ShopEZ:One-Stop Shop for Online Purchases

INTRODUCTION:

ShopEZ is your one-stop destination for effortless online shopping. With a user-friendly interface and a comprehensive product catalog, finding the perfect items has never been easier. Seamlessly navigate through detailed product descriptions, customer reviews, and available discounts to make informed decisions. Enjoy a secure checkout process and receive instant order confirmation. For sellers, our robust dashboard provides efficient order management and insightful analytics to drive business growth. Experience the future of online shopping with ShopEZ today.

Seamless Checkout Process

Effortless Product Discovery

Personalized Shopping Experience

Efficient Order Management for Sellers

Insightful Analytics for Business Growth

Scenario: Sarah's Birthday Gift

Sarah, a busy professional, is scrambling to find the perfect birthday gift for her best friend, Emily. She knows Emily loves fashion accessories, but with her hectic schedule, she hasn't had time to browse through multiple websites to find the ideal present. Feeling overwhelmed, Sarah turns to ShopEZ to simplify her search.

1. Effortless Product Discovery: Sarah opens ShopEZ and navigates to the fashion accessories category. She's greeted with a diverse range of options, from chic handbags to elegant jewelry. Using the filtering options, Sarah selects "bracelets" and refines her search based on Emily's preferred style and budget.

2. Personalized Recommendations: As Sarah scrolls through the curated selection of bracelets, she notices a section labeled "Recommended for You." Intrigued, she clicks on it and discovers a stunning gold bangle that perfectly matches Emily's taste. Impressed by the personalized recommendation, Sarah adds it to her cart.

3.Seamless Checkout Process: With the bracelet in her cart, Sarah proceeds to checkout. She enters Emily's address as the shipping destination and selects her preferred payment method.

Thanks to ShopEZ's secure and efficient checkout process, Sarah completes the transaction in just a few clicks.

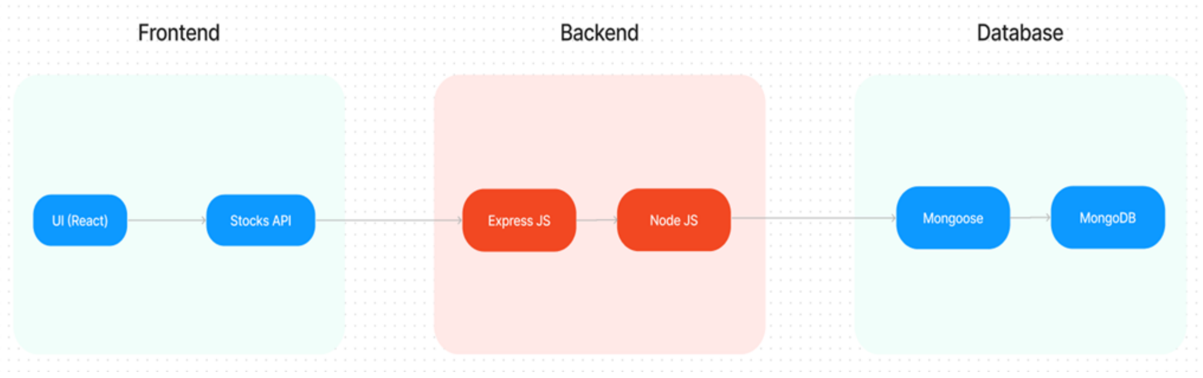
4. Order Confirmation: Moments after placing her order, Sarah receives a confirmation email from ShopEZ. Relieved to have found the perfect gift for Emily, she eagerly awaits its arrival.

5. Efficient Order Management for Sellers: Meanwhile, on the other end, the seller of the gold bangle receives a notification of Sarah's purchase through ShopEZ's seller dashboard. They quickly process the order and prepare it for shipment, confident in ShopEZ's streamlined order management system.

6. Celebrating with Confidence: On Emily's birthday, Sarah presents her with the beautifully packaged bracelet, knowing it was chosen with care and thoughtfulness. Emily's eyes light up with joy as she adorns the bracelet, grateful for Sarah's thoughtful gesture.

In this scenario, ShopEZ proves to be the perfect solution for Sarah's busy lifestyle, offering a seamless and personalized shopping experience. From effortless product discovery to secure checkout and efficient order management, ShopEZ simplifies the entire process, allowing Sarah to celebrate Emily's birthday with confidence and ease.

TECHNICAL ARCHITECTURE:



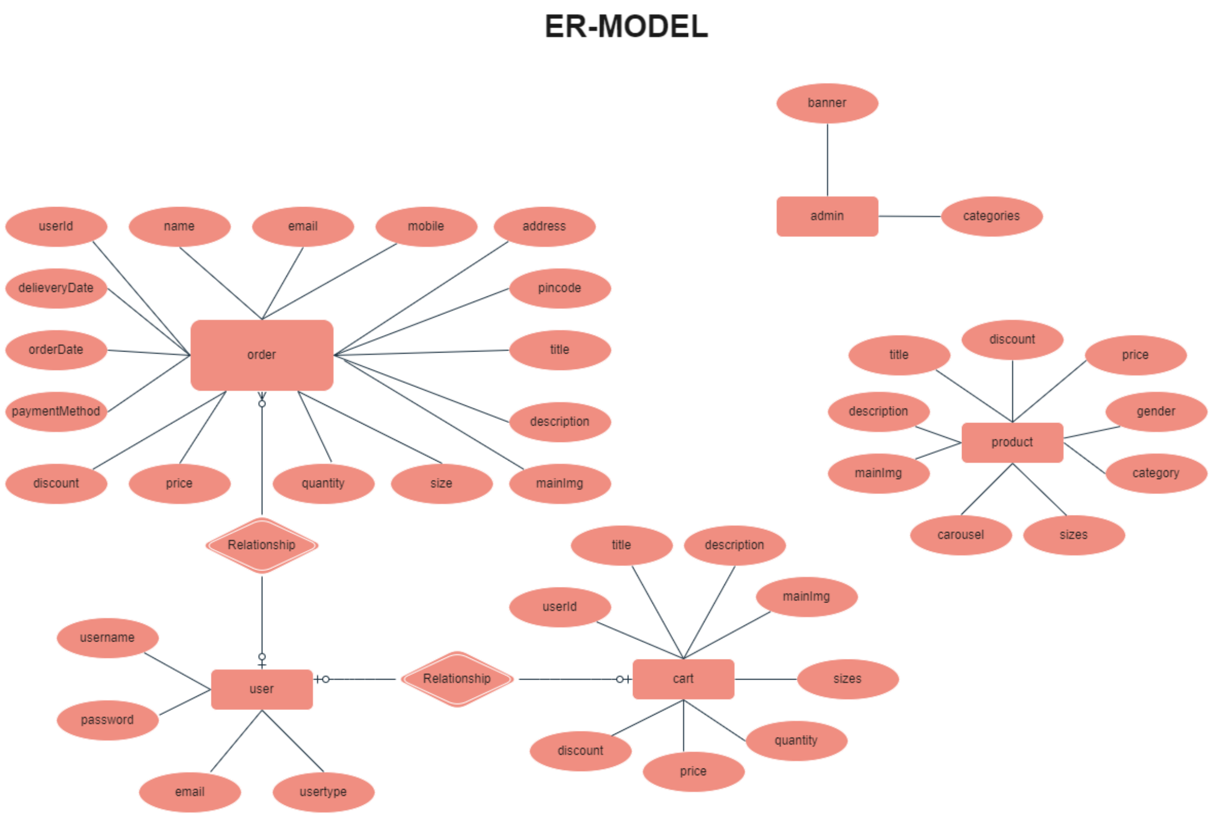
In this architecture diagram:

• The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Cart, Products, Profile, Admin dashboard, etc.,

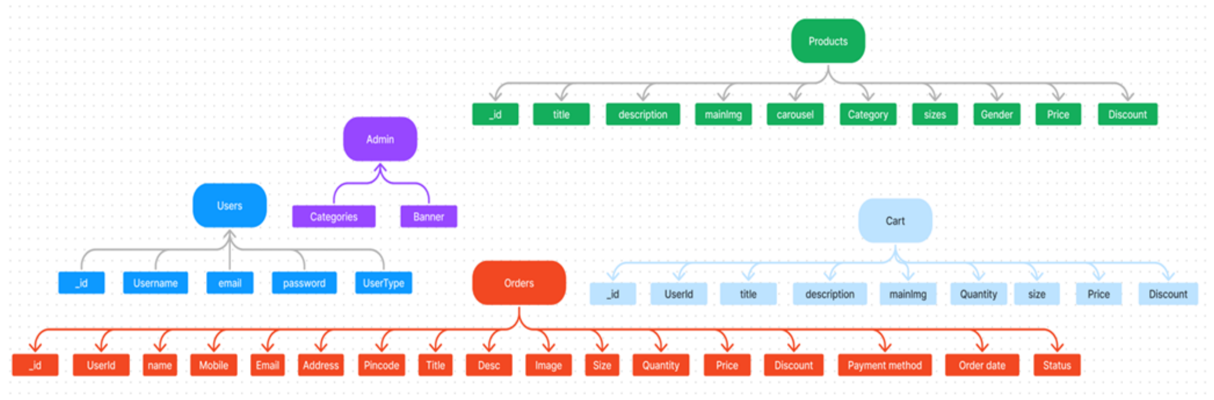
• The backend is represented by the "Backend" section, consisting of API endpoints for Users, Orders, Products, etc.,It also includes Admin Authentication and an Admin Dashboard.

• The Database section represents the database that stores collections for Users, cart, Orders and Product.

ER Diagram



• The Database section represents the database that stores collections for Users, Admin, Cart, Orders and products.



The ShopEZ ER-diagram represents the entities and relationships involved in an e-commerce system. It illustrates how users, products, cart, 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.

Admin: Represents a collection with important details such as Banner image and Categories.

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

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

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

Features:

1. **Comprehensive Product Catalog:** ShopEZ boasts an extensive catalog of products, 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 items for your needs.

2. Shop Now Button: Each product listing features a convenient "Shop Now" button. When you find a product that aligns with your preferences, simply click on the button to initiate the purchasing process.

3. **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.

4. **Secure and Efficient Checkout Process:** ShopEZ 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.

5. **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, ShopEZ provides a robust seller dashboard, offering sellers an array of functionalities to efficiently manage their products and sales. With the seller dashboard, sellers can add and oversee multiple product listings, view order history, monitor customer activity, and access order details for all purchases.

ShopEZ is designed to elevate your online shopping experience by providing a seamless and user-friendly way to discover and purchase products. With our efficient checkout process, comprehensive product catalog, and robust seller dashboard, we ensure a convenient and enjoyable online shopping experience for both shoppers and sellers alike.

Pre-requisites

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

Node.js and npm:

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

• Download: https://nodejs.org/en/download/

• Installation instructions: https://nodejs.org/en/download/package-manager/

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

• Download: https://www.mongodb.com/try/download/community

• Installation instructions: https://docs.mongodb.com/manual/installation/

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

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

React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. To install React.js, a JavaScript library for building user interfaces, follow the installation guide:

https://reactjs.org/docs/create-a-new-react-app.html

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

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

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

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

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

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

• Visual Studio Code: Download from https://code.visualstudio.com/download

• Sublime Text: Download from https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download

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

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

To run the existing ShopEZ App project downloaded from github: Follow below steps:

Clone the repository:

• Open your terminal or command prompt.

• Navigate to the directory where you want to store the e-commerce app. • Execute the following command to clone the repository:

Drive Link:

https://drive.google.com/drive/folders/1_6byLUBVSLIFut5Kv8ozuCoff3TEr7u1?usp=sharin
g

Install Dependencies:

• Navigate into the cloned repository directory:

cd ShopEZ—e-commerce-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 ShopEZ app on your local machine. You can now proceed with further customization, development, and testing as needed.

Application Flow

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. Admin Flow:

• Admins start by logging in with their credentials.

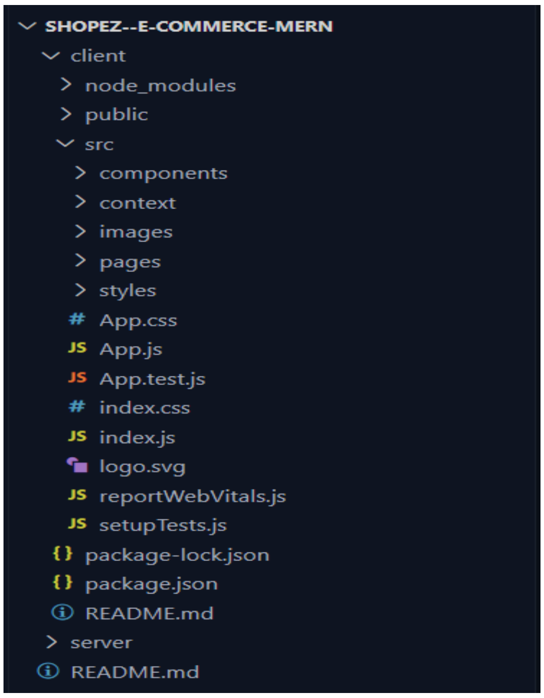
• Once logged in, they are directed to the Admin Dashboard.

