NAAN MUDHALVAN FULL STACK MERN DEVELOPMENT- “SB FOOD ORDERING APP”

A PROJECT REPORT

Submitted by

BHARATH. M (310821104017)

KAARTHIKEY.G (310821104045)

KATHIRGAMAN.M (310821104047)

JESHU DENZIL DEVE.M (310821104040)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING
in

COMPUTER SCIENCE AND ENGINEERING

JEPPIAAR ENGINEERING COLLEGE, CHENNAI

ANNA UNIVERSITY: CHENNAI 600 025

NOVEMBER 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **NAAN MUDHALVAN FULL STACK MERN DEVELOPMENT SB FOOD ORDERING APP** is the Bonafide work of **BHARATH.M, KAARTHIKEY.G, KATHIRGAMAN.M, JESHU DENZIL DEVE.M** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
DR. J. Anitha Gnanaselvi **HEAD OF THE DEPARTMENT**	Ms. D. Jeevitha **SUPERVISOR**
	Assistant Professor
CSE DEPARTMENT	CSE DEPARTMENT
JEPPIAAR ENGINEERING COLLEGE, CHENNAI- 600 119	JEPPIAAR ENGINEERING COLLEGE, CHENNAI- 600 119

Table of Contents

1) Introduction:

This is the detailed report of the project of a food ordering app which is done with the help of MERN Stack for the whole application and with my team members. It took 3 to 6 days of teamwork without rest and handled so many errors while running the project. But finally, we have done the project.

2) Project Overview

The **“SB FOOD ORDERING APP”** is a food ordering app which can be used by all users for ordering your favorite food from home and getting it delivered to have a fun meal with your friends and family. This app is free for all to use and it shows varieties of food from varieties of restaurant.

Purpose of the project:

This app have so many purposes and some of these are:

- Simplifies and streamlines the food ordering and delivery process.
- Connects customers with a wide range of restaurants and their menus.
- Allows users to browse and search for food items easily.
- Facilitates seamless order placement and payment.
- Provides realtime order tracking for customers.
- Offers an admin panel for restaurant managers to handle orders and menu updates.

3) Features:

1. **User Registration and Authentication:**
   - New users can register using their email and password.
   - Secure login mechanism using JWT (JSON Web Tokens).
   - Password hashing for security.
2. **User Roles and Permissions:**
   - Admin and user roles with differentiated permissions.
   - Admins have access to manage orders, menus, and users.
   - Users can browse menus, place orders, and manage their account details.
3. **Browse and Search Menus:**
   - View menus from various restaurants.
   - Search for food items using keywords.

- Filter options available based on cuisine, price, and rating.

4. **Add to Cart and Place Orders:**
   - Users can select and add food items to their cart.
   - Easy checkout process to review orders before placing them.
   - Option to modify cart items before finalizing the order.
5. **Payment Integration:**
   - Multiple payment methods supported (e.g., credit card, PayPal, digital wallets).
   - Secure payment gateway integration for safe transactions.
   - Realtime payment status updates.
6. **Order Management and Tracking:**
   - Users can track their order status from preparation to delivery.
   - Realtime updates on order status (e.g., confirmed, in progress, out for delivery).
   - Notification system to alert users about order updates.
7. **Admin Panel:**
   - Dashboard for managing orders, users, and menus.
   - Add, update, or delete menu items.
   - View order statistics and reports.
8. **Responsive User Interface:**
   - Optimized UI for both desktop and mobile devices.
   - Seamless navigation and interactive elements.
   - Intuitive design for user convenience.
9. **User Profile Management:**
   - Users can view and edit their profile information.
   - Manage saved addresses for faster checkout.
   - View past orders and reorder favourite items easily.

4) Architecture:

The SB Food Ordering App has a simple architecture which shows a perfect flow of the application.

1. Frontend:

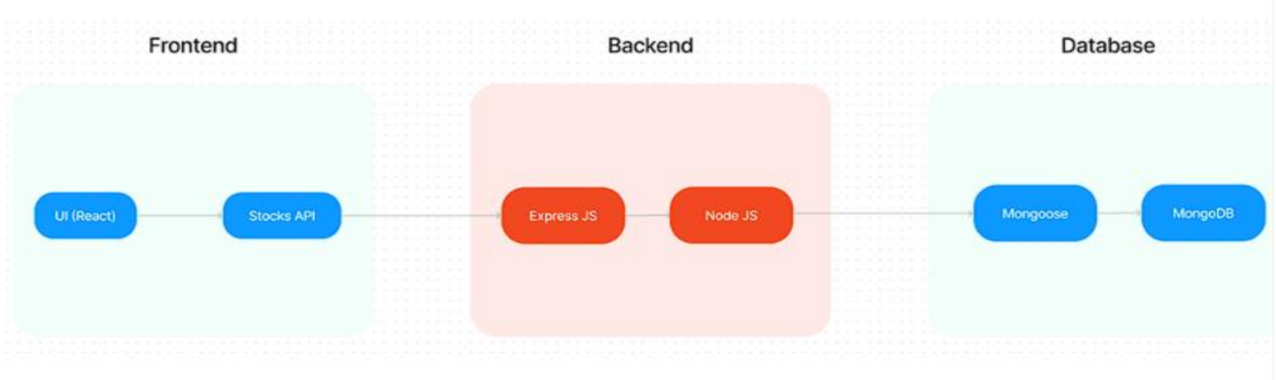
The frontend is developed using React.js, ensuring a dynamic and responsive user interface with efficient rendering of components and stock Api.

2. Backend:

The backend is built using Node.js and Express.js to provide RESTful APIs for various features such as user authentication, order processing, and payment handling.

3. Database:

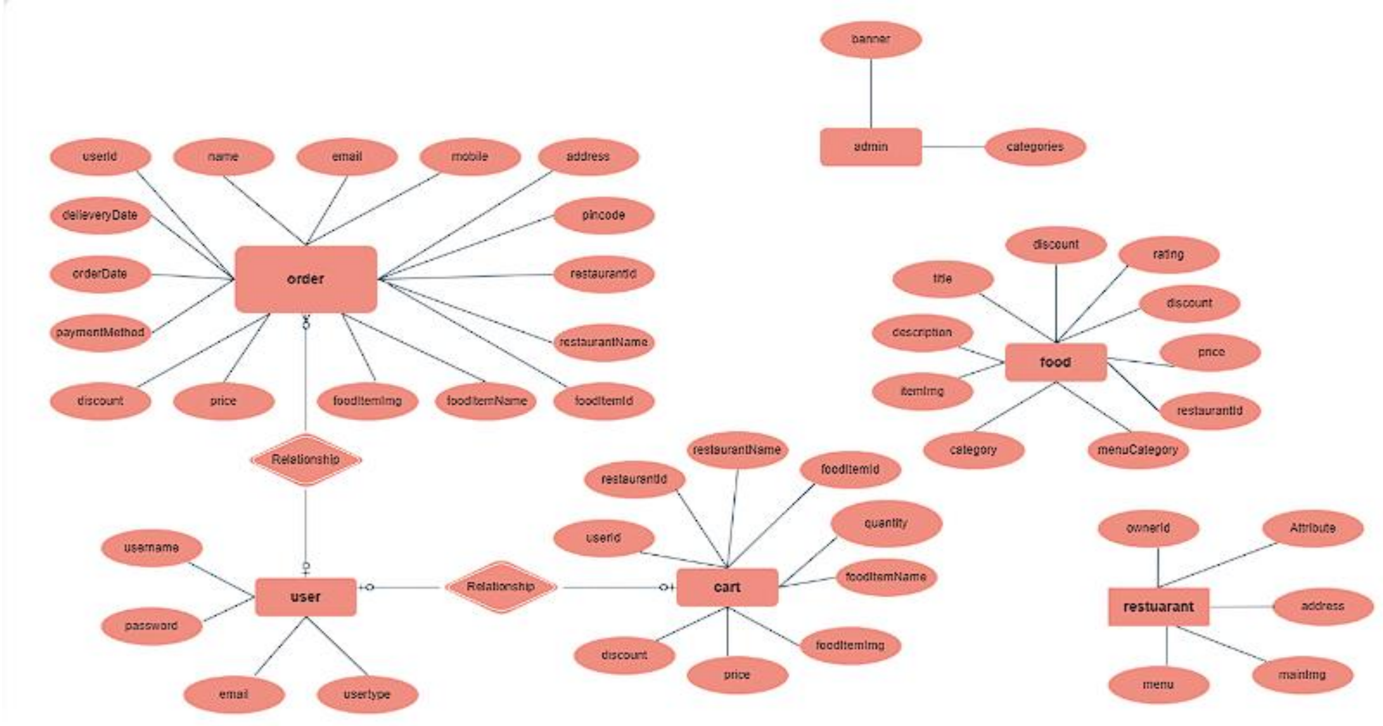
The database used for the system uses MongoDB to store and manage user data, menu items, orders, and payment information of the app.



5) ER Diagram:

The SB Foods ER Diagram represents the entities and relationships involved in a food ordering ecommerce system. It illustrates how users, restaurants, products, carts, and orders are interconnected. Here is a breakdown of the entities and their relationships:

- **User:** Represents the individuals or entities who are registered in the platform.
- **Restaurant**: This represents the collection of details of each restaurant in the platform.
- **Admin:** Represents a collection with important details such as promoted restaurants and Categories.
- **Products:** Represents a collection of all the food items available in the platform.
- **Cart:** This collection stores all the products that are added to the cart by users. Here, the elements in the cart are differentiated by the user Id.
- **Orders:** This collection stores all the orders that are made by the users in the platform.



6) Setup Instructions:

Pre-Requisite:

To develop a full stack food ordering app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

a) **Node.js and npm:** Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/packagemanager/

b) **MongoDB:** Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: https://www.mongodb.com/try/download/community
- Installation instructions: https://docs.mongodb.com/manual/installation/

c) **Express.js:** Express.js is a web application framework for Node.js. Install Express.js to handle server side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command:

"npm install express"

d) **React.js**: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide: "https://reactjs.org/docs/createanewreactapp.html"

e) **HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client side interactivity is essential.

f) Database Connectivity: Use a MongoDB driver or an Object Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

g) Frontend Framework: Utilize Angular to build the user facing part of the application, including product listings, booking forms, and user interfaces for the admin dashboard.

h) Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: https://git scm.com/downloads

I. **Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from https://code.visualstudio.com/download
- Sublime Text: Download from https://www.sublimetext.com/download
- WebStorm: Download from https://www.jetbrains.com/webstorm/download

II. **To Connect the Database with Node JS go through the below provided link:**

Link: https://www.section.io/engineeringeducation/nodejs mongoosejsmongodb/

III. **To run the existing SB Foods App project downloaded from GitHub:**

Follow below steps:

1. **Clone the repository:**

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the ecommerce app.
- Execute the following command to clone the repository:

2. **Git clone: https://github.com/jeshu2003/NAANMUDHALVANFOODORDERINGAPP**
3. **Install Dependencies:**

- Navigate into the cloned repository directory:

"cd NAANMUDHALVANFOODORDERINGAPP"

- Install the required dependencies by running the following command:

"npm install"

4. **Start the Development Server:**

- To start the development server, execute the following command:

"npm run dev" or "npm run start"

- The ecommerce app will be accessible at http://localhost:3000 by default. You can change the port configuration in the .env file if needed.

5. **Access the App:**

- Open your web browser and navigate to http://localhost:3000.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

7) Folder Structure

```
SB-Foods/
│
├── Projectphotos/
│   └── All images for the project
│
├── Client/
│   ├── public/
│   │   ├── favicon.ico
│   │   ├── index.html
│   │   ├── logo192.png
│   │   ├── logo512.png
│   │   ├── manifest.json
│   │   └── robots.txt
│   │
│   ├── src/
│   │   ├── components/
│   │   │   ├── Footer.jsx
│   │   │   ├── Login.jsx
│   │   │   ├── Navbar.jsx
│   │   │   ├── PopularRestaurants.jsx
│   │   │   ├── Register.jsx
│   │   │   └── Restaurants.jsx
│   │   │
│   │   ├── Context/
│   │   │   └── GeneralContext.js
│   │   │
│   │   ├── images/
│   │   │   ├── homebanner2.png
│   │   │   └── homebanner1.png
│   │   │
│   │   ├── 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/
│   │   │   ├── Admin.css
│   │   │   ├── AllOrders.css
│   │   │   ├── AllProducts.css
│   │   │   ├── AllUsers.css
│   │   │   ├── Authentication.css
│   │   │   ├── Cart.css
│   │   │   ├── CategoryProducts.css
│   │   │   ├── CheckOutPage.css
│   │   │   ├── Footer.css
│   │   │   ├── Home.css
│   │   │   ├── IndividualRestaurant.css
│   │   │   ├── Navbar.css
│   │   │   ├── NewProducts.css
│   │   │   ├── PopularRestaurants.css
│   │   │   ├── Products.css
│   │   │   ├── Profile.css
│   │   │   ├── RestaurantHome.css
│   │   │   └── Restaurants.css
│   │   │
│   │   ├── App.css
│   │   ├── App.js
│   │   ├── App.test.js
│   │   ├── index.css
│   │   ├── index.js
│   │   ├── logo.svg
│   │   ├── reportWebVitals.js
│   │   └── setupTests.js
│
├── Server/
│   ├── packagelock.json
│   └── package.json
│
```

8) Running the Application

To set up and run the SB FOOD ORDERING APP locally, follow the steps below:

Prerequisites:

- Ensure that Node.js and npm (Node Package Manager) are installed on your system.
- Confirm that MongoDB is installed and running if you are using a local database.
- Clone the project repository to your local machine.

