Product component Listing

Prepared By: Umair Jami

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:

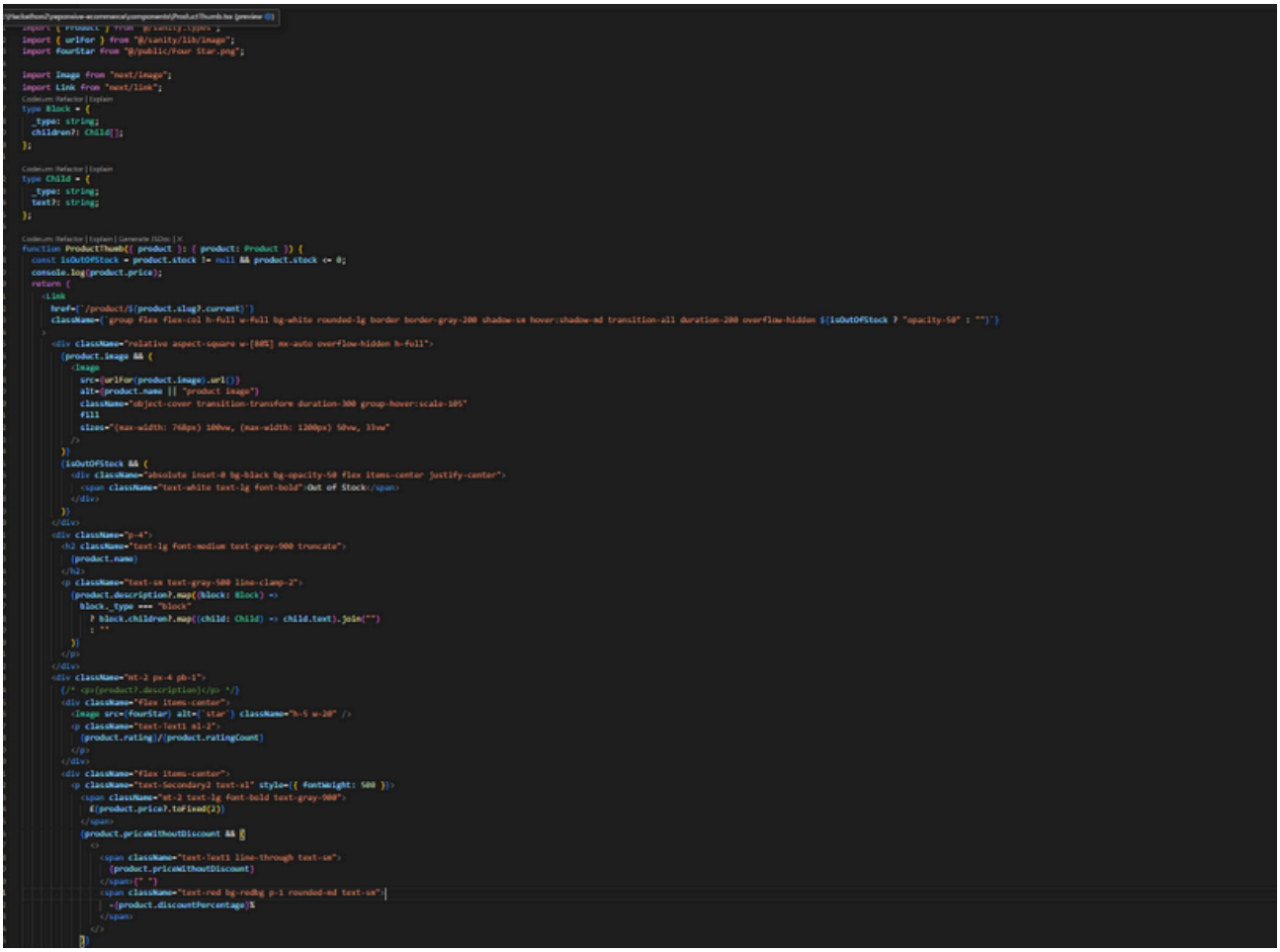
1. Fetch product data dynamically using Sanity CMS or an external API.
2. Display the data in a grid layout of cards with the following details:
    - Product Name
    - Price
    - Image
    - Discount Price
    - Rating
3. Ensure responsiveness across devices.
4. Implement modularity by breaking the component into smaller, reusable parts.

