Full Stack Development with MERN

Project Documentation

1. Introduction

- **Project Title:** Food ordering app
- **Team Members:**

  - Kishore Babu. U.
  - Buvaneswaran. A.S.
  - Sabaregirivasan. M.

2. Project Overview

- **Purpose:** The application will cater to both customers and restaurant owners, providing a user-friendly interface allowing users to order food online from multiple restaurants.
- **Features:**

I. Customer-Facing Features

1. **User Registration and Login:** Secure authentication using JWT, social media login options.
2. **Menu Browsing:** Comprehensive restaurant and menu browsing with search filters for cuisine, ratings, price range, and more.
3. **Order Placement:** Easy-to-use cart functionality, customization options for orders, and promo code application.
4. **Order Tracking:** Real-time tracking of order status from preparation to delivery with notifications.
5. **Order History:** Ability view order history for easy reordering and to see order details.

II. Restaurant-Facing Features

1. **Restaurant Registration and Login:** Secure registration process with verification and profile setup.
2. **Menu Management:** Add, edit, and remove items from the menu, including images, descriptions, prices, and availability.
3. **Order Management:** Real-time dashboard for managing incoming orders, updating order status, and handling special instructions.
4. **Promotion Management:** Create and manage promotional offers and discounts to attract more customers.
5. **Inventory Management:** Track inventory levels and receive alerts for low-stock items.

III. Admin-Facing Features

1. **Admin Dashboard:** Comprehensive dashboard providing an overview of application performance, user statistics, and system health.
2. **User Management:** View, manage, and moderate customer and restaurant accounts, including the ability to suspend or delete accounts.
3. **Content Moderation:** Monitor and approve/reject restaurant profiles, menu items, and customer reviews.
4. **Order Monitoring:** Oversee order activity across the platform, resolve disputes, and ensure compliance with platform policies.
5. **Analytics and Reporting:** Generate detailed reports on user activity, sales performance, and platform usage.
6. **Role-Based Access Control:** Define and manage different access levels for staff members, ensuring secure and appropriate access to features.
7. **Promotions and Advertisements:** Create and manage global promotions and advertisements to be displayed to users.

3. Architecture

- **Frontend:**

1. Public Directory

- Purpose: Hosts static files and the main entry point for the web application.

- Key Files:

- favicon.ico: The icon displayed in the browser tab.

- index.html: The starting file containing meta links, external sources, and script references.

- manifest.json: Configures web app details like name, icons, and start URL.

- robots.txt: Instructs web crawlers on which pages to index.

2. Src Directory

- Purpose: Contains all source code and main logic of the React application.

- Key Files:

- index.js: The entry point of the React application.

3. Components Directory

- **Purpose:** Houses reusable UI components, promoting modularity and code reuse.

- **Key Components:**

  - **Footer.jsx:** The footer section.

  - **Login.jsx:** The login page.

  - **Navbar.jsx:** The navigation bar.

  - **PopularRestaurants.jsx:** Displays popular restaurants.

  - **Register.jsx:** The registration page.

  - **Restaurants.jsx:** Displays a list of restaurants.

  - **Authentication.jsx:** Manages authentication (login/register).

  - **Home.jsx:** The main home page component.

4. Context Directory

- **Purpose:** Manages global state efficiently using React Context API.

- **Key Files:**

  - **GeneralContext.js:** Provides context providers and consumers for global state management.

5. Pages Directory

- **Purpose:** Contains components that represent different routes in the application.

- **Subdirectories:**

  - **Admin:** Components for administrative functions.

    - **Admin.jsx:** Main admin dashboard.

    - **AllOrders.jsx:** Displays all orders.

    - **AllProducts.jsx:** Displays all products.

    - **AllRestaurants.jsx:** Displays all restaurants.

    - **AllUsers.jsx:** Displays all users.

  - **Customer:** Components for customer-related functionalities.

- **Cart.jsx:** Shopping cart.

- **CategoryProducts.jsx:** Displays products by category.

