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. **Inventory Management**

Each functionality contributes to building a responsive and scalable marketplace.

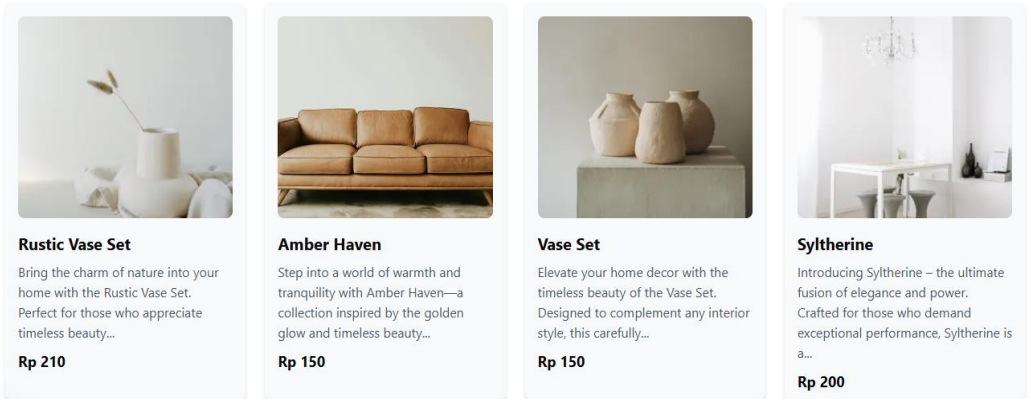
Step 2: Functionalities in Detail

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.

```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:

1. The page includes options to sort and filter products by price, category, or popularity, making it easier for users to find what they need.
2. Pagination helps handle large amounts of products smoothly, improving speed and user experience.
3. The design works well on all devices, from computers to mobile phones.
4. Sanity CMS integration keeps product updates instant, so any changes made in the backend show up right away.

2. Dynamic Route

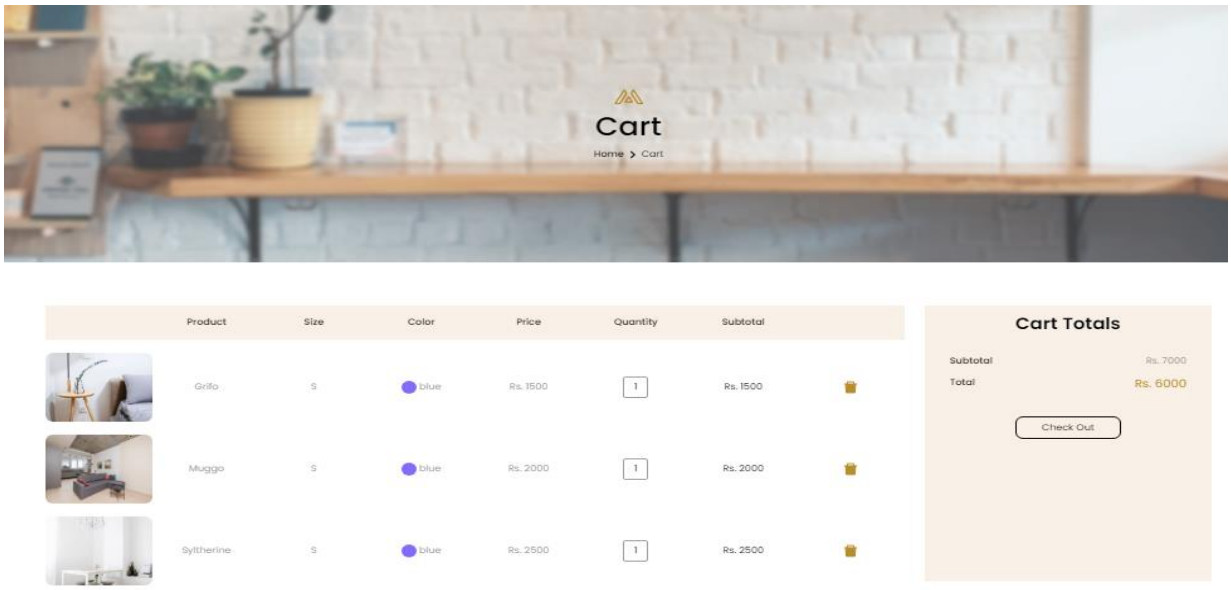
Dynamic routing lets you create separate pages for each product, allowing users to see detailed information about them.

