Day 4 - Dynamic Frontend Components Documentation

Overview

On Day 4, I focused on designing and developing dynamic frontend components for a marketplace application. These components retrieve data dynamically from **Sanity CMS** or APIs, ensuring a modular and scalable architecture.

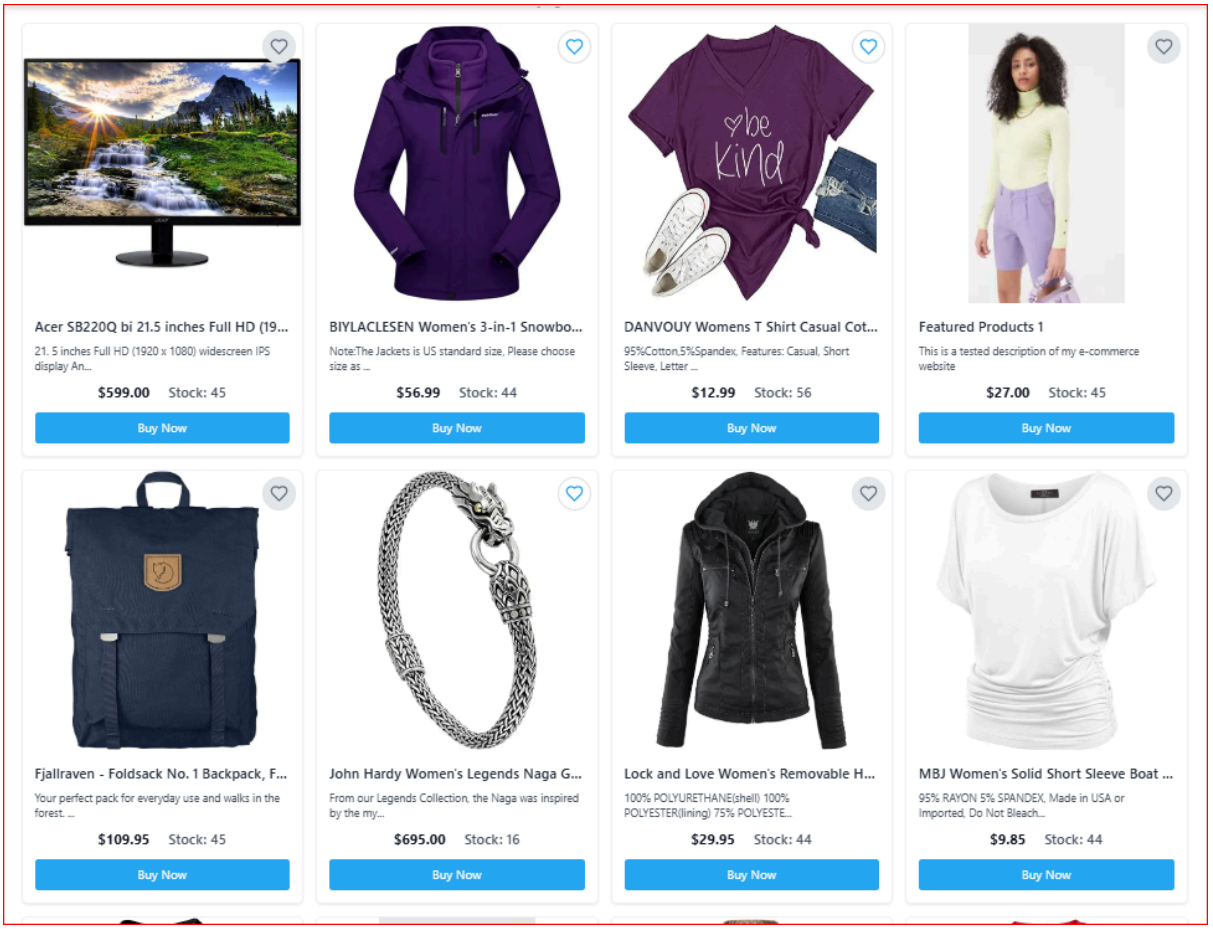
Completed Components

1. Product Listing Component

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.

- Dynamically renders products in a grid layout.
- Includes key fields:
    - **Product Name**
    - **Price**
    - **Image**
    - **Stock Status**
- Implemented using **Next.js dynamic routing**.
- Styled with **Tailwind CSS** for responsiveness.
- Integration with **Sanity CMS** ensures real-time updates, so any product changes in the backend are instantly reflected.

