# Hackathon 3

# API Integration and Data Migration

# [General E-Commerce]

## Introduction

This document the process of integration the sanity CMS API into a Next.js application for managing and displaying data. The main objective of this phase was to integrate the sanity CMS API for migrating product data, as well as to adjust the schema to organize the product information. The process includes setting up the API connection between the backend and frontend, enabling efficient data retrieval and visualization, creating the schema for the product page, and running the data migration script to populate the local database with product data.

## Sanity API Integration Process

The Sanity API process involved fetching product data from an external APIs and utilizing it in. The Next.js application,

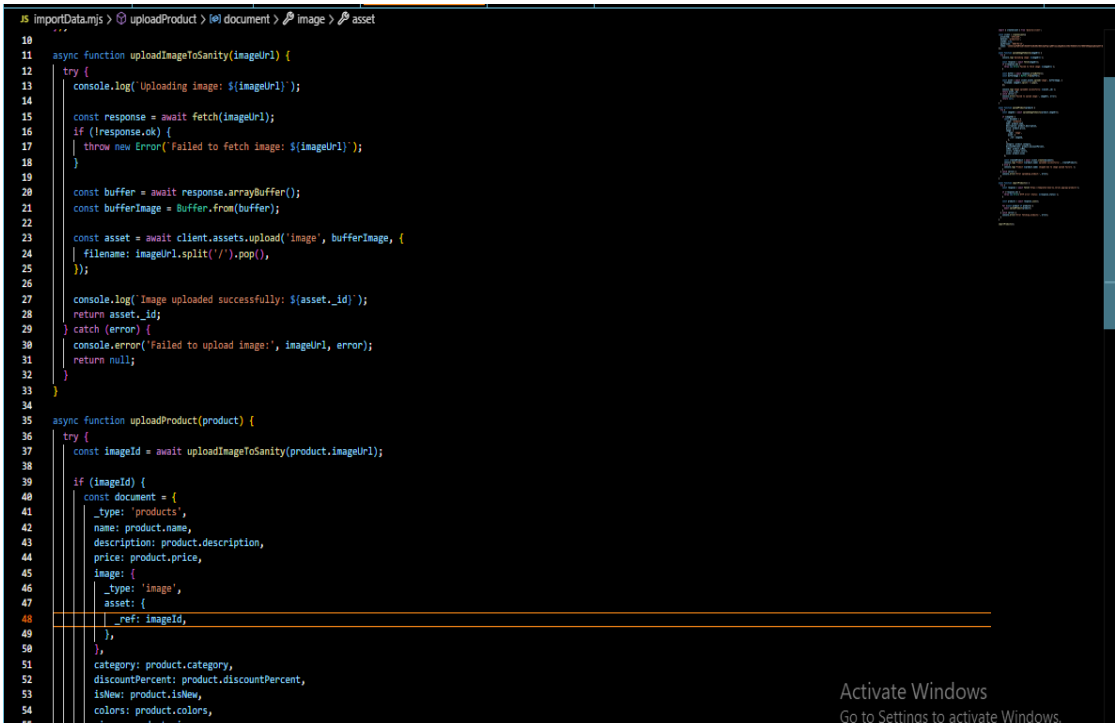
The following steps detail the approach used for seamless integration.

1) Fetching Data from the External API:
    - A custom importData.mjs was created to fetch product data.
    - The API endpoint used to fetch the data was: htttps://template1-neon-nu-vercel.app/api/products.

2) Data to be fetched:
    - Product Data

# Migration Script

The migration was handled via the importData.mjs script. Below is the code used to automate the import of product data.



```
JS importData.mjs > uploadProduct > document > image > asset
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: 'products',
        name: product.name,
        description: product.description,
        price: product.price,
        image: {
          _type: 'image',
          asset: {
            _ref: imageId,
          },
        },
        category: product.category,
        discountPercent: product.discountPercent,
        isNew: product.isNew,
        colors: product.colors,
```



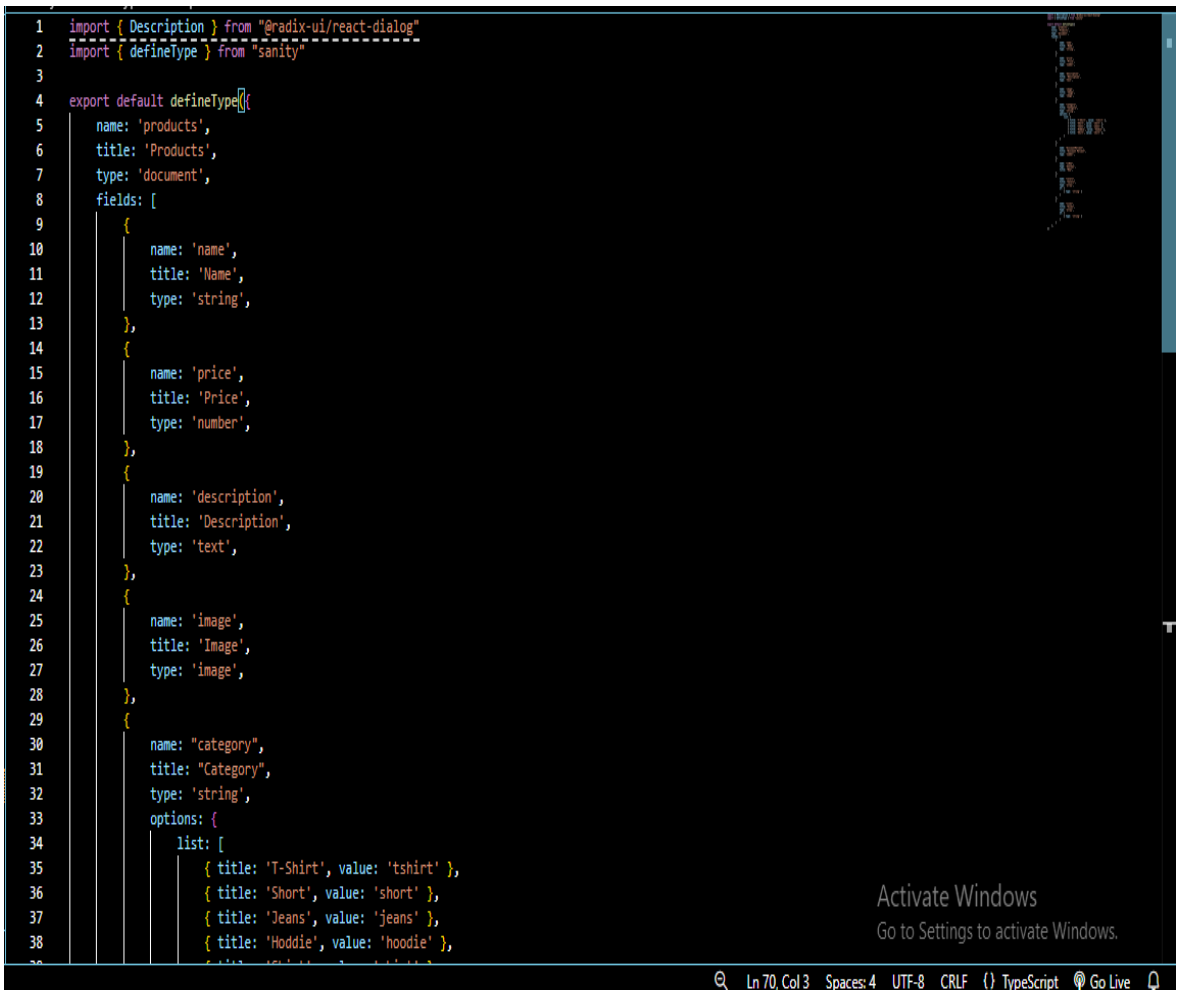
```
JS importData.mjs > uploadProduct > document > image > asset
async function uploadProduct(product) {
      const createdProduct = await client.create(document);
      console.log(`Product ${product.name} uploaded successfully:`, createdProduct);
    } else {
      console.log(`Product ${product.name} skipped due to image upload failure.`);
    }
  } catch (error) {
    console.error('Error uploading product:', error);
  }
}

async function importProducts() {
  try {
    const response = await fetch('https://template1-neon-nu.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) {
      await uploadProduct(product);
    }
  } catch (error) {
    console.error('Error fetching products:', error);
  }
}

importProducts();
```

## Schema Configuration

To manage the product data efficiently, a single schema file was created:

- **Product.ts:**
  This schema file is designed to manage product specific data; i.e.: product name, description, image, price, category, discount Percent, is New, color and sizes.

This schema ensures that product data is structured properly for easy querying and retrieval.




- **Script Creation:**

The script named **importData.mjs**, was responsible for automating the process of importing product data into the local database.

## Migration Execution

After configuring the schema and the APIs data structure, the next step was to write a migration script that would automate the process of importing the fetch data into sanity. A script was created that would:

- Fetch data from the API.
- Format the data according to the Sanity schema.
- Use the Sanity Client to push the data into Sanity.

The data migration was triggered using the following terminal command:

```
Node importData.mjs
```

```
    discountPercent: 0,
    isNew: true,
    price: 240,
    imageUrl: 'https://cdn.sanity.io/images/wf4viek5/production/68b4debeb5624646a50c99588354ca14bc43f6c0-295x298.png'
  },
  {
    name: 'Black Athletic Jogger Pants with Side Stripes',
    category: 'jeans',
    discountPercent: 0,
    isNew: true,
    price: 180,
    imageUrl: 'https://cdn.sanity.io/images/wf4viek5/production/1780596a854c7fafc6b9e10558b3da006ad6a39b-195x258.jpg',
    _id: '9pIJ0O0PMKhFhzCfbNPnaJ'
  },
  {
    imageUrl: 'https://cdn.sanity.io/images/wf4viek5/production/ec566c8c907ab673e5779342452a97a14f8b2cff-295x298.png',
    _id: 'Krp64FxN7xGb8FK9AFMyoM',
    name: 'LOOSE FIT BERMUDA SHORTS',
    category: 'short',
    discountPercent: 20,
    isNew: false,
    price: 78
  },
  {
    _id: 'Krp64FxN7xGb8FK9AFN2CV',
    name: 'Beige Slim-Fit Jogger Pants',
    category: 'jeans',
    discountPercent: 10,
    isNew: true,
    price: 269,
    imageUrl: 'https://cdn.sanity.io/images/wf4viek5/production/7739d01f547a127eca9fdfdb267935974e8d2b83-278x296.png'
  }
]
```

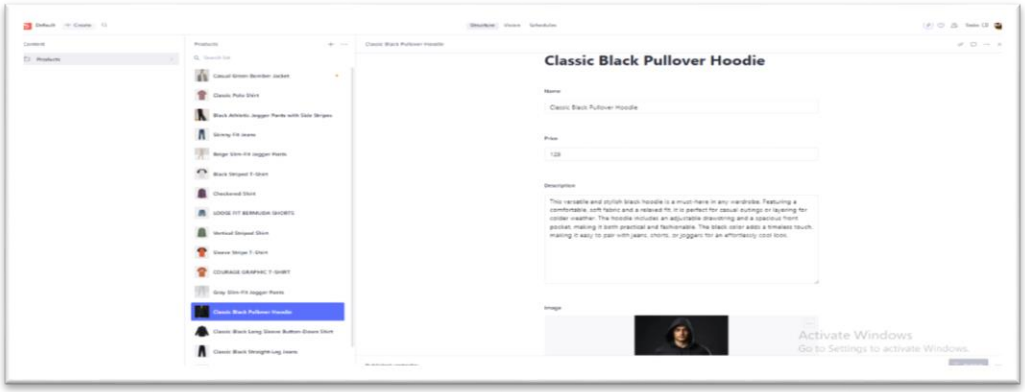



This command executes the importData.mjs script, which fetched the product data from sanity CMS and populated the local database with the necessary information.

## Migration Verification

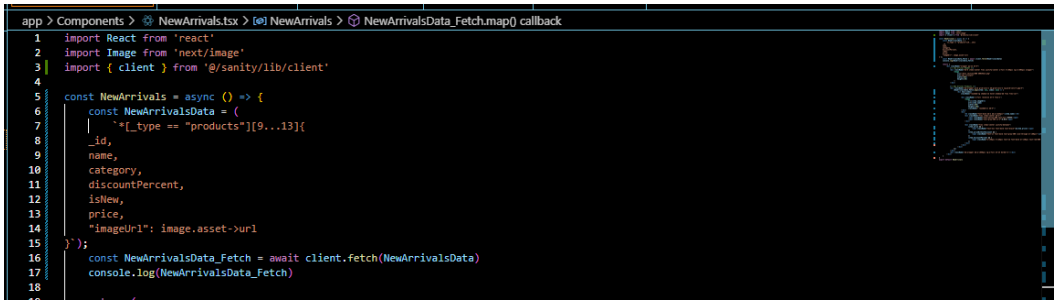
Once the data was migrate, the following steps were to verify the success of the migration:

- The local project was run and the /studio/structure endpoint was accessed to confirm that the migrated data appeared in Sanity.
- The same **Sanity login Credentials** used for the CMS were employed verify the accuracy of the data.

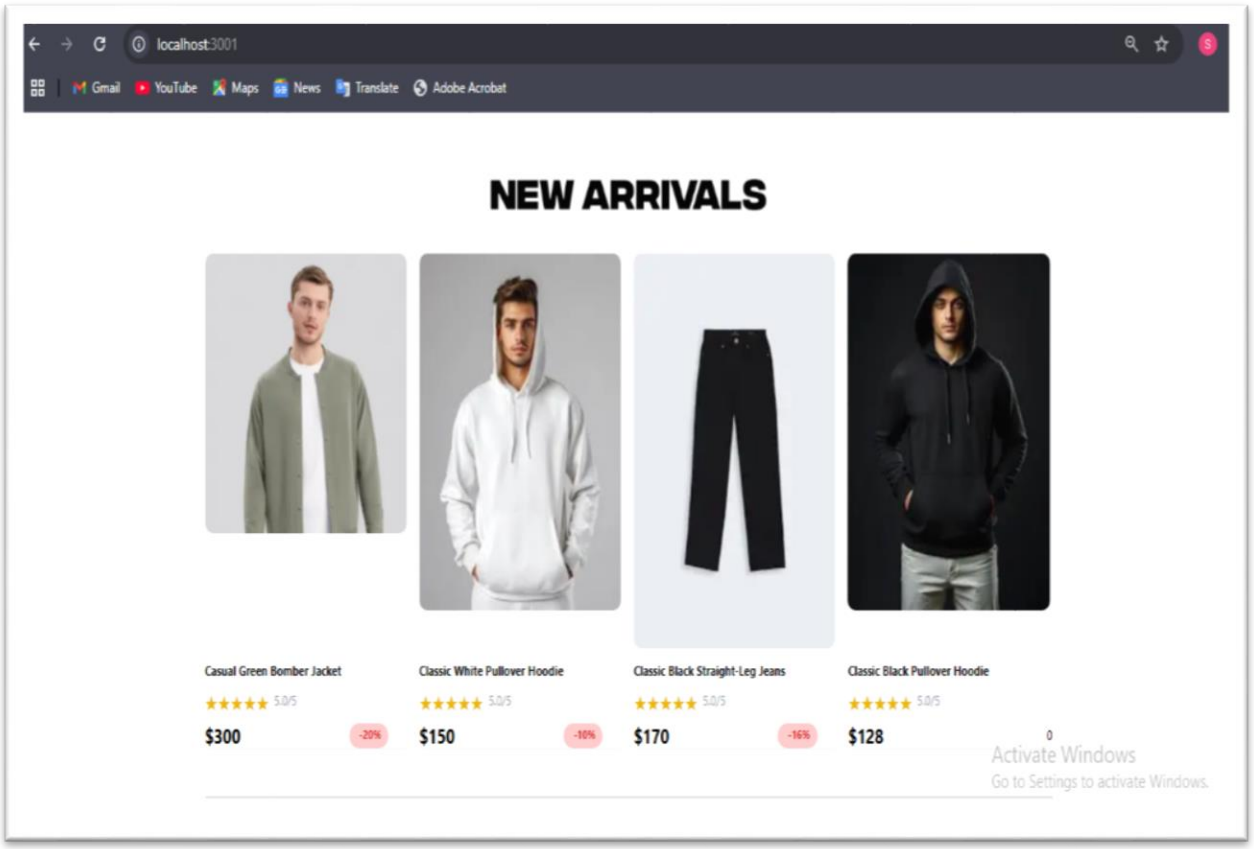


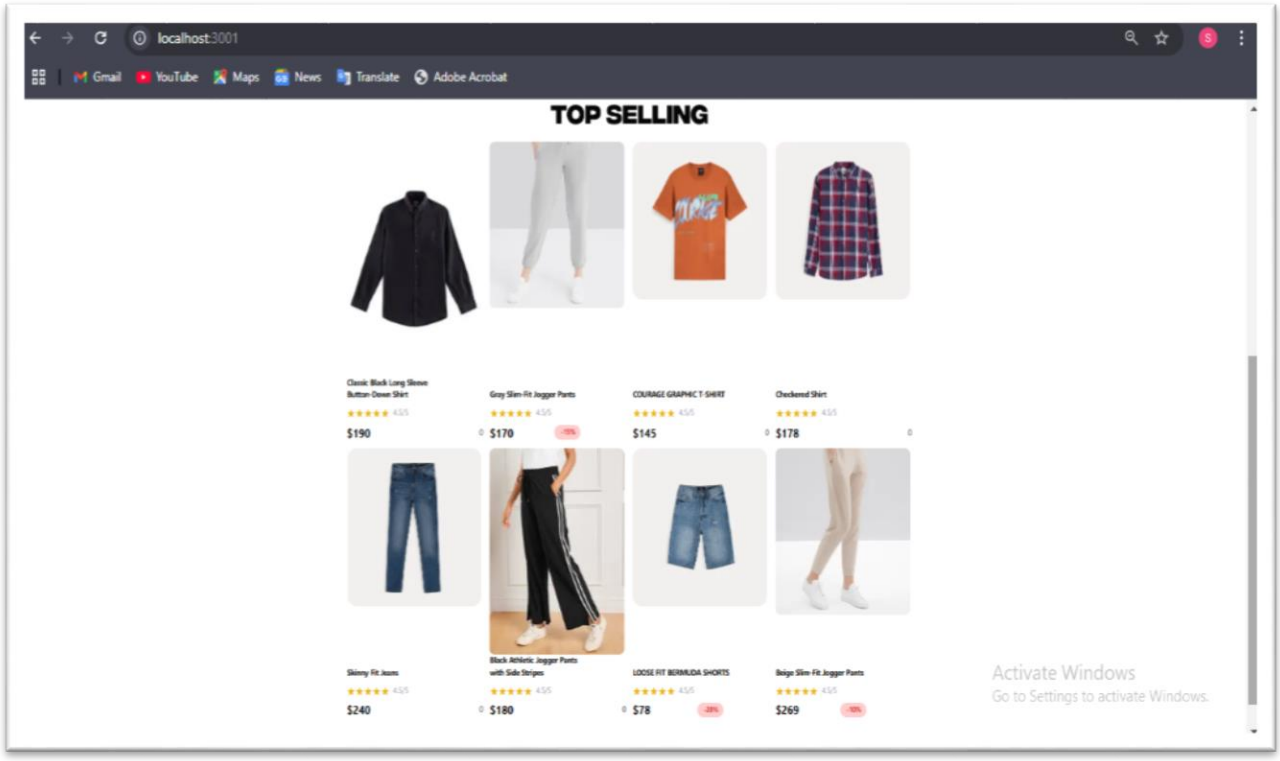
## Sanity Data to Frontend

After migration verification Sanity data show in Frontend (Webpage) through Groq Query as shown in below picture.




# Data Successfully Displayed In the Frontend






## Final Check List

| understand | Schema Validation | Data Migration | API Integration | Submission Presentation |
|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ |