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

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 or 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 whichwill 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 wantin 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 delivary 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.

ProductFunctions

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

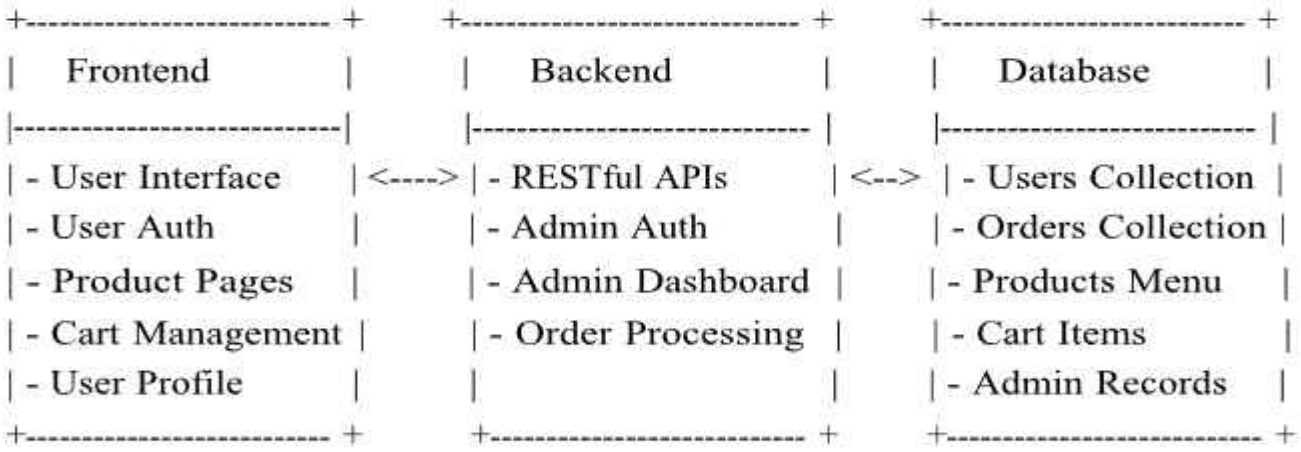
Meet Lisa, a college student working late on an assignment. It's midnight, and she realizes she skipped dinner. Cooking is not an option, and going out isn't safe or convenient.

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



Key Components:

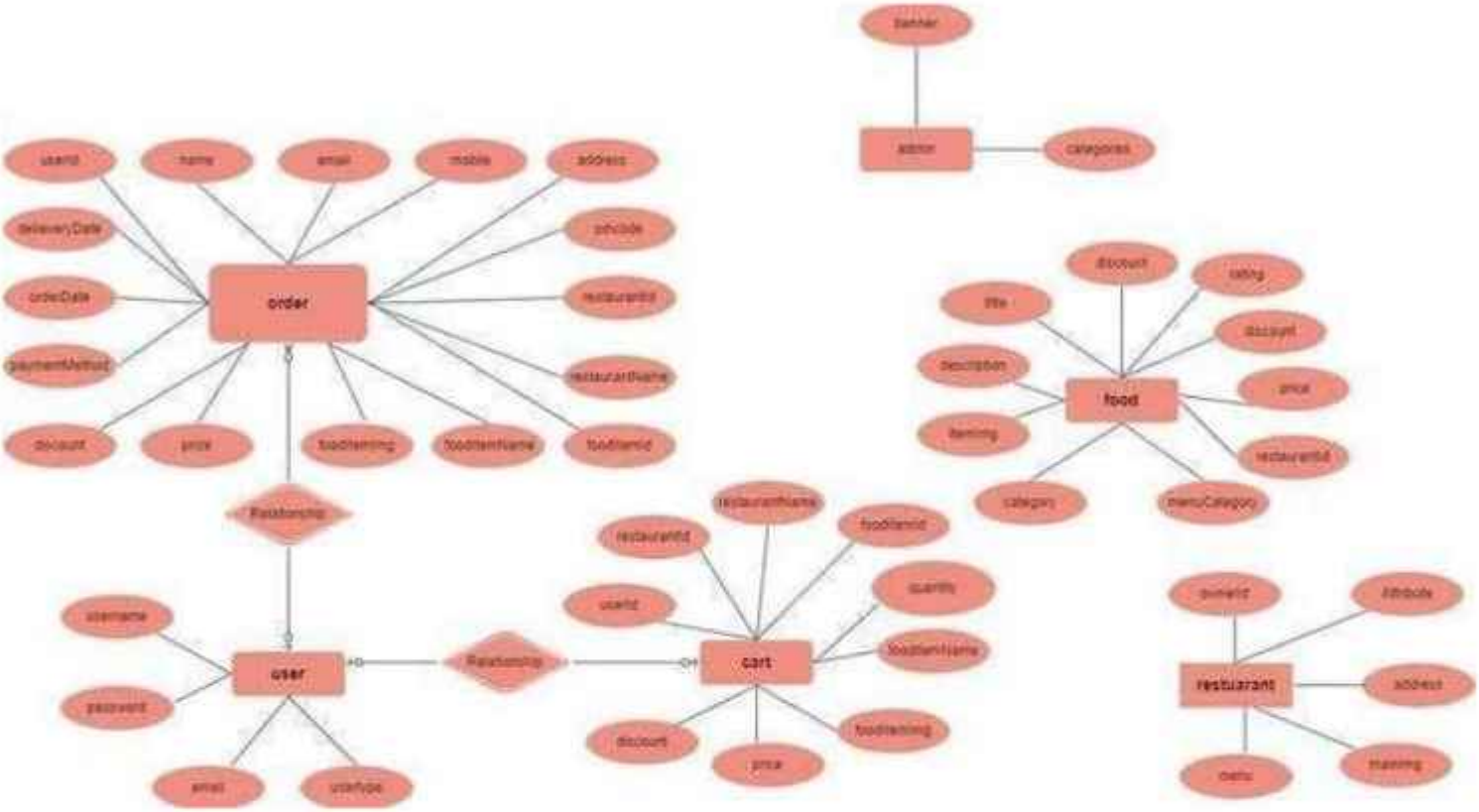
- Frontend: Built with responsive UI for user interaction, login, cart, and admin control.
- Backend: Node.js/Express or similar stack serving APIs for users, products, orders, and admin.
- Database: MongoDB or SQL storing persistent data for orders, users, and menu items.

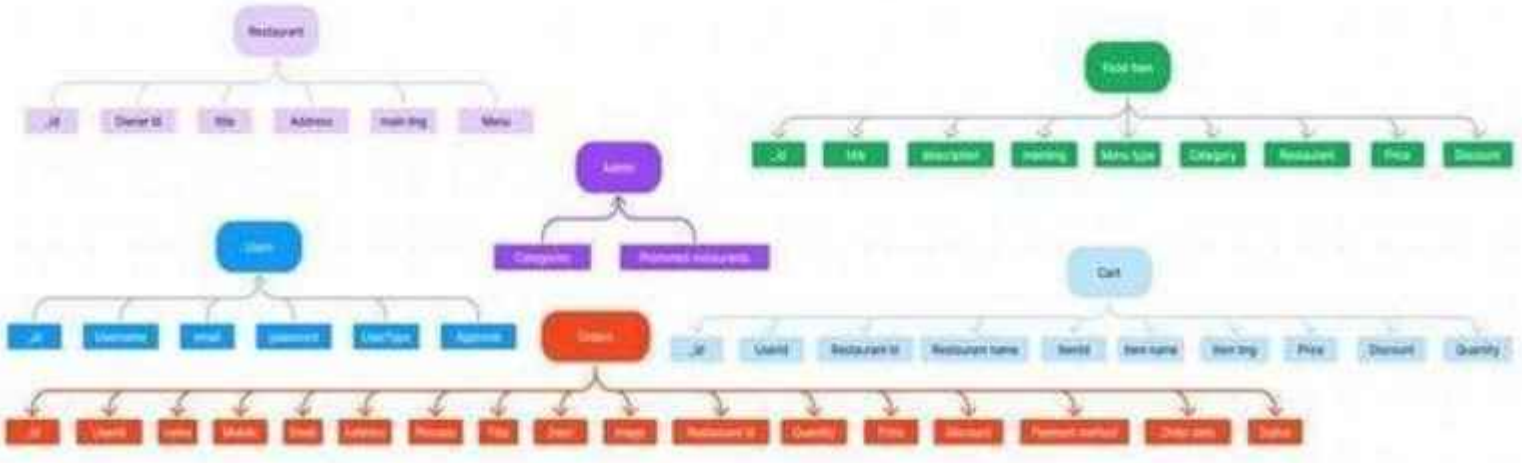
ER-Diagram Overview

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

Smart Internz

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.

