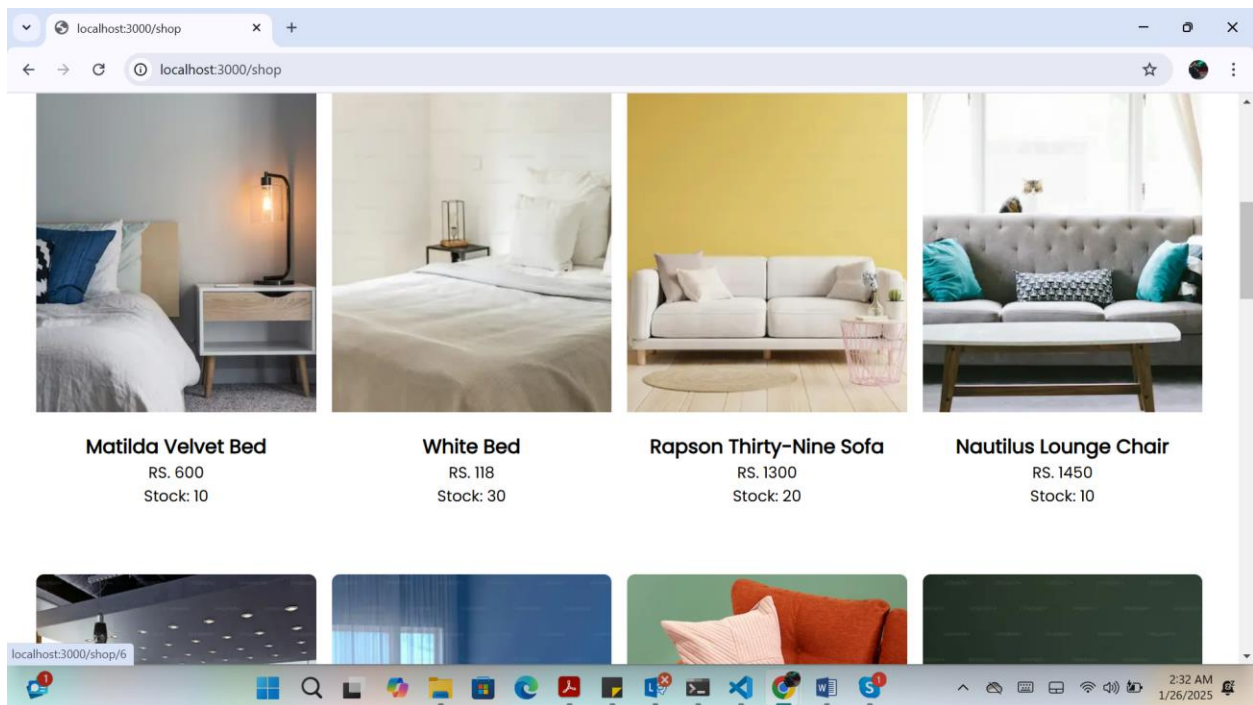


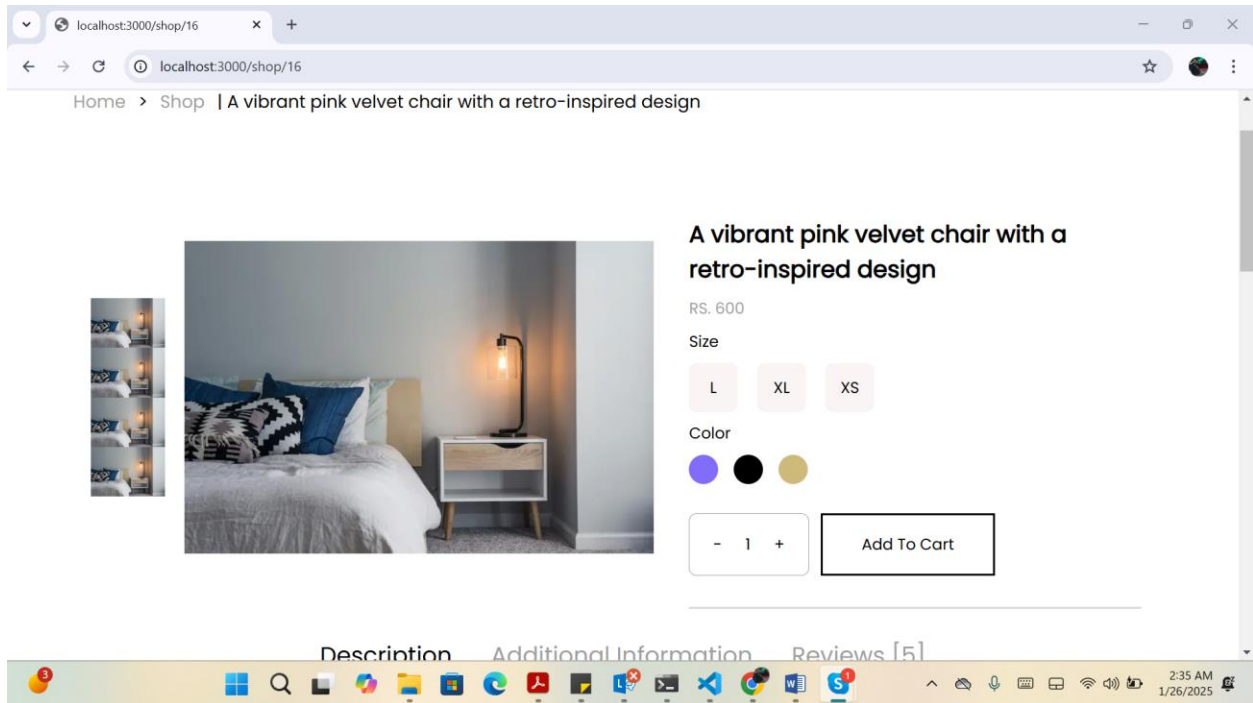
“Day 4 – Dynamic Frontend Components - General E-commerce”

