Marketplace Builder Hackathon 2025

Day 4- Dynamic Frontend Components – “Furniro”

TABLE OF CONTENTS

Section	Subsections
1. Overview	Summary of the Marketplace Project
	Key Features Implemented
2. Component Development and Features	Navbar.tsx
	Footer.tsx
	Hero.tsx
	Products.tsx
	ProductDetails.tsx
3. Add to Cart Functionality	Context and State Management
	Adding Products to Cart
	Calculating Total Price
	Accessing Cart Data
	Workflow of Adding a Product to the Cart
	Individual Product Page
	Cart Context
	Cart Display
	Code Example
4. Challenges Faced	Using the useState Hook
	Accessing Images from Sanity
	Minor Issues
5. Conclusion	Reflections on Overcoming Challenges
	Lessons Learned

Overview

This document summarizes the steps taken to implement key features for a marketplace project, assigned to us on Day 4 of Marketplace builder hackathon, including the product listing page, individual product detail pages, Add to cart functionality. The project involved dynamic API integration, state management, and responsive UI development.

Component Development and Features:

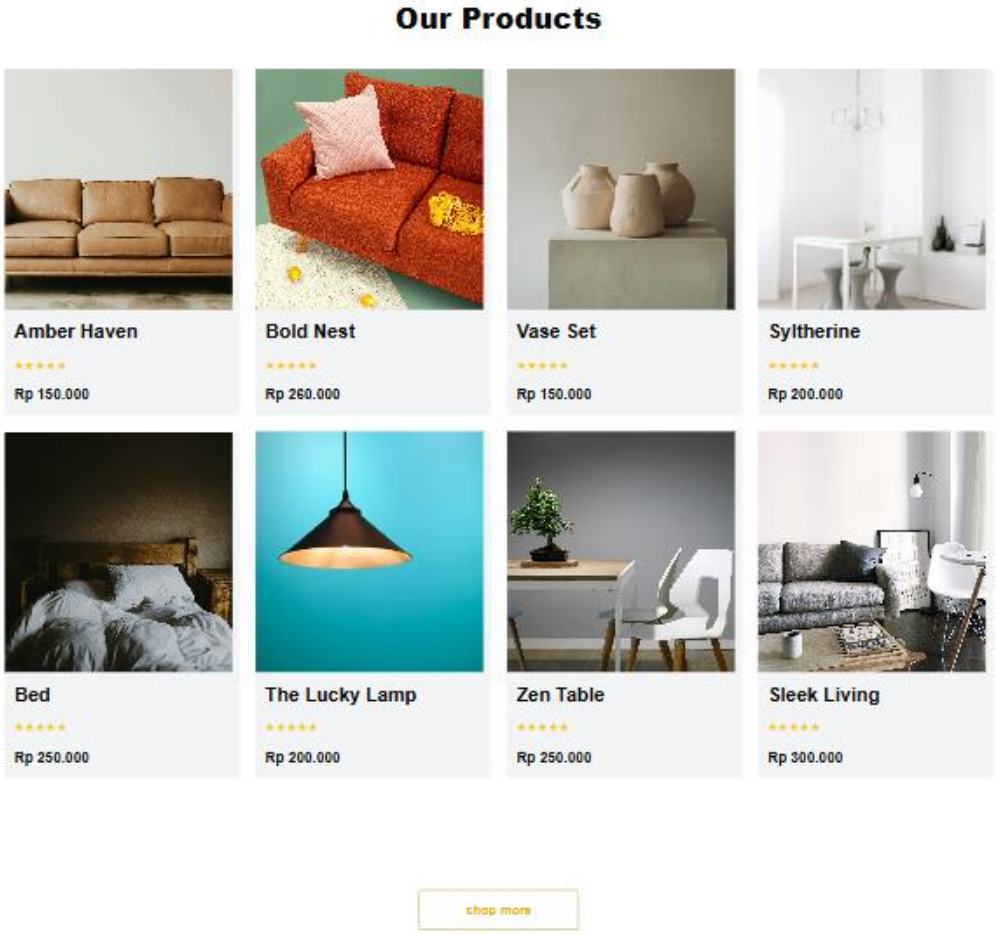
1) **Navbar.tsx :**
   I created the Navbar component and implemented it in the Layout.tsx file so as to have access to it throughout my project. It consists of all the dynamic routes and important pages like “shop” , “Contact” , “Cart” and “Blog”. It have a responsive design according to different screens breakpoints and is styed through Tailwind CSS.

