Day 3 - API Integration Report - [Avion]

1. API Integration Process:

To successfully integrate the provided API with the frontend of my website, I followed a systematic approach as outlined below:

- **API Review:**
  I carefully reviewed the provided API documentation to identify key endpoints, which include:
  - **Product Listings:** /products
  - **Categories:** /categories
    These endpoints provide critical data for populating the website with products and categories.
- **API Integration in Next.js:**
  I connected the Avion frontend with the provided API by using utility functions in the Next.js project. These functions were responsible for fetching data from the API endpoints and rendering it in the required components. Here's the approach used:
  - **Create Utility Functions:** I created functions to fetch data from the /products and /categories endpoints.
  - **Render Data in Components:** The data fetched from the API was rendered in the product and category components within the Next.js frontend.
  - **Error Handling:** Robust error handling was implemented to ensure that any API failure would display a user-friendly error message on the frontend.
- **Testing API Integration:**
  Using tools like **Postman** and browser developer tools, I tested the API endpoints to ensure data consistency and proper fetching. I also logged responses and validated the data being fetched for accuracy.

2. Adjustments Made to Schemas:

The schemas provided for Avion in Sanity CMS included two core document schemas: Category and Product. These schemas were adjusted to match the structure and data requirements from the API.

- **Category Schema:**
  The Category schema was used to store the different categories of products. It contained the following fields:
  - **Name** (string)
  - **Slug** (slug)

This schema was adjusted to align with the data retrieved from the /categories endpoint of the API.

- **Product Schema:**
  The Product schema represents individual product entries in the Avion marketplace. The fields include:
  - **Category** (reference to the Category schema)
  - **Name** (string)
  - **Slug** (slug)
  - **Image** (image)
  - **Price** (number)
  - **Quantity** (number)
  - **Tags** (array of strings)
  - **Description** (text)
  - **Features** (array of strings)
  - **Dimensions** (object containing height, width, and depth)

Adjustments were made to ensure compatibility with the data fetched from the /products endpoint. For instance, the API provides a product's price, quantity, and description, which directly map to the fields in the Product schema.

3. Migration Steps and Tools Used:

To ensure the migration of data from the provided API to Sanity CMS and then integrate it into the frontend, the following steps and tools were used:

- **Step 1: Data Fetching via API**
  I used scripts to fetch data from the API using utility functions. These scripts fetched the product and category data from the respective API endpoints.
- **Step 2: Data Migration with sanity-migration Folder:**
  To streamline the migration process, I utilized the sanity-migration folder that was provided. This helped automate the process of migrating data from the API to the Sanity CMS, saving time and reducing errors. The tool helped transform and map the data from the API to match the predefined schemas (Category and Product).
- **Step 3: Sanity Data Import:**
  Once the data was fetched and transformed using sanity-migration, it was imported into the Sanity CMS using the provided schemas. The category.js and product.js schemas were populated with the relevant data. This data was then available in the Sanity Studio.
- **Step 4: Schema Adjustments**
  I ensured the data structure in Sanity matched the data returned by the API. For example:
  - The name field in the API was mapped to the title field in the schema.
  - The category_id from the API was linked to the category reference field in the product schema.
- **Step 5: Validation and Testing**
  After data migration, I validated the integrity of the data within Sanity by cross-referencing with the API data. I also tested the API integration by ensuring that the frontend displayed the correct product listings and categories.
- **Step 6: Final Integration with Next.js**
  The last step involved integrating the fetched data into the Next.js frontend. The products and categories are displayed in the studio.