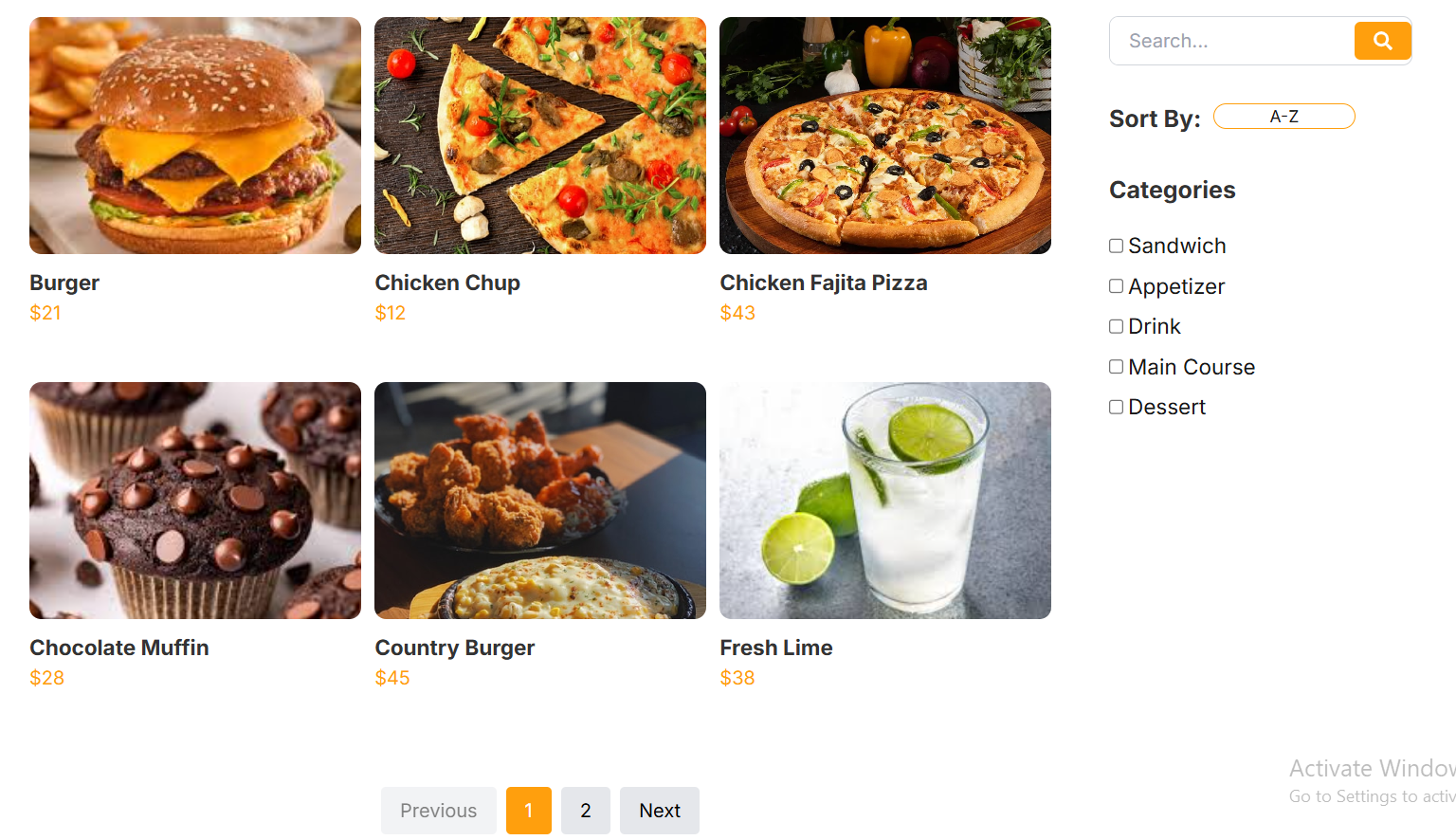
# Day 4 - Dynamic Frontend Components –