Pre-Requisites & 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/node js-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-vardhan-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

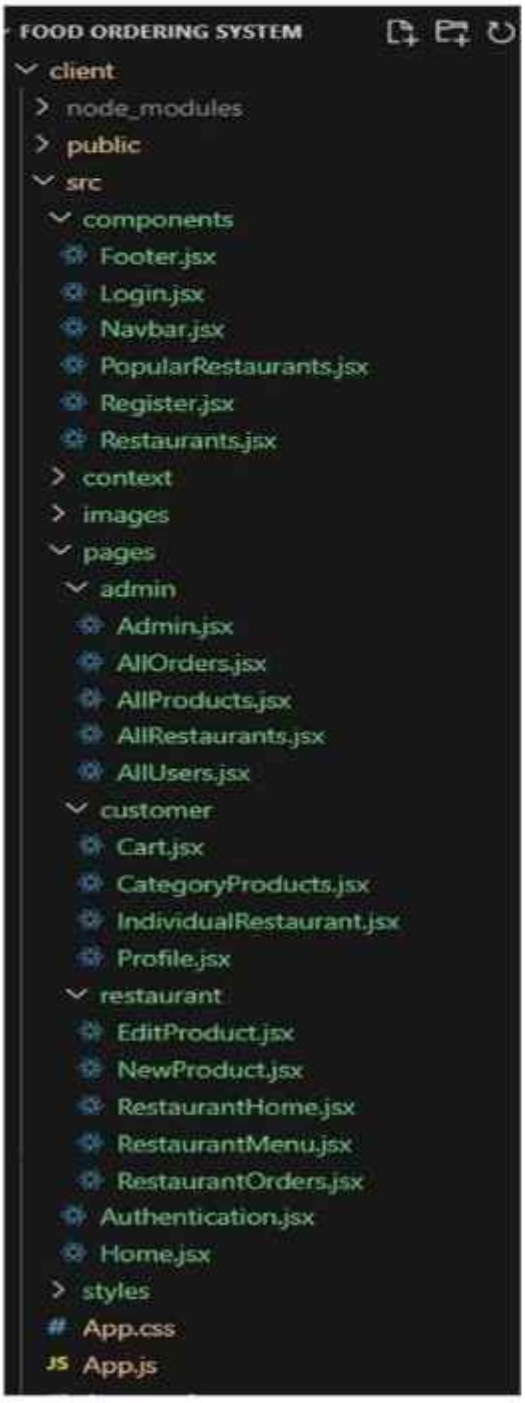
- Restaurants start by authenticating with their credentials.
- They need to get approval from the admin to start listing the products.
- They can add/edit the food items.

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.

ProjectStructure

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:

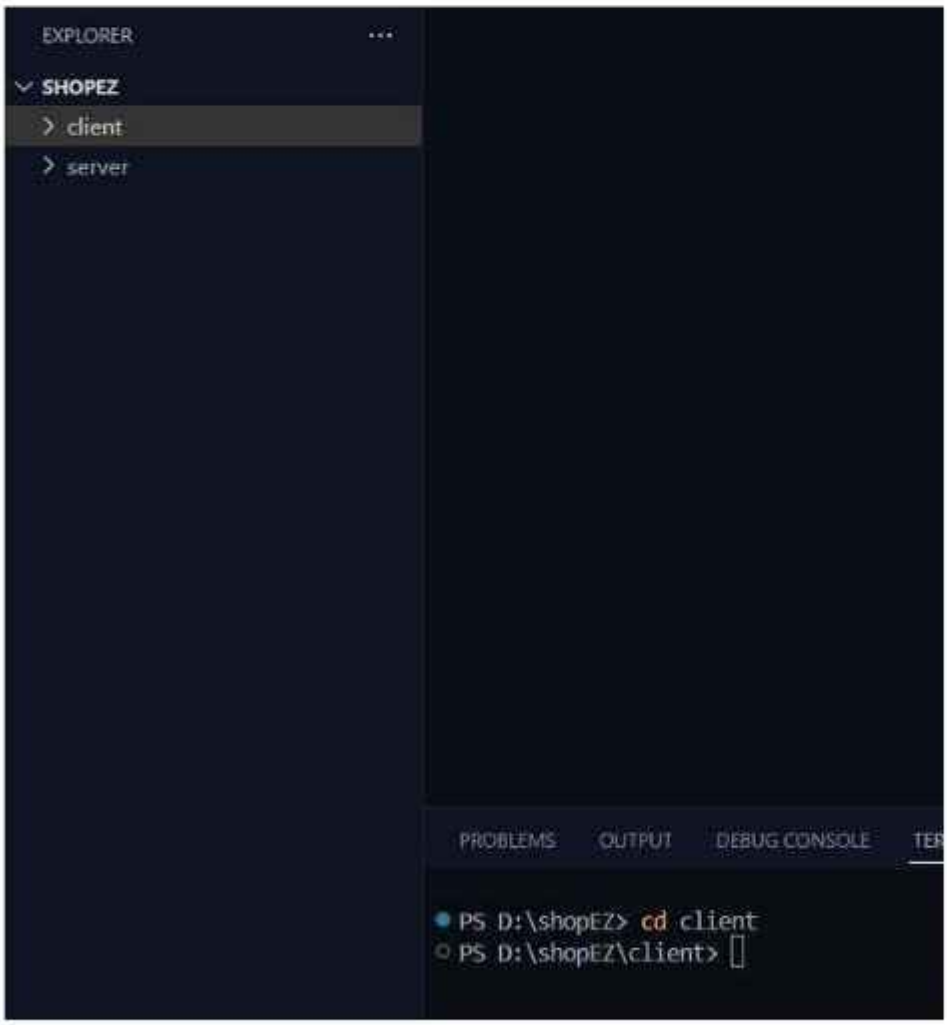
• Node.js.
Reference Article: https://www.geeksforgeeks.org/installation-of-node- js-on-windows/

• 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.

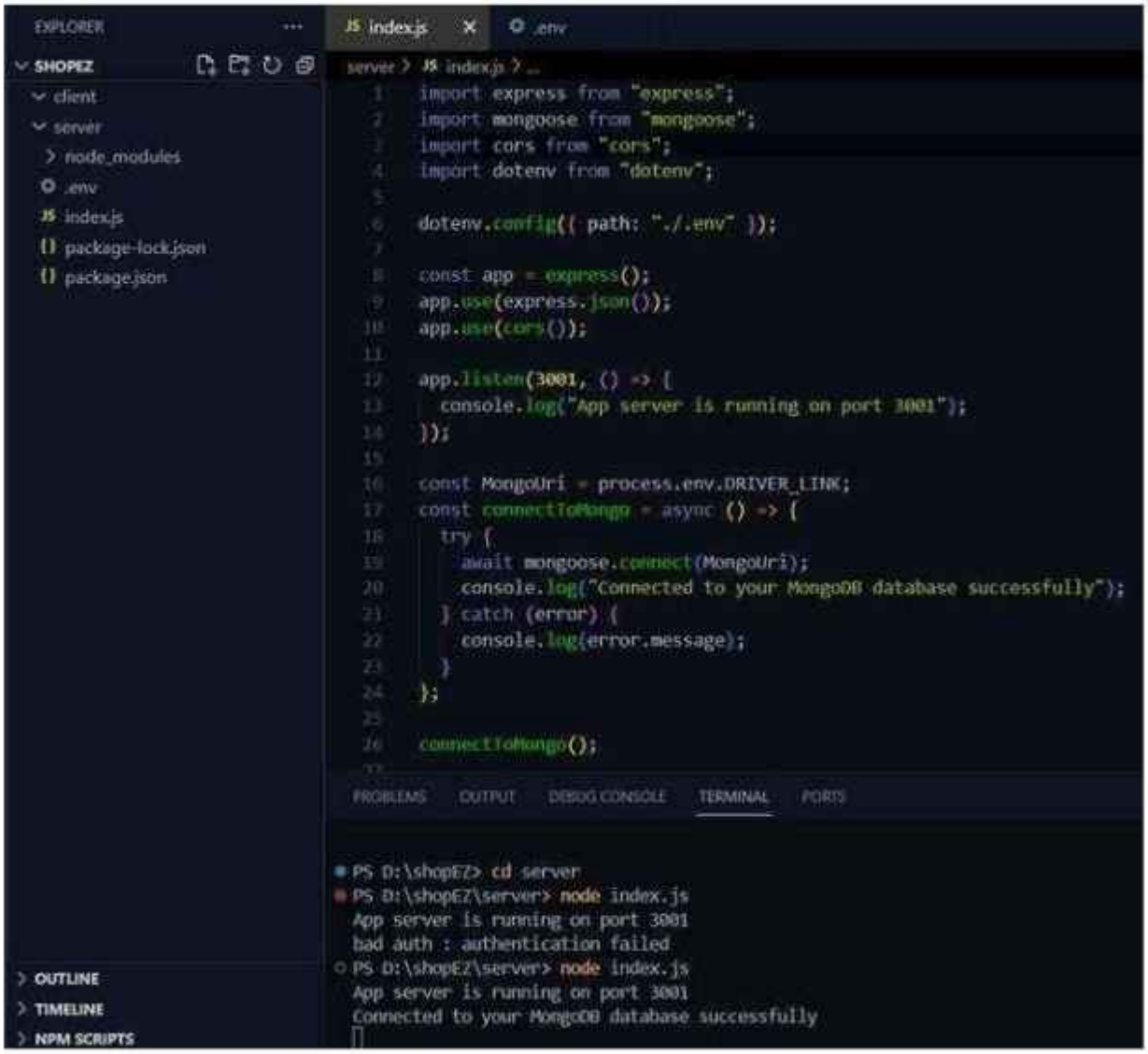
Database Development

Create database in cloud

- Install Mongoose.
- Create database connection.

