HACKATHON Day 3 – Report **Presented by: Rukhsar Ashraf**

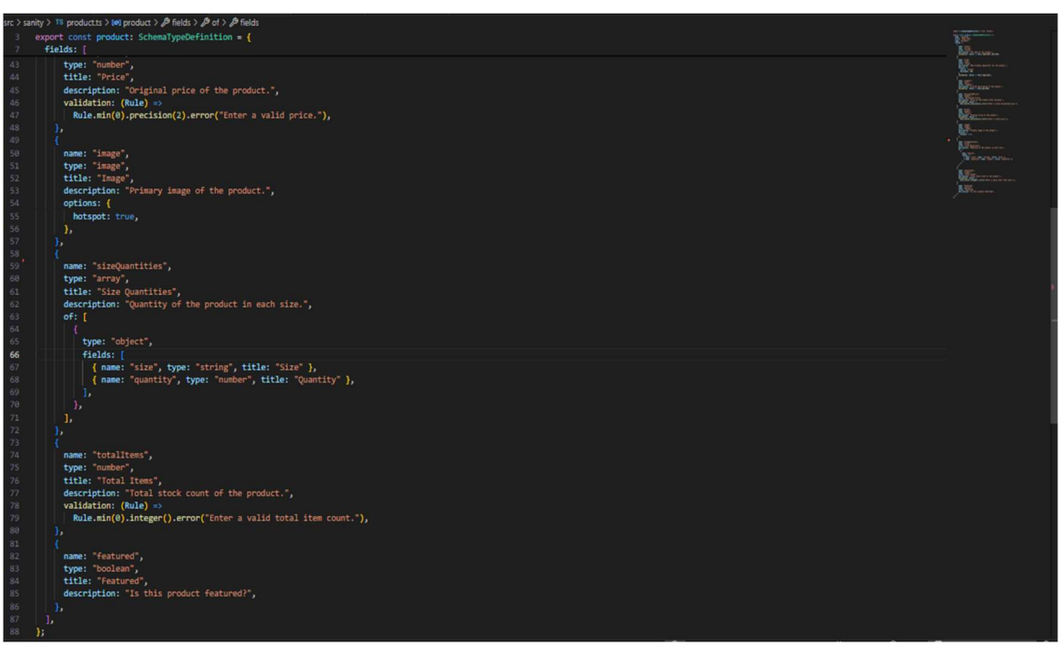
Day-3 API Integration And Data Migration

Objective:
This report provides a detailed overview of integrating Sanity CMS, fetching data, and displaying it on the frontend. It includes schema creation, data retrieval, and rendering it within the application.

1. **Sanity CMS Schema Design:**
   To efficiently manage product data, I created a schema named product in Sanity CMS. The schema includes the following fields:

   Main Fields:

   - id: A unique identifier for the product (string type).
   - name: The product's name (string type).
   - imagePath: The URL of the product image (URL type).
   - price: The price of the product (number type).
   - description: A detailed description of the product (text type).
   - discountPercentage: The discount percentage applied to the product (number type).
   - isFeaturedProduct: A boolean flag indicating if the product is featured (boolean type).
   - stockLevel: The quantity of the product available in stock (number type).



2: API Integration and Data Migration:

API Data Fetching:

I retrieved product data **internally within Sanity CMS**, which included key details such as images, titles, descriptions, prices, and sizes. This data was then mapped to the relevant fields in the **Sanity CMS schema**. The mapping process ensured that the data aligned perfectly with Sanity's internal structure, allowing for seamless integration without relying on external systems.

