

Orderontheego-your-on-demand-food-ordering-solution Application

INTRODUCTION

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

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

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

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

SCENARIO:

Late-Night Craving Resolution

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

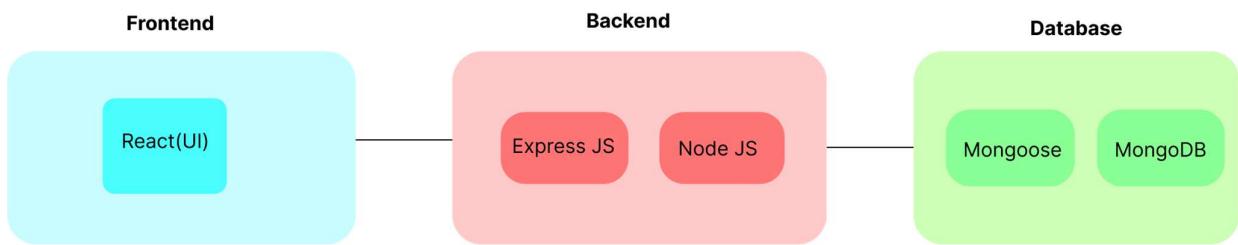
Solution with Food Ordering App:

1. Lisa opens the Food Ordering App on her smartphone and navigates to the late-night delivery section, where she finds a variety of eateries still open for orders.
2. She scrolls through the options, browsing menus and checking reviews until she spots her favorite local diner offering comfort food classics.

3. Lisa selects a hearty bowl of chicken noodle soup and a side of garlic bread, craving warmth and satisfaction in each bite.
4. With a few taps, she adds the items to her cart, specifies her delivery address, and chooses her preferred payment method.
5. Lisa double-checks her order details on the confirmation page, ensuring everything looks correct, before tapping the "Place Order" button.
6. Within minutes, she receives a notification confirming her order and estimated delivery time, allowing her to continue working with peace of mind.
7. As promised, the delivery arrives promptly at her doorstep, and Lisa eagerly digs into her piping hot meal, grateful for the convenience and comfort provided by the Food Ordering App during her late-night study session.

