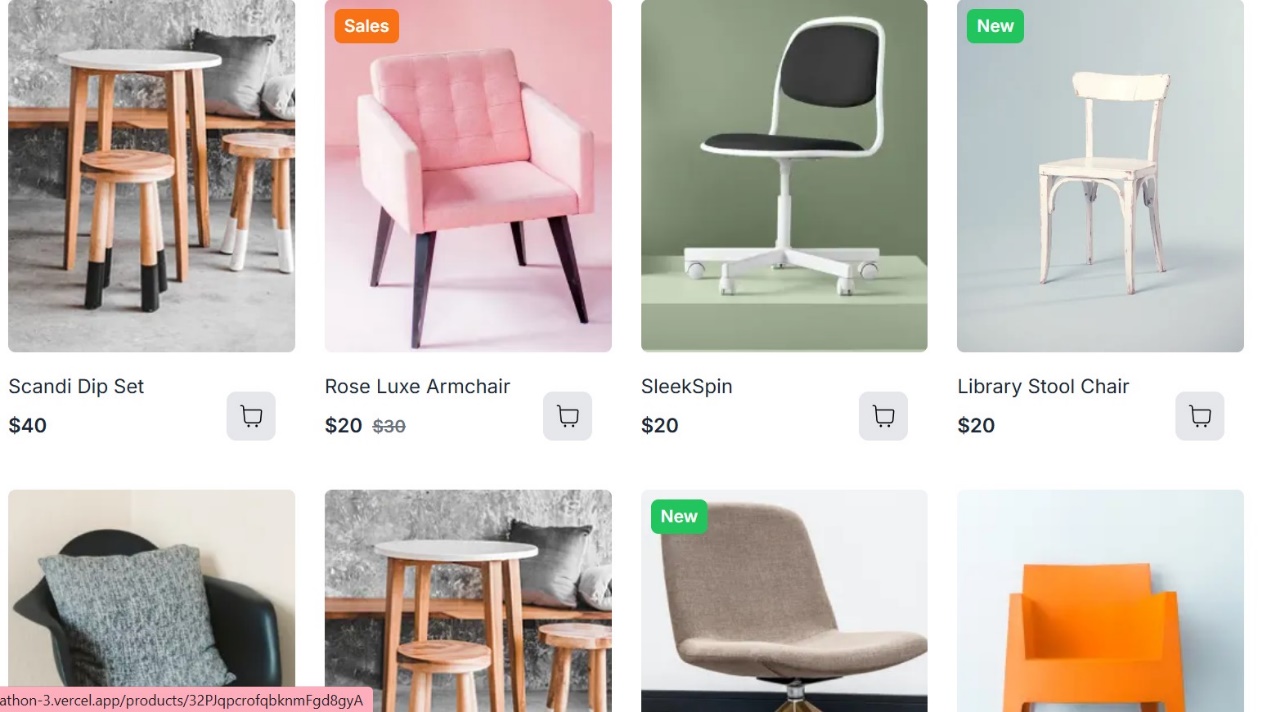
# Day 4 - Building Dynamic Frontend Components for Your Marketplace

**Document Title:** "Day 4 - Dynamic Frontend Components “Comforty”

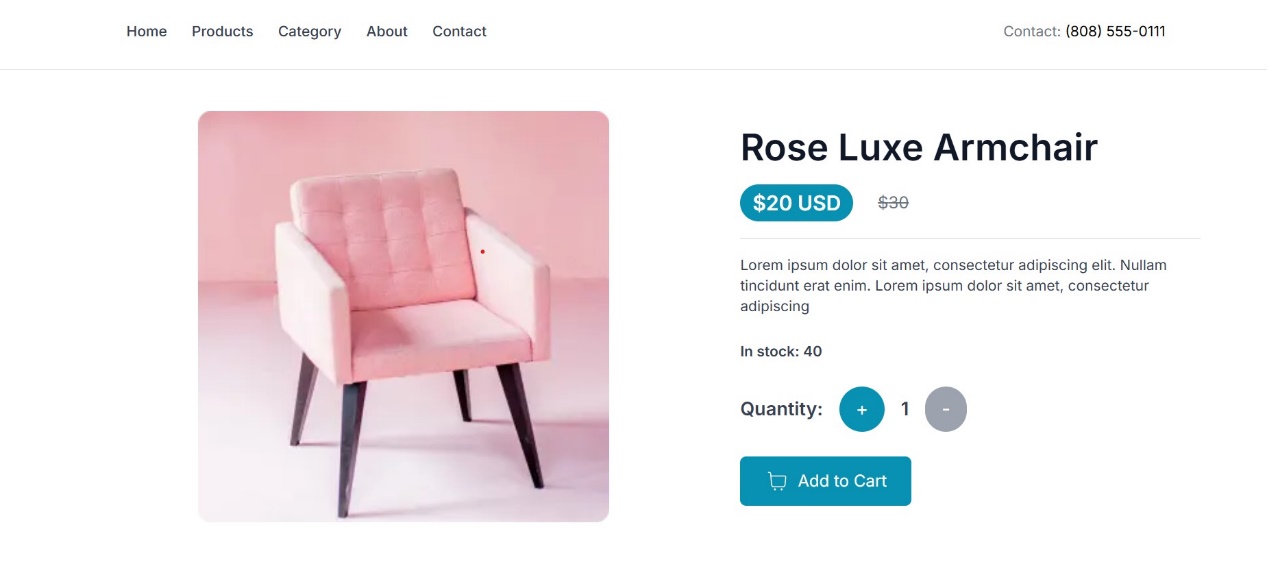
## 1. Functional Deliverables:

