ShopEZ: E-commerce Application

INTRODUCTION

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

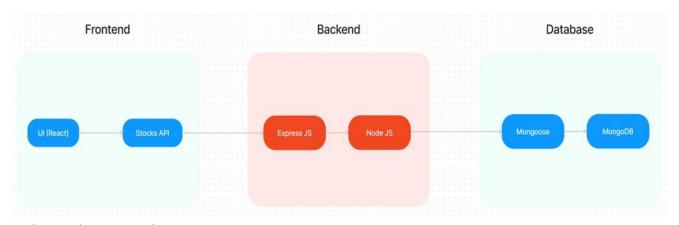
Sarah, a busy professional, is scrambling to find the perfect birthday gift for her best friend, Emily. She knows Emily loves fashion accessories, but with her hectic schedule, she hasn't had time to browse through multiple websites to find the ideal present. Feeling overwhelmed, Sarah turns to ShopEZ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



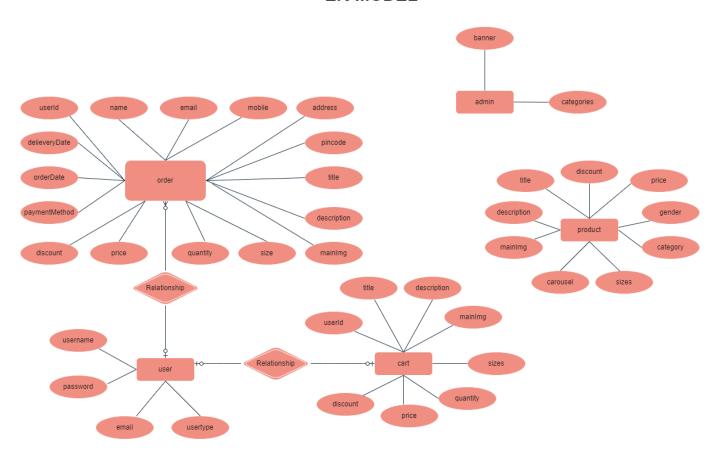
In this architecture diagram:

- 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

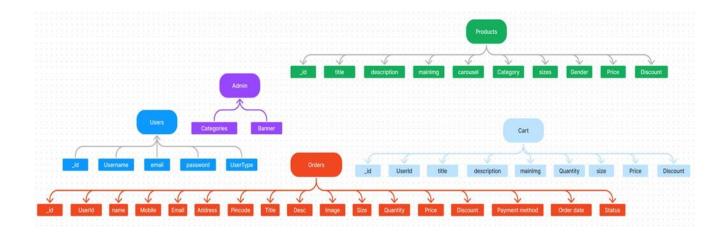
Users, cart, Orders and Product.

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:

- 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://reactis.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/nodeis-

mongooseis-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/Jyothiakshaya/shopEZ-E-commerce

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:

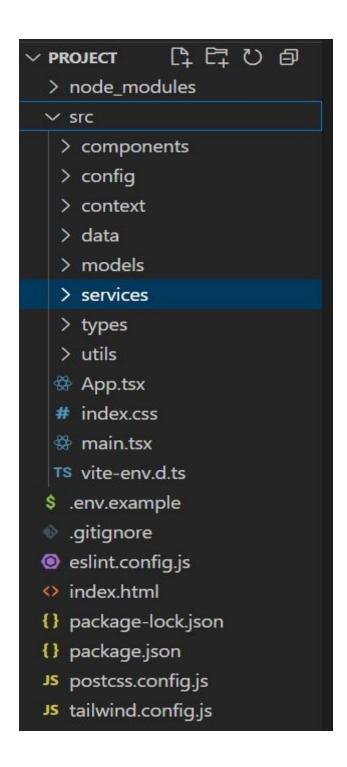
- Users start by registering for an account.
- After registration, they can log in with their credentials.
- Once logged in, they can check for the available products in the platform. Users can add the products they wish to their carts and order.
- They can then proceed by entering address and payment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

2. Admin Flow:

Admins start by logging in with their credentials.

