Food Delivery App

Using MERNSTACK

Presented By
SARAN.K
22EC145

Introduction

In today's digital age, food ordering and enjoyment have been transformed by technology, making it easier for consumers to satisfy their cravings and explore diverse cuisines. Food delivery apps have become integral to daily life, offering unparalleled convenience, choice, and efficiency. Whether during a busy workday or a cozy night in, these apps enable seamless food ordering from numerous restaurants via smartphones. We will explore creating a cutting-edge food delivery app using the MERN stack. Comprising MongoDB, Express.js, React.js, and Node.js, the MERN stack is a powerful framework for building modern web applications. It empowers developers to create dynamic, responsive platforms tailored to the evolving needs of users and businesses.

Existing System

User Registration and Authentication:

1.Enables users to create accounts and log in securely using email or phone numbers.

Menu Display:

1.Showcases the restaurant's full menu with detailed descriptions, prices, and images of the dishes.

Order Customization:

1.Allows users to customize their orders (e.g., adding extra ingredients, special instructions).

Table Reservation:

1.Provides options for users to reserve tables at the restaurant for dine-in.

Real-Time Order Tracking:

1.Enables users to track the status of their delivery orders in real-time from preparation to delivery.

Multiple Payment Options:

1. Supports various payment methods, including credit/debit cards, digital wallets, and cash on delivery.

User Reviews and Ratings:

1. Allows customers to leave reviews and ratings for their ordered dishes and overall service.

Customer Support:

1. Provides in-app customer support through chat, call, or email to address customer queries and issues.

Proposed System

User-Friendly Interface (React.js):

I.Develop an intuitive and responsive user interface using React.js to ensure seamless navigation and a smooth user experience.

2.Include features such as easy-to-browse menus, order customization, and a streamlined checkout process.

Efficient Backend (Node.js and Express.js):

1.Use Node.js and Express.js to build a robust and scalable backend server.

2.Implement RESTful APIs to handle user authentication, order processing, menu management, and real-time order tracking.

Database Management (MongoDB):

1. Utilize MongoDB for flexible and efficient data storage.

2. Create collections for users, orders, menu items, and promotions to manage and retrieve data quickly.

User Authentication and Authorization:

I.Implement secure user authentication using JWT (JSON Web Tokens) for session management.
Ensure secure user data handling and password encryption.

Admin Dashboard:

1.Develop an admin dashboard for restaurant management to monitor orders, update menu items, manage promotions, and view sales reports.

2.Include analytics tools to track performance metrics and customer feedback.

Testing and Quality Assurance:

1.Conduct thorough testing, including unit testing, integration testing, and user acceptance testing, to ensure the app is bug-free and performs well under various conditions.

Software Requirement

1.**Visual Studio Code** is a lightweight yet powerful code editor ideal for MERN stack development, offering features like syntax highlighting and debugging tools.

2. **Git** is a distributed version control system used for tracking code changes, collaborating with developers, and managing project history.

3. **Node.js** enables server-side JavaScript execution, forming a core component of the MERN stack for backend development.

4. **Express.js** simplifies building robust and scalable backend APIs within Node.js applications.

5. **MongoDB,** a NoSQL database, is favored for MERN stack projects due to its scalability, flexibility, and seamless integration with Node.js.

6. React.js is a JavaScript library for building dynamic user interfaces, essential for frontend development in MERN stack applications.

7. React Router facilitates routing and navigation management in React.js applications, enabling the creation of multi-page experiences within single-page applications (SPAs).

Modules Used

1.Jsonwebtoken

2.Cors

3.Stripe

4.Mongoose

5.Express

6.Bycrypt

7.Env

8.Bodyparser

9.Multer

10.Validater

11.Nodemon

jsonwebtoken: A library for generating and verifying JSON Web Tokens (JWT) used for authentication.

cors: A middleware for enabling Cross-Origin Resource Sharing (CORS) in Express.js applications.

stripe: A payment processing library that allows integration of payment gateways in web applications.

mongoose: An ODM (Object Data Modeling) library for MongoDB and Node.js, providing schema-based data modeling.

express: A minimalist web framework for Node.js, used for building robust APIs and web applications.

bcrypt: A library for hashing passwords, used to enhance security by encrypting sensitive data.

dotenv: A module for loading environment variables from a .env file into process.env in Node.js projects.

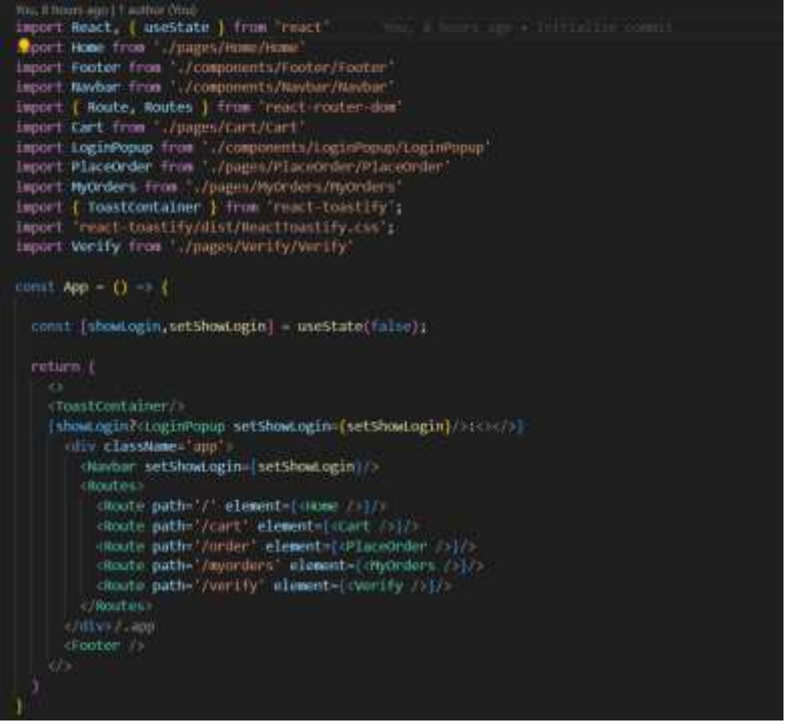
body-parser: A middleware for parsing incoming request bodies in a middleware before handling them.

multer: A middleware for handling multipart/form-data, commonly used for file uploads in Node.js.

validator: A library of string validators and sanitizers used to validate and sanitize user inputs.