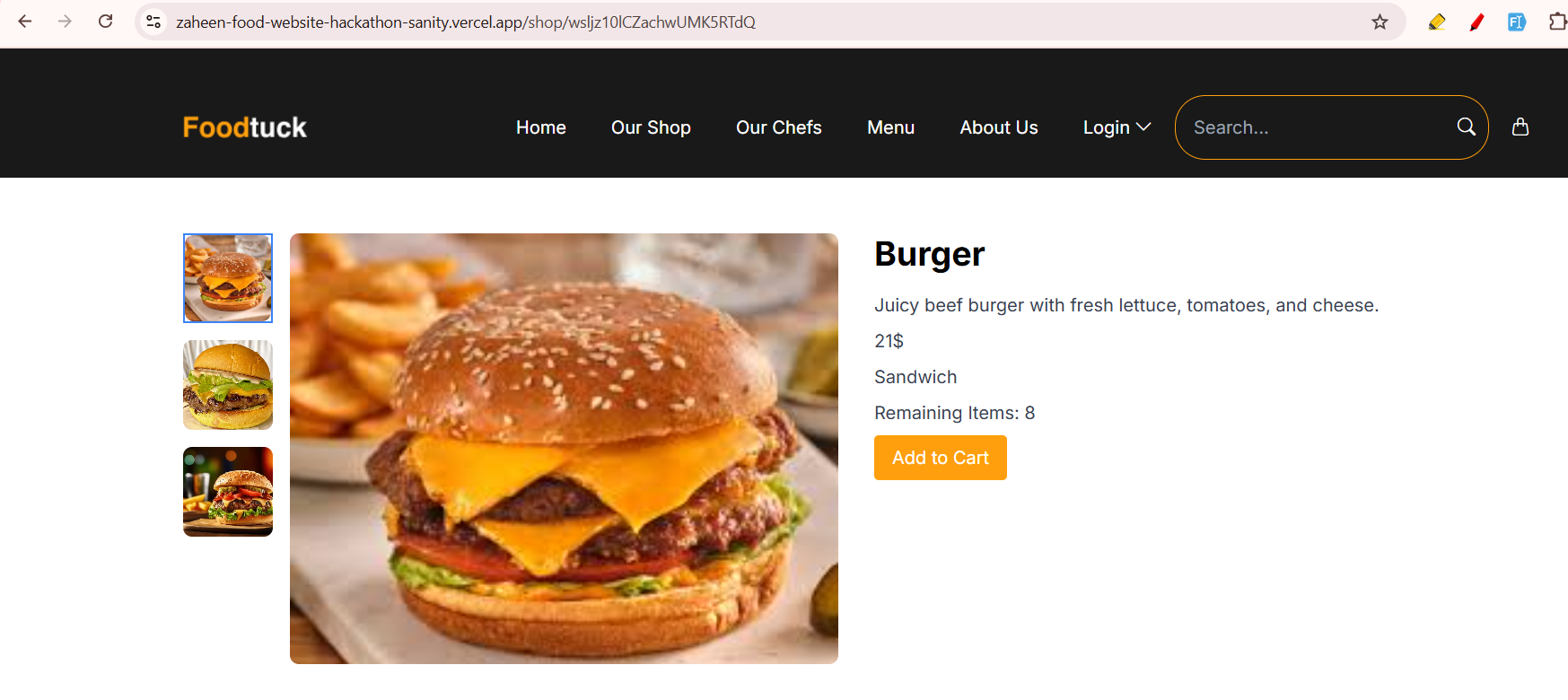
# Fusion Filling

Functional Deliverables:

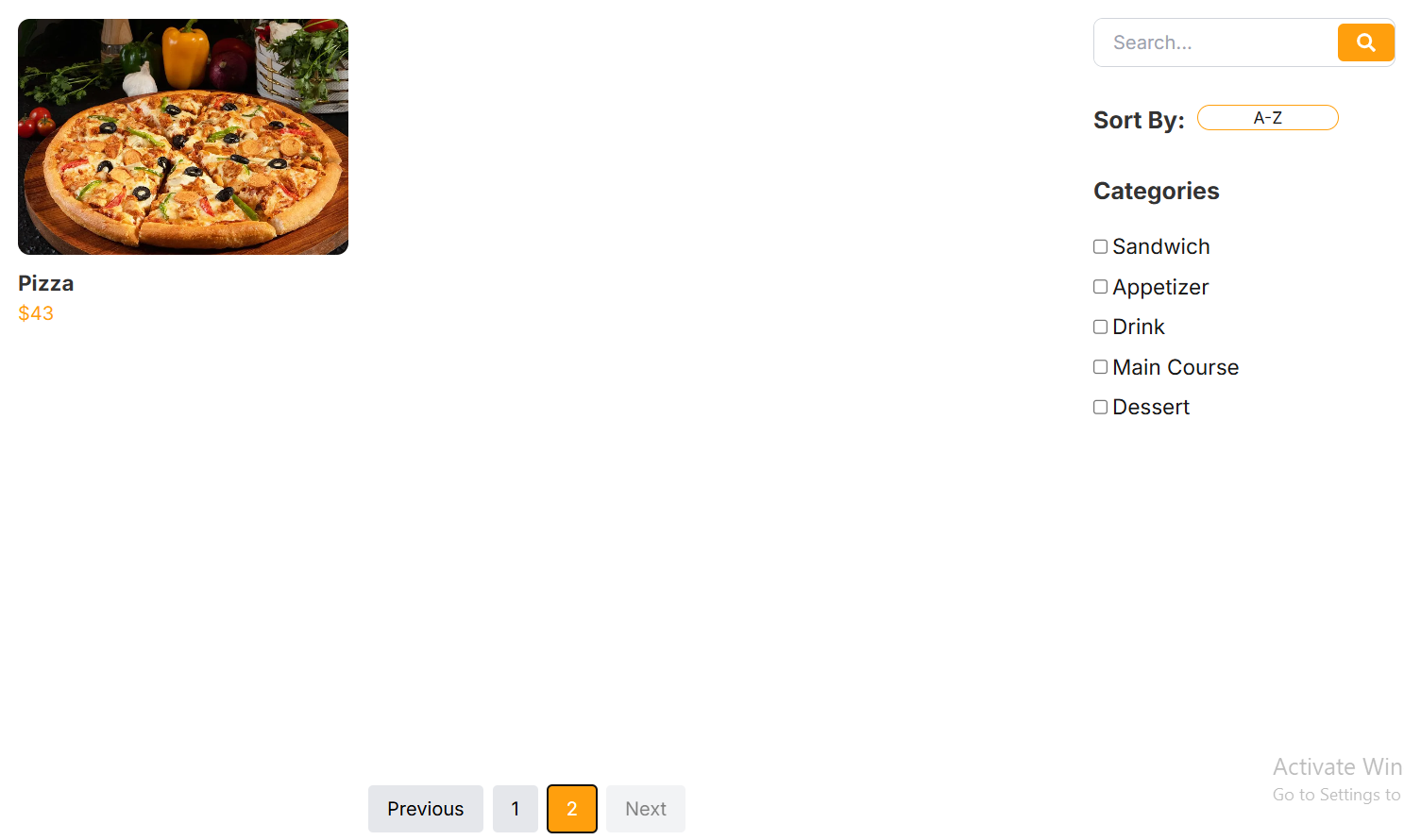
* Food Items Listing Page:

**Food items are dynamically fetched from Sanity Studio.**

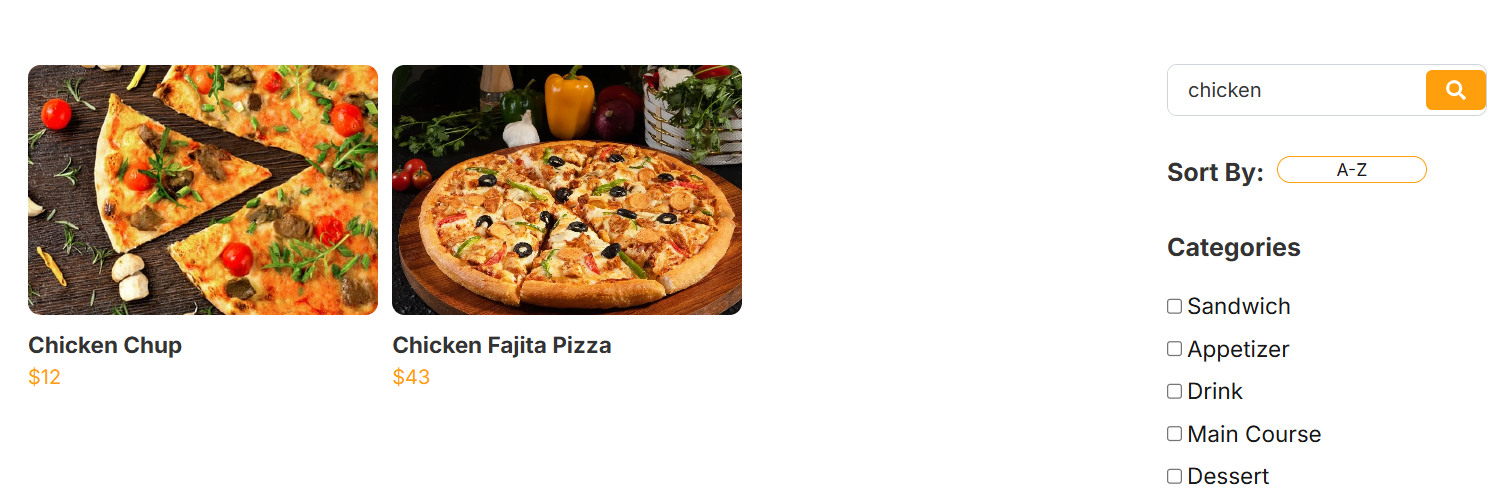
* Individual Food Item Detail Page:

**Food item detail page dynamically fetched from Sanity Studio.**

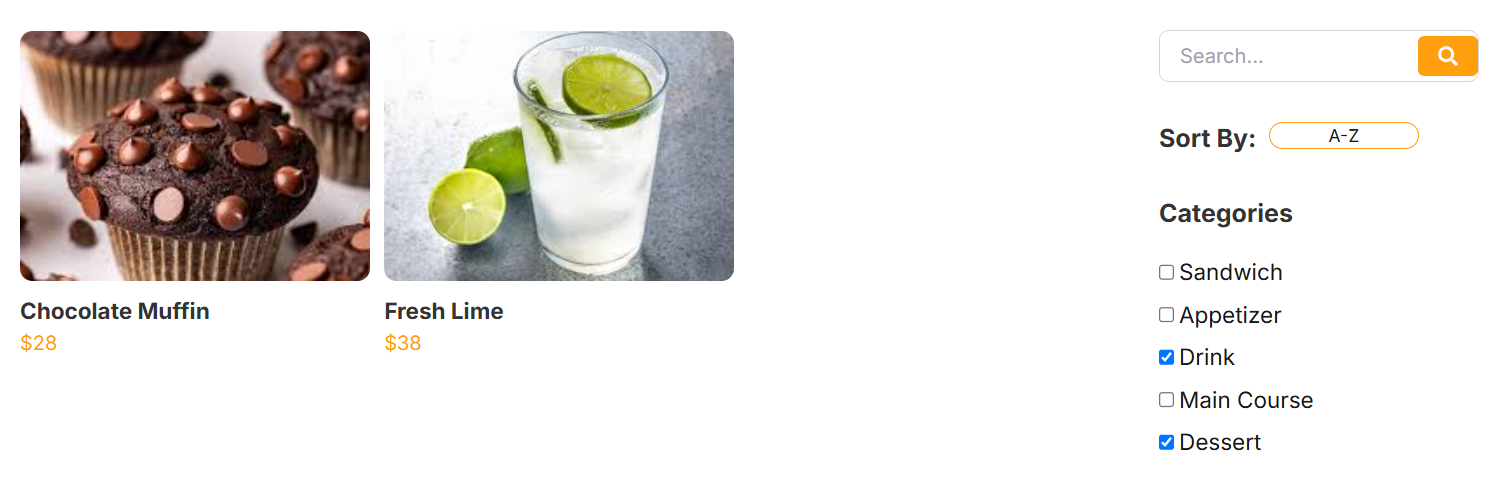
* Category filters, Search Bar, and Pagination:
* Pagination on Food Items Listing Page:



- Search using food item name:



- Filter using food item categories:



Code Deliverables:

* Food Item Listing Page:

"use client"

import { Checkbox } from "@heroui/checkbox";

import {

Dropdown,

DropdownTrigger,

DropdownMenu,

DropdownItem,

Button,

} from "@heroui/react";

import Image from "next/image";

import React, { useState, useEffect } from "react";

import { getCategoriesWithFoods } from "@/sanity/lib/data";

import { ICategoryWithFoods } from "@/sanity/lib/interfaces";

import CustomPagination from "../components/pagination";

import SearchBar from "../components/searchBar";

import { useSearchParams } from "next/navigation";

export function ShopContent() {

const [categoriesWithFoods, setCategoriesWithFoods] = useState<

ICategoryWithFoods[]

>([]);

const [selectedCategories, setSelectedCategories] = useState<string[]>([]);

const [sortOption, setSortOption] = useState<string>("az");

// const [searchQuery, setSearchQuery] = useState<string>(""); // State for search query

const [currentPage, setCurrentPage] = useState(1);

const itemsPerPage = 6;

const searchParams = useSearchParams();

const searchQueryFromURL = searchParams.get("search") || "";

const [searchQuery, setSearchQuery] = useState<string>(searchQueryFromURL);

useEffect(() => {

const fetchCategoriesWithFoods = async () => {

try {

const data = await getCategoriesWithFoods();

setCategoriesWithFoods(data);

} catch (error) {

console.error("Error fetching categories and foods:", error);

}

};

fetchCategoriesWithFoods();

}, []);

const handleCategorySelection = (categoryName: string) => {

setSelectedCategories((prevSelected) => {

if (prevSelected.includes(categoryName)) {

return prevSelected.filter((name) => name !== categoryName);

} else {

return [...prevSelected, categoryName];

}

});

};

const handleSortChange = (option: string) => {

setSortOption(option);

};

const filteredFoods = categoriesWithFoods

.filter(

(category) =>

selectedCategories.includes(category.name) ||

selectedCategories.length === 0

)

.flatMap((category) => category.foods)

.filter((food) =>

food.name.toLowerCase().includes(searchQuery.toLowerCase())

);

useEffect(() => {

setSearchQuery(searchQueryFromURL); // Update search query if the URL changes

}, [searchQueryFromURL]);

filteredFoods.sort((a, b) => {

if (sortOption === "az") {

return a.name.localeCompare(b.name);

} else if (sortOption === "za") {

return b.name.localeCompare(a.name);

} else if (sortOption === "lowhigh") {

return a.price - b.price;

} else if (sortOption === "highlow") {

return b.price - a.price;

}

return 0;

});

const startIndex = (currentPage - 1) \* itemsPerPage;

const currentFoods = filteredFoods.slice(

startIndex,

startIndex + itemsPerPage

);

const totalPages = Math.ceil(filteredFoods.length / itemsPerPage);

const handlePageChange = (page: number) => {

setCurrentPage(page);

};

return (

<>

<div>

<div

className=" pt-[150px] lg:pt-0 w-full bg-no-repeat bg-center flex justify-center "

style={{

backgroundImage: "url('/unsplash.png')",

backgroundSize: "cover",

backgroundPosition: "center top",

height: "300px",

}}

>

<div className="w-full max-w-5xl flex flex-col justify-center items-center text-white text-center py-16">

<p className="text-4xl sm:text-5xl md:text-6xl font-bold mb-4">

Our Shop

</p>

<div className="flex flex-col sm:flex-row items-center justify-center space-y-4 sm:space-y-0 sm:space-x-4">

<a href="/" className="text-xl sm:text-2xl md:text-3xl">

Home

</a>

<div className="flex items-center">

<Image

src="/Vector.png"

width={10}

height={10}

alt="Vector Icon"

