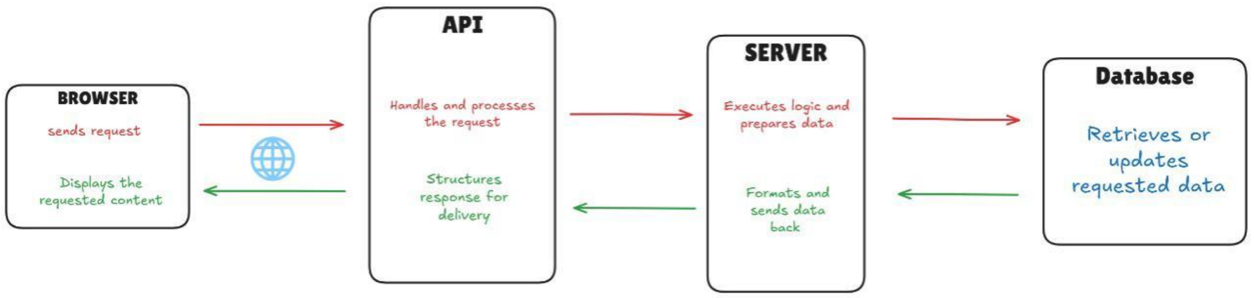
API Integration and Data Migration

By: Kulsoom Imran


BROWSER
sends request
Displays the requested content
API
Handles and processes the request
Structures response for delivery
SERVER
Executes logic and prepares data
Formats and sends data back
Database
Retrieves or updates requested data

API Integration Process

1. Configuring Sanity CMS

- Setting up a Sanity project with the required dataset.
- Installing necessary dependencies (`@sanity/client`).
- Defining schemas for storing API data (e.g., product data).

2. Defining Schemas in Sanity

- Creating **custom schemas** in Sanity to store relevant data (e.g., product name, price, images).
- Ensuring schema fields align with the API data structure.

3. Connecting to External API

- Obtaining necessary **API keys** for authentication.
- Setting up API client using **Axios**.

4. Store Data in Sanity

- Using the **Sanity client** to create documents based on the API data.

5. Query Data from Sanity

- Using **GROQ queries** to fetch the data from Sanity CMS.

6. Display Data on Frontend

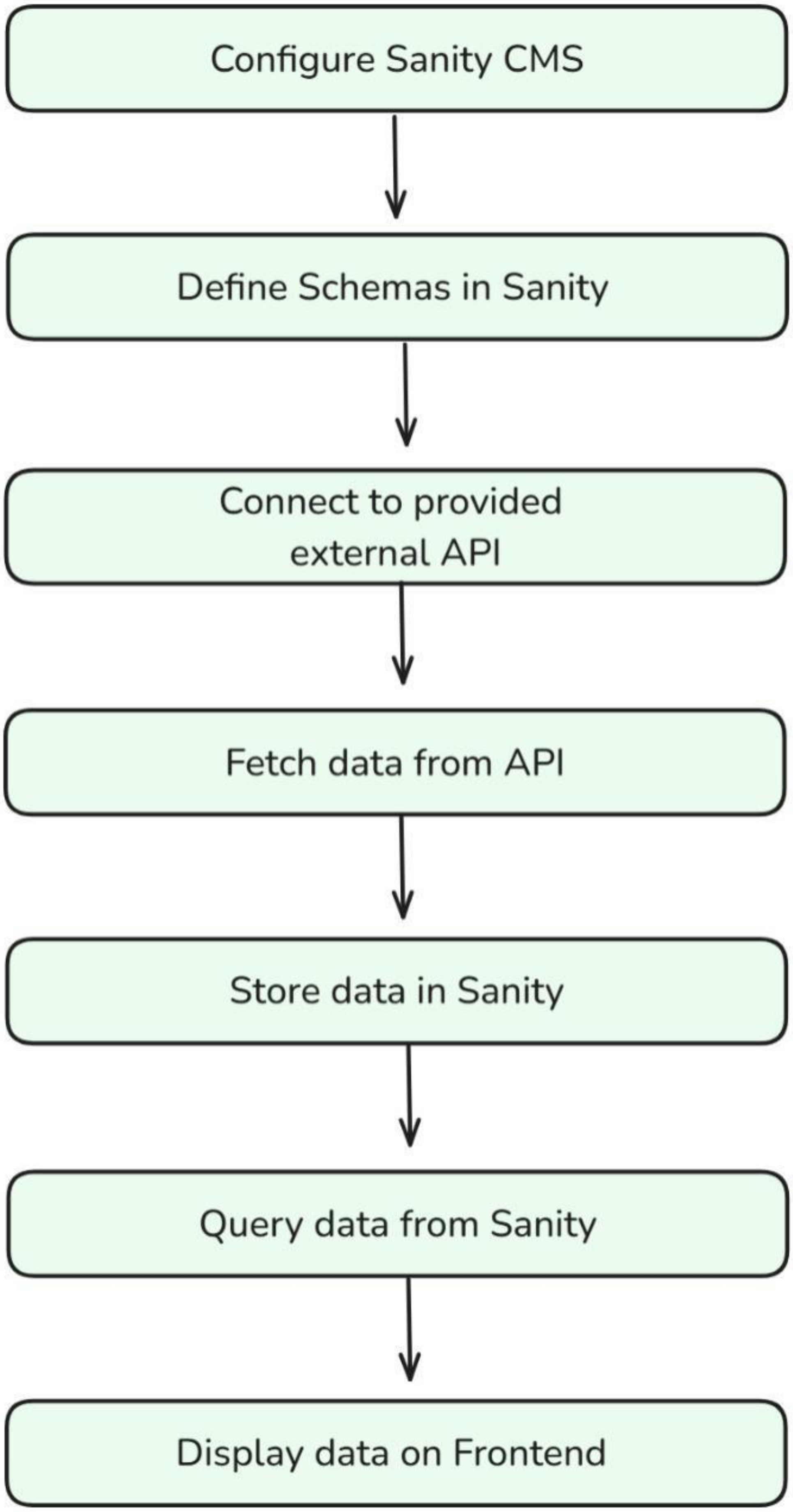
- Using a frontend framework, **Next.js** to displaying the fetched data.
- Rendering the data dynamically (e.g., list of products) in the **UI**.

