Day 3 - API Integration and Data Migration Documentation

Objective

To integrate APIs into a Next.js project and migrate data into Sanity CMS, creating a functional marketplace backend. This involves understanding APIs, adjusting schemas, migrating data, and implementing error handling to replicate real-world practices.

Key Learning Outcomes

1. Understand API integration techniques in a Next.js project.
2. Learn data migration methods to import API data into Sanity CMS.
3. Gain practical experience in handling schemas and ensuring compatibility with data sources.
4. Develop robust error-handling strategies.

STEPS FOR API INTEGRATION AND DATA MIGRATION

API Integration

Environmental Setup for Sanity CMS Integration

1. Install Necessary Dependencies

Install the required packages using npm:

```
npm install @sanity/client axios dotenv
```
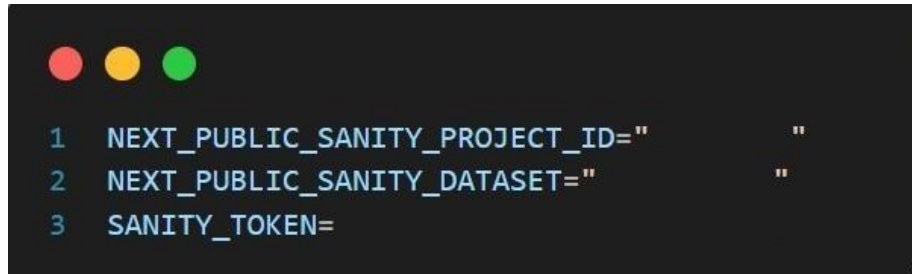
These packages are essential for:

- Communicating with the Sanity CMS (`@sanity/client`)
- Fetching data from APIs (`axios` or fetch methods)
- Managing environment variables (`dotenv`)

---

2. Configure Environment Variables

To securely store your Sanity project credentials, configure the environment variables in a `.env` file. Add the following entries:

```
SANITY_PROJECT_ID=<your_project_id>
SANITY_DATASET=<your_dataset_name>
SANITY_API_TOKEN=<your_api_token>
```

Steps:

```
1   NEXT_PUBLIC_SANITY_PROJECT_ID="          "
2   NEXT_PUBLIC_SANITY_DATASET="          "
3   SANITY_TOKEN=
```
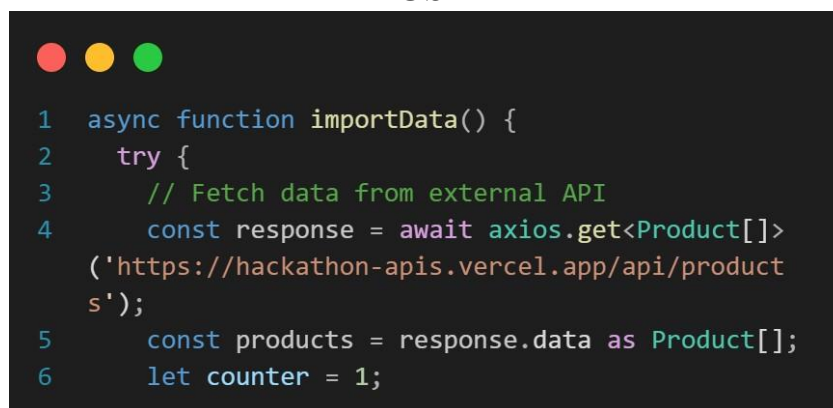
1. Replace `<your_project_id>`, `<your_dataset_name>`, and `<your_api_token>` with the appropriate values from your Sanity project.
2. Ensure the `.env` file is added to `.gitignore` to prevent exposing credentials in version control.

---

3. Data Fetching

Use `axios` or the native `fetch` method to retrieve data from an API endpoint. Example:

AXIOS

```
1   async function importData() {
2     try {
3       // Fetch data from external API
4       const response = await axios.get<Product[]>
    ('https://hackathon-apis.vercel.app/api/product
    s');
5       const products = response.data as Product[];
6       let counter = 1;
```

```typescript
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);
  }
}
```

