

PROJECTDOCUMENT

OrderOnTheGo:

Your On-Demand Food Ordering Solution

Submitted by

- 1.Sodasani Nagarjuna
- 2.Potnuri VSDL Harika
- 3 .Sujal Kumar
- 4.M Manasa

Team ID : LTVIP2025TMID42867

Department of Computer Science and Engineering Aditya College of Engineering and Technology,
Surampalem, Kakinada District, Andhra Pradesh – 533437, India

Affiliated to: Jawaharlal Nehru Technological University, Kakinada (JNTU -K)

Submitted to



Academic Year
2022 - 2026

Page 1

ABSTRACT

The ssp Online Food Ordering System is designed to streamline and simplify the process of ordering food for both customers and restaurant staff. This system provides a userfriendly interface that displays an up-to-date menu with all available items, allowing customers to conveniently select multiple food items to place an order. Before completing the transaction, customers can review their order details, and upon confirmation, the system notifies them and records the order in real-time.

The order is then added to a processing queue and stored in the database, enabling restaurant staff to monitor and fulfill orders efficiently. This real-time synchronization enhances the workflow within the restaurant, improving order accuracy and service speed.

Primarily developed for the food delivery industry, the system supports restaurants and hotels in boosting their online food ordering capabilities. With just a few clicks, customers can choose their meals, and the system ensures timely and accurate delivery to their specified location. Restaurant staff benefit from an intuitive, graphical interface that simplifies order tracking and management, ultimately improving customer satisfaction and operational efficiency.

INTRODUCTION

The Online Food Ordering System can be defined as a simple and convenient way for customers to purchase food online without physically visiting a restaurant. This system is powered by the internet, which connects restaurants or food service providers directly with customers. Through this system, customers can access the restaurant's website, browse available food items, and select and purchase the items of their choice. The selected food is then delivered to the customer's location at their preferred time by a delivery person.

Payments for online orders can be made using various methods, including debit or credit cards, cash on delivery, or digital wallets. This system offers a safe and secure platform for food ordering and is rapidly transforming the operations of the food service industry.

This project proposes an Online Food Ordering System designed specifically for fast food restaurants, take-out services, or college cafeterias. However, it can be adapted for any segment of the food delivery industry. The system automates the

entire order-taking process, reducing manual effort and improving efficiency for both customers and restaurant staff.

One of the key advantages of this project is its ability to significantly streamline the ordering process. When a customer visits the ordering webpage, they are presented with an interactive, real-time menu that displays all available food options, including dynamic pricing based on selected items. Once selections are made, items are added to the order, and the customer can view and modify their order at any time before final checkout. This provides instant visual confirmation and ensures a smooth, user-friendly experience.

MotiVAtion

The motivation for designing this application came because my family is involved in the fast food business and I personally do not like waiting for long in the store or to have to call store to

place an order especially during the peak lunch or dinner hours. Moreover, I value recent learning about the php Programming languages as well as seeing how powerful and dynamic they are when it comes to web designing and applications, whereas mysql database at the backend because I found them to be extremely useful while working on the technologies.

This system specifically e made for or following issues

5. Sometime payment issue is occurred.
6. Online food ordering system service now days increase your budget.
7. lack of a visual confirmation that the order was placed correctly.

Purpose r objectives And goals

The proposed system is developed to manage ordering activities in fast food restaurant. It helps

to record customer submitted orders. The system should cover the following functions in order to

support the restaurant's business process for achieving the objectives:

1. To allow the customer to make order, view order and make changes before submitting their order and allow them make payment through prepayment card or credit card or debit card.
2. To provide interface that allows promotion and menu.
3. To prevent interface that shows customers' orders detail to front-end and kitchen staffs for delivering customers' orders
4. Tools that generate reports that can be used for decision making
5. A tool that allows the management to modify the food information such as price, add a new menu and many others as well as tools for managing user, system menu and promotion records.

This will minimize the number of employees at the back of the counter.

The system will help to reduce labor cost involved.

The system will be less probable to make mistake, since it's a machine.

This will avoid long queues at the counter due to the speed of execution and number of optimum

screens to accommodate the maximum throughput.

The main objective of the Online Food Ordering System is to manage the details of Item Category, Food, Delivery Address, Order, Shopping Cart The purpose of the project is to build an application program to reduce the manual work for managing the Item Category, Food, Customer, Delivery Address

Objectives And Goals

1. To increase efficiency and improve services provided to the customers through better application of technology in daily operations.
2. To be able to stand out from competitors in the food service industry
3. To enable customers to order custom meals that aren't in the menu
4. To enable customers to have a visual confirmation that the order was placed correctly
5. To enable customers to know food ingredients before ordering
6. To reduce restaurant's food wastage
7. To ensure correct placement of orders through visual confirmation
8. Improve efficiency of restaurant's staff
9. Eliminate paper work and increase level of accuracy
10. Increase speed of service, sales volume and customer satisfaction
11. To increase efficiency by shortening the purchasing time and eliminating paper work like receipts through online transaction
12. To be able to stand out from competitors by automating daily operations which will give food service providers the opportunity to increase sales
13. To reduce restaurants food wastage and increasing efficiency of the restaurants staff by enabling the restaurants staff to know what food items the customers want in advance.
14. To increase customer satisfaction by speeding up food delivery
15. To reduce time wasting by eliminating long queues
16. More accuracy and easy order processing.
17. 24

Literature Survey

Various case studies have highlighted the problems faced While setting up a restaurant..
Some of the problems Found during the survey in the existing system are listed Below :

1. To place the orders customer visits the restaurant, Checks the menu items available in the restaurant, and chooses the items required, then places the order And then do the payment.
2. This method demands Manual work and time on the part of the customer.
3. When the customer wants to order over the phone, Customer is unable to see the physical copy of the Menu available in the restaurant, this also lacks the Verification that the order was placed for the appropriate menu items.
4. Every restaurant needs someone or the other to take order personally or over phone, to offer the Customer a rich experience and even to process the payment.

Project scope and limitation

Note-Refer old document format that are already sent you.

1. This system will help to customer and restaurants and administrator for the ordering process.
2. Easy to make ordering and hopefully can smoothen up the job of administrator and waiter.
3. This system produce a computerized system in defining the best solution in food delivery system.
4. Easy acces to any stage.
5. Lot of time is save.
6. Easy back up of data.

Limitations

1. Cost associated with backp storage to the system than the cost associate with maintaining on-site alone.
2. A potential for customer to fail to adapt to online ordering or tablesite checkout.

Project Perspective

The Online Food Ordering System (SB Foods) is a comprehensive, web-based application that allows users to conveniently browse menus, place food orders, and track deliveries. This system is designed for access via internet browsers on various devices like PCs, laptops, and smartphones. It ensures a smooth, secure, and efficient ordering experience for customers, while also offering powerful tools for restaurant staff and administrators.

System Model

The Online Food Ordering System (SB Foods) is a comprehensive, web-based application that allows users to conveniently browse menus, place food orders, and track deliveries. This system is designed for access via internet browsers on various devices like PCs, laptops, and smartphones. It ensures a smooth, secure, and efficient ordering experience for customers, while also offering powerful tools for restaurant staff and administrators.

The system is structured into three major logical components:

1. Web Ordering System:

Facilitates customers in placing orders and entering their delivery and payment details.

2. Menu Management:

Allows the restaurant admin to manage menu items, including additions, updates, and deletions.

3. Order Retrieval System:

Empowers the restaurant to process, monitor, and manage incoming orders in real-time.

Product Functions

1. Web Ordering System Module :

This module enables users to place orders and enter key details such as location, contact, and payment preferences. Key submodules include:

- Home Page
- Meal Plan Page
- My Cart Page
- Login/Signup Page

2. Menu Management Module :

Used by admins to manage food item listings, which are visible to users:

- Add/Edit/Delete Food Items
- Set Food Size Options
- Configure Pricing
- Upload Food Images
- Add Descriptions

3. Order Retrieval Module :

Allows restaurant staff to handle customer orders:

- View Order Plans
- Track Order Quantity
- Manage Delivery Status

User Scenario: Late-Night Craving Resolution

Meet Lisa, a college student working late on an assignment. It's midnight, and she realizes she skipped dinner. Cooking is not an option, and going out isn't safe or convenient.

Using SSP Foods, Lisa

1. Opens the app and goes to the "Late-Night Delivery" section.
2. Browses menus and finds her favorite diner still accepting orders.
3. Chooses chicken noodle soup and garlic bread.
4. Adds items to her cart, selects delivery address, and payment method.
5. Reviews order details and confirms with a tap.
6. Gets instant confirmation and estimated delivery time.
7. Receives her hot, delicious meal quickly and gets back to work.

This showcases how SSP Foods addresses user needs seamlessly even at odd hours—providing convenience, speed, and satisfaction without compromise.

Technical Architecture Overview

Frontend		Backend		Database
- User Interface	<---->	- RESTful APIs	<-->	- Users Collection
- User Auth		- Admin Auth		- Orders Collection
- Product Pages		- Admin Dashboard		- Products Menu
- Cart Management		- Order Processing		- Cart Items
- User Profile				- Admin Records

Key Components:

- Frontend: Built with responsive UI for user interaction, login, cart, and admin control.
- Backend: Node.js/Express or similar stack serving APIs for users, products, orders, and admin.
- Database: MongoDB or SQL storing persistent data for orders, users, and menu items.

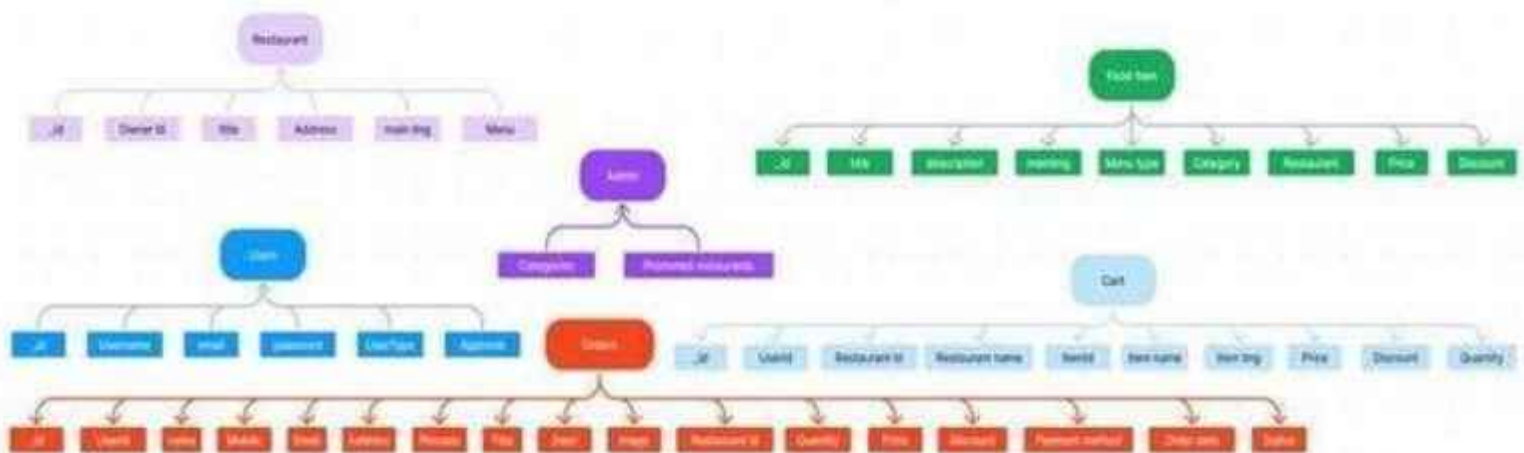
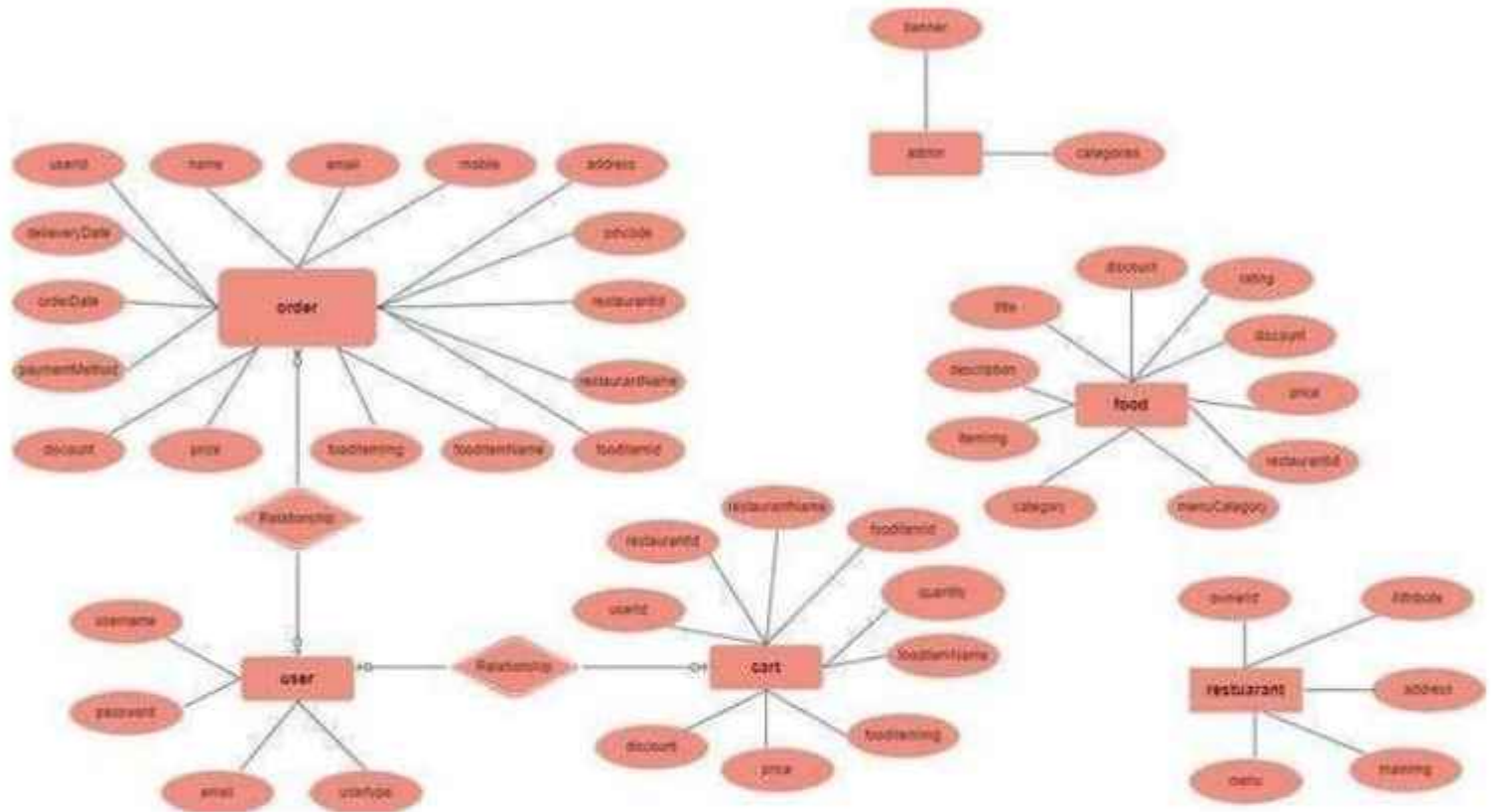


