

Day 4

Building Dynamic Frontend Components for FoodTuck Q-Commerce Website

Prepared by : [Farwa kanwal](#)

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

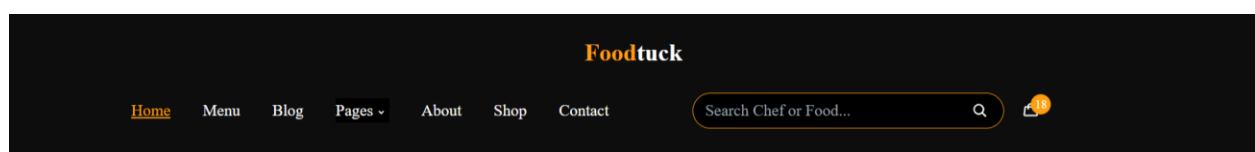
designs using utility-first classes..

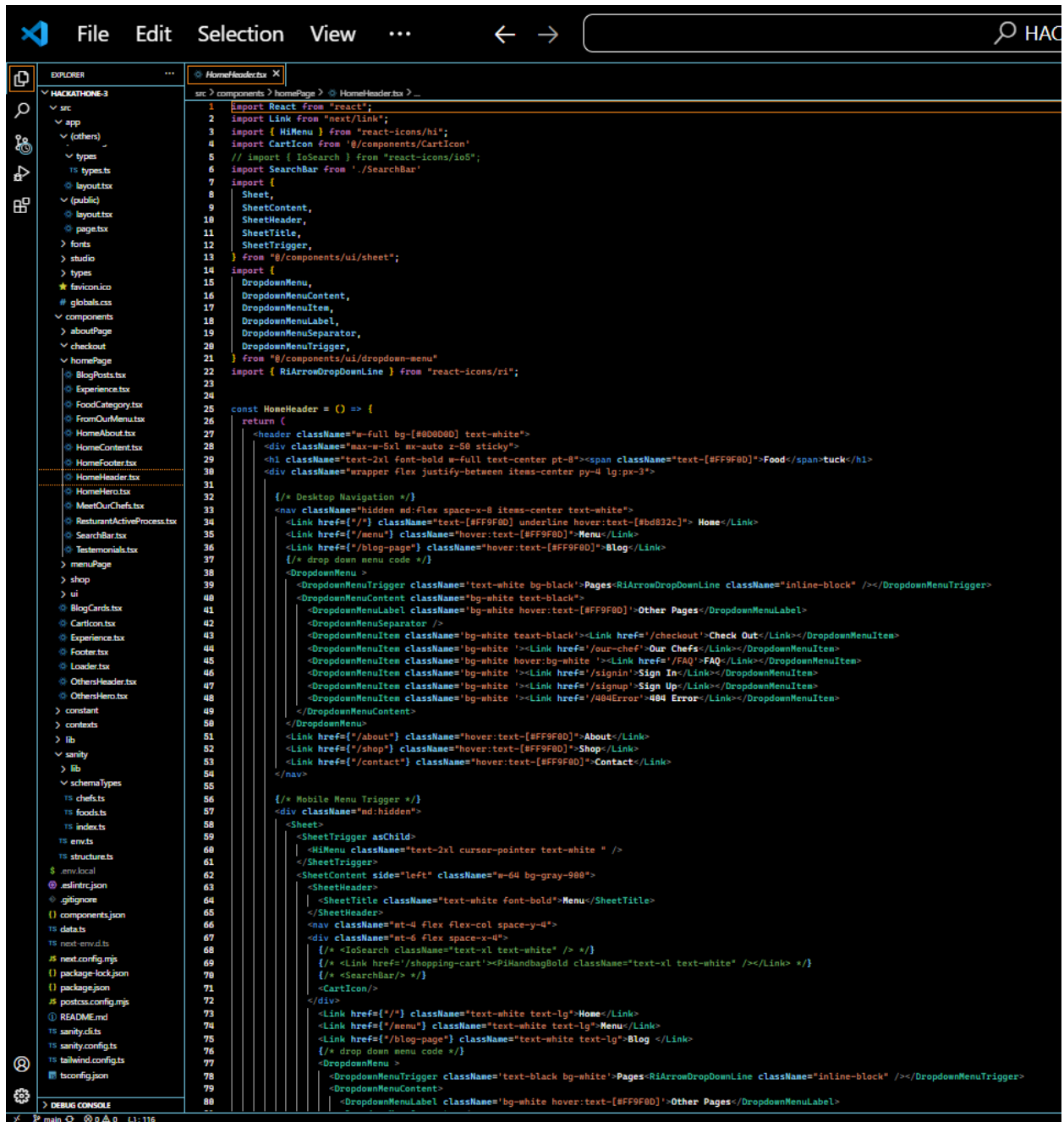
- Integrated ShadCN components for reusable and customizable frontend elements, improving design consistency.
- Implemented state management to handle dynamic content and user interactions efficiently, enhancing the overall user experience.
- Developed problem-solving skills while managing dynamic states and optimizing performance across the application.

