Day 3 - API Integration Report – “FURNIRO”

Task:” API INTEGRATION AND DATA MIGRATION”

API Integration Process

For the integration of the API into my marketplace project, I used a systematic approach to ensure data was fetched, processed, and displayed effectively. Below is an overview of the steps taken:

1. **Choosing the Right API**
   I used the API provided by the head of my Template 6 project. This API was key in fetching product data for the marketplace, which included product details like title, price, description, and images. The API served as the primary source for dynamic content in the project.

2. **Uploading Data into Sanity CMS**
   After retrieving the data, I used the API provided to upload the necessary content into Sanity CMS. I wrote the code for this in the importData.js file, which was based on the official documentation of Template 6. This script was responsible for processing the fetched data and creating documents in Sanity CMS, enabling seamless integration of the product information.

3. **Fetching Data from the API**
   To display the product information on the frontend, I wrote asynchronous functions using the fetch method to retrieve data from the external API. This data was then formatted to match the structure required by the frontend components in Next.js. This ensured the API response was properly mapped to the components on the page.

4. **Error Handling**
   I incorporated error handling throughout the process to ensure smooth API integration. In the event of an API failure or if the data retrieval did not succeed, error messages were displayed to the user, preventing any disruptions to the user experience. This ensured the stability and reliability of the application, even when unexpected issues occurred with the API.

5. **Displaying Data on the Frontend**
   I used Next.js and React to dynamically render the fetched data on the frontend. React

components were created to display product details such as images, descriptions, and pricing. The data fetched from the API was integrated into these components, allowing the marketplace to display real-time product information in an interactive and user-friendly manner.

Adjustments Made to Schemas

As part of the integration, I had to adjust my Sanity schemas to align with the incoming data. The changes made were as follows:

1. **Image Field**: I added a field for images in the schema. The images were uploaded via the Sanity assets API and referenced in the product document, enabling smooth integration with the frontend.

Migration Steps and Tools Used

The migration process involved moving the product data from an external API to Sanity CMS. Below are the steps I followed:

1. **Fetching Data**: I used a script that fetched product data from the external API.
2. **Processing Data**: I transformed the data to match the schema required by Sanity.
3. **Uploading to Sanity**: I wrote another function to upload the product data, including image references, to Sanity using its client library.
4. **Tools Used**: I used Node.js for writing the migration script and Sanity's official API client to interact with the CMS.

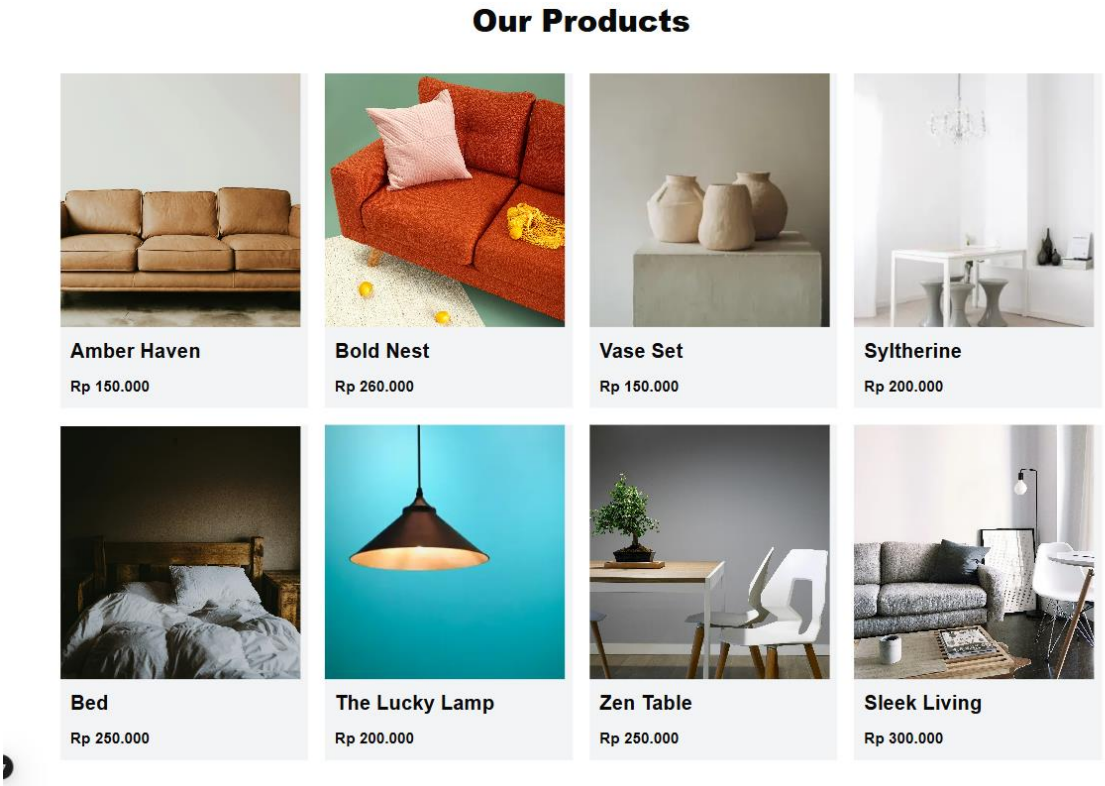
⇨ Screenshots

Query to fetch data into Next.js:

```
Codeium: Refactor | Explain | Generate JSDoc | ×
const getProducts = async() => {
  const products = await client.fetch(
    `*[_type=="product"][0..7]{
  _id,
  title,
  price,
    description,
    productImage

}`
  )
  return products
}
Codeium: Refactor | Explain | Generate JSDoc | ×
const Products = async () => {
  const products = await getProducts();
  console.log(products)
```

Output On Web(FR0NTEND):



Code Snippets for API Integration and Migration Scripts

1) Uploading data to sanity

```
async function uploadProduct(product) {
  try {
    const imageId = await uploadImageToSanity(product.imageUrl);

    if (imageId) {
      const document = {
        _type: 'product',
        title: product.title,
        price: product.price,
        productImage: {
          _type: 'image',
          asset: {
            _ref: imageId,
          },
        },
        tags: product.tags,
        dicountPercentage: product.dicountPercentage, // Typo in field name: dicountPercentage->
discountPercentage
        description: product.description,
        isNew: product.isNew,
      };

      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);
  }
}
```

Prepared by: Fatima Salman