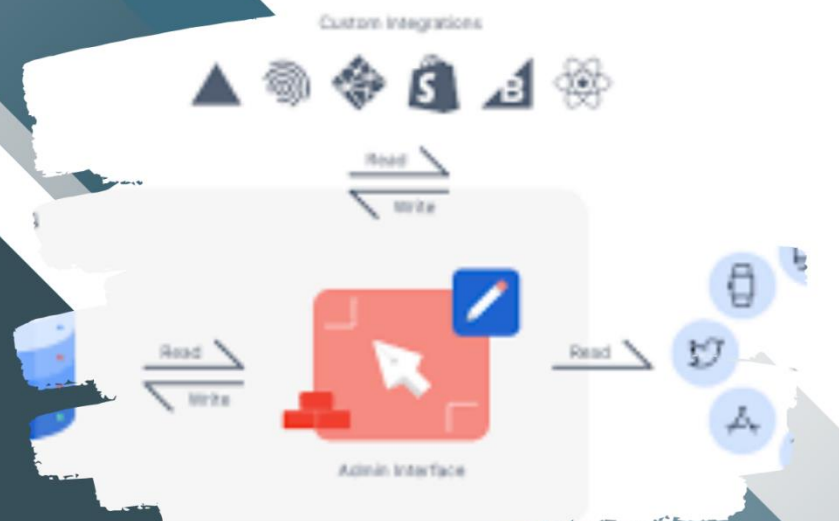


PREPARED BY
Muhammad Mubashir Saeedi

HACKATHON

DAY 3



API INTEGRATION MIGRATION REPORT BANDAGE

Day 3 – API Integration and Data Migration Report – Bandage

Objective:

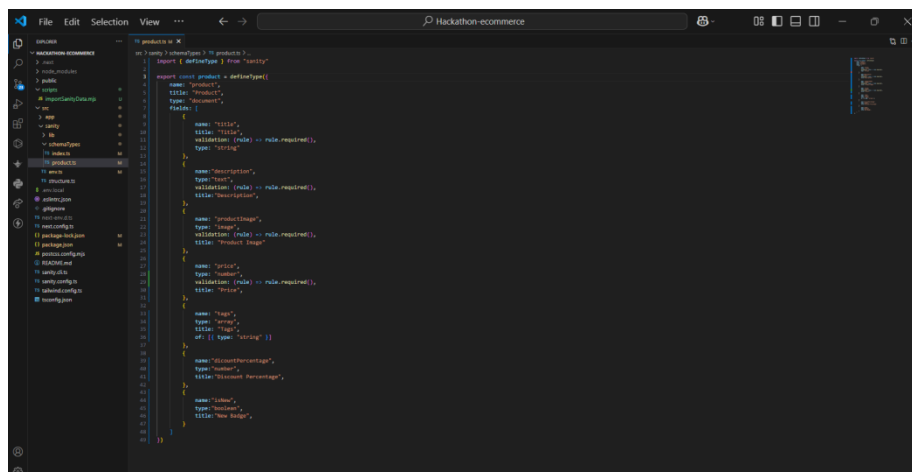
The goal for Day 3 was to integrate API data into Sanity CMS for the Bandage project, enabling dynamic content updates for the marketplace. Instead of manually entering data, the API integration provided a more efficient and scalable solution.

1. Sanity CMS Schema Design:

To ensure the seamless handling of product data, I designed a schema called `product` in Sanity CMS. The schema includes the following fields:

- **Main Fields:**
 - `title`: The product title (string type).
 - `description`: A detailed description of the product (text type).
 - `productImage`: The main product image (image type).
 - `price`: The price of the product (number type).
 - `tags`: An array of tags to categorize the product (array of strings).
 - `discountPercentage`: The discount percentage (number type).
 - `isNew`: A boolean flag indicating if the product is new (boolean type).

Code Snippet:

A screenshot of a code editor window titled 'Hackathon-e-commerce'. The editor shows a Sanity CMS schema for a 'product' type. The schema is written in JavaScript and includes fields for title, description, productImage, price, tags, discountPercentage, and isNew. The fields are defined with their respective types and validation rules. The 'tags' field is an array of strings, and 'discountPercentage' is a number. The 'isNew' field is a boolean. The schema is exported as a function that returns the product type definition.

