Day 3 - API Integration Report - [MArtNow] Hackathon

Submission Requirements for Day 3 - API Integration Report

Document Title:
"Day 3 - API Integration Report - [Your Marketplace Name]"

What to Submit:

- **Report Documentation**:
  - Detailed description of the API integration process.
  - Adjustments made to schemas.
  - Migration steps and tools used during the integration.
- **Screenshots**:
  - Screenshots of API calls.
  - Screenshots of data successfully displayed in the frontend.
  - Screenshots of populated Sanity CMS fields.
- **Code Snippets**:
  - Code snippets demonstrating API integration.
  - Code snippets for migration scripts.

Document Description:
This report outlines the key activities and achievements on Day 3 of the [MArtNow] Hackathon. The focus for the day was integrating APIs and Migration or import in to Sanity. Fetch the Sanity date in to our frontend UI. It will improve its functionality and ensure seamless connectivity. The primary objectives were to optimize the system for performance, data exchange, and user experience.

$$\begin{tikzcd}\text{API Import} \arrow[r] & \text{Sanity} \arrow[r] & \text{Fetch Data in to frontend UI}\end{tikzcd}$$

First We have start to Import Data in to Provided API into Sanity Studio.

1. Sanity CMS Integration

Sanity is a headless CMS that allows you to manage content for your projects. Below is a step-by-step guide for integrating Sanity into your project.

1.1 Install Sanity CLI

To begin, install Sanity's Command-Line Interface (CLI) globally using the following command:

```
npm install -g @sanity/cli
```

1.2 Create a Sanity Project

After installing Sanity CLI, the next step is to log into Sanity and create a new project. If you already have an account, log in. Otherwise, log in using your Gmail or GitHub account. Then, run the following command to create a new project:

```
sanity init
```

You will be prompted to:

1. **Choose a project name**: Enter a name for your project (e.g., "MArtNow").
2. **Select a dataset configuration**: Choose between "Blank" or "Blog" (Blank is recommended for this case).
3. **Select a template**: Choose an appropriate template based on your project's requirements (e.g., "Blog" or "E-commerce").
4. **Sanity will initialize the project**: Sanity will create the necessary files and configuration for your project.

Once the project is initialized, you will have access to Sanity Studio to manage content.

1.3 Navigate to Your Project Folder

Navigate to your newly created project folder:

```
cd martnow
```

To start Sanity Studio, run:

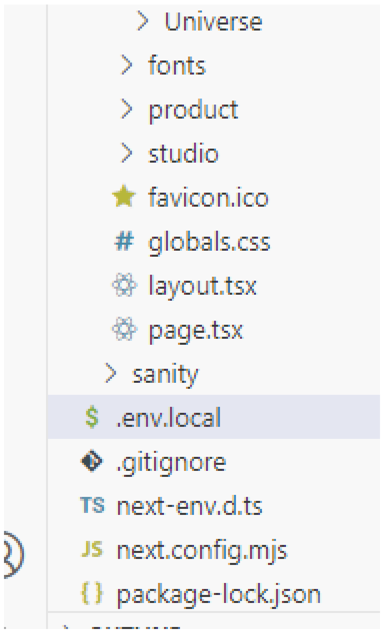
```
npm run dev
```

This will launch a local development environment where you can manage your project content at the following URL: http://localhost:3000/studio

---

2. Configuring Environment Variables

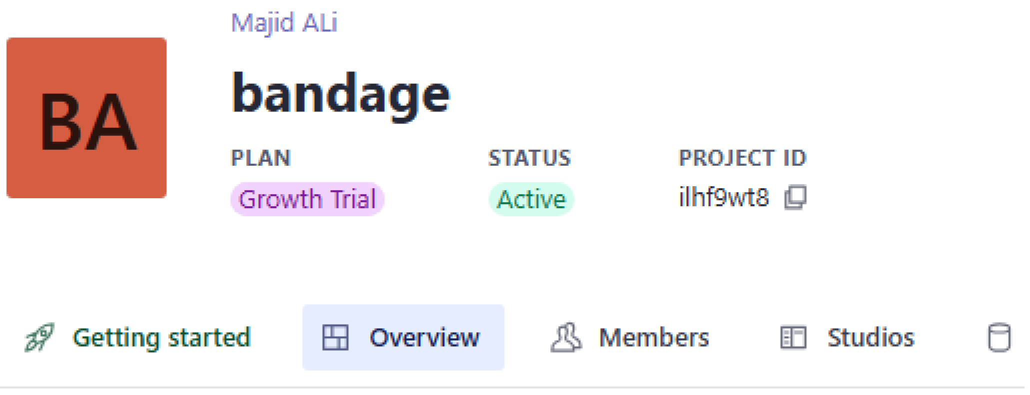
To configure environment variables for your project, follow these steps:



1. **Create a .env.local file** if it does not already exist in the root directory of your project.
2. Add the following variables to the file:
3. NEXT_PUBLIC_SANITY_PROJECT_ID=your_project_id
4. NEXT_PUBLIC_SANITY_DATASET=production
5. SANITY_API_TOKEN=your_sanity_token
6. Replace your_project_id and your_sanity_token with the appropriate values.

2.1 Retrieve the Project ID

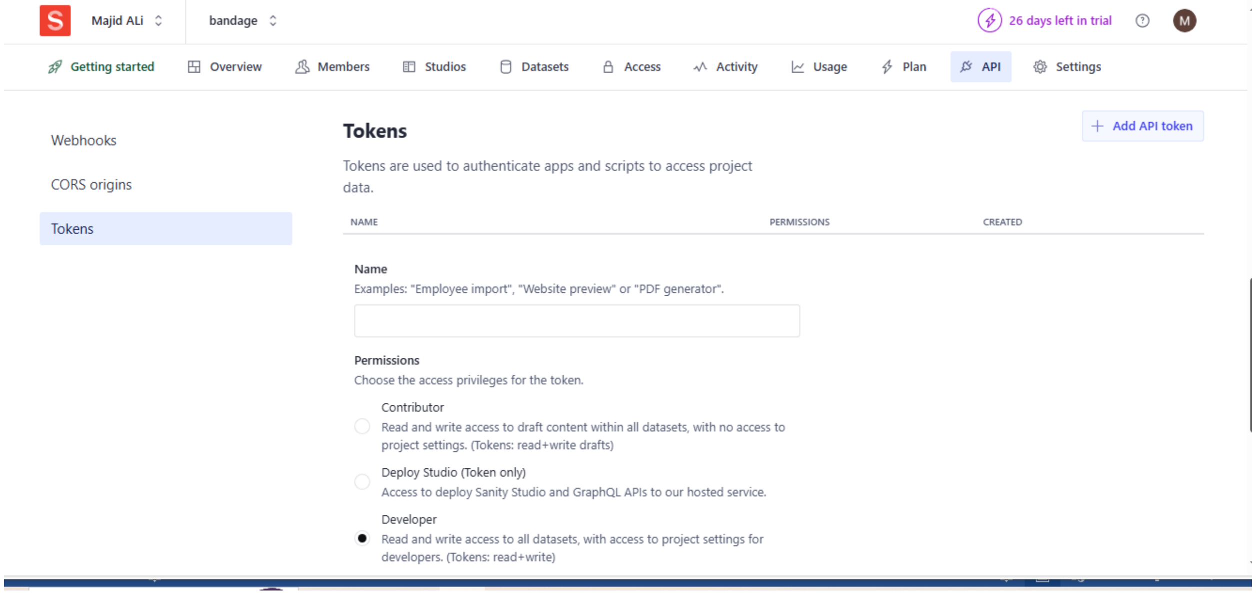
To find your Sanity Project ID:

1. Log in to your Sanity account at https://www.sanity.io/manage.
2. Select the project you want to use.
3. In the Project Dashboard, locate your unique Project ID. Use this ID for the NEXT_PUBLIC_SANITY_PROJECT_ID variable in your .env.local file.
4. 

2.2 Generate an API Token

To generate a Sanity API Token:

1. Go to https://www.sanity.io/manage and select your project.
2. Navigate to the "API" tab within the project settings.
3. Under the "Tokens" section, click **Add API Token**.
4. Name your token (e.g., "Frontend API Token").
5. Select the appropriate permissions (typically "Editor or Developer" for full read/write access).
6. Copy the generated token and paste it into the SANITY_API_TOKEN field in your .env.local file.



Tokens

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

NAME	PERMISSIONS	CREATED

Name

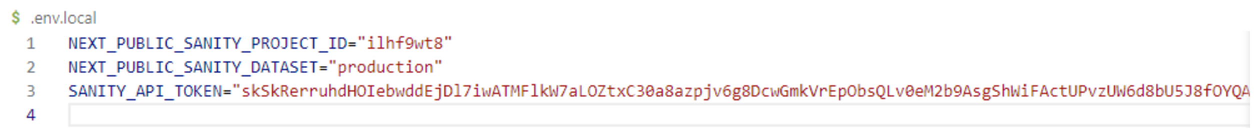
Examples: "Employee import", "Website preview" or "PDF generator".

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)

It look like below in the picture



```
$ .env.local
NEXT_PUBLIC_SANITY_PROJECT_ID="ilhf9wt8"
NEXT_PUBLIC_SANITY_DATASET="production"
SANITY_API_TOKEN="skSkRerruhdHOIebwddEjDl7iwATMFlkW7aLOZtxC30a8azpjv6g8DcwGmkVrEpObsQLv0eM2b9AsgShWiFActUPvzUW6d8bU5J8fOYQA
```

---

3. Creating the Sanity Schema

Creating a schema in Sanity allows you to define content structures, ensuring consistency and control over content storage and querying. Below is the schema for Day 3 of the Hackathon, which will be implemented into our Hackathon Template 5.

3.1 Schema Overview

For Day 3, the schema was provided and will be integrated into Template 5.

- **Template 5** - Sir Muhammad Bilal
- **Template 6 API Docs** - Sir Fahad Khan and Sir Hamza Alvi
- **API URL**: https://template6-six.vercel.app/api/products
- **Schema**: Link to Schema
- **Migration**: Link to Migration

3.2 Implementing the Schema

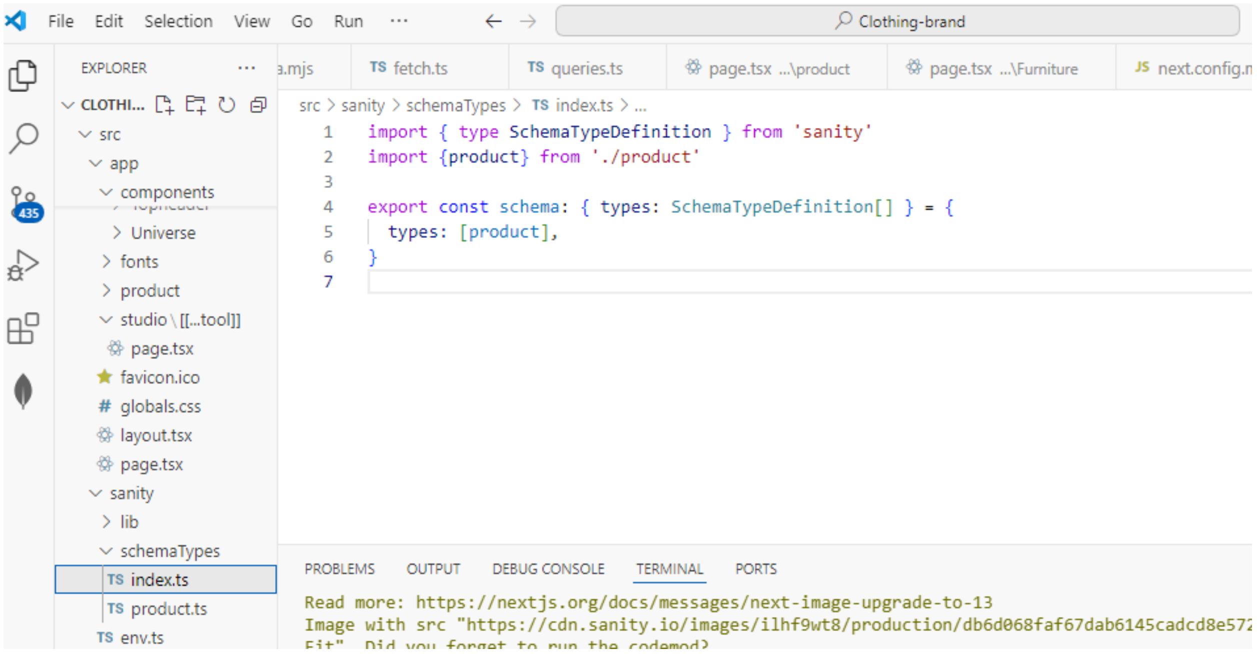
1. **Copy the Schema**:
   Create a new file in schemaType/product.ts and paste the schema as provided:

```
import { defineType } from "sanity";

export const product = defineType({
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    {
      name: "title",
      title: "Title",
      validation: (rule) => rule.required(),
      type: "string"
    },
    {
      name: "description",
      type: "text",
      validation: (rule) => rule.required(),
      title: "Description",
    },
    {
      name: "productImage",
      type: "image",
      validation: (rule) => rule.required(),
      title: "Product Image"
    },
    {
      name: "price",
      type: "number",
      validation: (rule) => rule.required(),
      title: "Price",
    },
    {
      name: "tags",
      type: "array",
      title: "Tags",
      of: [{ type: "string" }]
    },
    {
      name: "discountPercentage",
      type: "number",
      title: "Discount Percentage",
    },
    {
      name: "isNew",
      type: "boolean",
      title: "New Badge",
    }
  ]
});
```

Update index.ts:

Modify your sanity/schemaTypes/index.ts file to include the new product schema:

types: [product],



```
import { type SchemaTypeDefinition } from 'sanity'
import {product} from './product'

export const schema: { types: SchemaTypeDefinition[] } = {
  types: [product],
}
```

```
Read more: https://nextjs.org/docs/messages/next-image-upgrade-to-13
Image with src "https://cdn.sanity.io/images/ilhf9wt8/production/db6d068faf67dab6145cadcd8e572
Fit". Did you forget to run the codemod?
```

4. Data Migration Script

In order to import data from an external API into Sanity, follow these steps:

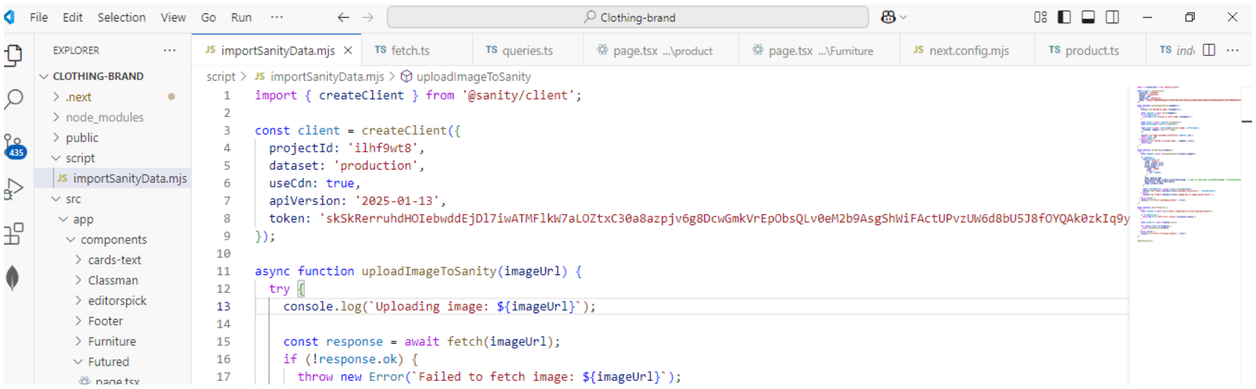
4.1 Create a Script File

1. Create a folder named script in the root directory of your project.
2. Create a file named importSanityData.mjs within the script folder.

4.2 Import External API Data

Use the following migration script to import data from the external API into Sanity.

- **Migration Script**: Link to Script



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

const client = createClient({
  projectId: 'ilhf9wt8',
  dataset: 'production',
  useCdn: true,
  apiVersion: '2025-01-13',
  token: 'skSkRerruhdHOIebwddEjDl7iwATMFlkW7aLOZtxC30a8azpjv6g8DcwGmkVrEpObsQLv0eM2b9AsgShWiFActUPvzUW6d8bU5J8fOYQAk0zkIq9y
});

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}`);
```

4.3 Install Dependencies

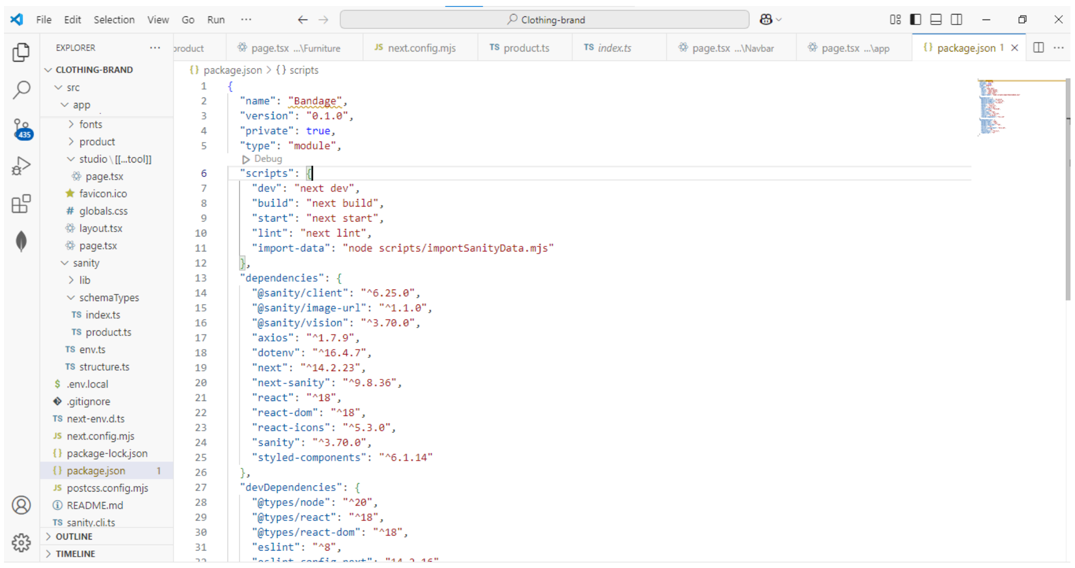
Run the following command in the terminal to install necessary dependencies:

```
npm install @sanity/client axios dotenv
```

4.4 Configure package.json

To run the migration script, add the following to your package.json file under the "scripts" section:

01- "type": "module",
02- "scripts": {"import-data": "node scripts/importSanityData.mjs"}



```json
{
  "name": "Bandage",
  "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.mjs"
  },
  "dependencies": {
    "@sanity/client": "^6.25.0",
    "@sanity/image-url": "^1.1.0",
    "@sanity/vision": "^3.70.0",
    "axios": "^1.7.9",
    "dotenv": "^16.4.7",
    "next": "^14.2.23",
    "next-sanity": "^9.8.36",
    "react": "^18",
    "react-dom": "^18",
    "react-icons": "^5.3.0",
    "sanity": "^3.70.0",
    "styled-components": "^6.1.14"
  },
  "devDependencies": {
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "eslint": "^8",
```

4.5 Run the Import Script

Execute the migration script with the following command:

```
npm run import-data
```

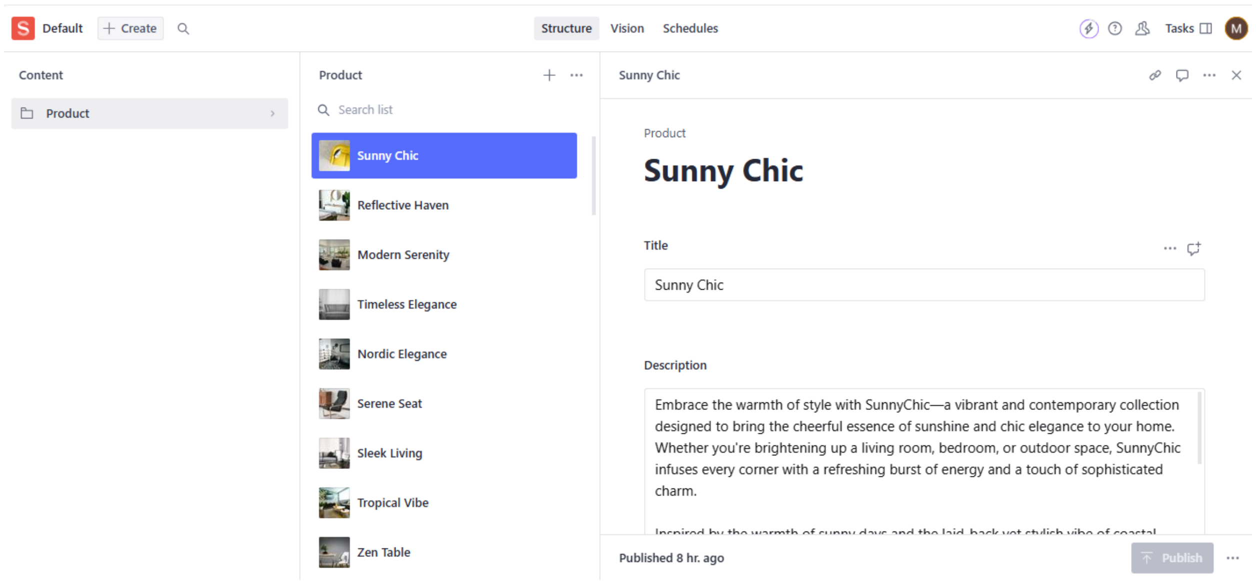
This will fetch product data from the provided API, upload associated images to Sanity's asset store, and create new product documents in your Sanity dataset.

To Check the date import in sanity you can run the command.

```
npm run dev
```

It will open the local host 3000 . http://localhost:3000.

Open /studio in to the local host. It looks like and it show the imported data in to the sanity.



Sunny Chic

Title

Sunny Chic

Description

Embrace the warmth of style with SunnyChic—a vibrant and contemporary collection designed to bring the cheerful essence of sunshine and chic elegance to your home. Whether you're brightening up a living room, bedroom, or outdoor space, SunnyChic infuses every corner with a refreshing burst of energy and a touch of sophisticated charm.

Steps to Fetch and Display Sanity Data on the Frontend

1. Query for Products in Sanity Studio

To fetch product data from your Sanity CMS, use the following GROQ query. This query selects all products and retrieves the necessary fields for each product.

Write the query ;

```
 *[_type == "product"]{
  _id,
  title,
  productImage {
    asset->{
      url
    }
  },
  price,
  tags,
  discountPercentage,
  isNew
}
```

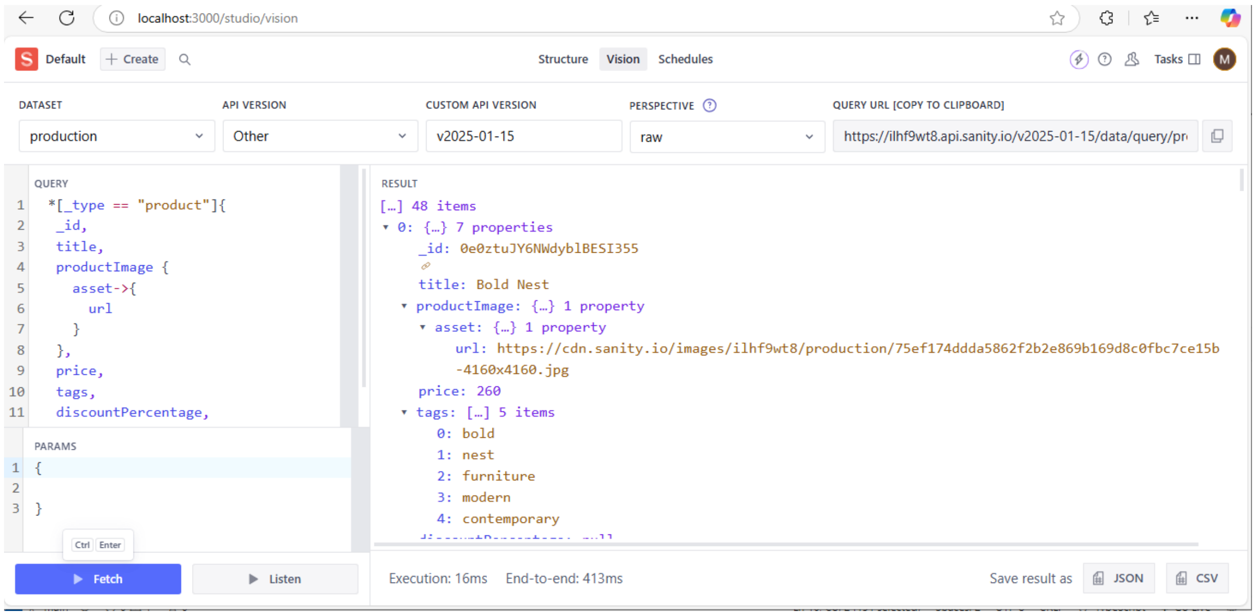
1. ***[_type == "product"]:**

   - The asterisk (*) is used to select all documents that match the condition inside the square brackets ([]).
   - [_type == "product"] filters the documents, retrieving only those whose _type field is equal to "product". In this case, it fetches all documents of the type product from the Sanity dataset.
2. **{ ... }**:
   - Inside the curly braces {}, we specify which fields to retrieve from each product document.
3. **Field Definitions**:
   - **_id**: The unique identifier of each product document.
   - **title**: The title or name of the product.
   - **productImage**: A reference field pointing to the product's image.
     - **asset-> { url }**: This uses the dereferencing operator (->) to fetch the asset associated with the image and retrieve its url field, which provides the URL of the image.
   - **price**: The price of the product.
   - **tags**: An array or list of tags associated with the product. Tags are typically used for categorization or filtering.
   - **discountPercentage**: A field representing the percentage discount applied to the product, if any.
   - **isNew**: A boolean field indicating whether the product is marked as "new."

Purpose:

This query is designed to retrieve the following information for every product in the dataset:

- Product ID (_id)
- Product Title (title)
- Product Image URL (productImage.asset.url)
- Product Price (price)
- Tags associated with the product (tags)
- Discount Percentage (discountPercentage)
- Whether the product is marked as new (isNew)



DATASET: production
API VERSION: Other
CUSTOM API VERSION: v2025-01-15
PERSPECTIVE: raw
QUERY URL [COPY TO CLIPBOARD]: https://ilhf9wt8.api.sanity.io/v2025-01-15/data/query/pr

QUERY
```
*[_type == "product"]{
  _id,
  title,
  productImage {
    asset->{
      url
    }
  },
  price,
  tags,
  discountPercentage,
```

PARAMS
```
{

}
```

RESULT
```
[…] 48 items
▾ 0: {…} 7 properties
    _id: 0e0ztuJY6NWdyblBESI355
    title: Bold Nest
  ▾ productImage: {…} 1 property
    ▾ asset: {…} 1 property
        url: https://cdn.sanity.io/images/ilhf9wt8/production/75ef174ddda5862f2b2e869b169d8c0fbc7ce15b
        -4160x4160.jpg
    price: 260
  ▾ tags: […] 5 items
      0: bold
      1: nest
      2: furniture
      3: modern
      4: contemporary
```

Execution: 16ms End-to-end: 413ms

.Now we have to fetch the sanity asset store data in to frontend.

.Open project in vs code . in the sanity folder there is a lib folder make a file name fetch.ts

Write the following code in to the fetch file.

```
import { createClient } from "next-sanity";

const client = createClient({
   projectId : "ilhf9wt8",
   dataset : "production",
   useCdn : true,
   apiVersion: '2025-01-13',

});

export async function sanityFetch({query, params = {}}: {query : string, params? : any}) {
   return await client.fetch(query, params)
}
```
This imports the createClient function from the next-sanity package, which is a library designed to integrate Sanity CMS with Next.js applications.

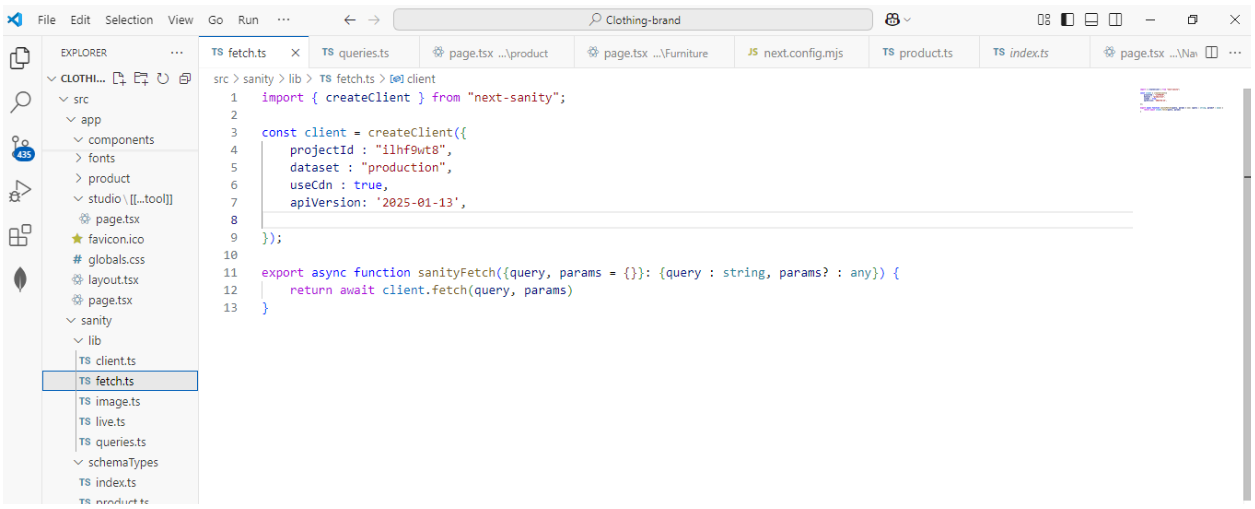
createClient: This function creates an instance of a Sanity client that allows you to communicate with your Sanity project.

- **projectId**: The unique ID of your Sanity project (e.g., "ilhf9wt8").

- **dataset**: The name of the dataset in your project. In this case, it's set to "production", which refers to the production environment.
- **useCdn**: When set to true, this tells Sanity to use the Content Delivery Network (CDN) for fetching content, optimizing for faster reads at the cost of potentially slightly outdated data (though typically updated within seconds).
- **apiVersion**: Specifies the version of the Sanity API to use. In this case, it's set to '2025-01-13', ensuring compatibility with that specific API version.

sanityFetch: This is an asynchronous function that fetches data from Sanity using a specified query and optional parameters.

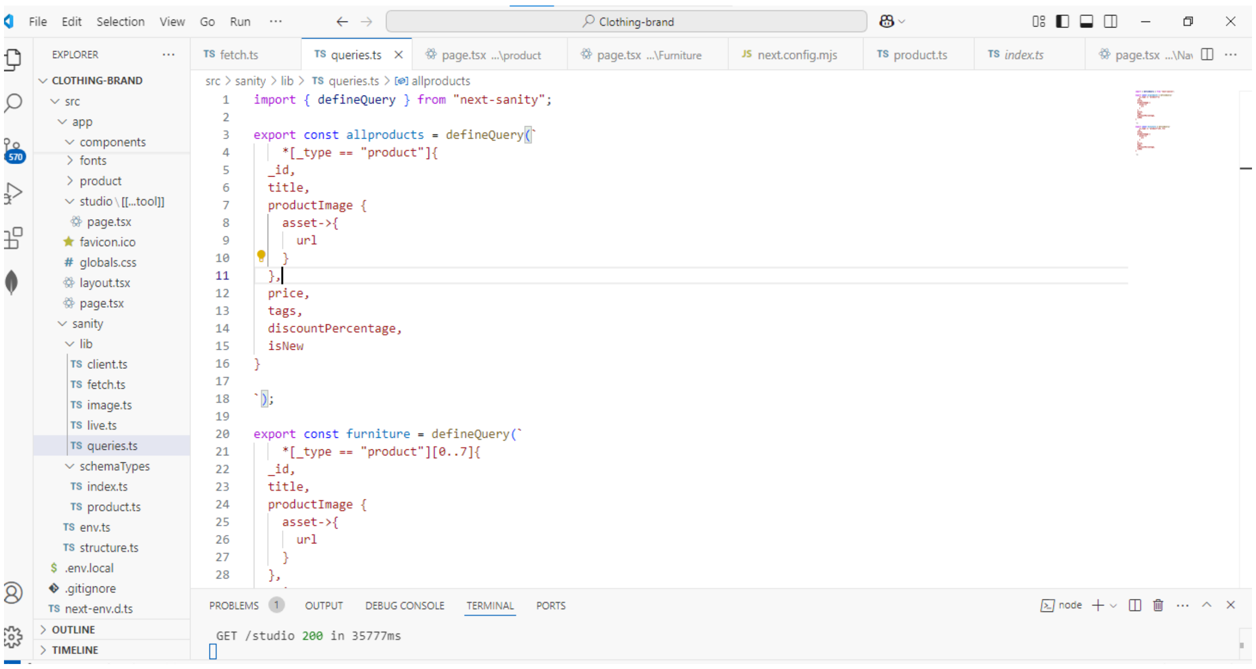
- **query**: A string representing the GROQ (Graph-Relational Object Queries) query that will be executed against the Sanity dataset. GROQ is the query language used by Sanity to retrieve content.
- **params**: Optional parameters that can be passed to the query. These can be used to filter or modify the data returned by the query.
- The function returns the result of the client.fetch() method, which is an asynchronous operation that fetches data from Sanity based on the provided query and parameters.



```
import { createClient } from "next-sanity";

const client = createClient({
    projectId : "ilhf9wt8",
    dataset : "production",
    useCdn : true,
    apiVersion: '2025-01-13',

});

export async function sanityFetch({query, params = {}}: {query : string, params? : any}) {
    return await client.fetch(query, params)
}
```

Now the second step is to open new file in lib in the name of queries.ts . copy the same queries which we have written in the vision in sanity studio.

```
import { defineQuery } from "next-sanity";

export const allproducts = defineQuery(`
   *[_type == "product"]{
  _id,
  title,
  productImage {
    asset->{
      url
    }
  },
  price,
  tags,
  discountPercentage,
  isNew
}
```



```
import { defineQuery } from "next-sanity";

export const allproducts = defineQuery(`
  *[_type == "product"]{
  _id,
  title,
  productImage {
    asset->{
      url
    }
  },
  price,
  tags,
  discountPercentage,
  isNew
}

`);

export const furniture = defineQuery(`
  *[_type == "product"][0..7]{
  _id,
  title,
  productImage {
    asset->{
      url
    }
  },
```

```
GET /studio 200 in 35777ms
```

Now all set to fetch Data in to frontend. But where to fetch it. We have open a file in src/app/product/page.tsx

```
import { sanityFetch } from "@/sanity/lib/fetch";
import { allproducts } from "@/sanity/lib/queries";
import Image from "next/image"; // Make sure to import the Image component
import Topheader from "../components/Topheader/page";
import Navbar from "../components/Navbar/page";
import Footer from "../components/Footer/page";


// Define the updated Product type
type Product = {
  _id: string;
  title: string;
  description: string;
  price: number;
```

```
    productImage: {
        asset: {
            url: string;
        };
    };
    tags: string[];
    discountPercentage: number;
    isNew: boolean;
};

export default async function Product() {
    const products: Product[] = await sanityFetch({ query: allproducts });

    return (
        <div>
            <Topheader />
            <Navbar />
            <h1 className="text-center text-3xl font-bold my-8">Products</h1>
            {/* Use responsive grid layout */}
            <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4">
                {products.map((product) => (
                    <div
                        className="border p-4 rounded-lg shadow-sm flex flex-col items-center"
                        key={product._id}
                    >
                        {/* Image container with relative position and fixed height */}
                        <div className="relative w-full h-64"> {/* Adjust height here if needed */}
                            <Image
                                src={product.productImage.asset.url}
                                alt={product.title}
                                layout="fill"
                                objectFit="cover"
                            />
                        </div>
                        <h2 className="text-xl font-bold text-center mt-4">{product.title}</h2> {/* Product title */}
                        <p className="text-center">{product.description}</p> {/* Product description */}
                        <p className="text-center text-lg font-semibold">${product.price}</p> {/* Price */}
                        <p className="text-center text-gray-500">{product.isNew ? 'New' : ''}</p> {/* New label */}
                        <p className="text-center text-sm text-gray-400">{product.discountPercentage}50 % Off</p>
{/* Discount */}
                        <p className="text-center text-sm text-gray-400">{product.tags.join(", ")}</p> {/* Tags */}
                    </div>

                ))}
            </div>
        <Footer />
      </div>
  );
}
```

This fetches and displays a list of products in a grid format. It uses the Sanity CMS for data fetching and includes UI components /page. Following are the details of code.

Imports

- **sanityFetch**: A custom function for fetching data from a Sanity CMS.
- **allproducts**: A query to fetch all the products from Sanity.
- **Image**: Next.js component for optimized image rendering.
- **Topheader, Navbar, Footer**: Custom components for page layout.

TypeScript Type (Product)

- The Product type defines the shape of a product object, which includes:
  - _id: Unique identifier for the product.
  - title: Product title.
  - description: Short description of the product.
  - price: Price of the product.
  - productImage: Object containing the URL of the product image.
  - tags: Array of tags related to the product.
  - discountPercentage: Percentage of discount applied to the product.
  - isNew: Boolean indicating if the product is new.

Product Component /Page

- **sanityFetch**: It fetches product data using the allproducts query.
- **Grid Layout**: Products are displayed in a responsive grid layout, where the number of columns changes based on screen size (1 column on small screens, 2 on medium, 3 on larger, and 4 on extra-large).
- **Product Card**: Each product is displayed in a div with:
  - **Image**: Rendered using the Next.js Image component, with layout="fill" for responsive sizing and objectFit="cover" to maintain aspect ratio.
  - **Product Title**: Displayed below the image.
  - **Description**: Brief product description.
  - **Price**: Displayed in a bold, larger font.
  - **New Label**: Displays "New" if the product is marked as new.
  - **Discount**: Displays the discount percentage (with a static "50%" appended).
  - **Tags**: Displays tags for the product, joined by commas.
- **Footer**: The Footer component is included at the bottom.

Layout and Styling

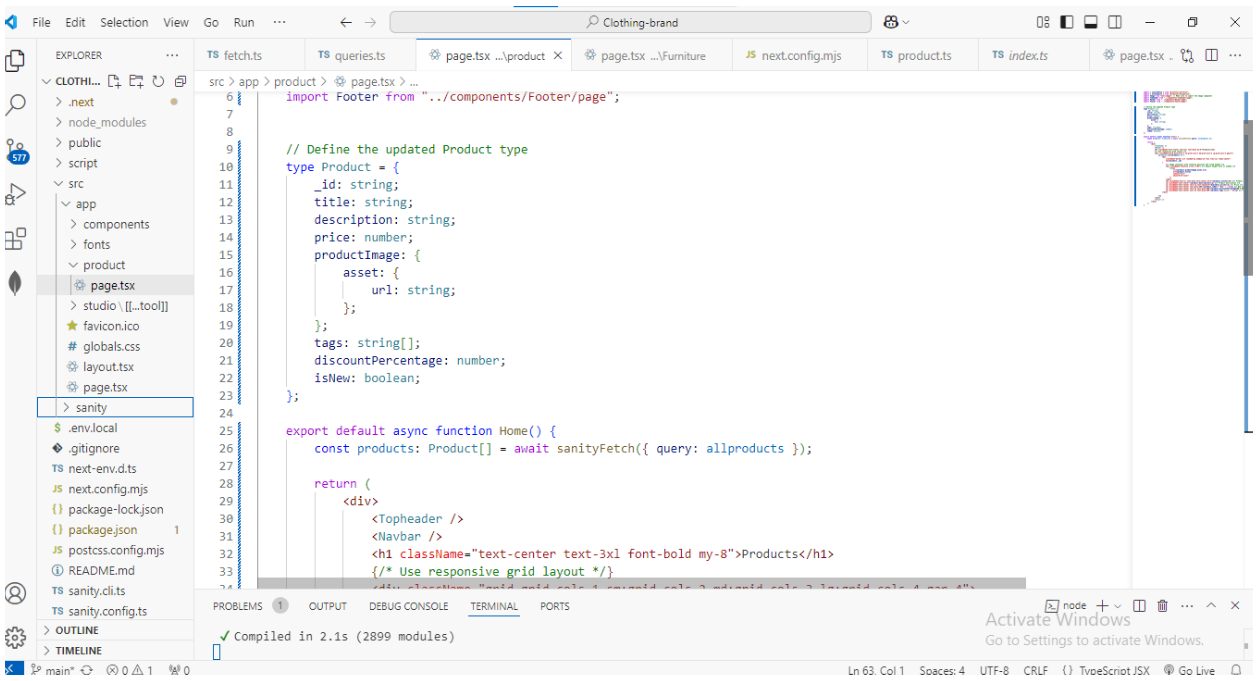
- The grid layout uses Tailwind CSS classes, ensuring that the product cards are responsive.
- Each product is styled with padding, borders, and shadow effects, making it visually distinct.

Overall Flow

1. The Home component fetches products from Sanity.
2. The fetched products are then displayed in a grid layout.
3. Each product has its image, title, description, price, discount, tags, and a "New" label (if applicable).
4. The layout is wrapped with header and footer components (Topheader, Navbar, Footer).

Potential Use Case

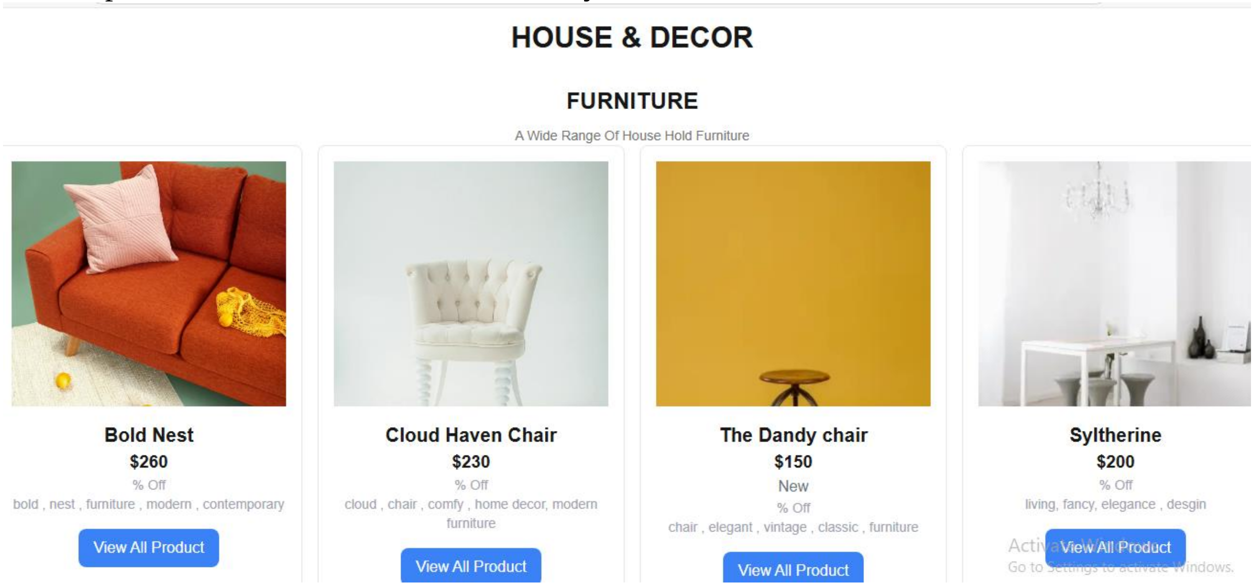
This component could be used for an e-commerce homepage or a product listing page, where products are dynamically fetched from a backend (Sanity CMS in this case) and displayed in a responsive, visually appealing grid.



```tsx
import Footer from "../components/Footer/page";


// Define the updated Product type
type Product = {
    _id: string;
    title: string;
    description: string;
    price: number;
    productImage: {
        asset: {
            url: string;
        };
    };
    tags: string[];
    discountPercentage: number;
    isNew: boolean;
};

export default async function Home() {
    const products: Product[] = await sanityFetch({ query: allproducts });

    return (
        <div>
            <Topheader />
            <Navbar />
            <h1 className="text-center text-3xl font-bold my-8">Products</h1>
            {/* Use responsive grid layout */}
```

```
✓ Compiled in 2.1s (2899 modules)
```

That is the final step to fetch data. Now npm run dev and check the previo on local server. Fetch the completed data in to front end successfully.



HOUSE & DECOR

FURNITURE

A Wide Range Of House Hold Furniture

Bold Nest
$260
% Off
bold , nest , furniture , modern , contemporary
View All Product

Cloud Haven Chair
$230
% Off
cloud , chair , comfy , home decor, modern furniture
View All Product

The Dandy chair
$150
New
% Off
chair , elegant , vintage , classic , furniture
View All Product

Syltherine
$200
% Off
living, fancy, elegance , desgin
View All Product

Vercel Link: [Bandage]()

GitHub Link: [Majidali80/Clothing-brand: Clothing brand]()