

The screenshot shows a website for "HomeDecour". The header includes the logo "HomeDecour" and navigation links for "Home", "Sofa", and "Contact", along with social media icons for Facebook, LinkedIn, and a shopping cart.

A large image of a green tufted sofa set is displayed, with a teal overlay text "Sofa Sets" and a placeholder text "Lorem ipsum, dolor sit amet consectetur adipisicing elit. Laborum aut in sapiente cumque nam dolorum nemo natus deserunt, nisi architecto ipsa nostrum fugit iste consequatur eos molestiae sit dolore facilis reprehenderit." Below the image is a "View More" button.

The section title "Our Sofas Collection" is centered above three smaller sofa images:

- Air Force 1 Mid '07**: A grey sofa with yellow pillows.
- Court Vision Low...**: A small sofa with a round ottoman.
- Air Jordan 1 Elevat...**: A grey sofa with a tufted backrest.

Each sofa has a small circular icon with a lightning bolt symbol and descriptive text below it.

Day 4 - Dynamic Frontend Components “HomeDecour”

Marketplace_builder_hackathon_2025

ABSTRACT

On Day 4, we will focus on designing and developing dynamic frontend components to display marketplace data fetched from Sanity CMS or APIs. This step emphasizes modular, reusable component design and real-world practices for building scalable and responsive web applications.

product listing page with dynamic data.:

product listed component section on front page.

Our Sofas Collection

| | | | |
|--|--|--|--|
|  |  |  |  |
| Air Force 1 Mid '07 Sofa sets are available in various styles, materials, and configurations to match... | Court Vision Low Next... two seater Sofa sets are available in various styles, materials, and configurations to match... | Air Jordan 1 Elevate Low Two Seater are available in various styles, materials, and configurations to match... | Air Max 271 Sofa sets are available in various styles, materials, and configurations to match... |
| \$10795 | \$44995 | \$11895 | \$73295 |
| Details | Details | Details | Details |
|  |  |  |  |

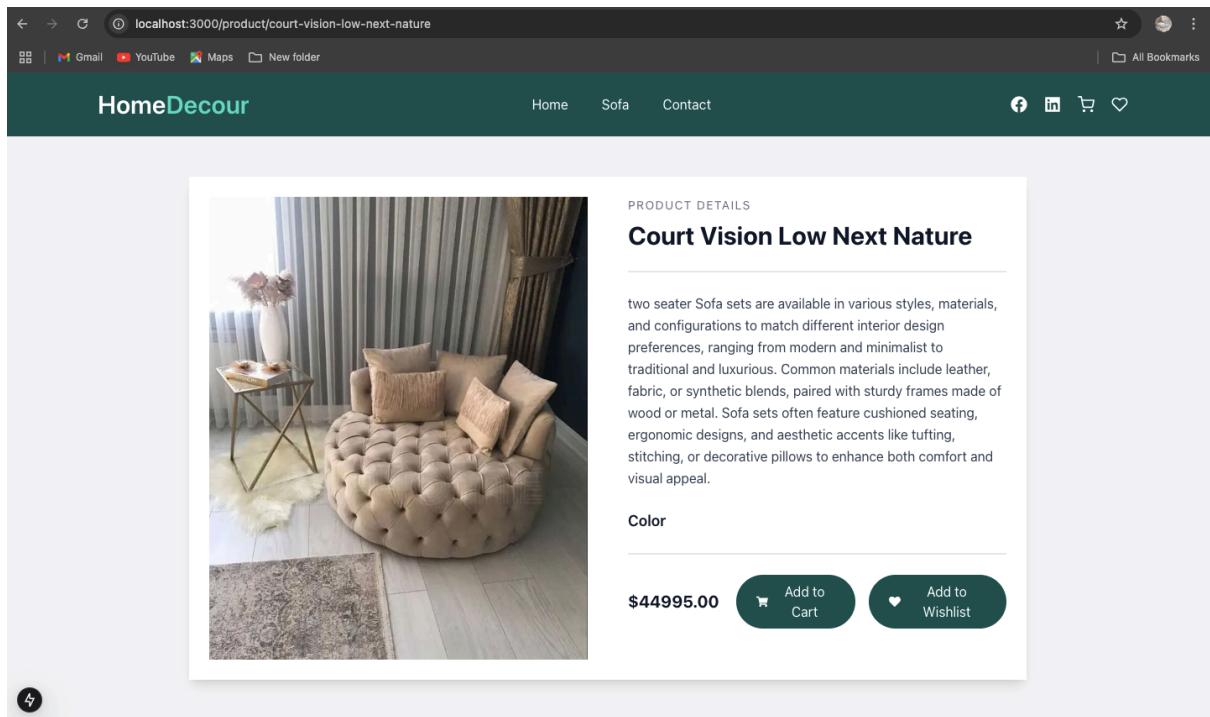
product listed page sofa/page.tsx.

