MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE

(An Autonomous Institution) Kodambakkam,Chennai-600024.

MERN STACK POWERED BY MONGODB

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

TOPIC:SHOPEZ:E-COMMERCEAPPLICATION

CATEGORY:FULL STACK DEVELOPMENT– MERN

A PROJECT REPORT

Submitted by

TEAM	NAME	REG.NO
TEAM LEADER	G.V.MADESHWARAN	311521243030
TEAM MATE 2	N.RAMYA	311521243039
TEAM MATE 3	S.NIVETHA	311521243035
TEAM MATE 4	J.M.ASWIN PRIYADHARSAN	311521243007

SHOPEZ:E-COMMERCEAPPLICATION

CATEGORY:FULL STACK DEVELOPMENT-MERN ABSTRACT

ShopEZ is a modern e-commerce platform built using the MERN stack (MongoDB, Express.js, React.js, and Node.js) to provide a seamless, personalized, and efficient online shopping experience. Designed to cater to both customers and sellers, ShopEZ simplifies the entire process of discovering, purchasing, and managing products.

For customers, ShopEZ offers effortless product discovery with advanced search and filtering options, personalized recommendations powered by user activity, and a secure, hassle-free checkout process. Sellers benefit from a robust dashboard that provides insightful analytics, efficient order management tools, and real-time notifications to streamline operations. Admins have access to comprehensive tools for platform management and monitoring key metrics.

 The platform integrates scalable frontend and backend architectures, robust APIs, and a well structured database for real-time operations, ensuring reliability and performance. By bridging the gap between user experience and seller productivity, ShopEZ positions itself as a one-stop solution for e-commerce, delivering value to users like Sarah—our featured customer—while empowering businesses to grow.

This project highlights the potential of full-stack development in solving real-world problems and creating impactful digital solutions.

TABLE OF CONTENTS

PROJECT DESCRIPRION

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

- Seamless Checkout Process
- Effortless Product Discovery
- Personalized Shopping Experience
- Efficient Order Management for Sellers
- Insightful Analytics for Business Growth

## Scenario: Sarah's Birthday Gift

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

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

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

3.Seamless Checkout Process: With the bracelet in her cart, Sarah proceeds to checkout. She enters Emily's address as the shipping destination and selects her preferred payment method. Thanks to ShopEZ's secure and efficient checkout process, Sarah completes the transaction in just a few clicks.

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

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

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

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

TECHNICAL ARCHITECTURE:
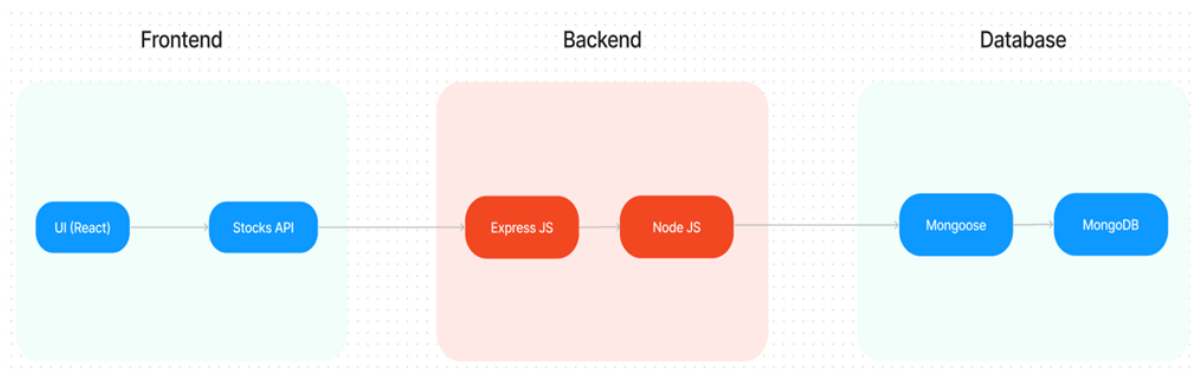

In this architecture diagram:

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

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

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

ER DIAGRAM

ER-MODEL
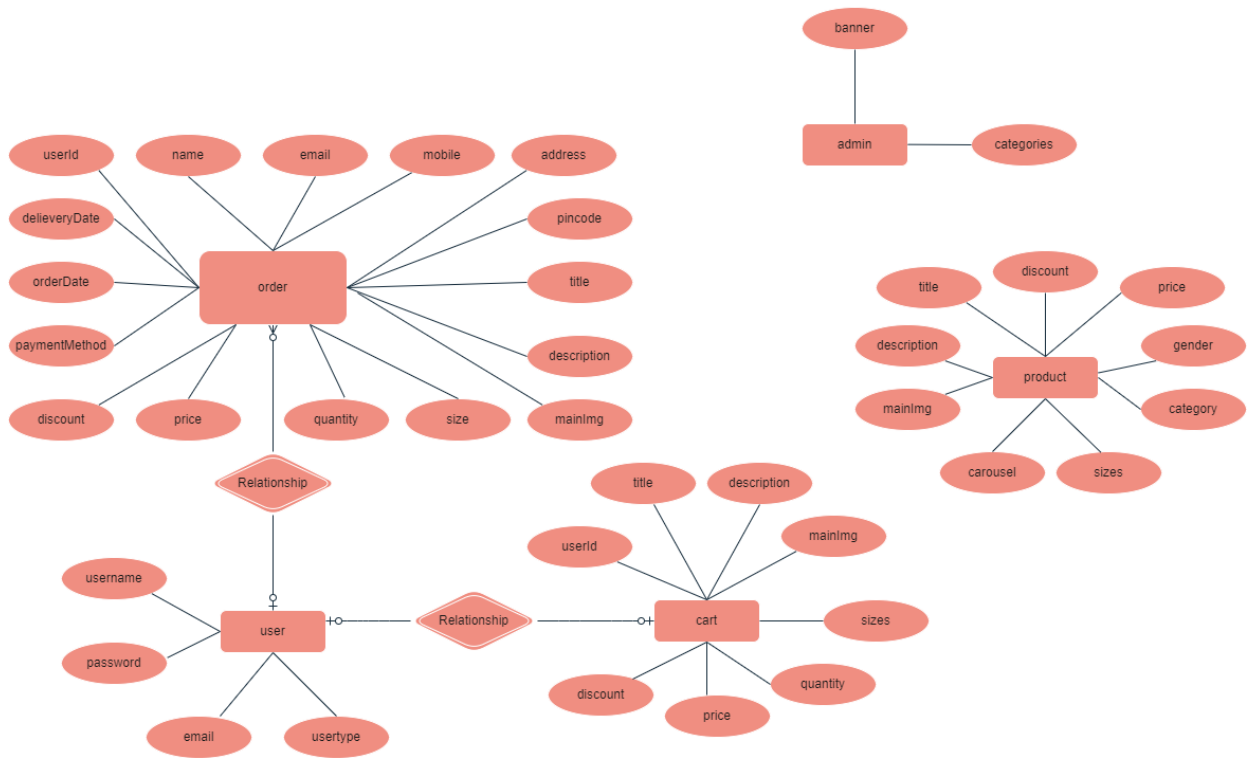
The Database section represents the database that stores collections for Users, Admin, Cart, Orders and products.
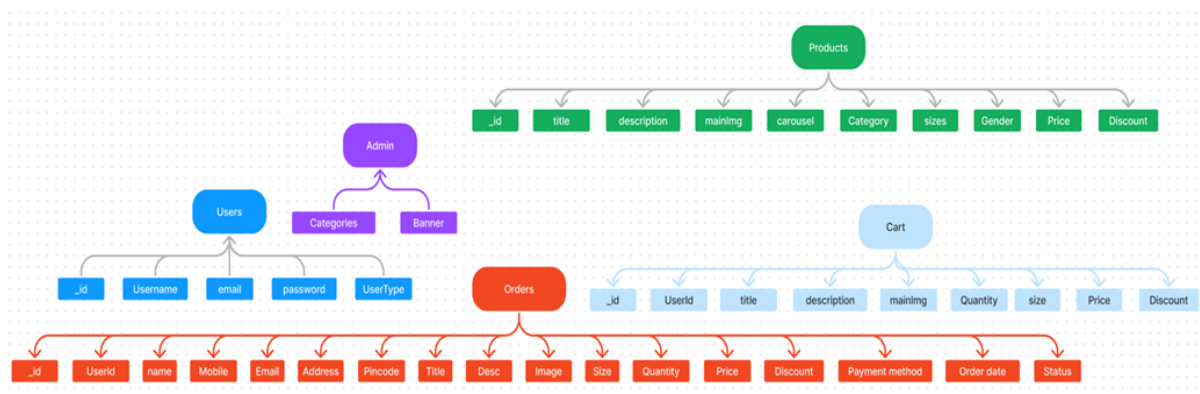
The ShopEZ ER-diagram represents the entities and relationships involved in an e-commerce system. It illustrates how users, products, cart, and orders are interconnected. Here is a breakdown of the entities and their relationships:

USER: Represents the individuals or entities who are registered in the platform.

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

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

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

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

 Features:

1. **Comprehensive Product Catalog:** ShopEZ boasts an extensive catalog of products, offering a diverse range of items and options for shoppers. You can effortlessly explore and discover various products, complete with detailed descriptions, customer reviews, pricing, and available discounts, to find the perfect items for your needs.

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

3. **Order Details Page**: Upon clicking the "Shop Now" button, you will be directed to an order details page. Here, you can provide relevant information such as your shipping address, preferred payment method, and any specific product requirements.

4. **Secure and Efficient Checkout Process:** ShopEZ guarantees a secure and efficient checkout process. Your personal information will be handled with the utmost security, and we strive to make the purchasing process as swift and trouble-free as possible.

5. **Order Confirmation and Details:** After successfully placing an order, you will receive a confirmation notification. Subsequently, you will be directed to an order details page, where you can review all pertinent information about your order, including shipping details, payment method, and any specific product requests you specified.

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

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

Pre-requisites:

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

Node.js and npm:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Clone the repository:

• Open your terminal or command prompt.

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

Drive Link: https://drive.google.com/drive/folders/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.

