SHOPEZ:ONE STOP SHOP FOR ONLINE PURCHASES

INTRODUCTION:

ShopEZ is a modern, full-stack e-commerce web application build using the MERN stack (Mongodb, Express.js, React.js and Node.js). Designed to provide a seamless online shopping experience, ShopEZ allows users to browse a wide variety of products, make secure purchases, and manage their profiles. For sellers, ShopEZ provides a robust dashboard to manage products, orders and analyze business performance.

KEY FEATURES:

- **User Features:**
  - Effortless Product Discovery: Filter and search through a comprehensive product catalog by category, price, and more.

- Personalized Recommendations: Suggests products based on user behavior and preferences.
- User Authentication: Register, login, and manage secure user accounts using JWT-based authentication.
- Profile Management: Users can view and update their profile, order history, and preferences.
- Shopping Cart: Add, update, or remove items from a persistent shopping cart.
- Secure Checkout: Quick and reliable checkout process with real-time order confirmation.
- Order Tracking: View current and past orders with real-time status updates.

- **Admin/Seller Features:**
  - Admin Dashboard: A dedicated dashboard for sellers/admins to view key metrics.
  - Product Management: Add, edit, or remove products with detailed descriptions, images, and pricing.

    - Order Management: Process and manage incoming orders efficiently.
    - User Management: View and manage registered users.
- **Technical Features:**
    - MERN Stack Architecture: React frontend, Express.js/Node.js backend, MongoDB for data storage.
    - Context API: Used for global state management (e.g., Auth & Cart).
    - Modular Codebase: Clean and scalable folder structure
    - Responsive Design: Mobile-friendly layout using CSS.
    - API Integration: RESTful API endpoints for all core operations.
    - Secure Authentication: JWT and password hashing using bcrypt.
    - Performance Monitoring: Using tools like report web vitals.

PROJECT DESCRIPTION:

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 jewellery . 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.

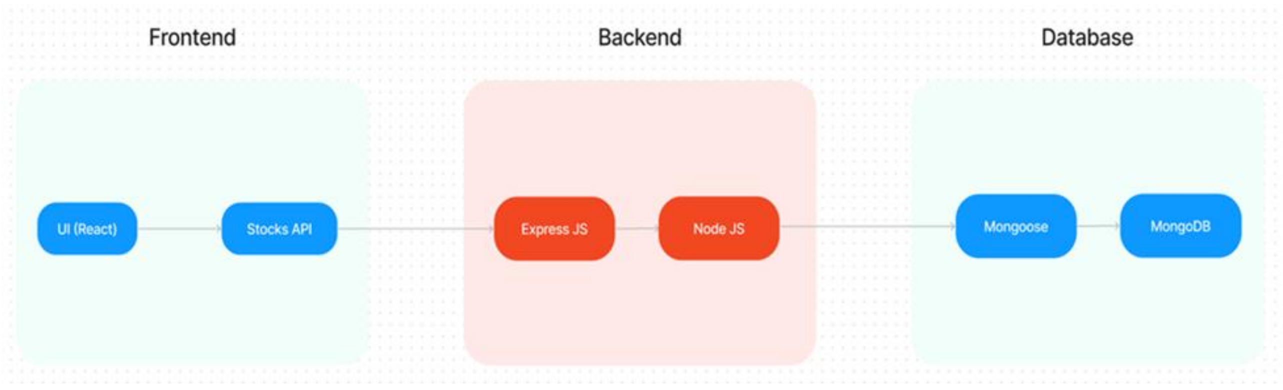
5. Efficient Order Management for Sellers: Meanwhile, on the other end, the seller of the gold bangle receives a notification of Sarah's purchase through ShopEZ's seller dashboard. They quickly process the order and prepare it for shipment, confident in ShopEZ's streamlined order management system.

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.

TECH STACK USED IN SHOPEZ PROJECT:

- **Frontend (Client-Side)**

  Built using React.js, the frontend is responsible for the user interface and interactions.

- **Technologies & Frameworks:**

  React.js – For building interactive UI components.

React Router DOM – For client-side routing.

Context API – For global state management (auth, cart).

CSS Modules / Plain CSS – For styling components.

Axios or Fetch API – For HTTP requests to the backend.

Jest / React Testing Library (optional) – For component testing.

- **Backend (Server-Side)**

  Built using Node.js and Express.js, the backend handles all business logic, API routing, and server-side operations.

- **Technologies & Frameworks:**

  Node.js – JavaScript runtime for server-side programming.

  Express.js – Web framework for building

RESTful APIs.
JWT (JSON Web Tokens) – For secure user authentication.
bcrypt.js – For password hashing.
Multer (optional) – For handling image uploads.
CORS – To handle cross-origin request.

- **Database (Data Layer)**

MongoDB is used as the NoSQL database to store and retrieve application data.

- **Technologies & Tools:**

MongoDB – Document-based NoSQL database.
Mongoose – ODM (Object Data Modeling) library for MongoDB, used to define schemas and interact with the database.

- **Other Tools & Utilities**

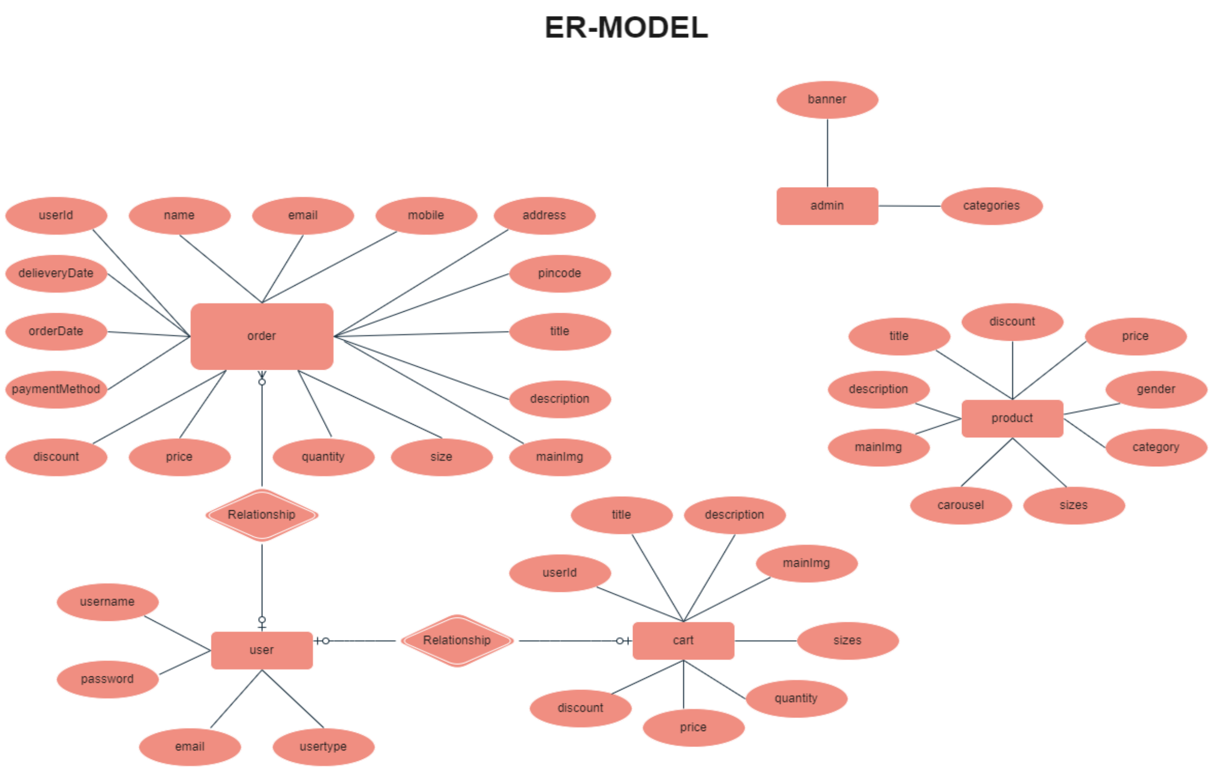
VS Code – Development environment.

Postman – API testing.

Nodemon – For live-reloading the backend during development.

Git + GitHub – Version control and code hosting.

ER DIAGRAMS:



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:

1. Technical Knowledge

Before working on this project, you should be familiar with:

- JavaScript (ES6+) – Core language for both frontend and backend.
- React.js- Component-based frontend library.
- Node.js – JavaScript runtime for backend development.
- Express.js – Framework for building RESTful APIs.
- MongoDB & Mongoose – For working with NoSQL databases and ODM.
- REST API Concepts – Understanding of HTTP methods (GET, POST, PUT, DELETE).

- Git & GitHub – For version control and project collaboration.

2.Software & Tools

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:

DriveLink:

https://drive.google.com/drive/folders/1QnZb2_S3rrupyv1hm8Kok2FKBqTwTebc?usp=drive_link

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.

3.Environment Setup

Set up a .env file in your server/ directory for:

```
MONGO_URI=your_mongo_connection_string
JWT_SECRET= your_jwt_secret
PORT=5000
```

Make sure MongoDB is running locally or provide an Atlas connection.

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.,

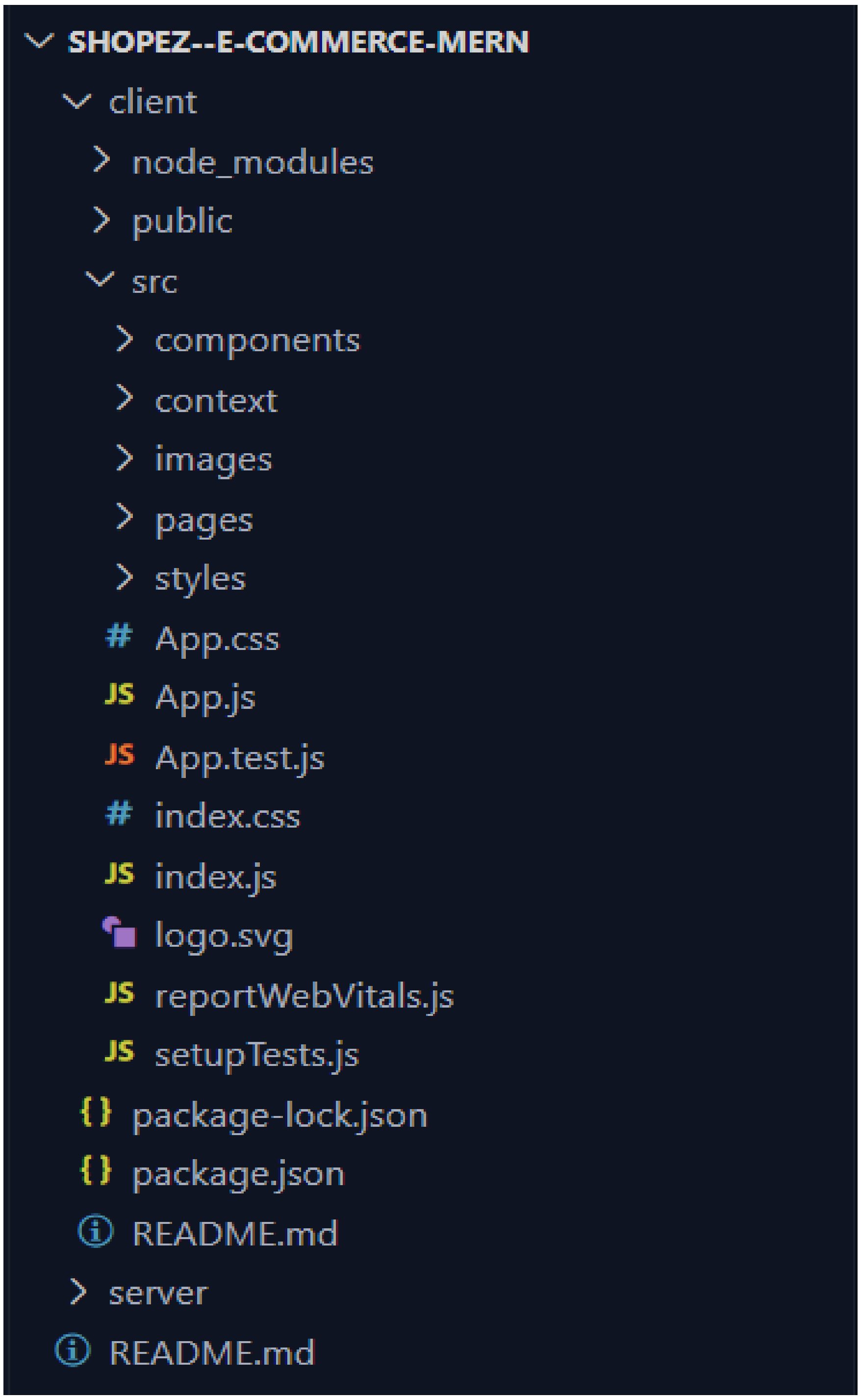
PROJECT STRUCTURE:

This structure assumes a React app and follows a modular approach.