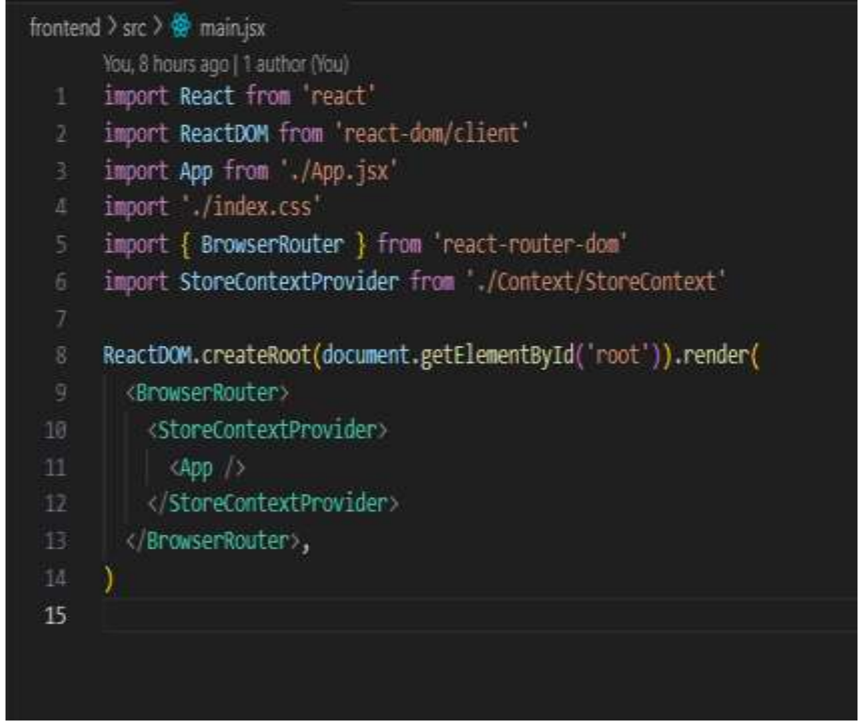
nodemon: A utility that automatically restarts the Node.js application when file changes in the directory are detected.

Frontend Code



You, 8 hours ago | 1 author (You)

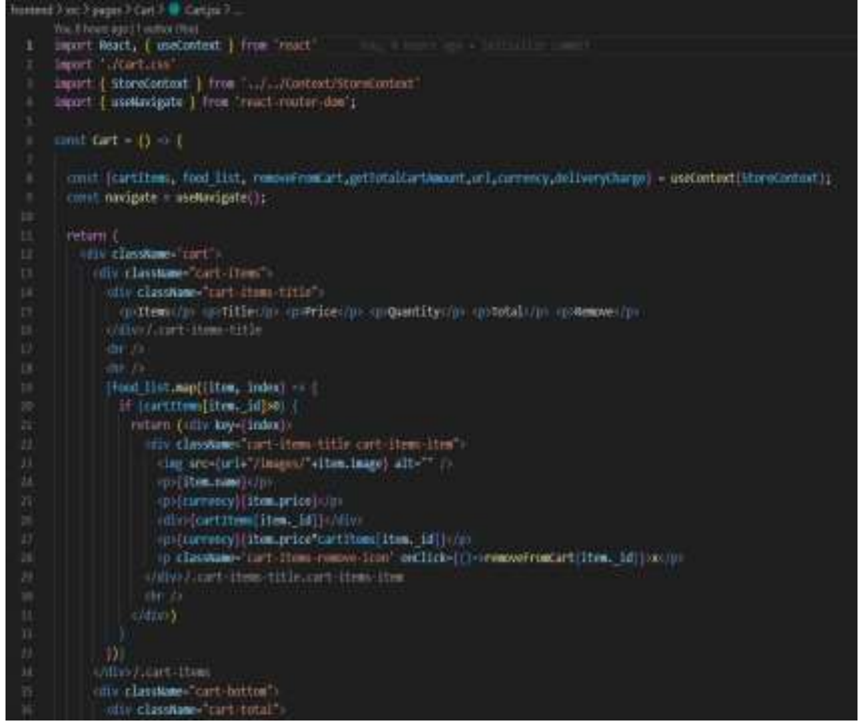
```jsx
import React, { useState } from 'react'      You, 8 hours ago • Initialize commit
import Home from './pages/Home/Home'
import Footer from './components/Footer/Footer'
import Navbar from './components/Navbar/Navbar'
import { Route, Routes } from 'react-router-dom'
import Cart from './pages/Cart/Cart'
import LoginPopup from './components/LoginPopup/LoginPopup'
import PlaceOrder from './pages/PlaceOrder/PlaceOrder'
import MyOrders from './pages/MyOrders/MyOrders'
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import Verify from './pages/Verify/Verify'

const App = () => {

  const [showLogin,setShowLogin] = useState(false);

  return (
    <>
    <ToastContainer/>
    {showLogin?<LoginPopup setShowLogin={setShowLogin}/>:<></>}
      <div className='app'>
        <Navbar setShowLogin={setShowLogin}/>
        <Routes>
          <Route path='/' element={<Home />}/>
          <Route path='/cart' element={<Cart />}/>
          <Route path='/order' element={<PlaceOrder />}/>
          <Route path='/myorders' element={<MyOrders />}/>
          <Route path='/verify' element={<Verify />}/>
        </Routes>
      </div>/.app
      <Footer />
    </>
  )
}
```



frontend > src > main.jsx

You, 8 hours ago | 1 author (You)