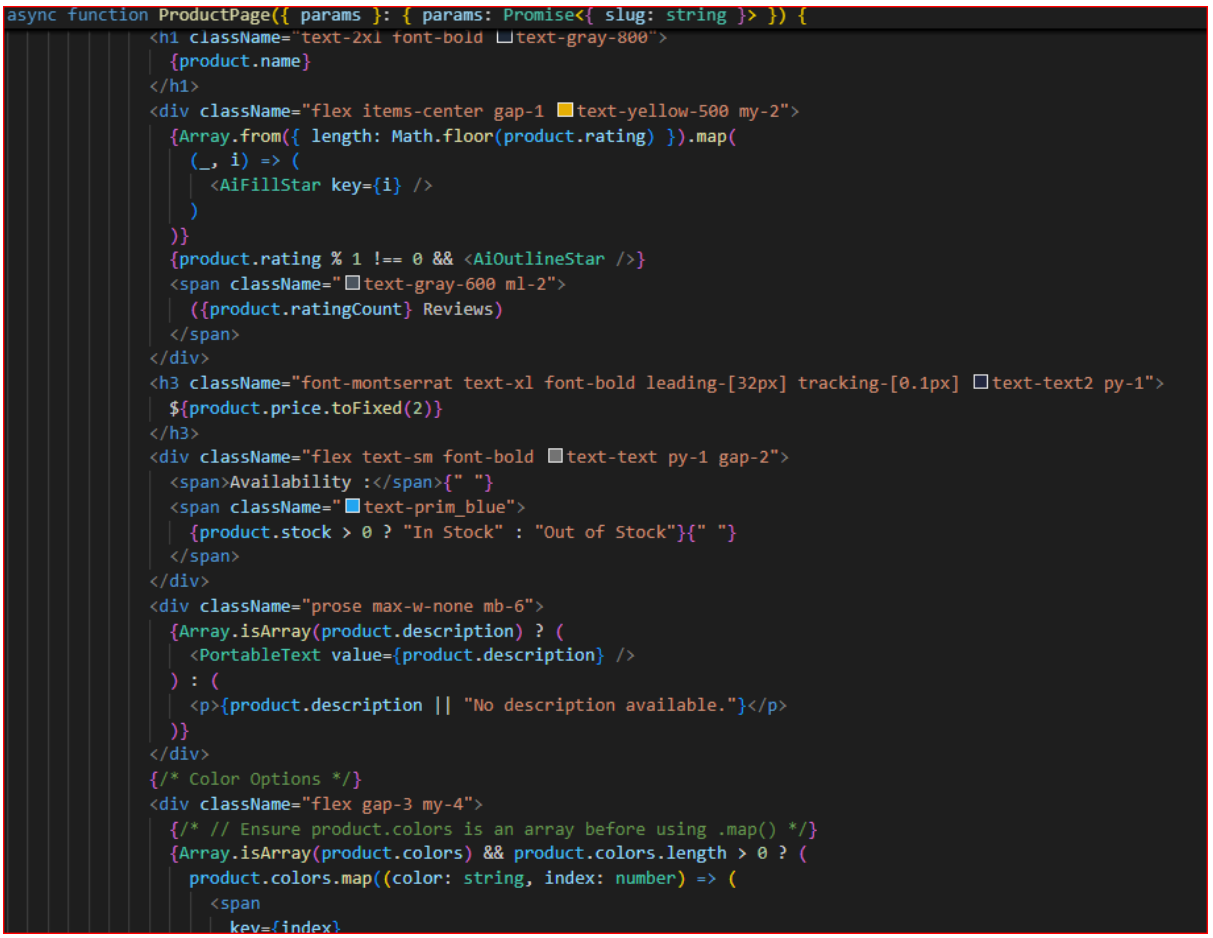
```jsx
const ProductGrid = ({products}: {products:Product[]}) => {
        <p className="font-montserrat text-[14px] font-normal text-text leading-[20px] t
          Problems trying to resolve the conflict between
        </p>
      </div>
    <div className="grid  grid-cols-1 sm:grid-cols-3 lg:grid-cols-4 mt-4 gap-4">
      {products?.map((product) => {
       return(
          <AnimatePresence key={product._id}>
            <motion.div
            layout
            initial={{opacity: 0.2}}
            animate={{opacity: 1}}
            exit={{opacity: 0}}
            className="flex justify-center w-full"
            >
            <ProductThumb key={product._id} product={product} />
            </motion.div>
          </AnimatePresence>
       )
      })}
    </div>
    </section>
  )
}

export default ProductGrid;
```



