Day 3 Hackathon Challenge Documentation

API Integration and Data Migration

This documentation outlines the work completed on Day 3 of the BandageWeb 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.

Overview of the Challenge

The goal for Day 3 of the Hackathon was to:

1. **Integrate API:** Dynamically fetch product data from an external API.
2. **Data Migration:** Use Sanity CMS to create and structure the product schema, then migrate the API data into the Sanity backend for further management.
3. **Frontend Integration:** Display the products dynamically using the migrated data in a clean, responsive frontend UI.
4. **Type-Safe Queries:** Use Typegen and Zod schemas for GROQ queries to ensure type safety.

Steps to Complete the Challenge

Step 1: API Integration

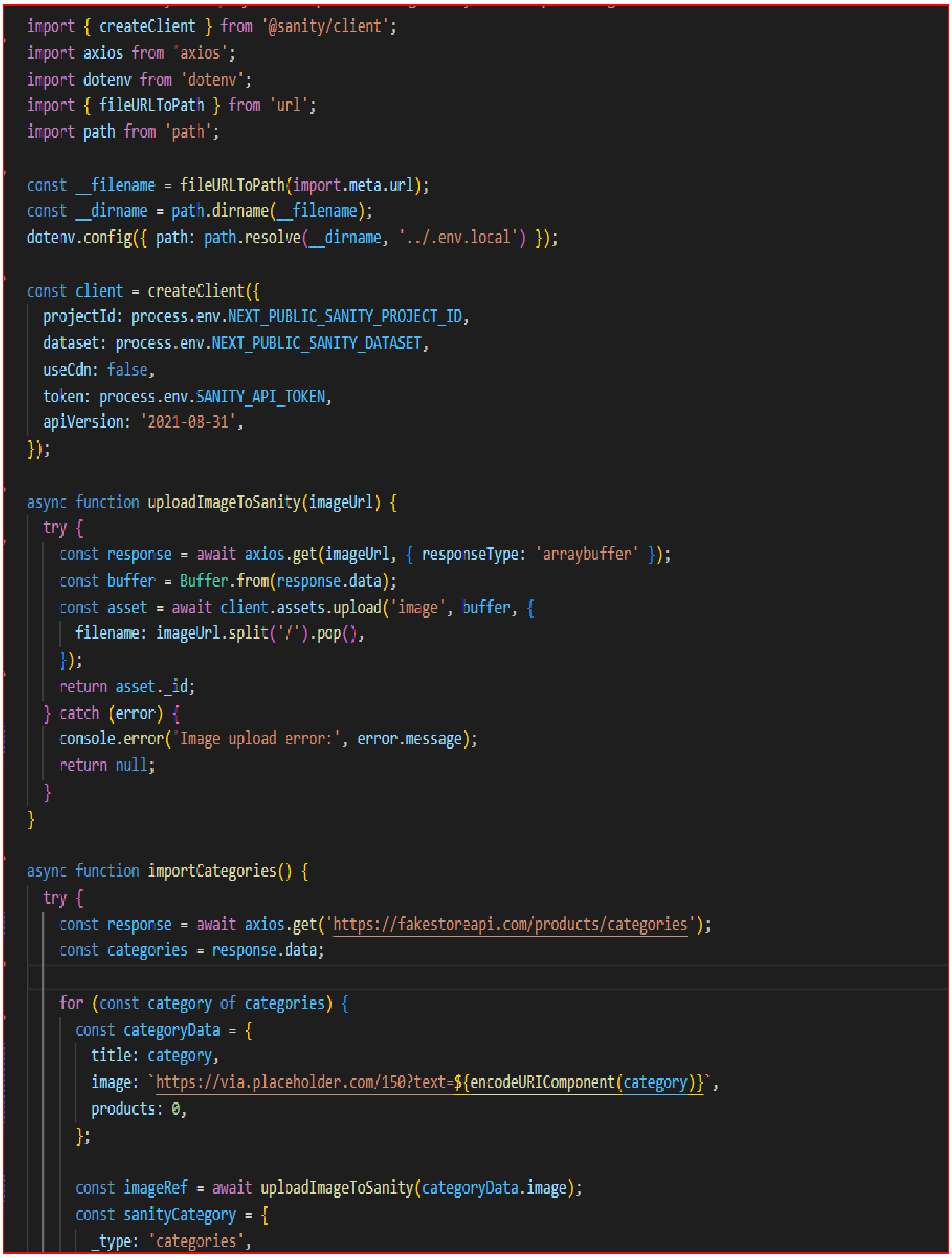
API Endpoint:
I used a sample API that provides product details (e.g., product name, price, stock, and images).