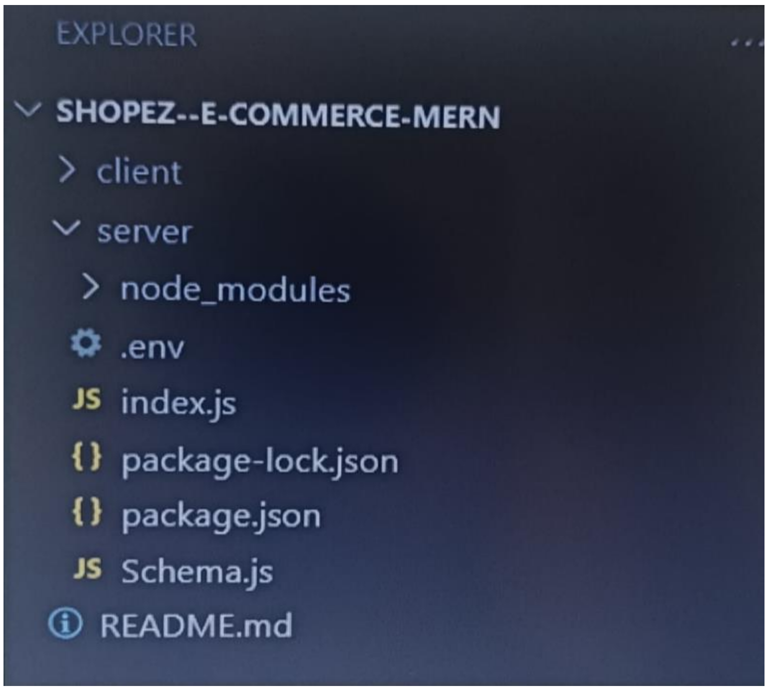
Here is a detailed explanation of the ShopEZ project structure based on the MERN stack:

```
shopez-ecommerce-mern/
├── client/
│   ├── node_modules/
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   │   └── ProductCard.js
│   │   ├── context/
│   │   │   └── ShopContext.js
│   │   ├── images/
│   │   ├── pages/
│   │   │   ├── HomePage.js
│   │   │   ├── ProductPage.js
│   │   │   └── CheckoutPage.js
│   │   ├── styles/
│   │   │   └── main.css
│   │   ├── 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/
│   └── (backend code here)
```

```
└── README.md
```



```
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
  v server
    > node_modules
      .env
      index.js
      package-lock.json
      package.json
      Schema.js
    README.md
```

PROJECT SETUP & CONFIGURATION:

Project Setup And Configuration

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- Node.js

  Reference Article:

  geeksforgeeks.org/installation-guide/install-node-js-on-windows/

- Git

  Reference Article:

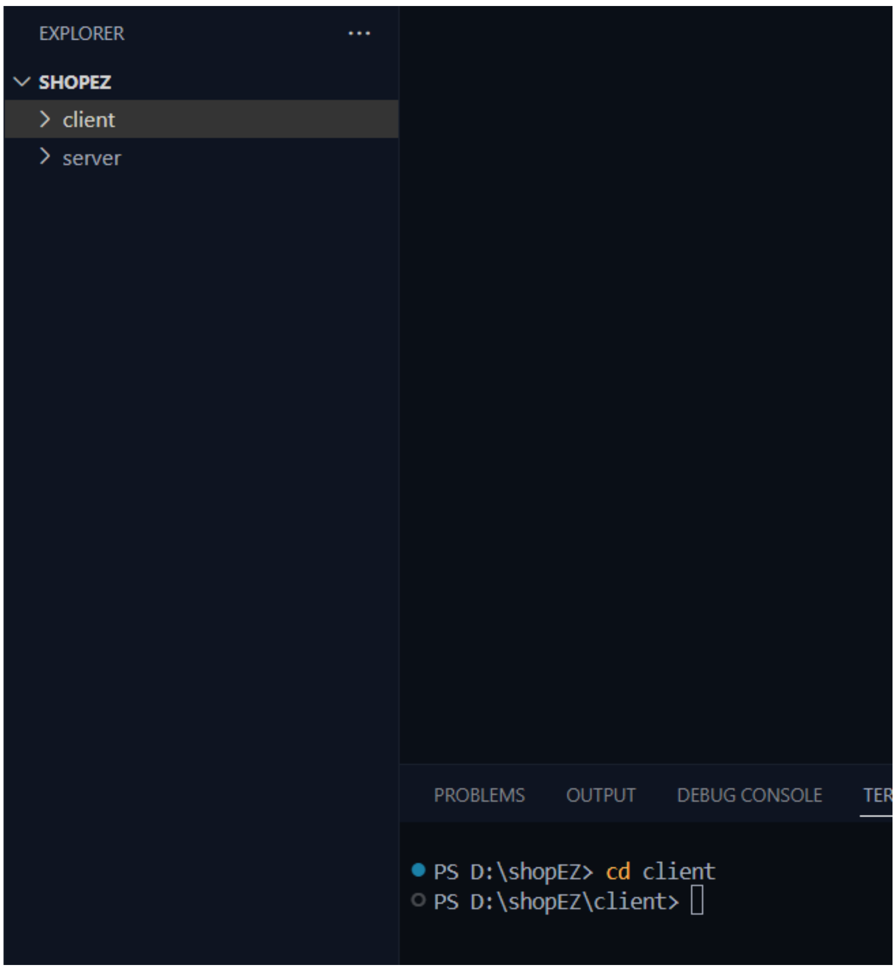
  [Git - Installing Git]()

2. Create project folders and files:

• Client folders.

• Server folders

Referral Image:



```
PS D:\shopEZ> cd client
PS D:\shopEZ\client>
```

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.

Tools & Libraries Used

Tool/Library	Purpose
React.js	Build reusable UI components
React Router DOM	Handle reusable between pages
Context API	Manage global states
Axios/Fetch	Make HTTP requests to the backend
CSS Modules/CSS	Style components responsively

React Icons	Add icons for UI enhancement

Key Features Implemented:

- **User Interface**

Responsive Layout: Works well on mobile and desktop.

Navigation Bar: Includes links to Home, Cart, Profile, etc.

Dynamic Pages: React Router enables route-based views (e.g., /product/:id).

- **Product Browsing**

Product Cards: Display products with name, price, image.

Filtering/Search (optional): Filter by category, price, etc.

- **Authentication**

Login/Register Pages: Allow user to sign in or register.

Auth Context: Stores and shares login state across components.

- **Cart Functionality**

Add to Cart: From product page or home.

Update Quantity / Remove Items: In the Cart page.

Cart Context: Maintains cart state globally**.**

- **Checkout Workflow**

Shipping Info Input

Order Summary

Place Order Button (calls backend API)

Order Confirmation Page

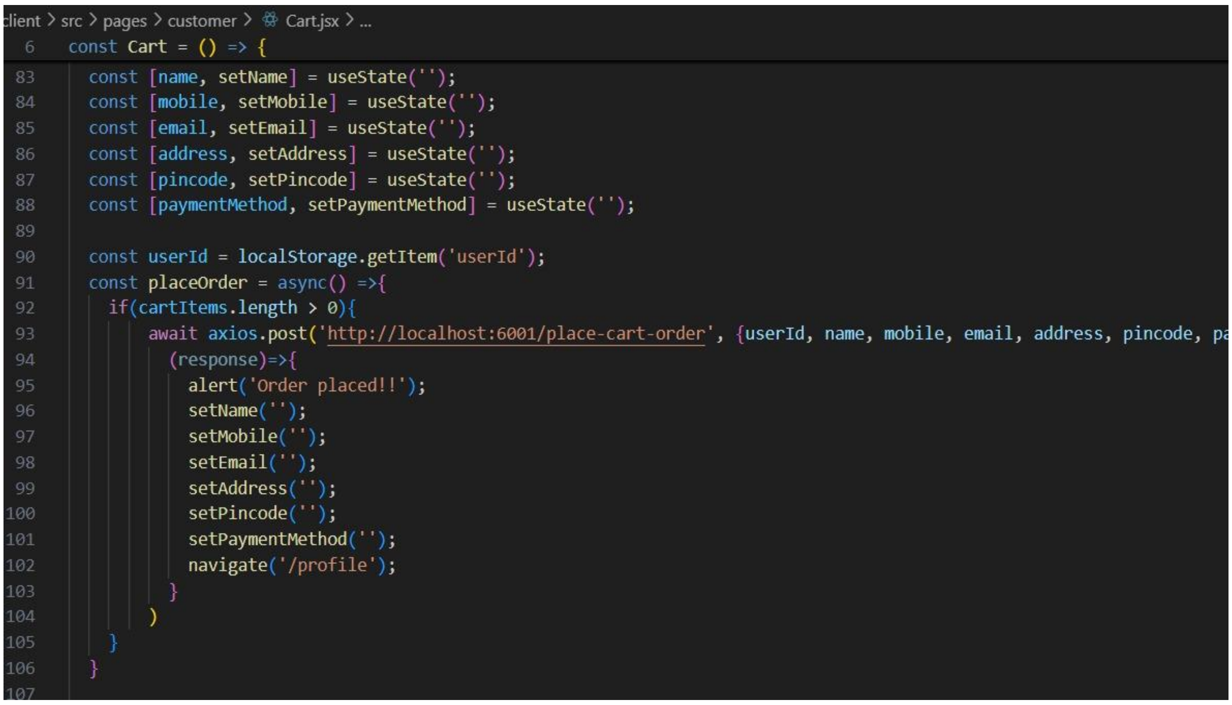
- **Admin Dashboard**

Restricted page (/admin) for managing:

Products

Orders

Users



```
lient > src > pages > customer > Cart.jsx > ...
const 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, pa
          (response)=>{
            alert('Order placed!!');
            setName('');
            setMobile('');
            setEmail('');
            setAddress('');
            setPincode('');
            setPaymentMethod('');
            navigate('/profile');
          }
        )
    }
  }
```

BACKEND DEVELOPMENT:

1. Setup express server:

• Create index.js file.

