

Full Stack Development with MERN

Project Documentation

1. Introduction

- **Project Title:** SB Foods.- Food Delivery Web App

- **Team Members:**

Utkarsha Aherrao: Frontend Developer

Anirban Mukherjee: Frontend Developer

Deeksha Kushwaha: Backend Developer

Rujula Malhotra: Backend Developer

2. Project Overview

- **Purpose:**

Our project focuses on building a simple food delivery web app using the MERN stack by combining basic but essential features. The main goal is to allow users to register, log in, view a list of available restaurants, browse food items by category, add items to their cart, and place orders smoothly.

- **Features:**

User registration and authentication

Responsive UI

Password encryption for user data privacy

Add to cart

Place order

Pay from the webapp

3. Architecture

- **Frontend:**

The frontend is built using React, a popular JavaScript library for building user interfaces. It uses useState for state management, react-router-dom for navigation, and Axios for making HTTP requests.

- **Backend:**

The backend is developed using Node.js and Express.js. It provides RESTful APIs for the frontend to interact with. The backend handles user authentication, food dishes and order retrieval.

- Database:

MongoDB is used as the database for this application. The database schema includes collections for users, orders and foods. Mongoose is used for object data modelling (ODM).

```
# const foodSchema = new mongoose.Schema({  
  name: {type:String,required:true},  
  description: {type:String,required:true},  
  price:{type:Number,required:true},  
  image:{type:String,required:true},  
  category:{type:String,required:true}  
})
```

```
# const orderSchema = new mongoose.Schema({  
  userId: { type: String, required: true },  
  items: { type: Array, required: true },  
  amount: { type: Number, required: true },  
  address: { type: Object, required: true },  
  status: { type: String, default: "Food Processing" },  
  date: { type: Date, default: Date.now },  
  payment: { type: Boolean, default: false }  
});
```

```
# const userSchema = new mongoose.Schema({  
  name:{type:String, required:true},  
  email:{type:String, required:true, unique:true},  
  password:{type:String, required:true},  
  cartData:{type:Object, default:{}},
```

}}

4. Setup Instructions

- Prerequisites:

Node.js (v14 or above),

Vite (v^6.2.0),

Axios,

React-router,

React-router-dom,

Bcrypt,

Body-parser,

Cors,

Dotenv,

Express (v^5.1.0)

Jsonwebtoken,

Mongoose,

Multer,

Nodemon,

Validator

- Installation:

Clone the repository: git clone <https://github.com/<username>/food-delivery-website>.git

Install frontend dependencies: cd frontend && npm install

Install backend dependencies: cd backend && npm install

Set up environment variables: Create a .env file in the server directory and add the required variables like MONGO_URI and JWT token.

5. Folder Structure

- Client:

The React frontend is organized as follows:

frontend/

```
├── nodemodules/
├── public/
├── src/
│   ├── assets/
│   ├── components/
│   ├── context/
│   ├── pages/
│   ├── App.jsx
│   ├── main.jsx
│   └── index.css
```

- Server:

The Node.js backend is organized as follows:

backend/

```
├── config/
├── controllers/
├── middleware/
├── models/
├── nodemodules/
└── server.js
```

6. Running the Application

- Commands to start the frontend and backend servers locally.
 - o **Frontend:**npm run dev in the client directory.
 - o **Backend:**npm run server in the server directory.

7. API Documentation

Cart Management

- **POST /api/cart/add** – Adds an item to the user's cart. Requires authentication.
- **POST /api/cart/remove** – Removes an item from the user's cart. Requires authentication.
- **POST /api/cart/get** – Retrieves the current user's cart. Requires authentication.
- Parameters:json
{ "foodId": "652f4ac3a1234567890abcdef", "quantity": 2 }
- Response:json
{ success:true,message:"Added To Cart"}

Food Management

- **POST /api/food/add** – Adds a new food item with an image. Uses multipart/form-data for image upload.
- **GET /api/food/list** – Retrieves a list of all food items.
- **POST /api/food/remove** – Removes a food item.
- Parameters:json
{ "name": "Veg Burger",
 "price": 99,
 "category": "Fast Food",
 "image": [attached image file]}
- Response:json
{ success:true,message:"Food Added"}