- **Example API URL:** `https://fakestoreapi.com/products`

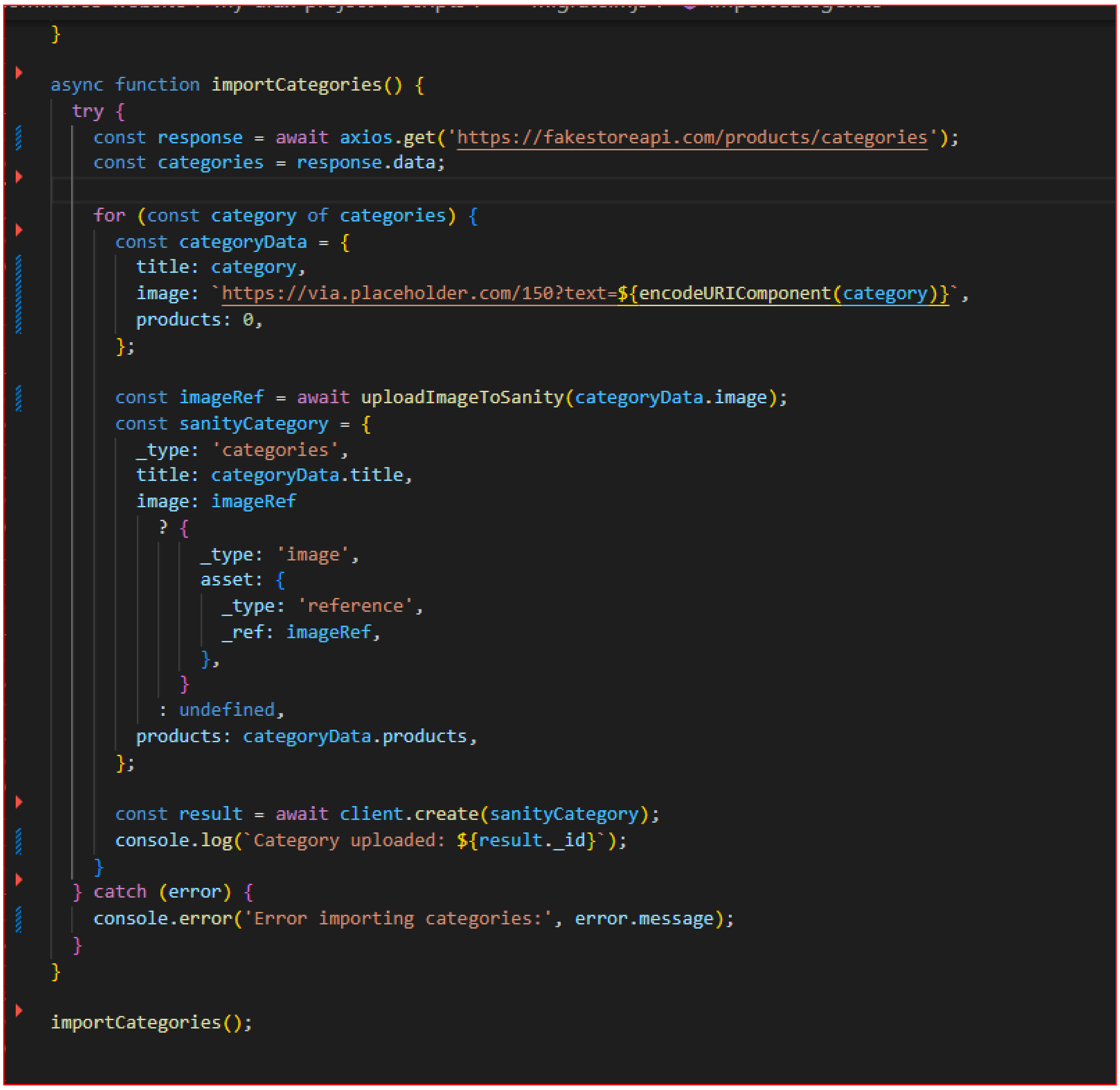
Fetching Data:

- Implemented API calls using `axios` in the Next.js frontend.

- Ensured error handling for failed API requests.
- Verified data mapping to match the frontend display requirements.



```
import { createClient } from '@sanity/client';
import axios from 'axios';
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,
  useCdn: false,
  token: process.env.SANITY_API_TOKEN,
  apiVersion: '2021-08-31',
});

async function uploadImageToSanity(imageUrl) {
  try {
    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(),
    });
    return asset._id;
  } catch (error) {
    console.error('Image upload error:', error.message);
    return null;
  }
}

async function importCategories() {
  try {
    const response = await axios.get('https://fakestoreapi.com/products/categories');
    const categories = response.data;

    for (const category of categories) {
      const categoryData = {
        title: category,
        image: `https://via.placeholder.com/150?text=${encodeURIComponent(category)}`,
        products: 0,
      };

      const imageRef = await uploadImageToSanity(categoryData.image);
      const sanityCategory = {
        _type: 'categories',
```



```
}

async function importCategories() {
  try {
    const response = await axios.get('https://fakestoreapi.com/products/categories');
    const categories = response.data;

    for (const category of categories) {
      const categoryData = {
        title: category,
        image: `https://via.placeholder.com/150?text=${encodeURIComponent(category)}`,
        products: 0,
      };

      const imageRef = await uploadImageToSanity(categoryData.image);
      const sanityCategory = {
        _type: 'categories',
        title: categoryData.title,
        image: imageRef
          ? {
              _type: 'image',
              asset: {
                _type: 'reference',
                _ref: imageRef,
              },
            }
          : undefined,
        products: categoryData.products,
      };

      const result = await client.create(sanityCategory);
      console.log(`Category uploaded: ${result._id}`);
    }
  } catch (error) {
    console.error('Error importing categories:', error.message);
  }
}

importCategories();
```