- **IndividualRestaurant.jsx:** Displays details of a single restaurant.

- **Profile.jsx:** Displays and edits user profiles.

- **Restaurant:** Components for restaurant-specific operations.

- **EditProduct.jsx:** Edits a product.

- **NewProduct.jsx:** Adds a new product.

- **RestaurantHome.jsx:** Restaurant home page.

- **RestaurantMenu.jsx:** Displays the restaurant menu.

- **RestaurantOrders.jsx:** Displays restaurant-related orders.

6. Styles Directory

- **Purpose:** Contains CSS stylesheets for individual components and global styles.

- **Contents:** CSS files organized to maintain style consistency and separation of concerns.

7. JavaScript Files

- **Purpose:** Contains essential JavaScript files for the app's functionality.

- **Key Files:**

- **App.js:** Main application component.

- **App.test.js**: Tests for the app component.

- **index.js:** React application's entry point.

- **reportWebVitals.js**: Measures performance metrics.

- **setupTests.js:** Configuration for setting up tests.

8. Root Files

- **Purpose:** Hosts configuration and metadata files.

- **Key Files:**

- **.gitignore:** Specifies files and directories to be ignored by Git.

- **package.json:** Lists project dependencies and scripts for managing the application.

- **Backend:**

    1. **Server directory:**

- Purpose: To host the server and to connect database and also to post and get API's

- Key files:

-index.js: it servers as the entry points for the backend servers for the application and it creates express application ,configures middleware connecting with mongoBD using mongoose, define the roots for various functions and starts the server.

-**package.json:** It lists project dependencies and scripts for managing the application.

-**Schema.js:** This file provides schema for various collections in a MongoDB database using Mongoose, an Object Data Modeling (ODM) library for MongoDB and Node.js.

- **Database:**

The database schema is designed to accommodate the following entities and relationships:

1. Users - Attributes: [list attributes like_id,username,password,email,usertype,approval,createdAt,updatedAt]

2. Admin - Attributes: [list attributes like _id, categories, promotedRestaurents, createdAt, updatedAt]

3. Restaurant - Attributes: [list attributes like _id, ownerId, title, address, mainImg, menu(refrences Fooditem),

createdAt, updatedAt]

4.FoodItem

- Attributes: [list attributes like

_id, title, descripition, itemIg, category, MenuCategory,

restaurantId, price, discount, rating , createdAt, updatedAt]

5.Orders - Attributes: [list attributes like _id, userId, name, email, mobile, address, pincode, restaurantId,

restaurantName, foodItemId, foodItemName, FoodItemImg, quantity, price,discount,

paymentMethod, orderdate, orderStatus, createdAt, updatedAt]

6.Cart - Attributes: [list attributes like _id, userId, restaurentId, restaurantName, foodItemName,

foodItemName, quantity, price, discount, createdAt, updatedAt].

--It interacts with MongoDB using mongoose.connect with the database

4. Setup Instructions

- **Prerequisites:**

  Server Dependencies

1. bcrypt
2. body-parser
3. cors
4. express
5. mongoose

   Client Dependencies

6. @testing-library/jest-dom
7. @testing-library/react
8. @testing-library/user-event
9. axios
10. bootstrap
11. react
12. react-dom
13. react-icons
14. react-router-dom
15. react-scripts
16. web-vitals

- **Installation:**

  Cloning the project:

  -git clone https://github.com/KishAsta/FOOD-ORDERING

  Setting up the server:

  -Navigate to server directory: cd server

  -Install server dependencies: npm install

Setting up the client:

-navigate to server directory: cd server

-Install client dependencies: npm install

5. Folder Structure

- **Client:**

1. **node_modules/:**

    - This directory contains all the npm packages and dependencies required for the project. It's usually generated automatically when dependencies are installed.

2. **public/:**

    - This directory holds static assets and the main HTML file (index.html) that serves as the entry point for the React application.
    - favicon.ico: The icon displayed in the browser tab or bookmark bar.
    - index.html: The main HTML file that includes a element ( ) where React renders the application.
    - manifest.json: Provides metadata used when the web app is installed on a user's mobile device or desktop.
    - robots.txt: Specifies which parts of the site should not be crawled or indexed by search engines.

