TECH UNDERSTANDING CASES

INTRODUCTION

Good job on making it this far. You've reached the final case interview round, the tech case. It's completely alright if you're from a non – computer science background, these cases *are not meant to test your coding skills*. Instead, they test a more fundamental skill, a skill that's prevalent amongst all engineers – curiosity. Again, we repeat. These cases will NOT test your coding skills. They will test your ability to figure out how various products and systems work. That's a skill that all engineers are ingrained with, regardless of the field of engineering they pursued.

Tech cases (also called system design cases) dive into how products are built. For example, how is WhatsApp designed? How was Google Maps made? These questions may seem really intimidating at first, and that's completely alright. None of these products were built in an hour, and hence the interviewer does not expect you to be perfect or get into the details of it. Instead, they are just looking for your ability to break down big hairy problems into small manageable chunks, which can then be built by engineers.

Things to keep in mind for tech cases:

1. The questions are broad. Make sure you ask a lot of clarifying questions and define the scope of the case. Remember, great products weren't built in a day and you aren't expected to do that either.
2. Ask questions in the beginning. Figure out what features you want to incorporate into your product, and what you should leave out. Don't make assumptions about what the product should do.
3. In an ideal world, features operate in the same conditions all the time. In the real world however, the user may be in all sorts of

environments while using the product (For example, the user may have bad data connectivity while using an app.). Hence, keep in mind the general conditions (a good, stable internet connection) as well as outliers (sudden loss of network), while designing features.

4. Just like other cases, tech interviews should be conversations. Actively engage with your interviewer and ask them if you're on the right track. Your interviewer will guide you through the case
5. You improve at system design questions by knowing more about the world. Be curious and find out more about the tech you use every day.

To help you learn more about the various products you use every day, we've added a few Tech Titbits after the cases, which briefly explain how certain products work. Make sure to further research the topics you like, as this will help improve your preparation by leaps and bounds!

CASE 1: HOW DOES WHATSAPP WORK?

A simple dive into one of the most useful creations of our time.

CASE STATEMENT

Design a messaging service like WhatsApp.

PRELIMINARY QUESTIONS

What features are we looking at?

The messaging app should have the following features:

1. One-to-One text: Users should be able to send text messages to each other.
2. Sent/Received/Read:
    a. The message should show sent (single tick) when it has been sent from the user's phone.
    b. The message should show delivered (double tick) when it has been received by the recipient's phone.
    c. The message should show read (blue double tick) when it has been seen by the recipient.
3. Last Seen: The app should show when a particular user last used the messaging app.

How many users do we have? What scale are we looking at?

The app should support around 450 million daily active users, sending approximately 2 billion texts every day. However, for the purpose of this case, we shall not go deep into scaling.

Should I look at app design as well?

Let's only look at the backend architecture, and not concern ourselves with app design.

Does the messaging service support media files?

No, let us assume that it only supports text.

Does the service support group chats?

Let us assume that it shall only support person-to-person messaging, with no group chats.

Should the service have a web application as well?

Let us assume that we are restricting ourselves only to a mobile phone app.

Does the app use the internet or send SMSs using cellular networks?

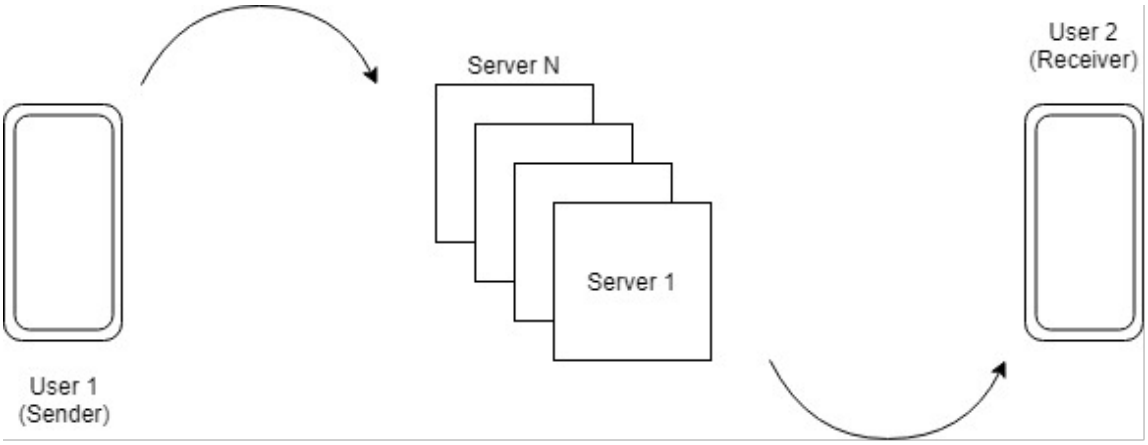
Let us assume it's an internet-based messaging app.