Tools & Technologies:

- Framework: Next.js
- CMS: Sanity CMS
- Styling: Tailwind CSS or plain 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.

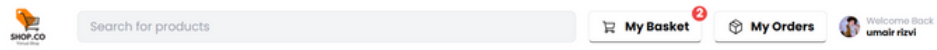


Common components

Navbar

Add navbar functionality with clerk authentication

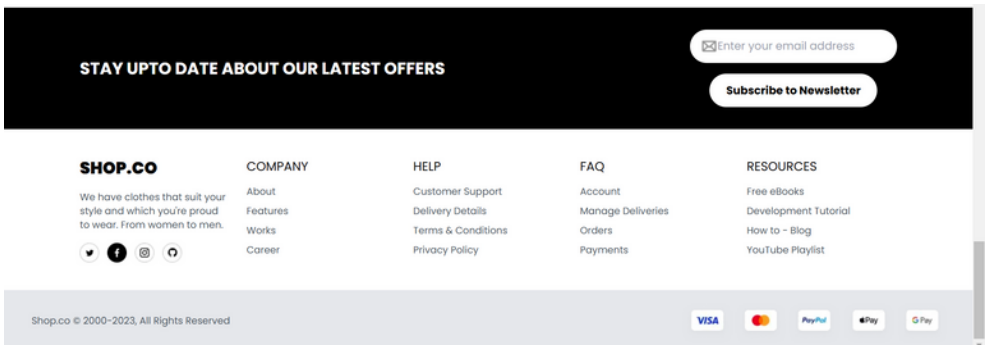
Ui display



Footer

Footer with functional links and company details

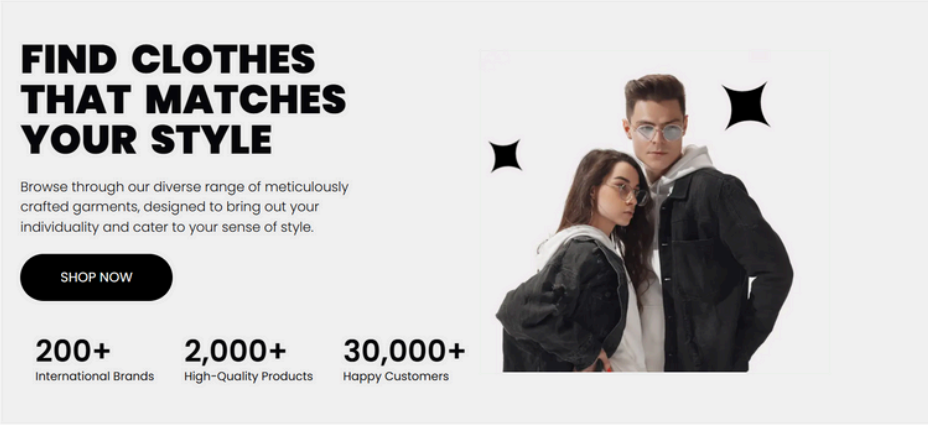
Ui display



Banner component

Banner component with great look and shop now Button from where you can go to all item display component

Ui dispaly



Coupon Code component

Coupon code component show coupon code and speacial Sales and discount value

Ui dispaly

By using this code You can get discont at the time of checkout

2. Product Detail Component

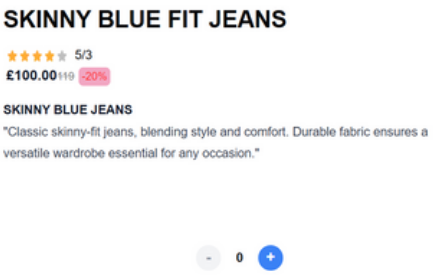
Objective:
Develop individual product detail pages using dynamic routing in Next.js. These pages will display detailed information about each product, including:

- **Name Product**
- **Description**
- **Price**
- **Cart btn**

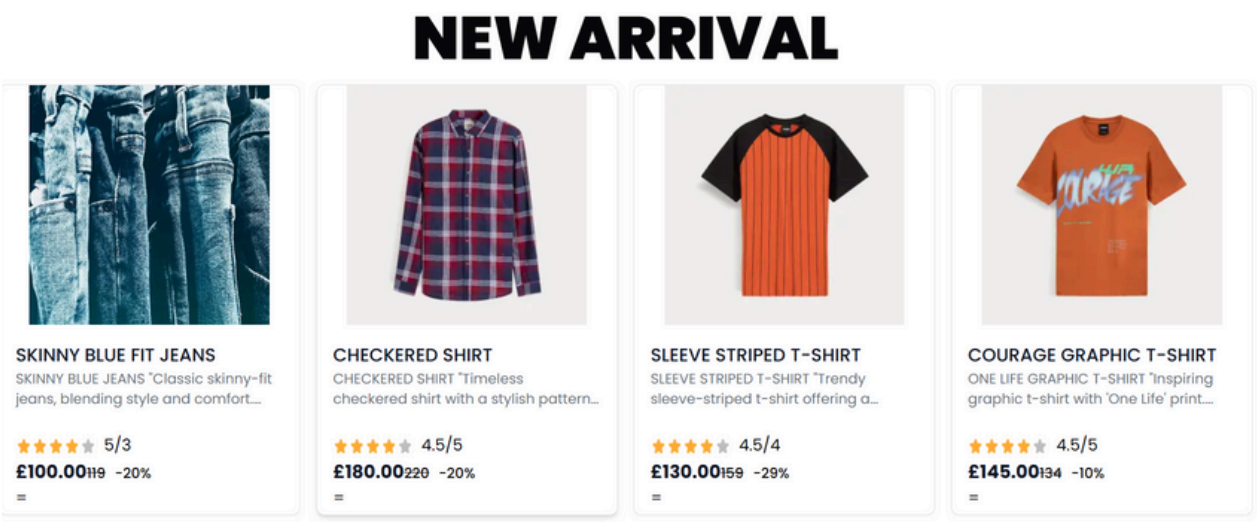
Implementation Plan:

1. Dynamic Routing:
    - Create dynamic routes using the [product].tsx file in the pages/ (store) 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.

4. Styling and Layout:
    - Use Tailwind CSS for styling and responisve design responsive design.
    - Ensure the layout highlights the product description and price for user clarity.



UI Display OF Product Detail Page:



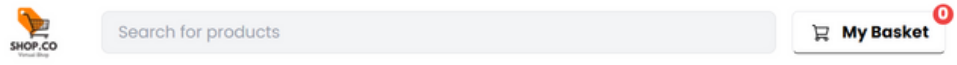
Step 3: Search Bar with Price Filter

To implement a search bar for find any product easily

1. Search Bar Functionality:
    - Filter products based on their name.
    - Update the product list in real-time as the user types.

```
app > (store) > search > page.tsx > SearchPage
4
    Codeium: Refactor | Explain | Generate JSDoc | ×
5   const SearchPage = async ({
6     searchParams,
7   }: {
8     searchParams: Promise<{ query: string }>;
9   }) => {
10    const { query } = await searchParams;
11    const products = await searchProductsByName(query);
12    if (!products.length) {
13      return (
14        <div className="flex flex-col items-center justify-top min-h-screen bg-gray-100 py-4">
15          <div className="bg-white p-8 rounded-lg shadow-md w-full max-w-4xl">
16            <h1>No Preducts found for: {query}</h1>
17            <p className="text-gray-500 tetx-center">
18              Try searching for something else
19            </p>
20          </div>
21        </div>
22      );
23    }
24
25    return (
26      <div className="flex flex-col justify-top min-h-screen bg-gray-100 p-4">
27      <div className="bg-white p-8 rounded-lg shadow-md w-full max-w-4xl">
28        <h1 className="text-3xl font-bold mb-6 text-center">
29          Search Results for: {query}
```