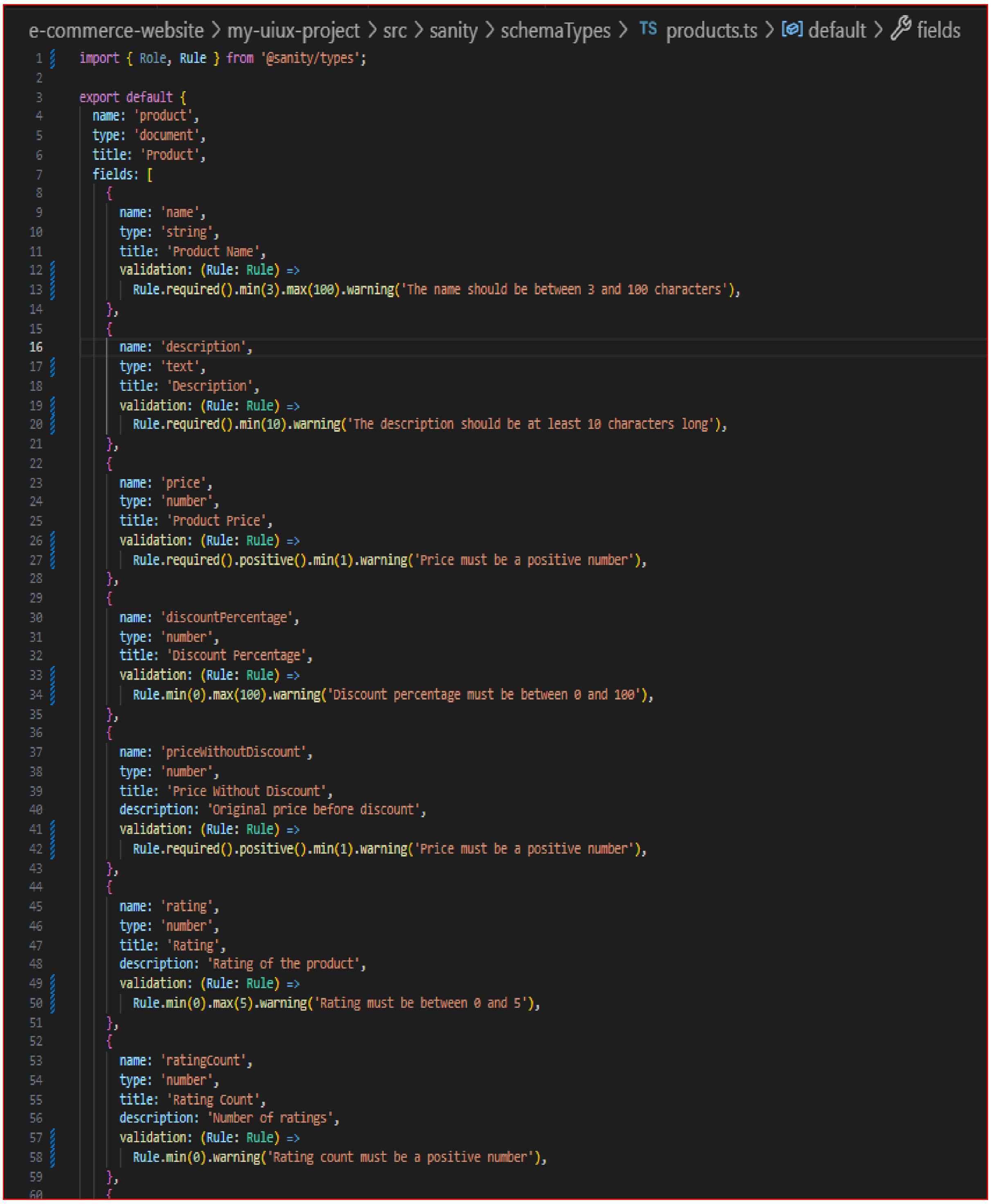
Code Highlights:

Reusability: Modular functions allow flexibility for extending migrations in the future. b. Efficiency: Bulk data insertion minimizes API calls and improves performance