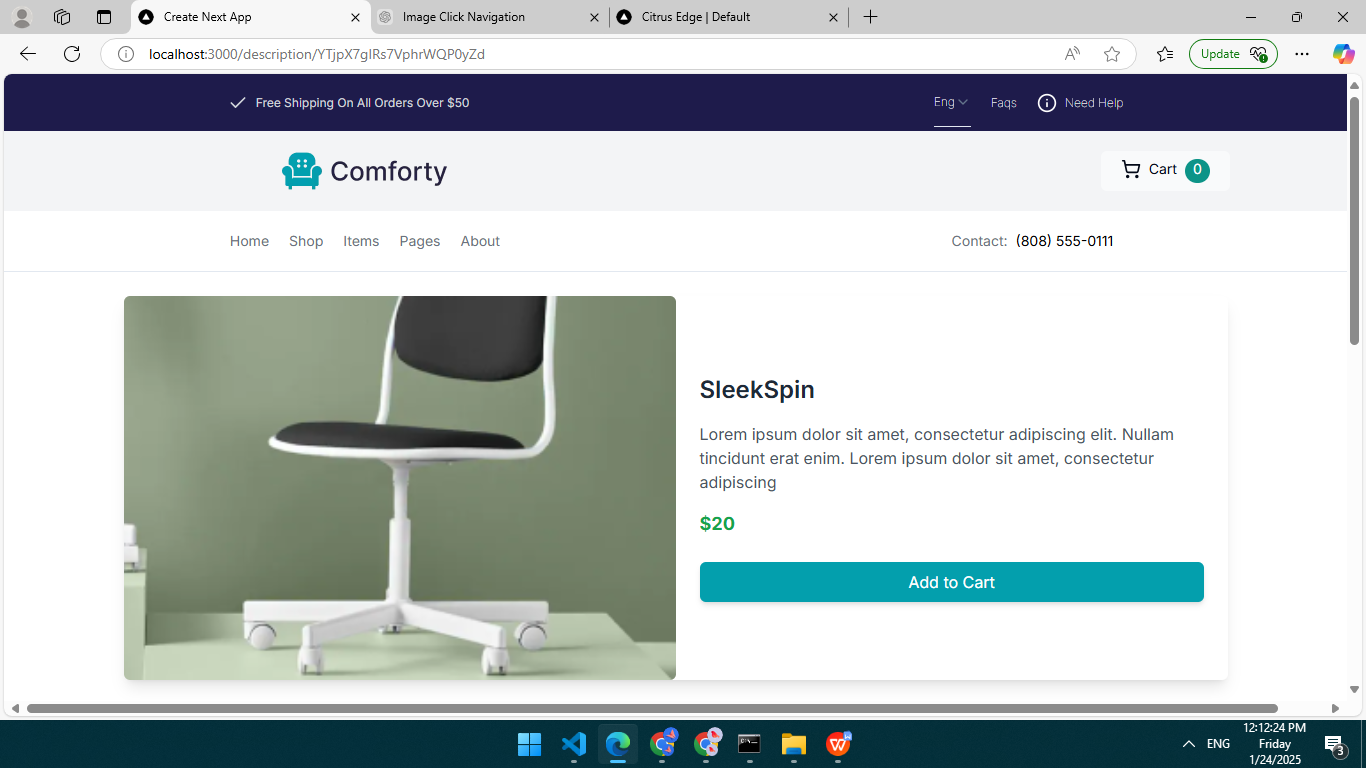
* **Screenshots of product listing page with dynamic data:**



