Day 4 - Dynamic Frontend Components for Marketplace - Comforty

1. Introduction

This document provides a comprehensive report on the development of dynamic frontend components for the marketplace application. It highlights the objectives, key learning outcomes, implementation steps, challenges, and best practices followed. Screenshots and code snippets are included to showcase the progress and final output.

---

2. Objective

The goal of this task was to design and develop **dynamic frontend components** that fetch and display **marketplace data** from Sanity CMS or APIs. The components were built with a focus on **modularity, reusability, and responsiveness** to ensure scalability in the application.

---

3. Key Learning Outcomes

- Development of **dynamic frontend components** using Next.js.
- Implementation of **modular and reusable** UI components.
- Application of **state management techniques**.
- Enhancement of **user experience (UX) and UI responsiveness**.
- Understanding of **real-world project workflows**.

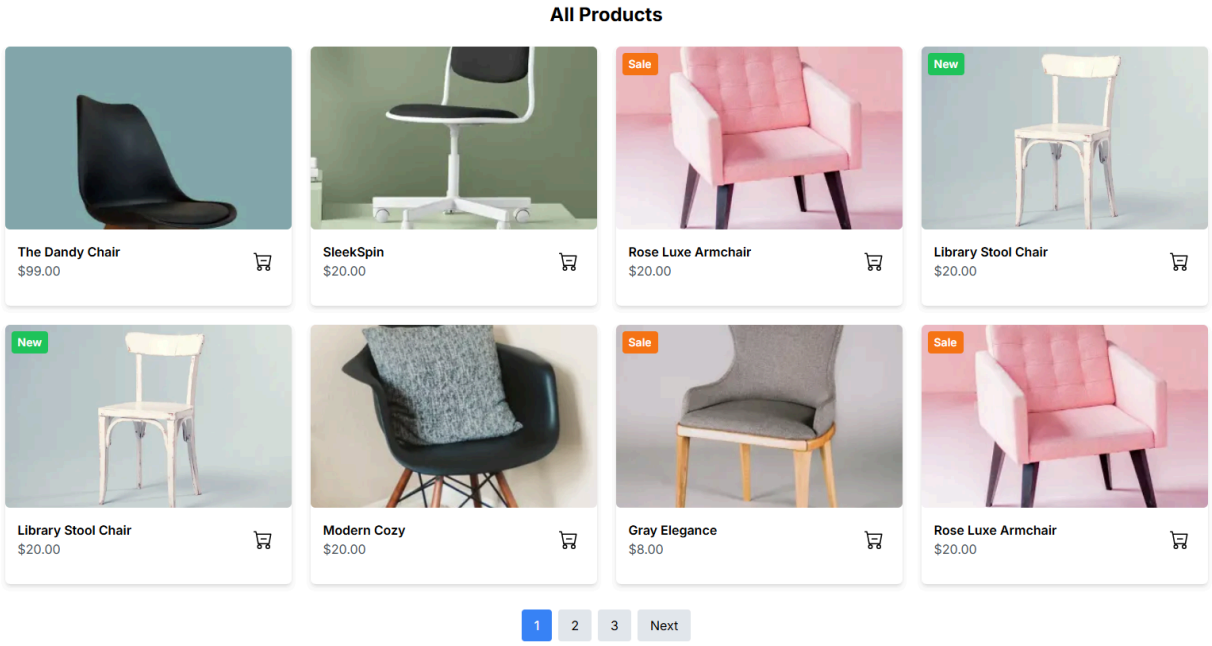
---

4. Functionalities Implemented

4.1 Product Listing

- Displays product details dynamically in a **grid layout** by fetching data from **Sanity CMS**.
- It offers the option of search, tag and category filter and pagination.

Screenshot:



Code Snippet of Product Card Component

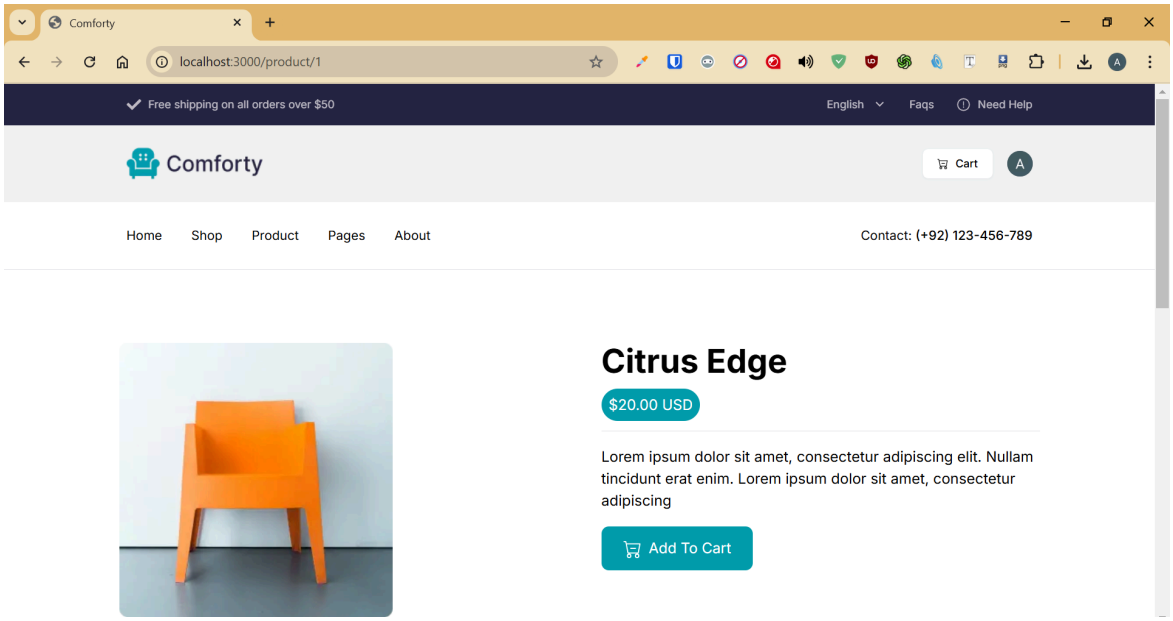
```tsx
import Image from "next/image";
import Link from "next/link";
import { BsCartDash } from "react-icons/bs";
import toast from 'react-hot-toast';

interface ProductType {
  id: string;
  name: string;
  price: number;
  image: string;
  slug: number;
  isNew?: boolean;
  onSale?: boolean;
};

import { useCart } from "@/app/cart/context/CartContext";

const ProductCard = ({ product }: { product: ProductType }) => {
  const cartContext = useCart(); // Access CartContext
  const addToCart = cartContext?.addToCart; // Safely access addToCart

  const handleAddToCart = () => {
    if (addToCart) {
      addToCart({
        id: product.id,
        name: product.name,
        price: product.price,
        image: product.image,
        quantity: 1,
      });

      toast(`${product.name} has been added to the cart!`);
    } else {
      console.error("addToCart function is not available");
      toast.error("Error adding to cart. Please try again.");
    }
  };


  return (
    <div className="rounded-md overflow-hidden relative shadow-md transition-transform transform
hover:scale-105 bg-white">
      {/* Link to Product Details */}
      <Link href={`/product/${product.slug}`}>
        <div className="relative w-full h-[230px] overflow-hidden rounded-md">
          {/* New/On Sale Labels */}
          {product.isNew && (
            <div className="absolute top-2 left-2 bg-green-500 text-white px-2 py-1 rounded-sm text-sm
font-semibold">
              New
            </div>
          )}
          {product.onSale && (
            <div className="absolute top-2 left-2 bg-orange-500 text-white px-2 py-1 rounded-sm text-sm
font-semibold">
              Sale
            </div>
          )}

          {/* Product Image */}
          <div className="flex items-center justify-center h-full">
            <Image
              src={product.image}
              alt={product.name}
              width={280}
              height={280}
              layout="responsive"
              objectFit="cover"
            />
          </div>
        </div>
      </Link>

      {/* Product Info */}
      <div className="p-4">
        <div className="flex justify-between items-center mb-4">
          <div>
            <h3 className="text-base font-semibold">{product.name}</h3>
            <p className="text-gray-600">${product.price.toFixed(2)}</p>
          </div>

          {/* Add to Cart Button */}
          <button
            className="p-2 rounded-full hover:bg-[#029FAE] hover:text-white transition-all duration-
200"
            aria-label="Add to cart"
            onClick={handleAddToCart} // Add click handler
          >
            <BsCartDash className="text-2xl" />
          </button>
        </div>
      </div>
    </div>
  );
};

export default ProductCard;
```

4.2 Product Detail Component

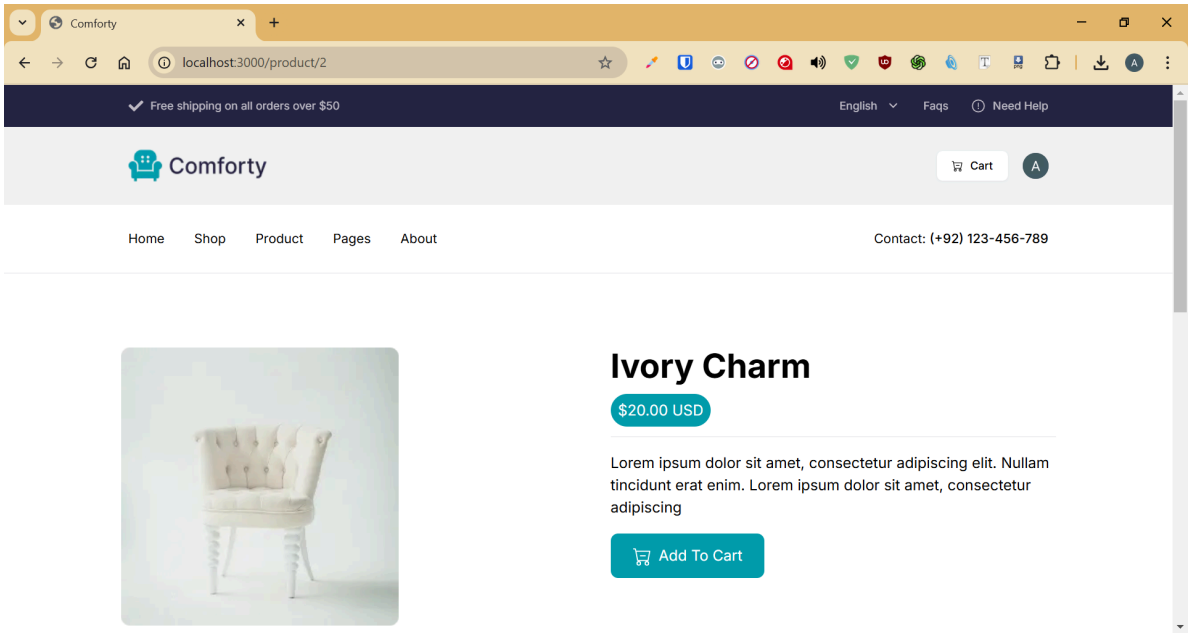
- Implements **dynamic routing** for individual product pages.
- Displays **product image, name, description and price**.