Our SofasSets

| | | | |
|--|--|--|--|
|  |  |  |  |
| Air Force 1 Mid '07 Sofa sets are available in various styles, materials, and configurations to match different... | Court Vision Low Next... two seater Sofa sets are available in various styles, materials, and configurations to match different... | Air Jordan 1 Elevate Low Two Seater are available in various styles, materials, and configurations to match different... | Air Max 271 Sofa sets are available in various styles, materials, and configurations to match different... |
| \$10795 | \$44995 | \$11895 | \$73295 |
| Add To Cart | Add To Cart | Add To Cart | Add To Cart |

Individual product detail pages with accurate routing and data rendering.

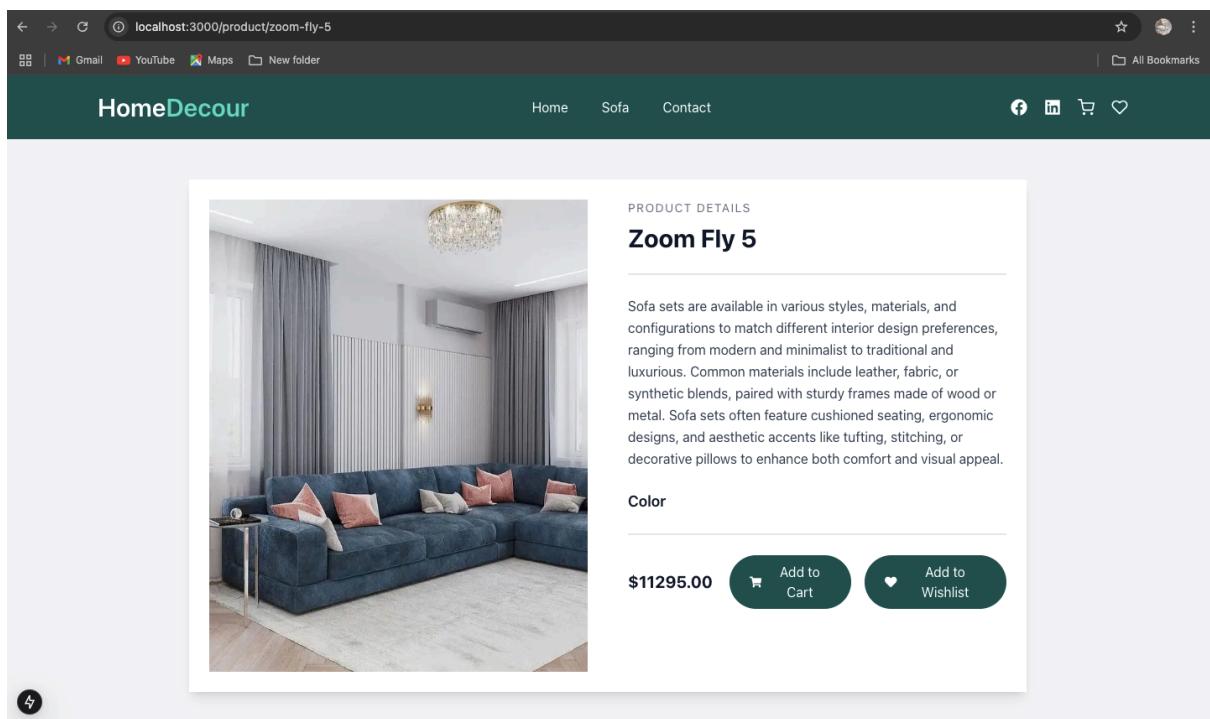
First item.



The screenshot shows a web browser window with the URL `localhost:3000/product/court-vision-low-next-nature`. The page is titled "HomeDecour" and has a navigation bar with links for Home, Sofa, and Contact, along with social media icons for Facebook, LinkedIn, and a shopping cart.

The main content area features a large image of a round, tufted sofa in a living room setting. To the right of the image, the product title "Court Vision Low Next Nature" is displayed in bold. Below the title, a detailed description of two-seater sofa sets is provided, mentioning various styles, materials, and design preferences. A "Color" section is present, followed by a price of \$44995.00 and two buttons: "Add to Cart" and "Add to Wishlist".

