DAY 4 NAME: MUHAMMED ANAS

ROLL NO: 00051245

TIMING: SAT 9-12AM

Day 4: Implementation of Dynamic Product Listing Component for Mens Apparel

Objective

The primary goal of this phase is to design and develop a dynamic frontend component that displays marketplace data fetched from Sanity CMS or an external API. The implementation emphasizes modularity, reusability, and responsiveness, aligning with industry-standard development practices for scalable and maintainable web applications.

Task Overview

Objective:

Develop a Product Listing Component that dynamically retrieves and displays product information in a structured layout.

Requirements:

1. **Dynamic Data Fetching:** Retrieve product data from Sanity CMS or an external API.
2. **Product Grid Layout:** Present products in a visually structured format with key details:
   - Product Name
   - Price
   - Image
3. **Responsive Design:** Ensure compatibility across multiple screen sizes.
4. **Modularity & Reusability:** Divide the component into smaller reusable parts.

Tools & Technologies

- **Framework:** Next.js
- **CMS:** Sanity CMS
- **Styling:** Tailwind CSS
- **State Management:** React Hooks

Implementation Plan

1. Set Up Data Fetching

- Integrate Sanity CMS or external API to dynamically fetch product data.

- Utilize React hooks:
    - `useEffect` to fetch data upon component mount.
    - `useState` to manage and store product data.

2. Design Reusable Components

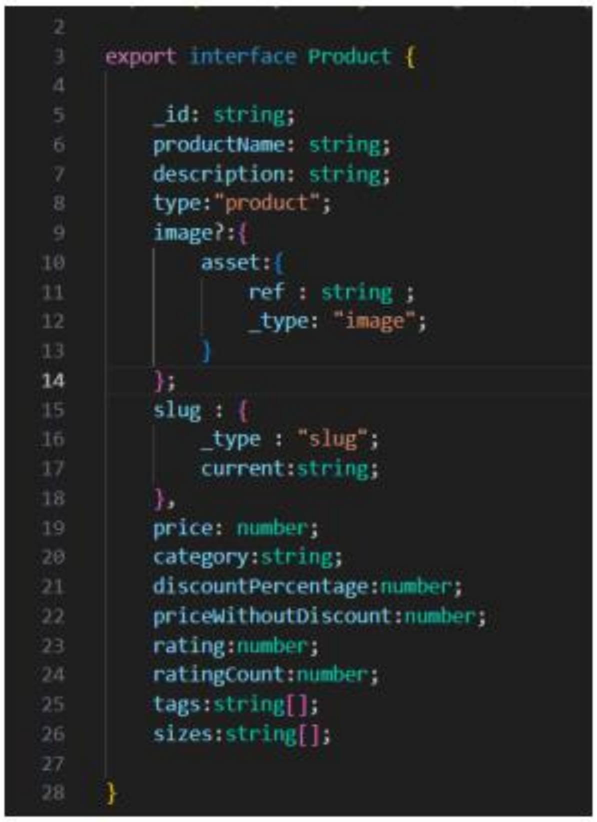
- **Product Card Component:** Displays individual product information.
- **Grid Layout Component:** Organizes product cards in a flexible, responsive grid layout.
- **Loader/Error Handling Component (optional):** Provides feedback during data fetch operations.