Screenshots of Different Product Pages:

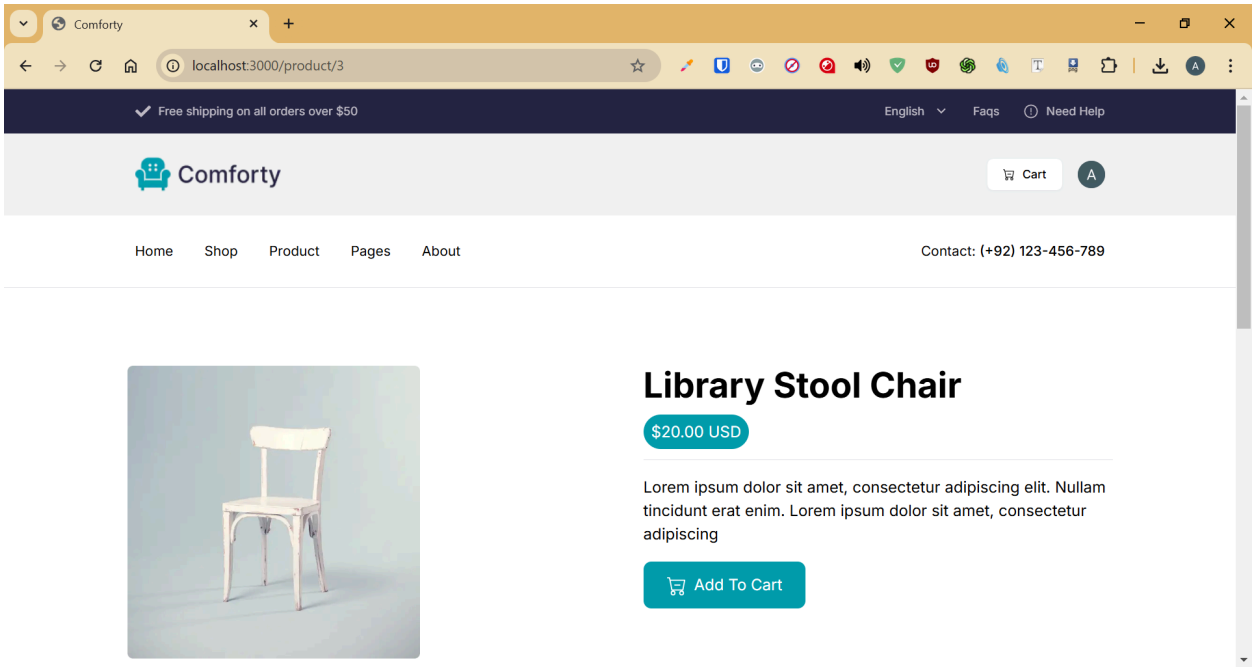
Product 1



Product 2



Product 3



Code Snippet of Product Details Page

```tsx
"use client";

import React, { useEffect, useState } from "react";
import Image from "next/image";
import { client } from "../../../sanity/lib/client";
import { BsCartDash } from "react-icons/bs";
import FeaturedProductsCard from "@/app/components/cards/FeaturedProductsCard";
import { useCart } from "@/app/cart/context/CartContext";

interface ProductType {
  id: string;
  name: string;
  price: number;
  onSale: boolean;
  isNew: boolean;
  image: string;
  description: string;
  slug: number;
}

const ProductPage = ({ params }: { params: { id: number } }) => {
  const [products, setProducts] = useState<ProductType[]>([]);
  const [featuredProducts, setFeaturedProducts] = useState<ProductType[]>([]);
  const [showAll, setShowAll] = useState(false); // State to toggle view
  const { addToCart } = useCart();

  useEffect(() => {
    const fetchProducts = async () => {
      try {
        const query1 = `*[_type == "products"]{
          "id": _id,
          "name": title,
          price,
          "onSale": badge == "Sales",
          "isNew": badge == "New",
          "image": image.asset->url,
          description,
          slug
        }`;

        const query2 = `*[_type == "products" && "featured" in tags]{
          "id": _id,
          "name": title,
          price,
          "onSale": badge == "Sales",
          "isNew": badge == "New",
          "image": image.asset->url,
          slug
        }`;

        const [fetchedProducts, fetchedFeaturedProducts] = await Promise.all([
          client.fetch(query1),
          client.fetch(query2),
        ]);

        setProducts(fetchedProducts);
        setFeaturedProducts(fetchedFeaturedProducts);
      } catch (error) {
        console.error("Error fetching products:", error);
      }
    };

    fetchProducts();
  }, []);
```

