Day 4 Building Dynamic Frontend Components for FoodTuck Q-Commerce Website

Date: 19/01/2025

Prepared By: Arishah

Objective

The objective of this task was to design and implement dynamic frontend components for a Q-commerce foodtuck website to enhance user interaction and simplify ordering processes

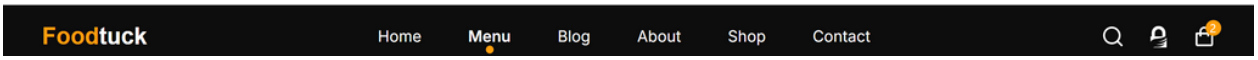
Key Learning Outcomes

- Used the Next.js framework for building a dynamic and performant Q-commerce website, with features like server-side rendering (SSR).
- Applied Tailwind CSS for creating responsive and flexible UI designs using utility-first classes..
- Integrated ShadCN components for reusable and customizable frontend elements, improving design consistency.
- Implemented state management to handle dynamic content and user interactions efficiently, enhancing the overall user experience.
- Developed problem-solving skills while managing dynamic states and optimizing performance across the application.

Components Built

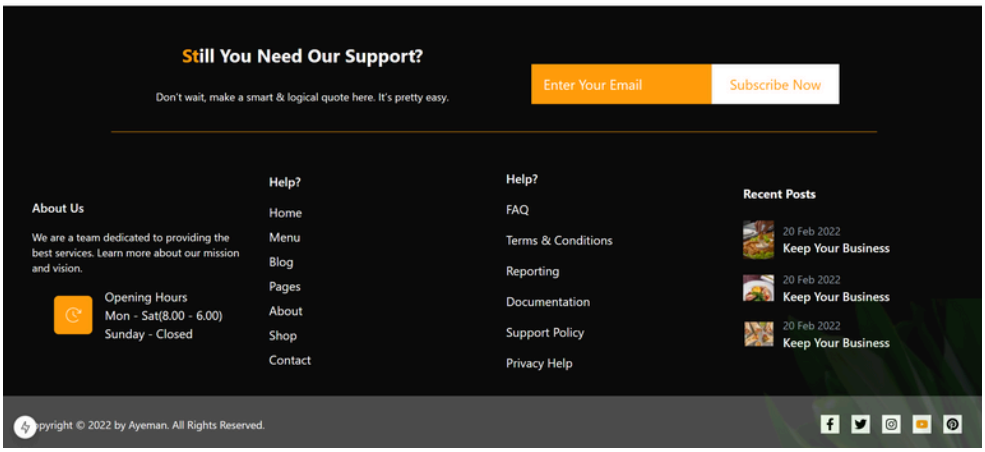
1. Header Component

The Header component contains the navigation menu, logo, and other important elements like the search bar or user profile. It's the first section of the website that users interact with.



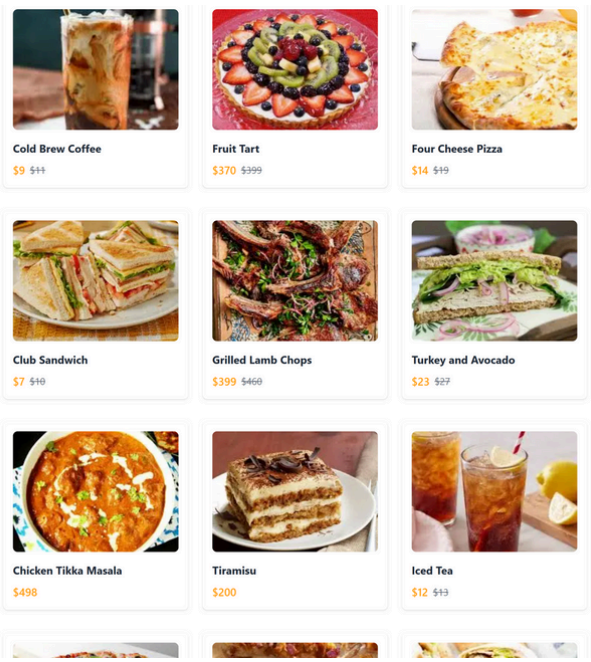
2. Footer Component

The Footer component includes the website’s contact information, social media links, and copyright notices. It’s typically placed at the bottom of every page.