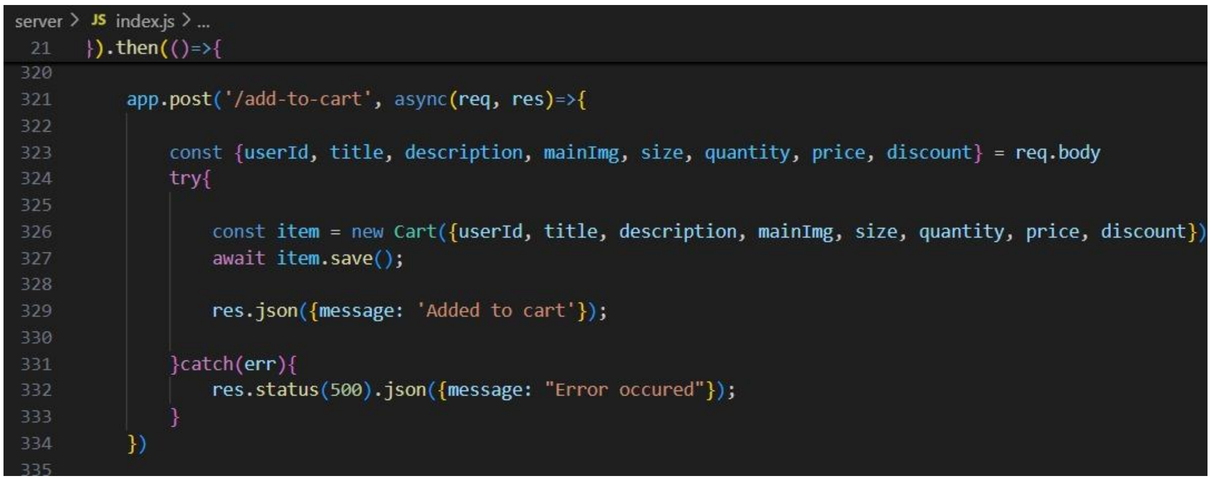
• Create an express server on your desired port number.

• Define API’s

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.



```
}).then(()=>{

    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"});
        }
    })
```

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:

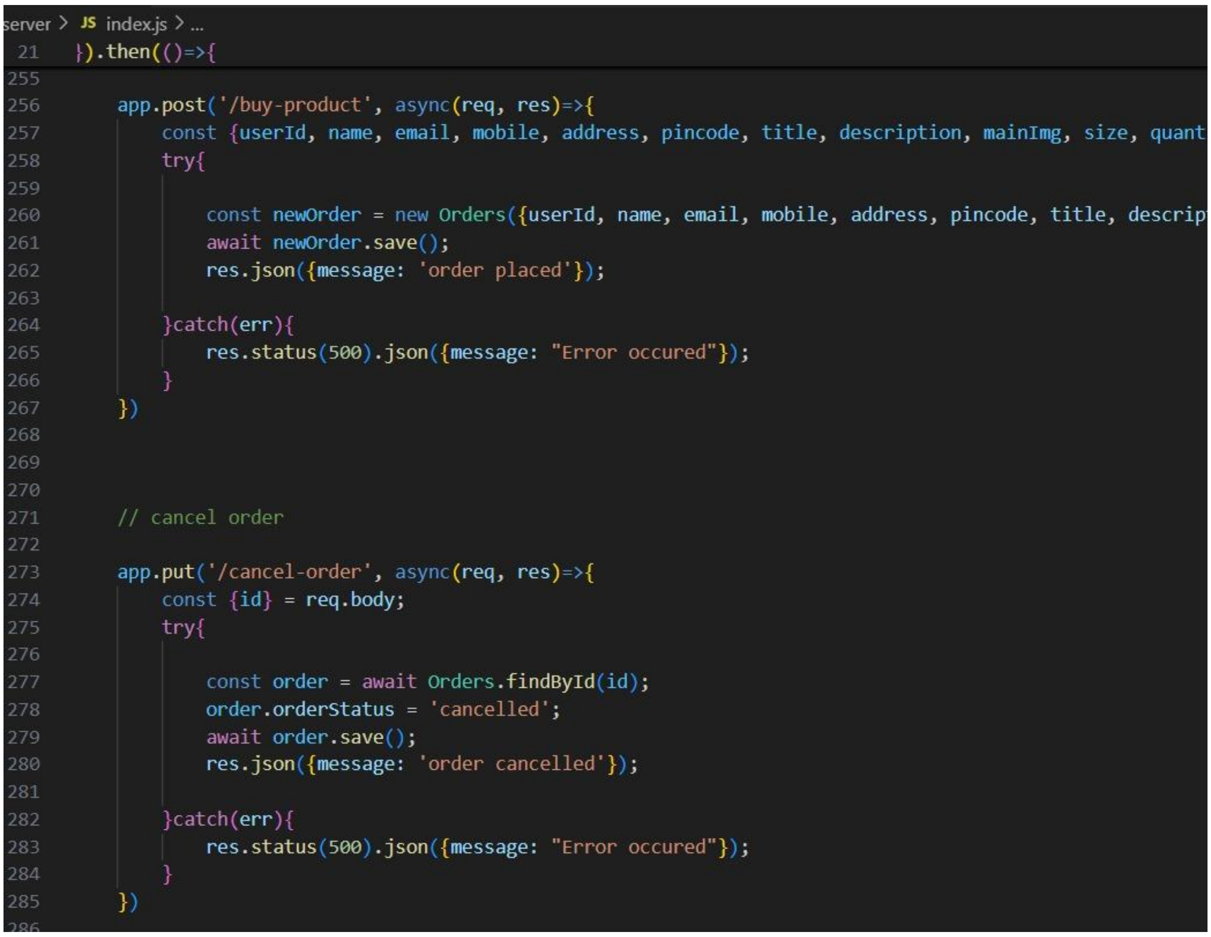
• Implement routes and controllers specific to admin functionalities such as adding products, managing user orders, etc.