Components Built:

1. Header Component:

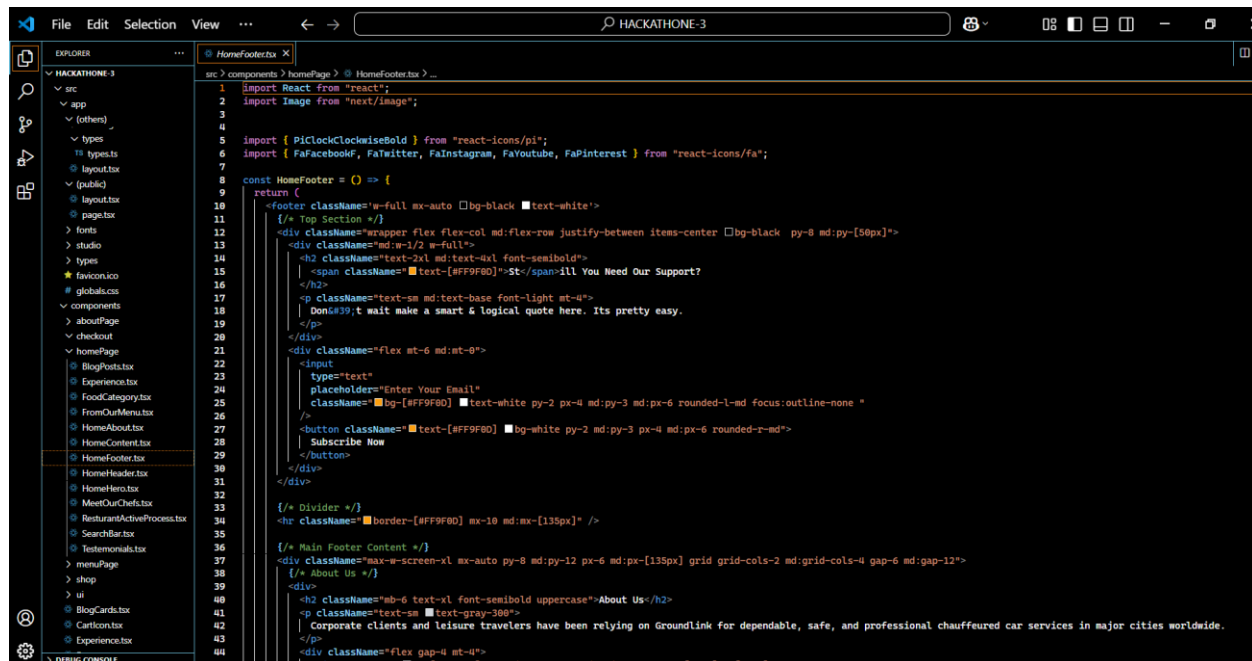
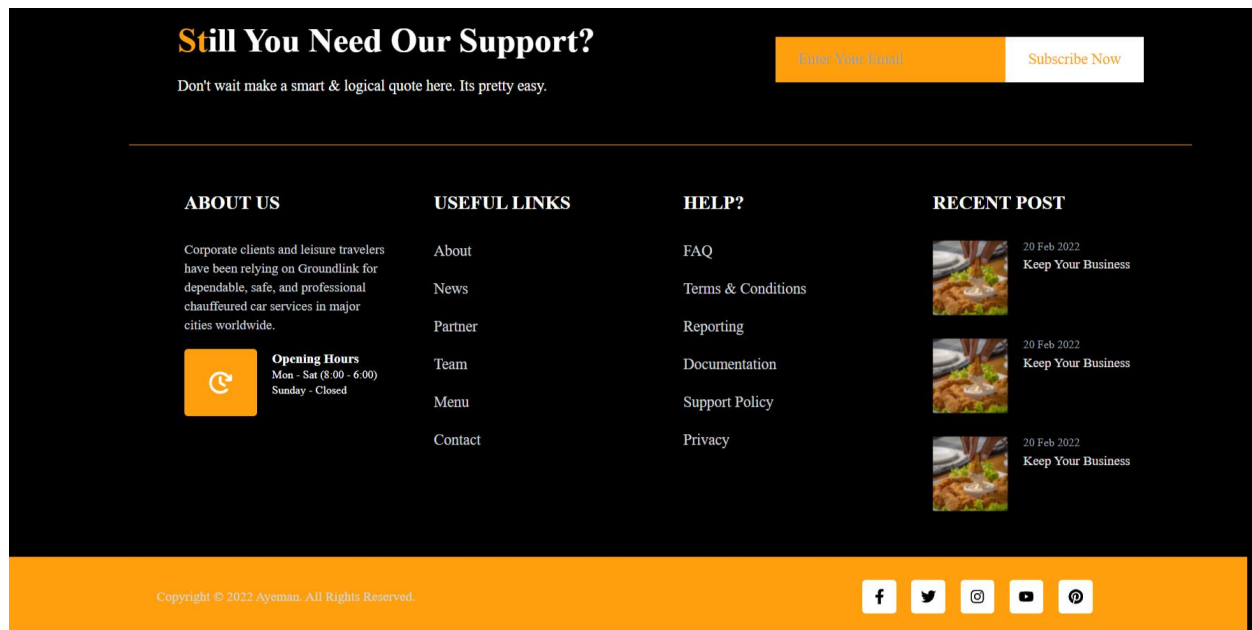
The Header component contains the navigation menu, logo, and other important elements like the search bar or user profile. It's the first section of the website that users interact with.





