

Day 4 - Dynamic Frontend Components – EcoFurnish

Prepared by: Zija Yaseen

1. Functional Deliverables:

Video Demonstration:

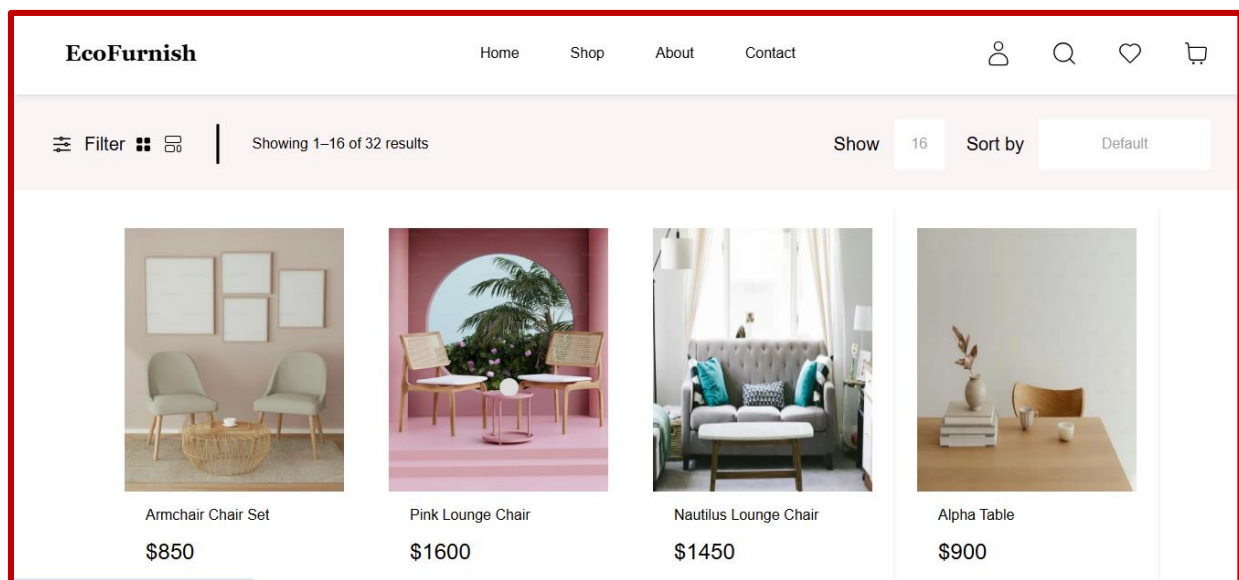
To see these features in action, watch the video demonstration: Click the Link below:

<https://res.cloudinary.com/destuyewf/video/upload/v1738165470/Day-4-sjvd1t.mp4>

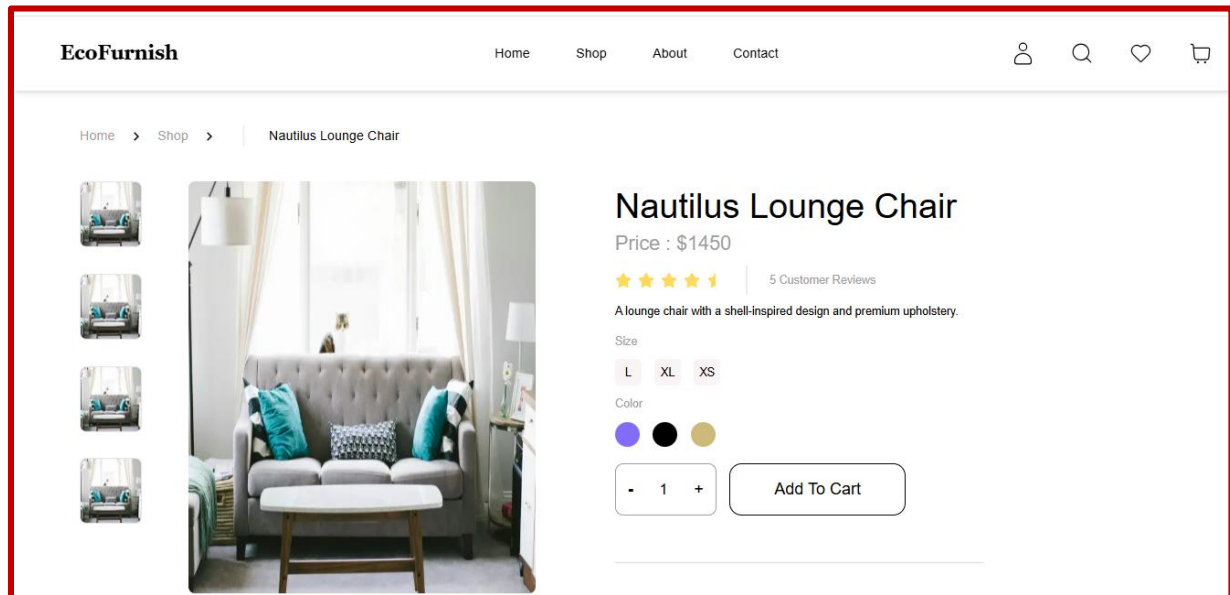
Screenshots:

