

DAY 4 - BUILDING DYNAMIC FRONTEND COMPONENTS

Objectives for Building Dynamic Frontend Components:

- Enhanced Reusability:
- Improved Maintainability:
- Enhanced User Experience:
- Better Collaboration:
- Testing and Scalability:

Key Components:

- Search Bar
- Product listing Component
- Product detail component
- Cart Component
- Checkout Work Flow Component
- Product Comparison Component

Key Considerations:

- **Component-Based Architecture:** Adopt a component-based architecture that breaks down the UI into smaller, reusable units.
- **State Management:** Implement a state management solution (e.g., Redux, Zustand) to manage data flow and component interactions effectively.
- **Testing:** Write unit tests for individual components to ensure their correct functionality.
- **Accessibility:** Design and develop components with accessibility in mind to ensure they are usable by people with disabilities.
- **Performance Optimization:** Optimize components for performance by minimizing re-renders and efficiently handling data updates.

1. Search Bar Component:

The search bar uses dynamic routing in Next.js to fetch and display products based on user input. A GROQ query retrieves matching products from Sanity CMS, showing details like title and price. If no results are found, a fallback message appears.

Push query function:

```
const router = useRouter();
const searchQueryHandler = () => {
  router.push(`/search/${query}`);
}
```

[illegible]

Components > Header.tsx

[illegible]

Search > [query] > page.tsx

Output:

Users can search for products by entering a name or product tag in the search bar. The dynamic routing fetches and displays matching results, ensuring a seamless search experience.



2. Product Listing Component:

The Product Listing page uses GROQ queries to fetch products from Sanity CMS, displaying titles, prices, images, and descriptions for a seamless browsing experience with up-to-date data.

```

        context.Products.Where(p => p.ProductID == productId)
        .FirstOrDefault() == null
        ? context.Products.Create(Product) : context.Products.Update(Product);

        context.SaveChanges();

        return RedirectToAction("Index", "Products");
    }

    public ActionResult Details(int productId)
    {
        return View(context.Products.Where(p => p.ProductID == productId).
            FirstOrDefault());
    }
}

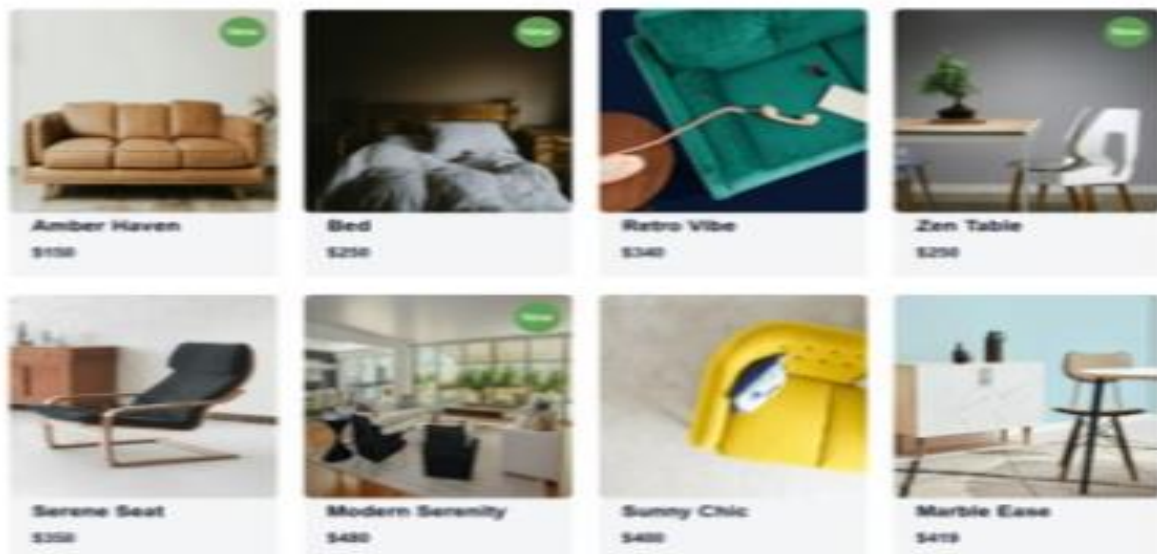
```

