Hackathon Day 4!

Detailed Documentation for Dynamic Components and Functionalities

This documentation provides an in-depth analysis of the key functionalities for a dynamic marketplace, emphasizing modularity, reusability, and integration with Sanity CMS. Each feature is described comprehensively, followed by a conclusion summarizing the approach.

Step 1: Functionalities Overview

The project implements the following core functionalities:

1. **Product Listing Page**
2. **Dynamic Route**
3. **Cart Functionality**
4. **Checkout**
5. **Price Calculation**
6. **Product Comparison**
7. **Inventory Management**

Each functionality contributes to building a responsive and scalable marketplace.

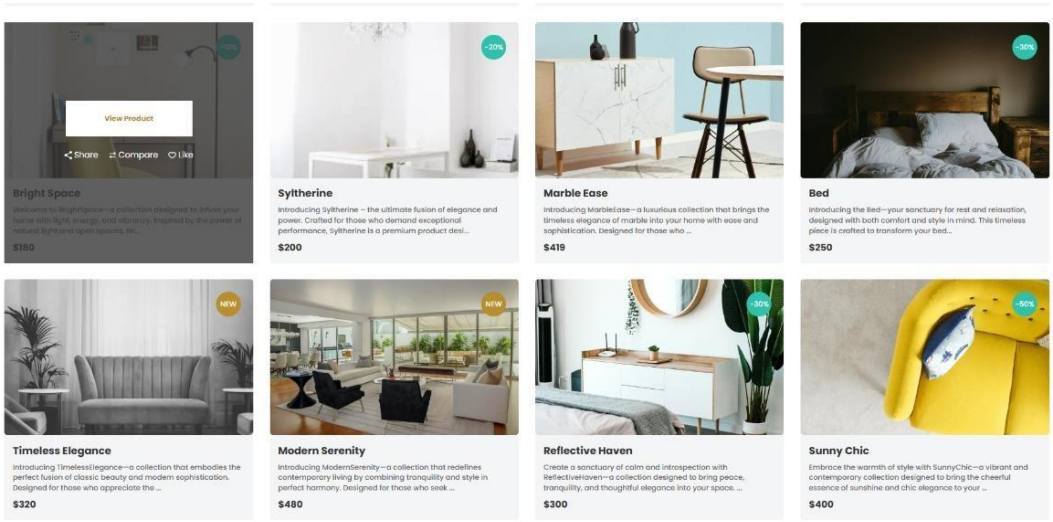
Step 2: Functionalities in Detail

1. Product Listing Page

The Product Listing Page is the primary interface where users can view all the available products in a structured and visually appealing format. Products are displayed dynamically, fetched from Sanity CMS, and rendered in a grid or list layout.

By Umar Farooq

```tsx
"use client";
import ProductCard from "@/components/cards/ProductCard";
import { client } from "@/sanity/lib/client";
import { ImportedData } from "@/types";
import { query } from "@/utils/query";
import { useEffect, useState } from "react";

function ShopProductSection() {
  const [PRODUCTS, setPRODUCTS] = useState<ImportedData[]>([]);
  useEffect(() => {
    const fetchDataFromSanity = async () => {
      try {
        const PRODUCTS = await client.fetch(query);
        setPRODUCTS(PRODUCTS);
      } catch (error) {
        console.log(error);
      }
    }
    fetchDataFromSanity();
  }, [])
  return (
    <section>
      <div className="grid grid-cols-1 md:grid-cols-2 xl:grid-cols-4 gap-[32px] mt-[46px]">
        {PRODUCTS.map((item, index) => (
          <ProductCard {...item} key={index} />
        ))}
      </div>
    </section>
  );
}
```



Detailed Description:

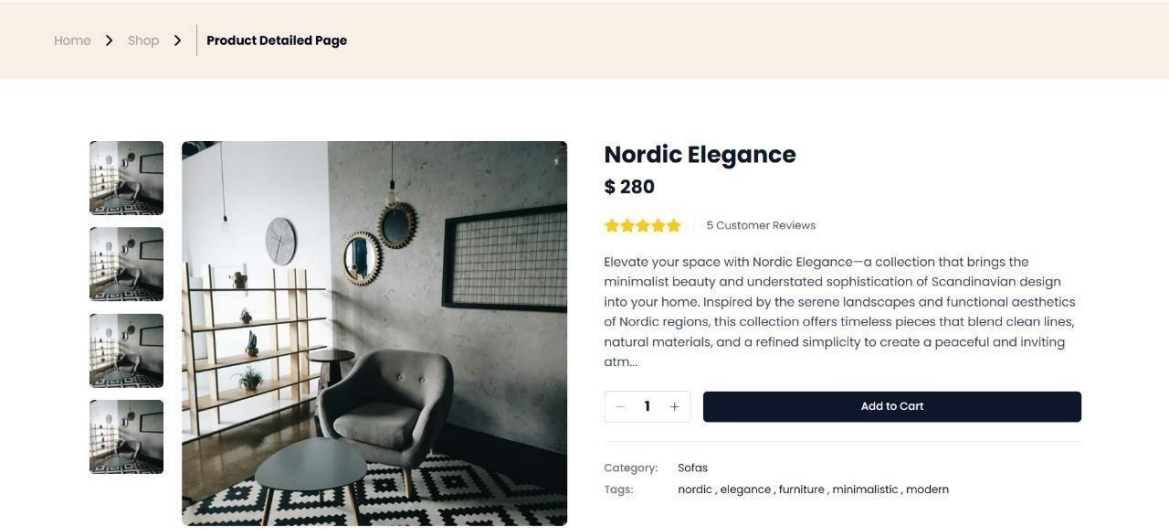
- The page offers sorting and filtering options to enhance usability, allowing users to organize products based on price, categories, or popularity.
- Pagination ensures the seamless handling of large datasets, improving performance and user experience.

By Umar Farooq

- Responsive design ensures compatibility across devices, from desktops to mobile phones.
- Integration with Sanity CMS ensures real-time updates, so any product changes in the backend are instantly reflected.

2. Dynamic Route

Dynamic routing allows for the creation of individual product detail pages, enabling users to view detailed information about each product.

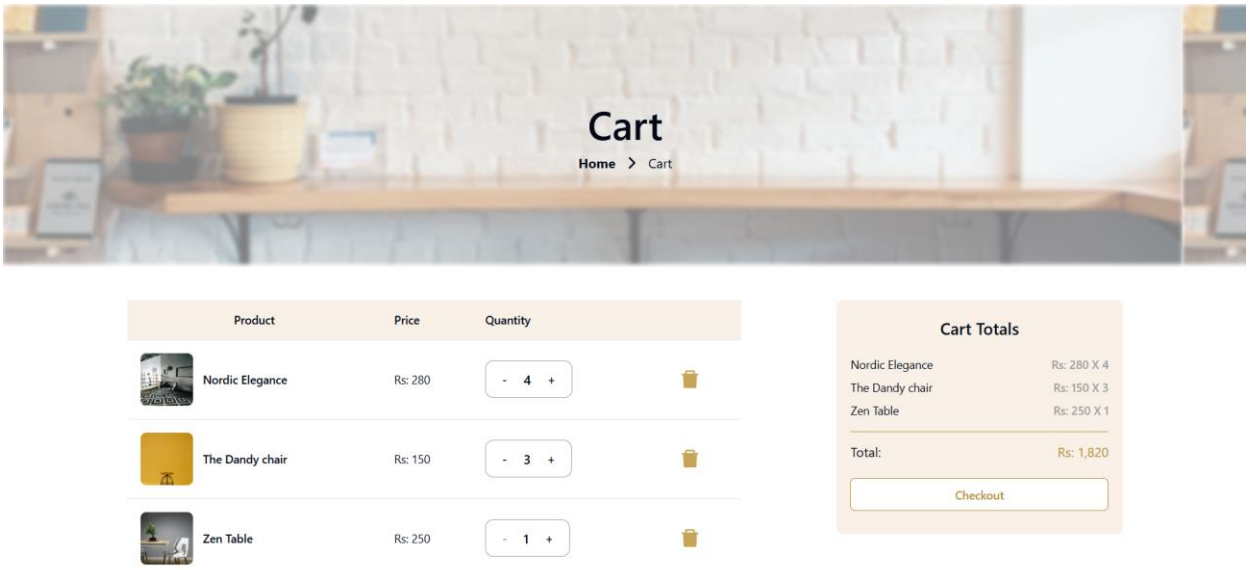


Detailed Description:

- Every product has a unique identifier (ID or slug) used to dynamically generate its URL (e.g., `/product/[id]`).
- These pages are server-rendered to improve SEO and provide faster initial load times.
- Dynamic routes display details like product descriptions, images, price, stock status, and reviews.
- This approach ensures scalability, allowing new products to automatically generate corresponding pages without manual intervention.

