MERN STACK E-Commerce Web Application

# Introduction :

ShopeZ is true that technology has become an essential tool for online marketing nowadays. However, there are numerous small shops and grocery stores with a mostly oﬄine business model in Vietnam recently. With this commerce model, it will bring a lot of bad experiences for both buyers and sellers. For instance, the seller has the product they want to oﬀer but the buyer may not know it, or the buyer may urgently need to purchase something, but the store is out of stock. Moreover, online shopping helps customers to choose a wide range of products, and prices and they can compare them to each other easily.

Customers can quickly search for the products if they are available and come to the store to pick them up and they can contact the shop owner to learn more about the products that they are looking for. In order to make a website that can acquire the needs of both customers and retailers, MERN (MongoDB, Express.js framework, ReactJS library, NodeJS platform) is one of the powerful stacks that can help us to develop an e-commerce web application.

**TEAM MEMBERS:**

|  |  |
| --- | --- |
| **GAYATHRI T** | **513421104009** |
| **KEERTHANA PRIYA D R** | **513421104014** |
| **PAVITHRA N** | **513421104027** |
| **POONGAVANAM P** | **513421104028** |

# Project Overview:

### Deﬁnition :

E-commerce, EC for short (E-commerce) is a concept referring to transactions, purchases, and sale of goods and services via the Internet.

E-commerce was ﬁrst known in the 1960s. After years of development, as mobile devices became popular, social media increasingly aﬃrmed the power and the boom of the webpage. Launchers promote the rapid development of commerce (E-commerce).

**Purpose :**

To create a robust e-commerce platform where users can browse products, make secure purchases, and track orders. Vendors can list products, manage inventory, and review sales analytics. The platform emphasises a seamless shopping experience with features like personalised recommendations, multi-payment options, and real-time order tracking

# Features:

***Global market:*** Clearly, when you open a physical store, you will only be able to deliver your goods and services in a small geographic area.

E-commerce will help you solve that problem. E-commerce helps you reach the market quickly, expanding the market to the maximum level compared to direct sales so that products and services are easily introduced, purchased, and sold through retailers. and online market.

***Always open:*** In e-commerce, running an online business is much easier, it's always open 24h / 7/365. For businesses, it's a great opportunity to increase sales opportunities all the time.

***Budget savings:*** Compared with traditional forms of commercial business, all costs when e-commerce business are reduced: the cost of renting booths, salespeople, and management is much more economical. Naturally,

when sellers save operating costs, they can oﬀer more incentives and better discounts for their customers. At this time, the customer is the next beneﬁciary. Mutual beneﬁt, isn't it great?

***Inventory management:*** By using electronic tools to speed up the ordering, delivery, and payment processes, e-commerce businesses can save billions of operating costs and reduce the amount of inventory.

***Most accurate customer marketing:*** With access to customer data and the opportunity to track customers' buying habits, e-commerce businesses can quickly identify and market products and services. service. Service most suitable for consumers.

***Work anywhere, buy anywhere:*** Running an E-Commerce business allows you to not need to sit in the oﬃce, and buying does not force you to go to the supermarket. Everything the seller and the buyer needs is an

internet-connected device and that's all.

# Architecture of MERN Stack :

* Frontend: HTML, CSS, JavaScript, React
* Backend: Node.js, Express, MongoDB, JWT, Twilio
* Database: The database used in this freelancers' website is MongoDB. It’s a NoSQL database suitable for handling large amounts of unstructured or semi-structured data, making it a popular choice for web applications that need ﬂexibility and scalability, such as freelancing platforms.
  1. JavaScript :

JavaScript is a scripting, object-oriented, cross-platform programming language. Objects of the host and the environment can be connected to JavaScript and arranged in ways to operate them.

* + - **Client-side JavaScript:** JavaScript is developed by implementing objects for controlling the browser and DOM. For instance, an application is granted by client-side extensions to inﬂuence components on an HTML page and answer to user behavior like mouse hovers, form input, and page

changeover.

* **Server-side JavaScript:** JavaScript is developed by implementing the supplementary objects required to run JavaScript on the server. For instance, an application is granted by this server-side extension to connect to a database, transfer data frequently from one request to another section of the application or execute an application with another function ﬁle on the server.

### NodeJS :

Node.js is an open source, a system application, and furthermore is an environment for servers. Nodejs is an independent development platform built on Chrome's JavaScript Runtime that can build network applications quickly and easily. Google V8 JavaScript engine is used by Node.js to execute code.

Moreover, a huge proportion of essential modules are written in JavaScript 6 Node.js accommodates a built-in library that allows applications to

serve as a Web server left out of demanding software like Apache HTTP Server, Nginx, or IIS.

An event-driven, non-blocking I / O mechanism (Input / Output) is implemented by Node.js. It optimizes application throughout and is exceptionally highly extensible. Node.js use asynchronous in its functions.

Therefore, Node.js processes and executes all tasks in the background (background processing).

Products that have a lot of traﬃc are applying Node.js. Nonetheless, Node.js handles an application that needs to spread expeditiously, develop innovation, or build Startup projects as rapidly as possible.

Applications using NodeJS:

* WebSocket server
* Notiﬁcation system
* Applications that need to upload ﬁles on the client.
* Other real-time data applications.
  1. **Express.js :**

Express.js is a framework built on top of Nodejs. It provides powerful features for web or mobile development. Express.js supports HTTP and middleware methods, making the API extremely powerful and easy to use.

Express implements extra features to developers which help them get a better programming environment, not scaling down the speed of NodeJS.

Importantly, the well-known frameworks of NodeJS apply Express.js as a substance function, for instance: Sails.js, and MEAN.

## MongoDB :

MongoDB is an open-source database; it is also the leading NoSQL (\*) database currently used by millions of people. It is written in one of the most popular programming languages today. In addition, MongoDB is

cross-platform data that operates on the concepts of Collections and Documents, providing high performance with high availability and ease of expansion.

(\*) NoSQL is a source database format that does not use Transact-SQL to access information, this database was developed on JavaScript Framework on JSON data type. With its introduction, it has overcome the disadvantages of the RDBMS relational data model to improve operating speed, functionality, model scalability, and cache.

## ReactJS :

* + 1. Virtual-DOM :

Virtual-DOM is a JavaScript object, each object contains all the information needed to create a DOM, when the data changes it will calculate the change between the object and the real tree, which will help optimize the re-render DOM tree. It can be assumed that a virtual model can handle client data.

* + 1. Component :

React is built around components, not templates like other frameworks. A component can be created by the create Class function of the React object, the starting point when accessing this library. ReactJS creates HTML tags unlike we normally write but uses Component to wrap HTML tags into stratiﬁed objects to render.

Among React Components, the render function is the most important. It is a function that handles the generation of HTML tags as well as a demonstration of the ability to process via Virtual-DOM. Any changes in data at any time will be processed and updated immediately by Virtual-DOM.

