Day 3 - API Integration and Data Migration Documentation

Overview

On Day 3, we focused on integrating APIs and migrating data for our e-commerce application. The main goal was to fetch product data dynamically from an API and ensure smooth data migration into our system. This document provides a comprehensive guide on the implementation, including setup, integration steps, and potential issues faced during the process.

API Integration

1. Selecting the API

We used **Sanity API** to fetch dynamic product data. This API provides structured and easily manageable content, making it ideal for our e-commerce website.

2. Fetching Data from API

We utilized **Next.js server-side fetching** to retrieve products dynamically. The following steps were taken:

Step 1: Install Dependencies

```
npm install @sanity/client
```

Step 2: Configure Sanity Client

Create a `sanityClient.ts` file in the `lib/` directory:

```
import { createClient } from 'next-sanity'

import { apiVersion, dataset, projectId } from '../env'

export const client = createClient({
  projectId,
  dataset,
  apiVersion,
  useCdn: true, // Set to false if statically generating pages, using ISR or tag-based revalidation
})
```

Step 3: Fetch Data in Next.js Component

Example implementation in `Products.tsx`:

```
import { fetchProducts } from "@/sanity/lib/fetch";
import { allproducts } from "@/sanity/lib/queries";
import Image from "next/image";
import Link from "next/link"; // Import Link for navigation

type Product = {
  _id: string;
  productName: string;
  price: number;
  imageUrl: string;
  description: string;
  colors: string;
  status: string;
  category: string;
};

export default async function ProductCards() {
  const products: Product[] = await fetchProducts({ query: allproducts });

  return (
    <main>
      <div className="w-[90%] mx-auto grid grid-cols-1 md:grid-cols-2 lg:grid-
cols-3 gap-12 items-center">
        {products.map((product) => (
          <Link key={product._id} href={`/products/${product._id}`}>
            <div className="rounded-lg cursor-pointer">
              <Image
                src={product.imageUrl || "/placeholder-image.png"} // Fallback
image
                alt={product.productName || "Product Image"}
                width={348}
                height={348}
                className="rounded-lg"
              />
              <h1 className="text-red-700 mt-3 text-[15px] font-
medium">{product.status || "N/A"}</h1>
              <p className="text-black text-[15px] mt-1 font-
semibold">{product.productName || "Unnamed Product"}</p>
```

```
              <p className="text-black opacity-70 text-[15px] mt-
1">{product.category || "Uncategorized"}</p>
              <p className="text-black opacity-70 text-[15px] mt-
1">{product.colors || "No Colors Listed"}</p>
              <p className="text-black text-[15px] mt-5 font-
medium">{product.price || "0.00"}</p>
            </div>
          </Link>
        ))}
      </div>
    </main>
  );
}
```

Data Migration

1. Why Data Migration?

Migrating data ensures that legacy product data or manually entered records are seamlessly transferred into our new API-based system without data loss.

2. Migration Steps

Step 1: Export Old Data

If data existed in a previous database (e.g., Firebase, MongoDB), we exported it in JSON format.

Step 2: Transform Data to API Structure

Since APIs may have different data structures, we ensured that the exported data matched the schema expected by **Sanity API**.

Example JSON Format:

```
[
  {
    "_id": "product1",
    "name": "Men's T-Shirt",
    "price": 25.99,
    "category": "Clothing"
  },
  {
    "_id": "product2",
    "name": "Women's Hoodie",
    "price": 45.99,
```

```
    "category": "Clothing"
  }
]
```

Step 3: Import Data into Sanity

Using a script to upload data to Sanity:

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

const client = createClient({
    projectId: "nws68vng",
    dataset: "production",
    useCdn: true,
    apiVersion: "2023-10-10",
});

export const fetchProducts = async ({ query, params = {} }: { query: string;
params?: Record<string, unknown> }) => {
    return await client.fetch(query, params);
};
```

Issues & Fixes

Issue 1: Slow Data Fetching

- **Fix:** Used Sanity's CDN (`useCdn: true`) for caching and faster response times.

Issue 2: API Rate Limits

- **Fix:** Implemented request throttling using `setTimeout()` to reduce frequent API calls.

Issue 3: Inconsistent Data Formats

- **Fix:** Ensured all fields matched the API schema before importing data.

Conclusion

By completing API integration and data migration, we successfully transitioned our e-commerce application to a dynamic and scalable structure. The API now dynamically updates product

listings, and historical data has been migrated seamlessly. Moving forward, we can extend this implementation to include filtering, sorting, and advanced caching for even better performance.

Next Steps:

- Implement search and filter features
- Optimize API calls for performance improvements
- Enable caching and real-time updates