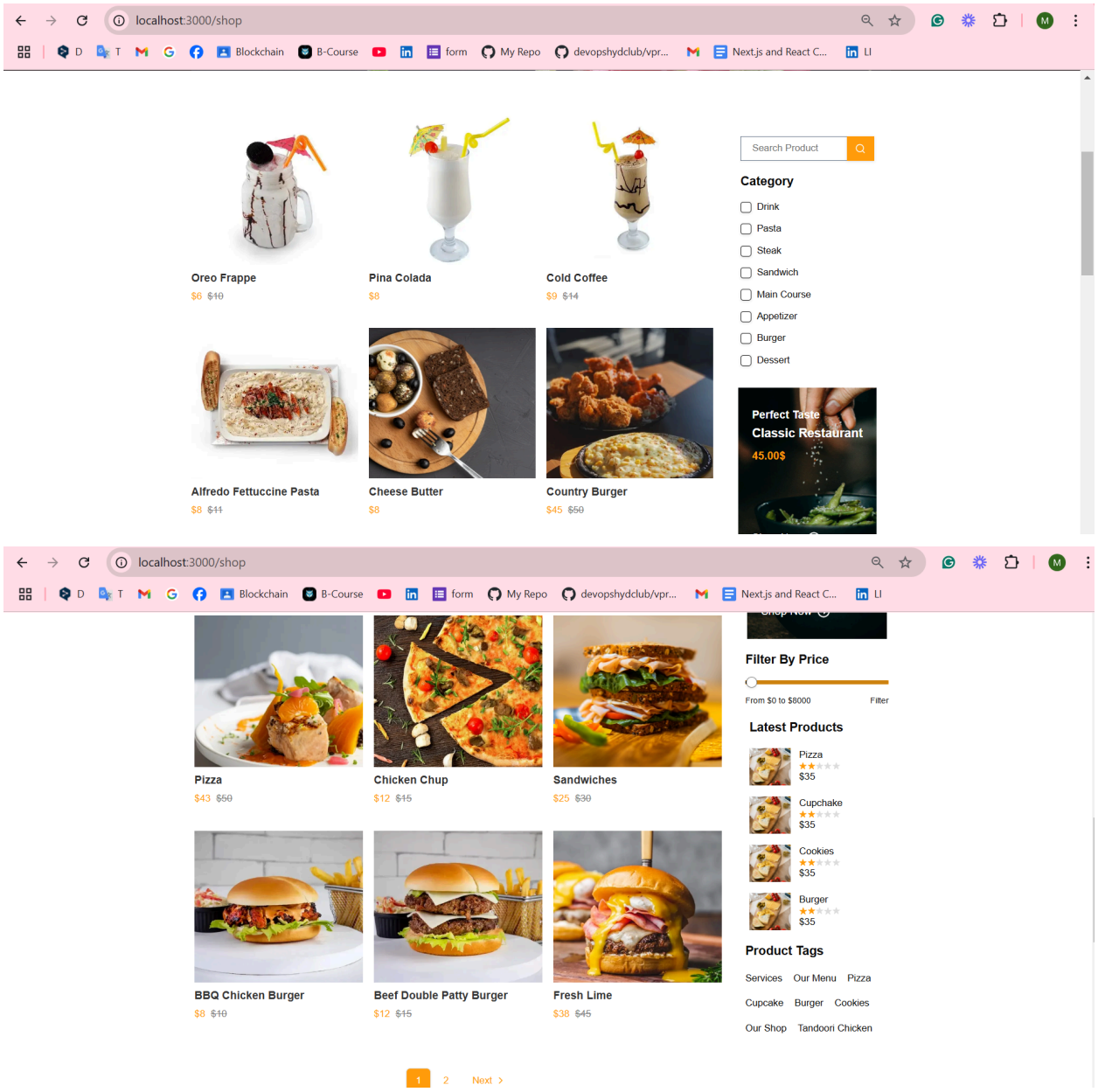
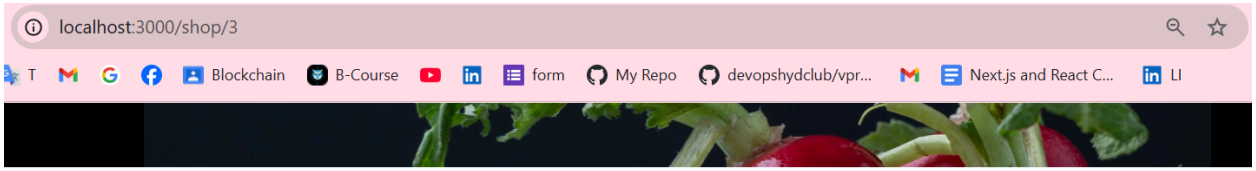
