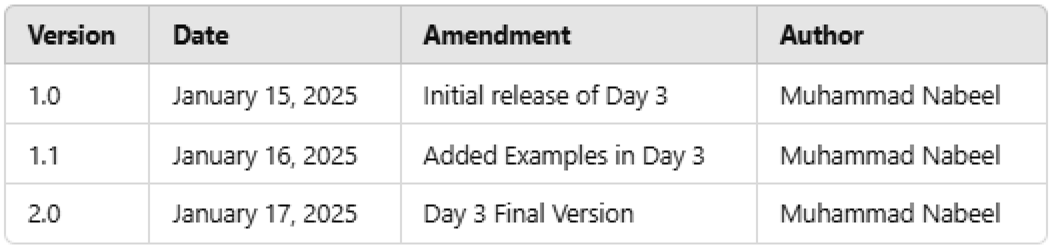
Prepared by Muhammad Nabeel

DAY 3 - API INTEGRATION AND DATA MIGRATION

Published: January 18, 2025

Document Revision Information



Version	Date	Amendment	Author
1.0	January 15, 2025	Initial release of Day 3	Muhammad Nabeel
1.1	January 16, 2025	Added Examples in Day 3	Muhammad Nabeel
2.0	January 17, 2025	Day 3 Final Version	Muhammad Nabeel

Objective

The focus of Day 3 is to integrate APIs and migrate data into Sanity CMS to build a functional marketplace backend. This exercise aims to replicate real-world practices and help participants handle diverse client requirements effectively.

Key Learning Outcomes

1. Understand how to integrate APIs into a Next.js project.
2. Learn to migrate data into Sanity CMS.
3. Develop skills to validate schemas for seamless API integration.
4. Implement best practices in API error handling and schema adjustments.

Steps for Day 3

API Overview

Provided APIs

Below are references for the APIs for various templates. Use these APIs to populate your Sanity CMS, or alternatively, import data manually using JSON or CSV.

Template References:

- **Template 1:** API Documentation
    - **Schema**: Product Schema
    - **Migration Script**: importData.js
- **Template 2:** Hackathon API
    - **Schema**: Category, Product
- **Template 3:** API
    - **Schema**: Sanity Schema