Order Management

- **POST /api/order/place** – Places a new order. Requires authentication.
- **POST /api/order/userorders** – Retrieves all orders placed by the logged-in user. Requires authentication.
- Parameters:json
{
 "userId": "652f2b93a1c123456789abcd",
 "items": [
 { "foodId": "652f4ac3a1234567890abcdef", "quantity": 2 },
 { "foodId": "652f4ac3a1234567890abcde0", "quantity": 1 }
],
}

```
"amount": 397,  
"address": "123 Street Name, City"  
}
```

- Response:json
{ success:true,message:"added order"}

User Management

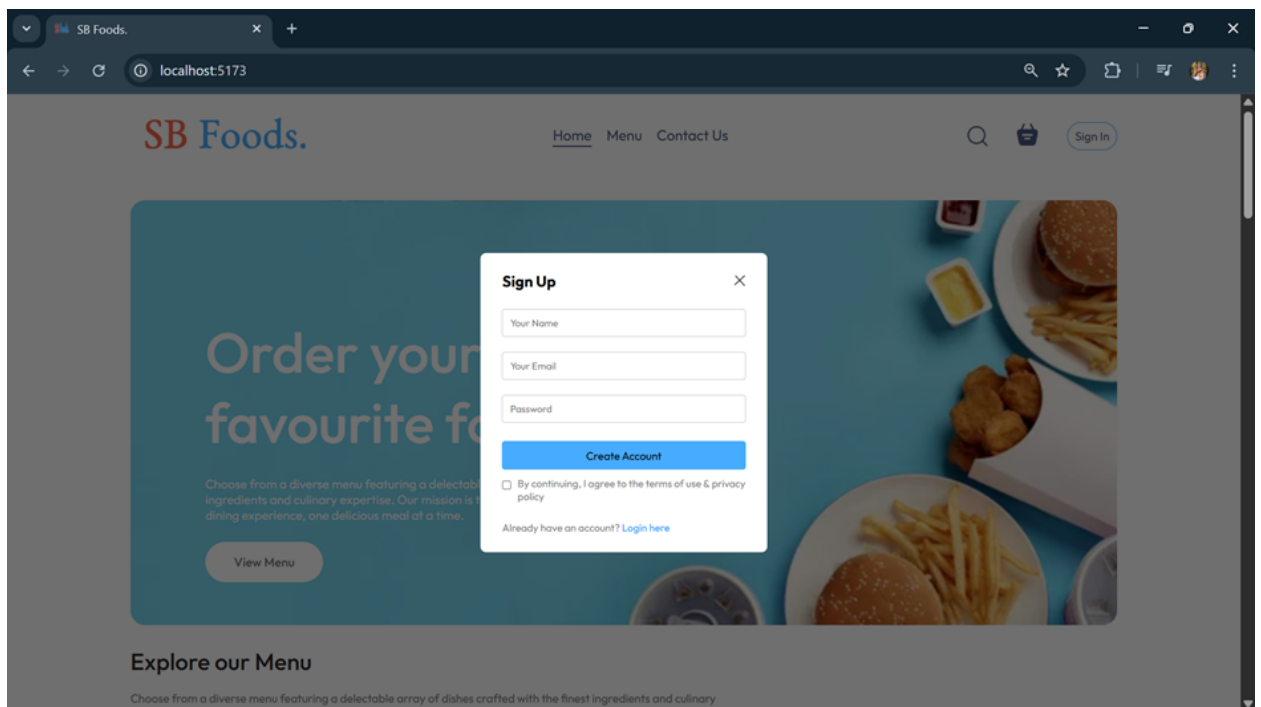
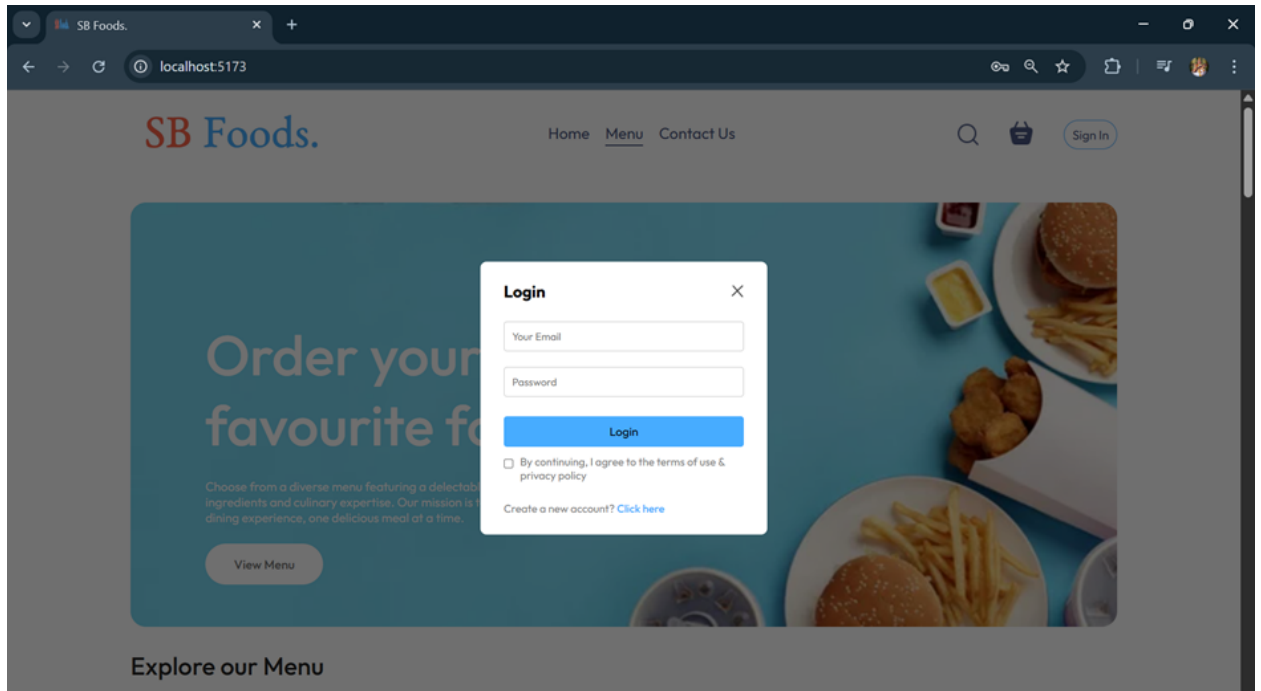
- **POST /api/user/register** – Registers a new user.
- **POST /api/user/login** – Logs in a user.
- Parameters:json
{
 "name": "John Doe",
 "email": "john@example.com",
 "password": "weakpassword"
}
- Response:json
{ success:false,message:" Please enter a strong password"}

