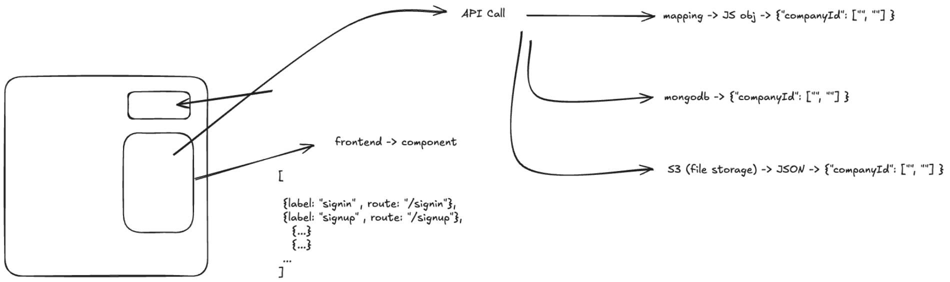
1. normal users
2. airbnb owners
3. small businesses
4. large companies


API Call
mapping -> JS obj -> {"companyId": ["", ""] }
mongodb -> {"companyId": ["", ""] }
S3 (file storage) -> JSON -> {"companyId": ["", ""] }
frontend -> component

```
[

 {label: "signin" , route: "/signin"},
 {label: "signup" , route: "/signup"},
   {...}
   {...}
 ...
]
```

1. Assume this is a mobile app (weekly, biweekly), ---> release on the playstore and app store

2. Translations

3. requirements can become complex to just handle on frontend

Airbnb Software design

What the user will be able to do on the platform ?

Functional requirements:

1. Users should be able to list the hotels/airbnbs

2. Users should be able to filter the hotels based on price ranges, locations, and many more .....

3. Users should be able make reservations for rooms in the hotel.

4. Users should be able see the details of their reservations and if reqd, cancel it as well.
5. Users should be able to add reviews of the hotels they booked

behaviour of the platform

Non functional requirements:

1. Double charges should be avoided and double booking as well.

2. The system should be able to handle concurrency (during peak season)

3. The system should ensure if during the booking any operation fails then the complete booking should be discarded.

4. More users will be searching for hotels rather than booking the hotels. Search can be 10x to 20x of final booking requests. Overall system is a lot read heavy.

5. 10,000 Hotels support we should atleast, assume that every hotel has 100 rooms -> total 10^6 rooms -> 1Million rooms

6. 1B - MAU, 1% of MAU -> 10M DAU

Calculations:

- Read requests: (Searches)

10M DAU -> atleast 10 queries each user does → load testing

Total search request we get in a day -> 100M

Search req per sec -> (100M)/ 10^5 -> 10^8 / 10^5 -> 10^3 qps

Peak load -> 10X peak load -> 10^4qps

- Write requests: (Bookings)

-> 50% hotel rooms are always booked -> 2night stay

-> 0.5M / 2 -> 0.25M bookings per day

-> peak load 2x of it -> 0.5 million -> 5 * 10^5

-> per sec booking load -> 5 * 10^ 5 / 10^5 -> 5qps booking

Api contract designing:

GET /api/v1/hotels. -> list all the hotels

 -> /api/v1/hotels?price_start=1000&price_end=7000&city=bengaluru&check_in=...&check_out=...

GET /api/v1/hotels/:hotelId -> list the details of a particular hotel/airbnb

POST /api/v1/hotels => create the hotel
{name:"", address: "", location: "", ...}

DELETE /api/v1/hotels/:hotelId -> delete the hotel

GET /api/v1/hotels/:hotelId/room/:roomId -> details of a room

POST /api/v1/hotels/:hotelId/room -> add a room

DELETE /api/v1/hotels/:hotelId/room/:roomId -> delete a room

PUT /api/v1/hotels/:hotelId/room/:roomId -> update the room details

GET /api/v1/bookings -> all the bookings of a user

GET /api/v1/bookings/:bookingId -> details of a particular bookings

POST /api/v1/bookings -> create a new booking
{ hotelId, roomId, startDate, endDate, numberOfGuests .... }

DELETE /api/v1/bookings/:bookingId -> cancel a booking

google3

monorepo

↑

-> db -> offerId - entityId (can be a hotel or city) - entityType

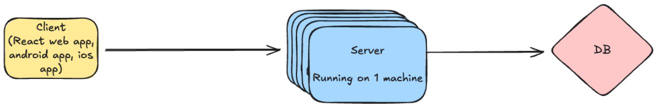
-> /api/v1/offers?city=bengaluru -> List<Hotels> -> minimum discount - maximum discount vertical scaling

horizontal scaling

monolith
project

One NodeJS project
- Review
- booking
- admin
- listing
- user

scaleup -> adding machines
scaledown -> removing machines



autoscaling

warm instances

Ruby on rails -> monolith → code a gon (annual coding contest)