Second item.



The screenshot shows a web browser window with the URL `localhost:3000/product/zoom-fly-5`. The page is titled "HomeDecour" and has a navigation bar with links for Home, Sofa, and Contact, along with social media icons for Facebook, LinkedIn, and a shopping cart.

The main content area features a large image of a long, blue velvet sofa with several pillows in a modern living room. To the right of the image, the product title "Zoom Fly 5" is displayed in bold. Below the title, a detailed description of sofa sets is provided, mentioning various styles, materials, and design preferences. A "Color" section is present, followed by a price of \$11295.00 and two buttons: "Add to Cart" and "Add to Wishlist".

Add TO Cart functionality page:

The screenshot shows a web browser window for 'localhost:3000/cart'. The header includes a back arrow, forward arrow, refresh button, and a search bar with the URL. Below the header are links to Gmail, YouTube, Maps, and a 'New folder' icon. The top right corner has a star, a person icon, and three dots. The main navigation bar at the top has 'HomeDecour' on the left and 'Home', 'Sofa', 'Contact' on the right, along with social media icons for Facebook, LinkedIn, a shopping cart, and a heart.

Your Cart

A product item 'Zoom Fly 5' is listed with a price of 'Rs 11295.00'. To its right are quantity controls ('-', '1', '+') and a trash bin icon. To the right of the cart area is the 'Order Summary' section, which displays the following table:

| Order Summary | |
|------------------------|--------------------|
| Subtotal | Rs 11295.00 |
| Delivery Charges | TBD |
| Sales Tax | TBD |
| Estimated Total | Rs 11295.00 |

Below the summary is a large green button labeled 'Proceed to Checkout'. A note below the button states: 'Delivery charges and sales tax will be calculated on the checkout page.'

The footer is dark teal with white text. It features the 'HomeDecour' logo, a brief mission statement 'Our vision is to provide convenience', and links to 'About', 'Community', and 'Social' sections. The 'About' section includes 'How it works' and 'Featured' links. The 'Community' section includes 'Events' and 'Blog' links. The 'Social' section includes 'Discord' and 'Instagram' links.

Alert message for product procced:

The screenshot shows the same cart page as above, but with a modal dialog box overlaid. The dialog has a light gray background with a large question mark icon in the center. Below the icon is the text 'Proceed to checkout?'. Underneath that, a smaller message says 'Please review your cart to checkout!'. At the bottom of the dialog are two buttons: a green 'Yes, proceed!' button and a white 'Cancel' button.

The rest of the page, including the 'Order Summary' table and the 'Proceed to Checkout' button, is visible in the background.

Your Cart

A product item 'Zoom Fly 5' is listed with a price of 'Rs 11295.00'. To its right are quantity controls ('-', '1', '+') and a trash bin icon. To the right of the cart area is the 'Order Summary' section, which displays the following table:

| Order Summary | |
|------------------------|--------------------|
| Subtotal | Rs 11295.00 |
| Delivery Charges | TBD |
| Sales Tax | TBD |
| Estimated Total | Rs 11295.00 |

Below the summary is a large green button labeled 'Proceed to Checkout'. A note below the button states: 'Delivery charges and sales tax will be calculated on the checkout page.'

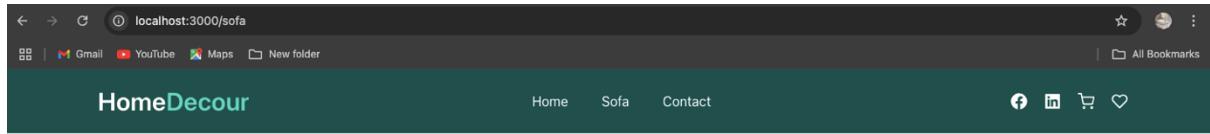
After Procced message checkout page open:

The screenshot shows a checkout page for "HomeDecour". The URL in the address bar is "localhost:3000/checkout". The page has a dark green header with the website name "HomeDecour" and navigation links for "Home", "Sofa", and "Contact". Social media icons for Facebook, LinkedIn, and a shopping cart are also present. The main content area is titled "Shipping Details" and contains fields for First Name (azan), Last Name (malik), Email (@gmail), Phone (12345), Address (dubai), ZIP Code (0000), and City (karachi malir). To the right, there is an "Order Summary" section showing Subtotal (Rs 11295.00), Discount (-Rs 0.00), and a final Total (Rs 11295.00). A large green "Place Order" button is at the bottom of this summary.



