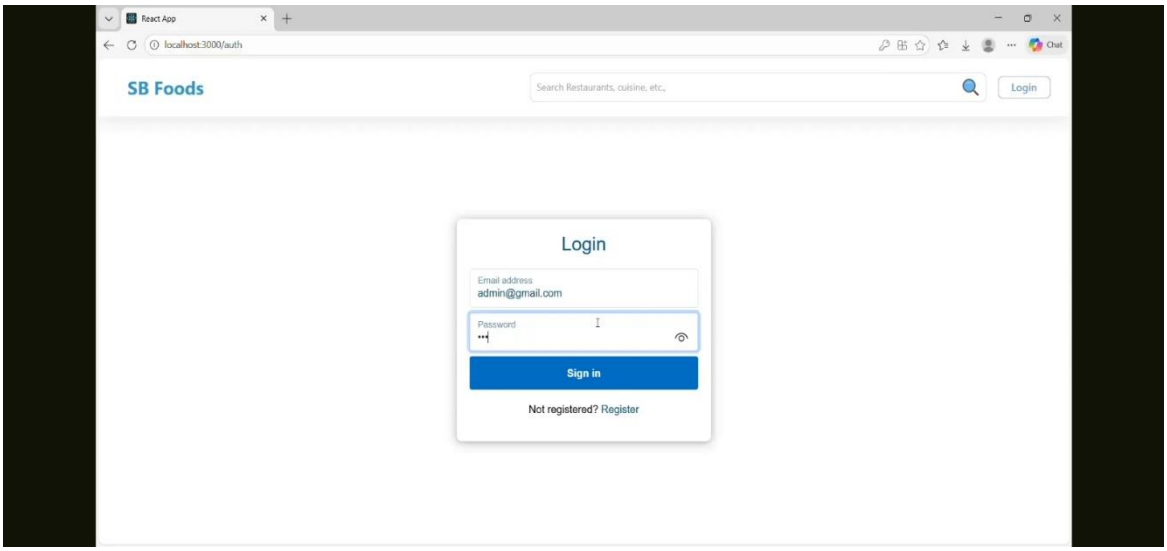
A. HOME PAGE



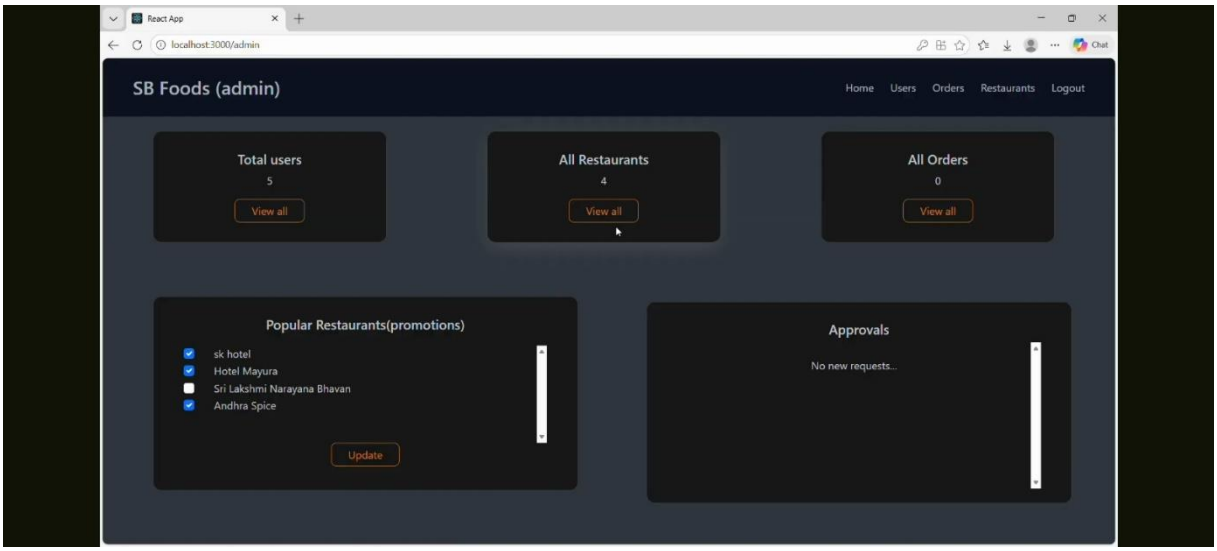
B. RESTAURANT PAGE



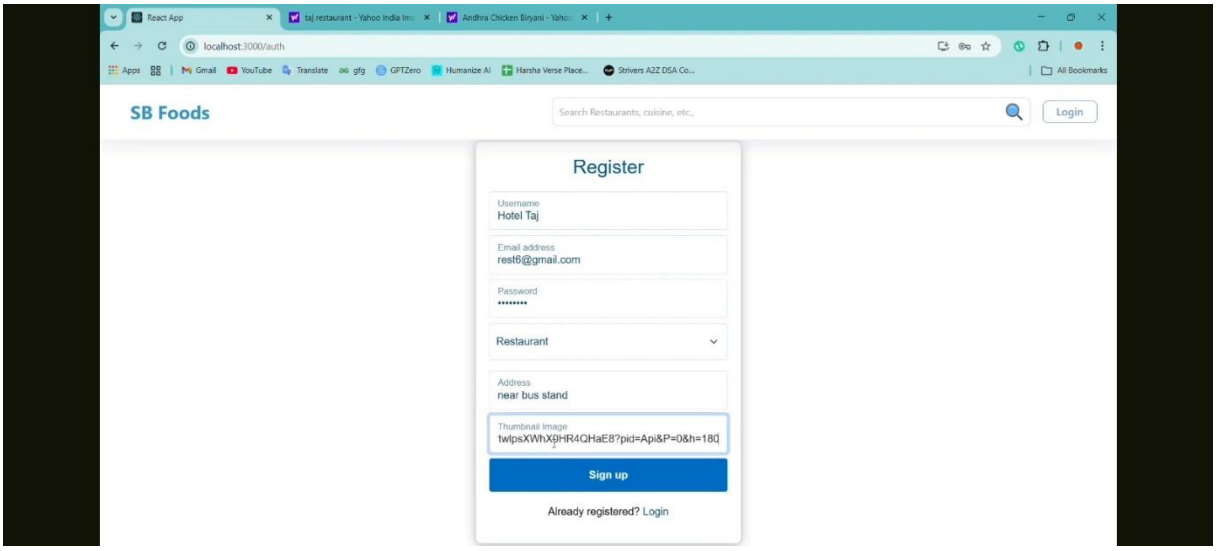
C. ADMIN LOGIN



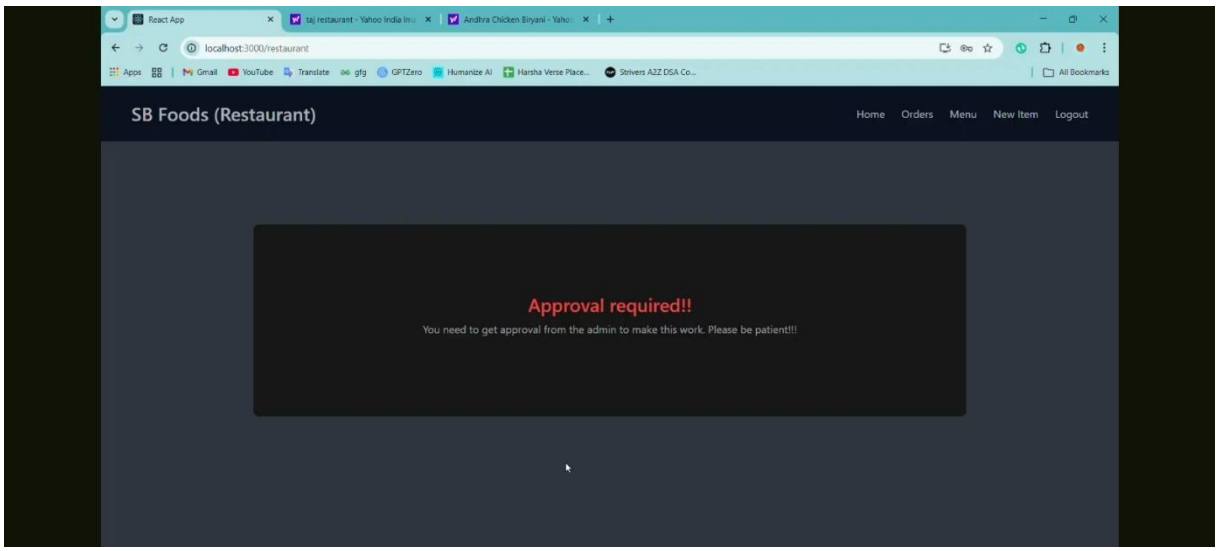
D. ADMIN DATABASE



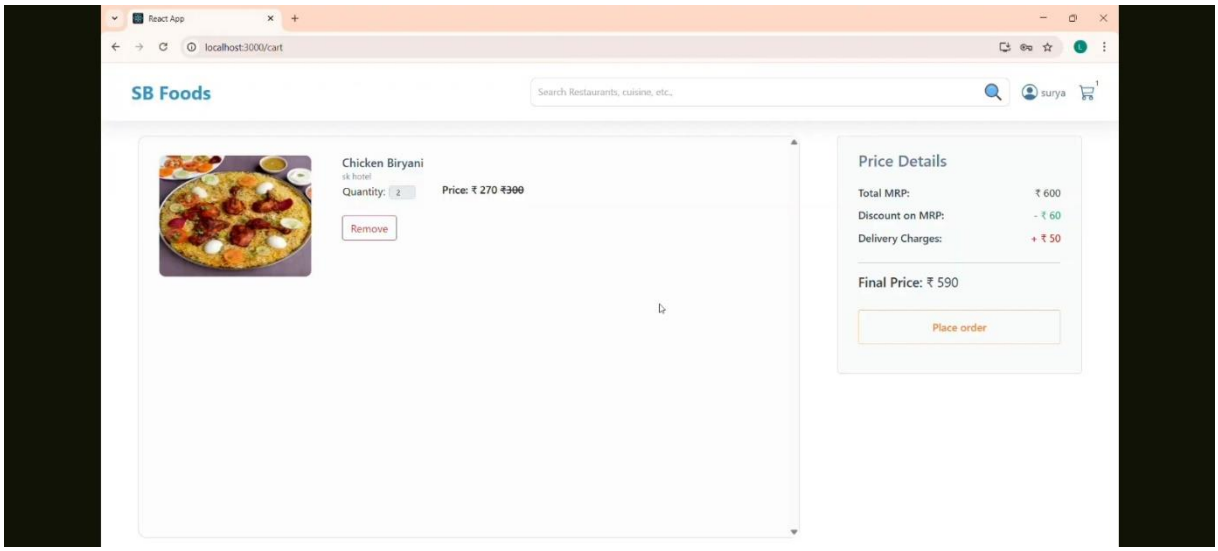
E. RESTAURANT REGISTRATION



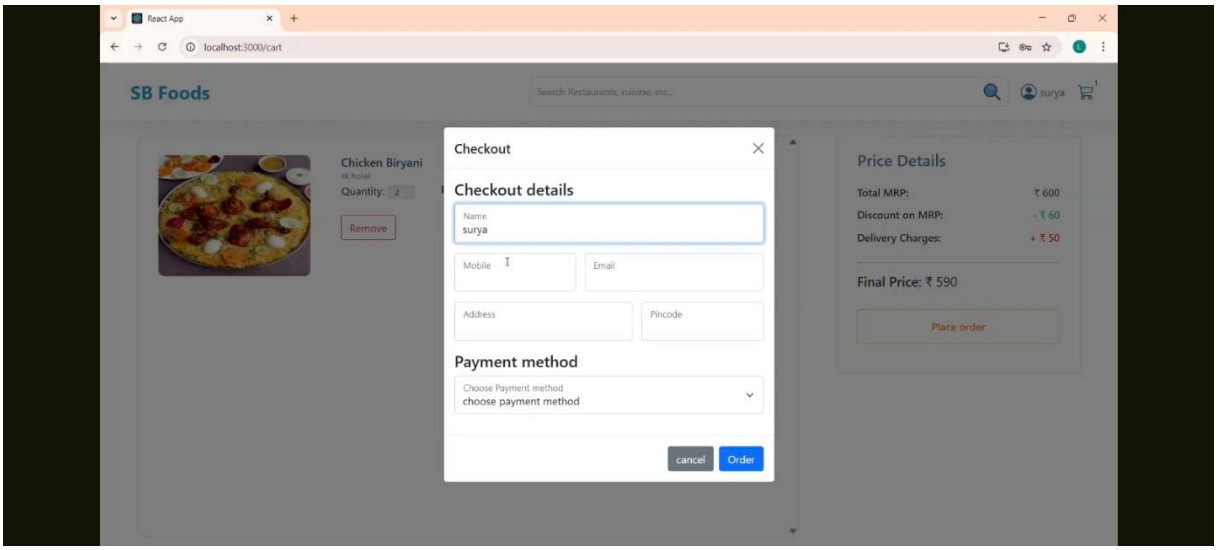
F. ADMIN APPROVAL



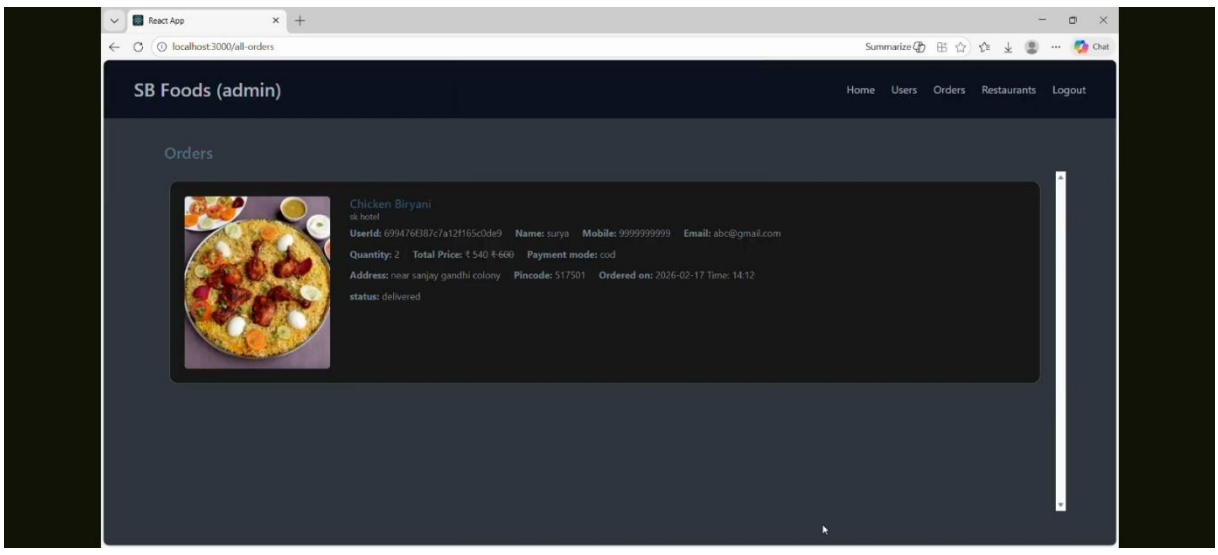
G. ORDERS IN THE CART



H. ORDERING FOOD



I. ORDER DELIVERED



12. KNOWN ISSUES

Although the application is fully functional, the following minor limitations and known issues were identified during testing:

12.1 Performance Limitations

- Application performance may slightly decrease when handling a very large volume of data.
