# Hotel Reservation System

EDS: 5.000 hotels chain, 1 mill rooms. Customer pay in full.
Reservation through http/app. Cancel: yes. Allow 10% overbooking.
Room price dynamically change

NFR: High concurrency
Latency (high response)

BEE: $\frac{1\text{ mill} \times 0.7}{3} = 133,333 \approx 240.000$ Daily Reservations (70% rooms occupied)

$\frac{240.000}{10^5 \text{ sec. in day}} = \sim 3$ Reservations per second. Not high.

QPS rooms = 3
Order booking page = 30
View hotel/room detail = 300

HLD:

- API design

Hotel / Room / Reservation related API's

Post /v1/reservations

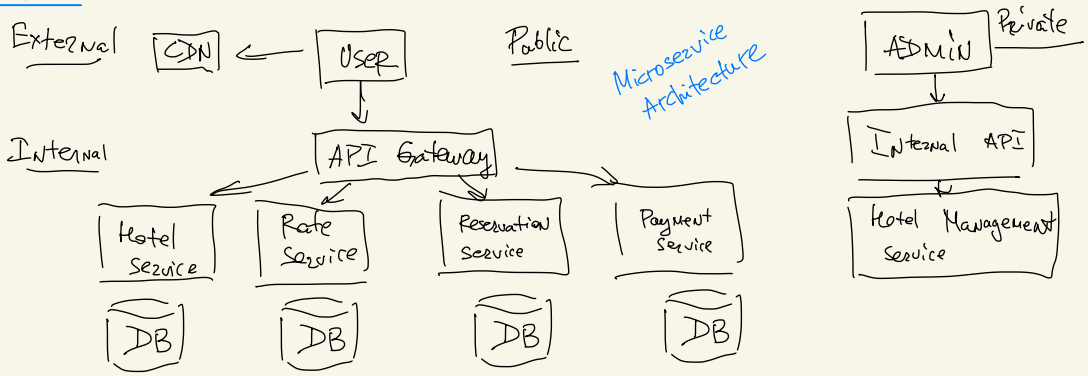
{ start
end
hotelId
roomId ← room Type
reservation ID ← idempotency key }

- Data Model

Relational DB (read-heavy, ACID, clear relational structure between entities)

User reserves type of room: Standard, King-size, Queen size, etc.

## High Level Design:



DDD :

1. Improved data model
2. Concurrency Issues
3. Scaling
4. Data inconsistency in the microservice architecture

1. RDBMS 5,000 hotels x 20 types of rooms
   x 2 years x 365 days = 73 mill rows.
   Enough for single DB but SPF!
   HA : DB replication across multiple AZ's.
   Composite Key (hotel_id, room_type_id, data)
   - Store current and future data
   - Sharding by hash (hotel_id) % number_of_servers

2. Avoid double booking.
   - click the book btn multiple times
   - book the same room at the same time
   - client-side implement. hide button
   - Idempotent API (key as double action restriction)
     idempotency_key != reservation_id

   Serializable isolation level.
   - Pessimistic Locking → User1 ●lock ← User2
   - Optimistic Locking → version number / timestamp . without any lock
   - DB constraints → easy to implement DB constraint

3. DB Sharding : hotel_id % number_of_dbs
   Caching : Redis → TTL, LRU
   Each shard do Async Update Cache (Redis)

4. Data inconsistency in Microservices approach.
   But we can use centralized DB among with several services.
   - Two-phase commit. (2PC) . DB protocol to guarantee atomic transaction commit. Blocking protocol.
   - Saga. Sequence of local transactions. Provide eventual consistency.