Marketplace Technical Foundation
DesignAura Marketplace
[DAY # 3]

On Day 3, I integrated an external API into our marketplace using **Sanity CMS**. The integration involved making API requests to fetch data, updating the Sanity schema to store this data, and populating the CMS with the API response. This data was then displayed dynamically on the frontend of the marketplace.

API Integration

The goal of this process is to integrate an API into a Next.js application. This allows us to fetch and display data dynamically from an external source, ensuring the application is scalable, maintainable, and user-friendly

Step 1. Installed Axios

First, installed **Axios** using the command `npm install axios`. Axios is a popular JavaScript library used for making HTTP requests. It makes handling requests simpler and provides features like automatic error handling and JSON parsing.

Step 2. Created a Function to Fetch Data

then created a function called `fetchData()` that uses **Axios** to send an HTTP request to an external API and get data.

Here’s how it works:

- The function makes a `GET` request to the API.
- It returns the data if the request is successful or logs an error if something goes wrong..

Step 4. Displayed Data in Your Component

Once the data is fetched, you passed it as props to your component and used it to dynamically display the information on the page.

You used `.map()` to loop over the data and render it as a list of items, showing details like name and description.

Step 5. Tested the API Integration

To make sure the API integration was working:

- You logged the fetched data using `console.log()` to check if it was being retrieved correctly.
- You also tested the API manually using tools like **Postman** to ensure the external API was returning the expected data.

Step 6. Handled Errors Gracefully

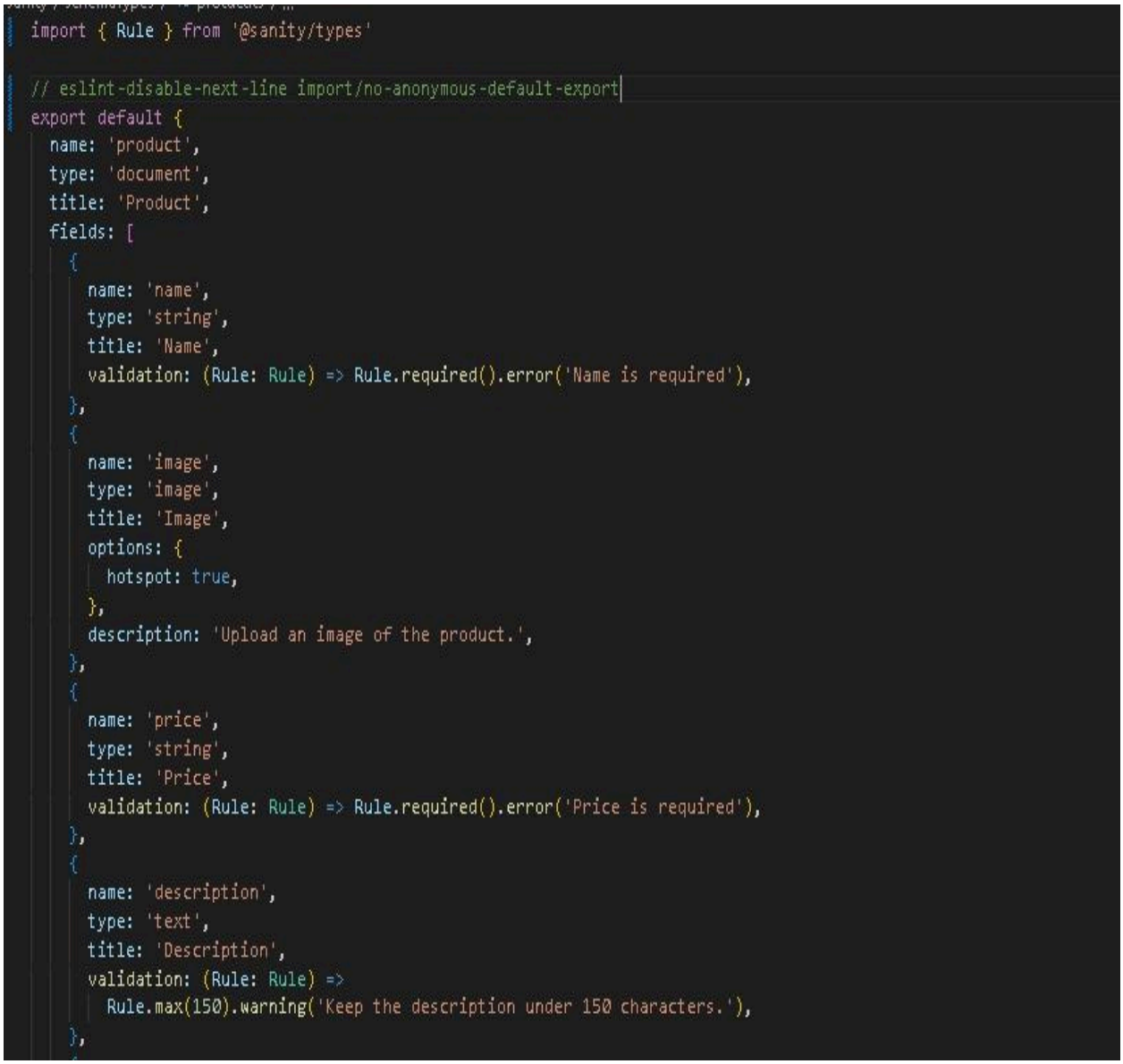
You made sure that if the API call failed (e.g., due to no internet or a bad API URL), the application would show a fallback message to the user. This ensures that users are not left with a blank or broken page