FUNCTIONAL DELIVERABLES:

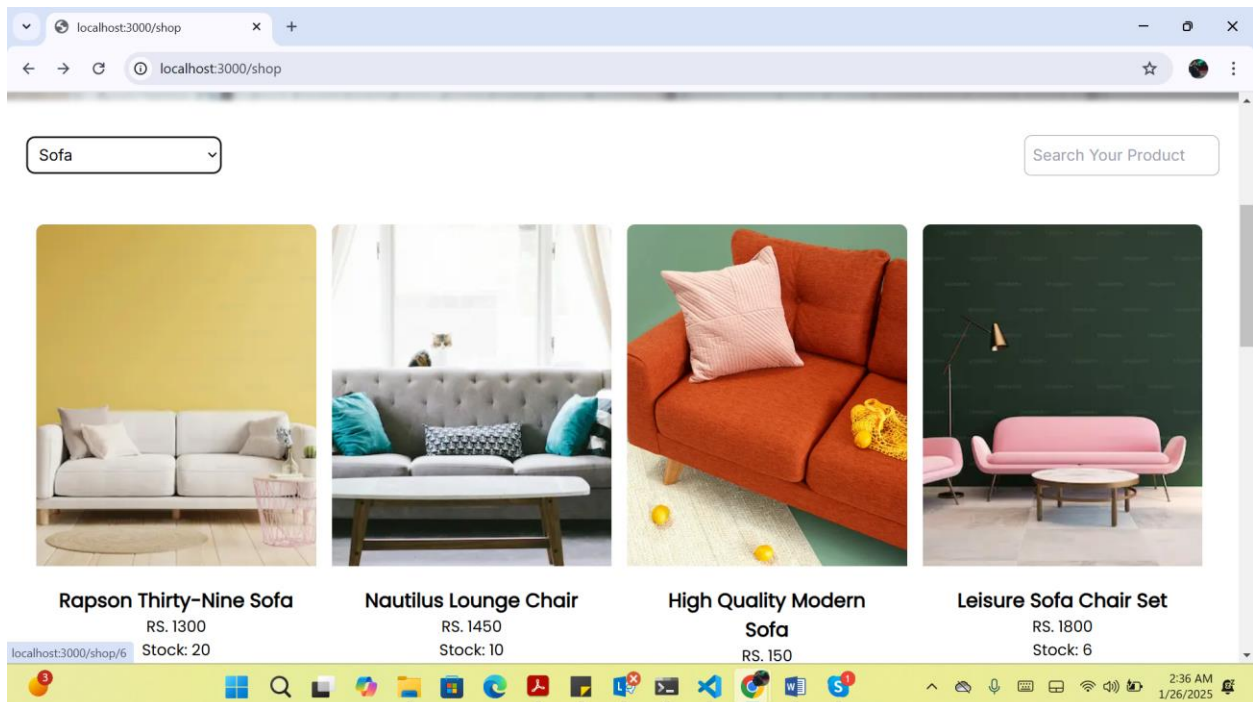
- Screen shots of showcasing:
1.product listing page with dynamic data