/>

<a

href="/shop"

className="ml-2 text-xl sm:text-2xl md:text-3xl text-[#FF9F0D]"

>

Our Shop

</a>

</div>

</div>

</div>

</div>

<section className="max-w-[1320px] mx-auto py-[20px] lg:py-[50px] px-[20px] lg:px-[60px] text-black body-font bg-white">

<div className="md:grid md:grid-cols-4 gap-4 flex flex-col-reverse">

<div className="col-span-full md:col-span-3 p-4">

<div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-x-[10px] gap-y-4 min-h-[600px]">

{currentFoods.length > 0 ? (

currentFoods.map((food) => (

<a

href={`/shop/${food.\_id}`}

className="flex flex-col items-center md:items-start gap-y-[10px] border border-transparent hover:border-[#FF9F0D] rounded-lg transition duration-300"

key={food.\_id}

>

<div className="w-[100%] h-[200px]">

<img

src={food.imageUrl}

className="w-[100%] h-[100%] object-cover rounded-[10px]"

alt={food.name}

/>

</div>

<div className="flex flex-col items-center md:items-start w-full">

<h4 className="text-[18px] font-bold text-[#333333]">

{food.name}

</h4>

<div className="flex">

<p className="text-[16px] font-normal text-[#FF9F0D]">

${food.price}

</p>

</div>

</div>

</a>

))

) : (

<div className="col-span-full text-center text-gray-500">

No food items found.

</div>

)}

</div>

<div className="flex gap-4 justify-center items-center mt-[50px]">

<CustomPagination

currentPage={currentPage}

totalPages={totalPages}

onChange={handlePageChange}

/>

</div>

</div>

<div className="flex gap-y-[30px] flex-col items-center md:items-start col-span-full md:col-span-1 p-4 text-[#333333]">

<div className="w-full max-w-md">

<SearchBar query={searchQuery} setQuery={setSearchQuery} />

</div>

<div className="flex gap-[10px] items-center md:items-start">

<h3 className="font-bold text-[20px] p-0 m-0">Sort By:</h3>

<div className="sort-dropdown">

<Dropdown className="">

<DropdownTrigger className="min-w-[120px] hover:bg-[#FF9F0D] hover:text-white border border-[#FF9F0D]">

<Button variant="bordered">

{sortOption === "az" && "A-Z"}

{sortOption === "za" && "Z-A"}

{sortOption === "lowhigh" && "Low-High"}

{sortOption === "highlow" && "High-Low"}

</Button>

</DropdownTrigger>

<DropdownMenu

aria-label="Static Actions"

className="bg-white text-[#FF9F0D] border border-[#FF9F0D] min-w-[120px]"

>

<DropdownItem

key="az"

onPress={() => handleSortChange("az")}

className="hover:bg-[#FF9F0D] hover:text-white"

>

A-Z

</DropdownItem>

<DropdownItem

key="za"

onPress={() => handleSortChange("za")}

className="hover:bg-[#FF9F0D] hover:text-white"

>

Z-A

</DropdownItem>

<DropdownItem

key="lowhigh"

onPress={() => handleSortChange("lowhigh")}

className="hover:bg-[#FF9F0D] hover:text-white"

>

Low-High

</DropdownItem>

<DropdownItem

key="highlow"

onPress={() => handleSortChange("highlow")}

className="hover:bg-[#FF9F0D] hover:text-white"

>

High-Low

</DropdownItem>

</DropdownMenu>

</Dropdown>

</div>

</div>

<div className="flex flex-col items-center md:items-start w-full">

<h3 className="font-bold text-[20px] ">Categories</h3>

<div className="flex flex-col sm:flex-row md:flex-col justify-around md:justify-start my-[15px] w-unset sm:w-full">

{categoriesWithFoods.map((category) => (

<Checkbox

key={category.\_id}

checked={selectedCategories.includes(category.name)}

onChange={() => handleCategorySelection(category.name)}

>

<p className="m-0 p-0 py-[5px] text-[18px] font-normal">

{category.name}

</p>

</Checkbox>

))}

</div>

</div>

</div>

</div>

</section>

</div>

;

