

Day4 : Dynamic Frontend Components.

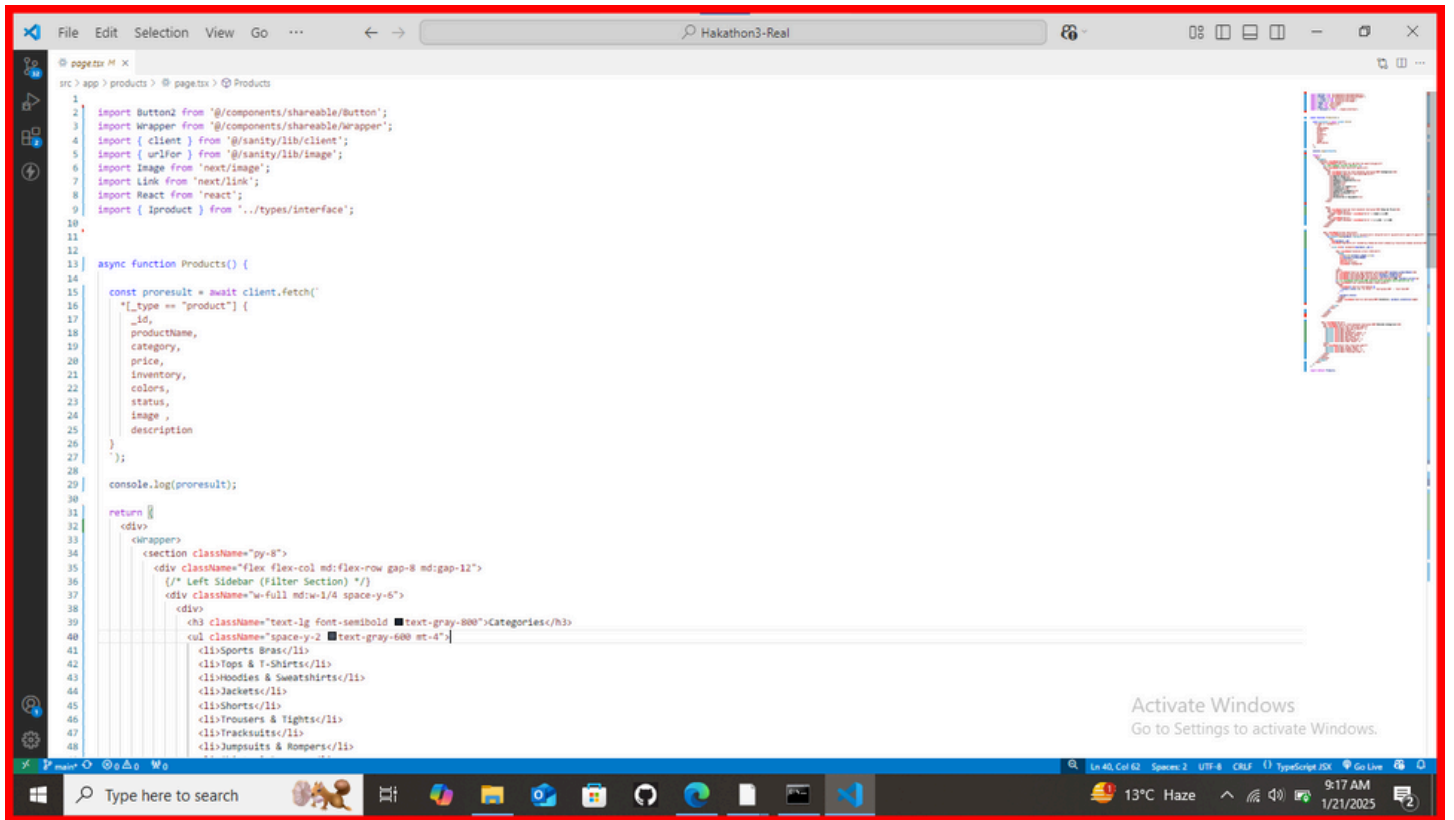
Functionalities:

Product listing page With dynamic data:

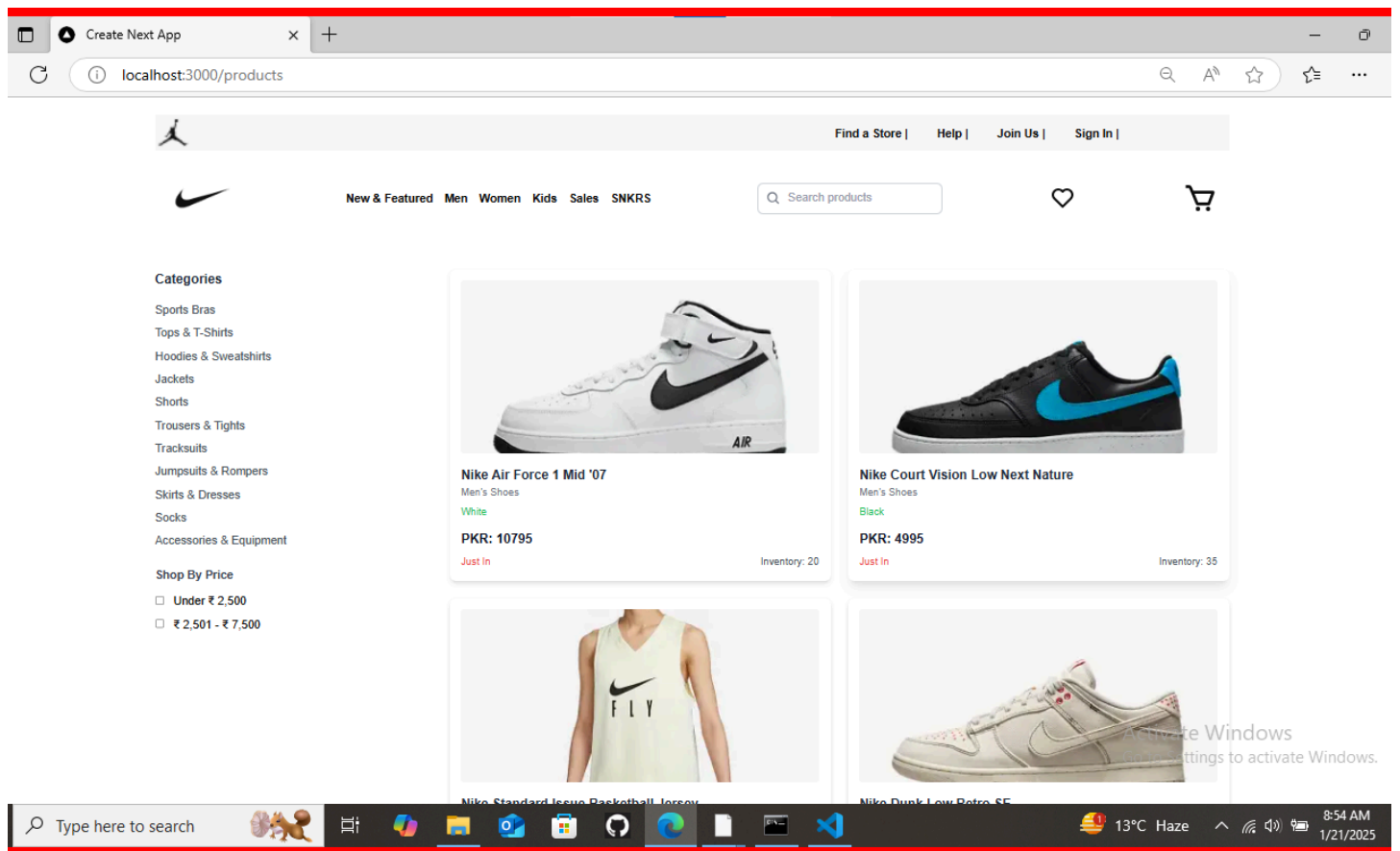
Description:-

The product list page dynamically display all the products by unique id .In product list page ,the products displays with name ,price, category ,inventory and Color

