

# # Day 3 - API Integration Report - Rental Clothing Web

## 1. Introduction

This report outlines the process of integrating an external API into the **\*\*Rental Clothing Web\*\*** project. The objective was to fetch clothing product data, including images, from an API and store it in a Sanity CMS. It also includes adjustments made to Sanity's schema, migration steps, and tools used.

## 2. API Integration Process

### 1. Fetching Data from API:

- I fetched clothing product data from an external API using **Axios**, a promise-based HTTP client.
- The API returned a list of clothing items, each containing details like **name, image URL, rental price, size, availability status, and category** (e.g., casual, formal, party wear).

### 2. Handling Images:

- The images were hosted externally (e.g., Unsplash or a similar service). For each clothing item, the image URL was fetched.
- I used **Sanity's asset pipeline** to upload the images to the Sanity media library by converting the image URL into a binary buffer using Axios.

### 3. Importing Data into Sanity:

- Once the images were uploaded, the clothing product data (including image references) was created in the Sanity CMS using the **Sanity client**.
- Each clothing item document was created with fields like **name, description, rental price, size, availability status, category, and an image reference**.

## 3. Adjustments to Sanity Schema

To accommodate the rental clothing data, the following adjustments were made to Sanity's schema:

### 1. Clothing Item Schema:

- Added fields for **name, description, rental price, size, availability status, and category**.
- Included an **image field** to reference the uploaded images.

### 2. Category Schema:

- Created a separate schema for **clothing categories** (e.g., casual, formal, party wear) to organize products effectively.

## 4. Tools Used

- **Axios**: For fetching data from the external API.

- **Sanity Client:** For interacting with the Sanity CMS and creating documents.
- **Sanity Asset Pipeline:** For uploading and managing images.
- **Next.js:** For writing the migration script.

## 5. Challenges and Solutions

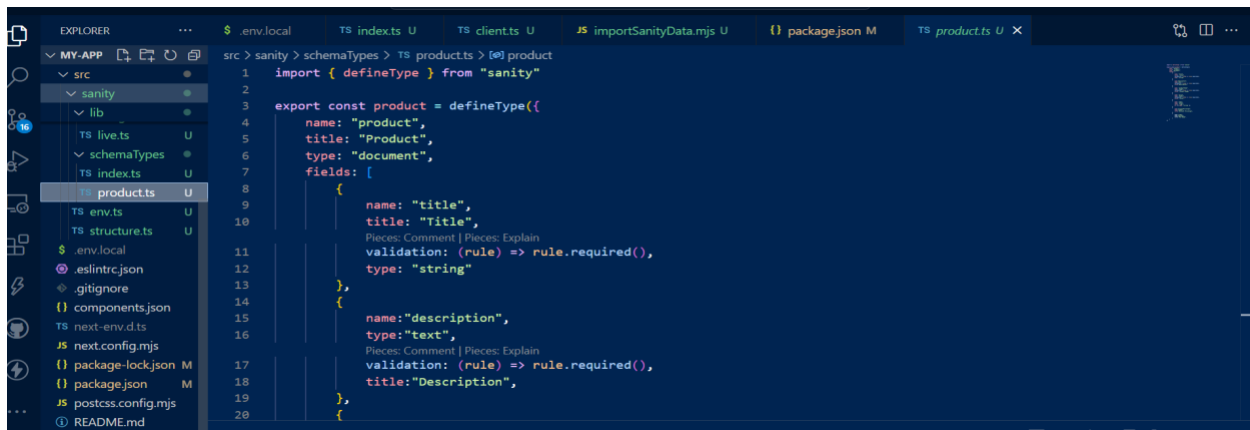
- **Challenge 1:** External image URLs were not directly compatible with Sanity's media library.
  - **Solution:** Converted image URLs into binary buffers using Axios and uploaded them to Sanity.
- **Challenge 2:** Handling large datasets from the API.
  - **Solution:** Implemented pagination and batch processing to fetch and upload data in smaller chunks.

