Full Stack Development with MERN

Project Name

OrderOnTheGo
Your On-Demand Food Ordering Solution

Submitted by

Team Id : LTVIP2025TMID59338
Team Members
Ojeswari Devi kilari - Frontend Developer
Abhinaya Kalisetty - Backend Developer
Harini Kanna - UI/UX Designer

1.INTRODUCTION

Introducing CraveKart, the cutting-edge digital platform poised to revolutionize the way you order food online. With CraveKart, your food ordering experience will reach unparalleled levels of convenience and efficiency.

Our user-friendly web app empowers foodies to effortlessly explore, discover, and order dishes tailored to their unique tastes. Whether you're a seasoned food enthusiast or an occasional diner, finding the perfect meals has never been more straightforward.

Imagine having comprehensive details about each dish at your fingertips. From dish descriptions and customer reviews to pricing and available promotions, you'll have all the information you need to make well-informed choices. No more second-guessing or uncertainty – CraveKart ensures that every aspect of your online food ordering journey is crystal clear.

The ordering process is a breeze. Just provide your name, delivery address, and preferred payment method, along with your desired dishes. Once you place your order, you'll receive an instant confirmation. No more waiting in long queues or dealing with complicated ordering processes – CraveKart streamlines it, making it quick and hassle-free.

1.1 SCENARIO:

Late-Night Craving Resolution

Meet Lisa, a college student burning the midnight oil to finish her assignment. As the clock strikes midnight, her stomach grumbles, reminding her that she skipped dinner. Lisa doesn't want to interrupt her workflow by cooking, nor does she have the energy to venture outside in search of food.

Solution with Food Ordering App:

1. Lisa opens the Food Ordering App on her smartphone and navigates to the late-night delivery section, where she finds a variety of eateries still open for orders.

2. She scrolls through the options, browsing menus and checking reviews until she spots her favorite local diner offering comfort food classics.

3. Lisa selects a hearty bowl of chicken noodle soup and a side of garlic bread, craving warmth and satisfaction in each bite.

4. With a few taps, she adds the items to her cart, specifies her delivery address, and chooses her preferred payment method.

5. Lisa double-checks her order details on the confirmation page, ensuring everything looks correct, before tapping the "Place Order" button.

6. Within minutes, she receives a notification confirming her order and estimated delivery time, allowing her to continue working with peace of mind.

7. As promised, the delivery arrives promptly at her doorstep, and Lisa eagerly digs into her piping hot meal, grateful for the convenience and comfort provided by the Food Ordering App during her late-night study session.

This scenario illustrates how a Food Ordering App caters to users' needs, even during unconventional hours, by offering a seamless and convenient solution for satisfying late-night cravings without compromising on quality or convenience.

2.PROJECT OVERVIEW

2.1 PURPOSE

The purpose of the OrderOnTheGo project is to build a user-friendly, full-stack online food ordering platform that connects customers with local restaurants. The goal is to streamline the process of discovering restaurants, placing food orders, and managing them efficiently. It aims to provide seamless experiences for customers, restaurant owners, and administrators, ensuring quick access to food, better order tracking, and efficient restaurant management.

2.2 FEATURES:

1. **Comprehensive Product Catalog:** CraveKart boasts an extensive catalog of food items from various restaurants, offering a diverse range of items and options for shoppers. You can effortlessly explore and discover various products, complete with detailed descriptions, customer reviews, pricing, and available discounts, to find the perfect food for your hunger.

2. **Order Details Page**: Upon clicking the "Place Order" button, you will be directed to an order details page. Here, you can provide relevant information such as your shipping address, preferred payment method, and any specific product requirements.

3. **Secure and Efficient Checkout Process:** CraveKart guarantees a secure and efficient checkout process. Your personal information will be handled with the utmost security, and we strive to make the purchasing process as swift and trouble-free as possible.

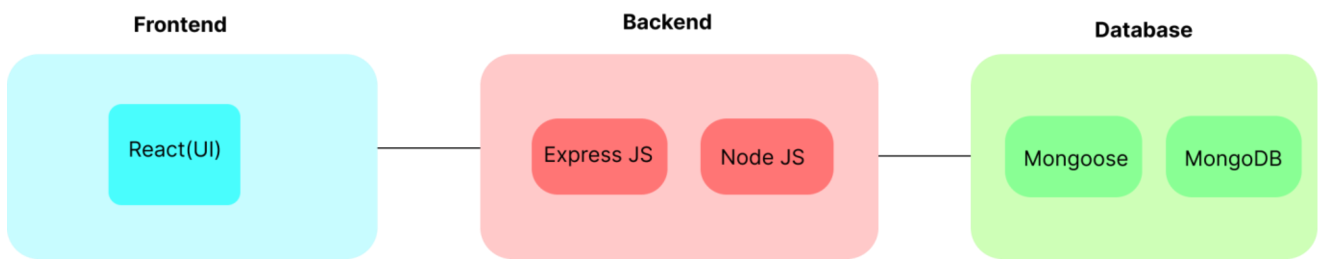
4. **Order Confirmation and Details:** After successfully placing an order, you will receive a confirmation notification. Subsequently, you will be directed to an order details page, where you can review all pertinent information about your order, including shipping details, payment method, and any specific product requests you specified.

In addition to these user-centric features, CraveKart provides a robust restaurant dashboard, offering restaurants an array of functionalities to efficiently manage their products and sales. With the restaurant dashboard, restaurants can add and oversee multiple product listings, view order history, monitor customer activity, and access order details for all purchases.

CraveKart is designed to elevate your online food ordering experience by providing a seamless and user-friendly way to discover your desired foods. With our efficient checkout process, comprehensive product catalog, and robust restaurant dashboard, we ensure a convenient and enjoyable online shopping experience for both shoppers and restaurants alike.

3. ARCHITECTURE

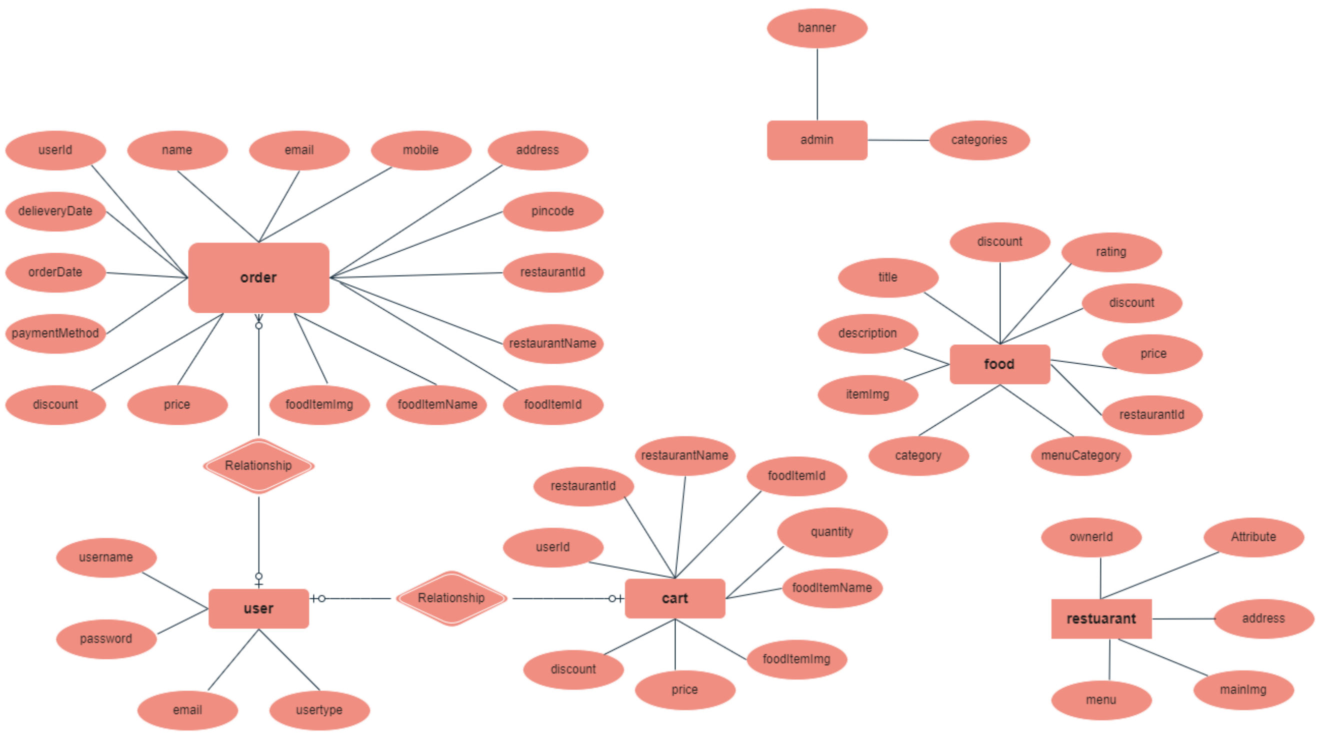
3.1 TECHNICAL ARCHITECTURE:

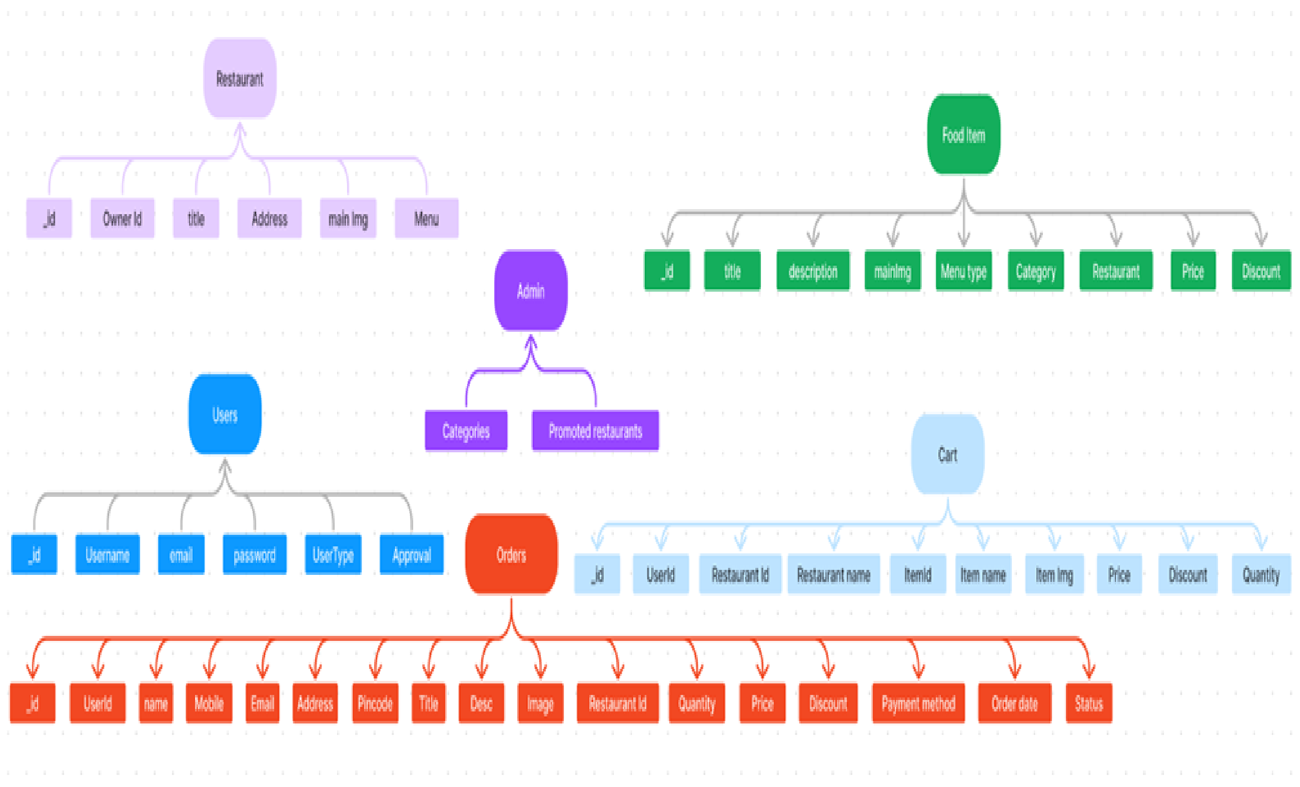


In this architecture diagram:

- The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Cart, Products, Profile, Admin dashboard, etc.,
- The backend is represented by the "Backend" section, consisting of API endpoints for Users, Orders, Products, etc., It also includes Admin Authentication and an Admin Dashboard.
- The Database section represents the database that stores collections for Users, Admin, Cart, Orders, and products.

3.2 ER DIAGRAM





The CraveKart ER-diagram represents the entities and relationships involved in an food ordering e-commerce 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.

4.SETUP INSTRUTIONS

4.1 PREREQUISITES:

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:

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/package-manager/

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/

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**

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/create-a-new-react-app.html

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.

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.

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

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

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

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

Link: https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/

To run the existing CraveKart App project downloaded from github:

Follow below steps:

Clone the repository:

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

Git clone: https://github.com/KalisettyAbhi234/OrderOnTheGo-Your-On-Demand-

Food-Ordering-Solution/tree/main

4.2 INSTALL DEPENDENCIES

- Navigate into the cloned repository directory:

  cd Food-Ordering-App-MERN

- Install the required dependencies by running the following command:

  npm install

Start the Development Server:

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

  npm run dev or npm run start

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

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.

You have successfully installed and set up the CraveKart app on your local machine. You can now proceed with further customization, development, and testing as needed.

5.FOLDER STRUCTURE

5.1 USER & ADMIN FLOW:

1. User Flow:

• Users start by registering for an account.

• After registration, they can log in with their credentials.

• Once logged in, they can check for the available products in the platform. • Users can add the products they wish to their carts and order.

• They can then proceed by entering address and payment details. • After ordering, they can check them in the profile section.

2. Restaurant Flow:

• Restaurants start by authenticating with their credentials.
• They need to get approval from the admin to start listing the products. • They can add/edit the food items.

3. Admin Flow:

• Admins start by logging in with their credentials.

• Once logged in, they are directed to the Admin Dashboard.

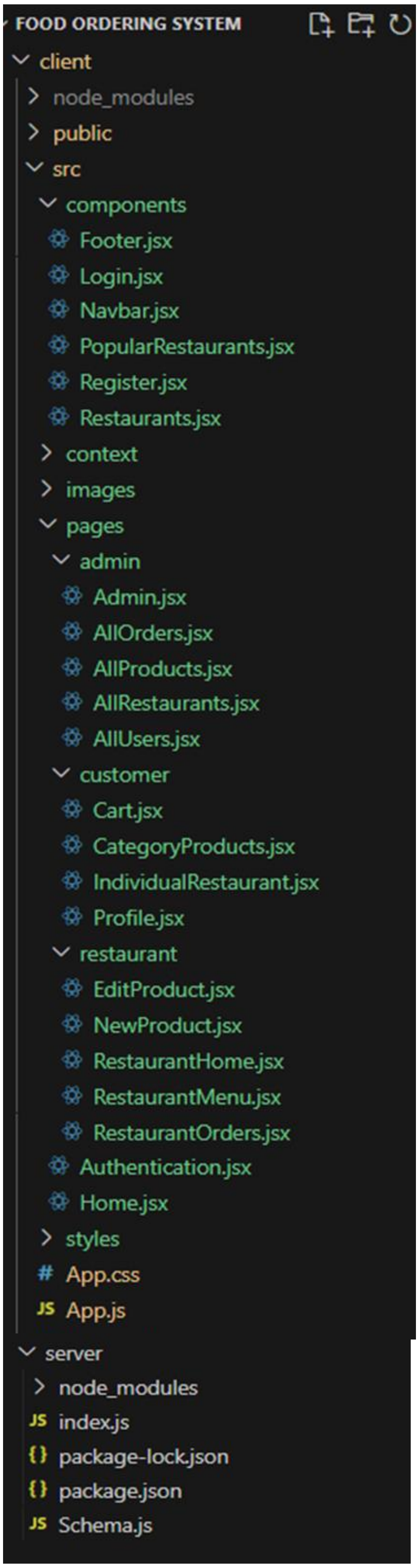
• Admins can access the users list, products, orders, etc.

5.2 PROJECT STRUCTURE

This structure assumes a React app and follows a modular approach. Here's a brief explanation of the main directories and files:

• src/components: Contains components related to the application such as, register, login, home, etc.,

• src/pages has the files for all the pages in the application



```
FOOD ORDERING SYSTEM
∨ client
  > node_modules
  > public
  ∨ src
    ∨ components
      Footer.jsx
      Login.jsx
      Navbar.jsx
      PopularRestaurants.jsx
      Register.jsx
      Restaurants.jsx
    > context
    > images
    ∨ 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
    App.css
    App.js
∨ server
  > node_modules
  index.js
  package-lock.json
  package.json
  Schema.js
```

.

6.RUNNING THE APPLICATION

6.1 PROJECT SETUP AND CONFIGURATION:

Install required tools and software:

• Node.js.

Reference Article: https://www.geeksforgeeks.org/installation-of-node-js-on-windows/

• Git.

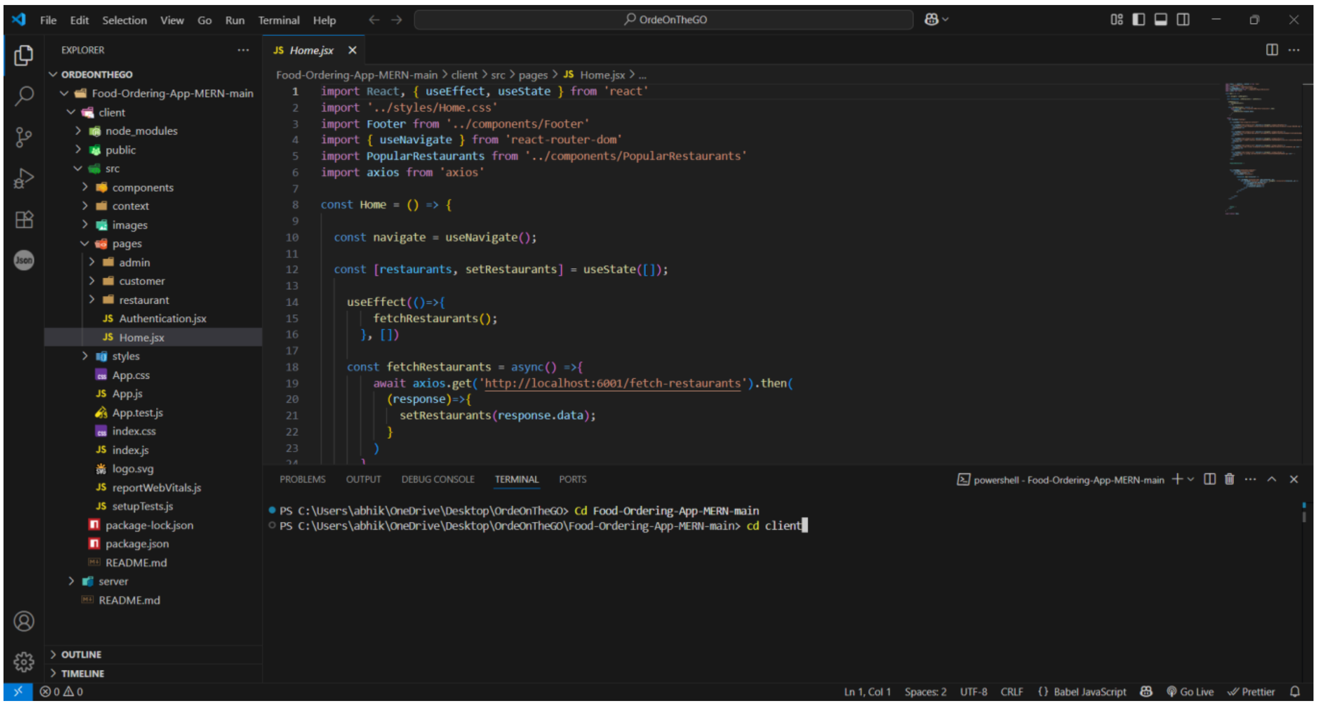
Reference Article: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Create project folders and files:

• Client folders.

