Overview

This document outlines the development of several essential features for a **car rental ecommerce website**, focusing on:

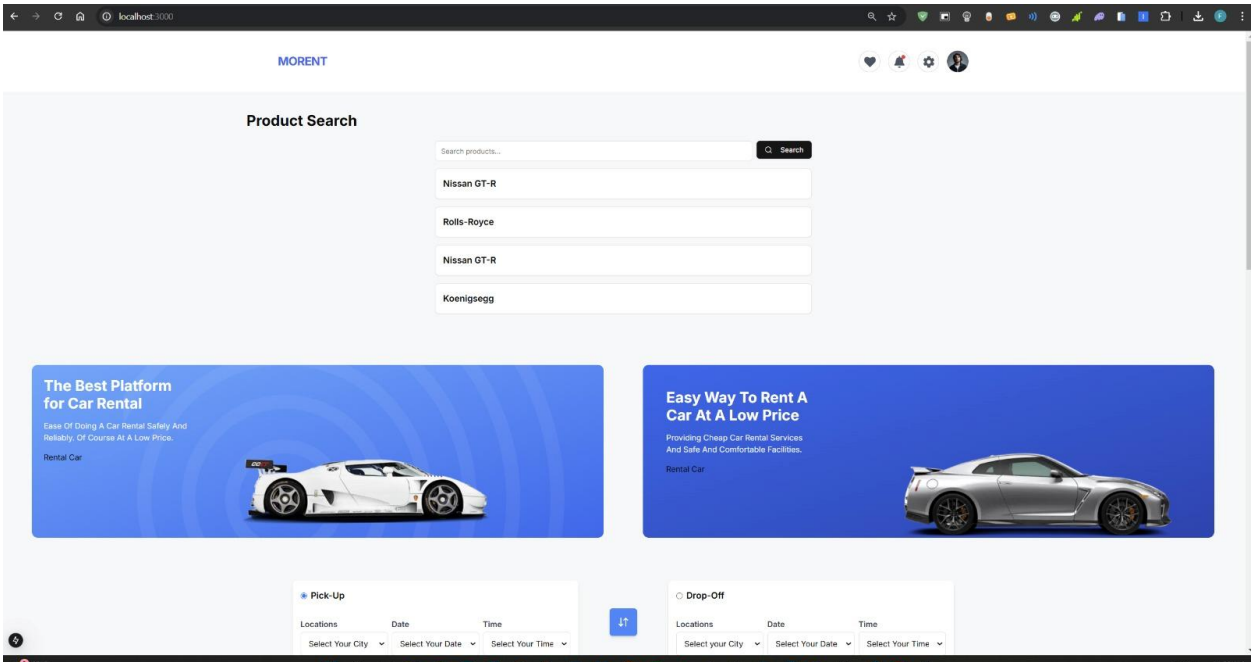
- Filter search section
- Pagination
- Dynamic routing
- Product listing page with dynamic data
- Product detail page
- Working category filters

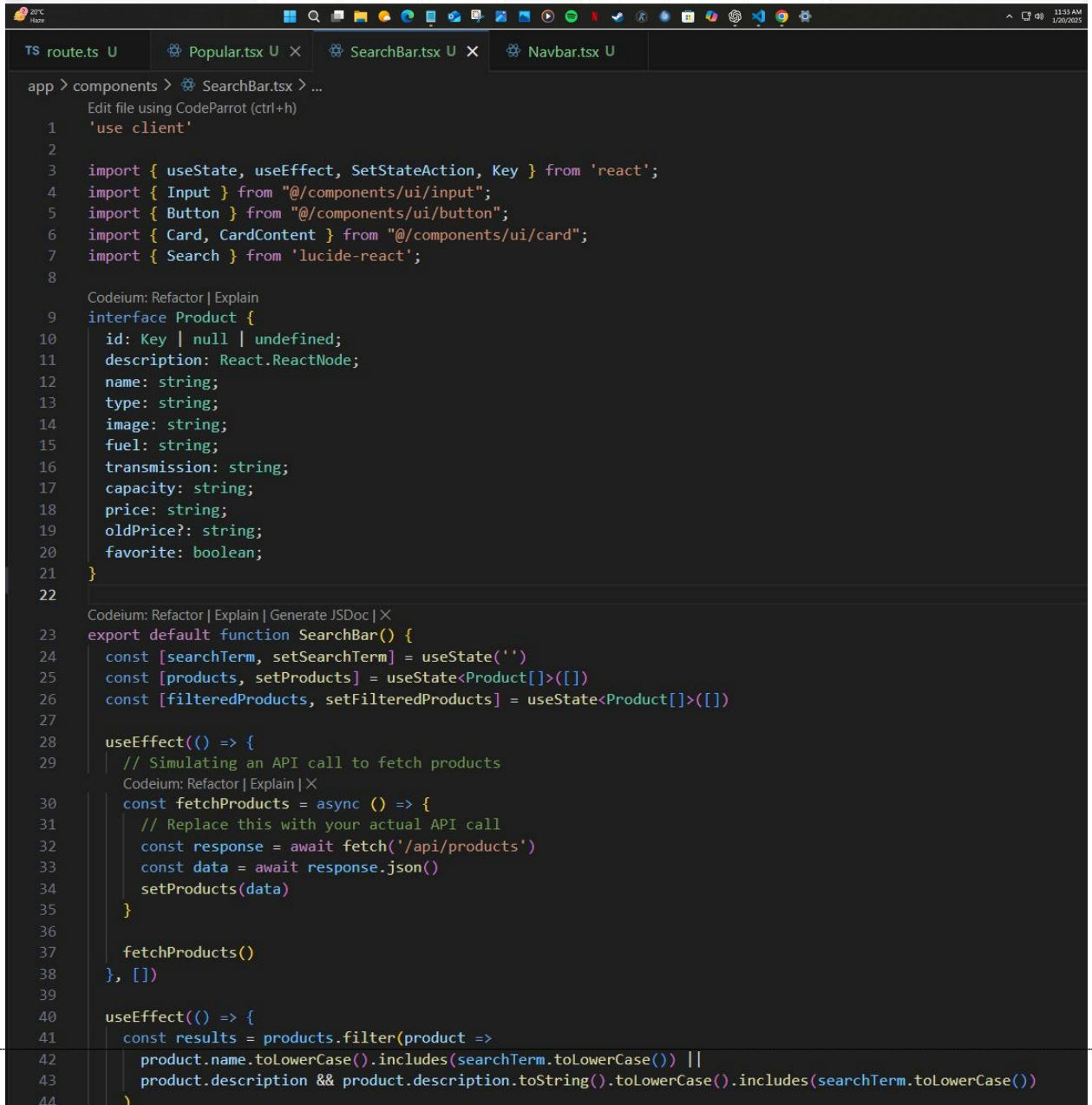
These components are designed to enhance the **user experience** by providing seamless navigation and efficient data handling.