3. Apply Responsive Design

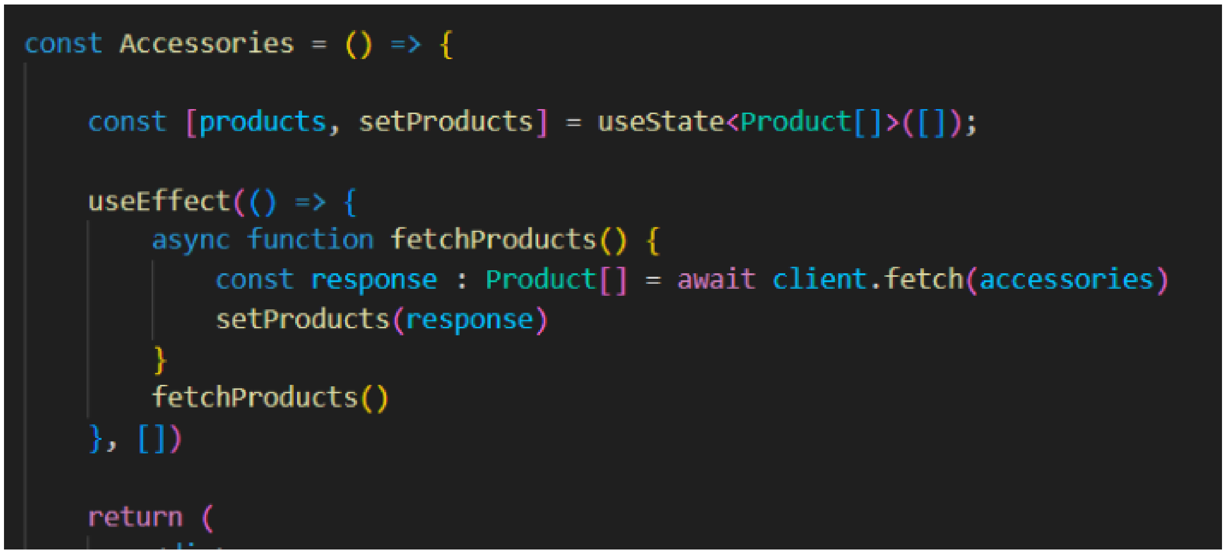
- Use Tailwind CSS to apply utility-based responsive styles.
- Implement CSS Grid or Flexbox to create a dynamic grid layout.

4. Enhance User Experience

- Utilize conditional formatting to highlight key details such as stock availability.
- Apply hover effects for interactivity and improved UI feedback.
- Ensure smooth transitions and animations for a polished experience.



```
export interface Product {

    _id: string;
    productName: string;
    description: string;
    type:"product";
    image?:{
        asset:{
            ref : string ;
            _type: "image";
        }
    };
    slug : {
        _type : "slug";
        current:string;
    },
    price: number;
    category:string;
    discountPercentage:number;
    priceWithoutDiscount:number;
    rating:number;
    ratingCount:number;
    tags:string[];
    sizes:string[];

}
```



```
const Accessories = () => {

    const [products, setProducts] = useState<Product[]>([]);

    useEffect(() => {
        async function fetchProducts() {
            const response : Product[] = await client.fetch(accessories)
            setProducts(response)
        }
        fetchProducts()
    }, [])

    return (
```

2. Product Detail Component

Objective:

Develop individual product detail pages using dynamic routing in Next.js. These pages will display detailed information about each product, including:

- Name
- Product Description
- Price
- Category

Implementation Plan:

1. Dynamic Routing:

- Create dynamic routes using the `[slug].tsx` file in the `pages/products` directory.
- Fetch product data based on the product ID from a CMS like Sanity or an API.

2. Data Fields:

Each product detail page should include the following fields:

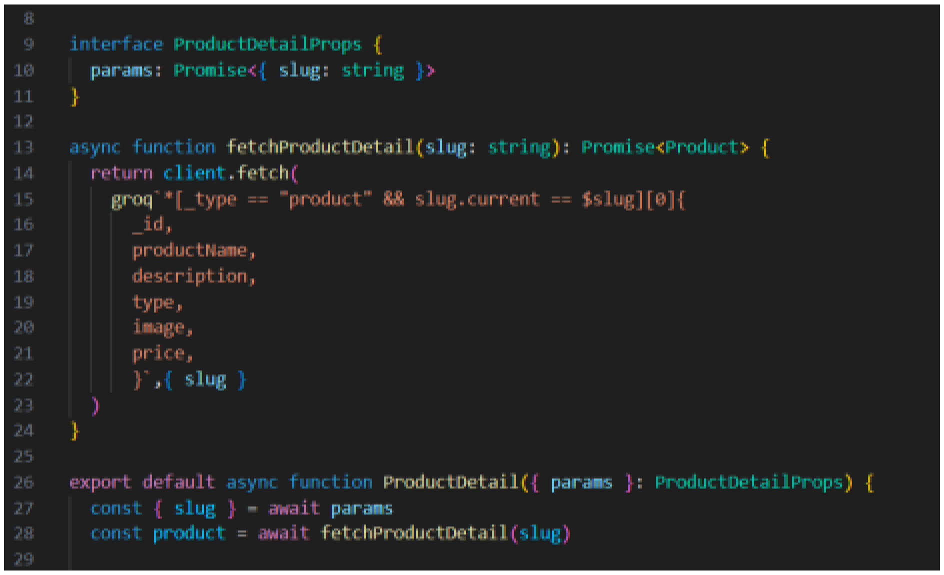
- **Product Description:** A detailed explanation of the product, fetched from the backend.
- **Price:** Displayed prominently for clear visibility.