</>

);

}

* Food item Detail Page:

import { getFoodItemById } from "@/sanity/lib/data";

import ImageGallery from "@/app/components/imageGallery";

import { IFoodItem } from "@/sanity/lib/interfaces";

import AddToCartButton from "@/app/components/addToCartButton";

interface ProductPageProps {

params: {

slug: string;

};

}

export default async function FoodDetail({ params }: ProductPageProps) {

const { slug } = params; // Get the slug from the URL

const foodItem: IFoodItem = await getFoodItemById(slug);

if (!foodItem) {

return <div>Product not found</div>; // Handle invalid slug

}

return (

<div className="max-w-[1320px] pt-[150px] mx-auto py-[20px] lg:py-[50px] px-[20px] lg:px-[60px] text-black body-font bg-white">

<div className="grid grid-cols-1 md:grid-cols-2 gap-8">

{/\* Image Gallery \*/}

<div>

<ImageGallery

mainImageUrl={foodItem.mainImageUrl}

images={foodItem.images}

/>

</div>

{/\* Product Details \*/}

<div>

<h1 className="text-3xl font-bold mb-4">{foodItem.name}</h1>

<p className="text-gray-700 mb-2"> {foodItem.description}</p>

<p className="text-gray-700 mb-2"> {foodItem.price}$</p>

<p className="text-gray-700 mb-2"> {foodItem.category}</p>

<p className="text-gray-700 mb-2">

Remaining Items: {foodItem.stock}

</p>

{/\* <button className="bg-[#FF9F0D] text-white px-4 py-2 rounded">

Add to Cart

</button> \*/}

<AddToCartButton product={foodItem} />

</div>

</div>

</div>

);

}

* Data Fetching Scripts:
* Get all categories of food items:

export const getAllCategories = async () => {

try {

const getAllCategoriesQuery = `\*[\_type == "category" && available == true] {

\_id,

name,

"imageUrl": image.asset->url,

available

}

`;

const categories: ICategory[] = await client.fetch(getAllCategoriesQuery, {}, { next: { revalidate: 1800 } });

return categories;

} catch (error) {

console.log(error);

throw new Error("Failed to fetch categories. Please try again later.");

}

};

* Get all categories with all food items:

export const getCategoriesWithFoods = async (): Promise<ICategoryWithFoods[]> => {

try {

const query = `\*[\_type == "category" && available == true] {

\_id,

name,

"imageUrl": image.asset->url,

available,

"foods": \*[\_type == "food" && references(^.\_id) && available == true] {

\_id,

name,

price,

"imageUrl": image.asset->url,

description,

available

}

}`;

const categoriesWithFoods: ICategoryWithFoods[] = await client.fetch(query, {}, { next: { revalidate: 1800 } });

return categoriesWithFoods;

} catch (error) {

console.error("Error fetching categories with foods:", error);

throw new Error("Failed to fetch categories with foods. Please try again later.");

}

};

* Get food item by its id:

export const getFoodItemById = async (slug: string) => {

const query = `\*[\_type == "food" && \_id == $slug][0] {

\_id,

name,

price,

"category": category->name,

stock,

description,

"mainImageUrl": image.asset->url, // Resolve the main image URL

"images": images[].asset->url // Resolve the array of image URLs

}`;

const foodItem:IFoodItem = await client.fetch(query, { slug }, { next: { revalidate: 0 } });

if (foodItem) {

const mainImageUrl = foodItem.mainImageUrl || '/default-image.jpg';

foodItem.images = [mainImageUrl, ...(foodItem.images || [])];

}

return foodItem;

};

Documentation:

#### **1. Steps Taken to Build and Integrate Components:**

The development of the food e-commerce website involved several key steps to build and integrate components effectively. Below is a summary of the process:

##### **a. Setting Up the Project**

* **Next.js Framework**: The project was built using Next.js for server-side rendering (SSR), static site generation (SSG), and client-side rendering (CSR). This ensured optimal performance and SEO.
* **Sanity CMS**: Sanity was used as the headless CMS to manage food items, categories, and orders. The sanity/client library was used to fetch data from Sanity.

