Day 3 - API Integration and Data Migration

**Objective:**

The focus for Day 3 is integrating APIs and migrating data into Sanity CMS to build a functional marketplace backend. Students will learn how to utilize APIs for reference, migrate data into Sanity CMS, and ensure compatibility with their templates. This

approach replicates real-world practices and prepares students to handle diverse client requirements, including integrating headless APIs or migrating existing data from popular eCommerce platforms.

**Document Title: "Day 3 - API Integration Report - [Hussaini E-Commerce]"**

**1. API Integration Process**

On Day 3, our primary objective was to integrate APIs into the marketplace project using Next.js and migrate data into Sanity CMS for building a functional marketplace backend. This involved working with template 2 of Sir Bilal and Sir Aneeq, which provided a foundation for API references and template schemas.

**API Overview**: The provided APIs included the following key endpoints, which are essential for the development of the marketplace:

* **Product Listings**: /products
* **Categories**: /categories
* **Order History**: (if applicable)

These API endpoints served as a guide for validating schema compatibility and migrating data into Sanity CMS.

**Steps for API Integration:**

1. **Review the API Documentation**: We reviewed the API documentation for template 2, focusing on the product listings and categories endpoints. These endpoints provided JSON data for products, including fields like name, description, price, image URLs, tags, etc.
2. **Create Utility Functions in Next.js**: We created utility functions in Next.js for fetching and displaying product data from the API. These functions handled GET requests to the API and processed the data to display products on the front-end. Example utility function to fetch product data:

const fetchProducts = async () => {

const res = await fetch('/api/products');

const data = await res. json ();

setProducts(data);

};

1. **Render Data in Components**: We utilized the fetched data and displayed it within our components (e.g., AllProducts.js). This included rendering product names, prices, descriptions, images, and tags in the UI.
2. **Test API Integration**: We used tools like **Postman** and the browser's developer tools to test API responses and ensure that the correct data was being returned. This allowed us to confirm the accuracy of the API calls and ensure no data discrepancies.

### ****2. Adjustments Made to Schema****

To ensure compatibility between the API data and the existing schema in Sanity CMS, we validated and adjusted the schema fields as needed. This involved mapping API field names to schema field names and ensuring proper data types.

**Key Adjustments**:

* **API Field: product\_title** → **Schema Field: name**
* **API Field: product\_description** → **Schema Field: description**
* **API Field: product\_price** → **Schema Field: price**
* **API Field: imageUrl** → **Schema Field: image.asset->url**
* **API Field: tags** → **Schema Field: tags**

These adjustments were made to ensure that the data being migrated from the API would be aligned with the predefined schema in Sanity CMS

### ****3. Migration Steps and Tools Used****

We used multiple methods to migrate data into Sanity CMS based on the available options for data migration. These methods ensured that we were able to populate the CMS with product data effectively.

#### ****Using the Provided API****:

* We wrote scripts to fetch and transform product data from the API. These scripts extracted product details (name, price, description, image URL, etc.) and populated Sanity CMS using the @sanity/client package.

### ****Expected Output****

1. **Sanity CMS populated with imported data**:
   * Product data was successfully populated from the API, external sources, and manual imports into Sanity CMS.
2. **Functional API integration in Next.js**:
   * Product listings and categories were dynamically displayed in the frontend using Next.js components.
3. **Data Validation**:
   * We validated the imported data by running test cases and ensuring it aligned with our schema.