3. Integration with Product Listing:

- Link each product card in the Product Listing Component to its corresponding detail page using the `Link` component in Next.js.

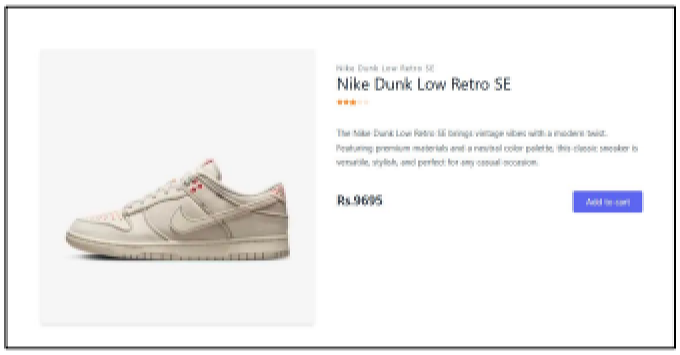
4. Styling and Layout:

- Use Tailwind CSS for a clean and responsive design.
- Ensure the layout highlights the product description and price for user clarity.



```
interface ProductDetailProps {
  params: Promise<{ slug: string }>
}

async function fetchProductDetail(slug: string): Promise<Product> {
  return client.fetch(
    groq`*[_type == "product" && slug.current == $slug][0]{
      _id,
      productName,
      description,
      type,
      image,
      price,
      }`,{ slug }
  )
}

export default async function ProductDetail({ params }: ProductDetailProps) {
  const { slug } = await params
  const product = await fetchProductDetail(slug)
```

UI Display of Product Detail Page



Nike Dunk Low Retro SE

Nike Dunk Low Retro SE

The Nike Dunk Low Retro SE brings vintage vibes with a modern twist. Featuring premium materials and a neutral color palette, this classic sneaker is versatile, stylish, and perfect for any casual occasion.

Rs.9695

---

3. Search Bar with Price Filter

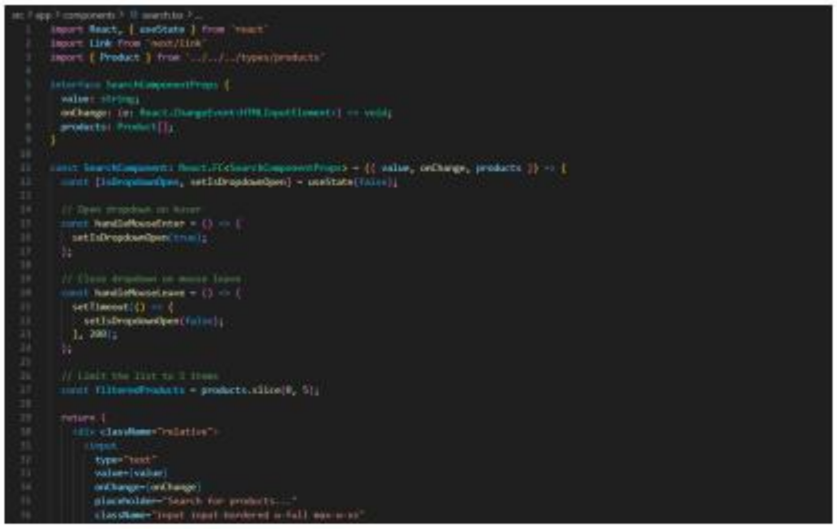
Objective:

To implement a search bar and price filters to enhance the product browsing experience.