Tools & Technologies Used:

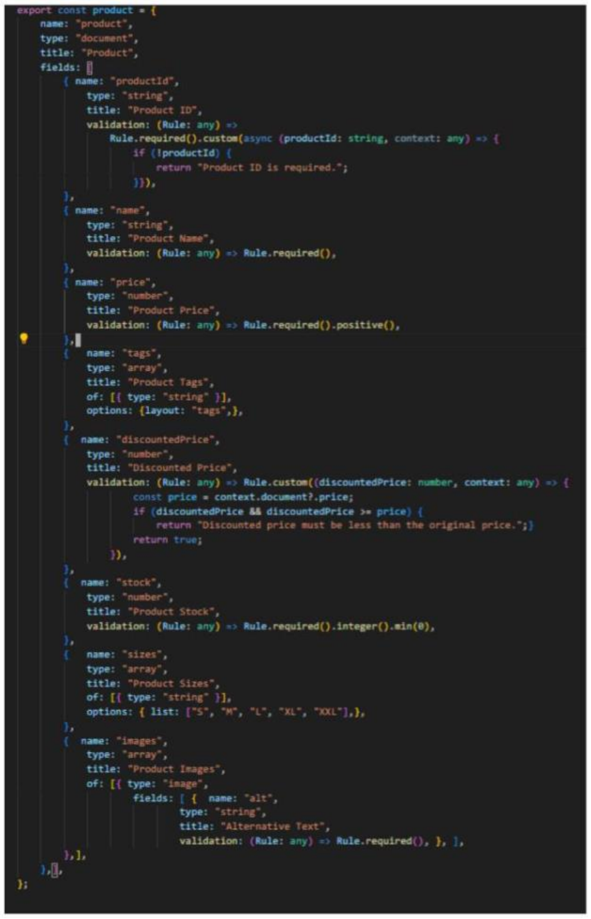
- **Sanity CMS:** Content management and data storage.
- **API Client:** Fetch API data.
- **GROQ:** For querying data from Sanity.
- **Frontend (Next.js):** For displaying data.

Conclusion:

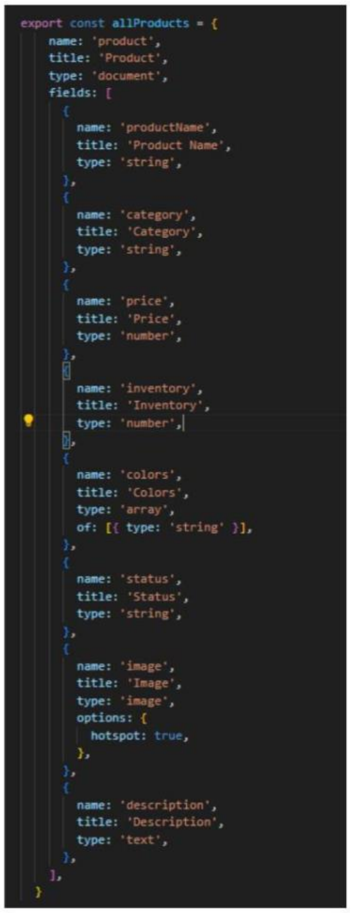
API data is fetched, stored in Sanity CMS, and rendered on the frontend for seamless integration of dynamic content.


