ShopEZ: E-commerce Application - Detailed Documentation

Introduction:

ShopEZ is a comprehensive and user-friendly e-commerce application designed to provide a smooth and satisfying online shopping experience. It offers a rich catalog of products and intelligent seller dashboards that simplify the buying and selling process. The platform includes all essential features for a full-fledged e-commerce service.

Key Features:

- ShopEZ enables users to discover products quickly and easily through intuitive navigation and smart searchtools.
- The system provides personalized recommendations based on browsing history, preferences, and purchasebehavior.
- Users can complete transactions securely through a seamless checkout system that supports modernpayment methods.
- Once an order is placed, customers receive immediate confirmation and can track their delivery in real-time.
- Sellers benefit from a comprehensive dashboard that allows efficient product and order management.- Business owners gain valuable insights through integrated analytics tools to drive growth and make informed decisions.

Scenario: Sarahs Birthday Gift

Consider the example of Sarah, a working professional looking for a gift for her friend Emily. Sarah logs into ShopEZ and follows this process:

1. She begins by browsing the Fashion Accessories category and uses filters to narrow down her search bystyle and budget.
2. Among the filtered results, she sees a gold bangle listed under 'Recommended for You' which matches

Emilys taste.

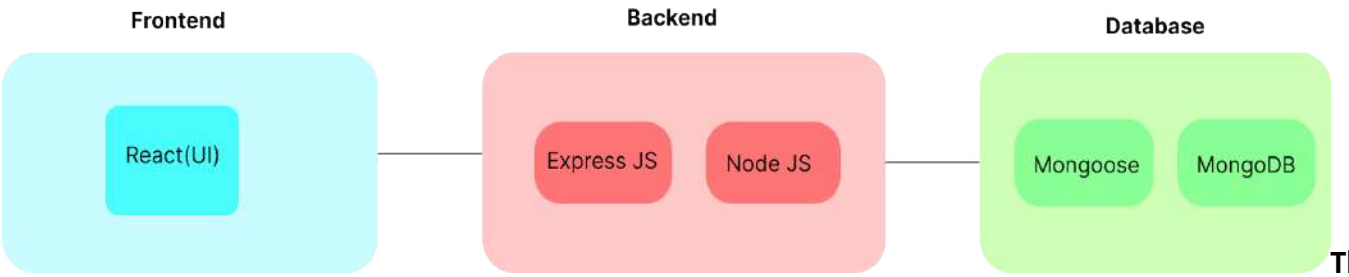
3. Sarah proceeds to checkout, where she securely enters her shipping and payment details.

4. She instantly receives an email confirming the order, which gives her confidence and assurance.

5. The seller is alerted via their dashboard and quickly prepares the product for dispatch.

6. Emily receives the gift on time and is thrilled, while Sarah appreciates how simple and enjoyable theprocess was with ShopEZ.

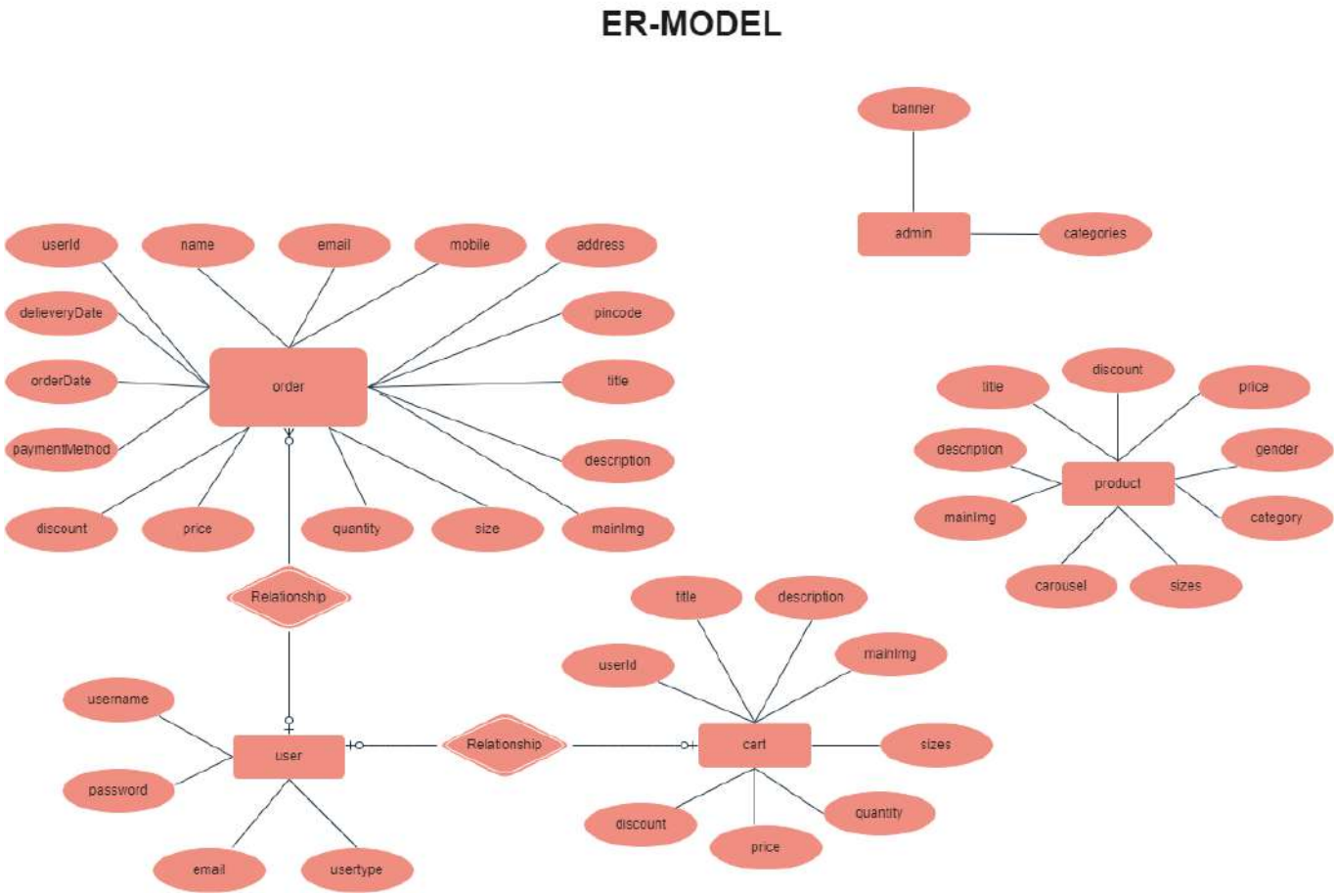
Technical Architecture:

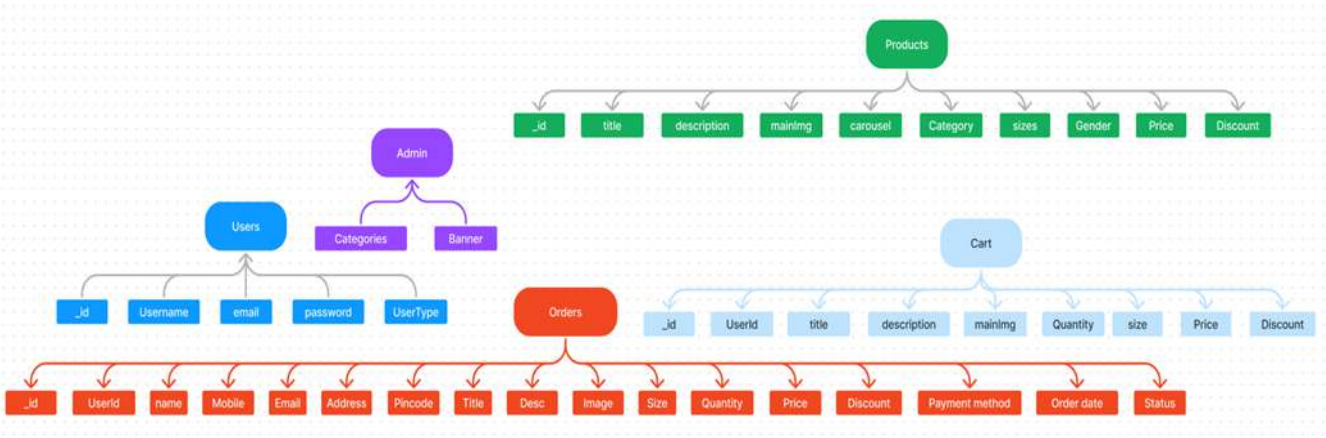


The ShopEZ platform is built using the MERN stack (MongoDB, Express.js, React.js, Node.js), ensuring flexibility, scalability, and performance.