* **Screenshots of Individual product detail pages with accurate routing and**

**data rendering**





1. **Code Deliverables:**

* **Code snippets for key components (e.g., ProductCard,**

**Product-list)**

**Product Cards**

"use client";

import { useSelector, useDispatch } from "react-redux";

import { RootState } from "../store";

import { removeFromCart, updateQuantity } from "@/features/cart/cartSlice";

import Image from "next/image";

import { TextField } from "@mui/material";

import { AiOutlineDelete } from "react-icons/ai";

import { toast } from "react-toastify"; // Import toast for notifications

import { CartItem } from "@/features/cart/cartSlice";

const sanity = sanityClient({

  projectId: "b53sh8tt”

  dataset: "production",

  apiVersion: "2021-08-31",

  useCdn: true,

});

export interface IProducts {

  \_id: string;

  title: string;

  price: number;

  priceWithoutDiscount?: number;

  badge?: string;

  image: string;

  tags?: string[];

  description?: string;

  inventory?: number;

}

}

}

const ProductList = ({ products }: { products: IProducts[] }) => {

  const dispatch = useDispatch();

  const router = useRouter();

  const fetchProducts = async () => {

  const query = `

        \*[\_type == "products" && "featured" in tags][0...4]{

            \_id,

            title,

            price,

            priceWithoutDiscount,

            badge,

            image,

            tags

        }

    `;

  const products = await client.fetch(query);

  return products;

};

  useEffect(() => {

    fetchProducts();

  }, []);

  const handleAddToCart = (product: Product) => {

    const cartItem: CartItem = { ...product, quantity: 1 };

    addToCart(cartItem);

    alert(`${product.title} added to cart!`);

  };

 return (

    <div className="px-4 sm:px-8 lg:px-16 py-6 bg-white">

      {/\* Page Heading \*/}

      <h1 className="text-2xl font-bold text-gray-900 mb-6 text-center sm:text-left px-4 sm:px-16">

        Featured Products

      </h1>

      {/\* Product Grid \*/}

      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6 px-4 sm:px-16">

        {products.map((product: IProducts) => (

          <div key={product.\_id} className="relative rounded-lg cursor-pointer bg-white">

            {/\* Badge \*/}

            {product.badge && (

              <span

                className={`absolute top-2 left-2 px-2 py-1 text-white text-sm font-semibold rounded-md ${product.badge === "New" ? "bg-green-500" : "bg-orange-500"

                  }`}

              >

                {product.badge}

              </span>

            )}

            {/\* Product Image \*/}

            <div className="w-full h-72 mb-4">

              <Image

                src={urlFor(product.image).url()}

                alt={product.title}

                width={250}

                height={500}

                className="w-full h-full object-cover rounded-md"

              />

            </div>

            {/\* Product Title \*/}

            <h3 className="text-base text-gray-800 hover:text-cyan-600 mb-2">

              {product.title}

            </h3>

            {/\* Prices \*/}

            <div className="mb-4">

              <span className="text-base font-semibold text-gray-800">

                ${product.price}

              </span>

              {product.priceWithoutDiscount && (

                <span className="text-sm font-semibold text-gray-500 line-through ml-2">

                  ${product.priceWithoutDiscount}

                </span>

              )}

            </div>

            {/\* Add to Cart Button \*/}

            <div

              className="absolute bottom-4 right-4 p-2 bg-gray-200 text-black rounded-lg transition-all hover:text-white hover:bg-cyan-600 cursor-pointer"

              onClick={(e) => {

                e.preventDefault(); // Prevents navigation

                handleAddToCart(product); // Add to Cart on click

              }}

            >

              <CiShoppingCart className="w-6 h-6" />

            </div>

            {/\* Product Detail Link \*/}

            <Link href={`/products/${product.\_id}`}>

              {/\* No need to modify Link, it's used for product detail page \*/}

            </Link>

          </div>

        ))}

      </div>

    </div>

  );

};

export default Featured;

**ProductList:**

"use client";

import { useDispatch } from "react-redux";

import { addToCart } from "@/features/cart/cartSlice";

import { IProducts } from "@/types/IProducts";

import { urlFor } from "@/sanity/lib/image";

import { CiShoppingCart } from "react-icons/ci";

import Image from "next/image";

import Link from "next/link"; // Import Link component for dynamic routing

import { toast } from "react-toastify"; // Import toast

import "react-toastify/dist/ReactToastify.css"; // Make sure you import styles

import { useRouter } from "next/navigation";

const ProductList = ({ products }: { products: IProducts[] }) => {

  const dispatch = useDispatch();

  const router = useRouter();