• Server folders

Referral Image:



```
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);
          }
        )
```

```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO> Cd Food-Ordering-App-MERN-main
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main> cd client
```

6.2 DATABASE DEVELOPMENT

Create database in cloud video link:-
https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view

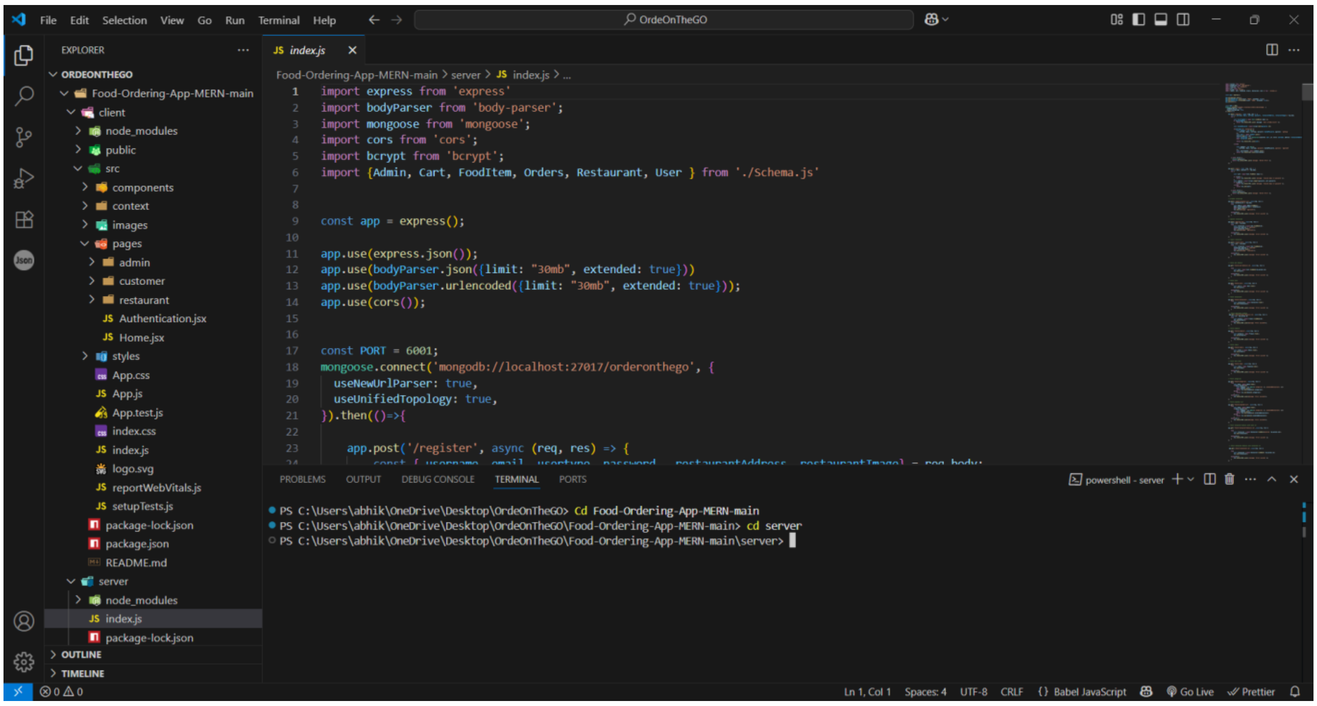
• Install Mongoose.

• Create database connection.

Reference Video of connect node with mongoDB database: https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

Reference Article: https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/

Reference Image:



```javascript
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://localhost:27017/orderonthego', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(()=>{

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

```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO> Cd Food-Ordering-App-MERN-main
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main> cd server
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main\server>
```

Schema use-case:

1. User Schema:

- Schema: userSchema
- Model: ‘User’
- The User schema represents the user data and includes fields such as username, email, and password.

2. Product Schema:

• Schema: productSchema

• Model: ‘Product’

• The Product schema represents the data of all the products in the platform.

• It is used to store information about the product details, which will later be useful for ordering.

3. Orders Schema:

• Schema: ordersSchema

• Model: ‘Orders’

• The Orders schema represents the orders data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,

4. Cart Schema:

• Schema: cartSchema

• Model: ‘Cart’

• The Cart schema represents the cart data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,

• The user Id field is a reference to the user who has the product in cart.

5. Admin Schema:

• Schema: adminSchema

• Model: ‘Admin’

• The admin schema has essential data such as categories, promoted restaurants, etc.,

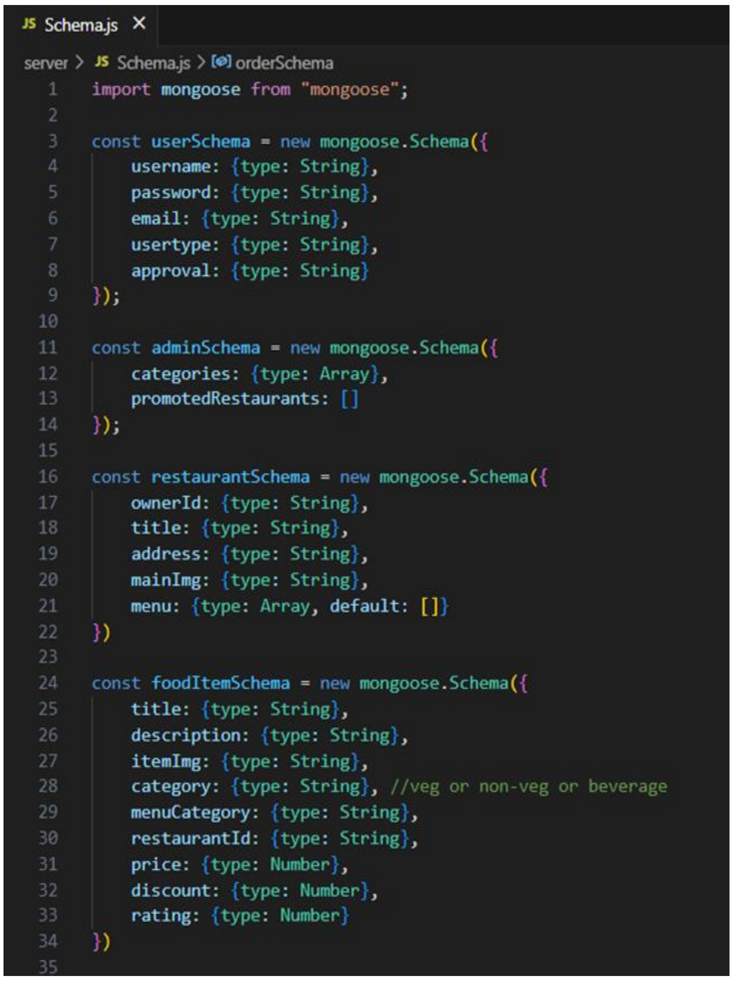
6. Restaurant Schema:

• Schema: restaurantSchema

• Model: ‘Restaurant’

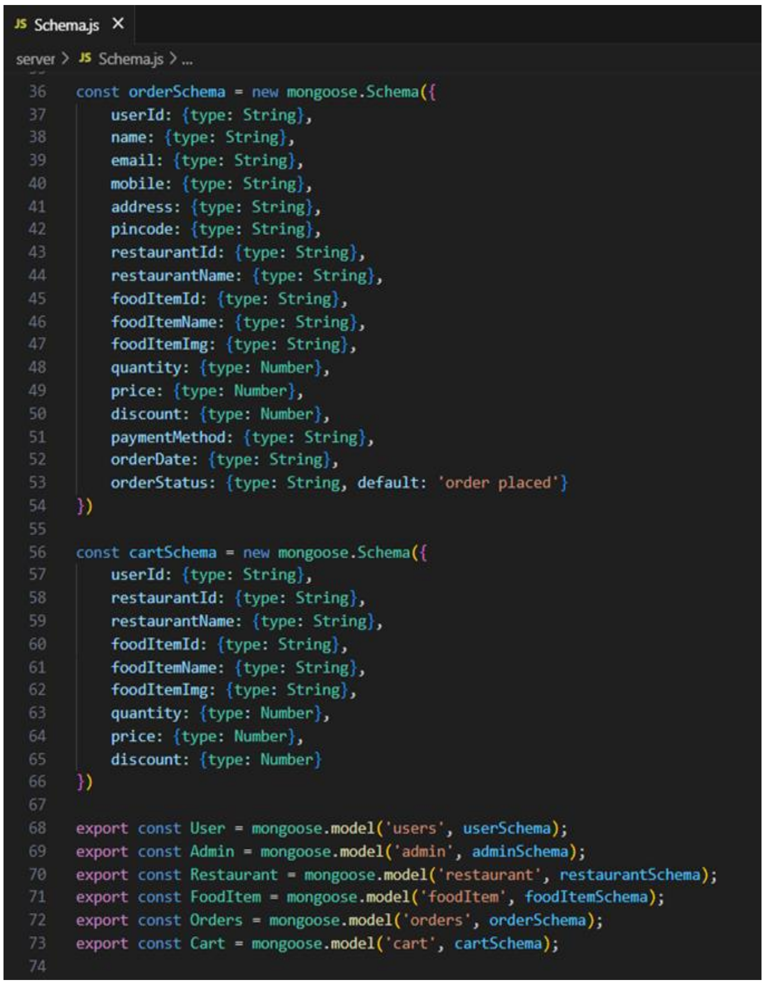
• The restaurant schema has the info about the restaurant and it’s menu

Schemas: Now let us define the required schemas


Schema.js

server > Schema.js > orderSchema

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    username: {type: String},
    password: {type: String},
    email: {type: String},
    usertype: {type: String},
    approval: {type: String}
});

const adminSchema = new mongoose.Schema({
    categories: {type: Array},
    promotedRestaurants: []
});

const restaurantSchema = new mongoose.Schema({
    ownerId: {type: String},
    title: {type: String},
    address: {type: String},
    mainImg: {type: String},
    menu: {type: Array, default: []}
})

const foodItemSchema = new mongoose.Schema({
    title: {type: String},
    description: {type: String},
    itemImg: {type: String},
    category: {type: String}, //veg or non-veg or beverage
    menuCategory: {type: String},
    restaurantId: {type: String},
    price: {type: Number},
    discount: {type: Number},
    rating: {type: Number}
})

```


Schema.js

server > Schema.js > ...

```
const orderSchema = new mongoose.Schema({
    userId: {type: String},
    name: {type: String},
    email: {type: String},
    mobile: {type: String},
    address: {type: String},
    pincode: {type: String},
    restaurantId: {type: String},
    restaurantName: {type: String},
    foodItemId: {type: String},
    foodItemName: {type: String},
    foodItemImg: {type: String},
    quantity: {type: Number},
    price: {type: Number},
    discount: {type: Number},
    paymentMethod: {type: String},
    orderDate: {type: String},
    orderStatus: {type: String, default: 'order placed'}
})

const cartSchema = new mongoose.Schema({
    userId: {type: String},
    restaurantId: {type: String},
    restaurantName: {type: String},
    foodItemId: {type: String},
    foodItemName: {type: String},
    foodItemImg: {type: String},
    quantity: {type: Number},
    price: {type: Number},
    discount: {type: Number}
})

export const User = mongoose.model('users', userSchema);
export const Admin = mongoose.model('admin', adminSchema);
export const Restaurant = mongoose.model('restaurant', restaurantSchema);
export const FoodItem = mongoose.model('foodItem', foodItemSchema);
export const Orders = mongoose.model('orders', orderSchema);
export const Cart = mongoose.model('cart', cartSchema);

```

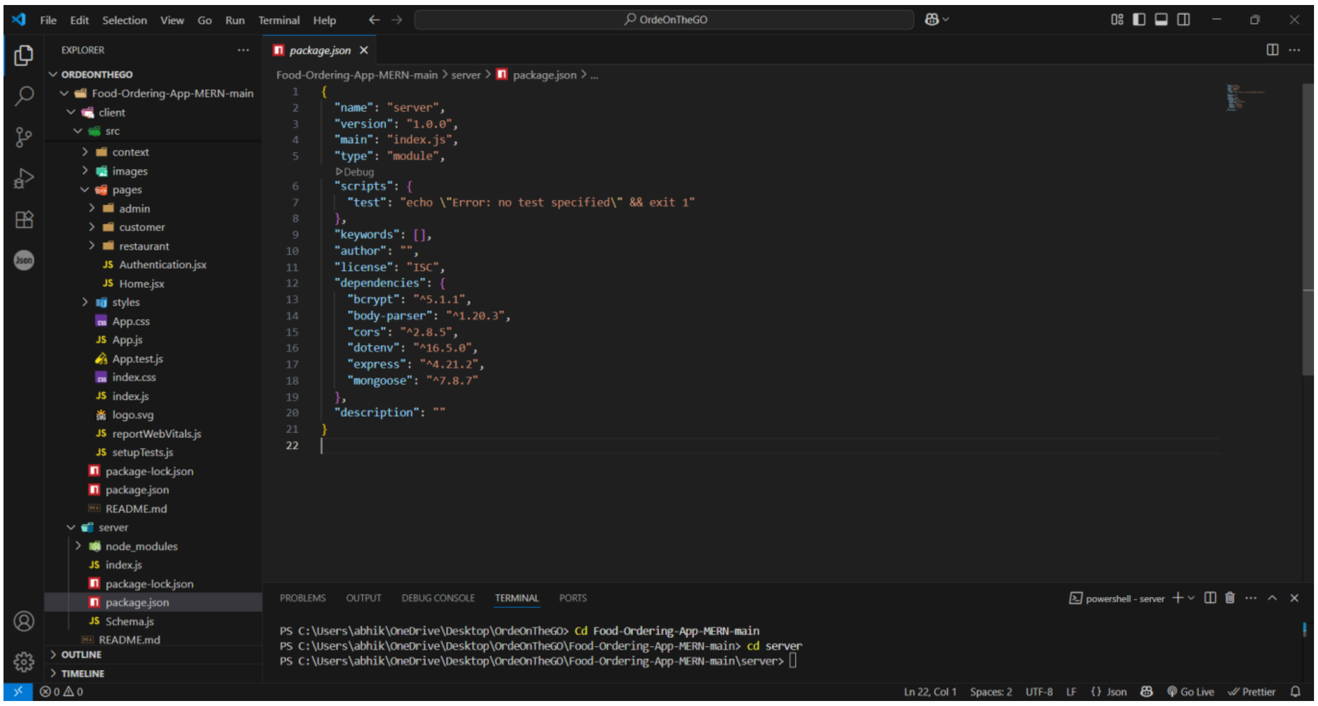
6.3 BACKEND DEVELOPMENT:

Set Up Project Structure:

- Create a new directory for your project and set up a package.json file using the npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

Reference Video: https://drive.google.com/file/d/19df7NU-gQK3DO6wr7ooAfJYIQwnemZoF/view?usp=sharing

Reference Image:



```json
{
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.3",
    "cors": "^2.8.5",
    "dotenv": "^16.5.0",
    "express": "^4.21.2",
    "mongoose": "^7.8.7"
  },
  "description": ""
}
```

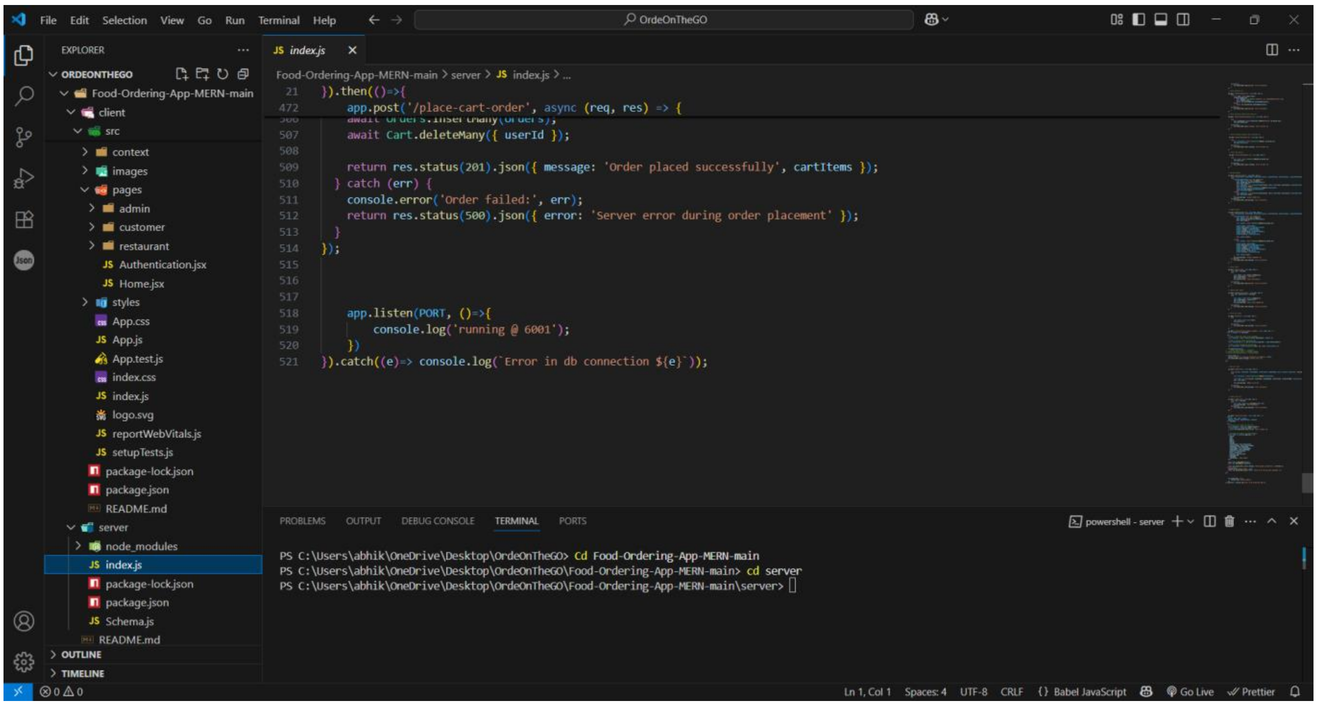
```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO> Cd Food-Ordering-App-MERN-main
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main> cd server
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main\server>
```

6.3.1. Setup express server:

- Create index.js file.
- Create an express server on your desired port number.
- Define API’s

Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

Reference Image:



```
}).then(()=>{
    app.post('/place-cart-order', async (req, res) => {
    await Orders.insertMany(orders);
    await Cart.deleteMany({ userId });

    return res.status(201).json({ message: 'Order placed successfully', cartItems });
  } catch (err) {
    console.error('Order failed:', err);
    return res.status(500).json({ error: 'Server error during order placement' });
  }
});



    app.listen(PORT, ()=>{
        console.log('running @ 6001');
    })
}).catch((e)=> console.log(`Error in db connection ${e}`));
```

```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO> Cd Food-Ordering-App-MERN-main
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main> cd server
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main\server> 
```

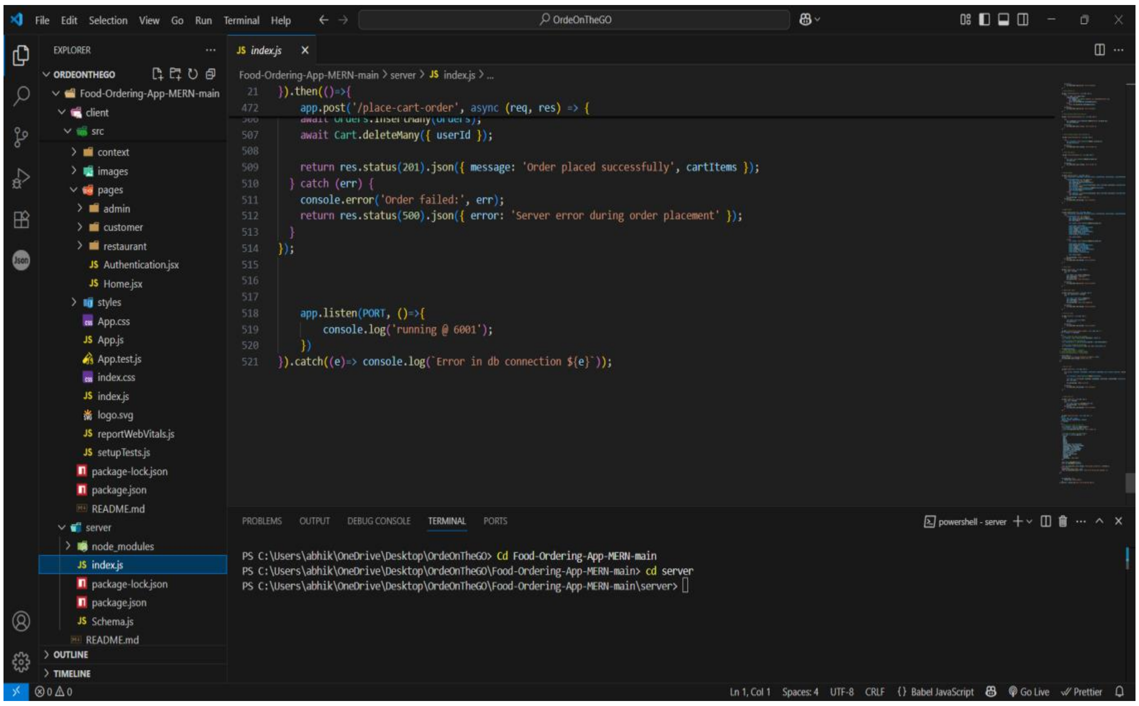
6.3.2. Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.
- Create a database and define the necessary collections for admin, users, restaurants, food products, orders,and other relevant data.

Reference Video of connect node with mongoDB database: https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

Reference Article: https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/

Reference Image:



```
}).then(()=>{
    app.post('/place-cart-order', async (req, res) => {
    await Orders.insertMany(orders);
    await Cart.deleteMany({ userId });

    return res.status(201).json({ message: 'Order placed successfully', cartItems });
  } catch (err) {
    console.error('Order failed:', err);
    return res.status(500).json({ error: 'Server error during order placement' });
  }
});



    app.listen(PORT, ()=>{
        console.log('running @ 6001');
    })
}).catch((e)=> console.log(`Error in db connection ${e}`));
```

```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO> Cd Food-Ordering-App-MERN-main
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main> cd server
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main\server> 
```

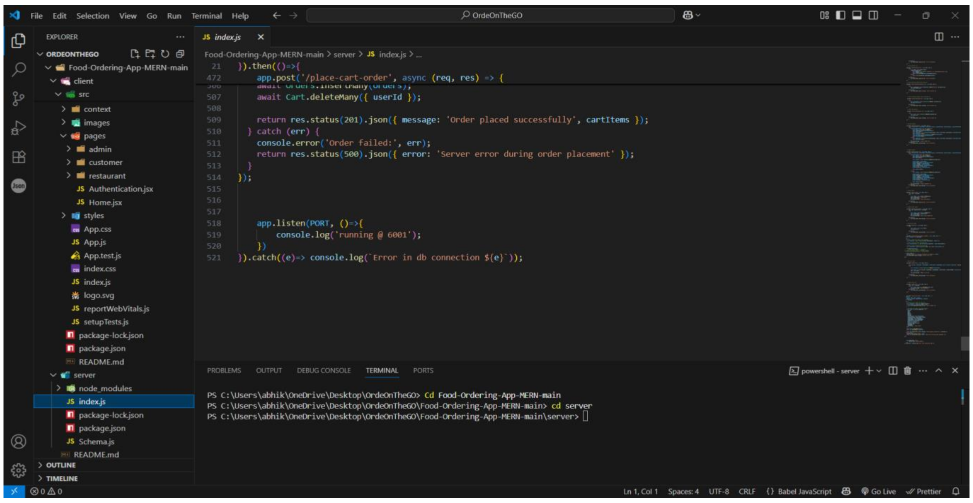
6.3.3. Create Express.js Server:

• Set up an Express.js server to handle HTTP requests and serve API endpoints.

• Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

Reference Image:



```
}).then(()=>{
    app.post('/place-cart-order', async (req, res) => {
    await Orders.insertMany(orders);
    await Cart.deleteMany({ userId });

    return res.status(201).json({ message: 'Order placed successfully', cartItems });
  } catch (err) {
    console.error('Order failed:', err);
    return res.status(500).json({ error: 'Server error during order placement' });
  }
});



    app.listen(PORT, ()=>{
        console.log('running @ 6001');
    })
}).catch((e)=> console.log(`Error in db connection ${e}`));
```

```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO> cd Food-Ordering-App-MERN-main
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main> cd server
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main\server> 
```

6.3.4. Define API Routes:

• Create separate route files for different API functionalities such as users, orders, and authentication.

• Define the necessary routes for listing products, handling user registration and login,managing orders, etc.

• Implement route handlers using Express.js to handle requests and interact with the database.

6.3.5. Implement Data Models:

• Define Mongoose schemas for the different data entities like products, users, and orders.

• Create corresponding Mongoose models to interact with the MongoDB database.

• Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6.3.6. User Authentication:

• Create routes and middleware for user registration, login, and logout.

• Set up authentication middleware to protect routes that require user authentication.

6.3.7. Handle new products and Orders:

• Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.

• Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

6.3.8. Admin Functionality:

• Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.

• Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

6.3.9. Error Handling:

• Implement error handling middleware to catch and handle any errors that occur during the API requests.

• Return appropriate error responses with relevant error messages and HTTP status codes.

6.4 FRONTEND DEVELOPMENT:

6.4.1. Setup React Application:

- Create a React app in the client folder.
- Install required libraries
- Create required pages and components and add routes.

6.4.2. Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

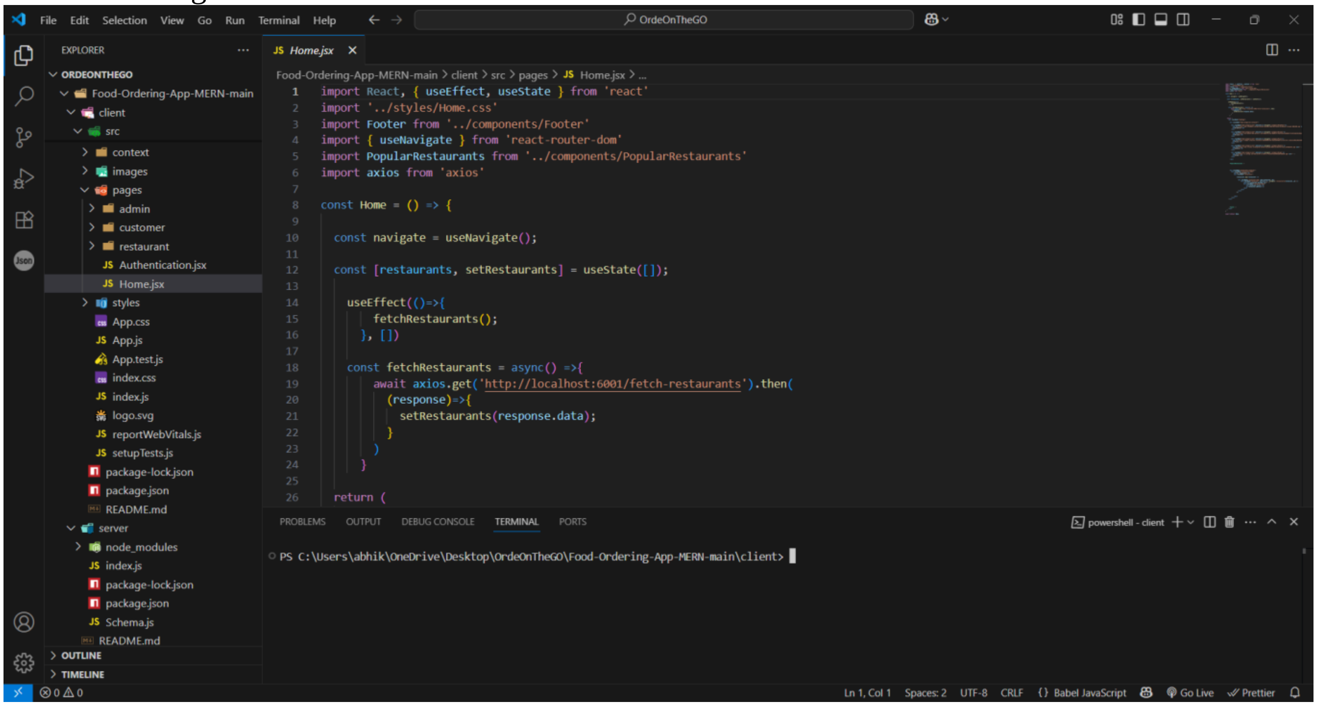
6.4.3. Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Video Link:
https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing

Reference Article Link:
https://www.w3schools.com/react/react_getstarted.asp

Reference Image:



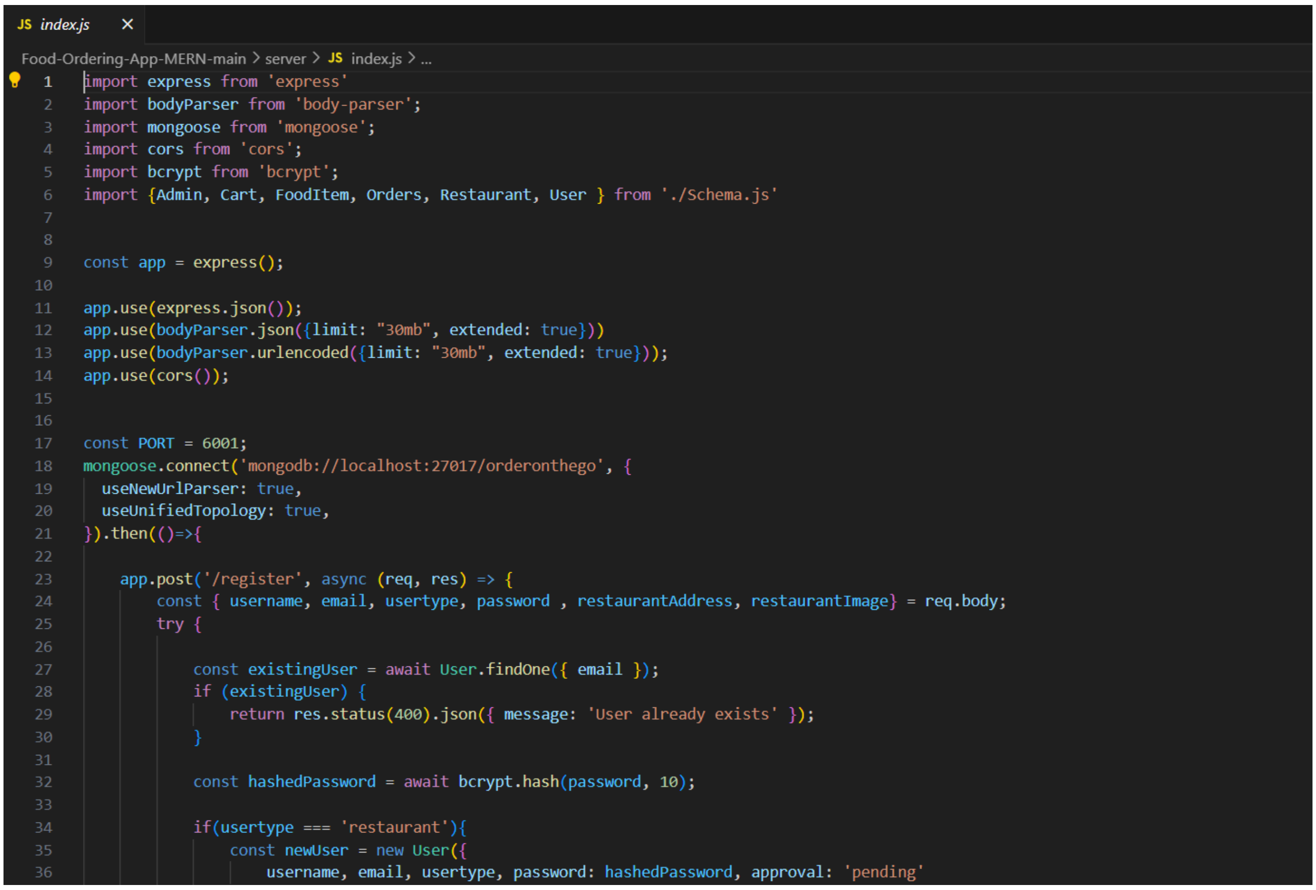
```jsx
import React, { useEffect, useState } from 'react'
import '../styles/Home.css'
import Footer from '../components/Footer'
import { useNavigate } from 'react-router-dom'
import PopularRestaurants from '../components/PopularRestaurants'
import axios from 'axios'

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);
          }
        )
    }

  return (
```

```
PS C:\Users\abhik\OneDrive\Desktop\OrdeOnTheGO\Food-Ordering-App-MERN-main\client>
```

6.5 CODE EXPLANATION

Server setup:

Let us import all the required tools/libraries and connect the database.



```
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://localhost:27017/orderonthego', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(()=>{

    app.post('/register', async (req, res) => {
        const { username, email, usertype, password , restaurantAddress, restaurantImage} = req.body;
        try {

            const existingUser = await User.findOne({ email });
            if (existingUser) {
                return res.status(400).json({ message: 'User already exists' });
            }

            const hashedPassword = await bcrypt.hash(password, 10);

            if(usertype === 'restaurant'){
                const newUser = new User({
                    username, email, usertype, password: hashedPassword, approval: 'pending'
```

7.API DOCUMENTATION

Here's a sample API Documentation section for OrderOnTheGo project backend:

All endpoints follow the base URL:
https://your-backend-url.com/api

1. User Authentication
POST /auth/register
Registers a new user (Customer, Restaurant, Admin)

Body Parameters:
```
{
  "role": "customer",
  "email": "user@example.com",
  "password": "123456"
}
```

Response:

```
{
  "message": "User registered successfully"
}
```

POST /auth/login
Logs in a user

Body Parameters:

```
{
  "email": "user@example.com",
  "password": "123456"
}
```

Response:

```
{
  "token": "jwt-token",
  "user": {
    "id": "userId",
    "role": "customer"
  }
}
```

2. Restaurant APIs
GET /restaurants
Get all restaurants

Response:

```
[
  {
    "id": "rest1",
    "name": "SB Foods",
    "category": "Indian",
    "rating": 4.5
  }
]
```

POST /restaurants
Add new restaurant (Admin only)

Body Parameters:

```
{
  "name": "AHO Foods",
  "category": "Chinese"
}
```

3. Food Items
GET /fooditems/:restaurantId
Get all food items for a restaurant

POST /fooditems
Add food item (Restaurant role)

Body Parameters:
```
{
  "restaurantId": "rest1",
  "name": "Chicken Pizza",
  "price": 250,
  "discount": 10,
  "rating": 4.2
}
```

4. Cart
GET /cart/:userId
Get user's cart items

POST /cart
Add item to cart

Body Parameters:
```
{
  "userId": "cust1",
  "foodItemId": "item123",
  "quantity": 2
}
```

DELETE /cart/:cartItemId
Remove item from cart

5. Orders
POST /orders
Place an order

Body Parameters:
```
{
  "userId": "cust1",
  "items": [
    {
      "foodItemId": "item123",
      "quantity": 2
    }
  ],
  "payment": "COD"
}
```

GET /orders/user/:userId
Get all orders placed by a user

GET /orders/restaurant/:restaurantId
Get all orders received by a restaurant

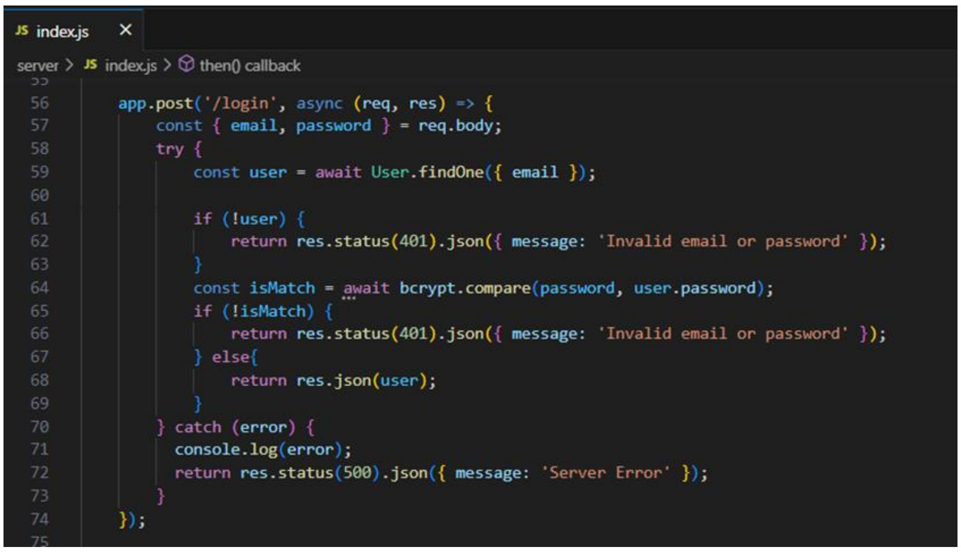
6. Invoice
GET /invoice/:orderId
Download invoice as PDF
Response: File download

8.AUTHENTICATION

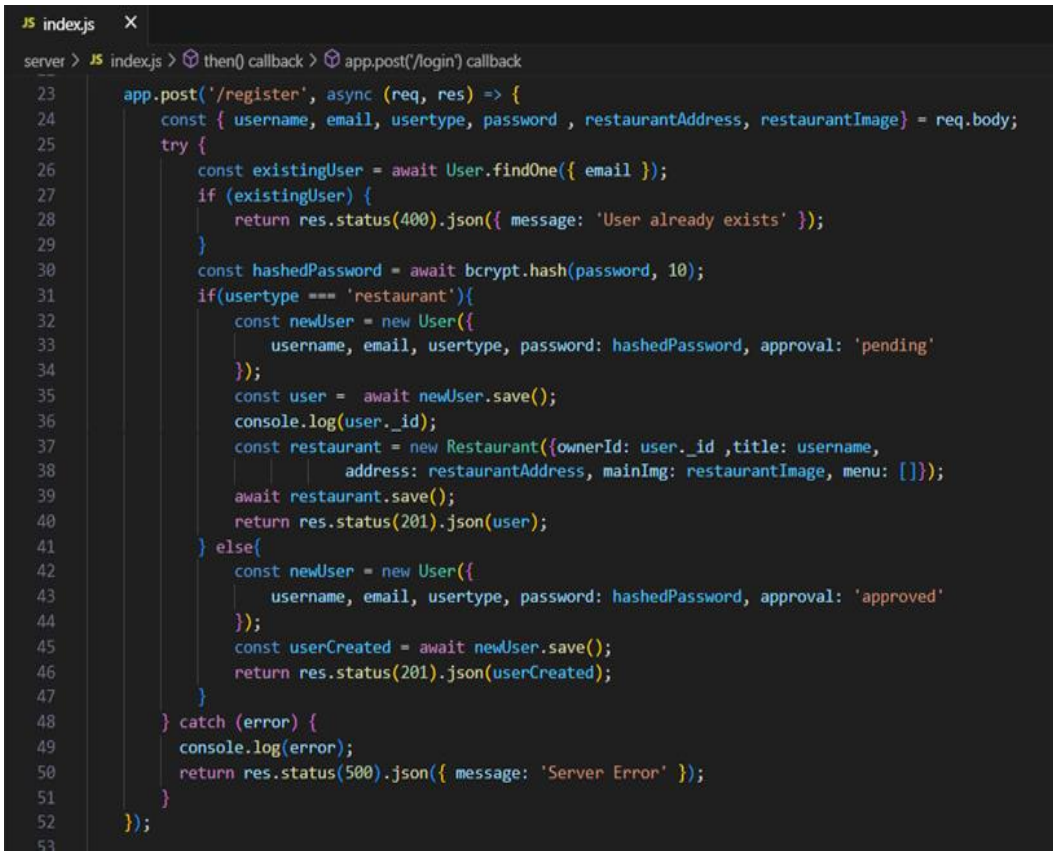
User Authentication:

- **Backend**

Now, here we define the functions to handle http requests from the client for authentication.



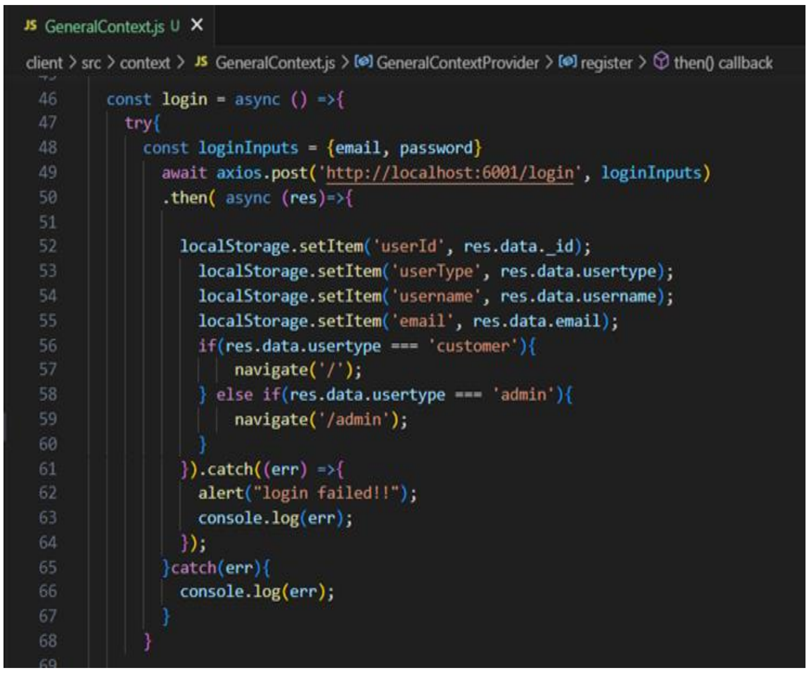
```javascript
app.post('/login', async (req, res) => {
    const { email, password } = req.body;
    try {
        const user = await User.findOne({ email });

        if (!user) {
            return res.status(401).json({ message: 'Invalid email or password' });
        }
        const isMatch = await bcrypt.compare(password, user.password);
        if (!isMatch) {
            return res.status(401).json({ message: 'Invalid email or password' });
        } else{
            return res.json(user);
        }
    } catch (error) {
      console.log(error);
      return res.status(500).json({ message: 'Server Error' });
    }
});
```


index.js
server > index.js > then() callback > app.post('/login') callback

```javascript
app.post('/register', async (req, res) => {
    const { username, email, usertype, password , restaurantAddress, restaurantImage} = req.body;
    try {
        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'User already exists' });
        }
        const hashedPassword = await bcrypt.hash(password, 10);
        if(usertype === 'restaurant'){
            const newUser = new User({
                username, email, usertype, password: hashedPassword, approval: 'pending'
            });
            const user =  await newUser.save();
            console.log(user._id);
            const restaurant = new Restaurant({ownerId: user._id ,title: username,
                        address: restaurantAddress, mainImg: restaurantImage, menu: []});
            await restaurant.save();
            return res.status(201).json(user);
        } else{
            const newUser = new User({
                username, email, usertype, password: hashedPassword, approval: 'approved'
            });
            const userCreated = await newUser.save();
            return res.status(201).json(userCreated);
        }
    } catch (error) {
      console.log(error);
      return res.status(500).json({ message: 'Server Error' });
    }
});
```

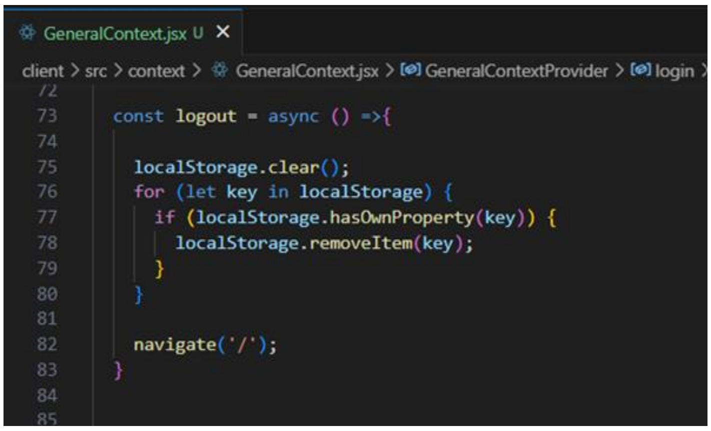
· **Frontend**

Login:


GeneralContext.js U
client > src > context > GeneralContext.js > GeneralContextProvider > register > then() callback

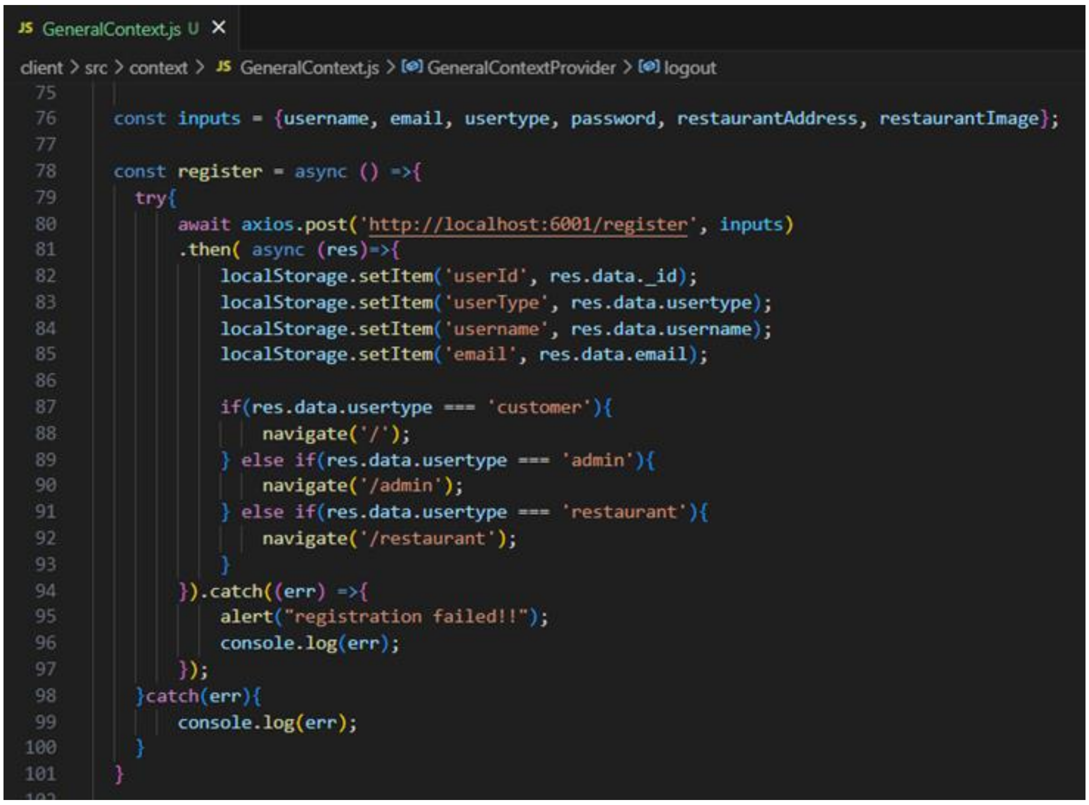
```javascript
const login = async () =>{
  try{
    const loginInputs = {email, password}
      await axios.post('http://localhost:6001/login', loginInputs)
      .then( async (res)=>{

        localStorage.setItem('userId', res.data._id);
          localStorage.setItem('userType', res.data.usertype);
          localStorage.setItem('username', res.data.username);
          localStorage.setItem('email', res.data.email);
          if(res.data.usertype === 'customer'){
              navigate('/');
          } else if(res.data.usertype === 'admin'){
              navigate('/admin');
          }
        }).catch((err) =>{
          alert("login failed!!");
          console.log(err);
        });
      }catch(err){
        console.log(err);
      }
    }
```

Logout:


GeneralContext.jsx U
client > src > context > GeneralContext.jsx > GeneralContextProvider > login

```
const logout = async () =>{

  localStorage.clear();
  for (let key in localStorage) {
    if (localStorage.hasOwnProperty(key)) {
      localStorage.removeItem(key);
    }
  }

  navigate('/');
}
```

Register:


GeneralContext.js U
client > src > context > GeneralContext.js > GeneralContextProvider > logout

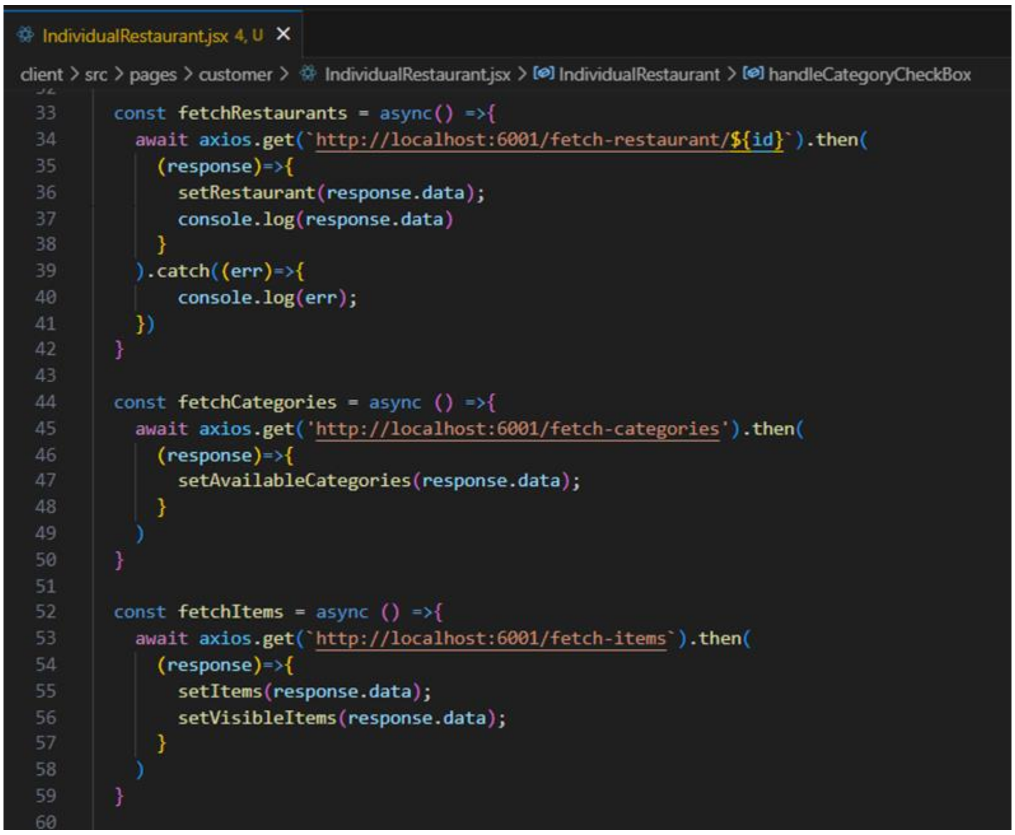
```

const inputs = {username, email, usertype, password, restaurantAddress, restaurantImage};

const register = async () =>{
  try{
      await axios.post('http://localhost:6001/register', inputs)
      .then( async (res)=>{
          localStorage.setItem('userId', res.data._id);
          localStorage.setItem('userType', res.data.usertype);
          localStorage.setItem('username', res.data.username);
          localStorage.setItem('email', res.data.email);

          if(res.data.usertype === 'customer'){
              navigate('/');
          } else if(res.data.usertype === 'admin'){
              navigate('/admin');
          } else if(res.data.usertype === 'restaurant'){
              navigate('/restaurant');
          }
      }).catch((err) =>{
          alert("registration failed!!");
          console.log(err);
      });
  }catch(err){
      console.log(err);
  }
}
```

All Products (User):

· Frontend

In the home page, we’ll fetch all the products available in the platform along with the filters.

Fetching food items:

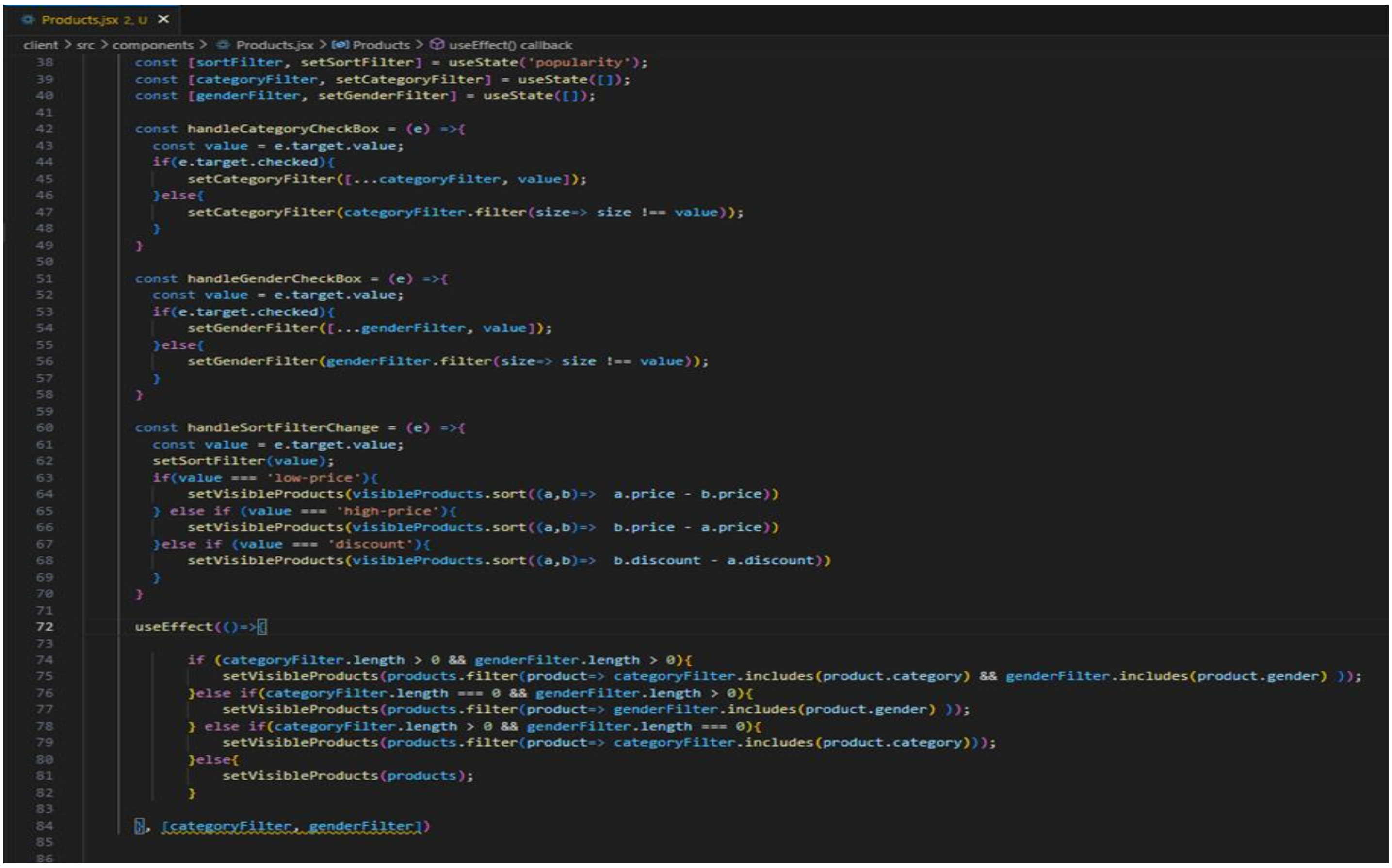


IndividualRestaurant.jsx 4, U ×

client > src > pages > customer > IndividualRestaurant.jsx > IndividualRestaurant > handleCategoryCheckBox

```
  const fetchRestaurants = async() =>{
    await axios.get(`http://localhost:6001/fetch-restaurant/${id}`).then(
      (response)=>{
        setRestaurant(response.data);
        console.log(response.data)
      }
    ).catch((err)=>{
        console.log(err);
    })
  }

  const fetchCategories = async () =>{
    await axios.get('http://localhost:6001/fetch-categories').then(
      (response)=>{
        setAvailableCategories(response.data);
      }
    )
  }

  const fetchItems = async () =>{
    await axios.get(`http://localhost:6001/fetch-items`).then(
      (response)=>{
        setItems(response.data);
        setVisibleItems(response.data);
      }
    )
  }

```

Filtering products:



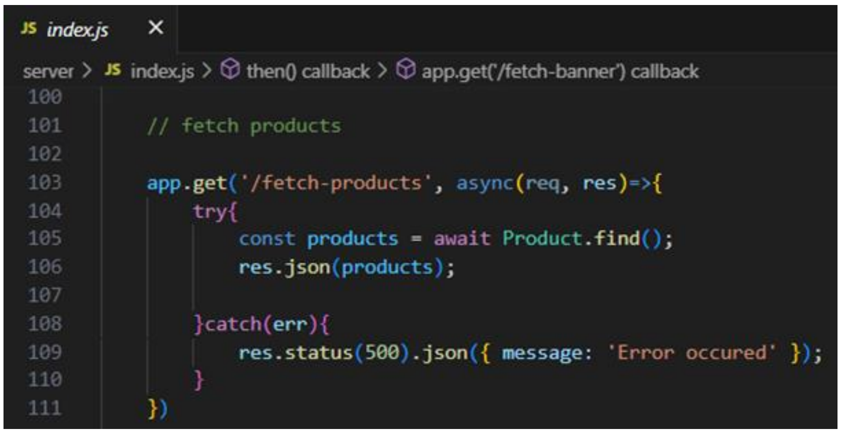
Products.jsx 2, U ×

client > src > components > Products.jsx > Products > useEffect() callback

```
    const [sortFilter, setSortFilter] = useState('popularity');
    const [categoryFilter, setCategoryFilter] = useState([]);
    const [genderFilter, setGenderFilter] = useState([]);

    const handleCategoryCheckBox = (e) =>{
      const value = e.target.value;
      if(e.target.checked){
          setCategoryFilter([...categoryFilter, value]);
      }else{
          setCategoryFilter(categoryFilter.filter(size=> size !== value));
      }
    }

    const handleGenderCheckBox = (e) =>{
      const value = e.target.value;
      if(e.target.checked){
          setGenderFilter([...genderFilter, value]);
      }else{
          setGenderFilter(genderFilter.filter(size=> size !== value));
      }
    }

    const handleSortFilterChange = (e) =>{
      const value = e.target.value;
      setSortFilter(value);
      if(value === 'low-price'){
          setVisibleProducts(visibleProducts.sort((a,b)=>  a.price - b.price))
      } else if (value === 'high-price'){
          setVisibleProducts(visibleProducts.sort((a,b)=>  b.price - a.price))
      }else if (value === 'discount'){
          setVisibleProducts(visibleProducts.sort((a,b)=>  b.discount - a.discount))
      }
    }

    useEffect(()=>{

        if (categoryFilter.length > 0 && genderFilter.length > 0){
            setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category) && genderFilter.includes(product.gender) ));
        }else if(categoryFilter.length === 0 && genderFilter.length > 0){
            setVisibleProducts(products.filter(product=> genderFilter.includes(product.gender) ));
        } else if(categoryFilter.length > 0 && genderFilter.length === 0){
            setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category)));
        }else{
            setVisibleProducts(products);
        }

    }, [categoryFilter, genderFilter])

