Shopez:one-stop shop for online purchases

★ Project Title:

Shopez:one-stop shop for online purchases

★ Team Name:

Code Crafters

★ Team ID:

LTVIP2025TMID41850

★ Team Members:

→ *Ramya Peluri* – Frontend, Backend and implementation of project(Role)

→ *Kolla Vivek* – Frontend, Backend and implementation of project

→ *Madiki Ram Surya Ganesh* - NULL

→ *Durga Sai Mahesh Peyyala* – NULL

Abstract

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

2. Project Overview

• Purpose:

The purpose of the **ShopEZ** project is to design and develop a full-stack e-commerce web application that simplifies and enhances the online shopping experience for customers while providing powerful tools for sellers to manage and grow their businesses. The project aims to deliver a user-friendly, secure, and scalable platform where users can effortlessly discover products, review detailed descriptions, view discounts, and complete purchases through a seamless checkout process.

For sellers, ShopEZ offers an intuitive admin dashboard for managing products, processing orders, and accessing analytics that provide valuable insights into business performance. The primary goals of the project include:

- Providing a smooth and engaging shopping experience for users.
- Enabling sellers to efficiently manage their products and orders.
- Supporting business growth through analytics and reporting features.
- Ensuring data security and reliable performance through modern technologies (React JS, Node.js, MongoDB).
- Demonstrating the effective integration of frontend, backend, and database systems in a full-stack application.

ShopEZ strives to create a future-ready e-commerce solution that benefits both customers and businesses.

• Features:

ShopEZ offers a comprehensive set of features designed to enhance the shopping experience for customers and provide efficient management tools for sellers. Key features and functionalities include:

For Customers:

- **Effortless Product Discovery**
  Easily browse and search through a wide range of products with categorized listings and filtering options.
- **Personalized Shopping Experience**
  View product recommendations based on user preferences and browsing history (extendable in future versions).
- **Detailed Product Descriptions**
  Access complete product information, specifications, pricing, and customer reviews to make informed purchase decisions.
- **Seamless Checkout Process**
  Secure and streamlined checkout with order confirmation and payment processing.
- **Cart Management**
  Add, update, or remove items from the shopping cart before placing orders.

For Sellers/Admin:

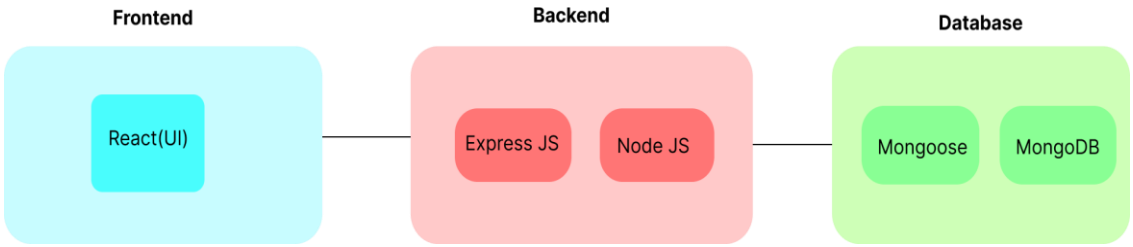
- **Efficient Order Management**
  Manage orders, track statuses, and update delivery information through the admin dashboard.
- **Product Management**
  Add, update, or delete products, and manage product details and availability.

- **Insightful Business Analytics**
  View reports and analytics to monitor sales trends, product performance, and customer activity.
- **Admin Authentication**
  Secure login and access control for admin and seller functionalities.

General:

- **Secure Authentication System**
  User and admin authentication to protect user data and ensure authorized access.
- **Scalable Architecture**
  Built using React JS, Node.js, and MongoDB to ensure scalability and maintainability.
- **Responsive Design**
  Optimized for use across devices (desktops, tablets, mobiles).

3. Architecture



2.1 FRONTEND ARCHITECTURE

- **React:** React is used to build a dynamic and user-friendly interface with reusable components, enabling customers to seamlessly browse products, search items, manage their cart, and complete purchases. Sellers and admins can efficiently manage products, orders, and view analytics through the admin dashboard. React's virtual DOM ensures high performance by updating only the components that change, resulting in a fast and responsive user experience.
- **Axios:** Axios facilitates communication between the frontend and backend through API calls, handling tasks such as fetching product data, managing user authentication, processing orders, and interacting with admin functionalities in a smooth and efficient manner.

2.2 BACKEND ARCHITECTURE

- **Node.js:** Node.js serves as the runtime environment for the backend, enabling non-blocking, asynchronous operations that allow the application to handle multiple concurrent requests, such as product retrieval, order processing, and user authentication.
- **Express.js:** Express.js functions as the web framework to manage server-side logic, including routing, API endpoint creation, order management, product updates, and admin-

specific operations. It helps structure the backend into modular, maintainable components that support RESTful API interactions with the frontend.

- **JWT Authentication:** JSON Web Tokens (JWT) secure the application by validating user and admin identities, enforcing role-based access control, and protecting restricted features like order management, product editing, and access to analytics.

2.3 DATABASE ARCHITECTURE

- **MongoDB:** MongoDB is the NoSQL database selected for its flexibility and scalability, storing collections that represent users, products, carts, orders, and analytics data. Its document-oriented design simplifies managing diverse and dynamic data structures required by the application.
- **Document Structure:** The database schema is organized into collections such as **Users** (user profiles, authentication data), **Products** (product details, categories, pricing, discounts), **Orders** (order records, statuses, payment info), and **Cart** (items added by users). This structure enables efficient querying and updating of data relevant to both customers and sellers.

4. Setup Instructions

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://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: 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.

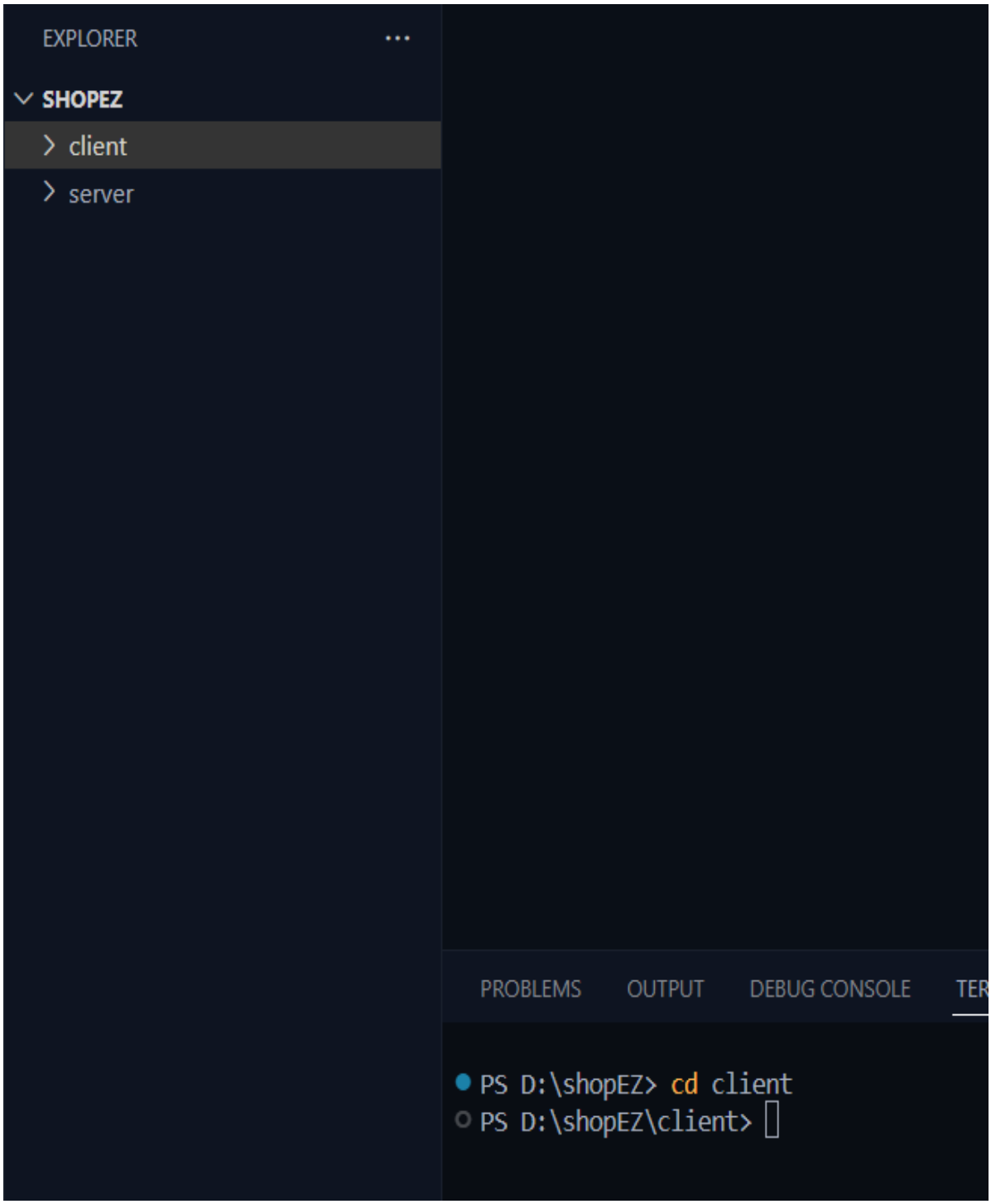
2. Admin Flow:

• 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.,

5. Folder Structure



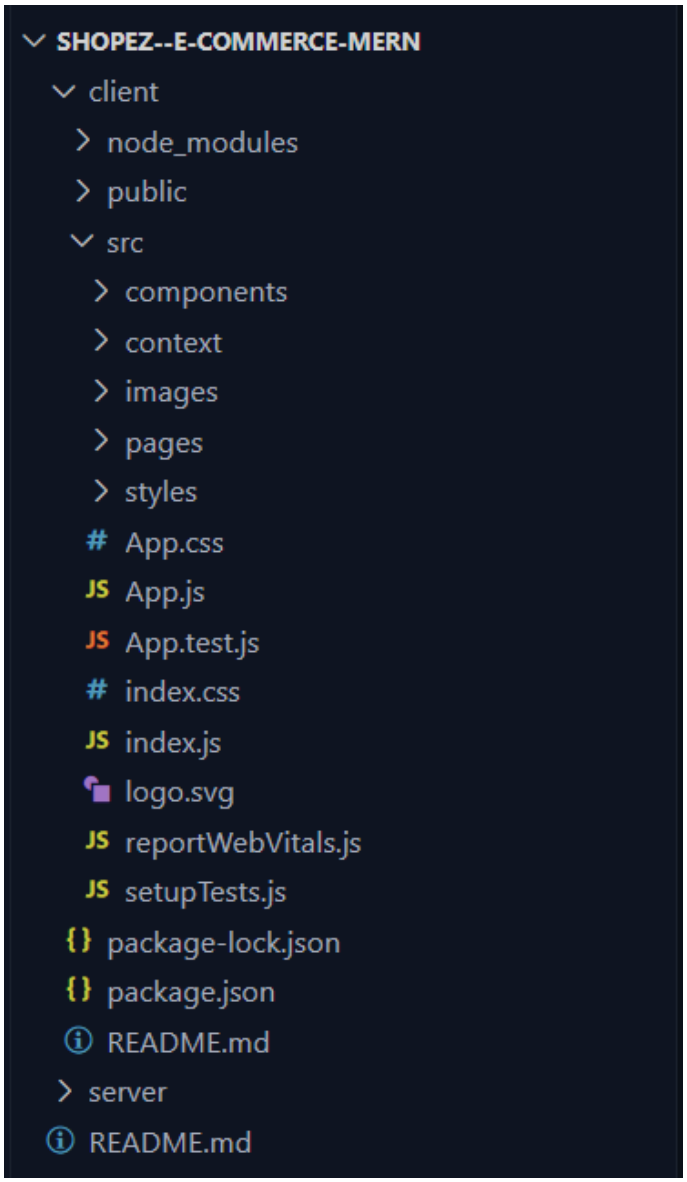
1. User Flow:

- 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.,

- **Client:**

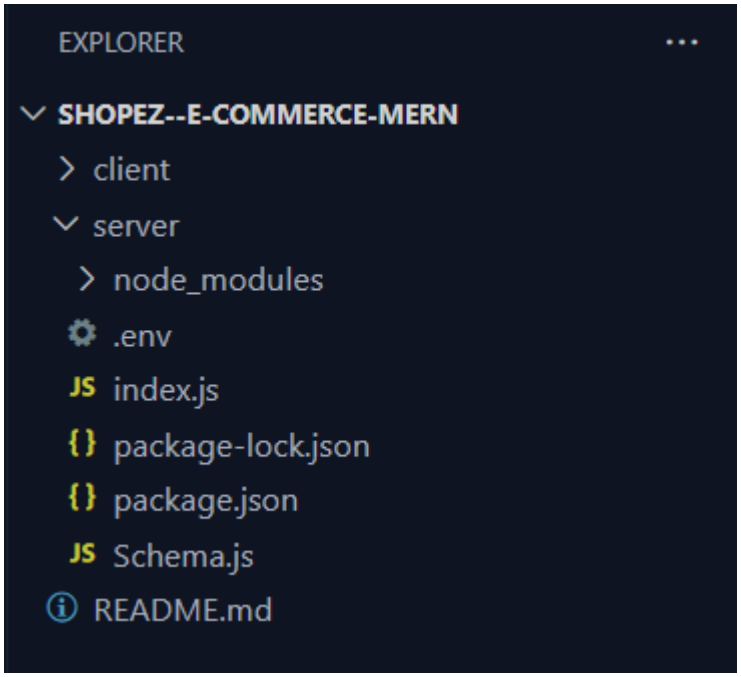


This structure assumes a React app and follows a modular approach. Here's a brief explanation of the main directories and files:

- 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.

• Server:



6. Running the Application

Commands to Start the Frontend and Backend Servers Locally

If node_modules folder is not present (for example, right after cloning the project), first install the dependencies.

→ **Frontend (client directory)**

1️⃣ Install dependencies:

cd client

npm install

2️⃣ Start the server:

npm start

→ **Backend (server directory)**

1️⃣ Install dependencies:

cd server

npm install

2️⃣ Start the server:

npm start

7. API Documentation

BACKEND DEVELOPMENT:

Setup express server:

• Create index.js file.

• Create an express server on your desired port number.

• Define API’s

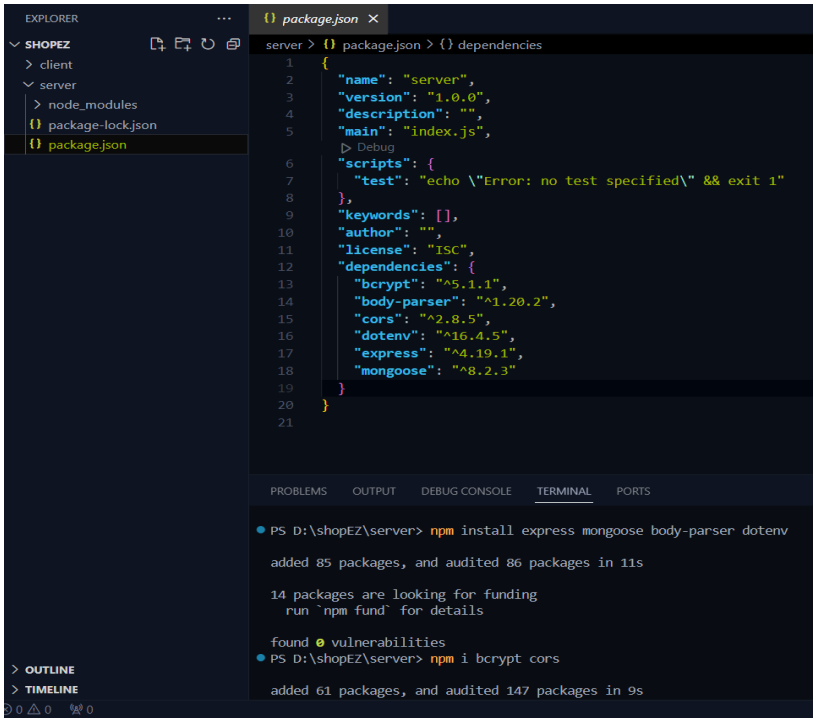
Reference Video: https://drive.google.com/file/d/1-uKMIcrok_ROHyZl2vRORggrYRio2qXS/view?usp=sharing

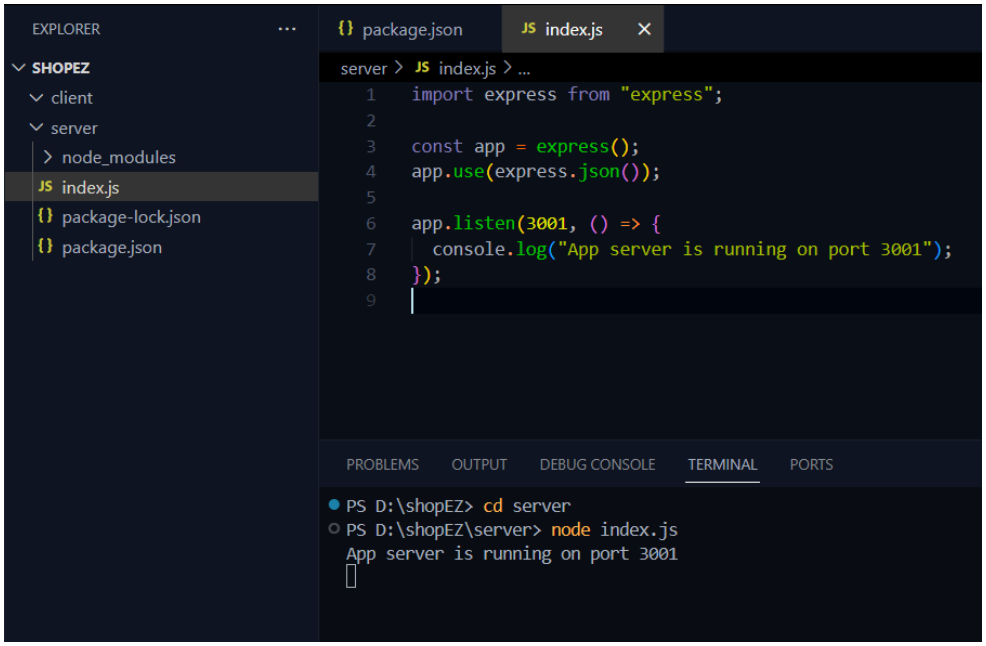
Set Up Project Structure:

• 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





2. Database Configuration:

• Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas or use locally with MongoDB compass.

• Create a database and define the necessary collections for admin, users, products, orders and other relevant data.

3. Create Express.js Server:

• Set up an Express.js server to handle HTTP requests and serve API endpoints.

• Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

4. Define API Routes:

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

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

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

5. Implement Data Models:

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

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

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

6. User Authentication:

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

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

7. Handle new products and Orders:

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

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

8. Admin Functionality:

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

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

9. Error Handling:

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

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

DATABASE DEVELOPMENT:

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

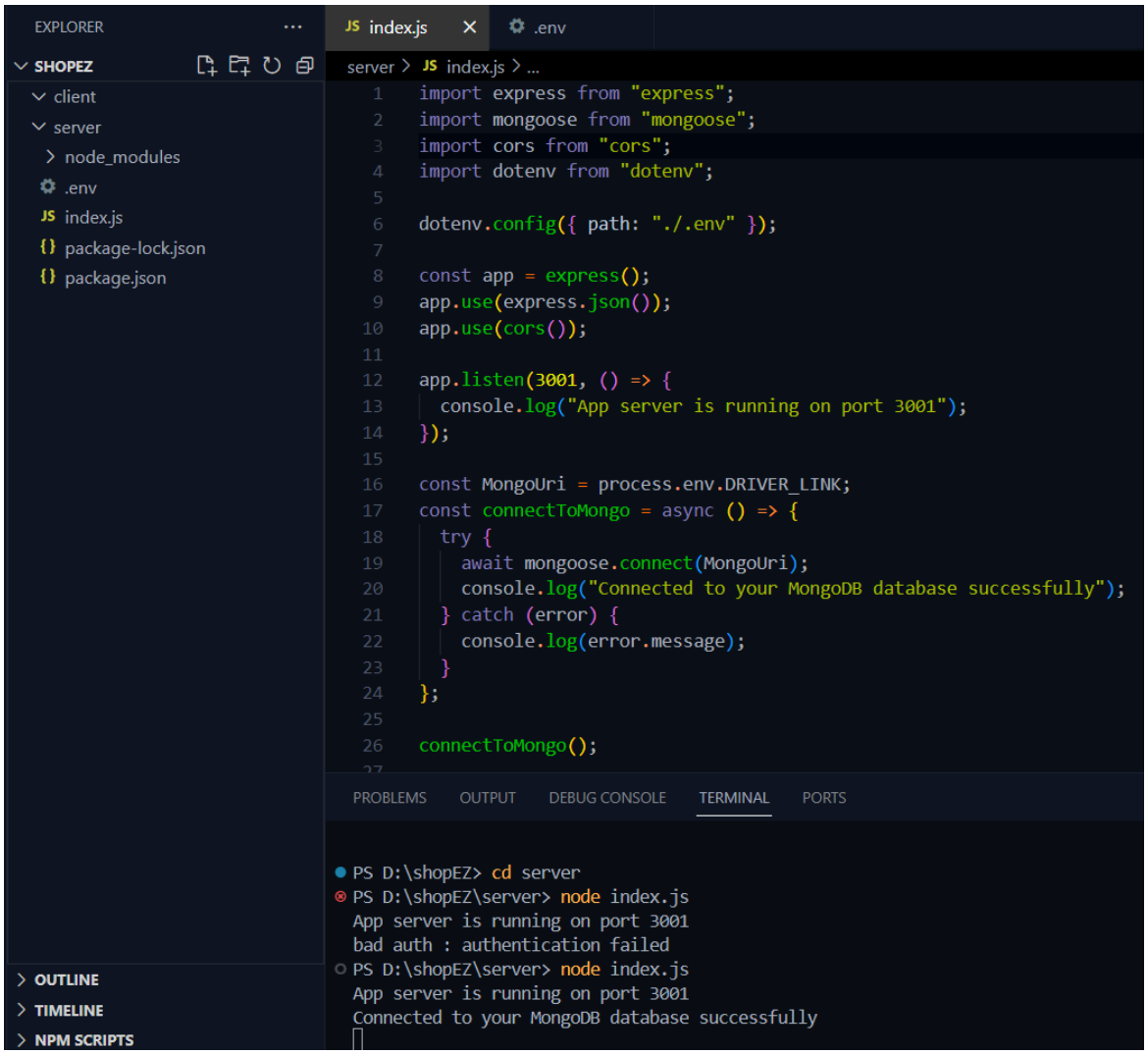
• Install Mongoose.

• 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:



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

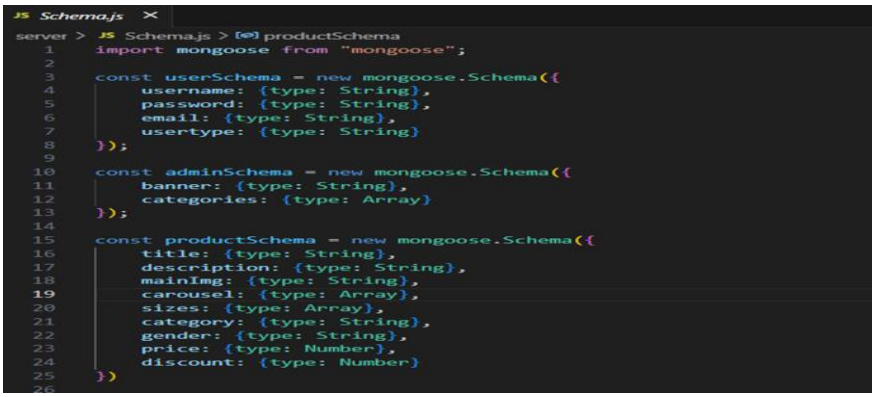
• Model: ‘Admin’

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

Code Explanation:

Schemas:

Now let us define the required schemas



Schema.js

server > 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);

```

8. AUTHENTICATION

→ *Authentication and Authorization Handling*:

In the ShopEZ project, authentication and authorization are implemented to ensure secure access for both customers and admins. The system uses JSON Web Tokens (JWT) for stateless authentication, allowing the server to verify user identity without maintaining session data on the server side.

- When a user (customer or admin) logs in successfully, the backend generates a JWT token containing encoded information such as the user's ID and role.
- This token is sent back to the frontend and typically stored on the client side (e.g., in local storage or session storage).
- For each protected API request (e.g., viewing orders, managing products), the client sends this token in the Authorization header.
- The backend verifies the token's validity and decodes it to authorize access based on the user's role (e.g., customer vs admin).

→ Token Details:

- Type: JSON Web Token (JWT)
- Contents: User ID, role, and expiry time
- Storage: Client-side storage (localStorage or sessionStorage — depending on the implementation choice)
- Expiry: Tokens are set to expire after a configured duration (e.g., 1 hour), requiring the user to log in again for security.
-

→ Authorization:

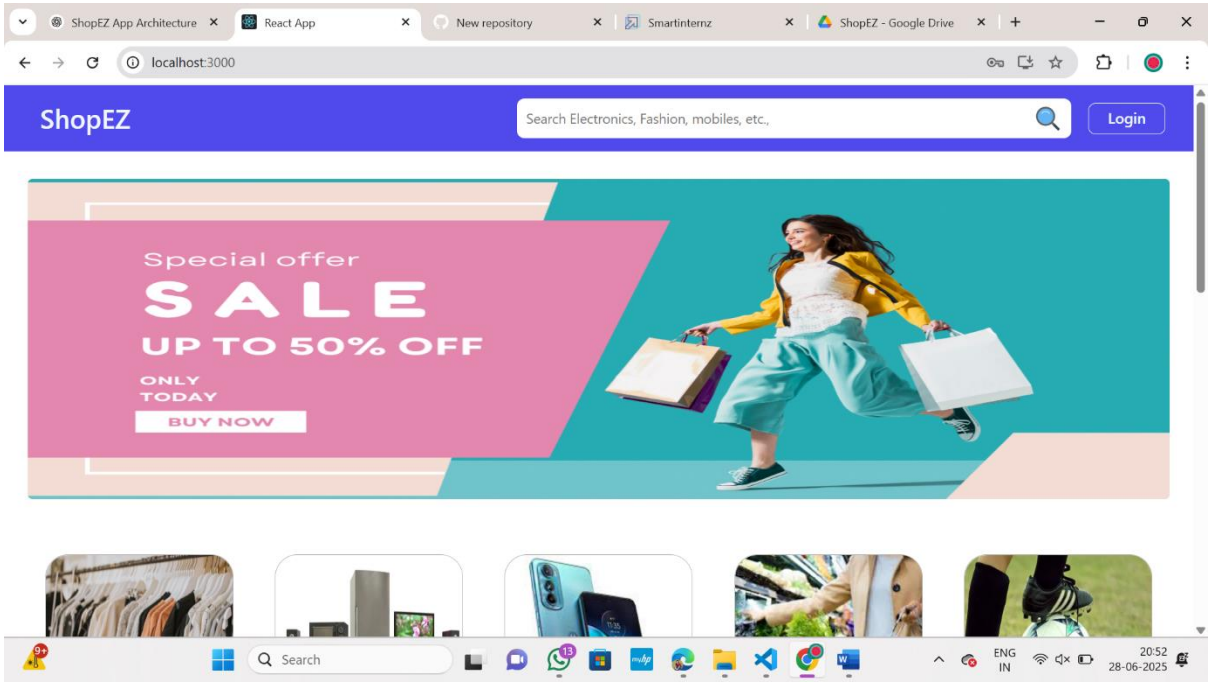
- Role-based access control is enforced.
- Customers can access endpoints like product browsing, cart

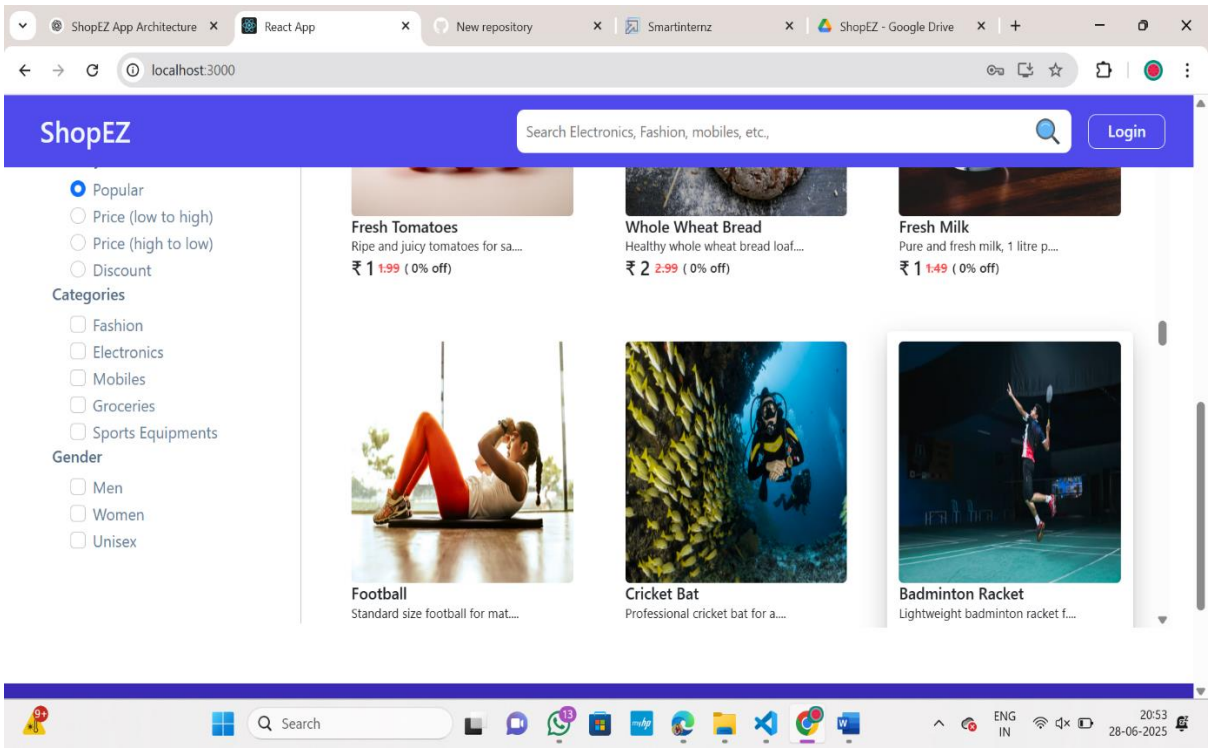
management, and order placement.

- Admins have additional access to endpoints for managing products, orders, and viewing analytics.

→ No sessions or cookies:
ShopEZ uses token-based authentication rather than traditional session-based or cookie-based authentication, making it suitable for RESTful API architecture.

9. User Interface





10. Testing

→ **Testing Strategy:**

The **ShopEZ** project adopts a combination of manual and automated testing strategies to ensure the reliability, functionality, and performance of the application. Testing covers both the **frontend** and **backend** components, focusing on key areas such as:

- **Unit testing:** Testing individual functions, components, and modules to verify they work as expected in isolation.
- **Integration testing:** Ensuring that the interaction between frontend, backend, and database is smooth and correct (e.g., placing orders, logging in, managing products).
- **End-to-end (E2E) testing (optional/extendable):** Simulating user flows across the full system, such as logging in, adding products to cart, and completing checkout.

→ **Tools Used:**

- **Frontend Testing:**
    - *Jest:* For unit testing React components and utility functions.
    - *React Testing Library:* For testing React components in a way that simulates user interactions.
- **Backend Testing:**
    - *Jest* or *Mocha + Chai:* For testing API endpoints and backend logic.
    - *Supertest:* For testing HTTP requests and verifying API responses.

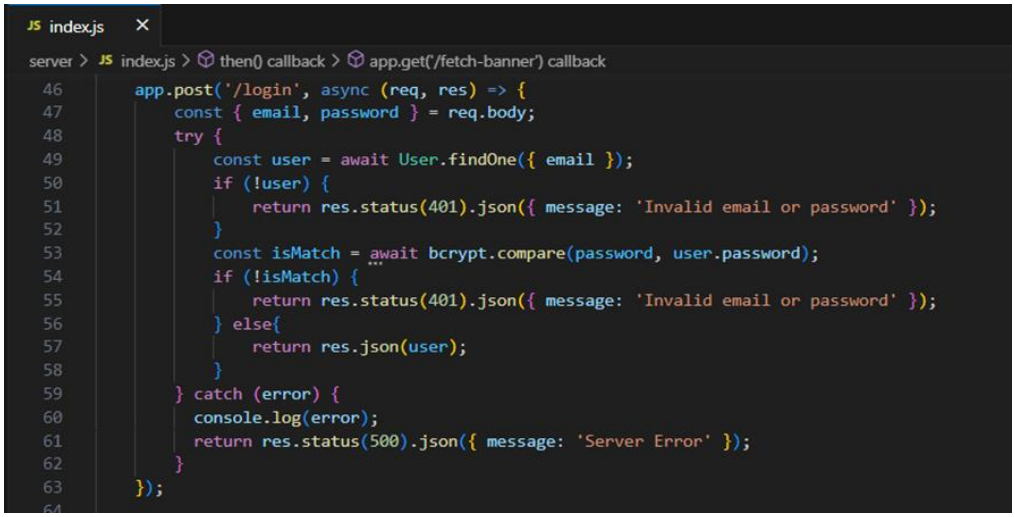
- **Manual Testing:**
    - Browser-based testing across Chrome and Firefox for user interface and flow validation.
    - API testing using tools like *Postman* to verify API endpoints, authentication, and error handling.

PROJECT IMPLEMENTATION & EXECUTION:

User Authentication:

· Backend

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



Frontend

Login:

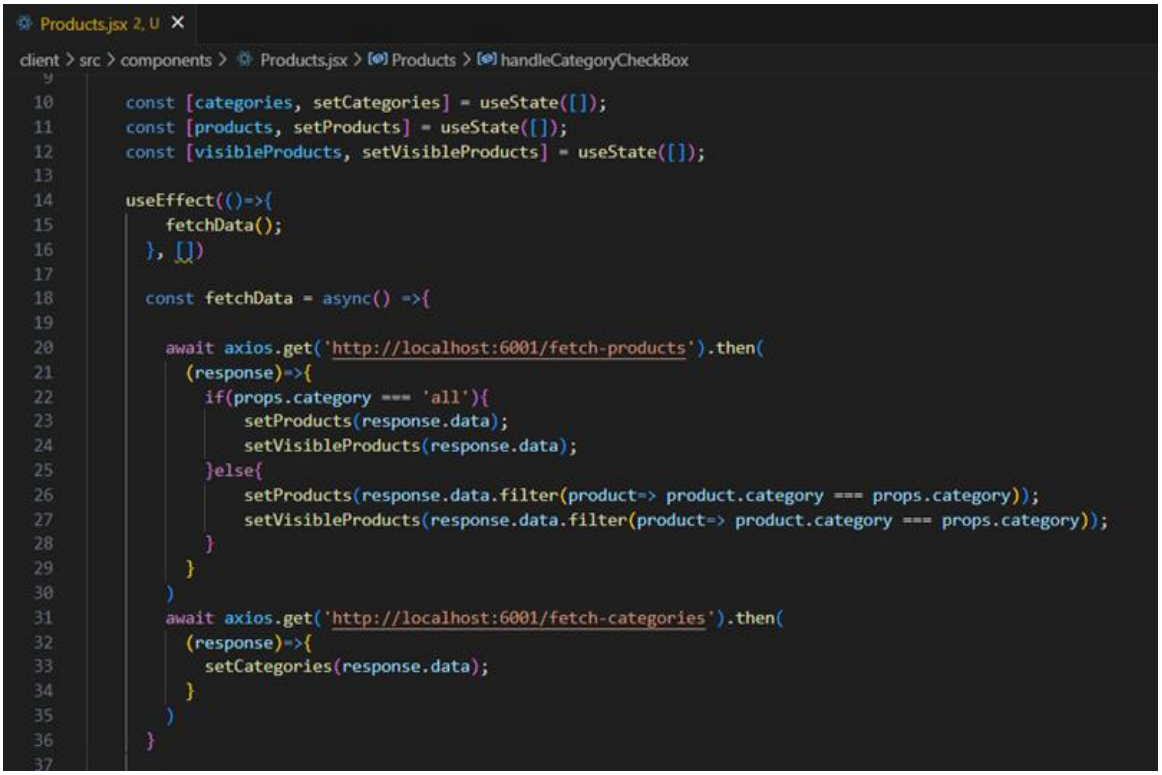
Register:

logout:

All Products (User): In the home page, we'll fetch all the products

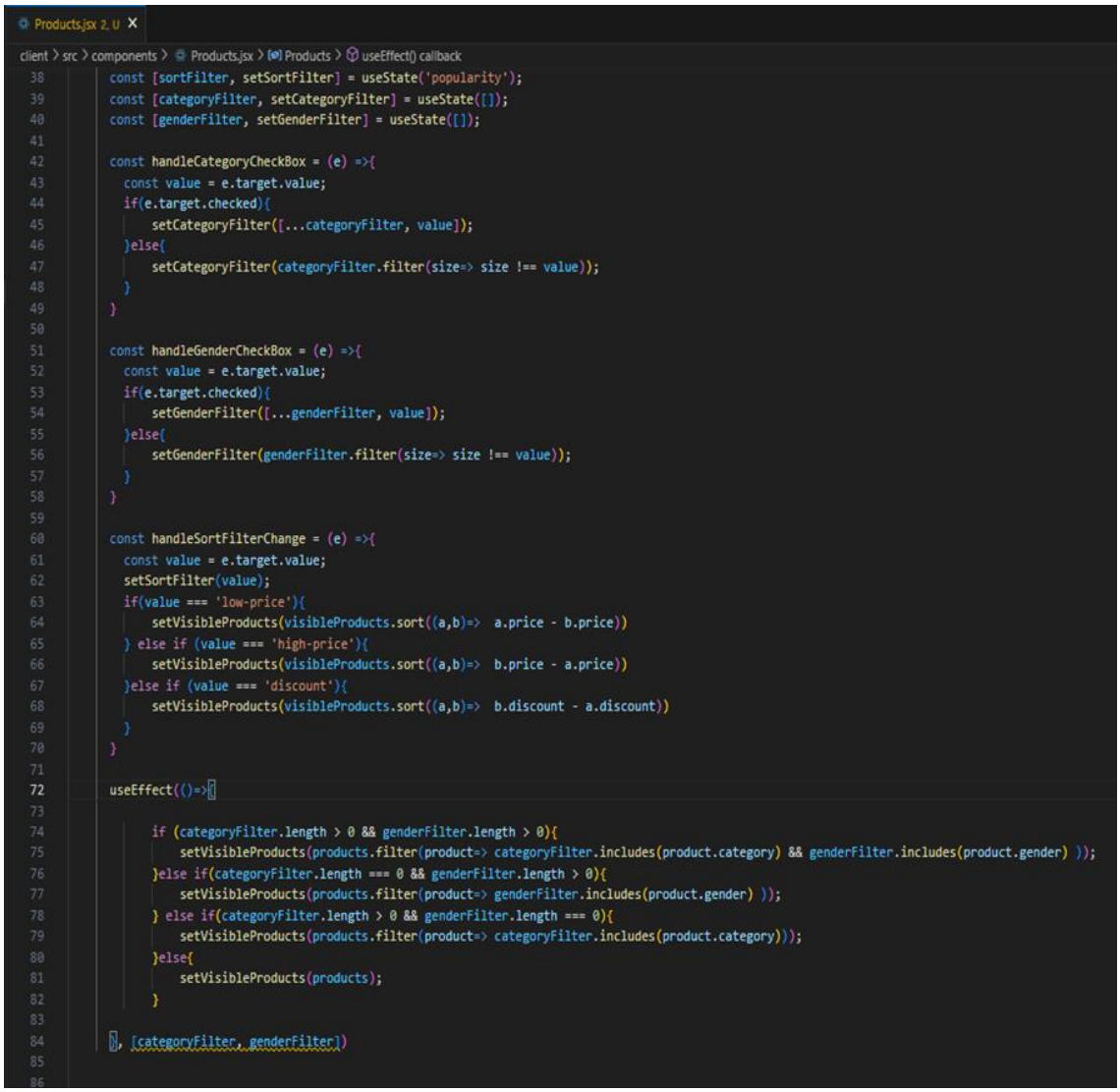
available in the platform along with the filters.

Fetching products:

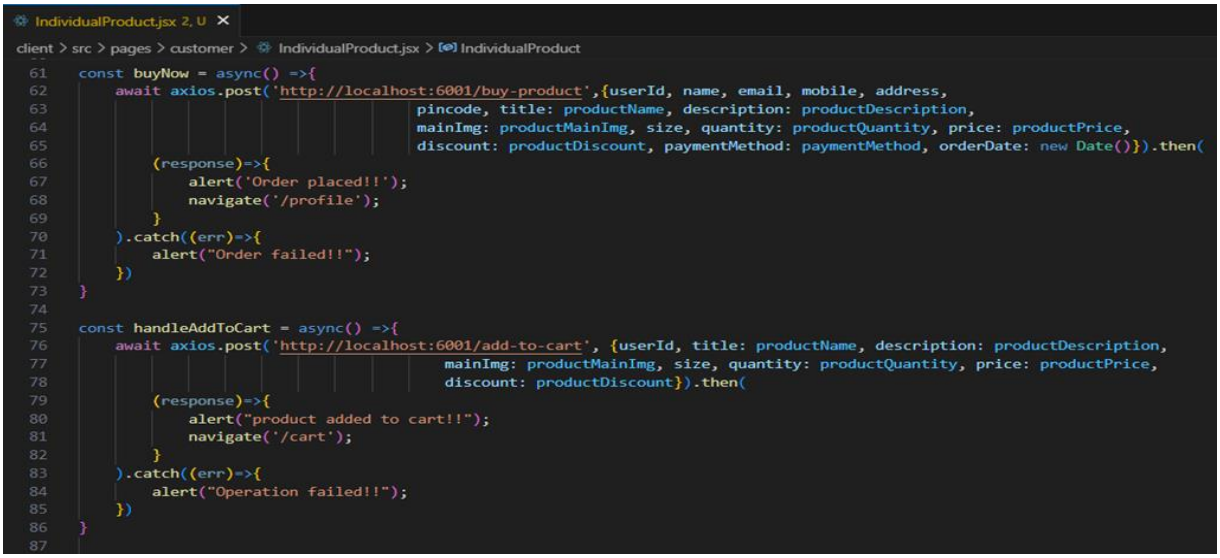


In the backend, we fetch all the products and then filter them on the client side.

Filtering



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



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:

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');
        }
      )
  }
}
```