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.

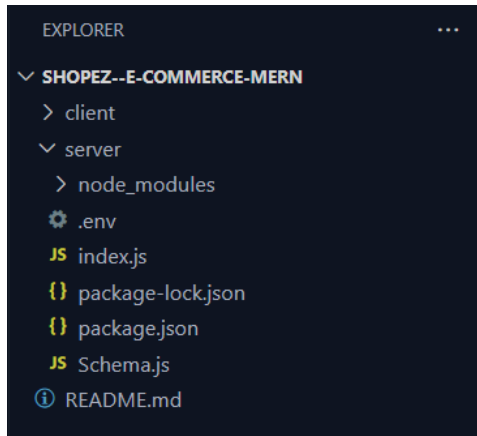• Admins can access the users list, products, orders, etc.,

Project Structure

```
∨ SHOPEZ--E-COMMERCE-MERN
  ∨ client
    > node_modules
    > public
    ∨ src
      > components
      > context
      > images
      > pages
      > styles
      # App.css
      JS App.js
      JS App.test.js
      # index.css
      JS index.js
      🔖 logo.svg
      JS reportWebVitals.js
      JS setupTests.js
    {} package-lock.json
    {} package.json
    ⓘ README.md
  > server
  ⓘ README.md
```

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.

EXPLORER   ...

∨ SHOPEZ--E-COMMERCE-MERN
  > client
  ∨ server
    > node_modules
    ⚙ .env
    JS index.js
    {} package-lock.json
    {} package.json
    JS Schema.js
  ⓘ README.md

Project Flow

Let's start with the project development with the help of the given activities.

Project Setup And Configuration

Milestone 1: Project Setup and Configuration:

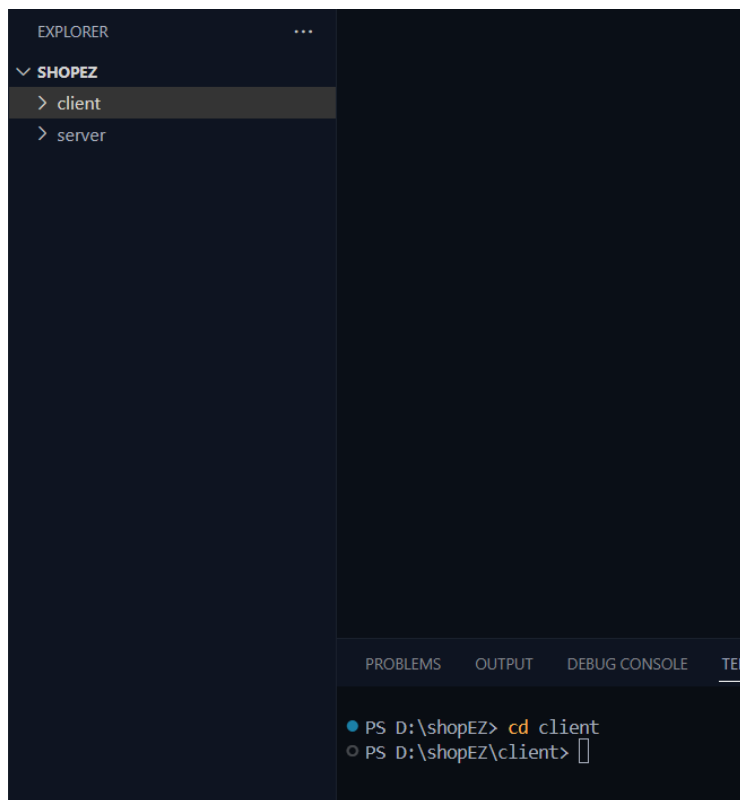**1. Install required tools and software:**

• Node.js.

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

• Git.

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

2. Create project folders and files:

• Client folders.

• Server folders

Referral Image:

Reference video:

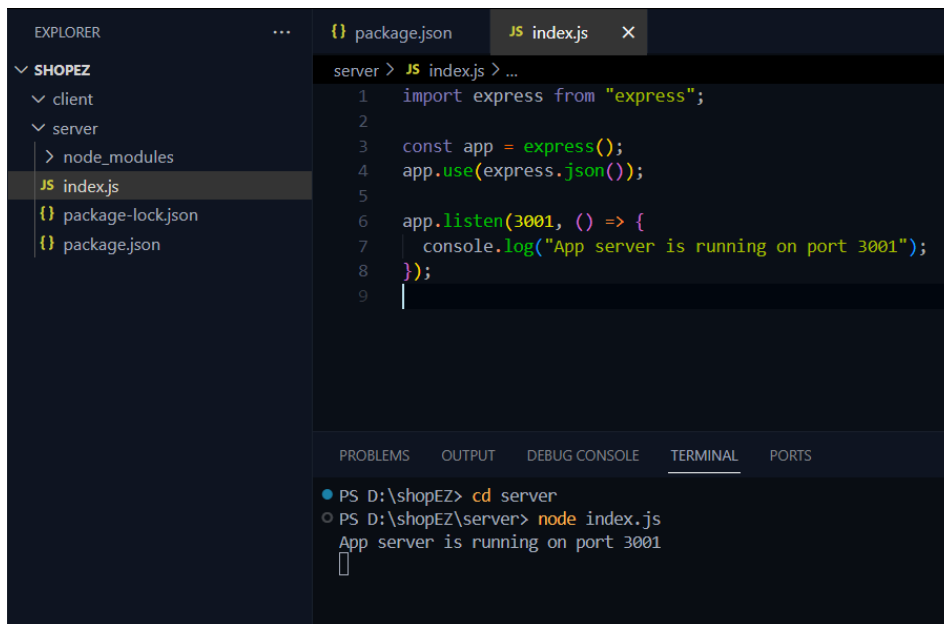folder.mp4 - Google Drive..