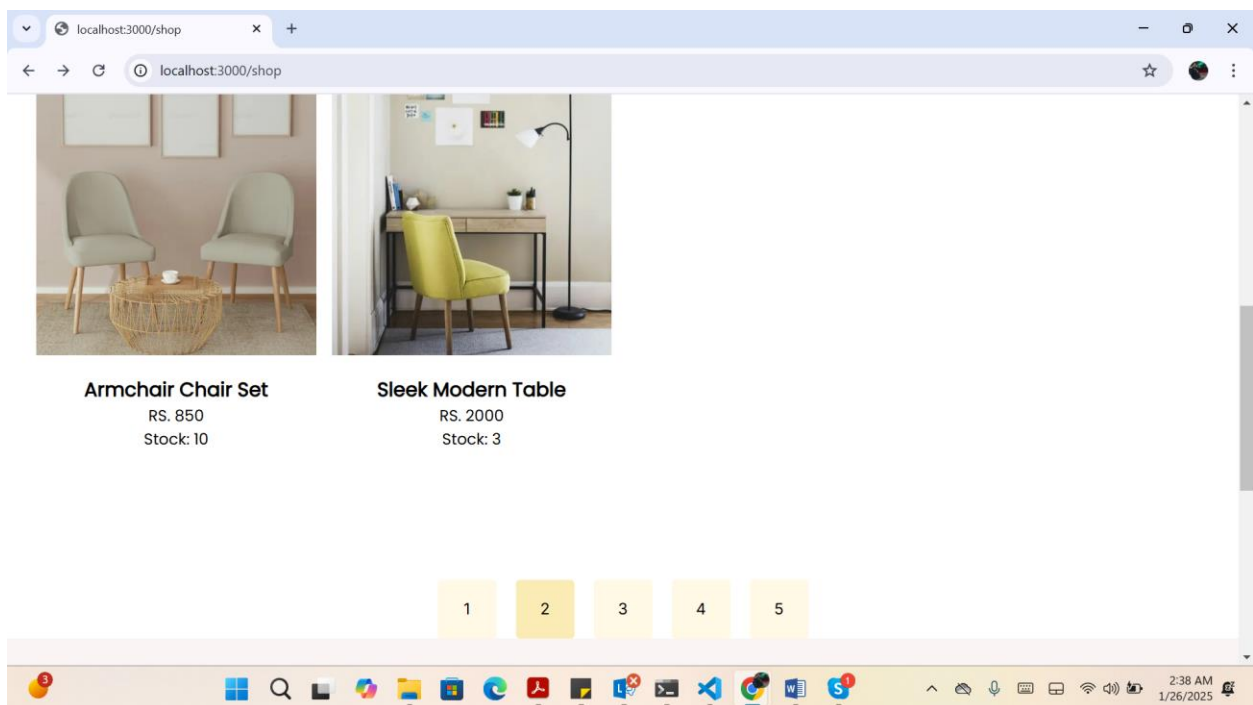
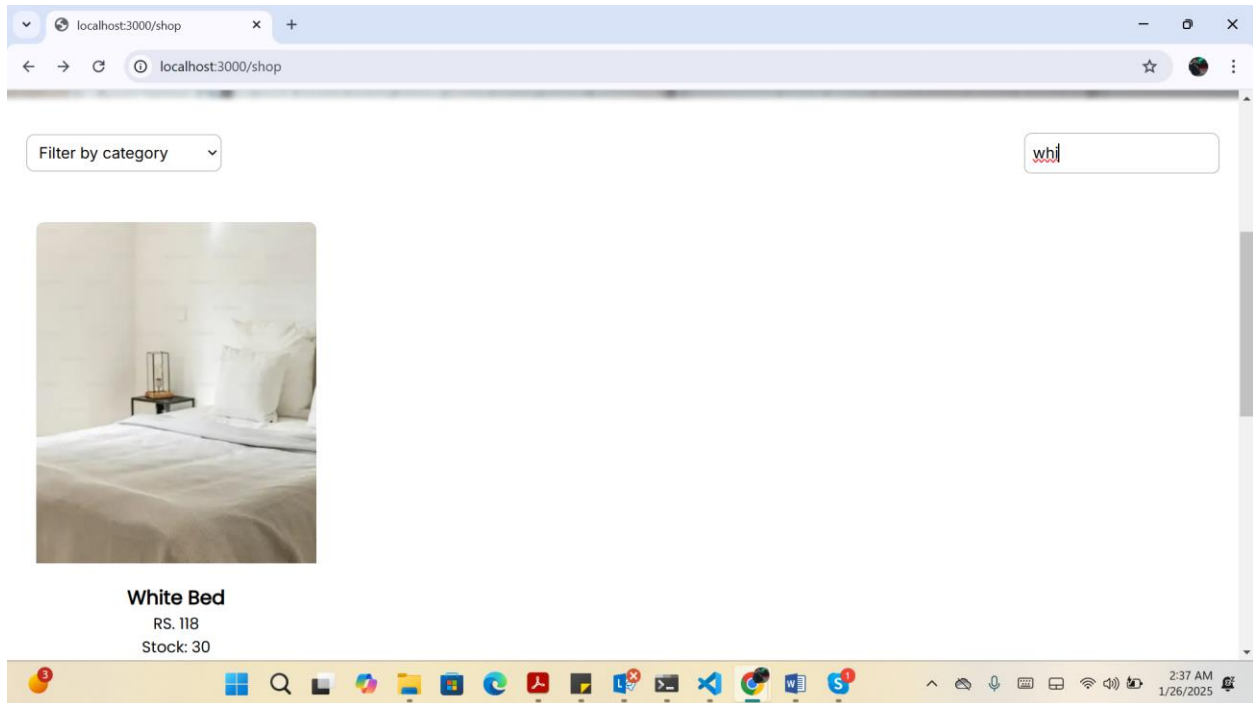


