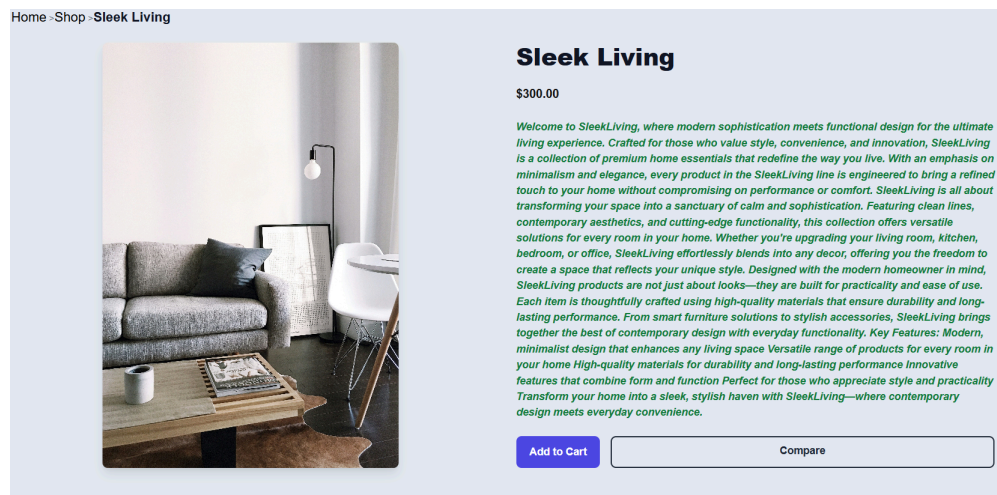


## Day 4 - Dynamic Frontend Components General E-commerce Website

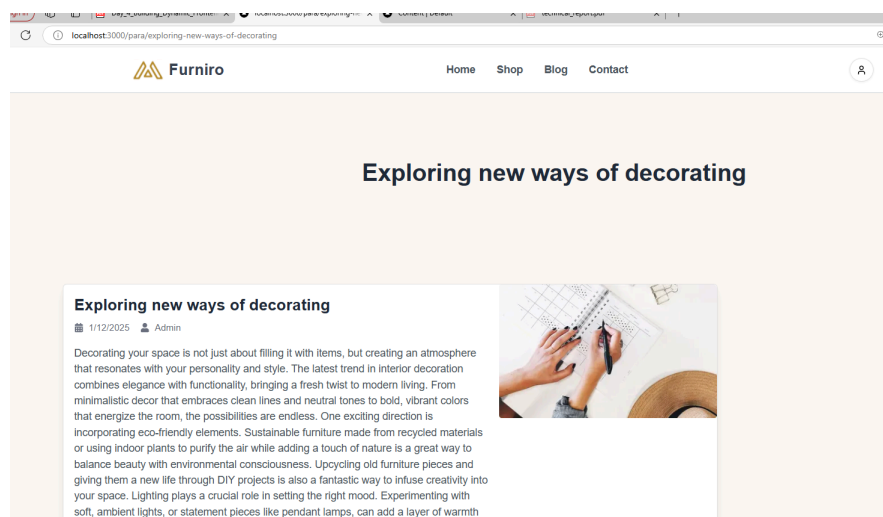
Prepared By: Soban Saud

### 1) FUNCTIONAL DELIVERY

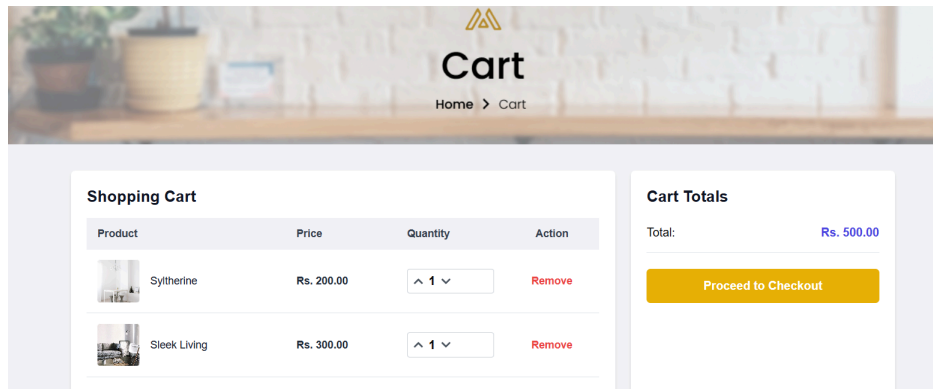
#### a) Dynamic Product page



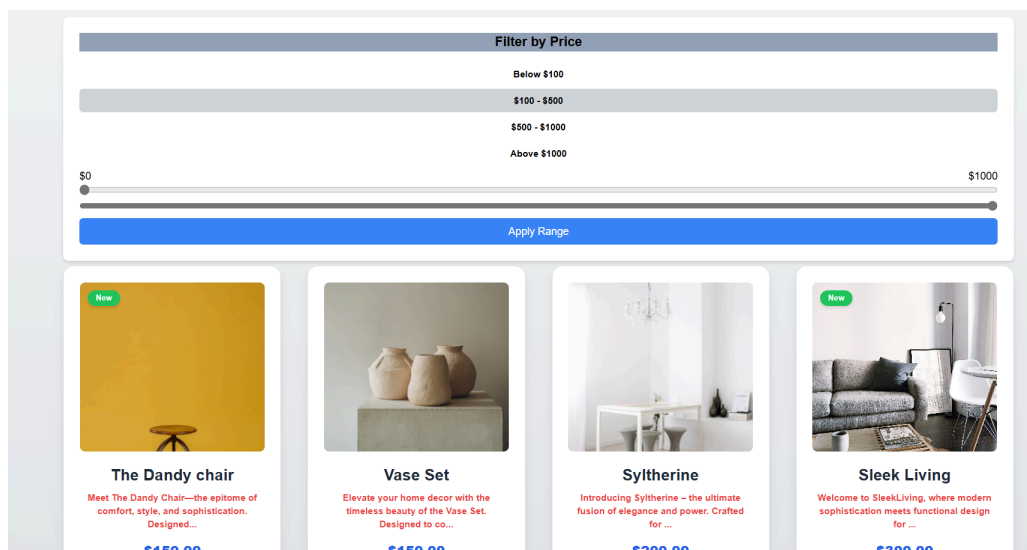
#### b) Dynamic Blog Page



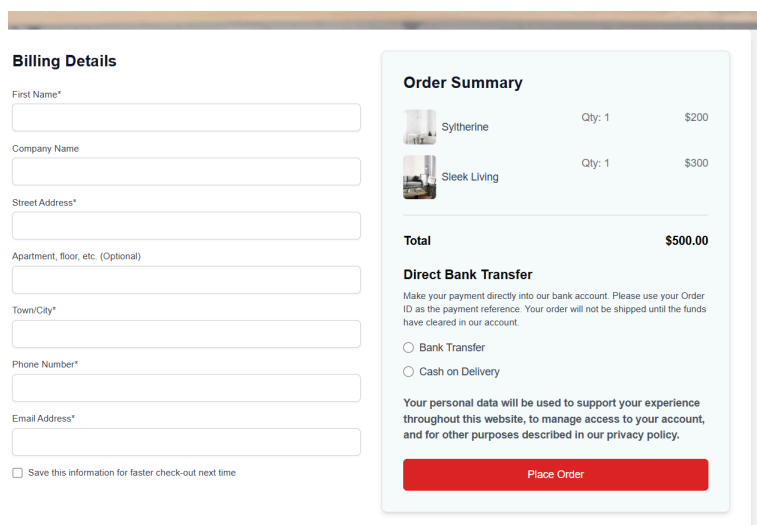
#### c) Cart Page With Functionality



#### d) Price Filter And pagination



#### e) Checkout Page With Functionality



## 2) Code Deliverables

- Code snippet for components

### a) Shop page

This pages show frontend display with the help of query .

```
app > shop > @ pagetSX > ...
792 import ShopClient from '@components/ShopClient';
793
794 interface Product {
795   _id: string;
796   title: string;
797   description: string;
798   price: number;
799   discountPercentage?: number;
800   isNew?: boolean;
801   tags: string[];
802   imageUrl: string;
803 }
804
805 // Server-side data fetching (Server Component)
806 const ShopPage = async () => {
807   const query = `
808     {[_type == "product"]{
809       _id,
810       title,
811       description,
812       price,
813       discountPercentage,
814       isNew,
815       tags,
816       "imageUrl": productImage.asset->url
817     }
818   `;
819   const products = await client.fetch(query);
820
821   const productsPerPage = 8;
822   const totalPages = Math.ceil(products.length / productsPerPage);
```

### b) ProductCard

This component displays individual product details and includes the "Add to Cart" button.

```
components > @ ProductCard.tsx > ...