```

- **Backend**

In the backend, we fetch all the products and then filter them on the client side.



index.js

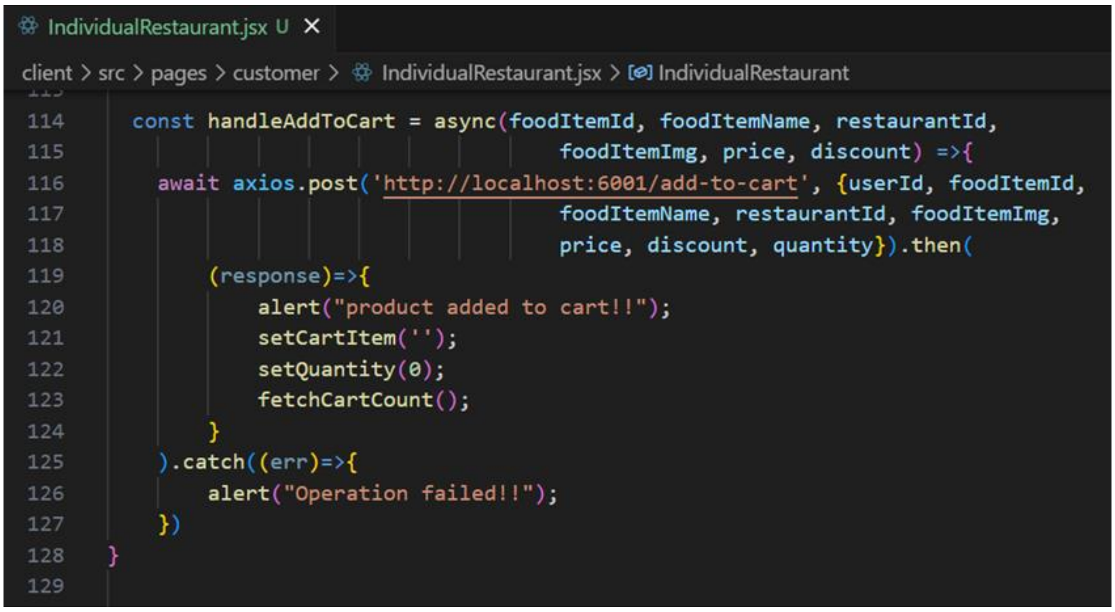
server > index.js > then() callback > app.get('/fetch-banner') callback

```
    // fetch products

    app.get('/fetch-products', async(req, res)=>{
        try{
            const products = await Product.find();
            res.json(products);

        }catch(err){
            res.status(500).json({ message: 'Error occured' });
        }
    })
```

Add product to cart:

- **Frontend**

Here, we can add the product to the cart and later can buy them.



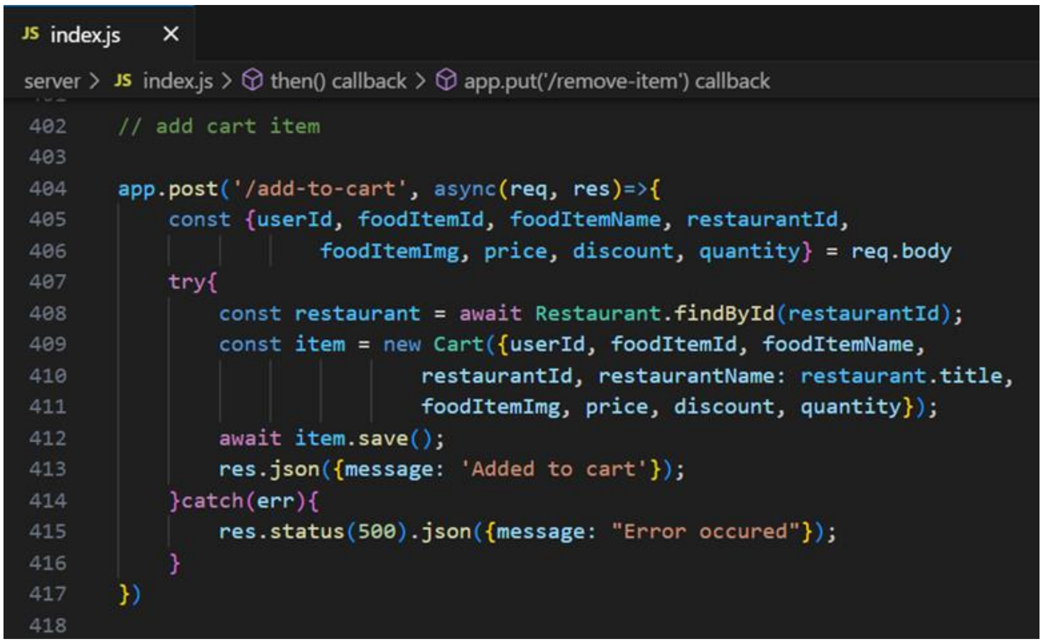
IndividualRestaurant.jsx U

client > src > pages > customer > IndividualRestaurant.jsx > IndividualRestaurant

```
  const handleAddToCart = async(foodItemId, foodItemName, restaurantId,
                                   foodItemImg, price, discount) =>{
    await axios.post('http://localhost:6001/add-to-cart', {userId, foodItemId,
                                   foodItemName, restaurantId, foodItemImg,
                                   price, discount, quantity}).then(
        (response)=>{
            alert("product added to cart!!");
            setCartItem('');
            setQuantity(0);
            fetchCartCount();
        }
    ).catch((err)=>{
        alert("Operation failed!!");
    })
  }
```

- **Backend**

Add product to cart:

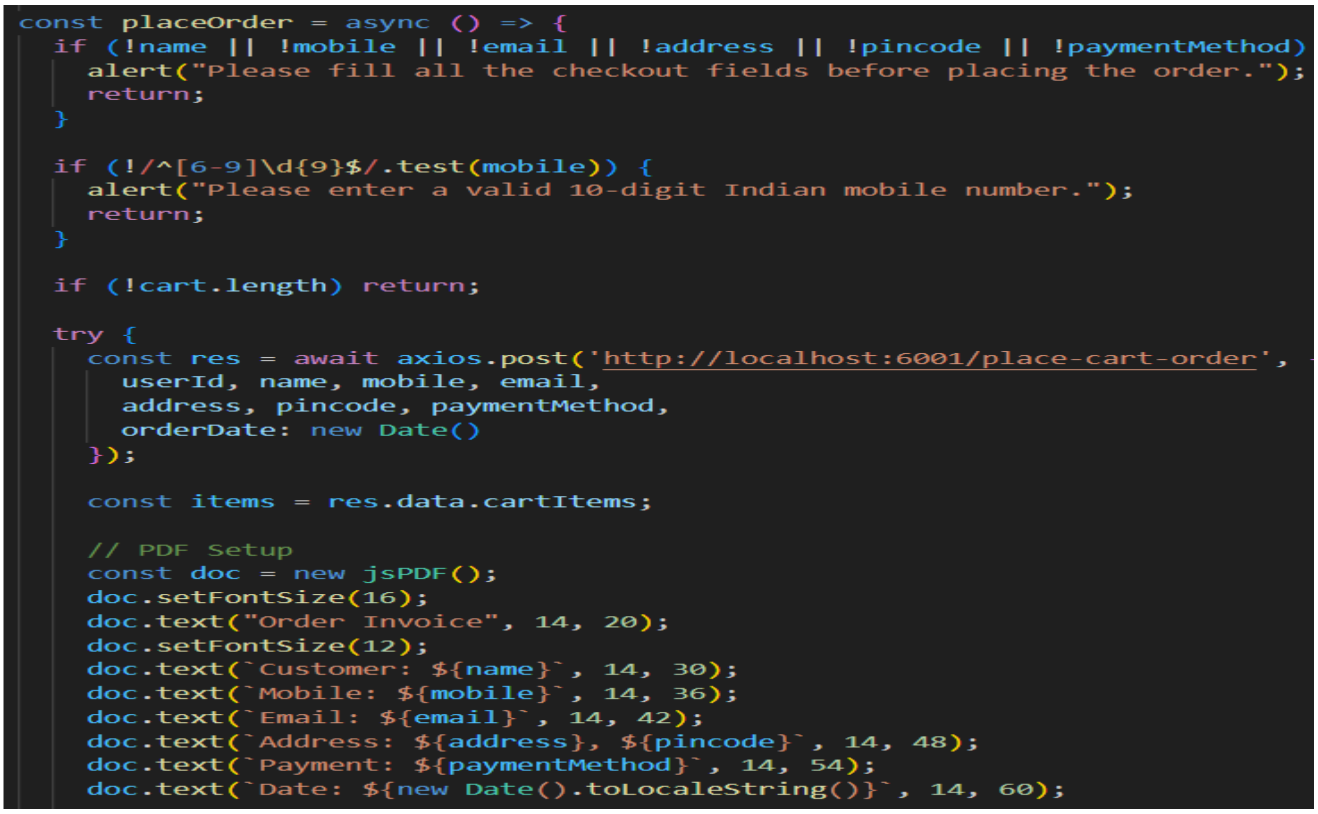

index.js
server > index.js > then() callback > app.put('/remove-item') callback

```javascript
// add cart item

app.post('/add-to-cart', async(req, res)=>{
    const {userId, foodItemId, foodItemName, restaurantId,
                foodItemImg, price, discount, quantity} = req.body
    try{
        const restaurant = await Restaurant.findById(restaurantId);
        const item = new Cart({userId, foodItemId, foodItemName,
                        restaurantId, restaurantName: restaurant.title,
                        foodItemImg, price, discount, quantity});
        await item.save();
        res.json({message: 'Added to cart'});
    }catch(err){
        res.status(500).json({message: "Error occured"});
    }
})
```

Order products:

Now, from the cart, let's place the order

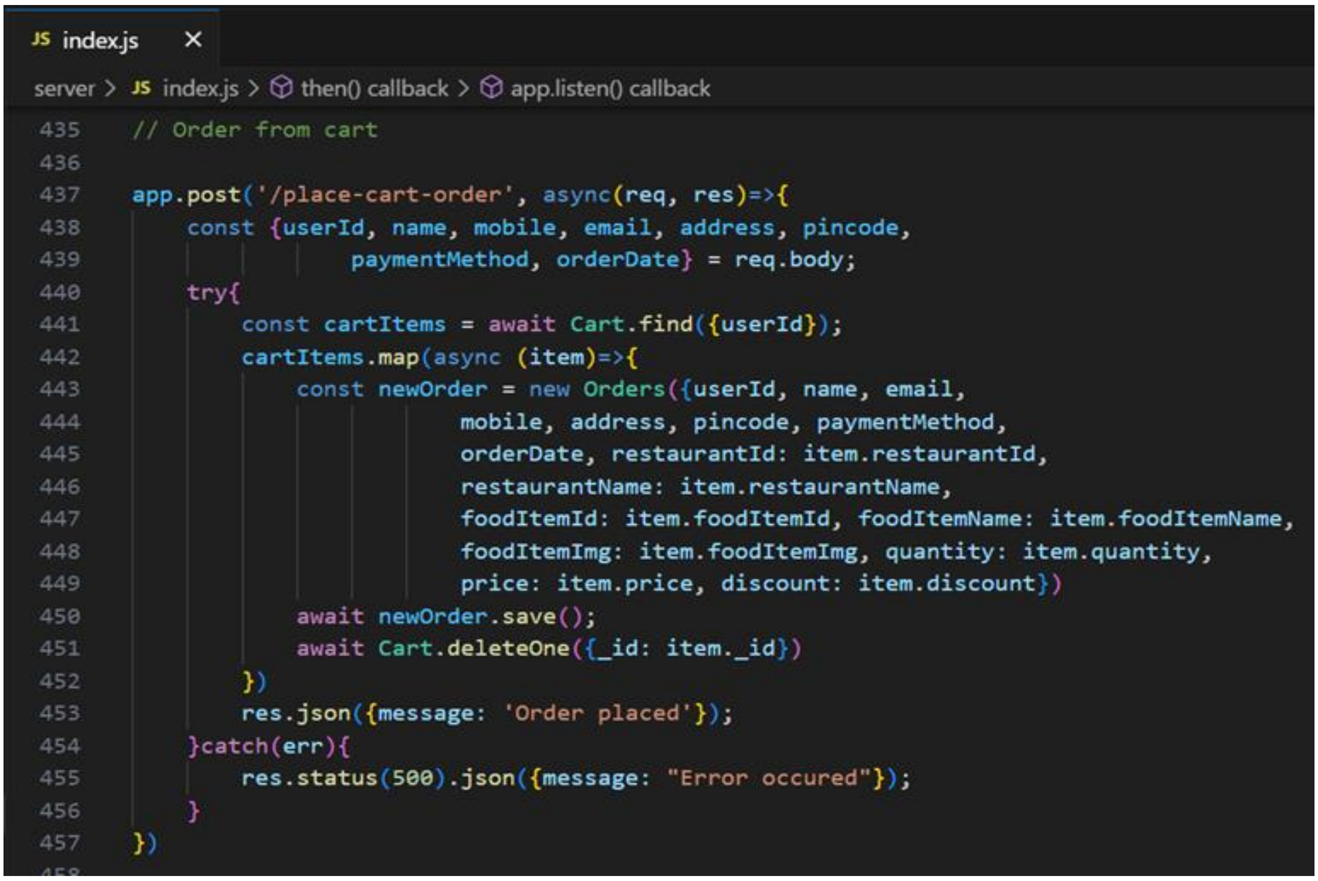
- Frontend



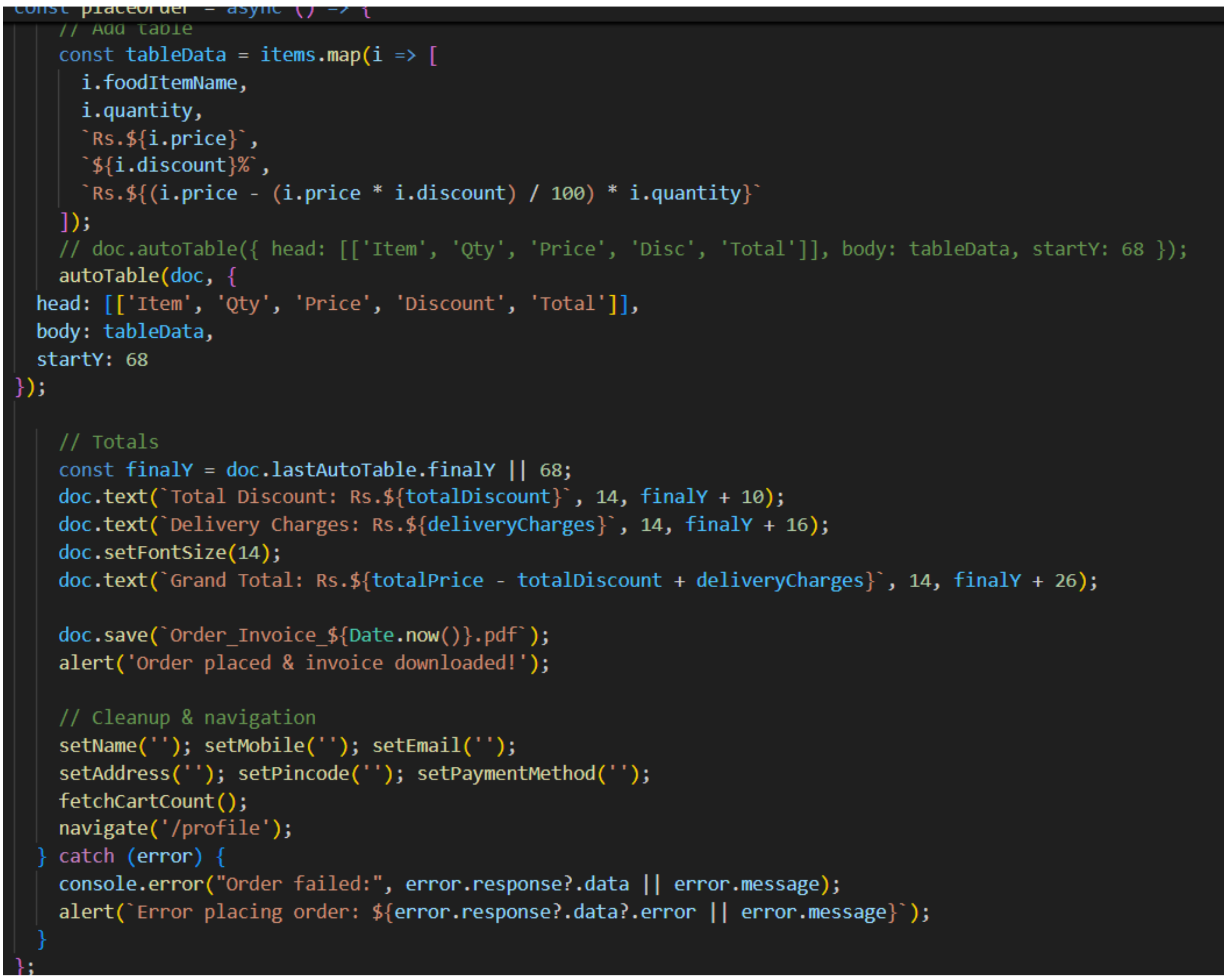
```javascript
const placeOrder = async () => {
  if (!name || !mobile || !email || !address || !pincode || !paymentMethod)
    alert("Please fill all the checkout fields before placing the order.");
    return;
  }

  if (!/^[6-9]\d{9}$/.test(mobile)) {
    alert("Please enter a valid 10-digit Indian mobile number.");
    return;
  }

  if (!cart.length) return;

  try {
    const res = await axios.post('http://localhost:6001/place-cart-order',
      userId, name, mobile, email,
      address, pincode, paymentMethod,
      orderDate: new Date()
    });

    const items = res.data.cartItems;

    // PDF Setup
    const doc = new jsPDF();
    doc.setFontSize(16);
    doc.text("Order Invoice", 14, 20);
    doc.setFontSize(12);
    doc.text(`Customer: ${name}`, 14, 30);
    doc.text(`Mobile: ${mobile}`, 14, 36);
    doc.text(`Email: ${email}`, 14, 42);
    doc.text(`Address: ${address}, ${pincode}`, 14, 48);
    doc.text(`Payment: ${paymentMethod}`, 14, 54);
    doc.text(`Date: ${new Date().toLocaleString()}`, 14, 60);
```

- **Backend**

In the backend, on receiving the request from the client, we then place the order for the products in the cart with the specific user Id.



```
// Order from cart

app.post('/place-cart-order', async(req, res)=>{
  const {userId, name, mobile, email, address, pincode,
        paymentMethod, orderDate} = req.body;
  try{
    const cartItems = await Cart.find({userId});
    cartItems.map(async (item)=>{
      const newOrder = new Orders({userId, name, email,
            mobile, address, pincode, paymentMethod,
            orderDate, restaurantId: item.restaurantId,
            restaurantName: item.restaurantName,
            foodItemId: item.foodItemId, foodItemName: item.foodItemName,
            foodItemImg: item.foodItemImg, quantity: item.quantity,
            price: item.price, discount: item.discount})
      await newOrder.save();
      await Cart.deleteOne({_id: item._id})
    })
    res.json({message: 'Order placed'});
  }catch(err){
    res.status(500).json({message: "Error occured"});
  }
})
```

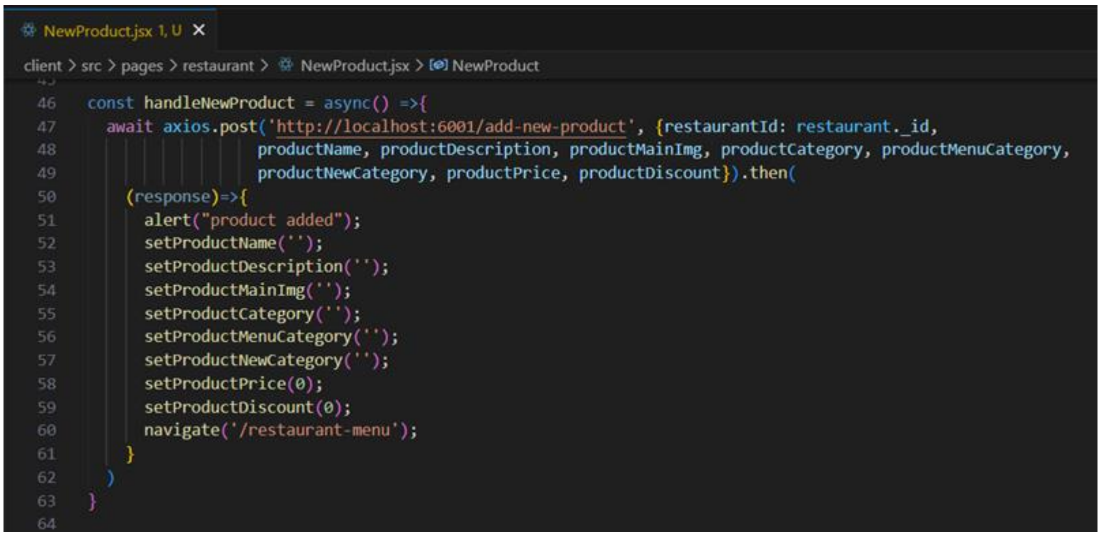


```
const placeOrder = async () => {
    // Add table
    const tableData = items.map(i => [
      i.foodItemName,
      i.quantity,
      `Rs.${i.price}`,
      `${i.discount}%`,
      `Rs.${(i.price - (i.price * i.discount) / 100) * i.quantity}`
    ]);
    // doc.autoTable({ head: [['Item', 'Qty', 'Price', 'Disc', 'Total']], body: tableData, startY: 68 });
    autoTable(doc, {
  head: [['Item', 'Qty', 'Price', 'Discount', 'Total']],
  body: tableData,
  startY: 68
});

    // Totals
    const finalY = doc.lastAutoTable.finalY || 68;
    doc.text(`Total Discount: Rs.${totalDiscount}`, 14, finalY + 10);
    doc.text(`Delivery Charges: Rs.${deliveryCharges}`, 14, finalY + 16);
    doc.setFontSize(14);
    doc.text(`Grand Total: Rs.${totalPrice - totalDiscount + deliveryCharges}`, 14, finalY + 26);

    doc.save(`Order_Invoice_${Date.now()}.pdf`);
    alert('Order placed & invoice downloaded!');

    // Cleanup & navigation
    setName(''); setMobile(''); setEmail('');
    setAddress(''); setPincode(''); setPaymentMethod('');
    fetchCartCount();
    navigate('/profile');
  } catch (error) {
    console.error("Order failed:", error.response?.data || error.message);
    alert(`Error placing order: ${error.response?.data?.error || error.message}`);
  }
};
```

Add new product:

Here, in the admin dashboard, we will add a new product.

- Frontend:

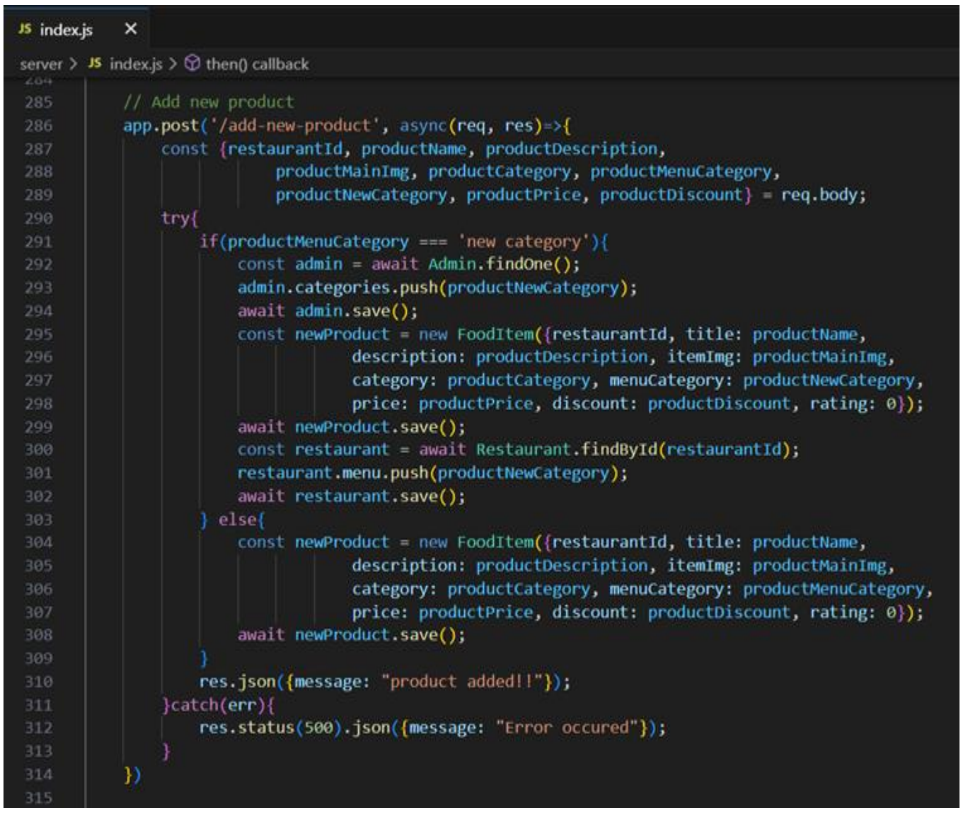


NewProduct.jsx 1, U

client > src > pages > restaurant > NewProduct.jsx > NewProduct

```
const handleNewProduct = async() =>{
  await axios.post('http://localhost:6001/add-new-product', {restaurantId: restaurant._id,
                productName, productDescription, productMainImg, productCategory, productMenuCategory,
                productNewCategory, productPrice, productDiscount}).then(
    (response)=>{
      alert("product added");
      setProductName('');
      setProductDescription('');
      setProductMainImg('');
      setProductCategory('');
      setProductMenuCategory('');
      setProductNewCategory('');
      setProductPrice(0);
      setProductDiscount(0);
      navigate('/restaurant-menu');
    }
  )
}
```

- **Backend:**



index.js

server > index.js > then() callback

```
    // Add new product
    app.post('/add-new-product', async(req, res)=>{
        const {restaurantId, productName, productDescription,
                    productMainImg, productCategory, productMenuCategory,
                    productNewCategory, productPrice, productDiscount} = req.body;
        try{
            if(productMenuCategory === 'new category'){
                const admin = await Admin.findOne();
                admin.categories.push(productNewCategory);
                await admin.save();
                const newProduct = new FoodItem({restaurantId, title: productName,
                            description: productDescription, itemImg: productMainImg,
                            category: productCategory, menuCategory: productNewCategory,
                            price: productPrice, discount: productDiscount, rating: 0});
                await newProduct.save();
                const restaurant = await Restaurant.findById(restaurantId);
                restaurant.menu.push(productNewCategory);
                await restaurant.save();
            } else{
                const newProduct = new FoodItem({restaurantId, title: productName,
                            description: productDescription, itemImg: productMainImg,
                            category: productCategory, menuCategory: productMenuCategory,
                            price: productPrice, discount: productDiscount, rating: 0});
                await newProduct.save();
            }
            res.json({message: "product added!!"});
        }catch(err){
            res.status(500).json({message: "Error occured"});
        }
    })
```

Along with this, implement additional features to view all orders, products, etc., in the admin dashboard.

9.USER INTERFACE

1. Design Theme

- Light Theme with soft backgrounds
- Clean layout, large clickable buttons
- Responsive design (works on desktops, tablets, mobiles)
- Smooth animations and transitions for a modern feel

2. Customer Interface

- Home Page: Explore restaurant categories (e.g., Indian, Fast Food, Chinese)
- Restaurant List: Displays restaurant cards with name, rating, and category
- Menu Page: View food items with name, price, rating, discount, and “Add to Cart” button
- Cart Page: View added items
- Place order
- Order Confirmation: Shows success message and download invoice button
- Order History Page: View past orders with details

3. Restaurant Interface

- Dashboard: Overview of new and active orders
- Menu Management: Add/Edit/Delete food items
- Set price, rating, and availability
- Order List: View incoming orders with customer details
- Update order status

4. Admin Interface

Admin Panel:
- View list of users (customers/restaurants)
- Remove/block users
- Monitor total orders, system activity

5. UI Features

- Navigation Bar: Role-based dynamic links (e.g., Cart for customers, Dashboard for restaurants)
- Toast Notifications: Success and error messages for actions (e.g., "Item added to cart", "Login failed")
- Modals: For confirmation actions (e.g., deleting food item, placing order)
- Forms: Styled and validated for login, registration, add food, etc.

10. TESTING

10.1 Testing

The testing approach in OrderOnTheGo focused on validating functionality, reliability, and role-based access across all major user flows. The strategy was executed in the following phases:

1. Manual Testing
Primary strategy during development
All core features tested manually (e.g., registration, order placement, invoice generation)

2. Functional Testing
Verified that all endpoints and UI components function as intended
Focused on each role (Customer, Restaurant, Admin) independently

3. User Acceptance Testing (UAT)
Conducted at the end of development
Real-world scenarios used to validate the project against user expectations

4. Role-Based Testing
Ensured that users only access allowed functionalities
Example: Customers cannot access restaurant dashboards

5. Black Box Testing
Inputs and outputs tested without checking internal logic
Applied to modules like login, add to cart, place order, etc.

10.2 Tools used

1. Postman
Used for API testing and verification
Validated response codes, payloads, and error handling for endpoints like /login, /orders, /fooditems

2. Browser DevTools (Chrome/Edge)
Inspected frontend rendering
Checked for network errors, response time, and responsiveness

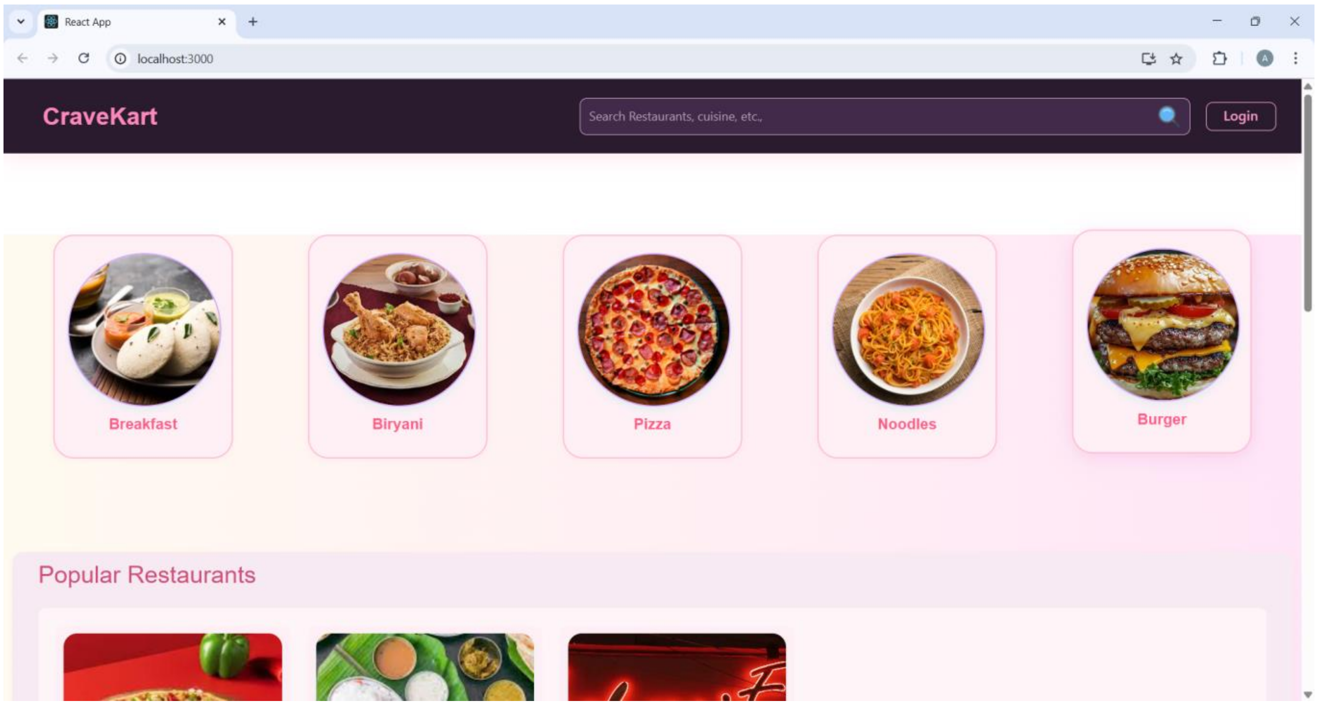
3. VS Code Debugger
Tracked console errors during local development
Ensured proper flow of data between frontend and backend

4. Manual Checklists
Test cases were written and marked manually
Covered positive/negative cases and edge scenarios (e.g., empty cart, invalid credentials)

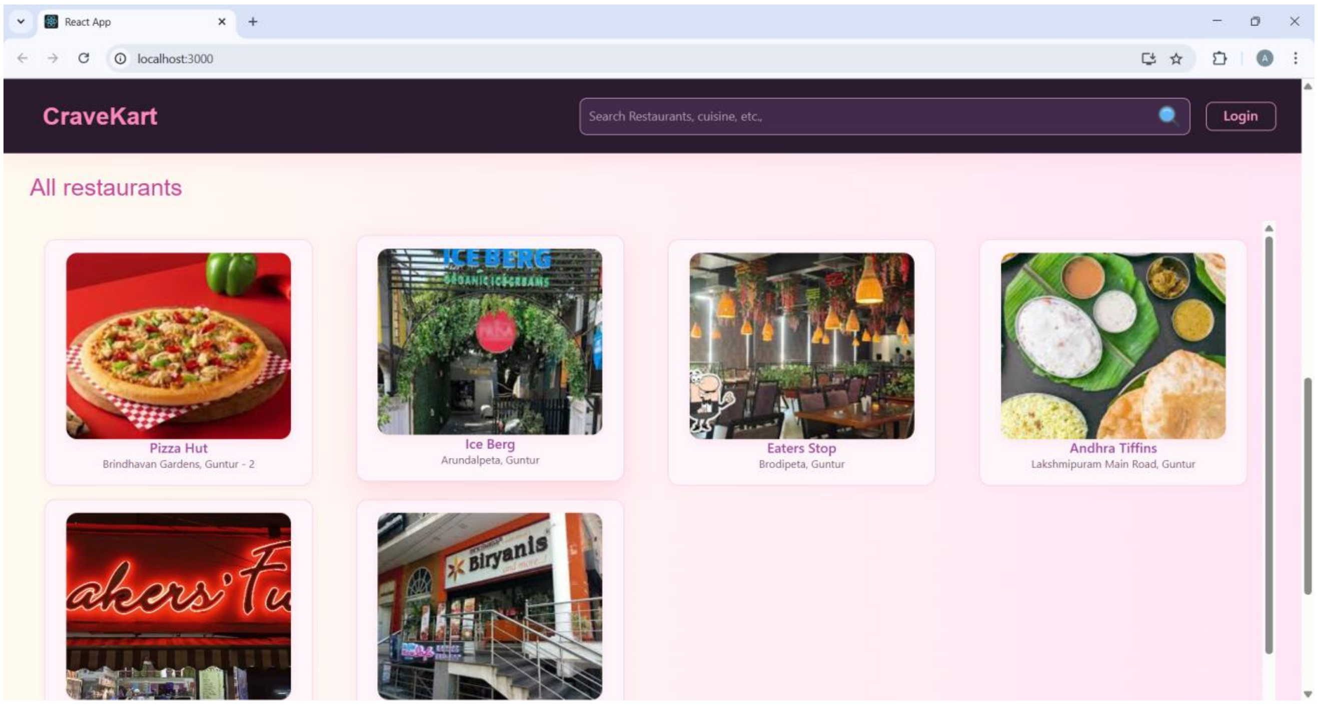
5. PDF Invoice Viewer
Tested generated invoices for accuracy and readability using built-in browser PDF viewer