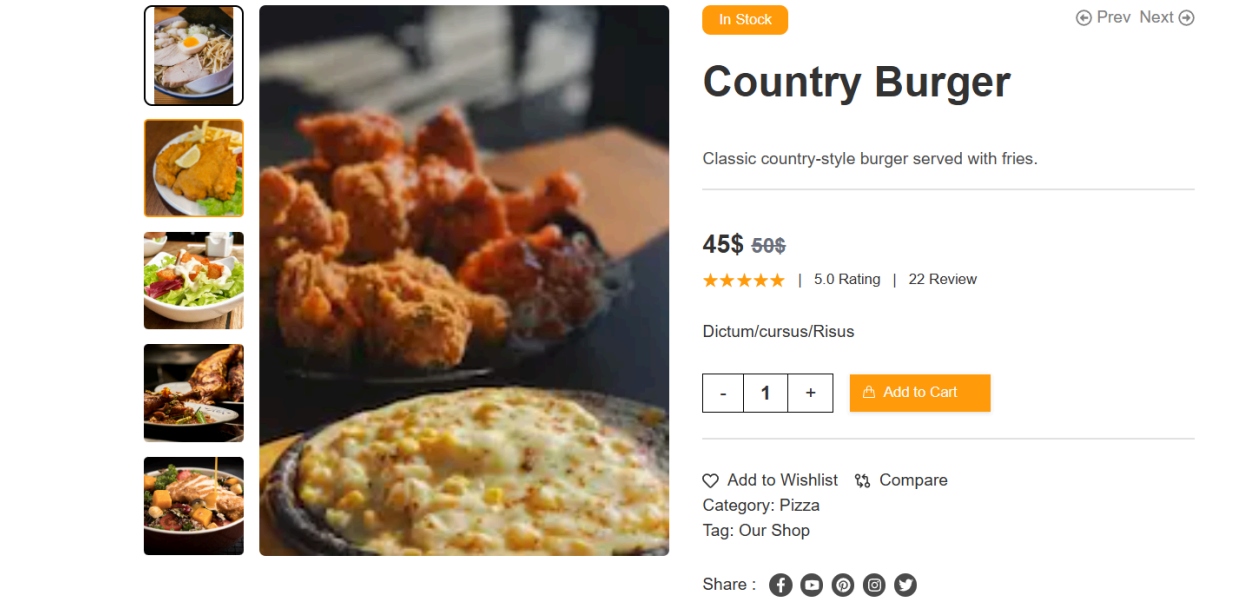
Functional Deliverables