Configure Sanity CMS
Define Schemas in Sanity
Connect to provided external API
Fetch data from API
Store data in Sanity
Query data from Sanity
Display data on Frontend

Adjustments made to Schemas



```
export const product = {
    name: "product",
    type: "document",
    title: "Product",
    fields: [
        { name: "productId",
            type: "string",
            title: "Product ID",
            validation: (Rule: any) =>
                Rule.required().custom(async (productId: string, context: any) => {
                    if (!productId) {
                        return "Product ID is required.";
                    }}),
        },
        { name: "name",
            type: "string",
            title: "Product Name",
            validation: (Rule: any) => Rule.required(),
        },
        { name: "price",
            type: "number",
            title: "Product Price",
            validation: (Rule: any) => Rule.required().positive(),
        },
        {   name: "tags",
            type: "array",
            title: "Product Tags",
            of: [{ type: "string" }],
            options: {layout: "tags",},
        },
        {  name: "discountedPrice",
            type: "number",
            title: "Discounted Price",
            validation: (Rule: any) => Rule.custom((discountedPrice: number, context: any) => {
                    const price = context.document?.price;
                    if (discountedPrice && discountedPrice >= price) {
                        return "Discounted price must be less than the original price.";}
                    return true;
                }),
        },
        {  name: "stock",
            type: "number",
            title: "Product Stock",
            validation: (Rule: any) => Rule.required().integer().min(0),
        },
        {   name: "sizes",
            type: "array",
            title: "Product Sizes",
            of: [{ type: "string" }],
            options: { list: ["S", "M", "L", "XL", "XXL"],},
        },
        {  name: "images",
            type: "array",
            title: "Product Images",
            of: [{ type: "image",
                    fields: [ {  name: "alt",
                            type: "string",
                            title: "Alternative Text",
                            validation: (Rule: any) => Rule.required(), }, ],
        },],
    },],
};
```



```
export const allProducts = {
    name: 'product',
    title: 'Product',
    type: 'document',
    fields: [
      {
        name: 'productName',
        title: 'Product Name',
        type: 'string',
      },
      {
        name: 'category',
        title: 'Category',
        type: 'string',
      },
      {
        name: 'price',
        title: 'Price',
        type: 'number',
      },
      {
        name: 'inventory',
        title: 'Inventory',
        type: 'number',
      },
      {
        name: 'colors',
        title: 'Colors',
        type: 'array',
        of: [{ type: 'string' }],
      },
      {
        name: 'status',
        title: 'Status',
        type: 'string',
      },
      {
        name: 'image',
        title: 'Image',
        type: 'image',
        options: {
          hotspot: true,
        },
      },
      {
        name: 'description',
        title: 'Description',
        type: 'text',
      },
    ],
  }
```

Migration Steps and Tools used

1. Setup Environment Variables

➢ Create a .env.local file to store sensitive data (e.g., Sanity project ID, dataset, and API token).

2. Initialize Sanity Client

- Use createClient from @sanity/client to configure the connection to your Sanity project.

3. Fetch Data

- Use Axios to fetch product data from the API endpoint **https://template-03-api.vercel.app/api/products**.

4. Image Upload

- Download images from the source URL using Axios and convert them to a buffer using Buffer.from().
- Upload images to Sanity using the assets.upload() method.

5. Transform and Upload Products

- Loop through the fetched products and transform them to match the Sanity schema.

6. Create Product Entries

- Use client.create() to insert the transformed product data into Sanity.

7. Error Handling

- Implement try-catch blocks to log errors during the upload or data transformation process.

8. Run Migration Script

- ➢ Execute the importData function to start the migration process.
- ➢ Logs indicate progress and completion status.

Tools Used:

Sanity.io

- ➢ For content management and data storage.
- ➢ Used @sanity/client library for interacting with the Sanity API.