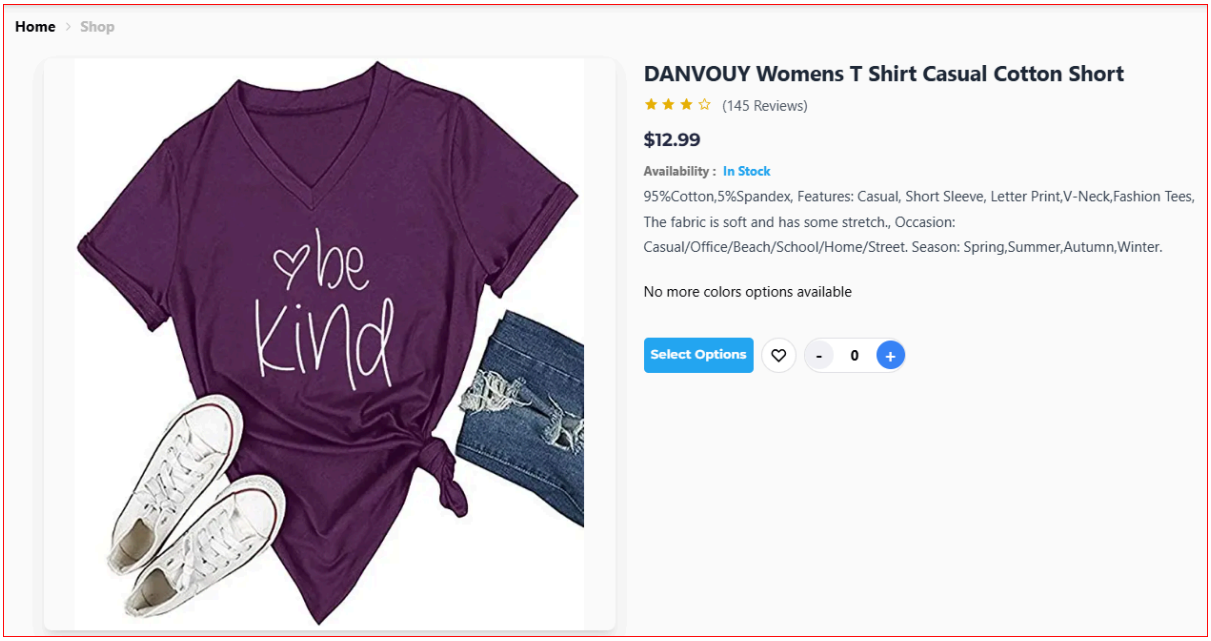
2. Product Detail Component

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

- Displays individual product details using **dynamic routing** (`/product/[slug]`).
- Includes:
    - **Product Description**
    - **Rating**
    - **Price**
    - **Available Sizes/Colors**
    - **Add to Cart** and **Wishlist** buttons.
- Uses API integration to fetch product data dynamically.
- Responsive design ensures compatibility across devices, from desktops to mobile phones.



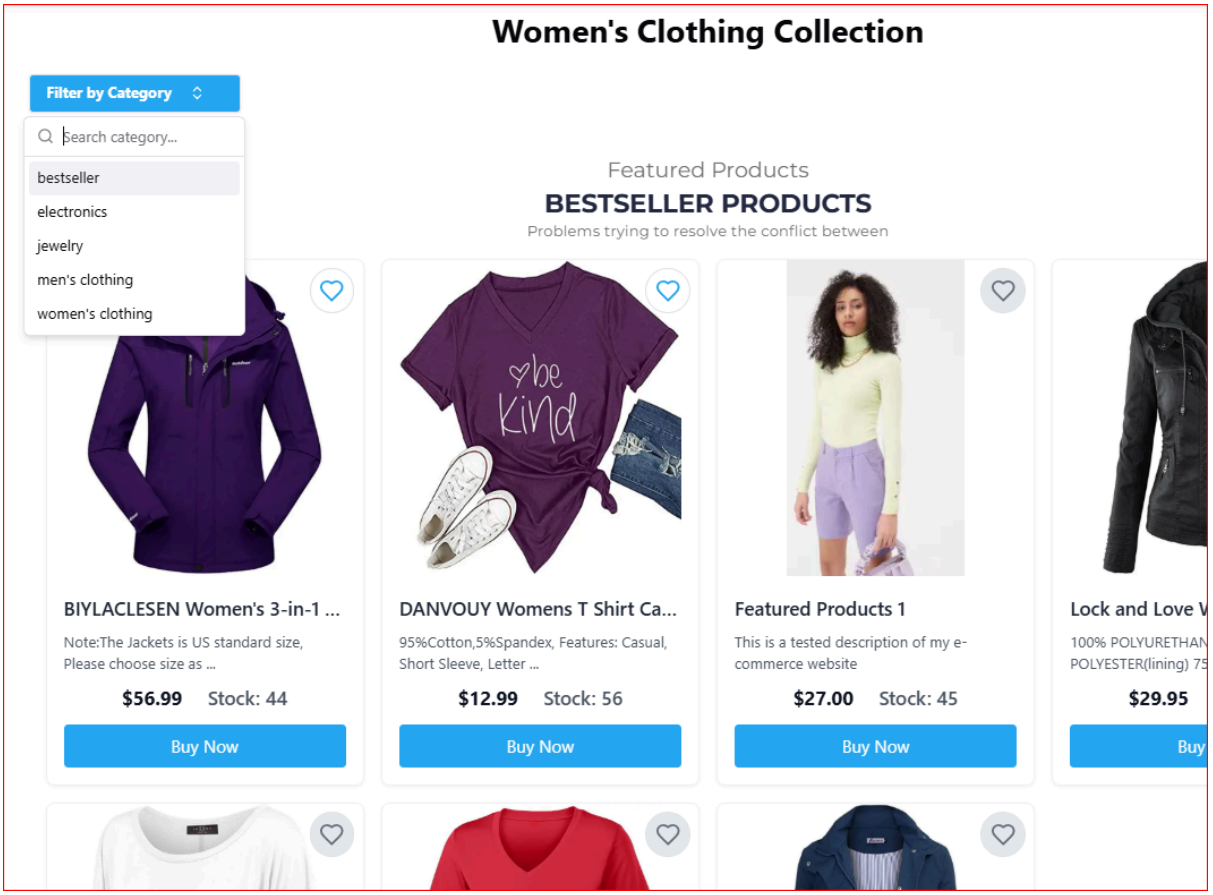
```
async function ProductPage({ params }: { params: Promise<{ slug: string }> }) {
            <h1 className="text-2xl font-bold text-gray-800">
              {product.name}
            </h1>
            <div className="flex items-center gap-1 text-yellow-500 my-2">
              {Array.from({ length: Math.floor(product.rating) }).map(
                (_, i) => (
                  <AiFillStar key={i} />
                )
              )}
              {product.rating % 1 !== 0 && <AiOutlineStar />}
              <span className="text-gray-600 ml-2">
                ({product.ratingCount} Reviews)
              </span>
            </div>
            <h3 className="font-montserrat text-xl font-bold leading-[32px] tracking-[0.1px] text-text2 py-1">
              ${product.price.toFixed(2)}
            </h3>
            <div className="flex text-sm font-bold text-text py-1 gap-2">
              <span>Availability :</span>{" "}
              <span className="text-prim_blue">
                {product.stock > 0 ? "In Stock" : "Out of Stock"}{" "}
              </span>
            </div>
            <div className="prose max-w-none mb-6">
              {Array.isArray(product.description) ? (
                <PortableText value={product.description} />
              ) : (
                <p>{product.description || "No description available."}</p>
              )}
            </div>
            {/* Color Options */}
            <div className="flex gap-3 my-4">
              {/* // Ensure product.colors is an array before using .map() */}
              {Array.isArray(product.colors) && product.colors.length > 0 ? (
                product.colors.map((color: string, index: number) => (
                  <span
                    key={index}
```