CASE FLOW

For the sake of this case, let's assume there are 2 people – the sender of the text message S, and the receiver of the message R.

Stage 1

Feature 1: One to one text:



1. Asynchronous texting: The candidate should realize that users may not be online at the same time. Messages might be sent from a user when the recipient is not online. Similarly, when the recipient is online, the sender of the message may not be online.
2. Hence, the candidate should infer that one would need an intermediary computer (a server) to store messages when the receiver is not online. *Once the candidate has understood this, the interviewer*

should specify that the server need not store the messages, once the recipient has received them.

3. Order of messaging: The recipient should receive messages in the same order as it was sent by the sender. Hence, the server would need to send the messages in the same order. The server can create a queue in its memory for every user, with messages that were sent to that user first, being delivered to the user first. This queue follows the First In, First Out principle.
4. Scaling: While we won't go deep into scaling in this case, the interviewer can give it a passing mention. The candidate should realize 2 things:
    a. A single server will not be enough to handle all the traffic.
    b. Instead of creating one server with substantial computational power, it makes more sense to have multiple servers. This way, in case one server shuts down by accident, the app can still run.
    c. Ensuring messages reach the recipient in order becomes more complicated if there are multiple servers. However, once the candidate has realized this, the interviewer need not go into further detail about ensuring they arrive in the right order.

Stage 2

Feature 2: Sent/Received/Read

1. Sent: The server should reply with an acknowledgment on receiving the message from the sender. Upon receiving the acknowledgement, the app can show a single tick for the message.
2. Received: The sender can receive the double tick icon only once the receiver has confirmed that it has received the message. Therefore, an acknowledgment must come from the receiver whenever it

comes online, and this has to be sent to the sender when the sender comes online. Thus, the Double Tick can be thought of as a special kind of message. Every time the recipient receives a message, the "Double Tick special message" gets sent to the server. The server adds this special message to the sender's message queue. When the sender comes online, it receives the message, and the app shows it as a double tick.

3. Read: The read feature can also be thought of as a type of special message sent from the recipient's phone every time they open a particular chat. It would follow the same procedure as the received message.

Stage 3

Feature 3: Last Seen

The last seen feature displays the time at which each contact of a user last used the app.

1. User side: Each user continuously sends a "Last Seen" message to the server, letting it know that they are online. The message can be a single bit letting the server know that the user is online. This message can be sent at a fixed interval (like 5 times every minute). (These messages are called heartbeat messages as they're sent at regular intervals and let the server know that the user is "alive" aka has a heartbeat.)

   Note: It is not sufficient for the user's app to only send the time at which the user came online and the time at which it went offline, as the user might go offline abruptly. If that happens, the user would not be able to communicate to the server that it's gone offline.
2. Server-side: The server has a table with each user and their "Last Seen" timestamp. As the server receives the "Last Seen" message

from the user, it continuously updates the table with the time at which it received the Last Seen message.

3. Displaying the Last Seen time: Each user's app sends to the server a list of users for which it needs to see the Last Seen time. The server accordingly retrieves the information and sends it to the user's app, allowing it to display the Last Seen time in an appropriate format.
4. Once the candidate has solved for these 3 features, she/he may summarize and close the case.