ER-Diagram OVERVIEW

System Design :

Design Constraints-Entity Relationship Diagram

The SSP Foods ER-Diagram visualizes the key entities and their relationships within the food ordering e-commerce platform. It outlines how users, restaurants, products, carts, and orders are structured and interact with each other in the system.



Entity Descriptions

- **User:**

Represents individuals registered on the platform. Each user has a unique ID and personal information like name, email, and address.

- **Restaurant:**

Stores details of restaurants listed on the platform. Includes restaurant name, contact, cuisine type, and availability.

- **Admin:**

Represents platform administrators who manage promoted restaurants, food categories, and have access to system-level controls.

- **Products:**

Contains all food items available for order. Each product is associated with a restaurant and includes name, description, price, size, image, and category.

- **Cart:**

Stores items added by users before placing an order. Each cart entry links products with the corresponding user ID and quantity.

- **Orders:**

Maintains records of all user purchases. Each order is linked to a user and contains order details like items, quantity, status, delivery address, and payment method.

Key Features of SSP Foods

1. Comprehensive Product Catalog

SSP Foods offers a wide variety of dishes from multiple restaurants. Users can explore food items through an organized catalog with:

- Detailed food descriptions
- Customer reviews
- Prices and discounts
- Availability by category or restaurant



2. Order Details Page

After choosing their meals, users are guided to a dedicated Order Details Page where they can:

- Enter shipping/delivery address
- Select payment method (e.g., UPI, card, cash)
- Add notes or instructions (e.g., spice level, no onions)

3. Secure & Efficient Checkout

SSP Foods prioritizes a secure and smooth checkout experience:

- SSL-secured transactions
- Fast order processing
- Minimal steps to complete payment

4. Order Confirmation & Summary

Once the order is placed:

- A confirmation notification is immediately sent to the user
- Users are directed to an Order Summary Page showing:
 - Shipping address
 - Order contents
 - Estimated delivery time
 - Payment method used

Restaurant Dashboard Features

SSP Foods also offers a powerful backend dashboard for restaurants, enabling them to:

- Manage Products: Add, update, and remove menu items
- Track Orders: View order history and real-time order data
- Monitor Customers: Track regular customers and popular dishes
- Access Order Details: See specific order instructions and delivery data

Summary

SSP Foods is designed to enhance the food ordering experience for both users and restaurant partners. It combines a rich product catalog, an intuitive user interface, secure transactions, and a feature-rich admin dashboard. Whether you're placing a quick midnight snack order or managing a full restaurant menu, SSP Foods delivers efficiency and convenience at every step.

Prerequisites & Setup Guide

REQUIRED TOOLS & TECHNOLOGIES

1. Node.js and npm

Node.js enables JavaScript to run on the server side. npm manages project dependencies.

Download: <https://nodejs.org/en/download>

2. MongoDB

A NoSQL database to store user, order, and product data.

Download: <https://www.mongodb.com/try/download/community>

3. Express.js

A Node.js web framework for building APIs.

Install: `npm install express`

4. React.js

Frontend library for building dynamic user interfaces.

Guide: <https://reactjs.org/docs/create-a-new-react-app.html>

5. HTML, CSS, and JavaScript

Core technologies for building the web interface.

6. Mongoose

MongoDB ODM for Node.js.

Guide: <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

7. Git

Version control system.

Download: <https://git-scm.com/downloads>

8. Code Editor (IDE)

Examples: -

Visual Studio Code: <https://code.visualstudio.com/download>

Sublime Text: <https://www.sublimetext.com/download>

WebStorm: <https://www.jetbrains.com/webstorm/download>

Project Setup Instructions

Step 1: Clone the Repository

git clone <https://github.com/harsha-varadhan-reddy-07/Food-Ordering-App-MERN>

Step 2: Navigate to Project Directory

cd Food-Ordering-App-MERN

Step 3: Install Dependencies

npm install

Step 4: Start the Development Server

npm run dev OR npm run start

Access the App:

Visit <http://localhost:3000> in your browser.

You should see the SB Foods homepage.

ApplicationFlow

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

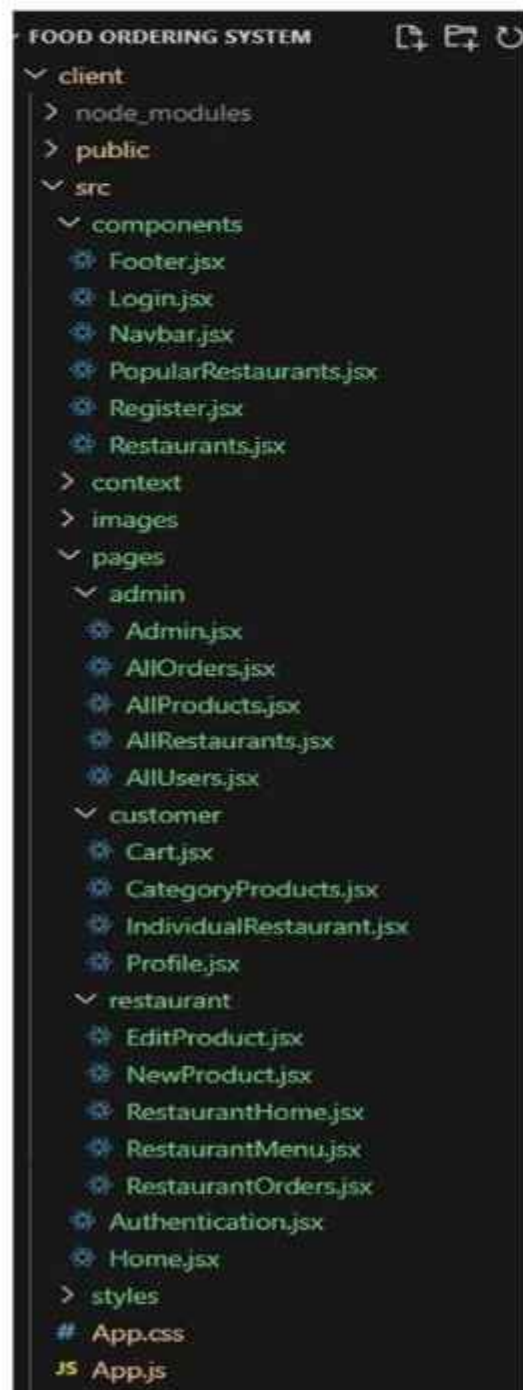
- Restaurants start by authenticating with their credentials.
- They need to get approval from the admin to start listing the products.
- They can add/edit the food items.

3. Admin Flow

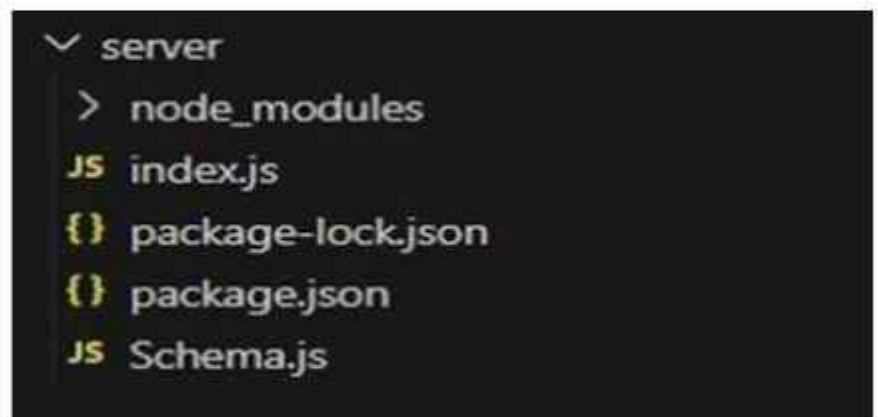
- Admins start by logging in with their credentials.
- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, products, orders, etc.

Project Structure

client structure



server 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.

Project Setup And Configuration

Install required tools and software:

- Node.js.

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

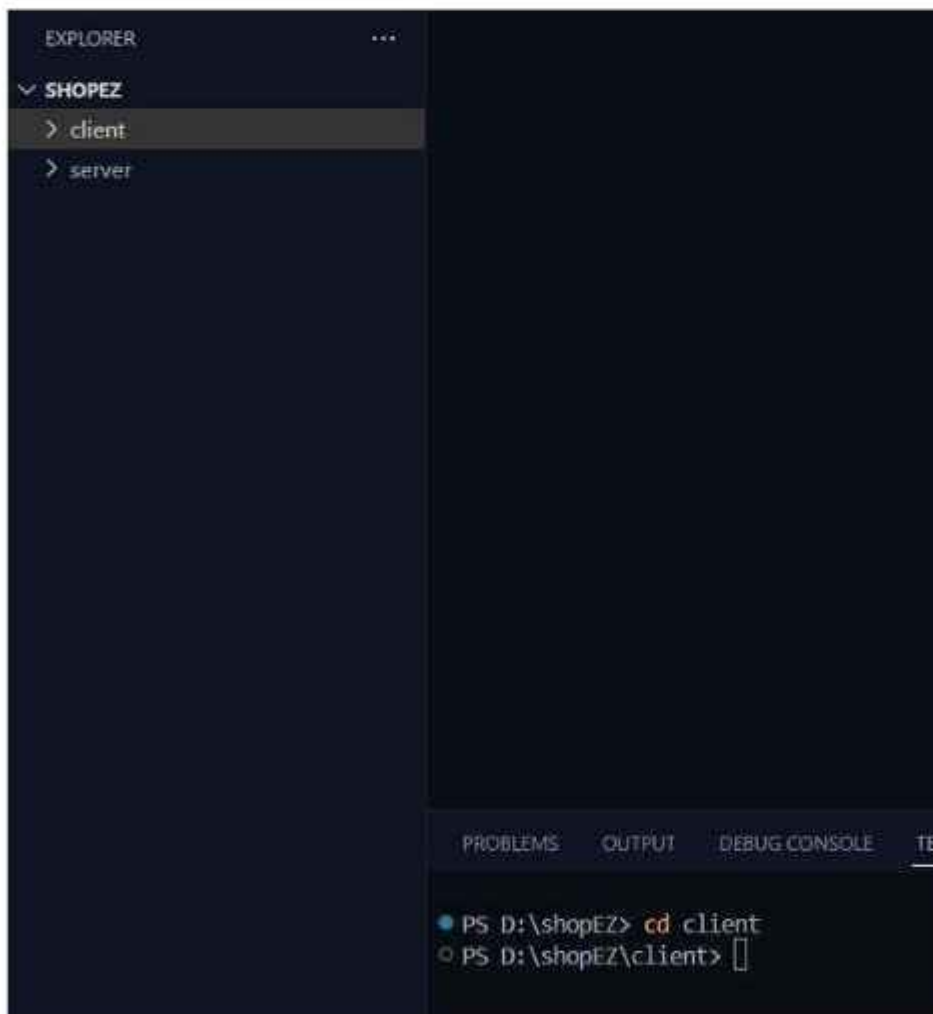
- Git.

