# **A Step-by-Step Approach to Acing your System Design Interview**

While every system design interview is different, there are some common steps you should cover, even if the conversation might not be as sequential as your ideal thought process.

The system design interview is an open-ended conversation, which you’ll be expected to lead. Try using the following steps to guide your discussion:

## **Step 1 — Understand the Goals**

Clarifying ambiguities early in the interview is critical. Candidates who spend time clearly defining the end goals of the system have a better chance of success than those who don’t. For that reason, make sure you understand the basic requirements and ask clarification questions. Start with the most basic assumptions:

* What is the goal of the system?
* Who are the users of the system? What do they need it for? How are they going to use it?
* What are the inputs and outputs of the system?

Even if you’re asked about a well-known product, you should still share your assumptions about it with your interviewer. You may find that you and your interviewer don’t have the same assumptions about products like Twitter, Facebook, or Reddit. In addition to helping you focus, it also demonstrates product sensibility and good teamwork.

## **Step 2 — Establish the Scope**

Now that you understand the system, try to describe the feature set that you’ll be talking about. Try to define all the features that you think of by their importance to the user. You don’t have to get it right on your first attempt, but make sure that you and your interviewer agree.

Ask clarifying questions, such as:

* Do we want to discuss the end-to-end experience or just the API?
* What clients do we want to support (mobile, web, etc)?
* Do we require authentication? Analytics? Integrating with existing systems?

Take a few minutes to discuss this with your interviewer, and write it down.

## **Step 3 — Design for the Right Scale**

The same feature set requires a very different approach for different scales. It’s important to determine the scale so that you know whether your data can fit on one machine or whether you need to scale the reads. You might ask:

* What is the expected read-to-write ratio?
* How many concurrent requests should we expect?
* What’s the average expected response time?
* What’s the limit of the data we allow users to provide?

Different answers require very different designs, so getting the scale right is key to success.

## **Step 4 — Start High-Level, then Drill-Down**

Start with covering the end-to-end process, based on the goals you’ve established. This might include detailing different clients, APIs, backend services, offline processes, network architecture, data stores, and how they all come together to meet the requirements.

This is also a good point to identify the system’s entry-points, such as:

* User interaction
* External API calls
* Offline processes

This will allow the conversation to drill down into potential performance bottlenecks, and decisions about the separation of responsibilities. Whichever approach you choose to start with, remember to always **start simple, and iterate.**

## **Step 5 — Data Structures and Algorithms (DS&A)**

Wait, this again? Yes! Turns out, this is actually important in designing software systems.

URL shortener? Makes me think of a hashing function. Oh, you need it to scale? Sharding might help. Concurrency? Redundancy? Generating keys becomes even more complicated. Same goes for designing an analytics system, a news feed, or a Q&A forum, each with its own set of common DS&A.

Don’t forget to account for your scaling requirements, where analyzing runtime and memory complexity really becomes handy.

## **Step 6 — Tradeoffs**

Almost every decision will involve a trade off. Being able to describe them in real time, as you’re suggesting solutions, shows that you understand that complex systems often require compromises and allow you to demonstrate your knowledge regarding the pros and cons of different approaches. Since there’s no one correct answer, having this discussion will give your interviewer the impression that you’re practical and will use the right tool for the job.

A few common examples might include:

* What type of database would you use and why?
* What caching solutions are out there? Which would you choose and why?
* What frameworks can we use as infrastructure in your ecosystem of choice?

# **Practice is Key**

While you can definitely hone down the theory by yourself, the last piece of the puzzle is practice. Make sure to apply these steps time and time again, answering questions about modern system design with real peers.

It’s the combination of knowledge, technique, and practice, that’ll enable you to land your dream job.

# **Additional Resources**

## **Articles/Blogs:**

* [System Design Cheat Sheet](https://gist.github.com/vasanthk/485d1c25737e8e72759f) — A guide on the key topics within system design
* [The Log: What every software engineer should know about real-time data’s unifying abstraction](https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying) — A lengthy article about logs and tradeoffs
* [High Scalability](http://highscalability.com/) — A blog about how real-world systems are designed
* [The Architecture of Open Source Applications (Volume 2): Scalable Web Architecture and Distributed Systems](http://www.aosabook.org/en/distsys.html) — An article about some of the key issues to consider when designing large websites
* [Distributed Systems Reading List](http://dancres.github.io/Pages/) — A list of articles on distributed systems to help change the way you think

## **Videos:**

* [Intro to Architecture and Systems Design Interviews](https://www.youtube.com/watch?v=ZgdS0EUmn70) — YouTube tutorial made by an ex-Facebook engineer about system design interviews
* [System Design Introduction For Interview.](https://www.youtube.com/watch?v=UzLMhqg3_Wc) — YouTube tutorial made by Tushar Roy about system design interviews