WhatsApp

High Level Design

Interview Experience

Interviewer's Expectations:

1. Gather both functional and non-functional requirements.
2. Estimate scale and perform accurate back-of-the-envelope calculations.
3. Determine if the system is read-heavy or write-heavy.
4. Assess the storage requirements.
5. Identify the sharding requirements.
6. Evaluate the bandwidth requirements.
7. Provide justification for choosing either SQL or NoSQL databases.
8. Develop a high-level component architecture and provide explanations.
9. List all the APIs.
10. Design a high-level database schema.
11. Create a complete system design with all necessary components.
12. Explain the trade-offs made in the proposed solution.
13. Identify at least one major issue with the proposed solution.

Q1. Design a WhatsApp System

Ans:

5-step approach to High-Level Design (HLD)

1. Problem Statement
2. Requirement Gathering Functional Requirements (MVP)
3. Non-functional Requirements (Design Goals)
4. Scale Estimation
5. System Design

Functional Requirements (MVP)

- Support one-on-one chat
- Group chats (max 100 people)

Extended requirements

- Sent, Delivered, and Read receipts of the messages.
- Show the last seen time of users.
- Support file sharing (image, video, etc.).

Simple but required to mention

- Authentication

Non-Functional Requirements

- High availability with minimal latency (PACELC)
- Highly scalable to handle millions of concurrent users
- Fault tolerance
- Design Goals:
  - Read/Write heavy system
  - Storage requirements
  - Sharding requirements
  - Bandwidth requirements

Estimations and Scale:

Complete Design Goals

Calculate Total Writes (Our Assumptions):

- **250 million total users**
- **50 million daily active users (DAU), which is 20% of the total.**

  We can apply the Pareto principle (80/20 rule).
- **Each user sends 10 messages to 4 different people every day.**
- **Total: 2 billion messages per day.**

  50 million users× 40 (10x4) messages per user = 2 billion messages per day

Estimations Basics

Thousand	**000 (3)**	**KB (Kilobytes)**
Million	**000,000 (6)**	**MB (Megabytes)**
Billion:	**000,000,000 (9)**	**GB (Gigabytes)**
Trillion:	**000,000,000 ,000 (12)**	**TB (Terabytes)**
Quadrillion:	**000,000,000,000,000 (15)**	**PB (Petabytes)**

Calculation of 2 billion messages using the above table

50M (50 x 10^6) x 40

2000 x 10 ^ 6

2x10^3 x 10 ^ 6 = **2B msg / per day**

Calculate Total Reads (Our assumption)

50 Million daily active users (DAU)

Aprrox 100 messages are read by each user every day

50x 10^6 x 100 = 5 B message reads/daily

Write to Read Ratio = Writes per day / Reads per day

1. If the ratio > 1: The system is likely write-heavy (more writes than reads).
2. If the ratio < 1: The system is likely read-heavy (more reads than writes).
3. If the ratio ≈ 1: The system has a balanced load between reads and writes.

2/5 = 0.4 / System is read heavy

Requests Per Second (RPS) (***)

2B (2 x 10^9)/ msg per day x (86400 sec in 1 day) approx 1 lac 10^5

2x10^9 / 1 x 10^5 = **20,000 req per second**

Storage per day

2B msg / per day

Required Storage for Text messages per day

1 msg approx 100 bytes for

(2 x 10^9) bytes x 100 bytes = **200GB/day**

Required Storage for Media with every message

assumption approx 5% of the message uses media

5% of 2B message (2 x 10^9) x 5 / 100 = **100 M message**

Assumption 100kb size of each media per message

100M (100 x 10^6) x (100 x 10^3)

10,000 x 10^6 x 10^3

10 x 10^3 x 10^6 x 10^3 = 10TB / day

Storage Estimations for 10 years

Approx. **365 days** for 1 year; take it **400** for simple calculation
(400 day/years x 10 years) **4000 days**

4000x 10TB/day = **40000TB -** media space
4000 x 200GB/day = **800000GB -** text space

