Day 2 - Planning the Technical Foundation

1. ***Project Overview***

The Furniro Marketplace is an e-commerce platform designed to provide a seamless shopping experience for customers. The platform supports product browsing, order placement, and shipment tracking, along with an admin panel for managing products and users.

2. Technology Stack

- **Frontend: React.js with Tailwind CSS for a modern and responsive UI.**
- **Backend: Node.js with Express.js for handling API requests.**
- **Database: MongoDB with Mongoose ORM for data management.**
- **Authentication: JWT (JSON Web Tokens) for secure login and session management.**
- **Payment Integration: Stripe API for processing transactions.**
- **Shipping Integration: ShipEngine API for order tracking.**

3. System Architecture

The system follows a microservices architecture, where the frontend communicates with the backend via REST APIs. Key components include:

- **Client Side (Frontend):**
  - **UI for product listing and order processing**
  - **Authentication and user session management**
- **Server Side (Backend):**
  - **Handles API requests and business logic**
  - **Manages user authentication and authorization**
  - **Processes payments and shipments**
- **Database Layer:**
  - **Stores product, user, and order data**
  - **Ensures data consistency with schema validation**

4. Database Schema

The main entities in the database are:

- **Users: (Customer, Admin)**
  - `id`, `name`, `email`, `password`, `role`
- **Products:**
  - `id`, `name`, `description`, `price`, `category`, `image`, `stock`
- **Orders:**
  - `id`, `userId`, `productId`, `quantity`, `totalPrice`, `status`

5. API Endpoints

Method	Endpoint	Description
POST	`/api/auth/register`	Register a new user
POST	`/api/auth/login`	Authenticate user
GET	`/api/products`	Retrieve all products
POST	`/api/orders`	Place an order
GET	`/api/orders/:id`	Retrieve order details

6. User Workflows

Customer Journey:

1. User registers and logs in.
2. Browses products and adds items to the cart.
3. Proceeds to checkout and makes a payment.

4. Receives order confirmation and tracks the shipment.

Admin Workflow:

1. Admin logs in to the dashboard.
2. Adds, updates, or removes products.
3. Manages user orders and shipment status.
4. Reviews system analytics and reports.

7. Security Measures

- Authentication & Authorization: JWT-based authentication ensures secure access control.
- Data Validation: Input validation to prevent SQL injection and XSS attacks.
- Secure Payments: Stripe API ensures encrypted transactions.
- Rate Limiting: To prevent API abuse and DDoS attacks.

Design by : Sanaullah Ahmed

Day-03

API INTEGRATION AND DATA SETUP FOR FURNIRO FURNITURE WEBSITE

Overview

This documentation outlines the progress made on Day 3 of the Furniro website development. It covers API integration, setting up custom schemas for furniture data, and using GROQ queries to display content in a Next.js app. Since data is manually added through the Sanity dashboard, this guide focuses on schema setup and querying, excluding data migration.

CUSTOM SCHEMA SETUP IN SANITY CMS

The **custom schema** defines how the **furniture data**, such as **product details, pricing, and category information**, is structured in the **Sanity CMS**.

Custom Validation

The schema incorporates **validation rules** to maintain **data accuracy and integrity**.

- **Price** must be a **positive number**.
- **Title** and **Description** fields **cannot be left blank**.

Sanity API Integration

The application connects to **Sanity's API** using a **configured project ID and dataset**.

- **Authentication** is handled securely via an **API token**.
- **Environment variables** ensure **sensitive information** is **protected**.

Fetching Data from Sanity

- **GROQ queries** are used to retrieve **structured content** from **Sanity CMS**.
- **Essential furniture-related data** includes:
    - **Product names**
    - **Categories** (e.g., sofas, chairs, tables, lighting)
    - **Prices**
    - **Descriptions**
    - **Images**

Example Query

A **typical query** fetches **furniture data**, ensuring all necessary details are **accessible for display**.

Mapping and Formatting

- The **fetched data** is **mapped and formatted** to align with the website's **schema** and **design specifications**.
- **Each product record** (e.g., sofa, dining table) is **restructured** to match the **front-end display requirements** of **Furniro**.
- Data is structured for **optimal rendering** on the website.