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

Reference Image:



Smart Internz

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}
})
```

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
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);
```

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.

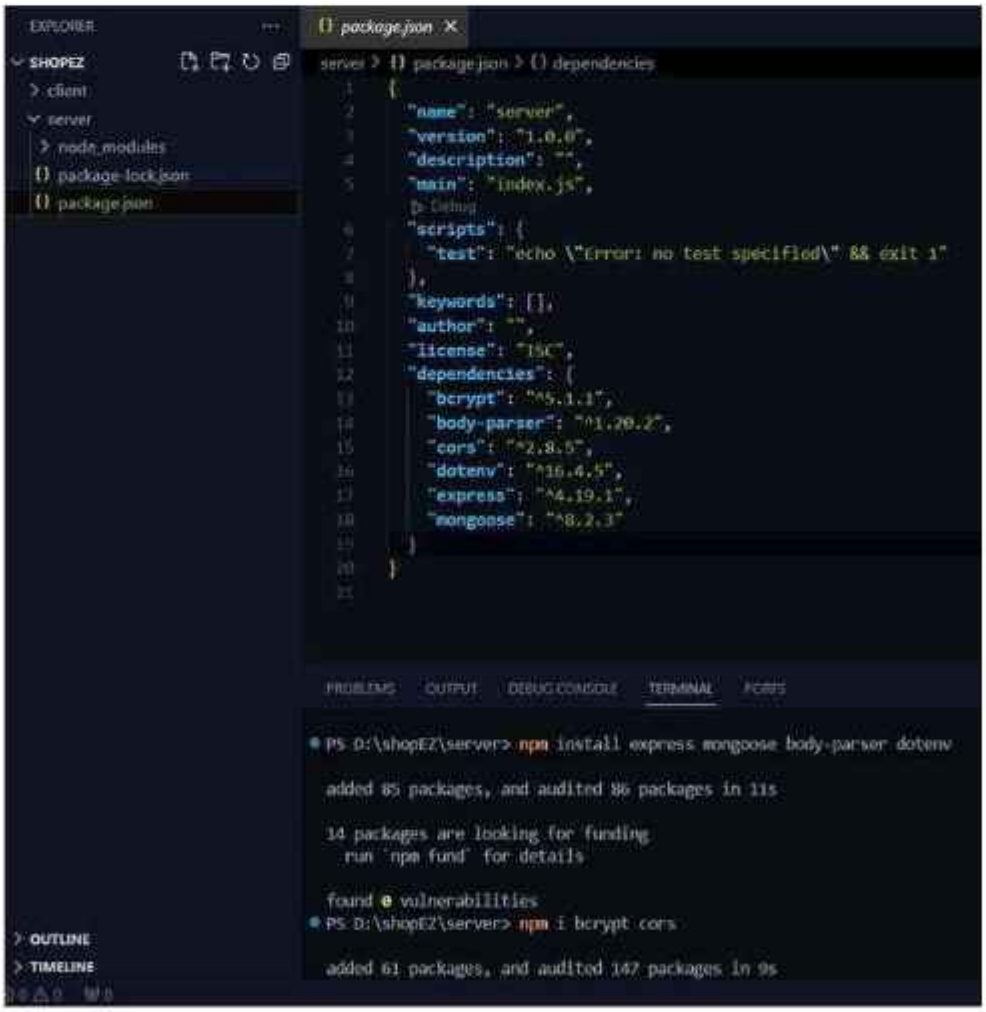
BackEnd Development

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:

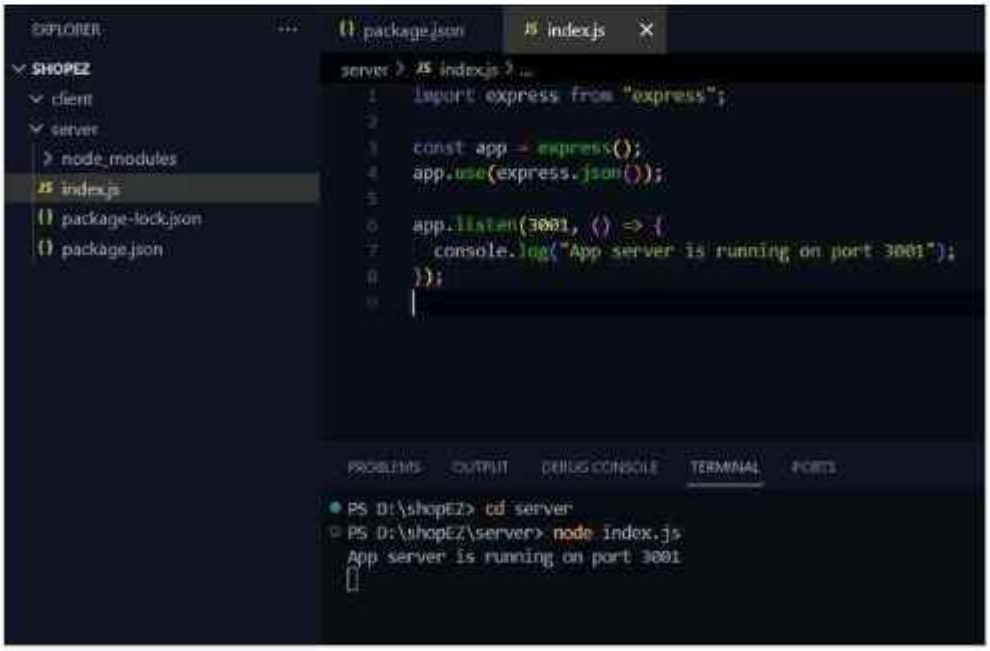


2. Setup Express Server:

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:

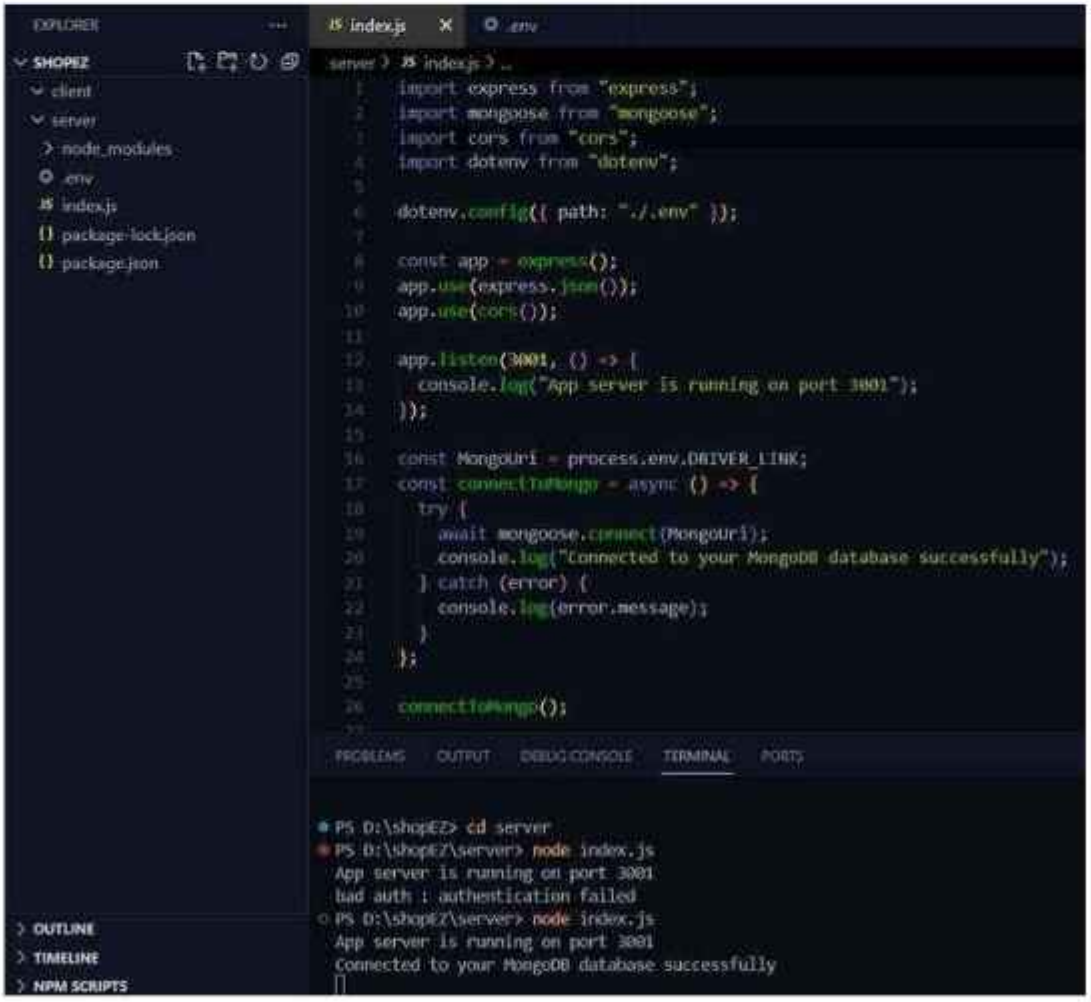
  MONGO_URI=your_mongodb_connection_string