- No pagination implemented in some data listing pages (if applicable), which may slow down loading time for large datasets.

12.2 Validation Limitations

- Frontend validation works properly, but additional backend validation improvements can be implemented for stronger security.
- Error messages could be made more user-friendly in certain edge cases.

12.3 Authentication Limitations

- JWT token expiration handling can be improved with automatic session refresh.
- Currently, there is no multi-factor authentication (MFA).

12.4 UI/UX Improvements

- UI can be enhanced further with better animations and loading indicators.
- Dark mode feature is not implemented.

12.5 Deployment-Related Issues

- Environment variables must be configured correctly; otherwise, the server may fail to start.
- CORS configuration must match the frontend URL during deployment.

13. FUTURE ENHANCEMENT

To improve the application further and enhance user experience, the following features and improvements can be implemented in future versions:

13.1 Performance Improvements

- Implement pagination and lazy loading for large datasets.
- Add caching mechanisms to reduce API response time.
- Optimize database queries for faster data retrieval.

13.2 Advanced Authentication & Security

- Implement Multi-Factor Authentication (MFA).
- Add refresh token mechanism for better session management.
- Introduce role-based access control (Admin/User).
- Implement rate limiting and enhanced input sanitization.

13.3 UI/UX Enhancements

- Add dark mode support.
- Improve dashboard design with charts and analytics.
- Add loading spinners and skeleton screens for better user feedback.
- Make the application fully mobile-optimized.

13.4 Feature Enhancements

- Add search and filter functionality for records.
- Enable file upload support (if applicable).
- Add email notifications for important actions (e.g., registration confirmation).
- Implement export options (PDF/CSV download).

13.5 Deployment & Scalability

- Deploy using cloud platforms (e.g., AWS, Render, Vercel).
- Use Docker for containerization.
- Implement CI/CD pipeline for automated testing and deployment.

GITHUB REPOSITORY:

https://github.com/shalom-m-vishal/ORDERONTHEGO-APP.git

DEMO VIDEO:

https://drive.google.com/file/d/1CSk_wgEstAHhXtKUvN3XhHfs4Vn2Qs1g/view?usp=drive_link

----------THANK YOU----------