Covert GB in TB 800000/1000 = **800TB**

Total storage required
40000TB + 800TB = **40800TB / 1000** = **aprox 40PB**

Bandwidth Requirement

Total storage required a day for 2B MSg

Media -> **10TB**

Text -> (200GB / 1000) = **.2TB**

10TB + .2TB = **10.2TB/Day**

10.2 x 10^12 byts / 10^5 (100000 sec) day

10.2 x 10^7 = 10.2 x 10 x 10^6

102MB/second

Justification for Design Goals

- Read-heavy system (more reads than writes) (Caching required)
- High storage requirements (Large File system)
- High bandwidth requirements (CDN crucial for content delivery)
- Sharding is required

Database Schema

users

---

id
name
phoneNumber
status
lastSeenAt

groups

---

id
name (nullable for personal chats)
type ('group' or 'personal')
memberCount

users_groups (N:M)

---

userId
groupId

Messages

---

id
userId
groupId
content
sentAt
deliveredAt
seenAt

SQL vs NoSQL

While our system **appears relational**, we will **avoid** storing everything in a **single database**, as this can limit scalability and become a **bottleneck**. Instead, we will **adopt a hybrid approach**: using a relational database (**SQL**) for **users and groups**, and a non-relational database (**NoSQL**) for **chats and media**. We will also implement **horizontal sharding** based on **group_id** to improve scalability.

API design

Get all chats

getAll(groupID: UUID): Chat[] | Group[]

Get messages

getMessages(groupID: UUID, sessionID: UUID): Message[]

Send message

sendMessage(groupID: UUID, sessionID: UUID, message: Message): boolean

Join or leave a group

joinGroup(groupID: UUID, sessionID : UUID): boolean

leaveGroup(groupID: UUID, sessionID : UUID): boolean

High-level design

Architecture ->

We will use microservices, which help us to horizontal scale

- User Service
- Chat Service
- Notification Service
- Presence Service
- Media service

How inter-service communicate?

We will use REST or HTTP for internal communication

How Real-time messaging works

How do we efficiently send and receive messages? We have two different options:

Pull model - The client can periodically send HTTP requests to server to check for new messages, something like Long polling. unnecessary request overhead