3. **src/:**

    - This is the heart of the React application, containing all the source code.
    - components/: Directory for reusable UI components. Components are typically organized based on functionality or domain.
    - context/: Contains React Contexts, used for managing global state and providing data throughout the component tree.
    - pages/: Each major route or view of the application can have its own folder (admin/, customer/, restaurant/) with components specific to that view.
    - Authentication.jsx: Component responsible for handling user authentication and authorization.
    - Home.jsx: The main landing or home page component.
    - styles/: Directory for CSS or SCSS files related to styling components and pages.
    - App.js: The root component of the application that defines the overall structure and routes.
    - index.css: Global styles that apply to the entire application.

- index.js: The entry point where React is initialized and the application is rendered into the DOM.
- logo.svg: Logo or icon used throughout the application.
- reportWebVitals.js: Utility for reporting performance metrics.
- setupTests.js: Configuration file for setting up testing utilities and frameworks.
- .gitignore: Specifies files and directories that Git should ignore.

- **Server:**

1. **node_modules/:**
    - Similar to the client-side structure, this directory holds all the npm packages and dependencies required for the server-side application to run.
2. **index.js:**
    - This file serves as the entry point for your Node.js application. It typically initializes the server, sets up routes, and connects to databases or other services.
3. **package-lock.json:**
    - This file is automatically generated by npm. It locks down the versions of dependencies that were installed, ensuring consistency across different environments.
4. **package.json:**
    - Contains metadata about the project and configuration details. It also lists all dependencies required by the project, scripts to run, and other metadata.
5. **Schema.js:**
    - Presumably contains Mongoose schemas or other data structure definitions used for interacting with your MongoDB database.

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
    - **Frontend:** `npm start` in the client directory.
    - **Backend:** `npm start` in the server directory.

7. API Documentation

1. **User Authentication**
    - POST /api/user/register - Registers a new user.
    - POST /api/user/login - Authenticates a user and returns a token.
2. **User Management**
    - GET /api/user/- Retrieves user information by ID.
    - PUT /api/user/- Updates user information by ID.
3. **Restaurant Management**
    - POST /api/update-promote-list - Updates the promoted restaurant list.
    - POST /api/approve-user - Approves a restaurant user.
    - POST /api/reject-user - Rejects a restaurant user.
    - GET /api/fetch-restaurants - Fetches all restaurants.
    - GET /api/fetch- restaurant-details/- Fetches restaurant details by owner ID.
    - GET /api/fetch-restaurant/- Fetches restaurant details by restaurant ID.
4. **Order Management**

- GET /api/fetch-orders - Fetches all orders.
- PUT /api/cancel-order - Cancels an order.
- PUT /api/update-order-status - Updates the status of an order.
- POST /api/place-cart-order - Places an order from the cart.

5. **Food Item Management**
   - GET /api/fetch-items - Fetches all food items.
   - GET /api/fetch-item-details/- Fetches details of a food item by ID.
   - POST /api/add-new-product - Adds a new product (food item).
   - PUT /api/update-product/- Updates a product (food item) by ID.
6. **Cart Management**
   - GET /api/fetch-cart - Fetches all cart items.
   - POST /api/add-to-cart - Adds an item to the cart.
   - PUT /api/remove-item - Removes an item from the cart.
7. **Administrative Management**
   - GET /api/fetch-categories - Fetches all categories.
   - GET /api/fetch-promoted-list - Fetches the promoted restaurant list.
   - User Details GET /api/fetch-user-details/- Fetches details of a user by ID.
   - GET /api/fetch-users - Fetches all users.

8. Authentication

1. **JWT Tokens (JSON Web Tokens):**

- Client Side: When a user logs in with valid credentials, the server generates a JWT token and sends it back to the client.
- Server Side: The server signs the token using a secret key and includes user information (like user ID and possibly roles) in the payload.
- Storage: Typically stored in local storage or session storage in the browser to persist authentication state across sessions.

