Prepared by Kulsoom Adnan

Date : 20-January -2025

HACKATHON DAY 04

Dynamic Frontend Components

Introduction:

In this documentation, we present the dynamic frontend components developed **Furniro** as part of the hackathon project. The aim was to implement various dynamic functionalities that enhance the user experience by providing real-time data, smooth navigation, and interaction with the marketplace platform. The project was built using **Next.js, Tailwind CSS, and integrated with Sanity CMS for content management.**

On Day 4, the focus was on the development of core dynamic frontend components, including the **product listing page, product detail pages, cart, wishlist functionality, pagination, related products section, and search bar.** These components were designed to be **responsive, user-friendly, and seamlessly** integrated into the marketplace environment.

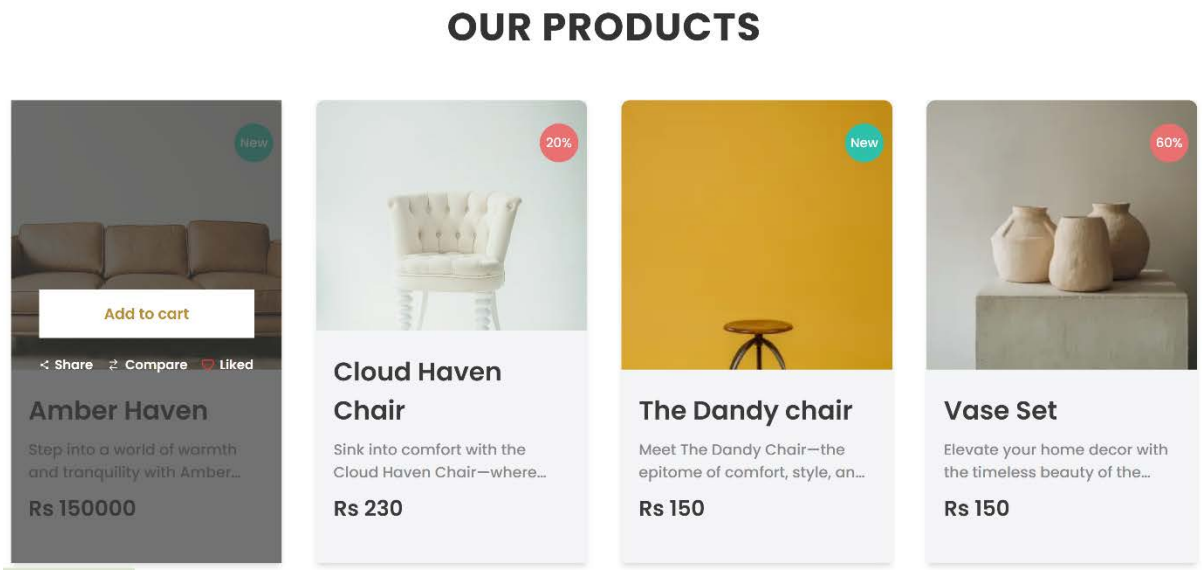
The following sections will outline the key components developed, the functionality behind them, and the corresponding code implementations.

Functional Deliverable:

1. Product Listing Page with Dynamic Data:

The **Product Listing Page** dynamically renders the products available in the marketplace. It is built to fetch data from Sanity CMS and display it in a grid format. The page includes essential product details such as the product name, price, anddescription. This dynamic rendering ensures that when new products are added or existing products are updated in the database, the changes are reflected instantly on the frontend without needing to reload the page.

Page Overview:



