

# Day 3 - API Integration Report - General E-Commerce

## API Integration Process

### 1. Overview of the API Integration

The API integration was intended to fetch product data from the Provided API and display it dynamically in the frontend. Key endpoints included `"/products"` for fetching product details. The integration made data synchronization between the API and the frontend components seamless.

### 2. Step-by-Step Integration

**Step 1:** select API endpoints.

**Step 2:** Setting up environment variables for API keys and other sensitive information.

**Step 3:** Implementing TypeScript interfaces to define the shape of data.

**Step 4:** Writing API calls using fetch.

**Step 5:** Error handling with retries and fallbacks.

**Step 6:** Connecting data to frontend

## Add Changes Made to Schemas

### Changes Made to Schemas

#### 1. Changes to Sanity Schema

- No modifications were performed on the sanity schema in regard to the API integration.

# Migration Steps and Tools Used

## 1. Tools Used for Migration

Describe all tools used within the migration steps

## 2. Migration

The migration utilized a given script to transfer as well as convert data when applicable. The Steps were:

**Step 1:** Run the script for migration based on the correct schema structure according to the specifications.

**Step 2:** Validate the output data to confirm successful transformation.

**Step 3:** Test the migrated data within the development environment.

**Step 4:** Deploy the migration script in the production environment.

**Step 5:** Verify the migration by comparing the old and new data sources.

The script provided above made the migration much easier and reduced manual intervention, thereby minimizing errors.