3: Food Item Listing Component (Dynamic Data Fetching)

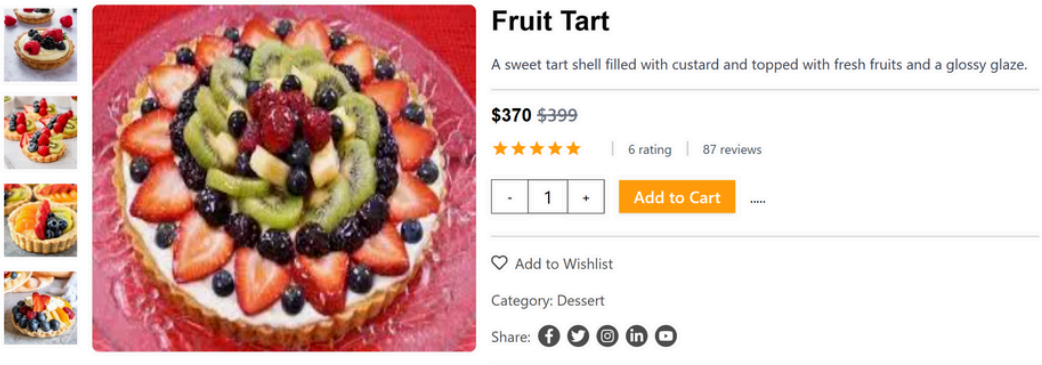
This component fetches food items dynamically from Sanity using Groq and displays them in a list or grid. Each food item contains details like name, price, and image, allowing users to browse the menu.