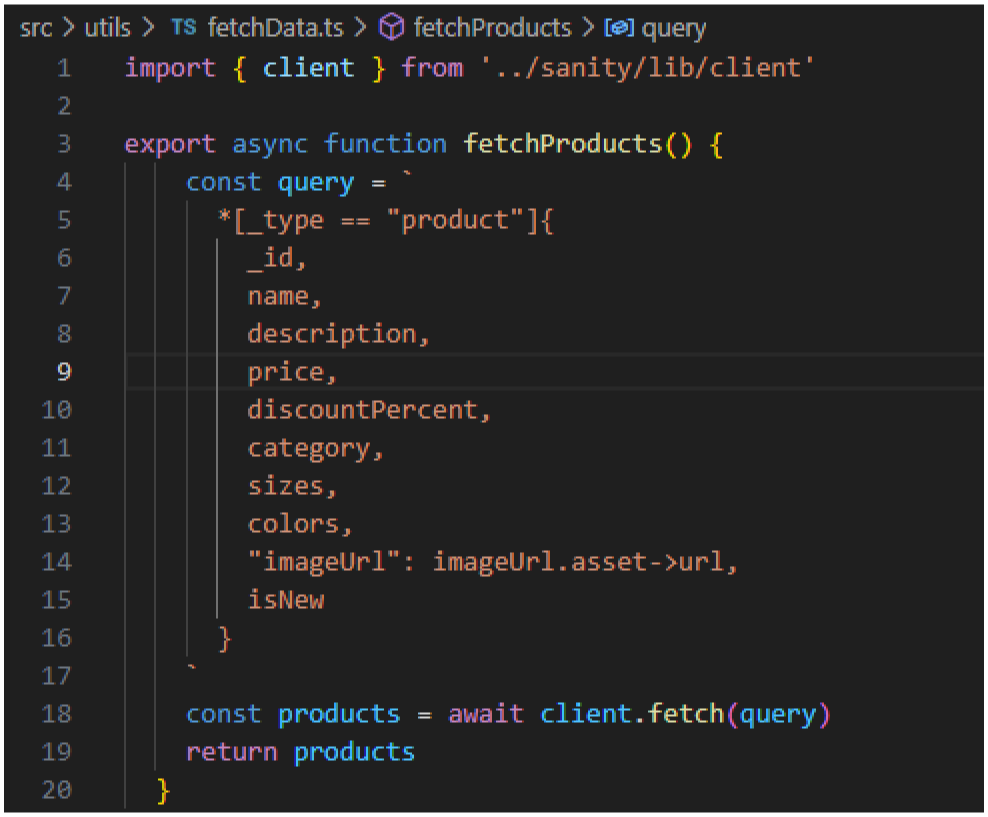
Implementation Details

API Integration Process

Step 1: API Endpoint Setup

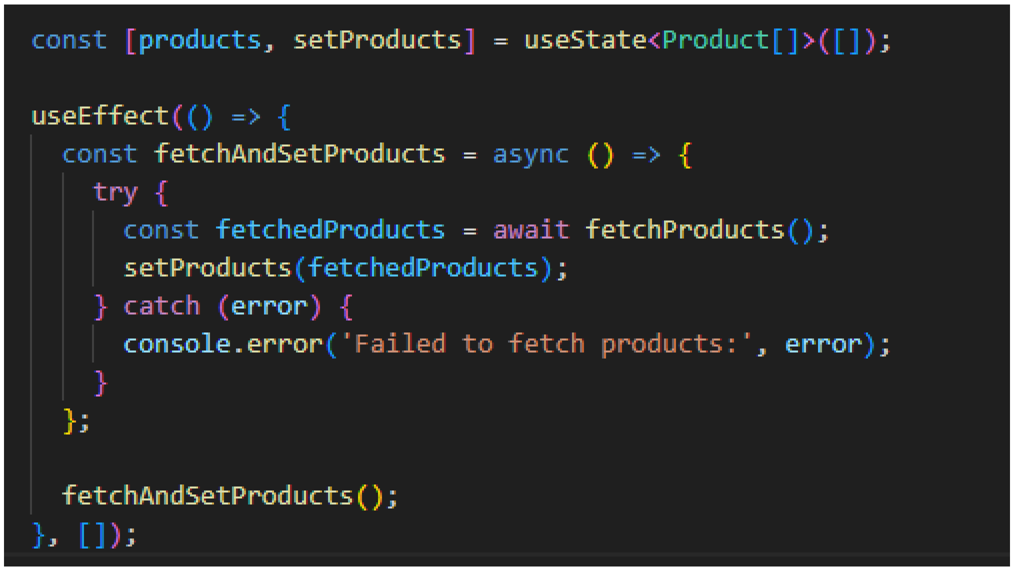
- **API Endpoint:** `https://template1-neon-nu.vercel.app/api/products`
- **Data Structure:**
    - Fields: `productName`.
- **Library Used:** Axios was used for making HTTP requests to fetch data from the API.

Code Snippet for API Call:



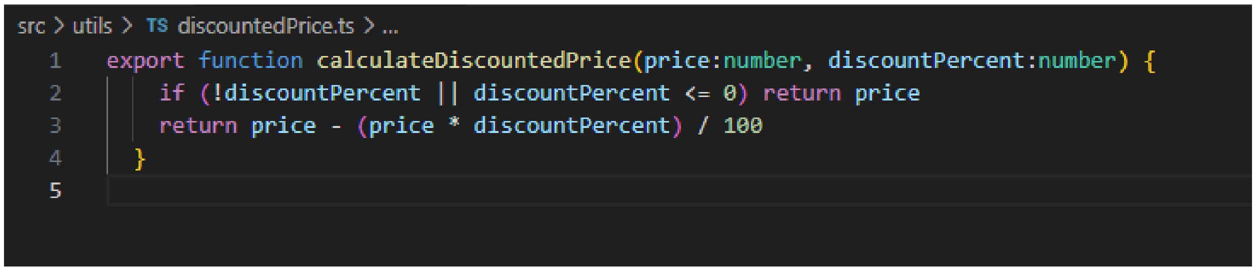
src > utils > TS fetchData.ts > fetchProducts > query

```ts
import { client } from '../sanity/lib/client'

export async function fetchProducts() {
    const query = `
      *[_type == "product"]{
        _id,
        name,
        description,
        price,
        discountPercent,
        category,
        sizes,
        colors,
        "imageUrl": imageUrl.asset->url,
        isNew
      }
    `
    const products = await client.fetch(query)
    return products
  }
```



```ts
const [products, setProducts] = useState<Product[]>([]);

useEffect(() => {
  const fetchAndSetProducts = async () => {
    try {
      const fetchedProducts = await fetchProducts();
      setProducts(fetchedProducts);
    } catch (error) {
      console.error('Failed to fetch products:', error);
    }
  };

  fetchAndSetProducts();
}, []);
```