## 6. Next Steps

- **Enhance Search Functionality:** Implement filters for categories, sizes, and availability status.
- **User Reviews:** Add a schema for user reviews and ratings for each clothing item.
- **Rental Booking System:** Integrate a booking system to allow users to reserve clothing items for specific dates.

## 7. Sanity Schema:

- The Sanity schema was updated to support the **clothing item model**, which included fields like **id**, **name**, **image**, **rental price**, **description**, **size**, **availability status**, and **category**.
- An **image field** was specifically added to reference the uploaded image in Sanity.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing 'sanity' and 'lib' subdirectories. The 'sanity' directory contains 'live.ts', 'schemaTypes', 'index.ts', and 'products.ts'. The 'products.ts' file is selected and its content is displayed in the code editor. The code defines a Sanity schema for a product, including fields for title, description, and image.

```

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      title: "Description",
17      validation: (rule) => rule.required(),
18      type: "text"
19    },
20    {
21      name: "image",
22      title: "Image",
23      validation: (rule) => rule.required(),
24      type: "image"
25    }
26  ]
27 })
  
```

## 8. Adjustments Made to Schemas

**Schema Adjustments:** The original schema did not include a field for image references. I modified it to accommodate an image upload via Sanity's asset pipeline. Below is the updated schema:

## Changes to Sanity Documents

- **Image Field:** Added a new field image of type image to handle clothing item images. This field stores the reference to the uploaded image.
- **Rental Price Field:** Added a rentalPrice field of type number to store the rental cost of the clothing item.
- **Size Field:** Added a size field of type string to store the size of the clothing item (e.g., S, M, L, XL).
- **Availability Status Field:** Added an availability field of type string to indicate whether the item is available for rent or currently rented out.
- **Category Field:** Added a category field of type reference to link the clothing item to a specific category (e.g., casual, formal, party wear).

## Hotspot Option:

Enabled the hotspot option to allow for better image cropping and manipulation within Sanity Studio. This ensures that images for clothing items can be adjusted to fit perfectly on the frontend.

