

# Day:4 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:

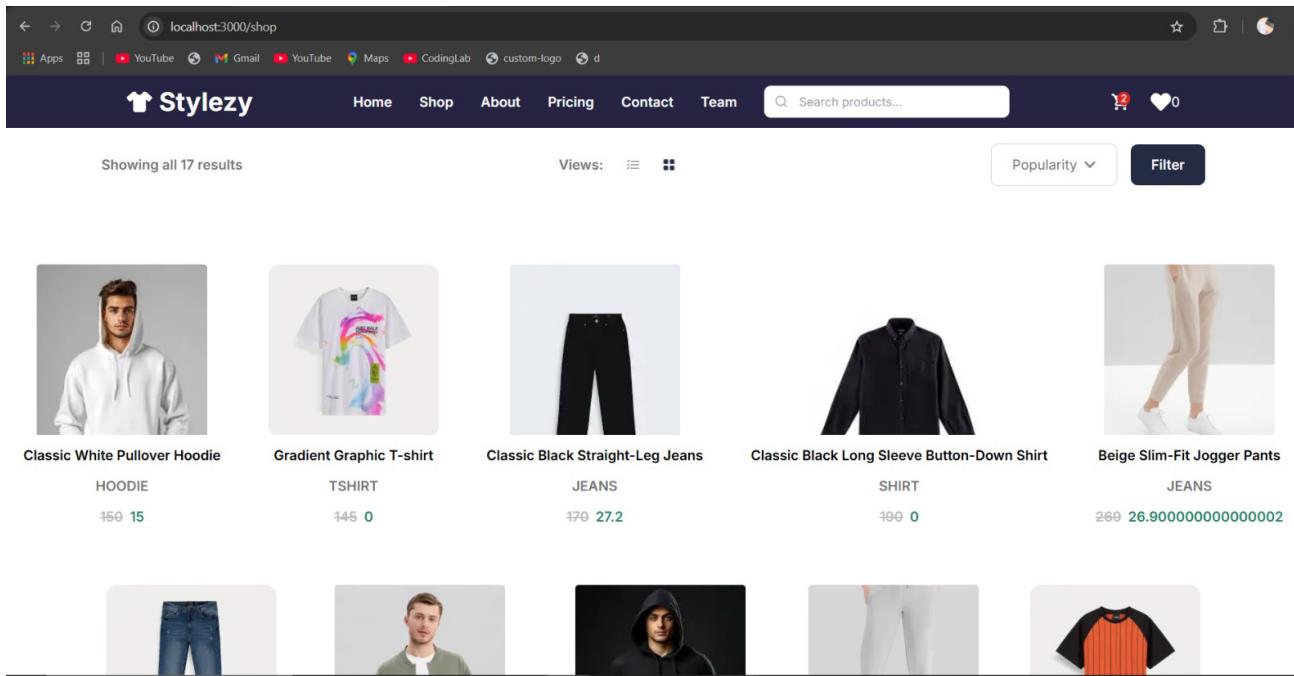
1. Product Listing Page
2. Dynamic Route
3. Cart Functionality
4. Checkout
5. Filter
6. Search
7. Price Calculation
8. WishList

Each functionality contributes to building a responsive and scalable marketplace.

## Step 2: Functionalities in Detail

### ProductListing 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.



### Detailed Description:

The page offers sorting and filtering options to enhance usability, allowing users to organize products based on price, categories, or popularity.