1-this is code



Product list page displays on the Browser :-

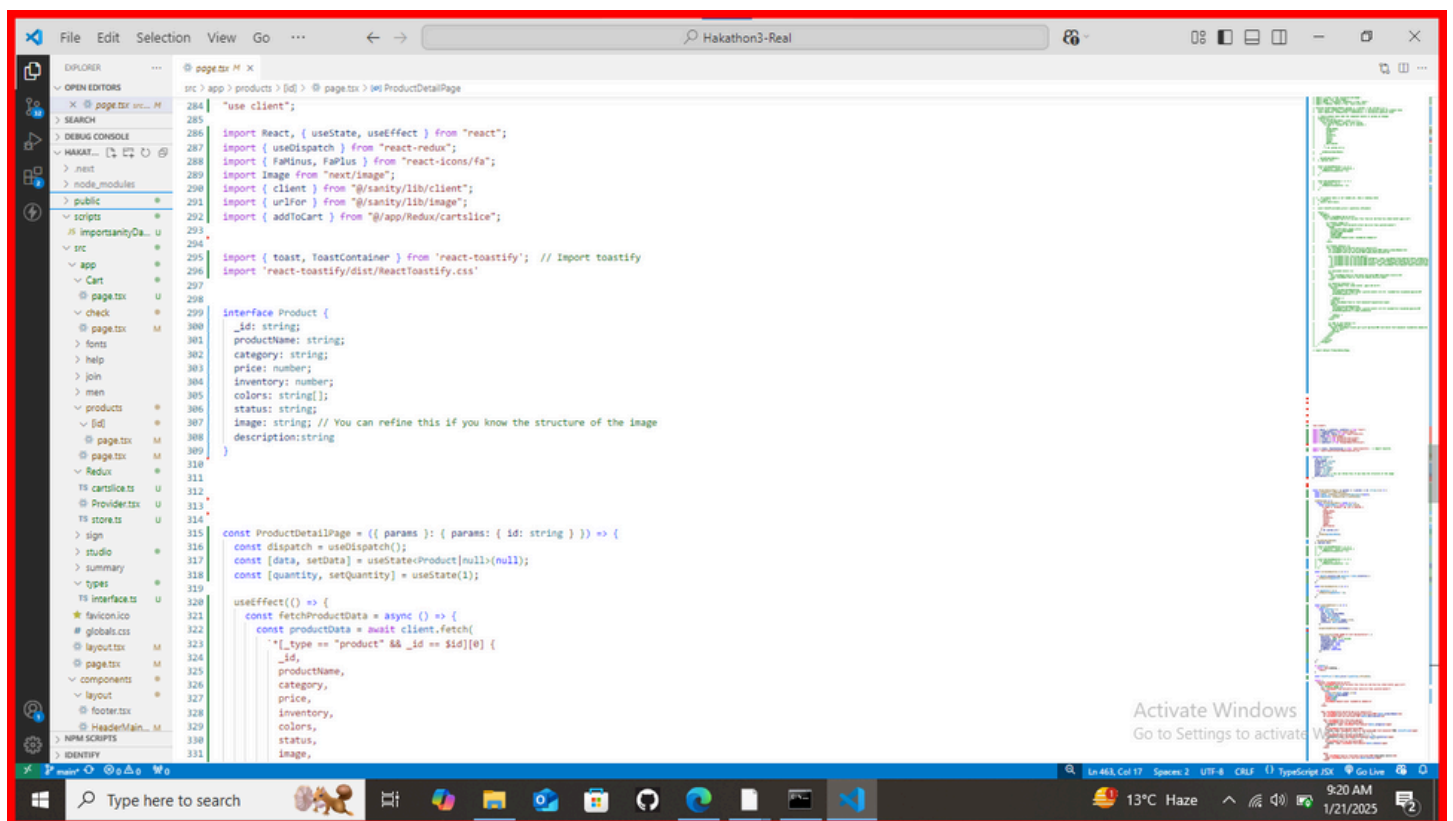


step 2: 3. Implementing the Product Detail Component (Dynamic Routing):-

Building a detailed page for each product that can be accessed by

clicking on a product from the product listing.

.How I Did It.

