Market Blace Builder Hackathon 2025

Day 3 Report

API Integration & Data Migration

Project Name: "Avion(FurnitureShop): A General-Ecommerce Marketplace"

Date: 18/01/2025

1. Objective

The goal of Day 3 was to integrate the Sanity CMS backend with the Next.js frontend, enabling dynamic data fetching for furniture products. This included setting up schemas, configuring the client, and displaying fetched data on the website.

2. Tasks Completed

- ***Sanity CMS Setup:***

1. Created schemas for **Products** and **Categories**.
2. Configured the Sanity client to connect the Next.js application with the Sanity backend.

- ***Performed Migration:***

- Ensured the schemas were correctly migrated to the Sanity backend.

- ***Data Fetching:***

- Fetched product data dynamically from the Sanity backend.
- Ensured data was displayed correctly on the product listing page.

- ***Dynamic Routing:***

- Implemented dynamic routes for individual **Product Detail Pages**.

- ***Error Handling:***

- Added basic error handling to display fallback messages when no data is available.

3. Schemas

a) Product Schema

The schema for furniture products included fields such as name, category, price, tags, image, description, and dimensions.

- **Code Snippet**:

```
import { defineType, defineField } from "sanity"

export const product = defineType({
    name: "product",
    title: "Product",
    type: "document",
    fields: [
        defineField({
            name:"category",
            title:"Category",
            type:"reference",
            to:[{
                type:"category"
            }]
        }
        ),
        defineField({
            name: "name",
            title: "Title",
            validation: (rule) => rule.required(),
            type: "string"
        }),
        defineField({
            name: "slug",
            title: "Slug",
            validation: (rule) => rule.required(),
            type: "slug"
        }),
        defineField({
            name: "image",
            type: "image",
```

```
            validation: (rule) => rule.required(),
            title: "Product Image"
        }),
        defineField({
            name: "price",
            type: "number",
            validation: (rule) => rule.required(),
            title: "Price",
        }),
        defineField({
            name: "quantity",
            title: "Quantity",
            type: "number",
            validation: (rule) => rule.min(0),
          }),
        defineField({
            name: "tags",
            type: "array",
            title: "Tags",
            of:[{
                type: "string"
            }]
        }),
        defineField({
            name: 'description',
            title: 'Description',
            type: 'text',
            description: 'Detailed description of the product',
          }),
          defineField({
            name: 'features',
            title: 'Features',
            type: 'array',
            of: [{ type: 'string' }],
            description: 'List of key features of the product',
          }),
          defineField({
            name: 'dimensions',
            title: 'Dimensions',
            type: 'object',
            fields: [
              { name: 'height', title: 'Height', type: 'string' },
              { name: 'width', title: 'Width', type: 'string' },
              { name: 'depth', title: 'Depth', type: 'string' },
            ],
            description: 'Dimensions of the product',
          }),
    ]
})
```

b) Category Schema

The schema for categories included fields such as name and slug.

- **Code Snippet**:

```
import { defineType,defineField } from "sanity";

export const Category = defineType({
    name: "category",
    title: "Category",
    type: "document",
    fields:[
        defineField({
            name: "name",
            title: "Name",
            type: "string",
            validation: (rule) => rule.required(),
        }),
        defineField({
            name: "slug",
            title: "Slug",
            type: "slug",
            validation: (rule) => rule.required(),
            options: {
                source: "name",
            }
        })
    ]
})
```

4. Sanity Client Configuration

Configured the client in `client.ts` to enable communication between Sanity and Next.js.

- **Code Snippet**:

```
import { createClient } from 'next-sanity'
import imageUrlBuilder from '@sanity/image-url';
import { apiVersion, dataset, projectId } from '../env'

export const client = createClient({
  projectId,
  dataset,
  apiVersion,
  useCdn: true, // Set to false if statically generating pages, using ISR or tag-based revalidation
})

const builder = imageUrlBuilder(client);

export function urlFor(source: any) {
  return builder.image(source).url();
}

export { imageUrlBuilder };
```

5. Migration

Migration ensured that the defined schemas in the codebase were applied to the Sanity backend, enabling proper data structure and integrity.

