API INTEGRATION REPORT

NIKE-ECOMMERCE-MARKETPLACE

Objective: The focus for Day 3 is integrating APIs and migrating data into Sanity CMS to build functional marketplace backend.

1. Understanding The API:

For your Nike e-commerce marketplace, I am streamlining the product management process by leveraging a custom mock API to import detailed product data directly into my Sanity schema. This approach will save me from the tedious task of manually entering long and complex product details. However, for other key data—like different product categories—i'll continue to manually manage them in Sanity, ensuring a balance of automation and personalization.

Integrating APIs into my Next.js project involves following steps. Here's an overview of the process:

- **Identifying endpoints:**
  Key endpoints are the main access points of an API that provide data or perform actions. For my Products Listing I am using my mockApi with the ending point "**/products**".
  https://677e7e1994bde1c1252c2704.mockapi.io/products .

- **Setting Up Axios for API Requests:**
  To make API requests, it's common to use Axios or the built-in fetch function.

- **Data Fetching in Next.js Using Sanity and Server Components:**
  The method used is **Server-Side Data Fetching** using an async function inside a React Server Component .This approach takes advantage of the Next.js App Router, where i defined asynchronous logic directly in a

component to fetch data from external sources, such as the Sanity CMS, on the server side.

2. Validated and Adjusted my Schema:

After comparing my existing Sanity CMS schema (created on Day 2) with the API data structure, I realized no significant adjustments were necessary. Since I designed the API myself, it already contained the required data. However, I decided to remove one field that I found redundant:

Removed Field: brand
Reason: As my product listings exclusively feature Nike products,

3. Data Migration:

To streamline my data migration, here are the methods I used.

- **Using the Api :**

  Scripts: I wrote scripts (using JavaScript) to interact with the source API, fetch the data, and transform it into a format that matches my **Sanity schema**.

  Sanity's client libraries: I used Sanity's client libraries (such as @sanity/client) to push the transformed data into my Sanity project.

- **Manual Import :**

  1. Imported Data from API into Sanity Using Axios

  - Installed Axios and Sanity Client: First, I installed both Axios and the Sanity client installed in my project.
  - Setted Up Sanity Client: Initialize the Sanity client with my project details. This client will allow me to push data into Sanity.
  - Fetched Data from API using Axios: I used Axios to fetch data from my API. This step will retrieve my data, which I formatted to match my Sanity schema.

2. **Imported Data from Sanity Using GROQ Queries:**

- Setted Up Groq query**:** After importing the data into Sanity, I retrieved it using GROQ queries. Setted up a query to fetch the data, I needed from my Sanity dataset.
- Executed the Query with Sanity Client: I used the Sanity client to execute the GROQ query and retrieve the data from Sanity.

4. API Integration in Next.js:

1. Created Utility Functions:

To keep my code modular and reusable, I wrote utility functions to handle API calls

Utility Function for Fetching Data:

```
async function importData() {
  try {
    console.log('Fetching products from API...');
    const productsResponse = await axios.get(
      'https://677e7e1994bde1c1252c2704.mockapi.io/products'
    );


    const products = productsResponse.data || [];
    console.log(`Fetched ${products.length} products `);

   for (const product of products) {
      console.log(`Processing product: ${product.name}`);
      let imageRef = null;
      if (product.image) {
        imageRef = await uploadImageToSanity(product.image);
      }
```

2. Render Data in Components:

I used the utility functions to fetch and display data within my Next.js components.

Fetch and Render Data in Components :

```
async function SNRKSsection() {
  const query = `*[_type == "product"]{
    name,
    color,
    image
  }`;
  const products: Product[] = await client.fetch(query);
```

```
useEffect(() => {
    const fetchProducts = async () => {
      const query = `*[_type == "products"]{
        _id, name, "imageUrl": image.asset->url, description, price, sizes,
rating,
        stock, discount, category, color, details, style, tag, id
      }`;
      try {
        const productsData: Product[] = await client.fetch(query);
        setProducts(productsData.map((product) => ({
          ...product,
          imageUrl: urlFor(product.imageUrl).url(),
        })));
      } catch (error) {
        console.error('Error fetching products:', error);
      }
    };

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

```
async function NikeAir () {
  const query = `*[_type == "category"] {
    _id,
    name,
    "image": image.asset->url,
    description,
    price,
    sizes,
    rating,
```

```
    stock,
    discount,
    category,
    color,
    details,
    style,
    tag,
  }`;
    const categories: Category[] = await client.fetch(query);
```

3. Test API Integration:

Testing my API integration ensured data consistency and identified potential issues.

Using Thunder Client & Postman:

- I Installed Thunder Client & Postman via vscode extensions and used it to test your API endpoints (e.g., /api/products).
- I Checked the response status, headers, and body for accuracy.

4.Error Handling and Scalability:

Try-catch blocks:

I wrapped API calls in try-catch blocks and handle errors gracefully to improve user experience.

Log Errors:

Used a centralized logger like logError for easy debugging.

Data Successfully Displaying in FrontEnd:

My Products Listings:

Tops & T-Shirts
Hoodies & Sweatshirts
Jackets
Trousers & Tights
Shorts
Tracksuits
Jumpsuits & Rompers
Skirts & Dresses
Socks
Accessories & Equipment

Gender

- ☐ Men
- ☐ Women
- ☐ Unisex

Shop by Price

- ☐ Under ₹ 2,500.00
- ☐ ₹ 2,501.00 - ₹

Just In
Nike Air Force 1 Mid '07
Men's Shoes
RedBlueWhite
MRP : ₹10795.00

Just In
Nike Court Vision Low Next Nature
Men's Shoes
whiteblackredgrey
MRP : ₹4995.00

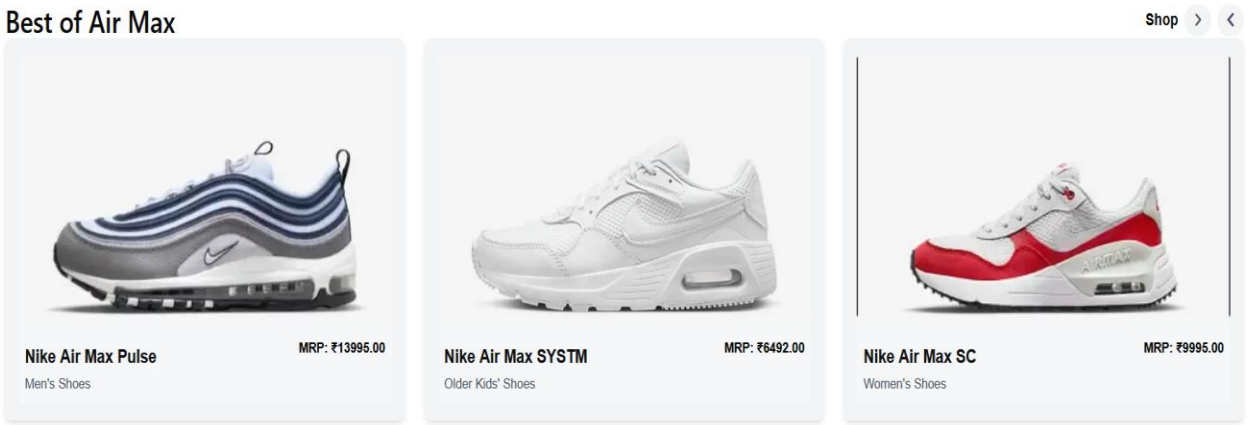
Just In
Nike Air Force 1 React
Men's Shoes
BlackWhite
MRP : ₹13295.00

Just In
Nike Dunk Low Retro
Men's Shoes
Black/WhiteRed/WhiteBlue/White
MRP : ₹8695.00

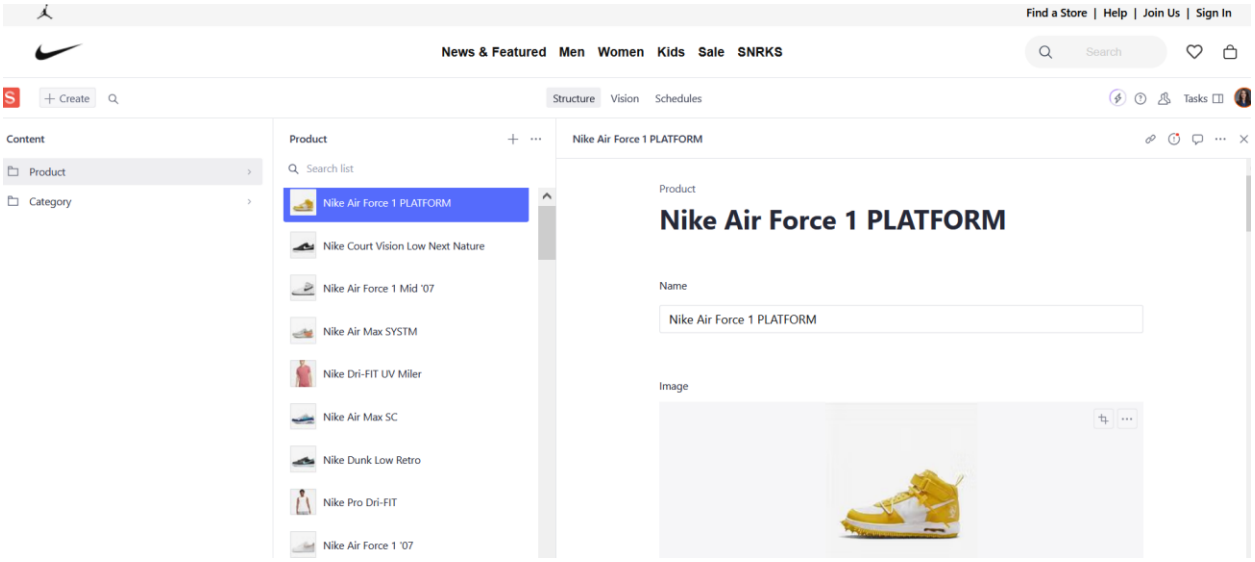
Just In
Nike Air Max SYSTM
Older Kids' Shoes
Red/WhiteGrey/Black
MRP : ₹5695.00

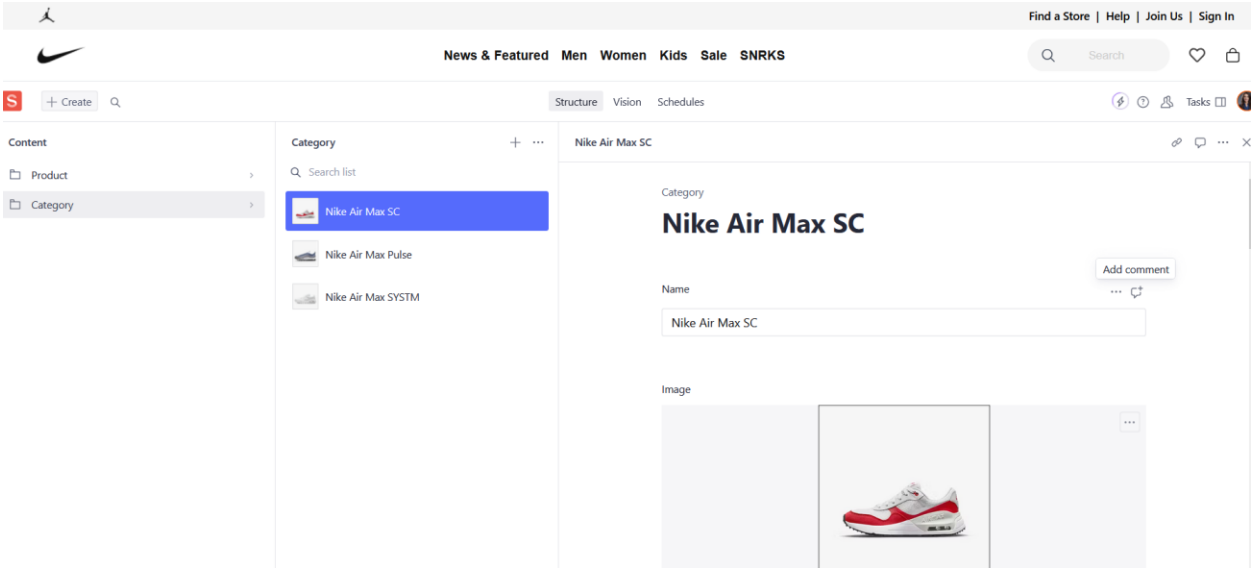
Just In
Nike Court Legacy Lift
Women's Shoes
White/PinkWhite/Black
MRP : ₹9695.00

For Category:



Populated Sanity CMS Field:




Find a Store | Help | Join Us | Sign In
News & Featured Men Women Kids Sale SNRKS
Search
Create
Structure Vision Schedules
Tasks
Content
Product
Category
Category
Search list
Nike Air Max SC
Nike Air Max Pulse
Nike Air Max SYSTM
Nike Air Max SC
Category
Nike Air Max SC
Add comment
Name
Nike Air Max SC
Image