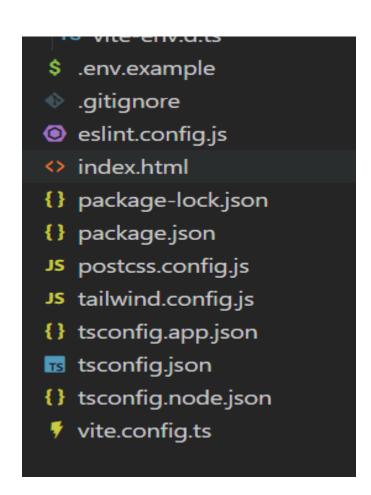
• Once logged in, they are directed to the Admin Dashboard. Admins can access the users list, products, orders, etc.

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-is-on-windows/

• Git.

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

2. Create project folders and files:

Create files on the format of jsx

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 ROHyZI2vRORggrYRio2gXS/view?usp=sharing

Reference Image:

```
EXPLORER
                                {} package.json
                                                 JS index.js X
∨ SHOPEZ
                                server > JS index.js > ...
                                       import express from "express";

✓ client

✓ server

                                       const app = express();
 > node_modules
                                      app.use(express.json());
 JS index.js
 {} package-lock.json
                                       app.listen(3001, () => {
 {} package.json
                                        console.log("App server is running on port 3001");
                                 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
                               PS D:\shopEZ> cd server
                               ○ PS D:\shopEZ\server> node index.js
                                 App server is running on port 3001
```

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:

```
import mongoose, { ConnectOptions } from 'mongoose';
const MONGODB_URI =
 (typeof process !== 'undefined' && (process as any).env?.MONGODB_URI) ||
 (typeof process !== 'undefined' && (process as any).env?.REACT_APP_MONGODB_URI) ||
 'mongodb+srv://jyothiakshaya12:<123>@cluster0.do6jpy2.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0';
export const connectDB = async () => {
   const options: ConnectOptions = {
    serverSelectionTimeoutMS: 5000,
     socketTimeoutMS: 45000,
     family: 4,
     bufferCommands: false,
   const conn = await mongoose.connect(MONGODB_URI, options);
   console.log(`MongoDB Atlas Connected: ${conn.connection.host}`);
   console.log(`Database: ${conn.connection.name}`);
   mongoose.connection.on('connected', () => {
     console.log('Mongoose connected to MongoDB Atlas');
   mongoose.connection.on('error', (err: any) => {
     console.error('Mongoose connection error:', err);
```

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-gQK3DO6wr7ooAfJYIQwnem ZoF/view?usp=sharing

Reference Image:

```
package-lock.json
{} package-lock.json > ...
        {
    "name": "shopez-ecommerce",
    "" "a a a",
          "version": "0.0.0",
          "lockfileVersion": 3,
          "requires": true,
          "packages": {
              "name": "shopez-ecommerce",
               "version": "0.0.0",
               "dependencies": {
                "axios": "^1.6.0",
                "lucide-react": "^0.344.0",
                "mongodb": "^6.17.0", "mongoose": "^8.0.0",
                 "react": "^18.3.1",
                "react-dom": "^18.3.1",
                 "react-router-dom": "^6.26.0"
Problems 15 Output Debug Console Terminal Ports
  run `npm fund` for details
7 vulnerabilities (2 low, 4 moderate, 1 high)
To address all issues, run:
  npm audit fix
Run `npm audit` for details.
PS C:\Users\jyoth\OneDrive\Desktop\project>
```

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 unctionalities
- Add necessary authentication and authorization checks to ensure only authorized admins can access these routes.
- Implement error handling middleware to catch and handle any errors that occur during the API requests.

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 X
server > JS Schema.js > [0] productSchema
  1 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}
```

```
IS Schema.js X
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},
           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},
      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.

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

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.

