Full Stack Development with MERN Project

Documentation format

1. Introduction

- **Project Title:** OrderOnTheGo - SB Foods
- **Team Members:**

1. **Team Leader :** Reddy Nandini

   Coordinator

   Builds RESTful APIs using Node.js and Express.js, manages authentication and server logic.

2. **Team member :** Rajulapati Chandrika

   Works on the React-based UI, handles component design, page routing, and user interactions.

3. **Team member :** Rajulapati Vijay

   Designs and manages MongoDB schemas, handles CRUD operations and ensures data consistency.

4. **Team member :** Ramagani Srikanth

   Responsible for overall planning, coordination, GitHub management, and integration of frontend and backend.

2. Project Overview

Purpose: The purpose of the **OrderOnTheGo - SB Foods** project is to develop a full-stack web application that simplifies the process of browsing, selecting, and ordering food online. It aims to provide users with a seamless food ordering experience through a modern and responsive web interface.

The application is designed to:

- Enable customers to browse food items anytime
- Add items to a cart and place orders conveniently
- Eliminate the need for physical visits or calls to restaurants
- Provide a backend system to handle product management and order storage

Ultimately, the goal is to replicate the core functionality of platforms like **Swiggy**, **Zomato**, or **Uber Eats** using open-source technologies.

Features: For Users:

- **Sign Up / Log In** – Create an account and access your orders.
- **Browse Food Items** – View a list of available dishes with images, prices, and descriptions.
- **Add to Cart** – Add favorite food items to your cart.
- **Cart Storage** – Your cart items are saved even if you refresh the page.
- **Place Orders** – Enter your address and choose payment method to place an order.

- **Order Confirmation** – Get a message when your order is successfully placed.

For Admin (Future Scope):

- **Add or Update Products** – Admin can manage food items.
- **View Orders** – Admin can see orders placed by users.

3. Architecture

Frontend (React.js)

- Built using React with multiple pages (Home, Products, Cart, etc.)
- Uses React Router for navigation and Context API for managing the cart
- Axios is used for API calls to the backend
- Cart and user info are stored in localStorage

Backend (Node.js + Express.js)

- Handles API routes like register, login, get products, and place orders
- Uses Express middleware for JSON handling and CORS
- Connects to MongoDB using Mongoose

Database (MongoDB)

- Stores user, product, and order data
- Collections:
  - users: name, email, password, address
  - products: name, description, price, image
  - orders: userId, items, address, payment method

4. Setup Instructions

Prerequisites

- **Node.js & npm** – For running frontend and backend
- **MongoDB** – Local database (use Compass or terminal)
- **Git** – To clone the project
- **VS Code** – Recommended editor

Installation Steps

Clone the Project

```
git clone https://github.com/srikanthramagani/OrderGo.git
cd OrderGo
```

1. **Install & Run Backend**

```
cd server
npm install
node server.js
```

2. **Install & Run Frontend**
   Open a new terminal:

```
cd client
```

```
npm install
npm start
```

3. **Start MongoDB**
   - Use MongoDB Compass or run `mongod` in terminal.

Your app will run at:

- Frontend: http://localhost:3000
- Backend API: http://localhost:5000

5. Folder Structure

- **Client(React frontend):**

```
client/

├── public/              → Static assets

├── src/

│   ├── components/

│   │   └── pages/        → All page components (Home, Cart, Login, etc.)

│   ├── context/          → Cart context (global state)

│   ├── App.jsx           → Main component with routes

│   └── index.js          → Entry point of the app
```

- **Server(Node.js backend):**

```
server/
├── models/              → Mongoose schemas (User, Product, Order)
├── server.js            → Main Express server file
```

6. Running the Application

Frontend :

```
cd client
npm start
```

Runs the React app at: http://localhost:3000

Backend :

```
cd server
npm start   # Or use: node server.js
```

Runs the Node.js server at: http://localhost:5000

7. API Documentation

- **POST /api/register :** Registers a new user.
- **POST /api/login** : Logs in an existing user.
- **GET /api/products** : Retrieves a list of available food products**.**
- **POST /api/orders** : Places a new order.

8. Authentication

How Authentication Works:

- Users register by providing their name, email, password, and address using the endpoint:

  POST /api/register

- They log in with their email and password using:

  POST /api/login

Method Used:

- The current setup uses **basic email and password matching**.
- There is **no token-based authentication** or sessions implemented at this stage.
- After login, the user's details can be stored on the frontend (e.g., in localStorage) to maintain the login state.

