Shopsmart: your digital grocery store experience

INTRODUCTION

ShopSmart is your go-to destinationfor convenient, fast, and reliable grocery shopping. With an intuitive interface and a wide range of fresh produce, pantry staples, and household essentials, getting your groceries has never been easier. Effortlessly explore product details, check real-time availability, read verified customer reviews, and take advantage of exclusive discounts.

Enjoy a secure, quick checkout and receive instant order confirmations. For local vendors and suppliers, our seller dashboard offers streamlined order management and powerful analytics to fuel business growth. Experience the future of grocery shopping with ShopSmart—where smart choices meet smart shopping.

Key Features

- Effortless Product Discovery
- Seamless Checkout Experience
- Personalized Grocery Recommendations
- Efficient Order Management for Vendors
- Insightful Analytics to Boost Sales

Scenario: **Raj's Last-Minute Dinner Party**

Raj, a working professional and amateur chef, decides to host a last-minute dinner party for his friends. With little time to spare, he needs to stock up on fresh ingredients, snacks, and drinks —fast.

1. Effortless Product Discovery
Raj opens ShopSmart on his phone and navigates to the "Dinner Essentials" section. He's immediately presented with curated grocery kits, fresh vegetables, meat, and beverages. Using filters like "delivery within 2 hours" and "on sale," he quickly narrows down his choices.

2. Personalized Recommendations
As he browses, Raj sees a section labeled **"Picked for Your Recipes."** Based on his past orders and browsing history, ShopSmart suggests ingredients for popular party recipes like paneer tikka, mocktails, and pasta salad. Grateful for the help, Raj adds everything to his cart.

3. Seamless Checkout Process
With just a few taps, Raj selects his preferred delivery slot and chooses UPI as his payment method. Thanks to ShopSmart's streamlined and secure checkout, he places his order in under a minute.

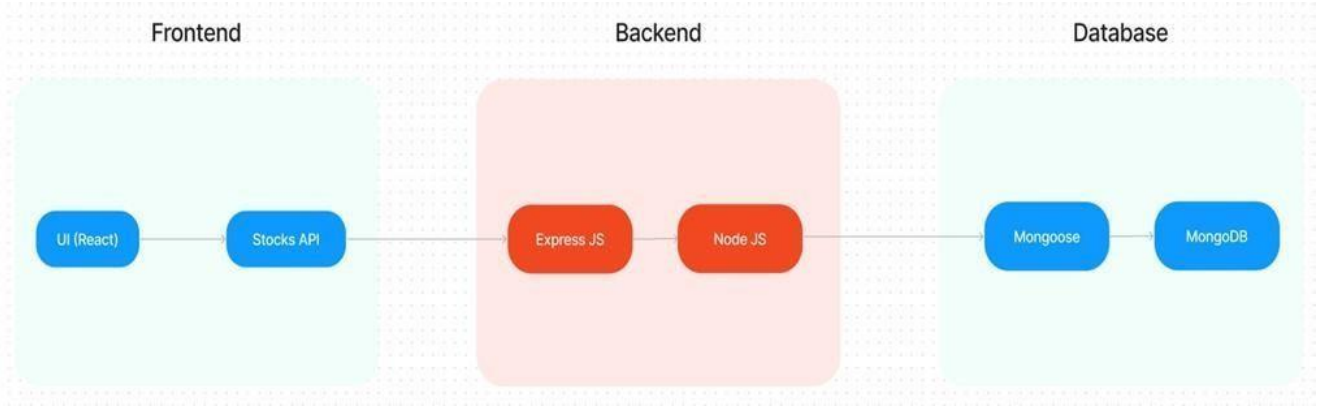
4. Order Confirmation
Within seconds, Raj receives an order confirmation via email and SMS, including the estimated delivery time and live tracking link. With that peace of mind, he starts prepping for the party.

5. Efficient Order Management for Vendors
Meanwhile, the local vendors supplying Raj's groceries receive his order through ShopSmart's seller dashboard. The dashboard highlights priority delivery requests, allowing the sellers to pack and dispatch the items swiftly.

6. A Perfect Dinner Party
Right on time, Raj receives his groceries—fresh, neatly packed, and exactly as ordered. His friends arrive and enjoy a delicious home-cooked meal. Raj feels proud and stress-free, knowing ShopSmart made his last-minute prep look effortless.

In this scenario, ShopSmart stands out as the ideal digital grocery solution for Raj's busy lifestyle. From smart product discovery to fast checkout and reliable delivery, ShopSmart makes grocery shopping efficient, personalized, and stress-free—for both customers and sellers.

TECHNICAL ARCHITECTURE:

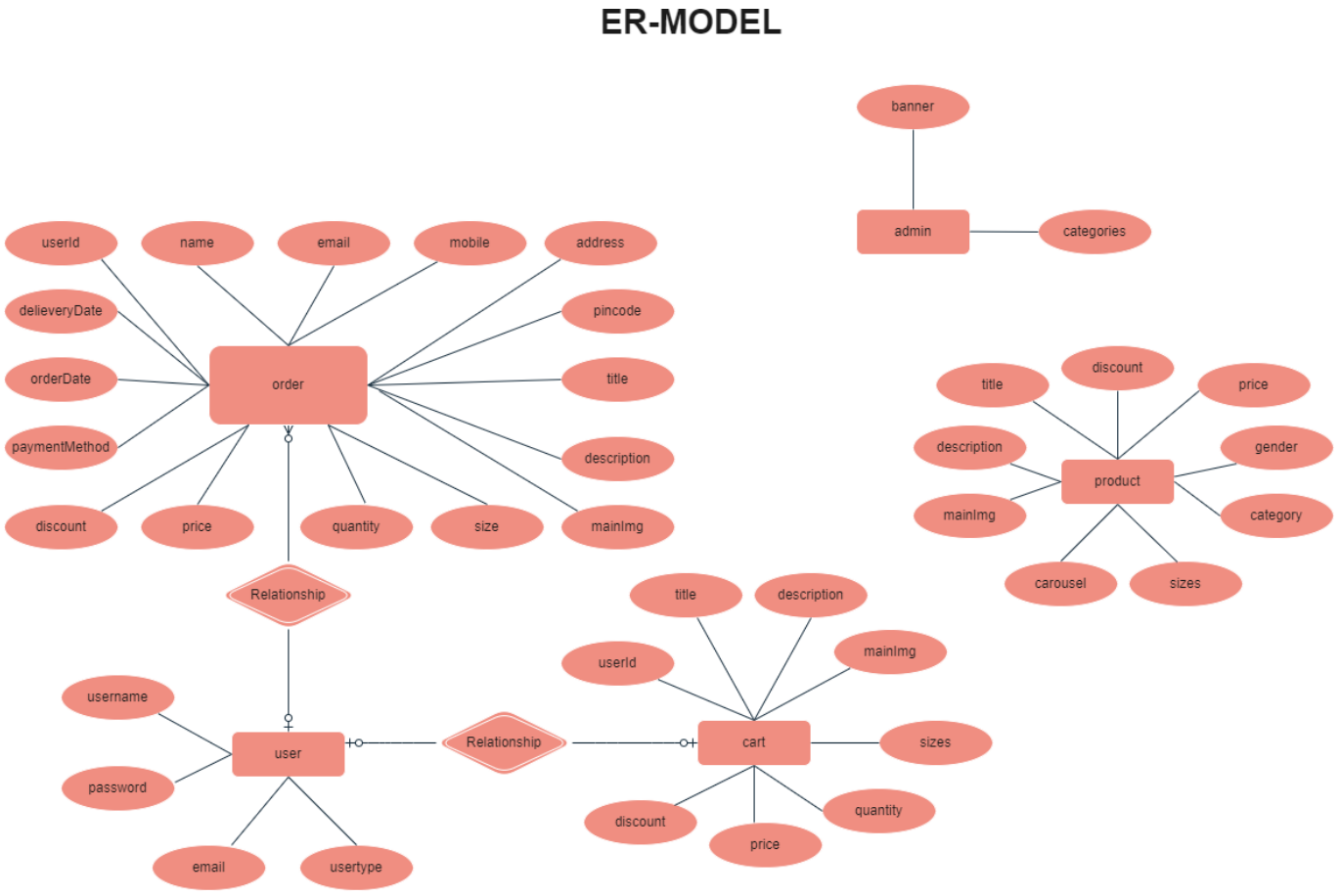


In this architecture diagram:

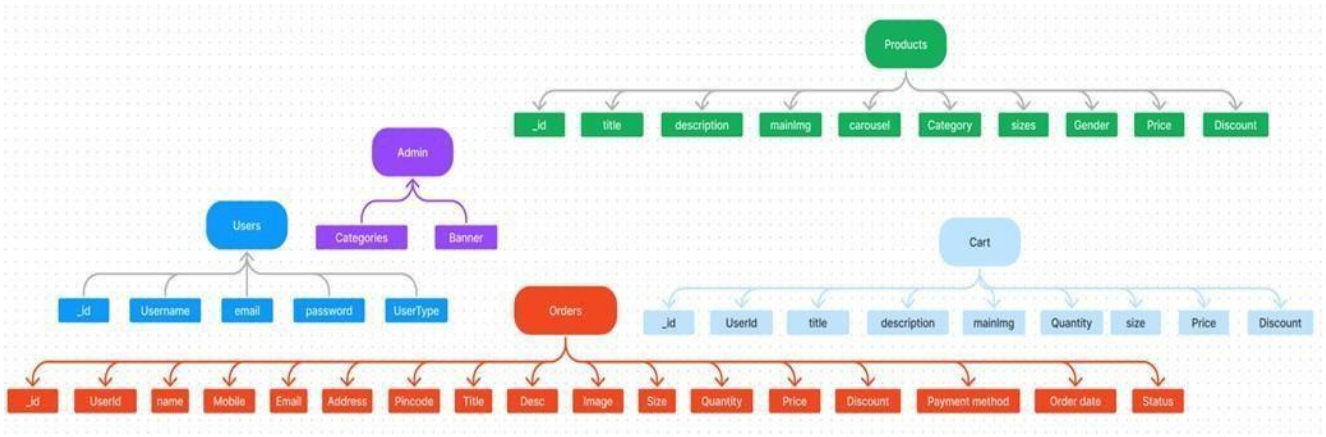
- The frontend is represented by the "Frontend" section, including user interface components such as User Authentication, Cart, Products, Profile, Admin dashboard, etc.,
- The backend is represented by the "Backend" section, consisting of API endpoints for Users, Orders, Products, etc.,It also includes 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.

PREREQUISITES:

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://gitscm.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: https://github.com/harsha-vardhan-reddy-07/shopEZ--e-commerce-MERN

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 & ADMIN FLOW:

1. User Flow:

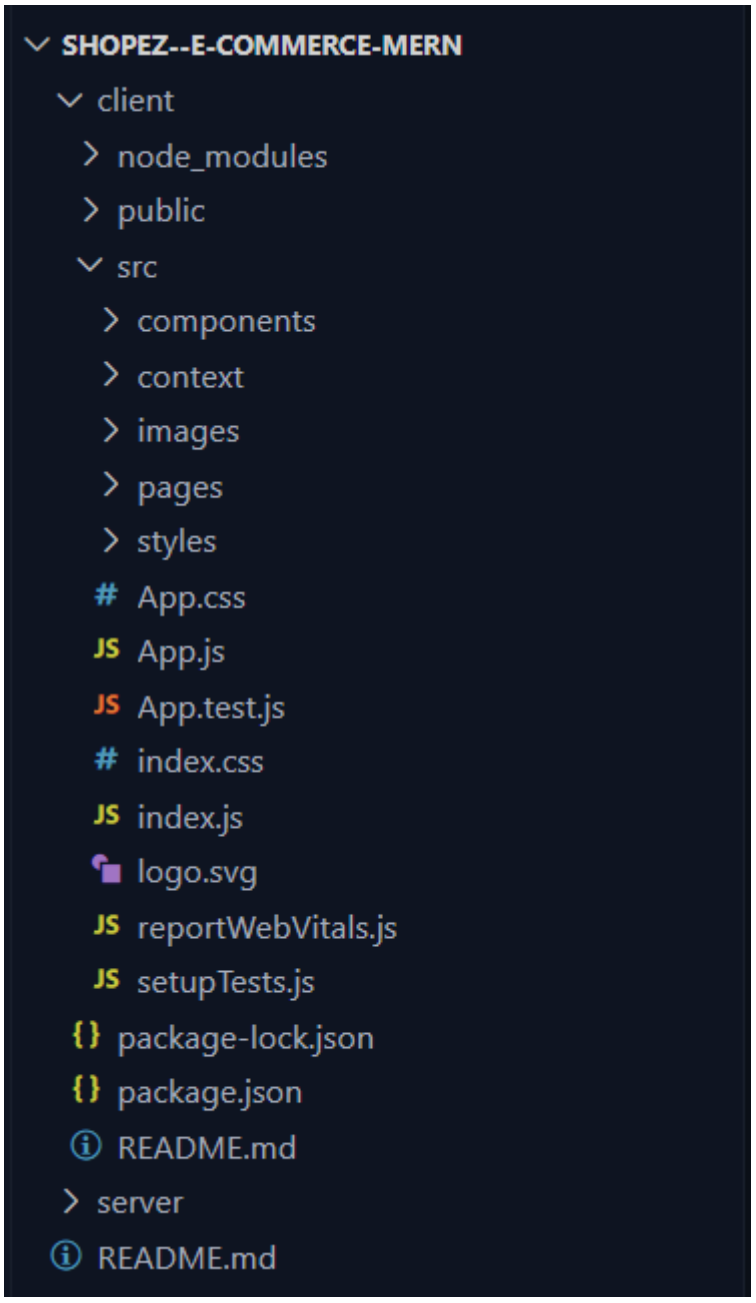
- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform. • Users can add the products they wish to their carts and order.
- They can then proceed by entering address and payment details. • After ordering, they can check them in the profile section.

2. Admin Flow:

- Admins start by logging in with their credentials.

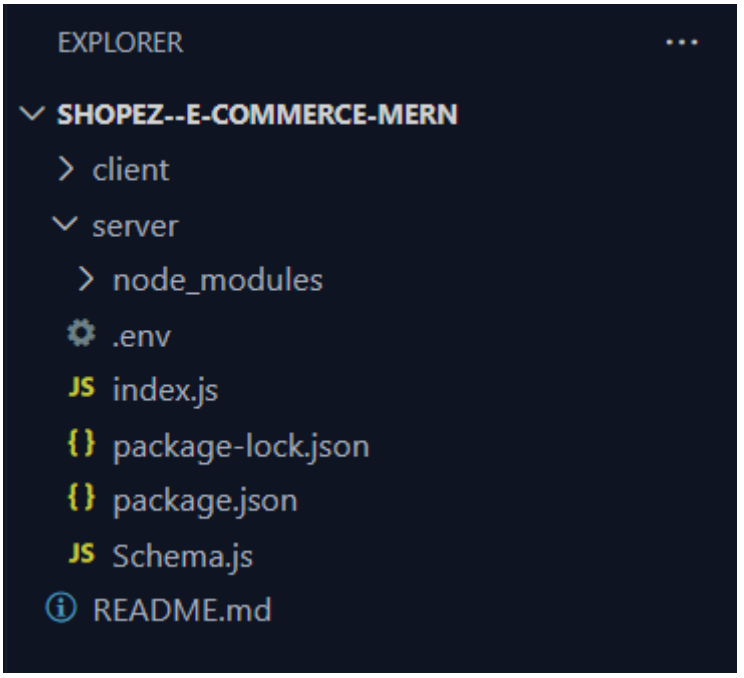
- Once logged in, they are directed to the Admin Dashboard.
- Admins can access the users list, products, orders, etc.,

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.



Project Flow:

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.

