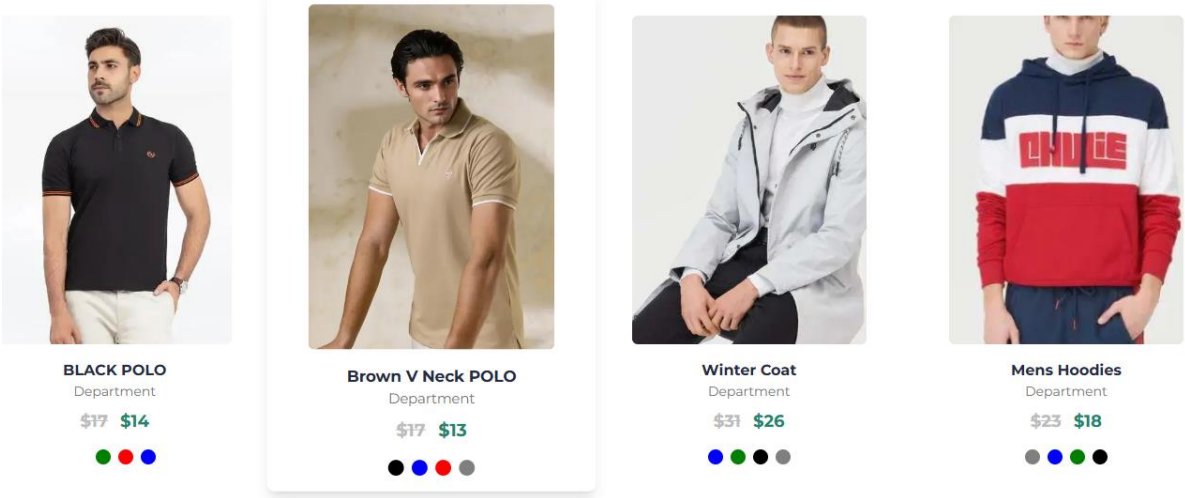
# DAY 4 - BUILDING DYNAMIC FRONTEND COMPONENTS

## (Banadage E-Commerce Store)

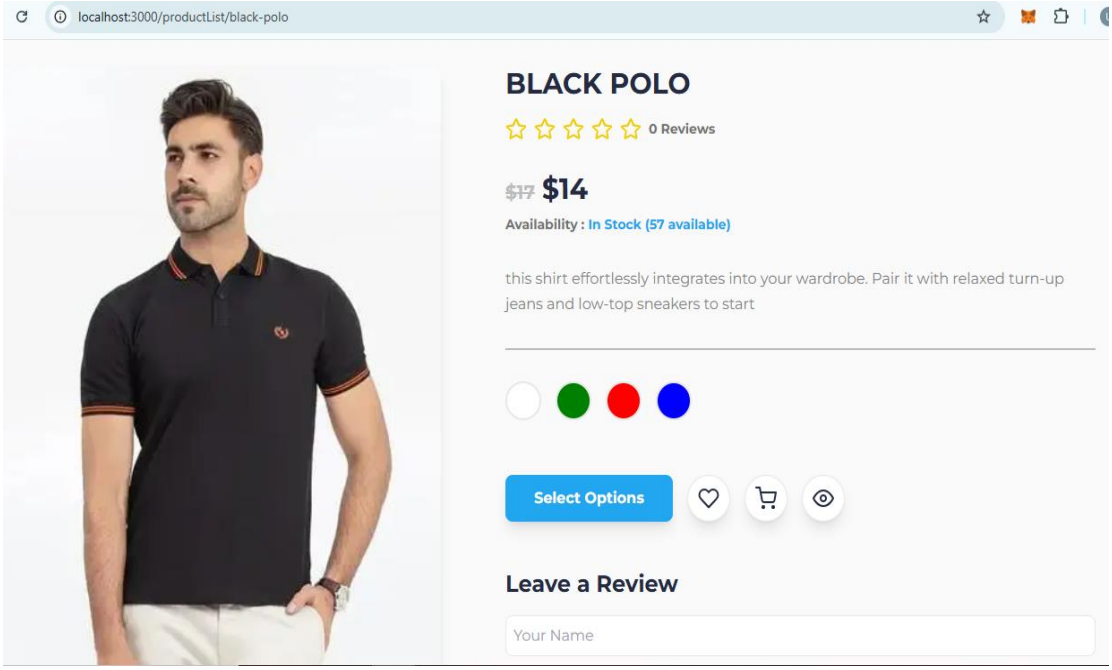
Screenshot of :

### 1. Product Listing Component:

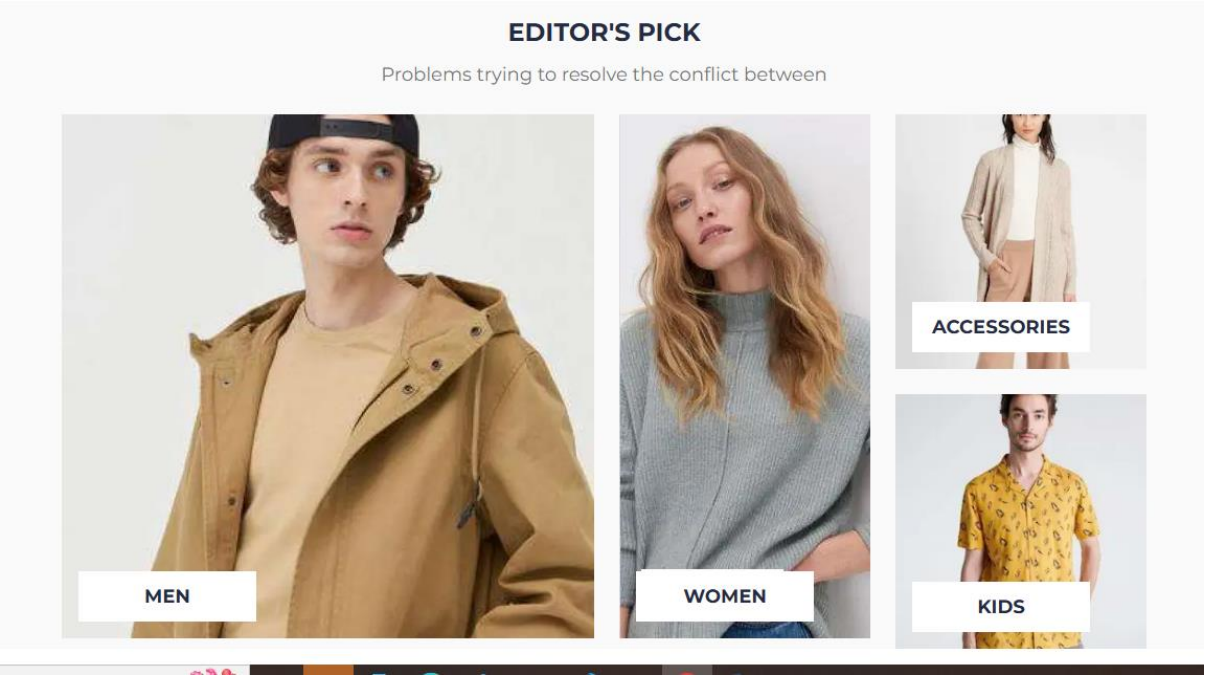



Screenshot of :