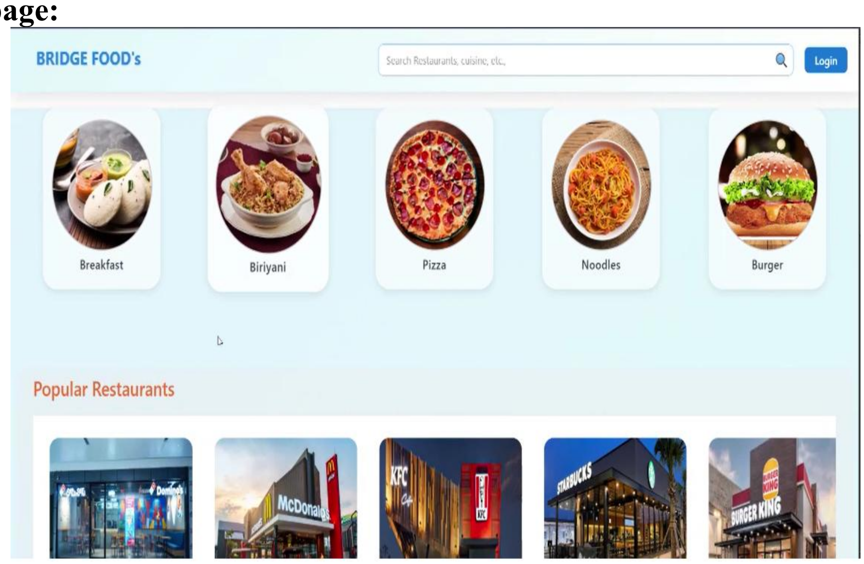
Recommendations for Improvement:

To enhance security in the future, it is recommended to:

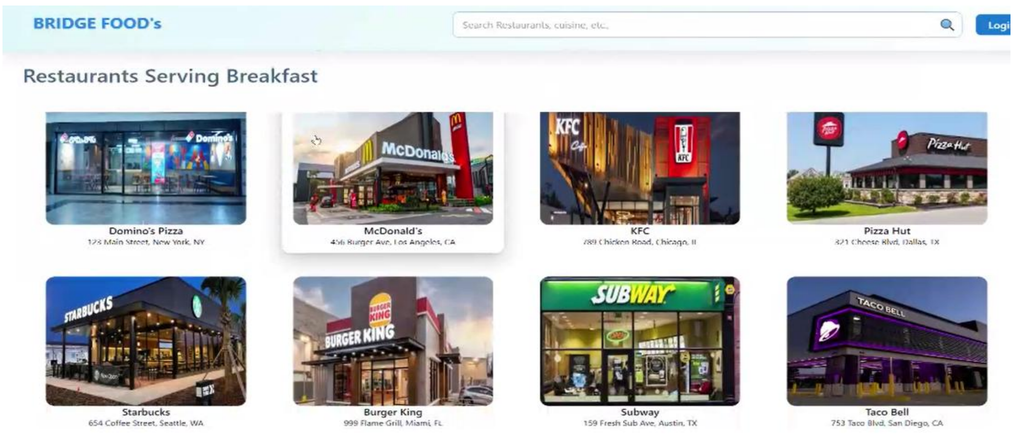
- Implement **JWT (JSON Web Token)** authentication.
- Use **middleware** to protect private API routes.
- Store tokens securely (e.g., in localStorage or HTTP-only cookies).

9. User Interface

Home page:



All breakfast serving restaurants:



Restaurants Serving Breakfast

Domino's Pizza
123 Main Street, New York, NY

McDonald's
456 Burger Ave, Los Angeles, CA

KFC
789 Chicken Road, Chicago, IL

Pizza Hut
321 Cheese Blvd, Dallas, TX

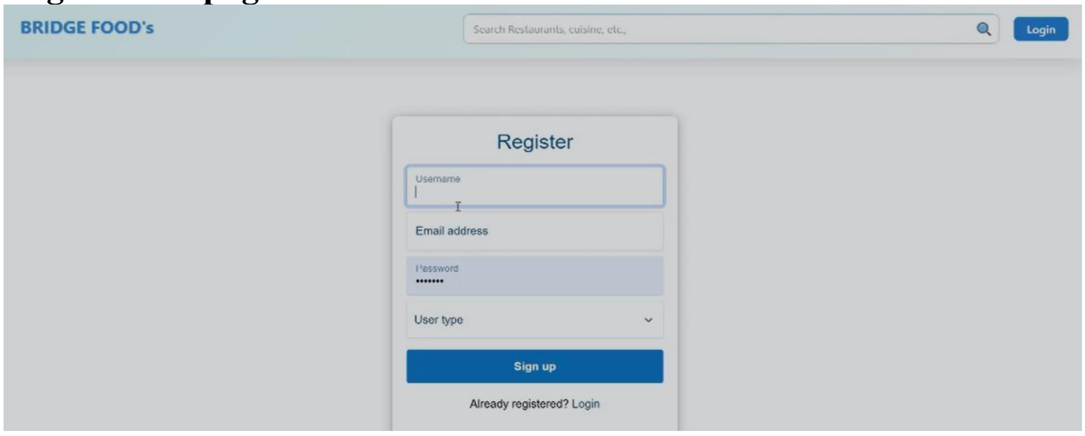
Starbucks
654 Coffee Street, Seattle, WA