No description..

Backend Development

1. Setup express server:

• Create index.js file.

• Create an express server on your desired port number.

• Define API's

Reference Image:

Now your express is successfully created.

Set Up Project Structure:

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

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

Reference Images:

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.

Setup Express Server:

Setup Project Structure:
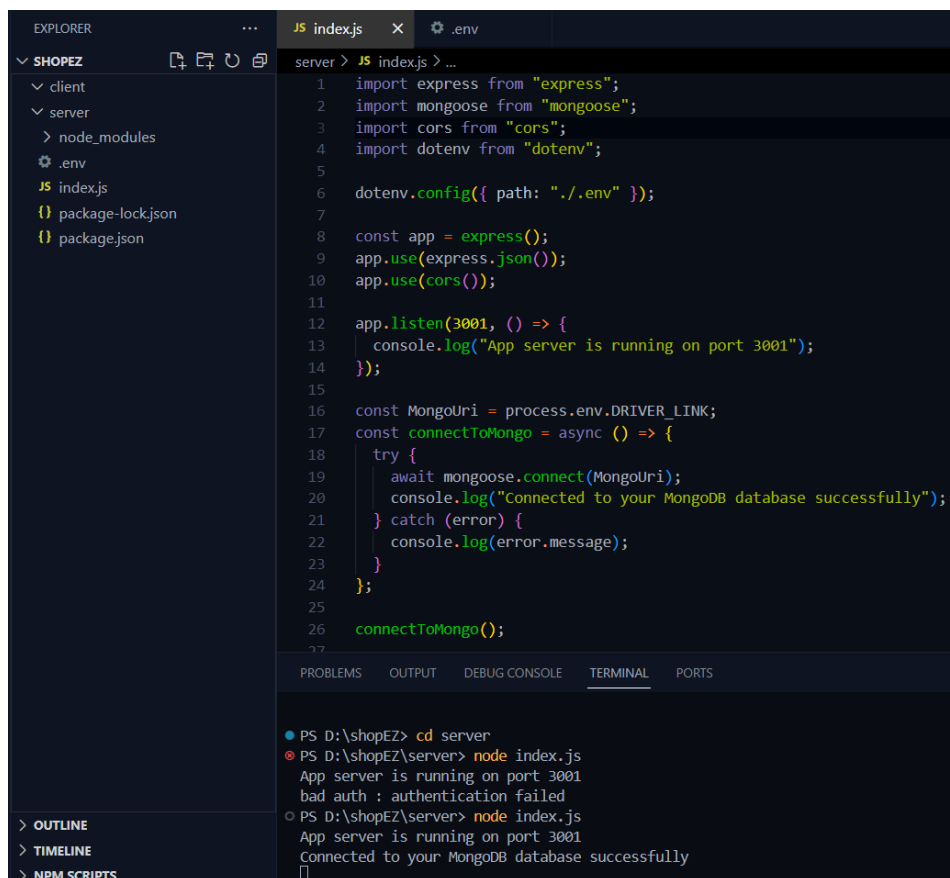
Database Development:

Create database in cloud

• Install Mongoose.

• Create database connection.

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

Reference Image:

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'