11.SCREENSHOTS AND DEMO

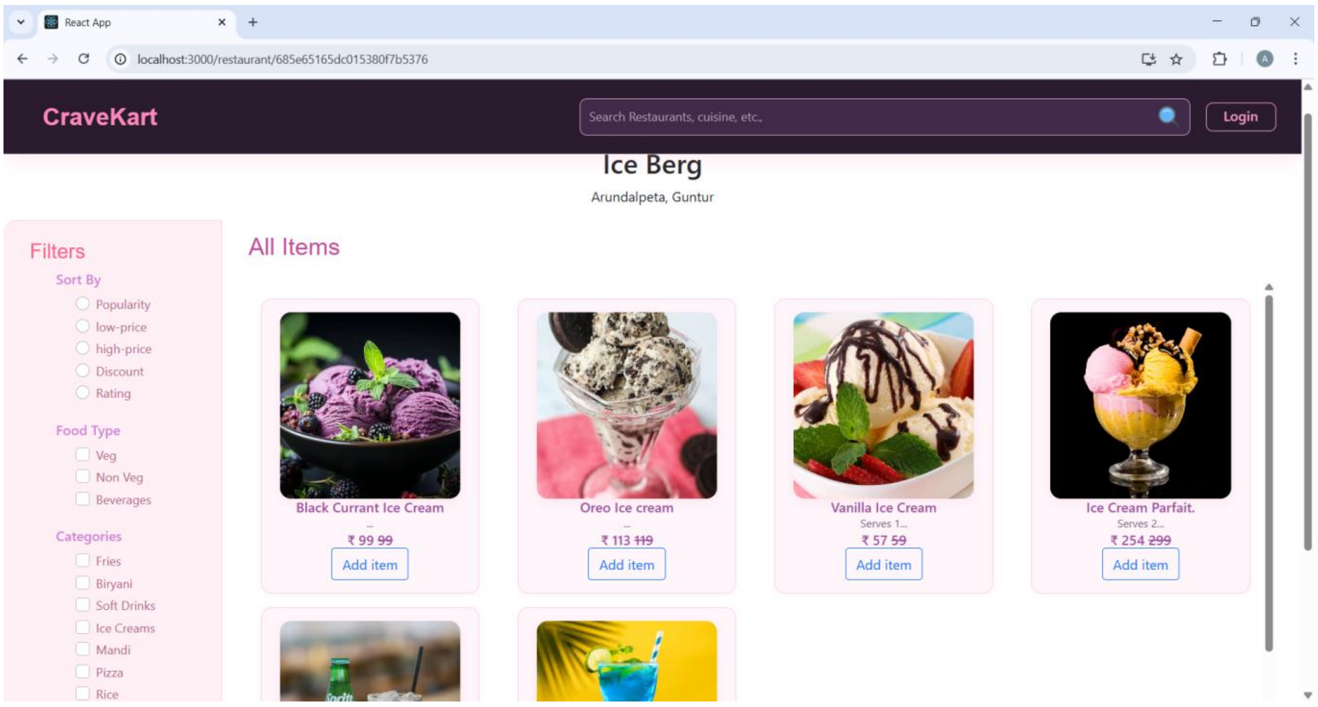
- **Landing page**



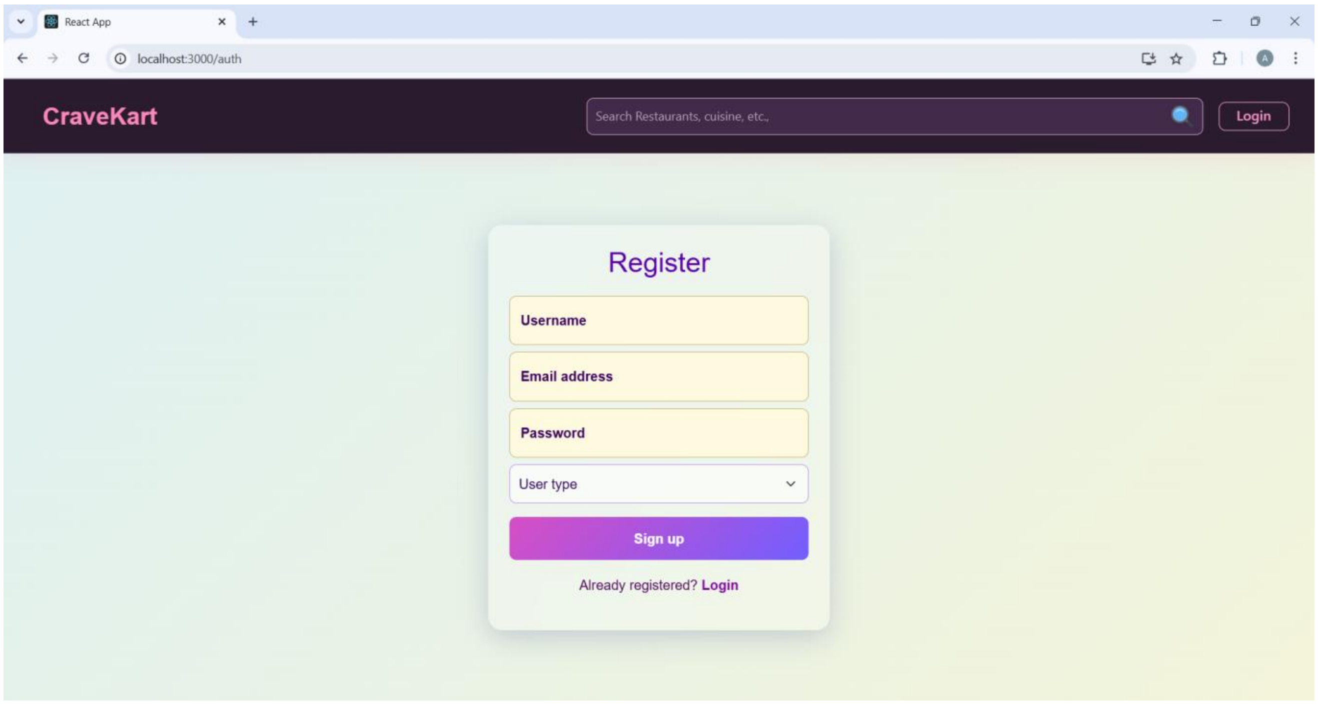
- **Restaurants**



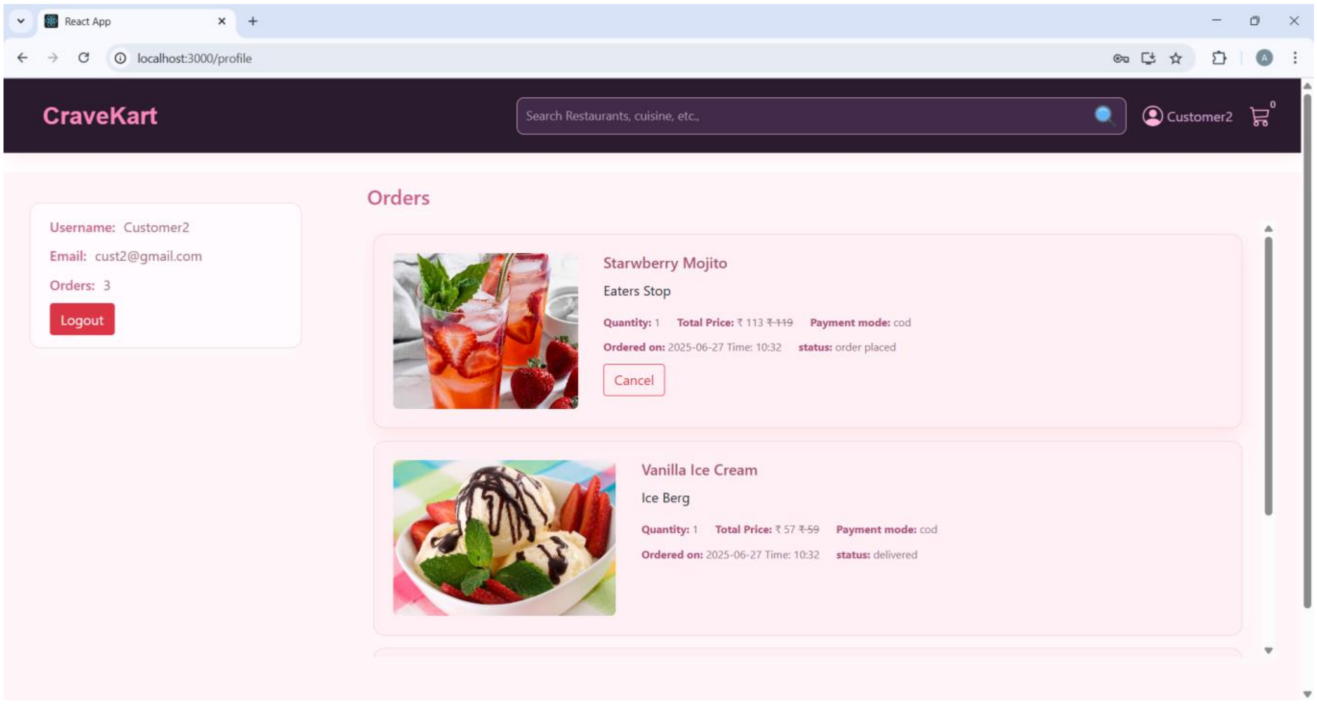
- **Restaurant Menu**



- **Authentication**



- **User Profile**


React App

localhost:3000/profile

CraveKart

Search Restaurants, cuisine, etc.,

Customer2

Username: Customer2

Email: cust2@gmail.com

Orders: 3

Logout

Orders

Starwberry Mojito

Eaters Stop

Quantity: 1 Total Price: ₹ 113 ~~₹ 119~~ Payment mode: cod

Ordered on: 2025-06-27 Time: 10:32 status: order placed

Cancel

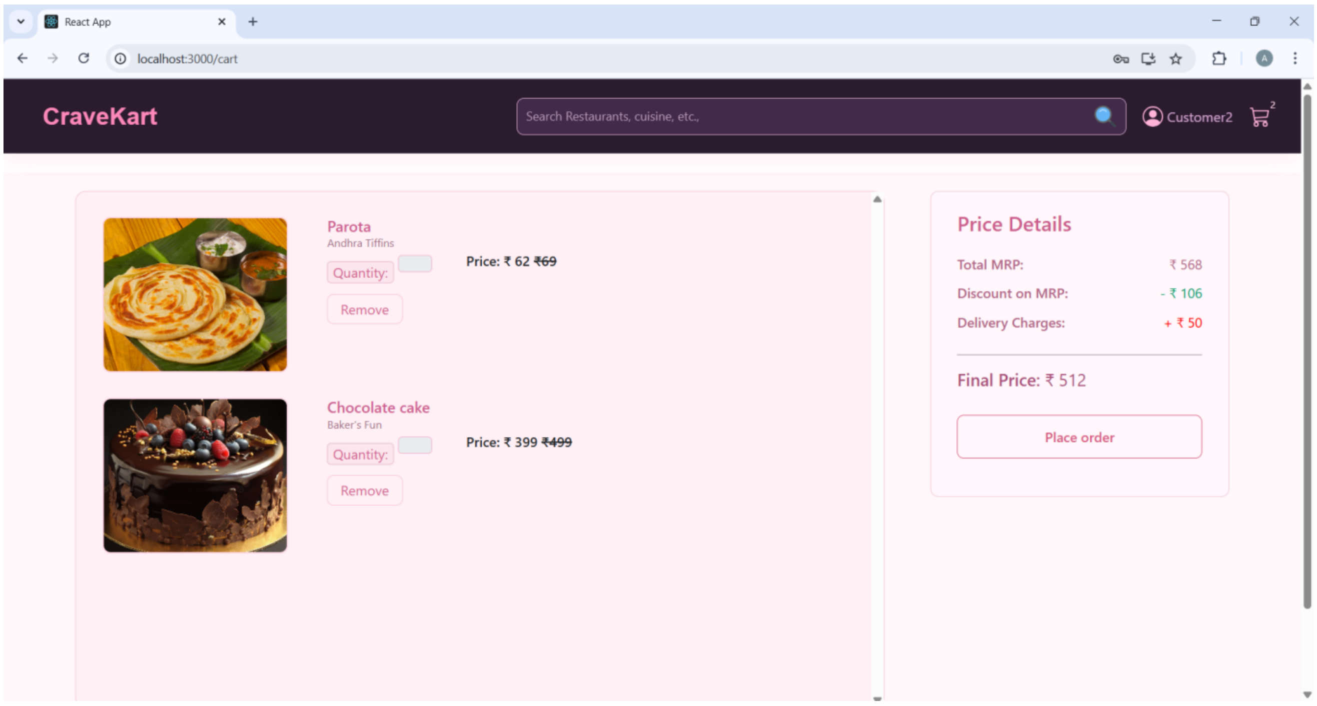
Vanilla Ice Cream

Ice Berg

Quantity: 1 Total Price: ₹ 57 ~~₹ 59~~ Payment mode: cod

Ordered on: 2025-06-27 Time: 10:32 status: delivered

- **Cart**


React App

localhost:3000/cart

CraveKart

Search Restaurants, cuisine, etc.,

Customer2

Parota

Andhra Tiffins

Quantity:

Remove

Price: ₹ 62 ~~₹69~~

Chocolate cake

Baker's Fun

Quantity:

Remove

Price: ₹ 399 ~~₹499~~

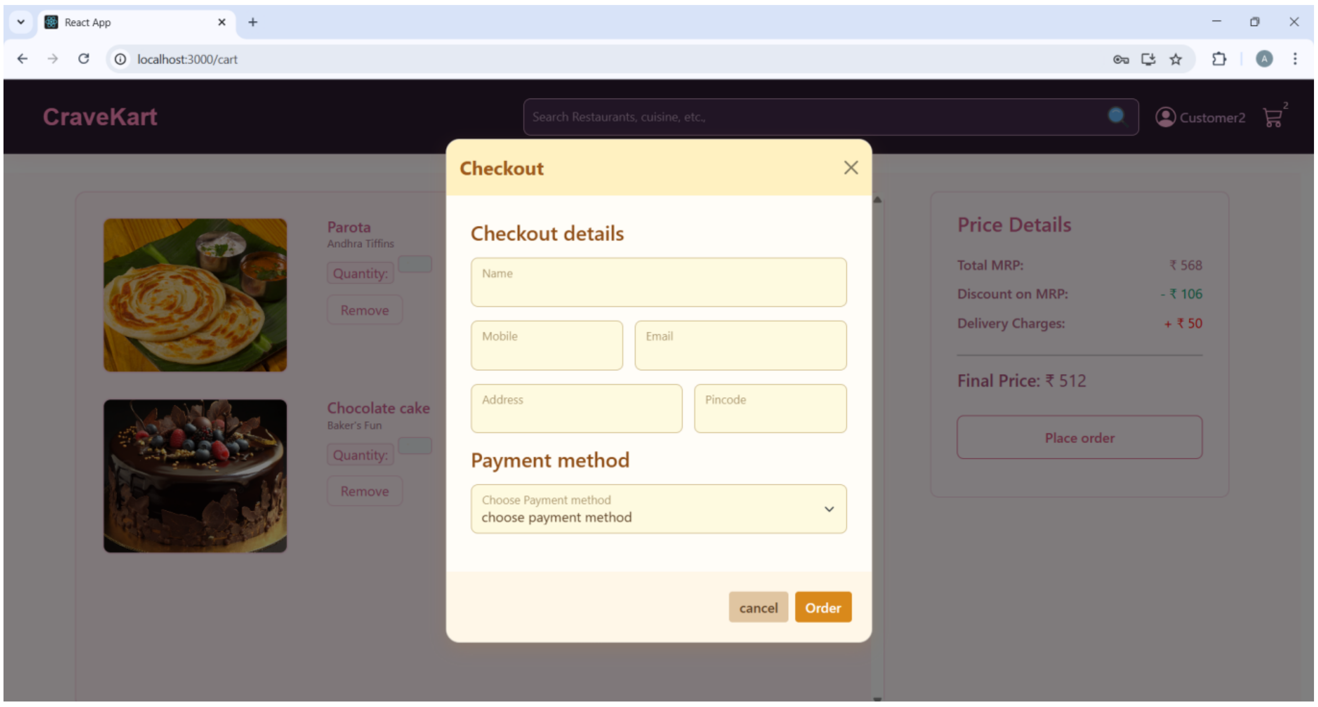
Price Details

Total MRP:	₹ 568
Discount on MRP:	- ₹ 106
Delivery Charges:	+ ₹ 50

Final Price: ₹ 512

Place order

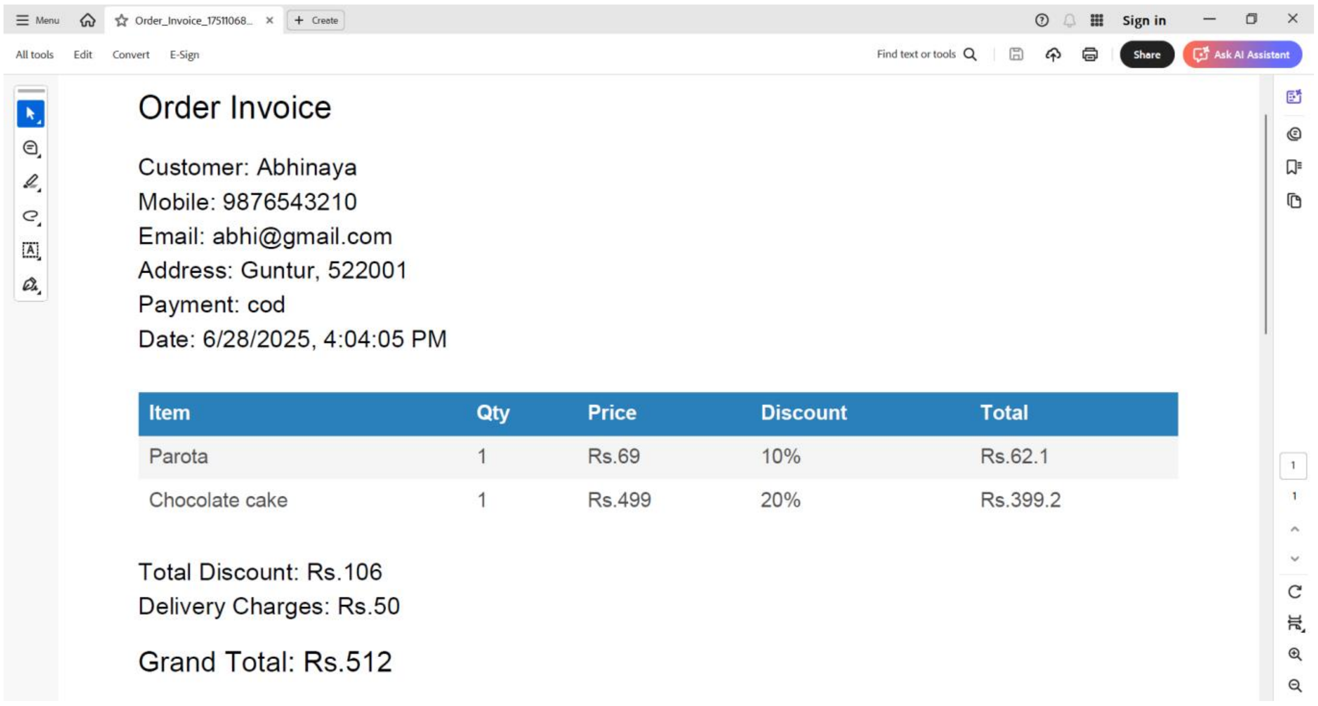
- **Checkout details**



Total MRP:	₹ 568
Discount on MRP:	- ₹ 106
Delivery Charges:	+ ₹ 50

Final Price: ₹ 512

- **Invoice Download**



Order Invoice

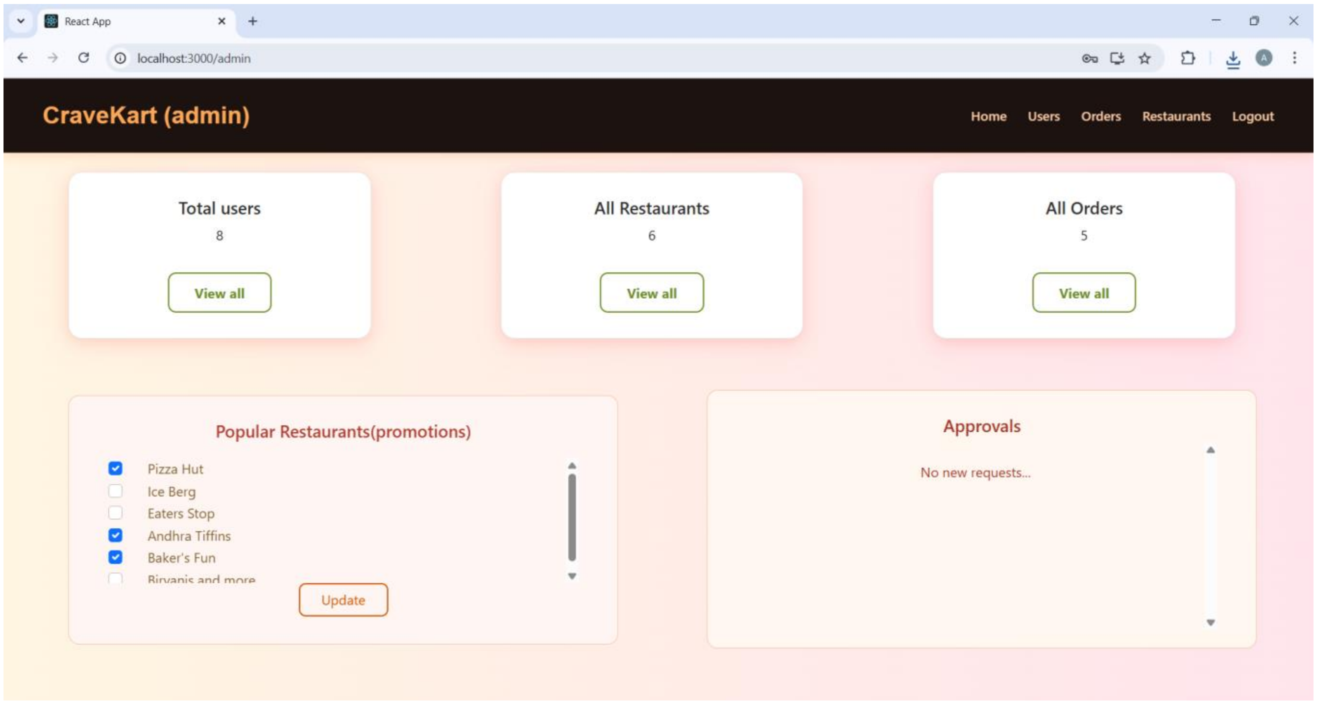
Customer: Abhinaya
Mobile: 9876543210
Email: abhi@gmail.com
Address: Guntur, 522001
Payment: cod
Date: 6/28/2025, 4:04:05 PM