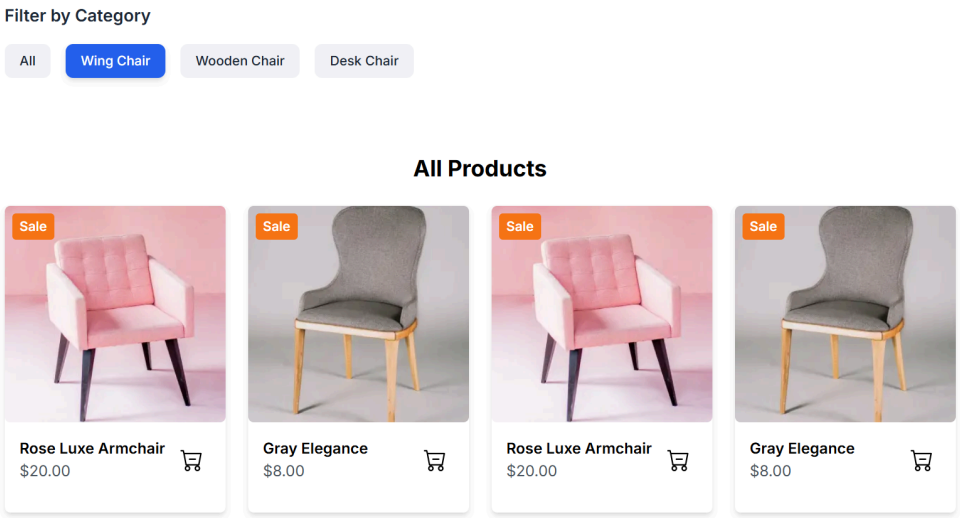
```
  // Convert params.id to a number if it comes as a string
  const productId = Number(params.id);

  // Find the product with the matching slug
  const product = products.find((p) => p.slug === productId);

  if (!product) {
    return <div className="text-center mt-20">Product not found</div>;
  }

  const { id, name, price, description, image } = product;

  // Determine the products to display in the featured section
  const displayedFeaturedProducts = showAll
    ? featuredProducts
    : featuredProducts.slice(0, 5);

  return (
    <div className="w-[80%] mx-auto mt-20">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-12">
        <div>
          <Image
            src={image}
            alt={name}
            width={300}
            height={300}
            style={{ objectFit: "contain" }}
          />
        </div>
        <div className="text-center md:text-start space-y-4">
          <h1 className="text-xl lg:text-2xl xl:text-4xl 2xl:text-6xl font-bold mb-4">
            {name}
          </h1>
          <span className="bg-[#029FAE] text-white rounded-full p-2">{`$${price.toLocaleString()}.00
USD`}</span>
          <hr />
          <p>{description}</p>
          <button
            className="bg-[#029FAE] text-white px-6 py-3 rounded-lg"
            onClick={() => addToCart({ ...product, quantity: 1 })}
          >
            <BsCartDash className="inline mr-2 text-xl" />
            Add To Cart
          </button>
        </div>
      </div>
      <div className="mt-20">
        <div className="flex justify-between items-center">
          <h1 className="md:text-xl lg:text-2xl xl:text-3xl font-bold mb-10 text-center md:text-start">
            FEATURED PRODUCTS
          </h1>
          {/* Conditionally render the "View All" button */}
          {featuredProducts.length > 5 && (
            <button
              className="text-[#029FAE]"
              onClick={() => setShowAll(!showAll)}
            >
              {showAll ? "View Less" : "View All"}
            </button>
          )}
        </div>

        <div className="flex flex-col md:flex-row flex-wrap gap-4">
          {displayedFeaturedProducts.map((product: ProductType) => (
            <FeaturedProductsCard
              key={product.id}
              id={product.id}
              image={product.image}
              name={product.name}
              price={product.price}
              onSale={product.onSale}
              isNew={product.isNew}
              slug={product.slug}
            />
          ))}
        </div>
      </div>
    </div>
  );
};

export default ProductPage;
```