Displaying Data

- The structured data fetched from **Sanity** is dynamically **displayed** on the **Furniro website**.
- Categories include:

    - **Sofas**
    - **Chairs**
    - **Tables**
    - **Lighting**
- Since the data is **pulled live** from **Sanity**, no **manual API insertion** or **database storage** is required.
- The website reflects **real-time updates** automatically.

CLIENT-SIDE CODE

GROQ Query to Fetch Data

The **getServerSideProps** function utilizes a **GROQ query** to fetch data from **Sanity** during **server-side rendering (SSR)**.

- Ensures **fast content delivery** before serving the page to the user.

Rendering Items

- The **ClientPage** component renders **furniture items** passed via **props**.
- Uses **React's .map() method** to display:
    - **Product name**
    - **Description**
    - **Image**

Dynamic Routing

- **Dynamic routing links** allow users to click a product and navigate to a **detailed page** (e.g., `/product/[id]`).

Code Highlights

- **SSR Optimization**: Improves **loading times** and enhances **SEO**.
- **Responsive Design**: Works on **mobile, tablet, and desktop** devices.

Responsive Design

- The **Furniro website** uses **modern CSS** or **Tailwind CSS**.
- The layout **adapts seamlessly** to all devices.
- Ensures **accessibility** for all users.

Interactive User Experience

- **Add to Cart** button and **View Details** option enhance engagement.
- **Optimized images** for **fast loading** and **high-quality visuals**.

Reusable Components

- **Card components** are designed for **consistent use** across:
  - **Homepage**
  - **Category pages**
  - **Promotional sections**

Secure Configuration

- **API keys** and **database credentials** are securely **managed** using a **.env file**.

Environment Configuration

Project Identification

- **SANITY_PROJECT_ID** ensures all API interactions are **directed to the correct Sanity project**.

Dataset Management

- **SANITY_DATASET** specifies the **environment type** (e.g., **production** or **development**).

API Authentication and Security

Secure API Token

- The **SANITY_API_TOKEN** authenticates requests to **Sanity APIs**.
- Never **exposed to frontend** or **unauthorized users**.

Backend Connections

- Additional **environment variables** manage **database connections**.

Best Practices for Security

Protected Access

- **Environment variables** are stored in **.env files**.
- Accessed securely via `process.env`.

Mitigating Risks

- **Sensitive information** like tokens, keys, and database configurations are **protected**.

Frontend Implementation Highlights

Day 3 Achievements

The focus was on **backend setup and dynamic integration** for the **Furniro website**.

Key Highlights:

✅ **Schema Design**: Structured **product data** in **Sanity**.
✅ **Dynamic Content**: Integrated **GROQ queries** for **fetching/rendering** data.
✅ **Responsive UI**: Designed a **mobile-friendly** layout.
✅ **Secure Setup**: Managed **sensitive configurations** with **environment variables**.

Thank You!

Designed by Sanaullah

Day 4 - Building Dynamic Frontend Components for Furniro Marketplace

This document offers a detailed analysis of the core functionalities of a dynamic marketplace, focusing on modularity, reusability, and seamless integration with **Sanity CMS**. Each feature is thoroughly explained, culminating in a conclusion that summarizes the overall approach.

Step 1: Overview of Core Functionalities

The project incorporates the following essential features to create a responsive, scalable, and efficient marketplace:

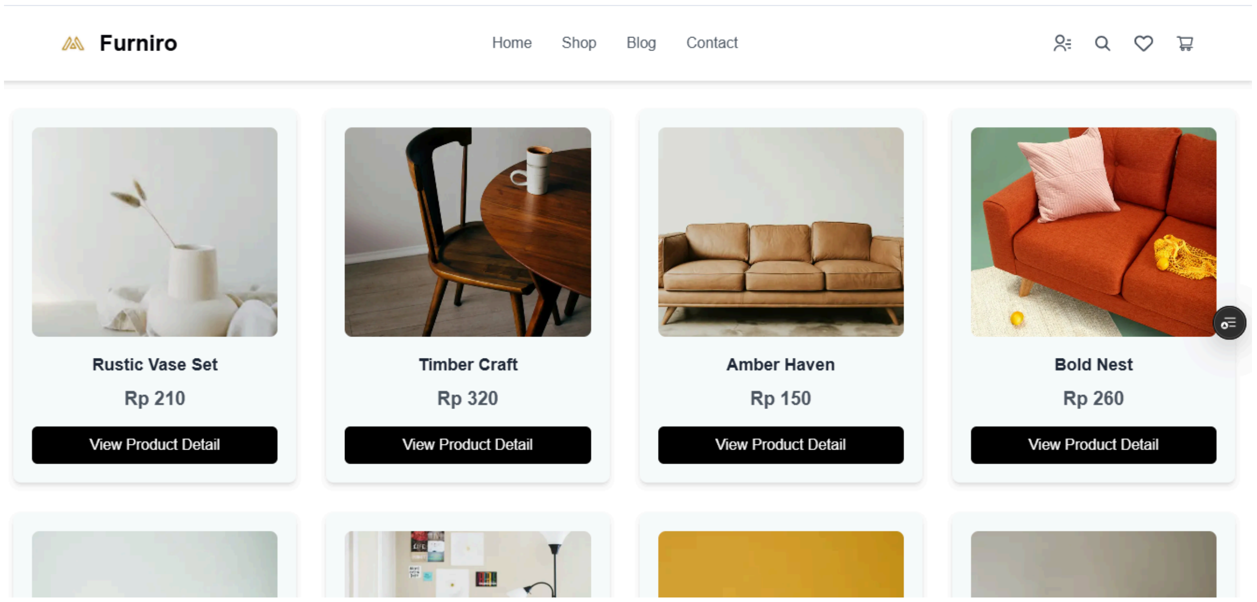
1. **Product Listing Page**
2. **Dynamic Routing**
3. **Shopping Cart Functionality**
4. **Secure Checkout Process**
5. **Advanced Price Calculation**
6. **Product Comparison Tool**

Each feature plays a critical role in delivering a seamless and user-friendly experience.

Step 2: Detailed Functionalities

1. Product Listing Interface

The Product Listing Interface serves as the main gateway for users to explore available products. Leveraging dynamic data fetched from **Sanity CMS**, products are presented in a structured, visually engaging format, with options to display them in grid or list layouts for optimal usability.



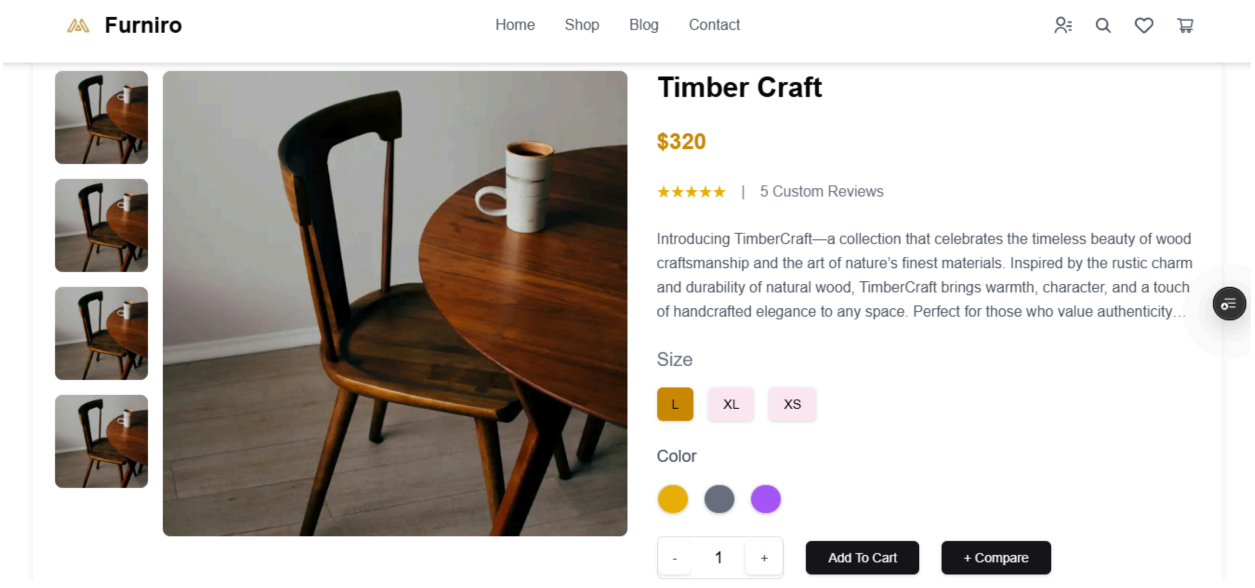
Product Listing Page Features:

- **Advanced Sorting and Filtering:**
  Provides enhanced usability by allowing users to sort and filter products based on attributes such as price, categories, and popularity.
- **Efficient Pagination:**
  Ensures smooth navigation and optimal performance when handling large datasets, enhancing user experience.
- **Responsive Design:**
  Guarantees seamless compatibility across a range of devices, from desktops to mobile phones, ensuring a consistent user experience.
- **Real-Time Integration with Sanity CMS:**
  Enables instant synchronization of product updates from the backend, ensuring the displayed data is always current.

---

2. Dynamic Routing

Dynamic routing facilitates the generation of dedicated product detail pages, offering users comprehensive information about individual products in a structured and user-friendly format.


Furniro

Home Shop Blog Contact

Timber Craft

$320

★★★★★ | 5 Custom Reviews

Introducing TimberCraft—a collection that celebrates the timeless beauty of wood craftsmanship and the art of nature's finest materials. Inspired by the rustic charm and durability of natural wood, TimberCraft brings warmth, character, and a touch of handcrafted elegance to any space. Perfect for those who value authenticity...

Size

L XL XS

Color

- 1 +

Add To Cart

+ Compare

Product Pages with Dynamic Routing:

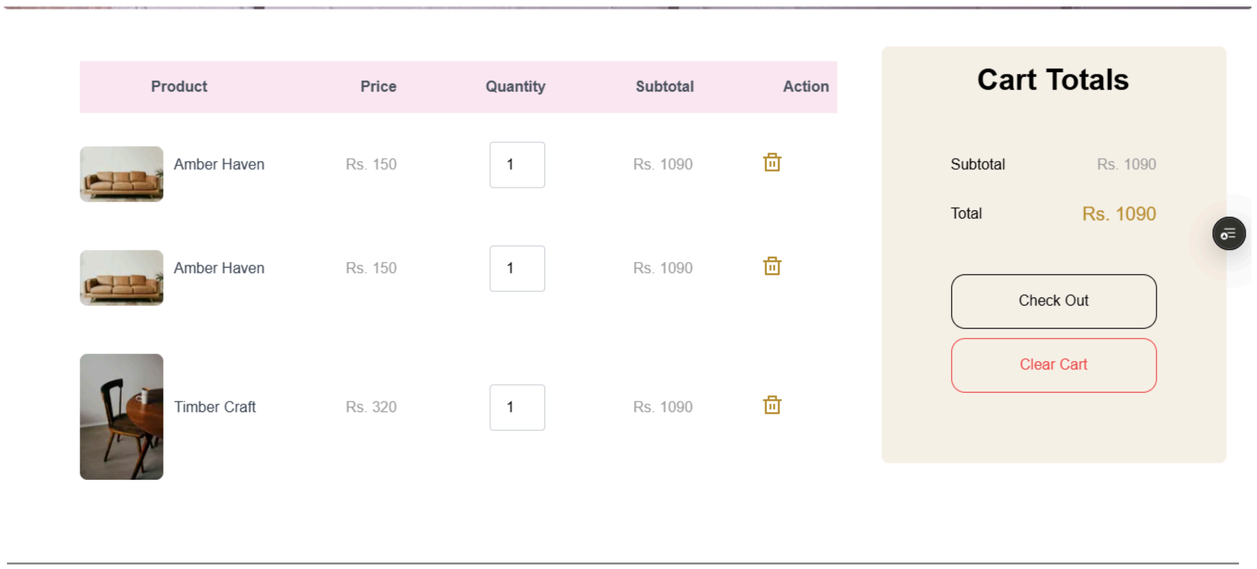
1. Each product is uniquely identified by an **ID** or **slug**, which is used to dynamically generate its dedicated URL (e.g., `/product/[id]`).
2. These pages are **server-rendered** to optimize SEO and ensure faster initial load times, improving the overall user experience.
3. Dynamic routing enables the seamless display of essential product information, such as **descriptions, images, pricing, stock availability, and customer reviews**.
4. This scalable solution allows **new products** to be automatically assigned corresponding pages, eliminating the need for manual updates.