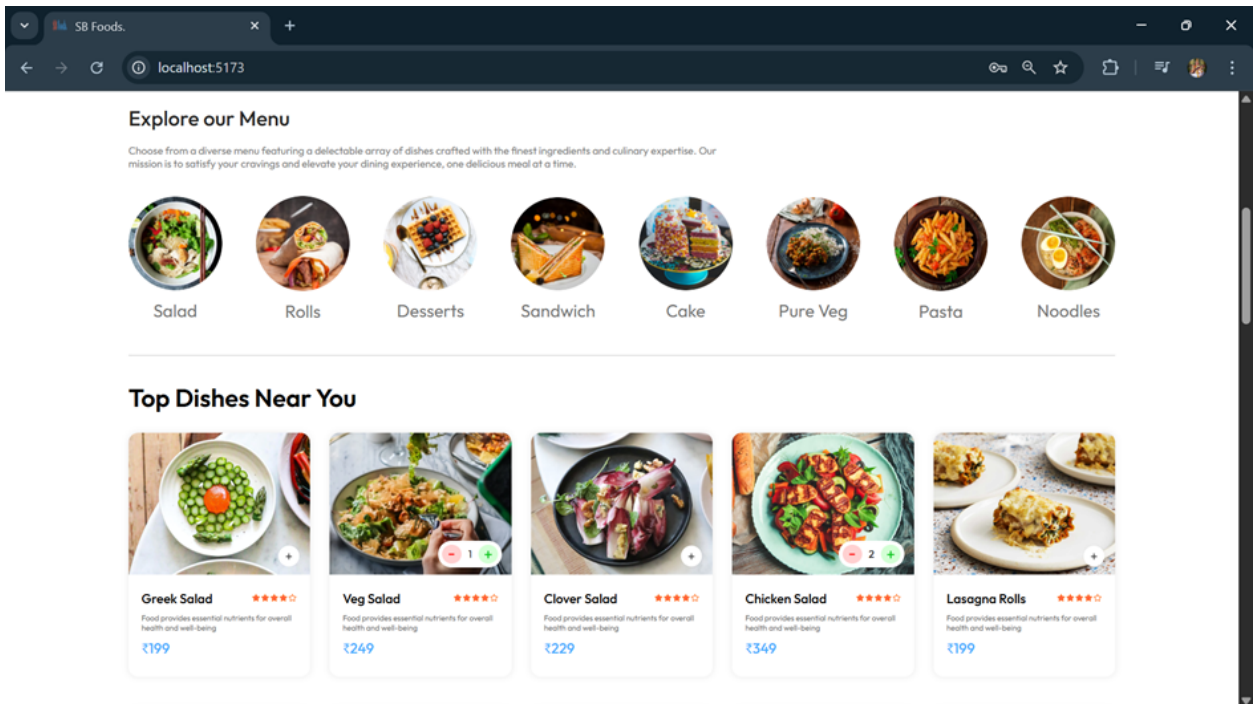
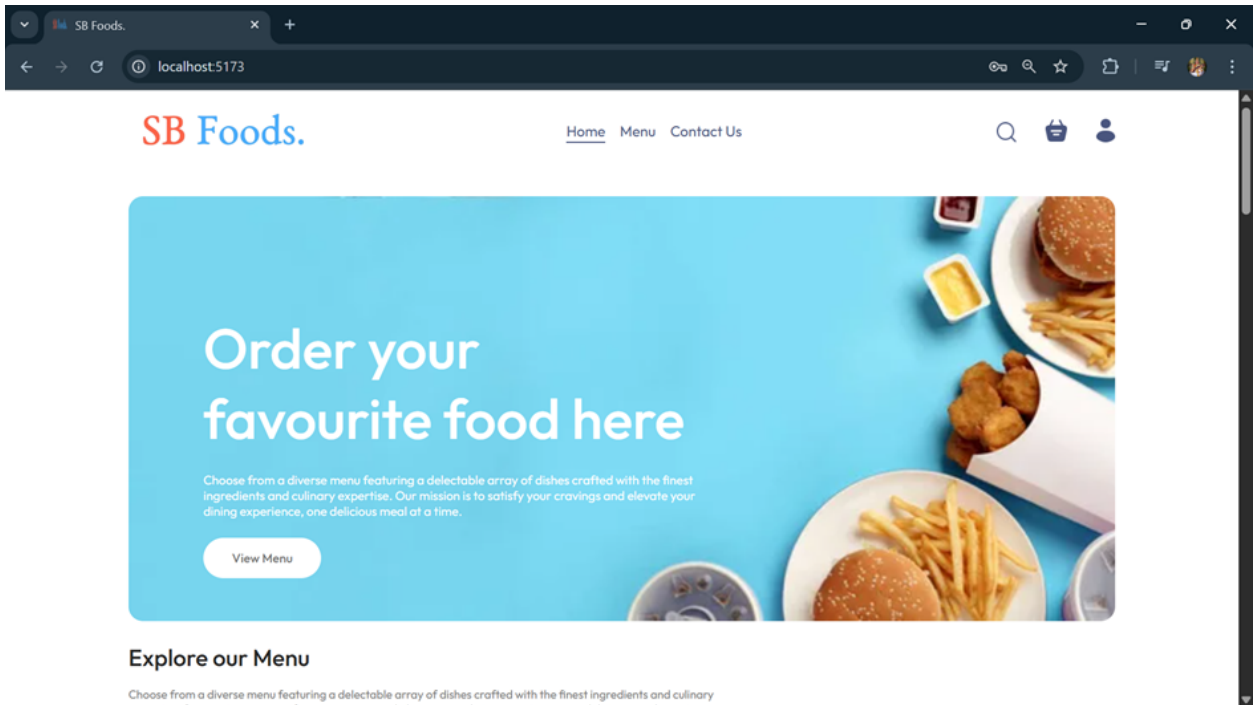
8. Authentication

