Hackathon Day 4 - Dynamic Frontend Components – Nike Ecommerce

---

Objective

The objective of Day 4 in the Hackathon is to design and develop dynamic frontend components that display marketplace data fetched from Sanity CMS or APIs. This includes creating modular, reusable components, implementing state management, and ensuring responsive and user-friendly designs.

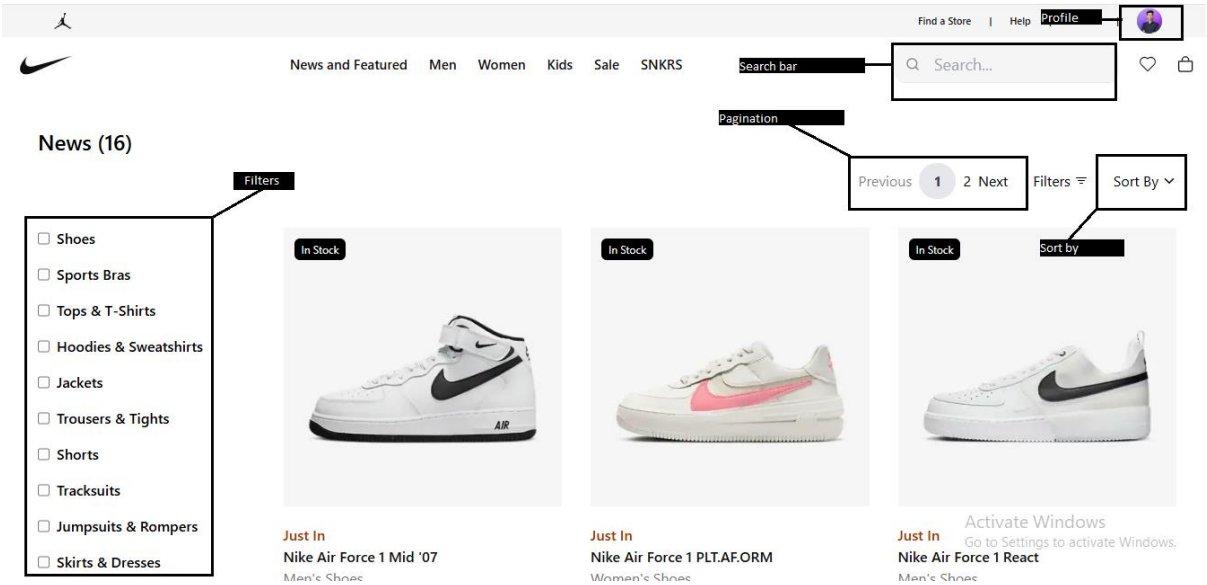
Key Learning Outcomes

1. Build dynamic frontend components to display data from Sanity CMS or APIs.
2. Implement reusable and modular components for scalability.
3. Apply state management techniques effectively.
4. Ensure responsive design and UX/UI best practices.
5. Replicate professional workflows to prepare for real-world client projects.

Components Built

1. Product Listing Component

- Product card contain price, category, stock status and total colours of product.



2. Product Detail Component

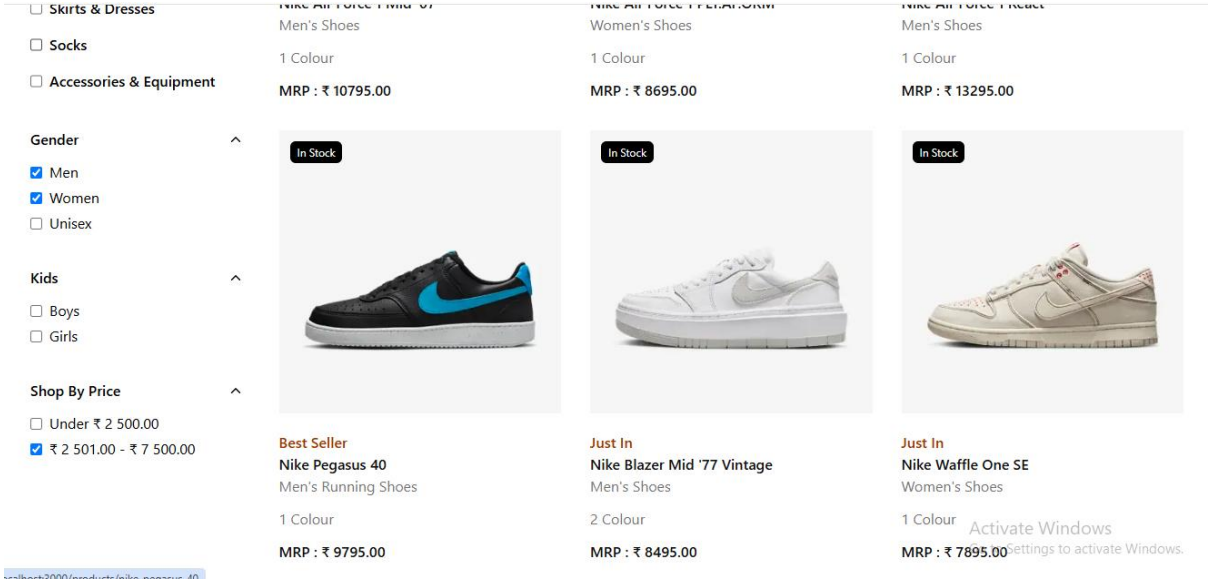
- I have created product detail component including colour, name , image, and much more.
- I also add Quantity control button.

