Since the questions are intentionally weakly defined, jumping onto designing the solution immediately without fully understanding is liable to get you in trouble. Spend a few minutes questioning the interviewer to comprehend the full scope of the system. Never assume things that are not explicitly stated. For instance, the “URL shortening service” could be serving just a few thousand users, but each could be sharing millions of URLs. It could also mean to handle millions of clicks on the shortened URLs or just a few dozens. The service may also require providing extensive statistics about each shortened URL (which will increase our data size), or statistics may not be a requirement at all. Therefore, don’t forget to make sure you gather all the requirements as the interviewer will not be listing them for you.  
  
The main difference between design interviews and the rest is that you are not given the full detail of the problem, rather you are required to scale the breadth and depth of a blurred problem. You are supposed to take the details and figure out the issue by asking probing questions. Your questions for clarifying the problem reflects your evaluating ability and competence, which would be an asset to the company.  
  
In design and architecture interviews the problems presented are quite significant. They definitely cannot be solved in 40 minutes’ time implying that the objective is to test the technical depth and diversity the interviewee invokes during the interview. That also speaks strongly for your would be ‘level’ in the company. And your level in the company should come from your analytical ability to sort out the problem besides your ability to work in a team (your behavioral and background side of the interview), and your capacity to perform as a strong technical leader. In a nutshell, the basic idea of hiring at a level is to scale a person’s ability to contribute value to the company’s wants and needs. For that, you must exhibit your strengths by showing reasonable technical breadth.  
  
Try to learn from the existing systems: How have these been designed? Another critical point to be kept in mind is that the interviewer expects that candidate’s analytical ability and questioning on the problem must comparable to his/her experience. If you have a few years of software development experience, you are expected to have certain knowledge and should avoid divulging into asking basic questions that might have been appropriate coming from a fresh graduate. For that, you should prepare sufficiently ahead of time. Try to go through real projects and practices well in advance of the interview as most questions are based on real-life products, issues, and challenges.  
  
Leading the conversation: It is not the ultimate solution to the problem, instead the discussion process itself that is important in the interview. And it is the candidate who should lead the conversation going both broad and deep into the components of the problem. Hence, take the interviewer along with you during the course of solving the problem by communicating with him/her step by step as you move along  
  
Solving by breaking down: Design questions at first might look complicated and intimidating. But whatever the complexity level of the problem, a top-down and modularization approach can help a lot in solving the problem. Therefore, you should break the problem into modules and then tackle each of them independently. Subsequently, each component can be explained as a sub-problem by reducing it to the level of a known algorithm. This strategy will not only make the design much clearer to you and your interviewer but make evaluation much easier for the interviewer. However, while doing so, keep this thing in mind that mostly the problems presented in high skill design interviews don’t have the solutions. The most important thing is the way how you make progress tackling the problem and the strategies you adopt.  
  
Dealing with the bottlenecks: Working on the solution, you might confront some bottlenecks. This is very normal. While resolving bottlenecks, your system might require a load balancer with many machines behind it to handle the user requests or the data might be so huge that you need to distribute your database on multiple servers. It might also be possible that the interviewer wants to take the interview in a particular direction. If that is the case, you are supposed to move in that direction and should go deep leaving everything else aside. If you feel stuck somewhere, you can ask for a hint so that you may keep going. Keep in mind that each solution is a kind of trade-off; hence, changing something may worsen something else. Here, the important thing is your ability to talk about these trade-offs and to measure their impact on the system keeping all the constraints and use cases in mind. After finishing your high-level design and making sure that the interviewer is ok with it, you can go for making it more detailed. Usually, that means making your system scale.  
  
Summary  
Solving system design questions could be broken down into three steps:  
  
Scoping the problem: Don’t make assumptions; Ask clarifying questions to understand the constraints and use cases.  
Sketching up an abstract design Illustrating the building blocks of the system and the relationships between them.  
Identifying and addressing the bottlenecks by using the fundamental principles of scalable system design.  
Conclusion  
Design interviews are formidable, open-ended problems that cannot be solved in the allotted time. Therefore, you should try to understand what your interviewer intends to focus on and spend sufficient time on it. Be well aware of the fact that the discussion on system design problem could go in different directions depending on the preferences of the interviewer. The interviewers might be unwilling to see how you create a high-level architecture covering all aspects of the system or they could be interested in looking for specific areas and diving deep into them. This means that you must deal with the situation strategically as there are chances of even the good candidates failing the interview, not because they don’t have the knowledge, but because they lack the ability to focus on the right things while discussing the problem.  
  
If you have no idea how to solve these kinds of problems, you can familiarize yourself with the typical patterns of system design by reading diversely from the blogs, watching videos of tech talks from conferences. It is also advisable to arrange discussions and even mock interviews with experienced engineers at big tech companies.  
  
Remember there is no ONE right answer to the question because any system can be built in different ways. The only thing that is going to be looked into is your ability to rationalize ideas and inputs.

**System Design Interviews: A step by step guide**

Lots of people struggle with system design interviews (SDIs) primarily because of 1) Unstructured nature of SDIs, where you’re asked to work on an open-ended design problem that doesn’t have a standard answer, 2) Your lack of experience in developing large scale systems and, 3) You did not spend enough time to prepare for SDIs.

Just like coding interviews, candidates who haven’t put a conscious effort to prepare for SDIs, mostly perform poorly. This gets aggravated when you’re interviewing at the top companies like Google, Facebook, Uber, etc. In these companies, if a candidate doesn’t perform above average, they have a limited chance to get an offer. On the other hand, a good performance always results in a better offer (higher position and salary), since it reflects upon your ability to handle large complex systems - a skill that all such companies require.