Item	Qty	Price	Discount	Total
Parota	1	Rs.69	10%	Rs.62.1
Chocolate cake	1	Rs.499	20%	Rs.399.2

Total Discount: Rs.106
Delivery Charges: Rs.50

Grand Total: Rs.512

- **Admin dashboard**


React App

localhost:3000/admin

CraveKart (admin)

Home Users Orders Restaurants Logout

Total users

8

View all

All Restaurants

6

View all

All Orders

5

View all

Popular Restaurants(promotions)

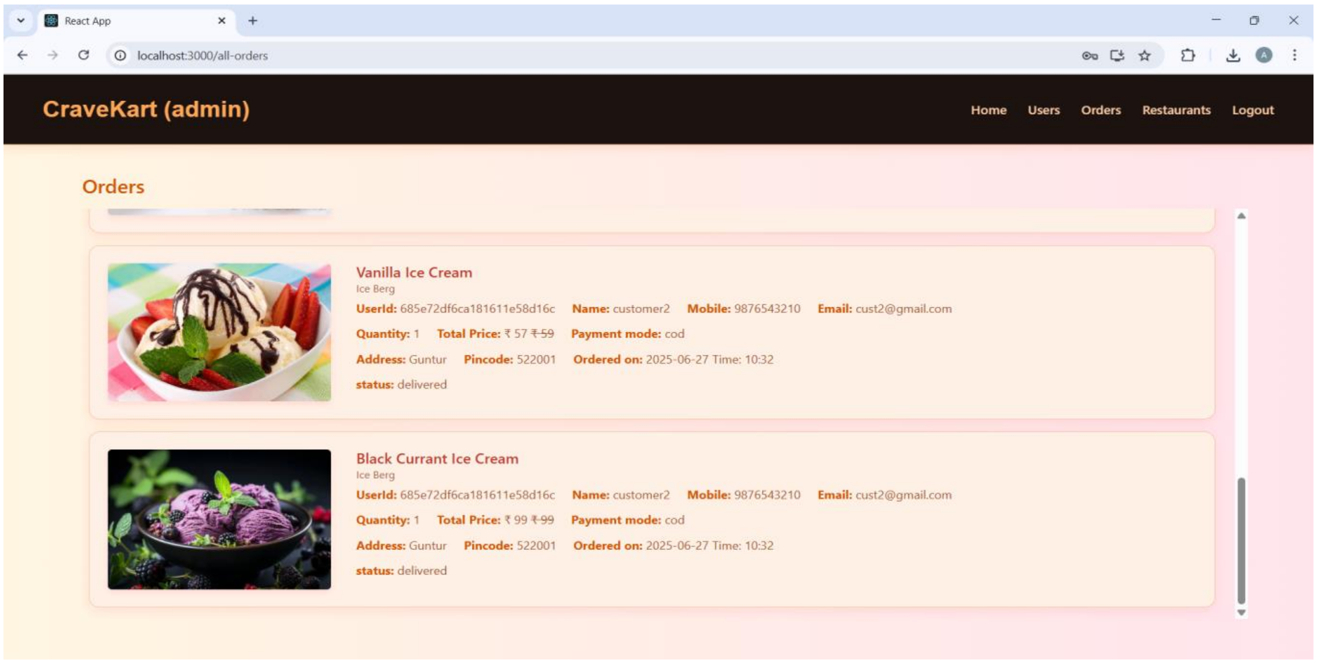
- [x] Pizza Hut
- [ ] Ice Berg
- [ ] Eaters Stop
- [x] Andhra Tiffins
- [x] Baker's Fun
- [ ] Biryanis and more

Update

Approvals

No new requests...

- **All Orders**


React App

localhost:3000/all-orders

CraveKart (admin)

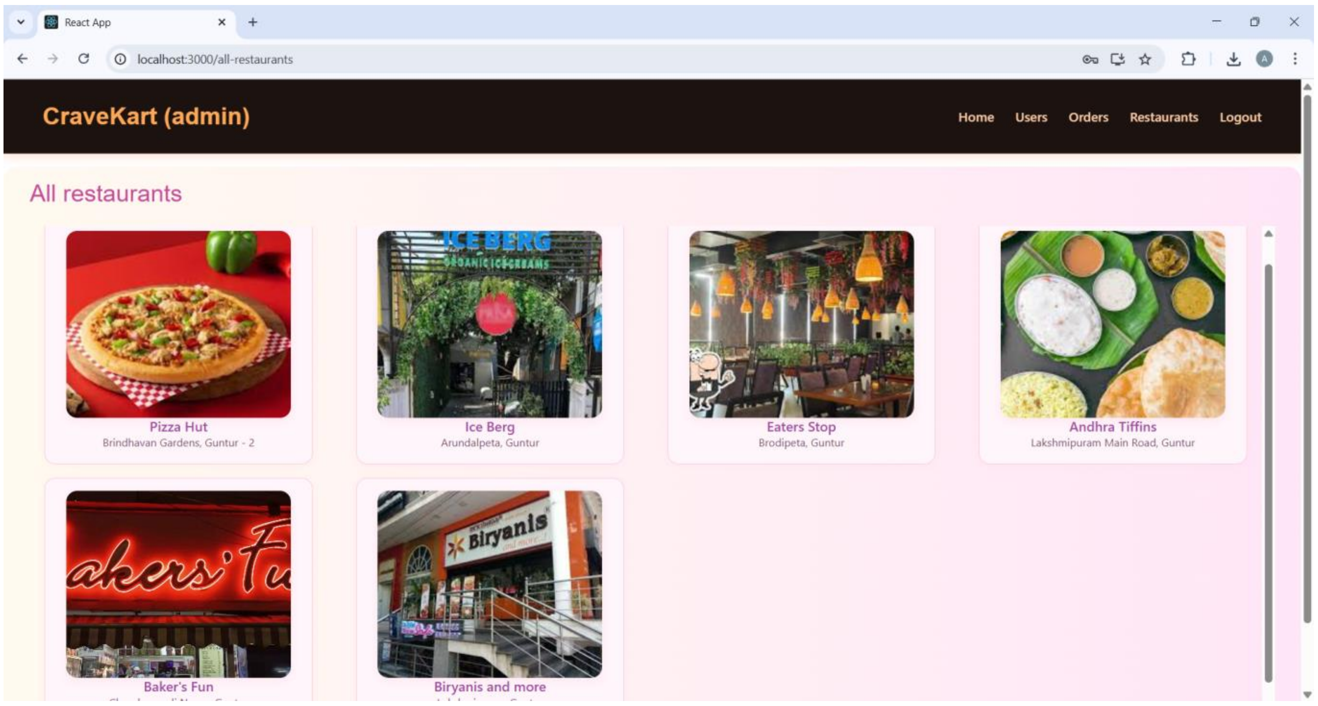
Home Users Orders Restaurants Logout

Orders

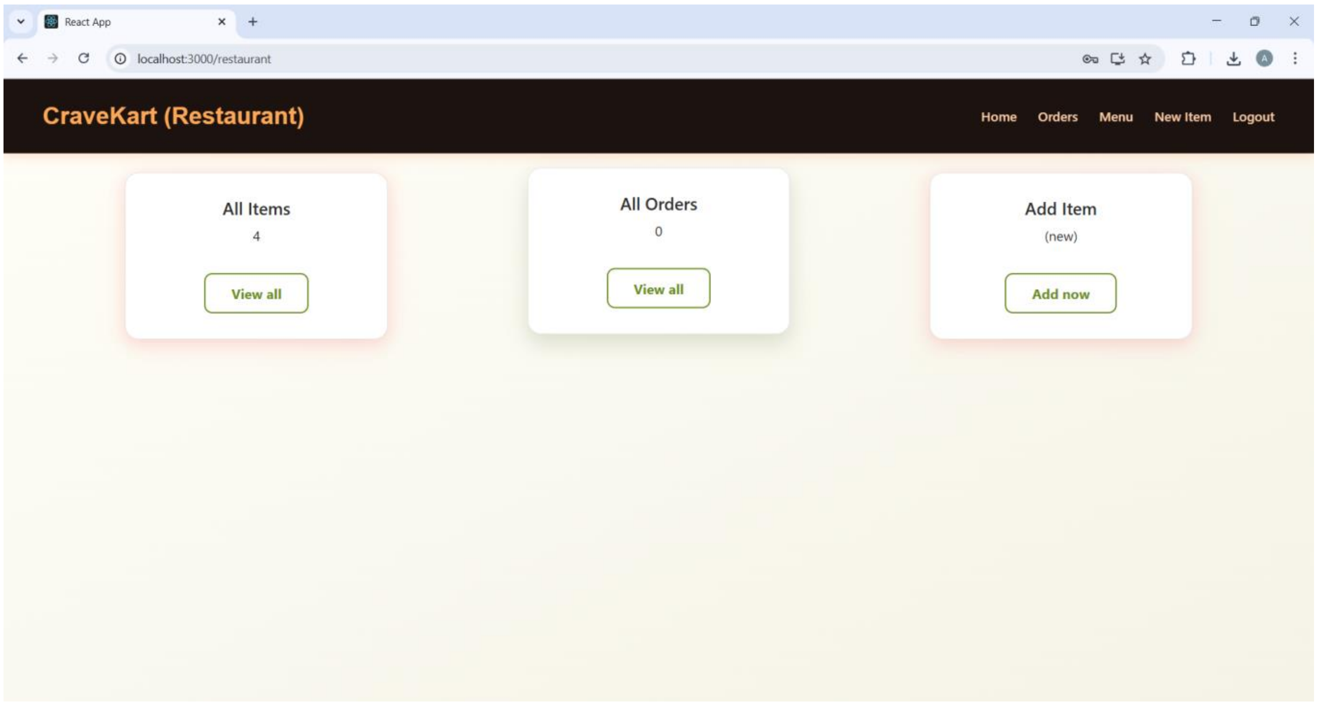
Vanilla Ice Cream
Ice Berg
UserId: 685e72df6ca181611e58d16c **Name:** customer2 **Mobile:** 9876543210 **Email:** cust2@gmail.com
Quantity: 1 **Total Price:** ₹ 57 ~~₹ 59~~ **Payment mode:** cod
Address: Guntur **Pincode:** 522001 **Ordered on:** 2025-06-27 Time: 10:32
status: delivered

Black Currant Ice Cream
Ice Berg
UserId: 685e72df6ca181611e58d16c **Name:** customer2 **Mobile:** 9876543210 **Email:** cust2@gmail.com
Quantity: 1 **Total Price:** ₹ 99 ~~₹ 99~~ **Payment mode:** cod
Address: Guntur **Pincode:** 522001 **Ordered on:** 2025-06-27 Time: 10:32
status: delivered

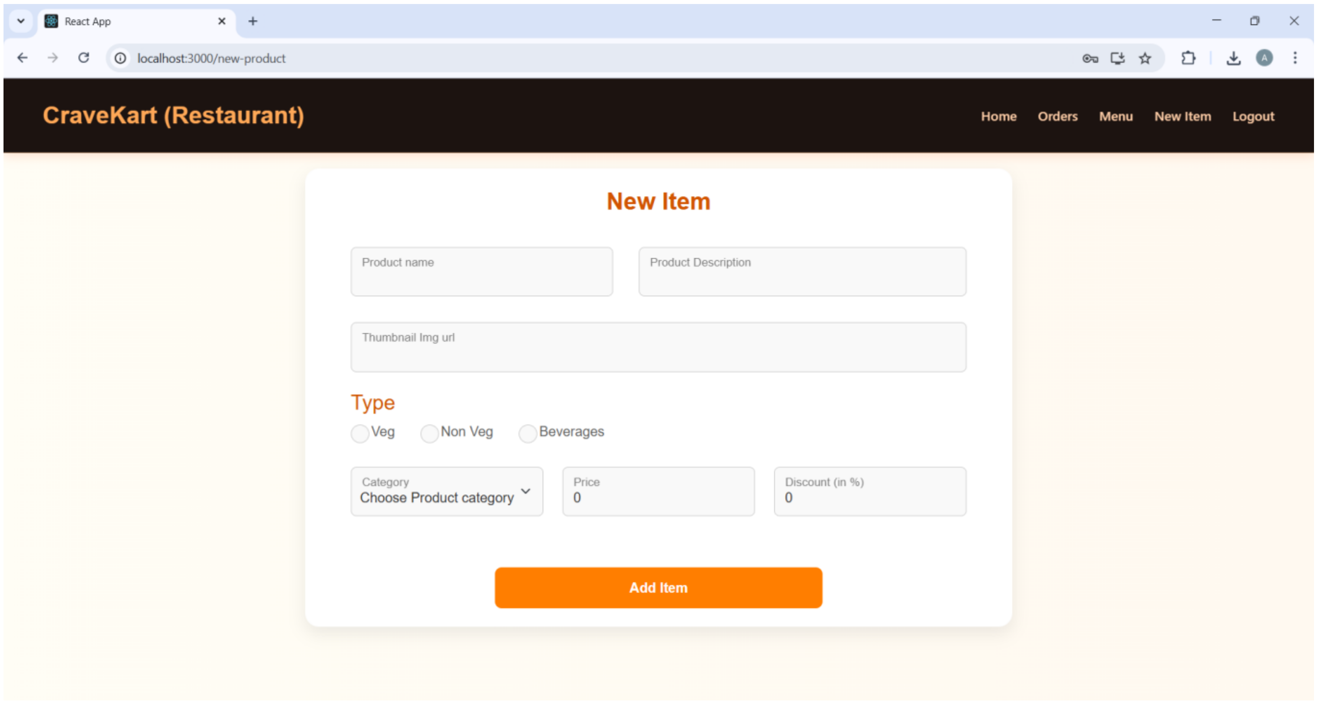
- **All restaurants**



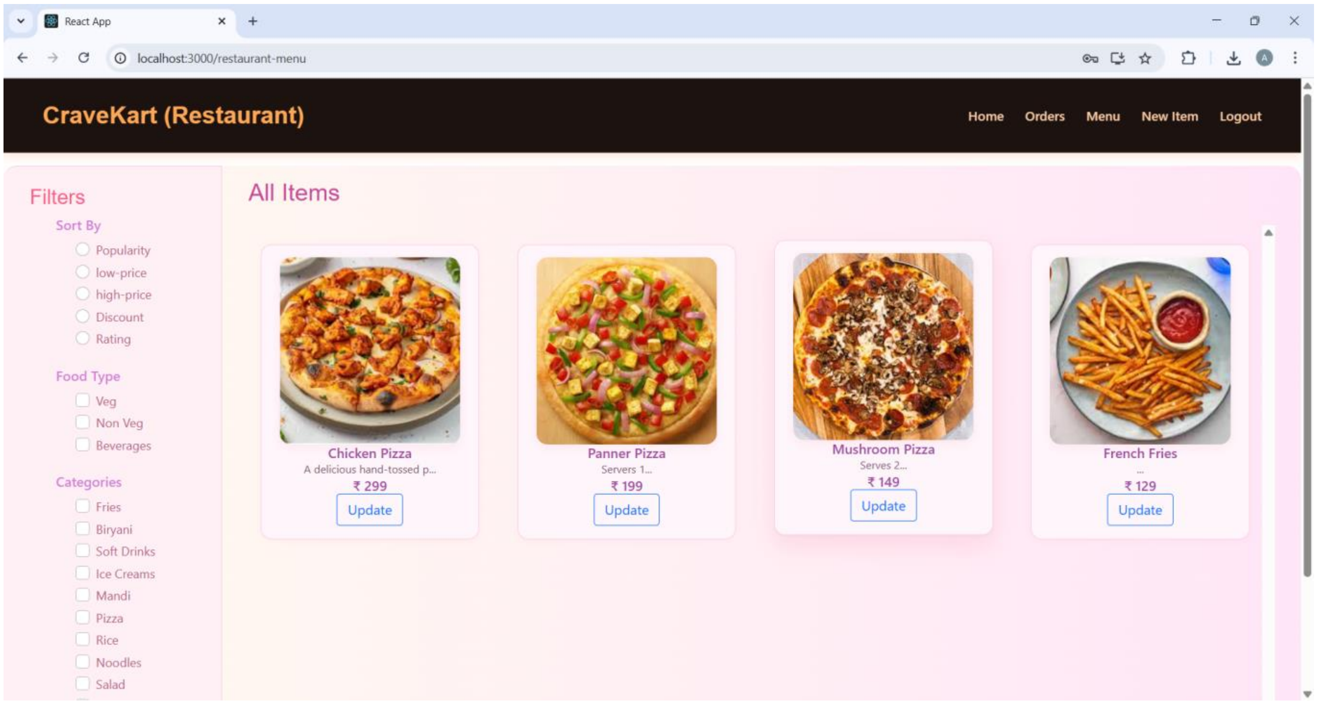
- **Restaurant Dashboard**



- **New Item**



All Items



For any further doubts or help, please consider the GitHub repo, https://github.com/KalisettyAbhi234/OrderOnTheGo-Your-On-Demand-Food-Ordering-Solution/tree/main

The demo of the app is available at:

https://drive.google.com/file/d/1aT5kXVJX83ht5Yap2FwRCRcOqnS3_Mxb/view?usp=sharing

12. KNOWN ISSUES

Below are the currently identified bugs and limitations in the application that are yet to be resolved or are under monitoring.

Customer Side

1. Invoice Download Naming

- Issue: Invoices are downloaded with random filenames.
- Impact: May confuse users managing multiple orders.
- Status: Improvement planned (add OrderID or date in filename).

2. No UPI or Online Payment Integration

- Issue: Only Cash on Delivery (COD) is supported.
- Impact: May reduce convenience for online-first users.
- Status: Planned for next release.

Restaurant Side

1. Image Upload Without Preview

- Issue: No real-time image preview when adding food items.
- Impact: Can lead to incorrect uploads.
- Status: Under development.

2. Order Status Update Not Real-Time

- Issue: Status changes are not auto-refreshed.
- Impact: Restaurant must manually reload to see new orders.
- Status: To be improved using WebSockets or polling.

Admin Side

1. Limited Analytics

- Issue: Admin dashboard lacks graphs/stats of user and order data.
- Impact: Makes system monitoring less effective.
- Status: Feature planned in future roadmap.

13. FUTURE ENHANCEMENTS

These features are planned for future development to improve user experience, functionality, and scalability.

1. Online Payment Integration

Integrate UPI, debit/credit card, and wallet payment gateways (e.g., Razorpay, Stripe)
Support for multiple payment methods instead of only COD

2. Real-Time Order Notifications

Use WebSockets or Firebase to notify users and restaurants about order status updates instantly
Improve responsiveness for both customers and restaurant owners

3. Live Order Tracking

Enable customers to track delivery agents and estimated delivery time
Integrate Google Maps API for live tracking interface

4. Mobile App Version

Develop native or hybrid Android/iOS app using React Native or Flutter
Push notifications for offers and order updates

5. Admin Analytics Dashboard

Add visual charts and reports for admins to monitor:
Total users, orders, revenue trends
Top-performing restaurants and food items

6. Reviews & Ratings System

Allow users to rate food items and restaurants
Display average ratings for better decision making

7. Multi-language Support

Support regional and international languages
Expand accessibility and localization

8. Automated Testing

Integrate unit testing (Jest) and E2E testing (Cypress)
Enable CI/CD pipelines for seamless updates