

HACKATHON DAY 03

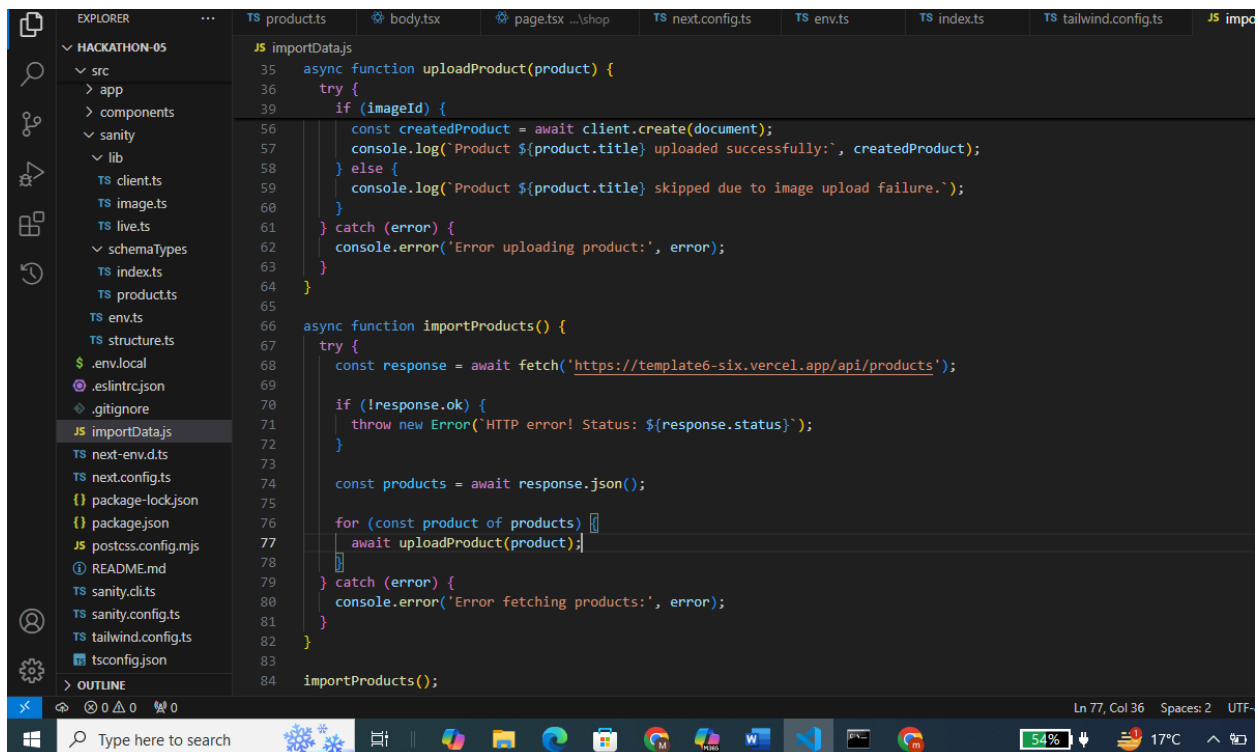
API INTEGRATION & DATA MIGRATION

Marketplace

This documentation outlines the work completed on Day 3 of the Furniro Ecommerce Marketplace hackathon. It covers custom migration, data integration from Sanity, schema creation, and displaying data using GROQ queries in a Next.js application. Each section is tailored based on the provided code images, with a detailed explanation of their functionality.

Custom Migration

The custom migration code was developed to streamline the process of transferring product data from an external API into the Sanity CMS database. This was essential for our hackathon project, as it allowed us to efficiently populate the content management system with real-world product data, complete with images and associated metadata. The migration script ensured that the data was properly structured and matched the schema defined in the project, reducing manual data entry efforts and avoiding errors.