4.3 Category Component

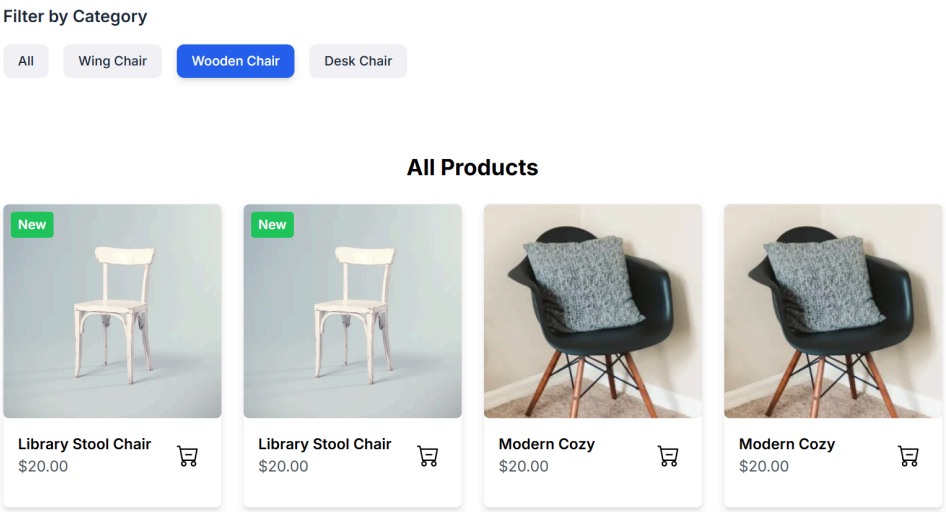
- Enables users to filter products by **category**.

Screenshots of Products Sorted by Categories:

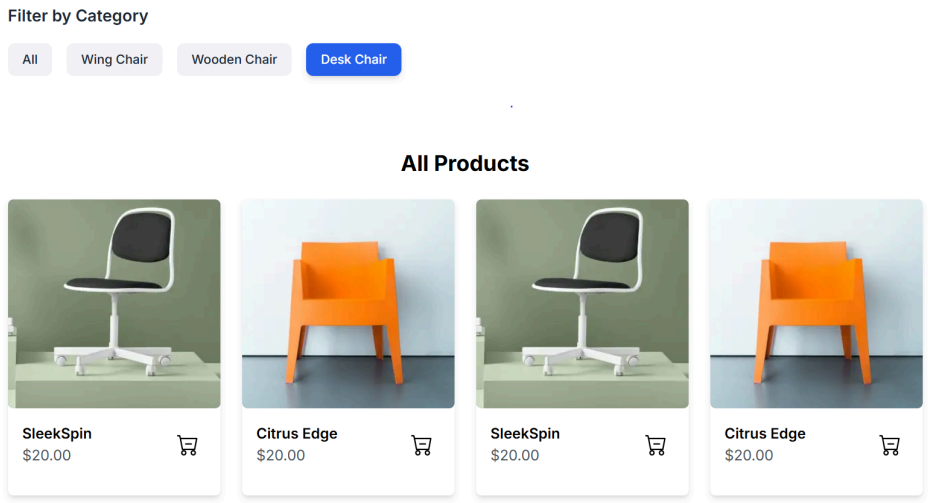
Wing Chair Category Products



Wooden Chair Category Products



Desk Chair Category Products



Code Snippet of Category Component

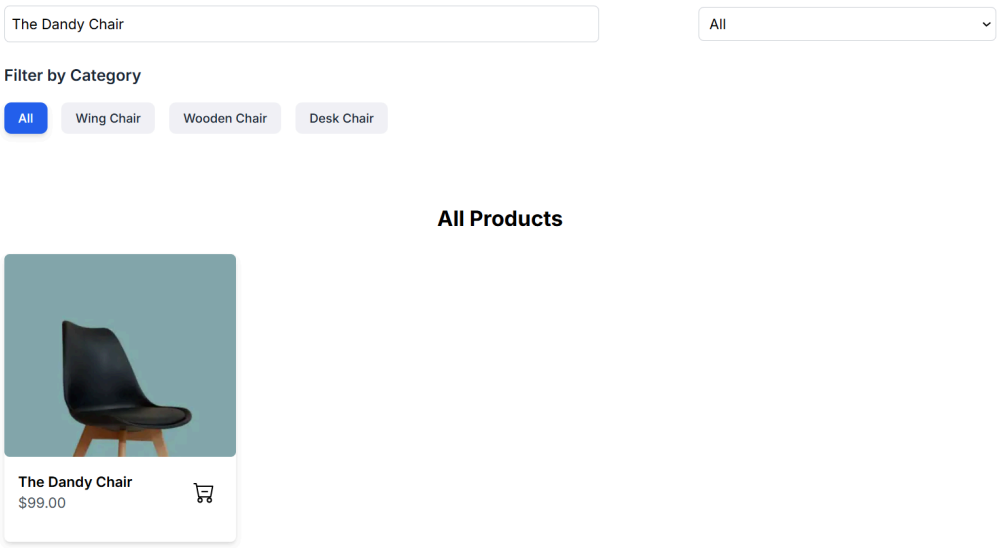
```tsx
import React from 'react';

interface CategoryType {
  id: string;
  name: string;
}

interface CategoriesProps {
  categories: CategoryType[];
  selectedCategory: string | null;
  onCategorySelect: (category: string | null) => void;
}

const Categories: React.FC<CategoriesProps> = ({ categories, selectedCategory, onCategorySelect }) => {
  return (
    <div className="w-[80%] mx-auto mt-6">
      <h2 className="text-lg font-semibold text-gray-800 mb-4">Filter by Category</h2>
      <ul className="flex flex-wrap gap-4">
        {/* All Categories Option */}
        <li
          className={`px-4 py-2 rounded-lg cursor-pointer text-sm font-medium transition-colors ${
            selectedCategory === null
              ? 'bg-blue-600 text-white shadow-md'
              : 'bg-gray-100 text-gray-800 hover:bg-blue-100'
          }`}
          onClick={() => onCategorySelect(null)}
        >
          All
        </li>
        {/* Render Categories */}
        {categories.map((category) => (
          <li
            key={category.id}
            className={`px-4 py-2 rounded-lg cursor-pointer text-sm font-medium transition-colors ${
              selectedCategory === category.name
                ? 'bg-blue-600 text-white shadow-md'
                : 'bg-gray-100 text-gray-800 hover:bg-blue-100'
            }`}
            onClick={() => onCategorySelect(category.name)}
          >
            {category.name}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default Categories;
```

4.4 Search Bar and Filters

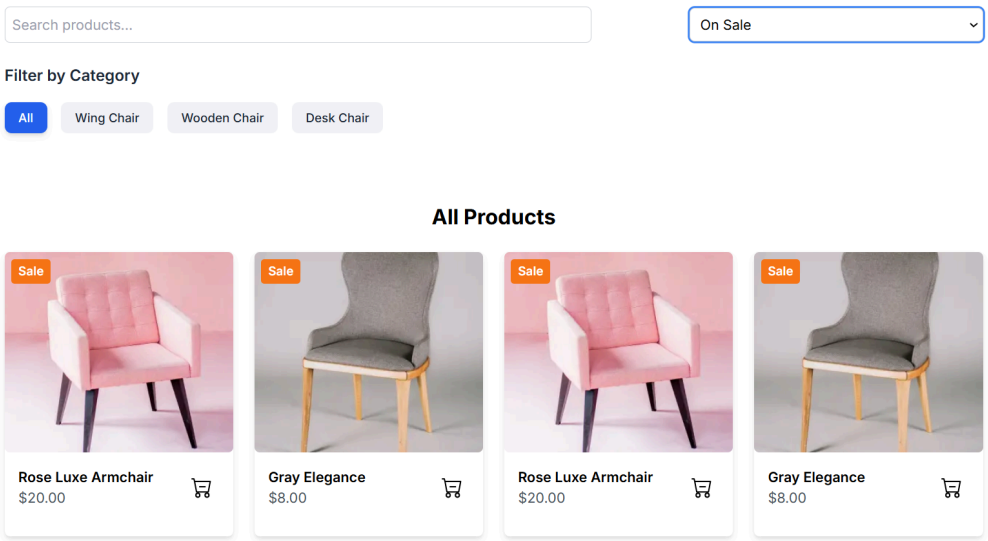
- Implements **search functionality** to filter products by name and tags.

Screenshots:

Search by Name



Search by Tag (Example: On Sale Tag)



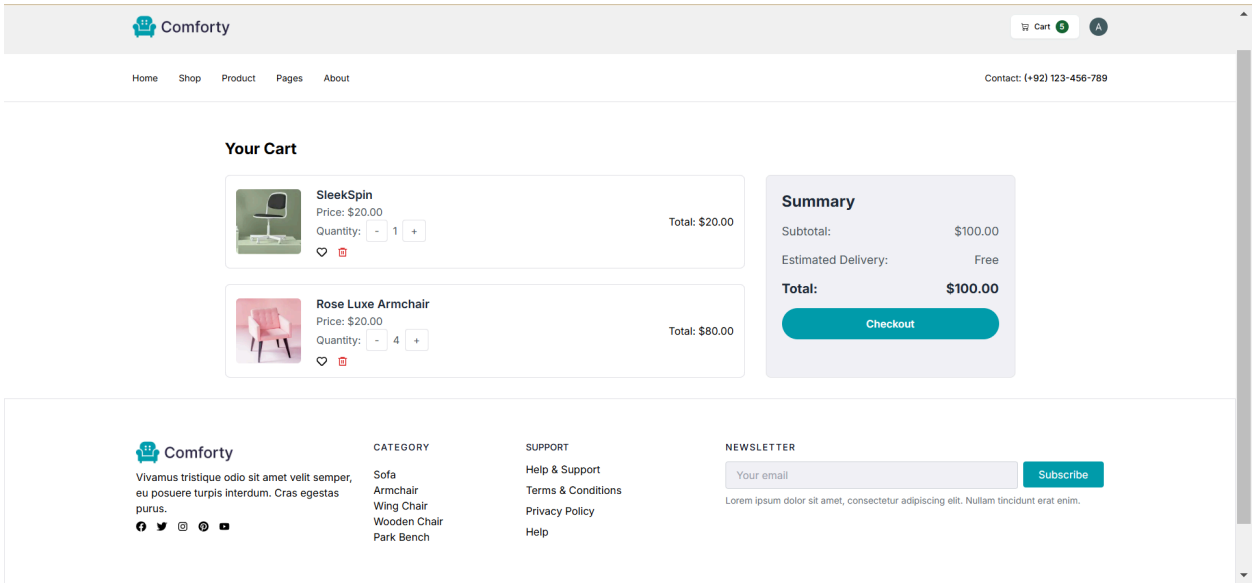
Code Snippet of Search and Filter Component

```tsx
import React from 'react';

interface SearchFilterProps {
  searchQuery: string;
  setSearchQuery: (value: string) => void;
  filter: string;
  setFilter: (value: string) => void;
}

const SearchFilter: React.FC<SearchFilterProps> = ({
  searchQuery,
  setSearchQuery,
  filter,
  setFilter,
}) => {
  return (
    <div className="w-[80%] mx-auto mt-10">
      <div className="flex flex-col sm:flex-row items-center justify-between gap-4">
        {/* Search Input */}
        <input
          type="text"
          value={searchQuery}
          onChange={(e) => setSearchQuery(e.target.value)}
          placeholder="Search products..."
          className="w-full sm:w-[60%] p-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
        />

        {/* Filter Dropdown */}
        <select
          value={filter}
          onChange={(e) => setFilter(e.target.value)}
          className="w-full sm:w-[30%] p-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-blue-500"
        >
          <option value="all">All</option>
          <option value="sale">On Sale</option>
          <option value="new">New Arrivals</option>
        </select>
      </div>
    </div>
  );
};

export default SearchFilter;
```