  // Handle adding to cart

  const handleAddToCart = (product: IProducts) => {

    dispatch(

      addToCart({

        id: product.\_id,

        name: product.title,

        image: urlFor(product.image).url(),

        price: product.price,

        quantity: 1, // Default quantity is 1

      })

    );

    // Show toast notification

    toast.success(`${product.title} has been added to the cart!`, {

      position: "bottom-right",

      autoClose: 3000, // Toast disappears in 3 seconds

      closeOnClick: true,

      pauseOnHover: true,

      draggable: true,

      progress: undefined, // Default progress bar for smooth closing

      theme: "dark", // Optional: Use "dark" if you prefer

      onClick: () => router.push("/cart"), // Navigate to cart on click

    });

  };

  return (

    <div className="px-4 sm:px-8 lg:px-16 py-6 bg-white">

      {/\* Page Heading \*/}

      <h1 className="text-2xl font-bold text-gray-900 mb-6 text-center sm:text-left px-4 sm:px-16">

        All Products

      </h1>

      {/\* Product Grid \*/}

      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-6 px-4 sm:px-16">

        {products.map((product) => (

          <div key={product.\_id} className="relative rounded-lg cursor-pointer bg-white">

            {/\* Link to Product Detail Page \*/}

            <Link href={`/products/${product.\_id}`} passHref>

              {/\* Badge \*/}

              {product.badge && (

                <span

                  className={`absolute top-2 left-2 px-2 py-1 text-white text-sm font-semibold rounded-md ${product.badge === "New" ? "bg-green-500" : "bg-orange-500"

                    }`}

                >

                  {product.badge}

                </span>

              )}

              {/\* Product Image \*/}

              <div className="w-full h-72 mb-4">

                <Image

                  src={urlFor(product.image).url()}

                  alt={product.title}

                  width={250}

                  height={300}

                  className="w-full h-full object-cover rounded-md"

                />

              </div>

              {/\* Product Title \*/}

              <h3 className="text-base text-gray-800 hover:text-cyan-600 mb-2">

                {product.title}

              </h3>

              {/\* Prices \*/}

              <div className="mb-4">

                <span className="text-base font-semibold text-gray-800">

                  ${product.price}

                </span>

                {product.priceWithoutDiscount && (

                  <span className="text-sm font-semibold text-gray-500 line-through ml-2">

                    ${product.priceWithoutDiscount}

                  </span>

                )}

              </div>

            </Link>

            {/\* Cart Icon \*/}

            <div

              className="absolute bottom-4 right-4 p-2 bg-gray-200 text-black rounded-lg transition-all hover:text-white hover:bg-cyan-600 cursor-pointer"

              onClick={() => handleAddToCart(product)} // Add to Cart on click

            >

              <CiShoppingCart className="w-6 h-6" />

            </div>

          </div>

        ))}

      </div>

    </div>

  );

};

export default ProductList;

**Scripts or logic for API integration and dynamic routing**

// Import environment variables from .env.local

import "dotenv/config";

// Import the Sanity client to interact with the Sanity backend

import { createClient } from "@sanity/client";

// Load required environment variables

const {

  NEXT\_PUBLIC\_SANITY\_PROJECT\_ID, // Sanity project ID

  NEXT\_PUBLIC\_SANITY\_DATASET, // Sanity dataset (e.g., "production")

  NEXT\_PUBLIC\_SANITY\_AUTH\_TOKEN, // Sanity API token

} = process.env;

// Check if the required environment variables are provided

if (!NEXT\_PUBLIC\_SANITY\_PROJECT\_ID || !NEXT\_PUBLIC\_SANITY\_AUTH\_TOKEN) {

  console.error("Missing required environment variables. Please check your .env.local file.");

  process.exit(1); // Stop execution if variables are missing

}

// Create a Sanity client instance to interact with the target Sanity dataset

const targetClient = createClient({

  projectId: NEXT\_PUBLIC\_SANITY\_PROJECT\_ID, // Your Sanity project ID

  dataset: NEXT\_PUBLIC\_SANITY\_DATASET || "production", // Default to "production" if not set

  useCdn: false, // Disable CDN for real-time updates

  apiVersion: "2023-01-01", // Sanity API version

  token: NEXT\_PUBLIC\_SANITY\_AUTH\_TOKEN, // API token for authentication

});

