

FOOD ORDERING 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.

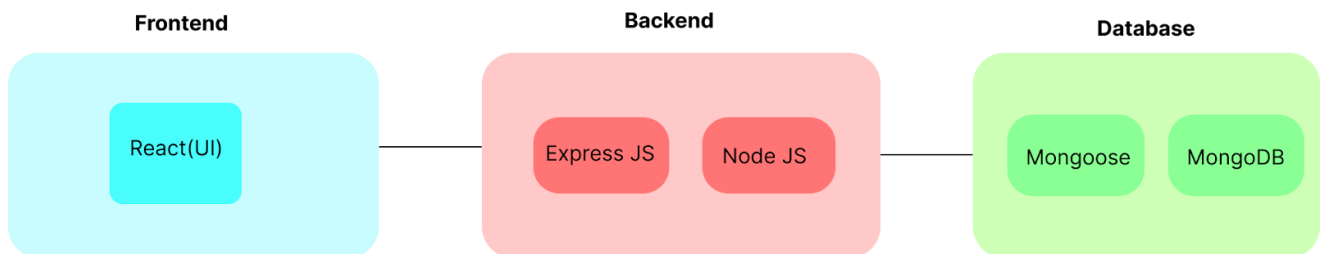
FOOD ORDERING APPLICATION

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:

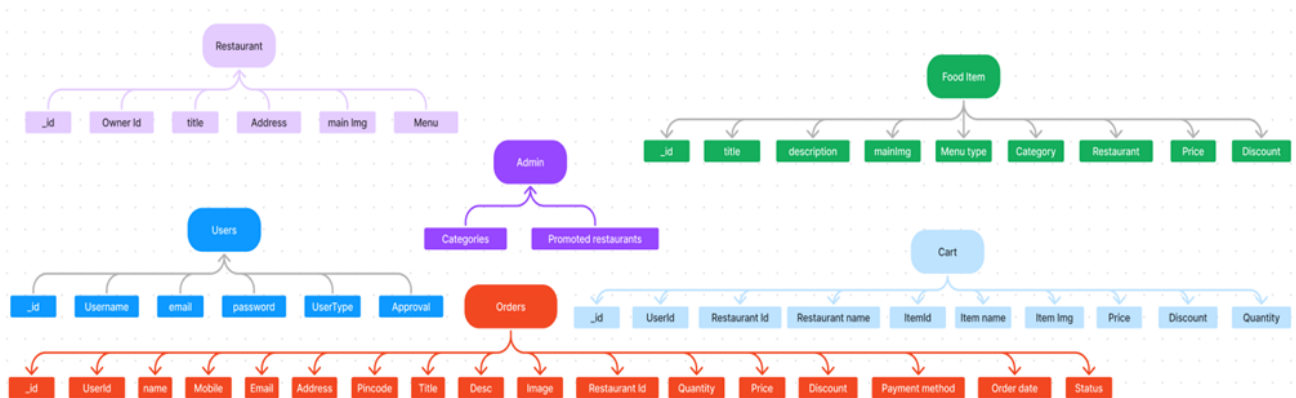
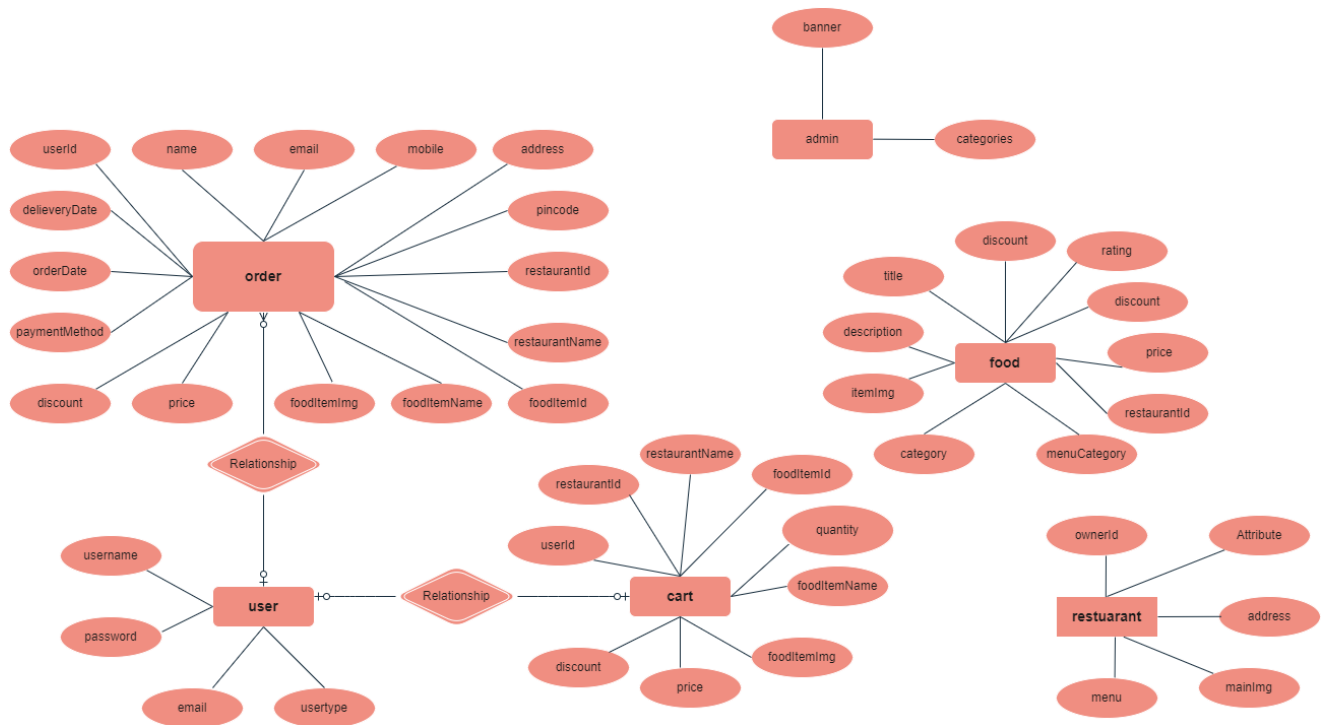