4.5 Cart Component

- Tracks items added to the **cart using state management**.

Screenshot:



Code Snippet of Cart Component

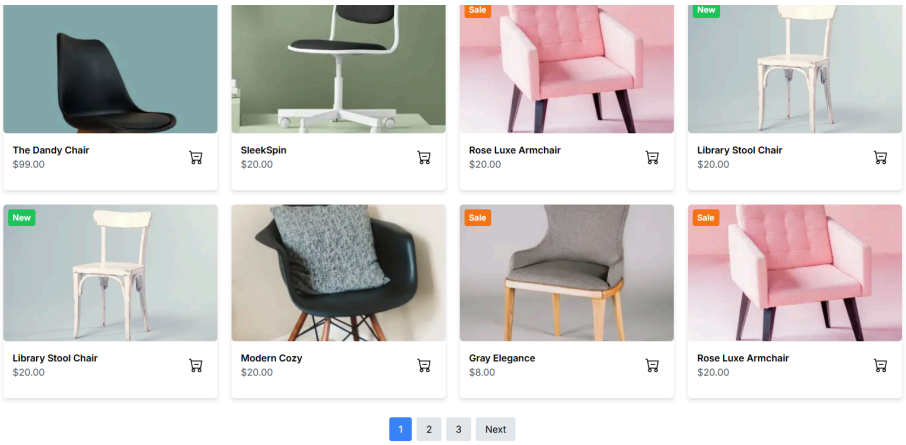
```tsx
"use client";

import { createContext, ReactNode, useContext, useEffect, useState } from "react";

// Define CartItem type
interface CartItem {
  id: string;
  name: string;
  price: number;
  quantity: number;
  image: string;
}

// Define CartContextType
interface CartContextType {
  cart: CartItem[];
  cartItems: CartItem[];
  addToCart: (item: CartItem) => void;
  removeFromCart: (id: string) => void;
  updateQuantity: (id: string, quantity: number) => void;
  clearCart: () => void;
}

// Create Cart context
const CartContext = createContext<CartContextType | undefined>(undefined);

// CartProvider component
export const CartProvider = ({ children }: { children: ReactNode }) => {
  const [cart, setCart] = useState<CartItem[]>([]);

  // 🟢 Load cart from local storage on mount
  useEffect(() => {
    const savedCart = localStorage.getItem("cart");
    if (savedCart) {
      setCart(JSON.parse(savedCart));
    }
  }, []);

  // 🟢 Save cart to local storage whenever it changes
  useEffect(() => {
    localStorage.setItem("cart", JSON.stringify(cart));
  }, [cart]);

  // Function to add item to cart
  const addToCart = (item: CartItem) => {
    console.log("Adding item to cart: ", item);
    setCart((prevItems) => {
      const existingItem = prevItems.find((cartItem) => cartItem.id === item.id);

      if (existingItem) {
        console.log("Item already in cart, updating quantity");
        return prevItems.map((cartItem) =>
          cartItem.id === item.id
            ? { ...cartItem, quantity: cartItem.quantity + 1 }
            : cartItem
        );
      }

      console.log("New item added to cart");
      return [...prevItems, { ...item, quantity: 1 }];
    });
  };

  // Function to remove item from the cart
  const removeFromCart = (id: string) => {
    setCart((prevCart) => prevCart.filter((cartItem) => cartItem.id !== id));
  };

  // Function to update item quantity in the cart
  const updateQuantity = (id: string, quantity: number) => {
    setCart((prevCart) =>
      prevCart.map((cartItem) =>
        cartItem.id === id ? { ...cartItem, quantity } : cartItem
      )
    );
  };

  // Function to clear the entire cart
  const clearCart = () => {
    setCart([]);
  };

  return (
    <CartContext.Provider value={{ cart, cartItems: cart, addToCart, removeFromCart, updateQuantity,
clearCart }}>
      {children}
    </CartContext.Provider>
  );
};

// Custom hook to use Cart context
export const useCart = () => {
  const context = useContext(CartContext);

  if (!context) {
    throw new Error("useCart must be used within a CartProvider");
  }

  return context;
};
```

