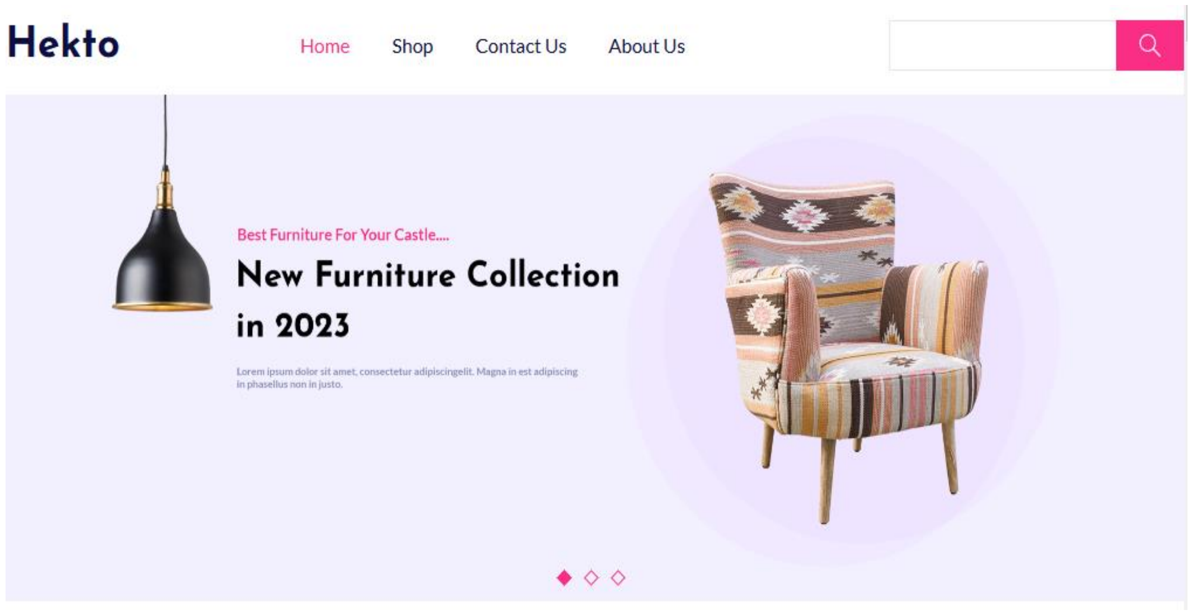
Name: Hammad Noor Khan

Roll No: 00457174

Day 4 - Building Dynamic Frontend Components for Your Marketplace [Bandage-E-Commerce]

Introduction

This report documents the entire process I followed to develop the dynamic frontend components for my e-commerce platform. The goal of this platform is to create a seamless shopping experience for users, featuring dynamic product listings, category filtering, search functionality, and other essential e-commerce features. Below, I outline each step of the development process, challenges I faced, solutions I implemented, and additional ideas for improvement.

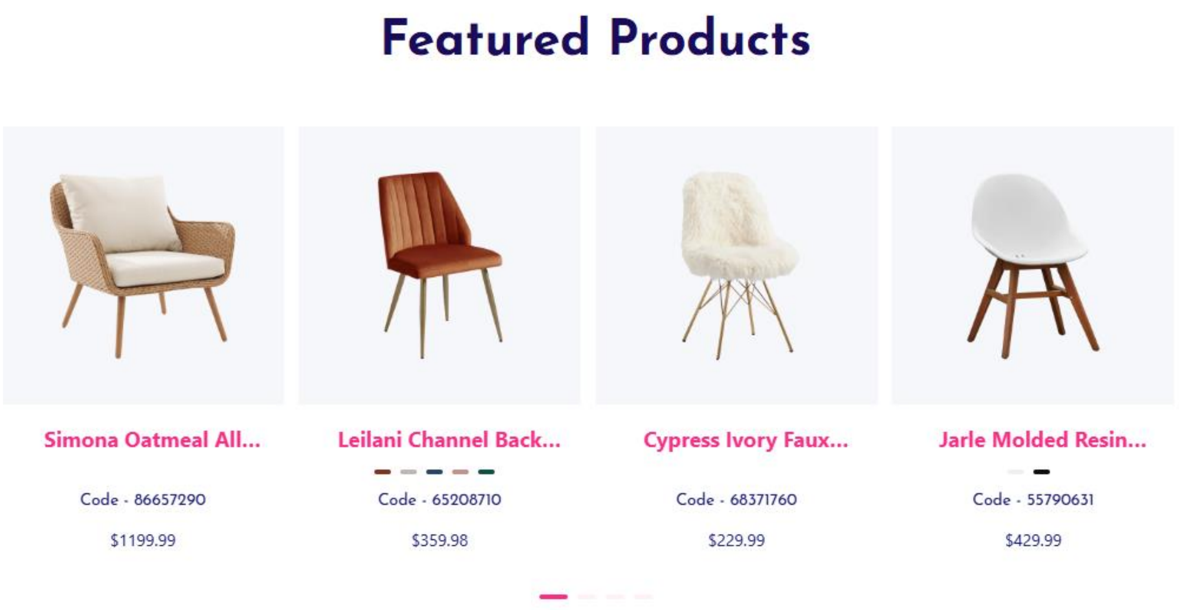

Hekto
Home
Shop
Contact Us
About Us
Best Furniture For Your Castle....
New Furniture Collection in 2023
Lorem ipsum dolor sit amet, consectetur adipiscingelit. Magna in est adipiscing in phasellus non in justo.