API calls. (screenshot)


```
Fetched Data: {
  products: [
    {
      id: '1',
      name: 'Tribù Elio Chair',
      imagePath: 'https://next-ecommerce-template-4.vercel.app/product/Chair (5
).png',
      price: '1200',
      description: 'A sleek outdoor chair with natural wooden elements and a mo
dern look.',
      discountPercentage: 10,
      isFeaturedProduct: true,
      stockLevel: 25,
      category: 'Chair'
    },
    {
      id: '2',
      name: 'Armchair Tortuga',
      imagePath: 'https://next-ecommerce-template-4.vercel.app/product/Chair (3
5).png',
      price: '850',
      description: 'An elegant armchair with plush cushions and a curved design
 for comfort.',
      discountPercentage: 0,
      isFeaturedProduct: false,
      stockLevel: 10,
      category: 'Chair'
    },
    {
      id: '3',
      name: 'Uchiwa Quilted Lounge Chair',
      imagePath: 'https://next-ecommerce-template-4.vercel.app/product/Chair (6
).png',
      price: '1600',
      description: 'A spacious lounge chair with a quilted back and soft cushio
ning.',
      discountPercentage: 20,
      isFeaturedProduct: true,
      stockLevel: 5,
      category: 'Chair'
    },
    {
      id: '19',
      name: 'Nordic Net Red Chair',
      imagePath: 'https://next-ecommerce-template-4.vercel.app/product/Chair (2
2).png',
      price: '320',
      description: 'An acrylic dining chair with a sleek and minimalist Nordic
design.',
      discountPercentage: 10,
      isFeaturedProduct: true,
      stockLevel: 20,
      category: 'Chair'
    },
    {
      id: '20',
      name: 'Cantilever Chair',
      imagePath: 'https://next-ecommerce-template-4.vercel.app/product/Chair (2
3).png',
      price: '780',
      description: 'A modern cantilever chair with a unique floating effect.',
      discountPercentage: 15,
      isFeaturedProduct: true,
      stockLevel: 5,
      category: 'Chair'
    },
    {
      id: '21',
      name: 'Luxury Flower Shell Sofa Chair',
      imagePath: 'https://next-ecommerce-template-4.vercel.app/product/Chair (3
4).png',
      price: '2500',
      description: 'A luxurious shell-shaped chair with gold brass metal legs.'
,
      discountPercentage: 0,
      isFeaturedProduct: true,
      stockLevel: 2,
      category: 'Sofa'
    }
  ]
}
```

Schema (Screenshot):

A schema in Sanity defines the structure of your data, specifying the fields, types, and relationships of your documents. For example, if you're managing product data, a schema will define the format for fields like product name, price, description, and image, and specify what type of data each field should accept (string, number, array, etc.).