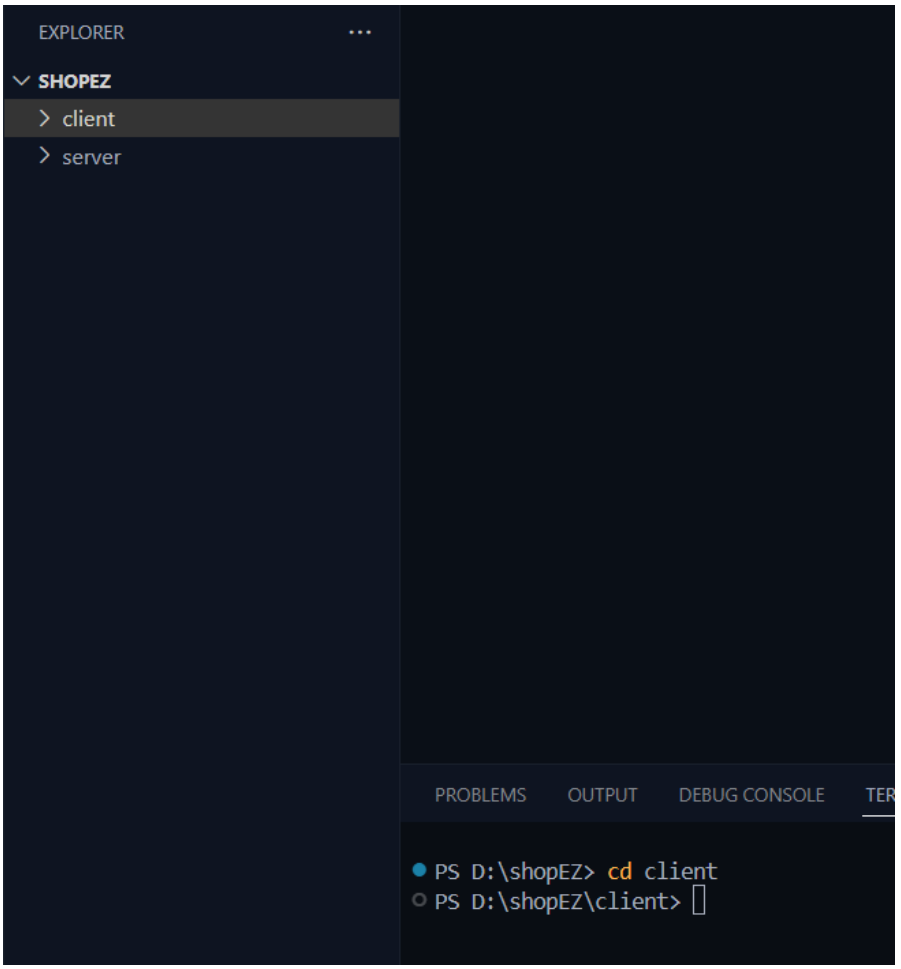
Reference Article: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

2. Create project folders and files:

- Client folders.
- Server folders

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

Referral Image:



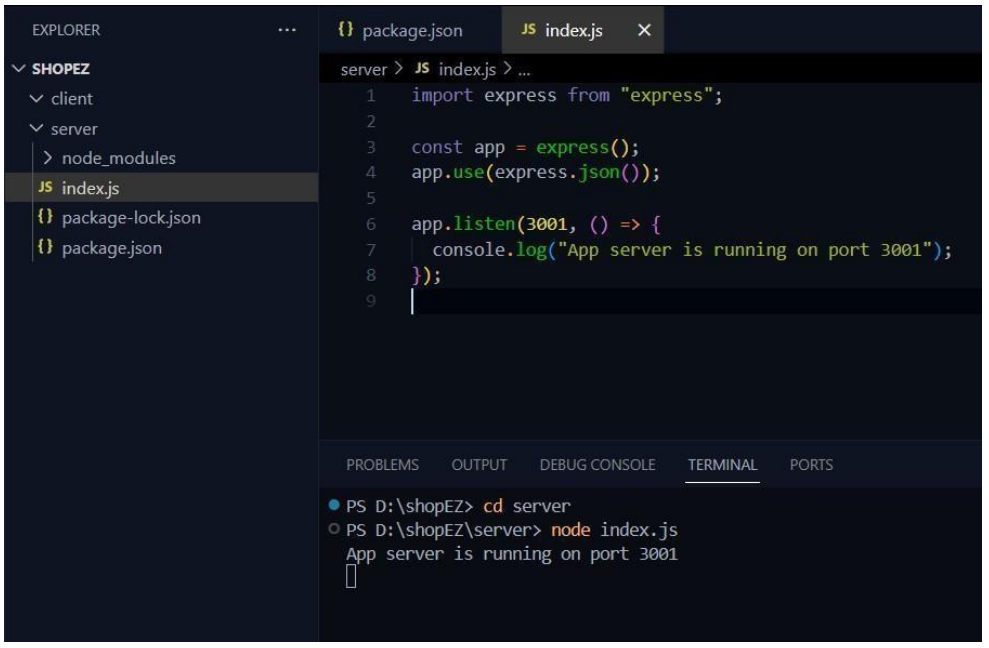
Milestone 2: Backend Development:

1. Setup express server:

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

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

Reference Image:



Now your express is successfully created.