As Discounted price is not given so, I used the utility function to find the Discounted Price.



```
src > utils > TS discountedPrice.ts > ...
export function calculateDiscountedPrice(price:number, discountPercent:number) {
    if (!discountPercent || discountPercent <= 0) return price
    return price - (price * discountPercent) / 100
  }
```

Schema Adjustments

Field Updates:

- `colors`: Added as an optional array of strings to accommodate multiple color options.
- `image`: Configured as a Sanity image field with hotspot enabled for better cropping and scaling.

```
// schemas/product.ts
export default {
    name: 'product',
    title: 'Product',
    type: 'document',
    fields: [
      {
        name: 'name',
        title: 'Product Name',
        type: 'string',
      },
      {
        name: 'description',
        title: 'Description',
        type: 'text',
      },
      {
        name: 'price',
        title: 'Price',
        type: 'number',
      },
      {
        name: 'discountPercent',
        title: 'Discount Percentage',
        type: 'number',
      },
      {
        name: 'category',
        title: 'Category',
        type: 'string',
        options: {
          list: ['hoodie', 'tshirt', 'jeans', 'shirt', 'short'],
        },
      },
      {
        name: 'sizes',
        title: 'Available Sizes',
        type: 'array',
        of: [{ type: 'string' }],
      },
      {
        name: 'colors',
        title: 'Available Colors',
        type: 'array',
        of: [{ type: 'string' }],
      },
      {
        name: 'imageUrl',
        title: 'Image',
        type: 'image',
        options: {
          hotspot: true,
        },
      },
      {
        name: 'isNew',
        title: 'Is New',
        type: 'boolean',
      },
    ],
  };
```

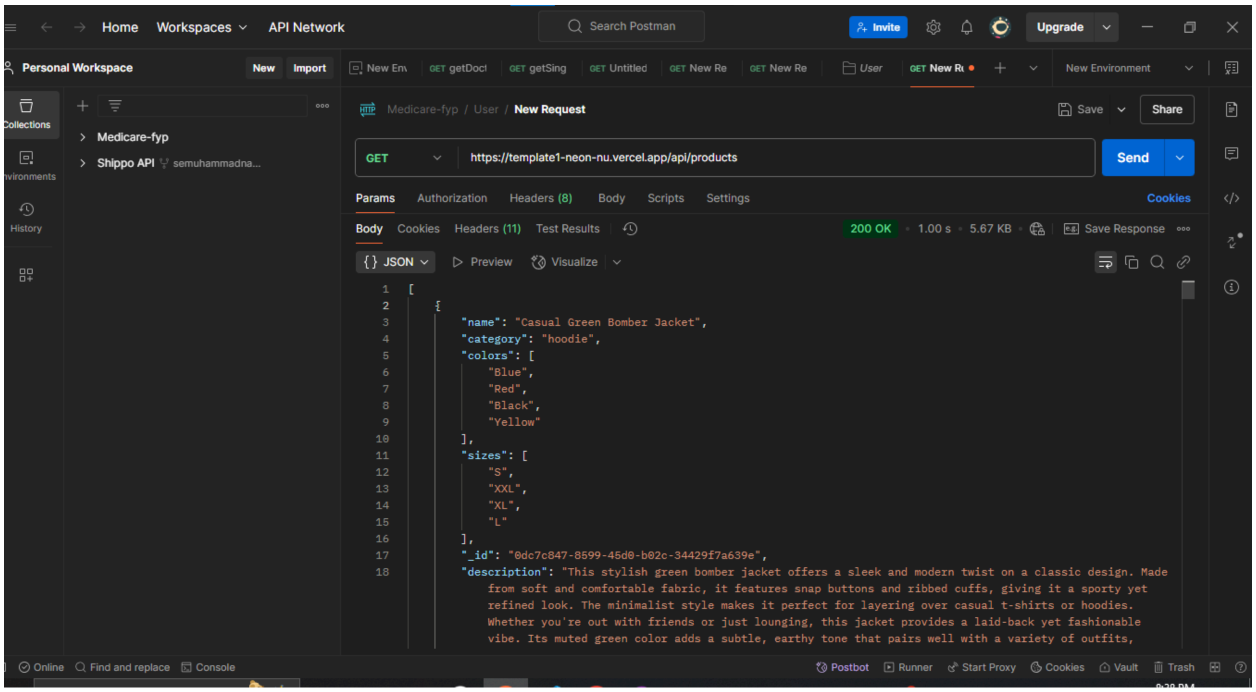
Migration Steps

1. **Environment Setup:**
   - Install dependencies: `@sanity/client`, `axios`, `dotenv`.
   - Create `.env.local` file to store environment variables securely.
2. **Fetch and Transform Data:**
   - Retrieve product data using Axios.
   - Parse and validate the data structure.