* + 1. Pros and Cons of ReactJS :

**Pros of ReactJS**:

* Update data changes quickly.
* React is not a framework so it oﬄoads the constraints of libraries together.
* Easy access to those who understand JS.

#### Cons of ReactJS:

* ReactJS only serves the View tier, but the library size is on par with Angular while Angular is a complete framework.
* Incorporating ReactJS within common MVC frameworks demands reconﬁguration.
* Hard to reach for beginners on website development.

# Setup Instructions:

To develop a freelancing web application with the speciﬁed tech stack, these are the key prerequisites:

Technical Knowledge:

* Web Fundamentals: Proﬁciency in HTML, CSS, and JavaScript.
* Front-End Framework: Experience with React for building dynamic user interfaces.
* CSS Styling Libraries: Familiarity with Styled Components and MUI for styling React components.
* Server-Side Development: Knowledge of Node.js and Express for backend logic and API creation.
* Database Management: Understanding of MongoDB for data storage and retrieval.
* Authentication & Authorization: Knowledge of JWT for secure user authentication.
* Communication API: Familiarity with Twilio or similar APIs for SMS/email notiﬁcations
* Installation: Create conﬁg. env ﬁle in the backend folder and Fill your . env variables:

PORT= DATABASE= SECRET\_KEY=

* Install deps: npm install
* Run React server from Client folder : npm start
* Run Node.js Server : node server.js

# Folder Structure:

* **Client:** The client folder contains components, pages, services, and styles, with React managing the UI and Axios for API calls.
* **Server:** The server folder includes conﬁgurations, models, controllers, routes, and middleware for handling database, authentication, and business logic.

With current software, there are usually two parts:

**Client** side and **Server** side, also known as frontend and backend.

Therefore, people also split the backend stack, and the frontend stack as well. We often use the ﬁrst letter to name the technical stack: LAMP (Linux, Apache, MySQL, PHP), MEAN (MongoDB, Express, Angular, NodeJS).

# ShopZ E-commerce Web Application:

ShopeZ is an E-commerce Web Application using the MERN stack that can help companies bring their products to the customers.

**Main function:**

* *Sign up and log in:* Requires Users to register using their phone number or email
* *Shopping cart:* this feature helps users buy and check goods directly on

the application

* *Search:* Users can search directly by typing in the search box for the product they want to see.
* *Buy and pay:* Customers who buy through the app can pay through many diﬀerent payment gateways.

#### SOURCE CODE :

**FRONTEND:**

1. Index.html :

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />

<link

rel="stylesheet" type="text/css"

href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/f ont-awesome.min.css"

/>

<meta name="viewport" content="width=device-width, initial-scale=1" />

<meta name="theme-color" content="#000000" />

<meta

name="description"

content="Web site created using create-react-app"

/>

<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png"

/>

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

<title>SHOPEZ</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root"></div>

</body>

</html>

1. App.js :

import "./App.css";

import { BrowserRouter as Router, Route } from "react-router-dom";

import Home from "./pages/home/Home";

import Login from "./pages/auth/login/Login"; import Products from "./pages/products/Products";

import Register from "./pages/auth/register/Register"; import { useEffect, useState } from "react";

import store from "./store";

import { loadUser } from "./actions/userActions";

import Dashboard from "./pages/admin/dashboard/Dashboard"; import NewProduct from "./pages/admin/newProduct/NewProduct"; import ProductsList from "./pages/admin/products/ProductsList"; import ProtectedRoute from "./components/route/ProtectedRoute"; import ProductDetails from "./pages/admin/productDetails/ProductDetails";

import UpdateProduct from "./pages/admin/updateProduct/UpdateProduct";

import SingleProduct from "./pages/singleProduct/SingleProduct"; import Cart from "./pages/cart/Cart";

import Shipping from "./pages/cart/shipping/Shipping"; import ConfirmOrder from "./pages/cart/confirmOrder/ConfirmOrder";

// payment

import { Elements } from "@stripe/react-stripe-js"; import { loadStripe } from "@stripe/stripe-js"; import Payment from "./pages/cart/payment/Payment"; import Success from "./pages/cart/success/Success"; import Users from "./pages/admin/users/Users"; import UserDetails from "./pages/admin/users/userDetails/UserDetails"; import Orders from "./pages/admin/orders/Orders"; import ProcessOrder from "./pages/admin/orders/processOrder/ProcessOrder"; import Profile from "./pages/user/Profile";

import UpdateProfile from "./pages/user/updateProfile/UpdateProfile"; import ChangePassword from "./pages/user/changePassword/ChangePassword";

import MyOrders from "./pages/user/myOrders/MyOrders"; import OrderDetails from "./pages/user/orderDetails/OrderDetails";

import ProductReview from "./pages/admin/productReview/ProductReview"; import ForgotPassword from "./pages/auth/forgotPassword/ForgotPassword"; import ResetPassword from "./pages/auth/resetPassword/ResetPassword";

import About from "./pages/about/About"; import Contact from "./pages/contact/Contact"; import { axiosInstance } from "./config";

function App() {

const [stripeApiKey, setStripeApiKey] = useState(""); useEffect(() => {

store.dispatch(loadUser());

async function getStripApiKey() { const { data } = await

axiosInstance.get("/api/v1/stripeapi");

setStripeApiKey(data.stripeApiKey);

}

getStripApiKey();

}, []);

return (

<div className="app">

<Router>

<Route path="/" component={Home} exact />

<Route path="/login" component={Login} exact />

<Route path="/register" component={Register} exact />

<Route path="/password/forgot" component={ForgotPassword} exact />

<Route path="/password/reset/:token" component={ResetPassword} exact />

<Route path="/about" component={About} exact />

<Route path="/contact" component={Contact} exact />

<Route path="/products" component={Products} exact />

<Route path="/products/search/:keyword" component={Products} />

<Route path="/product/:id" component={SingleProduct} exact />

<Route path="/cart" component={Cart} exact />

<ProtectedRoute path="/me" component={Profile} exact />

<ProtectedRoute path="/me/update" component={UpdateProfile} exact />

<ProtectedRoute path="/me/password" component={ChangePassword} exact />

<ProtectedRoute path="/orders/me" component={MyOrders} exact />

<ProtectedRoute path="/order/:id" component={OrderDetails} exact />

<ProtectedRoute path="/shipping" component={Shipping} />

<ProtectedRoute path="/confirm" component={ConfirmOrder}

/>

{stripeApiKey && (

<Elements stripe={loadStripe(stripeApiKey)}>

<ProtectedRoute path="/payment" component={Payment}

/>

</Elements>

)}

<ProtectedRoute path="/success" component={Success} />

<ProtectedRoute path="/admin" isAdmin={true} component={Dashboard} exact

/>

<ProtectedRoute path="/admin/products/new" isAdmin={true} component={NewProduct} exact

/>

<ProtectedRoute path="/admin/products" isAdmin={true} component={ProductsList} exact

/>

<ProtectedRoute path="/admin/product/details/:id" component={ProductDetails}

exact

/>

<ProtectedRoute path="/admin/product/:id" isAdmin={true} component={UpdateProduct} exact

/>

<ProtectedRoute path="/admin/users" isAdmin={true} component={Users} exact

/>

<ProtectedRoute path="/admin/user/details/:id" isAdmin={true} component={UserDetails}

exact

/>

<ProtectedRoute path="/admin/orders" isAdmin={true} component={Orders}

exact

/>

<ProtectedRoute path="/admin/order/:id" isAdmin={true} component={ProcessOrder} exact

/>

<ProtectedRoute path="/admin/reviews" isAdmin={true} component={ProductReview} exact

/>

</Router>

</div>

);

}

export default App;

1. Index.js:

import React from "react";

import ReactDOM from "react-dom"; import "./index.css";

import App from "./App";

import reportWebVitals from "./reportWebVitals"; import { Provider } from "react-redux";

import store from "./store";

import "bootstrap/dist/css/bootstrap.min.css";

import { positions, transitions, Provider as AlertProvider } from "react-alert";

import AlertTemplate from "react-alert-template-basic";

const options = { timeout: 5000,

position: positions.BOTTOM\_CENTER, transition: transitions.SCALE,

};

ReactDOM.render(

<Provider store={store}>

<AlertProvider template={AlertTemplate} {...options}>

<App />

</AlertProvider>

</Provider>, document.getElementById("root")

);

reportWebVitals();

1. Store.js:

import { createStore, combineReducers, applyMiddleware } from "redux";

import thunk from "redux-thunk";

import { composeWithDevTools } from "redux-devtools-extension"; import {

newProductReducer, newReviewReducer, productDetailsReducer, productReducer, productReviewsReducer, productsReducer, reviewReducer,

} from "./reducers/productReducers"; import {

allUsersReducer, authReducer, forgotPasswordReducer, userDetailsReducer, userReducer,

} from "./reducers/userReducers";

import { cartReducer } from "./reducers/cartReducers"; import {

allOrdersReducer, myOrdersReducer, newOrderReducer, orderDetailsReducer, orderReducer,

} from "./reducers/orderReducers";

const reducer = combineReducers({ auth: authReducer,

forgotPassword: forgotPasswordReducer, products: productsReducer,

newProduct: newProductReducer, productDetails: productDetailsReducer, product: productReducer,

cart: cartReducer, newOrder: newOrderReducer, allUsers: allUsersReducer, user: userReducer,

userDetails: userDetailsReducer, allOrders: allOrdersReducer,

order: orderReducer,

orderDetails: orderDetailsReducer, myOrders: myOrdersReducer, productReviews: productReviewsReducer, review: reviewReducer,

newReview: newReviewReducer,

});

let initialState = { cart: {

cartItems: localStorage.getItem("cartItems")

? JSON.parse(localStorage.getItem("cartItems"))

: [],

shippingInfo: localStorage.getItem("shippingInfo")

? JSON.parse(localStorage.getItem("shippingInfo"))

: {},

},

};

const middlware = [thunk]; const store = createStore(

reducer, initialState,

composeWithDevTools(applyMiddleware(...middlware))

);

export default store;

1. Navbar:

import React from "react";

import styles from "./Navbar.module.scss";

const Navbar = () => { return (

<div className={styles.navbar}>

<h3>Admin</h3>

</div>

);

};

export default Navbar;

1. Loader:

import React from "react"; import "./ButtonLoader.scss";

const ButtonLoader = () => {

return <div className="lds-dual-ring"></div>;

};

export default ButtonLoader;

1. Header.jsx:

import React from 'react';

import { useNavigate } from 'react-router-dom';

import { Navbar, Nav, Container, Badge, NavDropdown } from 'react-bootstrap';

import { FaShoppingCart, FaUser } from 'react-icons/fa'; import { LinkContainer } from 'react-router-bootstrap'; import { useSelector, useDispatch } from 'react-redux'; import { useLogoutMutation } from '../slices/usersApiSlice'; import { logout } from '../slices/authSlice';

import { toast } from 'react-toastify'; import SearchBox from './SearchBox';

const Header = () => {

const { cartItems } = useSelector(state => state.cart); const { userInfo } = useSelector(state => state.auth); const [logoutApiCall] = useLogoutMutation();

const dispatch = useDispatch(); const navigate = useNavigate();

const logoutHandler = async () => { try {

await logoutApiCall().unwrap(); dispatch(logout());

navigate('/login'); toast.success('Logout successful');

} catch (error) {

toast.error(error?.data?.message || error.error);

}

};

return (

<Navbar bg='dark' variant='dark' expand='md'

collapseOnSelect className='fixed-top z-2 '

>

<Container>

<LinkContainer to='/'>

<Navbar.Brand>MERN Shop</Navbar.Brand>

</LinkContainer>

<Navbar.Toggle aria-controls='basic-navbar-nav' />

<Navbar.Collapse id='basic-navbar-nav'>

<Nav className='ms-auto m-2'>

<SearchBox />

<LinkContainer to='/cart'>

<Nav.Link>

<FaShoppingCart style={{ marginRight: '5px' }} /> Cart

{cartItems.length > 0 && (

<Badge

pill bg='warning'

style={{ marginLeft: '5px' }} className='text-dark'

item.qty, 0)}

>

<strong>

{cartItems.reduce((acc, item) => acc +

</strong>

</Badge>

)}

</Nav.Link>

</LinkContainer>

{userInfo ? (

<NavDropdown title={`Hello, ${userInfo.name}`} id='username'>

<LinkContainer to='/profile'>

<NavDropdown.Item>Profile</NavDropdown.Item>

</LinkContainer>

<NavDropdown.Item onClick={logoutHandler}> Logout

</NavDropdown.Item>

</NavDropdown>

) : (

<LinkContainer to='/login'>

<Nav.Link>

<FaUser style={{ marginRight: '5px' }} /> Sign In

</Nav.Link>

</LinkContainer>

)}

{/\* {userInfo && userInfo.isAdmin && (

<NavDropdown title='Admin' id='adminmenu'>

<LinkContainer to='/admin/product-list'>

<NavDropdown.Item>Products</NavDropdown.Item>

</LinkContainer>

<LinkContainer to='/admin/order-list'>

<NavDropdown.Item>Orders</NavDropdown.Item>

</LinkContainer>

<LinkContainer to='/admin/user-list'>

<NavDropdown.Item>Users</NavDropdown.Item>

</LinkContainer>

</NavDropdown>

)} \*/}

</Nav>

</Navbar.Collapse>

</Container>

</Navbar>

);

};

export default Header;