- 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



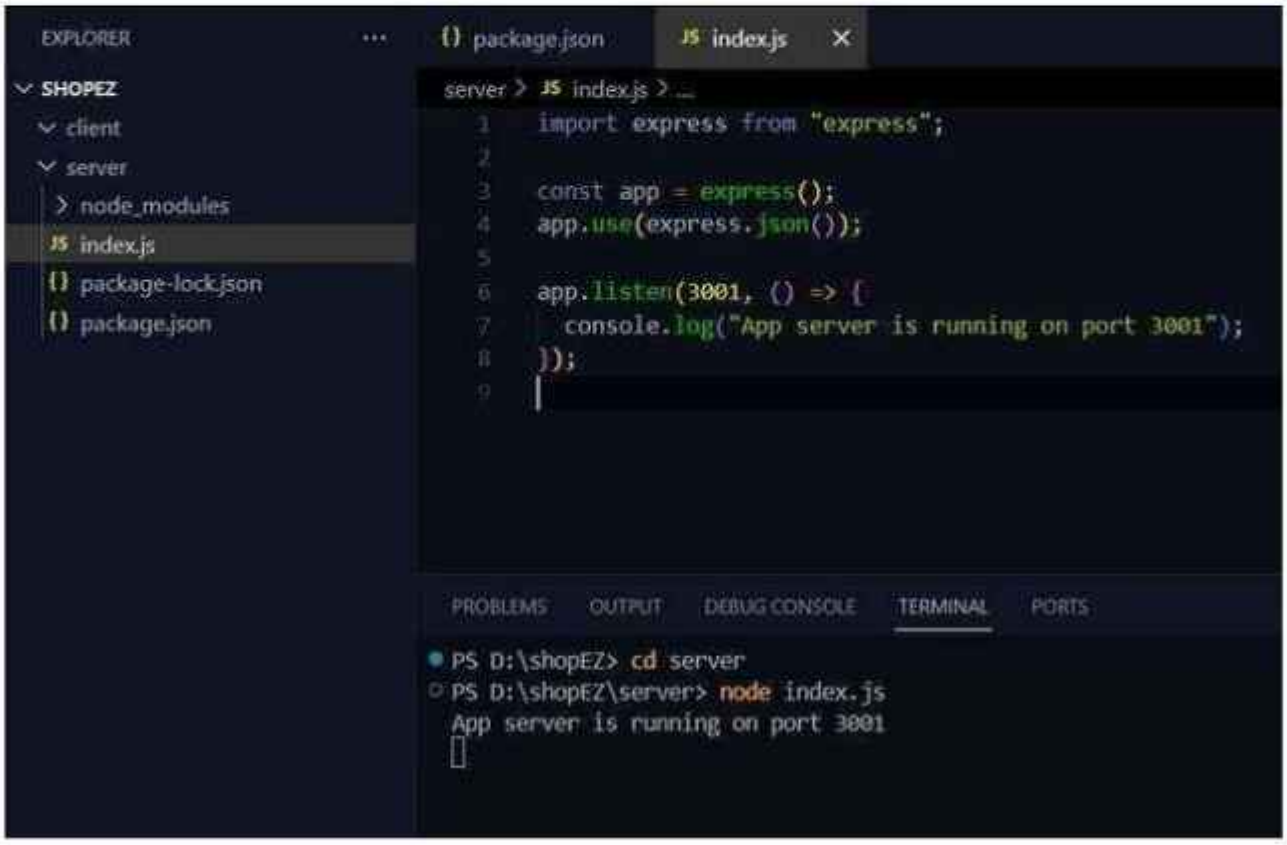
4. Create Express Routes:

- 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;
```



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.

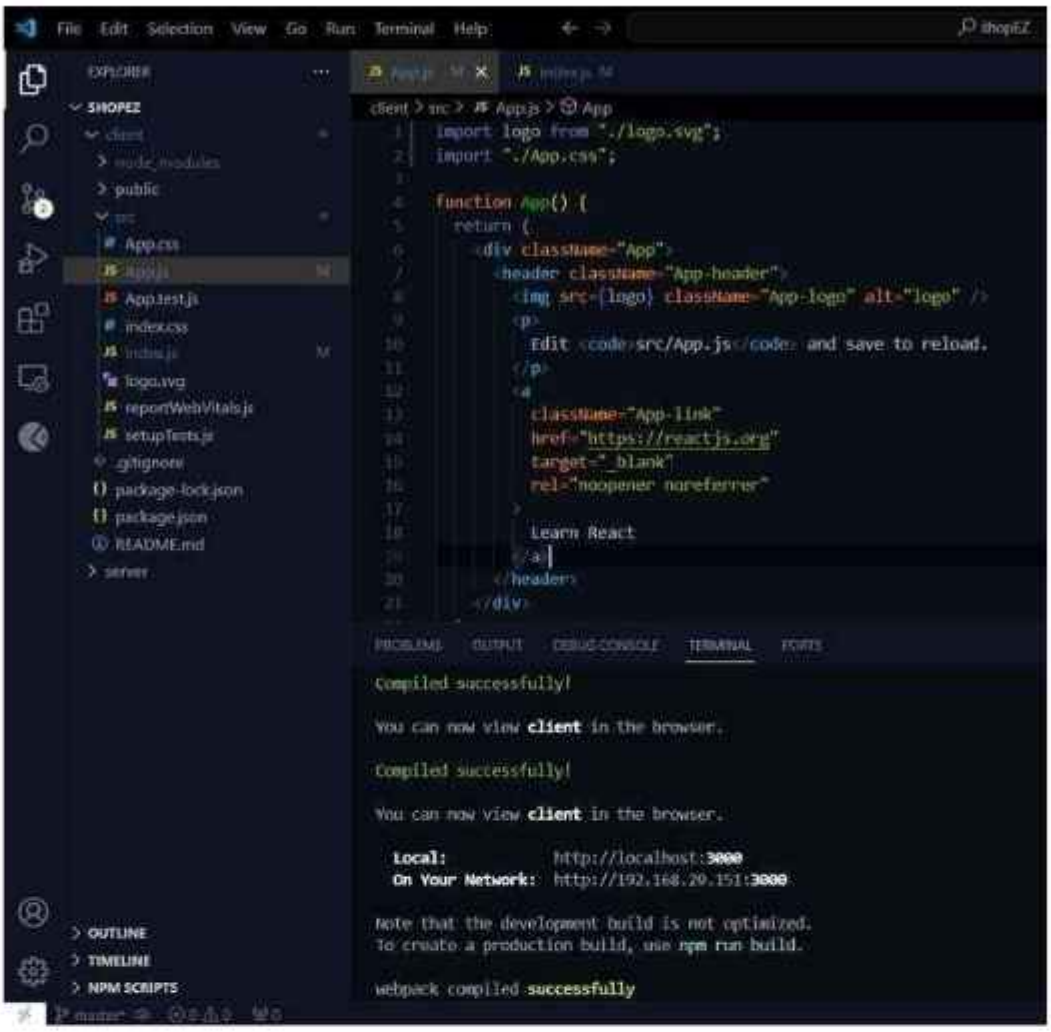
2. Design UI Components:

- 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



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

```
index.js
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(()=>{
```

Reference Im

Code Explanation:

Server setup:

User Authentication:

Backend

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

index.js

server > index.js > then() callback

```
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' });
    }
});
```

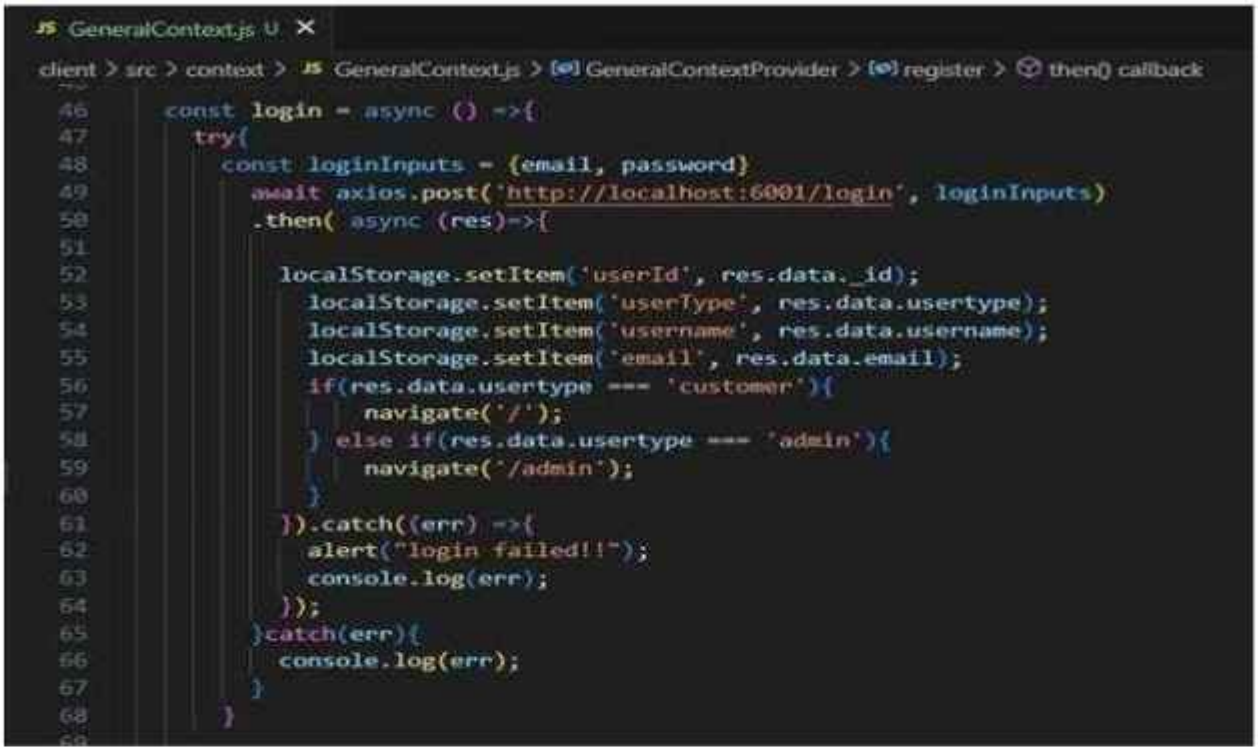
index.js

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