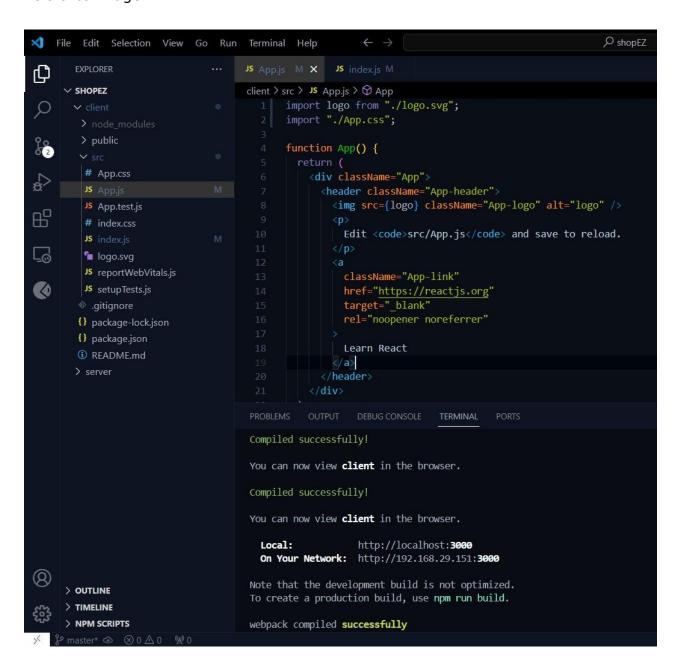
Reference Video Link:

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:

Registration page:

```
JS GeneralContext.js U X
client > src > context > JS GeneralContext.js > [@] GeneralContextProvider > [@] logout
         const inputs = {username, email, usertype, password};
         const register = async () =>{
                await axios.post('http://localhost:6001/register', inputs)
                .then( async (res)=>{
                     localStorage.setItem('userId', res.data._id);
                    localStorage.setItem('userType', res.data.usertype);
localStorage.setItem('username', res.data.username);
                    localStorage.setItem('email', res.data.email);
                    if(res.data.usertype === 'customer'){
                         navigate('/');
                     } else if(res.data.usertype === 'admin'){
                         navigate('/admin');
                }).catch((err) =>{
                    alert("registration failed!!");
                    console.log(err);
            }catch(err){
                console.log(err);
```

Logout:

```
GeneralContext.jsx U X

client > src > context >  GeneralContext.jsx >  GeneralContextProvider >  GeneralContextProvi
```

All Products (User):

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

```
JS index.js × ₩ ProductCard.tsx ×
src > C:\Users\jyoth\OneDrive\Desktop\project\src\config\index.js
     import React from 'react';
     import { Heart, Star, ShoppingCart, ShoppingBag } from 'lucide-react';
     import { Product } from '../types';
     import { useApp } from '../context/AppContext';
     interface ProductCardProps {
       product: Product;
       onProductClick: (product: Product) => void;
       onOrderClick: (product: Product) => void;
     export default function ProductCard({ product, onProductClick, onOrderClick }: ProductCardProps) {
       const { state, dispatch } = useApp();
        const isFavorite = state.favorites.includes(product.id);
        const handleAddToCart = (e: React.MouseEvent) => {
          e.stopPropagation();
          dispatch({ type: 'ADD_TO_CART', payload: product });
        const handleOrderNow = (e: React.MouseEvent) => {
         e.stopPropagation();
onOrderClick(product);
        const handleToggleFavorite = (e: React.MouseEvent) => {
           e.stopPropagation();
           dispatch({ type: 'TOGGLE_FAVORITE', payload: product.id });
```

```
export default function ProductCard({ product, onProductClick, onOrderClick }: ProductCardProps) {
     className="bg-white rounded-lg shadow-md hover:shadow-xl transition-all duration-300 cursor-pointer g
     onClick={() => onProductClick(product)}
      <div className="relative overflow-hidden">
         src={product.image}
         alt={product.name}
         className="w-full h-48 object-cover group-hover:scale-105 transition-transform duration-300"
         onClick={handleToggleFavorite}
         className={`absolute top-3 right-3 p-2 rounded-full transition-all duration-200 ${
           isFavorite
             ? 'bg-red-500 text-white'
             : 'bg-white text-gray-600 hover:bg-red-500 hover:text-white'
         <Heart className="h-4 w-4" fill={isFavorite ? 'currentColor' : 'none'} />
        {discountPercentage > 0 && (
          <div className="absolute top-3 left-3 bg-red-500 text-white px-2 py-1 rounded-md text-sm font-sem</pre>
           -{discountPercentage}%
        {!product.inStock && (
         <div className="absolute inset-0 bg-black bg-opacity-50 flex items-center justify-center">
```