"I am working on placing an order in Sanity to store data. However, I am facing an issue where the data is not being stored in Sanity. I have tried fixing all the bugs, but it is still not working."

search bar



Our SofasSets



Waffle One SE

Sofa sets are available in various styles, materials, and configurations to match different...



eggone

Code snippets for key components (ProductCard,

A screenshot of a code editor (VS Code) showing the code for the `ProductCard` component. The code is written in TypeScript and JSX. It uses the `useEffect` hook to fetch products from a client and map them to a grid. The code includes imports for React, useState, useEffect, and various utility classes and interfaces. The code editor interface shows the file structure on the left and the code editor on the right.

```
src > components > sofa.tsx > Recomended > product.map() callback
  1 "use client";
  2 import React, { useEffect, useState } from "react";
  3 import Image from "next/image";
  4 import { Product } from "../../../../types/products";
  5 import { client } from "@sanity/lib/client";
  6 import { eight } from "@sanity/lib/queries";
  7 import { urlFor } from "@sanity/lib/image";
  8 import Link from "next/link";
  9
 10 const Recomended = () => {
 11   const [product, setProduct] = useState<Product>([]);
 12
 13   useEffect(() => {
 14     async function fetchProduct() {
 15       const fetchedProduct: Product[] = await client.fetch(eight);
 16       setProduct(fetchedProduct);
 17     }
 18     fetchProduct();
 19   }, []);
 20
 21   return (
 22     <div className="container mx-auto p-5">
 23       <h2 className="text-3xl font-bold text-center mb-8">
 24         Our <span className="text-teal-400">Sofas</span> Collection
 25       </h2>
 26       <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-2 lg:grid-cols-4 gap-6">
 27         {product.map((product) => (
 28           <div
 29             key={product._id}
 30             className="bg-white rounded-lg shadow-md overflow-hidden p-3"
 31           >
 32             <div>
 33               {product.image &&
 34                 <Image
 35                   alt={product.productName}
 36                   height={400}
 37                   width={400}
 38                   loading="lazy"
 39                   className="w-full h-48 object-cover"
 40                 />
 41               }
 42             </div>
 43           <div className="mt-4">
```

SearchBar

The screenshot shows the VS Code interface with the 'SearchBar' component code open in the editor. The code is a functional component named 'SearchBar' that takes a prop 'onSearch' and a placeholder string. It uses useState to manage the search term and useEffect to debounce the search input. The component returns a form with an input field and a button.

```
src > components > searchBar.tsx > SearchBar
6
7  interface SearchBarProps {
8    onSearch: (searchTerm: string) => void
9    placeholder?: string
10   }
11
12  export default function SearchBar({
13    onSearch,
14    placeholder = 'Search products...'
15  ): SearchBarProps {
16    const [searchTerm, setSearchTerm] = useState('')
17
18    // Debounce search input
19    useEffect(() => {
20      const timeoutId = setTimeout(() => {
21        onSearch(searchTerm.toLowerCase())
22      }, 300)
23
24      return () => clearTimeout(timeoutId)
25    }, [searchTerm, onSearch])
26
27    const handleSearch = (searchTerm: string): void => {
28      console.log(`Searching for: ${searchTerm}`);
29      // Implement your search logic here
30    }
31
32    return (
33      <div className="w-full max-w-2xl mx-auto mb-8">
34        <div className="relative">
35          <input
36            type="text"
37            placeholder={placeholder}
38            value={searchTerm}
39            onChange={(e) => setSearchTerm(e.target.value)}
40            className="w-full px-3 py-3 pl-12 text-gray-700 bg-white border border-gray-300 rounded-lg shadow-sm focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent transition-all duration-300" />
41          <FiSearch className="absolute left-1/2 top-1/2 -translate-y-1/2 text-gray-400 text-xl" />
42        </div>
43      </div>
44    )
45  }
46
47  </div>
48 }
```

