Title: ***Three-Tier Distributed Solution for a Car Hire Company***

This document describes the design of a three-tier distributed architecture for a hypothetical car hire company, including a hand-drawn diagram and an explanation of its benefits, drawbacks, and distributed design principles.

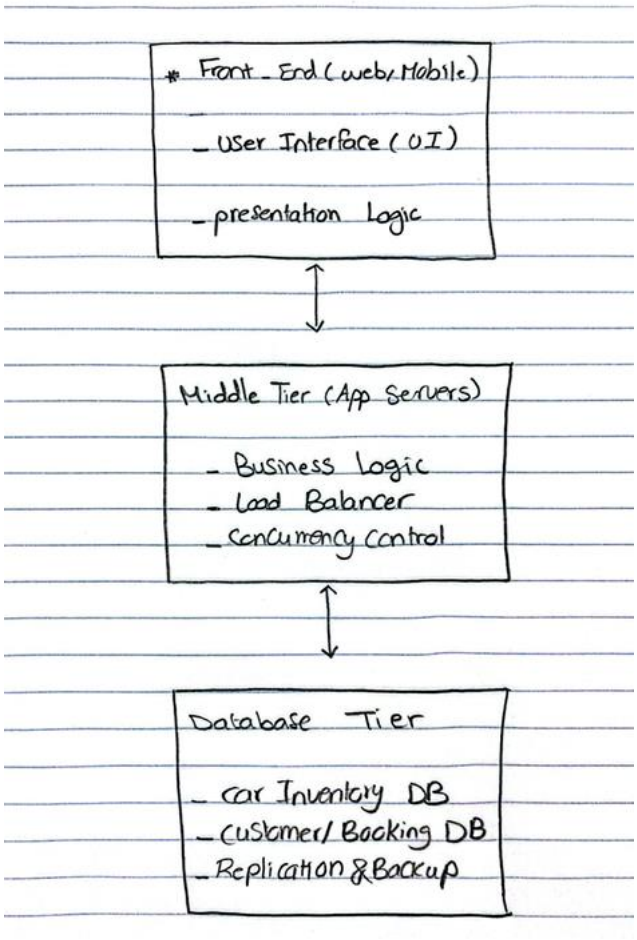

Figure 1. Three-Tier Distributed Architecture for Car Hire System

Three-Tier Architecture Overview

- **Presentation Layer (Front-End):** Web/Mobile UI for customers and admins. Handles searching cars, prices, bookings, and admin tasks.
- **Application Layer (Middle-Tier):** Application servers handling business logic, booking rules, payments, and load balancing.

- **Data Layer (Database):** Stores car inventory, customer accounts, reservations, and payments. Supports replication and backup.

Benefits

- *Performance*: Load distributed across tiers, caching, and optimized queries improve response.
- *Scalability*: Each tier scales independently (extra app servers, replicated databases).
- *Failure handling*: Replication, failover, and graceful error handling minimize downtime.
- *Maintainability*: Separation of concerns makes updates easier (UI, logic, data evolve separately).

Drawbacks

- Extra latency because requests must pass through three tiers.
- More components = higher complexity and more potential failure points.

Distributed Design Principles Applied

- **Separation of concerns** (UI, business logic, data).
- **Replication & backups** (for databases).
- **Load balancing** (application servers).
- **Failover mechanisms** (backup servers, database replicas).
- **Modularity** (independent evolution of tiers).