Dynamic Frontend Components for Furniture E-Commerce Marketplace

Introduction:

In this project, I'm building a dynamic and responsive marketplace using Next.js and Tailwind CSS. The backend is powered by Sanity CMS, which serves as the data source for our product listings, categories, and other details. The goal is to create a seamless user experience with features like dynamic product listings, individual product detail pages, advanced category component, a search bar, pagination, and more. This document outlines the steps taken, challenges faced, and solutions implemented during the development process.

Step 1: Setup

Achievements:

- Successfully connected the Next.js project to Sanity CMS.
- Configured the API endpoints to fetch data from Sanity.
- Tested data fetching to ensure the availability of product data.
- Verified that the data is being fetched and logged in the console for debugging purposes.

Key Points:

- Used GROQ (Sanity's query language) to fetch product data.
- Integrated Sanity's client library (@sanity/client) for seamless data fetching.
- Ensured environment variables are securely stored using .env.local.

Step 2: Building Components

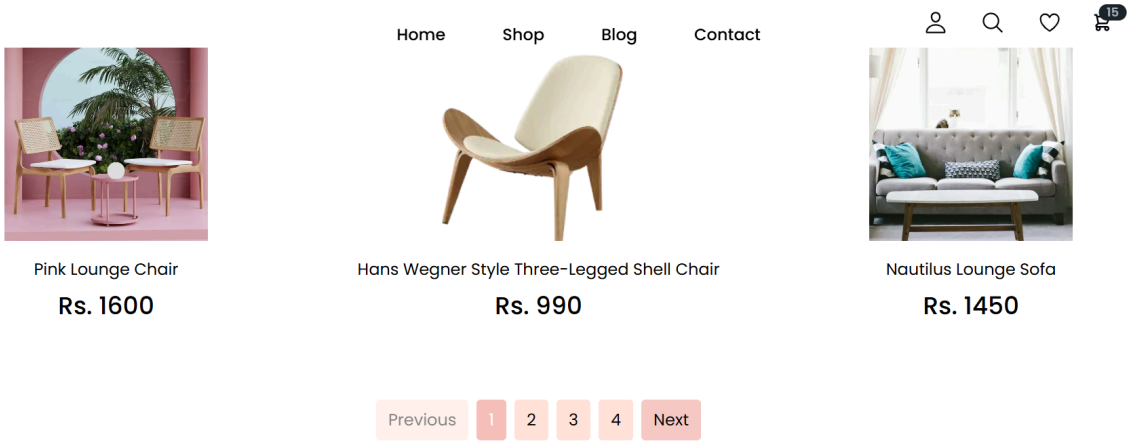
1. Product Listing Page

Description:
Displays all products dynamically fetched from Sanity CMS.

Key Features:

- Responsive design using Tailwind CSS.
- Clean and professional interface.
- Each product is displayed in a ProductCard component with an image, title and price.
- Pagination implemented to handle large datasets.

Overview:



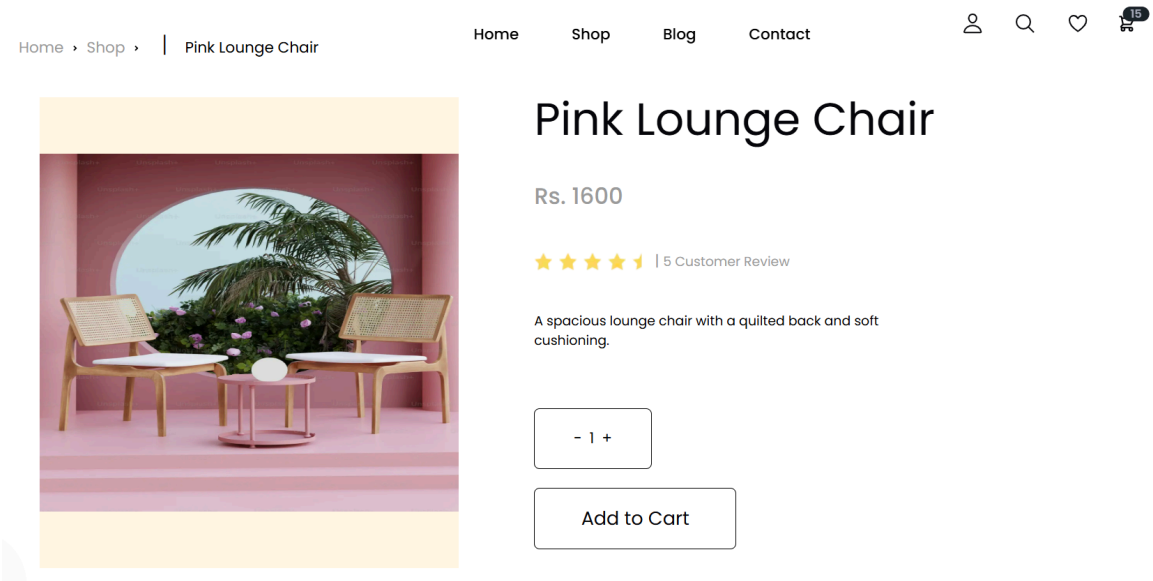
2. Product Detail Page

Description:
Individual pages for each product, accessed via dynamic routing.