Step 1: Set Up the Frontend:

1. Navigate to the Client Directory:
   - Open a terminal window.
   - Change your current directory to the client folder:

     “cd SB-Foods/Client”

2. Install Frontend Dependencies:
   Run the following command to install all required npm packages for the frontend:
   “npm install”

3. Start the Frontend Development Server:
   Once dependencies are installed, start the frontend server by running:
   “npm start”

4. Verify the Frontend:

- Check that the UI loads correctly, and all components render without errors.
- Ensure there are no errors in the browser console.

Step 2: Set Up the Backend

1. Navigate to the Server Directory:

- Open a new terminal window.
- Change your current directory to the server folder:

  “cd SB-Foods/Server”

2. Install Backend Dependencies:

Run the following command to install all necessary npm packages for the backend:

“npm install”

3. Configure Environment Variables:

If your application requires environment variables, create a `.env` file in the `Server` directory with necessary configurations like MongoDB URI, JWT secret, and other API keys. Example:

```
“PORT=3000
MONGODB_URI=mongodb://localhost:3000/sbfoodordering
JWT_SECRET=your_jwt_secret_key”
```

4. Start the Backend Server:

Start the backend server by running:

“npm start”

5. Verify the Backend:

- Check that the backend server starts without errors.
- You should see a message in the terminal indicating that the server is running and connected to the database.

Step 3: Access the Application

- Frontend (React Application):

Open http://localhost:3000 in your web browser to interact with the frontend UI.

- Backend (API Server):

Ensure that the backend server is running and can be accessed via http://localhost:5000 (or another port if specified).

Step 4: Testing End-to-end Functionality

1. Register or log in as a user to verify the authentication and authorization features.
2. Browse restaurants, add items to the cart, and place an order to test the core functionalities.
3. Use the admin account (if applicable) to manage orders, users, and menu items.

Troubleshooting

Port Conflicts:

If you encounter port conflicts, you can change the frontend or backend port in the respective configurations.

Database Connection Issues:
Ensure that MongoDB is running and accessible. If you are using a cloud database, verify that the URI in the “.env” file is correct.

9) API Documentation:

Endpoints Overview:
`POST /api/users/register` User registration
`POST /api/users/login` User login
`GET /api/menu` Get menu items
`POST /api/orders` Place an order
`GET /api/orders/: userId` Retrieve user orders

Example Response:

```
json
{
  "success": true,
  "message": "Order placed successfully",
  "orderId": "12345"
}
```

10) Authentication:

- The app uses JWT (JSON Web Tokens) for user authentication and authorization.
- Tokens are generated upon login and stored in the client's local storage for subsequent API calls.

11) User Interface:

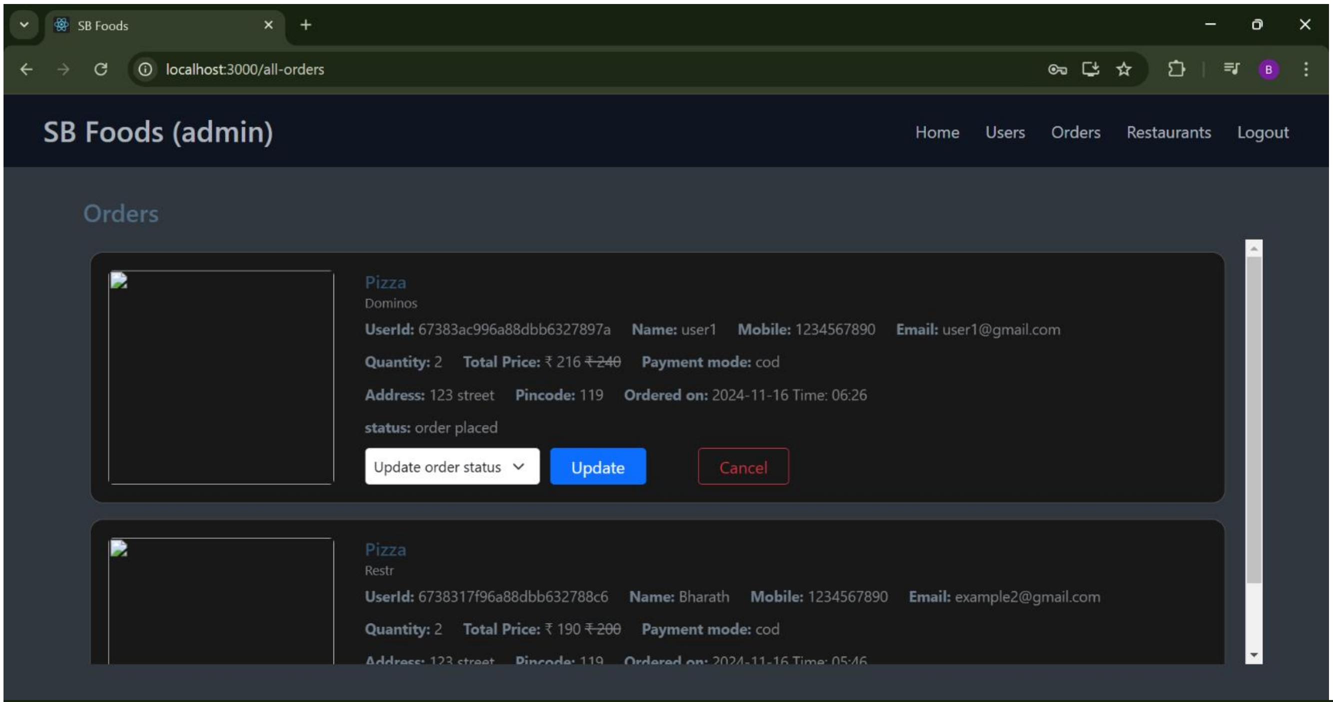
The User Interface or, UI, is the layout or window of the application. The UI of the app consists of:

- A homepage with featured restaurants
- Menu pages with filtering options
- Cart and checkout pages

Screenshots:
The screenshots of the app will give you a clear vision about the appearance and layout of the application. It includes the screenshots of:

1. Admin
2. Customer
3. Restaurants and
4. App’s UI

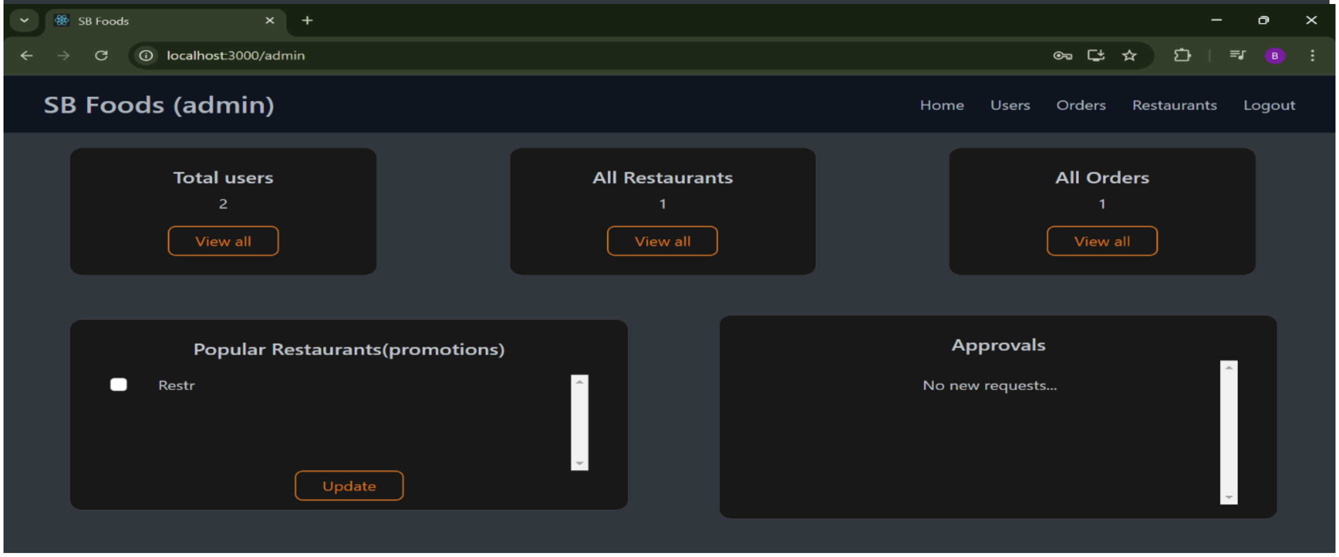
Admin:



Orders

Pizza
Dominos
UserId: 67383ac996a88dbb6327897a **Name:** user1 **Mobile:** 1234567890 **Email:** user1@gmail.com
Quantity: 2 **Total Price:** ₹ 216 ~~₹ 240~~ **Payment mode:** cod
Address: 123 street **Pincode:** 119 **Ordered on:** 2024-11-16 Time: 06:26
status: order placed

Pizza
Restr
UserId: 6738317f96a88dbb632788c6 **Name:** Bharath **Mobile:** 1234567890 **Email:** example2@gmail.com
Quantity: 2 **Total Price:** ₹ 190 ~~₹ 200~~ **Payment mode:** cod
Address: 123 street **Pincode:** 119 **Ordered on:** 2024-11-16 Time: 05:46



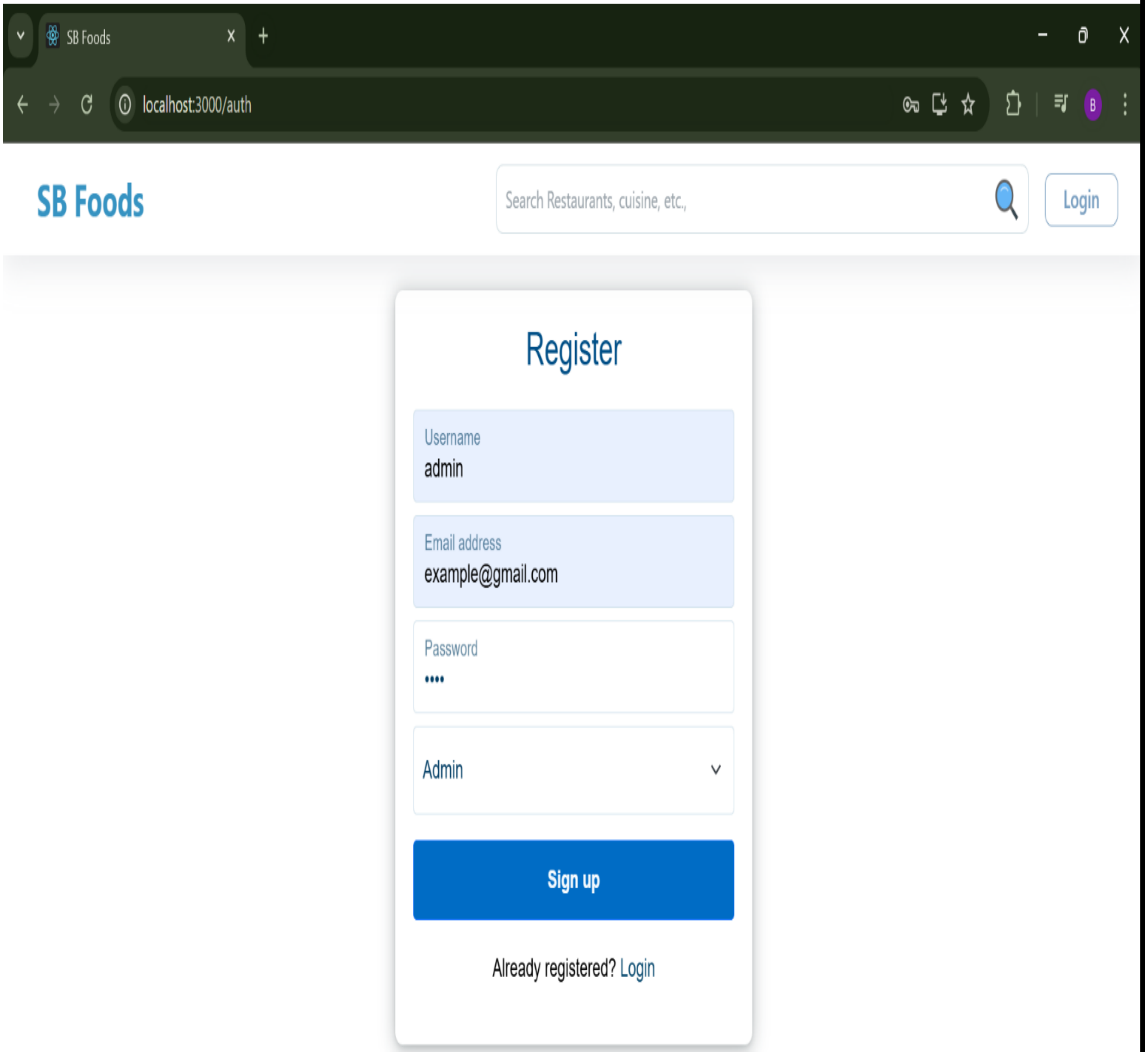
Total users
2

All Restaurants
1

All Orders
1

Popular Restaurants(promotions)
☐ Restr

Approvals
No new requests...



