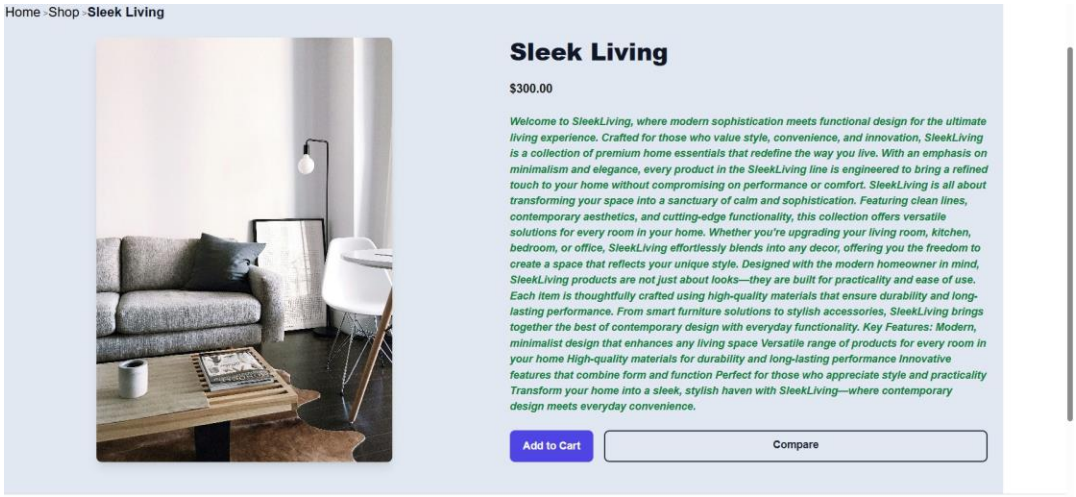
Day 4 - Dynamic Frontend Components General E-commerce Website