Key Features:

- Dynamic routing using Next.js.
- Displays detailed product information, including images, description, price, add to cart button and more.

Overview:



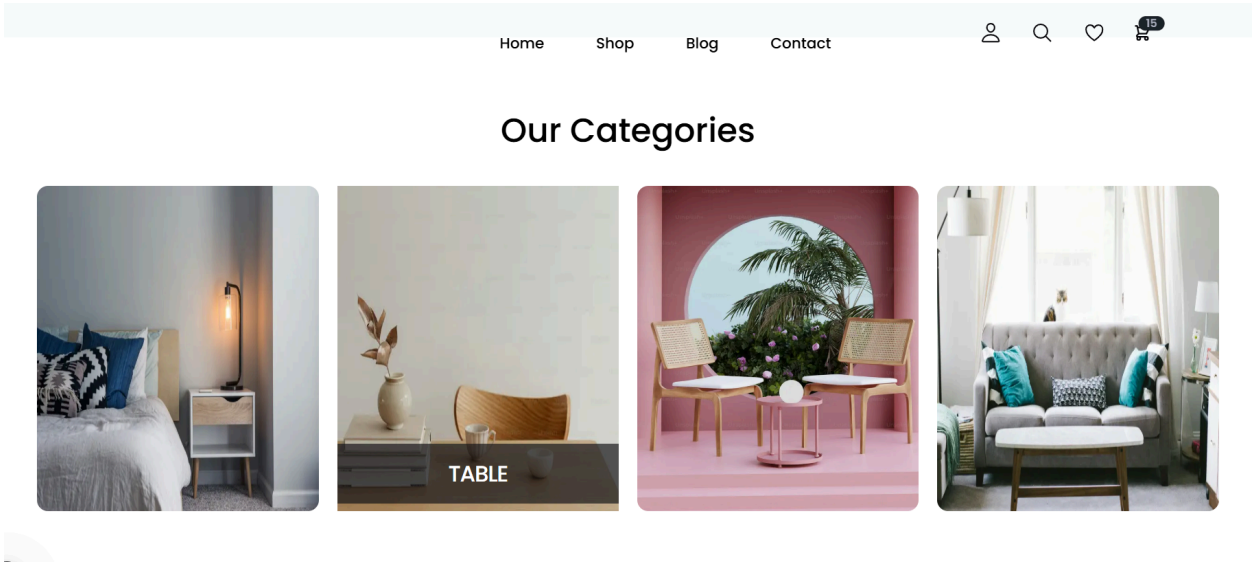
3. Category Component

Description:
The Category Component is designed to filter and display products based on the selected category. When a user clicks on a specific category (e.g., Chairs, Tables, Beds, Sofas), the component dynamically fetches and displays all products related to that category. This ensures a seamless and organized browsing experience for users.

Key Features:

- Dynamic filtering based on selected categories.
- Real-time updates to the product list without page reload.
- Categories fetched from Sanity CMS.
- The category cards and filtered product list are fully responsive, ensuring a smooth experience across all devices (desktop, tablet, mobile).
- Visual feedback (e.g., hover effects, active state) to indicate the selected category.

Overview:



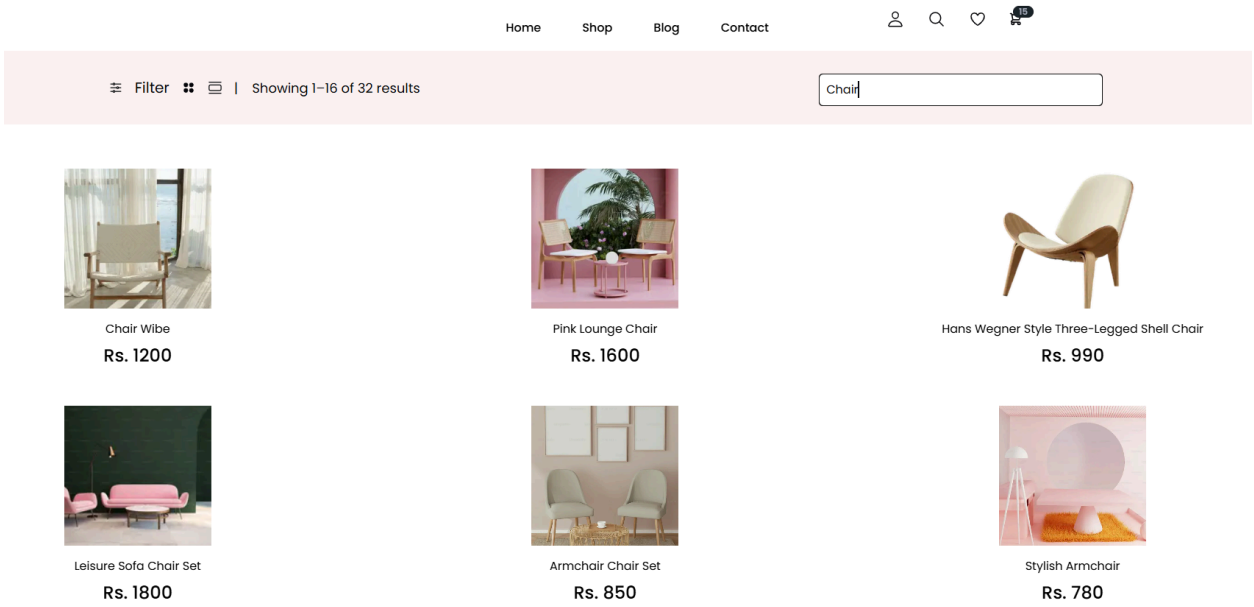
4. Search Bar

Description:
Enables users to search for products by name or categories.

Key Features:

- Real-time filtering of products as the user types.
- Case-insensitive search functionality.
- Highlights matching results.

Overview:



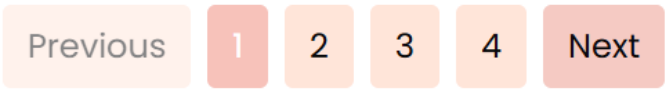
5. Pagination

Description:
Splits product listings into multiple pages for better usability.

Key Features:

- Dynamic page numbers based on the total number of products.
- Smooth navigation between pages.

Overview:



6. Header and Footer Components

Description:

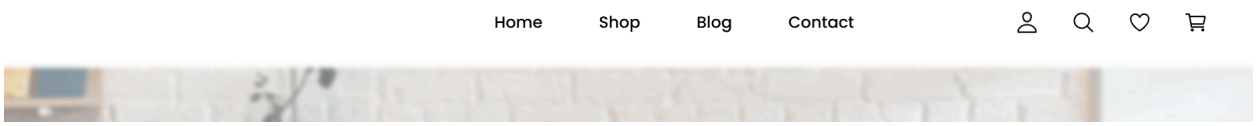
Reusable components for consistent navigation and branding.

Key Features:

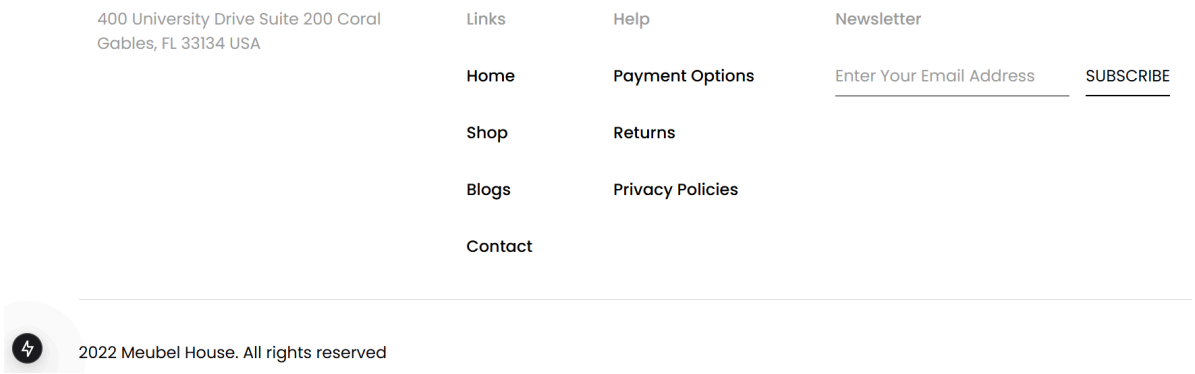
- Header includes navigation links, the search bar, the cart and wishlist icons.
- Footer contains links to important pages and social media icons.
- Fully responsive design.

