1. API integration process.

When implementing API integrations, the process can vary depending on the project's specific needs. Here is a structured explanation of how the API integration was handled in the context of a customized marketplace:

1. Initial Situation

APIs were provided as part of the project, intended for integration with the marketplace. I encountered a challenge: the provided APIs did not align with the niche of my platform. To address this issue, I decided to create a custom JSON file tailored to the specific needs of my marketplace

2. Custom Solution

To address the incompatibility issue, a custom approach was adopted:

A specific type of JSON file was generated to align with the updated niche and the data requirements of the marketplace.

This JSON file served as a data source, containing all the necessary information tailored to the marketplace's needs.

3. Integration with Sanity CMS

The custom JSON file was used to import data directly into Sanity CMS.

This method ensured that the data structure matched the requirements of the CMS, enabling seamless integration and management.

2. Adjustments made to schemas.

I made several adjustments to the product schema. Below is a comparison of the key differences and modifications:

Field	API Schema	Current Schema	Adjustment
title	String	String	Retained as is to represent the product name.
slug	Not Provided	String	Added for SEO-friendly URLs.
price	Number	Number	Retained as is to indicate the original price.
salePrice	Not Provided	Number	Added to specify the discounted price.
stock	Not Provided	Number	Added to track product availability.
reviewsCount	Not Provided	Number	Added to display the count of product reviews.
imageUrl	String	String	Renamed to image for consistency.
description	String	String	Retained as is for product details.
tags	Array	Not Included	Excluded and replaced with salePrice for a more direct

			approach.
isNew	Boolean	Not Included	Excluded as it wasn't needed for the marketplace.
_id	String	Managed by CMS	Handled internally by Sanity CMS, so not explicitly included in the schema.

3. Migration steps and tools used.

Step 1: Prepare Data Schema

- Defined the data schema for products with fields such as `title`, `slug`, `price`, `salePrice`, `stock`, `reviewsCount`, and `description`.

Step 2: Create Data in NDJSON Format

- Created a list of products with relevant information in NDJSON format, each product on a new line, ensuring proper JSON structure and inclusion of `_type` property.

Step 3: Save Data File

- Saved the NDJSON data into a file named `products.ndjson`.

Step 4: Install Sanity CLI

Installed the Sanity CLI using the command:
bash
CopyEdit

```
npm install -g @sanity/cli
```

Step 5: Initialize Sanity Project

- Initialized the Sanity project or ensured access to an existing project where the data would be imported.

Step 6: Import Data

Used the Sanity CLI to import the NDJSON file into the project:
bash
CopyEdit

```
sanity dataset import products.ndjson production
```

- Replaced `production` with the target dataset name.

Step 7: Verify Data Import

- Logged into the Sanity studio to verify that all products were successfully imported and available within the dataset.

4. Screenshots

Below are the relevant screenshots to provide visual context for the integration process:

1. **API Calls**

```ts
import { Product } from "@/sanity.types";

export const fetchProducts = async (): Promise<Product[]> => {
  const baseURL = process.env.NEXT_PUBLIC_BASE_URL;
  const response = await fetch(`${baseURL}/api/products`, {
    cache: "no-store",
  });
  const products: Product[] = await response.json();
  return products;
};