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

Create project folders and files:

- Client folders.
- Server folders

Referral Image:



This section outlines the key steps and tools required to set up the SSP Foods Full-Stack Application on your local development environment.

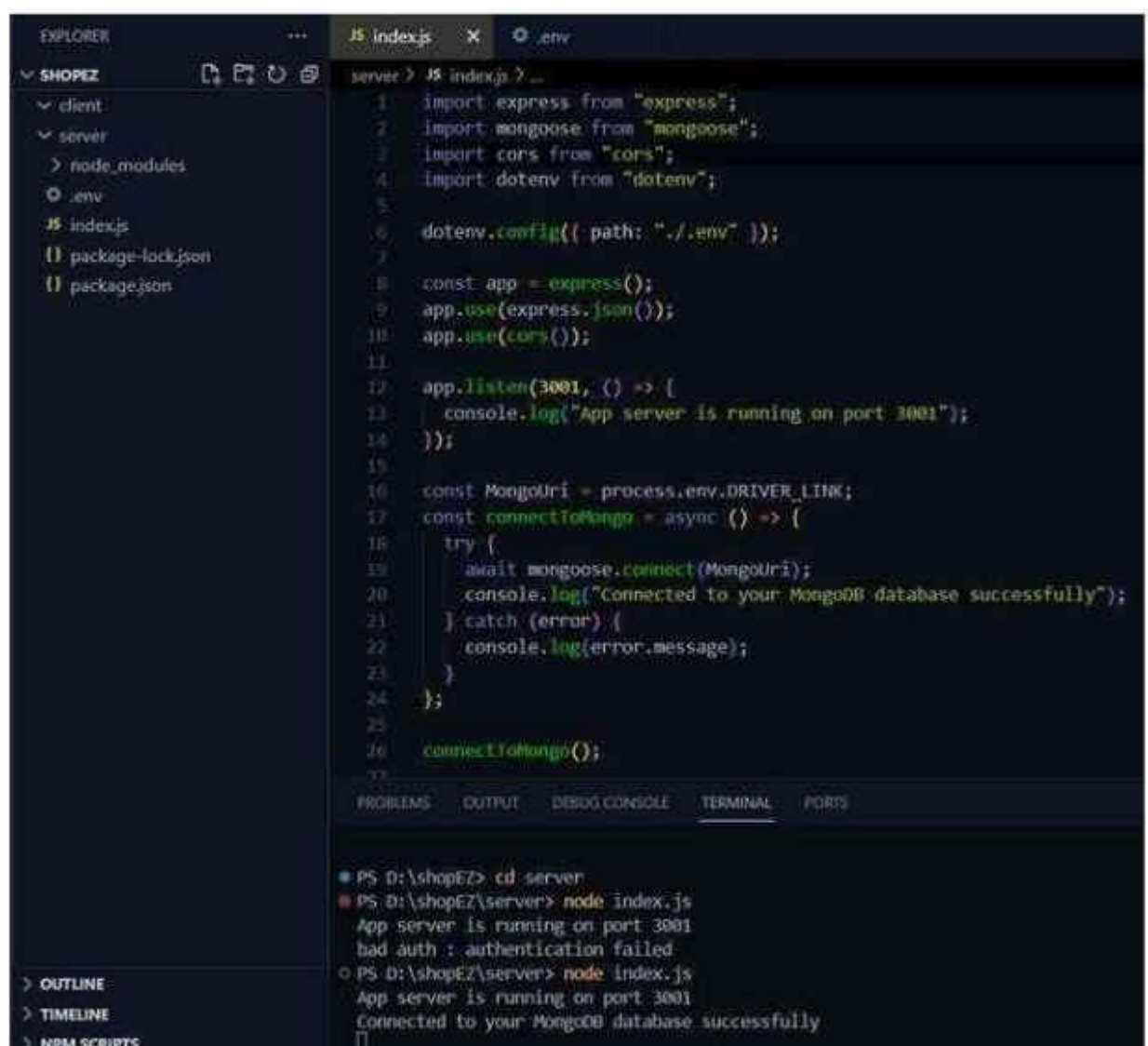
Database Development

Create database in cloud

- Install Mongoose.
- Create database connection.

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

Reference Image:



```
server > JS index.js > ...
1  import express from "express";
2  import mongoose from "mongoose";
3  import cors from "cors";
4  import dotenv from "dotenv";
5
6  dotenv.config({ path: ".env" });
7
8  const app = express();
9  app.use(express.json());
10 app.use(cors());
11
12 app.listen(3001, () => {
13   console.log("App server is running on port 3001");
14 });
15
16 const MongoUri = process.env.DRIVER_LINK;
17 const connectToMongo = async () => {
18   try {
19     await mongoose.connect(MongoUri);
20     console.log("Connected to your MongoDB database successfully");
21   } catch (error) {
22     console.log(error.message);
23   }
24 };
25
26 connectToMongo();
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• PS D:\shopEZ> cd server
• PS D:\shopEZ\server> node index.js
App server is running on port 3001
bad auth : authentication failed
○ PS D:\shopEZ\server> node index.js
App server is running on port 3001
Connected to your MongoDB database successfully

```


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

```
JS Schemajs X
server > JS Schemajs > [0] orderSchema
1  import mongoose from "mongoose";
2
3  const userSchema = new mongoose.Schema({
4    username: {type: String},
5    password: {type: String},
6    email: {type: String},
7    usertype: {type: String},
8    approval: {type: String}
9  });
10
11 const adminSchema = new mongoose.Schema({
12   categories: {type: Array},
13   promotedRestaurants: []
14 });
15
16 const restaurantSchema = new mongoose.Schema({
17   ownerId: {type: String},
18   title: {type: String},
19   address: {type: String},
20   mainImg: {type: String},
21   menu: {type: Array, default: []}
22 });
23
24 const foodItemSchema = new mongoose.Schema({
25   title: {type: String},
26   description: {type: String},
27   itemImg: {type: String},
28   category: {type: String}, //veg or non-veg or beverage
29   menuCategory: {type: String},
30   restaurantId: {type: String},
31   price: {type: Number},
32   discount: {type: Number},
33   rating: {type: Number}
34 });
35
```

This section outlines the setup, connection, and schema design for the MongoDB database used in the SSP Foods full-stack food ordering application.

Think of the SSP Foods App as a busy digital restaurant. Behind the beautiful menu and seamless ordering experience lies a smart kitchen — the MongoDB Database — quietly making everything work.

This code defines two important Mongoose schemas for managing Orders and Carts in your food ordering application using MongoDB.

```
# Schema.js X
server> # Schema.js > ...
36 const orderSchema = new mongoose.Schema({
37   userId: {type: String},
38   name: {type: String},
39   email: {type: String},
40   mobile: {type: String},
41   address: {type: String},
42   pincode: {type: String},
43   restaurantId: {type: String},
44   restaurantName: {type: String},
45   foodItemId: {type: String},
46   foodItemName: {type: String},
47   foodItemImg: {type: String},
48   quantity: {type: Number},
49   price: {type: Number},
50   discount: {type: Number},
51   paymentMethod: {type: String},
52   orderDate: {type: String},
53   orderStatus: {type: String, default: 'order placed'}
54 })
55
56 const cartSchema = new mongoose.Schema({
57   userId: {type: String},
58   restaurantId: {type: String},
59   restaurantName: {type: String},
60   foodItemId: {type: String},
61   foodItemName: {type: String},
62   foodItemImg: {type: String},
63   quantity: {type: Number},
64   price: {type: Number},
65   discount: {type: Number}
66 })
67
68 export const User = mongoose.model('users', userSchema);
69 export const Admin = mongoose.model('admin', adminSchema);
70 export const Restaurant = mongoose.model('restaurant', restaurantSchema);
71 export const FoodItem = mongoose.model('foodItem', foodItemSchema);
72 export const Orders = mongoose.model('orders', orderSchema);
73 export const Cart = mongoose.model('cart', cartSchema);
74
```

This block exports Mongoose models so that other files (like routes or controllers) can use them

```
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);
```

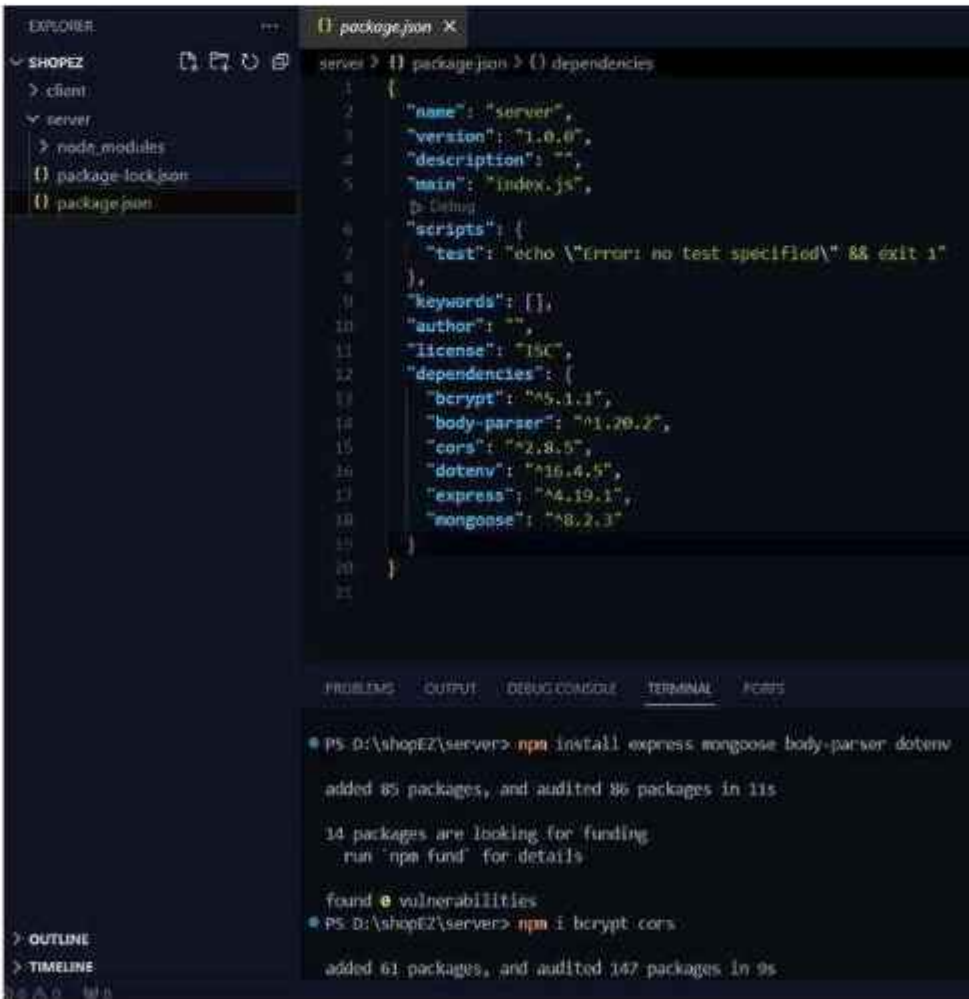
These models are essential for querying and manipulating your MongoDB collections.