Filter Search Section Component

The **Filter Search Section** allows users to refine their search results based on specific criteria, such as:

- Car brand (e.g., Koenigsegg, Rolls-Royce, Lamborghini)
- Price range
- Availability (e.g., currently rented or available)
- Transmission type (automatic or manual) **Features:**
- Real-time Filtering: Users can see the filtered results update instantly without refreshing the page.
- Responsive Design: Optimized for both desktop and mobile devices.
- Performance: Uses debouncing techniques to improve search performance.


localhost:3000
MORENT
Product Search
Search products...
Search
Nissan GT-R
Rolls-Royce
Nissan GT-R
Koenigsegg
The Best Platform for Car Rental
Ease Of Doing A Car Rental Safely And Reliably. Of Course At A Low Price.
Rental Car
Easy Way To Rent A Car At A Low Price
Providing Cheap Car Rental Services And Safe And Comfortable Facilities.
Rental Car
Pick-Up
Locations
Date
Time
Select Your City
Select Your Date
Select Your Time
Drop-Off
Locations
Date
Time
Select your City
Select Your Date
Select Your Time


route.ts U
Popular.tsx U
SearchBar.tsx U
Navbar.tsx U
app > components > SearchBar.tsx > ...
Edit file using CodeParrot (ctrl+h)
'use client'
import { useState, useEffect, SetStateAction, Key } from 'react';
import { Input } from "@/components/ui/input";
import { Button } from "@/components/ui/button";
import { Card, CardContent } from "@/components/ui/card";
import { Search } from 'lucide-react';
Codeium: Refactor | Explain
interface Product {
id: Key | null | undefined;
description: React.ReactNode;
name: string;
type: string;
image: string;
fuel: string;
transmission: string;
capacity: string;
price: string;
oldPrice?: string;
favorite: boolean;
}
Codeium: Refactor | Explain | Generate JSDoc |
export default function SearchBar() {
const [searchTerm, setSearchTerm] = useState('')
const [products, setProducts] = useState<Product[]>([])
const [filteredProducts, setFilteredProducts] = useState<Product[]>([])
useEffect(() => {
// Simulating an API call to fetch products
Codeium: Refactor | Explain |
const fetchProducts = async () => {
// Replace this with your actual API call
const response = await fetch('/api/products')
const data = await response.json()
setProducts(data)
}
fetchProducts()
}, [])
useEffect(() => {
const results = products.filter(product =>
product.name.toLowerCase().includes(searchTerm.toLowerCase()) ||
product.description && product.description.toString().toLowerCase().includes(searchTerm.toLowerCase())

Technologies Used:

- Frontend: React components for interactivity.
- Backend: GROQ queries to fetch filtered data from Sanity CMS.

Pagination

Pagination was implemented to improve the user experience by:

- Dividing the product listing into smaller, more manageable pages.
- Displaying navigation buttons (e.g., Previous, Next) for easy browsing.

Features:

- Dynamically calculates the total number of pages based on the number of products.
- Shows a limited number of pagination links to avoid clutter.
- Highlights the current page for better visibility.

Implementation:

- API Integration: The backend API returns paginated data based on the requested page number and page size.
- Frontend Logic: React handles dynamic rendering of pages and pagination links.