Overview:

Header:



Footer:



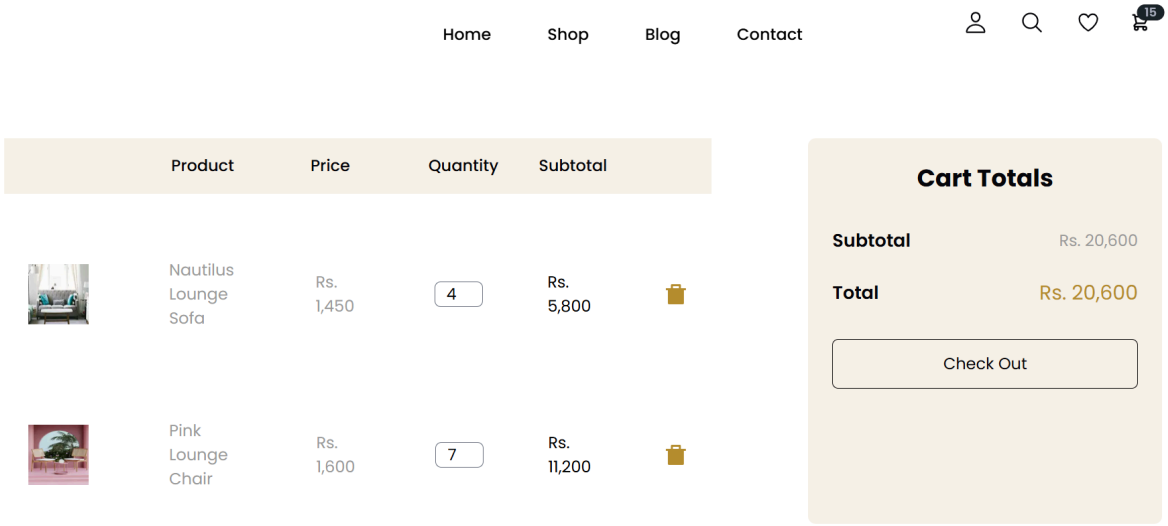
7. Add to Cart Functionality

Description:

Allows users to add, update, and remove products from the cart.

Key Features:

- Interactive popups to confirm product addition or removal.
- Cart state managed using React Context API.
- Real-time updates to the cart icon and total items.



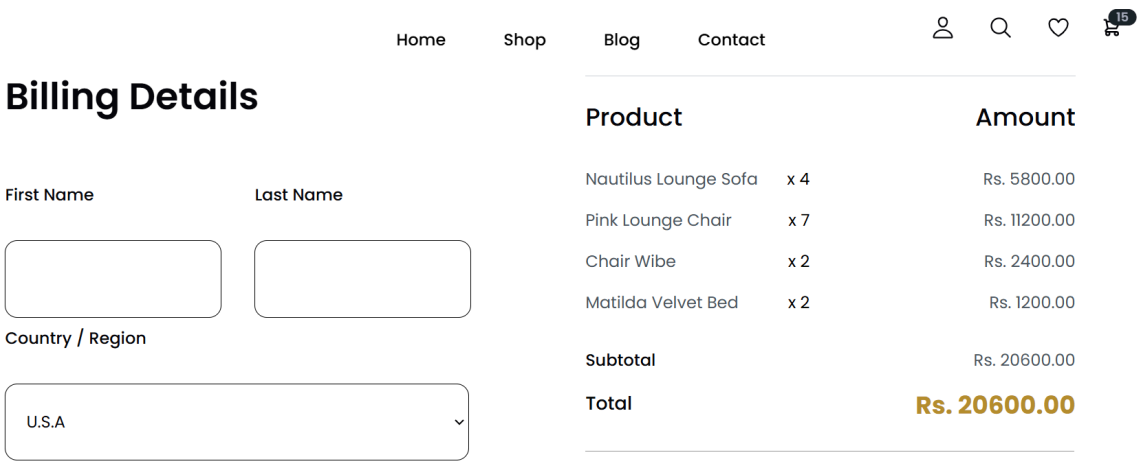
8. Checkout Page

Description:
Displays the final cart items and allows users to proceed to payment.

Key Features:

- Dynamic rendering of cart items.
- Interactive popups for confirming actions.
- Responsive layout for seamless checkout on all devices.