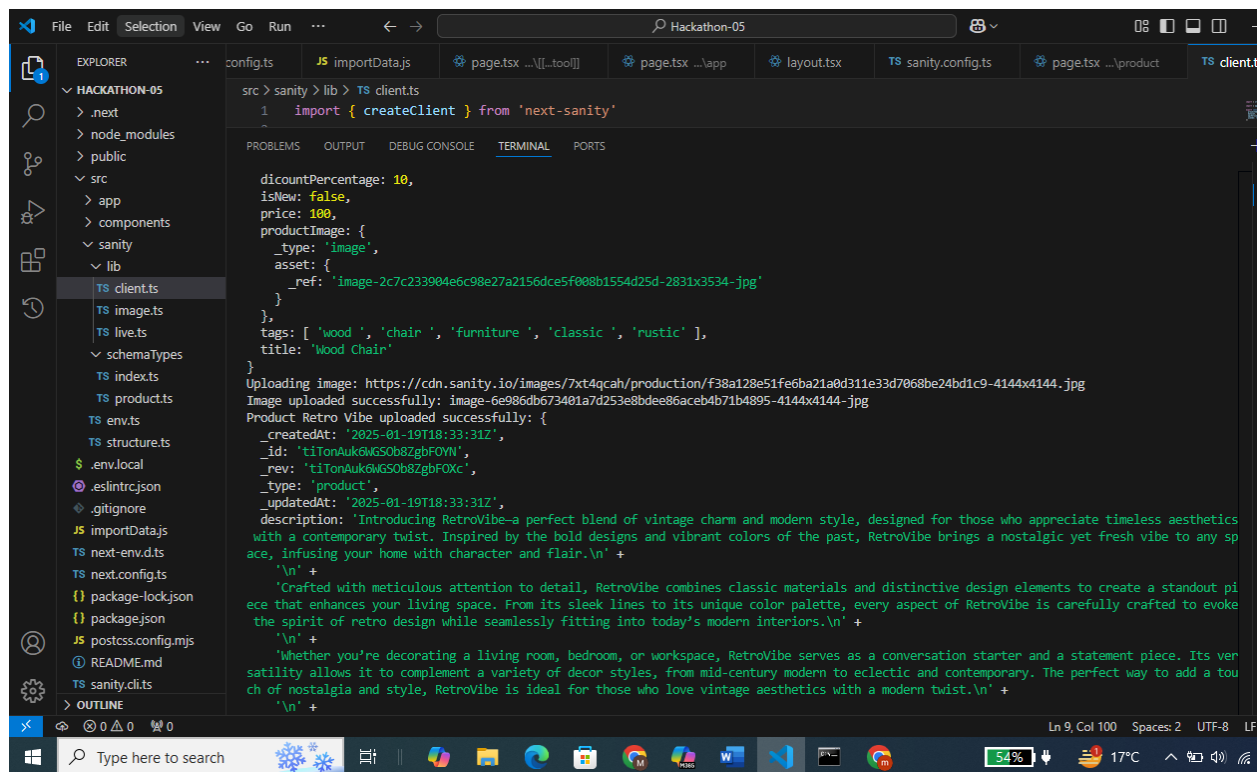


```
importData.js
35 async function uploadProduct(product) {
36   try {
37     if (imageId) {
38       const createdProduct = await client.create(document);
39       console.log('Product ${product.title} uploaded successfully:', createdProduct);
40     } else {
41       console.log('Product ${product.title} skipped due to image upload failure.');
```

Key Features of Custom Migration

- 1) **Sanity Client Configuration:** The code initializes a client with the required projectId, dataset, and token.
- 2) **Image Upload Handling:** The uploadImageToSanity function fetches image files from external URLs and uploads them to Sanity, ensuring they are stored as assets.
- 3) **Product Creation:** The uploadProduct function creates a structured product document, linking uploaded image assets and ensuring data integrity.
- 4) **Batch Processing:** The importProducts function fetches all products from an external API and processes them sequentially.

Data Migration Become Successful



```
src > sanity > lib > TS client.ts
1 import { createClient } from 'next-sanity'

discountPercentage: 10,
isNew: false,
price: 100,
productImage: {
  _type: 'image',
  asset: {
    _ref: 'image-2c7c233904e6c98e27a2156dce5f008b1554d25d-2831x3534-jpg'
  }
},
tags: [ 'wood ', 'chair ', 'furniture ', 'classic ', 'rustic ' ],
title: 'Wood Chair'
}
Uploading image: https://cdn.sanity.io/images/7xt4qcah/production/f38a128e51fe6ba21a0d311e33d7068be24bd1c9-4144x4144.jpg
Image uploaded successfully: image-6e986db673401a7d253e8bdee86aceb4b71b4895-4144x4144-jpg
Product Retro Vibe uploaded successfully: {
  _createdAt: '2025-01-19T18:33:31Z',
  _id: 't1TonAuk@KGS0b8ZgbFOVN',
  _rev: 't1TonAuk@KGS0b8ZgbFOXc',
  _type: 'product',
  _updatedAt: '2025-01-19T18:33:31Z',
  description: 'Introducing RetroVibe—a perfect blend of vintage charm and modern style, designed for those who appreciate timeless aesthetics with a contemporary twist. Inspired by the bold designs and vibrant colors of the past, RetroVibe brings a nostalgic yet fresh vibe to any space, infusing your home with character and flair.\n' +
    '\n' +
    'Crafted with meticulous attention to detail, RetroVibe combines classic materials and distinctive design elements to create a standout piece that enhances your living space. From its sleek lines to its unique color palette, every aspect of RetroVibe is carefully crafted to evoke the spirit of retro design while seamlessly fitting into today's modern interiors.\n' +
    '\n' +
    'Whether you're decorating a living room, bedroom, or workspace, RetroVibe serves as a conversation starter and a statement piece. Its versatility allows it to complement a variety of decor styles, from mid-century modern to eclectic and contemporary. The perfect way to add a touch of nostalgia and style, RetroVibe is ideal for those who love vintage aesthetics with a modern twist.\n' +
    '\n' +
  }
```