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

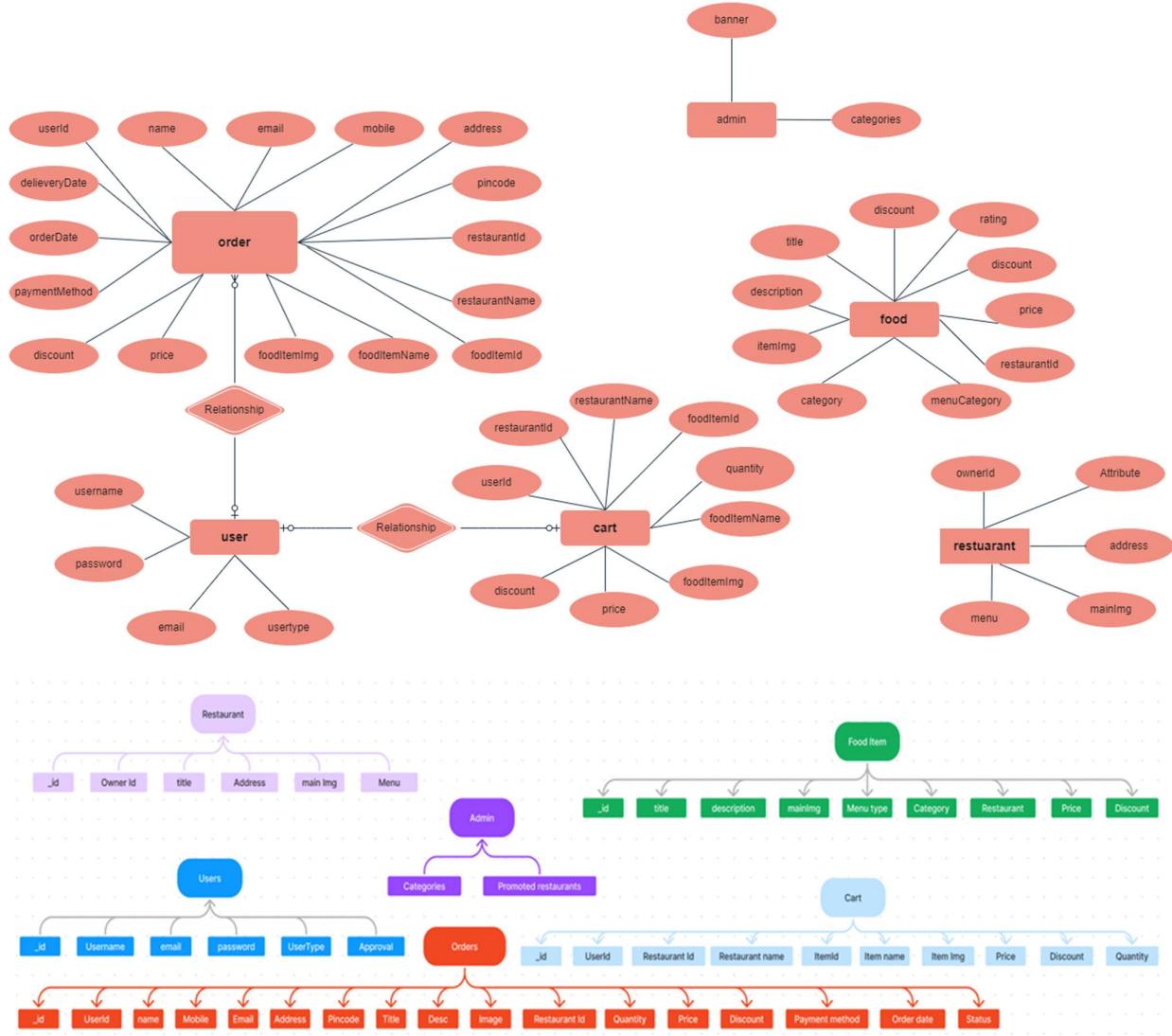
TECHNICAL ARCHITECTURE:



In this architecture diagram:

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

ER DIAGRAM:



The SB Foods ER-diagram represents the entities and relationships involved in an food ordering e-commerce system. It illustrates how users, restaurants, products, carts, and orders are interconnected. Here is a breakdown of the entities and their relationships:

User: Represents the individuals or entities who are registered in the platform.

Restaurant: This represents the collection of details of each restaurant in the platform.

Admin: Represents a collection with important details such as promoted restaurants and Categories.

Products: Represents a collection of all the food items available in the platform.

Cart: This collection stores all the products that are added to the cart by users. Here, the elements in the cart are differentiated by the user Id.

Orders: This collection stores all the orders that are made by the users in the platform.

FEATURES:

1. **Comprehensive Product Catalog:** SB Foods boasts an extensive catalog of food items from various restaurants, offering a diverse range of items and options for shoppers. You can effortlessly explore and discover various products, complete with detailed descriptions, customer reviews, pricing, and available discounts, to find the perfect food for your hunger.
2. **Order Details Page:** Upon clicking the "Shop Now" button, you will be directed to an order details page. Here, you can provide relevant information such as your shipping address, preferred payment method, and any specific product requirements.
3. **Secure and Efficient Checkout Process:** SB Foods guarantees a secure and efficient checkout process. Your personal information will be handled with the utmost security, and we strive to make the purchasing process as swift and trouble-free as possible.
4. **Order Confirmation and Details:** After successfully placing an order, you will receive a confirmation notification. Subsequently, you will be directed to an order details page, where you can review all pertinent information about your order, including shipping details, payment method, and any specific product requests you specified.

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

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

PREREQUISITES:

To develop a full-stack food ordering app using React JS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server

- side. • Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

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

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>

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

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

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

To run the existing SB Foods App project downloaded from github:

Follow below steps:

Clone the repository:

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

Git clone: <https://github.com/subbu-vishnu-86/orderonthego-your-on-demand-food-ordering-solution.git>

Install Dependencies:

- Navigate into the cloned repository directory:
cd Food-Ordering-App-MERN
- Install the required dependencies by running the following command:
npm install

Start the Development Server:

- To start the development server, execute the following command:
npm run dev or npm run start
- The e-commerce app will be accessible at <http://localhost:3000> by default.
You can change the port configuration in the .env file if needed.