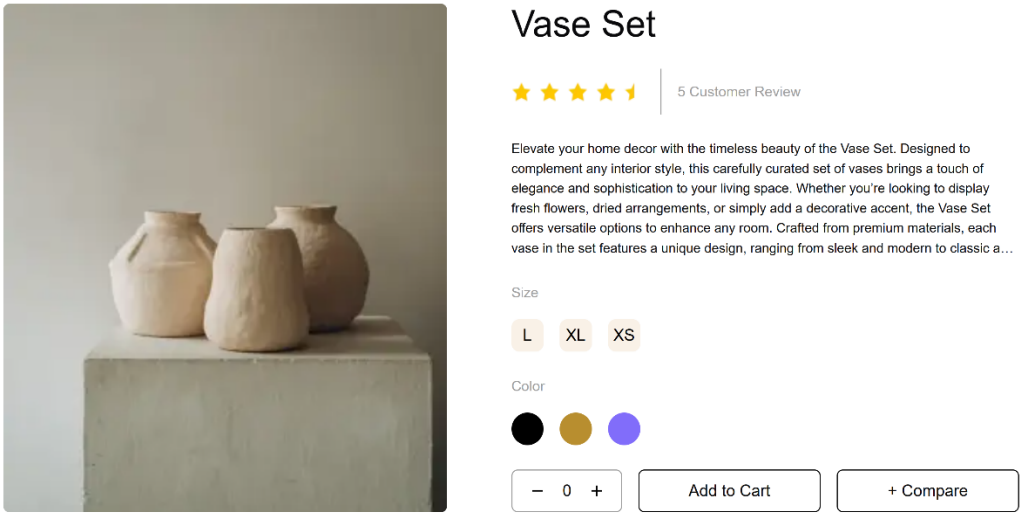
Key Features:

- The page is responsive, adjusting automatically for mobile and desktop views.
- Products are displayed as individual cards with hover effects for interaction.
- Dynamic rendering of products based on data fetched from the backend.

2. Individual Product Detail Pages:

Each product card links to an individual **Product Detail Page**. This page showcases more comprehensive product information, including description, available sizes/colors, and the option to add the product to the cart.

Page Overview :



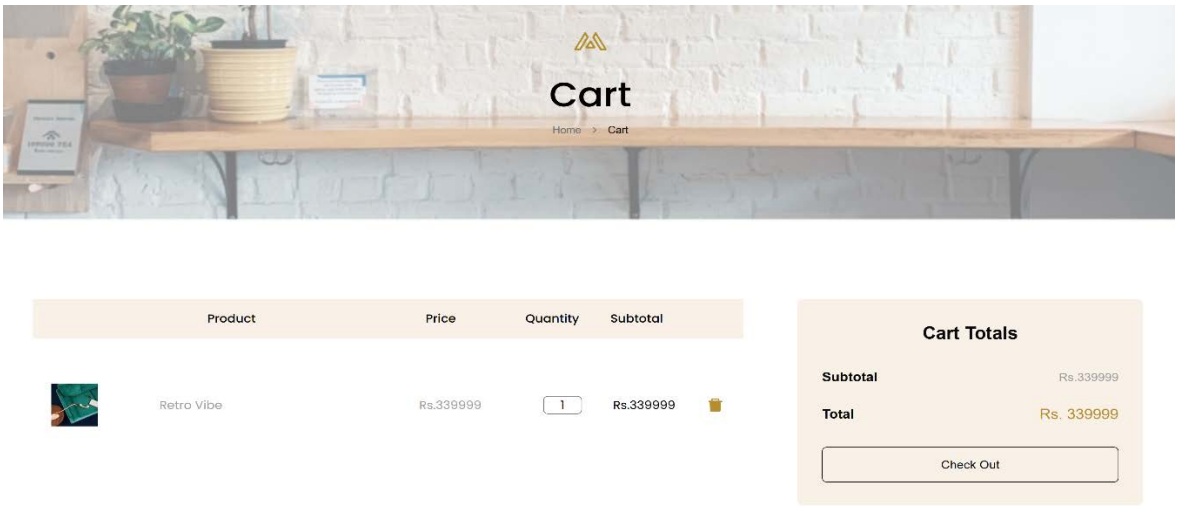
Key Features:

- Dynamic routing based on product ID.
- Real-time product details fetched from an API.
- Option to add products to the cart directly from the page.

3. Add to Cart Page

The **Add to Cart Page** provides an overview of the items the user has added to their cart. It shows product details such as the product name, price, quantity, and the total price. Users can modify the quantities of items, remove products, or proceed to checkout from this page.

