Day 3 - API Integration Report - Hiperstar

MARKETPLACE BUILDER HACKATHON 2025

DAY 3: API Integration Report - Hiperstar

1. API Understanding

Understanding the API is crucial for integrating it efficiently into our application. Below are the key details:

Base URL: https://hiperstar.vercel.app/api
Endpoints:

- GET /products - Fetch all products.
- POST /orders - Create a new order.
- GET /customers – Fetch all customers
- POST /customers – Create a new customer

Response Format: JSON
Error Handling: Standard HTTP status codes are used for error handling.

2. Schema Validation

Schema validation ensures our data structure aligns with the marketplace requirements. We used:

- **Sanity Schema Validation**: Ensured fields, types, and relationships between collections were correct.
- **Manual Testing**: Tested API responses against expected outputs.
- **Validation Methods**: Used tools like Thunderclient to validate responses and data integrity.

3. Data Migration

Did manual entries of products because of custom schema and ensured data consistency and correctness by validating after insertion.

4. API Integration in Next.js

Integration involved the following steps:

- **Fetching Data from API**:

Fetching Products:

```
import { type SanityDocument } from "next-sanity";
import { client } from "@/sanity/client";
import { NextResponse } from "next/server";
export async function GET() {
  const PRODUCTS_QUERY = `*[_type == "product"]{
        _id,
        product_name,
        added_at,
        slug,
        main_image,
•           rating,
•           variation_details,
•           price,
•           description,
•           product_images,
•           stock,
•           category->{
•             category_name
•           }
•         }`;
•     const options = { next: { revalidate: 30 } };
•     const products = await client.fetch<SanityDocument[]>(PRODUCTS_QUERY, {}, options);
•     return NextResponse.json(products)
•   }
```

Fetching Customers

```
import { client } from "@/sanity/client";
import { SanityDocument } from "next-sanity";
import { NextResponse } from "next/server";
export async function GET() {
    const CUSTOMERS_QUERY = `*[_type == "customer"]{
            _id,
            email
        }`;
    const options = { next: { revalidate: 30 } };
    const customers = await client.fetch<SanityDocument[]>(CUSTOMERS_QUERY, {}, options);
    return NextResponse.json(customers)
}
```

- **Displaying Data in UI**:
  Used Next.js pages and components to render the data dynamically.
  Implemented paginationfor better user experience.

- **Posting Data to API**:
  Posting Customer Data to sanity

```
•  import { client } from "@/sanity/client";
•  import { SanityDocument } from "next-sanity";
•  import { NextResponse } from "next/server";
•  export async function POST(request: Request) {
•      const formData = await request.json()
•      const customerData = {
•          _type: "customer",
•          address: formData.address,
•          postal_code: formData.zipcode,
•          email: formData.email,
•          firstname: formData.firstname,
•          lastname: formData.lastname,
•          account_creation_date: new Date().toISOString(),
        username: formData.firstname.toLowerCase() + "" + formData.lastname.toLowerCase(),
        phone_number: formData.phone,
        city: {
            _ref: formData.city,
            _type: "reference"
        },
        country: {
            _ref: formData.country,
            _type: "reference"
        }
    }
    const createCustomer = await client.mutate([
        {
            create : customerData
        }
    ])
    return NextResponse.json(createCustomer)
}
```

- **Posting Order Data to sanity**

```

import { client } from "@/sanity/client";
import { NextResponse } from "next/server";
const generateKey = () => `${Date.now()}-${Math.random().toString(36).substr(2, 9)}`;
type order_item = {
    product: string,
    quantity: string,
    variation: {
        _key: string
        variation_name: string,
        variation_option: string
    },
    price: string
}
export async function POST(request: Request) {
    const formData = await request.json()
    console.log("orders->", formData)
    const order_items = formData.productData.map((order_item: order_item) => {
        return {
            _key: generateKey(),
            product: {
                _ref: order_item.product
            },
            quantity: order_item.quantity,
            variation: order_item.variation,
            price: String(order_item.price)
        }
    })
    const createOrder = client.mutate([
        {
            create: {
                _type: "order",
                customer: {
                    _ref: formData.customer
                },
                order_items: order_items,
                total_price: String(formData.total_price),
                order_note: formData.order_note,
                order_status: "pending",
                order_date: new Date().toISOString()
            }
        }
    ])
    return NextResponse.json(await createOrder)
}
```

Conclusion

We successfully understood the API, validated the schema, manually migrated data, and integrated the API into our Next.js application. The next step is testing and optimization to ensure a seamless user experience.