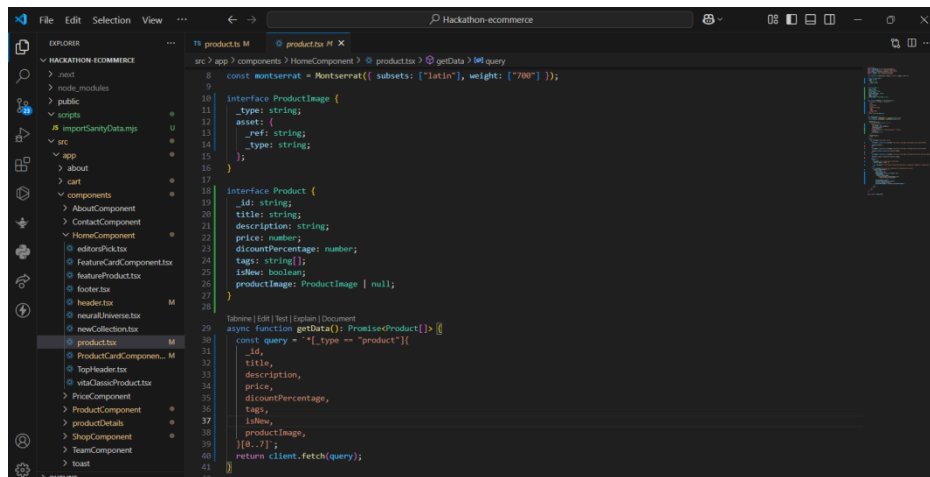
```
import { defineType } from 'sanity'

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

2. API Integration and Data Migration:

API Data Fetching:

I fetched product data from an external API, which included details like images, titles, descriptions, prices, and tags. This data was then mapped to the corresponding fields in the Sanity CMS schema.



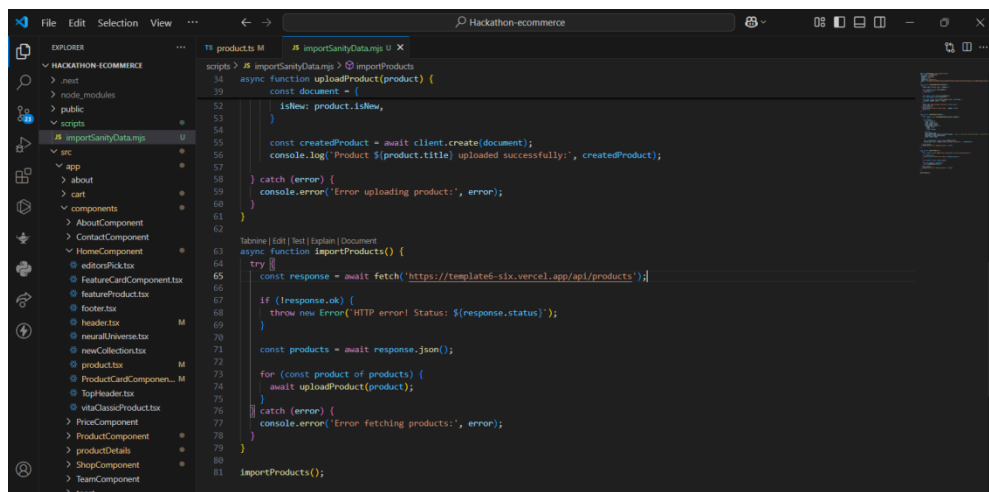
```
src > app > components > HomeComponent > product.js > fetchData > fetchData
1  const montserrat = Montserrat({ subsets: ['latin'], weight: ['700'] });
2
3  interface ProductImage {
4    _type: string;
5    asset: {
6      _ref: string;
7      _type: string;
8    };
9  }
10
11  interface Product {
12    _id: string;
13    title: string;
14    description: string;
15    price: number;
16    discountPercentage: number;
17    tags: string[];
18    isNew: boolean;
19    productImage: ProductImage | null;
20  }
21
22  export async function fetchData(): Promise<Product[]> {
23    const query = '*[_type == "product"]';
24
25    const response = await client.fetch(query);
26    return response || [];
27  }
28
29  export default fetchData;
```

Data Population in Sanity CMS:

After fetching the API data, I populated the product fields in Sanity CMS dynamically. This allowed for the automated population of product information, ensuring consistency and accuracy across the platform.

Data Migration:

Using the Sanity CLI, I exported the dataset from Sanity CMS for backup purposes and later re-imported it for testing. This migration ensured that all data was properly structured and displayed as intended on the frontend.

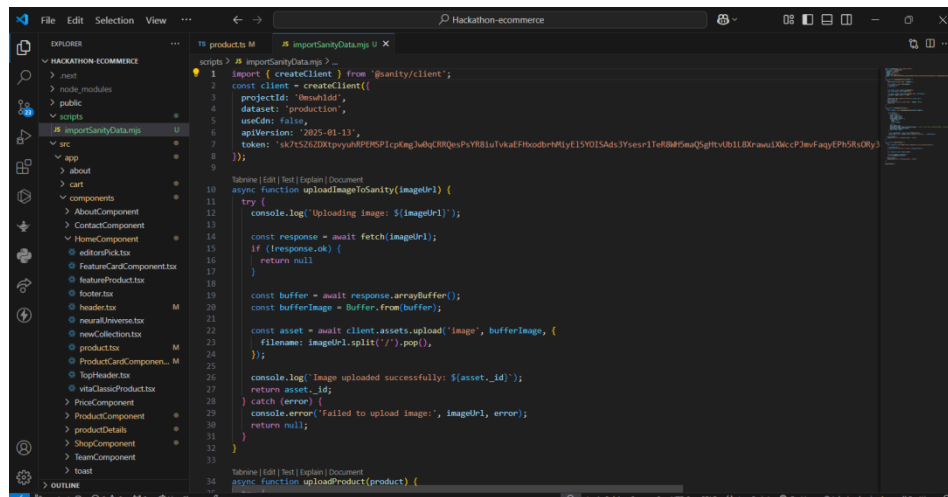