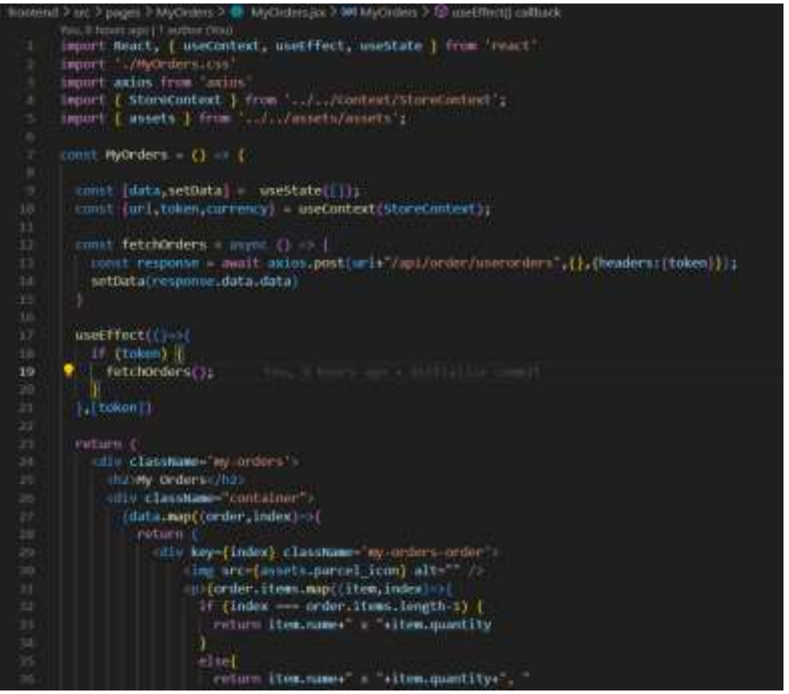
```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'
import { BrowserRouter } from 'react-router-dom'
import StoreContextProvider from './Context/StoreContext'

ReactDOM.createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <StoreContextProvider>
      <App />
    </StoreContextProvider>
  </BrowserRouter>,
)

```



frontend > src > pages > Cart > Cart.jsx > ...

You, 8 hours ago | 1 author (You)

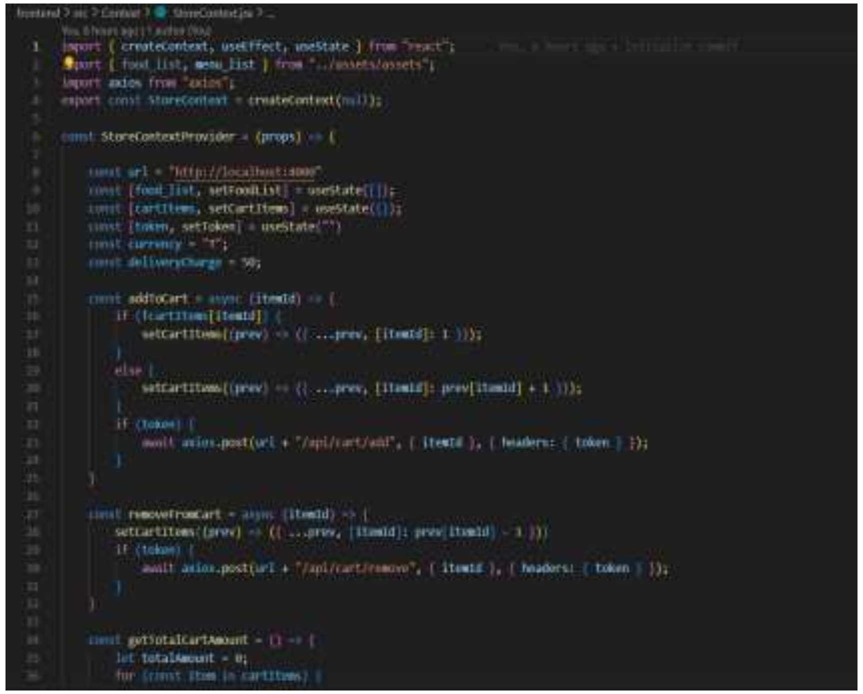
```jsx
import React, { useContext } from 'react'      You, 8 hours ago • Initialize commit
import './Cart.css'
import { StoreContext } from '../../Context/StoreContext'
import { useNavigate } from 'react-router-dom';

const Cart = () => {

  const {cartItems, food_list, removeFromCart,getTotalCartAmount,url,currency,deliveryCharge} = useContext(StoreContext);
  const navigate = useNavigate();

  return (
    <div className='cart'>
      <div className="cart-items">
        <div className="cart-items-title">
          <p>Items</p> <p>Title</p> <p>Price</p> <p>Quantity</p> <p>Total</p> <p>Remove</p>
        </div>/.cart-items-title
        <br />
        <hr />
        {food_list.map((item, index) => {
          if (cartItems[item._id]>0) {
            return (<div key={index}>
              <div className="cart-items-title cart-items-item">
                <img src={url+"/images/"+item.image} alt="" />
                <p>{item.name}</p>
                <p>{currency}{item.price}</p>
                <div>{cartItems[item._id]}</div>
                <p>{currency}{item.price*cartItems[item._id]}</p>
                <p className='cart-items-remove-icon' onClick={()=>removeFromCart(item._id)}>x</p>
              </div>/.cart-items-title.cart-items-item
              <hr />
            </div>)
          }
        })}
      </div>/.cart-items
      <div className="cart-bottom">
        <div className="cart-total">
```



frontend > src > pages > MyOrders > MyOrders.jsx > MyOrders > useEffect() callback

You, 8 hours ago | 1 author (You)

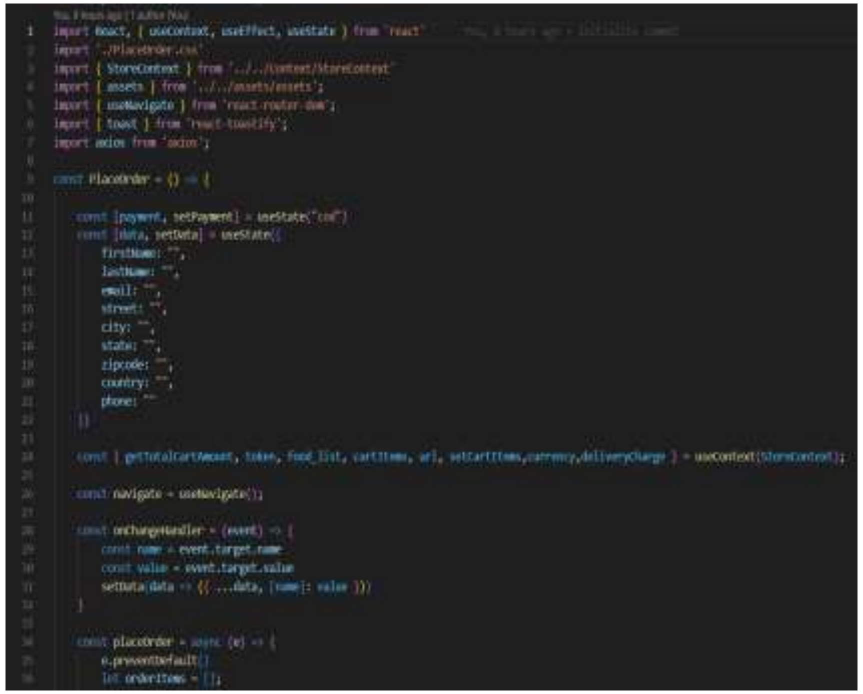
```jsx
import React, { useContext, useEffect, useState } from 'react'
import './MyOrders.css'
import axios from 'axios'
import { StoreContext } from '../../Context/StoreContext';
import { assets } from '../../assets/assets';

const MyOrders = () => {

  const [data,setData] =  useState([]);
  const {url,token,currency} = useContext(StoreContext);

  const fetchOrders = async () => {
    const response = await axios.post(url+"/api/order/userorders",{},{headers:{token}});
    setData(response.data.data)
  }

  useEffect(()=>{
    if (token) {
      fetchOrders();      You, 8 hours ago • Initialize commit
    }
  },[token])

  return (
    <div className='my-orders'>
      <h2>My Orders</h2>
      <div className="container">
        {data.map((order,index)=>{
          return (
            <div key={index} className='my-orders-order'>
                <img src={assets.parcel_icon} alt="" />
                <p>{order.items.map((item,index)=>{
                  if (index === order.items.length-1) {
                    return item.name+" x "+item.quantity
                  }
                  else{
                    return item.name+" x "+item.quantity+", "
```