CASE 2: GOOGLE MAPS

Navigate your way through the technical underbelly of Google Maps.

CASE STATEMENT

Design an app-based maps service, like Google Maps.

PRELIMINARY QUESTIONS

What features of Maps should the app support?

The app should support the following features:

1. A user should be able to find their location on the map.
2. Given a source and a destination address, the app should be able to find the fastest route from source to destination (for a car).

How detailed should the map be? Are we trying to be as detailed as Google Maps?

Keep the map as detailed as you can. This will however, depend on your data collection methods, which we shall discuss in the case.

CASE FLOW

This case involves a lot of assumptions, to simplify the case solving process. The interviewer should encourage the candidate to make these assumptions, in case the candidate gets stuck or needs some clarifications to proceed with the case.

Stage 1

Feature 1: Finding a user's position on a map.

This process involves 2 things:

- Building a map.
- Calculating and showing a user's position on the map, based on their GPS coordinates.

Let's look at these individually:

Building a Map

1. The candidate should look at 2 things here:
    a. Sources of data for making the map.
    b. Representation/storage of the map in the database.
2. The candidate should think of and propose sources of data for building a map. Here are some avenues Google uses to collect map data:
    a. Civic authorities
    b. Satellite data
    c. Crowd-sourced data:
        i. Users can pin locations on the map and label them. This is especially useful to keep the map updated.
        ii. Using GPS coordinates. By knowing the GPS coordinates of people using maps, Google can better understand which roads are one-ways, which roads have signals, etc.
        i. Cars with cameras: Google drives cars around with cameras on them. This helps them collect data for street view as well as collect map data.
3. Storing the map: The map can be represented as a graph, with buildings, lakes, intersections and other spots as nodes of the graph. Roads form the edges (the connecting lines) of the graph.

Locating a user on the map

1. The candidate should realize that the entire map cannot be loaded onto a user's phone.
2. The map should store data about each building and road on its database, such as the start and end GPS coordinates, etc.
1. Based on the GPS coordinates of the user, a certain section of the map (the candidate can assume say 2km x 3km) around the user is

retrieved from the database and loaded on the phone. Then, the GPS coordinate of the user is pinned on this map.

Stage 2

Feature 2: Finding the shortest route between a start point and an end point on the map.

1. The first thing to realize here is that the entire map cannot be searched for an optimal route, as this process is unnecessary and would simply take too long.
1. The candidate can discuss various ways to choose a section of the map for searching with the interviewer. It is believed (but not confirmed) that Google draws a line between the start and end point, and then chooses all roads within a pre-specified distance of that line. This map is then searched for an optimal route.
2. Each road (graph edge) is assigned a weight, based on how fast one can travel on that road.
3. These weights are assigned in the following ways:
    a. Based on the speed limit of various roads
    b. By gathering traffic data. This is crowd-sourced.
    a. Historical data (gathered by analyzing day-to-day data)
4. For the sake of this case, let us assume that there are no one-way roads, and that additional issues like traffic lights are accounted for in the weight assigned for each road.
2. Once you have two nodes on the graph, Djikstra's algorithm can be used to find a path between them. *Note that the candidate need not know how Djikstra's algorithm works.* It is recommended that readers look up a brief explanation of the algorithm later, if interested. (The actual algorithm used by Google is believed to be a variation of Djikstra's called the A Star algorithm.) The candidate can assume

that there exists an algorithm for finding the shortest path, and proceed with the case.

3. The interviewer should mention that one problem with Djikstra's algorithm is that the time it takes to give you the shortest path increases greatly as the size of the map increases. Hence, the interviewer should guide the candidate to come up with a different approach, in case the start and end points are far from each other. (For example, if Maps is used to go from one city to another.) Given below are the approaches for calculating fastest path, based on how far the source and destination are:
    a. Short distances: Use Djikstra's algorithm to find the optimal path.
    b. Long distances: Maps first sends you to the closest major road/highway. Roads are categorized into different types (highway, major road, small alleys, etc.). The candidate can suggest that only major roads are considered by Maps for finding the shortest path between 2 points close to source and destination. Finally, to get to these major roads (near the source and the destination), all types of roads can be considered.