```
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:



Logout:

Register:

```
GeneralContext.js U
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):

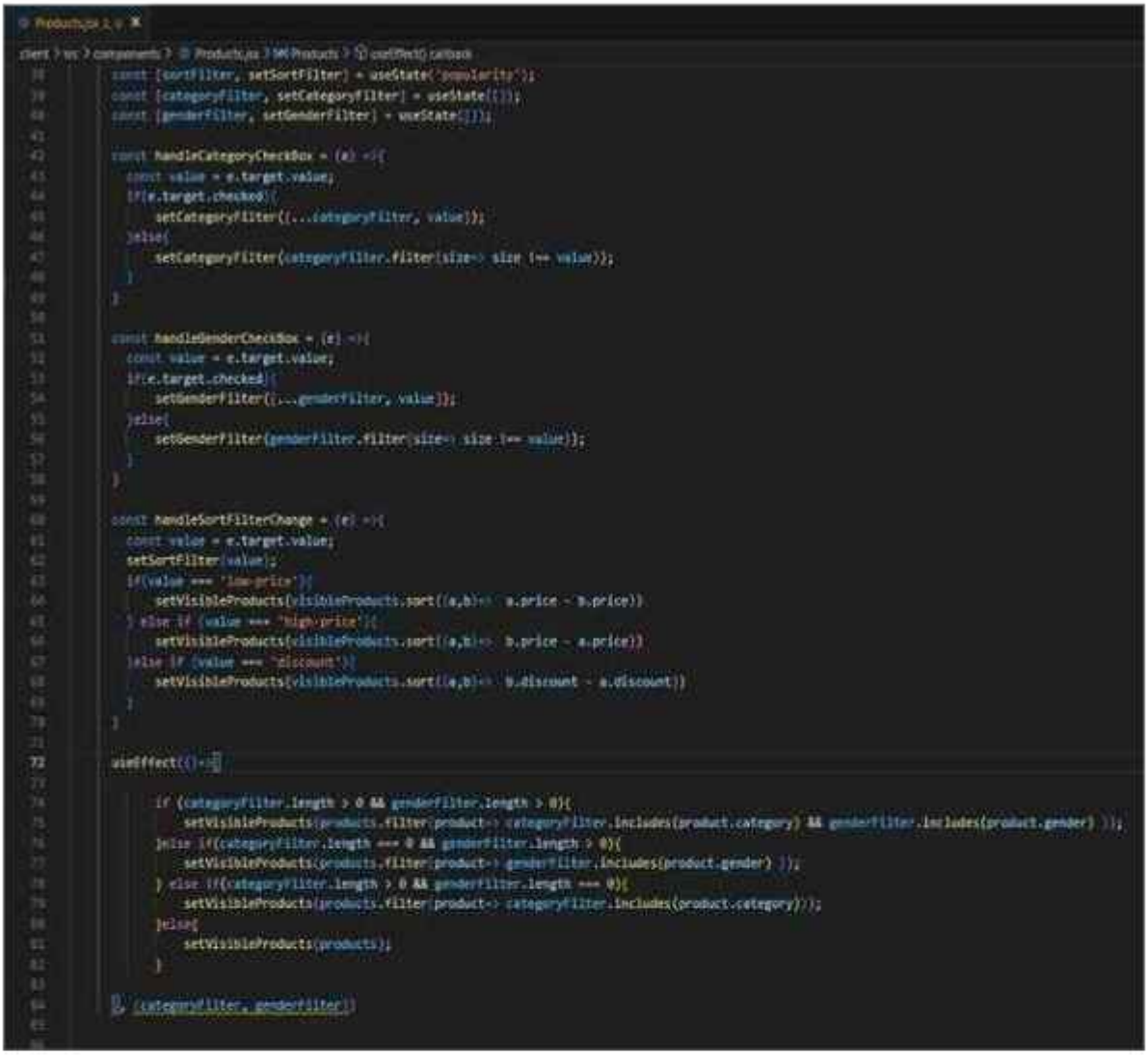
Frontend

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

Fetching food items:

```
IndividualRestaurant.jsx 4, U
client > src > pages > customer > IndividualRestaurant.jsx > IndividualRestaurant > handleCategoryCheckBox
33    const fetchRestaurants = async() =>{
34      await axios.get(`http://localhost:6001/fetch-restaurant/${id}`).then(
35        (response)=>{
36          setRestaurant(response.data);
37          console.log(response.data)
38        }
39      ).catch((err)=>{
40          console.log(err);
41      })
42    }
43
44    const fetchCategories = async () =>{
45      await axios.get('http://localhost:6001/fetch-categories').then(
46        (response)=>{
47          setAvailableCategories(response.data);
48        }
49      )
50    }
51
52    const fetchItems = async () =>{
53      await axios.get('http://localhost:6001/fetch-items').then(
54        (response)=>{
55          setItems(response.data);
56          setVisibleItems(response.data);
57        }
58      )
59    }
```

Filtering products:



```
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.

```
server > JS index.js > then() callback > app.get('/fetch-banner') callback
100
101     // fetch products
102
103     app.get('/fetch-products', async(req, res)=>{
104         try{
105             const products = await Product.find();
106             res.json(products);
107
108         }catch(err){
109             res.status(500).json({ message: 'Error occured' });
110         }
111     })
```

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

```
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:

index.js

server > 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 occured"});
416       }
417   })
418
```

Order products:

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

Frontend

Cart.jsx 2, U

client > src > pages > customer > Cart.jsx > 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

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 occured"});
456       }
457   })
```

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

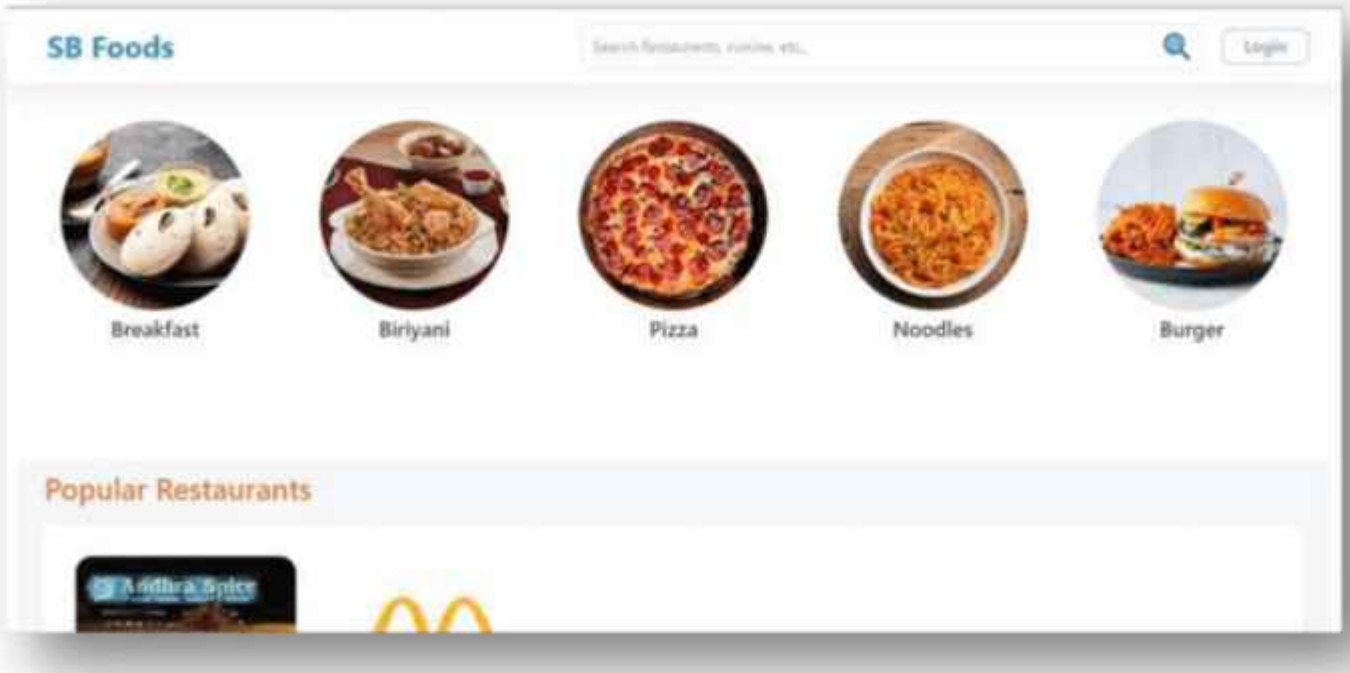
server > index.js > then() callback