Implementation Plan:

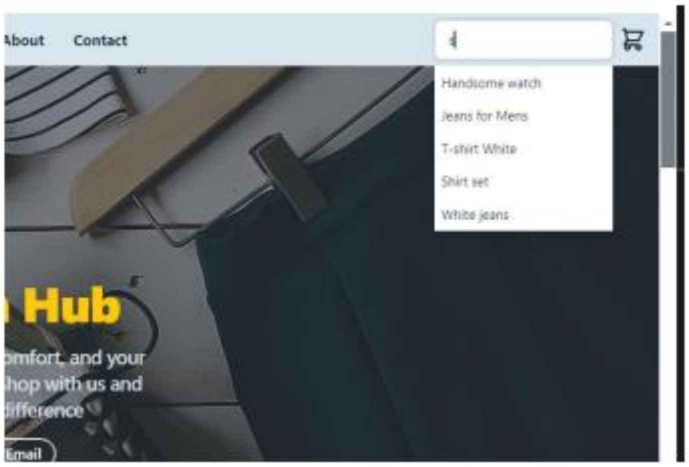
1. Search Bar Functionality:

- Filter products based on their name or associated tags.
- Update the product list in real-time as the user types.



```
src > app > components > searchbar.tsx > ...
import React, { useState } from 'react'
import Link from 'next/link'
import { Product } from '../../../types/products'

interface SearchComponentProps {
  value: string;
  onChange: (e: React.ChangeEvent<HTMLInputElement>) => void;
  products: Product[];
}

const SearchComponent: React.FC<SearchComponentProps> = ({ value, onChange, products }) => {
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);

  // Open dropdown on hover
  const handleMouseEnter = () => {
    setIsDropdownOpen(true);
  };

  // Close dropdown on mouse leave
  const handleMouseLeave = () => {
    setTimeout(() => {
      setIsDropdownOpen(false);
    }, 200);
  };

  // Limit the list to 5 items
  const filteredProducts = products.slice(0, 5);

  return (
    <div className="relative">
      <input
        type="text"
        value={value}
        onChange={onChange}
        placeholder="Search for products..."
        className="input input-bordered w-full max-w-xs"
```

UI Display of Search Bar functionality



4. Cart Component

Objective:

To create a Cart Component that displays the items added to the cart, their quantity, and the total price of the cart dynamically.

Implementation Plan:

1. State Management:

- Use React state or a state management library like Redux for storing cart data.

