Smart Airport Ride Pooling Backend - DSA Approach

DSA Approach (Human-Readable Explanation)

The core problem in the Smart Airport Ride Pooling system is to assign incoming passenger ride requests to available cabs in real time while respecting constraints like seat capacity, luggage space, and detour tolerance. Since requests arrive continuously and decisions must be made instantly, the system uses a Greedy matching approach supported by efficient data structures.

Algorithm Used:
The system follows a Greedy First-Fit strategy similar to bin-packing problems. Each incoming ride request is matched to the first cab that can accommodate it based on available seats and luggage capacity. If no suitable cab is found, a new cab is created.

Step-by-Step Logic:
1. Fetch all active cabs from the database.
2. For each cab:
   - Check if seats are available.
   - Check if luggage capacity is available.
   - Try to acquire a Redis lock for that cab (to prevent concurrent conflicts).
   - If lock is acquired and constraints are satisfied, assign the passenger.
3. If no cab fits, create a new cab and assign the passenger.

This approach is fast and practical for real-time systems where decisions must be taken immediately without heavy computations.

Time Complexity Analysis:
Let n be the number of active cabs.

For each incoming request:
- Fetch active cabs → O(n)
- Loop through cabs → O(n)
- Seat check → O(1)
- Luggage check → O(1)
- Redis lock attempt → O(1)
- Database update → O(1)

Total Time Complexity per request = O(n)

Since checks inside the loop are constant time, performance remains efficient even with

many cabs.

Space Complexity:
- Active cabs stored → O(n)
- Ride requests stored → O(r)
Total Space Complexity = O(n + r)

Concurrency Consideration:
To prevent race conditions when multiple users try to book the same cab at the same time, Redis distributed locks are used. Only one request can update a cab at a time, ensuring seat counts never become incorrect.

Why Greedy Approach?
The system works in real time, so heavy algorithms like dynamic programming or route optimization are not suitable. A greedy strategy provides fast decisions and scales well for high traffic systems.

Conclusion:
The chosen DSA approach balances speed, simplicity, and scalability. It ensures quick matching, prevents overbooking using locks, and supports high request throughput suitable for production environments.