```
// 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"});
    }
})
```

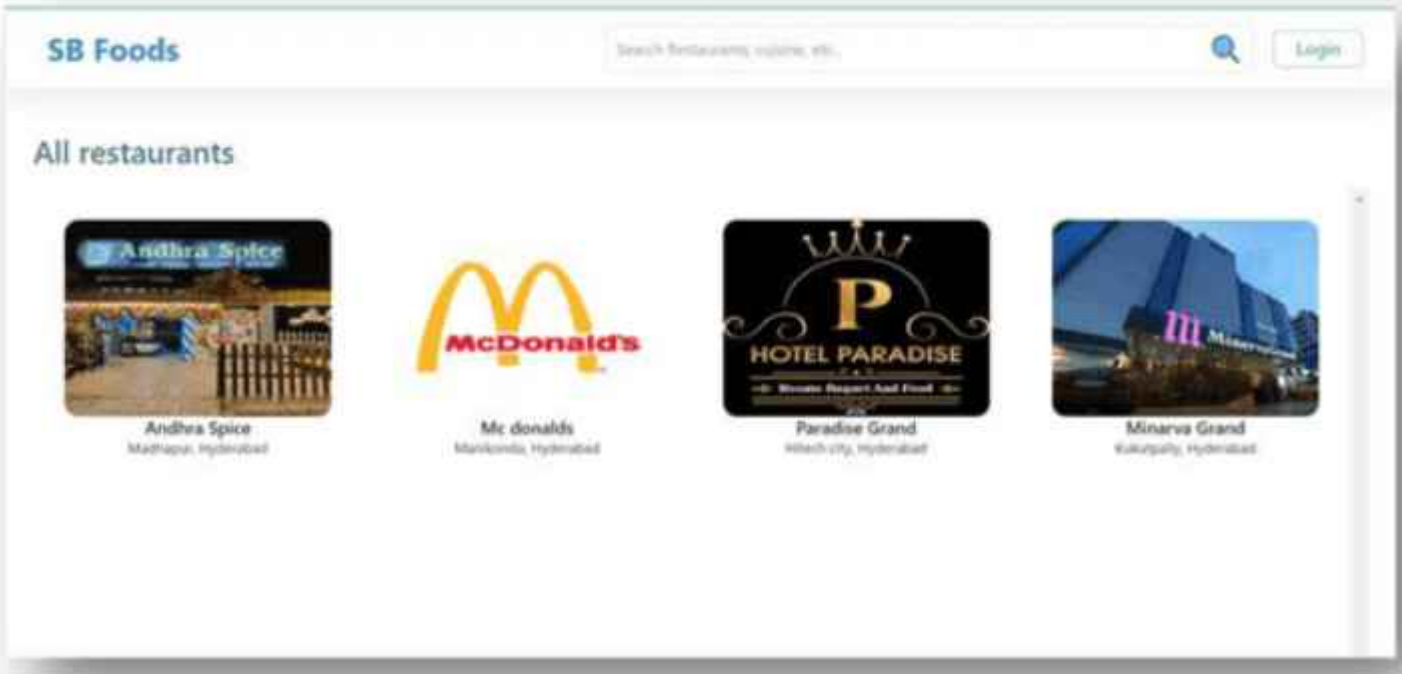
Along with this, implement additional features to view all orders, products, etc., in the admin dashboard.

Project Implementation & Execution

Landing page

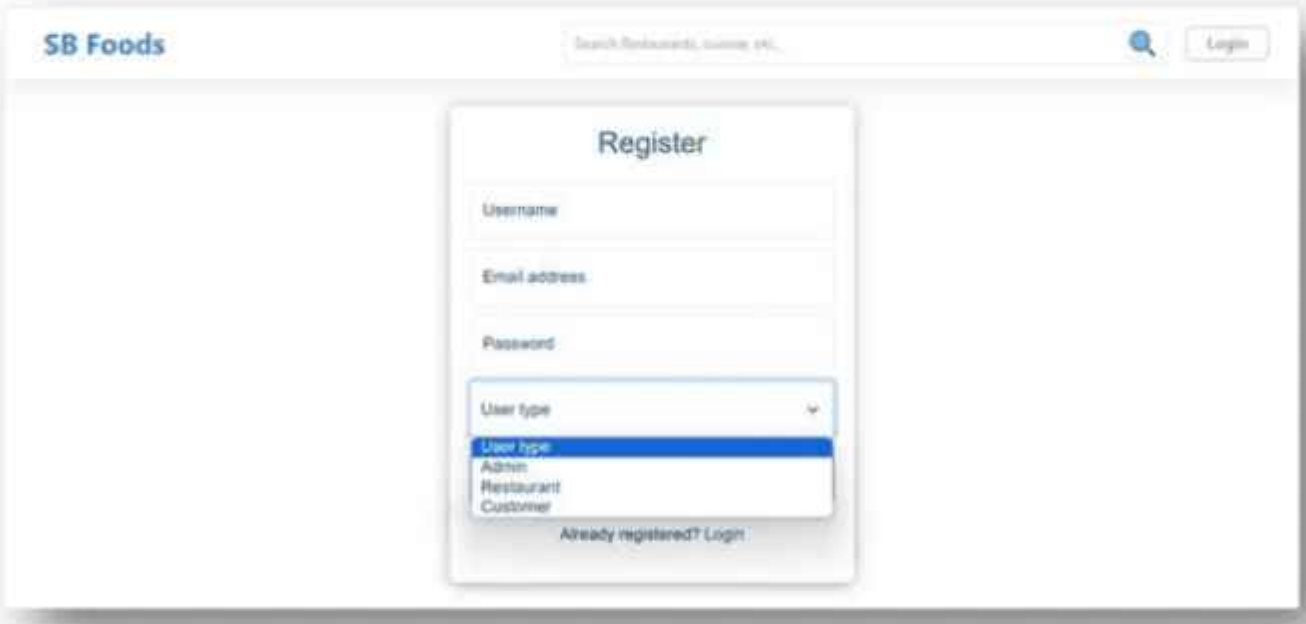


Landing page



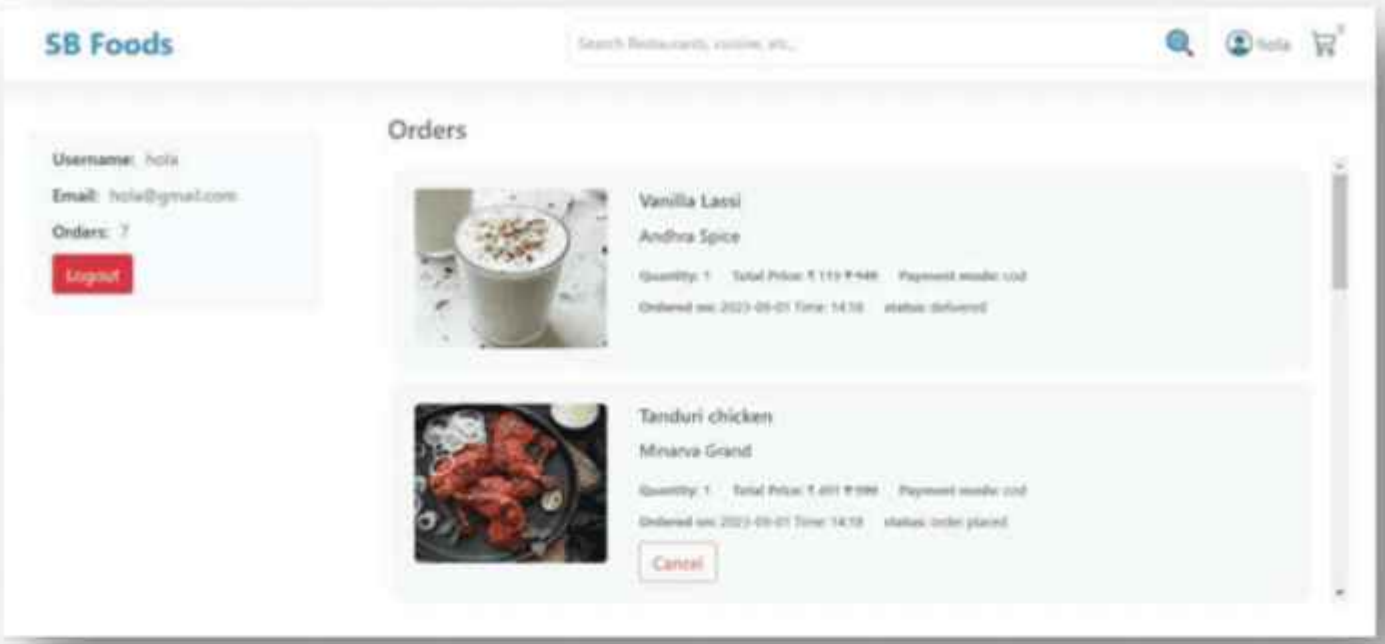