• 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
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
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);
```

Database creation:

database.mp4 - Google Drive..

No description..

https://drive.google.com/file/d/1CQil5KzGnPvkVOPWTLP0h-Bu2bXhq7A3/view

Reference video of connect node with mongodb database

mongodb-node-.mp4 - Google Drive..

No description..

https://drive.google.com/file/d/1cTS3_-EOAAvDctkibG5zVikrTdmoY2Ag/view?usp=sharing

Frontend development

1. Setup React Application:

• Create a React app in the client folder.

• Install required libraries

• Create required pages and components and add routes.

2.Design UI components:

• Create Components.

• Implement layout and styling.

• Add navigation.

3.Implement frontend logic:

• Integration with API endpoints.

• Implement data binding.

Reference Image:
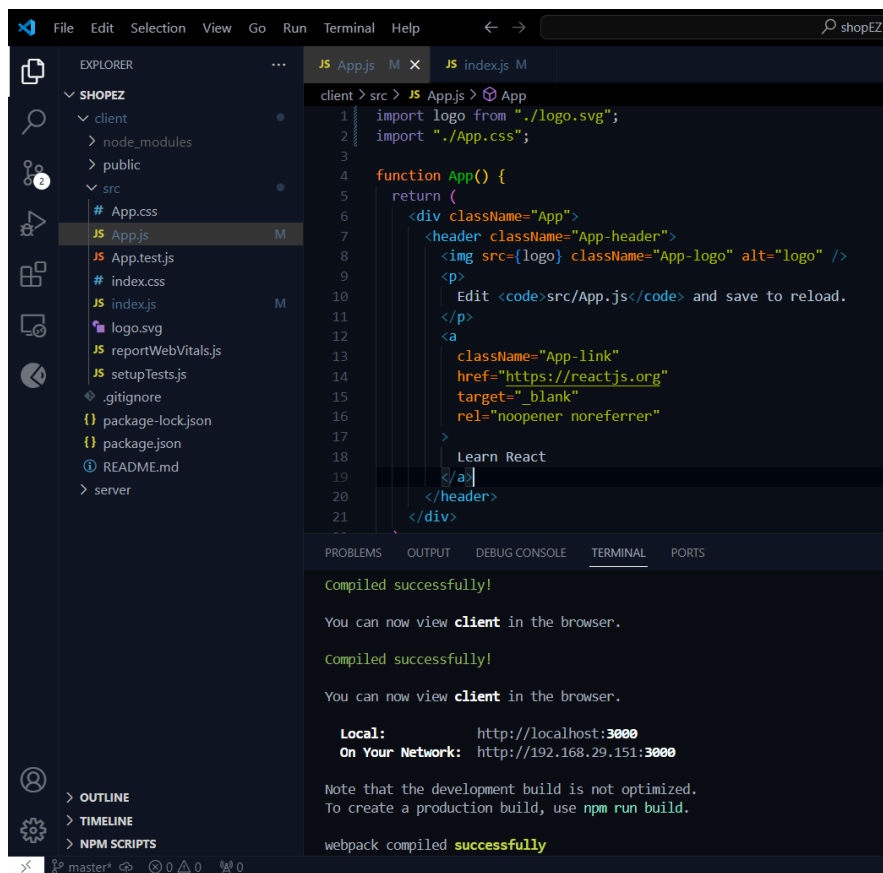
Reference video

react-tutorial.mp4 - Google Drive..

No description..

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

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        });
64
```

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        }
69
```

Register:

```
JS GeneralContext.js U  ×

client > src > context > JS GeneralContext.js > [∅] GeneralContextProvider > [∅] logout
70
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:

```jsx
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:

```jsx
    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)=>{
            setCategories(response.data);
          }
        )
    }
```

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

```js
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:

```jsx
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.

```jsx
 61    const buyNow = async() =>{
 62        await axios.post('http://localhost:6001/buy-product',{userId, name, email, mobile, address,
 63                                    pincode, title: productName, description: productDescription,
 64                                    mainImg: productMainImg, size, quantity: productQuantity, price: productPrice,
 65                                    discount: productDiscount, paymentMethod: paymentMethod, orderDate: new Date()}).then(
 66            (response)=>{
 67                alert('Order placed!!');
 68                navigate('/profile');
 69            }
 70        ).catch((err)=>{
 71            alert("Order failed!!");
 72        })
 73    }
 74
 75    const handleAddToCart = async() =>{
 76        await axios.post('http://localhost:6001/add-to-cart', {userId, title: productName, description: productDescription,
 77                                        mainImg: productMainImg, size, quantity: productQuantity, price: productPrice,
 78                                        discount: productDiscount}).then(
 79            (response)=>{
 80                alert("product added to cart!!");
 81                navigate('/cart');
 82            }
 83        ).catch((err)=>{
 84            alert("Operation failed!!");
 85        })
 86    }
 87
```

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
234    // buy product
235
236    app.post('/buy-product', async(req, res)=>{
237        const {userId, name, email, mobile, address, pincode, title,
238                        description, mainImg, size, quantity, price,
239                        discount, paymentMethod, orderDate} = req.body;
240        try{
241            const newOrder = new Orders({userId, name, email, mobile, address,
242                            pincode, title, description, mainImg, size, quantity, price,
243                            discount, paymentMethod, orderDate})
244            await newOrder.save();
245            res.json({message: 'order placed'});
246        }catch(err){
247            res.status(500).json({message: "Error occured"});
248        }
249    })
```

Authentication:

Backend

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

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

Frontend

Login:

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

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

Order products:

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

Frontend

```jsx
Cart.jsx 3, U ×

client > src > pages > customer > Cart.jsx > [●] Cart
83    const [name, setName] = useState('');
84    const [mobile, setMobile] = useState('');
85    const [email, setEmail] = useState('');
86    const [address, setAddress] = useState('');
87    const [pincode, setPincode] = useState('');
88    const [paymentMethod, setPaymentMethod] = useState('');
89
90    const userId = localStorage.getItem('userId');
91    const placeOrder = async() =>{
92      if(cartItems.length > 0){
93          await axios.post('http://localhost:6001/place-cart-order', {userId, name, mobile,
94                              email, address, pincode, paymentMethod, orderDate: new Date()}).then(
95          (response)=>{
96            alert('Order placed!!');
97            setName('');
98            setMobile('');
99            setEmail('');
100           setAddress('');
101           setPincode('');
102           setPaymentMethod('');
103           navigate('/profile');
104         }
105       )
106     }
107   }
```