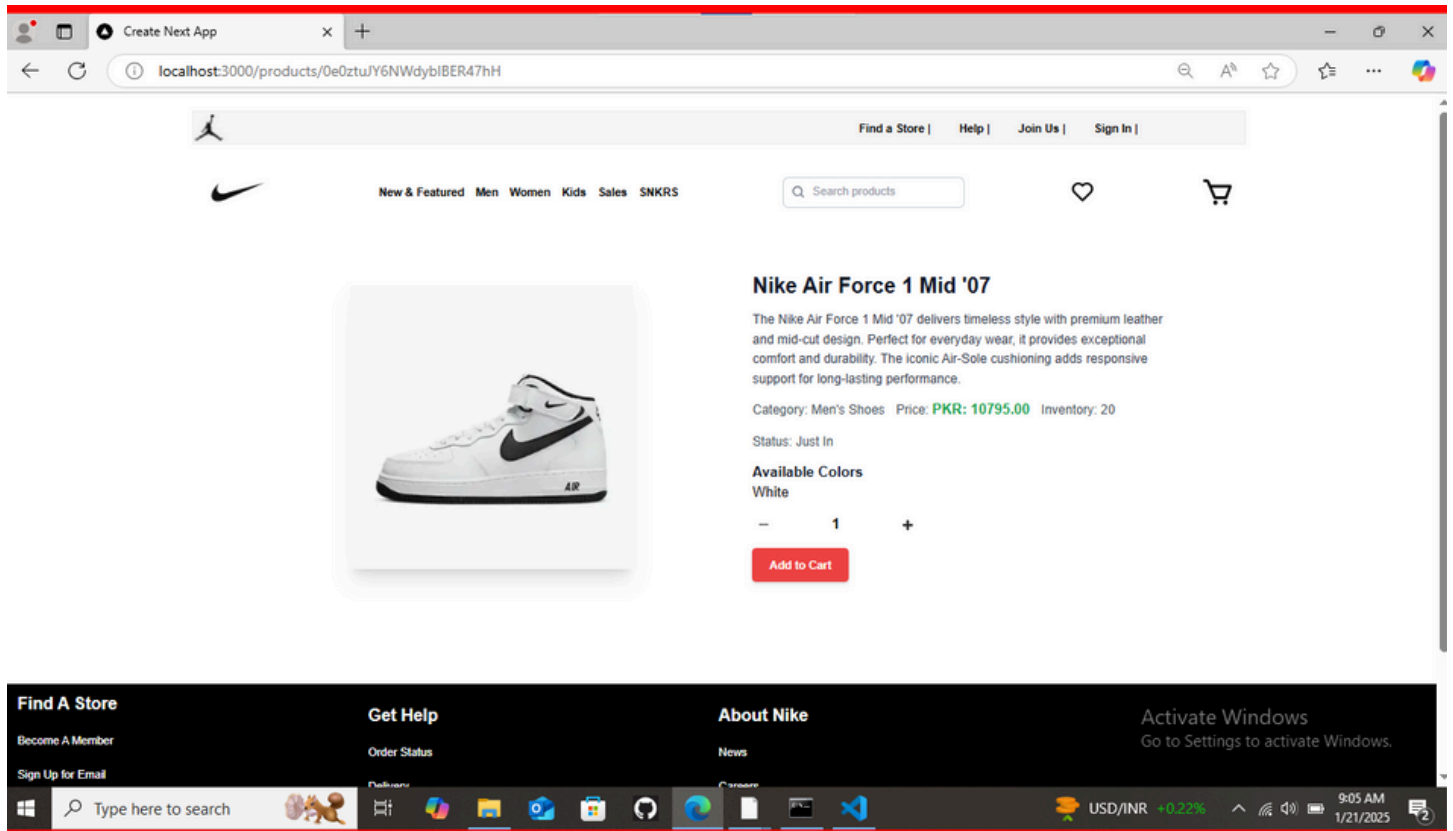


```
284 | "use client";
285 |
286 | import React, { useState, useEffect } from "react";
287 | import { useDispatch } from "react-redux";
288 | import { FaMinus, FaPlus } from "react-icons/fa";
289 | import Image from "next/image";
290 | import { client } from "@sanity/lib/client";
291 | import { urlFor } from "@sanity/lib/image";
292 | import { addToCart } from "@app/Redux/cartSlice";
293 |
294 |
295 | import { toast, ToastContainer } from 'react-toastify'; // Import toastify
296 | import 'react-toastify/dist/ReactToastify.css'
297 |
298 |
299 | interface Product {
300 |   _id: string;
301 |   productName: string;
302 |   category: string;
303 |   price: number;
304 |   inventory: number;
305 |   colors: string[];
306 |   status: string;
307 |   image: string; // You can refine this if you know the structure of the image
308 |   description: string;
309 | }
310 |
311 |
312 | const ProductDetailPage = ({ params }: { params: { id: string } }) => {
313 |   const dispatch = useDispatch();
314 |   const [data, setData] = useState<Product>[null](null);
315 |   const [quantity, setQuantity] = useState(1);
316 |
317 |   useEffect(() => {
318 |     const fetchProductData = async () => {
319 |       const productData = await client.fetch(
320 |         `*[_type == "product" && _id == ${id}][0] {
321 |           _id,
322 |           productName,
323 |           category,
324 |           price,
325 |           inventory,
326 |           colors,
327 |           status,
328 |           image,
329 |         }`
330 |       );
331 |     };
332 |     fetchProductData();
333 |   });
```

- 1.I utilized Next.js dynamic routing by creating a file under the pages/product/[id].js path.
- 2.Fetched product-specific data using the product ID in the URL [useRouter hook] and rendered it dynamically.
- 3.Included product description, price, category, color options, and inventory details, ensuring everything was pulled dynamically from the API or CMS.