4.7 Pagination Component

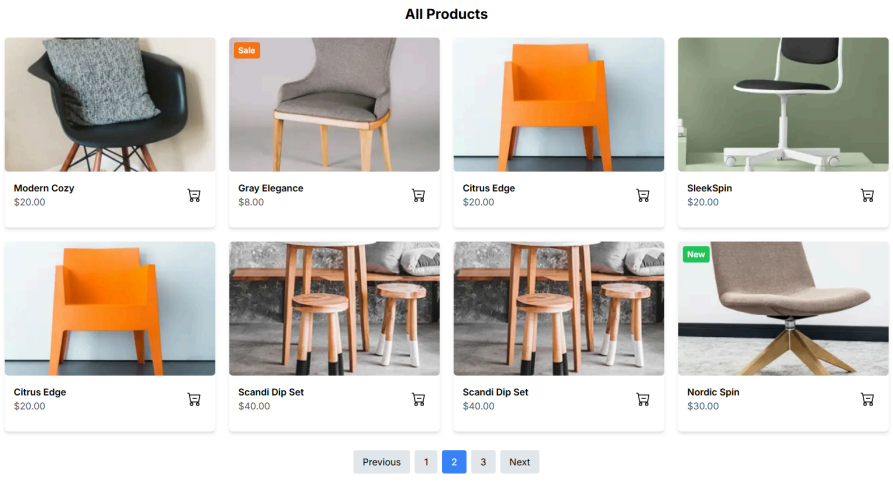
- Handles **large product lists using pagination**.

Screenshots:

Page 1



Page 2



Code Snippet of Pagination Component

```tsx
import React from 'react';

interface PaginationProps {
  currentPage: number;
  totalProducts: number;
  productsPerPage: number;
  onPageChange: (pageNumber: number) => void;
}

const Pagination: React.FC<PaginationProps> = ({
  currentPage,
  totalProducts,
  productsPerPage,
  onPageChange,
}) => {
  const totalPages = Math.ceil(totalProducts / productsPerPage);

  if (totalPages <= 1) return null; // No pagination needed for a single page

  const pageNumbers = Array.from({ length: totalPages }, (_, index) => index +
1);
  return (
    <div className="flex justify-center items-center mt-8">
      <ul className="flex gap-2">
        {currentPage > 1 && (
          <li>
            <button
              onClick={() => onPageChange(currentPage - 1)}
              className="px-4 py-2 bg-gray-200 rounded hover:bg-gray-300"
            >
              Previous
            </button>
          </li>
        )}
        {pageNumbers.map((number) => (
          <li key={number}>
            <button
              onClick={() => onPageChange(number)}
              className={`px-4 py-2 rounded ${
                number === currentPage
                  ? 'bg-blue-500 text-white'
                  : 'bg-gray-200 hover:bg-gray-300'
              }`}
            >
              {number}
            </button>
          </li>
        ))}
        {currentPage < totalPages && (
          <li>
            <button
              onClick={() => onPageChange(currentPage + 1)}
              className="px-4 py-2 bg-gray-200 rounded hover:bg-gray-300"
            >
              Next
            </button>
          </li>
        )}
      </ul>
    </div>
  );
};

export default Pagination;
```

---

5. Steps for Implementation

1. **Setup:** Ensured **Next.js 14.2.2** was connected to **Sanity CMS**.
2. **API Fetching:** Integrated **dynamic data fetching**.
3. **Component Design:** Styled using **Tailwind CSS**.
4. **State Management:** Utilized **useState, useEffect**
5. **Routing:** Implemented **Next.js dynamic routing**.
6. **Interactivity:** Enhanced UX with **event listeners**.

---

6. Challenges & Solutions

- API request failures → Implemented error handling & fallback UI.
- State mismanagement → Used React Context API for global state.
- Responsive issues → Applied Flexbox & Grid layout.
- Slow image loading → Used Next.js Image component.

---

7. Best Practices Followed

- Reusable Components: Modular component design.
- State Management: Optimized data handling.
- Responsive Design: Mobile-first UI with Tailwind.
- Lazy Loading: Performance optimization.

---

8. Conclusion

The development of dynamic frontend components for Comforty Marketplace has been a valuable exercise in modern web development. By implementing reusable components, integrating API data, and ensuring a responsive and scalable design, we have created a flexible and efficient frontend.