• Admins can access the users list, products, orders, etc.,

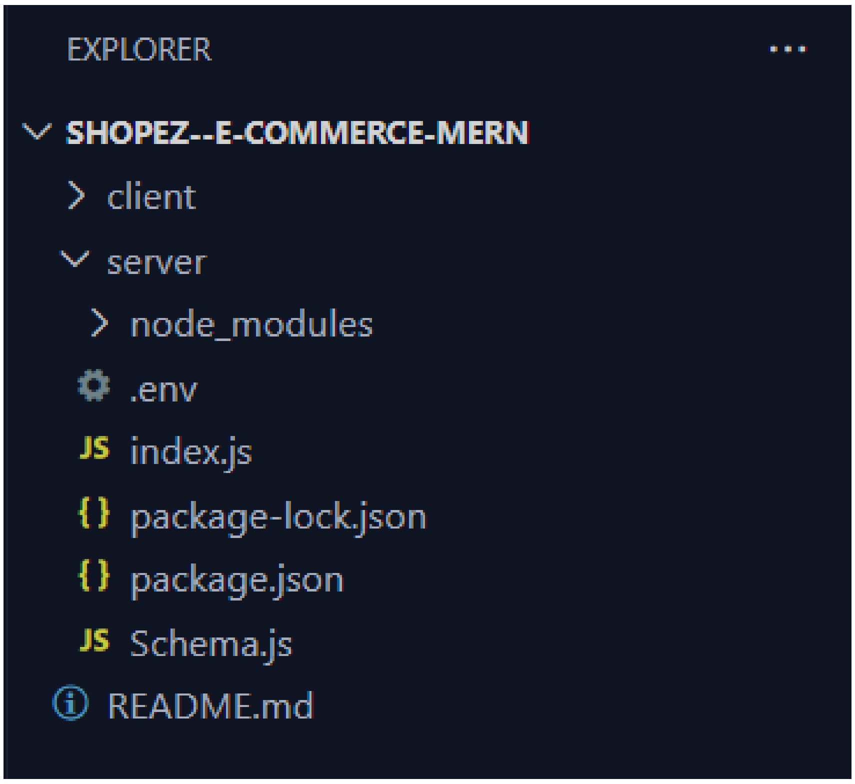
Project Structure:

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.



```
SHOPEZ--E-COMMERCE-MERN
  client
    node_modules
    public
    src
      components
      context
      images
      pages
      styles
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
      reportWebVitals.js
      setupTests.js
    package-lock.json
    package.json
    README.md
  server
  README.md
```


```
EXPLORER
SHOPEZ--E-COMMERCE-MERN
  client
  server
    node_modules
    .env
    index.js
    package-lock.json
    package.json
    Schema.js
  README.md
```

Project Flow:

Project Setup And Configuration:

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

• Node.js.

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

• Git.

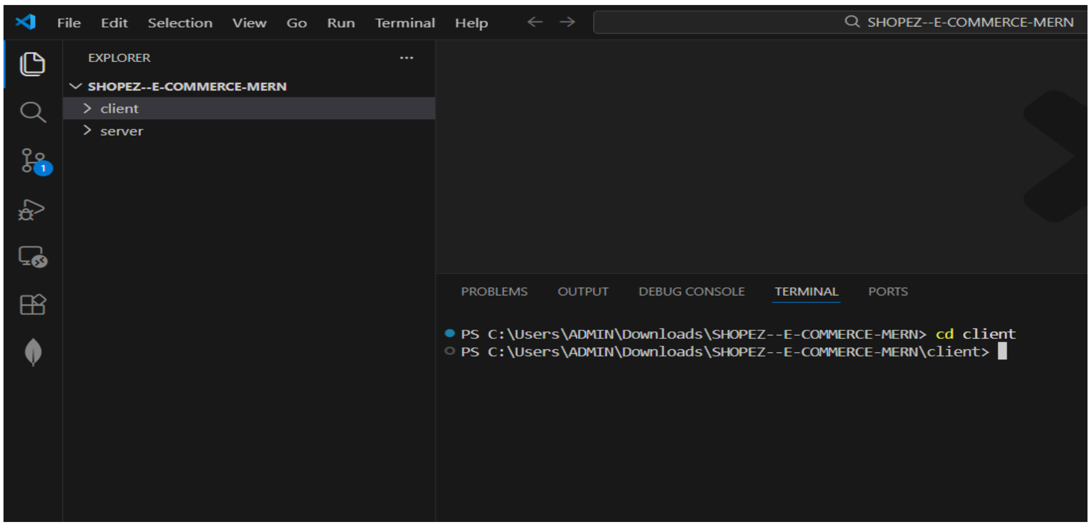
Reference Article: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

2. Create project folders and files:

• Client folders.

• Server folders

Referral Image:


File Edit Selection View Go Run Terminal Help
SHOPEZ--E-COMMERCE-MERN
EXPLORER
SHOPEZ--E-COMMERCE-MERN
client
server
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN> cd client
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN\client>
```

Backend Development

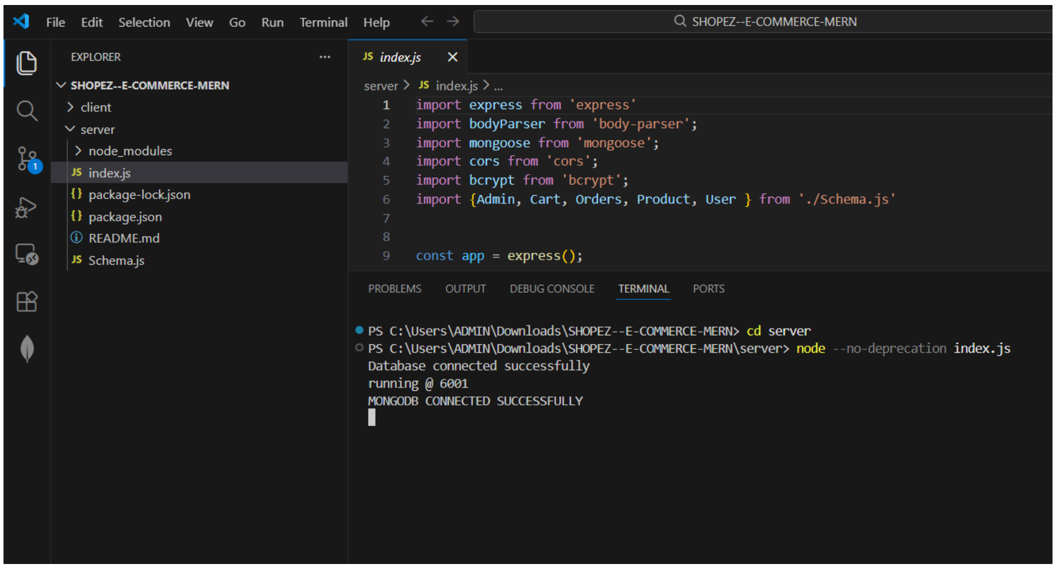
1. Setup express server:

• Create index.js file.

• Create an express server on your desired port number.

• Define API's

Reference Images:



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


const app = express();
```

```
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN> cd server
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN\server> node --no-deprecation index.js
Database connected successfully
running @ 6001
MONGODB CONNECTED SUCCESSFULLY
```

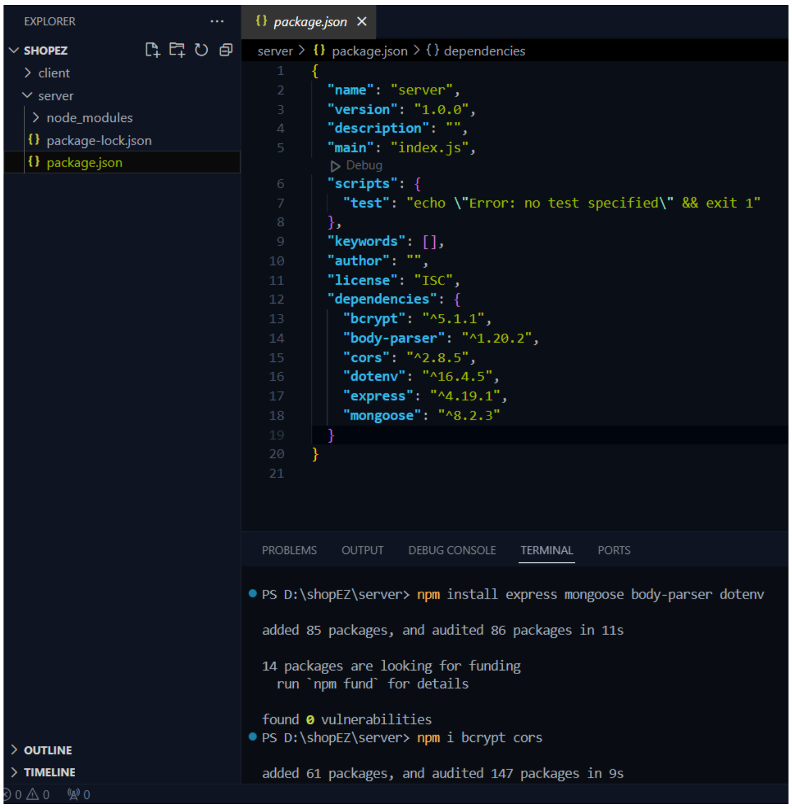
Now your express is successfully created.

Set Up Project Structure:

• Create a new directory for your project and set up a package.json file using the npm init command.

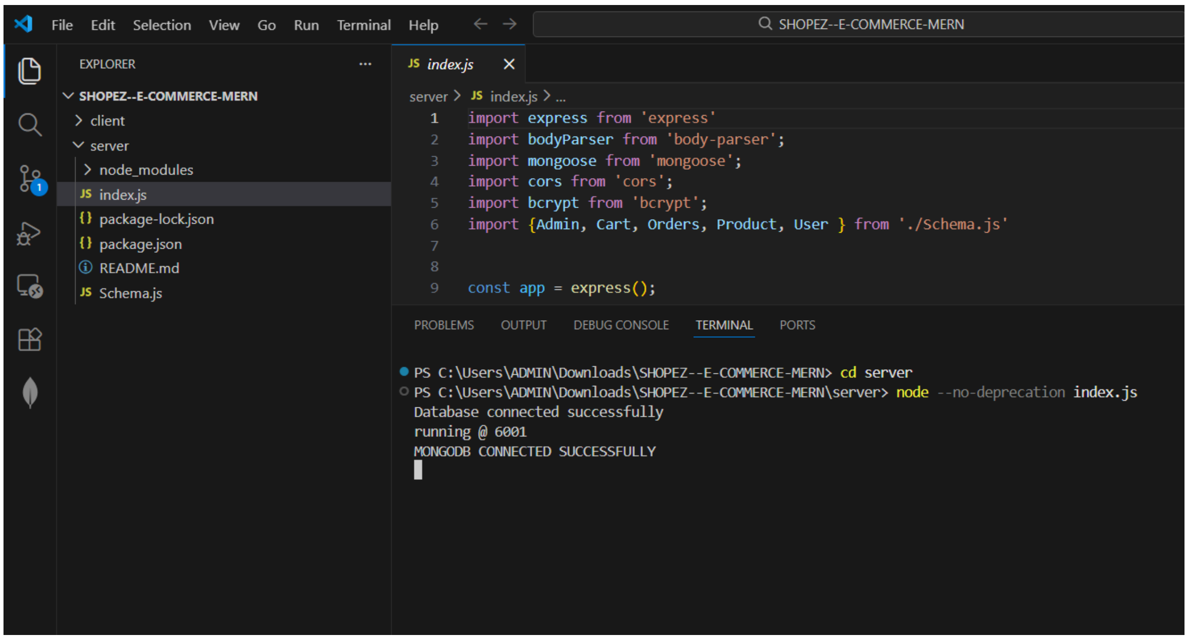
• Install necessary dependencies such as Express.js, Mongoose, and other required packages.

Reference Images:



```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.1",
    "mongoose": "^8.2.3"
  }
}
```

```
PS D:\shopEZ\server> npm install express mongoose body-parser dotenv

added 85 packages, and audited 86 packages in 11s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\shopEZ\server> npm i bcrypt cors

added 61 packages, and audited 147 packages in 9s
```



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


const app = express();
```

```
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN> cd server
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN\server> node --no-deprecation index.js
Database connected successfully
running @ 6001
MONGODB CONNECTED SUCCESSFULLY
```

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, products, orders and other relevant data.

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.

4. Define API Routes:

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

• Define the necessary routes for listing products, handling user registration
and login,managing orders, etc.

• Implement route handlers using Express.js to handle requests and interact with the database.

5. Implement Data Models:

• Define Mongoose schemas for the different data entities like products, users, and orders.

• Create corresponding Mongoose models to interact with the MongoDB database.

• Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

6. User Authentication:

• Create routes and middleware for user registration, login, and logout.

• Set up authentication middleware to protect routes that require user authentication.

7. Handle new products and Orders: • Create routes and controllers to handle new product listings, including fetching products data from the database and sending it as a response.

• Implement ordering(buy) functionality by creating routes and controllers to handle order requests, including validation and database updates.

8. Admin Functionality:

• Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.

• Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.

9. Error Handling:

• Implement error handling middleware to catch and handle any errors that occur during the API requests.

• Return appropriate error responses with relevant error messages and HTTP status codes.

Database Development

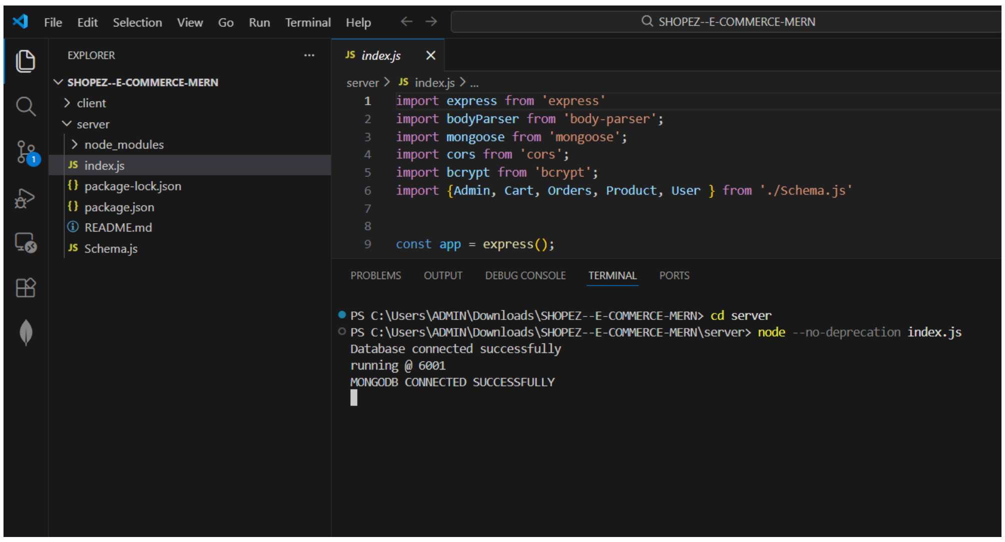
Create database in cloud

• Install Mongoose.

• Create database connection.

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

Reference Image:



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


const app = express();
```

```
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN> cd server
PS C:\Users\ADMIN\Downloads\SHOPEZ--E-COMMERCE-MERN\server> node --no-deprecation index.js
Database connected successfully
running @ 6001
MONGODB CONNECTED SUCCESSFULLY
```

Schema use-case:

1. User Schema:

• Schema: userSchema

• Model: ‘User’

• The User schema represents the user data and includes fields such as username, email, and password.

