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:

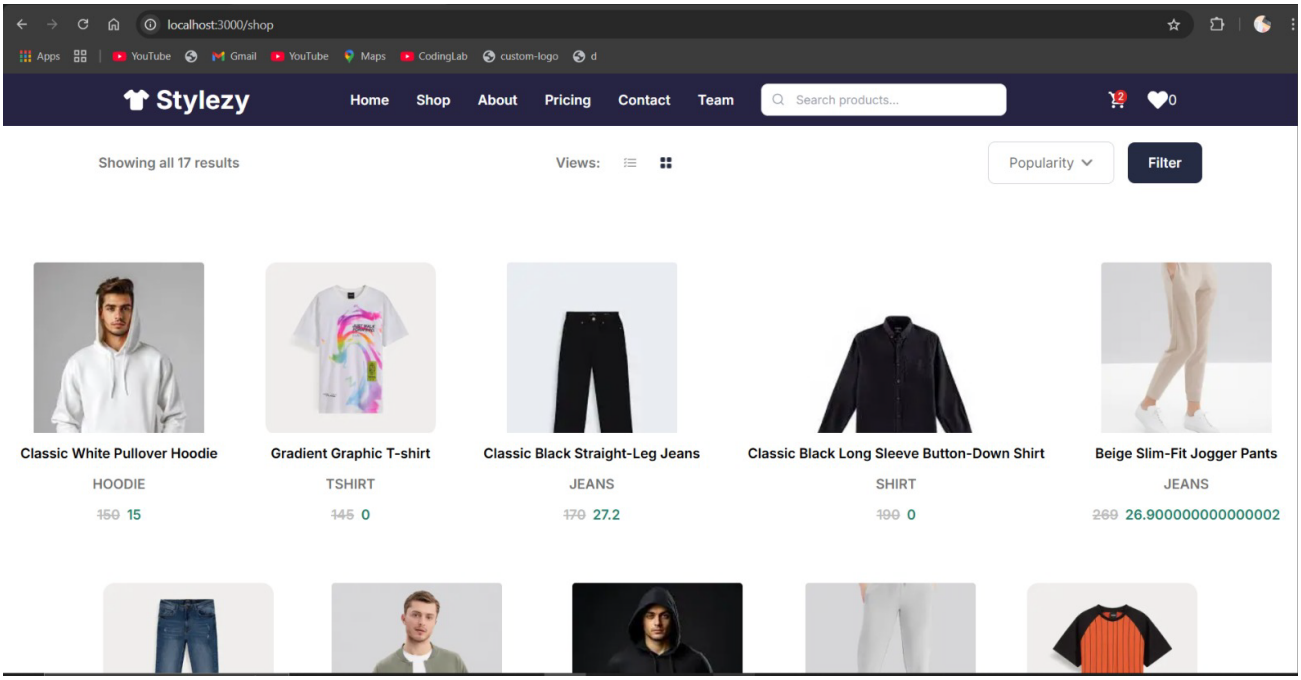
1. Product Listing Page
2. Dynamic Route
3. Cart Functionality
4. Checkout
5. Filter
6. Search
7. Price Calculation
8. WishList

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.