3. Pagination Component

- Implemented pagination to break down large product lists into manageable pages.
- I added "Previous" and "Next" buttons along with numbered pagination.

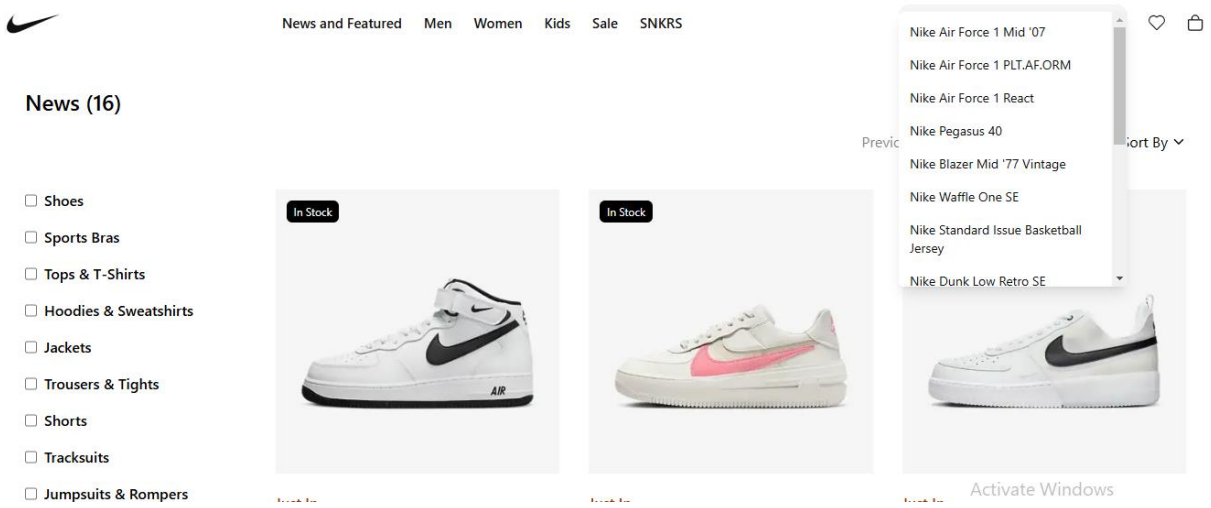
4. Filter Panel Component

- I added options to filter products by categories, price and gender.