3. Category Component

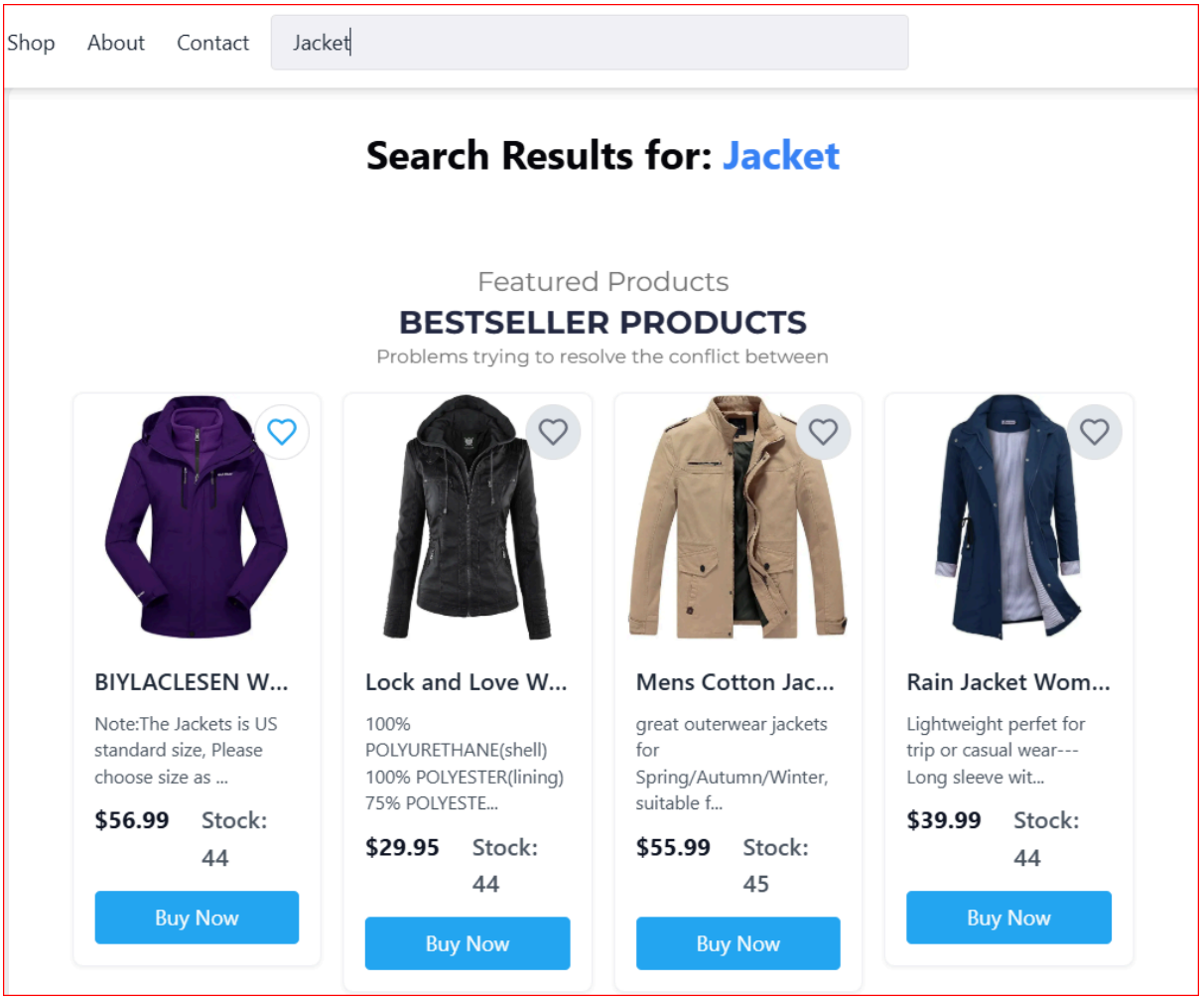
- Displays product categories dynamically.
- Allows filtering products by category.
- Uses API calls to fetch category data.

```
const CategoryPage: React.FC<CategoryPageProps> = async ({ params }) => {

  try {
    const [products, categories] = await Promise.all([
      getProductsCategory(slug),
      getAllCategories(),
    ]);

    return (
      <div className="flex flex-col items-center w-full justify-top max-w-screen-2xl min-h-screen mx-auto bg-gray-100 p-4">
        <div className="bg-white p-8 rounded-lg shadow-md w-full">
          <h1 className="text-3xl font-bold mb-6 text-center">
            {formatTitle(slug)} Collection
          </h1>
          <ProductsView products={products} categories={categories} />
        </div>
      </div>
    );
  } catch (error) {
    console.error('Error fetching data:', error);
    return (
      <div className="flex flex-col items-center justify-center min-h-screen bg-gray-100 p-4">
        <div className="bg-white p-8 rounded-lg shadow-md w-full max-w-4xl">
          <h1 className="text-3xl font-bold mb-6 text-center text-red-500">
            Error
          </h1>
          <p className="text-center">Failed to load the category page.</p>
        </div>
      </div>
    );
  }
};

export default CategoryPage;
```