2. **Token Expiry:**

- JWT tokens have an expiration time (e.g., 1 hour). Clients can refresh tokens using a refresh token if supported.
- Refresh tokens are stored securely on the client side and are exchanged with the server to obtain a new JWT token without requiring the user to re-enter credentials.

3. **Role-based Access Control (RBAC):**

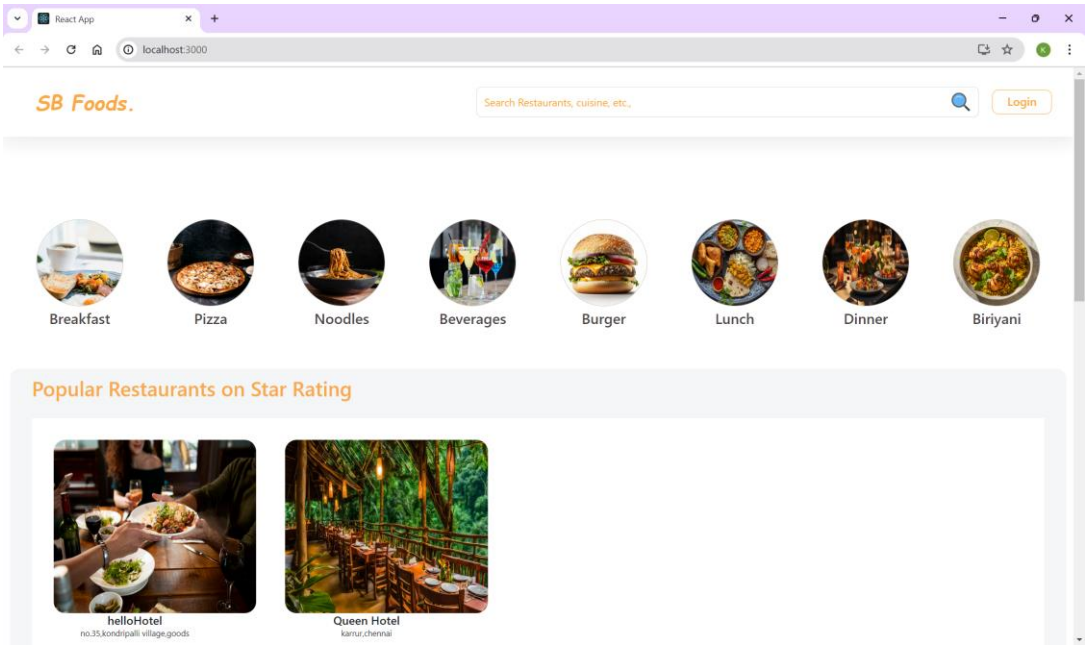
- Users are assigned roles (e.g., admin, customer, restaurant owner) that dictate their permissions.
- Access to certain routes or functionalities is restricted based on these roles.

4. **Middleware:**

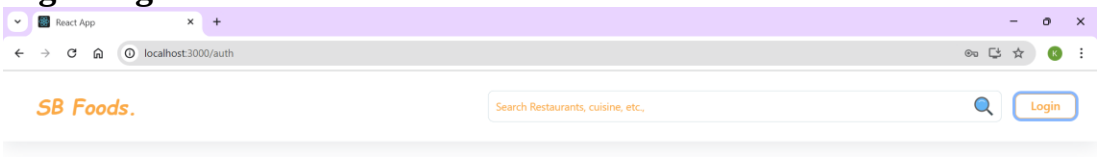
- Express middleware is often used to protect routes that require authentication.
- Before allowing access to protected resources, middleware verifies the validity of the JWT token.
- If the token is valid, the server extracts user information from the token's payload and grants access based on the user's role and permissions.