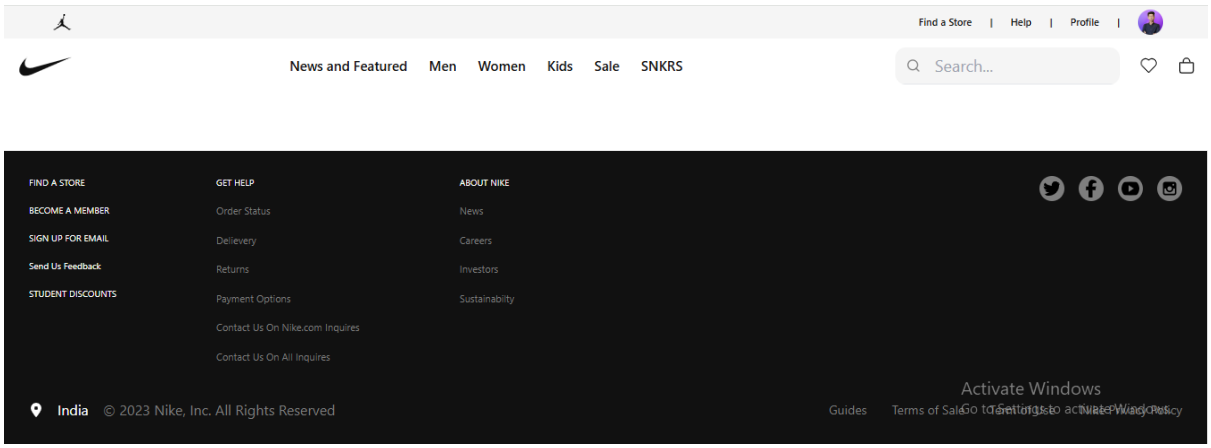
5. Search Bar Component

- I have implemented a search bar to filter products by name dynamically.



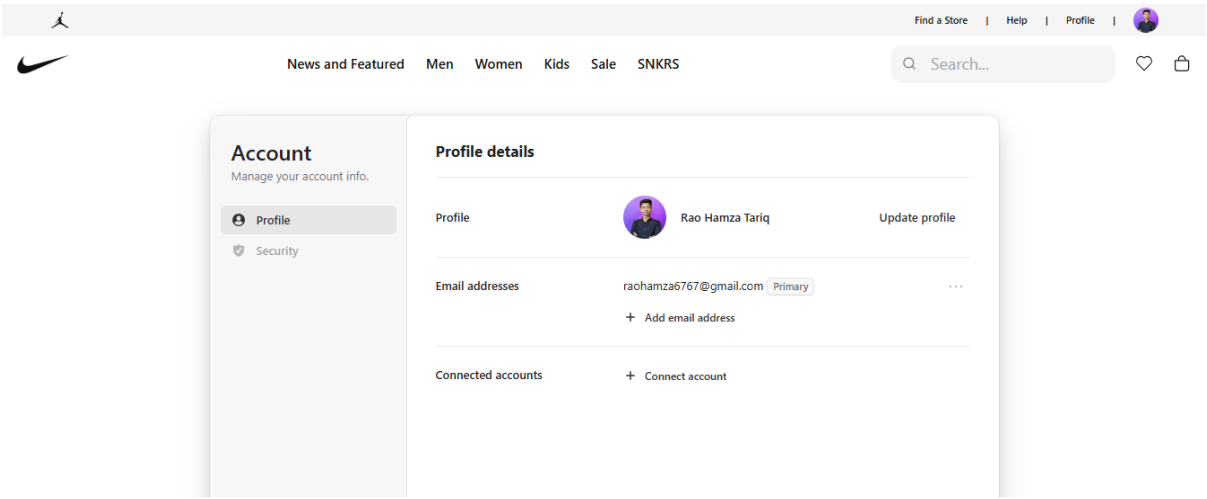
6. Responsive Header and Footer Components

- I made responsive header and footer with branding elements and navigation link such as sign in, sign up, products, cart, profile.



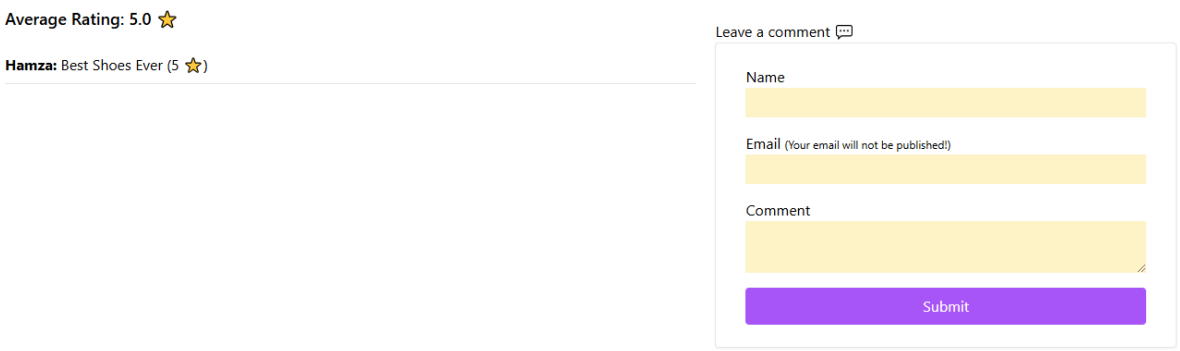
7. User Profile

- I use User Profile component of clerk. Later i will make my own component.