```
src > app > components > HomeComponent > importProducts.js > importProducts
1  export async function importProducts() {
2    const response = await fetch('https://template6-six.vercel.app/api/products');
3    if (!response.ok) {
4      throw new Error('HTTP error! Status: ' + response.status);
5    }
6    const products = await response.json();
7
8    for (const product of products) {
9      await uploadProduct(product);
10    }
11  }
12
13  export default importProducts;
```

3. Steps Taken for Data Migration:

Exporting Data:

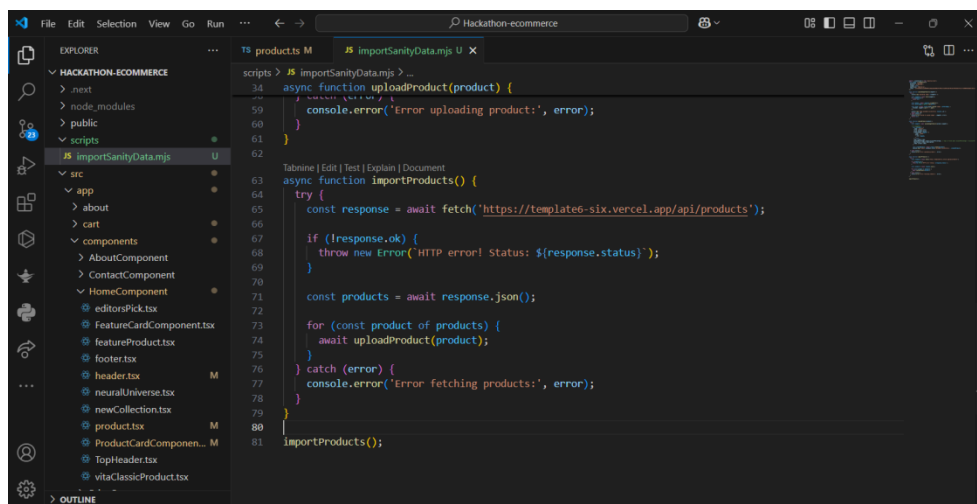
The first step was exporting the data from Sanity CMS using the Sanity CLI. This ensured that all product data was safely backed up before any further operations.



