

## 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

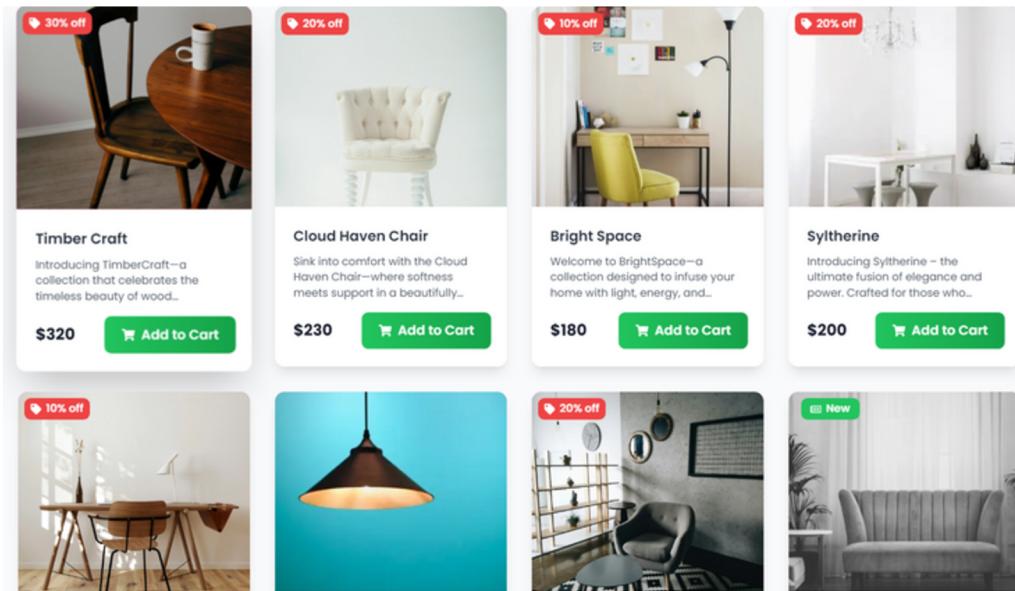
##### **1. Product Listing Page**

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.

MY-PROJECT

```

src > app > components > Our_Products > Our-Products.jsx > ProductsSection > useEffect() callback > fetchProducts
  13  const ProductsSection = () => {
  14    const handleCartToggle = (product: Products) => {
  15      if (isInCart) {
  16        dispatch(removeFromCart(Number(product._id)));
  17      } else {
  18        dispatch(
  19          addToCart({
  20            id: Number(product._id),
  21            title: product.title,
  22            price: product.price,
  23            image: urlFor(product.productImage).url(),
  24            quantity: 1,
  25          })
  26        );
  27        setNotificationProduct(product._id); // Show notification for the clicked product
  28        setTimeout(() => setNotificationProduct(null), 3000); // Hide notification after 3 seconds
  29      }
  30    };
  31
  32    const isProductInCart = (productId: string | number) => {
  33      return cartItems.some((item: { id: number | string }) => item.id === productId);
  34    };
  35
  36    return (
  37      <section className="py-12 px-6 bg-gray-50">
  38        <div className="max-w-screen-xl mx-auto grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-8">
  39          {products.map((product) => (
  40            <div key={product._id} className="group relative bg-white rounded-xl shadow-lg hover:shadow-2xl hover:scale-105 transition-transform duration-300">
  41              <a href={`/product/${product._id}`} passHref>
  42                <div className="relative w-full h-64 overflow-hidden rounded-t-xl bg-gradient-to-br from-purple-400 via-pink-500">
  43                  <img alt={product.title} fill/>
  44                </div>
  45                <div className="absolute top-0 left-0 w-full h-full flex items-center justify-center text-white font-medium text-sm">
  46                  <span>30% off</span>
  47                </div>
  48              </a>
  49              <div>
  50                <span>20% off</span>
  51              </div>
  52            </div>
  53          ))
  54        </div>
  55      </section>
  56    );
  57  };
  58
  59  const OurProducts = () => {
  60    return <Our-Products />;
  61  };
  62
  63  export default OurProducts;
  