---

3. Seamless Cart Functionality

The cart functionality is designed to streamline the shopping experience by managing and tracking the user's selected items.

1. Efficiently **handles item selection**, calculates the total cost, and provides a **clear summary** of the user's choices, ensuring a smooth checkout process.
2. This feature enhances user satisfaction by providing **real-time updates** on product selections and overall costs.



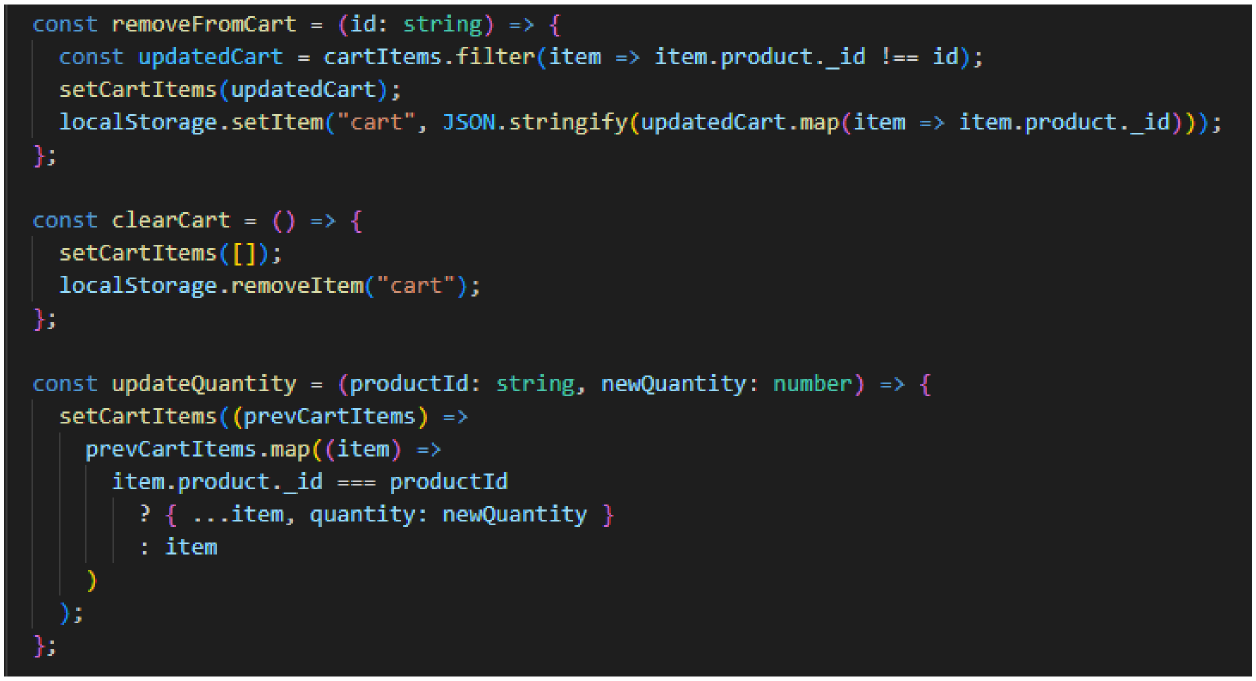
Product	Price	Quantity	Subtotal	Action
Amber Haven	Rs. 150	1	Rs. 1090	
Amber Haven	Rs. 150	1	Rs. 1090	
Timber Craft	Rs. 320	1	Rs. 1090	

Cart Totals

Subtotal	Rs. 1090
Total	Rs. 1090

4. Checkout Process

1. The checkout process is **streamlined** into multiple clear steps: **Billing Details and Payment Information**, ensuring an organized and efficient user experience.
2. A **Dynamic Progress Tracker** visually indicates the user's current step, helping users stay informed throughout the process.
3. **Input Validation** is integrated at each step to ensure that all required fields are filled correctly, minimizing the risk of errors during order submission.
4. While payment integration can initially be simulated, the checkout system is designed to be **extensible**, supporting payment gateways such as **Stripe** or **PayPal** for secure transactions.
5. At the final stage, **Order Summaries** are displayed, allowing users to review and confirm their order details before proceeding with the **Place Order** step, ensuring accuracy before finalizing the purchase.



