**Day 3 Roadmap: API Integration & Data Migration**

**Objective**

Integrate a **Mock API** into your Sanity CMS and migrate data to populate your e-commerce website’s product schema.

**Step-by-Step Process**

**Phase 1: Mock API Setup**

1. **Create a Mock API**
   * Use platforms like **MockAPI.io** or **Postman** to design a mock API.
   * Define your resource schema (e.g., products with title, price, description, image).
   * Generate an API endpoint (e.g., https://mockapi.io/api/v1/products).
2. **Sanity Schema Integration**
   * Use your productApi schema in Sanity to mirror the Mock API structure.
   * Example schema fields:

// product.ts (Sanity Schema)

export const productApi = defineType({

name: "productss",

title: "Product",

type: "document",

fields: [

defineField({ name: "title", type: "string" }),

defineField({ name: "description", type: "text" }),

defineField({ name: "productImage", type: "image" }),

defineField({ name: "price", type: "number" }),

// ...other fields

],

});

1. **Data Migration**
   * Fetch data from the Mock API endpoint and migrate it to Sanity:
     + Create a sanityData.mjs file in your Next.js project.
     + Import the Mock API URL and Sanity client:

// sanityData.mjs

import { createClient } from "@sanity/client";

const client = createClient({ projectId: "YOUR\_ID", dataset: "production" });

// Fetch Mock API data

const mockData = await fetch("https://mockapi.io/api/v1/products");

const products = await mockData.json();

// Migrate to Sanity

products.forEach(async (product) => {

await client.create({

\_type: "productss",

title: product.title,

price: product.price,

description: product.description,

productImage: { asset: { \_ref: product.image } },

});

});

* + Run node sanityData.mjs in the terminal to migrate data.

**Phase 2: Sanity Integration with Next.js**

1. **Fetch Data in Next.js**
   * Use GROQ queries in Next.js to fetch product data from Sanity:

// pages/index.tsx

import { client } from "@/lib/sanity.client";

import { groq } from "next-sanity";

const query = groq`\*[\_type == "productss"]`;

const products = await client.fetch(query);

1. **Display Products**
   * Render the migrated data in your frontend:

// components/ProductList.tsx

{products.map((product) => (

<div key={product.\_id}>

<img src={urlFor(product.productImage).url()} />

<h3>{product.title}</h3>

<p>${product.price}</p>

</div>

))}

**Visual Structure**

**Data Flow Diagram**

Copy

[1. Mock API]

| (Fetch data)

↓

[2. sanityData.mjs] → [3. Sanity CMS]

| (Migrate data)

↓

[4. Next.js App] → [5. Frontend UI]

1. **Mock API**: Hosts dummy product data.
2. **sanityData.mjs**: Script to fetch Mock API data and push it to Sanity.
3. **Sanity CMS**: Stores migrated product data in the productss schema.
4. **Next.js App**: Fetches data from Sanity using GROQ queries.
5. **Frontend UI**: Displays products dynamically.

**Key Deliverables for Day 3**

1. **Mock API Endpoint**
   * Contains sample product data (title, price, image).
2. **Sanity Schema**
   * productss schema with fields matching the Mock API.
3. **Data Migration Script**
   * sanityData.mjs to automate data transfer.
4. **Integrated Frontend**
   * Dynamic product listings using Sanity data.

**Testing & Validation**

1. **Verify Data Migration**
   * Check Sanity Studio to ensure products are added to the productss collection.
2. **Frontend Test**
   * Ensure products render correctly on the homepage.
3. **API Test**
   * Confirm the Mock API endpoint returns valid JSON data.

**Tools Used**

* **Mock API**: MockAPI.io / Postman
* **Sanity CMS**: For schema design and data storage.
* **Next.js**: To fetch and display data.
* **TypeScript**: For type-safe API integration.