Restaurant Menu

Authentication

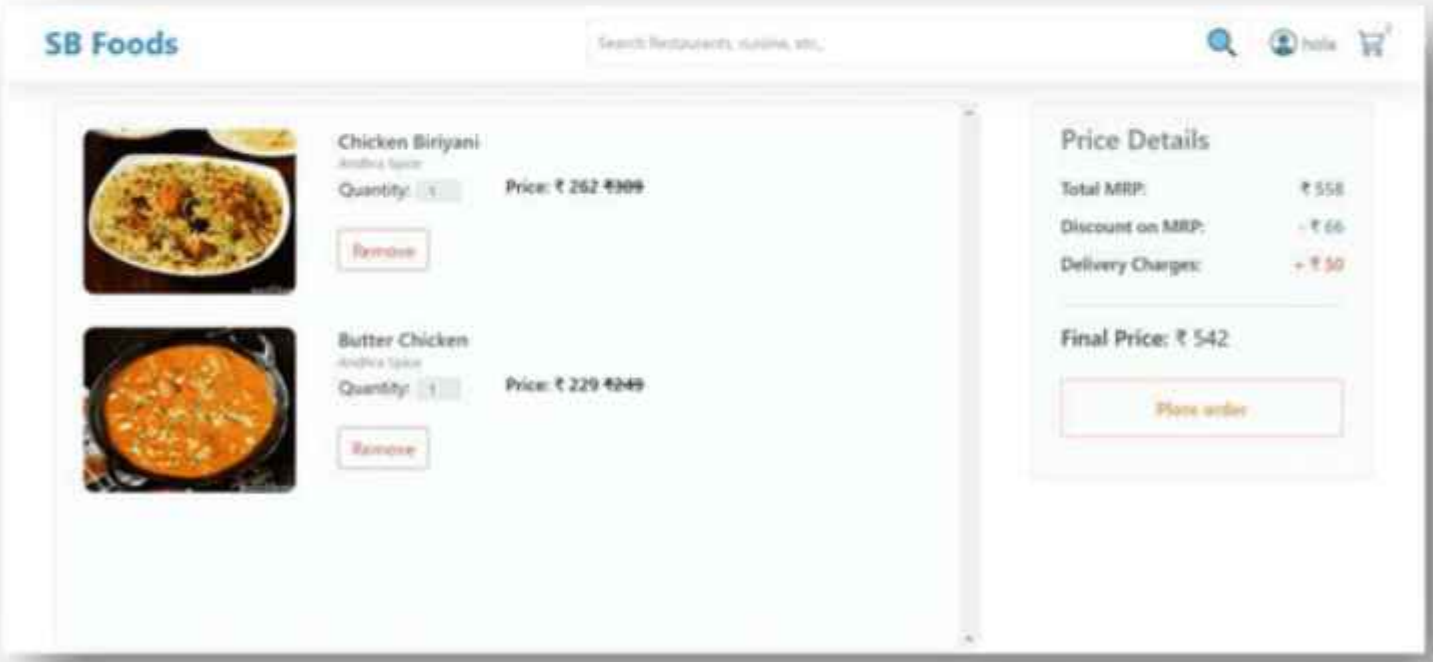


Smart Internz

User Profile

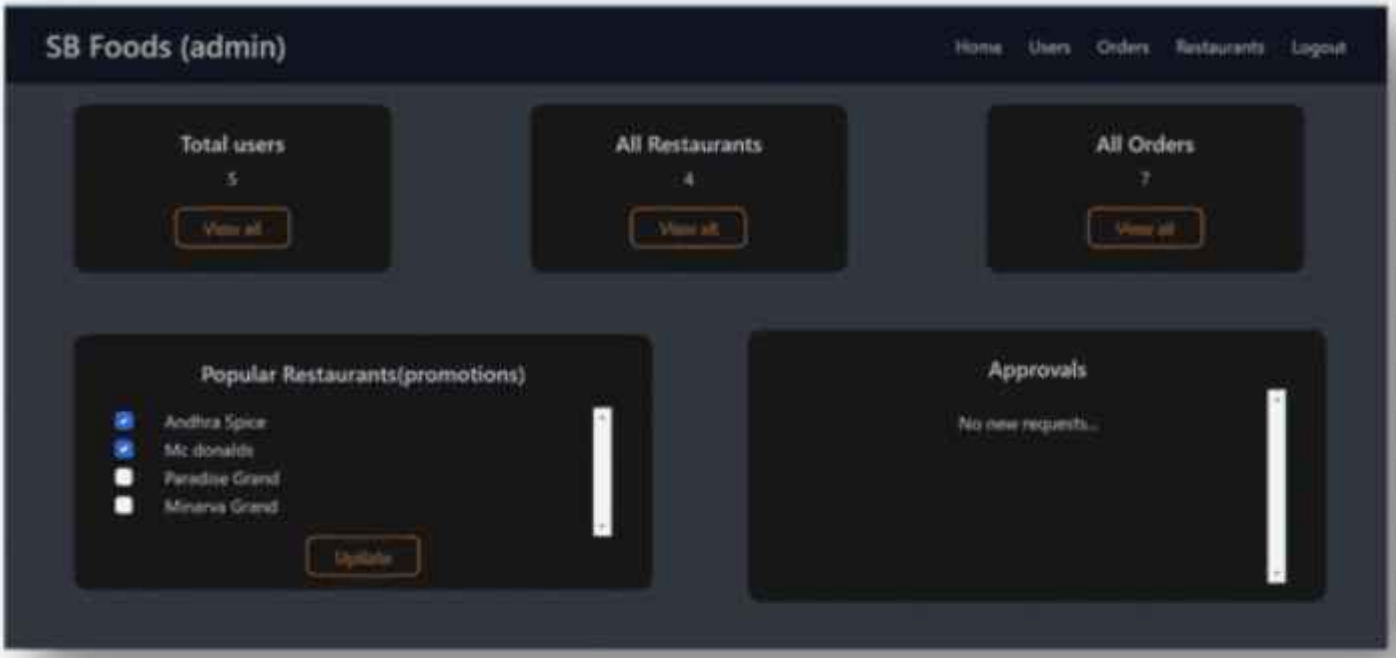


Cart

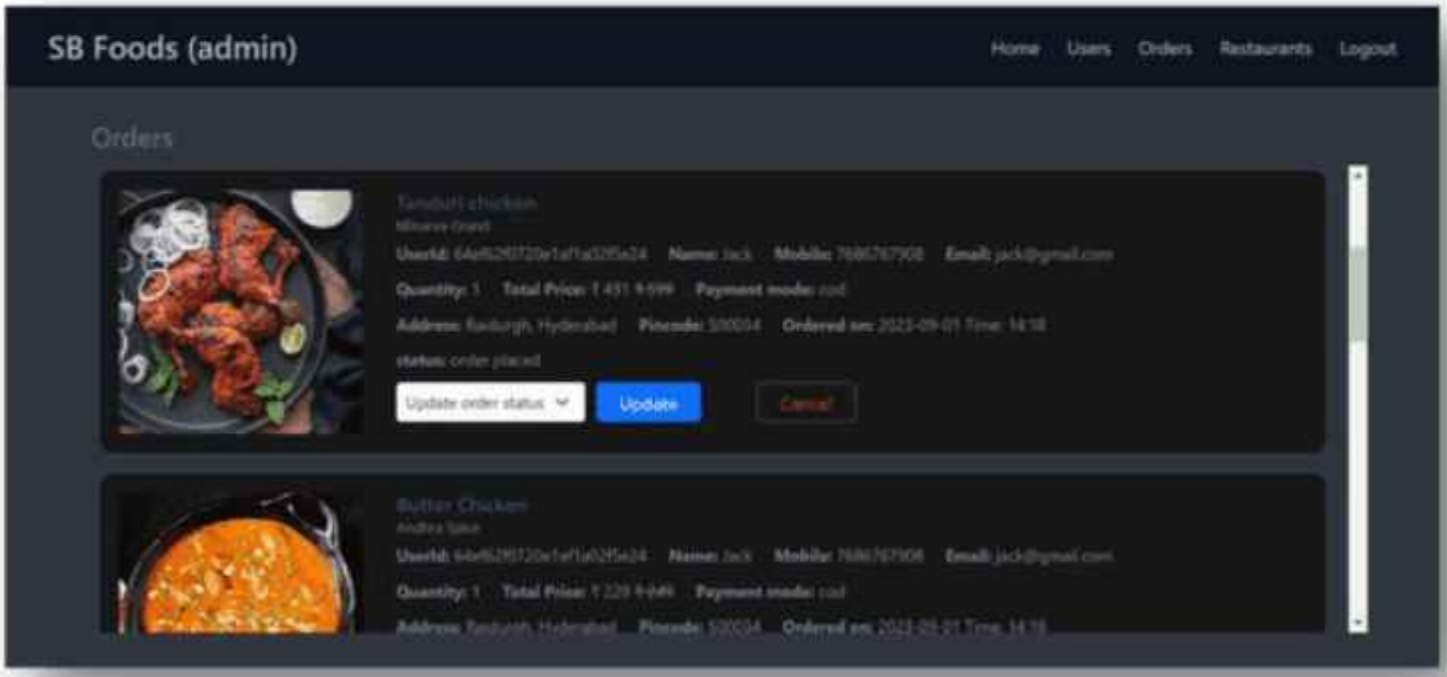


Smart Internz

Admin dashboard

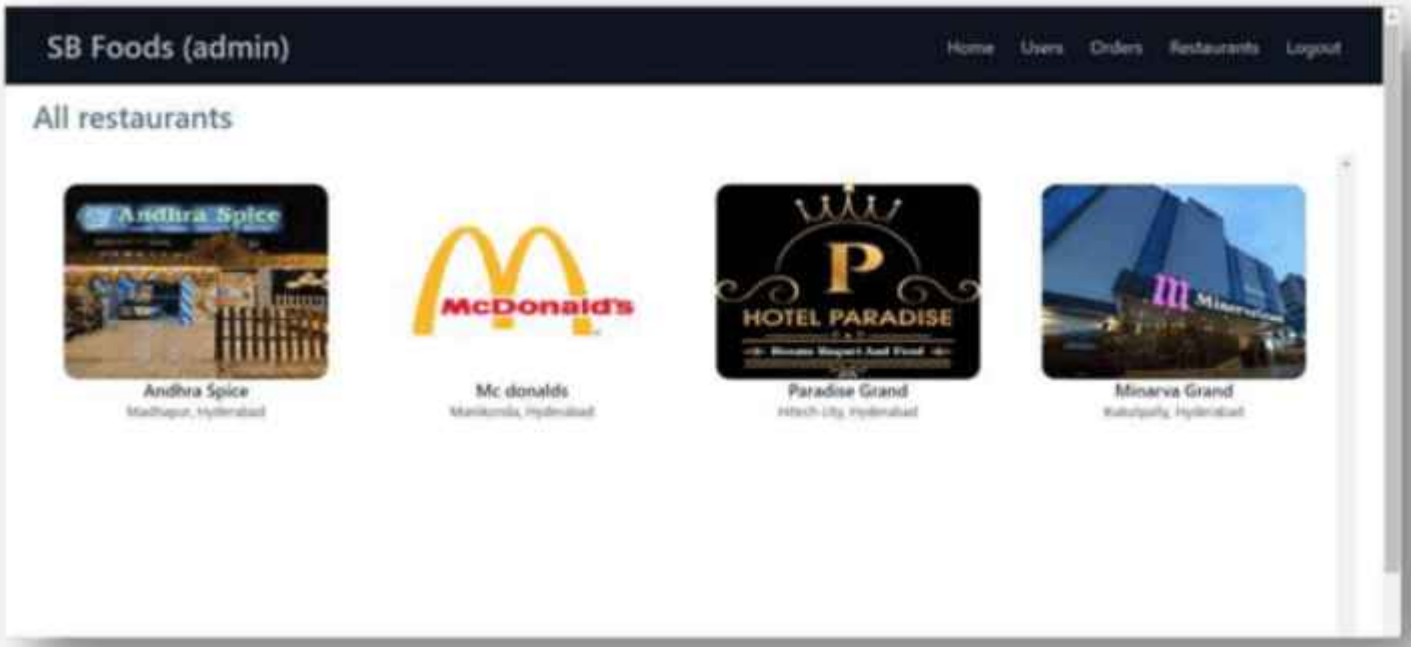


All Orders

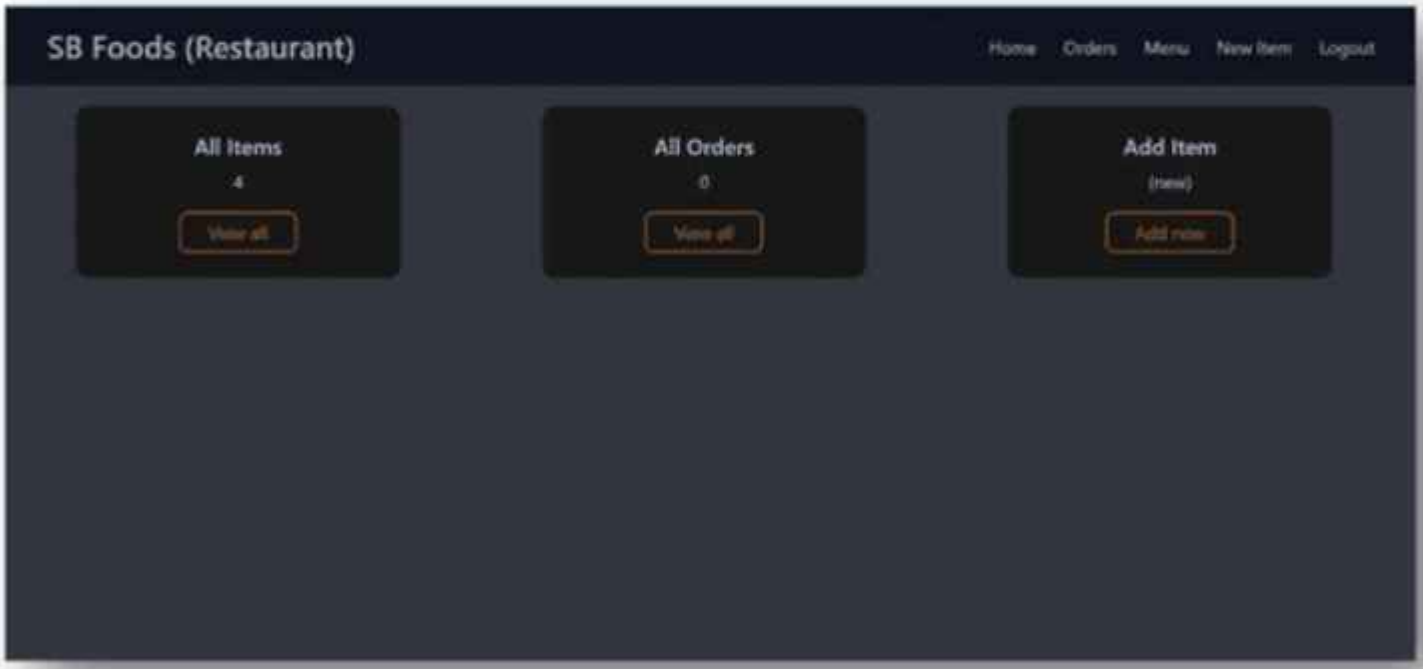


Smart Internz

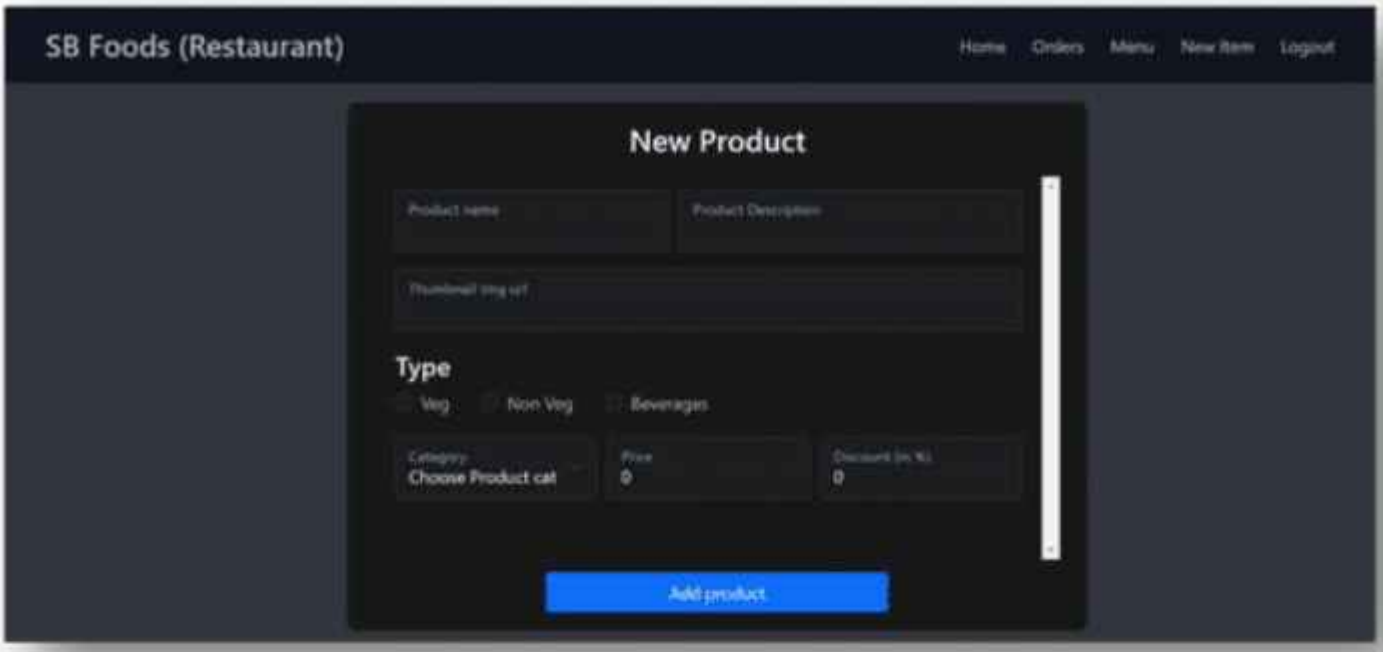
All restaurants



Restaurant Dashboard



New Item



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 internsh coordination team for offering this exceptional opportunity to gain rea world industry experience. The well-structured learning modules, expert mentorship, and continuous support provided by the platform played 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.

Smart Internz