Products and Category Schema Report

1. Introduction

This report outlines the details of the products and category schemas fetched from an external API and subsequently posted to Sanity CMS. The report covers the schema structure, API data mapping, and implementation steps.

2. API Data Fetching

2.1 API Overview

- **API Endpoint:** https://hackathon-apis.vercel.app/api/products
- **Purpose:** Fetch product and category data.

Response Example


```
[
  {
    "name": "The Poplar suede sofa",
    "description": "A timeless design, with premium materials features as one of our most popular and iconic pieces. The dandy
    "image": "https://cdn.sanity.io/images/ri847jqu/production/9b6a4fc8c65bbb4e5793fb0e1116b510d73dc9e8-630x375.png",
    "_id": "65453ffd-e476-4b6b-a388-7e3de1bb632a",
    "features": [
        "Premium material",
        "Handmade upholster",
        "Quality timeless classic"
    ],
    "dimensions": {
        "width": "110cm",
        "height": "110cm",
        "depth": "50cm"
    },
    "category": {
        "name": "Tableware",
        "slug": "tableware"
    },
    "price": 980,
    "tags": [
        "popular products"
    ]
  }
]
```

]3. Sanity CMS Schemas

3.1 Product Schema

This Schema represents the products in Sanity CMS

Schema 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",
        }),
```

3.2 Category Schema

This schema represents the categories of products in the Sanity CMS.

Schema 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. Data Mapping and Posting

4.1 Data Mapping

The data fetched from the API was mapped to the corresponding fields in the Sanity schemas:

Product Data Mapping:

API Field	Sanity Field	Description
`id`	`_id`	Unique identifier for the product
`name`	`name`	Product name
`price`	`price`	Product price
`images`	`images`	Array of product images
`slug`	`slug`	URL-friendly identifier
`category`	`category`	Reference to the category
`Feature`	`Features`	Describe features of the product
`Tags`	`tags`	Additional for filtering products

Category Data Mapping:

API Field	Sanity Field	Description
`id`	`_id`	Unique identifier for the category
`name`	`name`	Category name
`description`	`description`	Category description
`image`	`image`	image

4.2 Posting Data to Sanity

The following steps were taken to post the fetched data to Sanity:

1. **Fetching Data:** Data was fetched from the API using an asynchronous function.
2. **Sanity Store creation:** A store was created and its “id”, “dataset”, and “token” was passed in the sanity-migration repo provided in the day_3 hackathon document.
3. **Connecting with NextJs:** Sanity store was connected with our NextJs app.

5. Conclusion

The product and category data fetched from the API have been successfully mapped and posted to Sanity CMS. This ensures a dynamic and scalable data management system for the aesthetic products marketplace.