### 2. Product Detail Component:

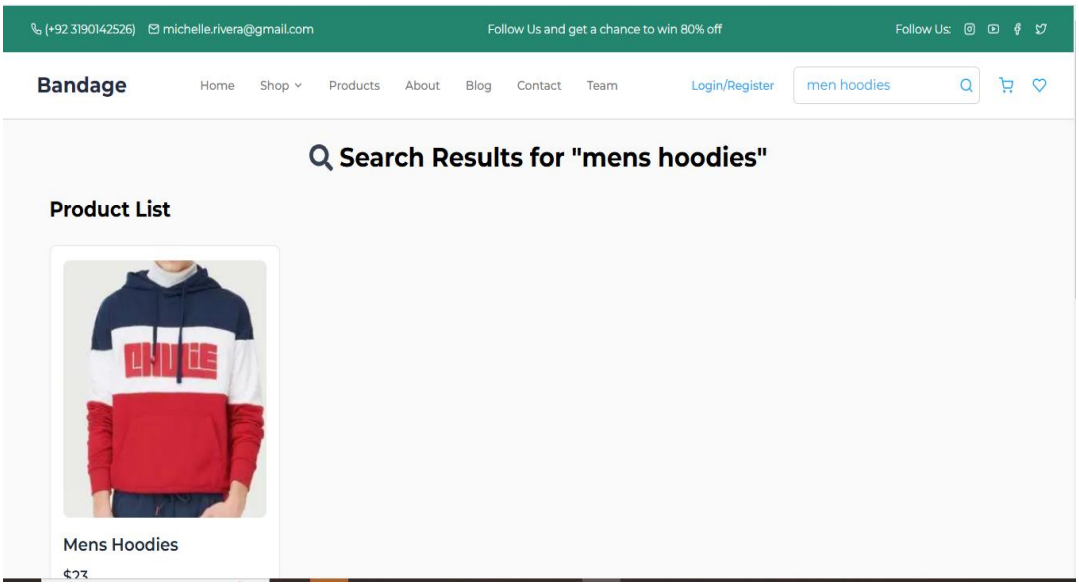


## 3 .Category Component:




Screenshot of :

## 4.Search Bar:



## Search Bar Funtionality :

- The search bar extracts the query from the URL and updates the searchQuery state.
- It then  triggers fetchSearchResults to retrieve matching products from Sanity.
- The results are stored in state variables and displayed dynamically. A loading indicator is shown while fetching data

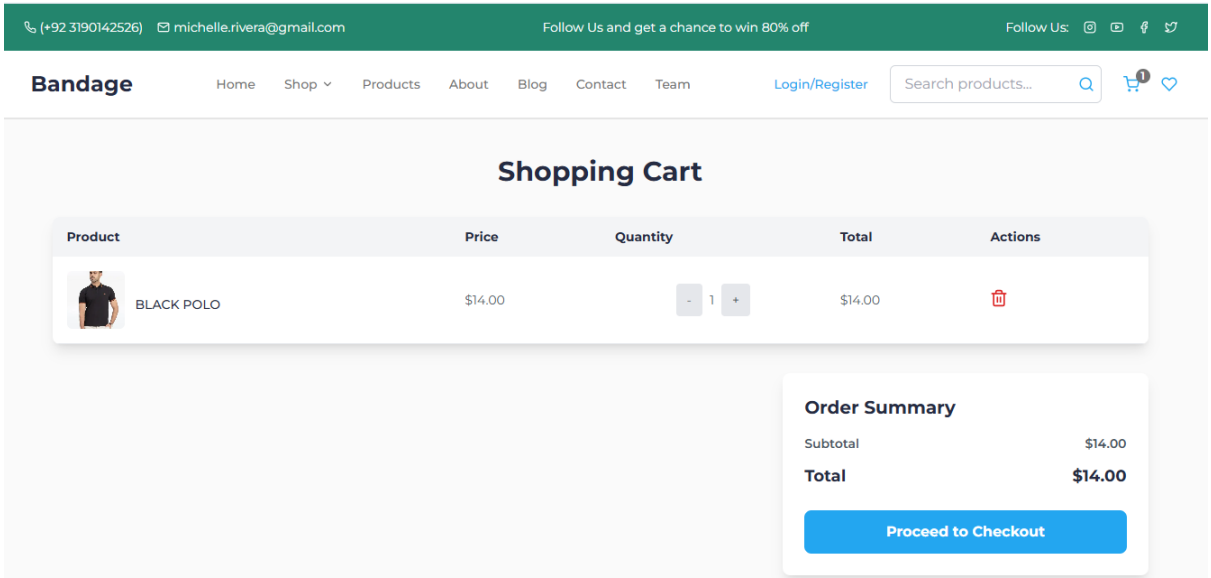
## Code Snnipet:

```
src > app > (main) > search > page.tsx > Product
interface Product {
Tabnine | Edit | Explain
}const SearchResultsPage = () => {
  const [searchQuery, setSearchQuery] = useState("");
  const [productListResults, setProductListResults] = useState<Product[]>([]);
  const [productResults, setProductResults] = useState<Product[]>([]);
  const [isLoading, setIsLoading] = useState(false);

  // Get the search query from the URL
  useEffect(() => {
    const query = new URLSearchParams(window.location.search).get("q");
    if (query) {
      setSearchQuery(query);
      fetchSearchResults(query);
    }
  }, []);

  // Fetch search results from Sanity
  const fetchSearchResults = async (query: string) => {
    setIsLoading(true);
    try {
      // Fetch productList results
      const productListData: Product[] = await client.fetch(
        `*[_type == "productList" && name match $searchQuery] {
          _id,
          name,
          "image": image.asset->url,
          price,
          slug,
        }`,
        { searchQuery: `*${query}*` }
      );

      // Fetch product results
      const productData: Product[] = await client.fetch(
        `*[_type == "products" && name match $searchQuery] {
          _id,
          name,
          "image": image.asset->url,
          price,
          slug,
        }`,
        { searchQuery: `*${query}*` }
      );

      setProductListResults(productListData);
      setProductResults(productData);
    } catch (error) {
main*  ⊗0 ⚠0  AWS
```