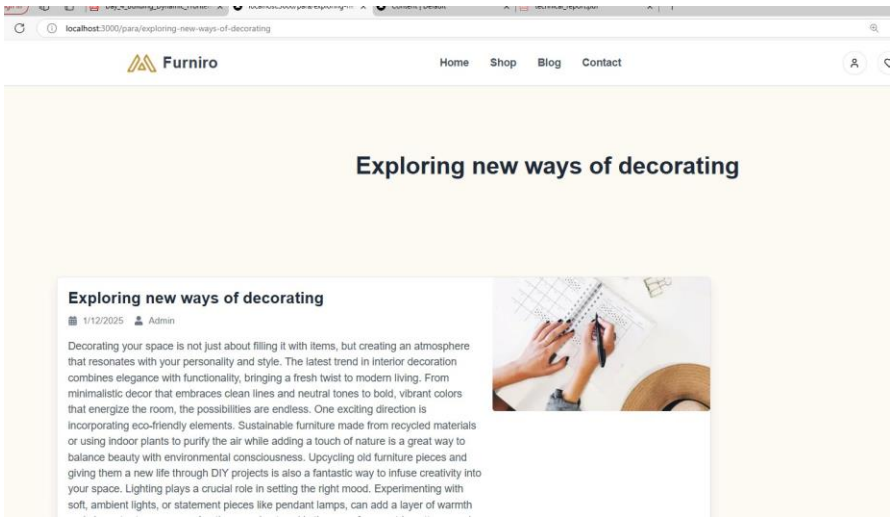
Prepared By: Fatima Mudassir

1) FUNCTIONAL DELIVERY

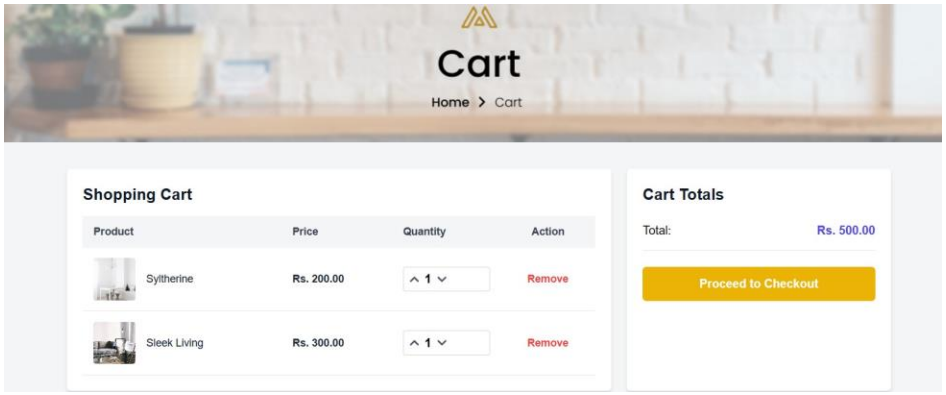
a) Dynamic Product page



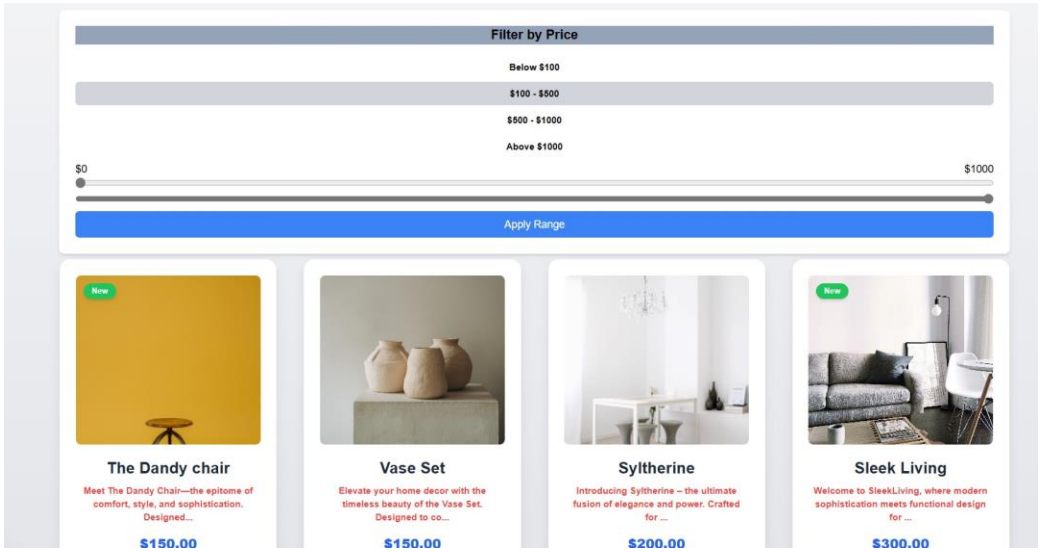
b) Dynamic Blog Page



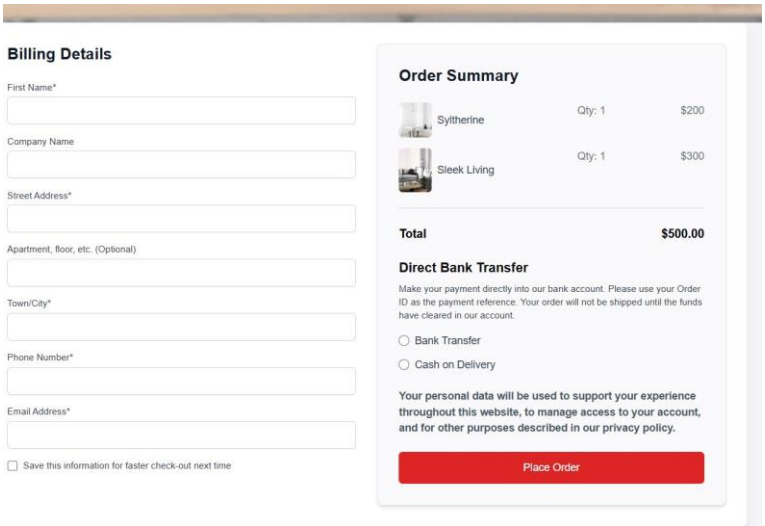
c) Cart Page With Functionality



d) Price Filter And pagination



e) Checkout Page With Functionality

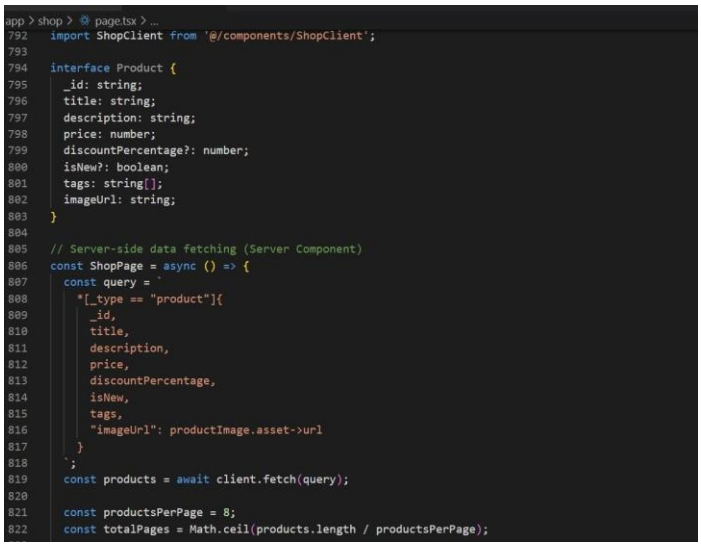


2) Code Deliverables

- Code snippet for components

    a) Shop page

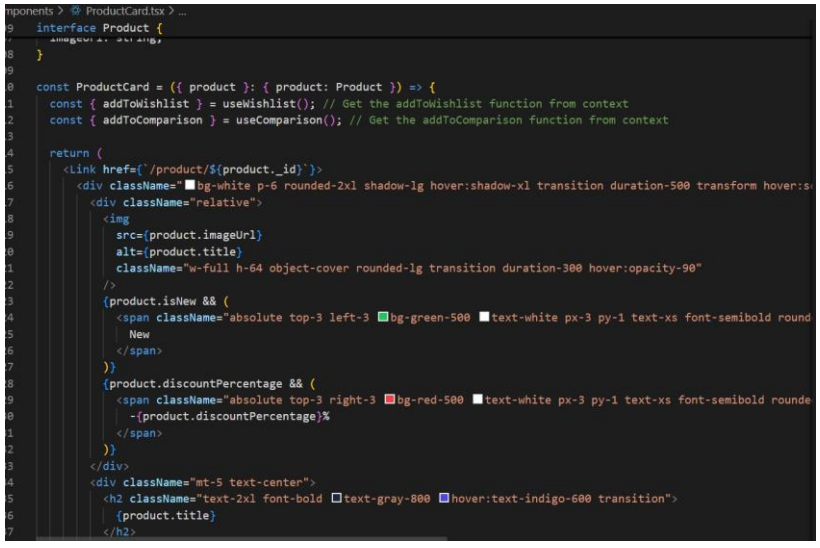
This pages show frontend display with the help of query .



```
app > shop > page.tsx > ...
import ShopClient from '@/components/ShopClient';

interface Product {
  _id: string;
  title: string;
  description: string;
  price: number;
  discountPercentage?: number;
  isNew?: boolean;
  tags: string[];
  imageUrl: string;
}

// Server-side data fetching (Server Component)
const ShopPage = async () => {
  const query = `
    *[_type == "product"]{
      _id,
      title,
      description,
      price,
      discountPercentage,
      isNew,
      tags,
      "imageUrl": productImage.asset->url
    }
  `;
  const products = await client.fetch(query);

  const productsPerPage = 8;
  const totalPages = Math.ceil(products.length / productsPerPage);
```