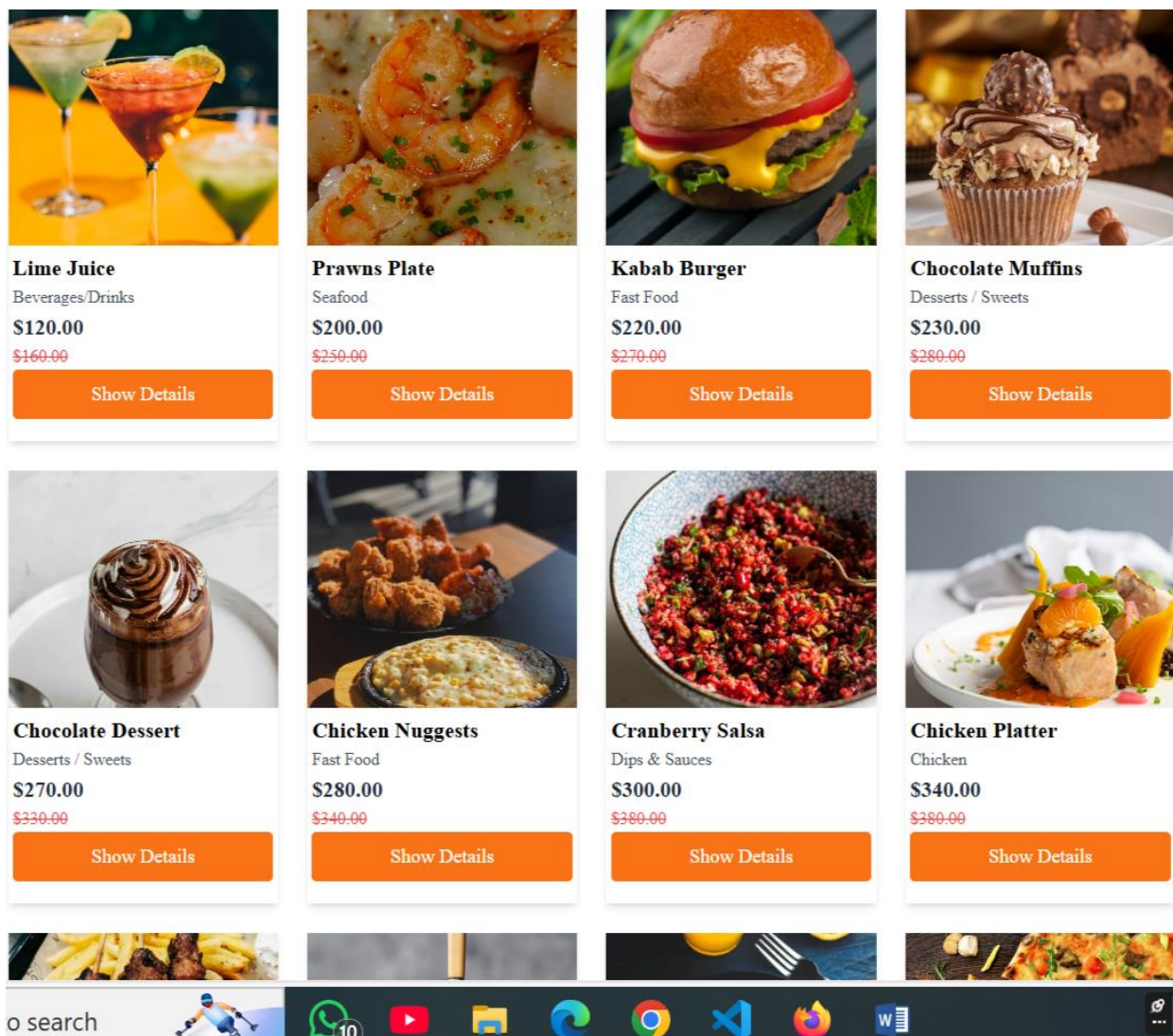
2. Footer Component

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.







```

191 > app > [others] > shop > [c] > @ page:ba > ...
192 "use client"
193
194 import { useState, useEffect, useRef } from "react"
195 import Image from "next/image"
196 import { Button } from "@components/ui/button"
197 import { Input } from "@components/ui/input"
198 import { Star, Facebook, Twitter, Instagram, PinIcon as Pinterest } from "lucide-react"
199 import Hero from "@components/OtherHero"
200 import { Loader } from "@components/Loader"
201 import { useCart } from "@contexts/CartContext"
202 import { client } from "@sanity/lib/client"
203 import { toast } from "react-hot-toast"
204 import Link from "next/link"
205
206 interface ProductData {
207   _id: string
208   name: string
209   imageUrl: string
210   description: string
211   price: number
212 }
213
214 interface FeaturedImage {
215   _id: string
216   imageUrl: string
217 }
218
219 export default function ProductPage({ params }: { params: { id: string } }) {
220   const [quantity, setQuantity] = useState(1)
221   const [selectedImage, setSelectedImage] = useState(0)
222   const [product, setProduct] = useState<ProductData | null>(null)
223   const [featuredImages, setFeaturedImages] = useState<FeaturedImage[]>([])
224   const [translateX, setTranslateX] = useState(0)
225   const [isHovering, setIsHovering] = useState(false)
226   const { addToCart } = useCart()
227   const sliderRef = useRef<HTMLDivElement>(null)
228
229   useEffect(() => {
