Day 4 - Dynamic Frontend Components - ShoeVibe

Steps Taken to Build and Integrate Components

Component Development:

- **Navbar:**
  - Integrated a dropdown search feature that dynamically filters products by name and navigates to the product page when clicked.
- **Product Page:**
  - Fetched individual product data based on id using Next.js API routes.
- **Cart Management:**
  - Used React context (CartContext) to manage global cart state.
- **Styling:**
  - Used React context (CartContext) to manage global cart state.

Challenges Faced and Solutions Implemented

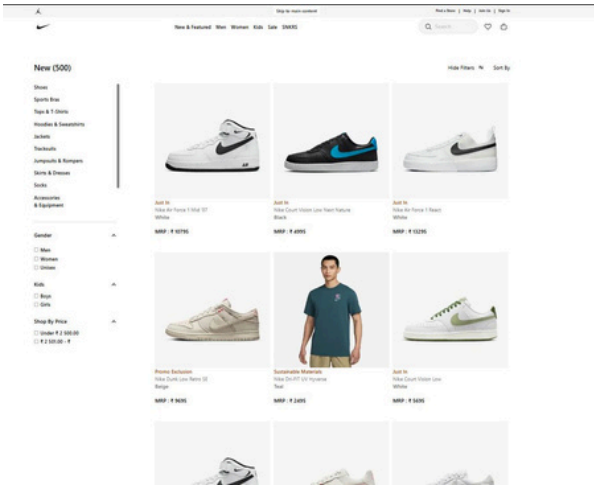
- **Search Functionality Across Components:**
  - **Challenge:** Integrating the search functionality in the Navbar with the allproducts route while maintaining state.
  - **Solution:** Dynamically filtered products based on user input and used programmatic navigation to route to the relevant product.
- **API Data Handling:**
  - **Challenge:** Handling errors and loading states during API fetches.
  - **Solution:** Implemented robust error handling with try-catch blocks and loading states using useState.

Best Practices Followed

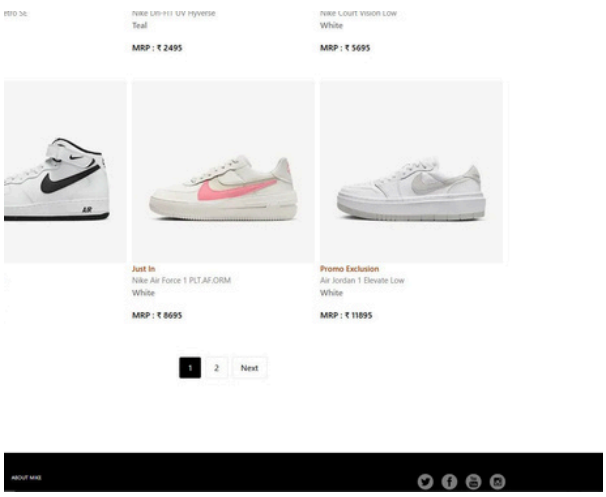
- **Code Organization:**
  - Followed modular component structure for better reusability and maintainability.
- **State Management:**
  - Leveraged React context for cart management, ensuring global state consistency.
- **Responsive Design:**
  - Maintained consistent and mobile-friendly layouts using Tailwind CSS utilities.
- **Performance Optimization:**
  - Used debouncing in the search functionality to minimize API calls and improve efficiency.