• It is used to store user information for registration and authentication purposes.

• The email field is marked as unique to ensure that each user has a unique email address

2. Product Schema:

• Schema: productSchema

• Model: ‘Product’

• The Product schema represents the data of all the products in the platform.

• It is used to store information about the product details, which will later be useful for ordering .

3. Orders Schema:

• Schema: ordersSchema

• Model: ‘Orders’

• The Orders schema represents the orders data and includes fields such as userId, product Id, product name, quantity, size, order date, etc.,

• It is used to store information about the orders made by users.

• The user Id field is a reference to the user who made the order.

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.,

• It is used to store information about the products added to the cart by users. • The user Id field is a reference to the user who has the product in cart.

5. Admin Schema:

• Schema: adminSchema

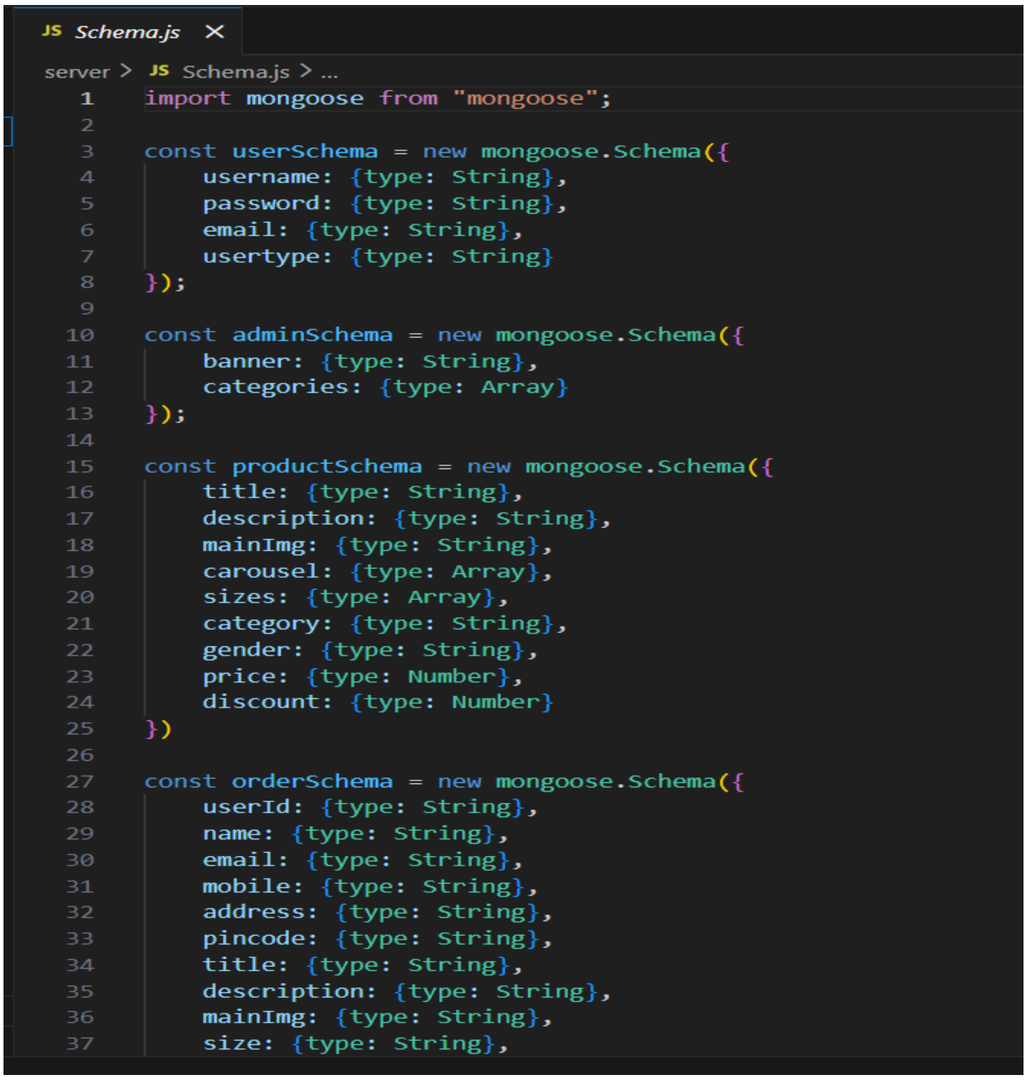
• Model: 'Admin'

• The admin schema has essential data such as categories, banner.

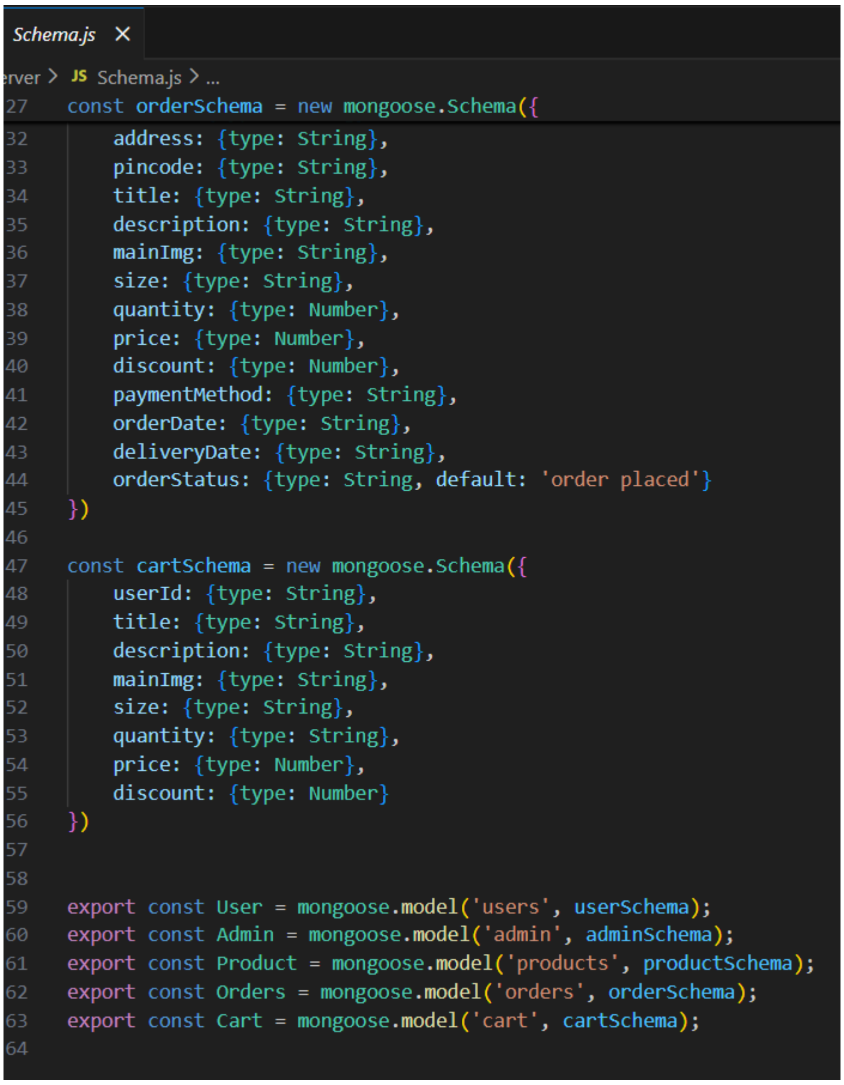
Code Explanation:

Schemas

Now let us define the required schemas



```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema({
    username: {type: String},
    password: {type: String},
    email: {type: String},
    usertype: {type: String}
});

const adminSchema = new mongoose.Schema({
    banner: {type: String},
    categories: {type: Array}
});

const productSchema = new mongoose.Schema({
    title: {type: String},
    description: {type: String},
    mainImg: {type: String},
    carousel: {type: Array},
    sizes: {type: Array},
    category: {type: String},
    gender: {type: String},
    price: {type: Number},
    discount: {type: Number}
})

const orderSchema = new mongoose.Schema({
    userId: {type: String},
    name: {type: String},
    email: {type: String},
    mobile: {type: String},
    address: {type: String},
    pincode: {type: String},
    title: {type: String},
    description: {type: String},
    mainImg: {type: String},
    size: {type: String},
```



```
const orderSchema = new mongoose.Schema({
    address: {type: String},
    pincode: {type: String},
    title: {type: String},
    description: {type: String},
    mainImg: {type: String},
    size: {type: String},
    quantity: {type: Number},
    price: {type: Number},
    discount: {type: Number},
    paymentMethod: {type: String},
    orderDate: {type: String},
    deliveryDate: {type: String},
    orderStatus: {type: String, default: 'order placed'}
})

const cartSchema = new mongoose.Schema({
    userId: {type: String},
    title: {type: String},
    description: {type: String},
    mainImg: {type: String},
    size: {type: String},
    quantity: {type: String},
    price: {type: Number},
    discount: {type: Number}
})


export const User = mongoose.model('users', userSchema);
export const Admin = mongoose.model('admin', adminSchema);
export const Product = mongoose.model('products', productSchema);
export const Orders = mongoose.model('orders', orderSchema);
export const Cart = mongoose.model('cart', cartSchema);
```

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.

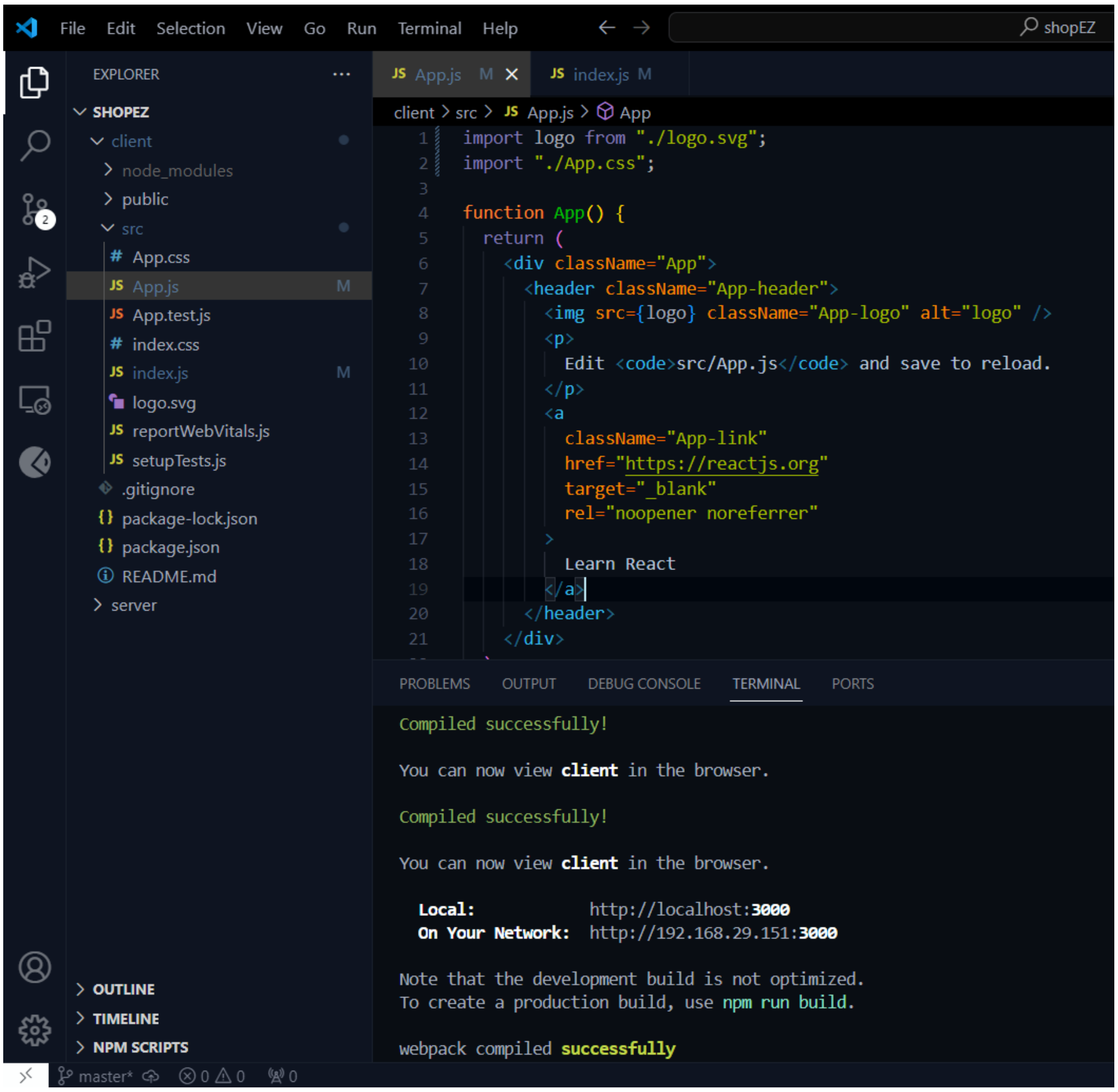
• Add navigation.

