Hackathon 5 - Day 4

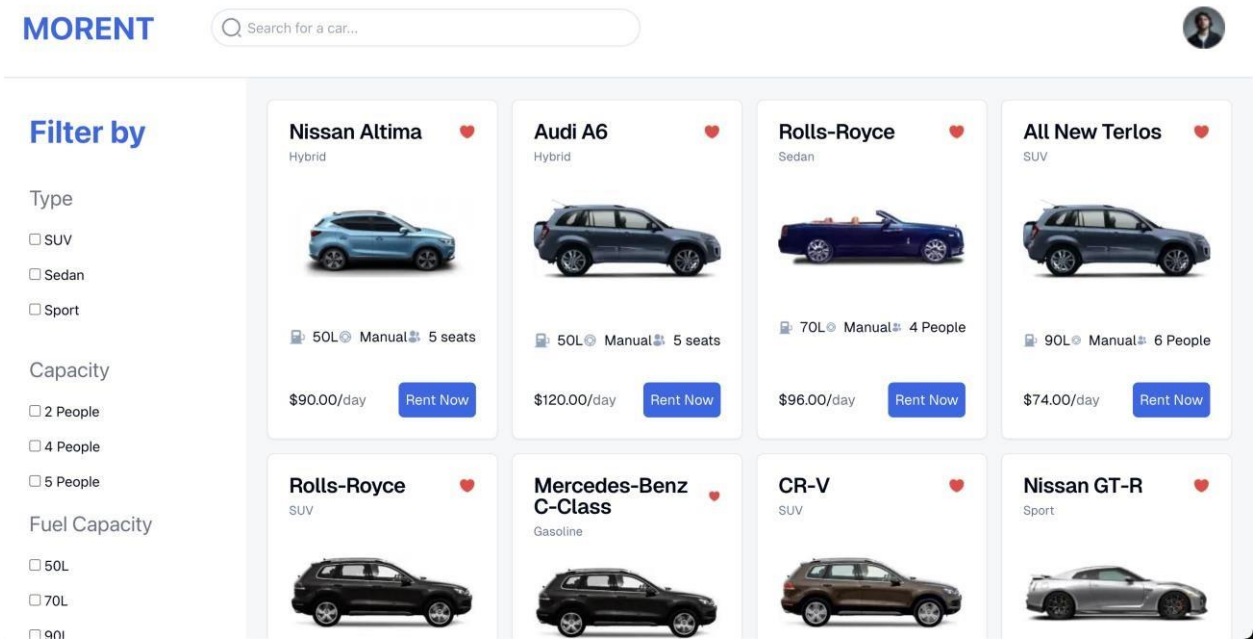
Dynamic Frontend Components Report

Project Overview

This project focuses on building a dynamic frontend for a car rental website using Next.js and Sanity CMS. The main goal is to implement functional, reusable components to enable users to filter cars based on attributes such as type, seating capacity, and fuel capacity.

Step 1: Planning the Filters

1. **Identified Attributes:**
    a. Car type (SUV, Sedan, Sport)
    b. Seating capacity (2 People, 4 People, 5 People)
    c. Fuel capacity (50L, 70L, 90L) **2. Defined Objectives:**
    a. Enable users to filter cars dynamically.
    b. Ensure the filters are reusable and modular.



Step 2: Setting Up the Filter Component

1. **Created a Reusable Component:**

    a. Developed a FilterComponent using React functional components with TypeScript.
    b. Integrated a useState hook to manage filter state locally.
2. **Implemented Input Elements:**
    a. Added checkbox inputs for each attribute.
    b. Handled filter updates using an onChange event listener.
3. **Code Implementation:**