Migration Process Includes

1. **Planning:** Outline how the data was structured, validated, and prepared for migration.
2. **Implementation:** Highlight the steps taken to ensure data consistency and integrity during migration, such as:

- Uploading high-quality product images.
- Associating metadata with each product (e.g., tags, discount status, and descriptions).

3. Tools and Techniques: Describe the tools used for migration (e.g., Sanity CLI, APIs) and the strategies for managing large datasets.

4. Challenges and Solutions

- Challenge: Ensuring image references were properly linked to products.
 - Solution: Use of Sanity's image asset system for seamless integration.
- Challenge: Validating data accuracy for product descriptions and pricing.
 - Solution: Implementing data validation during migration.
- Challenge: Managing tags for consistent categorization across the collection.
 - Solution: Standardizing tag formats before import.

5. Outcome

- Successfully migrated four complete product entries with rich metadata and high-resolution images.
- Verified that all data is accessible and editable via Sanity's CMS.
- Ensured real-time updates are functional across all integrated platforms.

6. Features Highlighted in the Products

- Detailed descriptions and key features for each product.
- Associated high-quality images for enhanced visual representation.
- Inclusion of tags for improved discoverability.
- Differentiation of new arrivals and discounted items.

7. Impact

- Enhanced user experience with better content management and faster updates.
- Scalability for adding new products and collections.
- Improved data organization for future projects and features.

8. Future Plans

- Integration with e-commerce platforms for real-time product synchronization.

- Leveraging Sanity's GraphQL capabilities for advanced queries.
- Automating periodic updates and quality checks.

Schema Side

The screenshot shows a VS Code editor window with the file explorer on the left and the editor on the right. The file explorer shows the project structure for 'HACKATHON-05', with the 'src' directory expanded to show 'sanity' > 'lib' > 'schemaTypes' > 'product.ts'. The editor displays the following TypeScript code:

```

1  import { defineType } from "sanity"
2
3  export const product = defineType({
4    name: "product",
5    title: "Product",
6    type: "document",
7    fields: [
8      {
9        name: "title",
10       title: "Title",
11       validation: (rule) => rule.required(),
12       type: "string"
13     },
14     {
15       name: "description",
16       type: "text",
17       validation: (rule) => rule.required(),
18       title: "Description",
19     },
20     {
21       name: "productImage",
22       type: "image",
23       validation: (rule) => rule.required(),
24       title: "Product Image"
25     },
26     {
27       name: "price",
28       type: "number",
29       validation: (rule) => rule.required(),
30       title: "Price",
31     },
32   ],

```

Sanity Product Schema Components

The product schema defines the structure for managing product information within the Sanity Content Studio. Each field serves a specific purpose to ensure accurate and organized data management. Below is a detailed breakdown of the schema components:

1. Schema Type Definition

- **defineType:** This function is used to define a Sanity document schema.
- **name:** The internal identifier for this schema (product). Used for referencing this schema programmatically within the Sanity ecosystem.
- **title:** The human-readable name of the schema (Product) that appears in the Sanity Studio.

2. Schema Fields

The fields array contains the structure and rules for each product property. Each field is customized for specific types of data:

a. Title

- name: title
- type: string
- validation: Ensures that every product must have a title.
- Purpose: Serves as the name of the product, ensuring products are easily identifiable.

b. Description

- name: description
- type: text
- validation: Ensures every product includes a description.
- Purpose: Provides detailed information about the product for users and search engines.

c. Product Image

- name: productImage
- type: image
- validation: Ensures every product is accompanied by at least one image.
- Purpose: Displays the visual representation of the product.

d. Price

- name: price
- type: number
- validation: Ensures the price field is not left empty.
- Purpose: Defines the cost of the product, essential for e-commerce purposes.