2. Configure MongoDB:

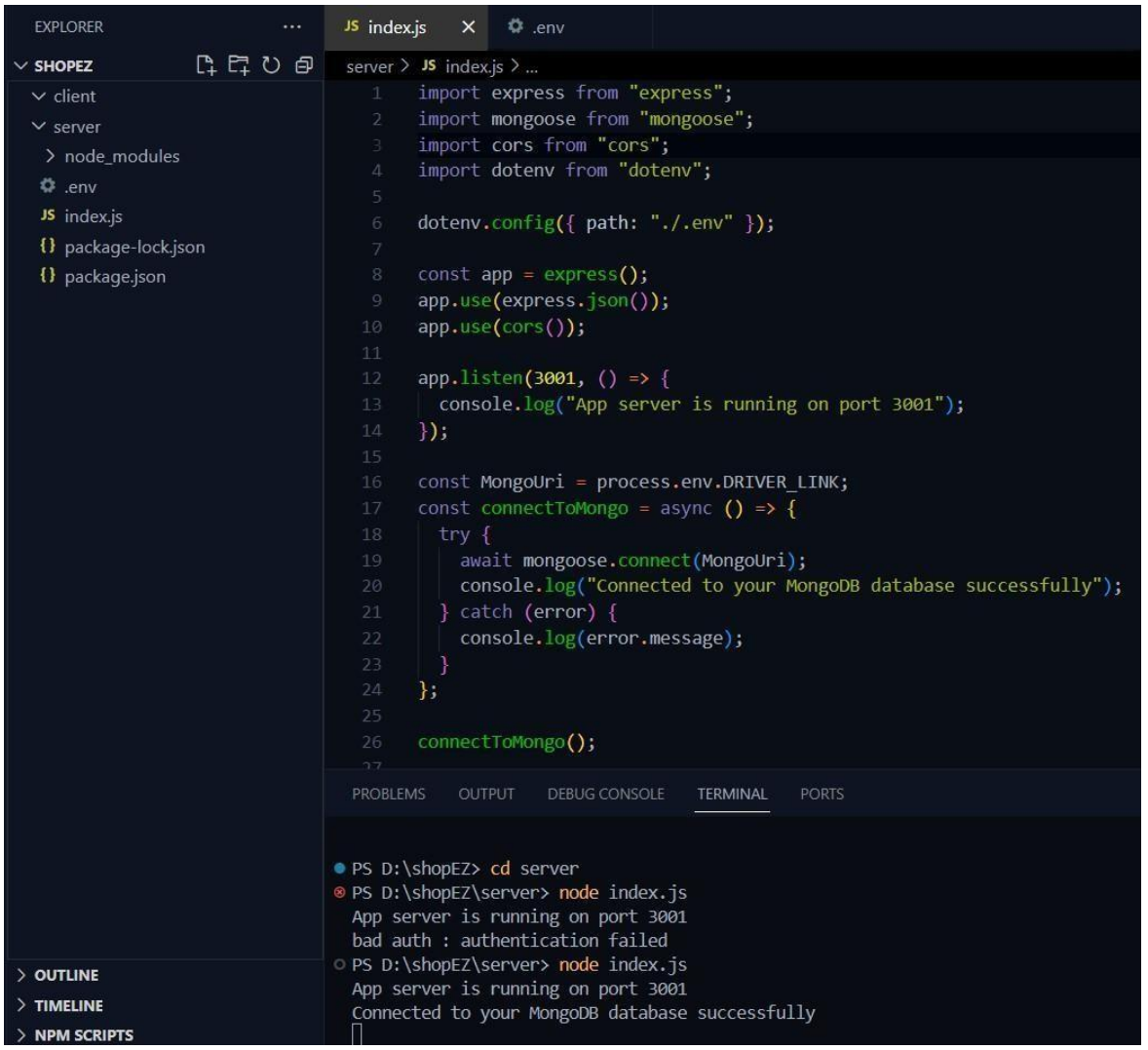
Create database in cloud video link:-
https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view

- 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:



3. Implement API endpoints:

- Implement CRUD operations.
- Test API endpoints.

Backend:

1. Set Up Project Structure:

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

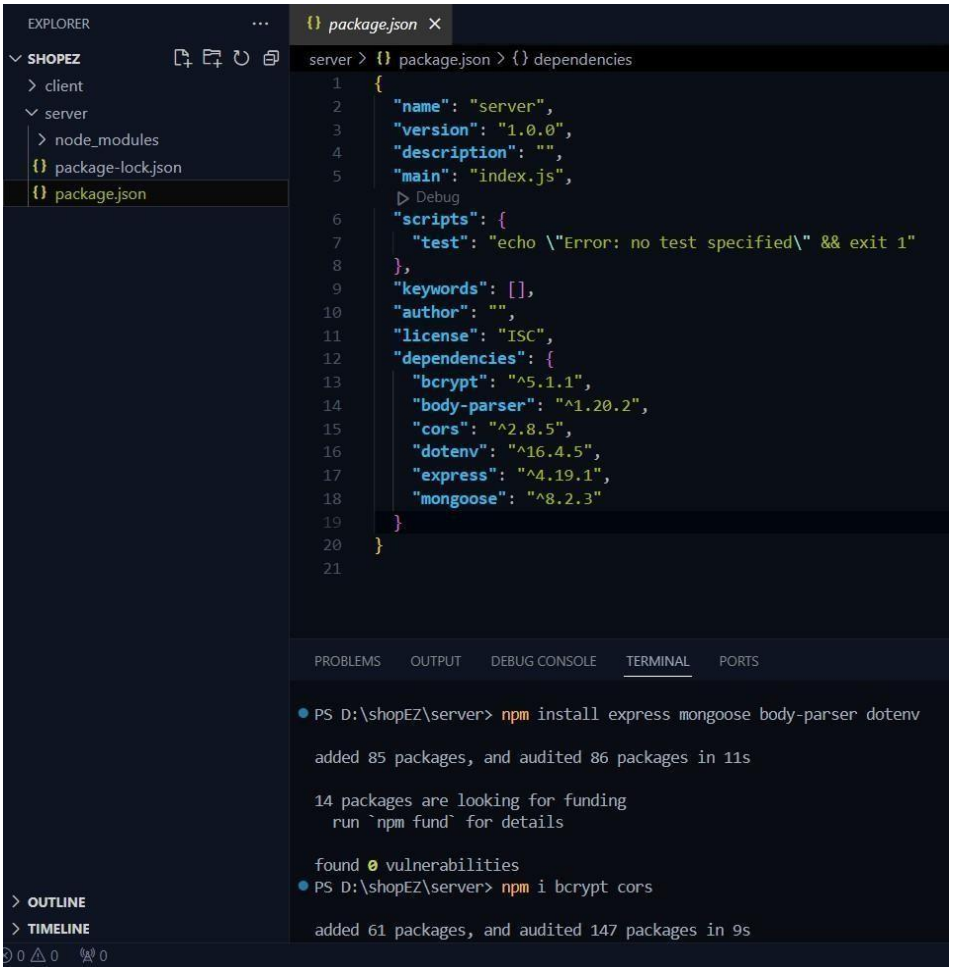
- Install necessary dependencies such as Express.js, Mongoose, and other

required packages.

Reference Video:

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

Reference Image:



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.