```



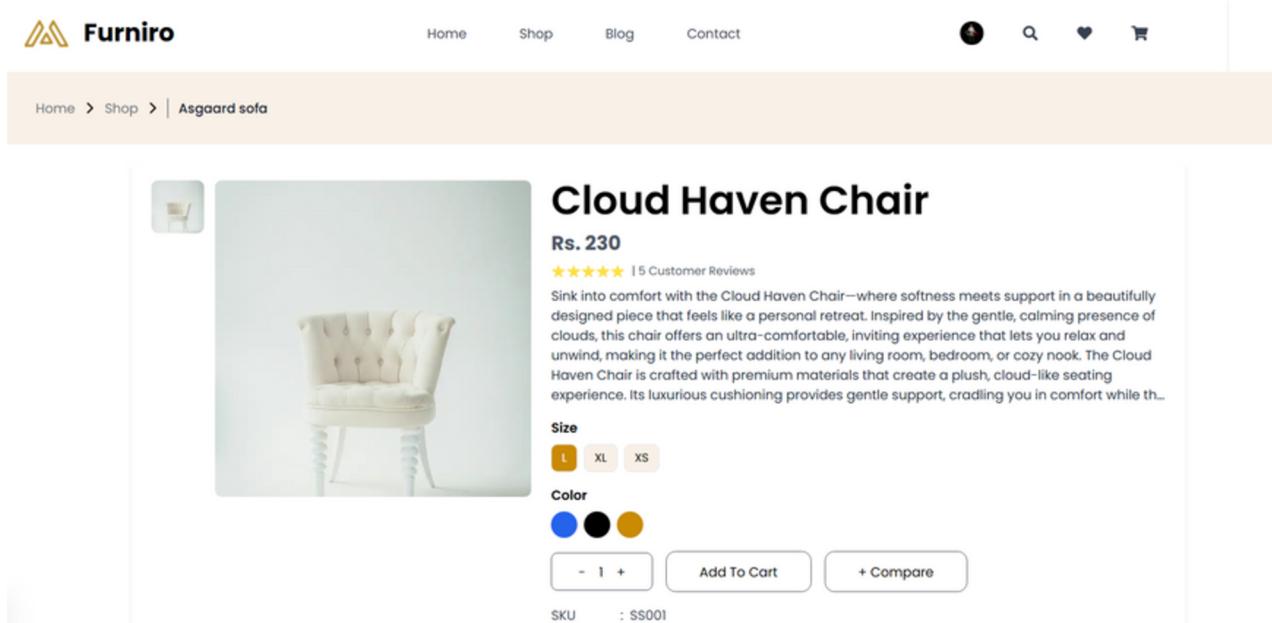
## Detailed Description:

- 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.
- 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.

The screenshot shows a 'Cart' page with a header 'Cart' and a breadcrumb 'Home > Cart'. Below the header is a blurred background image of a room with furniture. The main content area has two sections: a 'Product' table and a 'Cart Totals' summary.

Product	Price	Quantity	Subtotal	Action
Cloud Haven Chair	\$230	1	\$230	
Bright Space	\$180	1	\$180	
Slytherine	\$200	1	\$200	
Timber Craft	\$320	1	\$320	

**Cart Totals**

Subtotal	\$1820
Total	\$1820 \$1838.00

**CheckOut**

### 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.

<b>Billing details</b>	
First Name	Last Name
<input type="text"/>	<input type="text"/>
Company Name (Optional)	
<input type="text"/>	
Country / Region	
<input type="text"/> Sri Lanka	
Street Address	
<input type="text"/>	
Town / City	
<input type="text"/>	
Province	
<input type="text"/> Western Province	
ZIP Code	
<input type="text"/> ZIP Code	
Phone	
<input type="text"/> Phone	
Email Address	
<input type="text"/> Email Address	
<b>Product</b>	<b>Subtotal</b>
Asgaard sofa x 1	Rs. 250,000.00
<b>Subtotal</b>	<b>Rs. 250,000.00</b>
<b>Total</b>	<b>Rs. 250,000.00</b>
<b>Payment Method</b>	
<input checked="" type="radio"/> Direct Bank Transfer <input type="radio"/> Cash On Delivery	
<small>Your personal data will be used to support your experience throughout this website, to manage access to your account, and for other purposes described in our <a href="#">privacy policy</a>.</small>	
<b>Place order</b>	