8. Review and Rating Section

- I add the review section where customer can review the product and also check other reviews and ratings



9. Notifications and Alerts

- I use Shadcn Toast component for alerks like for add to cart, remove from cart, and many more

---

Functional Features

Dynamic Data Fetching

- Make product api route which fetch product data from sanity

```
import { client } from '@/sanity/lib/client';
import { NextRequest, NextResponse } from 'next/server';

export async function GET(req:NextRequest) {
  try {
    const searchParams = req.nextUrl.searchParams
    const query = searchParams.get('query')
    const slug = searchParams.get('slug')
    if(slug){
      const data = await client.fetch(`*[_type == "product" && slug.current == "${slug}"][0]`);
    return NextResponse.json({ data }, { status: 200 });
    }
    if(query=="name"){
      const data = await client.fetch(`*[_type == "product"]{productName,slug}`);

    return NextResponse.json({ data }, { status: 200 });
    }
  const data = await client.fetch(`*[_type == "product"]`);
    return NextResponse.json({ data }, { status: 200 });

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

export async function POST(req:NextRequest) {
  const { slug } = await req.json();
  try {
    const data = await client.fetch(`*[_type == "product" && slug.current == "${slug}"][0]`);

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

- Connected the frontend to APIs to fetch and render data dynamically.

```
const Products =  () => {

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

  const [currentPage, setCurrentPage] = useState(0);
const [productsPerPage] = useState(12); // Adjust as needed
const [totalProducts, setTotalProducts] = useState(0);

  useEffect(()=>{
    const fetchData = async () => {
      try {
        const response = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/api/products`);
        if (!response) {
          throw new Error('Failed to fetch data');
        }
        const data = await response.json();
        setProducts(data.data)
        setTotalProducts(data.data.length);
      } catch (error) {
        console.error(error);
        return [];
      }
    };

    fetchData();
  },[])
```

Register User

- Make register route which register the user data in Sanity CMS after sign up

```
import { client } from '@/sanity/lib/client';
import { NextRequest, NextResponse } from 'next/server';

export async function POST(req: NextRequest) {
  const data = await req.json();

  const { email, firstName, lastName, dateOfBirth, country, gender } = data;

  try {
    // Validate user input (e.g., email format, password strength)
    if (!email || !firstName || !lastName) {
      return NextResponse.json({ message: 'Missing required fields' }, { status: 400 });
    }

    // Check if the email is already in use
    const existingUser = await client.fetch(`*[_type == "user" && email == $email][0]`, { email });

    if (existingUser) {
      return NextResponse.json({ message: 'Email already in use' }, { status: 409 }); // Conflict status
    }


    // Create a new user document in Sanity
    const newUser = await client.create({
      _type: 'user',
      email,
      firstName,
      lastName,
      dateOfBirth,
      country,
      gender,
    });

    return NextResponse.json({ message: 'Registration successful', newUser }, { status: 201 });
  } catch (error) {
    console.error('Registration error:', error);

    // Return a more detailed error message if available
    return NextResponse.json({ message: 'Registration failed', error }, { status: 500 });
```

Reusable and Modular Components

- Designed components such as ProductCard and Carousal to be reusable across the application.

Responsive Design

- Styled components with Tailwind CSS to ensure adaptability across devices.

Challenges Faced and Solutions

Challenge 1: API Data Fetching Errors

- **Problem:** API response errors and latency issues.
- **Solution:** Implemented error handling and loading states to enhance user experience.

Challenge 2: Managing State for Complex Features

- **Problem:** Combining filtering, pagination, and search functionality required consistent state updates.

- **Solution:** Used controlled states for categories and search inputs, ensuring synchronized updates across components.

Challenge 3: Ensuring Responsive UI

- **Problem:** Components did not render optimally on smaller screens.
- **Solution:** Leveraged Tailwind CSS utilities to create responsive layouts, such as grid-cols-1, sm:grid-cols-2, and lg:grid-cols-3.

Expected Output

1. A functional product listing page displaying dynamic data from Sanity CMS or APIs.
2. Individual product detail pages implemented using dynamic routing.
3. Functional pagination and category filters.
4. Search bar for dynamic product filtering.
5. Responsive and reusable components styled for professional appearance.
6. User Auth using Clerk with custom sign in and sign up page.

This documentation serves as the submission for Hackathon Day 4. It highlights the development of dynamic frontend components for a marketplace, including challenges faced, best practices followed, and the solutions implemented. The project adheres to professional standards and demonstrates a robust and scalable approach to frontend development.

Prepared by: RAO HAMZA TARIQ