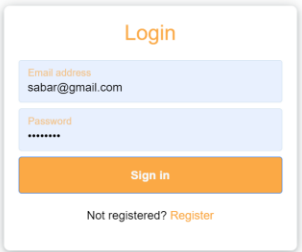
9. User Interface

- **Home**

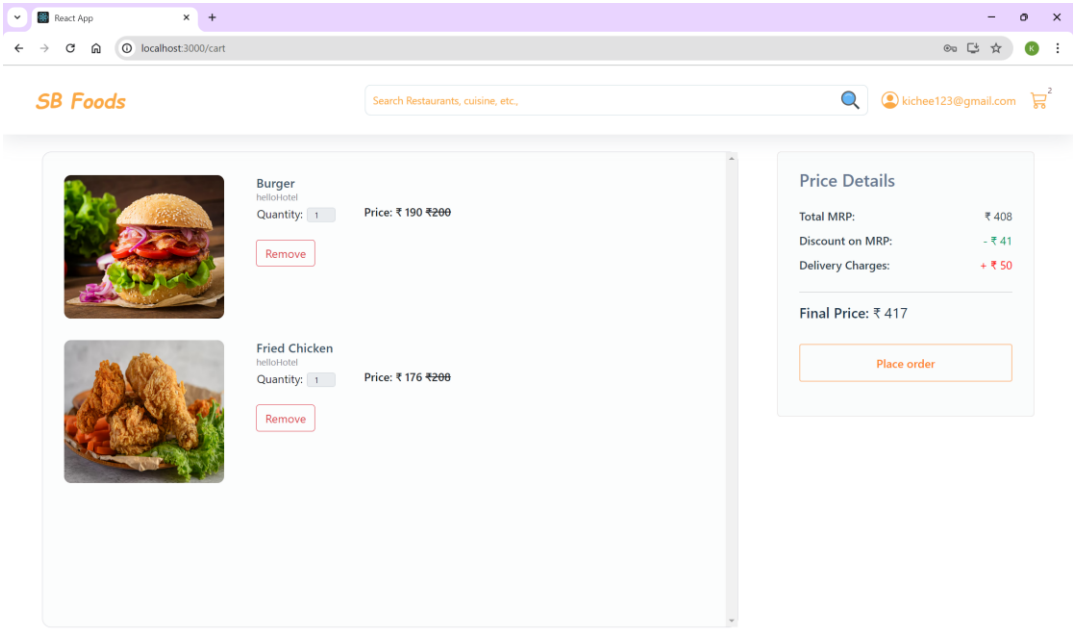


- **Login Page**

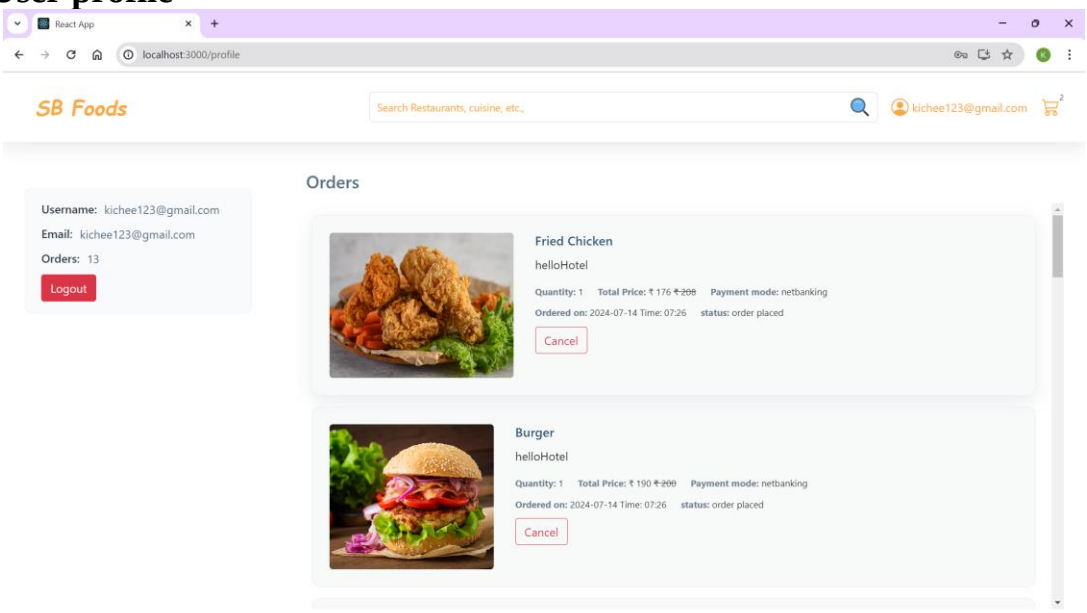




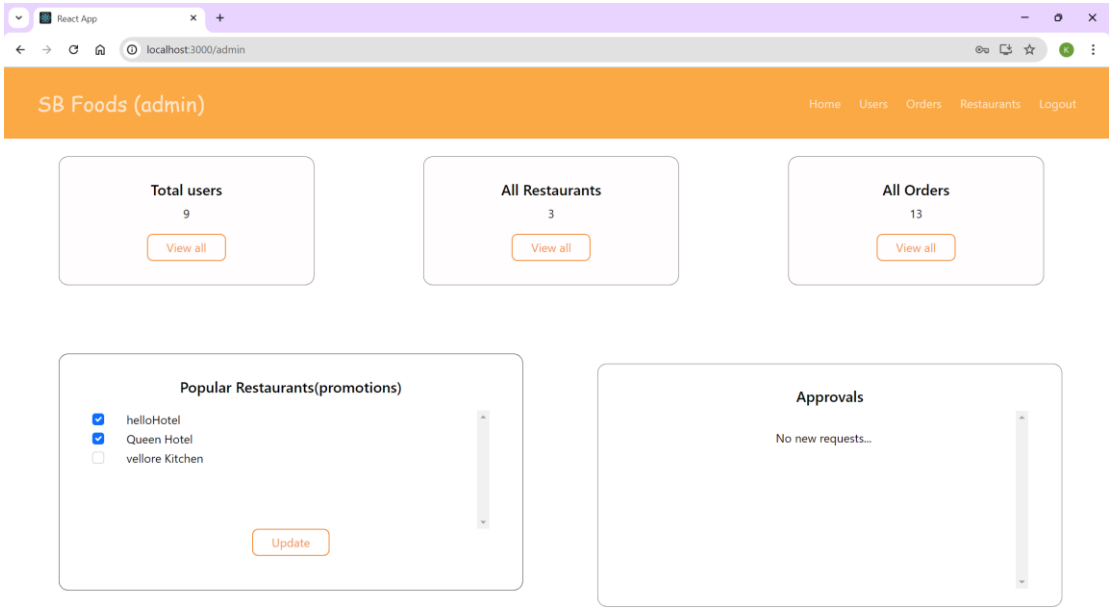
- **Cart**



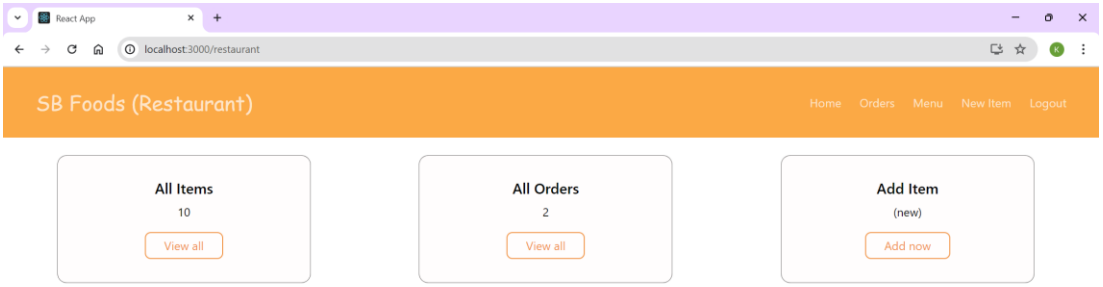
- **User profile**



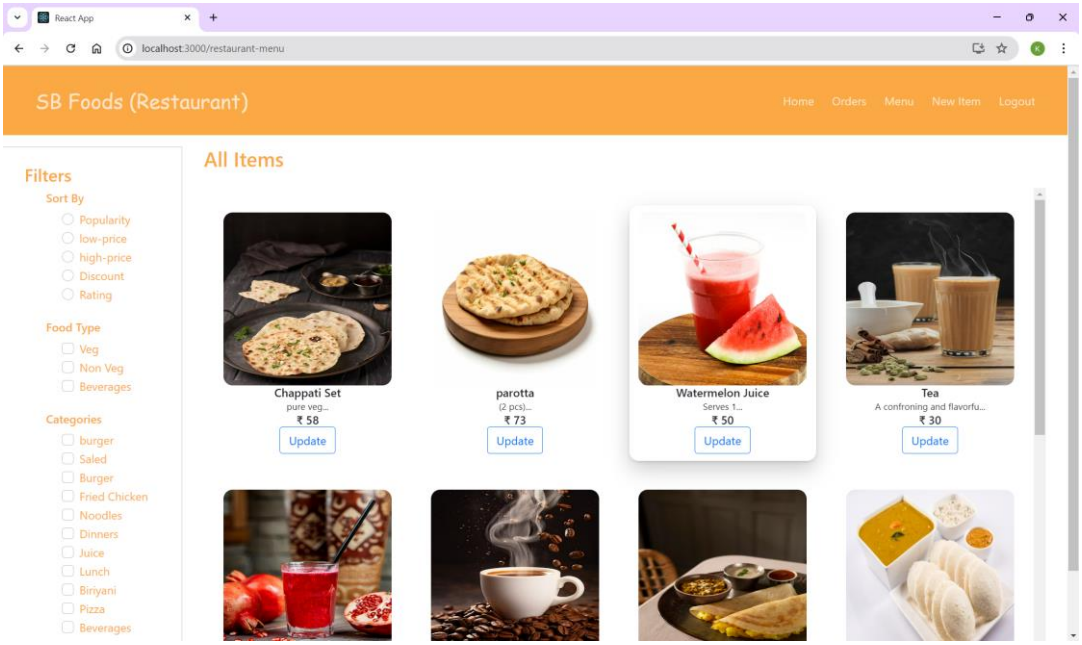
- **Admin UI**



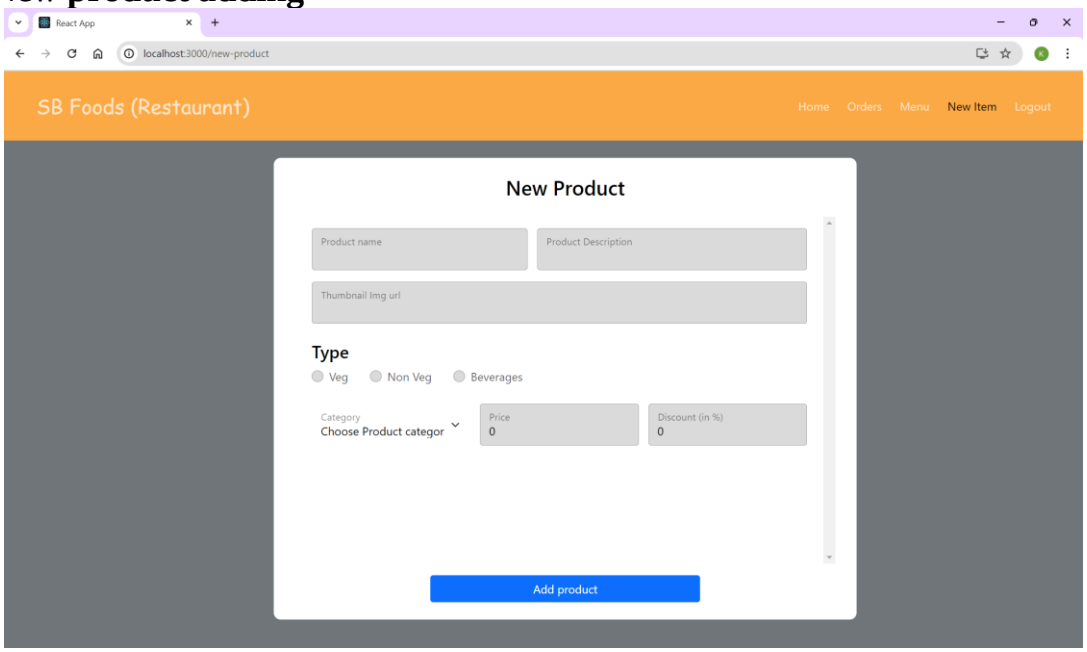
- **Restaurant UI**



- **Restaurant Menu**



- **New product adding**



10. Testing

Unit Testing:

Backend:

Jest: A testing framework for JavaScript applications, ideal for unit testing functions and modules.

Frontend:

Jest: Included by default with Create React App for unit testing React components.

React Testing Library: For testing React components with a focus on testing user interactions and component behavior.

Integration Testing:

Backend:

Supertest: Used in combination with Jest to test API endpoints and ensure they function correctly when integrated with each other.

Frontend:

React Testing Library: Also used for integration tests, verifying the interaction between different components.