9 interface Product {
10   _id: string;
11 }
12
13 const ProductCard = ({ product }: { product: Product }) => {
14   const { addToWishlist } = useWishlist(); // Get the addToWishlist function from context
15   const { addToComparison } = useComparison(); // Get the addToComparison function from context
16
17   return (
18     <Link href={`/product/${product._id}`} >
19       <div className="bg-white p-6 rounded-2xl shadow-lg hover:shadow-xl transition duration-500 transform hover:s
20         <div className="relative" >
21           <img
22             src={product.imageUrl}
23             alt={product.title}
24             className="w-full h-64 object-cover rounded-lg transition duration-300 hover:opacity-90"
25           />
26           {product.isNew && (
27             <span className="absolute top-3 left-3 bg-green-500 text-white px-3 py-1 text-xs font-semibold round
28               New
29             </span>
30           )}
31           {product.discountPercentage && (
32             <span className="absolute top-3 right-3 bg-red-500 text-white px-3 py-1 text-xs font-semibold rounde
33               {product.discountPercentage}%
34             </span>
35           )}
36         </div>
37         <div className="mt-5 text-center" >
38           <h2 className="text-2xl font-bold text-gray-800 hover:text-indigo-600 transition" >
39             {product.title}
40           </h2>
41         </div>
42       </Link>
43     )
44   );
45 }
```

### c) Product List

Fetches and Display a list of products

```
components > ProductList.tsx > ProductList
3 import React from "react";