Ln 30, Col 5 Spaces: 2 UTF-8 LF ⓘ TypeScript JSX ⓘ Go Live ⓘ Prettier ⓘ

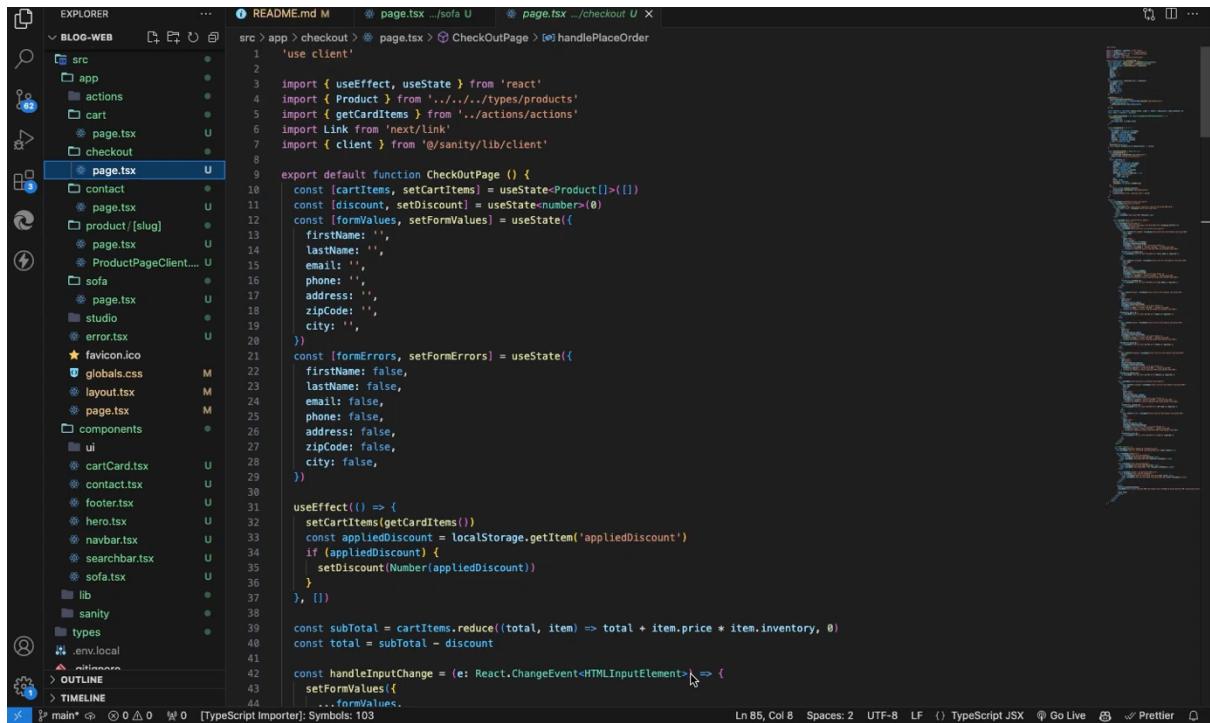
Cart Product Component

The screenshot shows the VS Code interface with the 'CartCard' component code open in the editor. The code defines an interface 'CartCardProps' with methods for removing, incrementing, and decrementing items. The 'CartCard' function takes these props and returns a card component with an image, product name, price, and buttons for managing inventory.

```
src > components > cartCard.tsx > CartCard
8
9  interface CartCardProps {
10    onRemove: () => void
11    onIncrement: () => void
12    onDecrement: () => void
13  }
14
15  export default function CartCard({ item, onRemove, onIncrement, onDecrement }: CartCardProps) {
16    return (
17      <div className="flex flex-col sm:flex-row items-center justify-between bg-white shadow-lg rounded-lg p-4 mb-4">
18        <div className="flex items-center space-x-4">
19          <img
20            src={urlFor(item.image).url()}
21            alt={item.productName}
22            className="w-20 h-20 object-cover rounded-lg" />
23        </div>
24        <div>
25          <h2 className="text-lg font-semibold text-gray-900">{item.productName}</h2>
26          <p className="text-gray-700">Rs {item.price.toFixed(2)}</p>
27        </div>
28      </div>
29
30      <div className="flex items-center space-x-4 mt-4 sm:mt-0">
31        <div className="flex items-center space-x-2">
32          <button
33            onClick={onDecrement}
34            className="p-2 bg-gray-100 rounded-lg hover:bg-gray-200 transition-colors"
35            title="decrement" />
36          <FiMinus className="text-gray-700" />
37          <span className="text-lg font-semibold text-gray-900">{item.inventory}</span>
38          <button
39            onClick={onIncrement}
40            className="p-2 bg-gray-100 rounded-lg hover:bg-gray-200 transition-colors"
41            title="increment" />
42          <FiPlus className="text-gray-700" />
43        </div>
44      </div>
45
46      <button
47        onClick={onRemove}
48        className="p-2 bg-red-100 rounded-lg hover:bg-red-200 transition-colors"
49        title="remove" />
50    )
51  }
52
53  </div>
54
```