In this course, we’ll follow a step by step approach to solve multiple design problems. Here are those seven steps:

Step 1: Requirements clarifications

Always ask questions to find the exact scope of the problem you’re solving. Design questions are mostly open-ended, and they don’t have ONE correct answer, that’s why clarifying ambiguities early in the interview becomes critical. Candidates who spend enough time to define the end goals of the system, always have a better chance to be successful in the interview. Also, since you only have 35-40 minutes to design a (supposedly) large system, you should clarify what parts of the system you would be focusing on.

Under each step, we’ll try to give examples of different design considerations for developing a Twitter-like service.

Here are some questions for designing Twitter that should be answered before moving on to the next steps:

* Will users of our service be able to post tweets and follow other people?
* Should we also design to create and display user’s timeline?
* Will tweets contain photos and videos?
* Are we focusing on backend only or are we developing front-end too?
* Will users be able to search tweets?
* Do we need to display hot trending topics?
* Would there be any push notification for new (or important) tweets?

All such question will determine how our end design will look like.

Step 2: System interface definition

Define what APIs are expected from the system. This would not only establish the exact contract expected from the system but would also ensure if you haven’t gotten any requirements wrong. Some examples for our Twitter-like service would be:

postTweet(user\_id, tweet\_data, tweet\_location, user\_location, timestamp, …)

generateTimeline(user\_id, current\_time, user\_location, …)

markTweetFavorite(user\_id, tweet\_id, timestamp, …)

Step 3: Back-of-the-envelope estimation

It’s always a good idea to estimate the scale of the system you’re going to design. This would also help later when you’ll be focusing on scaling, partitioning, load balancing and caching.

* What scale is expected from the system (e.g., number of new tweets, number of tweet views, how many timeline generations per sec., etc.)?
* How much storage will we need? We will have different numbers if users can have photos and videos in their tweets.
* What is the network bandwidth usage we expect? This would be crucial in deciding how would we manage traffic and balance load between servers.

Step 4: Defining data model

Defining the data model early will clarify how data will flow among different components of the system. Later, it will guide towards data partitioning and management. Candidate should be able to identify various entities of the system, how they will interact with each other and different aspect of data management like storage, transportation, encryption, etc. Here are some entities for our Twitter-like service:

**User:** UserID, Name, Email, DoB, CreationData, LastLogin, etc.

**Tweet:** TweetID, Content, TweetLocation, NumberOfLikes, TimeStamp, etc.

**UserFollowos:** UserdID1, UserID2

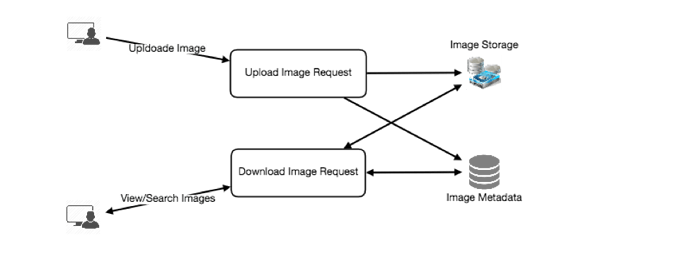
**FavoriteTweets:** UserID, TweetID, TimeStamp

Which database system should we use? Would NoSQL like [Cassandra 32](https://en.wikipedia.org/wiki/Apache_Cassandra) best fits our needs, or we should use a MySQL-like solution. What kind of block storage should we use to store photos and videos?

Step 5: High-level design

Draw a block diagram with 5-6 boxes representing the core components of your system. You should identify enough components that are needed to solve the actual problem from end-to-end.

For Twitter, at a high level, we would need multiple application servers to serve all the read/write requests with load balancers in front of them for traffic distributions. If we’re assuming that we’ll have a lot more read traffic (as compared to write), we can decide to have separate servers for handling these scenarios. On the backend, we need an efficient database that can store all the tweets and can support a huge number of reads. We would also need a distributed file storage system for storing photos and videos.



**[5759778777202688.png975×363 41.9 KB](https://coursehunters.online/uploads/default/original/1X/9f05aba6ce6f0c19ca07a89487f0cd8aa2fe3d65.png)**

Step 6: Detailed design

Dig deeper into 2-3 components; interviewers feedback should always guide you towards which parts of the system she wants you to explain further. You should be able to provide different approaches, their pros and cons, and why would you choose one? Remember there is no single answer, the only important thing is to consider tradeoffs between different options while keeping system constraints in mind.

* Since we will be storing a massive amount of data, how should we partition our data to distribute it to multiple databases? Should we try to store all the data of a user on the same database? What issue can it cause?
* How would we handle hot users, who tweet a lot or follow lots of people?
* Since user’s timeline will contain most recent (and relevant) tweets, should we try to store our data in such a way that is optimized to scan latest tweets?
* How much and at which layer should we introduce cache to speed things up?
* What components need better load balancing?

Step 7: Identifying and resolving bottlenecks

Try to discuss as many bottlenecks as possible and different approaches to mitigate them.

* Is there any single point of failure in our system? What are we doing to mitigate it?
* Do we’ve enough replicas of the data so that if we lose a few servers, we can still serve our users?
* Similarly, do we’ve enough copies of different services running, such that a few failures will not cause total system shutdown?
* How are we monitoring the performance of our service? Do we get alerts whenever critical components fail or their performance degrade?

In summary, preparation and being organized during the interview are the keys to be successful in system design interviews.