- The frontend is developed using React.js and includes reusable components such as Authentication, Cart,Products, User Profile, and Admin Panel.
- The backend uses Node.js with the Express.js framework to provide RESTful APIs that handle users,product listings, order management, and administrative functions.
- MongoDB serves as the NoSQL database, used to store structured data related to users, products, carts,orders, and admin configurations.

ER Diagram Overview:





The application data is organized into several interconnected entities:

- User: This entity stores user registration, login credentials, and personal information.

- Admin: This entity is responsible for managing categories, promotional banners, and overseeing platform activity.

- Products: Each product listed on ShopEZ contains metadata including name, price, category, description,and stock information.
- Cart: This entity stores the products that users add to their cart and is linked to each user via userId.
- Orders: Represents purchases made by users, storing order details, status, and payment history.

Key Features:

ShopEZ offers a variety of essential and advanced features to support users and sellers:

1. A detailed product catalog with filtering and categorization for easy browsing.
2. A prominent Shop Now button for quick engagement.
3. An order details page that allows users to view their past and current purchases.
4. A secure checkout process that protects personal and financial information.
5. Instant order confirmation via email or notification.
6. A seller dashboard that supports product management, order tracking, and customer interaction.

Prerequisites:

To run the ShopEZ application, the following tools and technologies must be installed:

- 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 as the database:

  - 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:

Git clone:

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.

User and Admin Flow:

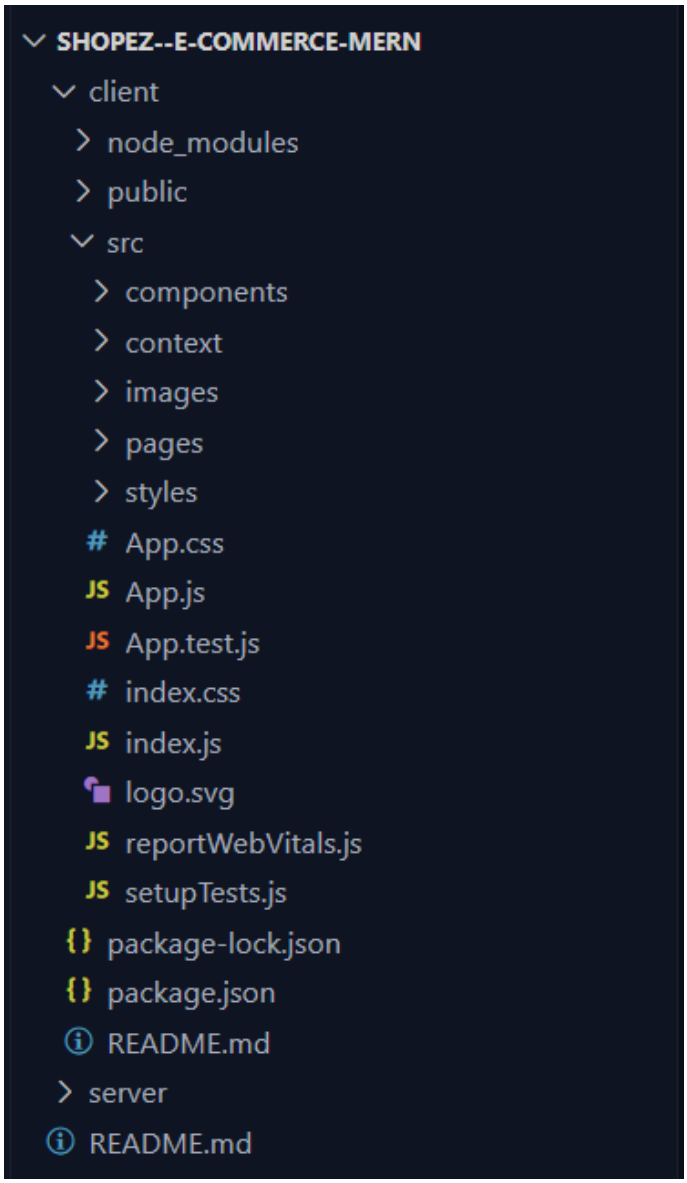
User Flow:

- Users begin by registering on the platform.
- After logging in, they can browse the product catalog, add items to their cart, and proceed to checkout.
- They can then track the status of their orders from their profile.

Admin Flow:

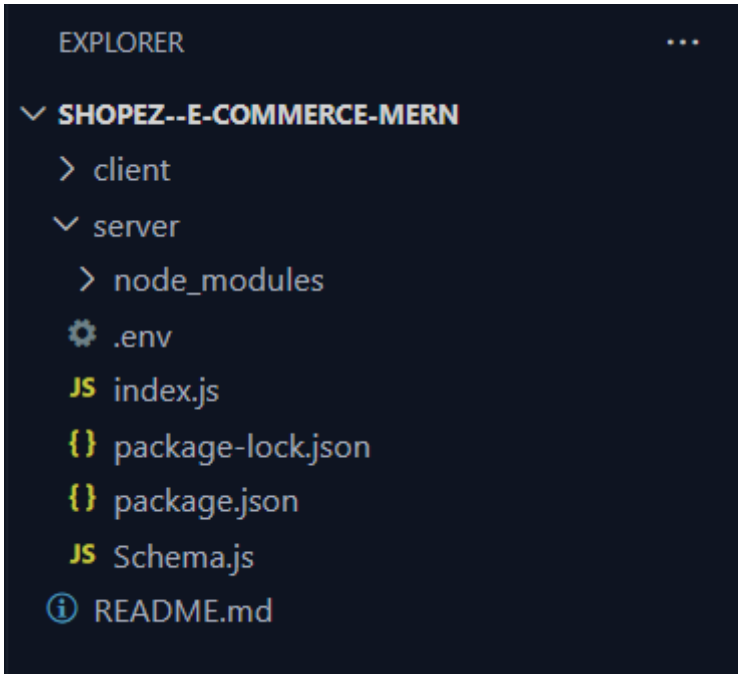
- Admins log in using special credentials.
- They access a dashboard where they can add, update, or delete products, and manage users and orderrecords.

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.



PROJECT SETUP AND CONFIGURATION:

Install required tools and software:

• Node.js.

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

• Git.

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

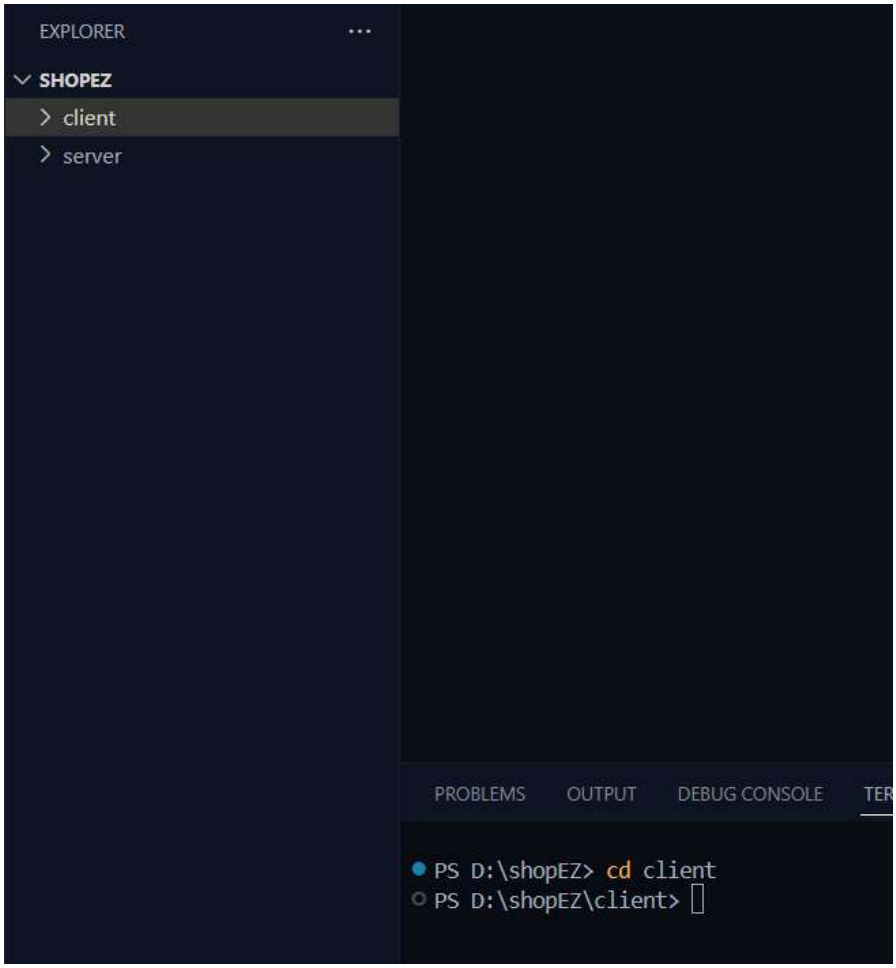
Create project folders and files:

• Client folders.