1. Product.jsx:

import React, { useState } from 'react';

import { Button, Card } from 'react-bootstrap'; import { Link, useNavigate } from 'react-router-dom'; import { useDispatch } from 'react-redux';

import { addCurrency } from '../utils/addCurrency'; import { addToCart } from '../slices/cartSlice'; import Rating from './Rating';

const Product = ({ product }) => { const [qty, setQty] = useState(1); const dispatch = useDispatch(); const navigate = useNavigate();

const addToCartHandler = () => { dispatch(addToCart({ ...product, qty })); navigate('/cart');

};

return (

<Card className='my-3 p-3 rounded text-center'>

<Link

to={`/product/${product.\_id}`} style={{ textDecoration: 'none' }} className='text-dark'

>

<Card.Img variant='top' src={product.image}

style={{ height: '200px', objectFit: 'contain' }}

/>

<Card.Body>

<Card.Title as='div' className='product-title'>

<strong>{product.name}</strong>

</Card.Title>

<Card.Text as='div' className='mb-3'>

<Rating value={product.rating}

text={`(${product.numReviews} reviews)`}

/>

</Card.Text>

<Card.Text as='h3'>{addCurrency(product.price)}</Card.Text>

</Card.Body>

</Link>

<Button variant='warning' type='button'

disabled={product.countInStock === 0} onClick={addToCartHandler}

>

Add To Cart

</Button>

</Card>

);

};

export default Product;

1. ProﬁeLinks.jsx:

import React, { Fragment } from "react";

import { useDispatch, useSelector } from "react-redux"; import { Link } from "react-router-dom";

import Loader from "../loader/Loader";

import styles from "./ProfileLink.module.scss";

import { AiOutlineEdit, AiOutlineLogout, AiOutlineUser } from "react-icons/ai";

import { RiLockPasswordLine } from "react-icons/ri"; import { MdFavoriteBorder } from "react-icons/md"; import { logout } from "../../actions/userActions"; import { useAlert } from "react-alert";

const ProfileLink = () => {

const { user, loading } = useSelector((state) => state.auth);

const alert = useAlert();

const dispatch = useDispatch();

const logoutHandler = () => { dispatch(logout());

alert.success("Logged out successfully.");

};

return (

<Fragment>

{loading ? (

<Loader />

) : (

<Fragment>

<div className={styles.profile\_links}>

<div className="text-center mt-3">

{user && (

<img src={user.avatar.url}

alt={user?.name} />

)}

<h4

className="mt-3">{user?.name}</h4>

<p>{user?.email}</p>

</div>

<hr className="text-primary" />

size={25} />{" "}

size={25} />{" "}

<div className={`mt-3 ${styles.links}`}>

<Link to="/me">

<AiOutlineUser className="me-3"

Profile

</Link>

<Link to="/me/update">

<AiOutlineEdit className="me-3"

Edit Profile

</Link>

<Link to="/me/password">

<RiLockPasswordLine className="me-3" size={25}

/> Password

</Link>

<Link to="/orders/me">

<MdFavoriteBorder

className="me-3" size={25} />

My Order

size={25} />

)}

</Link>

<button onClick={logoutHandler}>

<AiOutlineLogout className="me-3"

Logout

</button>

</div>

</div>

</Fragment>

</Fragment>

);

};

export default ProfileLink;

1. HomePage.jsx:

import React, { Fragment, useEffect } from "react"; import { useAlert } from "react-alert";

import { useDispatch, useSelector } from "react-redux";

import { getAdminProducts } from "../../actions/productAction"; import Footer from "../../components/footer/Footer";

import Navbar from "../../components/header/Navbar"; import Loader from "../../components/loader/Loader"; import MetaData from "../../components/MetaData"; import Banner from "./banner/Banner";

import Category from "./category/Category"; import Fashion from "./fashion/Fashion";

const Home = () => {

const alert = useAlert();

const dispatch = useDispatch();

const { loading, error, products } = useSelector((state) => state.products);

// filter products by types

const mens = products.filter((item) => item.type === "Men"); const womens = products.filter((item) => item.type ===

"Women");

const kids = products.filter((item) => item.type === "Kids");

useEffect(() => { dispatch(getAdminProducts());

if (error) {

return alert.error(error);

}

}, [dispatch, alert, error]); return (

<Fragment>

<MetaData title={"Home"} />

<Navbar />

<Banner />

<Category />

{loading ? (

<>

<Loader />

</>

) : (

<>

<Fashion products={mens} type="mens" />

<Fashion products={womens} type="womens" />

<Fashion products={kids} type="kids" />

</>

)}

<Footer />

</Fragment>

);

};

export default Home;

# BACKEND:

Db.js:

import mongoose from 'mongoose'; const connectDB = async () => {

try {

const connection = await mongoose.connect(process.env.MONGO\_URI);

// console.log(

// `MongoDB connected successfully on host:

${connection.connection.host}, database:

${connection.connection.db.databaseName}`

// );

return connection;

} catch (error) {

console.error(`MongoDB connection error: ${error.message}`); process.exit(1);

}

};

export default connectDB;

# . API Documentation:

The API supports a range of operations, including user authentication, product management, and payment processing. It provides endpoints for secure registration, login, and order tracking. The use of JWT ensures that only authenticated users can access protected routes, such as viewing purchase history or managing products.

The API should support the following:

*Authentication:* User registration, login, password reset.

*Product Management:* Add, update, and delete products (admin/vendor). *Cart Management:* Add to cart, update quantities, remove items.