- **Code Snippet**:

```
import axios from 'axios';
import { client } from './sanityClient.js';
import slugify from 'slugify';

async function uploadImageToSanity(imageUrl: string): Promise<string|null> {

  try {
    // Fetch the image from the URL and convert it to a buffer
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer',timeout: 10000 });
    const buffer = Buffer.from(response.data);

    // Upload the image to Sanity
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(), // Extract the filename from URL
    });

    // Debugging: Log the asset returned by Sanity
    console.log('Image uploaded successfully:', asset);

    return asset._id; // Return the uploaded image asset reference ID
  } catch (error) {
    console.error('❌ Failed to upload image:', imageUrl, error);
    return null
    //throw error;
  }
}

interface Category {
  _id?:string,
  _type?:string,
  name:string,
  slug:string
}

async function createCategory(category:Category,counter:number) {

  try {
    const categoryExist = await client.fetch(`*[_type=="category" &&
slug==$slug][0]`,{slug:category.slug})
    if(categoryExist)
    {
      return categoryExist._id
    }
    const catObj = {
      _type:"category",
      _id:category.slug+"-"+counter,
      name:category.name,
      slug:category.slug
```

```
    }
    const response = await client.createOrReplace(catObj)


    // Debugging: Log the asset returned by Sanity
    console.log('Category created successfully', response);

    return response._id; // Return the uploaded image asset reference ID
  } catch (error) {
    console.error('❌ Failed to category:', category.name, error);
    //throw error;
  }
}


async function importData() {
    try {
      // Fetch data from external API
      const response = await axios.get('https://hackathon-apis.vercel.app/api/products');
      const products = response.data;
      //console.log(products)
      let counter=1;
      // Iterate over the products
      for (const product of products) {
        let imageRef = null;
        let catRef=null;

        // Upload image and get asset reference if it exists
        if (product.image) {
          //imageRef = await uploadImageToSanity(product.imageUrl);
          imageRef = await uploadImageToSanity(product.image);
        }

        if(product.category.name){
          catRef = await createCategory(product.category,counter)
        }

        const sanityProduct = {
          _id: `product-${counter}`, // Prefix the ID to ensure validity
          _type: 'product',
          name: product.name,
          slug: {
            _type: 'slug',
            current: slugify(product.name || 'default-product', {
              lower: true, // Ensure the slug is lowercase
              strict: true, // Remove special characters
            }),
          },
          price: product.price,
          category:{
            _type: 'reference',
            _ref:catRef?catRef:undefined
          },
          tags: product.tags?product.tags:[],
          quantity:50,
          image: imageRef ? {
            _type: 'image',
```

```
            asset: {
              _type: 'reference',
              _ref: imageRef, // Set the correct asset reference ID
            },
          } : undefined,
          description: product.description?product.description: "A timeless design, with premium materials
features as one of our most popular and iconic pieces. The dandy chair is perfect for any stylish living
space with beech legs and lambskin leather upholstery.",
          features: product.features?product.features: [
            "Premium material",
            "Handmade upholstery",
            "Quality timeless classic",
          ],
          dimensions: product.dimensions?product.dimensions : {
            _type: 'dimensions', // Custom object type for dimensions
            height: "110cm",
            width: "75cm",
            depth: "50cm",
          }
        };
        counter++
        // Log the product before attempting to upload it to Sanity
        console.log('Uploading product:', sanityProduct);

        // Import data into Sanity
        await client.createOrReplace(sanityProduct);
        console.log(`☑ Imported product: ${sanityProduct.name}`);
      }

      console.log('☑ Data import completed!');
    } catch (error) {
      console.error('❌ Error importing data:', error);
    }
}

importData();
```