• Server folders

ReferralVideoLink:

https://drive.google.com/file/d/1uSMbPIAR6rfAEMcb_nLZAZd5QIjTpnYQ/view?usp=sharing



DATABASE DEVELOPMENT:

Create database in cloud video link:- https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view

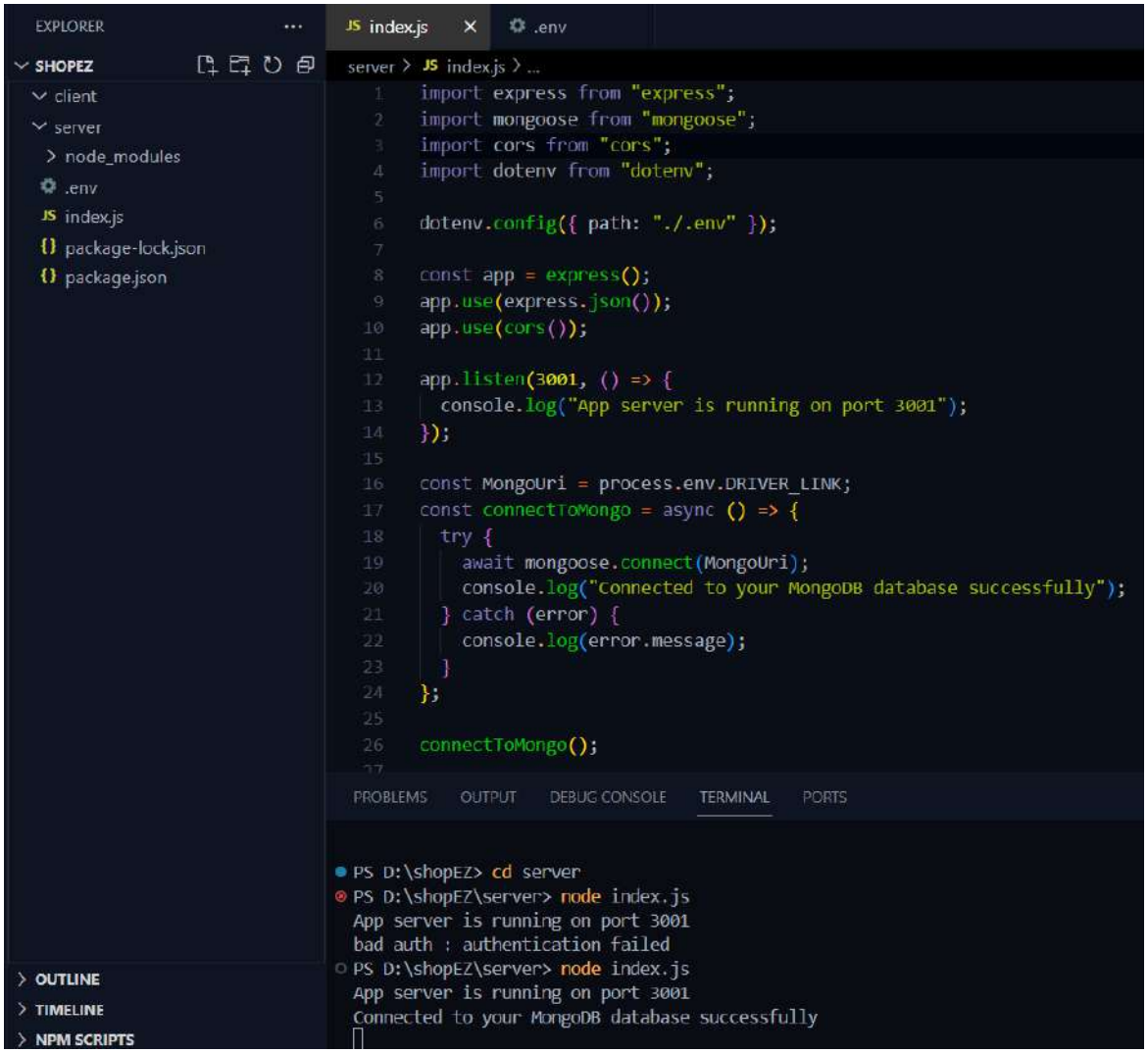
• Install Mongoose.

• Create database connection.

Reference Video of connect node with mongoDB database: https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

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

Reference Image:



Database Design:

The application uses the following database schemas:

- userSchema: Stores user data and authentication details.

- productSchema: Contains details about each product listed.

- cartSchema: Links users with products added to their cart.

- ordersSchema: Records each order made by users.

- adminSchema: Contains admin account data and privileges.

Code Explanation:

Schemas:

Now let us define the required schemas

Schema.js

server > Schema.js > productSchema

```javascript
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}
})
```

Schema.js

server > Schema.js > productSchema

```javascript
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},
    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);
```

Backend Development

Setup express server:

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

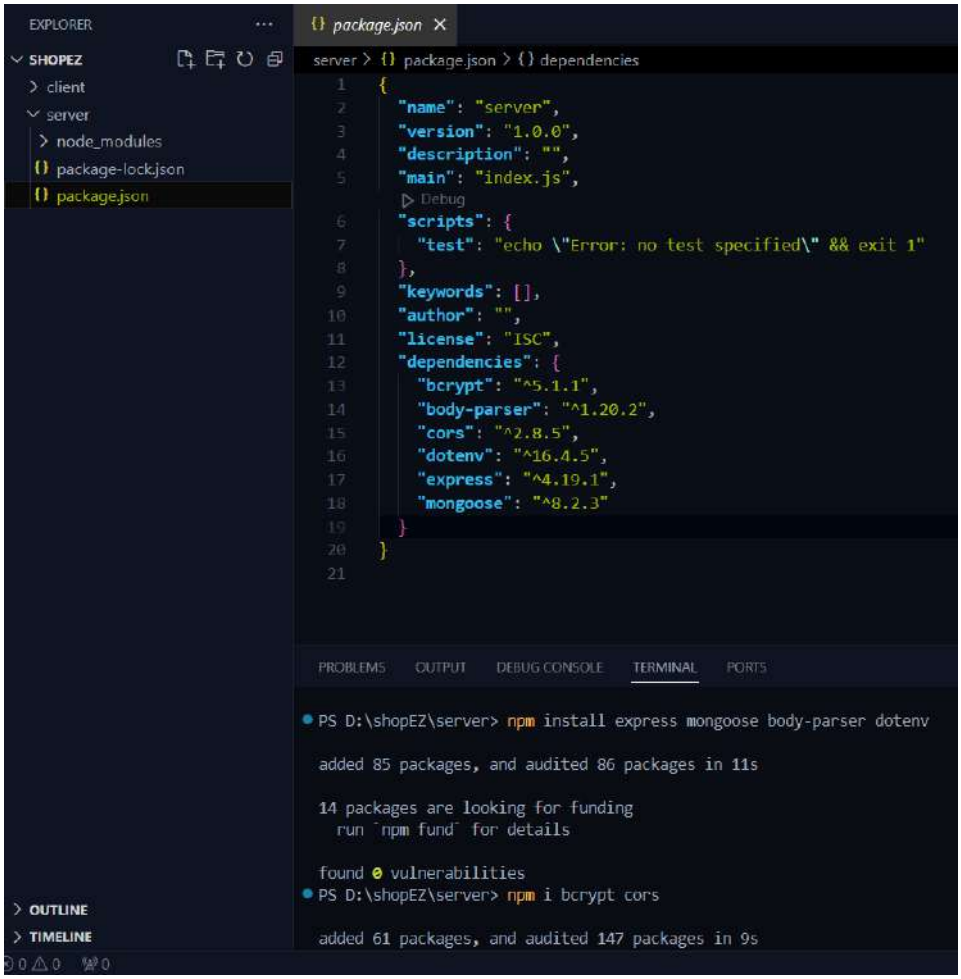
Referencevideo:https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

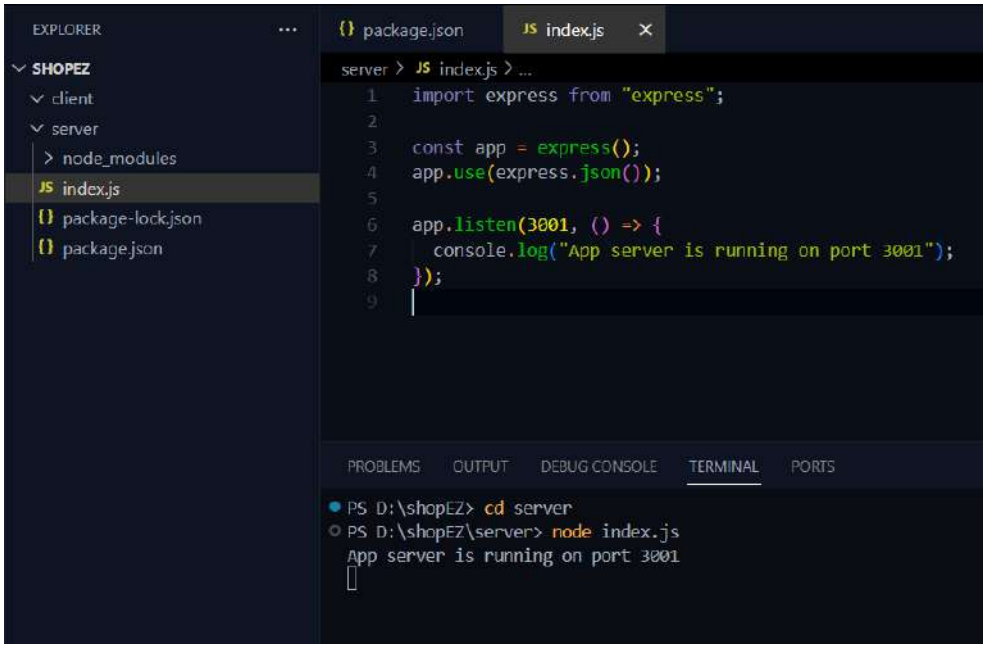
Set Up Project Structure:

- 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.