Burger King
999 Flame Grill, Miami, FL

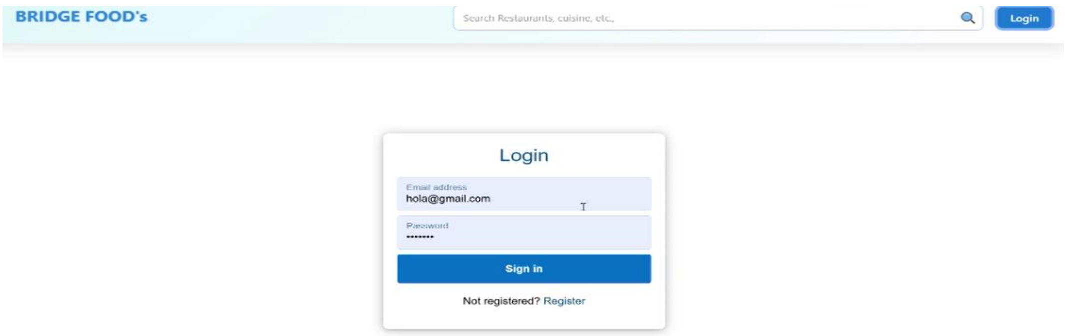
Subway
159 Fresh Sub Ave, Austin, TX

Taco Bell
753 Taco Blvd, San Diego, CA

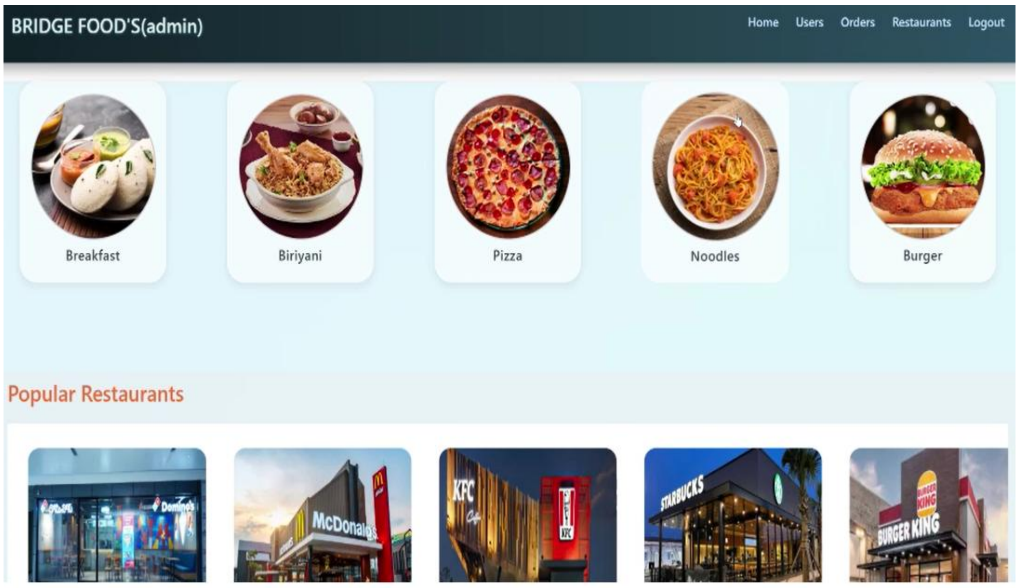
Registration page:

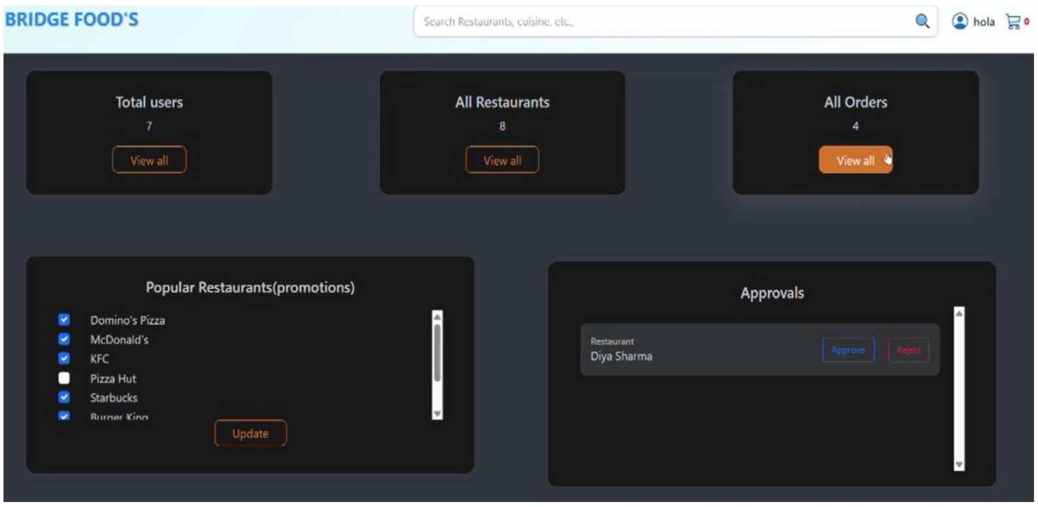


Login page:



Admin dashboard:




BRIDGE FOOD'S

Search Restaurants, cuisine, etc.,

hola 0

Total users
7
View all

All Restaurants
8
View all

All Orders
4
View all

Popular Restaurants(promotions)

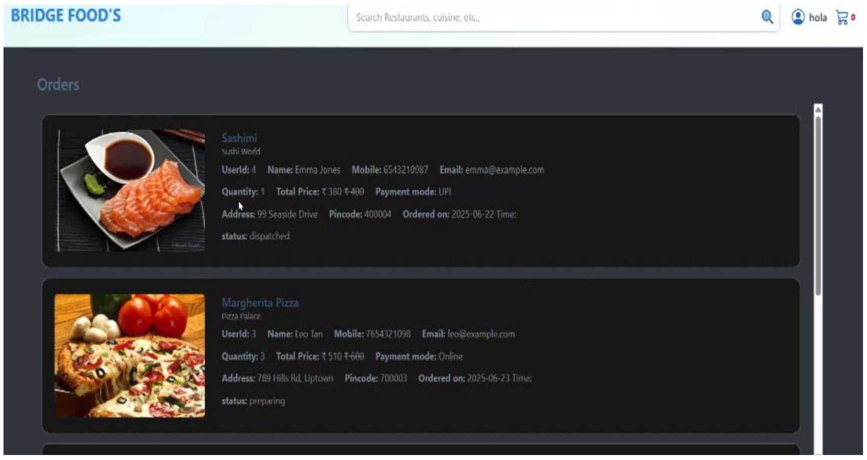
- [x] Domino's Pizza
- [x] McDonald's
- [x] KFC
- [ ] Pizza Hut
- [x] Starbucks
- [x] Burger King

Update

Approvals

Restaurant
Diya Sharma
Approve Reject

List of orders:


BRIDGE FOOD'S

Search Restaurants, cuisine, etc.,

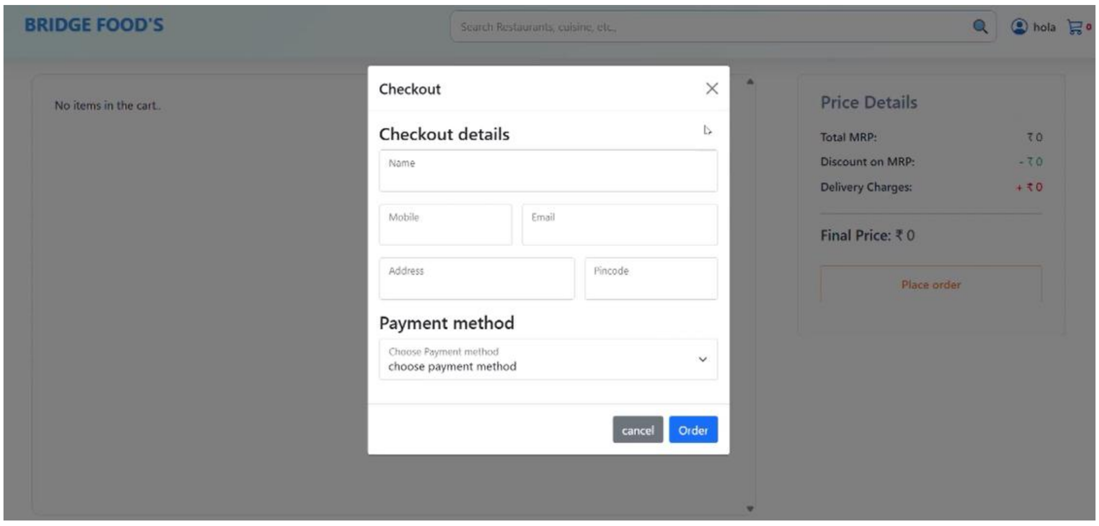
hola 0

Orders

Sashimi
Sushi World
UserId: 4 **Name:** Emma Jones **Mobile:** 6543210987 **Email:** emma@example.com
Quantity: 1 **Total Price:** ₹ 380 ~~₹ 400~~ **Payment mode:** UPI
Address: 99 Seaside Drive **Pincode:** 400004 **Ordered on:** 2025-06-22 Time:
status: dispatched

Margherita Pizza
Pizza Palace
UserId: 3 **Name:** Leo Tan **Mobile:** 7654321098 **Email:** leo@example.com
Quantity: 3 **Total Price:** ₹ 510 ~~₹ 600~~ **Payment mode:** Online
Address: 789 Hills Rd, Uptown **Pincode:** 700003 **Ordered on:** 2025-06-23 Time:
status: preparing

Checkout:


BRIDGE FOOD'S

Search Restaurants, cuisine, etc.,

hola 0

No items in the cart.

Checkout

Checkout details

Name

Mobile

Email

Address

Pincode

Payment method

Choose Payment method
choose payment method

cancel Order

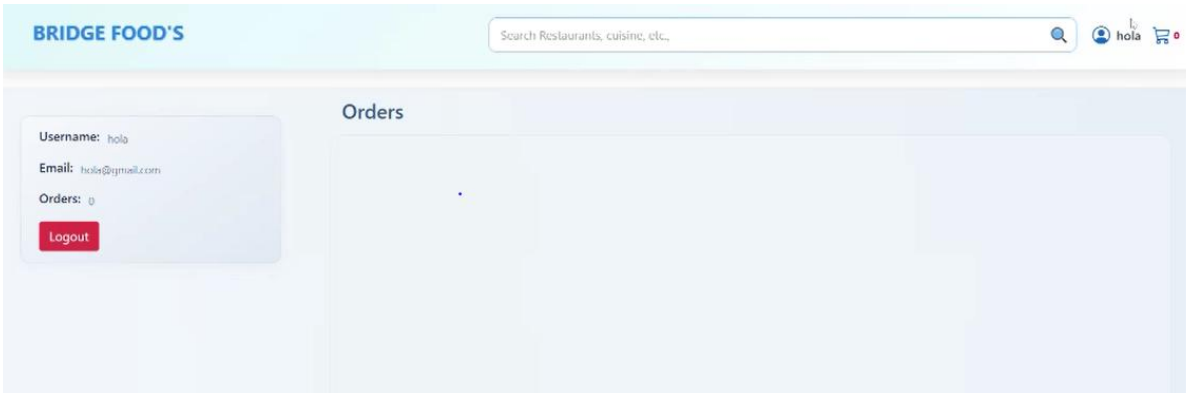
Price Details

Total MRP: ₹ 0
Discount on MRP: - ₹ 0
Delivery Charges: + ₹ 0

Final Price: ₹ 0

Place order

Orders:


BRIDGE FOOD'S

Search Restaurants, cuisine, etc.,

hola 0

Username: hola
Email: hola@gmail.com
Orders: 0

Logout

Orders

10. Testing

- **Manual testing** was done by using the app (register, login, cart, order flow).
- **Postman** was used to test backend APIs.
- **Browser DevTools** helped inspect React components and API requests.

11. Screenshots or Demo

Demo Video Check out a quick demo of OrderGo in action:Watch Demo on YouTube https://youtu.be/Pdqh0A7nmxo

12. Known Issues

- **No authentication tokens** – Login does not use JWT or sessions, so user sessions are not fully secure.
- **No order history** – Users cannot view past orders after placing them.
- **Cart resets on logout** – Cart is stored in localStorage and clears when browser data is cleared or user logs out.
- **No automated testing** – All testing is manual; no test scripts are in place.
- **No real-time updates** – Admin actions like order status changes aren't reflected instantly on user side.

13. Future Enhancements

- Use **Jest** for frontend tests.
- Use **Supertest** for backend API testing.
- Payment integration with Razorpay/Stripe
- Role-based admin access