frontend > src > Context > StoreContext.jsx > ...

You, 8 hours ago | 1 author (You)

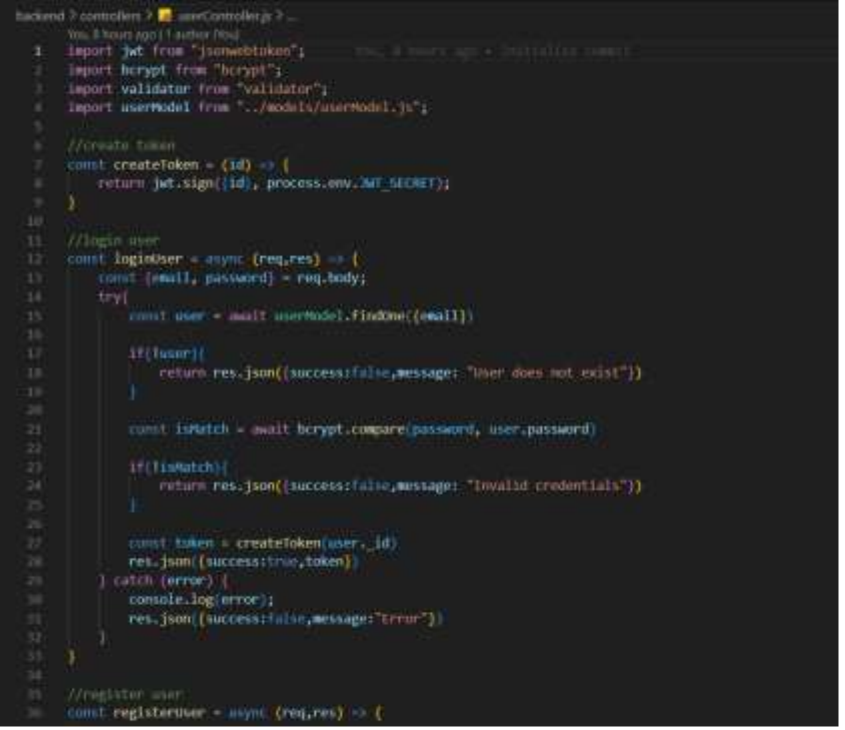
```jsx
import { createContext, useEffect, useState } from "react";      You, 8 hours ago • Initialize commit
import { food_list, menu_list } from "../assets/assets";
import axios from "axios";
export const StoreContext = createContext(null);

const StoreContextProvider = (props) => {

    const url = "http://localhost:4000"
    const [food_list, setFoodList] = useState([]);
    const [cartItems, setCartItems] = useState({});
    const [token, setToken] = useState("")
    const currency = "₹";
    const deliveryCharge = 50;

    const addToCart = async (itemId) => {
        if (!cartItems[itemId]) {
            setCartItems((prev) => ({ ...prev, [itemId]: 1 }));
        }
        else {
            setCartItems((prev) => ({ ...prev, [itemId]: prev[itemId] + 1 }));
        }
        if (token) {
            await axios.post(url + "/api/cart/add", { itemId }, { headers: { token } });
        }
    }

    const removeFromCart = async (itemId) => {
        setCartItems((prev) => ({ ...prev, [itemId]: prev[itemId] - 1 }))
        if (token) {
            await axios.post(url + "/api/cart/remove", { itemId }, { headers: { token } });
        }
    }

    const getTotalCartAmount = () => {
        let totalAmount = 0;
        for (const item in cartItems) {
```



You, 8 hours ago | 1 author (You)

```jsx
import React, { useContext, useEffect, useState } from 'react'      You, 8 hours ago • Initialize commit
import './PlaceOrder.css'
import { StoreContext } from '../../Context/StoreContext'
import { assets } from '../../assets/assets';
import { useNavigate } from 'react-router-dom';
import { toast } from 'react-toastify';
import axios from 'axios';

const PlaceOrder = () => {

    const [payment, setPayment] = useState("cod")
    const [data, setData] = useState({
        firstName: "",
        lastName: "",
        email: "",
        street: "",
        city: "",
        state: "",
        zipcode: "",
        country: "",
        phone: ""
    })

    const { getTotalCartAmount, token, food_list, cartItems, url, setCartItems,currency,deliveryCharge } = useContext(StoreContext);

    const navigate = useNavigate();

    const onChangeHandler = (event) => {
        const name = event.target.name
        const value = event.target.value
        setData(data => ({ ...data, [name]: value }))
    }

    const placeOrder = async (e) => {
        e.preventDefault()
        let orderItems = [];
```