Ln 52, Col 24 Spaces: 2 UTF-8 LF ⓘ TypeScript JSX ⓘ Go Live ⓘ Prettier ⓘ

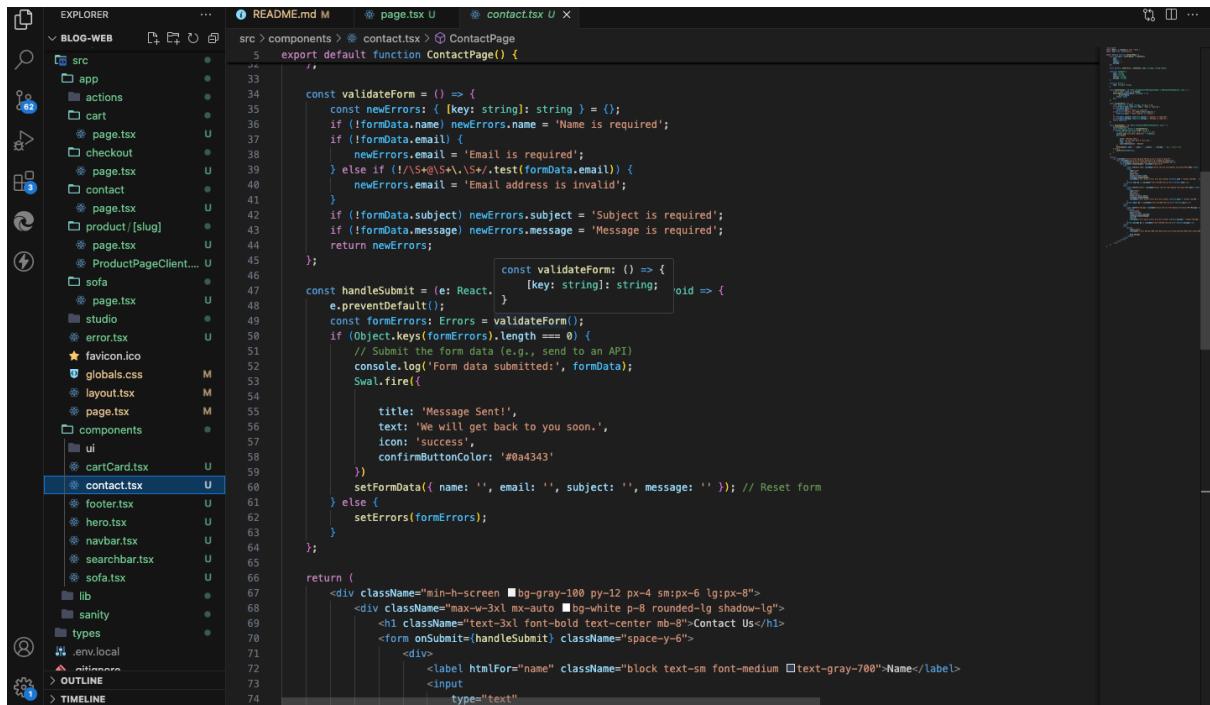
CheckOut Component:



The screenshot shows the VS Code interface with the file `page.tsx` open in the editor. The code is a React component named `CheckOutPage`. It uses state management with `useState` to handle cart items, discount, form values, and errors. It also uses `useEffect` to fetch card items and apply a discount from local storage. The code includes validation logic for form fields like name, email, subject, and message. The `handleInputChange` function updates the form values. The `handleSubmit` function handles the form submission, logs the data, and uses Swal to show a success message.

```
src > app > checkout > page.tsx > CheckOutPage > handlePlaceOrder
1  'use client'
2
3  import { useEffect, useState } from 'react'
4  import { Product } from '../types/products'
5  import { getCardItems } from '../actions/actions'
6  import Link from 'next/link'
7  import { client } from '@sanity/lib/client'
8
9  export default function CheckOutPage () {
10    const [cartItems, setCartItems] = useState<Product[]>([])
11    const [discount, setDiscount] = useState<number>(0)
12    const [formValues, setFormValues] = useState({
13      firstName: '',
14      lastName: '',
15      email: '',
16      phone: '',
17      address: '',
18      zipCode: '',
19      city: ''
20    })
21    const [formErrors, setFormErrors] = useState({
22      firstName: false,
23      lastName: false,
24      email: false,
25      phone: false,
26      address: false,
27      zipCode: false,
28      city: false
29    })
30
31    useEffect(() => {
32      setCartItems(getCardItems())
33      const appliedDiscount = localStorage.getItem('appliedDiscount')
34      if (appliedDiscount) {
35        setDiscount(Number(appliedDiscount))
36      }
37    }, [])
38
39    const subTotal = cartItems.reduce((total, item) => total + item.price * item.inventory, 0)
40    const total = subTotal - discount
41
42    const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {
43      setFormValues({
44        ...formValues,
45        [e.target.name]: e.target.value
46      })
47
48      const validateForm: () => {
49        const newErrors: { [key: string]: string } = {}
50        if (!formData.name) newErrors.name = 'Name is required'
51        if (!formData.email) {
52          newErrors.email = 'Email is required'
53        } else if (!/^\S+@\S+\.\S+$/.test(formData.email)) {
54          newErrors.email = 'Email address is invalid'
55        }
56        if (!formData.subject) newErrors.subject = 'Subject is required'
57        if (!formData.message) newErrors.message = 'Message is required'
58        return newErrors
59      }
60
61      const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
62        e.preventDefault()
63        const formErrors: Errors = validateForm()
64        if (Object.keys(formErrors).length === 0) {
65          // Submit the form data (e.g., send to an API)
66          console.log('Form data submitted:', formData)
67          Swal.fire({
68            title: 'Message Sent!',
69            text: 'We will get back to you soon.',
70            icon: 'success',
71            confirmButtonColor: '#0a4343'
72          })
73          setFormData({ name: '', email: '', subject: '', message: '' }) // Reset form
74        } else {
75          setErrors(formErrors)
76        }
77      }
78
79      return (
80        <div className="min-h-screen bg-gray-100 py-12 px-4 sm:px-6 lg:px-8">
81          <div className="max-w-3xl mx-auto bg-white p-8 rounded-lg shadow-lg">
82            <h1 className="text-3xl font-bold text-center mb-8">Contact Us</h1>
83            <form onSubmit={handleSubmit} className="space-y-6">
84              <div>
85                <label htmlFor="name" className="block text-sm font-medium text-gray-700">Name</label>
86                <input type="text" name="name" />
87              </div>
88            </form>
89          </div>
90        </div>
91      )
92    }
93  }
```

Contact Component:



The screenshot shows the VS Code interface with the file `contact.tsx` open in the editor. The code defines a component named `ContactPage`. It includes validation logic for the `name`, `email`, `subject`, and `message` fields. The `handleSubmit` function handles the form submission by logging the data and using `Swal` to show a success message. The component returns a form with input fields for these fields.

```
src > components > contact.tsx > ContactPage
5  export default function ContactPage() {
6
7    const validateForm = () => {
8      const newErrors: { [key: string]: string } = {}
9      if (!formData.name) newErrors.name = 'Name is required'
10     if (!formData.email) {
11       newErrors.email = 'Email is required'
12     } else if (/^\S+@\S+\.\S+$/.test(formData.email)) {
13       newErrors.email = 'Email address is invalid'
14     }
15     if (!formData.subject) newErrors.subject = 'Subject is required'
16     if (!formData.message) newErrors.message = 'Message is required'
17     return newErrors
18   };
19
20   const handleSubmit = (e: React.FormEvent<HTMLFormElement>) => {
21     e.preventDefault();
22     const formErrors: Errors = validateForm();
23     if (Object.keys(formErrors).length === 0) {
24       // Submit the form data (e.g., send to an API)
25       console.log('Form data submitted:', formData);
26       Swal.fire({
27         title: 'Message Sent!',
28         text: 'We will get back to you soon.',
29         icon: 'success',
30         confirmButtonColor: '#0a4343'
31       })
32       setFormData({ name: '', email: '', subject: '', message: '' }) // Reset form
33     } else {
34       setErrors(formErrors);
35     }
36   };
37
38   return (
39     <div className="min-h-screen bg-gray-100 py-12 px-4 sm:px-6 lg:px-8">
40       <div className="max-w-3xl mx-auto bg-white p-8 rounded-lg shadow-lg">
41         <h1 className="text-3xl font-bold text-center mb-8">Contact Us</h1>
42         <form onSubmit={handleSubmit} className="space-y-6">
43           <div>
44             <label htmlFor="name" className="block text-sm font-medium text-gray-700">Name</label>
45             <input type="text" name="name" />
46           </div>
47           <div>
48             <label htmlFor="email" className="block text-sm font-medium text-gray-700">Email</label>
49             <input type="text" name="email" />
50           </div>
51           <div>
52             <label htmlFor="subject" className="block text-sm font-medium text-gray-700">Subject</label>
53             <input type="text" name="subject" />
54           </div>
55           <div>
56             <label htmlFor="message" className="block text-sm font-medium text-gray-700">Message</label>
57             <input type="text" name="message" />
58           </div>
59         </form>
60       </div>
61     </div>
62   )
63 }
```