• 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.



```
server > JS index.js > ...
 21   }).then(()=>{

        app.post('/buy-product', async(req, res)=>{
            const {userId, name, email, mobile, address, pincode, title, description, mainImg, size, quant
            try{

                const newOrder = new Orders({userId, name, email, mobile, address, pincode, title, descrip
                await newOrder.save();
                res.json({message: 'order placed'});

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



        // cancel order

        app.put('/cancel-order', async(req, res)=>{
            const {id} = req.body;
            try{

                const order = await Orders.findById(id);
                order.orderStatus = 'cancelled';
                await order.save();
                res.json({message: 'order cancelled'});

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

DATBASE DEVELOPMENT:

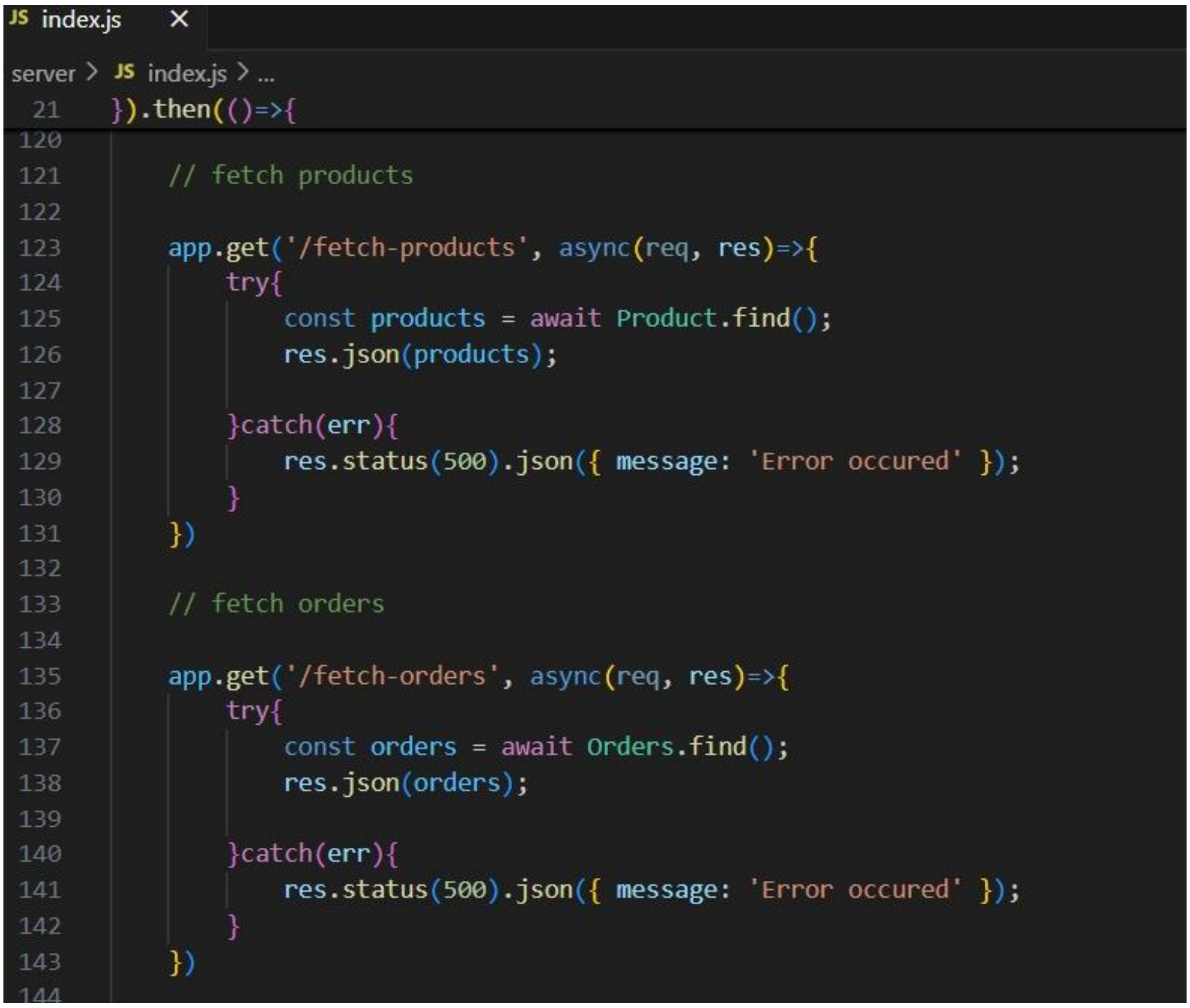
Create database in cloud

• Install Mongoose.

• Create database connection.

Reference Article:

Connect to Your Cluster - Atlas - MongoDB Docs



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

    // fetch orders

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

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

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.,

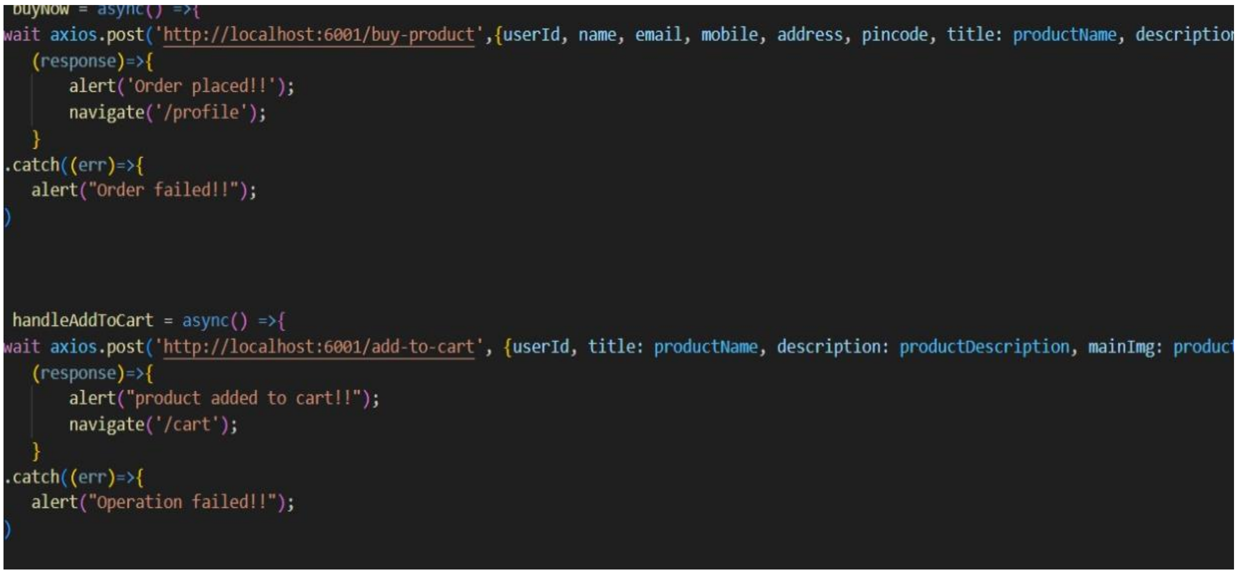
• It is used to store information about the products added to the cart by users. • The user Id field is a reference to the user who has the product in cart.

5. Admin Schema:

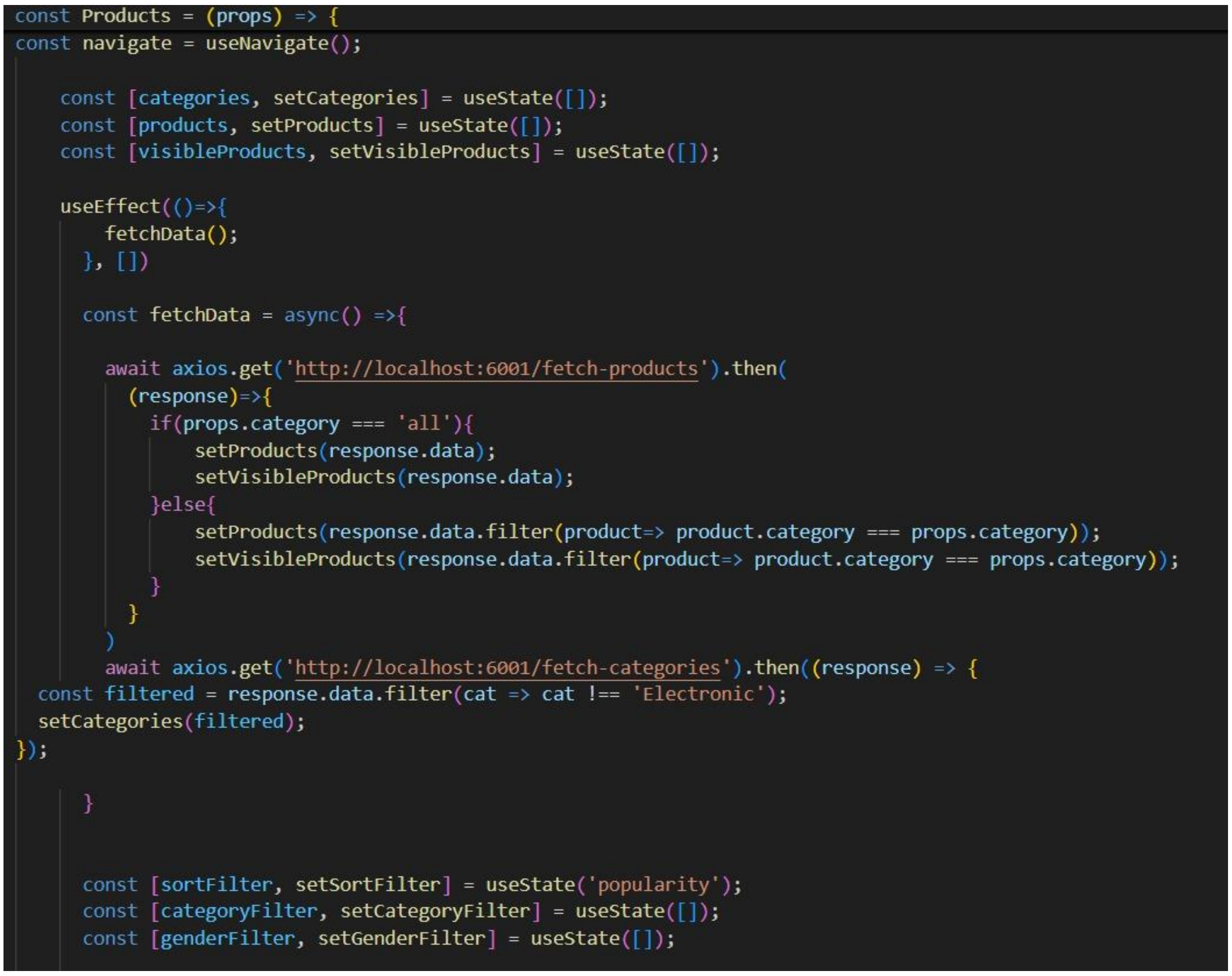
• Schema: adminSchema

• Model: ‘Admin’

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



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



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



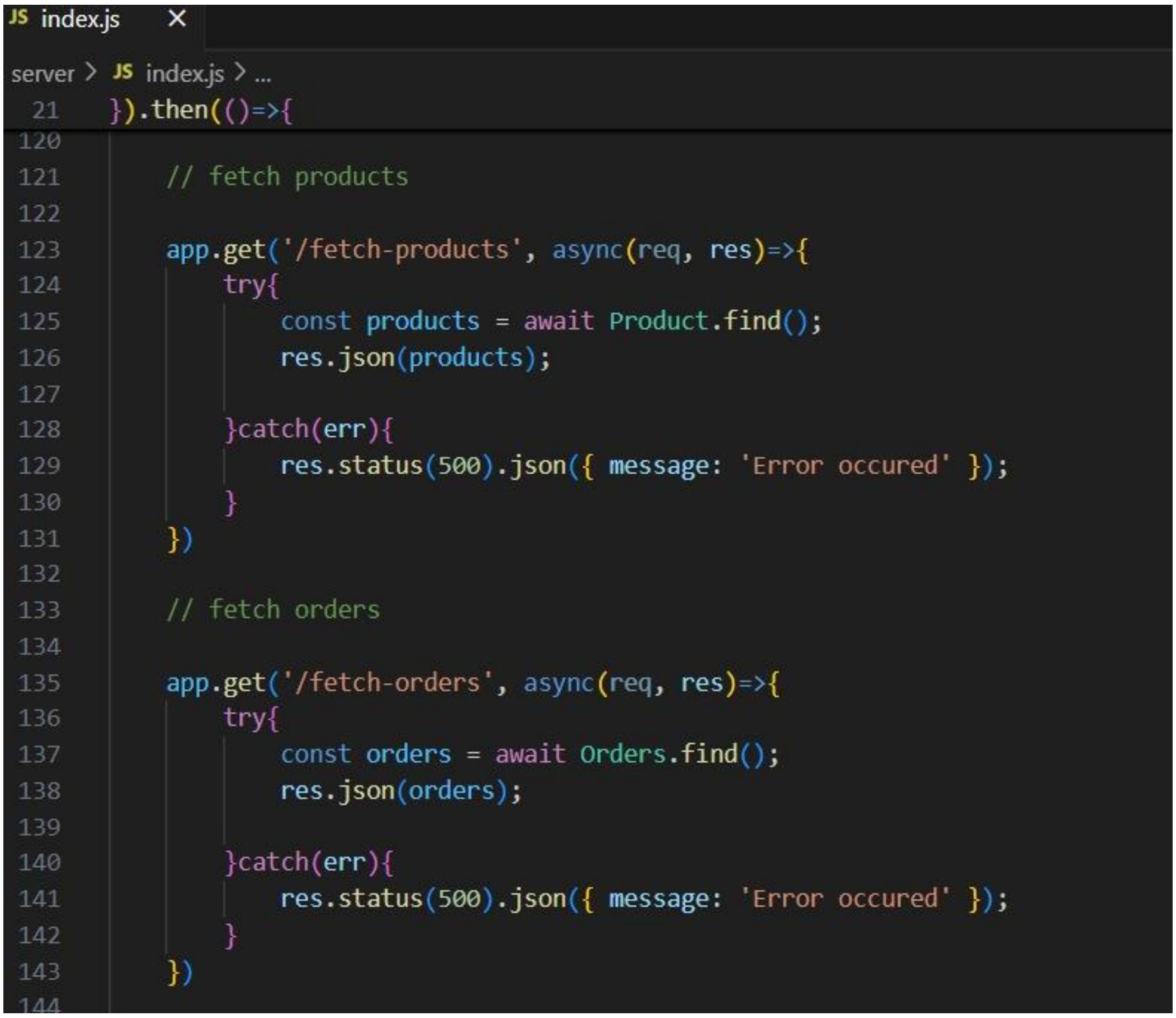
```
const Products = (props) => {
const navigate = useNavigate();

    const [categories, setCategories] = useState([]);
    const [products, setProducts] = useState([]);
    const [visibleProducts, setVisibleProducts] = useState([]);

    useEffect(()=>{
        fetchData();
      }, [])

      const fetchData = async() =>{

        await axios.get('http://localhost:6001/fetch-products').then(
          (response)=>{
            if(props.category === 'all'){
                setProducts(response.data);
                setVisibleProducts(response.data);
            }else{
                setProducts(response.data.filter(product=> product.category === props.category));
                setVisibleProducts(response.data.filter(product=> product.category === props.category));
            }
          }
        )
        await axios.get('http://localhost:6001/fetch-categories').then((response) => {
    const filtered = response.data.filter(cat => cat !== 'Electronic');
    setCategories(filtered);
});

      }


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

PROJECT IMPLEMENTATION & EXECUTION:

User Authentication

BACKEND

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



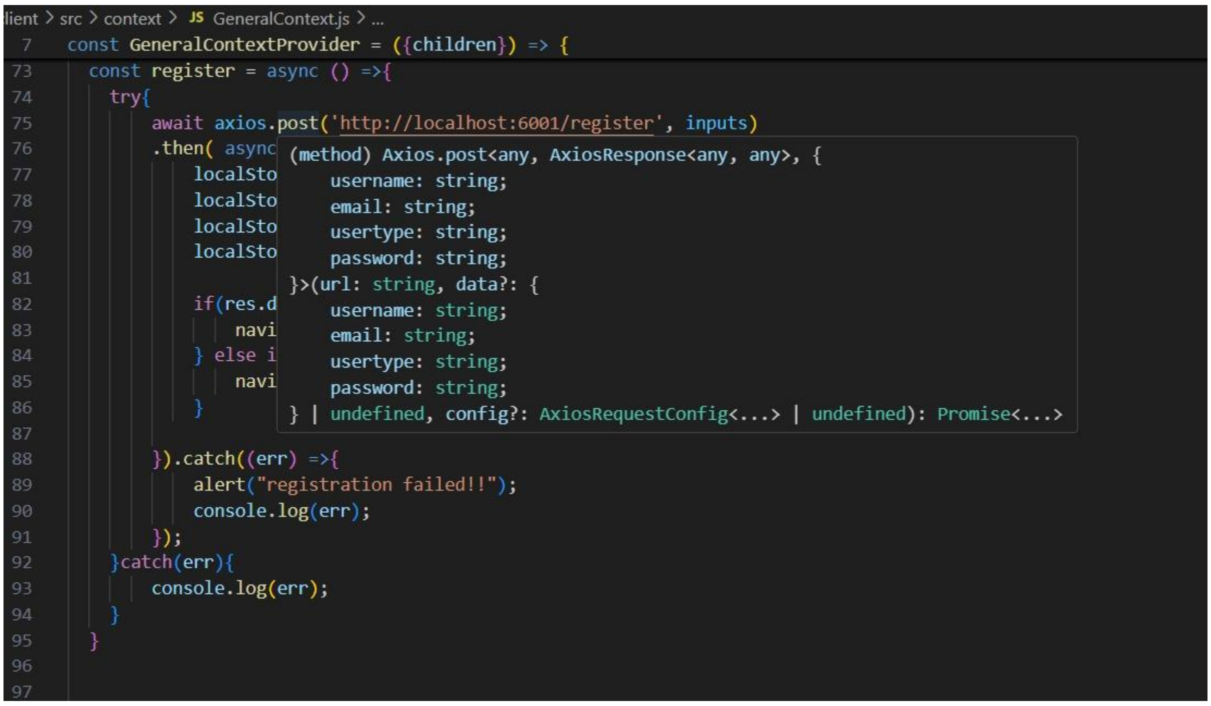
index.js

server > index.js > ...

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

    // fetch orders

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

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

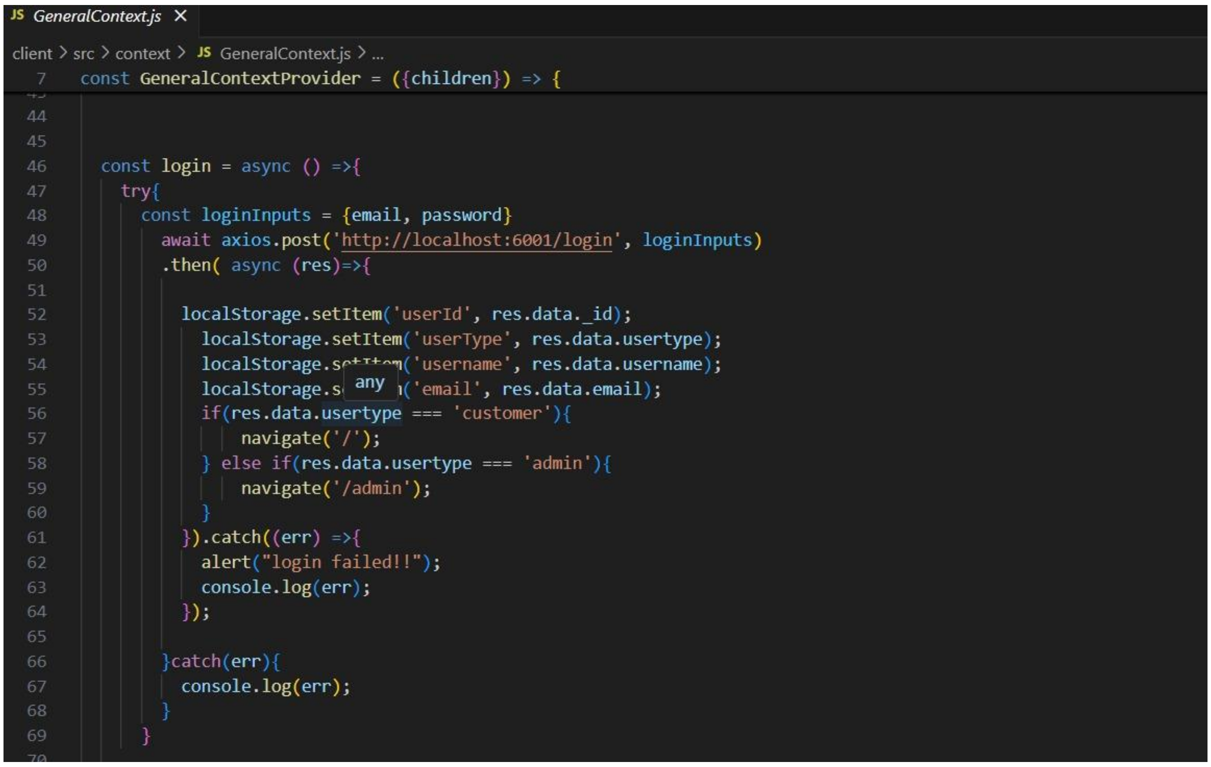
FRONTEND

Register:



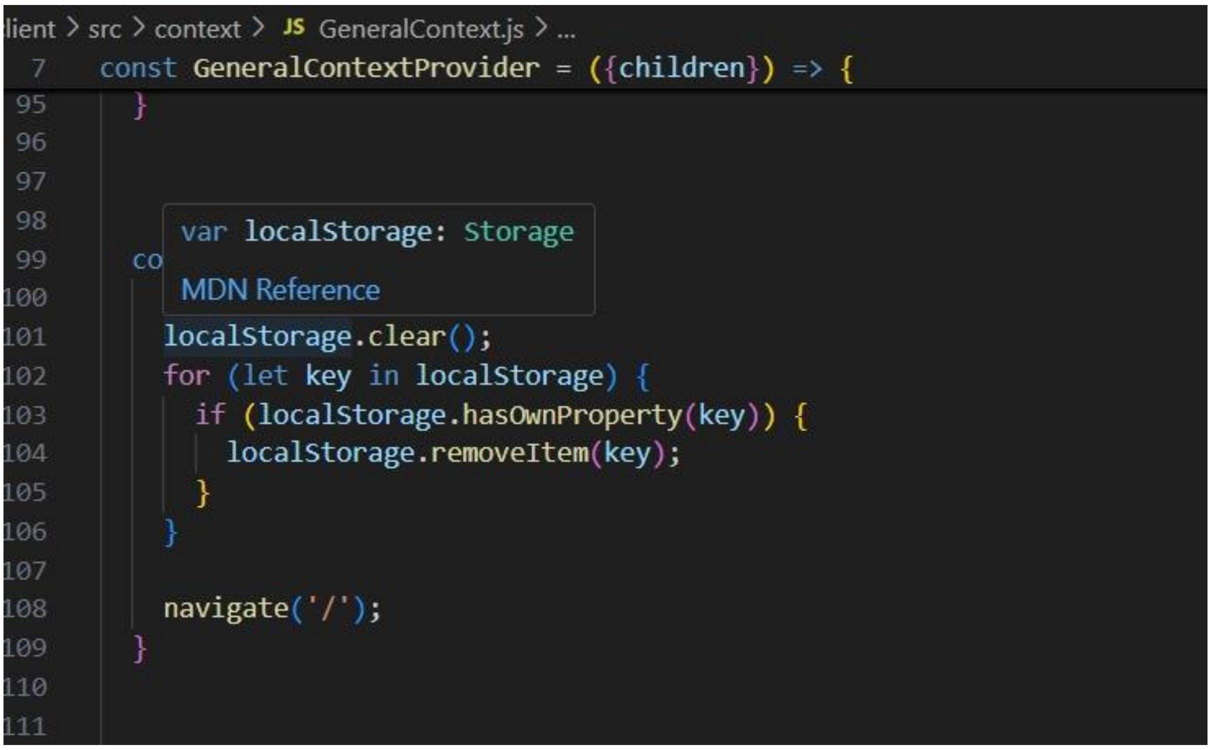
```
lient > src > context > JS GeneralContext.js > ...
const GeneralContextProvider = ({children}) => {
  const register = async () =>{
    try{
        await axios.post('http://localhost:6001/register', inputs)
        .then( async (method) Axios.post<any, AxiosResponse<any, any>, {
            localSto    username: string;
            localSto    email: string;
            localSto    usertype: string;
            localSto    password: string;
                    }>(url: string, data?: {
            if(res.d    username: string;
                navi    email: string;
            } else i    usertype: string;
                navi    password: string;
            }       } | undefined, config?: AxiosRequestConfig<...> | undefined): Promise<...>

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

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.s[illegible]('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);
      }
    }
```

Logout:


client > src > context > GeneralContext.js > ...
var localStorage: Storage
MDN Reference

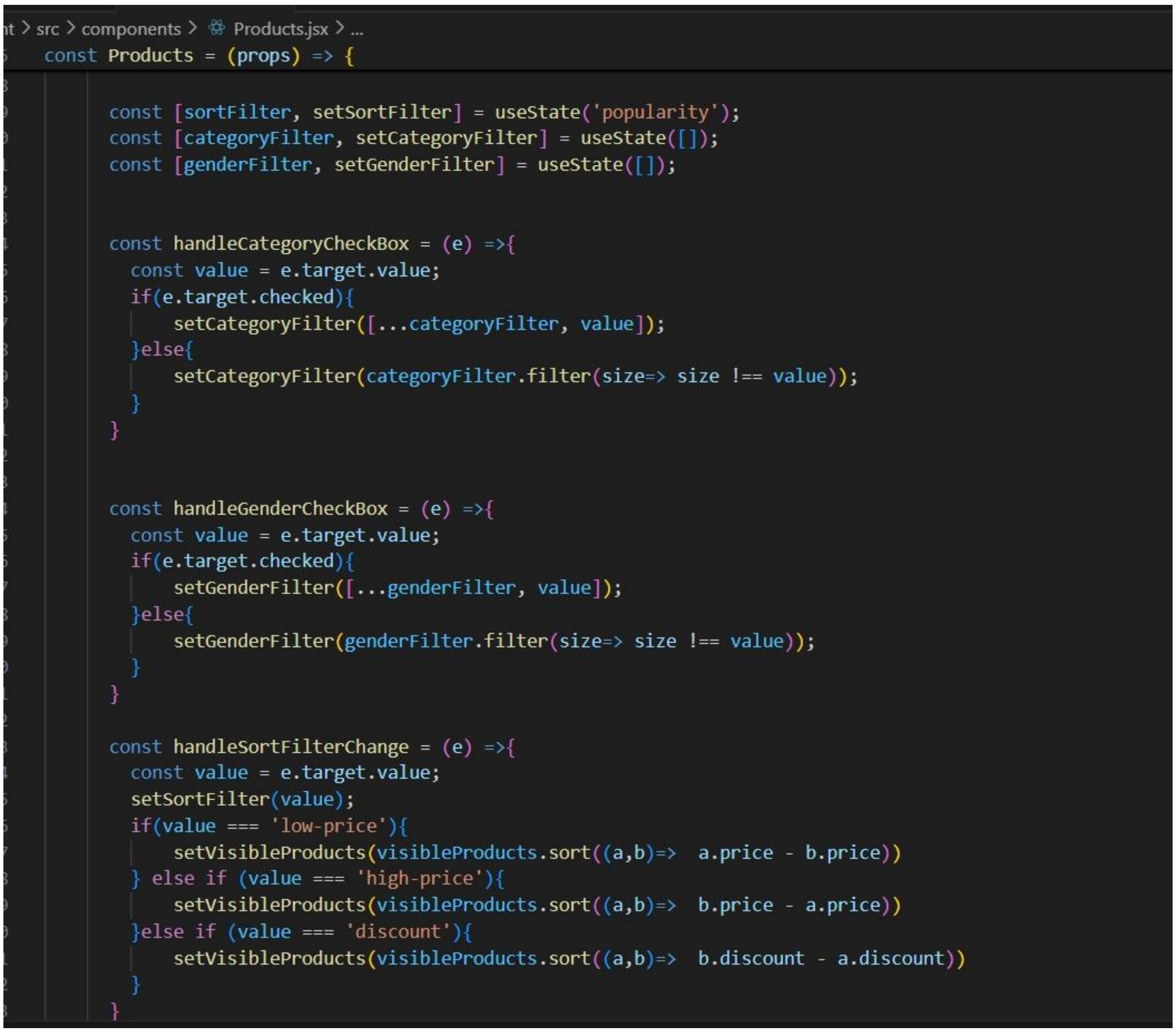
```
const GeneralContextProvider = ({children}) => {
    }



    co[illegible]

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

      navigate('/');
    }
```

