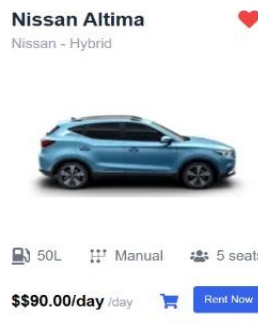
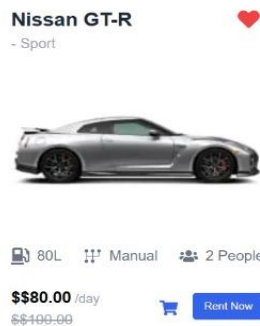


Popular Cars

[View All](#)



[Show more cars](#)

3 Cars

Type

☐ Sport
 ☐ SUV
 ☐ MPV
 ☐ Sedan
 ☐ Coupe
 ☐ Hatchback

Capacity


☐ 2 People
 ☐ 4 People
 ☒ 6 People
 ☐ 8 or More

Price

\$0\$73

Rolls-Royce

- SUV




70L

Manual

6 People

\$72.00 /day



Rent Now

Show more cars

1 Cars

Shopping Cart

Nissan GT-R

Model: - Sport

Price: \$/day



Days:

Remove

Koenigsegg

Model: - Sport

Price: \$NaN/day



Days:

Remove

Rolls-Royce

Model: - SUV

Price: \$NaN/day



Days:

Remove

Continue Shopping

Order Summary

Subtotal	\$0.00
Tax	\$0.00
Total	\$0.00

Proceed to Checkout

Features and Implementation

1. *Filters Component*

- Dynamically filters cars by type, seating capacity, and price.
- Fetches data from the Sanity CMS and applies filters on the client side.
- Utilizes controlled inputs for range sliders and checkboxes.

2. *Cart Context*

- Provides a context for managing the shopping cart, including adding, updating, and removing items.
- Persists the cart in localStorage to maintain state across page reloads.

- Includes a reusable useCart hook for accessing cart-related actions and state.

3. *Search Context*

- Offers a live search functionality with real-time suggestions.
- Filters cars based on user input, matching against names, brands, types, and tags.

4. *Reusable Components*

- ProductCard: Displays individual car details and provides options to add to the cart or navigate to a details page.
- CartItem: Handles cart item display and quantity adjustments.
- CartSummary: Shows order summary with subtotal, tax, and total calculations.

5. *Design and Usability*

- Uses Tailwind CSS for styling, ensuring responsiveness and a modern look.
- Provides skeleton loaders for a better user experience during data fetching.

Potential Improvements

1. *Pagination and Infinite Scroll*

- Implement pagination or infinite scrolling to handle large datasets instead of rendering all filtered items at once.

2. *Server-Side Filtering*

- Move filtering logic to the server or CMS to improve performance for large datasets.

3. *Enhanced Search*

- Add fuzzy search or autocomplete to improve search accuracy and user experience.

4. *Validation and Error Handling*

- Include user-friendly error messages and validations for cart operations and filters.

5. *Optimization*

- Optimize image loading using a placeholder or lazy loading to improve page performance.
- Memoize components like ProductCard to avoid unnecessary re-renders.

6. *Testing*

- Add unit and integration tests using a framework like Jest or React Testing Library to ensure robust functionality.

```
File Edit Selection View Go Run Terminal Help < -> hackathon governer house
EXPLORER
HACKATHON GOVERNER HOUSE
src
  app
    pages
      dashboard
        page.tsx
      detailcarRent
        page.tsx
      studio \[...tool\]
        page.tsx
      favicon.ico
      # globals.css
      layout.tsx
      page.tsx
    components
      contexts
        CarContext.tsx
        SearchContext.tsx
      ui
        CarGrid.tsx
        CarItem.tsx
        CartSummary.tsx
        Footer.tsx
        Header.tsx
        Hero.tsx
        LocationComponent.tsx
        Payment.tsx
        ProductCard.tsx
      lib
        utils.ts
        sanity
        lib

src > components > contexts > SearchContext.tsx > * SearchContextType > searchQuery
1 'use client'
2
3 import React, { createContext, useState, useContext, ReactNode, useEffect } from 'react';
4 import { client } from '@sanity/lib/client';
5
6 interface Car {
7   _id: string;
8   name?: string;
9   brand?: string;
10  type: string;
11  tags?: string[];
12 }
13
14 interface SearchContextType {
15   searchQuery: string;
16   setSearchQuery: (query: string) => void;
17   suggestions: string[];
18   setSuggestions: (suggestions: string[]) => void;
19 }
20
21 const SearchContext = createContext<SearchContextType | undefined>(undefined);
22
23 export const useSearch = () => {
24   const context = useContext(SearchContext);
25   if (context === undefined) {
26     throw new Error('useSearch must be used within a SearchProvider');
27   }
28   return context;
29 };
30
31 export const SearchProvider: React.FC<{ children: ReactNode }> = ({ children }) => {
32   const [searchQuery, setSearchQuery] = useState('');
33   const [suggestions, setSuggestions] = useState<string[]>([]);
34   const [allCars, setAllCars] = useState<Car[]>([]);
35
36   Pieces: Comment | Pieces: Explain
```

```
<div className="flex-1 flex flex-col justify-between">
  <div className="flex justify-between">
    <div>
      <h3 className="font-medium">{name}</h3>
      <p className="text-sm text-gray-500">
        {brand} - {type}
      </p>
    </div>
    <button onClick={onRemove} className="text-gray-400 hover:text-gray-600">
      <X className="w-5 h-5" />
    </button>
  </div>

  <div className="flex items-center justify-between">
    <div className="flex items-center gap-2">
      <button
        className="h-8 w-8 flex items-center justify-center border rounded-full"
        onClick={() => onUpdateDays(Math.max(1, days - 1))}
      >
        <Minus className="w-4 h-4" />
      </button>
      <span className="w-12 text-center">{days} days</span>
      <button
        className="h-8 w-8 flex items-center justify-center border rounded-full"
        onClick={() => onUpdateDays(days + 1)}
      >
        <Plus className="w-4 h-4" />
      </button>
    </div>
    <div className="text-right">
      <p className="font-semibold">${(pricePerDay * days).toFixed(2)}</p>
      <p className="text-xs text-gray-500">${pricePerDay.toFixed(2)}/day</p>
    </div>
  </div>
```