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

```
JS Schema.js ×
server > JS Schema.js > productSchema
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}
})
```

```
JS Schema.js ×
server > JS Schema.js > productSchema
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);
```

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.post('/login') callback
app.post('/register', async (req, res) => {
    const { username, email, usertype, password } = req.body;
    try {

        const existingUser = await User.findOne({ email });
        if (existingUser) {
            return res.status(400).json({ message: 'User already exists' });
        }

        const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = new User({
            username, email, usertype, password: hashedPassword
        });
        const userCreated = await newUser.save();
        return res.status(201).json(userCreated);

    } catch (error) {
      console.log(error);
      return res.status(500).json({ message: 'Server Error' });
    }
});
```

```
index.js
server > 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' });
    }
});
```

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-banner') callback

    // 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' });
        }
    })
```

Milestone 3: Web 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.

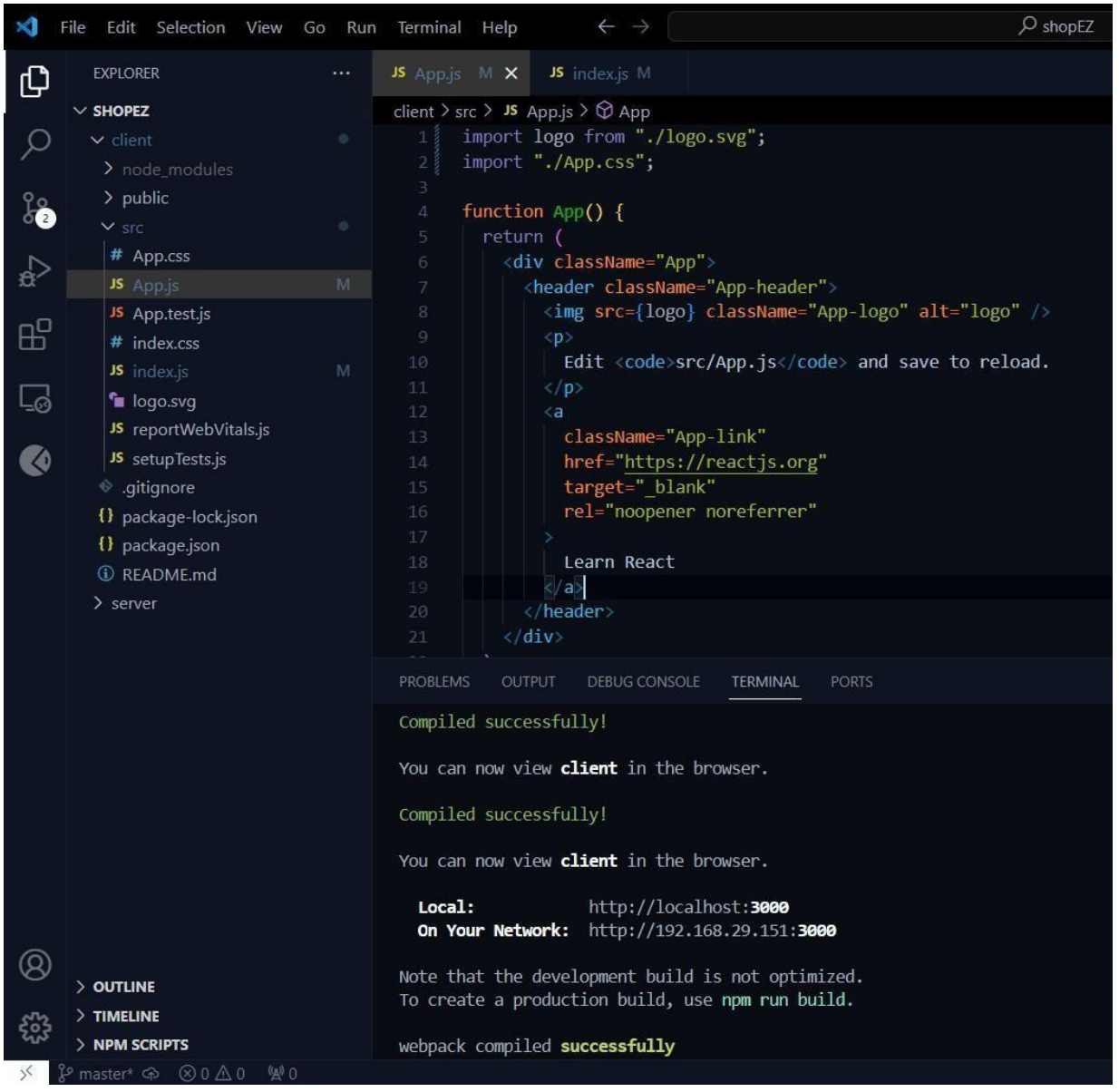
3. Implement frontend logic:
- Integration with API endpoints.
- Implement data binding.

Reference Video Link:

https://drive.google.com/file/d/1EokogagcLMUGiIluwHGYQo65x8GRpDcP/view?usp=sharing

Reference Article Link:
https://www.w3schools.com/react/react_getstarted.asp

Reference Image:



Code Explanation:

- **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:

```
JS GeneralContext.js U ×
client > src > context > JS GeneralContext.js > GeneralContextProvider > logout
71    const inputs = {username, email, usertype, password};
72
73    const register = async () =>{
74      try{
75          await axios.post('http://localhost:6001/register', inputs)
76          .then( async (res)=>{
77              localStorage.setItem('userId', res.data._id);
78              localStorage.setItem('userType', res.data.usertype);
79              localStorage.setItem('username', res.data.username);
80              localStorage.setItem('email', res.data.email);
81
82              if(res.data.usertype === 'customer'){
83                  navigate('/');
84              } else if(res.data.usertype === 'admin'){
85                  navigate('/admin');
86              }
87          }).catch((err) =>{
88              alert("registration failed!!");
89              console.log(err);
90          });
91      }catch(err){
92          console.log(err);
93      }
94    }
95
```

logout:

```
GeneralContext.jsx U ×
client > src > context > GeneralContext.jsx > GeneralContextProvider > login >
73    const logout = async () =>{
74
75      localStorage.clear();
76      for (let key in localStorage) {
77        if (localStorage.hasOwnProperty(key)) {
78          localStorage.removeItem(key);
79        }
80      }
81
82      navigate('/');
83    }
84
85
```

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      }
37
```

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 ×
client > 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.