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
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**:**

```jsx
NewProduct.jsx U X
client > src > pages > admin > NewProduct.jsx > [@] NewProduct
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      }
```

Backend**:**

```js
JS index.js X
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      })
```
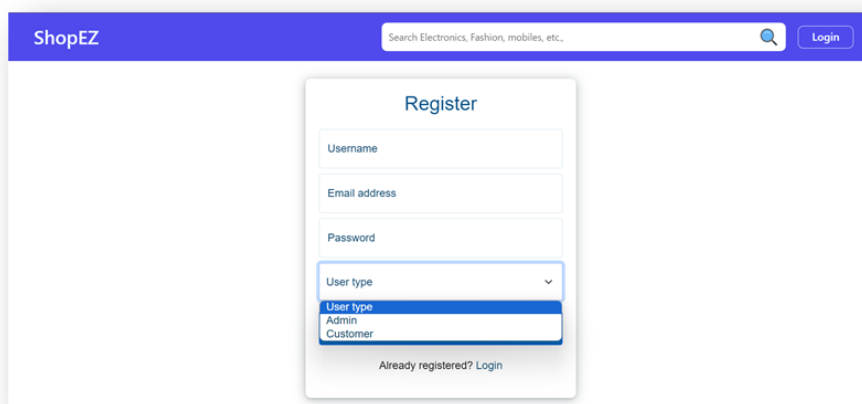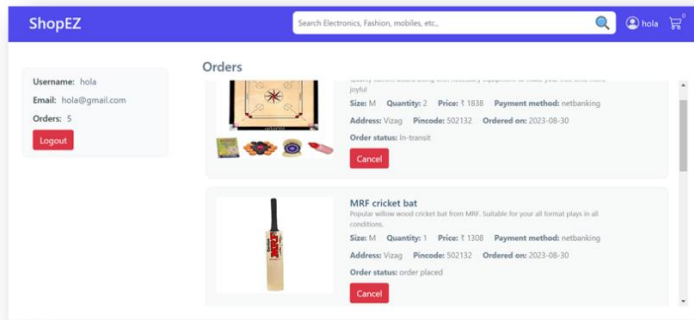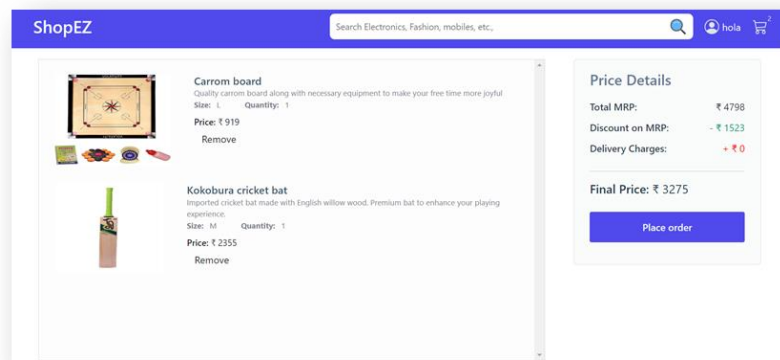
Demo UI images:

Landing page
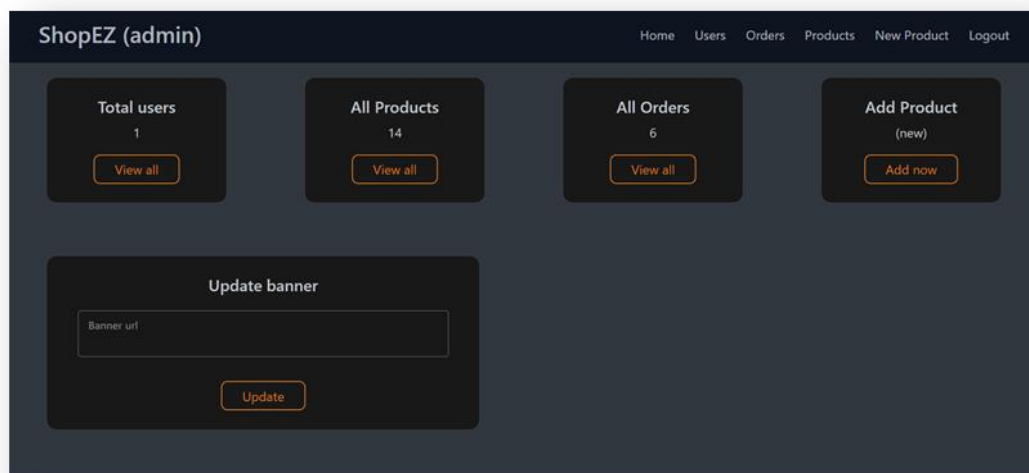

Products:
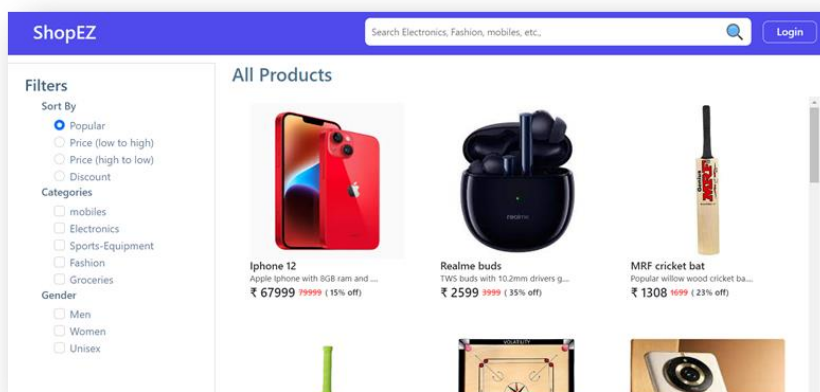
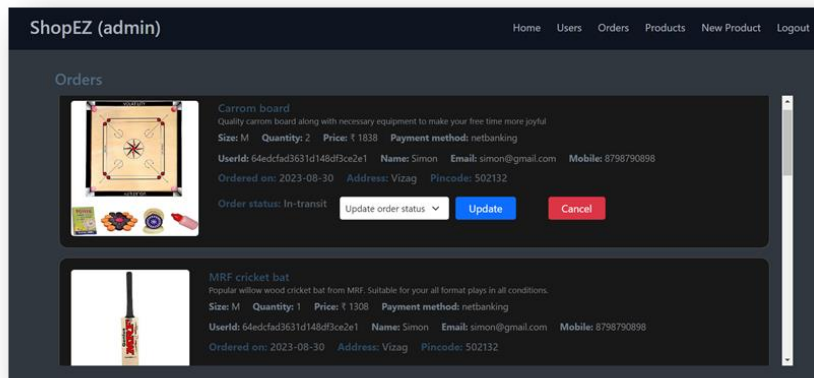## Authentication:

User profile:
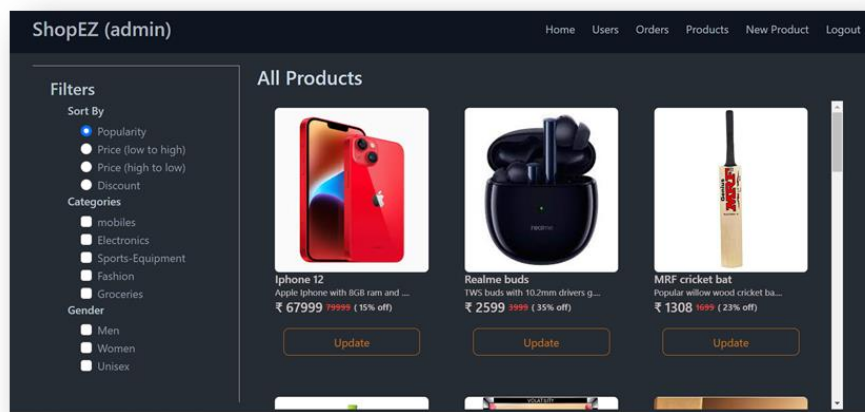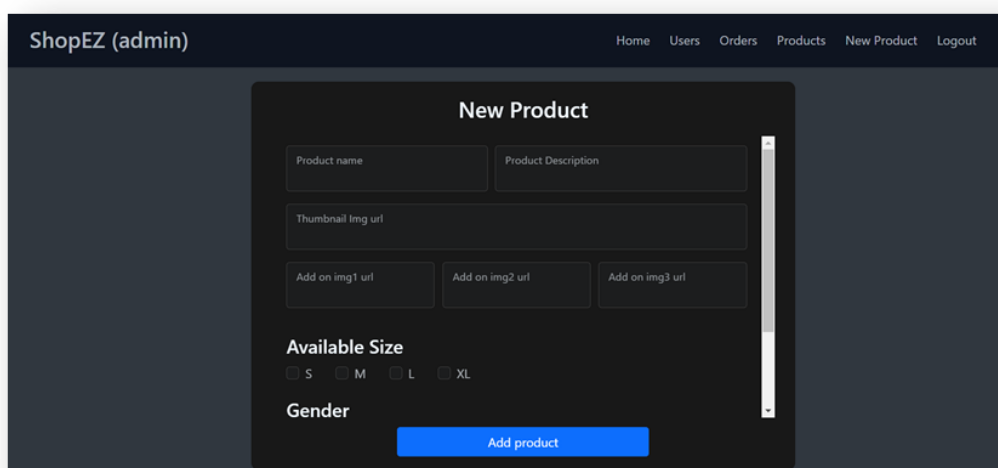

Cart:

Admin dashboard:

All Orders

All Products

New Product Page

CHALLENGES AND SOLUTIONS:

The development and deployment of the ShopEZ e-commerce platform presented a number of challenges that required thoughtful solutions. Below are some of the key challenges encountered during the project and the solutions implemented to overcome them.

1.Handling User Authentication and Authorization

One of the significant challenges faced was ensuring secure user authentication and authorization across different user roles, such as customers, sellers, and admins. Managing secure login, token generation, and access control for multiple user roles required careful planning.