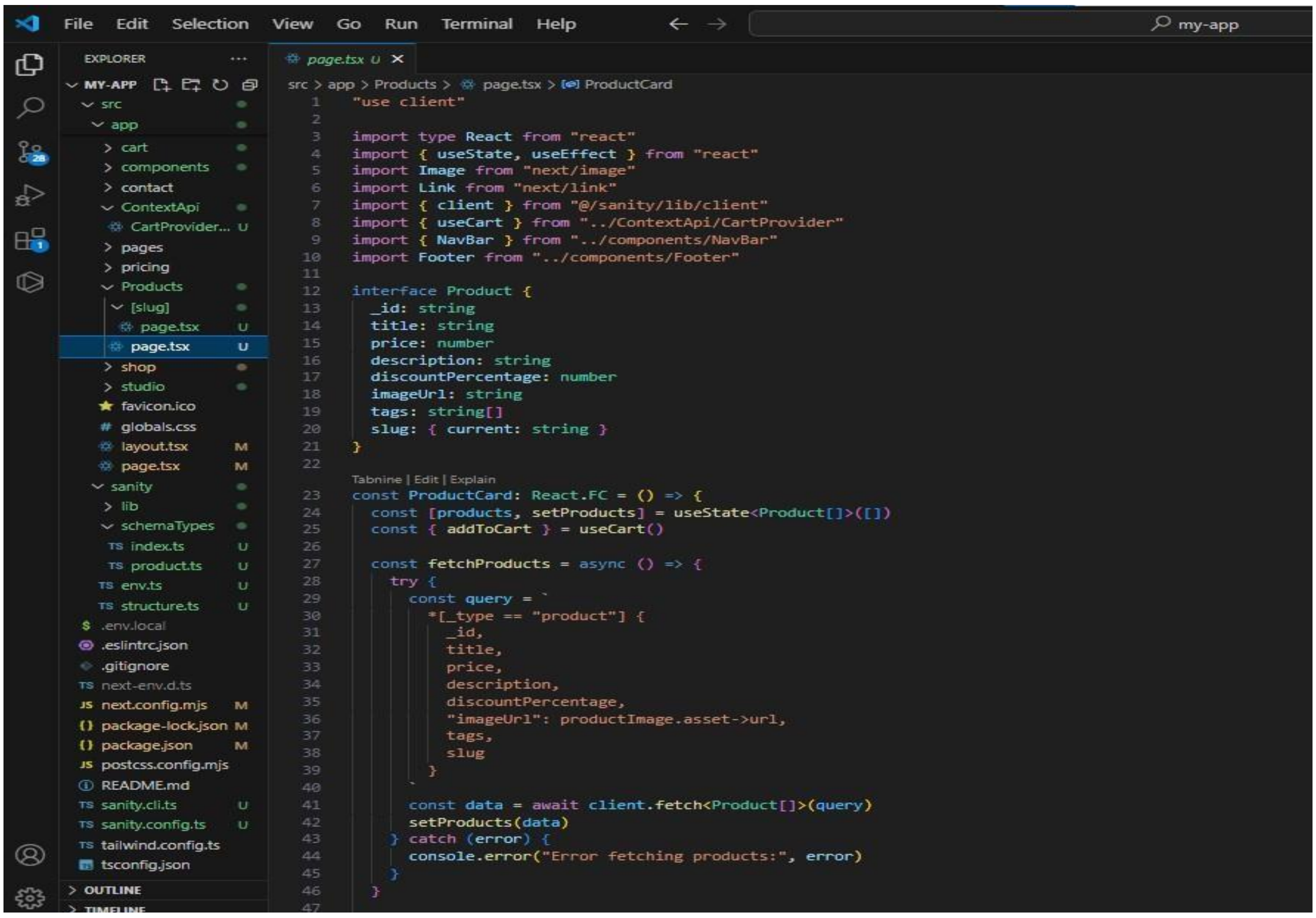
Process Overview

1. Product Card

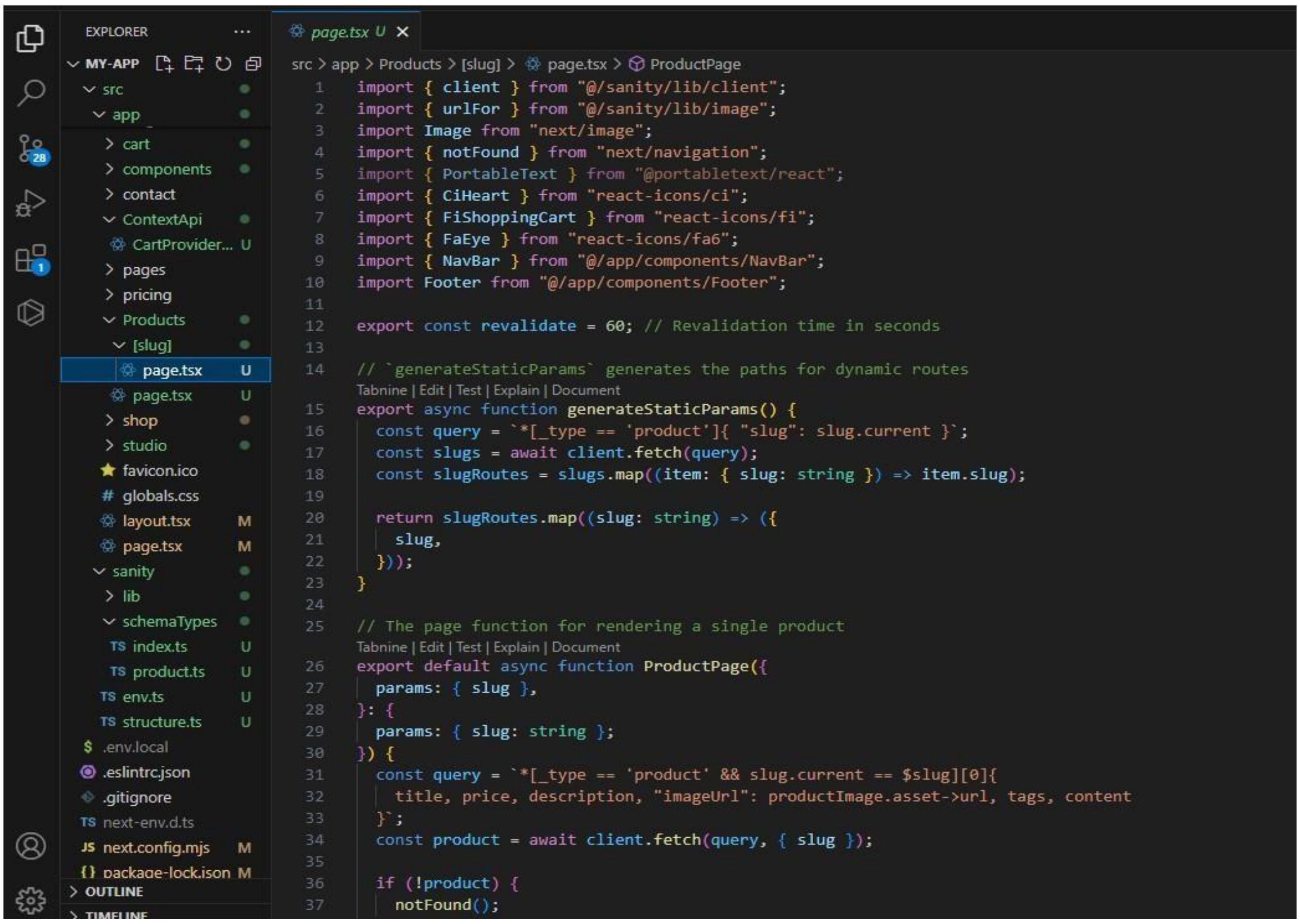
- **What I Did:**
  - Designed a ProductCard component to display details like name, price, and image.
  - Styled the cards using Tailwind CSS for responsiveness.
  - Added interactive button for “Add to Cart”.
- **Challenges I Faced:**
  - Handling inconsistent product image sizes.
- **Solution:**
  - Adjust it by Tailwind CSS properties to maintain uniformity.