2) **Footer.tsx :**
   The footer Is also a component which is used in layout file so it can be accessed from anywhere in the project and is shown in all routes. It have a responsive design according to different screens breakpoints and is styed through Tailwind CSS.

3) **Hero.tsx:**
   It’s the top most component shown on the main page or landing age of the application.

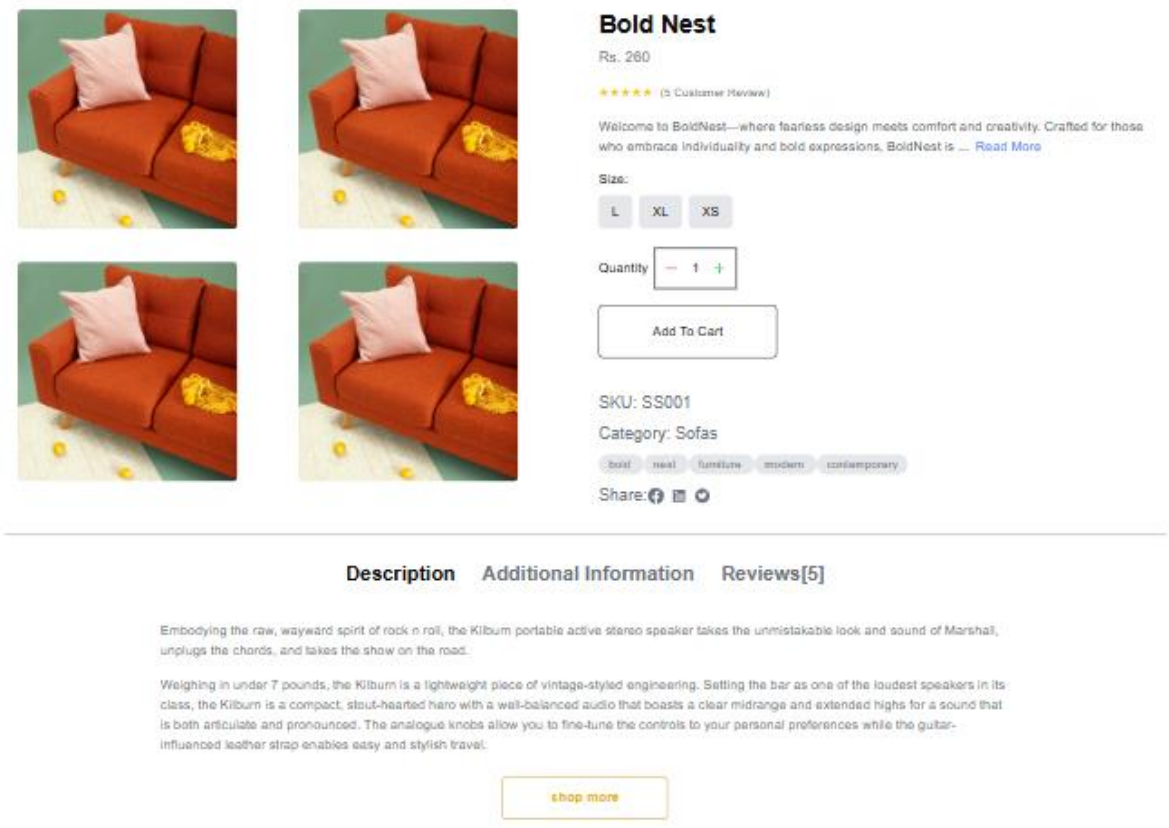
4) **Products.tsx:**

The product component consists of the products list fetched via query from sanity through an external API provided to us . It have a proper responsive grid styling and may be used on several pages on my application. By pressing a product it redirects us to the it's on individual page consisting detail about that specific product.