ReferenceVideo:https://drive.google.com/file/d/19df7NU-gQK3DO6wr7ooAfJYIQwnemZoF/view?usp=sharing

Reference Images:





The backend is built using Express.js and includes:

- An Express server to handle API requests.

- Defined routes for users, products, carts, and orders.

- JWT-based authentication to protect endpoints.

- Mongoose models to interact with MongoDB.

- Middleware for error handling and data validation.

Frontend Development:

Frontend development involves:

- Setting up a React project structure.

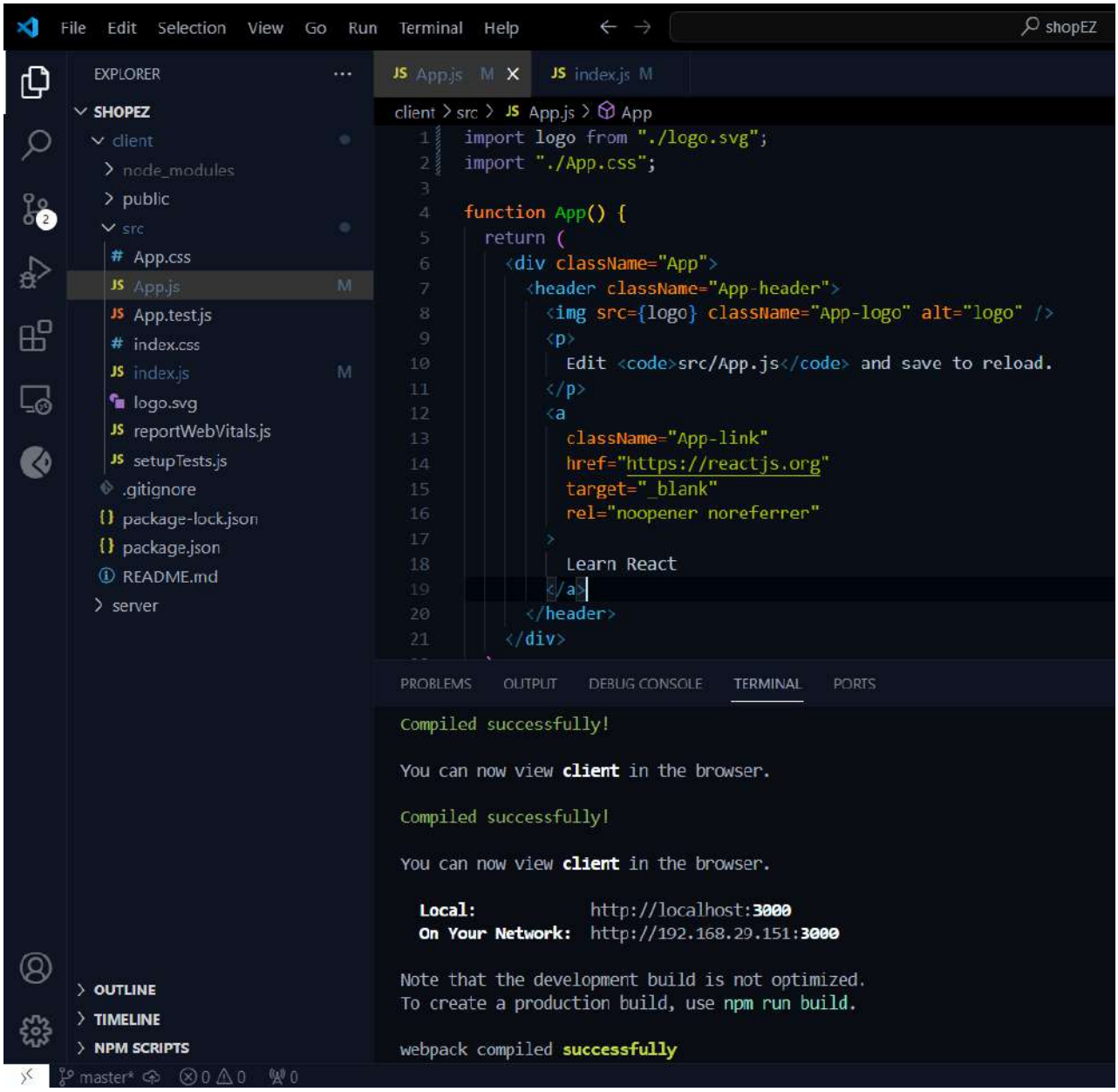
- Designing the user interface using reusable components.

- Implementing routing between different pages.

- Integrating APIs for dynamic data rendering.

Implementation Examples

Reference Image:



Some key implementation examples include:

- Authentication: Users can register and log in with validation.

- Product Browsing: Products can be filtered by category or price.- Cart Functionality: Users can add, remove, and purchase items.

- Admin Controls: Admins can create or update products and manage orders.

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
46      app.post('/login', async (req, res) => {
47          const { email, password } = req.body;
48          try {
49              const user = await User.findOne({ email });
50              if (!user) {
51                  return res.status(401).json({ message: 'Invalid email or password' });
52              }
53              const isMatch = await bcrypt.compare(password, user.password);
54              if (!isMatch) {
55                  return res.status(401).json({ message: 'Invalid email or password' });
56              } else{
57                  return res.json(user);
58              }
59          } catch (error) {
60            console.log(error);
61            return res.status(500).json({ message: 'Server Error' });
62          }
63      });
```

- **Frontend:**

Login:

```
JS GeneralContext.js U ×
client > src > context > JS GeneralContext.js > GeneralContextProvider > register > then() callback
46      const login = async () =>{
47        try{
48          const loginInputs = {email, password}
49            await axios.post('http://localhost:6001/login', loginInputs)
50            .then( async (res)=>{
51
52              localStorage.setItem('userId', res.data._id);
53                localStorage.setItem('userType', res.data.usertype);
54                localStorage.setItem('username', res.data.username);
55                localStorage.setItem('email', res.data.email);
56                if(res.data.usertype === 'customer'){
57                    navigate('/');
58                } else if(res.data.usertype === 'admin'){
59                    navigate('/admin');
60                }
61              }).catch((err) =>{
62                alert("login failed!!");
63                console.log(err);
64              });
65            }catch(err){
66              console.log(err);
67            }
68          }
```

Register:

```
GeneralContext.js U ×
client > src > context > GeneralContext.js > GeneralContextProvider > logout
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.jsx U ×
client > src > context > GeneralContext.jsx > GeneralContextProvider > login >
const logout = async () =>{

  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.