API Testing:

Backend:

Postman: For manual and automated API testing. Postman allows you to send requests to your API endpoints and validate responses. You can also create test scripts in Postman to automate and validate API responses.

11. Screenshots or Demo

https://drive.google.com/file/d/1NHIU5ZZaOIiepHzC4L87kI0Zlw0CFTaF/view?usp=sharing

12. Known Issues

1. **BG-001: Login button not responding**

   Description: Users cannot log in to their accounts, making the app unusable.

2. **BG-002: Error message not displayed for invalid login**

   Description: Users do not receive feedback for incorrect login attempts, leading to confusion.

3. **BG-003: Registration page not loading**

   Description: New users cannot register, preventing them from using the app.

4. **BG-004: Items not being added to cart**

   Description: Users cannot add items to the cart, affecting the core functionality of ordering food.

5. **BG-006: Order history not displaying past orders**

   Description: Users cannot see their past orders, affecting their ability to reorder.

6. **BG-010: Duplicate items in order history**

   Description: Duplicate entries appear in the order history, causing confusion for users.

7. **BG-011: Incorrect total price calculation**

   Description: The total price shown at checkout is incorrect, leading to potential overcharging.

8. **BG-012: Push notifications not received**

   Description: Users are not receiving push notifications for order updates.

13. Future Enhancements

1. Real-time Order Tracking

Description: Allow users to track their orders in real-time from preparation to delivery.

2. Advanced Search and Filters

Description: Implement advanced search options and filters (e.g., by cuisine, dietary restrictions, price range).

3. Loyalty Program

Description: Introduce a loyalty program where users can earn points on each purchase and redeem them for discounts or free items.

4. Social Media Integration

Description: Allow users to share their orders and reviews on social media platforms.

5. Voice Ordering

Description: Enable users to place orders using voice commands.

6. In-app Chat Support

Description: Introduce a live chat feature for users to get instant support.

7. AR Menu Preview

Description: Use augmented reality (AR) to allow users to visualize food items in 3D before ordering.

8. Subscription Plans

Description: Offer subscription plans for regular customers with benefits like free delivery, special discounts, etc.

9. Integration with Smart Home Devices

Description: Enable users to place orders through smart home devices like Amazon Echo or Google Home.

10. Sustainability Features

Description: Highlight restaurants with sustainable practices and allow users to choose eco-friendly packaging options.

11. Personalized Recommendations

Description: Use machine learning to provide personalized food and restaurant recommendations based on user preferences and order history.

12. Multi-language Support

Description: Add support for multiple languages to cater to a diverse user base.

13. Order Scheduling

Description: Enable users to schedule orders in advance for a specific date and time.