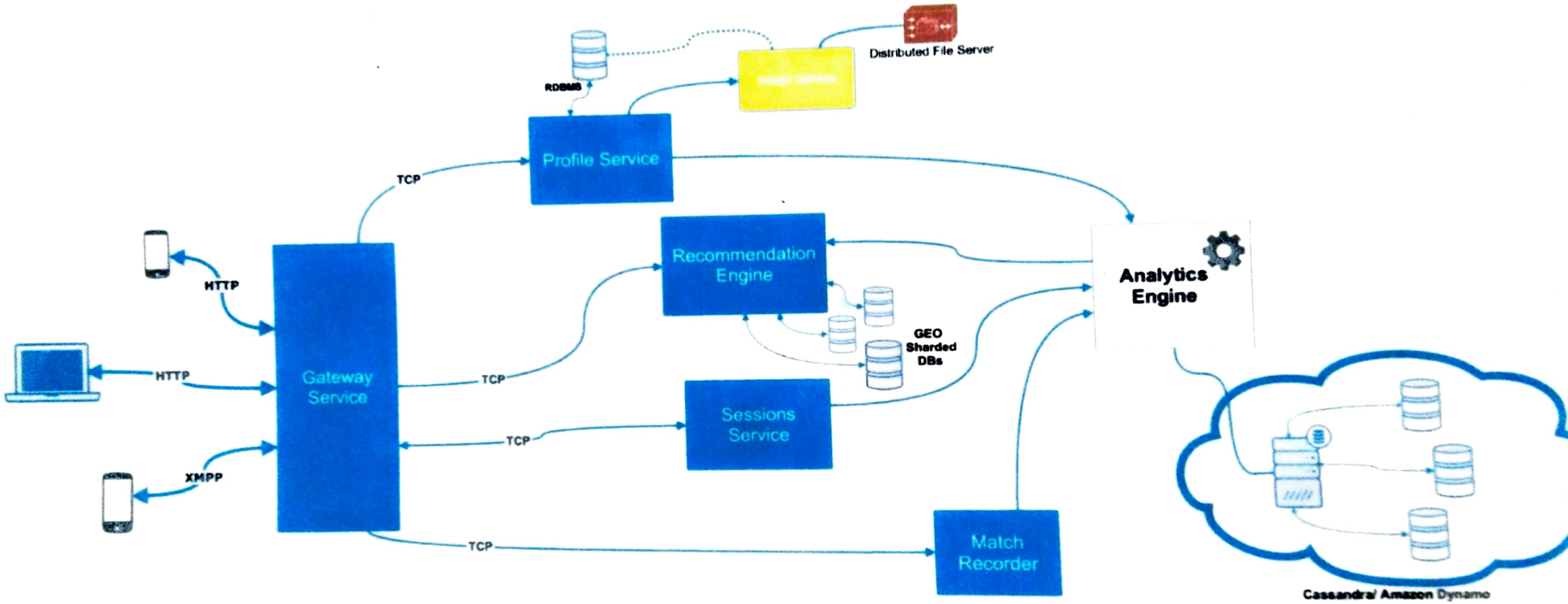

Distributed File Server
RDBMS
Profile Service
TCP
HTTP
HTTP
XMPP
Gateway Service
TCP
TCP
TCP
Recommendation Engine
GEO Sharded DBs
Sessions Service
Match Recorder
Analytics Engine
Cassandra/ Amazon Dynamo

Designing Tinder (as a microservice architecture)

→ Designing front to back i.e. think abt what the users need as features → then think abt how your services are going to be actually broken down → then individual data reqmts. and implementⁿ / 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) Transacⁿ → not reqd. in any part of Tinder