- **Additional Features Can Be Added:**
    - Displayed a quick “View Details” button for a modal preview. ○ Highlighted items on sale with a discount badge.





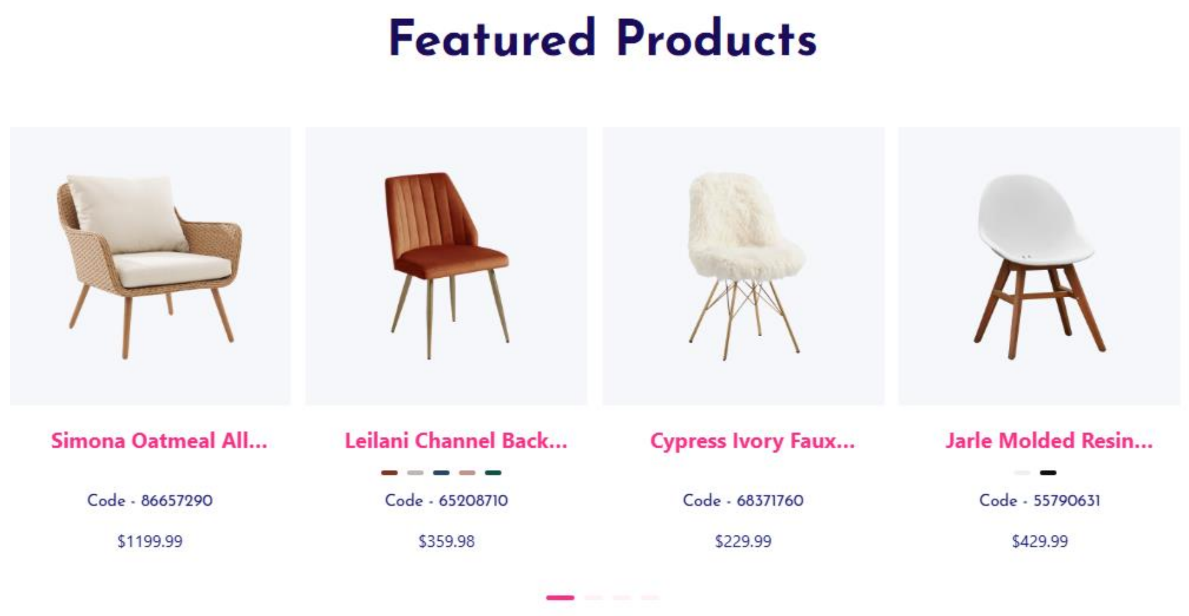
```
"use client"

import type React from "react"
import { useState, useEffect } from "react"
import Image from "next/image"
import Link from "next/link"
import { client } from "@/sanity/lib/client"
import { useCart } from "../ContextApi/CartProvider"
import { NavBar } from "../components/NavBar"
import Footer from "../components/Footer"

interface Product {
  _id: string
  title: string
  price: number
  description: string
  discountPercentage: number
  imageUrl: string
  tags: string[]
  slug: { current: string }
}

const ProductCard: React.FC = () => {
  const [products, setProducts] = useState<Product[]>([])
  const { addToCart } = useCart()

  const fetchProducts = async () => {
    try {
      const query = `
        *[_type == "product"] {
          _id,
          title,
          price,
          description,
          discountPercentage,
          "imageUrl": productImage.asset->url,
          tags,
          slug
        }
      `
      const data = await client.fetch<Product[]>(query)
      setProducts(data)
    } catch (error) {
      console.error("Error fetching products:", error)
    }
  }

```



```tsx
import { client } from "@/sanity/lib/client";
import { urlFor } from "@/sanity/lib/image";
import Image from "next/image";
import { notFound } from "next/navigation";
import { PortableText } from "@portabletext/react";
import { CiHeart } from "react-icons/ci";
import { FiShoppingCart } from "react-icons/fi";
import { FaEye } from "react-icons/fa6";
import { NavBar } from "@/app/components/NavBar";
import Footer from "@/app/components/Footer";

export const revalidate = 60; // Revalidation time in seconds

// `generateStaticParams` generates the paths for dynamic routes
export async function generateStaticParams() {
  const query = `*[_type == 'product']{ "slug": slug.current }`;
  const slugs = await client.fetch(query);
  const slugRoutes = slugs.map((item: { slug: string }) => item.slug);

  return slugRoutes.map((slug: string) => ({
    slug,
  }));
}

// The page function for rendering a single product
export default async function ProductPage({
  params: { slug },
}: {
  params: { slug: string };
}) {
  const query = `*[_type == 'product' && slug.current == $slug][0]{
    title, price, description, "imageUrl": productImage.asset->url, tags, content
  }`;
  const product = await client.fetch(query, { slug });

  if (!product) {
    notFound();
```