Filtering Products:

```
src > components > 🖶 FilterSidebar.tsx > ...
       import React from 'react';
       import { categories } from '../data/products';
      interface FilterSidebarProps {
        selectedCategory: string;
        onCategoryChange: (category: string) => void;
        priceRange: [number, number];
        onPriceRangeChange: (range: [number, number]) => void;
        sortBy: string;
        onSortChange: (sort: string) => void;
 12
       export default function FilterSidebar({
 13
        selectedCategory,
        onCategoryChange,
        priceRange,
 17
        onPriceRangeChange,
        sortBy,
        onSortChange
       }: FilterSidebarProps) {
         return (
           <div className="bg-white p-6 rounded-lg shadow-md">
             <h3 className="text-lg font-semibold mb-4">Filters</h3>
             {/* Categories */}
             <div className="mb-6">
               <h4 className="font-medium mb-3">Categories</h4>
               <div className="space-y-2">
 29
                 {categories.map((category) => (
```

```
> components > 🤀 FilterSidebar.tsx >
   export default function FilterSidebar({
         {/* Price Range */}
         <div className="mb-6">
           <h4 className="font-medium mb-3">Price Range</h4>
           <div className="space-y-3">
                 type="range"
                 max="1000"
                value={priceRange[1]}
onChange={(e) => onPriceRangeChange([priceRange[0], Number(e.target.value)])}
                 className="w-full h-2 bg-gray-200 rounded-lg appearance-none cursor-pointer
             <div className="flex justify-between text-sm text-gray-600">
              <span>${priceRange[0]}</span>
               <span>${priceRange[1]}</span>
         <div className="mb-6">
           <h4 className="font-medium mb-3">Sort By</h4>
             value={sortBy}
              onChange={(e) => onSortChange(e.target.value)}
```

```
export default function FilterSidebar({
     <div className="mb-6">
      <h4 className="font-medium mb-3">Sort By</h4>
        value={sortBy}
        onChange={(e) => onSortChange(e.target.value)}
        className="w-full p-2 border border-gray-300 rounded-lg focus:ring-2 focus:ring-primary-500 focus
        <option value="featured">Featured</option>
        <option value="price-low">Price: Low to High</option>
        <option value="price-high">Price: High to Low</option>
        <option value="rating">Highest Rated</option>
        <option value="newest">Newest</option>
    {/* Clear Filters */}
      onClick={() => {
        onCategoryChange('all');
        onPriceRangeChange([0, 1000]);
        onSortChange('featured');
      className="w-full bg-gray-100 text-gray-700 py-2 px-4 rounded-lg hover:bg-gray-200 transition-color
      Clear All Filters
```

Add product to cart:

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

```
import React from 'react';
import { X, Plus, Minus, Trash2 } from 'lucide-react';
import { useApp } from '../context/AppContext';
interface CartProps {
 isOpen: boolean;
 onClose: () => void;
 onCheckout: () => void;
export default function Cart({ isOpen, onClose, onCheckout }: CartProps) {
 const { state, dispatch } = useApp();
 if (!isOpen) return null;
 const totalAmount = state.cart.reduce(
    (total, item) => total + item.product.price * item.quantity,
  const updateQuantity = (id: string, quantity: number) => {
   if (quantity <= 0) {</pre>
      dispatch({ type: 'REMOVE_FROM_CART', payload: id });
      dispatch({ type: 'UPDATE_CART_QUANTITY', payload: { id, quantity } });
 return (
```

Order products:

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

· Frontend

```
return (
 <div className="fixed inset-0 z-50 overflow-hidden">
   <div className="absolute inset-0 bg-black bg-opacity-50" onClick={onClose}></div>
   <div className="absolute right-0 top-0 h-full w-full max-w-2xl bg-white shadow-xl">
     <div className="flex flex-col h-full">
      <div className="flex items-center justify-between p-6 border-b">
        <h2 className="text-xl font-semibold">My Orders</h2>
          onClick={onClose}
          className="p-2 hover:bg-gray-100 rounded-lg transition-colors"
          <X className="h-5 w-5" />
       <div className="flex-1 overflow-y-auto p-6">
        {loading ? (
          <div className="text-center py-8">
            <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary-600 mx-auto"></div</pre>
            Loading orders...
         ) : orders.length === 0 ? (
```

```
<div className="flex-1 overflow-y-auto p-6">
 {loading ? (
   <div className="text-center py-8">
     <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-primary-600 mx-auto"></div>
     Loading orders...
 ) : orders.length === 0 ? (
   <div className="text-center py-12">
     <Package className="h-16 w-16 text-gray-300 mx-auto mb-4" />
     <h3 className="text-lg font-medium text-gray-900 mb-2">No orders yet</h3>
     When you place orders, they'll appear here.
   <div className="space-y-4">
     {orders.map((order) => (
       <div key={order.id} className="bg-gray-50 rounded-lg p-4 hover:bg-gray-100 transition-colors">
         <div className="flex items-center justify-between mb-3">
           <div className="flex items-center space-x-2">
            {getStatusIcon(order.status)}
             <span className="font-medium">Order #{order.orderNumber}</span>
           <span className={`px-2 py-1 text-xs font-semibold rounded-full ${getStatusColor(order.status)}</pre>
            {order.status.charAt(0).toUpperCase() + order.status.slice(1)}
```

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.

```
export default function Orders({ isOpen, onClose }: OrdersProps) {
 const fetchOrders = async () => {
     const mockOrders: Order[] = [
         items: [
             product: {
               tags: ['gold', 'bracelet', 'elegant']
             quantity: 1
         total: 129.99,
         status: 'delivered',
         shippingAddress: {
           name: 'John Doe',
           street: '123 Main St',
           city: 'New York',
           state: 'NY',
           zipCode: '10001',
           country: 'United States'
         createdAt: '2024-01-15T10:30:00Z',
         updatedAt: '2024-01-18T14:20:00Z'
         id: 'ORD-002',
         userId: state.user.id,
         orderNumber: 'ORD-002',
          items: [
```

```
const getStatusIcon = (status: string) => {
  switch (status) {
   case 'pending':
     return <Clock className="h-5 w-5 text-yellow-500" />;
    case 'processing':
     return <Package className="h-5 w-5 text-blue-500" />;
   case 'shipped':
     return <Truck className="h-5 w-5 text-purple-500" />;
   case 'delivered':
     return <CheckCircle className="h-5 w-5 text-green-500" />;
    case 'cancelled':
     return <XCircle className="h-5 w-5 text-red-500" />;
   default:
     return <Clock className="h-5 w-5 text-gray-500" />;
const getStatusColor = (status: string) => {
  switch (status) {
   case 'pending':
     return 'bg-yellow-100 text-yellow-800';
   case 'processing':
     return 'bg-blue-100 text-blue-800';
   case 'shipped':
```

Add new product:

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

o Frontend:

```
import React from 'react';
import { Heart, Star, ShoppingCart, ShoppingBag } from 'lucide-react';
import { Product } from '../types';
import { useApp } from '../context/AppContext';
interface ProductCardProps {
 product: Product;
  onProductClick: (product: Product) => void;
 onOrderClick: (product: Product) => void;
export default function ProductCard({ product, onProductClick, onOrderClick }: ProductCardProps) {
  const { state, dispatch } = useApp();
  const isFavorite = state.favorites.includes(product.id);
  const handleAddToCart = (e: React.MouseEvent) => {
   e.stopPropagation();
    dispatch({ type: 'ADD_TO_CART', payload: product });
  const handleOrderNow = (e: React.MouseEvent) => {
   e.stopPropagation();
    onOrderClick(product);
  const handleToggleFavorite = (e: React.MouseEvent) => {
    e.stopPropagation();
    dispatch({ type: 'TOGGLE_FAVORITE', payload: product.id });
```

Backend:

```
JS index.js X
server > 5 index.js > ♥ then() callback > ♥ app.put('/update-product/:id') callback
          app.post('/add-new-product', async(req, res)=>{
              const {productName, productDescription, productMainImg, productCarousel,
                          productSizes, productGender, productCategory, productNewCategory,
                          productPrice, productDiscount} = req.body;
                  if(productCategory === 'new category'){
                      const admin = await Admin.findOne();
                      admin.categories.push(productNewCategory);
                      await admin.save();
                      const newProduct = new Product({title: productName, description: productDescription,
                              mainImg: productMainImg, carousel: productCarousel, category: productNewCategory,
                               sizes: productSizes, gender: productGender, price: productPrice, discount: productDiscount});
                      await newProduct.save();
                               mainImg: productMainImg, carousel: productCarousel, category: productCategory,
                               sizes: productSizes, gender: productGender, price: productPrice, discount: productDiscount});
                      await newProduct.save();
                  res.json({message: "product added!!"});
                  res.status(500).json({message: "Error occured"});
```