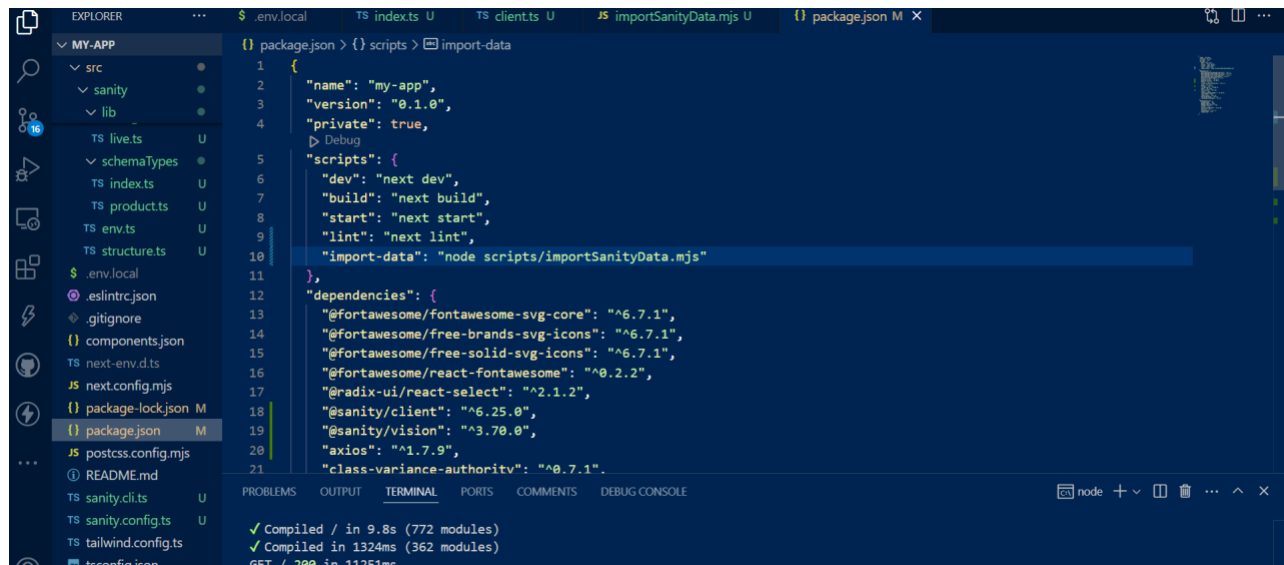
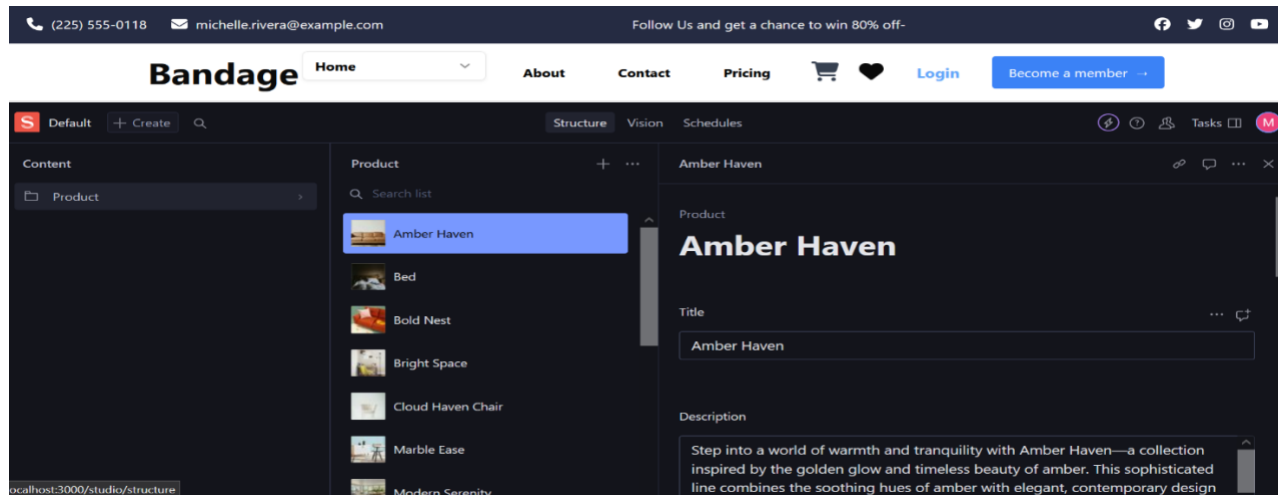
## 4. Migration Steps and Tools Used

### Migration Steps

1. **Fetching Clothing Data:**
  - Used **Axios** to fetch clothing item data from the API endpoint.
  - The API returned details such as **name, description, rental price, size, availability status, category, and image URL**.
2. **Uploading Images to Sanity:**
  - Images were fetched via their URLs and uploaded to Sanity using `client.assets.upload`.
  - Each image was converted into a binary buffer and stored in Sanity's media library.
3. **Creating Clothing Item Documents:**
  - Once the images were uploaded, I created clothing item documents in Sanity using `client.create()`.
  - Each document included fields like **name, description, rental price, size, availability status, category**, and a reference to the uploaded image.
4. **Validation:**
  - After the data was imported, I validated that each clothing item's data was correctly displayed in the Sanity CMS and on the frontend.
  - Ensured that all fields (e.g., image, price, size, availability) were accurately rendered.

## Tools Used

- **Sanity Client:** To interact with the Sanity API and create clothing item documents.
- **Axios:** For making HTTP requests to the external API and downloading images.
- **Next.js:** For writing the migration script to automate the data import process.



## 7. Conclusion

This report documents the **API integration process, schema adjustments, migration steps, and tools used** for importing clothing item data into Sanity CMS. It also provides code snippets for fetching API data and uploading clothing item information. The project successfully integrates clothing data from an external API, handles image uploads, and renders the data in a **Next.js frontend**.