6. Pages Created

a) Products Page

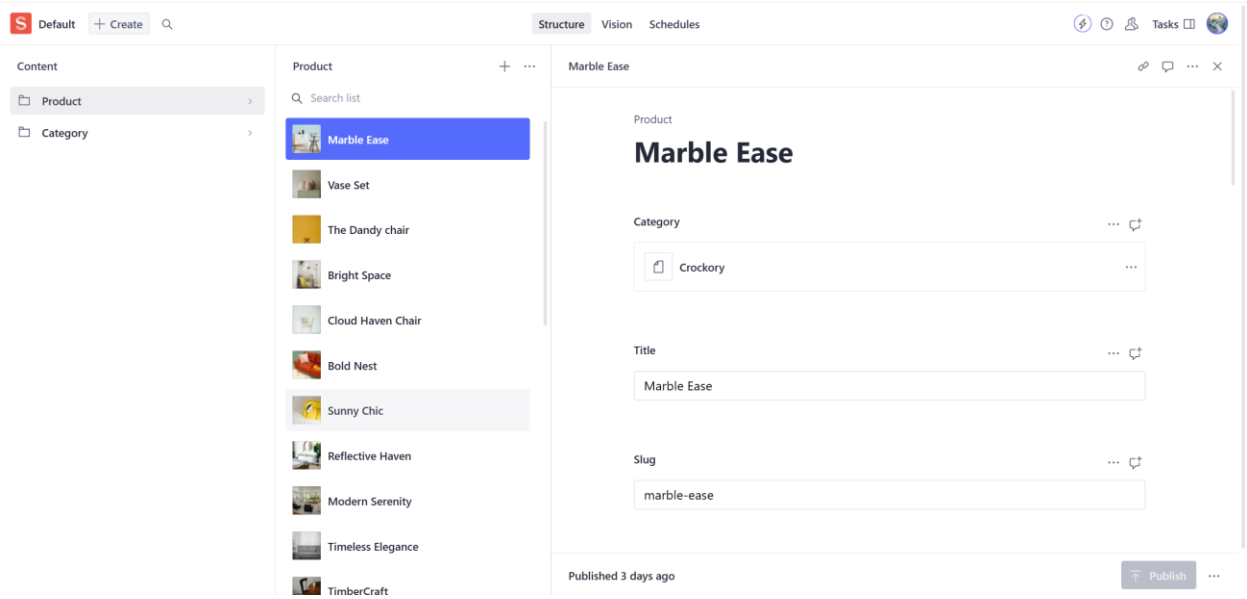
- Displays a list of furniture products fetched from Sanity.
- Implements dynamic routing to individual Product Detail Pages.

b) Product Detail Page

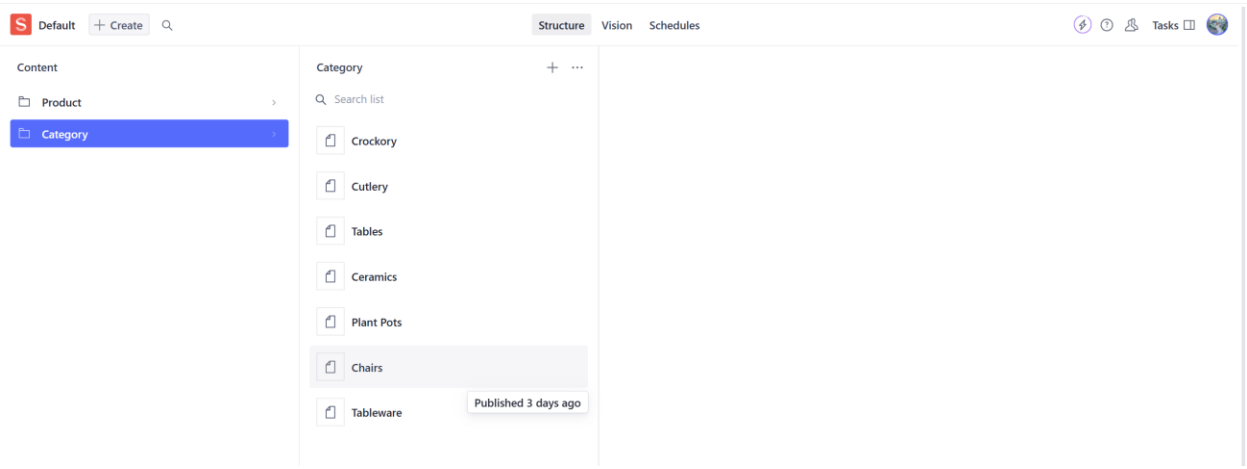
- Displays detailed information for a selected product, including name, price, description, image, and dimensions.