3.Implement frontend logic:

• Integration with API endpoints.

• Implement data binding.

Reference Image:



```
import logo from "./logo.svg";
import "./App.css";

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
```

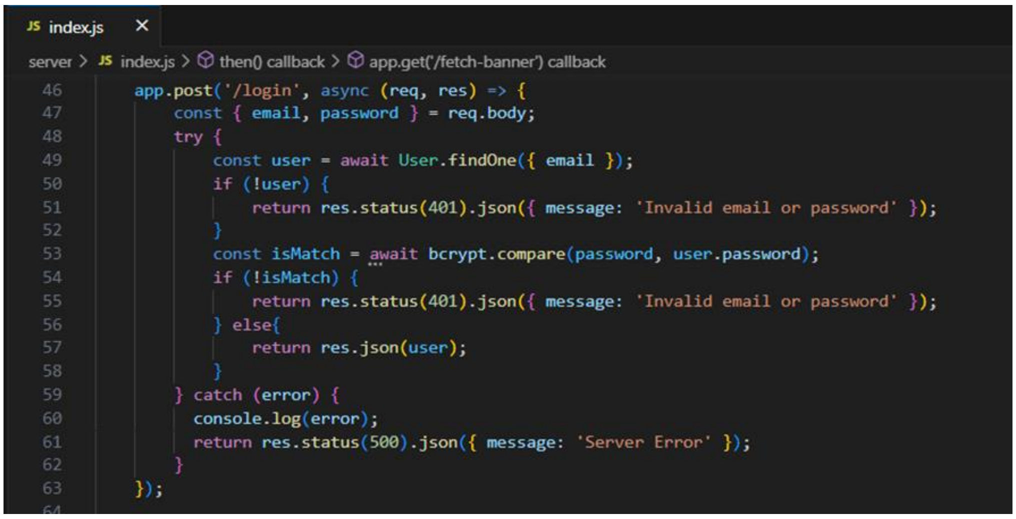
```
Compiled successfully!

You can now view client in the browser.

Compiled successfully!

You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.29.151:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Project Implementation & Execution

User Authentication:

- **Backend**

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



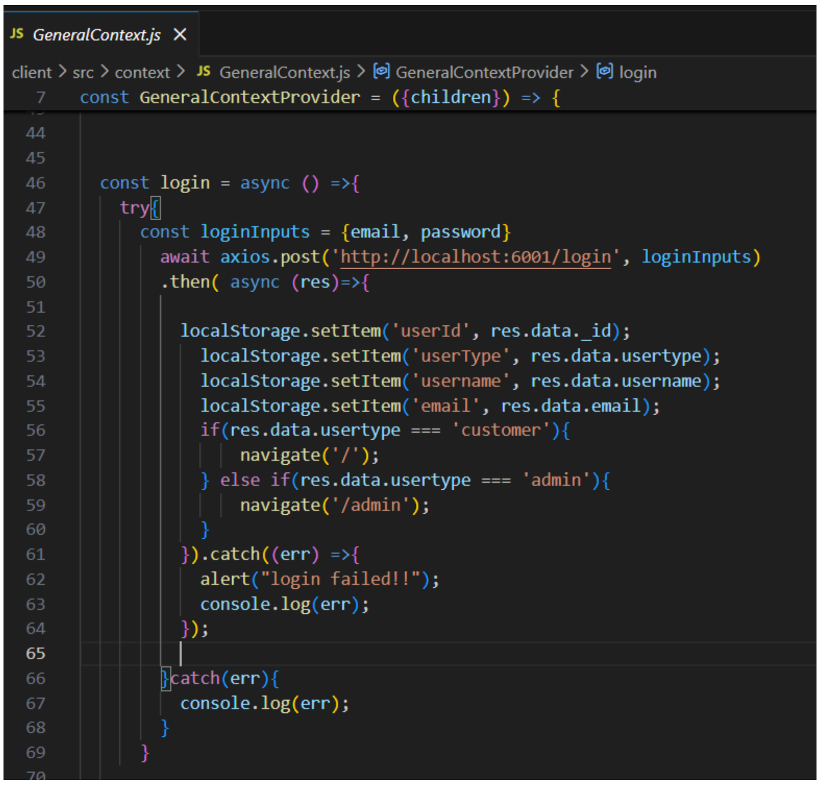
JS index.js ×

server > JS index.js > then() callback > app.get('/fetch-banner') callback

```
    app.post('/login', async (req, res) => {
        const { email, password } = req.body;
        try {
            const user = await User.findOne({ email });
            if (!user) {
                return res.status(401).json({ message: 'Invalid email or password' });
            }
            const isMatch = await bcrypt.compare(password, user.password);
            if (!isMatch) {
                return res.status(401).json({ message: 'Invalid email or password' });
            } else{
                return res.json(user);
            }
        } catch (error) {
          console.log(error);
          return res.status(500).json({ message: 'Server Error' });
        }
    });
```

- **Frontend**

Login:

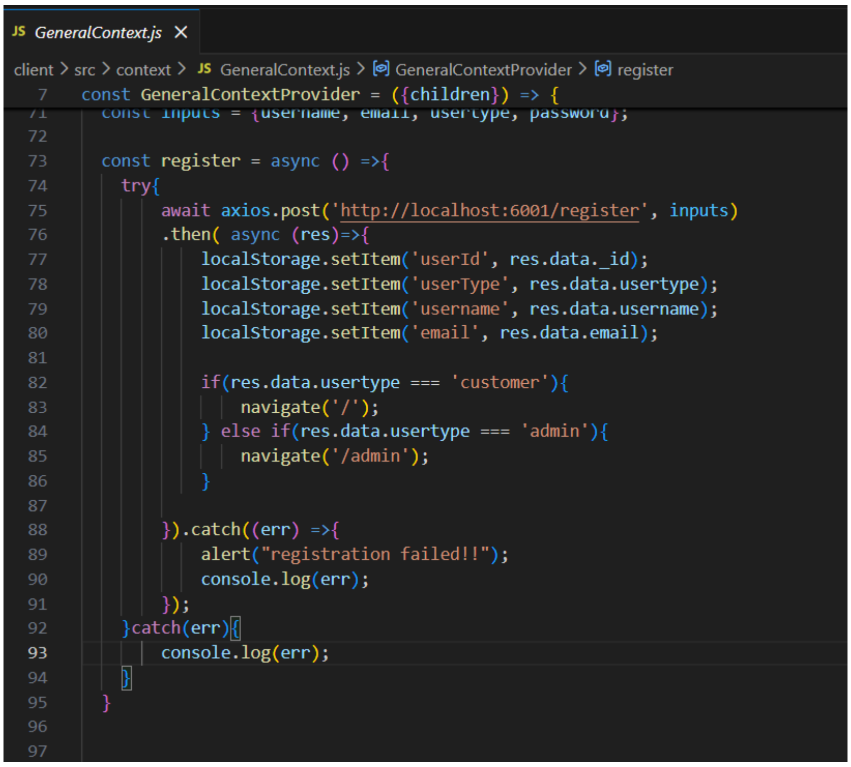


JS GeneralContext.js ×

client > src > context > JS GeneralContext.js > GeneralContextProvider > login

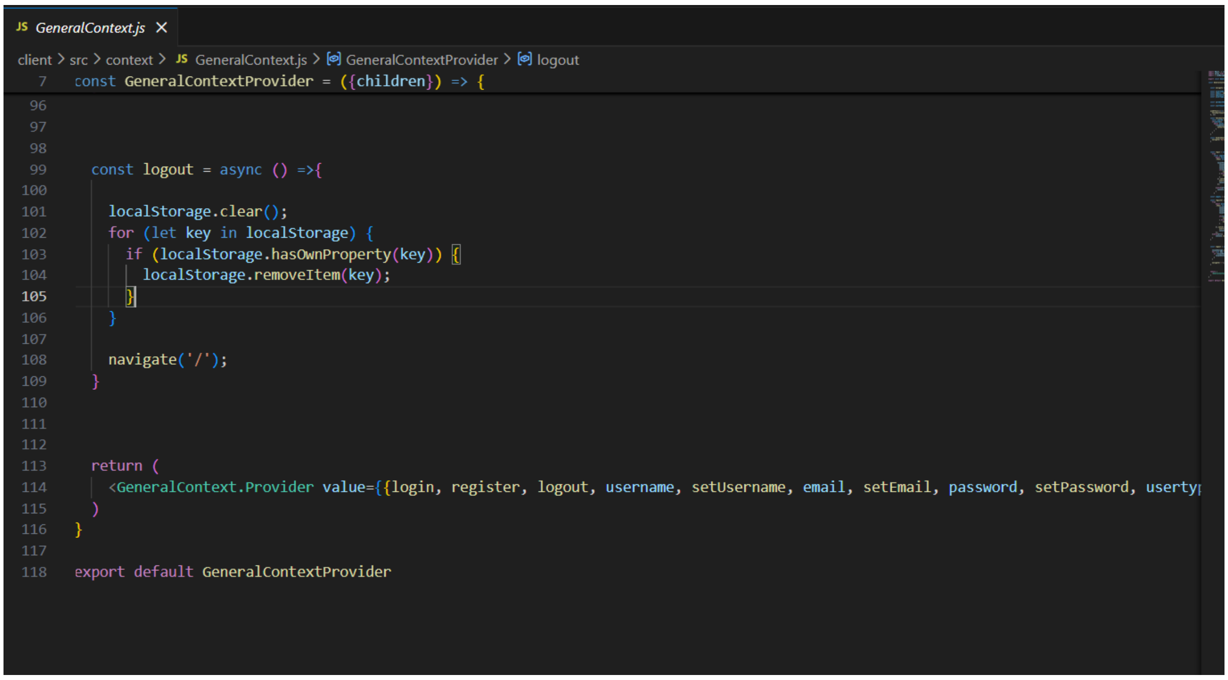
```
const GeneralContextProvider = ({children}) => {


  const login = async () =>{
    try{
      const loginInputs = {email, password}
        await axios.post('http://localhost:6001/login', loginInputs)
        .then( async (res)=>{

          localStorage.setItem('userId', res.data._id);
            localStorage.setItem('userType', res.data.usertype);
            localStorage.setItem('username', res.data.username);
            localStorage.setItem('email', res.data.email);
            if(res.data.usertype === 'customer'){
                navigate('/');
            } else if(res.data.usertype === 'admin'){
                navigate('/admin');
            }
          }).catch((err) =>{
            alert("login failed!!");
            console.log(err);
          });

        }catch(err){
          console.log(err);
        }
      }
```

Register:


GeneralContext.js
client > src > context > GeneralContext.js > GeneralContextProvider > register

```
const GeneralContextProvider = ({children}) => {
  const inputs = {username, email, usertype, password};

  const register = async () =>{
    try{
        await axios.post('http://localhost:6001/register', inputs)
        .then( async (res)=>{
            localStorage.setItem('userId', res.data._id);
            localStorage.setItem('userType', res.data.usertype);
            localStorage.setItem('username', res.data.username);
            localStorage.setItem('email', res.data.email);

            if(res.data.usertype === 'customer'){
                navigate('/');
            } else if(res.data.usertype === 'admin'){
                navigate('/admin');
            }

        }).catch((err) =>{
            alert("registration failed!!");
            console.log(err);
        });
    }catch(err){
        console.log(err);
    }
  }
```

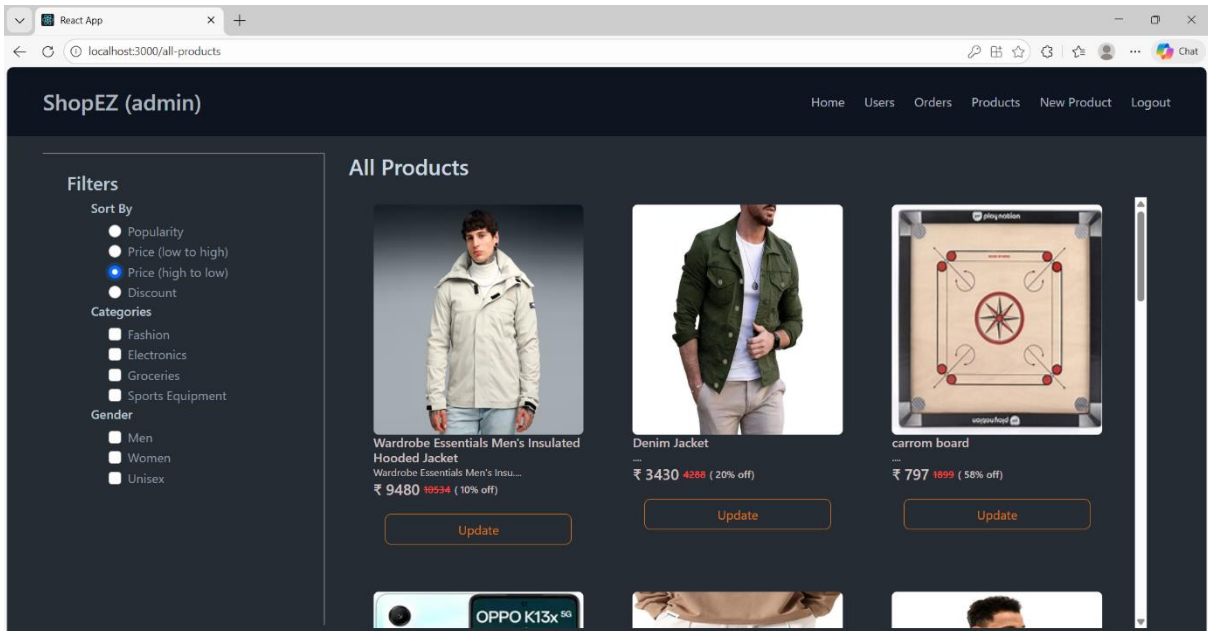
logout:


GeneralContext.js
client > src > context > GeneralContext.js > GeneralContextProvider > logout