### 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.

```

const Cart = () => {
  const handleRemoveItem = (id: number) => {
    console.log("Removing item with ID:", id); // Debug Log
    dispatch(removeFromCart(id));
  };

  return (
    <div className="flex flex-col lg:flex-row justify-center items-start p-5 lg:p-10">
      /* Cart Table */
      <div className="w-full lg:w-2/3 bg-white p-5 shadow-md rounded-md mb-5 lg:mb-0 lg:p-10">
        <table className="w-full text-left">
          <thead>
            <tr className="bg-[#F9F1E7]">
              <th>Product</th>
              <th>Price</th>
              <th>Quantity</th>
              <th>Subtotal</th>
            </tr>
          </thead>
          <tbody>
            {items.length === 0 ? (
              <tr>
                <td colSpan={5} className="text-center text-xl mt-10">
                  Your cart is empty
                </td>
              </tr>
            ) : (
              items.map((item) => {
                // Ensure valid and unique key
                const itemID = item.id ? item.id : Math.random();

                return (
                  <tr key={itemID}> /* Ensure the key is always valid */
                    <td className="p-2 md:p-3 flex items-center">
                      <Image

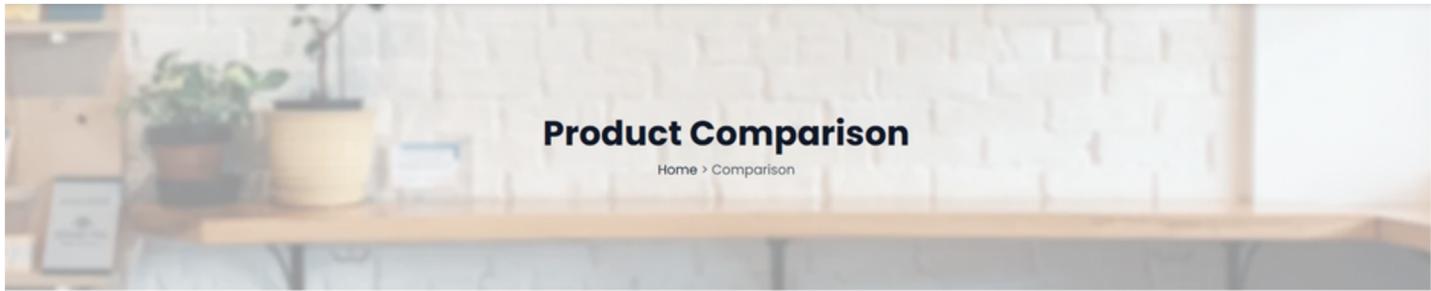
```

## 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.



Go to Product  
page for more  
Products

[View More](#)



Asgaard Sofa

Rs. 250,000.00

204 Reviews



Outdoor Sofa Set

Rs. 224,000.00

145 Reviews

Add A Product

Choose a Product

### 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 decision-making 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.

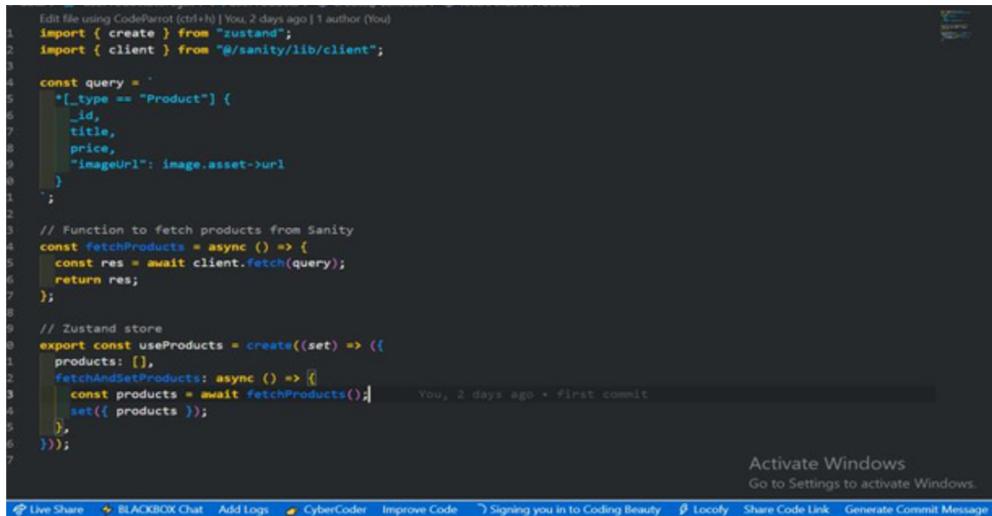
### Detailed Description:

- 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.



```

1 Edit file using CodeParrot (ctrl+h) | You, 2 days ago | 1 author (You)
2 import { create } from "zustand";
3 import { client } from "@sanity/lib/client";
4
5 const query = `
6   *[_type == "Product"] {
7     _id,
8     title,
9     price,
10    "imageUrl": image.asset->url
11  }
12 `;
13
14 // Function to fetch products from Sanity
15 const fetchProducts = async () => {
16   const res = await client.fetch(query);
17   return res;
18 };
19
20 // Zustand store
21 export const useProducts = create((set) => {
22   products: [],
23   fetchAndSetProducts: async () => [
24     const products = await fetchProducts();
25     set({ products });
26   ],
27 });
28
29 
```

Activate Windows  
Go to Settings to activate Windows.

Live Share   BLACKBOX Chat   Add Logs   CyberCoder   Improve Code   Signing you in to Coding Beauty   Locality   Share Code Link   Generate Commit Message

### 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!