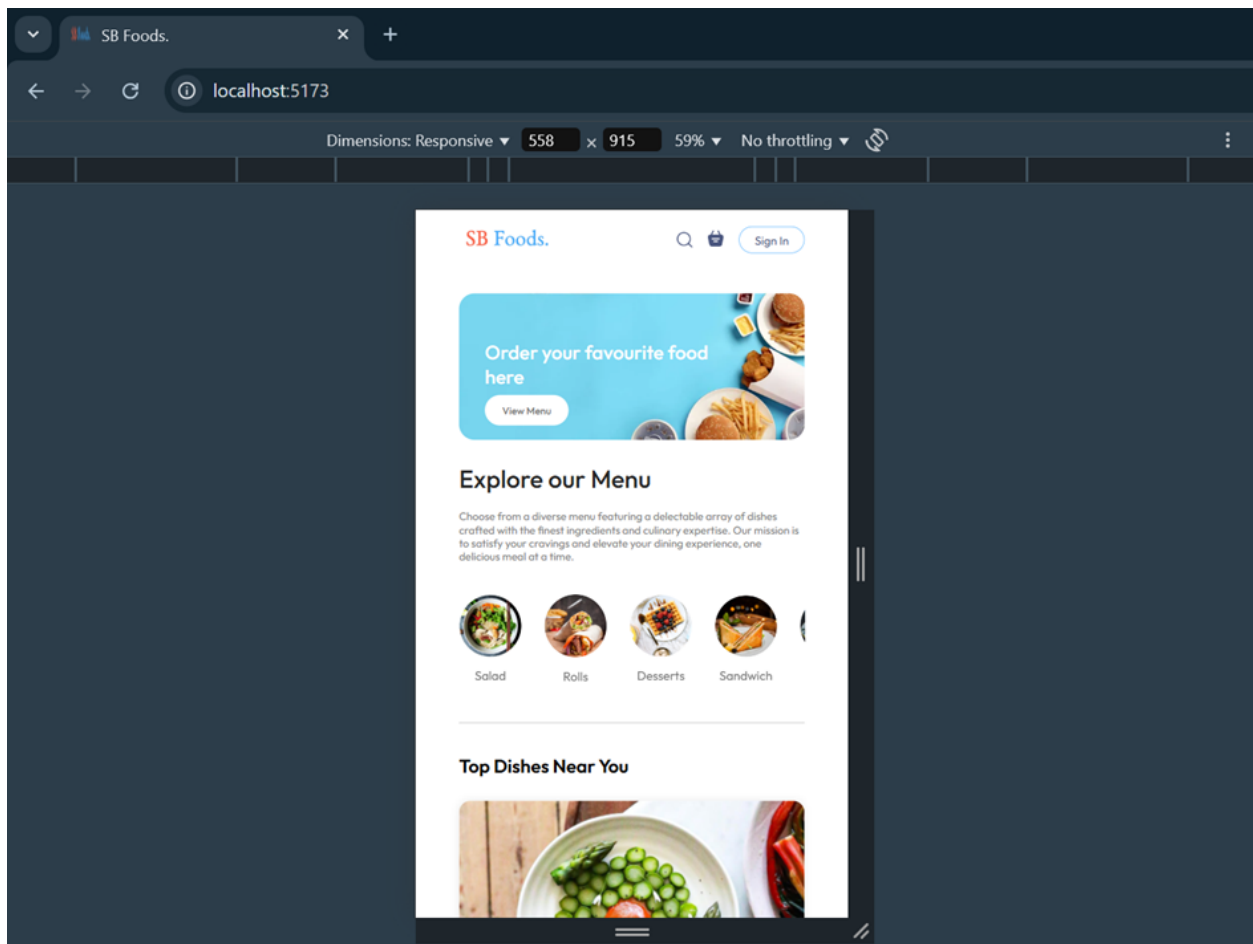
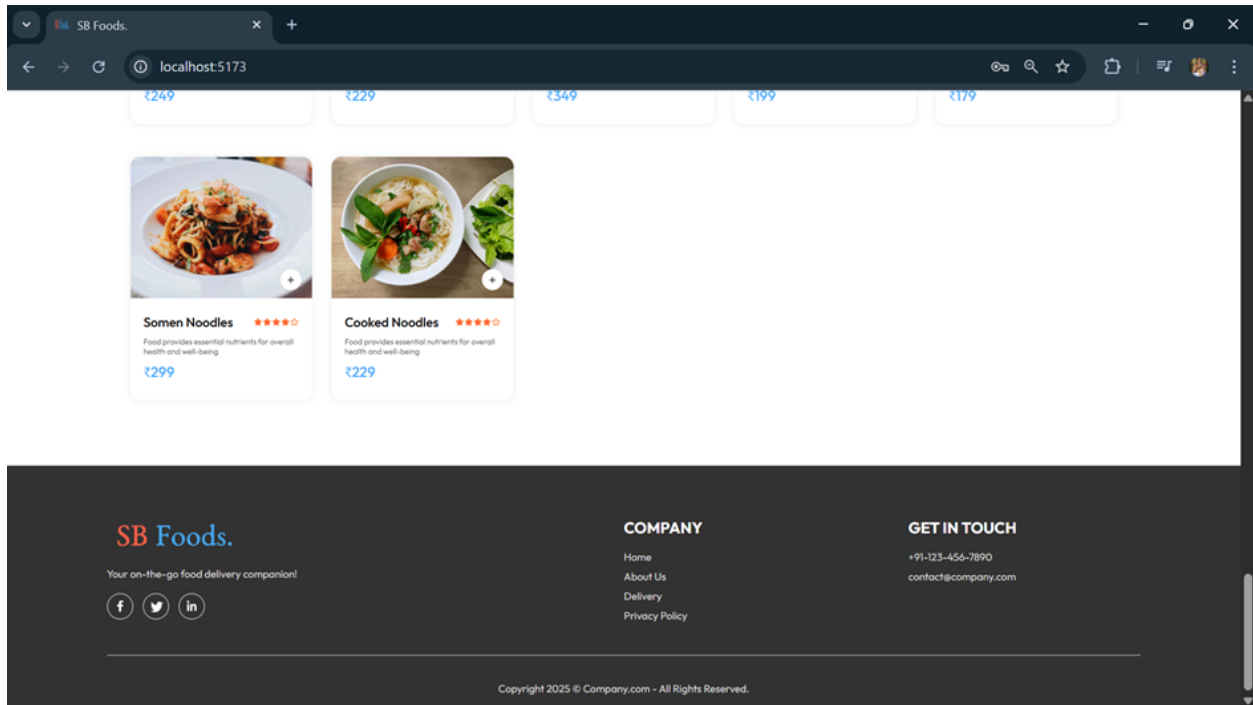
- Authentication and authorization are handled using JWT (JSON Web Tokens).
- Upon successful login, a token is generated and sent to the client. This token must be included in the Authorization header of subsequent requests.
- The server verifies the token on each request — making our setup **stateless**.