Page Overview:



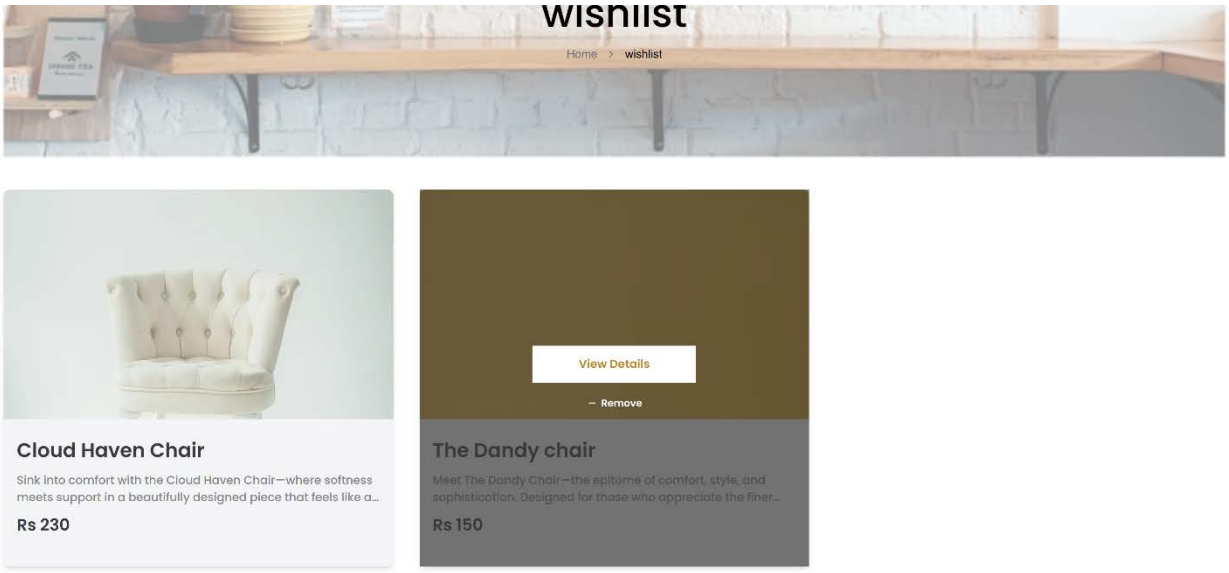
Key Features:

- Users can update the quantity of products or remove them entirely.
- A summary of the cart total is displayed.
- Implemented using React's state management to update the UI in real-time as the cart changes.

4. Wishlist Page

The **Wishlist Page** allows users to save their favorite products for future purchase. This page dynamically loads saved items from the **local storage** or the global state, and users can move items from the **wishlist** to the shopping cart.

Page Overview:



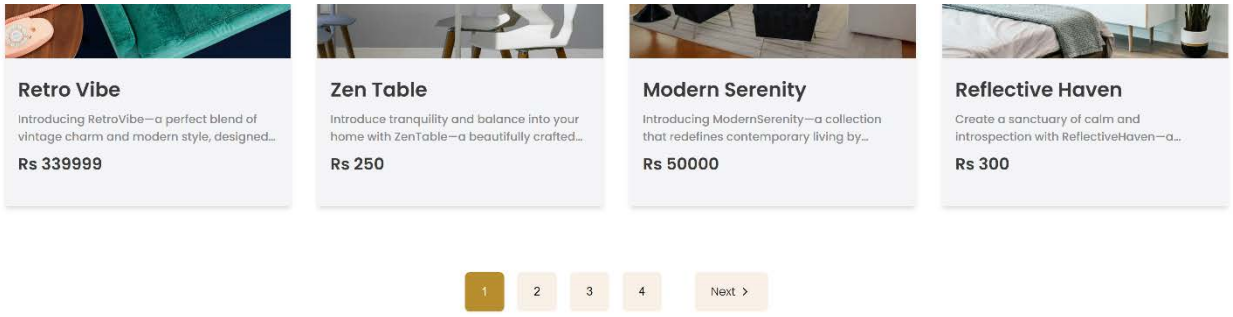
Key Features:

- Users can add and remove items from the wishlist.
- User can navigate to the **Product details** page by clining View Details Button
- Wishlist data is persisted using local storage or React context API.
- A seamless transition to the cart for easy checkout.

5. Pagination Component

For handling large product datasets, the Pagination Component was developed. This component allows users to navigate through pages of products, displaying a subset of the entire catalog at a time to improve performance and usability.