```
JS index.js ×
server > JS index.js > then() callback
359
360     // Order from cart
361
362     app.post('/place-cart-order', async(req, res)=>{
363         const {userId, name, mobile, email, address, pincode, paymentMethod, orderDate} = req.body;
364         try{
365             const cartItems = await Cart.find({userId});
366             cartItems.map(async (item)=>{
367
368                 const newOrder = new Orders({userId, name, email, mobile, address,
369                             pincode, title: item.title, description: item.description,
370                             mainImg: item.mainImg, size:item.size, quantity: item.quantity,
371                             price: item.price, discount: item.discount, paymentMethod, orderDate})
372                 await newOrder.save();
373                 await Cart.deleteOne({_id: item._id})
374             })
375             res.json({message: 'Order placed'});
376         }catch(err){
377             res.status(500).json({message: "Error occured"});
378         }
379     })
380
```

Add new product:

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

o Frontend**:**

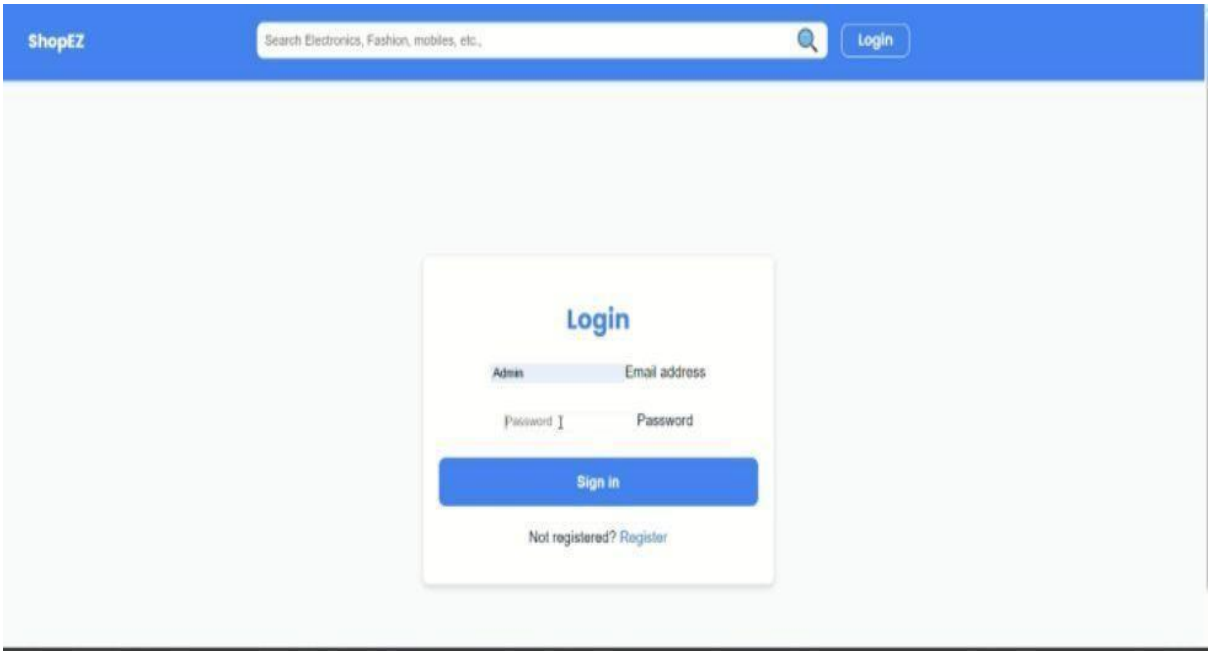
```
NewProduct.jsx U ×
client > src > pages > admin > NewProduct.jsx > NewProduct
47
48     const handleNewProduct = async() =>{
49       await axios.post('http://localhost:6001/add-new-product', {productName, productDescription, productMainImg,
50                         productCarousel: [productCarouselImg1, productCarouselImg2, productCarouselImg3], productSizes,
51                         productGender, productCategory, productNewCategory, productPrice, productDiscount}).then(
52         (response)=>{
53           alert("product added");
54           setProductName('');
55           setProductDescription('');
56           setProductMainImg('');
57           setProductCarouselImg1('');
58           setProductCarouselImg2('');
59           setProductCarouselImg3('');
60           setProductSizes([]);
61           setProductGender('');
62           setProductCategory('');
63           setProductNewCategory('');
64           setProductPrice(0);
65           setProductDiscount(0);
66
67           navigate('/all-products');
68         }
69       )
70     }
```

o 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     })
```