- 2.individual product detail page with accurate routing and dynamic Content.



3. working with search bar, filter

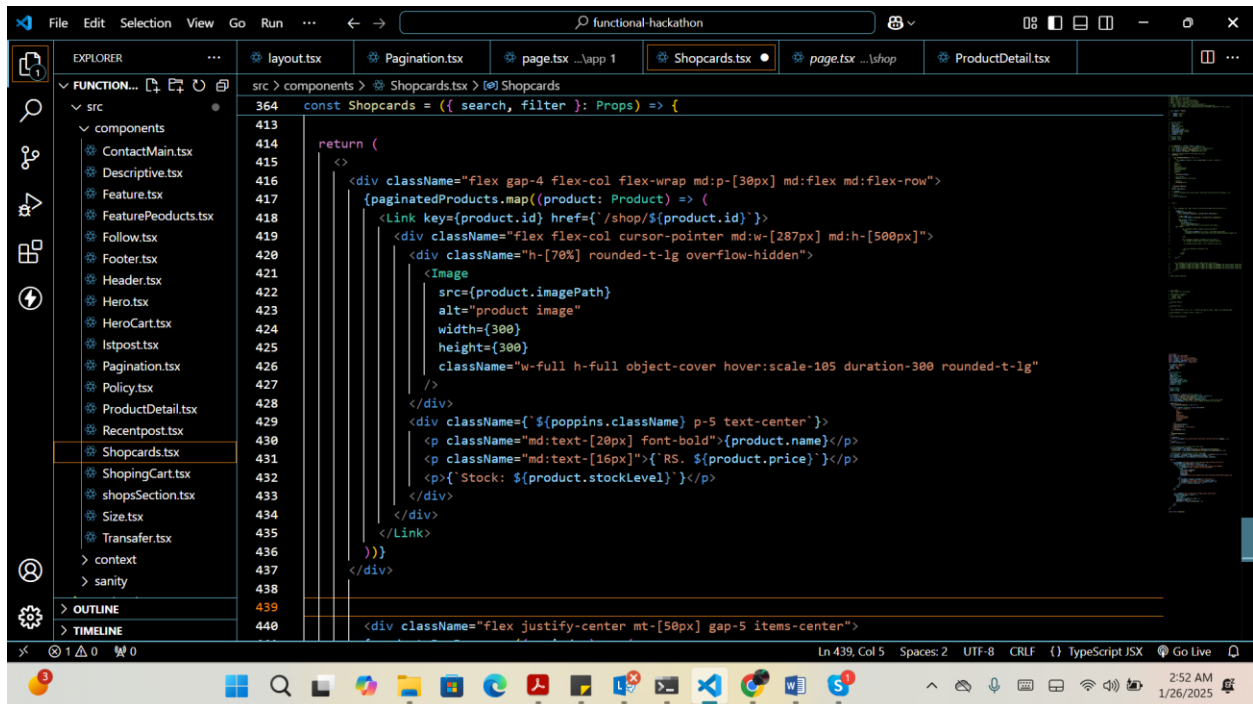




CODE DELIVERABLES:

Code snippet for key components:

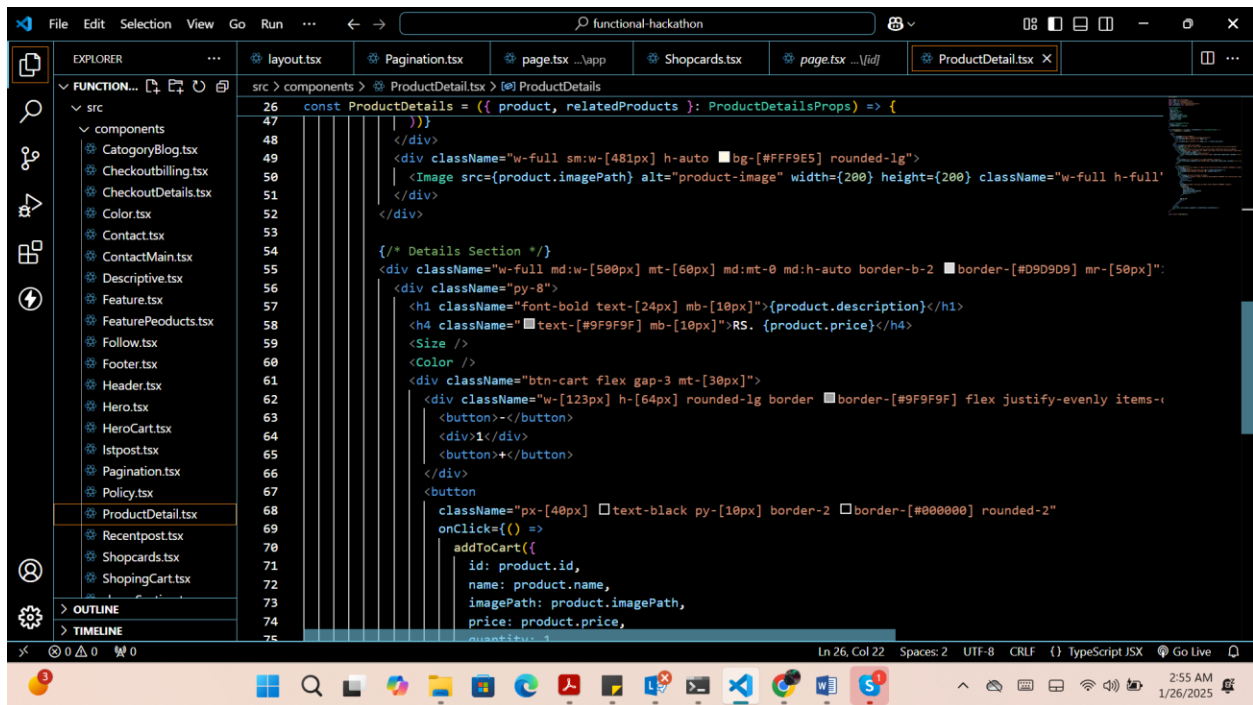
1.product card



The screenshot shows a Visual Studio Code editor window with the file explorer on the left and the code editor in the center. The file explorer shows a project structure with a 'components' folder containing various files, including 'Shopcards.tsx'. The code editor displays the 'Shopcards.tsx' file, which defines a 'Shopcards' component. The component is a function that takes 'search' and 'filter' props and returns a JSX element. The JSX element is a 'div' with a 'flex gap-4 flex-col flex-wrap md:p-[30px] md:flex md:flex-row' class. It contains a 'Link' element with a 'key={product.id}' and 'href={`/shop/\${product.id}`' prop. Inside the 'Link' is a 'div' with a 'flex flex-col cursor-pointer md:w-[287px] md:h-[500px]' class. This 'div' contains an 'Image' element with a 'src={product.imagePath}' and 'alt="product image"' prop. Below the image is a 'div' with a 'poppins.className p-5 text-center' class, containing three 'p' elements: 'product.name', 'RS. {product.price}', and 'Stock: {product.stockLevel}'. The 'Shopcards' component is used in the 'ShopcardSection' component, which is also shown in the file explorer.