```
1  #!/usr/bin/env node
2  import { createClient } from '@sanity/client';
3  const client = createClient({
4    projectId: '0mshldd',
5    dataset: 'production',
6    useCdn: false,
7    apiVersion: '2025-01-13',
8    token: 'sk71525Z0t7pyuHfP5P1cpkgJubqCRQesPaY8liuVkaEFHodbrHmlyE15Y0IS4ds3Yses1TefBMfGuaQgntvUblL8xrawLXiccP7wFayEPH5ts0Ry3';
9  });
10
11  async function uploadImageToSanity(imageUrl) {
12    try {
13      console.log('Uploading image: ${imageUrl}');
14      const response = await fetch(imageUrl);
15      if (!response.ok) {
16        return null;
17      }
18      const buffer = await response.arrayBuffer();
19      const bufferImage = Buffer.from(buffer);
20      const asset = await client.assets.upload('image', bufferImage, {
21        filename: imageUrl.split('/').pop(),
22      });
23      console.log('Image uploaded successfully: ${asset.id}');
24      return asset._id;
25    } catch (error) {
26      console.error('Failed to upload image:', imageUrl, error);
27      return null;
28    }
29  }
30
31  async function uploadProduct(product) {
32    // ... (rest of the code) ...
33  }
34
35  // ... (rest of the code) ...
36  }
```

Verification of Data:

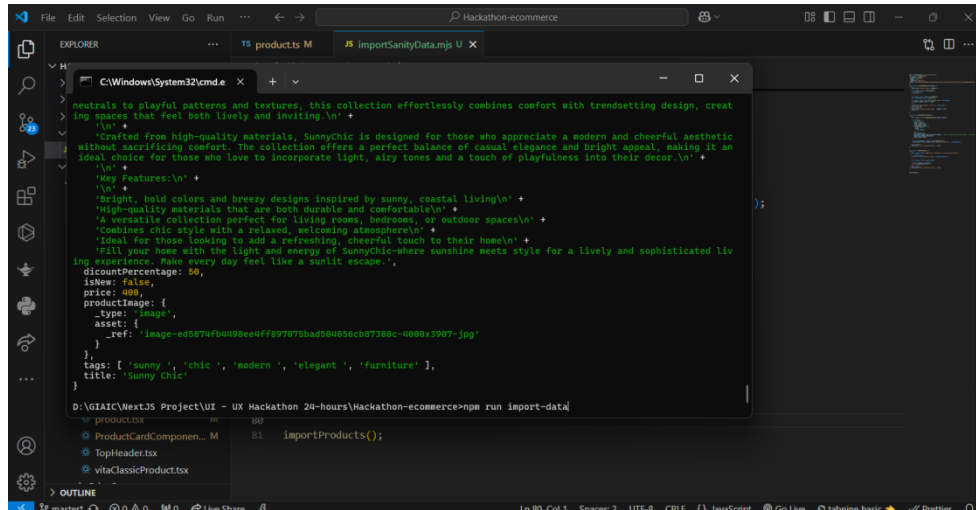
The exported JSON structure was reviewed to ensure that all fields were populated correctly. This step ensured that the data would be accurately displayed when fetched and rendered on the frontend.