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

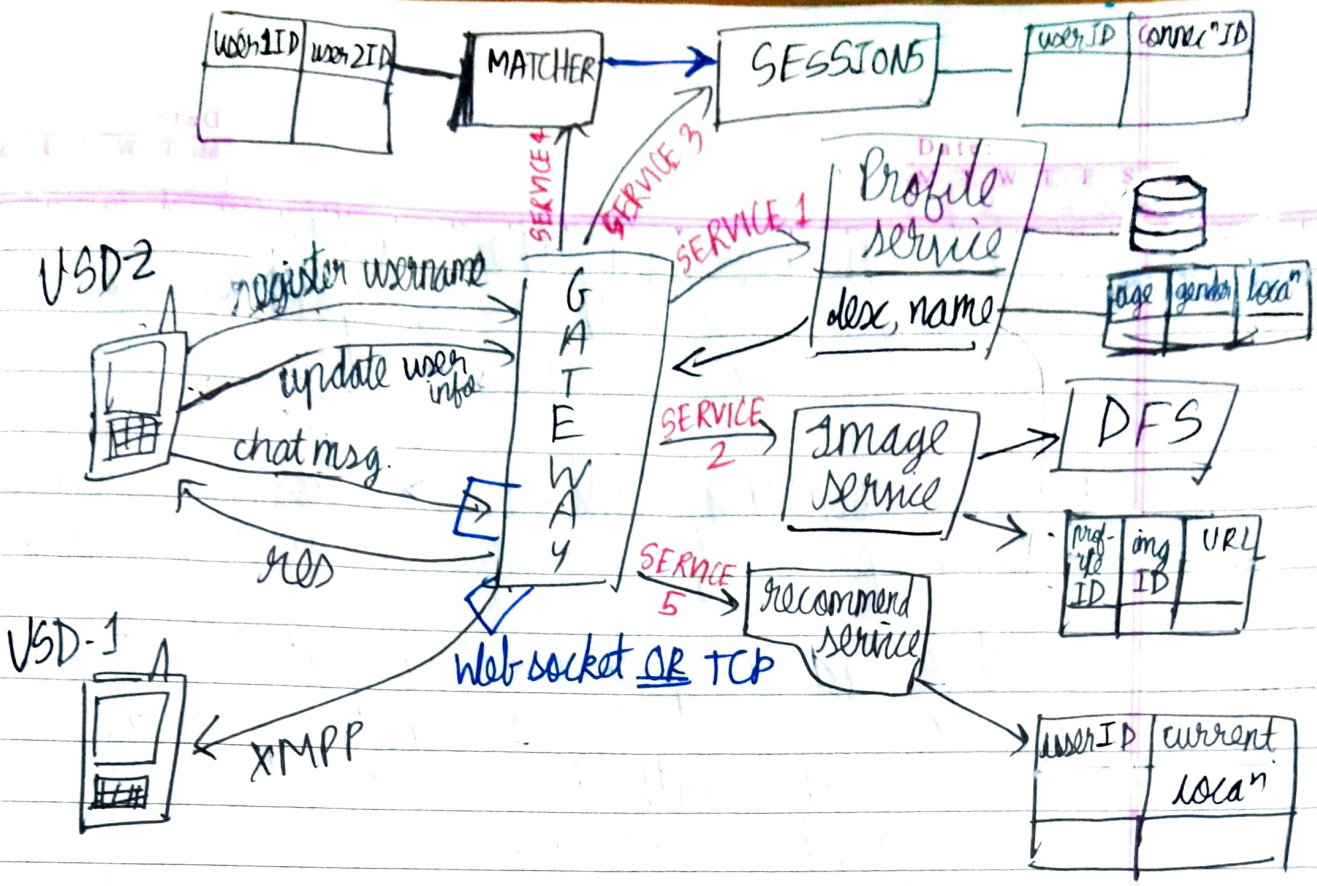
profileID	imageID	file URL

→ 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

~~Key~~ Salient features of this sys.

(1) Client initially registers with username & password ~~Pass~~ This auth. data is stored in "Profile service"
Next 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

client reqs, get them auth-ed by profile service and then gateway routes this auth.ed req. to the reqd. services.

Decoupling Systems → Sys in which every piece of sys does just 1 task.

Another reason → we separated out the protocols reqd. to (1) auth a req. and (2) process the authed req. to get the res. that the client is looking for.
This becomes really useful when these 2 protocols are diff. eg:- when using chat msging we must the XMPP protocol but for auth we use the HTTP protocol

(3) Now, the img service → stored as a separate service cuz it has a unique fn. (Reason:- now more than 1 services can use it eg:- client's req, an ML model to do some sorting of imgs,). Also this area might perform some heavy computations on imgs so it is wise to store it separately.

FEATURE 2 - DIRECT MSGING

PROTOCOL TYPE 1 → We won't use HTTP protocol → it is a client-server comm protocol → ie. the client talks to server for data not the server talks to client to give data the only way these msgs can be recd is via polling the server every fixed interval (say 5 secs) for msgs viz very inefficient

PROTOCOL TYPE 2

∴ We don't wanna poll the server for msgs, we wanna hv msgs pushed to client
→ We use a P2P protocol

↓

everyone is equal

∴ Now, the server and the client receiving msgs are both "peers", so they can easily push data to each other
Thus, here the client & server talk as equals usin the XMPP protocol viz. an eg. of this P2P protocol.

XMPP is gonna take ~~an internal conn~~ a connec" for comm" → can either be a ① web socket, or a ② custom written protocol on TCP

Sessions Service → basically stores data of who's chattin with who i.e. it stores:

① user ID

② connec"ID (the ID user ID is sendin msgs to, recevin msgs from)

FEATURE 3 – Noting recommenda"s

Matcher service → just keeps a track of who's matched with who i.e. it stores

① user1 ID
② user2 ID

Can put idxes over either of the user ID and then just duplicate the records (if A matched with B, B matched with A)

Date:
M T W T F S S

Matcher 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 Matcher service
② people you can text, from the Session 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	locaⁿ

We basically idx this dB on locaⁿ and then ~~retrieved the~~ partiⁿ the data based on this locaⁿ prop. and provide just 1 such partiⁿ to a user.

Date:
M T W T F S S

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

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

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 rules 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".