100K concurrent users - existing load 3K - 5K users - peak load 10K

< 4 weeks

DB - MySQL (single AWS RDS mysql data store instance)

AWS Elasticcache - redis cache

EC2 machines

1. DB front efforts -> to optimise searches, indexes, cache, N+1 queries
Bulk writes

2. finding access pattern

3. test load was fired on the infrastructure 2-3 mins before the start of contest

4. random load timer

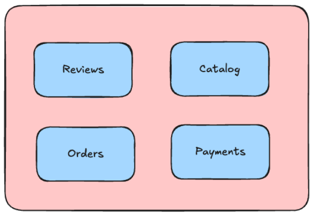
8 - 8.10

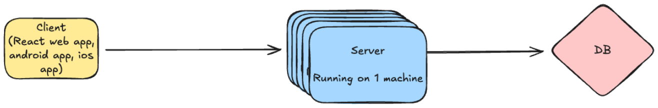
5. fine tuning db - > db instance was veritcally scaled

75K concurrent users , 1M code submissions , < 150ms

load testing

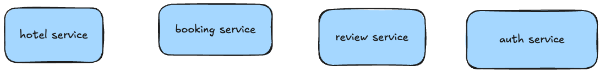
Ecom app





- we might get more load on the hotel cataloging service when compared to the booking or review relates services.

microservice

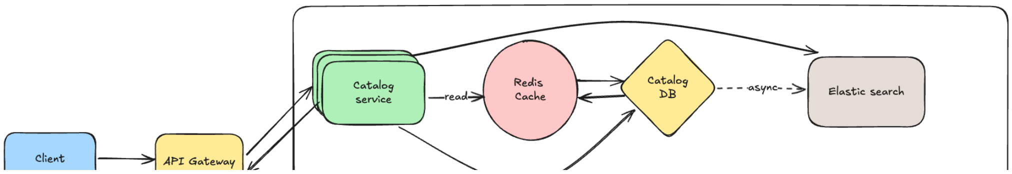


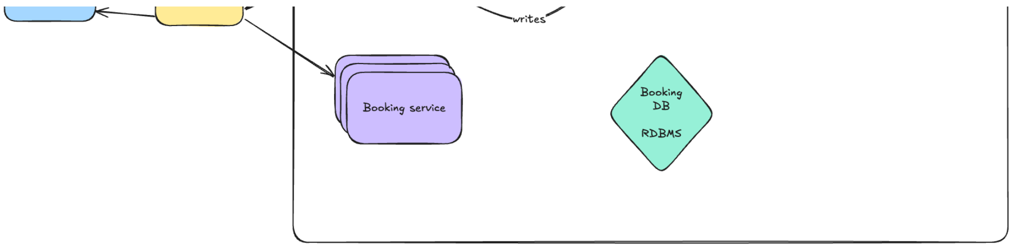
interservice communication

inverted index , lucene index

API Gateway -> Rate limiting , auth,

CDC - change data capture





Problems:

1. We should save our users from double booking. --- Idempotency ✓

2. Concurrent bookings ---- Controlling isolation levels, pessimistic locking, optimistic locking, distributed locking

3. Distributed transactions - 2Phase commit , Saga - Orchestration | chroreography

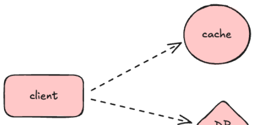
4. How the DBs will scale from here

1. Write through cache

a. write goes to both cache and database

b. writes are expensive

c. cache always has the latest data

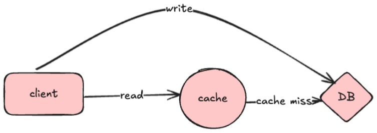


2. Write around cache

a. If a write comes, that write only goes to a db

b. faster writes compared to write through cache

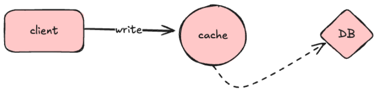
c. your first read will be slow



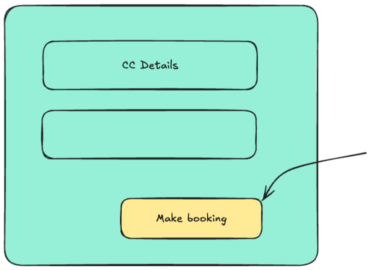
3. Write back cache

a. fastest writes, when a write comes
it only writes to the cache

b. risk of data loss



cache updates the db
behind the scenes in a async way
or you can say in a delayed fashion



We have atleast 2 rooms of the required hotel available.