7. Screenshots of Functionality and Results

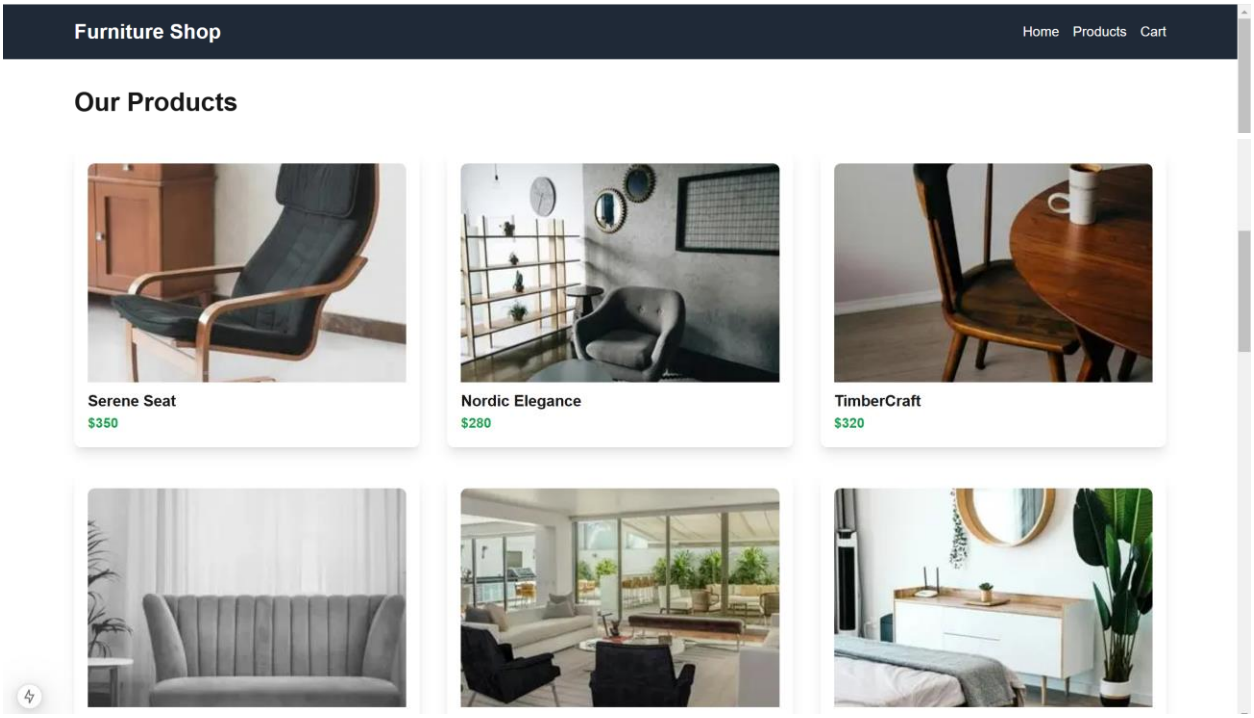
1. **Successfully created ‘product’ schema and Migrate ‘product’ data into Sanity CMS**



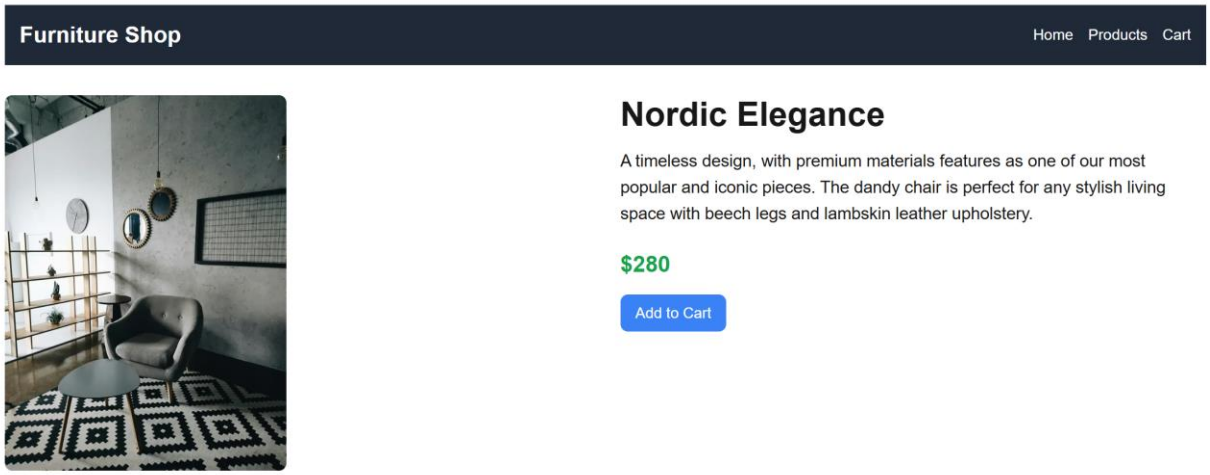
2. **Successfully created ‘category’ schema and Migrate ‘category’ data into Sanity CMS**



3. **Products Page**: Showing all products in a grid format.



4. **Product Detail Page**: Displaying detailed information for a selected product.



8. Challenges Faced

Data Fetching:

- Ensured correct queries were used to fetch product data dynamically from Sanity.

Dynamic Routing:

- Implemented `/products/[slug]` for detailed product pages.

Error Handling:

- Handled empty or missing product data gracefully by displaying fallback UI.

9. Conclusion

The integration of Sanity CMS was successful, with dynamic data fetching for products fully implemented. Dynamic routing and error handling were also achieved, laying the foundation for further enhancements.

Prepared by: Kinza M. Ayub