Project document

Introduction

**Project Title**: SB-Foods Odering App

**Team Member**: Meera Jasmin R --Project Coordinator

Vanmathi N—Backend Implementation Role

Varsha M—Frontend Implementation Role

Mathumitha S –Test and Application support Role

Project overview

**Purpose:** The purpose of a food ordering app is to simplify the process of ordering meals from restaurants. It aims to provide an easy-to-use platform for browsing menus, placing orders, making secure payments, and tracking deliveries in real time. Key goals include enhancing customer convenience, streamlining restaurant operations, ensuring secure transactions, and offering personalized recommendations.

**Features:** A food ordering app allows users to register and manage their profiles, browse restaurants with detailed menus and reviews, and place customized orders effortlessly. It includes features like interactive menus with filters for dietary preferences, secure payment options such as cards, wallets, and cash on delivery, and real-time order tracking with live delivery updates. The app ensures convenience for users while streamlining restaurant operations, making it an efficient and user-friendly solution for ordering meals.

**Architecture**

**Frontend**: src/

├── assets/ # Static files like images, icons, fonts

├── components/ # Reusable components (e.g., Button, Modal, Card)

├── containers/ # Page-level components (e.g., HomePage, CartPage)

├── contexts/ # Global state management using Context API

├── hooks/ # Custom hooks (e.g., useAuth, useCart)

├── services/ # API calls (e.g., user, orders, menu)

├── utils/ # Utility functions (e.g., formatPrice, debounce)

├── styles/ # Global and modular styles

├── App.js # Main App component

├── index.js # Application entry point

├── routes.js # Centralized route definitions

**Backend:** src /

├── config/ # Configuration files (e.g., database, environment variables)

├── controllers/ # Route handlers for each feature/module

├── middle wares/ # Custom middleware for authentication, logging, etc.

├── models/ # Database models (e.g., User, Order, Menu Item)

├── routes/ # Route definitions

├── services/ # Business logic (separate from controllers)

├── utils/ # Utility functions (e.g., error handling, validation)

├── validations/ # Request validation logic (e.g., using Joi or Yup)

├── app.js # Main Express app setup

├── server.js # Application entry point

DataBase

1. **Users Collection**
   * **Stores**: User details (name, email, phone), hashed passwords, roles (customer/admin), and addresses.
   * **Purpose**: User authentication and delivery address management.
2. **MenuItems Collection**
   * **Stores**: Food item details (name, price, description, category, availability).
   * **Purpose**: Manage and display menu items to customers.
3. **Cart Collection**
   * **Stores**: User-specific cart items with quantities and total price.
   * **Purpose**: Temporary storage of selected food items before order placement.
4. **Orders Collection**
   * **Stores**: Order details (user, items, total price, delivery address, and status).
   * **Purpose**: Track order processing and delivery status.
5. **Payments Collection**
   * **Stores**: Payment info (order ID, user ID, amount, payment method, and status).
   * **Purpose**: Track payment transactions and statuses.
6. **Setup Instructions**
7. **Backend**: Node.js, Express.js, MongoDB, Mongoose, JWT, bcrypt, dotenv, cors.
8. **Frontend**: React.js, React Router, Axios, Redux Toolkit, Material-UI.
9. **Tools**: Postman, Nodemon, Git.
10. **Deployment**:Docker, AWS/Heroku/DigitalOc
11. **Folder Structure**
13. **Client**
14. . **src/ Directory**:
15. **components/**: Reusable UI components (e.g., Navbar, ProductCard, CartItem).
16. **pages/**: Individual pages (e.g., Home, Menu, Cart, Checkout).
17. **redux/**: Redux state management (actions, reducers, store).
18. **services/**: API calls to backend (e.g., api.js for fetching menu, placing orders).
19. **assets/**: Images, icons, and other static files.
20. **styles/**: Global CSS or styled-components.
21. **Routing**:
22. **React Router** to manage navigation between pages (Home, Menu, Cart, etc.).
23. **State Management**:
24. **Redux or Context API** to manage global state (user, cart items, orders).
25. **Components**:
26. Functional components for UI elements like buttons, forms, and list items.
27. **Server**

**server.js or app.js**: Main entry point to configure the app, middleware, and routes.

**controllers/**: Handles the logic for requests (e.g., user registration, placing orders, managing menu items).

**routes/** Defines Mongoose schemas for database collections (e.g., User, MenuItem, Order).

1. **utils/**: Utility functions, such as password hashing or date formatting.
2. **services/**: Handles third-party integrations, e.g., payment gateway (e.g., Stripe) or email notifications.
3. Running the Application
4. Running the Application
5. **Running the Application**
6. Frontend: npm start in the client directory
7. Backend: npm start in the server direct
8. **API Documentation**

**Authentication**

* **POST** /api/auth/register: Register a new user.
* **POST** /api/auth/login: Login and get JWT token.
* **GET** /api/users/me: Get user profile (requires JWT).

**Menu**

* **GET** /api/menu: Get all menu items.
* **GET** /api/menu/:id: Get a specific menu item by ID.
* **Cart**

**GET** /api/cart: Get current cart.

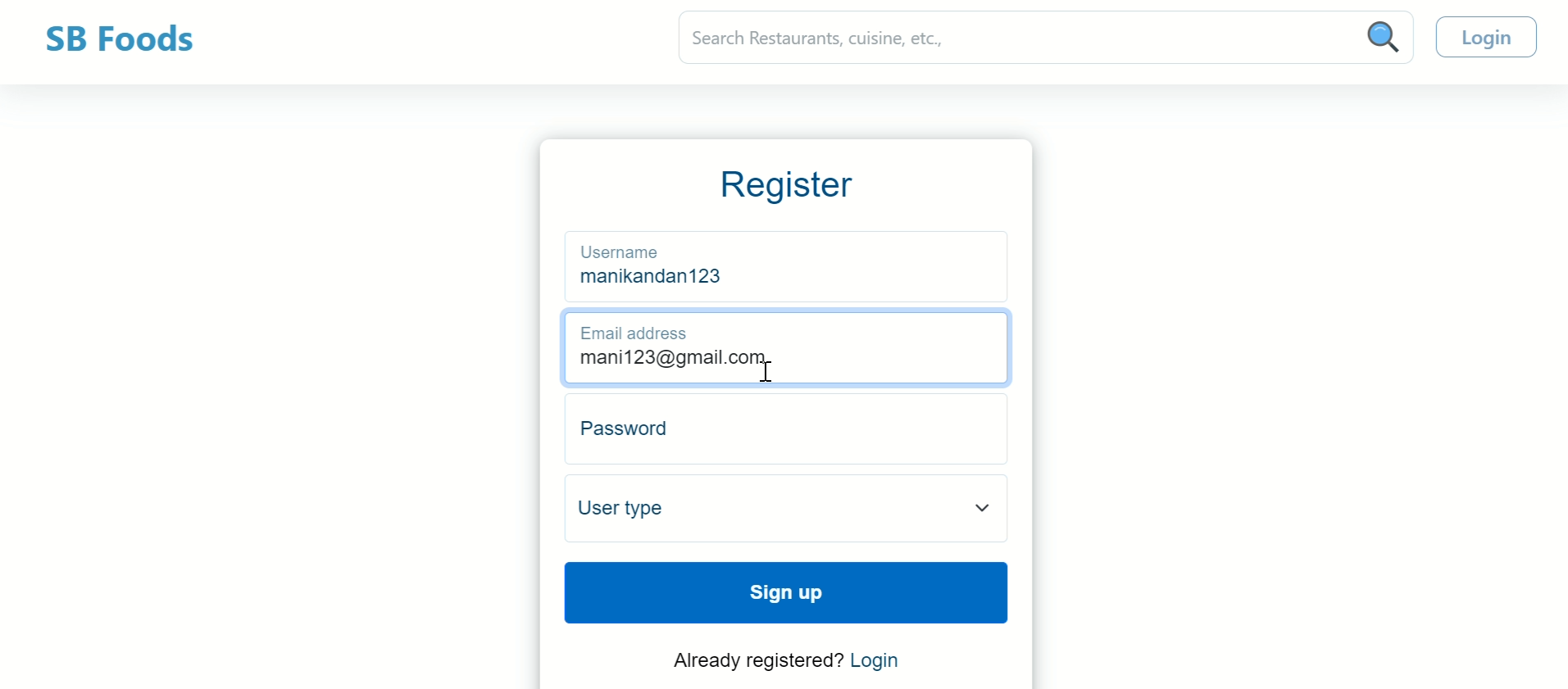
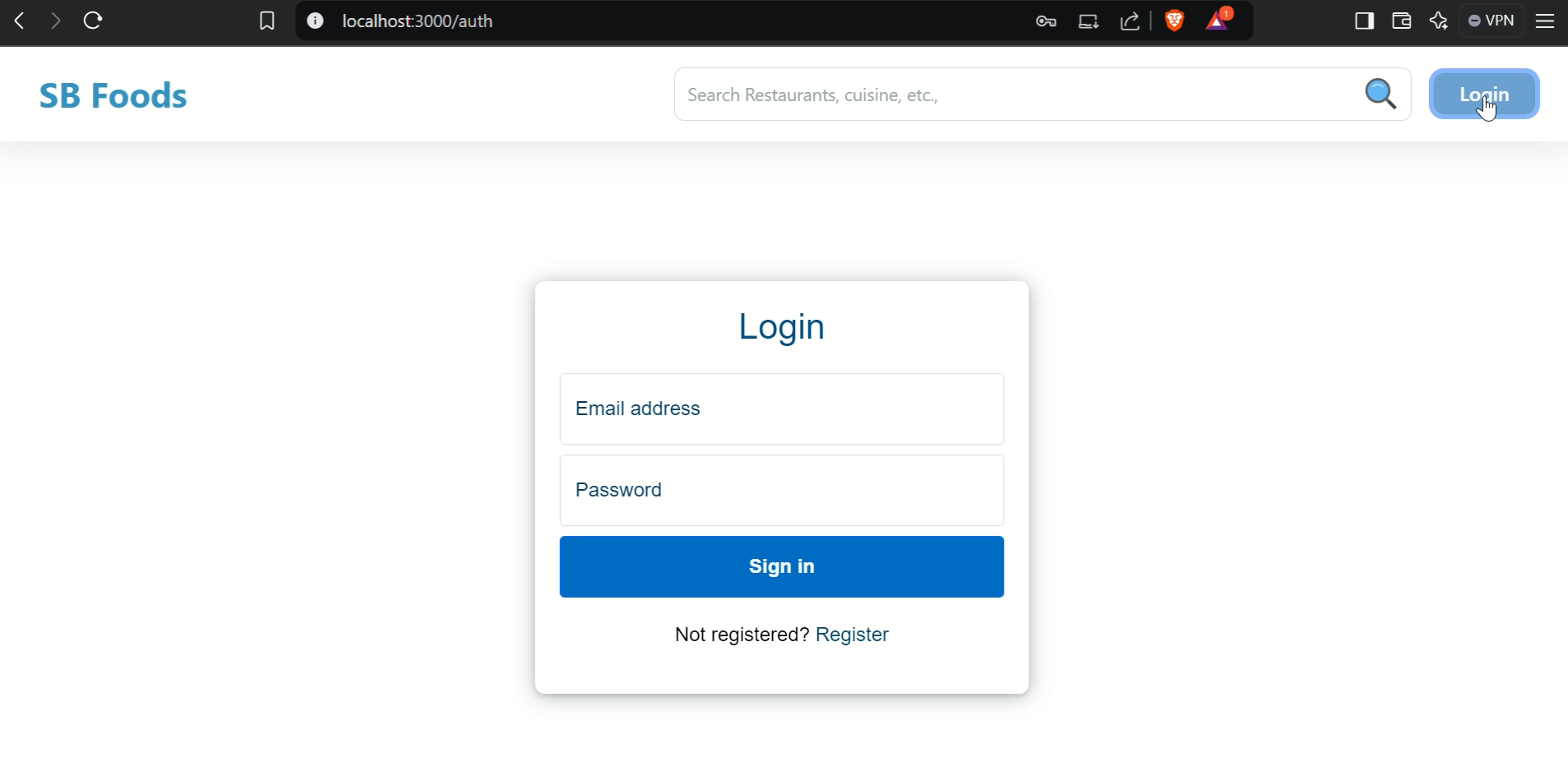
* **POST** /api/cart: Add item to cart.
* **DELETE** /api/cart/:id: Remove item from cart.
* **Orders**
* **POST** /api/orders: Place an order (from cart).
* **GET** /api/orders: Get all orders.
* **GET** /api/orders/:id: Get specific order details.
* **Payments**
* **POST** /api/payments: Process payment for an order.
* **GET** /api/payments/:id: Get payment status.

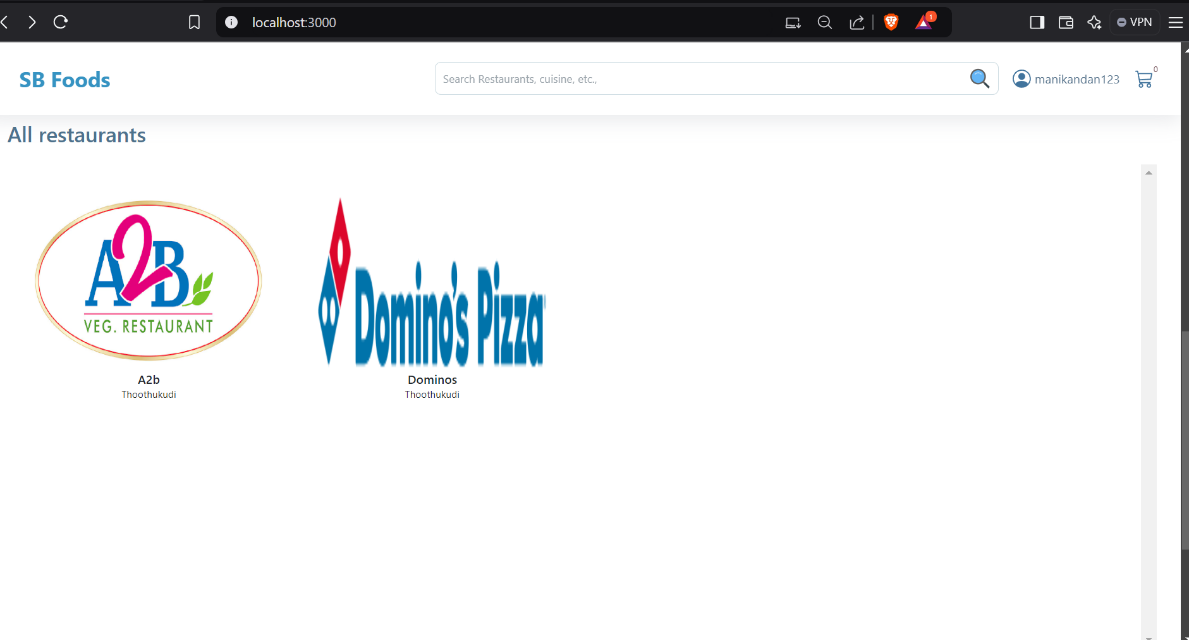
**Authentication**:

* **JWT (JSON Web Token)** is used for user authentication.
* Users register with their credentials, and upon successful login, they receive a JWT token.
* The token is stored on the client-side (e.g., in localStorage) and included in the Authorization header of requests to access protected routes.

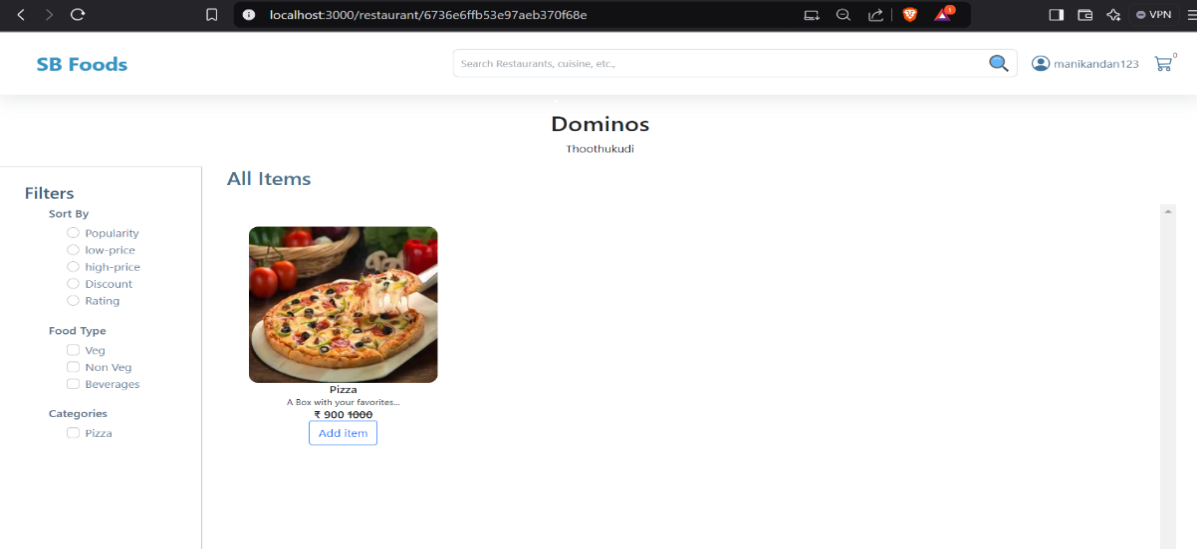
**User Interface**

**Admin Dashboard**

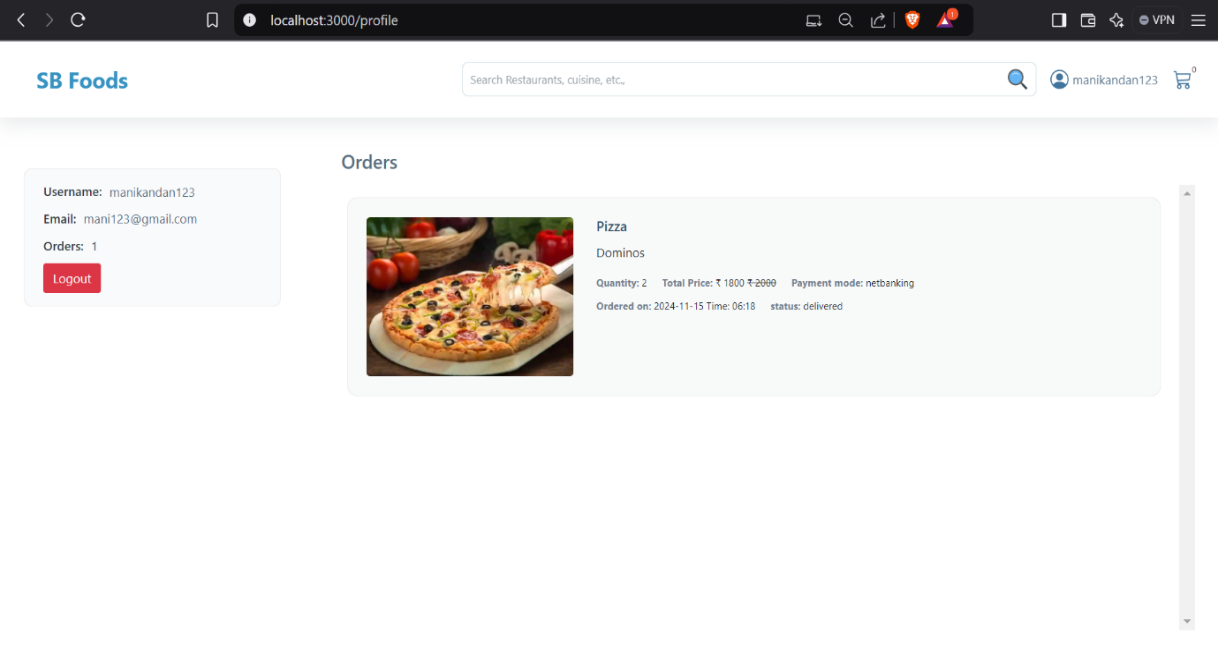
* 
* 
* **Available Restaurant**



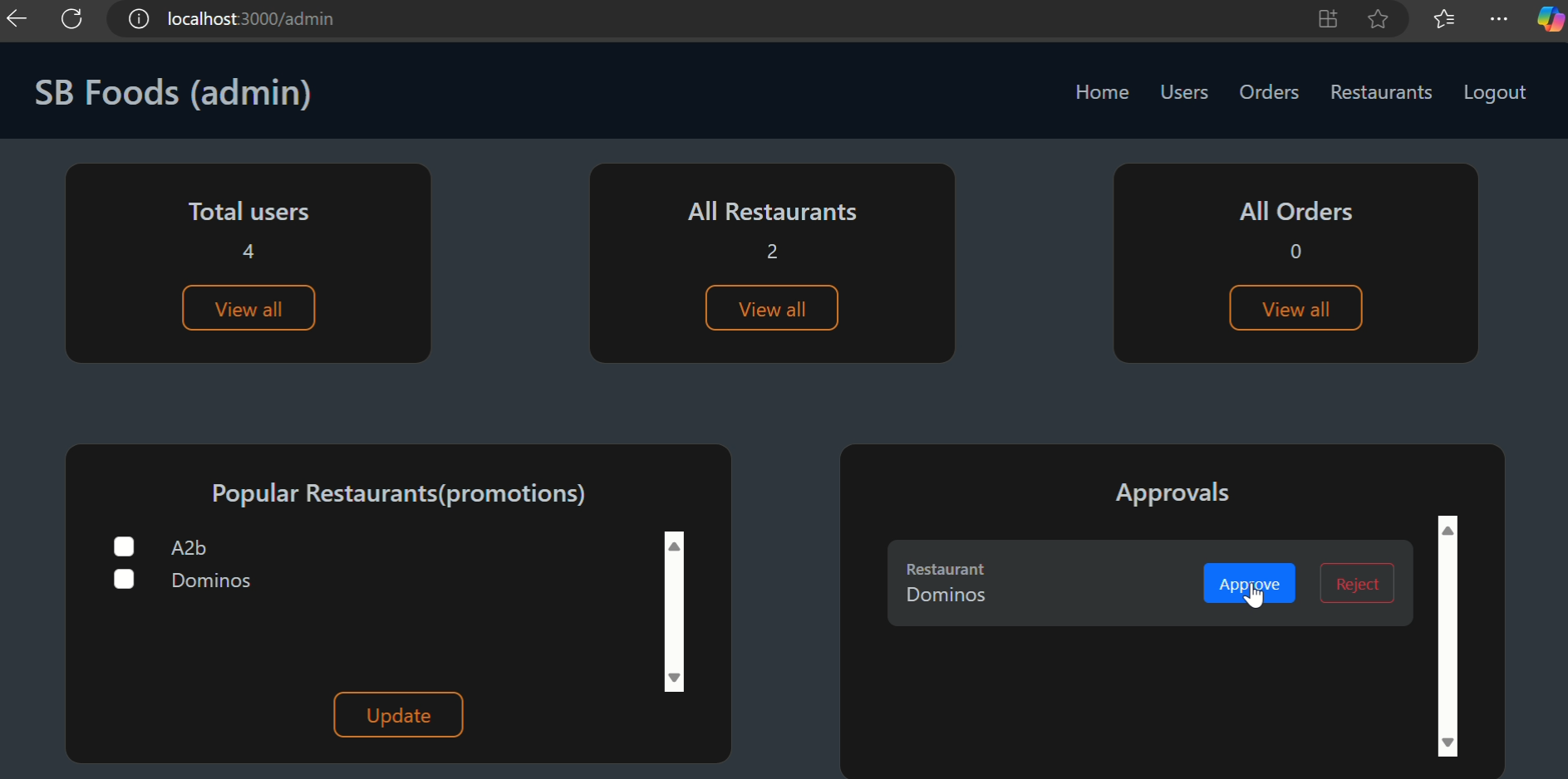
**Restaurant Menu**



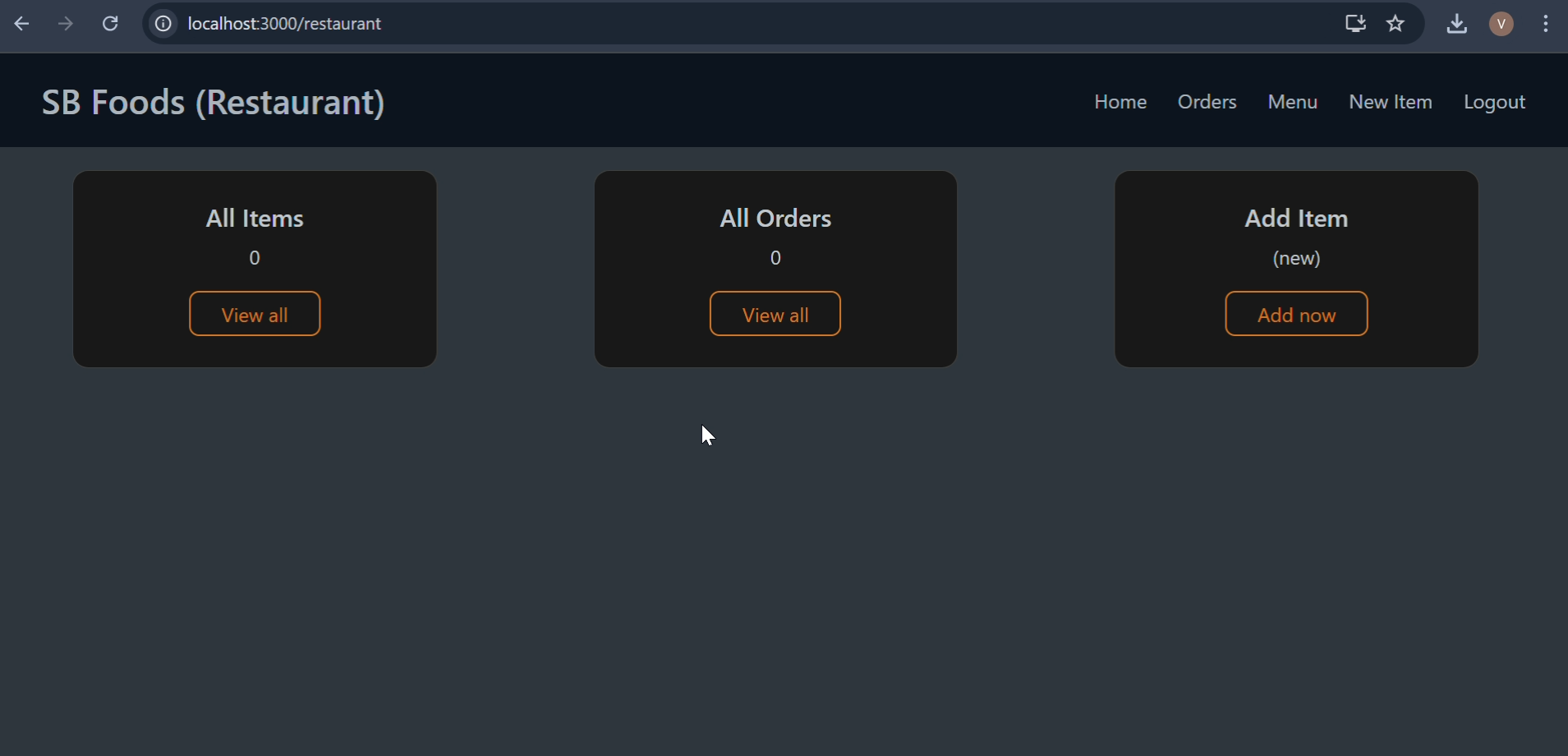
**User profile**



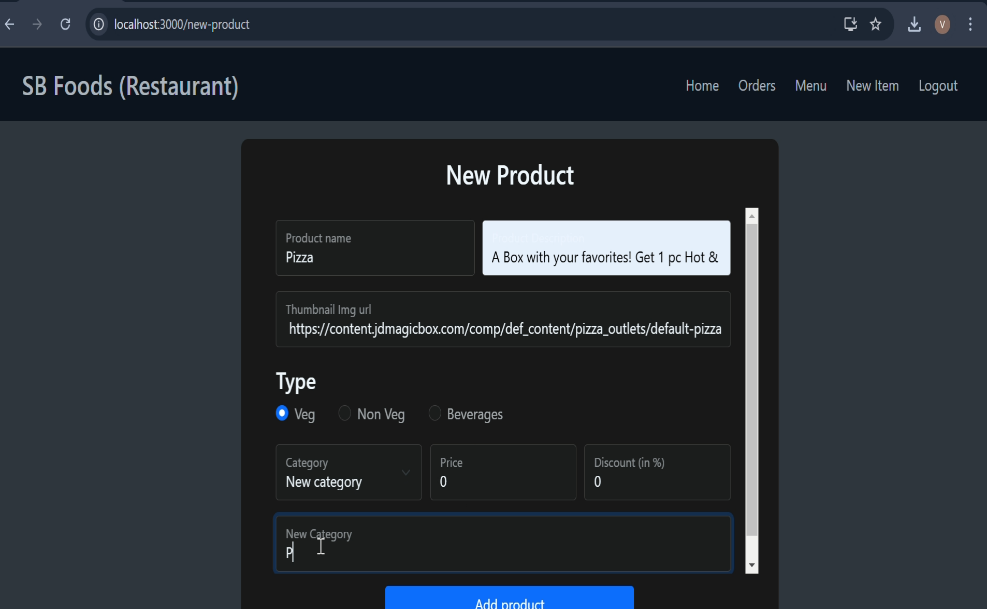
**Admin Dashboard**



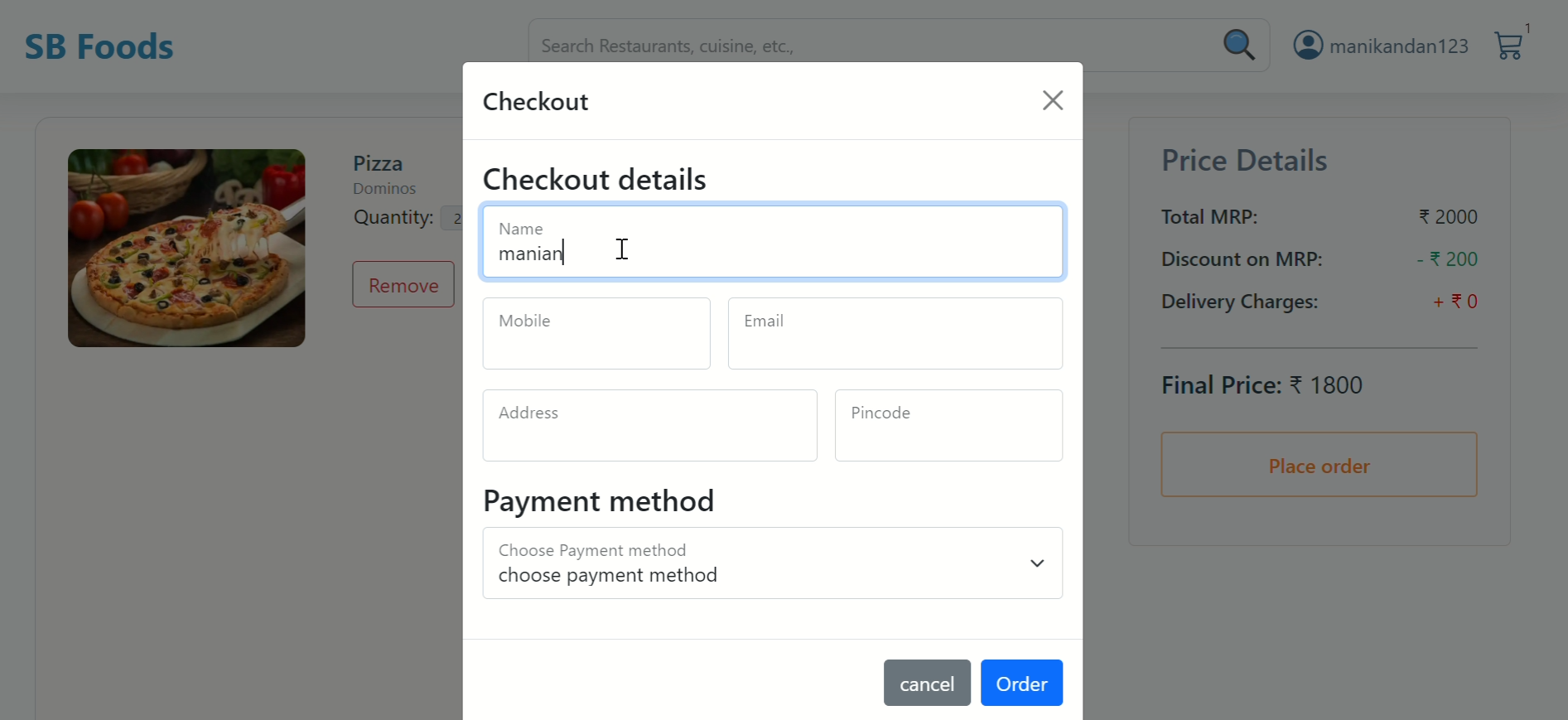
* **Restaurant DashBoard**



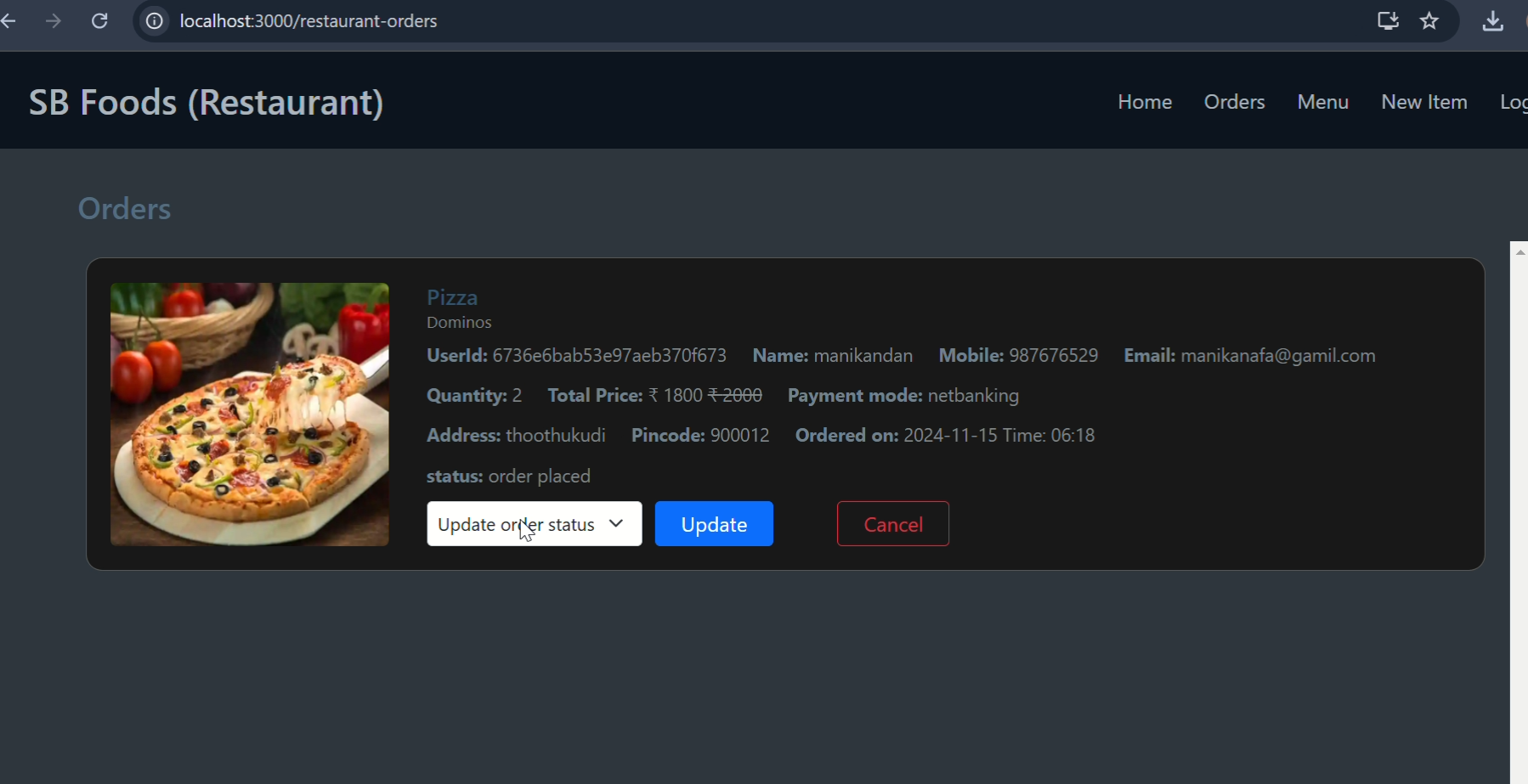
**New product**



**Check out page**



**All orders status**



**Known Issues**

* Token expiration may log users out unexpectedly.
* multiple items load is delays in real-time

**Future Enhancements**

* Introduce Special Meal Plans: Offer curated meal plans based on user preferences and dietary needs.
* Implement a Payment Gateway: Enable secure payments for premium dishes and exclusive promotions.
* Enhance Real-Time Communication: Integrate live video streaming for interactive cooking classes and events.