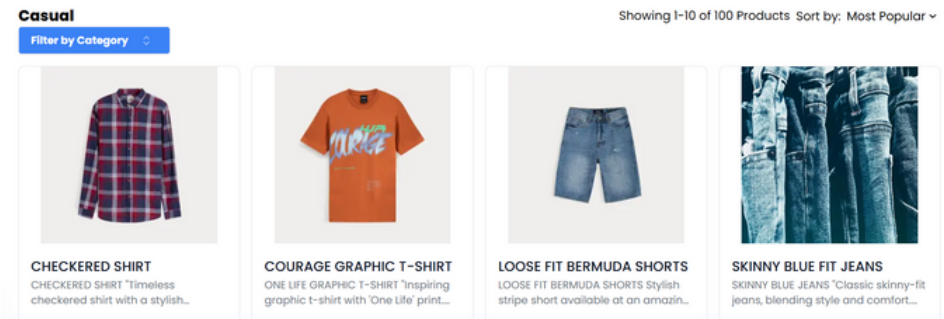
UI Display :



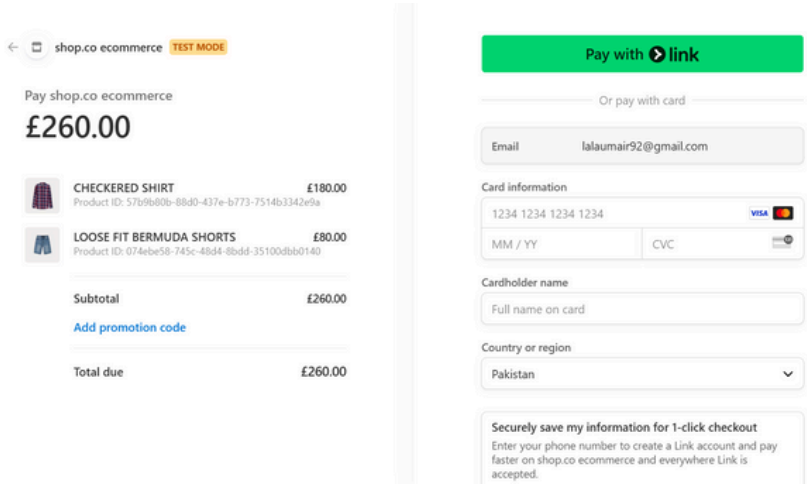
2. Category filter
    - Add categoy bar for search any product by his category

```tsx
import ProductsView from "@/components/ProductsView";
import { getAllCategories } from "@/sanity/lib/products/getAllCategories";
import { getProductsByCategory } from "@/sanity/lib/products/getProductsByCategory";

Codeium: Refactor | Explain | Generate JSDoc | ×
async function CategoryPage({ params }: { params: Promise<{ slug: string }> }) {
  const { slug } = await params;
  const products = await getProductsByCategory(slug);
  const categories = await getAllCategories();
  return (
    <div className="flex flex-col items-center justify-top min-h-screen bg-gray-100 p-4">
      <h1 className="text-4xl font-bold mb-6 text-center">
        {slug
          .split("_")
          .map((word) => word.charAt(0).toUpperCase() + word.slice(1))
          .join(" ")}{" "}
        Collection
      </h1>
      <ProductsView products={products} categories={categories} />
    </div>
  );
}

export default CategoryPage;
```

UI Display:



- ***checkout Page***



Features Implemented:

1 ○ order product detail in sidebar

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.

