**SB FOODS-FOODS ORDERING APP BY MERN**

**1.Introduction**

Project Title**:** Food Ordering app using MERN

Team Members:

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | NAME | ROLE | RESPONSIBILITIES |
| 1 | Barkavi E | Full-Stack  Developer | Responsible for overall development,Including front-end, back-end, server-side logic, and database design |
| 2 | Balaji A | Frontend  Developer | Responsible for UI/UX design using React, Material UI, and Bootstrap. |
| 3 | Abinaya | Database  Administrator | Responsible for MongoDB setup and ensuring data integrity. |
| 4 | Aravind Raagavendhar S | Backend Developer | Responsible for Express.js setup and API development. |
| 5 | Abraham P | Backend Developer | Responsible for Express.js setup and API development. |

## **2.Project Overview**

**Purpose:**

A food-ordering app built using the MERN(MongoDB,Express.js, React.js, Node.js) stack, providing a comprehensive solution for ordering food online. With its user-friendly interface and powerful features, the app aims to simplify the food ordering process for users and enhance their overall experience. The application consists of a customer-facing app for ordering food and an admin app for managing orders, menu items, and more.

**Features:**

* User Authentication & Authorization
* Users (customers, restaurant owners, and admin) can register,

log in, and manage their profiles.

* Restaurant & Menu Management
* Restaurants can add, edit, or delete menu items, and manage restaurant details (address, operating hours).
* User-Friendly Food Browsing & Search
* Users can browse restaurants, filter by cuisine, and search for specific dishes.
* Cart & Order Management
* Users can add items to the cart, customize orders, and proceed to checkout.
* Stripe Payment Integration
* Secure and reliable payment processing using Stripe.

**Architecture**

Frontend:

* The frontend is built with React.js, providing a component-based architecture that allows for reusable UI elements.
* This template provides a minimal setup to get React working in Vite with HMR and some ESL int rules.
* Next.js/Nuxt.js used for server-side rendering and improved SEO.
* Uses Axios or Fetch API to interact with backend services (fetch menu data, process orders).

**Backend:**

* The backend tech stack uses Node.js and Express.js a web application framework for handling HTTP requests and building APIs.
* The MongoDB database is used to store data like menu items, user information, and orders.
* Mongoose ODM (Object Data Modeling) library for MongoDB, making it easier to work with MongoDB data in Node.js.
* The backend architecture involves Middleware for authentication,logging and validation

3. Database

MongoDB, a NoSQL database, to store and manage the data. MongoDB is chosen for its flexibility and scalability.

It is used to define the application's schemas and models. the main collections could be:

* users: Stores user information (e.g., name, email, address).
* menuItems: Stores details of food items (e.g., name, price, description).
* orders: Stores order information (e.g., user, items, status).
* carts: Stores temporary cart data for users.

4. Setup Instructions

Prerequisites:

* **Node.js and npm:** Install Node.js, which includes npm (Node Package Manager. Node.js is required to run JavaScript on the server side.
* MongoDB: Ensure that you have a running instance of MongoDB. You can use either a local MongoDB setup or a cloud-based solution like MongoDB Atlas.
* **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:

1. .Clone the repository:

git clone https://github.com/yourusername/Food-Ordering-app.git

1. Install Dependencies:

* Install all required Node.js packages for the app

npm install

1. start the development server

* To start the development server,

npm run dev or npm run start

1. Configure Environment Variables

* Create a .env file in the root directory.

MONGO\_URI=mongodb://localhost:27017/sb\_food Or your MongoDB Atlas URI

PORT=5000 Or any port you want the app to run on

JWT\_SECRET=your\_jwt\_secret For handling secure login sessions

1. Start the Backend Server: Run the following command in the server directory:

npm start

1. Start the Frontend Server: Run the following command in the client directory:

npm start

5. Folder Structure

Client:

* src/components/: Reusable UI components, like NavBar, Footer, MenuItemCard, and OrderForm.
* src/pages/: Main pages of the application, like Home, Menu, Order, and Profile.
* src/services/: Contains helper functions for making API calls, such as api.js or individual files for handling user, order, and menu requests.
* env: Environment variables specific to the client (like API URL for backend server).
* package.json: Lists dependencies and scripts for building and running the React app.