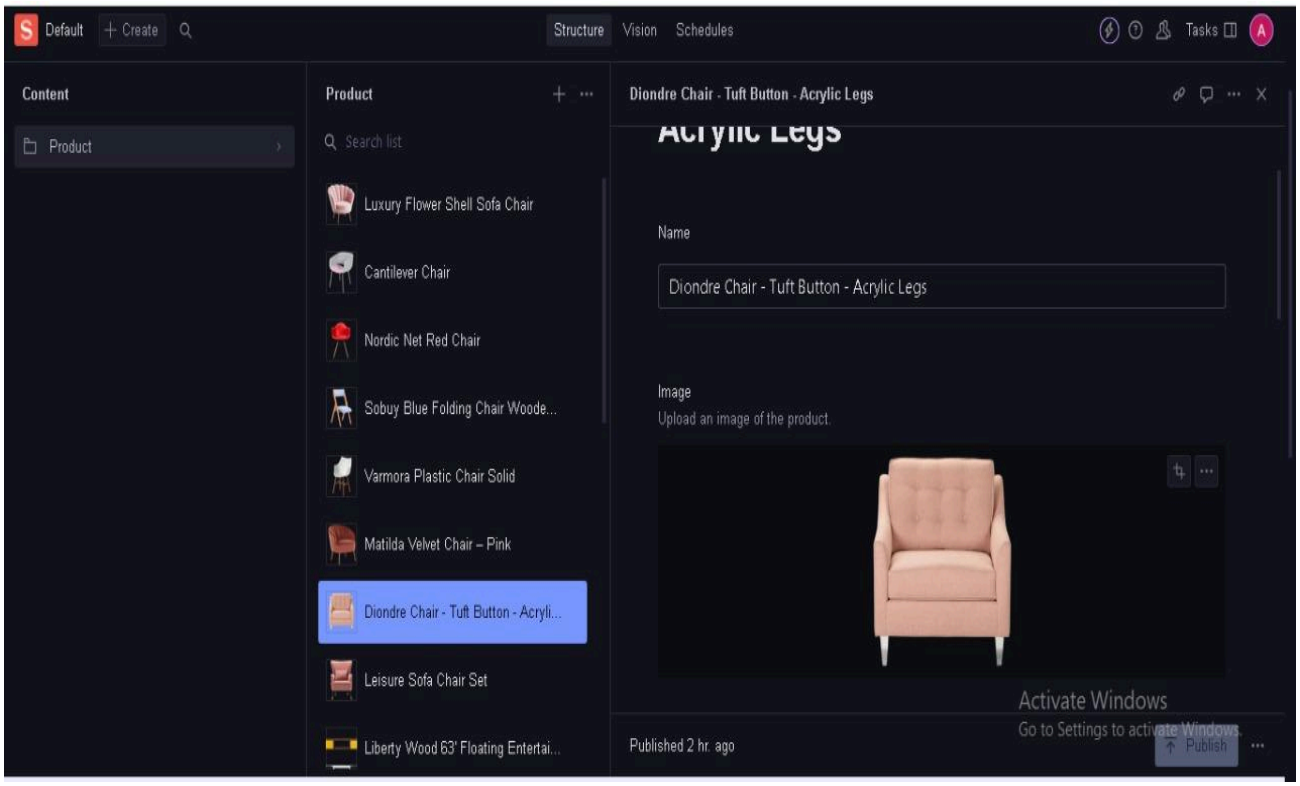


```
import { Rule } from '@sanity/types'

// eslint-disable-next-line import/no-anonymous-default-export
export default {
  name: 'product',
  type: 'document',
  title: 'Product',
  fields: [
    {
      name: 'name',
      type: 'string',
      title: 'Name',
      validation: (Rule: Rule) => Rule.required().error('Name is required'),
    },
    {
      name: 'image',
      type: 'image',
      title: 'Image',
      options: {
        hotspot: true,
      },
      description: 'Upload an image of the product.',
    },
    {
      name: 'price',
      type: 'string',
      title: 'Price',
      validation: (Rule: Rule) => Rule.required().error('Price is required'),
    },
    {
      name: 'description',
      type: 'text',
      title: 'Description',
      validation: (Rule: Rule) =>
        Rule.max(150).warning('Keep the description under 150 characters.'),
    },
```

Populated Sanity CMS Fields: (Screenshot) :

