API INTEGRATION AND DATA MIGRATION

Introduction

This report outlines the steps taken to integrate APIs into the Furniture Marketplace using Next.js and migrate data into Sanity CMS as part of **Day 3** of the hackathon. The main objectives were to:

- Integrate external data using APIs.
- Adjust the existing schema to align with the incoming data.
- Migrate data into Sanity CMS.
- Implement error handling and ensure seamless data flow.

Step 1: Understand the Provided API

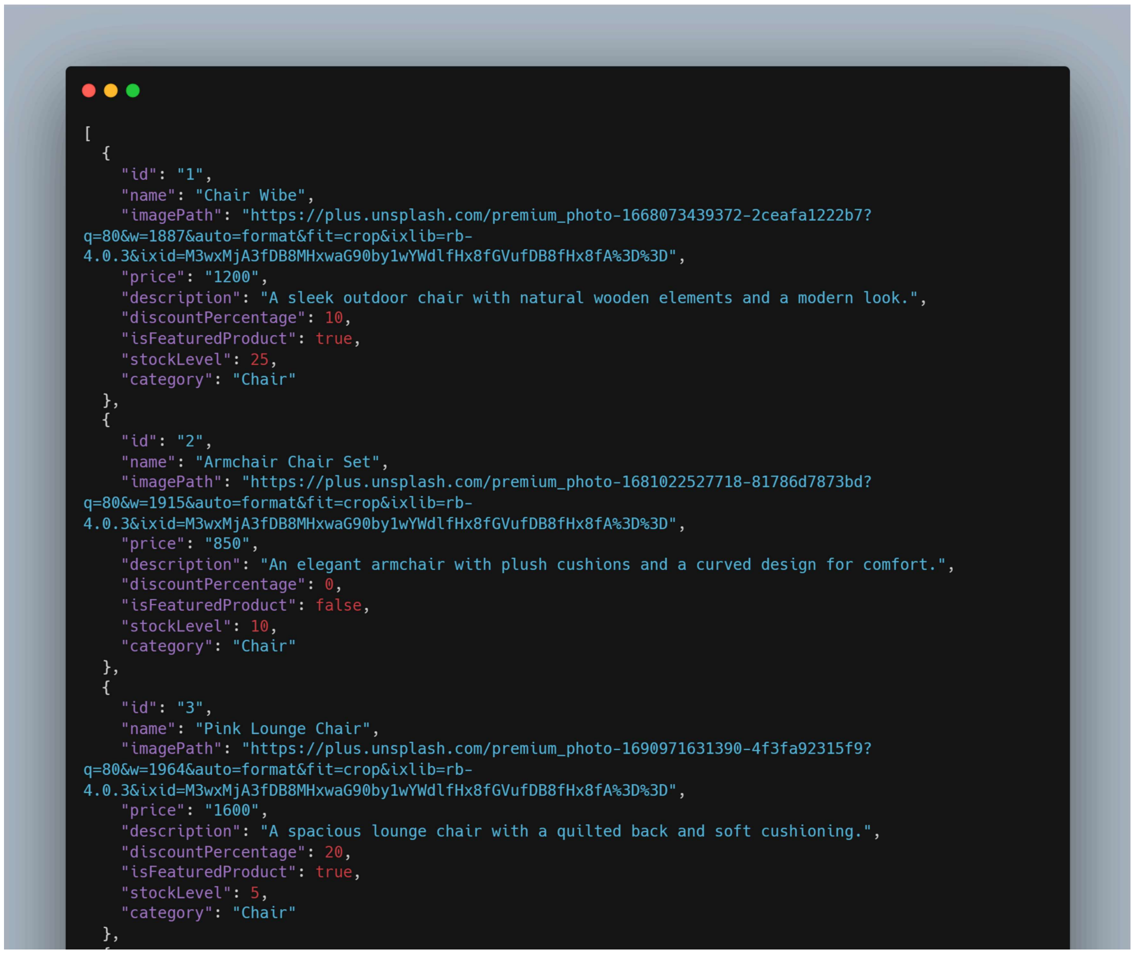
- **Objective**: The provided API, available at https://template-0-beta.vercel.app/api/product, returns a collection of products in JSON format. This includes details like product `id`, `name`, `description`, `price`, `stockLevel`, `category`, and `imagePath`. It is crucial to understand the structure of the data being returned, as we need to map it into a new schema for migration into Sanity CMS.
- **Process:** To begin, we reviewed the API documentation to familiarize ourselves with the available endpoints. The key endpoint identified for this project is:

  Product Listings: `/products`

  This endpoint provides essential product data, including:

  - Names(Titles)
  - Descriptions
  - Prices
  - Images
  - Is Featured Product
  - Category
  - Stock Level
  - Discount percentage

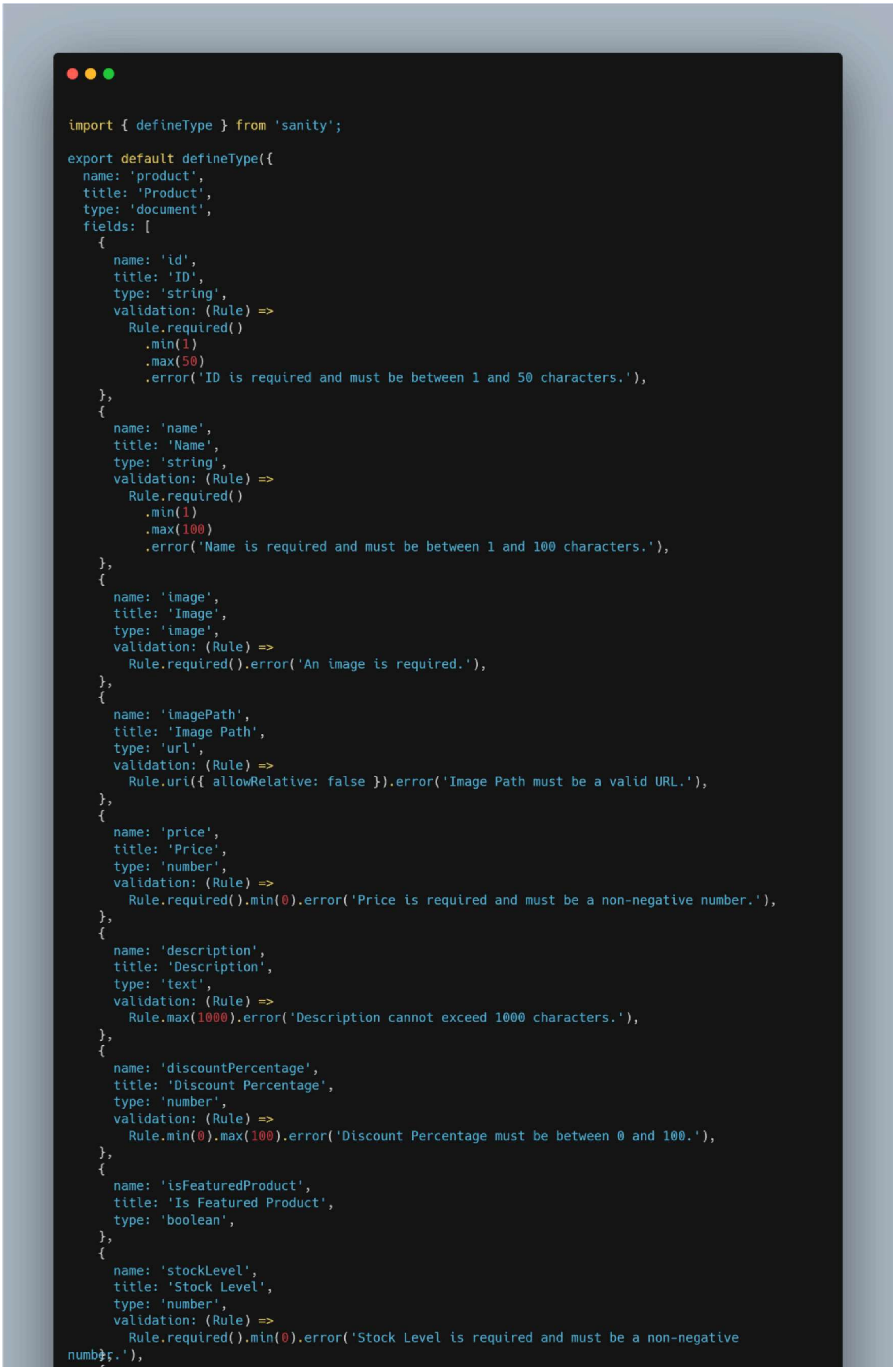
The JSON Format of API



```
[
  {
    "id": "1",
    "name": "Chair Wibe",
    "imagePath": "https://plus.unsplash.com/premium_photo-1668073439372-2ceafa1222b7?
q=80&w=1887&auto=format&fit=crop&ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D",
    "price": "1200",
    "description": "A sleek outdoor chair with natural wooden elements and a modern look.",
    "discountPercentage": 10,
    "isFeaturedProduct": true,
    "stockLevel": 25,
    "category": "Chair"
  },
  {
    "id": "2",
    "name": "Armchair Chair Set",
    "imagePath": "https://plus.unsplash.com/premium_photo-1681022527718-81786d7873bd?
q=80&w=1915&auto=format&fit=crop&ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D",
    "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": "Pink Lounge Chair",
    "imagePath": "https://plus.unsplash.com/premium_photo-1690971631390-4f3fa92315f9?
q=80&w=1964&auto=format&fit=crop&ixlib=rb-
4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D",
    "price": "1600",
    "description": "A spacious lounge chair with a quilted back and soft cushioning.",
    "discountPercentage": 20,
    "isFeaturedProduct": true,
    "stockLevel": 5,
    "category": "Chair"
  },
```

Step 2: Make Changes to the Sanity Schema

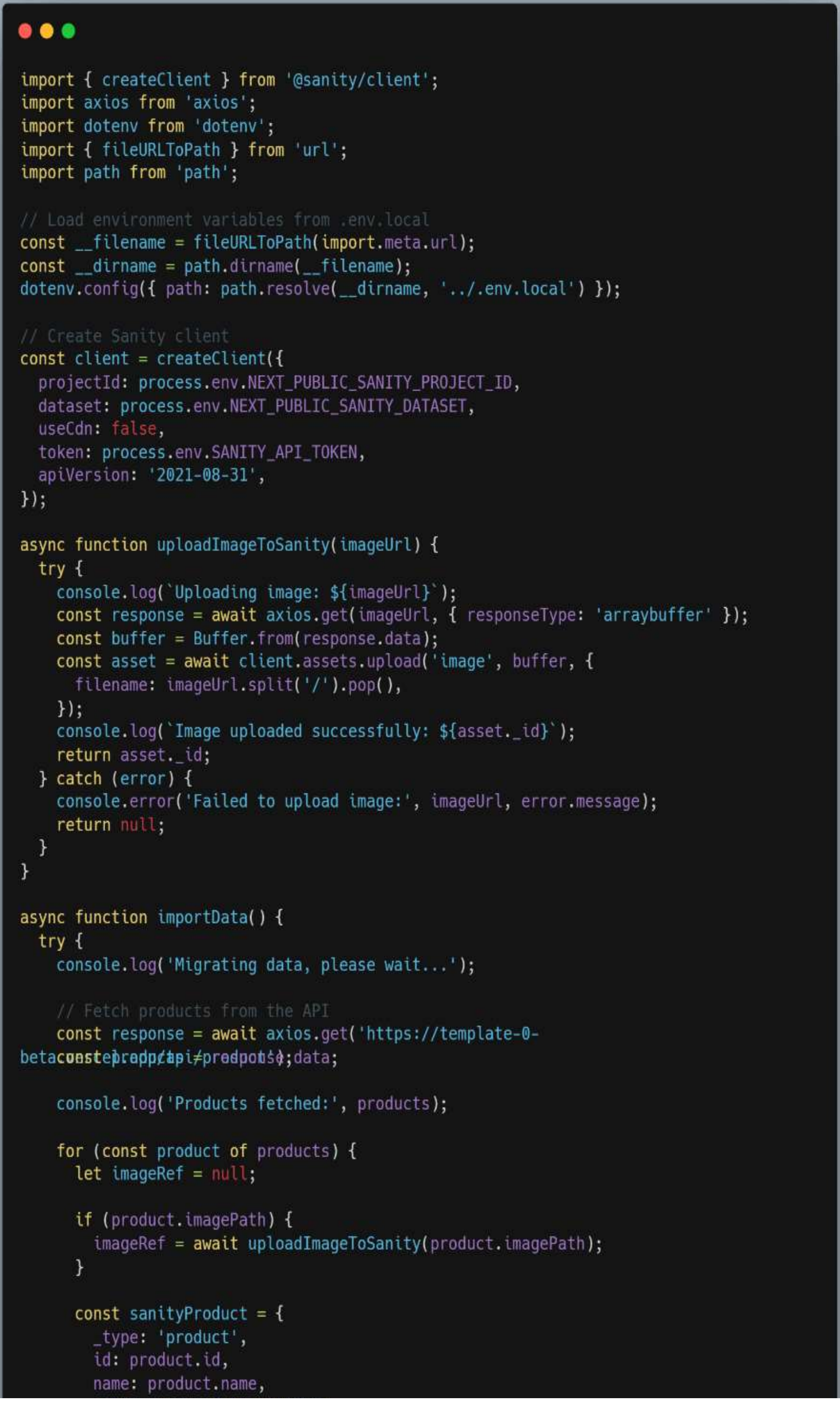
- **Objective**: The provided schema needed to be modified to store the incoming product data into Sanity CMS. The Sanity schema defines how your data should be structured inside Sanity.
- **Process**:
  - A schema called `product,` focusing on the validations was created in Sanity with the following fields:
    - `id`: String (required, with length constraints).
    - `name`: String (required, with length constraints).
    - `image`: Image (required).
    - `imagePath`: URL (required, for the image URL).
    - `price`: Number (required, must be a non-negative value).
    - `description`: Text (up to 1000 characters).
    - `discountPercentage`: Number (range from 0 to 100).
    - `isFeaturedProduct`: Boolean (to mark featured products).
    - `stockLevel`: Number (required, must be a non-negative value).
    - `category`: String (required, with length constraints).
- **Outcome**: The Sanity schema was tailored to store product information efficiently and in a structured manner.



```
import { defineType } from 'sanity';

export default defineType({
  name: 'product',
  title: 'Product',
  type: 'document',
  fields: [
    {
      name: 'id',
      title: 'ID',
      type: 'string',
      validation: (Rule) =>
        Rule.required()
          .min(1)
          .max(50)
          .error('ID is required and must be between 1 and 50 characters.'),
    },
    {
      name: 'name',
      title: 'Name',
      type: 'string',
      validation: (Rule) =>
        Rule.required()
          .min(1)
          .max(100)
          .error('Name is required and must be between 1 and 100 characters.'),
    },
    {
      name: 'image',
      title: 'Image',
      type: 'image',
      validation: (Rule) =>
        Rule.required().error('An image is required.'),
    },
    {
      name: 'imagePath',
      title: 'Image Path',
      type: 'url',
      validation: (Rule) =>
        Rule.uri({ allowRelative: false }).error('Image Path must be a valid URL.'),
    },
    {
      name: 'price',
      title: 'Price',
      type: 'number',
      validation: (Rule) =>
        Rule.required().min(0).error('Price is required and must be a non-negative number.'),
    },
    {
      name: 'description',
      title: 'Description',
      type: 'text',
      validation: (Rule) =>
        Rule.max(1000).error('Description cannot exceed 1000 characters.'),
    },
    {
      name: 'discountPercentage',
      title: 'Discount Percentage',
      type: 'number',
      validation: (Rule) =>
        Rule.min(0).max(100).error('Discount Percentage must be between 0 and 100.'),
    },
    {
      name: 'isFeaturedProduct',
      title: 'Is Featured Product',
      type: 'boolean',
    },
    {
      name: 'stockLevel',
      title: 'Stock Level',
      type: 'number',
      validation: (Rule) =>
        Rule.required().min(0).error('Stock Level is required and must be a non-negative
number.'),
    },
```

Step 3: Handle Data Migration

- **Objective**: To move the product data from the provided API into the Sanity CMS, a data migration strategy was needed. This was handled using a custom script (`data-migration.mjs`).
- **Process**:
    - The migration script performed the following tasks:
        1. **Fetching Product Data**: Using `axios`, the script fetched the product data from the API.
        2. **Uploading Images**: For each product, it checked if an image was provided (via the `imagePath` field) and uploaded it to Sanity using the `client.assets.upload` method. This created a reference to the image asset.
        3. **Creating Sanity Products**: After obtaining all necessary product details, the script used the Sanity client to create new documents in the Sanity database with the fetched data.
- **Outcome**: The data migration was successfully completed, and all the products were migrated into the Sanity CMS.



```
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.message);
    return null;
  }
}

async function importData() {
  try {
    console.log('Migrating data, please wait...');

    // Fetch products from the API
    const response = await axios.get('https://template-0-
beta.vercel.app/api/products');
    const products = response.data;

    console.log('Products fetched:', products);

    for (const product of products) {
      let imageRef = null;

      if (product.imagePath) {
        imageRef = await uploadImageToSanity(product.imagePath);
      }

      const sanityProduct = {
        _type: 'product',
        id: product.id,
        name: product.name,
```

Step 4:

Objective: To set up a seamless integration between the backend API and the frontend Next.js application, ensuring accurate data migration to Sanity CMS and successful display of product data on a responsive UI**.**

Backend Setup:

1. **API Integration**:
   - I integrated a product data API available at: `https://template-0-beta.vercel.app/api/product`.
   - A `product` document schema was created in **Sanity CMS** with the following fields:
     - **ID**: Unique identifier for the product.
     - **Name**: Product name.
     - **Image**: Product image URL.
     - **Price**: Price of the product.
     - **Description**: Description of the product.
     - **Discount Percentage**: Any discount applied to the product.
     - **Stock Level**: Indicates product availability.
     - **Category**: Specifies the product type or category.
2. **Data Migration**:
   - A migration script (`data-migration.mjs`) was developed to:
     - Fetch product data from the API.
     - Upload product images to **Sanity CMS**.
     - Save the fetched product data into the Sanity dataset using the `create` method.
3. **Outcome**:
   - The backend was set up successfully, and all product data was migrated to **Sanity CMS** as intended.

Frontend Setup:

1. **Framework**:

   I used **Next.js**, a React-based framework, for the frontend development.

2. **API Data Fetching**:

The `axios` library was used to make an HTTP GET request to fetch product data from the API.
Example code:
```
import axios from 'axios';

const fetchProducts = async () => {
    try {
        const response = await
axios.get('https://template-0-beta.vercel.app/api/product');
        console.log(response.data);
        return response.data;
    } catch (error) {
        console.error('Error fetching product data:', error);
    }
};
```

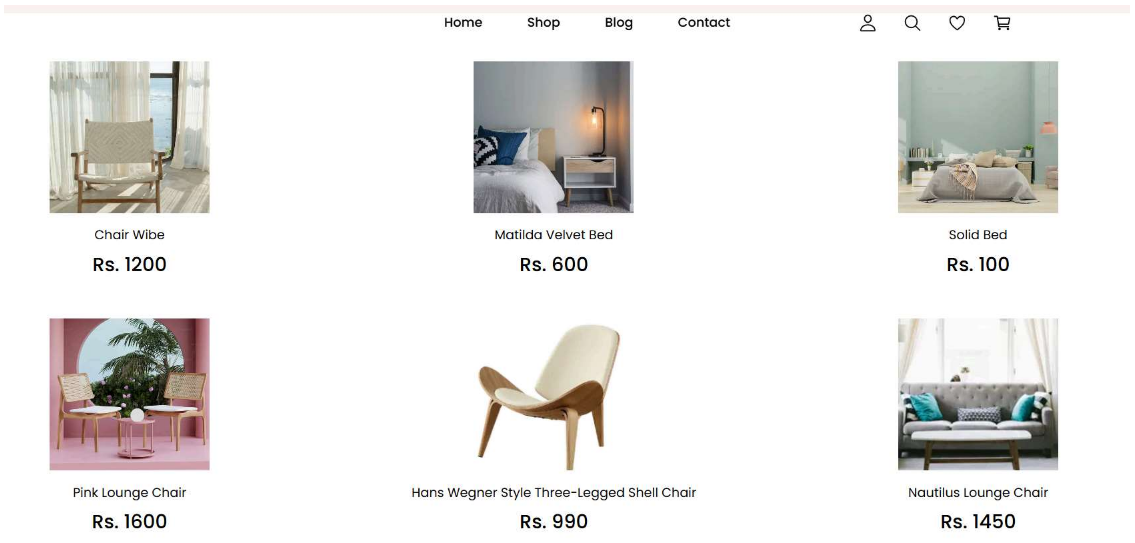
3. **Display Data on UI**:

Fetched data was displayed on the frontend using a React component. Example:

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

const ProductList = () => {
    const [products, setProducts] = useState([]);

    useEffect(() => {
        const fetchProducts = async () => {
            try {
                const response = await
axios.get('https://template-0-beta.vercel.app/api/product');
                setProducts(response.data);
            } catch (error) {
                console.error('Error fetching products:', error);
            }
        };

        fetchProducts();
    }, []);
    return (
        <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3
gap-4">
            {products.map((product) => (
                <div key={product.id} className="border p-4 rounded-lg
shadow-md">
                    <img src={product.image} alt={product.name}
className="w-full h-48 object-cover" />
                    <h3 className="text-lg
font-semibold">{product.name}</h3>
                    <p>{product.description}</p>
                    <p className="text-green-500
font-bold">${product.price}</p>
                </div>
            ))}
        </div>
    );
};

export default ProductList;
```

4. **Outcome**:
    - The product data fetched from the API was successfully displayed in a **responsive grid layout**. The layout ensured a 3-column display on large screens and adjusted for small/medium screens.

Data Successfully Displayed on the UI:



Chair Wibe
Rs. 1200

Matilda Velvet Bed
Rs. 600

Solid Bed
Rs. 100

Pink Lounge Chair
Rs. 1600

Hans Wegner Style Three-Legged Shell Chair
Rs. 990

Nautilus Lounge Chair
Rs. 1450

Key Observations:

- Backend and frontend setups were both robust and well-coordinated.
- The use of **Sanity CMS** for backend data storage and **Next.js** for frontend ensured a smooth integration.
- Responsive UI with proper grid layout improved the overall presentation.

Self Validation Checklist:

Checklist	Status
API Understanding	☑ ✅
Schema Validation	☑ ✅
Data Migration	☑ ✅
API Integration in Next J.S	☑ ✅
Submission Preparation	☑ ✅