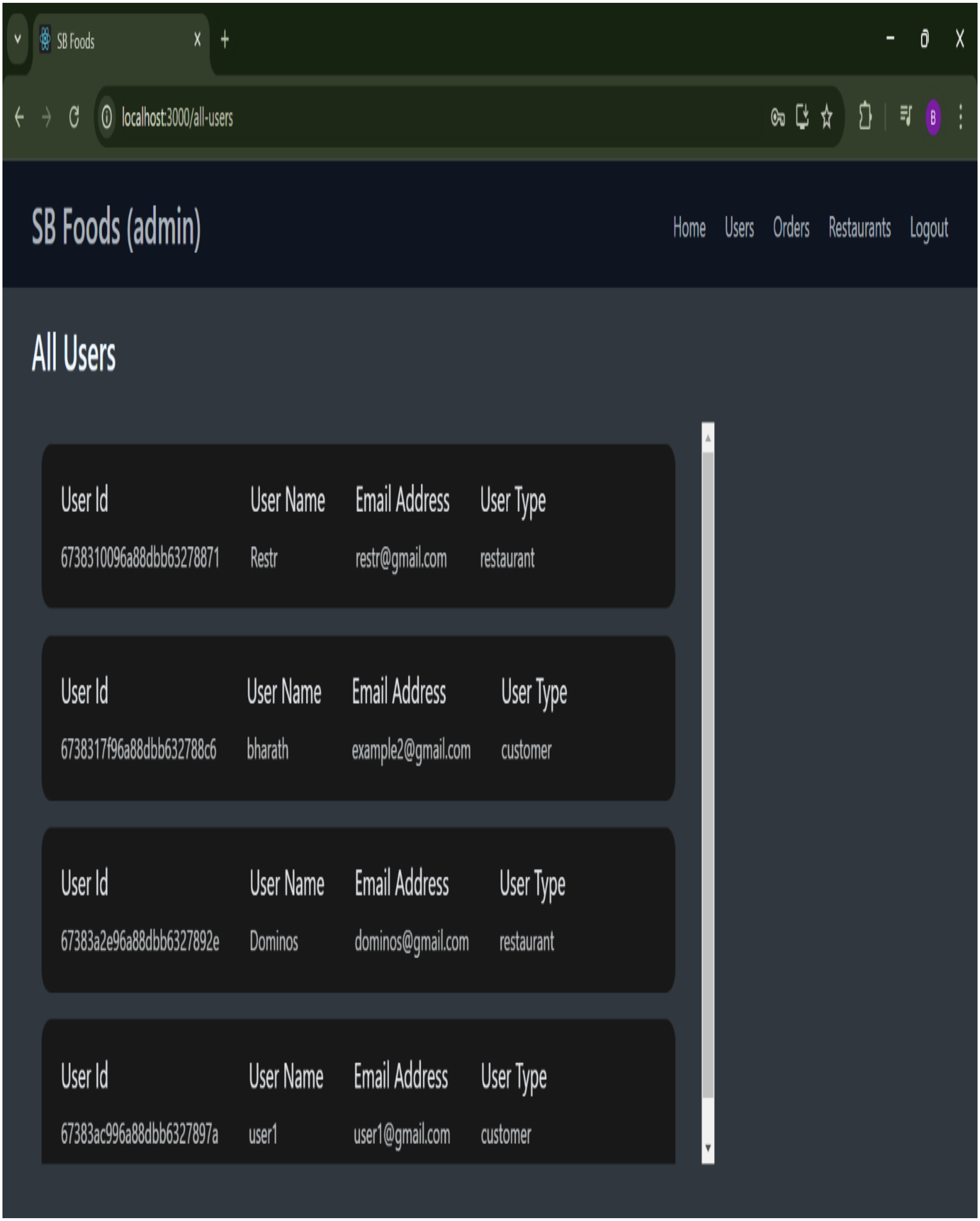
Register

Username: admin

Email address: example@gmail.com

Password: ••••

Admin



SB Foods (admin)

All Users

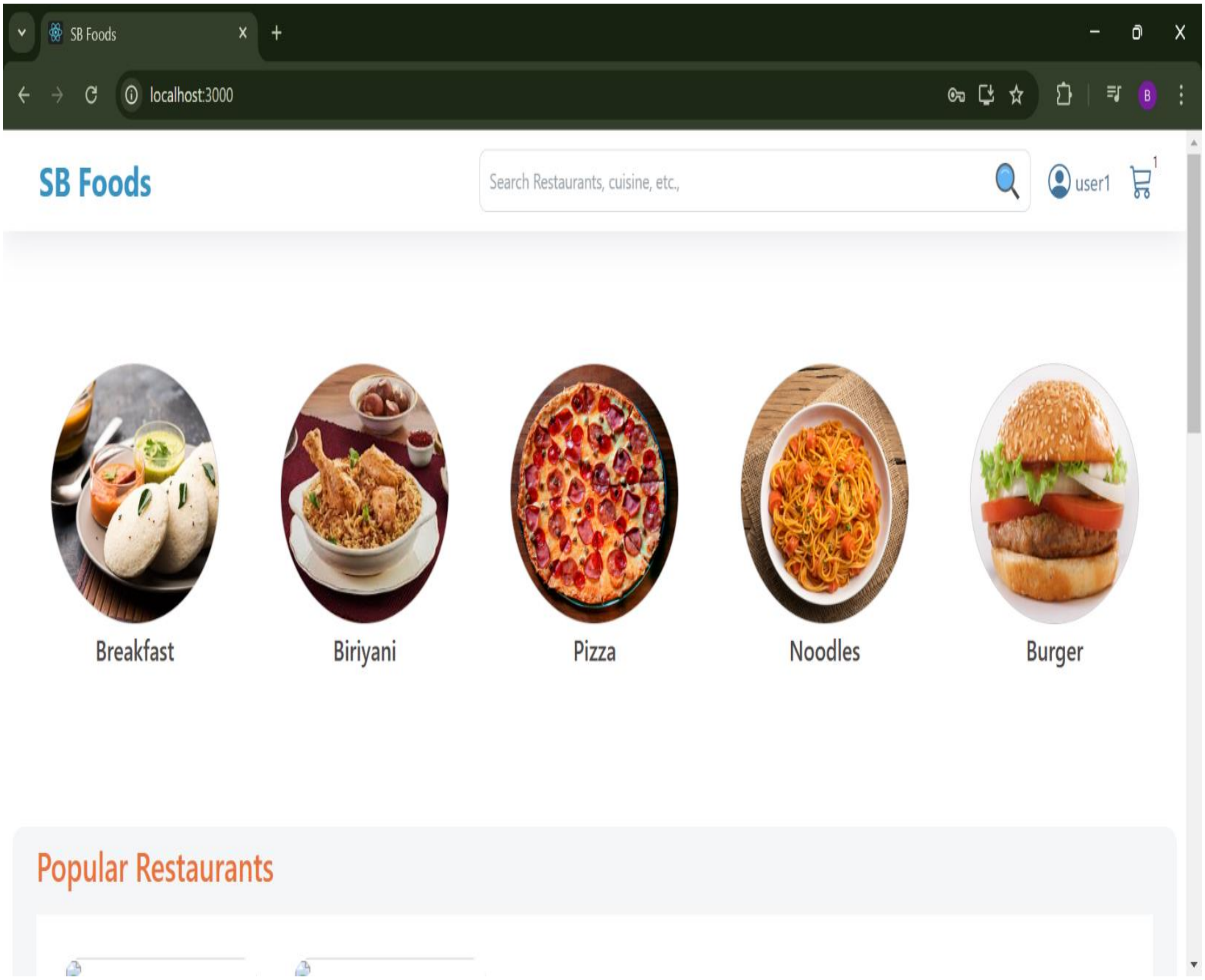
User Id	User Name	Email Address	User Type
6738310096a88dbb63278871	Restr	restr@gmail.com	restaurant

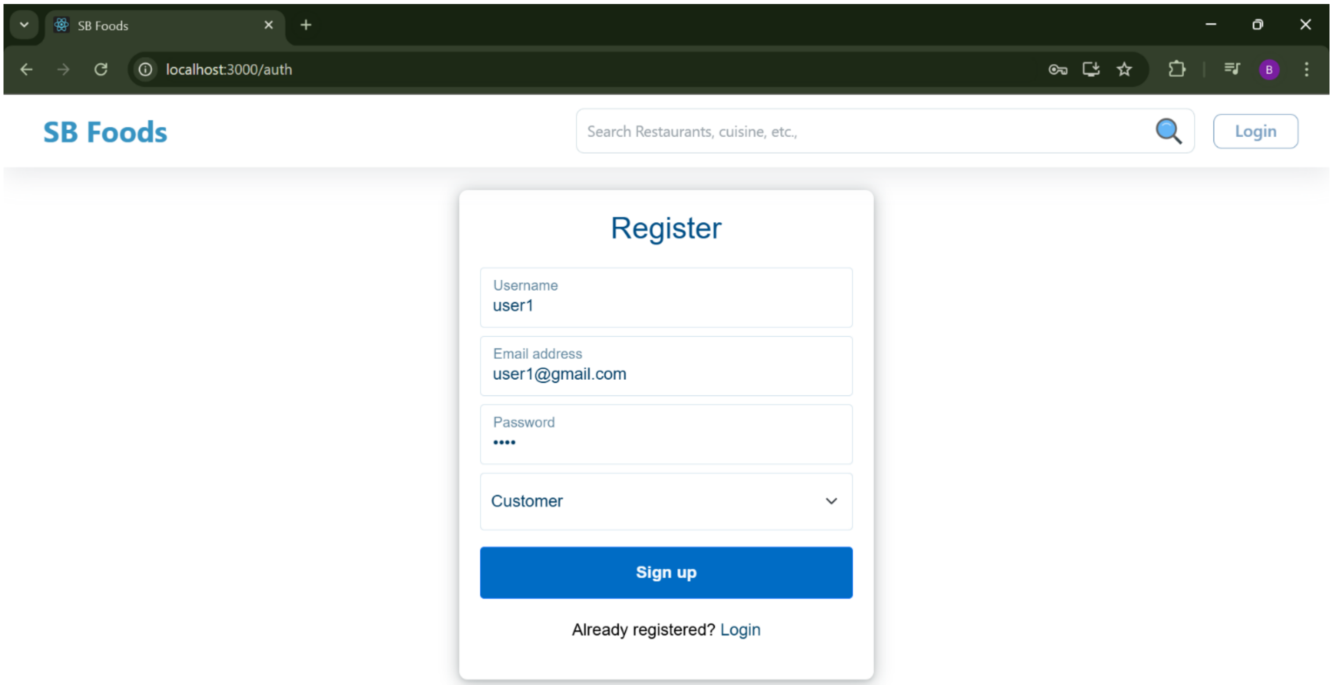
User Id	User Name	Email Address	User Type
6738317f96a88dbb632788c6	bharath	example2@gmail.com	customer

User Id	User Name	Email Address	User Type
67383a2e96a88dbb6327892e	Dominos	dominos@gmail.com	restaurant

User Id	User Name	Email Address	User Type
67383ac996a88dbb6327897a	user1	user1@gmail.com	customer

Customer:


SB Foods
localhost:3000
SB Foods
Search Restaurants, cuisine, etc.,
user1
1
Breakfast
Biriyani
Pizza
Noodles
Burger
Popular Restaurants