3. **Image Uploads:**
    - Download images from API and upload them to Sanity Asset Manager using the Sanity client.

Code Snippet for Data Migration:

```
import { createClient } from '@sanity/client';

const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: NEXT_PUBLIC_SANITY_DATASET,
  useCdn: true,
  apiVersion: '2025-01-17',
  token: process.env.NEXT_PRIVATE_SANITY_TOKEN,
});

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) {
  try {
    const imageId = await uploadImageToSanity(product.imageUrl);

    if (imageId) {

      const document = {
        _type: 'product',
        name: product.name,
        description: product.description,
        price: product.price,
        discountPercent: product.discountPercent,
        category: product.category,
        sizes: product.sizes,
        colors: product.colors,
        isNew: product.isNew,
        imageUrl: {
          _type: 'image',
          asset: {
            _type: 'reference',
            _ref: imageId,
          },
        },
      };

      const createdProduct = await client.create(document);
      console.log(`Product ${product.name} uploaded successfully:`,
createdProduct);
    } else {
      console.log(`Product ${product.name} skipped due to image upload
failure.`);
    }
  } catch (error) {
    console.error('Error uploading product:', error);
  }
}

async function importProducts() {

  try {
    const response = await
fetch('https://template1-neon-nu.vercel.app/api/products');

    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }

    const products = await response.json();

    for (const product of products) {
      await uploadProduct(product);
    }
  } catch (error) {
    console.error('Error fetching products:', error);
  }
}

importProducts();
```



```
GET https://template1-neon-nu.vercel.app/api/products
```

```
200 OK • 1.00 s • 5.67 KB
```

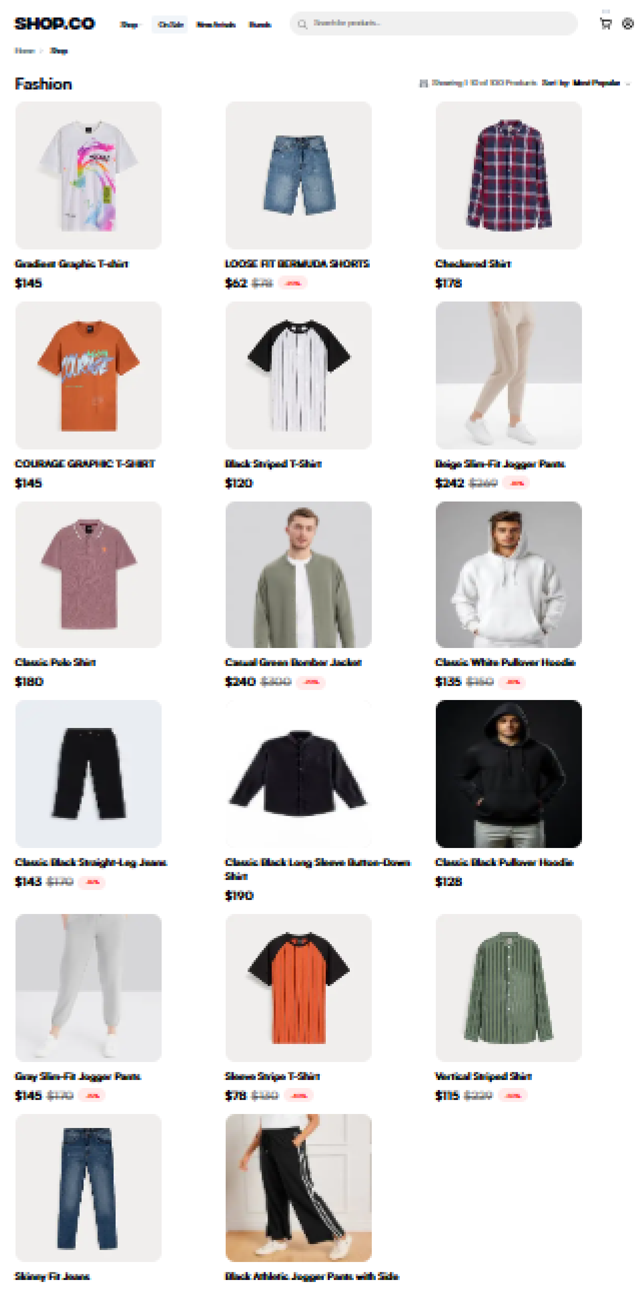
```
[
    {
        "name": "Casual Green Bomber Jacket",
        "category": "hoodie",
        "colors": [
            "Blue",
            "Red",
            "Black",
            "Yellow"
        ],
        "sizes": [
            "S",
            "XXL",
            "XL",
            "L"
        ],
        "_id": "0dc7c847-8599-45d0-b02c-34429f7a639e",
        "description": "This stylish green bomber jacket offers a sleek and modern twist on a classic design. Made
            from soft and comfortable fabric, it features snap buttons and ribbed cuffs, giving it a sporty yet
            refined look. The minimalist style makes it perfect for layering over casual t-shirts or hoodies.
            Whether you're out with friends or just lounging, this jacket provides a laid-back yet fashionable
            vibe. Its muted green color adds a subtle, earthy tone that pairs well with a variety of outfits,
```