// Function to delete documents from Sanity by type

async function deleteDocumentsByType(documentType) {

  try {

    console.log(`Fetching documents of type: ${documentType}...`);

    // Fetch all documents of the given type

    const documents = await targetClient.fetch(`\*[\_type == "${documentType}"]`);

    if (documents.length === 0) {

      console.log(`No documents found for type: ${documentType}`);

      return;

    }

    console.log(`Found ${documents.length} documents. Deleting...`);

    // Delete each document

    for (const doc of documents) {

      await targetClient.delete(doc.\_id);

      console.log(`Deleted document: ${doc.\_id} (${doc.title || "No Title"})`);

    }

    console.log(`All documents of type "${documentType}" deleted successfully.`);

  } catch (error) {

    console.error(`Error deleting documents of type "${documentType}":`, error.message);

  }

}

// Main function to clean up categories and products

async function cleanupData() {

  console.log("Starting data cleanup...");

  try {

    // Delete all products

    await deleteDocumentsByType("products");

    // Delete all categories

    await deleteDocumentsByType("categories");

    console.log("Data cleanup completed successfully!");

  } catch (error) {

    console.error("Error during cleanup:", error.message);

    process.exit(1); // Stop execution if an error occurs

  }

}

// Start the cleanup process

cleanupData();

**migrate.mjs**

// Import environment variables from .env.local

import \* as dotenv from "dotenv";

dotenv.config({ path: ".env.local" });

// Import the Sanity client to interact with the Sanity backend

import { createClient } from "@sanity/client";

// Load required environment variables

const {

  NEXT\_PUBLIC\_SANITY\_PROJECT\_ID, // Sanity project ID

  NEXT\_PUBLIC\_SANITY\_DATASET, // Sanity dataset (e.g., "production")

  NEXT\_PUBLIC\_SANITY\_AUTH\_TOKEN, // Sanity API token

  BASE\_URL, // API base URL for products and categories

} = process.env;

// Check if the required environment variables are provided

if (!NEXT\_PUBLIC\_SANITY\_PROJECT\_ID || !NEXT\_PUBLIC\_SANITY\_AUTH\_TOKEN) {

  console.error("Missing required environment variables. Please check your .env.local file.");

  process.exit(1); // Stop execution if variables are missing

}

// Create a Sanity client instance to interact with the target Sanity dataset

const targetClient = createClient({

  projectId: NEXT\_PUBLIC\_SANITY\_PROJECT\_ID, // Your Sanity project ID

  dataset: NEXT\_PUBLIC\_SANITY\_DATASET || "production", // Default to "production" if not set

  useCdn: false, // Disable CDN for real-time updates

  apiVersion: "2023-01-01", // Sanity API version

  token: NEXT\_PUBLIC\_SANITY\_AUTH\_TOKEN, // API token for authentication

});

// Function to upload an image to Sanity

async function uploadImageToSanity(imageUrl) {

  try {

    // Fetch the image from the provided URL

    const response = await fetch(imageUrl);

    if (!response.ok) throw new Error(`Failed to fetch image: ${imageUrl}`);

    // Convert the image to a buffer (binary format)

    const buffer = await response.arrayBuffer();

    // Upload the image to Sanity and get its asset ID

    const uploadedAsset = await targetClient.assets.upload("image", Buffer.from(buffer), {

      filename: imageUrl.split("/").pop(), // Use the file name from the URL

    });

    return uploadedAsset.\_id; // Return the asset ID

  } catch (error) {

    console.error("Error uploading image:", error.message);

    return null; // Return null if the upload fails

  }

}

// Main function to migrate data from REST API to Sanity

async function migrateData() {

  console.log("Starting data migration...");

  try {

    // Fetch categories from the REST API

    const categoriesResponse = await fetch(`${BASE\_URL}/api/categories`);

    if (!categoriesResponse.ok) throw new Error("Failed to fetch categories.");

    const categoriesData = await categoriesResponse.json(); // Parse response to JSON

    // Fetch products from the REST API

    const productsResponse = await fetch(`${BASE\_URL}/api/products`);

    if (!productsResponse.ok) throw new Error("Failed to fetch products.");

    const productsData = await productsResponse.json(); // Parse response to JSON

    const categoryIdMap = {}; // Map to store migrated category IDs

    // Migrate categories

    for (const category of categoriesData) {

      console.log(`Migrating category: ${category.title}`);

      const imageId = await uploadImageToSanity(category.imageUrl); // Upload category image