All Products (User):

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

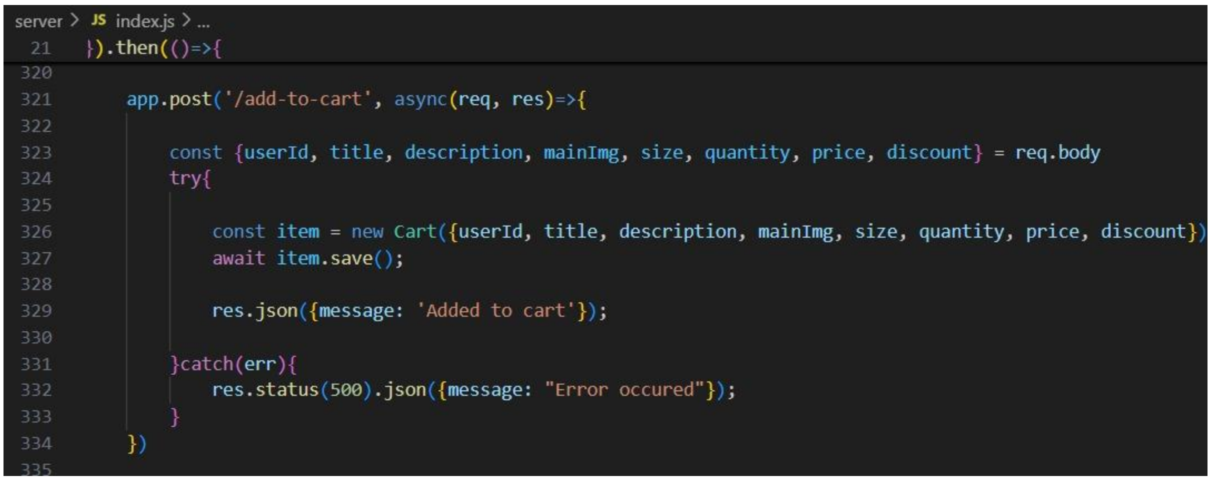
Filtering Products:

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



```
t > src > components > Products.jsx > ...
  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));
          }
        }


        const handleGenderCheckBox = (e) =>{
          const value = e.target.value;
          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))
          }
        }
```

Add Product to Cart:

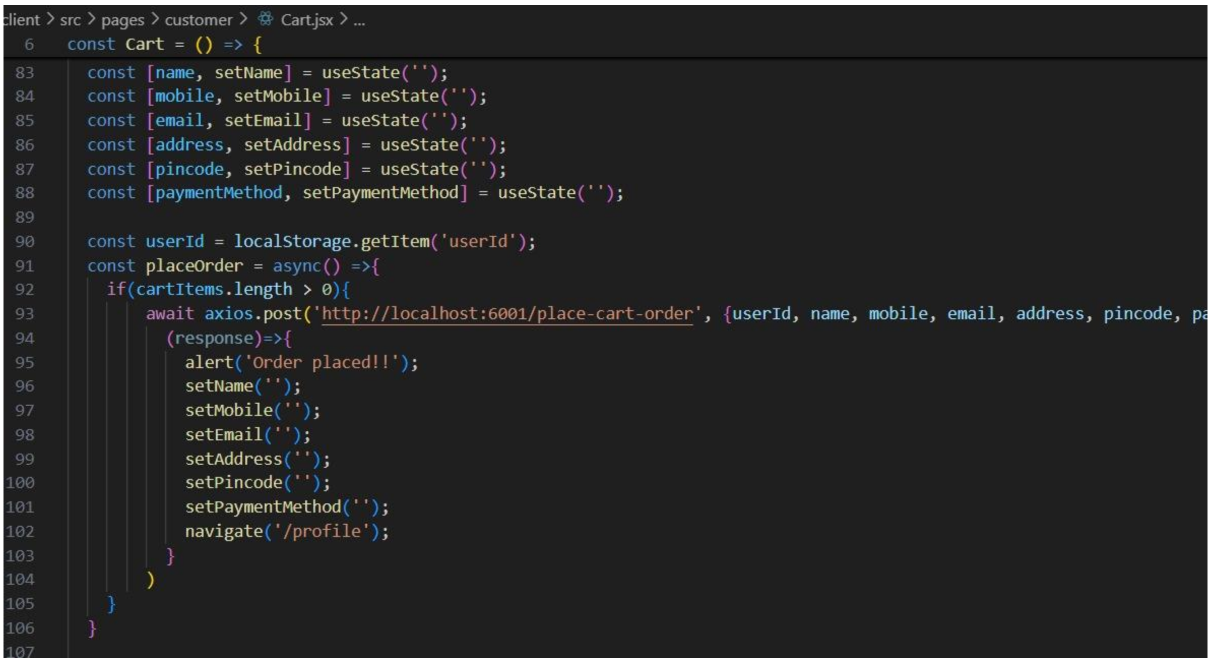


```
server > JS index.js > ...
}).then(()=>{

    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

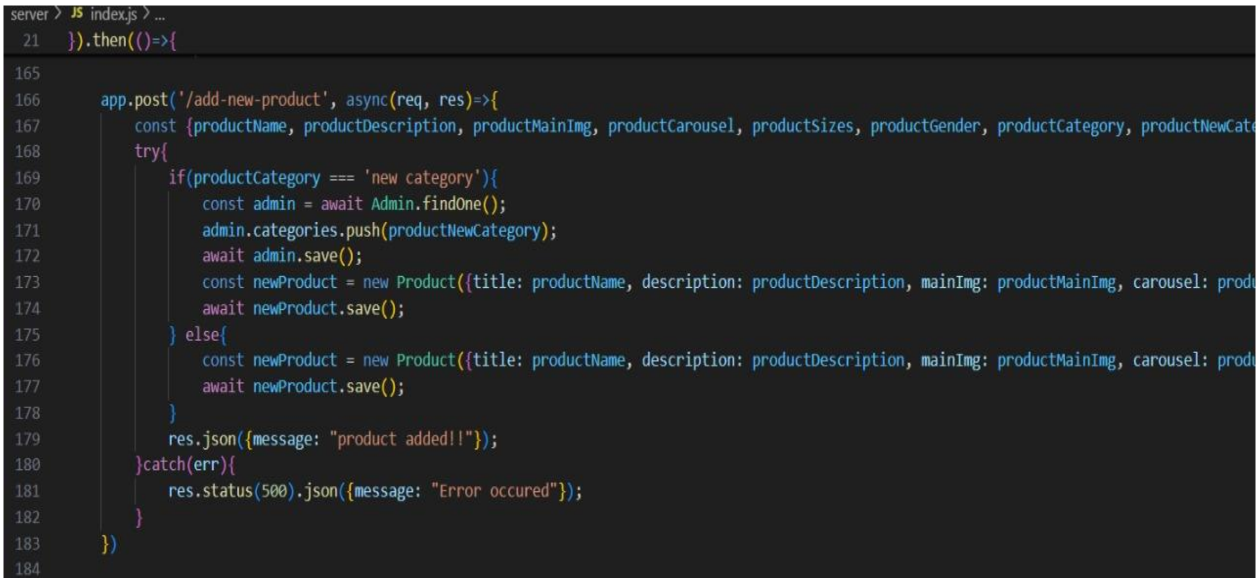


```
lient > src > pages > customer > Cart.jsx > ...
const Cart = () => {
    const [name, setName] = useState('');
    const [mobile, setMobile] = useState('');
    const [email, setEmail] = useState('');
    const [address, setAddress] = useState('');
    const [pincode, setPincode] = useState('');
    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, pa
            (response)=>{
              alert('Order placed!!');
              setName('');
              setMobile('');
              setEmail('');
              setAddress('');
              setPincode('');
              setPaymentMethod('');
              navigate('/profile');
            }
          )
      }
    }
```

In the backend , on receiving the request from the client, we then place the order for the products in the cart with specific user id.

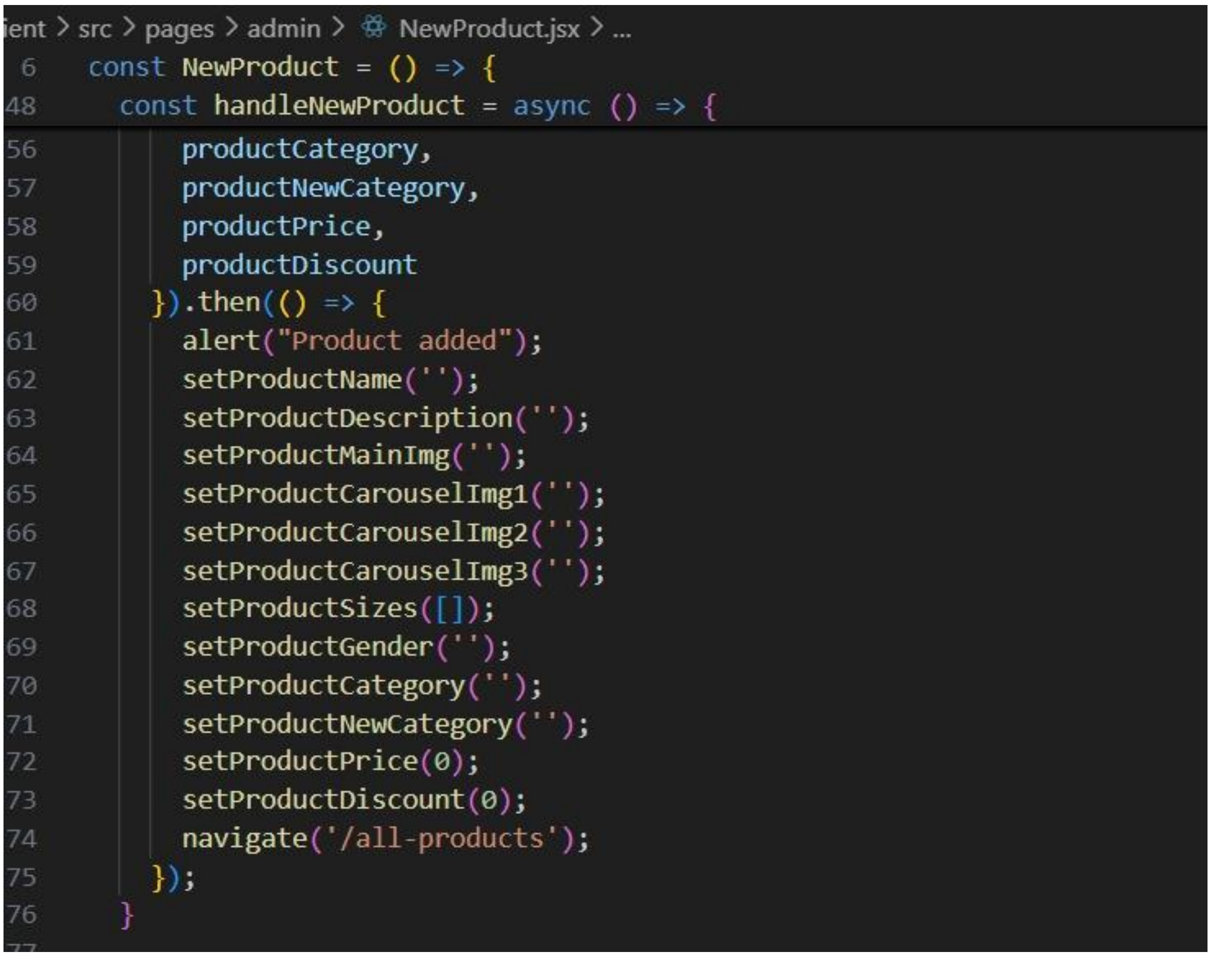
Add New Product:

BACKEND