Axios

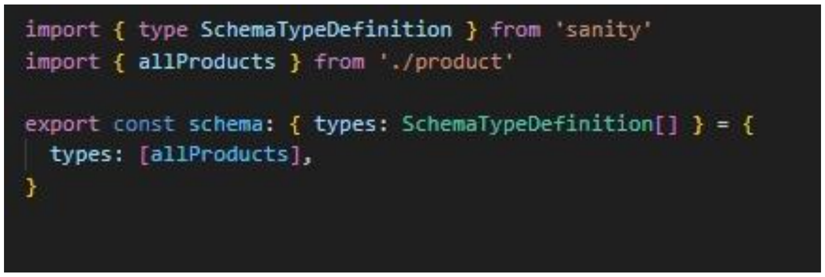
- ➢ For making HTTP requests to fetch data and images from the source API.

Dotenv

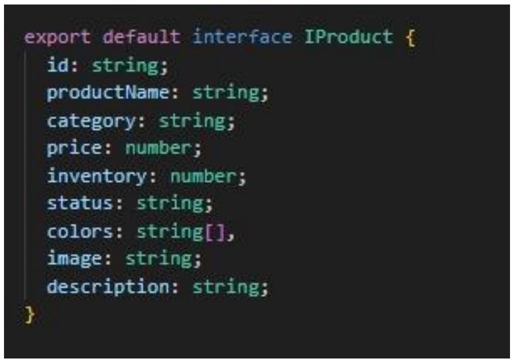
- ➢ For loading environment variables from .env.local.

Node.js Built-in Modules

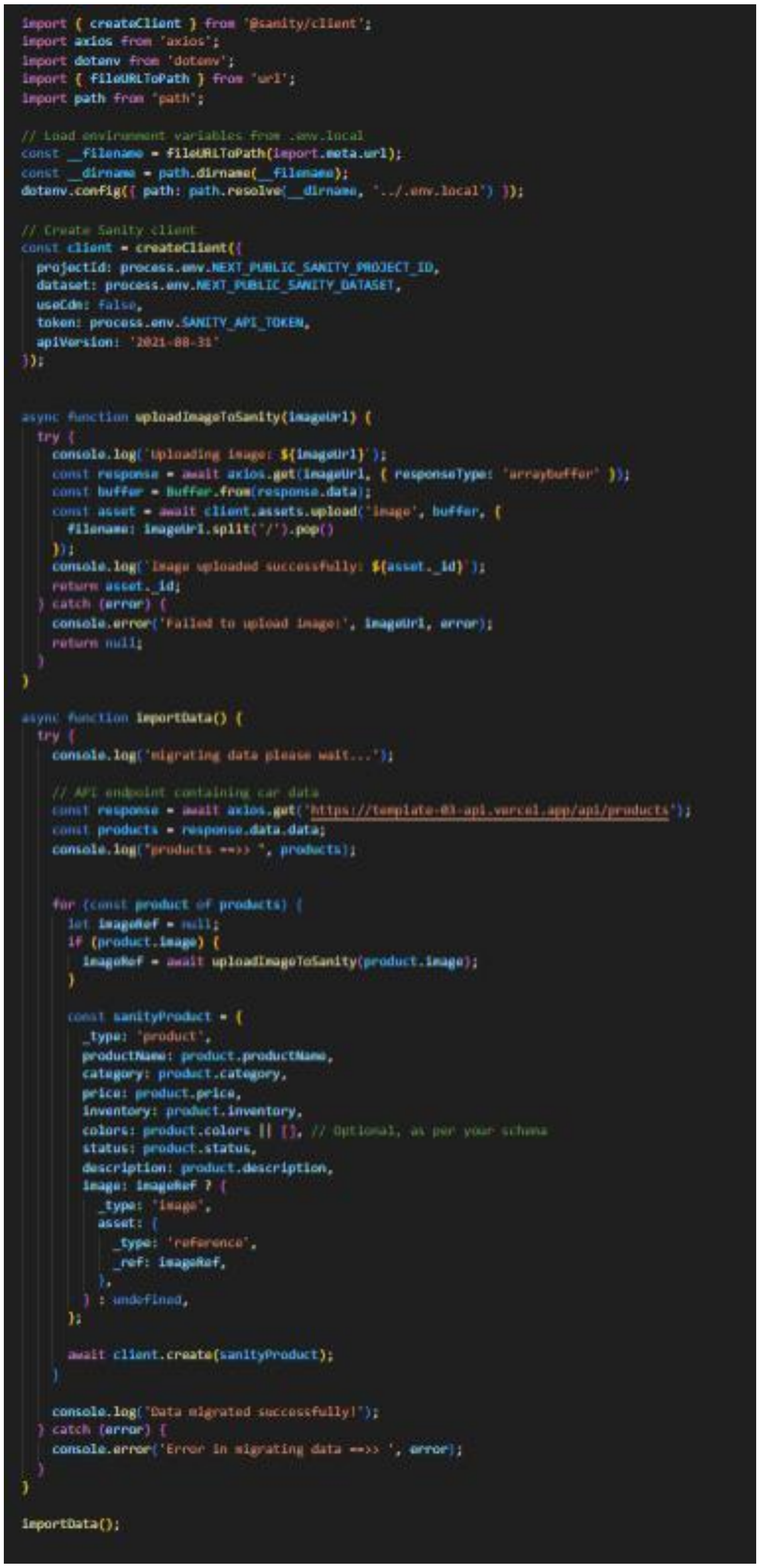
- ➢ url, path, and Buffer for file handling and environment setup.



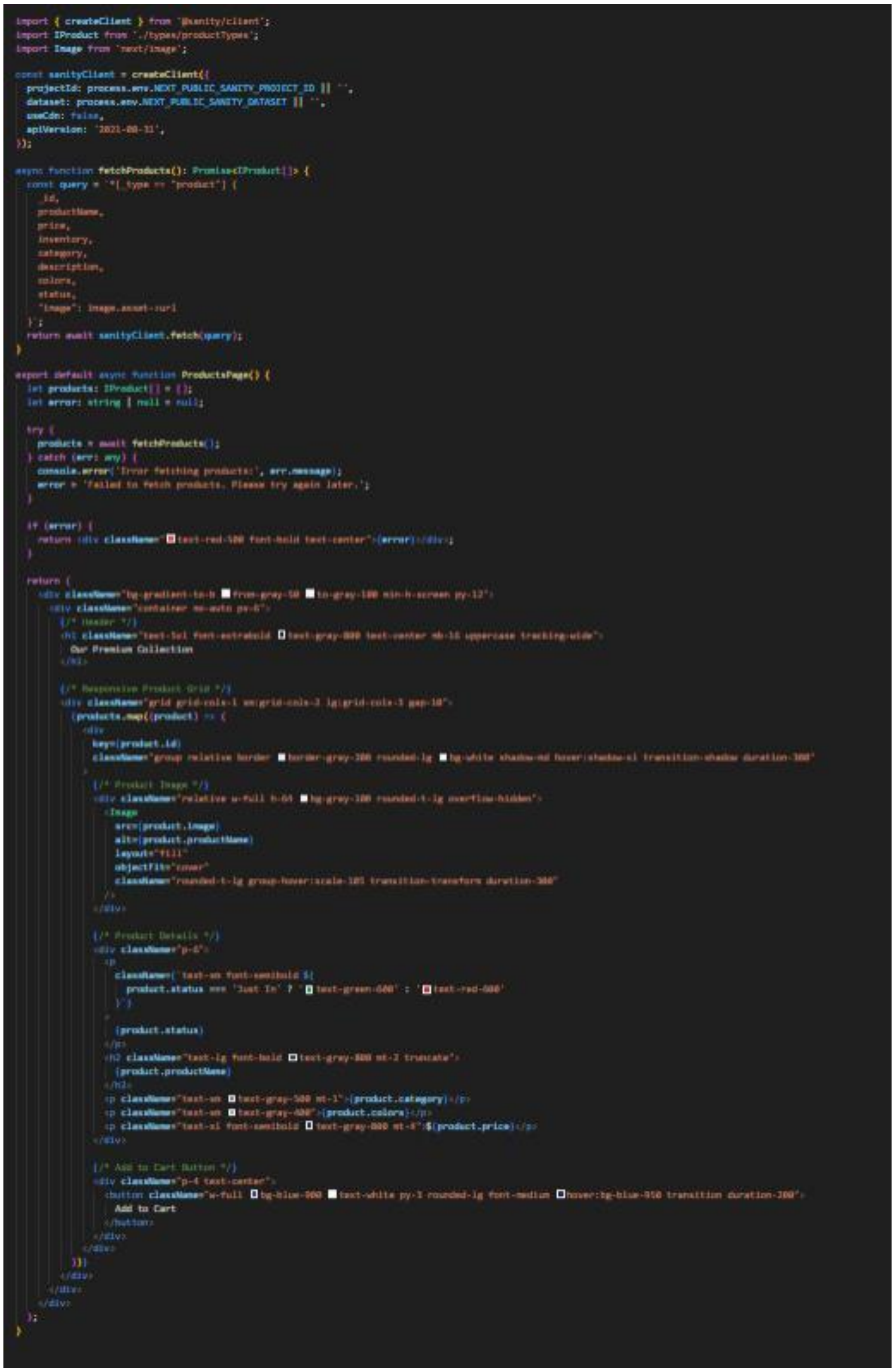
```
import { type SchemaTypeDefinition } from 'sanity'
import { allProducts } from './product'

export const schema: { types: SchemaTypeDefinition[] } = {
  types: [allProducts],
}
```



```
export default interface IProduct {
  id: string;
  productName: string;
  category: string;
  price: number;
  inventory: number;
  status: string;
  colors: string[],
  image: string;
  description: string;
}
```



