Day 3 Report: API Integration and Dynamic Data Fetching for Furniro

1. Objective

The focus of Day 3 was to integrate APIs with Sanity CMS and dynamically fetch data for the Furniro marketplace frontend. This report outlines key activities, including reviewing API documentation, aligning the schema, and fetching data from Sanity into the Next.js application.

2. Key Activities

Step 1: Reviewing API and Sanity Documentation

- **Reviewed Sanity CMS Documentation:**
  Gained a clear understanding of the data schema for products and their structure in Sanity CMS.
- **Sanity API Overview:**
  Identified query structures and endpoints for dynamic data fetching.

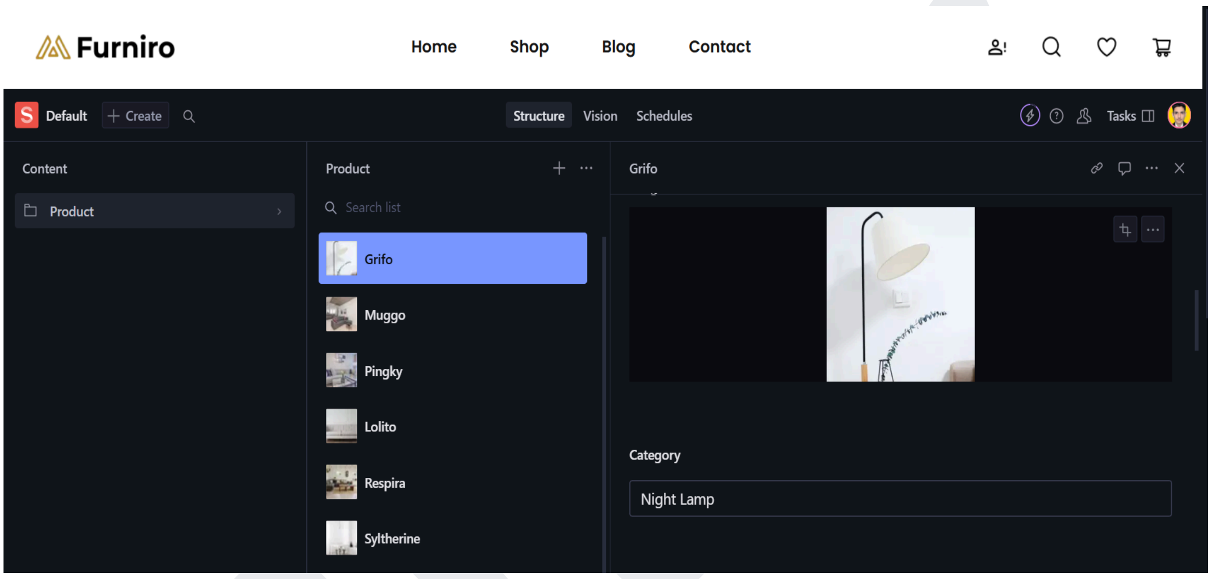
Step 2: Schema Validation and Adjustment in Sanity CMS

- Ensured schema compatibility between the API and Sanity CMS by validating field names and data types:
  - **API Field ➡️ Sanity Schema Field**
    - `product_name` ➡️ `name`
    - `product_price` ➡️ `price`
    - `imageURL` ➡️ `image`
  - Adjusted types and added array fields for categories, colors, and sizes.

Step 3: Manually Migrating Data into Sanity CMS

- **Actions Taken:**
    - Added product data manually using Sanity Studio.
    - Verified the correct mapping of product fields in the CMS.
- **Results:**
    - Product entries in Sanity matched the expected schema and structure.

Screenshot of Products Data :



Category

Night Lamp

Step 4: Dynamic Data Fetching in Frontend

- **Implemented Sanity Client in Next.js:**

Installed the Sanity client for seamless data retrieval:

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

- Created a utility file (`sanityClient.ts`) for configuring the Sanity client.

- **Fetched Product Data Dynamically:**

```
• const query = '*[_type == "product"]';
const fetchedProducts = await client.fetch(query);
        ○
```

- Integrated a `useEffect` hook to dynamically fetch and render product data in the frontend.

---

3. Results

Successfully achieved the following:

1. **Schema Validation:** Ensured compatibility between the API and Sanity CMS.
2. **Data Migration:** Migrated product data into Sanity CMS without errors.
3. **Dynamic Fetching:** Retrieved product data dynamically and displayed it in a responsive frontend layout.

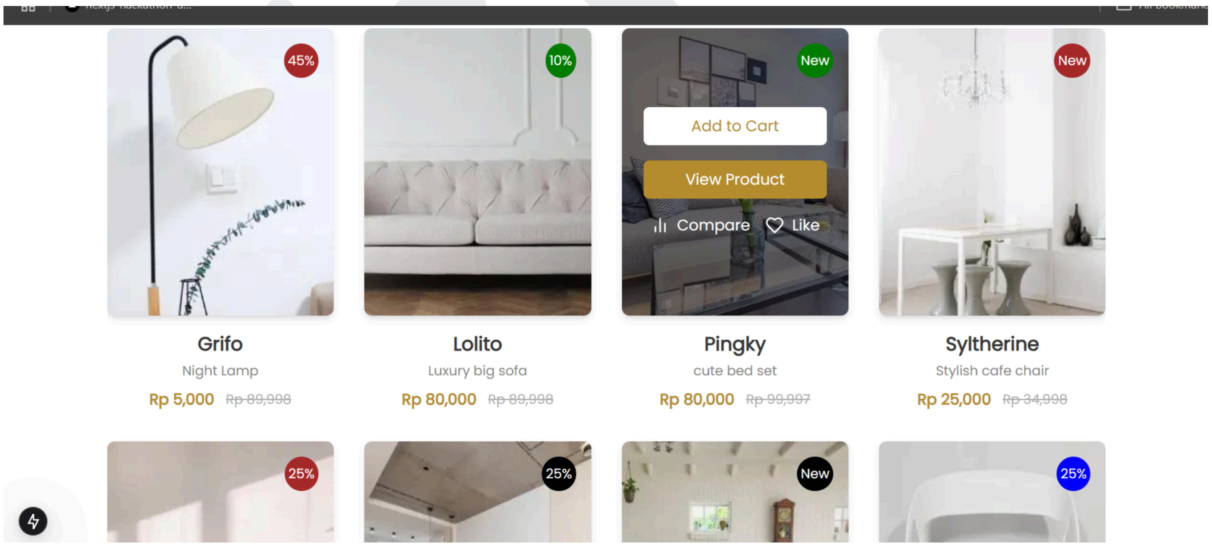
4. **Frontend Rendering:** Rendered accurate product data with schema-aligned fields.

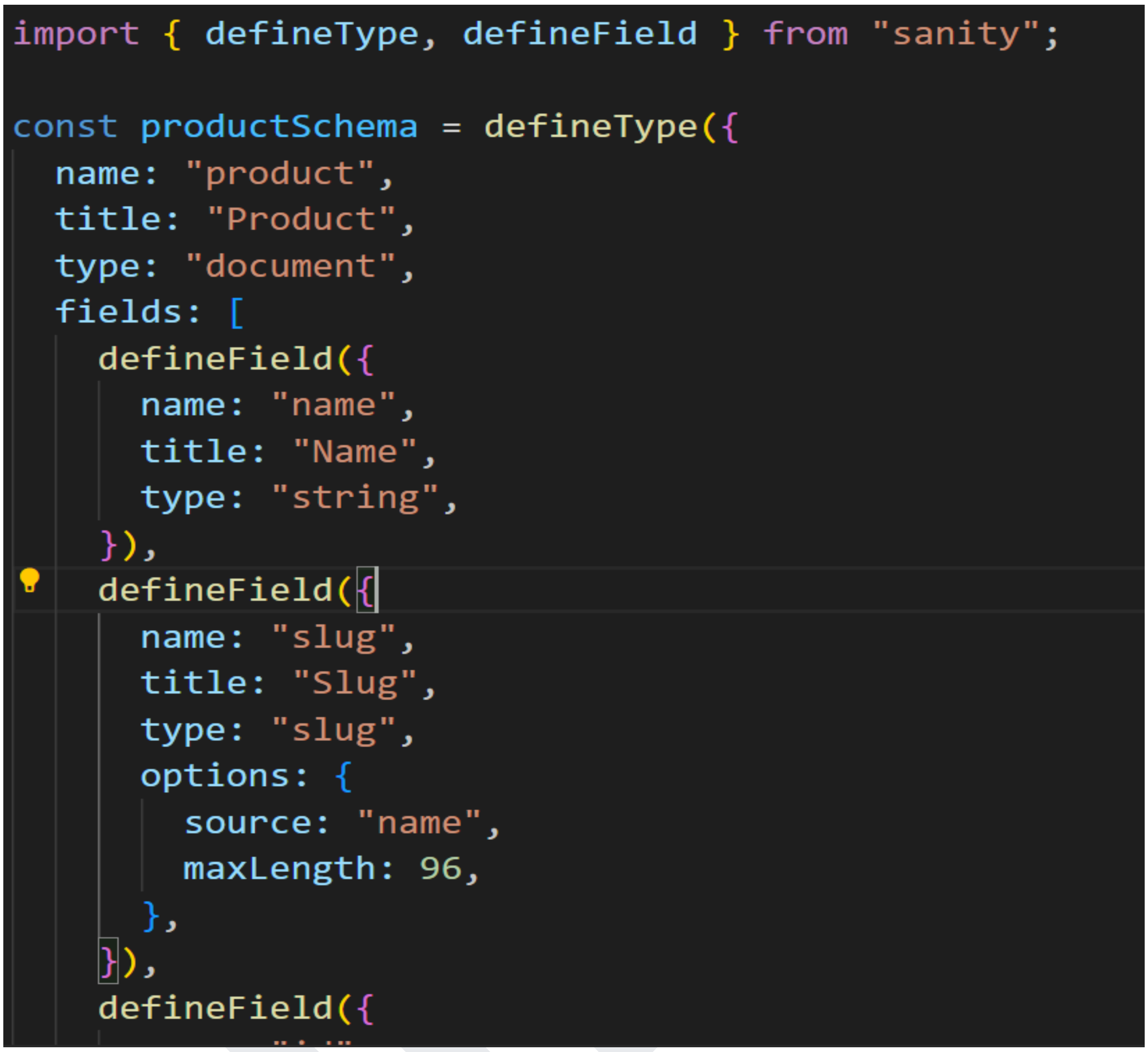
---

4. Submission Checklist

Task	Status
Review API and Sanity Docs	✓
Validate and adjust Schema	✓
Manually Migrate Data	✓
Implement Dynamic Fetching	✓
Render Data on Fronted	✓

DISPLAYING DATA ON FRONTEND





```
import { defineType, defineField } from "sanity";

const productSchema = defineType({
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    defineField({
      name: "name",
      title: "Name",
      type: "string",
    }),
    defineField({
      name: "slug",
      title: "Slug",
      type: "slug",
      options: {
        source: "name",
        maxLength: 96,
      },
    }),
    defineField({
```

5. Conclusion

Day 3 focused on end-to-end integration of APIs with Sanity CMS, culminating in dynamically rendering data on the frontend. The successful implementation highlights the system's scalability and readiness for future enhancements.