User clicks this button twice !

This can lead to double booking, there is a chance that user is charged twice as well.

1. Naive solution: We disable the button on the frontend.

Problem 1: User not uses this button, and say they are using POSTMAN like client to send the API Request ?

Problem 2: Because this disabling is controlled by JS, may be the user has disabled JS on the browser.

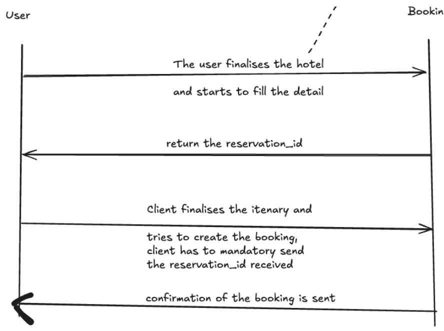
Problem 3: Assume there is 3rd party vendor, they might not put these client side checks.

2. Idempotent APIs --> More widely accepted solution.

In simple terms, indempotency means that if an operation is qualified as
an idempotent operation then, if u apply that operation once or more than once
the outcome will be same.

Idempotence (UK: /ˌɪdɛmˈpoʊtəns/, US: /ˈaɪdəm-/) is the property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application.

Our agenda is to ensure the final booking API is idempotent, that means if you try to do multiple bookings together it should not work.



User

Booking service

The user finalises the hotel

and starts to fill the detail

return the reservation_id

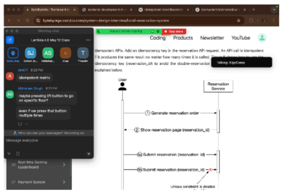
Client finalises the itenary and

tries to create the booking,
client has to mandatory send
the reservation_id received

confirmation of the booking is sent

Log in or sign up to book

Generate a new reservation
in pending state for the user, hence we have a
reservation id (PK -- unique)



bytebytego.com/courses/system-design-interview/hotel-reservation-system

Meeting chat

Lambda 4.0 May 12 Class

ANKIT 9:20 PM

idempotent matrix

Abhishek Singh 9:21 PM

maybe pressing lift button to go on specific floor?

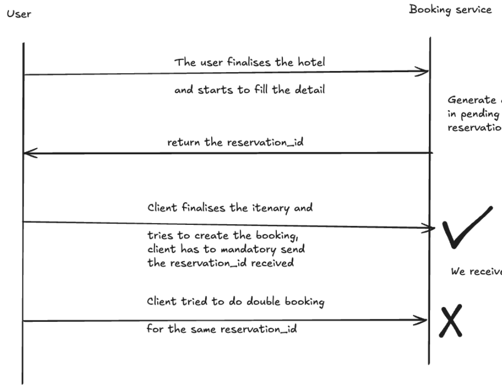
even if we press that button multiple times

Message everyone

Idempotent APIs. Add an idempotency key in the reservation API request. An API call is idempotent if it produces the same result no matter how many times it is called. [illegible] idempotency key (reservation_id) to avoid the double-reservation [illegible] explained below.

Coding
Products
Newsletter
YouTube
Talking: AlgoCamp
User
Reservation Service
① Generate reservation order
② Show reservation page (reservation_id)
(3a) Submit reservation (reservation_id)
(3b) Submit reservation (reservation_id)
Unique constraint is violated
26 Real-time Gaming Leaderboard
27 Payment System

We receive the reservation_id, and complete the booking



User

Booking service

The user finalises the hotel

and starts to fill the detail

return the reservation_id

Client finalises the itenary and

tries to create the booking,
client has to mandatory send
the reservation_id received

Client tried to do double booking

for the same reservation_id

Generate a new reservation
in pending state for the user, hence we have a
reservation id (PK -- unique)

We receive the reservation_id, and complete the booking

The contract from client side req will be having this reservation_id as a mandatory property (we can expect this in headers)

1. If the client doesn't send a reservation_id (idempotency key) we reject the booking req.

2.

1. There is a simple table called as idempotency_keys. This table stores the idempotency keys of those reservation which are completed once. But before we complete the booking, we should check if the same idempotency key is already existing in the table or not ? If it does then reject the booking.

2. May be in the reservation table, we keep a separate column (optional & unique) idempotency key (and in that case we should keep the idempotency key diff than reservation_id)

Action items

1. Think about what all situations can be there where u might need an idempotent api.

2. Now while confirming a booking, we have to insert the idempotency key in the indempotent_keys table along with changing the status of the booking from pending to success, can these two operations exist outside a transaction or they should be bound in a single transaction.

3. Make a simple api which does some side effect (insert, update, delete) and try to make this API idempotent.