Overview:



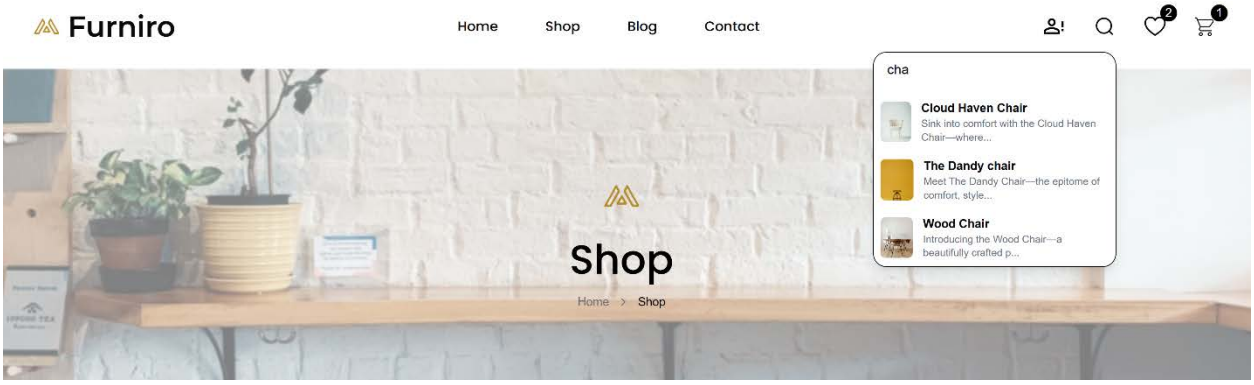
Key Features:

- Paginated product listings to optimize load times and improve the browsing experience.
- Displayed either as page numbers or next/previous buttons.
- The component dynamically updates the list of products shown based on the user's page selection.

6. Search Bar

The Search Bar component allows users to quickly find products by searching for names, categories, or tags. The search results update dynamically as the user types, thanks to real-time filtering.

Overview:



7. Key Features:

- **Real-time search functionality using React hooks.**
- **Filters products based on user input.**
- **Utilizes both client-side filtering and server-side queries for better performance.**

Header Component

The Header Component is a vital part of the marketplace, providing navigation links to important sections of the site, including the homepage, product categories, cart, and user profile. It ensures that users can easily navigate through the website at any time.

Key Features:

 Logo and Navigation: Displays the site logo, home button, and links to product categories and other pages.

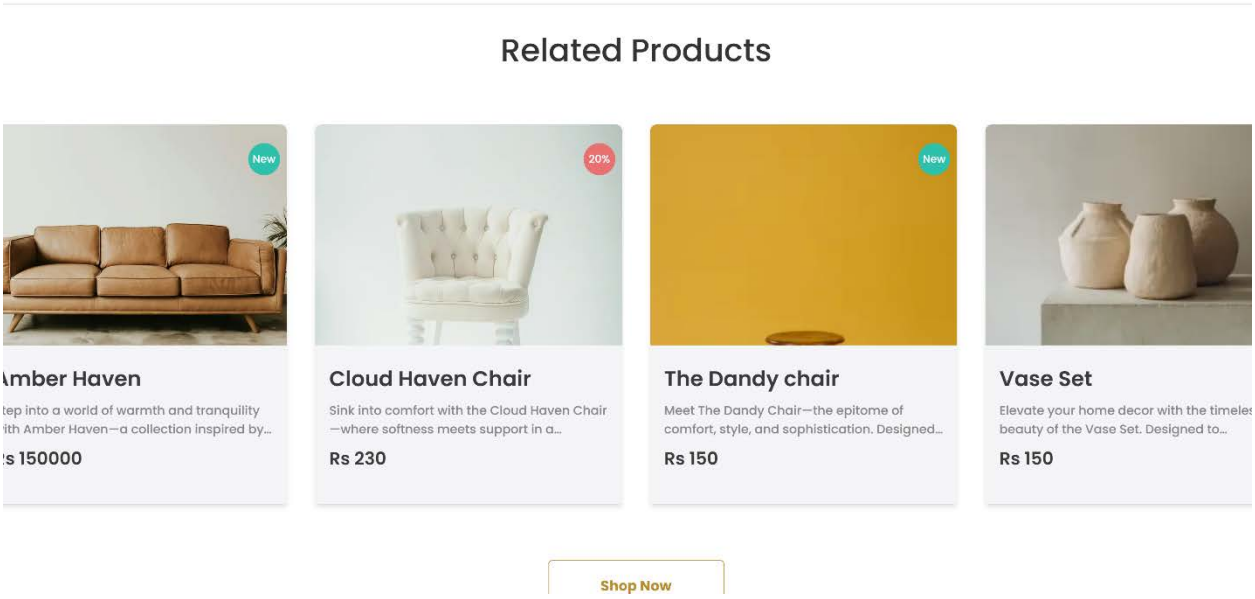
Search Bar: A search bar integrated in the header allows users to search for products quickly without navigating away from the page.

