Day 3: API Integration and Data Migration

Objective:

The goal of Day 3 is to integrate APIs and migrate data into Sanity CMS, creating a functional backend for the clothing marketplace. By replicating real-world scenarios, this exercise equips us to meet diverse client needs, such as integrating headless APIs or transferring data from popular eCommerce platforms.

Overview: This report discusses the integration of product and category information from an external API into the backend system of Shop.co, a clothing e-commerce site. The integration uses Sanity CMS for managing content and Next.js for frontend rendering.

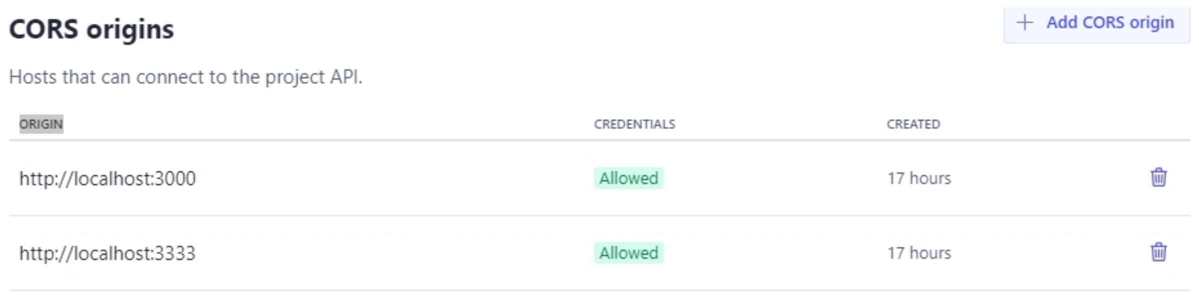
The primary goals of this integration include:

1. Retrieving data from an external API.
2. Storing and organizing this data in Sanity CMS.
3. Dynamically displaying the data on the frontend of the platform.

Step 1: Fetching Data From an External API

Choose the External API

For this exercise, let's assume you're integrating with an external eCommerce API (e.g., Shopify, WooCommerce, or any custom API for product data). You will likely need to fetch product data like product names, descriptions, images, prices, and stock levels.


CORS origins

Hosts that can connect to the project API.

ORIGIN	CREDENTIALS	CREATED
http://localhost:3000	Allowed	17 hours
http://localhost:3333	Allowed	17 hours

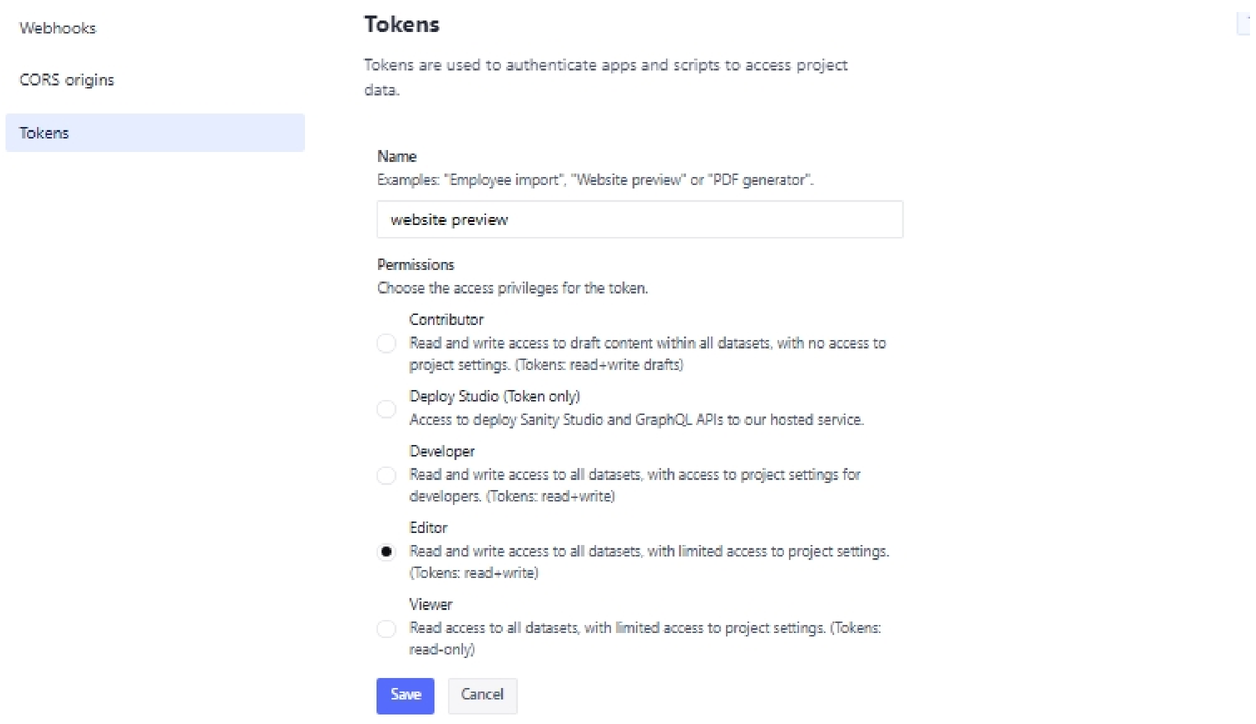
Step 2:Storing and Managing the Data in Sanity CMS

1. **Log in to Sanity.io**:
    - Go to Sanity.io and log in to your account.
2. **Go to the Project Settings**:
    - From the Sanity dashboard, click on the **"Manage project"** link of the project you want to generate the token for.
    - This will take you to the **Project Settings** page.
3. **Navigate to the API Section**:
    - In the **Project Settings** page, on the left sidebar, click on the **API** section.
4. **Create a New API Token**:
    - In the API section, you will see an option for **Tokens**. Click on **"Create token"**.
    - You will be asked to specify the name of the token (e.g., "Frontend API Token") and choose the **permissions** for this token.

Choose the appropriate permissions based on your use case:

- **Read**: If your application only needs to read data (e.g., displaying products or categories).
- **Write**: If you want to push or update data into the Sanity CMS.
- **Delete**: If you want to delete data (be cautious with this option).

Best Practice: For a frontend application that only needs to fetch data, select **"Read"** permission.



Tokens

Tokens are used to authenticate apps and scripts to access project data.

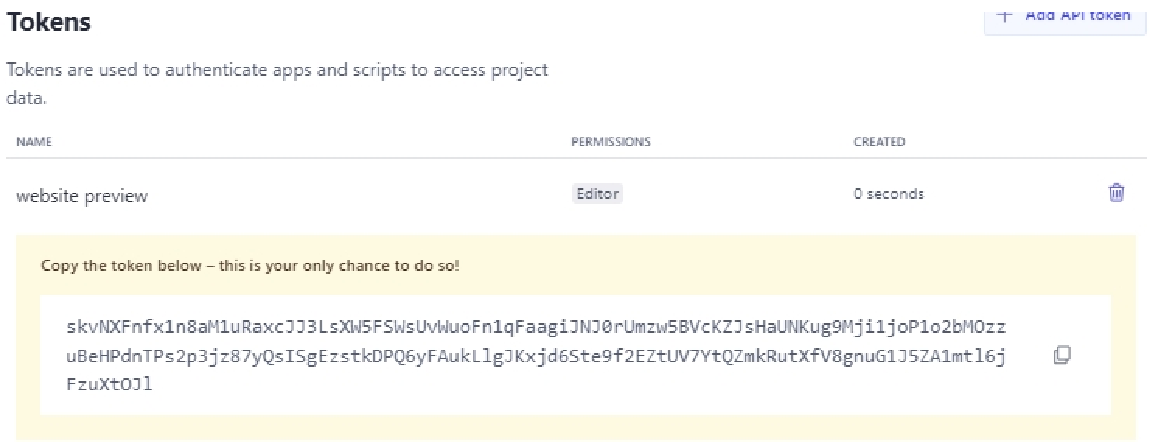
Name
Examples: "Employee import", "Website preview" or "PDF generator".

website preview

Permissions
Choose the access privileges for the token.

- Contributor
  Read and write access to draft content within all datasets, with no access to project settings. (Tokens: read+write drafts)
- Deploy Studio (Token only)
  Access to deploy Sanity Studio and GraphQL APIs to our hosted service.
- Developer
  Read and write access to all datasets, with access to project settings for developers. (Tokens: read+write)
- [✓ Editor]
  Read and write access to all datasets, with limited access to project settings. (Tokens: read+write)
- Viewer
  Read access to all datasets, with limited access to project settings. (Tokens: read-only)

1. **Save the Token**:
   - After specifying the permissions, click **"Save"**. You will be shown your new API token. **Copy and store the token** securely, as you won't be able to see it again once you leave the page.



Tokens

Tokens are used to authenticate apps and scripts to access project data.

NAME	PERMISSIONS	CREATED
website preview	Editor	0 seconds

Copy the token below – this is your only chance to do so!

```
skvNXFnfx1n8aM1uRaxcJJ3LsXW5FSWsUvWuoFn1qFaagiJNJ0rUmzw5BVcKZJsHaUNKug9Mji1joP1o2bMOzz
uBeHPdnTPs2p3jz87yQsISgEzstkDPQ6yFAukLlgJKxjd6Ste9f2EZtUV7YtQZmkRutXfV8gnuG1J5ZA1mtl6j
FzuXtOJl
```

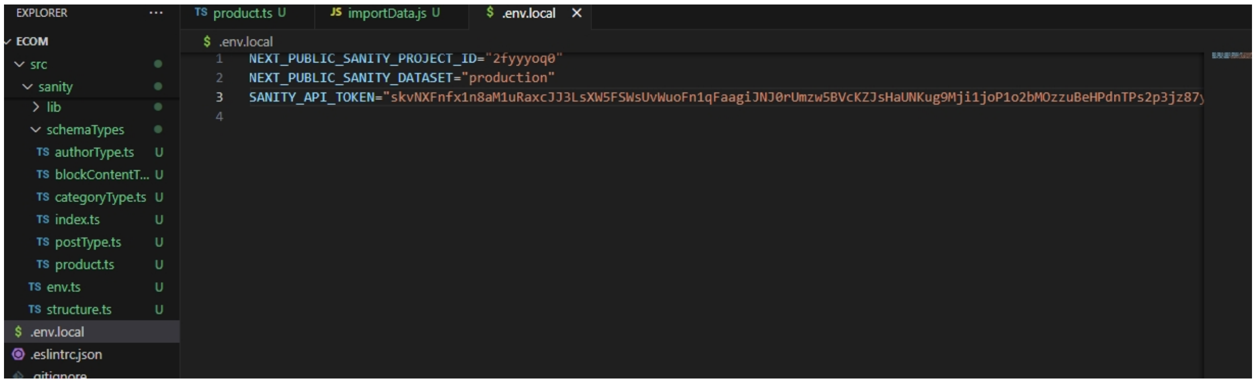
Modifications in .env.local

For this project, I added the following key environment variables to the .env.local file:

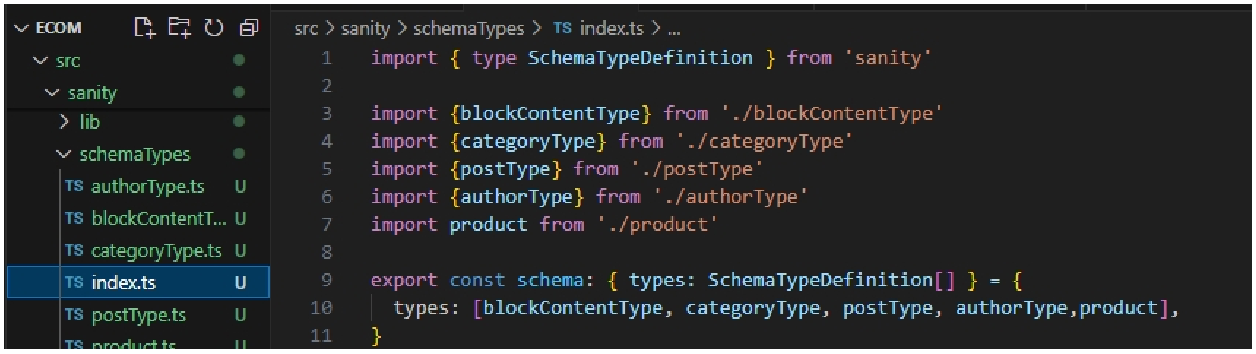
1.SANITY_API_TOKEN: This is the token used to authenticate API requests to Sanity CMS. This token was generated in the Sanity dashboard and grants the necessary permissions (read/write) to interact with the Sanity CMS.

2.**SANITY_PROJECT_ID**: This is the unique identifier for my Sanity project, which is required for connecting to the correct project in Sanity CMS.

3. **SANITY_DATASET**: This specifies the dataset I'm working with (e.g., "production"). The dataset is where all the content (such as products, categories, and other documents) will be stored.

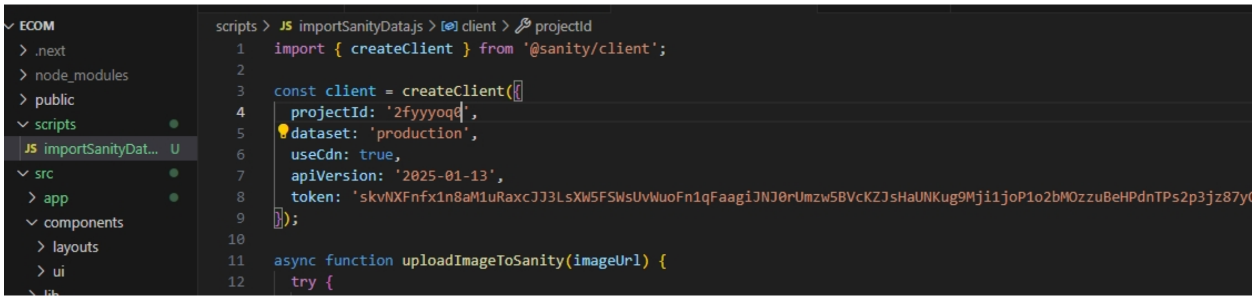


```
NEXT_PUBLIC_SANITY_PROJECT_ID="2fyyyoq0"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN="skvNXFnfx1n8aM1uRaxcJJ3LsXW5FSWsUvWuoFn1qFaagiJNJ0rUmzw5BVcKZJsHaUNKug9Mji1joP1o2bMOzzuBeHPdnTPs2p3jz87y
```

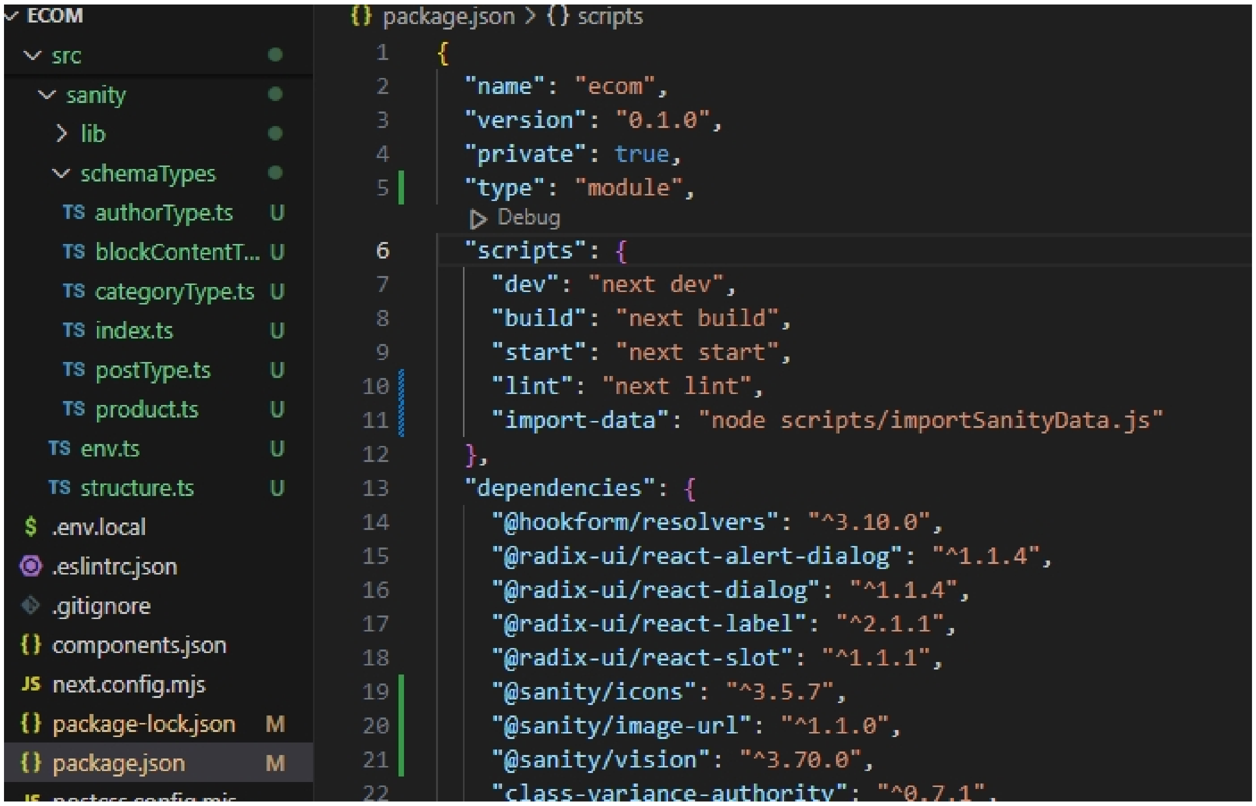


```
import { type SchemaTypeDefinition } from 'sanity'

import {blockContentType} from './blockContentType'
import {categoryType} from './categoryType'
import {postType} from './postType'
import {authorType} from './authorType'
import product from './product'

export const schema: { types: SchemaTypeDefinition[] } = {
  types: [blockContentType, categoryType, postType, authorType,product],
}
```

Sanity Client Setup: In **importSanityData.js**, the script uses the **Sanity Client API** to authenticate and interact with Sanity's backend. The client is configured using API tokens, project IDs, and datasets, which are securely stored in environment variables.



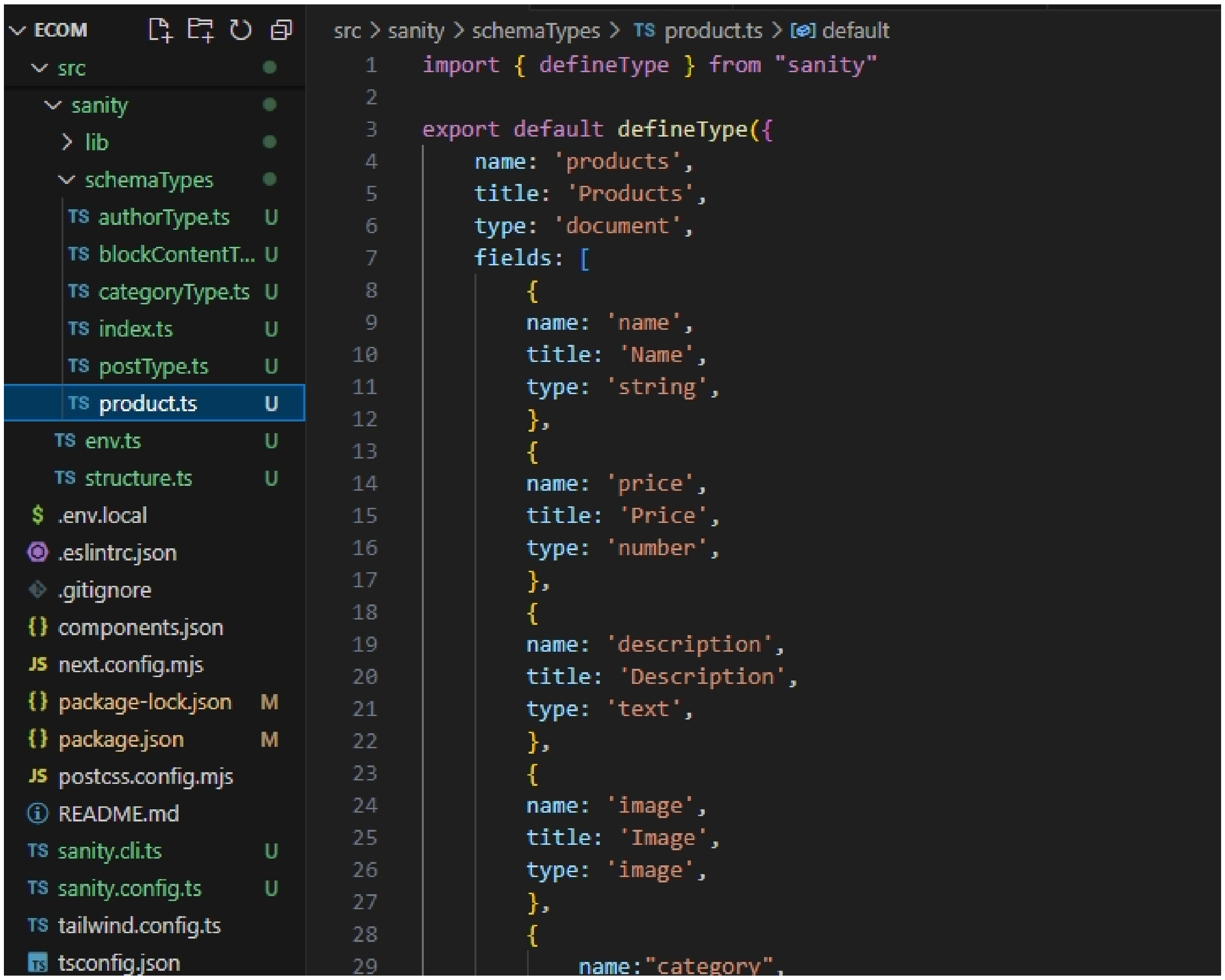
```
import { createClient } from '@sanity/client';

const client = createClient({
  projectId: '2fyyyoq0',
  dataset: 'production',
  useCdn: true,
  apiVersion: '2025-01-13',
  token: 'skvNXFnfx1n8aM1uRaxcJJ3LsXW5FSWsUvWuoFn1qFaagiJNJ0rUmzw5BVcKZJsHaUNKug9Mji1joP1o2bMOzzuBeHPdnTPs2p3jz87y
});

async function uploadImageToSanity(imageUrl) {
  try {
```



```json
{
  "name": "ecom",
  "version": "0.1.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "import-data": "node scripts/importSanityData.js"
  },
  "dependencies": {
    "@hookform/resolvers": "^3.10.0",
    "@radix-ui/react-alert-dialog": "^1.1.4",
    "@radix-ui/react-dialog": "^1.1.4",
    "@radix-ui/react-label": "^2.1.1",
    "@radix-ui/react-slot": "^1.1.1",
    "@sanity/icons": "^3.5.7",
    "@sanity/image-url": "^1.1.0",
    "@sanity/vision": "^3.70.0",
    "class-variance-authority": "^0.7.1",
```

Prepare Data for Sanity Schema

Just like with API integration, map the CSV data to the Sanity CMS schema. If the data is in CSV format, you can use a CSV parser:



```
import { defineType } from "sanity"

export default defineType({
    name: 'products',
    title: 'Products',
    type: 'document',
    fields: [
        {
        name: 'name',
        title: 'Name',
        type: 'string',
        },
        {
        name: 'price',
        title: 'Price',
        type: 'number',
        },
        {
        name: 'description',
        title: 'Description',
        type: 'text',
        },
        {
        name: 'image',
        title: 'Image',
        type: 'image',
        },
        {
            name:"category",
```

Testing and Validation

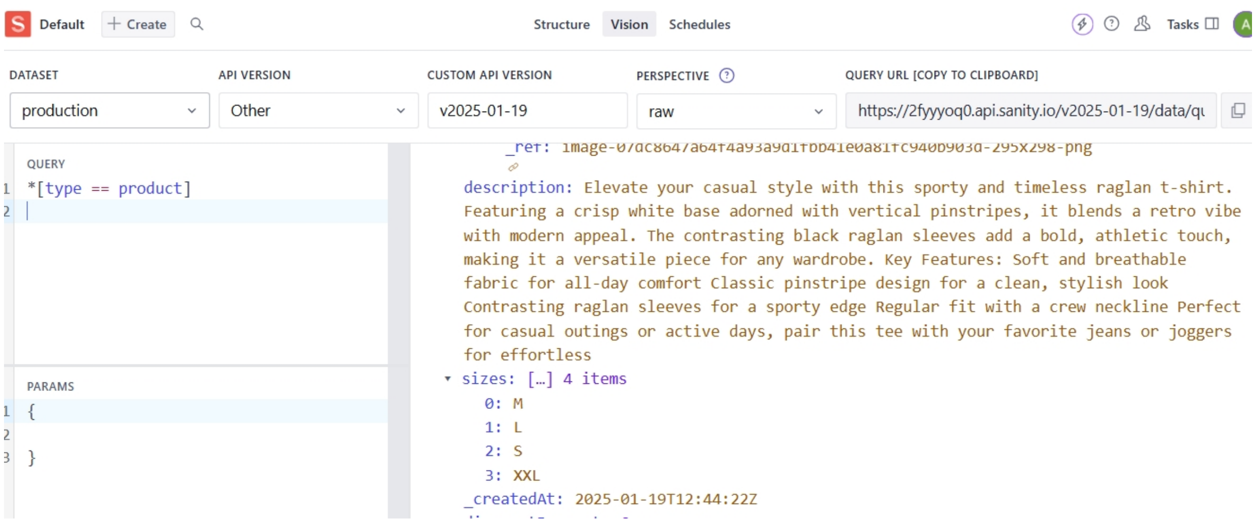
1. Validate Data

After pushing data, check your Sanity Studio to ensure that the data appears correctly. You should see your products listed with all the attributes (e.g., name, price, category).

2. Error Handling

Ensure that your scripts handle errors gracefully. For instance:

- Log errors during data fetching and pushing.
- Handle API rate limits.
- Ensure your mapping logic accounts for missing or incomplete data.



```
*[type == product]
```

```
{

}
```

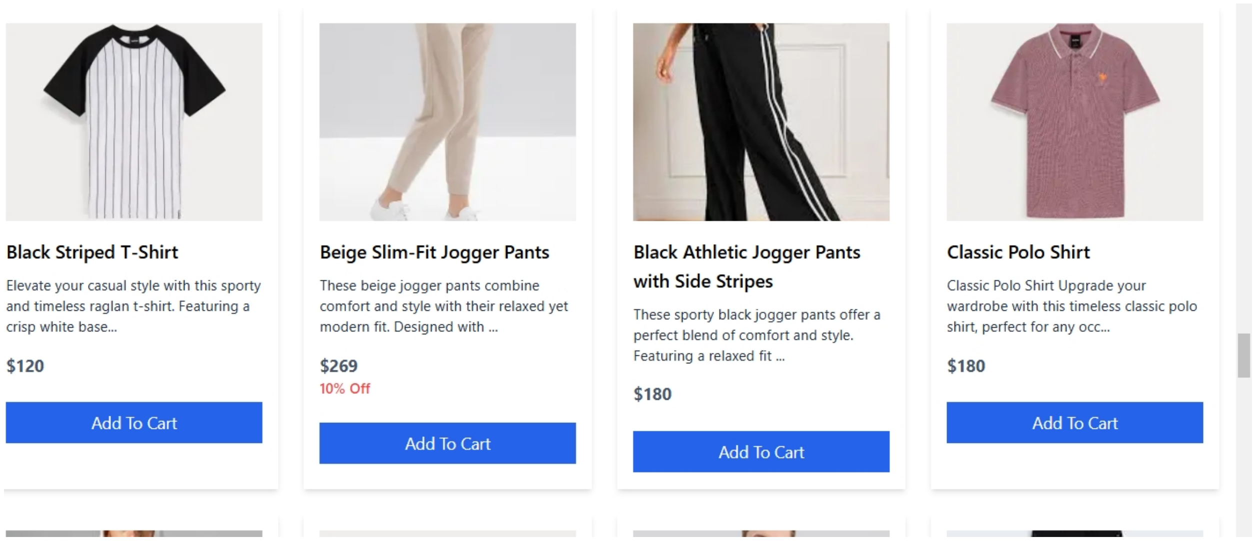
```
    _ref: image-07dc8647a64f4a93a9d1fbb41e0a81fc940b903d-295x298-png
  description: Elevate your casual style with this sporty and timeless raglan t-shirt.
  Featuring a crisp white base adorned with vertical pinstripes, it blends a retro vibe
  with modern appeal. The contrasting black raglan sleeves add a bold, athletic touch,
  making it a versatile piece for any wardrobe. Key Features: Soft and breathable
  fabric for all-day comfort Classic pinstripe design for a clean, stylish look
  Contrasting raglan sleeves for a sporty edge Regular fit with a crew neckline Perfect
  for casual outings or active days, pair this tee with your favorite jeans or joggers
  for effortless
▾ sizes: […] 4 items
    0: M
    1: L
    2: S
    3: XXL
  _createdAt: 2025-01-19T12:44:22Z
```

Step 3: Display the Fetched Data Dynamically On the Frontend

In the Home component above, we're using the products prop (fetched from Sanity CMS) to display product information. This includes:

- **Product Image**: We assume the product schema in Sanity has an image field, and we're displaying it using the img tag.
- **Product Title and Description**: Displaying product names and descriptions.
- **Product Price**: Displaying the product price.

Make sure you adjust the query to match your Sanity schema. For instance, if you have different fields for the price, title, or description, adjust the query accordingly.



Black Striped T-Shirt
Elevate your casual style with this sporty and timeless raglan t-shirt. Featuring a crisp white base...
$120

Beige Slim-Fit Jogger Pants
These beige jogger pants combine comfort and style with their relaxed yet modern fit. Designed with ...
$269
10% Off

Black Athletic Jogger Pants with Side Stripes
These sporty black jogger pants offer a perfect blend of comfort and style. Featuring a relaxed fit ...
$180

Classic Polo Shirt
Classic Polo Shirt Upgrade your wardrobe with this timeless classic polo shirt, perfect for any occ...
$180

Conclusion:

The integration of the API for ECOMMERCE WEBSITE has been successfully executed. This process involved retrieving data, storing it in Sanity CMS, and dynamically displaying it on the frontend. The attached screenshots confirm the successful implementation at each stage. With this integration, managing products and categories has become more efficient, offering a smooth experience for both administrators and users.