5) **ProductDetails.tsx:**

Each product detail page is dynamically routed and fetches data based on the product's unique ID. It consists of an ADD TO CART button which allows us to add an item to the cart along with the specified quantity which we can choose by the counter. The read more option allows to expand the description about that product which is all fetch via API.



ADD TO CART FUNCTIONALITY:

In my project, I implemented the "Add to Cart" functionality using React's Context API, which enabled seamless state management and sharing of cart-related data across the application. To achieve this functionality, I created a separate file named CartContext.tsx where I encapsulated all the necessary logic for managing the cart.

Context and State Management

I used the createContext hooks to define and access the cart context. Additionally, I utilized the useState hook to maintain the state of the cart. This approach allowed me to keep track of the products added to the cart, along with their respective quantities, prices, and the total price of all products combined.

The CartContext file included the following key functionalities:

1. **Adding Products to Cart**:
    - When a user clicks the "Add to Cart" button on an individual product details page, the product's name, quantity, and price are added to the cart.
    - I ensured that if the product already exists in the cart, the quantity is updated instead of adding a duplicate entry.
2. **Calculating Total Price**:
    - I implemented a function that dynamically calculates the total price of all the products in the cart whenever an item is added, removed, or its quantity is updated.
3. **Accessing Cart Data**:
    - The cart data, including product details and the total price, is made accessible throughout the application using the useContext hook.

Workflow of Adding a Product to the Cart

1. **Individual Product Page**:
    - Each product's details page includes an "Add to Cart" button. Clicking this button triggers a function that adds the selected product to the cart.
2. **Cart Context**:
    - The function in the CartContext handles the logic for updating the cart state. It checks if the product already exists and updates its quantity or adds a new entry if it doesn’t.

3. **Cart Display**:
    - The cart page or component retrieves the cart data using the useContext hook and displays the product details, quantities, and total price.

Below is a simplified example of how I structured the context:

```
"use client"
import { Product } from "@/types/product";
import { createContext, useState} from "react";

export const CartContext = createContext({});

// Create the provider
Codeium: Refactor | Explain | ×
export const CartProvider = ({ children }:any) => {

  const [qty, setQty] = useState<number>(1);
 const [cartItems , setCartItems] = useState<any[]>([]);
 const[totalQuantity,setTotalQuantity] = useState(0);
const [totalPrice , setTotalPrice] = useState(0)
 Codeium: Refactor | Explain | Generate JSDoc | ×
 const incQty = () => {···
 }
 Codeium: Refactor | Explain | Generate JSDoc | ×
 const decQty = () => {···
 }

 Codeium: Refactor | Explain | Generate JSDoc | ×
 const addProduct = (product: Product, quantity: number) => {···
  };

  Codeium: Refactor | Explain | Generate JSDoc | ×
  const onRemove = (product:Product)=>{···
    }

  return (
    <CartContext.Provider value={{onRemove, qty , incQty , decQty , cartItems, addProduct , totalQuantity , totalPrice}}>
      <div>{children}</div>
    </CartContext.Provider>
  );
};
```

By structuring the "Add to Cart" functionality in this way, I achieved a scalable and reusable solution for managing cart-related state in my project.

Challenges Faced:

While implementing dynamic routing and functionality in this project, I encountered several challenges:

1. **Using the useState Hook**:
    - Initially, I found it difficult to implement the useState hook effectively for managing dynamic data such as cart items and their states. Debugging and understanding the reactivity of state updates required careful attention and practice.
2. **Accessing Images from Sanity**:
    - Fetching and displaying images stored in the Sanity CMS posed challenges due to formatting and URL generation issues. After researching the Sanity documentation and experimenting with their APIs, I was able to resolve this problem.
3. **Minor Issues**:
    - I faced several smaller issues, such as handling edge cases in cart functionality and ensuring proper routing for dynamic product details pages. These issues required persistence and multiple iterations to debug and fix.

Despite these challenges, I was able to overcome them with strong determination and support from my peers. Collaborating with others and seeking guidance helped me learn and apply effective solutions to improve the functionality and user experience of the project.