Fetching products:

```
Products.jsx 2, U ×
client > src > components > Products.jsx > Products > handleCategoryCheckBox
10      const [categories, setCategories] = useState([]);
11      const [products, setProducts] = useState([]);
12      const [visibleProducts, setVisibleProducts] = useState([]);
13
14      useEffect(()=>{
15          fetchData();
16        }, [])
17
18        const fetchData = async() =>{
19
20          await axios.get('http://localhost:6001/fetch-products').then(
21            (response)=>{
22              if(props.category === 'all'){
23                  setProducts(response.data);
24                  setVisibleProducts(response.data);
25              }else{
26                  setProducts(response.data.filter(product=> product.category === props.category));
27                  setVisibleProducts(response.data.filter(product=> product.category === props.category));
28              }
29            }
30          )
31          await axios.get('http://localhost:6001/fetch-categories').then(
32            (response)=>{
33              setCategories(response.data);
34            }
35          )
36        }
```

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

```
JS index.js ×
server > JS index.js > then() callback > app.get('/fetch-banner') callback
100
101         // fetch products
102
103         app.get('/fetch-products', async(req, res)=>{
104             try{
105                 const products = await Product.find();
106                 res.json(products);
107
108             }catch(err){
109                 res.status(500).json({ message: 'Error occured' });
110             }
111         })
```

Filtering products:

Products.jsx 2, U

client > src > components > Products.jsx > Products > useEffect() callback

```
38      const [sortFilter, setSortFilter] = useState('popularity');
39      const [categoryFilter, setCategoryFilter] = useState([]);
40      const [genderFilter, setGenderFilter] = useState([]);
41
42      const handleCategoryCheckBox = (e) =>{
43        const value = e.target.value;
44        if(e.target.checked){
45            setCategoryFilter([...categoryFilter, value]);
46        }else{
47            setCategoryFilter(categoryFilter.filter(size=> size !== value));
48        }
49      }
50
51      const handleGenderCheckBox = (e) =>{
52        const value = e.target.value;
53        if(e.target.checked){
54            setGenderFilter([...genderFilter, value]);
55        }else{
56            setGenderFilter(genderFilter.filter(size=> size !== value));
57        }
58      }
59
60      const handleSortFilterChange = (e) =>{
61        const value = e.target.value;
62        setSortFilter(value);
63        if(value === 'low-price'){
64            setVisibleProducts(visibleProducts.sort((a,b)=>  a.price - b.price))
65        } else if (value === 'high-price'){
66            setVisibleProducts(visibleProducts.sort((a,b)=>  b.price - a.price))
67        }else if (value === 'discount'){
68            setVisibleProducts(visibleProducts.sort((a,b)=>  b.discount - a.discount))
69        }
70      }
71
72      useEffect(()=>{
73
74          if (categoryFilter.length > 0 && genderFilter.length > 0){
75              setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category) && genderFilter.includes(product.gender) ));
76          }else if(categoryFilter.length === 0 && genderFilter.length > 0){
77              setVisibleProducts(products.filter(product=> genderFilter.includes(product.gender) ));
78          } else if(categoryFilter.length > 0 && genderFilter.length === 0){
79              setVisibleProducts(products.filter(product=> categoryFilter.includes(product.category)));
80          }else{
81              setVisibleProducts(products);
82          }
83
84      }, [categoryFilter, genderFilter])
85
86
```

Add product to cart:

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

```
IndividualProduct.jsx 2, U ×
dient > src > pages > customer > IndividualProduct.jsx > IndividualProduct
const buyNow = async() =>{
    await axios.post('http://localhost:6001/buy-product',{userId, name, email, mobile, address,
                                pincode, title: productName, description: productDescription,
                                mainImg: productMainImg, size, quantity: productQuantity, price: productPrice,
                                discount: productDiscount, paymentMethod: paymentMethod, orderDate: new Date()}).then(
        (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: productMainImg, size, quantity: productQuantity, price: productPrice,
                                    discount: productDiscount}).then(
        (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:

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

```
index.js
server > 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

    // 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.description,
                            mainImg: item.mainImg, size:item.size, quantity: item.quantity,
                            price: item.price, discount: item.discount, paymentMethod, orderDate})
                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.

- Frontend:

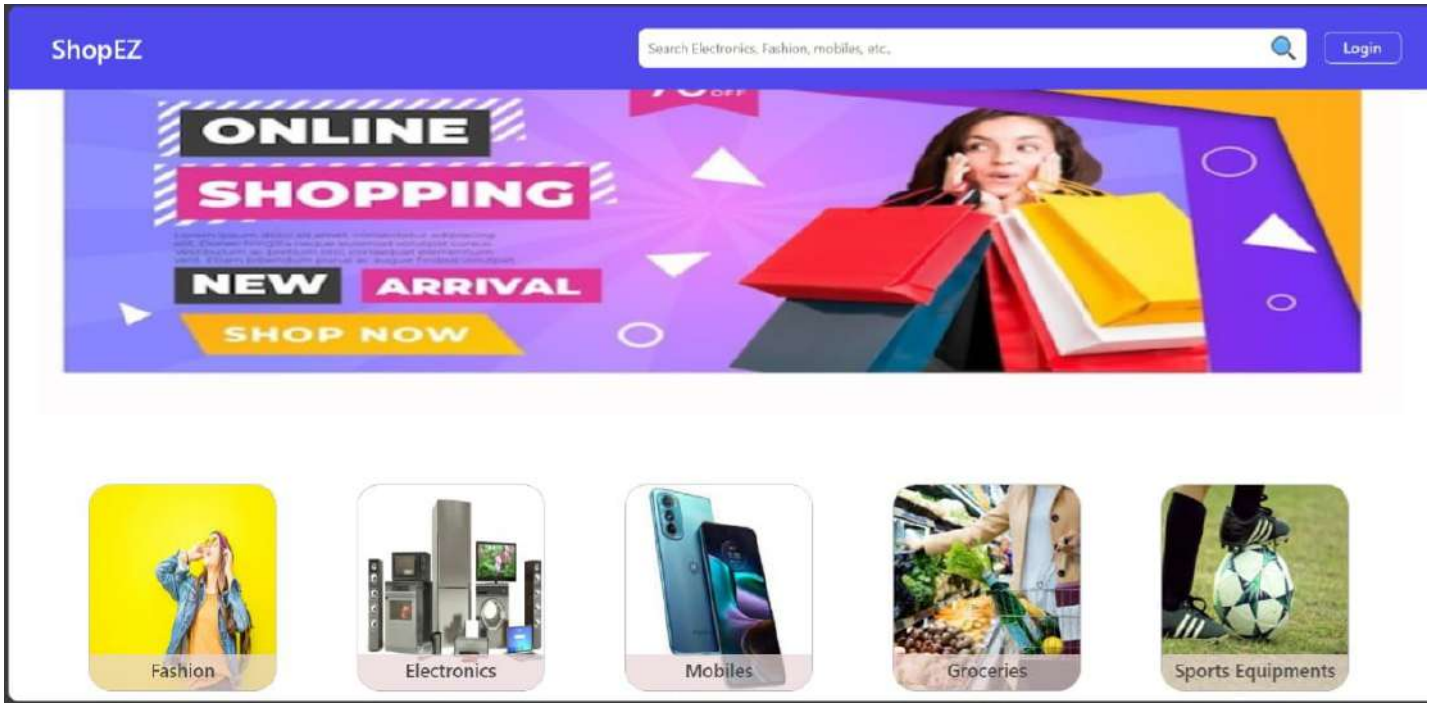
```
NewProduct.jsx U
client > src > pages > admin > NewProduct.jsx > NewProduct

  const handleNewProduct = async() =>{
    await axios.post('http://localhost:6001/add-new-product', {productName, productDescription, productMainImg,
                productCarousel: [productCarouselImg1, productCarouselImg2, productCarouselImg3], productSizes,
                productGender, productCategory, productNewCategory, productPrice, productDiscount}).then(
      (response)=>{
        alert("product added");
        setProductName('');
        setProductDescription('');
        setProductMainImg('');
        setProductCarouselImg1('');
        setProductCarouselImg2('');
        setProductCarouselImg3('');
        setProductSizes([]);
        setProductGender('');
        setProductCategory('');
        setProductNewCategory('');
        setProductPrice(0);
        setProductDiscount(0);

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

Backend:

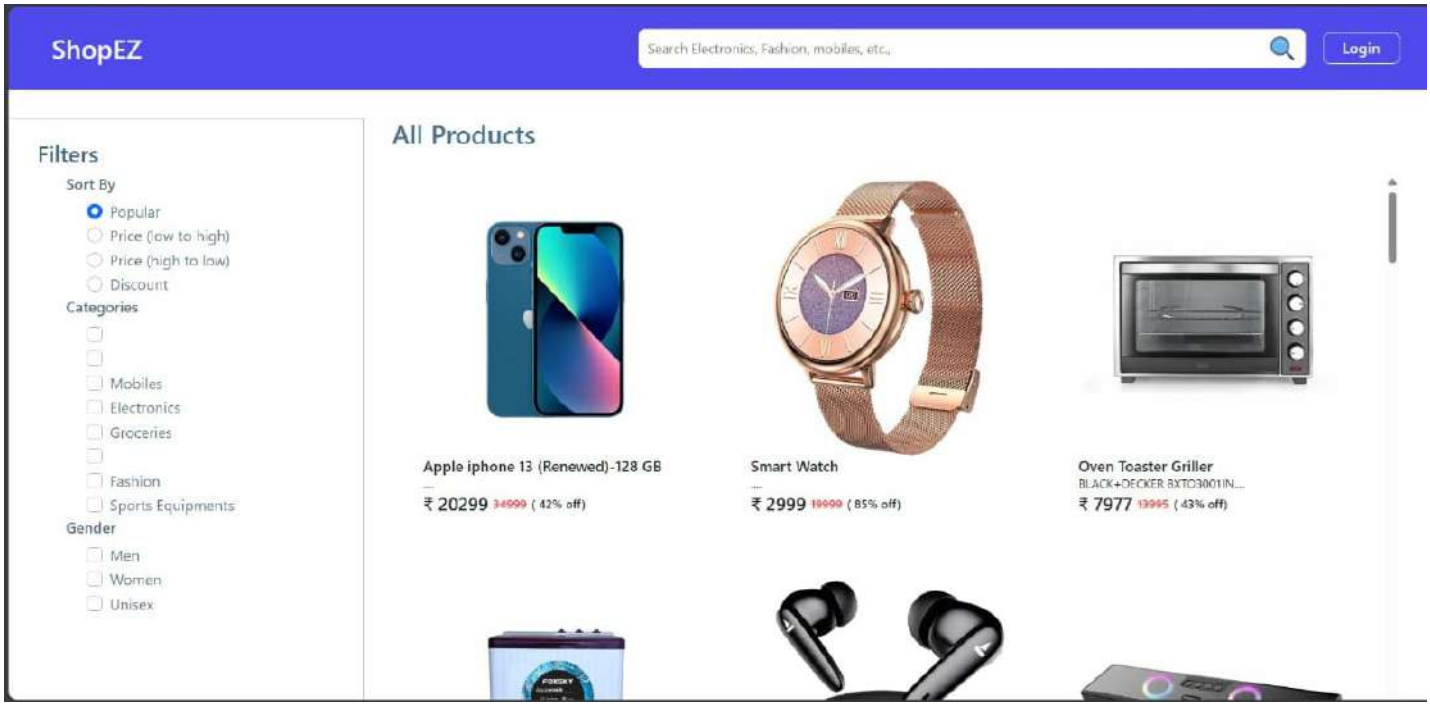
```
JS index.js ×
server > JS index.js > then() callback > app.put('/update-product/:id') callback
143
144     // Add new product
145
146     app.post('/add-new-product', async(req, res)=>{