Below are the screenshots showcasing the implemented features:

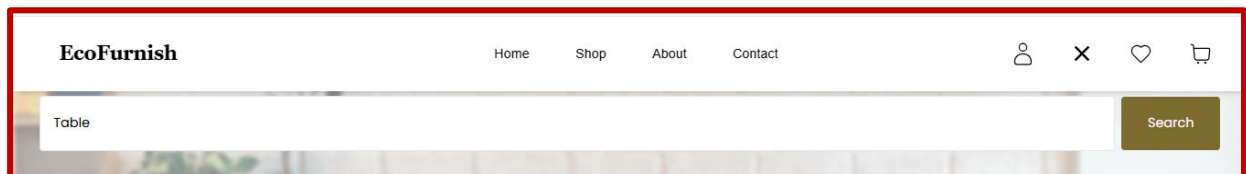
- **Product Listing Page:** Displaying dynamically fetched product data.



- **Individual Product Detail Pages:** Proper routing and data rendering for selected products.



- **Category Filters, Search Bar, and Pagination:** Demonstrating functionality.



2. Code Deliverables:

Key Component Code Snippets

- **ProductList.tsx**

```
"use client"

import React, { useEffect, useState } from 'react';
import { UseAppSelector, UseAppDispatch } from '@redux/hooks';
import { fetchAllProducts, selectPaginatedProducts } from
'@/redux/Search/searchActions';
import Image from 'next/image';
import Link from 'next/link';
import { IProduct } from '@data';

const ProductsList: React.FC = () => {
  const paginatedProducts: IProduct[] = UseAppSelector(selectPaginatedProducts);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const dispatch = UseAppDispatch();

  useEffect(() => {
    const fetchProducts = async () => {
      try {
        setLoading(true);
        setError(null); // Reset any previous error
        if (paginatedProducts.length === 0) {
          await dispatch(fetchAllProducts());
        }
      } catch (err) {
        setError('Failed to load products. Please try again later.');
```

```

    if (error) {
      return (
        <div className="flex justify-center items-center h-[300px] mt-14 font-bold text-2xl text-red-500">
          {error}
        </div>
      );
    }

    return (
      <div className="grid grid-cols-2 md:grid-cols-4 py-4 place-items-center px-6 md:px-16 lg:px-28 gap-[31px]">
        {paginatedProducts.map((product) => (
          <Link href={`Shop/${product._id}`} key={product._id}>
            <div className="w-36 md:w-72 h-auto md:h-[422px] hover:shadow-md flex flex-col justify-center mx-auto">
              <Image
                src={product.imagePath}
                alt={product.name}
                width={600}
                height={600}
                className="w-52 h-32 md:w-60 md:h-72 flex flex-col justify-center mx-auto"
              />
              <div className="w-[130px] md:w-[194px] flex-col justify-center mx-auto">
                <p className="font-semibold text-sm md:text-base mt-2 md:mt-[14px] flex-col justify-center mx-auto">
                  {product.name}
                </p>
                <h3 className="font-medium text-base md:text-xl mt-2 md:mt-3 flex-col justify-center mx-auto">
                  Price: <span className="font-medium text-lg md:text-2xl">${product.price}</span>
                </h3>
              </div>
            </div>
          </Link>
        )))
      </div>
    );
  };
};

export default ProductsList;

```

- **Pagination.tsx**

```
"use client"

import React from 'react';
import { UseAppSelector, UseAppDispatch } from '@redux/hooks';
import { setCurrentPage } from '@redux/Search/searchSlice';

const Pagination = () => {
  const dispatch = UseAppDispatch();
  const { currentPage, itemsPerPage, filteredProducts } = UseAppSelector((state) => state.search);

  const totalPages = Math.ceil(filteredProducts.length / itemsPerPage);

  const handleClick = (page: number) => {
    if (page >= 1 && page <= totalPages) {
      dispatch(setCurrentPage(page));
    }
  };

  return (
    <div className="flex items-center justify-center gap-[20px] my-10">
      <button
        className={`px-4 py-2 rounded-[10px] ${currentPage === 1 ? 'cursor-not-allowed bg-[#FFF9E5]' : 'hover:bg-[#FBE9B5] bg-[#FFF9E5]'}`}
        onClick={() => handleClick(currentPage - 1)}
        disabled={currentPage === 1}
      >
        Prev
      </button>
      {Array.from({ length: totalPages }, (_, i) => (
        <button
          key={i + 1}
          className={`px-4 py-2 rounded-[10px] ${currentPage === i + 1 ? 'bg-[#FBE9B5]' : 'bg-[#FFF9E5]'}`}
          onClick={() => handleClick(i + 1)}
        >
          {i + 1}
        </button>
      ))}
      <button
        className={`px-4 py-2 rounded-[10px] ${currentPage === totalPages ? 'cursor-not-allowed bg-[#FFF9E5]' : 'hover:bg-[#FBE9B5] bg-[#FFF9E5]'}`}