ui-ux-hackathon

FilterComponent.tsx

src > app > components > FilterComponent.tsx > FilterComponent

You, 10 minutes ago | 1 author (You)

```react-filter-component.tsx
import { useState } from 'react';

interface FilterProps {
  onFilterChange: (filters: any) => void;
}

const FilterComponent: React.FC<FilterProps> = ({ onFilterChange }) => {
  const [filters, setFilters] = useState({
    type: [],
    seatingCapacity: [],
    fuelCapacity: [],
  });

  const handleFilterChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const { name, value, checked } = e.target;
    const updatedFilters: any = { ...filters };

    if (checked) {
      updatedFilters[name] = [...updatedFilters[name], value];
    } else {
      updatedFilters[name] = updatedFilters[name].filter((item: string) => item !== value);
    }

    setFilters(updatedFilters);
    onFilterChange(updatedFilters);
  };

  return (
    <div className="filter-component flex flex-col gap-10 p-10">
      <h3 className="text-4xl text-[#3563e9] font-bold">Filter by</h3>

      <div className="flex flex-col gap-5">
        <h4 className="text-2xl text-gray-500">Type</h4>
        <div className="flex flex-col gap-4">
          <label>
            <input
              type="checkbox"
              name="type"
              value="SUV"
              onChange={handleFilterChange}
            />{" "}
            SUV
```

interface HTMLInputElement

Provides special properties and methods for manipulating the options, layout, and presentation of elements.