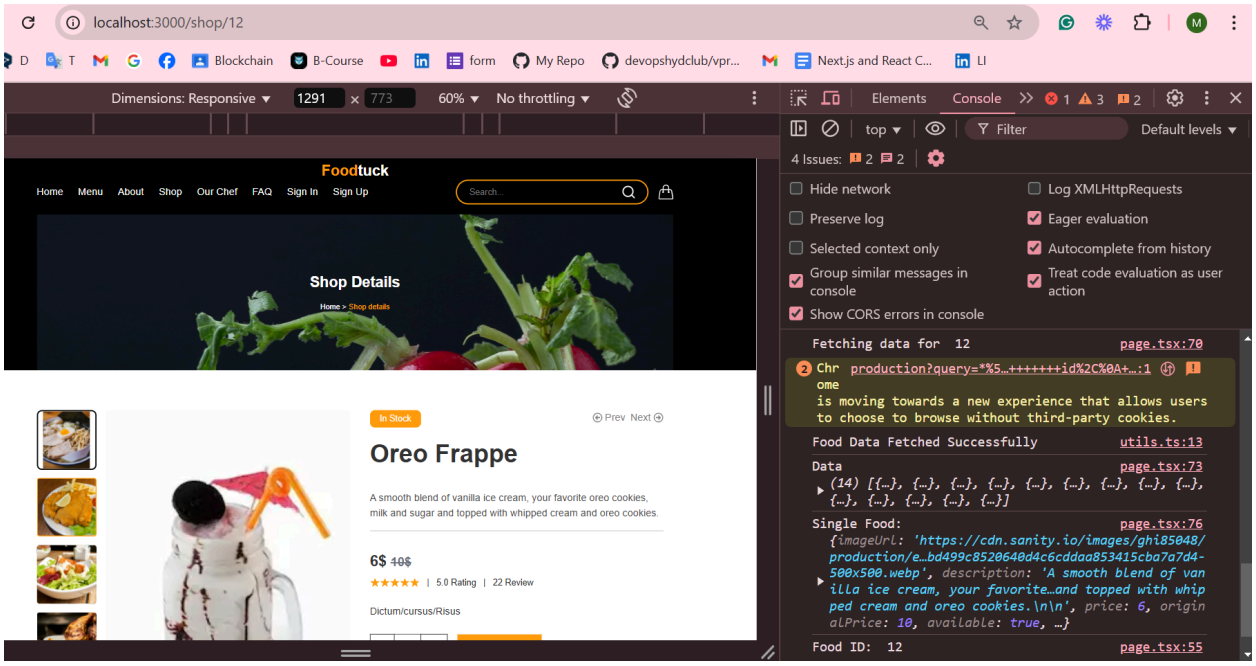
Product Listing Page with Dynamic Data



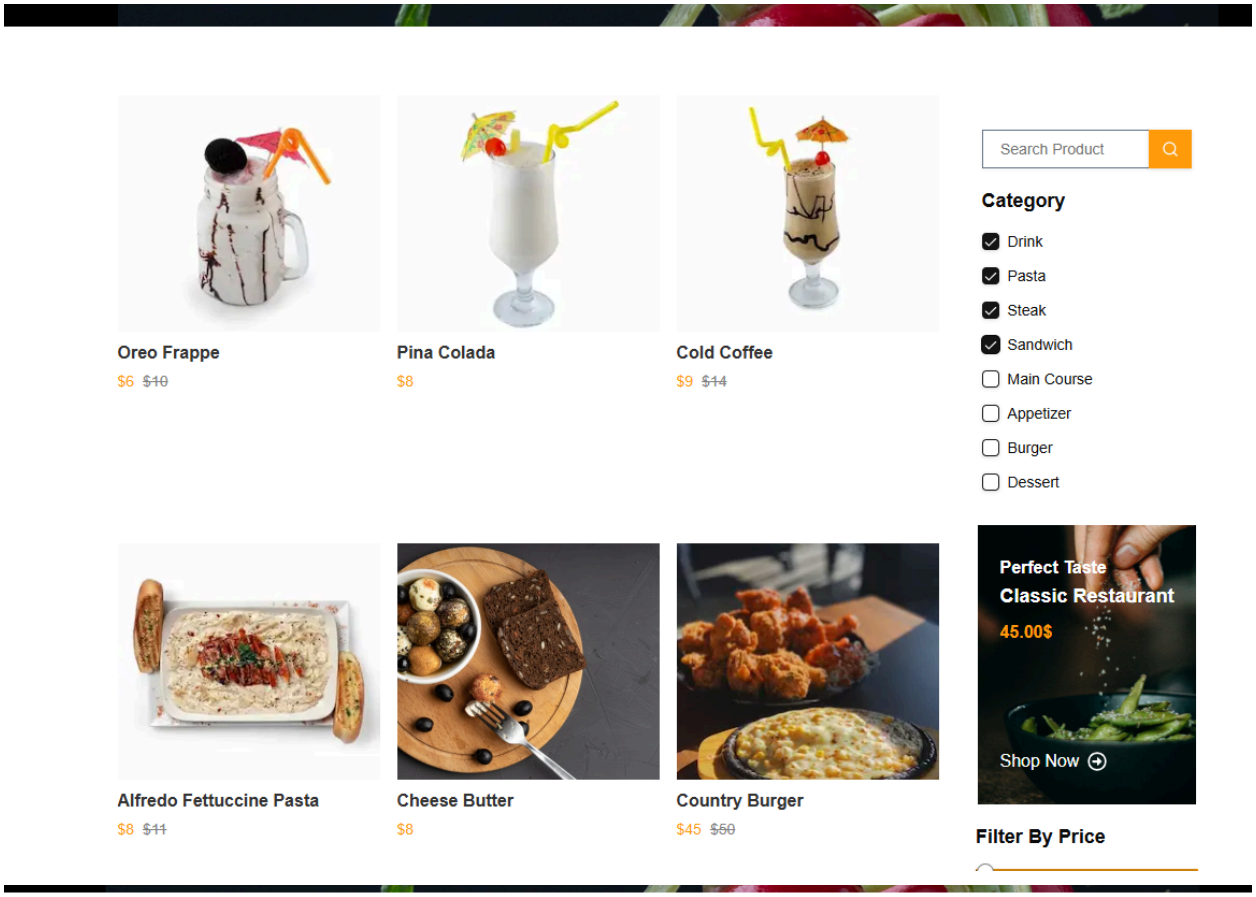
Individual Product Detail Pages with Accurate Routing and Data Rendering