Access the App:

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the flight booking app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the SB Foods app on your local machine. You can now proceed with further customization, development, and

testing as needed.

USER & ADMIN FLOW:

1. User Flow:

- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform.
- Users can add the products they wish to their carts and order.
- They can then proceed by entering address and payment details.
- After ordering, they can check them in the profile section.

2. Restaurant Flow:

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

3. Admin Flow:

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

PROJECT STRUCTURE

```
FOOD ORDERING SYSTEM      ⌂ ⌃ ⌚
client
  node_modules
  public
src
  components
    Footer.jsx
    Login.jsx
    Navbar.jsx
    PopularRestaurants.jsx
    Register.jsx
    Restaurants.jsx
  context
  images
pages
  admin
    Admin.jsx
    AllOrders.jsx
    AllProducts.jsx
    AllRestaurants.jsx
    AllUsers.jsx
  customer
    Cart.jsx
    CategoryProducts.jsx
    IndividualRestaurant.jsx
    Profile.jsx
  restaurant
    EditProduct.jsx
    NewProduct.jsx
    RestaurantHome.jsx
    RestaurantMenu.jsx
    RestaurantOrders.jsx
    Authentication.jsx
    Home.jsx
  styles
# App.css
JS App.js
```

```
✓ server
  > node_modules
  JS index.js
  {} package-lock.json
  {} package.json
  JS Schema.js
```

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>

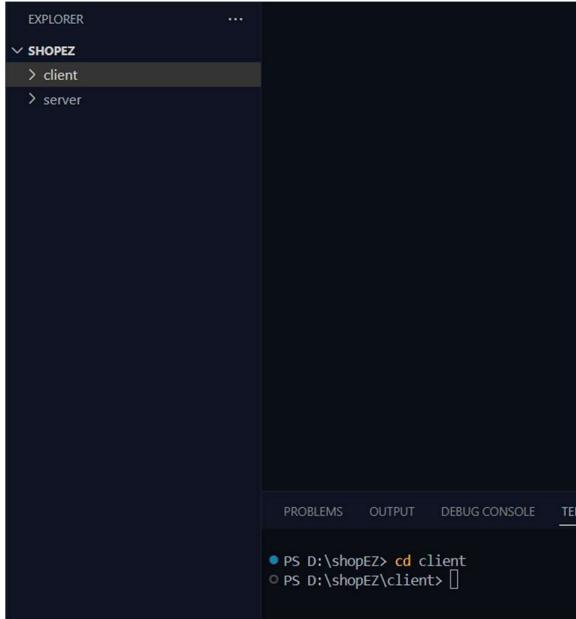
Create project folders and files:

- Client folders.
- Server folders

Referral Video Link:

https://drive.google.com/file/d/1uSMbPIAR6rfAEMcb_nLZAZd5QIjTpnyQ/view?usp=sharing

Referral Image:



DATABASE DEVELOPMENT:

Create database in cloud video link:-

<https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLPOh-Bu2bXhq7A3/view>

- Install Mongoose.
- Create database connection.

Reference Video of connect node with mongoDB database:

https://drive.google.com/file/d/1cTS3_EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

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

Reference Image:

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with a tree view of the project structure under the 'SHOPEZ' folder. The main area shows the 'index.js' file with code for setting up an Express app and connecting to MongoDB. Below the code editor is the Terminal tab, which displays command-line logs from running the application.

```
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 Schema.js X
server > JS Schema.js > [e] 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
```

```
JS Schema.js X
server > JS 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
```

BACKEND DEVELOPMENT:

Set Up Project Structure:

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

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

Reference Image:

The screenshot shows a dark-themed code editor interface. On the left is the Explorer sidebar with a project structure for 'SHOPEZ' containing 'client', 'server' (which has 'node_modules'), and 'package-lock.json'. Below these are two 'package.json' files, one for the client and one for the server. The right side shows the content of the 'server/package.json' file. The terminal at the bottom shows npm install commands for various packages like express, mongoose, body-parser, and dotenv.

```
EXPLORER          ...          {} package.json ×
✓ SHOPEZ          ⌂ ⌂ ⌂ ⌂
  > client
  ✓ server
    > node_modules
  {} package-lock.json
  {} package.json

server > {} package.json > {} dependencies
1  {
2   "name": "server",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   ▷ Debug
7   "scripts": {
8     "test": "echo \\\"Error: no test specified\\\" && exit 1"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "bcrypt": "^5.1.1",
15    "body-parser": "^1.20.2",
16    "cors": "^2.8.5",
17    "dotenv": "^16.4.5",
18    "express": "^4.19.1",
19    "mongoose": "^8.2.3"
20  }
21

PROBLEMS      OUTPUT      DEBUG CONSOLE      TERMINAL      PORTS

● PS D:\shopEZ\server> npm install express mongoose body-parser dotenv
added 85 packages, and audited 86 packages in 1s

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
```

1. Setup express server:

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

Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRi02qXS/view?usp=sharing

Reference Image:

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure for 'SHOPEZ' with 'client', 'server', and 'node_modules' folders, along with 'index.js', 'package-lock.json', and 'package.json' files. The 'index.js' file is selected and shown in the main editor area with the following code:

```
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
```

Below the editor, the 'TERMINAL' tab is active, showing command-line output:

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

2. Database Configuration:

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

Reference Video of connect node with mongoDB database:

https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

Reference Article:

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

Reference Image:

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure for "SHOPEZ" with folders "client" and "server", and files ".env", "index.js", "package-lock.json", and "package.json".
- Code Editor:** Displays the content of "index.js". The code sets up an Express.js application, configures middleware (body-parser and cors), and connects to a MongoDB database using mongoose.
- Terminal:** Shows the command-line output of running the application. It includes the command "PS D:\shopEZ> cd server", the execution of "node index.js", and the resulting log message: "App server is running on port 3001" followed by "bad auth : authentication failed". A successful connection attempt is also shown: "PS D:\shopEZ\server> node index.js" followed by "Connected to your MongoDB database successfully".

3. Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRi02qXS/view?usp=sharing

Reference Image:

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure for "SHOPEZ" with "client" and "server" folders. Inside "server", there are "node_modules", "index.js", "package-lock.json", and "package.json".
- Code Editor:** The file "index.js" is open, displaying the following code:

```
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
```
- Terminal:** Shows the command "PS D:\shopEZ> cd server" followed by the output of "node index.js" which prints "App server is running on port 3001".

4. Define API Routes:

- Create separate route files for different API functionalities such as users, orders, and authentication.
- Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

- Define Mongoose schemas for the different data entities like products, users, and orders.
- Create corresponding Mongoose models to interact with the MongoDB database.
 - Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

- Create routes and middleware for user registration, login, and logout.
- Set up authentication middleware to protect routes that require user

authentication.

7. Handle new products and Orders:

- Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.
- Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

- Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

FRONTEND DEVELOPMENT:

1. Setup React Application:

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

2. Design UI components:

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

3. Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Reference Video Link:

<https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing>

Reference Article Link:

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

Reference Image:

The screenshot shows a VS Code interface with the following details:

- File Explorer (Left):** Shows a project structure for "SHOPEZ" with branches for "client", "node_modules", "public", and "src". Inside "src", files include "App.css", "App.js", "App.test.js", "index.css", "index.js", "logo.svg", "reportWebVitals.js", "setupTests.js", ".gitignore", "package-lock.json", "package.json", and "README.md".
- Code Editor (Top Right):** Displays the content of "App.js". The code defines a functional component "App" that returns a div containing a header with a logo and a link to reactjs.org.
- Terminal (Bottom):** Shows the output of a build process:
 - "Compiled successfully!"
 - "You can now view **client** in the browser."
 - "Compiled successfully!"
 - "You can now view **client** in the browser."
 - Local host information: "Local: http://localhost:3000" and "On Your Network: http://192.168.29.151:3000"
 - Note: "Note that the development build is not optimized. To create a production build, use `npm run build`.
 - "webpack compiled **successfully**"

