API Integration Report: Layers

Contents

Overview

On Day 3, the focus was on integrating APIs into the **Layers** project and populating Sanity CMS with data sourced from a local API. This report documents the API integration process, schema adjustments, migration steps, and the tools used. Code snippets are included to provide a comprehensive understanding of the implementation.

API Integration Process

1. ### Data Source:
   Data was fetched from the local API endpoint: http://localhost:3000/api/products.
   The product data included fields such
   as name, description, images, price, discountPercent, category, subcategory, sizes, colors , reviews, and slug.

2. ### Integration Steps:

   - Schema Design: Created a custom schema for products in Sanity CMS to align with the structure of the imported data.
   - Import Script: Developed a migration script to fetch product data from the API, process it, and upload it to Sanity CMS.
   - Image Handling: Enhanced the script to fetch images as an array and upload them to Sanity CMS with unique references.
   - Data Validation: Incorporated validation rules for fields like slug, price, and reviews to ensure data consistency.
   - API Endpoints: Created endpoints to retrieve data from Sanity CMS for use in the frontend.

Schema Adjustments

The schema was customized to accommodate data fields such as tags, colors, and images. Key validation rules and slug uniqueness checks were implemented.

Schema Source:

```
export const product = defineType({
  name: "product",
  type: "document",
  title: "Product",
  fields: [
    defineField({
      name: "name",
      type: "string",
      title: "Product Name",
      validation: (Rule) => Rule.required(),
    }),
    // ... (other fields as provided)
  ],
});
```

Migration Steps

1. ### Tools Used:
   - **Sanity Client**: For uploading data to Sanity CMS.
   - **Axios**: For API calls to fetch product data.
   - **UUID**: For generating unique keys for images.
   - **.env:** For managing environment variables.
2. ### Migration Script:
   The script automated the process of fetching data, processing it, and importing it into Sanity CMS while handling images as arrays.

Script Source:

```
import { createClient } from "@sanity/client";
import axios from "axios";
import { v4 as uuidv4 } from "uuid";

const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: "production",
  token: process.env.SANITY_API_TOKEN,
  useCdn: false,
});

async function importData() {
  const response = await axios.get("http://localhost:3000/api/products");
  const products = response.data;

  for (const product of products) {
    const imageRefs = await Promise.all(
      product.imageUrl.map(async (url) => {
        const response = await axios.get(url, { responseType: "arraybuffer" });
        const asset = await client.assets.upload("image", Buffer.from(response.data), {
          filename: url.split("/").pop(),
        });
        return { asset: { _ref: asset._id }, _key: uuidv4() };
      })
    );
    const sanityProduct = {
      _type: "product",
      name: product.name,
      price: product.price,
      images: imageRefs,
      // Additional fields
    };

    await client.create(sanityProduct);
  }
}

importData();
```

API Calls

1. ### Fetching All Products:

   The following query fetches all products from Sanity CMS:

Query Source:

```
const query = `
  *[_type == "product"] | order(createdAt desc){
    name,
    description,
    "images": images[].asset->url,
    price,
    discountPercent,
    subcategory,
    stock,
    sizes,
    colors,
    "slug": slug.current,
    reviews[],
    "discountedPrice": price - (price * discountPercent / 100),
    _id,
  }`;
```

Next Steps

The next steps for the **Layers** project include:

- **Improving the current implementation** by optimizing queries and enhancing the schema.
- **Implementing advanced features** like search, filtering, and sorting.
- **Scaling the API** to handle more data and users.