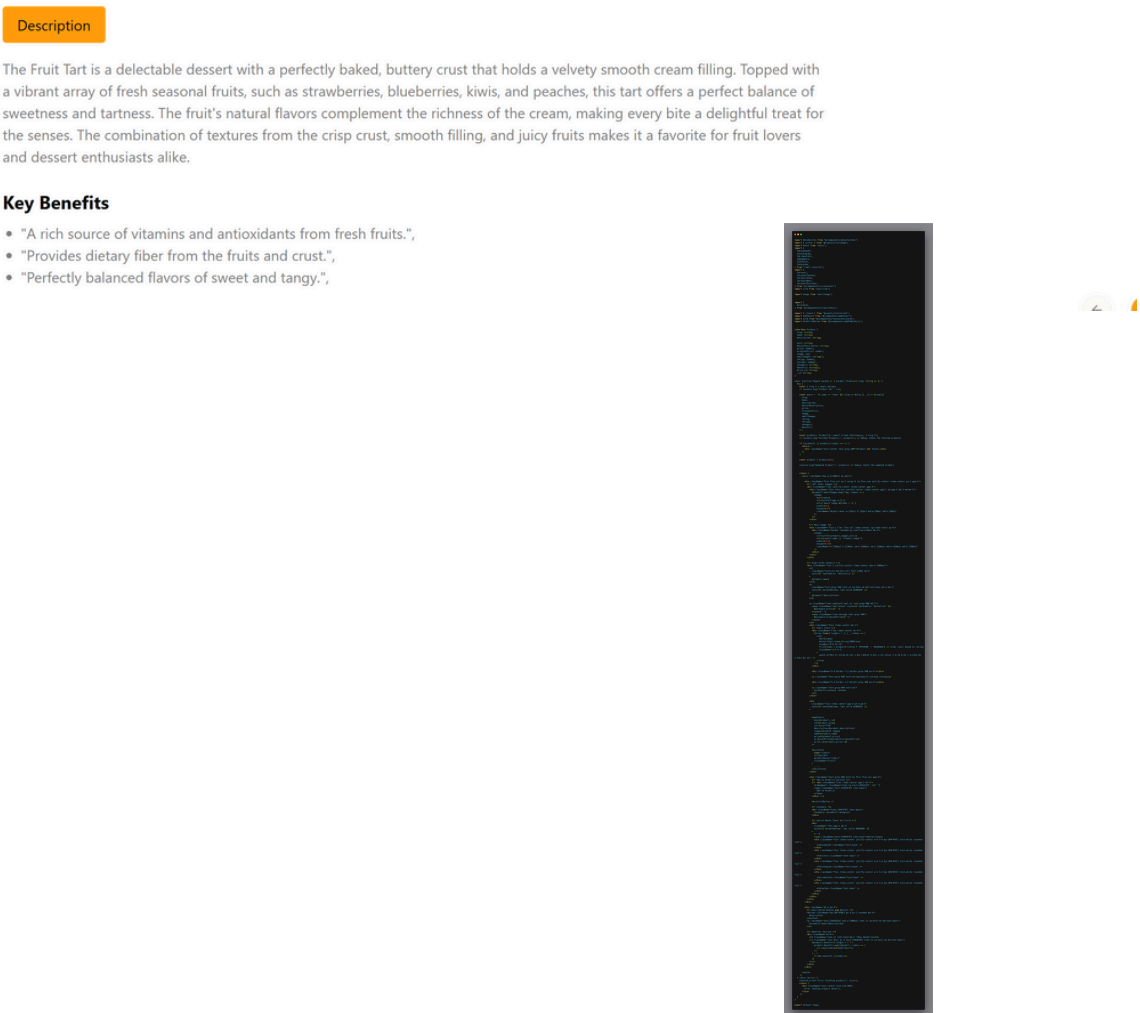



```
interface Food {
  _id: string;
  name: string;
  category: string;
  price: number;
  originalPrice: number;
  image: any;
  tags: string[];
  slug: string;
}

export default function Shop() {

const fetchFoods = async () => {
    try {
      const query = `*[_type == "food"]{_id, name, category, price, originalPrice, image, slug}`;
      const foods = await client.fetch(query);
      setAllFoods(foods);
    } catch (error) {
      console.error("Failed to fetch foods:", error);
      setAllFoods([]);
    }
  };
  return (
    <section className="order-2 md:order-1 flex-1 grid grid-cols-1 sm:grid-cols-2 md:grid-cols-2 lg:grid-
cols-3 gap-6">
          {currentFoods.map((food) => (
            <Link href={`/shop/card/${food.slug}`} key={food.slug}>
              <div className="p-4 bg-white rounded-lg shadow hover:shadow-lg">
                <Image
                  src={
                    food.image?.asset?._ref
                      ? urlFor(food.image).url()
                      : "/path/to/default-image.jpg"
                  }
                  alt={food.name}
                  width={300}
                  height={200}
                  className="rounded-lg object-cover w-full h-[200px]"
                />
                <h3 className="mt-4 text-lg font-bold text-gray-800">{food.name}</h3>
                <div className="flex items-center gap-2 mt-2">
                  {food.price ? (
                    <>
                      <p className="text-[#FF9F0D] text-lg font-semibold">
                        ${food.price}
                      </p>
                      <p className="line-through text-gray-500">
                        ${food.originalPrice}
                      </p>
                    </>
                  ) : (
                    <p className="text-[#FF9F0D] text-lg font-semibold">
                      ${food.originalPrice}
                    </p>
                  )}
                </div>
              </div>
            </Link>
          ))}
        </section>

    )
}
```

4. Food Item Detail Component (Dynamic Routing)

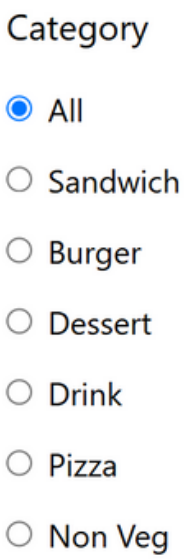
This component uses dynamic routing in Next.js to display the details of a selected food item. It fetches data dynamically from Sanity for each item, including the name, image, description, benefits, category, tags, price, and functionalities like Add to Cart, Increment, and Decrement.





5. Category Component

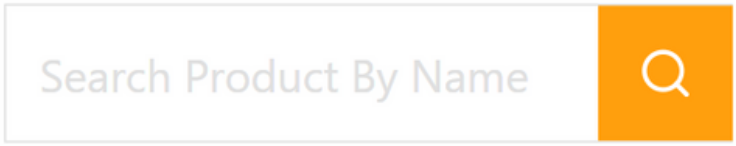
The Category Component lets users filter food items based on different categories such as Sandwich, Burger, Dessert, Drink, Pizza, and Non-Veg. This component helps users navigate through the food items more efficiently by showing them items that belong to specific categories.



```tsx
// components/CategoryFilter.tsx
interface CategoryFilterProps {
    selectedCategory: string;
    onCategoryChange: (category: string) => void;
  }

  const staticCategories = [
    "Sandwich",
    "Burger",
    "Dessert",
    "Drink",
    "Pizza",
    "Non Veg",
  ];

  const CategoryFilter = ({ selectedCategory, onCategoryChange }: CategoryFilterProps) => {
    return (
      <div>
        <h3 className="text-lg text-black mb-4">Category</h3>
        <div className="space-y-3">
          <div className="flex items-center">
            <input
              type="radio"
              id="All"
              name="category"
              checked={selectedCategory === "All"}
              onChange={() => onCategoryChange("All")}
              className="mr-2"
            />
            <label htmlFor="All" className="text-black">
              All
            </label>
          </div>
          {staticCategories.map((category) => (
            <div key={category} className="flex items-center">
              <input
                type="radio"
                id={category}
                name="category"
                checked={selectedCategory === category}
                onChange={() => onCategoryChange(category)}
                className="mr-2"
              />
              <label htmlFor={category} className="text-black">
                {category}
              </label>
            </div>
          ))}
        </div>
      </div>
    );
  };

  export default CategoryFilter;
```

6. Search Bar Component

The Search Bar Component enables users to search for food items by name. As users type in the search bar, the list of food items dynamically updates to display matches based on the entered query. This helps users quickly find specific dishes from the menu.




```tsx
// components/SearchBar.tsx
import { FiSearch } from "react-icons/fi";

interface SearchBarProps {
  search: string;
  onSearchChange: (value: string) => void;
}

const SearchBar = ({ search, onSearchChange }: SearchBarProps) => {
  return (
    <div className="mb-6">
      <div className="relative">
        <input
          type="text"
          placeholder="Search Product By Name"
          value={search}
          onChange={(e) => onSearchChange(e.target.value)}
          className="w-full p-3 pr-12 border border-[#E0E0E0] placeholder-[#E0E0E0] text-black
focus:border-gray-300 focus:ring-gray-300 focus:ring-1 focus:outline-none"
        />
        <div className="bg-[#FF9F0D] absolute right-0 top-1/2 transform -translate-y-1/2 flex items-
center justify-center w-12 h-12">
          <FiSearch className="text-white text-xl" />
        </div>
      </div>
    </div>
  );
};

export default SearchBar;
```

7. Cart Component (with use-shopping-cart Library and Increment/Decrement Functionality)

The Cart Component allows users to add food items to their cart, modify the quantities using Increment and Decrement buttons, and manage their cart's contents. The use-shopping-cart library is used to handle the cart state, and the Increment and Decrement buttons enable users to easily adjust the number of items they want to order. The total price updates dynamically based on the quantity of each item.