Model checkout:

```
xport default function CheckoutModal({ isOpen, onClose }: CheckoutModalProps) {
  const { state, dispatch } = useApp();
  const [step, setStep] = useState(1);
  const [customerInfo, setCustomerInfo] = useState<CustomerInfo>({{
      name: state.user?.name || '',
      phone: ''
    });
  const [shippingAddress, setShippingAddress] = useState<Address>({{
      name: '',
      street: '',
      city: '',
      state: '',
      country: 'United States'
    });
  const [paymentMethod, setPaymentMethod] = useState('card');
  const [isProcessing, setIsProcessing] = useState(false);
  if (!isOpen) return null;
  const totalAmount = state.cart.reduce(
      (total, item) => total + item.product.price * item.quantity,
      0
    );
  const generateOrderNumber = () => {
```

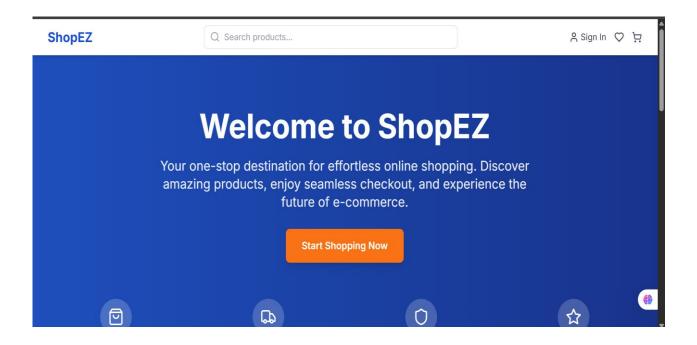
```
const generateOrderNumber = () => {
 return 'ORD-' + Math.random().toString(36).substr(2, 9).toUpperCase();
};
const validateStep1 = () => {
 return customerInfo.name && customerInfo.email && customerInfo.phone;
};
const validateStep2 = () => {
  return shippingAddress.name && shippingAddress.street &&
   shippingAddress.city && shippingAddress.state && shippingAddress.zipCode
};
const handleOrderComplete = async () => {
 setIsProcessing(true);
 try {
   // Simulate API call delay
   await new Promise(resolve => setTimeout(resolve, 2000));
    // Create order object
    const newOrder: Order = {
     id: Math.random().toString(36).substr(2, 9),
     userId: state.user?.id | '',
     orderNumber: generateOrderNumber(),
     items: [...state.cart],
```

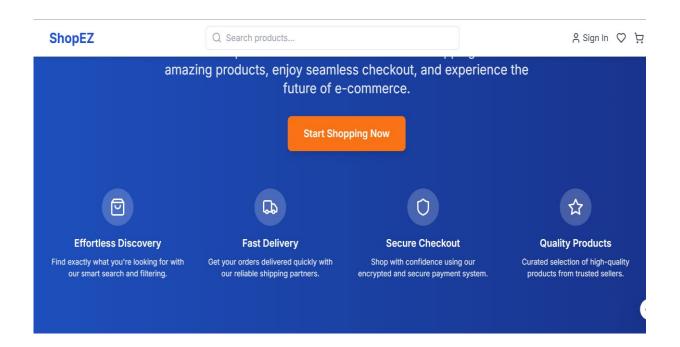
Seller dashboard:

```
<div className="min-h-screen bg-gray-50">
 <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
  <div className="mb-8">
    <h1 className="text-3xl font-bold text-gray-900">Seller Dashboard</h1>
    Welcome back, {state.user?.name}! Here's how your store is perform
  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">
    {stats.map((stat, index) => {
     const Icon = stat.icon;
     return (
       <div key={index} className="bg-white p-6 rounded-lg shadow-md">
        <div className="flex items-center justify-between">
           {stat.label}
           {stat.value}
           {stat.change} from last month
          <div className={`p-3 rounded-full bg-${stat.color}-100`}>
           <Icon className={`h-6 w-6 text-${stat.color}-600`} />
```