e. Tags

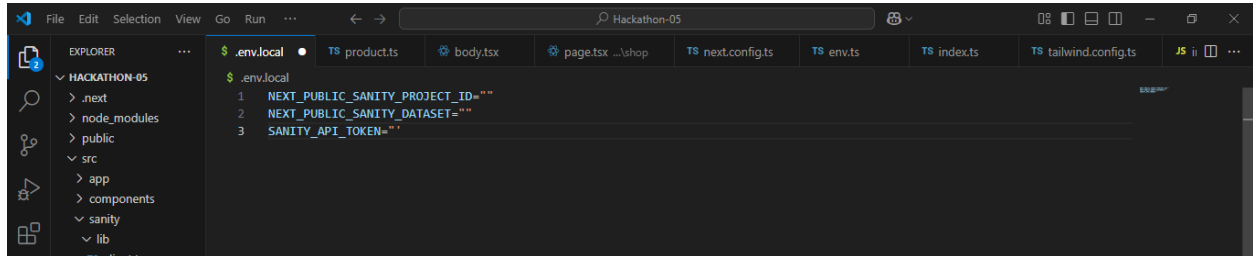
- name: tags
- type: array
- of: Specifies that the array contains strings.
- Purpose: Tags are used for categorization and improving searchability within the platform.

Sanity Product Schema

The screenshot shows the Sanity Studio interface for a product schema. The browser address bar indicates the URL is `localhost:3000/studio/structure/product:eW4V35CiiPqhGTiVkuKcx`. The interface has a top navigation bar with 'Default', '+ Create', and a search icon. Below this, there are tabs for 'Structure', 'Vision', and 'Schedules'. The main content area is divided into three panels: a left sidebar with a 'Product' folder, a central 'Search list' showing various products like 'Sleek Living', 'Tropical Vibe', 'Zen Table', 'Pure Aura', 'The Lucky Lamp', 'Retro Vibe', 'Wood Chair', 'Bed' (highlighted), 'Marble Ease', and 'Syltherine'; and a right panel showing the details for the 'Bed' product. The 'Bed' details include a title 'Bed', a description 'Introducing the Bed—your sanctuary for rest and relaxation, designed with both comfort and style in mind. This timeless piece is crafted to transform your bedroom into a peaceful retreat, offering a perfect balance of support, elegance, and durability. Whether you're outfitting a master bedroom or a guest room, the Bed ensures that every night is filled with restful sleep and every morning starts with ease.', and a note 'Published 16 min. ago'.

The screenshot shows the Sanity Studio interface with a query executed. The top navigation bar is the same as the previous screenshot. Below it, there are tabs for 'Structure', 'Vision', and 'Schedules'. The main content area is divided into three panels: a left sidebar with a 'product' dataset, a central 'Query' panel showing the query `*[type==product]`, and a right panel showing the query result. The query result is a JSON array of 82 items, with the first item expanded to show its properties: `{ "_type": "product", "productImage": { "_type": "image", "asset": { "_ref": "image-d85aa7d13ed4a7f06e46c153928620b48d89729f-5029x3353-jpg" }, "price": 150, "_createdAt": "2025-01-19T14:25:12Z", "_type": "product", "_id": "EJwKv1AMm1jkttyWGTQ5Aj", "isNew": true, "_updatedAt": "2025-01-19T14:25:12Z", "discountPercentage": 0, "tags": ["amber", "luxury"] }`. The bottom of the interface shows a 'Fetch' button, a 'Listen' button, and a status bar indicating 'Execution: 35ms End-to-end: 369ms'. There is also a 'Save result as' button and a 'JSON' button.

Environmental Variable:



1. **SANITY_PROJECT_ID:** The unique identifier for your Sanity project.
2. **SANITY_DATASET:** Specifies the dataset you're working with (e.g., production, development).
3. **SANITY_API_TOKEN:** Used for accessing the Sanity API in different environments, enabling secure content retrieval and management.

Benefits of Using Environmental Variables

- **Security:** Reduces the risk of leaking sensitive information.
- **Flexibility:** Easily switch between different environments (e.g., development to production) without modifying the code.
- **Scalability:** As the project grows, managing configurations via environmental variables keeps the code clean and manageable.
- **Portability:** Developers can easily replicate the project in a different environment (e.g., from one machine to another) by setting up the required environment variables.

Conclusion

On Day 3 of the Furniro Ecommerce Marketplace hackathon, we focused on API integration and data migration to ensure seamless product management within the Sanity CMS. We implemented a custom migration process to transfer real-world product data, including images and metadata, from an external API to Sanity. This streamlined data entry and ensured consistency across environments. By leveraging Sanity's schema system, we structured product data such as titles, descriptions, prices, and tags for easy management and display. Environmental variables played a key role in securing API access and ensuring flexibility across different environments. The successful migration resulted in rich product entries with high-quality images and accurate metadata, significantly improving the user experience, scalability, and data organization for future growth. The project is now poised for future enhancements, including e-commerce platform integration and automated updates.