Dynamic Routing:

```
src > app > product > [slug] > page.tsx > getProduct
1 // app/product/[slug]/page.tsx (Server Component)
2 import { Product } from "../../../../../types/products";
3 import { client } from "@sanity/lib/client";
4 import { groq } from "next-sanity";
5 import ProductPageClient from "./ProductPageClient";
6
7 async function getProduct(slug: string): Promise<Product> {
8   return client.fetch(`*[_type == "product" && slug.current == $slug][0]{
9     _id,
10    name,
11    _type,
12    image,
13    originalPrice,
14    seatingCapacity,
15    fuelCapacity,
16    pricePerDay,
17    description,
18    productName,
19    price
20  }`,
21    { slug }
22  );
23}
24
25
26 export default async function ProductPage({ params }: { params: { slug: string } }) {
27   const product = await getProduct(params.slug);
28   return <ProductPageClient product={product} />;
29 }
```

SUMMARY:

1. Introduction:

This report summarizes the steps taken to build and integrate key components (e.g., `ProductCard`, `ProductList`, `SearchBar`) for an e-commerce application. It also highlights the challenges faced, solutions implemented, and best practices followed during development.

2. Steps Taken to Build and Integrate Components

Component Breakdown:

The application was built using reusable React components. Below are the steps taken to create and integrate each component:

ProductCard Component:

Purpose: Displays individual product details (e.g., image, name, price, description).

Steps:

- Created a functional React component.
- Used Tailwind CSS for styling.
- Added an "Add to Cart" button for interactivity.

Integration: Used inside the `ProductList` component to render each product.

ProductList Component:

Purpose: Displays a list of products using the `ProductCard` component.

Steps:

- Mapped over an array of products and rendered a `ProductCard` for each item.
- Used a responsive grid layout (Tailwind CSS) to ensure the list looks good on all screen sizes.

Integration: Integrated with the `SearchBar` component to filter products dynamically.

SearchBar Component:

Purpose: Allows users to search for products by name.

Steps:

- Added a controlled input field to capture the search term.
- Implemented a search handler to filter products based on the search term.
- Used Tailwind CSS for responsive design.

Integration:

Connected to the `ProductList` component to update the displayed products dynamically.

Parent Component (App):

Purpose: Acts as the main container for all components.

Steps:

- Managed the state for products and filtered products.
- Passed necessary props to child components (`SearchBar`, `ProductList`).

Integration: Combined all components into a cohesive user interface.

3. Challenges Faced and Solutions Implemented

Challenge: Dynamic Product Filtering

Issue: The `SearchBar` component needed to filter products in real-time without reloading the page.

Solution:

- Used React's `useState` to manage the search term and filtered products.
- Implemented a search handler that filters the product list based on the search term.

Challenge: Responsive Design

Issue: Ensuring the components looked good on all screen sizes (desktop, tablet, mobile).

Solution:

- Used Tailwind CSS utility classes to create responsive layouts (e.g., `grid-cols-1 sm:grid-cols-2 md:grid-cols-3`).
- Tested the design on multiple devices and screen sizes.

Challenge: State Management:

Issue:

Managing the state of products and filtered products without unnecessary re-renders.

Solution:

- Used React's `useState` and `useEffect` hooks effectively.
- Ensured that state updates were optimized to avoid performance issues.

Best Practices Followed During Development:

Component Reusability

- Created modular and reusable components (e.g., `ProductCard`, `SearchBar`).
- Ensured components were decoupled and could be used in different parts of the application.

Responsive Design

- Used Tailwind CSS to create a mobile-first, responsive design.
- Tested the application on multiple devices and screen sizes.

Code Readability and Maintainability

- Followed consistent naming conventions for variables, functions, and components.

- Added comments to explain complex logic.
- Used meaningful prop names for better understanding.

Performance Optimization

- Avoided unnecessary re-renders by using React's state and effect hooks efficiently.
- Implemented debouncing for the search functionality to reduce API calls or filtering operations.

Testing and Debugging

- Tested each component in isolation using tools like Cypress or manual testing.
- Used `console.log` and React DevTools to debug state and props.

Conclusion:

The development process involved building reusable components (`ProductCard`, `ProductList`, `SearchBar`) and integrating them into a cohesive user interface. Challenges such as dynamic filtering, responsive design, and state management were addressed using React hooks and Tailwind CSS. Best practices like component reusability, responsive design, and performance optimization were followed to ensure a high-quality, maintainable application.