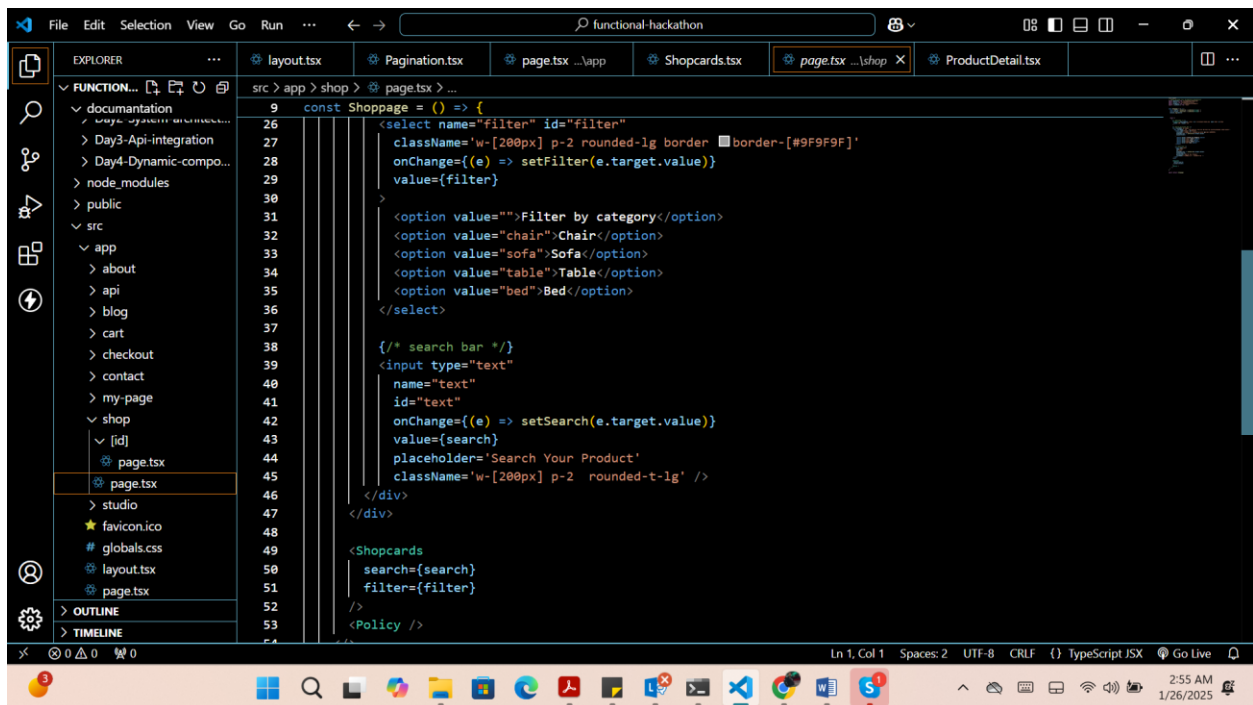
```
364 const Shopcards = ({ search, filter }: Props) => {
413
414   return (
415     <div className="flex gap-4 flex-col flex-wrap md:p-[30px] md:flex md:flex-row">
416       {paginatedProducts.map((product: Product) => (
417         <Link key={product.id} href={`/shop/${product.id}`}>
418           <div className="flex flex-col cursor-pointer md:w-[287px] md:h-[500px]">
419             <div className="h-[70%] rounded-t-lg overflow-hidden">
420               <Image
421                 src={product.imagePath}
422                 alt="product image"
423                 width={300}
424                 height={300}
425                 className="w-full h-full object-cover hover:scale-105 duration-300 rounded-t-lg"
426               />
427             </div>
428             <div className={`${poppins.className} p-5 text-center`}>
429               <p className="md:text-[20px] font-bold">{product.name}</p>
430               <p className="md:text-[16px]">{`RS. ${product.price}`}</p>
431               <p>{`Stock: ${product.stockLevel}`}</p>
432             </div>
433           </Link>
434         </div>
435       ))}
436     </div>
437   )
438 }
439
440 <div className="flex justify-center mt-[50px] gap-5 items-center">
```