4. Data Processing

Map the fetched data to match the desired schema structure for Sanity CMS.

```jsx
<div className="flex flex-col md:flex-row gap-8 mt-8">
        {products.map((product) => (
          <div
            key={product._id}
            className="w-full md:w-[350px] lg:w-[700px] h-auto group"
          >
            <Image
              src={product.image}
              alt={product.name}
              width={800}
              height={800}
              priority=
              {product._id === products[0]?._id} // Set priority for the first image
              className="w-full h-[80%] object-cover transition-transform duration-300 ease-in-out group-hover:scale-105"
            />
            <div className="mt-4 text-[#2A254B]">
              <p className="py-2 font-medium">{product.name}</p>
              <p className="font-semibold">${product.price}</p>
            </div>
          </div>
        ))}
      </div>
```

5. Sanity Document Creation

Use Sanity's `createOrReplace` method to populate the CMS with the processed data.
Example:

```javascript
import { createClient } from '@sanity/client';
import dotenv from "dotenv"

dotenv.config()
export const client = createClient({
  projectId: process.env.projectId, // Replace with your project ID
  dataset: 'production',           // Or your dataset name
  apiVersion: '2024-01-04',        // Today's date or latest API version
  useCdn: false,                   // Disable CDN for real-time updates
  token: process.env.token,
});
```

6. Error Handling

Ensure smooth data processing with robust error handling and data validation.

Try-Catch Blocks

Wrap API calls and processing logic in `try-catch` blocks to handle errors gracefully.

```javascript
async function importData() {
  try {
    // Fetch data from external API
    const response = await axios.get<Product[]>('https://hackathon-api
s.vercel.app/api/products');
    const products = response.data as Product[];
    let counter = 1;

    // Iterate over the products
    for (const product of products) {
      let imageRef: string | null = null; // Define type for imageRef
      let catRef: string | null = null; // Define type for catRef

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

      if (product.category.name) {
        const category: Category = {
          _id: undefined, // or provide a specific ID if needed
          _type: "category",
          name: product.category.name,
          slug: slugify(product.category.name, { lower: true, strict: t
rue }) // Generate slug from category name
        };
        catRef = await createCategory(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 tim
eless design, with premium materials features as one of our most popula
r 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();
```

.

Adjustments Made to Schemas

This schema was directly copied from the original source without any modifications. No adjustments or changes have been made to the structure, attributes, or properties of the schema. It is being used as-is to maintain consistency with the original design or requirements.

PRODUCT SCHEMAS

```
1   import { defineType, defineField } from "sanity";
2
3   export const product = defineType({
4     name: "product",
5     title: "Product",
6     type: "document",
7     fields: [
8       defineField({
9         name: "category",
10        title: "Category",
11        type: "reference",
12        to: [{ type: "category" }],
13        validation: (rule) => rule.required(),
14      }),
15      defineField({
16        name: "name",
17        title: "Title",
18        validation: (rule) => rule.required(),
19        type: "string",
20      }),
21      defineField({
22        name: "slug",
23        title: "Slug",
24        validation: (rule) => rule.required(),
25        type: "slug",
26        options: {
27          source: "name", // Automatically generate the slug based on the product name
28          maxLength: 96,  // Optionally, limit the length of the slug
29        },
30      }),
31      defineField({
32        name: "image",
33        title: "Product Image",
34        type: "image",
35        validation: (rule) => rule.required(),
36      }),
37      defineField({
38        name: "price",
39        title: "Price",
40        type: "number",
41        validation: (rule) => rule.required().min(0), // Ensure price is non-negative
42      }),
43      defineField({
44        name: "quantity",
45        title: "Quantity",
46        type: "number",
47        validation: (rule) => rule.min(0), // Allow 0 for out-of-stock items
48      }),
49      defineField({
50        name: "tags",
51        title: "Tags",
52        type: "array",
53        of: [{ type: "string" }],
54      }),
55      defineField({
56        name: "description",
57        title: "Description",
58        type: "text",
59        description: "Detailed description of the product",
60      }),
61      defineField({
62        name: "features",
63        title: "Features",
64        type: "array",
65        of: [{ type: "string" }],
66        description: "List of key features of the product",
67      }),
68      defineField({
69        name: "dimensions",
70        title: "Dimensions",
71        type: "object",
72        fields: [
73          { name: "height", title: "Height", type: "string" },
74          { name: "width", title: "Width", type: "string" },
75          { name: "depth", title: "Depth", type: "string" },
76        ],
77        description: "Dimensions of the product",
78      }),
79    ],
80  });
81
```

CATEGORY SCHEMAS

```
1   import { defineType, defineField } from "sanity";
2
3   export const Category = defineType({
4     name: "category",
5     title: "Category",
6     type: "document",
7     fields: [
8       defineField({
9         name: "name",
10        title: "Name",
11        type: "string",
12        validation: (rule) =>
13          rule.required().min(2).max(50).warning("Name should be descriptive."),
14      }),
15      defineField({
16        name: "slug",
17        title: "Slug",
18        type: "slug",
19        validation: (rule) => rule.required(),
20        options: {
21          source: "name", // Automatically generate the slug from the "name" field
22          maxLength: 96, // Limit the length of the slug
23        },
24      }),
25      defineField({
26        name: "description",
27        title: "Description",
28        type: "text",
29        description: "Optional description of the category.",
30        validation: (rule) => rule.max(200),
31      }),
32    ],
33  });
34
```

Migration Steps and Tools Used

1. Preparation

- **Objective:** Verified API access and prepared a comprehensive plan to map the API response to Sanity schemas.
- **Actions Taken:**
  - o Analyzed the API structure and ensured necessary endpoints were accessible.
  - o Designed a schema mapping strategy to align the API data with Sanity's schema requirements.

---

2. Data Import Script

- **Objective:** Created a robust script to fetch, process, and import data into Sanity.
- **Actions Taken:** o Developed a script using Node.js to fetch data from the API.

o Processed the API response to format it according to Sanity schema definitions. o Imported the processed data into Sanity using the Sanity Client.

---

3. Image Handling

- **Objective:** Managed image assets by uploading and associating them with relevant documents.
- **Actions Taken:** o Utilized Sanity's Assets API to upload images efficiently. o Attached the uploaded images to the corresponding documents. o Implemented hotspot and crop options for enhanced image customization within Sanity.

---

4. Document Creation

- **Objective:** Iteratively created and updated documents in Sanity using the processed data.
- **Actions Taken:**
    - o Used the `createOrReplace` method in Sanity to ensure seamless updates or additions of documents.
    - o Performed multiple iterations to validate and refine document creation for accuracy.


Tools Used

Node.js Modules

1. **sanity/client:** Used for interacting with the Sanity API to create and manage content.
2. **axios:** Used for fetching data from the external API.
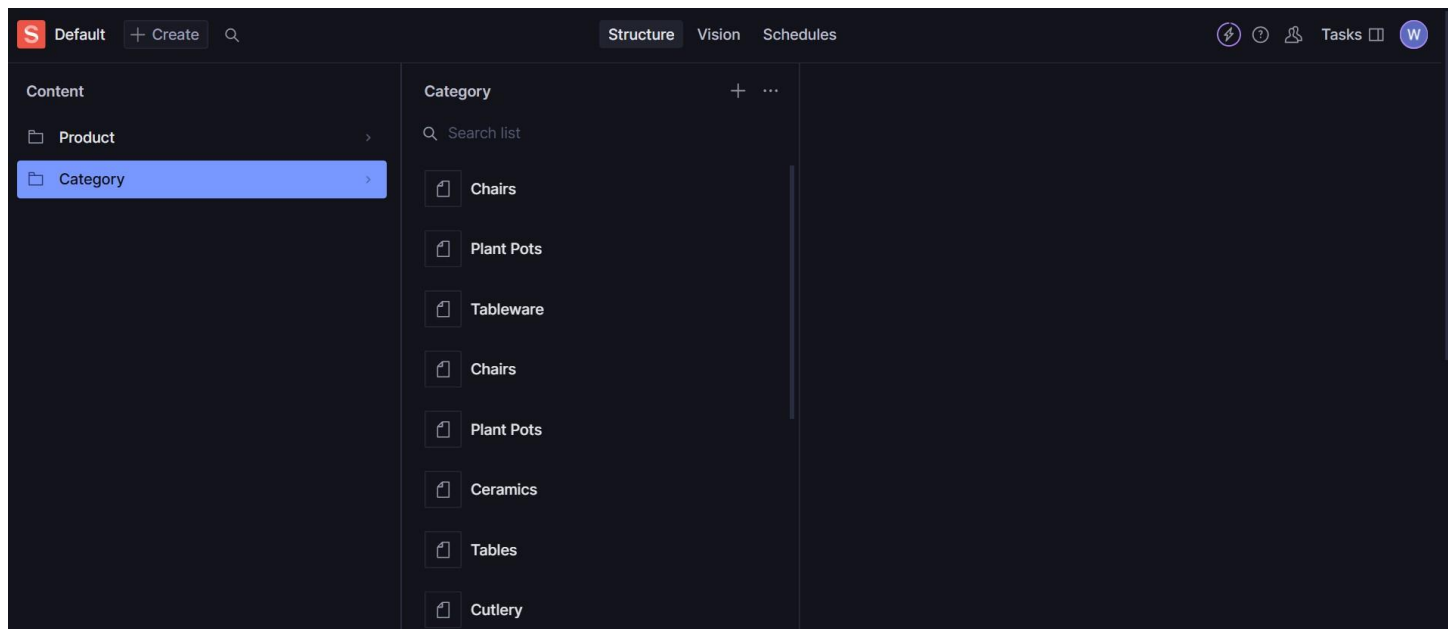3. **dotenv:** Used for managing environment variables securely.

Sanity Features

- **Hotspot and Crop Options:** Leveraged for precise image adjustments and customization.
- **Create or Replace Method:** Ensured smooth and error-free document updates or additions.

SANITY CMS FIELDS
PRODUCTS
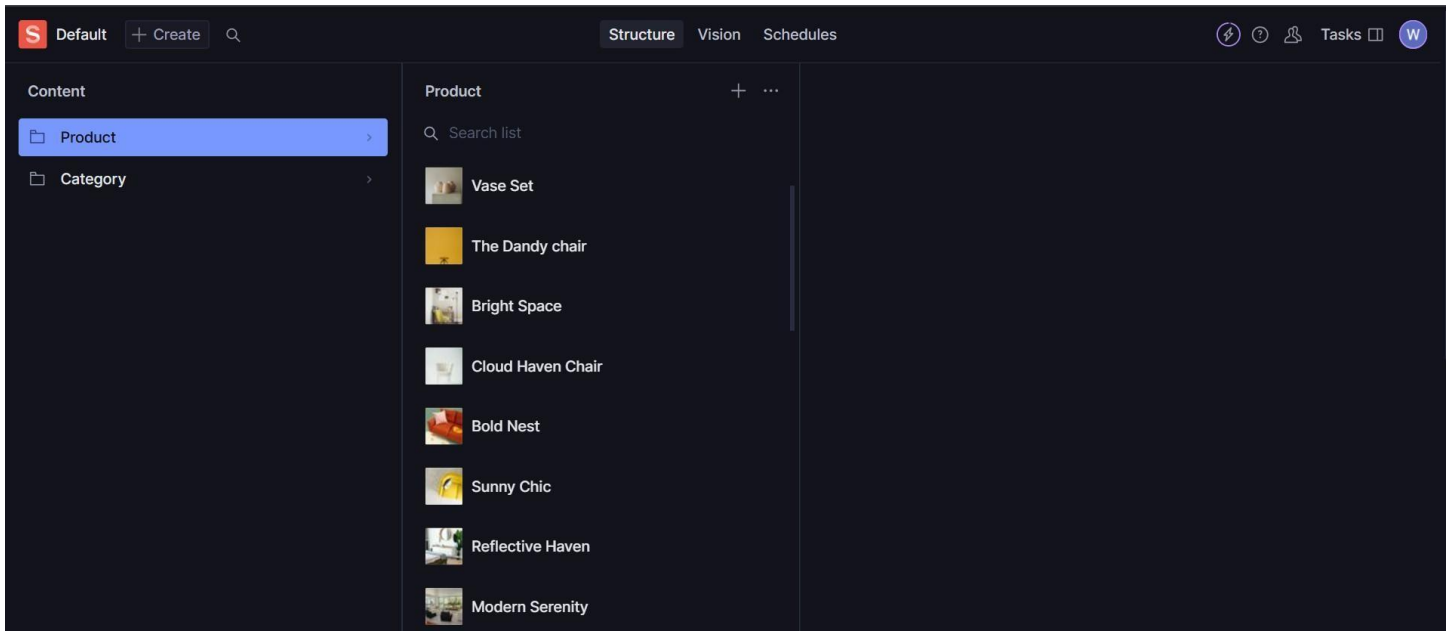
SANITY CMS FIELDS
CATEGORY

DATA DISPLAYED BY

SANITY IN FRONTEND

Day 3 Checklist:
Self-Validation Checklist:
API Understanding:

☐ ✔

Schema Validation:

☐ ✔

Data Migration:

☐ ✔

API Integration in Next.js:

☐ ✔

Submission Preparation:

☐ ✔