API Call with Postman

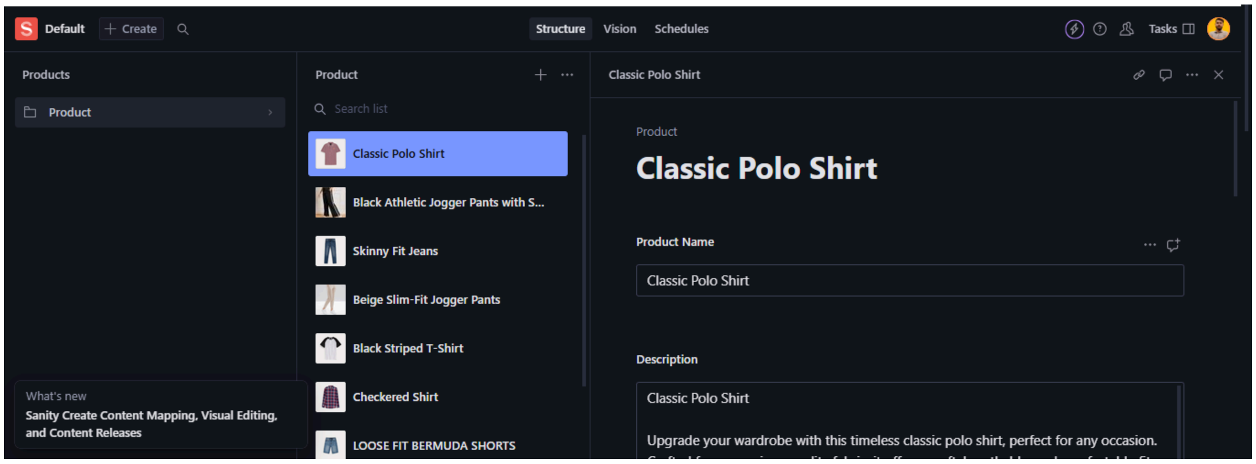
To test the API call using Postman, follow these steps:

1. Open Postman and create a new request.
2. Set the request type to **GET**.
3. Enter the API Endpoint:
   `https://template1-neon-nu.vercel.app/api/products`
4. Click **Send** to retrieve the product data.
5. Verify the response payload structure and log details for confirmation.

FrontEnd Display



Sanity CMS Fields:



Classic Polo Shirt
Black Athletic Jogger Pants with S...
Skinny Fit Jeans
Beige Slim-Fit Jogger Pants
Black Striped T-Shirt
Checkered Shirt
LOOSE FIT BERMUDA SHORTS

Classic Polo Shirt

Product

Classic Polo Shirt

Product Name: Classic Polo Shirt

Description: Classic Polo Shirt

Upgrade your wardrobe with this timeless classic polo shirt, perfect for any occasion.

Checklist

Task	Status (✓ or ✗)
API Endpoint Setup	✓
Schema Adjustments	✓
Data Migration	✓
Testing and Validation	✓
Frontend Display	✓
Populated Sanity CMS Fields	✓

Prepared by: **Muhammad Nabeel**