Designing Tinder (as a microservice architecture)

→ Designing front to back i.e. think abt what the user need as features → then think abt how your services are going to be actually broken down → then individual data reqmts. and implementation / services

features we gonna des.:

1) Store Profiles → Images will be stored in profile (5 imgs / user)

2) Recommend matches → No. of active users

3) Note matches → For every swipe you hv a 0.1% chance 0.1% of no. of active users

4) Direct Messaging → Chatting with someone after matching with them.

FEATURE 1 → Storing Profiles

Imgs can be stored in 2 ways → as a file (file sys is used)
→ as a blob (DBMS is used)

(1) Using a Database Management Sys (DBMS) and store imgs as blobs

DBMS provides several features
1) Mutability → not reqd. as instead of changing an image, you just remove old one and insert new one.
2) Transac" → not reqd. in any part of Tinder

Date:
M T W T F S S

3) Indexes (search) → useless for blobs as search is based on content viz 1s & 0s.

4) Access control → useful.

→ To store large objs. separately in a DBMS we gotta implement 'vertical partitioning', i.e. img stored somewhere else but referenced as a col to a dB row

id	image
1234	1 → img 1

in SQL

Gotta do select * here to get img.

distributed file system.

(2) → Using file sys instead (we'll use a DFS)

dB is storin all the data so its gotta hv some reference to the file

stores files

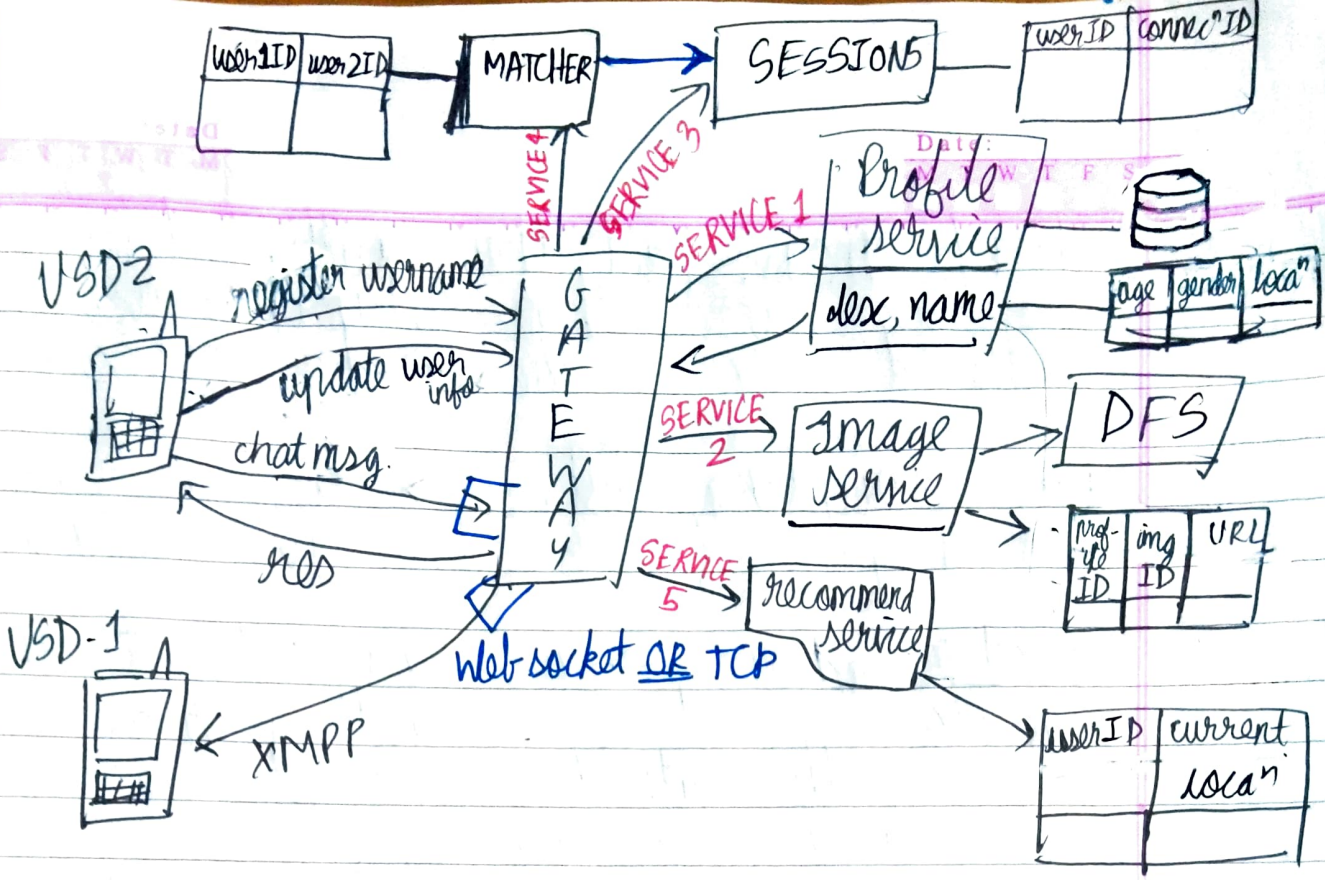
profileID	imageID	file URL

→ in a distributed file sys

(2) Using a file sys and images stored as files,

→ Cheaper
→ Faster (large objects stored separately)
→ Content Delivery Network (CDN) makes accessin static data (the images) really fast.

NOTE: varchar → a set of character data of indeterminate length.
Refers to a data type of a field (or col.) in a DBMS which can hold letters or nos.



MICROSERVICE ARCHITECTURE DIAG OF TINDER

Salient features of this sys.

(1) Client initially registers with username, password. This auth. data is stored in "Profile Service".
Now for any subseq. reqs to a sys client will hv a 'login' token generated that they gonna use for other reqs.

(2) Also for any req. to be authed it doesn't make sense to make profile service to handle → 1st auth, & then talkin to the service the req. is for.
Let Profile service just handle profile auth.
We use a Gateway Service (the only service which'll actually talk to the client) to receive

Matches service checks if you're matched with a particular person & depending on that, it tells session service whether you can chat with another person or not.

∴ When you uninstall the app, you can recover your:

1. matches from the Matches service
2. people you can text, from the sessions service
3. profile from the Profile service

FEATURE 4 - RECOMMENDING MATCHES

Core of recommendn → who in your requested choice (eg:- gender: female, age: 21) is actually geographically close to you.

∴ The Profile service itself will also comm. with a dB which stores 3 things

age	gender	locn

We basically idx this dB on locn and then ~~search the~~ partition the data based on this locn prop. and provide just 1 such partn to a user.

eg:- a user in South Bombay will now hv access to data primarily in ~~South~~ South Bombay only.

This can be done in 2 ways of choosin this main dB we gonna pull out chunks frm

(1) Use a no SQL dB → (eg:- Cassandra, DynamoDB)

(2) Use a SQL dB → (eg:- PostgreSQL) but be sure to "shard" this dB to get out some chunks that we'll send to the user

IMP → be sure to have "leader-elec" at each shard for redundancy

→ This is better → as it uses a doc-collec" model so each doc is itself a very small piece and we just gotta write queries based on "loca" to get a "chunk" that we gonna send to user

Recommenda" service → Basically, this'll just have access to a dB hvin (1) userID (2) current "loca" of user (updated, say, every 2-3 hrs)

And based on this current "loca" of user, this service will talk to "Profile" service to get and serve the reqd. "chunk".