2. Top Product Listing

- **What I Did:**
  - Created a Top Product List component to display a grid of products dynamically fetched from the API which fetch 8 products.
- **Challenges I Faced:**
  - Managing large datasets.
- **Solution:**
  - Added a view more button and link it to page having more products
- **Additional Features Can Be Added:**
  - Implement control over infinite scrolling.



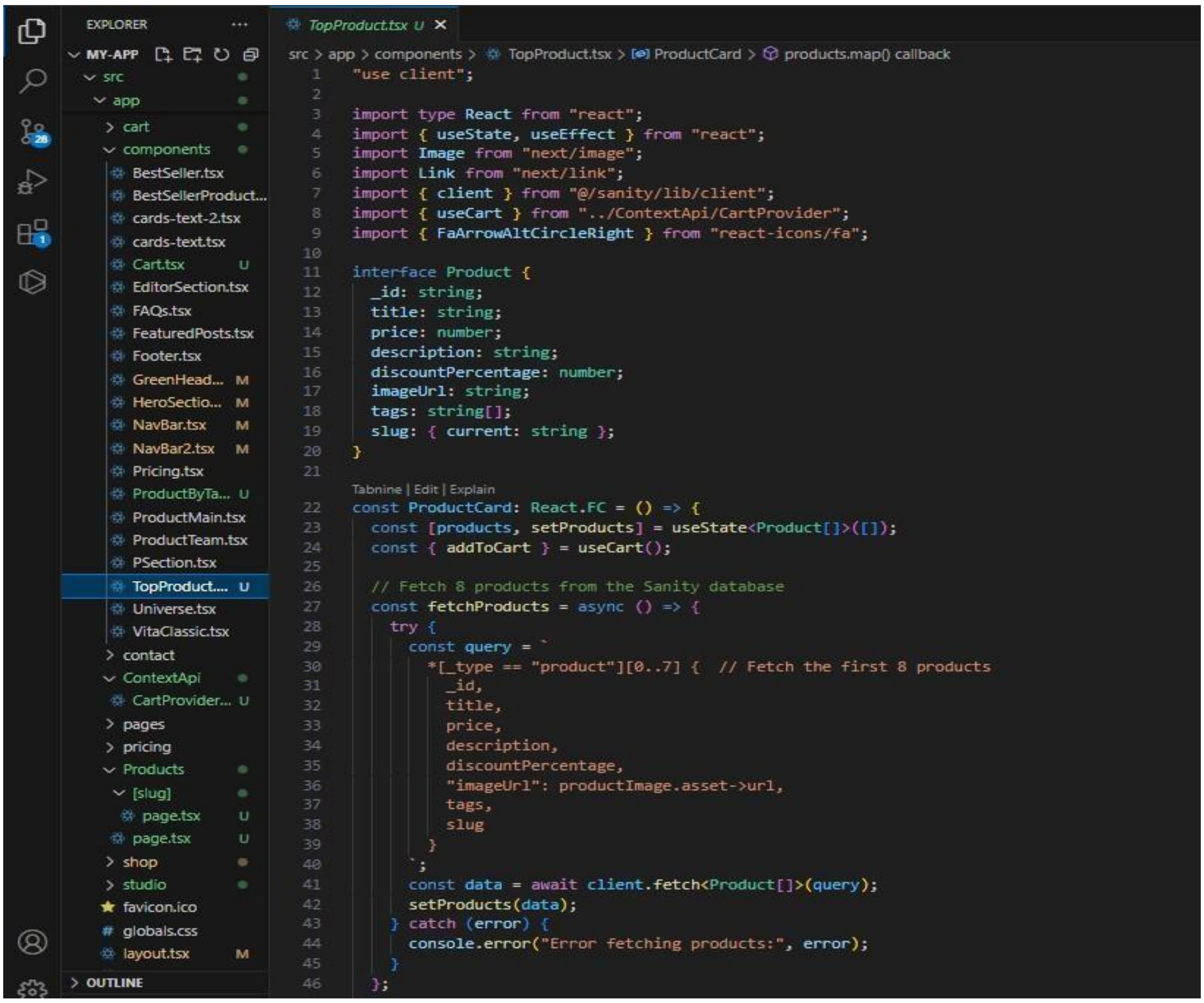
Featured Products

Simona Oatmeal All...
Code - 86657290
$1199.99

Leilani Channel Back...
Code - 65208710
$359.98

Cypress Ivory Faux...
Code - 68371760
$229.99

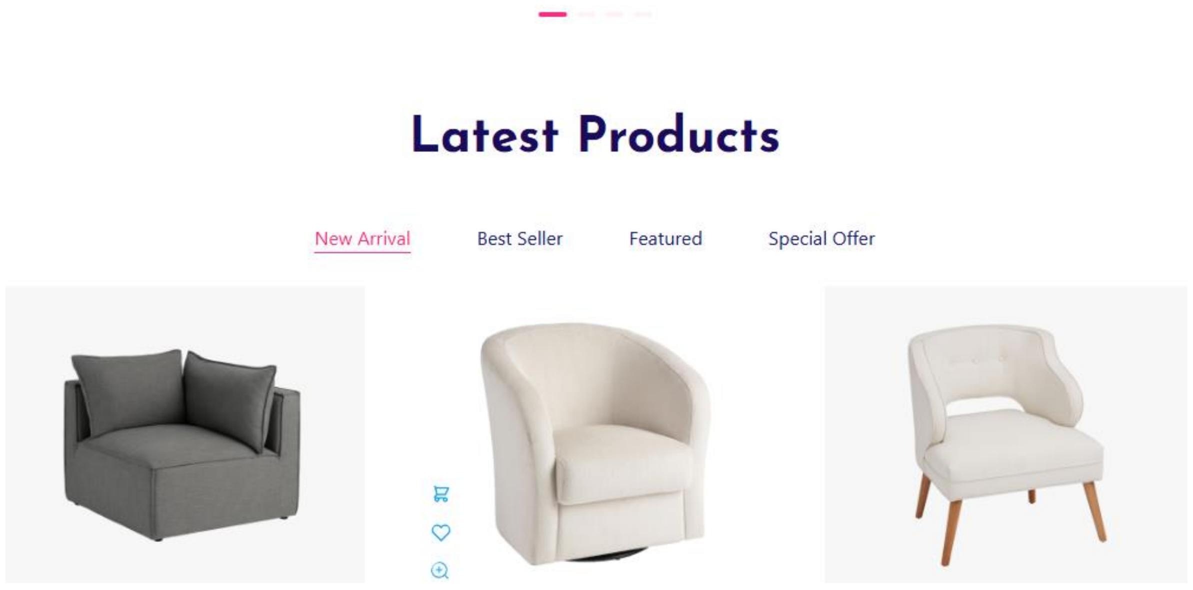
Jarle Molded Resin...
Code - 55790631
$429.99

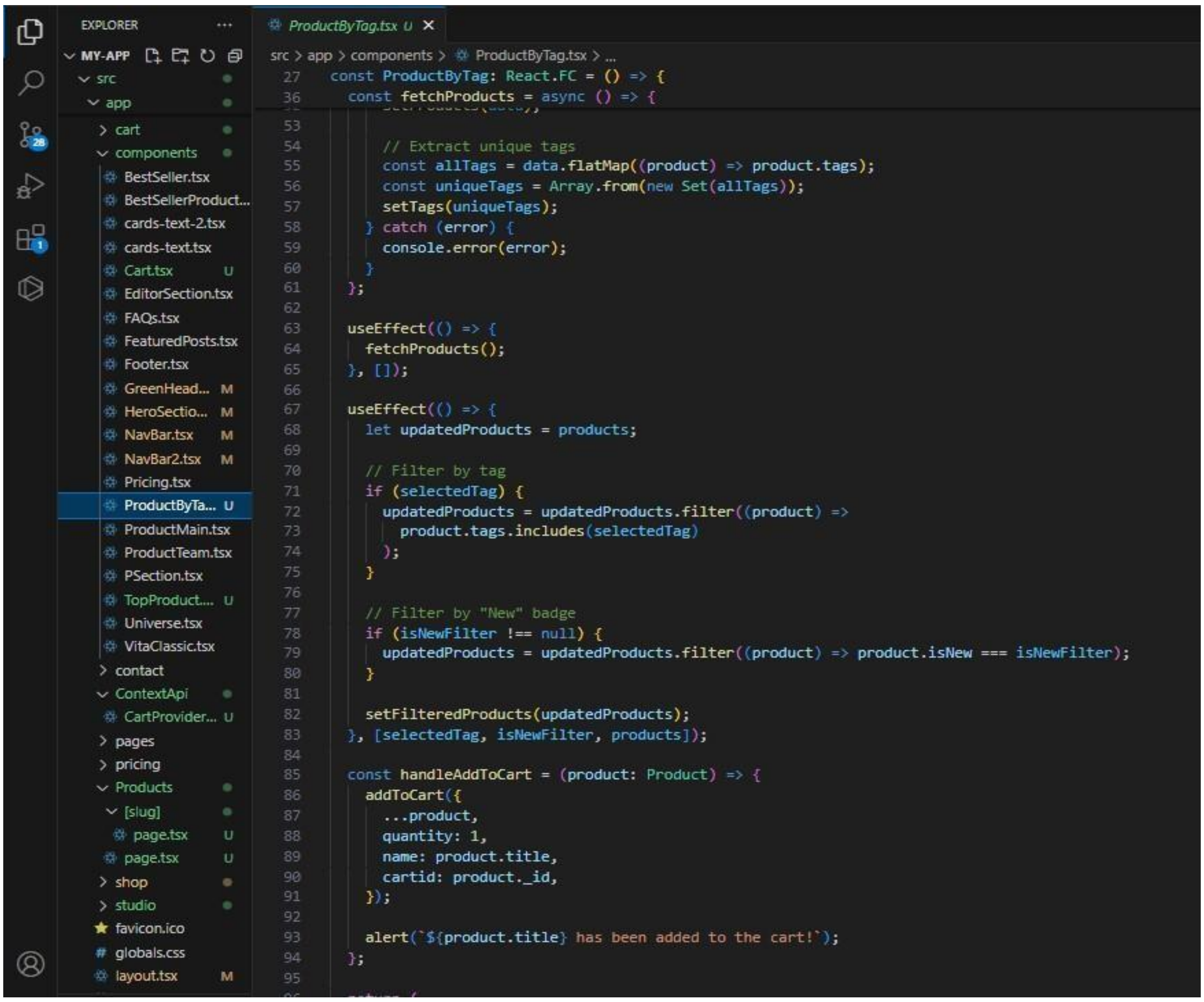


```
"use client";

import type React from "react";
import { useState, useEffect } from "react";
import Image from "next/image";
import Link from "next/link";
import { client } from "@/sanity/lib/client";
import { useCart } from "../ContextApi/CartProvider";
import { FaArrowAltCircleRight } from "react-icons/fa";

interface Product {
  _id: string;
  title: string;
  price: number;
  description: string;
  discountPercentage: number;
  imageUrl: string;
  tags: string[];
  slug: { current: string };
}

const ProductCard: React.FC = () => {
  const [products, setProducts] = useState<Product[]>([]);
  const { addToCart } = useCart();

  // Fetch 8 products from the Sanity database
  const fetchProducts = async () => {
    try {
      const query = `
        *[_type == "product"][0..7] {  // Fetch the first 8 products
          _id,
          title,
          price,
          description,
          discountPercentage,
          "imageUrl": productImage.asset->url,
          tags,
          slug
        }
      `;
      const data = await client.fetch<Product[]>(query);
      setProducts(data);
    } catch (error) {
      console.error("Error fetching products:", error);
    }
  };
```