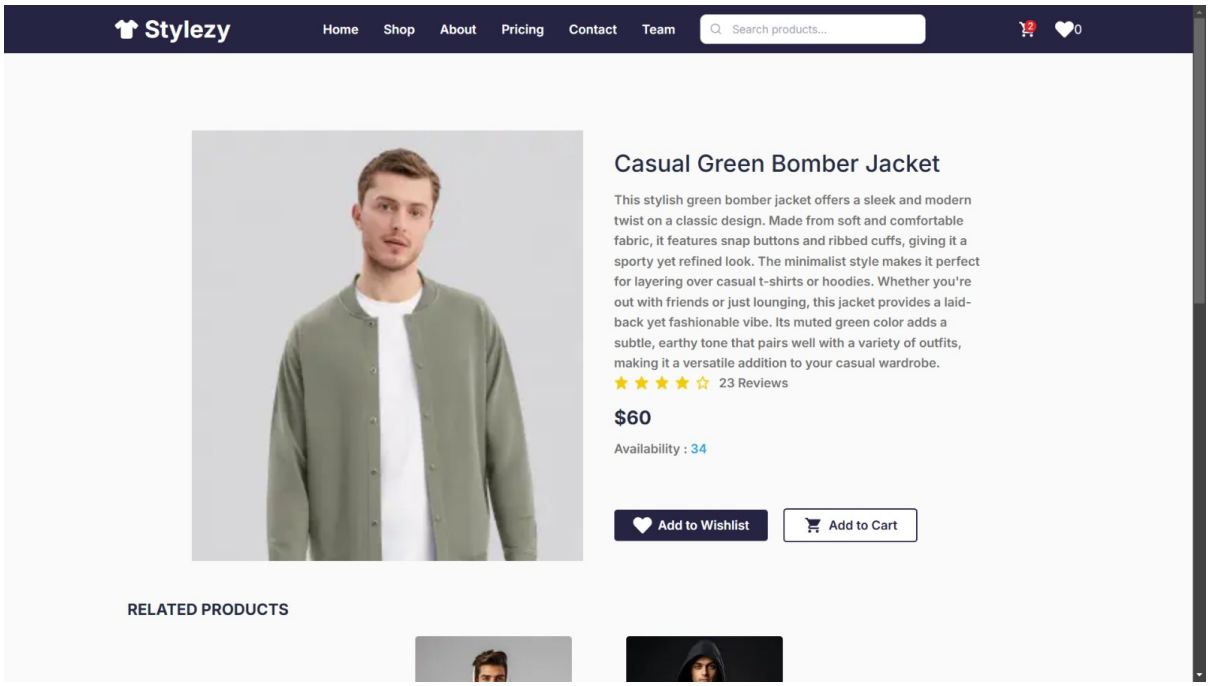
Pagination ensures the seamless handling of large datasets, improving performance and user experience.

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.

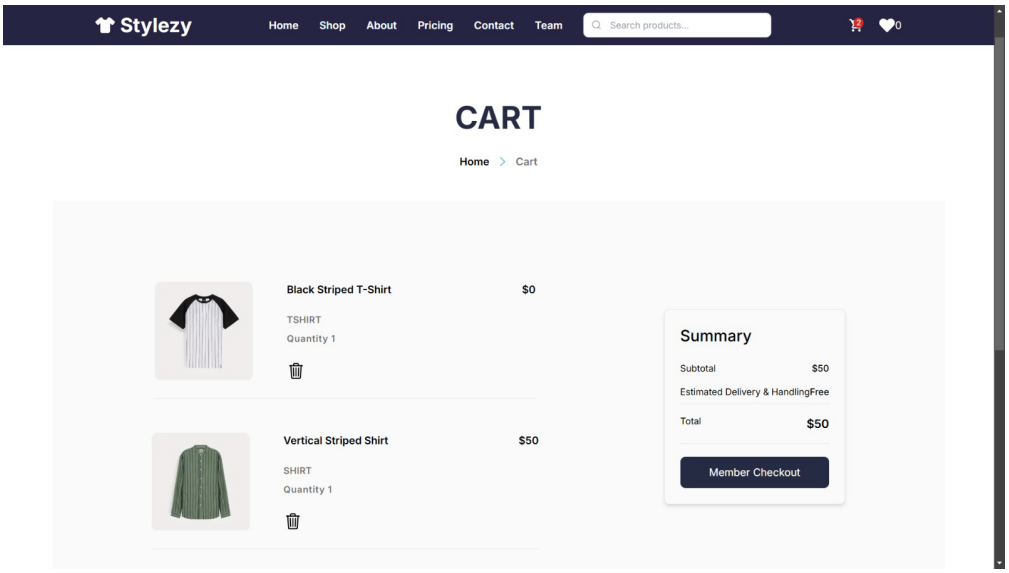


Detailed Description:

- 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.