Backend Development

1. Initialize Project:

- Create a new directory:
`mkdir sb-foods-app && cd sb-foods-app`
- Initialize with npm:
`npm init -y`
- Install dependencies:
`npm install express mongoose dotenv cors body-parser`

Reference Image:



The screenshot shows a Visual Studio Code editor with a project named 'SHOPEZ'. The Explorer sidebar on the left shows the project structure: 'client' and 'server' folders, with 'node_modules', 'package-lock.json', and 'package.json' files inside 'server'. The main editor area displays the 'package.json' file for the 'server' project. The file contains the following JSON:

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.1",
    "mongoose": "^8.2.3"
  }
}
```

Below the editor, the TERMINAL panel shows the output of the following commands:

```
PS D:\shopEZ\server> npm install express mongoose body-parser dotenv
added 85 packages, and audited 86 packages in 11s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\shopEZ\server> npm i bcrypt cors
added 61 packages, and audited 147 packages in 9s
```


2. Setup Express Server:

Create index.js in root

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const bodyParser = require("body-parser");
require("dotenv").config();

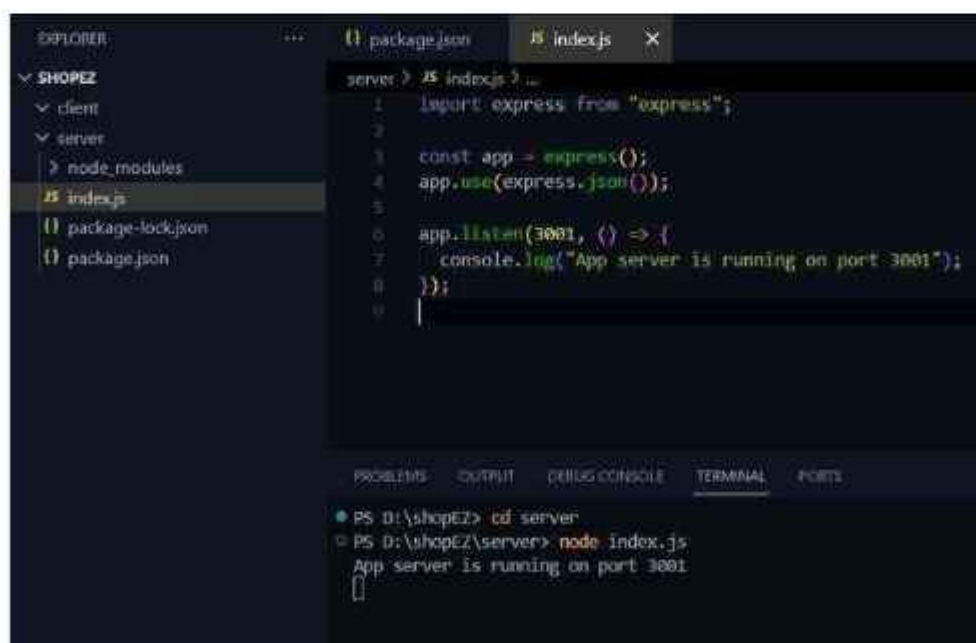
const app = express();
const PORT = process.env.PORT || 5000;

app.use(cors());
app.use(bodyParser.json());

app.get("/", (req, res) => {
  { res.send("Server is running");
});

app.listen(PORT, () => {
  console.log(`Server started on port ${PORT}`);
});
```

Reference Image:



3. Database Configuration:

- Use MongoDB Atlas or local MongoDB Compass.
- Set .env variables:

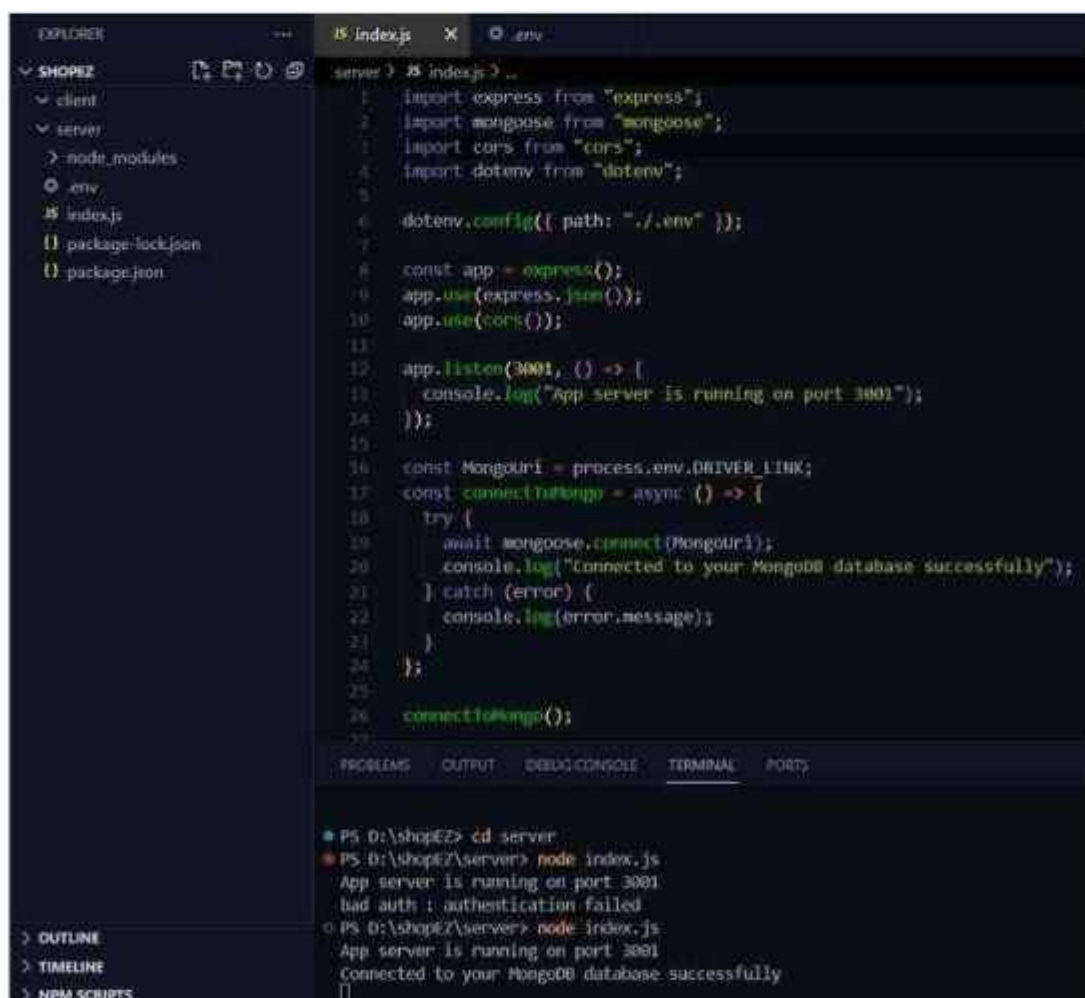
MONGO_URI=your_mongodb_connection_string

- Connect to DB:

```
mongoose.connect(process.env.MONGO_URI,
  { useNewUrlParser: true,
    useUnifiedTopology: true,
  })
.then(() => console.log("MongoDB Connected"))
.catch((err) => console.error(err));
```

Reference Video: [Google Drive MongoDB Setup](#)

Reference Article: [MongoDB Atlas Guide](#)



The screenshot shows a Visual Studio Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'SHOPEZ' with a 'server' folder containing 'index.js', 'package-lock.json', and 'package.json'. The code editor shows the content of 'index.js', which includes imports for express, mongoose, cors, and dotenv, followed by server setup and database connection logic. The terminal at the bottom shows the command prompt output for running the server.

```
server > .\index.js > ...
1  import express from "express";
2  import mongoose from "mongoose";
3  import cors from "cors";
4  import dotenv from "dotenv";
5
6  dotenv.config({ path: "../.env" });
7
8  const app = express();
9  app.use(express.json());
10 app.use(cors());
11
12 app.listen(3001, () => {
13   console.log("App server is running on port 3001");
14 });
15
16 const MongooseUri = process.env.MONGO_URI;
17 const connectToMongo = async () => {
18   try {
19     await mongoose.connect(MongooseUri);
20     console.log("Connected to your MongoDB database successfully");
21   } catch (error) {
22     console.log(error.message);
23   }
24 };
25
26 connectToMongo();
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\shopeZ> cd server
PS D:\shopeZ\server> node index.js
App server is running on port 3001
bad auth : authentication failed
PS D:\shopeZ\server> node index.js
App server is running on port 3001
Connected to your MongoDB database successfully
[]
```

4. Create Express Routes:

- Structure:

/routes

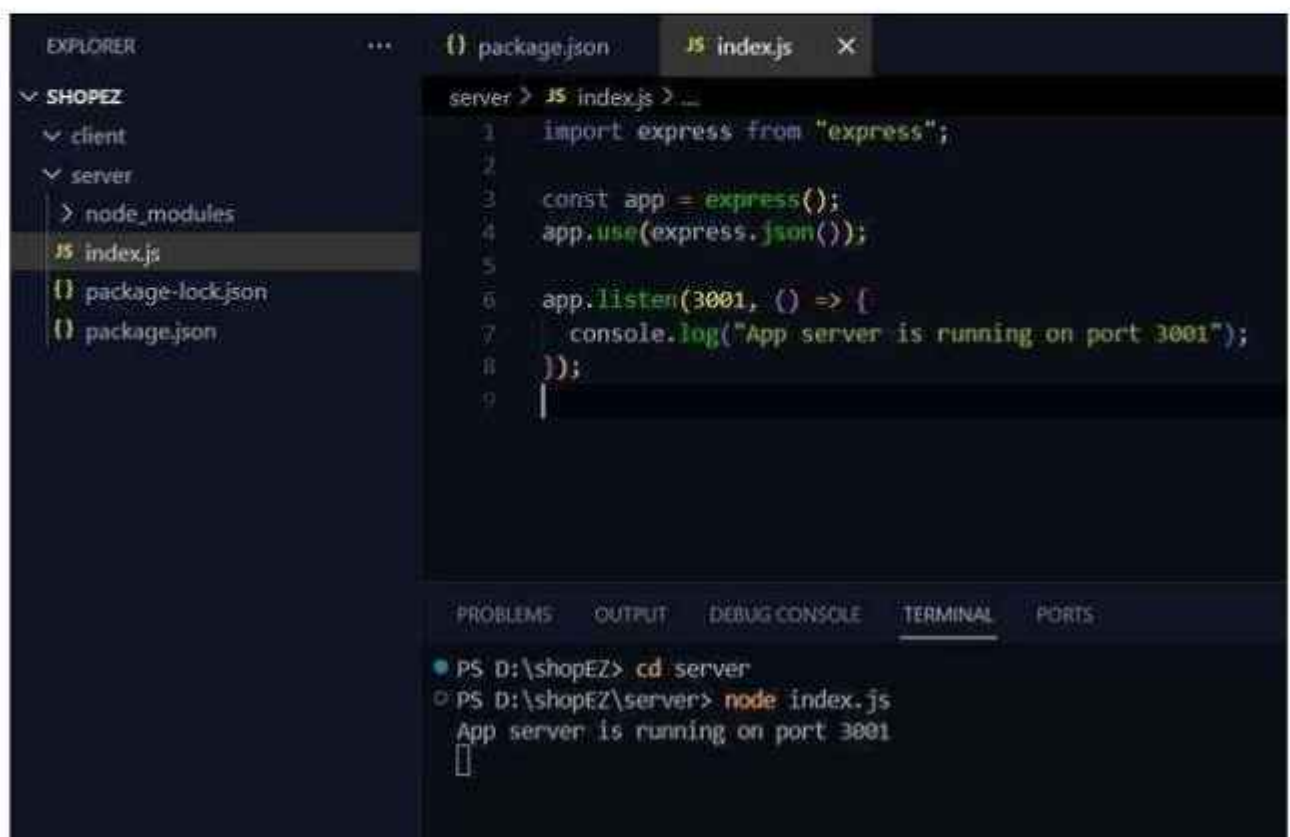
- users.js
- products.js
- orders.js

- Sample route:

```
const express = require("express");  
const router = express.Router();
```

```
router.get("/", (req, res) =>  
  { res.send("All users");  
});
```

```
module.exports = router;
```



The screenshot shows a Visual Studio Code editor with a project named 'SHOPEZ'. The file explorer on the left shows the project structure: 'client' and 'server' folders. The 'server' folder is expanded, showing 'node_modules', 'index.js', 'package-lock.json', and 'package.json'. The 'index.js' file is selected and its content is displayed in the editor. The code in 'index.js' is as follows:

```
server > JS index.js > ...  
1  import express from "express";  
2  
3  const app = express();  
4  app.use(express.json());  
5  
6  app.listen(3001, () => {  
7    console.log("App server is running on port 3001");  
8  });  
9
```

The terminal at the bottom shows the command prompt output:

```
PS D:\shopEZ> cd server  
PS D:\shopEZ\server> node index.js  
App server is running on port 3001  
█
```

5. Implement Data Models (Schemas):

- Use Mongoose schemas for Users, Products, Orders, Carts, Admins, and Restaurants
- Example schema:

```
const mongoose = require("mongoose");
const userSchema = new
  mongoose.Schema({ username: String,
    email: String,
    password: String,
  });
module.exports = mongoose.model("User", userSchema);
```

6. User Authentication:

- Implement registration, login, and protected routes.
- Use JWT for auth tokens.