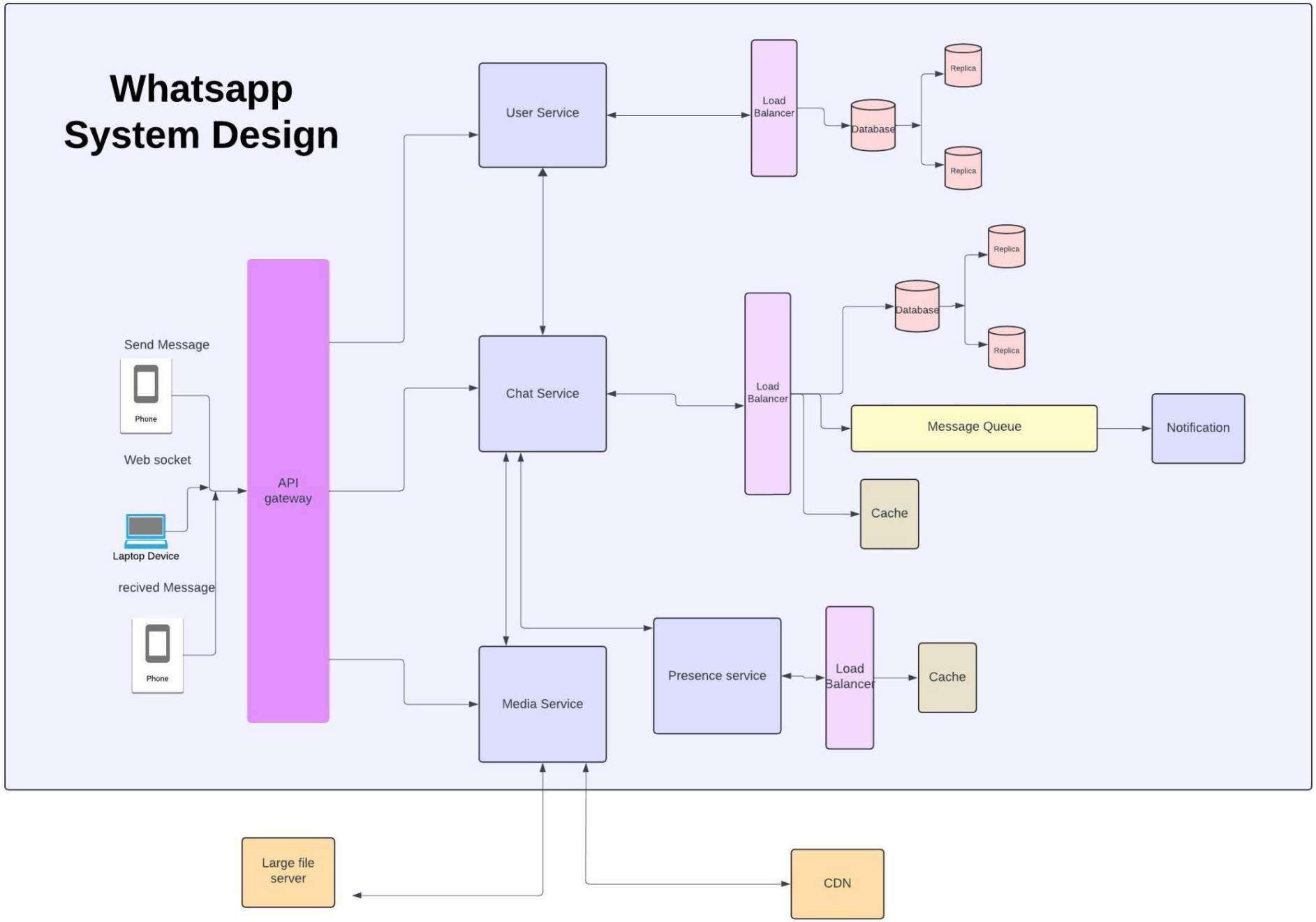
Push model - The client opens a long-lived connection with the server and once new data is available, it will be pushed to the client. We can use WebSockets

Last seen feature

To implement the **last seen** functionality, we can use a **heartbeat** mechanism, where the client can **periodically ping the servers** indicating its liveness and **save last active timestamp** in **cache**

Read feature

Once a **message is sent** in a **chat or a group** we will first **check is recipient active** or not if the recipent is not active the **chat service** will **add an topic to message queue** with additional **metadta** such as the **client's device** the **notication service** then **consume the message** from queue.


Whatsapp
System Design
User Service
Load Balancer
Database
Replica
Replica
Send Message
Phone
Web socket
Laptop Device
recived Message
Phone
API gateway
Chat Service
Load Balancer
Database
Replica
Replica
Message Queue
Notification
Cache
Presence service
Load Balancer
Cache
Media Service
Large file server
CDN

Feedback given by Saran Balaji from Microsoft:

Requirement gathering: Good. Came up with most of the requirements. Scale estimation: Good but need to estimate QPS as well. APIs: Need to come up with extensive list of APIs. Design: Good understanding of components. Need to mention about Async processing using kafka.

Final Result -> Successfully Cleared

How do I prepare for my interview:

I spent 4 days reviewing **my notes on HLD 4** to refresh my knowledge.
This practice helped me answer questions better.

I also practiced

- Desney-Hotstar System Design
- Job Application System Design
- Type Ahead System Design
- Uber System Design

@GeekySanjay

Thank You

==========

Contact Me

If you have any questions or suggestions regarding the video,
please feel free to reach out to me
on WhatsApp: **Sanjay Yadav Phone: 8310206130**
https://www.linkedin.com/in/yadav-sanjay | https://www.youtube.com/@GeekySanjay

==========