Fetching data from Sanity

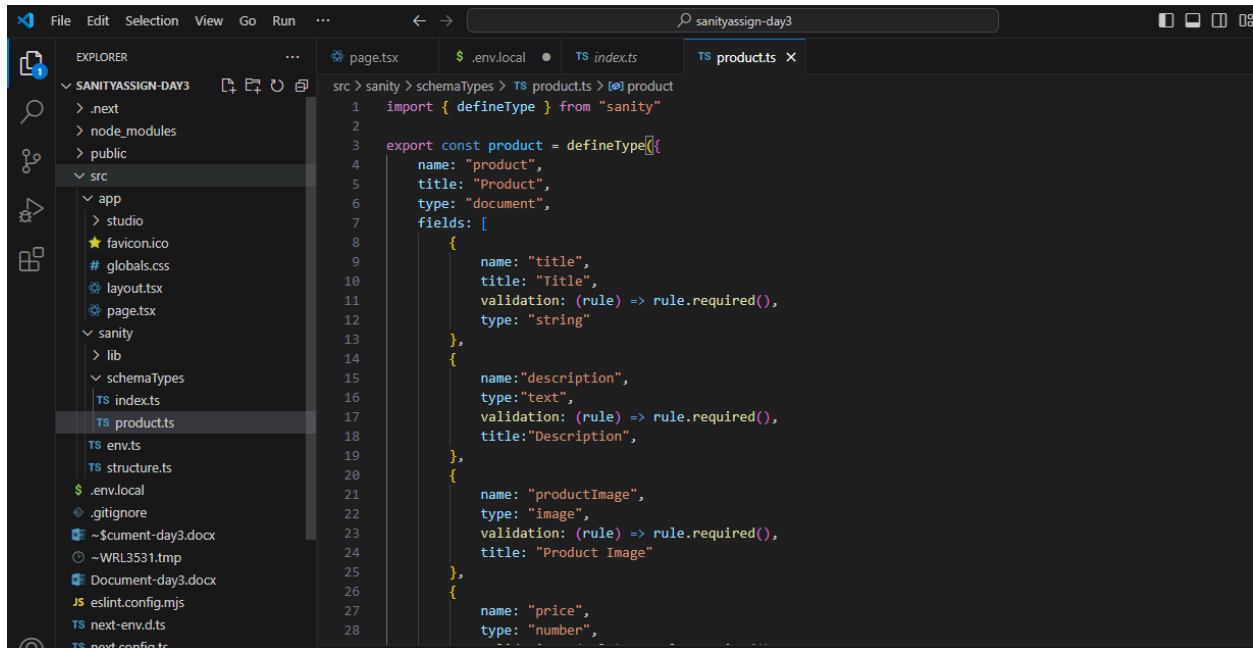
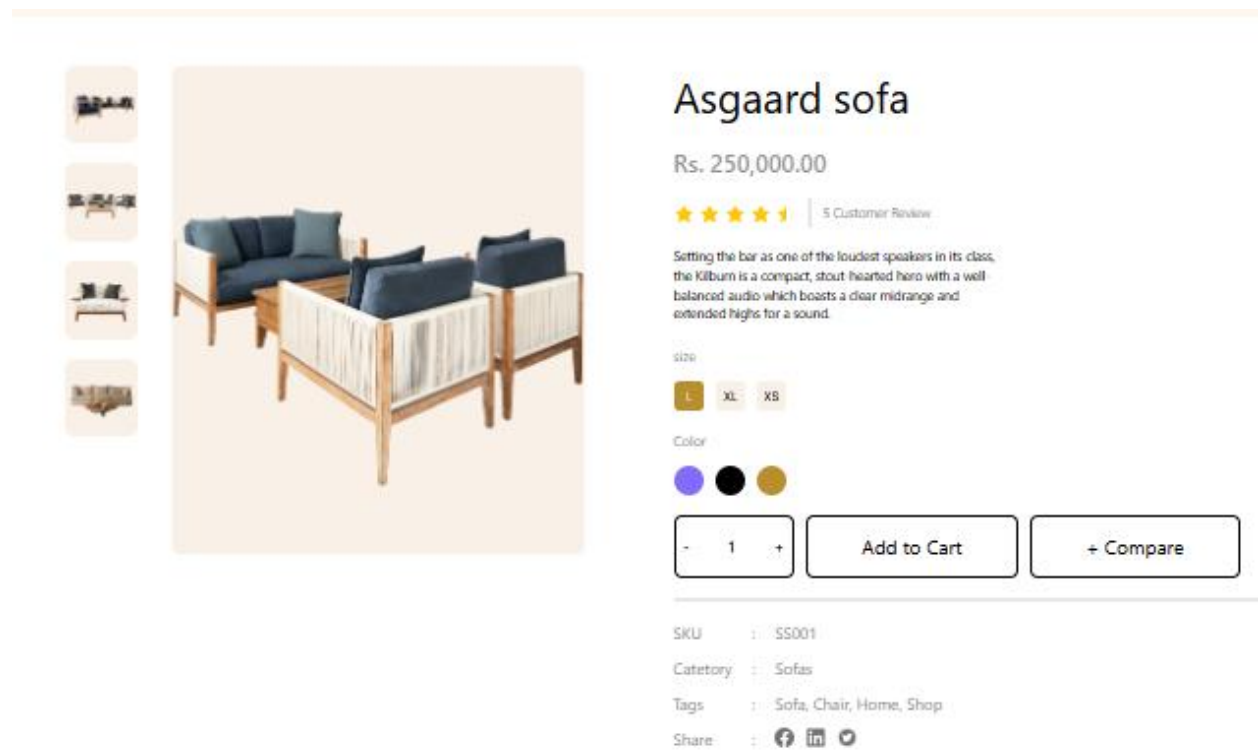
[illegible]