CODE EXPLANATION

Server setup:

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

```
JS index.js  X
server > JS 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
```

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
55
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

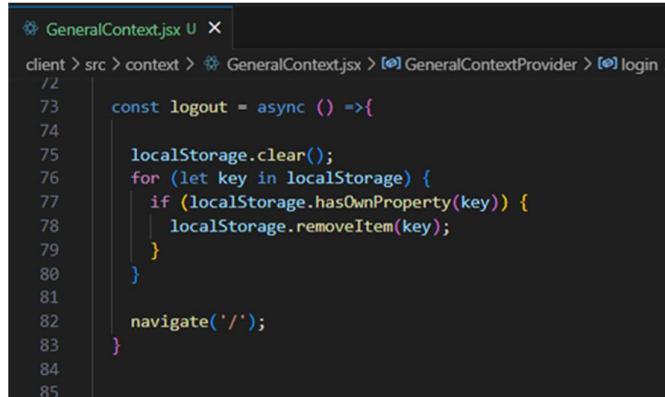
Login:

```

JS GeneralContext.js U X
client > src > context > JS GeneralContext.js > ⚡ GeneralContextProvider > ⚡ register > ⚡ 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

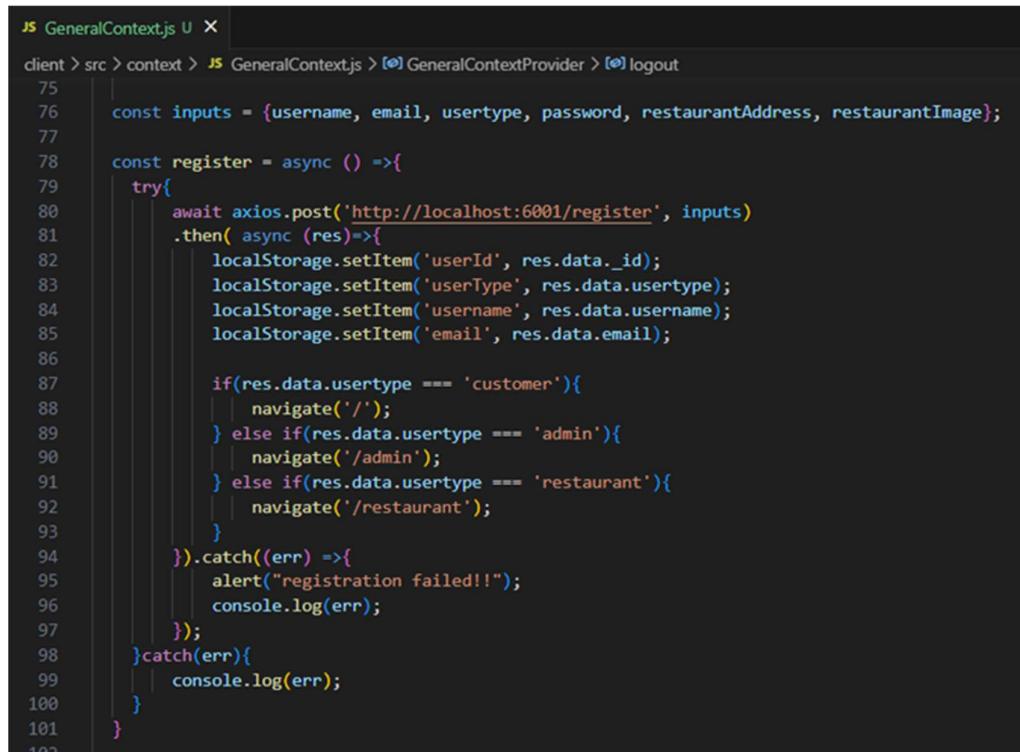
```

Logout:



```
client > src > context > GeneralContext.jsx > GeneralContextProvider > login >
/2
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:



```
JS 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   }
102 }
```

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:

```
client > src > pages > customer > IndividualRestaurant.jsx > 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   }
60 }
```

Filtering products:

```
client > src > components > Products.jsx > [o] Products > ⚡ useEffect() callback
  38 |   const [sortFilter, setSortFilter] = useState('popularity');
  39 |   const [categoryFilter, setCategoryFilter] = useState([]);
  40 |   const [genderFilter, setGenderFilter] = useState([]);
  41 |
  42 |   const handleCategoryCheckBox = (e) =>{
  43 |     const value = e.target.value;
  44 |     if(e.target.checked){
  45 |       setCategoryFilter([...categoryFilter, value]);
  46 |     }else{
  47 |       setCategoryFilter(categoryFilter.filter(size=> size !== value));
  48 |     }
  49 |   }
  50 |
  51 |   const handleGenderCheckBox = (e) =>{
  52 |     const value = e.target.value;
  53 |     if(e.target.checked){
  54 |       setGenderFilter([...genderFilter, value]);
  55 |     }else{
  56 |       setGenderFilter(genderFilter.filter(size=> size !== value));
  57 |     }
  58 |   }
  59 |
  60 |   const handleSortFilterChange = (e) =>{
  61 |     const value = e.target.value;
  62 |     setSortFilter(value);
  63 |     if(value === 'low-price'){
  64 |       setVisibleProducts(visibleProducts.sort((a,b)=> a.price - b.price))
  65 |     } else if (value === 'high-price'){
  66 |       setVisibleProducts(visibleProducts.sort((a,b)=> b.price - a.price))
  67 |     }else if (value === 'discount'){
  68 |       setVisibleProducts(visibleProducts.sort((a,b)=> b.discount - a.discount))
  69 |     }
  70 |   }
  71 |
  72 |   useEffect(()=>[
  73 |
  74 |     if (categoryFilter.length > 0 && genderFilter.length > 0){
  75 |       setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category) && genderFilter.includes(product.gender) ));
  76 |     }else if(categoryFilter.length === 0 && genderFilter.length > 0){
  77 |       setVisibleProducts(products.filter(product=> genderFilter.includes(product.gender) ));
  78 |     } else if(categoryFilter.length > 0 && genderFilter.length === 0){
  79 |       setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category)));
  80 |     }else{
  81 |       setVisibleProducts(products);
  82 |     }
  83 |
  84 |   ], [categoryFilter, genderFilter])
  85 |
  86 |
```

- **Backend**

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

```
JS index.js X
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 occurred' });
110        }
111    })

```

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 > [e] 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

```
⚙️ Cartjsx 2, U  X
client > src > pages > customer > ⚙️ Cartjsx > [?] 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.

```

JS index.js  X
server > JS 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 }
458

```

Add new product:

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

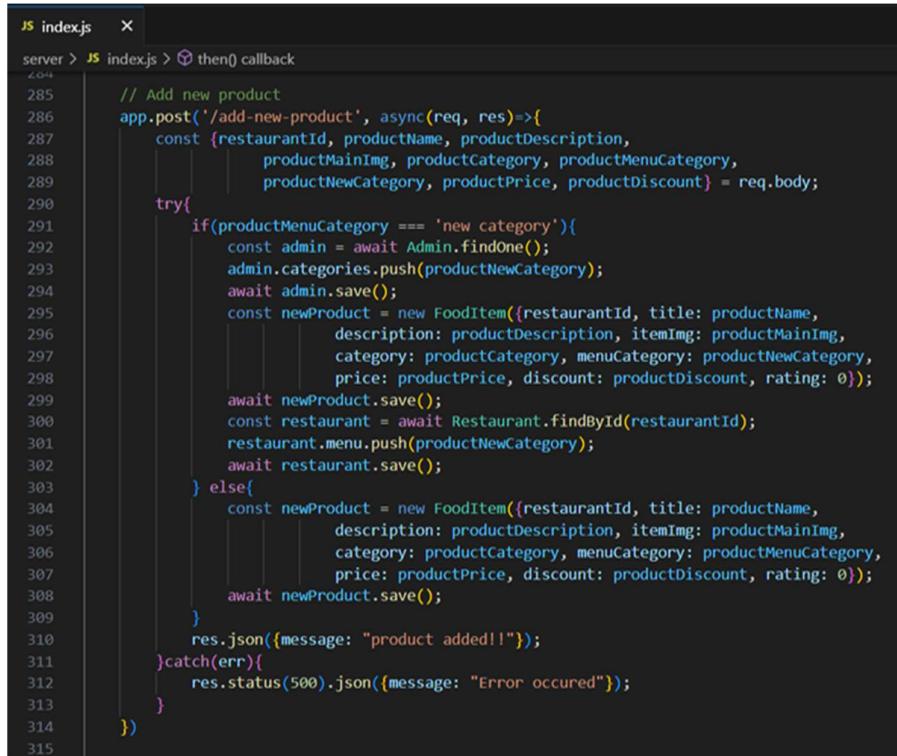
- Frontend:

```

JS NewProduct.jsx 1, U  X
client > src > pages > restaurant > ⚡ NewProduct.jsx > [●] 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         setDescription('');
54         setMainImg('');
55         setCategory('');
56         setMenuCategory('');
57         setNewCategory('');
58         setPrice(0);
59         setDiscount(0);
60         navigate('/restaurant-menu');
61     }
62   )
63 }
64

```

- Backend:



```
JS index.js
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.

Demo UI images:

- **Landing page**

SB Foods

Search Restaurants, cuisine, etc.



Login



Breakfast



Biryani



Pizza



Noodles



Burger

Popular Restaurants



Restaurants

SB Foods

Search Restaurants, cuisine, etc.



Login

All restaurants



Andhra Spice
Madhapur, Hyderabad



Mc donalds
Manikonda, Hyderabad



Paradise Grand
Hitech city, Hyderabad



Minerva Grand
Kukutpally, Hyderabad

Restaurant Menu

SB Foods

Search Restaurants, cuisine, etc.

hola

Mc donalds

Manikonda, Hyderabad

All Items

Filters

Sort By

- Popularity
- low-price
- high-price
- Discount
- Rating

Food Type

- Veg
- Non Veg
- Beverages

Categories

- Biriyani
- Curry



Mc Maharaj
Big size burger with chic...
₹ 175 209
[Add item](#)



French fries
Long French fries made fr...
₹ 134 149
[Add item](#)



Cold Coffee
Tinder coffee made from s...
₹ 201 249
[Add item](#)



Chicken Pizza
Crispy pizza with tasty c...
₹ 314 349
[Add item](#)

Authentication

SB Foods

Search Restaurants, cuisine, etc.

[Login](#)

Register

Username

Email address

Password

User type

- Admin
- Restaurant
- Customer

Already registered? [Login](#)

User Profile

SB Foods

Search Restaurants, cuisine, etc.



hola



Username: hola

Email: hola@gmail.com

Orders: 7

[Logout](#)

Orders



Vanilla Lassi

Andhra Spice

Quantity: 1 Total Price: ₹ 119 ₹149 Payment mode: cod

Ordered on: 2023-09-01 Time: 14:18 status: delivered



Tanduri chicken

Minarva Grand

Quantity: 1 Total Price: ₹ 491 ₹599 Payment mode: cod

Ordered on: 2023-09-01 Time: 14:18 status: order placed

[Cancel](#)

Cart

SB Foods

Search Restaurants, cuisine, etc.



hola



Chicken Biriyani

Andhra Spice

Quantity: 1

Price: ₹ 262 ₹309

[Remove](#)



Butter Chicken

Andhra Spice

Quantity: 1

Price: ₹ 229 ₹249

[Remove](#)

Price Details

Total MRP: ₹ 558

Discount on MRP: - ₹ 66

Delivery Charges: + ₹ 50

Final Price: ₹ 542

[Place order](#)

Admin dashboard

SB Foods (admin)

Home Users Orders Restaurants Logout

Total users 5 [View all](#)

All Restaurants 4 [View all](#)

All Orders 7 [View all](#)

Popular Restaurants(promotions)

- Andhra Spice
- Mc donalds
- Paradise Grand
- Minarva Grand

[Update](#)

Approvals

No new requests...

- **All Orders**

SB Foods (admin)

Home Users Orders Restaurants Logout

Orders

Tanduri chicken
Minarva Grand

UserId: 64ef62f0720e1af1a02f5e24 Name: Jack Mobile: 7686767908 Email: jack@gmail.com

Quantity: 1 Total Price: ₹ 491 ₹ 599 Payment mode: cod

Address: Raidurgh, Hyderabad Pincode: 500034 Ordered on: 2023-09-01 Time: 14:18

status: order placed

[Update order status](#) [Update](#) [Cancel](#)

Butter Chicken
Andhra Spice

UserId: 64ef62f0720e1af1a02f5e24 Name: Jack Mobile: 7686767908 Email: jack@gmail.com

Quantity: 1 Total Price: ₹ 229 ₹ 249 Payment mode: cod

Address: Raiduroh, Hyderabad Pincode: 500034 Ordered on: 2023-09-01 Time: 14:18

- **All restaurants**

All restaurants



Andhra Spice
Madhapur, Hyderabad



Mc donalds
Manikonda, Hyderabad



Paradise Grand
Hitech city, Hyderabad



Minarva Grand
Kukutpally, Hyderabad

• Restaurant Dashboard

All Items

4

[View all](#)

All Orders

0

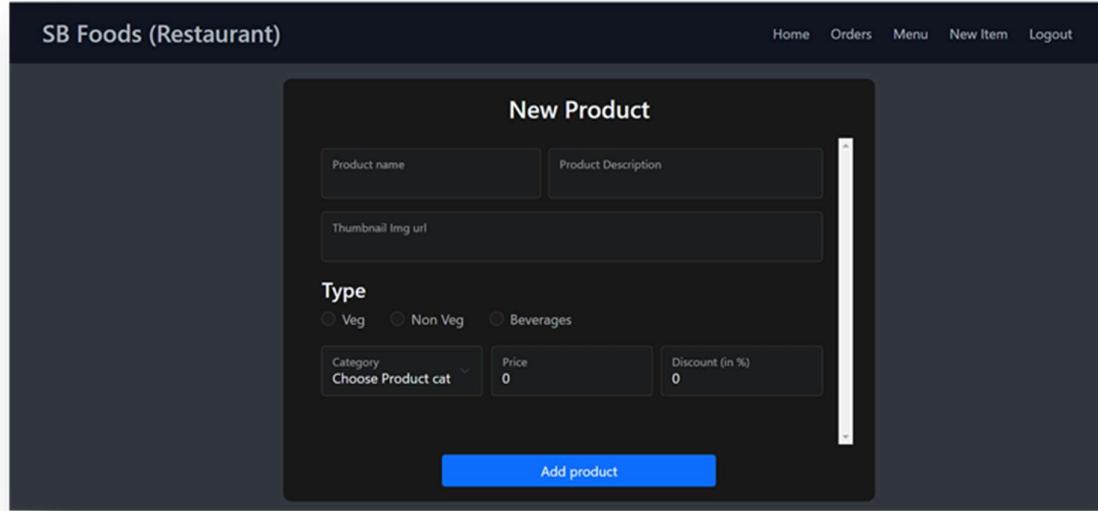
[View all](#)

Add Item

(new)

[Add now](#)

• New Item



For any further doubts or help, please consider the GitHub repo,

<https://github.com/subbu-vishnu-86/orderonthego-your-on-demand-food-ordering-solution.git>

The demo of the app is available at:

<https://drive.google.com/drive/folders/1NYmJvWeBi5gpn0oTtMzSCFOrFJOYe-ZZ>

** Happy Coding **