After understanding all this, the candidate should summarize the entire system that they designed.

CASE 3: HOW DO YOU TWEET?

Tweety bird turned from a yellow canary to a corporate icon

CASE STATEMENT

Design a social media application like Twitter

PRELIMINARY QUESTIONS

What is my application supposed to do?

We need to do the following:

- Allow users to create and maintain an account.
- Each account should be able to connect with other accounts they are interested in seeing, using a follow feature.
- Each person should be able to create and post short 140-character messages for everyone to view.
- There should be a search feature which allows people to find a particular tweet.

Should I look at app design as well?

Let's only look at the backend architecture and not concern ourselves with app design.

How many users are we looking at catering to?

Let's not worry about scale, for the purpose of this case.

CASE FLOW

Account Creation

Each user is given a unique ID which is linked to their profile details such as name, age, email ID, phone number, username, password, etc.

This account must be stored on the device/server and should be updated on the server when changed.

Connecting accounts

Each account is now identifiable by a unique ID. The next step is to link different IDs. This can be done by building a graph. A subtlety to note is A-B connection is not the same as B-A connection, as a person you follow may not be following you. With this, you should build a network for every user. Each user's account has a list of accounts it is linked to.

Tweeting

1. The candidate should infer that one would need an intermediary computer (a server) to store tweets, so that others can view this.
2. The tweet must be limited in size on input and rejected if greater than 140 characters.
3. Each user ID should be linked to her/his tweets.
4. Each tweet should have a privacy tag (Which defines visibility based on the connection network we have made).
5. Any user who fulfils the privacy requirement should be able to access the tweet.
6. A priority list must be made amongst the available tweets so that they show up in a list. This priority can be chronological or otherwise.

Searching for Tweets

1. The candidate must realize that searching all tweets for the search phrase is an inefficient process.
2. The candidate must come up with a search strategy. This can be done in different ways.

Note: For this method it is important to talk about how the data would be sorted in order to search through the database of tweets.

TECH TITBITS

Bite sized tech knowledge for all!

TITBIT 1- HOW THE INTERNET WORKS

What happens when you type in a website name on your browser? This tit-bit will focus on the process that happens behind the scenes, whenever you click on a website link.

Just like how every house in the world has an address, every computer has an address too. This address is called an IP address, and is generally a number (ex. 172.16.254.1). The IP address helps to uniquely identify every computer on the internet. Your Internet Service Provider (Jio, Vodafone etc.) assigns you your IP address.

When you click on a website, the information required to load that website is fetched from a server. Just like any other computer, servers have IP addresses too. How does your computer know which server to fetch the information from? That's where DNS comes in.

A DNS (Domain Name System) is like a huge phone book, it has a list of websites and the IP addresses of the servers where the website information is stored. When you enter a domain name (Eg. www.google.com) on your browser, your browser sends a request to the DNS server to find its corresponding IP address (The IP address of google.com is 172.217.163.110.). After it gets the IP address, the browser simply forwards the data request to the IP address of the server, which receives it and sends back the information. Information is transmitted in the form of packets, and often travels thousands of kilometres through underwater giant optical cables, before it reaches you!

Further Reading Topics: IPv4, IPv4 Address Exhaustion, IPv6, ISO Network Layers.

TITBIT 2 - PASSWORD PROTECTION AND HASHING

Have you ever wondered how companies store your password? What does encryption mean, and how do companies stop hackers from stealing your password? They use a process called hashing, which we will now examine.

A hash function can take any input (regardless of its length), and convert it into a fixed length text output. For example, if we use the SHA3 hash function, the output is 256 bits long, regardless of what we input.