Register

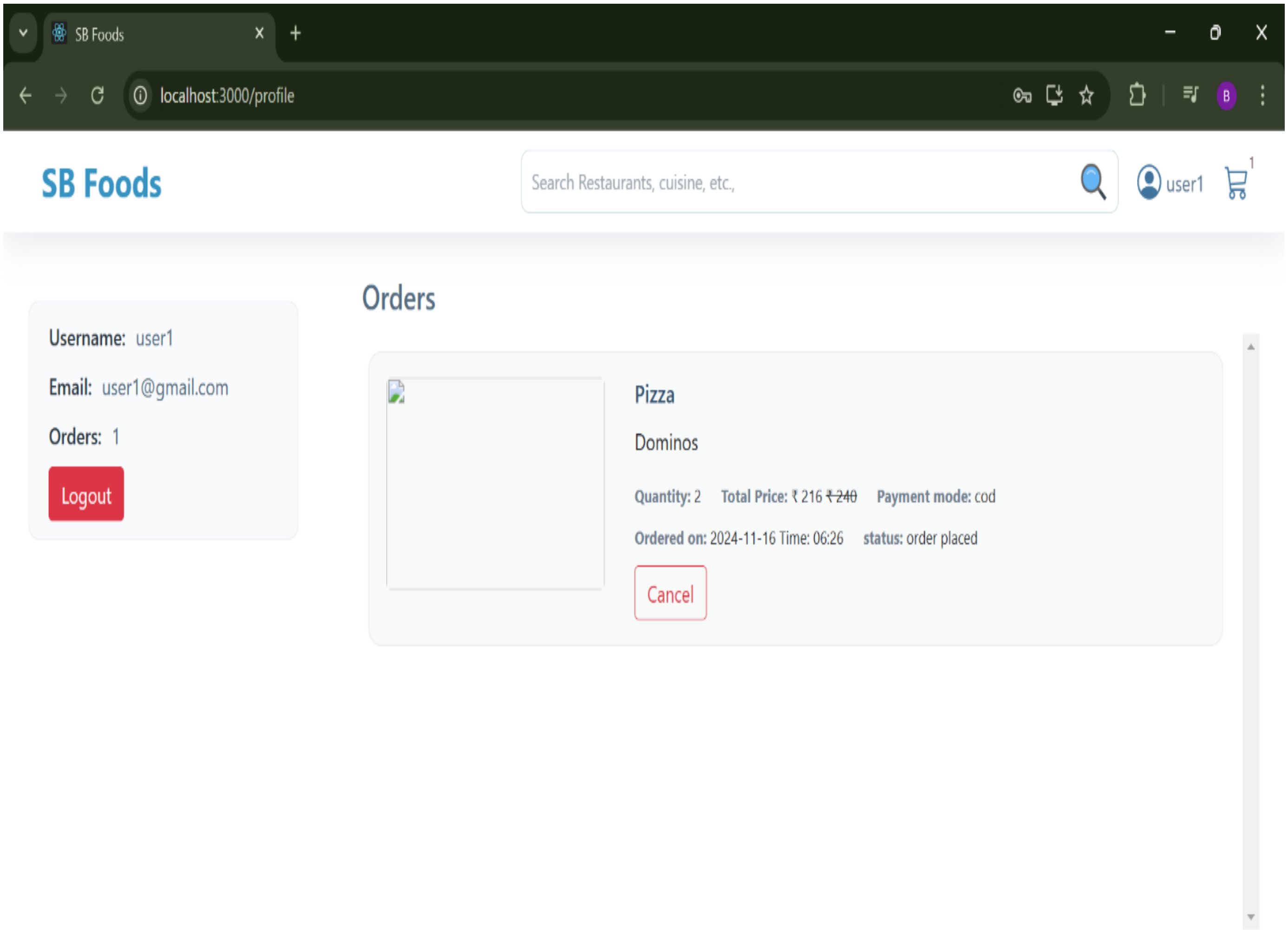
Username
user1

Email address
user1@gmail.com

Password
••••

Customer

Already registered? Login



Username: user1

Email: user1@gmail.com

Orders: 1

Orders

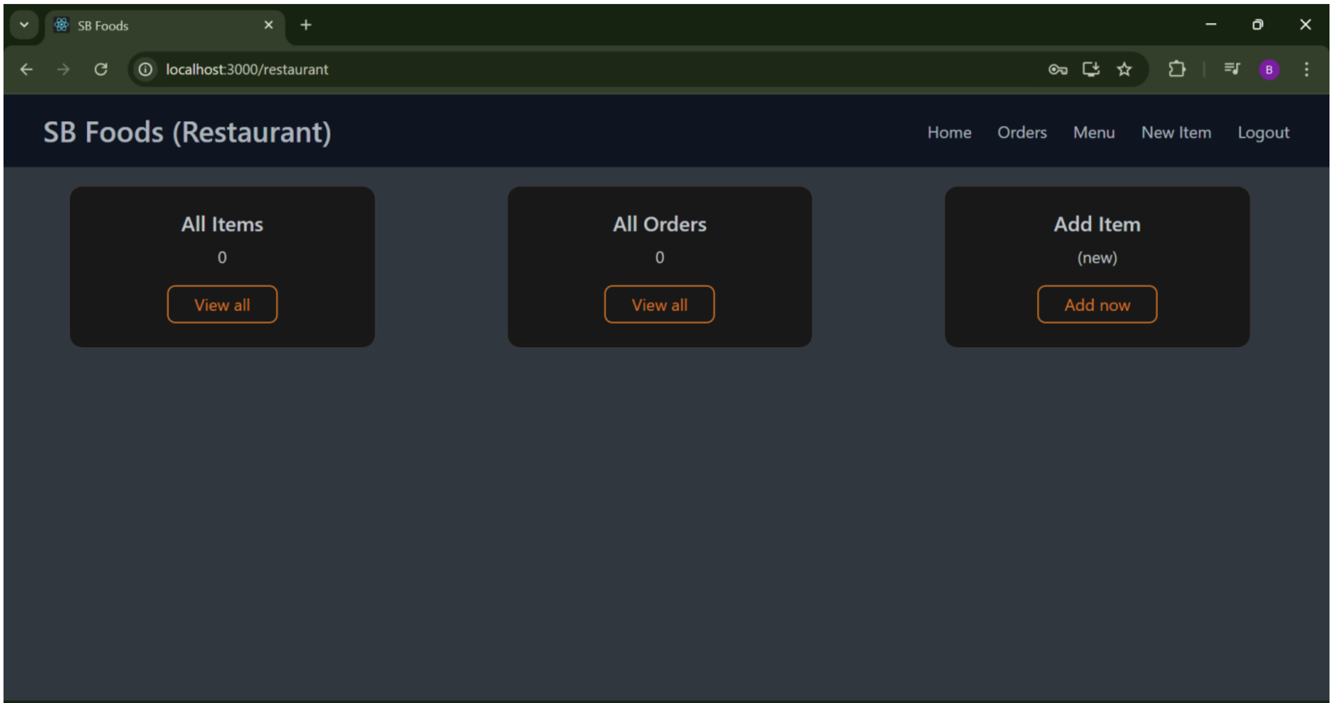
Pizza

Dominos

Quantity: 2 Total Price: ₹ 216 ~~₹ 240~~ Payment mode: cod

Ordered on: 2024-11-16 Time: 06:26 status: order placed

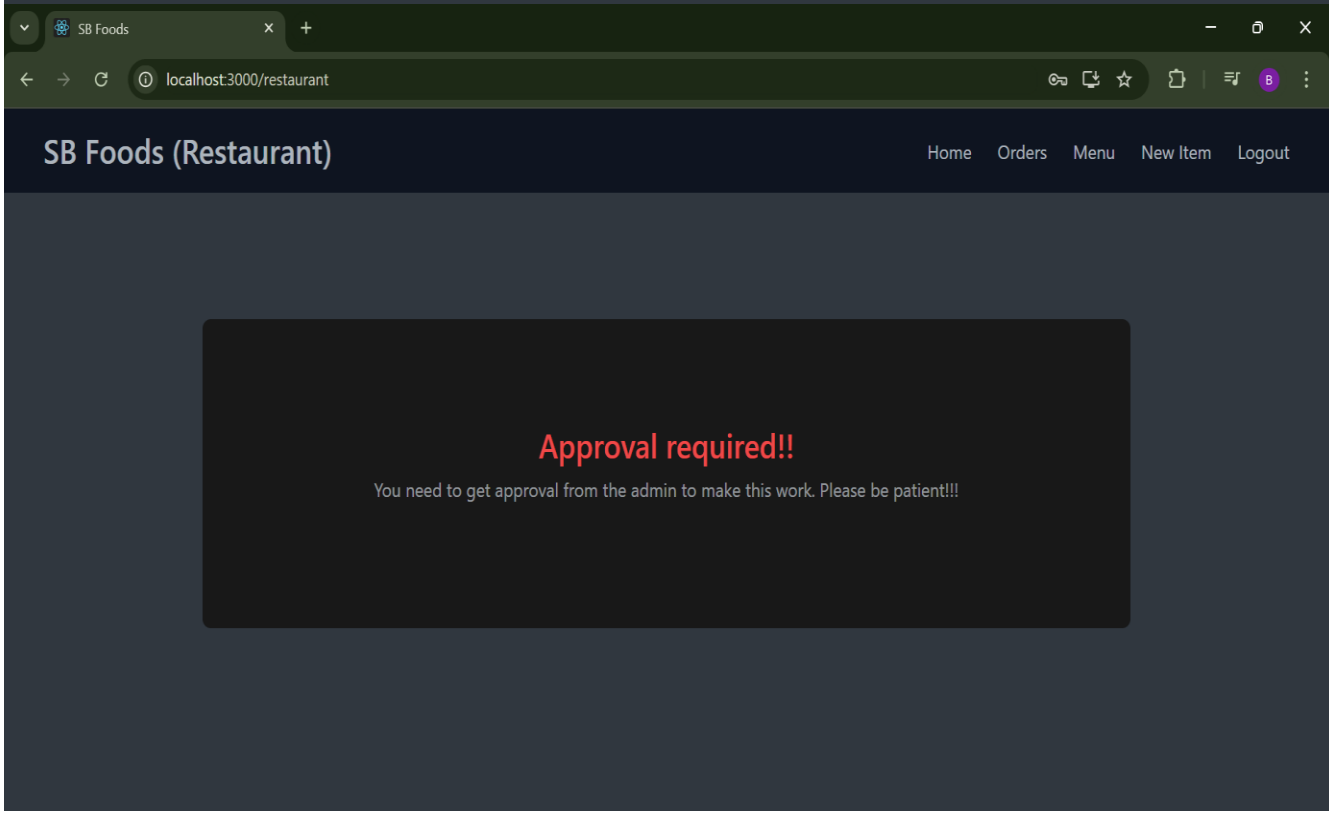
Restaurants:



All Items
0

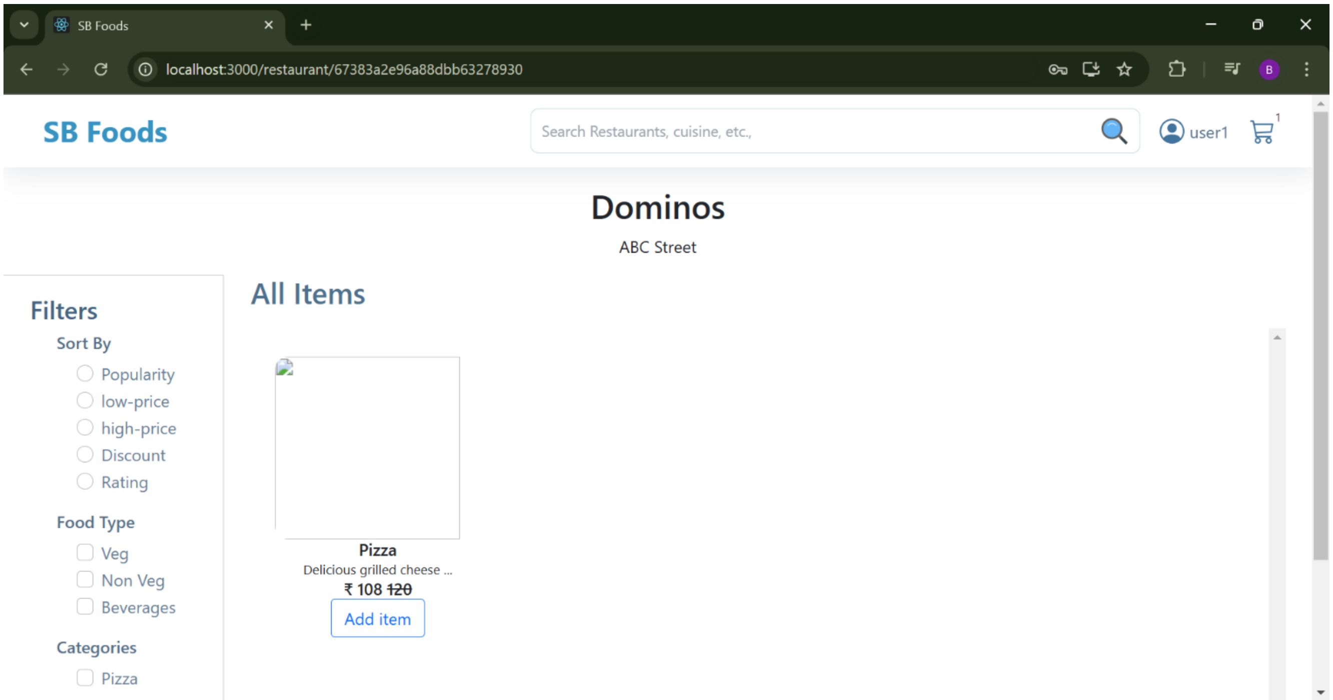
All Orders
0

Add Item
(new)



Approval required!!

You need to get approval from the admin to make this work. Please be patient!!!