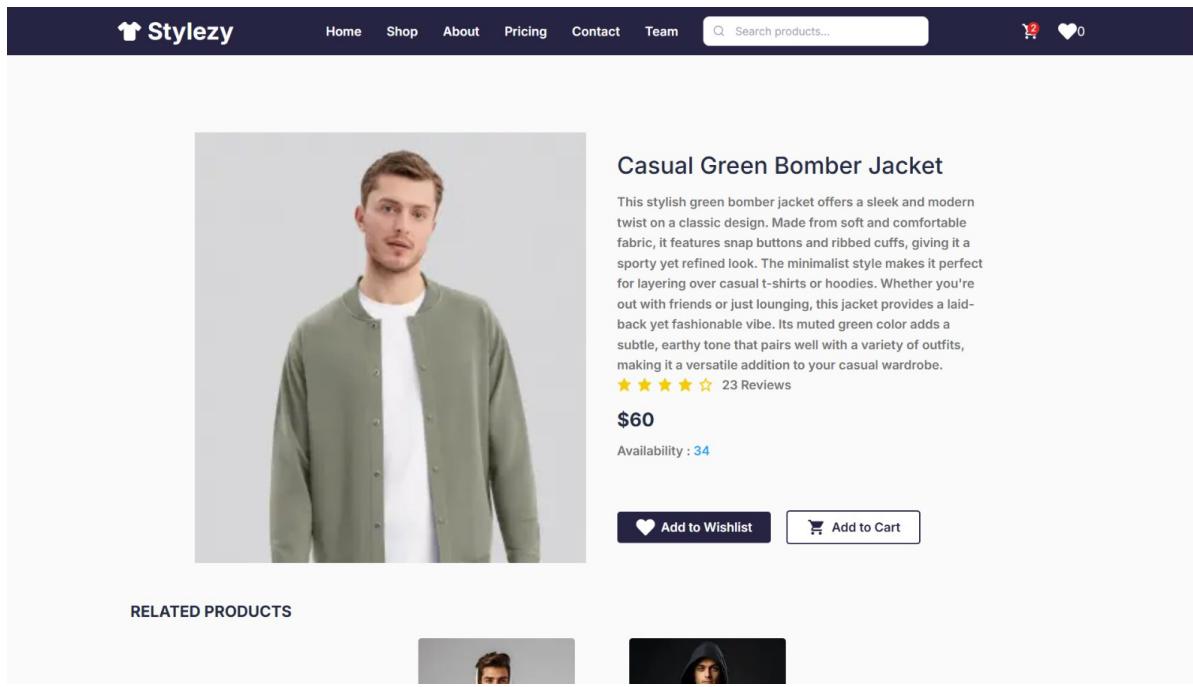
Pagination ensures the seamless handling of large datasets, improving performance and user experience.

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.

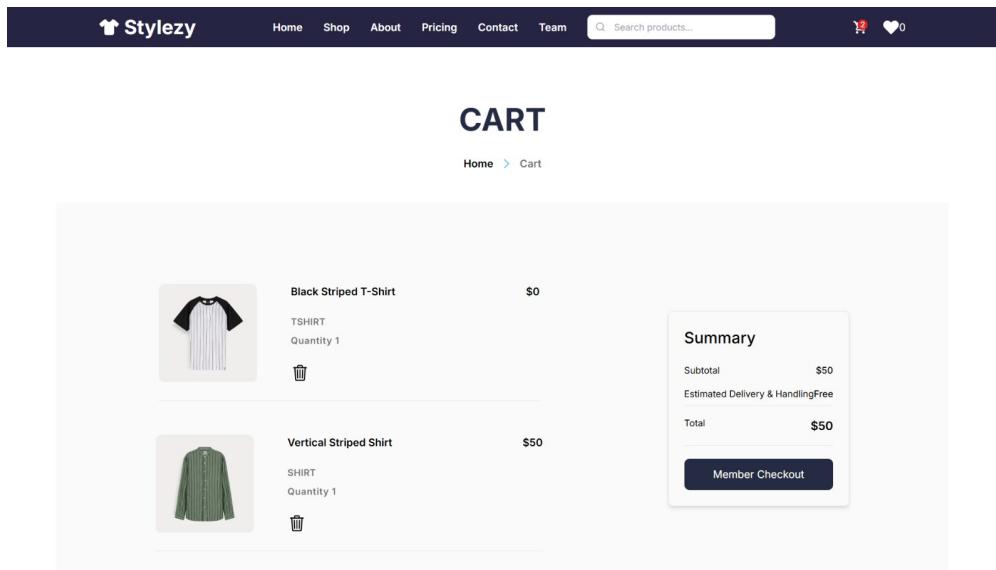


### Detailed Description:

- 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.