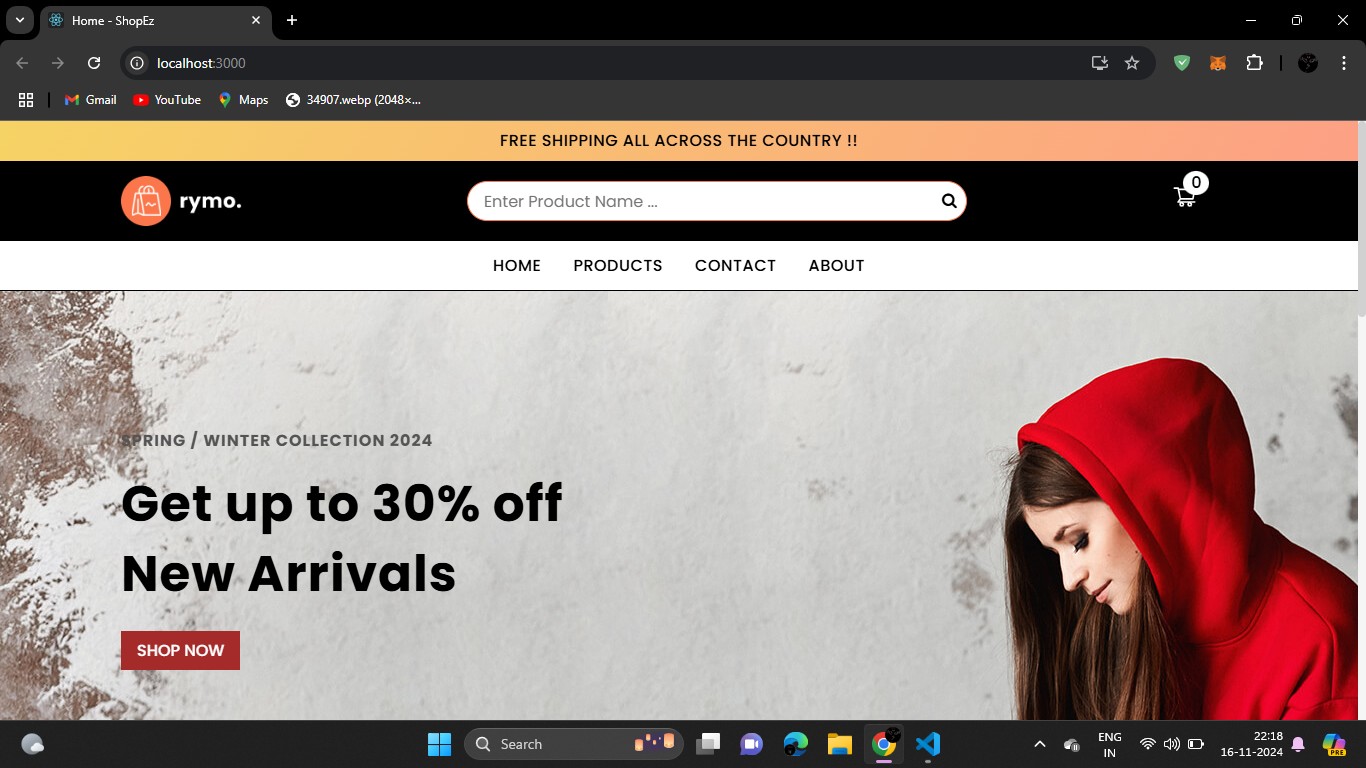
*Order Processing:* Place orders, view order history.

*Payment Gateway:* Secure payment APIs for transactions.

# . Authentication:

Authentication in an e-commerce application is a critical feature that ensures secure and personalised interactions for users. It typically includes the following components:

* 1. ***User Registration:*** Users, whether customers or vendors, can create accounts by providing necessary details like email, password, and other relevant information. Passwords are securely hashed using a library like bcrypt before being stored in the database (e.g., MongoDB). This process ensures that user credentials are protected even if the database is compromised.
  2. ***User Login:*** Registered users can log in by entering their email and password. The application validates the credentials by comparing the entered password with the hashed password stored in the database. Upon successful authentication, a JSON Web Token (JWT) is generated and sent to the client.
  3. ***JWT-Based Authentication:*** The client securely stores the JWT (typically in local storage or cookies) and includes it in the Authorization header of all subsequent requests to protected routes. The server veriﬁes the token to ensure the user is authenticated and authorised to perform speciﬁc actions.
  4. ***Role-Based Access Control (RBAC):*** Diﬀerent user roles (e.g., customer, vendor, admin) may require diﬀerent levels of access. Middleware can validate the user's role to allow or restrict access to speciﬁc functionalities, such as managing products (vendor) or viewing analytics (admin).
  5. ***Session Management:*** Sessions are maintained using JWTs, and expiration times are set to balance security and user convenience. Users are logged out automatically once the token expires, or they can manually log out by clearing the JWT from the client-side storage.
  6. ***Protected Routes:*** Certain routes, like viewing the cart, placing an order, or managing inventory, are restricted to authenticated users. Middleware ensures that these routes are accessible only to users with valid tokens.

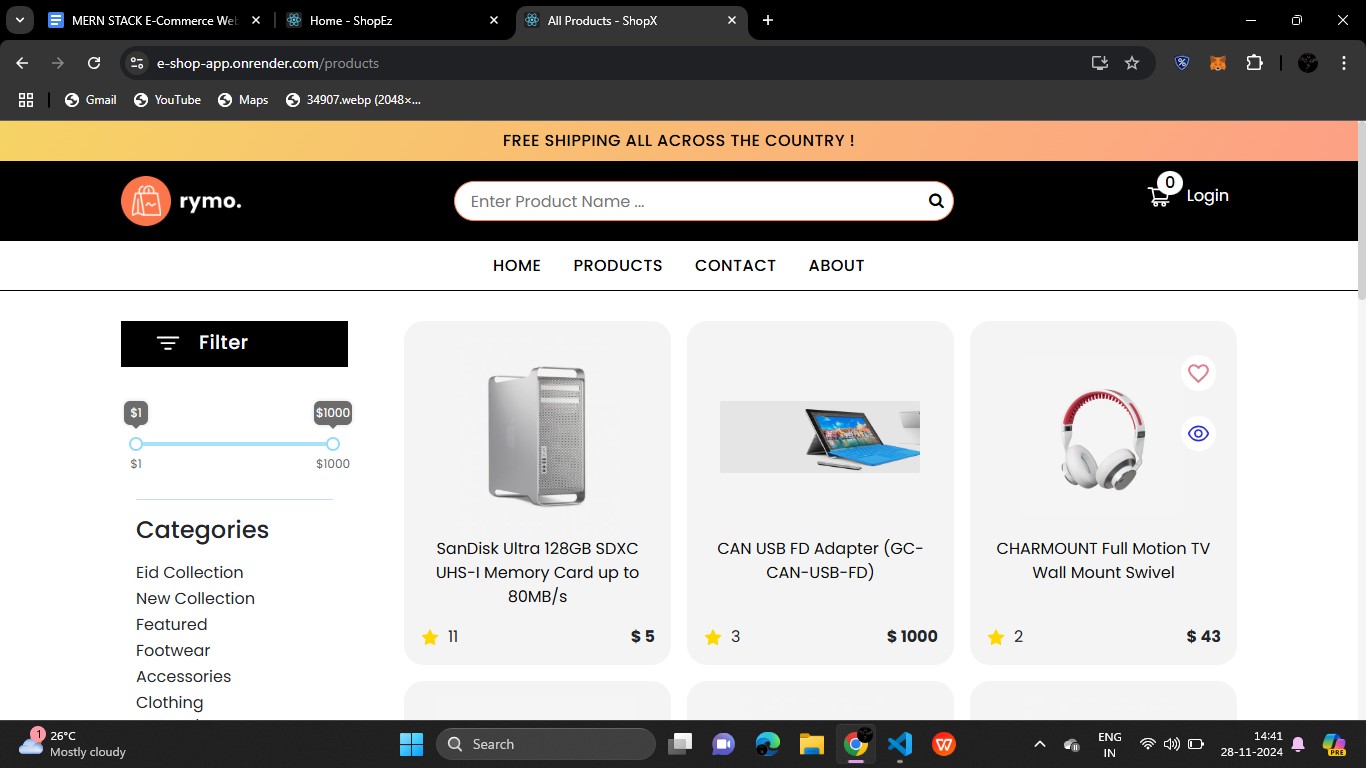


* 1. ***Password Reset and Account Recovery:*** Users can reset their passwords if they forget them by requesting a password reset link. A unique token is sent to the user’s registered email, allowing them to securely set a new password.
  2. ***Activity Logging:*** To improve security and provide transparency, user activities like login attempts, password changes, and account updates can be logged. This helps identify potential breaches or unauthorised access. Authentication not only secures user data but also ensures that the platform provides a seamless and personalised shopping experience, enabling features like saved carts, order history, and secure payment processing.