In this architecture diagram:

- 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:

FOOD ORDERING APPLICATION



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:

FOOD ORDERING APPLICATION

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>

FOOD ORDERING APPLICATION

- 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.

FOOD ORDERING APPLICATION

- 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/harsha-varadhan-reddy-07/Food-Ordering-App-MERN>

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:

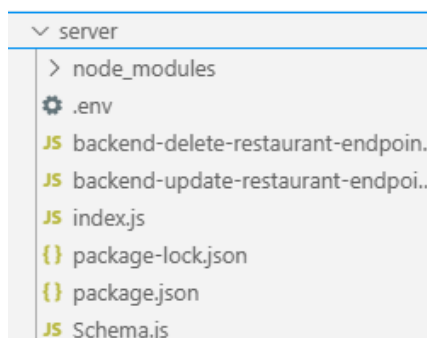
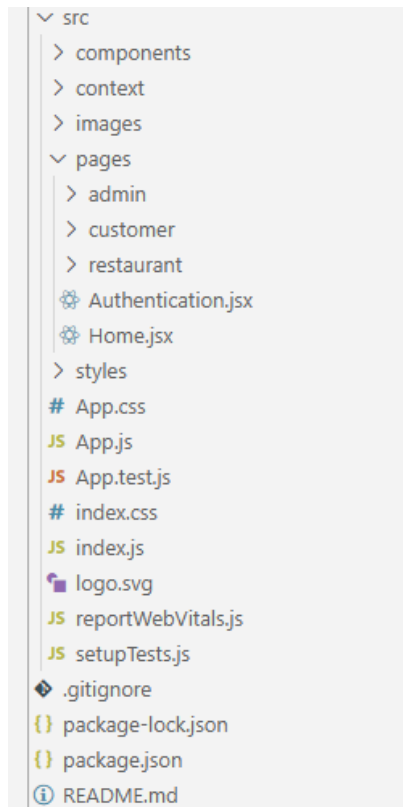
- Restaurants start by authenticating with their credentials.
- They need to get approval from the admin to start listing the products.
- They can add/edit the food items.