##### **b. Building the Shop Page**

* **Dynamic Data Fetching**: The getCategoriesWithFoods function was created to fetch food categories and their associated items from Sanity. This data was used to populate the shop page.
* **Search and Filter Functionality**:
  + A SearchBar component was implemented to allow users to search for food items by name.
  + Filters for sorting (A-Z, Z-A, Low-High, High-Low) and category selection were added using checkboxes and a dropdown menu.
* **Pagination**: A CustomPagination component was built to handle pagination for the food items, ensuring a smooth user experience.

##### **c. Implementing the Checkout Process**

* **Cart Management**: Redux was used to manage the cart state. Actions like adding items, removing items, and clearing the cart were implemented.
* **Checkout API**: A /api/checkout route was created to handle order creation in Sanity. The API validates the order data, creates the order, and updates the stock of each product.
* **Order Confirmation**: After a successful checkout, the user is redirected to a /checkout-success page, where the order ID is displayed.

##### **d. Integrating useSearchParams**

* The useSearchParams hook was used to handle search queries in the shop page. To avoid static rendering issues, the component was wrapped in a Suspense boundary.

##### **e. Styling and UI Components**

* **Component Library**: The project used a custom component library (@heroui) for UI elements like checkboxes, dropdowns, and buttons.
* **Responsive Design**: The layout was designed to be responsive, ensuring a seamless experience across multiple devices.

#### **2. Challenges Faced and Solutions Implemented:**

During the development process, several challenges were encountered and addressed:

##### **a. Static Rendering Issues with useSearchParams**

* **Challenge**: The useSearchParams hook caused errors during static rendering because it relies on client-side data.
* **Solution**: The component using useSearchParams was wrapped in a Suspense boundary with a fallback (e.g., Loading...). This deferred rendering until the necessary data was available.

##### **b. Dynamic Order ID Display**

* **Challenge**: After checkout, the order ID needed to be displayed on the /checkout-success page. However, the order ID was only available after the API call.
* **Solution**: The /api/checkout route was updated to return the order ID in the response. The order ID was then passed to the /checkout-success page as a query parameter.

##### **c. Managing Cart State**

* **Challenge**: Managing the cart state across different pages (e.g., shop, checkout) required a centralized state management solution.
* **Solution**: Redux was implemented to manage the cart state. Actions like addToCart, removeFromCart, and clearCart were created to handle cart updates.

##### **d. Fetching Data from Sanity**

* **Challenge**: Fetching nested data (e.g., food items within categories) from Sanity required complex GROQ queries.
* **Solution**: The getCategoriesWithFoods function was created to fetch categories and their associated food items in a single query. This reduced the number of API calls and improved performance.

##### **e. Handling Stock Updates**

* **Challenge**: After an order was placed, the stock of each product needed to be updated in Sanity.
* **Solution**: A separate /api/updateStock route was created to handle stock updates. This route was called for each item in the order after the order was successfully created.

#### **3. Best Practices Followed During Development:**

To ensure a maintainable and scalable codebase, the following best practices were followed:

##### **a. Modular Component Design**

* Components like SearchBar, CustomPagination, and CountryDropdown were built as reusable, modular components. This promoted code reusability and made the codebase easier to maintain.

##### **b. TypeScript for Type Safety**

* TypeScript was used throughout the project to enforce type safety. Interfaces like ICategoryWithFoods were defined to ensure consistent data structures.

##### **c. Separation of Concerns**

* Logic for data fetching, state management, and UI rendering was separated into distinct modules. For example:
  + Data fetching was handled in the getCategoriesWithFoods function.
  + State management was handled by Redux.
  + UI rendering was handled by React components.

##### **d. Error Handling**

* Robust error handling was implemented at every level:
  + API routes included try-catch blocks to handle errors gracefully.
  + User-friendly error messages were displayed using toast notifications.

##### **e. Performance Optimization**

* **Pagination**: The shop page implemented pagination to limit the number of items rendered at once, improving performance.
* **Lazy Loading**: Images were lazy-loaded using the next/image component to reduce initial page load time.