A screenshot of the Stylezy website's cart page. At the top, there is a dark header bar with the logo 'Stylezy' (a t-shirt icon followed by the word 'Stylezy'), navigation links for Home, Shop, About, Pricing, Contact, Team, a search bar with placeholder text 'Search products...', and a user icon with a heart and the number '0'. Below the header, the page title 'CART' is centered. Underneath it, a breadcrumb trail shows 'Home > Cart'. The main content area displays two items in the cart: 'Black Striped T-Shirt' (TSHIRT, Quantity 1) and 'Vertical Striped Shirt' (SHIRT, Quantity 1). Each item has a small thumbnail image, the product name, category, quantity, and price (\$0 and \$50 respectively). To the right of the items is a 'Summary' box containing a table with three rows: Subtotal (\$50), Estimated Delivery & Handling Free, and Total (\$50). A 'Member Checkout' button is located at the bottom of the summary box. The overall design is clean and modern, using a light gray background for the main content area.

#### 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.

The screenshot shows a two-column checkout interface. The left column contains sections for 'Contact Information' (with a 'Log in' button and an email input field), 'Delivery Details' (with fields for First Name, Last Name, Address, City, Postal Code, and Phone), and a summary at the bottom. The right column displays a shopping cart with two items: 'Black Striped T-Shirt' and 'Vertical Striped Shirt', both listed as 'TSHIRT' with a quantity of 1 and a price of \$50. A summary table at the bottom right shows Subtotal (\$50), Estimated Delivery & Handling (Free), and Total (\$50). A 'Member Checkout' button is located at the bottom of the right column.

## Detailed Description:

payment information.

- The checkout process is divided into multiple steps: billing details, shipping address, and
- 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.

src > app > cart > page.tsx > Page

```
7 const Page = () => {
8   </>
9   </div>
10  </div>
11
12  {cartItems.length > 0 ? (
13    <div className="bg-[#fafafa] py-24 w-[90%] mx-auto flex flex-col md:flex-row justify-center items-center gap-12 md:gap-12 lg:gap-36 xl:gap-48">
14      <div className="flex justify-center items-center flex-col gap-6">
15          {cartItems.map((item: CartItem) => (
16              <CartCard key={item._id} item={item} />
17          )))
18      </div>
19      {cartItems && [
20        <div className="border rounded-lg p-6 shadow-md">
21          <h4 className="text-2xl font-medium mb-4">Summary</h4>
22          <div className="mt-3 text-sm">
23              <div className="flex justify-between py-2">
24                  <span>Subtotal</span>
25                  <span className="font-medium" ${cartTotal}>${cartTotal}</span>
26              </div>
27              <div className="flex justify-between py-2">
28                  <span>Estimated Delivery & Handling</span>
29                  <span className="font-medium">Free</span>
30              </div>
31              <div className="flex justify-between py-4 border-t border-b bg-gray-200">
32                  <span>Total</span>
33                  <span className="text-lg font-semibold" ${cartTotal}>${cartTotal}</span>
34              </div>
35          </div>
36      </div>
37      <Link
38          href="/checkout"
39          className="block w-full mt-5 px-4 py-3 text-center bg-white bg-myHeading rounded-lg hover:bg-myBlue-dark transition-all duration-300"
40      >
41          Member Checkout
42      </Link>
43  ]}>
44  </div>
45  <div>
46      <img alt="Cart icon" />
47      <span>1</span>
48  </div>
49
```

## **Detailed Description:**

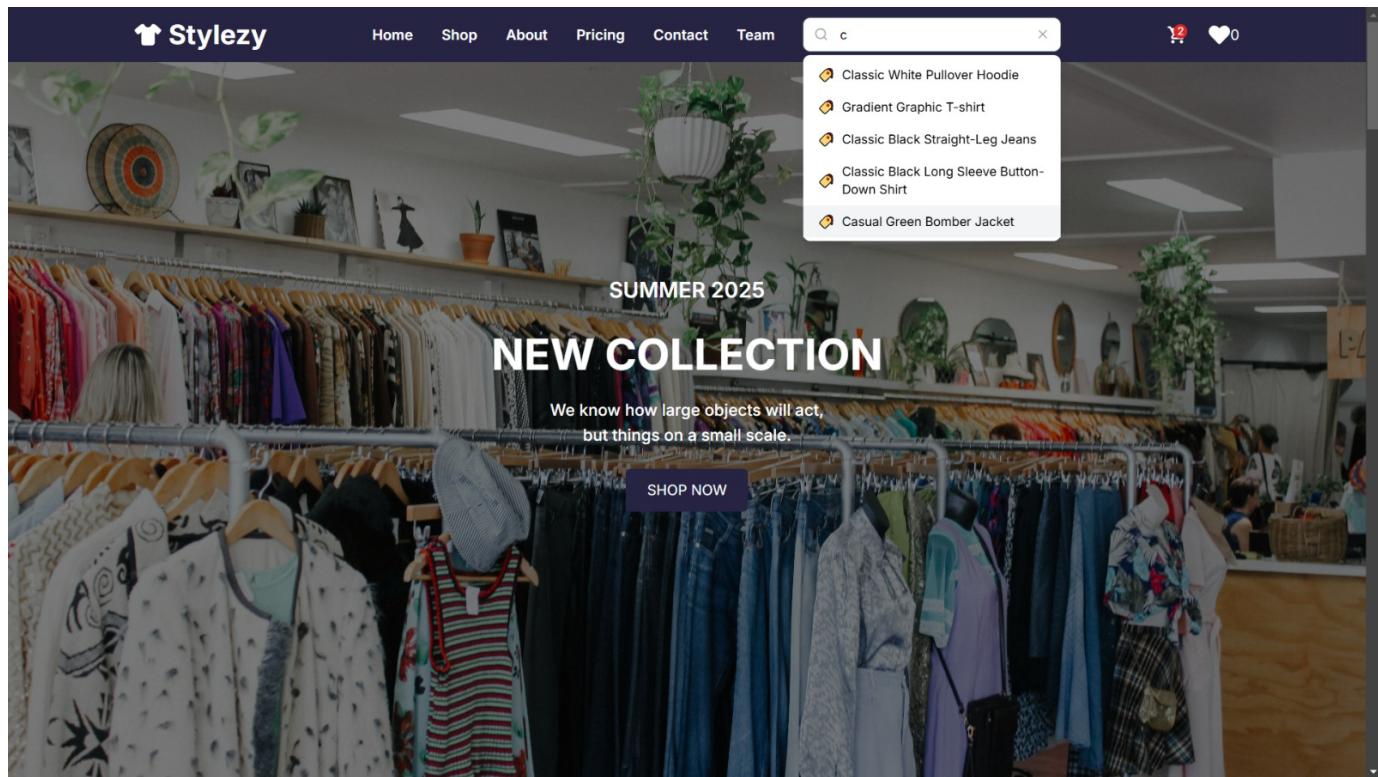
The subtotal updates in real adjusted.

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. Search



```
Run Terminal Help ⌘P E-commerce-marketplace-hackathon
page.tsx ...\\cart M page.tsx ...\\search X
app > search > page.tsx > [SearchPage]
const SearchPage = () => {
  useEffect(() => {
    const fetchSearchResults = async (): Promise<void> => {
      setLoading(true);
      try {
        const results = products?.filter(
          (product) =>
            product.name.toLowerCase().includes(query.toLowerCase()) ||
            product.tags.some((tag) =>
              tag.toLowerCase().includes(query.toLowerCase())
            )
        );
        setSearchResults(results || []);
      } catch (error) {
        console.error("Error fetching search results:", error);
      } finally {
        setLoading(false);
      }
    };
    fetchSearchResults();
  }, [query, products]);
  if (loading) {
    return (
      <div className="min-h-screen p-8">
        <div className="max-w-7xl mx-auto">
          <p className="text-center text-gray-500">Loading...</p>
        </div>
      </div>
    );
  }
  return (
    <div className="pt-40 bg-[#fafafa] text-black">
      /* Search header */
    </div>
  );
}

