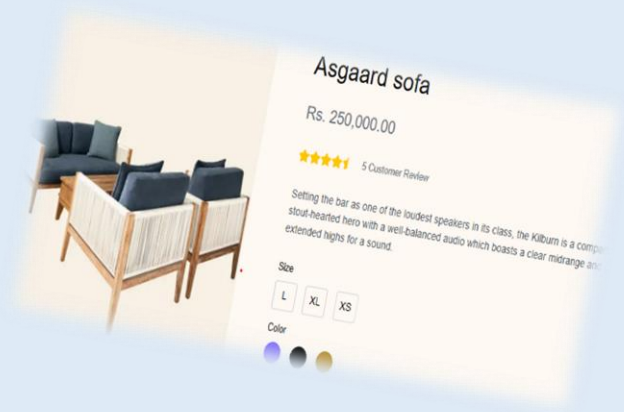
Day 4 - Dynamic Frontend Components General E-commerce Website

Prepared By: Saira Rizvi

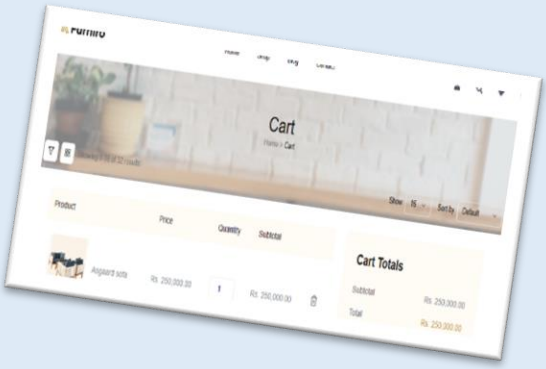
1) **OPERATIONAL DELIVERY**

- Functional Product page

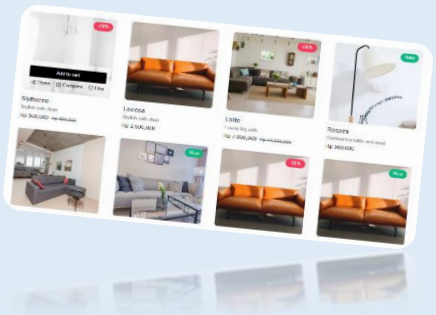


- Functional Blog Page

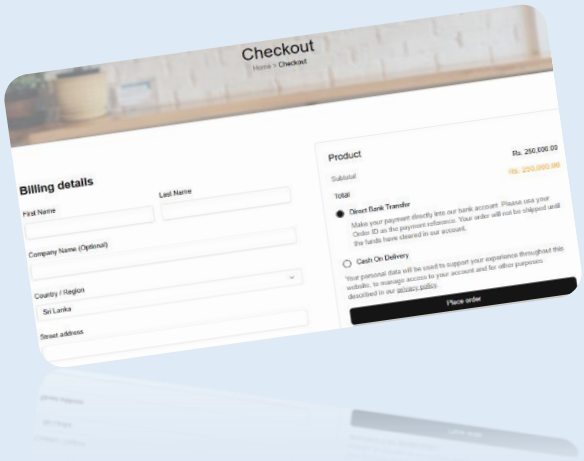
- Cart Page with Functionality



- Price Filter And pagination



- Checkout Page with Functionality



- **Code Deliverables**

- Code snippet for components

a) Shop page “These pages show frontend display with the help of code”

- Product Card

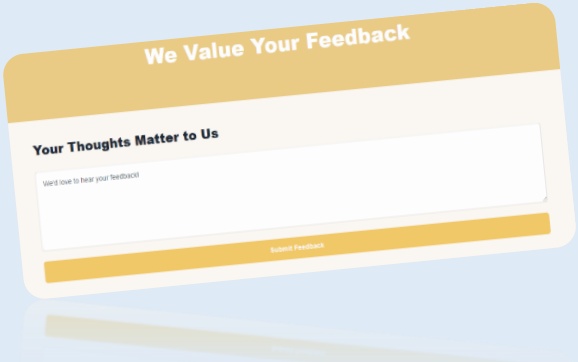
b) “This component displays individual product details and includes the "Add to Cart" button”.

- Product List

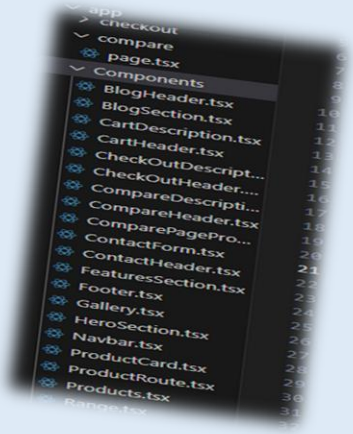
C) “Fetches and Display a list of products”.

c) Feedback Page

“This page allows user to share valuable thoughts and their feedback.”



d) Additional components



➢ Project Overview

✓ Key Components

Component	Description
AddToCartButton.tsx	Handles adding products to the cart using Cart.
ComparisonContext.tsx	Stores and manages product comparison data.
footer.tsx	Displays the website footer with links and relevant information.
hero.tsx	The main banner or hero section for the homepage.
navbar.tsx	The navigation bar containing links and the cart icon.
ProductCard.tsx	Displays individual product details in a card format.
ProductClient.tsx	Manages product-related API calls or client-side logic.
ProductList.tsx	Dynamically fetches and displays a list of products.
ShopClient.tsx	Handles the shop page functionality and client-side logic.

Scripts and Logic for API Integration & Dynamic Routing

- ✓ API Integration
- ✓ API Endpoint Used: https://template6-six.vercel.app/api/products

Key Steps:

- **Fetching Data:** Products are fetched using fetch or axios in ProductList.

Error Handling: Added try-catch blocks for smooth error handling during API requests

---

➢ Technical Report

✓ Steps Taken to Build and Integrate Components

1. **Project Setup:**
   - Initialized a Next.js project and configured the required dependencies.
2. **Component Structure:**
   - Built reusable components like ProductCard.tsx,Navbar.tsx, and Footer.tsx.
3. **API Integration**
   - Fetched product data from the API endpoint and dynamically displayed it.

4. **UI Enhancements:**
    - Designed a responsive and visually appealing UI using **Tailwind CSS**.
5. **Testing & Optimization:**
    - Debugged API calls, optimized rendering, and added error handling for better user experience.

Challenges Faced and Solutions Implemented	
Challenge	**Solution**
API data was not displaying properly.	Used use Effect to fetch data on mount and added error handling.
Pagination logic wasn't updating.	Fixed state updates with use State and ensured proper current Page handling.
Dynamic routes were not functioning.	Used Next.js use Router to extract parameters and fetch data dynamically.

➢ Best Practices Followed

✅ **Component Reusability**

- Created modular, reusable components (e.g., Hero,tsx, Navbar.tsx, Footer.tsx) for maintainability.

✅ **Optimized API Calls**

- Implemented efficient fetching with try-catch error handling for smooth data retrieval.

✅ **Code Maintainability**

- Followed a clean project structure (components/ui/, lib/, etc.) for organized code.

✅ **Performance Optimization**

- Used lazy loading and memorization to improve page load times.

🚀 **Summary**

This project demonstrates a scalable and professional e-commerce solution using **Next.js**. It combines efficient API integration, state management, reusable components, and dynamic routing to deliver a seamless user experience. The use of best practices like responsive design, modular architecture, and performance optimizations ensures that the project is robust and maintainable. 🌟