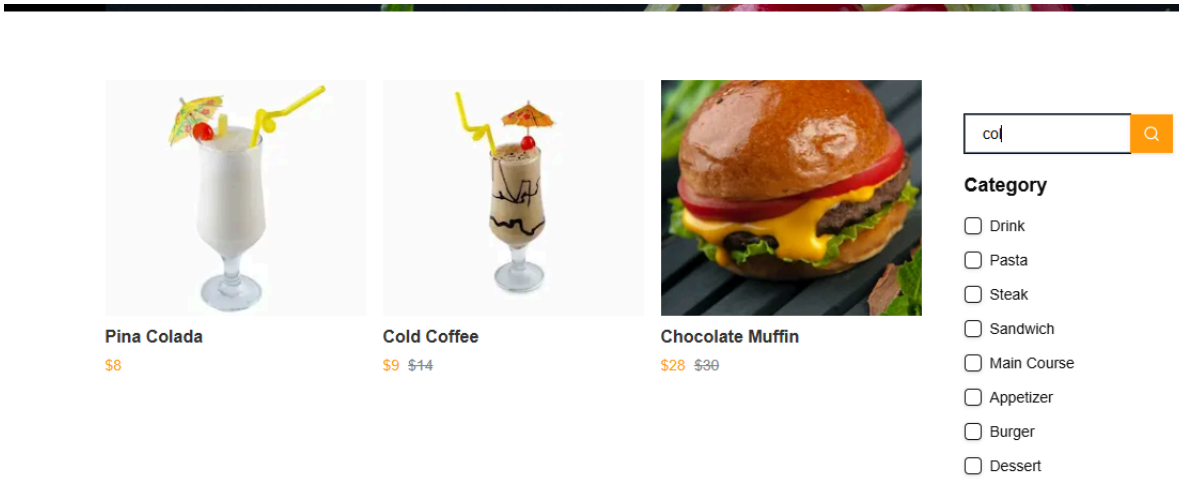






Category Filter, Search Bar, Pagination





Pizza

$43 ~~$50~~

Chicken Chup

$12 ~~$15~~

Sandwiches

$25 ~~$30~~

BBQ Chicken Burger

$8 ~~$10~~

Beef Double Patty Burger

$12 ~~$15~~

Fresh Lime

$38 ~~$45~~

Latest Products

Pizza

★★☆☆☆

$35

Cupchake

★★☆☆☆

$35

Cookies

★★☆☆☆

$35

Burger

★★☆☆☆

$35

Product Tags

Services Our Menu Pizza

Cupcake Burger Cookies

Our Shop Tandoori Chicken

1 2 Next ›

★★☆☆☆

$35

Product Tags

Services Our Menu Pizza

Cupcake Burger Cookies

Our Shop Tandoori Chicken

‹ Previous 1 2

Similar Products

Oreo Frappe

$6 ~~$10~~

Pina Colada

$8

Cold Coffee

$9 ~~$14~~

Alfredo Fettuccine Pasta

$8 ~~$11~~

Code Deliverables

Shop Card

src > components > ShopCard.tsx > ShopCard

```tsx
import Image from "next/image";

interface ShopCardType {
    id?: number;
    ImagePath: string;
    AltText: string;
    ImageWidth?: number;
    ImageHeight?: number;
    CurrentPrice: number;
    OldPrice?: number;
    DishName: string;
}

export default function ShopCard({ ImagePath, ImageHeight, ImageWidth, AltText, CurrentPrice, DishName, OldPrice}: ShopCardType) {
    return(
        <div className="w-full overflow-hidden text-white bg-transparent hover:scale-105 transition duration-300" >
            {/* <div className="relative " > */}
                <Image
                    className="object-cover h-[220px]"
                    src={ImagePath}
                    alt={AltText}
                    width={ImageWidth}
                    height={ImageHeight}
                />
            {/* </div> */}
            <div className="my-2">
                <p className=" font-bold text-wrap text-[#333333] ">{DishName}</p>
                <p className="text-[#FF9F0D] text-sm mt-1">${CurrentPrice}
                {OldPrice && (<span className="line-through text-[#828282] ml-2">${OldPrice}</span>)}
                </p>
            </div>
        </div>
    )
}
```

