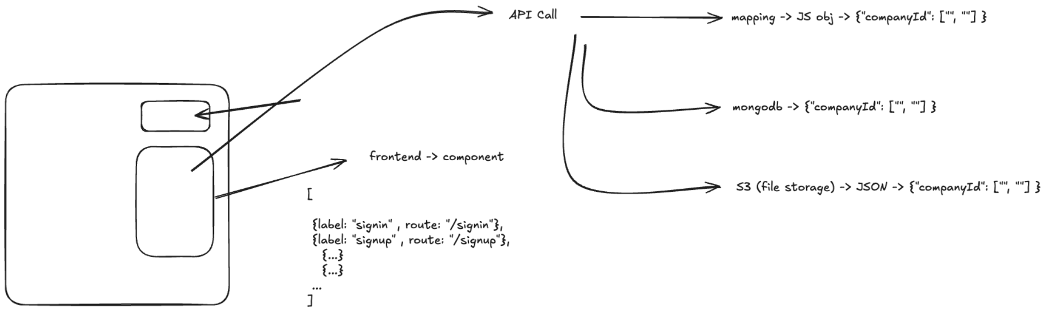
1. normal users
2. airbnb owners
3. small businesses
4. large companies


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

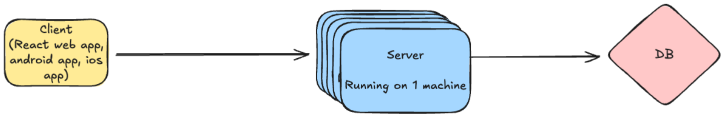
vertical scaling

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

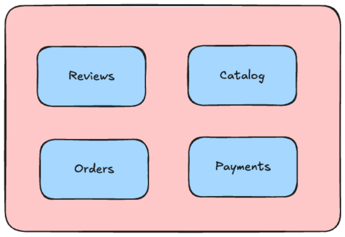
8 - 8.10

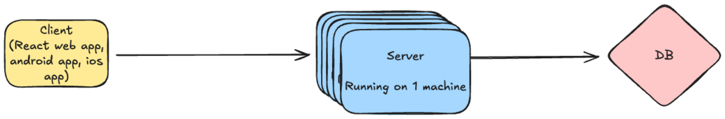
5. fine tuning db - > db instance was veritcally scaled

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

load testing

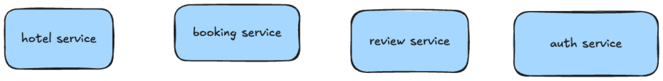
Ecom app





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

microservice

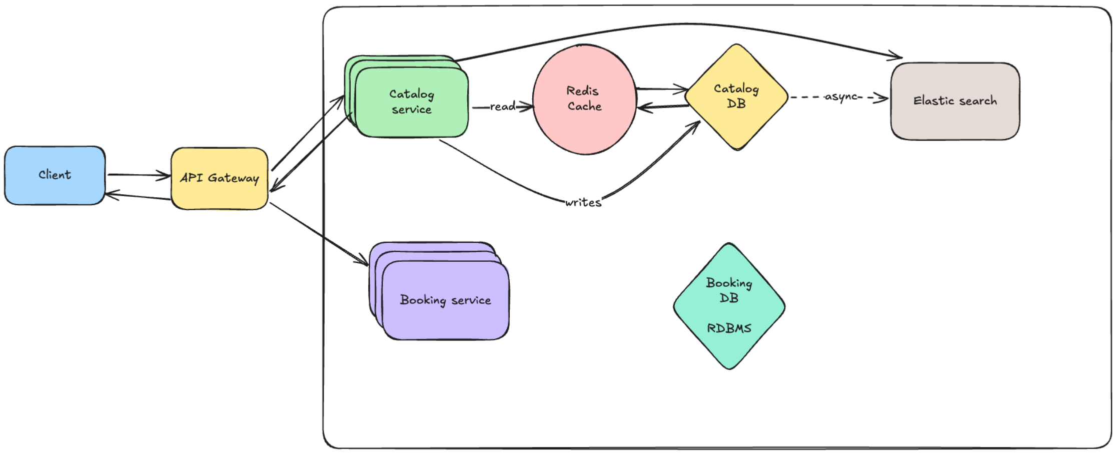


interservice communication

inverted index , lucene index

CDC - change data capture

API Gateway -> Rate limiting , auth,



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

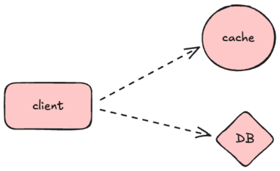
4. How the DBs will scale from here

1. Write through cache

a. write goes to both cache and database

b. writes are expensive

c. cache always has the latest data

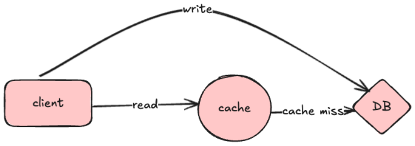


2. Write around cache

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

b. faster writes compared to write through cache

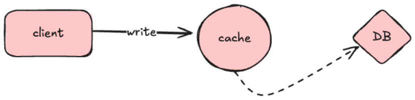
c. your first read will be slow



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