4. Search Bar

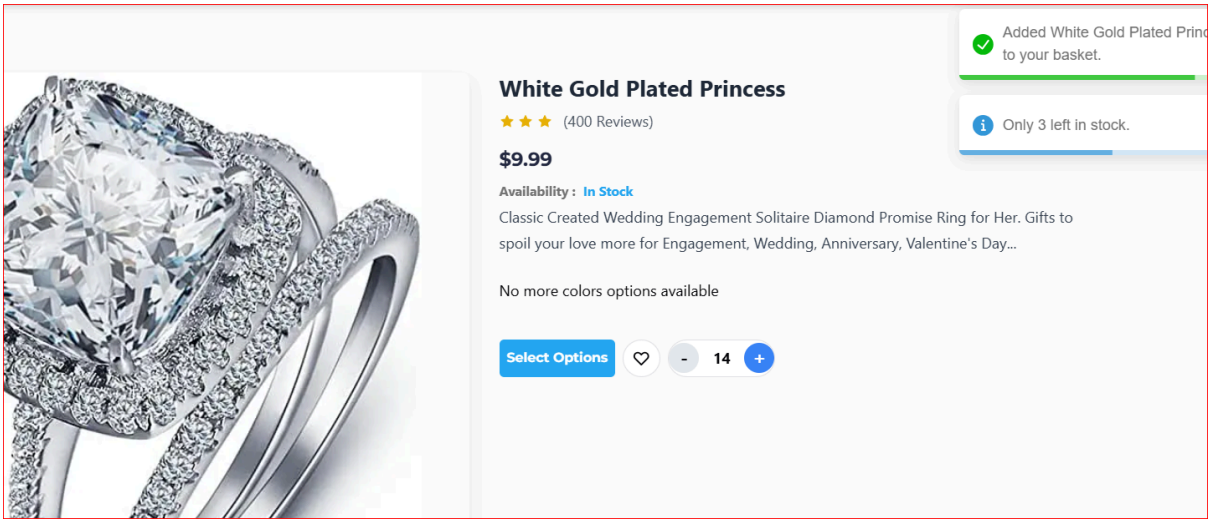
- Enables users to search products by **name or tags**.
- Implements **debouncing** for optimized performance.
- Uses **useState** and **useEffect** for search query handling.