Detailed Description:

- Each product has a unique ID or slug to create its own URL such as, /category/[id].
- These pages are server-reduced to improve SEO and load faster.
- Dynamic routes show details like product descriptions, images, prices, stock availability, and reviews.
- This method automatically creates pages for new products, making it scalable and easy to manage.

3. Cart Functionality

The Cart Functionality tracks the items a user picks, making shopping easy by showing their choices and total cost.

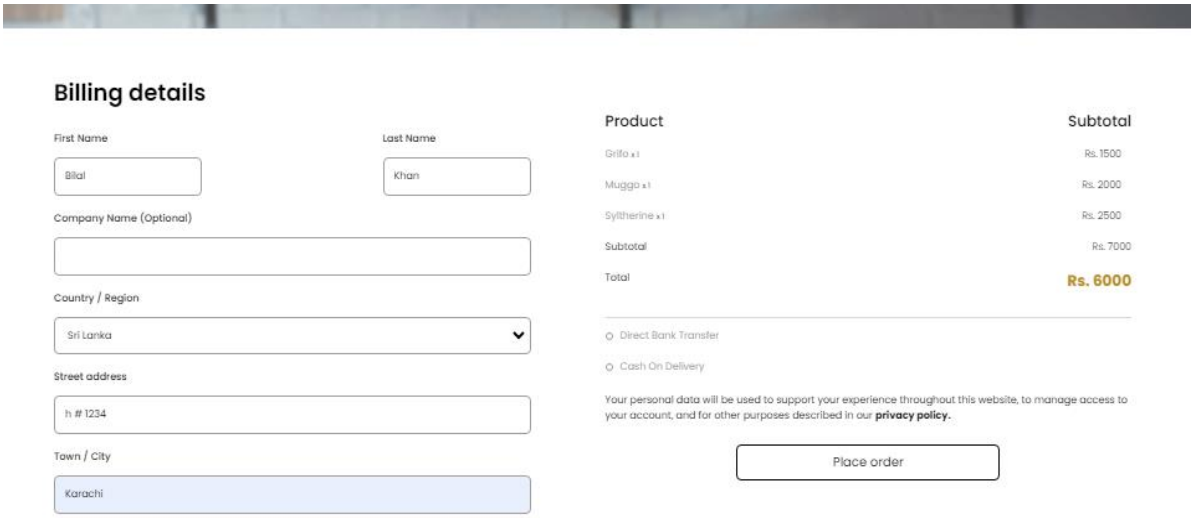


Detailed Description:

- Users can add items to the cart from the product list or details page.
- The cart quickly updates the number of items and total cost.
- A mini cart gives a brief overview, and the full cart page lets users change or remove items.
- Tools like React Context or Redux keep the cart data consistent across the app.
- Local or session storage ensures the cart stays the same after the page is refreshed.

4. Checkout

The checkout process makes buying easier by collecting and verifying your information to finish the order.



Detailed Description:

- The checkout has a few steps: billing info, shipping address, and payment details.
- A progress bar shows where you are in the process, making it easier to follow.
- Validation checks make sure all fields are filled in correctly to avoid mistakes.
- Payment can be tested at first but can later include options like Stripe or PayPal.
- An order summary is shown at the end so users can review their details before finishing 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. Authentication and Authorization

• **Authentication** confirms who the user is by checking their login details like username, password, or third-party logins (e.g., Google or Facebook).

• **Authorization** decides what a user can do or see based on their role or permissions (e.g., admin, editor, or viewer).

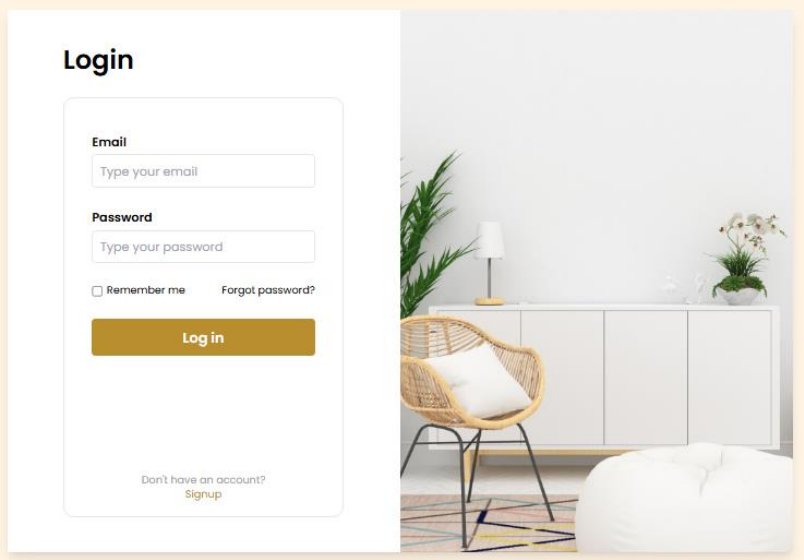
• These steps help keep your data safe by stopping unauthorized users from accessing sensitive information.

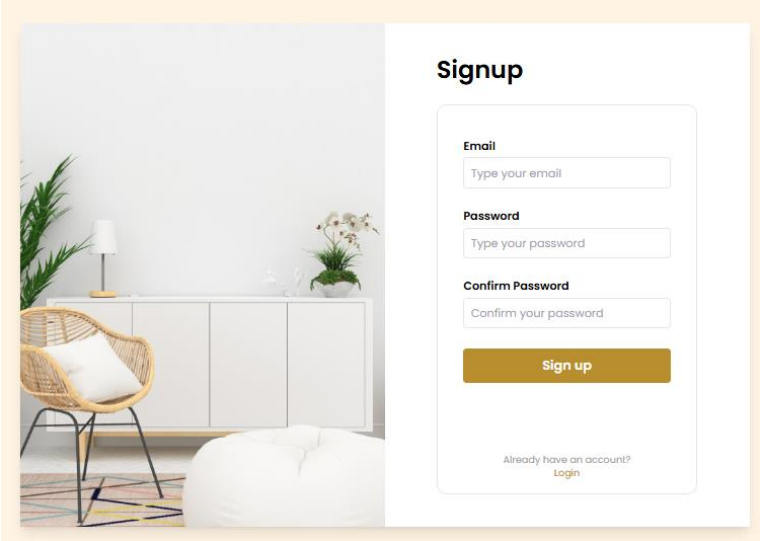
• In **Next.js**, you can use tools like NextAuth.js for easy and secure user login management

with session support.

• **React** works well with services like Firebase Auth or Auth0 to handle user login and access control.

• **Sanity** connects smoothly with authentication tools, making it simple to secure your content and control who can make changes.





7. Inventory Management

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

Detailed Description:

• Real-time stock monitoring stops overselling and informs users when items are low or out of stock.

• Inventory is updated instantly with the backend, keeping data accurate at all times.

• Messages like "Hurry, only a few items remaining" create urgency, motivating customers to buy.

• Admin panels make it easy to manage and update stock quickly.

• This feature helps keep customers happy by avoiding problems with unavailable products

Step 3: Integration with Sanity CMS

Sanity CMS acts as the backend system for handling and fetching product data in real-time.

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

Detailed Description:

- Sanity CMS keeps products, categories, and other details, letting admins make updates without changing the code.
- A strong client fetches data from Sanity CMS quickly and easily.
- Any updates in the CMS show up instantly on the website, making content management simple.
- The system can grow, allowing new fields or data types to be added as the marketplace grows.

Conclusion

This guide shows how to create a responsive marketplace using Sanity CMS and modular frontend tools. It’s built for scalability, efficiency, and smooth user experience. Features like product listing and inventory management ensure a practical solution.