230     const fetchProduct = async () => {
231       try {
232         const query = `*[_type == "food" && _id == ${id}][0] {
233           _id,
234           name,
235           description,
236           "imageUrl": image.asset->url,
237           price
238         }`
239         const data = await client.fetch(query, { id: params.id })
240         setProduct(data)
241       } catch (error) {
242         console.error("Error fetching product:", error)
243       }
244     }
245
246     const fetchFeaturedImages = async () => {
247       try {
248         const query = `*[_type == "food"] {
249           _id,
250           "imageUrl": image.asset->url
251         }`
252         const data = await client.fetch(query)
253         setFeaturedImages(data)
254       } catch (error) {
255         console.error("Error fetching featured images:", error)
256       }
257     }
258
259     fetchProduct()
260     fetchFeaturedImages()
261   }, [params.id])
262
263   useEffect(() => {
264     const slider = sliderRef.current
265     if (!slider || featuredImages.length === 0) return
266
267     const slideImages = () => {
268       if (!isHovering) {
269         setTranslateX((prevTranslateX) => prevTranslateX - 0.5)
270       }
271     }
272   })

```


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.



Chocolate Muffins





★★★★★ (5 reviews)

\$230.00

Experience the taste in every bite! Our fresh and delicious food items make your special moments even more memorable. Prepared with the finest ingredients and authentic flavors, each dish brings a new delight to your table!

