DAY 3 - API INTEGRATION AND DATA MIGRATION

Report: API Integration and Data Migration for Sanity and Next.js Project

1. Introduction

This report outlines the steps for integrating the Sanity CMS API with a Next.js application and migrating data effectively. The integration involves fetching data from Sanity, displaying it on a Next.js page, and addressing issues related to the App Router in Next.js 13+.

2. Project Structure

The structure of the project includes both the Sanity CMS setup and Next.js pages.

- **Sanity Configuration (sanity.ts)**
  - Stores API credentials and the connection setup for Sanity.
  - Fetches data from the Sanity backend using the sanityClient.
- **Next.js Page (page.tsx)**
  - Displays data fetched from Sanity using async functions.
  - Involves dynamic data rendering based on the fetched products.

3. API Integration Steps

3.1 Installing Required Packages

- Install the necessary libraries to interact with Sanity from Next.js:
- npm install next-sanity @sanity/client

3.2 Sanity Client Configuration

Create a sanity.ts file in the sanity/ folder and configure the client to connect to the Sanity backend:

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

export const sanityClient = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID || '<YOUR_PROJECT_ID>',
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET || '<YOUR_DATASET>',
  apiVersion: '2023-01-01',
  useCdn: true,
});
```

3.3 Fetching Data from Sanity

In the page.tsx file, use the sanityClient.fetch() method to fetch product data using a GROQ query:

```
import { sanityClient } from '../sanity/sanity';
```

```
export default async function Home() {
  const query = `*[_type == "products"]{
    _id,
    name,
    price,
    description,
    "imageUrl": image.asset->url,
    category,
    discountPercent,
    new,
    colors,
    sizes
  }`;

  // Fetch data from Sanity
  const products = await sanityClient.fetch(query);

  return (
    <div>
      <h1>Products</h1>
      <div style={{ display: 'grid', gridTemplateColumns: 'repeat(3, 1fr)', gap: '20px' }}>
        {products.map((product: any) => (
          <div key={product._id} style={{ border: '1px solid #ddd', padding: '10px' }}>
            <h2>{product.name}</h2>
            <p>Price: ${product.price}</p>
            <p>{product.description}</p>
            {product.imageUrl && <img src={product.imageUrl} alt={product.name} style={{ width: '100px', height:
'100px' }} />}
            <p>Category: {product.category}</p>
            <p>Discount: {product.discountPercent}%</p>
            <p>New: {product.new ? 'Yes' : 'No'}</p>
            <p>Colors: {product.colors.join(', ')}</p>
            <p>Sizes: {product.sizes.join(', ')}</p>
          </div>
        ))}
      </div>
    </div>
  );
}
```

3.4 Environment Variables

- To secure the Sanity API credentials, store them in the .env.local file:
- NEXT_PUBLIC_SANITY_PROJECT_ID=your_project_id
- NEXT_PUBLIC_SANITY_DATASET=your_dataset

4. Data Migration Steps

4.1 Sanity Data Structure

Ensure that your data schema on Sanity is set up correctly. Example for the products schema:

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

```
export default defineType({
  name: 'products',
  title: 'Products',
  type: 'document',
  fields: [
    { name: 'name', title: 'Name', type: 'string' },
    { name: 'price', title: 'Price', type: 'number' },
    { name: 'description', title: 'Description', type: 'text' },
    { name: 'image', title: 'Image', type: 'image' },
    { name: 'category', title: 'Category', type: 'string' },
    { name: 'discountPercent', title: 'Discount Percent', type: 'number' },
    { name: 'new', type: 'boolean', title: 'New' },
    { name: 'colors', title: 'Colors', type: 'array', of: [{ type: 'string' }] },
    { name: 'sizes', title: 'Sizes', type: 'array', of: [{ type: 'string' }] },
  ],
});
```

4.2 Data Importing/Exporting

If migrating data from an existing database to Sanity, use Sanity's built-in tools or custom scripts. Data can be exported from the existing system (e.g., JSON or CSV format) and then imported into Sanity using their API or Studio.

5. Testing

Once the integration and migration are complete:

- Test the data fetching by running the Next.js app (`npm run dev`).
- Verify that products appear on the page.
- Check if all product attributes (name, price, image, description, etc.) are correctly displayed.

6. Conclusion

By following the steps mentioned, you can successfully integrate Sanity with a Next.js.This approach ensures a seamless and efficient data-fetching workflow while complying with Next.js's new structure for server-side rendering.