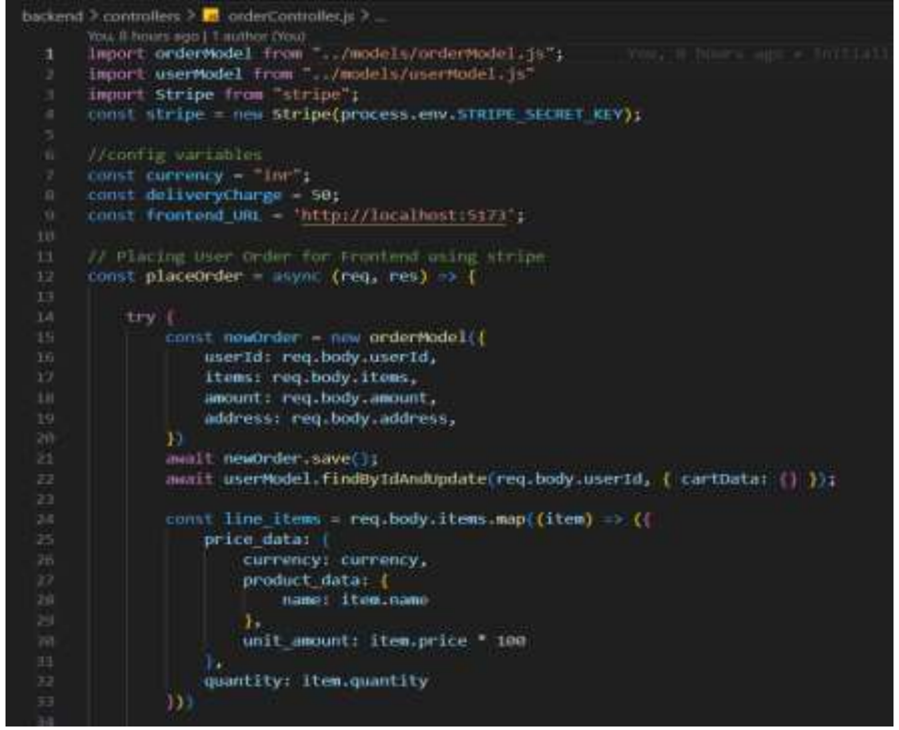
Backend Code



backend > controllers > userController.js > ...

You, 8 hours ago | 1 author (You)

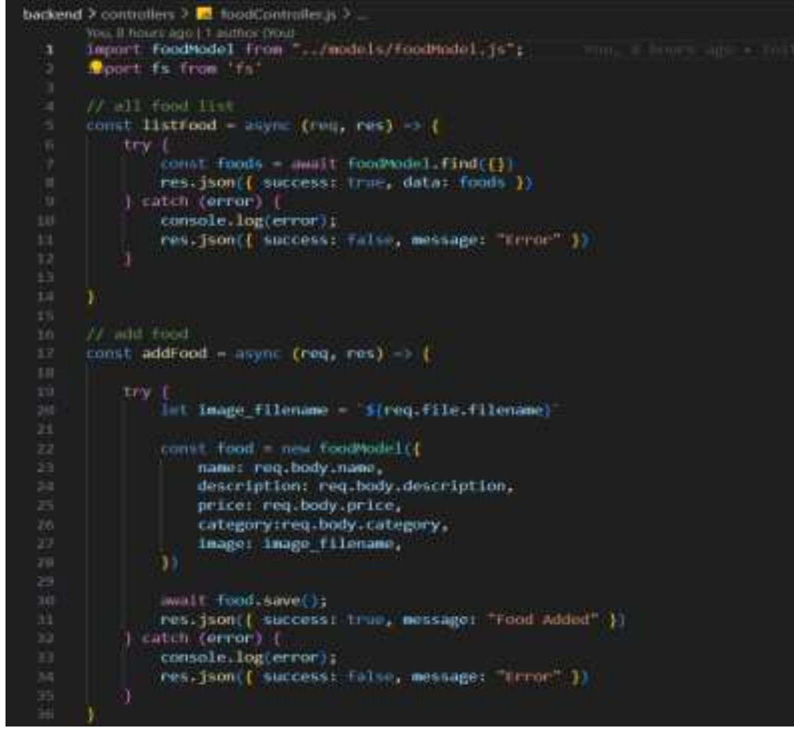
```javascript
import jwt from "jsonwebtoken";          You, 8 hours ago • Initialize commit
import bcrypt from "bcrypt";
import validator from "validator";
import userModel from "../models/userModel.js";

//create token
const createToken = (id) => {
    return jwt.sign({id}, process.env.JWT_SECRET);
}

//login user
const loginUser = async (req,res) => {
    const {email, password} = req.body;
    try{
        const user = await userModel.findOne({email})

        if(!user){
            return res.json({success:false,message: "User does not exist"})
        }

        const isMatch = await bcrypt.compare(password, user.password)

        if(!isMatch){
            return res.json({success:false,message: "Invalid credentials"})
        }

        const token = createToken(user._id)
        res.json({success:true,token})
    } catch (error) {
        console.log(error);
        res.json({success:false,message:"Error"})
    }
}

//register user
const registerUser = async (req,res) => {
```



backend > controllers > orderController.js > ...

You, 8 hours ago | 1 author (You)

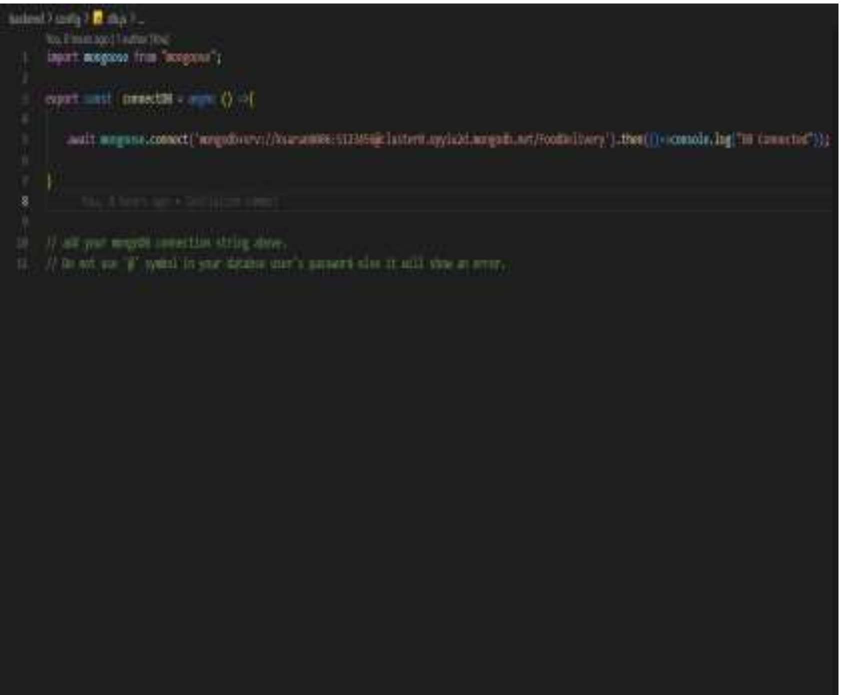
```javascript
import orderModel from "../models/orderModel.js";          You, 8 hours ago • Initiali
import userModel from "../models/userModel.js"
import Stripe from "stripe";
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

//config variables
const currency = "inr";
const deliveryCharge = 50;
const frontend_URL = 'http://localhost:5173';

// Placing User Order for Frontend using stripe
const placeOrder = async (req, res) => {

    try {
        const newOrder = new orderModel({
            userId: req.body.userId,
            items: req.body.items,
            amount: req.body.amount,
            address: req.body.address,
        })
        await newOrder.save();
        await userModel.findByIdAndUpdate(req.body.userId, { cartData: {} });

        const line_items = req.body.items.map((item) => ({
            price_data: {
                currency: currency,
                product_data: {
                    name: item.name
                },
                unit_amount: item.price * 100
            },
            quantity: item.quantity
        }))
```