# User Interface:

* 1. **Testing:**
     1. **Functional Testing**:Check homepage, search, ﬁlters, product pages, cart, and checkout.Ensure payment processing works correctly.
     2. **User Experience**:Test navigation, responsiveness (desktop/mobile), and

UI consistency.Check load speed and ease of use.



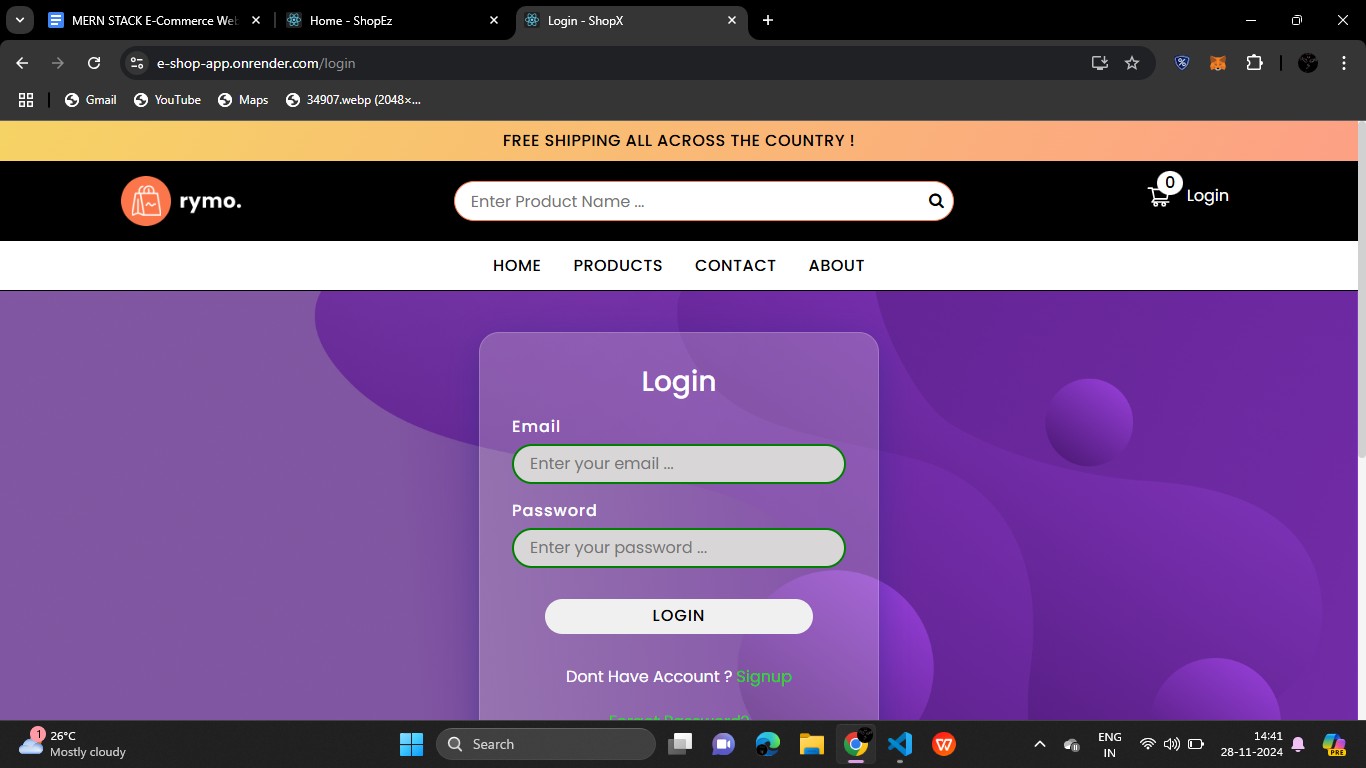
* + 1. **Security Testing**:Test login, payment security (SSL), and data protection.Verify session management (logout, timeout).
    2. **Performance Testing**:Load and stress test for high traﬃc and scalability.
    3. **Compatibility Testing**:Ensure compatibility across browsers and devices.
    4. **Integration Testing**:Test third-party services (payment, shipping, email).
    5. **Compliance**:Check GDPR, PCI compliance, and accessibility (WCAG).
    6. **End-to-End Testing**:Test the entire purchase ﬂow, from browsing to order conﬁrmation.