```
server > JS index.js > ...
}).then(()=>{

    app.post('/add-new-product', async(req, res)=>{
        const {productName, productDescription, productMainImg, productCarousel, productSizes, productGender, productCategory, productNewCate
        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: produ
                await newProduct.save();
            } else{
                const newProduct = new Product({title: productName, description: productDescription, mainImg: productMainImg, carousel: produ
                await newProduct.save();
            }
            res.json({message: "product added!!"});
        }catch(err){
            res.status(500).json({message: "Error occured"});
        }
    })

```

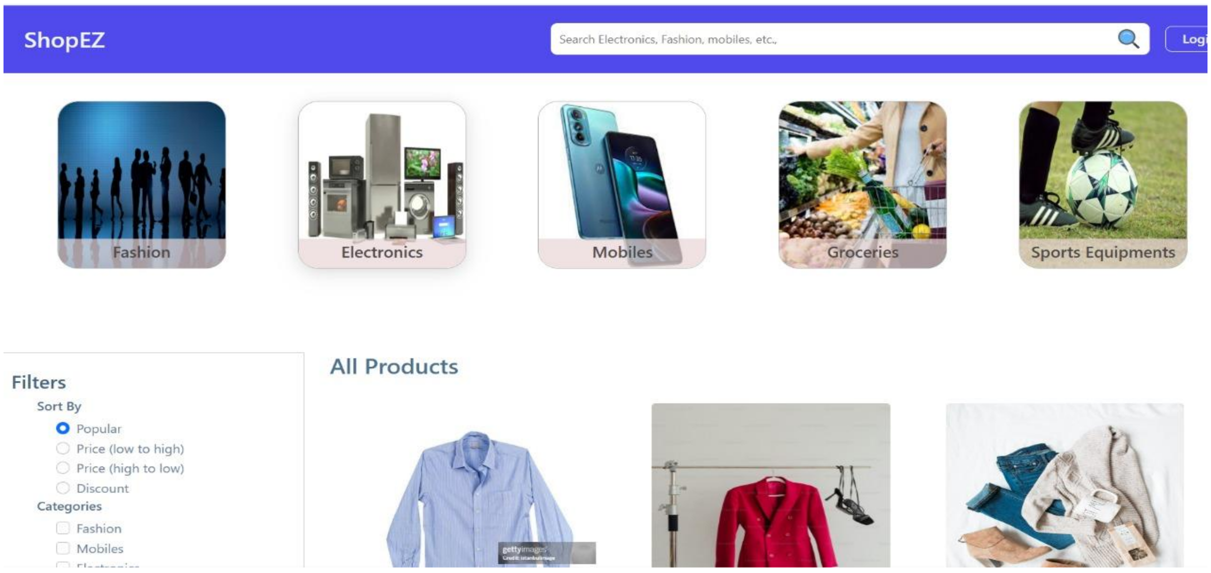
FRONTEND



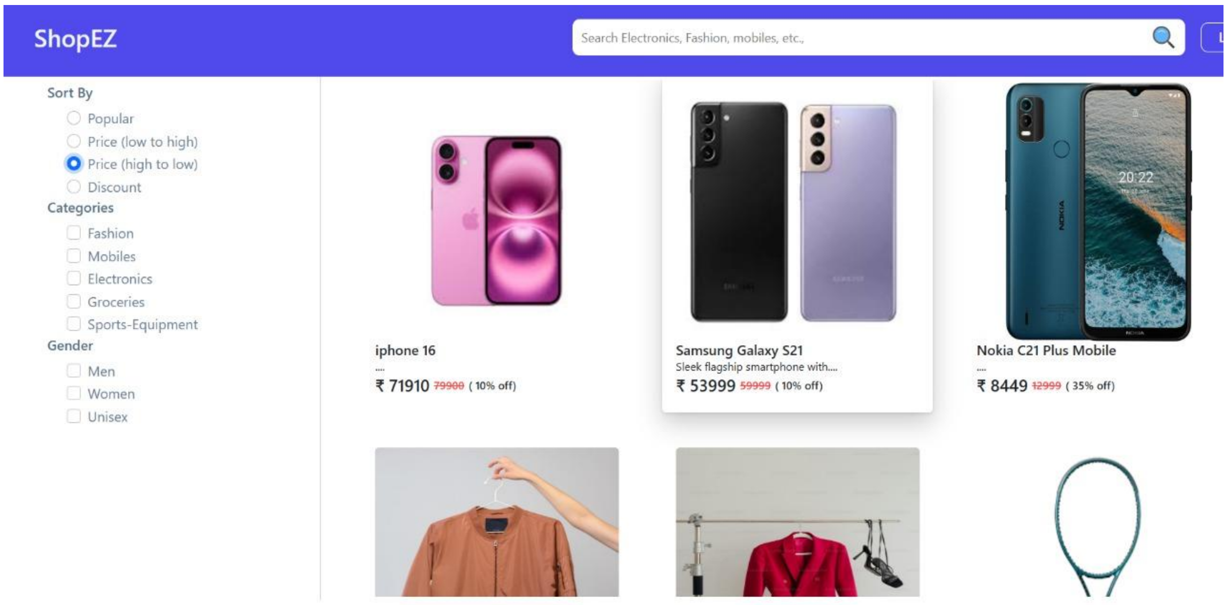
```
ient > src > pages > admin > NewProduct.jsx > ...
const NewProduct = () => {
  const handleNewProduct = async () => {
        productCategory,
        productNewCategory,
        productPrice,
        productDiscount
      }).then(() => {
        alert("Product added");
        setProductName('');
        setProductDescription('');
        setProductMainImg('');
        setProductCarouselImg1('');
        setProductCarouselImg2('');
        setProductCarouselImg3('');
        setProductSizes([]);
        setProductGender('');
        setProductCategory('');
        setProductNewCategory('');
        setProductPrice(0);
        setProductDiscount(0);
        navigate('/all-products');
      });
  }
```

DEMO UI IMAGES:

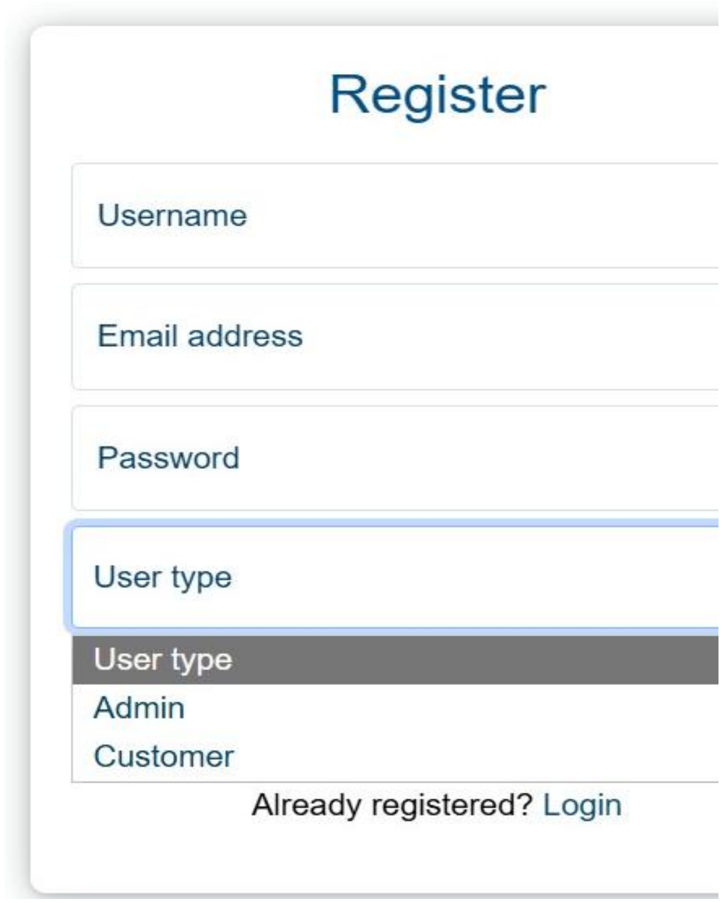
Landing Page



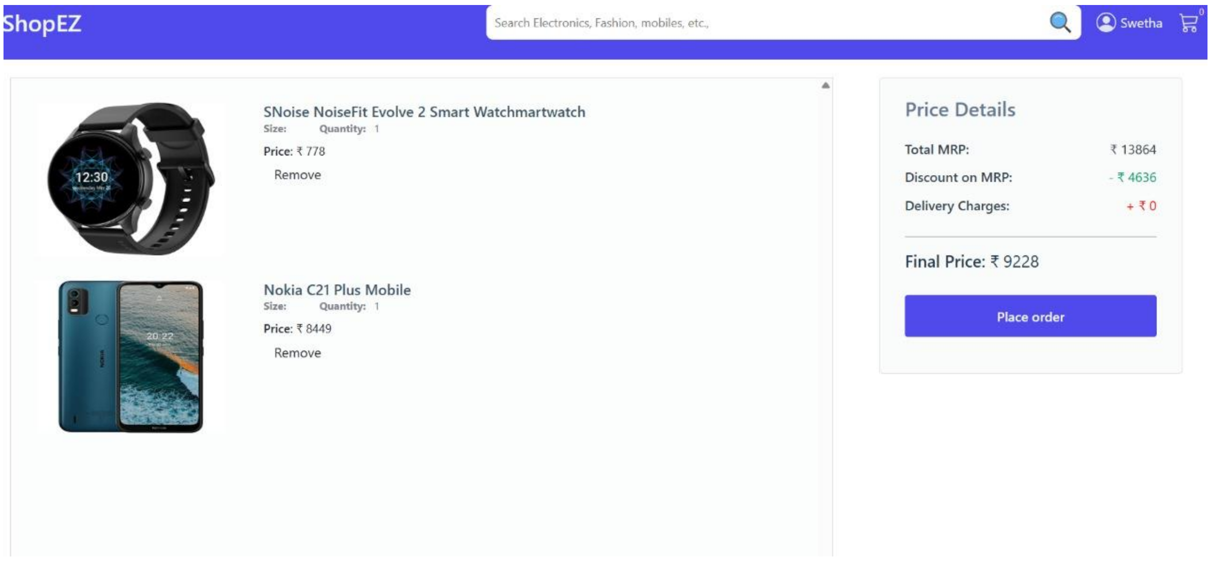
Products



Authentication



Cart



SNoise NoiseFit Evolve 2 Smart Watchmartwatch
Size: Quantity: 1
Price: ₹ 778
Remove

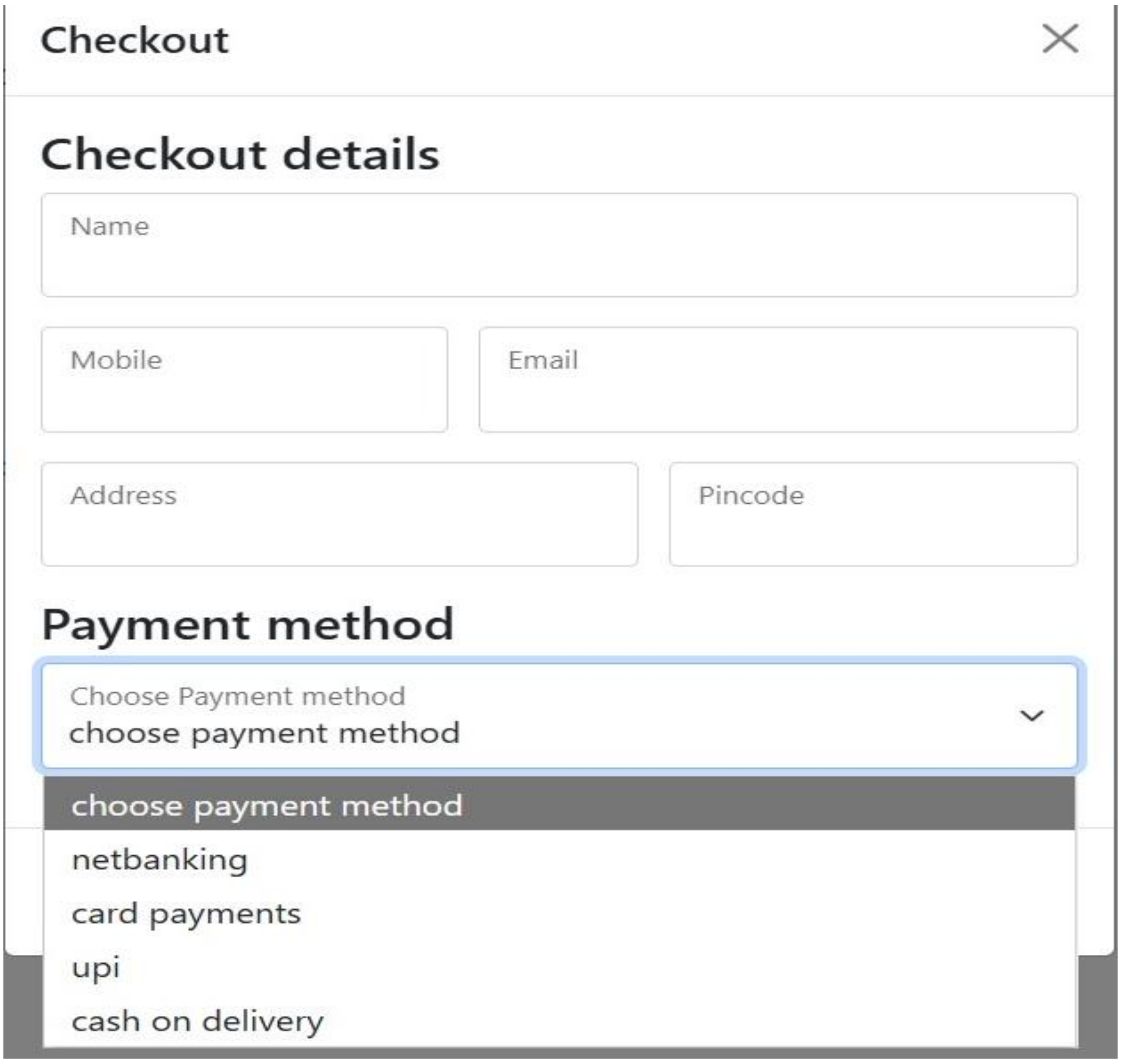
Nokia C21 Plus Mobile