MDN Reference

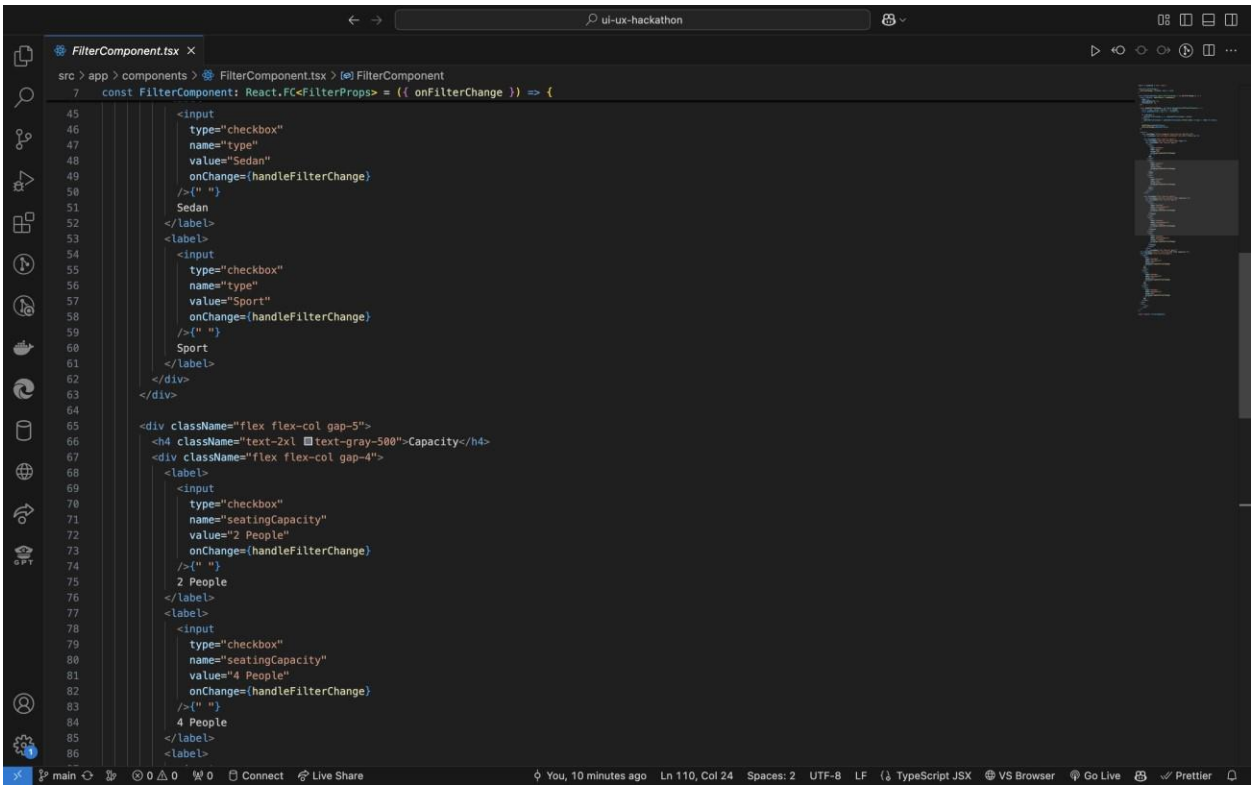
ui-ux-hackathon

FilterComponent.tsx

src > app > components > FilterComponent.tsx > FilterComponent

```react-filter-component.tsx
  const FilterComponent: React.FC<FilterProps> = ({ onFilterChange }) => {
            <input
              type="checkbox"
              name="seatingCapacity"
              value="5 People"
              onChange={handleFilterChange}
            />{" "}
            5 People
          </label>
        </div>
        <div className="flex flex-col gap-5">
    <h4 className="text-2xl text-gray-500">Fuel Capacity</h4>
    <div className="flex flex-col gap-4">
      <label>
        <input
          type="checkbox"
          name="fuelCapacity"
          value="50L"
          onChange={handleFilterChange}
        />{" "}
        50L
      </label>
      <label>
        <input
          type="checkbox"
          name="fuelCapacity"
          value="70L"
          onChange={handleFilterChange}
        />{" "}
        70L
      </label>
      <label>
        <input
          type="checkbox"
          name="fuelCapacity"
          value="60L"
          onChange={handleFilterChange}
        />{" "}
        90L
      </label>
    </div>
    </div>
        </div>
```

You, 10 minutes ago • updated



Challenges Faced

1. **Data Structure Complexity:**
    a. Initially faced difficulty managing nested filter states.
    b. Resolved by creating a unified state object for all filters.
2. **Performance Optimization:**
    a. Filtering large datasets caused noticeable lag.
    b. Improved efficiency by applying filters sequentially in the applyFilters function.

Outcomes and Learnings

1. **Functional Filters:**
    a. Successfully implemented dynamic filtering for car type, seating capacity, and fuel capacity.
2. **Improved React Skills:**
    a. Gained deeper understanding of state management and reusable component design.
3. **Sanity Integration:**
    a. Enhanced familiarity with GROQ queries and Sanity's CMS capabilities.

Code Deliverables

- Search bar

ui-ux-hackathon

Explorer (⇧⌘E)

src > app > components > Search.tsx > Search > handleSearch > filtered > data.filter() callback

You, 4 hours ago | 1 author (You)

```
'use client';

import React, { useState } from 'react';
import Image from 'next/image';
import Link from 'next/link';
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from "@/components/ui/card";
import { urlFor } from "../lib/sanity";

You, 6 hours ago | 1 author (You)
interface simplifiedCar {
  _id: string;
  name: string;
  type: string;
  slug: { current: string };
  image: string;
  fuelCapacity: string;
  transmission: string;
  seatingCapacity: string;
  pricePerDay: string;
}

You, 6 hours ago | 1 author (You)
interface SearchProps {
  data: simplifiedCar[];
}

export default function Search({ data }: SearchProps) {
  const [searchQuery, setSearchQuery] = useState('');
  const [filteredCars, setFilteredCars] = useState<simplifiedCar[]>([]);

  const handleSearch = (e: React.ChangeEvent<HTMLInputElement>) => {
    const query = e.target.value;
    setSearchQuery(query);

    if (query.trim()) {
      const filtered = data.filter((car) => {
        console.log(car.name.toLowerCase());      You, 4 hours ago • updated
        return car.name.toLowerCase().includes(query.toLowerCase());
      });
      setFilteredCars(filtered);
    } else {
      setFilteredCars([]);
      setSearchQuery('');
```