2.product list



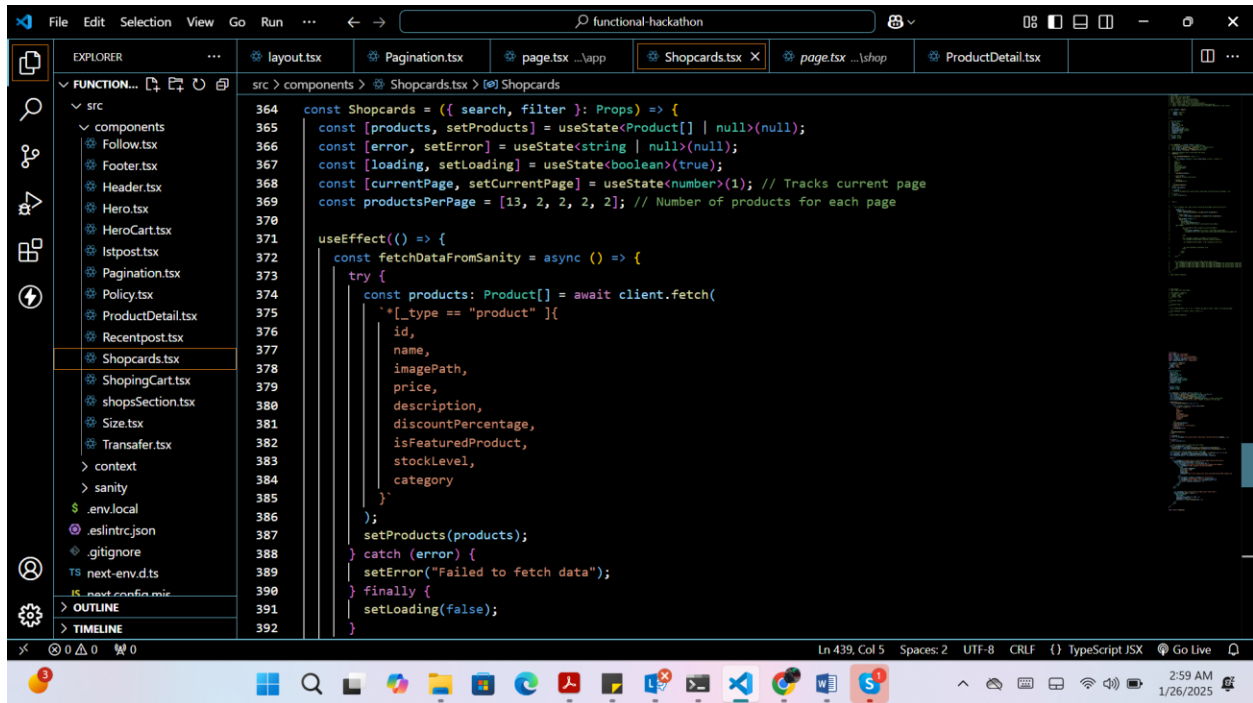
```
26 const ProductDetails = ({ product, relatedProducts }: ProductDetailsProps) => {
47   }
48 }
49 <div className="w-full sm:w-[481px] h-auto bg-[#FFF9E5] rounded-lg">
50   <Image src={product.imagePath} alt="product-image" width={200} height={200} className="w-full h-full" />
51 </div>
52 </div>
53
54 /* Details Section */
55 <div className="w-full md:w-[500px] mt-[60px] md:mt-0 md:h-auto border-b-2 border-[#D9D9D9] mr-[50px]">
56   <div className="py-8">
57     <h1 className="font-bold text-[24px] mb-[10px]">{product.description}</h1>
58     <h4 className="text-[#9F9F9F] mb-[10px]">RS. {product.price}</h4>
59     <Size />
60     <Color />
61     <div className="btn-cart flex gap-3 mt-[30px]">
62       <div className="w-[123px] h-[64px] rounded-lg border border-[#9F9F9F] flex justify-evenly items-center">
63         <button></button>
64         <div>1</div>
65         <button></button>
66       </div>
67       <button
68         className="px-[40px] text-black py-[10px] border-2 border-[#000000] rounded-2"
69         onClick={() =>
70           addToCart({
71             id: product.id,
72             name: product.name,
73             imagePath: product.imagePath,
74             price: product.price,
75             quantity: 1
76           })
77         ></button>
78     </div>
79   </div>
80 </div>
```

3.Search bar



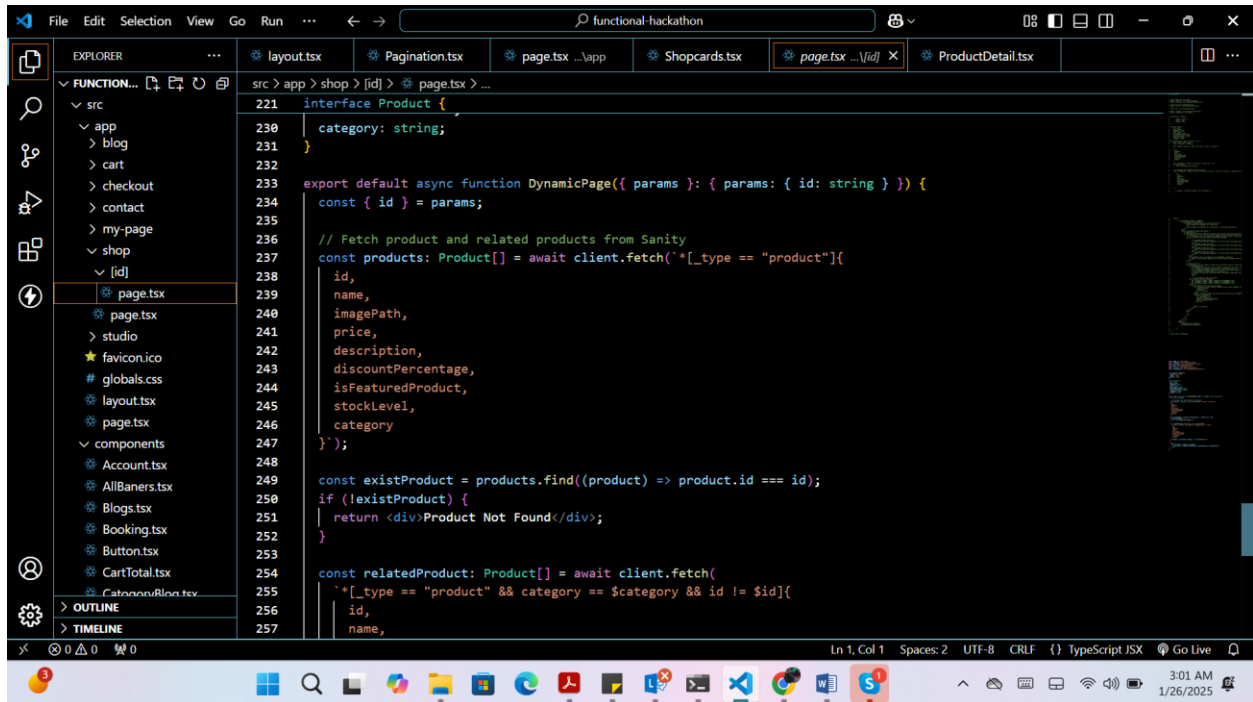
```
9 const Shoppage = () => {
26   <select names="filter" id="filter"
27     className="w-[200px] p-2 rounded-lg border border-[#9F9F9F]"
28     onChange={(e) => setFilter(e.target.value)}
29     value={filter}>
30   </select>
31   <option value="">Filter by category</option>
32   <option value="chair">Chair</option>
33   <option value="sofa">Sofa</option>
34   <option value="table">Table</option>
35   <option value="bed">Bed</option>
36 </select>
37
38 /* search bar */
39 <input type="text"
40   name="text"
41   id="text"
42   onChange={(e) => setSearch(e.target.value)}
43   value={search}
44   placeholder="Search Your Product"
45   className="w-[200px] p-2 rounded-t-lg" />
46 </div>
47 </div>
48
49 <Shopcards
50   search={search}
51   filter={filter}>
52 </Shopcards>
53 </Policy />
```