```

2. Data rendered on the frontend

Today's Menu

Paya
PKR 400 ~~PKR 450~~

Karahi
PKR 650 ~~PKR 700~~

Aloo Keema
PKR 450 ~~PKR 500~~

Seekh Kebab
PKR 300 ~~PKR 350~~

Kofta Curry
PKR 500 ~~PKR 550~~

Gajar Ka Halwa
PKR 250 ~~PKR 300~~

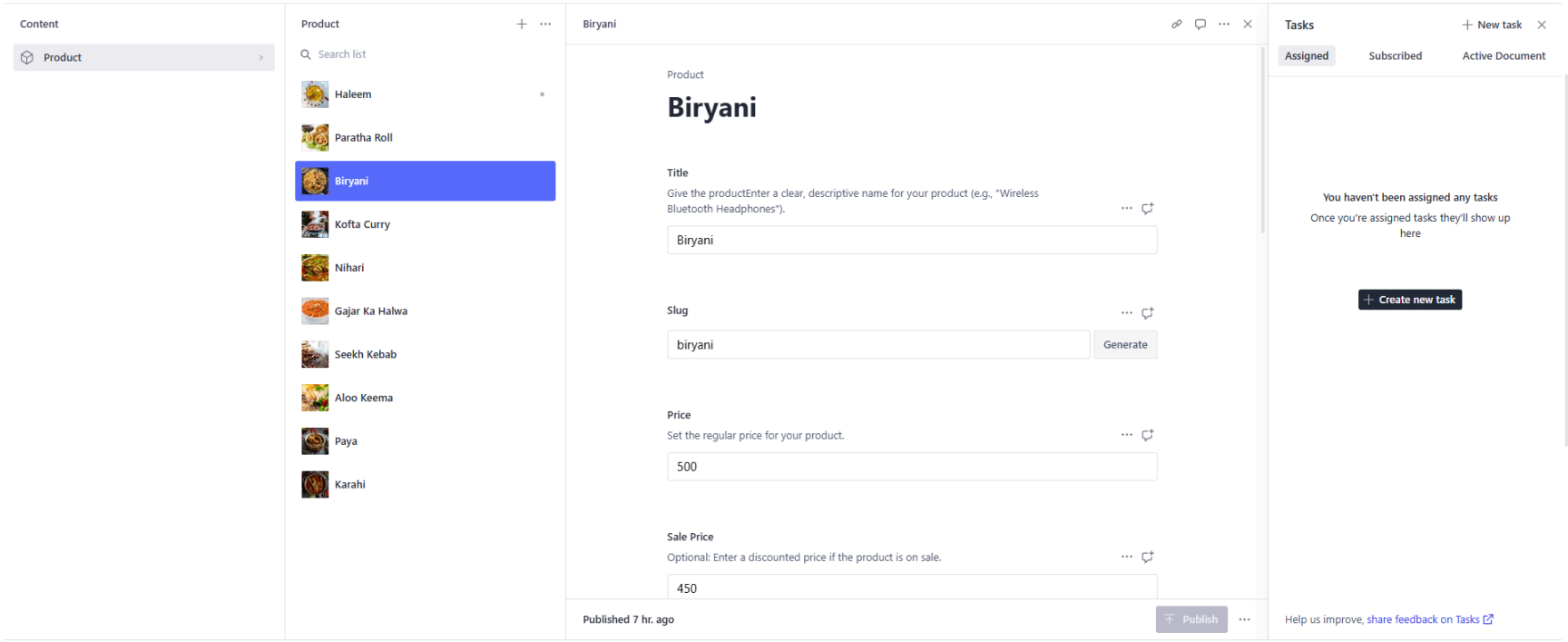
Nihari
PKR 550 ~~PKR 600~~

Paratha Roll
PKR 200 ~~PKR 250~~

Biryani
PKR 450 ~~PKR 500~~

3. Populated Sanity CMS fields.

Code Snippets for API Integration and Migration Scripts



Code snippets for API integration and migration scripts.

To fetch and display products, I created a reusable utility function named `fetchProducts`, which simplifies API calls and ensures consistent data retrieval across the application.

Here is the implementation:

Utility Function (helpers.ts):

```
import { Product } from "@/sanity.types";

export const fetchProducts = async (): Promise<Product[]> => {
  const baseURL = process.env.NEXT_PUBLIC_BASE_URL;
  const response = await fetch(`${baseURL}/api/products`, {
    cache: "no-store",
  });
  const products: Product[] = await response.json();
  return products;
};
```

Usage Example in `page.tsx`:

```
const products = await fetchProducts();
```

API Endpoint (API Code in Next.js):

```
import { client } from "@/sanity/lib/client";
import { NextRequest, NextResponse } from "next/server";

export async function GET(request: NextRequest) {
```

```
  try {
    const products = await client.fetch(
      '*[_type == "product"]{title,slug,price,salePrice,"imageUrl": image.asset->url}',
      {},
      { cache: "no-store" }
    );

    return NextResponse.json(products);
  } catch (error) {
    console.error("Error fetching products:", error);
    return NextResponse.json(
      { error: "Error fetching products" },
      { status: 500 }
    );
  }
}
```