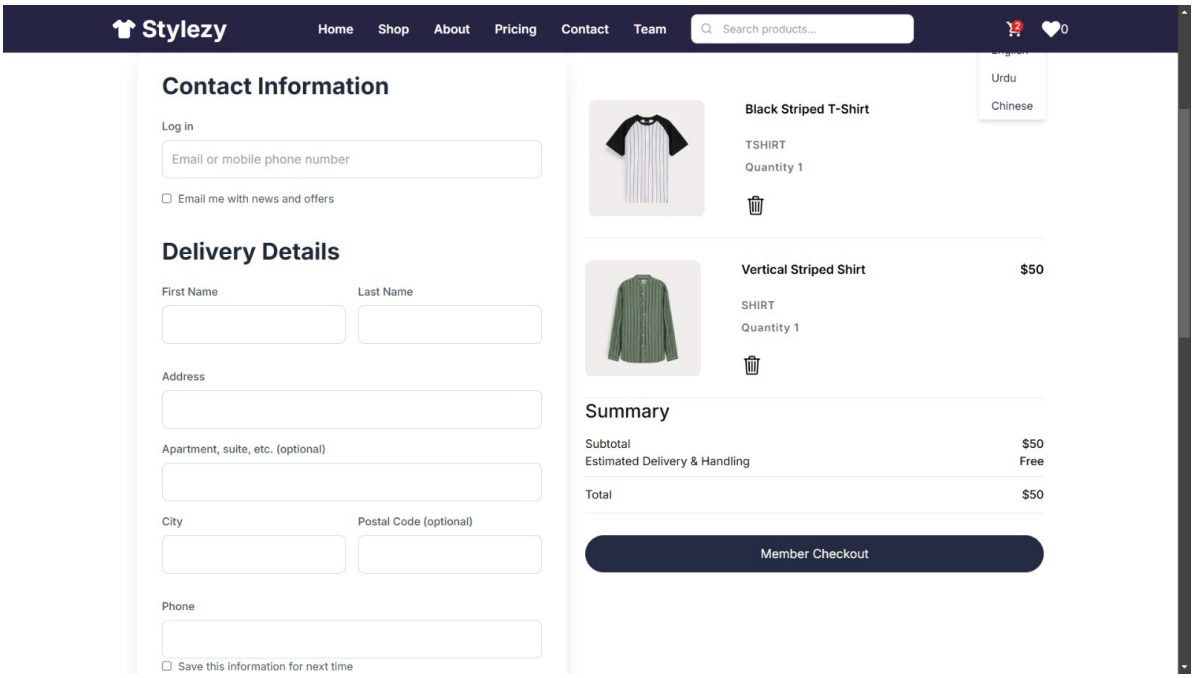


Detailed Description:

- 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.



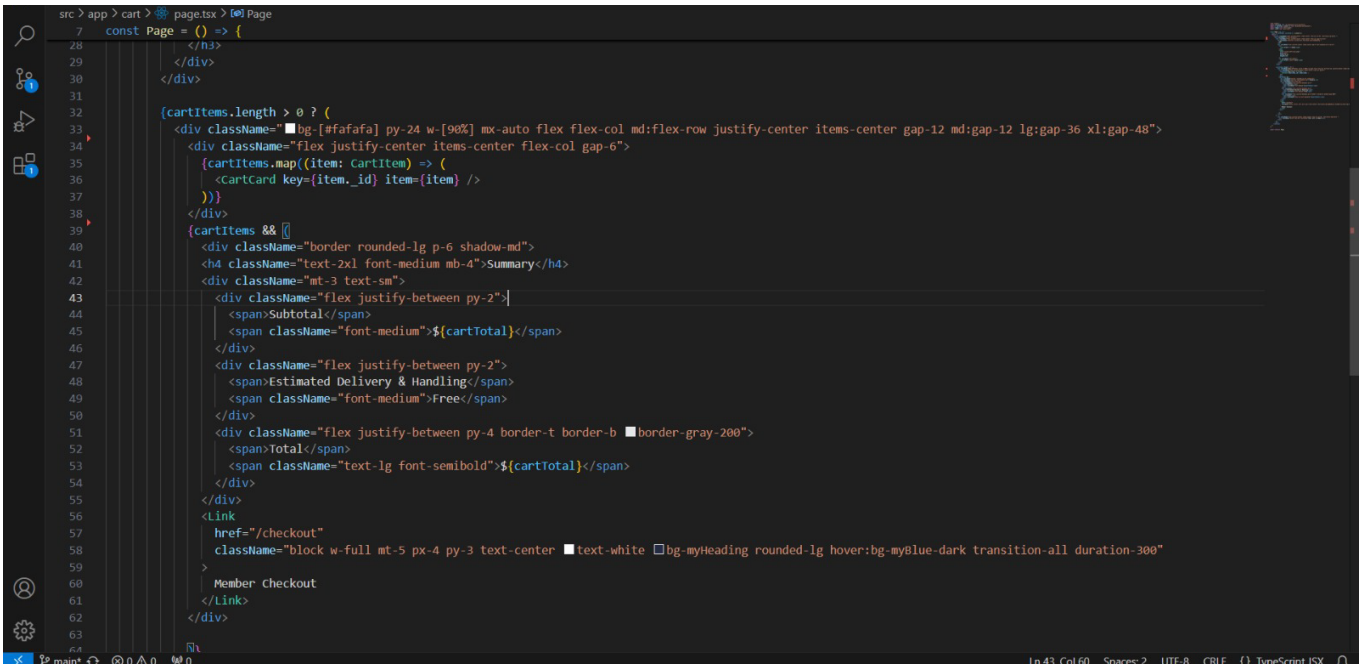
Detailed Description:

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

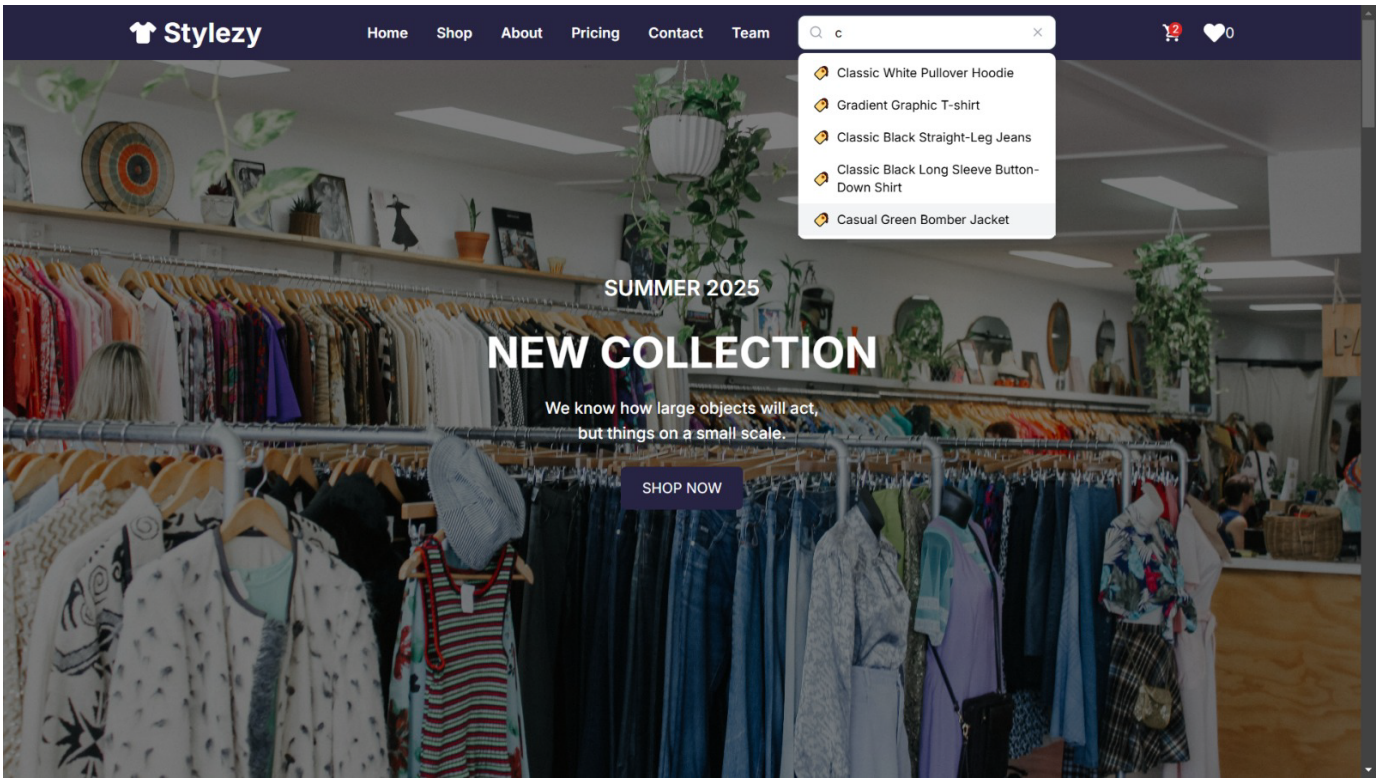
Price Calculation dynamically computes the total cost of items in the cart, factoring in taxes, discounts, and other adjustments.



