Smart Airport Ride Pooling Backend - System Design Documents

1. Concurrency Handling Strategy

The system handles multiple booking requests at the same time using a Redis-based distributed locking mechanism.
This prevents race conditions when many users try to book seats in the same cab simultaneously.

How it works:
• When a booking request arrives, the system attempts to lock a specific cab using Redis.
• Only one request can acquire the lock for a cab at a time.
• Once locked, the system checks seat and luggage availability and updates the database.
• After the update, the lock automatically expires using TTL (time-to-live), allowing the next request to proceed.

Why this approach:
• Prevents double booking
• Ensures seat counts never go negative
• Supports high concurrency
• Works across multiple server instances

Flow Diagram (Conceptual):

User Request → API Server → Redis Lock → Check Seats → Update DB → Release Lock

2. Database Schema & Indexing Strategy

The system uses PostgreSQL to store persistent data related to cabs and ride requests.

Main Tables:

Cabs Table:
• id (Primary Key)
• capacity
• availableSeats
• luggageCapacity
• availableLuggage
• status
• createdAt
• updatedAt

RideRequests Table:
• id (Primary Key)
• passengerId
• originLat
• originLng
• destLat
• destLng
• seatCount
• luggageUnits
• detourTolerance
• cabId (Foreign Key reference to Cabs)
• status
• createdAt
• updatedAt

Indexing Strategy:

• Primary index on cab.id for fast lookup
• Index on cab.status to quickly find active cabs
• Index on RideRequests.cabId for fast passenger retrieval per cab
• Index on RideRequests.status to filter active vs cancelled requests

Benefits:
• Faster search performance
• Reduced query latency
• Scales well with increasing ride data

3. Dynamic Pricing Formula Design

The pricing model calculates ride cost dynamically based on distance, number of seats, and luggage units.

Pricing Components:
• Base Fare: Fixed minimum charge
• Distance Cost: Based on estimated travel distance
• Seat Multiplier: More seats increase cost
• Luggage Multiplier: Extra luggage adds surcharge

Formula Used:

Price = BaseFare

+ (Distance × PerKmRate)
+ (SeatCount × SeatFactor)
+ (LuggageUnits × LuggageFactor)

Example:
If:
• Base Fare = 50
• Distance = 10 km
• SeatCount = 2
• LuggageUnits = 1

Then:
Price = 50 + (10×10) + (2×20) + (1×15)
= 50 + 100 + 40 + 15
= 205

Why Dynamic Pricing:
• Fair pricing based on usage
• Encourages ride sharing
• Handles varying demand
• Scales with real-time system needs