Hackathon Day 3

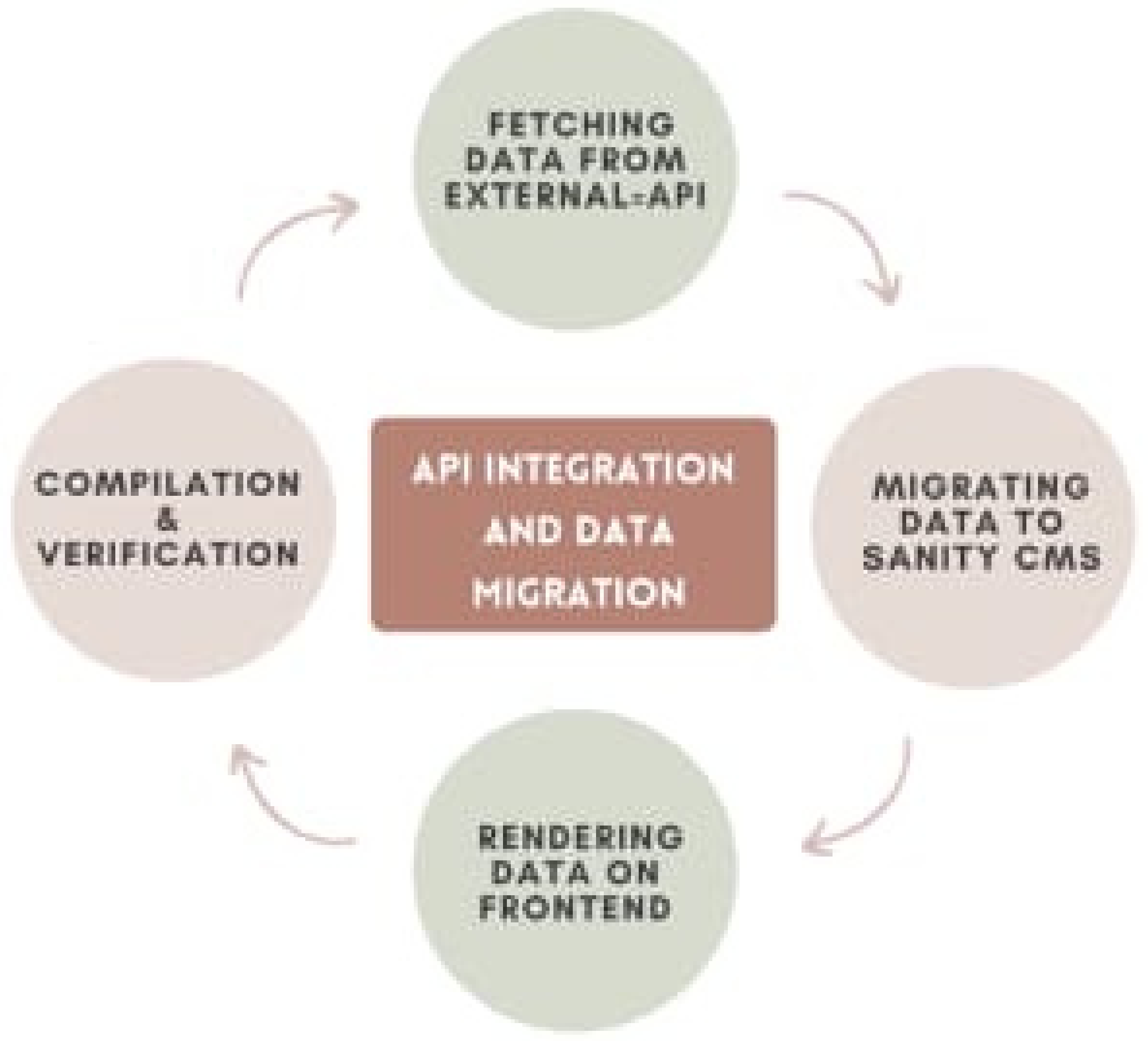
API INTEGRATION AND DATA MIGRATION:

1. Introduction

This report outlines the integration process of product and category data from an external API into the backend system. A clothing e-commerce plateform. The integration leverages sanity CMS for content management and Next.js for frontend rendering.