## 5. Cart Component:

- `"use client"`: ensures the component runs on the client side in Next.js, allowing interactivity.
- The `useCart` hook is used to manage cart state, enabling adding, removing, and updating quantities of items.
- The `ClerkProvider` is used for authentication, wrapping the `Header` to manage user sessions securely.
- The `Image` component from `next/image` optimizes images for performance and lazy loading.

Screenshot of :

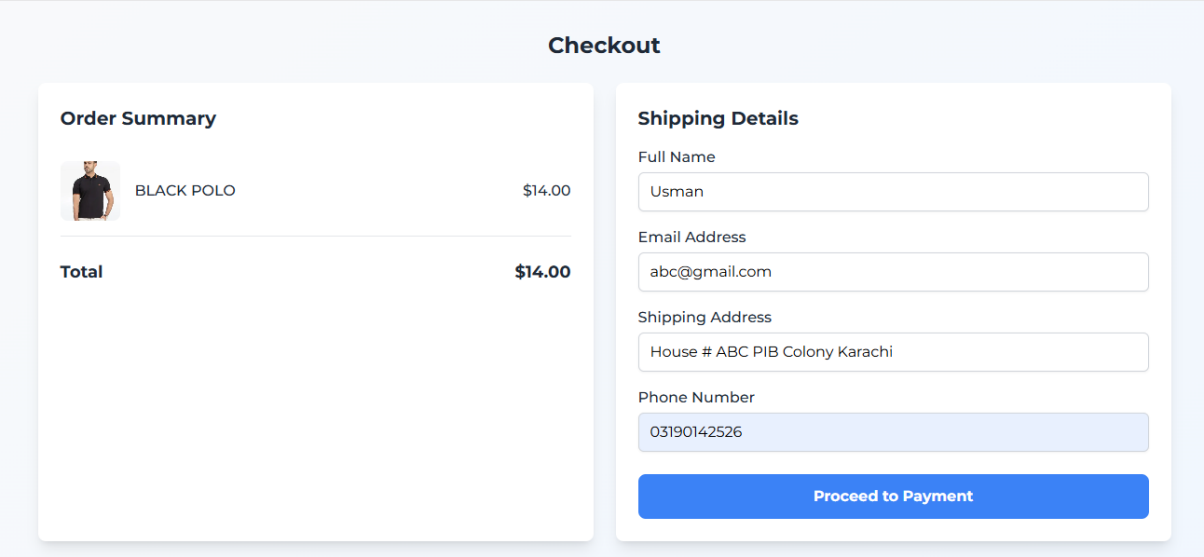



## 6. Checkout Flow Component:

It uses Stripe for payment processing and integrates with a cart context to manage items. Users can fill in their shipping details, and once the form is submitted, it updates the stock and creates a Stripe Checkout session. After successful order creation, the user is redirected to Stripe for payment. The checkout form is validated using React Hook Form and Zod, ensuring all required fields are filled. Once the payment is processed, order details are saved in Sanity for further tracking.