      // Prepare the new category object

      const newCategory = {

        \_id: category.\_id, // Use the same ID for reference mapping

        \_type: "categories",

        title: category.title,

        image: imageId ? { \_type: "image", asset: { \_ref: imageId } } : undefined, // Add image if uploaded

      };

      // Save the category to Sanity

      const result = await targetClient.createOrReplace(newCategory);

      categoryIdMap[category.\_id] = result.\_id; // Store the new category ID

      console.log(`Migrated category: ${category.title} (ID: ${result.\_id})`);

    }

    // Migrate products

    for (const product of productsData) {

      console.log(`Migrating product: ${product.title}`);

      const imageId = await uploadImageToSanity(product.imageUrl); // Upload product image

      // Prepare the new product object

      const newProduct = {

        \_type: "products",

        title: product.title,

        price: product.price,

        priceWithoutDiscount: product.priceWithoutDiscount,

        badge: product.badge,

        image: imageId ? { \_type: "image", asset: { \_ref: imageId } } : undefined, // Add image if uploaded

        category: {

          \_type: "reference",

          \_ref: categoryIdMap[product.category.\_id], // Use the migrated category ID

        },

        description: product.description,

        inventory: product.inventory,

        tags: product.tags,

      };

      // Save the product to Sanity

      const result = await targetClient.create(newProduct);

      console.log(`Migrated product: ${product.title} (ID: ${result.\_id})`);

    }

    console.log("Data migration completed successfully!");

  } catch (error) {

    console.error("Error during migration:", error.message);

    process.exit(1); // Stop execution if an error occurs

  }

}

// Start the migration process

migrateData();

### ****Technical Report for E-commerce Web Application Development****

#### ****1. Steps Taken to Build and Integrate Components****

**Component Design:**

* + Reusable components were created for key areas like Navbar, Footer, and ProductPages. These components ensured a consistent UI/UX across all pages.
  + The CartContext was implemented to manage cart state globally, enabling seamless updates across components.

**Styling and Responsiveness**

* + Tailwind CSS was used for styling. Responsive utilities provided by Tailwind ensured compatibility across devices without distorting the design.
  + Special attention was given to dynamic layouts, ensuring a smooth user experience on small and large devices.

**Data Fetching and Display**:

* + Product data was fetched from a Sanity CMS backend using GROQ queries, ensuring scalable and real-time data management.
  + Conditional logic handled scenarios like badges (New, Sale, etc.), ensuring contextual display of product attributes.

#### ****2. Challenges Faced and Solutions Implemented****

**CORS Issue**:  
Initially, product data was not fetched due to a CORS restriction. This was resolved by configuring the correct Vercel deployment URL in the Sanity CMS's CORS settings.

**Badge Logic**:  
Complex badge logic was implemented to display conditional badges like New or Sale. The solution involved mapping badge values to colors and ensuring fallback options for undefined values.

**Responsive Design**:  
Ensuring a consistent design across all devices, particularly for navigation and sidebars, required multiple iterations. Tailwind's responsive utilities (e.g., sm, md, lg) simplified this process.

#### ****3. Best Practices Followed During Development****

**Coding Standards**:

* + ESLint and Prettier were used to maintain clean and consistent code.
  + Code modularity ensured easy maintenance and reusability.

**Version Control**:

* + Git was used for version control, with a branching model to handle feature-specific changes and bug fixes.

**Testing and Debugging**:

* + Test cases for core functionalities like cart management (addToCart and useCart) were implemented.
  + Lighthouse was used to optimize performance, accessibility, and SEO.

**Deployment**:

* + Environment-specific configurations were set during Vercel deployment, ensuring proper integration with Sanity CMS and dynamic data fetching.

#### ****4. Performance and Accessibility Optimizations****

**Performance**:

* + Lazy loading was applied to images using Next.js' <Image> component.
  + A reduced bundle size was achieved by ensuring only necessary packages and scripts were included.

**Accessibility**:

* + Tailwind utilities and semantic HTML were used to ensure WCAG compliance.
  + Interactive elements (e.g., buttons, links) included proper focus states and ARIA labels where necessary.

Checklist for Day 4:

Self-Validation Checklist:

*Frontend Component Development:*

✔

*Styling and Responsiveness:*

✔

*Code Quality:*

✔

*Documentation and Submission:*

✔ KETPLACE

*Final Review:*

✔