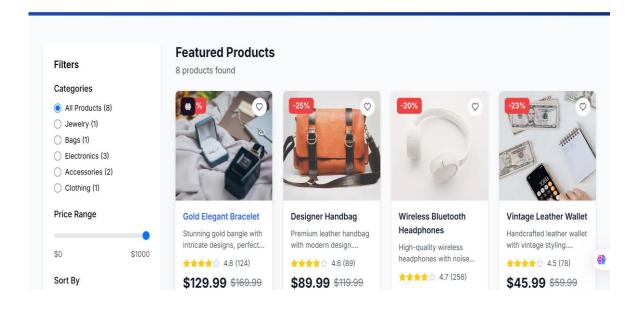
Demo UI images: ·

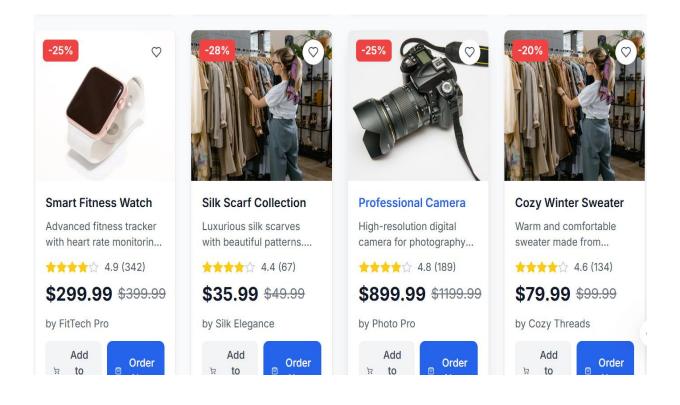
Landing page





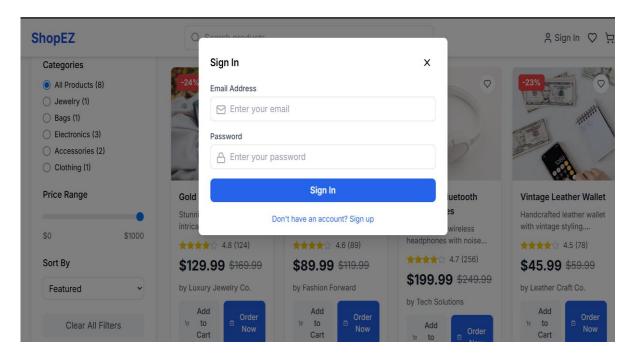
Products:



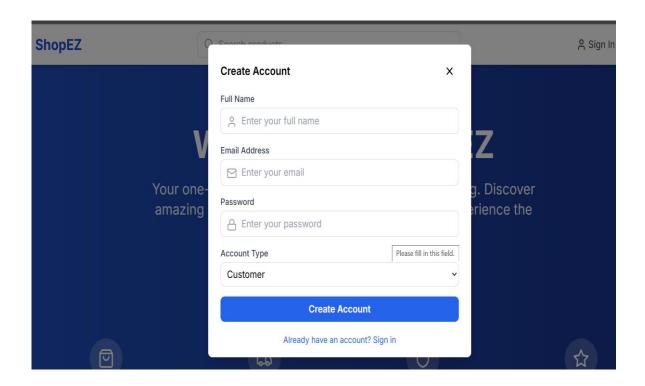


Authentication:

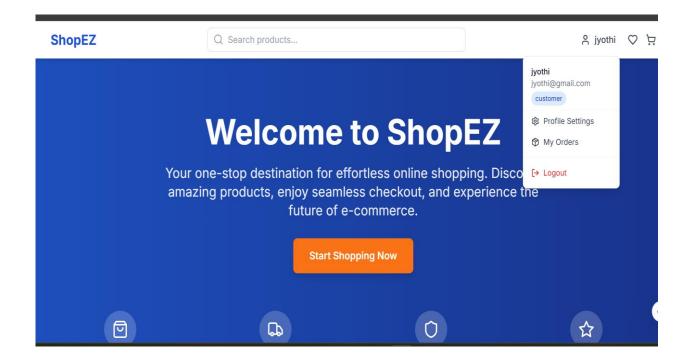
Sign in:



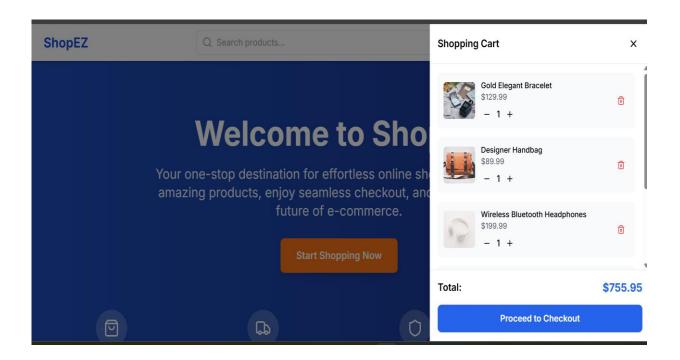
Register:



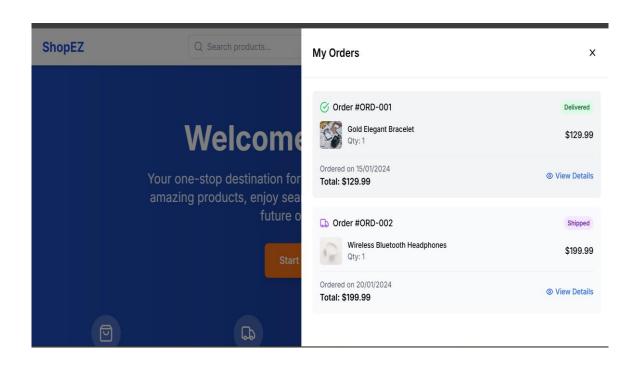
User Profile:

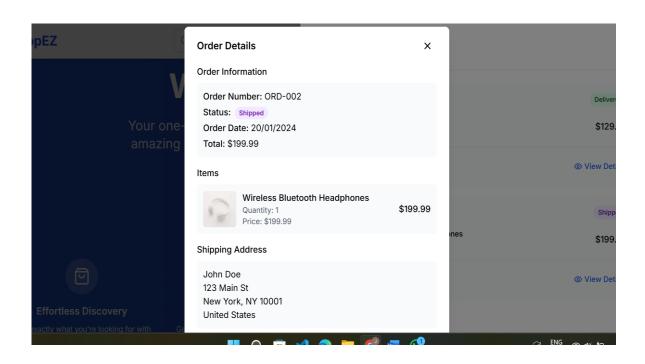


Cart:

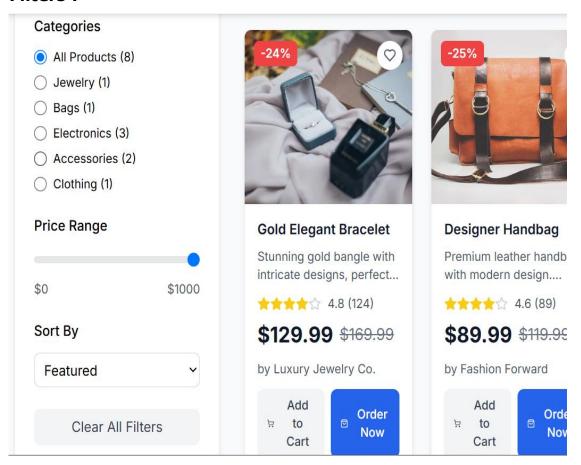


All orders:

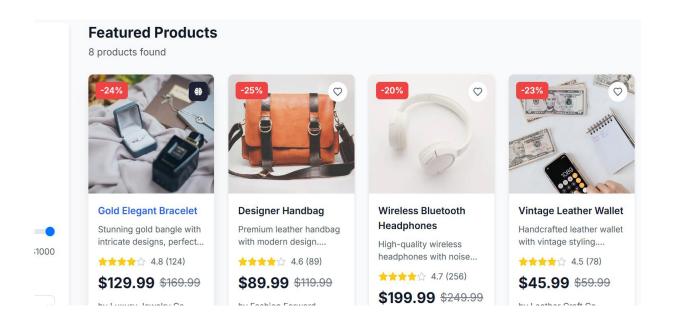




Filters:



All products page:



For any further doubts or help, please consider the GitHub repo, https://github.com/Jyothiakshaya/shopEZ-E-commerce

** Happy Coding **