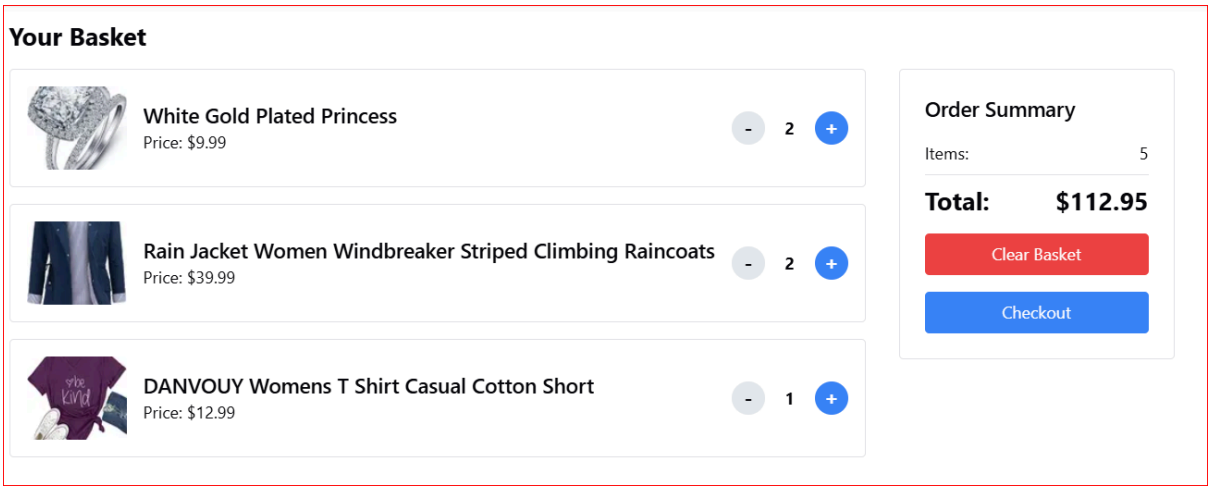


5. Cart Component

Users can add products to their cart directly from the product listing or detail page.

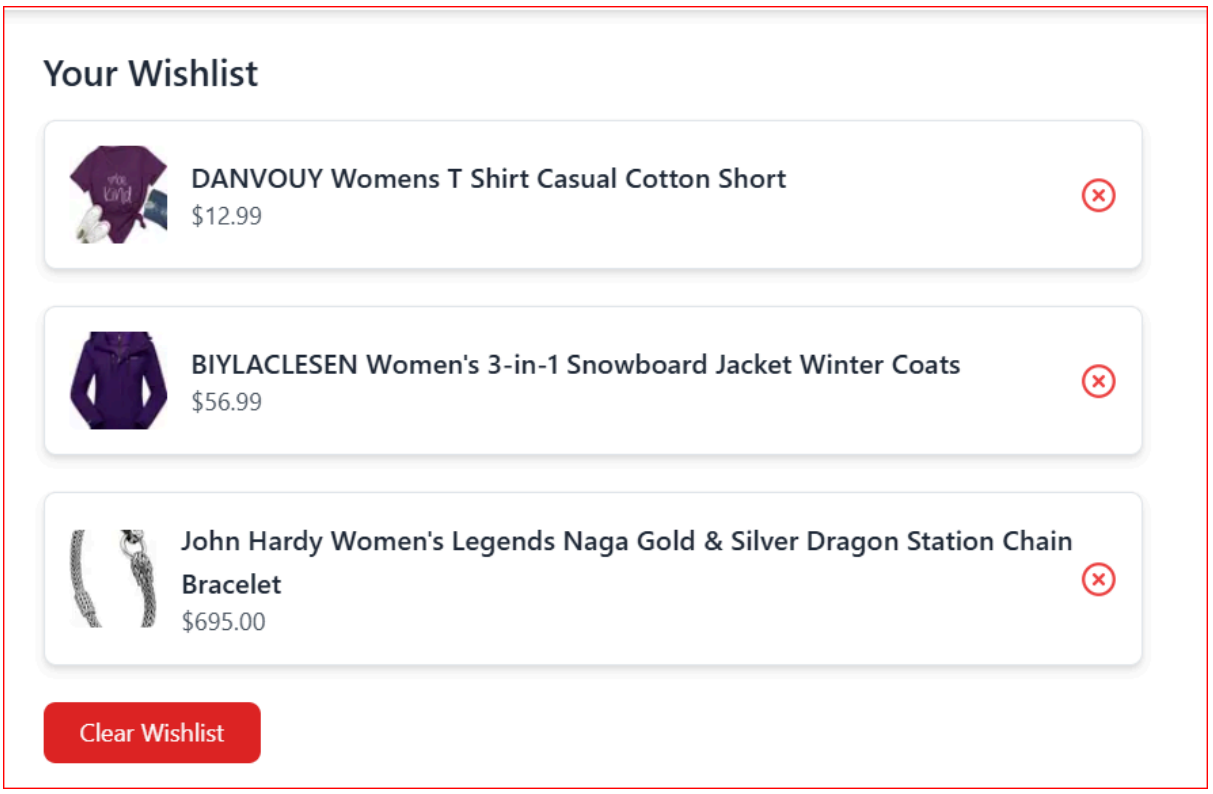
- Displays added items, their quantity, and total price.
- Uses **local state** for cart management.
- Provides functions to **increase/decrease quantity** and **remove items**.
- Persistent data using **local storage** for session retention.
- Alerts and indicators, such as "Only 3 left in stock," create a sense of urgency, encouraging purchases.