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.

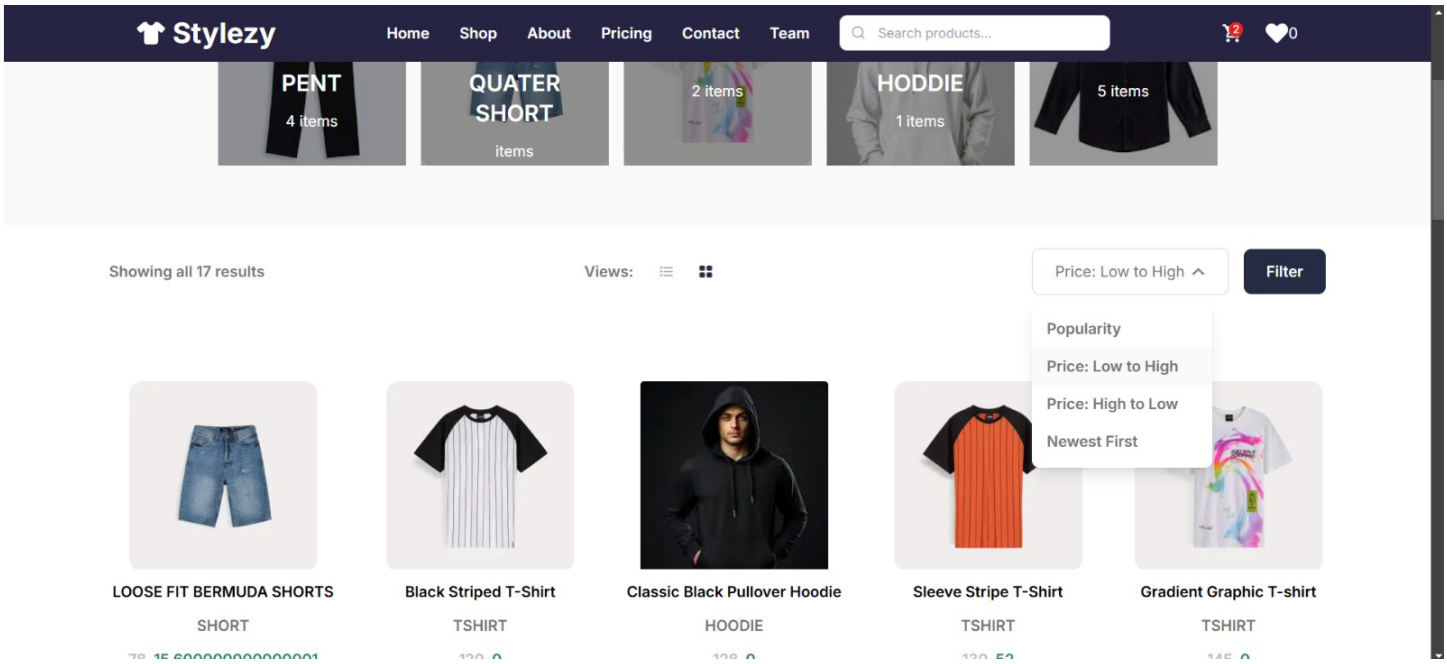
6. Srearch

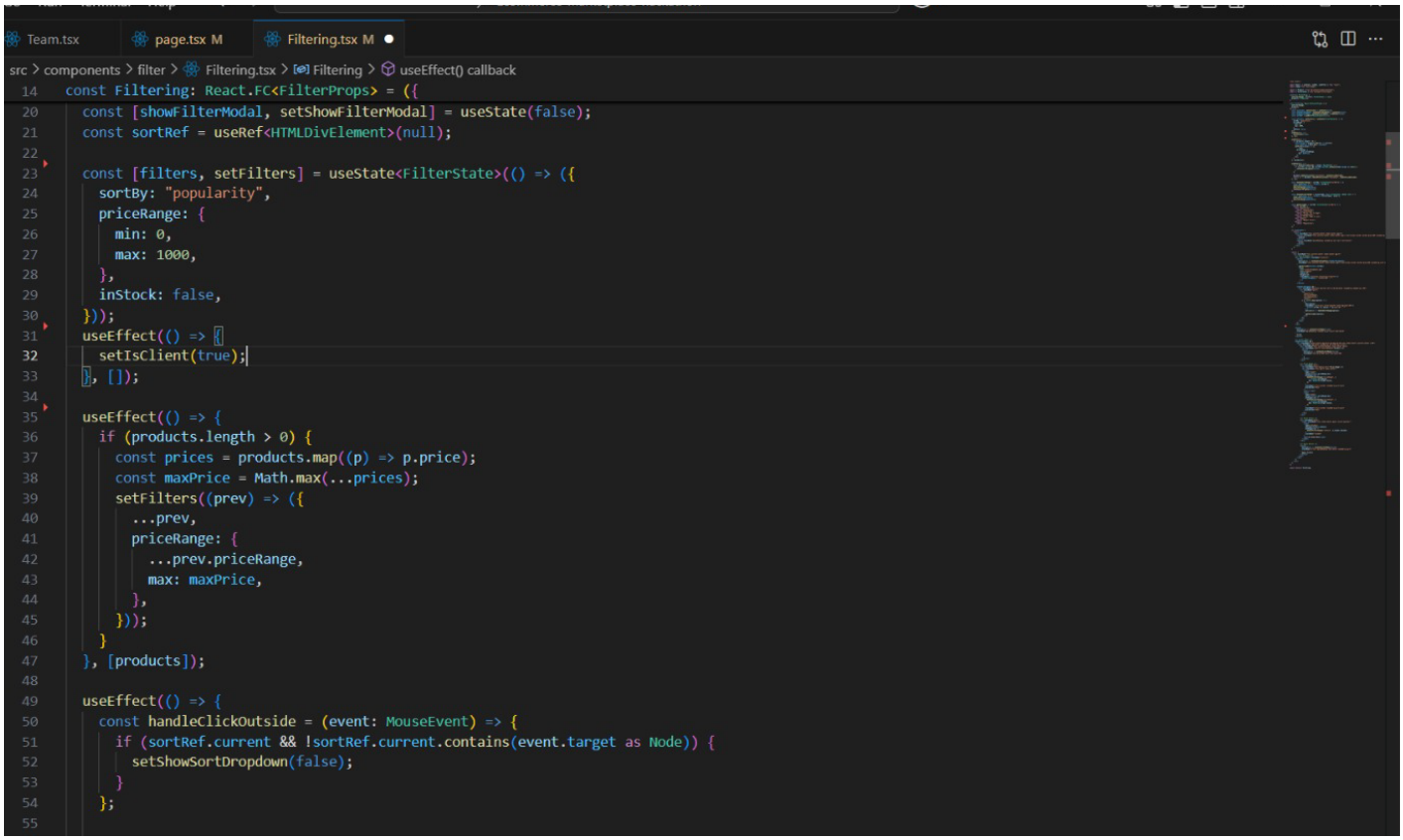


-

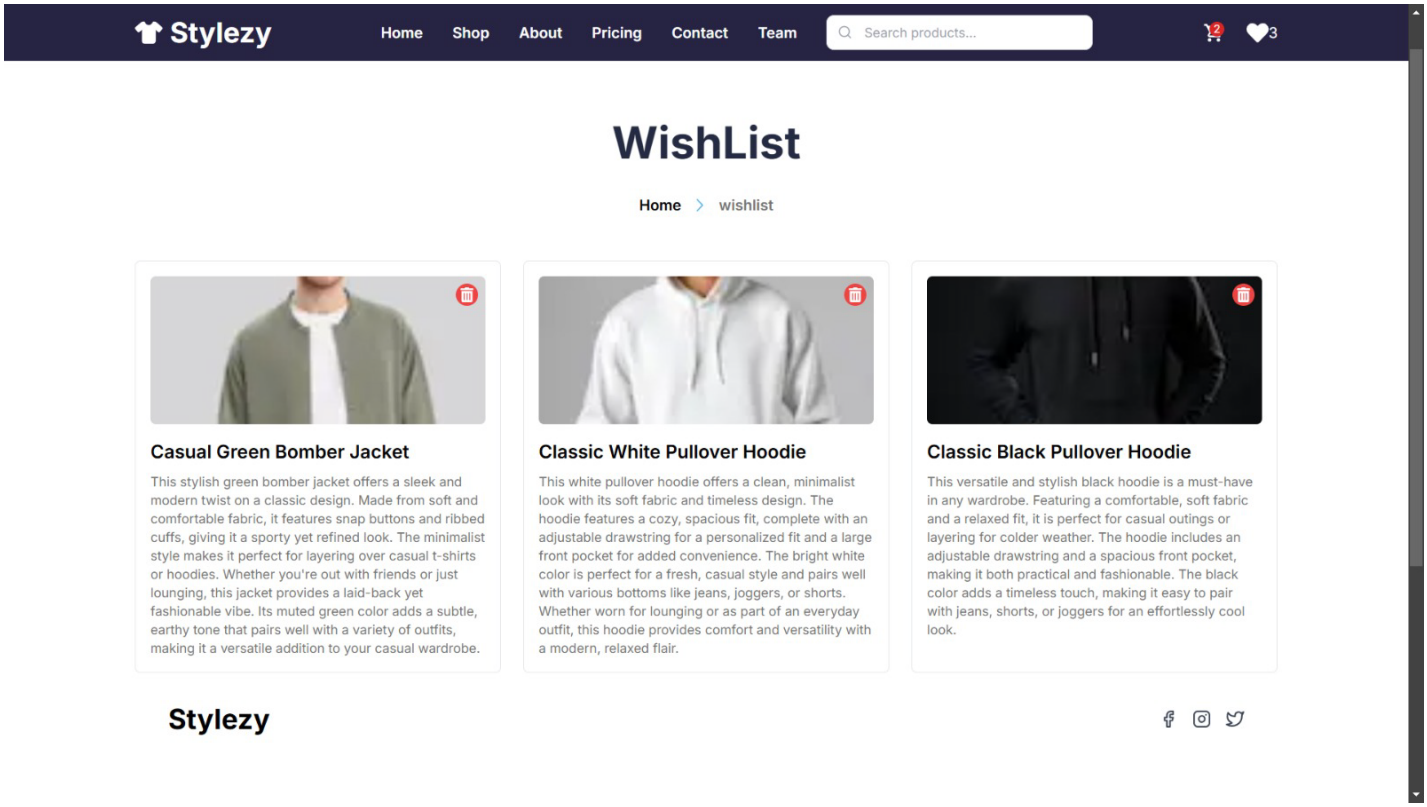
```tsx
const SearchPage = () => {
  useEffect(() => {
    const fetchSearchResults = async (): Promise<void> => {
      setLoading(true);
      try {
        const results = products?.filter(
          (product) =>
            product.name.toLowerCase().includes(query.toLowerCase()) ||
            product.tags.some((tag) =>
              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 */}
```

7. Filter


Stylezy
Home
Shop
About
Pricing
Contact