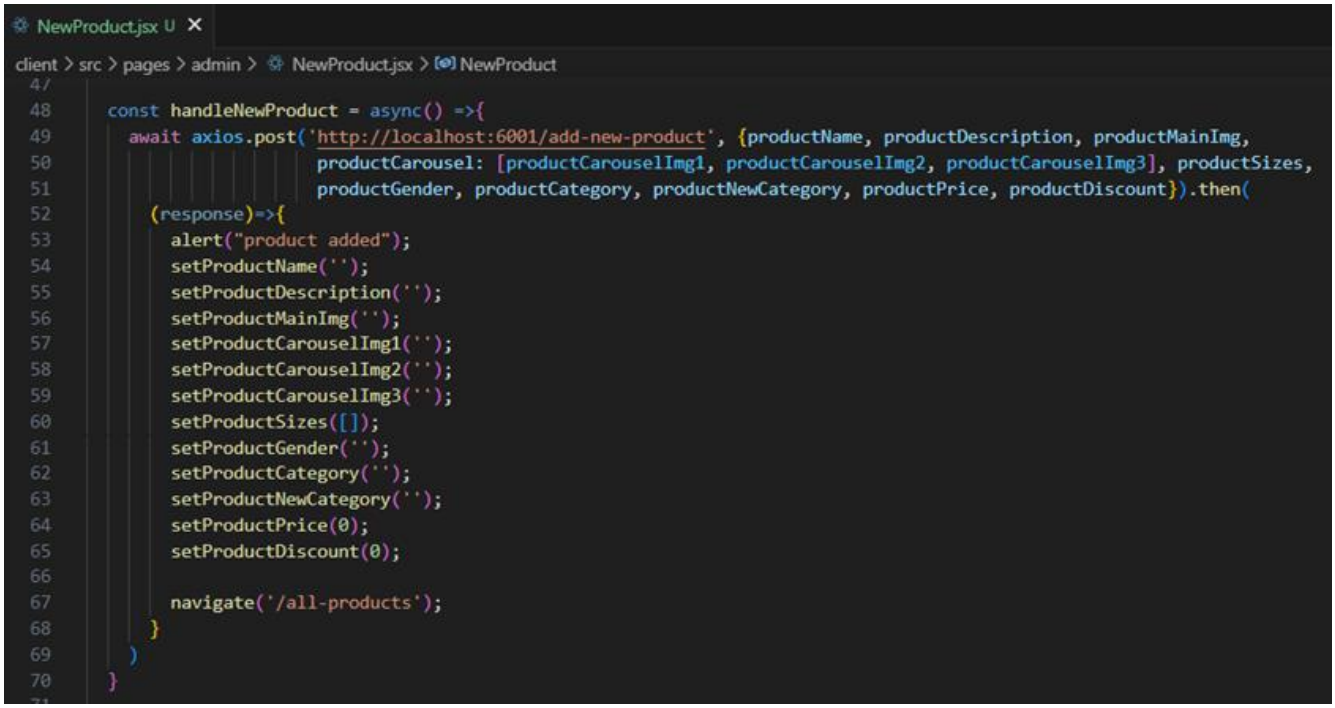
· Backend

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.