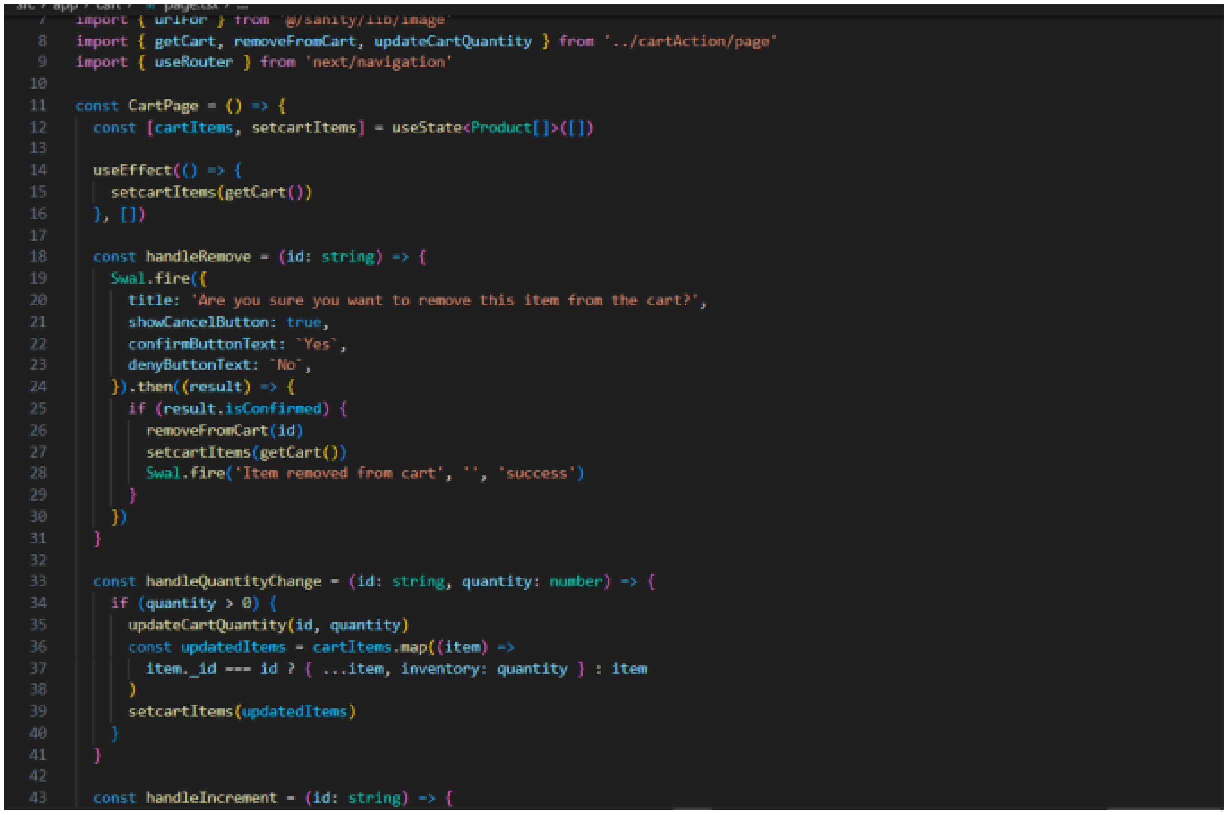
2. Cart Data:

Each product in the cart should include the following details:

- **Product Name**
- **Price**
- **Quantity**
- Calculate and display the total price dynamically based on the items in the cart.

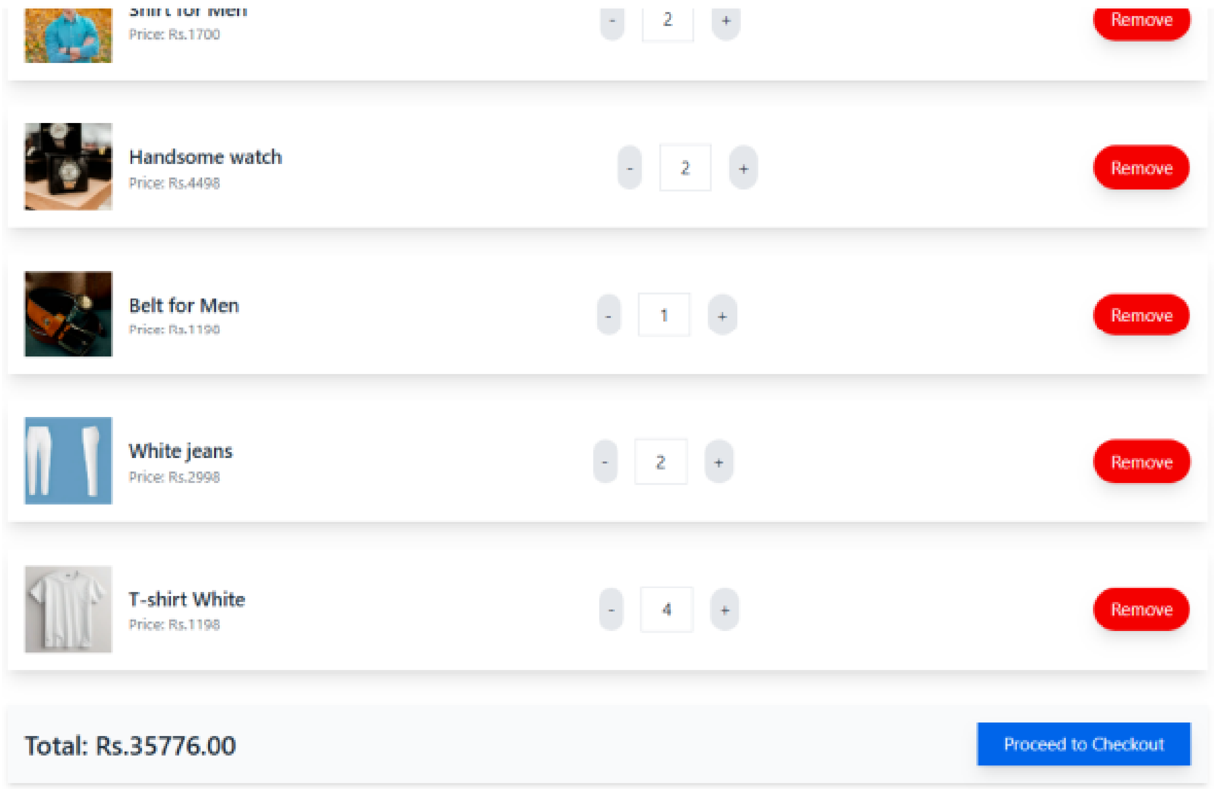
3. Cart Interactions:

- Allow users to increase or decrease the quantity of items.
- Automatically update the total price when the quantity changes.



```
import { urlFor } from '@/sanity/lib/image'
import { getCart, removeFromCart, updateCartQuantity } from '../cartAction/page'
import { useRouter } from 'next/navigation'

const CartPage = () => {
  const [cartItems, setcartItems] = useState<Product[]>([])

  useEffect(() => {
    setcartItems(getCart())
  }, [])

  const handleRemove = (id: string) => {
    Swal.fire({
      title: 'Are you sure you want to remove this item from the cart?',
      showCancelButton: true,
      confirmButtonText: `Yes`,
      denyButtonText: `No`,
    }).then((result) => {
      if (result.isConfirmed) {
        removeFromCart(id)
        setcartItems(getCart())
        Swal.fire('Item removed from cart', '', 'success')
      }
    })
  }

  const handleQuantityChange = (id: string, quantity: number) => {
    if (quantity > 0) {
      updateCartQuantity(id, quantity)
      const updatedItems = cartItems.map((item) =>
        item._id === id ? { ...item, inventory: quantity } : item
      )
      setcartItems(updatedItems)
    }
  }

  const handleIncrement = (id: string) => {
```

UI Display of Cart Page



Shirt for Men
Price: Rs.1700
- 2 +
Remove

Handsome watch
Price: Rs.4498
- 2 +
Remove

Belt for Men
Price: Rs.1190
- 1 +
Remove

White jeans
Price: Rs.2998
- 2 +
Remove

T-shirt White
Price: Rs.1198
- 4 +
Remove

Total: Rs.35776.00
Proceed to Checkout

Features Implemented

1. Dynamic Item Display:

- Each item in the cart is displayed with its name, price, and quantity.
- Subtotal for each item is dynamically calculated.

2. Quantity Update:

- Buttons to increase (+) or decrease (-) the quantity of an item.
- Quantity cannot go below 1.

3. Total Price Calculation:

- The total price updates dynamically as items are added or quantities are changed.

4. Remove Item:

- Users can remove individual items from the cart.

Conclusion

On Day 4 of building dynamic frontend components for a marketplace, the focus was on creating modular, reusable, and responsive components. The following key components were successfully implemented:

1. Product Listing Component:

- Dynamically displayed products in a grid layout with details such as product name, price, image, and stock status.

2. Product Detail Component:

- Built individual product pages using dynamic routing in Next.js, including fields like product description, price, and image.

3. Search Bar and Filters:

- Implemented functionality to filter products by name or tags and added price filters (high to low and low to high).

4. Cart Component:

- Displayed items added to the cart, quantity management, and total price calculation with dynamic updates.