4. logic for Api integration and dynamic data



This screenshot shows the Visual Studio Code editor with the file `Shopcards.tsx` open. The Explorer sidebar on the left shows a project structure with a `src` directory containing `components` and `sanity` subdirectories. The `Shopcards.tsx` file is selected in the Explorer. The main editor area displays the code for `Shopcards`, which is a functional component that manages state for products, error, loading, and current page. It uses `useState` and `useEffect` hooks. The `useEffect` hook calls `fetchDataFromSanity` to fetch product data from a Sanity database. The code is written in TypeScript and includes comments for tracking the current page and products per page.

```
364 const Shopcards = ({ search, filter }: Props) => {
365   const [products, setProducts] = useState<Product[] | null>(null);
366   const [error, setError] = useState<string | null>(null);
367   const [loading, setLoading] = useState<boolean>(true);
368   const [currentPage, setCurrentPage] = useState<number>(1); // Tracks current page
369   const productsPerPage = [13, 2, 2, 2, 2]; // Number of products for each page
370
371   useEffect(() => {
372     const fetchDataFromSanity = async () => {
373       try {
374         const products: Product[] = await client.fetch(
375           `*[_type == "product"]{
376             id,
377             name,
378             imagePath,
379             price,
380             description,
381             discountPercentage,
382             isFeaturedProduct,
383             stockLevel,
384             category
385           }`
386         );
387         setProducts(products);
388       } catch (error) {
389         setError("Failed to fetch data");
390       } finally {
391         setLoading(false);
392       }
393     };
394   });
395 }
```



This screenshot shows the Visual Studio Code editor with the file `page.tsx` open. The Explorer sidebar on the left shows a project structure with a `src` directory containing `app`, `components`, and `sanity` subdirectories. The `page.tsx` file is selected in the Explorer. The main editor area displays the code for `page.tsx`, which is a functional component that manages state for products, error, loading, and current page. It uses `useState` and `useEffect` hooks. The `useEffect` hook calls `fetchDataFromSanity` to fetch product data from a Sanity database. The code is written in TypeScript and includes comments for tracking the current page and products per page.

```
221 interface Product {
222   category: string;
223 }
224
225 export default async function DynamicPage({ params }: { params: { id: string } }) {
226   const { id } = params;
227
228   // Fetch product and related products from Sanity
229   const products: Product[] = await client.fetch(`*[_type == "product"]{
230     id,
231     name,
232     imagePath,
233     price,
234     description,
235     discountPercentage,
236     isFeaturedProduct,
237     stockLevel,
238     category
239   }`);
240
241   const existProduct = products.find((product) => product.id === id);
242   if (!existProduct) {
243     return <div>Product Not Found</div>;
244   }
245
246   const relatedProduct: Product[] = await client.fetch(
247     `*[_type == "product" && category == $category && id != $id]{
248       id,
249       name,
250       category
251     }`
252   );
253 }
```