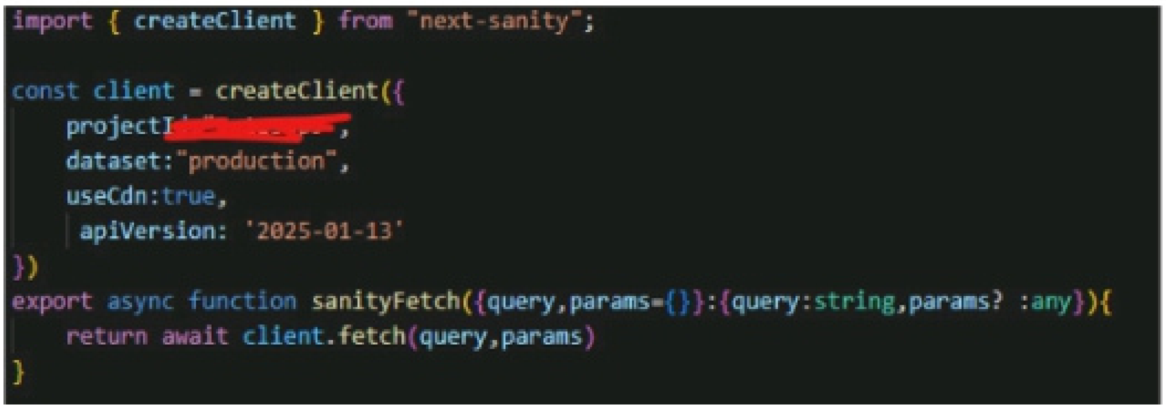
The main objectives of this integration were:

- **Fetching data from an external API.**
- **Storing and managing the data in sanity CMS.**
- **Displaying the fetched data dynamically on the frontend.**


API INTEGRATION AND DATA MIGRATION
FETCHING DATA FROM EXTERNAL-API
MIGRATING DATA TO SANITY CMS
RENDERING DATA ON FRONTEND
COMPILATION & VERIFICATION

2 . Fetching Data in the Frontend:

- set up a function in the fetch folder to fetch data from sanity.
- fetch folder picture:



```
import { createClient } from "next-sanity";

const client = createClient({
    projectI[redacted],
    dataset:"production",
    useCdn:true,
     apiVersion: '2025-01-13'
})
export async function sanityFetch({query,params={}}:{query:string,params? :any}){
    return await client.fetch(query,params)
}
```

Adjustments Made to Schemas:

- Included image and description fields in the product schema to handle additional API data.
- Changed price field from string to number for consistency.

3 . API Integration Process:

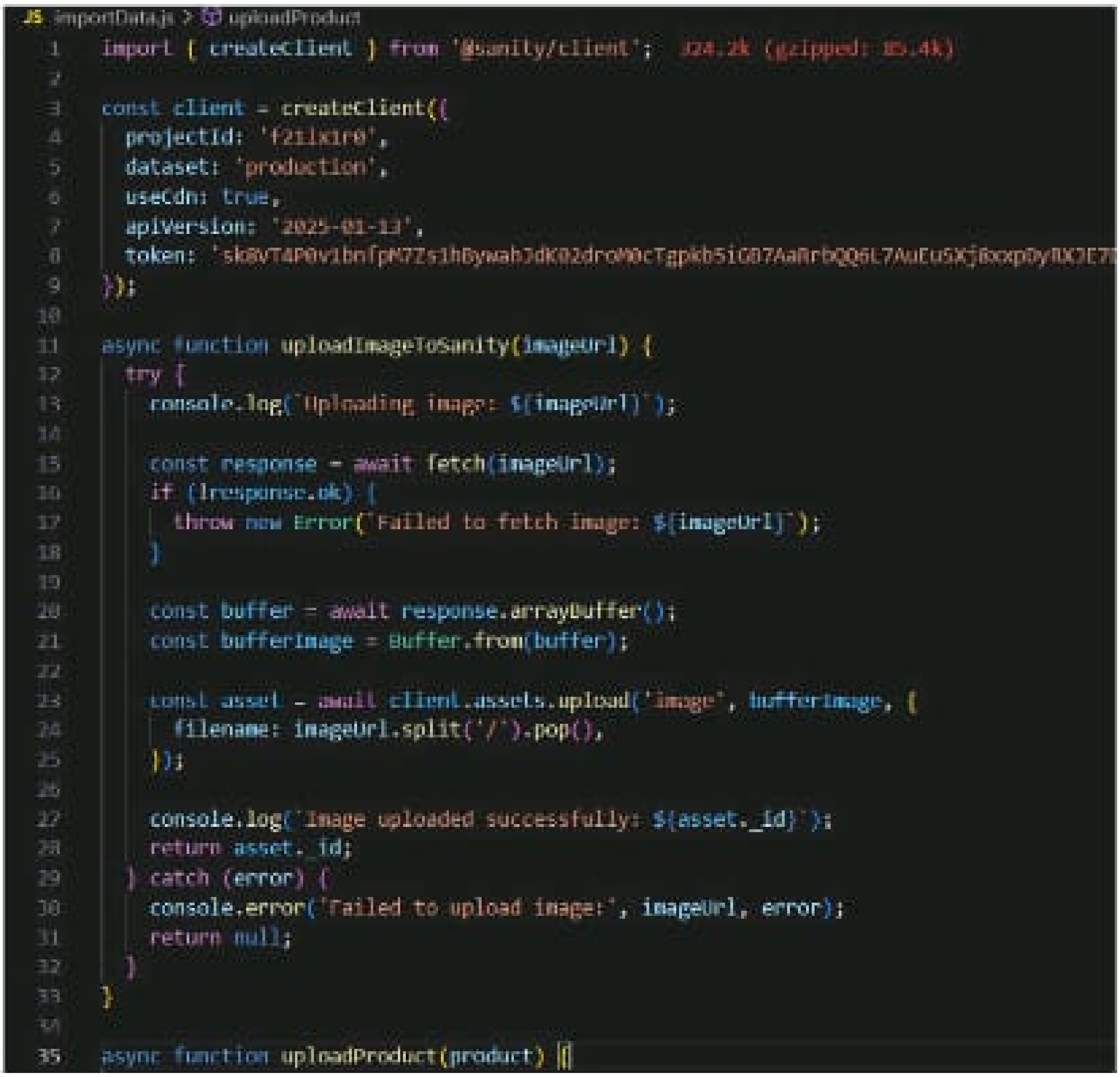
Fetching dat from API

The API provided the following endpoints:

- **Products Endpoint: Includes details such as titles, prices, description, categories, inventory and images.**

screenshot:

Included the following screenshots:



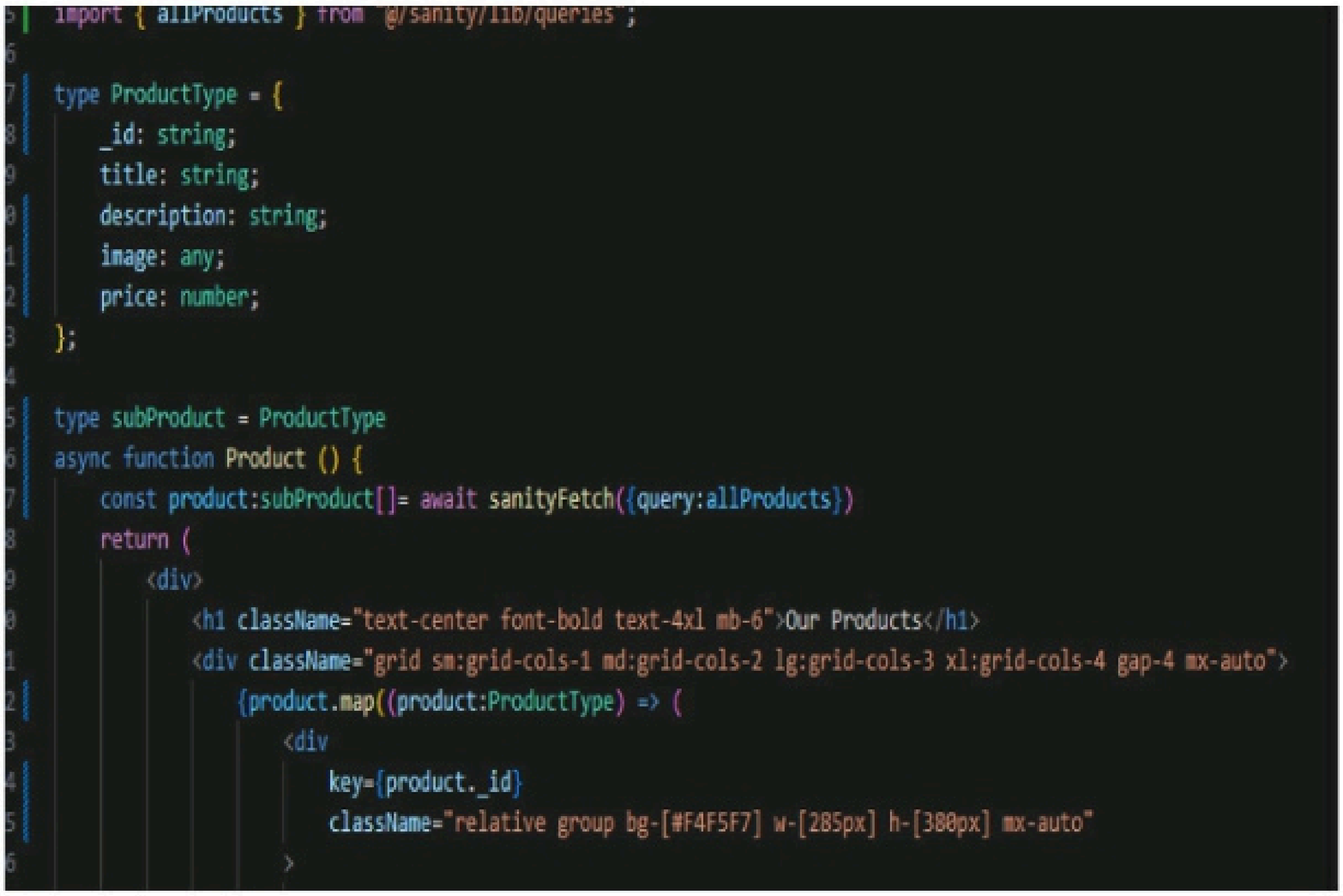
```
JS importData.js > uploadProduct
import { createClient } from '@sanity/client';  324.2k (gzipped: 85.4k)

const client = createClient({
  projectId: 'f21lx1r0',
  dataset: 'production',
  useCdn: true,
  apiVersion: '2025-01-13',
  token: 'sk0VT4P0v1bnfpM7Zs1hBywahJdK02droM0cTgpkb5iG87AaRrbQQ6L7AuEu5Xj6xxpDyRXJE7
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);

    const response = await fetch(imageUrl);
    if (!response.ok) {
      throw new Error(`Failed to fetch image: ${imageUrl}`);
    }

    const buffer = await response.arrayBuffer();
    const bufferImage = Buffer.from(buffer);

    const asset = await client.assets.upload('image', bufferImage, {
      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 uploadProduct(product) {
```

sanity Dashboard Screenshot Where Our Products is stored:

Product

Classic Polo Shirt
Black Athletic Jogger Pants with ...
Skinny Fit Jeans
Beige Slim Fit Jogger Pants
Black Printed T-Shirt
Checkered Shirt
LOOSE FIT BERMUDA SHORTS
Vertical Striped Shirt
Orange Stripe T-Shirt
COURAGE GRAPHIC T-SHIRT
Gray Slim Fit Jogger Pants

API Calls:



```
import { allProducts } from "@/sanity/lib/queries";

type ProductType = {
    _id: string;
    title: string;
    description: string;
    image: any;
    price: number;
};

type subProduct = ProductType
async function Product () {
    const product:subProduct[]= await sanityFetch({query:allProducts})
    return (
        <div>
            <h1 className="text-center font-bold text-4xl mb-6">Our Products</h1>
            <div className="grid sm:grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-4 mx-auto">
                {product.map((product:ProductType) => (
                    <div
                        key={product._id}
                        className="relative group bg-[#F4F5F7] w-[285px] h-[380px] mx-auto"
                    >
```

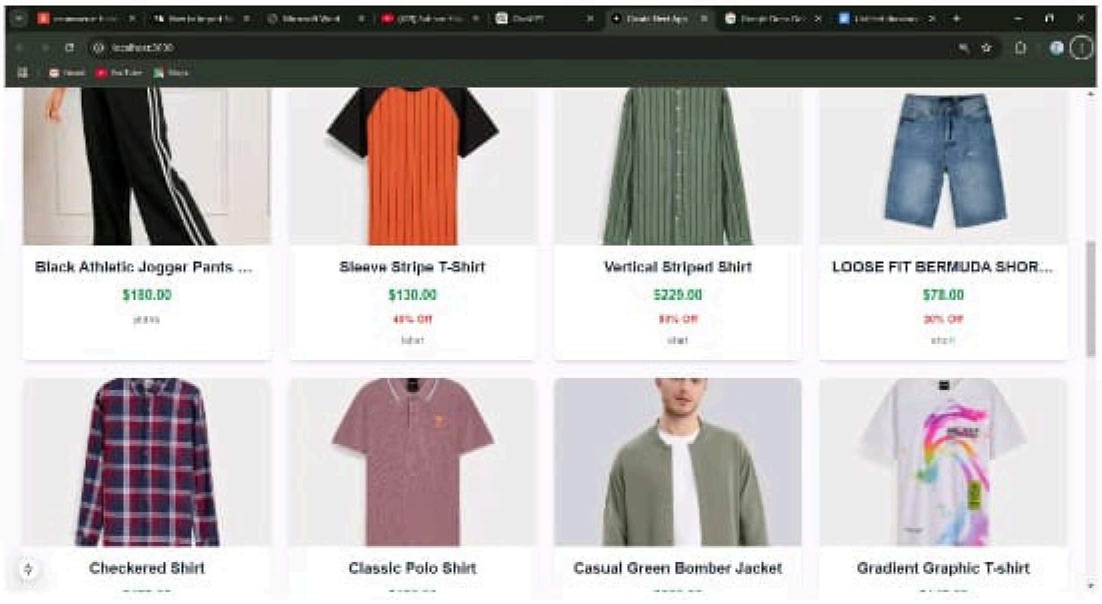
4. Frontend Integration

Rendering Data on Frontend

The next step was to fetch the stored data from sanity CMS and display it on the shop.co frontend. Using next.js, the fetching data was dynamically rendered into the product listing UI.

Display Data on the UI

Finally, I used Next.js, to create a UI that dynamically rendered the fetched project data. Each product was displayed in a card layout with the name, price, description and image.


Black Athletic Jogger Pants ...
$180.00
Sleeve Stripe T-Shirt
$130.00
Vertical Striped Shirt
$229.00
LOOSE FIT BERMUDA SHOR...
$78.00
Checkered Shirt
Classic Polo Shirt
Casual Green Bomber Jacket
Gradient Graphic T-shirt