Tools: Selenium, JMeter, Postman, Google Lighthouse, OWASP

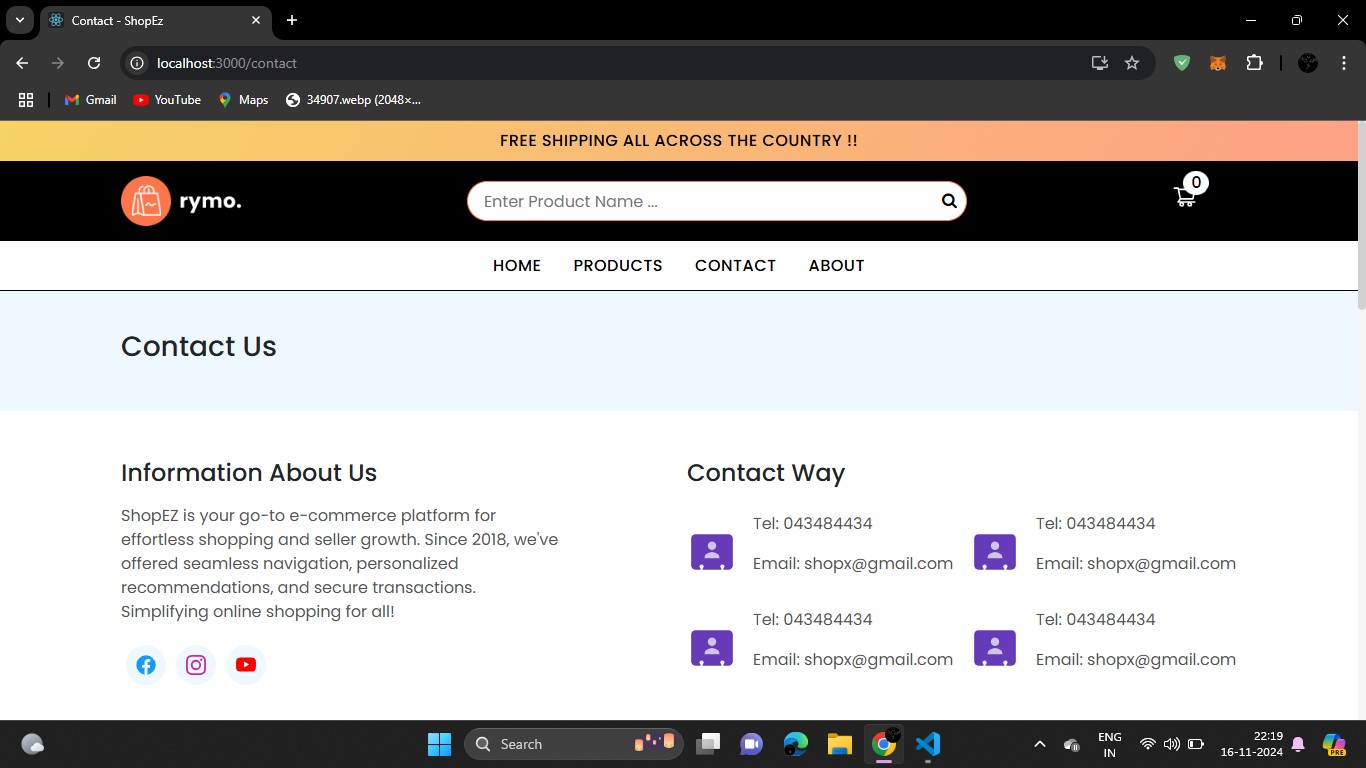
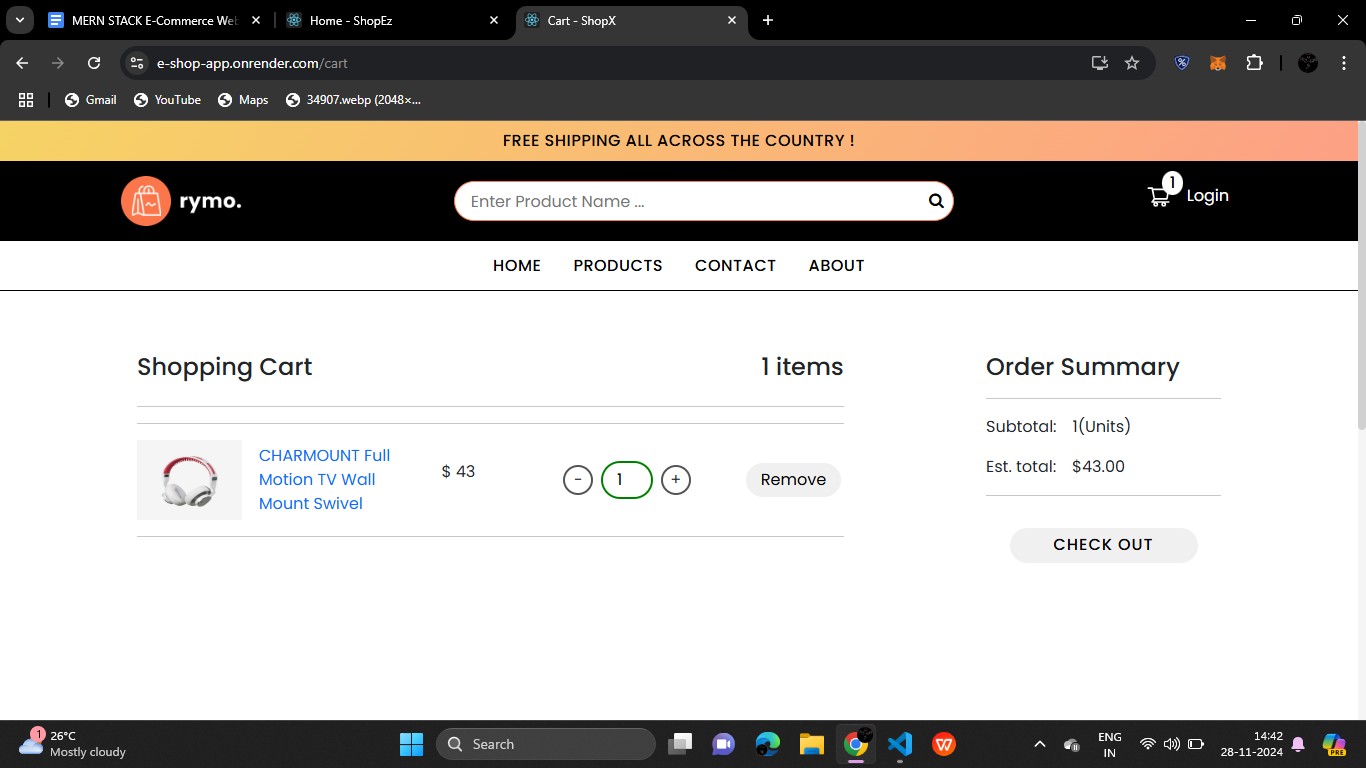
**Screenshots:**