 Cart Icon: Displays the number of items currently in the cart, allowing quick access to the cart page.

8. Related Products Component

The Related Products Component is integrated into the product detail page, displaying products that are similar or complementary to the current product. This feature is designed to increase user engagement by encouraging cross-selling and upselling.

Screenshot :



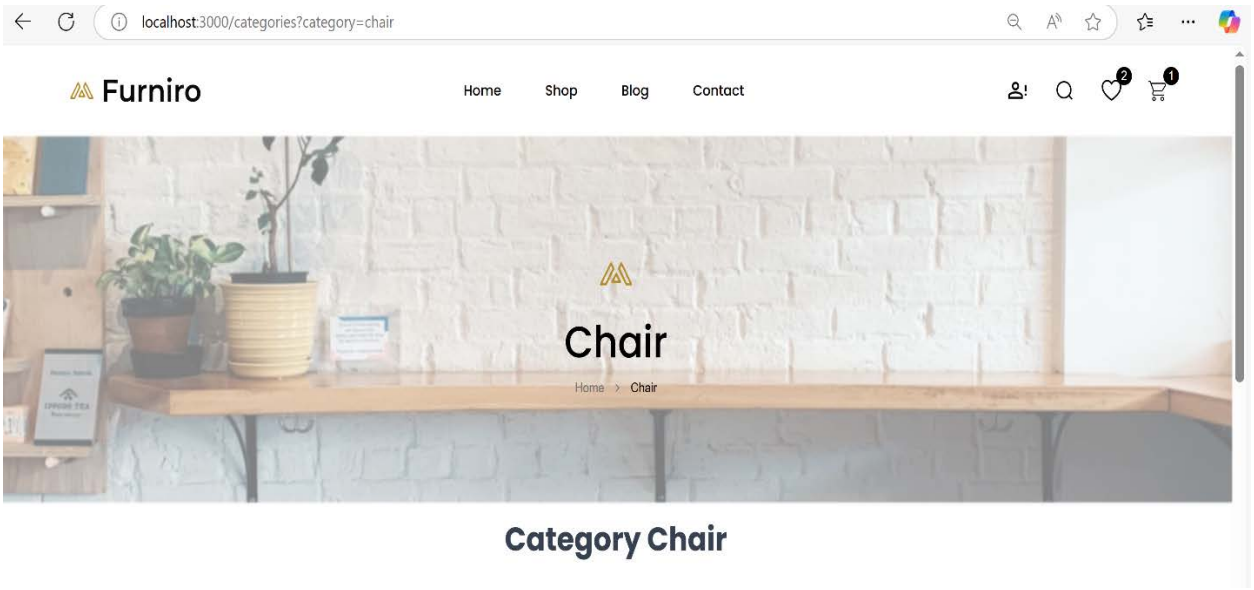
Key Features:

- Related products are displayed dynamically based on categories, tags, or purchase history.
- Fetches related product data from the API or CMS.
- Provides an interactive way for users to explore more products.

9. Category Page

The Category Page serves as a dynamic filter for the products, grouping them by predefined categories (e.g., sofas, chairs, tables). When a user selects a category image or link, they are taken to this page, which displays all the products related to that category.

Overview:



Key Features:

- Dynamic routing to generate category-specific pages.
- Fetches category-specific product data from the backend API.
- Displays product cards filtered by category, with options to sort or filter further.
- Integrates a breadcrumb navigation to help users track their location within the site.