SB Foods
localhost:3000/restaurant/67383a2e96a88dbb63278930
SB Foods
Search Restaurants, cuisine, etc.,
user1
1

Dominos

ABC Street

Filters

Sort By

○ Popularity
○ low-price
○ high-price
○ Discount
○ Rating

Food Type

□ Veg
□ Non Veg
□ Beverages

Categories

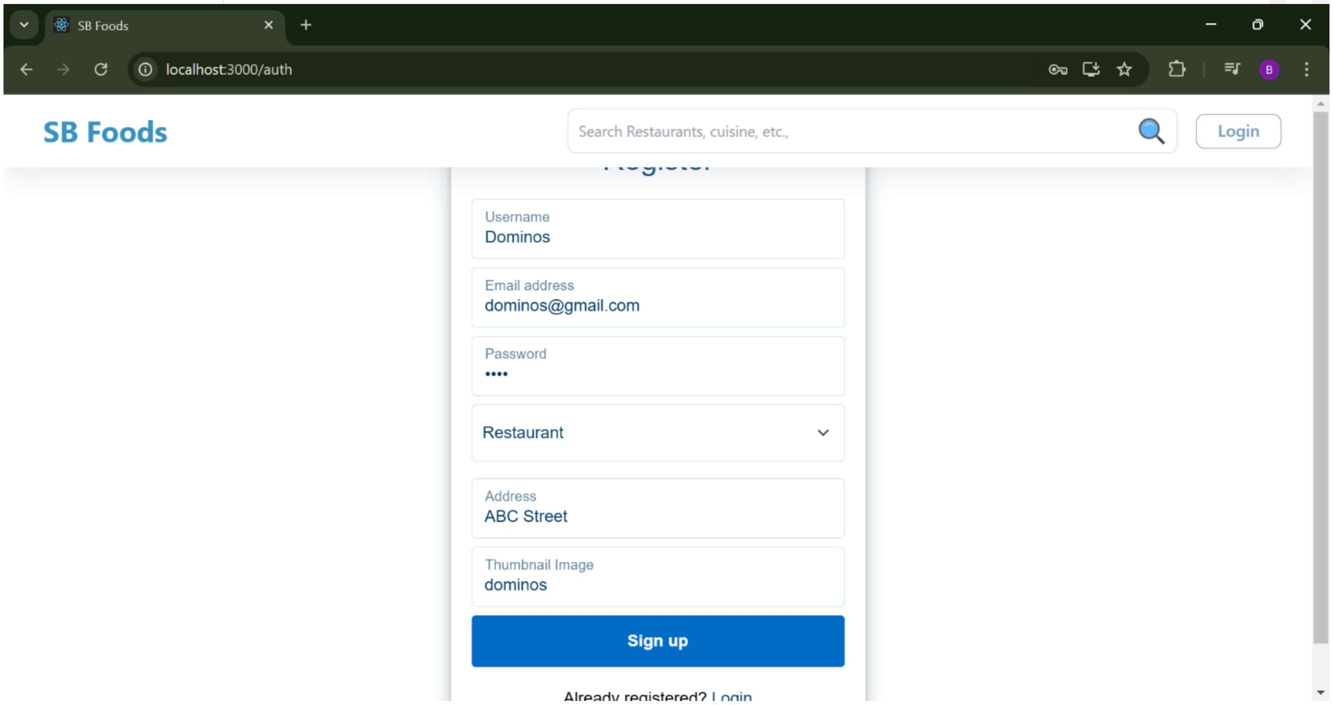
□ Pizza

All Items

Pizza

Delicious grilled cheese ...

₹ 108 ~~120~~



Username
Dominos

Email address
dominos@gmail.com

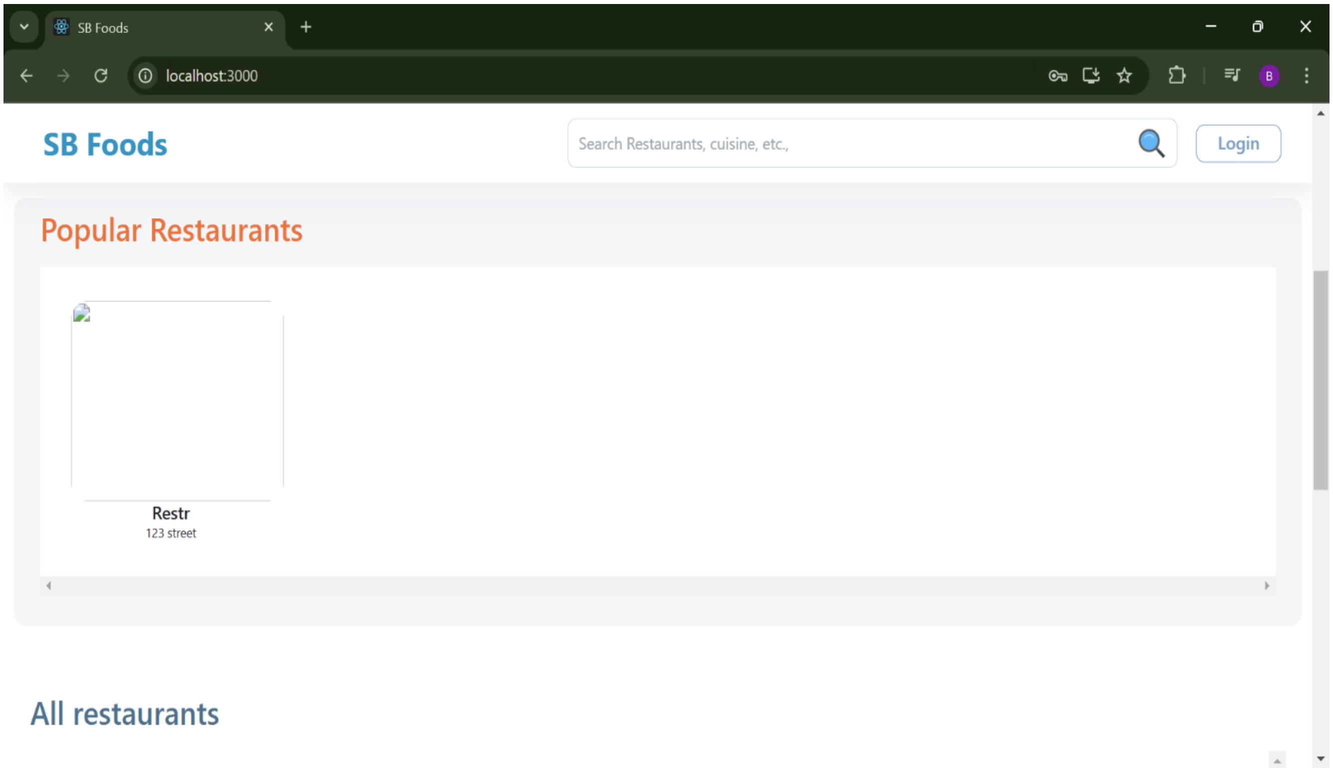
Password
••••

Restaurant

Address
ABC Street

Thumbnail Image
dominos

Already registered? Login



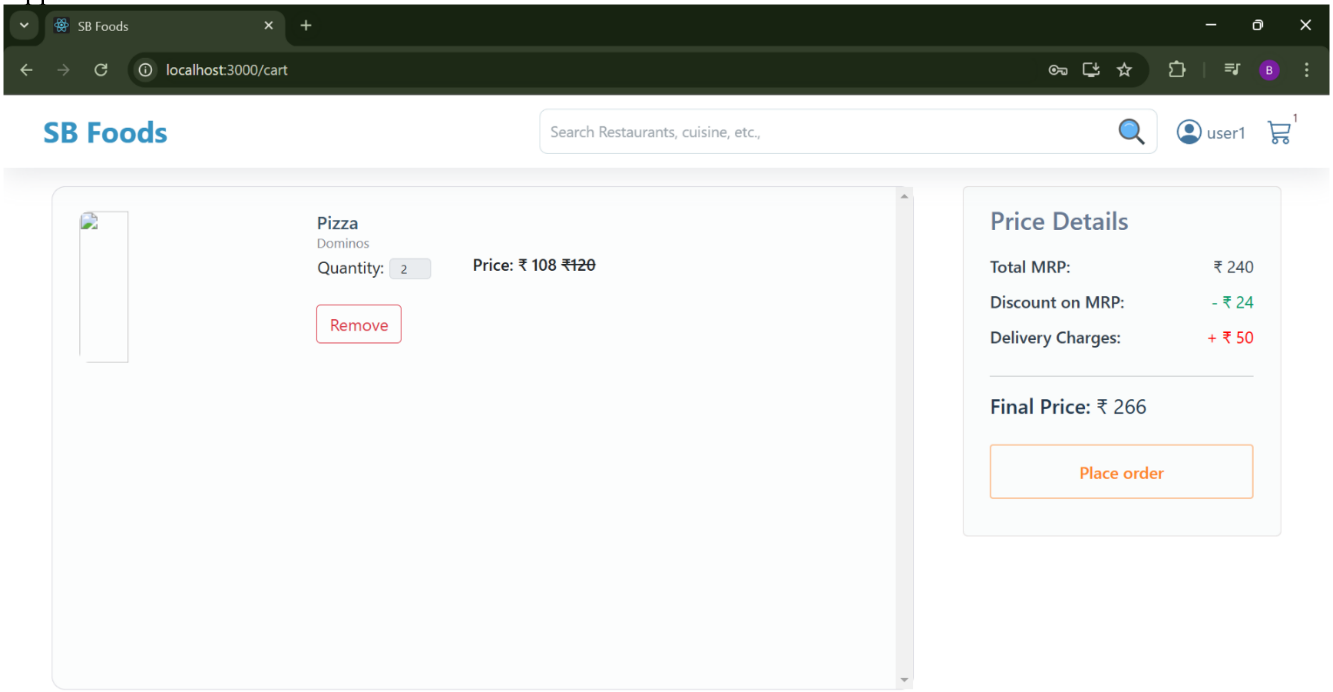
SB Foods

Popular Restaurants

Restr
123 street

All restaurants

App’s UI:



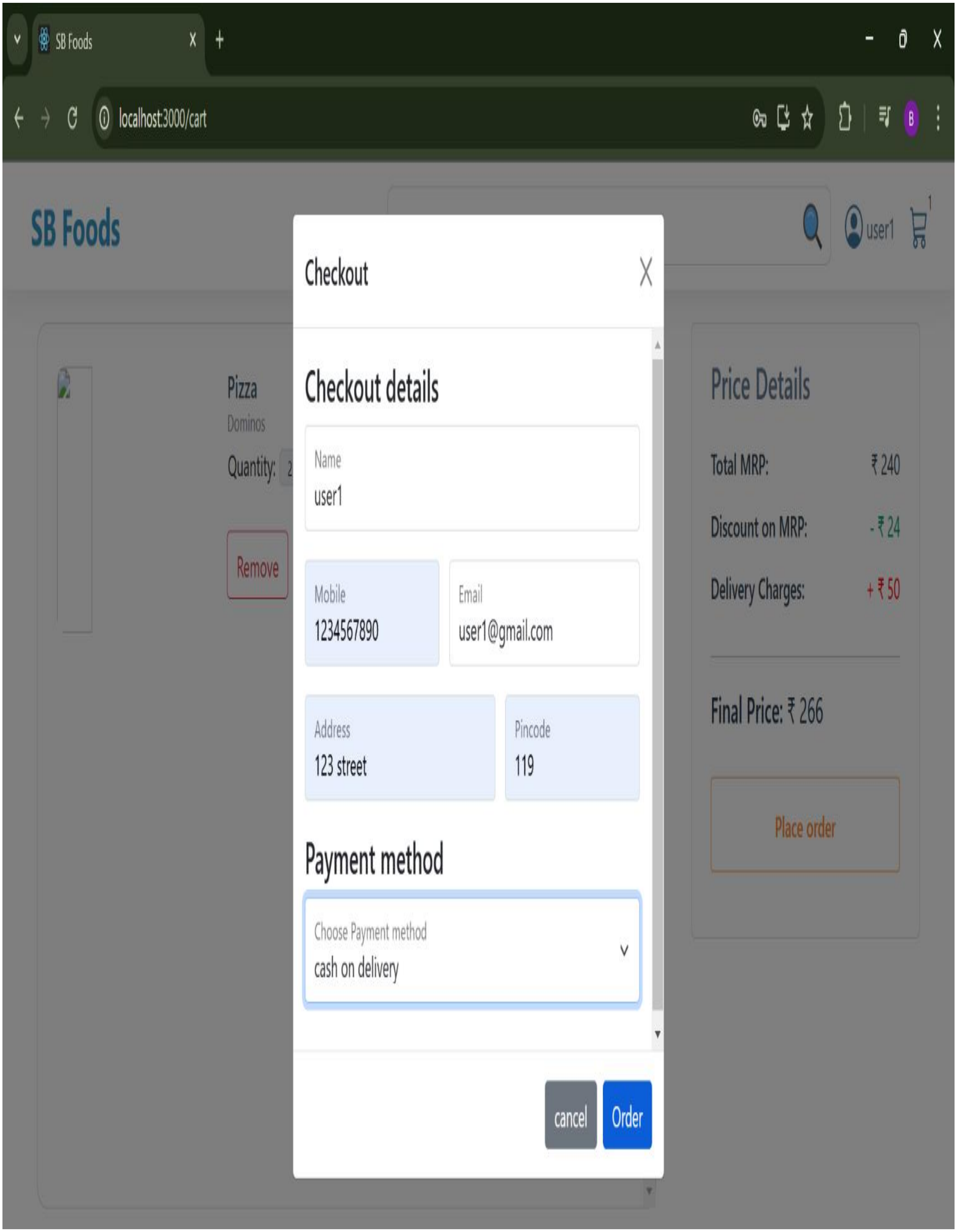
SB Foods

Pizza
Dominos
Quantity: 2
Price: ₹ 108 ~~₹120~~

Price Details

Total MRP:	₹ 240
Discount on MRP:	- ₹ 24
Delivery Charges:	+ ₹ 50

Final Price: ₹ 266



Checkout details

Name
user1

Mobile
1234567890

Email
user1@gmail.com

Address
123 street

Pincode
119

Payment method

Choose Payment method
cash on delivery

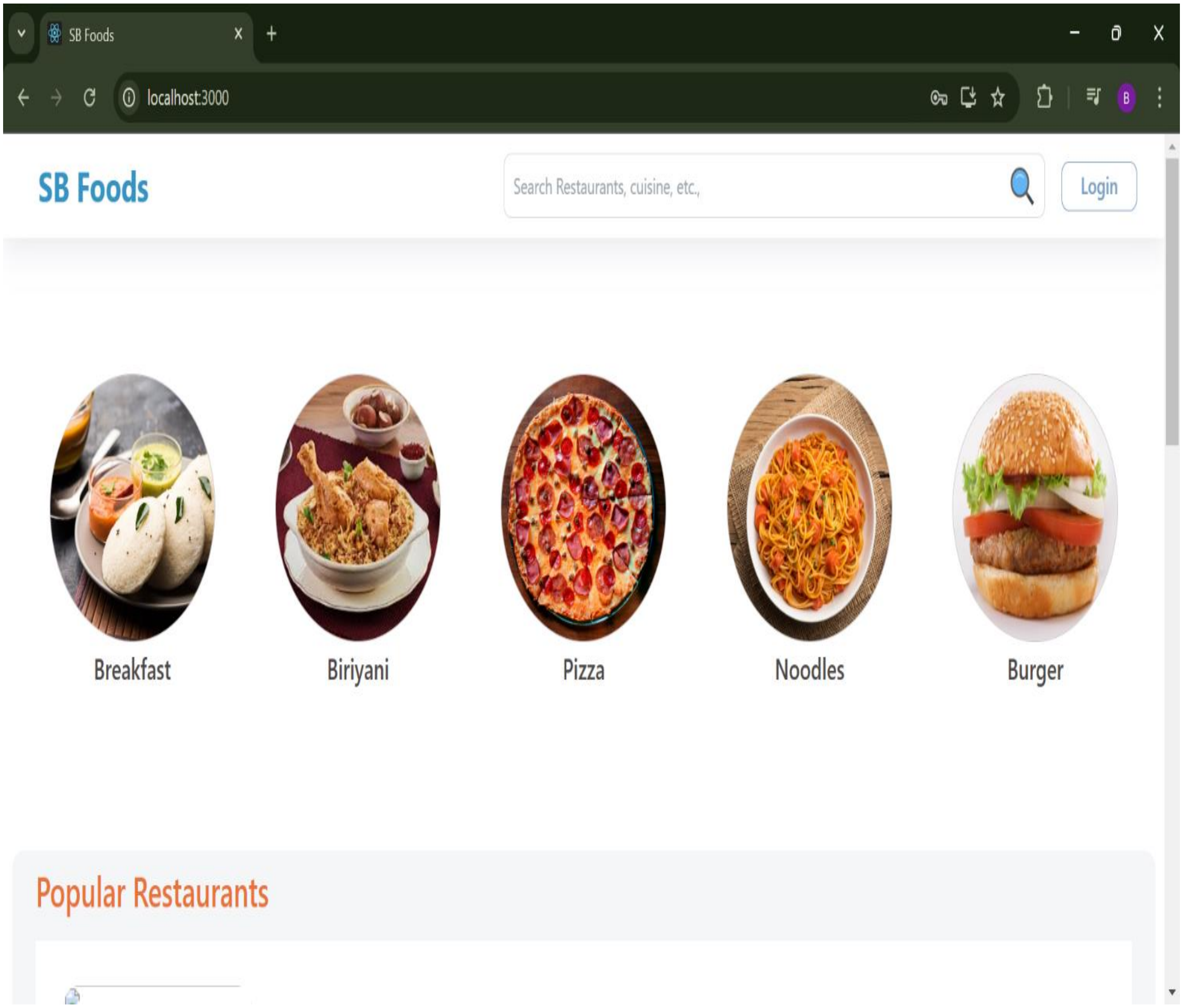
Price Details

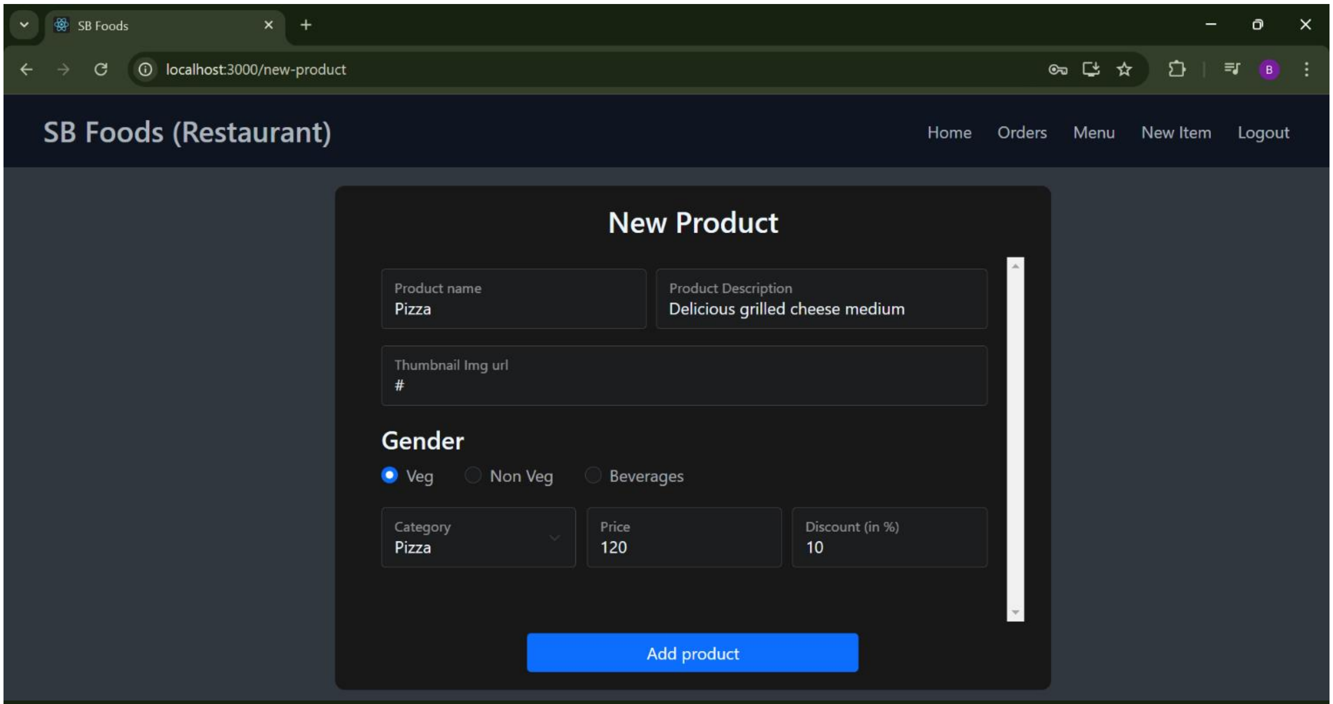
Total MRP: ₹ 240

Discount on MRP: - ₹ 24

Delivery Charges: + ₹ 50

Final Price: ₹ 266


SB Foods
localhost:3000
SB Foods
Search Restaurants, cuisine, etc.,
Login
Breakfast
Biriyani
Pizza
Noodles
Burger
Popular Restaurants



New Product

Product name: Pizza

Product Description: Delicious grilled cheese medium

Thumbnail Img url: #

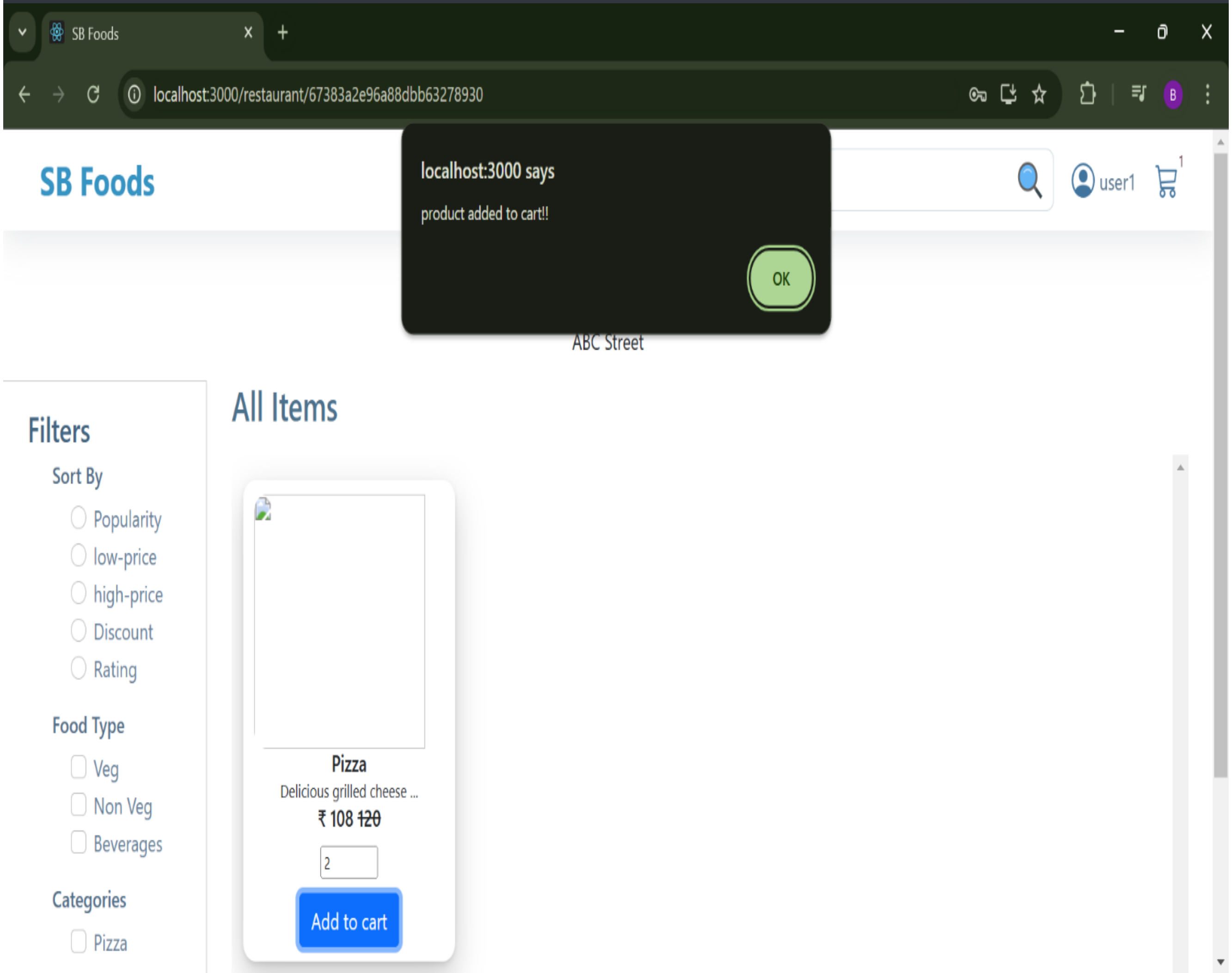
Gender

● Veg ○ Non Veg ○ Beverages

Category: Pizza

Price: 120

Discount (in %): 10



localhost:3000 says

product added to cart!!

Filters

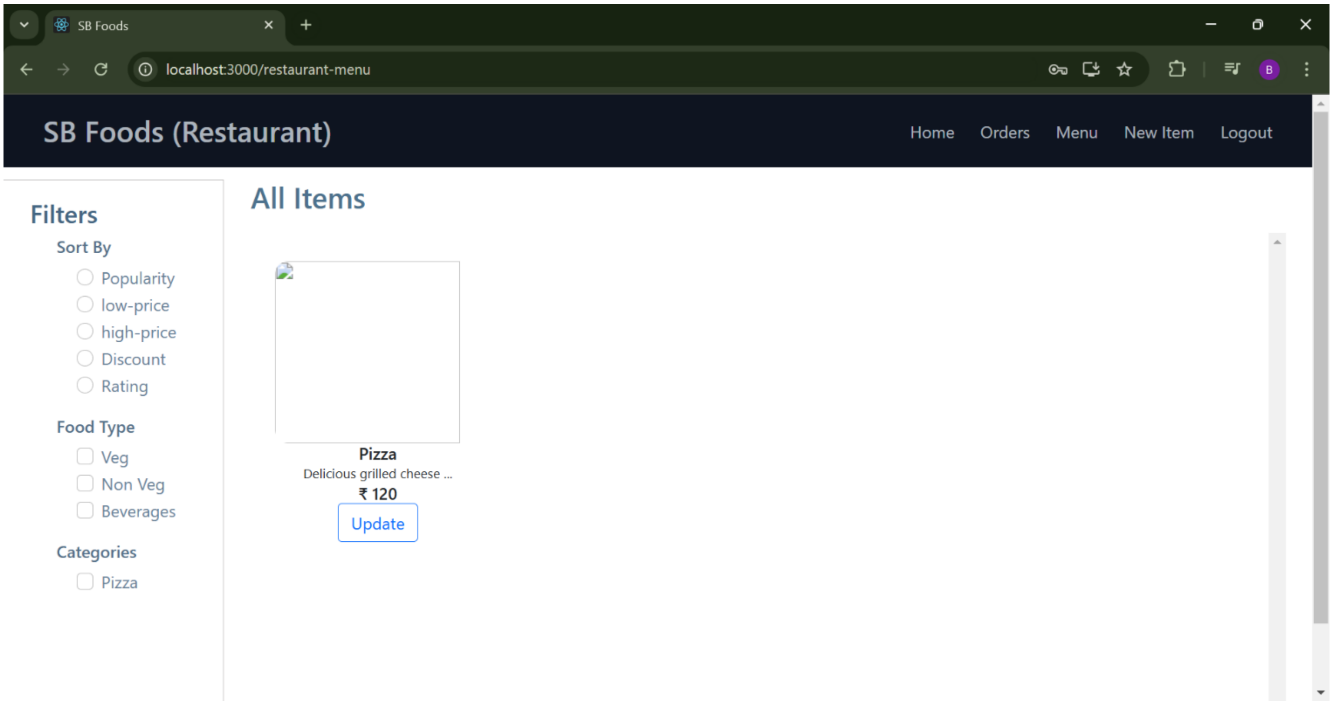
All Items

Pizza

Delicious grilled cheese ...