## Products:

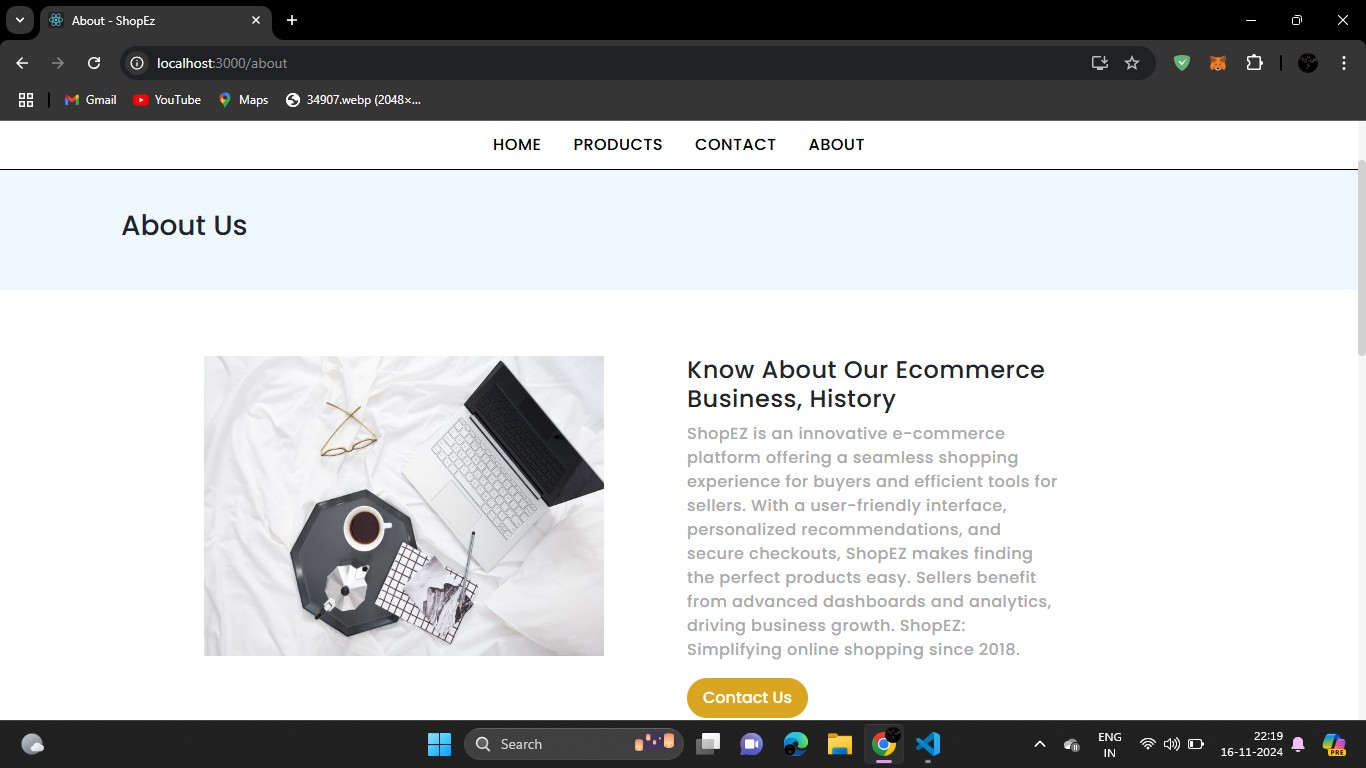
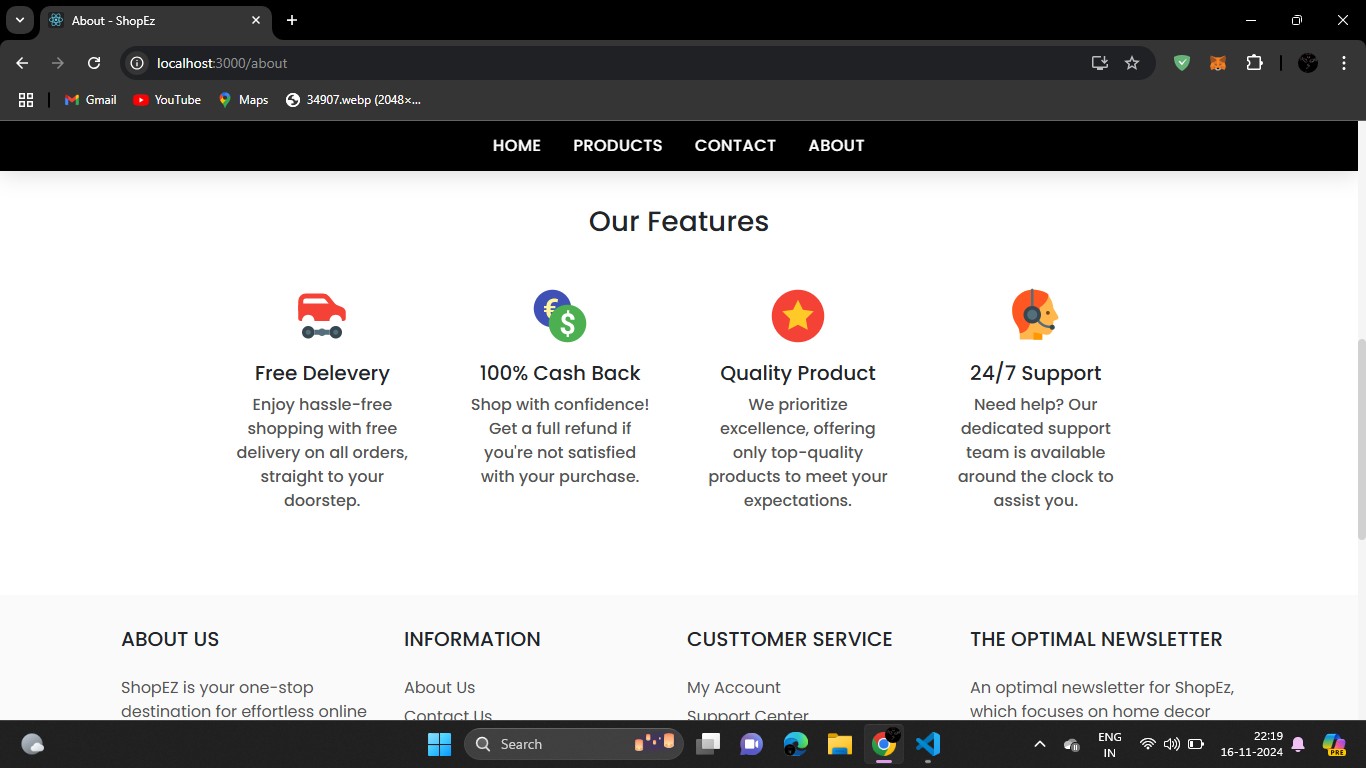
**LoginPage:**



**Cart Page**



**Contact:**



**About:**

**Demo:**

[https://drive.google.com/ﬁle/d/1ajdaeB6kga2ljvOcvEMLRInZVoDXOcwH/vi](https://drive.google.com/file/d/1ajdaeB6kga2ljvOcvEMLRInZVoDXOcwH/view?usp=drivesdk) [ew?usp=drivesdk](https://drive.google.com/file/d/1ajdaeB6kga2ljvOcvEMLRInZVoDXOcwH/view?usp=drivesdk)

# Known Issues:

1. **Mobile Responsiveness:** Some UI elements may not align correctly on smaller screens, causing slight layout inconsistencies.
2. **Search Filter Performance:** Course search and ﬁltering can be slow with a large number of courses.
3. **Video Playback Compatibility:** Certain video formats may not play on older devices or browsers.

# . Future Enhancements:

* 1. **Advanced Search Filters:** Add ﬁlters for course categories, diﬃculty levels, and duration for better search results.
  2. **User Ratings & Reviews:** Enable learners to rate courses and provide

feedback to guide others.

* 1. ***Gamiﬁcation Features:*** Introduce badges, points, or leaderboards to enhance learner engagement.
  2. **Payment Integration:** Implement secure payment options for premium courses or certiﬁcations.

# . Conclusion :

In conclusion, the "Shopez" e-commerce website project successfully developed a user-friendly online platform that enables seamless product browsing, secure payment processing, and convenient order management, allowing customers to access a wide range of products with ease while providing the store with an efficient means to expand its reach and sales potential. By incorporating key features like intuitive navigation, detailed product descriptions, and robust customer support, Shopez aims to enhance customer satisfaction and establish a strong online presence in the market,

paving the way for future growth and expansion through targeted marketing strategies and continuous optimization based on user feedback.