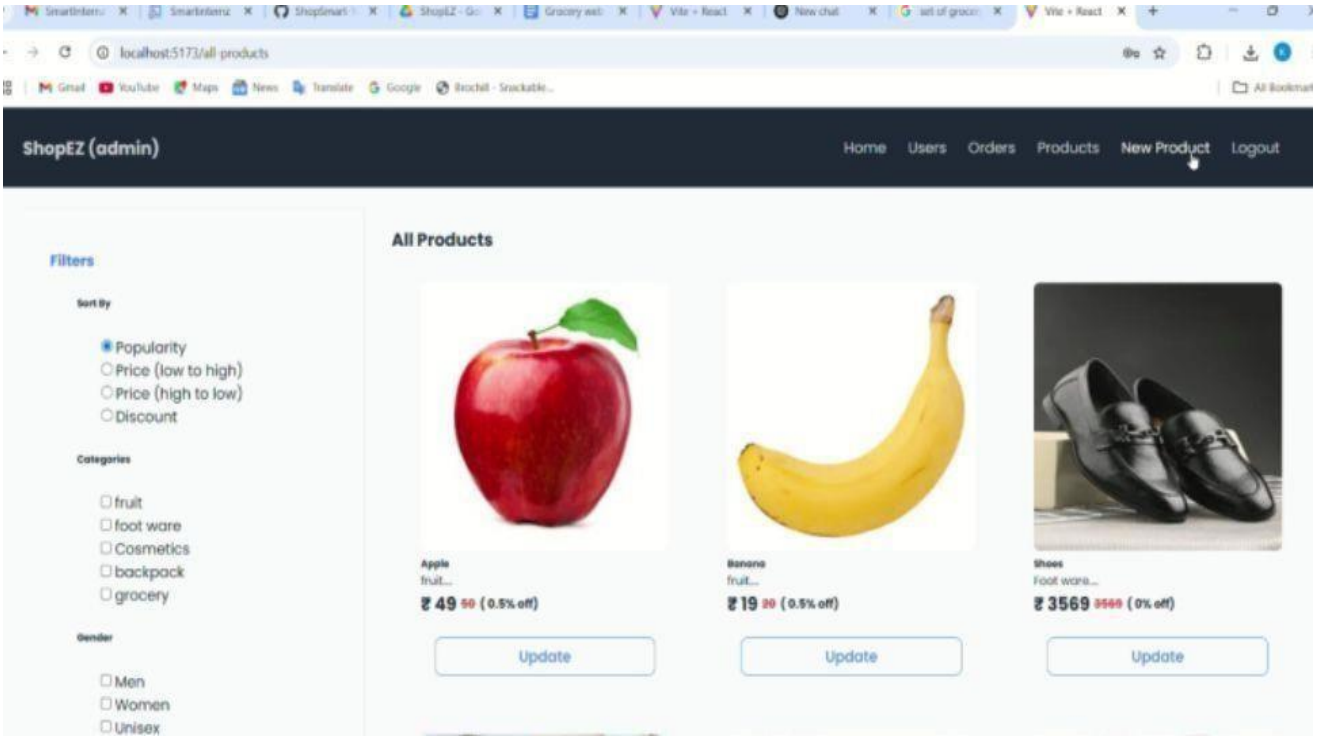
Demo UI images:

- **Login page:**

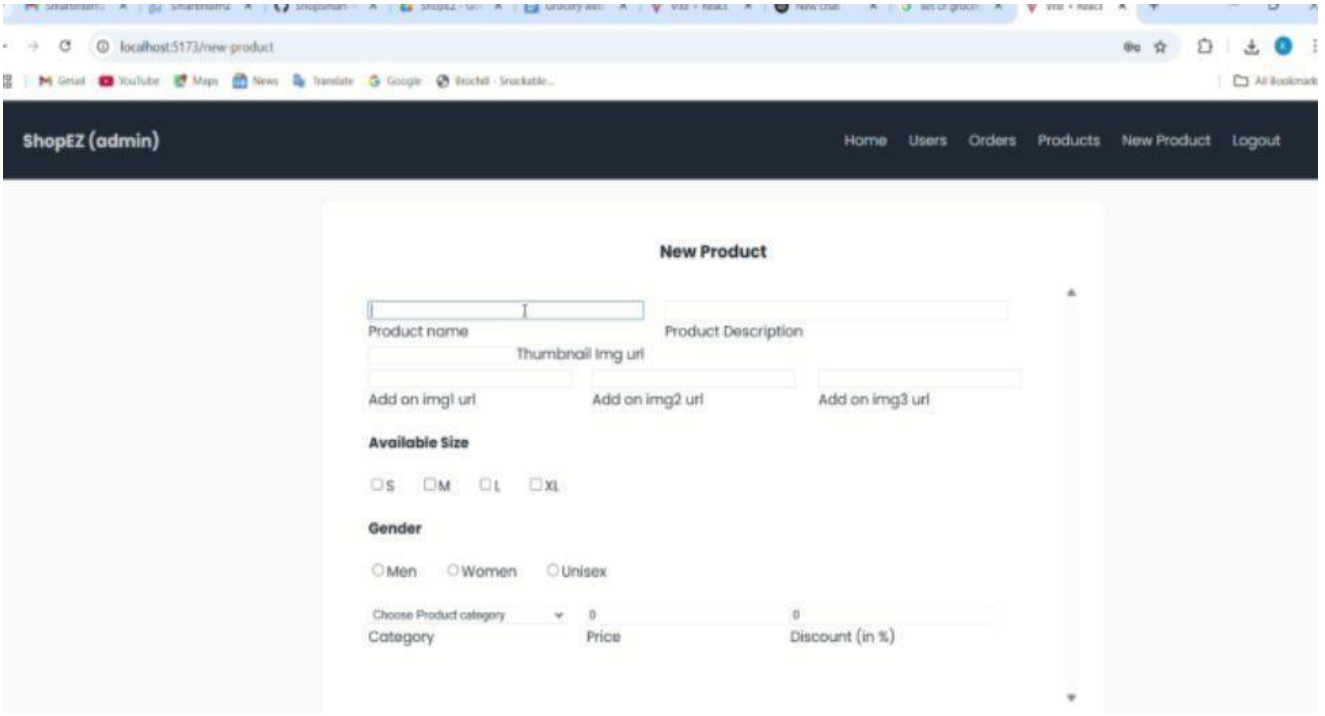


.

Products:



· **New product:**



For any further doubts or help, please consider the GitHub repo

https://github.com/sanjay8099/shopsmart-your-digital-grocery-store-experience

The demo of the app is available at:
https://drive.google.com/file/d/1Qye9ikhVYpQ0Sn3cQ-iGhJw1fDC4G43V/view?usp=sharing