backend > controllers > foodController.js > ...

You, 8 hours ago | 1 author (You)

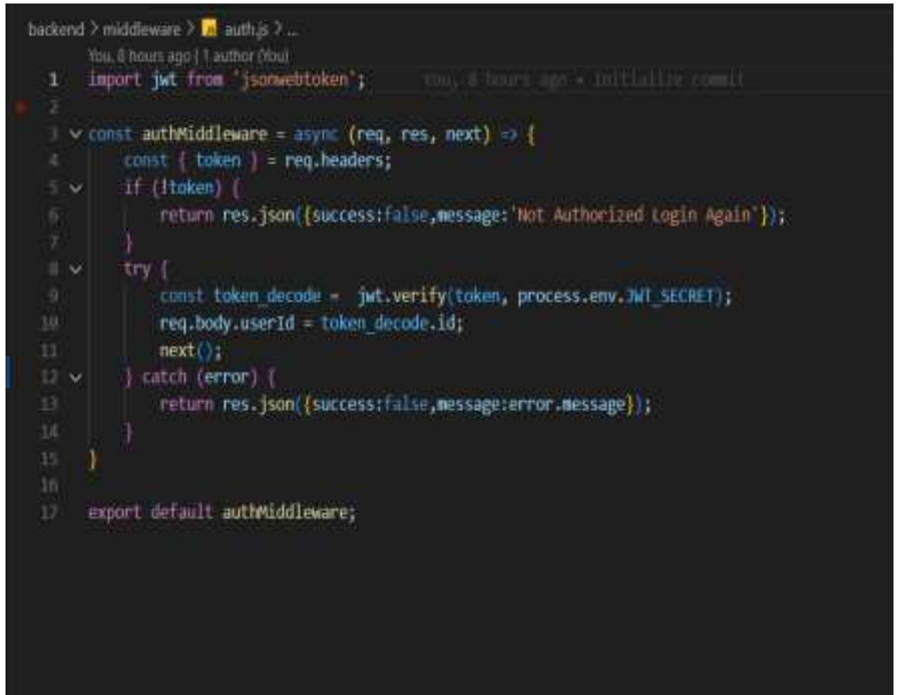
```javascript
import foodModel from "../models/foodModel.js";          You, 8 hours ago • Ini
import fs from 'fs'

// all food list
const listFood = async (req, res) => {
    try {
        const foods = await foodModel.find({})
        res.json({ success: true, data: foods })
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: "Error" })
    }

}

// add food
const addFood = async (req, res) => {

    try {
        let image_filename = `${req.file.filename}`

        const food = new foodModel({
            name: req.body.name,
            description: req.body.description,
            price: req.body.price,
            category:req.body.category,
            image: image_filename,
        })

        await food.save();
        res.json({ success: true, message: "Food Added" })
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: "Error" })
    }
}
```



backend > config > db.js > ...

You, 8 hours ago | 1 author (You)

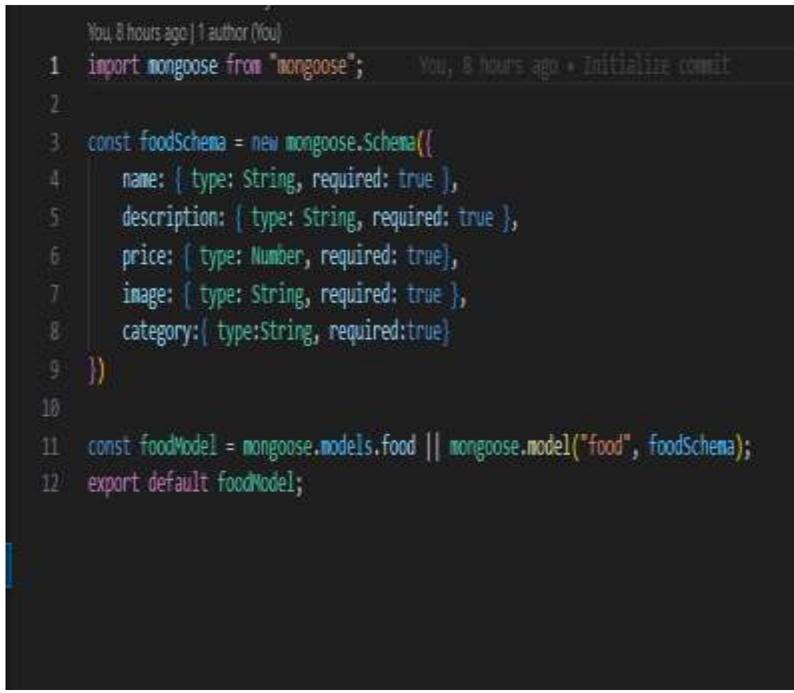
```javascript
import mongoose from "mongoose";

export const  connectDB = async () =>{

    await mongoose.connect('mongodb+srv://[illegible]@cluster0.[illegible].mongodb.net/food-delivery').then(()=>console.log("DB Connected"));

}
    You, 8 hours ago • Initialize commit

// add your mongoDB connection string above.
// Do not use '@' symbol in your databse user's password else it will show an error.
```



backend > middleware > auth.js > ...