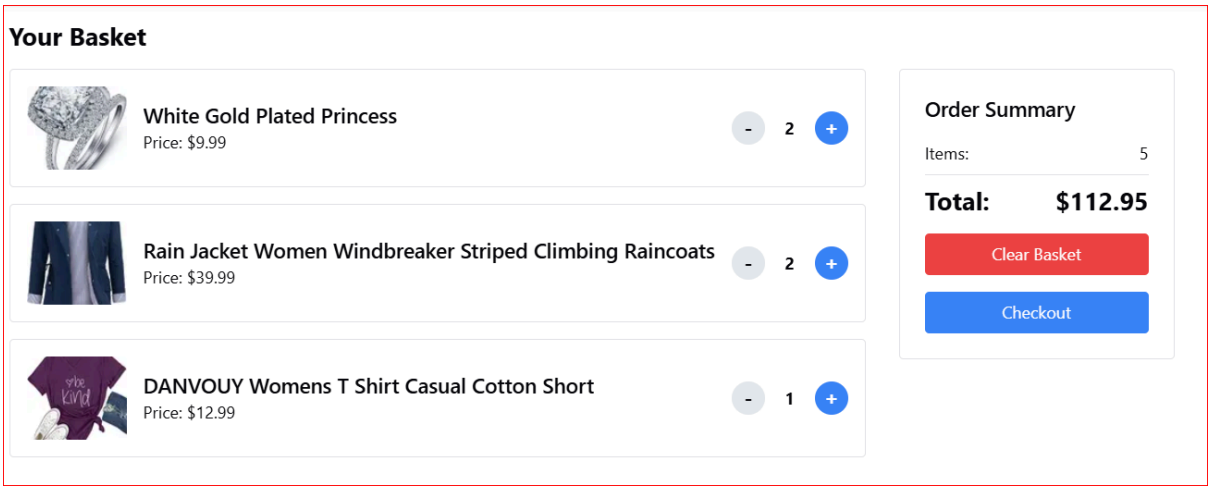
6. Wishlist Component

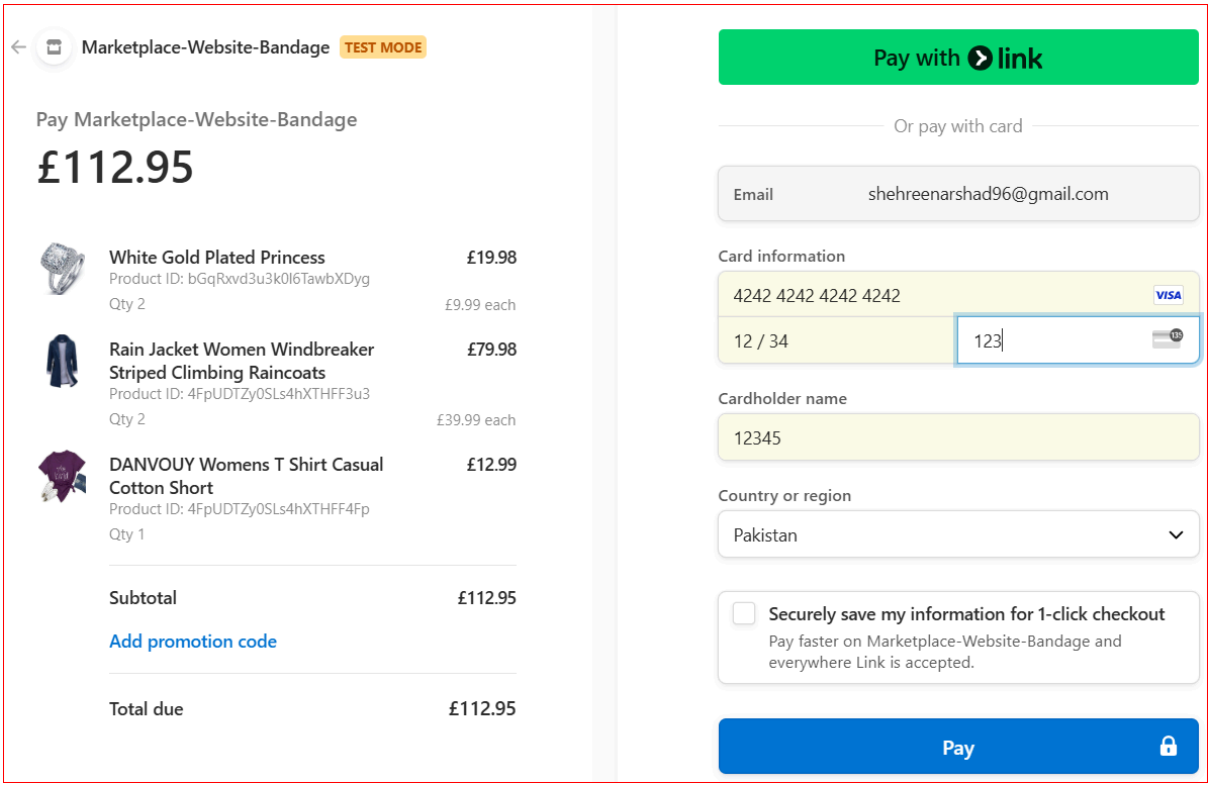
- Allows users to **save favorite products** for later.
- Implemented using **local state** and **local storage**.
- Provides an "Add to Wishlist" button in the **Product Detail Component**.
- Wishlist items persist across page reloads.
- Add **real-time notifications** (Toasts) for adding/removing cart and wishlist items.

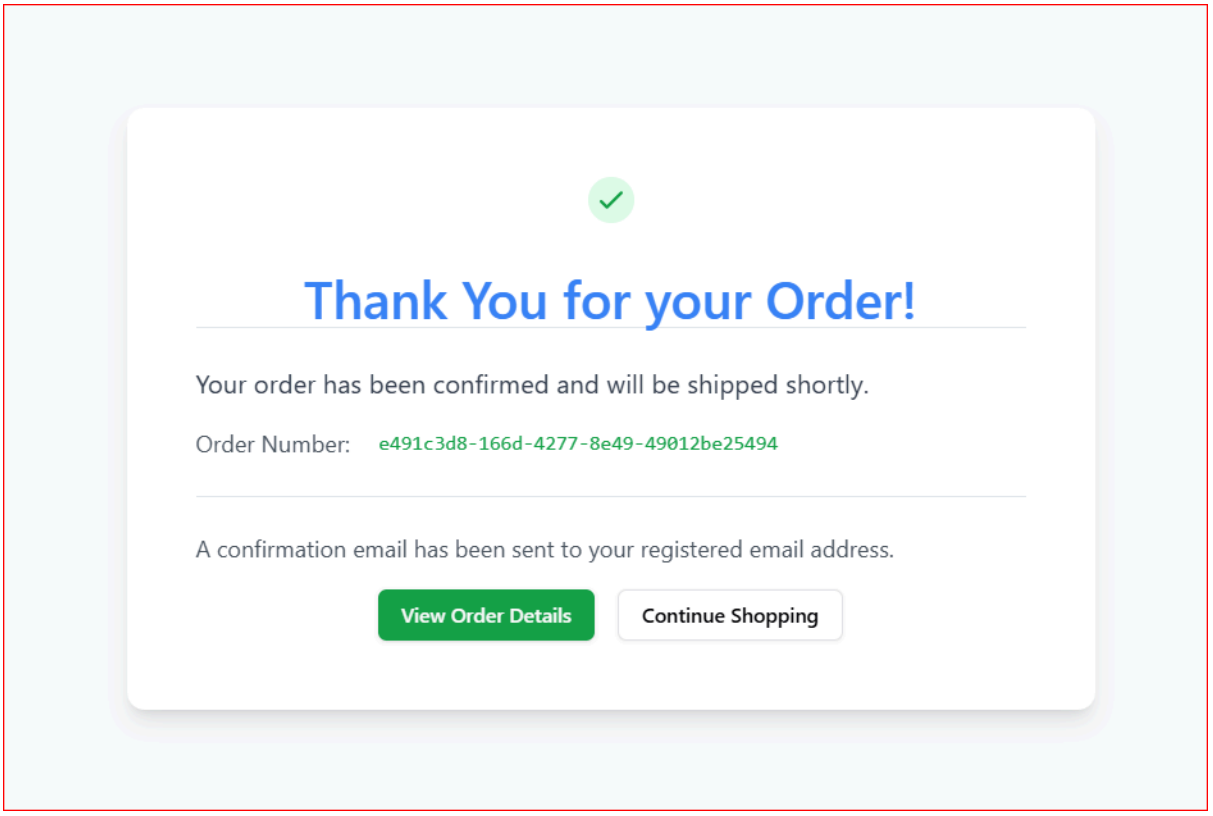


7. Checkout Flow Component

- Implements a **multi-step checkout process**:
  - **Payment Method (Stripe)**
  - **Order Confirmation**
- Ensures **form validation** for required fields.
- Uses **Next.js pages for step-by-step navigation**.







Technologies Used

- **Next.js** for dynamic routing and component-based architecture.
- **Tailwind CSS** for styling and responsive design.

- **Sanity** for storing all data.
- **React Hooks** (`useState`, `useEffect`) for state management.
- **Local Storage** for persisting cart and wishlist data.

Challenges & Solutions

Challenge	Solution
API data not loading immediately	Used `useEffect` to fetch data on component mount
Search bar lagging with fast input	Implemented **debouncing** to optimize performance
Wishlist not persisting after page reload	Stored wishlist data in **local storage**

Next Steps

- Implement **pagination** for the Product Listing page.
- Integrate **Sanity backend** for authentication-based wishlist persistence.

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!