- 1 -






Add to Cart

Share:    

Description

*Indulge in the rich and delightful taste of our Moist & Sweet Chocolate Bunnies. These adorable and fluffy treats are made with the finest cocoa, ensuring a deep chocolate flavor in every bite. Perfectly moist and irresistibly sweet, these bunnies are a hit with both kids and adults. Whether you're enjoying them as a snack, dessert, or a special treat, our Chocolate Bunnies are sure to bring a smile to your face. Pair them with a glass of milk or your favorite hot beverage for the ultimate comfort food experience. A must-try for all chocolate lovers!

Featured Products



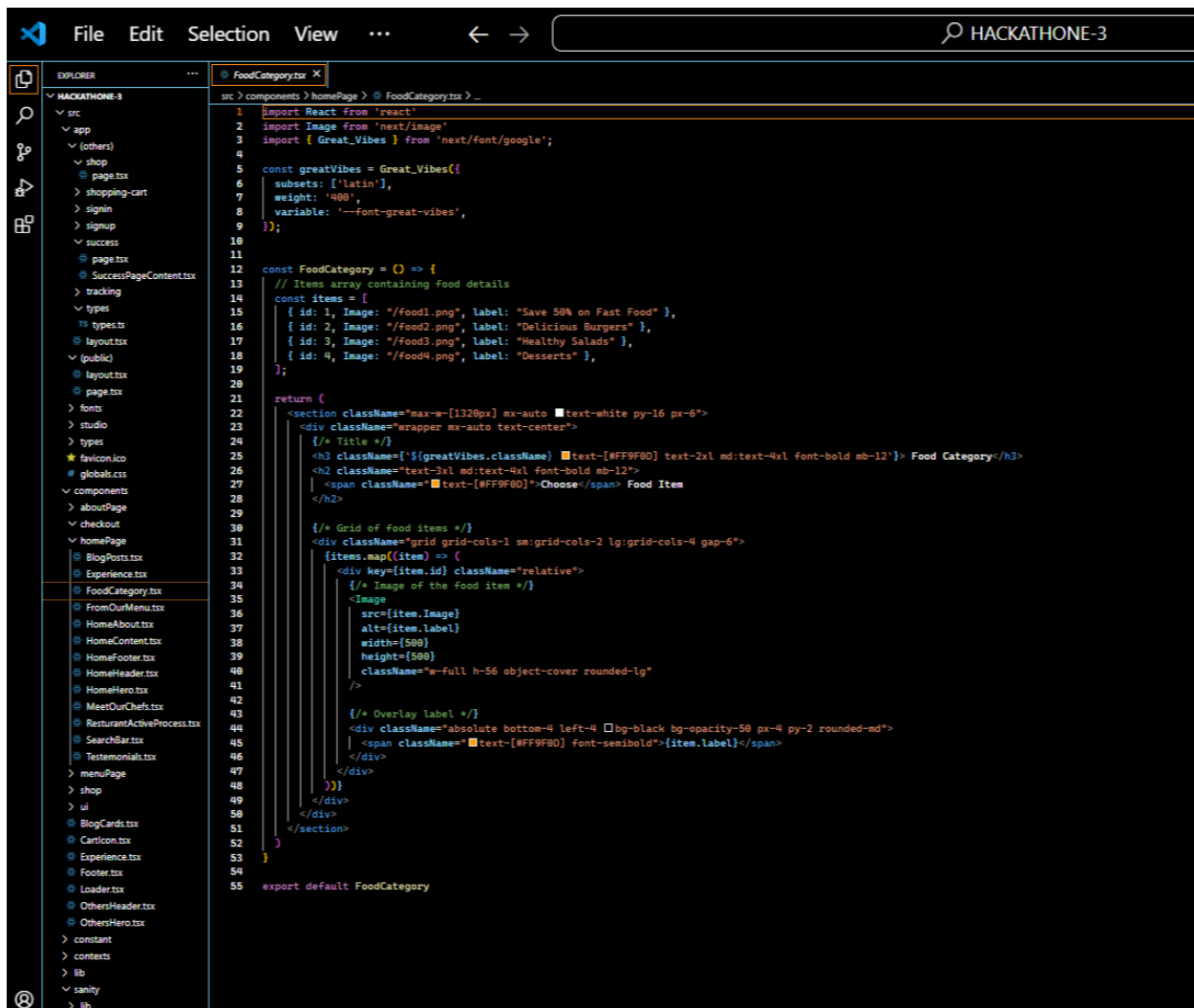


5. Category Component

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.