Step 2: Data Migration to Sanity CMS

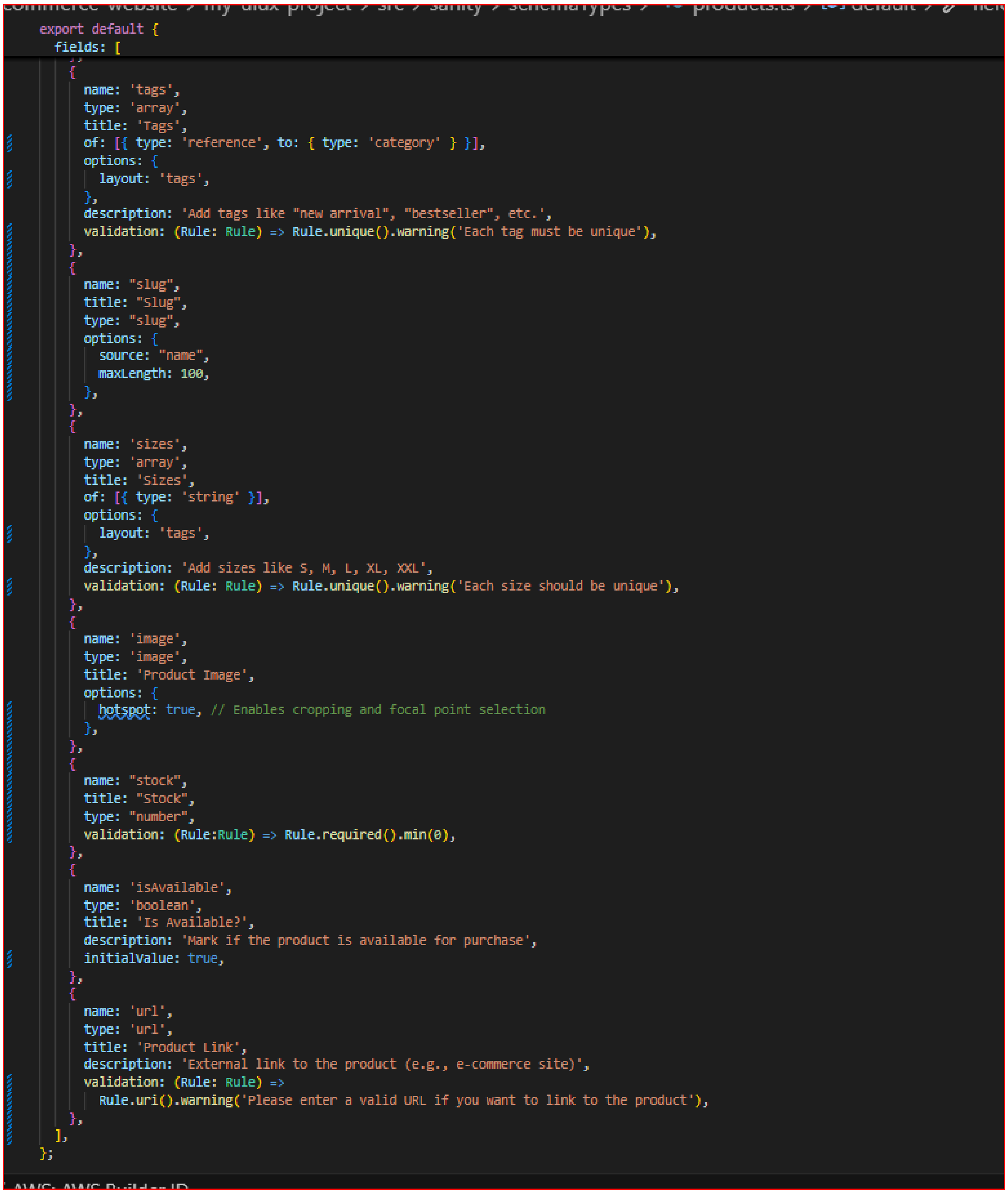
1. **Sanity Schema Creation:**
   I created a custom Sanity schema for managing product data. The schema includes fields such as:
   - `name` (string): Product Name
   - `description` (text): Product Description
   - `price` (number): Product Price
   - `stock` (number): Available Stock

- `image` (array): List of image URLs



e-commerce-website > my-uiux-project > src > sanity > schemaTypes > products.ts > default > fields

```typescript
import { Role, Rule } from '@sanity/types';

export default {
  name: 'product',
  type: 'document',
  title: 'Product',
  fields: [
    {
      name: 'name',
      type: 'string',
      title: 'Product Name',
      validation: (Rule: Rule) =>
        Rule.required().min(3).max(100).warning('The name should be between 3 and 100 characters'),
    },
    {
      name: 'description',
      type: 'text',
      title: 'Description',
      validation: (Rule: Rule) =>
        Rule.required().min(10).warning('The description should be at least 10 characters long'),
    },
    {
      name: 'price',
      type: 'number',
      title: 'Product Price',
      validation: (Rule: Rule) =>
        Rule.required().positive().min(1).warning('Price must be a positive number'),
    },
    {
      name: 'discountPercentage',
      type: 'number',
      title: 'Discount Percentage',
      validation: (Rule: Rule) =>
        Rule.min(0).max(100).warning('Discount percentage must be between 0 and 100'),
    },
    {
      name: 'priceWithoutDiscount',
      type: 'number',
      title: 'Price Without Discount',
      description: 'Original price before discount',
      validation: (Rule: Rule) =>
        Rule.required().positive().min(1).warning('Price must be a positive number'),
    },
    {
      name: 'rating',
      type: 'number',
      title: 'Rating',
      description: 'Rating of the product',
      validation: (Rule: Rule) =>
        Rule.min(0).max(5).warning('Rating must be between 0 and 5'),
    },
    {
      name: 'ratingCount',
      type: 'number',
      title: 'Rating Count',
      description: 'Number of ratings',
      validation: (Rule: Rule) =>
        Rule.min(0).warning('Rating count must be a positive number'),
    },
    {
```