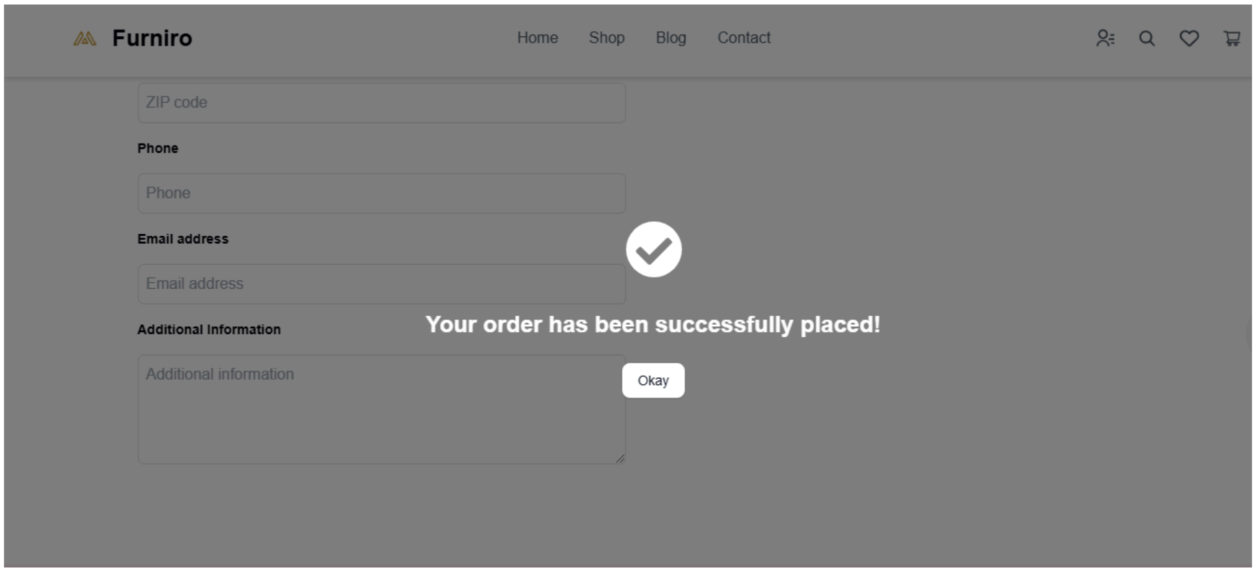
```
const removeFromCart = (id: string) => {
  const updatedCart = cartItems.filter(item => item.product._id !== id);
  setCartItems(updatedCart);
  localStorage.setItem("cart", JSON.stringify(updatedCart.map(item => item.product._id)));
};

const clearCart = () => {
  setCartItems([]);
  localStorage.removeItem("cart");
};

const updateQuantity = (productId: string, newQuantity: number) => {
  setCartItems((prevCartItems) =>
    prevCartItems.map((item) =>
      item.product._id === productId
        ? { ...item, quantity: newQuantity }
        : item
    )
  );
};
```

Cart Functionality Breakdown:

- **Remove Product from Cart:** Deletes a product using its unique **ID** and syncs the changes with the cart state and `localStorage`.
- **Empty the Cart:** Clears all products, resets the cart state, and removes cart data from `localStorage`.
- **Adjust Product Quantity:** Updates the quantity of a product with validation to prevent invalid values (e.g., negative or zero) while reflecting changes in the cart state and `localStorage`.



Your order has been successfully placed!