```
const GeneralContextProvider = ({children}) => {



  const logout = async () =>{

    localStorage.clear();
    for (let key in localStorage) {
      if (localStorage.hasOwnProperty(key)) {
        localStorage.removeItem(key);
      }
    }

    navigate('/');
  }



  return (
    <GeneralContext.Provider value={{login, register, logout, username, setUsername, email, setEmail, password, setPassword, userty
  )
}

export default GeneralContextProvider
```

All Products (User):

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

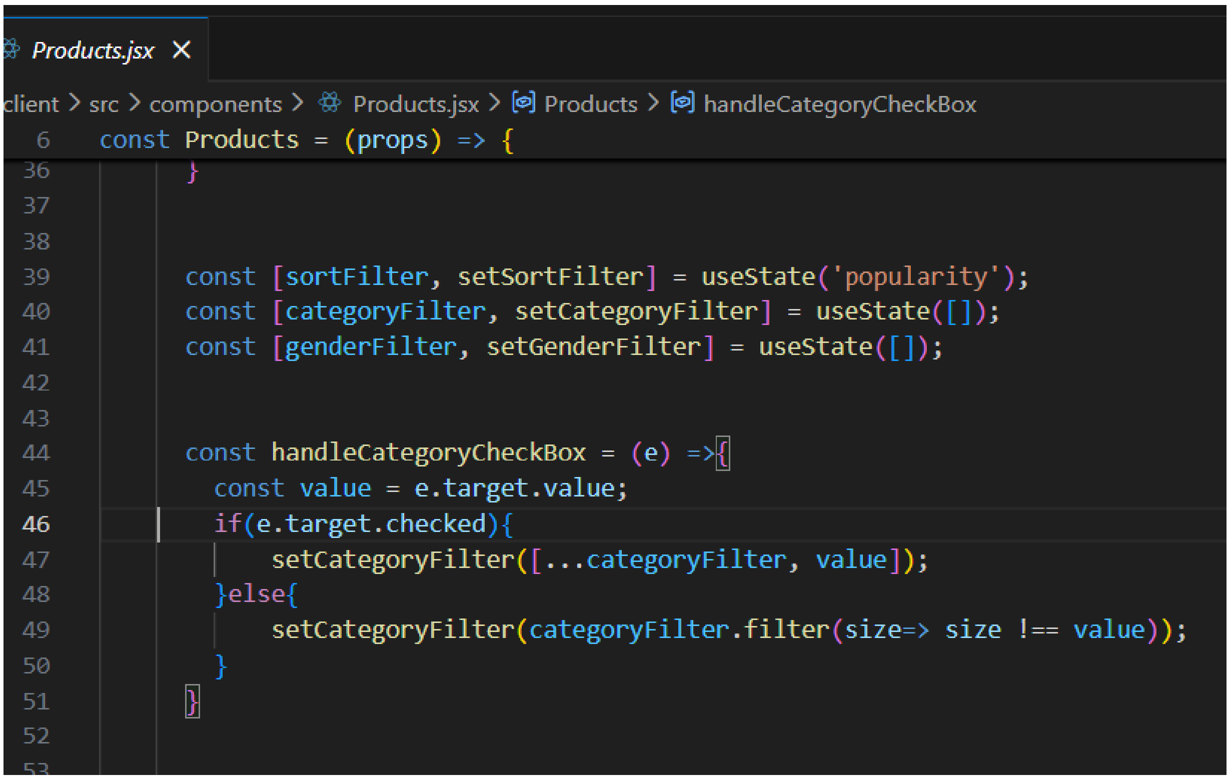


Wardrobe Essentials Men's Insulated Hooded Jacket
Wardrobe Essentials Men's Insu...
₹ 9480 ~~10534~~ (10% off)

Denim Jacket
...
₹ 3430 ~~4288~~ (20% off)

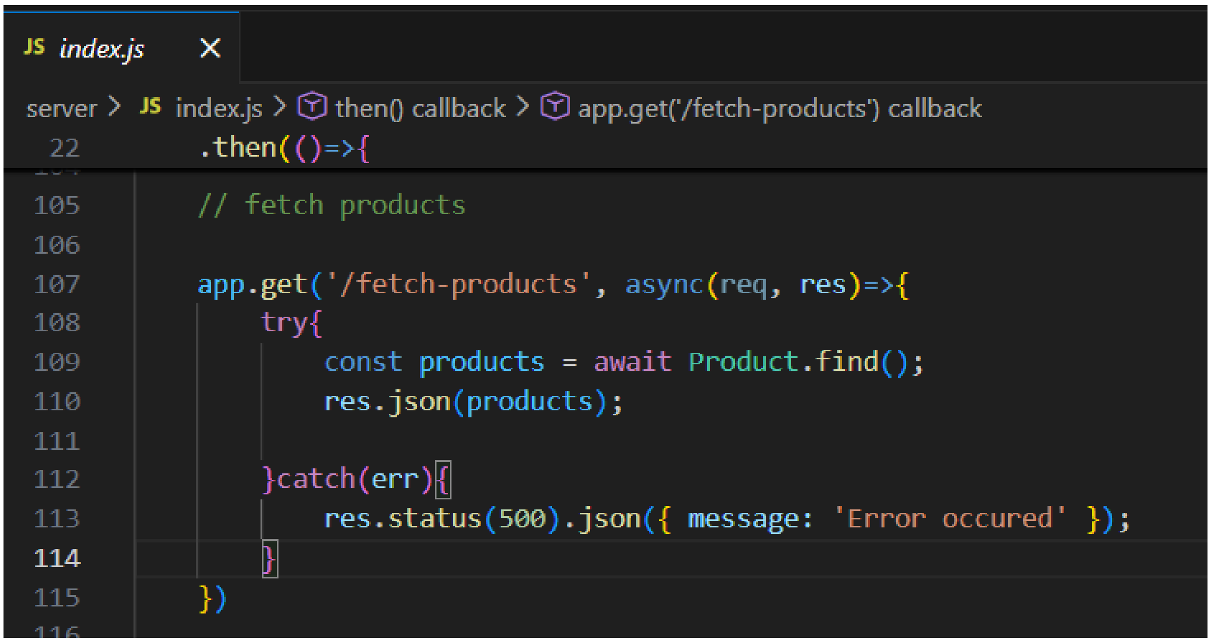
carrom board
...
₹ 797 ~~1899~~ (58% off)

Fetching products:



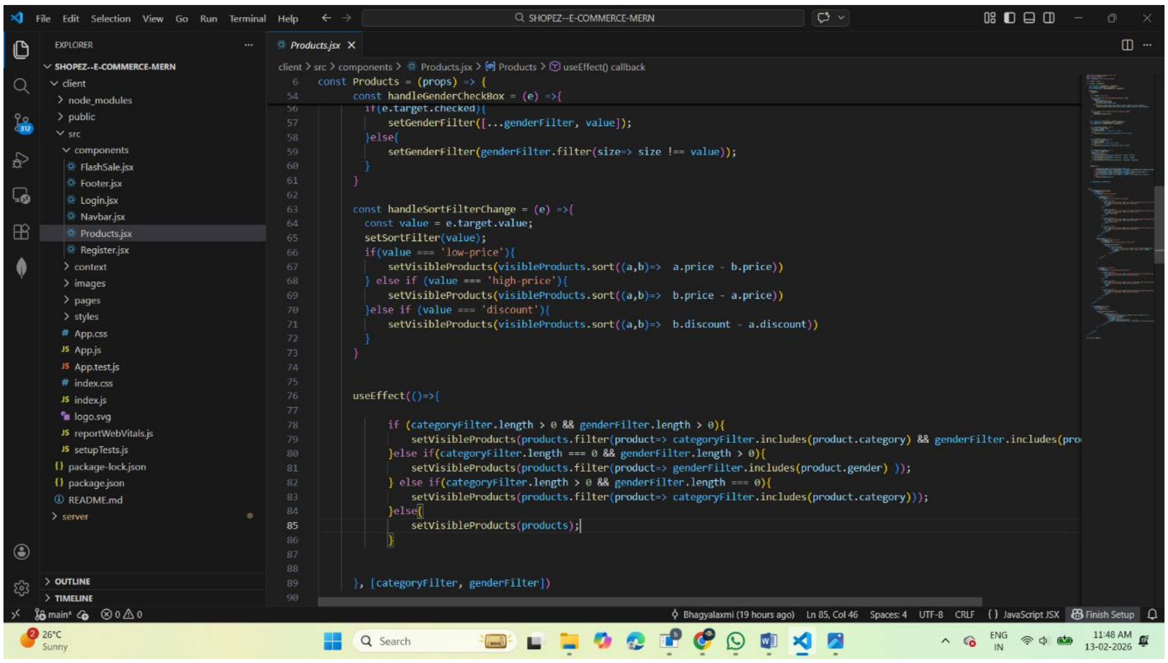
```
Products.jsx
client > src > components > Products.jsx > Products > handleCategoryCheckBox
const Products = (props) => {
        }


        const [sortFilter, setSortFilter] = useState('popularity');
        const [categoryFilter, setCategoryFilter] = useState([]);
        const [genderFilter, setGenderFilter] = useState([]);


        const handleCategoryCheckBox = (e) =>{
          const value = e.target.value;
          if(e.target.checked){
              setCategoryFilter([...categoryFilter, value]);
          }else{
              setCategoryFilter(categoryFilter.filter(size=> size !== value));
          }
        }
```

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


index.js
server > index.js > then() callback > app.get('/fetch-products') callback

```
    .then(()=>{

    // fetch products

    app.get('/fetch-products', async(req, res)=>{
        try{
            const products = await Product.find();
            res.json(products);

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

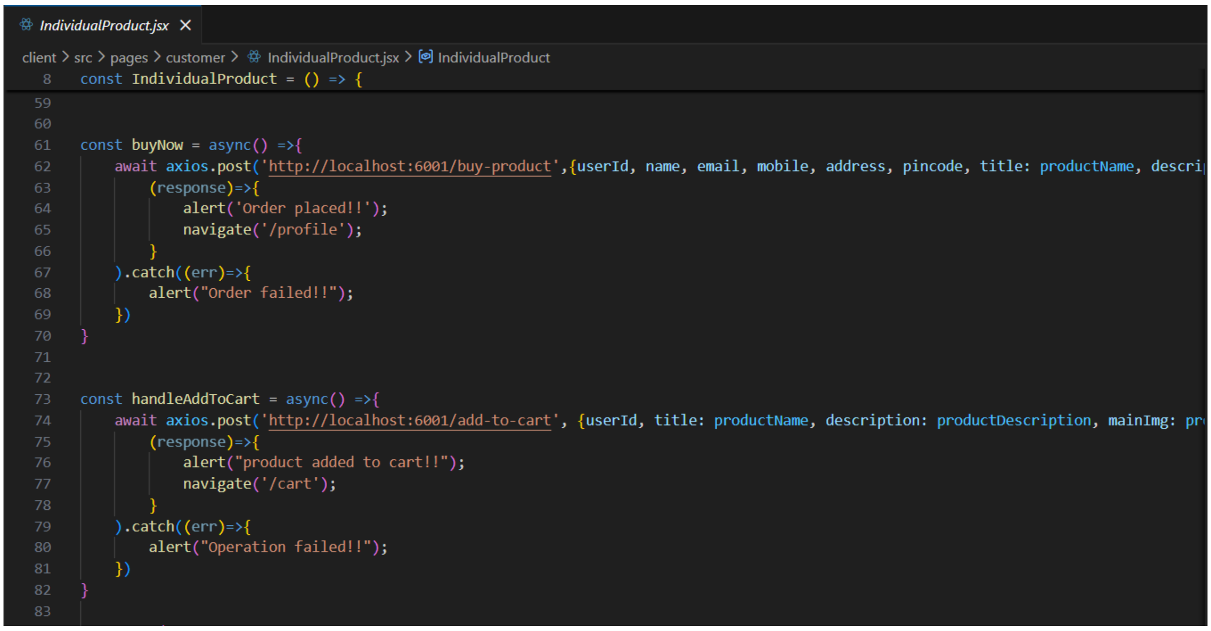
Filtering products:


SHOPEZ--E-COMMERCE-MERN
Products.jsx
client > src > components > Products.jsx > Products > useEffect() callback

```
const Products = (props) => {
    const handleGenderCheckBox = (e) =>{
      if(e.target.checked){
          setGenderFilter([...genderFilter, value]);
      }else{
          setGenderFilter(genderFilter.filter(size=> size !== value));
      }
    }

    const handleSortFilterChange = (e) =>{
      const value = e.target.value;
      setSortFilter(value);
      if(value === 'low-price'){
          setVisibleProducts(visibleProducts.sort((a,b)=>  a.price - b.price))
      } else if (value === 'high-price'){
          setVisibleProducts(visibleProducts.sort((a,b)=>  b.price - a.price))
      }else if (value === 'discount'){
          setVisibleProducts(visibleProducts.sort((a,b)=>  b.discount - a.discount))
      }
    }


    useEffect(()=>{

        if (categoryFilter.length > 0 && genderFilter.length > 0){
            setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category) && genderFilter.includes(pro
        }else if(categoryFilter.length === 0 && genderFilter.length > 0){
            setVisibleProducts(products.filter(product=> genderFilter.includes(product.gender) ));
        } else if(categoryFilter.length > 0 && genderFilter.length === 0){
            setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category)));
        }else{
            setVisibleProducts(products);
        }


    }, [categoryFilter, genderFilter])
```

Add product to cart:

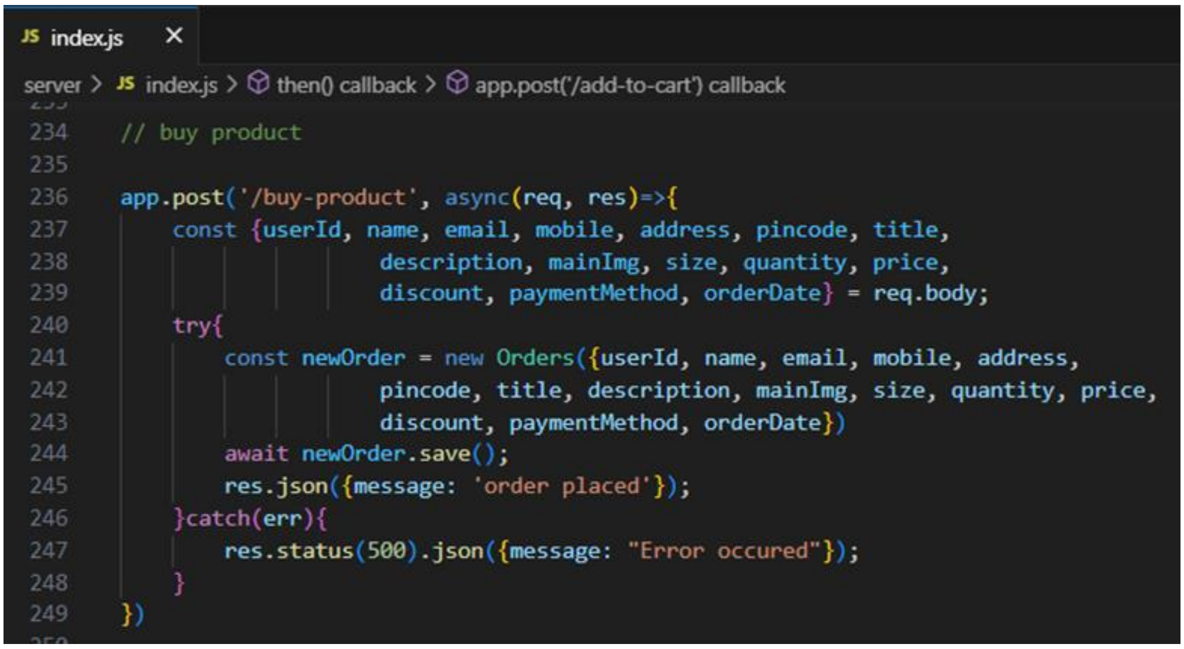
Here, we can add the product to the cart or can buy directly.


IndividualProduct.jsx
client > src > pages > customer > IndividualProduct.jsx > IndividualProduct

```jsx
const IndividualProduct = () => {


const buyNow = async() =>{
    await axios.post('http://localhost:6001/buy-product',{userId, name, email, mobile, address, pincode, title: productName, descri
        (response)=>{
            alert('Order placed!!');
            navigate('/profile');
        }
    ).catch((err)=>{
        alert("Order failed!!");
    })
}


const handleAddToCart = async() =>{
    await axios.post('http://localhost:6001/add-to-cart', {userId, title: productName, description: productDescription, mainImg: pr
        (response)=>{
            alert("product added to cart!!");
            navigate('/cart');
        }
    ).catch((err)=>{
        alert("Operation failed!!");
    })
}

```

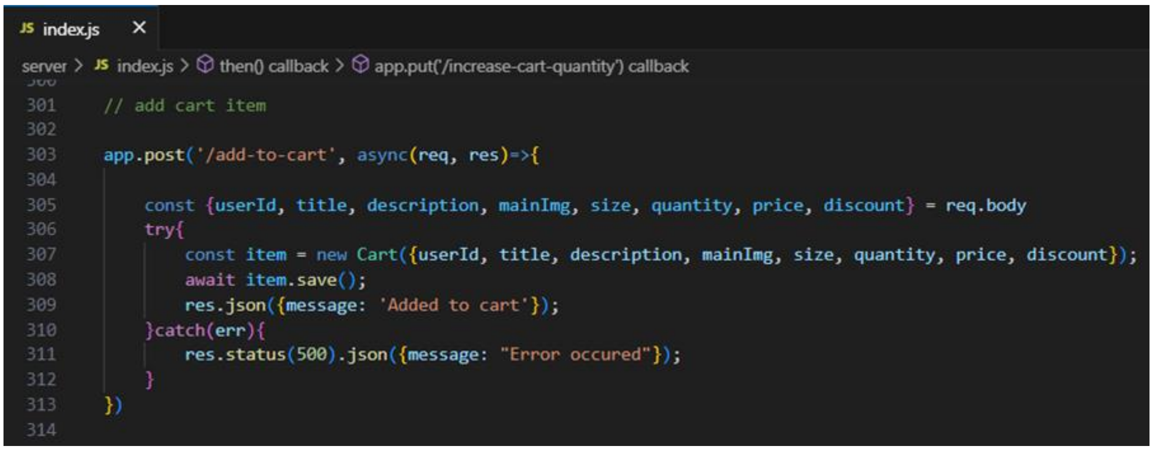
· Backend**:** In the backend, if we want to buy, then with the address and payment method, we process buying. If we need to add the product to the cart, then we add the product details along with the user Id to the cart collection.

Buy product:


index.js
server > index.js > then() callback > app.post('/add-to-cart') callback

```js
// buy product

app.post('/buy-product', async(req, res)=>{
    const {userId, name, email, mobile, address, pincode, title,
                description, mainImg, size, quantity, price,
                discount, paymentMethod, orderDate} = req.body;
    try{
        const newOrder = new Orders({userId, name, email, mobile, address,
                pincode, title, description, mainImg, size, quantity, price,
                discount, paymentMethod, orderDate})
        await newOrder.save();
        res.json({message: 'order placed'});
    }catch(err){
        res.status(500).json({message: "Error occured"});
    }
})
```

Add product to cart:


JS index.js ×

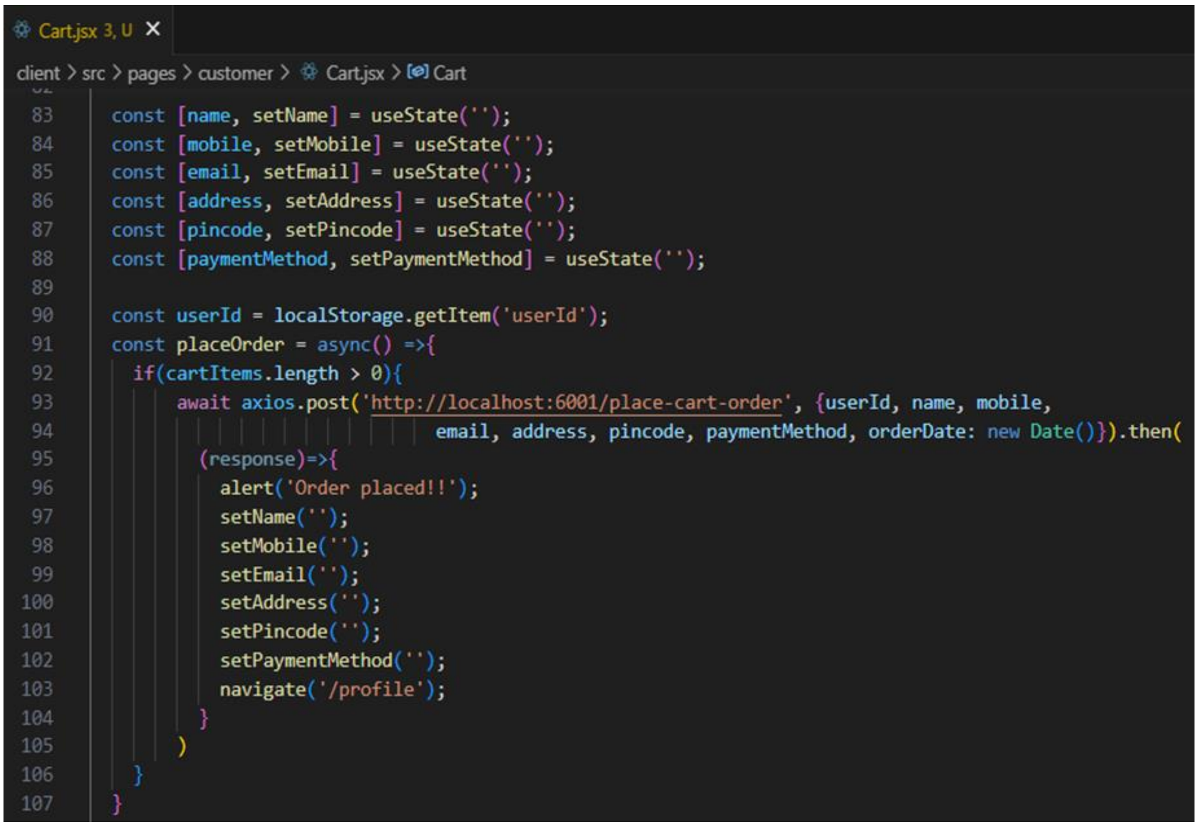
server > JS index.js > then() callback > app.put('/increase-cart-quantity') callback

```
// add cart item

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

    const {userId, title, description, mainImg, size, quantity, price, discount} = req.body
    try{
        const item = new Cart({userId, title, description, mainImg, size, quantity, price, discount});
        await item.save();
        res.json({message: 'Added to cart'});
    }catch(err){
        res.status(500).json({message: "Error occured"});
    }
})

```

Order products:

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

· Frontend

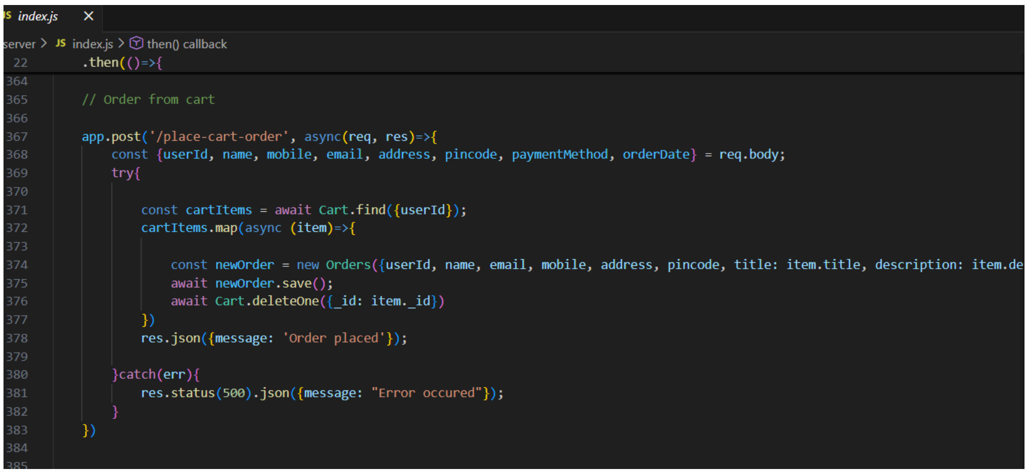

Cart.jsx 3, U ×

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

```
const [name, setName] = useState('');
const [mobile, setMobile] = useState('');
const [email, setEmail] = useState('');
const [address, setAddress] = useState('');
const [pincode, setPincode] = useState('');
const [paymentMethod, setPaymentMethod] = useState('');

const userId = localStorage.getItem('userId');
const placeOrder = async() =>{
  if(cartItems.length > 0){
      await axios.post('http://localhost:6001/place-cart-order', {userId, name, mobile,
                                    email, address, pincode, paymentMethod, orderDate: new Date()}).then(
        (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.



index.js

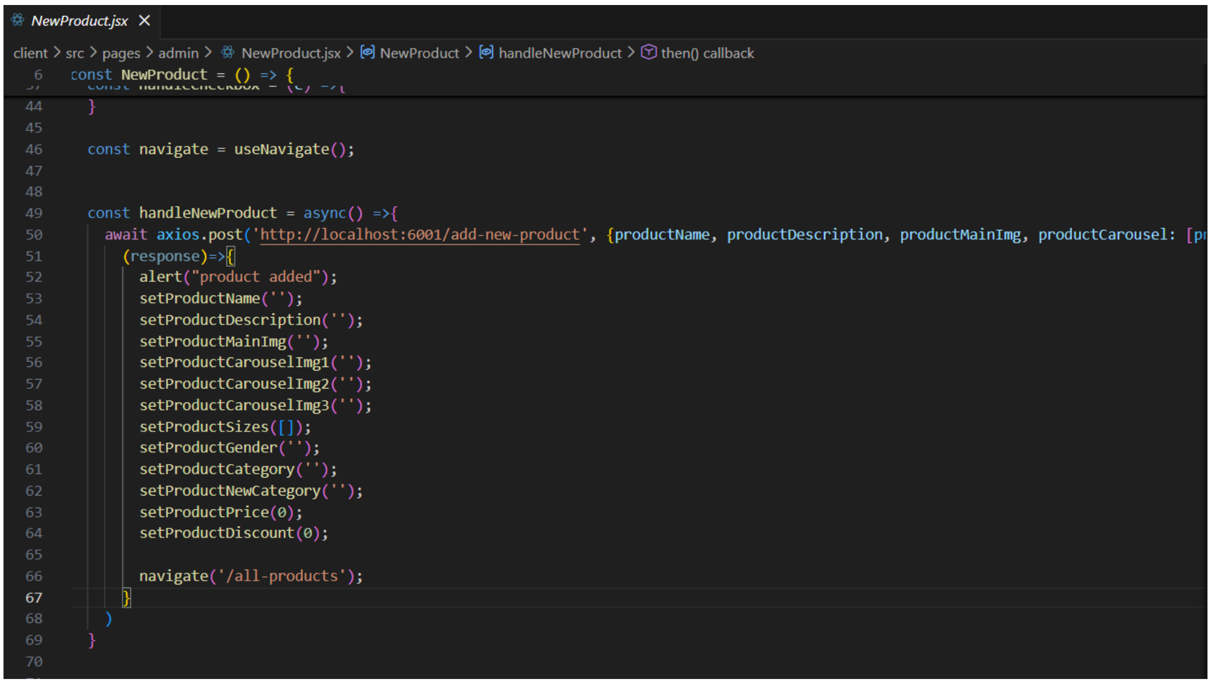
server > index.js > then() callback

```
      .then(()=>{

    // Order from cart

    app.post('/place-cart-order', async(req, res)=>{
        const {userId, name, mobile, email, address, pincode, paymentMethod, orderDate} = req.body;
        try{

            const cartItems = await Cart.find({userId});
            cartItems.map(async (item)=>{

                const newOrder = new Orders({userId, name, email, mobile, address, pincode, title: item.title, description: item.de
                await newOrder.save();
                await Cart.deleteOne({_id: item._id})
            })
            res.json({message: 'Order placed'});

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

Add new product:

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

o　Frontend:

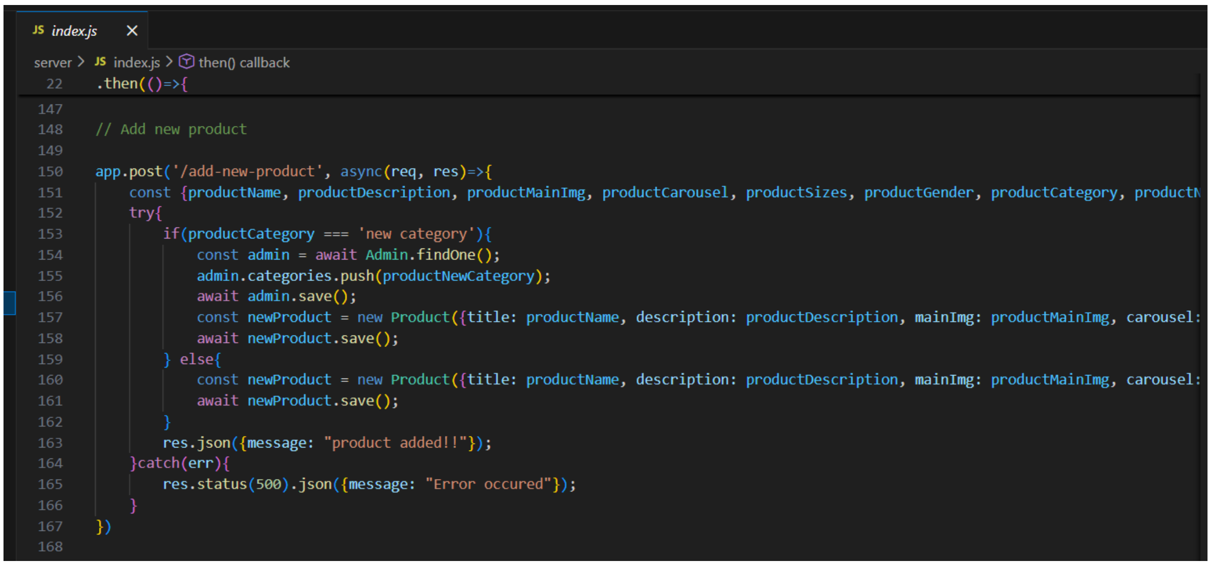


NewProduct.jsx

client > src > pages > admin > NewProduct.jsx > NewProduct > handleNewProduct > then() callback

```
const NewProduct = () => {
    }

    const navigate = useNavigate();


    const handleNewProduct = async() =>{
      await axios.post('http://localhost:6001/add-new-product', {productName, productDescription, productMainImg, productCarousel: [pr
        (response)=>{
          alert("product added");
          setProductName('');
          setProductDescription('');
          setProductMainImg('');
          setProductCarouselImg1('');
          setProductCarouselImg2('');
          setProductCarouselImg3('');
          setProductSizes([]);
          setProductGender('');
          setProductCategory('');
          setProductNewCategory('');
          setProductPrice(0);
          setProductDiscount(0);

          navigate('/all-products');
        }
      )
    }
```

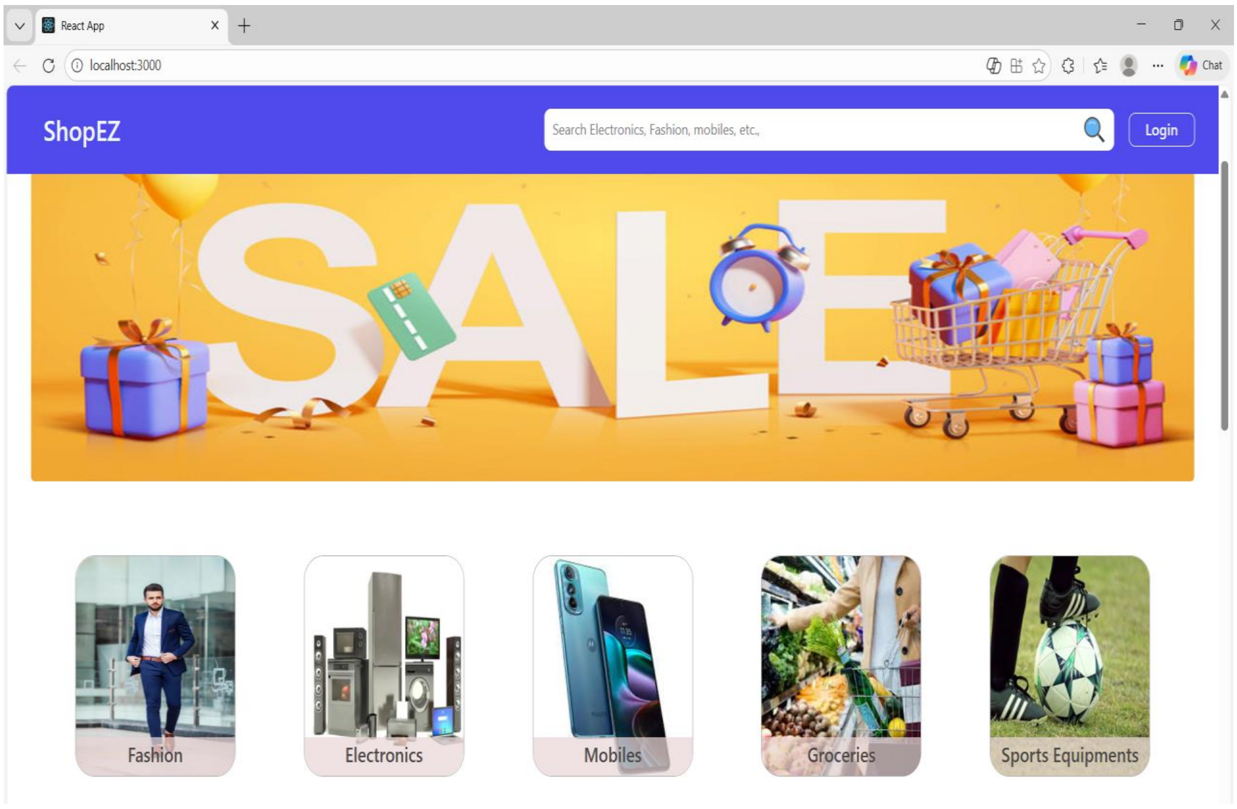
o Backend:



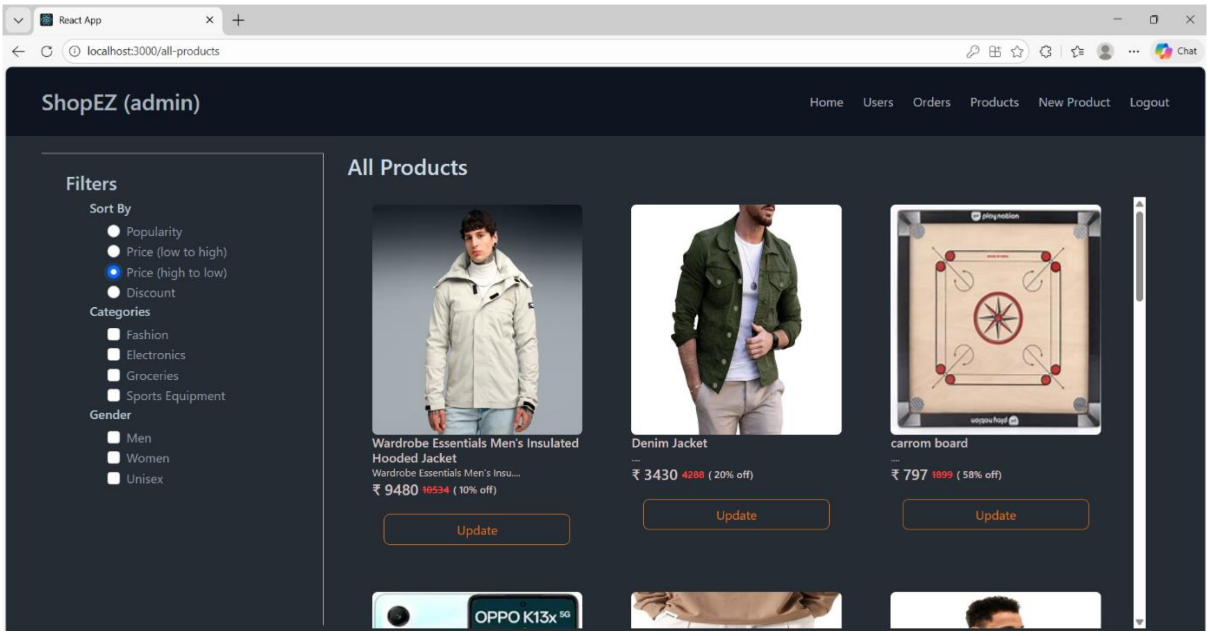
```
index.js
server > index.js > then() callback
.then(()=>{

// Add new product

app.post('/add-new-product', async(req, res)=>{
    const {productName, productDescription, productMainImg, productCarousel, productSizes, productGender, productCategory, productN
    try{
        if(productCategory === 'new category'){
            const admin = await Admin.findOne();
            admin.categories.push(productNewCategory);
            await admin.save();
            const newProduct = new Product({title: productName, description: productDescription, mainImg: productMainImg, carousel:
            await newProduct.save();
        } else{
            const newProduct = new Product({title: productName, description: productDescription, mainImg: productMainImg, carousel:
            await newProduct.save();
        }
        res.json({message: "product added!!"});
    }catch(err){
        res.status(500).json({message: "Error occured"});
    }
})
```

Demo UI images:

· **Landing page**



· **Products**


React App
localhost:3000/all-products

ShopEZ (admin)

Home Users Orders Products New Product Logout

Filters

Sort By
- Popularity
- Price (low to high)
- Price (high to low)
- Discount

Categories
- Fashion
- Electronics
- Groceries
- Sports Equipment

Gender
- Men
- Women
- Unisex

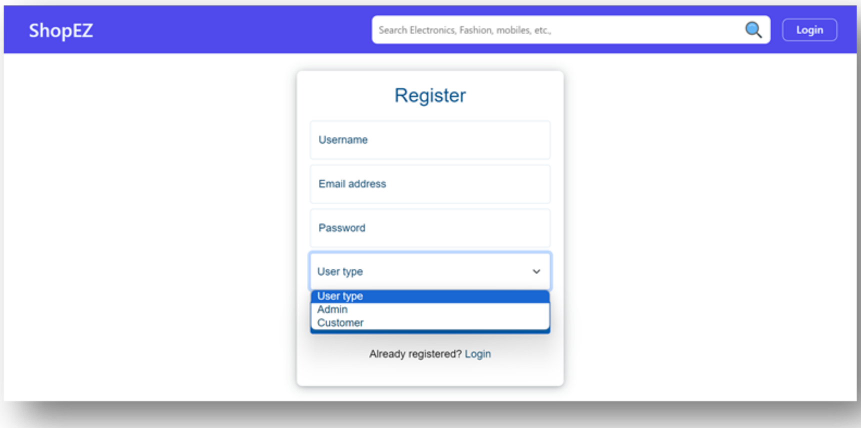
All Products

Wardrobe Essentials Men's Insulated Hooded Jacket
Wardrobe Essentials Men's Insu...
₹ 9480 ~~10534~~ (10% off)
Update