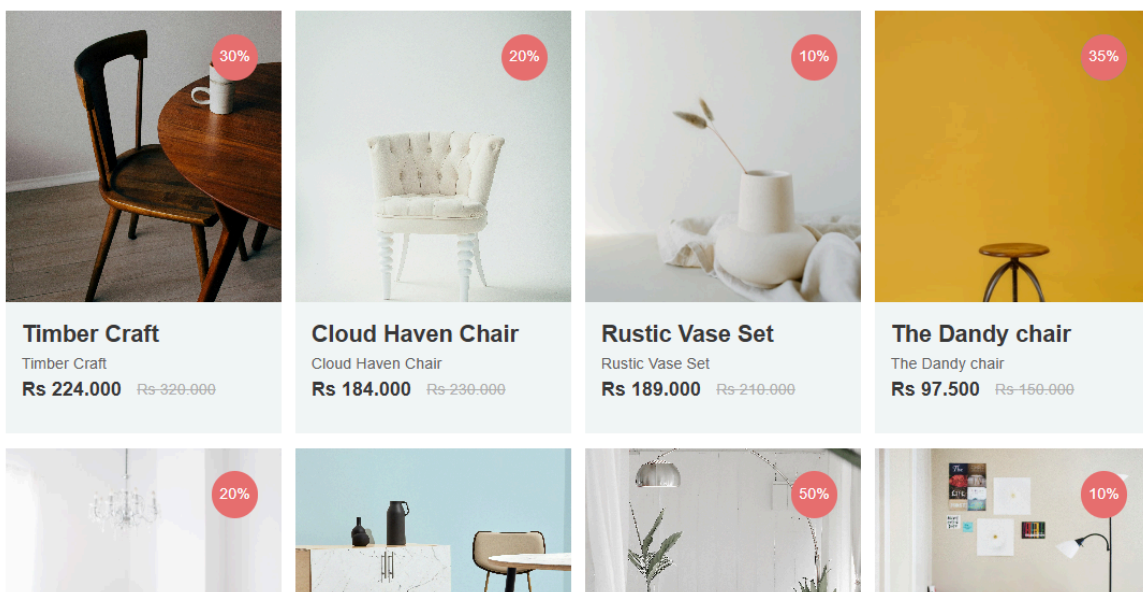
---

## Screenshots

### 1. API Calls

```
[...] 24 items
  0: {...} 12 properties
    isNew: false
    discountPercentage: 30
    tags: [...] 5 items
      0: wooden
      1: craftsmanship
      2: furniture
      3: modern
      4: nature inspired
    productImage: {...} 2 properties
      _type: image
      asset: {...} 1 property
        _ref: image-35a8872e48a8b88142ff8e96e0f3af21811f5b9f-3264x4928-j
        _rev: RXqg4VAYQ0632QeYmbtslw
      _type: product
      _id: RXqg4VAYQ0632QeYmbtslw
    title: Timber Craft
    _updatedAt: 2025-01-17T14:12:21Z
    price: 328
    _createdAt: 2025-01-17T14:12:21Z
    description: Introducing TimberCraft—a collection that celebrates the
      a touch of handcrafted elegance to any space. Perfect for those who v
      in the TimberCraft collection is meticulously crafted to highlight th
      beautifully designed pieces that blend rustic appeal with contemporar
      quality, craftsmanship, and the enduring beauty of wood. From strikin
      from high-quality, sustainable timber for durability and lasting appe
      highlights the authenticity of each piece Perfect for creating a cozy
      stand out in both style and substance.
  1: {...} 12 properties
  2: {...} 12 properties
```

## 2. Data Display in Frontend



## 3. Sanity CMS Fields

```

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     {
33       name: "tags",
34       type: "array",
35       title: "Tags",
36       of: [{ type: "string" }]
37     },
38     {
39       name: "dicountPercentage",
40       type: "number",
41       title: "Discount Percentage",
42     },
43     {
44       name: "isNew",
45       type: "boolean",
46       title: "New Badge",
47     }
48   ]
49 })

```

---

## Code Snippets

### 1. API Integration Code

Provide snippets of:

- Fetching data from the API.
- Parsing and validating API responses.
- Handling errors and retries.

```

"use client";
import React, { createContext, useContext, useState, useEffect } from "react";
import { client } from "../sanity/lib/client";

const ProductContext = createContext<any[]>([]);

export const ProductProvider: React.FC<{ children: React.ReactNode }> = ({
  children,
}) => {
  const [products, setProducts] = useState<any[]>([]);

  useEffect(() => {
    const fetchProducts = async () => {
      try {
        const res =
          await client.fetch(`*[_type == "product"] {
            title,
            description,
            "imageUrl": productImage.asset->url,
            price,
            tags,
            dicountPercentage,
            "discountedPrice": price - (price * dicountPercentage / 100),
            isNew,
            "slug": slug.current
          }`);
        setProducts(res);
      } catch (error) {
        console.error("Error fetching products:", error);
      }
    };

    fetchProducts();
  }, []);

  return (
    <ProductContext.Provider value={products}>
      {children}
    </ProductContext.Provider>
  );
};

```

## 2. Migration Script Code

```

importData.js > client > projectId
1  import { createClient } from '@sanity/client';
2
3  const client = createClient({
4    projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
5    dataset: 'production',
6    useCdn: true,
7    apiVersion: '2025-01-13',
8    token: process.env.Token,
9  });
10
    Tabnine | Edit | Test | Explain | Document
11  async function uploadImageToSanity(imageUrl) {
12    try {
13      console.log(`Uploading image: ${imageUrl}`);
14
15      const response = await fetch(imageUrl);
16      if (!response.ok) {
17        throw new Error(`Failed to fetch image: ${imageUrl}`);
18      }
19
20      const buffer = await response.arrayBuffer();
21      const bufferImage = Buffer.from(buffer);
22
23      const asset = await client.assets.upload('image', bufferImage, {
24        filename: imageUrl.split('/').pop(),
25      });
26
27      console.log(`Image uploaded successfully: ${asset._id}`);
28      return asset._id;
29    } catch (error) {
30      console.error('Failed to upload image:', imageUrl, error);
31      return null;
32    }
33  }
34
    Tabnine | Edit | Test | Explain | Document
35  async function uploadProduct(product) {

```