---

5. Placing the Order

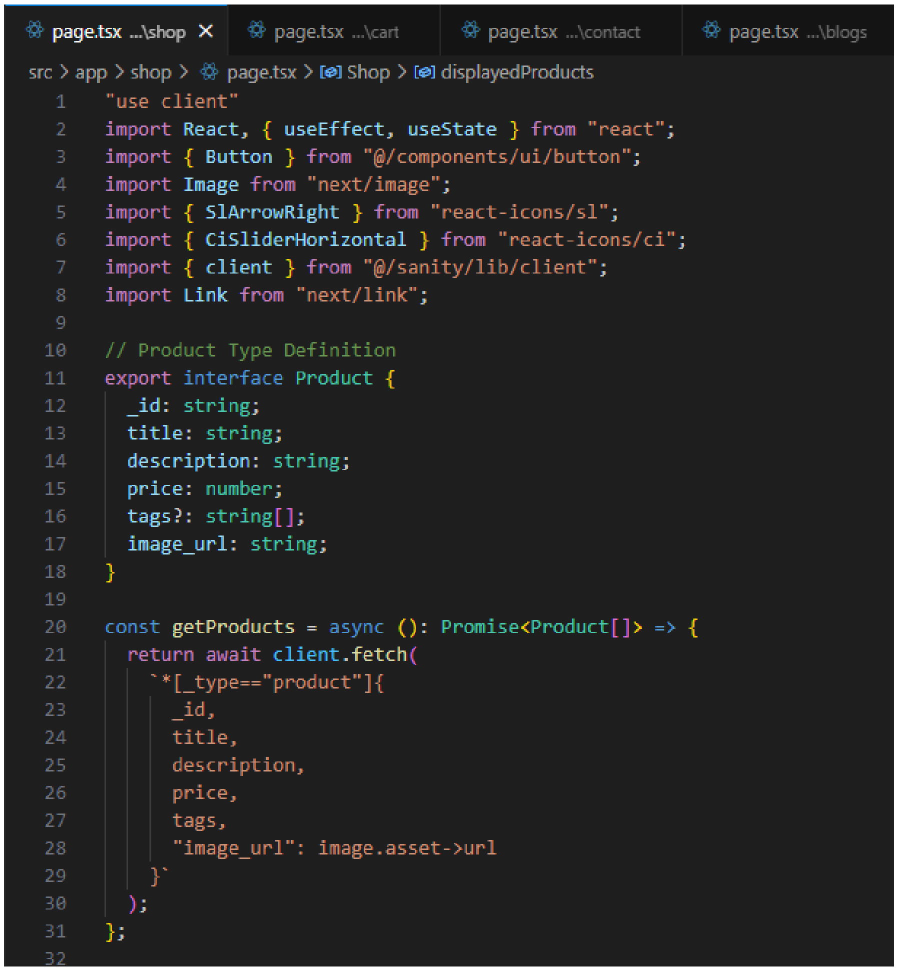
The **"Place Order"** feature marks the successful completion of the shopping journey by:

1. **Review and Confirmation:** Providing users with a detailed summary of their order, including item details, billing, and payment information, to ensure everything is accurate before proceeding.
2. **Secure Order Submission:** Sending the finalized order details to the backend for secure processing and storage.
3. **Real-Time Acknowledgment:** Displaying a **success notification** or redirecting to a confirmation page to reassure users that their order has been placed successfully.

---

6. Connecting with Sanity CMS

Sanity CMS acts as the backend, enabling **dynamic management and retrieval** of product data.



```tsx
"use client"
import React, { useEffect, useState } from "react";
import { Button } from "@/components/ui/button";
import Image from "next/image";
import { SlArrowRight } from "react-icons/sl";
import { CiSliderHorizontal } from "react-icons/ci";
import { client } from "@/sanity/lib/client";
import Link from "next/link";

// Product Type Definition
export interface Product {
  _id: string;
  title: string;
  description: string;
  price: number;
  tags?: string[];
  image_url: string;
}

const getProducts = async (): Promise<Product[]> => {
  return await client.fetch(
    `*[_type=="product"]{
      _id,
      title,
      description,
      price,
      tags,
      "image_url": image.asset->url
    }`
  );
};

```

Comprehensive Overview:

1. **Sanity CMS** stores products, categories, and metadata, empowering admins to update content without modifying the codebase.
2. A **powerful client** efficiently queries **Sanity CMS**, ensuring dynamic and reliable data fetching.
3. **Real-time updates** in the CMS are instantly reflected on the frontend, delivering a seamless content management experience.
4. The integration is highly **scalable**, allowing for the effortless addition of new data types or fields as the marketplace evolves.

Final Thoughts

This guide presents an **in-depth strategy** for creating dynamic, responsive marketplace components. With **Sanity CMS** managing the backend and employing **modular frontend development** practices, the application ensures **scalability, performance, and an exceptional user experience**.

Every feature—from **product listings to dynamic content updates**—contributes to building a **professional marketplace tailored to real-world demands**. Future upgrades, such as **advanced analytics** or **AI-driven recommendations**, hold the potential to further enhance the platform's capabilities.

Designed by SANAULLAH