Category

- ☐ Chicken
- ☒ Fast Food
- ☐ Pizza
- ☐ Vegetables
- ☒ Desserts / Sweets
- ☐ Seafood
- ☐ Dips & Sauces
- ☐ Beverages/Drinks
- ☐ Pizza



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.



```
File Edit Selection View ... HACKATHONE-3
EXPLORER
HACKATHONE-3
  src
  app
  (others)
  shop
    searchBar.tsx
  shopping-cart
  signin
  signup
  success
  page.tsx
  SuccessPageContent.tsx
  tracking
  types
  types.ts
  layout.tsx
  (public)
  layout.tsx
  page.tsx
  fonts
  studio
  types
  favicon.ico
  # globals.css
  components
    aboutPage
    checkout
    searchBar.tsx

src > components > shop > searchBar.tsx > ...
1 "use client"
2
3 import { CiSearch } from "react-icons/ci"
4
5 interface SearchBarProps {
6   searchQuery: string
7   setSearchQuery: (query: string) => void
8 }
9
10 export function SearchBar({ searchQuery, setSearchQuery }: SearchBarProps) {
11   return (
12     <div className="flex w-[248px] h-[46px] mt-[72px] text-center">
13       <input
14         type="text"
15         placeholder="Search Product"
16         value={searchQuery}
17         onChange={(e) => setSearchQuery(e.target.value)}
18         className="w-full bg-[#ebe2d5] pl-4 text-gray-400"
19       />
20       <CiSearch size={20} className="bg-[#FF9F0D] p-3 text-[#FFFFFF] w-[46px] h-[46px] -translate-x-10" />
21     </div>
22   )
23 }
24
25
```

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 and manage their cart's contents. The total price updates dynamically based on the quantity of each item

-	1	+
---	---	---

Add to Cart



8. Filter Component

The Filter Component allows users to filter food items based on their price range. Users can select a price range (from \$0 to \$500), 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:

Price Range Filter: Users can choose a range (from \$0 to \$500) to filter food items.

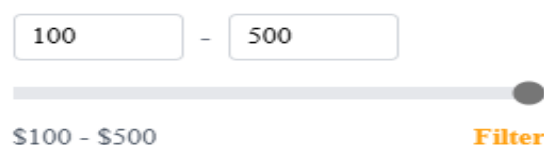
1.

Dynamic Filtering: The list of food cards updates automatically to display only items within the selected price range.

2.

Food Cards: Each food card includes a name, price, and image, making it easy for users to browse the items.

Filter By Price



100 - 500

\$100 - \$500 **Filter**

FileEditSelectionView...<-->HACK

EXPLORER

HACKATHONE-3

src

app

fonts

studio

types

favicon.ico

globals.css

components

aboutPage

checkout

homePage

BlogPosts.tsx

Experience.tsx

FoodCategory.tsx

FromOurMenu.tsx

HomeAbout.tsx

HomeContent.tsx

HomeFooter.tsx

HomeHeader.tsx

HomeHero.tsx

MeetOurChefs.tsx

RestaurantActiveProcess.tsx

SearchBar.tsx

Testimonials.tsx

menuPage

shop

CategoryFilters.tsx

LatestProducts.tsx

PriceFilter.tsx

ProductGrid.tsx

ProductTags.tsx

SearchBar.tsx

ShopContent.tsx

Slider.tsx

SortAndShowControls.tsx

ui

BlogCards.tsx

CartIcon.tsx

Experience.tsx

Footer.tsx

Loader.tsx

OthersHeader.tsx

OthersHero.tsx

constant

contexts

lib

sanity

lib

schemaTypes

chefs.ts

foods.ts

index.ts

env.ts

structure.ts

env.local

eslint.config

PriceFilter.tsx

src > components > shop > PriceFilter.tsx > ...