7. Product and Order Handling:

- Endpoints for:
- Product listing (GET)
- Add product (POST)
- Place order (POST)

8. Admin Functionality:

- Admin dashboard endpoints:
- Manage users
- Add/edit/delete products
- View orders
- Middleware for role-based access

9. Error Handling:

- Global error middleware:

```
app.use((err, req, res, next) => {
  res.status(500).json({ error: err.message });
});
```


Frontend Development

1. Setup React Application:

- Create a React app in the client folder:
`npx create-react-app client`
- Install required libraries:
`npm install axios react-router-dom bootstrap`
- Create required pages and components and set up routing using `react-router-dom`.

2. Design UI Components:

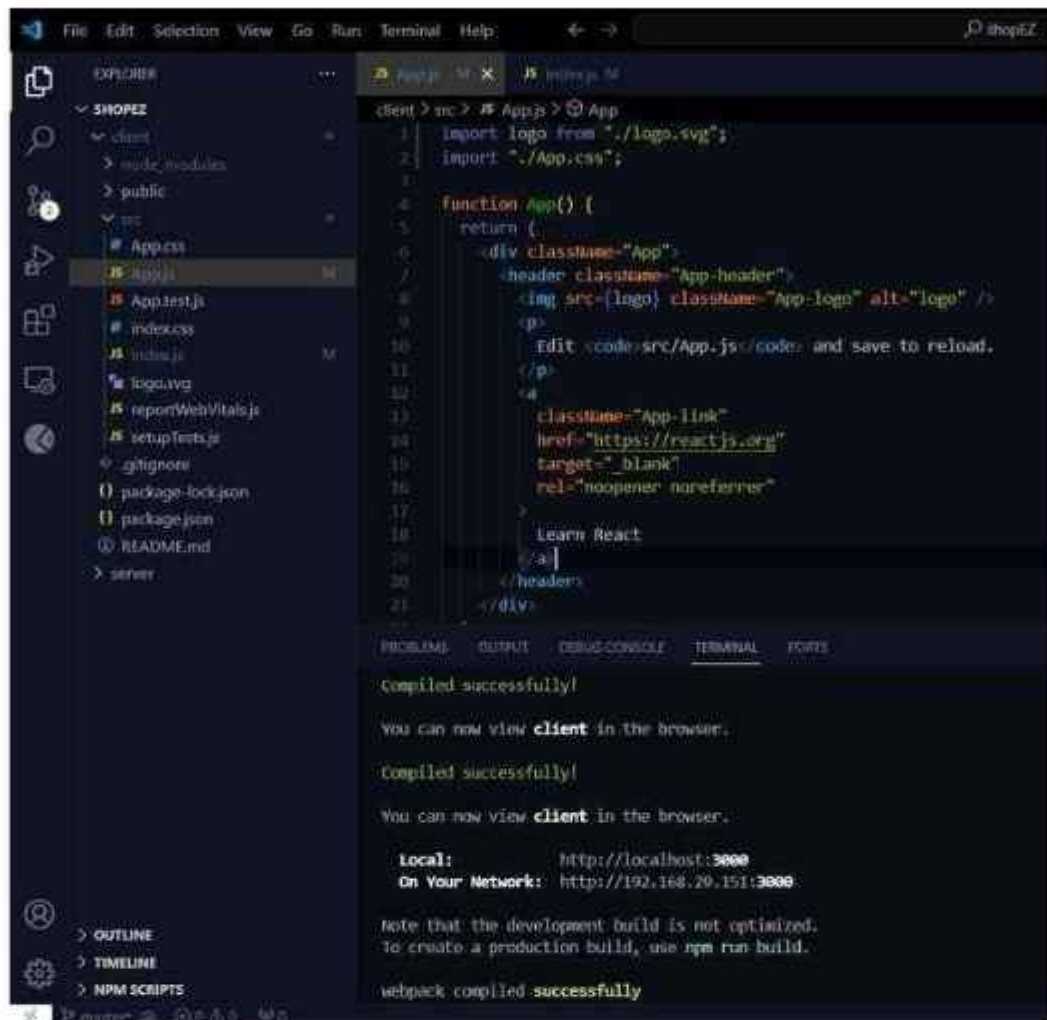
- Break down the UI into reusable components like Navbar, Footer, ProductCard, etc.
- Use CSS or libraries like Bootstrap/Tailwind for styling.
- Implement responsive layouts using Flexbox/Grid.
- Add navigation between pages using `<Link>` and `<Route>` from `react-router-dom`.

3. Implement Frontend Logic :

- Fetch data using `axios` from backend API endpoints (e.g., `/api/products`, `/api/users/login`).
- Use React state (`useState`, `useEffect`) for dynamic data binding.
- Store user info or tokens in `localStorage` after login.

Reference Article Link:

https://www.w3schools.com/react/react_getstarted.asp



```
File Edit Selection View Go Run Terminal Help
client > src > .\App.js > App
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12       </div>
13       <div className="App-link">
14         <a href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer">
17           Learn React
18         </a>
19       </div>
20     </div>
21   );
22 }
```

Compiled successfully!

You can now view **client** in the browser.

Compiled successfully!

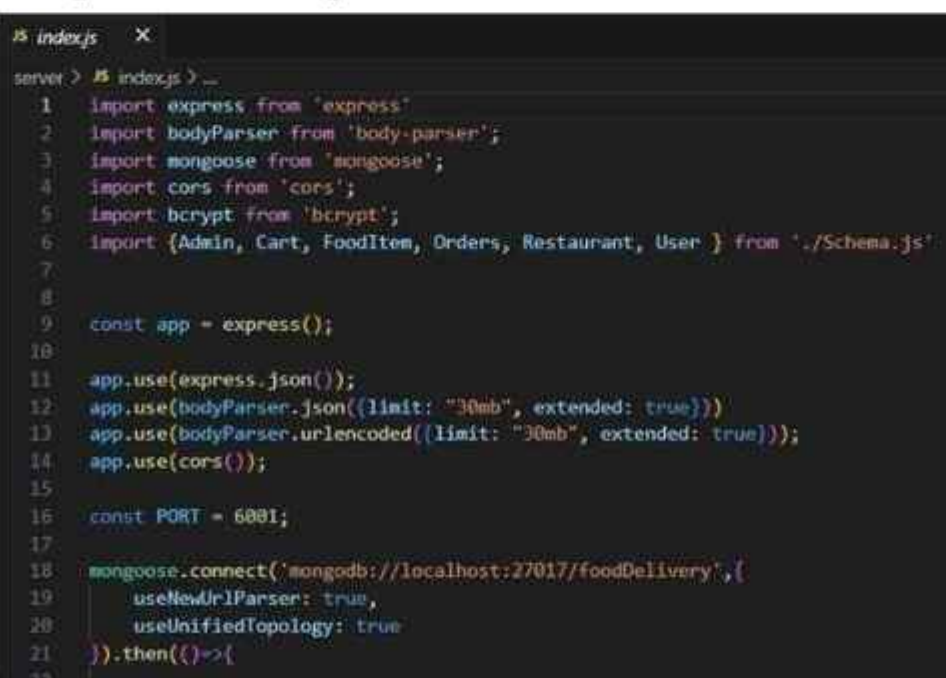
You can now view **client** in the browser.

Local: http://localhost:3000
On Your Network: http://192.168.29.151:3000

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

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



```
server > .\index.js > _
1 import express from 'express';
2 import bodyParser from 'body-parser';
3 import mongoose from 'mongoose';
4 import cors from 'cors';
5 import bcrypt from 'bcrypt';
6 import {Admin, Cart, FoodItem, Orders, Restaurant, User } from './Schema.js';
7
8
9 const app = express();
10
11 app.use(express.json());
12 app.use(bodyParser.json({limit: "30mb", extended: true}));
13 app.use(bodyParser.urlencoded({limit: "30mb", extended: true}));
14 app.use(cors());
15
16 const PORT = 6001;
17
18 mongoose.connect('mongodb://localhost:27017/foodDelivery',{
19   useNewUrlParser: true,
20   useUnifiedTopology: true
21 }).then(()=>{
22
```

Reference Image

Code Explanation:

Server setup:

User Authentication:

Backend

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

```
JS index.js X
server > JS index.js > then() callback
>>
56 app.post('/login', async (req, res) => {
57   const { email, password } = req.body;
58   try {
59     const user = await User.findOne({ email });
60
61     if (!user) {
62       return res.status(401).json({ message: 'Invalid email or password' });
63     }
64     const isMatch = await bcrypt.compare(password, user.password);
65     if (!isMatch) {
66       return res.status(401).json({ message: 'Invalid email or password' });
67     } else {
68       return res.json(user);
69     }
70   } catch (error) {
71     console.log(error);
72     return res.status(500).json({ message: 'Server Error' });
73   }
74 });
75
```

```
JS index.js X
server > JS index.js > then() callback > app.post('/login') callback
23 app.post('/register', async (req, res) => {
24   const { username, email, usertype, password, restaurantAddress, restaurantImage } = req.body;
25   try {
26     const existingUser = await User.findOne({ email });
27     if (existingUser) {
28       return res.status(400).json({ message: 'User already exists' });
29     }
30     const hashedPassword = await bcrypt.hash(password, 10);
31     if (usertype === 'restaurant') {
32       const newUser = new User({
33         username, email, usertype, password: hashedPassword, approval: 'pending'
34       });
35       const user = await newUser.save();
36       console.log(user._id);
37       const restaurant = new Restaurant({ ownerId: user._id, title: username,
38         address: restaurantAddress, mainImg: restaurantImage, menu: [] });
39       await restaurant.save();
40       return res.status(201).json(user);
41     } else {
42       const newUser = new User({
43         username, email, usertype, password: hashedPassword, approval: 'approved'
44       });
45       const userCreated = await newUser.save();
46       return res.status(201).json(userCreated);
47     }
48   } catch (error) {
49     console.log(error);
50     return res.status(500).json({ message: 'Server Error' });
51   }
52 });
53
```


Frontend

Login:

```
JS GeneralContext.js U X
client > src > context > JS GeneralContext.js > [e] GeneralContextProvider > [e] register > [e] then() callback
46   const login = async () =>{
47     try{
48       const loginInputs = {email, password}
49       await axios.post('http://localhost:6001/login', loginInputs)
50       .then( async (res)=>{
51
52         localStorage.setItem('userId', res.data._id);
53         localStorage.setItem('userType', res.data.usertype);
54         localStorage.setItem('username', res.data.username);
55         localStorage.setItem('email', res.data.email);
56         if(res.data.usertype === 'customer'){
57           navigate('/');
58         } else if(res.data.usertype === 'admin'){
59           navigate('/admin');
60         }
61       }).catch((err) =>{
62         alert("login failed!!");
63         console.log(err);
64       });
65     }catch(err){
66       console.log(err);
67     }
68   }
69 }
```

Logout:

```
GeneralContext.jsx U X
client > src > context > GeneralContext.jsx > [e] GeneralContextProvider > [e] login >
72
73   const logout = async () =>{
74
75     localStorage.clear();
76     for (let key in localStorage) {
77       if (localStorage.hasOwnProperty(key)) {
78         localStorage.removeItem(key);
79       }
80     }
81
82     navigate('/');
83   }
84
85 }
```

Register:

```
GeneralContext.js U X
client > src > context > GeneralContext.js > GeneralContextProvider > logout
75
76 const inputs = {username, email, usertype, password, restaurantAddress, restaurantImage};
77
78 const register = async () =>{
79   try{
80     await axios.post('http://localhost:6001/register', inputs)
81     .then( async (res)=>{
82       localStorage.setItem('userId', res.data._id);
83       localStorage.setItem('usertype', res.data.usertype);
84       localStorage.setItem('username', res.data.username);
85       localStorage.setItem('email', res.data.email);
86
87       if(res.data.usertype === 'customer'){
88         navigate('/');
89       } else if(res.data.usertype === 'admin'){
90         navigate('/admin');
91       } else if(res.data.usertype === 'restaurant'){
92         navigate('/restaurant');
93       }
94     }).catch((err) =>{
95       alert("registration failed!!");
96       console.log(err);
97     });
98   }catch(err){
99     console.log(err);
100   }
101 }
```