8. Pagination Component

The Pagination Component divides the food items (or products) into pages, letting users easily browse through different items without overwhelming them with too many choices at once. It typically includes:

1. Next and Previous Buttons: These buttons allow users to navigate between pages.
2. Page Numbers: Users can directly click on a specific page number to jump to that page.
3. Dynamic Rendering: The content on the page changes based on the selected page, showing a subset of food items.


```
]interface PaginationProps {
    currentPage: number;
    totalPages: number;
    onPageChange: (page: number) => void;
  }

  const Pagination = ({ currentPage, totalPages, onPageChange }: PaginationProps) => {
    const maxVisiblePages = 3;

    const getVisiblePages = () => {
      const half = Math.floor(maxVisiblePages / 2);
      const startPage = Math.max(1, currentPage - half);
      const endPage = Math.min(totalPages, startPage + maxVisiblePages - 1);
      return Array.from({ length: endPage - startPage + 1 }, (_, i) => startPage + i);
    };

    return (
      <div className="flex justify-center items-center space-x-4 my-10">
        <button
          onClick={() => onPageChange(currentPage - 1)}
          disabled={currentPage === 1}
          className="px-3 py-2 bg-gray-200 text-[#ff9f0d] rounded disabled:opacity-50"
        >
          «
        </button>
        {getVisiblePages().map((page) => (
          <button
            key={page}
            onClick={() => onPageChange(page)}
            className={`px-3 py-2 ${currentPage === page ? "bg-orange-500 text-white" : "bg-gray-100
text-[#ff9f0d]"}`}
          >
            {page}
          </button>
        ))}
        <button
          onClick={() => onPageChange(currentPage + 1)}
          disabled={currentPage === totalPages}
          className="px-3 py-2 bg-gray-200 text-[#ff9f0d] rounded disabled:opacity-50"
        >
          »
        </button>
      </div>
    );
  };

  export default Pagination;
```

9. Filter Component

TThe Filter Component allows users to filter food items based on their price range. Users can select a price range (from $0 to $1000), and the food items (displayed as cards) will be filtered to show only those within the selected price range. Each food item card includes the name, price, and image.

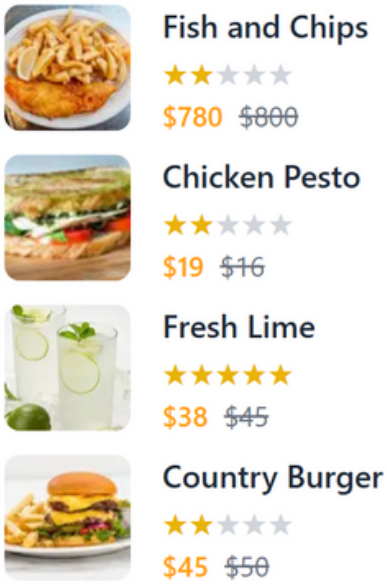
Key Features:

1. Price Range Filter: Users can choose a range (from $0 to $1000) to filter food items.
2. Dynamic Filtering: The list of food cards updates automatically to display only items within the selected price range.
3. Food Cards: Each food card includes a name, price, and image, making it easy for users to browse the items.

```
"use client";

import React, { useState } from "react";

interface PriceRangeProps {
  onRangeChange: (min: number, max: number) => void;
}

export default function PriceRange({ onRangeChange }: PriceRangeProps) {
  const [minPrice, setMinPrice] = useState(0);
  const [maxPrice, setMaxPrice] = useState(800);

  const handleSliderChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    const { name, value } = event.target;
    if (name === "min") {
      const min = Number(value);
      setMinPrice(min);
      onRangeChange(min, maxPrice);
    } else {
      const max = Number(value);
      setMaxPrice(max);
      onRangeChange(minPrice, max);
    }
  };

  return (
    <div className="price-range p-4 border rounded-lg">
      <h3 className="text-orange-500 font-bold mb-2">Filter by Price</h3>
      <div className="flex items-center gap-2">
        <input
          type="range"
          name="max"
          min="0"
          max="1000"
          value={maxPrice}
          onChange={handleSliderChange}
          className="w-full accent-orange-500"
        />
      </div>
      <div className="flex justify-between mt-2">
        <span className="text-sm">From: ${minPrice}</span>
        <span className="text-sm">To: ${maxPrice}</span>
      </div>
    </div>
  );
}
```