    b) ProductCard

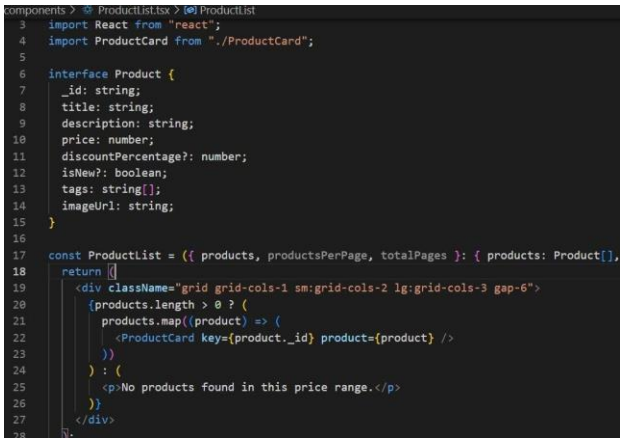
This component displays individual product details and includes the "Add to Cart" button.



```
nponents > ProductCard.tsx > ...
  interface Product {
}

const ProductCard = ({ product }: { product: Product }) => {
  const { addToWishlist } = useWishlist(); // Get the addToWishlist function from context
  const { addToComparison } = useComparison(); // Get the addToComparison function from context

  return (
    <Link href={`/product/${product._id}`}>
      <div className="bg-white p-6 rounded-2xl shadow-lg hover:shadow-xl transition duration-500 transform hover:s
        <div className="relative">
          <img
            src={product.imageUrl}
            alt={product.title}
            className="w-full h-64 object-cover rounded-lg transition duration-300 hover:opacity-90"
          />
          {product.isNew && (
            <span className="absolute top-3 left-3 bg-green-500 text-white px-3 py-1 text-xs font-semibold round
              New
            </span>
          )}
          {product.discountPercentage && (
            <span className="absolute top-3 right-3 bg-red-500 text-white px-3 py-1 text-xs font-semibold rounde
              -{product.discountPercentage}%
            </span>
          )}
        </div>
        <div className="mt-5 text-center">
          <h2 className="text-2xl font-bold text-gray-800 hover:text-indigo-600 transition">
            {product.title}
          </h2>
```

    c) Product List

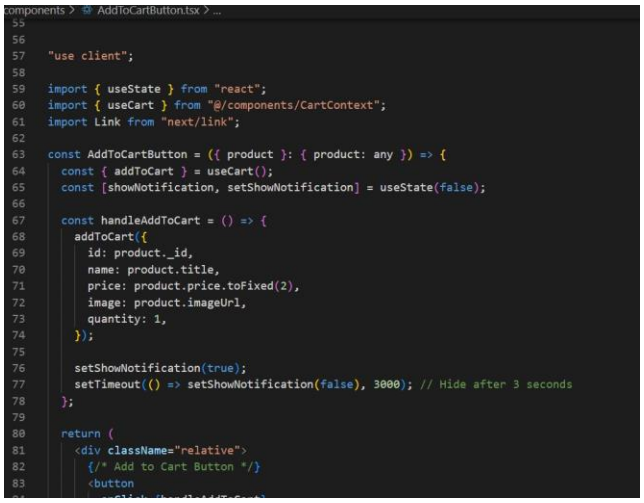
Fetches and Display a list of products



```
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>
```

d) Add to Cart

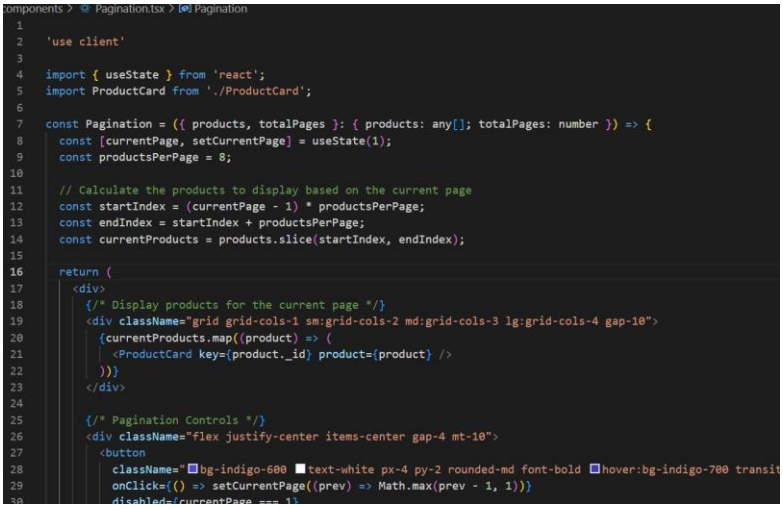
This components allows user to add products to the cart from the shop page