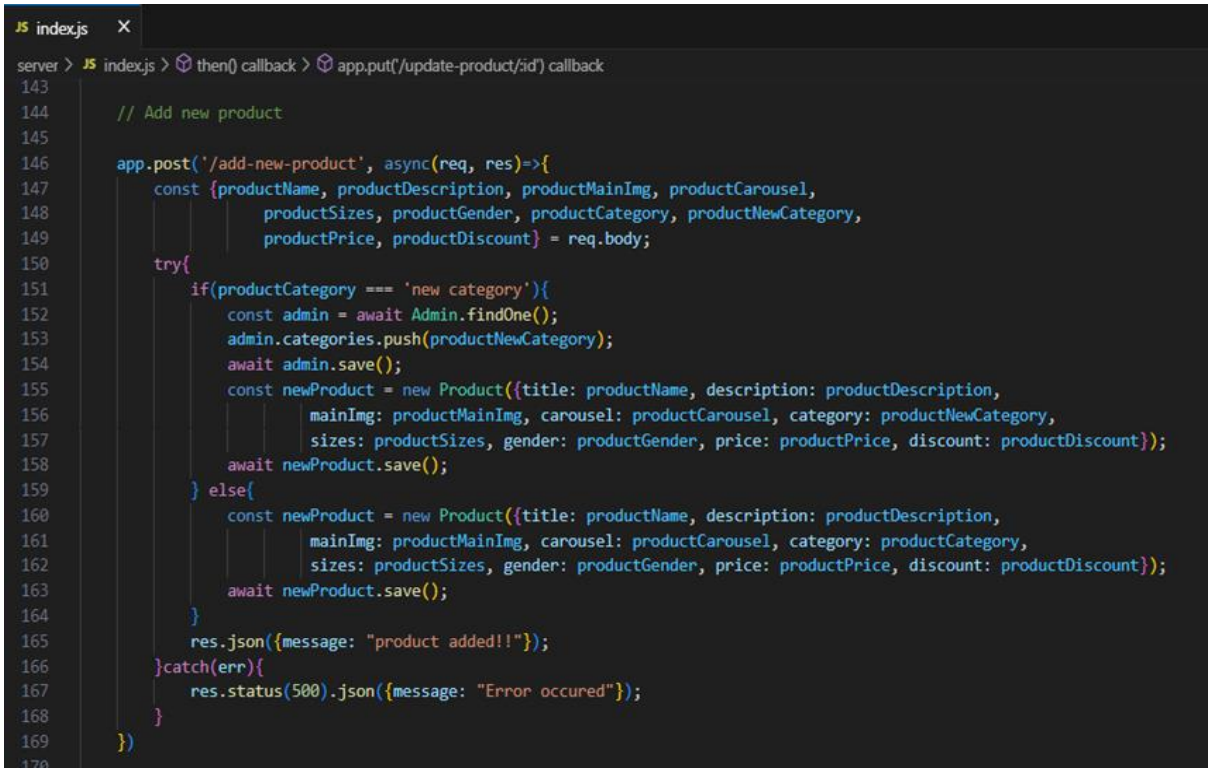
Add new product:

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

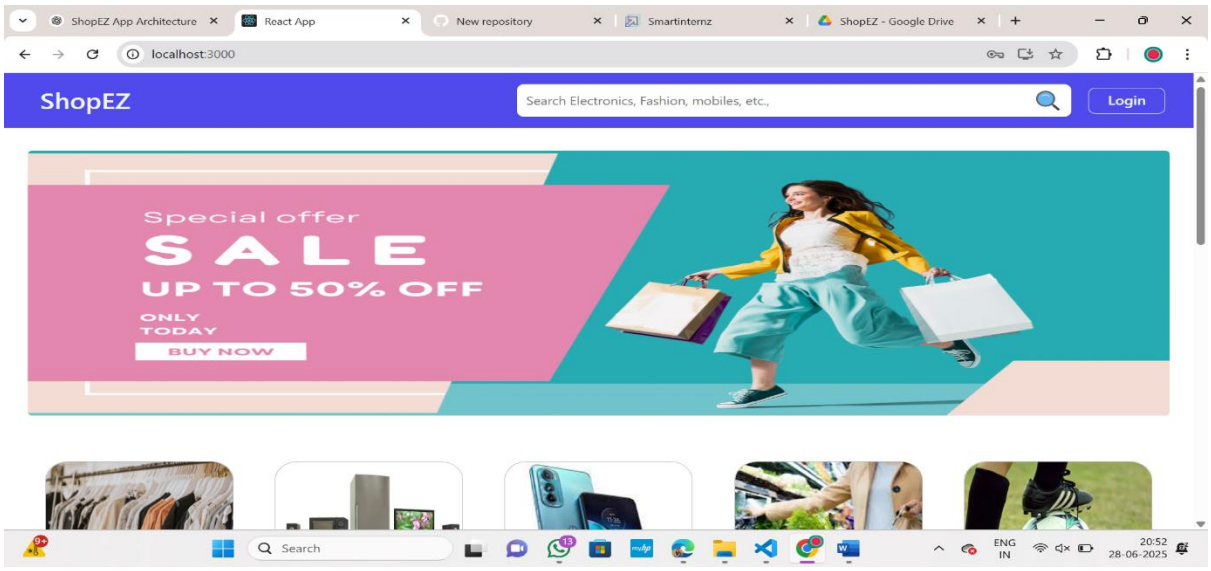
o Frontend:

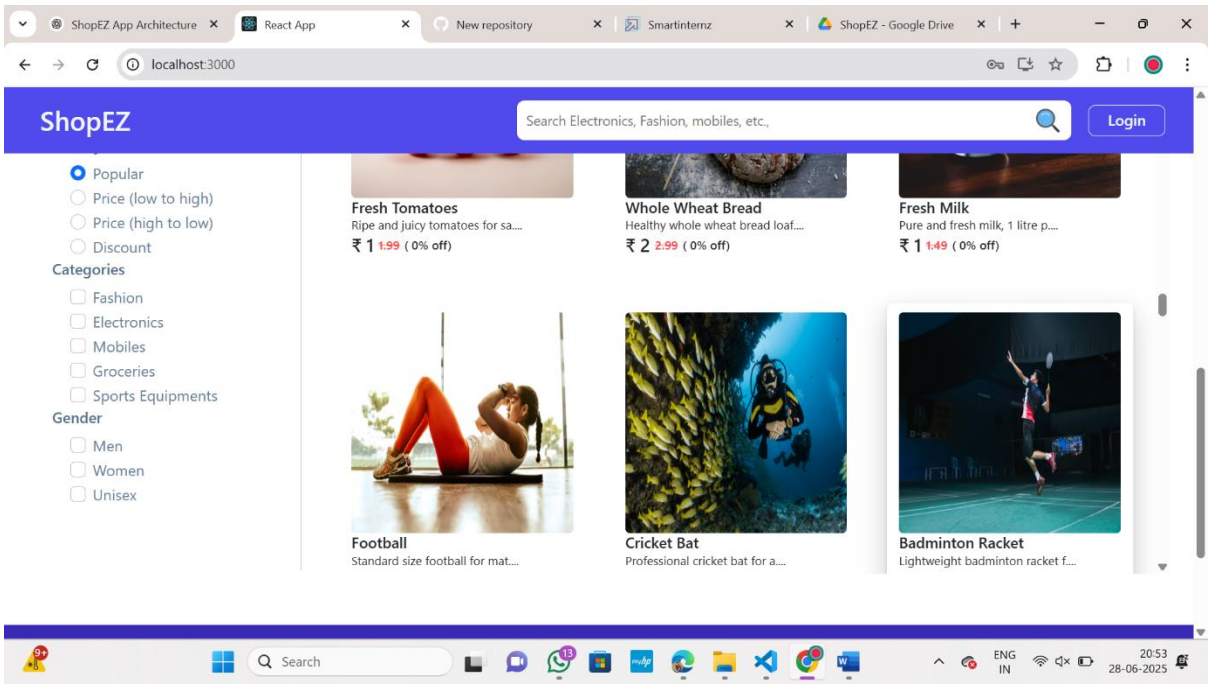


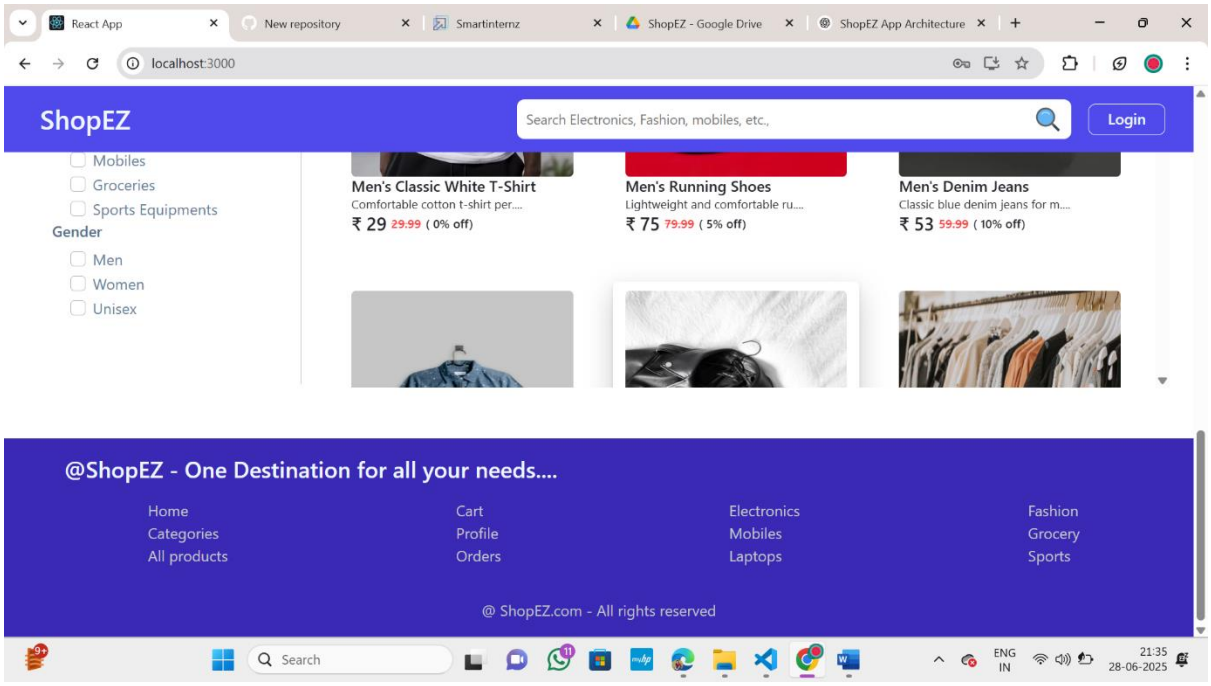
Backend:

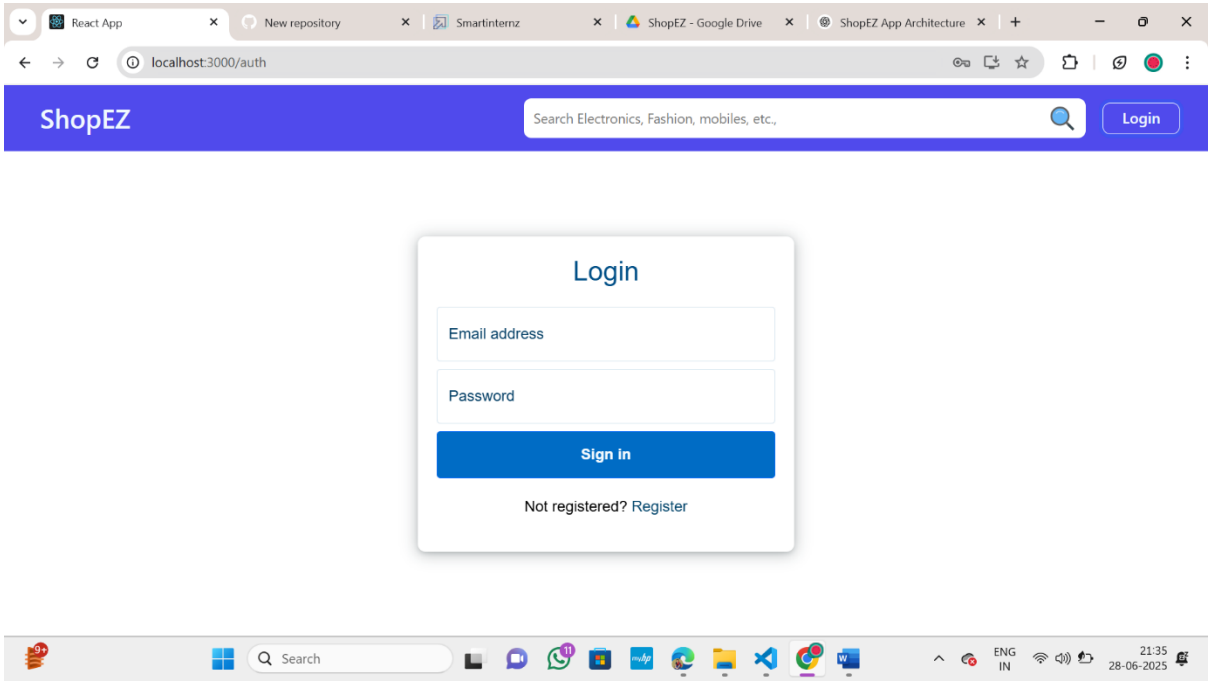

index.js
server > index.js > then() callback > app.put('/update-product/:id') callback
143
144 // Add new product
145
146 app.post('/add-new-product', async(req, res)=>{
147 const {productName, productDescription, productMainImg, productCarousel,
148 productSizes, productGender, productCategory, productNewCategory,
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 })

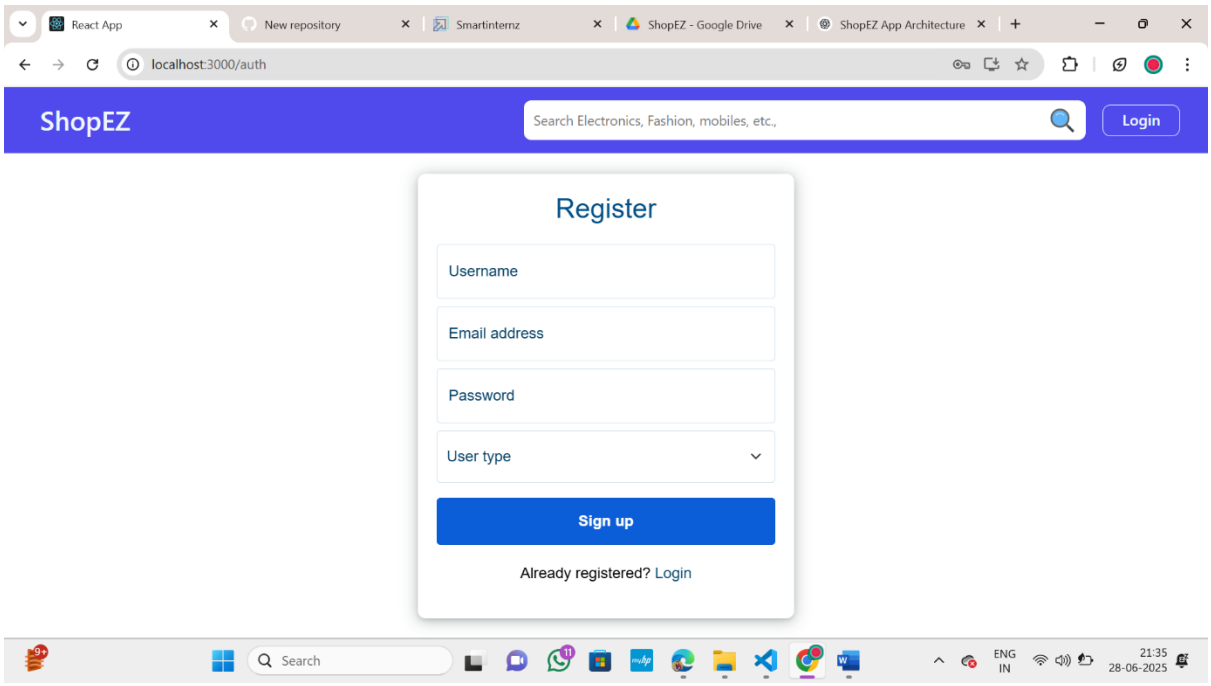
11. Screenshots or Demo

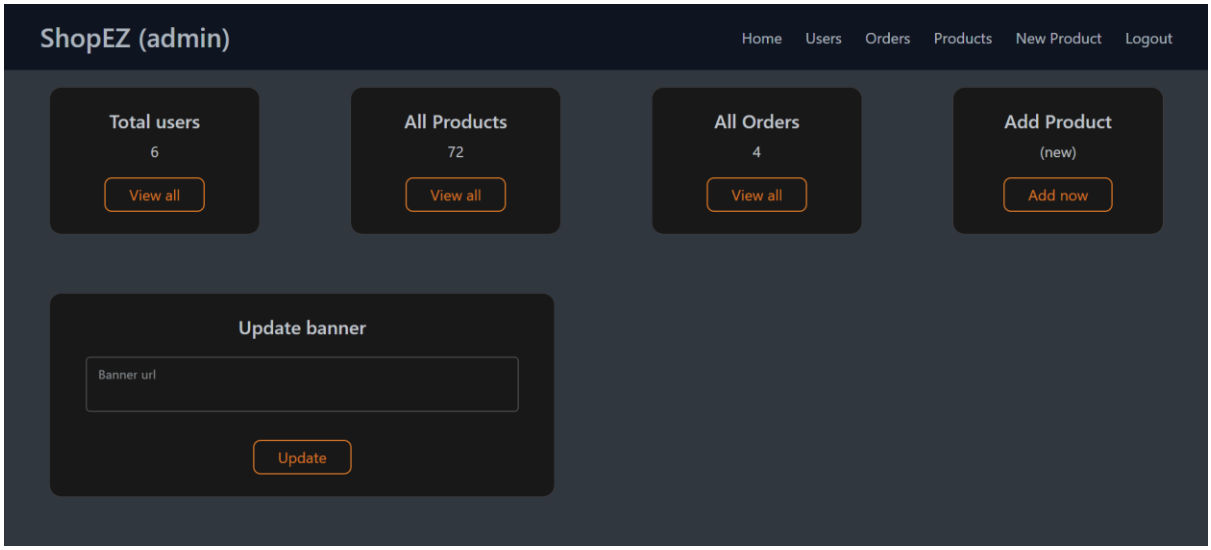





React App
New repository
Smartinternz
ShopEZ - Google Drive
ShopEZ App Architecture
localhost:3000
ShopEZ
Search Electronics, Fashion, mobiles, etc.,
Login
Mobiles
Groceries
Sports Equipments
Gender
Men
Women
Unisex
Men's Classic White T-Shirt
Comfortable cotton t-shirt per....
₹ 29 29.99 (0% off)
Men's Running Shoes
Lightweight and comfortable ru....
₹ 75 79.99 (5% off)
Men's Denim Jeans
Classic blue denim jeans for m....
₹ 53 59.99 (10% off)
@ShopEZ - One Destination for all your needs....
Home
Categories
All products
Cart
Profile
Orders
Electronics
Mobiles
Laptops
Fashion
Grocery
Sports
@ ShopEZ.com - All rights reserved
Search
ENG
IN
21:35
28-06-2025


