Hackathon-Day:03

API INTEGRATION AND DATA MIGRATION

Overview

*Welcome to Day 3 of the Hackathon! Today, we are diving into one of the most crucial aspects of building an ecommerce website - **API Integration and Data Migration** .*

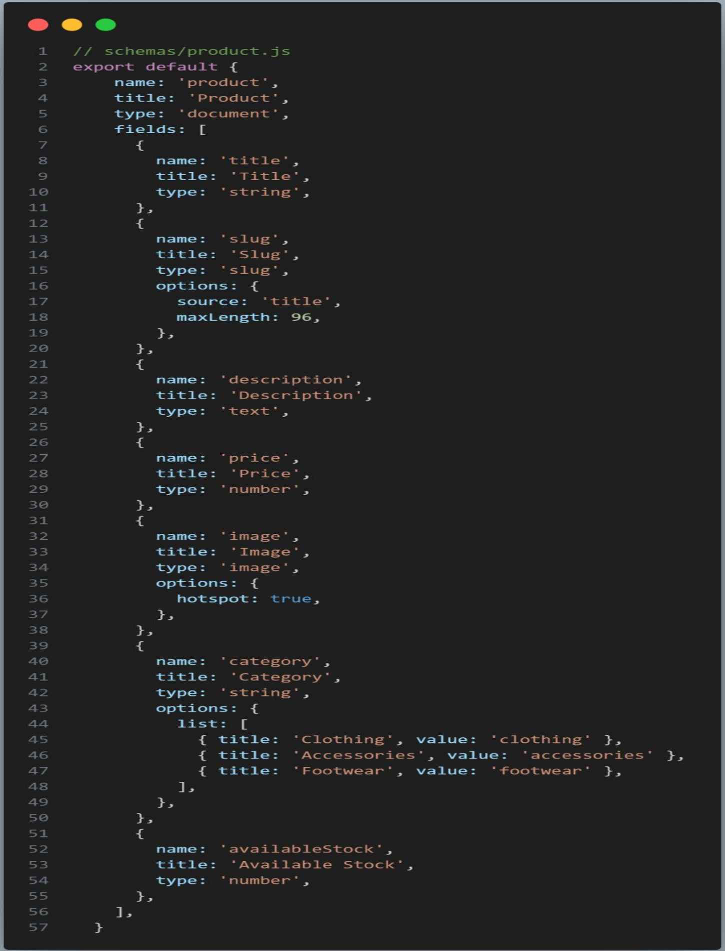
Introduction

API Integration:

Connecting our website to external data sources and services. This could include integrating payment gateways, product databases, user authentication services, and more. We will explore how to fetch and manage dynamic data to offer a seamless user experience on our ecommerce platform.

Data Migration:

Ensuring that data from legacy systems or external sources is transferred correctly into our ecommerce website. We'll focus on structuring data efficiently, moving product information, customer details, and order histories from old databases to new systems, ensuring data integrity and consistency.



```
// schemas/product.js
export default {
    name: 'product',
    title: 'Product',
    type: 'document',
    fields: [
      {
        name: 'title',
        title: 'Title',
        type: 'string',
      },
      {
        name: 'slug',
        title: 'Slug',
        type: 'slug',
        options: {
          source: 'title',
          maxLength: 96,
        },
      },
      {
        name: 'description',
        title: 'Description',
        type: 'text',
      },
      {
        name: 'price',
        title: 'Price',
        type: 'number',
      },
      {
        name: 'image',
        title: 'Image',
        type: 'image',
        options: {
          hotspot: true,
        },
      },
      {
        name: 'category',
        title: 'Category',
        type: 'string',
        options: {
          list: [
            { title: 'Clothing', value: 'clothing' },
            { title: 'Accessories', value: 'accessories' },
            { title: 'Footwear', value: 'footwear' },
          ],
        },
      },
      {
        name: 'availableStock',
        title: 'Available Stock',
        type: 'number',
      },
    ],
  }
```

API Integration and Data Migration Schema

A schema provides a blueprint or structure for the data being transferred, specifying how the data should be organized, stored, and accessed. Below is an example schema for both API integration and data migration processes.

Data Migration

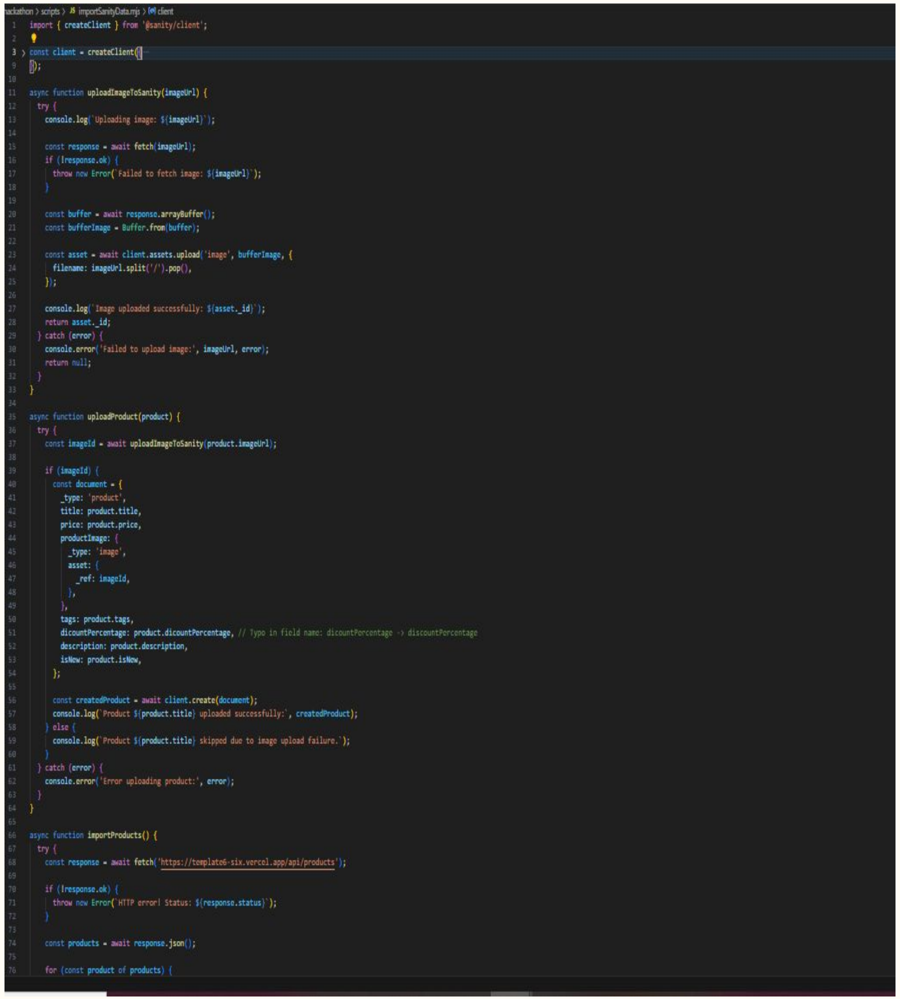
*A **data migration script** transfers data from one database or system to another. In the context of Sanity, a migration script is used to modify or transfer data within a Sanity dataset. Here's how you can write a migration script for a specific purpose, such as updating or adding the description field to all product documents.*

Prerequisites:

Sanity CLI *: Ensure you have the Sanity CLI installed (npm install -g @sanity/cli).*

API Token *: Obtain a write-enabled API token from your Sanity project settings.*

Sanity Client *: Use the Sanity JavaScript client to interact with the dataset.*



ackathon > scripts > JS importSanityData.mjs > client

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

const client = createClient({ ···
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);

    const response = await fetch(imageUrl);
    if (!response.ok) {
      throw new Error(`Failed to fetch image: ${imageUrl}`);
    }

    const buffer = await response.arrayBuffer();
    const bufferImage = Buffer.from(buffer);

    const asset = await client.assets.upload('image', bufferImage, {
      filename: imageUrl.split('/').pop(),
    });

    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
  }
}

async function uploadProduct(product) {
  try {
    const imageId = await uploadImageToSanity(product.imageUrl);

    if (imageId) {
      const document = {
        _type: 'product',
        title: product.title,
        price: product.price,
        productImage: {
          _type: 'image',
          asset: {
            _ref: imageId,
          },
        },
        tags: product.tags,
        dicountPercentage: product.dicountPercentage, // Typo in field name: dicountPercentage -> discountPercentage
        description: product.description,
        isNew: product.isNew,
      };

      const createdProduct = await client.create(document);
      console.log(`Product ${product.title} uploaded successfully:`, createdProduct);
    } else {
      console.log(`Product ${product.title} skipped due to image upload failure.`);
    }
  } catch (error) {
    console.error('Error uploading product:', error);
  }
}

async function importProducts() {
  try {
    const response = await fetch('https://template6-six.vercel.app/api/products');

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const products = await response.json();

    for (const product of products) {
```

Importing Data into Sanity

Create the Schema

First, we design the **Sanity schema**. This schema defines what data we want to store and its structure.

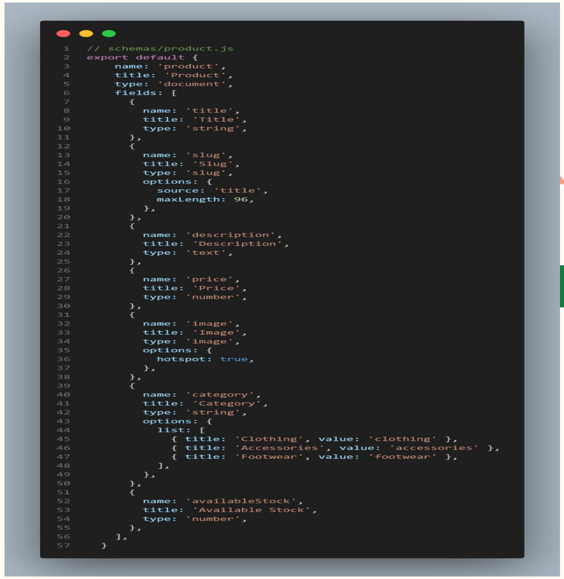
Migrate the Data

After verifying the schema and the API data structure, the next step was to write a migration script in the existing file `data-migration.mjs` that would automate the process of importing the fetched data into Sanity. A script was developed that would:

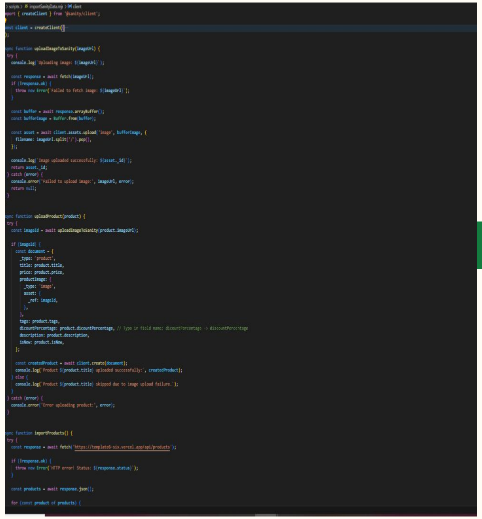
Prepare the Data.(JSON) :Ensure your data is stored in a JSON file that aligns with the Sanity schema.

Import Data into Sanity

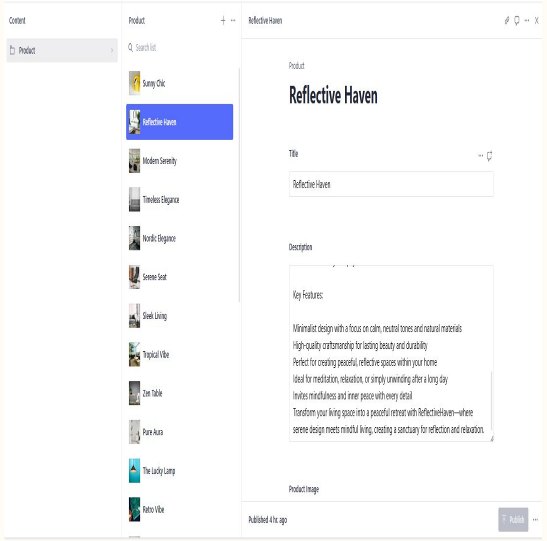
Execute the script to start importing data into Sanity. You can do this via the command line with node data-migration.mjs



```
// schemas/product.js
export default {
    name: 'product',
    title: 'Product',
    type: 'document',
    fields: [
      {
        name: 'title',
        title: 'Title',
        type: 'string',
      },
      {
        name: 'slug',
        title: 'Slug',
        type: 'slug',
        options: {
          source: 'title',
          maxLength: 96,
        },
      },
      {
        name: 'description',
        title: 'Description',
        type: 'text',
      },
      {
        name: 'price',
        title: 'Price',
        type: 'number',
      },
      {
        name: 'image',
        title: 'Image',
        type: 'image',
        options: {
          hotspot: true,
        },
      },
      {
        name: 'category',
        title: 'Category',
        type: 'string',
        options: {
          list: [
            { title: 'Clothing', value: 'clothing' },
            { title: 'Accessories', value: 'accessories' },
            { title: 'Footwear', value: 'footwear' },
          ],
        },
      },
      {
        name: 'availableStock',
        title: 'Available Stock',
        type: 'number',
      },
    ],
  }
```



```
> scripts > JS importSanityData.mjs > [@] client
nport { createClient } from '@sanity/client';

onst client = createClient({
);

sync function uploadImageToSanity(imageUrl) {
 try {
   console.log(`Uploading image: ${imageUrl}`);

   const response = await fetch(imageUrl);
   if (!response.ok) {
     throw new Error(`Failed to fetch image: ${imageUrl}`);
   }

   const buffer = await response.arrayBuffer();
   const bufferImage = Buffer.from(buffer);

   const asset = await client.assets.upload('image', bufferImage, {
     filename: imageUrl.split('/').pop(),
   });

   console.log(`Image uploaded successfully: ${asset._id}`);
   return asset._id;
 } catch (error) {
   console.error('Failed to upload image:', imageUrl, error);
   return null;
 }


sync function uploadProduct(product) {
 try {
   const imageId = await uploadImageToSanity(product.imageUrl);

   if (imageId) {
     const document = {
       _type: 'product',
       title: product.title,
       price: product.price,
       productImage: {
         _type: 'image',
         asset: {
           _ref: imageId,
         },
       },
       tags: product.tags,
       dicountPercentage: product.dicountPercentage, // Typo in field name: dicountPercentage -> discountPercentage
       description: product.description,
       isNew: product.isNew,
     };

     const createdProduct = await client.create(document);
     console.log(`Product ${product.title} uploaded successfully:`, createdProduct);
   } else {
     console.log(`Product ${product.title} skipped due to image upload failure.`);
   }
 } catch (error) {
   console.error('Error uploading product:', error);
 }


sync function importProducts() {
 try {
   const response = await fetch('https://template6-six.vercel.app/api/products');

   if (!response.ok) {
     throw new Error(`HTTP error! Status: ${response.status}`);
   }

   const products = await response.json();

   for (const product of products) {
```



Product

Reflective Haven

Title

Reflective Haven

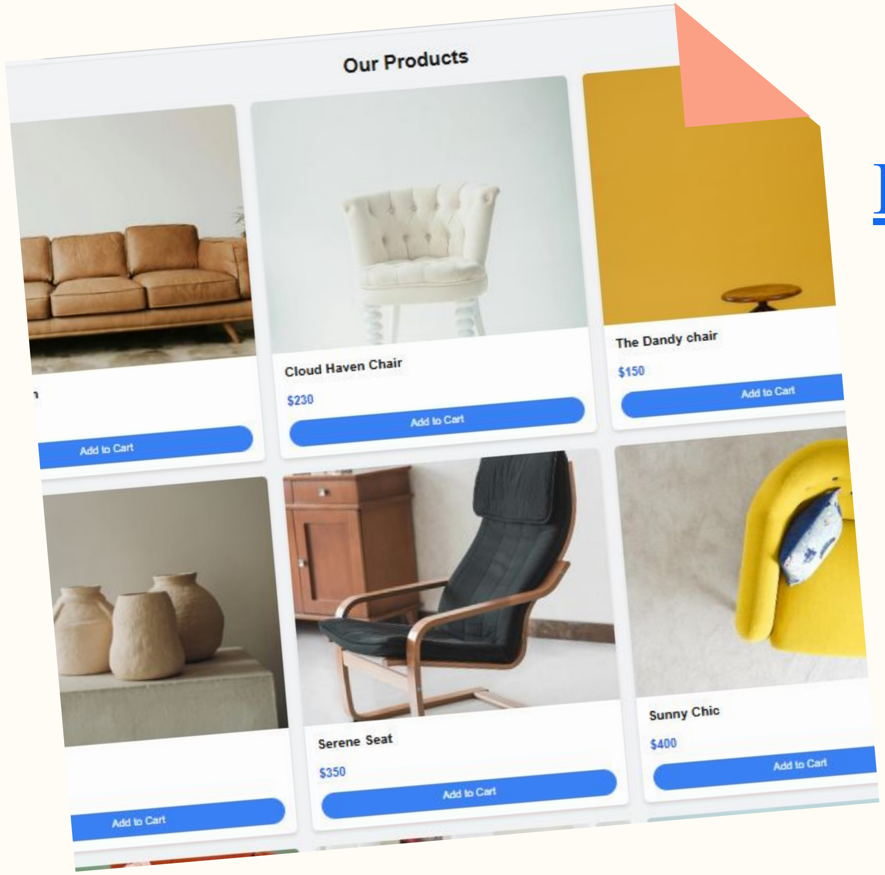
Description

Key Features:

Minimalist design with a focus on calm, neutral tones and natural materials
High-quality craftsmanship for lasting beauty and durability
Perfect for creating peaceful, reflective spaces within your home
Ideal for meditation, relaxation, or simply unwinding after a long day
Invites mindfulness and inner peace with every detail
Transform your living space into a peaceful retreat with ReflectiveHaven—where serene design meets mindful living, creating a sanctuary for reflection and relaxation.

Product Image

Published 4 hr. ago



Display Data on the Frontend

In the Home component, you can map over the products array and display each product's information, such as the title, description, and price, along with the image.

- **Images**: Use the `urlFor` function to create the correct image URL from the Sanity data.
- **Product Details**: Use the data retrieved from Sanity to display product details like the title, price, and description.

Conclusion

Incorporating API integration and data migration into a Next.js project requires a well-organized approach to ensure data flows smoothly from the API to the frontend. First, understanding the structure and details of the provided API is crucial, as it helps in validating and adjusting the Sanity schema to match the data. Once the schema is set up, the next step involves migrating the data into Sanity, which can be done efficiently using migration scripts or import tools. Proper integration in Next.js ensures that the fetched data is dynamically rendered on the frontend, providing a seamless user experience.

Finally, attention to error handling is essential throughout the process to ensure robustness and prevent data inconsistencies. By following these steps, you can successfully integrate API data, migrate it to Sanity, and display it on your website, creating a fully functional and scalable application.

Self-Validation Checklist:

API Understanding:

- ✔

Schema Validation:

- ✔

Data Migration:

- ✔

API Integration in Next.js:

- ✔

Submission Preparation:

- ✔

Thank u

Presented by Niba Khan ❤️🙂