3. Cart Functionality

The Cart Functionality manages the user's selected items, providing a seamless shopping experience by tracking their choices and summarizing costs.

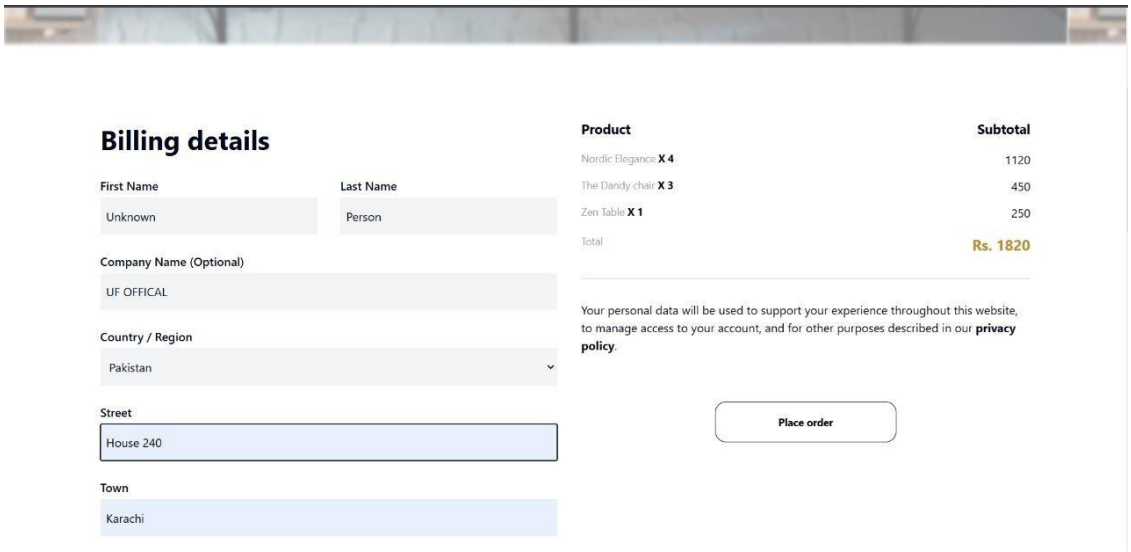


Detailed Description:

- Users can add products to their cart directly from the product listing or detail page.
- The cart dynamically updates quantities and calculates the total cost, ensuring a real-time experience.
- A mini-cart displays a quick summary of selected items, while a detailed cart page offers options to edit or remove items.
- State management tools, such as React Context or Redux, are used to maintain the cart state across the application.
- Cart data persistence is achieved using local storage or session storage, ensuring the cart remains intact even if the page is refreshed.

4. Checkout

The Checkout functionality streamlines the purchase process, collecting and validating user information to finalize the order.



Detailed Description:

- The checkout process is divided into multiple steps: billing details, shipping address, and payment information.
- A dynamic progress tracker indicates the current step, enhancing the user experience.
- Input validation ensures that all required fields are filled correctly, reducing errors during order submission.
- Although payment integration can be mocked initially, it is designed to be extendable with payment gateways like Stripe or PayPal.
- Order summaries are displayed at the end, allowing users to confirm their details before finalizing the purchase.

5. Price Calculation

Price Calculation dynamically computes the total cost of items in the cart, factoring in taxes, discounts, and other adjustments.

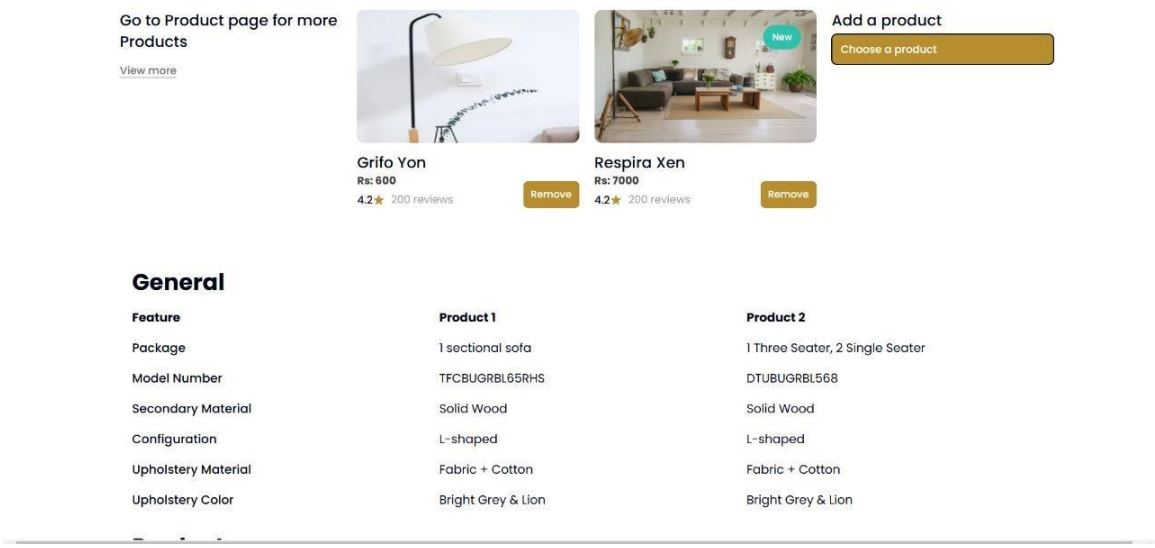
```tsx
import LinkOutlineButton from "@/components/sections/link-btn-outline";
import { ICart } from "@/types";
import { useEffect, useState } from "react";

export default function CartTotals() {
  const [cartItems, setCartItems] = useState<ICart[]>([]);

  useEffect(() => {
    const storedCart = localStorage.getItem("CART_ITEMS");
    if (storedCart) {
      setCartItems(JSON.parse(storedCart));
    }
  }, [cartItems]);

  const calculateTotal = () =>
    cartItems.reduce((total, item) => total + item.quantity * item.price, 0);

  if (cartItems.length === 0) {
    return null;
  }
```

Detailed Description:

- The subtotal updates in real-time as items are added, removed, or quantities are adjusted.
- Taxes and discounts are calculated dynamically based on predefined rules, offering flexibility to apply promotional codes.
- The calculation logic is optimized to handle multiple scenarios, such as bulk discounts or tiered pricing.
- This functionality improves transparency by breaking down costs, giving users a clear understanding of the final price.

6. Product Comparison

Product Comparison enables users to evaluate multiple products side by side, facilitating informed purchasing decisions.



Detailed Description:

- Users can add products to a comparison list from the product listing or detail page.
- Key attributes, such as price, features, ratings, and availability, are displayed in a table format.
- The interface highlights differences and similarities, simplifying the decisionmaking process.
- This feature is especially useful for marketplaces offering diverse products with varying specifications.
- The comparison functionality is designed to handle multiple products while maintaining readability and user-friendliness.

7. Inventory Management

Inventory Management tracks the availability of products, ensuring users are informed about stock level.

Detailed Description:

- Real-time stock tracking helps prevent overselling and notifies users when items are low in stock or unavailable.
- Inventory updates are synchronized with the backend, ensuring accurate data at all times.

- Alerts and indicators, such as "Only 3 left in stock," create a sense of urgency, encouraging purchases.
- Admin interfaces for managing stock levels provide flexibility to update inventory quickly.
- This functionality plays a critical role in maintaining customer satisfaction by avoiding issues related to out-of-stock products.

Step 3: Integration with Sanity CMS

Sanity CMS serves as the backend for managing and retrieving product data dynamically.

```
const query = `*[_type == "product"]{
  _id,
  name,
  "imagePath": imagePath.asset->url,
  price,
  description,
  discountPercentage,
  isFeaturedProduct,
  stockLevel,
  category,
}`;
 const products = await client.fetch(query);
```

Detailed Description:

- Products, categories, and other metadata are stored in Sanity CMS, allowing admins to update content without touching the codebase.
- A robust client is used to query Sanity CMS, fetching data dynamically and efficiently.
- Changes made in the CMS are instantly reflected on the frontend, providing a seamless content management experience.
- The integration is designed to be extendable, allowing the addition of new data types or fields as the marketplace grows.

Conclusion

This documentation outlines a comprehensive approach to building dynamic and responsive marketplace components. By leveraging Sanity CMS for backend management and modular frontend development techniques, the application achieves scalability, efficiency, and a superior user experience.

Each functionality—from product listing to inventory management—plays a vital role in delivering a professional marketplace that meets real-world needs. Future enhancements, such as integrating advanced analytics or AI-based recommendations, can further elevate the platform.

For any additional details, enhancements, or implementation support, feel free to reach out!

By Umar Farooq