Team
Search products...
PENT
4 items
QUATER SHORT
items
2 items
HODDIE
1 items
5 items
Showing all 17 results
Views:
Price: Low to High
Filter
Popularity
Price: Low to High
Price: High to Low
Newest First
LOOSE FIT BERMUDA SHORTS
SHORT
Black Striped T-Shirt
TSHIRT
Classic Black Pullover Hoodie
HOODIE
Sleeve Stripe T-Shirt
TSHIRT
Gradient Graphic T-shirt
TSHIRT


Team.tsx
page.tsx M
Filtering.tsx M
src > components > filter > Filtering.tsx > Filtering > useEffect() callback
const Filtering: React.FC<FilterProps> = ({
const [showFilterModal, setShowFilterModal] = useState(false);
const sortRef = useRef<HTMLDivElement>(null);
const [filters, setFilters] = useState<FilterState>(() => ({
sortBy: "popularity",
priceRange: {
min: 0,
max: 1000,
},
inStock: false,
}));
useEffect(() => {
setIsClient(true);
}, []);
useEffect(() => {
if (products.length > 0) {
const prices = products.map((p) => p.price);
const maxPrice = Math.max(...prices);
setFilters((prev) => ({
...prev,
priceRange: {
...prev.priceRange,
max: maxPrice,
},
}));
}
}, [products]);
useEffect(() => {
const handleClickOutside = (event: MouseEvent) => {
if (sortRef.current && !sortRef.current.contains(event.target as Node)) {
setShowSortDropdown(false);
}
};

8. WishList



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!