Hash("mypassword")=42a00eaa9468b08adc80d9dfb877a6d30c3ed64eea293e945228669ee3de4f1e

The magical thing about these hash functions is the fact that they are one-way functions ie. It's easy to compute the output of a hash function given an input (there's a formula for it), but it is almost impossible to figure out what the input to the function was, if we have only the output. The output is completely different for very similar inputs as well! For example, if we change our input by just one characters, we get:

Hash("myp@ssword")=89c8571ca21760aef65789105b49f7df90b9c05fb7d1013b6e437d84c529175e

We see that the output for "mypassword" and "myp@ssword" are completely different! This is one of the reasons why it's almost impossible to find an inverse for the hash function. And it's this property that makes the function incredibly valuable.

Tech companies never store your password as it is. They always hash your password, and store the hashed version of it. Now, when you type in the password in your browser, the text you type in is immediately hashed, and then compared with the hash stored in the database. If the hashes match, that means you know the password! The ingenious thing about this is that even if the hackers hack into the database and steal all the data, they only have access

to the hashed versions of the passwords. And hence it's impossible to figure out the original password from the hash, they still don't know your password!

Of course, hackers can try to guess your password, create its hash, and compare it with the hashes they've stolen to see if they match. However, this process is incredibly computationally expensive, and would probably take them years to figure out! Hackers however generally have a pre-computed list of commonly used passwords and their hashes, and hence it's always recommended to use a password that isn't common (therefore if your password is "password" or "qwerty", go change it immediately!).

Further reading topics: Rainbow Tables, Salt, Pepper.

TITBIT 3 – COMPRESSION (LOSSLESS)

Zip folders are a common occurrence on our laptops. We use them to save space without losing information. How do we achieve that? What allows us to make a file occupy less space? Let's look at how this works!

To understand how this works, take this sentence from Roberto Bolano's book 2666:

> I felt happy because I saw the others were happy and because I knew I should feel happy, but I wasn't really happy.

This quote has 115 characters (including spaces, apostrophes, commas, and periods). As you may notice this sentence has a lot of "I"s and "happy"s . So what does compression do? It says hey let me replace happy by some special character, say '~' and let me have another file that lets my computer know that ~ is a place holder for happy. Currently if I assume one character as one unit of memory, then this sentence has 115 units of memory. Now replacing happy with ~ allows me to reduce the memory consumed by 4 characters for each happy. That's 16 units of memory. Now

I need another file which tells me ~ represents happy. So that would need say at least 6 characters ("~happy"). So totally I have saved 16 - 6 = 10 units of memory. That's an 8% memory reduction!

Let's be a bit smarter though, maybe I shouldn't replace "happy" but instead " happy ". So now I'm actually saving 6 units for each happy. In my new calculation I save 16 (24 – 8 = 16) units of memory. That's a 14% memory reduction .

The crux of compression is to find patterns and map them to a dictionary. The challenge is finding the best patterns for a given file!

Note: This compression is called lossless compression as there is no loss in data. There are other types of compression for images (like the .jpeg format!) where you lose data but retain almost all the required features of the image.

Further Reading Topics: Lossless Image Compression (png), Lossy Image Compression (jpeg)

TITBIT 4 – SEARCHING FOR SOMETHING

Any searching and reporting engine follows 3 steps. (Note: A searching and reporting engine is not limited to Google or Bing. Facebook uses such engines to decide the order in which posts show up on your timeline. Swiggy uses an engine like this to choose the right restaurants to show you on the Swiggy home page. Almost every website/app which shows you information will use an engine like this.) The three steps are:

1. **Crawl**: Crawling is the act of looking through all the information available. This could be the set of all FB posts online, all websites on the internet (yes, Google and Bing regularly do this!), or all the restaurants with Swiggy. How one goes through this data set (mostly by dividing it and parallelizing) is unique to each organization. The

faster you can traverse this space, the better. Organizations do this activity periodically, in order to ensure they are up to date.

