Marketplace Builder Hackathon 2025 - Day 3

Overview

This documentation provides an overview of the work completed during Day 3 of the Marketplace Builder Hackathon 2025. The day focused on:

- API Integration
- Data Migration
- Error Handling
- Asynchronous Operations

Key Accomplishments

- Successfully integrated APIs into the marketplace application.
- Migrated and optimized data to improve performance and scalability.
- Handled edge cases for efficient and smooth data flows.
- Enhanced application performance through improved asynchronous operations.

Sanity Schema for Product

```
import { ShoppingBagIcon } from "lucide-react";

import { defineType } from "sanity";


export const product = defineType({

    name: "product",

    title: "Product",
type: "document",

icon: ShoppingBagIcon,

fields: [

    {

        name: "title",

        title: "Title",

        validation: (rule) => rule.required(),

        type: "string"

    },

    {

        name: "description",

        type: "text",

        validation: (rule) => rule.required(),

        title: "Description",

    },

    {

        name: "productImage",

        type: "image",

        validation: (rule) => rule.required(),

        title: "Product Image"

    },

    {

        name: "price",

        type: "number",

        validation: (rule) => rule.required(),

        title: "Price",
        },
        {
            name: "tags",
            type: "array",
            title: "Tags",
            of: [{ type: "string" }]
        },
        {
            name: "dicountPercentage",
            type: "number",
            title: "Discount Percentage",
        },
        {
            name: "isNew",
            type: "boolean",
            title: "New Badge",
        }
    ]
});
```

API Fetching Code Snippet

```
const res = await client.fetch(groq`*[_type=="product"]{
    _id,
    title,
    "imageUrl": productImage.asset->url,
    price,
    tags,

    dicountPercentage,

    description,

    isNew

}`);
```

Importing Data to Sanity

```
import { createClient } from '@sanity/client';

import axios from 'axios';

import dotenv from 'dotenv';

import { fileURLToPath } from 'url';

import path from 'path';


// Load environment variables from .env.local

const __filename = fileURLToPath(import.meta.url);

const __dirname = path.dirname(__filename);

dotenv.config({ path: path.resolve(__dirname, '../.env.local') });


// Create Sanity client

const client = createClient({

  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID, // Replace with your project ID

  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET, // Replace with your dataset name

  useCdn: false,

  token: process.env.SANITY_API_TOKEN, // Replace with your API token

  apiVersion: '2021-08-31', // Use a compatible API version

});
```

```
async function insertProductData(product) {

  try {

    console.log(`Inserting product: ${product.name}`);

    await client.create({

      _type: 'product',

      id: product.id,

      name: product.name,

      imagePath: product.imagePath,

      price: parseFloat(product.price),

      description: product.description,

      discountPercentage: product.discountPercentage,

      isFeaturedProduct: product.isFeaturedProduct,

      stockLevel: product.stockLevel,

      category: product.category,

    });

    console.log(`Product inserted successfully: ${product.name}`);

  } catch (error) {

    console.error(`Failed to insert product: ${product.name}`, error);

  }

}



async function fetchAndInsertData() {

  try {

    console.log('Fetching products from API...');

    const { data: products } = await axios.get(
      'https://template-0-beta.vercel.app/api/product'

    );



    console.log(`Fetched ${products.length} products.`);

    for (const product of products) {

      await insertProductData(product);

    }

    console.log('All products inserted successfully!');

  } catch (error) {

    console.error('Error fetching or inserting data:', error);

  }

}



fetchAndInsertData();
```

Key Learnings

- Error Handling: Importance of robust error handling mechanisms in API calls.

- Data Migration: Strategies to build efficient and maintainable migration scripts.

- Asynchronous Operations: Techniques to improve efficiency and reduce bottlenecks.

- Teamwork: Collaboration and persistence are essential for overcoming challenges.