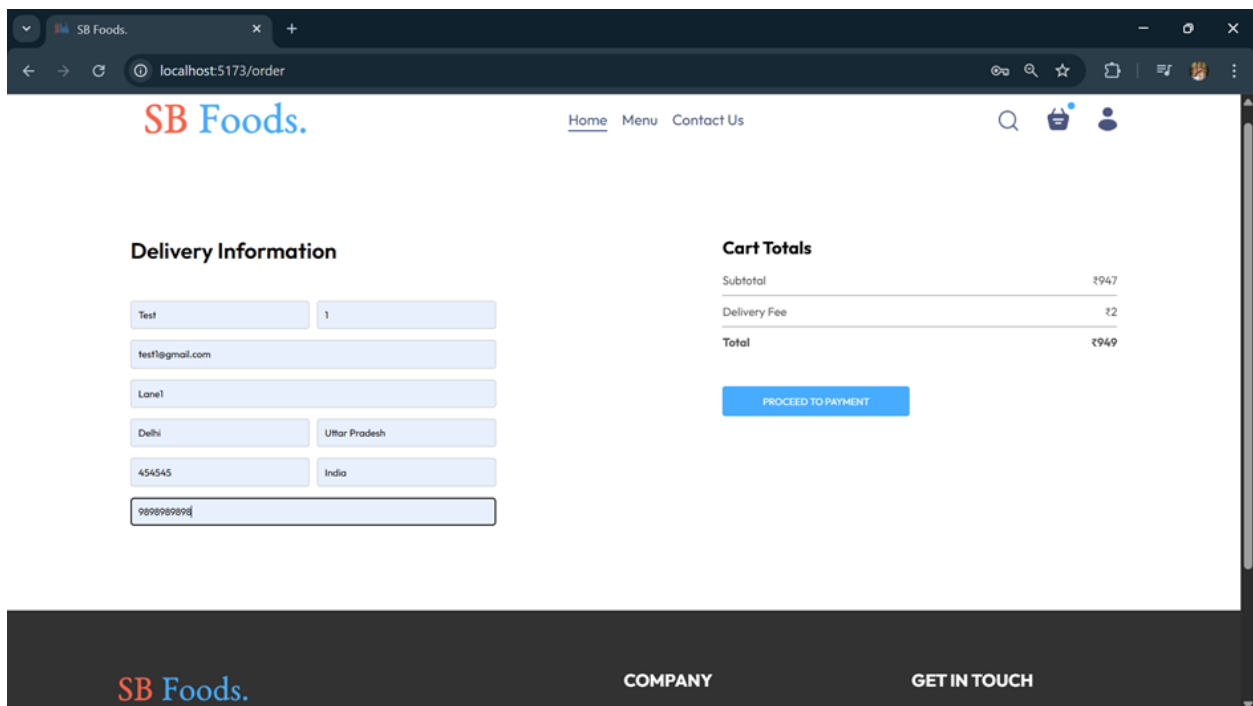
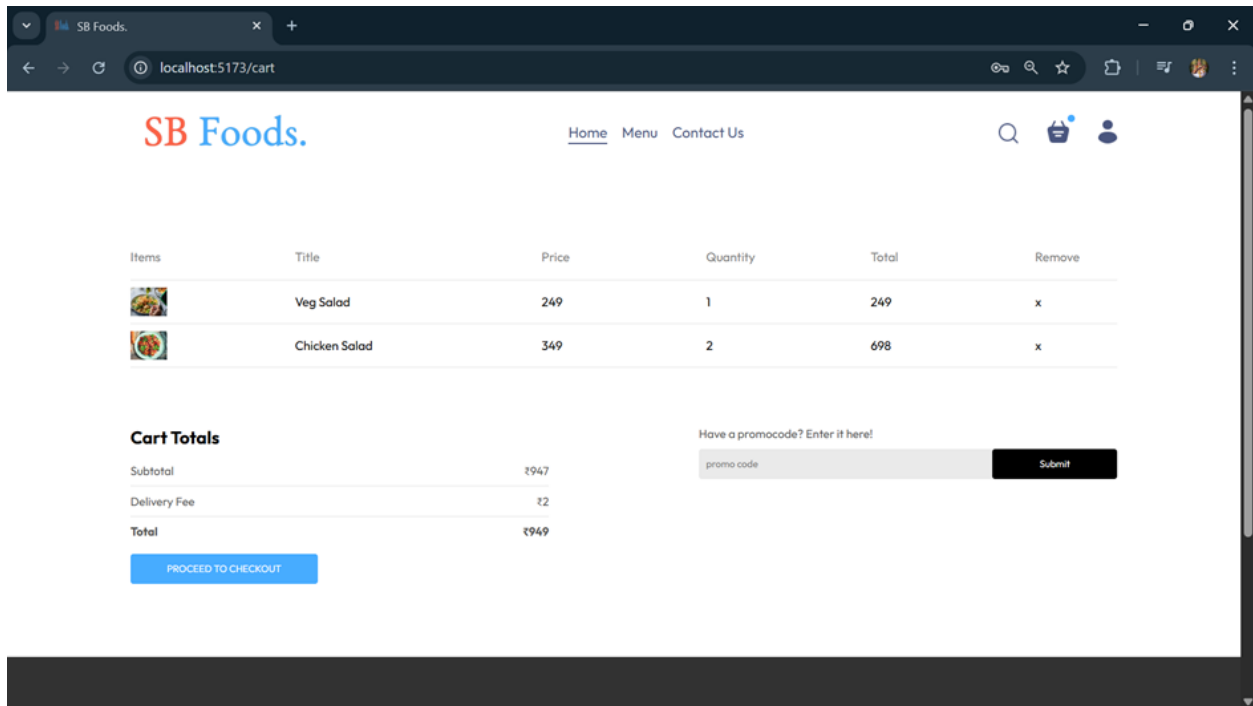
9. User Interface

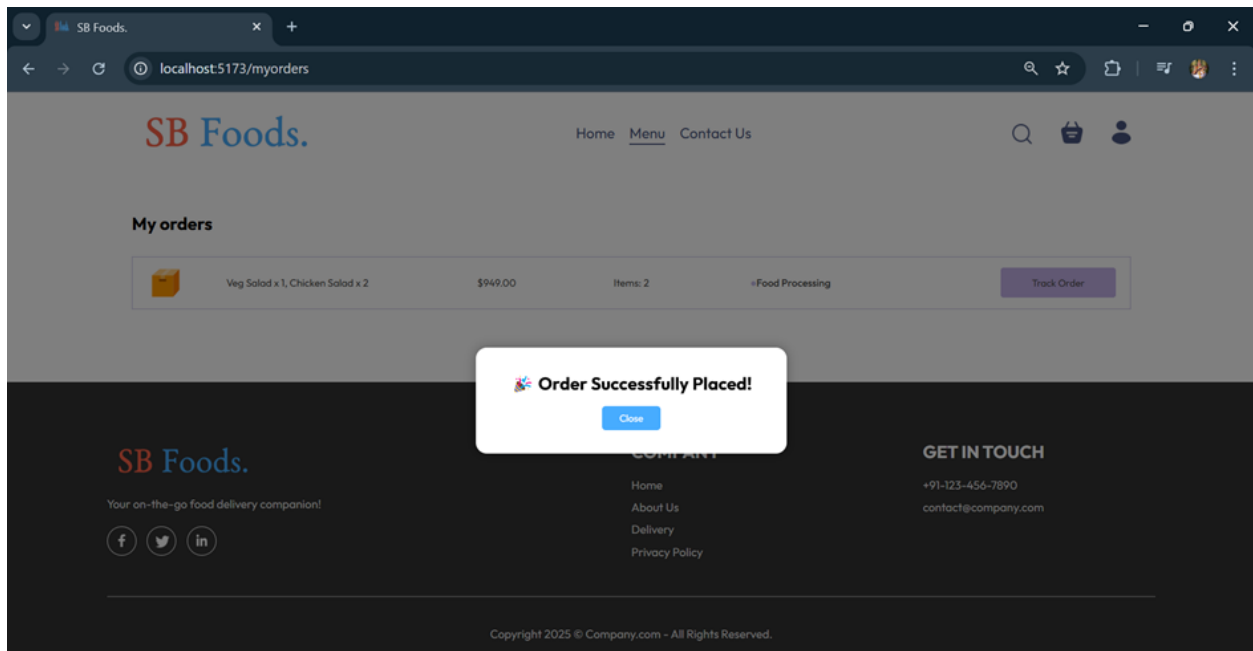
The UI is designed to be user-friendly and intuitive. Below are some screenshots showcasing different features:











10. Testing

- The testing strategy includes both unit and integration tests.
 - To ensure the reliability of a food delivery web application, implement a **multi-layered testing strategy** that includes:
- **Unit Testing** for validating individual components such as cart logic, order placement, and user authentication.
- **Integration Testing** to verify interactions between modules like food listing, cart updates, and order processing.
- **End-to-End (E2E) Testing** to simulate real user workflows – from browsing food items to checkout and order tracking.
- **Performance Testing** to ensure smooth and responsive operation during peak hours or high traffic situations.

11. Demo

- Demo video link :
https://drive.google.com/drive/folders/1mwGR6zltl07l1i7120zVY_acVyZjPzZ?usp=sharing

12. Known Issues

- The application may experience slow performance under heavy load. Further optimization is planned.
- Because of stripe being restricted and invite-only for Indian users, we were not able to integrate a payment gateway. We are looking for alternate payment gateways like razorpay.

13. Future Enhancements

- Integration with real-time food order tracking.
- Mobile app development for iOS and Android.
- Adding coupon code features.
- Integrating payment gateway.

14. APPENDIX:

Drive Link: https://drive.google.com/drive/folders/1DM8WrcWWJapNeYyYvgo4-G9PH7sNzyUQ?usp=drive_link

GitHub & Project Demo Link:

- GitHub: https://github.com/Deeks779/Food_Delivery_Website
- Project Demo Link:
https://drive.google.com/drive/folders/1mwGR6zltl07l1i7120zVY_acVyZzjPzZ?usp=sharing