147         const {productName, productDescription, productMainImg, productCarousel,
148                 productSizes, productGender, productCategory, productNewCategory,
149                 productPrice, productDiscount} = req.body;
150         try{
151             if(productCategory === 'new category'){
152                 const admin = await Admin.findOne();
153                 admin.categories.push(productNewCategory);
154                 await admin.save();
155                 const newProduct = new Product({title: productName, description: productDescription,
156                         mainImg: productMainImg, carousel: productCarousel, category: productNewCategory,
157                         sizes: productSizes, gender: productGender, price: productPrice, discount: productDiscount});
158                 await newProduct.save();
159             } else{
160                 const newProduct = new Product({title: productName, description: productDescription,
161                         mainImg: productMainImg, carousel: productCarousel, category: productCategory,
162                         sizes: productSizes, gender: productGender, price: productPrice, discount: productDiscount});
163                 await newProduct.save();
164             }
165             res.json({message: "product added!!"});
166         }catch(err){
167             res.status(500).json({message: "Error occured"});
168         }
169     })
```

Demo and References:

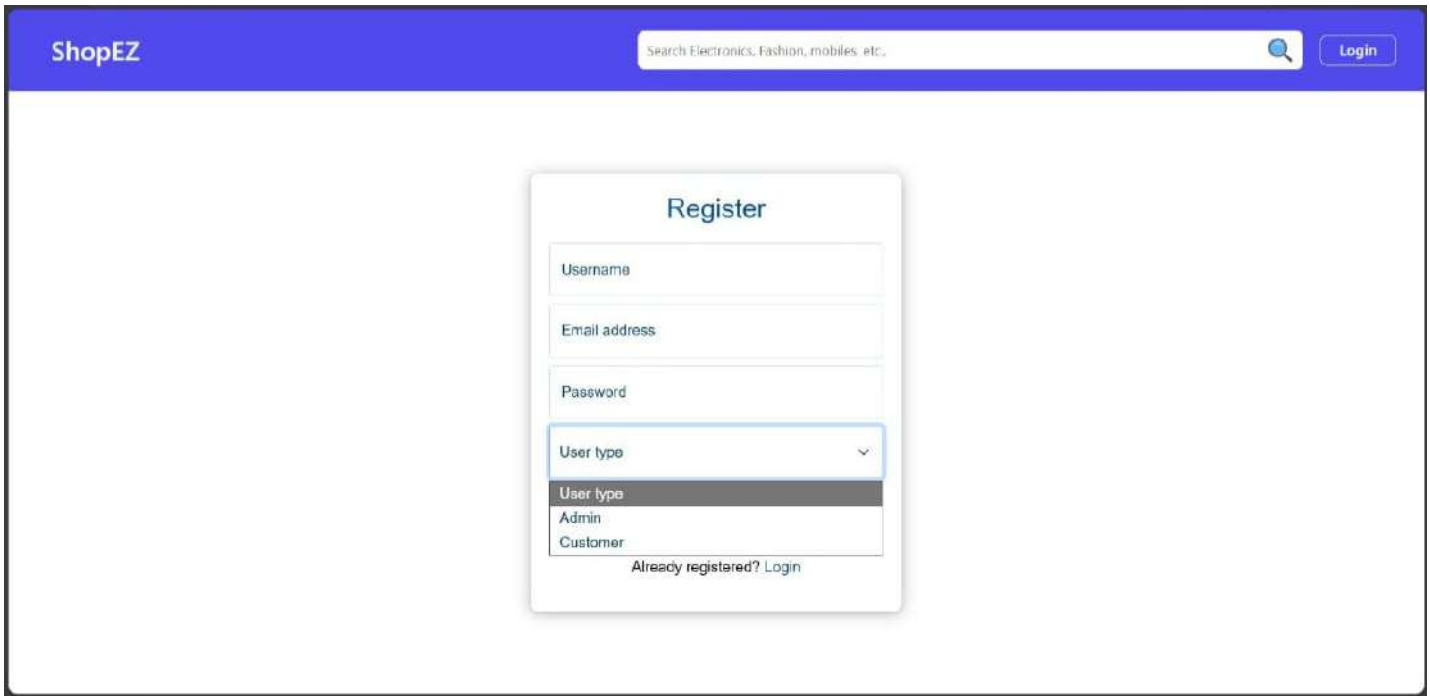
Landing page:



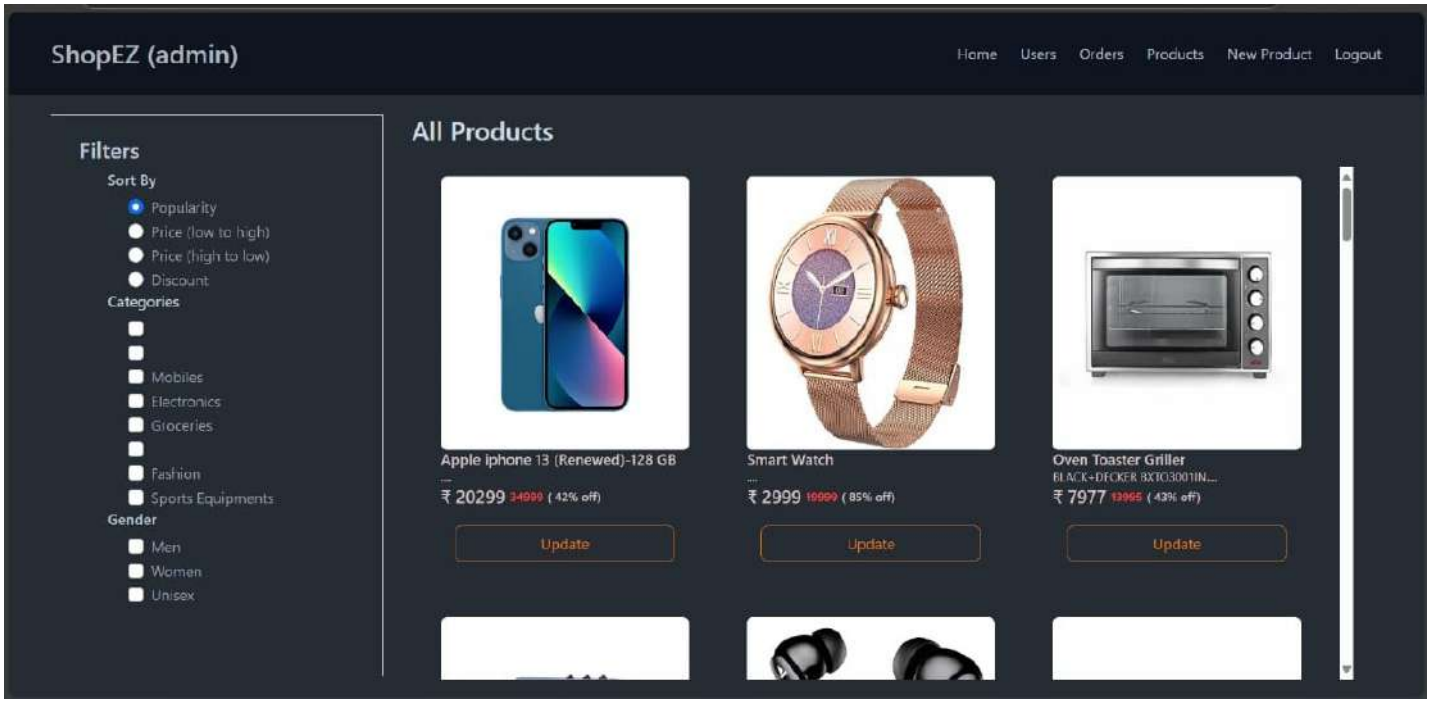
- **Products:**



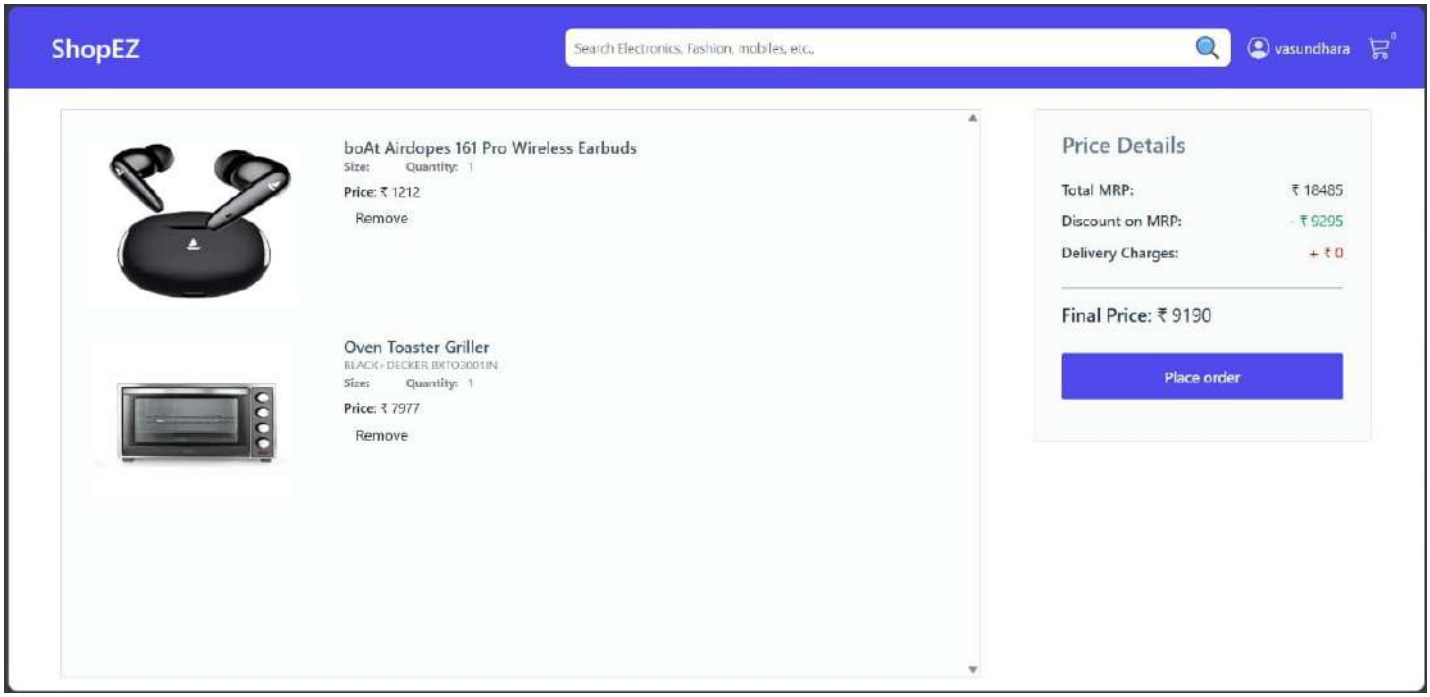
Authentication:



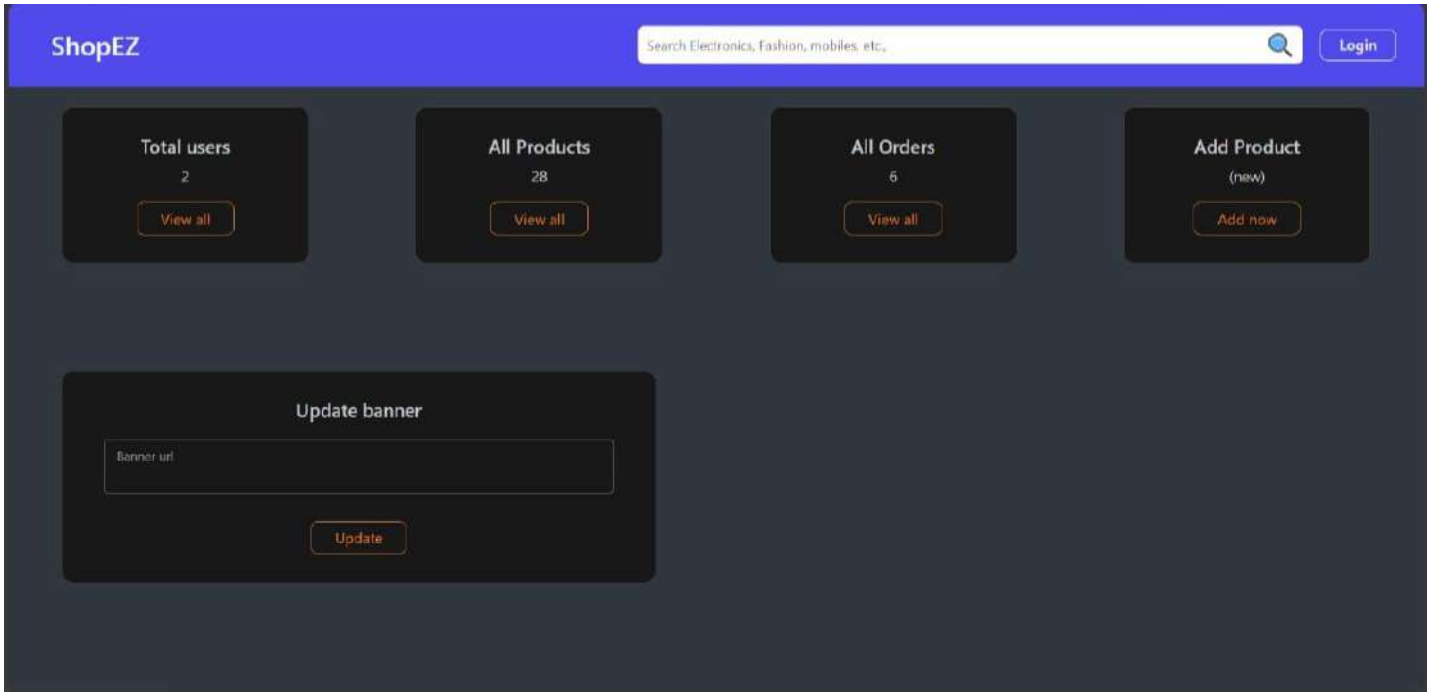
- **User Profile:**



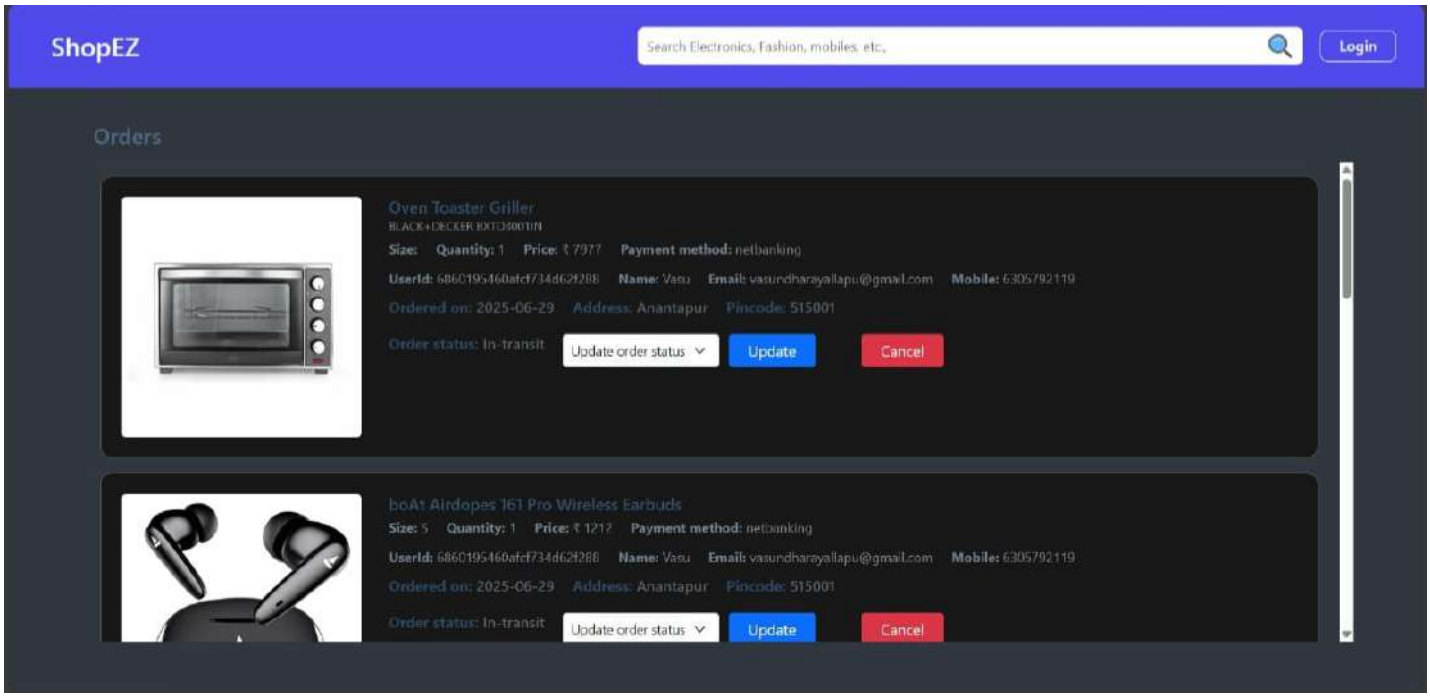
Cart:



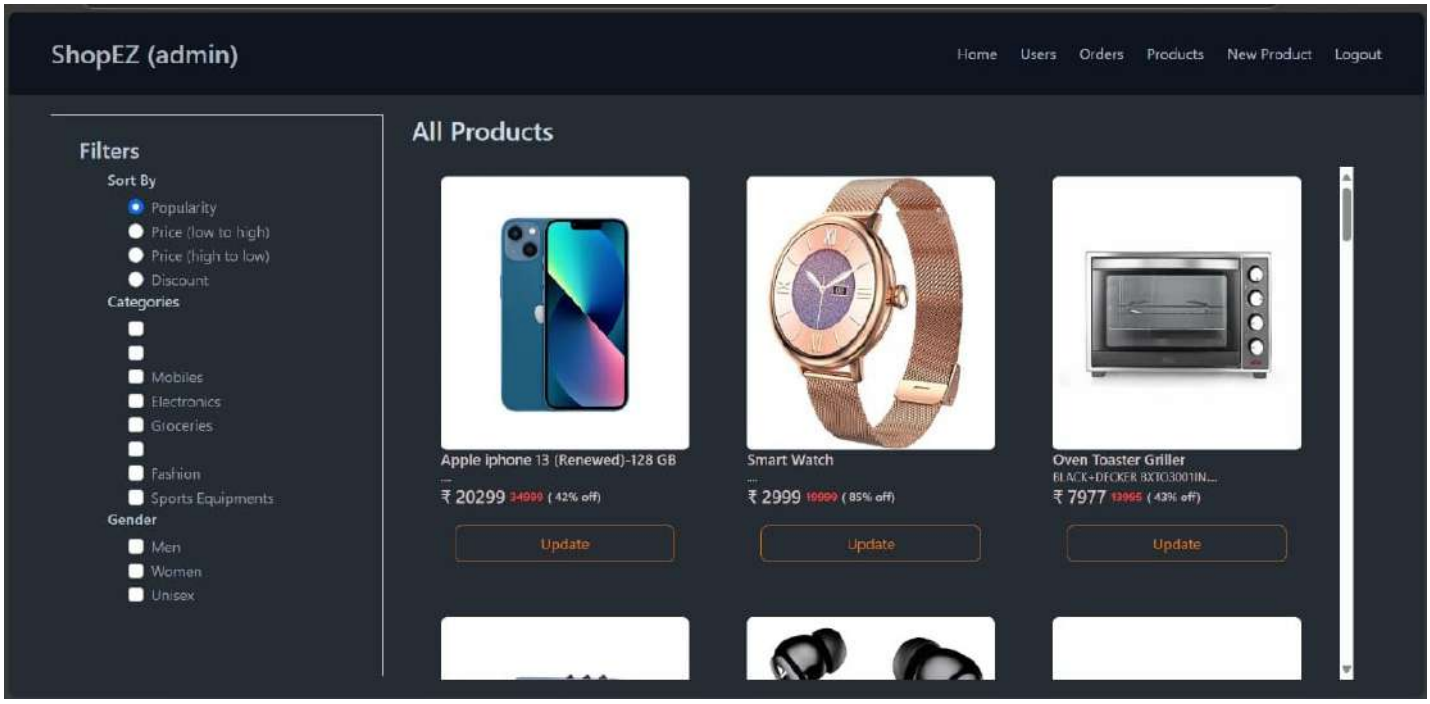
· **Admin dashboard:**



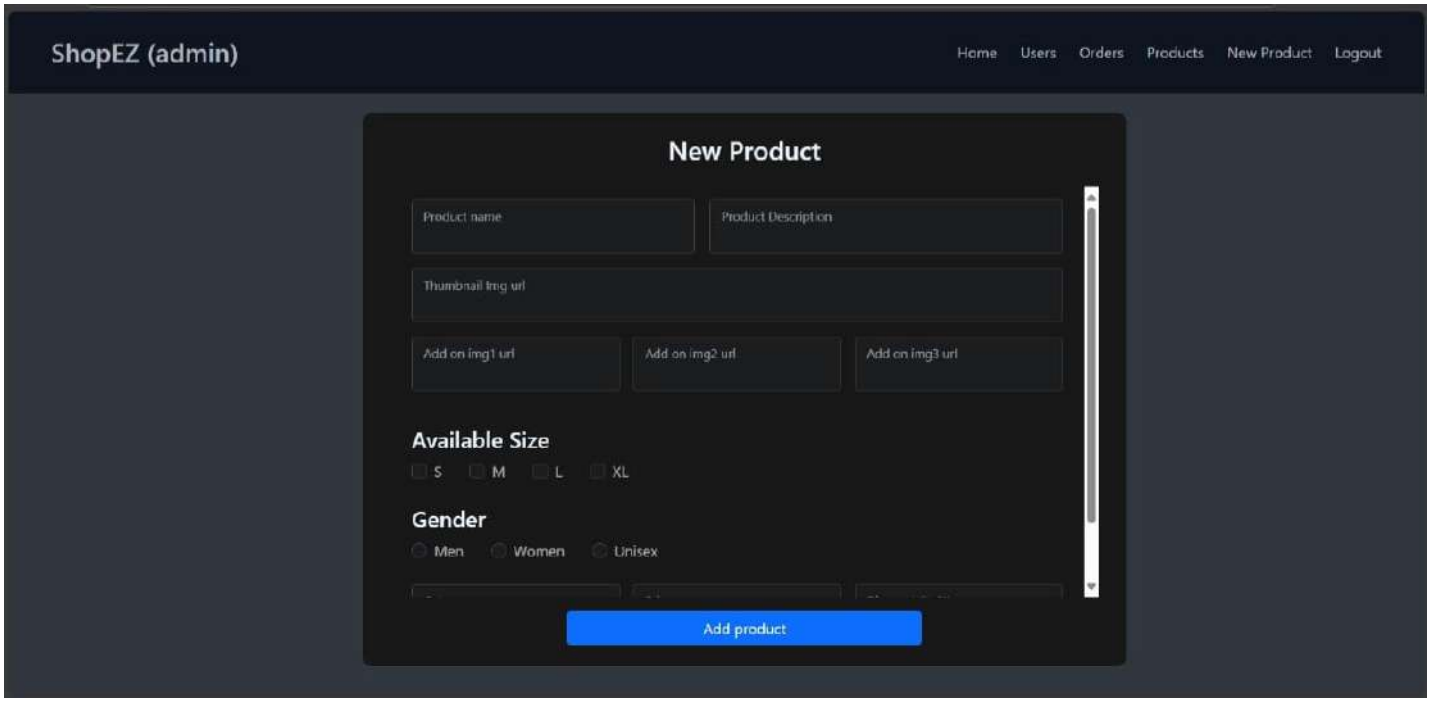
- **All Orders:**



- **All Products:**



· **New Product Page:**



Useful resources to understand and extend the project:

- App Demo: https://drive.google.com/file/d/1SwDamD8onQEdxB47pE1wdGG34cH-ryz4/view?usp=drive_link

Conclusion:

ShopEZ is a robust and scalable e-commerce platform that combines backend functionality with a seamless frontend user experience. It empowers users with convenience, while providing sellers with a platform to

manage their business effectively through analytics and management tools. It is a well-suited solution for launching an online marketplace.