Code Deliverables

The following are the key code snippets from the dynamic components developed:

ProductCard Component:

ProductList Component:

```tsx
"use client";
import Filter from "@/components/Filter";
import RouteHero from "@/components/RouteHero";
import Services from "@/components/Services";
import React, { useEffect, useState } from "react";
import Skeleton from "@/components/Skeleton";
import ProductCard from "@/components/ProductCard";
import PaginationUi from "@/components/Pagination";
import { ProductInterface } from "@/components/Types";

function page() {
  const [products, setProducts] = useState<ProductInterface[]>([]); // State to store products
  const [isLoading, setIsLoading] = useState(true); // State for loading indicator
  const [currentPage, setCurrentPage] = useState(1); // State for current page
  const itemsPerPage = 8; // Number of products to display per page

  useEffect(() => {
    async function getData() {
      try {
        let response = await fetch("http://localhost:3000/api/products");
        const data = await response.json();
        if (data.success) {
          setProducts(data.products);
        } else {
          console.error("Failed to fetch products:", data.message);
        }
      } catch (error) {
        console.error("Error fetching products:", error);
      } finally {
        setIsLoading(false); // Stop loading
      }
    }

    getData();
  }, []); // Empty dependency array ensures this runs once when the component mounts

  // Pagination Logic
  const indexOfLastProduct = currentPage * itemsPerPage;
  const indexOfFirstProduct = indexOfLastProduct - itemsPerPage;
  const currentProducts = products?.slice(
    indexOfFirstProduct,
    indexOfLastProduct
  );

  // Change page
  const paginate = (pageNumber: number) => setCurrentPage(pageNumber);

  if (isLoading) {
    return <Skeleton />;
  }
  console.log(products);
  return (
    <div className="h-auto overflow-hidden">
      <RouteHero prop="Shop" />
      <Filter />

      {/* Product Grid */}
      <div className="w-full py-20 px-10">
        <ul className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-8">
          {currentProducts &&
            currentProducts.map((product) => (
              <ProductCard key={product._id} product={product} />
            ))}
        </ul>
      </div>
      <div className="pb-10">
        <PaginationUi
          totalProducts={products?.length}
          productsPerPage={itemsPerPage}
          paginate={paginate}
          currentPage={currentPage}
        />
      </div>
      <Services />
    </div>
  );
}

export default page;

```

Challenges and Solutions

During the development of these components, several challenges were encountered:

Challenge 1: Handling Large Data Sets

When rendering large product catalogs, performance can suffer. To mitigate this, I implemented **pagination** and **lazy loading**, which allowed the marketplace to only load the data that was needed at any given time, resulting in faster load times and smoother performance.

Challenge 2: State Management Across Components

Managing state across multiple components, such as the cart and wishlist, required careful planning. I utilized the **React Context API** for global state management to ensure that the cart and wishlist were updated in real-time and persisted across different pages.

Challenge 3: Categorizing Products

Displaying products by category required careful filtering and dynamic routing. I ensured that the **Category Page** was implemented to fetch products dynamically based on the category selected by the user.

Conclusion :

By Day 4 of the hackathon, I have successfully implemented several key dynamic components for the marketplace project. These components, including the product listing page, product detail pages, category page, cart, and wishlist functionality, significantly enhance the user experience by providing real-time updates and easy navigation.

Self-Validation Checklist:

- **Frontend Component Development**:
  ✓ Completed product listing page, product detail pages, category page, and cart functionality.
  ✗ Yet to implement advanced features like product reviews and ratings.
- **Styling and Responsiveness**:
  ✓ All pages and components are responsive using Tailwind CSS.
  ✗ Testing responsiveness on various devices is ongoing.
- **Code Quality**:
  ✓ Code is well-structured, modular, and reusable across components.
  ✗ Code documentation is still being refined.
- **Documentation and Submission**:
  ✓ Functional components are documented with proper explanations.
  ✗ Final submission for Day 4 is pending.

Final Review:

- **Frontend Components**: ✓ All major components were developed and are functional, including the addition of the category page and header component.
- **User Experience**: ✓ Smooth navigation, responsive design, and real-time updates have been implemented.
- **Performance**: ✓ Pagination and lazy loading ensure efficient handling of large product catalogs.
- **Areas to Improve**: ✗ Final testing on mobile devices for responsiveness and load times.