10: Related Food by Tags Component

This component was created to display food items that are related to the current product based on shared tags. I fetched the data from Sanity using the tags field, which holds the tags associated with each food item. By matching the tags, this component dynamically displays related food items.



```
import { useEffect, useState } from "react";
import { client } from "@/sanity/lib/client";
import { urlFor } from "@/sanity/lib/image";
import Image from "next/image";
import Link from "next/link";

interface Food {
  _id: string;
  name: string;
  category: string;
  price: number;
  originalPrice: number;
  image: any;
  tags: string[];
  slug: string;
  rating: number;
}

const LatestProducts = () => {
  const [products, setProducts] = useState<Food[]>([]);

  useEffect(() => {
    const fetchLatestFoodsByTag = async () => {
      try {
        const query = `*[_type == "food" && "latest" in tags]{slug, name, price, originalPrice, image,
rating} | order(_createdAt desc)`;
        const foods = await client.fetch(query);
        setProducts(foods);
      } catch (error) {
        console.error("Failed to fetch foods:", error);
        setProducts([]);
      }
    };

    fetchLatestFoodsByTag();
  }, []);

  const renderStars = (rating: number) => {
    let stars = [];
    for (let i = 0; i < 5; i++) {
      if (i < rating) {
        stars.push(
          <span key={i} className="text-yellow-500">
            ★
          </span>
        );
      } else {
        stars.push(
          <span key={i} className="text-gray-300">
            ★
          </span>
        );
      }
    }
    return stars;
  };

  return (
    <div className="flex flex-wrap gap-2 py-6">
      {products.map((food) => (
                  <Link href={`/shop/card/${food.slug}`} key={food.slug}>

        <div
          key={food._id}
          className="flex flex-row justify-center items-center"
        >
          <div className="">
            <Image
              src={
                food.image?.asset?._ref
                  ? urlFor(food.image).url()
                  : "/path/to/default-image.jpg"
              }
              alt={food.name}
              width={50}
              height={50}
              className="w-16 h-16 object-cover rounded-lg"
            />
          </div>

          <div className="ml-4 ">
            <h3 className="font-semibold text-base text-gray-800">{food.name}</h3>

            <div className=" flex">{renderStars(food.rating)}</div>

            <div className=" flex justify-start items-center gap-2">
              {food.price ? (
                <>
                  <p className="text-[#FF9F0D] text-sm font-semibold">
                    ${food.price}
                  </p>
                  <p className="line-through text-sm text-gray-500">
                    ${food.originalPrice}
                  </p>
                </>
              ) : (
                <p className="text-[#FF9F0D] text-sm font-semibold">
                  ${food.originalPrice}
                </p>
              )}
            </div>
          </div>
        </div>
        </Link>
      ))}
    </div>
  );
};

export default LatestProducts;
```

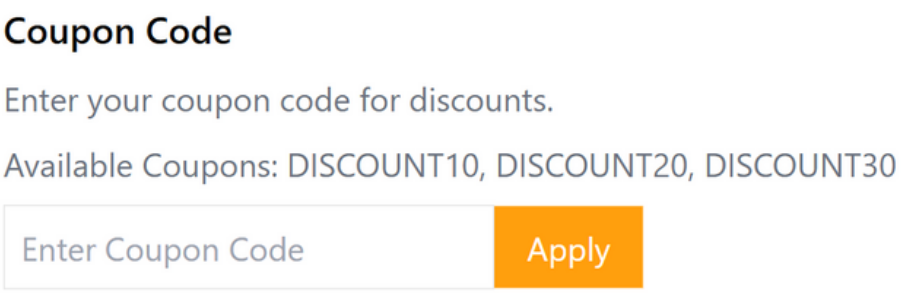
11. Discount Section with Voucher Code Functionality

I have created this section where users can enter a voucher code to apply a discount to their order. The system checks if the voucher code is valid and calculates the discount accordingly. If the voucher code is valid, it reduces the total price, giving the user a special offer or promotion.

Key Features:

1. Voucher Code Input: A text input where users can enter the voucher code.
2. Discount Validation: The component checks if the entered voucher code is valid and applies the corresponding discount.
3. Dynamic Price Update: The total price of the order is updated in real-time once the voucher is applied.
4. Error Handling: If the voucher code is invalid, an error message is shown to the user.