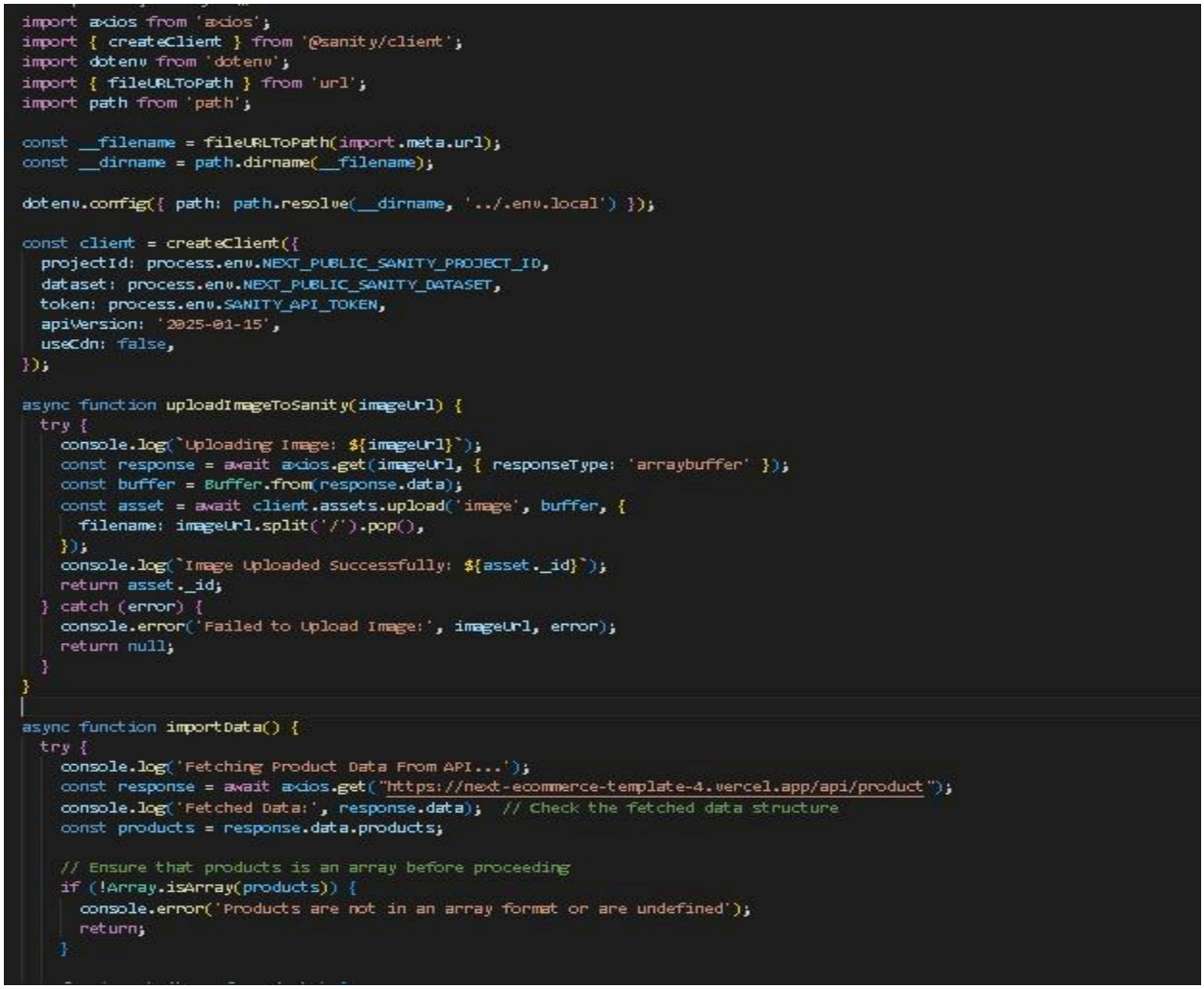
This refers to adding real content (like text, images, or other data) into the fields you've set up in your Sanity CMS. These fields are part of your content model (schema), and populating them means filling them with actual data that your website or app will use. This can be done manually in the Sanity Studio or automatically through scripts or APIs.



Code snippets for API integration and migration scripts (Screenshot) :

API Integration: This is when your app connects to another service (like a weather or product API) to get or send data. It helps your app stay dynamic and fetch real-time data from external sources.

Migration Scripts: These are scripts that help move or update data when you change your app's database or structure. For example, if you add new fields to your data or change its format, migration scripts help update everything automatically.



```
import axios from 'axios';
import { createClient } from '@sanity/client';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

dotenv.config({ path: path.resolve(__dirname, '../.env.local') });

const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  token: process.env.SANITY_API_TOKEN,
  apiVersion: '2025-01-15',
  useCdn: false,
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading Image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop(),
    });
    console.log(`Image Uploaded Successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to Upload Image:', imageUrl, error);
    return null;
  }
}

async function importData() {
  try {
    console.log('Fetching Product Data From API...');
    const response = await axios.get("https://next-ecommerce-template-4.vercel.app/api/product");
    console.log('Fetched Data:', response.data);  // Check the fetched data structure
    const products = response.data.products;

    // Ensure that products is an array before proceeding
    if (!Array.isArray(products)) {
      console.error('Products are not in an array format or are undefined');
      return;
    }
```

Day 3 Learning Checklist

- API Understanding ✔️
- Schema Validation ✔️
- Data Migration ✔️
- API Integration in Next.js ✔️
- Submission Preparation ✔️