```
1  #!/usr/bin/env node
2  import { createClient } from '@sanity/client';
3  const client = createClient({
4    projectId: '0mshldd',
5    dataset: 'production',
6    useCdn: false,
7    apiVersion: '2025-01-13',
8    token: 'sk71525Z0t7pyuHfP5P1cpkgJubqCRQesPaY8liuVkaEFHodbrHmlyE15Y0IS4ds3Yses1TefBMfGuaQgntvUblL8xrawLXiccP7wFayEPH5ts0Ry3';
9  });
10
11  async function uploadImageToSanity(imageUrl) {
12    try {
13      console.log('Uploading image: ${imageUrl}');
14      const response = await fetch(imageUrl);
15      if (!response.ok) {
16        return null;
17      }
18      const buffer = await response.arrayBuffer();
19      const bufferImage = Buffer.from(buffer);
20      const asset = await client.assets.upload('image', bufferImage, {
21        filename: imageUrl.split('/').pop(),
22      });
23      console.log('Image uploaded successfully: ${asset.id}');
24      return asset._id;
25    } catch (error) {
26      console.error('Failed to upload image:', imageUrl, error);
27      return null;
28    }
29  }
30
31  async function uploadProduct(product) {
32    // ... (rest of the code) ...
33  }
34
35  // ... (rest of the code) ...
36  }
```

Re-importing Data:

After verification, the dataset was re-imported into Sanity CMS. This confirmed that the data migration was successful and the system was working as expected.

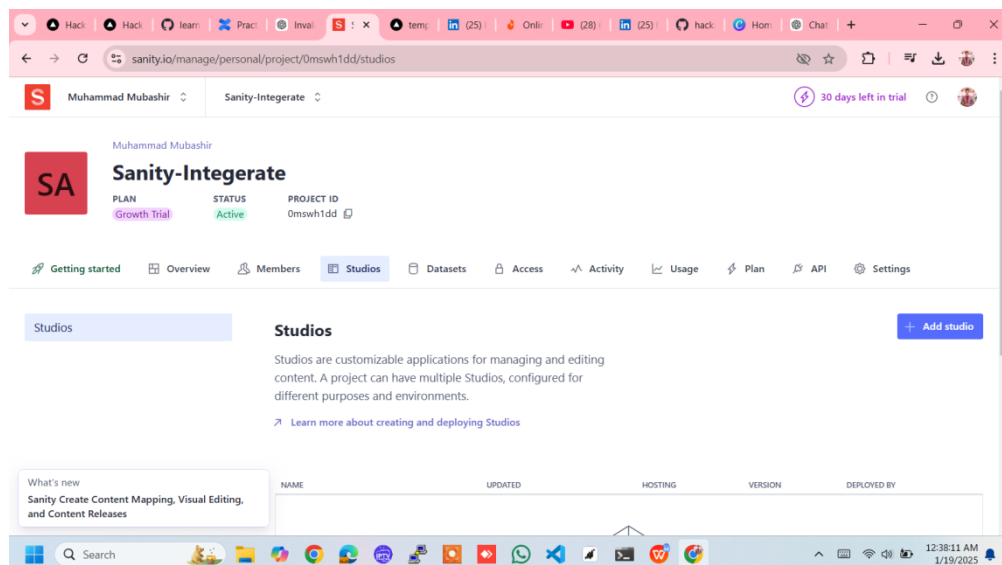