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 U X
client > src > pages > customer > IndividualRestaurant.jsx > IndividualRestaurant > handleCategoryCheckBox
33
34 const fetchRestaurants = async() =>{
35   await axios.get('http://localhost:6001/fetch-restaurant/${id}').then(
36     (response)=>{
37       setRestaurant(response.data);
38       console.log(response.data)
39     }
40   ).catch((err)=>{
41     console.log(err);
42   })
43 }
44
45 const fetchCategories = async () =>{
46   await axios.get('http://localhost:6001/fetch-categories').then(
47     (response)=>{
48       setAvailableCategories(response.data);
49     }
50   )
51 }
52
53 const fetchItems = async () =>{
54   await axios.get('http://localhost:6001/fetch-items').then(
55     (response)=>{
56       setItems(response.data);
57       setVisibleItems(response.data);
58     }
59   )
60 }
```

Filtering products:

```
Products.jsx
client > src > components > Products.jsx > 94 useEffect() callback
94
95 const [sortFilter, setSortFilter] = useState('popularite');
96 const [categoryFilter, setCategoryFilter] = useState('');
97 const [genderFilter, setGenderFilter] = useState('');
98
99
100 const handleCategoryCheckBox = (e) => {
101   const value = e.target.value;
102   if (e.target.checked) {
103     setCategoryFilter([...categoryFilter, value]);
104   } else {
105     setCategoryFilter(categoryFilter.filter(size => size !== value));
106   }
107 }
108
109 const handleGenderCheckBox = (e) => {
110   const value = e.target.value;
111   if (e.target.checked) {
112     setGenderFilter([...genderFilter, value]);
113   } else {
114     setGenderFilter(genderFilter.filter(size => size !== value));
115   }
116 }
117
118 const handleSortFilterChange = (e) => {
119   const value = e.target.value;
120   setSortFilter(value);
121   if (value === 'low-price') {
122     setVisibleProducts(visibleProducts.sort((a,b) => a.price - b.price));
123   } else if (value === 'high-price') {
124     setVisibleProducts(visibleProducts.sort((a,b) => b.price - a.price));
125   } else if (value === 'discount') {
126     setVisibleProducts(visibleProducts.sort((a,b) => b.discount - a.discount));
127   }
128 }
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
267
```


Add product to cart:

Frontend

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

```
IndividualRestaurant.jsx U X
client > src > pages > customer > IndividualRestaurant.jsx > [10] IndividualRestaurant
114 const handleAddToCart = async(foodItemId, foodItemName, restaurantId,
115                                foodItemImg, price, discount) =>{
116   await axios.post('http://localhost:6001/add-to-cart', {userId, foodItemId,
117                                foodItemName, restaurantId, foodItemImg,
118                                price, discount, quantity}).then(
119     (response)=>{
120       alert("product added to cart!!");
121       setCartItem('');
122       setQuantity(0);
123       fetchCartCount();
124     }
125   ).catch((err)=>{
126     alert("Operation failed!!");
127   })
128 }
129
```

Backend

Add product to cart:

```
JS index.js X
server > JS index.js > then() callback > app.put('/remove-item') callback
402 // add cart item
403
404 app.post('/add-to-cart', async(req, res)=>{
405   const {userId, foodItemId, foodItemName, restaurantId,
406         foodItemImg, price, discount, quantity} = req.body
407   try{
408     const restaurant = await Restaurant.findById(restaurantId);
409     const item = new Cart({userId, foodItemId, foodItemName,
410                           restaurantId, restaurantName: restaurant.title,
411                           foodItemImg, price, discount, quantity});
412     await item.save();
413     res.json({message: 'Added to cart'});
414   }catch(err){
415     res.status(500).json({message: "Error occurred"});
416   }
417 })
418
```


Order products:

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

Frontend

```
Cart.jsx 2. U X
client > src > pages > customer > Cart.jsx > [0] Cart

72  const placeOrder = async() =>{
73    if(cart.length > 0){
74      await axios.post('http://localhost:6001/place-cart-order', {userId, name,
75        mobile, email, address, pincode, paymentMethod,
76        orderDate: new Date()}).then(
77        (response)=>{
78          alert('Order placed!!!');
79          setName('');
80          setMobile('');
81          setEmail('');
82          setAddress('');
83          setPincode('');
84          setPaymentMethod('');
85          navigate('/profile');
86        })
87    }
88  }
89 }
```

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.

```
index.js X
server > index.js > then() callback > app.listen() callback

435  // Order from cart
436
437  app.post('/place-cart-order', async(req, res)=>{
438    const {userId, name, mobile, email, address, pincode,
439      paymentMethod, orderDate} = req.body;
440    try{
441      const cartItems = await Cart.find({userId});
442      cartItems.map(async (item)=>{
443        const newOrder = new Orders({userId, name, email,
444          mobile, address, pincode, paymentMethod,
445          orderDate, restaurantId: item.restaurantId,
446          restaurantName: item.restaurantName,
447          foodItemId: item.foodItemId, foodItemName: item.foodItemName,
448          foodItemImg: item.foodItemImg, quantity: item.quantity,
449          price: item.price, discount: item.discount});
450        await newOrder.save();
451        await Cart.deleteOne({_id: item._id})
452      })
453      res.json({message: 'Order placed'});
454    }catch(err){
455      res.status(500).json({message: "Error occurred"});
456    }
457  })
```

Add new product:

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

Frontend:

```
NewProduct.jsx 1,1 X
client > src > pages > restaurant > NewProduct.jsx > [0] NewProduct
46 const handleNewProduct = async() =>{
47   await axios.post('http://localhost:6001/add-new-product', {restaurantId: restaurant._id,
48     productName, productDescription, productMainImg, productCategory, productMenuCategory,
49     productNewCategory, productPrice, productDiscount}).then(
50     (response)=>{
51       alert("product added");
52       setProductName('');
53       setProductDescription('');
54       setProductMainImg('');
55       setProductCategory('');
56       setProductMenuCategory('');
57       setProductNewCategory('');
58       setProductPrice(0);
59       setProductDiscount(0);
60       navigate('/restaurant-menu');
61     })
62   }
63 }
64
```

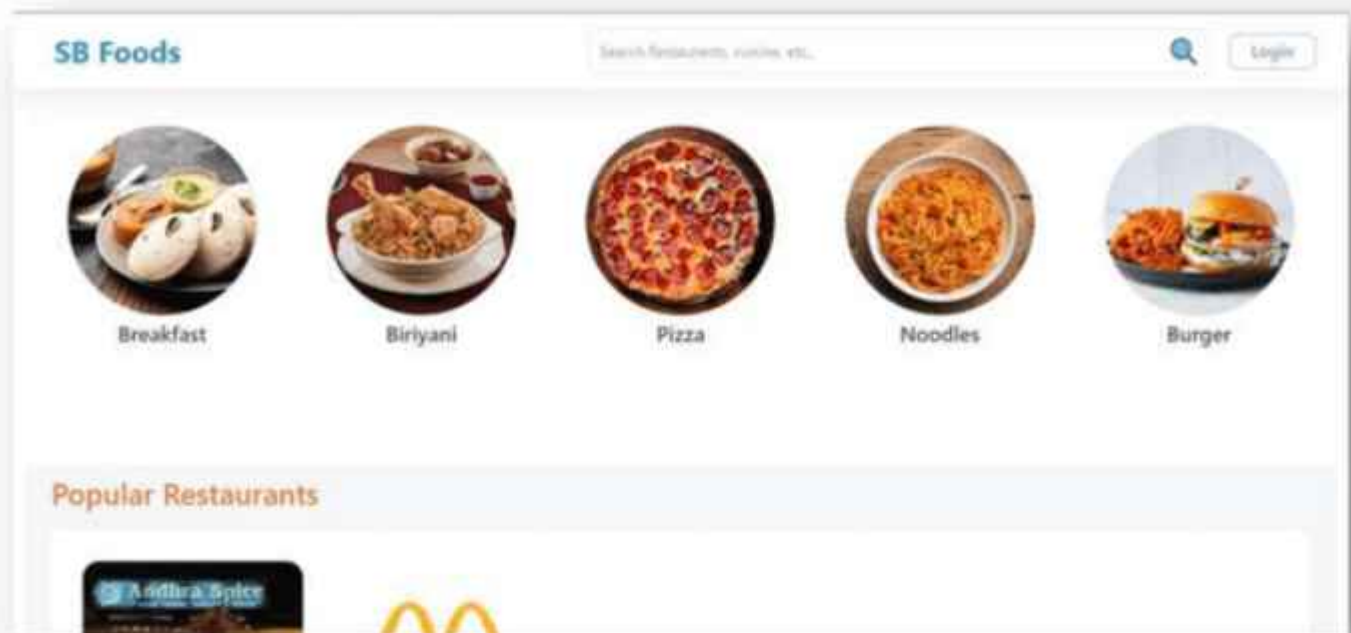
Backend:

```
JS index.js X
server > JS index.js > then() callback
285 // Add new product
286 app.post('/add-new-product', async(req, res)=>{
287   const {restaurantId, productName, productDescription,
288     productMainImg, productCategory, productMenuCategory,
289     productNewCategory, productPrice, productDiscount} = req.body;
290   try{
291     if(productMenuCategory === 'new category'){
292       const admin = await Admin.findOne();
293       admin.categories.push(productNewCategory);
294       await admin.save();
295       const newProduct = new FoodItem({restaurantId, title: productName,
296         description: productDescription, itemImg: productMainImg,
297         category: productCategory, menuCategory: productNewCategory,
298         price: productPrice, discount: productDiscount, rating: 0});
299       await newProduct.save();
300       const restaurant = await Restaurant.findById(restaurantId);
301       restaurant.menu.push(productNewCategory);
302       await restaurant.save();
303     } else{
304       const newProduct = new FoodItem({restaurantId, title: productName,
305         description: productDescription, itemImg: productMainImg,
306         category: productCategory, menuCategory: productMenuCategory,
307         price: productPrice, discount: productDiscount, rating: 0});
308       await newProduct.save();
309     }
310     res.json({message: "product added!"});
311   }catch(err){
312     res.status(500).json({message: "Error occured"});
313   }
314 })
315
```

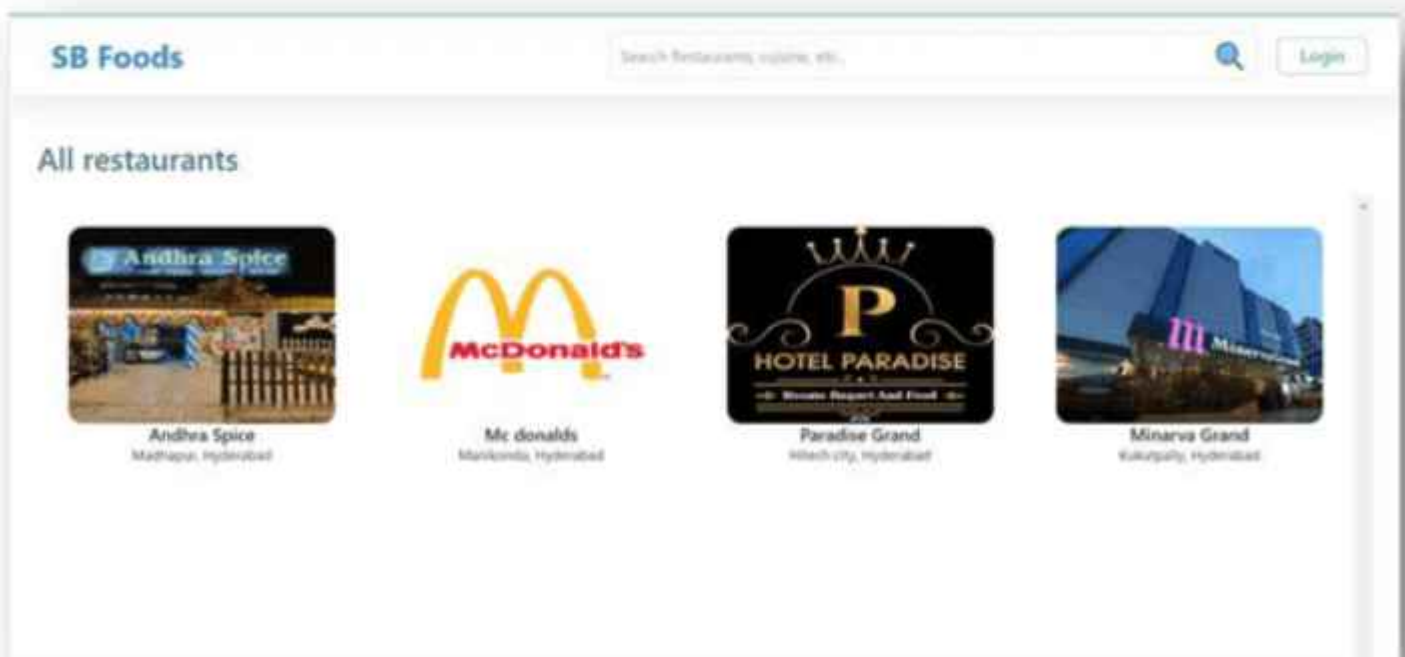
Along with this, implement additional features to view all orders, products, etc., in the admin dashboard.

Project Implementation & Execution

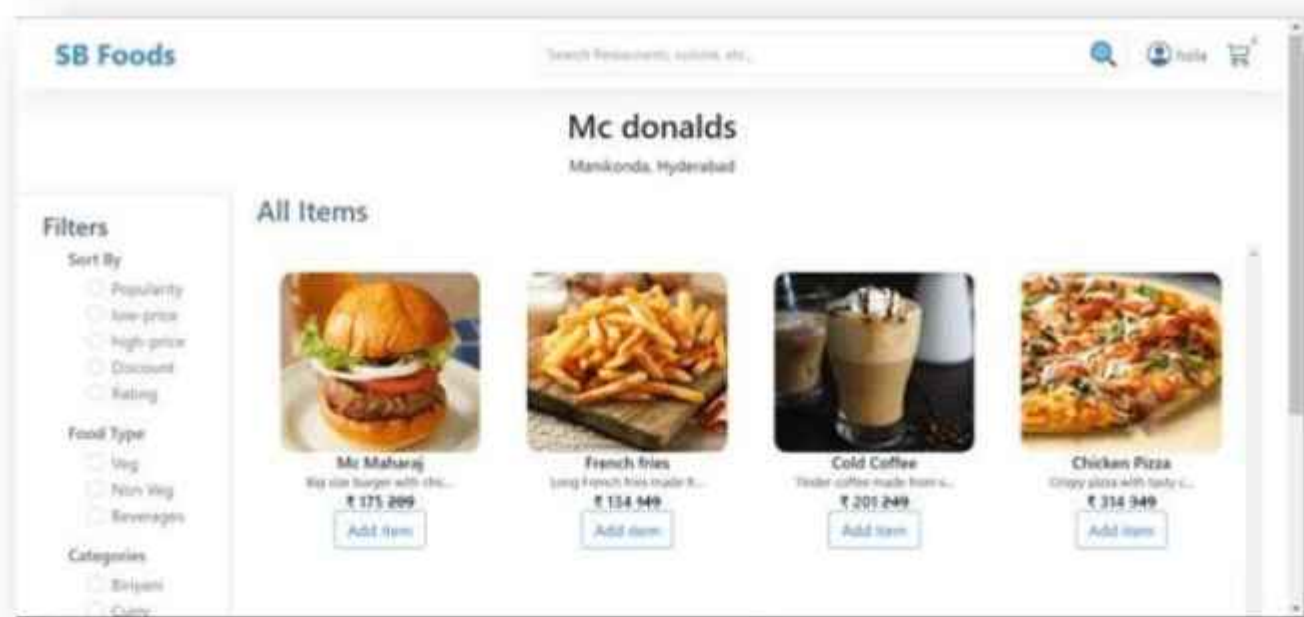
Landing page



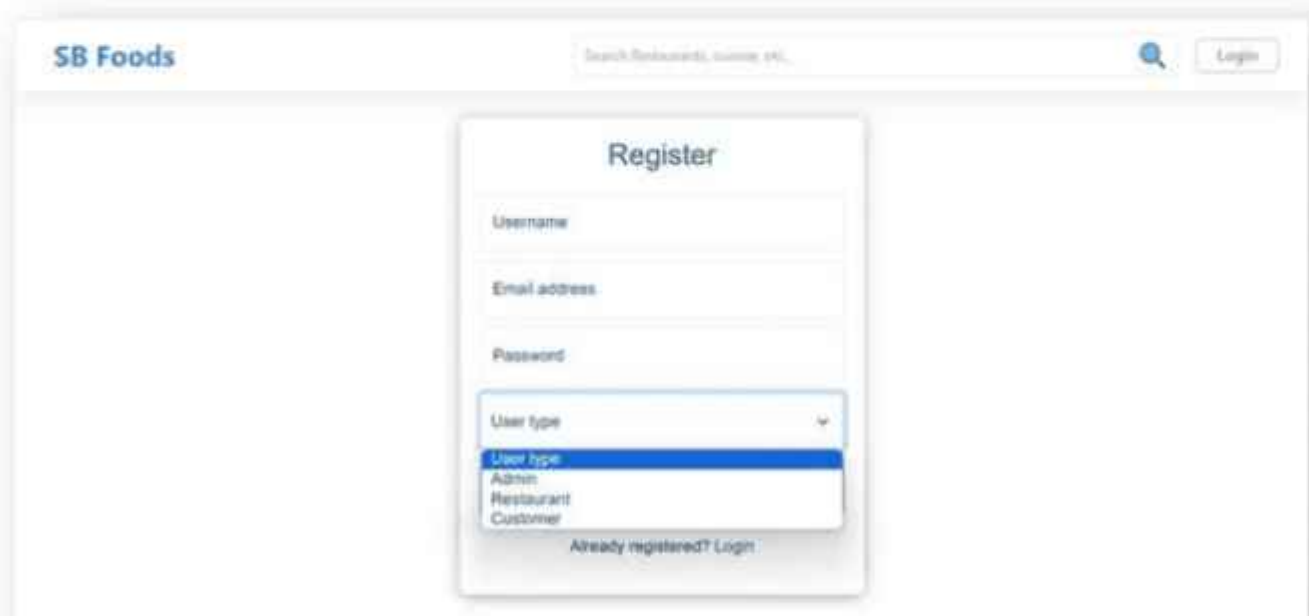
Landing page



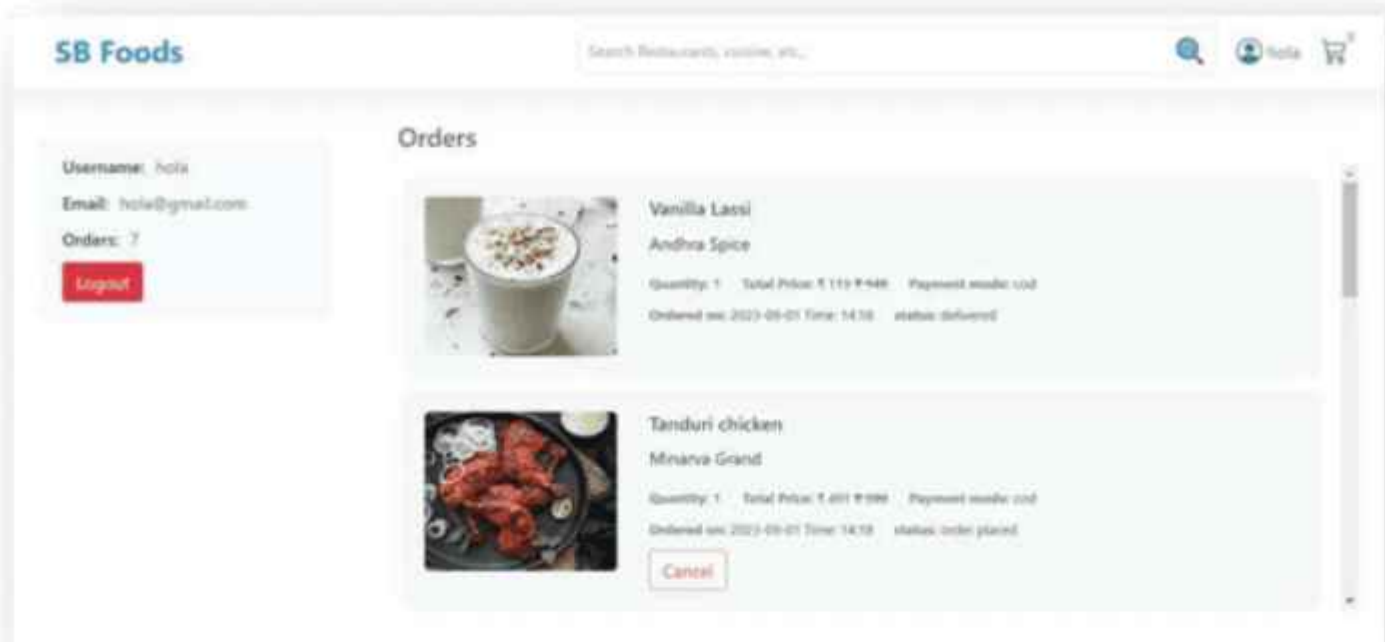
Restaurant Menu



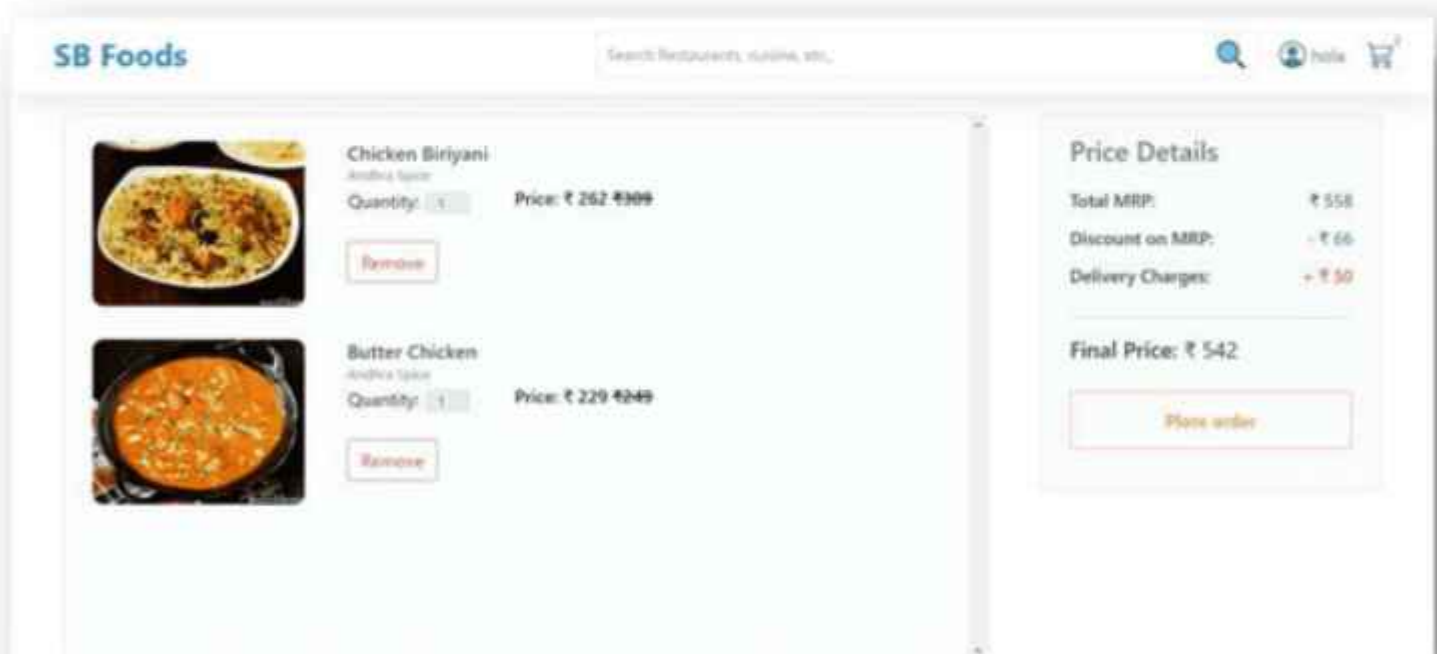
Authentication



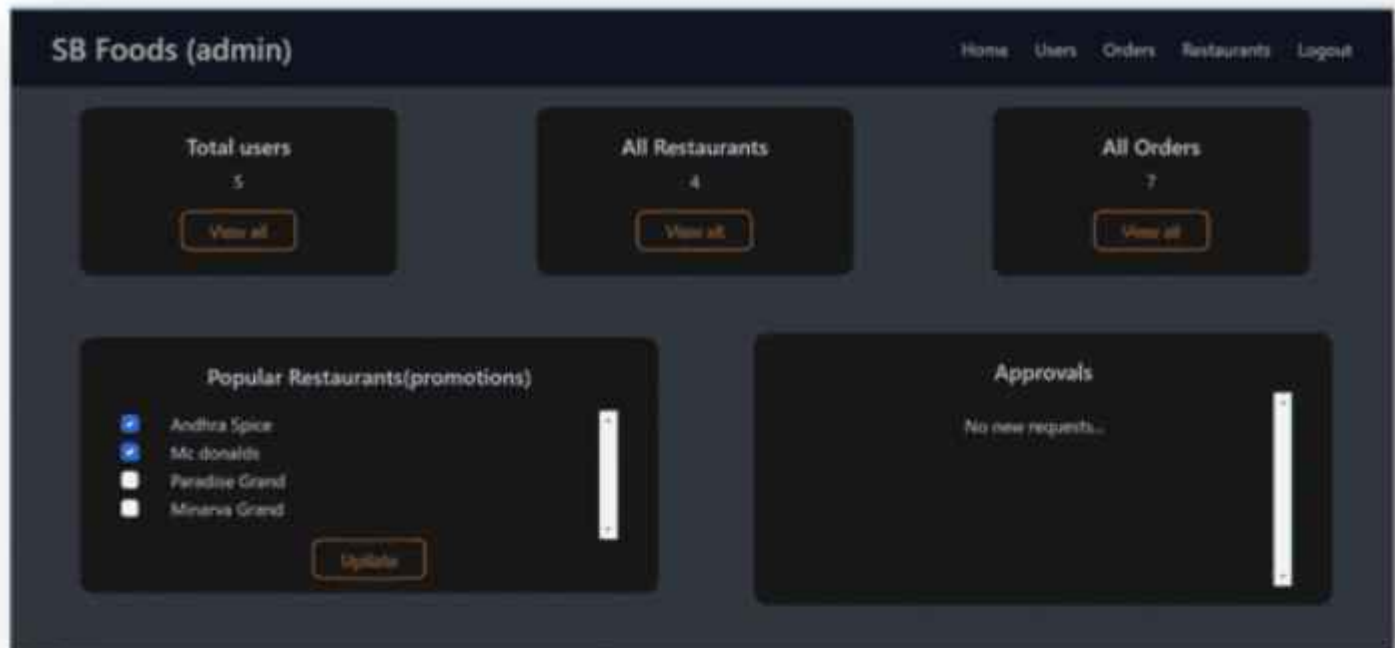
User Profile



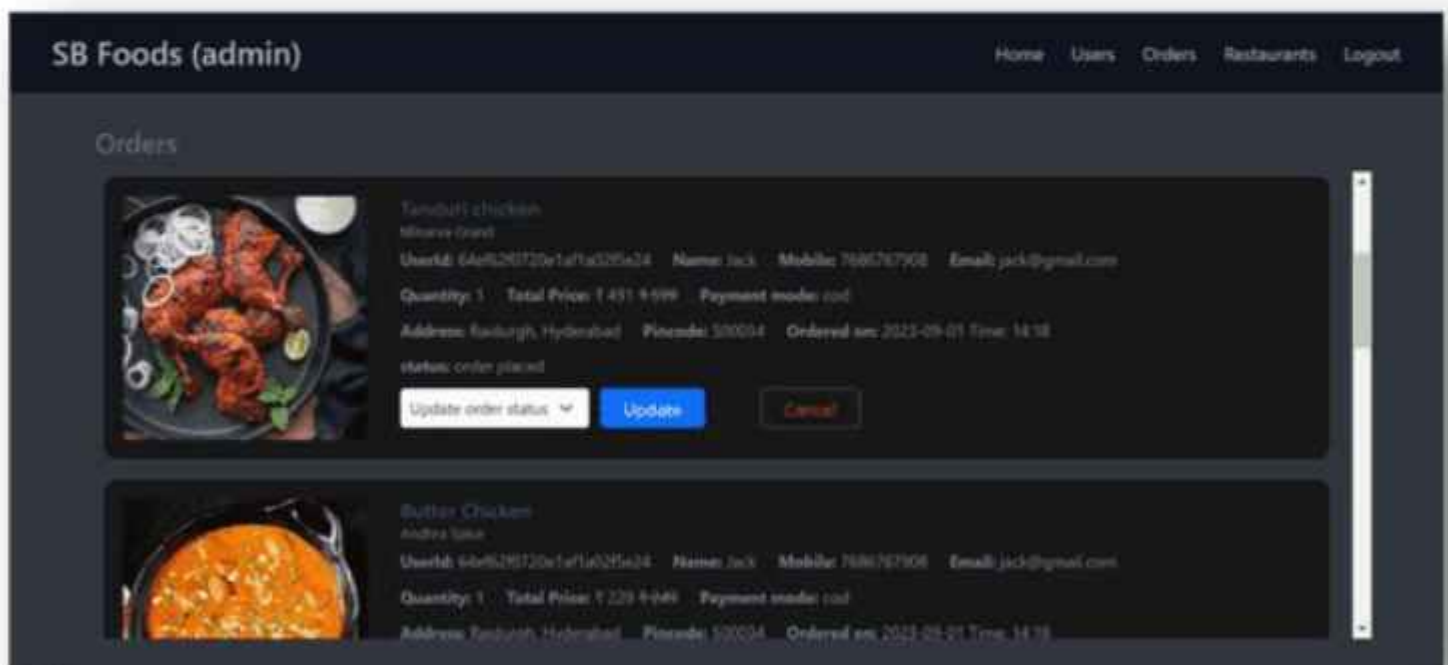
Cart



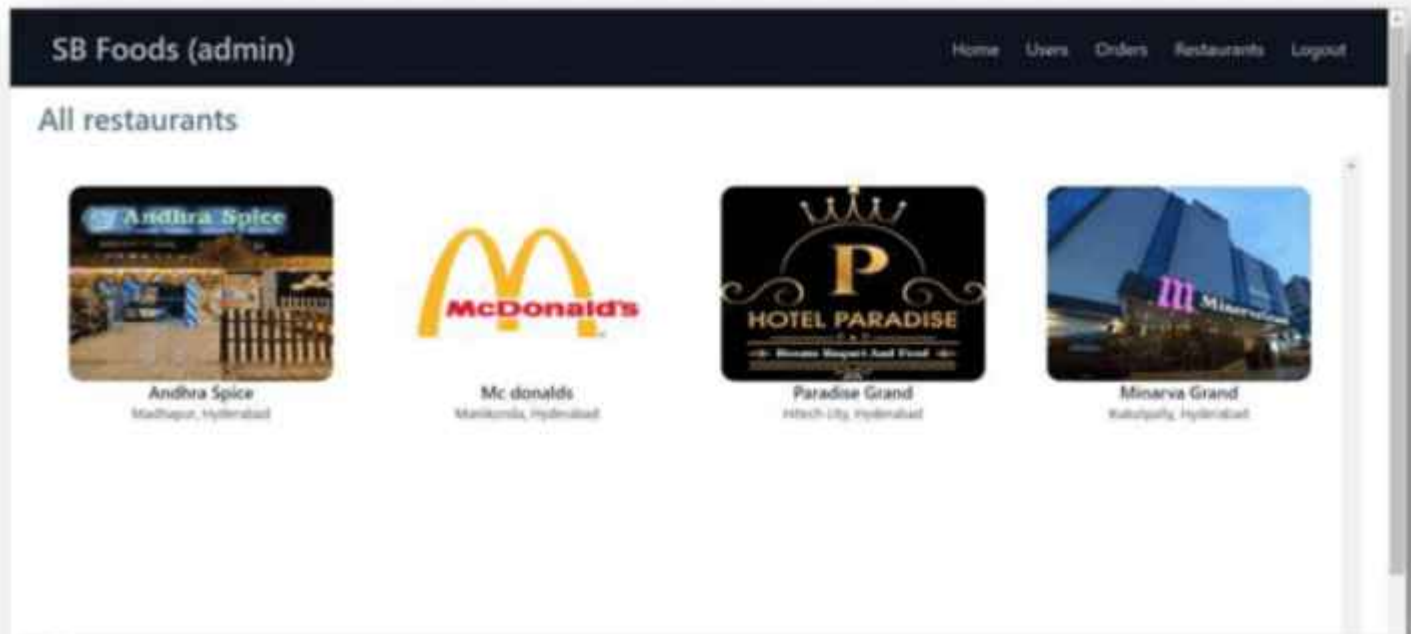
Admin dashboard



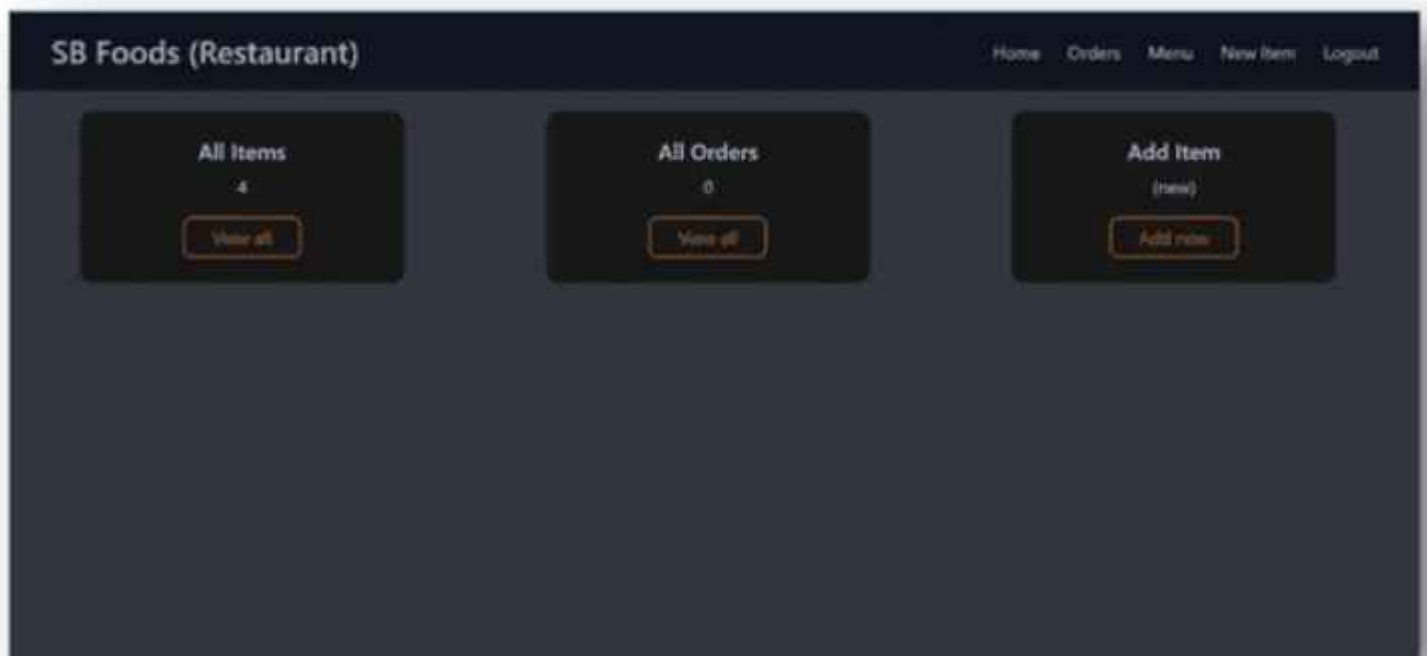
All Orders



All restaurants



Restaurant Dashboard



The screenshot shows a web application interface for 'SB Foods (Restaurant)'. At the top, there is a navigation bar with links: Home, Orders, Menu, New Item, and Logout. The main content area is titled 'New Product' and contains a form with the following fields:

- Product name
- Product Description
- Thumbnail img url
- Type: ☐ Veg, ☐ Non Veg, ☐ Beverages
- Category: Choose Product cat
- Price: 0
- Discount in %: 0

At the bottom of the form is a blue button labeled 'Add product'.

Conclusion

The development of OrderOnTheGo: Your On-Demand Food Ordering Solution has been a highly enriching experience. This project enabled the application of theoretical knowledge in a practical environment, integrating technologies such as React.js, Node.js, Express, and MongoDB to build a dynamic and responsive full-stack application.

The journey involved real-world challenges such as UI/UX optimization, secure API integration, state management, and deployment strategies. Through this project, I have significantly enhanced my technical, problem-solving, and project management skills. This solution not only simulates the functionality of modern food ordering platforms but also lays the groundwork for future innovations in the domain. It stands as a testament to continuous learning and the potential of web technologies in transforming business operations.

Project Links

Git hub repository:

Demo video(you tube): <https://youtu.be/r4hmxT6eeEs?t=74>

About the developers team

Team ID : LTVIP2025TMID55809

Team members :

- 1.Sripriya Akula
- 2.Poojitha Pasupuleti
- 3.Parasa Sundar Singh
- 4.Sarva Sree Lakshmi Manaswini

sInstitution: DMS SVH College of Engineering

ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to our project guide, the faculty members, and the institution for their invaluable support, guidance, and encouragement throughout the duration of this project.

We are especially thankful to SmartInternz and the entire internshub coordination team for offering this exceptional opportunity to gain real world industry experience. The well-structured learning modules, expert mentorship, and continuous support provided by the platform played a pivotal role in shaping our project and enhancing our skills.

Our sincere appreciation also goes to our team members of Team ID: LTVIP2025TMID55809, whose dedication, cooperation, and collaborative spirit were essential to the successful completion of the OrderOnTheGo project. This experience has significantly improved our ability to work as a team, face challenges together, and deliver effective technical solutions.