Dynamic Routing

Dynamic routing ensures scalability and improves the navigational flow of the website. Examples include:

- Product Listing Page: /products
- Product Detail Page: /products/[id] (e.g., /products/12345 for a specific car) • Category Filter Pages: /categories/[category] (e.g., /categories/sports) **Benefits:**
- Enables sharing of specific car details or filtered results through unique URLs.
- Seamless integration with the Next.js router for server-side rendering (SSR) or static site generation (SSG).

route.ts U | Popular.tsx U | SearchBar.tsx U | Navbar.tsx U

app > api > products > route.ts > PRODUCT_QUERY

Edit file using CodeParrot (ctrl+h)

```ts
import { NextResponse } from 'next/server';
import { createClient } from 'next-sanity';

// Sanity client setup
const sanityClient = createClient({
  projectId: 'bpqk9m66', // Replace with your Sanity project ID
  dataset: 'production', // Replace with your dataset name
  apiVersion: '2021-08-31', // Replace with your preferred API version
  useCdn: true, // Use true for faster reads if no need for fresh data
});

// GROQ query to fetch products
const PRODUCT_QUERY = `
  *[_type == "car" && "popular" in tags] {
  name,
  brand,
  type,
  fuelCapacity,
  transmission,
  seatingCapacity,
  pricePerDay,
  originalPrice,
  "imageUrl": image.asset->url
}
`;

Codeium: Refactor | Explain | Generate JSDoc | X
export async function GET() {
  try {
    // Fetch data from Sanity
    const products = await sanityClient.fetch(PRODUCT_QUERY);
    return NextResponse.json(products);
  } catch (error) {
    console.error('Error fetching products:', error);
    return NextResponse.json(
      { error: 'Failed to fetch products' },
      { status: 500 }
    );
  }
}
```

Product Listing Page with Dynamic Data

The **Product Listing Page** fetches dynamic data from Sanity CMS and displays a grid of available cars with key information, including:

- Car name
- Price
- Thumbnail image
- Short description **Key Features:**
- Dynamic Data: Automatically updates when new products are added to the database.
- Lazy Loading: Loads images and data as the user scrolls, reducing initial load time.

Example:

```
fetch('/api/products')
  .then((response) => response.json())
  .then((data) => setProducts(data));
```

Product Detail Page

The **Product Detail Page** provides detailed information about a specific car, including:

- *High-resolution images*
- *Full description*
- *Specifications (e.g., horsepower, engine type, seating capacity)*
- *Booking options* ***Implementation:***
- *Fetches data dynamically using the car's unique ID.*
- *Includes a "Back to Listing" button for easy navigation.*

Working Category Filters

The **Category Filter** allows users to browse cars based on predefined categories, such as:

- Sports Cars
- Luxury Cars
- Economy Cars

route.ts U | CategoryFilters.tsx U

app > components > CategoryFilters.tsx > CategoryFilters

```tsx
Edit file using CodeParrot (ctrl+h)
"use client"

import { useState, useEffect } from "react"
import Image from "next/image"
import { Slider } from "@/components/ui/slider"
import { Checkbox } from "@/components/ui/checkbox"
import { Button } from "@/components/ui/button"
import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from "@/components/ui/card"
import { Badge } from "@/components/ui/badge"
import { Skeleton } from "@/components/ui/skeleton"
import { mockItems } from "@/data/mockItems"

const categories = ["SUV", "Sports", "Sedan", "Coupe", "Convertible", "Hatchback", "Wagon"]

const brands = ["Audi", "Lamborghini", "Rolls-Royce", "Koenigsegg", "Mercedes", "Ferrari"]

Codeium: Refactor | Explain | Generate JSDoc | ×
export default function CategoryFilters() {
  const [selectedCategories, setSelectedCategories] = useState<string[]>([])
  const [items, setItems] = useState<typeof mockItems>([])
  const [filteredItems, setFilteredItems] = useState<typeof mockItems>([])
  const [priceRange, setPriceRange] = useState([0, 3000000])
  const [selectedBrands, setSelectedBrands] = useState<string[]>([])
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    Codeium: Refactor | Explain | Generate JSDoc | ×
    const fetchData = async () => {
      // Simulate API call
      await new Promise((resolve) => setTimeout(resolve, 1000))
      setItems(mockItems)
      setFilteredItems(mockItems)
      setLoading(false)
    }
    fetchData()
  }, [])

  Codeium: Refactor | Explain | Generate JSDoc | ×
  const toggleCategory = (category: string) => {
    setSelectedCategories((prev) =>
      prev.includes(category) ? prev.filter((c) => c !== category) : [...prev, category],
    )
  }

  Codeium: Refactor | Explain | Generate JSDoc | ×
  const handlePriceRangeChange = (value: number[]) => {
    setPriceRange(value)
  }
```

Key Features

- ***Dynamic Querying:*** *GROQ queries fetch filtered data based on the selected category.*
- ***Interactive UI:*** *Clicking on a category updates the listing page without a full page reload.*

Conclusion

These features collectively enhance the functionality and usability of the car rental e-commerce website. By implementing efficient filtering, pagination, dynamic routing, and detailed product pages, the platform delivers an ***engaging*** *and* ***user-friendly experience*** *for potential customers.*