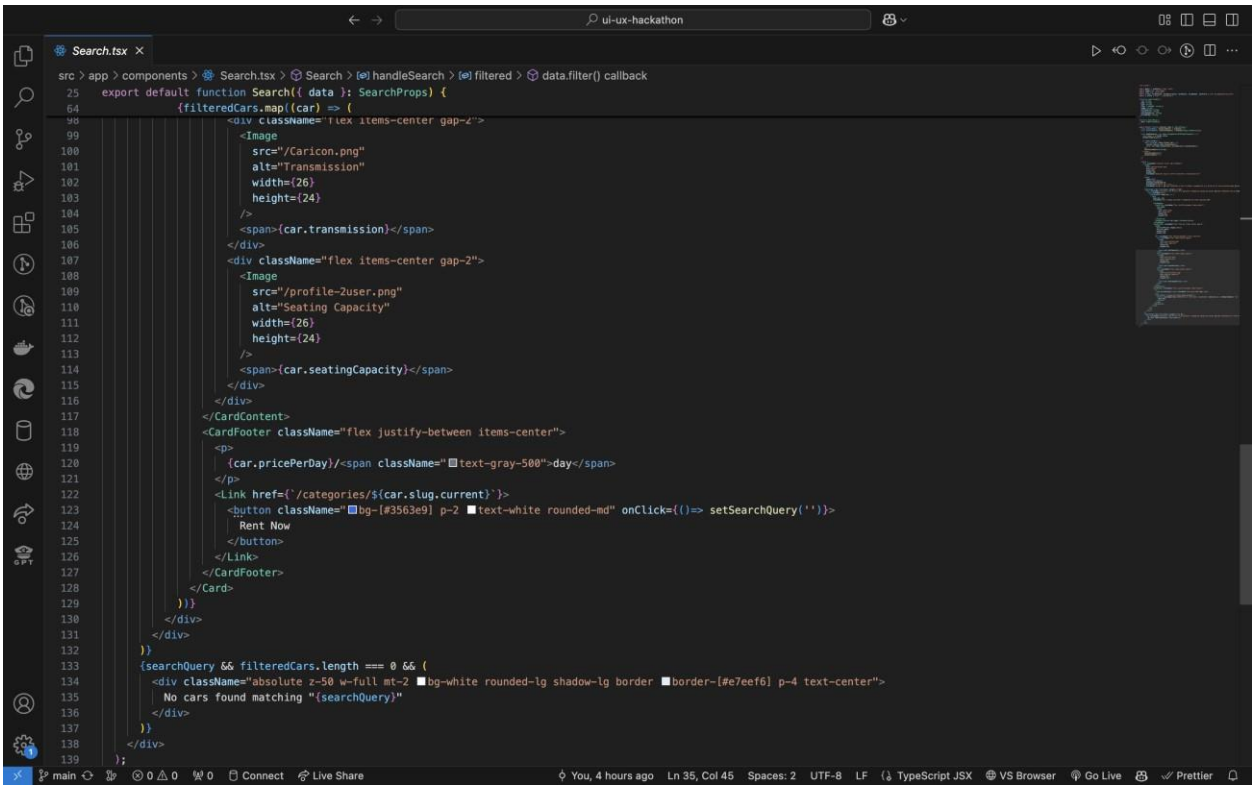
ui-ux-hackathon

Search.tsx

src > app > components > Search.tsx > Search > handleSearch > filtered > data.filter() callback

```
export default function Search({ data }: SearchProps) {
  return (
    <div className="relative w-full md:w-[492px]">
      <Image
        src="/search-normal.png"
        alt="Search"
        width={24}
        height={24}
        className="absolute top-1/2 left-3 transform -translate-y-1/2"
      />
      <input
        type="text"
        value={searchQuery}
        onChange={handleSearch}
        placeholder="Search for a car..."
        className="border-2 border-[#e7eef6] w-full h-[44px] rounded-full p-2 pl-10 pr-12 focus:outline-none focus:border-[#3563e9]"
      />
      {searchQuery && filteredCars.length > 0 && (
        <div className="absolute z-50 w-full mt-2 bg-white rounded-lg shadow-lg border border-[#e7eef6] max-h-[400px] overflow-y-auto">
          <div className="p-4">
            {filteredCars.map((car) => (
              <Card
                key={car._id}
                className="mb-4 shadow-lg border rounded-md p-2 hover:bg-gray-100"
              >
                <CardHeader>
                  <CardTitle className="flex justify-between items-center">
                    {car.name}
                    <Image
                      src="/heart.png"
                      alt="Favorite"
                      width={20}
                      height={20}
                    />
                  </CardTitle>
                  <CardDescription>{car.type}</CardDescription>
                </CardHeader>
                <CardContent className="flex flex-col items-center gap-4">
                  <Image
                    src={urlFor(car.image).url()}
                    alt={car.name}
                    width={220}
                    height={68}
                  />
```

main | 0 0 | 0 | Connect | Live Share | You, 4 hours ago | Ln 35, Col 45 | Spaces: 2 | UTF-8 | LF | TypeScript JSX | VS Browser | Go Live | Prettier



Conclusion

The "Dynamic Frontend Components" project successfully demonstrates the creation of a user-friendly, filterable and searchable car rental interface. The modular approach ensures scalability and ease of future development.