Functional Deliverables

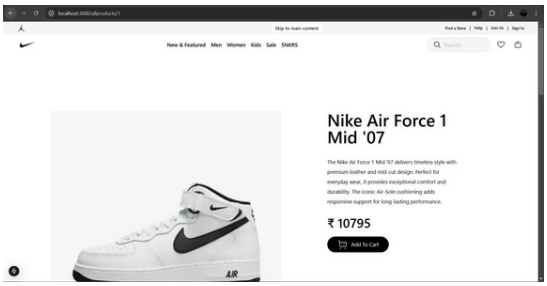
Product listing page



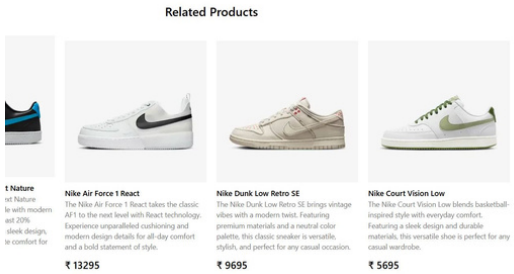
Pagination



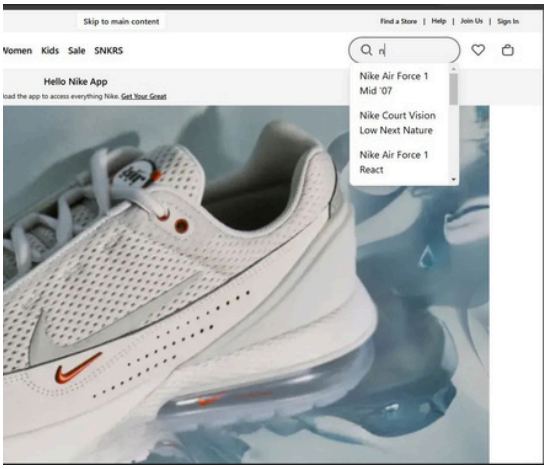
Detail pages



Related products



Search bar



Code Deliverables

Product List

```
const page = () => {

  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);


  const [currentPage, setCurrentPage] = useState(1);
  const itemsPerPage = 9;

  useEffect(() => {
    const fetchProducts = async () => {
      setLoading(true);
      setError(null);
      try {
        const res = await fetch(`/api/products`);
        if (!res.ok) {
          throw new Error("Failed to fetch products");
        }
        const data = await res.json();
        setProducts(data);
      } catch (error) {
        setError(error.message);
      } finally {
        setLoading(false);
      }
    };
    fetchProducts();
  }, []);


  const indexOfLastProduct = currentPage * itemsPerPage;
  const indexOfFirstProduct = indexOfLastProduct - itemsPerPage;
  const currentProducts = products.slice(indexOfFirstProduct, indexOfLastProduct);
```

Search Bar

```
24
25    const [searchQuery, setSearchQuery] = useState('');
26    const [products, setProducts] = useState([]);
27    const [filteredProducts, setFilteredProducts] = useState([]);
28    const [isDropdownVisible, setIsDropdownVisible] = useState(false);
29
30    useEffect(() => {
31      const fetchProducts = async () => {
32        try {
33          const res = await fetch('/api/products');
34          if (!res.ok) {
35            throw new Error('Failed to fetch products');
36          }
37          const data = await res.json();
38          setProducts(data);
39        } catch (error) {
40          console.error(error.message);
41        }
42      };
43
44      fetchProducts();
45    }, []);
46
47    useEffect(() => {
48      if (searchQuery) {
49        const filtered = products.filter((product) =>
50          product.productName.toLowerCase().includes(searchQuery.toLowerCase())
51        );
52        setFilteredProducts(filtered);
53        setIsDropdownVisible(true);
54      } else {
55        setIsDropdownVisible(false);
56      }
57    }, [searchQuery, products]);
```

API integration

```
import { NextResponse } from 'next/server';
import {client} from '../../../sanity/lib/client';

export async function GET() {
    try {
        const query = `*[_type == "product"]{
            id,
            productName,
            category,
            price,
            inventory,
            colors,
            status,
            "imageUrl": image.asset->url,
            description
        }`;
        const products = await client.fetch(query);

        return NextResponse.json(products, { status: 200 });
    } catch (error) {
        console.error('Error fetching products:', error);
        return NextResponse.json({ message: 'Failed to fetch products' }, { status: 500 });
    }
}
```