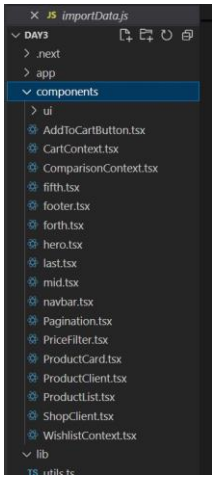
```
components > AddToCartButton.tsx > ...
55
56
57  "use client";
58
59  import { useState } from "react";
60  import { useCart } from "@/components/CartContext";
61  import Link from "next/link";
62
63  const AddToCartButton = ({ product }: { product: any }) => {
64    const { addToCart } = useCart();
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        {/* Add to Cart Button */}
83        <button
```

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.



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) {
57      const createdProduct = await client.create(document);
58      console.log(`✅ Product ${product.title} uploaded successfully:`, createdProduct);
59    } else {
60      console.log(`❌ Product ${product.title} skipped due to image upload failure.`);
61    }
62  } catch (error) {
63    console.error('Error uploading product:', error);
64  }
65  }
66
67  async function importProducts() {
68    try {
69      const response = await fetch('https://template6-six.vercel.app/api/products');
70
71      if (!response.ok) {
72        throw new Error(`HTTP error! Status: ${response.status}`);
73      }
74
75      const products = await response.json();
76      console.log('Fetched Products:', products); // Debugging
77
78      for (const product of products) {
79        await uploadProduct(product);
80      }
81    } catch (error) {
82      console.error('Error fetching products:', error);
83    }
84  }
```

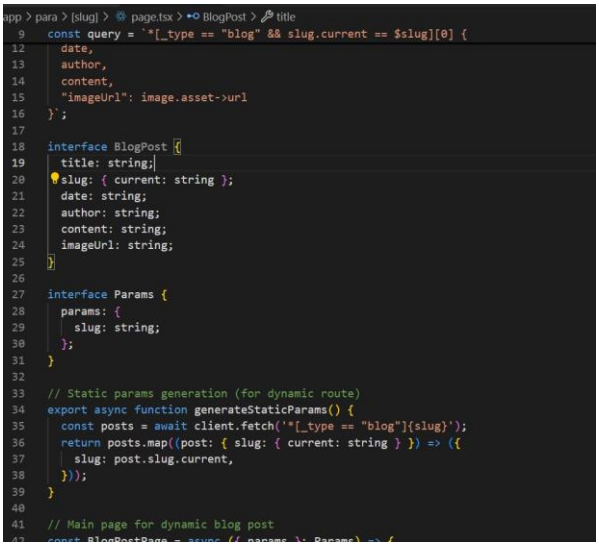
- Dynamic Routing

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 {
771    imageUrl: string;
772    colors?: string[];
773    sizes?: string[];
774  }
775
776  const fetchProduct = async (id: string): Promise<Product | null> => {
777    const query = `*[_type == "product" && _id == $id][0]{
778      _id,
779      title,
780      description,
781      price,
782      discountPercentage,
783      isNew,
784      tags,
785      "imageUrl": productImage.asset->url,
786      colors,
787      sizes
788    }`;
789    const product = await client.fetch(query, { id });
790    return product || null;
791  };
792
793  const ProductPage = async ({ params }: { params: { id: string } }) => {
794    const product = await fetchProduct(params.id);
795
796    if (!product) {
797      notFound();
798    }
799
800    return (
```

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



- 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.