2. **Index**: While traversing through the information, each bit of this information needs to be indexed. For example, Google might do this by creating a set of keywords for every webpage it visits, so that the page can be identified. These keywords could be the various words appearing on the website, or in Swiggy's case, the type of cuisine the restaurant offers.
3. **Rank**: After indexing, the next stage is ranking. Ranking is the process of mapping each index to the relevance of your query. In a search engine, the query is what you'd type in, on Swiggy it could be your previous order behaviour. The rank is assigned based on a set of factors (for Google these could be credibility, match with the query, number of links to/from the website and so on.). After ranking, the preferred results are displayed.

And that's how search engines work, in a nutshell. Of course, incredibly powerful search engines like Google do a ton of other stuff as well, but these other things are all done with the intention of improving these 3 stages – crawling, indexing and ranking!

Further Reading Topics: PageRank, Search Engine Optimization.

TITBIT 5 – CLOUD STORAGE

We've all used apps like Google Drive and One Drive. We've also all heard the term 'cloud' being thrown around. What is cloud storage though? Why is it useful? Let's take a trip to the clouds, to find out.

Meet our good friend Jack. Jack just finished making a super important business document. His boss asked him to keep the document somewhere safe so that they could use it in the future. So Jack saved it on his laptop and his pendrive, and took a print out of it. Now, Jack was very paranoid, so he

locked the paper and pendrive in the company safe. When Jack was on his way home, his laptop got drenched in the Chennai rains and refused to switch on after that. Jack was happy that he had a pen drive left in the safe. He took the pendrive out of the safe the next day to make changes to the document. Jack wondered to himself, if there was an easier way to keep his documents secure and safe.

Then Jack found the cloud. The cloud and Jack's pendrive aren't very different entities. They operate on the same idea. It's a place that stores his documents/files/photos in a manner by which he can access them anytime and from anywhere as long as he is connected to the storage device. Generally, this is done using the internet. In some cases (generally corporate organizations) it can be done through intranet as well.

Anyone with Jack's pendrive has access to his files. Similarly, anyone with access to the file in the cloud can access the documents. Since everything is on the internet, file transfers can happen almost instantaneously.

Jack decided that he would give the pendrive to his friend, tell him to keep it safe, and give it back to him whenever he needs it. Then Jack's friend Bob would be responsible for making sure the documents are safe. To ensure this, he may create copies on his computer, another pendrive, etc. Bob is essentially guaranteeing two things: 1) The file access will only be with Jack and whoever Jack allows access to. 2) The file won't get lost or destroyed. These two tasks are what organizations like Google Drive and One Drive do. Of course, the real hard work here is figuring out a way to store files so as to minimise the amount of storage space used, while maintaining ease of access. All these companies have giant physical storage servers where the user's files are stored.

Further reading topics: IaaS, PaaS, SaaS.

TITBIT 6 – GOOGLE ADS AUCTION

To show your ad on Google seems like a smart idea. How does Google decide whether and where to show your ad? Let's take a look into the Google Ads auction system.

Have you noticed the ads which show up on top of the search page when you search for something on Google? A lot of companies are fighting for that top spot, to be the one that appears on the top of your sponsored search results. However, Google has a limit on the number of sponsored results it can show you, and hence can't accommodate all these companies' requests. Hence, every time Google has to show an ad for a user, it holds an automated auction to decide who wins the ad space!

Any advertiser first needs to identify the search keywords she/he wants to bid on. These keywords are the set of words or phrases for which they would like their ad displayed. After this, the advertiser sets the amount she/he would like to place as the bid amount for any keyword/key phrase. When a person makes a Google search, a list of all relevant advertisers is made. Each advertiser is entered with a limit of one keyword bid per advertiser. These advertisers are then ranked, to determine who shows up first. This ranking is decided based on two factors - the bid amount and the quality score. The quality score is a metric meant to establish the relevance and usefulness of your ad to the user. If the advertiser does both these things right, their ad can show up on Google! The important take away is that for a successful google ad, you need to be relevant and useful to your target customer, not just cough up the most cash!

Further reading topics: Google Ads Quality Score, Google AdSense, Google Display Network.