Day 4 : Implementing a Dynamic Product Listing Component

Mens Apparel

Objective:

The primary objective of Day 4 is to design and develop dynamic frontend components that can display marketplace data fetched from Sanity CMS or external APIs. This process focuses on modularity, reusability, and applying real-world development practices to build scalable and responsive web applications.

Task Overview

Objective:

Build a Product Listing Component for a marketplace.

Requirements:

1. Fetch product data dynamically using Sanity CMS or an external API.

2. Display the data in a grid layout of cards with the following details:

   - Product Name
   - Price
   - Image

3. Ensure responsiveness across devices.

4. Implement modularity by breaking 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 API endpoints to fetch the product data dynamically.
   - Use React hooks (useEffect) for data fetching and (useState) to store and manage the data.
2. **Design Reusable Components:**
   - Break down the Product Listing Component into smaller parts:

- **Product Card Component:** Displays individual product details.
- **Grid Layout Component:** Arranges the product cards in a responsive grid.

3. **Apply Responsive Design:**
   Use Tailwind CSS or CSS Grid/Flexbox to ensure the grid layout adapts to all screen sizes.
4. **Enhance User Experience:**
   - Highlight important details like stock status with conditional formatting.
   - Add hover effects for better interactivity.

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

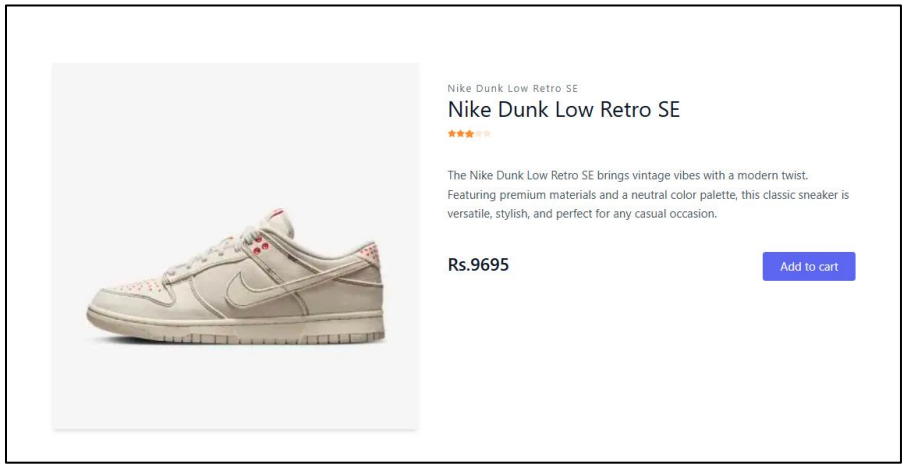
2. **Data Fields:**
   Each product detail page should include the following fields:
   - o **Product Description:** A detailed explanation of the product, fetched from the backend.
   - o **Price:** Displayed prominently for clear visibility.

3. **Integration with Product Listing:**
   - o 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:**
   - o Use Tailwind CSS for a clean and responsive design.
   - o 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



Step 3: Search Bar with Price Filter

Objective:

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

Implementation Plan:

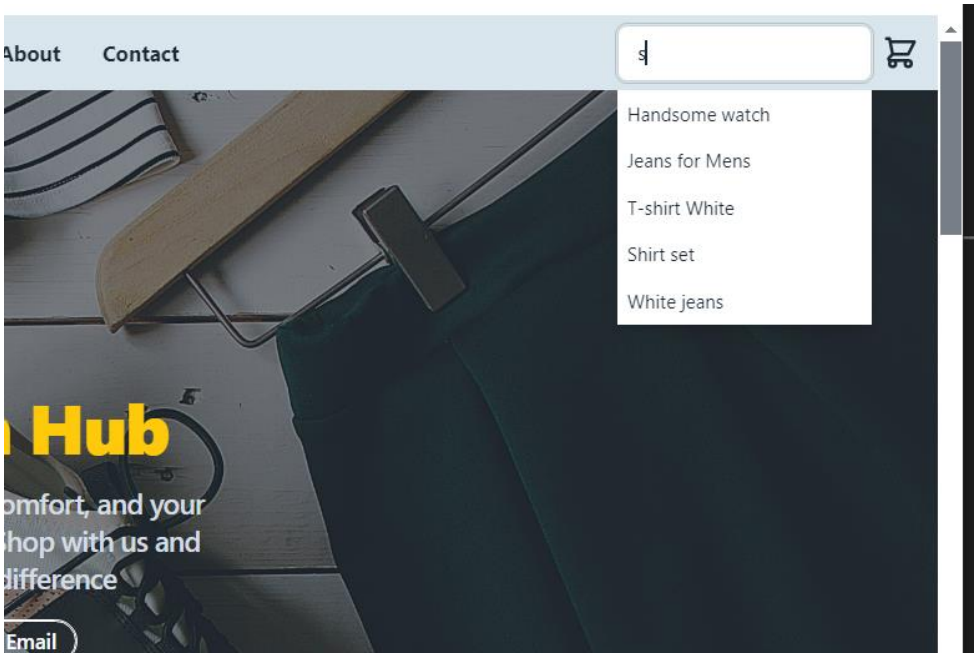
1. Search Bar Functionality:

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

src > app > components > search.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



Step 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:**

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

2. **Cart Data:**

   o Include details for each product in the cart:
   - Product Name
   - Price
   - Quantity

   o Calculate and display the **total price** dynamically based on the items in the cart.

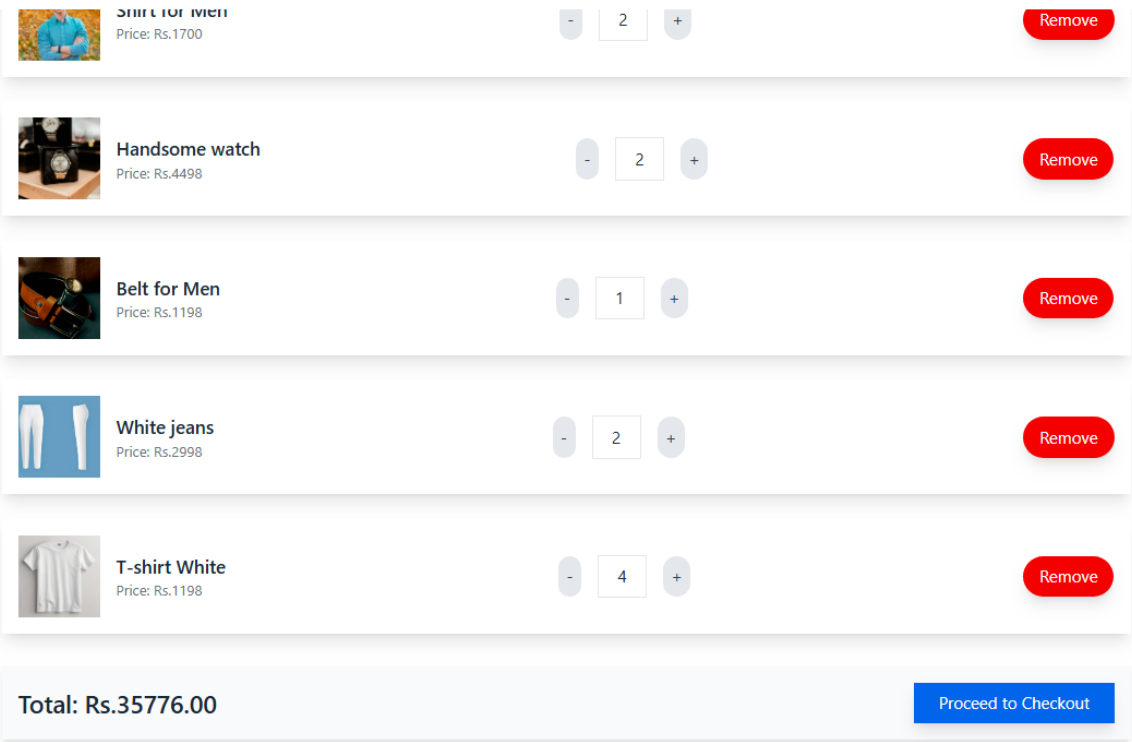
3. **Cart Interactions:**

   o Allow users to **increase or decrease the quantity** of items.

   o 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



Features Implemented:

1. **Dynamic Item Display:**
   o Each item in the cart is displayed with its name, price, and quantity.
   o Subtotal for each item is dynamically calculated.

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

3. **Total Price Calculation:**
   o The total price updates dynamically as items are added or quantities are changed.

4. **Remove Item:**
   o 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:**
   o Dynamically displayed products in a grid layout with details such as product name, price, image, and stock status.

2. **Product Detail Component:**
   o Built individual product pages using dynamic routing in Next.js, including fields like product description, price, and image.

3. **Search Bar and Filters:**
   o Implemented functionality to filter products by name or tags and added price filters (high to low and low to high).

4. **Cart Component:**
   o Displayed items added to the cart, quantity management, and total price calculation with dynamic updates.