Google drawings

Shared space for me and interviewer

Step 1: Requirements clarifications:

1. What will users be able to do?
   1. Search function?
   2. Post stuff?
2. What will users be able to see?
   1. Some specifics on this, is it large amounts of data being displayed (photos, videos) or just text?
3. Focus on backend, frontend, or both?
4. What do we need to do?
   1. Push notifications?
   2. Authentication?
   3. Save/store data?
   4. Provide service to local or global scale?

Step 2: Back of the envelope estimation – estimate the scale of the designed system

1. What scale is expected? Volume of data moving back and forth?
2. How much storage?
3. What kind of storage? (block, object, etc?)
4. Network bandwidth, load balance requirements, traffic management?

Step 3: system interface definition

1. Define what APIs are expected from the system
   1. Establish contract expected from the system and ensure we understand requirements
2. Example API
   1. postTweet(userID, tweetData, tweetLocation, userLocation, timeStamp)
   2. generateTimeline(userID, currentTime, userLocation)
   3. markTweetFavorite(userID, tweetID, timeStamp)
3. Other simple example
   1. Message(userID, messageText)

Step 4: Defining data model

1. Defining the data model in the early part of the interview will clarify how data will flow between different system components. Later, it will guide for data partitioning and management.
2. Identify various system entities
   1. How they will interact with each other
   2. Different aspects of data management like
      1. Storage
      2. Transportation
      3. Encryption
   3. Example entities
      1. User: userID, Name, Email, DoB, creationDate, LastLogin, etc.
      2. Tweet: tweetID, content, tweetLocation, numberOfLikes, timestamp, etc.
      3. UserFollow: userID1, userID2, etc.
      4. FavoriteTweets: userID, tweetID, timestamp
3. Which data system should we use?
   1. NoSQL
   2. MySQL
   3. What kind of block storage to use for photos/videos?

Step 5: High-level design

1. Draw a block diagram with 5-6 boxes representing the core components of our system.
2. Identify enough components that are needed to solve the actual problem from end to end.
3. Some questions to consider
   1. How many application servers are needed?
   2. How many databases?
   3. What kind of file storage?

Step 6: Detailed design

1. Dig deeper into 2 or 3 of the major components
   1. Interviewers feedback should guide us to what parts of the system need further discussion
2. Present different approaches, pros and cons, and explain why we will prefer one approach over the other.
3. Proactively consider trade-offs and system constraints, voice them quickly
4. Some questions to consider:
   1. How should we partition our data to distribute is to multiple databases?
   2. Should we try to store all data of a user to a single database?
      1. What issues could occur? Tradeoffs?
   3. How will we handle “big” users?
   4. How efficient does storage need to be? What data needs to be displayed quicker than others?
   5. How much and at what layer should we introduce cache to speed things up?
   6. What components need better load balancing?

Step 7: Identifying and resolving bottlenecks

1. Try to discuss as many bottlenecks as possible and different approaches to mitigate them.
2. Questions to consider
   1. Is there any single point of failure in the system?
      1. What are we doing to mitigate it?
   2. Do we have enough replicas of the data so that we can still serve our users if we lose a few servers?
   3. Do we have enough copies of different services running such that a few failures will not cause a total system shutdown?
   4. How are we monitoring the performance of our service?
      1. Do we get alerts whenever critical components fail or their performance degrades?

**Example:**

Design a read-only team email inbox that displays the received time, subject, recipient, and email content for emails sent to all team members. If an email is sent to multiple people on a team it should show up as a single row in the inbox with both recipients. You have access to the Gmail api that takes in a user credential and time range and returns a list of emails. The system should be fault tolerant, and should never have data loss, data delay of up to 30 mins is fine though.

**Step 1: requirements verification**

Can anybody send an email? Do you need special access to send an email?

Who can read the emails? Can someone from outside the team read from this inbox? i.e. SA, Higher level mgr?

When you say “recipient,” do you mean only those people can open the email? Can anybody open and read even if it’s not addressed to them?

Can everybody read the subject line?

Who is sending the emails? Customers/outside entities? Or is this for in-house usage?

**Step 2: Scale Estimation**

How many emails will be sent per day (assuming a 24 hour day for a fully remote team spread across the world)?

How many times per day will the inbox be accessed?

How long do we need to store emails?

System should never have data loss, that means lots of backups

**Step 3: System APIs**

Gmail api

Email {

userID: str,

timeRange: tuple(dateTime, dateTime)

} Response {

Emails: list[emails]

} status 200 ok