FOOD ORDERING APPLICATION

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



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

- `src/components`: Contains components related to the application such as, register, login, home, etc.,
- `src/pages` has the files for all the pages in the application.

FOOD ORDERING 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>

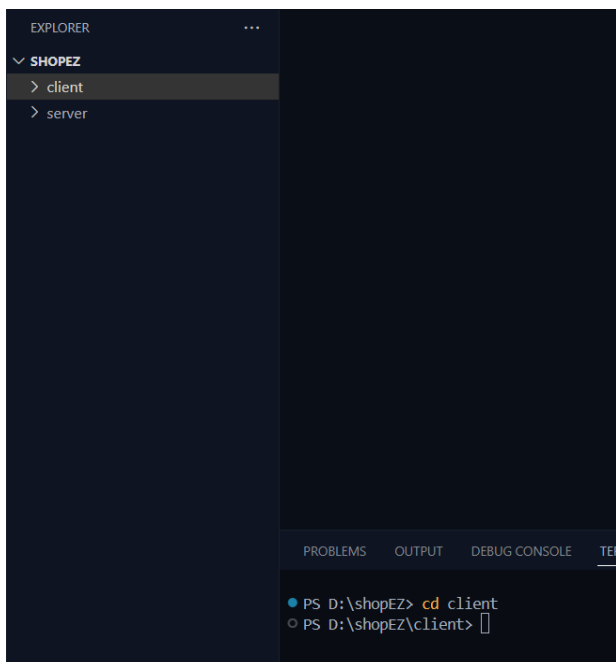
Create project folders and files:

- Client folders.
- Server folders

Referral Video Link:

https://drive.google.com/file/d/1uSMbPIAR6rfAEMcb_nLZAZd5QljTpnYQ/view?usp=sharing

Referral Image:



DATABASE DEVELOPMENT:

Create database in cloud video link:-

<https://drive.google.com/file/d/1CQiI5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view>

- Install Mongoose.
- Create database connection.

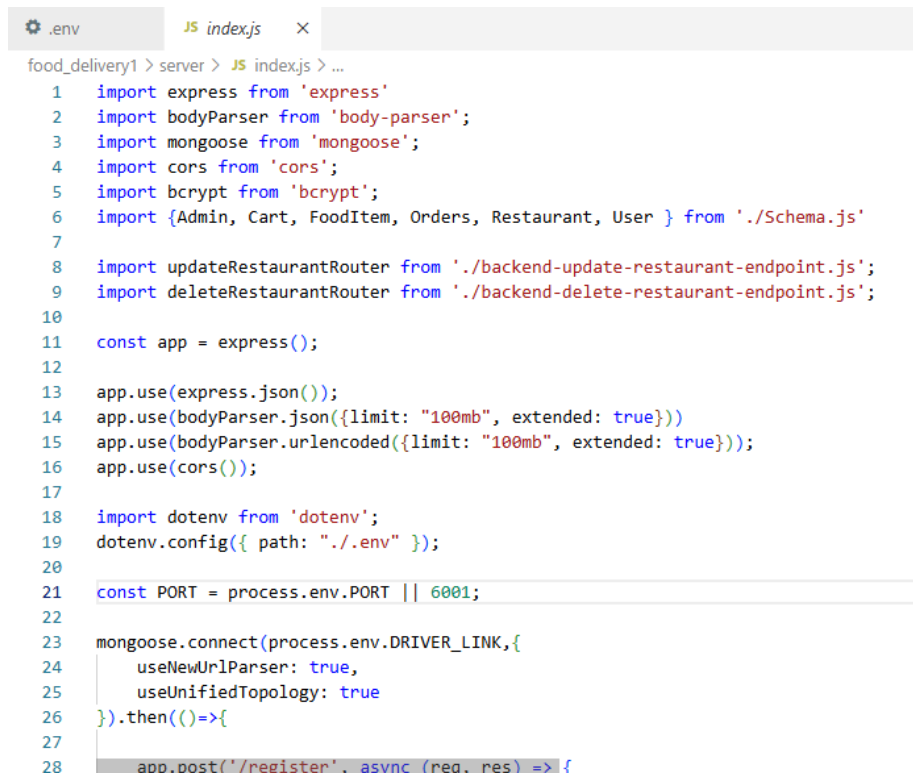
FOOD ORDERING APPLICATION

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:



```
.env JS index.js ×
food_delivery1 > 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 import updateRestaurantRouter from './backend-update-restaurant-endpoint.js';
9 import deleteRestaurantRouter from './backend-delete-restaurant-endpoint.js';
10
11 const app = express();
12
13 app.use(express.json());
14 app.use(bodyParser.json({limit: "100mb", extended: true}))
15 app.use(bodyParser.urlencoded({limit: "100mb", extended: true}));
16 app.use(cors());
17
18 import dotenv from 'dotenv';
19 dotenv.config({ path: "./.env" });
20
21 const PORT = process.env.PORT || 6001;
22
23 mongoose.connect(process.env.DRIVER_LINK,{
24   useNewUrlParser: true,
25   useUnifiedTopology: true
26 }).then(()=>{
27
28   app.post('/register', async (req, res) => {
```

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.

FOOD ORDERING APPLICATION

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

```
.env x JS Schema.js x
C:\Users\USER\Desktop\React JS\food_delivery\server\env hema
1 import mongoose from "mongoose";
2 const userSchema = new mongoose.Schema({
3   username: {type: String},
4   password: {type: String},
5   email: {type: String},
6   usertype: {type: String},
7   approval: {type: String}
8 });
9 const adminSchema = new mongoose.Schema({
10   categories: {type: Array},
11   promotedRestaurants: []
12 });
13 const restaurantSchema = new mongoose.Schema({
14   ownerId: {type: String},
15   title: {type: String},
16   description: {type: String},
17   address: {type: String},
18   mainImg: {type: String},
19   menu: {type: Array, default: []}
20 });
21 const foodItemSchema = new mongoose.Schema({
22   title: {type: String},
23   description: {type: String},
24   itemImg: {type: String},
25   category: {type: String}, //veg or non-veg or beverage
26   menuCategory: {type: String},
27   restaurantId: {type: String},
28   price: {type: Number},
```

FOOD ORDERING APPLICATION

```
.env JS Schema.js X
food_delivery1 > server > JS Schema.js > [o] orderSchema
32 const orderSchema = new mongoose.Schema({
33   userId: {type: String},
34   name: {type: String},
35   email: {type: String},
36   mobile: {type: String},
37   address: {type: String},
38   pincode: {type: String},
39   restaurantId: {type: String},
40   restaurantName: {type: String},
41   foodItemId: {type: String},
42   foodItemName: {type: String},
43   foodItemImg: {type: String},
44   quantity: {type: Number},
45   price: {type: Number},
46   discount: {type: Number},
47   paymentMethod: {type: String},
48   orderDate: {type: String},
49   orderStatus: {type: String, default: 'order placed'}
50 })
51 const cartSchema = new mongoose.Schema({
52   userId: {type: String},
53   restaurantId: {type: String},
54   restaurantName: {type: String},
55   foodItemId: {type: String},
56   foodItemName: {type: String},
57   foodItemImg: {type: String},
58   quantity: {type: Number},
```

BACKEND DEVELOPMENT:

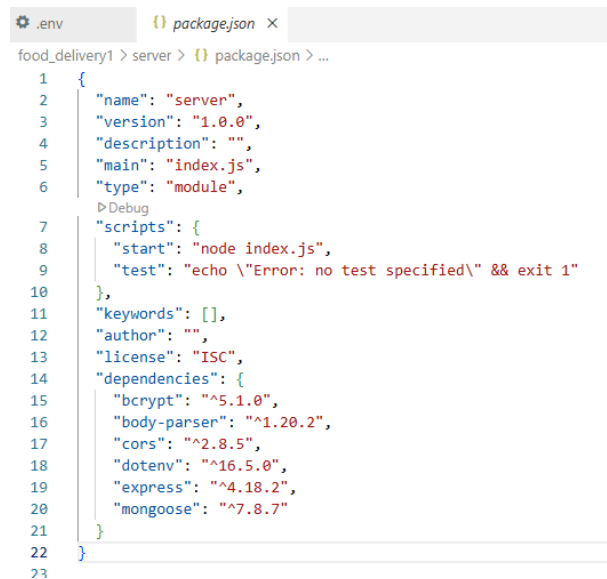
Set Up Project Structure:

- 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-gQK3DO6wr7ooAfJYlQwnemZoF/view?usp=sharing>

Reference Image:

FOOD ORDERING APPLICATION



```
.env {} package.json X
food_delivery1 > server > {} package.json > ...
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "type": "module",
7   "scripts": {
8     "start": "node index.js",
9     "test": "echo \"Error: no test specified\" && exit 1"
10  },
11  "keywords": [],
12  "author": "",
13  "license": "ISC",
14  "dependencies": {
15    "bcrypt": "^5.1.0",
16    "body-parser": "^1.20.2",
17    "cors": "^2.8.5",
18    "dotenv": "^16.5.0",
19    "express": "^4.18.2",
20    "mongoose": "^7.8.7"
21  }
22 }
```

1. Setup express server:

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

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

Reference Image:

```
app.use(updateRestaurantRouter);
app.use(deleteRestaurantRouter);

app.listen(PORT, ()=>{
  console.log('running @ 6001');
})
```

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

FOOD ORDERING APPLICATION

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

Reference Image:

```
import express from 'express'
import bodyParser from 'body-parser';
import mongoose from 'mongoose';
import cors from 'cors';
import bcrypt from 'bcrypt';
import {Admin, Cart, FoodItem, Orders, Restaurant, User } from './Schema.js'

import updateRestaurantRouter from './backend-update-restaurant-endpoint.js';
import deleteRestaurantRouter from './backend-delete-restaurant-endpoint.js';

const app = express();

app.use(express.json());
app.use(bodyParser.json({limit: "100mb", extended: true}))
app.use(bodyParser.urlencoded({limit: "100mb", extended: true}));
app.use(cors());

import dotenv from 'dotenv';
dotenv.config({ path: "./.env" });

const PORT = process.env.PORT || 6001;

mongoose.connect(process.env.DRIVER_LINK,{
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()=>{
```

3. Create Express.js Server:

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

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

Reference Image:

```
const PORT = process.env.PORT || 6001;

mongoose.connect(process.env.DRIVER_LINK,{
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(()=>{
```

4. Define API Routes:

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

FOOD ORDERING APPLICATION

- 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.

FOOD ORDERING APPLICATION

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/1EokogagcLMUGilluwHGYQo65x8GRpDcP/view?usp=sharing>

Reference Article Link:

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

Reference Image:

```
function App() {  
  return (  
    <div className="App">  
      <Navbar />  
      <Routes>  
        <Route path="/auth" element={Authentication} /> />  
        <Route exact path="/" element={Home} /> />  
        <Route path="/cart" element={Cart} /> />  
        <Route path="/restaurant/:id" element={IndividualRestaurant} /> />  
        <Route path="/category/:category" element={CategoryProducts} /> />  
        <Route path="/profile" element={Profile} /> />  
        <Route path="/auth" element={Authentication} /> />  
        <Route path="/admin" element={Admin} /> />  
        <Route path="/all-restaurants" element={AllRestaurants} /> />  
        <Route path="/admin/edit-restaurant/:id" element={EditRestaurant} /> />  
        <Route path="/admin/restaurant-dashboard/:id" element={AdminRestaurantDashboard} /> />  
        <Route path="/all-users" element={AllUsers} /> />  
        <Route path="/all-orders" element={AllOrders} /> />  
        <Route path="/restaurant" element={RestaurantHome} /> />  
        <Route path="/restaurant-orders" element={RestaurantOrders} /> />  
        <Route path="/restaurant-menu" element={RestaurantMenu} /> />  
        <Route path="/new-product" element={NewProduct} /> />  
        <Route path="/update-product/:id" element={EditProduct} /> />  
      </Routes>  
    </div>  
  );  
}
```

CODE EXPLANATION

Server setup:

FOOD ORDERING APPLICATION

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

```
.env JS App.js JS index.js ×
food_delivery1 > src > JS index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6 import {BrowserRouter} from 'react-router-dom';
7 import GeneralContextProvider from './context/GeneralContext';
8
9 const root = ReactDOM.createRoot(document.getElementById('root'));
10 root.render(
11   <React.StrictMode>
12     <BrowserRouter>
13       <GeneralContextProvider>
14         <App />
15       </GeneralContextProvider>
16     </BrowserRouter>
17   </React.StrictMode>
18 );
19
20 // If you want to start measuring performance in your app, pass a function
21 // to log results (for example: reportWebVitals(console.log))
22 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
23 reportWebVitals();
```

User Authentication:

• Backend

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

```
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: restaurantAddr
      await restaurant.save();

      return res.status(201).json(user);
    } else{

      const newUser = new User({
        username, email, usertype, password: hashedPassword, approval: 'approved'
```

• Frontend

Login:

FOOD ORDERING APPLICATION

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

Register:

```
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, image: restaurantImage});
      await restaurant.save();

      return res.status(201).json(user);
    } else{
      const newUser = new User({
        username, email, usertype, password: hashedPassword, approval: 'approved'
      });
      await newUser.save();
    }
  } catch (error) {
    console.log(error);
    return res.status(500).json({ message: 'Server Error' });
  }
});
```

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:

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

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

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

FOOD ORDERING APPLICATION

Filtering products:

```
const NewProduct = () => {  
  const [productPrice, setProductPrice] = useState(0);  
  const [productDiscount, setProductDiscount] = useState(0);  
  
  const [AvailableCategories, setAvailableCategories] = useState([]);  
  
  const [restaurant, setRestaurant] = useState();  
  
  useEffect(()=>{  
    fetchCategories();  
    fetchRestaurant();  
  }, [])  
  const fetchCategories = async () =>{  
    await axios.get('http://localhost:6001/fetch-categories').then(  
      (response)=>{  
        setAvailableCategories(response.data);  
      }  
    )  
  }  
  
  const fetchRestaurant = async () =>{  
    await axios.get('http://localhost:6001/fetch-restaurant-details/${userId}').then(  
      (response)=>{  
        setRestaurant(response.data);  
      }  
    )  
  }  
}
```

Backend

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

```
// // Fetch individual product  
app.get('/fetch-product-details/:id', async(req, res)=>{  
  const id = req.params.id;  
  try{  
    const product = await FoodItem.findById(id);  
    res.json(product);  
  }catch(err){  
    res.status(500).json({message: "Error occurred"});  
  }  
})
```

Add product to cart:

Frontend

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

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

Backend

Add product to cart:

FOOD ORDERING APPLICATION

```
// add cart item

app.post('/add-to-cart', async(req, res)=>{

  const {userId, foodItemId, foodItemName, restaurantId, foodItemImg, price, discount, quantity} = req.body;
  try{

    const restaurant = await Restaurant.findById(restaurantId);

    const item = new Cart({userId, foodItemId, foodItemName, restaurantId, restaurantName: restaurant.name, price, discount, quantity});
    await item.save();

    res.json({message: 'Added to cart'});

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

Order products:

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

- Frontend

```
const placeOrder = async() =>{
  if(cart.length > 0){
    await axios.post('http://localhost:6001/place-cart-order', {userId, name, mobile, email, address, pincode, paymentMethod, orderDate});
    (response)=>{
      alert('Order placed!!');
      setName('');
      setMobile('');
      setEmail('');
      setAddress('');
      setPincode('');
      setPaymentMethod('');
      navigate('/profile');
    }
  }
}
```

- 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.

```
.env JS App.js JS index.js X Cart.jsx
food_delivery1 > server > JS index.js > ...
26   }).then(()=>{
451   // Order from cart
452
453   app.post('/place-cart-order', async(req, res)=>{
454     const {userId, name, mobile, email, address, pincode, paymentMethod, orderDate} = req.body;
455     try{
456
457       const cartItems = await Cart.find({userId});
458       cartItems.map(async (item)=>{
459
460         const newOrder = new Orders({userId, name, email, mobile, address, pincode, paymentMethod, orderDate});
461         await newOrder.save();
462         await Cart.deleteOne({_id: item._id})
463       })
464       res.json({message: 'Order placed'});
465
466     }catch(err){
467       res.status(500).json({message: "Error occurred"});
468     }
469   })
470 }
```

Add new product:

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

- Frontend:

FOOD ORDERING APPLICATION