```

## 7. Filter

The screenshot shows the Stylezy website interface. At the top, there's a navigation bar with links for Home, Shop, About, Pricing, Contact, Team, and a search bar. Below the navigation is a grid of categories: PENT (4 items), QUATER SHORT (items), HODDIE (1 items), and another category with 5 items. The main content area displays 17 results, with a sorting dropdown set to "Price: Low to High" and a "Filter" button. On the right, there's a sidebar with filtering options: Popularity, Price: Low to High, Price: High to Low, and Newest First. Below the sidebar, five products are listed with their images, names, and details:

Image	Name	Type	Details
	LOOSE FIT BERMUDA SHORTS	SHORT	15.00 - 60.00
	Black Striped T-Shirt	TSHIRT	15.00 - 60.00
	Classic Black Pullover Hoodie	HOODIE	15.00 - 60.00
	Sleeve Stripe T-Shirt	TSHIRT	15.00 - 60.00
	Gradient Graphic T-shirt	TSHIRT	15.00 - 60.00

The screenshot shows a code editor with a dark theme, displaying the `Filtering.tsx` component. The code uses React functional components and hooks like `useState` and `useEffect`. It manages state for filters, sort order, and price range, and handles user interactions like clicks outside the modal.

```
src > components > filter > Filtering.tsx > Filtering > useEffect() callback
14 const Filtering: React.FC<FilterProps> = ({{
15   const [showFilterModal, setShowFilterModal] = useState(false);
16   const sortRef = useRef<HTMLDivElement>(null);
17
18   const [filters, setFilters] = useState<FilterState>(() => ({
19     sortBy: "popularity",
20     priceRange: {
21       min: 0,
22       max: 1000,
23     },
24     inStock: false,
25   }));
26   useEffect(() => [
27     setIsClient(true),
28   ], []);
29
30   useEffect(() => {
31     if (products.length > 0) {
32       const prices = products.map((p) => p.price);
33       const maxPrice = Math.max(...prices);
34       setFilters((prev) => ({
35         ...prev,
36         priceRange: {
37           ...prev.priceRange,
38           max: maxPrice,
39         },
40       }));
41     }
42   }, [products]);
43
44   useEffect(() => {
45     const handleClickOutside = (event: MouseEvent) => {
46       if (sortRef.current && !sortRef.current.contains(event.target as Node)) {
47         setShowSortDropdown(false);
48       }
49     };
50   });
51
52   const handleSortChange = (sortType: string) => {
53     setShowSortDropdown(true);
54   };
55 }}
```

## 8. WishList

The screenshot shows the 'WishList' section of the Stylezy website. At the top, there's a navigation bar with links for Home, Shop, About, Pricing, Contact, Team, a search bar, and icons for a shopping cart (2 items) and a heart (3 items). Below the navigation is a large title 'WishList'. Underneath it, a breadcrumb navigation shows 'Home > wishlist'. Three product cards are displayed in a grid:

- Casual Green Bomber Jacket**: An image of a person wearing a green bomber jacket over a white t-shirt. A red trash bin icon is in the top right corner of the image.
- Classic White Pullover Hoodie**: An image of a person wearing a white pullover hoodie. A red trash bin icon is in the top right corner of the image.
- Classic Black Pullover Hoodie**: An image of a person wearing a black pullover hoodie. A red trash bin icon is in the top right corner of the image.

Below the cards, the 'Stylezy' logo is centered, and social media sharing icons for Facebook, Instagram, and Twitter are on the right.

## 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!