Solution: To address this, bcrypt.js was used to securely hash user passwords, while• jsonwebtoken (JWT) was utilized to generate secure tokens for authentication. Role based access control was implemented on the backend to restrict certain API endpoints based on the user's role, ensuring that customers, sellers, and admins could only access the resources pertinent to their roles.

2. Ensuring Data Integrity Between Collections With multiple collections (Users, Products, Orders, and Cart), it was essential to maintain data consistency and relationships. For instance, when a user places an order, the order must be linked to both the user and the product(s) without introducing errors.

Solution: Mongoose was used to establish relationships between collections and define• validation rules for fields like emails, passwords, and product stock levels. Additionally, middleware was added to handle complex data operations, such as updating product stock when an order is placed or deleting items from the cart when a purchase is completed. Data integrity checks were put in place to ensure these relationships remained consistent.

3. Optimizing Performance with MongoDB As the platform grew, it became clear that performance could be impacted by the increasing amount of data in the database, especially when handling large numbers of products and orders. Queries, especially those involving product search and filtering, needed optimization to ensure fast response times.

Solution: Indexing was implemented on frequently queried fields, such as product• names and categories, to speed up search operations. Additionally, caching strategies were used to temporarily store frequently accessed data, reducing database load and improving performance.

4. Cross-Browser Compatibility and Responsive Design Ensuring the application worked seamlessly across different devices and browsers was a challenge. Different devices, such as smartphones and tablets, and browsers like Chrome, Firefox, and Safari, can display UI components differently.

Solution: The frontend was built using React and Material-UI, which provides a set• of responsive design components. CSS modules were used for styling to ensure the application was modular and maintainable. Tools like Chrome Developer Tools were used to test the app across different screen sizes and browsers, and adjustments were made to ensure consistency.

5. Securing the Application With sensitive data such as passwords, payment information, and user details being processed, security was a top priority. Ensuring the platform was secure from vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) was essential.

Solution: Data was sanitized before being entered into the database to prevent injection• attacks. For security against XSS and CSRF, the application used libraries like helmet to set HTTP headers for enhanced security and cors to control cross-origin resource sharing. Additionally, passwords were hashed using bcrypt.js, ensuring they were never stored in plain text.

FUTURE ENHANCEMENTS:

While ShopEZ is already a fully functional e-commerce platform, there are several areas that could be enhanced or expanded in future versions to improve the user experience, security, and functionality:

1.Advanced Search and Filtering

The current search and filter capabilities are basic, allowing users to search by product name, category, or price. In the future, advanced search options can be implemented, such as:

- Filters based on multiple criteria (e.g., color, size, brand)
- Personalized product recommendations based on browsing history and user preferences.

2. Real-Time Order Tracking

Currently, users receive email notifications upon order placement and shipment. A real-time tracking feature can be added where customers can view the status of their orders directly on the platform. This could be integrated with third-party logistics services or custom-built APIs to track shipment statuses.

3. Multi-Language and Multi-Currency Support

As ShopEZ expands, supporting multiple languages and currencies will be essential to cater to a global audience. This would allow users from different regions to shop in their preferred language and currency, creating a more personalized shopping experience.

4. Mobile App Development

While the web application is fully functional, creating a native mobile application for iOS and Android would enhance user accessibility. The mobile app could offer features like push notifications for order status, special offers, and promotions.

5. AI-Powered Chatbot for Customer Support

Incorporating a chatbot powered by Artificial Intelligence (AI) would enhance the customer experience by providing instant support for common inquiries such as order status, product availability, and return policies. The chatbot could integrate with backend systems for real time assistance.

6. Enhanced Seller Dashboard

Expanding the seller dashboard with advanced analytics features, such as:

- Sales trends and revenue breakdown.
- Detailed customer insights (e.g., purchase behavior, demographics).
- Inventory management tools to automatically update product availability.

CONCLUSION:

The ShopEZ platform is a comprehensive e-commerce solution built using the MERN stack, offering a seamless shopping experience for customers and a robust backend for sellers. The platform incorporates key features such as secure user authentication, an intuitive UI, real-time product management, and efficient order handling. Testing and deployment were carefully executed to ensure a bug-free, secure, and scalable system. While the project has been successfully implemented, there is always room for improvement and growth. Future enhancements, such as advanced search capabilities, mobile app support, and AI-powered chatbots, can help elevate the platform to the next level, making it even more user-friendly and versatile. With the growing demand for online shopping, platforms like ShopEZ offer valuable opportunities for both customers and sellers. By continuing to innovate and improve, ShopEZ can become a leading e-commerce platform in the market.

REFERENCES

1. MERN Stack Documentation - MongoDB, Express, React, Node.js: https://www.mongodb.com/mern-stack

2. JWT Authentication in Node.js: https://jwt.io/introduction/

3. Mongoose Documentation: https://mongoosejs.com/docs/

4. React Documentation: https://reactjs.org/docs/getting-started.html

5. MongoDB Atlas Documentation: https://www.mongodb.com/cloud/atlas

These references provide detailed guides and documentation for the technologies and tools used to build, deploy, and optimize the ShopEZ platform.