3. Tags and New Arrival

- **What I Did:**
  - Implemented dynamic filtering based on New Arrival and tags.
  - Used dropdowns for user input.
- **Challenges I Faced:**
  - Efficiently handling API calls.
- **Solution:**
  - Applied debouncing and caching mechanisms to optimize performance.
- **Additional Features Can Be Added:**
  - Implemented a "Clear All Filters" button to reset filters for better usability.

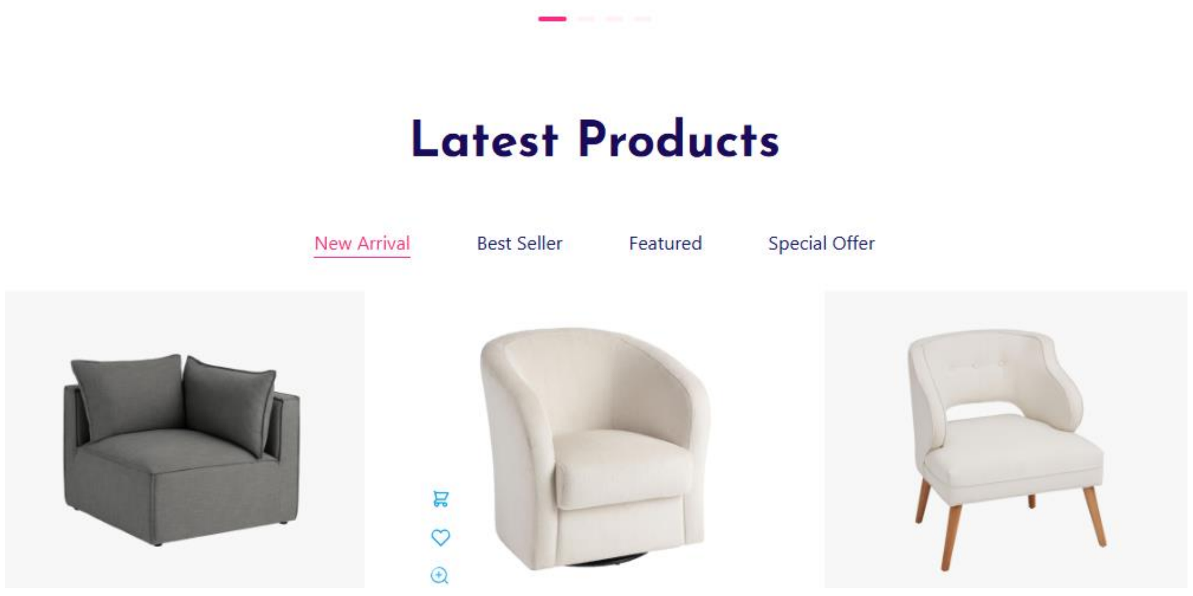




```tsx
const ProductByTag: React.FC = () => {
  const fetchProducts = async () => {

      // Extract unique tags
      const allTags = data.flatMap((product) => product.tags);
      const uniqueTags = Array.from(new Set(allTags));
      setTags(uniqueTags);
    } catch (error) {
      console.error(error);
    }
  };

  useEffect(() => {
    fetchProducts();
  }, []);

  useEffect(() => {
    let updatedProducts = products;

    // Filter by tag
    if (selectedTag) {
      updatedProducts = updatedProducts.filter((product) =>
        product.tags.includes(selectedTag)
      );
    }

    // Filter by "New" badge
    if (isNewFilter !== null) {
      updatedProducts = updatedProducts.filter((product) => product.isNew === isNewFilter);
    }

    setFilteredProducts(updatedProducts);
  }, [selectedTag, isNewFilter, products]);

  const handleAddToCart = (product: Product) => {
    addToCart({
      ...product,
      quantity: 1,
      name: product.title,
      cartid: product._id,
    });

    alert(`${product.title} has been added to the cart!`);
  };
```