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

// Load environment variables from .env.local
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config({ path: path.resolve(__dirname, '../.env.local') });

// Create Sanity client
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 {
    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('migrating data please wait...');

    // API endpoint containing car data
    const response = await axios.get('https://template-03-api.vercel.app/api/products');
    const products = response.data.data;
    console.log("products ==>> ", products);


    for (const product of products) {
      let imageRef = null;
      if (product.image) {
        imageRef = await uploadImageToSanity(product.image);
      }

      const sanityProduct = {
        _type: 'product',
        productName: product.productName,
        category: product.category,
        price: product.price,
        inventory: product.inventory,
        colors: product.colors || [], // Optional, as per your schema
        status: product.status,
        description: product.description,
        image: imageRef ? {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageRef,
          },
        } : undefined,
      };

      await client.create(sanityProduct);
    }

    console.log('Data migrated successfully!');
  } catch (error) {
    console.error('Error in migrating data ==>> ', error);
  }
}

importData();
```



```tsx
import { createClient } from '@sanity/client';
import IProduct from './types/productTypes';
import Image from 'next/image';

const sanityClient = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID || '',
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET || '',
  useCdn: false,
  apiVersion: '2021-08-31',
});

async function fetchProducts(): Promise<IProduct[]> {
  const query = `*[_type == "product"] {
    _id,
    productName,
    price,
    inventory,
    category,
    description,
    colors,
    status,
    "image": image.asset->url
  }`;
  return await sanityClient.fetch(query);
}

export default async function ProductsPage() {
  let products: IProduct[] = [];
  let error: string | null = null;

  try {
    products = await fetchProducts();
  } catch (err: any) {
    console.error('Error fetching products:', err.message);
    error = 'Failed to fetch products. Please try again later.';
  }

  if (error) {
    return <div className="text-red-500 font-bold text-center">{error}</div>;
  }

  return (
    <div className="bg-gradient-to-b from-gray-50 to-gray-100 min-h-screen py-12">
      <div className="container mx-auto px-6">
        {/* Header */}
        <h1 className="text-5xl font-extrabold text-gray-800 text-center mb-16 uppercase tracking-wide">
          Our Premium Collection
        </h1>

        {/* Responsive Product Grid */}
        <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-10">
          {products.map((product) => (
            <div
              key={product._id}
              className="group relative border border-gray-200 rounded-lg bg-white shadow-md hover:shadow-xl transition-shadow duration-300"
            >
              {/* Product Image */}
              <div className="relative w-full h-64 bg-gray-100 rounded-t-lg overflow-hidden">
                <Image
                  src={product.image}
                  alt={product.productName}
                  layout="fill"
                  objectFit="cover"
                  className="rounded-t-lg group-hover:scale-105 transition-transform duration-300"
                />
              </div>

              {/* Product Details */}
              <div className="p-6">
                <p
                  className={`text-sm font-semibold ${
                    product.status === 'Just In' ? 'text-green-600' : 'text-red-600'
                  }`}
                >
                  {product.status}
                </p>
                <h2 className="text-lg font-bold text-gray-800 mt-2 truncate">
                  {product.productName}
                </h2>
                <p className="text-sm text-gray-500 mt-1">{product.category}</p>
                <p className="text-sm text-gray-400">{product.colors}</p>
                <p className="text-xl font-semibold text-gray-800 mt-4">${product.price}</p>
              </div>

              {/* Add to Cart Button */}
              <div className="p-4 text-center">
                <button className="w-full bg-blue-900 text-white py-3 rounded-lg font-medium hover:bg-blue-950 transition duration-300">
                  Add to Cart
                </button>
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
}
```