React App
New repository
Smartinternz
ShopEZ - Google Drive
ShopEZ App Architecture
localhost:3000/auth
ShopEZ
Search Electronics, Fashion, mobiles, etc.,
Login
Login
Email address
Password
Sign in
Not registered? Register
Search
ENG
IN
21:35
28-06-2025


React App
New repository
Smartinternz
ShopEZ - Google Drive
ShopEZ App Architecture
localhost:3000/auth
ShopEZ
Search Electronics, Fashion, mobiles, etc.,
Login
Register
Username
Email address
Password
User type
Sign up
Already registered? Login
Search
ENG
IN
21:35
28-06-2025


ShopEZ (admin)
Home
Users
Orders
Products
New Product
Logout
Total users
6
View all
All Products
72
View all
All Orders
4
View all
Add Product
(new)
Add now
Update banner
Banner url
Update

ShopEZ (admin)

Home Users Orders Products New Product Logout

All Users

User Id	User Name	Email Address	Orders
685f9a96497bc15ee09b4627	user1	user1@example.com	0
685f9a96497bc15ee09b4629	user2	user2@example.com	0
685f9a96497bc15ee09b462b	user3	user3@example.com	0

ShopEZ (admin)

Home U

Orders

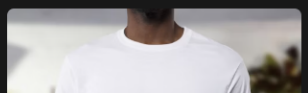
Men's Classic White T-Shirt

Size: M **Quantity:** 1 **Price:** ₹ 29 **Payment method:** netbanking

UserId: 685f91f03e480abce7d25cc5 **Name:** kollavivek **Email:** kollavivek20@g

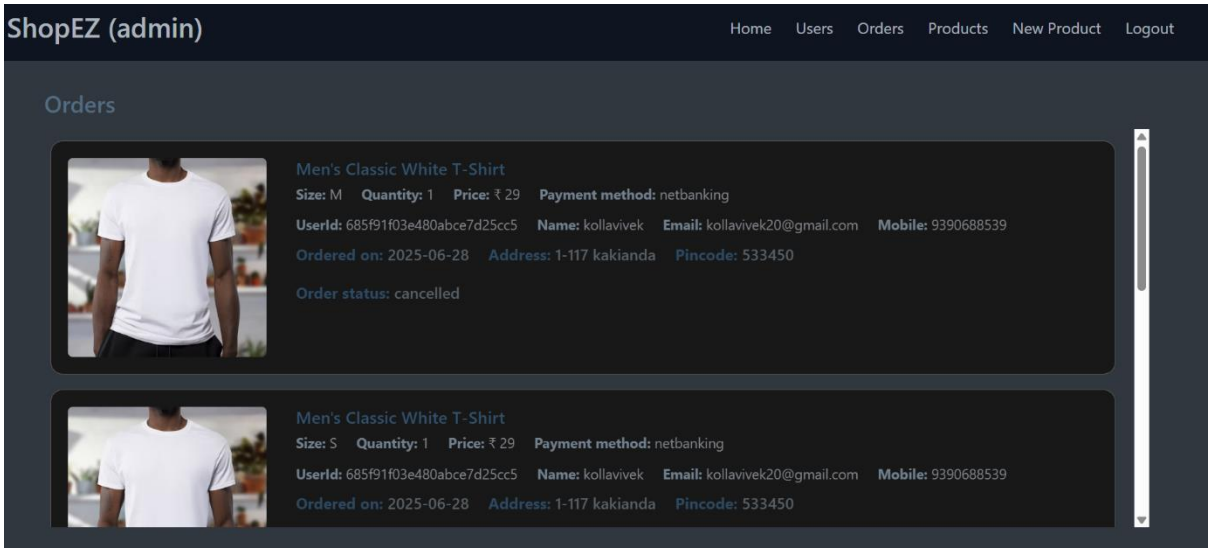
Ordered on: 2025-06-28 **Address:** 1-117 kakianda **Pincode:** 533450

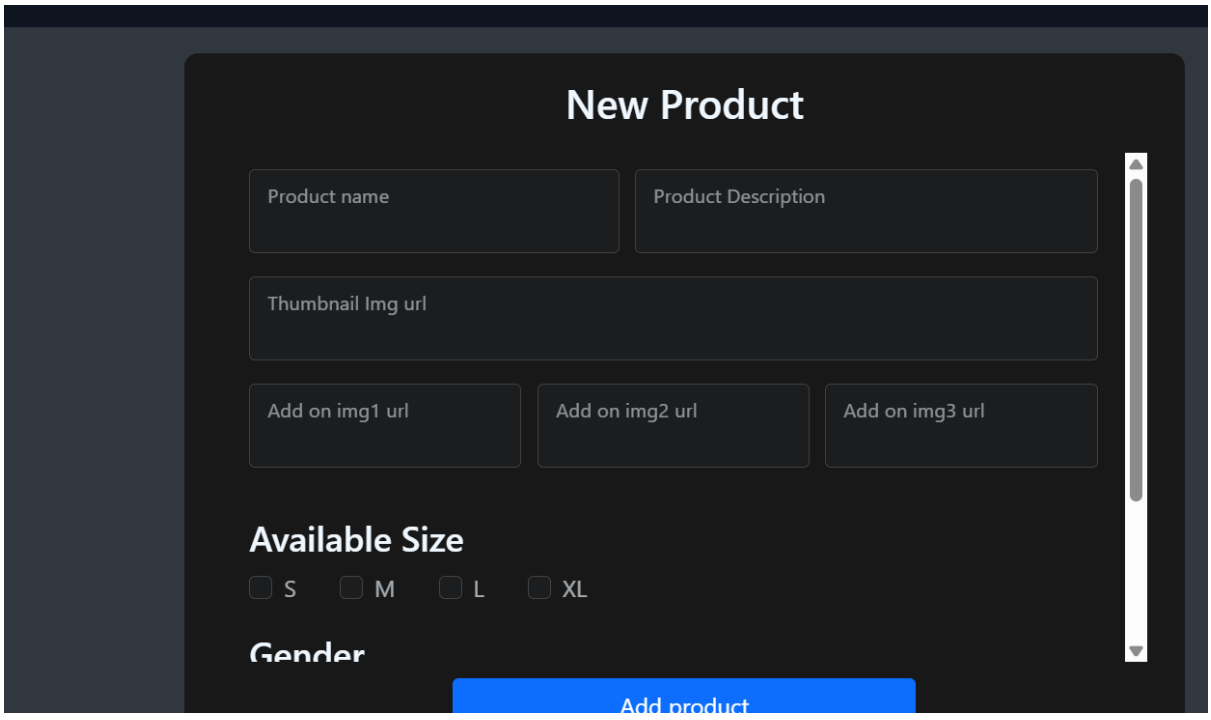
Order status: cancelled



Men's Classic White T-Shirt

Size: S **Quantity:** 1 **Price:** ₹ 29 **Payment method:** netbanking


ShopEZ (admin)
Home
Users
Orders
Products
New Product
Logout
Orders
Men's Classic White T-Shirt
Size: M Quantity: 1 Price: ₹ 29 Payment method: netbanking
UserId: 685f91f03e480abce7d25cc5 Name: kollavivek Email: kollavivek20@gmail.com Mobile: 9390688539
Ordered on: 2025-06-28 Address: 1-117 kakianda Pincode: 533450
Order status: cancelled
Men's Classic White T-Shirt
Size: S Quantity: 1 Price: ₹ 29 Payment method: netbanking
UserId: 685f91f03e480abce7d25cc5 Name: kollavivek Email: kollavivek20@gmail.com Mobile: 9390688539
Ordered on: 2025-06-28 Address: 1-117 kakianda Pincode: 533450


New Product
Product name
Product Description
Thumbnail Img url
Add on img1 url
Add on img2 url
Add on img3 url
Available Size
S
M
L
XL
Gender
Add product

12. Known Issues

→ Responsive Design Limitations

- Some pages (e.g., admin dashboard or product management views) may not render perfectly on smaller mobile screens. Additional CSS adjustments are planned to improve mobile responsiveness.

→ Error Handling

- Certain API error messages (such as database connection failures) are generic. Detailed error reporting and logging mechanisms need to be enhanced.

→ Admin Role Management

- The current admin role system is basic. There is no support for multiple admin levels (e.g., super admin vs regular admin). This could lead to limited control granularity.

→ No Email Notifications

- Features like email order confirmations or password reset emails are not yet implemented. These are planned for future versions.

→ Cart Persistence

- The shopping cart is cleared on logout or browser refresh, as persistent cart storage (e.g., linked to user account in the database) is not yet implemented.

13. Future Enhancements

→ **Persistent Cart Storage**

- Implement cart persistence by saving cart items to the database so users can retain their cart across sessions and devices.

→ **Email Notifications**

- Add features to send email confirmations for orders, password resets, and promotional offers.

→ **Advanced Admin Roles**

- Introduce role-based admin levels (e.g., super admin, product manager, order manager) for better control and security.

→ **Payment Gateway Integration**

- Integrate real payment gateways (e.g., Razorpay, Stripe, PayPal) for secure and flexible payment options.

→ **Search and Filter Improvements**

- Enhance product discovery with advanced search options, filters, and sorting features (e.g., price range, rating, category).

→ **User Profile Enhancements**

- Allow users to update profile pictures, addresses, and preferences.

→ **Analytics Dashboard (Advanced)**

- Expand analytics for admins to include detailed sales trends, customer demographics, and inventory reports.

→ **Mobile App Version**

- Develop a mobile application (using React Native or Flutter) to complement the web app and reach a wider audience.