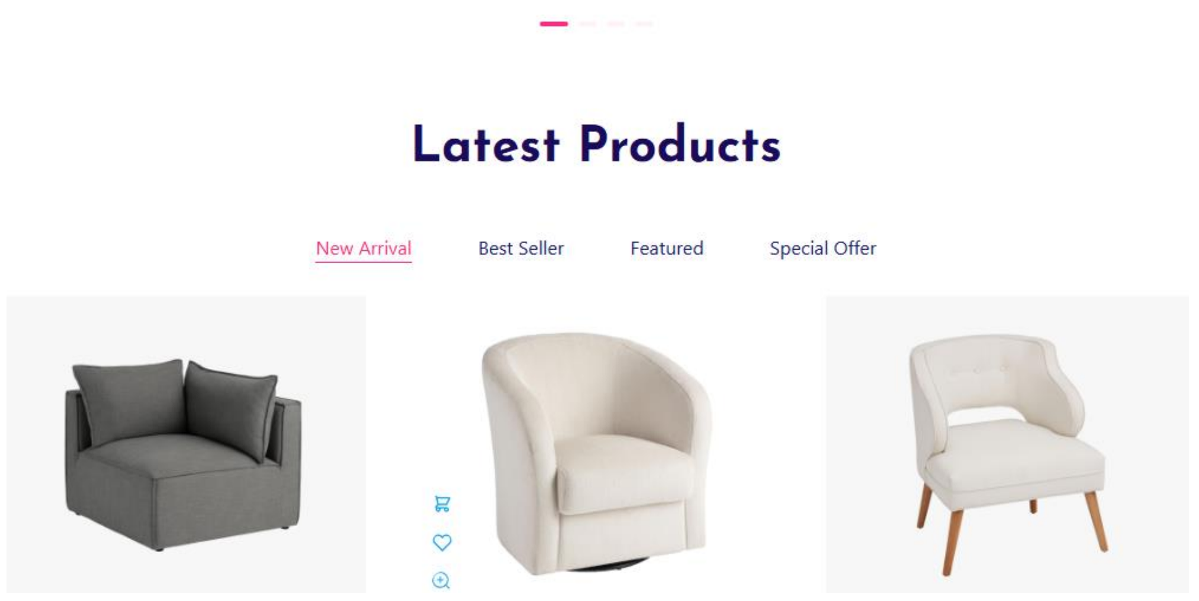
4. Pagination

- **What I Did:**
  - Developed a Pagination Component to divide products into smaller, navigable pages.
  - Styled navigation buttons for user-friendly interaction.
- **Challenges I Faced:**
  - Smooth page transitions.
- **Solution:**
  - I will solve it after hackathon.
- **Additional Features Can Be Added:**
  - Integrated page number highlighting to indicate the current page. ○ Used Next.js’s getStaticProps and getServerSideProps for efficient data fetching. ○ Added "Jump to Page" functionality for quicker navigation.



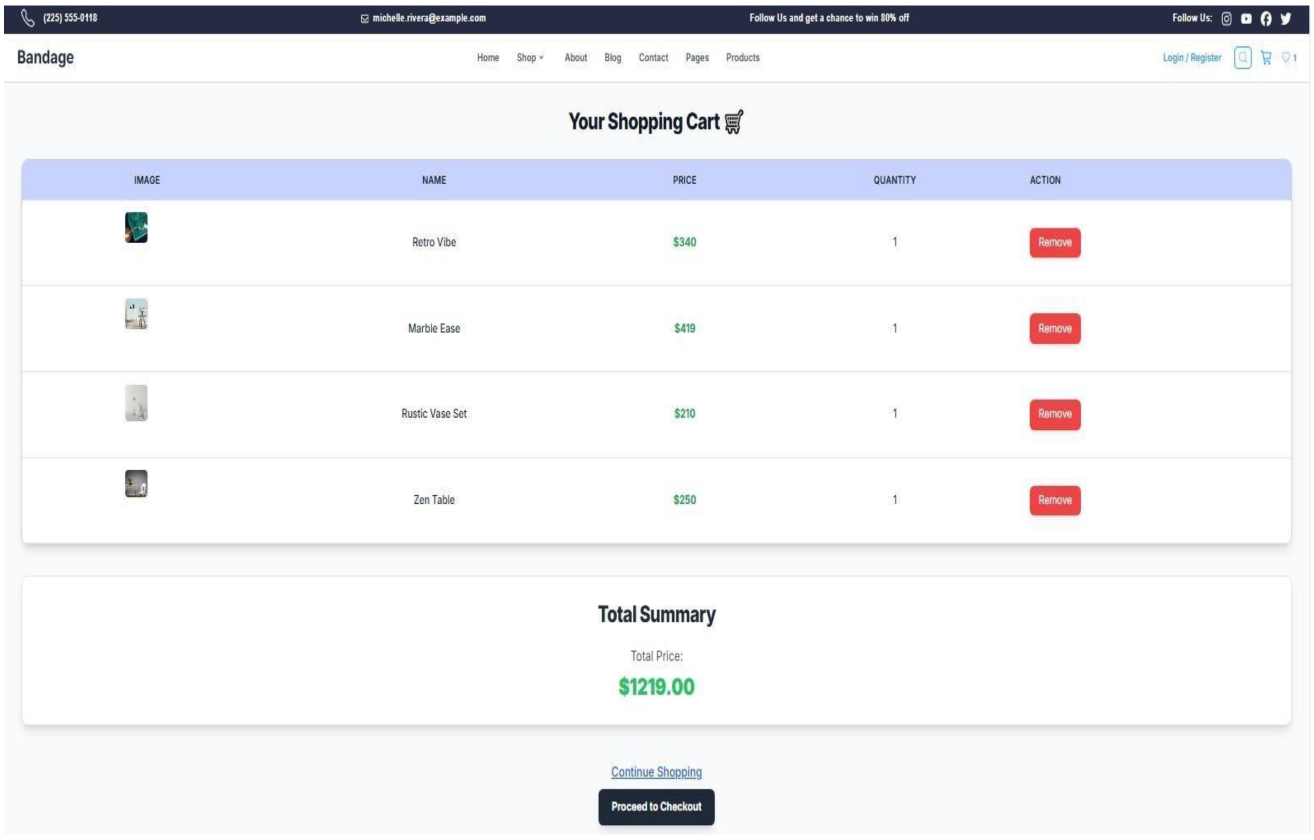
5. Add to Cart

- **What I Did:**
  - Developed the cart functionality using the React Context API to manage the global state.
  - Enabled dynamic quantity updates for cart items.
- **Challenges I Faced:**
  - Syncing cart state across multiple components.
- **Solution:**
  - Used local storage to persist cart data.
- **Additional Features Added:**
  - Integrated a cart summary that dynamically updates the total price and discounts



Latest Products

New Arrival Best Seller Featured Special Offer



(225) 555-0118 michelle.rivera@example.com Follow Us and get a chance to win 80% off Follow Us:

Bandage

Home Shop About Blog Contact Pages Products

Login / Register

Your Shopping Cart 🛒

IMAGE	NAME	PRICE	QUANTITY	ACTION
	Retro Vibe	$340	1	Remove
	Marble Ease	$419	1	Remove
	Rustic Vase Set	$210	1	Remove
	Zen Table	$250	1	Remove