Size: Quantity: 1
Price: ₹ 8449
Remove

Price Details

Total MRP:	₹ 13864
Discount on MRP:	- ₹ 4636
Delivery Charges:	+ ₹ 0

Final Price: ₹ 9228

Checkout Page


Checkout

Checkout details

Name

Mobile

Email

Address

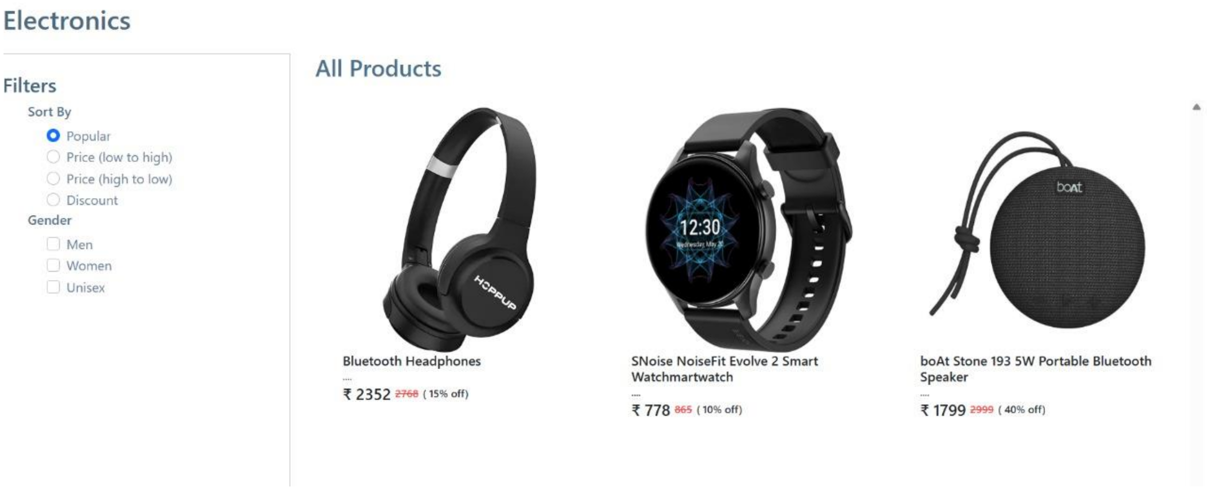
Pincode

Payment method

Choose Payment method
choose payment method

choose payment method
netbanking
card payments
upi
cash on delivery

All Products


Electronics

Filters

Sort By

[✓ Popular]
Price (low to high)
Price (high to low)
Discount

Gender

Men
Women
Unisex

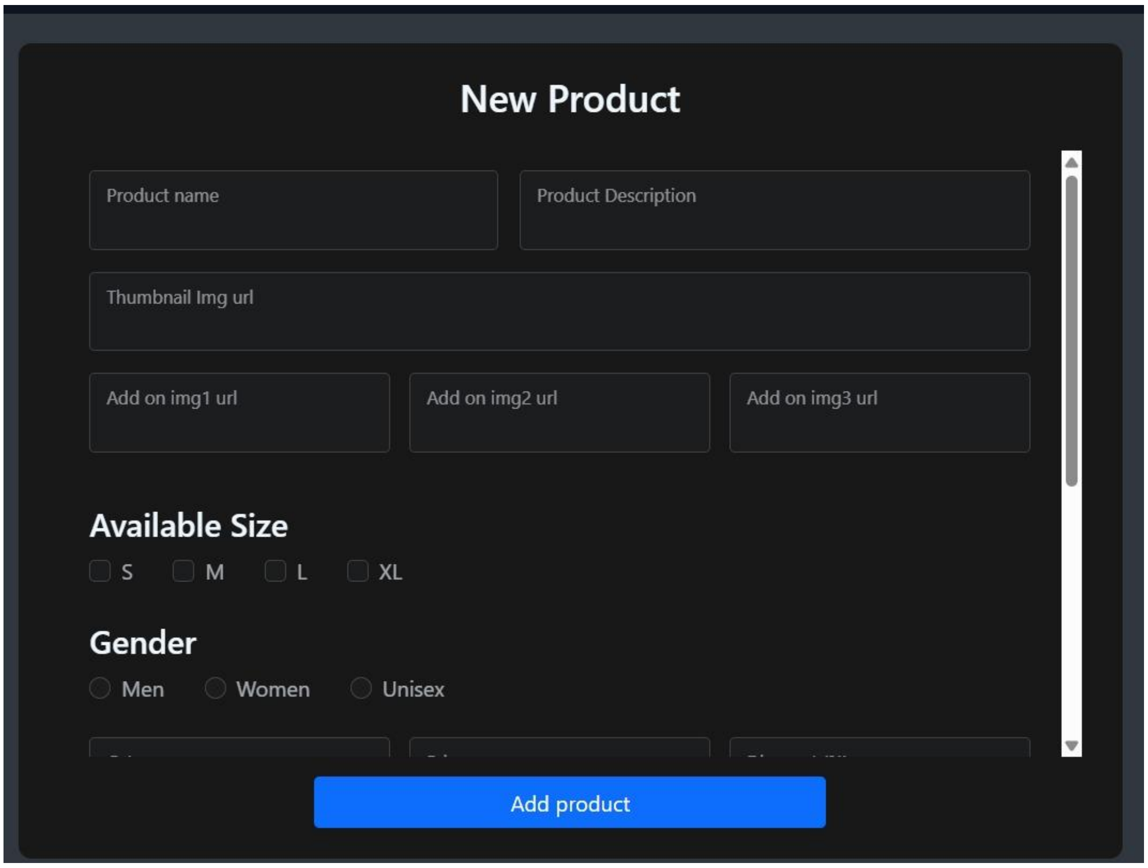
All Products

Bluetooth Headphones
₹ 2352 ~~2768~~ (15% off)

SNoise NoiseFit Evolve 2 Smart Watchmartwatch
₹ 778 ~~865~~ (10% off)

boAt Stone 193 5W Portable Bluetooth Speaker
₹ 1799 ~~2999~~ (40% off)

New Product Page



CONCLUSION

The ShopEZ project successfully demonstrates the development of a full-stack e-commerce web application using the MERN stack (MongoDB, Express.js, React.js, Node.js). The platform is designed to deliver a seamless and personalized shopping experience for users while also providing powerful tools for sellers to manage their products and orders efficiently.

By implementing key e-commerce functionalities such as user authentication, product catalog, shopping cart, checkout process, and admin dashboards, ShopEZ achieves both frontend responsiveness and backend robustness. It leverages React for dynamic UI, Express for building APIs, MongoDB for flexible data storage, and JWT/bcrypt for secure user authentication.

The ShopEZ project illustrates how modern web technologies can be combined to create a responsive, secure, and user-friendly e-commerce solution. It also provides practical experience in full-stack development, making it an excellent capstone or portfolio project for aspiring web developers.