4. In product detail page you can see the image, productName, Product price, Product Category, Product Color, Product Inventory

2-Dynamic page For Each Product:-



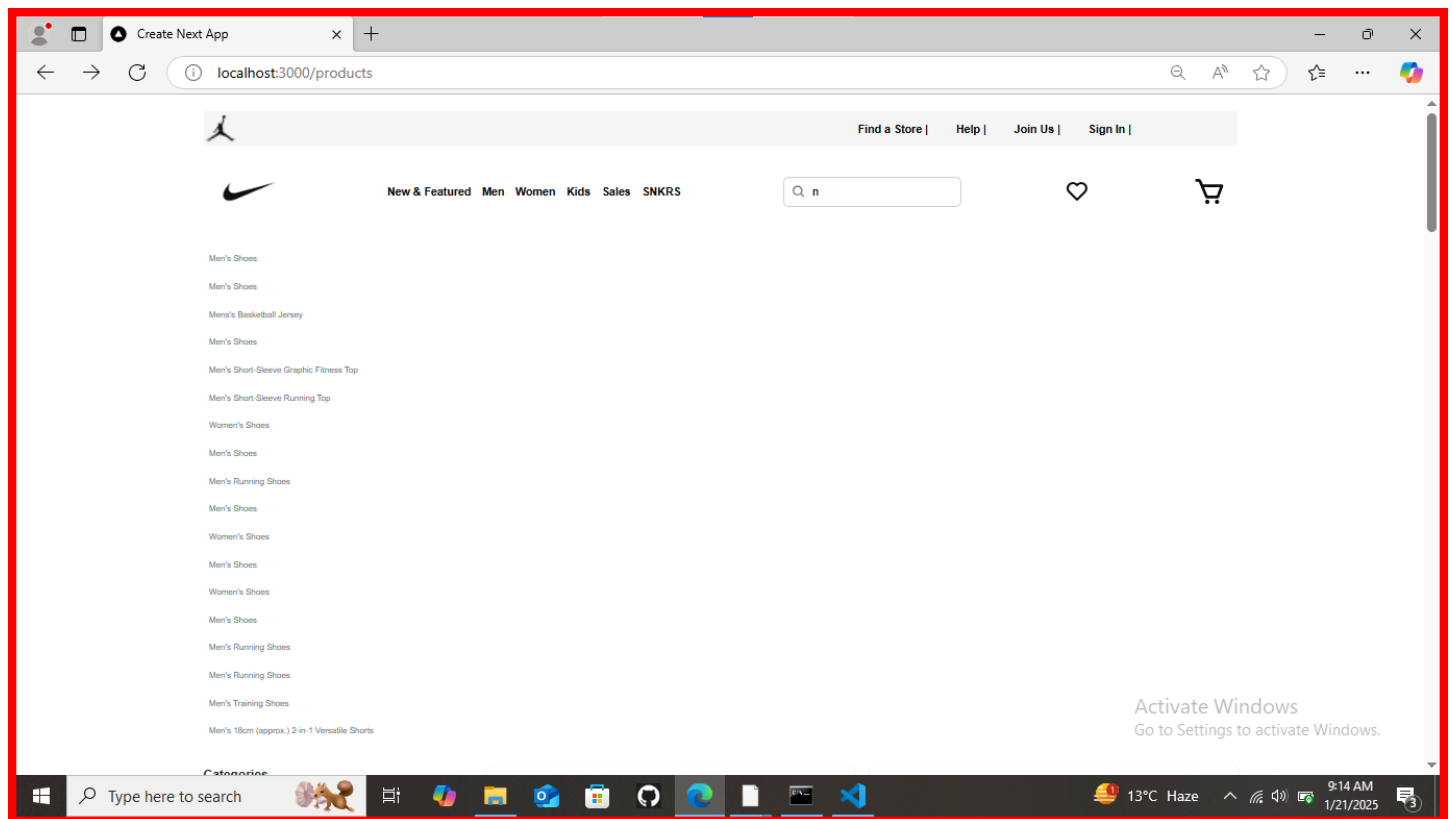
. Adding Search Bar Functionality:

How I can do it.

1-Built a simple search bar component that updates the state and filters the products list based on the entered search query.

2-Incorporated debouncing to optimize performance when searching.

Searching Bar Functionality-



Conclusion:

- I successfully built dynamic frontend components for the Nike e-commerce website, including product listings, product detail pages, category filters, and more. Each component was modular and reusable, ensuring scalability. Data was dynamically fetched from Sanity CMS, providing real-time updates on product availability and details. The design is fully responsive, providing an optimal shopping experience across all devices. With seamless integration of features like the cart, and checkout flow, this platform is ready to serve as a professional e-commerce solution.