You, 8 hours ago | 1 author (You)

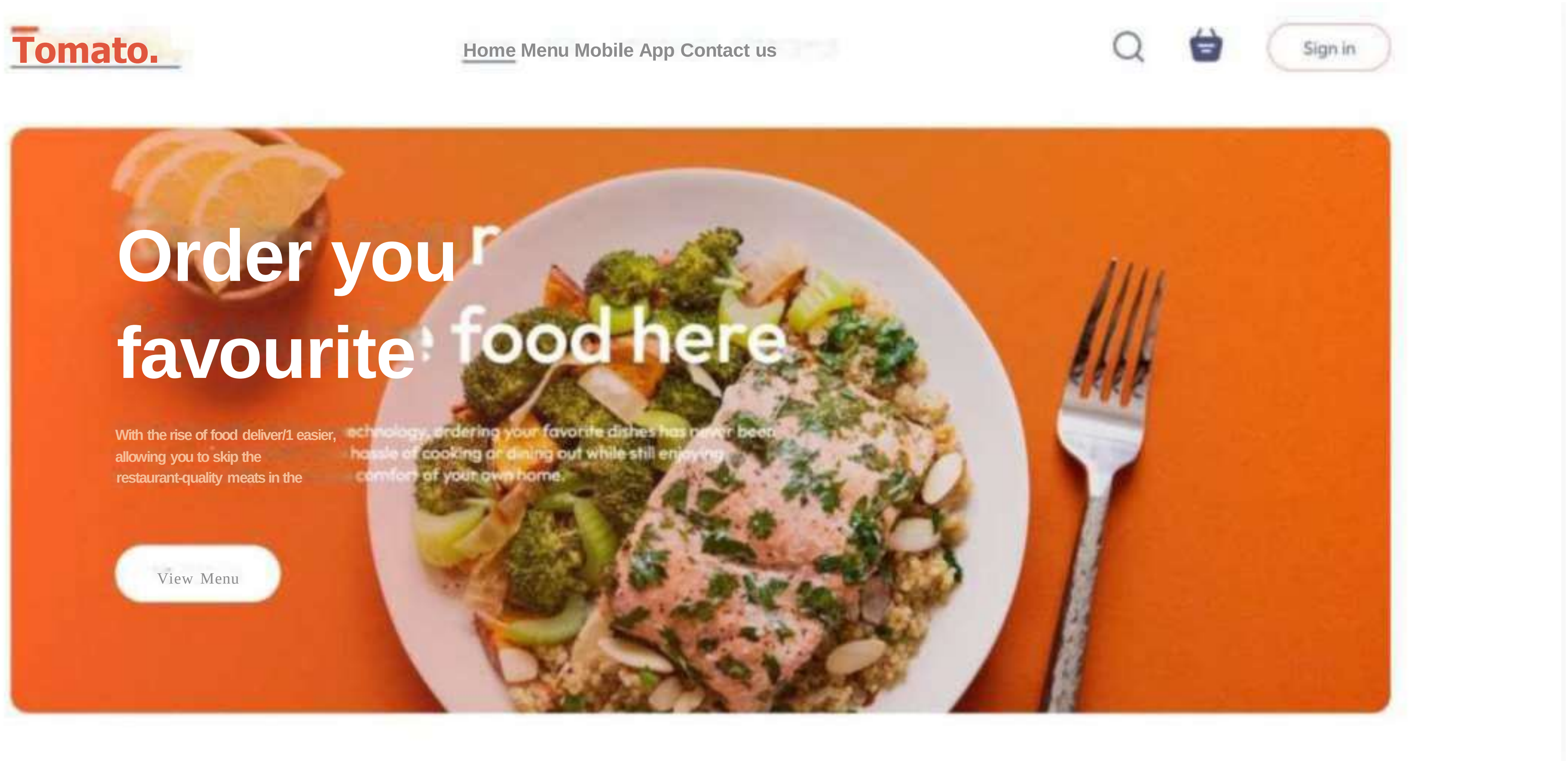
```javascript
import jwt from 'jsonwebtoken';          You, 8 hours ago • Initialize commit

const authMiddleware = async (req, res, next) => {
    const { token } = req.headers;
    if (!token) {
        return res.json({success:false,message:'Not Authorized Login Again'});
    }
    try {
        const token_decode =  jwt.verify(token, process.env.JWT_SECRET);
        req.body.userId = token_decode.id;
        next();
    } catch (error) {
        return res.json({success:false,message:error.message});
    }
}

export default authMiddleware;
```



You, 8 hours ago | 1 author (You)

```javascript
import mongoose from "mongoose";          You, 8 hours ago • Initialize commit

const foodSchema = new mongoose.Schema({
    name: { type: String, required: true },
    description: { type: String, required: true },
    price: { type: Number, required: true},
    image: { type: String, required: true },
    category:{ type:String, required:true}
})

const foodModel = mongoose.models.food || mongoose.model("food", foodSchema);
export default foodModel;
```