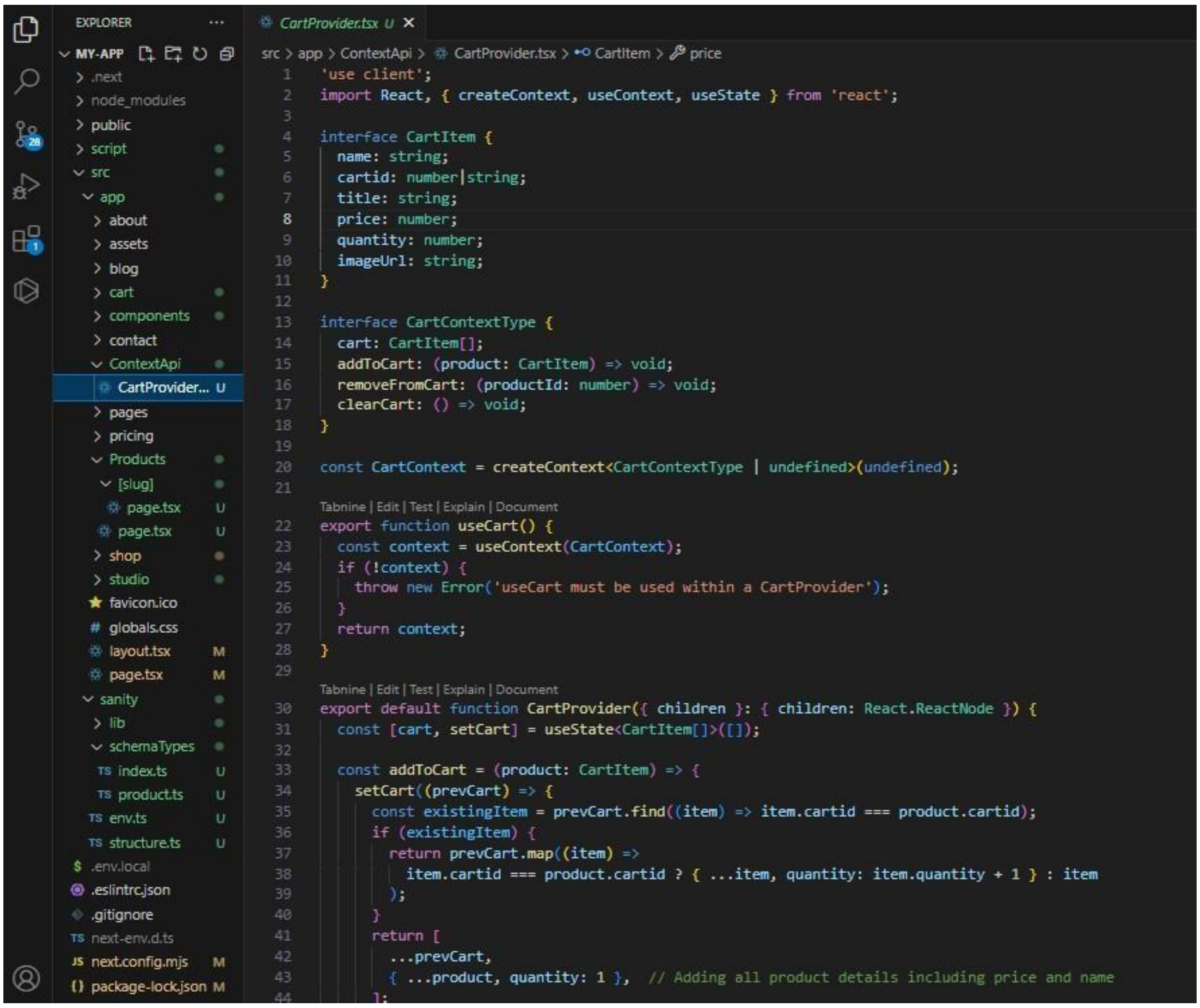
Total Summary

Total Price:

$1219.00

Continue Shopping

Proceed to Checkout



```tsx
'use client';
import React, { createContext, useContext, useState } from 'react';

interface CartItem {
  name: string;
  cartid: number|string;
  title: string;
  price: number;
  quantity: number;
  imageUrl: string;
}

interface CartContextType {
  cart: CartItem[];
  addToCart: (product: CartItem) => void;
  removeFromCart: (productId: number) => void;
  clearCart: () => void;
}

const CartContext = createContext<CartContextType | undefined>(undefined);

export function useCart() {
  const context = useContext(CartContext);
  if (!context) {
    throw new Error('useCart must be used within a CartProvider');
  }
  return context;
}

export default function CartProvider({ children }: { children: React.ReactNode }) {
  const [cart, setCart] = useState<CartItem[]>([]);

  const addToCart = (product: CartItem) => {
    setCart((prevCart) => {
      const existingItem = prevCart.find((item) => item.cartid === product.cartid);
      if (existingItem) {
        return prevCart.map((item) =>
          item.cartid === product.cartid ? { ...item, quantity: item.quantity + 1 } : item
        );
      }
      return [
        ...prevCart,
        { ...product, quantity: 1 },  // Adding all product details including price and name
      ];
```

6. Login and Sign-Up Page

- **What I Did:**
  - Built user authentication pages for login and sign-up with form validation.
  - Integrated Clerk for secure authentication and user management.
- **Challenges I Faced:**
  - Handling errors during login and sign-up.
- **Solution:**
  - Provided user-friendly error messages for better feedback.
- **Additional Features Can Be Added:**
  - Added "Forgot Password" functionality with email recovery support.
  - Enabled login via social accounts like Google.

Summary:

By the end of the development process, I successfully delivered the following:

1. A fully functional product listing page displaying dynamic data from the API.
2. Individual product detail pages implemented with dynamic routing.
3. Advanced filters for New Arrival and tags.
4. Pagination for better user experience with large datasets.
5. Responsive and professional styling for all components.
6. Modular and reusable components for future scalability.
7. Enhanced cart and user authentication functionality for a smoother shopping experience.