₹ 108 ~~120~~

2



12) Testing:

Tools:

- **Jest**:
  This comprehensive testing framework is widely used with React applications, providing a powerful combination of assertion, mocking, and snapshot testing. Jest offers efficient test running, built-in mocking capabilities for functions and modules, and detailed test result outputs, which are critical for React component testing.
- **Mocha**:
  An adaptable testing framework for Node.js, often utilized for server-side and backend testing. Mocha allows for asynchronous testing, supports hooks like before(), after(), beforeEach(), and afterEach() for controlling the testing lifecycle, and pairs well with assertion libraries like Chai.

Strategy:

1. **Unit Testing**:

   - **Objective**: Test individual units such as functions, components, and services to ensure they behave correctly in isolation.

   - **Key Activities**:

     - **Testing React Components**: Verify that React components render correctly given a set of props or state changes.

     - **Function Validation**: Test utility functions, input validation, business logic, and transformations, ensuring expected outcomes for various inputs.

     - **Mocking**: Employ Jest's mock functions to simulate dependent services, APIs, or module behaviours during testing.

2. **Integration Testing**:

   - **Objective**: Test the interaction between different components, modules, or services, ensuring that they work together as expected.

   - **Key Activities**:
     - **API Testing**: Use Mocha to test server-side RESTful API endpoints for correct HTTP responses, payloads, and error handling.

     - **Database Queries**: Validate integration with database operations to ensure queries return expected results and handle edge cases correctly.

     - **User Interaction Flows**: Test user paths that involve multiple components, simulating real-world interactions (e.g., registering, logging in, ordering food).

Technical Details:

- **Mocking and Stubbing**: Dependency injection and mocking allow isolation of units and controlled environments for reliable testing.

- **Assertions and Matchers**: Leverage Jest’s and Mocha’s built-in assertions to ensure expected values and states, checking for elements, API status codes, etc.

- **Automated Test Suites**: Organized suites for both unit and integration tests, with continuous integration hooks for automated test execution on code commits.

- **Code Coverage Analysis**: Employ coverage tools like Istanbul (integrated with Jest) to measure the extent of tested code paths, identifying potential gaps in test coverage.

13) Technical Issues:

Even though the app is working perfectly, we won’t forget the issues faced during this. As Thomas Alva Edison tried 1000 times to build a light bulb, we tried 4 times to create and make our app work. Let’s see about that now:

1. **State Management**:

- **Complexity**: Managing and synchronizing application state across numerous components, especially in large applications, can be difficult due to prop drilling and maintaining UI consistency.

- **Solution**:

  - Utilize **Redux** or **React Context** for predictable state updates, centralized state storage, and global state access.

  - Apply middleware like **Redux Thunk** or **Redux Saga** for asynchronous actions and side-effects management.

2. Database Schema Complexity:

- **Challenge**:

  - Structuring complex data relationships, such as one-to-many or many-to-many, within

MongoDB requires thoughtful modelling.

    - Nesting documents can lead to large data sizes, while referencing can cause slower joins.

- **Solution**:

    - Optimize by determining when to **embed** (for denormalized, frequently-read data) vs. **reference** (for more normalized data with less redundancy).

    - Utilize **Mongoose's schema design capabilities**, such as schema validation, population (for joins), and indexing to enhance data query efficiency.

3. Handling Edge Cases:

- **Challenge**:
  Scenarios such as user input validation, unexpected API response errors, race conditions, server outages, and invalid data can lead to application crashes or poor user experience.

- **Solution**:

    - Implement robust input validation using libraries like **Joi** and enforce type checks.

    - Use **try-catch blocks** and error-handling middleware in **Express** to manage exceptions gracefully.

    - Integrate retry mechanisms, fallbacks, and circuit breakers for network errors, and perform comprehensive testing (unit, integration, and boundary testing) to handle edge cases.

4. API Design:

- **Challenge**:
  Ensuring scalable, secure, and maintainable API design is crucial for the backend architecture in **Node.js/Express**.

- **Solution**:
    - Follow **RESTful design principles** by structuring routes cleanly (using versioning and semantic endpoints).

    - Implement **JWT (JSON Web Token)** for secure authentication and authorization mechanisms.

    - Optimize data fetching using **pagination** and **filtering** to minimize payload size and latency.

    - Use **middleware** for input validation, logging, and rate limiting to enforce security best practices.

    - Testing and documentation tools, such as **Postman** or **Swagger**, ensure proper API functionality and ease of consumption for developers.

14) Future Improvements for SB Food Ordering App:

SB Food Ordering App aims to continuously enhance the user experience by adding innovative features. Here are some key future improvements that will elevate the app's functionality and service:

1. Location-Based Services: This feature will enable the app to provide personalized recommendations based on the user's geographical location. By utilizing GPS, the app can suggest nearby restaurants, special offers, and delivery options, ensuring a more relevant and efficient food ordering experience.

2. Real-Time Order Tracking: Users will be able to track their orders from the moment they place them until they are delivered. With real-time updates on the order's status, including preparation and delivery stages, customers will have more transparency, leading to greater satisfaction and trust.

3. Credit Points and Discounts: To enhance customer loyalty, the app will introduce a system where users can earn credit points with each order. These points can be redeemed for discounts on future purchases, providing an incentive to order more frequently while saving money.

Detailed Overview:

1. Location-Based Services

By integrating GPS and geolocation services, the app will recognize the user's location and display a list of nearby restaurants. This will allow customers to choose from a wider variety of options, based on proximity and delivery availability. Furthermore, location-based promotions can be implemented, providing users with special deals in their area, encouraging them to order more frequently.

2. Real-Time Order Tracking

The addition of real-time tracking will make it easy for users to stay informed about their order's status. Notifications will update users on key stages such as "Order Confirmed," "Being Prepared," "Out for Delivery," and "Delivered." A live map feature can show the exact location of the delivery driver, offering precise delivery times. This will reduce uncertainty and improve the overall experience.

3. Credit Points and Discounts

To retain customers and foster loyalty, the app will reward users with credit points for every order placed. These points can be accumulated and redeemed for discounts on future orders. The introduction of tier-based rewards can also motivate customers to place larger or more frequent orders to unlock greater benefits. Additionally, personalized discount offers based on order history can be provided, ensuring that users feel valued and incentivized to continue using the app.

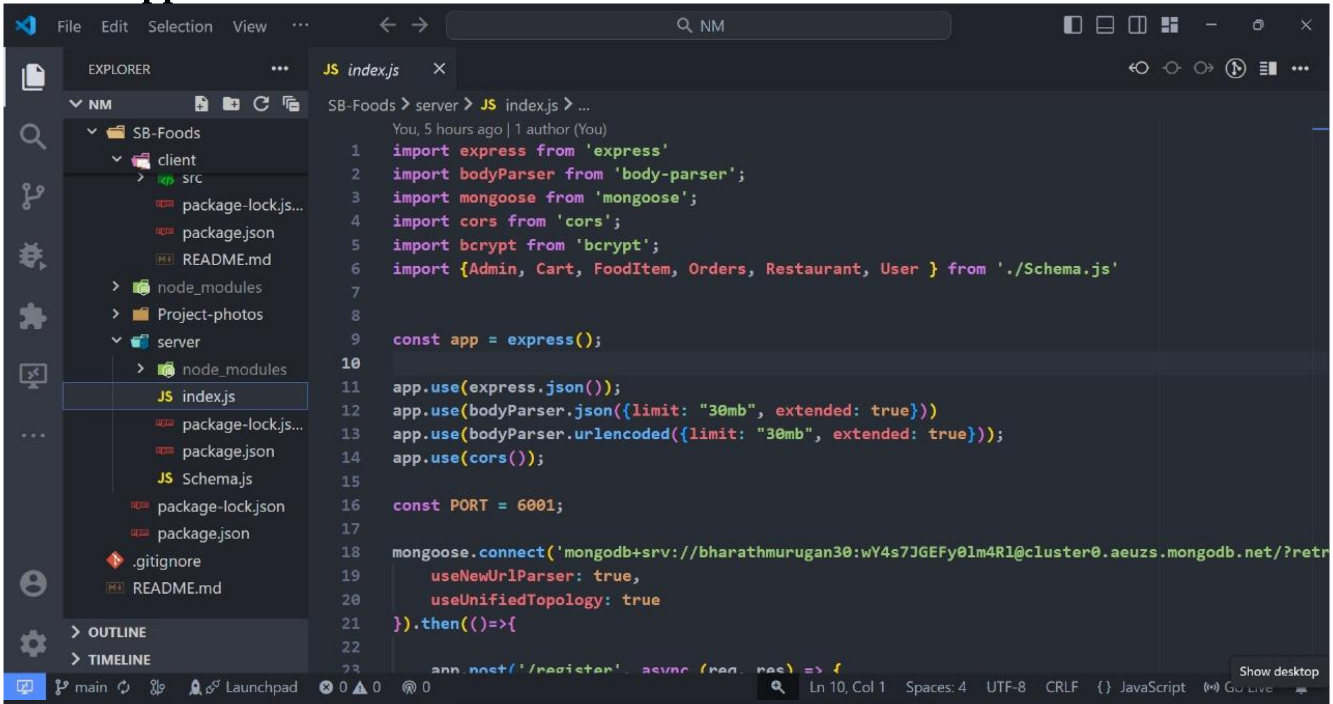
Conclusion:

The SB Food Ordering App aims to provide a seamless and enjoyable experience for users, continuously evolving to meet their needs. By focusing on improving convenience, efficiency, and user satisfaction, the app stays aligned with the core principles of innovation. As Steve Jobs once said, "You've got to start with the customer experience and work back toward the technology – not the other way around", This mindset will guide the app's development, ensuring it consistently delivers value to its users while staying ahead of technological advancements. The future of the app holds endless possibilities for enhanced user engagement.