```
File Edit Selection View Go Run ... Hackathon-e-commerce
EXPLORER
  TS products.ts M
  JS importSanityData.mjs U
C:\Windows\System32\cmd.exe
neutrals to playful patterns and textures, this collection effortlessly combines comfort with trendsetting design, creating spaces that feel both lively and inviting.\n' +
'\n' +
'Crafted from high-quality materials, SunnyChic is designed for those who appreciate a modern and cheerful aesthetic without sacrificing comfort. The collection offers a perfect balance of casual elegance and bright appeal, making it an ideal choice for those who love to incorporate light, airy tones and a touch of playfulness into their decor.\n' +
'\n' +
'Key Features:\n' +
'\n' +
'Delight in bold colors and breezy designs inspired by sunny, coastal living\n' +
'High-quality materials that are both durable and comfortable\n' +
'A versatile collection perfect for living rooms, bedrooms, or outdoor spaces\n' +
'Combines chic style with a relaxed, welcoming atmosphere\n' +
'Ideal for those looking to add a refreshing, cheerful touch to their home\n' +
'Fill your home with the light and energy of SunnyChic-where sunshine meets style for a lively and sophisticated living experience. Make every day feel like a sunlit escape.'
discountPercentage: 50,
isNew: false,
price: 400,
productImage: {
  _type: 'image',
  asset: {
    _ref: 'image-ed587afba098ee0ff897873bad580856cb87388c-0080x3907-jpg'
  }
},
tags: ['sunny', 'chic', 'modern', 'elegant', 'furniture'],
title: 'Sunny Chic'
D:\GIAIC\NextJS Project\UI - UX Hackathon 24-hours\Hackathon-e-commerce>npm run import-data
npm error Missing script: "import-data"
npm error
npm error To see a list of scripts, run: npm run -s
npm error
npm error If you want to run a custom command, please turn on the `--script-shell` flag.
npm error
npm error A complete log of this run can be found in: C:\Users\mubashir\AppData\Local\npm-cache\_logs\2025-01-19T08:11:11.000Z-debug-0.log
In 20 Ctrl+1 Sources 2 LINT-8 CRLF 1 LineCount 0 Go Live - Debugging Basics - Docker
```

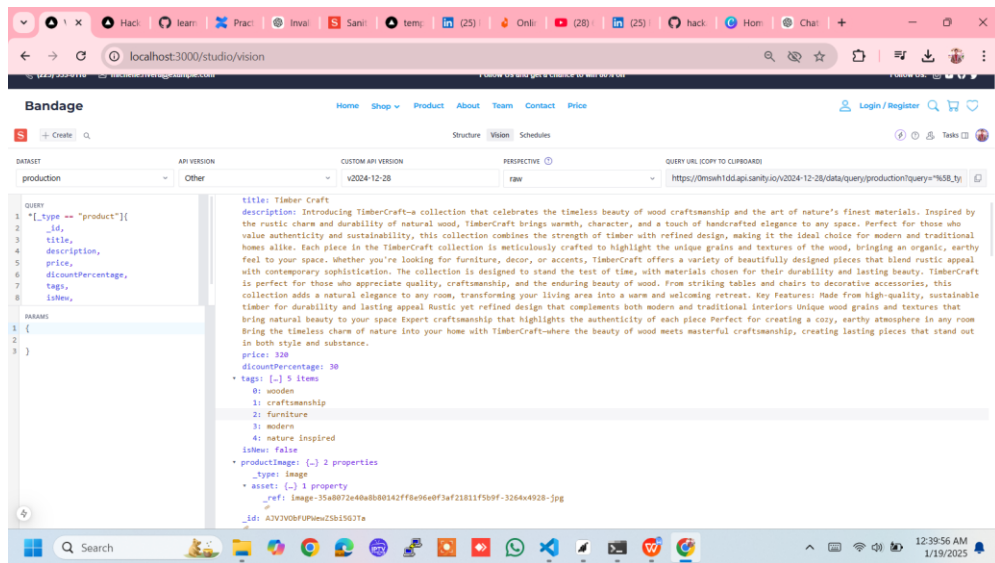
4. Tools Used:

Sanity Studio:

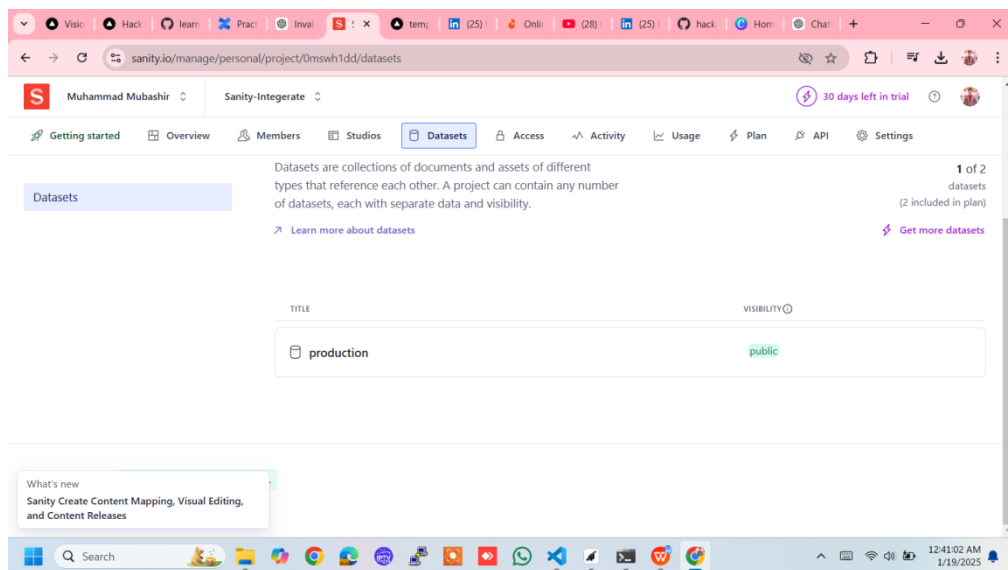
Used for schema creation, content management, and displaying product data.



Sanity Vision



Sanity Database:



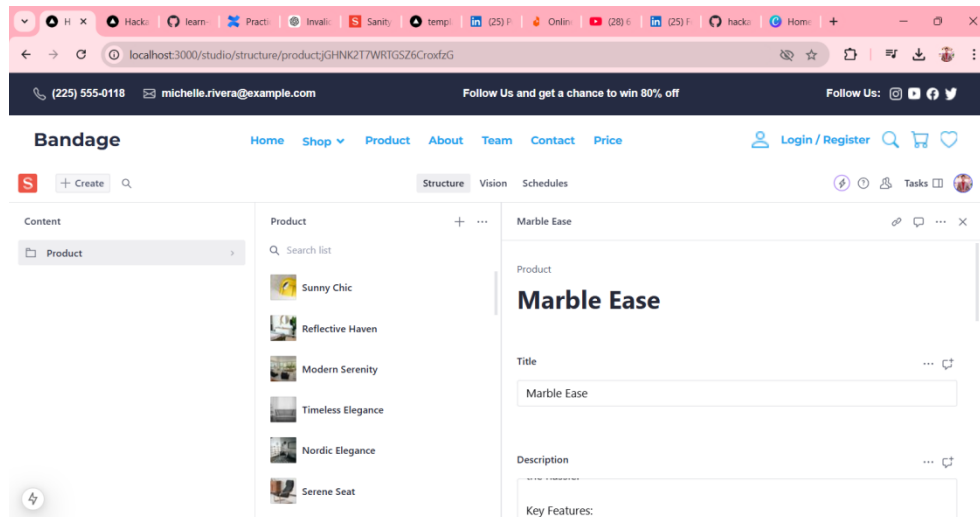
