# **Functional Deliverables:**

# **Product Listing**



PKR 450

PKR 500





Karahi PKR 650 PKR-700



Haleem
PKR 350 PKR-400



Aloo Keema PKR 450



Seekh Kebab PKR 300 PKR-350



Kofta Curry
PKR 500 PKR 550

#### **Product Detail**

Home > Shop Kofta Curry



# Kofta Curry

**PKR 550 PKR 500** 

★ ★ ★ ★ | 12 Customer Reviews

Kofta Curry is a rich and flavorful dish originating from South Asia, featuring tender, spiced meatballs (koftas) simmered in a luscious, aromatic curry sauce. The koftas, typically made from minced meat such as lamb, beef, or chicken, are blended with a mix of herbs, spices, and sometimes grated vegetables or lentils, creating a delicate and

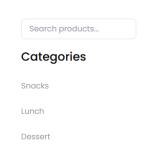


Units in Stock: 22 Category : Food

Additional Information Description Reviews [12]

Kofta Curry is a rich and flavorful dish originating from South Asia, featuring tender, spiced meatballs (koftas) simmered in a luscious, aromatic curry sauce. The koftas, typically made from minced meat such as lamb, beef, or chicken, are blended with a mix of herbs, spices, and sometimes grated vegetables or lentils, creating a delicate and juicy texture.

## **Search and Categories**



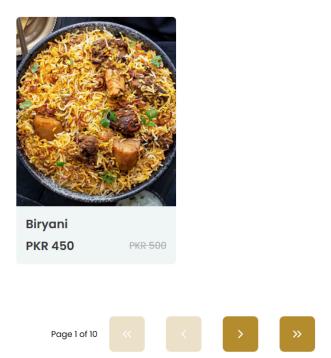






**PKR 650** PKR 700 **PKR 350** PKR 400

## **Pagination**



# **Code Deliverables:**

## **Fetching Products**

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

## utils/helpers.ts fetchProducts()

```
export const fetchProducts = async (query?: string): Promise<Product[]> => {
  const baseURL = process.env.NEXT_PUBLIC_BASE_URL;
  const url = new URL(`${baseURL}/api/products`);

if (query) {
  url.searchParams.append("query", query);
}
```

```
const response = await fetch(url.toString(), {
   cache: "no-store",
});

if (!response.ok) {
   throw new Error("Failed to fetch products");
}

const products: Product[] = await response.json();
   return products;
};
```

#### api/products/route.ts

```
import { client } from "@/sanity/lib/client";
import { NextRequest, NextResponse } from "next/server";
export async function GET(request: NextRequest) {
try {
   const { searchParams } = new URL(request.url);
   const query = searchParams.get("query");
   const sanityQuery = `
   *[_type == "product" && title match "${query || ''}*"]{
     title,
     slug,
     price,
     salePrice,
     "imageUrl": image.asset->url
 `;
   const products = await client.fetch(sanityQuery,
     {},
     { cache: "no-store" }
   );
```

#### **Product List Component**

## **Product List Component Definition and ProductCard Component**

### **Dynamic Routing**

```
interface Props {
  params: { slug: string };
}
```

```
export async function generateMetadata({ params }: Props) {
  const product = await fetchProduct(params.slug);
 if (!product) {
   return {
      title: "Product Not Found - My Store",
      description: "The product you are looking for does not exist.",
   };
  }
 return {
   title: `${product.title} - NextCart`,
   description: product.description || "Buy the best products at NextCart!",
 };
const ProductDetailPage = async ({ params }: Props) => {
  const product = await fetchProduct(params.slug);
 return (
   <article className={`product-${product.slug?.current}`}>
      <Breadcrumb currentPage={product.title!} />
      <ProductDetails product={product} />
   </article>
 );
};
export default ProductDetailPage;
```

## **Search and Category**

#### **Documentation**

### What Steps I took to build and integrate a component

- First, I define the component's interface, determining the necessary props and the expected output to ensure clarity in its functionality.
- Next, I develop the foundational structure of the component, focusing on its layout without integrating real data at this stage.
- Finally, I pass the required props and render the component on the page, verifying its integration and functionality.

In conclusion, this structured approach ensures that the component is well-defined, built with a clear purpose, and seamlessly integrated into the application, promoting maintainability and efficiency.

## What challenge I faced

I faced a challenge where the component was not showing updated data. After debugging, I discovered that the fetch request was using cached data because no specific configuration was set. To solve this, I added a configuration to the fetch request with cache: 'no-store', which resolved the issue by ensuring the latest data was fetched.