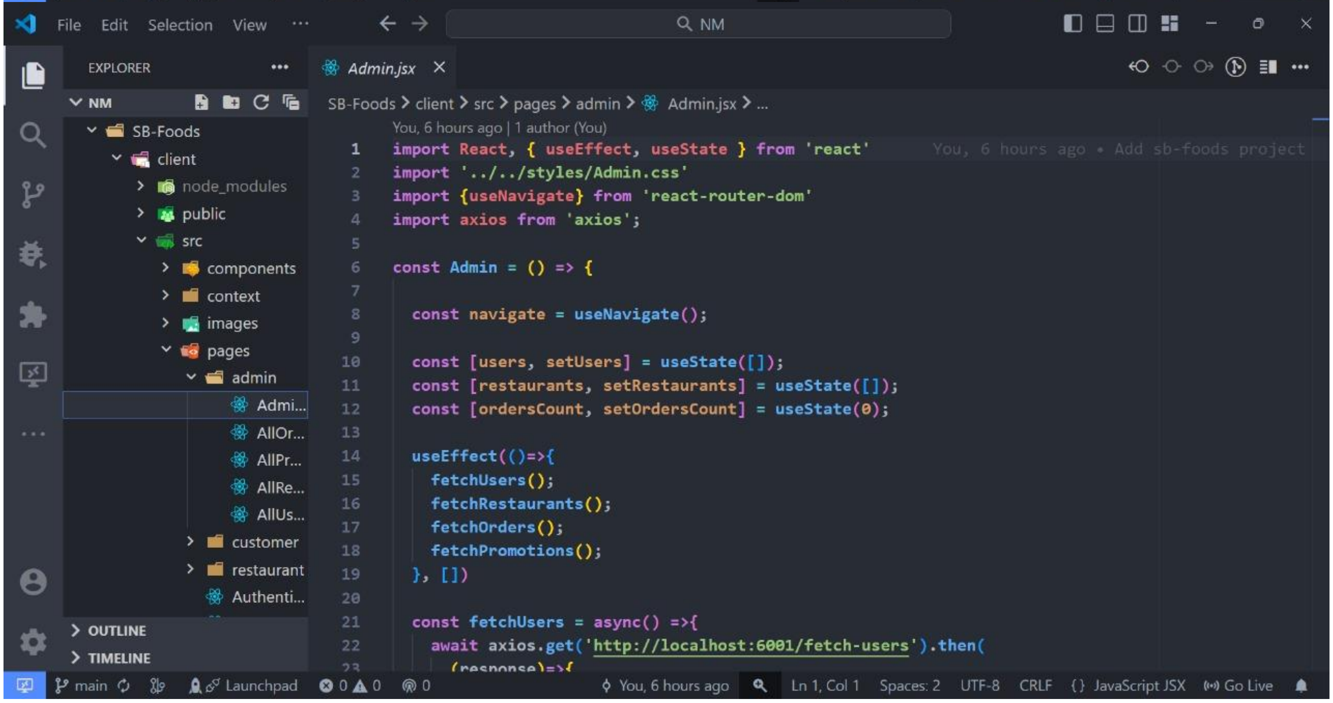
****

Appendices

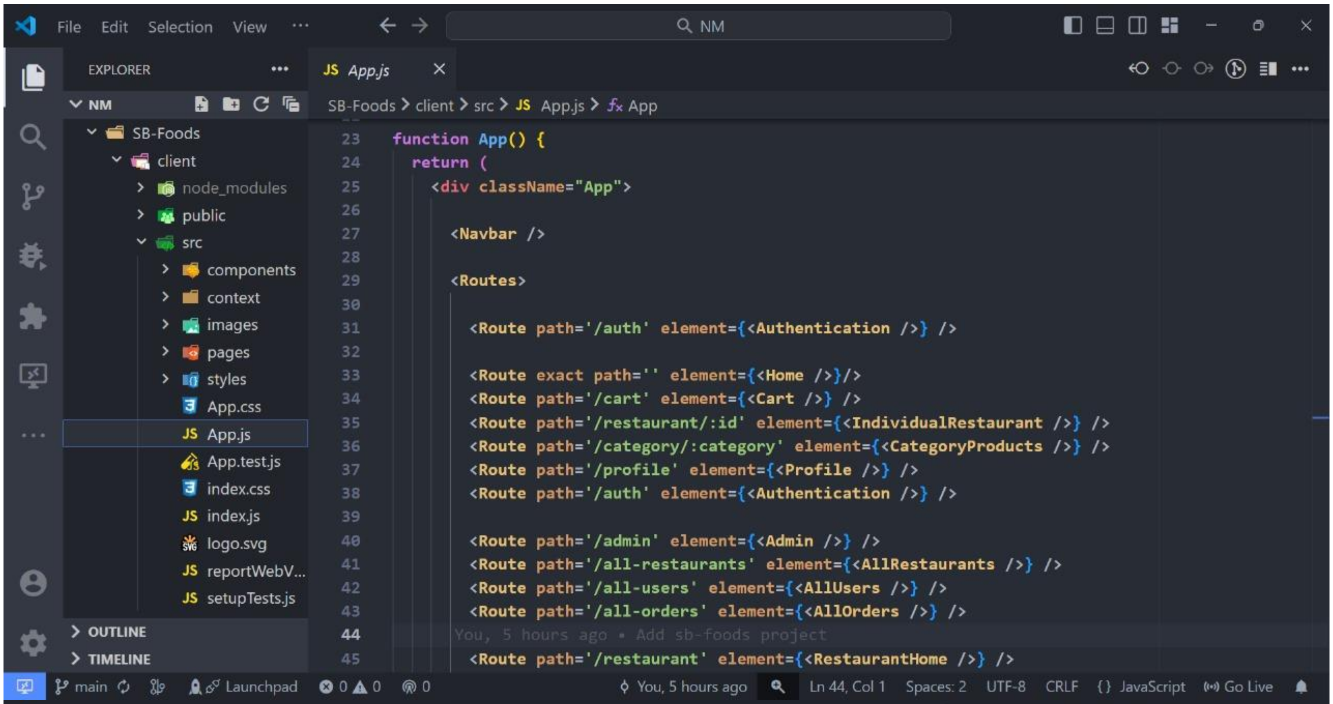
Code snippets:



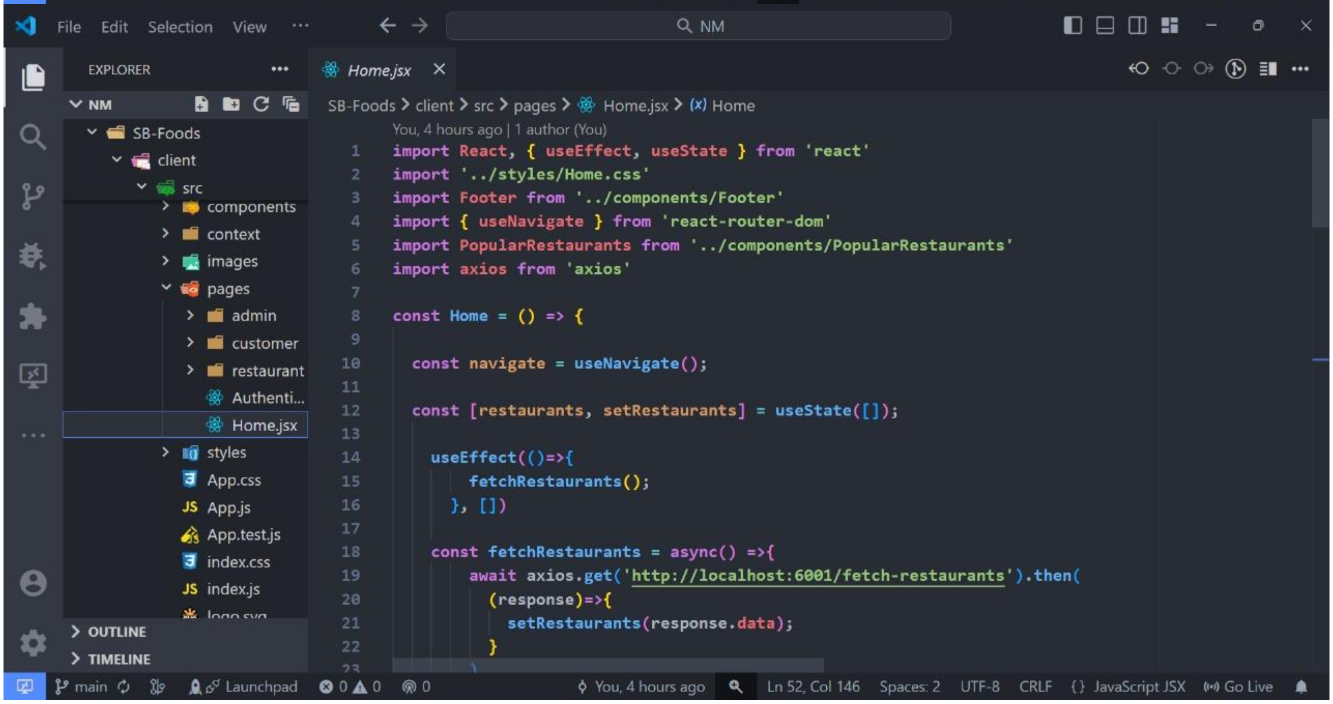
```
import express from 'express'
import bodyParser from 'body-parser';
import mongoose from 'mongoose';
import cors from 'cors';
import bcrypt from 'bcrypt';
import {Admin, Cart, FoodItem, Orders, Restaurant, User } from './Schema.js'


const app = express();

app.use(express.json());
app.use(bodyParser.json({limit: "30mb", extended: true}))
app.use(bodyParser.urlencoded({limit: "30mb", extended: true}));
app.use(cors());

const PORT = 6001;

mongoose.connect('mongodb+srv://bharathmurugan30:wY4s7JGEFy0lm4Rl@cluster0.aeuzs.mongodb.net/?retr
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(()=>{

    app.post('/register', async (req, res) => {
```



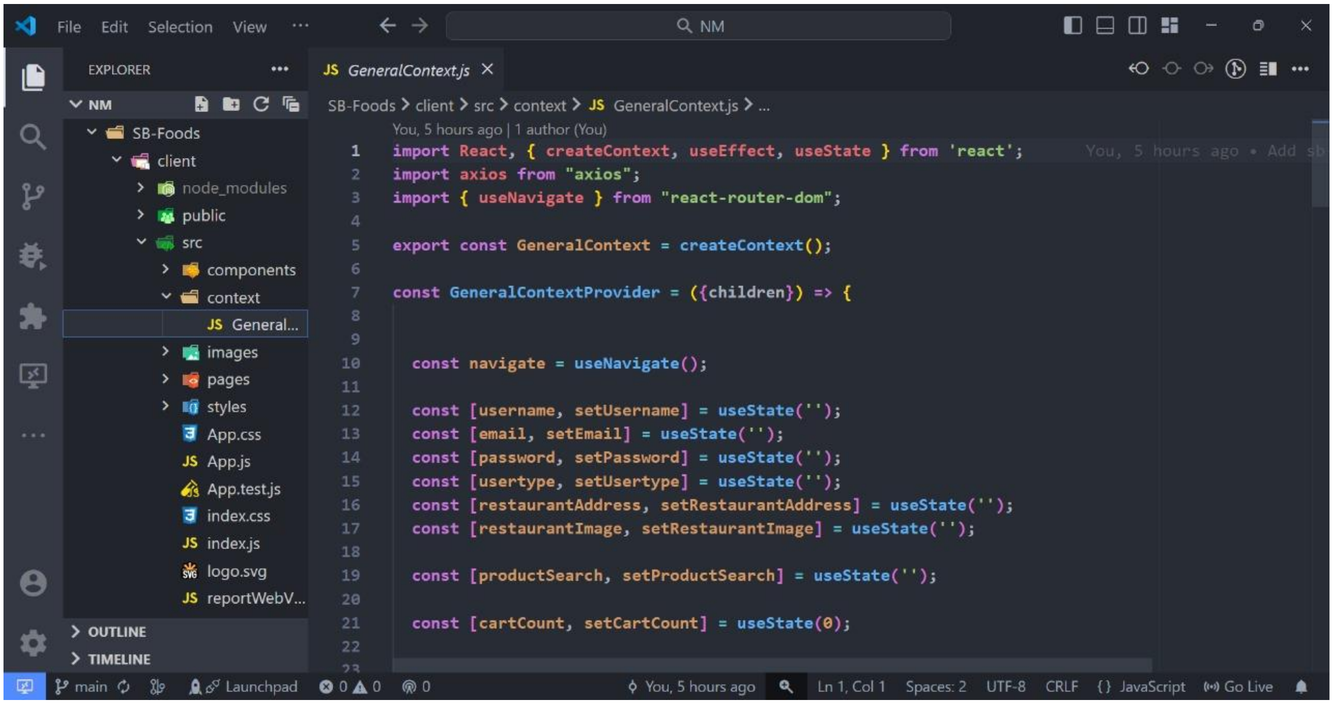
```
import React, { useEffect, useState } from 'react'      You, 6 hours ago • Add sb-foods project
import '../../styles/Admin.css'
import {useNavigate} from 'react-router-dom'
import axios from 'axios';

const Admin = () => {

  const navigate = useNavigate();

  const [users, setUsers] = useState([]);
  const [restaurants, setRestaurants] = useState([]);
  const [ordersCount, setOrdersCount] = useState(0);

  useEffect(()=>{
    fetchUsers();
    fetchRestaurants();
    fetchOrders();
    fetchPromotions();
  }, [])

  const fetchUsers = async() =>{
    await axios.get('http://localhost:6001/fetch-users').then(
      (response)=>{
```



```
function App() {
  return (
    <div className="App">

      <Navbar />

      <Routes>

        <Route path='/auth' element={<Authentication />} />

        <Route exact path='' element={<Home />}/>
        <Route path='/cart' element={<Cart />} />
        <Route path='/restaurant/:id' element={<IndividualRestaurant />} />
        <Route path='/category/:category' element={<CategoryProducts />} />
        <Route path='/profile' element={<Profile />} />
        <Route path='/auth' element={<Authentication />} />

        <Route path='/admin' element={<Admin />} />
        <Route path='/all-restaurants' element={<AllRestaurants />} />
        <Route path='/all-users' element={<AllUsers />} />
        <Route path='/all-orders' element={<AllOrders />} />
                                You, 5 hours ago • Add sb-foods project
        <Route path='/restaurant' element={<RestaurantHome />} />
```



```
import React, { useEffect, useState } from 'react'
import '../styles/Home.css'
import Footer from '../components/Footer'
import { useNavigate } from 'react-router-dom'
import PopularRestaurants from '../components/PopularRestaurants'
import axios from 'axios'

const Home = () => {

  const navigate = useNavigate();

  const [restaurants, setRestaurants] = useState([]);

    useEffect(()=>{
      fetchRestaurants();
    }, [])

    const fetchRestaurants = async() =>{
      await axios.get('http://localhost:6001/fetch-restaurants').then(
        (response)=>{
          setRestaurants(response.data);
        }
```



```javascript
import React, { createContext, useEffect, useState } from 'react';
import axios from "axios";
import { useNavigate } from "react-router-dom";

export const GeneralContext = createContext();

const GeneralContextProvider = ({children}) => {


  const navigate = useNavigate();

  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [usertype, setUsertype] = useState('');
  const [restaurantAddress, setRestaurantAddress] = useState('');
  const [restaurantImage, setRestaurantImage] = useState('');

  const [productSearch, setProductSearch] = useState('');

  const [cartCount, setCartCount] = useState(0);

```