Overview:



9. Notification Component:

Description:

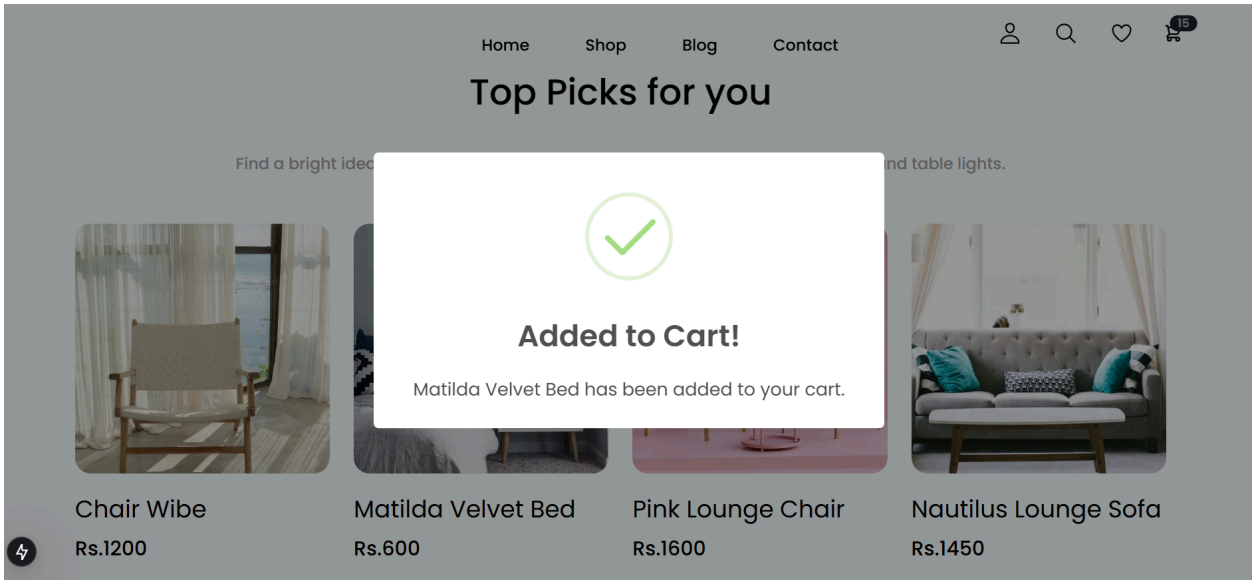
The Notification Component provides real-time feedback using SweetAlert2 for actions like adding/removing items, checkout, and error handling. Its visually appealing popups enhance user experience with clarity and interactivity.

Key Features:

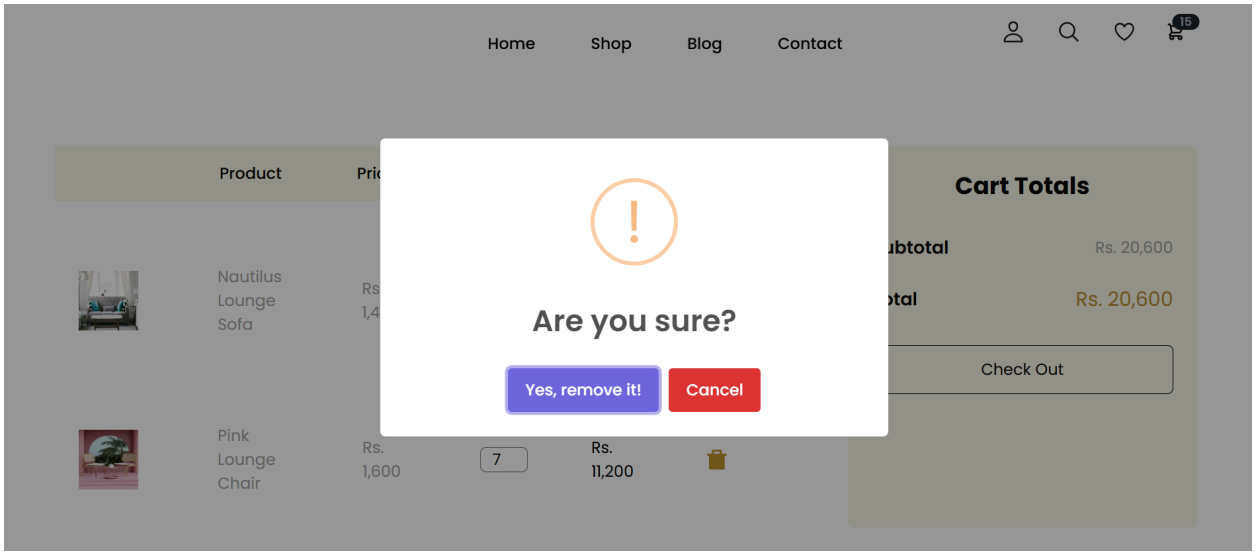
- Real-time feedback for actions like adding/removing items and checkout.
- Customizable popups using SweetAlert2 for a visually appealing design.
- Error handling with clear messages and retry options.
- User-friendly interactions with confirmation prompts and auto-close timers.