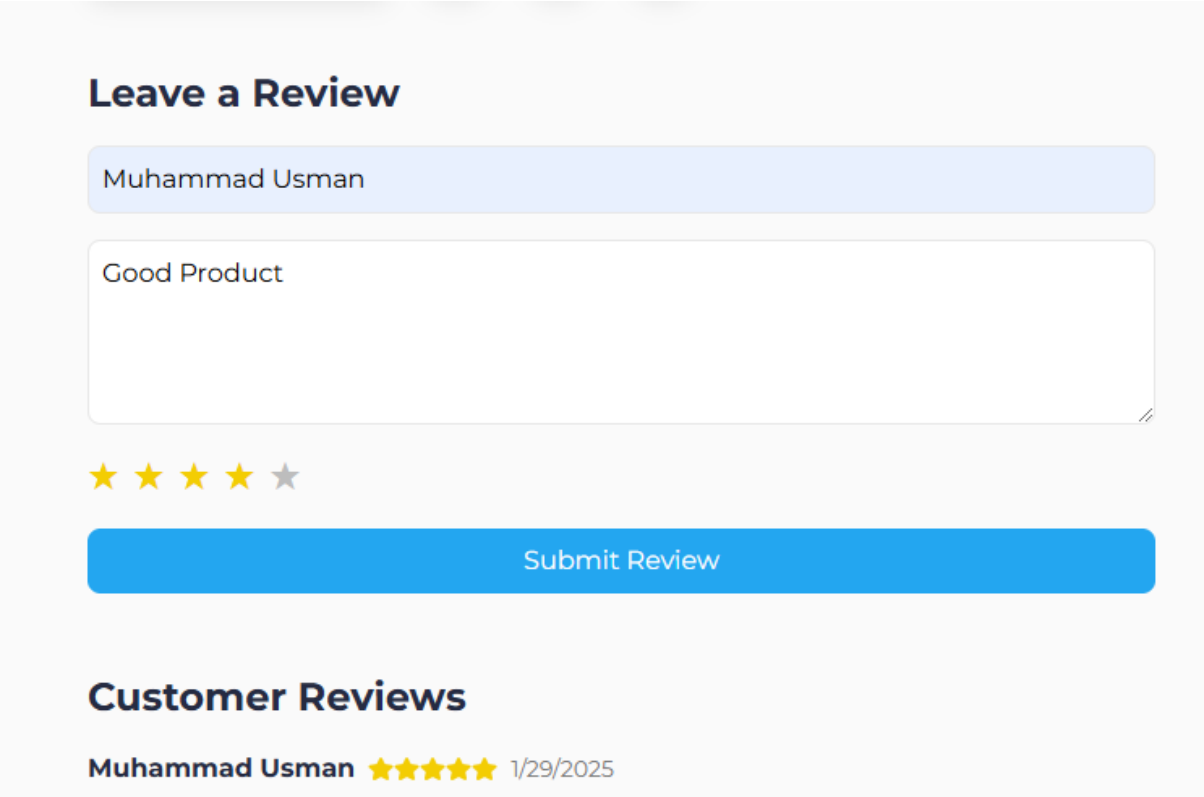
Screenshot of :

## Checkout Flow Component:




Screenshot of :

## 7. Reviews and Ratings



## Final CheckList

| Frontend Component Development: | Styling and Responsiveness: | Code Quality: | Documentation and Submission: | Final Review: |
|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ |