Professional Report: Data Import and Integration with Sanity

This report provides a detailed account of the API integration process, adjustments made to schemas, and the migration steps and tools used to migrate data to Sanity for the project. The objective of this process is to facilitate a seamless integration with Sanity, enabling smooth data import from JSON files to Sanity's CMS platform for the project's various content types such as decorations, catering services, consultancy managers, and reserving destinations.

API Making and MarketPlace Establishment Process:

The document I am creating is about the project we received. For this project, we did not use any old templates because our marketplace is entirely unique compared to existing templates. Therefore, we designed a new frontend from scratch, wrote our backend logic, integrated the data, and uploaded it to the schema. This comprehensive approach allowed us to complete the document effectively.

It is crucial to mention that during this process, we did not use any provided API. Instead, we created our own migration script as well as our own API from scratch.

THEN:

First, we created our own API and made a JSON file. We prepared different JSON files for various endpoints in our project. Among these, there was one for decorations, one for catering, one for reserving destinations, and another for consultancy managers.

In the JSON file we created, we included four main keys: an ID, a name, details, and images. However, the images contained URLs, and since our API was initially developed quickly, some of these URLs were accessible while others were not. So, when the data was transferred to the schema, some images were displayed, and others were not.

When integrating this into the website, we prioritized excluding the images in such cases. As a result, fewer images appeared on the website, and we focused more on integrating the data itself.

API Integration Process

The integration process involved connecting to Sanity's API to upload and manage the content from local JSON files to the Sanity platform. Below are the detailed steps involved:

1. **Environment Variables Setup**: The environment variables were loaded from a `.env.local` file using the `dotenv` package. This file contains critical configuration settings like the `NEXT_PUBLIC_SANITY_PROJECT_ID`, `NEXT_PUBLIC_SANITY_DATASET`, and `NEXT_PUBLIC_SANITY_AUTH_TOKEN`. These variables are essential for authenticating and configuring the connection to the Sanity project.

The script verifies the existence of `.env.local` and loads the environment variables. If any of the required environment variables are missing, an error message is displayed, and the process is halted.

2. **Sanity Client Creation**: The `@sanity/client` package was used to create the Sanity client, which establishes the connection to the Sanity API. The client is configured with the project ID, dataset, and authentication token, ensuring that the requests made to the Sanity API are authorized and properly mapped to the correct project.

   The client is initialized with the following configuration:

   - `projectId`: Refers to the specific Sanity project that holds the data.
   - `dataset`: Defines which dataset within the project will be used for importing data.
   - `useCdn`: Set to `false` to avoid caching issues when making requests.
   - `token`: Used for authenticating API requests.
   - `apiVersion`: Ensures compatibility with a specific version of the Sanity API.
3. **File Reading and Parsing**: The `fs` and `path` modules from Node.js are used to read and parse the JSON files that contain the data to be imported. The `readJsonFile` function reads a given file from the local filesystem and parses it into a JavaScript object, which is then used to upload data to Sanity.
4. **Image Uploading**: The `uploadImageToSanity` function facilitates the uploading of images from external URLs to Sanity. It uses `axios` to fetch the image data and `Buffer.from` to process it before uploading the image to Sanity's asset management system. The function returns the asset ID of the uploaded image, which is then included in the Sanity document for the respective data type.
5. **Data Processing and Uploading**: The core logic of uploading data from JSON to Sanity is encapsulated in the `processData` function. This function processes each item in the data, prepares it for the Sanity schema, and uploads it using `client.createOrReplace`. This ensures that the data is either created or updated based on the presence of an existing `_id`.
6. **Handling Missing Data**: The script ensures that if an item in the JSON data does not have an image, it proceeds with the data upload without it. It also handles optional fields like `location` and `availability`, making sure the data is clean and consistent before being uploaded to Sanity.

Adjustments Made to Schemas

The data to be imported includes multiple content types, each corresponding to a unique Sanity schema. The key adjustment made to the schemas is the creation or modification of the following document types:

1. **Decoration**: The schema for decorations includes fields like `name`, `details`, `price`, `location`, and `image`. The image field uses a reference to the uploaded image asset, ensuring that each decoration item has an associated image in the Sanity media library.
2. **Catering**: Similar to the decoration schema, the catering schema includes basic fields like `name`, `details`, `price`, and `availability`. It also includes an `image` field, referencing the uploaded image asset.

3. **Consultancy Manager**: The consultancy manager schema follows the same structure as the other schemas, with fields like `name`, `details`, and an `image` field that links to the uploaded image asset.
4. **Reserving Destination**: The schema for reserving destinations contains fields like `name`, `details`, `location`, and an optional `availability` field. The `image` field is also included to represent images of the destinations.

Each of these schemas was tailored to meet the specific needs of the project and ensure that the imported data would be correctly structured in Sanity. The `createOrReplace` method used in the process ensures that the data is either created or replaced based on its unique `_id`.

Migration Steps and Tools Used

The migration of data to Sanity was performed using the following steps:

1. **Preparation**: The process began with ensuring that the necessary configuration files were in place, specifically the `.env.local` file with the required environment variables. This setup was essential for securely connecting to the Sanity project and dataset.
2. **Data Import**: The main data import process involved reading and parsing the JSON files from the local filesystem using the `fs` module. For each content type (decorations, catering, consultancy managers, and reserving destinations), the script iterates over the data and prepares it for upload to Sanity. Each item is formatted according to its corresponding schema, and any necessary image assets are uploaded before the data is stored.
3. **Sanity Client**: The integration with the Sanity API was achieved using the `@sanity/client` library, which allowed seamless communication with Sanity's backend. This tool was essential for uploading documents and assets to the Sanity platform.
4. **Image Upload**: Images were uploaded from external URLs to Sanity's asset management system using `axios` to fetch the image data, which was then processed and uploaded as binary data. The uploaded images are referenced in the content documents as part of their respective fields.
5. **Error Handling**: Throughout the process, error handling mechanisms were implemented to catch any issues with reading files, uploading images, or interacting with the Sanity API. This ensures that any failure points are logged, and the process is halted when necessary.
6. **Final Verification**: After the data was successfully uploaded, a final check was performed to verify the correctness of the imported data. This included checking the presence of images and ensuring that all required fields were correctly populated.

Conclusion

The data import process from local JSON files to Sanity was successfully implemented using a robust integration approach. The use of environment variables, Sanity's client library, and the `axios` package ensured smooth API communication and image handling. Adjustments to the schemas allowed for seamless mapping of the data to Sanity's backend, while error handling and logging mechanisms ensured that any issues could be quickly identified and

addressed. The migration process was carried out efficiently, with all data correctly migrated to Sanity's CMS platform, ready to be accessed and displayed on the front-end of the project.