Denim Jacket
...
₹ 3430 ~~4288~~ (20% off)
Update

carrom board
...
₹ 797 ~~1899~~ (58% off)
Update

· **Authentication**

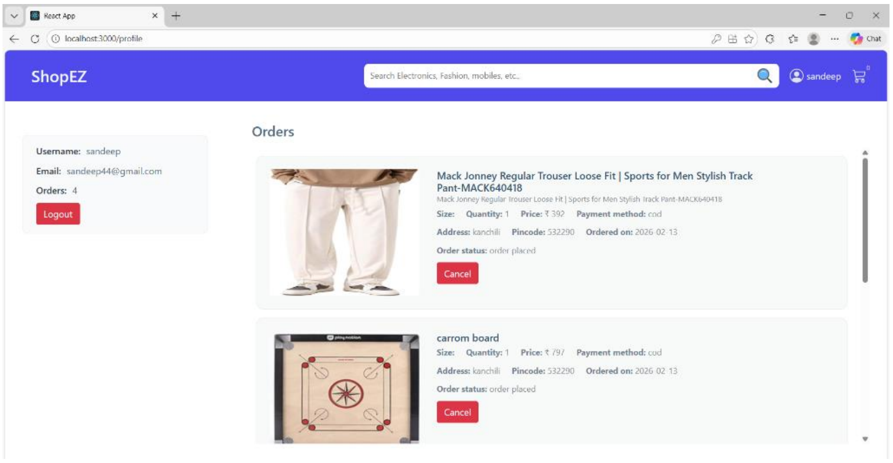

ShopEZ
Search Electronics, Fashion, mobiles, etc.,
Login

Register

Username
Email address
Password
User type
- User type
- Admin
- Customer

Already registered? Login

· **User Profile**


React App
localhost:3000/profile

ShopEZ
Search Electronics, Fashion, mobiles, etc.,
sandeep

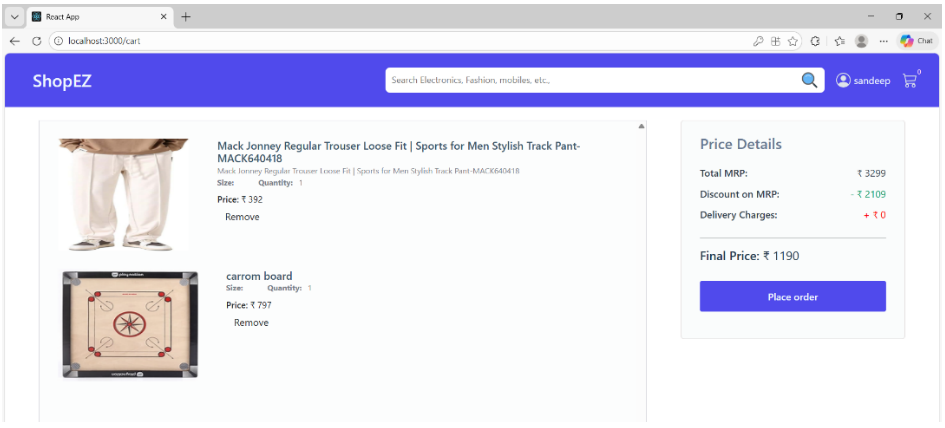
Username: sandeep
Email: sandeep44@gmail.com
Orders: 4
Logout

Orders

Mack Jonney Regular Trouser Loose Fit | Sports for Men Stylish Track Pant-MACK640418
Mack Jonney Regular Trouser Loose Fit | Sports for Men Stylish Track Pant-MACK640418
Size: **Quantity:** 1 **Price:** ₹ 392 **Payment method:** cod
Address: kanchili **Pincode:** 532290 **Ordered on:** 2026-02-13
Order status: order placed
Cancel

carrom board
Size: **Quantity:** 1 **Price:** ₹ 797 **Payment method:** cod
Address: kanchili **Pincode:** 532290 **Ordered on:** 2026-02-13
Order status: order placed
Cancel

- **Cart**



localhost:3000/cart

ShopEZ

Mack Jonney Regular Trouser Loose Fit | Sports for Men Stylish Track Pant-MACK640418
Mack Jonney Regular Trouser Loose Fit | Sports for Men Stylish Track Pant-MACK640418
Size: Quantity: 1
Price: ₹ 392

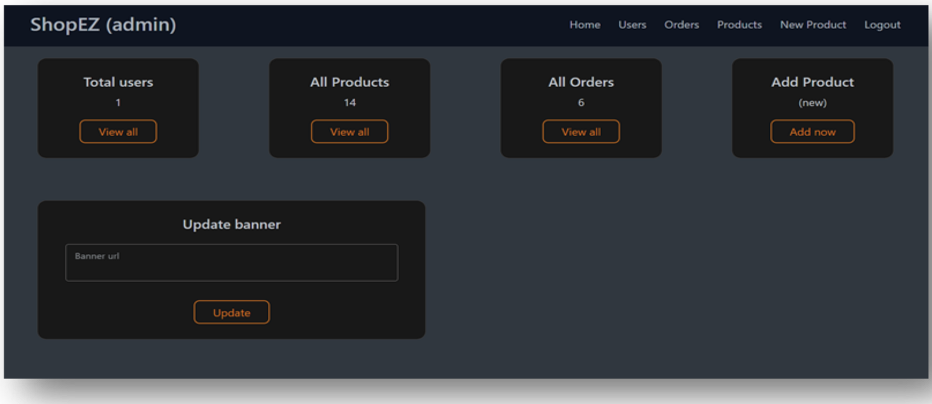
carrom board
Size: Quantity: 1
Price: ₹ 797

Price Details

Total MRP: ₹ 3299
Discount on MRP: - ₹ 2109
Delivery Charges: + ₹ 0

Final Price: ₹ 1190

- **Admin dashboard**

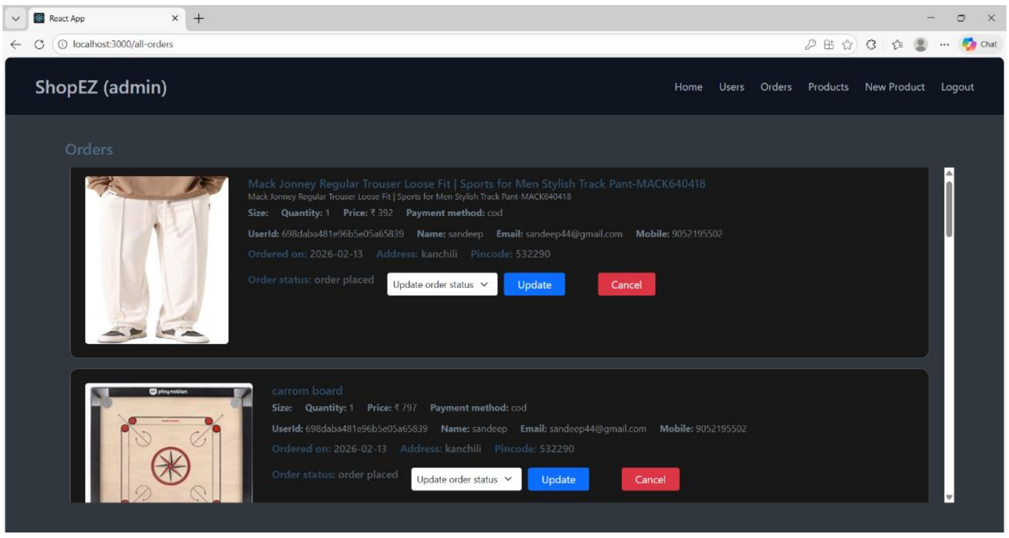


ShopEZ (admin)

Total users 1
All Products 14
All Orders 6
Add Product (new)

Update banner

- **All Orders**



localhost:3000/all-orders

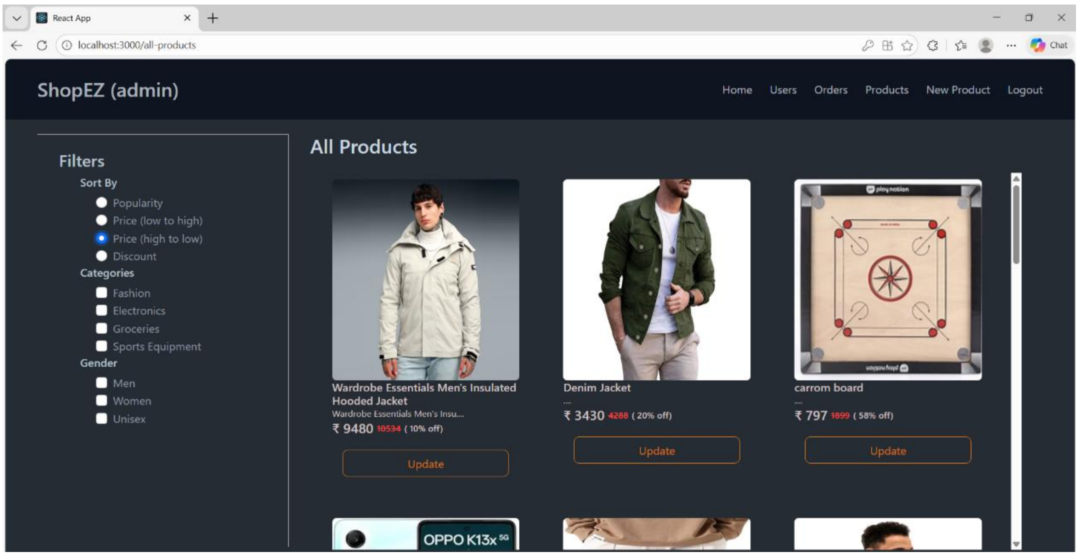
ShopEZ (admin)

Orders

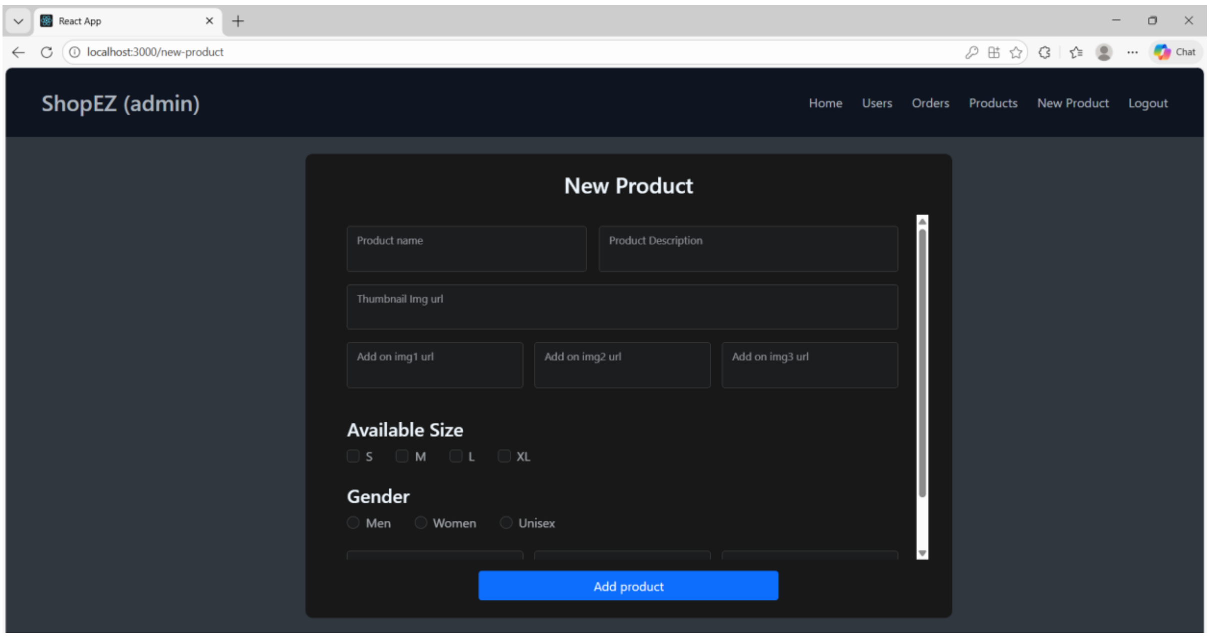
Mack Jonney Regular Trouser Loose Fit | Sports for Men Stylish Track Pant-MACK640418
Mack Jonney Regular Trouser Loose Fit | Sports for Men Stylish Track Pant-MACK640418
Size: Quantity: 1 Price: ₹ 392 Payment method: cod
UserId: 698daba481e96b5e05a65839 Name: sandeep Email: sandeep44@gmail.com Mobile: 9052195502
Ordered on: 2026-02-13 Address: kanchili Pincode: 532290
Order status: order placed

carrom board
Size: Quantity: 1 Price: ₹ 797 Payment method: cod
UserId: 698daba481e96b5e05a65839 Name: sandeep Email: sandeep44@gmail.com Mobile: 9052195502
Ordered on: 2026-02-13 Address: kanchili Pincode: 532290
Order status: order placed

· **All Products**



· **New Product Page**



The demo of the app is available at:

https://drive.google.com/drive/folders/1_6byLUBVSLIFut5Kv8ozuCoff3TEr7u1?usp=sharing