Server:

* config/db.js: Handles MongoDB connection setup using Mongoose.
* controllers/: Contains functions for handling the business logic of each route.
* models/: Contains Mongoose schemas and models for MongoDB collections (User.js,Order.jsMenuItem.js)
* routes/: Defines Express routes for different resources(orderRoutes.js,userRoutes.js,menuRoutes.js)
* middleware/: Custom middleware functions for the backend.
* package.json: Lists dependencies and scripts for running the backend.

6.Running the Application

1. Frontend: To start the frontend server, run the following command in the client folder:

npm start

1. Backend: To start the backend server, run the following command in the server folder:

npm run

7. API Documentation

1. POST /api/user/register: Register a new user (either.

* Request Body:

{

"username": "string",

"email": "string",

"password": "string"

}

* Response:

{

"message": "User registered successfully",

"user": {

"id": "string",

"username": "string",

"email": "string"

},

"token": "string"

}

2. POST /api/users/login: Authenticate a user and returns a JWT token.

* Request Body:

{

"email": "string",

"password": "string"

}

* Response:

{

"message": "Login successful",

"user": {

"id": "string",

"username": "string",

"email": "string"

},

"token": "string"

}

3. GET /api/menu: Retrieves all available menu items

* Response:

[

{

"id": "string",

"name": "string",

"description": "string",

"price": "number",

"category": "string",

"imageUrl": "string"

},

...

]

1. POST /api/menu: Adds a new menu item

* Request Body:

{

"name": "string",

"description": "string",

"price": "number",

"category": "string",

"imageUrl": "string"

}

* Response:

{

"message": "Menu item created",

"menuItem": {

"id": "string",

"name": "string",

"description": "string",

"price": "number",

"category": "string",

"imageUrl": "string"

}

}

4.PUT/api/menu/:id: Updates details of an existing menu item

* Request Body:

{

"name": "string",

"description": "string",

"price": "number",

"category": "string",

"imageUrl": "string"

}

* Response:

{

"message": "Menu item updated successfully",

"menuItem": {

"id": "string",

"name": "string",

"description": "string",

"price": "number",

"category": "string",

"imageUrl": "string"

}

}

5.POST/api/orders: places a new order for authenticated data

* Request Body:

{

"items": [

{

"menuItemId": "string",

"quantity": "number"

}

],

"totalAmount": "number",

"deliveryAddress": "string"

}

Response:

{

"message": "Order placed successfully",

"order": {

"id": "string",

"userId": "string",

"items": [

{

"menuItemId": "string",

"quantity": "number"

}

],

"totalAmount": "number",

"status": "Pending",

"deliveryAddress": "string"

}

}

6.PUT /api/orders/:id Updates the status of an order (e.g., from Pending to Delivered).

* Request Body:

{

"status": "string" // e.g., "Delivered", "Canceled"

}

* Response:

{

"message": "Order status updated",

"order": {

"id": "string",

"status": "string"

}

}

Authentication

Authentication :

In this guide, we will set up a user authentication system using MongoDB, Node.js, and JWT (JSON Web Token) for a food ordering application (SB Foods). Authentication will allow users to securely register, log in, and access protected features (e.g., placing orders, viewing order history).

Authorization :

In addition to authentication, your SB Foods app needs an authorization layer to control access to resources and features based on user roles (e.g., admin, customer). Authorization ensures that users can only access the data and perform actions that they are allowed to.

* Authentication: Verifies who the user is (e.g., login).
* Authorization: Determines what the authenticated user can do (e.g., view orders, manage items)

User Interface

The user interface (UI) design will be a front-end system that communicates with a MongoDB backend. Typically, the UI can be built using technologies like React (for web applications) or React Native (for mobile applications). MongoDB will serve as the database to store user, menu, and order information. The user interface design provides a smooth experience for customers and admins.

UI Components Overview:

1. Landing Page

The landing page provides an overview of the restaurant and options to navigate.

2. Login Page

The login page allows users to enter their email and password.

3. Register Page

Allows users to create an account.

4. Menu Page

Displays a list of food items fetched from MongoDB.

6. Admin Dashboard

Allows the admin to manage the menu and view order.

Testing

Testing involves both frontend and backend aspects, and given the use of MongoDB, it's important to validate database operations as well. Here's a breakdown of the testing strategy and tools commonly used:

Testing Tools & Libraries

* Frontend Testing:

Jest and React Testing Library for unit and integration testing of React components.

Cypress for end-to-end testing, simulating user actions.

* Backend Testing:

Jest and Supertest for testing API endpoints.

MongoDB In-Memory Server for mock database testing (to avoid impacting real data).

* Database Testing:

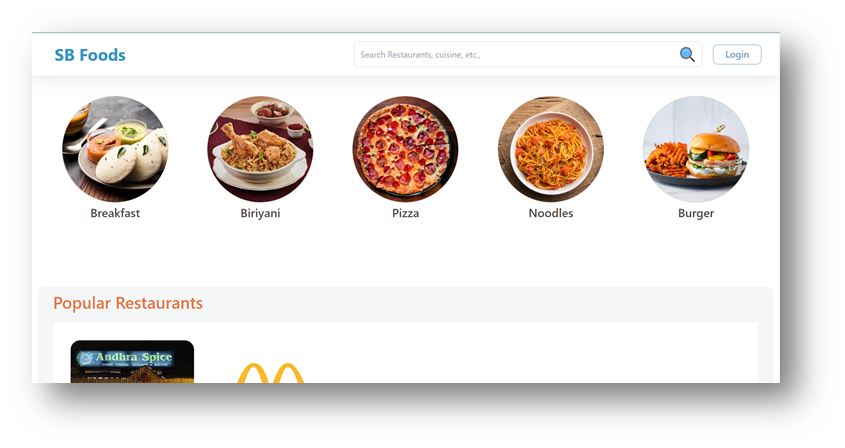
MongoDB In-Memory Server or Mock MongoDB for testing CRUD operations without affecting the live database.

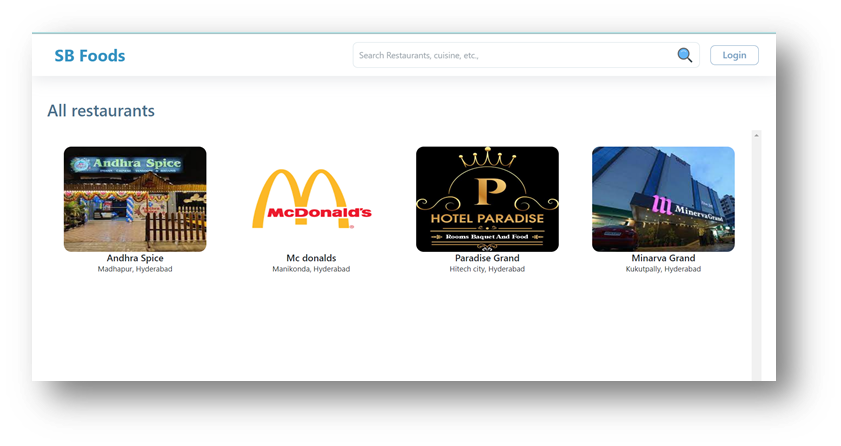
* End-to-End (E2E) Testing:

Testing the entire user flow, from logging in to placing an order.

Screenshots

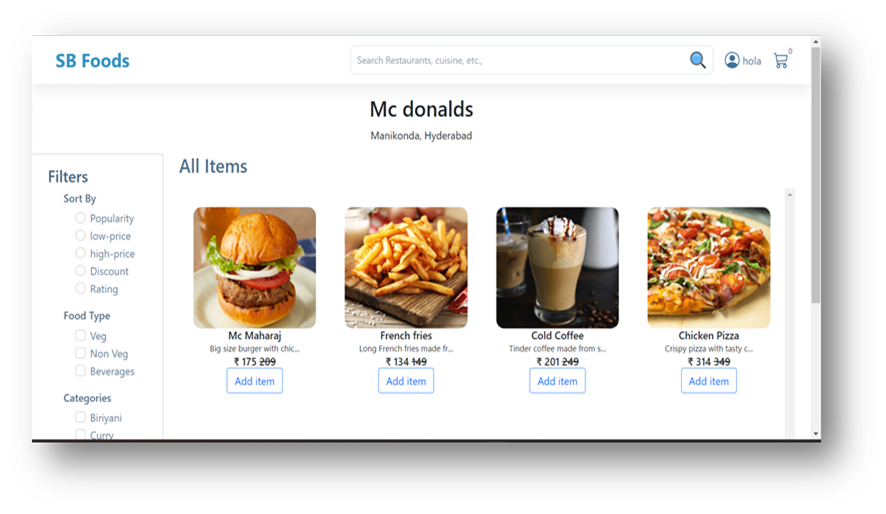
Landing page:

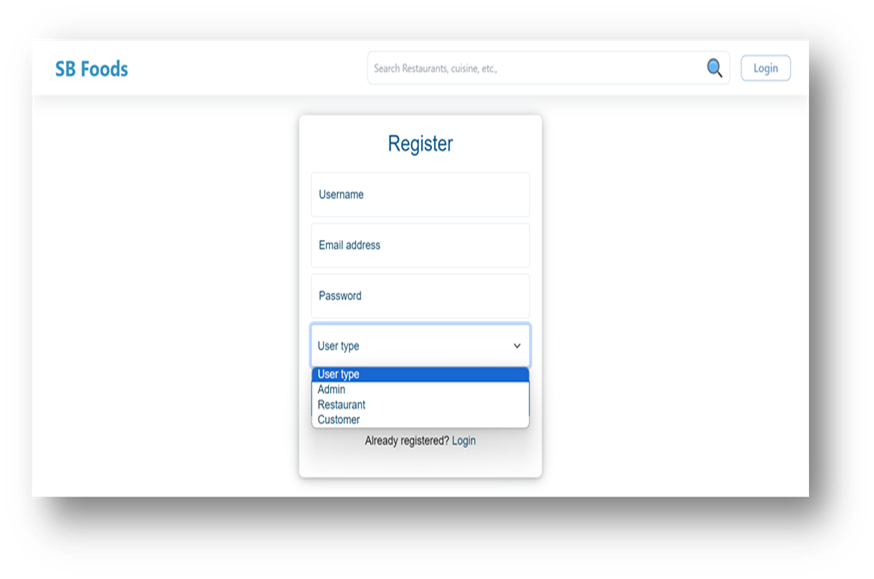


Restaurants:

Restaurant

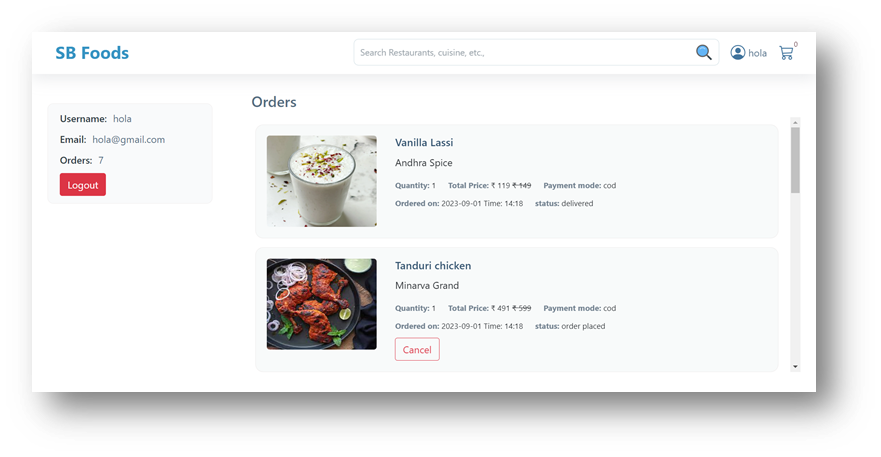
Menu:

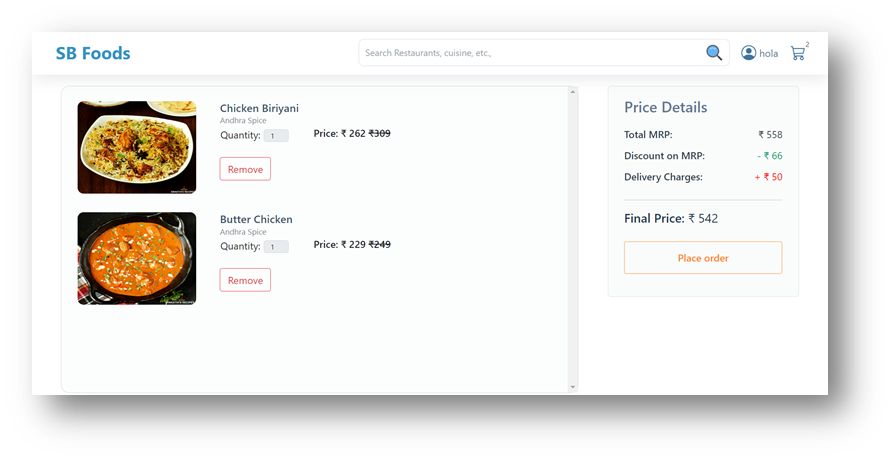


Authentication:

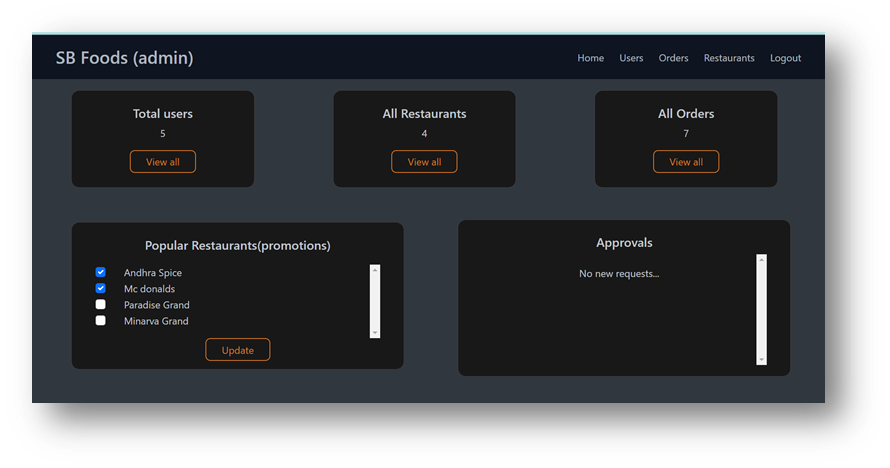
User

profile:

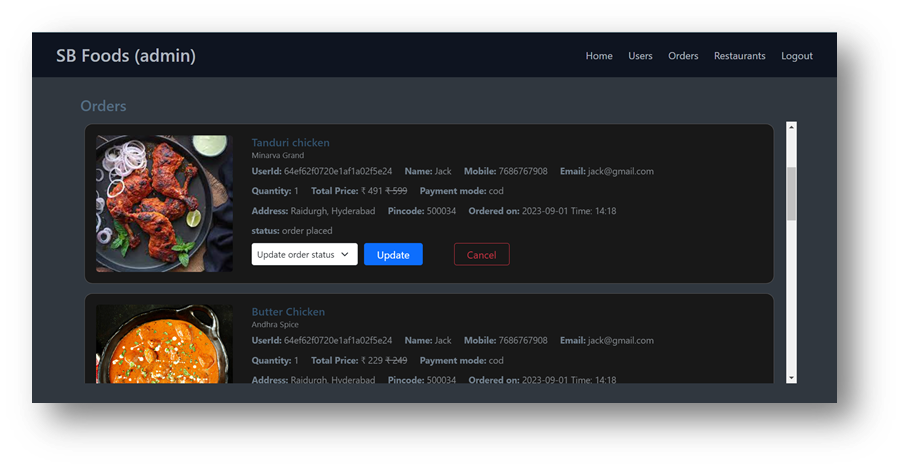


Cart:

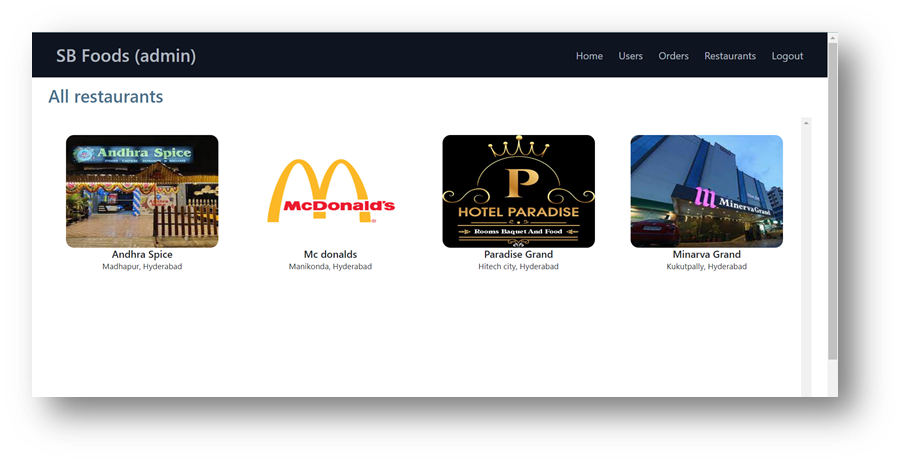
Admin

dashboard:

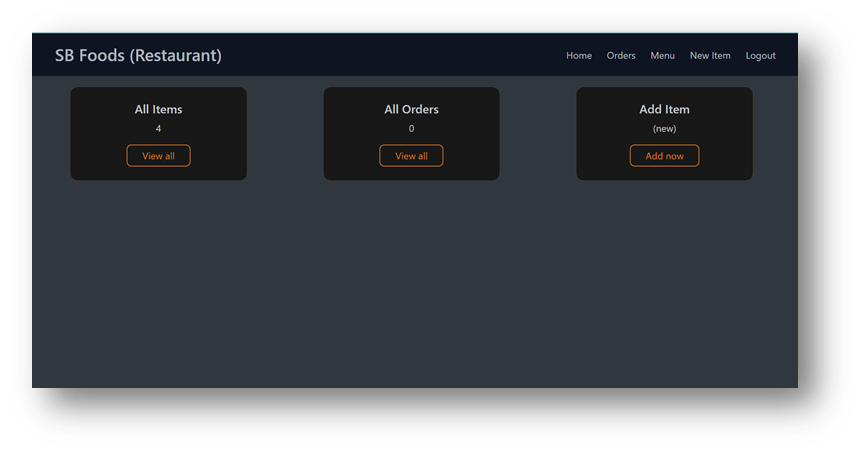
All

orders:

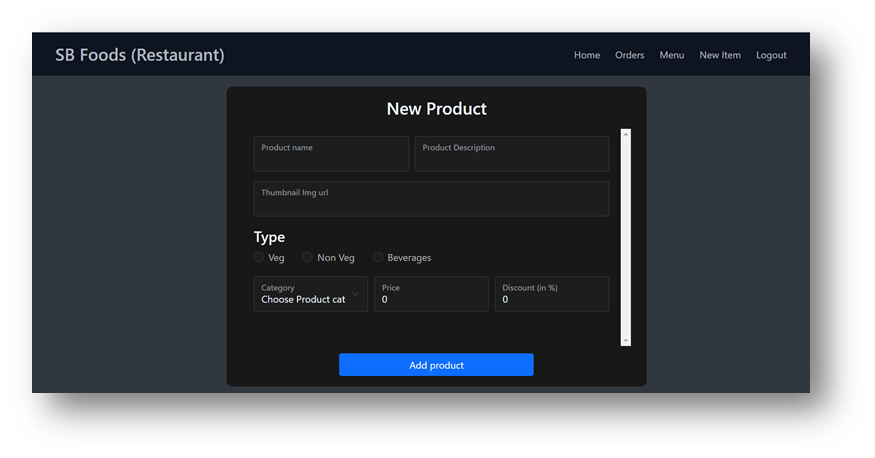
All

restaurants:

Restaurant

Dashboard:

New

item:

Known Issues

While most of the application is fully functional, there are a few issues that developers or users should be aware of:

Performance Issues with Large Datasets

* As the number of users, menu items, and orders increases, MongoDB queries can become slower, especially if indexes are not properly configured.

Security Vulnerabilities:

* The application might be vulnerable to attacks such as NoSQL injection, weak password storage, and insecure API endpoints.

Payment Gateway Integration Issues:

* Payment failures or discrepancies in order status updates.

High Disk Usage:

* The application may consume a lot of disk space due to large collections or lack of cleanup for old data.

Workarounds:

* Users can't track their orders properly or the status updates are delayed.
* Issues with Push Notifications,Users are not receiving timely updates or are overwhelmed with notifications.

Future Enhancements

There are several potential improvements and features that could be added to the Food ordering app with MERN application:

* Enhanced User Experience

Personalized Recommendations: Implement machine learning models (e.g., TensorFlow.js) for real-time suggestions.

* Improved Payment System

Cryptocurrency Payments: Add support for cryptocurrency payments using APIS like Coinbase Commerce for users who prefer crypto transactions.

* Integrations with Third-Party Services

Integration with Food Delivery Services: Integrate with third-party delivery platforms (e.g., Uber Eats, DoorDash) for additional delivery options during peak times.

* Dynamic Pricing and Discounts

Feature: Offer dynamic pricing, time-based discounts, or personalized promotions based on user behavior and demand.