4 import ProductCard from "../ProductCard";
5
6 interface Product {
7   _id: string;
8   title: string;
9   description: string;
10  price: number;
11  discountPercentage?: number;
12  isNew?: boolean;
13  tags: string[];
14  imageUrl: string;
15 }
16
17 const ProductList = ({ products, productsPerPage, totalPages }: { products: Product[],
18   return []
19   <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-6">
20     {products.length > 0 ? (
21       products.map((product) => (
22         <ProductCard key={product._id} product={product} />
23       ))
24     ) : (
25       <p>No products found in this price range.</p>
26     )}
27   </div>
28 }
```

### d) Add to Cart

This components allows user to add products to the cart from the shop page

```
components > AddToCartButton.tsx > ...
55
56
57 "use client";
58
59 import { useState } from "react";
60 import { useCart } from "@components/CartContext";
61 import Link from "next/link";
62
63 const AddToCartButton = ({ product }: { product: any }) => {
64   const { addToCart } = useCart();
65   const [showNotification, setShowNotification] = useState(false);
66
67   const handleAddToCart = () => {
68     addToCart({
69       id: product._id,
70       name: product.title,
71       price: product.price.toFixed(2),
72       image: product.imageUrl,
73       quantity: 1,
74     });
75
76     setShowNotification(true);
77     setTimeout(() => setShowNotification(false), 3000); // Hide after 3 seconds
78   };
79
80   return (
81     <div className="relative">
82       <button
83         </button>
84     </div>
85   );
86 }
```

### e) Pagination

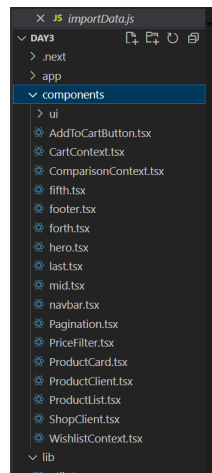
The Pagination component is responsible for displaying paginated products on the shop page. It ensures that only a limited number of products are shown per page while providing navigation controls to switch between pages.

```

components > Pagination.tsx > @ Pagination
1
2 'use client'
3
4 import { useState } from 'react';
5 import ProductCard from './ProductCard';
6
7 const Pagination = ({ products, totalPages }: { products: any[]; totalPages: number }) => {
8   const [currentPage, setCurrentPage] = useState(1);
9   const productsPerPage = 8;
10
11   // Calculate the products to display based on the current page
12   const startIndex = (currentPage - 1) * productsPerPage;
13   const endIndex = startIndex + productsPerPage;
14   const currentProducts = products.slice(startIndex, endIndex);
15
16   return (
17     <div>
18       <!-- Display products for the current page -->
19       <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-10">
20         {currentProducts.map((product) => (
21           <ProductCard key={product._id} product={product} />
22         ))}
23       </div>
24
25       <!-- Pagination Controls -->
26       <div className="flex justify-center items-center gap-4 mt-10">
27         <button
28           className="bg-indigo-600 text-white px-4 py-2 rounded-md font-bold hover:bg-indigo-700 transition"
29           onClick={() => setCurrentPage((prev) => Math.max(prev - 1, 1))}
30           disabled={currentPage === 1}

```

## f) Other components



AddToCartButton.tsx – Handles adding products to the cart using CartContext.

CartContext.tsx – Manages global cart state, including add/remove functions

ComparisonContext.tsx – Stores and manages product comparison data.

fifth.tsx – Likely a section or UI component used in the layout.

footer.tsx – Displays the website footer with relevant links and information

forth.tsx – Another layout or section component used in the UI.

hero.tsx – The main banner or hero section for the homepage.

last.tsx – Possibly the final section of a webpage layout.

mid.tsx – Likely represents a middle section in a webpage

navbar.tsx – The navigation bar containing links and the cart icon.

Pagination.tsx – Implements pagination for product listings.

PriceFilter.tsx – Allows users to filter products based on price.

ProductCard.tsx – Displays individual product details in a card format.

ProductClient.tsx – Manages product-related API calls or client-side logic.


ProductList.tsx – Fetches and displays a list of products dynamically.

ShopClient.tsx – Handles the shop page functionality and client-side logic.

WishlistContext.tsx – Manages the wishlist state for storing favorite products.

- Scripts and Logic for API Integration and Dynamic Routing
- API Integration

The project integrates APIs to fetch and manage product data dynamically. The API endpoint used for product data retrieval is:

 <https://template6-six.vercel.app/api/products>

Key steps for API integration:

Fetching Data – Products are fetched using fetch or axios inside ProductList.tsx or ShopClient.tsx.

State Management – The data is stored in state using useState and updated using useEffect.

Error Handling – Try-catch blocks ensure smooth error handling during API requests

Filtering & Searching – Filters like PriceFilter.tsx refine the displayed products.

```

$ importData.js > ...
35  async function uploadProduct(product) {
36
37      const createdProduct = await client.create(document);
38      console.log('✅ Product ${product.title} uploaded successfully:', createdProduct);
39  } else {
40      console.log('❌ Product ${product.title} skipped due to image upload failure.');
```

- Dynamic Routing

The project utilizes Next.js dynamic routes to handle product and blog pages:

Product Page: `[id]/page.tsx` dynamically fetches and displays a product based on its ID.

```

app > product > [id] > page.tsx > ...
762  interface Product {
763      // @
764      imageUrl: string;
765      colors?: string[];
766      sizes?: string[];
767  }
768  }
769
770  const fetchProduct = async (id: string): Promise<Product | null> => {
771      const query = `[_type == "product" && _id == ${id}[0]{
772          _id,
773          title,
774          description,
775          price,
776          discountPercentage,
777          isNew,
778          tags,
779          "imageUrl": productImage.asset->url,
780          colors,
781          sizes
782      }`;
783      const product = await client.fetch(query, { id });
784      return product || null;
785  };
786
787  const ProductPage = async ({ params }: { params: { id: string } }) => {
788      const product = await fetchProduct(params.id);
789
790      if (!product) {
791          notFound();
792      }
793
794      return (
795          <div className="container mx-auto px-6 py-12 lg:col-span-2">
```

Blog Page: `[slug]/page.tsx` dynamically loads blog content based on the slug.

```

app > para > [slug] > * BlogPost > title
9   const query = `[_type == "blog" && slug.current == ${slug}[0] {
12     date,
13     author,
14     content,
15     "imageUrl": image.asset->url
16   }`;
17
18   interface BlogPost {
19     title: string;
20     slug: { current: string };
21     date: string;
22     author: string;
23     content: string;
24     imageUrl: string;
25   }
26
27   interface Params {
28     params: {
29       slug: string;
30     };
31   }
32
33   // Static params generation (for dynamic route)
34   export async function generateStaticParams() {
35     const posts = await client.fetch(`*[_type == "blog"]{slug}`);
36     return posts.map((post: { slug: { current: string } }) => ({
37       slug: post.slug.current,
38     }));
39   }
40
41   // Main page for dynamic blog post
42   const BlogPostPage = async ({ params }: Params) => {

```

- Technical Report

- Steps Taken to Build and Integrate Components

Project Setup – Initialized the Next.js project and configured the necessary dependencies.

Component Structure – Created reusable components such as ProductCard.tsx, Pagination.tsx, Navbar.tsx, and Footer.tsx

State Management – Used useState and useContext (e.g., CartContext.tsx,

WishlistContext.tsx) for cart and wishlist functionality.

API Integration – Fetched product data from <https://template6-six.vercel.app/api/products> and displayed it dynamically.

Dynamic Routing – Implemented Next.js dynamic routes for product ([id]/page.tsx) and blog ([slug]/page.tsx) pages.

Filtering & Pagination – Integrated PriceFilter.tsx for filtering products and Pagination.tsx for browsing through multiple pages.

UI Enhancements – Improved the UI with Tailwind CSS and ensured responsiveness across devices.

Testing & Optimization – Debugged API calls, optimized rendering, and handled errors for a smoother user experience.



- Challenges Faced and Solutions Implemented

Issue1): API data was not displaying properly on the Product List page.

Solution: Implemented `useEffect` to fetch data on mount and added error handling.

Issue2): Cart and Wishlist functionality needed to persist across components.

Solution: Used React Context API (`CartContext.tsx` and `WishlistContext.tsx`) for global state management.

Issue3): Pagination logic wasn't updating correctly.

Solution: Fixed state updates using `useState` and ensured `currentPage` updates were handled properly.

Issue4): Dynamic routes weren't working as expected.

Solution: Used Next.js `useRouter` to extract parameters and fetch data accordingly.

- Best Practices Followed During Development

✓ Component Reusability – Developed modular components (`ProductCard.tsx`, `Navbar.tsx`, `Footer.tsx`) to maintain clean code.

✓ State Management – Used Context API for cart and wishlist to avoid prop drilling.

✓ Optimized API Calls – Implemented efficient data fetching with error handling to improve performance.

✓ Code Maintainability – Followed proper file structure (`components/ui/`, `lib/`, etc.) to keep the codebase organized.

✓ Performance Optimization – Used lazy loading and memoization techniques to improve load times.