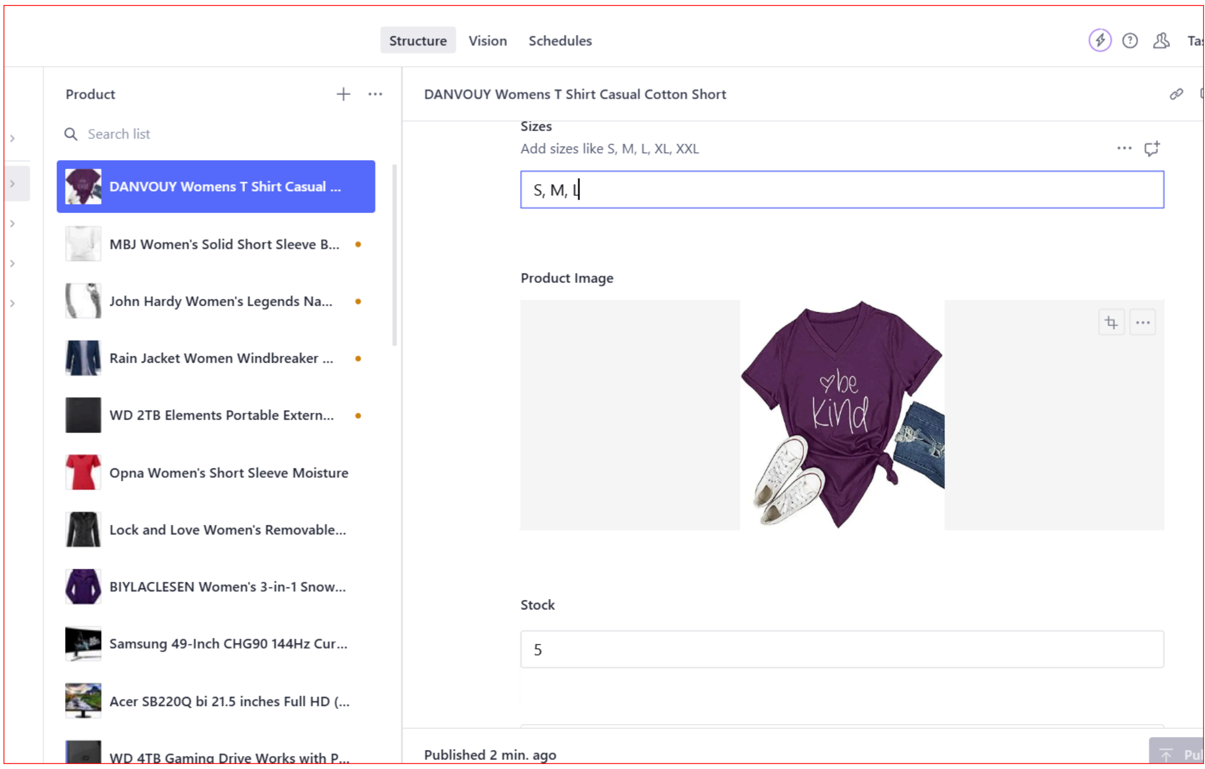


```
export default {
  fields: [
    {
      name: 'tags',
      type: 'array',
      title: 'Tags',
      of: [{ type: 'reference', to: { type: 'category' } }],
      options: {
        layout: 'tags',
      },
      description: 'Add tags like "new arrival", "bestseller", etc.',
      validation: (Rule: Rule) => Rule.unique().warning('Each tag must be unique'),
    },
    {
      name: "slug",
      title: "Slug",
      type: "slug",
      options: {
        source: "name",
        maxLength: 100,
      },
    },
    {
      name: 'sizes',
      type: 'array',
      title: 'Sizes',
      of: [{ type: 'string' }],
      options: {
        layout: 'tags',
      },
      description: 'Add sizes like S, M, L, XL, XXL',
      validation: (Rule: Rule) => Rule.unique().warning('Each size should be unique'),
    },
    {
      name: 'image',
      type: 'image',
      title: 'Product Image',
      options: {
        hotspot: true, // Enables cropping and focal point selection
      },
    },
    {
      name: "stock",
      title: "Stock",
      type: "number",
      validation: (Rule:Rule) => Rule.required().min(0),
    },
    {
      name: 'isAvailable',
      type: 'boolean',
      title: 'Is Available?',
      description: 'Mark if the product is available for purchase',
      initialValue: true,
    },
    {
      name: 'url',
      type: 'url',
      title: 'Product Link',
      description: 'External link to the product (e.g., e-commerce site)',
      validation: (Rule: Rule) =>
        Rule.uri().warning('Please enter a valid URL if you want to link to the product'),
    },
  ],
};
```

Imported the API data into Sanity using the Sanity client.

Ensured that the data was mapped correctly to match the schema fields.

Uploaded product images to Sanity’s asset pipeline.



DANVOUY Womens T Shirt Casual ...
MBJ Women's Solid Short Sleeve B...
John Hardy Women's Legends Na...
Rain Jacket Women Windbreaker ...
WD 2TB Elements Portable Extern...
Opna Women's Short Sleeve Moisture
Lock and Love Women's Removable...
BIYLACLESEN Women's 3-in-1 Snow...
Samsung 49-Inch CHG90 144Hz Cur...
Acer SB220Q bi 21.5 inches Full HD (...
WD 4TB Gaming Drive Works with P...

DANVOUY Womens T Shirt Casual Cotton Short

Sizes
Add sizes like S, M, L, XL, XXL
S, M, L

Product Image

Stock
5

Published 2 min. ago

Step 3: Frontend Integration

1. **Dynamic Product Display:**
   - Fetched products from Sanity CMS using `groq` queries.
   - Rendered the products dynamically using React components.
   - Styled the product grid using Tailwind CSS for a responsive design.
2. **Responsive UI:**
   - Ensured a clean layout for desktop, tablet, and mobile views.
   - Used Tailwind utilities to maintain consistent spacing and alignments.

Code Snippet for Product Rendering:

```
import React, { useEffect, useState } from 'react';

import fetchProductsFromSanity from '../lib/fetchProducts';

const ProductGrid = () => {

    const [products, setProducts] = useState([]);

    useEffect(() => {
        const fetchProducts = async () => {

            const data = await fetchProductsFromSanity();

            setProducts(data);

        };

        fetchProducts();

    }, []);

    return (

        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">

            {products.map((product) => (

                <div key={product._id} className="p-4 border rounded
shadow-sm">

                    <img src={product.image} alt={product.name}
className="h-48 w-full object-cover" />

                    <h2 className="text-lg
font-bold">{product.name}</h2>

 <p className="text-gray-600">${product.price.toFixed(2)}</p>

    </div>

     ))}

   </div>

    )};


export default ProductGrid;
```

Step 4: Testing and Validation

1. **Functionality Testing:**
    - Verified API calls successfully retrieve product data.
    - Checked that all migrated data is visible in Sanity CMS.

   - Validated dynamic rendering on the frontend.
2. **Responsiveness Testing:**
   - Tested on different devices using Chrome DevTools.
   - Confirmed consistent behavior across mobile, tablet, and desktop resolutions.
3. **Error Handling:**
   - Added error messages for failed API requests.
   - Ensured fallback content displays when no products are available.
   -

Challenges Faced

1. Handling data inconsistencies from the API.
2. Debugging Sanity schema issues during data migration.

Lessons Learned

1. How to structure and manage Sanity schemas effectively.
2. Best practices for API integration in Next.js.
3. Learning and integrating Typegen.
4. Enhancing responsiveness with Tailwind CSS.

Submitted By: Shehreen Arshad