112

113 "use client"

114

115 import { useState, useEffect } from "react"

116

117 interface PriceFilterProps {

118 | minPrice: number

119 | maxPrice: number

120 | setMinPrice: (price: number) => void

121 | setMaxPrice: (price: number) => void

122 }

123

124 export function PriceFilter({ minPrice, maxPrice, setMinPrice, setMaxPrice }: PriceFilterProps) {

125 | const [localMin, setLocalMin] = useState(minPrice)

126 | const [localMax, setLocalMax] = useState(maxPrice)

127

128 | useEffect(() => {

129 | | const timer = setTimeout(() => {

130 | | | setMinPrice(localMin)

131 | | | setMaxPrice(localMax)

132 | | }, 500)

133 |

134 | return () => clearTimeout(timer)

135 | }, [localMin, localMax, setMinPrice, setMaxPrice])

136

137 | return (

138 | | <div className="mt-6 mb-4">

139 | | | <h2 className="font-helvetica text-[20px] font-bold mb-3">Filter By Price</h2>

140 | | | <div className="flex items-center space-x-2 mb-2">

141 | | | | <input

142 | | | | | type="number"

143 | | | | | value={localMin}

144 | | | | | onChange={(e) => setLocalMin(Number(e.target.value))}

145 | | | | | className="w-20 px-2 py-1 border border-gray-300 rounded text-sm"

146 | | | | | min={100}

147 | | | | | max={localMax}

148 | | | | | />

149 | | | | -

150 | | | | <input

151 | | | | | type="number"

152 | | | | | value={localMax}

153 | | | | | onChange={(e) => setLocalMax(Number(e.target.value))}

154 | | | | | className="w-20 px-2 py-1 border border-gray-300 rounded text-sm"

155 | | | | | min={localMin}

156 | | | | | max={500}

157 | | | | | />

158 | | | | </div>

159 | | | <input

160 | | | | type="range"

161 | | | | min={100}

162 | | | | max={500}

163 | | | | value={localMax}

164 | | | | onChange={(e) => setLocalMax(Number(e.target.value))}

165 | | | | className="w-full h-2 bg-gray-200 rounded-lg appearance-none cursor-pointer"

166 | | | | />

167 | | | <div className="flex justify-between items-center mt-2">

168 | | | | <div className="text-sm text-gray-600">

169 | | | | | \${localMin} - \${localMax}

170 | | | | </div>

171 | | | | <button className="text-sm text-[#FF9F00] hover:underline font-semibold">Filter</button>

172 | | | </div>

173 | | </div>

174 |)

175 | }

176

9. 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:

Voucher Code Input: A text input where users can enter the voucher code.1.

Discount Validation: The component checks if the entered voucher code is valid and

applies the corresponding discount.

2.

Dynamic Price Update: The total price of the order is updated in real-time once the

voucher is applied.

3.

Error Handling: If the voucher code is invalid, an error message is shown to the user.

Enter your Coupon Code for discount!
Available coupon code is:"DISCOUNT10"

Enter coupon code

Apply

```

138 </div>
139 > blog-page
140 </div>
141
142
143 /* Cart Summary Section */
144 <div className="mt-8 space-y-6 lg:space-y-0 lg:flex lg:space-x-8">
145   /* Coupon Section */
146   <div className="lg:flex-1">
147     <div className="bg-white p-6 rounded-lg shadow-sm">
148       <h2 className="text-lg font-semibold mb-4">Have a Coupon?</h2>
149       <p>Enter your Coupon Code for discount!</p>
150       <p>Available coupon code is:"DISCOUNT10"</p>
151       <div className="flex space-x-2 mt-2">
152         <input
153           type="text"
154           value={couponCode}
155           onChange={(e) => setCouponCode(e.target.value)}
156           placeholder="Enter coupon code"
157           className="flex-1"
158         />
159         <button
160           onClick={handleApplyCoupon}
161           className="bg-orange-400 hover:bg-orange-500 active:bg-orange-900">
162           Apply
163         </button>
164       </div>
165     </div>
166   </div>
167 </div>

```

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

[← Back to All Chefs](#)



Chef Details



Specialty
Italian Cuisine



Experience
12 years



Rating
Master Chef



Availability
Available for Events

About Tahmina Rumi

Expert in crafting authentic Italian dishes and pastries.

[Book Tahmina Rumi for Your Event](#)



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 Quality	✓	Code is clean and well-documented..
Documentation and Submission	✓	Technical report and code are submitted.