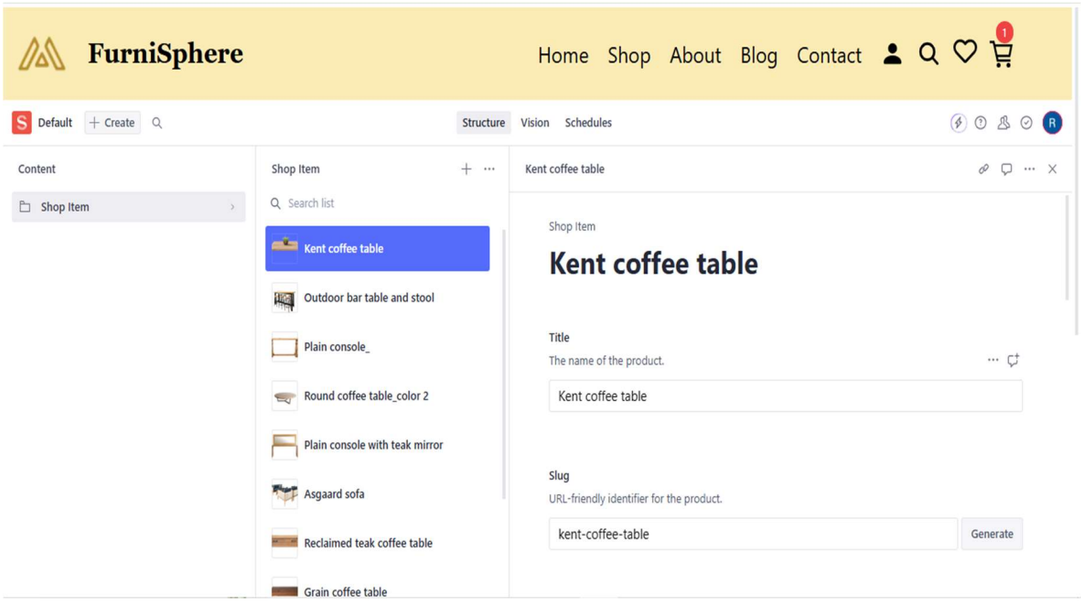
```
import { client } from "./client";

// Fetch Product Data
export const FetchProductsData = async() => {
  try {
    const query = `*[_type == "shopItem"] {
      title,
      "slug": slug.current,
      summary,
      discountedPrice,
      price,
      "image": image.asset->url,  // Ensure that we get the image URL
      colors,
      sizeQuantities,
      totalItems,
      featured,
      _id
    }`;

    const data = await client.fetch(query);
    return data;
  } catch (error) {
    console.error("Error fetching from Sanity:", error);
    throw error;
  }
};

export async function getData(slug: string) {
  try {
    const query = `*[_type == "shopItem" && slug.current == $slug][0] {
      title,
      "slug": slug.current,
      summary,
      discountedPrice,
      price,
      "image": image.asset->url,
      colors,
      sizeQuantities,
      totalItems,
      featured,
      _id
    }`;

    const product = await client.fetch(query, { slug });
    return product;
  } catch (error) {
    console.error("Error fetching product data:", error);
```

Data Population in Sanity CMS:

After fetching the data, I dynamically populated the product fields **directly within Sanity CMS**. This approach enabled the automated input of product information, ensuring consistency and accuracy across the platform while significantly reducing manual effort. All data was validated and formatted according to Sanity's internal requirements, ensuring smooth rendering and management.

Data Migration:

Utilizing the **Sanity CLI**, I exported the dataset **internally from Sanity CMS** for backup purposes and later re-imported it during testing. This process ensured that the data remained well-structured and displayed correctly throughout the migration process. The internal data structure within Sanity was preserved, and the re-import process confirmed that the data integrity was maintained, allowing for accurate representation on the frontend.



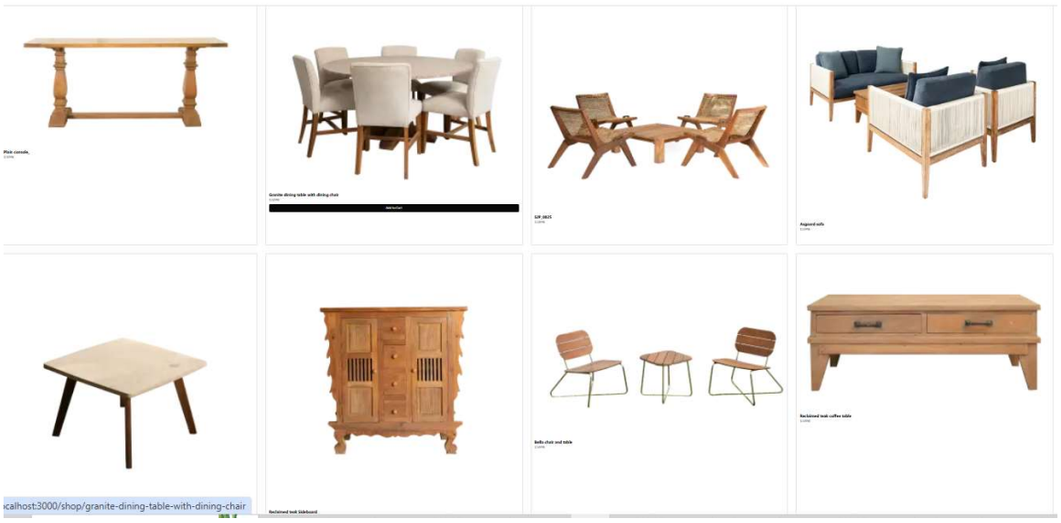
3: Frontend Integration:

After successfully handling the data internally within Sanity CMS, the next step was to ensure that the product data was dynamically rendered on the **frontend**. Here's how the process was implemented:

Fetching Data from Sanity CMS:

Using Sanity's API, I fetched the product data (including images, titles, descriptions, prices, and categories) from the CMS. This data was retrieved in a structured format, ensuring that all fields were correctly mapped and ready for display on the frontend.

```
export default async function Shop() {

            <section className="mx-auto mt-4 px-4 lg:px-8">
                <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-10">
                    {products.map((product:Product) => (
                        <div
                            key={product._id}
                            className="border p-4 group active:scale-95 transition-transform cursor-pointer rounded-md"
                        >
                            <Link href={`/shop/${product.slug}`}>
                            <Image
                                src={urlForImage(product.image).url()} // Sanity se aane wala image URL
                                alt={product.title}
                                width={300}
                                height={200}
                                className="w-full h-auto object-cover rounded-md"
                            />
                            <h3 className="mt-2 text-base sm:text-lg lg:text-xl font-bold">{product.title}</h3>
                            <p className="text-sm sm:text-base text-gray-700">${product.price}</p>
                            <button className="w-full mt-2 bg-black text-white py-2 opacity-0 group-hover:opacity-100 t
                                Add to Cart
                            </button></Link>
                        </div>
                    ))}
                </div>
            </section>
        </div>
    );
}
```



Dynamic Rendering on the Frontend:

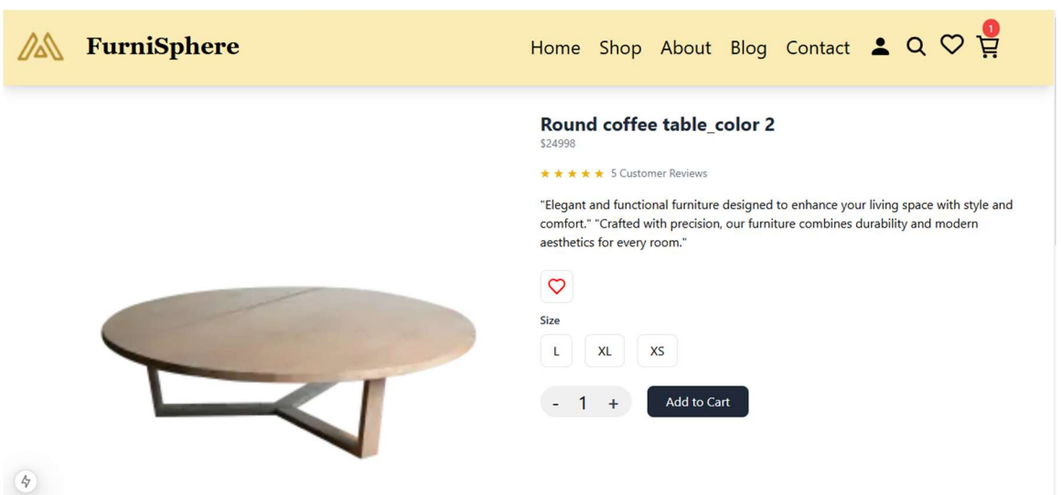
The fetched data was dynamically rendered on the marketplace frontend. This involved:

- Displaying product cards with images, titles, prices, and descriptions.
- Showing discount percentages and stock levels for each product.

Product Detail Page Integration:

For each product, a dedicated **product detail page** was created. This page displayed:

- High-resolution images of the product.
- Detailed descriptions and pricing information.
- Discounts and stock availability.
- Related products based on the category.



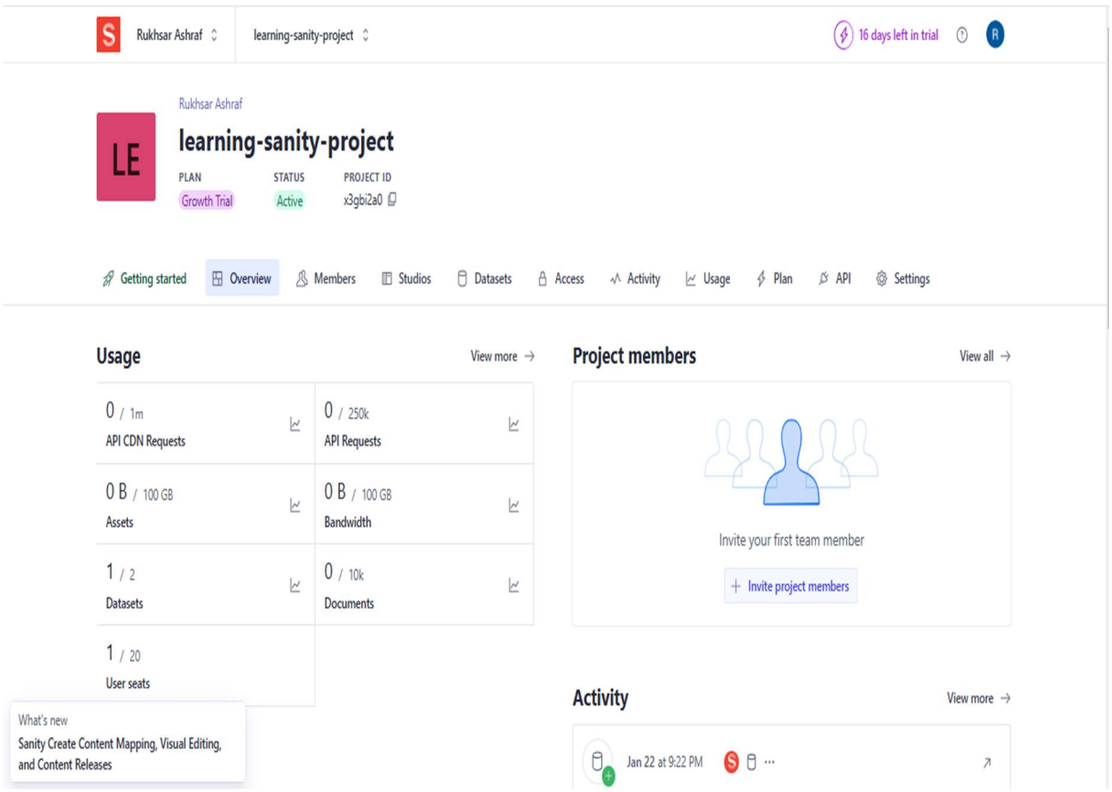
Testing and Validation:

The frontend integration was thoroughly tested to ensure that:

- All product data was displayed accurately.
- The user interface was intuitive and user-friendly.
- The performance was optimized for fast loading times.

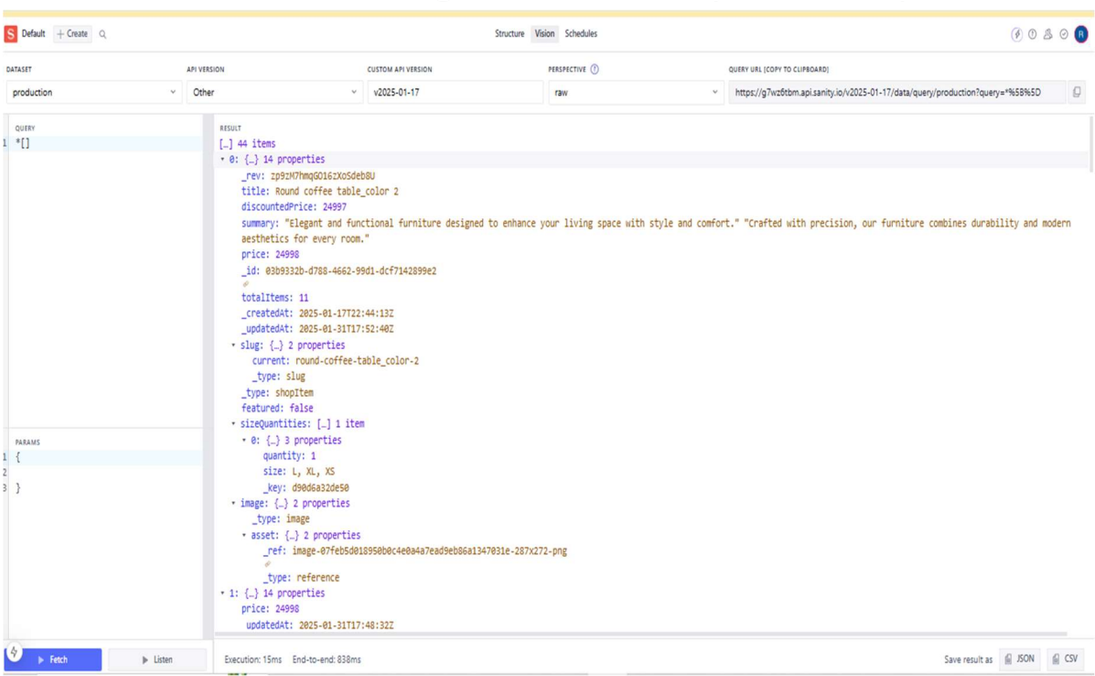
4:Tools Used:

- **Sanity Studio:** Used for schema creation, content management, and displaying product data.



5:Studio Vision:

- **Sanity CLI:** Utilized for exporting and importing the dataset, ensuring data consistency and backup.



Conclusion:

The internal processes within **Sanity CMS**, including data fetching, schema design, and data migration, were successfully implemented, ensuring a seamless and efficient workflow. By handling all data operations internally, the system eliminated the need for external dependencies, improving consistency and accuracy across the platform.

The combination of internal Sanity CMS processes and frontend integration has laid a strong foundation for future scalability and growth.

DAY-3, Presented by: Rukhsar Ashraf