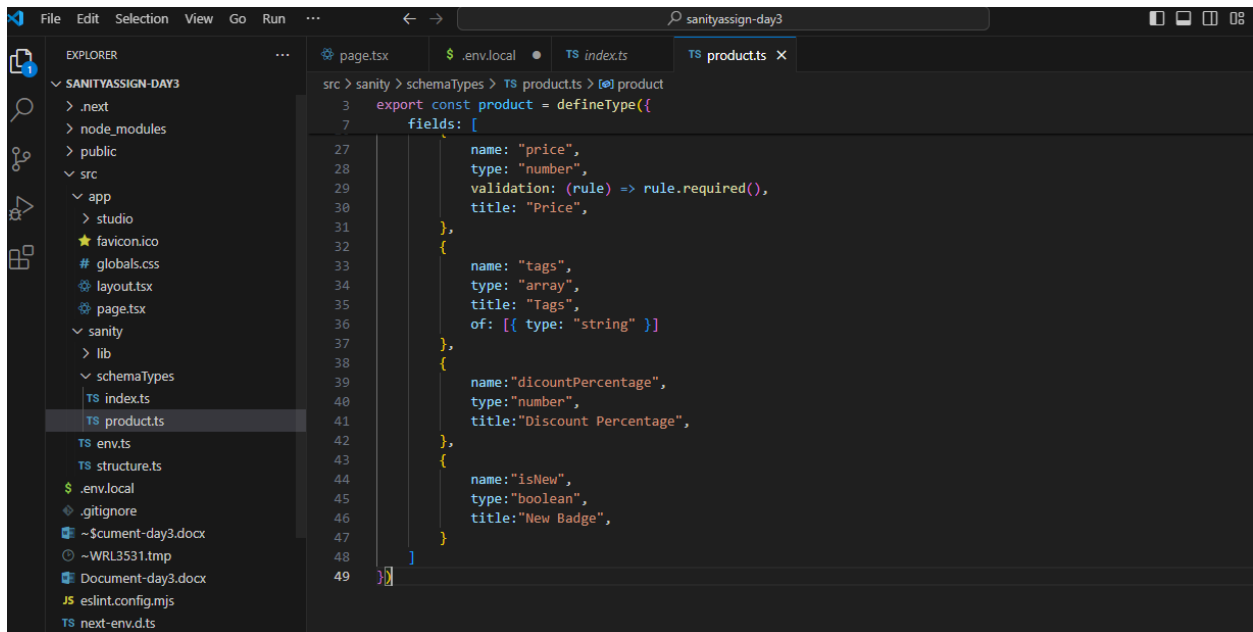
Display Data

Output:

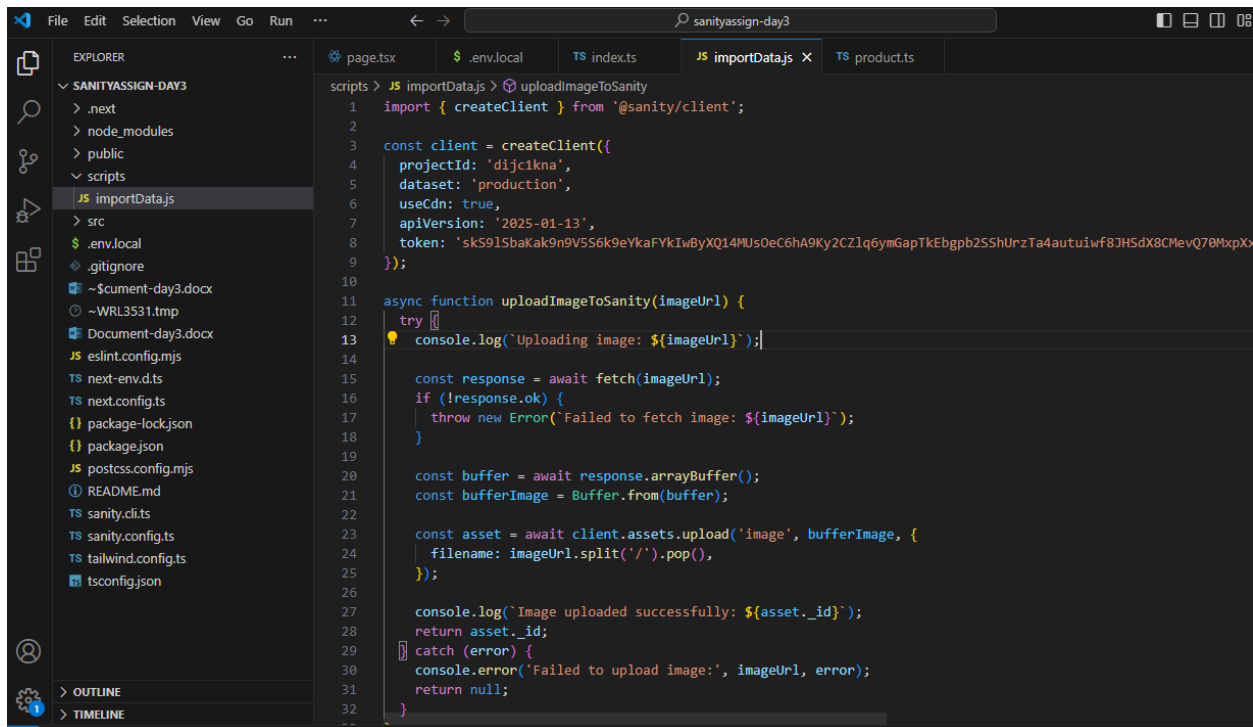


PRODUCT DETAIL COMPONENT





```
src > sanity > schemaTypes > TS products.ts > [0] product
3   export const product = defineType({
7     fields: [
27       {
28         name: "price",
29         type: "number",
30         validation: (rule) => rule.required(),
31         title: "Price",
32       },
33       {
34         name: "tags",
35         type: "array",
36         title: "Tags",
37         of: [{ type: "string" }]
38       },
39       {
40         name: "discountPercentage",
41         type: "number",
42         title: "Discount Percentage",
43       },
44       {
45         name: "isNew",
46         type: "boolean",
47         title: "New Badge",
48       }
49     ]
50   });
```



```
scripts > JS importData.js > uploadImageToSanity
1   import { createClient } from '@sanity/client';
2
3   const client = createClient({
4     projectId: 'dijc1kna',
5     dataset: 'production',
6     useCdn: true,
7     apiVersion: '2025-01-13',
8     token: 'skS91SbaKak9n9V5S6k9eYkaFYkIwByXQ14MUs0eC6hA9Ky2CZ1q6ymGapTkEbgpb2SShUrnzTa4autuifw8JHSdX8CMevQ70MxpXx
9   });
10
11  async function uploadImageToSanity(imageUrl) {
12    try {
13      console.log(`Uploading image: ${imageUrl}`);
14
15      const response = await fetch(imageUrl);
16      if (!response.ok) {
17        throw new Error(`Failed to fetch image: ${imageUrl}`);
18      }
19
20      const buffer = await response.arrayBuffer();
21      const bufferImage = Buffer.from(buffer);
22
23      const asset = await client.assets.upload('image', bufferImage, {
24        filename: imageUrl.split('/').pop(),
25      });
26
27      console.log(`Image uploaded successfully: ${asset._id}`);
28      return asset._id;
29    } catch (error) {
30      console.error('Failed to upload image:', imageUrl, error);
31      return null;
32    }
33  }
```

