

AURA

ELEVATE YOUR FASHION

Day 3 - API Integration Report - AURA

Steps Taken for DAY 3 - AURA

API Integration Process

Integrated the external API to fetch Aura's product data seamlessly into the system.

Fetching Data from External API

Fetches product details like price, stock, and descriptions from the API and transformed them to fit Aura's specific needs.

Adjustments Made to Schemas

Customized Aura's Sanity CMS schemas by adding fields like discountPercentage, tags, and isNew to meet business requirements.

Explanation of Migrating Data from API to Aura CMS

Migrated product data from the external API to Aura's CMS while addressing format mismatches to ensure seamless integration.

Comparing API Data with Schema

Verified and adjusted API data to align perfectly with Aura's CMS schema for error-free compatibility.

Writing the Migration Script

Created a tailored script to automate the process of importing Aura's product data into Sanity CMS efficiently.

Setting Up the Environment

Configured environment variables, API keys, and CMS credentials to enable smooth integration between Next.js, Sanity CMS, and the external API.

Modifying package.json to Add Script for Import-Data

Added custom commands in Aura's package.json to easily execute the migration script.

Importing Data into Aura CMS

Imported product data into Aura's Sanity CMS using the migration script and validated it within the CMS studio

Calling Data in Frontend

Leveraged Next.js features like getStaticProps and dynamic routing to fetch and display Aura's product data dynamically on the frontend.

1. API Integration Process

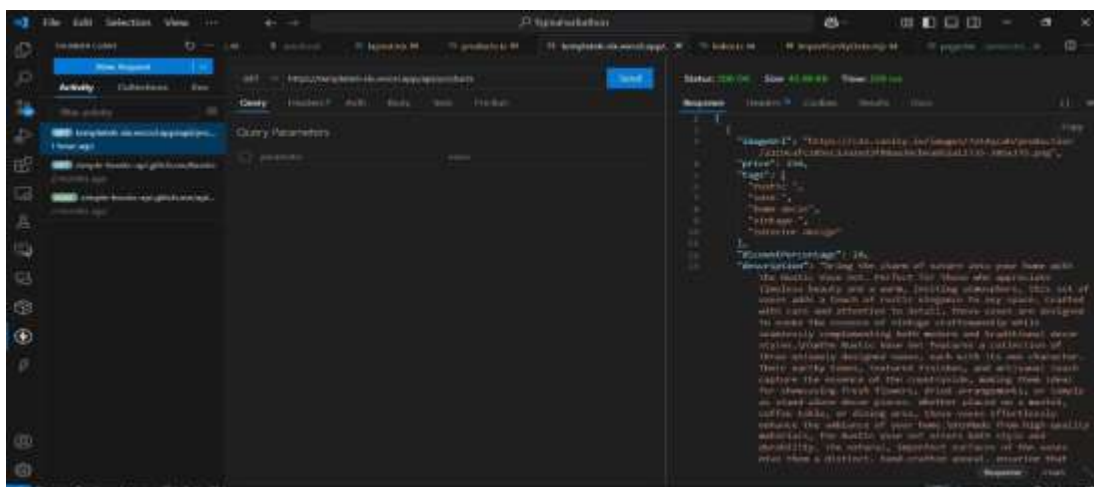
In this section, we describe the process of integrating the external API to fetch product data. A GET request was made using `fetch()` or `axios` to retrieve product details like name, price, and stock information. The data was processed, mapped to fit the product schema, and displayed dynamically on the frontend using Next.js components.

- Integrated external API: <https://template6-six.vercel.app/api/products>
- Used `fetch()` or `axios` to retrieve product details in JSON format.
- Processed and mapped the data to the product schema.
- Displayed data dynamically using Next.js components.

2. Fetching Data from External API

Data was fetched from the external API using a GET request. The fetched product data was mapped to the Sanity CMS schema, and necessary transformations like calculating `priceWithoutDiscount` were applied. The data was then inserted into Sanity CMS.

- Fetched data from the external API using a GET request.
- Mapped API data to the Sanity schema, applying necessary transformations.
- Inserted the formatted data into Sanity CMS.



3. Adjustments Made to Schemas

To meet Aura's business needs, adjustments were made to the Sanity CMS schema. Fields like discountPercentage, tags, isNew, and priceWithoutDiscount were added, while oldPrice was removed. This new schema ensures better product management and scalability.

- **New Fields Added:**

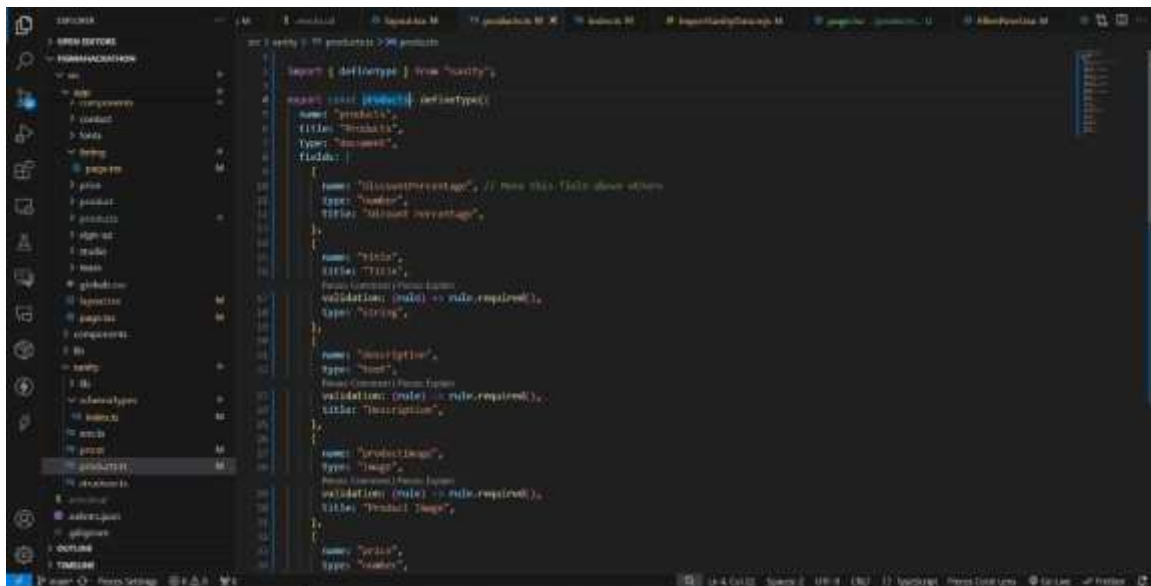
- discountPercentage: Tracks product discounts.
- tags: Categorizes products like “sale” or “new”.
- isNew: Indicates if a product is new.
- priceWithoutDiscount: Stores the original price.

- **Fields Updated/Removed:**

- Replaced ‘oldPrice’ with priceWithoutDiscount.

- **Schema Structure:**

- Fields: name (String), price (Number), discountPercentage (Number), tags (Array).



4. Explanation of Migrating Data from API to Sanity CMS

The migration involved transforming API data to fit the new schema in Sanity CMS. Challenges like data format mismatches were resolved by adjusting field mappings, ensuring seamless integration.

- Migrated product data by fetching it from the API.
- Data was transformed to match the Sanity schema.
- Resolved field mapping issues during migration.

5. Comparing API Data with Schema

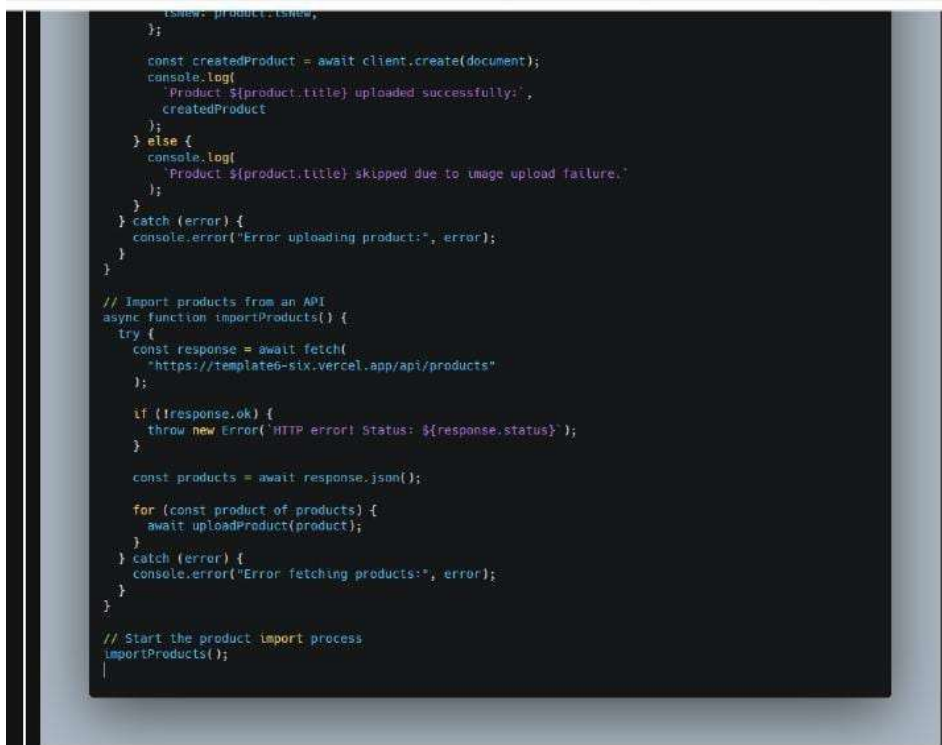
We compared the API data with the updated Sanity schema to ensure compatibility. Field names and data types were adjusted as needed, ensuring a smooth integration between the API and the CMS.

- Compared API data with the Sanity schema for compatibility.
- Adjusted mismatched fields and data types.

6. Writing the Migration Script

The `import-data.mjs` script automates the migration of product data from the API to Sanity CMS. It fetches data, transforms it, and inserts it into Sanity CMS, streamlining the data transfer process.

- Created `import-data.mjs` script to automate data migration.
- The script fetches API data, transforms it, and inserts it into Sanity CMS.



```
const product = {
  title: 'Product Title',
  description: 'Product Description',
  price: 100,
  image: 'product-image.jpg'
};

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

// Import products from an API
async function importProducts() {
  try {
    const response = await fetch(
      'https://template6-six.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);
  }
}

// Start the product import process
importProducts();
```

7. Setting Up the Environment

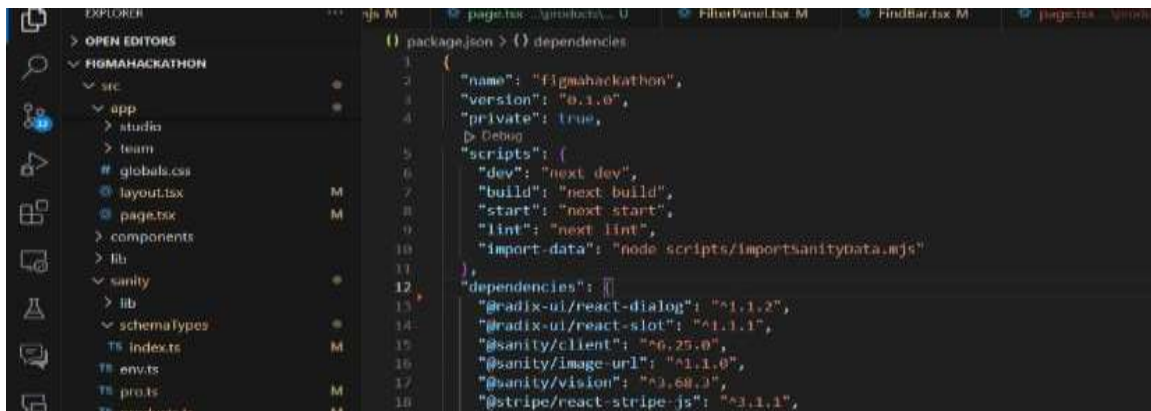
We configured the environment for API integration and data migration, including setting environment variables for API keys and Sanity CMS credentials. This setup ensured proper communication between Next.js, Sanity CMS, and the external API.

- Configured environment variables for API keys and CMS credentials.
- Ensured smooth communication between Next.js, Sanity CMS, and the external API.

8. Modifying package.json to Add Script for Import-Data

We added a script to package.json to simplify the execution of the migration. This allows the migration script to be run using the `npm run import-data` or `yarn import-data` command.

- Added the `import-data` script to package.json.
- Enabled execution using `npm run import-data` or `yarn import-data`.

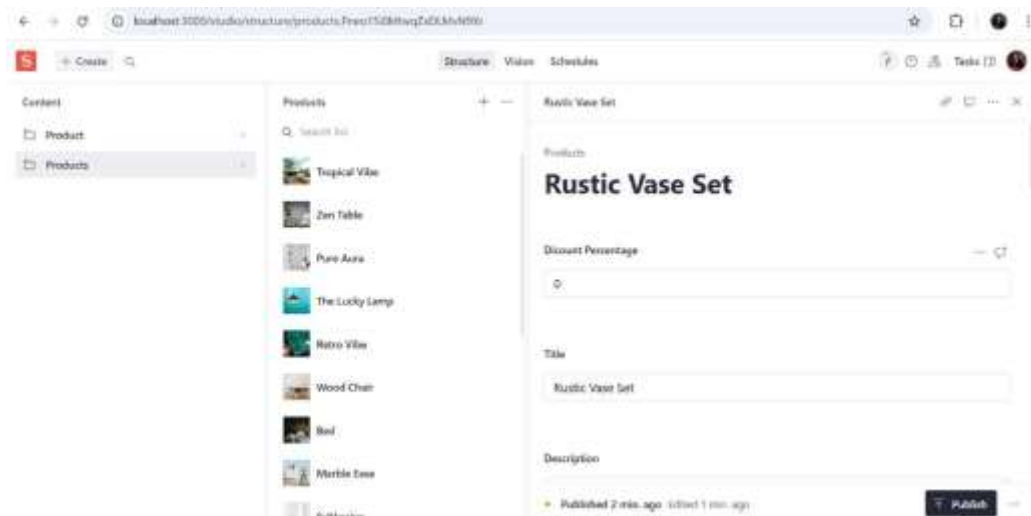


```
{} package.json > {} dependencies
1
2 {
3   "name": "figmahackathon",
4   "version": "0.1.0",
5   "private": true,
6   "scripts": {
7     "dev": "next dev",
8     "build": "next build",
9     "start": "next start",
10    "lint": "next lint",
11    "import-data": "node scripts/importsanitydata.mjs"
12  },
13  "dependencies": {
14    "@radix-ui/react-dialog": "^1.1.2",
15    "@radix-ui/react-slot": "^1.1.1",
16    "@sanity/client": "^6.25.0",
17    "@sanity/image-url": "^1.1.0",
18    "@sanity/vision": "^3.60.3",
19    "@stripe/react-stripe-js": "^3.1.1",
```

9. Importing Data into Sanity

We ran the migration script to import the product data into Sanity CMS. The data was validated by checking the Sanity CMS studio and ensuring products were correctly displayed.

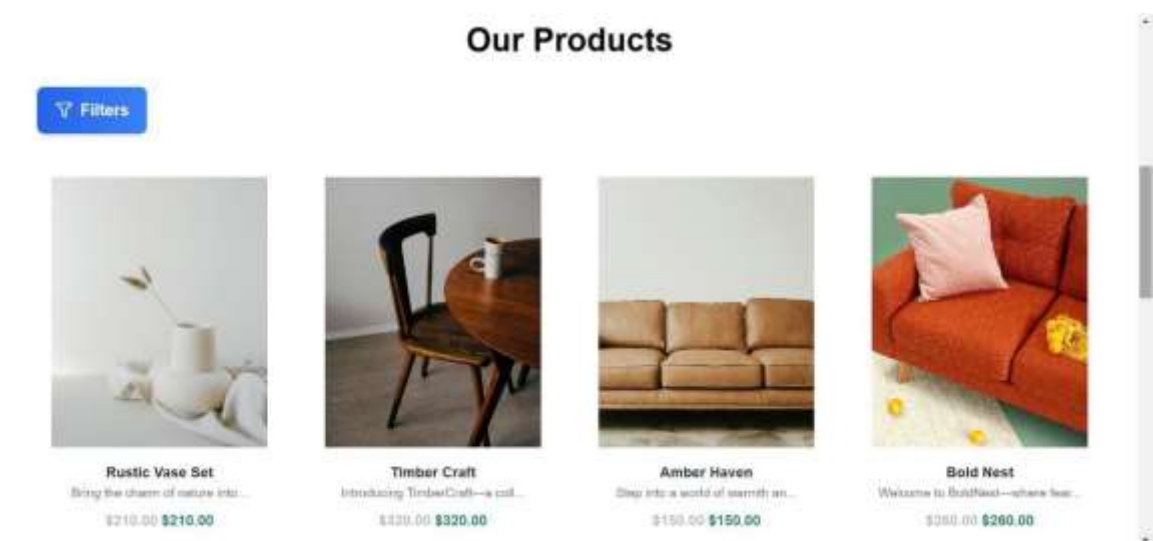
- Executed the migration script to import product data into Sanity CMS.
- Validated the data by reviewing the CMS studio.



10. Calling Data in Frontend

The product data was fetched during the build process using `getStaticProps` and displayed on the frontend. Dynamic routing was implemented to display products based on the slug, ensuring each product had a dedicated page.

- Used `getStaticProps` to fetch product data at build time.
- Created dynamic routes using `[id].tsx` for individual products.
- Displayed products dynamically based on their slug.



CHECKLIST FOR DAY 3 – AURA

Category	Status
API Understanding	✓
Schema Validation	✓
Data Migration	✓
API Integration in Next.js	✓
Submission Preparation	✓

Thank You for Reading!

Let's work together to make Aura a global success in fashion and technology.

Contact: hafeez27isbest@gmail.com

TOGETHER, WE CAN ELEVATE AURA TO NEW HEIGHTS!