Search Bar

src > components > SearchBar.tsx > ...

```tsx
import { ChangeEvent } from "react";
import { Button } from "./ui/button";
import { Input } from "./ui/input";
import { FiSearch } from "react-icons/fi";


interface SearchCardType {
    PlaceholderText: string;
    InputBgColor?: string;
    value: string;
    onChange: (e: ChangeEvent<HTMLInputElement>) => void;
}

export default function SearchBar({PlaceholderText, InputBgColor = "transparent", value, onChange}: SearchCardType){
    return(
        <div className="flex mt-8">
            <Input
            className="py-4 pl-4 border-slate-500 rounded-none border-r-0"
                type="text"
                placeholder={PlaceholderText}
                style={{backgroundColor: InputBgColor}}
                value={value}
                onChange={onChange}
            />
            <Button type="button" className="bg-[#FF9F0D] rounded-none px-3 py-[16.7px]">
                <FiSearch />
            </Button>
        </div>
    )
}
```

API Integration Methods in utils.ts

src > lib > TS utils.ts > ...

```ts
import { client } from "@/sanity/lib/client";
import { clsx, type ClassValue } from "clsx"
import { twMerge } from "tailwind-merge"

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs))
}

export async function fetchShopData(query: string) {
  try {
      // const query = `*[_type == "food"]{id, "imageUrl":image.asset->url, name, price, originalPrice}`;
      const food = await client.fetch(query);
      console.log("Food Data Fetched Successfully");
      if (!food.length) throw new Error('No Food Items Found!');
      return food;
  } catch (error) {
      console.error('Error fetching data:', error);
      throw error;
  }
}

export async function fetchChefData(query: string) {
  try {
      // const query = `*[_type == "chef"]{"imageUrl": image.asset->url, name,position,}`;
      const chef = await client.fetch(query);
      console.log("Chef Data Fetched Successfully");
      if (!chef.length) throw new Error('No Chef Found!');
      return chef;
  } catch (error) {
      console.error('Error fetching data:', error);
      throw error;
  }
}

```

Dynamic Routing

src > app > shop > [food_id] > page.tsx > ...

```tsx
} from "@/components/ui/carousel"
import { useParams } from "next/navigation";
import { fetchShopData } from "@/lib/utils";

interface Food {
    id: number;
    name: string;
    imageUrl: string;
    description: string;
    price: number;
    originalPrice?: number;
    available: boolean;
  }

export default function ShopDetails() {
    const [foodData, setFoodData] = useState<Food | null>(null);
    const [allFood, setAllFood] = useState<Food[]>([]);
    const [error, setError] = useState<string>('');
    const [loading, setLoading] = useState<boolean>(true);
    const [activeImage, setActiveImage] = useState<string>("/assets/images/shop/shop_detail/Food_1.svg");
    const [currentIndex, setCurrentIndex] = useState<number>(0);
    const itemsPerPage: number = 4;

    const thumbnails = [
        "/assets/images/shop/shop_detail/Food_1.svg",
        "/assets/images/shop/shop_detail/Food_2.svg",
        "/assets/images/shop/shop_detail/Food_3.svg",
        "/assets/images/shop/shop_detail/Food_4.svg",
        "/assets/images/shop/shop_detail/Food_5.svg",
    ];

    const {food_id} = useParams();

    console.log("Food ID: ", food_id);

```

Technical Report

Steps Taken to Build and Integrate Components

Component Development

Requirement Analysis: Defined the purpose and functionality of each component.

Component Design: Structured the components using modular and reusable patterns.

Implementation: Developed components using Nextjs and TailwindCSS.

Styling and Responsiveness: Applied TailwindCSS to ensure a responsive UI.

Testing: Conducted unit testing and integration testing to validate component behavior.

Integration Process

State Management: Utilized appropriate state management techniques for seamless data flow.

API Integration: Fetched and displayed data from APIs using best practices for error handling.

Dynamic Routing: Implemented dynamic routing using useParams() to retrieve route parameters.

Event Handling: Ensured event-driven interactions were handled efficiently.

Challenges Faced and Solutions Implemented

API Integration Errors
Issue: The fetched data id type did not match the id type from parameters, causing the data to be undefined in the console. The id from parameters was a string, while the expected id in the API query was a number.
Solution: Converted the parameter id from a string to a number using Number(food_id) before making the API request to ensure type consistency.

Handling Dynamic Routes
Issue: Difficulty accessing and managing dynamic route parameters.

Solution: Used useParams() from React Router to extract route parameters dynamically.

Best Practices Followed During Development

Code Quality and Maintainability

- Followed DRY (Don't Repeat Yourself) principle.
- Used meaningful variable and function names.
- Documented code with comments for clarity.

Testing and Debugging

- Wrote unit tests for key components.
- Used debugging tools to identify and fix issues efficiently.

Version Control and Collaboration

- Used Git for version control with clear commit messages.