Sanity CLI:

Utilized for exporting and importing the dataset, ensuring data consistency and backup.

5. Screenshots and Frontend Display:

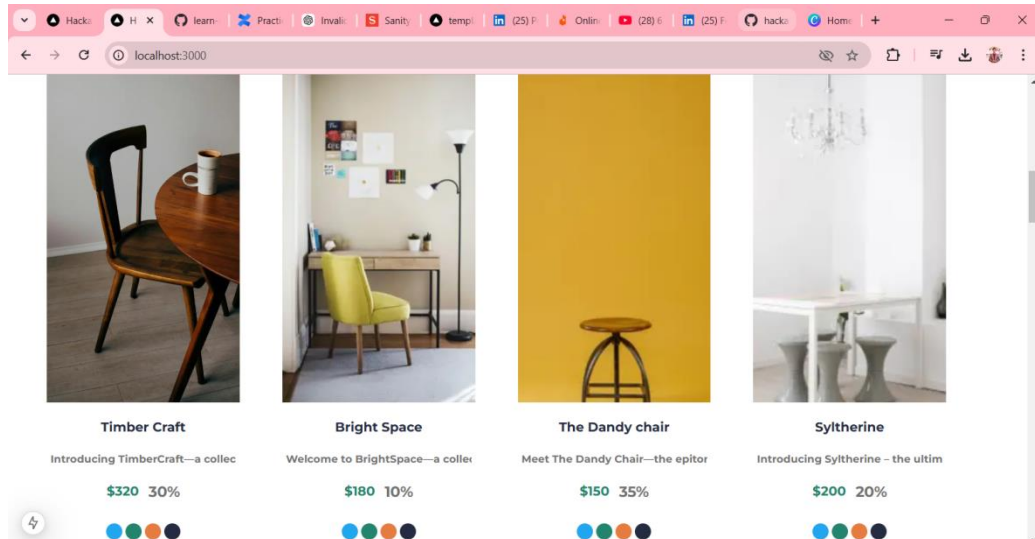
Sanity CMS Fields:

Screenshot showing the populated fields in Sanity Studio, displaying product details like images, descriptions, and prices.

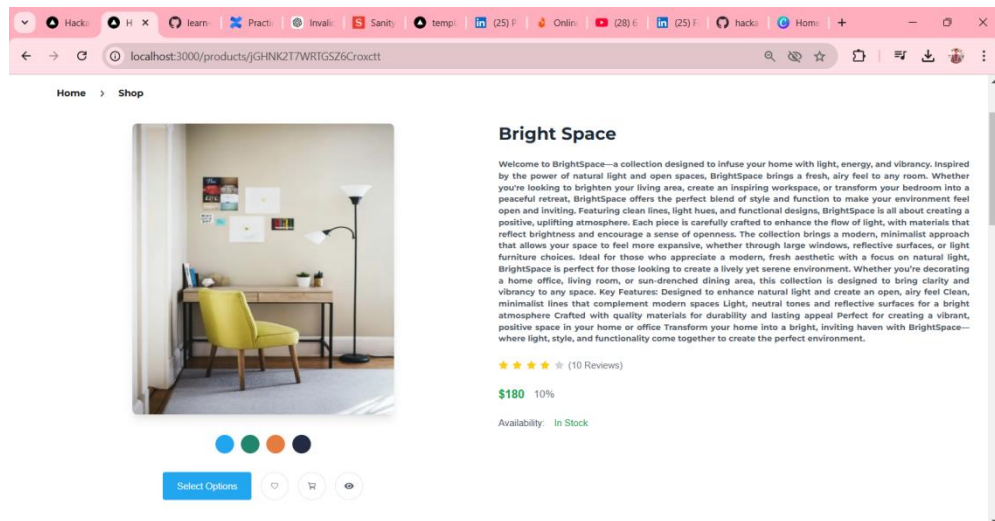


Frontend Display:

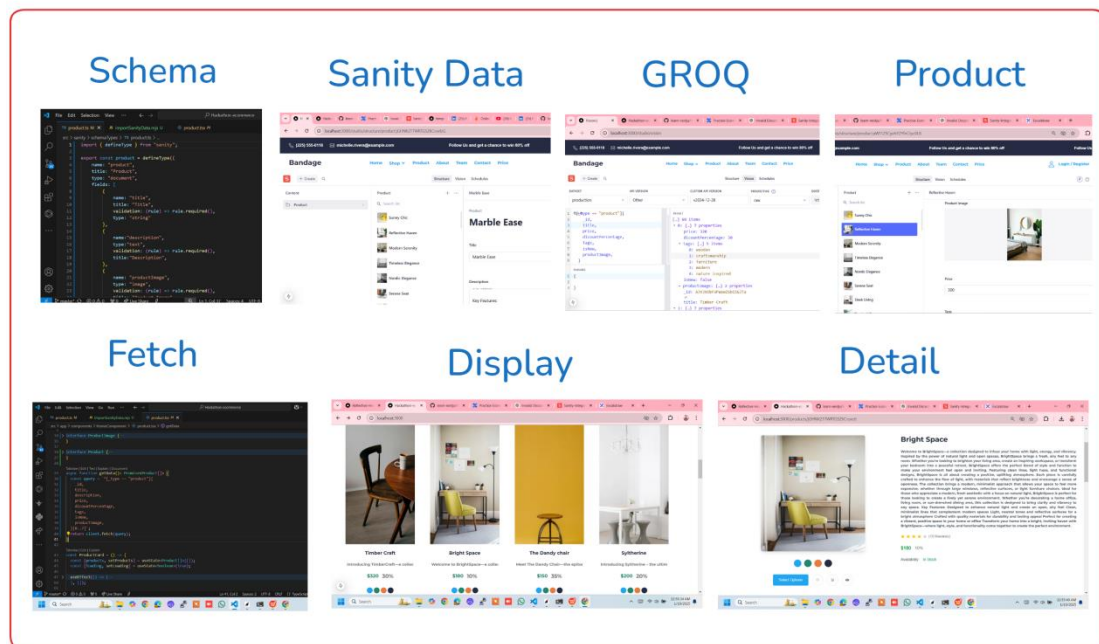
Screenshot demonstrating how the product data is displayed dynamically on the frontend of the marketplace.



Product Detail Page Display:



Process of Get Data From Sanity



Conclusion:

The API integration and data migration were successfully completed, enhancing the efficiency and scalability of the Bandage project. This integration streamlined the process of adding and updating product data in the marketplace, while the migration steps ensured data consistency and accuracy across the system. With this setup, the Bandage project is now more dynamic and easier to maintain.

Checklist

API Understanding



Schema Validation



Data Migration



API Integration in Next.js



Submission Preparation