Fetching data

Our Product



Marble Ease
Rs.419

marble luxury furniture modern elegance
Discount : 0 %



Bed
Rs.250

bed furniture sleep cozy modern
Discount : 30 %



Wood Chair
Rs.100

wood chair furniture classic rustic
Discount : 10 %



Amber Haven
Rs.150

amber luxury cozy elegant furniture
Discount : 0 %



Cloud Haven Chair
Rs.230

cloud chair comfy home decor modern furniture
Discount : 20 %



Vase Set
Rs.150

vase decor interior design elegant home
Discount : 60 %



Syltherine
Rs.200

living fancy elegance design
Discount : 20 %



Retro Vibe
Rs.340

retro vintage furniture modern decor
Discount : 0 %

Show More

4.Cart Component:

The Cart component leverages Redux to manage the add-to-cart functionality efficiently. By wrapping the entire application with the `ReduxProvider`, the cart state becomes accessible throughout the app. This allows seamless addition, removal, and updating of cart items from any component. The Cart component dynamically fetches and displays selected products, showcasing details like title, price, and quantity, ensuring a consistent and responsive shopping experience for users.

Add to Cart Button

```
<button
  onClick={() => {
    dispatch(addToCart(element));
  }}
  type="button"
  className="text-[#D69100] w-[150px]
>
  Add to cart
</button>
```

Get Cart Data & Calculate Total Price

Product Quantity Control

```

[/* Quantity Controls */]
<div class={classNames} style={{transform: gap > 3 ? 'lg grow-1' : 'shrink-0'}}>
  <div class={classNames} onClick={() => {
    if (val.quantity > 1) {
      dispatch(incrementQuantity(val));
    }
  }}>
    <div class={classNames} text="lg font-bold">
      +
    </div>
    <div class={classNames} text="font-size-20">{val.quantity}</div>
    <div class={classNames} onClick={() => {
      dispatch(decrementQuantity(val));
    }}>
      <div class={classNames} text="lg font-bold">
        -
      </div>
    </div>
  </div>
</div>

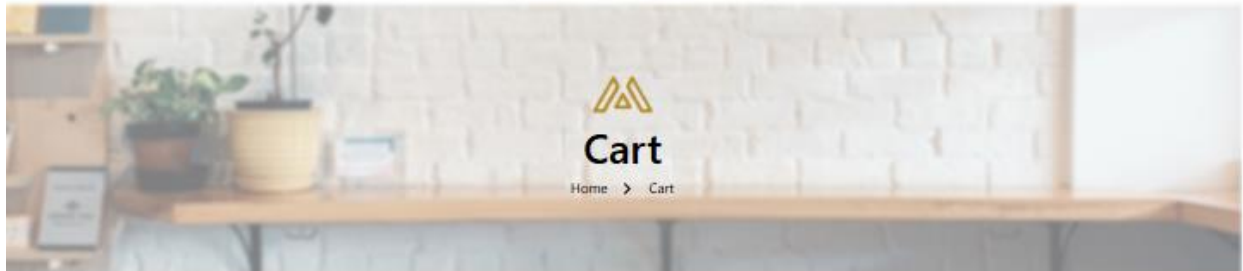
```

[Remove Product From Cart](#)

```

    /* Delete Button */
    <button
      onClick={() => {
        dispatch(removeFromTheCart(val._id));
      }}
    >

```

Product	Price	Quantity	Subtotal	
	Asgard sofa	Rs. 250,000.00	<input type="text" value="1"/>	Rs. 250,000.00 

Cart Totals

Subtotal Rs. 250,000.00

Total Rs. 250,000.00

[Check out](#)