```

```

        onClick={() => handlePageClick(currentPage + 1)}
        disabled={currentPage === totalPages}
      >
        Next
      </button>
    </div>
  );
};

export default Pagination;

```

- **SearchBar.tsx**

```

"use client";

import React, { useState } from "react";
import { UseAppDispatch } from "@redux/hooks";
import { performSearch } from "@redux/Search/searchActions";
import { useRouter } from "next/navigation"; // Correct useRouter import for App Directory
import { MdClose } from "react-icons/md";
import { CiSearch } from "react-icons/ci";

const SearchBar: React.FC = () => {
  const [searchOpen, setSearchOpen] = useState(false);
  const [searchTerm, setSearchTerm] = useState("");
  const dispatch = UseAppDispatch();
  const router = useRouter(); // Ensure it's imported from next/navigation
  console.log(router);

  const handleSearch = () => {
    if (searchTerm.trim() === "") {
      alert("Please enter a search term!");
      return;
    }
    dispatch(performSearch(searchTerm));
    router.push("/Shop"); // Redirect user to the Shop page
  };

  return (

```

```

<div>
  <div className="block z-50 cursor-pointer">
    {searchOpen ? (
      <MdClose
        size={28}
        className="cursor-pointer w-6 h-6 lg:w-8 lg:h-8"
        onClick={() => setSearchOpen(false)}
      />
    ) : (
      <CiSearch
        size={28}
        className="cursor-pointer w-6 h-6 lg:w-8 lg:h-8"
        onClick={() => setSearchOpen(true)}
      />
    )}
  </div>
  {searchOpen && (
    <div className="fixed z-10 md:top-24 top-16 left-4 right-4 md:right-8
md:left-8 max-w-[1440vw]">
      <div className="flex items-center gap-2 w-full">
        <input
          type="text"
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
          placeholder="Search products..."
          className="border px-4 py-2 md:py-5 rounded w-full focus:outline-
none" />
        <button onClick={handleSearch} className="px-4 md:px-8 py-2 md:py-5 bg-
[#7e6b2f] hover:bg-[#b1a067] text-white rounded">
          Search
        </button>
      </div>
    </div>
  )}
</div>
);
};

export default SearchBar;

```

3. Scripts And Logic For API Integration & Dynamic Routing:

Global State Management (Redux):

- Redux is used for global state management.
- The `productSlice` stores fetched product data.
- The `cartSlice` manages the user's cart and checkout process.
- Redux Toolkit's `createAsyncThunk` is used for handling asynchronous API calls.

API Integration with Sanity:

- Product data is stored in Sanity CMS.
- API calls are made using Redux middleware.
- Data is structured according to the Sanity schema and fetched using `axios`.

Pagination and Search Functionality:

- `currentPage` and `itemsPerPage` are stored in Redux state.
- Pagination logic is applied to API responses.
- Search functionality updates the Redux state and filters the displayed results dynamically.

Dynamic Routing in Product Pages:

- Next.js dynamic routing (`[id].tsx`) is implemented.
- The `useRouter()` hook fetches the product ID from the URL.
- Product details are dynamically fetched via Redux state or API calls.

Cart & Checkout System

- Products can be added/removed from the cart using Redux state.
- `cartSlice` defines actions to update the cart.
- The checkout process calculates the total price and processes the order based on Redux state.

3. Documentation

Steps Taken

- **Designed Components:** Created reusable components like ProductCard, ProductList, and SearchBar.
- **Implemented Routing:** Used Next.js dynamic routes for individual product pages.
- **Integrated API:** Connected frontend to backend for fetching product data.
- **Added Filters & Pagination:** Implemented search, category filters, and pagination for better user experience.

Challenges & Solutions

- **Challenge:** Handling dynamic data fetching delays.
 - **Solution:** Implemented loading states and error handling.
- **Challenge:** Ensuring proper routing for individual products.
 - **Solution:** Used Next.js useRouter for dynamic paths.
- **Challenge:** Styling the components for responsiveness.
 - **Solution:** Used Tailwind CSS for a responsive layout.

Best Practices Followed

- **Component Reusability:** Broke down UI into modular and reusable components.
 - **State Management:** Used useState and API calls efficiently.
 - **Error Handling:** Implemented error handling for API requests.
 - **Performance Optimization:** Used lazy loading for images and optimized rendering.
-

Submission Format

- **Document Title:** "Day 4 - Dynamic Frontend Components - EcoFurnish"
 - **Format:** PDF or Markdown
 - **Contents:**
 1. Screenshots/Recordings
 2. Code Snippets
 3. Technical Documentation
-