```
.env JS App.js JS index.js NewProduct.jsx X
food_delivery1 > src > pages > restaurant > NewProduct.jsx > default
6  const NewProduct = () => {
45
46  const handleNewProduct = async() =>{
47    await axios.post('http://localhost:6001/add-new-product', {restaurantId: restaurant._id, productName, productDescription, productMainImg, productCategory, productMenuCategory, productNewCategory, productPrice, productDiscount});
48    (response)=>{
49      alert("product added");
50      setProductName('');
51      setProductDescription('');
52      setProductMainImg('');
53      setProductCategory('');
54      setProductMenuCategory('');
55      setProductNewCategory('');
56      setProductPrice(0);
57      setProductDiscount(0);
58
59      navigate('/restaurant-menu');
```

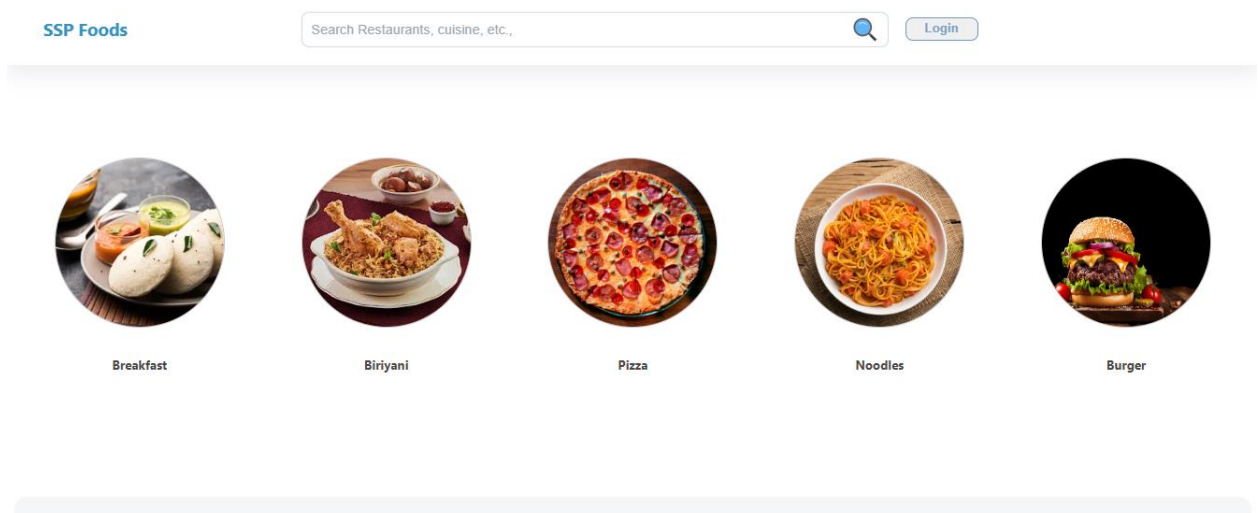
Backend:

```
.env JS App.js JS index.js X NewProduct.jsx
food_delivery1 > server > JS index.js > ...
26  }).then(()=>{
301  app.post('/add-new-product', async(req, res)=>{
302    const {restaurantId, productName, productDescription, productMainImg, productCategory, productMenuCategory, productNewCategory, productPrice, productDiscount} = req.body;
303    try{
304      if(productMenuCategory === 'new category'){
305        const admin = await Admin.findOne();
306        admin.categories.push(productNewCategory);
307        await admin.save();
308        const newProduct = new FoodItem({restaurantId, title: productName, description: productDescription, mainImage: productMainImg, category: productCategory, menuCategory: productMenuCategory, price: productPrice, discount: productDiscount});
309        await newProduct.save();
310        const restaurant = await Restaurant.findById(restaurantId);
311        restaurant.menu.push(productNewCategory);
312        await restaurant.save();
313      } else{
314        const newProduct = new FoodItem({restaurantId, title: productName, description: productDescription, mainImage: productMainImg, category: productCategory, menuCategory: productMenuCategory, price: productPrice, discount: productDiscount});
315        await newProduct.save();
316      }
317      res.json({message: "product added!!"});
318    }catch(err){
319      res.status(500).json({message: "Error occurred"});
320    }
321  })
```

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

Demo UI images:


Landing page



FOOD ORDERING APPLICATION

Restaurants

Popular Restaurants



Durga
Hyderabad

Restaurant Menu

Filters

Sort By

☐ Popularity

☐ low-price

☐ high-price

☐ Discount

☐ Rating

Food Type

☐ Veg

☐ Non Veg

☐ Beverages

Categories


☐ Rice

☐ Items


Durga

Hyderabad


All Items




Biryani
₹ 550-550
Add item



Meals
₹ 200-200
Add item



Puri
₹ 50-50
Add item



Dosa
₹ 90-100
Add item

Authentication

Register

username

Username

name@example.com

Email address

Password

Password

User type

▼

Sign up

Already registered?

Login

FOOD ORDERING APPLICATION

- User Profile

SSP Foods

Search Restaurants, cuisine, etc.,

Bhavani

0

Username: Bhavani

Email: madhu@gmail.com

Orders: 1

Logout

Orders

Biryani

Durga

Quantity: 0

Total Price: ₹ 0 ₹0

Payment mode: cod

Ordered on: 2025-06-27 Time: 09:44

status: order placed

Cancel

- Cart

SSP Foods

Search Restaurants, cuisine, etc.,

Bhavani

Meals

Durga

Quantity:

0

Price: ₹ 200 ₹200

Remove

Dosa

Durga

Quantity:

0

Price: ₹ 90 ₹100

Remove

Total MRP:

₹ 0

Discount on MRP:

- ₹ 0

Delivery Charges:

+ ₹ 50

Final Price: ₹ 50

Place order

- Admin dashboard

SSP Foods (admin)

Home

Users

Orders

Restaurants

Logout

Total users

2

View all

All Restaurants

1

View all

All Orders

1

View all

Popular Restaurants(promotions)

☒

Durga

Update

Approvals

No new requests...


- All Orders

FOOD ORDERING APPLICATION

SSP Foods (admin)

HomeUsersOrdersRestaurantsLogout

Orders



Biryani

Durga

Userid: 685e679eb30a4de1937d3f70Name: MadhuMobile: 6303879527Email:

Quantity: 0Total Price: ₹ 0 ₹-0Payment mode: cod

Address:Pincode:Ordered on: 2025-06-27 Time: 09:44

status: order placed

Update order status

Update

Cancel


- All restaurants

SSP Foods

Search Restaurants, cuisine, etc.,

Login

All restaurants



Durga
Hyderabad

- Restaurant Dashboard

SSP Foods (Restaurant)

HomeOrdersMenuNew ItemLogout

All Items

9

View all

All Orders

1

View all

Add Item

(new)

Add now

- New Item

FOOD ORDERING APPLICATION

The screenshot shows a web application interface for 'SSP Foods (Restaurant)'. The top navigation bar includes links for 'Home', 'Orders', 'Menu', 'New Item', and 'Logout'. The main content area features a 'New Product' form. The form has a dark theme with white text and input fields. It includes a 'Product name' field, a 'Product Description' field, and a 'Thumbnail Img url' field. Below these is a 'Gender' section with three radio buttons: 'Veg', 'Non Veg', and 'Beverages'. There is also a 'Choose Product category' dropdown menu. At the bottom of the form are two input fields for 'Price' and 'Discount (in %)', both with the value '0'. An 'Add product' button is located at the bottom right of the form.

SSP Foods (Restaurant)

Home Orders Menu New Item Logout

New Product

Product name Product Description Thumbnail Img url

Gender

☐ Veg ☐ Non Veg ☐ Beverages

Choose Product category Category 0 Price 0 Discount (in %)

Add product

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

https://github.com/harsha-varadhan-reddy-07/SB_Foods--e-commerce-MERN The demo of the app is available at:

<https://drive.google.com/file/d/1RJzLn timer 63AIDz6dUwKgoZcZq9fA9gZwX/view?usp=sharing>