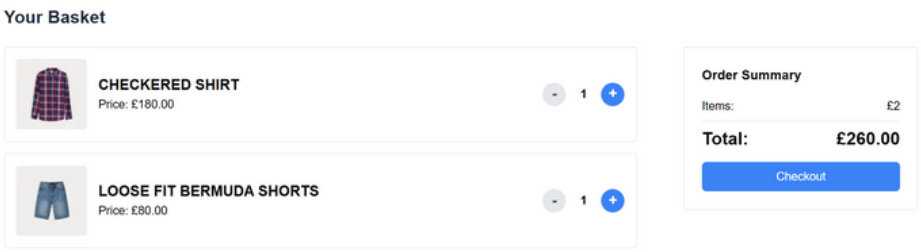
Implementation Plan:

1. State Management:
   - Use **zustand** for storing cart data.
2. Cart Data:
   - Include details for each product in the cart:
     - Product Name
     - Price Price
     - Product Image
   - o 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.

```
addItem: (product) =>
  set((state) => {
    const existingItem = state.items.find(
      (item) => item.product._id === product._id
    );
    if (existingItem) {
      return {
        items: state.items.map((item) =>
          item.product._id === product._id
            ? { ...item, quantity: item.quantity + 1 }
            : item
        ),
      };
    }
    return {
      items: [...state.items, { product, quantity: 1 }],
    };
  }),
Codeium: Refactor | Explain | Generate JSDoc | ×
removeItem: (productId) =>
  set((state) => ({
    items: state.items.reduce((acc, item) => {
      if (item.product._id === productId) {
        if (item.quantity > 1) {
          acc.push({ ...item, quantity: item.quantity - 1 });
        }
      } else {
        acc.push(item);
      }
      return acc;
    }, [] as BasketItem[]),
  })),
clearBasket: () => set({ items: [] }),
Codeium: Refactor | Explain | Generate JSDoc | ×
getTotalPrice: () => {
  return get().items.reduce(
    (total, item) => total + (item.product.price ?? 0) * item.quantity,
    0
  );
},
Codeium: Refactor | Explain | Generate JSDoc | ×
getItemCount: (productId) => {
```

UI Display Of Cart Page:



Features Implemented:

1. Dynamic Item Display:
   - Each item in the cart is displayed with its image, 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.

Step 6: **Completed order Page**

Objective:

To create a order page first snd to to sanity after checkout and then display it in the completed order page

---

```
const orders = await getMyOrders(userId);

return (
  <div className="flex flex-col items-center justify-center min-h-screen bg-gray-50 p-4">
    <div className="bg-white p-4 sm:p-8 rounded-xl shadow-lg w-full max-w-4xl">
      <h1 className="text-4xl font-bold text-gray-900 tracking-tight mb-8">
        My Orders
      </h1>
      {orders.length === 0 ? (
        <div className="text-center text-gray-600">
          <p>You have not placed any orders yet.</p>
        </div>
      ) : (
        <div className="space-y-6 sm:space-y-8">
          {orders.map((order) => (
            <div
              key={order.orderNumber}
              className="bg-white border border-gray-200 rounded-lg shadow-sm overflow-hidden"
            >
              <div className="p-4 sm:p-6 border-b border-gray-200">
                <div className="flex flex-col gap-4 sm:flex-row sm:justify-between sm:items-center mb-4">
                  <div>
                    <p className="text-sm text-gray-600 mb-1 font-bold">
                      Order Number
                    </p>
                    <p className="font-mono text-sm text-green-600 break-all">
                      {order.orderNumber}
                    </p>
                  </div>
                  <div className="sm:text-right">
                    <p className="text-sm text-gray-600 mb-1">Order Date</p>
                    <p className="font-medium">
                      {order.orderDate
```