Output

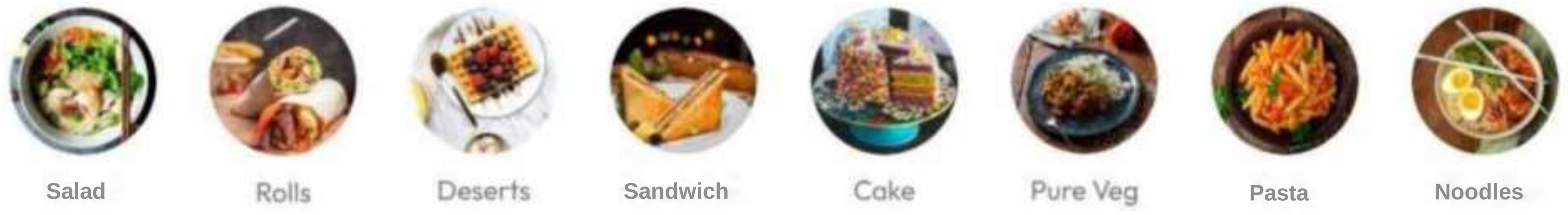


Order your favourite food here

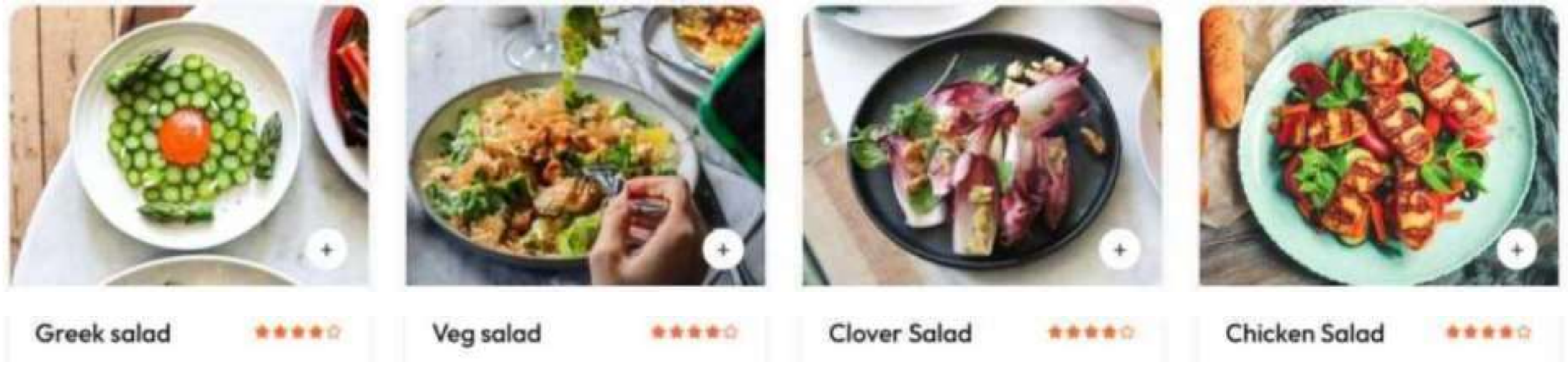
With the rise of food deliver/1 easier, echnology, ordering your favorite dishes has never been
allowing you to skip the hassle of cooking or dining out while still enjoying
restaurant-quality meats in the comfort of your own home.

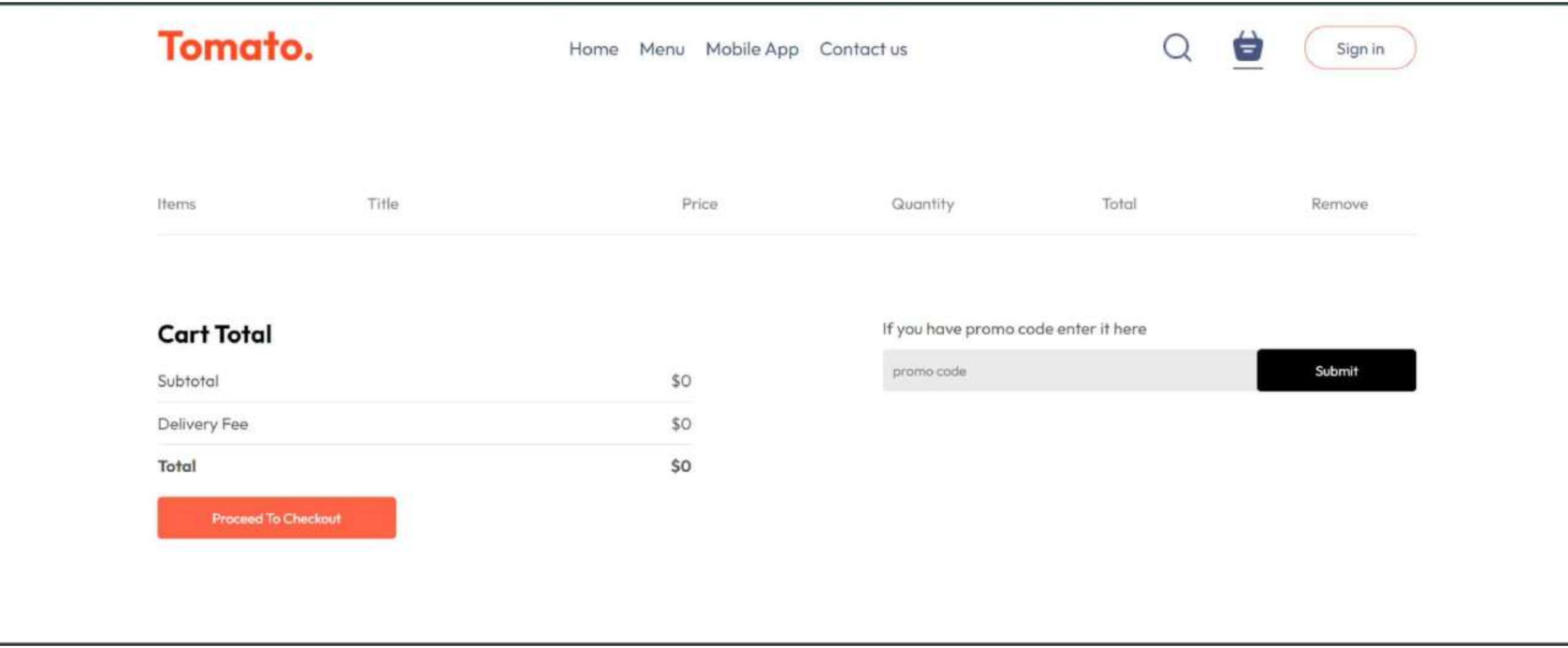
Explore the menu

Explore our diverse menu featuring mouthwatering appetizers, flavorful entrees, refreshing salads, decadent desserts, and refreshing beverages to satisfy every craving.



Top dishes near you





Items	Title	Price	Quantity	Total	Remove

Cart Total

Subtotal	$0
Delivery Fee	$0
Total	**$0**

If you have promo code enter it here

Conclusion

In conclusion, building a food delivery app using the MERN stack offers a powerful and flexible solution tailored to modern needs. By leveraging tools like Visual Studio Code for development, Git for version control, and various specialized libraries such as jsonwebtoken for authentication and stripe for payment processing, developers can create robust and dynamic applications. The combination of MongoDB, Express.js, React.js, and Node.js ensures a seamless integration between the backend and frontend, providing users with an efficient and enjoyable experience. With real-time features, secure transactions, and an intuitive interface, the proposed solution not only meets but exceeds the expectations of today's tech-savvy consumers.

Thank You