Overview:

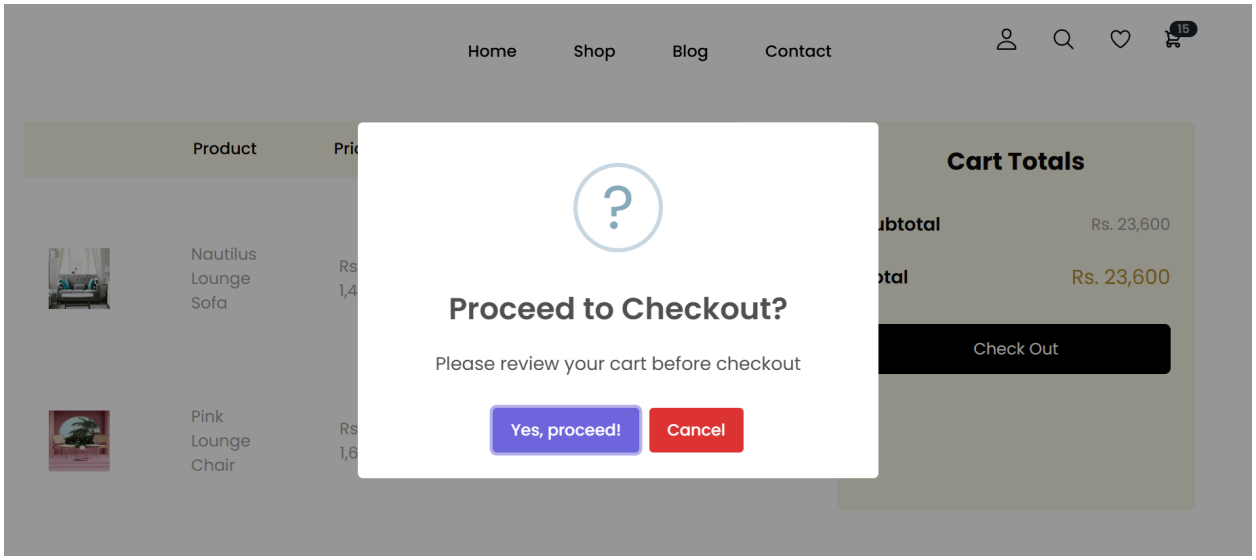
Add to Cart:



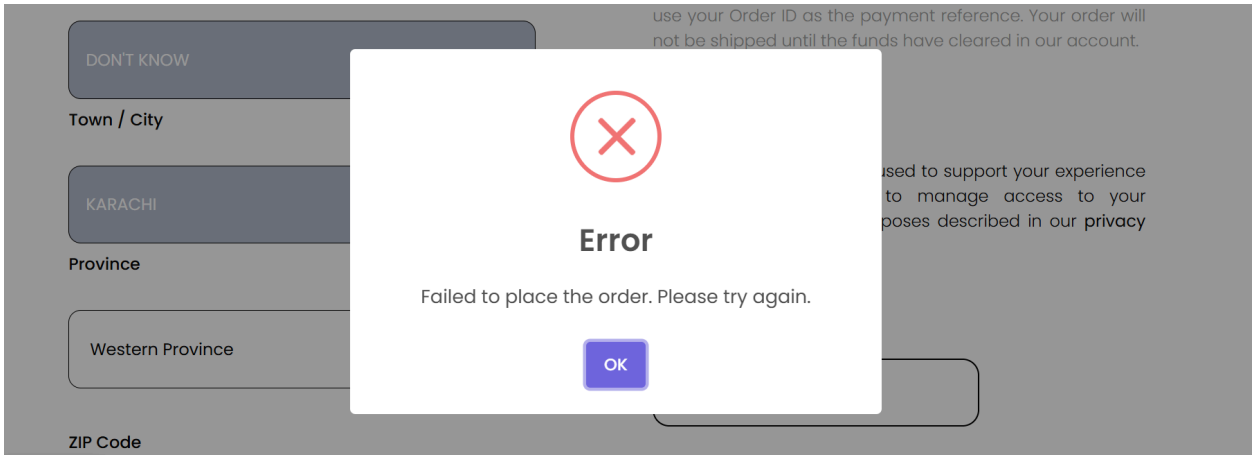
Remove item from Cart:



Proceed to Checkout:



Error:



Code Deliverables:

To showcase the implementation of key components and functionality, below are code snippets and logic for ProductList, ProductDetail, and dynamic routing. These examples demonstrate how the components are built and integrated into the marketplace.

Product Detail:

```
'use client'

import React, { useState, useEffect } from 'react';
import { RiArrowDropRightLine } from 'react-icons/ri';
import Image from 'next/image';
import { TbMoodXFilled } from 'react-icons/tb';
import Link from 'next/link';
import Swal from 'sweetalert2'
import {
  Sheet,
  SheetContent,
  SheetDescription,
  SheetHeader,
  SheetTitle,
  SheetTrigger,
} from '@/components/ui/sheet';
import stars from '../images/Group 88.png';
import { urlFor } from '@/sanity/lib/image';
import { FaFacebook, FaLinkedin } from 'react-icons/fa';
import { AiFillTwitterCircle } from 'react-icons/ai';

const Productdetail = ({ product }: { product: any }) => {
  const [cart, setCart] = useState<any[]>([]);
  const [quantity, setQuantity] = useState(1);

  // On component mount, load cart from localStorage
  useEffect(() => {
    const storedCart = JSON.parse(localStorage.getItem('cart') || '[]');
    setCart(storedCart);
  }, []);


  const increaseQuantity = () => {
    setQuantity((prev) => prev + 1);
  };

  const decreaseQuantity = () => {
    if (quantity > 1) {
      setQuantity((prev) => prev - 1);
    }
  };

  // Add the product to cart (or update quantity if it exists)
 const addToCart = (prod: any) => {
    const existingIndex = cart.findIndex((item) => item.id === prod.id);
    let newCart;
    if (existingIndex !== -1) {
      newCart = [...cart];
      newCart[existingIndex].quantity += quantity;
    } else {
      newCart = [...cart, { ...prod, quantity }];
    }
    setCart(newCart);
    localStorage.setItem('cart', JSON.stringify(newCart));
    Swal.fire({
      title: "Added to Cart!",
      text: `${prod.name} has been added to your cart.`,
      icon: "success",
      confirmButtonColor: "#B88E2F",
      confirmButtonText: "OK",
      timer: 2000, // Auto close after 2 seconds
      showConfirmButton: false, // Hide the confirm button for auto-close
    });

  };

  // Remove the product from the cart and update localStorage
  const removeProduct = () => {
    const newCart = cart.filter((item) => item.id !== product.id);
    setCart(newCart);
    localStorage.setItem('cart', JSON.stringify(newCart));
  };

  // Check if the product is in the cart
  const productInCart = cart.find((item) => item.id === product.id);

  return (
    <div className='w-full overflow-hidden mb-20'>
      <div>
        <div className='max-w-[1150px] p-7 mx-auto'>
          <div className='ml-10 sm:ml-0 mb-10'>
            <p className='text-[#9F9F9F] font-normal'>
              Home{' '}
              <RiArrowDropRightLine className='inline text-black' /> Shop{' '}
              <RiArrowDropRightLine className='inline text-black' />
              <span className='inline text-black font-light ml-3 mr-3 text-2xl'> | </span>
              <span className='inline font-normal text-black'>{product.name}</span>
            </p>
          </div>

          <div className='xl:w-[1150px] flex flex-col xl:flex-row items-start justify-center gap-20'>
            {/* Product Image */}
            <div className='w-52 h-52 max-w-[423px] max-h-[500px] bg-[#FFF9E5] flex items-center justify-
center mx-auto sm:mx-0'>
              <Image
                src={urlFor(product.image).url()}
                alt={product.name}
                width={400}
                height={500}
                className='h-[390px]'
              />
            </div>

            <div>
              <div className='lg:w-[500px] sm:h-[730px]'>
                <div className='flex flex-col justify-between h-[450px]'>
                  <p className='w-[300px] mx-auto text-3xl sm:text-5xl font-normal'>
                    {product.name}
                  </p>
                  <p className='text-2xl font-medium text-[#9F9F9F]'>Rs. {product.price}</p>
                  <span>
                    <Image
                      src={stars}
                      alt='Rating'
                      width={124}
                      height={20}
                      className='inline'
                    />
                    <span className='text-[#9F9F9F] font-normal'> | </span>
                    <span className='text-[#9F9F9F] text-[14px] font-normal'>
                      5 Customer Review
                    </span>
```

```tsx
'use client'
import React, { useState, useEffect } from 'react'
import Image from 'next/image'
import Link from 'next/link'
import { Product } from '@/types/Products'
import { client } from '@/sanity/lib/client'
import { urlFor } from '@/sanity/lib/image'
import Pagination from './Pagination'
interface ProductsListingProps {
  searchQuery: string;
}

export default function Productslisting({ searchQuery }: ProductsListingProps) {
  const [products, setProducts] = useState<Product[]>([]);
  const [currentPage, setCurrentPage] = useState(1);
  const itemsPerPage = 6;

  useEffect(() => {
    async function fetchProducts() {
      const fetchedProduct: Product[] = await client.fetch(`*[_type == "product"]{
        _id,
        "id": id,
        name,
        price,
        category,
        "image": image.asset._ref,
      }`);
      setProducts(fetchedProduct);
    }
    fetchProducts();
  }, []);

  // Filter products based on search query
  const filteredProducts = products.filter((product) => {
    const productName = product.name.toLowerCase();
    const productCategory = product.category?.toLowerCase() || '';
    const searchTerm = searchQuery.toLowerCase();

    return productName.includes(searchTerm) || productCategory.includes(searchTerm);
  });

  const totalPages = Math.max(1, Math.ceil(filteredProducts.length / itemsPerPage));
  const startIndex = (currentPage - 1) * itemsPerPage;
  const paginatedProducts = filteredProducts.slice(startIndex, startIndex + itemsPerPage);

  return (
    <div>
      {filteredProducts.length === 0 ? (
        <div className="flex flex-col items-center justify-center my-20">
          <p className="text-gray-500 text-lg font-semibold">🚫 No products found</p>
          <p className="text-gray-400 text-sm mt-2">Try searching with different keywords.</p>
        </div>
      ) : (
        <>
          <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-3 py-10">
            {paginatedProducts.map((product) => (
              <div key={product._id} className="w-full h-auto flex flex-col p-5">
                <Link href={`/product/${product.id}`}>
                  <div className="flex items-center justify-center mb-4">
                    <Image
                      src={urlFor(product.image).url()}
                      alt="Product Image"
                      width={200}
                      height={190}
                      className="object-cover h-[190px]"
                    />
                  </div>
                </Link>

                <div className="flex flex-col gap-2 items-center">
                  <h3 className="font-normal text-black">{product.name}</h3>
                  <p className="text-black font-medium text-2xl">Rs. {product.price}</p>
                </div>
              </div>
            ))}
          </div>

          {/* Pagination (Always Visible) */}
```

Dynamic Route:

```tsx
import React from 'react';
import { client } from "@/sanity/lib/client";
import Productdetail from '../../components/Productdetail'

const page = async ({ params: { id } }: { params: { id: string } }) => {
  const query = `*[ _type == "product" && id == $id]{
    name,
    "id": id,
    price,
    description,
    category,
    stockLevel,
    "image": image.asset._ref,
    "images": gallery[].asset._ref
  }[0]`;

  const product = await client.fetch(query, { id });

  if (!product) {
    return (
      <div className="flex items-center justify-center h-screen bg-gray-100">
        <h1 className="text-2xl font-semibold text-gray-600">Product not found</h1>
      </div>
    );
  }

  return <Productdetail product={product} />;
};

export default page;
```

Challenges Faced and Solutions Implemented

Challenge:

Add to Cart Functionality: Initially, the logic for adding, updating, and removing items from the cart was unclear. Managing the cart state and ensuring proper synchronization with localStorage was challenging.

Solution:

- Implemented a cart utility module (cart.ts) to handle all cart-related operations.
- Used localStorage to persist cart data across sessions.
- Added functions like addToCart, removeFromCart, updateCartQuantity, and calculateTotal to manage cart operations efficiently.

Challenge:

Dynamic Routing: Implementing dynamic routing for individual product pages was initially challenging. Fetching the correct product data based on the URL parameter (id) and handling cases where the product doesn't exist required careful planning.

Solution:

- Used Next.js dynamic routing with [id]/page.tsx to create unique pages for each product.
- Fetched product data from Sanity CMS using a GROQ query based on the id parameter.
- Added a fallback UI to handle cases where the product is not found.

Challenge:

Component Creation: Building complex components like the SearchBar, Pagination, and Category Component took a significant amount of time due to the need for dynamic functionality, responsiveness, and seamless integration with the backend.

Solution:

- Leveraged ChatGPT to accelerate the development process.
- Used AI-generated code snippets and logic to quickly implement the required features.
- Customized the generated code to fit the specific requirements of the project.

Challenge:

Cart to Checkout Data Transfer: Transferring cart data to the checkout page and managing form validation was complex and time-consuming.

Form Validation Logic: Implementing robust validation for fields like email, phone, and postal code required careful handling.

Solution:

Cart Data Transfer:

- Used localStorage to persist cart data across sessions.
- Fetched cart data on the checkout page using useEffect and displayed it dynamically.

Form Validation:

- Created a validateForm function to check for required fields and validate inputs (e.g., email, phone, postal code).
- Displayed error messages dynamically for invalid fields.

Challenge:

Order Placement to Sanity: Transferring order data from the checkout page to Sanity CMS was extremely challenging. Despite multiple attempts, the order data was not being correctly structured or sent to Sanity.

Best Practices Followed

Modular Components:

Created reusable components (e.g., ProductListing, SearchBar) for scalability.

Responsive Design:

Ensured all components are mobile-friendly using Tailwind CSS.

Error Handling:

Added error boundaries and fallback UI for better user experience.

Documentation:

Maintained clear and concise comments in the code for future reference.