.




```tsx
import React from "react";

type DiscountSectionProps = {
  couponCode: string;
  setCouponCode: React.Dispatch<React.SetStateAction<string>>;
  handleCouponApply: () => void;
};

const DiscountSection: React.FC<DiscountSectionProps> = ({
  couponCode,
  setCouponCode,
  handleCouponApply,
}) => {
  return (
    <div className="flex flex-col max-w-full lg:max-w-[500px] mb-6 lg:mb-0">
      <label htmlFor="coupon" className="text-lg font-semibold">
        Coupon Code
      </label>
      <p className="text-sm sm:text-base text-gray-500 py-2">Enter your coupon code for discounts.</p>
      <p className="text-sm sm:text-base text-gray-500">
        Available Coupons: DISCOUNT10, DISCOUNT20, DISCOUNT30
      </p>
      <div className="flex justify-start items-center  mt-2">
        <input
          id="coupon"
          type="text"
          className="p-2 border rounded-none w-full lg:w-60"
          placeholder="Enter Coupon Code"
          value={couponCode}
          onChange={(e) => setCouponCode(e.target.value)}
        />
        <button
          onClick={handleCouponApply}
          className="bg-[#FF9F0D] text-white px-4 py-2 rounded-none"
        >
          Apply
        </button>
      </div>
    </div>
  );
};

export default DiscountSection;
```

12. Chef Detail Component

This component is responsible for displaying detailed profiles of chefs fetched from the Sanity database. It retrieves information about each chef, including their name, position, specialty, image, description, and availability status.

.

```
"use client";

import { useEffect, useState } from "react";
import Image from "next/image";
import { client } from "@/sanity/lib/client";
import { urlFor } from "@/sanity/lib/image";

// Define the Chef interface
interface Chef {
  _id: string;
  name: string;
  position: string;
  specialty: string;
  image: any;
  description: string;
  available: boolean;
}

function ChefPage() {
  const [chefs, setChefs] = useState<Chef[]>([]);

  const fetchChefs = async () => {
    try {
      const query = `*[_type == "chef"]{_id, name, position, specialty, image, description,
available}`;const fetchedChefs = await client.fetch(query);
      setChefs(fetchedChefs);
    } catch (error) {
      console.error("Failed to fetch chefs:", error);
      setChefs([]);
    }
  };

  useEffect(() => {
    fetchChefs();
  }, []);

  return (
    <div className="px-4 py-8 bg-gray-100 min-h-screen">
      <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-4">
        {chefs.map((chef, index) => (
          <div
            key={index}
            className=" rounded-lg  p-4 hover:shadow-xl transition-shadow duration-300"
          >
            <div className="mb-4">
              <Image
                src={urlFor(chef.image).url()}
                alt={chef.name}
                width={300}
                height={200}
                 className="md:w-[280px] md:h-[300px] object-contain rounded-lg mb-4"
                />
            </div>
            <h2 className="text-xl sm:text-2xl md:text-3xl font-semibold text-gray-800">{chef.name}</h2>
            <p className="text-gray-500 text-sm sm:text-base">{chef.position}</p>
            <p className="text-gray-400 text-xs sm:text-sm">{chef.specialty}</p>
            <p className="text-gray-600 text-xs sm:text-sm mt-2">{chef.description}</p>
            {chef.available ? (
              <p className="text-green-500 text-xs sm:text-sm mt-2">Currently Active</p>
            ) : (
              <p className="text-red-500 text-xs sm:text-sm mt-2">Currently Inactive</p>
            )}
          </div>
        ))}
      </div>
    </div>
  );
}

export default ChefPage;
```

Self-Validation Checklist:

Task/Component	Status	Comments
Frontend Component Development	✓	All components are dynamically loaded and functional.
Styling and Responsiveness	✓	Design is responsive on all screen sizes.
Code Ùuality	✓	Code is clean and well-documented..
Documentation and Submission	✓	Technical report and code are submitted.